# Gestion Automatique d'Environnement Virtuel (GAEV)

Loic Houde, Daniel Jacob, Tovo Rabemanantsoa, Jean-François Rey

▶ **To cite this version:**

Loic Houde, Daniel Jacob, Tovo Rabemanantsoa, Jean-François Rey. Gestion Automatique d'Environnement Virtuel (GAEV). [0] INRAE. 2021. hal-03192628

**HAL Id: hal-03192628**
**https://hal.science/hal-03192628v1**

Submitted on 8 Apr 2021

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - ShareAlike 4.0 International License

RÉPUBLIQUE FRANÇAISE
*Liberté*
*Égalité*
*Fraternité*

INRAE

**GAEV (Gestion Automatique d'Environnement Virtuel)**

DipSO

# Rapport / Bilan - Année 2020

Le 07/04/2021

## Auteurs


Loic Houde | CATI IMOTEP
Daniel Jacob | CATI PROSODie
Tovo Rabemanantsoa | CATI PROSODie
Jean-François Rey | CATI IMOTEP (porteur principal)

# Introduction

De nos jours, le partage d'applications scientifiques est indispensable mais cela reste parfois très compliqué. Effectivement une application scientifique impliquant un ou plusieurs outils informatiques n'est pas forcément facile à stocker, à trouver, à installer et à utiliser. Ce projet a pour but de montrer qu'il est possible d'élaborer une infrastructure permettant de faciliter cela en automatisant le maximum de choses et en utilisant des environnements virtuels.

De ce fait, le projet **GAEV** (**G**estion **A**utomatique d'**E**nvironnement **V**irtuel) vise à établir une preuve de concept (POC) d'une infrastructure logicielle permettant de **créer**, **configurer, référencer** et **stocker** à la demande des environnements sous forme de machines virtuelles lourdes déployables à multiple échelle.

La finalité d'un tel projet est de **faciliter la mise à disposition** d'applications scientifiques préinstallées et configurées sur des machines virtuelles référencées. Cela s'inscrit dans une démarche qualité de science ouverte pour le **partage** et la **reproductibilité** tout en s'appuyant sur les principes FAIR.

# Présentation du projet

## Cadre

Ce projet est soutenu par la DipSO[1] dans la cadre d'appel à projet 2020. Il a été proposé par et réunit quatre ingénieurs INRAE[2] des CATIs IMOTEP[3] et PROSODie[4] :

- Loic Houde
- Daniel Jacob
- Tovo Rabemanantsoa
- Jean-François Rey (porteur du projet)

Ce projet se déroule sous la direction des co-animateurs du CATI IMOTEP Thierry Hoch et Hervé Richard et du directeur d'unité de BioSP[5] Samuel Soubeyrand.

## Contexte

Ce projet souhaite mettre en place un service de **G**estion **A**utomatique d'**E**nvironnement **V**irtuel à la demande dans le cadre de l'amélioration, de l'intégration et de la livraison continue des productions développées par nos chercheurs et collègues INRAE, ainsi que nos

[1] https://ist.inrae.fr/list-a-inrae/dipso/
[2] https://www.inrae.fr/
[3] https://imotep.inrae.fr/
[4] https://prosodie.cati.inrae.fr/
[5] https://informatique-mia.inrae.fr/biosp/

collaborateurs extérieurs et des différents CATIs. De tels environnements disponibles à différents niveaux (applicatif, développement, tests et production) permettraient une diffusion plus rapide, de qualité et une meilleure accessibilité des projets open sources développés par notre communauté scientifique. Cela intervient aussi dans un dispositif qui vise à faciliter la reproductibilité scientifique grâce à des **environnements contrôlés**, **identifiés** et **réutilisables**.

Plus concrètement, cette étude s'inscrit dans une démarche pour **faciliter la mise à disposition** et le **partage** d'applications nécessitant de fonctionner sur une **machine virtuelle (VM)**, c.à.d. que l'application ne fonctionne pas sous le système d'exploitation (OS) de l'hôte, mais fonctionne sous un système d'exploitation spécifique (en mode virtuel) nécessitant un accès à l'interface pour l'utilisateur.

La génération automatique de ces environnements (applications, outils de développement et de tests) doit faciliter l'intégration et la livraison continue des productions informatiques scientifiques nécessitant des **environnements divers**, **variés** et **reproductibles**. L'infrastructure permettrait également le **stockage**, la **description** et le **référencement** des environnements générés.

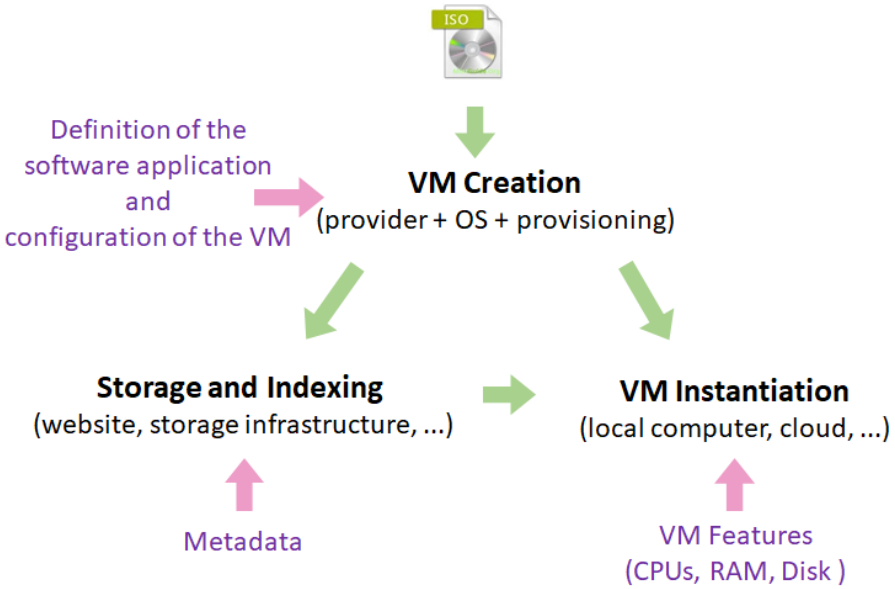
# Conceptualisation

Le projet dans sa globalité, voudrait mettre à disposition un site web pour créer, configurer, déposer et référencer des machines virtuelles (VM) sous différentes couches de virtualisation (providers) et pour différents systèmes d'exploitations (OS), lesdites machines virtuelles embarquant des applications scientifiques préinstallées et configurées.

Du point de vue des chercheurs cela faciliterait la création, la gestion et la reproductibilité d'une VM contenant leurs applications et du point de vue utilisateur cela permettrait la recherche, le téléchargement de la VM et l'utilisation des applications scientifiques.

Pour réaliser ce POC, trois couches ont été identifiées :

1. Création des VMs (providers + OS VMs + provisioning)
2. Stockage et indexation (site web, infrastructure de stockage, ...)
3. Instanciation des VMs (ordinateur, cloud, ...)




Dans les parties qui suivent nous allons aborder ces trois couches, tout particulièrement les couches ‘Création’ et ‘Instanciation’. La couche ‘Stockage’ sera abordée plus comme perspective pouvant servir de base pour une version 2 du projet.

# Phases de travail

Une première phase du POC a consisté à identifier des solutions/technologies existantes qui pouvaient nous être utiles et répondant aux critères énoncés ci-après.

Dans une deuxième phase, nous avons effectué des tests sous forme de cas d'utilisations (**use-cases**) pour valider nos choix. Les use-cases sélectionnés doivent servir à illustrer des situations typiques potentiellement rencontrées par les scientifiques et ingénieurs souhaitant faire fonctionner des applications impliquant un environnement logiciel complexe non disponible sur leur environnement habituel (par exemple : PC sous Windows et application scientifique sous Linux).

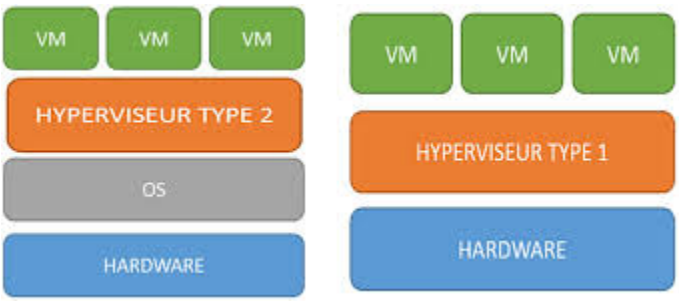
# Phase 1 : Outils et Technologies

## Veille technologique

Pour accomplir ce projet et avant toute réalisation, une veille technologique sur les outils existants a été établie afin d'identifier les outils les plus pertinents… Nos critères de recherches pour ces **outils** ont été de préférence : libres, **open sources** et possédants une forte **communauté**. Plus précisément, ces outils devaient nous permettre de **créer**, **modifier**, **configurer**, **instancier** et **stocker** des machines virtuelles (VM) et permettre leurs **indexations** et **reproductions** à l'identique.

Vous pouvez trouver en annexes les tableaux résumant les principales recherches effectuées. (outils, virtualisations, clouds)

Il ressort un concept important ; les "**Providers**" : Il s'agit des outils logiciels permettant d'instancier (faire tourner) une machine virtuelle. Ils permettent de simuler des ressources informatiques matérielles quelle que soit la machine physique sur laquelle ils fonctionnent. Cette émulation plus couramment appelée **Virtualisation**, peut intervenir au niveau matériel et l'on parlera alors de virtualisation de type 1 (ex : Hyper-V, VMware ESX, Proxmox), ou au niveau de l'OS de la machine hôte et dans ce cas on parlera de virtualisation de type 2 (ex : VirtualBox, VMware Workstation).



Le concept de provider est un peu plus élargi suivant son contexte et il peut inclure parfois des services de virtualisation de type cloud, qui offrent une couche d'abstraction de la partie virtualisation en fournissant (via leurs interfaces de programmation d'applications API) des instances de machines virtuelles déjà prêtes et configurées.

