Maud LAURENT - Valentin NORBERTO - Valentin PEYREGNE
Tuteurs : Cyril MARCH - Guillaume PENAUD

25 mars 2017

Projet tuteuré
Déploiement et provisionnement d'un IAAS avec OpenStack et Terraform

Table des matières

1 Introduction **2**
1.1 Le cloud computing . . . 2
1.2 Infrastructure as code . . . 3
1.3 Mouvance du Devops . . . 4
1.4 Contexte du projet . . . 4
1.5 Enjeu et problématique . . . 5

2 Présentation d'OpenStack et utilisation du client python-nova **6**
2.1 Introduction . . . 6
2.2 Les modules : services d'OpenStack. . . . 6
2.3 Le client Python nova . . . 7
2.4 Création de l'instance . . . 8

3 Présentation et mise en oeuvre de Terraform **10**
3.1 Présentation . . . 10
3.2 Caractéristiques . . . 11
3.3 Quelques cas d'utilisations . . . 11
3.4 Syntaxe utilisée . . . 12
3.5 Fonctionnement . . . 12
3.6 Configurations effectuées . . . 14

4 Provisionnement des instances Terraform avec Ansible **18**
4.1 Présentation . . . 18
4.2 Utilisation . . . 18
4.3 Fonctionnement . . . 19
4.4 Intégration avec Terraform . . . 21
4.5 Problèmes rencontrés . . . 22

5 Gestion du projet **23**

6 Qu'avait-il avant Terraform ? **24**
6.1 AWS CloudFormation . . . 24
6.2 Heat . . . 24
6.3 Terraform vs les autres logiciels . . . 24

7 Conclusion **26**

8 Bibliographie et Webographie **27**

9 Annexes **28**

1 Introduction

1.1 Le cloud computing

Le Cloud computing est un concept qui va permettre d'utiliser des ressources informatiques sans les posséder réellement, de fournir des services ou des applications accessibles partout depuis internet. Il y a de nombreux avantages à utiliser un cloud computing.

Tout d'abord, l'utilisateur n'a pas d'infrastructure à gérer, ce qui est parfois plus simple pour des entreprises, car c'est le fournisseur cloud qui s'occupe de la maintenance de ses équipements. Il permet donc une réduction des coûts en n'ayant pas besoin d'investir dans une infrastructure interne, mais en payant uniquement ce qu'il consomme à son fournisseur cloud.

Cependant, on a bien entendu des inconvénients comme le fait de savoir où le prestataire de service stocke nos données (territoire national, ou non -> problèmes de loi), si elles sont sécurisées vis-à-vis des hackers, confidentielles... On doit donc avoir confiance avec le prestataire hébergeant nos données.

Il existe trois catégories de services pour le cloud computing.

- Le cloud privé : infrastructure pouvant être gérée en interne par l'entreprise ou par un prestataire qui se verra confier les tâches relatives à l'administration et l'optimisation des performances. Il est conçu uniquement pour un seul utilisateur pour répondre au mieux aux besoins. Ce modèle a pour avantage de laisser à l'entreprise le contrôle à la fois sur la gestion des services, des données et de l'infrastructure. Le fait que ce soit un système fermé permet de mieux connaître les paramètres de sécurité, les garanties de service et la politique de confidentialité. Cependant, le déploiement de ce type d'infrastructure est très coûteux à mettre en place.
- Le cloud public : structure souple et ouverte proposée par des tiers spécialisés comme Amazon Web Services, Microsoft Azure, IBM, Google Compute Engine ou encore Cloudwatt. Le plus souvent, ces services sont vendus sur demande, le client va donc être facturé sur ce qu'il consomme. L'ensemble de l'infrastructure est géré par le fournisseur de service, ce qui permet une utilisation plus souple pour le client. Le cloud public s'adapte rapidement aux différents besoins, c'est ce qui charme le plus les entreprises (ne pas être limitées par le volume de données). L'un des inconvénients est l'absence de contrôle sur cette solution, que ce soit sur les données ou sur la rapidité (beaucoup d'utilisateurs, serveurs mutualisés) pas forcément adaptée à nos besoins.Ce service est intéressant d'un point de vue économique. Il n'y a pas de matériel ou d'informaticiens à gérer, plus le client utilisera le service plus la facture sera élevée.
- le cloud hybride : c'est un système mixte qui mélange le cloud privé et public. Le client va faire appel à plusieurs clouds indépendants les uns des autres, ce qui permet de placer les données sensibles et confidentielles dans un cloud privé et les autres dans un cloud public. Avec ce type de cloud, on va aussi pouvoir réduire les coûts d'exploitation en tirant l'avantage des deux infrastructures, on va ainsi dimensionner son cloud privé pour une charge moyenne et le cloud public pour répondre aux montées de charge.

Les différents modèles de cloud englobent plusieurs types de services, que l'on peut regrouper en trois parties :

- **IaaS - Infrastructure as a service** : Corresponds à la partie infrastructure du cloud. Il est surtout destiné aux entreprises, qui vont pouvoir externaliser leur infrastructure matérielle. En effet, dans le cas du IaaS, le service de cloud va fournir si besoin toute une infrastructure

informatique virtualisée.
L'entreprise va alors faire appel à ce type de service pour accéder aux ressources informatiques, mais dans un environnement virtualisé à travers une connexion internet. L'entreprise cliente va également pouvoir choisir le système d'exploitation et y installer les différents outils dont elle a besoin. De plus, ce type de service représente un réel avantage économique pour les entreprises qui vont pouvoir, faire évoluer rapidement une infrastructure sans avoir besoin d'acheter et d'installer le matériel elle-même.
L'IaaS peut avoir plusieurs utilités pour une entreprise, tout d'abord, elle peut tout simplement servir d'hébergement cloud à la fois pour le stockage de données pour une entreprise qui décide de ne pas avoir de données en interne, ou encore servir d'hébergement pour un site web, ce qui peut être très utile, car comme on peut avoir une extensibilité à la demande, il sera facilement possible de gérer les fortes demandes dont le site pourra faire l'objet. Elle peut également servir comme expliqué précédemment, d'infrastructure pour l'entreprise cliente, cette dernière va alors louer les ressources dont elle a besoin, il est alors possible d'augmenter son infrastructure sans investir dans du matériel.
Les acteurs français du IaaS sont Online.net, OVH ou encore Cloudwatt.

— **PaaS - Platform as a service** : Ce modèle se "pose" sur le modèle IaaS, c'est-à-dire que le système est déjà installé, on va alors pouvoir installer les applications, les configurer et les utiliser. On peut ainsi installer de nombreuses applications comme PostgreSQL, Gitlab, Wordpress, Piwik, LAMP, ...
Ce type de cloud va permettre aux entreprises d'avoir un environnement d'exécution rapide et très utile pour les développeurs. Ces derniers sont ainsi assistés depuis la conception jusqu'à la phase de déploiement de leurs applications.
Comme pour le IaaS, les clients ne paient que les ressources qu'ils utilisent. Le PaaS offre ainsi différents avantages comme le fait que les entreprises vont pouvoir mettre en place la plateforme correspondant à leurs besoins, il n'y a pas besoin d'investir dans une infrastructure physique et c'est une solution très pratique pour les développeurs.

— **SaaS - Software as a Service** : Le logiciel en tant que service est la couche finale du Cloud. C'est la plus simple à utiliser pour le client. L'utilisateur de ce type de service va pouvoir accéder à des applications via internet. Ces derniers n'ont rien à gérer, l'ensemble de l'infrastructure étant administrée par le prestataire. Google, Twitter, Facebook sont des exemples de SaaS. L'utilisateur peut accéder au service depuis un smartphone, un ordinateur, une application ou un navigateur web à la seule condition d'avoir une connexion internet. Contrairement à l'IaaS et le PaaS pensé pour les développeurs, le SaaS est pensé pour n'importe quel utilisateur lambda.
Il y a de nombreux avantages aux SaaS comme le fait qu'il n'y ait pas besoin de coûts d'équipement étant donné que l'application est sur le Cloud. Elle va ainsi disposer de son infrastructure pour fonctionner. Il n'y a pas de frais d'installation, pas de maintenance à effectuer et l'on peut y accéder depuis n'importe quel équipement disposant d'une connexion internet.

1.2 Infrastructure as code

L'infrastructure as code ou encore « infrastructure programmable »est une technique qui vise à administrer une infrastructure via du code (avec divers langages possibles suivant la technologie utilisée).

