Introduction to Roboconf

By the Roboconf team / @Roboconf

PROJECT

Roboconf

- Launched in 2013
- Open source (Apache v2)
- Hosted on GitHub

Roboconf is based on a prototype developed by Université Joseph Fourier (Grenoble, France). It is now being industrialized by Linagora, a French Software company.

In Few Words

Roboconf is a solution based on asynchronous exchanges to deploy and reconfigure distributed applications.

It translates concepts from component models (like OSGi) to Software deployment and management.

Target Users

Roboconf is intended for project teams with requirements about dynamic deployments and where scalability matters.

The ideal user profile is a **DevOp**.
Someone involved in the project development but with a focus on production issues.

Roboconf Features - 1/2

Deployment

Installs and configures Software applications.

Deployment Targets

Allows deployment over various targets: cloud infrastructures (IaaS), devices, on-premise hosts... Hybrid deployments are supported.

Roboconf Features - 2/2

Reconfiguration

Application parts notify each others, which allow them to reconfigure themselves through scripts. Roboconf handles applications life cycle.

Monitoring

Monitoring support is in progress.

Covered Use Cases

- Cloud deployment
- Hybrid deployments
- Highly dynamic environments (with lots of reconfiguration)

Roboconf can deploy any kind of Software.
It suits well for distributed applications. There is no constraint on languages or frameworks. If you can script it, you can make it automatic with Roboconf.

Large-scale deployments are not well covered (yet).

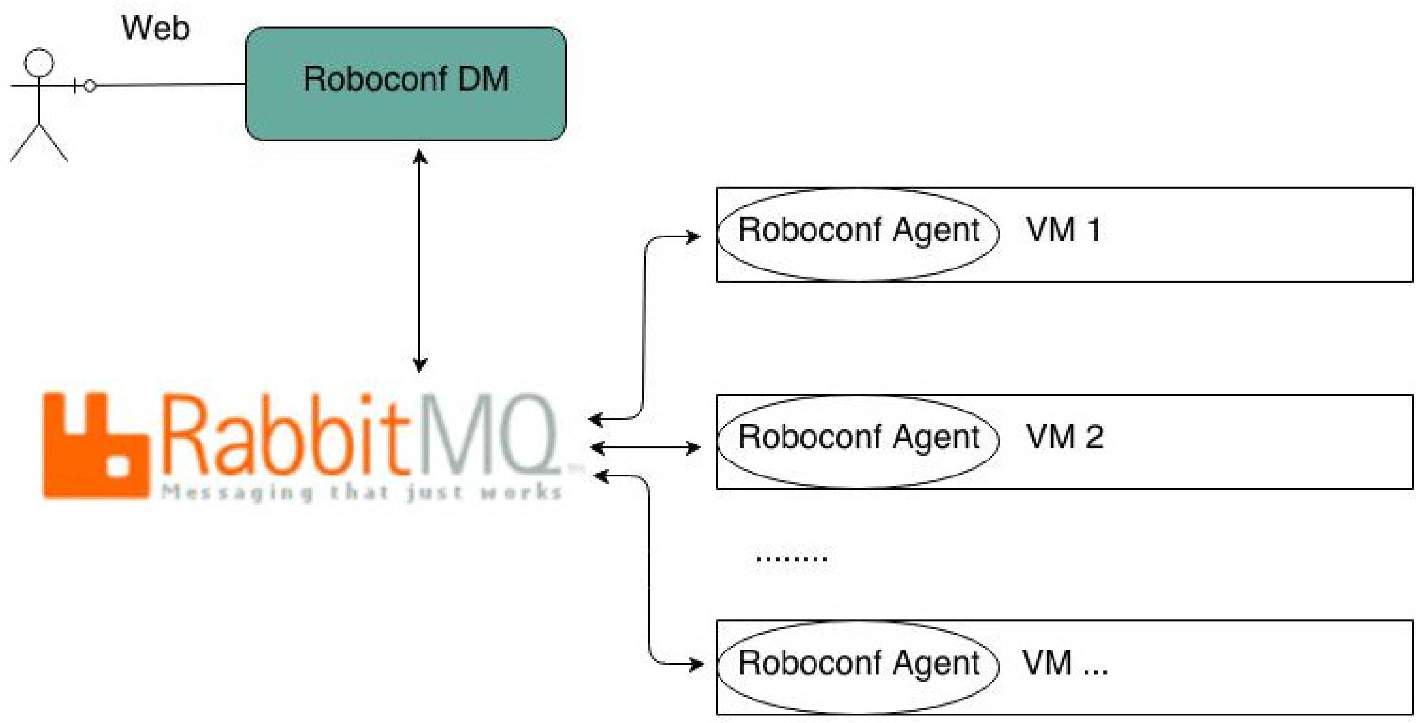
Roboconf...

