RAPPORT DE PROJET
Prise en main d'un environnement de Cloud : OpenStack

Groupe :
BENAIS Charles,
BRESSAND Jérémy,
CULTY Alexandre,
ROGLIANO Théo

Tutrice : BOUZIANE Hinde

2015 - 2016

Table des matières

Remerciements

Nous tenons tout d'abord à remercier Mme BOUZIANE Hinde, notre tutrice de projet, qui nous a permis de réaliser ce projet dans de bonnes conditions.

Nous voulons aussi remercier l'équipe Grid5000 qui nous à permis de réaliser ce projet sur leur plateforme distribuée par l'intermédiaire de notre tutrice, ce qui nous à offert la possibilité de réaliser ce projet dans un environnement réaliste et adapté à la notion de Cloud Computing.

1. Introduction

> *«Le Cloud Computing est l'ensemble des disciplines, technologies et modèles commerciaux utilisés pour délivrer des capacités informatiques (logiciels, plateformes, matériels) comme un service à la demande.» [12]*

Le Cloud est né à la suite de la multiplication des centres de données, de l'expension des communications et des capacités de calculs, ainsi que de la maitrise de la virtualisation.
La virtualisation consiste à offrir une certaine souplesse dans l'utilisation des ressources de son infrastructure informatique en permettant l'instanciation de multiples systèmes d'exploitation. Le Cloud (OpenStack) vient s'intégrer à la couche de virtualisation du système et offre une interface pour la réservation de capacités de calcul et de stockage ainsi que son orchestration. Ce point de vue offre les aspects fondamentaux de l'infrastructure en tant que service (Infrastructure as a Service ou IaaS), celle qui nous intéressera ici en particulier, et permet la mise en place d'autres services plus spécialisés, présentés succinctement ci-dessous.

— **IaaS** (Infrastructure as a Service) : donne accès à une machine virtuelle où l'utilisateur choisi son système d'exploitation et est libre d'installer les applications qu'il souhaite. OVH, Online, FirstHeberg sont des entreprises françaises offrant ce service à travers des offres de location de VPS (Virtual Private Server).

— **PaaS** (Platform as a Service) : le système d'exploitation est déjà installé, l'utilisateur installe les applications qu'il souhaite. Dans cette catégorie on trouve notamment les offres d'hébergement web mutualisées.

— **SaaS** (Software as a Service) : l'application est directement mis à disposition des utilisateurs, celui-ci ne s'occupe en rien de la gestion du serveur. Par exemple Gmail, Dropbox ou encore ShareLatex font parti de cette catégorie.

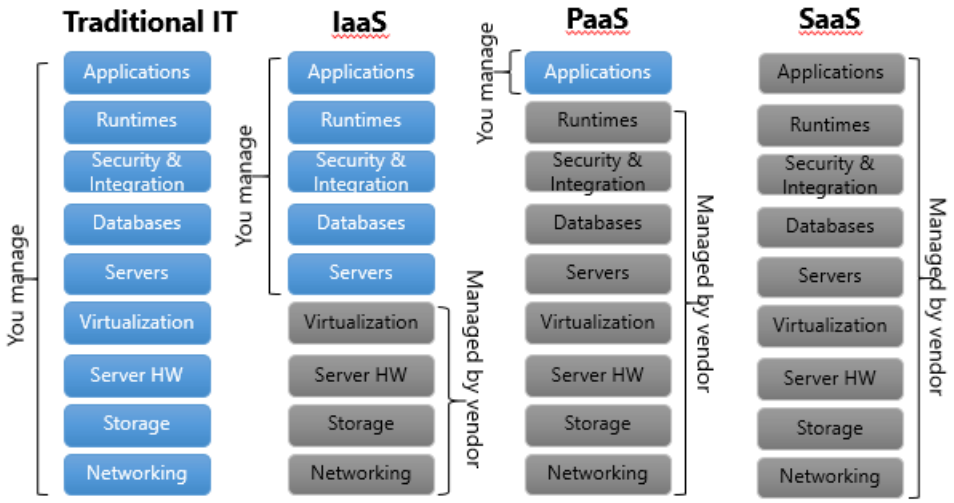


FIGURE 1.1 – Les différents niveaux de Clouds [1]

Le Cloud public est constitué de grandes entreprises offrant des Web-Services similaires en proposant la location d'une partie de son infrastructure informatique. Par exemple Amazon avec Elastic Compute Cloud (EC2) ou encore Microsoft avec Azure.
Présentons OpenStack, la problématique et les objectifs de notre projet, ainsi que l'organisation mise en place.

1.1 L'environnement de cloud OpenStack

> *«OpenStack est un ensemble de logiciels open source permettant de déployer des infrastructures de cloud computing (IaaS). La technologie possède une architecture modulaire composée de plusieurs projets corrélés (Nova, Swift, Glance...) qui permettent de contrôler les différentes ressources des machines virtuelles telles que la puissance de calcul, le stockage ou encore le réseau inhérents au centre de données sollicité.» [2]*

