PROJET TUTEURÉ

Mise en place d'une PaaS

Mars 2022

Thématique : Mise en place d'une PaaS

Module : Projet tuteuré

Etudiants : Bilale BOUMADIENE – Gautier DELPIERRE – Bastien GROSCLAUDE – Axel NICLAUSSE – Louis SAGET

Tuteur : Mathieu GOULIN

Formation : LP ASRALL

Promotion : 2021 – 2022

IUT DE LORRAINE
IUT Nancy-Charlemagne
Département Informatique
2Ter Boulevard Charlemagne
54000 NANCY

IUT DE LORRAINE
IUT Nancy-Charlemagne
Département Informatique
2Ter Boulevard Charlemagne
54000 NANCY

REMERCIEMENTS

Nous souhaitons avant tout à remercier l'ensemble du jury pour leurs riches et variés enseignements tout au long de cette année universitaire. Nous avons pu appréhender ce sujet ainsi que les différentes thématiques sous un meilleur angle.

Nous remercions vivement notre tuteur de projet, Monsieur Mathieu GOULIN, pour son aide et sa disponibilité. Nous avons apprécié son encadrement sur ce projet vaste, passionnant et d'actualité qu'est la mise en place d'une PaaS.

Nous souhaitons enfin remercier l'Université de Lorraine et Monsieur Philippe DOSCH, responsable de formation, qui nous ont offert la possibilité d'effectuer un projet à caractère professionnel dans le cadre universitaire. Ce projet nous a introduit des connaissances indispensables qui nous apporteront une certaine polyvalence sur de futurs projets d'entreprises auxquels nous serons confrontés.

IUT DE LORRAINE
IUT Nancy-Charlemagne
Département Informatique
2Ter Boulevard Charlemagne
54000 NANCY

TABLES DES MATIERES

INTRODUCTION

Aujourd'hui, la rationalisation des coûts et la flexibilité exigés par les projets demandent une adaptation permanente. Le système informatique y est également confronté et doit s'adapter aux nouvelles et différentes contraintes des projets.

Ainsi, on constate l'apparition de nouveaux mouvements dans le monde professionnel comme le DevOps ou la méthode « Agile » qui sont maintenant très présents. Ces mouvements ont émergé avec la promesse de rendre la mise en production plus rapide tout en accroissant l'amélioration et l'innovation des produits ou des services concernés.

Les performances et la qualité des services fournis pour les entreprises pionnières de l'informatique comme les GAFAM (Google, Amazon, Facebook, Apple et Microsoft) ont rendu les clients d'aujourd'hui plus exigeants. En conséquence, beaucoup d'entreprises de toutes les industries prennent désormais exemple afin de mieux se positionner sur le marché et de s'adapter à la compétition et des changements technologiques du monde de l'informatique.

Il est évident que le but pour toute organisation dans l'IT est de mener ses projets à bien et de répondre aux attentes de leurs clients, mais la réalité est que 15% des projets IT prévus sont considérés comme des échecs et que 57% des projets ont respecté leurs budgets initiaux.

La rationalisation des coûts définit l'ensemble des processus visant à contrôler les coûts engendrés par une activité afin qu'elles soient conformes au budget préalablement défini, pour qu'une activité soit considérée comme une réussite elle doit :

- Répondre aux demandes du client.
- Respecter la date limite de livraison.
- Respecter son budget initial.
- Être exécutée dans les règles de l'art.

La rationalisation des coûts fait donc partie des piliers qui composent la gestion de projet et est aussi applicable dans d'autres domaines que l'IT.

Ces nouvelles méthodes de travail demandent aussi de la flexibilité au niveau de l'infrastructure. En effet, un développeur qui a par exemple besoin d'une base de données ou d'un middleware sur mesure ne doit pas perdre de temps à leur installation.

Afin de répondre aux besoins d'adaptation des SI, les administrateurs systèmes doivent mettre en œuvre des solutions de PaaS (Platform as a service). Dans ce type de cloud *computing*, un fournisseur de services fournit une plateforme à ses clients, leur permettant de développer, d'exécuter et de gérer des applications sans avoir à construire et à maintenir l'infrastructure que ces processus de développement de logiciels requièrent généralement.

Ces solutions permettent de déployer de façon standard des composants dans le SI à la demande et en fonction des besoins. Avec ce type de technologie, le SI gagne donc en flexibilité et s'adapte plus facilement aux besoins.

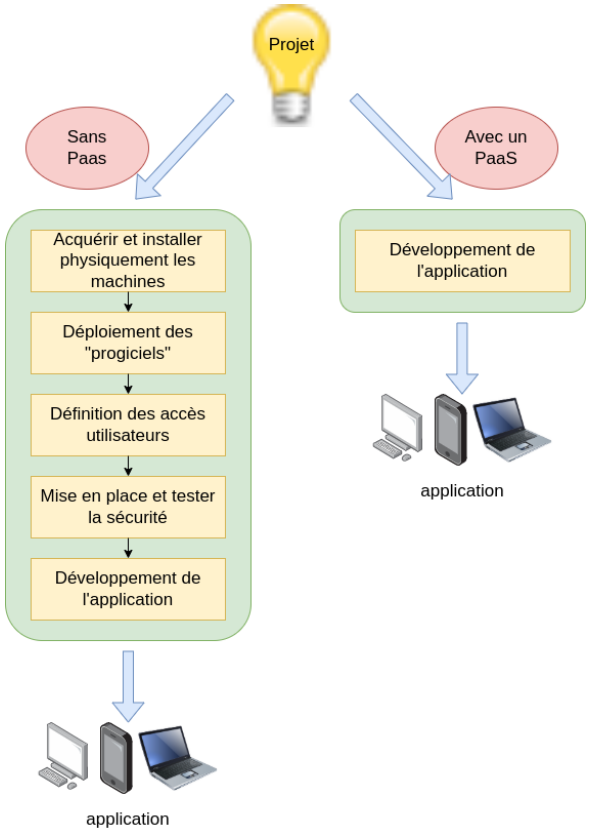

Figure 1 - Avantages d'une PaaS

La solution PaaS imposée et étudiée dans la suite du rapport est constituée de 3 logiciels libres. Après une étude du sujet et des logiciels employés, nous montrerons comment ces 3 logiciels s'inscrivent dans une optique de PaaS et d'adaptation du SI.

Nous commencerons par la définition globale du sujet (les différentes technologies existantes et la solution PaaS imposée). Ensuite, nous aborderons la mise en place de ces technologies, y compris l'installation et la configuration. Nous présenterons par la suite des scénarios d'utilisation pertinents et leur mise en œuvre. Enfin, nous présenterons de manière plus générale notre organisation tout au long du projet.

I- DEFINITION DES TECHNOLOGIES

1) Contexte et solutions courantes

a) Contexte d'une solution PaaS

La solution de PaaS est de nos jours une solution qui permet à n'importe quelle organisations de se démarquer et de gagner aussi bien en productivité qu'en simplicité dans un contexte de concurrence.

La plateforme permet en effet d'être plus rapide. C'est par exemple le cas avec la mise en œuvre des couches de stockages, des serveurs, des virtualisations ou de tout autre middleware qui est pris en charge par la plateforme. Cela permet donc de se concentrer sur les projets en cours et non sur la gestion des couches d'infrastructure qui sont désormais disponibles immédiatement.

La tendance est d'avoir à disposition un workflow intégré du début à la fin du projet. Cela peut aller de l'IDE Cloud sur lequel on code jusqu'au repository source comme GitHub. Des outils intermédiaires d'automatisation de tests et de déploiement sont également possibles.

L'unique inconvénient d'une solution PaaS réside dans la dépendance de l'utilisateur à l'infrastructure et au logiciel du fournisseur. En d'autres termes, à moins d'adapter la PaaS à chaque cas d'utilisation et de la tenir à jour, les utilisateurs dépendent de ce qui est déjà mis en place sur la PaaS. La mise en place d'une PaaS peut se faire de deux façons différentes :

- Il est possible de mettre en place une PaaS publique qui permettra l'administration à grande échelle par un organisme tiers. Il sera donc nécessaire de payer un abonnement. C'est un inconvénient supplémentaire qui renforce la dépendance de l'utilisateur au fournisseur. En effet, la PaaS doit être gardée à jour face aux contraintes évolutives. En revanche, cela permet aux petites et moyennes entreprises d'avoir un accès à une PaaS.

- Il est aussi possible de développer son propre environnement avec le cas d'une PaaS privée. Cette possibilité est plus souvent présente dans des grandes organisations car cela nécessite du personnel formé pour garder la PaaS à jour et pour l'adapter continuellement aux utilisateurs.

b) Définition de critères de comparaison

Les critères suivants sont importants dans le choix d'une solution PaaS. Nous nous servirons de ces critères que nous avons choisis pour comparer les différentes solutions PaaS par la suite :

- Le prix du PaaS et de son utilisation.
- Les langages et les technologies supportées.
- Les contrats de résilience et de sécurités des données.
- Le support disponible pour aider à mettre en place la PaaS.
- La possibilité de superviser et de monitorer la PaaS.
- Les bases de données utilisées ou possiblement utilisables par le PaaS.

c) Comparaison de solutions

Aucune PaaS ne répond à toutes les contraintes et cela nécessite parfois de combiner plusieurs plateformes.

Dans notre cas nous utiliserons la solution PaaS AWX car elle nous est imposée. Cependant nous pensons qu'il est important de pouvoir la comparer à d'autres solutions afin d'avoir une idée des solutions possibles sur le marché.

