FORMATION OPENSTACK ADMINISTRATEUR

osones

CONCERNANT CES SUPPORTS DE COURS 1/2

Supports de cours réalisés par **Osones**

https://osones.com

Logo Osones

AUTEURS

- Adrien Cunin adrien.cunin@osones.com
- Pierre Freund pierre.freund@osones.com
- Jean-François Taltavull jft@osones.com
- Romain Guichard romain.guichard@osones.com
- Kevin Lefevre kevin.lefevre@osones.com

CONCERNANT CES SUPPORTS DE COURS 2/2

- Copyright © 2014-2016 Osones
- Licence : Creative Commons BY-SA 4.0

- Sources : https://github.com/Osones/Formations/
- Online : https://osones.com/formations.html

Licence Creative Commons BY-SA 4.0

OBJECTIFS DE LA FORMATION : CLOUD

- Comprendre les principes du cloud et son intérêt
- Connaitre le vocabulaire inhérent au cloud
- Avoir une vue d’ensemble sur les solutions existantes en cloud public et privé
- Posséder les clés pour tirer parti au mieux du IaaS
- Pouvoir déterminer ce qui est compatible avec la philosophie cloud ou pas
- Adapter ses méthodes d’administration système à un environnement cloud

OBJECTIFS DE LA FORMATION : OPENSTACK

- Connaitre le fonctionnement du projet OpenStack et ses possibilités
- Comprendre le fonctionnement de chacun des composants d'OpenStack
- Pouvoir faire les bons choix de configuration
- Savoir déployer manuellement un cloud

- Savoir déployer manuellement un cloud OpenStack pour fournir du IaaS
- Connaitre les bonnes pratiques de déploiement d'OpenStack
- Être capable de déterminer l'origine d'une erreur dans OpenStack
- Savoir réagir face à un bug

PRÉ-REQUIS DE LA FORMATION

- Compétences d’administration système Linux tel qu’Ubuntu
  - Gestion des paquets
  - Manipulation de fichiers de configuration et de services
  - LVM et systèmes de fichiers
- Notions :
  - Virtualisation : KVM, libvirt
  - Réseau : iptables, namespaces
- Peut servir :
  - À l’aise dans un environnement Python

DÉFINITION FORMELLE

CARACTÉRISTIQUES

Fournir un (des) service(s)...

- Self service
- À travers le réseau
- Mutualisation des ressources
- Élasticité rapide
- Mesurabilité

Inspiré de la définition du NIST

http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nist145.pdf

SELF SERVICE

- L'utilisateur accède *directement* au service
- Pas d'intermédiaire humain
- Réponses immédiates
- Catalogue de services permettant leur découverte

À TRAVERS LE RÉSEAU

- L'utilisateur accède au service à travers le réseau
- Le *fournisseur* du service est distant du *consommateur*
- Réseau = internet ou pas
- Utilisation de protocoles réseaux standards (typiquement : HTTP)

MUTUALISATION DES RESSOURCES

- Un cloud propose ses services à de multiples utilisateurs/organisations → *Multi-tenant*
- Les ressources sont disponibles en grandes quantités
- La localisation précise et le taux d'occupation des ressources ne sont pas visibles

ÉLASTICITÉ RAPIDE

- Provisionning et suppression des ressources quasi instantané
- Possibilité d'automatiser ces actions de *scaling* (passage à l'échelle)
- Virtuellement pas de limite à cette élasticité

MESURABILITÉ

- L'utilisation des ressources cloud est monitorée par le fournisseur
- Le fournisseur peut gérer son *capacity planning* à partir de ces informations
- L'utilisateur est facturé en fonction de son usage précis des ressources

MODÈLES

On distingue :

- modèles de service : IaaS, PaaS, SaaS
- modèles de déploiement : public, privé, hybride

IAAS

- *Infrastructure as a Service*
- Infrastructure :
  - Compute (calcul)
  - Storage (stockage)
  - Network (réseau)
- Utilisateurs cibles : administrateurs (système, stockage, réseau)

PAAS

- *Platform as a Service*
- Environnement permettant de développer/déployer une application
- Spécifique à un language/framework (exemple : Python/Django)
- Ressources plus haut niveau que l'infrastructure, exemple : BDD
- Utilisateurs cibles : développeurs d'application

SAAS

- *Software as a Service*
- Utilisateurs cibles : utilisateurs finaux

QUELQUECHOSE AS A SERVICE ?

- Load balancing as a Service (Infra)
- Database as a Service (Platform)
- MonApplication as a Service (Software)
- etc.

LES MODÈLES DE SERVICE EN UN SCHÉMA

IaaS - PaaS - SaaS

CLOUD PUBLIC OU PRIVÉ ?

À qui s'adresse le cloud ?

- Public : tout le monde, disponible sur internet
- Privé : à une organisation, disponible sur son réseau

Concernant le cloud hybride : utilisation mixte de multiples clouds privés et/ou publics

CLOUD HYBRIDE

- Concept séduisant
- Mise en œuvre a priori difficile
- Certains cas d'usages s'y prêtent très bien
  - Exemple : jobs de CI
- *Cloud bursting*

L'INSTANT VIRTUALISATION

Mise au point.

- La virtualisation est une technologie permettant d'implémenter la fonction *compute*
- Un cloud fournissant du compute *peut* utiliser la virtualisation
- Mais peut également utiliser :
  - Du bare-metal
  - Des containers

LES APIS, LA CLÉ DU CLOUD

- Interface de programmation (via le réseau, souvent HTTP)
- Frontière explicite entre le fournisseur (provider) et l'utilisateur (user)
- Définit la manière dont l'utilisateur communique avec le cloud pour gérer ses ressources
- Gérer : CRUD (Create, Read, Update, Delete)
- REST

POURQUOI LE CLOUD ? CÔTÉ ÉCONOMIQUE

- Appréhender les ressources IT comme des services “fournisseur”
- Faire glisser le budget “investissement” (Capex) vers le budget “fonctionnement” (Opex)
- Réduire les coûts en mutualisant les ressources, et éventuellement avec des économies d'échelle
- Réduire les délais
- Aligner les coûts sur la consommation réelle des ressources

POURQUOI LE CLOUD ? CÔTÉ TECHNIQUE

- Abstraire les couches basses (serveur, réseau, OS, stockage)
- S’affranchir de l’administration technique des ressources et services (BDD, pare-feux, load-balancing, etc.)
- Concevoir des infrastructures scalables à la volée
- Agir sur les ressources via des lignes de code et gérer les infrastructures “comme du code”

LE MARCHÉ

AMAZON WEB SERVICES (AWS), LE LEADER

- Lancement en 2006
- À l'origine : services web "e-commerce" pour développeurs
- Puis : d'autres services pour développeurs
- Et enfin : services d'infrastructure
- Récemment, SaaS