L'IAC est utilisée à la fois pour gérer des configurations, mais aussi pour déployer et automatiser le provisionnement d'une infrastructure. L'entreprise possède donc le code pour pouvoir fournir

et gérer nos serveurs tout en permettant d'automatiser les tâches.

L'IAC, n'est plus seulement utilisée par les administrateurs système. En effet, les développeurs de logiciels et d'applications peuvent facilement écrire un code d'infrastructure pour pouvoir se créer un environnement à des fins expérimentales pour tester leurs logiciels.

De nombreux outils proposent un système d'infrastructure as code. Par exemple, Vagrant va permettre de créer un environnement virtuel grâce au code contenu dans le Vagrantfile, ou encore Ansible qui va permettre l'installation de logiciels sur une instance.
Plus récemment, le logiciel Docker permet d'automatiser le déploiement d'applications dans des conteneurs logiciels. Il utilise également l'IAC avec les dockerfile.

L'IAC permet de suivre les modifications d'une infrastructure, en cas de problème, il sera alors très simple de revenir à la configuration précédente.

Cependant, l'infrastructure as code possède aussi des inconvénients. Une mauvaise configuration sera dupliquée sur toutes les machines concernées. Il faut aussi faire attention aux modifications des machines pour éviter un éventuel problème de configuration au sein de l'infrastructure.

1.3 Mouvance du Devops

Devops est la concaténation du mot anglais *development* et *operations*. Au début de l'informatique en entreprise, les applications ne jouaient pas un grand rôle et étaient peu intégrées, il n'y avait alors pas de séparation entre développement et opérations, la même équipe d'informaticiens se chargeait à la fois du développement de l'application et de sa maintenance.

L'évolution de l'informatique d'entreprise a entraîné l'évolution de l'utilisation des logiciels. Aujourd'hui, les logiciels ont une place beaucoup plus importante ce qui a conduit à une séparation du développement et de la partie opérationnelle en deux équipes distinctes. L'équipe de développement apportait les changements aux logiciels souvent le plus rapidement possible pour un moindre coût tandis que l'équipe opération garantissait la stabilité du système en se concentrant sur la qualité.

L'objectif du mouvement devops est de coordonner l'ensemble des équipes du système d'information.

1.4 Contexte du projet

Xilopix [1] est une start-up basée à Épinal qui développe un moteur de recherche pensé pour le tactile. Leur technologie se diffère d'autres navigateurs web. Lors d'une recherche, il sera possible de l'affiner en choisissant à quelles informations on souhaite avoir accès parmi une combinaison d'éléments de différentes natures (textes, images, vidéos, pages web, sons, etc.).

De fait, Xilopix va permettre d'améliorer la pertinence des résultats de recherche tout en offrant une nouvelle expérience utilisateur à la fois visuelle, tactile et ludique.
Xilopix utilise une infrastructure cloud avec plusieurs hébergeurs (ovh et Cloudwatt) tous sur OpenStack. OpenStack permet de faire du IaaS (le consommateur peut choisir pour ses machines : le système d'exploitation et les différents outils dont il a besoin).

1. http ://pro.xilopix.com

Xilopix souhaiterait ne plus être dépendant d'OpenStack et être libre d'utiliser d'autres systèmes tels qu'AWS par exemple. L'infrastructure as code, est aussi une technologie qui les intéressent pour sa simplicité et la possibilité de versionnement avec des systèmes comme git. C'est dans cette optique qu'intervient Terraform.

1.5 Enjeu et problématique

L'objectif du projet est de développer un proof of concept sur l'outil Terraform pour pouvoir démontrer les avantages de cet outil.

> Un proof of concept est une réalisation expérimentale concrète et préliminaire, courte ou incomplète, illustrant une certaine méthode ou idée afin d'en démontrer la faisabilité. [2]

Pour ce faire, nous avons mis en place un cluster de quatre machines virtuelles ainsi qu'un réseau, un sous-réseau et un routeur pour recréer une infrastructure minimale fonctionnelle avec l'outil Terraform. Xilopix ayant une infrastructure utilisant OpenStack, nous avons créé des configurations fonctionnant pour administrer l'outil OpenStack.

À partir de ces études, Xilopix pourra choisir d'utiliser ou non cet outil.

La finalité du projet est d'obtenir une création rapide et demandant un minimum d'intervention humaine pour déployer plusieurs machines aussi bien sous OpenStack, sous AWS ou tout autre provider. Terraform va donc nous permettre de réduire les dépendances entre ces outils et les infrastructures qui les utilisent tout en facilitant leurs mises en place. Dans un premier temps,

nous allons vous présenter OpenStack, ainsi que son client python-nova.
Nous passerons ensuite à la présentation de Terraform, ainsi qu'aux configurations que nous avons effectuées pour le projet.
Ensuite, nous verrons le provisionnement avec Ansible, son intégration à Terraform et les problèmes que nous avons rencontrés à ce sujet.
Chapitre 5, vous trouverez comment nous avons géré le projet.
Le chapitre 6 présente et compare quelques technologies semblables à Terraform.
Nous conclurons ce rapport dans le chapitre 7.

2. source : https ://fr.wikipedia.org/wiki/Preuve_de_concept

2 Présentation d'OpenStack et utilisation du client python-nova

Xilopix utilise l'outil OpenStack, nous avons donc dû apprendre rapidement son fonctionnement pour pouvoir ensuite travailler pleinement sur sa mise en place avec Terraform.

2.1 Introduction

OpenStack est un projet qui est né en 2010 (licence Apache 2.0) par l'entreprise RackSpace. OpenStack est un logiciel libre qui va nous permettre de faire du cloud computing et qui permet de faire de l'IaaS pour du cloud privé ou public.
Le but d'OpenStack est d'offrir à son utilisateur une multitude de modules qui va lui permettre de faire de l'infrastructure as a service c'est-à-dire déployer des machines virtuelles en optimisant les ressources matérielles. On peut les déployer dynamiquement pour une courte durée, mais il est également possible de créer un ensemble de machines constituant une infrastructure externe.

2.2 Les modules : services d'OpenStack.

OpenStack se compose de plusieurs modules pour fonctionner, avec des modules plus ou moins importants pour la création d'infrastructure.

2.2.1 Keystone, le service d'identité

Keystone fournit un service d'authentification et d'autorisation pour les autres services d'OpenStack. Il fournit un catalogue d'end points pour tous les services d'OpenStack.

2.2.2 Glance, la gestion d'images

Glance stocke et récupère des images disques de machines virtuelles. OpenStack Compute les utilise lors du provisioning d'instance.

2.2.3 Nova, le Compute

Nova est le coeur du projet OpenStack, il gère le cycle de vie des instances dans un environnement OpenStack. Les tâches incluent la planification, la création et la mise hors service de machines virtuelles à la demande.

2.2.4 Horizon, l'interface web

Fournis un portail libre-service de type web permettant d'interagir avec les services sous-jacents d'OpenStack, comme le lancement d'une instance, l'attribution d'adresses IP ou la configuration des contrôles d'accès.

2.2.5 Cinder, le service de disques persistants

Fournis un stockage "bloc persistant" aux instances en cours d'exécution. Son architecture basée sur des drivers de type plug-in facilite la création et la gestion des devices de stockage bloc.

2.2.6 Neutron, la gestion de réseaux

Permets le Network-Connectivity-as-a-Service pour d'autres services d'OpenStack, comme Compute. Il fournit une API utilisateur pour définir les réseaux et les attachements à ces réseaux. Il possède une architecture modulaire qui permet le support de la plupart des fournisseurs et des technologies réseau.

2.2.7 Swift, le stockage d'objet

Stocke et récupère des objets de données non structurées via une API RESTful basée sur HTTP. Le service est hautement tolérant aux pannes avec sa réplication de données et son architecture de type scale-out. Son implémentation diffère des serveurs de fichiers à répertoires montables. Le service écrit les objets et les fichiers vers plusieurs disques, en s'assurant que les données sont répliquées sur un cluster de serveurs.

2.2.8 Heat, le service d'orchestration

Orchestre de nombreuses applications de cloud composites en utilisant soit le format de template natif HOT ou le format CloudFormation d'AWS, soit au travers d'une API REST native OpenStack, soit au travers d'une API compatible avec CloudFormation.

2.2.9 Ceilometer, le service de métrologie

Surveille et mesure un cloud OpenStack dans un but de facturation, de mesure de performances, de scalabilité et de statistiques. Voici un schéma qui permet de montrer le lien entre tous les modules.

