Utilisation d'*OpenStack Ironic* en tant que gestionnaire de bureau virtuel

Amaury MAILLÉ

Mai 2018

Résumé

Ce projet a pour but d'étudier la faisabilité de la mise en place d'un *cluster OpenStack* doté du service *Ironic* en remplacement du système de *templates* actuellement utilisé sur les machines des salles TP Réseaux (TPR) du bâtiment Nautibus.

Mots-clés : cloud, OpenStack, CEPH, Ironic, baremetal, diskless.

1 Introduction

Dans le cadre de l'UE « Projet d'Orientation en Master » du master 1 d'Informatique, j'ai effectué un travail de recherche autour du sujet « Utilisation d'*OpenStack Ironic* en tant que gestionnaire de bureau virtuel », sous la supervision de Mr Fabien RICO, enseignant chercheur, et de Mr Romain CHANU, administrateur de l'*OpenStack* de l'université. Ce travail s'est effectué sur le site de la Doua, de Février 2018 à Mai 2018.

L'objectif de ce projet est de conclure sur la possibilité d'utilisation d'*OpenStack* doté du service *Ironic* en tant que remplaçant du système de *templates*. Cela a été réalisé au travers de la mise en place d'un *cluster* (ensemble de machines travaillant ensemble) de test, sur lequel a été configuré un *OpenStack* minimaliste, auquel le service *Ironic* a été ajouté.

2 Système actuel

A partir de la troisième année de licence d'informatique, les étudiants sont amenés à réaliser des travaux pratiques dans le cadre d'enseignements tels que « LIFASR6 - Réseaux », « M1IF18 – Programmation système & temps réel », ou encore « M2 – Sécurité réseau ». Ces travaux pratiques nécessitent d'effectuer des configurations avancées au niveau du système d'exploitation, comme la configuration du réseau ou l'installation de services. Ces opérations requièrent des privilèges administrateur (appelés également privilèges *root*), ce qui donne un contrôle total de la machine à l'utilisateur ; des erreurs de manipulations peuvent donc endommager gravement le système.

Deux éléments se dégagent de cet état de fait : la nécessité d'un processus de préservation de la machine, permettant de la ramener rapidement à un état fonctionnel en cas d'erreur de manipulation ; et la nécessité d'une sauvegarde individuelle des configurations effectuées par un étudiant lors d'un TP. En effet, il est courant qu'un TP se déroule sur plusieurs séances, et que d'autres TPs aient lieu entre-temps. Une préservation des modifications par étudiant et par machine est donc nécessaire pour éviter que les étudiants n'aient besoin de reprendre à zéro à la séance suivante. Mr Thierry EXCOFFIER a proposé une solution inspirée par le système *aufs*[1]. Une explication plus détaillée d'*aufs* est disponible en annexe.

L'idée de Mr EXCOFFIER a été de faire en sorte que les modifications apportées au système soient sauvées dans un répertoire séparé, stocké sur la machine. Ainsi, quand un étudiant récupère un *template* de TP et

1 Système de fichier permettant de monter un répertoire comme une somme d'autres répertoires.

nomme sa sauvegarde, ou dès qu'il reprend une sauvegarde précédente, il va travailler dans ce répertoire séparé. Ce répertoire est sommé avec le système présent de base pour que l'étudiant se retrouve avec un système complet et fonctionnel, le tout de façon transparente. Une explication plus complète est disponible en annexe. Une visualisation du fonctionnement se trouve ici.

3 Motivation

Ce système présente quelques inconvénients. Les modifications appliquées sur *init* rendent les mises à jour du système particulièrement complexes, changer de distribution Linux ou effectuer un changement de version demanderait de réinstaller tout le système de Mr EXCOFFIER, et ce pour chaque machine. Par conséquent, il n'est pas possible de proposer plusieurs distributions Linux de façon simple ; le choix d'*upgrader* les installations actuelles doit être soigneusement évalué, afin de s'assurer que la nouvelle version de Linux permette toujours aux étudiants de pratiquer les différents TPs proposés. Par exemple, il n'est pas envisageable d'installer un *OS* temps réel sur les machines des salles TPR, alors que la possibilité de manipuler un tel *OS* serait très intéressant dans l'UE « Temps réel ». Enfin, la mise en place d'une administration centrale des machines est rendue complexe, et se fait actuellement au travers de scripts shell nécessitant que toutes les machines soient préalablement allumées.

4 Solution envisagée : *OpenStack*

4.1 Présentation

OpenStack [Ope18a] est une plateforme libre et *open source* de *cloud-computing*, déployée en tant qu'*Infrastructure as a Service* (*IaaS*[2]). *OpenStack* est, en général, utilisé pour créer des machines virtuelles (*Virtual Machine, VM)* à la demande, tout en offrant des *backend* pour le stockage d'images système (*Glance*), de fichiers sous forme d'objet (*Swift*), ou de volumes (*Cinder*), ainsi que pour la création de réseaux virtuels (*Neutron*). Ce qui va nous intéresser ici est le service *Ironic* qui permet de provisionner (fournir un système sur lequel la machine va démarrer) des machines physiques (*baremetal*). Cela permettrait de gérer les machines des salles TPR de façon centralisée, mais également de *booter* plusieurs systèmes d'exploitation différents (Windows, Linux), sur des versions différentes.

4.2 Pertinence

Un *OpenStack* en remplacement du système actuel des salles TPR devra, à minima, contenir les services suivants :

- *Keystone* : le service chargé de l'authentification ; chaque requête nécessitant un certain niveau de privilèges lui est d'abord envoyée afin qu'il puisse déterminer si l'utilisateur émetteur de la requête est en droit de l'effectuer ;
- *Glance* : le service chargé du stockage des images de machines virtuelles, nécessaires pour instancier les *VM* ;
- *Nova* : le service chargé de la gestion des machines virtuelles : instanciation, sauvegarde de l'état, sauvegarde du disque ;

2 Un modèle de *cloud-computing* où des ressources de calcul sont rendues disponibles par-dessus l'Internet

- *Neutron* : le service chargé de fournir un accès réseau aux *VMs* ; cet accès peut se faire via divers mécanimes virtuels : pont, *switch*, *NAT*…
- *Ironic* : le service *baremetal* d'*OpenStack*, il permet de gérer le *provisionnement* de machines *physiques* grâce à un mécanisme de *management out-of-band* ; la gestion des différentes architectures physiques se fait au travers de *plugins*..

Les quatre premiers sont les services de cœur nécessaires pour mettre en place un *cluster* minimaliste permettant d'instancier des *VM*, tout en leur fournissant un accès réseau et du stockage. L'ajout d'*Ironic* permettra de provisionner des machines *physiques* et non plus virtuelles. Les relations dans un *cluster* minimaliste sont représentées ici.

Afin qu'*OpenStack* puisse être envisagé comme remplaçant du système actuel, il doit accomplir certaines opérations et répondre à certains besoins. Les deux principaux sont la préservation de la machine si jamais un étudiant vient à commettre une erreur de manipulation, et une possibilité de sauvegarde du travail effectué.