ALTERNATIVES IAAS PUBLICS À AWS

- Google Cloud Platform
- Microsoft Azure
- Rackspace
- DreamHost
- DigitalOcean
- En France :
  - Cloudwatt

- Cloudwatt
- Numergy
- OVH
- Ikoula
- Scaleway
- Outscale

FAIRE DU IAAS PRIVÉ

- OpenStack
- CloudStack
- Eucalyptus
- OpenNebula

OPENSTACK EN QUELQUES MOTS

- Naissance en 2010
- Fondation OpenStack depuis 2012
- Écrit en Python et distribué sous licence Apache 2.0
- Soutien très large de l'industrie et contributions variées

LES CONCEPTS INFRASTRUCTURE AS A SERVICE

LA BASE

- Infrastructure :
  - Compute
  - Storage
  - Network

RESSOURCES *COMPUTE*

- Instance
- Image
- Flavor (gabarit)
- Paire de clé (SSH)

L'INSTANCE

- Éphémère, durée de vie typiquement courte
- Dédiée au compute
- Ne doit pas stocker de données persistantes
- Disque racine non persistant

IMAGE CLOUD

- Image disque contenant un OS déjà installé
- Instanciable à l'infini
- Sachant parler à l'API de metadata

API ... DE METADATA

- http://169.254.169.254
- Accessible depuis l'instance
- Fournit des informations relatives à l'instance
- cloud-init permet d'exploiter cette API

FLAVOR (GABARIT)

- *Instance type* chez AWS
- Définit un modèle d'instance en termes de CPU, RAM, disque (racine), disque éphémère
- Le disque éphémère a, comme le disque racine, l'avantage d'être souvent local donc rapide

PAIRE DE CLÉ

- Clé publique + clé privée SSH
- Le cloud manipule et stocke la clé publique
- Cette clé publique est utilisée pour donner un accès SSH aux instances

RESSOURCES RÉSEAU 1/2

- Réseau L2
  - Port réseau
- Réseau L3
  - Routeur
  - IP flottante
  - Groupe de sécurité

RESSOURCES RÉSEAU 2/2

- Load Balancing as a Service
- VPN as a Service
- Firewall as a Service

RESSOURCES STOCKAGE

Le cloud fournit deux types de stockage

- Block
- Objet

STOCKAGE BLOCK

- **Volumes** attachables à une instance
- Accès à des raw devices type */dev/vdb*
- Possibilité d'utiliser n'importe quel système de fichiers
- Possibilité d'utiliser du LVM, du chiffrement, etc.
- Compatible avec toutes les applications existantes

"BOOT FROM VOLUME"

Démarrer une instance avec un disque racine sur un **volume**

- Persistance des données du disque racine
- Se rapproche du serveur classique

STOCKAGE OBJET

- Pousser et retirer des objets dans un container/bucket
- Pas de hiérarchie des données, pas de système de fichiers
- Accès par les APIs
- L’application doit être conçue pour tirer parti du stockage objet

ORCHESTRATION

- Orchestrer la création et la gestion des ressources dans le cloud
- Définition de l'architecture dans un *template*
- Les ressources créées à partir du *template* forment la *stack*

BONNE PRATIQUES D'UTILISATION

HAUTE DISPONIBILITÉ (HA)

- Le control plane (les APIs) du cloud est HA
- Les ressources provisionnées ne le sont pas forcément

PETS VS CATTLE

Comment considérer ses instances ?

- Pet
- Cattle

INFRASTRUCTURE AS CODE

Avec du code

- Provisionner les ressources d'infrastructure
- Configurer les dites ressources, notamment les instances

Le métier évolue : Infrastructure Developer

SCALING, PASSAGE À L'ÉCHELLE

- Scale out plutôt que Scale up
  - Scale out : passage à l'échelle horizontal
  - Scale up : passage à l'échelle vertical
- Auto-scaling

APPLICATIONS CLOUD READY

- Stockent leurs données au bon endroit
- Sont architecturées pour tolérer les pannes
- Etc.

Cf. http://12factor.net/

DERRIÈRE LE CLOUD

COMMENT IMPLÉMENTER UN SERVICE DE COMPUTE

- Virtualisation
- Containers
- Bare metal

IMPLÉMENTATION DU STOCKAGE : (SOFTWARE DEFINED STORAGE) SDS

- **Attention** : ne pas confondre avec le sujet block vs objet
- Utilisation de commodity hardware
- Pas de RAID matériel

- Le logiciel est responsable de garantir les données
- Les pannes matérielles sont prises en compte et gérées
- Le projet **Ceph** et le composant **OpenStack Swift** implémentent du SDS

- Voir aussi **Scality**

SDS - THÉORÈME CAP

Consistency - Availability - Partition tolerance

TOUR D'HORIZON

VUE HAUT NIVEAU

Version simple

HISTORIQUE

- Démarrage en 2010
- Objectif : le Cloud Operating System libre
- Fusion de deux projets de Rackspace (Storage) et de la NASA (Compute)
- Logiciel libre distribué sous licence Apache 2.0
- Naissance de la Fondation en 2012

LES RELEASES

- Austin (2010.1)
- Bexar (2011.1), Cactus (2011.2), Diablo (2011.3)
- Essex (2012.1), Folsom (2012.2)
- Grizzly (2013.1), Havana (2013.2)
- Icehouse (2014.1), Juno (2014.2)
- Kilo (2015.1), Liberty (2015.2)
- Mitaka (2016.1), **Newton** (2016.2)
- Début 2017 : Ocata

STATISTIQUES : KILO

- 1492 contributeurs (Liberty : 1933)
- 169 organisations
- 394 nouvelles fonctionnalités et 7257 bugs corrigés
- 113 drivers/plugins
- 792200 chaines traduites

Source :
http://lists.openstack.org/pipermail/foundatio

board/2015-April/000050.html

QUELQUES SOUTIENS/CONTRIBUTEURS ...

