Projet de Recherche et de Développement

Outils de déploiement

Mathieu Dreyer

Année 2019-2020

Projet de deuxième année réalisé dans l'entreprise LMI Solutions

en vue de l'obtention du diplôme d'ingénieur de TELECOM Nancy

Maître de stage : Rémi Santato

Encadrant universitaire : Olivier Airaud

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Dreyer, Mathieu

Élève-ingénieur(e) régulièrement inscrit(e) en 2e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 1215022015D

Année universitaire : 2019-2020

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Outils de déploiement automatisés centralisés

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Nancy, le 26 décembre 2019

Signature :

Projet de Recherche et de Développement

Outils de déploiement

Mathieu Dreyer

Année 2019-2020

Projet de deuxième année réalisé dans l'entreprise LMI Solutions

en vue de l'obtention du diplôme d'ingénieur de TELECOM Nancy

Mathieu Dreyer
20, rue Malvina Cezard
54180, HOUDEMONT
+33 (0)6 14 79 73 07
mathieu.dreyer@telecomnancy.eu

TELECOM Nancy
193 avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 83 68 26 00
contact@telecomnancy.eu

LMI Solutions
3, rue du chapitre
54 670, Millery
+33 (0)3 83 32 14 15

Remerciements

Merci à toutes les personnes qui ont contribué au déroulement de mon projet de recherche et qui m'ont aidé lors de la rédaction de ce rapport.

Je tiens à remercier mon encadrant universitaire, M Olivier Airaud, pour sa disponibilité et ses conseils qui m'ont aidé tout au long de mon projet.

J'adresse mes remerciements à mon maitre d'apprentissage, M Rémi Santato, pour le temps passé ensemble et le partage de son expertise au quotidien.

Enfin, je tiens à remercier toutes les personnes qui m'ont conseillé et relu lors de la rédaction de ce rapport de stage : ma famille, mon ami Pierre Bouillon camarade de promotion.

Table des matières

1 Introduction

Dans le monde de l'informatique, les nouveautés et les évolutions se font plus nombreuses chaque jour. Ceci nécessite que les applications actuelles soient adaptées constamment. Ces méthodes entraînent des complications entre les développeurs et les équipes opérationnelles. On peut alors se demander quels sont les outils permettant de centraliser les déploiements, en rapport avec l'existant à LMI Solutions ?

Dans un premier temps, ce projet portera donc sur la définition d'une méthode, permettant de faire le lien entre plusieurs, voire toutes les phases d'un projet. Ensuite, une étude sera faite sur les différentes catégories, avec les solutions leader de chaque catégorie. Enfin ce rapport traitera de la catégorie qui concerne l'entreprise, avec dans un premier temps les solutions disponibles et dans un second temps un choix d'une solution la plus appropriée et un exemple non-exhaustif des possibilités qu'offre cette solution. Ce choix sera effectué sur les différentes options que fournissent les différentes solutions. Pour LMI Solutions, il est nécessaire que le logiciel choisit soit utilisable sous tout types de systèmes d'exploitations. Il faut également que la solution permette de déployer des mises à jour, des scripts. Enfin, il faut être capable, avec la solution choisie,de récupérer des informations des différents serveurs pour en permettre la supervision.

Actuellement, Saltstack est déjà en place, le choix se basera donc sur les fonctionnalités des autres solutions, ainsi du temps nécessaire au déploiement d'une nouvelle solution.

La solution choisie doit répondre à plusieurs besoins de l'entreprise. Il faut pouvoir automatiser les maintenances préventives, avec la possibilité de récupérer rapidement des données de serveur, comme l'OS, la version et de savoir si le serveur est un serveur virtuel. Il doit également répondre à des besoins différents, tel que le fait de pouvoir changer le mot de passe d'un firewall en quelques secondes.

2 Présentation de l'entreprise

La société LMI Solutions a été fondée par son gérant actuel Monsieur Frédéric DECLE en 2013. C'est une Société Anonyme à Responsabilités Limitées (SARL) spécialisée dans l'informatique. La société compte 12 salariés. Son siège social se situe à Belleville dans la Meurthe-et-Moselle. L'entreprise à trois domaines d'activités, représentée par ces trois femmes, qui sont l'hébergement, l'informatique et la communication.
L'hébergement consiste à mettre à disposition dans un data center régional, une plateforme d'hébergement de haute disponibilité et de haute performance pour sécuriser les applications métiers et sites web.
L'informatique se résume au déploiement de matériel informatique, de la sécurité et du développement.
Enfin la communication est le fait de faire de la promotion des produits des clients, via des flyers, cartes de visite ou encore sur différents réseaux sociaux.
La clientèle ciblée par LMI Solutions sont les administrations publiques, les TPE et PME et les hôtels.

3 Le DevOps

Aujourd'hui pour permettre le déploiement, il existe une multitude d'outils. De base l'équipe développement logiciel code son application, la build et la test. Ensuite, l'équipe d'administration des infrastructures gère le déploiement, les opérations et le monitoring. Le DevOps vient de cette notion, qui est la fusion du mot Dev[8] et du mot Ops[4].
Le DevOps est une pratique adaptée, qui consiste à lier le Développement logiciel (Dev) et l'administration des infrastructures informatiques (Ops).

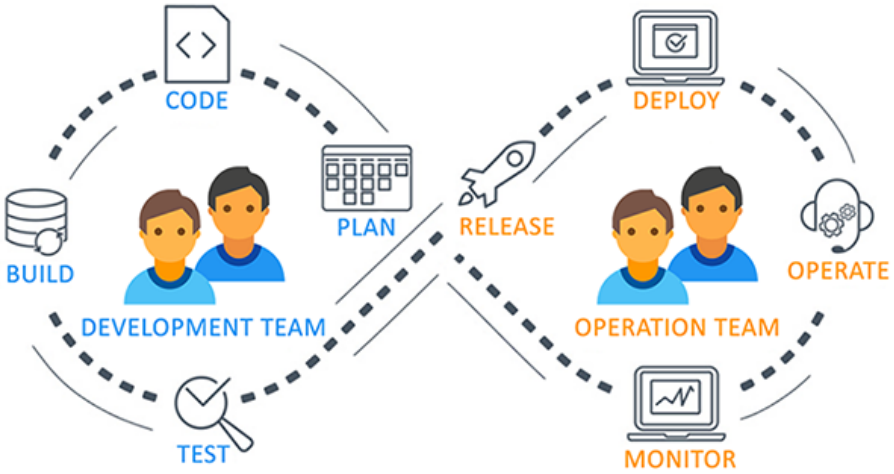

FIGURE 3.1 – Représentation des équipes de développement et des équipes d'administration

Comme le montre la figure 3.1, les différents cycles d'un projet sont répartis entre deux équipes sans DevOps. grâce à lui, les équipes de développement et d'opérations ne sont plus séparées. Il se peut qu'elles soient réunies en une seule et même équipe.
Cette pratique se répand rapidement dans la communauté technique. C'est un concept assez large, dont il existe plusieurs définitions, nuances. Par exemple, il existe le DevSecOps. Cette démarche garde l'approche du DevOps, avec le fait qu'il faut déployer des nouvelles fonctionnalités, de manière rapide. Elle ajoute l'intégration de la sécurité à tous niveaux, ce qui nécessite que les accès aux comptes administrateurs doivent être systématiquement mis à jour, doivent être contrôlés.
La recherche sur le DevOps sera basée sur l'origine du DevOps, et sa première apparition.