- La préservation de la machine est immédiatement garantie : si l'utilisateur a effectué une manipulation qui rend le système inutilisable, un redémarrage de la machine accompagné d'un déploiement de l'image système source résout le problème ;
- La sauvegarde du travail des étudiants peut se faire au travers du service *OpenStack Cinder*, qui gère des volumes pouvant être montés sur les *VMs* : chaque étudiant possèderait un volume personnel par TP, et pourrait y stocker de façon permanente ses données.

OpenStack doit également apporter une solution aux inconvénients du système actuel, en particulier sa complexité de maintenance. Le principal problème vient du couplage très étroit entre le système de gestion de l'environnement et le système d'exploitation ; l'existence de l'un pose des contraintes sur l'évolution de l'autre. Au travers d'*Ironic*, la gestion des machines serait découplée de tout système présent sur les machines elles-mêmes. Ainsi, si un enseignant souhaite proposer un TP de manipulation d'un *OS* temps réel, il installe cet *OS* sur une machine, apporte les configuration nécessaire, en fait une image *Glance*, et les étudiants n'auront plus qu'à *booter* les machines physiques avec cette image. Cela résout le problème d'existence de plusieurs distributions, ainsi que les dilemmes lors des *upgrades* de machines.

Dans un tel contexte se pose la question du stockage et de la persistence des données. Stocker les fichiers modifiés directement sur le disque est trivial, stocker des images de systèmes ou de volumes est plus complexe, et volumineux. Il est nécessaire d'introduire de la redondance dans les données, pour répondre aux pannes mais également aux éventuelles pertes. Pour cela, nous avons décidé de faire communiquer le *cluster OpenStack* avec un *cluster CEPH*, dédié au stockage.

5 *CEPH, Object Storage*

CEPH [Cep18a] est une plateforme de stockage libre, implémentant du stockage objet[3]. Une explication détaillée du stockage objet est disponible en annexe.

Un *cluster CEPH*, dans sa version la plus simple, contient trois types de nœuds différents :

3 Objet = données + métadonnées personnalisables. Les métadonnées permettent de regrouper plus facilement des objets similaires, et accélèrent la recherche à travers des masses importantes de données.

- Un nœud « *admin* », utilisé en général pour le *monitoring* et l'expansion du cluster, avec notamment la configuration de nouveaux nœuds ;
- Des nœuds « *OSD* » pour *Object Storage Data*, qui correspondent aux nœuds qui formeront l'espace de stockage unifié objet ;
- Des nœuds « *MON* » pour *Monitor*, qui correspondent aux nœuds chargés d'effectuer la réplication des données dans le cluster, afin d'assurer la redondance, tout en offrant un accès aux données, soit via des requêtes directes, soit via une *API REST* (RADOS *Gateway*, compatible avec *OpenStack Swift*, le service chargé du stockage fichier dans *OpenStack*).

Grâce au stockage objet, *CEPH* peut mettre en place de la redondance de façon assez simple, tout en permettant un placement optimal des données.

Considérons l'exemple suivant : deux objets presque identiques sont demandés à être stockés dans *CEPH*, par exemple deux volumes *Cinder*. Pourquoi stocker les deux volumes tels quels et ne pas plutôt stocker la partie commune et ajouter des métadonnées sur les parties différentes pour ensuite reformer le volume à la volée ? C'est ce que *CEPH* va faire.

L'utilisation de *CEPH* comme stockage pour *Glance*, *Nova* ou encore *Cinder* est possible grâce à la modularité de ces services. Ils sont à même d'utiliser plusieurs types de stockage : *file*, qui stocke les images et volumes dans un système de fichiers (/var/lib/nova et /var/lib/glance en général) ; *rbd* (*RADOS*[4] *Block Device*) qui permet de les stocker dans un *cluster CEPH* ; *swift* qui permet de les stocker dans le service *OpenStack Swift* (et également dans un *cluster CEPH* au travers de la *RADOS Gateway API*, compatible *Swift*). *Schéma récapitulatif*.

6 Travail réalisé

6.1 Installation d'un *cluster OpenStack* minimaliste

La première étape de l'étude a consisté en la mise en place d'un *cluster OpenStack* minimaliste. Cela s'est déroulé en deux temps : la configuration d'un environnement de travail, puis l'installation du *cluster* proprement dite. Cette installation s'est faite sur un ensemble de quatre machines. J'ai choisi deux machines comme machines principales du *cluster OpenStack* ; l'une est devenue le *controller*, sur laquelle tourne *Nova Controller*, le composant maître de *Nova* chargé de la distribution à travers les nœuds *compute*, et chargé de la gestion des nœuds *compute* ; l'autre est devenue un nœud *compute*, sur lequel tourne *Nova Compute*, le service chargé de créer les *VMs* sur l'hyperviseur, de les éteindre etc.

6.1.1 Mise en place

Au cours des réunions initiales, nous avons décidé que la version d'*OpenStack* utilisée serait la version 16 « *Pike* », version la plus récente au 01 Février, et dans laquelle *Ironic* a vu l'arrivée du support du *boot diskless*, ainsi que la possibilité d'atteindre des machines physiques dans des *VLAN* ; auparavant, *Ironic* ne pouvait atteindre que des machines se trouvant dans un *flat network*. Nous avons également décidé d'utiliser le système d'exploitation « *Ubuntu* » version 16, version au support le plus long au 01 Février. Depuis, *OpenStack* est passé sur « *Queens* » et *Ubuntu* est passé en version 18 « *Bionic Beaver* ».

4 *Reliable Autonomic Distributed Object Store*

Les salles TPR accèdent à l'Internet d'une façon différente par rapport aux autres salles de TP, afin d'éviter que les étudiants ne commettent des erreurs de manipulation sur le réseau.

En termes techniques, les salles TPR utilisent une segmentation du réseau en *VLAN*. Par défaut, les machines sont placées dans le *VLAN* 100, sans accès Internet, puis, après authentification, basculées dans le *VLAN* 101, où elles bénéficient d'un accès Internet. Par ailleurs, les requêtes HTTP sont redirigées vers un *proxy* avant d'être servies. Cela m'a donc conduit à mettre en place deux choses : la segmentation en *VLAN* du *cluster* et une configuration de proxy ; pour ce dernier point j'ai choisi le programme Squid [Squ18].

La segmentation en *VLAN* permet une meilleure séparation des différents composants du *cluster*, tout en évitant également les communication parasites entre entités. Par exemple, une *VM* n'a pas à communiquer avec *CEPH*, et ignore par ailleurs son existence. Les mises à jour du fichier représentant le volume virtuel de la *VM* sont effectués par *OpenStack Nova*. *Nova* et *VM* dans un *VLAN*, *Nova* et *CEPH* dans un autre.