- Editeurs : Red Hat, HP, IBM, Suse, Canonical, Vmware, ...
- Constructeurs : Dell, Hitachi, Juniper, Cisco, NetApp, ...
- En vrac : NASA, Yahoo, OVH, Citrix, SAP, Rackspace, ...
- **Google** ! (depuis juillet 2015)

http://www.openstack.org/foundation/compa

... ET UTILISATEURS

- Tous les contributeurs précédemment cités
- En France : **Cloudwatt** et **Numergy**
- Wikimedia
- CERN
- Paypal
- Comcast
- BMW
- Etc. Sans compter les implémentations confidentielles

http://www.openstack.org/user-stories/

LES DIFFÉRENTS SOUS-PROJETS

- OpenStack Compute - Nova
- OpenStack (Object) Storage - Swift
- OpenStack Block Storage - Cinder
- OpenStack Networking - Neutron
- OpenStack Image Service - Glance
- OpenStack Identity Service - Keystone
- OpenStack Dashboard - Horizon
- OpenStack Telemetry - Ceilometer
- OpenStack Orchestration - Heat
- OpenStack Database Service - Trove

LES DIFFÉRENTS SOUS-PROJETS (2)

- Mais aussi :
  - Bare metal (Ironic)
  - Queue service (Zaqar)
  - Data processing (Sahara)
  - DNS service (Designate)
  - Shared File Systems (Manila)
  - Key management (Barbican)
  - PaaS (Solum)
  - Container (Magnum)
- Autres
  - Les clients (python-*client)

LES 4 OPENS

- Open Source
- Open Design
- Open Development
- Open Community

LA FONDATION OPENSTACK

- Entité de gouvernance principale du projet
- Représentation juridique du projet
- Les membres du board sont issus des entreprises sponsors et élus par les membres individuels
- Tout le monde peut devenir membre individuel (gratuitement)

LA FONDATION OPENSTACK

- La fondation supporte le projet par différents moyens :
  - Événements : organisation (Summits) ou participation (OSCON, etc.)
  - Infrastructure de développement (serveurs)
  - Ressources humaines : marketing, release manager, quelques développeurs (principalement sur l'infrastructure)
- 500 organisations à travers le monde
- 23000 membres individuels dans 160 pays

LA FONDATION OPENSTACK

Les principales entités de la fondation

INTERFACE WEB / DASHBOARD : HORIZON

Screenshot Horizon (Liberty)

RESSOURCES

- Annonces/sécurité : openstack-announce@lists.openstack.org
- Documentation : http://docs.openstack.org/
- Gouvernance du projet : http://governance.openstack.org/
- Support :
  - https://ask.openstack.org
  - openstack@lists.openstack.org
  - #openstack@Freenode
- SDK/APIs : http://developer.openstack.org/

RESSOURCES

- Applications : http://apps.openstack.org/
- Actualités :
  - Blog officiel : http://www.openstack.org/blog/
  - Planet : http://planet.openstack.org
  - Superuser : http://superuser.openstack.org/
  - OpenStack Community Weekly Newsletter

- Newsletter
- Ressources commerciales : http://www.openstack.org/marketplace/ entre autres

RESSOURCES - COMMUNAUTÉ FRANCOPHONE

Logo OpenStack-fr

- Site web : http://openstack.fr/
- Association des utilisateurs francophones d'OpenStack : https://asso.openstack.fr/
- Meetups : Paris, Rhônes-Alpes, Toulouse, Montréal, etc.
- Présence à des événements tels que *Paris Open Source Summit*
- Canaux de communication :
  - openstack-fr@lists.openstack.org
  - #openstack-fr@Freenode

FONCTIONNEMENT INTERNE ET MODE DE DÉVELOPPEMENT

ARCHITECTURE

Vue détaillée des services

IMPLÉMENTATION

- Chaque sous-projet est découpé en plusieurs services
- Communication entre les services : AMQP (RabbitMQ)
- Base de données : relationnelle SQL (MySQL/MariaDB)
- Réseau : OpenVSwitch
- En général : réutilisation de composants existants

existants
- Tout est développé en Python (Django pour la partie web)
- APIs supportées : OpenStack et équivalent AWS
- Multi tenancy

DÉVELOPPEMENT DU PROJET : EN DÉTAILS

- Ouvert à tous (individuels et entreprises)
- Cycle de développement de 6 mois débuté par un (design) summit
- Développement hyper actif : 25000 commits dans Liberty
- Sur chaque patch proposé :
  - Revue de code (peer review) : Gerrit
  - Intégration continue (continous integration) : Jenkins, Zuul, etc.
- Outils : Launchpad → Storyboard (blueprints, bugs) + Git + GitHub (code)

DÉVELOPPEMENT DU PROJET : EN DÉTAILS

Workflow de changements dans OpenStack

CYCLE DE DÉVELOPPEMENT : 6 MOIS

- Le planning est publié, exemple : https://releases.openstack.org/ocata/sched
- Milestone releases
- Freezes : FeatureProposal, Feature, String
- RC releases
- Stable releases
- Ce modèle de cycle de développement a évo

ce modèle de cycle de développement a évo
depuis le début du projet
- Cas particulier de Swift et de plus en plus de composants
- Depuis Liberty, chaque composant gère son versionnement

VERSIONNEMENT DEPUIS LIBERTY

- *Semantic versioning*
- Chaque projet est indépendant
- Dans le cadre du cycle de release néanmoins
- http://releases.openstack.org/

LE NOUVEAU MODÈLE “BIG TENT”

- Évolutions récemment implémentées
- Objectif : résoudre les limitations du précéde incubation/intégré
- Inclusion a priori de l’ensemble de l’écosystè
- *Programs → Project Teams*
  http://governance.openstack.org/reference/
- Utilisation de tags factuels et objectifs
  https://www.openstack.org/software/projec

QUI CONTRIBUE ?

- *Active Technical Contributor*
- ATC : personne ayant au moins une contribution récente dans un projet OpenStack reconnu
- Les ATC sont invités aux summits et ont le droit de vote
- *Core reviewer* : ATC ayant les droits pour valider les patchs dans un projet
- *Project Team Lead* (PTL) : élu par les ATCs

- *Project Team Lead* (PTL) : élu par les ATCs de chaque projet
- Stackalytics fournit des statistiques sur les contributions

http://stackalytics.com/

OÙ TROUVER DES INFORMATIONS SUR LE DÉVELOPPEMENT D'OPENSTACK

- Principalement sur le wiki
  - https://wiki.openstack.org
- Les blueprints et bugs sur Launchpad/StoryBoard
  - https://launchpad.net/openstack
  - https://storyboard.openstack.org
  - http://specs.openstack.org/
- Comment contribuer
  - http://docs.openstack.org/contributor-guide/

OÙ TROUVER DES INFORMATIONS SUR LE DÉVELOPPEMENT D'OPENSTACK

- Les patchs proposés et leurs reviews sont sur Gerrit
  - https://review.openstack.org
- L'état de la CI (entre autres)
  - http://status.openstack.org
- Le code (Git) et les tarballs sont disponibles
  - https://git.openstack.org
  - http://tarballs.openstack.org/

OPENSTACK INFRA

- Équipe projet en charge de l'infrastructure de développement d'OpenStack
- Travaille comme les équipes de dev d'OpenStack et utilise les mêmes outils
- Résultat : une infrastructure entièrement open source et développée comme tel

