ÉCOLE POLYTECHNIQUE
DE L'UNIVERSITE FRANÇOIS RABELAIS DE TOURS
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Rapport de projet de fin d'études 2015

Un datacenter dans le cloud Public/HA IaaS

Encadrants
Sébastien AUPETIT
sebastien.aupetit@univ-tours.fr

Maître de conférences

Étudiants
Quentin MACHU
me@quentin-machu.fr

DI5 2014 - 2015

Version du 8 mai 2015

Table des matières

Table des figures

Liste des tableaux

Introduction

1.1 Qu'est ce que le cloud-computing ?

Tant le terme cloud-computing, et plus particulièrement son abréviation « cloud » est devenu au cours des dernières années un terme galvaudé et marketing, il paraît crucial d'en définir ici une définition claire. Le *National Institute of Standards and Technology (NIST)* définit le cloud-computing par « Le cloud computing est un modèle économique 'pay per use' pour accéder et utiliser, à travers du réseau, à un ensemble de ressources informatiques configurables et partagées (i.e. réseaux, serveurs, stockage, applications, etc.) qui peuvent être rapidement achetées et utilisées avec un effort de gestion minimal et une interaction limitée avec le fournisseur de services » [44].

Le cloud-computing est le fruit de l'évolution des technologies informatique et d'un changement de paradigme. L'industrie informatique est généralement basée sur une modèle d'investissement en infrastructure matérielle/logicielle (CAPEX), possédée, gérée et maintenue par le consommateur. Sous la pression de la situation économique globale et du marché réduisant les temps de cycles, les entreprises veulent désormais ce concentrer directement sur leurs cœurs de métiers, en n'étant plus limité par les coûts opérationnels, par les frais d'investissements et le temps nécessaire aux différentes évolutions nécessaires à leurs épanouissements. C'est pour ces raisons de coûts, de simplicité de déploiement / d'utilisation et de besoin d'agilité que l'industrie commence à externaliser ses infrastructures et à en déléguer la gestion. Les entreprises ont tendance à ne plus consommer l'informatique en tant que produit mais en tant que service (OPEX), ne payant alors plus que ce qu'elles consomment réellement, à tout moment, à un prestataire externe. Elles se déchargent alors de l'ensemble des contraintes de financement, des tâches de gestions et de maintenances, à l'instar de l'usage de l'électricité – utilisée et payée en fonction des besoins réels, sans se préoccuper de la façon dont elle est produite ni de la façon dont il est nécessaire de faire évoluer la production.

Cinq caractéristiques clés définissent le cloud-computing bien qu'il en existes de nombreuses autres :

- ressources en self-service : le consommateur peut à tout moment provisionner son infrastructure (en capacité de calcul, de stockage, etc), automatiquement si besoin est, sans nécessité de réaliser une quelconque interaction humaine.
- ouverture : les services sont proposés et mis à dispositions via Internet, en utilisant une interface standardisée et interopérable, permettant au consommateur d'accéder uniformément aux services à partir de différentes plate-formes/supports mais aussi de pouvoir changer de fournisseur aisément.
- mutualisation des ressources :le fournisseur organise des ressources hétérogènes (matérielles, logicielles, réseaux) selon un modèle « multi-tenant » permettant de les assigner

et de les ré-assigner automatiquement en fonction de la demande.
— élasticité rapide : le consommateur peut décider à tout moment d'augmenter (scale-up) ou de réduire (scale-down) la capacité de son infrastructure, éventuellement de façon automatique avec l'aide de l'utilisation de métriques, rapidement pour pouvoir répondre à la demande et de manière paressant illimitée.
— service mesuré : l'utilisation des ressources est contrôlée, mesurée, reportée de façon simple et abstraite pour le fournisseur et le consommateur, permettant notamment de réaliser une facture à l'usage.

Par ailleurs, il existe trois catégories de services offertes par le cloud-computing, qui peuvent être vues comme trois couches se superposant :

— IaaS (Infrastructure as a Service) : ce service consiste à offrir un accès (via Internet) à un parc informatique virtualisé contenant des machines virtuelles, espaces de stockages, réseaux, etc. Ce parc est évolutif et capable de s'adapter (automatiquement ou manuellement) aux besoins du consommateur. Les fournisseurs majeurs aujourd'hui sont Amazon Web Services et Google Compute Engine. En France, nous retrouvons par exemple Cloudwatt.
— PaaS (Platform as a service) : en plus de la couche matérielle gérée par l'IaaS, avec le PaaS, le fournisseur offre et maintient l'environnement système et les outils d'infrastructure tandis que le consommateur est en charge des applications et de l'ajout de ses propres outils. Dans les services PaaS existants, on recense entre autres Heroku, Cloudbees, Engine Yard ou encore Google App Engines.
— SaaS (Software as a service) : c'est la dernière couche pouvant être proposée par le cloud-computing. Dans le SaaS, les applications sont directement mises à dispositions du consommateurs, qui n'a alors qu'à les utiliser. Les travaux de mises à jours et autres incombent directement au fournisseur. Il existe de nombreux exemples de services SaaS existants comme Gmail, Flickr, Github, Dropbox, etc.

1.2 Devenir un acteur IaaS

Aujourd'hui, malgré la demande croissante de l'industrie informatique, l'offre IaaS est sous-développé [38, 37]. De plus, la majorité des offres disponibles sont sous l'influence de l'*USA PATRIOT Act*, ce qui freine sensiblement les entreprises, qui ne veulent évidemment pas faire courir de risques à leurs données [6]. Il en résulte que de nombreuses PME (notamment européennes) continuent alors à investir massivement dans leurs propres infrastructures ou louent des serveurs chez divers prestataires - sans pour autant avoir de réelles garanties de services ni énormément de marges de manœuvres. Par ailleurs, l'offre proposée actuellement par *Harmony-Hosting* est convenable pour de nombreux particuliers et pour les petites entreprises ayant relativement peu de contraintes, mais n'est pas adapté aux vrais attentes du monde professionnel. Ainsi, l'objectif de ce projet est de construire une plate-forme IaaS publique (Voir sous-section 1.2.1), haute-disponibilité (Voir sous-section 1.2.2, évolutive horizontalement (Voir sous-section 1.2.3) et automatisée (Voir sous-section 1.2.4. Chacun des quatre points sont cités sont détaillés ci-dessous.

1.2.1 Plate-forme IaaS publique

La plate-forme à construire se veut suffisamment vaste pour couvrir les différents besoins actuels et futurs des entreprises. Elle doit alors offrir :

- Des instances disposant d'une certaine puissance de calcul, de mémoire vive, d'un espace de stockage éphémère. Elles doivent pouvoir être crées, suspendues, redémarrées, redimensionnées et détruites. Pour la création des instances, différentes images pré-disponibles peuvent être utilisées mais aussi des images construites par le consommateur à partir de snapshots d'instances.
- Des services réseaux tels que l'assignation d'adresses IP privées aux instances, d'adresses IP publiques flottantes permettant l'exposition de l'instance sur le réseau extérieur (ie. Internet), la création de réseaux VLAN (entre les instances). D'autres possibilités comme la création de VPN, de pare-feus, de IDS, de balancements de charges inter-instances sont un plus non négligeables.
- Des services de stockages offrant un support de stockage permanent aux instances. Ces espaces de stockages (type « block ») doit être accessible depuis les instances par le biais de montages. Compte tenu des besoins du marché, il peut être particulièrement intéressant de disposer également d'un support de stockage de type « object » partagé par différentes instances simultanément.
- Une API standardisée permettant de contrôler l'ensemble.
- Des services de monitoring et de télémétries compilant des informations et statistiques importantes sur l'utilisation de la plate-forme pour le fournisseur et sur l'utilisation faite par chaque consommateur à des fins de facturations.

Entre outre, des systèmes d'accès contrôlés et de sécurités doivent être présents à tous les niveaux afin de protéger les consommateurs mais aussi la plate-forme IaaS en tant que telle.

1.2.2 Haute disponibilité

La plate-forme doit également disposer d'un caractère haute-disponibilité. Cela signifie entre autres qu'il ne doit pas y avoir de point unique de défaillance (SPoF). L'ensemble des systèmes de contrôle (à qui incombent les tâches de fonctionnement de l'IaaS) et des données de ces derniers doivent être redondés afin d'assurer une tolérance aux catastrophes de diverses natures (panne matérielle, panne électrique, problème réseau, désastres environnementaux tels les incendies, attaques ciblées). Il est également question de la redondance des données des clients (stockages permanents « block » et « object »). Idéalement, on parle d'un objectif de garantie de service de 99.99 %, ce qui équivaut à moins d'une heure d'indisponibilité par an. La conception doit donc être de telle sorte à ce qu'il n'y ai pas de réelle indisponibilité au-delà d'un éventuel échec sur panne (exemple appel API ou récupération d'une donnée arrivant strictement au moment d'une panne) , une nouvelle tentative dans l'instant qui suit doit en principe fonctionner.

Néanmoins, cela n'inclue pas nécessairement la haute-disponibilité des instances, qui peuvent être rendues indisponibles ou détruites, ainsi que leurs stockages éphémères à la suite d'une panne. Un éventuel balancement de charge inter-instances réalisé par le consommateur peut néanmoins continuer à garantir le service offert par la/les instances rendues

in-opérationnelles. La re-création immédiate de nouvelles instances est également possible, potentiellement automatiquement.

1.2.3 Évolutivité horizontale

Afin de pouvoir répondre à l'augmentation de l'utilisation de la plate-forme IaaS (en nombre de tenant et/ou en ressources), il est nécessaire que la capacité des services de contrôles puisse augmenter horizontalement (par l'ajout de nouveaux nœuds - contrairement à l'augmentation verticale qui consiste en l'amélioration du matériel des noeuds existants). Cela nécessite l'utilisation de balancements de charges afin de répartir les requêtes entre les différentes nœuds capables de traiter. Il en va de même avec les services de stockages dont la capacité totale doit pouvoir augmenter de façon illimitée (espace et traitement : IO/s, débits, etc).

Cette évolutivité horizontale combinée à la nécessité de haute-disponibilité incite donc directement l'utilisation de systèmes redondants de type « actif/actif ».

Les instances devront également pouvoir être redimensionnées. Des fonctions d' « Auto-Scaling » permettant d'ajouter/supprimer automatiquement des instances prédéfinies dans un cluster d'instances en balancement de charges peut être un point intéressant pour le consommateur.

1.2.4 Automatisation

Une telle plate-forme, complexe et potentiellement de taille importante (une dizaine à plusieurs centaines ou milliers de serveurs) se doit d'être la plus automatisée possible afin de limiter la quantité de travail nécéssaire, réduire les risques d'erreurs humaines (erreurs courantes et pouvant impacter la disponibilité) mais aussi garantir l'homogénéité des installations et configurations. Que ce soit pour l'installation, la configuration, les tests (matériels/logiciels) ou les opérations de maintenances, des scripts et/ou des outils adaptés doivent être disponibles.

A titre d'illustration, on imagine l'ajout d'une dizaine de nouveaux serveurs dédiés pour s'adapter à la nouvelle demande en infrastructure de la part de la clientèle ou à un pic de charge. Un technicien installe les nouveaux serveurs dans une des baies libres et les relie au réseau électrique et informatique. Les serveurs sont mis sous tension et démarrent automatiquement sur une image de démarrage spécialement conçue (Utilisation par exemple de DHCP+PXE). Cette image de démarrage installe automatiquement le système d'exploitation utilisé dans l'IaaS et exécute une série de tests matériels. Si le tests ont échoués, une alerte est remontée automatiquement. Dans le cas où les tests ont réussis, un outils configure le serveur dédié sans intervention humaine, installe des services sur celui-ci en fonction des besoins de l'IaaS (annoncés dans une base de données), vérifie le bon fonctionnement des services et l'intègre dans le cloud. Dans le cas où nous aurions besoins de modifier la répartition de services, on peut également imaginer des scripts capables de convertir un serveur dédié exécutant des services A en un serveur dédié exécutant des services B, après avoir

délégué ses précédentes tâches à un autre serveur. Ce ne sont que de simples exemples parmi les dizaines où l'utilisation de scripts d'automatisation peut s'avérer très utile.

La réalisation d'automatismes est une tâche moins prioritaire mais la réalisation de quelques scripts (de tests, voir de déploiement) est souhaitée dans le cadre de ce projet.

1.2.5 Budget et vision à long-terme

Idéalement, la plate-forme doit pouvoir être déployée en réalisant le minimum d'investissement. Les décisions seront alors prises dans l'optique d'être en mesure de mettre en œuvre un maximum de fonctionnalités, dans les meilleures conditions de fonctionnement et de fiabilité possibles, avec un coût minimal. Il s'agit d'assurer que les fonctionnalités les plus importantes soient disponibles immédiatement mais aussi d'être en mesure de faire évoluer les services par la suite. Il parait donc important que les technologies utilisées par la plateforme soient suffisamment modulables/extensibles afin qu'elles ne limitent pas l'avenir. De part ces contraintes budgétaires, la taille de l'infrastructure sera faible au départ et devra pouvoir grandir aisément. C'est à dire sans nécessité d'effectuer des modifications architecturales et/ou logicielles importantes qui pourraient influencer le fonctionnement et la disponibilité de l'IaaS.

On s'intéresse ainsi à réaliser une infrastructure modulaire étant capable de s'adapter rapidement à différentes conditions de charges mais aussi à différentes répartitions d'utilisations des services.

Dans un premier temps, l'intégration de la plate-forme dans plusieurs datacenter (géographiquement distants) via l'installation de serveurs en « Housing » ou via location de serveurs est une piste envisageable dans ce cadre si les conditions techniques le permettent.

1.3 Objectifs

Le tableau 1.1 explicite les objectifs décris ci-dessus de façon synthétique et en les classant par ordre de priorité décroissante (1 représente la plus haute importance).

Référence	Priorité	Dénomination
O00	0	Fonctionnement de la plateforme sur du matériel économique & d'usage courant
O01	1	Mise à disposition d'instances virtuelles (sur stockage éphémère)
O02	1	Capacités réseau basiques (IP privées, isolation VLAN)
O03	1	Accès à une API standardisée pour contrôler ses services
O04	1	Haute disponibilité des contrôleurs
O05	1	Evolutivité horizontale des contrôleurs
O06	1	Mise en place d'une plate-forme de démonstration fonctionnelle
O07	2	Stockage de volumes sur les instances virtuelles
O08	2	Haute disponibilité des volumes
O09	2	Evolutivité horizontale de l'espace de stockage des volumes
O10	3	Stockage d'objets
O11	3	Haute disponibilité des objets
O12	3	Evolutivité horizontale de l'espace de stockage des objets
O13	4	Automatisation de déploiements
O14	4	Automatisation de tests
O15	5	Balancements de charges entre instances virtuelles
O16	5	Snapshots et Création d'images d'instances virtuelles personnalisées par les utilisateurs
O17	5	Capacités réseau avancées (IP flottantes, VPN, pare-feus, IDS)
O18	6	Haute disponibilité des instances (re-création automatique)
O19	6	Services de télémétries et de monitoring
O20	6	Clusters d'instances virtuelles avec Auto-Scaling

TABLE 1.1 – Objectifs du projet de fin d'étude (priorité décroissante)

Solutions existantes

Dans le domaine du cloud-computing, il existe différentes solutions Open-Source et propriétaires permettant de gérer des plateformes IaaS, PaaS et parfois même SaaS. Chacunes sont à des stades de développement, d'adoption et de maturité différentes et proposent des fonctionnalités plus ou moins avancées et documentées. Cette section vise à les comparer en vu d'en choisir une.

Dans le cadre de ce projet, nous ne considérons uniquement que les solutions Open-Source étant donné que nous ne disposons pas du budget nécessaire à l'achat de licences. Nous prenons aussi la liberté d'exclure certaines solutions dont la documentation est proche du néant, dont la communauté/l'activité de développement est extrêmement faible ou dont la stabilité et les fonctionnalités sont clairement inadaptés à l'usage professionnel et potentiellement à grande échelle que nous prévoyons (comme OpenQRM).

Suite une phase de recherche initiale, il en ressort que nous avons quatre solutions Open-Source viables que nous allons comparer sur différents aspects.

2.1 Architectures & Services

Toutes les plateformes IaaS reposent sur un ensemble de besoins communs, toutes les solutions disposent donc de services similaires dans leurs rôles mais différents dans leurs implémentations et dans l'organisation architecturale de ces derniers. On retient par exemple les services suivants :

- Services API : reçoivent l'ensemble des requêtes d'intéractions avec l'IaaS et se chargent de la distribution de celles-ci aux services concernés.
- Ordonnanceur : organise la répartition des ressources (telles que les instances) à travers la plateforme
- Services de virtualisation : exécutent et gèrent les instances (machines virtuelles) en utilisant différents hyperviseurs
- Service d'authentification : assure la sécurité pour l'ensemble de la plate-forme et des différents services
- Service d'images : distribue les images permettant la création des instances à travers l'IaaS. Le stockage est parfois assuré par les services de stockages
- Services de stockages : stockent de manière sécurisée les volumes et données des instances
- Services de réseaux : offre les capacités réseaux aux machines virtuelles, dont routeurs virtuels, VLAN, adresses IPs flottantes publiques, etc
- Services de télémétrie et de monitoring : réalisent des activités de contrôle, de facturation, de réalisations statistiques
- Service de bases de données : enregistre l'ensemble des données liées à l'IaaS et aux

services

On dispose aussi souvent de services de balancements de charges et d'interface web. Le stockage est généralement délégué à des solutions externes.

Nous nommons d'un terme générique «Contrôleurs» le regroupement de différents services (pouvant être de différentes natures) permettant de gérer la plateforme. Une panne de ces derniers n'entrainent pas nécessairement une indisponibilité des instances du consommateur mais l'impossibilité pour ce dernier de les gérer : d'en créer de nouvelles, d'en modifier les caractéristiques, de les supprimer, etc. Certaines solutions (architecture «Monolithique») regroupent les services ensembles - ils sont installés comme un seul et même logiciel tandis que certaines solutions permettent la séparation de chacun des services en plusieurs logiciels inter-dépendants (architecture modulaire), ce qui accroit la flexibilité mais aussi la complexité. Les contrôleurs peuvent (et doivent) être rendus hautement disponibles et certains peuvent même supporter le balancement de charge (répartition des requêtes sur plusieurs contrôleurs). On peut alors considérer deux types de haute disponibilité :

— Actif/Passif : Un seul contrôleur est actif à un instant t, recevant et gérant les diverses requêtes. Si ce contrôleur venait à subir une défaillance, un des contrôleurs passifs prendrait la main et commencerait à traiter les requêtes.
— Actif/Actif : Plusieurs contrôleurs se voient répartir les requêtes. Les serveurs défaillant sortent du balancement de charge. Cela pré-suppose que les requêtes puissent être traitées indépendant des autres et que les contrôleurs ne gardent pas d'état («Stateless»).

Le type Actif/Actif est bien plus intéressant dans le sens où il permet d'augmenter la capacité de traitement de la plateforme par l'ajout de nouveaux contrôleurs (évolution horizontale) et non par évolution du matériel existant (évolution verticale). On supprime également le risque que les contrôleurs passifs ne soient pas fonctionnels (défaut quelconque) lorsque nous avons besoin d'eux, en Actif/Actif, on le constate immédiatement. On évite aussi de disposer de matériel "inutile"/froid à tout instant.

Quatre grandes catégories de déploiement de plateformes IaaS (et plus généralement de cloud) existent [44], dans la pratique, trois sont considérées :

— Public : L'infrastructure est mise à disposition du grand publique et utilisable par différentes organisations, entreprises, académies, gouvernement ou personnes.
— Private : L'infrastructure est mise à disposition d'une seule organisation, potentiellement composée de plusieurs consommateurs internes ou tiers. Elle est la propriété de cette organisation.
— Hybrid L'infrastructure est composée d'une part d'infrastructure(s) publique(s) et d'une part d'infrastructure(s) privée(s). Cela permet par exemple aux entreprises d'adapter leur infrastructure plus rapidement et à moins coûts en agrandissant simplement leur infrastructure privée sur une infrastructure publique externe.

Comme l'indique le titre de ce projet, seul l'infrastructure publique nous intéresse ici.

Le tableau 2.1 différencie les solutions selon les caractéristiques ci-dessus.

		OpenStack	Eucalyptus	CloudStack	OpenNebula
Type de Cloud		Pu/Hy/Pr	Pr	Pu/Hy/Pr	Pu/Hy/**Pr**
Architecture		Modulaire	5 Composants	Monolith.	Monolith.
Contrôleurs	Haute-Disponibilité	✓	✓	✓	✓
	Balancement de charge	✓		✓	

TABLE 2.1 – Type de Cloud et Architecture

	OpenStack	Eucalyptus	CloudStack	OpenNebula
Compatibilité	Local, LVM, NFS, ZFS, GlusterFS, Ceph, iSCSI, NetApp, Nexenta, SolidFire, Zadara, Fibre Channel, etc.	Local, DAS, Dell EqualLogic, NetApp, EMC VNX	Local, NFS, iSCSI, Fibre Channel, Ceph	NFS, Lustre, GlusterFS, ZFS, GPFS, MooseFS, LVM, iSCSI, VMFS, Ceph
Haute-Disponibilité	Cinder+Ceph, Swift+Ceph, Swift, sol. propriétaires	Dell EqualLogic, NetApp, EMC VNX, DRBD	DRBD, Ceph	DRBD, Ceph
Balancement de charge	Cinder+Ceph, Swift+Ceph, Swift,sol. propriétaires	Dell EqualLogic, NetApp, EMC VNX	Ceph	Ceph
Evolutivité	Cinder+Ceph, Swift+Ceph, Cinder, Swift, sol. propriétaires	Dell EqualLogic, NetApp, EMC VNX	Ceph	Ceph
Stockage d'objets	Swift+Ceph, Swift	Walrus		

TABLE 2.2 – Comparaison des supports de stockage pour les solutions IaaS

2.2 Stockage

Le stockage est un élément crucial des plateformes Cloud et souvent un goulot d'étranglement de part les grands besoins en volumes de données et en performances (en soi ou bien les réseaux sur lesquels transitent les données). De plus, l'aspect mécanique des supports de stockages tend à générer des pannes couramment. Les solutions sont nombreuses, parfois propriétaires, et surtout couteuses. Le tableau 2.2 recense les différents supports de stockage supportés par les solutions IaaS considérées.

OpenStack

La gestion du stockage est intégré à OpenStack et se base sur deux composants : Cinder permettant la gestion des volumes et Swift permettant le stockage d'objets. Les instances ont par défaut un stockage éphémère (détruit à la destruction de l'instance) auquel il est possible d'ajouter un volume de stockage persistent via Cinder.

Cinder supporte un très grand nombre de back-ends de stockage comme LVM, NFS, ZFS, GlusterFS, Ceph, l'iSCSI ou encore diverses solutions propriétaires comme NetApp, Nexenta, SolidFire, et Zadara. Il est possible d'en utiliser plusieurs à la fois. L'évolutivité est liée au back-end choisi. Afin de rendre Cinder hautement-disponible et à balancement de charge, il est d'usage de le coupler avec Ceph RBD [1].

Swift est un système autonome qui dispose de son propre système de haute-disponibilité et de balancement de charge, il remplit donc nativement ces fonctions. De part la nature du stockage d'objet, il est bien entendu évolutif. Il est cependant possible d'utiliser Ceph pour Swift de façon à uniformiser les deux formes de stockage (volume, objets) et rassembler en une seule les deux infrastructures matérielles requises - réduisant alors les coûts et la maintenance. Ce rassemblement présente également un avantage notable en performances en rendant possible l'exploitation du «copy-on-write» permettant de réaliser des démarrages d'instances très rapides.

Eucalyptus

Le contrôleur de stockage Eucalyptus gère 5 back-ends de stockage dont 2 non hautement disponibles : «Overlay» qui utilise directement un dossier du système de fichier local et «Direct Attached Storage» qui utilise un périphérique de stockage directement (comme */dev/sdb* ou un groupe de volumes LVM, un montage NFS, etc). Pour le stockage hautement-disponible, Eucalyptus propose donc 3 back-ends mais tous utilisant le même adaptateur : le SAN. Il supporte les périphériques suivants :

— Dell EqualLogic, séries PS4000 et séries PS6000 series
— NetApp, séries FAS2000 et séries FAS6000
— Séries EMC VNX

On note que certaines de ces périphériques peuvent gérer le balancement de charge. L'utilisation de ce matériel propriétaire constitue un énorme désavantage pour nous puisque le coût associé est très important.

Pour le stockage d'objet, c'est le module Walrus qui est utilisé. Encore une fois, les back-ends dépendent de l'utilisation ou non de la haute disponibilité. Il est possible d'utiliser le système de fichier local, un système LVM avec iSCSI, un montage NFS ou bien un «Fiber Channel LUN». Mais en haute-disponibilité, Walrus repose sur l'utilisation de la réplication de données via DRBD pour deux nodes. L'utilisation d'un montage NFS est impossible dans ce cas. L'utilisation de DRBD signifie qu'il n'y a pas de balancement de charge et que les évolutions sont difficiles, on fonctionne avec deux noeuds uniquement : un actif et un passif. De plus, cette infrastructure est forcément dissociée de celle permettant le stockage de volumes.

1. http ://ceph.com

CloudStack

Cloudstack sépare son Cloud en «Zones de disponibilité» (typiquement un datacentre), «Pods» (un rack par exemple) et en «Cluster» (regroupement de serveurs dédiés faisant fonctionner le même type d'instances - même hyperviseur). Cloudstack considére deux types de stockage : un stockage primaire pour chaque «Cluster» et secondaire pour chaque Zone de disponibilité» qui sert pour les images systèmes et les snapshots.

Pour le stockage primaire, CloudStack est prévu pour fonctionner avec un stockage local, NFS, iSCSI ou encore avec le Fibre Channel. CloudStack ne prévoit ni ne propose de solutions pour la haute-disponibilité de ce stockage - la philosophie de CloudStack repose sur l'utilisation de multiples stockages primaires au plus proches des machines virtuelles et sécurisée par des contrôleurs RAIDs et de nombreux disques.

Le stockage secondaire, quant à lui, n'est compatible que avec NFS ou bien avec OpenStack Swift (ce qui est relativement paradoxal). La mise en haute-disponibilité est possible via DRBD+Heartbeat par exemple mais le balancement de charge est impossible et l'évolutivité semble difficile. Il est cependant possible de créer plusieurs stockages secondaires pour permettre l'évolutivité.

Il est néanmoins possible depuis la version 4.0 de CloudStack d'utiliser Ceph comme back-end pour le stockage primaire - permettant alors de réduire les coûts, d'assurer la haute-disponibilité et l'évolutivité. Depuis la version 4.2, il est aussi possible d'utiliser Ceph comme back-end pour le stockage secondaire.

OpenNebula

OpenNebula supporte bon nombre de systèmes de fichiers (partagés ou non) comme NFS, Lustre, GlusterFS, ZFS, GPFS, MooseFS, etc ou bien encore LVM, iSCSI, VMFS et Ceph. Les mêmes remarques que ci-dessus s'appliquent.

Notes subsidiaires

Ceph supporte le stockage de volumes, d'objets, de fichiers en garantissant la haute-disponibilité et le balancement de charge, est compatible avec la plupart des solutions Cloud existantes et nécessite au minimum trois noeuds pour fonctionner (quorum).

2.3 Hyperviseurs & Instances

Un hyperviseur est une plateforme de virtualisation, il permet de faire fonctionner plusieurs systèmes d'exploitations sur une même machine physique. Dans notre cas, il permet de créer des instances dans le cloud. Le tableau 2.3 liste les différents hyperviseurs disponibles pour les différentes solutions de cloud-computing considérées.

Certaines solutions IaaS proposent également la possibilité de détecter l'indisponibilité d'une instance (crash de celle-ci, de l'hôte la supportant, du réseau, etc) et de proposer une

	OpenStack	Eucalyptus	CloudStack	OpenNebula
KVM	✓	✓	✓	✓
XenServer	✓		✓	
Xen	✓	✓	✓	✓
QEMU	✓		✓	
LXC	✓		✓	
VMWare	✓	✓	✓	✓
Hyper-V	✓		✓	
Docker	✓			
PowerKVM	✓			
Baremetal	✓		✓	
Haute-disponibilité d'instances	✓		✓	✓
Auto-Scaling d'instances	✓	✓	✓	✓

TABLE 2.3 – Hyperviseurs supportés par les solutions IaaS

stratégie de restauration comme la re-création sur un autre hôte ; Cette fonctionnalité est couramment appelée la haute-disponibilité des instances. Il est également parfois proposé de créer des groupes d"approvisionnement dynamique d'instances en fonction de la charge globale du groupe / de plages horaires. C'est à dire que l'IaaS est capable de générer des instances ou d'en supprimer (entre un nombre minimum et maximum) selon un certain gabarit

Xen et XenServer sont des solutions de virtualisation qu'il convient de différencier. XenServer est le produit commercial de Citrix basé sur la solution Open-Source Xen.

Il est possible d'utiliser différents hyperviseurs dans un même IaaS à condition qu'ils soient séparés sur différents serveurs d'instances et que les conditions requises à leurs utilisations soient remplies.

L'utilisation de Ceph en tant que support de stockage pour les volumes d'instances limite le choix des hyperviseurs. En effet, actuellement, seuls les hyperviseurs utilisant «libvirt» (KVM/QEMU) supportent Ceph. Certaines implémentations de XenServer supportent également Ceph.