... does not reinvent the wheel.

Thanks to a plug-in approach, it is possible to use Bash scripts and Puppet modules. You could also use other solutions like Chef or CFengine, provided you write a plug-in for this.

Life cycle steps can thus be handled through robust solutions.
Roboconf only plugs dynamics behind them.

ARCHITECTURE



Roboconf relies on 3 elements: the DM, agents and a messaging server (RabbitMQ). VM designate Virtual Machines or any kind of machine (server, device...).

The DM

The Deployment Manager (DM) is the application in charge of driving deployments. It is the interface through which one can pass requests (deploy this, start that...).

It provides a set of REST services.
It is generally packaged with a web application (roboconf-web-administration) that acts as a user-friendly front-end for the REST API.

Depending on the invoked REST operation, the DM will either...

- ... interact with a IaaS manager to create or destroy VM.
- ... communicate with Roboconf agents through a messaging server.

The Agents

Every machine on which Roboconf must deploy or manage something must have an agent installed. This agent is in charge of listening...

- ... instructions from the DM.
- ... notifications from other agents.

Depending on what it receives, an agent will execute recipes.
Recipes are associated with plug-ins (Puppet, Bash...). This results in a change, either in the life cycle or in the configuration files of an application.

There is one agent per machine.
But it can install and manage several elements on it. Installed applications are not aware of Roboconf (=> middleware).

The Messaging Server

The messaging server is the medium used to exchange information between...

- ... the DM and agents.
- ... an agent and other agents.

The current messaging server is RabbitMQ.
It is a robust, open source, message broker that implements AMQP and that supports clustering.

Tips about the DM

The DM can be deployed anywhere.
The good practice is generally to run it locally (in the information system) rather than putting it on a public server. This way, it skips a lot of work to secure it.

The DM was designed to be started when it is required.
It means you can stop it when you do not use it. Its state is persisted in consequence on a file system. It also makes procedures quite simple to restore it after a crash.

Tips about RabbitMQ

The messaging server is the key point so that everything works.
It must be reachable from the DM and all the agents. A public location is thus required.

Roboconf uses authentication mechanisms provided by RabbitMQ to secure access to the messaging server.

Tips about Agents

The most convenient use case with Roboconf is when VM are created in a cloud infrastructure (IaaS). The DM creates them from the IaaS image catalog. So, to have an agent installed and configured on every machine, a virtual appliance has to be created and stored in the image catalog.