	Heroku	Engine	Digital Ocean	Ansible Tower	AWX
Coût	Démarrage gratuit puis plusieurs offres disponibles	Payant	Payant	Payant	Gratuit et Open Source
Langage de programmation	Node.js, Ruby, PHP, Python, Go, Scala…	Node.js, Ruby, PHP, Python, Java	Node.js, Ruby, PHP, React, Laravel	Node.js, Ruby, PHP, Python, Go, Jenkins, Drupal	YML
Système de supervision	Non	Oui	Oui	Oui	Oui
Support	24/7 payant, 12/5 gratuit	24/7	24/7	24/7	Non
Sécurité	Certificat TLS et beaucoup d'autres features	Non communiqué	Certificat SSL	Certificat SSL	Certificat SSL
Base de données	MySQL, HerokuPostgres, MongoDB…	Engine Yard	Non communiqué	MySQL, Postgres, MongoDB	MySQL, Postgres, MongoDB

Tableau 1 - Tableau comparatif

2) Solution PaaS étudiée

a) Ansible

Figure 2 - Logiciel Ansible

Ansible est un logiciel libre pour la configuration et la gestion des ordinateurs. Il est développé par Red Hat. Ansible permet plus particulièrement l'automatisation du SI au moyen d'un ensemble de ressources et grâce à la prise en charge en parallèle des éléments suivants : déploiement de logiciels multi-nœuds, l'exécution des tâches ad-hoc, et la gestion de configuration.

Une tâche est une unité d'exécution, aussi appelée unité de travail. Il s'agit d'un terme global qui peut être précisé par une dénomination plus spécifique tel que le processus, le processus léger, le fil d'exécution et le mécanisme de requête-réponse. L'exécution de tâches représentera dans notre cas l'exécution de scénarios (plus ou moins complexes avec ou non des dépendances) dans le SI.

La gestion de configuration consiste quant à elle à :

- Gérer la description technique d'un système (et de ses divers composants).
- Gérer tout au long de l'évolution du système, l'ensemble des modifications apportées.

En d'autres termes, il s'agit de l'ensemble des processus permettant d'assurer la conformité d'un produit aux exigences auxquelles il est confronté, tout au long de son cycle de vie. Dans notre cas d'utilisation, il s'agira donc d'adapter le SI aux différentes contraintes et de le faire évoluer.

Ansible améliore les ressources sur plusieurs systèmes pour les gérer à partir d'une unique plateforme. Les systèmes gérés sont les cibles du système de gestion de configuration et peuvent être : des serveurs, le stockage, la mise en réseau ou des logiciels.

Le code, le cycle de vie et les modifications peuvent être gérés par le biais d'inventaires, de playbooks et de rôles que nous verrons par la suite.

Ansible gère les différents nœuds grâce au protocole SSH et ne nécessite l'installation d'aucun logiciel supplémentaire sur ceux-ci. De plus, Ansible peut gérer des nœuds sur une multitude de systèmes d'exploitation. Ces derniers aspects représentent un réel avantage permettant un déploiement simplifié sur le réseau.

En effet, le protocole SSH (Secure SHell) est à la fois un programme informatique et un protocole de communication sécurisé imposant un échange de clés de chiffrement en début de connexion.

Le système des fichiers au format YAML pour exprimer des descriptions réutilisables de systèmes, appelés playbooks que nous verrons par la suite, permettant d'automatiser la gestion de configuration.

Le format de fichier YAML (acronyme du Yet Another Markup Language) reprend des concepts d'autres langages comme le langage XML. Son objectif est de représenter des informations plus élaborées que le simple format CSV tout en gardant une lisibilité presque comparable, et bien plus grande en tout cas que le format XML.

b) AWX

Figure 3 - Logiciel AWX

AWX est un produit conçu par Red Hat. Ces derniers ont d'ailleurs sorti une version non libre d'AWX : Ansible Tower. C'est une version stable d'AWX et les mises à jour sortent légèrement plus tard que sur AWX. Les mises à jour sur Ansible Tower, sont définitives et fonctionnelles. Un des avantages majeurs d'AWX (et aussi d'Ansible Tower) est sa centralisation des playbooks par Git. Elle simplifie grandement la gestion des projets.

AWX se présente donc un peu comme une couche additionnelle à Ansible mais reste totalement transparent sur l'exécution des playbooks. Il ne complexifie pas non plus l'utilisation d'Ansible et des playbooks, ce qui constitue également une des forces du produit.

c) Squest

Figure 4 - Logiciel Squest

Squest est un Framework web Open Source sous forme de projet GitHub soutenu par Hewlett Packard. Il est développé en Python et le projet est lancé en avril 2021. Le projet reste relativement actif dans son développement avec environ 2 mises à jour par mois.

Le portail web de Squest a été développé en grande majorité depuis un autre framework existant également en Python. Cet autre portail se nomme Django et reste cependant conçu pour des sites web.

Squest fonctionne nativement de façon conteneurisée par docker ce qui permet un déploiement plus simplifié sur un large type de distributions et de configurations.

Il permet d'exposer un portail Web pour du EaaS (Everything As A Service) avec un catalogue de service qui pour chacun d'entre eux aboutit à l'exécution d'un template AWX défini au préalable dans ce dernier.

Squest communique avec AWX à travers son API REST. Il accède donc à AWX avec des droits d'écriture grâce à un jeton d'authentification généré au préalable et inséré dans Squest.

Dans cette thématique de PaaS, le logiciel Squest est réservé aux utilisateurs finaux qui souhaitent modifier le SI à la demande sans pour autant avoir de lien avec l'administration des systèmes et des réseaux. Les administrateurs systèmes et réseaux peuvent quant à eux être amenés à utiliser cette interface même si l'utilisation d'AWX est plus adaptée à leur besoin.

d) Schéma d’architecture

Avant de définir notre solution, voici le schéma d’architecture représentant en première partie les utilisateurs. Il s’agit d’un côté des administrateurs systèmes et réseaux qui vont utiliser principalement AWX et d’un autre côté les DevOps par exemple, qui vont principalement utiliser Squest.

Les DevOps vont se connecter sur l'interface web du serveur Nginx de Squest afin d’accéder au logiciel Squest qui va ensuite communiquer avec AWX afin d'exécuter les playbooks. Nous expliquerons cela en détail par la suite.

Le deuxième utilisateur possible est un administrateur système et réseaux. Il peut effectuer 3 actions :

- Se connecter sur Squest et dans ce cas la démarche sera identique à celle d’un DevOps.
- Accéder à GitHub grâce à une connexion Git afin d’importer des playbooks créés au préalable dans un dossier GitHub aussi appelé git repository. Les playbooks seront ensuite stockés dans la base de données d’AWX (MySQL).
- Se connecter à l’interface web AWX.

Si un administrateur souhaite se connecter à AWX, la démarche sera identique à celle où un DevOps souhaite se connecte à Squest. Dans un premier temps, il se connecte au serveur apache d’AWX et peut ensuite effectuer tous les changements nécessaires sur le logiciel AWX.

De manière plus globale et au sein de cette solution, AWX représente l’orchestrateur du PaaS. Tous les playbooks transitent forcément par AWX avant d’être déployés et exécutés par Ansible sur les différentes machines.

Lorsqu’un playbook demande à être exécuté par un administrateur ou par Squest (commandé par un DevOps), AWX va récupérer le playbook dans sa base de données. Il va ensuite l’envoyer au logiciel Ansible de la machine *manager* qui va enfin l’envoyer aux machines *nodes* grâce au protocole SSH. Les machines *nodes* vont recevoir les ordres d'exécutions et renvoyer une exécution sans problème ou avec les différents problèmes rencontrés avec les différents codes d’erreurs. Les messages seront remontés jusqu’à l’administrateur système ou au DevOps ayant ordonné l'exécution du playbook.

Ensuite, il s'agit de la partie infrastructure contenant les gestionnaires de conteneurs et donc les logiciels AWX et Squest. Ces derniers sont contenus sur la machine manager qui va ensuite communiquer avec les différents *node* grâce au protocole SSH que nous définirons par la suite.

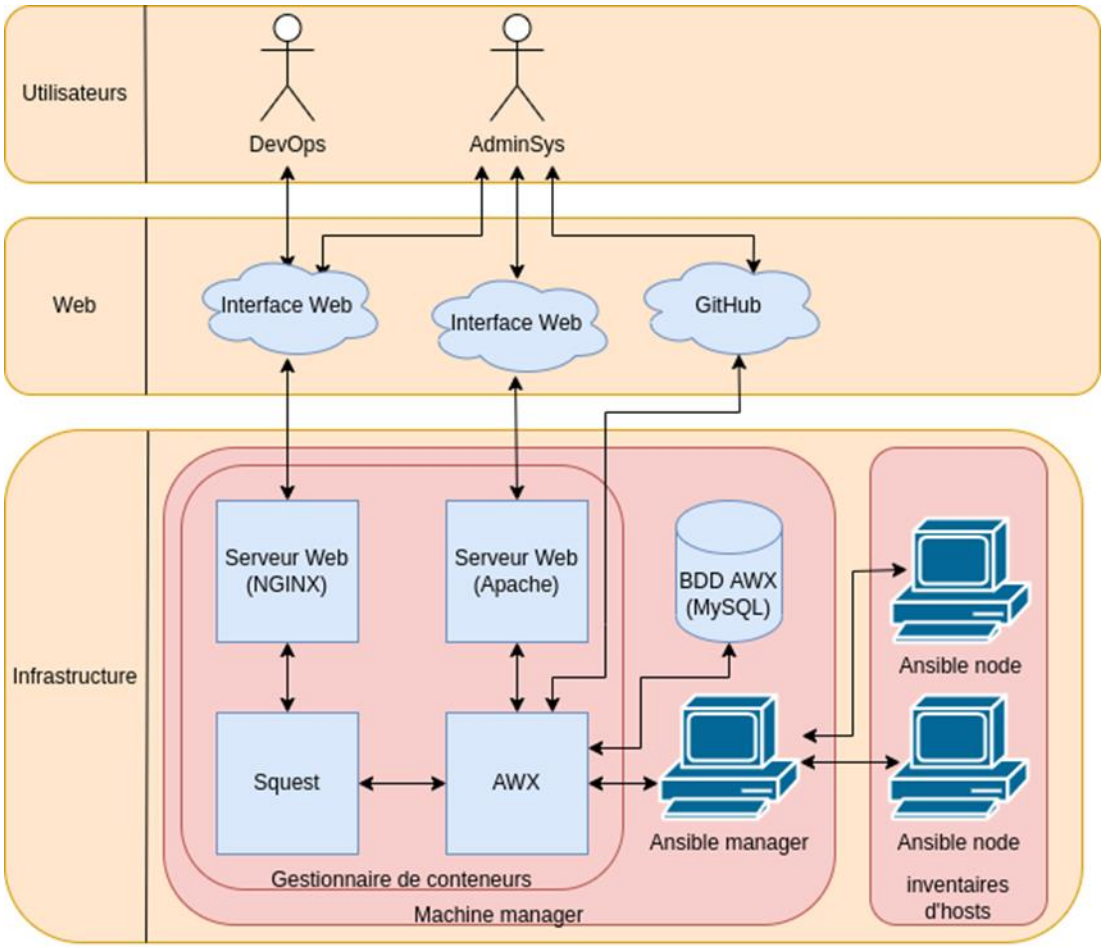

Figure 5 - Schéma d'architecture

II- INSTALLATION ET UTILISATION DES TECHNOLOGIES

1) Ansible

a) Installation

L'installation d'Ansible est très simple et peut se faire de plusieurs manières. La plus simple est de l'installer depuis le gestionnaire de paquets. En effet, le paquet Ansible est présent dans la plupart des dépôts natifs des distributions Linux.