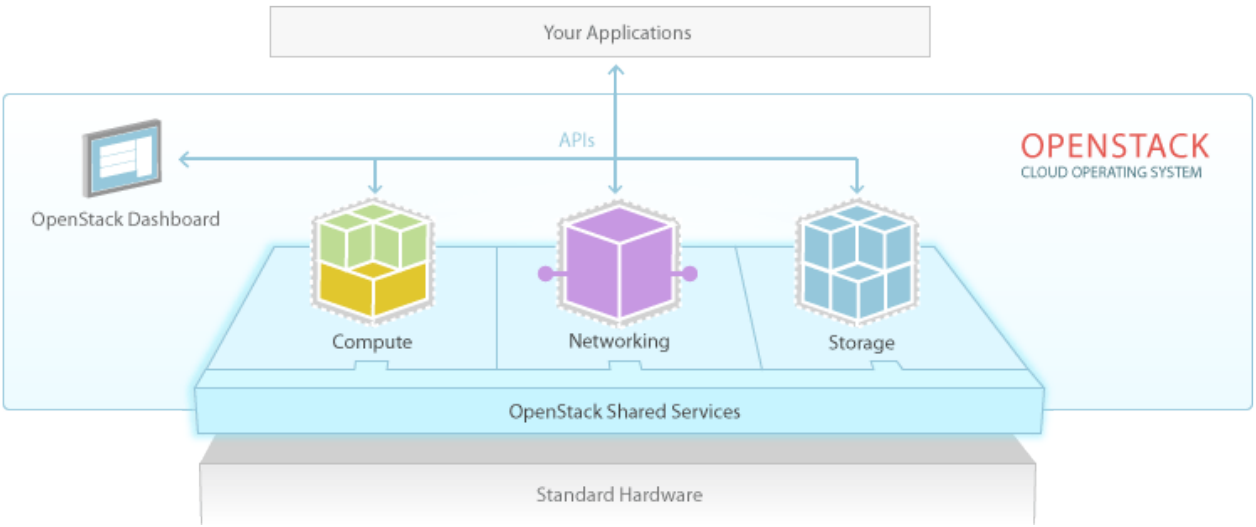


FIGURE 1.2 – OpenStack Software [3]

OpenStack est à ce jour composé de 17 logiciels (modules) mais nous nous intéresserons à un seul : Nova.

> *«Nova dirige le cycle de vie des instances de calcul dans un environnement OpenStack. Les responsabilités de Nova englobent la création, l'ordonnancement et le démantèlement des machines virtuelles sur demande.» [4]*

Nova gère la communication avec la plate-forme de virtualisation (l'hyperviseur KVM plus spécifiquement) intégrée au noyau Linux du système, permettant l'instantiation de plusieurs machines virtuelles sur une même machine physique. Nova est suffisant pour créer un environnement de cloud. En effet, il permet de créer un ensemble de machines virtuelles connectées au réseau publique de la machine hôte et d'y installer un système d'exploitation.

1.2 Problématique et objectifs du projet

Ce projet a pour mission principale la prise en main d'un environnement de cloud de type IaaS à travers OpenStack. Pour cela, nous devons mettre en oeuvre un système d'automatisation de déploiement d'applications sur des machines virtuelles.

Avec ces objectifs, nous avons défini les problématiques suivantes :
— Comment utiliser OpenStack pour créer des machines virtuelles ?
— Comment installer et démarrer des applications sur les machines virtuelles ?
— Comment automatiser le déploiement d'applications sur des machines virtuelles ?

1.3 Organisation du projet

Afin de mieux déterminer les tâches à effectuer, nous avons d'abord réalisé le diagramme de Gantt suivant :

Untitled Gantt Project

Tâches

Nom	Date de début	Date de fin
Installation + Prise en main OpenStack	18/01/16	23/02/16
Doc openstack/devstack	18/01/16	10/02/16
Deploiement single machine	18/01/16	10/02/16
Getting started grid5000	11/02/16	16/02/16
TP XP5K-openStack Grid5000	16/02/16	23/02/16
Procedure installation execution application	02/03/16	08/03/16
Mise en place application client serveur	02/03/16	08/03/16
Automatisation	09/03/16	29/03/16
Script déploiement application VM	09/03/16	15/03/16
Script création VM	09/03/16	15/03/16
Creation script nettoyage	16/03/16	22/03/16
Creation application serveur FTP	16/03/16	22/03/16
Automatisation déploiement application	16/03/16	22/03/16
Conception fichier utilisateur JSON	23/03/16	29/03/16
Diagramme de GANTT *Elaboration d'un diagramme de GANTT*	11/02/16	16/02/16
Redaction rapport	11/02/16	17/04/16
Plan du rapport	11/02/16	16/02/16
Redaction 1ere ébauche de rapport	17/02/16	05/04/16
Redaction 2eme ebauche de rapport	06/04/16	12/04/16
Finaliser rapport	13/04/16	17/04/16

FIGURE 1.3 – Liste des tâches de notre diagramme de gantt

Tout au long du projet nous nous sommes réunis en moyenne deux après-midi par semaine pour travailler. Nous tenions aussi une réunion hebdomadaire avec notre tutrice de projet afin de l'informer de nos avancements et de lui demander des conseils.

Dans la suite de ce rapport nous présenterons la prise en main de l'environnement OpenStack à travers nos premières expériences. Ensuite, nous développerons le travail effectué, notamment la conception de l'outil permettant le déploiement d'applications sur des machines virtuelles. Enfin, nous terminerons par un bilan de ce projet Cloud Computing à travers OpenStack.

2. Prise en main d'OpenStack

OpenStack met à disposition deux solutions pour mettre en place son cloud. La première est une interface web appelée dashboard (en français tableau de bord), mis à disposition grâce au module Horizon. La deuxième est de passer par une interface de programmation applicative (souvent désignée par le terme API pour Application Programming Interface). C'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. Afin d'atteindre nos objectifs, nous avons choisi d'utiliser l'API qui offre plus d'outils pour y parvenir.

Vous trouverez en annexe 4.1, page 15 une capture d'écran de l'interface web Horizon.

2.1 Installation d'OpenStack sur machines personnelles

Dans l'objectif de déployer un environnement OpenStack nous avons choisi d'utiliser DevStack, un projet qui a pour but de rendre l'installation d'OpenStack accessible (avec peu de près-requis) avec un script permettant le téléchargement et la configuration de tous les composants essentiels à OpenStack.[5]
Nos objectifs étaient de déployer OpenStack sur nos machines de manière individuelle dans un premier temps puis de déployer OpenStack sur un réseau comprenant l'ensemble de nos machines par la suite.

Il est conseillé d'utiliser DevStack sur des machines virtuelles car DevStack modifie la configuration réseau et la configuration des droits d'accès aux fichiers sur les machines hôtes. Faisant face à plusieurs difficultés lors de l'installation et du déploiement de DevStack, notre tutrice nous a offert la possibilité d'utiliser d'un environnement OpenStack dédié à une plateforme distribuée existante nommée Grid'5000.