## Nos choix

Nos investigations se sont arrêtées sur trois principaux outils : **Packer**[6], **Vagrant**[7] et **Ansible**[8]. Tous les trois disposent d'une forte communauté d'utilisateurs et ils sont tous open source et libres d'utilisation. Il est à noter que Ansible est disponible sous licence GPL et les deux autres sous licence MIT. Ils exploitent les concepts de IaC (Infrastructure as Code) c'est-à-dire qu'ils utilisent un ou des fichiers (Templates) lisibles par l'humain et la machine afin d'automatiser la gestion et la configuration de machine.

Packer et Vagrant sont des outils développés par Hashicorp[9] et sont complémentaires.

Packer permet de créer des images de machines identiques pour de multiples providers à partir d'images au format ISO. Construire de telles images (appelée VM de base ou base box[10]) est très utile s'il n'existe pas déjà de base box répondant à vos besoins ou si vous voulez maîtriser le contenu de votre VM.

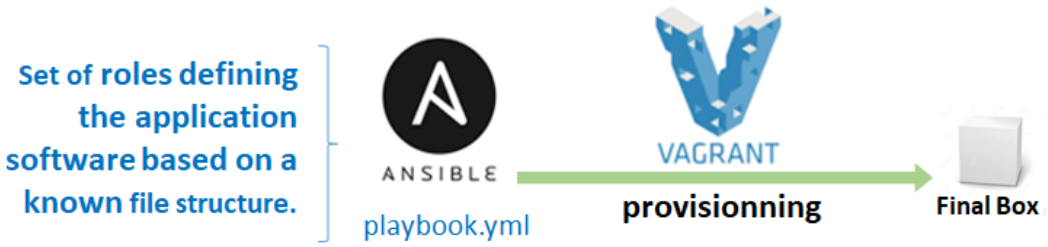

Vagrant permet de manipuler ces images pour les configurer, les provisionner (avec Ansible, cf ci-après) et les instancier. Ces images sont appelées "**boxes**" dans l'environnement Vagrant.


Avec ces deux outils, on dispose d'un très bon moyen de mettre au point et de diffuser des environnements de travail, et ce, de manière reproductible. Ils permettent donc d'éviter le syndrome "It works on my machine".

---

[6] https://www.packer.io/
[7] https://www.vagrantup.com/
[8] https://www.ansible.com/
[9] https://www.hashicorp.com/about
[10] https://www.vagrantup.com/docs/boxes/base.html

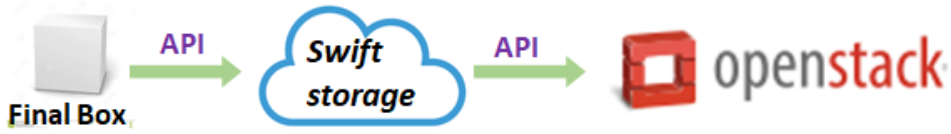
Ansible, développé par Red Hat[11], est un outil de gestion de la configuration qui automatise l'installation, le déploiement, l'exécution des tâches et la gestion de la configuration sur plusieurs machines en même temps. Plus précisément pour notre utilisation, il sert à alimenter la machine virtuelle (**provisioning)** avec des outils systèmes et/ou applicatifs selon les fichiers de définition (appelés **rôles**).




Ces trois outils nous permettent de créer, modifier, configurer et instancier **automatiquement** et de manière reproductible des machines virtuelles.

Pour ce qui est de l'indexation et du stockage nous n'avons pas trouvé de solution existante clé en main. Mais nous avons testé / exploré quelques solutions. La plus simple pour le stockage est d'utiliser Vagrant Cloud[12] qui permet de stocker les images (boxes) et le référencement (indexation) par type de provider. C'est d'ailleurs cette solution que nous utilisons pour stocker les images créées par Packer (appelées des "base boxes") et qui facilitent leurs utilisations dans Vagrant.


Une autre possibilité testée est le stockage via l'API S3 d'openstack[13] de l'institut[14] et de genouest[15] ou le téléchargement au sein d'une librairie d'images instanciables par openstack via l'API Swift/nova[16]. Possibilité qui nous permet d'héberger les machines virtuelles dans des infrastructures de recherche nationales.




[11] https://www.redhat.com
[12] https://app.vagrantup.com/GAEV
[13] https://www.openstack.org/
[14] https://cloud.inrae.fr
[15] https://www.genouest.org/
[16] https://docs.openstack.org/swift/queens/overview_backing_store.html

Une troisième solution non explorée encore serait d'envisager une collaboration avec l'Institut Français de Bioinformatique[17] (IFB) afin d'étendre les fonctionnalités de leur catalogue RANBio de Biophere[18] pour permettre le dépôt, le référencement et le téléchargement de VM, voire leur instanciation sur le cloud de l'IFB.

L'indexation et le référencement sont encore en cours d'investigation pour conceptualiser une des solutions.

Pour les besoins du POC et le fait que nous ayons un accès restreint à différents types de providers, nous avons choisi de réaliser ce POC en utilisant le provider VirtualBox[19] (type 2), qui nous permet de travailler sans restriction de format d'image sous des machines virtuelles avec des OS différents (UNIX-like, Windows et MacOS). De plus, VirtualBox est facilement utilisable autant sur un ordinateur standard que sur un serveur.

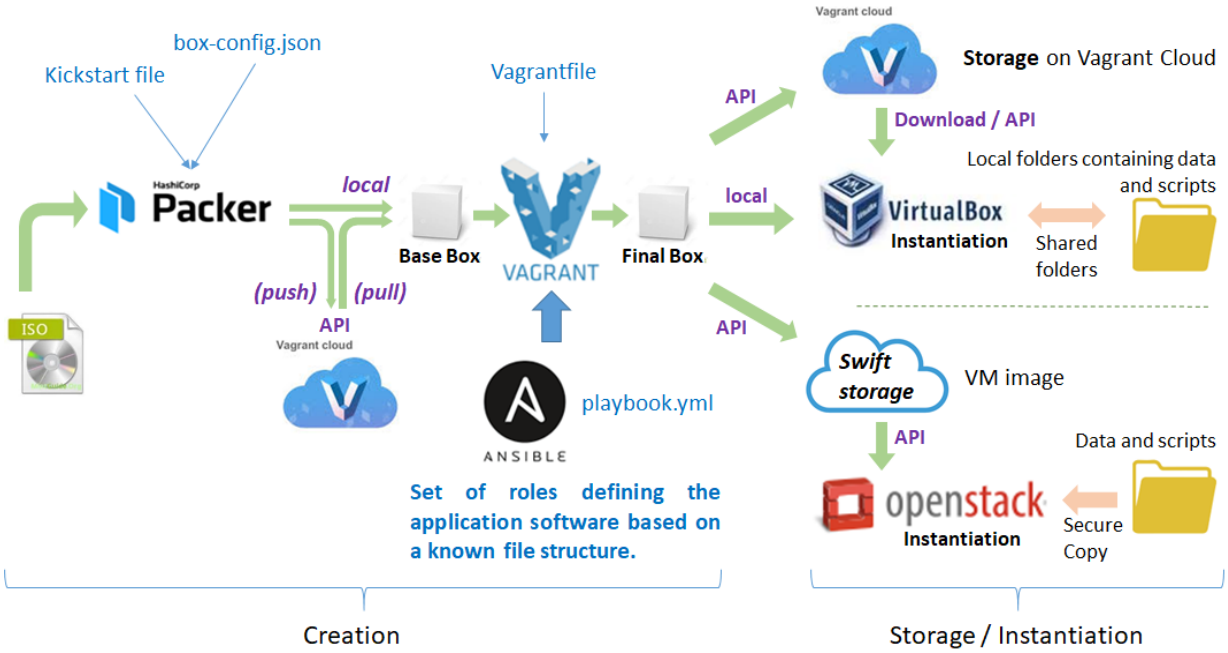
Le POC est facilement portable sur d'autres providers car ces derniers partagent les mêmes formats[20] et/ou ces derniers sont convertissables entre eux à l'aide d'outil comme qemu-img[21].

[17] https://www.france-bioinformatique.fr/
[18] https://biosphere.france-bioinformatique.fr/catalogue/
[19]https://www.virtualbox.org/
[20] Format standard example: Open Virtualization Archive (OVA), Virtual Machine Disk (VMDK), Virtual Hard Disk (VHD/VHDX), Open Virtualization Format (OVF)
[21] https://www.qemu.org/docs/master/interop/qemu-img.html

# Phase 2 : Cas d'usages (use-cases)

Dans cette partie, après une description d'un use-case générique pour obtenir une VM partageable, nous présentons différents use-cases réalisés avec ces outils.

## Use-case Générique




Construction d'une "VM finale" (Final box) pour partager une application ou une boîte à outils

Dans la plupart des cas, Vagrant est utilisé à partir d'une "VM de base" (**base box)** telle que celles que l'on peut trouver sur https://app.vagrantup.com/boxes/search. Mais Il peut être intéressant de constituer sa propre base box, par exemple :

- si vous voulez maîtriser le contenu de votre VM (c'est parfois une question de sécurité ou de taille limite des disques autorisée sur les services clouds) ;
- s'il n'existe pas déjà de base box répondant à vos besoins ;
- si le provisioning de votre box est long (compilation de paquets, téléchargements volumineux…) ;
- si vous voulez fixer une version exacte de votre environnement (pour ne pas dépendre d'un update, par exemple).