3.1 Origine

Le devOps est apparu en 2007 grâce à Patrick Debois, en Belgique. Il voulait améliorer la relation entre l'équipe d'administration des infrastructures et l'équipe de développement.

DevOps est la pratique selon laquelle des ingénieurs du développement et des administrateurs des infrastructures participent ensemble au cycle de vie du produit, de l'application, de la conception au support de production en passant par le processus de développement. Les principes Devops soutiennent des cycles de développement plus courts, une augmentation de la fréquence des déploiements et des livraisons continues pour une meilleure atteinte des objectifs économiques de l'entreprise.

3.2 Utilité

Le DevOps est utile dans divers domaines. Il permet d'améliorer le cycle du produit de la phase de planification de l'application, jusqu'au monitoring de celle-ci. En effet, il agit dans chaque phase, Il permet également d'automatiser des processus qui étaient jusqu'à là répétitifs, lents et manuels. Il permet de fournir un produit plus fiable grâce à différents tests, avec un système de surveillance des journaux et en mettant à dispositions des outils d'intégration continue. Mettre en place cette méthodologie permet d'augmenter la vitesse de l'équipe et de livraison du produit.
De plus, comme le cette méthode permet également de réduire les erreurs humaines dues aux répétitions.

4 Différentes familles de DevOps

Comme le DevOps regroupe toutes les phases d'un projet, il peut se diviser en plusieurs sous-catégories. Il se découpe facilement en cinq familles, qui seront regroupés ici en trois grandes familles, en fonction de la période durant laquelle s'utilise ces différents outils.

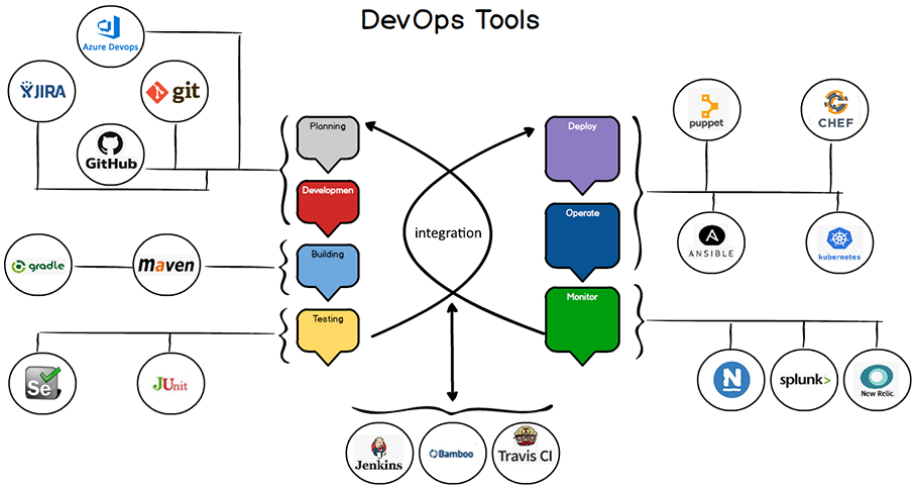

FIGURE 4.1 – Différentes familles de DevOps

Chacun des trois regroupements de catégories sera traité dans ce rapport. Conformément à la figure 4.1, la première famille est celle qui regroupe les phases de planification, développement, génération et test des applications. Cette famille est "intégration et livraison continue". La seconde catégorie est la famille qui s'occupe de la surveillance, elle s'appelle "surveillance journalisation et performances". Enfin la dernière famille regroupe le déploiement et l'opération, qui se nomme "infrastructure as code" Chacune de ces catégories sera analysé dans ce rapport. La catégorie "infrastructure as code" sera traitée en détails, car c'est le type d'outil qui est recherché par l'entreprise.

4.1 Intégration et livraison continue

Le but premier de la catégorie "intégration et livraison continue" est de pouvoir créer, déployer et tester rapidement les logiciels ou applications. Elle permet aussi de réduire le délai de commercialisation et d'augmenter la qualité, car les applications qui sont livrées, le sont de manière continue. Cela permet de réduire les problèmes de déploiement. Les principales solutions dans ce domaine sont Apica [1], GitHub [5] et Delphix [3]. Elle permettent toutes une intégration continue, et une livraison continue.

4.2 Surveillance, journalisation et performances

Le but de cette famille est de permettre aux entreprises de surveiller les journaux pour découvrir l'impact des performances des applications sur les infrastructures pour s'assurer que celle ci ne gênent pas l'expérience de l'utilisateur final du produit. Pour faire cela, il faut capturer, catégoriser les données des journaux puis les analyser pour comprendre l'effet de l'application pour les utilisateurs.
Ceci permet de rendre les services disponibles 24 heures sur 24 et 7 jours sur 7. La création d'alertes et l'analyse en temps réel de ces données aide également les entreprises à surveiller leurs services. Les principales solutions dans ce domaine sont Appdynamics, CA Technologie et ContrastSecurity.

5 Infrastructure As Code

5.1 Définition

L'Infrastructure as Code était initialement un type de configuration, une méthode permettant de gérer les machines virtuelles. Avec l'évolution de l'informatique dans la virtualisation, l'infrastructure as Code à la possibilité de gérer une infrastructure à part entière.
Il existe trois types d'Infrastructure as Code : déclarative (Fonctionnelle), impérative (Impératif) et intelligent (Basé sur l'environnement). Ils sont définis de cette façon :

- L'impératif se base sur le comment l'infrastructure va être changée, l'ordre est important ;
- Le fonctionnel se base sur la configuration attendue, le but étant la configuration finale, l'ordre en lui-même n'a pas d'importance majeure ;
- Basé sur l'environnement : Détermine le bon état avant de faire le nécessaire pour l'atteindre. Les ressources sont déclarées de manière à ce que leur configuration finale et leur état soient en cohérence avec le reste de l'environnement qui les entoure. La création des ressources est automatique.

Il existe deux méthodes d'infrastructure as code, "push" et "pull". Dans la méthode pull, le serveur à configurer va extraire la configuration du serveur principal. Dans l'autre méthode, le serveur principal va envoyer la configuration au système de destination.

5.2 Utilité

Cette pratique permet donc de retirer la partie manuelle dans les processus de gestion d'infrastructure. En d'autres termes, grâce à l'infrastructure as code, il est possible de gérer automatiquement des parcs informatiques en passant pas des scripts.
Cette technologie est donc une réponse aux besoins des entreprises pour l'automatisation et la simplification des infrastructures des projets informatiques.