2.4 Communauté, Développement, Documentation

Outre les fonctionnalités et spécificités techniques des solutions, il ne faut pas négliger la communauté qui encourage deux autres éléments importants : le développement et la documentation. Choisir une solution dont la communauté est trop faible (voir mourante), dont le développement est presque nul ou dont la documentation serait proche du néant pourrait

	OpenStack	Eucalyptus	CloudStack	OpenNebula
Documentation	Forte	Moyenne	Faible	Moyenne
Communauté	Grande	Faible	Moyenne	Faible
Activité dév.	Très Elevé	Faible	Moyenne	Faible

TABLE 2.4 – Comparaison des communautés et de l'activité de développement pour les solutions IaaS

mener le projet à l'échec avant la production (avec de la chance) ou imposer d'importantes difficultés pouvant aller d'instabilités de la plateforme jusqu'à l'obligation de changer l'infrastructure complète en production. Il s'impose alors d'analyser ces caractéristiques. Le Tableau 2.4 résume la situation.

La comparaison des communautés et de l'activité du développement prend source dans le rapport annuel « Open Source IaaS Community Analysis » [46]. Quant à l'avis sur la documentation, il est issu de mon propre jugement.

On remarquera que la très grande majorité de la communauté de CloudStack est située en Asie et qu'il n'existe que peu d'utilisateurs (et donc de retours entre autres) de CloudStack en Europe.

2.5 Réseau

Un ensemble d'instances virtuelles sans connectivité est une infrastructure morte et une ce même ensemble avec des possibilités de connectivité limitées est au mieux, une infrastructure non sécurisée. Certaines solutions IaaS actuelles proposent l'utilisation de *SDN* (Software-Defined-Networking) qui permet de créer un très grand nombre de scénarios réseaux virtuels à l'aide notamment de routeurs virtuels.

On note aussi parfois l'existence à ce niveau de services de balancements de charge en tant que service (*LBaaS*) qui offrent l'opportunité de créer des balancements de charges pour les instances. Certaines solutions permettent également de créer des accès sécurisés à une infrastructure privée et sans point d'accès évident, via VPN.

OpenStack utilise le service Neutron pour gérer le réseau. Il propose de nombreux plugins servant de « back-end », allant de l'utilisation de méthodes d'isolation simples comme le VLAN via *Linux Bridge* à l'exploitation de routeurs Cisco en passant par les très répandus *OpenFlow* et *Open vSwitch* [14]. Il est possible d'utiliser plusieurs mécanismes simultanément (*Linux Bridge* / *Open vSwitch* à l'aide de ML2 [15]. On note cependant qu'il existe une table de compatibilité entre les différentes implémentations réseaux et les hyperviseurs - et cela est vrai pour toutes les solutions. Au premier abord, l'utilisation de *OpenFlow* et *Open vSwitch* est une solution intéressante.

CloudStack propose deux types de réseaux dans l'IaaS [3] :
— « Basic » où toutes les instances sont sur un même réseau physique et où il est

	OpenStack	Eucalyptus	CloudStack	OpenNebula
Isolation Simple (type VLAN)	✓	✓	✓	✓
SDN (OpenFlow / Open vSwitch)	✓			✓
Support solutions matérielles	✓			
QoS	✓			✓
Intrusion detection system	✓			
Load Balancing	✓		✓	
Firewalls	✓		✓	✓
VPN	✓		✓	

TABLE 2.5 – Comparaison des fonctionnalités réseaux pour les solutions IaaS

éventuellement possible de faire de l'isolation via la gestion de groupes de sécurité (supporté uniquement par KVM et XenServer)
— « Advanced » où les instances sont isolées dans des VLANs et il peut y avoir plusieurs réseaux physiques différents. Des routeurs virtuels sont crées pour chaque réseau physique permettant de proposer des services comme pare-feu, VPN, balancement de charge.

D'après la documentation et les différents supports de présentations disponibles concernant le réseau CloudStack, il est possible d'utiliser *Open vSwitch* avec des tunnels GRE mais le support n'est pas suffisamment mature pour pouvoir être considéré en production avec confiance.

Quant à Eucalyptus, il gère quatre types de réseau distincts [51] :
— « SYSTEM » où les machines virtuelles sont directement connectés à l'interface réseau de l'hôte à l'aide d'un Bridge et acquière leurs adresses via un serveur DHCP pré-existant et externe à Eucalyptus. Aucune isolation n'est faite et aucun adressage flottant (où certaines adresses IP peuvent être déplacées d'une instance à une autre) n'est possible.
— « STATIC » qui est très similaire au type précédent à l'exception du fait que ici, le service DHCP est assuré par le contrôleur du cluster.
— « MANAGED » qui rend disponible les adresses IPs flottantes, la création de VLANs privés et la gestion de groupes de sécurités. Les adresses IPs sont attribuées par le service DHCP du contrôleur du cluster.
— « MANAGED-NOVLAN » est similaire au type précédent sauf qu'il ne gère pas de VLANs et que l'isolation est donc assurée uniquement à travers les groupes sécurités.

Dans le cas du « MANAGED-NOVLAN », un seul LAN existe mais fait cohabiter plusieurs sous-réseaux séparés par un filtre iptables (groupes de sécurité) - il est possible pour une instance de renifler d'autres échanges sur d'autres sous-réseaux. En outre, Eucalyptus ne supporte ni *Open vSwitch* ni *OpenFlow*.

OpenNebula supporte la mise en réseau simple, l'isolation par VLAN mais également via

ebtables qui présente l'avantage de ne nécessiter aucun besoin matériel mais qui présente le défaut de ne pas permettre l'utilisation d'un même bloc d'adresses IPs plusieurs fois sur plusieurs sous-réseaux différents. Il est également possible d'utiliser *Open vSwitch* en supplément de l'isolation VLAN [10].

2.6 Philosophie

Outre les différences en fonctionnalités, les différentes solutions disponibles présentes d'importantes divergences de philosophie. Cet aspect est à considéré avec importante car la philosophie des solutions oriente leurs développements et leurs cadres d'utilisation.

La première différence vient probablement de l'organisation interne des solutions : du fait que certaines solutions ont adoptés une construction monolithique (comme CloudStack et OpenNebula) tandis que d'autres utilisent une infrastructure modulaire : Eucalyptus avec 5 modules et OpenStack avec de très nombreux modules. La vision n'est pas la même car si l'approche monolithique facilite la mise en production et le développement - elle limite par ailleurs la flexibilité. Avec une solution modulaire, il est possible de choisir les modules à utiliser (et uniquement ceux-ci) mais aussi de les faire évoluer de manière autonome en fonction des besoins. Cependant, un important travail d'intégration et de coordination s'avère alors nécéssaire.

Il est également possible d'identifier une différence de gouvernance (et donc de gestion de *Road-Map*) parmi les solutions, citons par exemple les deux philosophies de gestions de OpenStack et de OpenNebula. OpenStack est géré par une fondation formée de différents acteurs de fournisseurs et d'entreprises alors que OpenNebula se proclame dirigée par une « Dictature bienveillante » animée par les besoins des utilisateurs.

Une différence de taille oppose OpenStack aux autres solutions. OpenStack est une technologie à part entière plutôt qu'un produit et n'est pas utilisable en soi en entreprise, contrairement aux autres solutions comme CloudStack ou OpenNebula qui sont parfaitement utilisable en état. Il est vivement recommandé de choisir une distribution (commerciale et offrant donc l'accès à un support commercial ou non) propulsant OpenStack comme « Ubuntu OpenStack », « Red Hat Enterprise Linux OpenStack Platform» ou encore « Suse Cloud ».

La plus grande particularité philosophie de OpenStack est qu'il apparait que les instances OpenStack soient orientées vers l'éphémérité. En effet, bien qu'il soit possible d'assigner des volumes permanents, les instances OpenStack utilisent toutes un stockage éphémère (ayant le même cycle de vie que l'instance elle-même) et sont prévues pour pouvoir être crées massivement et être opérationnelles dès le démarrage à partir d'une image - sans intervention et configuration supplémentaire, l'utilisation du stockage objet est alors très adapté. Cette philosophie est alors très pratique lorsque combinée avec l'outils d'orchestration « Heat » permettant de créer des gabarits d'infrastructure complète en fichiers texte, on peut alors déployer (ou re-déployer, personnaliser) très rapidement des systèmes complexes. Cet éphémérité permet une grande disponibilité des applications mais aussi une grande flexibilité mais

	OpenStack	Eucalyptus	CloudStack	OpenNebula
Web UI	Minimaliste	Très limitée	Complète	Très limitée
Compatibilité EC2	Bonne	Excellente	Partielle	Partielle
Service de Monitoring	Nagios	Nagios/Ganglia	Zenoss	Intégré ou Ganglia
Service de Télémétrie	Ceilometer		Usage Engine	Intégré

TABLE 2.6 – Eléments supplémentaires de comparaison pour les solutions IaaS

requiert que les applications soient adaptées. Il est cependant tout à fait possible d'utiliser ces instances comme de machines virtuelles *standards* en leurs associant des volumes permanents et les considérant en tant que tel. Cette philosophie offre une vision toute particulière et ouvre le champ à de nombreuses possibilités tournées vers l'avenir.

2.7 Autres éléments de comparaison

Le tableau 2.7 regroupe divers éléments comparant les différentes solutions mais ne trouvant pas forcément leurs places dans les sections précédentes.

En plus de l'interface web *Horizon* fournis par OpenStack, la distribution Ubuntu pour OpenStack fournit deux outils d'interfaces supplémentaires intéressants. On citera Juju [2] qui est un logiciel utilisable soit via une interface web très simple ou via la ligne de commande permettant de déployer, configurer et maintenir diverses applications sur une infrastructure IaaS OpenStack (mais pas que). Par de simples cliques, il est possible de créer des applications, de les relier entre-elles (par exemple Wordpress et MySQL) mais aussi de les dimensionner (nombre d'instances, etc). Il est possible de programmer de nouvelles applications dans presque n'importe quel langage. En outre, il est possible de déployer les services OpenStack à l'aide de Juju. Le second outils est MaaS [3] (Metal-as-a-service) qui permet de gérer facilement à l'aide d'une interface graphique les serveurs physiques en étant capable de les démarrer, de réaliser des tests matériels, de les ré-installer et de déployer des services sur ceux-ci. C'est un outils potentiellement très intéressent lorsqu'il s'agit de gérer plusieurs centaines de machines et de les utiliser au mieux.

2.8 Faire un choix

Dans cette section, il s'agit de décider quelle solution nous devons utiliser pour mettre en oeuvre la plate-forme IaaS en fonction de nos besoins et de nos contraintes.

La première décision est probablement la plus simple : il s'agit d'écarter la solution Eucalyptus. Plusieurs raisons mènent à ce choix. Tout d'abord, Eucalyptus est conçu et développé pour réaliser des IaaS privés alors que ce projet s'intéresse à la construction d'un IaaS public - c'est à dire une plateforme destinée au grand publique, et utilisée par différentes organisations, entreprises, académies, gouvernements ou personnes simultanément. De plus et

2. https ://juju.ubuntu.com/
3. http ://maas.ubuntu.com

sans doute en conséquence du premier point, Eucalyptus ne supporte pas la répartition de charges au niveau des services contrôleurs - ce qui est un point important du point de vue de notre vision à long terme où nous désirons pouvoir faire évoluer l'infrastructure horizontalement. Autres facteurs déterminants et limitants, peu d'hyperviseurs sont compatibles ce qui pourrait nous handicaper dans le futur et seul le SAN est disponible en tant que support de stockage hautement-disponible alors que nous désirons limiter le budget dans un premier temps.

Parmi les trois solutions restantes, nous notons de la même façon que ci-dessus que OpenNebula est orienté sur la création d'infrastructure IaaS privée bien qu'il soit possible de l'utiliser en hybride ou public. On constate également que les contrôleurs ne supportent pas non plus la répartition de charges et qu'il n'est donc pas possible de les faire évoluer facilement dans le futur. La haute-disponibilité de ces derniers se fait en actif/passif et il n'est pas possible techniquement d'utiliser un balancement de charge à cause du service « oned ». Pour ces raisons et similairement à Eucalyptus, OpenNebula ne semble pas une solution adaptée à nos besoins.

Reste encore deux solutions diamétralement opposées mais fournissant tout deux d'intéressantes fonctionnalités : OpenStack et CloudStack. En procédant dans l'ordre, on constate que la première différence qui oppose ces deux géants de l'IaaS porte sur l'architecture et est conséquence directe de leurs philosophies antagonistes : OpenStack est totalement modulaire, ce qui lui permet de s'adapter à de très nombreuses situations en fonction des besoins, au prix de la complexité que cela engendre tandis que CloudStack est monolithique et à l'architecture bien cloisonnée. Dans l'optique définie dans la sous-section 1.2.5, nous sommes beaucoup plus intéressés par le fait de pouvoir évoluer facilement - nos idées se tournent donc ici naturellement vers OpenStack.

Concernant le stockage maintenant, on constate rapidement que OpenStack propose plus de possibilités, ce qui laisse ouvertes des portes pour l'avenir. Mais cela ne fait pas tout, quel stockage devrions-nous utiliser dans un premier temps et quel stockage pourra nous intéresser à l'avenir ? Le stockage iSCSI et le Fibre Channel sont clairement des solutions très utilisées en production aujourd'hui et vues comme des standards, les deux solutions les supportent, cependant, ils ne nous conviennent pas (coût trop élevé, non évolutifs, ...). Actuellement, le besoin désigne un support de stockage extensible et résilient. Comme expliqué dans l'introduction de ce projet, on désire pouvoir partir d'une infrastructure à faible coût mais capable de grandir sans difficulté et sans changement majeur. Et en cela, le stockage Ceph répond strictement au besoin, permettant de créer une plateforme de stockage haute disponibilité, évolutive et fonctionnant sur n'importe quel matériel (bas de gamme ou non, de marque propriétaire ou non, ...). Autant les deux solutions le supportent, CloudStack ne le supporte que depuis très récemment (version 4.0 pour le stockage primaire et 4.2 pour le stockage secondaire) et l'architecture cloisonnée de CloudStack (où il y a un stockage primaire par cluster) est, à mon sens, un vrai frein au développement et n'est pas vraiment compatible avec l'esprit Ceph puisque Ceph est conçu pour la création d'un seul grand réseau Ceph distribué, auto-géré et hiérarchisé. Aussi, OpenStack supporte le stockage objet, qui représente un argument très intéressant dans le futur pour de nombreuses entreprises et clients. De plus, OpenStack est capable d'utiliser Ceph à la fois pour le stockage d'instances,

volumes et d'objets - sans distinction - nous permettant de faire d'importantes économies et gains de temps. Enfin, OpenStack présente des possibilités intéressantes grâce à la distinction entre le stockage éphémère, le stockage bloc et le stockage objet, il est alors possible par exemple de provisionner des machines virtuelles avec un stockage de travail (éphémère) local sur des supports de stockage haute-performance (type SSD/SAS) et déléguer le stockage de masse sur des systèmes distribués de plus grande capacité ou alors d'utiliser exclusivement un stockage distribué ce qui permet de profiter de gros avantages de performances lors de la création de nouvelles instances. Autant d'opportunités en adéquation avec nos besoins qu'il est donc possible d'exploiter.

Par rapport aux hyperviseurs, aucune contrainte particulière n'avait été définie. OpenStack et CloudStack sont très proches dans leurs compatibilités avec les hyperviseurs. Quoi qu'il en soit, il convient de noter que l'utilisation de Ceph restreint beaucoup l'utilisation des hyperviseurs puisque seul ceux basés sur *libvirt* sont aujourd'hui compatibles (QEmu/KVM). On note que OpenStack supporte le très émergent Docker qui plus est, est bientôt compatible avec Ceph, ce qui lui est favorable.

Un autre point important est le réseau et la sécurité. Alors que CloudStack n'offre que l'isolation par le biais de VLAN (au nombre limité à 4094 par l'IEEE 802.1Q), OpenStack lui, supporte parfaitement le SDN avec OpenFlow / Open vSwitch qui lui ouvre les portes vers de grandes possibilités et vers l'avenir. En outre, OpenStack disposent d'un IDS (Intrusion Detection System) et du support de différentes solutions matérielles de réseau.

Concernant l'interface utilisateur, l'interface CloudStack est bien plus complète que celle de OpenStack, qui est relativement minimaliste, ce qui signifie que l'utilisation de l'API et de la console s'impose avec OpenStack alors que l'interface de CloudStack permet de tout faire. Cela ne va pas à l'encontre de nos contraintes ou de nos besoins. On retiendra que la compatibilité EC2 est meilleure sous OpenStack que sous CloudStack. Les deux solutions disposent de services de monitoring et de télémétrie.

Dernier point permettant de départager les deux solutions envisagées, la communauté, la documentation et l'activité de développement. La documentation officielle de OpenStack est beaucoup plus importante que celle de CloudStack, tout comme l'activité de de développement. On remarque également que la communauté OpenStack est également plus grande, on trouve donc de nombreuses ressources issues de la communauté sur Internet mais aussi des conférences qui sont très régulièrement organisées.

A travers l'étude de l'état de l'art, nous avons découvert et comparé les différentes solutions qui composent aujourd'hui le monde de l'IaaS. Compte-tenu des arguments ci-dessus, nous avons également déterminé quelle solution est la plus adaptée vis-à-vis de notre besoins et de nos contraintes, et qui sera étudié lors de ce projet : OpenStack.

2.9 Quelle distribution OpenStack ?

Dans ce chapitre, il convient de faire un dernier choix, celui de la distribution qui supportera OpenStack. En effet, OpenStack est un vaste projet « upstream » - qui n'est pas prêt à la production en tant que tel et dont l'intégration est longue et difficile. C'est pour cela que différentes distributions existent et proposent des paquets plus « prêts à l'emploi » intégrant

différentes facilités. Il existe également des solutions « clés en main» où il suffit d'acheter le matériel vendu par la société et leurs services d'installation, tout est géré automatiquement ensuite. On cite par exemple *ElectraStack* et *Nebula One*.

Il va sans dire que étant donné l'important travail en amont réalisé par ces distributions, la plupart d'entre elles sont payantes. Certaines cependant proposent leur distribution gratuitement mais proposent des offres commerciales de support technique ou de plugins.

Parmi les différentes distributions disponibles, on écarte *HP Helion OpenStack® Community* qui est destiné aux IaaS privés. *Piston Cloud* est également conçu pour les IaaS privés et est payant. Les distributions *IBM Cloud Manager with OpenStack* et *SUSE Cloud* sont également entièrement payantes, tout comme les distributions proposées par Solaris et Oracle.

Peu de distributions sont gratuites, on note *RDO*, *Mirantis OpenStack* et *Canonical Ubuntu*. La distribution *RDO* est une distribution communautaire pour le déploiement de OpenStack sur systèmes Red Hat et dérivés (comme CentOS). Elle utilise un outils développé spécialement pour la distribution : « packstack » qui utilise lui-même « Puppet », dont les scripts sont disponibles publiquement et donc utilisable indépendamment. La documentation est relativement limitée et Red Hat propose une distribution commerciale nommée *Red Hat Enterprise Linux OpenStack Platform* en la mettant en valeur de la sorte « Lorsque vous êtes prêt pour la production, ... ». Cela signifie donc que cette distribution communautaire n'est pas un outils adapté à la production mais plus une distribution de découverte promouvant l'utilisation de la solution commerciale. En prenant en considération les besoins exprimés dans ce présent rapport, il est exclus d'utiliser ce genre de solution car cela pourrait signifier devoir effectuer de très importantes modifications à l'existant pour pouvoir évoluer. *Mirantis*, un leader dans le monde OpenStack, propose sa propre «distribution» qui repose sur les distributions *Ubuntu* ou *CentOS* avec une série de plugins (dont beaucoup sont payants) et qui offre une interface web permettant de déployer et de gérer son cloud OpenStack. Cette interface web est le principal argument de cette distribution et malheureusement à priori son seul atout : rien ne semble être proposé en dehors de cette interface web et de plugins/drivers propriétaires et payants. Dans notre cas, cette interface web ne nous intéresse pas vraiment, elle permet d'aider au déploiement mais ne s'adapte pas à des déploiements de masse et perd donc une grande partie de son intérêt. Intéressons nous maintenant à Ubuntu, système le plus couramment utilisé pour le déploiement de OpenStack [30]. Outre le fait qu'ils proposent des paquets OpenStack, *Canonical* met à disposition les outils *Juju* et *MaaS* dont nous avons déjà parlé précédemment dans ce rapport et qui permettent de réaliser facilement des déploiements et configurations OpenStack mais aussi des installations complètes à partir de machines vierges. Ces outils permettent entre autres de déployer (ou de convertir, ou de supprimer) des dizaines voir centaines de services en seulement quelques commandes. De plus, il est possible de souscrire à un support commercial relativement peu onéreux donnant accès notamment à un nouvel outils nommé *Landscape* qui offre de grandes facilités de gestion de masse - cette possibilité est intéressante pour l'avenir.

Nous faisons alors le choix d'utiliser la distribution *Ubuntu* pour les divers outils qu'elle

offre mais aussi pour son support commercial, sa grande notoriété, sa polyvalence, sa communauté et sa documentation en ligne des plus fournies.

Architecture

Ce chapitre se consacre à la définition de l'architecture de la plateforme IaaS. L'architecture s'intéresse à la répartition des différents services composant le système ainsi qu'aux concepts et éléments de configurations clés le définissant.

3.1 Services & Composants

Notre plateforme utilise un important nombre de services pour fonctionner, dont une grande partie est issue de OpenStack. Cette section vise à expliquer le rôle de chacun des services utilisés dans le système. La répartition des services vis-à-vis des types de noeuds est décrite en Figure 3.10. Les services de OpenStack sont organisés en composants et sont ainsi traités composant par composant.

3.1.1 Compute (Nova)

Compute est le composant OpenStack permettant la gestion de machines virtuelles. Nova ne fait pas office d'hyperviseur et ne peut donc pas créer d'instances par lui-même, au lieu de cela, il s'appuie sur un ou plusieurs hyperviseurs configurés pour fonctionner avec lui. Dans notre cas, il s'agit de KVM qui est manipulé à l'aide de *libvirt*.

Services du composant

nova-api est responsable de la réception et du traitement des requêtes concernant le composant.

nova-compute gère les machines virtuelles à proprement dites en communiquant avec les hyperviseurs.

nova-scheduler répartit les ressources à allouer à travers la plateforme, en fonction des ressources disponibles mais aussi de différents filtres.

nova-conductor permet l'utilisation du service nova-compute sans que ce dernier n'ait d'accès aux bases de données (et donc que les identifiants ne soient pas stockés sur les mêmes serveurs dédiés que les machines virtuelles, ce qui pourrait entraîner un risque de sécurité).

nova-novncproxy permet l'utilisation de clients VNC pour accéder aux consoles des machines virtuelles en faisant office de proxy.

nova-consoleauth réalise l'authentification des consoles proxy.

nova-cert permet la génération de certificats X509 pour EC2 (euca-bundle-image).

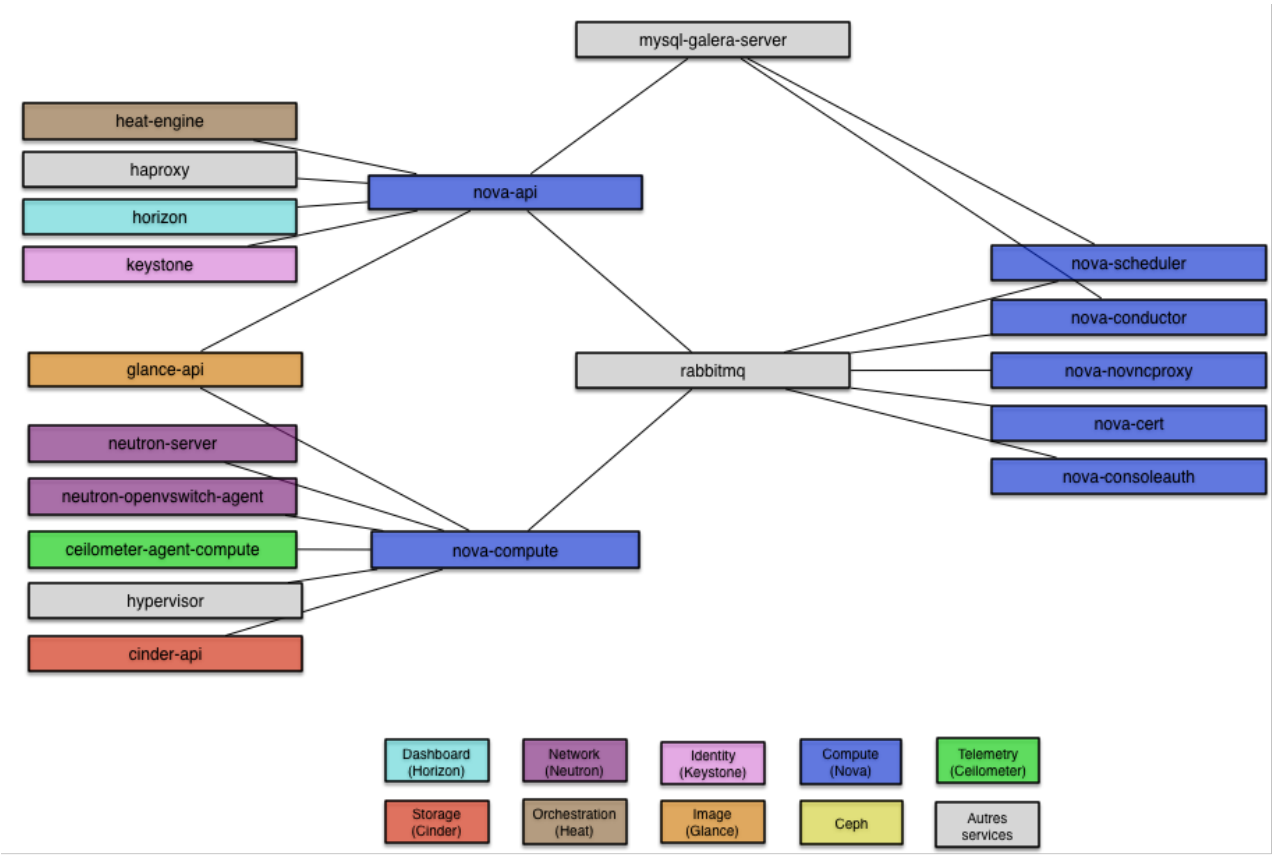

FIGURE 3.1 – Architecture - Interactions de Nova

3.1.2 Image Service (Glance)

Glance permet la découverte, la sauvegarde et la récupération des images de disques durs utilisées par les machines virtuelles.

Services du composant

glance-api est responsable de la réception et du traitement des requêtes concernant le composant.

glance-registry sert les méta-données liées aux images et communique avec les bases de données.

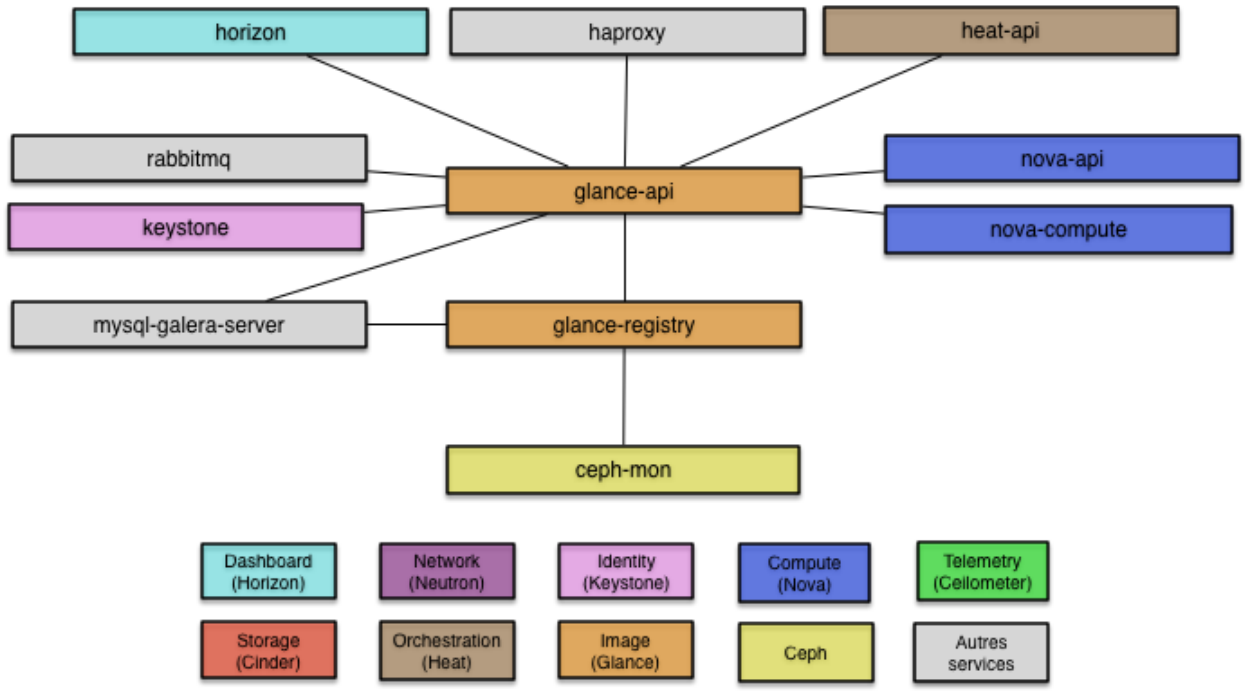

FIGURE 3.2 – Architecture - Interactions de Glance

3.1.3 Dashboard (Horizon)

Horizon offre une interface utilisateur à travers un navigateur, permettant de gérer les services offerts par OpenStack (mais pas tous !).

Service du composant

apache2 sert les pages de l'interface utilisateur.