2.3 Le client Python nova

Le client Python-nova est un client en ligne de commande pour le module Nova OpenStack , il va nous permettre de mettre en oeuvre 100% de l'API Nova, et aussi la gestion des instances, des images, etc. . . .

2.3.1 Installation de python-nova

Pour installer le plug-in python-nova, il faut avoir préalablement installé python et son système d'installation pip. Pour lancer l'installation, il suffit de taper pip install -U python-novaclient .

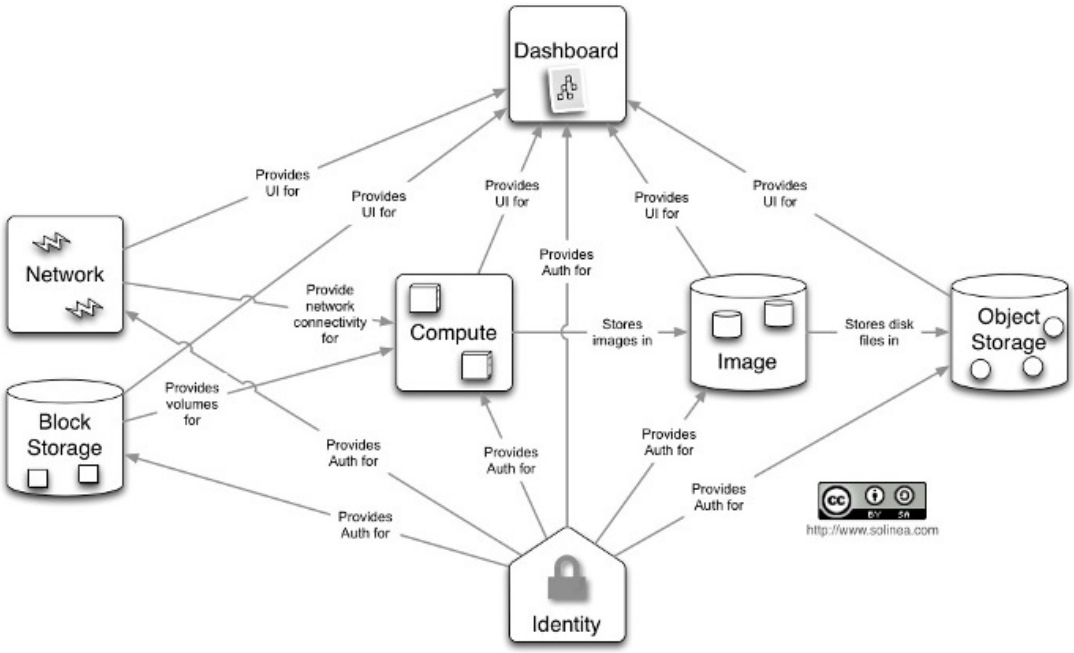


FIGURE 1 – Schéma d'organisation des modules d'OpenStack source : http ://lea-linux.org/documentations/OpenStack

2.3.2 Configuration des variables d'environnement pour OpenStack

Pour configurer toutes les variables, OpenStack génère un fichier RC contenant la totalité des variables d'environnement à configurer.

Depuis Cloudwatt, il faut aller dans les paramètres *accès et sécurité* puis *accès API* et enfin télécharger le fichier. En effet, Cloudwatt génère un fichier contenant toutes les variables d'environnement nécessaire à la configuration de la connexion OpenStack. Ce qui ne sera pas forcément le cas avec un autre service d'hébergement tel qu'Ovh.

L'exécution du fichier se fait grâce à la commande `source 0750182707_projet_tutore_2017-openrc.sh` et permet la configuration automatique des variables.

2.3.3 Liste des instances

La liste des instances créées est visible à l'aide de la commande `nova list`.

2.4 Création de l'instance

Dans cette partie, nous allons voir comment une instance est générée avec OpenStack.

2.4.1 Génération de la clef ssh

`ssh-keygen` Une clef ssh a été généré pour le projet, elle ne contient aucune passphrase pour éviter les problèmes de compatibilité avec Ansible. Ce point va être expliqué plus bas.

2.4.2 Intégration clef ssh au keypair OpenStack

```
nova keypair-add --pub-key .ssh/id_rsa.pub SSHKEY
```

2.4.3 Choix du flavor

`nova flavor-list` affiche la liste des flavors disponibles. Une fois choisi, il faut récupérer son ID qui sera renseigné lors de la création de l'instance.

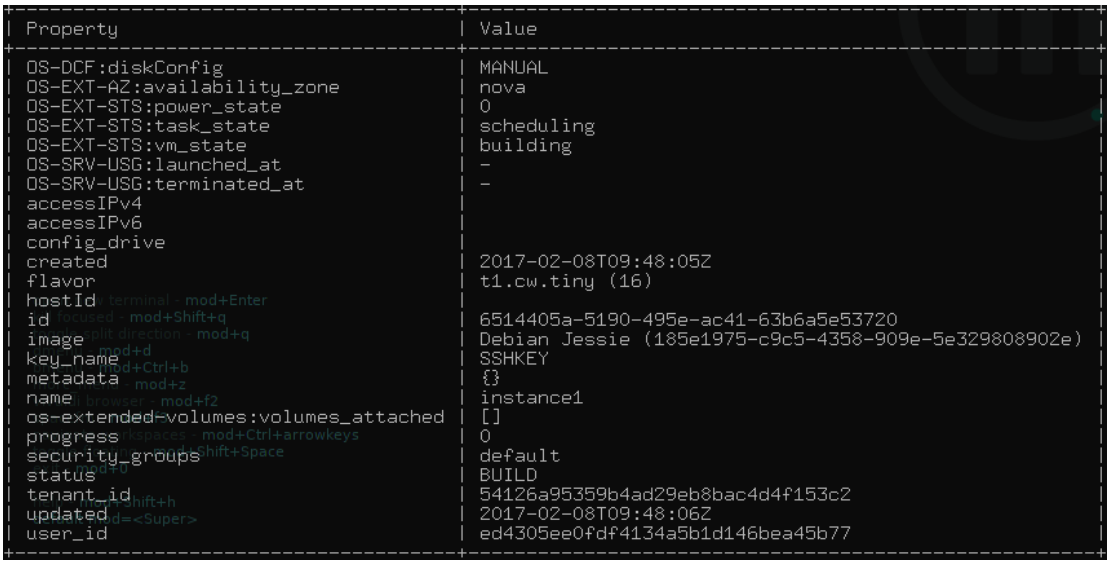
2.4.4 Choix de l'image (système installé)

`nova image-list` affiche la liste des images systèmes disponibles. Une fois choisi, il faut récupérer son ID qui sera demandé lors de la génération de l'instance.

2.4.5 Création de l'instance

`nova boot --key-name SSHKEY --flavor 16 --image 185e1975-c9c5-4358-909e-5e329808902e instance1` Pour la création de l'instance, on retrouve quatre éléments :

- le nom du keypair
- l'id du flavor
- l'id de l'image
- le nom de l'instance



Property	Value
OS-DCF:diskConfig	MANUAL
OS-EXT-AZ:availability_zone	nova
OS-EXT-STS:power_state	0
OS-EXT-STS:task_state	scheduling
OS-EXT-STS:vm_state	building
OS-SRV-USG:launched_at	-
OS-SRV-USG:terminated_at	-
accessIPv4	
accessIPv6	
config_drive	
created	2017-02-08T09:48:05Z
flavor	t1.cw.tiny (16)
hostId	
id	6514405a-5190-495e-ac41-63b6a5e53720
image	Debian Jessie (185e1975-c9c5-4358-909e-5e329808902e)
key_name	SSHKEY
metadata	{}
name	instance1
os-extended-volumes:volumes_attached	[]
progress	0
security_groups	default
status	BUILD
tenant_id	54126a95359b4ad29eb8bac4d4f153c2
updated	2017-02-08T09:48:06Z
user_id	ed4305ee0fdf4134a5b1d146bea45b77

FIGURE 2 – Paramètres demandés par OpenStack pour la création d'une instance

3 Présentation et mise en oeuvre de Terraform

Dans cette partie, nous allons tous d'abord faire une présentation du logiciel. Ensuite, nous expliquerons comment créer une infrastructure complète avec Terraform.

3.1 Présentation

Terraform est un outil développé en Go qui permet la gestion d'infrastructure à l'aide de recettes. L'objectif de ce logiciel est de permettre une configuration centralisée, rapide et efficace d'une infrastructure.