J'ai procédé comme suit pour la segmentation en *VLAN* :

- Le *VLAN* 101 correspond au *VLAN* de *management*. On y trouve le *controller OpenStack* ainsi que les nœuds *compute*. Après réflexion, il aurait été plus judicieux d'utiliser un *VLAN* différent du *VLAN* des étudiants des TPR. Cependant, ma maîtrise des *VLANs* était presque inexistante au commencement de ce projet, et il aurait été trop compliqué de reconfigurer l'intégralité des nœuds ;
- Le *VLAN* 102 correspond au *VLAN CEPH*. On y trouve l'ensemble des nœuds *CEPH*, ainsi que les nœuds *controller* et *compute* d'OpenStack, qui ont besoin d'un accès au *cluster CEPH*.
- Les *VLANs* 200 à 209 correspondent aux *VLAN VMs* et aux machines physiques *Ironic*. Les nœuds *compute* et *controller* d'*OpenStack* y sont également présents, étant donné que les *VMs* ainsi que les nœuds *baremetal* ont besoin de communiquer avec le *cluster*.

La configuration de proxy a été nécessaire pour la communication entre machines. En raison de la segmentation en *VLAN*, les communications HTTP dans le *VLAN* 102 (par exemple) ne doivent pas être envoyées au proxy standard des salles TPR, qui ne peut accéder à ce *VLAN*. De même, les machines ont des noms symboliques (*controller*, *compute*, etc.) qui ne peuvent être résolus ou autorisés par le proxy standard. Par conséquent j'ai choisi la machine *controller OpenStack* comme proxy HTTP intermédiaire, et l'ai configuré en lui laissant le soin de déléguer toutes les requêtes vers l'extérieur au proxy principal. J'ai choisi cette machine car qu'il s'agit de celle qui doit rester allumée en permanence et sans laquelle le *cluster* ne peut fonctionner.[5]

La configuration du proxy a été assez technique. D'une part je n'avais aucune expérience en la matière. D'autre part, le traitement du protocole *HTTPS* et plus particulièrement de la méthode *CONNECT*, dans le cas d'un *forward* entre proxys, est extrêmement mal expliqué dans la documentation et j'ai perdu plusieurs jours pour arriver à corriger le problème (*Squid* tentait de se connecter directement de puis la machine contrôleur à la machine distante, sans passer par le proxy permettant de sortir des salles TPR).

Une fois cette configuration initiale effectuée, j'ai pu commencer l'installation du *cluster* minimaliste.

6.1.2 *Cluster OpenStack*

Un *cluster OpenStack* minimaliste est composé des services *Keystone*, *Glance*, *Nova* et *Neutron*. Les configurations les plus techniques sont celles de *Neutron*, *Glance* et *Nova*. Pour effectuer l'installation, j'ai suivi

5 Idéalement, il faudrait plusieurs *controllers* pour éviter d'avoir un *single point of failure*.

le guide d'installation officiel [Ope18b]. J'ai pu procéder à la configuration de *Keystone*, *Glance* et *Nova* en l'espace d'une semaine et demi. En revanche, la configuration de *Neutron* a été beaucoup plus complexe. Une explication détaillée du fonctionnement de *Neutron* est disponible en annexe.

La configuration de *Neutron* est cruciale car c'est elle qui va permettre d'obtenir un accès aux *VMs*, en général via le protocole *SSH* (il est également possible de passer par le protocle *VNC*).

Dans le cas du présent *cluster*, j'ai choisi d'utiliser des *networks* de type *provider*[6], d'une part pour leur simplicité et d'autre part car les possibilités supplémentaires apportées par les *self-service networks*[7] ne sont pas utiles ici. Elles pourront l'être plus tard, en production, mais leur présence n'a aucun impact sur la mise en place d'*Ironic* (et pourrait au contraire la complexifier). En ce qui concerne le *mechanism*[8], j'ai choisi le *mechanism linuxbridge*, qui correspond à un *bridge* virtuel, pour sa simplicité de configuration. L'accès entre la *VM* et la machine physique hôte se fait donc au travers d'un pont. Enfin, compte tenu des remarques précédentes sur la segmentation en *VLAN*, j'ai choisi le *network type*[9] *VLAN* pour l'implémentation logique du réseau.

Cette configuration a été plus complexe que les autres en raison de la technicité du sujet, et de la documentation fournie. Le guide d'installation de *Neutron* [Neu18a] explique assez rarement les raisonnement derrière telle ou telle configuration. De plus, il n'incite pas, par exemple via des renvois vers d'autres articles, à comprendre comment fonctionne *Neutron*. Il n'est par exemple pas explicité qu'une option de configuration du *linuxbridge* (*physical_interface_mappings*) correspond à un tableau associatif de *nom_network* => *interface_pont*. La valeur *nom_network* est cruciale car elle doit impérativement être reportée dans la commande de création de *network*. Ce détail n'est pas explicité. De même, il n'est pas non plus explicité que *Neutron* créer des *IP namespaces* pour gérer les différents *networks*. Étant donné que j'ignorais jusqu'à l'existence du concept de *namespace* dans le noyau *Linux*, j'étais incapable d'accéder aux *VMs* que je créais. La configuration de *Neutron* a ainsi pris beaucoup plus de temps que l'installation de tout le reste du *cluster* minimaliste.

Les configurations de *Glance* et *Nova* sont quant à elles, similaires et plus simples. *Glance* stocke les images utilisées pour créer les *VMs*, *Nova* stocke les disques (volumes) des *VMs* en elles-mêmes.[10]Ces données sont volumineuses, nombreuses, et sensibles aux pannes. Si un étudiant ne peut pas récupérer son image de TP car le stockage *Glance* est temporairement indisponible, c'est problématique. Ceci est une situation où il est nécessaire de fournir un stockage extensible et capable de contenir des volumes de données extrêmement importants, mais qui peuvent également être très similaires dans une grande majorité. J'ai donc créé un *cluster CEPH* minimaliste pour répondre à ces problèmes.

J'ai utilisé deux machines supplémentaires pour configurer le *cluster CEPH* (c'est le minimum pour assurer un peu de réplication des données). Les deux machines sont à la fois *monitor* et *osd* (assurent la surveillance, la réplication et le stockage), et la première machine est également *admin*, c'est à dire la source de construction du *cluster CEPH*. Afin d'isoler les deux *clusters*, j'ai créé le *cluster CEPH* dans le *VLAN* 102. L'installation de *CEPH*

6 Un réseau virtuel sous sa forme la plus simple : pas de routage, d'adresses *IP* statiques ou autre

7 Un réseau virtuel plus technique, avec routage, adresses *IP* statiques, pare-feu, load balancing…

8 Un *mechanism* est une méthode que Neutron peut utiliser pour fournir l'accès à la VM (pont, switch…)

9 Le *network type* correspond à la structure du réseau virtuel dans l'infrastructure physique (VLAN, réseau plat…)

10 *Nova* stocke le volume virtuel correspondant au disque de la *VM*. *Cinder*, quant à lui, stocke des volumes que l'on peut monter en plus sur une *VM*.

s'est faite à l'aide de l'outil *ceph-deploy*,[Cep18b] qui est un outil développé spécialement dans le but de simplifier l'installation d'un *cluster CEPH*.

