Générer trois mot de passe :

**#Â openssl rand -base64 32**

1er: client password pour connection au middleware ave les permissions d'utiliser des commandes sur le servers hosts. Devrait Ãªtre attribuÃ© au client de l'utilisateur dans le fichier de /etc/activemq/activemq.xml et utilisÃ© pour plugin.activemq.pool.1.password dans /etc/mcollective/client.cfg

2eme: mot de passe server utilise par les serveurs pour ce connecter au broker avec des autorisations d'abonnement aux chaines de commandement.Devrait Ãªtre attribuÃ© au serveur de l'utilisateur dans le fichier de /etc/activemq/activemq.xml et utilisÃ© pour plugin.activemq.pool.1.password dans /etc/mcollective/server.cfg

3eme: la key Pre-partagÃ© utilisÃ© comme un sel dans le hachage cryptographique utilisÃ© pour valider les communications entre le serveur et le client, se assurer que personne ne peut modifier la demande de charge utile en transit.Devrait Ãªtre utilisÃ© comme valeur pour plugin.psk Ã  la fois /etc/mcollective/server.cfg et /etc/mcollective/client.cfg

installer les dépots sur Enterprise Linux 6:

yum install http://yum.puppetlabs.com/puppetlabs-release-el-6.noarch.rpm

Rabbitmq:

La premiÃ¨re Ã©tape est d'installer le middleware utilisÃ© pour la communication entre clients et serveurs. Vous pouvez l'installer sur un serveur existant Puppet. Sauf si vous avez des centaines de nÅ“uds.

yum install activemq

chkconfig activemq on

après le démarrage du service, vérifiez que le broker est Ã  l'écoute sur le port TCP 61613:

netstat -an | grep 61613

Si vous ne voyez pas un socket d'écoute disponibles pour les connexions entrantes, consultez le fichier journal

tail -f /var/log/xxxx/xxx.log

Firewall:

Vous devez vous assurer que les sessions TCP entrants vers le port 61613 peuvent Ãªtre crÃ©Ã©s Ã  partir de chaque serveur et le client mcollective.

# iptables --list --line-numbers

# iptables -I INPUT 20 -m state --state NEW -p tcp --source 192.168.0.0/24 --dport 61613 -j ACCEPTE

# service iptables save

# ip6tables -I INPUT 20 -m state --state NEW -p tcp --source 2001:DB8:6A:C0::/24 --dport 61613 -j ACCEPT

# service ip6tables save

Install MCollective:

#yum install mcollective

#chkconfig mcollective on

le fichier /etc/mcollective/server.cfg:

Le texte suivant est le fichier de configuration du serveur mcollective, qui doit Ãªtre installÃ© sur chaque hÃ´te que vous souhaitez contrÃ´ler. Notez que vous devez remplacer deux des mots de passe dans ce fichier et aussi le rÃ©pertoire de libdir.

XXXXXXXXXXXXXXXXx

\*Mcollective utilise toujours le protocole STOMP lors de la connexion avec les broker, mais cela ne figure pas dans ca configuration.Dans la configuration du broker, vous ne mentionnez pas mcollective mais dites au transportConnector de fournir STOMP transport protocole.Lorsque vous faites une recherche sur Internet, vous pouvez trouver des rÃ©fÃ©rences Ã  un connecteur STOMP. Ce connecteur a Ã©tÃ© dÃ©conseillÃ©e en mcollective 2.2.3 et retirÃ©.Toujours utiliser les ActiveMQ et RabbitMQ connecteurs natifs

le fichier /etc/mcollective/client.cfg.

le fichier de configuration du client, qui doit Ãªtre installÃ© uniquement sur l'hÃ´tes Ã  partir duquel vous pourrez contrloer votre parc. Avec le modÃ¨le prÃ©-partagÃ©e clÃ© de sÃ©curitÃ©, n'importe qui peut lire le fichier client.cfg peut trouver le mot de passe utilisÃ© pour publier des demandes. Je vous recommande de limiter les personnes qui peuvent lire le fichier client

#yum install mcollective-client

#chmod 640 /etc/mcollective/client.cfg

#chown root:wheel /etc/mcollective/client.cfg

demarrer le service:

# service mcollective start

Test de l'installation:

A ce moment, vous devriez voir le serveur liÃ© au serveur broker sur le port rÃ©pertoriÃ© dans les deux fichiers server.cfg (et activemq.xml):

Testez la connectivitÃ© de serveur en allant sur le systÃ¨me de middleware et confirmez que vous voyez les connexions au port 61613 de chacun des serveurs:

#netstat -an | grep 61613

#netstat -an -A inet6 | grep 61613

[root@pc-phobos opt]# netstat -an | grep 61613

tcp 0 0 ::1:61613 :::\* LISTEN

tcp 0 0 ::1:51751 ::1:61613 ESTABLISHED

tcp 0 0 ::1:61613 ::1:51751 ESTABLISHED

Si vous ne voyez pas les connexions de ce genre, alors il y a un pare-feu qui empêche les serveurs d'accédé au courtier de middleware.

Après avoir mis en place un hôte de middleware, au moins un serveur et un client, vous pouvez effectuer un test pour confirmer que vos paramètres de configuration sont corrects.

Notez que l'hôte à la fois le serveur et le logiciel client installé. Il recevra les demandes par l'intermédiaire du middleware le même que tous les autres serveurs.

# LE PING :

Le test de ping est une requête de bas niveau qui confirme que le noeud du serveur communique via le middleware: vous obtenez une liste de chaque node connecté à  votre middleware et son temps de réponse,

[11:38:33] root@col-rec-appv1:/etc/mcollective # mco ping

for-rec-appv1 time=11.19 ms

col-rec-appv1.adm.parimutuel.local time=12.90 ms

u05-adm-vm06.ass.parimutuel.local time=14.58 ms

u05-bw-vm12.ass.parimutuel.local time=15.87 ms

stge-lunea1.adm.parimutuel.local time=16.62 ms

stge-test.adm.parimutuel.local time=18.97 ms

osi-dev-appv2.dev.parimutuel.local time=19.64 ms

ams-jbs-vm48.dev.parimutuel.local time=20.94 ms

u05-ems-vm14.ass.parimutuel.local time=21.59 ms

u05-ems-vm15.ass.parimutuel.local time=22.39 ms

osi-dev-appv1.dev.parimutuel.local time=23.21 ms

Line de commande client:

La façon la plus courante d'interagir avec MCollective est le client de ligne de commande MCO, qui peut être utilisé de manière interactive ou dans des scripts.

Il est aussi relativement facile d’écrire d'autres clients en Ruby, qui peuvent être utilisés comme backends d’application.

Le fichier de configuration globale pour un client MCollective sera stocké dans le répertoire d'installation, **/etc/mcollective/client.cfg**.

Les utilisateurs peuvent créer leurs propres fichiers de configuration.

Le nom de fichier par défaut est **.mcollective** dans le répertoire personnel de l'utilisateur. Des fichiers de configuration alternatifs peuvent être spécifiés avec **"-c configfile"** sur la ligne de commande. On utilise des fichiers spécifiques à un utilisateur lors de l'utilisation de clés SSL pour l'authentification. Si vous spécifiez un fichier de configuration, le fichier de configuration global est ignoré.

# DEAMON MCOLLECTIVED :

Le logiciel installé sur les noeuds sont contrôlez avec un daemon appelé **mcollectived** configurer à l’aide du fichier **/etc/mcollective/server.conf**.

Chaque agent a un client de correspondance ou une application qui sait comment émettre des demandes spécifiques à cet agent.

Sur chaque noeud nous avons installé le service **mcollectived**. Pour que le daemon fonctionne correctement, il nécessite deux plugins:

-**connector plugin**: (connector = {activemq/rabbitmq})Permettant d’établir un lien avec le middleware et de s'enregistrer et publier dans des topics.

-**security plugin**: (securityprovider = psk) Un plugin de sécurité pour encrypter et décrypter les communications

Ces deux connecteurs doivent être les mêmes dans votre environnement.