Terraform fonctionne avec des fichiers textes pour configurer les futures infrastructures. Ces fichiers sont appelés « recette », ils servent à décrire l'architecture des providers tels qu'OpenStack ou AWS. La configuration se fait dans un fichier « main.tf »qui est écrit en HCL [3]. La configuration peut aussi être générée automatiquement par machine avec le format JSON [4]. L'extension du fichier sera alors « main.tf.json ».

Un provider pour Terraform est un service pour le cloud computing, généralement un IaaS comme c'est la cas pour OpenStack. Terraform permet de gérer les composants de bas niveau comme les IaaS, le stockage et la mise à niveau, ainsi que des composants hauts niveaux comme les entrées DNS et les fonctionnalités SaaS.

Terraform génère un plan d'exécution se basant sur les recettes et décrivant les étapes qu'il va effectuer. Puis exécute le plan précédemment défini pour mettre en place l'infrastructure. Terraform détecte les changements effectués dans les fichiers et créés de nouveaux plans d'exécution conformément à ces changements.

3. HashiCorp Configuration Language
4. JavaScript Object Notation

3.2 Caractéristiques

— **Infrastructure as code :** L'infrastructure est décrite en utilisant une syntaxe de configuration de haut niveau (HCL). Cela permet à une infrastructure d'être versionné et traité comme tout autre code.
— **Plan d'exécution :** Terraform a une étape de « planification »qui génère un plan d'exécution. Le plan d'exécution montre les actions que Terraform effectuera lorsqu'il sera lancé. Cela permet d'augmenter la stabilité en évitant d'avoir des surprises lorsque Terraform manipule l'infrastructure.
— **Graphique des ressources :** Terraform construit un graphique de toutes les ressources des infrastructures, et parallélise la création et la modification de toutes ces ressources non dépendantes. Grâce à cela, Terraform construit l'infrastructure aussi efficacement que possible, et les utilisateurs peuvent avoir un aperçu des dépendances de leur infrastructure.
— **Automatisation des changements :** Des ensembles complexes de changements peuvent être appliqués à une infrastructure avec une interaction humaine minimale. Pour se faire, Terraform se base sur le plan d'exécution et le graphique de ressources mentionnés précédemment, évitant ainsi des erreurs humaines possibles.

3.3 Quelques cas d'utilisations

3.3.1 Self-service Cluster

Dans de grandes organisations, il devient plus attrayant de créer une infrastructure « self-service », permettant aux équipes de gérer leur propre infrastructure à l'aide de l'outillage fourni par l'équipe centrale d'exploitation.

À l'aide du fichier de configuration de Terraform, on peut rapidement prendre connaissance de l'ensemble de l'infrastructure décrite dans le fichier. Il est alors possible de lancer le script Terraform depuis n'importe quel pc, la seule condition étant d'avoir l'exécutable. Cela représente un réel avantage de portabilité.

3.3.2 Démonstrations de logiciels

À l'instar de Vagrant qui permet la création d'environnement virtualisé, les éditeurs de logiciels peuvent fournir une configuration Terraform pour créer et démarrer une infrastructure de démonstration. Ceci permet aux utilisateurs finaux de mettre en place rapidement un environnement de test sur leur propre infrastructure.

3.3.3 Multi Cloud-Déploiement

Il est souvent attrayant de répandre l'infrastructure sur plusieurs clouds pour augmenter la tolérance aux pannes. En utilisant une seule région ou un seul fournisseur cloud, la tolérance aux pannes est limitée par la disponibilité de ce fournisseur. Avoir un déploiement multi cloud permet une meilleure récupération de la perte d'une région ou tout le fournisseur.

Terraform permet la configuration de plusieurs providers en une seule configuration. Cela simplifie la gestion et l'orchestration des providers, en aidant la création d'infrastructures multi cloud.

3.4 Syntaxe utilisée

Les configurations de Terraform sont écrites en HashiCorp Configuration Language (HCL). Ce langage se veut facile à écrire et à lire. L'écriture des configurations peut aussi se faire en JSON.

3.4.1 Les bases du langage

Commentaires - # sur une seule ligne - /* mon commentaire sur plusieurs lignes */

Affectation des valeurs

```
key = value # la valeur peut être une chaîne, un nombre ou un booléen
```

Chaînes multi lignes : On utilise `<< - EOF` et `EOF` pour créer des chaînes multi-lignes ce qui permet principalement d'intégrer des scripts dans la configuration.

Il existe également de nombreuses fonctions utilisables avec HCL comme la fonction format(format, args, ...) effectuant la mise en forme d'une chaîne de caractère. Les différents blocs se définissent avec des accolades dans le même principe que des fonctions dans les autres langages connus.
`language ressource "nom_type_ressource" "nomRessource" {  ... }`

3.5 Fonctionnement

Terraform étant développé en Go, il n'a pas besoin d'être installé. Il suffit de télécharger une archive .zip et de l'extraire. Il est ensuite possible d'utiliser les commandes associées à Terraform avec `./Terraform ...`. Pour faciliter l'utilisation des commandes, il est recommandé de copier le fichier dans */usr/local/* et d'ajouter ensuite le chemin menant jusqu'au fichier en question dans le PATH `PATH=/usr/local/...:$PATH`.

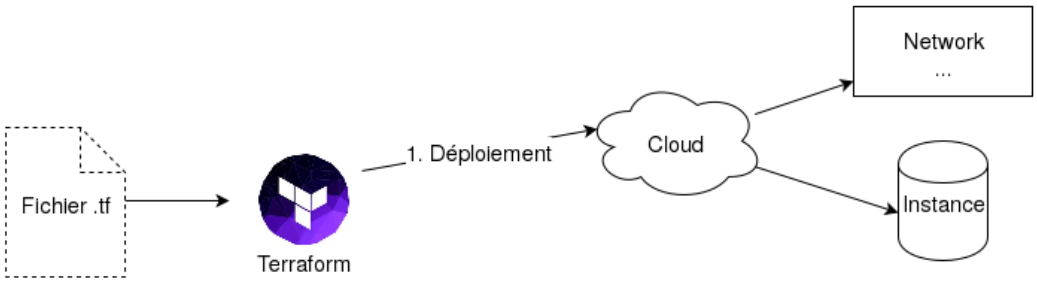


FIGURE 3 – Utilisation de Terraform dans un environnement

Terraform peut être composé de plusieurs fichiers de configuration pour une infrastructure. Dans ce cas, les fichiers sont lus par ordre alphabétique, mais la priorité reste au fichier *main.tf*. Les fichiers Terraform se composent de différents types de bloc : le bloc provider et le bloc ressource. Chacun de ses blocs peut se retrouver plusieurs fois dans un fichier.

3.5.1 Bloc `provider`

C'est la partie configuration du provider avec principalement les accès pour la connexion à celui-ci. Terraform peut contenir plusieurs blocs provider. Ce bloc gère le cycle de vie des ressources (create, read, update, delete).

```
provider "OpenStack" {
    user_name  = "admin"
    tenant_name = "admin"
    password  = "pwd"
    auth_url  = "http://myauthurl:5000/v2.0"
}
```

Cloudwatt offre la génération d'un fichier .sh avec la totalité des identifiants et accès pour OpenStack. Pour la connexion, nous avons pu omettre le bloc provider, Terraform se charge de récupérer les variables environnementales correspondant aux paramètres dont il a besoin pour retrouver le provider.

3.5.2 Bloc `ressource`

Partie de création et gestion des ressources. Les ressources sont les composants physiques et logiciels qui composent l'infrastructure.

```
resource "OpenStack_compute_instance_v2" "nomTerraform" {
}
```

3.5.3 Variables

Les variables peuvent être externalisées dans un fichier « variables.tf »ou « .tfvars ». Pour appliquer des variables enregistrées sous cette dernière extension, il faut lancer la commande suivante `Terraform apply -var-file=variables.tfvars`. Les variables sont généralement utilisées dans les fichiers .tf sous la forme suivante `${var.nomVar}`.

3.5.4 Modules

Il est possible d'intégrer des modules à Terraform. Terraform se charge du téléchargement et de l'intégration des modules.
L'appel d'un module se fait de la même manière qu'une ressource, mais avec le mot clef *module*. La source est ensuite indiquée dans une variable. La source peut provenir d'un dépôt github, d'une archive (de multiples formats sont reconnus comme tar.gz, zip, bz2...), d'un dossier local....

Concrètement, un module Terraform est un dossier contenant des fichiers *.tf* qui ont besoin d'être utilisé plusieurs fois. Pour régler ce problème de copie multiple du code, il est possible d'utiliser ce dossier en tant que module et ainsi avoir un code plus clair.