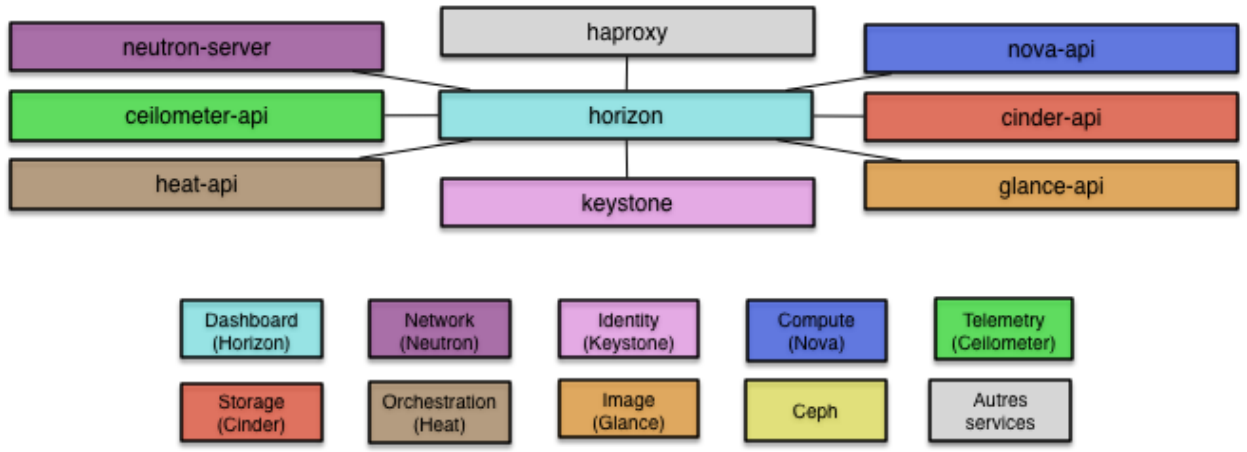

FIGURE 3.3 – Architecture - Interactions de Horizon

3.1.4 Identity (Keystone)

Keystone agit tel un annuaire qui centralise toutes les authentifications et autorisations nécessaires aux multiples services d'OpenStack. Il peut aussi servir de catalogue de service.

Service du composant

keystone-all assure l'intégralité des tâches gérées par le composant.

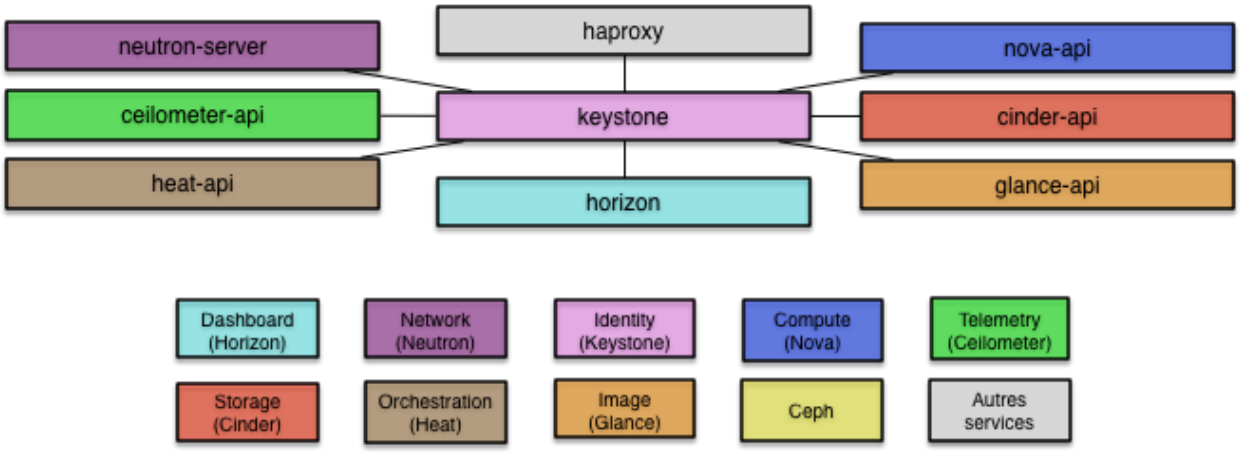

FIGURE 3.4 – Architecture - Interactions de Keystone

3.1.5 Network (Neutron)

Neutron est le composant OpenStack offrant le "réseau en tant que service". Il permet entre autres de créer des réseaux, d'assigner des adresses IPs aux machines virtuelles dans ces réseaux, mais aussi de créer des accès VPN, des balancements de charges, des pare-feus, etc.

Services du composant

neutron-server est responsable de la réception et du traitement des requêtes concernant le composant.

neutron-dhcp-client offre le service DHCP aux instances virtuelles de Nova.

neutron-l3-agent réalise le routage, le NAT et le *forwarding* réseau de niveau 3 en utilisant les *namespace* pour gérer de façon isolée de multiples réseaux se chevauchant.

neutron-metadata-agent permet aux instances virtuelles de communiquer Nova afin de récupérer leurs méta-données. Il fait donc office de passerelle entre les réseaux des instances et le réseau dans lequel Nova opère.

neutron-plugin-ovs-agent est un plugin dont se sert Neutron pour effectivement créer le réseau définit. Ici, c'est donc *Open vSwitch* qui est utilisé pour créer des réseaux, qui sont par conséquent virtuels et qui transitent via des tunnels GRE. Il est possible d'utiliser des plugins propriétaires tels que les plugins Cisco lorsque le réseau physique est adapté.

neutron-lbaas-agent s'occupe de la réalisation des "balancements de charge en tant que services".

neutron-metering-agent mesure l'activité réseau au niveau des routeurs.

neutron-vpnaas-agent permet de créer divers types de VPN, pour accéder aux réseaux virtuels ou bien encore pour connecter plusieurs réseaux ensembles à travers potentiellement plusieurs zones de disponibilités.

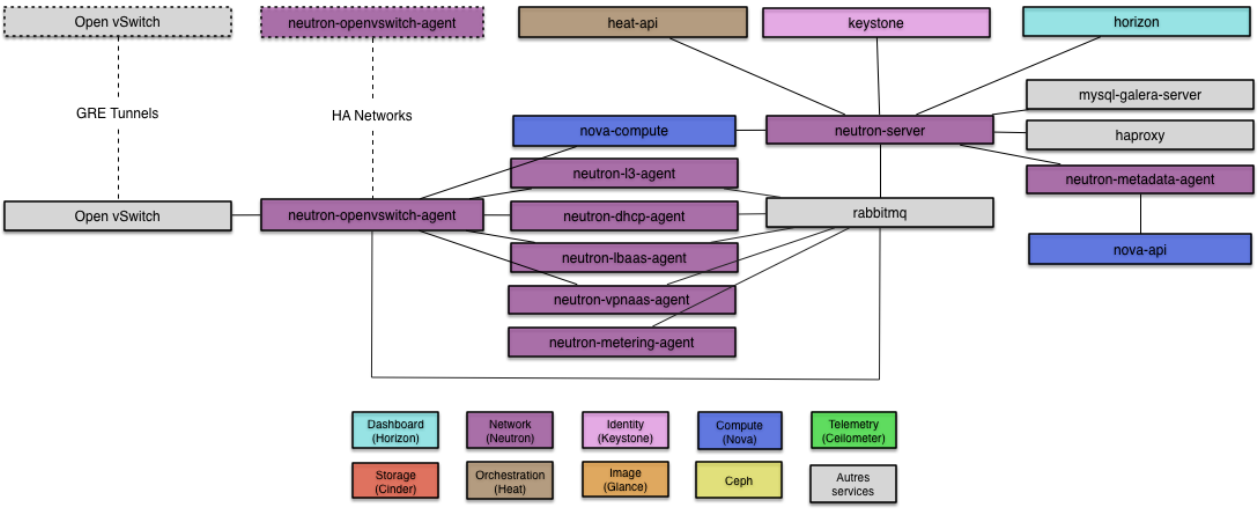

FIGURE 3.5 – Architecture - Interactions de Neutron

3.1.6 Storage (Cinder)

Cinder fournit le support de volumes (disques) persistants pour les machines virtuelles.

Services du composant

cinder-api est responsable de la réception et du traitement des requêtes concernant le composant.

cinder-scheduler répartit les volumes à allouer à travers la plateforme, en fonction des ressources disponibles.

cinder-volume gère les volumes effectivement. Il repose sur divers backends (supports physiques), dans notre cas, nous utilisons *rbd* pour nous servir de Ceph.

cinder-backup permet la création et la restauration de sauvegardes de disques.

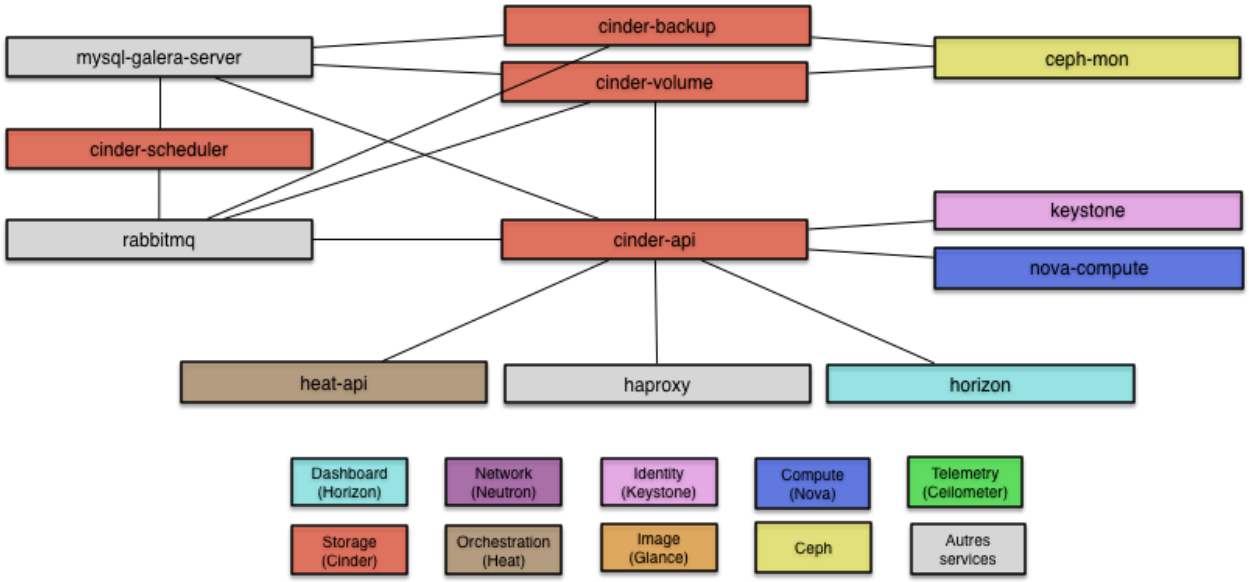

FIGURE 3.6 – Architecture - Interactions de Cinder

3.1.7 Orchestration (Heat)

Heat est un outils d'orchestration visant à créer un système humainement compréhensible et aisé pour la création, le déploiement et la gestion du cycle de vie d'infrastructures complètes à partir de définitions en fichiers texte.

Services du composant

heat-api est responsable de la réception et du traitement des requêtes concernant le composant.

heat-engine réalise tout le travail d'orchestration et est ordonné par heat-api.

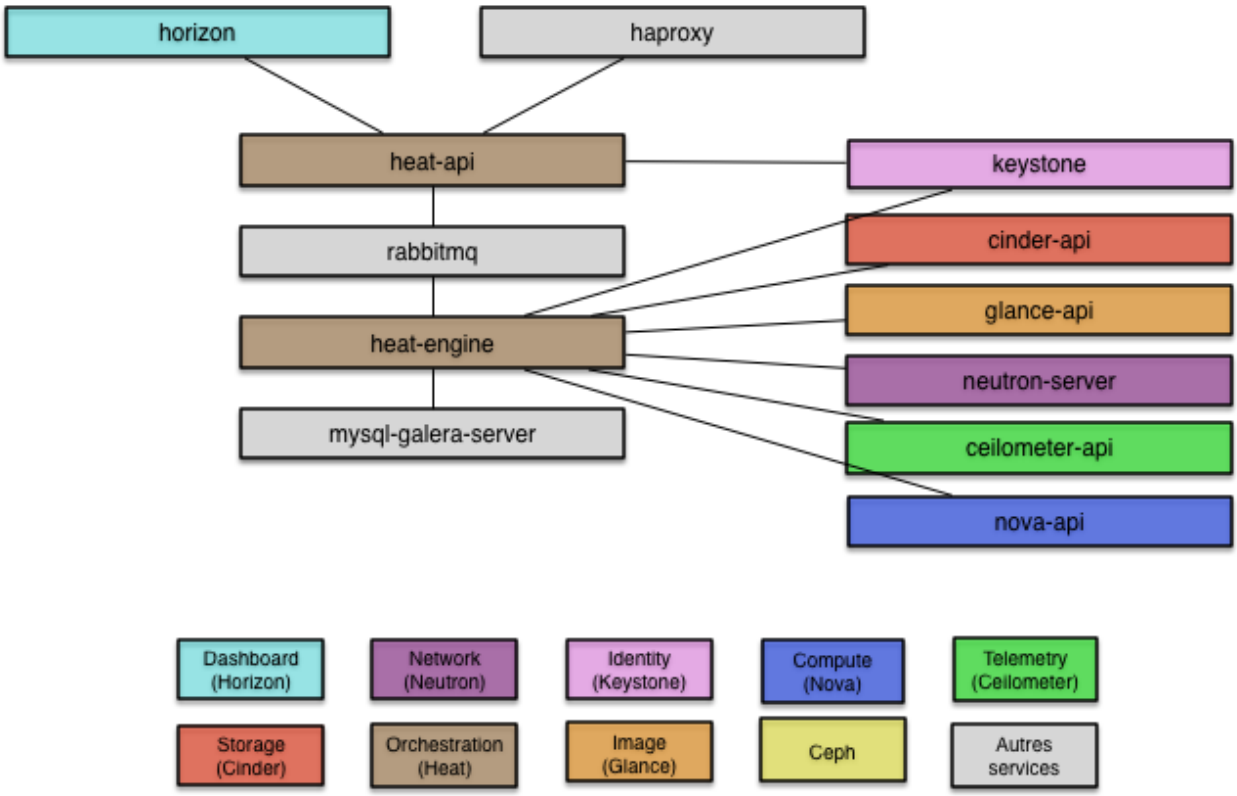

FIGURE 3.7 – Architecture - Interactions de Heat

3.1.8 Telemetry (Ceilometer)

Ceilometer est l'outil de mesure de OpenStack. Il enregistre l'utilisation des ressources physiques et virtuelles de l'ensemble de la plateforme et peut réaliser certaines actions prédéfinies en conséquence (exemple, ajouter une nouvelle instance de traitement des requêtes vers un site si les instances déjà en place sont saturées).

Comme montré sur la Figure 3.8 et décrit dans la documentation [4], Ceilometer utilise de deux méthodes distinctes pour collecter les données. La méthode privilégiée est l'écoute des files de messages de notifications où de nombreux services OpenStack envoient des données. La seconde méthode, plus active, est l'interrogation des services (via leurs APIs par exemple). Cette méthode est nécéssaire car plusieurs services ne fournissent pas l'ensemble des données nécessaires (comme Nova où les données sont collectées directement auprès de l'hyperviseur) mais est néanmoins dépréciée à cause de la charge que cela incombe.

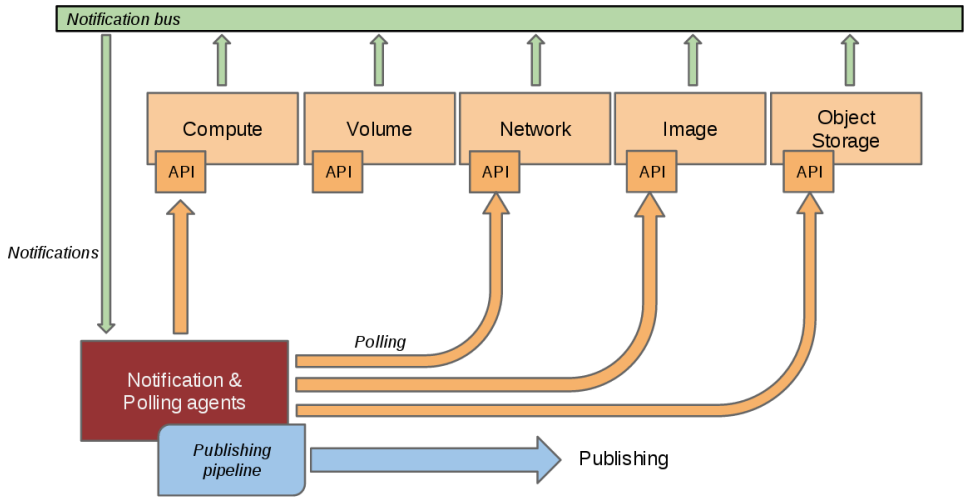

FIGURE 3.8 – Architecture - Ceilometer - Système de collecte

Services du composant

ceilometer-api est responsable de la réception et du traitement des requêtes concernant le composant.

ceilometer-agent-central interroge les services OpenStack afin de collecter des données.

ceilometer-collector est le service collectant les mesures depuis les agents de collectes (ceilometer-agent-central et ceilometer-agent-compute) et les notifications (via ceilometer-agent-notification) et les stockant en base de données.

ceilometer-agent-notification collectes les données provenant des files de messages de notifications.

ceilometer-alarm-evaluator détermine si certaines alarmes devraient être déclenchées en fonction des statistiques sur une période de temps donné.

ceilometer-alarm-notifier exécute les actions liées aux alarmes lorsqu'elles sont déclenchées.

ceilometer-agent-compute envoie des informations statistiques liées aux machines virtuelles aux services en charge de les collecter.

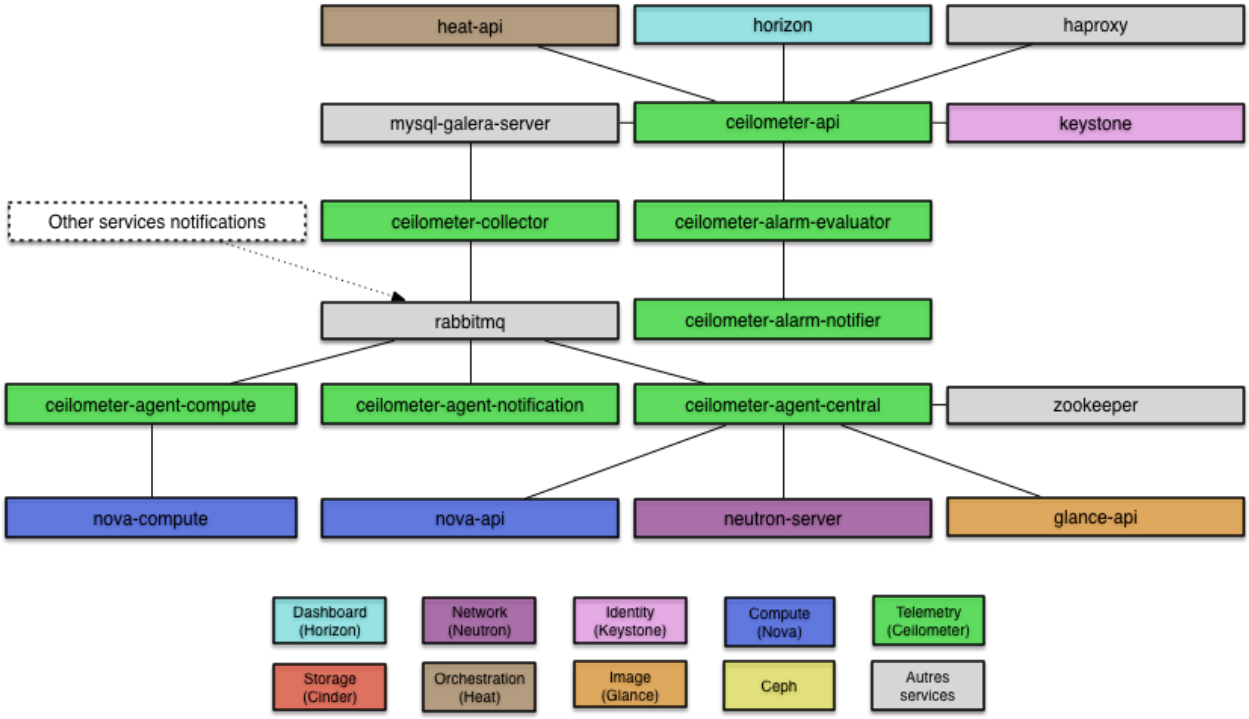

FIGURE 3.9 – Architecture - Interactions de Ceilometer

3.1.9 Tempest

Tempest est un ensemble de tests d'intégration pour plateformes OpenStack, qui se veut exhaustive. Tempest se place du côté du client et interroge l'ensemble des accès disponibles publiquement avec une série de pas moins de 1500 validations, scénarios et tests spécifiques permettant de valider le bon fonctionnement de la plateforme.

3.1.10 Services indépendants de OpenStack

Les services de cette catégorie sont utilisés ne font pas partie intégrante du projet OpenStack mais sont néanmoins nécéssaire à son fonctionnement. Ils sont utilisés par plusieurs composants de OpenStack et sont éventuellement interchangeables avec d'autres services libres ayant le même rôle.

HAProxy

HAProxy[1] est un logiciel libre, efficace et léger permettant de faire du balancement de charge HTTP et TCP. Chaque qui est capable de traiter entre 15 000 et 40 000 requêtes / seconde sur un processeur type Opteron à deux coeurs.

Dans notre plateforme, HAProxy est utilisé pour répartir les requêtes vers *galera*, *rabbitmq*, *Keystone (admin / public)*, *Glance (api / registry)*, Cinder (api), *Nova (api / metadata*

1. http ://www.haproxy.org

/ novncproxy), *Neutron (api)*, *Heat (api / cfn / cw)*, *Horizon* et *Ceilometer (api)*.

Ainsi, les requêtes à destination des services listés ci-dessus transitent systématiquement via HAProxy afin de répartir la charge sur l'ensemble de la plateforme mais aussi parce que par conséquent, c'est HAProxy et lui seul qui détient la liste des services actuellement fonctionnels et prêt à répondre. En effet, HAPRoxy se charge de tester très régulièrement les services enregistrés afin de vérifier si ils sont disponibles et dans le cas contraire, les exclure du groupe jusqu'à tant qu'ils soient à nouveau détectés comme opérationnels.

RabbitMQ

RabbitMQ [2] gère des files de messages afin de permettre aux différents services de communiquer ensemble très simplement et sans nécessité de connaissance particulière. RabbitMQ s'appuie sur le protocole AMPQ, tout comme OpenStack. Il est donc également possible d'utiliser *Qpid* par exemple. Plus précisément, le composant Nova utilisent les RPC pour communiquer. Il est possible d'en apprendre plus sur l'article [19].

tftpd

tftpd est un service répondant au protocole TFTP (Trivial File Transfer Protocol) - protocole simplifié de transfert de fichiers. Il est utilisé dans la plateforme pour fournir les images de démarrage des machines physiques dans le but de les installer.

bind9

Célèbre serveur DNS, bind9 traduit les noms d'hôtes des machines de la plateforme en adresses IP (et vice-versa). De la sorte, il n'est jamais nécéssaire de citer les serveurs de la plateforme avec leurs adresses IP (difficiles à retenir) mais avec leurs noms (plus compréhensibles).

dhcpd

dhcpd offre la configuration réseau (adresse ip, masque, passerelle, ...), via le protocole DHCP, des serveurs dédiés identifiés par leurs adresses MAC, ainsi que leur nom d'hôte (utilisé notamment pour le déploiement Puppet) et leur indique où trouver l'image de démarrage PXE pour installation.

C'est donc ce service qui détermine si un serveur dédié doit être ré-installé au prochain démarrage avec une image préfabriquée, si il doit être géré par Puppet et le cas échéant : de quelle façon.

ntpd

ntpd permet aux serveurs de synchroniser leurs horloges précisément, à l'aide du protocole NTP et de façon homogène. Il est essentiel que l'ensemble des serveurs partagent la même heure car elle est au coeur de certains protocoles de sécurité (notamment afin d'éviter les *replay attack*) mais surtout car les services mis en *cluster* et gérant un quorum (cas

2. https ://www.rabbitmq.com

de Ceph, de Zookeeper, de RabbitMQ, des serveurs de bases de données par exemples) ou gérant des transactions (cas des serveurs de base de données) ont besoin d'un référentiel temps commun, sans quoi ils seront dans l'impossibilité de fonctionner.

De nombreux serveurs de temps sont disponibles sur Internet mais leur utilisation ne permet pas de garantir une précision suffisante pour les services de la plateforme à cause de la latence et la guigue qui sont importantes sur le réseau Internet. L'effet est amplement réduit grâce à l'utilisation d'un serveur NTP local. Google a par ailleurs développé son propre système (TrueTime API) pour synchroniser l'heure sur de grands réseaux, potentiellement géo-distants [17].

zookeeper

Apache Zookeeper est un logiciel libre permettant de réaliser de la configuration, synchronisation et nommage distribuée. Il est utilisé dans notre plateforme pour la synchronisation des agents centraux de Ceilometer (ceilometer-agent-central) qui se répartissent ainsi les tâches.

puppet

puppet [3] est l'outils libre de déploiement et de configuration des serveurs. À l'aide d'un language de programmation déclaratif et optionnellement de Ruby [4], on déclare la façon dont laquelle les serveurs sont configurés et déployés (présence ou absence de logiciels, configuration, commandes à exécuter, présence/absence/contenu de fichiers, etc) tout en gérant une notion de dépendance permettant de définir l'ordre dans lequel les choses sont faites. De nombreux modules développés sont disponibles pour la communauté sur Puppet Forge [5] ce qui fait gagner un temps précieux. Puppet identifie chaque serveur par son nom d'hôte.

puppet , installé sur chaque serveur géré par Puppet, déploie et configure les serveurs. Il s'assure également qu'ils soient toujours conformes aux prérogatives du puppetmaster en vérifiant à intervalle régulier.

puppetmaster , installé sur un unique serveur, sert d'autorité de certification et met à disposition les modules définissant les serveurs.

galera

OpenStack utilise abondamment des bases de données afin d'y stocker l'ensemble des informations nécessaires à son fonctionnement, c'est pourquoi nous utilisons *Percona XtraDB Cluster* qui est une solution Open-Source, active/active, haute disponibilité et haute évolutivité pour la gestion de bases de données. Cette solution repose sur *Percona Server* (remplacement du standard MySQL Server), *Galera* (librairie de réplication multi-maitre), *wsrep API* (Write Set REPlication API) et de *Percona XtraBackup*. Pour former un cluster valide, on note qu'il faut trois noeuds au minimum.

3. https ://puppetlabs.com
4. https ://www.ruby-lang.org/fr/
5. https ://forge.puppetlabs.com

La réplication est synchrone et multi-maître : on peut donc écrire à partir de n'importe quel noeud du cluster mais la donnée ne sera effectivement considérée comme écrite que lorsque tous les noeuds l'auront écrites. Une limitation apparait alors puisque la vitesse d'écriture est limitée par le noeud le plus lent. Cela nous amène également à une deuxième limitation, le cluster utilise un contrôle de concurrence optimiste qui peut générer des erreurs (*Error : 1213 SQLSTATE : 40001 (ER_LOCK_DEADLOCK)*) [35] et l'annulation d'une transaction lorsque deux opérations s'effectuent sur deux noeuds différents mais sur le même enregistrement.

ceph

Ceph est une plateforme libre de stockage distribué et répliqué. Ceph est capable de stocker des objets, des périphériques blocs et des systèmes de fichiers. Il est utilisé ici afin de stocker les machines virtuelles, leurs volumes, les images d'installations des machines virtuelles ainsi que l'ensemble des sauvegardes et des instantanés. Ceph est utilisé notamment par Glance, Nova et Cinder.

ceph-mon gère le cluster de stockage, s'assure de son bon fonctionnement et répartit les données parmi les ceph-osd.

ceph-osd est responsable du stockage des données sur le système de fichier local et fournit l'accès à ces dernières à travers le réseau.

Ceph organise ses données en « pools » qui disposent de différentes spécificités telles que : le nombre de répliques des données, le nombre de groupes de placement qui définit en combien d'agrégats le pool est séparé, un ensemble de droits, et un ensemble de règles CRUSH qui spécifie de quelle façon les données sont réparties à travers le cluster.

Dans notre cas, nous définissons 4 pools de données, à savoir :

images utilisé par Glance pour stocker ses images d'installations

vms utilisé par Nova pour stocker les instances virtuelles

volumes utilisé par Cinder pour stocker les disques permanents des instances virtuelles

backups utilisé par Cinder Backup pour stocker les sauvegardes des disques permanents

Ces règles CRUSH sont très importantes dans un environnement de grand taille et de production car elles permettent d'organiser les données de façon précise, notamment en fonction de l'emplacement géographiques des serveurs et de l'organisation réseau. On peut ainsi faire en sorte par exemple que les instances virtuelles soient stockées au plus près des noeuds de calcul correspondant afin de disposer de bonnes performances et définir que les sauvegardes de volumes doivent être réparties dans différents endroits éloignés.

3.2 Types de noeuds

Bien qu'il soit possible de réaliser avec OpenStack un déploiement minimal en un seul noeud (serveur), nous considérons ici 5 types de noeuds, ceci afin de pouvoir répondre aux

exigences décrites en Section 1.2 (plus particulièrement les points d'évolutivité et de haute disponibilité). La distribution des services OpenStack est réalisée de sorte à ce que chacun des types assurent un rôle particulier dans le fonctionnement de l'IaaS et que les charges associées évoluent de façon « indépendantes » - c'est à dire que l'augmentation du besoin en une ressource gérée par un noeud de type donné X n'influe à priori uniquement sur la charge des noeuds de type X et non pas sur les noeuds des autres types.