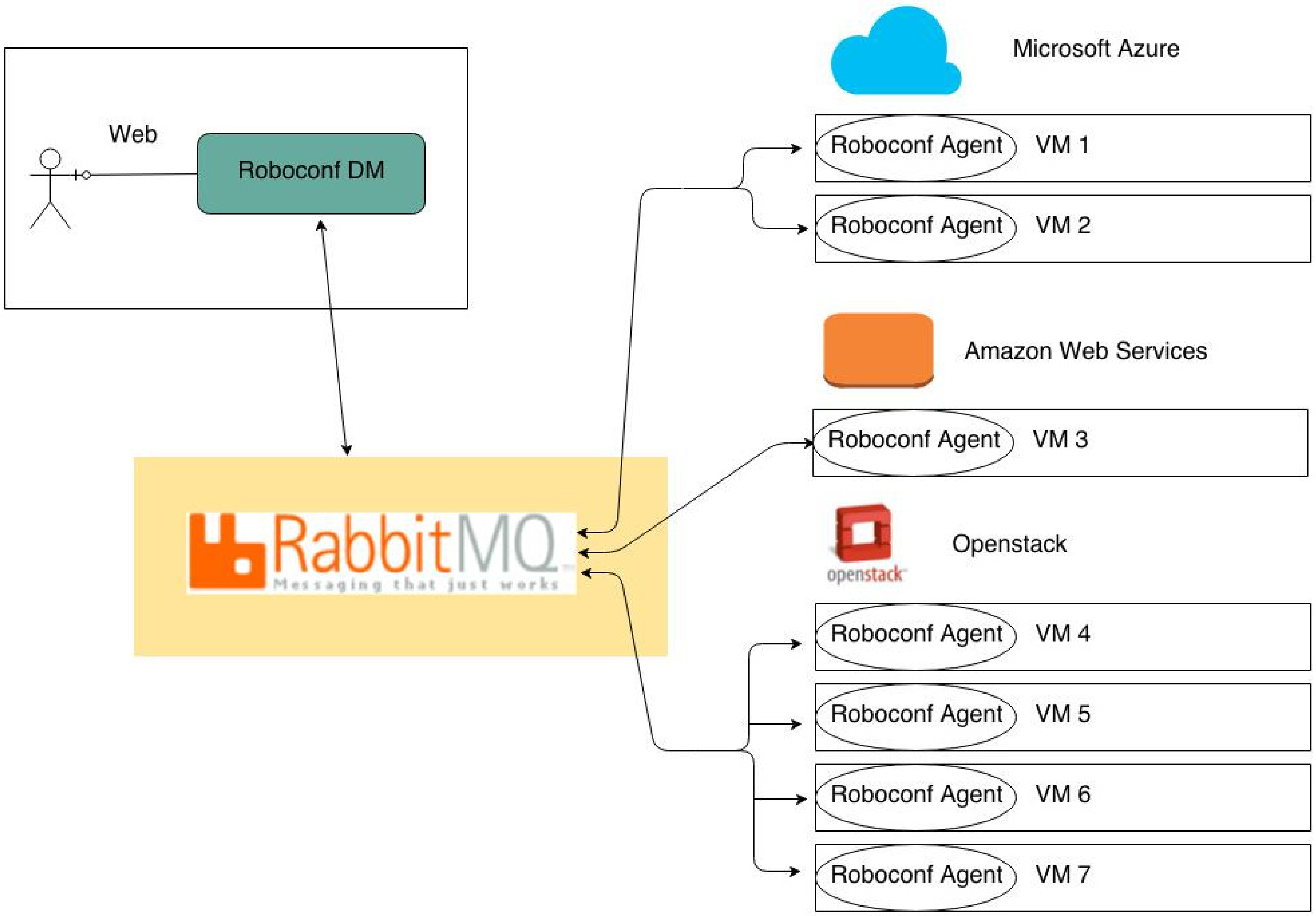
For other kind of hosts, one has to use other solutions to install and configure agents. Solutions like Puppet, or scripts that do SSH are suitable.

Most of the information an agent needs is supplied by the DM at runtime. This keep their configuration minimal.

Multi-IaaS

Multi-target and hybrid deployments are supported.
This can be used for critical periods (load increase) or for migration purpose.
It can also serve strategic objectives (like data security and privacy issues).

The next slide illustrates this with **some** cloud infrastructures.
It only shows interaction between Roboconf parts, and not between the parts of the applications that Roboconf deploys.


Web
Roboconf DM
RabbitMQ
Messaging that just works
Microsoft Azure
Roboconf Agent VM 1
Roboconf Agent VM 2
Amazon Web Services
Roboconf Agent VM 3
Openstack
openstack
Roboconf Agent VM 4
Roboconf Agent VM 5
Roboconf Agent VM 6
Roboconf Agent VM 7

CONCEPTS

The Graph

The graph defines Software elements and their relations.
Software elements include virtual machines, application servers, application packages... whatever. They are called **components**.

Two kinds of relations are considered.

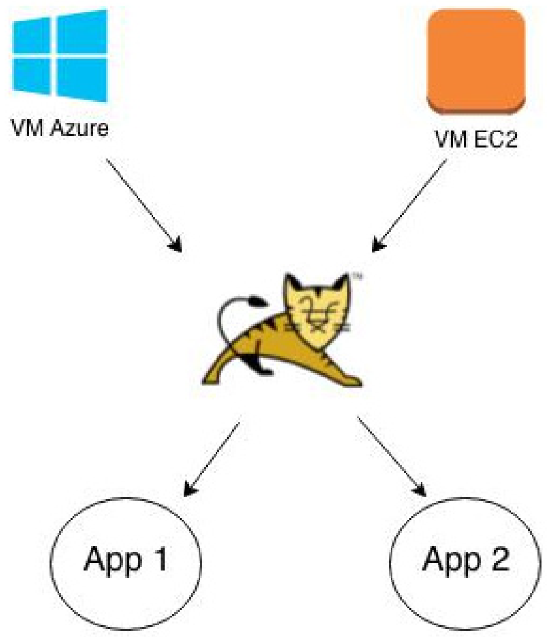
Containment

A component has to be deployed onto another one.
Example: a war onto a web application server.