# LES FACTERS:

La façon la plus complète pour identifier les groupes connexes de systèmes est par des Facts, qui sont des paires clée/valeur avec des informations sur votre node. La façon la plus courante pour obtenir des Facts est d’utiliser le programme de facter de PuppetLabs.

**# yum install facter**

[12:00:18] root@col-rec-appv1:/etc/mcollective # facter

architecture => x86\_64

augeasversion => 1.0.0

bios\_release\_date => 07/30/2013

bios\_vendor => Phoenix Technologies LTD

bios\_version => 6.00

...

C’est une bonne idée de remplir le fichier de "**facts.yaml**" avec quelques facts à utiliser. D'abord, éditez le fichier **/etc/mcollective/server.cfg** pour qu’il contienne les éléments suivants (La cible pour le paramétré **plugin.yaml** peut inclure plusieurs noms de fichiers séparés par deux points):

**# Facts**

**factsource = Yaml**

**plugin.yaml = /etc/mcollective/facts.yaml**

La façon la plus flexible pour obtenir des Facts pour MCollective est de laisser de Puppet (ou Chef) lui fournir. Un moyen rapide pour stocker beaucoup de Facts est de la passer en Crontab et de stocker le résultat dans le fichier yaml :

**/etc/cron.d/facts.sh**

**\*/30 \* \* \* \* facter -y > /etc/mcollective/facts.yaml**

Alternativement, vous pouvez simplement créer ce fichier et entrez quelques faits aléatoires à des fins d'apprentissage. Le fichier doit être en format YAML.

Une fois que vous avez effectué les modifications, vous pouvez utiliser la demande de l'inventaire de MCO et lire la sortie pour voir si les facts remonte bien:

# mco inventory u05-adm-vm06.ass.parimutuel.local | awk '/Facts:/','/^$/'

Il est possible également d’interroger pour savoir si des nodes partagent la même valeur de facts (OS, release, etc...).

# **mco facts operatingsystem**

Report for fact: operatingsystem

Debian found 1 times

RedHat found 10 times

Finished processing 11 / 11 hosts in 287.48 ms

Ou encore :

# **mco facts uptime\_days**

Report for fact: uptime\_days

1 found 1 times

11 found 2 times

18 found 1 times

6 found 1 times

7 found 5 times

9 found 1 times

Finished processing 11 / 11 hosts in 27.44 ms

**# mco facts chef\_environment**

**# mco facts**

ATTENTION !!!!!!!!!!!!!!! Ne pas utiliser "**mcollective-facter-facts**":

Cet agent peut être lent à  exécuter, car il invoque facter pour chaque évaluation. Il faut utiliser le facter par defaut de Puppet avec la crontab pour renseigner le fichier yaml.

# INVENTORY :

Une des commandes de base fournies dans le client MCollective est la commande d'inventaire. Cette commande vous permet de voir comment un serveur donné est configuré, les collects et diverses statistiques fonctionnement.

La sortie donne la listes des plugins sont installés sur l'hôte. Il sera également vous donner les classes Puppet connus (si Puppet est en cours d'exécution sur l'hôte) ainsi que les facts connus. Vous devez exécuter cette commande sur un de vos serveurs et examiner la sortie.

Vous pouvez générer des rapports en vrac de l'inventaire :

# mco inventory --script inventory.mc (execution d’un script perso développé en Ruby)

# DISCOVERY MC :

La plupart des opérations de base effectuées par le client MCollective est de découvrir quels serveurs sont disponibles. Il utilisera cette information au moment de décider d'émettre des commandes.

**# mco find --with-identity / a/ --verbose**

Discovering hosts using the mc method for 2 second(s) .... 11

for-rec-appv1

stge-lunea1.adm.parimutuel.local

u05-adm-vm06.ass.parimutuel.local

u05-bw-vm12.ass.parimutuel.local

osi-dev-appv2.dev.parimutuel.local

stge-test.adm.parimutuel.local

col-rec-appv1.adm.parimutuel.local

u05-ems-vm15.ass.parimutuel.local

u05-ems-vm14.ass.parimutuel.local

osi-dev-appv1.dev.parimutuel.local

ams-jbs-vm48.dev.parimutuel.local

Discovered 11 nodes in 2.00 seconds using the mc discovery plugin

Comment le client a t-il déterminer quels serveurs correspondait à la recherche ? Il a utilisé le plugin **mc-discovery** configuré dans le fichier **client.cfg**.