Le principe est alors de déployer un certains nombre de noeuds de chaque type afin de répondre à la charge mais aussi d'assurer la disponibilité. En effet, chacun des noeuds d'un type donné est capable de traiter n'importe quelle requête correspondant à son type, indépendamment des éventuelles requêtes précédentes (notion de *Stateless*, *serveur sans état*). Lorsque le besoin devient plus important, il suffit alors de déployer un nouveau noeud du type correspondant afin qu'il commence à traiter les requêtes. Lorsqu'un de ces noeuds subit une défaillance, il ne se voit plus attribué de requêtes et le traitement est alors assuré par les autres noeuds disponibles du même type. On comprend alors aisément qu'il faut à minima deux noeuds d'un type donné pour pouvoir assuré le service de façon hautement disponible, ce qui est à la fois correct et incorrect. En effet, certains services (décrits ci-après) nécessitent l'établissement d'un *quorum* pour pouvoir fonctionner convenablement (être capable de déterminer qui dispose d'un accès aux ressources partagées lors d'une défaillance, pour assurer la cohérence) et requiert donc la présence de trois noeuds.

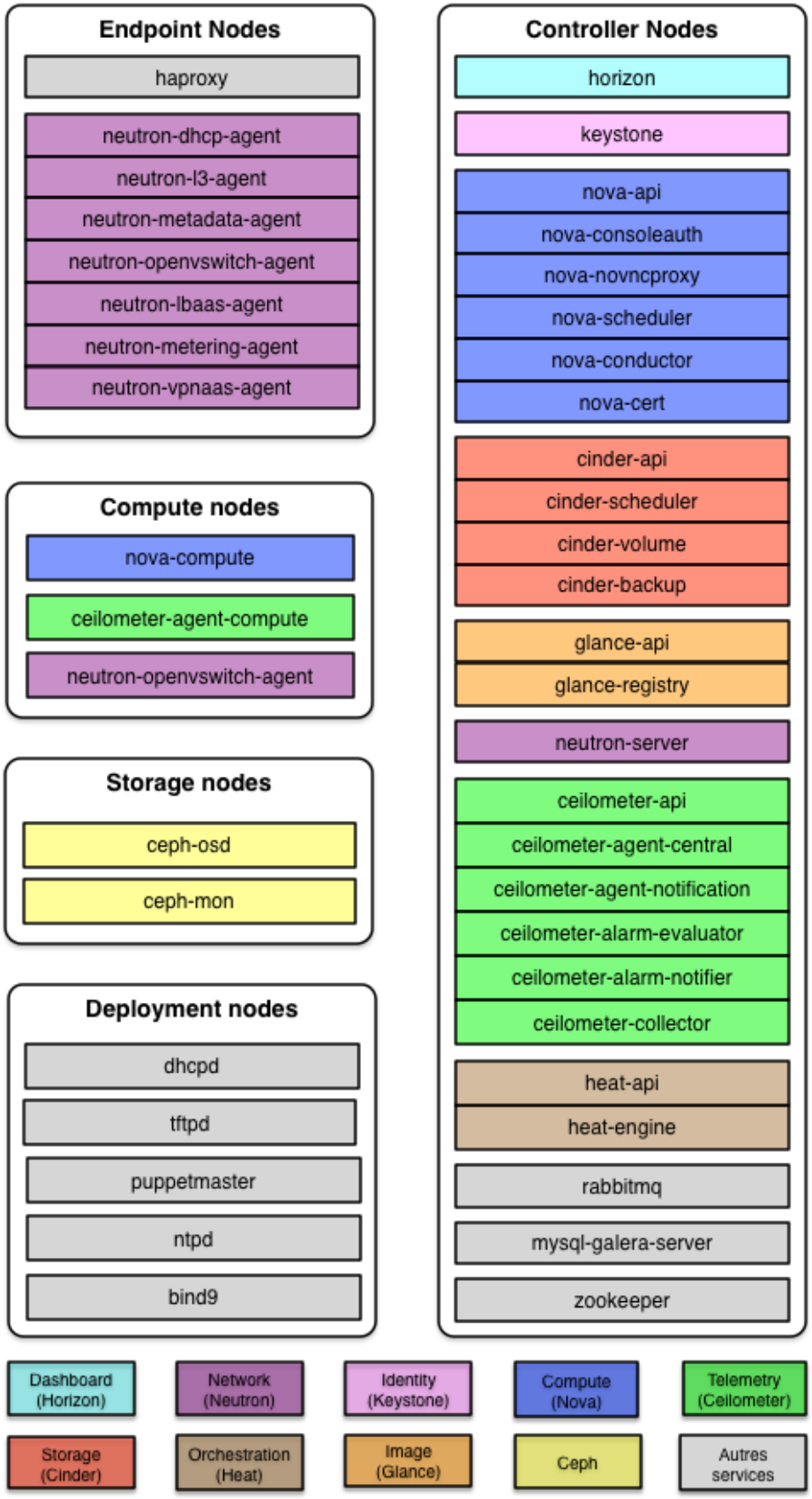

FIGURE 3.10 – Architecture - Répartition des services par type de noeud

3.2.1 Noeuds de balancement de charge & d'extrémité réseau

Dans le cadre de notre déploiement de démonstration, deux noeuds de ce type sont installés et sont en tête de la plateforme. Chacun des noeuds fait interface entre le monde extérieur (par exemple, le réseau Internet) et la plateforme Cloud. Chacun d'eux disposent d'une adresse IP extérieure. Ces noeuds ont deux rôles :

— Recevoir les requêtes à l'API de l'IaaS et les distribuer vers les noeuds de contrôle
— Héberger les routeurs virtuels des consommateurs, relayant ainsi les données des instances

Le balancement de charge des requêtes API se fait à l'aide de *HAProxy*. Dès lors qu'il est nécéssaire de traiter plus de requêtes, l'ajout de nouveaux noeuds de balancement de charge est alors possible grâce au mécanisme de *Round Robin DNS* [23, 9]. Afin d'illustrer les choses, admettons que nous n'ayons qu'un seul noeud de balancement de charge et que l'accès à l'API se fasse habituellement en utilisant l'adresse *api.eu-1.harmony-hosting.com*, les serveurs DNS résolvent ce sous-domaine en l'adresse IP du noeud de balancement de charge à l'aide de l'enregistrement A correspondant. Maintenant, considérons que nous disposions d'un nouveau noeud, on ajoute alors un nouvel enregistrement A pour le même-sous domaine et pointant vers l'adresse IP du nouveau noeud. À l'aide du *Round Robin DNS*, lorsqu'un client demandera à résoudre *api.eu-1.harmony-hosting.com*, il recevra maintenant une liste d'adresses IPs (deux adresses en l'occurence) à contacter et contactera la première de la liste - et si elle ne fonctionne pas, contactera la suivante. Maintenant, ce client ou un autre client réalise à nouveau une demande de résolution, l'ordre des adresses aura changé et contactera alors ainsi une autre adresse - ce qui correspond à un autre noeud de balancement de charge. De la sorte, chacun des noeuds de balancement de charge sera sollicité, ce qui a pour effet d'augmenter la capacité de traitement. Ce mécanisme est illustré ci-dessous dans la Figure 3.1 pour la résolution de *Google.com*. Il est important de noter que cette méthode ne peut soustraire l'utilisation de *HAProxy* car ce balancement de charge est grossier et inexact de part la nature du protocole DNS et des serveurs de cache. Il faut également assurer la haute-disponibilité, alors qu'advient-il lorsqu'un des noeuds de balancement de charge est indisponible ? Deux choses. D'abord et comme cela a été dis précédemment, un client ayant reçu l'indication de joindre un serveur indisponible va effectivement tenter de le contacter et faire face à une expiration de délai, il va alors contacter le deuxième serveur de la liste - la requête aura fonctionné finalement. Second point, certains serveurs DNS (tel que *lbnamed*[6]) sont en mesure d'effectuer des vérifications sur l'état des noeuds à intervalles réguliers et de mettre à jour leurs configurations en conséquences, de sorte, ces serveurs DNS n'incluront pas (ou, pas longtemps) dans leurs réponses des serveurs injoignables.

D'autre part, chaque instance des consommateurs voulant communiquer le fait via une passerelle, qui est en réalité un routeur virtuel configuré au préalable sur le réseau du locataire, par ce dernier. Ces routeurs virtuels sont définis sur les noeuds d'extrémité réseau. Ce fonctionnement est détaillé plus en détails dans la Section 3.3.

6. http://web.stanford.edu/~riepel/lbnamed/

Listing 3.1 – Mécanisme de Round-robin DNS avec Google.com

```
[quentinm@lab-iaas]# nslookup google.com
Server:         172.24.31.254
Address:        172.24.31.254#53

Non-authoritative answer:
Name:   google.com
Address: 193.51.224.144
Name:   google.com
Address: 193.51.224.152
Name:   google.com
Address: 193.51.224.187
Name:   google.com
Address: 193.51.224.165
Name:   google.com
Address: 193.51.224.154
[...]

[quentinm@lab-iaas]# nslookup google.com
Server:         172.24.31.254
Address:        172.24.31.254#53

Non-authoritative answer:
Name:   google.com
Address: 193.51.224.166
Name:   google.com
Address: 193.51.224.148
Name:   google.com
Address: 193.51.224.144
Name:   google.com
Address: 193.51.224.152
Name:   google.com
Address: 193.51.224.187
[...]
```

3.2.2 Noeuds de contrôle

Les noeuds de contrôle représentent le coeur ont pour rôle de gérer la plateforme : ils traitent l'ensemble des requêtes API leur provenant des noeuds de balancement de charge et assurent le fonctionnement de la plupart des services de la plateforme. Ils hébergent entre autres les bases de données et les files de messages (permettant aux services de communiquer entre eux).

Trois noeuds de contrôle minimum doivent exister afin d'assurer le quorum des bases de données - sans quoi, la haute-disponibilité ne pourrait être assuré et il est recommandé d'en disposer un nombre impair afin d'éviter qu'on ne puisse avoir deux groupes séparés de façon égale (et donc avec le même nombre de votes). Comme évoqué précédemment, les services sont sans état : l'ajout de nouveaux noeuds permet d'augmenter la capacité de traitement et on note de surcroit que l'indisponibilité complète de l'ensemble de ces noeuds n'impacterait pas le fonctionnement des machines virtuelles, de leurs réseaux et de leurs stockages - outre le fait qu'il ne soit plus possible de les manipuler ni possible de manipuler la plateforme de manière générale.

3.2.3 Noeuds de stockage

Les noeuds de stockage forment un cluster permettant de stocker de façon hautement disponible et évolutive l'ensemble des données de la plateforme. Ce cluster de stockage repose sur Ceph, comme discuté en Section 2.8. Il faut disposer d'au moins 3 noeuds comme pour les contrôleurs afin d'assurer un quorum.

3.2.4 Noeuds de calcul

Les noeuds de calculs sont dédiés à l'hébergement des instances. C'est sur ces noeuds que fonctionnent les hyperviseurs permettant de réaliser de la virtualisation et de créer des machines virtuelles.

3.2.5 Noeuds de déploiement

Ces noeuds aident au déploiement de l'infrastructure en offrant le service d'amorçage *PXE* (*Pre-boot execution environment*) et le serveur maître *Puppet*. Ils proposent en outre le *DHCP* afin de configurer le réseau sur les serveurs en leur indiquant notamment leur adresse IP, leur nom d'hôte, la passerelle, les serveurs DNS à utiliser et le masque réseau.

L'amorçage *PXE* permet à un serveur vierge de notre plateforme de démarrer et d'entrer en fonction automatiquement en téléchargeant une image du système sur le réseau. Cet amorçage repose sur deux briques logicielles : un serveur *DHCP* assignant une adresse IP au démarrage de la machine (lors de la séquence BIOS) et transmettant un fichier de configuration et un serveur *TFTP* distribuant l'image sur laquelle démarrée. Le *PXE* offre plusieurs possibilités comme l'installation sur un système vierge d'un système modifié prêt à fonctionner immédiatement (par exemple le système d'exploitation ainsi que les logiciels et la configuration nécessaires à assurer un certain rôle) ou bien encore le lancement d'un

système modifié prêt à fonctionner mais sans l'installer, à chaque démarrage de la machine, permettant d'avoir systématiquement un système homogène et mis à jour.

Dans notre cas, nous utilisons *CloneZilla* qui permet de restaurer des images disques (ou de partitions). Couplé avec le PXE, cela permet de faire du déploiement de masse : les serveurs démarrent, sont amorcés par PXE et lancent *CloneZilla* qui vient cloner une image pré-configurée (stockée et spécifiée au niveau du serveur de déploiement) sur le disque dur du serveur. Les images sont distribuées à travers un simple partage NFS.

Puppet est un outil de gestion de la configuration de serveurs, il permet le télédéploiement de configuration sur un ensemble de serveurs en quelques minutes, c'est lui qui automatisera l'installation et la configuration des noeuds de la plateforme. Puppet repose sur un système client/serveur et assure à tout instant que chaque noeud d'un certain type déployé conserve ses caractéristiques. Le serveur Puppet détient la configuration des types de noeuds et répond aux sollicitations des clients (tout noeud à installer ou demandant à vérifier son installation). Dès lors qu'un serveur a été cloné (avec *CloneZilla*), il redémarre et l'agent Puppet se lance, installe et configure la machine en fonction du nom d'hôte fournit par le *DHCP*.

Les noeuds de déploiement fournissent deux services supplémentaires, à savoir le DNS et le NTP. Le NTP permet à l'ensemble des serveurs de la plateforme de partager la même heure, précisément - ce qui est requis par un certain nombre de services, tout particulièrement lorsqu'un quorum est établis. L'utilisation d'un serveur NTP externe n'est pas idéal car la distance et le réseau engendre des temps de latence non constants qui ne permet pas de garantir que l'heure sera synchronisée partout. Le service DNS relaye les requêtes DNS des serveurs (traduction d'un nom d'hôte en adresse IP) : le fait de proposer notre propre service DNS et non pas un service DNS externe garantit le contrôle sur les informations offertes dans le réseau et offre la possibilité de donner des noms d'hôtes résolues aux serveurs et ainsi ne pas configurer la plateforme en utilisant des adresses IP mais uniquement des noms.

La façon la adaptée pour rendre ces noeuds hautement disponibles est la mise en place d'une redondance active/passive, avec par exemple *Pacemaker* - cela n'a pas été mis en oeuvre.

3.3 Réseaux

Le réseau est un élément crucial dans notre plateforme, non seulement parce qu'il permet d'interagir avec l'IaaS et les infrastructures crées, mais aussi pour des raisons de sécurité et de performances. En effet, qui dit accès dit risque. Ensuite, le réseau est un facteur important peut rapidement se transformer en goulet d'étranglement.

3.3.1 Réseaux de la plateforme

Notre plateforme repose sur cinq réseaux, détaillés ci-dessous et schématisés dans la Figure 3.11.

Réseau extérieur (API) C'est sur ce réseau que les répartiteurs de charge (HAProxy), présents sur les « Noeuds de balancement de charge & d'extrémité réseau

» écoutent. Tous les API permettant d'accéder à la plateforme sont donc exposés dans ce réseau, qui se veut ouvert au monde extérieur (Internet).

Réseau extérieur (IP flottantes) Disposant de nombreuses adresses IP (typiquement des blocs IP RIPE) réparties potentiellement sur plusieurs sous-réseaux, ce réseau sert à l'assignation d'adresses IPs publiques (aka IP flottante) aux machines virtuelles, routeurs virtuels, répartiteurs de charge virtuels, etc. Tout comme le précédent, c'est par définition un réseau ouvert au monde extérieur (Internet) et il est supporté intégralement par les « Noeuds de balancement de charge & d'extrémité réseau ».

Réseau de stockage Ceph Ce réseau interconnecte les « Noeuds de stockage » entre eux. Ceci afin de disposer d'un réseau isolé et dédié à la réplication de données et aux services de stockage. Ce réseau peut devenir crucial afin de garantir la sécurité des données mais aussi pour contrôler les performances de la plateforme étant donné l'ampleur des volumes transférés pour le stockage relativement aux données des API et des instances. Prévoir ce réseau dès maintenant permettra par exemple par la suite de réserver des liens physiques dédiés plus simplement.

Réseau de données (Tunnels GRE) C'est sur ce réseau que transitent les données émises et reçues par les instances. Des tunnels GRE (via Open vSwitch) sont établis entre chaque « Noeud de calcul » et chaque « Noeuds de balancement de charge & d'extrémité réseau » et transmettent le trafic isolé de couche 2 dans des paquets IPs. Cela permet d'une part d'isoler le réseau des consommateurs du réseau physique et d'autre part d'éviter d'avoir besoin de switchs configurés en trunk pour certaines plages de VLAN. Ce réseau n'est pas exposé au monde extérieur bien que les routeurs virtuels, gérés sur les « Noeuds de balancement de charge & d'extrémité réseau », peuvent donner un accès grâce au NAT et aux adresses IPs flottantes.

Réseau de contrôle Tous les serveurs de la plateforme y sont connectés, ce réseau est le réseau par défaut et sert à la communication générale et interne à la plateforme entre les différents noeuds. Ce réseau est par conséquent privé et n'est pas exposé au monde extérieur. Sur ce réseau transitent par exemple les requêtes entre les différents services OpenStack ou encore les données de Puppet. On note que ce réseau dispose entre autres de capacités DHCP/PXE offertes par les « Noeuds de déploiement ».

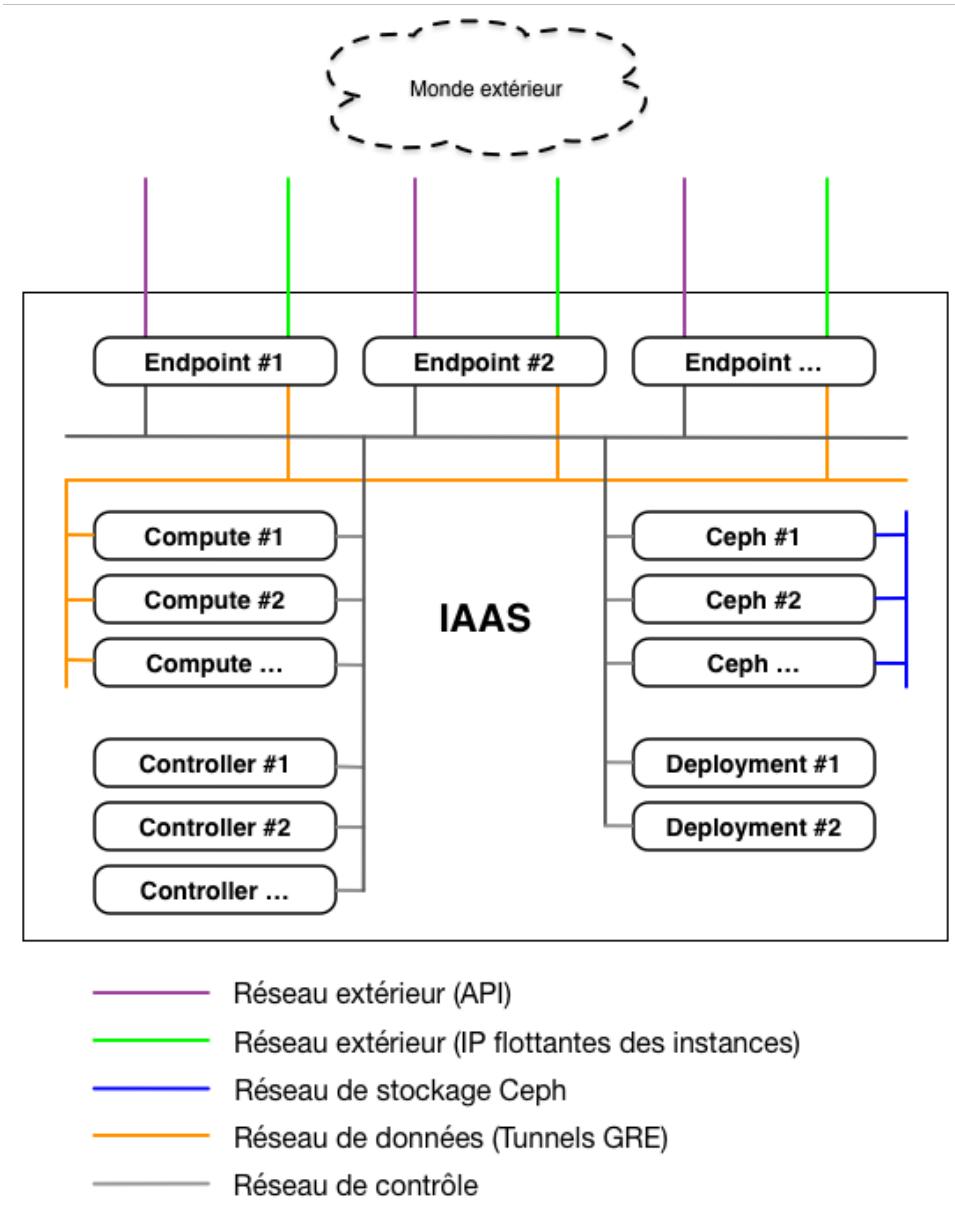

FIGURE 3.11 – Architecture - Réseaux

3.3.2 Réseaux des instances

Les consommateurs peuvent créer des réseaux, des sous-réseaux et des routeurs virtuels pour leurs infrastructures. Les machines virtuelles se voient assigner des adresses IPs dans les sous-réseaux à l'aide d'un service DHCP dédié et les clients peuvent également associer des adresses IPs flottantes (adresses IPs du « Réseau extérieur ») à chacune des des machines virtuelles, répartiteurs de charge, services VPNs, etc.

Du point de vue des consommateurs, l'infrastructure réseau est tout à fait classique : un exemple est montré à l'aide de la Figure 3.12. Le réseau externe et le routeur physique appartiennent à la plateforme tandis que les des réseaux privés, des routeurs virtuels et des machines virtuelles appartiennent aux consommateurs. On constate sur l'exemple que cer-

taines machines virtuelles ainsi que les routeurs disposent d'une adresse IP flottante, les rendant de fait disponibles sur le réseau externe (Internet).

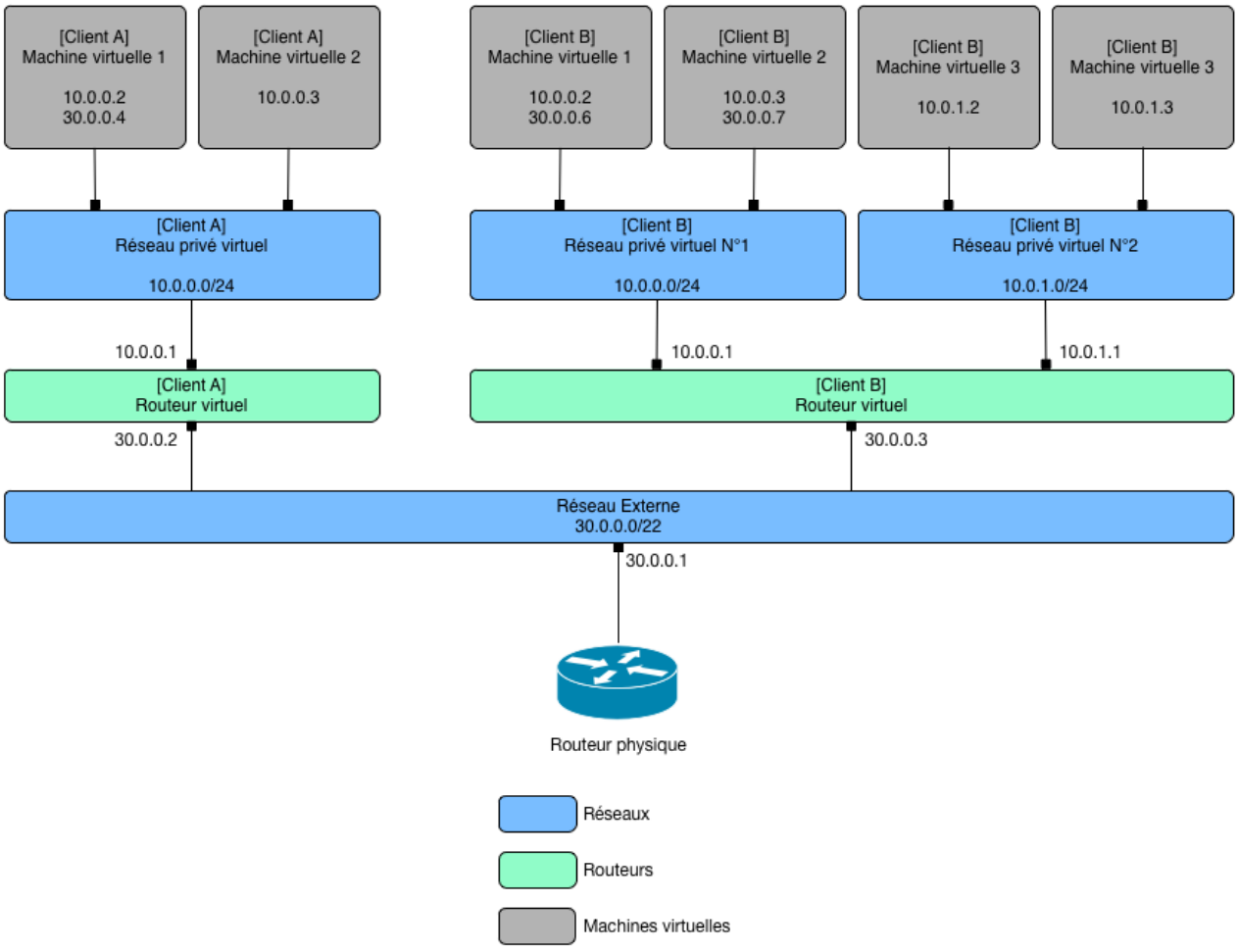

FIGURE 3.12 – Architecture - Réseaux vues par les clients

Maintenant, du point de vu de la plateforme, les choses se compliquent sensiblement. Tout d'abord, comprenons que chaque « Noeud de calcul » et chaque « Noeud de balancement de charge & d'extrémité réseau » sont interconnectés les uns aux autres à l'aide de tunnels GRE sur le réseau de données puis reprenons l'exemple présenté dans la Figure 3.12 en le simplifiant : deux clients ayant deux machines virtuelles chacun, deux sous-réseaux et deux routeurs (Figure 3.13).

Nous retrouvons les quatre machines virtuelles sur la Figure 3.14, ils sont dans notre cas tous regroupés sur un seul noeud. Lorsqu'une des machines virtuelles émet un paquet IP via son interface réseau *et*, il est d'abord transmit à un pont Linux *qbr* via un périphérique *tap* afin de subir le filtrage lié au groupe de sécurité auquel l'instance appartient. Le filtrage est effectué par iptables qui est configuré par *nova-compute*. Le paquet est ensuite transporté jusqu'au pont *Open vSwitch br-int* en passant par la paire *veth qvb, qvo* (nécéssaire pour

connecter deux ponts entre eux). Alors que jusqu'ici, le paquet était taggué dans un VLAN interne permettant de l'isoler des paquets des autres clients, l'identifiant du VLAN est traduit en identifiant de tunnel GRE lors du passage du paquet au pont *Open vSwitch br-tun*. Enfin, le paquet sort via *eth0* encapsulé dans un tunnel GRE.

Le paquet, acheminé jusqu'au « Noeud de répartition de charge et d'extrémité réseau » pénètre par l'interface *eth0* sur le pont *br-tun* sur la Figure 3.15. Inversement à ce qui se passait sur le « Noeud de calcul », le paquet passe du pont *br-tun* à *br-int*, ce qui traduit son identifiant de tunnel en identifiant de VLAN interne - le paquet est alors taggué. Finalement, le paquet transite jusqu'au routeur virtuel correspondant à son réseau en passant par le port *qr*. Le passage par ce port le fait rentrer dans l'espace de nom réseau correspondant à l'identifiant du routeur (*brouter-ddd*). Les espaces de noms réseaux Linux [5] permettent de disposer de plusieurs piles réseaux indépendantes et isolées disposant de routes, règles de pare-feu et périphériques réseaux - c'est grâce à celles-ci qu'il est possible de faire cohabiter plusieurs routeurs virtuels et services sur un même noeud. Finalement, selon la décision du routeur vis-à-vis du paquet, il peut soit se diriger vers le réseau extérieur (Internet) en subissant un NAT puis en passant par le port interne *qg*, le pont *br-ex* puis par *eth1*, soit se diriger à nouveau vers le réseau de données (pour atteindre une autre machine virtuelle par exemple) en empruntant le chemin inverse.

À cause de l'encapsulation des paquets IP des machines virtuelles dans les tunnels GRE, un paquet issue d'une machine virtuelle et remplit intégralement (1500 octets) sera fragmenté pour pouvoir être transporté par les tunnels, générant une charge importante et inutile de petits paquets. À cela, deux solutions : soit il faut augmenter le MTU des interfaces des serveurs (il faut aussi que les périphériques réseaux soient configurés en adéquations) soit il faut réduire le MTU des machines virtuelles. C'est cette deuxième solution qui est adoptée car elle se veut plus simple, engendre un impact plus faible sur le réseau existant et est conseillée dans la documentation OpenStack [12]. Nos tests ont montrés que jusqu'à 62 octets d'entêtes s'ajoutent, le MTU des machines virtuelles doit donc être de 1438. C'est les instances du service *dnsmasq* qui fournira cette informations aux machines virtuelles sur leurs sous-réseaux respectifs, en même temps que de leur fournir un adressage réseau, une passerelle, des serveurs DNS et un masque.