Runtime

A component needs another one to run.
A component provides a feature another component may need.
Example: a war application needs a MySQL database.

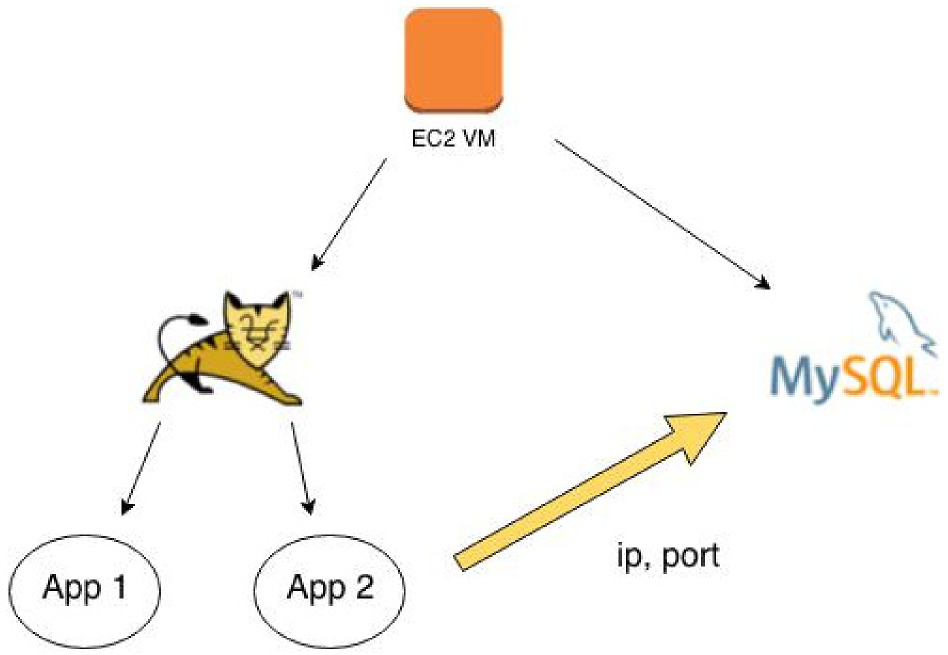
Containment Example



We can deploy Tomcat either on an Amazon VM, or on an Azure VM. And we can deploy given applications only on Tomcat. If we wanted to deploy a third web application, we would have to update the graph.

Containment can be recursive. Tomcat is here both contained and a container.

Runtime Example



Here shows a mix of containment relations with a runtime dependency. In this graph, **App 2** depends on a MySQL database, which is not the case of **App 1**. It means Roboconf cannot start **App 2** until it has the **ip** and **port** of a MySQL database.

Benefits of the Graph

For a given Software component...

- We know what can be deployed on it.
- We know on what it can be deployed.
- We know which other components it depends on.

Practical Use

When we want to deploy a new Software component...

- ... we can deploy it on an existing container.
- ... we can create its container at the same time.

The Graph(s)