Les VM de base générées pour le projet se trouvent sur le compte GAEV de Vagrant Cloud : https://app.vagrantup.com/GAEV

Description des étapes à suivre

1. Choix d'une "VM de base" (Base box) préconfigurée pour le pipeline.
    a. Critères du choix => (OS + Provider)
        i. Couche de virtualisation (provider, ex : VirtualBox, VMware, KVM, HyperV, ...)
        ii. OS ciblé (Windows, Linux, Mac)
    b. Absence de VM de base (c.à.d conforme aux spécifications requises) :
        i. Construire la VM avec Packer (*local*) en partant d'un fichier ISO (OS) puis à l'aide d'un template (*box-config.json*) et d'un fichier de configuration de la VM (*kickstart file*) spécifique à la couche de virtualisation (provider) choisie.
        ii. Déposer éventuellement la VM ainsi construite sur Vagrant Cloud (*push*).
    c. Disponibilité de la VM de la base
        i. Téléchargement depuis Vagrant Cloud (*pull*)

2. Construction de la VM finale (Final box) à partir de la VM de base (Base box)
    a. Étape 1 : Construction du playbook d'Ansible
        i. Ensemble des procédures concernant la configuration et l'installation des différents composants systèmes et/ou logiciels, décrites en Yaml (*roles* + *vars*)
        ii. Les procédures s'appliquent soit sur des éléments logiciels en ligne (repositories), soit sur des éléments logiciels (*scripts, données, …*) fournis par le créateur de la VM.
    b. Étape 2 : Création du template de Vagrant (*Vagrantfile*)
        i. Configurer les caractéristiques de la VM
        ii. Appliquer l'ensemble des procédures décrites par Ansible (*roles* + *vars*)

3. Utilisation de la VM finale (Final box)
    a. Utilisation sur son instance locale de virtualisation (ex VirtualBox).
    b. Partage via un stockage (Vagrant Cloud ou stockage type S3)
    c. Instanciation de l'image de la VM chez un fournisseur (type openstack, par ex genouest.org) / accès via https

# Exemples de use-cases

## Use-cases

- **JupiterHub** (configuré avec différents packages/environnement de travail) peut être spécialisé. Accessible aux chercheurs, TP écoles chercheurs, autres formations, ...
- **NMRProcFlow** : Application fournie sous forme de conteneur (Docker)
- **Automatisation de la création et du partage d'un environnement sous R** : Intégration Continue / Distribution Continue pour les développeurs et utilisateurs R sous différents OS.

## Sources

Les codes sources sont disponibles sur la Forge MathNUM dans le groupe de travail GAEV :

- https://forgemia.inra.fr/gaev.

## Use-Case : JupyterHub

JupyterHub[22] apporte la puissance des Jupyter notebook à des groupes d'utilisateurs. Il permet aux utilisateurs d'accéder à des environnements et des ressources de calcul sans leur imposer de tâches d'installation et de maintenance. Les utilisateurs - y compris les étudiants, les chercheurs et les spécialistes des données - peuvent effectuer leurs travaux dans leurs propres espaces de travail sur des ressources partagées qui peuvent être gérées efficacement par les administrateurs du système. En effet, à partir d'un compte administrateur depuis l'interface web, on peut administrer la gestion des utilisateurs. JupyterHub fonctionne dans le cloud ou sur votre propre PC et permet de servir un environnement scientifique de données préconfiguré à n'importe quel utilisateur dans le monde. Il est personnalisable et évolutif, et convient aux petites et grandes équipes, aux cours universitaires et aux infrastructures à grande échelle.

### Construction / Configuration d'une machine virtuelle

- On doit pouvoir choisir ses modules R et Python (par catégories ; ex biostatistics, data processing, ...)
- En utilisation locale, on doit pouvoir avoir accès à un répertoire partagé sur son disque
- En utilisation cloud (par ex : openstack), on doit pouvoir avoir accès via SCP (WinSCP[23], Cyberduck[24]) au répertoire de partage commun à tous les utilisateurs afin d'y télécharger des fichiers de données, des scripts, etc …

En Annexe B1 sont présentées les différentes étapes.

## Use-Case : automatisation de la création d'un environnement sous R

Ce use-case vise à automatiser la création et le partage d'une VM avec l'environnement R pré-installé et configuré en utilisant l'intégration continue sous GitLab. Cela afin de disposer de VMs sous différents OS avec un environnement R clé en main et assurer une qualité des livrables (les VMs).

Le logiciel de programmation R et son environnement est un outil très utilisé dans la communauté scientifique, que ce soit pour effectuer des analyses sur des données en statistique et mathématique ou bien le partage de package R contenant tous types de développement (modèles, outils, données, …). La difficulté d'homogénéité du logiciel suivant le système d'exploitation utilisé, rend parfois difficile le partage de modèle et des packages.

[22] https://jupyter.org/hub
[23] https://winscp.net/eng/index.php
[24] https://cyberduck.io/

L'automatisation vise à faciliter le partage d'une machine virtuelle contenant l'environnement R préinstallé et configuré pour permettre :

- aux scientifiques d'installer leurs scripts R et packages facilement,
- de reproduire à l'identique un environnement (clé en main),
- de partager cette VM avec la communauté,
- de mettre à disposition aux développeurs un environnement propre,
- d'utiliser cette VM comme GitLab-Runner pour l'intégration continue,
- permettre l'utilisation de la VM sur son PC ou sur un serveur.

Deux use-cases (l'un sous Windows et l'autre sous MacOS) sont disponibles pour le projet aux adresses suivantes :

- https://forgemia.inra.fr/gaev/usecases/windowsr
- https://forgemia.inra.fr/gaev/usecases/macosr

En Annexe B2 sont présentées les différentes étapes.

# Retours d'expériences

## Création et Provision

La création automatique de VM a été mise en œuvre par un grand nombre d'outils éprouvés et de ressources disponibles en ligne sur lesquels nous avons pu nous appuyer. L'outil Ansible facilite grandement le provisioning pour les OS Linux-like. Néanmoins, certains paramètres ou fonctionnalités restent très spécifiques à Windows ou à MacOS, ce qui ne permet pas une automatisation générique.

## Performance

Les machines virtuelles sous virtualbox ont un format standard qui permet de pouvoir les instancier sous différents providers. Faire tourner une VM sur une VM d'un hyperviseur (ex : proxmox -> VM virtualbox -> ma VM) même avec le mode *nested virtualization* activé, offre de faible performance. Effectivement trop de couches de virtualisation engendrent de faibles performances. Permettre la création depuis une machine physique est la meilleure solution et pour l'instanciation sur une machine local ou via un hyperviseur (proxmox ou vmware) fonctionne très bien.

## Stockage

Les boxes (VM) créés peuvent être très volumineuses, de 1Go à 100Go suivant leur OS et configuration. Pour utiliser une box, il faut la télécharger, ce qui peut être parfois long. Il faut ensuite l'instancier, ce qui augmente l'espace nécessaire à chaque nouvelle instanciation. En local, on peut très vite se retrouver avec le disque dur plein. Les solutions de stockage Vagrant Cloud ou S3 sont adaptées pour du stockage sur le long terme, mais là aussi il faut qu'il y ait beaucoup d'espace disponible.

## Environnement

Il est impératif de bien connaître l'environnement d'instanciation de la VM lors de sa phase de configuration et de provisioning. En effet, si l'on souhaite faire tourner la VM en local sur un PC, il est judicieux que la machine puisse avoir accès aux disques locaux en mode partagé. Il faut prévoir l'installation des pilotes appropriés au niveau de la couche de virtualisation. Dans le cas d'une utilisation sur un cloud, un outil de configuration lors du démarrage de la VM comme cloud-init[25] devra être installé sur la machine lors du provisioning. De même, la configuration du réseau (choix d'une adresse IP fixe ou en mode DHCP) devra tenir compte de l'environnement. Il est bon aussi de savoir qu'une machine instanciée sur un

[25] https://cloud-init.io/

cloud ne connaîtra ni son numéro IP ni son nom complet. Aussi, il faudra prévoir des scripts lors du son démarrage pour configurer les applications qui nécessitent de connaître leur URL externe (Base URL). Enfin la gestion des clés de sécurité (insecure private keys), des comptes utilisateurs et de leurs privilèges doivent faire parties des préoccupations lors de l'étape de provisioning.

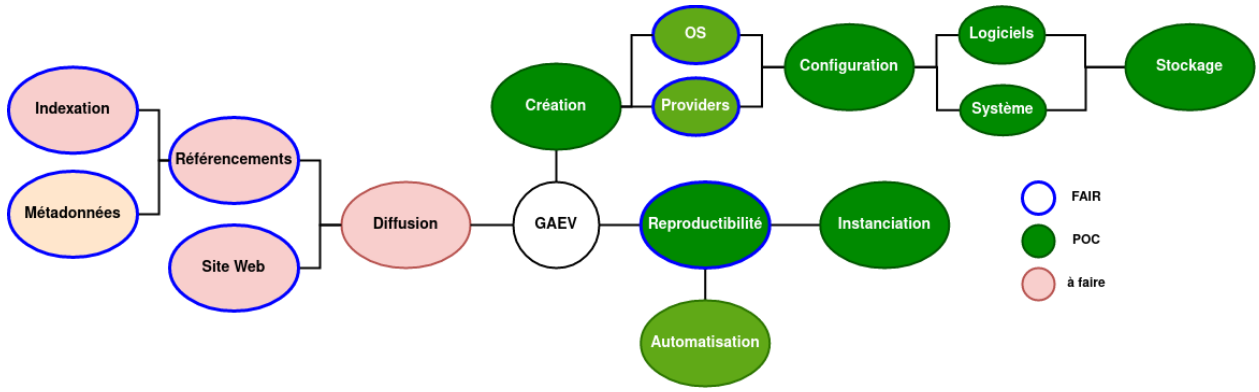
# Conclusion

Le projet a bien avancé malgré la crise sanitaire (ce qui nous a empêchés de nous réunir physiquement) et le calendrier a pu quand même être respecté pour les livrables. L'ampleur du PoC, le nombre de participants et les contraintes d'emploi du temps de chacun, nous ont permis de réaliser les deux tiers.

L'automatisation de la création, de la configuration, du stockage et de l'instanciation de VMs sous Virtualbox nous a permis de démontrer la faisabilité, surtout avec les moyens dont nous disposons (un remerciement à la DSI INRAE pour la mise à disposition d'espace de stockage et GenOuest pour l'accès à leur openstack).

La partie référencement (indexation et métadonnées) reste à étudier et à réaliser pour terminer ce PoC ; partie qui sera discutée dans les perspectives pour une prolongation du projet.

Pour conclure, nous avons démontré qu'il est possible de gérer des environnements virtuels automatiquement pour permettre leurs partages à la communauté avec des outils scientifiques préinstallés et configurés, et ce, de manière reproductible.




Schema objectif du projet

# Perspectives

Les suites qui pourraient être apportées au projet sont de deux ordres :

- Continuer le projet pour le réaliser dans sa globalité.
- Participer à des Retours d'EXpériences (REX) et/ou formations/ateliers avec nos confrères.