La deuxième option et la plus sure pour des questions de compatibilité consiste à utiliser l'outil Python du gestionnaire de paquets (*pip*) et à obtenir la dernière version du code source d'Ansible depuis le dépôt GitHub. Il faut dans un premier temps installer python. Il faut ensuite installer le paquet *ansible*.

b) Utilisation

L'exécution des playbooks sur les machines clientes nécessite au préalable l'installation de Python. De plus, il faut établir une connexion SSH entre le *manager* et les différents hosts (ou *nodes*) pris en charge.

Même si ce n'est pas présent dans le projet, Ansible est utilisable de façon autonome depuis son interface en ligne de commande. Dans le cadre de cet usage, il faut cependant définir son inventaire de machine manuellement en créant un fichier en YAML. Voici un exemple d'utilisation d'Ansible en CLI :

```
[root@rhel-8 ansible]# ansible-playbook greetings.yml

PLAY [all] *********************************************************************

TASK [Gathering Facts] *********************************************************
ok: [173.82.115.165]
ok: [173.82.202.239]

TASK [Ansible Basic Variable Example] ******************************************
ok: [173.82.115.165] => {
    "msg": "Hello world!"
}
ok: [173.82.202.239] => {
    "msg": "Hello world!"
}

PLAY RECAP *********************************************************************
173.82.115.165             : ok=2    changed=0    unreachable=0    failed=0
kipped=0    rescued=0    ignored=0
173.82.202.239             : ok=2    changed=0    unreachable=0    failed=0
kipped=0    rescued=0    ignored=0
```

Figure 6 - Ansible : output

Cette interface est également visible depuis l'interface web dans AWX que nous verrons par la suite.

2) AWX

a) Installation

L'installation d'AWX depuis la version 18 passe par une solution de conteneurisation Kubernetes. Cela engendre le déploiement des conteneurs nécessaires à la mise en service d'AWX.

En effet, la conteneurisation en informatique permet de packager tous les services, scripts, API et librairies dont une application a besoin. Cela permet l'exécution sur n'importe quel noyau compatible et également :

- D'éviter de se soucier d'interactions ou d'incompatibilités avec des conteneurs déjà présents.
- De ne pas occuper autant de ressources qu'une machine virtuelle avec son propre système d'exploitation.

Les étapes d'installation de la solution de conteneurisation pour la mise en service d'AWX sont les suivantes :

- Installer Kubernetes (k3s).
- Cloner depuis le dépôt git officiel AWX Operator qui est un approvisionneur de conteneurs Kubernetes. Cela permet d'installer correctement les différentes composantes d'AWX dans plusieurs conteneurs définis dans ce dernier.
- Construire le déploiement dans Kubernetes depuis le code source acquis grâce à un clonage avec *make*.
- Configurer et lancer les conteneurs Kubernetes afin que ces derniers puissent correctement faire fonctionner le logiciel AWX.
- Extraire le mot de passe temporaire du compte administrateur de l'interface web depuis un conteneur.
- Se connecter sur l'interface web sur compte « admin » avec le mot de passe acquis précédemment.

En ce qui concerne la configuration, la connexion avec Ansible se fait automatiquement.

b) Utilisation

AWX comporte différentes et importantes fenêtres à l'exécution des playbooks mais un utilisateur d'Ansible aguerri n'aura pas de mal à se retrouver dans ces dernières. Un projet Ansible basique regroupe un ensemble d'éléments. AWX rajoute uniquement d'autres fonctionnalités d'automatisation qui seront utiles aux utilisateurs. Comme dans des projets Ansible basiques, AWX a besoin de plusieurs éléments :

- Les inventaires : la fenêtre « Inventories » est liée à la fenêtre hosts. Elle permet en effet de définir des groupes d'hosts (ou machines). C'est relativement intéressant dans le cas où un administrateur souhaiterait déployer des playbooks sur un ensemble de postes spécifiques appartenant au même réseau.

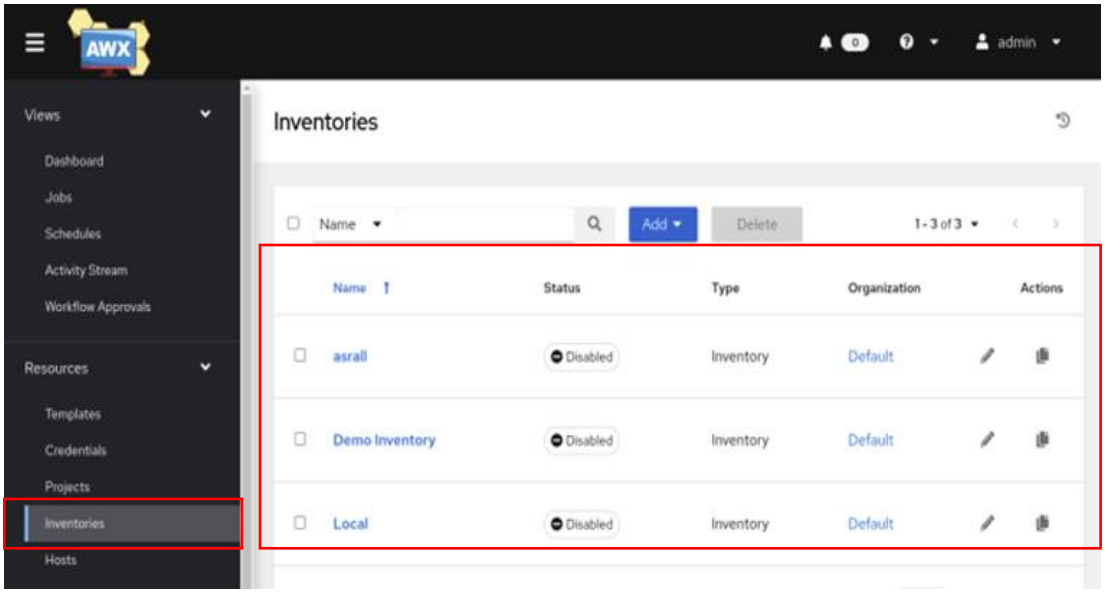

Figure 7 - AWX : Inventories

- Les hosts : la fenêtre « Hosts » permet de définir la totalité des hosts (ou machine) du réseau. Chacun de ces hosts appartiennent à un inventaire défini au préalable. C'est à partir de cette fenêtre que l'on peut également désactiver ou réactiver au cas par cas certains hosts sur lesquels on souhaite (ou non) exécuter des playbooks.

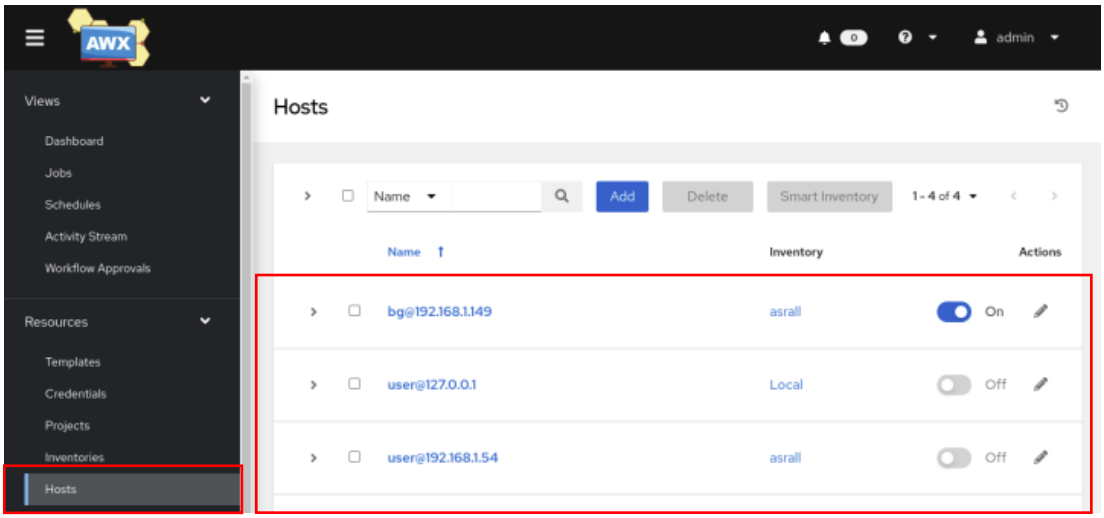

Figure 8 - AWX : Hosts

- Les credentials : la fenêtre « Credentials » contient les clés RSA privée de la machine *manager* (ou machine *master*) qui va exécuter les playbooks sur les clients. Cela s'effectue à condition que les postes clients contiennent la clé publique de la machine *manager* dans les clés publique autorisées.