Exemple d'un fichier correspondant à un module contenu dans le dossier child.

```
variable "memory" {}
output "received" {
  value = "${var.memory}"
```

```
}
```

Appel du module

```
module "child" {
  source = "./child"
  memory = "1G"
}
```

3.5.5 Plugins

Terraform a été créé avec la possibilité d'importer des plug-ins. Les plug-ins servent à rajouter de nouvelles fonctionnalités à Terraform.

Pour installer un plug-in, il faut récupérer le code du plug-in. Et créer un fichier *~/.terraformrc* contenant un bloc providers avec une variable indiquant le chemin jusqu'au dossier du plug-in. L'utilisation du plug-in se fait après dans le fichier .tf de l'infrastructure et s'utilise comme un bloc ressource normal en suivant la syntaxe définie dans la documentation du plug-in en question.

3.5.6 Les commandes

- `Terraform plan` – génère un plan d'action de la configuration. Le plan inclut toutes les actions faites et montre les modifications que va effectuer Terraform.
- `Terraform plan -destroy -out=destroy.tf` – génère un plan d'action qui a pour objectif de détruire tout le projet défini pas les fichiers de configuration. Le résultat est enregistré dans un fichier pour ensuite être appliqué avec `Terraform apply destroy.tf`.
- `Terraform destroy` – détruis l'infrastructure générée par Terraform avec le compte importé (identique à la technique du dessus).
- `Terraform apply` – applique le code Terraform en générant ce que `Terraform plan` a montré précédemment
- `Terraform graph` – permet la visualisation du plan. Cette commande génère un fichier dot qui peut ensuite être transformé en image ce qui permet d'obtenir toutes les étapes que va effectuer Terraform en schéma.
- `terraform show` – montre les infrastructures en place

3.6 Configurations effectuées

Nous avons effectué plusieurs configurations nécessaires pour créer notre infrastructure.

3.6.1 Keypair

Une des premières configurations effectuées fut l'ajout de clef ssh pour le projet. Cet ajout avait pour objectif de nous permettre de nous connecter en ssh avec les instances créées.

```
resource "OpenStack_compute_keypair_v2" "my_keypair" {
  name = "my_keypair"
  public_key = "${var.keypair}"
}
```

Terraform prend en paramètre pour cette ressource un nom et la clef publique à ajouter. Cette clef ssh est requise lors de la création d'instances. En effet, celles-ci prennent en paramètre une keypair - `key_pair = "${OpenStack_compute_keypair_v2.my_keypair.name}"` - pour permettre la connexion ssh à l'instance en question. Cependant, une seule keypair peut être intégrée dans la ressource *instance*. Pour avoir tous accès en ssh aux instances, nous nous sommes partagé la clef privée créée spécialement pour le projet et n'ayant pas de passphrase pour permettre à Ansible de se connecter ensuite.

3.6.2 Instances

Les instances sont la plus grande partie de la configuration, elles correspondent aux machines privées virtuelles (vps) qui vont être créées.

```
resource "OpenStack_compute_instance_v2" "vps" {
  count = 3
  name = "vps-test-${(count.index)+1}"
  image_id = "185e1975-c9c5-4358-909e-5e329808902e"
  flavor_id = "16"
  key_pair = "${OpenStack_compute_keypair_v2.my_keypair.name}"
  security_groups = ["${OpenStack_compute_secgroup_v2.terraform.id}"]
  floating_ip = "${element(OpenStack_compute_floatingip_v2.terraform.*.address,
  count.index+1)}"
  network {
    name = "${OpenStack_networking_network_v2.network_1.name}"
    fixed_ip_v4 = "192.168.0.1${(count.index)+1}"
  }
}
```

Une instance peut être créée dans une configuration unique, mais il est aussi possible d'en générer plusieurs automatiquement avec une seule ressource associée grâce au paramètre `count`. La création des instances se fait donc en partant de zéro. Avec `count.index`, nous pouvons récupérer l'index actuel de la boucle générée par Terraform. Chaque instance est rattachée à un ou plusieurs réseaux selon les besoins. Pour notre proof of concept, nous avons utilisé un seul réseau. Pour connecter les instances au réseau, il faut écrire un bloc *network* comme ci-dessus.

3.6.3 Security group

Le security group permet d'autoriser les transmissions sur certain port. Un security group fonctionne sur le même principe qu'un firewall. Il est composé de règles *rule*. Une règle est définie pour un port. Nous avons crées un security group nommé « terraform »composé d'une seule règle permettant la connexion ssh (port 22).

```
resource "OpenStack_compute_secgroup_v2" "terraform" {
  name        = "terraform"
  description = "security group"
  rule {
    from_port   = 22
    to_port     = 22
    ip_protocol = "tcp"
```

```
    cidr        = "0.0.0.0/0"
  }
}
```

3.6.4 Ip flottantes

Les ip flottantes permettent aux instances d'avoir une ip publique, permettant ainsi d'accéder en ssh aux instances.

Terraform offre la possibilité de générer automatiquement les adresses ip en les piochant dans un pool public d'adresses. Cependant, l'utilisation d'Ansible pour le provisionnement des instances requiert la connaissance des adresses ip flottantes attribuées aux machines.

Pour répondre à ce problème, plusieurs solutions s'offraient à nous.

- La première est l'importation des adresses ip avec `Terraform import`.
  L'importation avec Terraform fonctionne de la manière suivante : la ressource est importée avec la commande `Terraform import`, et pour être utilisable, elle doit avoir une ressource créée dans le fichier .tf. L'importation d'une multitude d'adresses ip entraîne la création du même nombre de ressources.
- La seconde solution est la création d'une liste contenant les adresses ip flottantes créées depuis l'interface web de l'hébergeur. L'appel de l'adresse se fait depuis la ressource instance de Terraform qui vas récupérer une adresse dans la liste.

```
# Fichier variables.tf
variable "id_ip_flottante" {
    default = ["84.39.49.19","84.39.46.157","84.39.44.165","84.39.41.206"]
}
# Fichier main.tf
resource "OpenStack_compute_instance_v2" "vps" {
        floating_ip = "${var.id_ip_flottante[(count.index)+1]}"
}
```

3.6.5 Réseau, sous-réseau et routeur

Terraform permettant de créer toute une infrastructure, nous nous sommes aussi penchés sur la création du réseau, de ses sous-réseaux et du routeur nous permettant un accès au monde extérieur.

Réseau et sous-réseau Pour être utilisables, les instances doivent être connectées à un réseau. Terraform offre la possibilité de créer rapidement et facilement un réseau ainsi que les sous-réseaux et ports utiles à celui-ci. Les instances sont donc configurées pour être intégrées à ce réseau et obtenir une adresse ip dans ce dernier.

```
resource "OpenStack_networking_network_v2" "network_1" {
  name = "resTerraform"
  admin_state_up = "true"
}
```

Sous-réseau

```
resource "OpenStack_networking_subnet_v2" "subnet_2" {
  name = "SousRes_2"
  network_id = "${OpenStack_networking_network_v2.network_1.id}"
  cidr = "192.168.0.0/24"
  ip_version = 4
}
```

Port du sous-réseau

```
resource "OpenStack_networking_port_v2" "port_1" {
  name = "port_1"
  network_id = "${OpenStack_networking_network_v2.network_1.id}"
  admin_state_up = "true"
}
```

Routeur Pour que l'infrastructure créée soit opérationnelle, il faut lui autoriser un accès à l'extérieur du réseau. Pour se faire, nous passons par un routeur qui est lui-même composé d'une interface le reliant à un des réseaux crée précédemment.

```
resource "OpenStack_networking_router_v2" "router_1" {
  name = "routerTerraform"
  admin_state_up = "true"
  external_gateway = "6ea98324-0f14-49f6-97c0-885d1b8dc517"
}
```

Avec la configuration que nous avons effectuée tout le long de notre projet, nous sommes arrivés à un environnement fonctionnel et accessible ressemblant à l'image ci-dessous.