OPENSTACK SUMMIT

- Aux USA jusqu'en 2013
- Aujourd'hui : alternance Amérique de Nord et Asie/Europe
- Quelques dizaines au début à 6000 participants aujourd'hui
- En parallèle : conférence (utilisateurs, décideurs) et Design Summit (développeurs)
- Détermine le nom de la release : lieu/ville à proximité du Summit
- *Upstream Training*

EXEMPLE DU SUMMIT D'AVRIL 2013 À PORTLAND

Photo : Adrien Cunin

EXEMPLE DU SUMMIT D'OCTOBRE 2015 À TOKYO

Photo : Elizabeth K. Joseph, CC BY 2.0, Flickr/pleia2

EXEMPLE DU SUMMIT D'OCTOBRE 2015 À TOKYO

Photo : Elizabeth K. Joseph, CC BY 2.0, Flickr/pleia2

EXEMPLE DU SUMMIT D'OCTOBRE 2015 À TOKYO

Photo : Elizabeth K. Joseph, CC BY 2.0, Flickr/pleia2

EXEMPLE DU SUMMIT D'OCTOBRE 2015 À TOKYO

Photo : Elizabeth K. Joseph, CC BY 2.0, Flickr/pleia2

TRADUCTION

- La question de la traduction est dorénavant prise en compte (officialisation de l'équipe *i18n*)
- Seules certaines parties sont traduites, comme Horizon
- La traduction française est aujourd'hui une des plus avancées
- Utilisation d'une plateforme web basée sur Zanata : https://translate.openstack.org/

DEVSTACK : FAIRE TOURNER RAPIDEMENT UN OPENSTACK

UTILITÉ DE DEVSTACK

- Déployer rapidement un OpenStack
- Utilisé par les développeurs pour tester leurs changements
- Utilisé pour faire des démonstrations
- Utilisé pour tester les APIs sans se préoccuper du déploiement
- Ne doit PAS être utilisé pour de la production

FONCTIONNEMENT DE DEVSTACK

- Un script shell qui fait tout le travail : stack.sh
- Un fichier de configuration : local.conf
- Installe toutes les dépendances nécessaires (paquets)
- Clone les dépôts git (branche master par défaut)
- Lance tous les composants dans un screen

CONFIGURATION : LOCAL.CONF

Exemple

```
[[local|localrc]]
ADMIN_PASSWORD=secrete
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=a682f596-76f3-11e3-b3b2-e716f9080d50
#FIXED_RANGE=172.31.1.0/24
#FLOATING_RANGE=192.168.20.0/25
#HOST_IP=10.3.4.5
```

CONSEILS D'UTILISATION

- DevStack installe beaucoup de choses sur la machine
- Il est recommandé de travailler dans une VM
- Pour tester tous les composants OpenStack dans de bonnes conditions, plusieurs Go de RAM sont nécessaires
- L'utilisation de Vagrant est conseillée

UTILISER OPENSTACK

LE PRINCIPE

- Toutes les fonctionnalités sont accessibles par l'API
- Les clients (y compris Horizon) utilisent l'API
- Des crédentials sont nécessaires
  - API OpenStack : utilisateur + mot de passe + tenant (+ domaine)
  - API AWS : access key ID + secret access key

LES APIS OPENSTACK

- Une API par service OpenStack
- http://developer.openstack.org/api-ref.html
- Chaque API est versionnée, la rétro-compatibilité est assurée
- REST
- Certains services sont aussi accessibles via une API différente compatible AWS

AUTHENTIFICATION ET CATALOGUE DE SERVICE

- Une fois authentifié, récupération d'un jeton (*token*)
- Récupération du catalogue de services
- Pour chaque service, un endpoint HTTP (API)

ACCÈS AUX APIS

- Direct, en HTTP, via des outils comme curl
- Avec une bibliothèque
  - Les implémentations officielles en Python
  - OpenStackSDK
  - D'autres implémentations, y compris pour d'autres langages (exemple : jclouds)
  - Shade

- Shade
- Avec les outils officiels en ligne de commande
- Avec Horizon

CLIENTS OFFICIELS

- Le projet fournit des clients officiels : python-PROJETclient
- Bibliothèques Python
- Outils CLI
  - L'authentification se fait en passant les credentials par paramètres ou variables d'environnement
  - L'option `--debug` affiche la communication HTTP

OPENSTACK CLIENT

- Client CLI unifié
- Commandes du type *openstack <ressource ><action >*
- Vise à remplacer à terme les clients spécifiques
- Permet une expérience utilisateur plus homogène
- Fichier de configuration clouds.yaml

CE QU'ON VA VOIR

- Installer OpenStack à la main http://docs.openstack.org/newton/install-guide-ubuntu/
- Donc comprendre son fonctionnement
- Passer en revue chaque composant plus en détails
- Tour d'horizon des solutions de déploiement

ARCHITECTURE DÉTAILLÉE

Vue détaillée des services

ARCHITECTURE SERVICES

Machines "physiques" et services

QUELQUES ÉLÉMENTS DE CONFIGURATION GÉNÉRAUX

- Tous les composants doivent être configurés pour communiquer avec Keystone
- La plupart doivent être configurés pour communiquer avec MySQL/MariaDB et RabbitMQ
- Les composants découpés en plusieurs

Les composants découpés en plusieurs services ont parfois un fichier de configuration par service

- Le fichier de configuration `policy.json` précise les droits nécessaires pour chaque appel API

SYSTÈME D'EXPLOITATION

- OS Linux avec Python
- Historiquement : Ubuntu
- Red Hat s'est largement rattrapé
- SUSE, Debian, CentOS, etc.

PYTHON

Logo Python

- OpenStack est aujourd'hui compatible Python 2.7
- Afin de ne pas réinventer la roue, beaucoup de dépendances sont nécessaires
- Un travail de portage vers Python 3 est en cours

BASE DE DONNÉES

- Permet de stocker la majorité des données gérées par OpenStack
- Chaque composant a sa propre base
- OpenStack utilise l'ORM Python SQLAlchemy
- Support théorique équivalent à celui de SQLAlchemy
- MySQL/MariaDB est l'implémentation la plus largement testée et utilisée

plus largement testée et utilisée
- SQLite est principalement utilisé dans le cadre de tests et démo
- Certains déploiements fonctionnent avec PostgreSQL

POURQUOI L'UTILISATION DE SQLALCHEMY

Logo SQLAlchemy

- Support de multiples BDD
- Gestion des migrations

Logo MySQL

PASSAGE DE MESSAGES

- AMQP : Advanced Message Queuing Protocol
- Messages, file d'attente, routage
- Les processus OpenStack communiquent via AMQP
- Plusieurs implémentations possibles : Qpid, 0MQ, etc.
- RabbitMQ par défaut