# mco plugin doc mc

mc

==

MCollective Broadcast based discovery

Author: R.I.Pienaar <rip@devco.net>

Version: 0.1

License: ASL 2.0

Timeout: 2

Home Page: http://marionette-collective.org/

DISCOVERY METHOD CAPABILITIES:

Filter based on configuration management classes

Filter based on system facts

Filter based on mcollective identity

Filter based on mcollective agents

Compound filters combining classes and facts..

…

..

Le plugin de découverte de **mc envoie une requête de diffusion à tous les noeuds avec le filtre que vous spécifiez**.

* Si plus de 10 serveurs répondent, alors il enverra la demande comme une émission.
* Si moins de 10 serveurs répondent, il va envoyer des messages directs à chaque serveur.

le paramétre **direct\_addressing\_threshold** du fichier de configuration du client permet de modifier ce comportement!

**direct\_addressing\_threshold = 10**

# DISCOVERY FLATFILE / STDIN :

Une façon d'éviter la découverte de diffusion utilisée par mc est d'utiliser un plugin de découverte différente.

Les autres plugins de découverte fournis par défaut sont les "**Flatfile**" et "**stdin**".

Ce sont des mécanismes de découverte plus limitées qui utilisent une liste de noms à  partir d'un fichier ou de l'entrée standard.

On les invoque via **--disc-method :**

**--nodes filname**

**--disc-method flatfile --discovery-option filename**

**--disc-method sdin**

**Avec l'une de ces invocations, aucune requête de diffusion ne sera utilisée**. La demande sera envoyée directement à  une file d'attente spécifique à chaque node. (voir l’aide **'mco plugin doc flatfile'**)

# RPC :

La commande ‘**mco RPC’** est une **méthode pour envoyer une demande à  l'agent sans utiliser l'application cliente**.

Envoi d’une commande ping via l’utilisation RPC et la méthode de découverte par défaut MC:

**# mco rpc rpcutil ping**

Envoi d’une commande ping via l’utilisation RPC et la méthode de découverte flatfile :

**# mco rpc rpcutil ping --disc-method flatfile --disc-o| /.../list-server**

Il y a un certain nombre d'autres plugins de découvert disponibles pour MCollective, y compris ceux pour PuppetDB, Chef, MongoDB, RiakDB et Elastic search.

# RECHERCHE ET FILTRES :

Les filtres sont **utilisés par le plugin de Discovery pour limiter l’envoi d’une commande à certain serveurs**. Ils peuvent être appliqués à  n’importe quelle commande MCollective.

**--with-identity :** recherche le nom de node contenant …

**--with-class** : recherche une classe Puppet utilisé par un node

**--with-agent :** recherche les agents installés sur un node

Ils permettent de produire une liste de serveurs MCollective qui correspondent aux critères.

Exemple:

Trouver tous les hôtes avec un i dans leur nom:

**# mco find --with-identity /i/**

Inscrivez tous les serveurs Web appelée suivi d'un numéro:

**# mco find --with-identity /web\d/**

Listez tous les nodes qui ont une classe Puppet 'webserver' qui leur est appliquée:

**# mco find --with-class webserver**

Afficher tous les nodes qui exécutent le système d'exploitation CentOS:

**# mco find --with-fact operatingsystem=CentOS**

Afficher tous les nodes qui ont l'agent package d'installé:

**# mco find --with-agent package**

# COMBINER LES FILTRES :

Il existe deux types de filtres de combinaison.

Le premier type regroupe les classes de Puppet et les facts de facter. Exemple : pinger seulement les nodes CentOS avec une classe Puppet 'nameserver':

**# mco ping --with "/nameserver/ operatingsystem=CentOS"**

Le deuxième type est appelé un **filtre de sélection** et est le filtre le plus puissant disponible. C’est le seul filtre où vous pouvez utiliser les opérandes **AND**, **OR** et **NOT OR**.