La difficulté majeure rencontrée dans cette partie est liée au script *ceph-deploy*. C'est un utilitaire qui permet de simplifier le déploiement d'un *cluster CEPH*. Le problème était lié au fait que *ceph-deploy* est conçu pour installer un *cluster CEPH Luminous* (12.0, la dernière version), mais, à moins que ce ne soit explicitement interdit, essayait d'installer *CEPH Jewel* (8.0). De fait, l'installation échouait en produisant des erreurs à priori absurdes. Ce n'était qu'un détail, mais assez subtil pour me faire perdre beaucoup de temps (trouver d'où venait le problème, comprendre comment le résoudre…).

Une fois cette installation effectuée, j'ai pu procéder à l'installation d'*Ironic*.

6.2 Installation d'*Ironic*

Ironic est plus technique à installer, en raison de ses interactions multiples avec les autres services (majoritairement *Neutron* et *Glance*). De plus, la gestion de machines physiques à distance est bien plus complexe que la gestion de machines virtuelles, pour lesquelles le service *nova-compute* est disponible en permanence. L'installation d'*Ironic* s'est faite en trois temps : une compréhension globale du fonctionnement d'*Ironic*, l'installation du service, et le *debug*.

6.2.1 Fonctionnement d'*Ironic*

Afin de procéder au démarrage d'une machine, *Ironic* a besoin de plusieurs choses. La phase de *boot* en elle-même se déroule en trois étapes, et chaque étape nécessite sa propre configuration.

Tout d'abord, le démarrage. Pour procéder au démarrage d'une machine éteinte, il est nécessaire d'effectuer du *out-of-band management* (contrôler la machine autrement que par son système d'exploitation), en contactant l'interface dédiée de la machine (si une telle interface n'est pas présente, *Ironic* ne peut pas gérer cette machine). Le démarrage peut être fait *via Wake-on-Lan*, *IPMI*[11], ou d'autres… *Ironic* gère cela au travers de *drivers*[12]. Des informations supplémentaires sur *Ironic* et les *drivers* sont disponible en annexe.

Vient ensuite la phase de d'initialisation. Au cours de cette phase, la machine démarre et contacte un serveur *DHCP* afin d'obtenir une adresse *IP*. Selon le mécanisme en place sur la machine au démarrage (*PXE*, *iPXE*), le serveur *DHCP* fournit des informations supplémentaires pour télécharger un *deploy ramdisk* et un *deploy kernel*. Ces deux éléments permettent à la machine de démarrer un système d'exploitation temporaire sur lequel se trouve un programme, l'*ironic-python-agent*. Le rôle de cet agent est de contacter *Ironic* pour obtenir le mode de récupération d'un *kernel*, d'un *initramfs* et d'un système complet, récupérer tout cela, et le mettre en place sur la machine. Enfin, le *deploy ramdisk* bascule la configuration de démarrage de la machine de *PXE / iPXE /* autre en configuration de service, et déclenche un *reboot*.

La machine démarre ensuite sur le système qui a été récupéré.

6.2.2 Installation d'*Ironic*

Ironic s'interface avec *Keystone* pour l'authentification, *Glance* pour le stockage des images (*deploy images* et images de système), *Nova* pour le lancement des machines et N*eutron* pour l'apport d'une connectivité. Dans

11 *Intelligent Platform Management Interface*, un ensemble de spécification ouvert pour effectuer du *out-of-band management*.

12 Un driver *Ironic* est un outil permettant de contrôler une des phases du démarrage : quelle méthode pour démarrer la machine à distance, quelle méthode pour récupérer les *deploy images*, les images système etc.

le cadre du *boot diskless*, il est nécessaire d'ajouter une interaction avec *Cinder*, qui offrira un volume virtuel pour effectuer le *boot* via *iSCSI*. Étant donné le temps qu'il me restait pour procéder à l'installation d'*Ironic*, j'ai pris la décision de ne pas implémenter le *boot diskless*.

Les interactions entre *Ironic* et *Keystone* ou *Nova* sont assez simples et ne nécessitent pas de configuration complexe, j'ai simplement dû créer un utilisateur pour *Keystone*, éditer la configuration d'*Ironic* pour lui renseigner la machine *controller* pour les différents services, et mettre à jour les autres services pour qu'ils prennent en compte *Ironic*. En revanche, dans le cas de *Neutron* et de *Glance*, c'est un peu plus délicat. Il faut effectuer les opérations que j'ai citées précédemment, mais il faut également en faire un peu plus.

Glance doit stocker plusieurs images, de trois à cinq en fonction du style de déploiement (*whole disk*[13] ou *partition image*[14]). Le problème se situe au niveau de la récupération des images et de l'authentification du client. Quand on décide de créer une *VM*, il n'y a pas de problème d'authentification entre *Keystone* et *Glance* ; cette authentification se fait soit au travers d'un *token* délivré par l'administrateur *OpenStack*, ou de façon plus simple via le « *Dashboard Horizon* », le service d'*OpenStack* qui offre une interface graphique pour gérer ses *VMs*. Dans le cas du *boot* d'une machine phsyique, il n'y a pas de *token* disponible (et on ne peut pas l'envoyer sur le réseau, même chiffré, au nœud *baremetal*). L'accès aux images doit donc se faire sans authentification pour certains *drivers*, ce qui n'est pas possible dans *Glance*, du moins pas sans configuration supplémentaire.

Afin de simplifier les choses, étant donné que je n'avais aucune idée de quel *driver* était présent sur les machines physiques, j'ai choisi d'utiliser un *driver* plus générique, fonctionnant avec une technologie que j'avais déjà configurée : le *driver pxe_wol_iscsi*. Ce *driver* effectue un *boot* via *Wake-on-Lan*, puis utilise *PXE* pour récupérer une adresse *IP*, une *gateway*, mais également une *IP* de serveur *TFTP*, afin de pouvoir récupérer les images de déploiement. Enfin, le déploiement du système se fait via *iSCSI*[15], qui ne demande pas configuration particulière de *Glance* pour fonctionner.

En ce qui concerne *Neutron*, les interactions sont particulières, par rapport à des *VMs*. Historiquement, *Ironic* ne supportait que des *networks* de *type* « *flat* ». Avec *Pike*, il est possible d'interfacer *Ironic* et *Neutron* pour qu'*Ironic* configure les interfaces virtuelles des machines physiques en passant par *Neutron*. Cela demande le mise en place de deux *networks* différents. La machine physique est démarrée dans un *network* de « *provisionnement* », puis basculée dans un *network* de « fonctionnement ».

6.2.3 *Debug* d'*Ironic*

Une fois *Ironic* installé, j'ai tenté de démarrer une machine physique. Cela a échoué ; d'après *Nova*, cela est dû au fait que « *Nova Scheduler* n'a pas trouvé de *cell* éligible pour démarrer le nœud ». Malheureusement, je suis arrivé à cours de temps à ce moment-là,et n'ait pas pu résoudre ce problème.