5.3 Avantages

L'infrastructure as code apporte beaucoup aux entreprises. grâce a cette méthode, il est possible de réagir rapidement à différents problèmes comme par exemple si un problème apparaît sur tous les serveurs, le correctif peut être appliquer de manière plus rapide plutôt que de corriger chaque serveur les un après les autres. Ensuite, il est possible également de maintenir la même version d'une application sur les différents hôtes. Les clients sont donc plus satisfaits.
Elle rend possible la création de meilleurs logiciels et applications avec de la flexibilité, car il y a

moins de temps d'arrêt, ce qui engendre une solution rentable pour l'entreprise.

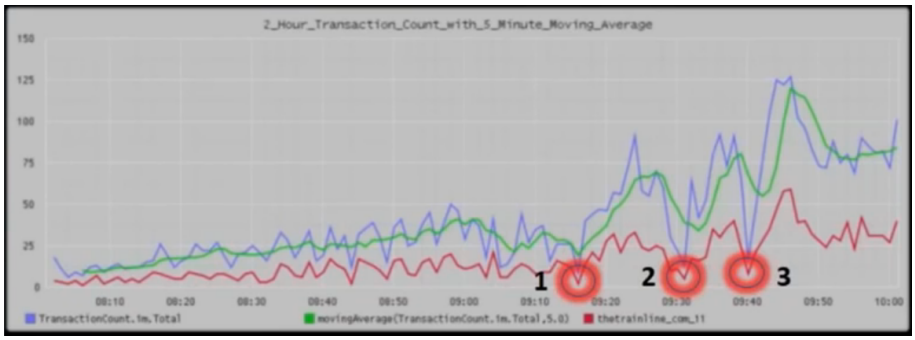

FIGURE 5.1 – Exemple de cas de perte de rentabilité causé par des dysfonctionnements

En effet, avec l'infrastructure as code, les applications qui sont déployées sont testées avant, ce qui évite des moments ou celle ci se retrouve hors ligne. Dans la figure 5.1, il y a trois temps durant lesquelles l'application se retrouve hors ligne, ce qui engendre pour l'entreprise une perte d'argent. Il est vrai qu'un client qui essaye de naviguer sur une application, si celle ci se retrouve brutalement éteinte, ce client risque d'utiliser une autre application.
Elle permet également une harmonisation de parc, c'est à dire d'avoir tout les serveurs avec les mêmes versions d'une application par exemple. De plus, cela réduit les erreurs éventuelles de configuration manuelle, avec une rapidité de réplication d'un environnement template. Un environnement template est le fait de déployer un environnement qui sera déjà prêt à l'utilisation.

5.4 Inconvénients

L'infrastructure as code apporte beaucoup d'avantages, mais cette méthode apporte également quelques inconvénients. Tout d'abord, comme il est possible de planifier des déploiements, l'infrastructure as code peut engendrer une mauvaise planification. En effet, si l'opérateur plannifie un déploiement pendant l'utilisation des serveurs, ceci peut engendrer quelques problèmes.
De plus, l'utilisation de cette méthode nécessite des connaissances supplémentaires, car en cas de mauvaise configuration, toute erreur sera répercutée sur l'ensemble des serveurs. Ensuite si la configuration contient une erreur, celle ci sera répliquée sur tous les hôtes. Enfin le risque principal est, comme les outils possèdent plusieurs capacités telles que celles de détruire des ressources de manière automatique, que l'outil détruise des données de façon accidentelle.

6 Etude de l'existant

L'étude des solutions existantes portera sur les quatre meilleures solutions actuelles sur le marché, qui sont Terraform, Ansible, Chef et Saltstack.

Une cinquième grande solution existe, qui est Pulumi [10], mais celle-ci sort du domaine d'action souhaité par l'entreprise, elle ne sera donc pas étudiée. Elle sert à gérer du *cloud computing*. Le cloud computing consiste à l'utilisation de serveurs informatiques distants, via le réseau, pour stocker des données, les exploiter.

6.1 Terraform

6.1.1 Détails de l'outils

Terraform est un outil open source DevOps édité par HashiCorp. Il est dans la catégorie infrastructure en tant que code. Il existe une version commerciale pour les entreprises, les prix ne sont pas communiqués. L'étude de cette solution portera sur la version gratuite.
Cet outils est écrit en Go. Son but principal est de permettre la création et la définition d'infrastructure. Terraform autorise également le versioning
C'est un outils de type fonctionnel. Il fonctionne avec la méthode push.

Sur la page d'introduction du projet, Terraform se compare avec d'autres solutions du marché. C'est une solution basée sur l'infrastructure. En d'autres termes, Terraform permet uniquement le déploiement et la modification d'infrastructures.
La solution n'assume pas les mêmes fonctionnalités que certaines solutions de l'infrastructure as code, comme les solutions qui seront traitées ultérieurement dans ce rapport. Actuellement, l'utilisation de scripts ou d'outils tiers est encore nécessaire pour que la solution réponde en tant qu'infrastructure as code.
Les fichiers de configuration s'écrivent en HCL[6] (HashiCorp Configuration Language). C'est un langage de configuration construit par HashiCorp. L'objectif de HCL est de créer un langage de configuration structuré qui soit à la fois convivial pour l'homme et la machine pour une utilisation avec des outils de ligne de commande, mais spécifiquement ciblé vers les outils DevOps, les serveurs. Il n'y a pas d'agent, il faut au préalable configurer les connexions qui se font en SSH ou avec un fichier d'installation sous windows.

```
# Copies the file as the root user using SSH
provisioner "file" {
  source      = "conf/myapp.conf"
  destination = "/etc/myapp.conf"

```

```
  connection {
    type     = "ssh"
    user     = "root"
    password = "${var.root_password}"
    host     = "${var.host}"
  }
}
```

Listing 6.1 – Exemple d'un code permettant la connexion

Voici un exemple de définition d'hôtes, avec le type de connexion. le fichier de configuration possède l'extension .tf, et les champs sont définis comme dans l'exemple précedent.

6.1.2 Capacités de l'outils

Les principales fonctionnalités de l'outil sont :

- Terraform propose un outil de prise en charge de services cloud (par exemple Azure[2]) ;
- Possibilité de détruire des ressources ;
- Prévisualisation des modifications avant de les appliquer ;
- Création d'infrastructure ;
- Possibilité d'appliquer des modifications, de façon incrémentales ;

6.1.3 Avantages

Terraform conserve l'état précédent de tout fichier, ce qui est donc utile lors d'ajout ou de suppression de données. Un autre avantage de Terraform est le fait que sa structure se base sur des plugins, ce qui permet au différentes personnes qui utiliseront Terraform d'étendre les fonctionnalités du logiciel à leur guise.
Terraform fonctionne sans agent. Il utilise un Secure Shell (SSH), qui est un protocole de connexion. Il impose un échange de clés de chiffrement en début de connexion. Par la suite, tous les segments TCP sont authentifiés et chiffrés. Il est donc impossible de voir ce que fait l'utilisateur.
Enfin, pour les personnes visuelles, Cet outil offre la possibilité de générer des graphiques de dépendances.