Pour continuer ce projet voici deux grandes étapes à réaliser :

- Priorité 1 : L'étude et la réalisation de la partie référencements.
    - Cette partie pourrait consister au développement d'un site-web pour permettre le stockage avec référencement unique (PID) et l'indexation via des métadonnées des VMs.
    - Cette partie peut aussi être effectuée via des collaborations avec des plateformes tel que l'IFB (Biosphère/RAMbio)
- Priorité 2 : Faire évoluer le site web et les pipelines sous-jacents pour permettre la gestion automatique des environnements virtuels, c.à.d. le POC actuel intégré dans le site web pour permettre l'automatisation de la création, de la configuration, du stockage, de l'indexation et du partage en click bouton. Cela pourrait ressembler au site web https://phansible.com/

Il est à noter qu'une collaboration avec le projet CSAN[26] (Comprehensive software archive network) pourrait être envisagée sur des parties communes comme les métadonnées, la mise en place du site web (dépôts référencé).

Dans tous les cas, pour rendre ce projet pérenne, un soutien tant financier que moral sera nécessaire afin de maintenir la motivation et l'implication des participants à la hauteur des ambitions souhaitées par l'institut.

[26] https://groupes.renater.fr/sympa/info/csan

# Remerciements

Nous tenons à remercier tout particulièrement la DipSO pour son engagement à soutenir ce projet (moralement et financièrement), qui sans celui-ci n'aurait jamais vu le jour.

Ainsi que les animateurs des CATI IMOTEP et PROSODie et du DU de l'unité BioSP pour leurs retours lors de la demande de financement et d'avoir accepté de soutenir ce projet.

Pour finir nous souhaitons remercier la DSI INRAE et GenOuest pour leur mise à dispositions des leurs infrastructures ce qui nous a permis de réaliser ce POC.





# ANNEXES

# Annexe A : Tableaux synthétisant les recherches sur la veille technologique

Tableau représentant les principaux outils de virtualisation étudiés (Providers). Il se peut qu'il manque des informations, celles présentes sont à titre explicatives.

| Virtualisation / Providers | mainteneur | Type(1) | Format(2) | License | OS (Host) | OS (Guest) |
|---|---|---|---|---|---|---|
| VirtualBox | Oracle | 2 | OVA / OVF<br>VMDK, VDI | GPL 2 - libre<br>full version propriétaire | Windows<br>MacOS<br>Linux | Linux<br>DOS<br>Windows<br>FreeBSD<br>Mac OS<br>Solaris<br>OpenBSD |
| VMware | VMware | 1 et 2 | OVA / OVF<br>VMDK | propiètaire | Windows<br>MacOS<br>Linux | Windows<br>Linux<br>Solaris<br>FreeBSD<br>OS/2 |
| KVM | Red Hat | 1 | | GPL | Linux<br>FreeBSD<br>illumos | FreeBSD<br>Linux<br>Windows<br>Solaris |
| docker | | isolé | | | | |
| lxc | Canonical Ltd | isolé | | GPL2 | Linux | Linux |
| parallels | Parallels, inc | 2 | OVA / OVF<br>VMDK, VDI | propriétaire | MacOS | DOS<br>Windows<br>Linux<br>macOS<br>FreeBSD<br>OS/2 |
| qemu | Fabrice Bellard et communauté | 2 | QCOW2 | GPL/LGPL | Windows<br>Linux<br>MacOS<br>Solaris<br>FreeBSD<br>OpenBSD | Beaucoup |
| Xen | XenSource | 1 | | GPL | Linux | Linux<br>Windows |

| | | | | | | |
|---|---|---|---|---|---|---|
| Proxmox VE | proxmox | 1 | qcow2<br>raw<br>vmdk | AGPLv3 | Debian | LXC<br>Windows<br>Linux<br>Solaris<br>FreeBSD |
| Hyper-V | Microsoft | 1 et 2 | VHD / VHDX | propriétaire | Windows | Windows |
| Jails | Communauté BSD | 1 | | BSD | FreeBSD | FreeBSD |

(2) Open Virtualization Archive (OVA), Virtual Machine Disk (VMDK), Virtual Hard Disk (VHD/VHDX), Open Virtualization Format (OVF)
(1) Type hyperviseur 1, 2 et isolé

Tableau des solutions cloud. Les prix sont à titre indicatif.

| **Plateforme** | **URL** | **Providers** | **OS** | **Stockage** | **Coûts (à titre indicatif)** | **Comment-aires** |
|---|---|---|---|---|---|---|
| AWS | https://aws.amazon.com/cloudformation/ | Containers<br>VMWare<br>Citrix<br>Hyper-V<br>Azure | Linux<br>Windows | S3 bucket<br>Block<br>Fichiers | stockage: 0,024 USD par Go/mois<br>Machine : 5 à 250$ / mois | |
| Azure | https://azure.microsoft.com/en-us/features/resource-manager/ | Hyper-V | Linux (RedHat, SUSE)<br>Windows (natif) | Blob<br>Block<br>Fichiers | Stockage : 30e/mois/TB<br>Machine : environ 200e/mois | |
| Google Cloud | https://cloud.google.com/ | Containers<br>VMs | Linux<br>Windows | Bucket<br>Blob<br>Fichier | machine standard 25 à 4000 e/mois<br>$0.080/GB/mois | |
| Openshift | https://www.openshift.com/ | Containers<br>VMs (CVM) | Linux<br>Windows | PV | 1500$ à 4000$ / ans | |
| openstack | https://www.openstack.org/ | Containers<br>VMs<br>Bare Metal | Linux<br>Nixos<br>Windows<br>plugins<br>extensions | Plugins principaux stockages | DSI Stockage 50e/TB/ans | opensource auto heberge |
| OVH Cloud | https://www.ovh.com/world/ | Containers<br>VMs<br>Bare Metal | Linux<br>Windows | Bucket<br>Blob<br>Fichier | stockage : $0.0112 /mois/GB<br>Cloud : 1500 $ à 12000 $ / mois | |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | Barmetal : 90 à 500 $/mois | |
| Vagrant Cloud | https://app.vagrantup.com/ | Infini | Tous | S3 | mode public : gratuit<br>mode privé : 5 $/mois<br>mode privé par équipe : 25 $/mois | uniquement du stockage |

Tableau des principaux outils étudiés.

| **Outils Création / Manipulation / Provisioning** | **Créateur / communauté** | **Interopérabilité** | **Interface ? CLI ?** | **License** | **Pérennité / Ages / %utilisateurs / Standard** |
|---|---|---|---|---|---|
| Ansible | Red Hat | Provisioning CaC | CLI + templates | GPL et Propriétaire Red Hat | Grosse communauté Standard |
| Vagrant | HashiCorp | Manipulation Différents Providers | CLI + templates | MIT, EULA | Très utilisé |
| Packer | HashiCorp | Création VM différents providers | CLI + templates | MIT, EULA | De + en + utilisé |
| Pulumi | Pulumi | IaC différents principaux providers cloud | CLI + templates | Opensource + propriétaire | Gros potentiel mais limité au cloud et infra |
| Chef | Chef, Progress | IaC différents principaux providers cloud | Ruby, Erlang | Apache 2.0 | Très répandu dans l'industrie |
| Puppet | Puppet | IaC différents principaux providers cloud | Ruby, templates | opensource + propriétaire | |
| Terraform | HashiCorp | IaC différents principaux providers | Cloud interface CLI + templates | Mozilla public 2.0 | Très utilisé dans la communautée |
| salt | SaltStack | IaC + outils avancés | Web | Propriétaire | |
| otter | Inedo | IaC | Web | Propriétaire + EULA | Communauté limité |
| rudder | Normation | Audi et Conf | Web + API | GPL 3 | communauté française |

| | | | | | |
|---|---|---|---|---|---|
| | | parc info | | | |
| (R)?EX | "Communauté" | IaC complet (Linux) | CLI + templates | Apache 2.0 | Petit communauté |
| CFEngine | northern | parc info | Web | GPL3 | |
| JUJU | Canonical | IaC differents providers | CLI/GUI | GPL3 | |
| baker | "Communauté" | meta IaC | CLI | Apache 2.0 | semble abandonné |
| veewee | "Communauté" | IaC limité, MacOS | CLI | MIT | pour OS exotique semble peu vivant |
| rundeck | rundeck | automatisation | Web | Apache 2.0 | automatiser des tâches et collaboratif |

Outils d'automatisation IaC (Infrastructure as Code), CCA (Continuous Configuration Automation) CaC (Configuration as Code)

# Annexe B : Cas d'utilisation

## B1: Use-Case : JupyterHub

Nous allons prendre comme exemple le cas d'utilisation (use-case) **JupyterHub**. Une distribution récente et évolutive conçue pour les petits déploiements, Littlest Jupiter Hub[27] (également connu sous le nom de TLJH) est une méthode légère pour installer JupyterHub sur une seule machine virtuelle. TLJH est une application développée sous python nécessitant de nombreuses dépendances. Même si cette distribution gère ces dépendances et prend en charge les configurations de chacun des éléments, son installation reste néanmoins complexe et chronophage.