**# mco ping --select "operatingsystem=CentOS and /nameserver/"**

Ping CentOS chaque node qui n’est pas dans l'environnement de développement:

**# mco ping --select "operatingsystem=CentOS and !environment=dev"**

Ping chaque serveur web virtualisé correspondre soit à la classe Puppet httpd ou nginx.

**# mco ping --select " ( /httpd/ or /nginx/ ) and is\_virtual=true"**

# LIMITER LES ENVOIS :

Au-delà de ce que peuvent faire les filtres, vous pouvez également limiter le nombre de serveurs qui reçoivent la demande ou le nombre de processus envoyé en même temps.

**# mco find --limit 15**

Un seul serveur CentOS:

**# mco facts architecture --one --with-fact operatingsystem=CentOS**

Cinq serveurs qui ont la classe Puppet serveur web qui leur est appliquée:

**# mco facts osfamily --limit 5 --with-class webserver**

Un tiers des serveurs qui ont la classe Puppet serveur web qui leur est appliquée:

**# mco facts is\_virtual --limit 33% --with-class webserver**

Requête du paquet sudo version dans des lots de 10 serveurs espacé 20 secondes:

**# mco package status sudo --batch 10 --batch-sleep 20**

Interroger la version Puppet de tous les serveurs allemands, cinq traitement toutes les 30 secondes:

**# mco package status puppet --batch 5 --batch-sleep 30 --with-fact country=de**

Ping chaque serveur avec ‘w’ dans son nom sans délai:

**# mco ping --with-identity /w/**

# OUTPUT:

Vous pouvez également contrôler la sortie. Elle fournit des données structurées au lieu de texte convivial en réponse: (# mco plugin --json command options....)

**--no-progress** : sans la status bar

Cela vous indique combien de temps prend la découverte, et vous donne des statistiques RPC complètes:

**# mco plugin --verbose command option ....**

Cela envoie les commandes mais ignore la file d'attente de réponse tout Ã :

**# mco plugin --no-results command option.....**

Pour afficher les réponses seulement échoué ou seulement réussi dans une requête:

**# mco plugin --display failed command options...**

**# mco plugin --display ok command options...**

**# mco plugin --display all command options...**

# PLUGIN PUPPET:

L'agent Puppet écrit les sortie "classes" dans le fichier catalogue Puppet 'classes.txt' dans le $statedir (/var/lib/puppet/classes.txt). Mcollective sait où elle se trouve par défaut.

Il faut que l'option 'classfile' présent dans la section [agent] du fichier puppet.conf corresponde au 'classesfile' du fichier de configuration server.cfg de mcollective.

Ainsi Mcollective est capable de savoir Ã  quel classe l'agent est associÃ©.

COMPLETION BASH:

Mcollective fournit un plugin pour bash pour permettre l'achÃ¨vement de ligne de commande.

# cd marionette-collective-2.5.3

# cp ext/bash/mco\_completion.sh /etc/bash\_completion.d/

WEB CLIENT:

Il existe deux interfaces Web disponibles pour la gestion mcollective. Puppet Labs fournit une interface Web pour contrÃ´ler mcollective dans leur ligne de produits de marionnettes Enterprise.

-mcomaster: interface free disponible sur: https://github.com/ajf8/mcomaster

AGENT ET PLUGINS CLIENT:

Puppet Labs fournit un certain nombre d'agents mcollective qui savent comment faire les tÃ¢ches de gestion de systÃ¨mes commune (par exemple, une requÃªte, dÃ©marrer et arrÃªter des processus, et de requÃªte, installer et supprimer des paquetages).

-mcollective-filemgr-agent

-mcollective--nettest-agent

-mcollective--package-agent

-mcollective--service-agent

Vous aurez besoin de le faire sur chaque serveur dans votre environnement. Sur les postes clients, vous aurez besoin d'installer le module client correspondant.

Le fichiers du plugin agent sont nommÃ©s libdir/mcollective/agent/name(rb|ddl|erb).. Il ya habituellement une application cliente dans libdir/mcollective/agent/NAME.rb. Il peut y avoir util ou d'autres rÃ©pertoires, qui doivent Ãªtre copiÃ©s mot Ã  mot.