13 L'image contient un système complet, avec *initramfs*, *kernel*, et partitions inclus.

14 L'image contient uniquement la partition *root*.

15 *Internet Small Computer Systems Interface*, un protocole qui permet de faire transiter des commandes *SCSI* à travers un réseau. Le protocole *SCSI* est utilisé pour effectuer des transferts de données entre un équipement et un périphérique (exemple : un disque dur).

7 Résultats

Malgré le fait que je n'aie pas réussi à faire fonctionner *Ironic*, ce que j'en ai vu m'a plutôt semblé positif. Bien qu'il soit plus conçu pour déployer des *clusters* de calcul intensif, il pourrait être utilisé afin d'avoir plusieurs *OS* sur une même machine.

Muni de l'ensemble des images nécessaires pour faire démarrer une machine, on peut évaluer la possibilité de faire démarrer 40 machines en même temps dans les salles TPR, au travers d'un exemple concret : pour une image *Ubuntu*, on a 1 *GB* d'images à transférer (*deploy ramdisk*, *deploy kernel* et système complet). Cela représente 40 *GB* d'images à transférer depuis *CEPH*. Dans les salles TPR, une ligne à 1 *GB* arrive et se divise en plusieurs lignes 100 *MB* vers chacun des *PC*. Un tel transfert peut s'effectuer en un peu plus de 5 minutes (on ignore les paramètres suivants : traitements internes dans *OpenStack*, délai de transmission au niveau de la machine *CEPH*, délai de transmission / traitement au niveau des *switchs* / routeurs). On peut raisonnablement envisager une borne supérieure à 10 minutes. C'est un temps un petit peu élevé, mais si l'on considère que les machines peuvent être lancées à distance par l'enseignant avant le début du TP, ou dans la pause entre deux TPs, c'est raisonnable.

En revanche, ce qui concerne la gestion des sauvegardes est plus complexe: *aufs* permet de sauver absolument tout, y compris le contenu modifié de /usr, /bin, /etc, /lib… Les volumes *Cinder* sont avant tout de l'espace de stockage supplémentaire. Si un étudiant est amené à installer des services ou des packages, ou plus généralement à effectuer une modification sur la partition principale, comment la sauvegarder ?

8 Conclusion

L'objectif de ce POM était d'étudier s'il est possible d'utiliser *OpenStack Ironic* en remplacement du système actuellement présent dans les salles TPR. Pour ce faire, un *cluster* minimaliste a été mis en place. Le travail s'est avéré plus important que prévu par mes encadants au départ, mais cela m'a permis de comprendre les concepts et le fonction d'*Ironic*. Plusieurs points vont devoir faire l'objet d'une étude future :

- La nécessité ou non d'utiliser des cartes réseaux afin d'effectuer du o*ut-of-Band Management* ; cela sera indispensable à partir d'*OpenStack Queens*, car le *driver* actuellement utilisé ne fonctionnera plus, et il sera obligatoire de passer par du *out-of-band management*.
- Il existe plusieurs modes de démarrage, et il est par ailleurs possible de rendre un boot permanent. Il serait intéressant de voir s'il est possible de mettre un système de façon « permanente » et de temps en temps passer sur un système spécifique.
- Est-il possible d'utiliser le boot diskless pour continuer un même TP sur une machine différente ?
- En ce qui concerne la gestion des sauvegardes, est-il possible de monter un volume Cinder en tant que /usr ou /lib par exemple, pour permettre de simuler le système de sauvegarde dans ces dossiers ?

Dans tous les cas, le *cluster* actuel est en place et pourra être utilisé pour poursuivre l'étude, et évaluer la possibilité de fonctionnement de ce système.

Ce projet m'aura appris énormément de choses différentes, pas nécessairement en lien avec le *cloud-computing*, comme *aufs*, la configuration de proxy, le fonctionnement des *VLAN*, la mise en place de *cache*, la configuration d'un serveur *DHCP*, *TFTP* etc. J'aurai également découvert le fonctionnement d'*OpenStack*, que je

n'avais jusque là utilisé qu'en tant qu'étudiant. J'ai pu prendre conscience de ce qu'un tel environnement peut apporter et en quoi le *cloud-computing* peut être utile.

9 Références

[Cep18a] Ceph Homepage https://ceph.com/ (dernière visite : 21 mai 2018)

[Cep18b] Ceph Deployment http://docs.ceph.com/docs/master/rados/deployment/ (dernière visite : 21 mai 2018)

[Neu18] Neutron : Installation Guide https://docs.openstack.org/neutron/pike/install/index.html (dernière visite : 21 mai 2018)

[Ope18a] OpenStack https://www.openstack.org/ (dernière visite : 21 mai 2018)

[Ope18b] OpenStack Installation Guide https://docs.openstack.org/install-guide/ (dernière visite : 21 mai 2018)

[Squ18] squid : Optimising Web Delivery http://www.squid-cache.org/ (dernière visite : 21 mai 2018)

10Annexes

10.1 Informations techniques

10.1.1 aufs

aufs est un système de fichier sous UNIX, signifiant originellement « *Another UnionFS* », puis transformé en « *Advanced multi-layered unification filesystem* ». Le principe d'*aufs* est de permettre de sommer des répertoires en un unique répertoire. Ce processus est appelé *union mount*. Les répertoires impliqués dans la somme sont appelés « branches ». Pour chaque branche, il est possible de choisir si elle sera montée en « écriture » ou en « lecture ». Un exemple pour visualiser est disponible ici.

10.1.2 Système de Mr EXCOFFIER

Le système mis en place par Mr EXCOFFIER apporte deux modifications au script *init* du système Debian sur les machines des salles TPR.

- En premier lieu, le disque dur est monté sous le nom /rootR0. Ce répertoire n'est pas destiné à être modifié par l'utilisateur ;
- La seconde modification est « l'interruption » temporaire du déroulé d'*init* pour amener l'utilisateur dans une console où il peut choisir un *template* de TP à télécharger, ou une sauvegarde qui a précédemment été effectuée sur la machine. Les *templates* sont des fichiers *tar.gz* qui contiennent tous les fichiers modifiés pour créer l'environnement de TP ; une fois un nom donné à la nouvelle sauvegarde, le *.tar.gz* est décompressé dans /sauvegarde/<nom_sauvegarde>.

Une fois que l'utilisateur a effectué son choix, *init* reprend son déroulé. Le dossier correspondant à la sauvegarde choisie (qu'elle soit ancienne ou nouvellement créée) est sommé avec /rootR0. Le dossier de sauvegarde est ajouté avec possibilité d'écriture alors que /rootR0 n'est ajouté qu'en lecture. Ainsi, si un étudiant installe un package, et que cela apporte des modifications dans /usr, /lib… Les modifications seront reportées dans la branche /sauvegarde, et non dans la branche /rootR0. Une fois la machine éteinte, /sauvegarde est conservé pour un certain temps.