RABBITMQ

Logo RabbitMQ

- RabbitMQ est implémenté en Erlang
- Une machine virtuelle Erlang est donc nécessaire

“HELLO WORLD” RABBITMQ

Illustration simple du fonctionnement de RabbitMQ

KEYSTONE : AUTHENTIFICATION, AUTORISATION ET CATALOGUE DE SERVICES

PRINCIPES

- Annuaire des utilisateurs et des groupes
- Liste des tenants/projets
- Catalogue de services
- Gère l’authentification et l’autorisation
- Support des domaines dans l’API v3
- Fournit un token à l’utilisateur

API

- API v2 admin : port 35357
- API v2 utilisateur : port 5000
- API v3 unifiée : port 5000
- L'API v2 est dépréciée
- Gère *utilisateurs*, *groupes*, *domaines* (API v3)
- Les utilisateurs ont des *rôles* sur des *tenants* (projets)
- Les *services* du catalogue sont associés à des *endpoints*

SCÉNARIO D'UTILISATION TYPIQUE

Interactions avec Keystone

INSTALLATION ET CONFIGURATION

- Paquet : keystone
- Backends : choix de SQL ou LDAP (ou AD)
- Backends tokens : SQL, Memcache, aucun
- Configuration d’un token *ADMIN* pour la configuration initiale
- Création des services et endpoints
- Création d'utilisateurs, groupes, domaines

ENREGISTRER UN SERVICE ET SON ENDPOINT

Il faut renseigner l'existence des différents services (catalogue) dans Keystone :

```
$ keystone service-create --name=cinderv2 --type=volumev2
  --description="Cinder Volume Service V2"
$ keystone endpoint-create \
  --region=myRegion
  --service-id=...
  --publicurl=http://controller:8776/v2/%\(tenant_id\)s \
  --internalurl=http://controller:8776/v2/%\(tenant_id\)s \
  --adminurl=http://controller:8776/v2/%\(tenant_id\)s
```

TESTER

```
$ keystone service-list
...
$ keystone user-list
...
$ keystone token-get
...
```

NOVA : COMPUTE

PRINCIPES

- Gère les instances
- Les instances sont créées à partir des images fournies par Glance
- Les interfaces réseaux des instances sont associées à des ports Neutron
- Du stockage block peut être fourni aux instances par Cinder

INTERACTIONS AVEC LES AUTRES COMPOSANTS

Instance, image et volume

PROPRIÉTÉS D'UNE INSTANCE

- Éphémère, a priori non hautement disponible
- Définie par une flavor
- Construite à partir d'une image
- Optionnel : attachement de volumes
- Optionnel : boot depuis un volume
- Optionnel : une clé SSH publique
- Optionnel : des ports réseaux

API

Gère :

- Instances
- Flavors (types d'instance)
- Indirectement : images, security groups (groupes de sécurité), floating IPs (IPs flottantes)

Les instances sont redimensionnables et

migrables d’un hôte physique à un autre.

LES GROUPES DE SÉCURITÉ

- Équivalent à un firewall devant chaque instance
- Une instance peut être associée à un ou plusieurs groupes de sécurité
- Gestion des accès en entrée et sortie
- Règles par protocole (TCP/UDP/ICMP) et par port
- Cible une adresse IP, un réseau ou un autre groupe de sécurité

FLAVORS

- *Gabarit*
- Équivalent des “instance types” d’AWS
- Définit un modèle d’instance en termes de CPU, RAM, disque (racine), disque éphémère
- Un disque de taille nul équivaut à prendre la taille de l’image de base
- Le disque éphémère a, comme le disque racine, l’avantage d’être souvent local donc rapide

NOVA API

- Double rôle
- API de manipulation des instances par l'utilisateur
- API à destination des instances : API de metadata
- L'API de metadata doit être accessible à l'adresse http://169.254.169.254/
- L'API de metadata fournit des informations de configuration personnalisées à chacune des instances

NOVA COMPUTE

- Pilote les instances (machines virtuelles ou physiques)
- Tire partie de libvirt ou d'autres APIs comme XenAPI
- Drivers : libvirt (KVM, LXC, etc.), XenAPI, VMWare vSphere, Ironic
- Permet de récupérer les logs de la console et une console VNC

NOVA SCHEDULER

- Service qui distribue les demandes d'instanc
  sur les nœuds compute
- Filter, Chance, Multi Scheduler
- Filtres, par défaut :
  AvailabilityZoneFilter,RamFilter,ComputeFil
- Tri par poids, par défaut : RamWeigher

LE SCHEDULER NOVA EN ACTION

Fonctionnement de nova-scheduler

NOVA CONDUCTOR

- Service facultatif qui améliore la sécurité
- Fait office de proxy entre les nœuds compute et la BDD
- Les nœuds compute, vulnérables, n’ont donc plus d’accès à la BDD

TESTER

```
$ nova list
...
$ nova create
...
```

GLANCE : REGISTRE D'IMAGES

PRINCIPES

- Registre d’images (et des snapshots)
- Propriétés sur les images
- Est utilisé par Nova pour démarrer des instances
- Peut utiliser Swift comme back-end de stockage

PROPRIÉTÉS DES IMAGES DANS GLANCE

L’utilisateur peut définir un certain nombre de propriétés dont certaines seront utilisées lors de l’instanciation

- Type d’image
- Architecture
- Distribution
- Version de la distribution
- Espace disque minimum
- RAM minimum
- Publique ou non

TYPES D'IMAGES

Glance supporte un large éventail de types d'images, limité par le support de l'hyperviseur sous-jacent à Nova

- raw
- qcow2
- ami
- vmdk
- iso

BACKENDS

- Swift ou S3
- Ceph
- HTTP
- Répertoire local

INSTALLATION

- Paquet glance-api : fournit l'API
- Paquet glance-registry : démon du registre d'images en tant que tel

TESTER

```
$ glance image-list
...
$ glance image-create
...
```

NEUTRON : RÉSEAU EN TANT QUE SERVICE

PRINCIPES

- *Software Defined Networking* (SDN)
- Auparavant Quantum et nova-network
- IP flottantes, groupes de sécurité
- neutron-server : fournit l'API
- Agent DHCP : fournit le service de DHCP pour les instances
- Agent L3 : gère la couche 3 du réseau, le routage
- Plugin : OpenVSwitch par défaut, d'autres implémentations libres/propriétaires, logicielles/matérielles existent

FONCTIONNALITÉS SUPPLÉMENTAIRES

Outre les fonctions réseau de base niveaux 2 et 3, Neutron peut fournir d’autres services :

- Load Balancing (HAProxy, ...)
- Firewall (vArmour, ...) : diffère des groupes de sécurité
- VPN (Openswan, ...) : permet d'accéder à un réseau privé sans IP flottantes