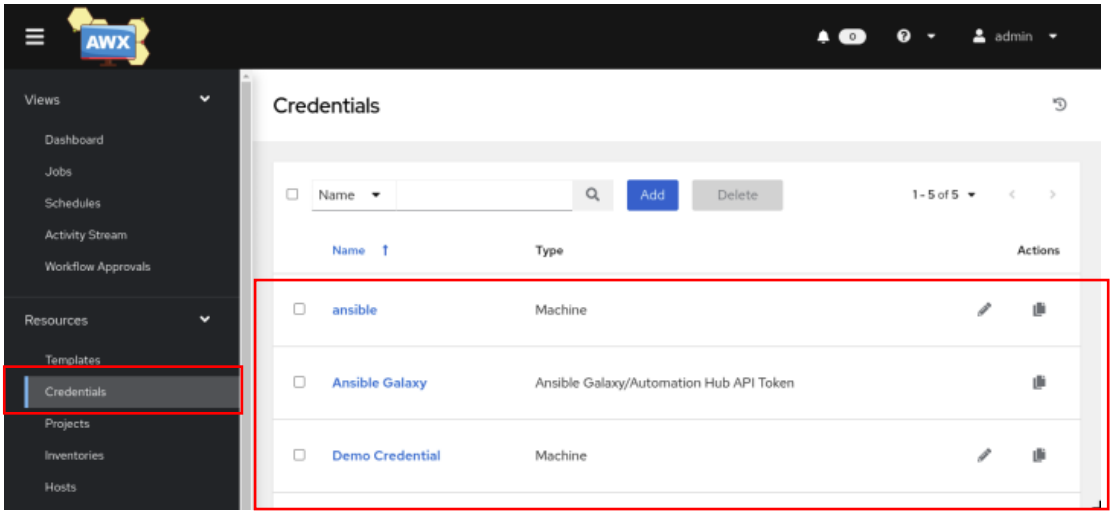

Figure 9 - AWX : Credentials

- Les projets : la fenêtre « Projects » peut contenir les liens vers les répertoires Git (ou repository Git) sur lesquelles les administrateurs publient les playbooks. Les projets peuvent également être stockés en local.

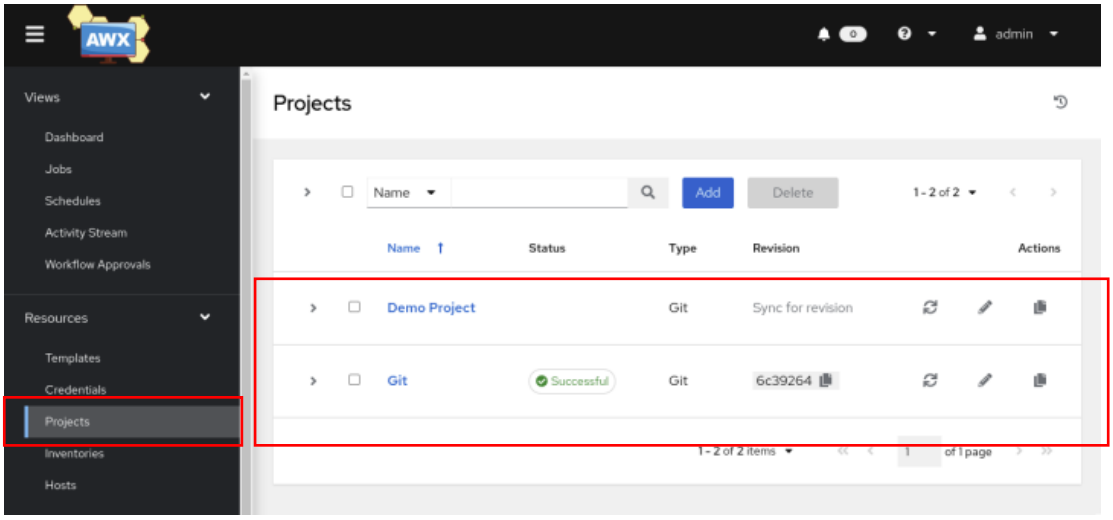

Figure 10 - AWX : Projects

- Les templates : Pour finir, la fenêtre « Templates » permet de configurer le choix du playbook et du dépôt Git (ou autre technologie de stockage de projets). La définition de templates nécessite de définir au préalable des hosts, des projets et des credentials. En effet, les templates connectent ces 3 éléments ensemble pour permettre l'exécution du bon playbook (appartenant à un projet spécifique) sur le bon groupe d'hosts en utilisant les bons credentials. Il est aussi possible depuis cette fenêtre de faire des « workflow template », ce qui permet de programmer une suite de playbook sur des hosts définis.

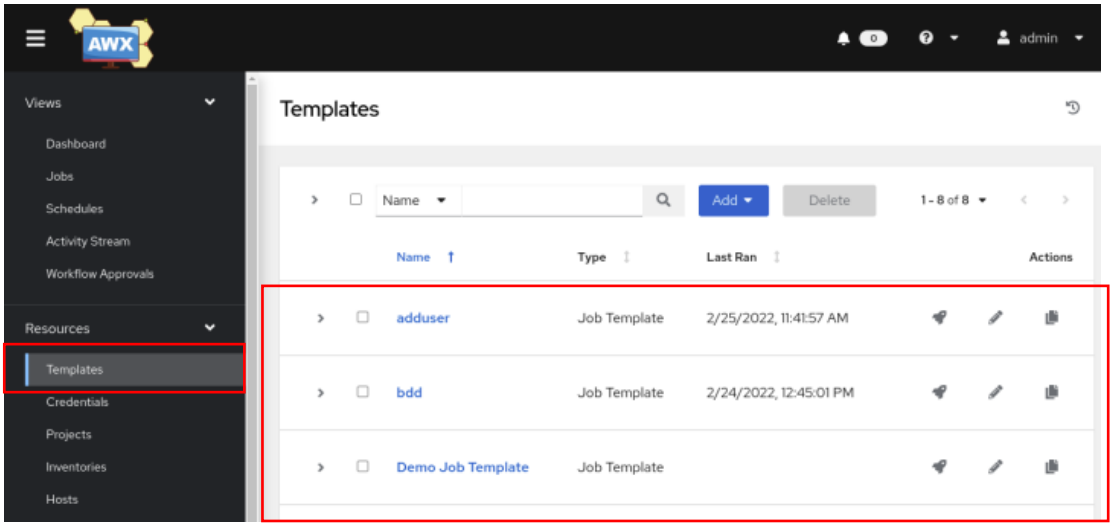

Figure 11 - AWX : Templates

- Une fois qu'un ensemble de templates a été produit et ajouté, on peut exécuter les templates qui donneront lieu à des jobs. Un bouton adéquat permet de lancer les templates. Lorsque ces derniers ont fini d'être exécutés, nous pouvons accéder à la sortie en CLI d'Ansible disponible sur l'interface graphique d'AWX.

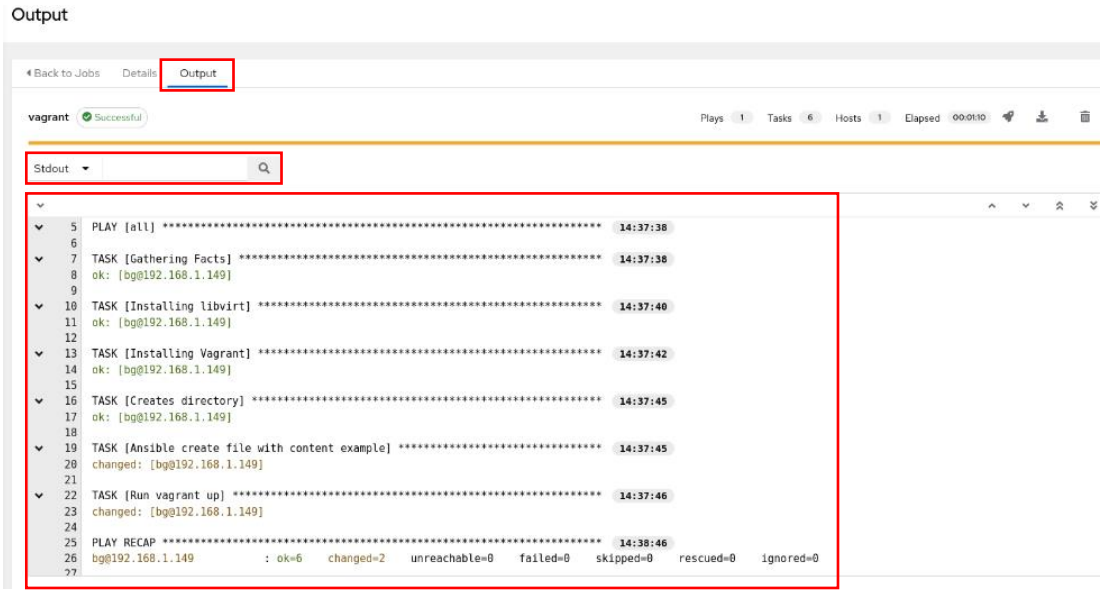

Figure 12 - AWX : output

- La signification du code couleur varie sur la sortie de l'exécution des templates :
  - En vert, il s'agit des tâches qui n'ont pas besoin d'être exécutées ou qui n'ont pas besoin d'écrire sur la machine cible.
  - En jaune, on peut voir les tâches exécutées avec succès sur les machines cibles et qui ont modifié le contenu.
  - Le compteur unreachable compte le nombre d'hôtes inaccessibles.
  - Le compteur failed compte le nombre de tâches où il y a eu une erreur lors de l'exécution.
  - Le compteur skipped compte le nombre de tâches qui ont été passées car la condition qui leur est liée n'a pas été remplie. Cet indicateur n'est pas signe d'erreur.
  - Le compteur rescued compte le nombre de tâches récupérées si le playbook a été conçu avec de la gestion d'erreur.
  - Enfin le compteur ignored compte le nombre de tâches ignorées toujours avec une conception du playbook avec de la gestion d'erreur.