10.1.3 Object Storage

Dans le stockage objet, les données sont représentées par des objets, qui contiennent les données elles-même, mais également un certain nombre de métadonnées en plus. Cette forme de stockage apporte une très grande *scalabilité*, mais également de bons résultats en matière de recherche, au travers des métadonnées.

La *scalabilité* est ici liée au fonctionnement même du stockage objet. Là où un stockage de type *file system* est limité par le volume sur lequel il se trouve, de par sa nature hiérarchique, un stockage objet considère l'ensemble des nœuds sur lesquels il est déployé comme un espace unifié, potentiellement extensible. Chaque objet présent dans cet espace possède un identifiant unique, mais également de nombreuses métadonnées qui, contrairement au stockage dans un *file system*, peuvent être définies par l'utilisateur. Cela permet d'effectuer des recherches de façon plus simple que dans un *file system*.

Pour effectuer une recherche dans un *file system*, il faut parcourir la hiérarchie, et regarder le contenu des fichiers pour trouver ce que l'on cherche. Une solution peut être de placer en tête de fichier les informations qui peuvent servir à l'identifier, mais cela demande un effort conscient de l'utilisateur, et ne peut pas toujours être facilement effectué. Dans un stockage objet, les métadonnées peuvent être ajoutées à la demande, de

façon simple pour l'utilisateur. Une recherche sur les métadonnées évite de devoir parcourir toutes les données elles-mêmes.

10.1.4 Fonctionnement d'*OpenStack Neutron*

Neutron permet de créer des réseaux virtuels et de donner un accès à ces réseaux aux *VMs*. On distingue deux types de réseaux virtuels :

- Les *provider networks* qui sont la forme la plus simple de réseau virtuel ;
- Les *self-service networks* qui sont plus techniques, mais offrent plus de possibilités, comme la configuration de pare-feu ou la mise en place de *load-balancing*.

Neutron permet ensuite de créer des *subnets* dans les réseaux virtuels. Tout comme en réseaux classiques, un *subnet* représente une plage d'adresses *IP* qui peuvent être attribuées à un ensemble de machines. Un même réseau virtuel peut posséder plusieurs *subnets*. Il est important de noter qu'une *VM* (ou une machine *baremetal* dans le cas d'*Ironic*) est placée dans un *subnet* par *Neutron* lui-même. A la création, on indique simplement le *network* dans lequel on veut la placer, *Neutron* détermine ensuite de lui-même dans quel *subnet* il va la placer.

L'implémentation de réseaux virtuels se fait au travers de *mechanisms*. Un *mechanism* est une méthode de création de réseau virtuel par-dessus une interface physique. Par exemple, le *mechanism linuxbridge* crée un pont virtuel entre une interface physique (qui sert de support pour un *network type*) et une interface virtuelle qui permet d'accéder à la *VM* (dans le cas particulier du *network type VLAN*, le pont est créé entre l'interface virtuelle correspondant au *VLAN* créé sur l'interface physique, et l'interface virtuelle permettant d'accéder à la *VM*). On peut aussi trouver le *mechanism OpenVSwitch*, qui représente un switch virtuel

Une fois le *mechanism* sélectionné, le réseau possédera un certain *type*, par exemple *flat* (le réseau virtuel est accédé à partir d'une interface physique, le réseau local agit en *overlay* par dessus), ou *VLAN* (le réseau virtuel est accédé à partir d'une interface virtuelle taggée 802.1q, créée à partir d'une interface physique).

10.2 Fonctionnement d'*OpenStack Ironic*

10.2.1 ironic-conductor

Il est nécessaire de fournir une représentation des nœuds physiques à *Ironic*. C'est ce qu'on appelle un *baremetal node* dans la terminologie d'*OpenStack*. Un *baremetal node* contient les informations relatives à une certaine machine : nombre de *CPU*, quantité de *RAM*, espace disque, mais également quels *drivers* sont utilisables pour démarrer cette machine.

La gestion des *baremetal node* se fait au travers du service *ironic-conductor*. Un *ironic-conductor* administre plusieurs *nodes* (la répartition des *nodes* à travers les différents *conductors* du cluster se fait via un *consistent hash ring*).

Le rôle du *conductor* est de fournir les informations nécessaires à l'*ironic-python-agent*, un service lancé dans la première phase de *boot* des nœuds *baremetal*, pour lui indiquer comment récupérer les images nécessaires pour démarrer le nœud par la suite.

10.2.2 Images

Lors d'un déploiement *Ironic*, trois à cinq images sont mises en jeu, en fonction du type de déploiement que l'on effectue. Ces deux types de déploiement sont :

- Le déploiement *whole disk*. Ici, *Ironic* utilise une seule image pour fournir l'intégralité du système. Cette image représente donc un disque entièrement partitionné, et contient un *kernel*, un *initramfs* et le système en lui-même. Il est possible d'ajouter *grub2* à ce type d'image pour rendre son déploiement permanent : dans ce cas, *GRUB* est installé sur le disque dur et agit en *bootloader* sur les *reboot* successifs, après le premier déploiement ;
- Le déploiement par *partition image*. Il est nécessaire ici de stocker trois images différentes : l'image système, l'image du *kernel* et l'image de l'*initramfs*. L'image système reçoit en métadonnée les références *Glance* des images *kernel* et *initramfs* auxquelles elle est associée. Ce type d'image ne supporte que les systèmes *GNU/Linux*.

Les deux autres images, que l'on retrouve dans les deux modes de déploiement, sont les images *deploy*. Elles représentent un petit système d'exploitation (plus précisément un *initramfs* et un *kernel* minimalistes) qui vont permettre à *Ironic* d'effectuer du *in-band management*. Cela se fait au travers du sevice *ironic-python-agent* qui est lancé par ce petit système. L'*agent* contacte l'*ironic-conductor* responsable du nœud *baremetal* sur lequel l'*agent* tourne, et obtient les informations nécessaires pour continuer le démarrage. Ces informations correspondent aux drivers qui ont été indiqués comme utilisables par le nœud *baremetal* pour chaque étape de la phase de *boot*.

10.2.3 Drivers

Un *driver Ironic* est un outil chargé d'une partie du processus de démarrage et de provisionnement d'un nœud *baremetal*. Il existe des *drivers* permettant d'indiquer quel protocole utiliser pour démarrer la machine (*Wake-On-Lan*, *IPMI*, *Redfish*, *Ilo*, etc.), quel protocole utiliser pour récupérer le *deploy ramdisk* (*PXE + TFTP*, *iPXE + HTTP*, etc.), quel protocole utiliser pour récupérer l'image système (*iSCSI*, téléchargement direct par le *deploy ramdisk*, etc.).