Ces fonctionnalités se basent également sur des plugins

API

L’API permet notamment de manipuler ces ressources

- Réseau (*network*) : niveau 2
- Sous-réseau (*subnet*) : niveau 3
- Port : attachable à une interface sur une instance, un load-balancer, etc.
- Routeur

PLUGINS ML2

- Modular Layer 2
- OpenVSwitch
- OpenDaylight
- Contrail, OpenContrail
- Nuage Networks
- VMWare NSX
- cf. OpenFlow

IMPLÉMENTATION

- Neutron tire partie des namespaces réseaux du noyau Linux pour permettre l'IP overlapping
- Le proxy de metadata est un composant qui permet aux instances isolées dans leur réseau de joindre l'API de metadata fournie par Nova

SCHÉMA

Vue utilisateur du réseau

SCHÉMA

Vue infra du réseau

CINDER : STOCKAGE BLOCK

PRINCIPES

- Auparavant nova-volume
- Fournit des volumes (stockage block) attachables aux instances
- Gère différents types de volume
- Gère snapshots et backups de volumes
- Attachement via iSCSI par défaut

DU STOCKAGE PARTAGÉ ?

- Cinder n'est **pas** une solution de stockage partagé comme NFS
- Le projet OpenStack Manila a pour objectif d'être un *NFS as a Service*
- AWS n'a introduit une telle fonctionnalité que récemment

UTILISATION

- Volume supplémentaire (et stockage persistant) sur une instance
- Boot from volume : l'OS est sur le volume
- Fonctionnalité de backup vers un object store (Swift ou Ceph)

INSTALLATION

- Paquet cinder-api : fournit l'API
- Paquet cinder-volume : création et gestion des volumes
- Paquet cinder-scheduler : distribue les demandes de création de volume
- Paquet cinder-backup : backup vers un object store

BACKENDS

- Utilisation de plusieurs backends en parallèle possible
- LVM (par défaut)
- GlusterFS
- Ceph
- Systèmes de stockage propriétaires type NetApp
- DRBD

HORIZON : DASHBOARD WEB

PRINCIPES

- Utilise les APIs existantes pour fournir une interface utilisateur
- Horizon est un module Django
- OpenStack Dashboard est l'implémentation officielle de ce module

Logo du framework web Python Django

CONFIGURATION

- local_settings.py
- Les services apparaissent dans Horizon s'ils sont répertoriés dans le catalogue de services de Keystone

UTILISATION

- Une zone “admin” restreinte
- Une interface par tenant

SWIFT : STOCKAGE OBJET

PRINCIPES

- SDS : *Software Defined Storage*
- Utilisation de commodity hardware
- Théorème CAP : on en choisit deux
- Accès par les APIs
- Architecture totalement acentrée
- Pas de base de données centrale

IMPLÉMENTATION

- Proxy : serveur API par lequel passent toutes les requêtes
- Object server : serveur de stockage
- Container server : maintient des listes d’objects dans des containers
- Account server : maintient des listes de containers dans des accounts
- Chaque objet est répliqué n fois (3 par défaut)

LE RING

- Problème : comment décider quelle donnée va sur quel object server
- Le ring est découpé en partitions
- On situe chaque donnée dans le ring afin de déterminer sa partition
- Une partition est associée à plusieurs serveurs

SCHÉMA

Architecture Swift

CEILOMETER : COLLECTE DE MÉTRIQUES

SURVEILLER L'UTILISATION DE SON INFRASTRUCTURE AVEC CEILOMETER

- Indexe différentes métriques concernant l'utilisation des différents services du cloud
- Fournit des APIs permettant de récupérer ces données
- Base pour construire des outils de facturation (exemple : CloudKitty)
- Utilise MongoDB (par défaut) pour le stockage

GNOCCHI : TIME-SERIES DATABASE

- Pourquoi Gnocchi ? Palier aux problème de scalabilité de Ceilometer
- Initié par des développeurs de Ceilometer et intégré à l'équipe projet Ceilometer
- Back-end remplaçant MongoDB pour Ceilometer

HEAT : ORCHESTRATION DES RESSOURCES

ORCHESTRER SON INFRASTRUCTURE AVEC HEAT

- Équivalent d’Amazon Cloud Formation
- Orchestrer les ressources compute, storage, network, etc.
- Doit se coupler avec cloud-init
- Description de son infrastructure dans un fichier template, format JSON (CFN) ou YAML (HOT)

AUTOSCALING AVEC HEAT

Heat implémente la fonctionnalité d'autoscaling

- Se déclenche en fonction des alarmes produites par Ceilometer
- Entraine la création de nouvelles instances

UN TEMPLATE HOT

parameters - resources - outputs

```
heat_template_version: 2013-05-23
description: Simple template to deploy a single compute instar
resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      key_name: my_key
      image: F18-x86_64-cfntools
      flavor: m1.small
```

FONCTIONNALITÉS AVANCÉES DE HEAT

- Nested stacks
- Environments
- Providers

CONSTRUIRE UN TEMPLATE À PARTIR D'EXISTANT

Multiples projets en cours de développement

- Flame (Cloudwatt)
- HOT builder
- Merlin

TROVE : DATABASE AS A SERVICE

PRINCIPE

- Fournit des bases de données relationnelles, à la AWS RDS
- A vocation à supporter des bases NoSQL aussi
- Gère notamment MySQL/MariaDB comme back-end
- Se repose sur Nova pour les instances dans lesquelles se fait l'installation d'une BDD

SERVICES

- trove-api : API
- trove-taskmanager : gère les instances BDD
- trove-guestagent : agent interne à l’instance

DESIGNATE : DNS AS A SERVICE

PRINCIPE

- Équivalent d’AWS Route 53
- Gère différents backends : BIND, etc.

QUELQUES AUTRES COMPOSANTS INTÉRESSANTS

IRONIC

- Anciennement Nova bare-metal
- Permet le déploiement d’instances sur des machines physiques (plutôt que VMs)
- Repose sur des technologies telles que PXE, TFTP

OSLO, OU OPENSTACK COMMON

- Oslo contient le code commun à plusieurs composants d'OpenStack
- Son utilisation est transparente pour le déployeur

ROOTWRAP

- Wrapper pour l’utilisation de commandes en root
- Configuration au niveau de chaque composant qui l’utilise
- Permet d’écrire des filtres sur les commandes

TRIPLEO

- OpenStack On OpenStack
- Objectif : pouvoir déployer un cloud OpenStack (*overcloud*) à partir d'un cloud OpenStack (*undercloud*)
- Autoscaling du cloud lui-même : déploiement de nouveaux nœuds compute lorsque cela est nécessaire
- Fonctionne conjointement avec Ironic pour le déploiement bare-metal

TEMPEST

- Suite de tests d'un cloud OpenStack
- Effectue des appels à l'API et vérifie le résultat
- Est très utilisé par les développeurs via l'intégration continue
- Le déployeur peut utiliser Tempest pour vérifier la bonne conformité de son cloud
- Cf. aussi Rally

BONNES PRATIQUES POUR UN DÉPLOIEMENT EN PRODUCTION

QUELS COMPOSANTS DOIS-JE INSTALLER ?

