



Présentation Générale

Par l'équipe de [Roboconf](#) / [@Roboconf](#)

LE PROJET

Roboconf

- Lancé 2013
- Open source (Apache v2)
- Hébergé sur GitHub

Roboconf est basé sur un prototype développé par l'**Université Joseph Fourier** à Grenoble (France). Le projet est en cours d'industrialisation par **Linagora**, un éditeur logiciel français spécialisé dans l'open source.

En Quelques Mots

Roboconf est une solution basée sur une messagerie asynchrone pour gérer le déploiement et la reconfiguration d'applications réparties.

Il transpose des concepts de modèles à composant (tels qu'OSGi) au déploiement et à la gestion de piles logicielles.

Utilisateurs Ciblés

Roboconf adresse des déploiements où la notion d'élasticité est centrale.

Le profil type d'un utilisateur de Roboconf est ce que l'on appelle un **DevOps**, c'est-à-dire quelqu'un à cheval entre le développement et des problématiques de production.

Fonctionnalités - 1/2

Déploiement

Installation et configuration d'applications.

Cibles de Déploiement

Déploiement sur de multiples cibles : infrastructures de cloud (IaaS), objets embarqués, serveurs plus classiques, etc. Les déploiements hybrides sont également supportés.

Fonctionnalités - 2/2

Reconfiguration

Grâce à Roboconf, des portions d'application répartie peuvent se notifier les unes les autres et se reconfigurer en conséquence au travers de scripts.

Roboconf prend en charge tous les aspects du cycle de vie.

Supervision

La gestion de la supervision est en cours d'implémentation.

Cas d'Usage Couverts

- Déploiement dans le Cloud
- Déploiements hybrides
- Environnements fortement dynamiques

Roboconf peut déployer n'importe quel type d'application. Il est bien adapté aux applications réparties. Il n'y a pas de contrainte sur les langages ou les frameworks utilisés par ces applications. Du moment qu'un déploiement ou une reconfiguration peut être scripté, Roboconf peut l'automatiser.

Les déploiements à grande échelle sont encore mal couverts.

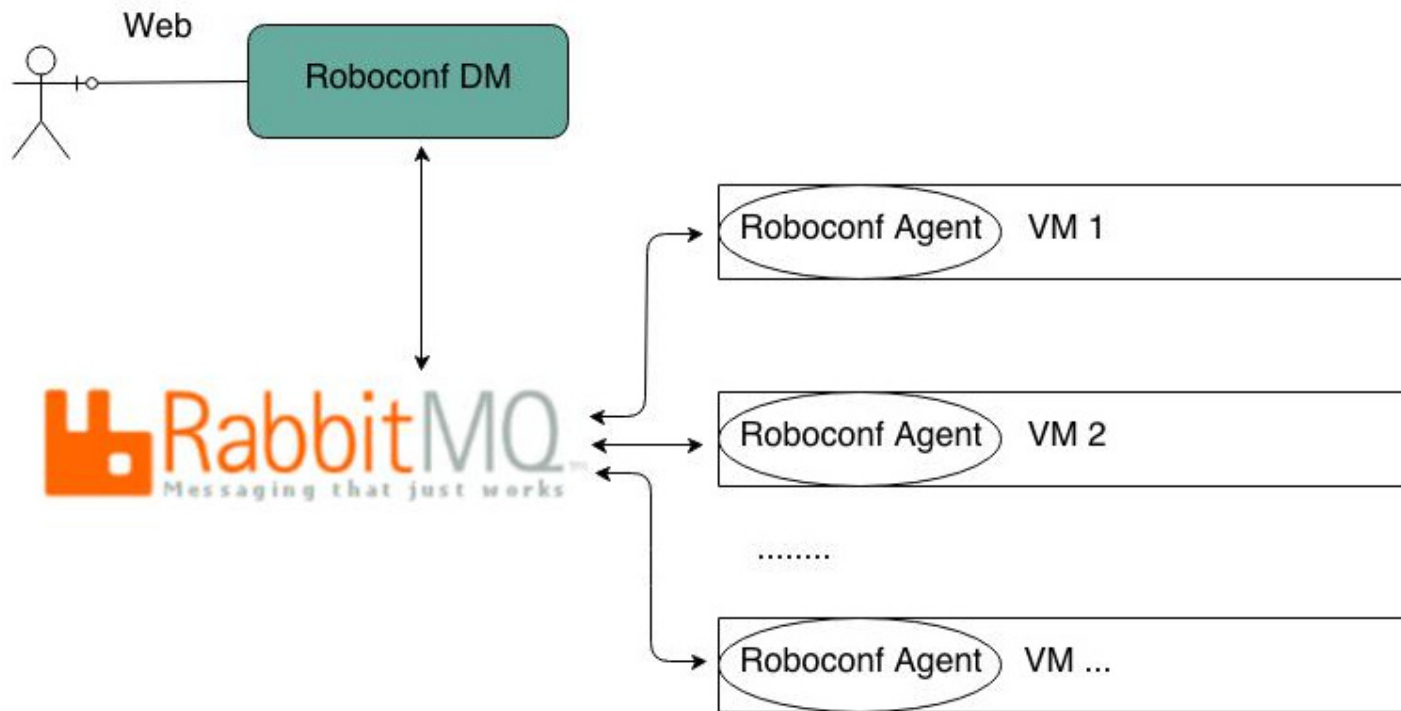
Roboconf...

... ne réinvente pas la roue.

Grâce à une approche à base de plug-ins, il est possible d'utiliser Bash ou Puppet. D'autres solutions sont envisageables, telles que Chef ou CFEngine. Il n'existe pas encore de plug-in pour ces solutions, mais leur écriture est à la portée de n'importe quel développeur Java.

C'est à travers de ces extensions que Roboconf gère les changements au niveau du cycle de vie d'une application. La vraie plus-value de Roboconf réside dans la dynamisme qu'il ajoute derrière ces plug-ins.

ARCHITECTURE



Roboconf repose sur 3 éléments : le DM, des agents et un serveur de messagerie AMQP (RabbitMQ). Les VM désignent des Machines Virtuelles ou n'importe quel type de machine (serveur, device...).

Le DM

Le DM (ou gestionnaire de déploiements en français) est chargé de piloter les déploiements. Il est l'interface par laquelle passent les commandes d'administration.

Le DM fournit un ensemble d'API REST.

Il est généralement accompagné d'une console web (roboconf-web-administration) qui fournit une interface graphique pour l'invocation de cette API REST.

En fonction des opérations REST invoquées, le DM peut...

- ... interagir avec un gestionnaire de cloud pour créer ou supprimer une machine.
- ... communiquer avec des agents Roboconf au travers du serveur de messagerie.

Les Agents

Chaque machine sur laquelle Roboconf doit déployer ou gérer quelque chose, doit avoir un agent installé et démarré. Un tel agent est à l'écoute...

- ... d'instructions provenant du DM.
- ... de notifications provenant d'autres agents.

En fonction de ce qu'il reçoit, un agent va exécuter une recette.

Une recette est un ensemble d'actions associées à une brique applicative et attachée à une extension de Roboconf (Puppet, Bash...). L'exécution d'une recette aboutit à un changement dans l'application.

Un agent peut installer et gérer plusieurs éléments sur cette même machine.

Les applications déployées par Roboconf ne sont pas conscientes de sa présence. De ce point de vue, Roboconf est bien un intergiciel (ou middleware).

Le Serveur de Messagerie

Le serveur de messagerie est le medium utilisé pour échanger des informations entre...

- ... le DM et les agents.
- ... un agent et d'autres agents.

Le serveur de messagerie actuellement utilisé est RabbitMQ.
C'est une solution robuste et open source qui implémente la norme AMQP.
De plus, elle supporte un mode *cluster*.

A propos du DM

Le DM peut être déployé n'importe où.

Généralement, il est considéré comme une bonne pratique de le faire tourner en local (c'est à dire au sein de son système d'information). Cela réduit les précautions à prendre comparé à ce qu'il faudrait faire s'il tournait sur un serveur public.

Le DM a été conçu pour ne fonctionner que lorsque cela est nécessaire. Cela signifie qu'il peut être arrêté lorsqu'aucune action d'administration n'est en cours. L'état courant des applications est alors persisté sur le système de fichiers. Cet état sera réactualisé au prochain démarrage en interrogeant les agents. L'utilisation d'un simple système de fichiers pour la persistance permet de mettre en place des procédures simples pour la restauration en cas de crash.

A propos de RabbitMQ

Le serveur de messagerie est le point clé qui assure le fonctionnement global. Il doit donc être accessible par le DM, ainsi que par tous les agents. C'est la raison pour laquelle il doit être accessible via une URL publique.

Roboconf utilise les mécanismes d'authentification fournis par RabbitMQ pour sécuriser l'accès au serveur de messagerie.

A propos des Agents

Le cas d'usage le plus pratique actuellement pour Roboconf est lorsque les machines cibles sont hébergées dans le cloud. Le DM crée alors les machines virtuelles depuis le catalogue d'images du IaaS. Afin de s'assurer que chaque machine cible a bien un agent installé et démarré, le plus simple est donc de créer une image virtuelle sur laquelle on pré-installe un agent Roboconf, puis d'enregistrer cette image dans le catalogue.