2.2 OpenStack sur une plateforme distribuée

2.2.1 La plateforme Grid'5000

La plate-forme Grid'5000 est une grille informatique, c'est-à-dire une infrastructure distribuée sur différents sites géographiques (Grenoble, Lyon, Toulouse, Bordeaux, Lille, Rennes, Reims, Nantes, Nancy, Luxembourg). Elle a été créée par l'INRIA (Institut national de recherche en informatique et en automatique) afin de permettre des expériences à grande échelle dans le domaine informatique. [6]

Elle dispose de 1200 nœuds physiques. La totalité des nœuds représentent 2200 processeurs, soit 8000 cœurs. Les sites géographiques sont reliés par une fibre optique 10 Gbits/s faisant partie du réseau RENATER qui relie les différentes universités et centres de recherche français.

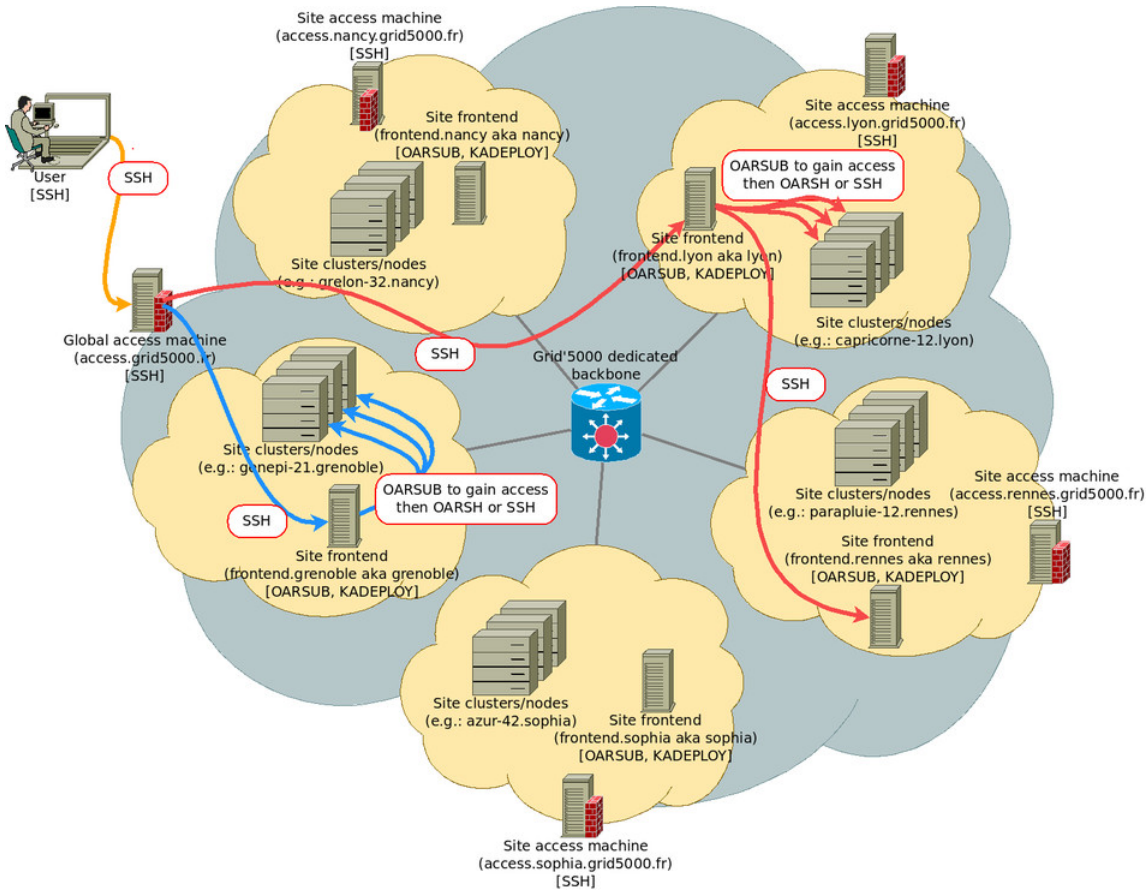


FIGURE 2.1 – Architecture de Grid'5000 [7]

Afin de se connecter à l'infrastructure G5K, il faut se connecter en SSH à acces.grid5000.fr, ce qui permet ensuite d'accéder aux différents sites sur lesquels il est possible d'effectuer des réservations de noeuds.

2.2.2 OpenStack sur Grid'5000 : XP5K-OpenStack

> «XP5K est une bibliothèque légère écrite en Ruby permettant d'aider les utilisateurs de Grid'5000 à préparer leur expérimentations en faisant appel à l'interface de programmation Grid'5000 pour :
> — Créer, consulter ou supprimer des réservations.
> — Créer des rôles (chaque nœud est dédié à un rôle, par exemple : contrôleur, unités de calcul), sans avoir à connaître le nom des nœuds qu'on utilise.
> — Faire un ou plusieurs déploiement sur des nœuds spécifiés par une réservation ou un rôle.» [8]

> «Puppet est un logiciel libre, écrit en Ruby, permettant la gestion de la configuration de serveurs esclaves. Il permet de gérer les déploiements système et applicatif.» [9]

XP5K-OpenStack est une interface de programmation conçue pour déployer OpenStack sur Grid'5000. XP5K-OpenStack utilise d'une part l'interface de programmation XP5K, permettant de manipuler les réservations et les nœuds sur Grid'5000, d'autre part l'outil de gestion de configuration Puppet configuré avec les modules Puppet-OpenStack qui, à partir d'un nœud maître (le puppet master) permet le déploiement d'OpenStack sur un ensemble de nœuds de calcul (esclave).

2.2.3 Déploiement et exécution d'applications sur XP5K-OpenStack

Après avoir compris les bases et le fonctionnement sur XP5K-OpenStack, nous avons identifié et listé les étapes nécessaires pour automatiser le déploiement des applications sur les machines virtuelles.