- Keystone est indispensable
- L'utilisation de Nova va de paire avec Glance et Neutron
- Cinder s'avérera souvent utile
- Ceilometer et Heat vont souvent ensemble
- Swift est indépendant des autres composants
- Neutron peut parfois être utilisé indépendamment (ex : avec oVirt)

http://docs.openstack.org/arch-design/

PENSER DÈS LE DÉBUT AUX CHOIX STRUCTURANTS

- Distribution et méthode de déploiement
- Hyperviseur
- Réseau : quelle architecture et quels drivers
- Politique de mise à jour

LES DIFFÉRENTES MÉTHODES D'INSTALLATION

- DevStack est à oublier pour la production
- TripleO est très complexe
- Le déploiement à la main comme vu précédemment n’est pas recommandé car peu maintenable
- Distributions OpenStack packagées et prêtes à l’emploi
- Distributions classiques et gestion de configuration
- Déploiement continu

MISES À JOUR ENTRE VERSIONS MAJEURES

- OpenStack supporte les mises à jour N → N+1
- Swift : très bonne gestion en mode *rolling upgrade*
- Autres composants : tester préalablement avec vos données
- Lire les release notes
- Cf. articles de blog du CERN

MISES À JOUR DANS UNE VERSION STABLE

- Fourniture de correctifs de bugs majeurs et de sécurité
- OpenStack intègre ces correctifs sous forme de patchs dans la branche stable
- Publication de *point releases* intégrant ces correctifs issus de la branche stable
- Durée variable du support des versions stables, dépendant de l'intérêt des intégrateurs

ASSIGNER DES RÔLES AUX MACHINES

Beaucoup de documentations font référence à ces rôles :

- Controller node : APIs, BDD, AMQP
- Network node : DHCP, routeur, IPs flottantes
- Compute node : Hyperviseur/pilotage des instances

Ce modèle simplifié n’est pas HA.

HAUTE DISPONIBILITÉ

Haute disponibilité du IaaS

- MySQL/MariaDB, RabbitMQ : HA classique (Galera, Clustering)
- Les services APIs sont stateless et HTTP : scale out et load balancers
- La plupart des autres services OpenStack sont capables de scale out également

Guide HA : http://docs.openstack.org/ha-guide/

HAUTE DISPONIBILITÉ DE L'AGENT L3 DE NEUTRON

- Plusieurs solutions et contournements possibles
- Depuis Juno : *Distributed Virtual Router* (DVR)

CONSIDÉRATIONS POUR UNE ENVIRONNEMENT DE PRODUCTION

- Des URLs uniformes pour toutes les APIs : utiliser un reverse proxy
- Apache/mod_wsgi pour servir les APIs lorsque cela est possible (Keystone)
- Utilisation des quotas
- Prévoir les bonnes volumétries, notamment pour les données Ceilometer
- Monitoring
- Backup
- QoS : en cours d'implémentation dans Neutron

Guide Operations :
http://docs.openstack.org/openstack-ops/content/

UTILISATION DES QUOTAS

- Limiter le nombre de ressources allouables
- Par utilisateur ou par tenant
- Support dans Nova
- Support dans Cinder
- Support dans Neutron

http://docs.openstack.org/user-guide-admin/content/cli_set_quotas.html

DÉCOUPAGE RÉSEAU

- Management network : réseau d'administration
- Data network : réseau pour la communication inter instances
- External network : réseau externe, dans l'infrastructure réseau existante
- API network : réseau contenant les endpoints API

CONSIDÉRATIONS LIÉES À LA SÉCURITÉ

- Indispensable : HTTPS sur l'accès des APIs à l'extérieur
- Sécurisation des communications MySQL/MariaDB et RabbitMQ
- Un accès MySQL/MariaDB par base et par service
- Un utilisateur Keystone par service

- Un utilisateur Keystone par service
- Limiter l'accès en lecture des fichiers de configuration (mots de passe, token)
- Veille sur les failles de sécurité : OSSA (*OpenStack Security Advisory*), OSSN (*... Notes*)

Guide sécurité :
http://docs.openstack.org/security-guide/

SEGMENTER SON CLOUD

- Host aggregates : machines physiques avec des caractéristiques similaires
- Availability zones : machines dépendantes d'une même source électrique, d'un même switch, d'un même DC, etc.
- Regions : chaque région a son API
- Cells : permet de regrouper plusieurs cloud différents sous une même API

http://docs.openstack.org/openstack-ops/content/scaling.html#segregate_cloud

HOST AGGREGATES / AGRÉGATS D'HÔTES

- Spécifique Nova
- L’administrateur définit des agrégats d’hôtes via l’API
- L’administrateur associe flavors et agrégats via des couples clé/valeur communs
- 1 agrégat ≡ 1 point commun, ex : GPU
- L’utilisateur choisit un agrégat à travers son choix de flavor à la création d’instance

AVAILABILITY ZONES / ZONES DE DISPONIBILITÉ

- Spécifique Nova et Cinder
- Groupes d’hôtes
- Découpage en termes de disponibilité : Rack, Datacenter, etc.
- L’utilisateur choisit une zone de disponibilité à la création d’instance
- L’utilisateur peut demander à ce que des instances soient démarrées dans une même zone, ou au contraire dans des zones différentes

RÉGIONS

- Générique OpenStack
- Équivalent des régions d'AWS
- Un service peut avoir différents endpoints dans différentes régions
- Chaque région est autonome
- Cas d'usage : cloud de grande ampleur (comme certains clouds publics)

CELLS / CELLULES

- Spécifique Nova
- Un seul nova-api devant plusieurs cellules
- Chaque cellule a sa propre BDD et file de messages
- Ajoute un niveau de scheduling (choix de la cellule)

PACKAGING D'OPENSTACK - UBUNTU

- Le packaging est fait dans de multiples distributions, RPM, DEB et autres
- Ubuntu est historiquement la plateforme de référence pour le développement d'OpenStack
- Le packaging dans Ubuntu suit de près le développement d'OpenStack, et des tests automatisés sont réalisés
- Canonical fournit la Ubuntu Cloud Archive, qui met à disposition la dernière version d'OpenStack pour la dernière Ubuntu LTS

UBUNTU CLOUD ARCHIVE (UCA)

Support d'OpenStack dans Ubuntu via l'UCA

PACKAGING D'OPENSTACK DANS LES AUTRES DISTRIBUTIONS

- OpenStack est intégré dans les dépôts officiels de Debian
- Red Hat propose RHOS/RDO (déploiement basé sur TripleO)
- Comme Ubuntu, le cycle de release de Fedora est synchronisé avec celui d’OpenStack

LES DISTRIBUTIONS OPENSTACK

- StackOps
- Mirantis
- HP Helion
- etc.