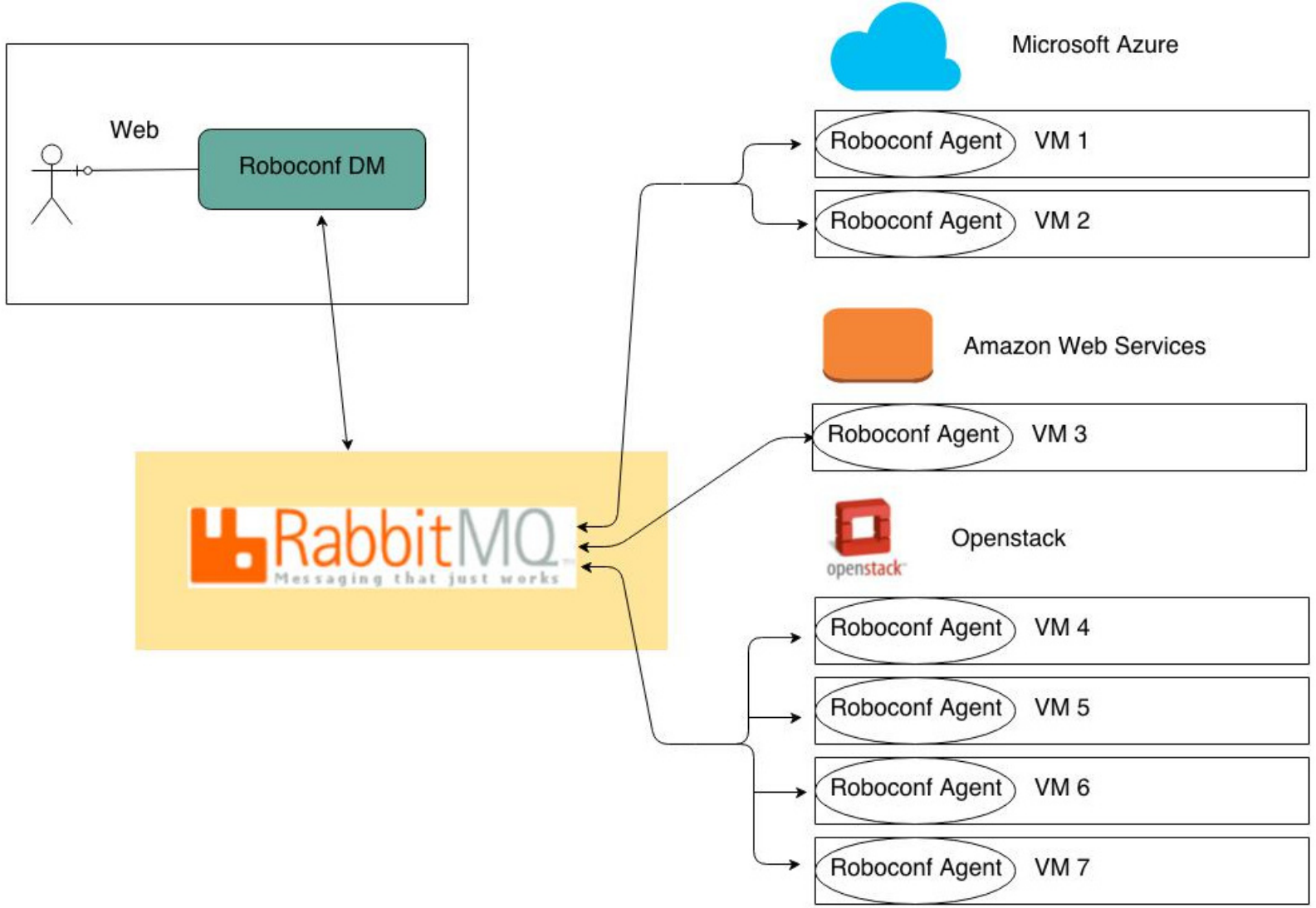
Pour d'autres cibles, il faut généralement installer et configurer l'agent à la main. Des solutions utilisant SSH peuvent être utilisées dans cette optique.

La plupart des informations requises par un agent, sont détenues et fournies par le DM. La configuration de l'agent est donc minimale.

Multi-**IaaS**

Les déploiements hybrides et multi-cibles sont supportés. Cette possibilité peut être utilisée par exemple en cas de périodes critiques (pic de charge) ou dans le cadre d'une migration. Cela peut aussi être utilisé pour la sécurité et la protection des données.

La prochaine image illustre ça avec **quelques** infrastructures de cloud. Pour rester simple, on ne montre que les interactions entre les briques Roboconf, et non entre les applications déployées par Roboconf.



CONCEPTS

Le Graphe

Le graphe définit des éléments logiciels et les relations qui les unissent. Un élément logiciel peut être une machine virtuelle, un serveur d'application, une application, etc. On les appelle aussi des **composants**.

Deux types de relations sont considérées.

Conteneur

Un composant doit être déployé sur un autre.

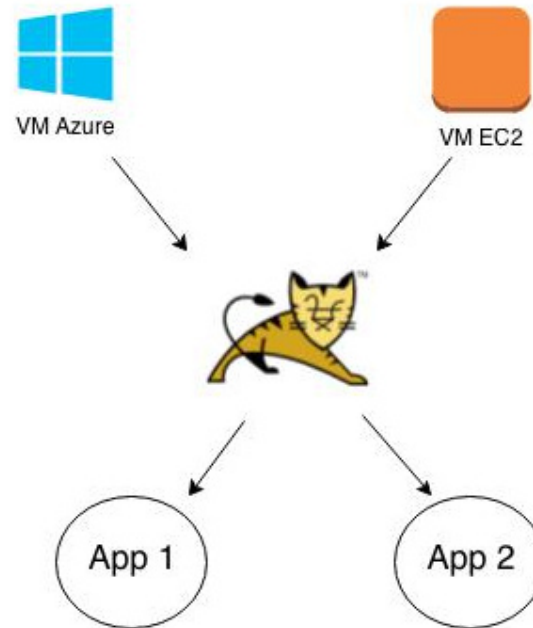
Exemple : un WAR se déploie sur un serveur d'application.

Exécution

Un composant a besoin d'un autre pour fonctionner.

Exemple : une application web a besoin d'une base de données.

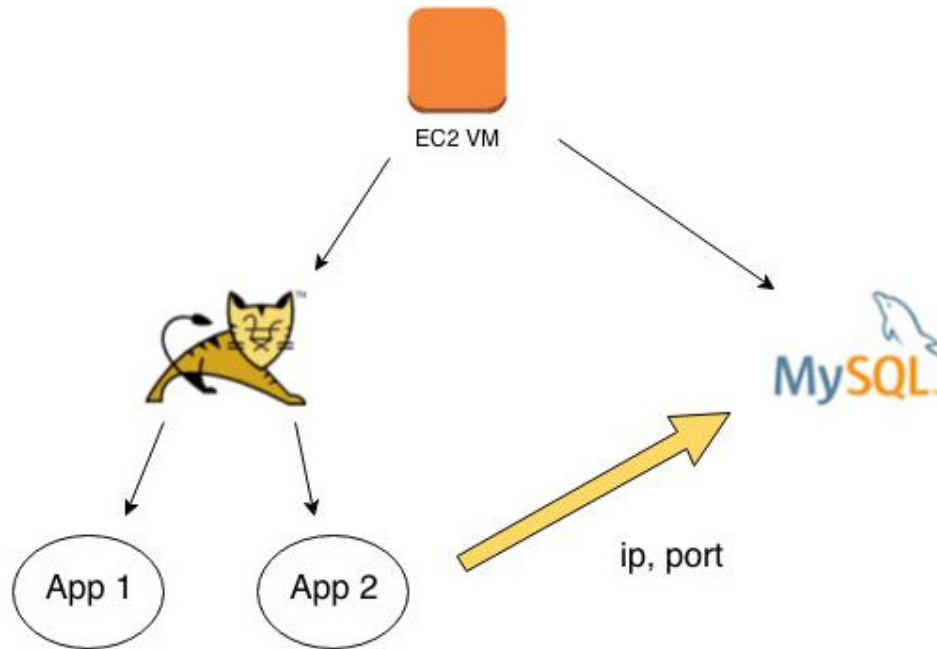
Exemple de Relation « Conteneur »



Ce graphe indique que l'on peut déployer Tomcat soit sur une VM Amazon, soit sur une VM Azure. Et on a listé 2 applications qui se déploient sur Tomcat. Si l'on souhaite déployer une troisième application web, il faut compléter ce graphe.

Les conteneurs peuvent être imbriqués. Ainsi, Tomcat est à la fois contenu et un conteneur.

Exemple de Relation « Exécution »



Nous avons ici, en plus d'une relation de type « conteneur », une relation de type « exécution ». Dans ce graphe, **App 2** dépend d'une base MySQL, ce qui n'est pas le cas de **App 1**. Cela signifie que Roboconf ne peut pas démarrer **App 2** tant qu'il ne connaît pas les propriétés **ip** et **port** d'une base MySQL.