On note l'existence d'un espace de nom supplémentaire par sous-réseau, nommé *qdhcp-XXX* où *XXX* correspond à l'identifiant unique de sous-réseau. Ces espaces de noms existent pour supporter le service *dnsmasq* qui offre le DHCP dans le sous-réseau.

Depuis *OpenStack Juno*, il est possible de rendre les routeurs virtuels hautement-disponible [39, 40, 1]. Lors de la création d'un nouveau routeur virtuel, celui-ci est en réalité crée plusieurs fois (il est possible de paramétrer ce nombre) et ordonnancé sur plusieurs noeuds. Cet ordonnancement assure une certaine forme de balancement de charge en répartissant les routeurs virtuels. Lors de la défaillance d'un noeud et par conséquent de ses routeurs virtuels, le protocole *Virtual Router Redundancy Protocol* (VRRP) assure le basculement rapide de l'ensemble des routeurs virtuels gérés, sur d'autres noeuds et garantit la disponibilité du réseau des instances. C'est les services KeepAlived (visibles dans la Figure 3.15) qui propulse ce protocole. Chaque client dispose d'un réseau *HA* dédié à cela et autant d'instances de *KeepAlived* qu'il

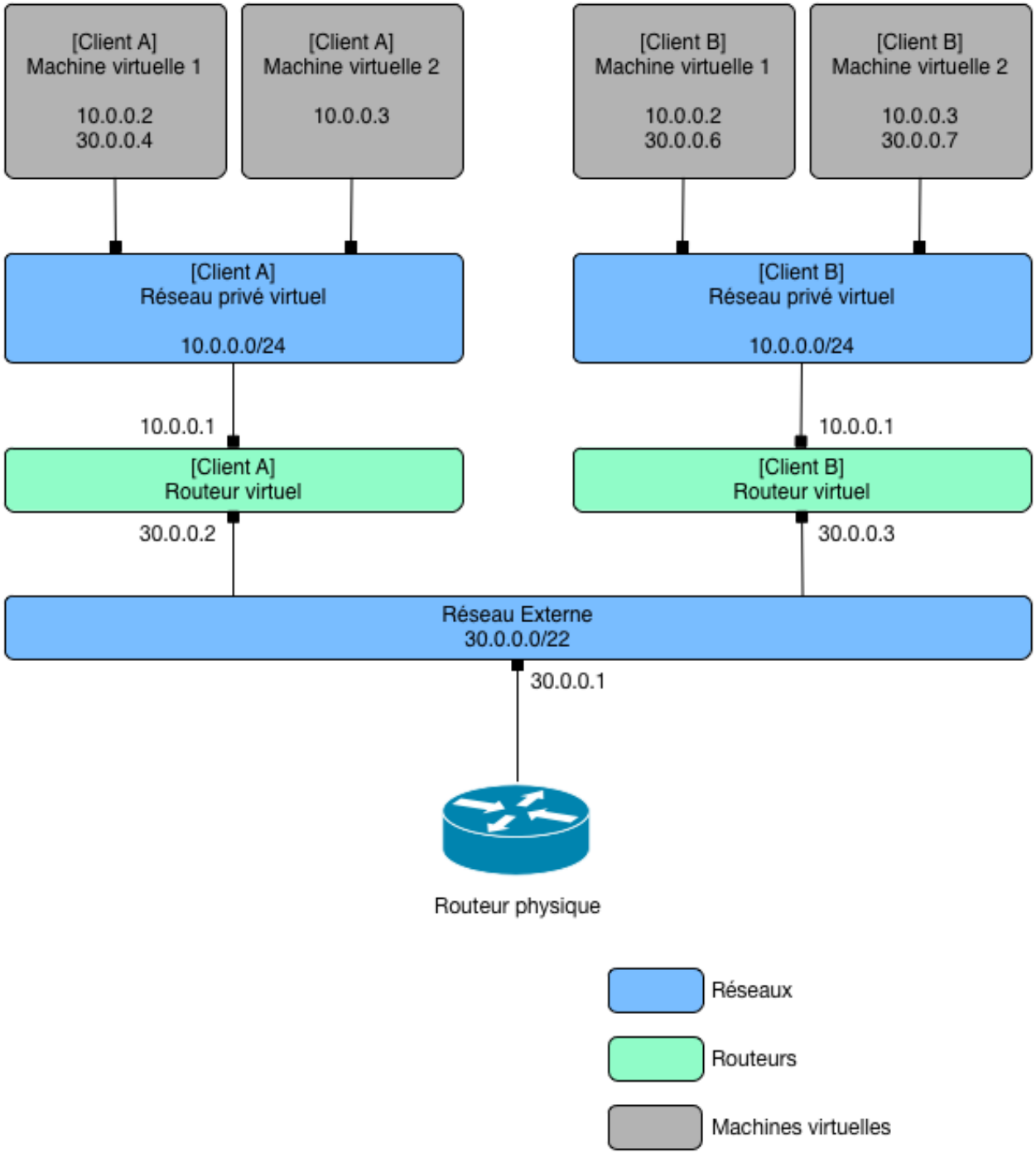

FIGURE 3.13 – Architecture - Réseaux vues par les clients (Simplifié)

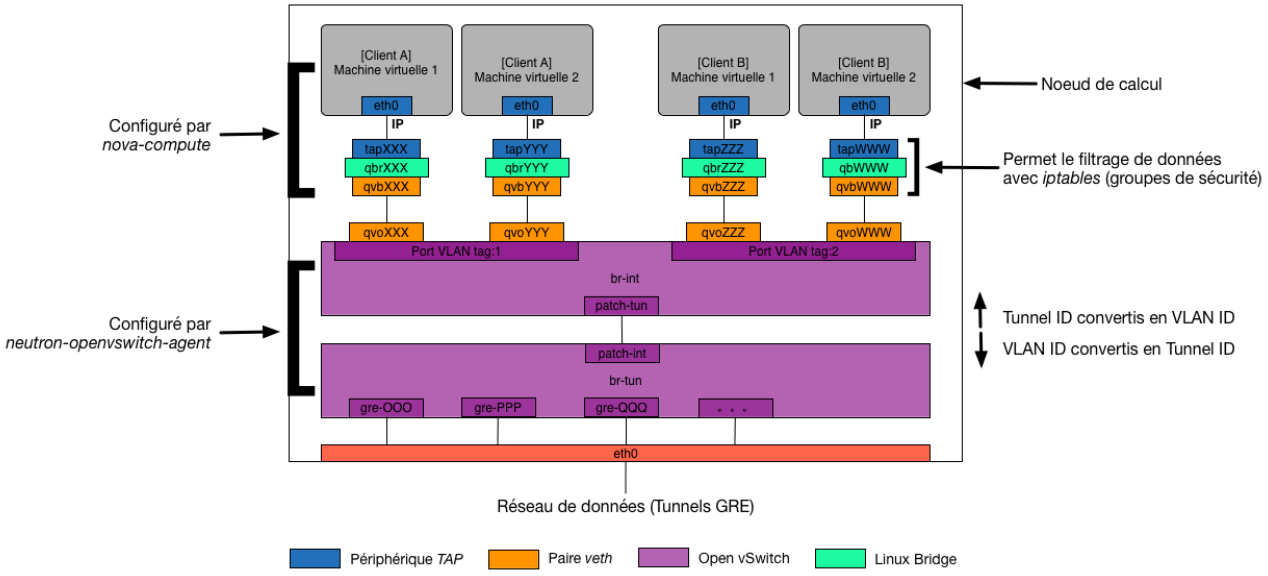

FIGURE 3.14 – Architecture - Réseaux sur un noeud de calcul

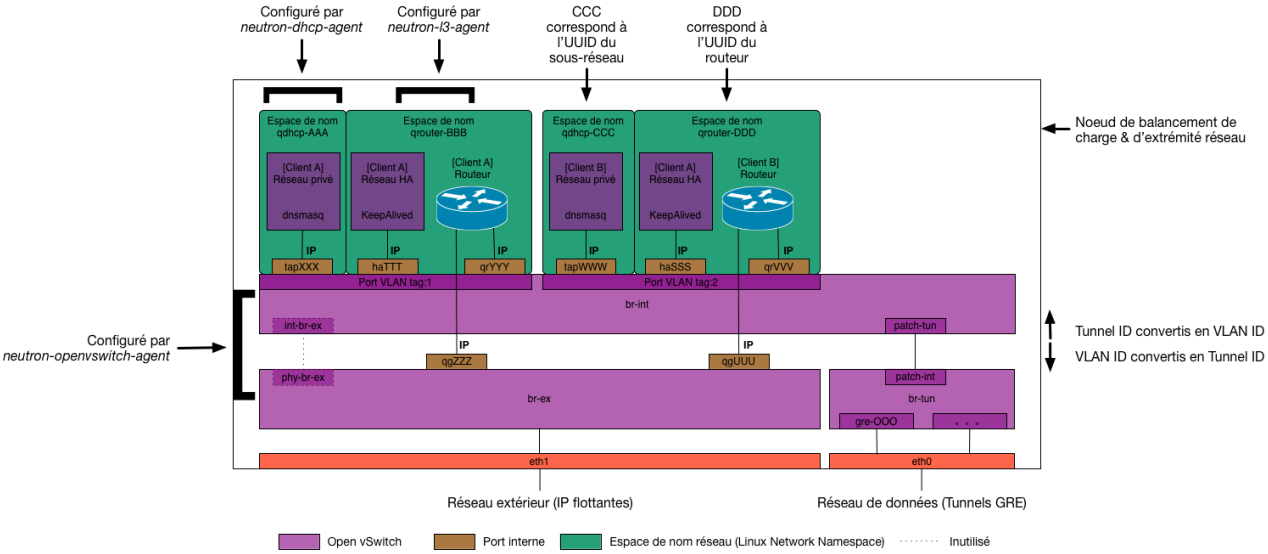

FIGURE 3.15 – Architecture - Réseaux sur un noeud d'extrémité réseau

n'y a de routeurs virtuels déclarés (4 instances pour 2 routeurs avec une réplication de 2). On note que *OpenStack Juno* introduit également une fonctionnalité primordiale nommée *Distributed Virtual Router* (DVR) re-localisant la tâche de routage directement sur les noeuds de calcul, ce qui évite la création du potentiel goulet d'étranglement au niveau des noeuds d'extrémité réseau, mais cette fonctionnalité est incompatible actuellement avec la notion de haute disponibilité [42].

Notons finalement que les figures ci-dessous ne représentent pas l'interface permettant d'accéder au réseau de contrôle.

3.4 Perspective

La catégorisation des noeuds en types et la répartition des services présentés ci-dessus définissent un support minimal à la haute disponibilité et à l'évolution horizontale mais n'excluent pas la possibilité future d'extraire certains services des types et de créer de nouveaux types. On peut ainsi imaginer séparer *haproxy* et les services *Neutron* si les charges de l'un ou de l'autre sont sensiblement différent, ou bien encore séparer *ceph-osd* de *ceph-mon* : avec par exemple des baies serveurs dédiés avec simplement un serveur faisant office de moniteur (*ceph-mon*) et plusieurs serveurs purement de stockage (*ceph-osd*).

Déploiement

Dans ce chapitre sera détaillé la façon dont l'infrastructure est déployée et les différentes astuces permettant son bon fonctionnement.

4.1 Installation matérielle

Pour commencer, parmi les douze serveurs mis à disposition par *Polytech Tours*, onze sont utilisées pour la plateforme. Un switch réseau est également à disposition. Les serveurs ne disposent que d'une interface réseau et d'un seul disque dur.

Les serveurs sont répartis parmi les différents types de noeuds définis dans la Section 3.2. Trois sont assignés en noeud de stockage (nombre minimal pour assurer le quorum de *Ceph*), trois autres en noeud de contrôle (nombre minimal pour assurer le quorum de *MySQL Galera*), deux noeuds de calculs (ce qui permet de vérifier le bon fonctionnement de la répartition des machines virtuelles, de la communication entre plusieurs machines virtuelles et de l'évacuation d'instances pour la mise en maintenance), deux noeuds de balancement de charge & d'extrémité réseau (pour tester la haute disponibilité de routeurs virtuels notamment) et un noeud de déploiement. Un douzième serveur serait nécéssaire en production pour assurer la disponibilité du noeud de déploiement. Nous ne rendons pas hautement disponible le noeud de déploiement dans cette plateforme, ce qui devrait se faire avec *DRBD+Pacemaker* car ce noeud n'est pas critique, que le défi que représente cette mise en oeuvre est mineur et que le sujet a déjà été traité en interne chez Harmony-Hosting par le passé. Le déploiement est schématisé dans la Figure 4.1.

4.2 Automatisation de la procédure d'installation du système

La première étape dans la mise en oeuvre de la plateforme est l'automatisation des installations serveurs avec système d'exploitation, adressage réseau et nom d'hôte. Cela s'inscrit dans le cadre de l'objectif *O13*.

4.2.1 Services de déploiement

Pour permettre l'installation de serveurs vierges, il est nécéssaire d'installer les services DHCP, PXE et TFTP et puppetmaster sur le noeud de déploiement. Pour cela, on se réfère aux documents [22, 21].

La configuration du serveur DHCP est simple et sans ambiguité, un extrait de configuration est présenté dans la Figure 4.2. Il configure ainsi les serveurs sur le réseau de contrôle

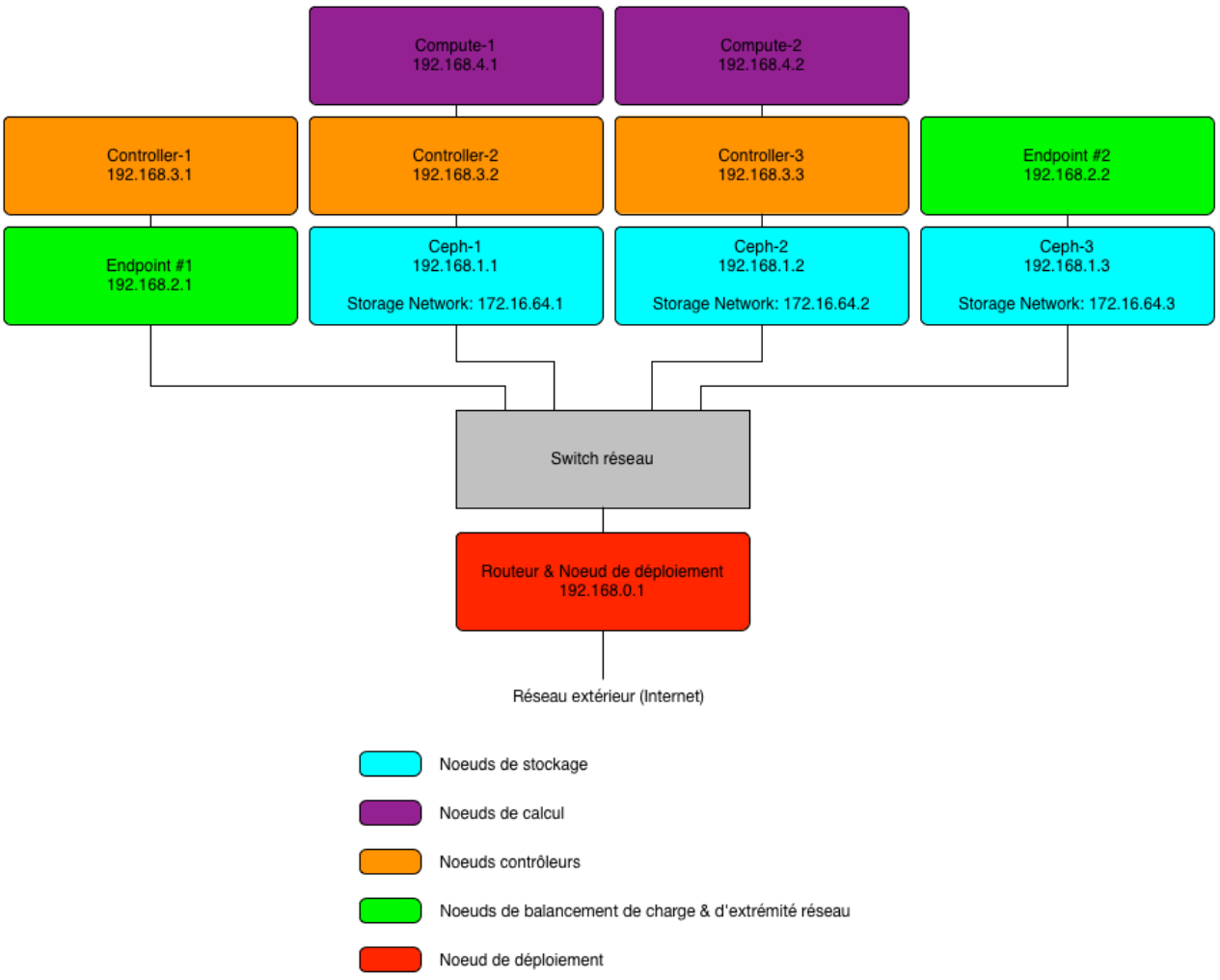

FIGURE 4.1 – Déploiement - Matériel

en fournissant fournit l'adresse du routeur (ici la même que le noeud de déploiement puisque les deux rôles sont regroupés dans notre plateforme de démonstration), le masque de sous-réseau, le serveur DNS à utiliser (le noeud de déploiement), qui résoudra les noms pour la plateforme, mais aussi l'adresse du serveur TFTP pour le PXE (*next-server*). On a ensuite la définition de deux contrôleurs. On constate qu'il faut qu'il y ait correspondance entre l'adresse Ethernet du serveur et une entrée de configuration DHCP pour que l'adresse IP et le nom d'hôte soit attribué - ce sont des baux statiques. Les serveurs qui ne sont pas définis explicitement sont attribués dans une courte plage identifiable, sans support du PXE. Tandis que le premier noeud contrôleur définit ne démarrera pas avec le PXE, le deuxième noeud, lui démarrera par PXE. De cette façon, il est possible de contrôler simplement quels serveurs sont à ré-installer et lesquels ne sont pas à réinstaller.

Deux images sont initialement stockées sur le TFTP :
— Ubuntu Trusty 64 bits, qui est le système d'exploitation utilisé pour la plateforme, et qui va servir de base pour créer une image d'installation (voir section suivante) pour l'ensemble du parc.
— Clonezilla pour la création d'image d'installation ou la copie d'une image sur un serveur

La configuration PXELinux dispose donc de 4 entrées (voir Figure 4.3) :
— Le clonage automatique sur disque de l'image utilisée dans la plateforme à l'aide de Clonezilla. L'image est stockée sur un partage NFS sur le serveur de déploiement.
— Le lancement de CloneZilla en interactif pour la création d'images.
— Ubuntu Trusty 64 bits pour installer un système normalement (en vu de créer une image)
— Le démarrage sur disque standard

4.2.2 Création d'une image

La procédure de création d'une nouvelle image à cloner est relativement simple mais nécessite souvent plusieurs ajustements ce qui en fait une tâche relativement chronophage. Il faut tout d'abord installer *Ubuntu Trusty* sur un serveur (on peut le faire à l'aide d'un démarrage PXE en choisissant l'item dans le menu).

Pour que Ceph puisse utiliser une partition plutôt qu'un disque, il faut s'assurer que la table de partition soit en GPT (voir Section 4.5) plutôt que en MBR. Sur des serveurs ne disposant pas de l'UEFI mais que d'un BIOS, cela pose problème car GRUB ne dispose alors plus de l'espace nécéssaire pour pouvoir s'installer totalement. Il faut donc créer une partition spéciale de 1M de type *bios_grub* dans les deux premiers TiB du disque pour résoudre [25].

Une fois que l'installation du système sur table GPT a été réalisée correctement, il est nécéssaire de développer un *hook DHCP* permettant à la machine de s'attribuer de façon permanente le nom d'hôte indiqué par DHCP. En effet, par défaut, Ubuntu n'attribue le nom d'hôte offert pas DHCP qu'uniquement si la machine n'en dispose pas déjà ou bien est nommé "localhost" et de façon temporaire (Voir */sbin/dhclient-script* sur Ubuntu Trusty). On vient alors créer un fichier */etc/dhcp/dhclient-exit-hooks.d/hostname* qui sera exécuté suite au fonctionnement normal du client DHCP, le contenu est décrit dans la Figure 4.4 [45].

```
ddns-update-style none;

subnet 192.168.0.0 netmask 255.255.192.0 {
    # Unknown clients will get IP in this range
    range 192.168.0.10 192.168.0.254;

    # Network
    option domain-name "iaas";
    option routers 192.168.0.1;
    option subnet-mask 255.255.192.0;
    option domain-name-servers 192.168.0.1;
    option broadcast-address 192.168.63.255;

    # PXE
    next-server 192.168.0.1;
}

group {
    # Set hostnames from DNS response on the fixed-address
    # get-lease-hostnames true;

    # To enable PXE, add : filename "pxelinux.0";

    ######################################
    # CONTROLLER
    ######################################
    host controller-1 {
        hardware ethernet 08:2e:5f:01:26:14;
        fixed-address 192.168.3.1;
        option host-name "controller-1";
    }

    host controller-2 {
        hardware ethernet 08:2e:5f:05:58:42;
        fixed-address 192.168.3.2;
        option host-name "controller-2";

        filename "pxelinux.0";
    }

    [...]
}
```

FIGURE 4.2 – Déploiement - Configuration DHCP sur le noeud de déploiement

```
timeout 300
DEFAULT vesamenu.c32

label Clonezilla - Clone : Golden-1
  kernel clonezilla/live/vmlinuz
        append initrd=clonezilla/live/initrd.img boot=live config noswap
           nolocales edd=on nomodeset ocs_live_run="ocs-live-restore"
           ocs_live_extra_param="-g auto -e1 auto -e2 --batch -j2 -p
           reboot restoredisk golden-1 sda" ocs_prerun="mount -t nfs
           192.168.0.1:/var/lib/tftpboot/clonezilla/images /home/partimag
           " keyboard-layouts="fr" ocs_live_batch="yes" locales="en_US.
           UTF-8" vga=788 nosplash noprompt fetch=tftp://puppet.iaas/
           clonezilla/live/filesystem.squashfs

label Clonezilla - Live
        kernel clonezilla/live/vmlinuz
        append initrd=clonezilla/live/initrd.img boot=live config noswap
           nolocales edd=on nomodeset ocs_live_run="ocs-live-general"
           ocs_live_extra_param="" keyboard-layouts="" ocs_live_batch="no
           " locales="" vga=788 nosplash noprompt fetch=tftp://puppet.
           iaas/clonezilla/live/filesystem.squashfs

label Trusty 64 bits installer
        kernel trusty/ubuntu-installer/amd64/boot-screens/vesamenu.c32
        append trusty/ubuntu-installer/amd64/boot-screens/menu.cfg

label Boot local disk
        kernel chain.c32
        append hd0
```

FIGURE 4.3 – Déploiement - Configuration PXELinux sur le noeud de déploiement

```
#!/bin/sh
# Filename:       /etc/dhcp/dhclient-exit-hooks.d/hostname
# Purpose:        Used by dhclient-script to set the hostname of the system
#                 to match the DNS information for the host as provided by
#                 DHCP.

if [ "$reason" != BOUND ] && [ "$reason" != RENEW ] \
   && [ "$reason" != REBIND ] && [ "$reason" != REBOOT ]
then
        return
fi

echo dhclient-exit-hooks.d/hostname: DHCP IP address = $new_ip_address
echo $new_host_name > /etc/hostname
hostname $new_host_name
echo dhclient-exit-hooks.d/hostname: DHCP Hostname = $new_host_name

chmod a+r /etc/dhcp/dhclient-exit-hooks.d/hostname
```

FIGURE 4.4 – Déploiement - Hook DHCP pour l'attribution permanence de nom d'hôte (*/etc/dhcp/dhclient-exit-hooks.d/hostname*)

On installe ensuite puppet, facter (qui permet de récupérer diverses valeurs pour puppet), activerecord (pour faire de l'exportation de ressources, voir Section 4.3), on corrige une erreur de Puppet, on configure le tout, on arrête puppet qui a dû démarré automatiquement et on supprime les certificats qu'il a généré (on fait de même pour le serveur SSH). De sorte que dès le prochain redémarrage de la machine (après un clonage donc), puppet démarrera, générera de nouveaux certificats uniques et réalisera le déploiement du système. Cette procédure est décrite dans la Figure 4.5.

Le système est maintenant prêt et il suffit désormais de créer l'image. Après avoir pris soin de vider les logs, il faut redémarrer le système et démarrer via PXE le système *Clonezilla*. Bien qu'il soit possible d'utiliser Clonezilla de façon interactive, les commandes décrite dans la Figure 4.6 permettent de réaliser rapidement l'image du disque et nomme l'image « golden-1 » et de la stocker sur le partage NFS du noeud de déploiement en vu de son clonage.

4.2.3 Procédure d'installation de serveurs

Le serveur a installé doit être amorcé en *PXE*. Pour cela, il faut tout d'abord que le serveur soit renseigné dans la configuration du serveur DHCP, avec son adresse Ethernet et son futur nom d'hôte. La ligne *filename "pxelinux.0";* doit également être présente dans la configuration afin que le serveur soit amorçable (voir Figure 4.2). Il faut également que la carte réseau soit support de démarrage prioritaire le *BIOS* ou l'*UEFI*. Ainsi, lors du démarrage, le serveur est lancé sur le menu de démarrage PXE où le choix des systèmes qu'il est possible de lancer est proposé. Comme spécifié, le choix par défaut est le clonage de l'image *Golden-1* sur le disque dur et ce choix se valide automatiquement après 30 secondes (configurable).

```
# Install Puppet
wget http://apt.puppetlabs.com/puppetlabs-release-trusty.deb
dpkg -i puppetlabs-release-trusty.deb
apt-get update && apt-get install -y puppet facter
# Fix US-ASCII Bug
cp /usr/share/puppet/ext/rack/config.ru /usr/share/puppet/ext/rack/config
    .ru.backup && awk '/puppet\/util\/command_line/ { print "Encoding.
    default_external = Encoding::UTF_8\n"; print; next }1' /usr/share/
    puppet/ext/rack/config.ru.backup > /usr/share/puppet/ext/rack/config.
    ru && rm /usr/share/puppet/ext/rack/config.ru.backup
gem install activerecord -v 3.0.10

# Configure Puppet
echo -e "path /run\nmethod save\nallow *" > /etc/puppet/auth.conf
echo -e "[puppetrunner]\nallow *\n[puppetbucket]\nallow *\n[puppetreports
    ]\nallow *\n[resource]\nallow *\n[kick]\nallow *\n" > /etc/puppet/
    namespaceauth.conf
sed -i /etc/default/puppet -e 's/START=no/START=yes/'
mv /etc/puppet/puppet.conf /etc/puppet/puppet.conf.backup && awk '/\[main
    \]/ { print; print "pluginsync = true"; next }1' /etc/puppet/puppet.
    conf.backup > /etc/puppet/puppet.conf && rm /etc/puppet/puppet.conf.
    backup

# Stop & Reset Puppet certificates
service puppet stop
rm -f /etc/puppetlabs/puppet/ssl/*

# Stop & Reset SSH certificates
service sshd stop
rm /etc/ssh/ssh_host_*
ssh-keygen -q -N "" -t dsa -f /etc/ssh/ssh_host_dsa_key
ssh-keygen -q -N "" -t rsa -f /etc/ssh/ssh_host_rsa_key
ssh-keygen -q -N "" -t ecdsa -f /etc/ssh/ssh_host_ecdsa_key
```

FIGURE 4.5 – Déploiement - Installation initiale du système (puppet, facter, activerecord)

```
sudo su
mount -t nfs puppet.iaas:/var/lib/tftpboot/clonezilla/images /home/
    partimag
ocs-sr -q2 -c -j2 -gs -z0 -i 1000000000000 -fsck-src-part -sc -p true
    savedisk golden-1 sda
```

FIGURE 4.6 – Sauvegarde rapide du disque dur en image avec Clonezilla

Lorsque le système cloné démarre, une requête *DHCP* est envoyée et une réponse contenant le nom d'hôte, l'adresse IP ainsi que toute la configuration réseau est envoyée par le serveur de déploiement, en fonction de l'adresse *Ethernet* du serveur.

Puppet démarre ensuite et grâce au nom d'hôte détermine le rôle du serveur. Il installe et configure le serveur en conséquence, de façon totalement automatique. Toutes les 30 minutes, Puppet se lancera à nouveau pour mettre à jour la configuration.

D'après l'expérience acquise, la procédure complète prend approximativement 3 minutes entre le moment où le bouton d'alimentation est pressé et le moment où le serveur a redémarré et Puppet est lancé.

4.3 Puppet

La troisième étape dans le déploiement de la plateforme est le développement Puppet gérant l'installation de l'ensemble des services de OpenStack et leurs configurations.

Puppet s'organise autour de modules qui sont des dossiers contenant chacun un ensemble de classes, de définitions de ressources et de gabarits, et ayant chacun un but précis. Chaque module se veut partageable, ré-utilisable, limité à sa fonction précise et peut se reposer sur d'autres modules. De nombreux modules existent et sont partagés sur Puppet Forge [1]. Dans le cadre de ce projet, un seul module est développé et s'appuie sur différents modules, en particulier ceux de *StackForge* [2]. Il existe plusieurs façons de définir des dépendances et de les télécharger, dans notre cas, nous utilisons *librarian-puppet* [3] car il ne dispose pas de résolutions de dépendance avancée ni de gestion de conflits : il s'est avéré que divers modules utilisent des définitions de dépendances abusives ou incorrectes ce qui générait de nombreuses erreurs si nous en tenions compte.