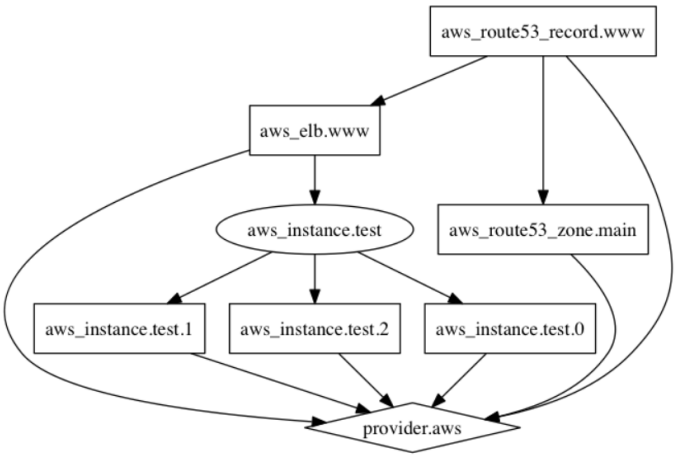

FIGURE 6.1 – Exemple de graphique de dépendance

La figure 6.1 montre un exemple du retour de la commande "Terraform graph" qui permet d'afficher les dépendances sous cette forme.

6.1.4 Inconvénients

C'est un outil particulier, qui demande un temps d'adaptation avec le langage, et l'utilisation. Si une erreur à été effectuée dans des modifications, il n'y a pas de possibilité de revenir en arrière.

6.1.5 Exemples

Terraform peut donc être utilisé pour faire des déploiements, sans récupérer d'informations des serveurs. Il est principalement utilisé pour gérer des clouds. Des exemples d'utilisations sont disponibles ici

6.2 Ansible

6.2.1 Détails de l'outils

Ansible est une plateforme open source écrite en python. Il existe Ansible Engine et Ansible Red Hat. La différence entre les 2 solutions et que Ansible Engine est gratuit, tandis que Ansible Red Hat est une solution commerciale, pour les entreprises.
La recherche portera sur la solution open-source.
C'est un outils fonctionnel et impératif. Il utilise la méthode push.

Ansible offre la possibilité d'automatiser l'informatique. Il permet également de gérer les configurations et de faire des déploiements de manière automatique.
Ansible utilise le langage YAML[12] pour ses configurations. Ansible est une liste de module, donc il est possible d'écrire ses propres modules. Un module Ansible est un script autonome réutilisable qui peut être utilisés par l'API Ansible ou par les programmes Ansible. Il renvoi des informations à Ansible. Il faut ensuite définir des playbooks (scénarios en français). Un playbook est une liste de taches définies en YAML. Si une échoue, il est possible de relancer le scénario à partir de la tache à laquelle il s'est arrêté.
Il existe ensuite Ansible Galaxy, qui est un endroit pour partager les modules, qui permet d'effectuer diverses opérations liées au rôle et à la collection de module.
Enfin cet outil met à disposition un moyen de protéger les informations sensibles, tel que des mots de passes, avec Ansible Vault, qui permet d'encoder et de décoder des informations privée, tel que des mots de passe.

```yaml
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: latest
  - name: write the apache config file
    template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf
```

Listing 6.2 – Exemple d'un playbook [9] Ansible

6.2.2 Capacités de l'outil

Les principales fonctionnalités de Ansible sont :

- La possibilité de définir facilement des hôtes qu'il faut superviser et de les supprimer facilement.

— Le serveur principal connaît toutes les IP des autres serveurs, grâce aux informations fournies.
— Le fait de pouvoir définir des variables, qui auront des valeurs par défaut pour tous les hôtes.
— Il est possible de lancer des commandes en vérifiant le résultat d'une autre commande.
— Ansible offre la possibilité de déployer des applications.

6.2.3 Avantages

Ansible fonctionne sans agent, car cette solution passe directement par le SSH. Pour pouvoir accèder au serveur, la connexion se fait via SSH. Un agent lui nécessite l'installation sur les serveurs Elle fonctionne sous n'importe quelle distribution Unix.
Le langage utilisé pour la configuration est un langage simple d'utilisation, et facilement lisible par l'homme. C'est un langage assez facile à apprendre et orienté administrateur, connu et commun. Par conséquent, la configuration et l'utilisation de l'outil sont plus rapides car il est facilement utilisable.

6.2.4 Inconvénients

Comme c'est une solution sans agent, si l'entreprise souhaite superviser des machines sous Windows, il faut au préalable que ces dernières disposent d'un serveur SSH
Le serveur principal doit obligatoirement être installé sous une machine Unix/Linux
Une fois qu'un playbook est en cours d'exécution, il n'est pas possible d' ajouter ou supprimer des hôtes d'un fichier d'inventaire, contrairement à Terraform. Cela vaut également pour les variables, c'est-à-dire que vous ne pouvez pas ajouter de variables au fichier de variables global au moment de l'exécution.

6.2.5 Exemples

Ansible est principalement utilisé pour la configuration, le déploiement d'applications et la livraison continue. Une liste d'exemples d'utilisation est disponible sur ce dépôt

6.3 Chef

6.3.1 Détails de l'outils

Chef est un logiciel open source, écrit en Ruby. C'est un logiciel qui a pour but d'aider à la configuration de système d'exploitation. Une version commercial existe, qui est gratuite jusqu'à 5 serveurs, puis propose un abonnement à hauteur de 6$ le serveur. L'analyse portera uniquement sur la version gratuite, open source.
C'est un outils avec une approche fonctionnelle et impérative. Il utilise la méthode pull.

Le fonctionnement repose sur trois composants, le Chef server, la ou les plusieurs workstation(s) et la/les node(s) définies ci-dessous.

- Un Chef server ou serveur Chef a pour but en tant que centre des opérations est de stocker, gérer et fournir des données de configuration à tous les autres composants Chef.
- Les workstations (ou station de travail) sont des ordinateurs ou des serveurs virtuels sur lesquels tout le code de configuration est créé, testé et modifié. Il peut y avoir autant de postes de travail que nécessaire.
- Les nodes (ou noeuds) sont les serveurs qui sont gérés par Chef. Il peut gérer des serveurs virtuels, des conteneurs, des périphériques réseau et des périphériques de stockage.