Avantages du Graphe

Pour un composant logiciel...

- On sait si l'on peut déployer des choses dessus.
- On sait sur quoi on peut le déployer.
- On sait de quels autres composants il dépend.

En Pratique

Quand on veut déployer un nouveau composant logiciel...

- ... on peut le déployer sur un conteneur existant.
- ... on peut créer son conteneur en même temps.

Les Graphes

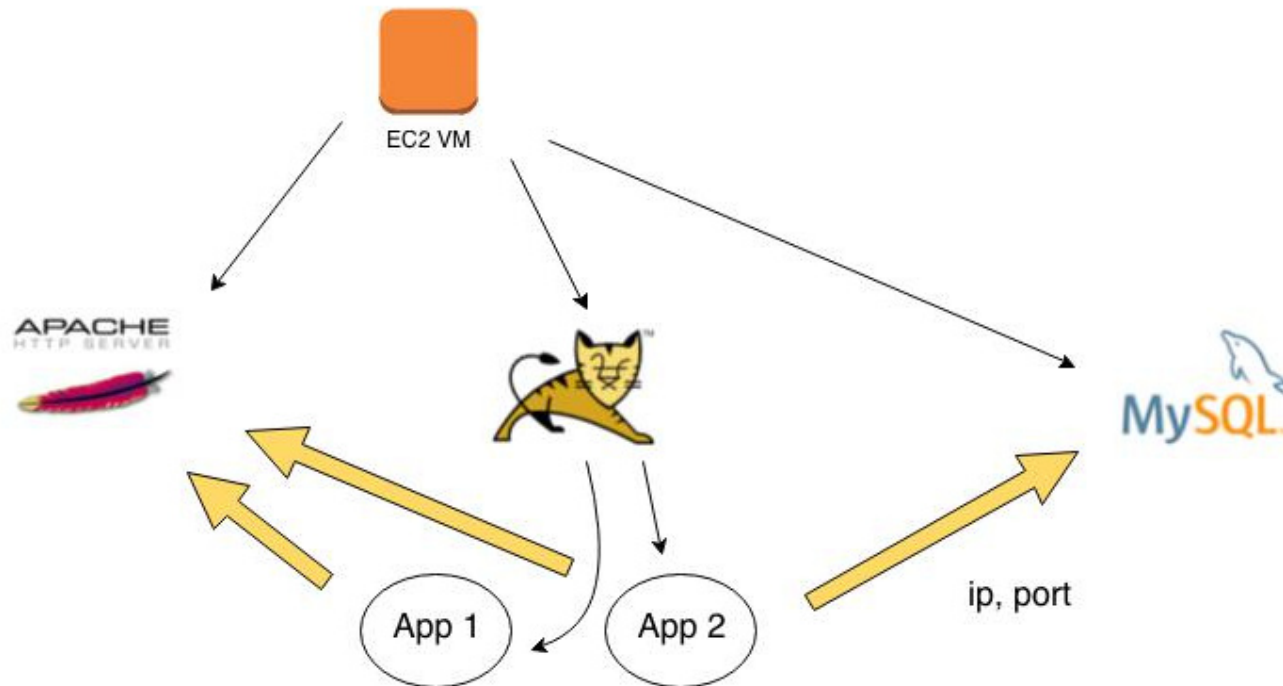
Le graphe est une structure avec 2 niveaux de lecture (relations de type conteneur et exécution). On peut le voir comme un ensemble de règles qui régissent les déploiements et les reconfigurations.

Toutefois, pour être précis, il n'y a pas qu'un seul graphe. En effet, il est possible d'en avoir plusieurs au travers des fichiers de configuration.

Cela permet par exemple de définir des *pseudo-appliances*.
On peut donc avoir des déploiements très contraints, ou bien très flexibles.

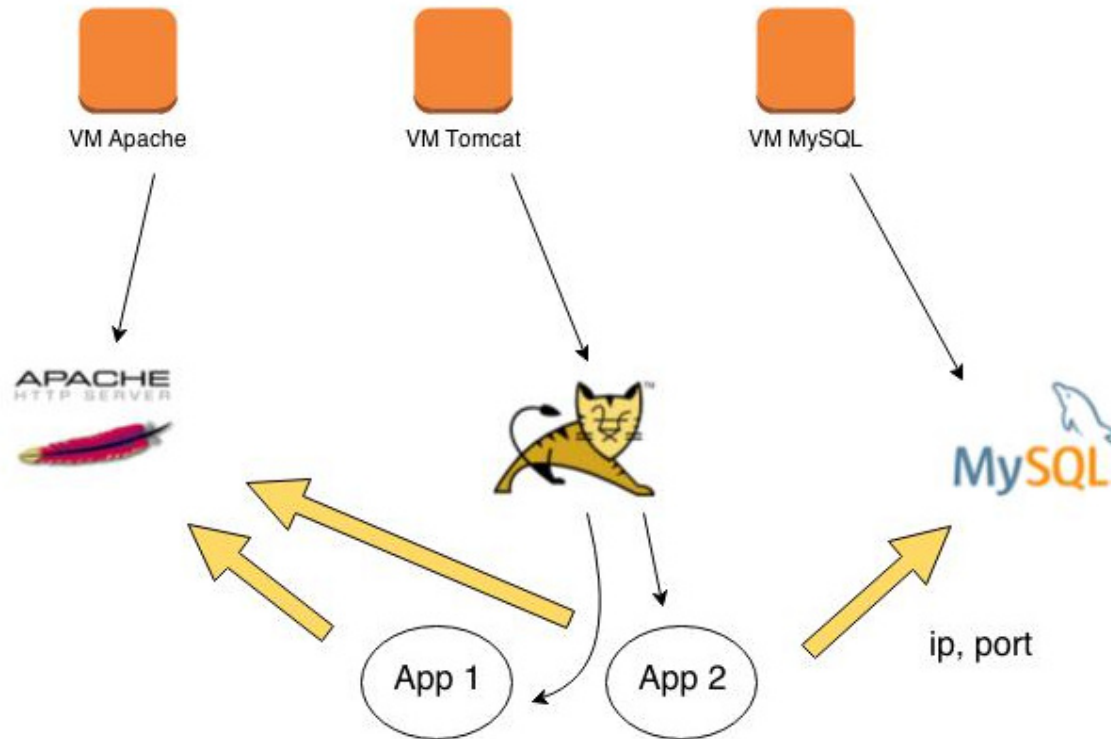
C'est une possibilité intéressante dans la mesure où ceux qui développent ou conçoivent l'application ne sont pas les mêmes que ceux qui la gèrent une fois en production. Le fait de contraindre le déploiement des parties élastiques (ou scalables) réduit le risque d'erreurs pour les équipes de production.

Un Graphe Flexible



Sur cet exemple, on considère que les 3 briques logicielles (équilibreur de charges Apache, Tomcat et MySQL) peuvent être déployées sur une même VM, ou en tout cas, sur des VM du même « type ».

Un Graphe Contraint



Contrairement au cas présent, on considère ici que chaque brique doit être déployée sur son propre type de VM. Donc, pas de mélange des applications.

Chaque type de VM peut avoir sa propre configuration (CPU, stockage, configuration du réseau...).

Les Recettes

Chaque composant du graphe est associé à un ensemble de recettes. Plus précisément, chaque composant est associé à une extension (propriété « installer »).

Un composant associé avec l'extension **Puppet** utilisera un module Puppet comme recette. Un composant associé avec l'extension **Bash** utilisera des scripts Bash comme recette.

Les recettes définissent les actions à entreprendre lorsqu'une action d'administration est reçue par un agent. Cela correspond à des actions sur le cycle de vie : *deploy*, *start*, *stop* et *undeploy*.

Il existe aussi une étape particulière appelée **update**. Elle est utilisée par Roboconf lors des phases de reconfiguration. Cette partie-là est détaillée un peu plus loin.

Les Instances

Les instances signifient des « instances de composants ».
Ces composants sont ceux définis dans le graphe. Le graphe est un ensemble de règles. On peut aussi le voir comme un méta-modèle pour des instances.
Par conséquent, toute instance est conforme à ce qui a été défini dans le graphe.

Une autre comparaison entre composants et instances pourrait être faite en prenant comme exemple classes et instances de classes dans la programmation orientée-objet.

Les instances héritent de propriétés définies dans leur composant. Elles peuvent les surcharger et même en définir de nouvelles (scope local). Les relations entre instances sont définies une fois pour toute dans le graphe.

Une instance n'est associée qu'avec un seul composant.

Les Méta-Données d'une Application

Un projet Roboconf est appelé une application.
Il vient avec la définition d'un graphe et potentiellement d'instances prédéfinies.

Il doit aussi contenir des informations à propos de l'application elle-même.
Nom, description, qualificatif (version)...