Dans *OpenStack Ocata* (15.0) et versions antérieures, *Ironic* offrait des *drivers* « classiques » qui s'occupaient de chaque phase de boot par eux-mêmes (c'est à dire qu'un seul *driver* pouvait gérer toutes les phases par lui-même). Depuis *OpenStack Pike* (16.0), les *drivers* « classiques » sont accompagnés d'*hardware types*, qui permettent de spécifier pour chaque étape de boot un processus ou un autre pour réaliser l'étape. A partir d'*OpenStack Queens* (17.0), les *drivers* « classiques » ne seront plus supportés par *Ironic*, et il faudra utiliser impérativement les *hardware types*.

10.3 Images

10.3.1 Démonstration d'*aufs*

```
root@amaury-5:/tmp# mkdir aufs
root@amaury-5:/tmp# cd aufs/
root@amaury-5:/tmp/aufs# mkdir -p branch1 branch2/usr
root@amaury-5:/tmp/aufs# ls -R
.:
branch1  branch2

./branch1:

./branch2:
usr

./branch2/usr:
root@amaury-5:/tmp/aufs# echo "branch2/usr" > branch2/usr/toto
root@amaury-5:/tmp/aufs# mkdir sum
root@amaury-5:/tmp/aufs# mount -t aufs -o br=/tmp/aufs/branch1:/tmp/aufs/branch2 -o udba=reval none /tmp/aufs/sum/
root@amaury-5:/tmp/aufs# ls -R
.:
branch1  branch2  sum

./branch1:

./branch2:
usr

./branch2/usr:
toto

./sum:
usr

./sum/usr:
toto
root@amaury-5:/tmp/aufs# echo "Seulement dans la branche 1" >> sum/usr/toto
root@amaury-5:/tmp/aufs# cat branch1/
usr/          .wh..wh.aufs  .wh..wh.orph/  .wh..wh.plnk/
root@amaury-5:/tmp/aufs# cat branch1/usr/toto
branch2/usr
Seulement dans la branche 1
root@amaury-5:/tmp/aufs# cat branch2/usr/toto
branch2/usr
root@amaury-5:/tmp/aufs#
```

*Illustration 1: Exemple d'illustration d'*aufs

On peut observer que le texte « Seulement dans la branche 1 » a été ajouté dans la branche « /tmp/aufs/branch1 », alors qu'il a visiblement été ajouté dans un fichier qui n'existe que dans la branche « /tmp/aufs/branch2 ». La hiérarchie complète a été recréée dans la branche montée avec accès en écriture. La branche montée avec accès en lecture seule n'a pas été modifiée. Dans le cadre des salles TPR, /tmp/aufs/branch1 serait l'équivalent de /sauvegarde/une_sauvegarde, et /tmp/aufs/branch2 serait /rootR0.