Cet exemple a été mis en œuvre sur un ordinateur de bureau sous Windows10 / Cygwin. Mais toute machine (portable, ordinateur de bureau, serveur, …) sous MacOS ou UNIX-like peut parfaitement convenir. Sachez que la mise en œuvre de cet exemple requiert des compétences techniques dans l'installation d'applications et suppose que l'édition de fichiers de configuration ne vous rebute pas. On trouvera l'ensemble des scripts et autres fichiers de configuration mentionnés dans cette description en accès libre sous le dépôt GiHub INRAE : https://github.com/inrae/jupyterhub-vm

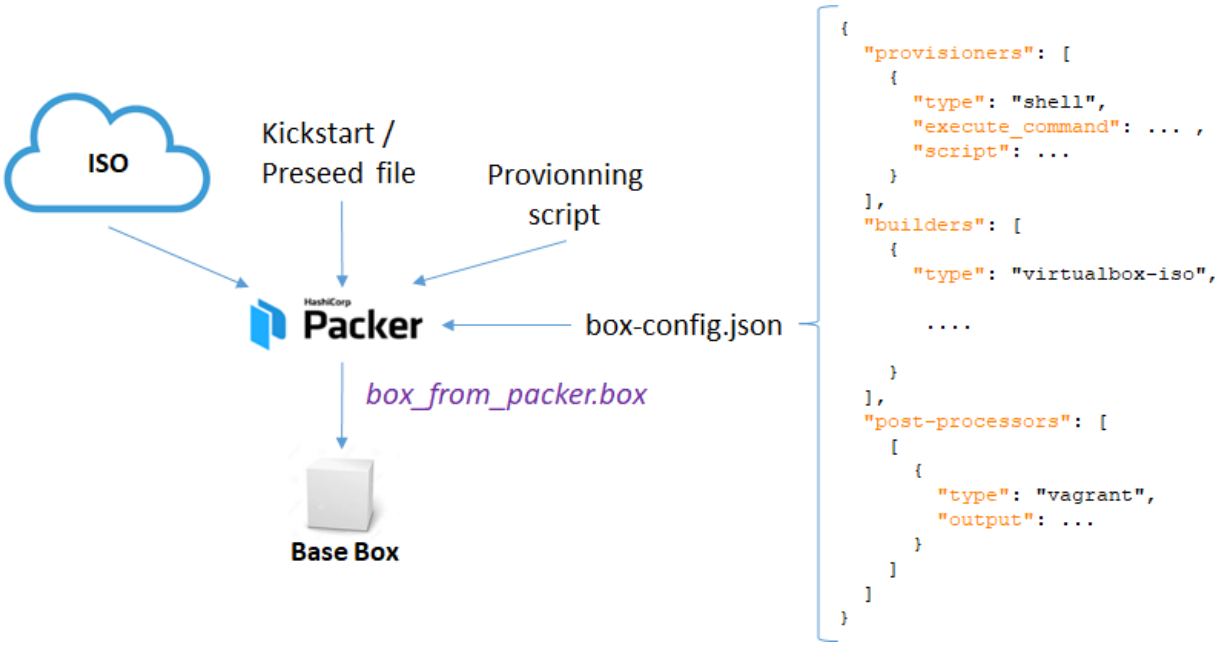
Pour construire notre VM finale, c.à.d. la VM qui sera instanciée pour une utilisation en local (virtualisation de type 2) ou sur un serveur distant (Datacenter ou Cloud), la procédure mise en œuvre s'appuie sur l'Use-case générique décrit précédemment et dont elle reprend les étapes. Ainsi nous allons procéder en deux grandes étapes pour la création. La première étape consistera dans la construction d'une VM de base (Base box). La deuxième étape consistera, à partir de la VM de base, à installer et à configurer l'ensemble des outils systèmes et applicatifs (packages) nécessaires à faire fonctionner correctement JupyterHub depuis un navigateur web. Une troisième étape décrira comment instancier la VM finale sur le cloud de GenOuest[28].

### Construction de la VM de base (Base box) pour Vagrant

On suppose que vous avez installé les outils Packer[29] et Vagrant[30], ainsi que le logiciel de virtualisation Oracle VirtualBox[31].

[27] https://tljh.jupyter.org/en/latest/
[28] https://www.genouest.org/2017/03/02/cluster/
[29] https://www.packer.io/downloads
[30] https://www.vagrantup.com/downloads
[31] https://www.virtualbox.org/wiki/Downloads

Les opérations permettant la création d'une base box sont documentées en ligne ; il faut suivre le guide de base[32] et appliquer les règles spécifiques au provider, à savoir pour VirtualBox[33]. La procédure est quelque peu fastidieuse et surtout, une erreur est vite arrivée. C'est pourquoi nous allons utiliser l'outil Packer, qui permet d'automatiser entièrement la création de base box Vagrant en suivant le processus illustré par la figure ci-dessous:




Nous nous basons sur la version 18.04 d'Ubuntu[34].

Une configuration Packer est définie avec un fichier JSON (*box-config.json*) comportant plusieurs sections:

- "*builders*": permet de définir les éléments pour la création de la machine virtuelle à partir d'une image ISO.
- "*provisioners*": sert à définir la configuration logicielle à partir de scripts Shell afin d'approvisionner la machine virtuelle.
- "*post-processors*": s'exécute une fois la machine virtuelle créée et approvisionnée. Cela permet en autre de définir le format de sortie de la VM.

### Section "builders":

Pour constituer cette section, il est souvent plus simple de partir d'exemples déjà fonctionnels et de modifier les quelques éléments spécifiques à sa configuration. Une requête dans un moteur

[32] http://docs.vagrantup.com/v2/boxes/base.html
[33] http://docs.vagrantup.com/v2/virtualbox/boxes.html
[34] http://cdimage.ubuntu.com/ubuntu/releases/18.04/release/

de recherche (Google par ex) avec les mots clés “github packer ubuntu 18.04 virtualbox” permet d’avoir suffisamment d’exemples. Ainsi nous sommes parti d’un exemple approprié à nos besoins[35], que nous avons ensuite adapté.

La box est constituée à partir d’une ISO Ubuntu spécifiée par son URL (`iso_urls`) et son checksum de type sha256 (`iso_checksum`) afin de créer la VM dans VirtualBox (“`type`”: “`virtualbox-iso`”) .

Un fichier de “preseed” (*http/preseed.cfg*) permet de configurer l’installation (cf plus d’infos sur le preseeding[36]). Nous avons adapté l’exemple en ajoutant des instructions concernant le compte root ainsi que la configuration du réseau.

Nous avons limité la taille maximale du disque à 18Go (`"disk_size": 18432`) afin que la VM finale puisse être acceptée par le cloud de GenOuest.

### Section “provisioners”:

Cette section permet d’exécuter des scripts Shell après que la machine virtuelle ait démarré correctement puis son système d’exploitation installé. C’est donc dans cette étape que nous pouvons configurer la VM afin qu’elle soit compatible avec Vagrant et VirtualBox. Un script Shell (*`scripts/setup.sh`*) est donc exécuté afin de: i) installer les pilotes pour VirtualBox, ii) de configurer les accès SSH en conformité avec les Boxes Vagrant.

### Section “post-processors”:

Une fois la machine virtuelle de base complètement installée et configurée, il suffit de l’exporter au format Vagrant.

## Création de la VM de base

Une fois la configuration établie, il suffit d’exécuter Packer de la façon suivante :

```
$> packer build box-config.json
```

L’ensemble des messages produits peuvent être consultés sur le dépôt github[37]

La VM de base après exécution se trouve sous le répertoire *`builds`*.

[35] https://github.com/geerlingguy/packer-boxes/tree/master/ubuntu1804
[36] https://help.ubuntu.com/18.04/installation-guide/amd64/apb.html
[37] https://github.com/inrae/jupyterhub-vm/blob/master/logs/packer.log

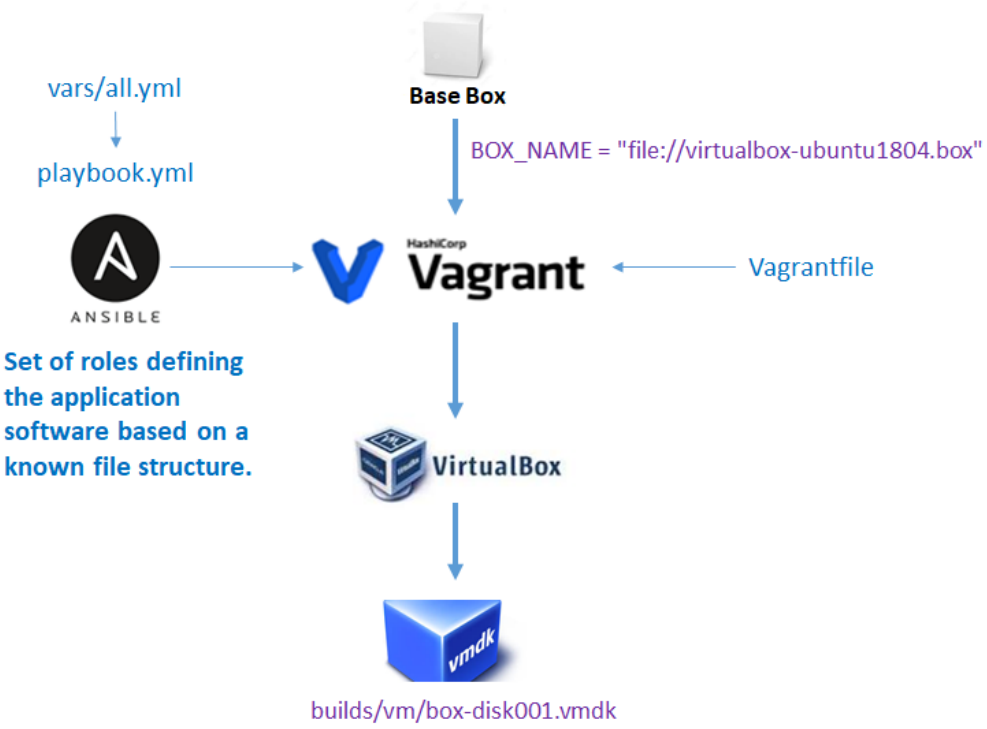
## Partir d'une VM de base (Base box) existante

Comme on le voit, la création d'une VM de base peut être complexe pour un non-initié. C'est pourquoi il est plus simple de partir d'une VM de base lorsque celle-ci est disponible. C'est aussi pourquoi la VM de base doit être la plus générique possible afin de pouvoir être utilisée facilement pour de nombreuses applications.

La VM de base générée et utilisée dans cet use-case a été déposé sur Vagrant Cloud[38], à l'aide de son interface de programmation (API, cf "Annexe B3").