EXEMPLE : LAMP

L'Exemple LAMP

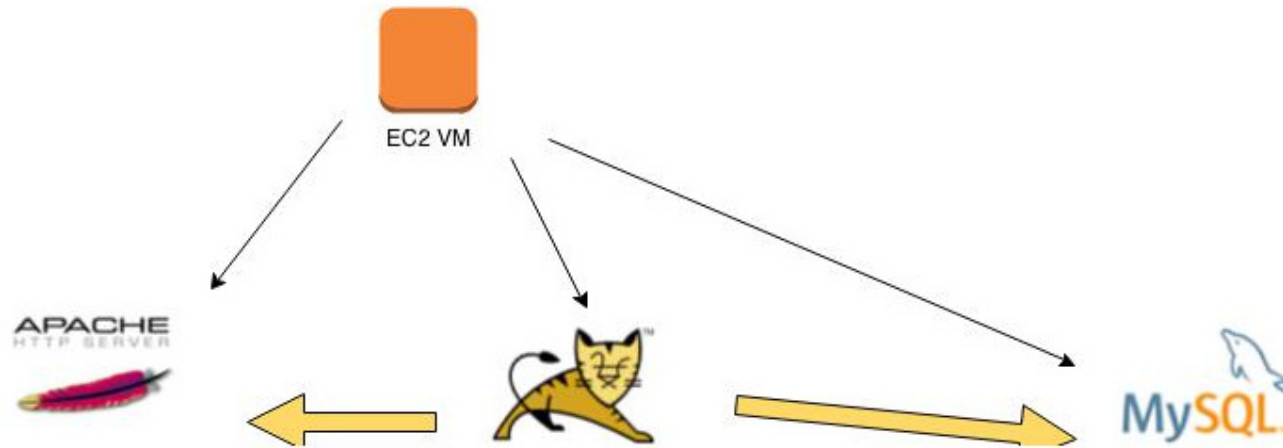
C'est l'exemple habituel pour expliquer Roboconf.
Il s'agit de déployer une pile usuelle et constituée...

- ... d'un équilibreur de charge (un serveur Apache avec modJK)...
- ... un serveur d'application (Tomcat)...
- ... et une base de données (MySQL).

Cet exemple illustre à la fois le déploiement et la reconfiguration.
Nous utilisons ici un graphe simple dans lequel notre application web est installée avec Tomcat. Elle n'apparaît donc pas en tant que telle dans le graphe, mais fait partie de la recette d'installation de Tomcat.

Pour la suite, nous supposons que nous déployons sur Amazon Web Services.

Le Graphe



Nos briques, Apache, Tomcat et MySQL, peuvent être déployées sur une VM. Tomcat a besoin d'une base MySQL. Et l'équilibreur de charge doit être notifié chaque fois qu'un nouveau Tomcat est instancié ou supprimé.

Encore une fois, Tomcat ne désigne pas ici que le serveur. Pour faire simple, notre recette de déploiement déploie en même temps une application web. Bien évidemment, on peut faire la distinction entre les deux si l'on souhaite aller plus loin.

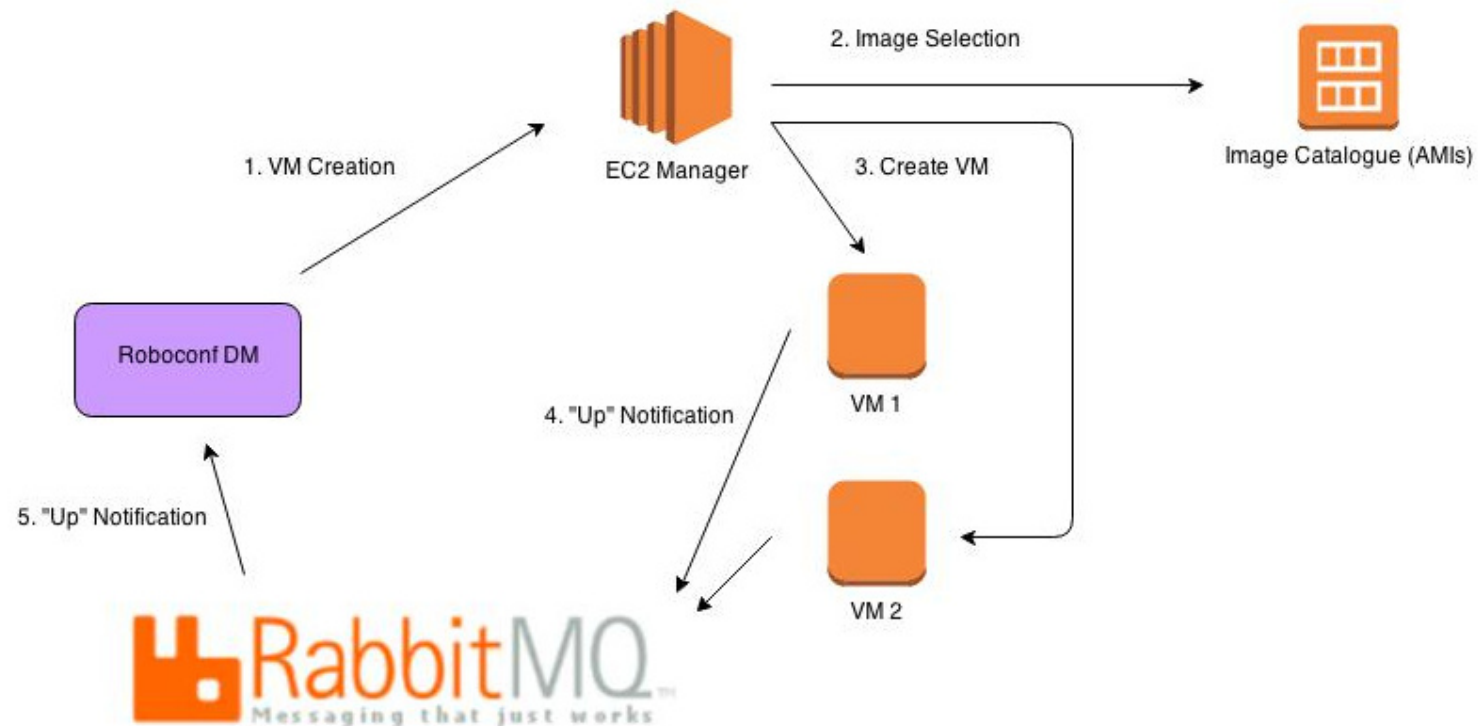
Définition des Instances

Nous avons notre graphe.
Supposons que nous définissions les instances suivantes.

- VM 1 (type: EC2 VM)
 - Apache (type: Apache)
 - Tomcat 1 (type: Tomcat)
- VM 2 (type: EC2 VM)
 - MySQL (type: MySQL)

Il ne s'agit encore que d'un modèle, c'est à dire quelque chose défini dans nos fichiers de configuration. Voyons voir ce qui se passe si nous **déployons tout** en même temps.