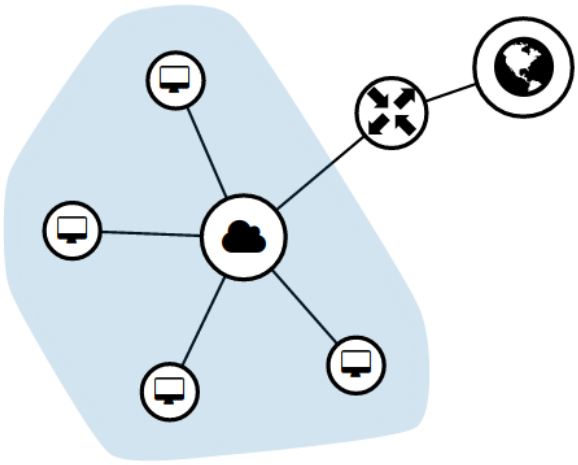


FIGURE 4 – Réseau obtenu avec notre configuration Terraform

4 Provisionnement des instances Terraform avec Ansible

Terraform permet aussi le provisionnement de ses instances avec différents provisionners comme Chef, Puppet ou encore Ansible. Cependant, Terraform n'offre un service que pour Chef.
Mais permet d'exécuter différentes commandes automatiquement depuis la machine lançant `Terraform apply` avec le provisionner `local_exec` ou depuis la machine générée avec Terraform grâce au provisionner `remote_exec`.

Celui-ci se complète avec le bloc `connection` effectuant une connexion ssh avec les identifiants désirés. Ces dernières se mettant dans les instances. Les ressources de type `null-ressource` permettent d'exécuter des commandes après la création de certaines ressources. Grâce à cela, il est possible de lancer Ansible automatiquement à la fin de la création des instances.

4.1 Présentation

Ansible est un logiciel libre de licence GPL-3.0, créé par Michael DeHaan et écrit en python. La première version du logiciel date d'avril 2012 et la dernière version est la 2.2.1.0 datant de janvier 2016.
Ansible est un outil de gestion de configuration qui se veut plus simple dans la prise en main que d'autres produits concurrents comme Puppet ou Chef.

4.2 Utilisation

D'une manière générale, on utilise un outil de gestion de configuration pour pouvoir maintenir une configuration identique sur différentes machines que ce soit des serveurs ou des machines virtuelles.
Il est difficile de pouvoir gérer à la main tout un parc informatique, il est peu probable que

chaque machine ait parfaitement la même configuration (du par exemple à un oubli) et de plus, la tâche peut être très répétitive et une réelle perte de temps suivant la taille du parc. Il est donc conseillé d'utiliser un outil de gestion de configuration qui va permettre de mettre à jour de façon automatisée des serveurs.
Par exemple si on souhaite changer un paramètre ou encore déployer un nouveau paquet, il nous suffit de modifier l'outil de gestion de configuration. Il permet également de contrôler et de corriger tout écart dans la configuration.

Pour plusieurs raisons, Ansible est plus simple d'utilisation que les logiciels concurrents. Tout d'abord, il est agent-less, c'est-à-dire qu'il n'y a pas de configuration de type client/serveur à gérer. Il faut seulement un serveur ssh sur les machines à gérer pour qu'elle puisse être configurée par Ansible.

Ansible utilise aussi un langage qui est très simple à aborder, le YAML. YAML acronyme de Yet Another Markup Language est proposé par Clark Evans en 2001, il reprend le concept d'autres langages comme le XML.
Le YAMl se veut être le plus lisible possible par tous. Tous les fichiers en YAML peuvent commencer par — et terminer par ... , c'est le format qui indique le début et la fin d'un document. Lorsque l'on fait une liste, chaque élément de la liste est de type « key : value »que l'on appelle "hash" et "dictionnaire". Tous les membres d'une liste correspondent à une ligne qui doivent commencer au même niveau d'indentation et par un tiret plus un espace.

exemple :

```
---
# Liste d'OS LInux
OS:
        - Debian
        - Manjaro
        - Linux Mint
        - Ubuntu
...
```

4.3 Fonctionnement

Ansible fonctionne avec des états. Par exemple : Pour un dépôt Git, Ansible va déjà vérifier s'il y a déjà eu un git clone, il va alors vérifier que le dossier existe et faire ensuite la mise à jour, dans le cas ou le dossier n'est pas présent il fera le git clone.

Ansible utilise des playbooks pour pouvoir déclarer des configurations, ce sont la base du système de gestion de configuration et de déploiement multi machines. Les playbooks sont écrits en YAML pour ne pas ressembler à un script de configuration, mais plutôt à un modèle descriptif d'une configuration. Ces derniers vont donc permettre de décrire les différentes étapes du déploiement.

Les hôtes et les utilisateurs Tout d'abord, quand on crée notre playbook, il va falloir choisir sur quelles machines on souhaite appliquer le provisionnement. Pour cela, on utilise un fichier hosts qui va nous permettre de cibler sur quelles machines on souhaite utiliser le playbook.
Il existe aussi une ligne remote_user qui permet elle, de choisir le compte utilisateur auquel on sera connecté sur le host, c'est-à-dire qu'on peut par exemple la remplir avec root.

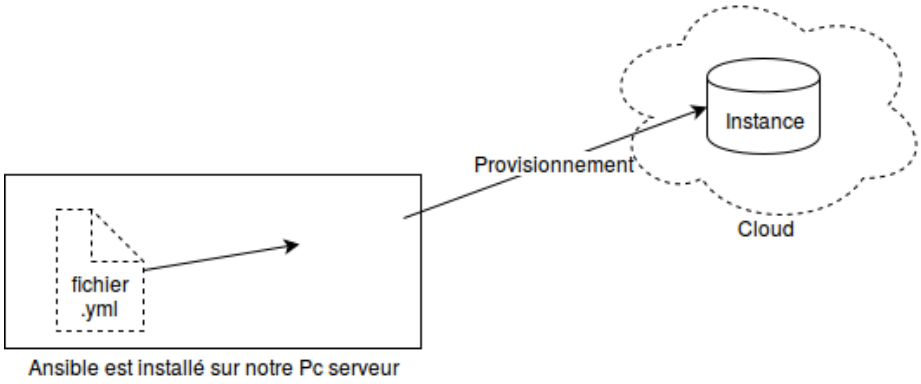


FIGURE 5 – Fonctionnement Ansible

La liste des tâches Dans chaque playbook, on va trouver une liste de tâches associée à un hôte. Ces dernières sont exécutées dans l'ordre et une par une. Une tâche va être identifiée par le champ **name** qui est affiché lors de l'exécution, elle doit être lisible et avoir une bonne description pour permettre de pouvoir l'identifier facilement. Et on utilise aussi le champ **state : present** qui va avant d'installer le paquet vérifier s'il est déjà présent.

Le premier playbook Pour tester le fonctionnement d'Ansible, nous avons tout d'abord essayé de provisionner une machine virtuelle que nous avions créée précédemment à l'aide de Terraform.

On se connecte en ssh sur cette instance avec la commande :

```
ssh -i id_rsa_nopass cloud@84.39.46.157
```

Sur cette machine, on va par exemple essayer d'installer des paquets non présents comme Nginx et Htop. Avec le playbook suivant, on va pouvoir provisionner notre machine virtuelle :

```
- hosts: all
  become: true
  tasks:
    - name: Mise à jour
      apt:
      update_cache: yes
    - name: Installation de Nginx
      apt:
      name: nginx
      state: present
    - name: Installation de Htop
      apt:
      name: htop
      state: present
```

Ensuite, on le lance avec la commande :

```
ansible-playbook --private-key=~/.ssh/id_rsa_nopass -u cloud -i 84.39.44.165,
provisionement.yml
```

On a : Ensuite, on peut vérifier la présence de ces derniers en se connectant sur l'instance :

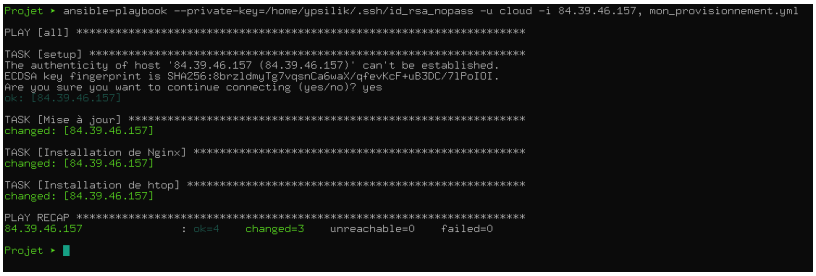


FIGURE 6 – Exécution Ansible

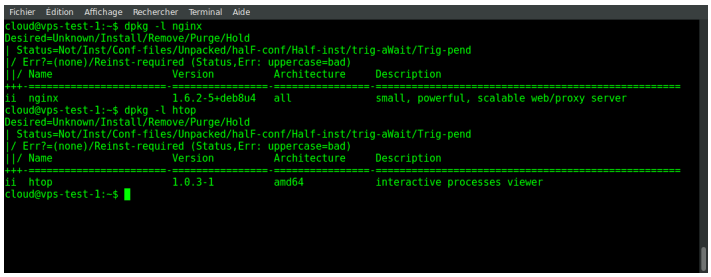


FIGURE 7 – Vérification des paquets

4.4 Intégration avec Terraform

Terraform et Ansible sont deux outils distincts, leurs emplois sont différents.
Terraform va permettre de mettre en place l'infrastructure. On va pouvoir décrire la topologie du réseau que l'on souhaite, la taille des instances, ...
Ansible, lui, va permettre la configuration de nos instances. Il va définir ce qui se trouve sur les machines, quels logiciels seront présents... C'est un outil de gestion de configuration. Ce n'est pas le cas de Terraform.