10.3.2 Illustration du démarrage d'une machine des salles TPR

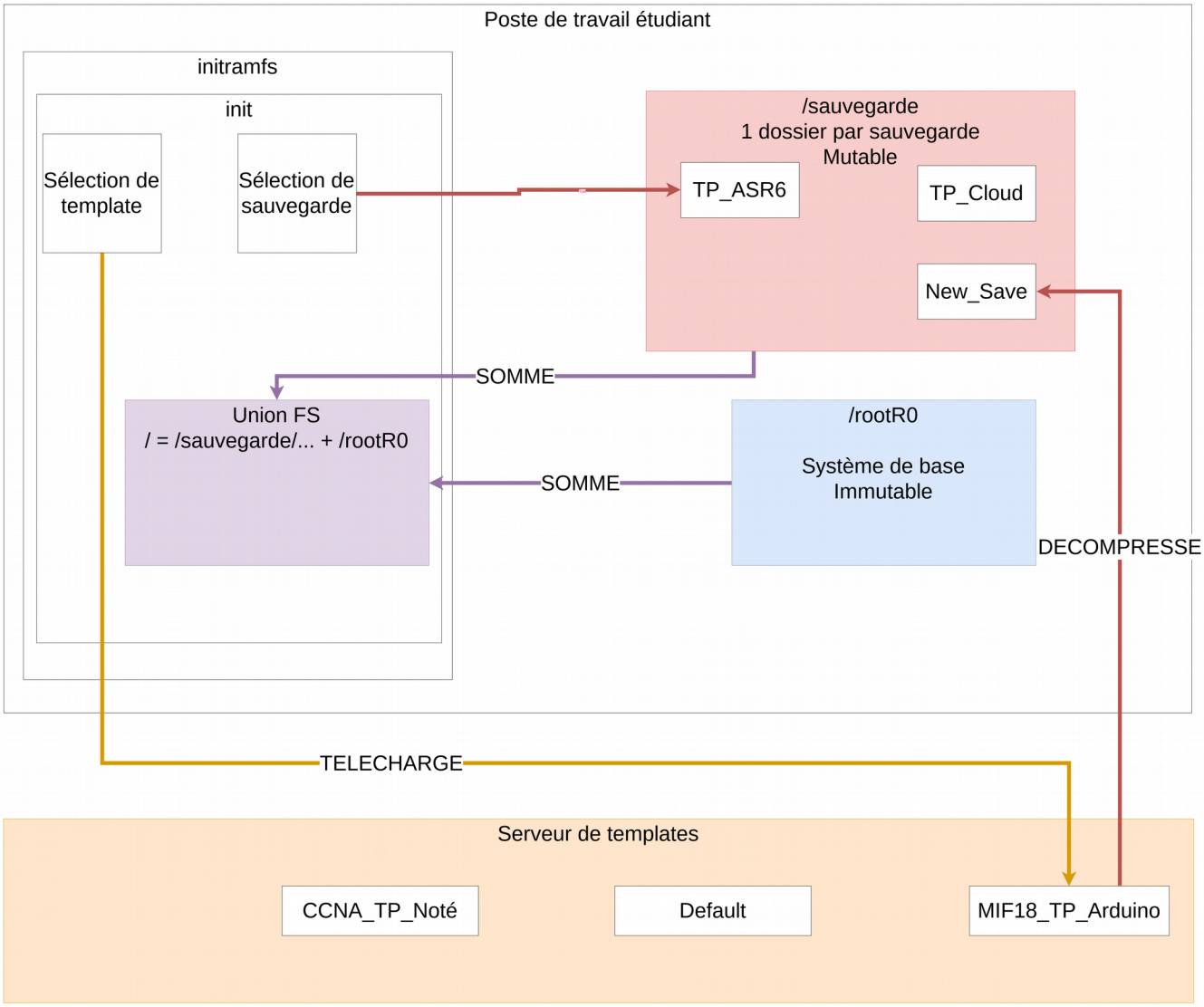

Illustration 2: Illustration du démarrage d'une machine des salles TPR

10.3.3 Illustration des relations dans un *cluster OpenStack* minimaliste

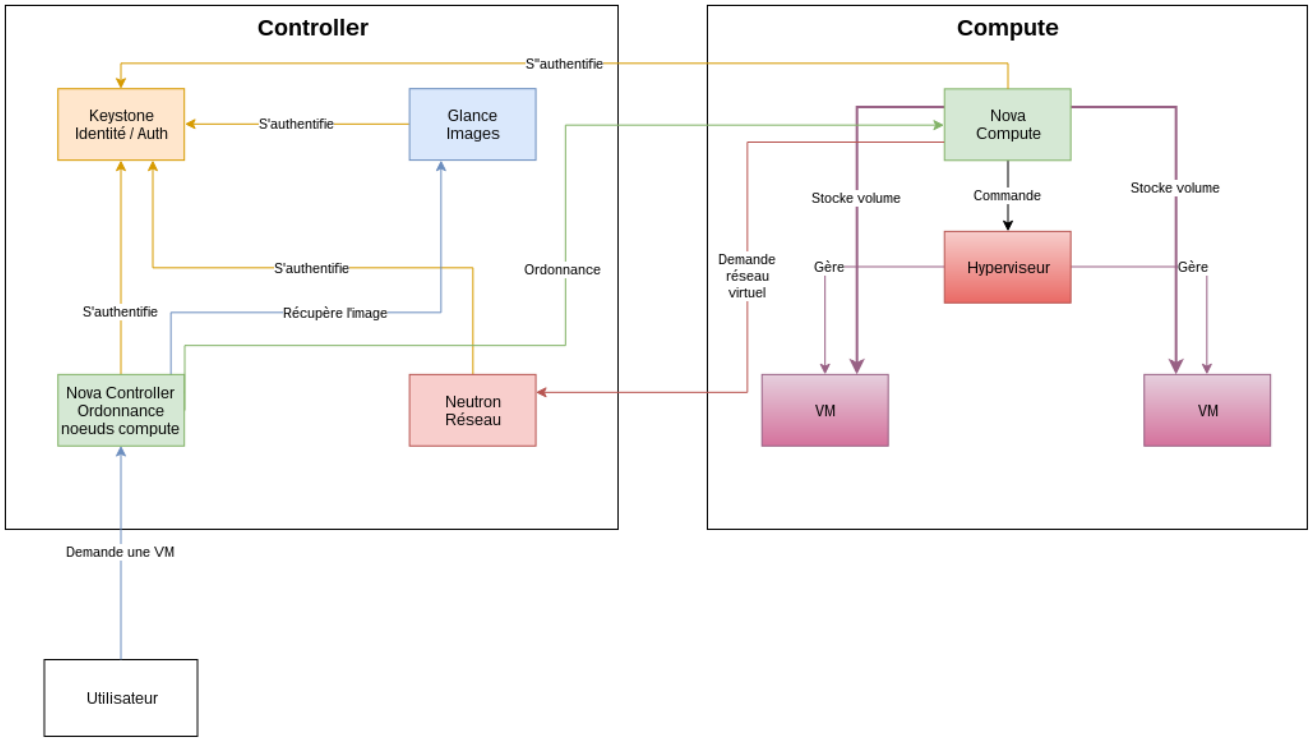

Illustration 3: Illustration des relations dans un cluster OpenStack *minimaliste*

10.3.4 Illustration des relations CEPH / OpenStack

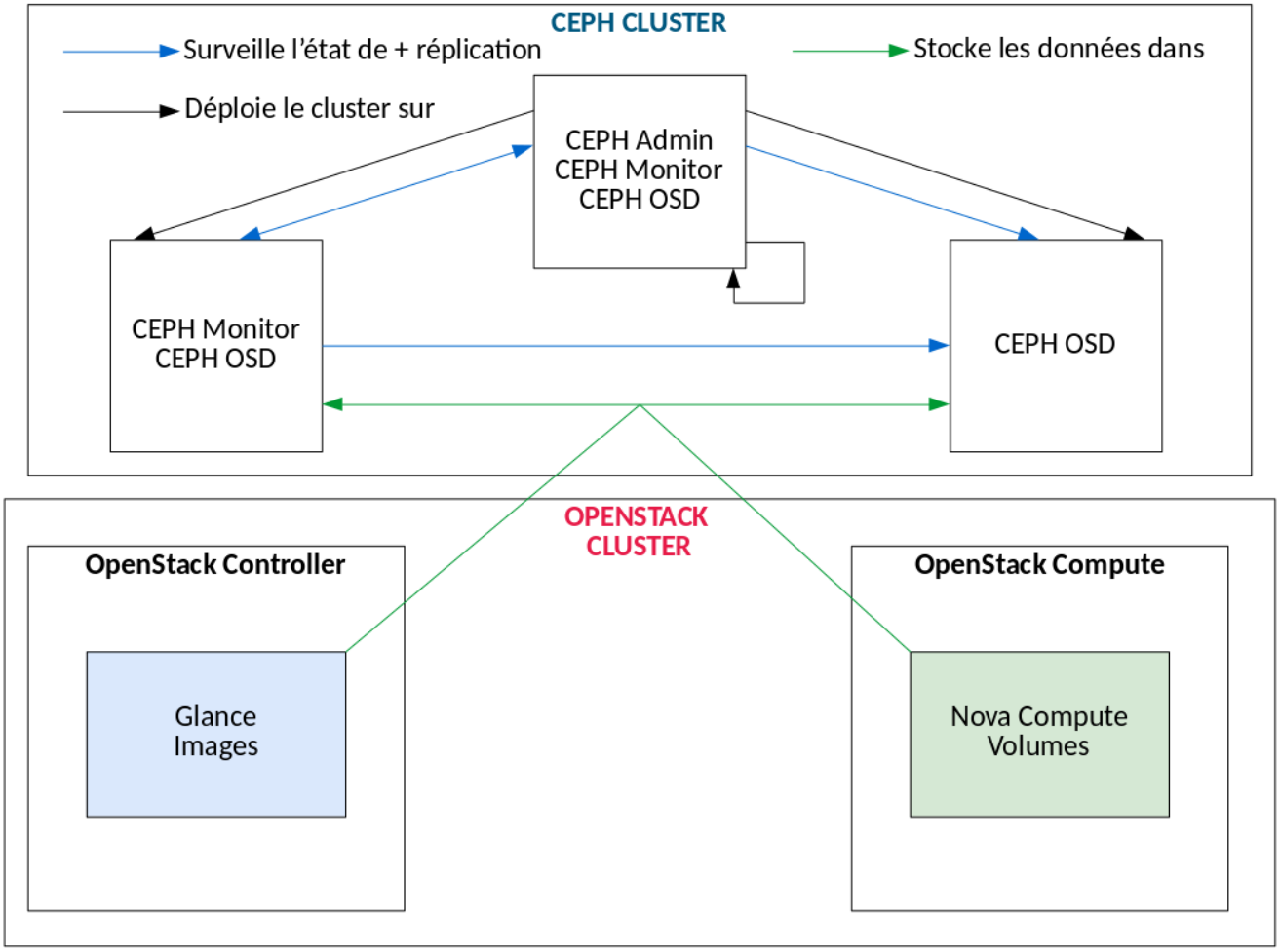

Illustration 4: Illustration des échanges entre CEPH *et* OpenStack