Une fois connecté sur Grid'5000 et XP5K-OpenStack importé sur le compte utilisateur, il faut l'éxécuter afin que celui ci déploie Openstack sur les noeuds réservés (durée de cette étape environ 30 minutes).

Une fois fait, il faut créer et paramétrer les machines virtuelles. À la fin de cette étape, il ne manque qu'à importer les fichiers nécessaires à l'installation de l'application, ou directement l'exécutable sur les machines virtuelles. Enfin, la dernière étape est simplement d'exécuter la ou les applications via SSH.

2.3 Réfléxion sur l'automatisation

Avant de commencer le développement de l'outil permettant l'automatisation du déploiement d'applications sur des machines virtuelles, nous avons d'abord réfléchi sur les parties les plus importantes à automatiser.
La manipulation d'OpenStack pour créer les machines virtuelles et y déployer des applications est le coeur du projet.
Cependant la connexion au compte et l'importation des fichiers d'applications sur le compte Grid'5000 étant des tâches secondaires, nous ne les avons pas automatisé.
Ces réflexions faites, nous avons commencé à développer cet outil d'automatisation.

3. Un outil pour automatiser le déploiement et l'exécution d'applications sur OpenStack

Pour la réalisation d'un outil d'automatisation, nous avons choisi une implémentation basée sur des scripts écrits en Bash et en Ruby. Un script est un code interprété qui, à la manière d'un script au théâtre édicte une succession d'actions à réaliser. Dans le cadre de ce projet les scripts servent à exécuter et coordonner les commandes de Nova automatiquement.

Dans un premier temps ce chapitre présente l'aspect algorithmique, suivi de l'aspect technique des scripts développés.

3.1 Description générale

La première étape du processus d'automatisation est la création d'un fichier de configuration structuré. Ce fichier contient des informations sur les machines virtuelles, tel que la puissance de calcul, le nombre et leurs noms. Il contient aussi des informations sur les applications devant être installées comme le nom, le type (client, serveur, ...), le port si besoin. Dans un second temps, ce fichier est l'entrée d'un script ; construct.rb ; (cf. p17 figure 4.1) qui analyse ces informations et les redistribue au script de création de machines virtuelles ; VMSetup.sh ; (cf. p19 figure 4.2) puis, pour chacune d'entre elles appelle le script d'installation ; appSetup.sh ; (cf. p22 figure 4.3)

3.2 Implémentation

Cette section détaille les algorithmes associés aux différents scripts. En premier le script construct.rb est présenté, en second VMSetup.sh et enfin appSetup.sh.

— **construct.rb** : analyse le fichier de configuration (config.json) et appelle les scripts de création de machine virtuelle (VMSetup.sh) et d'installation d'application (appSetup.sh).

```
Data: config.json
Begin
 décoder config.json;
for VM in VMs do
    for 1..nbClone do
        créer VM;
        if VM installée then
            for APP in APPs do
                installer APP;
            end
        end
    end
end
End
```

Algorithm 1: construct.rb

— **VMSetup.sh** : créée une machine virtelle.

Algorithme VMSetup

```
Algorithme : VMSetup
Entrée :- tailleVM : taille de la machine virtuelle à créer.
        - nomVM : nom de la machine virtuelle à créer.


Debut :
    Vérification tailleVM valide.
    Vérification nomVM valide
        ( nom n'étant pas déjà utilisé par une autre machine virtuelle ).
    Ajout de la clé de chiffrement dans la configuration Nova.
    Créer la machine virtuelle.
    Créer une IP publique.
    Récupérer adresse IP créée.
    Attribuer adresse IP à la machine virtuelle.
    Connection à la machine virtuelle.
    Mise à jour des briques logicielles de base.
    Fin de la connection.
Fin algorithme.
```

— **appSetup.sh** : installe et démarre une application.

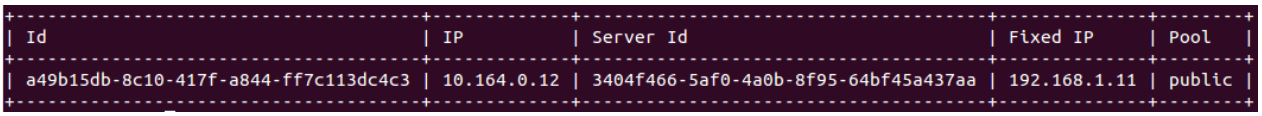
Algorithme appSetup

```
Algorithme : appSetup
Entrée :- nomVM : nom d'une machine virtuelle
        - nomAPP : nom de l'application à installer
        - typeApp : type d'application
        - pathAPP : chemin vers le repertoire
        - portAPP : port si application communicante
        - serveurAPP : nom de la VM jouant le role de serveur

Debut :
        Vérification de la validité des arguments.
        Récupération de l'IP de la machine virtuelle ciblée par nomVM.
        Copie des fichiers présents à pathApp vers la machine cible.
        Connection à la machine cible.
        Installation de l'application sur la machine cible.
        SI (typeAPP == client)
         | On execute l'application en passant serveurAPP en paramètre.
        SINON
         | On execute l'application sans argument.
        FINSI
Fin algorithme.
```

3.3 Vérification

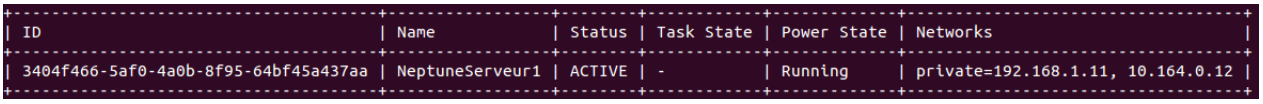
Pour les étapes où une vérification est nécessaire, Nova nous fournit deux commandes. La première : nova floating-ip-list, permet de voir la liste des IP créées.