Terraform permet d'utiliser différents outils de gestion de configuration (Ansible, Puppet ou chef) pour configurer et provisionner les ressources qu'il crée. C'est pour cela que dans notre projet, nous utilisons les deux ensembles, ils ne sont pas en concurrence.

L'objectif est de pouvoir intégrer Ansible avec Terraform. C'est-à-dire, que depuis notre script .tf on va appeler notre playbook Ansible. Comme dit précédemment Terraform permet d'utiliser différents provisionners, mais seul Chef possède un service.

Pour pouvoir utiliser Ansible, on va utiliser une **null-ressource**. Les *null-ressource* de Terraform sont des ressources n'étant rattachées à aucune autre ressource, ni aucun type de ressource, ce ne sont donc ni des instances, ni des networks...

La partie *triggers* permet d'attendre la fin de création des instances

```
triggers {
    cluster_instance_ids = "${join(",", OpenStack_compute_instance_v2.vps.*.id)}"
```

```
}
```

La partie *local-exec* permet de lancer une commande depuis la machine hôte, pour notre projet nous l'avons donc utilisée pour lancer notre commande Ansible.

```
command = "ansible-playbook --private-key=~/.ssh/id_rsa_nopass -u
cloud -i ${element(OpenStack_compute_instance_v2.vps.*.floating_ip, count.index)},
fichierAnsible.yml"
```

4.5 Problèmes rencontrés

Au début avant l'utilisation des null-ressources, nous avions mis l'appel à Ansible toujours dans un provisionner local-exec, mais la position de celui-ci était dans la création de l'instance. De ce fait, Ansible essayait de se connecter à l'instance avant que celle-ci ne soit complètement créée. Or, Ansible a besoin d'une connexion ssh pour pouvoir fonctionner, et comme les instances n'étant pas complètement créées, elles ne pouvaient en aucun cas assurer une connexion ssh, ce qui provoquait des erreurs.

Pour résoudre ce problème, on a commencé à regarder s'il était possible qu'Ansible attende que toutes instances soient créées avant de tenter une connexion ssh.
Pour cela, nous avons utilisé le module **wait_for** d'Ansible qui permet d'attendre une condition spécifique avant de continuer. Dans notre cas, ce que l'on voulait, c'était une attente sur le port 22 (port de connexion ssh). On a donc testé :

```
- wait_for :
    port: 22
```

Mais malgré cette attente, on avait toujours le même problème.

Au fil des tests, on a remarqué que les erreurs venaient le plus souvent lors de la connexion ssh. Pour éviter tous doutes, nous avons tenté de se connecter directement à une instance en ssh à la fin de sa création. On a vérifié que l'on ne pouvait pas se connecter en ssh sur nos instances. De fait, nous nous sommes rendu compte que le problème ne venait peut-être pas de nous, mais de Cloudwatt.

5 Gestion du projet

Pour pouvoir se partager et aussi pour versionner nos travaux, nous avons utilisé GitHub tout au long de notre projet.
Nous avons créé une architecture constituée de 4 dossiers Docs, Notes, Projet et Rapport. Dans le dossier Docs, nous avons mis toute la documentation que l'on a pu trouver et qui nous a été utile. Dans le dossier Notes, on peut trouver tout ce que l'on a pu rédiger sur différentes parties de notre projet, à la fois des petits tutoriels, des aide-mémoire, ou encore des parties du rapport. Le dossier Projet contient nos différents scripts Terraform et Ansible et pour finir dans Rapport, on peut y trouver le plan de notre rapport, les images utilisées, le rapport final ainsi que nos slides pour le jour de l'oral.

Le projet portait sur une seule technologie qui est Terraform. La répartition des tâches s'est donc fait par rapport aux différentes parties que nous avons dû créer pour faire fonctionner notre infrastructure.

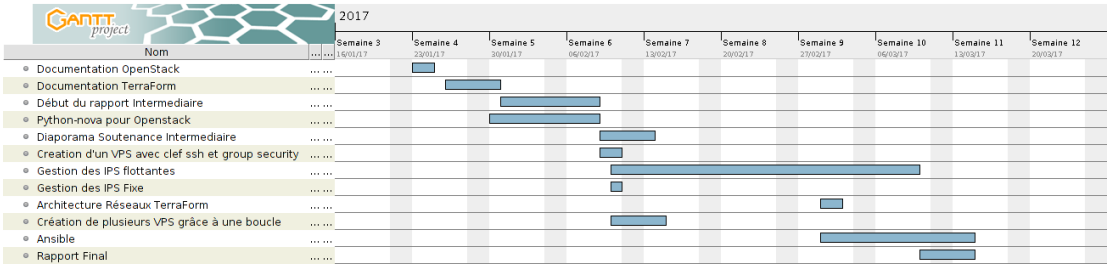


FIGURE 8 – Diagramme des tâches en fonction du temps

6 Qu'avait-il avant Terraform ?

Dans cette partie, nous allons parler des différents services concurrents à Terraform, de leurs avantages et inconvénients.

6.1 AWS CloudFormation

CloudFormation fournit par Amazon Web Service permet de créer et de gérer un ensemble de ressources qui sont liées, de les ordonner, les mettre en service et les actualiser.
Il permet d'avoir aussi une infrastructure as code avec de simples fichiers textes au format JSON ou YAML. Il fonctionne uniquement avec AWS, mais le fonctionnement est semblable à celui de Terraform. Il permet de créer un modèle qui décrit toutes les ressources AWS que l'on veut (telles que des instances Amazon EC2 ou des instances de base de données Amazon RDS). De plus, AWS CloudFormation s'occupe de leur mise en service et de leur configuration.
AWS CloudFormation propose des modèles d'exemples déjà créés qui peuvent être utilisés. Il est aussi possible de créer des modèles personnalisés.

6.2 Heat

Heat est un module de la partie orchestration d'OpenStack. La mission du programme OpenStack Orchestration est de créer un service accessible pour gérer l'ensemble du cycle de vie des infrastructures et des applications dans le cloud OpenStack.
Heat fournit une orchestration à base de modèle pour décrire une application cloud. En s'exécutant, OpenStack appelle différentes API pour générer l'exécution d'applications cloud. Un template Heat décrit l'infrastructure pour une application cloud dans des fichiers textes qui sont lisibles et modifiables par les humains, et peut être géré par des outils de contrôle de version. Le logiciel intègre d'autres composants d'OpenStack. Les modèles permettent la création de la plupart des types de ressources OpenStack tels que les instances, ip flottantes, des volumes, des groupes de sécurité, les utilisateurs, etc. Ainsi que certaines fonctionnalités plus avancées telles que la haute disponibilité. Heat gère principalement l'infrastructure, mais les templates intègrent aussi des outils de gestion de configuration logiciel tels que Puppet et Ansible.

6.3 Terraform vs les autres logiciels

Les outils comme CloudFormation, Heat, etc permettent à une infrastructure d'être décrite dans un fichier de configuration. Les fichiers de configuration permettent à l'infrastructure d'être élastiquement créé, modifiée et détruite. Terraform est inspiré par les problèmes qu'ils résolvent et élimine complètement les dépendances avec ces services.

Terraform utilise de la même façon des fichiers de configuration pour détailler la configuration de l'infrastructure, mais il va plus loin par le diagnostic ainsi que la permission de fournisseurs multiples et des services combinés et composés. Par exemple, Terraform peut être utilisé pour orchestrer un AWS et un groupe OpenStack en simultané, en permettant des fournisseurs du 3ème parti comme CloudFlare et DNSIMPLE d'être intégré pour fournir des services de DNS et CDN. Ceci permet à Terraform de représenter et gérer l'infrastructure entière avec ses services de soutien. Au lieu de seulement le sous-ensemble qui existe dans un fournisseur seul. Il fournit

une syntaxe unifiée, au lieu d'exiger que des opérateurs utilisent des outils indépendants et non interopérables pour chaque plateforme et service.

