Test de performance

Afin de tester la performance, nous avons fait 40 appels au serveur web de notre nuage sur les deux types d'architectures que nous avons mis en place grâce un script fourni en annexes. Voici les résultats :

Temps serveur simple : 64 secondes
Temps serveur réparti : 20 secondes

On peut voir que le script prend beaucoup plus de temps sur le serveur simple. Ceci est normal puisque un appel doit attendre que le précédent finisse avant de pouvoir être traiter. Avec le serveur réparti, le temps en moindre car nous profitons du parallélisme puisque deux serveurs travaillent en même. De plus, grâce au *LoadBalancer,* dès qu'il y a un appel, si un serveur est disponible, il effectuera cet appel.

Question 1:

OpenStack est en fait un regroupement de plusieurs composants distincts provenant originalement d'autres projets. Voici la description de cinq de ces composants.

Heat :

Le composant Heat est le projet principal de l'orchestration d'OpenStack. C'est le moteur pour pouvoir lancer de multiple applications dans le nuage en se basant sur un gabarit de la forme d'un fichier texte qui peut être traité comme du code. Il permet aussi d'utiliser le gabarit au format AWS CloudFormation d'Amazon. Le gabarit Heat permet de décrire l'infrastructure pour une application nuagique en spécifiant les relations entre les différentes ressources du nuage. Le gabarit Heat est lisible par un humain ainsi que par les machines.

Neutron :

Le composant Neutron est un projet qui travaille sur le noeud du réseau. Il permet de donner les services lié au réseau et donc de manipuler les réseaux et l'adressage IP au sein d'OpenStack. Grâce à ce service, les utilisateurs peuvent créer leurs propres réseaux et contrôler le trafic avec des groupes de sécurité. Neutron permet aussi de connecter les instances à un réseau et de gérer l'adressage IP des instances en leurs assignants une IP statique ou flottante comme nous l'avons fait pendant ce TP puisque nous avons donné une IP flottante à une instance présente sur le réseau afin de pouvoir communiquer avec les instances à l'intérieur du réseau. Dans son architecture, Neutron a été basé sur la philosophie dite SDN.

Nova :

Nova est un des composants principal d'OpenStack. Il permet de gérer les ressources de calculs. Il permet de contrôler les hyperviseurs par l'intermédiaire de la libvirt ou bien grâce à

aux API des hyperviseurs. L'architecture de Nova permet d'évoluer horizontalement et donc de rajouter du matériel.

Swift :

Le composant Swift est un service de stockage d'objet. C'est un système de stockage de données redondant et évolutif. Afin d'assurer l'intégrité des données, les fichiers sont écrit sur plusieurs disque dur répartis sur plusieurs serveurs. Comme Nova, il évolue de façon horizontale en ajoutant simplement de nouveaux serveurs. Si un disque ou un serveur tombe en panne, Swift réplique les données des noeud actif qui ont les mêmes informations dans d'autres serveurs. Swift est applicatif et permet donc l'utilisation de matériel peu coûteux et non spécialisé.

Horizon :

Horizon est le tableau de bord d'OpenStack. C'est une application Web qui permet la gestion du nuage grâce à une interface graphique. Cette application est accessible aux administrateurs ainsi qu'aux utilisateurs. La majorité des informations d'Horizon viennent des API REST des autres composants d'OpenStack comme celles décrites plus haut. Le code est libre comme toute les briques d'OpenStack et permet donc à certaines sociétés de pouvoir rajouter leurs propres services dans l'application.

Question 2:

La ressource OS::Heat::ResourceGroup permet d'instancier un groupe de noeud de calculs basé sur l'architecture décrite dans OS::Nova::Server.

La ressource OS::Neutron::HealthMonitor permet de s'assurer du bon fonctionnement des serveurs et de la marche à suivre en cas de panne suspectée.

La ressource OS::Neutron::Pool permet de regrouper une ensemble d'appareil comme par exemple des serveurs afin qu'ils reçoivent le trafic.

La ressource OS::Neutron::LoadBalancer permet de répartir les requêtes entres les deux serveurs.

La ressource OS::Nova::Server permet d'instancier les serveurs qui vont effectuer les calculs du serveur.

Question 3:

La ressource permettant de mieux gérer les ressources en modifiant dynamiquement le nombre d'instance du serveur s'appelle Heat::AutoScalingGroup, elle permet de configurer des Heat::ScalingPolicy qui sont les modifications de comportements à apporter lorsque le contrôleur Ceilometer::Alarm détecte les conditions d'exécution et lance une alerte.

La ressource permettant de lancer une alerte lorsque le taux d'utilisation du CPU de vos machines atteint des seuils prédéfinis est Ceilometer::Alarm.

La ressource permettant d'ajuster automatiquement le nombre de machines virtuelles en fonction de ces alertes est Heat::ScalingPolicy.

Expliquez les paramètres nécessaires pour chaque ressource.

Ceilometer a besoin du nom du compteur à surveiller (meter_name) et du seuil à ne pas passer (treshold).

ScalingPolicy a besoin du type de changement à apporter (adjustment type), de l'id du groupe d'auto-scaling sur lequel on veut appliquer la règle (auto_scaling_group_id) et de la taille de l'ajustement (scaling-adjustment).

Références :

1. OpenStack Orchestration In Depth, Part IV: Scaling
2. Heat tempalte autoscaling error
3. Welcome to the Heat documentation!
4. AWS CloudFormation
5. OpenStack
6. OpenStack Resource Types

Annexes

Script de test :

```
#!/bin/bash

debut="$(date +%s)"
for i in `seq 1 40`;
do
        wget -q 172.15.105.191:8000&
done
wait
fin="$(date +%s)"
let "temps = fin - debut"
echo $temps
```