Id	IP	Server Id	Fixed IP	Pool
a49b15db-8c10-417f-a844-ff7c113dc4c3	10.164.0.12	3404f466-5af0-4a0b-8f95-64bf45a437aa	192.168.1.11	public

FIGURE 3.1 – Affichage nova floating-ip-list

La seconde : nova list, permet de voir les machines virtuelles créées ainsi que leurs caractéristiques, notamment si une IP a été liée à une machine.



ID	Name	Status	Task State	Power State	Networks
3404f466-5af0-4a0b-8f95-64bf45a437aa	NeptuneServeur1	ACTIVE	-	Running	private=192.168.1.11, 10.164.0.12

FIGURE 3.2 – Affichage nova list

Nous avons testé notre script avec plusieurs applications de type client/serveur que nous avions programmé lors du précédent semestre. En premier, l'application Perroquet dans laquelle un client envoie un message au serveur et le serveur renvoie ce message au client. En deuxième l'application Chat dans laquelle un client envoie un message au serveur et le serveur transmet ce message aux autres clients. Pour terminer nous avons déployé l'application FTP dans laquelle le serveur transmet un ou plusieurs fichiers au client.
Pour vérifier l'exécution d'une application, il y a principalement deux façons de faire, la plus simple étant de récupérer un résultat dans un fichier de log. L'autre façon étant de se connecter à la machine virtuelle et de vérifier les processus actifs (par exemple : ps -ef | grep -o nom_Du_Prog).

3.4 Améliorations

Pour configurer le déploiement d'applications avec notre outil, l'utilisateur doit manipuler un fichier de configuration au format JSON. Ce format ne permet pas facilement de décrire l'installation d'applications complexes, qui prendraient plusieurs paramètres, ou qui auraient par exemple un ensemble de machines virtuelles qui communiqueraient entre elles de manière plus complexe que le cas d'une application client serveur.

Une solution pour configurer de manière plus simple le déploiement d'application serait d'utiliser Puppet qui permet de configurer et de déployer des applications complexes sur des machines virtuelles grâce à des templates s'adaptant à la configuration de la machine.
Puppet reste peu accessible pour un utilisateur, mais offre la possibilité de déplacer les informations concernant la configuration des machines dans une base de donnée.
L'idée serait donc de créer une application graphique, facile d'utilisation pour l'utilisateur, qui modifiera une base de données dans laquelle Puppet irait chercher les informations pour configurer les noeuds.

Pour manipuler les terminaux sur les différentes machines virtuelles, il est possible d'utiliser le programme Screen.

«Screen (GNU Screen) est un « multiplexeur de terminaux » permettant d'ouvrir plusieurs terminaux dans une même console, de passer de l'un à l'autre et de les récupérer plus tard.» [10]

Screen permettant d'afficher plusieurs terminaux, notre programme pourrait utiliser Screen pour afficher les terminaux des différentes machines virtuelles sur une même fenêtre.

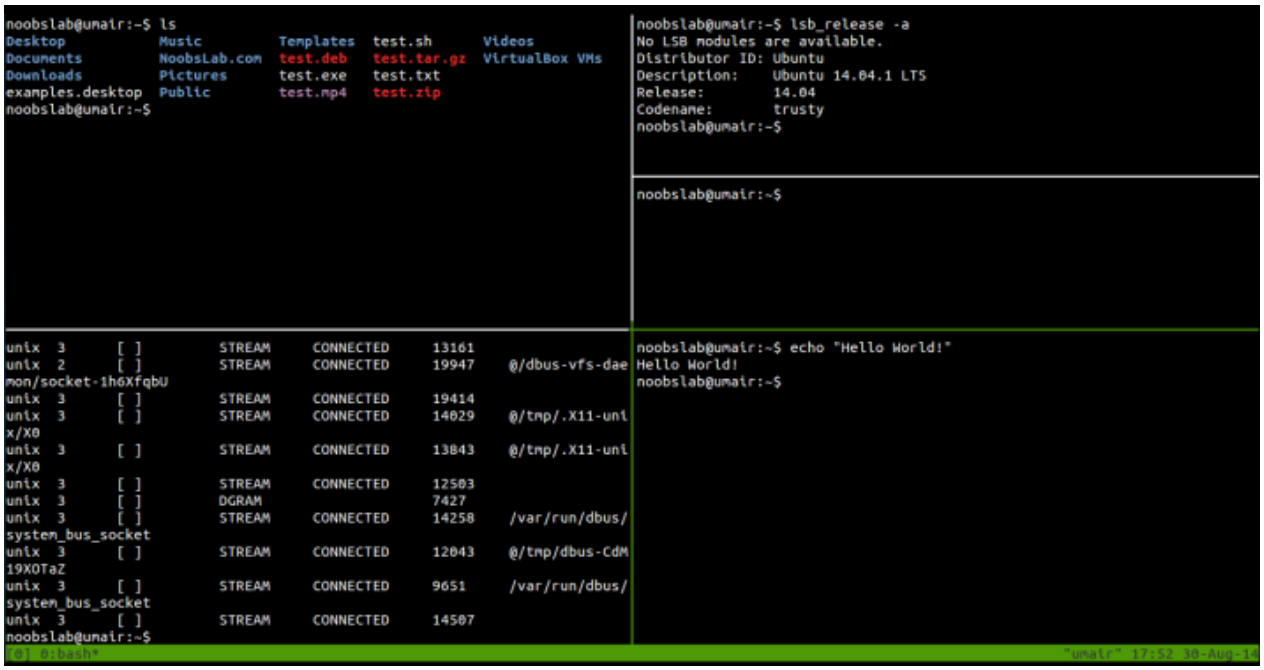


FIGURE 3.3 – Affichage de plusieurs terminaux virtuels sur une même fenêtre avec Screen [11]

4. Conclusion

L'objectif principal de ce projet était la prise en main d'un environnement de Cloud à travers OpenStack en mettant en oeuvre un système d'automatisation de déploiement d'applications sur des machines virtuelles.