## Construction de la VM finale

On va se servir de la VM de base afin de l'approvisionner (appelé le provisionning), c.à.d. installer et configurer l'ensemble des outils systèmes et applicatifs (packages) nécessaires à faire fonctionner correctement l'application JupyterHub. Pour cela, on va utiliser l'outil Ansible comme illustré dans la figure ci-dessous:




[38] https://app.vagrantup.com/GAEV/boxes/ubuntu1804-dd18Gb

### Création du fichier Vagrantfile

La première étape consiste à créer un fichier *Vagrantfile*[39] et le personnaliser en fonction de vos besoins. Ce fichier suit la syntaxe du langage Ruby[40]. Une version allégée en est donnée ci-dessous:

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

  config.vm.box = file://builds/virtualbox-ubuntu1804.box
  config.vm.hostname = jupyterhub

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

  config.ssh.insert_key = false

  config.vm.synced_folder ".", "/vagrant", type: "virtualbox"

  config.vm.provision "ansible_local" do |ansible|
      ansible.playbook = "ansible/playbook.yml"
      ansible.install = true
      ansible.limit = 'all'
  end

  config.vm.provision "shell", path: "scripts/cleanup.sh"

end
```

- *config.vm.box* : définit la base box en entrée
- *config.vm.hostname* : définit le nom de la machine (hostname)
- *config.vm.network* : définit la configuration réseau en mode dhcp
- *config.ssh.insert_key* : définit la clé SSH. Par défaut, ce sera celle définie dans la VM de base
- *config.vm.synced_folder* : définit une répertoire local comme répertoire partagé avec la VM. A minima, le répertoire "." doit être définie afin de configurer la VM. Ces répertoires de partage ne seront actifs que pendant l'étape de création puis de test avant l'exportation.
- *config.vm.provision* : définit le procédé par lequel la VM sera approvisionnée. Ici, deux procédés sont invoqués. i) 'ansible_local' : indique que l'outil ansible sera

[39] https://www.vagrantup.com/docs/vagrantfile
[40] https://www.ruby-lang.org/fr/about/

d'abord installé sur la VM puis utilisera le playbook définit dans `ansible/playbook`; ii) "`shell`": indique qu'un script shell définit par son chemin relatif (*`path`*) sera utilisé.

### Approvisionnement à l'aide de l'outil Ansible

Le choix a été fait de ne pas installer Ansible sur la machine hôte utilisée pour la création de la VM, mais sera plutôt installée sur la VM elle-même, ceci afin de simplifier le procédé.

L'approvisionnement est défini à partir d'un fichier au format YAML[41] appelé *`playbook.yml`*[42]

```
  ---
- hosts: all
  become: true
  become_user: root
  #gather_facts: no

  vars_files:
    - vars/all.yml

  environment:
    PYTHONHTTPSVERIFY: 0

  roles:
    - repositories
    - server
    - vm
    - install-r
    - jupyterhub
    - r_pkgs
    - python_pkgs
```

Ansible permet d'organiser les tâches dans une structure de répertoire appelée *Rôle*[43]. Dans cette configuration, les playbooks invoquent des rôles, eux-mêmes définissant un ensemble de tâches, de sorte que vous pouvez toujours regrouper des tâches et réutiliser les rôles dans d'autres playbooks. Les rôles vous permettent également de collecter des modèles, des fichiers statiques et des variables en même temps que vos tâches dans un format structuré.

Chaque rôle, puis chaque tâche au sein des rôles sont interprétés séquentiellement. Ainsi sont définis les rôles suivants:

- *`repositories`* : configure les dépôts (repositories) pour les packages binaires (systèmes, R, python, outils, …),

[41] https://en.wikipedia.org/wiki/YAML
[42] https://docs.ansible.com/ansible/2.4/playbooks_intro.html
[43] https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html

- *server* : configure le fuseau horaire (timezone)
- *vm* : configure le nom de la machine (hostname)
- *install-r* : installe l'application R avec des packages de bases
- *jupyterhub* : installe puis configure l'application jupyterhub
- *r_pkgs* : install un ensemble de packages R
- *python_pkgs* : install un ensemble de packages Python

Chaque rôle est défini par un ensemble de tâches (tasks), elles-mêmes définies par des fichiers au format YAML. Par exemple, le rôle '*repositories*' est défini par le fichier *roles/repositories/tasks/main.yml* suivant :

```
---

- name: Add an apt key by id from a keyserver
  apt_key:
    keyserver: "{{repository.keyserver}}"
    id: "{{repository.id}}"

- name: Add repositories
  apt_repository:
    repo: "{{item}}"
  with_items: "{{repository.repos}}"
```

Deux tâches sont ainsi définies et décrites par leur champ *name*. Ensuite la tâche proprement dite est définie à l'aide d'un nom se référant à un module[44]. Des centaines de modules[45] sont disponibles dans Ansible. Ici deux modules sont invoqués: i) apt_key[46]: ajouter ou supprimer une clé apt et ii) apt_repository[47]: ajouter ou supprimer un dépôt de packages binaires. Les variables définies par les doubles accolades {{ }} correspondent à celles définies dans le fichier *vars/all.yml* lui-même déclaré dans le playbook. Ainsi on trouve dans ce fichier les lignes suivantes :

```
repository:
    keyserver: hkp://keyserver.ubuntu.com:80
    id: E298A3A825C0D65DFD57CBB651716619E084DAB9
    repos:
      - ppa:marutter/c2d4u3.5
      - ppa:hnakamur/libgit2
      - deb https://cloud.r-project.org/bin/linux/ubuntu bionic-cran35/
```

[44] https://docs.ansible.com/ansible/latest/user_guide/modules_intro.html
[45] https://docs.ansible.com/ansible/2.8/modules/modules_by_category.html
[46] https://docs.ansible.com/ansible/2.8/modules/apt_key_module.html
[47] https://docs.ansible.com/ansible/2.8/modules/apt_repository_module.html

Ainsi c'est de cette manière qu'est défini l'ensemble des tâches d'approvisionnement par Ansible.

Le rôle '*jupyterhub*' a été construit à partir des instructions fournies sur le site de TLJH[48]. La liste des packages Python et R nécessaires à une utilisation spécifique sous JupyterHub sont à définir dans le fichier *vars/all.yml* dont l'installation est prise en charge par les rôles respectifs '`python_pkgs`' et '`r_pkgs`'.

### Approvisionnement par scripts Shell

Une fois terminée l'approvisionnement par Ansible, on procède à un nettoyage de la VM, à savoir désinstaller Ansible, nettoyer les fichiers temporaires et nettoyer le disque dur. On utilise un script Shell (*scripts/cleanup.sh*) pour cela.

### Création de la VM finale

Une fois la configuration établie, il suffit d'exécuter les commandes suivantes :

```
$> vagrant up
$> vagrant package --output ubuntu-box.tar.gz
```

L'ensemble des messages produits peuvent être consultés sur le dépôt github[49]

## Utilisation de la VM finale sur le cloud de GenOuest

GenOuest[50] est une plateforme nationale de Bioinformatique et fédérée par l'Institut Français de Bioinformatique (IFB[51]). Cette plateforme propose des services cloud pour la communauté française de la recherche publique. Ainsi tout chercheur de cette communauté peut faire la demande auprès de GenOuest ou de toute autre plateforme de l'IFB pour avoir accès aux services proposés.

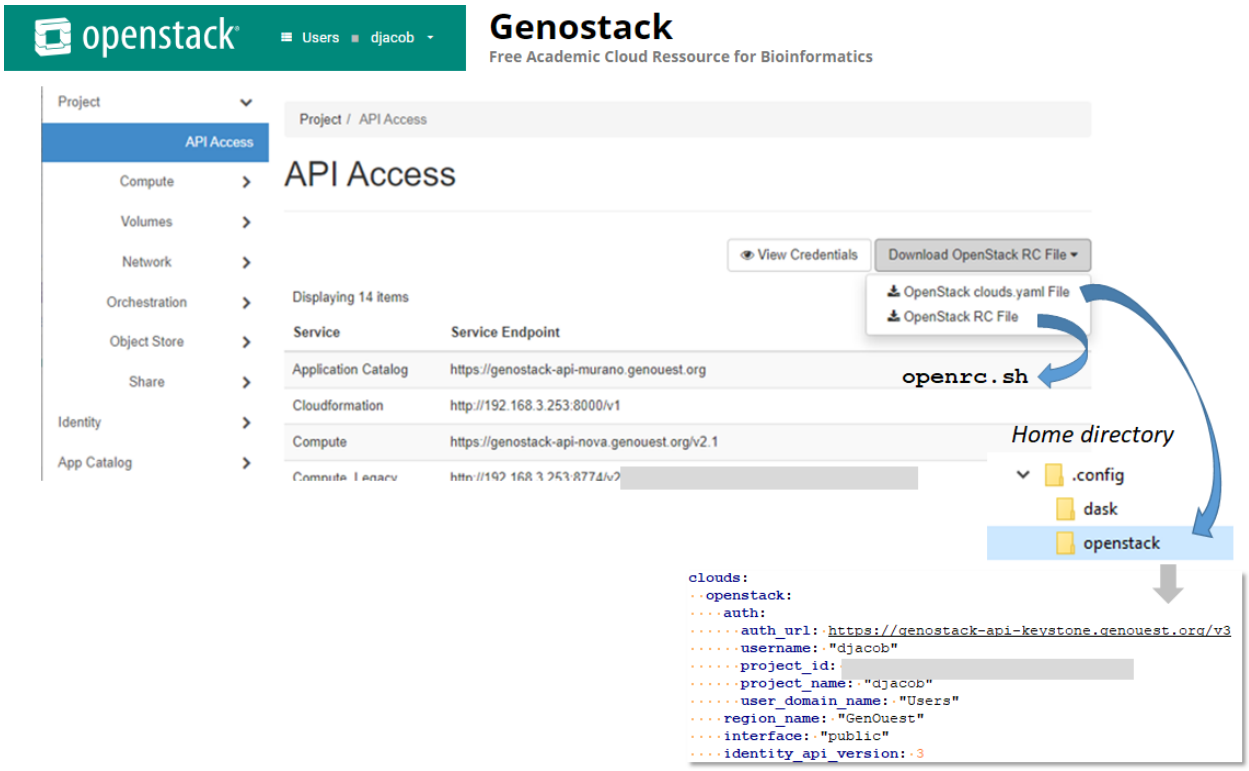
GenOuest propose sur son infrastructure datacenter un service de mise à disposition de ressources de calculs sous forme de machines virtuelles. L'infrastructure est gérée à l'aide de Openstack[52], qui est un ensemble de logiciels open source permettant de déployer des infrastructures de cloud computing (Infrastructure en tant que Service, `IaS`). Il est donc nécessaire de disposer d'un compte valide sur cette infrastructure (cf l'aide en ligne[53]).

[48] https://tljh.jupyter.org/en/latest/
[49] https://github.com/inrae/jupyterhub-vm/blob/master/logs/vagrant.log
[50] https://www.genouest.org/
[51] https://www.france-bioinformatique.fr/
[52] https://www.openstack.org/
[53] https://help.genouest.org/

On suppose ensuite que vous avez installer Python (≥2.7) ainsi que le package python-openstackclient[54].

### Récupération de ses paramètres de connexions depuis Openstack GenOuest

Pour définir les variables d'environnement requises pour les clients en ligne de commande openstack, vous devez télécharger le fichier d'environnement appelé *openrc.sh* à partir du tableau de bord openStack de GenOuest[55] en tant qu'utilisateur. Ce fichier d'environnement spécifique au projet contient les informations d'identification que tous les services openstack utilisent.




Il faut récupérer le script *openrc.sh* ainsi que le fichier de configuration *cloud.yml*. Ce dernier est à mettre dans le répertoire *<home directory>/.config/openstack*.

Ensuite exécuter le script *openrc.sh* dans le shell courant; votre mot de passe vous sera demandé:

```
$> . ./openrc.sh
Please enter your OpenStack Password for project <Project> as user <Users>:
```

[54] https://docs.openstack.org/newton/user-guide/common/cli-install-openstack-command-line-clients.html
[55] https://genostack.genouest.org/

<Project> et <User> correspondant à votre configuration.

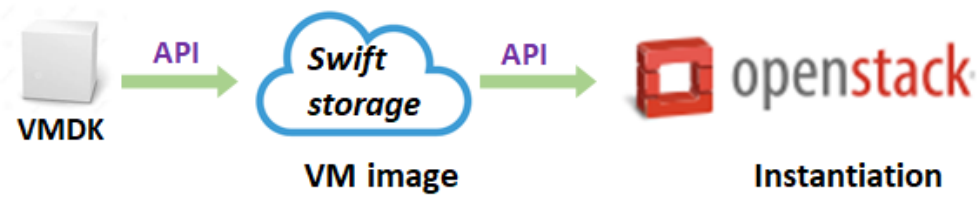
On va définir quelques alias afin de simplifier les commandes ensuites :

```
OS_SCRIPTS=<path of the python-openstackclient scripts>

alias ostack="$OS_SCRIPTS/openstack --os-cloud=openstack --os-password
$OS_PASSWORD"
alias onova="$OS_SCRIPTS/nova --os-password $OS_PASSWORD"
```

Si tout est configuré correctement, la commande suivante devrait vous fournir la liste des “flavors” disponibles. Les “flavors” (saveurs) définissent la configuration matérielle disponible pour un serveur. Elle définit la taille d'un serveur virtuel qui peut être lancé.

```
ostack flavor list

+------------+-------+------+-------+-----------+
 | Name       |   RAM | Disk | VCPUs | Is Public |
 +------------+-------+------+-------+-----------+
 | m2.xlarge  | 16384 |   20 |     4 | True      |
 | m1.small   |  2048 |   20 |     1 | True      |
 | m2.large   |  8192 |   20 |     2 | True      |
 | m1.medium  |  4096 |   20 |     2 | True      |
 | m2.4xlarge | 65536 |   20 |     8 | True      |
 | m2.medium  |  4096 |   20 |     1 | True      |
 | m1.2xlarge | 32768 |   20 |     8 | True      |
 | m1.large   |  8192 |   20 |     4 | True      |
 | m1.xlarge  | 16384 |   20 |     8 | True      |
 | m2.2xlarge | 32768 |   20 |     4 | True      |
 +------------+-------+------+-------+-----------+
```

Nous allons ensuite procéder en deux étapes : 1) création d’une image de VM au sein de la librairie, puis 2) création d’une instance de cette image comme illustré ci-dessous:



### Création de l'image sur l'infrastructure openstack

Il est d'abord nécessaire d'extraire le fichier de la VM au format VMDK de l'archive générée à l'étape précédente.

```
tar xvzf ubuntu-box.tar.gz box-disk001.vmdk
```

C'est ce fichier qui servira à créer l'image. Il faut aussi spécifier un nom pour l'image.

```
IMAGE_NAME=jupyterhub-image

ostack image create --disk-format vmdk \
  --file box-disk001.vmdk $IMAGE_NAME

ostack image set $IMAGE_NAME

ostack image show $IMAGE_NAME
```

### Création d'une instance à partir de l'image créée

A partir de l'image créée, il est maintenant possible de créer une instance. Il faut spécifier le nom de l'instance, sa flavor (cf plus haut) et la clé SSH associée à cette VM. La clé SSH (keypair) doit d'abord avoir été créée depuis le tableau de bord d'openstack.

Vous pouvez lister les clés SSH à l'aide de la commande:

```
ostack keypair list
```

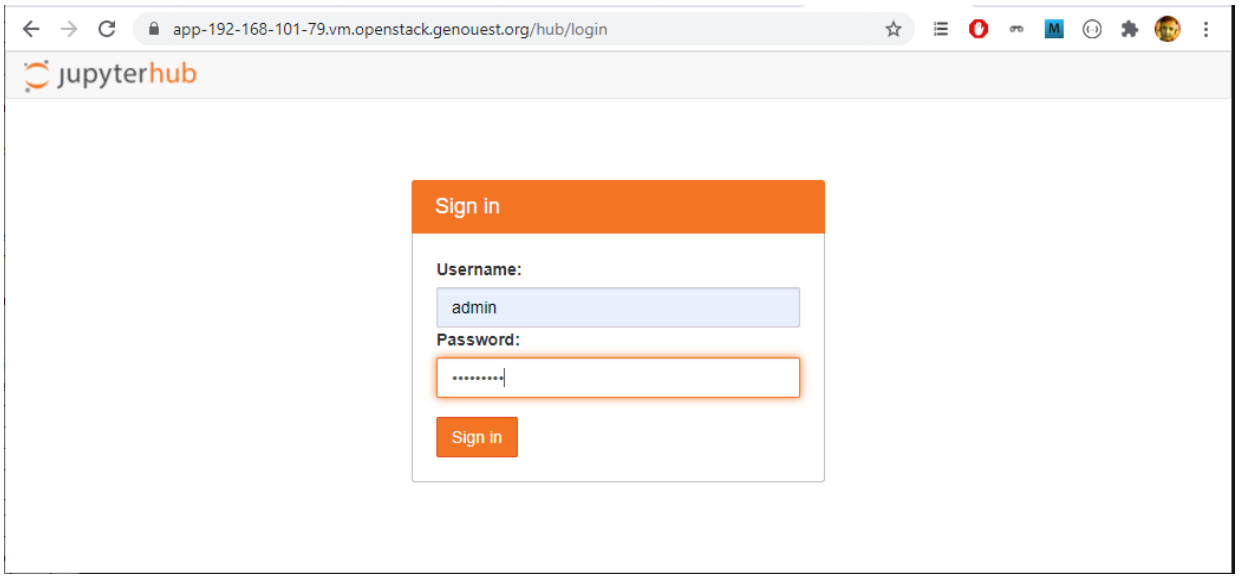
On peut aussi spécifier le fichier contenant les commandes à exécuter après le premier boot. Ici ce fichier est nécessaire (*user-data-jupystack.txt*) car on doit écraser le fichier /usr/local/bin/get-hostname donnant le nom complet de l'instance sur openstack afin de bâtir l'URL racine de l'application jupyterhub.

Dans notre exemple, on choisit une flavor correspondant à 8 CPUs, 16Go de RAM et 20Go de disque. Une clé SSH a été créée ayant pour nom *genostack* dans le tableau de bord

d'openstack. Cette clé SSH doit être une clé valide sur votre compte (linux) d'[openstack.genouest.org](openstack.genouest.org)[56] (fichier `~/.ssh/authorized_keys`).

```
IMAGE_NAME=jupyterhub-image
SERVER_NAME=jupystack
KEYPAIR=genostack
FLAVOR_NAME=m1.xlarge


IMAGEID=$(ostack image show $IMAGE_NAME | \
          grep "| id " | cut -d'|' -f3 | \
          sed -e "s/ //g")
FLAVORID=$(ostack flavor list | \
          grep "$FLAVOR_NAME"  | cut -d'|' -f2 | \
          sed -e "s/ //g")

onova boot --flavor $FLAVORID --image $IMAGEID --security-groups default \
          --user-data ./user-data-jupystack.txt \
          --key-name $KEYPAIR  $SERVER_NAME