Création de VM

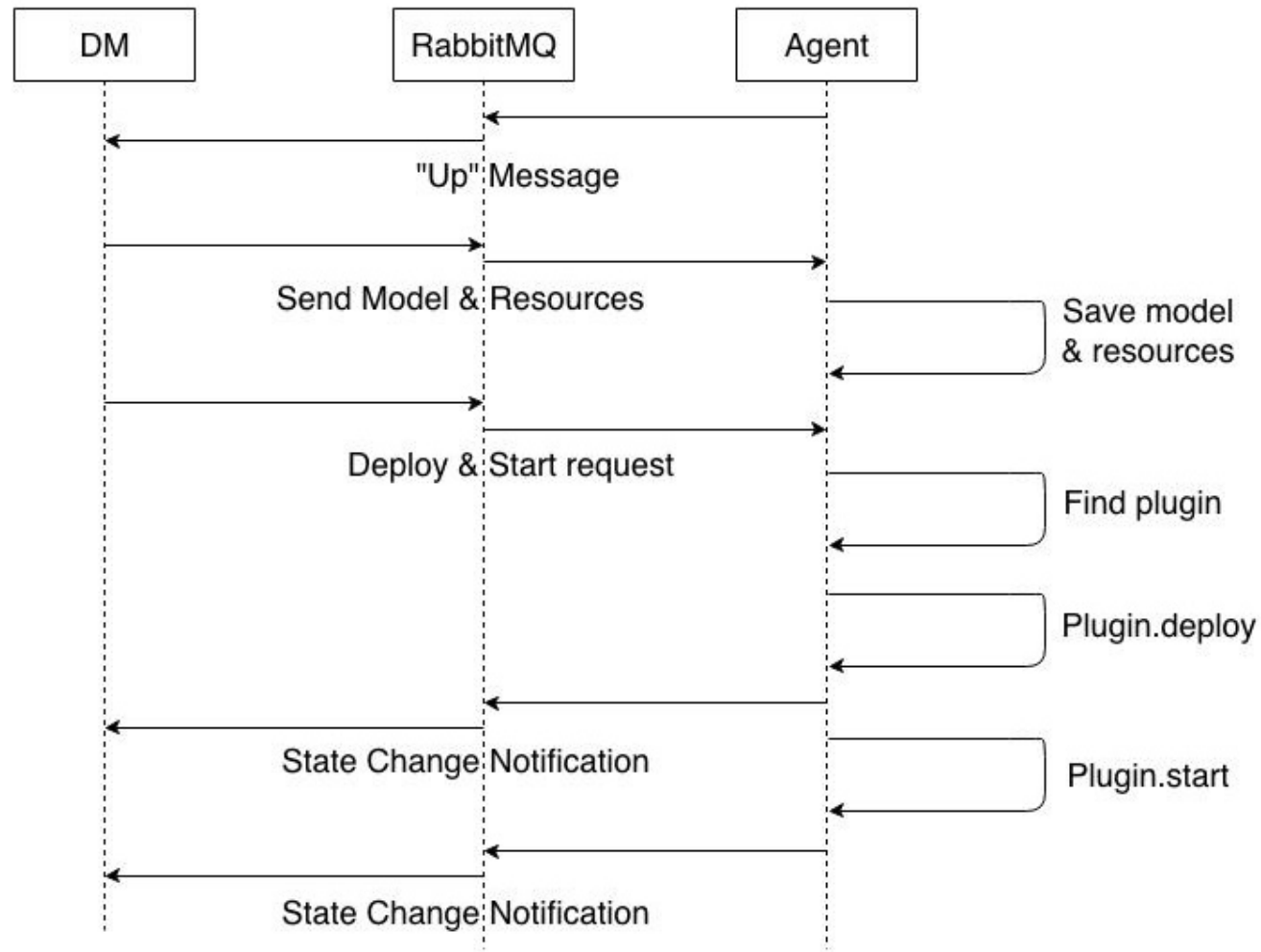


Le DM envoie une demande de création de VM au gestionnaire de cloud (EC2). La requête précise l'ID de l'image et la configuration matérielle à utiliser pour créer une nouvelle VM.

L'image virtuelle contient un OS et un agent Roboconf. Lorsque l'agent démarre avec le système, il récupère sa configuration, puis notifie le DM qu'il est opérationnel.

Intéractions entre le DM et les Agents

Voici ce qui se passe lorsqu'un agent démarre.



Cas Particulier pour l'opération « Start »

Un agent manipule des instances (associées à des composants).
Et un agent utilise aussi des extensions pour gérer le cycle de vie d'une instance.

Toutefois, « plugin.start » ne démarre pas vraiment une instance. En réalité, cela **essaye** de la démarrer. En effet, avant de pouvoir exécuter cette action, Roboconf doit vérifier que toutes les dépendances (tous les imports de variables) sont bien là. Si elles sont satisfaites, l'instance peut être démarrée.

Autrement, l'instance va rester dans l'état **starting** jusqu'à ce que ses dépendances soient résolues.

Avant Synchronisation

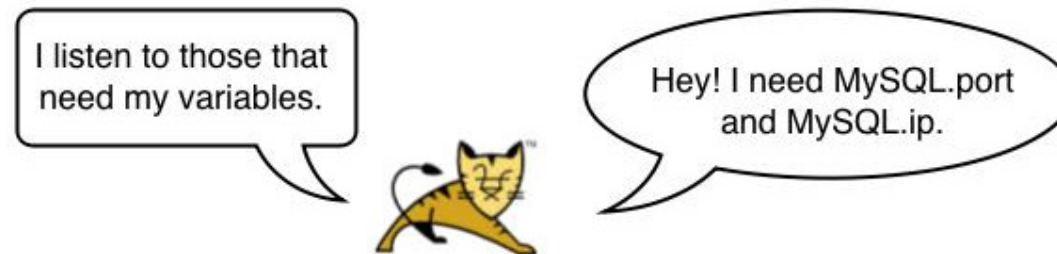
Reprenons notre exemple. Les VM ont été créées, les agents ont démarré et ils ont installé leurs applications respectives. Mais celles-ci ne sont pas encore lancées. Par ailleurs, Tomcat doit savoir où MySQL tourne, et Apache ne sait pas encore s'il y a des Tomcat quelque part.



En fait, au fur et à mesure que les déploiements progressent, les agents vont échanger des informations au travers de RabbitMQ.

Echange de Variables - 1/6

Imaginons que dans ce scénario, Tomcat soit le premier à être déployé.



Pour commencer, son agent local commence à écouter ceux qui pourraient avoir besoin des variables de Tomcat.

Puis l'agent va envoyer une requête pour être informé de la présence de ses dépendances (MySQL). Tomcat ne pourra pas démarrer tant que ces variables ne seront pas connues. Comme il est seul, personne ne peut répondre à sa requête.

Echange de Variables - 2/6

Imaginons maintenant que Apache soit le deuxième à être démarré.

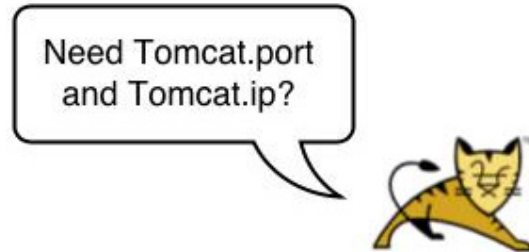


Son agent local commence à écouter ceux qui pourraient avoir besoin de ses variables.

Puis il envoie une requête pour résoudre ses dépendances (Tomcat). Apache ne pourra démarrer tant qu'il ne connaîtra pas au moins un Tomcat.

Echange de Variables - 3/6

L'agent en charge de Tomcat reçoit la requête émise par l'agent Apache.



Comme Tomcat n'a pas encore résolu ses dépendances, il n'est pas démarré.
Il ne peut donc pas propager ses variables.

Echange de Variables - 4/6

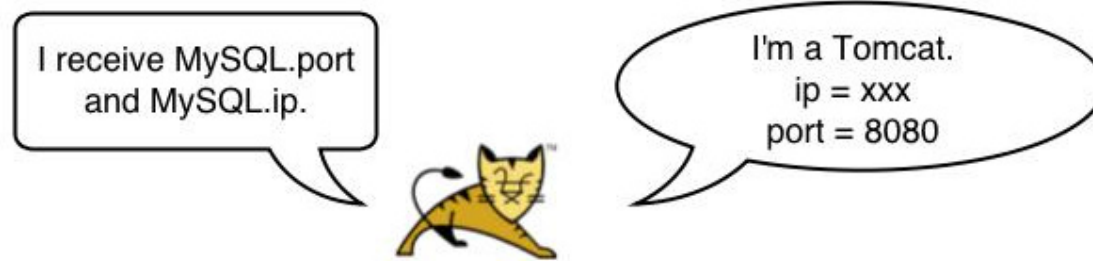
Finalemment, MySQL est le dernier à être déployé.
Le graphe indique que son composant n'a aucune dépendance. Il peut donc directement démarrer.



Lorsqu'il démarre, son agent publie ses variables. Et il n'a rien à écouter car pas de dépendance.

Echange de Variables - 5/6

Tomcat (enfin, son agent) reçoit la notification émise par l'agent de MySQL.
Toutes ses dépendances sont enfin résolues et il peut démarrer.



Il peut ainsi publier ses propres variables.

Echange de Variables - 6/6

Apache est finalement notifié des variables Tomcat.
Il peut alors démarrer.



Puisqu'il n'exporte aucune variable, l'agent Apache ne publie rien sur RabbitMQ.

Deployements en Parallèle

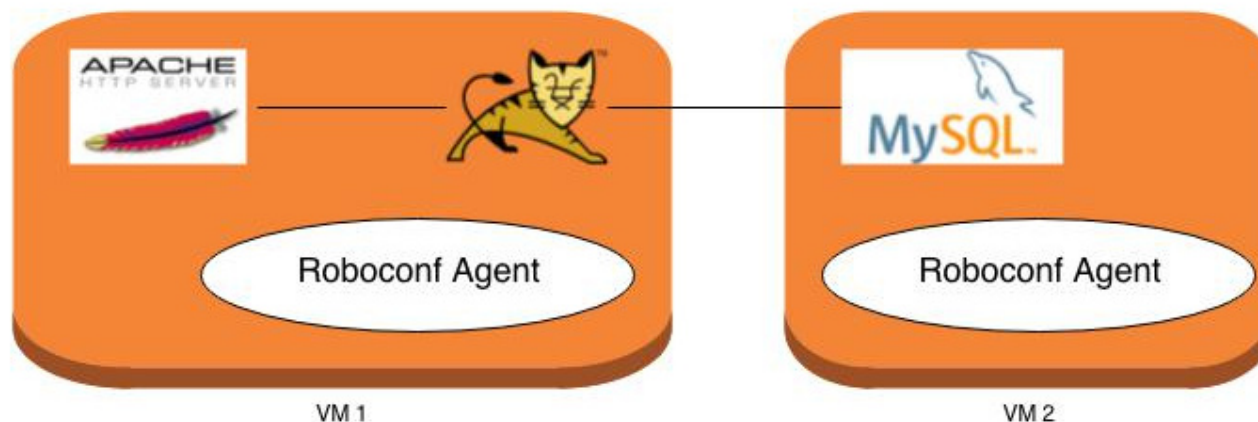
Roboconf parallélise tout ce qu'il peut.
Ici, deux flux ont été exécutés en parallèle.

- Créer la VM 1, deployer Apache et Tomcat.
- Créer la VM 2 et deployer MySQL.