Après quelques difficultés liés à l'utilisation de DevStack, nous avons réussi à nous intégrer à la plateforme de Grid'5000. Petit à petit, nous avons conçu des scripts réalisant des parties de notre objectif, pour arriver à un script global qui permet d'automatiser le déploiement d'une ou plusieurs applications sur une ou plusieurs machines virtuelles.

L'utilisation de Grid'5000 nous a permis d'acquérir d'importantes notions sur les infrastructures informatiques distribuées.

Ce projet nous a aussi permis d'acquérir compétences et notions dans le domaine du Cloud Computing.

Bibliographie

[1] URL : https://en.wikipedia.org/wiki/Cloud_computing#Infrastructure_as_a_service_.28IaaS.29.

[2] URL : https://fr.wikipedia.org/wiki/OpenStack.

[3] URL : http://www.openstack.org/software/.

[4] URL : http://www.openstack.org/software/releases/liberty/components/nova.

[5] URL : http://docs.openstack.org/developer/devstack/.

[6] URL : https://www.grid5000.fr/.

[7] URL : www.grid5000.fr/mediawiki/index.php/Getting%5C_Started.

[8] URL : https://github.com/pmorillon/xp5k.

[9] URL : https://fr.wikipedia.org/wiki/Puppet.

[10] URL : https://doc.ubuntu-fr.org/screen.

[11] URL : www.noobslab.com/2014/08/split-ubuntugnome-terminal-screen-and.html.

[12] Sylvain CAICOYA Jean-George Saury. *Cloud Computing le guide complet*. Micro Application. 2011.

Annexes

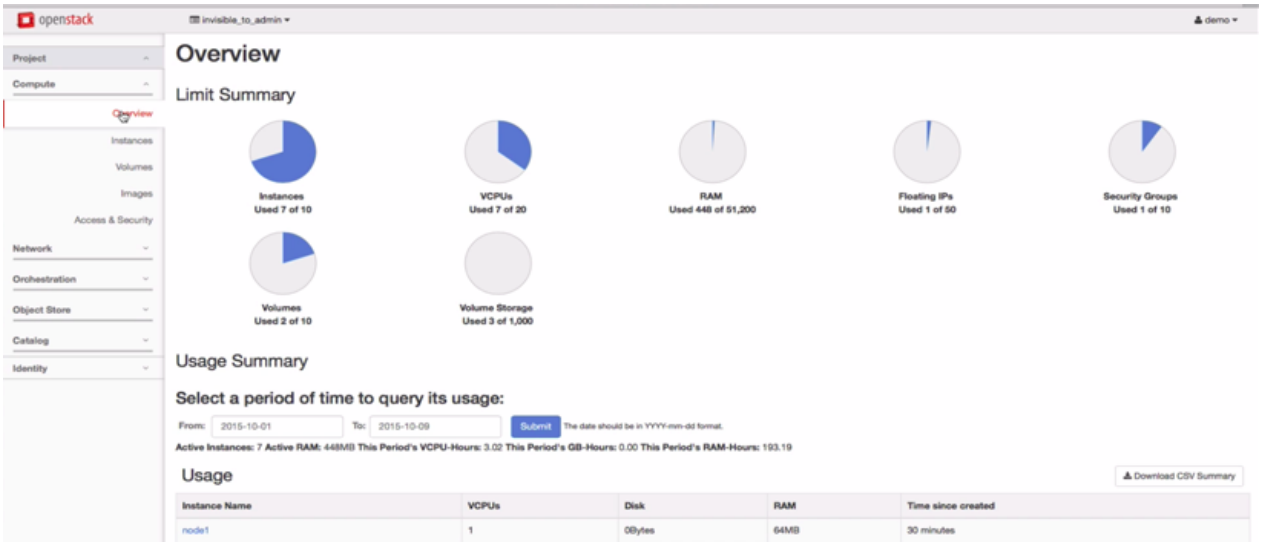


FIGURE 4.1 – Interface web Horizon (dashoard)