ostack server show $SERVER_NAME
```

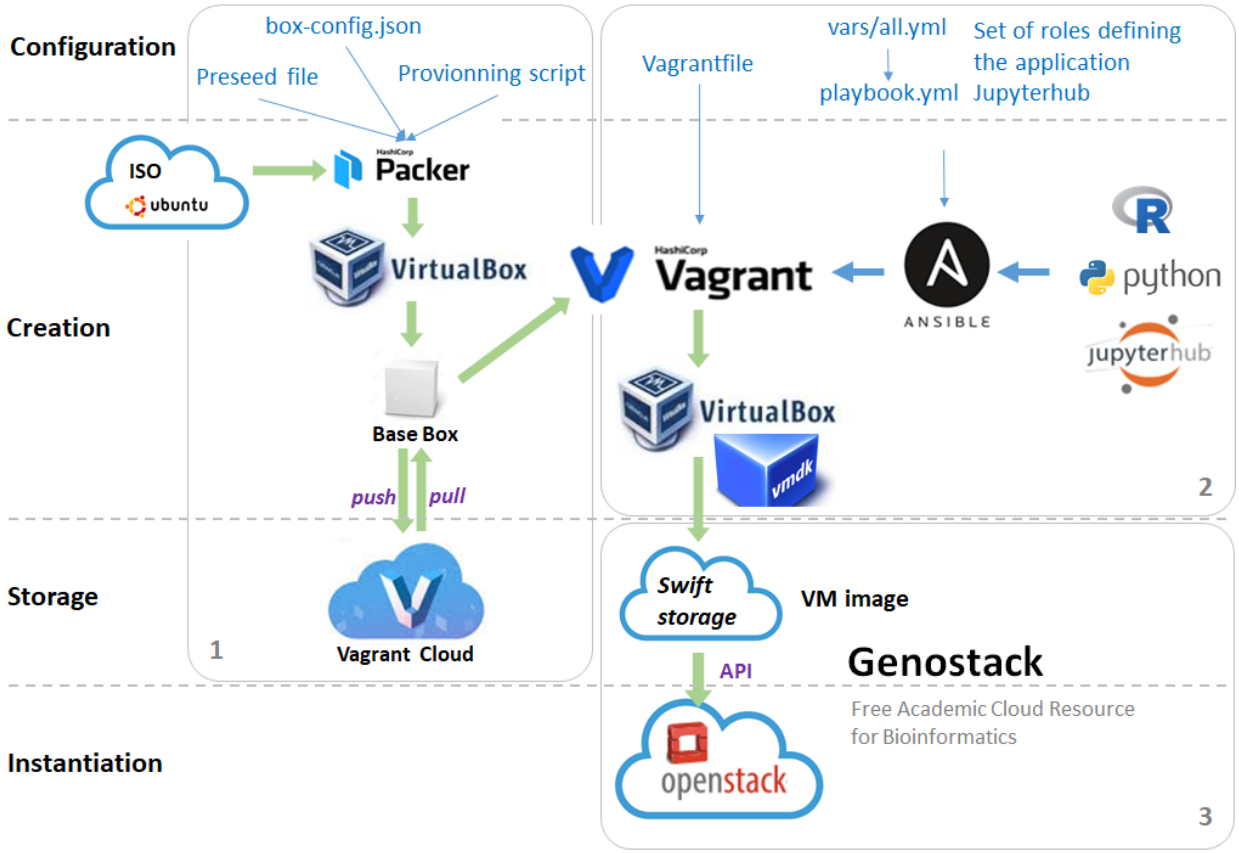
La dernière commande permet d'obtenir le numéro IP de la VM ainsi créée. Supposons que ce numéro IP soit 198.168.101.179.  On peut ainsi accèder à l'application depuis son navigateur web à l'URL : [https://app-192-168-101-79.vm.openstack.genouest.org/hub/login](https://app-192-168-101-79.vm.openstack.genouest.org/hub/login)




[56] http://www.genouest.org/outils/genostack/ssh-config.html

## Résumé de la procédure sous forme d'un schéma

Les cadres gris correspondent aux trois étapes décrites précédemment.




### Les trois grandes étapes:

1. Construction de la VM de base
2. Construction de la VM finale
3. Utilisation de la VM finale sur un cloud Openstack

### Le pipeline:

- En vert, le circuit du pipeline pour la création, stockage et instanciation de la machine virtuelle.
- En entrée du pipeline, un fichier ISO correspondant au système d'exploitation choisi et téléchargé depuis Internet.
- En sortie, une instance de la machine virtuelle opérationnelle sur le cloud Openstack de GenOuest.

Les différentes couches:

- La couche *configuration* correspond à l'ensemble des fichiers et scripts déposés dans le github.
- La couche *création* correspond à l'ensemble des outils installés et utilisés sur la machine locale de l'utilisateur (hormis Ansible qui est installé sur la machine virtuelle mais qui pourrait très bien être installé sur la machine locale).
- La couche *storage* correspond aux sites de stockage des VMs (de base et finale).
- La couche *instantiation* correspond à la machine virtuelle instanciée.

# B2: Automatisation de la création et du partage d'un environnement sous R

Les codes et fichiers sont disponibles sur ces deux dépôts :

- https://forgemia.inra.fr/gaev/usecases/windowsr
- https://forgemia.inra.fr/gaev/usecases/macosr

En reprennent le use-case générique voici les différentes étapes :

1. Choix d'une "VM de base" (Base box) préconfigurée pour le pipeline.
    a. Critères du choix => (OS + Provider)
        i. Utilisation du provider "*VirtualBox*"
        ii. OS ciblé (*Windows* et *MacOS*)
    b. Absence de VM de base (c.à.d conforme aux spécifications requises) :
        i. Construction de la VM de base avec Packer
            1. https://forgemia.inra.fr/gaev/packer/windows10
            2. https://forgemia.inra.fr/gaev/packer/macos
        ii. Dépôt des VMs (boxes) sur https://app.vagrantup.com/GAEV/
2. Construction de la VM finale (Final box) à partir de la VM de base (Base box)
    a. Étape 1 : Construction du playbook d'Ansible
        i. Playbook.yml
        ii. Différents rôles sont disponibles suivant l'OS :
            1. Windows : R, Rtools, msys2, R-package, Miktex, cicd
            2. MacOS : common, R, R-packages, MacTex, Xquartz, cicd
        iii. Les variables définies dans le rôle *R-packages/default/main.yml* peuvent être redéfini dans le playbook pour installer les packages R et librairies systèmes nécessaires.
    b. Étape 2 : Création du template de Vagrant (Vagrantfile)
        i. Configure les caractéristiques de la VM suivant l'OS hôte
        ii. Appliquer l'ensemble des procédures décrites par Ansible et les variables spécifiques situé dans *group_vars*.
    c. Automatisation des étapes précédentes via les GitLab runner du dépôt.
3. Utilisation de la VM finale (Final box)
    a. Utilisation sur son ordinateur via Virtualbox
    b. Partage des boxes via un stockage (S3 DSI INRAE) et VagrantCloud
    c. Instanciation de l'image de la VM comme nouveau GitLab Runner pour les CI/CD

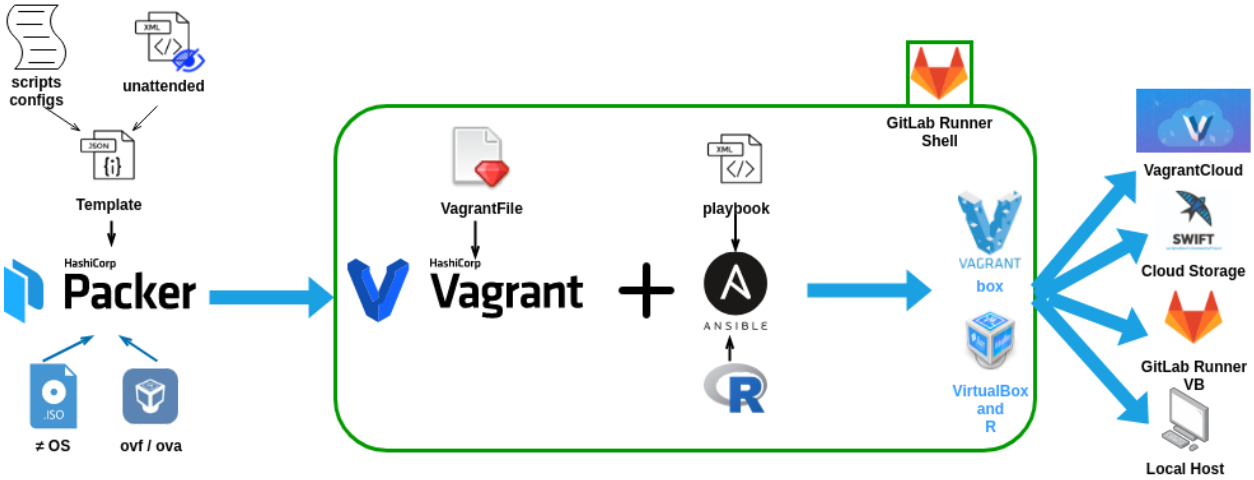



Schéma Use Case environnement R

## B3: Enregistrement d'une box sur Vagrant Cloud

Vagrant Cloud fournit une API permettant aux utilisateurs d'enregistrer leurs machines virtuelles (boxes) sur Vagrant Cloud afin qu'elles puissent être réutilisées par eux-mêmes ou par d'autres utilisateurs. L'utilisation de l'API est décrite en ligne[57].

L'enregistrement d'une machine virtuelles ne peut se faire qu'avec le format Vagrant, c.à.d une "box". Une "box" est en fait un fichier archive zippée (TAR + GZIP) ayant l'extension '.box'.

Enregistrer une "box" comprend 5 étapes :

1. Création d'une entrée pour la "box" : il faut spécifier au minimum le nom de la box (*boxname*). Sa description est optionnelle.
2. Création d'une version : il peut y avoir plusieurs versions pour une même entrée; il faut spécifier la version.
3. Création d'un provider : de même, pour une entrée (*boxname*) et une version, il peut y avoir plusieurs providers associés; il faut spécifier le provider.
4. Télécharger le fichier de la box.
5. Valider la box. (entrée + version + provider)

Toutes ces étapes peuvent se faire soit via l'interface web de Vagrant Cloud[58], soit par plusieurs appels de l'API. Afin de faciliter l'automatisation des enregistrements, l'outil Vagrant fournit une fonctionnalité 'cloud'[59] qui permet d'enregistrer sa box sur Vagrant Cloud. Il faut préalablement avoir créé un compte (appelé 'organisation') sur Vagrant Cloud.

[57] https://www.vagrantup.com/vagrant-cloud/api
[58] https://app.vagrantup.com/
[59] https://www.vagrantup.com/docs/cli/cloud

Exemple d'invocation :

```
$> vagrant cloud auth login
# ...
Vagrant Cloud username: GAEV
Vagrant Cloud password: XXXXXXX

$> vagrant cloud publish GAEV/centos7-dd8Gb 1.0.0 virtualbox
virtualbox-centos7_8Gb.box

You are about to create a box on Vagrant Cloud with the following options:
GAEV/centos7-dd8Gb (1.0.0) for virtualbox
Automatic Release:     true
Do you wish to continue? [y/N] y
Creating a box entry...
Creating a version entry...
Creating a provider entry...
Uploading provider with file /Vagrant/boxes/virtualbox-centos7_8Gb.box
Releasing box...
Complete! Published GAEV/centos7-dd8Gb
tag:                GAEV/centos7-dd8Gb
username:           GAEV
name:                 centos7-dd8Gb
private:              false
downloads:            0
created_at:           2020-07-25T17:53:04.340Z
updated_at:           2020-07-25T18:01:10.665Z
current_version:      1.0.0
providers:            virtualbox
```

La box ainsi enregistrée est consultable sur Vagrant Cloud[60].

[60] https://app.vagrantup.com/GAEV/boxes/centos7-dd8Gb