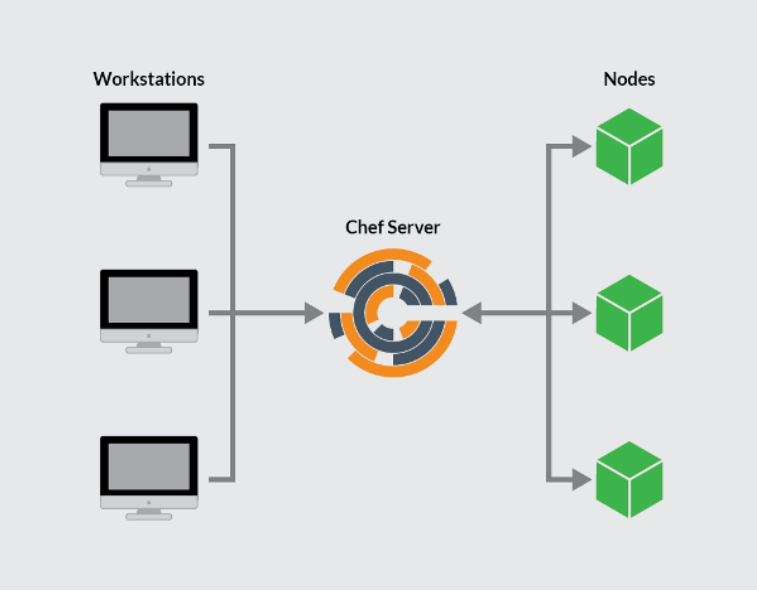

Figure 6.2 – Fonctionnement Chef

La figure 6.2 schématise le fonctionnement de Chef. Pour que Chef puisse gérer des nodes, il faut que l'agent Chef soit installé sur les nodes.
Ces trois composants communiquent de manière essentiellement linéaire, les modifications apportées étant push des postes de travail au serveur Chef, puis transférées du serveur vers les nœuds et mises en œuvre sur chaque nœud via leur client Chef.

Pour décrire la manière dont Chef doit gérer les applications et les configurer, l'utilisateur écrit des configurations, qui sont appelées sous Chef des "recettes". Ces recettes (qui peuvent être regroupées dans un "livre de recettes" pour une gestion simplifiée) décrivent une série de ressources qui devraient se trouver dans un état particulier : packages à installer, services à exécuter ou fichiers à écrire. Ces diverses ressources peuvent être configurées pour des versions spécifiques du logiciel à exécuter et peuvent garantir que le logiciel est installé dans le bon ordre en fonction des

dépendances. Chef s'assure que chaque ressource est correctement configurée et corrige toutes les ressources qui ne sont pas dans l'état souhaité.

```
...

vim_base_pkgs = value_for_platform_family(
  %w(debian arch) => ['vim'],
  %w(rhel fedora) => ['vim-minimal', 'vim-enhanced'],
  'default' => ['vim']
)

package vim_base_pkgs

package node['vim']['extra_packages'] unless node['vim']['extra_packages'].
    empty?
```

Listing 6.3 – Exemple d'une recette

L'exemple de cette recette fait partie du livre de recettes Chef sur Vim[11]. Il est responsable de l'installation du package Vim requis basé sur la distribution Linux d'un nœud.

6.3.2 Capacités de l'outils

Les principales capacités de l'outil sont :

- Chef s'assure que les configurations sont appliquées de manière uniforme dans tous les environnements, à toutes les échelles, avec l'automatisation de l'infrastructure
- Chef permet l'automatisation de serveur cloud
- Chef assure une conformité et une gestion de la sécurité

6.3.3 Avantages

Cette solution est pratique, car elle apporte un grand nombre d'avantages. Dans un premier temps, il existe une grande liste de recette disponible, ce qui permet de se servir de l'existant.
Le client Chef va vérifier qu'aucun changement n'est survenu et que rien ne doit changer, de manière ponctuelle. Si un changement à eu lieu, il s'applique.
Les correctifs et les mises à jour peuvent être déployés sur l'ensemble des infrastructures en modifiant la recette.
De plus, il n'y a pas la nécessité d'interagir avec chaque machine individuellement.

6.3.4 Inconvénients

Malgré cela, Chef possède quelques points faibles. En effet, comme les fichiers de configuration sont en Ruby, il nécessite un temps d'adaptation pour s'adapter à Ruby, pour ceux qui découvre le langage. De plus le serveur principal n'a pas beaucoup de contrôle sur le reste, car son rôle principal est de déposer la configuration des stations de travails sur les nodes.
En plus de cela, la première configuration est assez complexe, car la solution est assez complexe à mettre en place. Enfin le processus de pull est défini selon un planning, donc il ne s'exécute pas à chaque changement.

6.3.5 Exemples

Chef est principalement utilisé pour l'administration de serveur.
Une liste des différentes commandes disponibles, et d'exemples d'utilisation de Chef sont disponible sur le dépôt Chef.

6.4 Saltstack

6.4.1 Détails de l'outils

Saltstack (ou Salt) est un framework d'automatisation très puissant. Un framework est une abstraction dans laquelle un logiciel fournissant des fonctionnalités génériques peut être modifié de manière sélective par un code supplémentaire écrit par l'utilisateur, fournissant ainsi un logiciel spécifique à l'application. C'est une solution open source. L'architecture SaltStack est basée sur le concepte d d'exécuter des commandes à distance. Tous les réseaux sont conçus autour d'un aspect de l'exécution à distance. SaltStack fonctionne avec un agent.

SaltStack fonctionne avec différents composants, les salt reactors, minions, grains et pillars. Ils peuvent être décrits de cette façon :

- Les salt reactors écoutent les événements, pendant que les agents utilisent SSH pour exécuter des commandes sur un système cible. ils donnent à SaltStack la capacité de déclencher des actions en réponse à un événement.
- Le minion peut éventuellement être installé sur la cible pour envoyer des commandes en Python. Il s'agit de l'agent Salt.
- Les grains fournissent des informations sur le système cible aux minions. Il peut fournir tout types d'informations, tel que le système d'exploitation, la version, l'adresse IP.
- Les pillars correspondent aux fichiers de configuration. Ils stockent les données, les informations confidentielles.