Pour organiser le module, on utilise le *design pattern* « Roles/Profiles » [33, 34] qui permet de définir un certain nombre de rôles (correspondant aux types de noeuds définis dans la Section 3.2) composés de profiles (correspondant aux différents services à configurer).

La plateforme dispose d'un certain nombre de paramètres et de données spécifiques (adressage réseau, etc) qui peuvent changer d'un déploiement à un autre. Au lieu d'intégrer naïvement ces données dans le module, on utilise le système *Hiera* qui permet de stocker les données sous forme de clé/valeur dans des fichiers *YAML* de façon hiérarchisée tout en offrant la possibilité de les crypter (mots de passe par exemple). La hiérarchie utilisée pour la plateforme est la suivante : les données sont cherchées initialement dans le fichier *common.yaml* qui contient toutes les valeurs génériques, puis dans *nodes/% : :fqdn.yaml* c'est-à-dire le fichier correspondant au nom d'hôte du serveur sur lequel s'exécute Puppet

1. https ://forge.puppetlabs.com
2. http ://ci.openstack.org/stackforge.html
3. http ://librarian-puppet.com

```
Exec {
  path => '/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin'
}

node /^ceph-\d+.iaas$/ {
  include 'iaas::role::storage'
}

node /^endpoint-\d+.iaas$/ {
  include 'iaas::role::endpoint'
}

node /^controller-\d+.iaas$/ {
  include 'iaas::role::controller'
}

node /^compute-\d+.iaas$/ {
  include 'iaas::role::compute'
}
```

FIGURE 4.7 – Déploiement - Point d'entré Puppet

lors de la recherche.

Une autre *design pattern* très intéressante est l'exportation de ressources [20]. Il permet à des noeuds de partager des informations avec d'autres. Ce patron de conception est utilisé abondamment pour la gestion de *HAProxy* sur la plateforme. Ni Puppet ni les noeuds de déploiement ne connaissent la liste de l'ensemble des noeuds dont il faut prendre en charge les services par *HAProxy* et maintenir une telle liste serait particulièrement lourd. L'exportation de ressources répond à ce problème. Dans le profil correspondant à HAProxy, les différents services dont HAProxy est autorisé à gérer est établie, sans savoir en particulier quels serveurs disposent de ces services, chaque noeud assigné à ce profil "écoute" alors à l'affût d'informations concernant la localisation de serveurs offrant ces services. D'un autre côté, chaque noeud assigné à un des profils offrant un des dits services exporte l'information *"Je suis le noeud adressé 192.168.X.Y et je propose le service ZZZ, sur le port UUU avec les options VVV"*. Entre d'autres termes, chaque noeud gérant certains services s'enregistre auprès de HAPROXY pour s'inscrire dans la répartition de charge.

Enfin, le manifeste principale *site.pp* sert de point d'entré à Puppet. Dans notre cas, le fichier est très simple, il détermine quel role assigné à quel machine en fonction de son nom d'hôte. Afin d'éviter de saisir chaque nom d'hôte un par un, des expressions régulières sont utilisées. Ainsi par exemple tous les serveurs nommés *ceph-XXX.iaas* où *XXX* est un nombre seront des noeuds de stockage.

Afin de faciliter le développement, les noeuds et les services sont développés, installés, vérifiés dans un ordre précis. Cet ordre est le même que l'ordre dont les noeuds devraient être

déployés en partant de zéro. Cet ordre dépend des dépendances entre les services. Une fois l'infrastructure disposant de suffisamment de noeuds pour assurer l'ensemble des services et des quorum, l'ordre n'a plus d'importante.

1. Noeud de déploiement
2. Noeud de balancement de charge & d'extrémité réseau : HAProxy uniquement
3. Noeud de stockage
4. Noeud de contrôle :
   (a) Services de bases de données
   (b) Services de messages
   (c) Service d'authentification (Keystone)
   (d) Services d'images (Glance) et de volumes (Cinder)
   (e) Service réseau (Neutron)
5. Services réseau (Neutron)
6. Services d'instances (Nova)
7. Service Horizon
8. Services de télémétrie (Ceilometer)
9. Services d'orchestration (Heat)

Une fois le module Puppet développé, les serveurs installés à l'aide de la procédure d'installation automatique du système et disposant de noms d'hôtes adéquates, le déploiement est extrêmement simple et ne nécessite aucune intervention humaine particulière : dès lors que les serveurs sont sous-tensions et que Puppet est démarré (et il est lancé automatiquement) : les serveurs sont déployés.

4.4 Création des ressources initiales

Suite au déploiement et aux tests préliminaires confirmant que les APIs sont opérationnelles, on vient déclarer (Figure 4.8) le réseau extérieur dans Neutron ainsi qu'un sous-réseau où des IPs flottantes pourront être utilisées. Aussi, une image ultra légère : CirrOS [4] (39 Mo) pour l'installation de machines virtuelles est ajoutée (Figure 4.9) pour au moins pouvoir tester la création d'instances et ce qui s'en suit. L'image est initialement au format qcow2 mais il est nécéssaire de la convertir au format raw pour qu'elle soit supportée par le couple Glance+Ceph.

4.5 Adaptations

Parce qu'il n'est pas possible ni autorisé d'exposer des services ou des machines (IP flottante) sur le réseau extérieur (réseau de l'Université) et à la vue des contraintes matérielles, on se rend immédiatement compte qu'il ne sera pas possible, de déployer les cinq réseaux tel

4. https ://launchpad.net/cirros

```
neutron net-create ext-net --router:external --provider:physical_network
    external --provider:network_type flat
neutron subnet-create ext-net --name ext-subnet --allocation-pool start
    =192.168.50.1,end=192.168.50.255 --disable-dhcp --gateway 192.168.0.1
    192.168.0.0/18
```

FIGURE 4.8 – Déploiement - Création du réseau initial

```
wget http://download.cirros-cloud.net/0.3.3/cirros-0.3.3-x86_64-disk.img
qemu-img convert -f qcow2 -O raw cirros-0.3.3-x86_64-disk.img cirros
    -0.3.3-x86_64-disk.dsk

source env/glance
glance image-create --name='CirrOS' --is-public=true --container-format=
    bare --disk-format=raw < cirros-0.3.3-x86_64-disk.dsk
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | 0590d15336f919496ccc91b2c0f667bc     |
| container_format | bare                                 |
| created_at       | 2015-02-12T16:15:43                  |
| deleted          | False                                |
| deleted_at       | None                                 |
| disk_format      | raw                                  |
| id               | 5d912738-72df-4379-add1-de88d80d012a |
| is_public        | True                                 |
| min_disk         | 0                                    |
| min_ram          | 0                                    |
| name             | CirrOS                               |
| owner            | 58abbf4baa174b239fa634adf95ae479     |
| protected        | False                                |
| size             | 41126400                             |
| status           | active                               |
| updated_at       | 2015-02-12T16:15:56                  |
| virtual_size     | None                                 |
+------------------+--------------------------------------+
```

FIGURE 4.9 – Déploiement - Ajout de l'image CirrOS

que décrit dans l'architecture réseau en Section 3.3.1. L'enjeu est de taille puisqu'il s'agit de respecter un des objectifs du projet (Tableau 1.1, O00) concernant l'utilisation de matériel d'usage commun. Par conséquent, bien que les modules *Puppet* sont développés en vu de déployer une infrastructure concordant strictement à l'architecture idéale, nous configurerons les modules pour confondre tous les réseaux en un seul. Cela ne change rien dans le cadre d'un déploiement de démonstration tel que celui-ci car l'infrastructure n'est pas suffisamment grande pour bénéficier des avantages que l'architecture réseau présente et aucun problème de sécurité ne peut survenir (personne n'interroge la plateforme sauf moi-même et aucun client réel n'est sur la plateforme). Une simple configuration dans les fichiers *Puppet* adéquates suffisent en général, à l'exception des « noeuds de balancement de charge et d'extrémité réseau » qui cumulent 4 réseaux sur une seule interface réseau et où *Open vSwitch* réalise un important travail de routage et de SNAT/DNAT. Pour ces noeuds ci, il est nécéssaire de manipuler plus amplement la configuration réseau des interfaces et des ponts - manipulations automatisées par *Puppet*. La possibilité de réaliser une telle chose montre bien la capacité à réaliser l'objectif O00, sous réserve que les tests soient fonctionnels.

Aussi, une attention particulière a été portée sur l'architecture réseau des « noeuds de stockage ». En effet, alors que tous les autres types de noeuds pourraient profiter des interfaces comme un moyen de séparer physiquement les réseaux, les « noeuds de stockage » pourraient (et devraient) profiter des interfaces physiques à disposition pour augmenter la bande passante disponible pour le transfert de données. D'autant plus que en réalité, le *réseau de stockage Ceph* sert à Ceph en interne pour son transfert de données tandis que le *réseau de contrôle* sert à Ceph pour communiquer les données avec le reste de la plateforme : les deux réseaux servent au transfert massif de données. Par conséquent, il devient tout à fait justifié de réaliser une agrégation de liens [24, 52] sur ces deux réseaux surtout si plusieurs de ces serveurs sont regroupés physiquement : on pourrait alors ensuite profiter de routeurs/switchs intermédiaires pour réaliser la séparation. La solution la plus adéquate permettant de disposer de deux réseaux isolés, sur une même interface tout en permettant de faire de l'agrégation de liens est l'utilisation de VLAN avec le module noyau Linux 8021q [53]. On vient alors définir une nouvelle interface logique tagguée et avec une configuration réseau différente (pour le réseau de stockage Ceph). Ainsi, si plusieurs interfaces réseaux sont disponibles, nous aurions simplement qu'à réaliser une agrégation et de re-créer l'interface logique tagguée dessus [36]. La configuration réseau en place (une seule carte réseau, deux réseaux : réseau général et réseau de stockage) sur un des « noeuds de stockage » est montrée ci-dessous dans la Figure 4.10.

Finalement, Ceph est conçu pour être utilisé sur des disques durs lui étant dédiés, et non pas sur de simples partitions. Malheureusement, ne disposant que de serveurs munis d'un seul disque, il n'est pas possible de dédier un disque à Ceph puisqu'il est forcément utilisé pour le système. Il est cependant possible pour Ceph d'utiliser une partition plutôt qu'un disque moyennant quelques opérations manuelles (que l'on automatise) et une table de partition GPT plutôt que MBR [18].

```
root@ceph-3:# cat /etc/network/interfaces
auto eth0 eth0.8 lo
allow-hotplug eth0 eth0.8

iface eth0 inet static
address 192.168.1.3
netmask 255.255.192.0

iface eth0.8 inet static
address 172.16.64.3
netmask 255.255.252.0

iface lo inet loopback

root@ceph-3:# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref
Use Iface
0.0.0.0         192.168.0.1     0.0.0.0         UG    0      0
0 eth0
172.16.64.0     0.0.0.0         255.255.252.0   U     0      0
0 eth0.8
192.168.0.0     0.0.0.0         255.255.192.0   U     0      0
0 eth0
```

FIGURE 4.10 – Déploiement - Configuration réseau d'un noeud de stockage

Tests

Les tests réalisés au cours de ce projet permettent de s'assurer du fonctionnement du déploiement - et dans le cas contraire : de le corriger. Par ailleurs, les tests sont rapprochés des objectifs afin de les valider. On attend à ce qu'un maximum de tests soient automatisés et que les seules erreurs existantes soient insignifiantes (faible incidence), liées aux contraintes de la plateforme de démonstration (matériel, règles universitaire pour le réseau) ou à un effet indépendant de notre volonté et responsabilité mais allant se résoudre dans un futur proche (anomalie dans une dépendance logicielle par exemple).

5.1 Tests préliminaires

Lors de la phase de développement, il est important de réaliser divers tests relativement basiques permettant de vérifier le fonctionnement des services mais surtout pour découvrir ce qui ne fonctionne pas - et pourquoi. Dans un premier temps, l'exécution de commandes simples telles que la liste de ressources ou encore la création de ressources (indépendantes de tout autre service) auprès des services est essentielle dans un premier temps et permet la génération de logs dont la lecture permet d'identifier la source des potentielles erreurs.

Plusieurs scripts fut développés et représentèrent de premières briques très basiques permettant de vérifier le fonctionnement de Ceph, des serveurs de bases de données (et leurs quorum), de HAProxy ainsi que de faire des essais primitifs sur les services Keystone, Glance, Cinder, Nova et Neutron.

Ces tests étaient suffisants pour les quelques premières semaines de développement et de déploiement afin de réaliser quelques vérifications simples mais absolument non adapté à une vérification exhaustive et optimale de la plateforme.

5.2 Tests d'intégration Tempest

Une fois la plateforme déployée, il devient alors possible d'utiliser Tempest qui réalise pas moins de 1500 tests d'intégration sur l'ensemble des services disponibles, comprenant des tests via les API, via clients console, l'exécutions de scénarios complexes et la soumission à la plateforme à de fortes charges (pour mettre en évidence d'éventuelles *race conditions*). Tempest se veut le plus exhaustif possible et teste également de nombreux cas erronés, improbables ou très rares. Dans ses tests, il comprends également des manipulations incohérentes et s'attend à ce que OpenStack renvois les codes erreurs appropriés.

Pour lancer la procédure de test, on utilise les commandes décrites Figure 5.1. La première lance un test précis, ce qui permet de valider la correction d'un test qui ne fonctionnait

```
# Lancer Tempest pour realiser un test unique
cd /var/lib/tempest && ./run_tempest.sh -- tempest.api.compute.servers.
    test_servers_negative.ServersNegativeTestJSON.
    test_reboot_non_existent_server
# Lancer Tempest pour realiser la suite de tests complete
cd /var/lib/tempest && ./run_tempest.sh
# Lancer Tempest en arriere plan pour realiser la suite de tests complete
     et ecrire la sortie en fichier
screen -dm bash -c 'cd /var/lib/tempest && ./run_tempest.sh > tempest_`
    date "+%d-%m-%Y_%Hh%Mm%Ss"`.txt 2>&1'
```

FIGURE 5.1 – Tests - Lancement de la suite Tempest

pas auparavant, la deuxième lance simplement l'ensemble des tests disponibles tandis que la troisième lance les tests en arrière-plan et stocke les résultats dans un fichier. Cette troisième commande est importante car sur notre plateforme, la suite de tests complète peut prendre jusqu'à 8-10 heures.

5.3 Tests d'évolution

Dans le cadre de la réalisation des objectifs O05, O09, 012 (voir Figure 1.1), il est nécessaire de réaliser des tests concernant l'évolution horizontale des services. Cela passe par l'installation de nouveaux serveurs, leur déploiement et la vérification de l'insertion des nouveaux services dans le balancement de charge le cas échéant ou l'augmentation de l'espace de stockage disponible.

Ces tests sont réalisés manuellement avec l'utilisation de la douzième machine disponible qui est tantôt installé en noeud de contrôle et tantôt en noeud de stockage. Le succès est constaté par l'utilisation de l'interface de HAProxy (voir Section 5.6.1) qui montre l'ajout des nouveaux services de contrôle dans la répartition de charge (et de requêtes traitées par ces derniers) et par le contrôle de l'état de Ceph en ligne de commande qui indique l'apparition d'un nouveau service de contrôle de cluster *ceph-mon*, d'une nouvelle unité de stockage *ceph-osd* et par conséquent de nouveaux groupes de placement (PGs) et d'espace utile.

Les tests montrent que l'ajout de nouveaux noeuds de contrôle et de stockage est fonctionnel. Bien que des vérifications basiques ont été menées pour s'assurer de l'évolutivité horizontale, on note des analyses de performance précises (*benchmarking*) seraient nécessaires afin de valider et afin de découvrir les limites de la plateforme en fonction du nombre de serveurs déployés. Ces analyses n'ont pas été réalisées en l'état pour deux raisons principales : il a été préféré de se concentrer sur l'approfondissement de divers points du fonctionnement interne de OpenStack et sur la résolution de certains problèmes résiduels - et parce que nous n'estimons pas que la réalisation d'analyses de performance réalisées sur un si petit déploiement composé de machines anciennes, peu performantes, peu adaptées, et reliées à un unique switch 10/100 Mbps soit cohérent et fournisse des informations utiles. Pour réfé-

rence ultérieure, Rally [1] est l'outils de prédilection pour réaliser des analyses exactes, qu'il y ait 100 ou 1000 noeuds dans le déploiement.

5.4 Tests de disponibilité

Pour les objectifs O04, O08, 011 et O18 (voir Figure 1.1) qui concernent la haute disponibilité des contrôleurs, des volumes, des objets et des instances, des tests manuels sont réalisés dans un environnement dégradé ainsi que le lancement de la suite d'intégration Tempest. La dégradation de l'environnement correspond à l'arrêt de services ou bien de serveurs complets. Les arrêts pouvant être soit normaux (commandes d'extinction), inopinés (coupures électrique), simulés (coupures réseau) ou passagers (notion de *flapping* avec nombreuses coupures réseau ou commandes d'extinction/démarrage successives, courtes et rapprochées).

À chaque indisponibilité d'un serveur ou d'un service précis, il s'écoule un temps définit dans la configuration de HAProxy avant que celle-ci soit détectée (typiquement deux secondes). Pendant ce laps de temps, les clients réalisant des appels au service indisponible et ayant été dirigé vers celui-ci précisément par HAProxy rencontrent une erreur et doivent ré-itérer la requête. Ce phénomène est relativement présent dans un petit déploiement mais est négligeable et passerait presque inaperçu dans un déploiement de grande taille. De plus, la réduction du délai configuré peut être envisagé dans une certaine limite si les capacités des noeuds le permettent. Les interruptions passagères peuvent cependant passer inaperçues et demande une plus grande attention. Aussi, ne disposant que de trois à quatre noeuds de contrôle, il n'est tolérable que de perdre un seul noeud simultanément et il en est de même avec les noeuds de stockage - sinon il n'y a pas assez de vote disponible pour établir le quorum : c'est un problème dans le cadre de notre petit déploiement mais qui n'aurait pas lieu dans un déploiement de taille plus importante.

Finalement, la haute disponibilité des instances n'est pas opérationnel en l'état dans notre déploiement. En effet, bien que l'utilisation de Ceph pour le stockage permette d'"évacuer" (*nova evacuate*) les instances d'un noeud de calcul indisponible vers un nouveau noeud de calcul, il n'y a actuellement aucun système s'assurant de la disponibilité ou non d'un noeud de calcul. Cela pourrait se faire avec l'utilisation d'un système de monitoring tel que Zabbix [2] ou encore Nagios [3] couplé avec un système IPMI pour le *fencing*. Une autre approche a été étudiée avec l'outils d'orchestration Heat et plus particulièrement avec les composants *OS : :Heat : :HARestarter, AWS : :CloudWatch : :Alarm, AWS : :CloudFormation : :Init* et mais ce fonctionnement est désormais déprécié et n'est plus supporté au profit du futur système de convergence de Heat.

1. https ://wiki.openstack.org/wiki/Rally
2. http ://www.zabbix.com
3. http ://www.nagios.com

5.5 Suivi des tests

L'avancement des objectifs est synthétisé à l'aide des tests qui sont exécutés au moins une fois par semaine et reportés. Cela permet d'une part suivre le travail qui a été effectué ainsi que le chemin qu'il reste à parcourir pour compléter les objectifs du projet, et d'autre part de vérifier qu'il n'y a eu aucune regression à travers le temps. Le Tableau 5.1 recense les séries de test considérés pour la validation des objectifs tandis que la Figure 5.2 présente le suivi et l'évolution des résultats au cours du projet.

Alors que dans les premières semaines, le cahier de tests est principalement complété à l'aide des tests préliminaires et des tests de de disponibilité, à partir du 12 Mars (6ème semaine de déploiement), la plateforme est suffisamment mature pour être testée de façon précise et exhaustive avec la suite Tempest - la synthèse du nombre de tests passés et réussis sont alors reportés. L'augmentation du nombre de tests Tempest réalisés traduit l'implémentation de nouveaux services dans la plateforme et l'accroissement du taux de réussite est le fruit de l'effort de correction apporté au système. Le 9 Avril, le taux de succès est inférieur à celui du 2 Avril à cause de l'introduction de Heat et de ses tests dont nombre d'entre eux échouent.

Les 1.81% d'échecs restant sont liées à un problème résiduels décrit dans la section ci-dessous et à des tests non configurés : dont il est nécéssaire de fournir un contexte d'exécution spécifique (préparation d'images, ...) pour leurs réussites. Le temps n'a pas été alloué à ces derniers.

5.6 Outils pour l'identification & la résolution d'erreurs

L'outil principal pour l'identification et la résolution d'erreurs est bien entendu la consultation des logs des systèmes mais certains outils aident également dans la démarche et c'est ce dont cette section traite. Il a par ailleurs parfois été nécéssaire de corriger des bugs OpenStack ou Puppet - des *bug patch* ont ainsi été soumis.

5.6.1 Interfaces web

Deux interfaces en particulier sont intéressantes pour la résolution d'erreurs :

Interface RabbitMQ Voir Figures 5.3, 5.4, 5.5 (http://endpoint-1:1936/haproxy)
L'interface de RabbitMQ à la fois de vérifier l'état des instances du service et du cluster et à la fois de visualiser en temps réel l'ensemble des communications s'échangeant sur la plateforme. On peut ainsi voir les files de messages, les échanges et les messages en eux-mêmes mais également interagir en envoyant des messages, en en supprimant, etc. Cet outils fut particulièrement pratique dans la recherche d'erreurs car on trouve souvent dans les logs une erreur indiquant que le message *XXX* n'a pas reçu de réponse - il suffit alors de consulter ce panel pour identifier quelle opération devait être réalisée, et par qui, et ainsi de remonter au service correspondant pour en fouiller les logs (ou bien pour en augmenter la granularité).

Référence	Séries de test
O00 / O13 / O14	N/A
O01	*tempest.api.compute, tempest.scenario.test_aggregates_basic_ops, tempest.scenario.test_large_ops, tempest.scenario.test_minimum_basic, tempest.scenario.test_server_basic_ops, tempest.scenario.test_server_advanced_ops, tempest.scenario.test_shelve_instance*
O02	*tempest.api.compute.security_groups, tempest.scenario.test_minimum_basic, tempest.scenario.test_network_basic_server_ops, tempest.scenario.test_security_groups_basic_ops,*
O03	*tempest.api.compute, tempest.api.identity, tempest.api.image, tempest.api.network, tempest.api.orchestration, tempest.api.object_storage, tempest.api.telemetry, tempest.api.tempest.volume, scenario.test_minimum_basic*
O04	Tests de disponibilité (Section 5.4)
O05	Tests de d'évolution (Section 5.3)
O06	Tout les tests, réalisation de la démonstration, *tempest.api.messaging, tempest.scenario.test_dashboard_basic_ops*
O07	*tempest.api.compute.volume, tempest.cli.simple_read_only.volume, tempest.scenario.test_encrypted_cinder_volumes, tempest.scenario.test_minimum_basic, tempest.scenario.test_volume_boot_pattern*
O08	Ceph, Tests de disponibilité (Section 5.4)
O09	Ceph, Tests de d'évolution (Section 5.3)
O10	*tempest.api.object_storage*
O11	Ceph, Tests de disponibilité (Section 5.4)
O12	Ceph, Tests de d'évolution (Section 5.3)
O15	Tests préliminaires, *tempest.api.network.test_test_load_balancer, tempest.scenario.test_load_balancer_basic*
O16	*tempest.api.image, tempest.cli.simple_read_only.image, tempest.scenario.test_minimum_basic, tempest.scenario.test_snapshot_pattern, tempest.scenario.test_stamp_pattern*
O17	*tempest.api.compute.floating_ips, tempest.api.network, tempest.scenario.test_minimum_basic, tempest.scenario.test_network_basic_server_ops, tempest.scenario.test_network_advanced_server_ops*
O18	Tests de disponibilité (Section 5.4), *Ceph, tempest.api.telemetry*
O19	*tempest.api.telemetry*
O20	*tempest.api.orchestration, tempest.cli.simple_read_only.heat_templates, tempest.cli.simple_read_only.orchestration,*

TABLE 5.1 – Tests - Séries de tests réalisés pour valider les objectifs

Suivi des tests

		29 janvier	5 février	12 février	19 février	24 février	12 mars	19 mars	27 mars	2 avr.	9 avr.	16 avr.
O00 - Fonctionnement de la plateforme sur du matériel économique & d'usage courant					✓	✓	✓	✓	✓	✓	✓	✓
	Utilisation de Ceph avec un seul disque dur	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Utilisation de Open vSwitch avec une seule carte réseau				✓	✓	✓	✓	✓	✓	✓	✓
O01 - Mise à disposition d'instances virtuelles (sur stockage éphémère)						✓	✓	✓	✓	✓	✓	✓
	Création / Mise à jour / Suppression				✓	✓	✓	✓	✓	✓	✓	✓
	Accès console				✓	✓	✓	✓	✓	✓	✓	✓
	Accès SSH (clé publique)					✓	✓	✓	✓	✓	✓	✓
	Spécification de script de post-installation					✓	✓	✓	✓	✓	✓	✓
O02 - Capacités réseau basiques (IP privées, isolation VLAN)						✓	✓	✓	✓	✓	✓	✓
	Assignation d'adresses IP privées via DHCP					✓	✓	✓	✓	✓	✓	✓
	Fonctionnement des groupes de sécurité					✓	✓	✓	✓	✓	✓	✓
	Isolation des clients / machines virtuelles				✓	✓	✓	✓	✓	✓	✓	✓
O03 - Accès à une API standardisée pour contrôler ses services										✓	✓	✓
	Fonctionnement de Keystone		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Fonctionnement de Glance		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Fonctionnement de Nova		✓	✓	✓		✓	✓	✓	✓	✓	✓
	Fonctionnement de Cinder		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Fonctionnement de Neutron		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Fonctionnement de Heat										✓	✓
	Fonctionnement de Horizon		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Fonctionnement de Ceilometer									✓	✓	✓
O04 - Haute disponibilité des contrôleurs		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Fonctionnement de la détection d'indisponibilité & exclusion des répartiteurs de charges	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Cluster MySQL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Cluster RabbitMQ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
O05 - Evolutivité horizontale des contrôleurs		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Fonctionnement des balancements de charges	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Cluster MySQL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Cluster RabbitMQ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
O06 - Mise en place d'une plate-forme de démonstration fonctionnelle												✓
O07 - Stockage de volumes sur les instances virtuelles						✓	✓	✓	✓	✓	✓	✓
	Création / Mise à jour / Suppression d'instance permanente sur volume à partir d'image					✓	✓	✓	✓	✓	✓	✓
	Assignation de volumes aux instances					✓	✓	✓	✓	✓	✓	✓
	Création d'instance à partir de volumes					✓	✓	✓	✓	✓	✓	✓
O08 - Haute disponibilité des volumes			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Support de volumes (Ceph)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Fonctionnement de Cinder		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
O09 - Evolutivité horizontale de l'espace de stockage des volumes		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
O10 - Stockage d'objets		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
O11 - Haute disponibilité des objets		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
O12 - Evolutivité horizontale de l'espace de stockage des objets		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
O13 - Automatisation de déploiements							✓	✓	✓	✓	✓	✓
	Utilisation de Puppet	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Automatisation de la procédure d'installation système						✓	✓	✓	✓	✓	✓
O14 - Automatisation de tests							✓	✓	✓	✓	✓	✓
	Scripts de tests préliminaires	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Suite d'intégration Tempest						✓	✓	✓	✓	✓	✓
O15 - Balancements de charges entre instances virtuelles						✓		✓	✓	✓	✓	✓
O16 - Snapshots et Création d'images d'instances virtuelles personnalisées par les utilisateurs						✓	✓	✓	✓	✓	✓	✓
O17 - Capacités réseau avancées (IP flottantes, VPN, pare-feux, IDS)									✓	✓	✓	✓
	Assignation d'adresses IP flottantes					✓	✓	✓	✓	✓	✓	✓
	VPNaaS								✓	✓	✓	✓
	FWaaS							✓	✓	✓	✓	✓
	Création / Mise à jour / Suppression de réseaux				✓	✓	✓	✓	✓	✓	✓	✓
	Création / Mise à jour / Suppression de routeurs				✓	✓	✓	✓	✓	✓	✓	✓
O18 - Haute disponibilité des instances (re-création automatique)												
	Stockage partagé et haute disponibilité des données										✓	✓
	Reprise d'instance à froid										✓	✓
	Principe de convergence de Heat ou monitoring matériel de la plateforme											
O19 - Services de télémétries et de monitoring												
	Fonctionnement de Ceilometer									✓	✓	✓
	Monitoring de la plateforme											
O20 - Clusters d'instances virtuelles avec Auto-Scaling											✓	✓
Tempest	Nombre de tests réalisés avec succès	0	0	0	0	0	819	961	1012	1230	1318	1354
	Nombre de tests réalisés au total	0	0	0	0	0	1087	1155	1155	1267	1379	1379
	Pourcentage de succès	N/A	N/A	N/A	N/A	N/A	75,34 %	83,20 %	87,62 %	97,08 %	95,58 %	98,19 %

FIGURE 5.2 – Tests - Suivi des résultats

Interface HAProxy Voir Figure 5.6 (http://controller-1:15672/)
Idéal pour traquer les services API défaillants, cette interface permet en un seul coup d'oeil d'identifier les services fonctionnels et ceux qui sont arrêtés/plantés. L'interface permet également de vérifier le fonctionnement de la répartition de charge, de l'enregistrement des services auprès du répartiteur de charge, mais aussi et surtout de vérifier la bonne exclusion des services non disponibles dans le cadre des tests sur la haute-disponibilité.