Alors que les déploiements étaient en cours, des messages ont été échangés entre agents, de sorte à ce que chaque instance puisse résoudre ses dépendances avant de démarrer.

Peut importe dans quel ordre les déploiements se font, Roboconf garantit que tout tombera en marche à la fin. La clé réside dans les échanges asynchrones entre agents.

Résultat du Déploiement



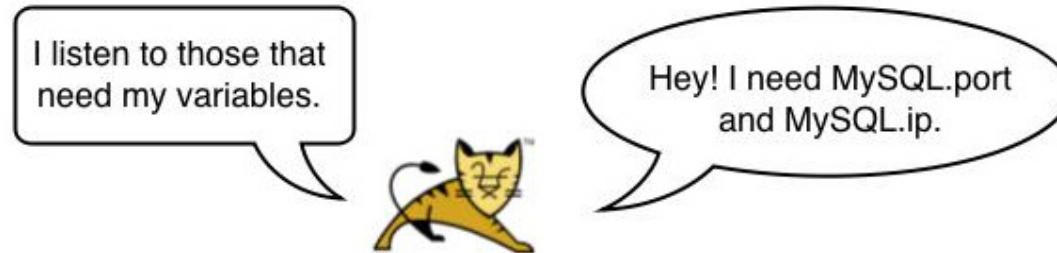
Voici ce que l'on obtient à la fin.

Essayons maintenant de rajouter un nouveau serveur Tomcat.

Dans notre modèle, nous définissons une nouvelle instance à déployer et démarrer.

- VM 3 (type: EC2 VM)
 - Tomcat 2 (type: Tomcat)

Reconfiguration - 1/4



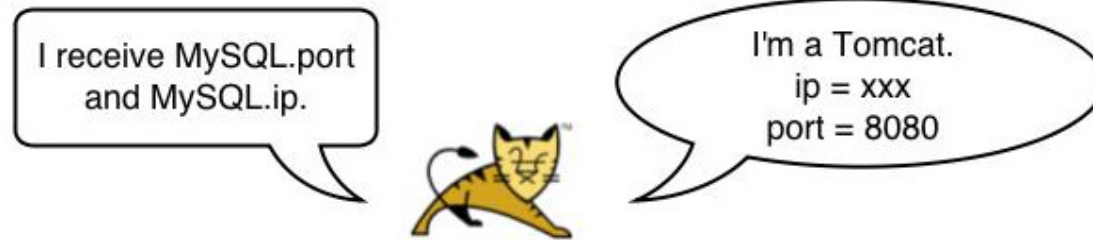
Une fois la VM créée, l'agent récupère son modèle local grâce au DM. Il peut ensuite déployer Tomcat. Avant de le démarrer, il va falloir résoudre ses dépendances. L'agent envoie donc une requête pour demander aux MySQL de se signaler à lui.

Reconfiguration - 2/4



Une instance de MySQL tourne déjà sur une autre machine.
Elle reçoit la requête du nouveau Tomcat et publie alors ses variables.

Reconfiguration - 3/4



Le nouveau Tomcat reçoit les variables du MySQL.
Il peut alors démarrer et publier ses variables.

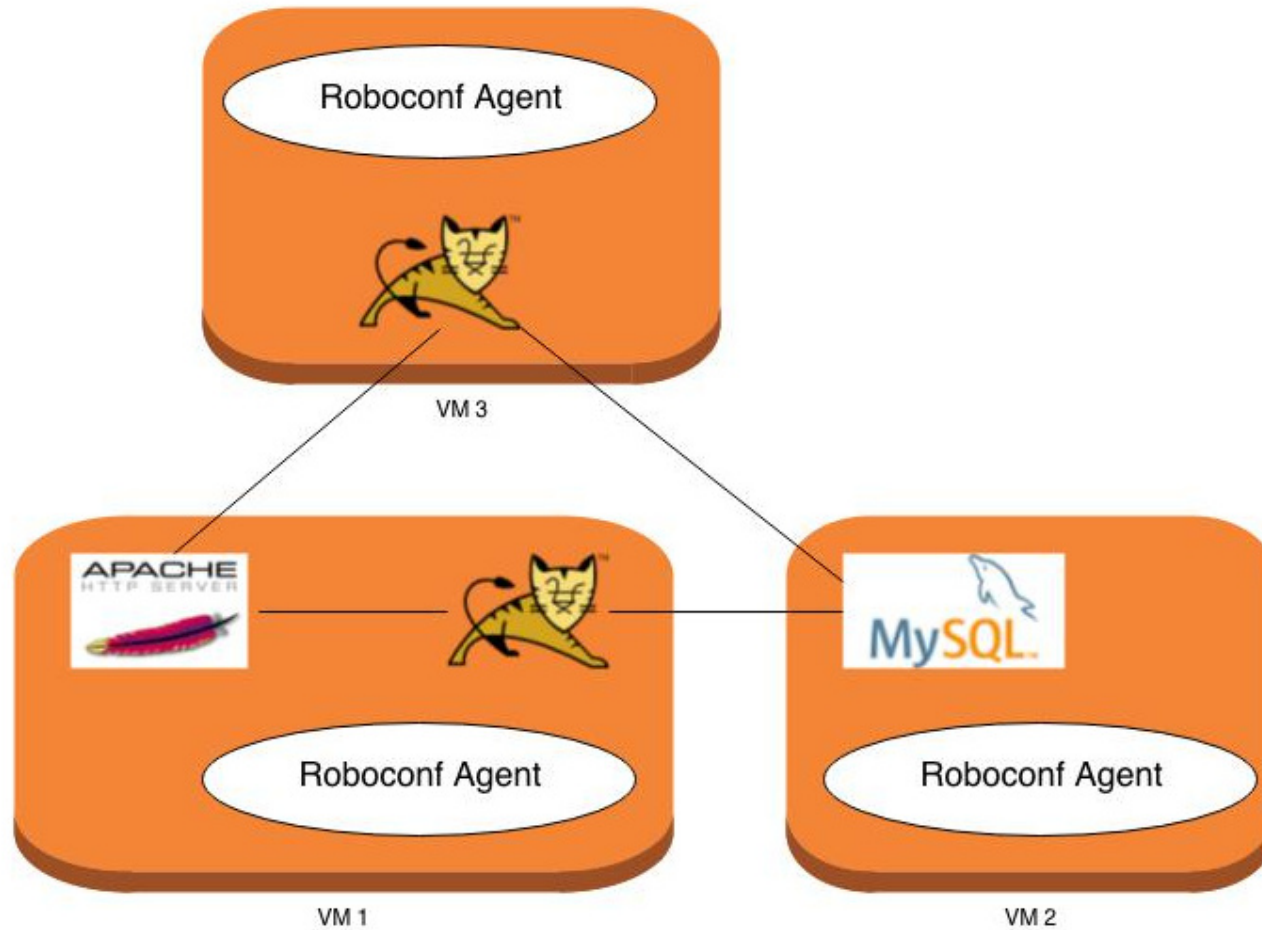
Reconfiguration - 4/4



Il y a déjà un équilibreur de charge Apache qui tourne. Il est informé qu'un nouveau Tomcat vient d'arriver. L'agent Apache exécute alors la recette associée avec l'opération **update**, qui met alors à jour la configuration d'Apache.

Le fait que notre apache utilise un seul serveur Tomcat ou tous dépend de ce que l'on a mis dans la recette. Pour un équilibreur de charges, on a plutôt envie de connaître TOUTES les instances. Pour une base relationnelle, UNE seule instance suffit (les autres peuvent être ignorées).

Résultat du Déploiement



Le serveur Apache peut maintenant répartir les requêtes sur 2 serveurs Tomcat. Les deux utilisent la même base MySQL.

Au Final

Roboconf peut déployer une application distribuée dans **n'importe quel ordre**.

Le parallélisme est poussé au maximum. La communication asynchrone entre agents Roboconf garantit que quoi qu'il arrive, l'application retombera sur ses pattes.



Ce mécanisme est utilisé au déploiement comme dans les phases de reconfiguration (rajout ou retrait de blocs applicatifs).

LAMP est l'exemple basique

Roboconf sait gérer ce cas d'usage.

Toutefois, une solution comme **Juju** (d'Ubuntu) doit aussi s'en sortir, et peut-être d'une manière plus accessible.

En revanche, pour des applications ou des architectures complexes, Roboconf apporte une vraie plus-value pour les équipes. De ce point de vue, l'exemple LAMP est plutôt à voir comme un moyen de comprendre le fonctionnement de Roboconf.

CAS D'USAGE INDUSTRIEL

OpenPaaS

OpenPaaS est une PaaS (Platform as a Service) open source dont l'objectif est d'améliorer la collaboration entre et au sein d'organisations. C'est un projet dans la mouvance des RSE (Réseaux Sociaux d'Entreprise).

Il couvre la création de communautés autour de sujets ou de projets professionnels. Ces communautés bénéficient de nombreux outils, tels qu'une messagerie instantanée, e-mail, agendas, partage de documents, etc.