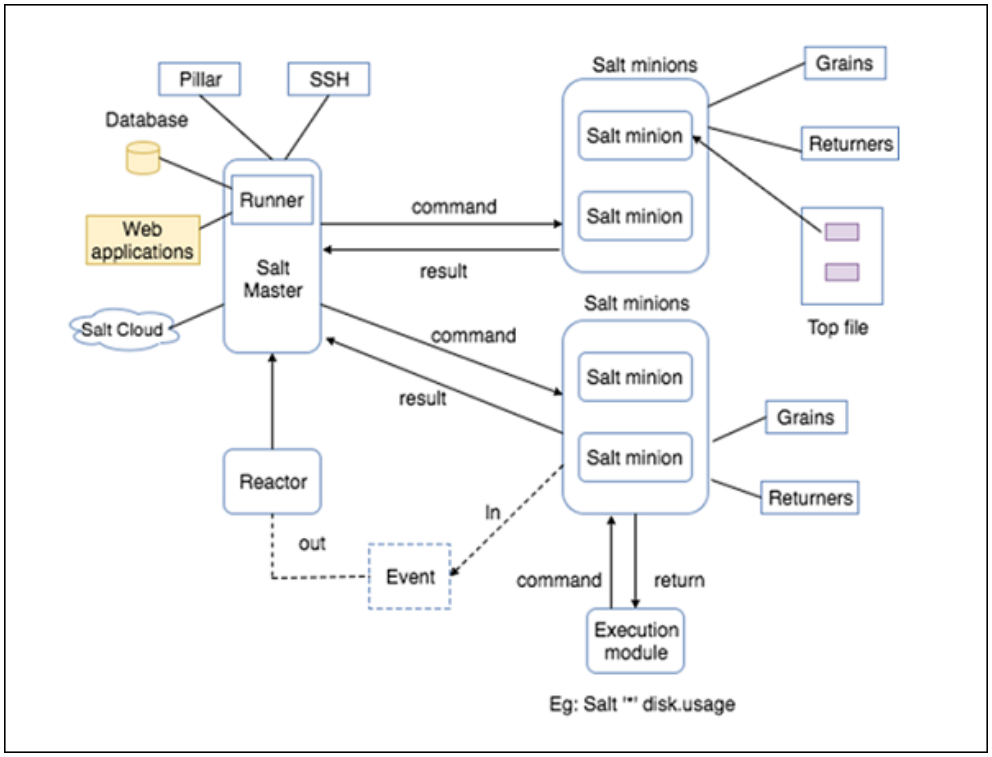

FIGURE 6.3 – Architecture SaltStack

SaltStack utilise les templates Jinja2 [7] pour insérer des instructions conditionnelles et réaliser d'autres configurations dans les fichiers Salt et pillar, entre autres.
Pour que SaltStack puisse communiquer avec les serveurs, il faut configurer un identifiant sur les serveurs, puis accepter la clé sur le serveur principal.
Ensuite, il suffit simplement d'appeler l'identifiant du serveur, et l'option de la commande.

```
salt '*' test.ping
```

Listing 6.4 – Affectation d'une tache à un serveur

Par exemple dans cette commande, le parametre "*" permet de cibler tous les serveurs, et test.ping est l'option qui sera exécutée sur les serveurs.

6.4.2 Capacités de l'outils

C'est un outil qui est à la fois simple, par exemple en demandant à un serveur Web distant d'afficher une page Web statique, et à la fois complexe avec par exemple l'utilisation d'une session shell pour émettre de manière interactive des commandes sur un serveur distant.

Lorsque SaltStack se connecte pour la première fois à un système, un script s'exécute et vérifie le système d'exploitation et la version cibles, puis installe les fichiers spécifiques à la configuration. L'outil exécute des modules, à la fois prédéfinis et personnalisés, à distance. Il permet l'automatisation pour l'Ops, et l'intégration continue d'un code et le déploiement de ce code. Il permet également la surveillance d'application.

6.4.3 Avantages

SaltStack est écrit et utilisé en Python. La plus part des systèmes Linux ont déjà Python d'installé. Il fonctionne sous n'importe quel OS, tant que l'agent est installé. SaltStack est un outil qui évolue beaucoup avec une grande communauté, il y a donc un support actif, avec une communauté dynamique. Une fois l'installation effectuée, il est facile de s'en servir directement car la syntaxe YAML est cohérente et riche en fonctionnalités et Python offre une faible courbe d'apprentissage pour les développeurs.

6.4.4 Inconvénients

Cependant, SaltStack engendre quelques inconvénients. Tout d'abord, l'interface graphique de cette solution n'est pas riche en fonctionnalités. Les utilisateurs effectuent des tâches via ligne de commande par conséquent.
Ensuite, les états cibles ne peuvent pas être vérifiés dans un ordre spécifique. Cette limitation diminue la possibilité de programmer des dépendances entre les systèmes. De plus, La plateforme est nouvelle et n'est pas entièrement mature par rapport à d'autres solutions telle que Chef.

6.4.5 Exemple

Des exemples d'utilisation de SaltStack sont disponibles sur ce site.

6.5 Choix de la solution

6.5.1 Comparatif des solutions

Le choix se basera donc sur les choix qu'offrent les différentes solutions, en comparaison avec les pré-requis principaux pour l'entreprise.

	Terraform	Chef	Ansible	Saltstack
Architecture	Client	Serveur/Client	Serveur	Serveur/Client
Facilité à mettre en place	Modéré	Modéré	Simple	Modéré
Interface graphique	non	non	oui	oui (peu renseignée)
Langage	Go	C++ / Ruby	Python	Python
Langage de configuration	HCL	Ruby	YAML	YAML
Gestion	difficile	difficile	simple	simple
Nombre de Commits	25 000	12 000	1 000	49 000
Agent ou SSH	Agent	Agent	SSH	Agent et SSH

Table 6.1 – Tableau comparatif des éléments principaux des solutions

Parmi ces solutions, certaines se différencies par leur niveau de difficulté à mettre en place, ou encore le langage.
Le nombre de Commit ici est utile, car il est révélateur de la communauté active autour des différentes solutions, si les commits sont récents.

6.5.2 Justification du choix

Chaque outil possède ses propres avantages et inconvénients en fonction de l'environnement d'utilisation. Ces outils sont tous très utile dans leur domaine, cela dépend uniquement de l'utilisation, des capacités de l'entreprise, et du type d'environnement qui va être géré.
Tous ces éléments permettent de déterminer quel est l'outil le plus approprié. De plus, chaques outils disposent de tutoriels disponibles sur internet et de vidéos qui permettent de guider l'installation, et l'utilisation. Tous ces éléments permettent d'apprendre rapidement comment utiliser l'outil. Il faut donc se baser sur les capacités des outils, et celle de l'équipe qui sera amenée à s'en servir.
Parmi les choix, il faut une solution qui soit utilisable sur n'importe quel type de système d'exploitation. En se réferant à l'étude effectuée, il n'y a qu'une solution qui pose problème, il s'agit d'Ansible qui n'utilise que le port SSH. Cela est problématique car il n'y a pas de serveur SSH installé sur les serveurs Windows. Il est plus pratique d'installer l'agent, car l'installation d'agent est rapide, et permet de récupérer plus d'information. La solutions Ansible peut donc être écartée.
Un second choix sera basée sur l'architecture, il faut une architecture qui propose un fonctionnement de serveur client, avec le serveur qui a la possibilité d'exécuter des scripts sur les clients, et que le serveur puisse aussi récupérer des données depuis les clients. Cela permet d'écarter la solution Terraform.
Le choix se basera donc sur SaltStack et Chef. Chef possède un gros inconvénient, qui est que le serveur principal n'a pas beaucoup de contrôle, comme il ne fait que de déposer des configurations.