5.6.2 Commandes utiles

Pour mémoire, la Figure 5.7 présente quelques commandes (bien sûr non exhaustive, sinon on ajouterait quelques dizaines de page, j'imagine) qui ont été utiles d'une façon ou d'une autre dans le processus de résolution d'erreurs.

On cite également [13, 26, 28] qui se sont révéler essentiels pour résoudre les problèmes liés aux réseaux Neutron et pour vérifier le fonctionnement de Ceph.

5.7 Problèmes résiduels

Dans l'infrastructure, deux problèmes subsistent :

RabbitMQ connections lack heartbeat or TCP keepalives
Ni l'implémentation actuelle de *Oslo Messaging* ni celle de *Kombu* ne supporte l'envoi de battements de coeurs (heartbeat) ou de keepalives [31]. Par conséquent, les répartiteurs de charges peuvent interrompre les connexions avec RabbitMQ et il n'est pas possible de détecter qu'une connexion a été interrompue (défaillance/indisponibilité). La réponse #13 [31] de *Vish Ishaya* résout partiellement le problème en permettant l'utilisation des TCP Keepalives mais cela n'est pas suffisant. Afin d'éviter l'interruption des connexions par les répartiteurs, qui entrainent des erreurs fatales lors de l'exécution de requêtes sur des services ayant été inactifs trop longtemps, les valeurs d'expirations de connexions au niveau des répartiteurs de charge ont été augmentés. La réponse #95 [31] montre que *Oslo Messaging* a été corrigé en date du 30 Mars 2015 mais cette correction s'applique à *Ubuntu Vivid* qui supporte *OpenStack Kilo* et n'a pas été backport pour *Ubuntu Trusty* et *OpenStack Juno* malgré ma demande. On note par ailleurs que *OpenStack Kilo* est devenu la nouvelle version stable de OpenStack le 30 Avril 2015.

Percona XtraDB Cluster Optimistic Locking
Le serveur de base de données utilise un contrôle optimiste de la concurrence, c'est à dire que les ressources ne sont pas verrouillées au niveau du cluster lors des transactions et cela peut générer des *deadlocks* lorsque plusieurs opérations d'écritures se déroulent simultanément sur plusieurs noeuds sur la même ressource [7, 29]. Les applications OpenStack n'étant pas adaptées (elles sont sans état et travaillent toutes indépendamment), cela pose bien évidemment problème alors que nous souhaiterions faire une répartition de charge sur les bases de données. Les transactions échouent

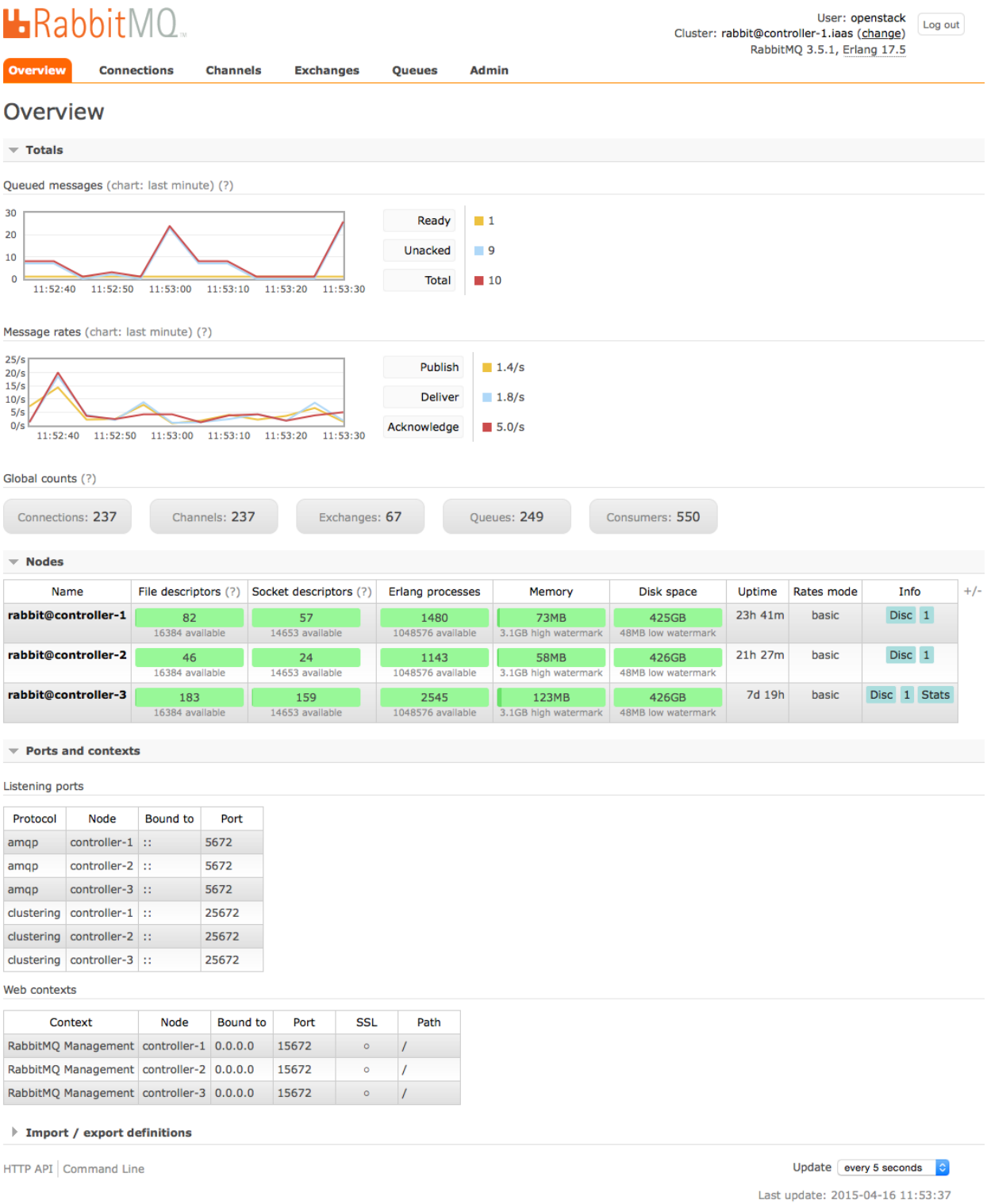

FIGURE 5.3 – Tests - Interface RabbitMQ

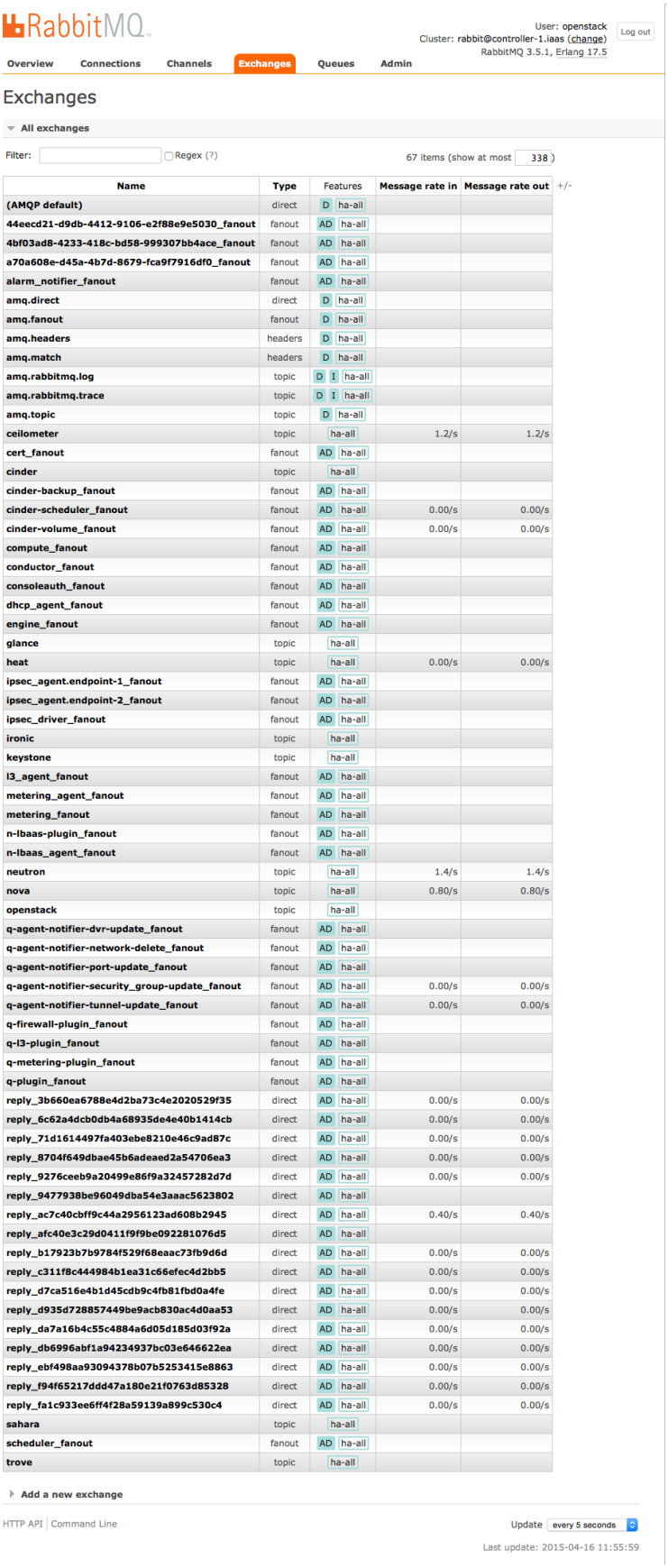

FIGURE 5.4 – Tests - Interface RabbitMQ (Exchanges)

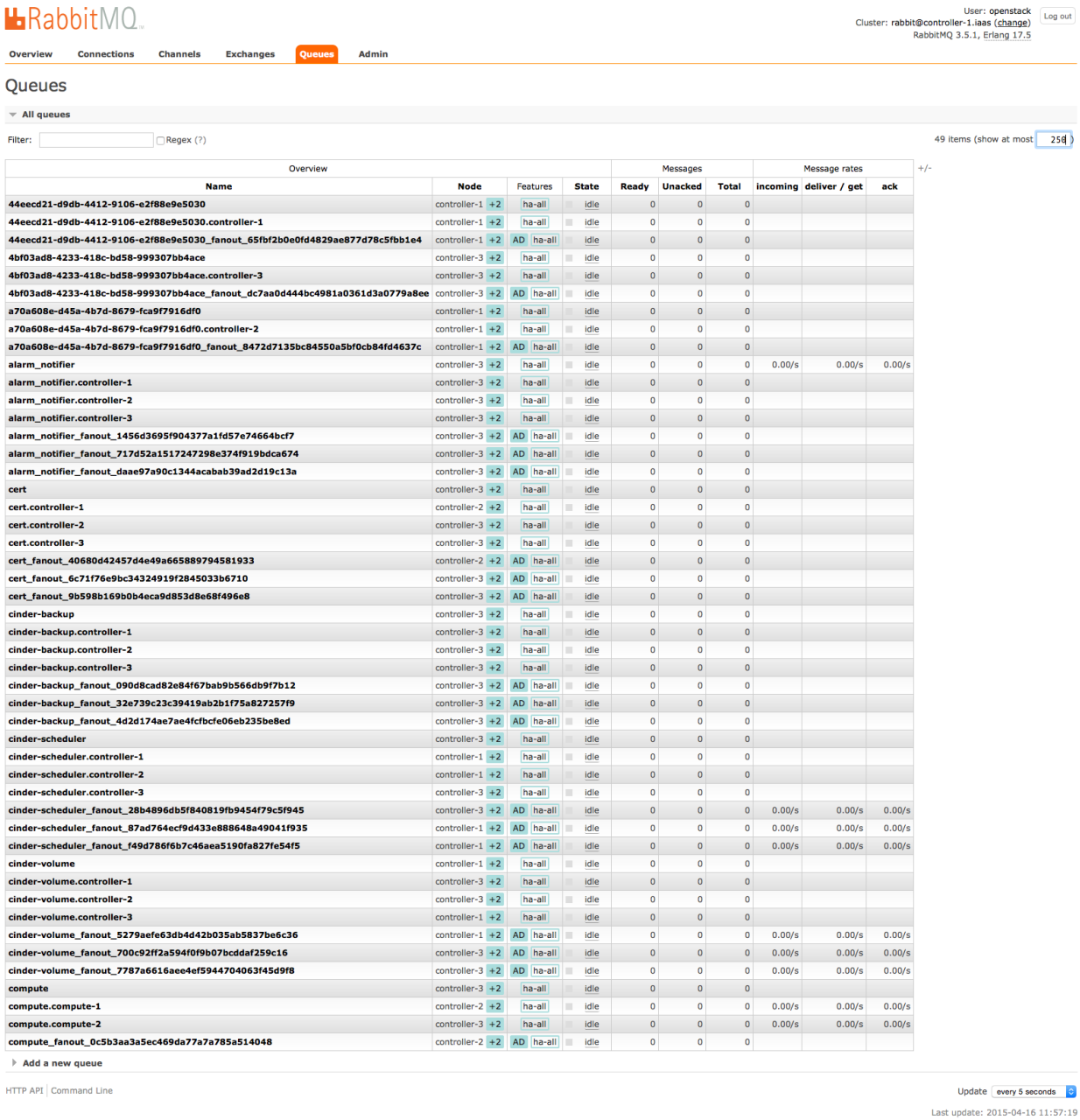

FIGURE 5.5 – Tests - Interface RabbitMQ (Queues)

HAProxy

Statistics Report for pid 14360 on endpoint-2.iaas

> General process information

pid = 14360 (process #1, nbproc = 1)
uptime = 0d 1h35m50s
system limits: memmax = unlimited; ulimit-n = 8076
maxsock = 8076; **maxconn** = 4000; **maxpipes** = 0
current conns = 560; current pipes = 0/0
Running tasks: 1/608

active UP
active UP, going down
active DOWN, going up
active or backup DOWN
active or backup DOWN for maintenance (MAINT)
backup UP
backup UP, going down
backup DOWN, going up
not checked

Note: UP with load-balancing disabled is reported as "NOLB".

Display option:
- Hide 'DOWN' servers
- Enable refresh
- Refresh now
- CSV export

External resources:
- Primary site
- Updates (v1.4)
- Online manual

ceilometer_api_cluster

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	1	-	0	1	8 092	289		75 929	1 333 920	0	0	0					OPEN								
controller-1	0	0	-	0	1		0	1	-	97	97	25 511	16 420		0		0	0	0	0	1h35m UP	L7OK/401 in 1ms	1	Y	-	0	0	0s	-
controller-2	0	0	-	0	1		0	1	-	96	96	25 209	658 750		0		0	0	0	0	1h35m UP	L7OK/401 in 1ms	1	Y	-	0	0	0s	-
controller-3	0	0	-	0	1		0	1	-	96	96	25 209	658 750		0		0	0	0	0	1h35m UP	L7OK/401 in 1ms	1	Y	-	0	0	0s	-
Backend	0	0		0	1		0	1	8 092	289	289	75 929	1 333 920	0	0		0	0	0	0	1h35m UP		3	3	0		0	0s	

cinder_api_cluster

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	3	-	1	5	8 092	86		28 146	100 524	0	0	0					OPEN								
controller-1	0	0	-	0	1		0	2	-	29	29	9 470	34 944		0		0	0	0	0	1h35m UP	L7OK/200 in 2ms	1	Y	-	0	0	0s	-
controller-2	0	0	-	0	1		1	2	-	29	29	9 025	30 489		0		0	0	0	0	1h35m UP	L7OK/200 in 3ms	1	Y	-	0	0	0s	-
controller-3	0	0	-	0	1		0	2	-	28	28	9 651	35 091		0		0	0	0	0	1h35m UP	L7OK/200 in 2ms	1	Y	-	0	0	0s	-
Backend	0	0		0	3		1	5	8 092	86	86	28 146	100 524	0	0		0	0	0	0	1h35m UP		3	3	0		0	0s	

glance_api_cluster

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	3	-	47	76	8 092	159		30 558	116 564	0	0	0					OPEN								
controller-1	0	0	-	0	1		15	24	-	53	53	10 431	39 441		0		0	0	0	0	1h35m UP	L7OK/300 in 1ms	1	Y	-	0	0	0s	-
controller-2	0	0	-	0	1		15	23	-	53	53	10 204	39 238		0		0	0	0	0	1h35m UP	L7OK/300 in 2ms	1	Y	-	0	0	0s	-
controller-3	0	0	-	0	1		17	32	-	53	53	9 923	37 885		0		0	0	0	0	1h35m UP	L7OK/300 in 2ms	1	Y	-	0	0	0s	-
Backend	0	0		0	3		47	75	8 092	159	159	30 558	116 564	0	0		0	0	0	0	1h35m UP		3	3	0		0	0s	

glance_registry_cluster

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	0	-	0	0	8 092	0		0	0	0	0	0					OPEN								
controller-1	0	0	-	0	0		0	0	-	0	0	0	0		0		0	0	0	0	1h35m UP	L7OK/401 in 1ms	1	Y	-	0	0	0s	-
controller-2	0	0	-	0	0		0	0	-	0	0	0	0		0		0	0	0	0	1h35m UP	L7OK/401 in 1ms	1	Y	-	0	0	0s	-
controller-3	0	0	-	0	0		0	0	-	0	0	0	0		0		0	0	0	0	1h35m UP	L7OK/401 in 1ms	1	Y	-	0	0	0s	-
Backend	0	0		0	0		0	0	8 092	0	0	0	0	0	0		0	0	0	0	1h35m UP		3	3	0		0	0s	

heat_api_cluster

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	6	-	0	2	8 092	12		3 522	13 452	0	0	0					OPEN								
controller-1	0	0	-	0	2		0	1	-	4	4	1 178	3 320		0		0	0	0	0	1h35m UP	L7OK/300 in 1ms	1	Y	-	0	0	0s	-
controller-2	0	0	-	0	2		0	1	-	4	4	1 146	3 360		0		0	0	0	0	1h35m UP	L7OK/300 in 1ms	1	Y	-	0	0	0s	-
controller-3	0	0	-	0	2		0	1	-	4	4	1 198	6 772		0		0	0	0	0	1h35m UP	L7OK/300 in 1ms	1	Y	-	0	0	0s	-
Backend	0	0		0	6		0	2	8 092	12	12	3 522	13 452	0	0		0	0	0	0	1h35m UP		3	3	0		0	0s	

horizon_cluster

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	17	-	0	13	8 092	954		2 421 853	22 003 208	0	0	0					OPEN								
controller-1	0	0	-	0	6		0	5	-	318	318	803 984	9 495 498		0		0	0	0	0	1h35m UP	L7OK/200 in 1ms	1	Y	-	0	0	0s	-
controller-2	0	0	-	0	6		0	5	-	318	318	809 666	6 509 543		0		0	0	0	0	1h35m UP	L7OK/200 in 1ms	1	Y	-	0	0	0s	-
controller-3	0	0	-	0	5		0	4	-	318	318	808 203	5 998 167		0		0	0	0	0	1h35m UP	L7OK/200 in 1ms	1	Y	-	0	0	0s	-
Backend	0	0		0	17		0	13	8 092	954	954	2 421 853	22 003 208	0	0		0	0	0	0	1h35m UP		3	3	0		0	0s	

keystone_admin_cluster

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	12	-	0	2	8 092	1 218		358 822	4 712 992	0	0	0					OPEN								
controller-1	0	0	-	0	4		0	1	-	406	406	119 737	1 568 258		0		0	0	0	0	1h35m UP	L7OK/300 in 2ms	1	Y	-	0	0	0s	-
controller-2	0	0	-	0	4		0	1	-	406	406	119 565	1 587 104		0		0	0	0	0	1h35m UP	L7OK/300 in 2ms	1	Y	-	0	0	0s	-
controller-3	0	0	-	0	4		0	1	-	406	406	119 520	1 557 630		0		0	0	0	0	1h35m UP	L7OK/300 in 2ms	1	Y	-	0	0	0s	-
Backend	0	0		0	12		0	2	8 092	1 218	1 218	358 822	4 712 992	0	0		0	0	0	0	1h35m UP		3	3	0		0	0s	

keystone_public_internal_cluster

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	3	-	0	2	8 092	141		47 292	463 122	0	0	0					OPEN								
controller-1	0	0	-	0	1		0	1	-	47	47	15 351	153 054		0		0	0	0	0	1h35m UP	L7OK/300 in 2ms	1	Y	-	0	0	0s	-
controller-2	0	0	-	0	1		0	1	-	47	47	15 225	149 347		0		0	0	0	0	1h35m UP	L7OK/300 in 2ms	1	Y	-	0	0	0s	-
controller-3	0	0	-	0	1		0	1	-	47	47	16 716	160 721		0		0	0	0	0	1h35m UP	L7OK/300 in 2ms	1	Y	-	0	0	0s	-
Backend	0	0		0	3		0	2	8 092	141	141	47 292	463 122	0	0		0	0	0	0	1h35m UP		3	3	0		0	0s	

neutron_api_cluster

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	19	-	0	3	8 092	1 665		389 344	16 006 700	0	0	0					OPEN								
controller-1	0	0	-	0	6		0	2	-	555	555	129 927	5 144 693		0		0	0	0	0	1h35m UP	L7OK/200 in 1ms	1	Y	-	0	0	0s	-
controller-2	0	0	-	0	6		0	3	-	555	555	129 294	6 013 347		0		0	0	0	0	1h35m UP	L7OK/200 in 1ms	1	Y	-	0	0	0s	-
controller-3	0	0	-	0	7		0	2	-	555	555	130 123	4 848 660		0		0	0	0	0	1h35m UP	L7OK/200 in 1ms	1	Y	-	0	0	0s	-
Backend	0	0		0	19		0	3	8 092	1 665	1 665	389 344	16 006 700	0	0		0	0	0	0	1h35m UP		3	3	0		0	0s	

nova_api_cluster

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	10	-	0	10	8 092	720		222 173	3 850 304	0	0	0					OPEN								
controller-1	0	0	-	0	3		0	3	-	240	240	73 348	1 457 224		0		0	0	0	0	1h35m UP	L7OK/200 in 1ms	1	Y	-	0	0	0s	-
controller-2	0	0	-	0	4		0	5	-	240	240	74 954	1 246 242		0		0	0	0	0	1h35m UP	L7OK/200 in 1ms	1	Y	-	0	0	0s	-
controller-3	0	0	-	0	3		0	4	-	240	240	73 871	1 146 838		0		0	0	0	0	1h35m UP	L7OK/200 in 2ms	1	Y	-	0	0	0s	-
Backend	0	0		0	10		0	10	8 092	720	720	222 173	3 850 304	0	0		0	0	0	0	1h35m UP		3	3	0		0	0s	

nova_metadata_api_cluster

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	1	-	0	2	8 092	3		1 149	687	0	0	0					OPEN								
controller-1	0	0	-	0	1		0	1	-	1	1	383	229		0		0	0	0	0	1h35m UP	L7OK/200 in 1ms	1	Y	-	0	0	0s	-
controller-2	0	0	-	0	1		0	1	-	1	1	383	229		0		0	0	0	0	1h35m UP	L7OK/200 in 1ms	1	Y	-	0	0	0s	-
controller-3	0	0	-	0	1		0	1	-	1	1	383	229		0		0	0	0	0	1h35m UP	L7OK/200 in 1ms	1	Y	-	0	0	0s	-
Backend	0	0		0	1		0	2	8 092	3	3	1 149	687	0	0		0	0	0	0	1h35m UP		3	3	0		0	0s	

nova_novncproxy

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	0	-	0	0	8 092	0		0	0	0	0	0					OPEN								
controller-1	0	0	-	0	0		0	0	-	0	0	0	0		0		0	0	0	0	1h35m UP	L4OK in 0ms	1	Y	-	0	0	0s	-
controller-2	0	0	-	0	0		0	0	-	0	0	0	0		0		0	0	0	0	1h35m UP	L4OK in 0ms	1	Y	-	0	0	0s	-
controller-3	0	0	-	0	0		0	0	-	0	0	0	0		0		0	0	0	0	1h35m UP	L4OK in 0ms	1	Y	-	0	0	0s	-
Backend	0	0		0	0		0	0	8 092	0	0	0	0	0	0		0	0	0	0	1h35m UP		3	3	0		0	0s	

rabbitmq

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	114	-	261	261	8 092	292		2 603 007	2 847 079	0	0	0					OPEN								
controller-1	0	0	-	0	38		87	87	-	98	98	1 121 253	614 832		0		0	11	0	0	1h35m UP	L4OK in 0ms	1	Y	-	0	0	0s	-
controller-2	0	0	-	0	38		87	87	-	97	97	169 094	2 140 416		0		0	10	0	0	1h35m UP	L4OK in 0ms	1	Y	-	0	0	0s	-
controller-3	0	0	-	0	38		87	87	-	97	97	1 312 660	91 831		0		0	10	0	0	1h35m UP	L4OK in 0ms	1	Y	-	0	0	0s	-
Backend	0	0		0	114		261	261	8 092	292	292	2 603 007	2 847 079	0	0		0	0	31	0	0	1h35m UP		3	3	0		0	0s	

rabbitmq

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	114	-	261	261	8 092	292		2 603 007	2 847 079	0	0	0					OPEN								
controller-1	0	0	-	0	38		87	87	-	98	98	1 121 253	614 832		0		0	11	0	0	1h35m UP	L4OK in 0ms	1	Y	-	0	0	0s	-
controller-2	0	0	-	0	38		87	87	-	97	97	169 094	2 140 416		0		0	10	0	0	1h35m UP	L4OK in 0ms	1	Y	-	0	0	0s	-
controller-3	0	0	-	0	38		87	87	-	97	97	1 312 660	91 831		0		0	10	0	0	1h35m UP	L4OK in 0ms	1	Y	-	0	0	0s	-
Backend	0	0		0	114		261	261	8 092	292	292	2 603 007	2 847 079	0	0		0	0	31	0	0	1h35m UP		3	3	0		0	0s	

FIGURE 5.6 – Tests - Interface HAProxy

```
# Supprimer entierement un Ceph OSD
ID=0
stop ceph-osd id=$ID && ceph osd out $ID && ceph osd crush remove osd.$ID
    && ceph auth del osd.$ID && ceph osd rm $ID && umount /var/lib/ceph/
    osd/ceph-$ID

# Verifier l'etat de Ceph
ceph -s && ceph health && ceph quorum_status --format json-pretty

# Forcer la recreation des Ceph PG
ceph pg dump_stuck stale | awk '{print $1}' | xargs -n 1 ceph pg
    force_create_pg

# Bootstraper PerconaDB
/etc/init.d/mysql bootstrap-pxc

# Forcer la suppression des machines virtuelles en echec
nova list | grep ERROR | awk '{print $2}' | xargs -n 1 -I % sh -c 'nova
    reset-state %; nova force-delete %'

# Supprimer tous les reseaux rapidement
neutron net-list | tail -n +4 | grep -v "ext-net" | grep -v "admin-net" |
    awk -F "|" '{print $3}' | sed 's/^ *//;s/ *$//' > netlist
cat netlist | while read line ; do neutron net-delete "$line" ; done
rm netlist

# Nettoyer les tokens Keystone en base de donnees sans ralentir le
    systeme
apt-get install percona-toolkit
pt-archiver --source h=endpoint-2,u=keystone,p=keystone,D=keystone,t=
    token --charset utf8 --where "expires < UTC_TIMESTAMP()" --purge --txn
    -size 500 --run-time 59m --statistics --primary-key-only | logger -t
    cleanup-keystone-tokens

# Vider entierement une base de donnees (not safe)
mysql -Nse 'show tables' $DB | while read table; do mysql -e "SET
    FOREIGN_KEY_CHECKS=0; truncate table $table; SET FOREIGN_KEY_CHECKS
    =1;" ceilometer; done
```

FIGURE 5.7 – Tests - Quelques commandes utiles

donc dans certains cas (deux transactions en conflit mène à l'échec d'une des deux transaction), et alors qu'une nouvelle tentative pourrait résoudre le problème, OpenStack ne le fait pas et cet échec se traduit par l'échec de la requête de l'utilisateur. On configure les répartiteurs de charge pour utiliser un seul serveur MySQL simultanément et laisser les autres en secours : on fait alors de l'actif/passif plutôt que de l'actif/actif et on perd les avantages d'évolution de ce côté [8]. Pour solutionner ce problème, on pourrait forcer les écriture à se produire sur un seul noeud et permettre les lectures sur chaque noeud mais cela semble compliqué à mettre en oeuvre.

Gestion de projet

6.1 Gestion des sources

L'intégralité du projet fut placé sous gestionnaire de versions. Les différents rapports et les documents liés à la gestion du projet étaient gérés par l'outil *git* et entreposés sur le dépôt distant Bitbucket [1] (choisi pour son caractère privé) où une mise à jour hebdomadaire fut réalisée. Quant aux sources Puppet permettant le déploiement de l'infrastructure, elles furent gérés par le même outil *git* mais cette fois-ci disponibles publiquement sur un dépôt Github (https://github.com/Quentin-M/puppet-iaas) et une mise à jour par amélioration a été réalisée.

Par ailleurs, certaines réalisations furent soumises en amont, sur les des systèmes de revues de code *Gerrit* de OpenStack mais aussi de StackForge pour être intégrés aux projets libres.