!!!!!!!!!!!!!!Le rÃ©pertoire mcollective va Ã  l'intÃ©rieur libdir. Dans le cas de Red Hat, cela signifie que le chemin d'accÃ¨s complet contient la chaÃ®ne mcollective/mcollective; veillez Ã  ne pas sauter accidentellement le deuxiÃ¨me mcollective

AprÃ¨s avoir installÃ© de nouveaux agents sur un nÅ“ud de serveur, vous dire mcollectived pour recharger les agents. La mÃ©thode la plus simple est de redÃ©marrer mcollectived. mco inventory nous indique si les nouveaux agent son disponible.

# mco inventory nodename |Â awk ' /Agents:/','/^$'

Vous pouvez Ã©galement interroger pour obtenir une liste de chaque serveur qui possÃ¨de l'agent installÃ©:

#Â mco find --with-agent filemgr

Pour interagir avec ces agents, nous avons besoin d'avoir installÃ© les plugins de clients

Desactiver un agent sans le desinstallÃ©:

Il ya deux faÃ§ons de dÃ©sactiver un agent. La premiÃ¨re option est dans le fichier de configuration du serveur.

plugin.plugin\_name.activate\_agent = false

L'autre faÃ§on est de crÃ©er un fichier de configuration pour cet agent particulier:

$ echo "activate\_agent = false" | tee -a /etc/mcollective/plugins.d/plugin\_name.cfg

ous pouvez obtenir la list des applications disponible sur un node avec la commande 'doc':

# mco plugin doc

ou

# mco plugin doc agent/package

Les applications ajoutent sous-commandes personnalisÃ©es (appelÃ©es faces) au client de mco, permettant un accÃ¨s facile aux commandes fournies par chaque plugin client.

La commande 'mco plugin package xxx' permet de crÃ©er des plugins.

# yum search --enablerepo=puppetlabs\* mcollective

SERVER STATISTIQUE:

En plus de la liste des agents disponibles sur un serveur, mcollective rapporte aussi de retour un bon nombre de statistiques de la demande d'inventaire

MONITORING SERVERS:

Un contrÃ´le actif serait de lancer un appel Ã  un agent disponible sur chaque nÅ“ud et valider les rÃ©sultats. Cela pourrait Ãªtre quelque chose d'aussi simple que mco ping, qui est un test de connectivitÃ© de bas niveau qui ne nÃ©cessite pas d'authentification ou d'autorisation. Ou vous pourriez tester Ã  un plugin spÃ©cifique (par exemple, un test de NRPE). Un exemple de la faÃ§on de vÃ©rifier cela avec Nagios peut Ãªtre trouvÃ© Ã  Puppet Labs wiki AgentRegistrationMonitor.

vÃ©rifier quels systÃ¨mes ont l'agent mcollective marionnettes installÃ©:

mco find --with-agent puppet

mco puppet count

mco puppet summary

CONTROLLER LE DAEMON PUPPET:

Lors de la maintenance, vous pouvez dÃ©sactiver l'agent de marionnettes sur certains noeuds. Lorsque vous dÃ©sactivez l'agent, vous pouvez fournir un message de laisser les autres savent ce que vous faites:

# mco puppet disable --with-identity test.node.local message="Arret du service puppet pour test"

#Â mco puppet runonce --with-identity test.node.local

Pour rÃ©activer l'agent:

# mco puppet enable --with-identity test.node.local

INVOQUER PUPPET RUN:

voir l'aide : # mco help puppet

L'invocation simple est naturellement Ã  fonctionner immÃ©diatement marionnettes sur un systÃ¨me:

# mco puppet runonce --with-identoty test.node.local

# mco puppet status --with-identity test.node.local

Que faire si vous avez besoin pour exÃ©cuter marionnettes instantanÃ©ment sur chaque hÃ´te CentOS pour fixer les fichier sudoers?

# mco puppet runonce --tags=sudo --with-fact operatingsystem=CentOS

#Â mco puppet status --wf operatingsystem=CentOS