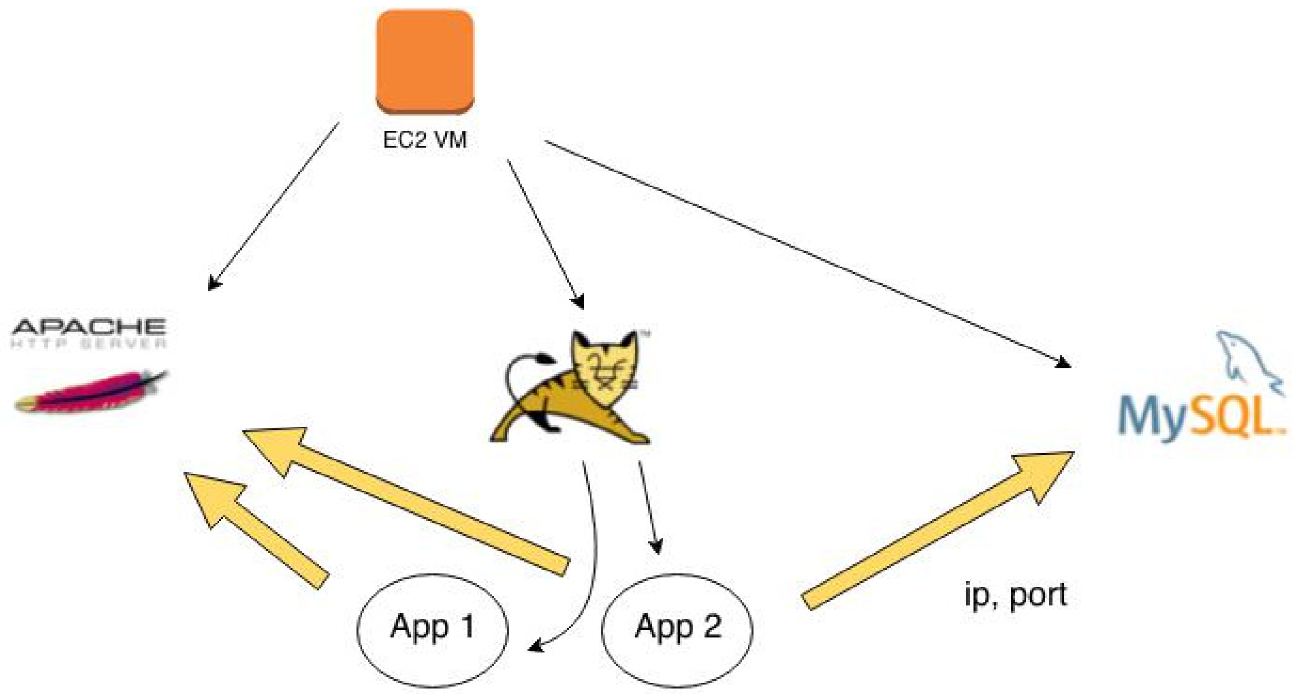
The graph is a structure with two levels of reading (containment and runtime relations). It is a set of rules that will drive deployments and reconfiguration.

But unlike what the previous slides let think, there is not only one graph. Indeed, several graphs can be defined in the configuration files.

Thus, it is possible to define pseudo-appliances.
It means deployments can be made very flexible or very constrained.

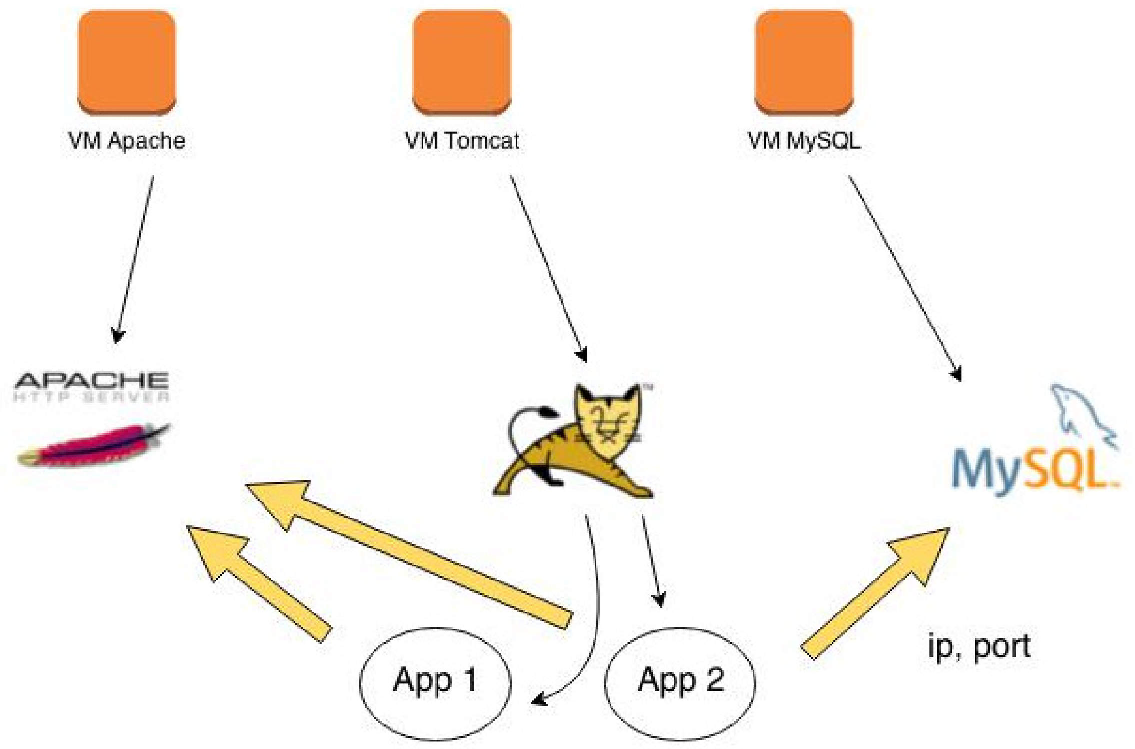
This is useful because those that develop an application or plan its deployment are not those who effectively deploy and manage it in production. Constraining deployment of scalable parts reduces the risk of errors by the production team.