3) Squest

a) Installation

L'installation de Squest se fait par Docker, système de conteneurisation. Pour mettre en place ces derniers, il faut :

- Installer Docker et Docker Compose.
- Cloner depuis le dépôt Git officiel toute l'arborescence du projet Squest.
- Dans le répertoire Squest créé après avoir cloné le dépôt, il faut allumer le conteneur. Docker compose est un système d'orchestration de conteneurs qui va provisionner et configurer nos conteneurs afin de créer l'instance Squest.
- Après s'être connecté avec les login par défaut dans Squest depuis un navigateur, il suffit de connecter AWX à Squest avec son adresse et son jeton d'authentification.

Après avoir installé Squest, il faut maintenant le connecter à AWX. Pour cela, il faut créer les accès adéquats pour l'application Squest sur l'interface d'AWX :

- Il faut d'abord créer un jeton (« token ») pour l'application Squest dans l'onglet « Users ».

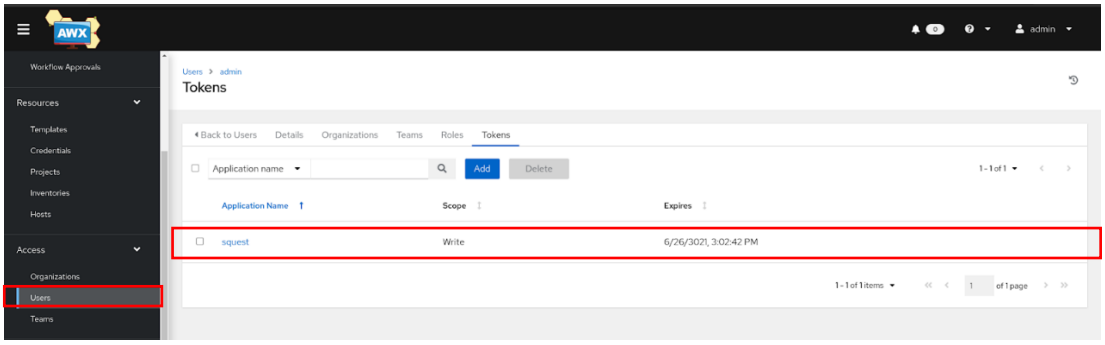

Figure 13 - AWX : Users

- Enfin sur Squest, il faut importer ce token. Dans l'onglet « Tower/AWX » il faut donner à notre serveur un nom, une adresse IP et le token généré précédemment.

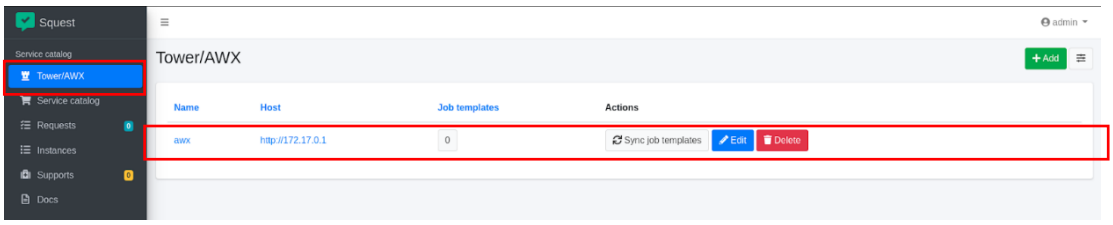

Figure 14 - Squest : Tower/AWX

b) Utilisation

- Une fois AWX connecté, il faut manuellement ajouter les différents services au catalogue. Pour cela, il faut se rendre dans l'onglet « Service catalog » et choisir « job template » que l'on a ajouté à AWX auparavant. On retrouve ensuite tous les services proposés.

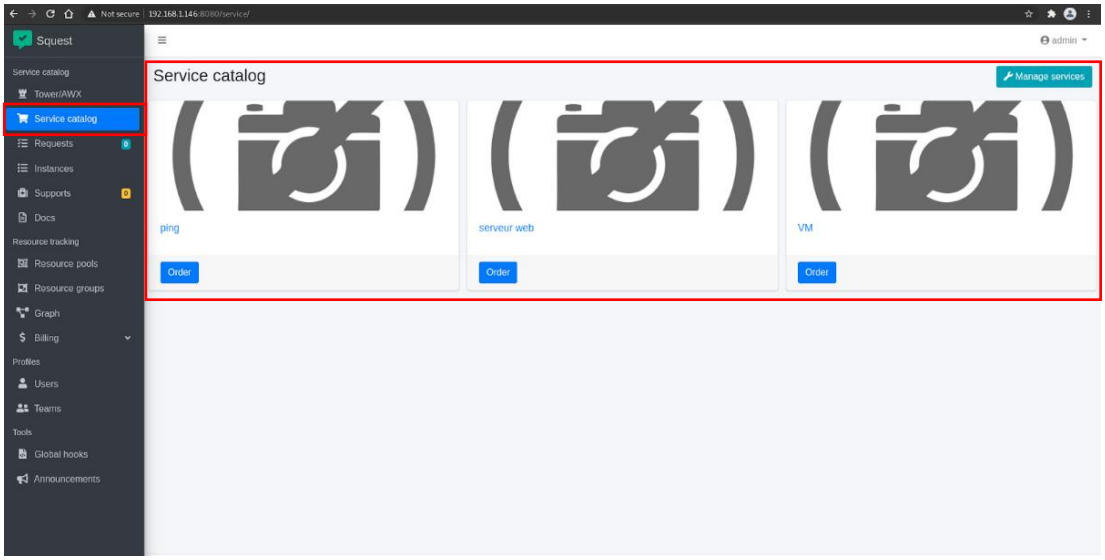

Figure 15 - Squest : Service catalog

- Lorsque le DevOps commande un service, nous devons renseigner les différentes variables dans un formulaire. En effet, il arrive que le DevOps ait besoin de personnaliser certains services avec des spécificités. Voici un exemple de formulaire à compléter pour l'exécution d'un template.

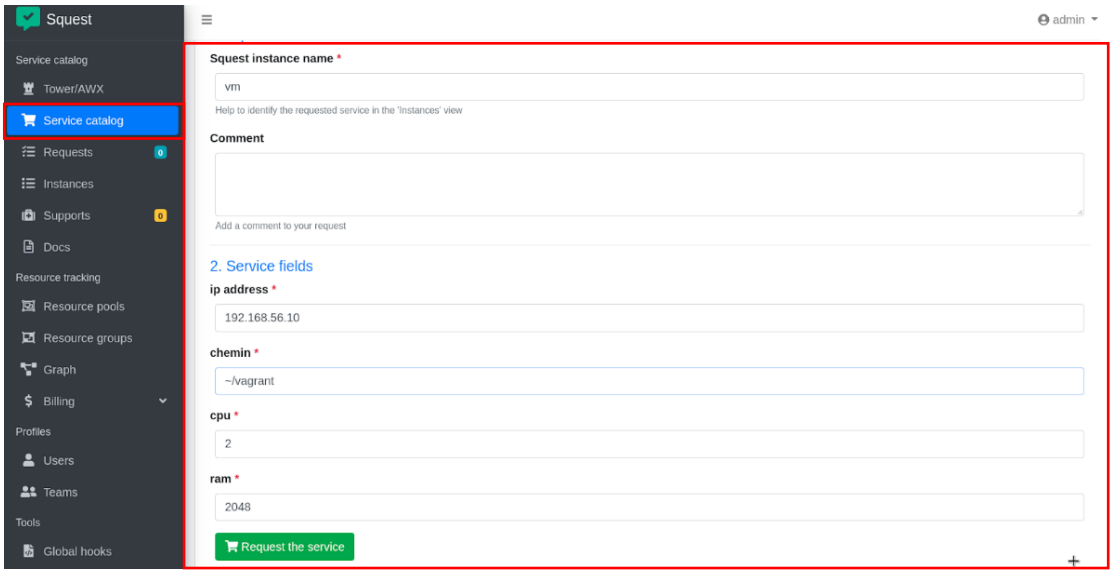

Figure 16

- Une fois que le service est demandé, pour des raisons de sécurité, l'administrateur doit le valider dans la fenêtre dédiée.

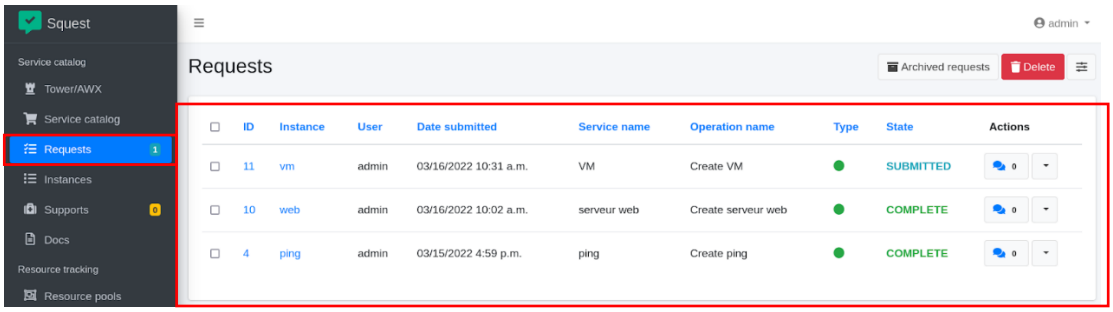

Figure 17 - Squest : Requests

- Une fois accepté, on peut retrouver une trace de tous les services fonctionnels dans l'onglet instances. Cela permet une meilleure gestion et une meilleure traçabilité.