6.6 Utilité pour l'entreprise

Chez LMI Solutions, plusieurs taches sont effectuées chez divers clients, tout en répétant les memes opérations. L'une des premières taches est la maintenance préventive.
L'objectif de celle-ci est d'assurer une maintenance sur l'infrastructure serveurs en production. Elle permet de vérifier le bon fonctionnement physique des machines, des switchs, de Windows et de ses services associés et d'effectuer les mises à jour nécessaires. Parmi les vérifications effectuées, certaines sont répétitives :

- contrôle du fonctionnement des services Windows utilisés.
- Mise à jour Windows.
- OS mis en place
- Vérifications licenses Trend Micro, Arcserve.

Il faut donc voir s'il est possible d'automatiser ces vérifications, avec SaltStack.

6.7 Installation de l'agent SaltStack

Pour que SaltStack fonctionne, il faut au préalable installer l'agent Salt sur les serveurs. Il est facile d'installation et se télécharge via cette URL.
Une fois l'installation effectuée, il faut ensuite configurer le fichier minion, de manière à renseigner le nom du master qui est ici "salt.lmgroupe.fr"

```
# Set the location of the salt master server. If the master server cannot be
# resolved, then the minion will fail to start.
#master: salt
```

FIGURE 6.4 – Fichier de configuration Salt partie master

```
# Set the location of the salt master server. If the master server cannot be
# resolved, then the minion will fail to start.
master: salt.lmgroupe.fr
```

FIGURE 6.5 – Saisie du Master Salt

Il faut ensuite décommenter la ligne avec l'id, et en fournir un. Pour respecter une nomenclature fixe, un id est défini de cette manière pour l'entreprise : "Nom_du _client"."Nom_du_seveur"

```
# Explicitly declare the id for this minion to use, if left commented the id
# will be the hostname as returned by the python call: socket.getfqdn()
# Since salt uses detached ids it is possible to run multiple minions on the
# same machine but with different ids, this can be useful for salt compute
# clusters.
#id:
```

FIGURE 6.6 – Fichier de configuration Salt partie ID

```
# Explicitly declare the id for this minion to use, if left commented the id
# will be the hostname as returned by the python call: socket.getfqdn()
# Since salt uses detached ids it is possible to run multiple minions on the
# same machine but with different ids, this can be useful for salt compute
# clusters.
id: foret.zabbix
```

FIGURE 6.7 – Saisie de l'ID

Il faut ensuite accepter les clés de chaque serveur. Une fois ces étapes effectuées, l'installation est terminée.

6.8 Configuration et utilisation

Actuellement, pour les maintenances préventives, le technicien est obligé de prendre la main sur chaque serveur, à distance ou sur place, pour vérifier différentes caractéristiques. Il faut donc vérifier que la solution permette de simplifier ce processus.

6.8.1 Utilisation dans les maintenances préventives

Avec SaltStack, il existe une multitude de commande de base, permettant de récupérer en quelques secondes les informations nécessaire dans une maintenance préventive.

Il est possible de vérifier si le serveur est physique ou virtuel avec une simple commande, pour les serveurs Windows :

```
salt -C '* and G@os:Windows' grains.item 'virtual'
```

Listing 6.5 – Commande pour connaitre l'état du serveur

Tous les serveurs Linux sont des serveurs virtuels, donc il n'y a pas de vérifications à effectuer

Si le serveur retourne une valeur de modèle tel que : VMware Virtual Platform, alors le serveur est virtuel, sinon il est physique. En effet, car les machines virtuelles sont toutes gérées par VMware. De meme pour l'OS, il suffit d'effectuer cette commande sous Windows :

```
salt -C '* and G@os:Windows' grains.item 'os'
```

Listing 6.6 – Commande pour connaitre l'OS

```
salt -C '* and G@os:Windows' grains.item 'osfullname'
```

Listing 6.7 – Commande pour connaitre la version de l'OS sous Windows

```
salt -C '* and G@kernel:Linux' grains.item 'osfullname'
```

Listing 6.8 – Commande pour connaitre l'OS sous linux

Cette utilisation répond donc aux besoins de l'entreprise sur les maintenances préventives.

Cependant il n'est pas possible de tout faire uniquement via des commandes. Salt offre la possibilité de définir ses propres scripts, avec diverses fonctionnalités. De plus il est possible d'utiliser des templates Jinja pour ses scripts, ce qui permet de récupérer des données depuis les pillars de SaltStack.

6.8.2 Utilisation pour changer le mot de passe de firewall

Actuellement, chaque firewall installé chez un client dispose d'un mot de passe. Il se peut que les mots de passe entre ces différents firewall changent, en fonction des demandes. Par exemple si le client à besoin du mot de passe, un mot de passe différents de celui défini par l'entreprise sera attribué. Le problème principal de ce type de fonction est que si le dossier dans le quel le mot de passe est renseigné n'est pas à jour, il n'est pas possible de retrouver le mot de passe. Pour récupérer le mot de passe il faut se connecter en port série sur ce dernier, et réinitialiser le mot de passe. Cela peut être très problématique car il faut être à proximité du firewall pour restaurer le mot de passe par défaut. Pour pallier à ce problème SaltStack va permettre de changer les mots de passe facilement via un script.

En effet, pour résoudre ce problème, un script a été créé en partant du script de ré-initialisation. Il suffit ensuite de changer le mot de passe par défaut par un champ saisie dans le dossier. Si celui-ci n'est pas renseigné, un mot de passe par défaut défini par l'entreprise et non par le firewall sera attribué.

```
1
2
3  # 1 : deposer fichier php
4  # 2 : droit fichier php
5  pfsense-password:
6    file.managed:
7      - name: /root/change_password
8      - source: salt://pfsense/bin/change_password
9      - mode: 755
10     - template: jinja
11 # 3 : recuperer pass pillar
12 # 4 : si existe, pass pillar
13 # 5 : si existe pas, pass defaut dur
14 # 6 : call php avec pass arg
15 change_pass:
16   cmd.run:
17     - name: '/root/change_password {{ salt['pillar.get']('pfsense-pass', '
       motdepasse') }}'
```

Listing 6.9 – script pour changer le mot de passe

Dans ce script, deux taches sont effectuées. La première consiste à copier un fichier sur le serveur, en lui affectant des droits. Ensuite, on exécute ce fichier, avec en paramètre un mot de passe, qui est soit défini dans les pillars, sinon qui est fixé à une valeur par défaut.
Il est donc possible de récupérer la valeur d'une variable avec les templates Jinja. Cette ligne " salt['pillar.get']('pfsense-pass', '**motdepasse') " signifie que l'on cherche dans les pillars le champ pfsense-pass. Si celui ci est absent, ou vide le mot de passe "**motdepasse" sera affecté par défaut.

Pour exécuter ce script, il faut effectuer cette commande :

```
1 salt 'nom_de_la_machine' state.apply pfsense.password
```

Listing 6.10 – Commande pour changer le mot de passe

Il est possible facilement de voir le résultat des différentes tâches, grâce au retour de la commande.