<http://open-paas.org>

Contraintes Techniques

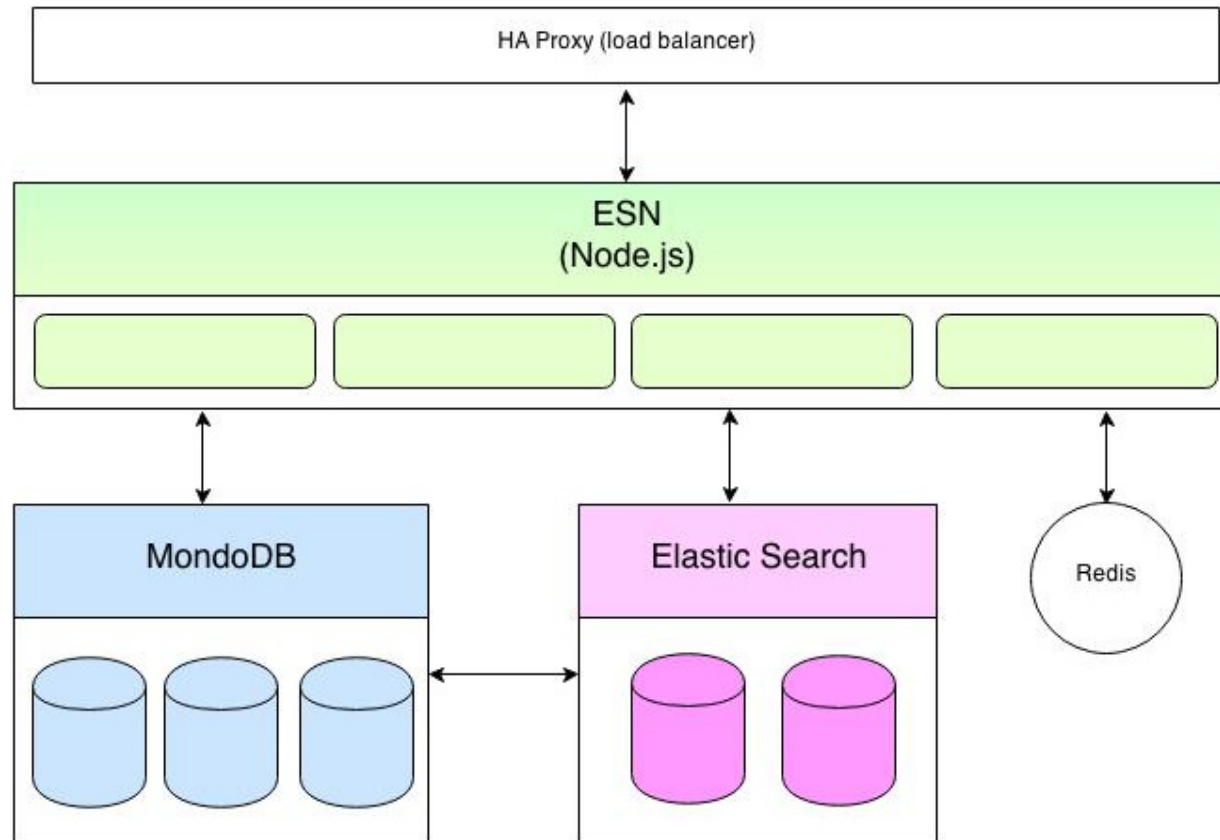
La plate-forme s'appuie sur des technologies web.

La partie serveur est implémentée avec Node.js. La partie cliente est développée avec AngularJS et les projets qui gravitent généralement autour.

Un des impératifs est que cette plat-forme doit pouvoir fonctionner dans le cloud, en bénéficiant des facilités offertes par ce type d'infrastructures. Elle doit notamment pouvoir faire face à des pics de charge, et pouvoir s'adapter avec le temps au nombre grandissant de données stockées. Cette partie repose notamment sur l'utilisation de base de données no-SQL.

D'autres briques logicielles viendront enrichir la plate-forme avec le temps. En résumé, OpenPaaS est une application répartie très complexe qui doit tourner dans le cloud.

Vue d'Ensemble



Pour le moment, OpenPaaS se concentre sur la pile web.
Les parties élastiques incluent l'application du RSE (Node.js), MongoDB et ElasticSearch.

Évolutions Prévues

OpenPaaS devraient bientôt compter d'autres briques logicielles.

- Apache James, un serveur de mails développé en Java.
- Cassandra, une autre base de données no-SQL.
- OpenLDAP et LinID.

Cassandra sera utilisé comme stockage distribué pour les e-mails.
LDAP sera utilisé pour l'authentification et la gestion des permissions.

Ces évolutions viendront rajouter des nouvelles parties élastiques dans la plate-forme. Roboconf permettra de rendre ces changements et leurs gestion aussi simples que possible.

FICHIERS DE CONFIGURATION

Structure de Projet

- descriptor/
 - application.properties
- graph/
 - *.graph
 - 1 répertoire par composant (pour les recettes)
- instances/
 - *.instances

Le graphe est défini dans des fichiers ***.graph**.

Cette définition peut être étalée dans plusieurs fichiers, pour la modularité.

Il en est de même pour les fichiers ***.instances**.

Cette structure de fichiers peut aussi être portée dans Maven, pour une utilisation conjointe du plug-in Maven pour Roboconf.

Fichiers du Graphe - 1/5

Ces fichiers utilisent une syntaxe particulière, inspirée de celle de CSS. Ils sont facilement lisibles, et éditables avec n'importe quel éditeur de texte. Ils sont moins verbeux que XML, et moins sujets aux maladdresses que YAML.

```
# Un composant appelé 'MyComponentName'  
MyComponentName {  
  installer: bash;  
}
```

Les commentaires et les propriétés doivent tenir sur une seule ligne. **installer** est l'extension Roboconf utilisée pour le déploiement, le démarrage, l'arrêt ou la désinstallation d'instances de ce composant.

Chaque composant est associé à un répertoire contenant des recettes. Ces ressources seront utilisées par l'extension Roboconf identifiée par la propriété **installer**.

Fichiers du Graphe - 2/5

Un aspect important des composants sont les imports et les exports. Ce sont des variables qu'un composant expose ou dont il dépend.

```
# WebApplication expose "WebApplication.ip".
WebApplication {
  exports: ip;
  imports: MySQL.ip, MySQL.port;
}

# MySQL expose "MySQL.ip" and "MySQL.port".
MySQL {
  exports: ip, port = 3306;
}
```

Si un composant importe une variable, alors un autre composant doit l'exporter. Une instance de composant ne peut pas démarrer tant que tous ses imports ne sont pas résolus.

Chaque fois qu'une instance de composant apparaît ou disparaît, elle notifie ceux qui importent ses variables. Les instances notifiées peuvent alors prendre les actions adéquates au travers de leurs recettes.

Fichiers du Graphe - 3/5

Il est possible de définir des imports optionnels.

Des imports optionnels signifient qu'un composant peut démarrer sans que ces imports n'aient été résolus. Toutefois, il sera averti si des instances les exportant apparaissent ou disparaissent.

```
# Un noeud de cluster
ClusterNode {
  exports: ip, port = 18741;
  imports: ClusterNode.ip (optional), ClusterNode.port (optional);
}
```

Quand une instance de composant est démarrée, et qu'un de ses imports n'est plus résolu, Roboconf l'arrête et le met dans un état appelé **starting**.

Lorsque la dépendance sera à nouveau résolue, l'instance sera automatiquement redémarrée.

ip est une variable particulière dont la valeur est mise automatiquement par Roboconf. Toute autre variable exportée doit avoir une valeur par défaut.

Fichiers du Graphe - 4/5

Les imports et les exports ont trait aux relations de type « exécution ». Les relations de type « conteneur » sont gérées via le mot-clé **children**.

```
# Tomcat et MySQL sont d'autres composants
VM_Tomcat {
    children: Tomcat;
}

VM_MySQL {
    children: MySQL;
}

VM_All {
    children: Tomcat, MySQL;
}
```

children liste les composants qui peuvent être déployés sur une instance de ce composant. On peut ainsi être très flexible ou bien très contraignant.

Fichiers du Graphe - 5/5

Des composants peuvent parfois partager un certain nombre de propriétés.
Il est possible de regrouper des propriétés au sein d'une **facette**.

```
# Facettes
facet VM {
  children: deployable;
  installer: target;
}

facet deployable {
  installer: puppet;
  exports: ip;
}
```

```
# Composants
VM_EC2 {
  facets: VM;
}

VM_Azure {
  facets: VM;
}

Tomcat {
  facets: deployable;
}
```

Recettes - 1/3

Les recettes sont des ressources utilisées par des extensions de Roboconf pour gérer le cycle de vie d'instances de composants.

```
VM_EC2 {  
    installer: target;  
}  
  
VM_Openstack {  
    installer: target;  
}
```

Ce graphe signifie que l'on va trouver 2 sous-répertoires appelés **VM_EC2** et **VM_Openstack**. Ces composants utilisent l'extension **target**, qui gère la création et la destruction de VM. Cette extension nécessite un fichier **target.properties**.