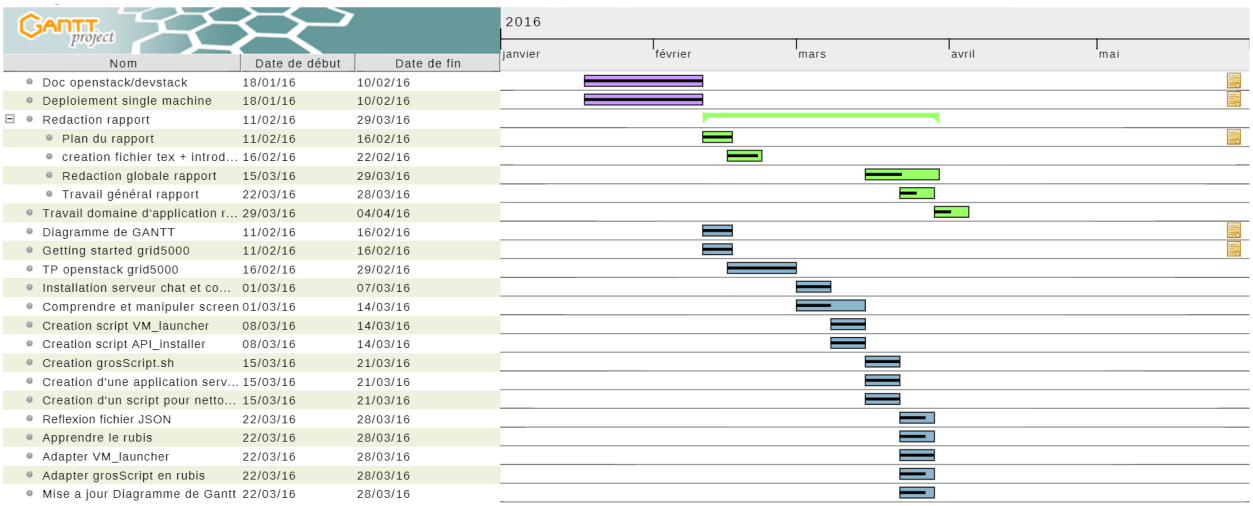


FIGURE 4.2 – Diagramme de Gantt

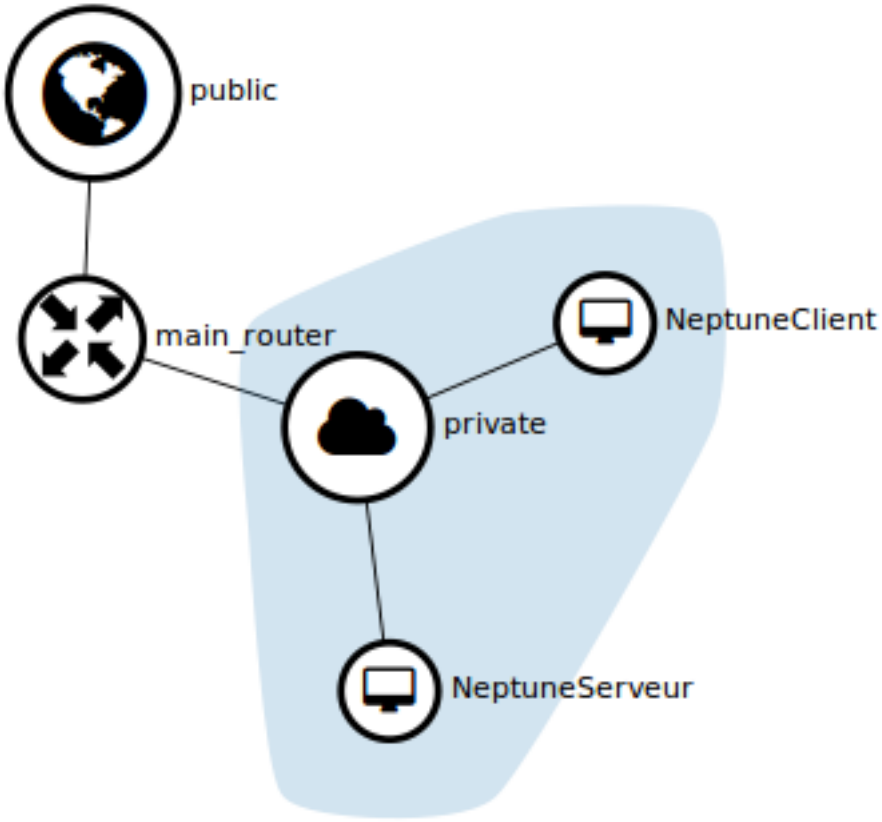


FIGURE 4.3 – Vue du cloud offerte par Horizon

Listing 4.1 – construct.rb

```
require 'rubygems'
require 'json'

file = File.read('config.json')
data = JSON.parse(file)

def getPortServeur(data, nomVM, nomAPP)
  data["vms"].each do |vm|
  if vm["nom"] == nomVM
    vm["apps"].each do |app|
        if app["nom"] == nomAPP and app["type"] == "serveur"
          return app["port"]
        end
      end
    end
  end
  return false
end

data["vms"].each do |vm|
  print "\n"
  for numVM in 1..vm["nombreCopie"]
    puts "### Ã©Cration de la machine virtuelle '#{vm["nom"]}
    #{numVM}'"
    resVM = system("./VMSetup.sh \"#{vm["type"]}\" \"#{vm["nom"]}
    #{numVM}\"")
    if resVM != false
      puts "### Machine virtuelle Ã©Ã©cre avec Ã¨ succs
      (#{vm["nom"]}#{numVM})"
      vm["apps"].each do |app|
        print "\n"
        if app["publique"] == true
          rep = "pub"
        else
          rep = "pers"
        end
        if File.directory?("../apps/#{rep}/#{app["nom"]}") == true
          puts "### Installation de l'application '#{app["nom"]}'"
          if app["type"] == "serveur"
            resApp = system("./appSetup.sh \"#{vm["nom"]}
            #{numVM}\" \"#{app["nom"]}\" \"#{app["type"]}\"
            \"../apps/#{rep}/#{app["nom"]}\" \"#{app["port"]}\"")
                else
            portServeur = getPortServeur(data, app["serveur"],
            app["nom"])
            resApp = system("./appSetup.sh \"#{vm["nom"]}
            #{numVM}\" \"#{app["nom"]}\" \"#{app["type"]}\"
              \"../apps/#{rep}/#{app["nom"]}\" \"
            #{app["portServeur"]}\"
```

```
            \"#{app["serveur"]}\"")
          end
          if resApp != false
            puts "### Application Ã©installe avec Ã¨succs
                (#{app["nom"]})"
          else
            puts "### Une erreur est survenue, l'application
            (#{app["nom"]})
            n'a pas Ã©Ã©t Ã©installe"
          end
        else
          puts "### Application introuvable"
        end
      end
    else
      puts "### Une erreur est survenue, la machine virtuelle
      (#{vm["nom"]}#{numVM}) n'a pas Ã©Ã©t Ã©Ã©cre"
    end
    print "\n\n"
  end
end
print "\n"
```