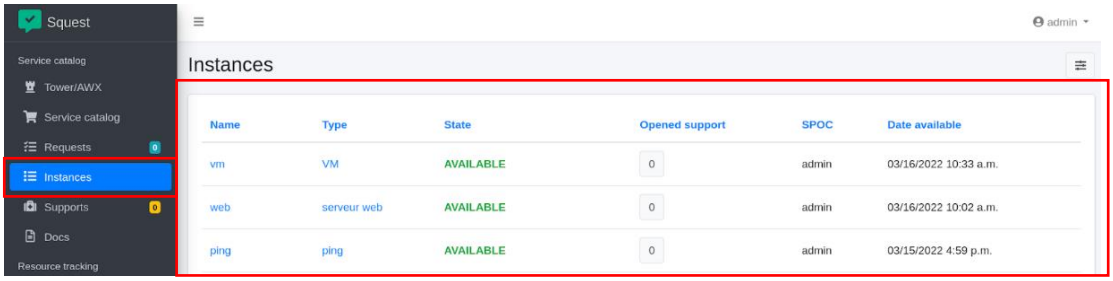

Figure 18 - Squest : Instances

Squest permet également aussi de faire un suivi des ressources. Lorsqu'un DevOps envoie une requête pour demander la création d'une nouvelle machine virtuelle, il doit renseigner ses caractéristiques (nombre de cpu et de mémoire par exemple). Du côté de l'administrateur, pour valider ou non le service, il doit s'assurer de pouvoir fournir les ressources demandées. Pour cela, il faut se rendre dans l'onglet « Graph » et vérifier les ressources disponibles. Une fois le service validé, on peut observer en temps réel l'utilisation des ressources, ce qui permet de mieux surveiller le système à la suite des modifications.

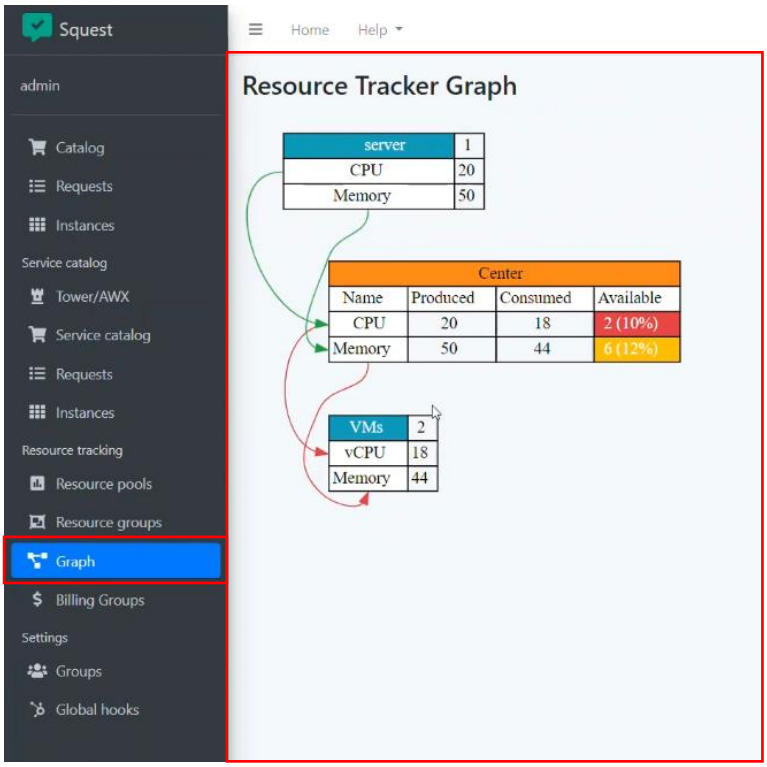

Figure 19 - Squest : Resource tracking

De nombreuses autres fonctionnalités comme les métriques (« metrics ») sont également possibles. Dans le cadre de notre thématique de mise en place d'une PaaS, nous ne nous sommes pas attardés sur ces points-là en partie à cause du manque de temps.

III- DEFINITION DE CAS D'UTILISATION

1) Définition de playbooks

Ansible permet le déploiement de playbooks. Ces playbooks offrent un système de gestion de configuration et de déploiement multi-machine reproductible, réutilisable, simple et bien adapté au déploiement d'applications complexes. Les playbooks sont des fichiers écrit au format YAML. Ils sont exécutés par Ansible et sont composés de 2 parties :

- L'entête : elle définit l'inventaire sur lequel les tâches sont appliquées ainsi que l'utilisateur utilisé pour cette tâche (comme le super utilisateur).

- Les tâches : Chaque tâche contient sa description, le module qu'elle appelle et sa fonction, ses arguments voire des conditions à son exécution.

Lies de haut en bas, elles sont exécutées sur les inventaires selon les potentielles conditions qui peuvent être définies. Chaque playbook peut en appeler un autre afin de mieux décomposer les tâches.

Un playbook Ansible est exprimé au format YAML. Ces playbooks contiennent différents rôles possédant une ou plusieurs tâches qu'Ansible doit exécuter. Cela peut aller de l'adaptation de l'environnement informatique de production jusqu'à l'orchestration de toutes les étapes de déploiement d'une application ou d'une mise à jour sur celui-ci. Une structuration commune d'un playbook à titre d'exemple peut se présenter comme ceci :

Figure 20 - Exemple d'un playbook

Voici la structure d'un playbook contenant les rôles d'installation uniquement des serveurs Apache et MariaDB. Des structures préalablement préparées sont disponibles sur la plateforme Ansible-Galaxy.

Ansible Galaxy fait référence au site web de Galaxy où les utilisateurs peuvent partager publiquement des rôles (de la même façon que GitHub propose aux développeurs de stocker et de partager, publiquement ou non, le code qu'ils créent).

La syntaxe quasi-naturelle utilisée pour décrire ces tâches est particulièrement simple. Le fichier YAML peut commencer par décrire les hôtes (ou serveurs) ciblés puis les variables du processus avant de dérouler les jobs à accomplir. Voici l'exemple d'un rôle effectuant un Ping sur n'importe quelle machine :

```
- hosts: all
  tasks:
   - name: test
     ping:
```

Figure 21 - Exemple d'un rôle

Les fichiers YAML appellent des modules Ansible pour orchestrer le processus de déploiement. Pour lancer les playbooks, il faut d'abord s'assurer que la syntaxe des fichiers ne contient pas d'erreur.

2) Définition de scénarios

Pour tout développeur de site web, nous avons une solution pour lui fournir un serveur web avec une solution sans développement (no-code). Cela lui permettra de mettre en ligne du contenu en gagnant beaucoup de temps de développement.

La solution qui lui est proposée dans notre sujet est le framework Wordpress polyvalent. Celui-ci fonctionne en effet grâce à une base de données MySQL (MariaDB). Ce SGBD est administrable grâce à PhpMyAdmin. Tous les fichiers de la machine sont accessibles grâce au protocole FTP (File Transfer Protocol, protocole de communication destiné au partage de fichiers sur un réseau) grâce à ProFTPd (serveur FTP libre).

Tous ces services sont déployés par Ansible à travers plusieurs playbooks qui peuvent s'appliquer à de nombreux scénarios. Ces playbooks peuvent être utilisés séparément ou ensemble dans le cadre de projets. Les principaux playbooks sont présents en Table des annexes, sans leurs dépendances.

- Le playbook de système de gestion de base de données (SGBD) :
    - En premier lieu, il va installer un serveur de base de données MariaDB et lui créera un utilisateur avec tous les droits.
    - Il va ensuite installer et configurer toutes les dépendances de PhpMyAdmin donc apache2 et PHP. Cela implique l'installation d'un serveur web Apache.
    - Il va donc installer et configurer PhpMyAdmin pour le nouvel utilisateur créé dans la base donnée.
    - Finalement, il relance les services actualisés durant le playbook.
- Le playbook FTP : celui-ci installe le serveur ProFTPd et télécharge sa configuration depuis notre GitHub. Ce dernier permet à tout utilisateur authentifié un accès à la racine du système. Enfin il relance le service afin d'appliquer les modifications apportées.
- Le Playbook Wordpress : sa première tâche sera d'appeler les 2 autres playbooks pour préparer l'environnement. Il va ensuite générer une chaîne de caractères aléatoire de 100 caractères sous forme de variable, celle-ci sera notre clé secrète. Puis ce playbook installera Wordpress sur la machine avant de créer sa base de données dans le SGBD installé précédemment. Ensuite le playbook va modifier la configuration par défaut du serveur web Apache afin d'y connecter Wordpress. Pour que Wordpress fonctionne correctement, il activera aussi certains modules Apache avant de redémarrer le service de ce dernier afin de recharger les modifications dans le service. Enfin le playbook va créer le fichier de configuration de Wordpress avec les informations de connexion de la base de données ainsi que la clé secrète générée plus tôt, et ceci avec les droits appropriés à ce fichier. Finalement afin de prévenir toute éventuelles perte de la clé secrète elle est aussi sauvegardée à la racine.

Pour parvenir à rédiger ces 3 playbooks/scénarios, nous avons dû produire plusieurs fichiers annexes qui ne seront pas joints au rapport. Les 3 playbooks présentés sont plus ou moins complexes et peuvent bien évidemment faire l'objet de modifications. C'est d'ailleurs l'un des intérêts et des avantages de cette technologie.

Les playbooks peuvent être améliorés grâce à des dépendances à d'autres fichiers ou à d'autres playbooks. Modifier ces dépendances revient à enrichir les playbooks et à former des scénarios toujours plus complexes. Ces scénarios peuvent aussi bien s'appliquer à des DevOps qu'à des administrateurs système et réseaux. C'est pourquoi nous avons choisis le scénario principal de mise en place d'un Wordpress, construit à partir de plusieurs sous-scénarios indépendants entre eux.

IV- ORGANISATION DU PROJET

1) Organisation générale

A la suite de l'affectation des groupes, nous avons tout d'abord mis en place un environnement de travail adéquat et complet afin d'échanger et de communiquer sur le projet dans de bonnes conditions. L'environnement est principalement composé des éléments suivants :

- GitHub : GitHub a été un outil que nous avons créé assez rapidement. En effet, comme nous l'avons expliqué plusieurs fois précédemment, les playbooks doivent être importés sur GitHub pour pouvoir être exécuté par AWX. Il était donc essentiel de mettre en place un dépôt GitHub.