Terraform sépare également la phase de planification de la phase d'exécution, en utilisant le concept d'un plan d'exécution. En exécutant Terraform plan, l'état actuel est actualisé et la configuration est consultée pour générer un plan d'action. Le plan comprend toutes les actions à entreprendre. Quelles ressources seront créées, détruites ou modifiées ? Terraform génère un plan d'exécution décrivant ce qu'il va faire pour atteindre l'état désiré. En utilisant Terraform graph, le plan peut être visualisé pour montrer les commandes qui vont être exécutées par celui-ci. Une fois que le plan est capturé, la phase d'exécution peut être limitée aux seules actions du plan. D'autres outils combinent les phases de planification et d'exécution, ce qui signifie que Terraform montre les effets de changement qui va se produire sur l'infrastructure, qui devient rapidement insoluble dans les grandes infrastructures. Terraform permet aux opérateurs d'appliquer des changements avec confiance, car ils savent exactement le résultat qui les attend.

6.3.1 Tableau comparatif des solutions

	Terraform	Heat	CloudFormation
Plan	Plan d'exécution des modifications, ajouts et suppressions	Pas de plan ; exécution directe	Pas de plan ; exécution directe
Provider	OpenStack, AWS, DigitalOcean, Google Cloud, CloudFlare et plein d'autres	OpenStack	AWS
Graph	Disponible au format dot {Terraform graph}	Non	Pas nativement
Rollback	Oui, pas automatiquement	Oui, option à activer dans la commande	Oui par défaut

7 Conclusion

Terraform est un outil permettant de déployer des infrastructures de manière simple et efficace. Il fournit une syntaxe simple et unifiée permettant de gérer presque toutes les ressources sans apprendre de nouveaux outils.En outre, Terraform est un outil open source. En plus de HashiCorp, la communauté autour de Terraform contribue à étendre ses fonctionnalités, corriger les bugs et documenter de nouveaux cas d'utilisation. Terraform aide à résoudre un problème qui existe dans chaque organisation et fournit un standard qui peut être adopté pour éviter de réinventer la roue entre et au sein des organisations.

Terraform permet de représenter tout matériel physique, machine virtuelle, conteneur, courrier électronique, fournisseur DNS. Grâce à cette flexibilité, Terraform permet de résoudre beaucoup de problèmes différents. Il peut gérer une seule application ou il peut très bien gérer un ensemble de centres de données. Dans ce projet, nous avons appris à manier Terraform, il possède encore certaines capacités que nous n'avons pas exploitées tels que les modules ou plug-ins dont nous n'avons pas eu l'utilité pour notre projet.

Ce projet nous a permis d'apprendre à travailler en groupe sur un projet conséquent. Nous avons appris à gérer séparément des tâches tout en mettant en commun nos avancées sur le projet. Nous avons aussi pu voir les difficultés que l'on peut avoir avec un hébergeur cloud comme Cloudwatt, notamment l'attente des accès et les quelques problèmes d'accès réseau que nous avons eus.

8 Bibliographie et Webographie

Terraform
Documentation Terraform : https://www.terraform.io/docs/index.html
Terraform : Up & Running : http://shop.oreilly.com/product/0636920061939.do
http://www.matt-j.co.uk/2015/03/27/OpenStack-infrastructure-automation-with-terraform-part-2/

OpenStack
Documentation OpenStack : Rapport donné en exemple https://members.loria.fr/LNussbaum/ptasrall2017/rapport_g_openstack2014.pdf
Documentation officielle : http://docs.openstack.org

Ansible
Documentation Ansible : http://docs.ansible.com/ansible/playbooks_intro.html
https://quentin.dufour.io/blog/2015-10-02/ansible

9 Annexes

Code Terraform

keypair.tf

```
# Keypair générale
resource "openstack_compute_keypair_v2" "my_keypair" {
  name = "my_keypair"
  public_key = "${var.keypair}"
}
```

network.tf

```
# Réseau
resource "openstack_networking_network_v2" "network_1" {
  name = "resTerraform"
  admin_state_up = "true"
}
# Sous-réseau
resource "openstack_networking_subnet_v2" "subnet_2" {
  name = "SousRes_2"
  network_id = "${openstack_networking_network_v2.network_1.id}"
  cidr = "192.168.0.0/24"
  ip_version = 4
}
#Port du sous-réseau
resource "openstack_networking_port_v2" "port_1" {
  name = "port_1"
  network_id = "${openstack_networking_network_v2.network_1.id}"
  admin_state_up = "true"
}
```

router.tf

```
# Router
resource "openstack_networking_router_v2" "router_1" {
  name = "routerTerraform"
  admin_state_up = "true"
  external_gateway = "6ea98324-0f14-49f6-97c0-885d1b8dc517"
}
resource "openstack_networking_router_interface_v2" "int_1" {
  router_id = "${openstack_networking_router_v2.router_1.id}"
  subnet_id = "${openstack_networking_subnet_v2.subnet_2.id}"
}
```

securityGroup.tf

```
# Secure group
resource "openstack_compute_secgroup_v2" "terraform" {
  name        = "terraform"
  description = "security group"

  rule {
    from_port   = 22
    to_port     = 22
    ip_protocol = "tcp"
    cidr        = "0.0.0.0/0"
  }
}
```

variables.tf

```
# pool
variable "pool" {
    default = "public"
}
# keypair
variable "keypair" {
    default = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCoj+AJej+p324R08l8Y7trRwe
+2OlmvGHIJNW6U7VWDS0jSFr3QJJQsBIJ1KLCxIP0alveWNqUMz4xbUBof8Ai8ULelN5Gk64EsRmkH2Bncxs
KkoraVYr0hBom+k6d7jyONZLYohtKrGyIRC5h2RfwypCDJkAjV10XpJWOsJCJkPPl
+8vuLNRdyPQqx6iBiy7mwDDuF6oNW7CZK3HPe2n9geZSTtjE18FNvHvCTOITVWSnGdSPy2e89ahemM0B3ROo
aogn76Xw0BeNrhUfywXyA4GaS4F1Y28uduvXIO8QRaIiIWRjhPH83wswjnPnDciPEg7/uAa/yZSy1SXFU5V"
}
#Ip flotantes
variable "id_ip_flottante" {
    default = ["84.39.49.19","84.39.46.157","84.39.44.165","84.39.41.206"]
}
```

main.tf

```
# vps
resource "openstack_compute_instance_v2" "vps" {
  count = 3
  name = "vps-test-${(count.index)+1}"
  image_id = "185e1975-c9c5-4358-909e-5e329808902e"
  flavor_id = "16"
  key_pair = "${openstack_compute_keypair_v2.my_keypair.name}"
  security_groups = ["${openstack_compute_secgroup_v2.terraform.id}"]
  floating_ip = "${var.id_ip_flottante[(count.index)+1]}"
  # le +1 c'est parce qu'on a test vps qui utilise le 0
  network {
    name = "${openstack_networking_network_v2.network_1.name}"
```

```
    fixed_ip_v4 = "192.168.0.1${(count.index)+1}"
  }
}
resource "openstack_compute_instance_v2" "test-network" {
  name = "test-network"
  image_id = "185e1975-c9c5-4358-909e-5e329808902e"
  flavor_id = "16"
  key_pair = "${openstack_compute_keypair_v2.my_keypair.name}"
  security_groups = ["${openstack_compute_secgroup_v2.terraform.id}"]
  floating_ip = "${var.id_ip_flottante[0]}"
  network {
    name = "${openstack_networking_network_v2.network_1.name}"
    fixed_ip_v4 = "192.168.0.2"

  }
}
resource "null_resource" "ansible-provision"{
  triggers {
    cluster_instance_ids = "${join(é",", openstack_compute_instance_v2.vps.*.id)}"
  }
  count = 3
  provisioner "local-exec" {
    connection {
        type = "ssh"
        user = "cloud"
       private_key = "${file("/home/ypsilik/.ssh/id_rsa_nopass")}"
    }
         command = "ansible-playbook --private-key=/home/ypsilik/.ssh/id_rsa_nopass
          -u cloud -i ${element(openstack_compute_instance_v2.vps.*.floating_ip,
           count.index)}, provisionement.yml"
  }
}
```

Code Ansible

```
- hosts: all
  become: true
  tasks:
    - name: Mise à jour
      apt:
        update_cache: yes
    - name: Installation de Nginx
      apt:
      name: nginx
      state: present
    - name: Installation de htop
      apt:
      name: htop
      state: present
```