6.2 Planification prévisionnel / réelle

En début de projet, une planification prévisionnelle avait été réalisée et reportée dans le cahier de spécifications : plusieurs tâches étaient décrites & planifiées et différents jalons avaient été définis. Dans cette section, on compare le prévisionnel avec le déroulement réel du projet. Pour cela, on utilise le suivi réalisé durant le projet mais également l'historique des gestionnaires de versions.

L'étude et la conception s'est déroulée normalement avec pour terminer des jalons de livraison du CDS/PDD (09/01) qui ont été respectés avec une validation au 19/12 par l'encadrement, soit avec 21 jours d'avance par rapport à l'échéance. La présentation de mi-projet s'est également déroulé comme prévu. Puis, le déploiement a débuté plus tôt que prévu comme en témoigne le *commit 70e2ea9* du 29/01 qui comprend déjà une quantité importante de travail. De la même façon, les tests ont étés réalisés en parallèle, comme prévu. Le rapport hebdomadaire du 16 Mars montre que la tâche d'automatisation d'évolution et des tests de flexibilité a débuté avec une semaine d'avance. Enfin, la phase de développement s'arrête le 16/04 avec le *commit 7c25d14* et les derniers tests - comptabilisant ainsi deux semaines d'avance. À partir de ce point, la préparation d'une démonstration débuta tout en essayant de valider l'objectif O18 et s'étendra jusqu'au début des vacances de Printemps, au 25/04. La décision de raccourcir la phase de mise en oeuvre deux semaines pour passer à la réalisation de la démonstration a permis d'une part de réaliser une démonstration ambitieuse - et de prendre le temps de la faire mais aussi de pouvoir utiliser et profiter de la plateforme durant près de deux semaines. Cela a permis notamment de valider la réponse offerte

1. bitbucket.org

par celle-ci pour les besoins de Harmony-Hosting. Cette décision a également pu mettre en exergue que l'idée d'utiliser Heat pour la réalisation de l'objectif O18 n'était pas possible dans cette version de OpenStack et qu'il fallait soit attendre que Heat aboutisse son principe de convergence - soit mettre en oeuvre un système de monitoring. Ces solutions étant simples à réaliser, la démonstration ayant une priorité supérieure dans les objectifs et le fait d'avoir pu expérimenter la plateforme font que cette décision est justifié. Finalement, à la date où ces lignes sont rédigées, le 5 Mai, la rédaction du rapport touche à sa fin, toujours comme prévu et la préparation de la présentation va débuter.

Pour synthétiser vis-à-vis de la planification, le projet a dans l'ensemble respecter son prévisionnel, avec néanmoins environ 2 semaines d'avance au global, qui ont étés mise à profit de la démonstration mais aussi de l'utilisation de la plateforme sous diverses coutures pour confirmer qu'elle répondait aux besoins. Cette avance et ce respect des délais ne sont pas anodins et sont la traduction d'un effort important durant le projet pour les respecter. Chaque disponibilité fut utilisée pour le bien du projet, y compris le Samedi matin afin de maximiser les choses d'assurer la réussite du projet - d'autant plus qu'il n'était pas possible de travailler sur le projet activement en dehors des horaires d'ouverture de l'établissement à cause des contraintes matérielles et réseaux.

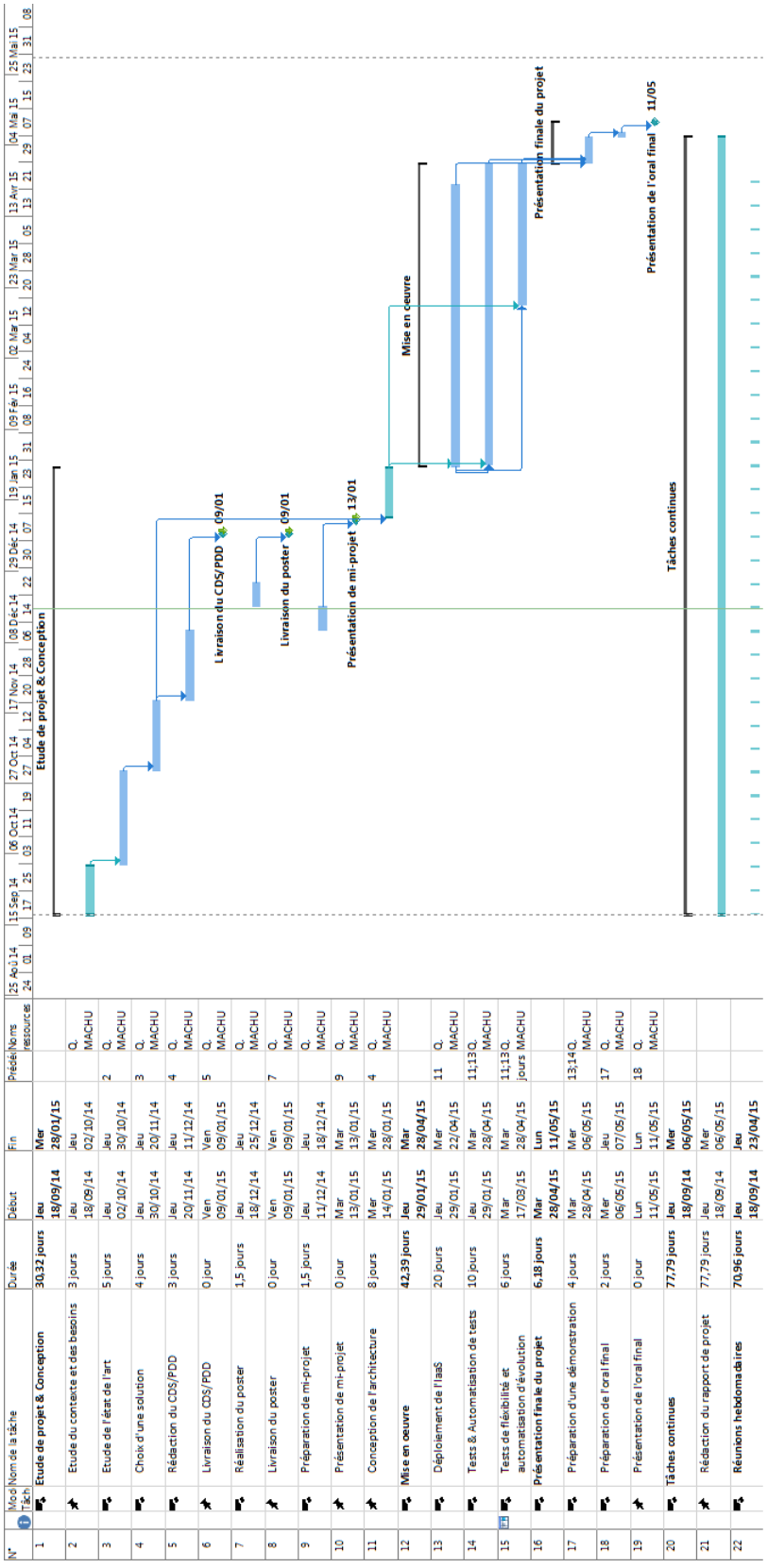

FIGURE 6.1 – Diagramme de Gantt prévisionnel pour le projet

Conclusion

Ce projet de relative longue durée a été imaginé à partir du contexte actuel de Harmony-Hosting et dans une perspective d'avenir pour l'entreprise. Le projet a ainsi débuté par une analyse des besoins, par la définition des objectifs en tenant compte des moyens disponibles, puis une enrichissante et exhaustive étude de l'état a été réalisée, menant à un choix technologique pour la réalisation du projet. Suite au découpage en tâches du projet et à leurs planification, l'architecture a adoptée a été définit et une longue et méthodique période de développement mais aussi de mise en oeuvre a été réalisée avec un suivi pro-actif de son avancement à travers les tests, effectués simultanément - qui deviennent alors des indicateurs de suivi lorsqu'ils mis en parallèle des objectifs. Enfin, le projet s'est finalisé, dès lors que le résultat répondit aux besoins, avec l'expérimentation du fruit du travail et avec une démonstration.

Vis à vis des 20 objectifs définis, 18 (90%) sont réalisés parfaitement - avec un taux de réussite aux tests pratiqués (près de 1500) de 98.19%, 1 objectif partiellement réalisé (O18) et qui est en attente d'une fonctionnalité logicielle planifiée par les équipes de développement en charge de Heat (une solution existait auparavant, elle a disparu au profit d'une future fonctionnalité) ou bien nécessite l'acquisition de matériel spécifique (IPMI), et 1 objectif lui aussi partiellement réalisé (O19) pour lequel il manque l'installation d'une solution de surveillance, dont la mise en oeuvre est simple mais qui a volontairement été omis au profit d'une phase d'expérimentation plus longue (accès physique à l'établissement nécessaire et impossible durant les vacances scolaires). Parmi les 1.81% d'échecs aux tests, une importante partie est simplement dû au fait que certains groupes de tests n'ont pas été configurés tandis qu'une seconde partie est liée à un problème résiduel - discuté dans la section correspondante - où une dépendance logicielle dispose d'une anomalie, corrigée en amont depuis très peu de temps. Par ailleurs, on note que dans le cadre de ce projet, diverses anomalies dans les logiciels libres utilisés ont pu être décelés, corrigées par moi-même et dont les correctifs ont été soumis auprès de la communauté de développement pour y être intégrés. En accord avec la validation des objectifs et la réussite des tests, à mes yeux (et par voie de conséquence, aux yeux de Harmony-Hosting également), la réalisation de ce projet est un succès qui saura servir au développement de l'entreprise. Certains éléments étudiés au cours de ce projet sont d'ailleurs d'ores et déjà intégrés à la production de Harmony-Hosting, je pense tout particulièrement à l'outil d'automatisation Puppet, au stockage Ceph et d'autres éléments qui sont en cours d'intégration devraient paraître durant l'été 2015 : utilisation de HAProxy pour la répartition de charge et Galera pour la gestion de bases de données distribuées.

Ce projet, destiné à préparer l'avenir de l'entreprise Harmony-Hosting, s'est révélé être tout à fait audacieux et rempli de défis. Seuls de grandes entreprises utilisent aujourd'hui, à titre privé, des plateformes IaaS hautement disponible et peu d'entreprises osent se lancer dans l'aventure. L'IaaS, et le cloud de manière plus générale est aujourd'hui un secteur qui

tend à se développer de façon importante et fait appel à une vaste de technologies toutes plus à la pointes les unes que les autres. Ce projet, réalisé dans le cadre de mes études d'ingénieur, a ainsi été une magnifique opportunité pour moi et m'a permis de réaliser une très bonne montée en compétences. Ayant la ferme conviction d'orienter ma carrière dans le domaine des systèmes distribués et disponibles, je suis persuadé que cet apprentissage et cette expertise m'aideront à propulser ma carrière dans cette direction. Enfin, je suis fier d'avoir mené ce projet de façon relativement autonomie, et sous l'encadrement de Mr Aupetit que je remercie vivement pour son soutient, son aide et sa disponibilité, qui sont sans faille.

Finalement, la question se pose alors à propos des perspectives du projet et des axes d'amélioration. Tout d'abord et comme cela a déjà été évoqué, ce monde évolue très rapidement : entre le début et la fin de ce projet, une nouvelle version majeure de OpenStack a été publiée. Il est alors essentiel de parler de la maintenance, du suivi de versions et de la logistique liée aux transitions de versions - dans des conditions respectant la disponibilité et l'efficacité de la plateforme - ceci constitue un véritable travail. D'autre part, dans le cadre de ce déploiement, les machines virtuelles fonctionnaient alors sur un outil de virtualisation lourd (KVM) mais le futur nous dirige vers l'utilisation de nano-ordinateurs ARM en environnement haute densité mais aussi vers les conteneurs (Docker, rkt, etc) qui tendent à devenir la norme dans les déploiement de masse et dont les spécifications standards [1] sont en train d'être définies, poussées par les géants que sont Google, Apcera, Red Hat ou encore VMWare [2]. C'est d'ailleurs précisément le sujet de mon stage chez CoreOS, une start-up basée à San Francisco éditant un système d'exploitation, divers outils et spéficiations devenus en très peu de temps de véritables piliers pour les géants du domaine.

1. https ://github.com/appc/spec
2. https ://coreos.com/blog/appc-gains-new-support/

Glossaire

API (Application Programming Interface) Une API est un ensemble normalisé de classes, de méthodes ou de fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels. Elle est offerte par une bibliothèque logicielle ou un service web, le plus souvent accompagnée d'une description qui spécifie comment des programmes consommateurs peuvent se servir des fonctionnalités du programme fournisseur.

Balancement de charge La répartition de charge est un ensemble de techniques permettant de distribuer une charge de travail entre différents ordinateurs d'un groupe. Ces techniques permettent à la fois de répondre à une charge trop importante d'un service en la répartissant sur plusieurs serveurs, et de réduire l'indisponibilité potentielle de ce service que pourrait provoquer la panne logicielle ou matérielle d'un unique serveur.

Cloud-computing Le cloud-computing est un modèle économique 'par per use' pour accéder et utiliser, à travers du réseau, à un ensemble de ressources informatiques configurables et partagées (i.e. réseaux, serveurs, stockage, applications, etc.) qui peuvent être rapidement achetées et utilisées avec un effort de gestion minimal et une interaction limitée avec le fournisseur de services.

Evolutivité horizontale L'évolutivité horizontale consiste à ajouter des ordinateurs pour faire face à une demande accrue d'un service. La méthode la plus courante est la répartition de charge.

IaaS (Infrastructure as a Service) Ce service consiste à offrir un accès (via Internet) à un porc informatique virtualité contenant des machines virtuelles, espaces de stockages, réseaux, etc. Ce parc est évolutif et capable de s'adapter (automatiquement ou manuellement) aux besoins du consommateur.

IaaS Publique L'infrastructure est mise à disposition du grand publique et utilisable par différentes organisations, entreprises, académies, gouvernement ou personnes. Ce terme est à opposer avec Private et Hybrid.

IDS (Intrusion Detection System) Un système de détection d'intrusion est un mécanisme destiné à repérer des activités anormales ou suspectes sur la cible analysée (un réseau ou un hôte). Il permet ainsi d'avoir une connaissance sur les tentatives réussies comme échouées des intrusions.

Instance virtuelle Une instance/machine virtuelle est une illusion d'un appareil informatique créée par un logiciel d'émulation. Le logiciel d'émulation simule la présence de ressources matérielles et logicielles telles que la mémoire, le processeur, le disque dur, voire le système d'exploitation et les pilotes, permettant d'exécuter des programmes dans les mêmes conditions que celles de la machine simulée.

IP Privée Les adresses IP privées représentent toutes les adresses IP que l'on peut utiliser dans un réseau local (LAN) et qui ne peuvent pas être utilisées sur internet (car elles ne peuvent pas être routées sur internet).

IP Flottante Les adresses IP flottantes sont des adresses IP fixes et publiques que l'on peut assigner/supprimer à une instance virtuelle à souhait.

Monitoring Le monitoring ou monitorage est une activité de surveillance et de mesure d'une activité informatique. On parle aussi de supervision. Les raisons peuvent variées : mesure de performance, disponibilité, intégrité, etc.

Point unique de défaillance (SPoF) Un point unique de défaillance est un point d'un système informatique dont le reste du système est dépendant et dont une panne entraîne l'arrêt complet du système.

Snapshot Un instantané est l'état d'un système à un instant donné.

Stockage objet Le stockage objet est une façon de stocker des données (souvent à l'aide d'une API) en considérant celles-ci comme des entités indépendantes et manipulables indépendamment. Par exemple, chaque fichier peut être un objet, il n'y a par ailleurs aucune arborescence classant ces objets.

Télémétrie La télémétrie est une activité permettant de remonter des informations

VLAN (Virtual Local Area Network) Un VLAN est un réseau local regroupant un ensemble de machines de façon logique et non physique. Un VLAN permet notamment l'isolation de machines par rapport aux autres.

Stateless Désigne un service ou un serveur traitant chaque requête comme une transaction indépendante et sans relation avec une quelconque précédente requête.

Quorum Ensemble de votants fournissant un moyen d'arbitrer un ensemble de noeuds regroupés au sein d'un groupe et d'aider à maintenir la cohérence de ce groupe. Il faut trois votants à minima afin de pouvoir prendre une décision.

Adresse IP virtuelle (VIP) Une adresse IP virtuelle est une adresse IP non connectée à un ordinateur ou une carte réseau spécifiques. Les paquets entrants sont envoyés à l'adresse IP virtuelle, mais en réalité ils circulent tous via des interfaces réseau réelles.

DNS Le Domain Name System est un service permettant de traduire un nom de domaine (comme *debian.org*) informations de plusieurs types qui y sont associées, notamment en adresses IP de la machine portant ce nom.

DHCP Le Dynamic Host Configuration Protocol est un protocole réseau dont le rôle est d'assurer la configuration automatique des paramètres IP d'une station, notamment en lui affectant automatiquement une adresse IP et un masque de sous-réseau. DHCP peut aussi configurer l'adresse de la passerelle par défaut et des serveurs de noms DNS.

Passerelle ou routeur Un routeur est un élément intermédiaire dans un réseau informatique assurant le routage des paquets. Son rôle est de faire transiter des paquets d'une interface réseau vers une autre, éventuellement d'un réseau à un autre.

Pre-boot eXecution Environment (PXE) L'amorçage PXE permet à une station de démarrer depuis le réseau en récupérant une image de système d'exploitation qui se trouve sur un serveur. L'image peut ainsi être utilisée pour installer la station.

TFTP Le Trivial File Transfer Protocol est un protocole simplifié (par rapport au FTP) de transfert de fichiers.

Bibliographie

[1] S. Afchain. Add high availability features on l3 agent. https://blueprints.launchpad.net/neutron/+spec/l3-high-availability. Consulté en ligne le 20 Janvier 2015.

[2] Shreeram Akilesh. Managing openstack internal/data/external network in one interface. https://fosskb.wordpress.com/2014/06/10/managing-openstack-internaldataexternal-network-in-one-interface/. Consulté en ligne le 5 Février 2015.

[3] Apache. Cloudstack documentation. http://docs.cloudstack.apache.org/en/latest/. Consulté en ligne le 23 Octobre 2014.

[4] Ceilometer System Architecture. Building a functional puppet workflow part 2 : Roles and profiles. http://docs.openstack.org/developer/ceilometer/architecture.html. Consulté en ligne le 27 Avril 2015.

[5] Eric W. Biederman. ip-netns - process network namespace management. http://man7.org/linux/man-pages/man8/ip-netns.8.html. Consulté en ligne le 10 Février 2015.

[6] Arnaud Bonneville. Prism et l'affaire nsa : un frein à l'adoption du cloud ? http://www.baccoubonneville.com/blog-cloud-computing/prism-et-affaire-nsa-un-frein-a-l-adoption-du-cloud.html. Consulté en ligne le 16 Octobre 2014.

[7] Peter Boros. Openstack users shed light on percona xtradb cluster deadlock issues. http://www.percona.com/blog/2014/09/11/openstack-users-shed-light-on-percona-xtradb-cluster-deadlock-issues/. Consulté en ligne le 2 Mars 2015.

[8] Peter Boros. Percona xtradb cluster reference architecture with haproxy. https://www.percona.com/blog/2012/06/20/percona-xtradb-cluster-reference-architecture-with-haproxy/. Consulté en ligne le 2 Mars 2015.

[9] T. Brisco. Dns support for load balancing. Rfc1794, Network Working Group, http ://-tools.ietf.org/html/rfc1794, April 1995.

[10] OpenNebula Community. Opennebula networking documentation. http://docs.opennebula.org/4.4/administration/networking/nm.html. Consulté en ligne le 23 Octobre 2014.

[11] OpenStack Community. Openstack cloud administration guide : Under the hood open vswitch. http://docs.openstack.org/admin-guide-cloud/content/under_the_hood_openvswitch.html. Consulté en ligne le 21 Janvier 2015.

[12] OpenStack Community. Openstack cloud administrator guide : Configure ovs plugin. http://docs.openstack.org/admin-guide-cloud/content/openvswitch_plugin.html. Consulté en ligne le 15 Février 2015.

[13] OpenStack Community. Openstack network troubleshooting. http://docs.openstack.org/openstack-ops/content/network_troubleshooting.html. Consulté en ligne le 24 Février 2015.

[14] OpenStack Community. Openstack networking documentation. http://docs.openstack.org/admin-guide-cloud/content/ch_networking.html. Consulté en ligne le 23 Octobre 2014.

[15] OpenStack Community. Openstack networking documentation - ml2. https://wiki.openstack.org/wiki/Neutron/ML2. Consulté en ligne le 23 Octobre 2014.

[16] Red Hat community. Networking in too much detail. https://www.rdoproject.org/Networking_in_too_much_detail. Consulté en ligne le 21 Janvier 2015.

[17] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner : Google's globally-distributed database. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 261–264, Hollywood, CA, October 2012. USENIX Association.

[18] Loïc Dachary. Create a partition and make it an osd. http://dachary.org/?p=3037. Consulté en ligne le 20 Janvier 2015.

[19] OpenStack Documentation. Limitations de percona xtradb. http://docs.openstack.org/developer/nova/devref/rpc.html. Consulté en ligne le 22 Avril 2015.

[20] Puppet documentation. Exported resource design patterns. https://docs.puppetlabs.com/guides/exported_resources.html. Consulté en ligne le 20 Janvier 2015.

[21] Ubuntu Documentation. Netboot : installation par tftp, pxe, dhcp. http://doc.ubuntu-fr.org/netboot. Consulté en ligne le 20 Janvier 2015.

[22] Ubuntu Documentation. Serveur dhcp : isc-dhcp-server. http://doc.ubuntu-fr.org/isc-dhcp-server. Consulté en ligne le 20 Janvier 2015.

[23] William Dutcher. Inside dns beats the heart of a load balancer. *PC Week*, 15(20) :118, 1998.

[24] Ubuntu France. Bonding : Agrégation de plusieurs interfaces réseaux en une interface logique. http://doc.ubuntu-fr.org/bonding. Consulté en ligne le 30 Avril 2015.

[25] Wensley Group. Grub gpt howto. http://www.wensley.org.uk/gpt. Consulté en ligne le 25 Janvier 2015.

[26] Phil Hopkins. Troubleshooting openstack networking. http://2014.texaslinuxfest.org/sites/default/files/HopkinsPPTdeck.pdf. Consulté en ligne le 24 Février 2015.

[27] IDG Communications Pty Ltd. 5 open source cloud computing projects to watch. *CIO (13284045)*, page 1, 2011.

[28] Mirantis Inc. Mirantis puppet module for ceph. https://github.com/stackforge/fuel-library/tree/master/deployment/puppet/ceph. Consulté en ligne le 18 Janvier 2015.

[29] Jay Janssen. Percona xtradb cluster : Multi-node writing and unexpected deadlocks. https://www.percona.com/blog/2012/08/17/percona-xtradb-cluster-multi-node-writing-and-unexpected-deadlocks/. Consulté en ligne le 2 Mars 2015.

[30] Sean M. Kerner. Ubuntu, puppet, grizzly play key roles in openstack deployments. *eWeek*, page 7, 2014.

[31] Rafi Khardalian. Rabbitmq connections lack heartbeat or tcp keepalives. https://bugs.launchpad.net/nova/+bug/856764. Consulté en ligne le 6 Mars 2015.

[32] Konstantinos Kostantos, Andrew Kapsalis, Dimosthenis Kyriazis, Marinos Themistocleous, and Rupino da Cunha Paulo. Open-source iaas fit for purpose : A comparison between opennebula and openstack. *International Journal of Electronic Business Management*, 11(3) :191–201, 2013.

[33] Puppet Labs. Designing puppet : Roles/profiles pattern. https://puppetlabs.com/presentations/designing-puppet-rolesprofiles-pattern. Consulté en ligne le 25 Janvier 2015.

[34] Gary Larizza. Building a functional puppet workflow part 2 : Roles and profiles. http://garylarizza.com/blog/2014/02/17/puppet-workflow-part-2/. Consulté en ligne le 25 Janvier 2015.

[35] Percona LLC. Limitations de percona xtradb. http://www.percona.com/doc/percona-xtradb-cluster/5.5/limitation.html. Consulté en ligne le 28 Janvier 2015.

[36] Matthew MacAulay. Ubuntu 12.04.2 lts interface bonding (802.3ad / lacp) with vlan tagging (802.1q). http://insertscream.blogspot.fr/2013/05/ubuntu-12042-lts-interface-bonding.html. Consulté en ligne le 30 Avril 2015.

[37] Markess. Baromètre des prestataires du cloud computing en france. http://www.etatsgenerauxducloud.fr/doc/eg2014-markess.pdf. Consulté en ligne le 16 Octobre 2014.

[38] Markess. Le cloud computing dans les pme françaises - réalités, besoins & perspectives 2014. http://www.eurocloud.fr/doc/markess-cloud-pme-2012.pdf. Consulté en ligne le 16 Octobre 2014.

[39] K. Mestery. Openstack 2014.2 (juno) release notes - openstack network service (neutron). https://wiki.openstack.org/wiki/ReleaseNotes/Juno#OpenStack_Network_Service_.28Neutron.29. Consulté en ligne le 20 Janvier 2015.

[40] K. Mestery. What's new in neutron for openstack juno. http://www.siliconloons.com/whats-new-in-neutron-for-openstack-juno/. Consulté en ligne le 20 Janvier 2015.

[41] Mirantis. Reference architectures. https://docs.mirantis.com/fuel/fuel-3.2.1/reference-architecture.html. Consulté en ligne le 15 Janvier 2015.

[42] A. Muller. Juno advanced routing compatibility. http://assafmuller.com/2014/12/30/juno-advanced-routing-compatibility/. Consulté en ligne le 20 Janvier 2015.

[43] Assaf Muller. Gre tunnels in openstack neutron. http://assafmuller.com/2013/10/14/gre-tunnels-in-openstack-neutron/. Consulté en ligne le 5 Février 2015.

[44] National Institute of Standards and Technology. The NIST Definition of Cloud Computing. *Special Publication 800-145*, September 2011.

[45] d_inevitable Oguz Bilgic. How to get the hostname from a dhcp server. http://askubuntu.com/questions/104918/how-to-get-the-hostname-from-a-dhcp-server. Consulté en ligne le 20 Février 2015.

[46] Jiang Qingye. Opensource iaas community analysis. http://www.qyjohn.net/?p=3399. Consulté en ligne le 2 Octobre 2014.

[47] Ilya Shakhat. Openstack networking. http://fr.slideshare.net/shakhat/openstack-networking. Consulté en ligne le 25 Janvier 2015.

[48] Paul Sim. Openstack networking - juno - l3 h/a & dvr. http://fr.slideshare.net/janghoonsim/open-stack-networking-juno-l3-ha-dvr. Consulté en ligne le 5 Février 2015.

[49] Piotr Siwczak. Understanding your options : Deployment topologies for high availability (ha) with openstack. https://www.mirantis.com/blog/understanding-options-deployment-topologies-high-availability-ha-openstack/. Consulté en ligne le 15 Janvier 2015.

[50] Assaf Muller Sylvain Afchain. Openstack summet 2014 - neutron network node high availability. https://www.youtube.com/watch?v=go4fOYOUkmE.

[51] Eucalyptus Systems. Eucalyptus documentation. https://www.eucalyptus.com/docs/eucalyptus/4.0.2/index.html. Consulté en ligne le 23 Octobre 2014.

[52] Jay Vosburgh Thomas Davis. Linux ethernet bonding driver howto. http://www.cyberciti.biz/howto/question/static/linux-ethernet-bonding-driver-howto.php. Consulté en ligne le 30 Avril 2015.

[53] Ubuntu Wiki. Vlan. https://wiki.ubuntu.com/vlan. Consulté en ligne le 30 Avril 2015.

[54] Kimi Zhang. Building redundant and distributed l3 network in juno. https://kimizhang.wordpress.com/2014/11/25/building-redundant-and-distributed-l3-network-in-juno/. Consulté en ligne le 20 Janvier 2015.

Un datacenter dans le cloud
Public/HA IaaS

Rapport de projet de fin d'études 2015

Résumé : Dans le cadre de la fin de mes études à Polytech Tours au département informatique et de l'évolution de l'entreprise Harmony-Hosting que je dirige, ce projet s'intéresse à la mise en place d'une plateforme IaaS (*Infrastructure-as-a-service*) publique, hautement disponible, évolutive et automatisée. Ce rapport contient analyse préliminaire du contexte et des besoins - traduite en une série d'objectifs priorisés, puis un l'état de l'art, un chapitre concernant l'architecture de la plateforme suivi d'un chapitre détaillant la façon dont celle-ci est déployée de façon automatique. Ensuite, un chapitre décrit la façon dont les tests sont menés, automatisés et comment ils sont utilisés pour valider les objectifs. Finalement, la gestion de projet est discutée et un chapitre final conclut le travail et énonce des perspectives de continuité.

Mots clefs : IaaS, Cloud, systèmes distribués, Haute disponibilité, OpenStack, Open vSwitch

Abstract: In the framework of the end of my computer-science studies at Polytech Tours and of the development of Harmony-Hosting which I manage, this project is about setting up a public, highly available, scalable, and automated IaaS platform. This report contains a preliminary context and needs analysis - translated into a set of prioritized objectives, then a state of the art, a chapter about the platform architecture, followed by a chapter detailing how it is actually automatically deployed. Afterwards, a chapter describes the way tests are directed, automated and how they are used to validate the objectives. Finally, project management is discussed and a final chapter concludes the work and opens with future perspectives.

Keywords: IaaS, Cloud, Distributed systems, High availability, OpenStack, Open vSwitch

---

Encadrants
Sébastien AUPETIT
sebastien.aupetit@univ-tours.fr

Maître de conférences

Étudiants
Quentin MACHU
me@quentin-machu.fr

DI5 2014 - 2015