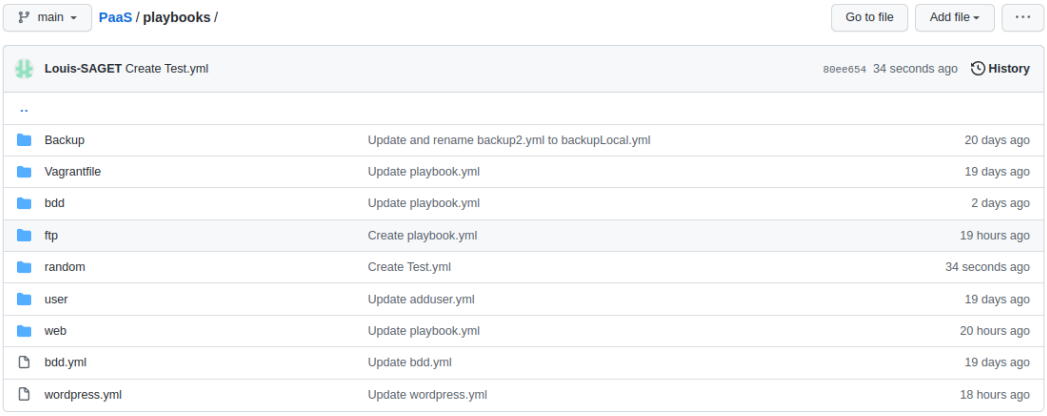

Figure 22 - GitHub

- Google Drive : la création d'un Google Drive nous a permis de stocker nos documents relatifs au projets sur un espace de stockage facilement accessible. En effet, chaque membre du groupe peut y accéder sans aucun risque de sauvegarde, et ce, depuis n'importe quel poste connecté à internet (postes de l'IUT ou postes personnels).

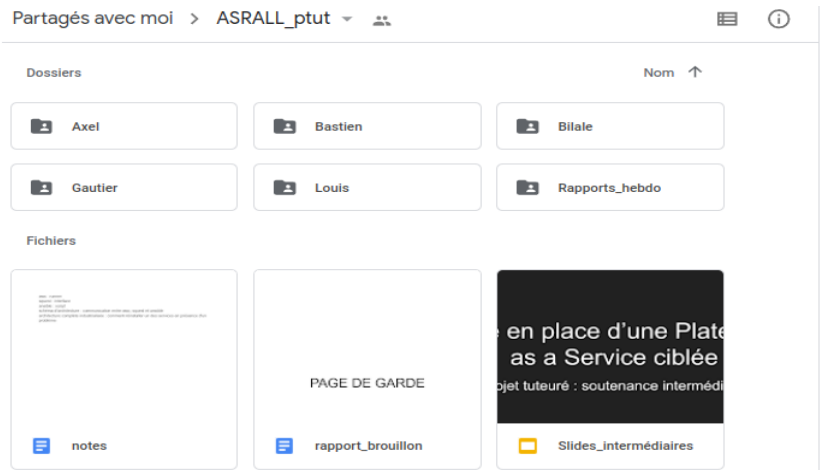

Figure 23 - Google Drive

- Serveur Discord : le serveur nous a permis de communiquer textuellement de façon instantanée et de partager des schémas et d'autres documents rapidement. Le serveur Discord nous a aussi permis de faire des réunions et travailler en gardant un contact vocal les week-ends ou lorsqu'un membre du groupe ne pouvait pas venir travailler à l'IUT.

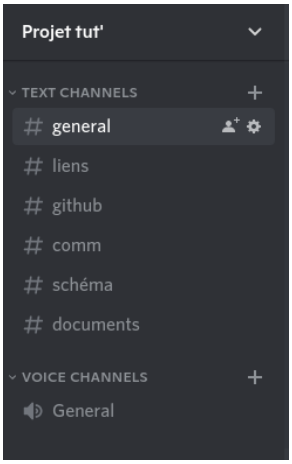

Figure 24 - Serveur Discord

- Trello : le site Trello permet de garder un suivi sur les tâches du groupe et de les organiser plus simplement (à faire, en cours ou terminé). Il est aussi possible d'affecter les membres du groupe à leurs tâches respectives, d'ajouter des stickers ou plus de couleurs pour avoir une meilleure vue d'ensemble du projet.

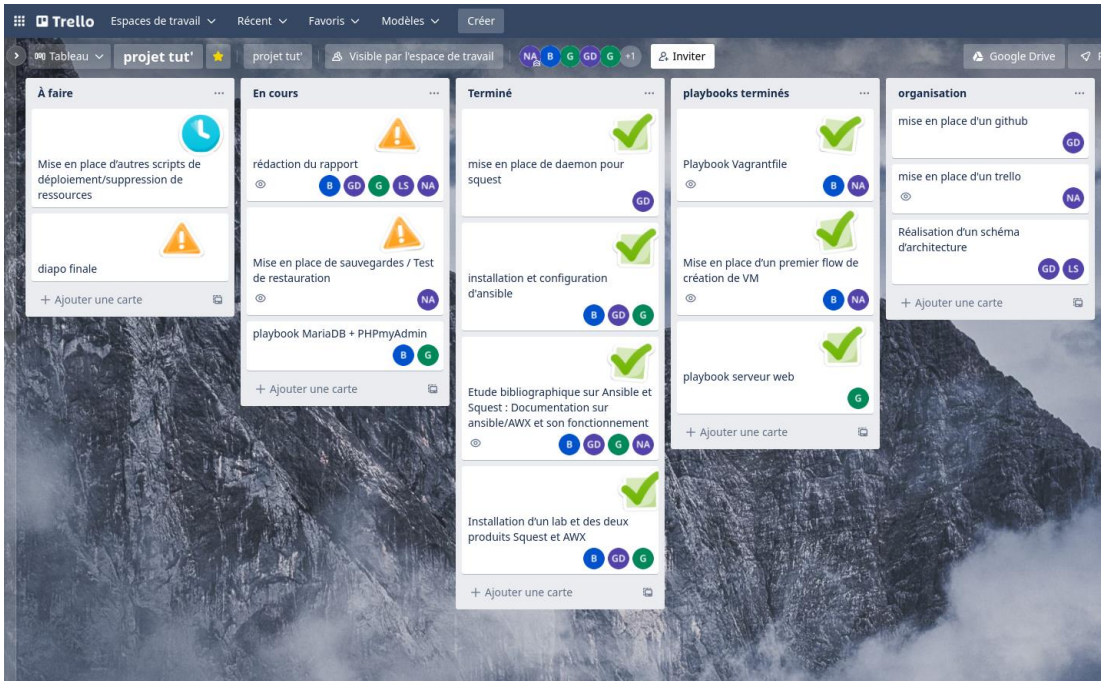

Figure 25 - Trello

Une fois l'environnement mis en place, nous nous sommes concentrés sur la prise en main du sujet et nous avons commencé par définir les premières tâches qui pouvaient composer le projet :

Tâches	Groupe
Documentation sur Ansible et Squest	L'ensemble du groupe
Installation et configuration d'AWX Installation et configuration de Squest	Bilale BOUMADIENE Gautier DELPIERRE Bastien GROSCLAUDE Louis SAGET
Documentation sur les playbooks	Axel NICLAUSSE Louis SAGET
Production de playbooks : o Création de machines virtuelles o Déploiement d'un serveur web o Déploiement d'une base de données	Bilale BOUMADIENE Gautier DELPIERRE Bastien GROSCLAUDE Axel NICLAUSSE

Tableau 2 - Répartition des tâches

Après avoir appréhendé le sujet et après avoir défini certaines tâches essentielles, nous avons répartis les membres de notre groupe sur les différentes tâches. Au cours du projet, nous avons inversés les rôles à plusieurs reprises afin que chaque membre du groupe soit sensiblement au même niveau de compétence sur les différents aspects abordés.

2) Difficultés rencontrées

Au cours du projet, nous avons rencontré quelques problèmes qui ont ralenti la mise en production de la solution. Voici les différents obstacles dans l'ordre chronologique que nous avons dû régler pour que la solution soit fonctionnelle et conforme aux attentes :

- L'appréhension du sujet et la prise en main des outils : prendre en main des outils jamais utilisés auparavant n'est jamais simple. Le manque de support de la part de la communauté et la documentation peu intuitive ont également rendu la prise en main plus longue. La mise en place des outils n'a pas été simple et nous avons été ralenti au début du projet.

- Les différentes versions de Squest : lors de l'exécution des premières tâches, nous avons été confrontés à un obstacle : lorsque nous tentions de nous connecter à l'interface web après l'installation de Squest, nous accédions à une interface non-fonctionnelle. Malheureusement, l'outil Squest reste encore peu exploité par la communauté du libre et sa documentation est encore un peu faible. Nous avons donc dû contacter directement le développeur Nicolas MARCQ, ingénieur de fiabilité des sites chez Hewlett-Packard. La version de Squest n'était pas compatible avec AWX et nous avons été invité à utiliser Squest en mode développement en attendant le déploiement de la nouvelle version. Cette dernière est apparue quelques jours après et résolvait notre problème.

- L'interaction entre AWX et Squest : le principal obstacle pendant notre projet consistait à gérer la configuration et l'interaction entre Squest et AWX. C'est notre tuteur Mathieu GOULIN, qui au cours d'une réunion hebdomadaire, nous as aidé. Nous avons eu des explications sur le fonctionnement général entre les deux solutions et nous avons obtenu une configuration d'exemple pour effectuer un Ping depuis AWX (expliqué précédemment).

- Le playbook RSYNC : l'écriture d'un playbook permettant de faire une sauvegarde de fichiers vers un hôte local ou distant posait un problème, car un certain module (*synchronize*) n'était pas fonctionnel. Bien que la syntaxe du playbook était correcte, le script ne transférait pas les fichiers comme prévu. Après avoir communiqué le problème à notre tuteur, nous avons compris que le problème pouvait provenir d'un bug Python qui était hors de notre portée.