Listing 4.2 – VM_Setup.sh

```
#!/bin/bash

ERR_ARGS=-1

if [ $# -ne 2 ]; then
        echo "Usage: `basename $0` taille_VM nom_VM"
        exit $ERR_ARGS
fi

echo "- Verification de la taille de la VM ($1)";
case $1 in
    xs|small|medium|large|xlarge)
        ;;
    *)
    echo "taille_VM: xs | small | medium | large | xlarge : $1";
    exit $ERR_ARGS;;
esac

mkdir -p logs/$2/
LOG="logs/$2/VMSetup.log"

# On recupere l'adresse du controller
ADR=`rake roles:show | grep 'controller' |
grep -o -E '[^: ]*\.grid5000\.fr'`;
echo "# Controleur: $ADR" >> $LOG

KEYPAIR=`ssh root@$ADR 'source openstack-openrc.sh &&
nova keypair-list' | grep -o -E 'mainKey'`;
if [ -z $KEYPAIR ]; then
    echo "- Ajout de la Key Pair";
    cat ~/.ssh/id_rsa.pub | ssh root@$ADR "source openstack-
    openrc.sh && nova keypair-add --pub_key - mainKey"
fi

NOMSVMS=`ssh root@$ADR 'source openstack-openrc.sh && nova list' |
cut -d '|' -f 3 | grep -o -E '([a-zA-Z0-9_]+)' | grep -v 'Name'`;

echo "- Verification du nom de la VM ($2)";
if ! [[ "$2" =~ ^[a-zA-Z0-9_]+$ ]]; then
    echo "Le nom de la VM doit Ãªtre alphanumerique (a-zA-Z0-9_)"
    exit $ERR_ARGS
fi
for NOMVM in $NOMSVMS; do
        if [ "$NOMVM" = "$2" ]; then
            echo "Le nom de la VM $2 existe deja,
            veuillez en choisir un autre"
            exit $ERR_ARGS
        fi
done
```

```
echo "- Creation de la VM...";
rake cmd cmd="
echo '# Execution du source openstack';
source openstack-openrc.sh;

VAR_RULE=\`nova secgroup-list-rules default | grep -o 10000\`;
if [ -z \"\$VAR_RULE\" ]; then
    echo '# Ajout des regles du parfeu'
    nova secgroup-add-rule default tcp 10000 10100 0.0.0.0/0;
    nova secgroup-add-rule default udp 10000 10100 0.0.0.0/0;
fi

echo '# Creation de la VM';
nova boot --flavor m1.$1 --image 'Debian Jessie 64-bit'
--nic net-id=\$(neutron net-show -c id -f value private)
--key_name mainKey $2

echo '# On attend que la VM soit disponible';
while [ \"\`nova list --name $2 | grep -o -E 'ACTIVE|ERROR'\`\" !=
'ACTIVE' ]; do
    if [ \"\`nova list --name $2 | grep -o -E 'ACTIVE|ERROR'\`\" =
    'ERROR' ]; then
        nova delete $2;
        nova boot --flavor m1.$1 --image 'Debian Jessie 64-bit'
        --nic net-id=
        \$(neutron net-show -c id -f value private) --key_name
        mainKey $2
    fi
    sleep 2;
done

echo '# Creation de l IP publique';
IP_PUB=\`nova floating-ip-create public | grep -o -E
'(([0-9]{1,3}\.){3}[0-9]{1,3})'\`;
echo \"# Ajout de l IP publique (\$IP_PUB) a la VM\";
nova add-floating-ip $2 \$IP_PUB;
" host=controller >> $LOG

echo "- VM creee";

# Connexion au controleur pour recuperer l'IP de la VM
IP=`ssh root@$ADR "source openstack-openrc.sh && nova list --name
$2" | grep -o -E '(10\.([0-9]{1,3}\.){2}[0-9]{1,3})'`;

echo "- En attente de connexion ssh disponible ($IP)";
while ! ssh -q debian@$IP 'exit'; do
    sleep 2;
done
```

```
echo "- Mise a jour de la VM...";
ssh -q debian@$IP "sudo apt-get -y update; sudo apt-get -y
install gcc make;" >> $LOG;
echo "- Mise a jour reussie";

exit 0
```

Listing 4.3 – APP_Setup.sh

```
#!/bin/bash

ERR_ARGS=-1

if [ $# -ne 5 ] && [ $# -ne 6 ]; then
    echo "Usage: `basename $0` nom_VM nom_APP type_APP path_APP
    port_APP
    [type_APP=client=>serveur_APP]"
    exit $ERR_ARGS
fi

echo "- Verification du type de l'application et des parametres";
case $3 in
    client)
    if [ $# -ne 6 ]; then
        echo "Usage: `basename $0` nom_VM nom_APP type_APP
        path_APP port_APP serveur_APP"
        exit $ERR_ARGS
    fi
    ;;
    normal)
        ;;
    serveur)
    echo "- Verification du port de l'application";
    if [ $5 -lt 10000 ] && [ $5 -gt 10100 ]; then
        echo "port_APP: 10000 a 10100 : $5"
        exit $ERR_ARGS
    fi
    ;;
    *)
    echo "type_APP: client | serveur | normal : $3"
    exit $ERR_ARGS;;
esac

mkdir -p logs/$1/
LOG="logs/$1/appSetup.log"

# On recupere l'adresse du controller
ADR=`rake roles:show | grep 'controller' |
grep -o -E '[^: ]*\.grid5000\.fr'`;

# On se connecte au controleur pour recuperer l'IP de la VM
IP=`ssh root@$ADR "source openstack-openrc.sh && nova list
--name $1"
| cut -d '|' -f 7 | grep -o -E
'(10\.([0-9]{1,3}\.){2}[0-9]{1,3})'`;

if [ "$3" = "client" ]; then
    # On se connecte au controleur pour recuperer l'IP de la
```

```
    VM serveur
    IPServeur=`ssh root@$ADR "source openstack-openrc.sh &&
    nova list --name $6"
    | cut -d '|' -f 7 | grep -o -E
    '(10\.([0-9]{1,3}\.){2}[0-9]{1,3})'`;
    echo "IP VM serveur : $6:$IPServeur"
    if [ -z "$IPServeur" ]; then
        echo "L'IP de la VM serveur est introuvable"
        exit $ERR_ARGS
    fi
fi

echo "- Copie de l'application ($2) sur la VM"
scp -q -p -r $4 debian@$IP: >> $LOG

echo "- Demarrage de l'application"
if [ "$3" = "client" ]; then
    ssh -q debian@$IP "cd $2; make $3; ls -l; ./$3 $IPServeur $5"
    >> $LOG &
else
    ssh -q debian@$IP "cd $2; make $3; ls -l; ./$3 $5" >> $LOG &
fi

exit 0
```