Il y aura donc un répertoire avec les paramètres pour EC2.
Et il y en aura un autre avec les paramètres pour Openstack.

Recettes - 2/3

Voici un exemple de fichier `target.properties`.

```
# Propriétés spécifiques à Amazon Web Services
target.id          = ec2
ec2.endpoint       = eu-west-1.ec2.amazonaws.com
ec2.access.key     = YOUR_EC2_ACCESS_KEY
ec2.secret.key    = YOUR_EC2_SECRET_KEY
ec2.ami            = Your AMI identifier (ami-...)
ec2.instance.type = t1.micro
ec2.ssh.key        = Your SSH Key
ec2.security.group = Your Security Group
```

Recettes - 3/3

L'extension associée à l'identifiant **target** est en charge des machines (VM, serveur, device...). Sa configuration dépend de la cible de déploiement.

L'extension associée à l'identifiant **bash** gère le cycle de vie d'une instance de composant à l'aide de scripts Bash (deploy, start, stop, undeploy et update). Ces scripts doivent être idem-potents.

L'extension associée à l'identifiant **puppet** gère le cycle de vie d'une instance de composant au travers d'un module Puppet. Le nom du module doit démarrer par **roboconf_**.

Ce sont les extensions actuellement disponibles.
D'autres peuvent être créées au besoin (Chef, ANT, extension personnalisée...).

Instances - 1/2

Les composants définissent des règles et des relations (comment).
Les instances définissent ce qui tourne vraiment (quoi et où).

Contrairement au graphe, qui désigne un ensemble de graphes, les définitions d'instances sont des arbres.

```
# Déploiement d'une VM "VM1" sur EC2
instance of VM_EC2 {
  name: VM1;
  instance of MyApp {
    name: MyApp_on_EC2;
  }
}

# Déploiement d'une VM "VM1" sur Openstack
instance of VM_Openstack {
  name: VM2;
  instance of MyApp {
    name: MyApp_on_Openstack;
  }
}
```

Instances - 2/2

Outre son nom et son composant, une instance peut définir de nouvelles propriétés et/ou surcharger des propriétés héritées de son composant.

Le fait de rajouter de nouvelles propriétés est inutile du point de vue des exports. Mais cela peut s'avérer utile dans les recettes.

```
# Le composant
Tomcat {
  exports: ip, port = 8080;
}
```

```
# Une instance
instance of Tomcat {
  name: Tomcat;
  port: 8080;
  log_level: INFO; # utilisé dans la recette
}
```

Descripteur d'Application

Le descripteur d'une application est un simple fichier de propriétés. Celui-ci contient des méta-données sur le projet lui-même.

```
# Descripteur d'application pour Roboconf
application-name = MyApp
application-qualifier = sample
application-description = Une application pour l'exemple

application-namespace = net.robconf
application-dsl-id = roboconf-1.0

graph-entry-point = main.graph
instance-entry-point = initial-model.instances
```

OUTILS

Outillage

Roboconf a été pensé pour être aussi simple que possible à l'utilisation. Cette question de la simplicité se retrouve dans la syntaxe des fichiers de configuration.

En dépit de cette simplicité, des outils sont toujours un complément agréable pour des utilisateurs.



x Creation Wizards
x Text Editor



x Web Administration
x CLI Administration?

maven

x Validation
x Packaging
x Testing



x Report Generation
For Project & Support

L'Administration Web - 1/2

L'administration web est, comme son nom l'indique, une application web (AngularJS) qui fournit une interface graphique pour l'API REST du DM.



☰ Applications ⚙ Settings

Applications

This page lists all the applications handled by the same Deployment Manager.
The deployment manager is hosted at <http://localhost:8080/roboconf-dm-webapp/rest>.

[➕ New Application](#)

Linagora RSE :: demo



The Open PaaS project aims at developing a PaaS (Platform as a Service) technology dedicated to enterprise collaborative applications deployed on hybrid clouds (private / public). Open PaaS is a platform that allow to design and deploy applications based on proven technologies provided by partners such as collaborative messaging system, integration and workflow technologies that is extended in order to address Cloud Computing requirements.

L'Administration Web - 2/2

Cette console permet de donner des ordres aux agents Roboconf et d'avoir une vue globale des différentes parties applicatives.

Linagora RSE

This page lists all the instances of the **Linagora RSE** application. Here, you can control their life cycle.

Global Actions ▾

VM HAProxy	deployed and started
HAProxy is deployed and started.	
VM Mongo replica set	deployed and started
Mongo replica set node is deployed and started.	
VM Mongo replica set 2	not deployed
Mongo replica set node 2 is not deployed.	
VM Node-RSE 1	deployed and started
RSE instance is deployed and started.	
VM Node-RSE 2	not deployed
RSE instance is not deployed.	
VM Redis	deployed and started
Redis is deployed and started.	

HAProxy

Instance Path: /VM HAProxy/HAProxy
Instance Status: deployed and started
Component: HAProxy

```
graph TD; NotDeployed([Not Deployed]) --> Stopped([Stopped]); Stopped --> NotDeployed; Stopped --> Started([Started]); Started --> Stopped;
```

The following actions can be undertaken.

■ Stop this Instance	⚠ Undeploy this Instance
----------------------	--------------------------

Description
The instance is started. It can be only be stopped.
The life cycle of this instance is handled by Roboconf's **logger** installer.

Plug-in Maven

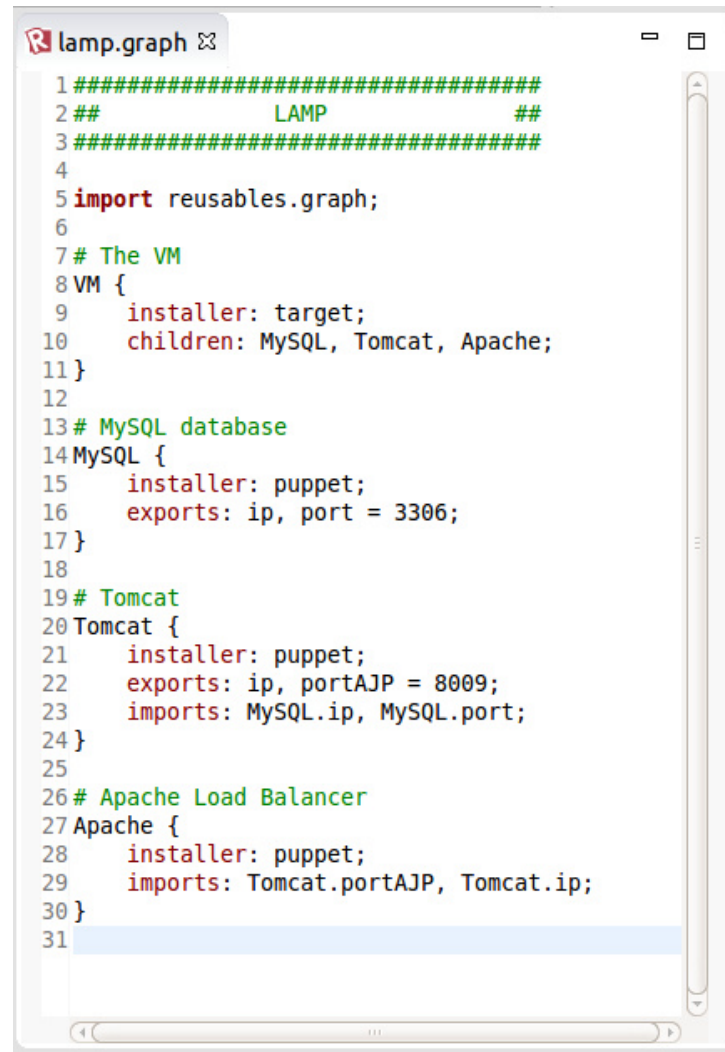
Roboconf possède son propre plug-in Maven.
Il permet...

- ... de valider ses fichiers de configuration (graphe, instances)...
- ... de conditionner son projet sous la forme d'une archive déployable...
- ... d'insérer des propriétés Maven dans ses fichiers de configuration.

Des travaux sont en cours pour faciliter les tests et la réutilisation de recettes. D'autres outils devraient suivre.

Plug-in Eclipse

Roboconf a aussi son propre plug-in Eclipse.
Il fournit un assistant de création et des éditeurs dédiés.



```
lamp.graph
1 #####
2 ##          LAMP          ##
3 #####
4
5 import reusables.graph;
6
7 # The VM
8 VM {
9     installer: target;
10    children: MySQL, Tomcat, Apache;
11}
12
13 # MySQL database
14 MySQL {
15     installer: puppet;
16     exports: ip, port = 3306;
17}
18
19 # Tomcat
20 Tomcat {
21     installer: puppet;
22     exports: ip, portAJP = 8009;
23     imports: MySQL.ip, MySQL.port;
24}
25
26 # Apache Load Balancer
27 Apache {
28     installer: puppet;
29     imports: Tomcat.portAJP, Tomcat.ip;
30}
31
```



RESOURCES

Téléchargement & Documentation

roboconf.net

Code Source

github.com/roboconf

Contact & Questions

twitter.com/roboconf

github.com/roboconf/roboconf/issues