DÉPLOIEMENT BARE-METAL

- Le déploiement des hôtes physiques OpenStack peut se faire à l'aide d'outils dédiés
- MaaS (Metal as a Service), par Ubuntu/Canonical : se couple avec Juju
- Crowbar / OpenCrowbar (initialement Dell) : utilise Chef
- eDeploy (eNovance) : déploiement par des images
- Ironic via TripleO

GESTION DE CONFIGURATION

- Puppet, Chef, CFEngine, Saltstack, Ansible, etc.
- Ces outils peuvent aider à déployer le cloud OpenStack
- ... mais aussi à gérer les instances (section suivante)

MODULES PUPPET, PLAYBOOKS ANSIBLE

- *Puppet OpenStack* et *OpenStack Ansible* : modules Puppet et playbooks Ansible
- Développés au sein du projet OpenStack
- https://wiki.openstack.org/wiki/Puppet
- http://docs.openstack.org/developer/opens ansible/install-guide/

DÉPLOIEMENT CONTINU

- OpenStack maintient un master (trunk) toujours stable
- Possibilité de déployer au jour le jour le master (CD : *Continous Delivery*)
- Nécessite la mise en place d'une infrastructure importante
- Facilite les mises à jour entre versions majeures

GÉRER LES PROBLÈMES

PROBLÈMES : RESSOURCES FAILED/ERROR

- Plusieurs causes possibles
- Possibilité de supprimer la ressource ?
- L'appel API *reset-state* peut servir

LES RÉFLEXES EN CAS D'ERREUR OU DE COMPORTEMENT ERRONÉ

- Travaille-t-on sur le bon tenant ?
- Est-ce que l'API renvoie une erreur ? (le dashboard peut cacher certaines informations)
- Si nécessaire d'aller plus loin :
  - Regarder les logs sur le cloud controller (/var/log/<composant>/*.log)

(/var/log/<composant>/*.log)

- Regarder les logs sur le compute node et le network node si le problème est spécifique réseau/instance
- Éventuellement modifier la verbosité des logs dans la configuration

EST-CE UN BUG ?

- Si le client CLI crash, c’est un bug
- Si le dashboard web ou une API renvoie une erreur 500, c’est peut-être un bug
- Si les logs montrent une stacktrace Python, c’est un bug
- Sinon, à vous d’en juger

DEUX VISIONS

Une fois un cloud IaaS en place, deux optiques possibles :

- Garder les mêmes pratiques tout en profitant du self service et de l'agilité de la solution pour des besoins test/dev
- Faire évoluer ses pratiques, tant côté applicatif que système "Pets vs Cattle"

SINON ?

Faire tourner des applications *legacy* dans le cloud est une mauvaise idée :

- Interruptions de service
- Pertes de données
- Incompréhensions “le cloud ça marche pas”

CÔTÉ APPLICATIONS

ADAPTER OU PENSER SES APPLICATIONS “CLOUD READY” 1/3

Cf. les design tenets du projet OpenStack et Twelve-Factor http://12factor.net/

- Architecture distribuée plutôt que monolithique
  - Facilite le passage à l'échelle
  - Limite les domaines de *failure*
- Couplage faible entre les composants

ADAPTER OU PENSER SES APPLICATIONS “CLOUD READY” 2/3

- Bus de messages pour les communications inter-composants
- Stateless : permet de multiplier les routes d’accès à l’application
- Dynamicité : l’application doit s’adapter à son environnement et se reconfigurer lorsque nécessaire
- Permettre le déploiement et l’exploitation par des outils d’automatisation

ADAPTER OU PENSER SES APPLICATIONS “CLOUD READY” 3/3

- Limiter autant que possible les dépendances à du matériel ou du logiciel spécifique qui pourrait ne pas fonctionner dans un cloud
- Tolérance aux pannes (*fault tolerance*)

Tolérance aux pannes (*fault tolerance*) intégrée

- Ne pas stocker les données en local, mais plutôt :
  - Base de données
  - Stockage objet
- Utiliser des outils standards de journalisation

CÔTÉ SYSTÈME

ADOPTER UNE PHILOSOPHIE DEVOPS

- Infrastructure as Code
- Scale out plutôt que scale up (horizontalement plutôt que verticalement)
- HA niveau application plutôt qu'infrastructure
- Automatisation, automatisation, automatisation
- Tests

MONITORING

- Prendre en compte le cycle de vie des instances
- Monitorer le service plus que le serveur

BACKUP

- Être capable de recréer ses instances (et le reste de son infrastructure)
- Données (applicatives, logs) : block, objet

UTILISER DES IMAGES CLOUD

Une image cloud c’est :

- Une image disque contenant un OS déjà installé
- Une image qui peut être instanciée en n machines sans erreur
- Un OS sachant parler à l'API de metadata du cloud (cloud-init)
- Détails : http://docs.openstack.org/image-guide/openstack-images.html
- La plupart des distributions fournissent aujourd'hui des images cloud.

CIRROS

- Cirros est l'image cloud par excellence
- OS minimaliste
- Contient cloud-init

https://launchpad.net/cirros

CLOUD-INIT

- Cloud-init est un moyen de tirer parti de l'API de metadata, et notamment des user data
- L'outil est intégré par défaut dans la plupart des images cloud
- À partir des user data, cloud-init effectue les opérations de personnalisation de l'instance
- cloud-config est un format possible de user data

EXEMPLE CLOUD-CONFIG

```
#cloud-config
mounts:
 - [ xvdc, /var/www ]
packages:
 - apache2
 - htop
```

COMMENT GÉRER SES IMAGES ?

- Utilisation d'images génériques et personnalisation à l'instanciation
- Création d'images intermédiaires et/ou totalement personnalisées : *Golden images*
  - libguestfs, virt-builder, virt-sysprep
  - diskimage-builder (TripleO)
  - Packer
  - solution "maison"

CONFIGURER ET ORCHESTRER SES INSTANCES

- Outils de gestion de configuration (les mêmes qui permettent de déployer OpenStack)
- Juju

POUR CONCLURE

- Le cloud révolutionne l'IT
- OpenStack est le projet libre phare sur la partie IaaS
- Déployer OpenStack n'est pas une mince affaire
- L'utilisation d'un cloud IaaS implique des changements de pratique
- Les métiers d'architecture logicielle et infra évoluent