CONCLUSION

Après plusieurs semaines de recherches sur une thématique bien précise, le Portail as a Service est une technologie assez vaste et encore en développement. Néanmoins, cette pratique devient de plus en plus courante et prend forme progressivement au sein des différents organismes.

Une solution de PaaS représente de manière plus générale une évolution d'un ensemble de domaines professionnels, applicables ou non en entreprise. L'intérêt d'une PaaS réside avant tout dans le gain de temps, et ce grâce à une grande flexibilité. Cela apporte un retour sur investissement très rapide même si un grand nombre de playbooks reste encore à définir. La seule mise en place d'une solution PaaS reste rentable.

En suivant nos scénarios présentés précédemment, si un administrateur met en moyenne 1 heure pour créer une machine virtuelle à un DevOps et environ 2 heures à créer le playbook correspondant, après que le DevOps a créé 2 machines virtuelles avec les playbooks, la technologie d'une PaaS offre déjà un gain de temps à l'administrateur. Ce dernier pourra donc travailler sur d'autres projets comme la création d'autres playbooks pour automatiser d'autres mécanismes. Voici donc un des principaux avantages de cette technologie.

L'installation et la configuration d'AWX et de Squest paraissent fastidieuses aux premiers abords. Elles ne sont en réalité pas tellement compliquées. Une fois les manipulations et l'architecture générale comprise, cette technologie est relativement accessible. De plus, si la mise en place d'une solution PaaS peut s'avérer complexe pour certains organismes, un support réactif est proposé à partir du moment où la solution n'est pas gratuite.

La solution que nous avons étudié ces dernières semaines s'inscrit bien dans une optique de PaaS. Cette élaboration plus ou moins complexe des logiciels Ansible, AWX et Squest, forme une plateforme permettant une modification du SI à la demande. Le logiciel AWX joue le rôle central d'orchestrateur dans cette structure et offre de nombreuses interactions. C'est notamment le cas avec l'exécution possible de playbooks depuis le logiciel Squest.

De plus, la création d'une infinité de playbooks représente un service : ils permettent d'automatiser certains mécanismes et d'adapter le SI. Ce service est accessible uniquement depuis une plateforme dédiée et élaborée en amont. Ces critères révèlent l'aspect PaaS de cette technologie.

Après une étude approfondie, nous sommes d'avis que le PaaS est une technologie particulièrement intéressante. Elle est vouée à être de plus en plus utilisée, quel que soit le domaine d'application. Nous avons apprécié de découvrir et d'expérimenter cette vaste et intéressante thématique qu'est la mise en place d'une solution PaaS.

Dans la majeure partie des cas, une PaaS n'est jamais achevée et reste en constante évolution. Il est toujours possible d'améliorer cette technologie. Cela peut passer par l'ajout de playbooks ou par l'ajout de solutions de supervision variées.

Simplifier le travail des administrateurs système et réseaux et automatiser les tâches des utilisateurs représente finalement une difficulté majeure. Cela revient à définir les éléments flexibles ainsi que les éléments stables au sein d'un SI en constante évolution aux technologies variées et changeantes. C'est pourquoi on pourra se demander dans quelles mesures il est pertinent de rendre flexible certains éléments au sein du SI dans un contexte de développement de PaaS.

TABLE DES ANNEXES

TABLE DES ILLUSTRATIONS

ANNEXE 2 – BACKUP.YML

```
# Ce playbook fait une sauvegarde d'un répertoire d'un hôte sur le serveur.
# Le répertoire de sauvegarde et le nom de sauvegarde sont définis par l'utilisateur.
# Pour ce faire il va :
# - Créer une archive en tarball des fichier à sauvegarder avec un nom permettant d'établir des versions.
# - Déplacer cette archive en SCP sur le serveur.
# - Supprimer sur l'hôte local l'archive de sauvegarde
---
- hosts: all
  tasks:
  - name: Create backup archive
    command: tar -czf {{ansible_hostname}}_{{ansible_date_time.epoch}}_{{archive_name}}.tgz {{files_path}}
  - name: Upload backup archive to server
    command: scp {{ansible_hostname}}_{{ansible_date_time.epoch}}_{{archive_name}}.tgz user@192.168.1.146:/home/user/backup
  - name: Delete local backup archive
    file:
      path: "{{ansible_hostname}}_{{ansible_date_time.epoch}}_{{archive_name}}.tgz"
      state: absent
```

ANNEXE 3 – RESTORE.YML

```
# Ce playbook restaure une sauvegarde depuis le serveur.
# Le répertoire de restauration et le nom du fichier sauvegarde sont définis par l'utilisateur.
# Pour ce faire il va :
# - Télécharger depuis le serveur cette archive en SCP.
# - Extraire le tarball de sauvegarde dans le répertoire de restauration.
# - Supprimer sur l'hôte local l'archive de sauvegarde.
---
- hosts: all
  tasks:
  - name: Import backup archive to host
    command: scp user@192.168.1.146:/home/user/backup/{{archive_name}} {{save_path}}
  - name: Extract backup archive
    command: tar -xzf {{save_path}}/{{archive_name}} -C {{save_path}}
  - name: Delete local backup archive
    file:
      path: "{{save_path}}/{{archive_name}}"
      state: absent
```

ANNEXE 4 – VAGRANTFILE.YML

```
# Ce playbook permet de créer un machine virtuelle sur un hôte.
# La configuration de cette VM est définie par l'utilisateur (Répertoire d'installation, nom, nombre de cœur CPU, taille de la RAM, adresse IP).
# Pour ce faire il va :
# - Installer Libvirt et Vagrant si il ne sont pas déjà installés.
# - Créer le répertoire d'installation.
# - Ecrire le Vagrantfile de la machine avec la configuration définie par l'utilisateur.
# - Démarrer la machine.
---
- hosts: all

  tasks:
  - name: Installing libvirt
    become: yes
    apt: name=qemu-kvm update_cache=yes state=latest
    apt: name=libvirt-daemon-system update_cache=yes state=latest
    apt: name=libvirt-clients update_cache=yes state=latest
    apt: name=bridge-utils update_cache=yes state=latest
    apt: name=virtinst update_cache=yes state=latest
    apt: name=virt-manager update_cache=yes state=latest

  - name: Installing Vagrant
    become: yes
    apt: name=vagrant update_cache=yes state=latest

  - name: Creates directory
    file:
      path: "{{path}}"
      state: directory

  - name: Ansible create file with content example
    copy:
      dest: "{{path}}/Vagrantfile"
      content: |
        Vagrant.configure("2") do |config|
          config.vm.box = "debian/buster64"
         config.vm.provider "libvirt" do |vb|
            vb.memory = "{{ram}}"
            vb.cpus = "{{cpu}}"
          end
            config.vm.define "{{ nom_vm }}" do |b|
              b.vm.hostname = "{{ nom_vm }}"
              b.vm.network :private_network, :ip => "{{ip_address}}"
            end
        end

  - name: Run vagrant up
    command: vagrant up
    args:
      chdir: "{{path}}"
```

ANNEXE 5 – MARIADB.YML

```
# Ce playbook va installer un SGBD et PhpMyAdmin sur la machine hôte.
# Pour ce faire il va :
# - Installer mariaDB, PIP & PHP.
# - Active le serveur mariaDB dans la liste des services.
# - Installer et configurer les modules PHP nécessaire à la mise en service de PhpMyAdmin.
# - Installer phpMyAdmin.
---
- name: installation serveur bdd
  hosts: all
  become: yes

  tasks:
    - name: install mariadb
      apt:
        name:
          - mariadb-server
        state: latest
    - name: start mariadb
      service:
        name: mariadb
        enabled: true
        state: started
    - name: Template script
      ansible.builtin.template:
        src: template.j2
        dest: /tmp/script.sh
        mode: '0777'
    - name: requete sql
      command: bash /tmp/script.sh
```

ANNEXE 6 – FTP.YML

```
# Ce playbook installe et configure un serveur FTP sur l'hôte.
# Pour ce faire il va :
# - Installer le serveur FTP ProFTPd.
# - Télécharger et installer la configuration adaptée depuis GitHub.
# - Redémarrer le service afin d'appliquer les modifications.
---
- name: installation serveur FTP
  hosts: all
  become: yes

  tasks:
    - name: install ProFTPd
      apt: name=proftpd

    - name: Configure ProFTPd
      template: src=proftpd.conf dest=/etc/proftpd/proftpd.conf
      notify: relancer ProFTPd

  handlers:
    - name: relancer ProFTPd
      service: name=proftpd state=reloaded
```

ANNEXE 7 – APACHE.YML

```
# Ce playbook installe un serveur web avec PHP sur l'hôte.
# Pour ce faire il va :
# - Installer le serveur web Apache 2 et installer et configurer PHP.
# - Importer et installer le fichier de configuration de l'hôte virtuel apache.
# - Modifier le fichier index par défaut en fichier phpinfo permettant de vérifier le bon
fonctionnement de PHP.
# - Relancer le service apache2 afin d'appliquer les modifications.
---
- name: installation serveur web
  hosts: all
  become: yes

  tasks:
    - name: install apache
      apt: name=apache2
    - name: install php
      apt: name=php
    - name: install lib apache2 mod php
      apt: name=libapache2-mod-php

    - name: virtualhost
      template: src=virtualhost.conf dest=/etc/apache2/sites-available/{{ domain }}.conf

    - name: a2ensite
      command: a2ensite {{ domain }}
      args:
        creates: /etc/apache2/sites-enabled/{{ domain }}.conf
      notify: relancer Apache

    - name: phpinfo
      copy: src=index.php dest=/var/www/html/ mode=0644

    - name: remove default index.html
      file:
        path: /var/www/html/index.html
        state: absent

  handlers:
    - name: relancer Apache
      service: name=apache2 state=reloaded
```