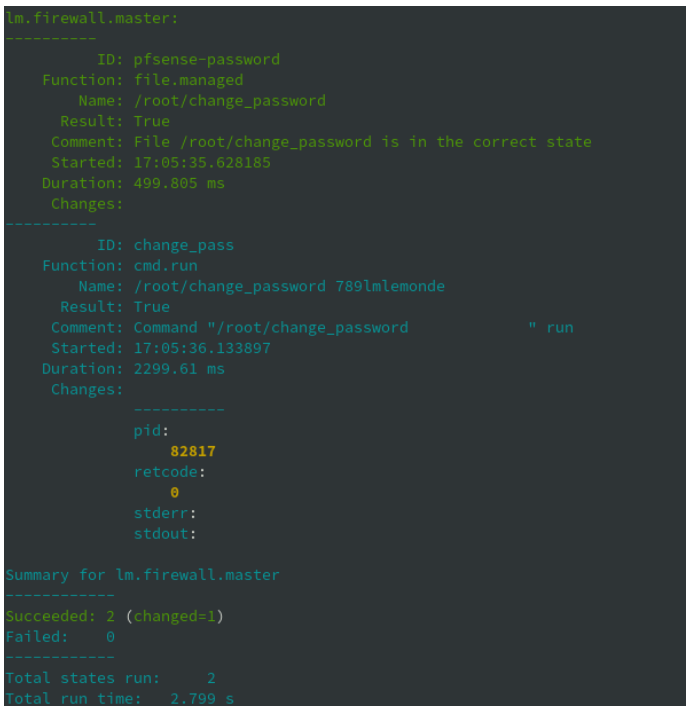

FIGURE 6.8 – Résultat d'une commande

Les états en vert dans la figure 6.8 correspondent aux états inchangés. Ici le fichier pfsense-password est déjà présent sur la machine. Enfin, les états en bleu correspondent aux états qui ont subit un changement, mais dont le changement a été effectué avec succès. Les état en noir sont ceux qui sont échoués.

Ces exemples et la définition de SaltStack, montrent que cet outil est clairement l'outil le mieux adapté aux besoins de l'entreprise. En effet, il est possible de récupérer via une commande beaucoup d'informations sur le serveur. Ensuite, si les commandes fournies ne permettent pas de récupérer les informations voulus, ou ne permettent pas d'effectuer une opération, il est possible de définir sa propre commande, qui effectuera la suite d'opération voulu, avec un état de chacune des différentes opérations du script.

7 Conclusion

Ce rapport avait pour ambition d'effectuer une étude sur le DevOps, en étudiant les différentes catégories existantes, en les décrivant. Les besoins de l'entreprise étant définie, il fallait trouver soit une solution plus adaptée à celle existante, soit justifier le choix de l'existant, et décrire les possibilités fournies de celle ci. L'entreprise possédait SaltStack comme réponse aux besoins.

Il a fallu dans un premier temps définir la notion de DevOps et examiner les familles, et choisir la plus adaptée aux besoins de l'entreprise.

Il convenait alors de s'intéresser à la catégorie *Infrastructure as Code*. Une étude a été adoptée dans cette étape, sur les différents logiciels d'Infrastructure as Code disponible. L'étude a consisté à rechercher sans vraiment sélectionner, les meilleures solutions dans cette catégorie. Cette analyse a laissé des solutions avec des types de fonctionnements différents et a donnée de la matière au choix de la solution la plus adaptée.

Ensuite, un comparatif à été effectué sur les différents logiciels, en mettant en avant les différences et les avantages de chacun. Suite à ce comparatif, la solution la plus adaptée parmi celles retenues est SaltStack, car elle est assez simple à mettre en place, possède un langage simple pour définir les scripts, et permet de faire des tâches simples, mais aussi des tâches complexes.

Enfin, ce rapport à donc ensuite décrit de manière non-exhaustive les possibilités de configurations, de supervision, et de définition de script possible avec cette solution tout en répondant aux besoins de l'entreprise.

Bibliographie / Webographie

[1] Apica. Site officiel de la solution apica. https://www.apicasystems.com/apica-connectors/. 5

[2] Azure. Site officiel de azure. https://azure.microsoft.com/en-us/. 10

[3] Delphix. Site officiel de delphix. https://www.delphix.com/. 5

[4] Administration des infrastructures informatiques. Définition du terme ops. https://en.wikipedia.org/wiki/Data_center_management#Operations. 3

[5] Github. site officiel de github. https://github.com/. 5

[6] Langage HCL. dépot github du langage hcl. https://github.com/hashicorp/hcl. 9

[7] Jinja2. Site web des templates jinja2. https://jinja.palletsprojects.com/en/2.10.x/. 17

[8] Developpement logiciel. Définition du terme dev. https://en.wikipedia.org/wiki/Software_development. 3

[9] Playbook. Exemple d'un playbook sous ansible. https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html. 12, 29

[10] Pulumi. Site web officiel de pulumi. https://www.pulumi.com/docs/get-started/gcp/create-project/. 9

[11] Vim. Site web de vim. https://www.vim.org/. 15

[12] YAML. Définition du langage yaml. https://en.wikipedia.org/wiki/YAML. 12

Liste des illustrations

Liste des tableaux

Listings

Annexes

Sur ce lien se trouve une liste des différents liens qui m'ont été utiles et que j'ai retenu pour la rédaction du rapport.

Résumé

Dans l'informatique, il existe de multitude de solutions permettant de faciliter, d'automatiser des taches répétitives. Un terme décrit cette méthodologie, le DevOps. Selon les fonctionnalités des solutions, trois grandes familles sortent, l'intégration et livraison continue, qui permet de créer et déployer des applications, la famille surveillance, journalisation et performances, qui permet d'avoir un retour sur les effets d'une ou plusieurs applications sur l'expérience de l'utilisateur, et enfin l'infrastructure as code qui permet de répéter d'automatiser des processus. Suite à une étude des différentes solutions, quatre solutions sont étudiées dans ce rapport, qui sont Terraform, Ansible, Chef et SaltStack. Avec une recherche sur les différentes fonctionnalités, et en prenant compte de la solution déjà en place au sein de l'entreprise, la solution retenue est SaltStack. Une bref présentation de ce qui est possible de faire avec SaltStack à été effectuée, en passant d'une simple ligne de commande, à la création de script permettant de changer un mot de passe sur un firewall FreeBSD.

Mots-clés : DevOps, Infrastructure as Code

Abstract

In the computer industry, there are a multitude of solutions to facilitate and automate repetitive tasks. One term describes this methodology, DevOps. Depending on the functionalities of the solutions, three main families come out, the integration and continuous delivery family, which allows to create and deploy applications, the monitoring, logging and performance family, which allows to have a feedback on the effects of one or several applications on the user experience, and finally the infrastructure as code, which allows to repeat to automate processes. Following a study of the different solutions, four solutions are examined in this report, which are Terraform, Ansible, Chef and SaltStack. With a research on the different functionalities, and taking into account the solution already in place within the company, the chosen solution is SaltStack. A brief presentation of what is possible to do with SaltStack was made, going from a simple command line to the creation of a script to change a password on a FreeBSD firewall.

Keywords :