Lax Graph



We here consider the 3 main applications (Apache load balancer, Tomcat and MySQL) can be deployed over a same VM, or at least, on VM of the same « type ».

Appliance-like Graphs



Here, we consider each of the 3 main applications must be deployed on their own VM type. It means we cannot deploy different applications on a given VM.

Every VM type can have its own template and configuration (tailored CPU, RAM, storage, operating system, network configuration...).

Recipes

Every component in the graph is associated with a set of recipes.
More exactly, every component in the graph is associated with a plug-in (« installer »).

A component associated with the **Puppet** installer will use a Puppet module as its recipe. A component associated with the **Bash** installer will use a set of Bash scripts.

Recipes define the actions to undertake when an administration decision was sent. It matches life cycle steps: deploy, start, stop and undeploy.

There is also a step called **update**.
It is invoked automatically by Roboconf on reconfiguration. This is discussed farther.

Instances

Instances means « instances of components from the graph ».
The graph is a set of rules. It could be seen as a meta-model for instances. So, instances comply with what was defined in the graph.

Another comparison could be classes and instances from Object-Oriented Programming.

Instances inherit properties from their component.
They can override them, and even define new ones (local scope). Their relations with other instances are ruled by the graph(s).

An instance is associated with a single component.

Application Meta-data

A Roboconf project is called an application.
It comes with the definition of graphs and predefined instances.

But it also has some information about the application itself.
Name, description, qualifier (version)...

EXAMPLE: LAMP

The LAMP Example

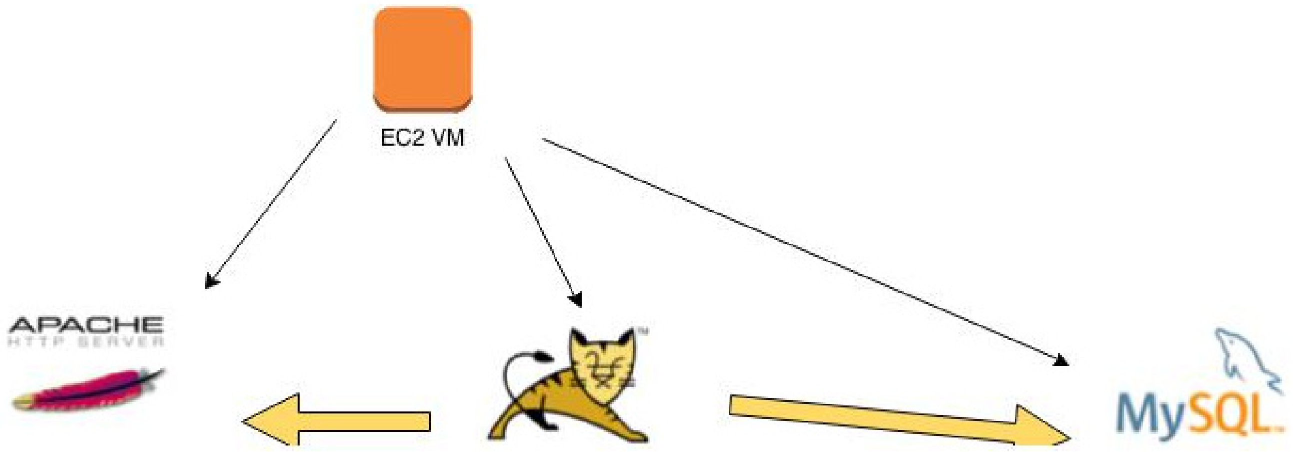
This is the classical example to introduce Roboconf.
It consists in deploying a standard stack made up of...

- ... a load balancer (here, an Apache server with modJK)...
- ... an application server (Tomcat)...
- ... and a database (MySQL).

It illustrates both deployment and reconfiguration.
We use a very simple graph where web applications are installed with Tomcat (they are not in the graph, but part of the recipes).

We also assume the deployment is performed on Amazon Web Services.

The Graph



The Apache load balancer, Tomcat and MySQL can be deployed over a VM. Tomcat needs a MySQL database. And the load balancer must be notified every time we deploy an instance of of Tomcat.

Tomcat is not the stand-alone server.
The recipe behind deploys both the server and at least a web application in it.
This is to keep things simple for the example.

Instances Definition

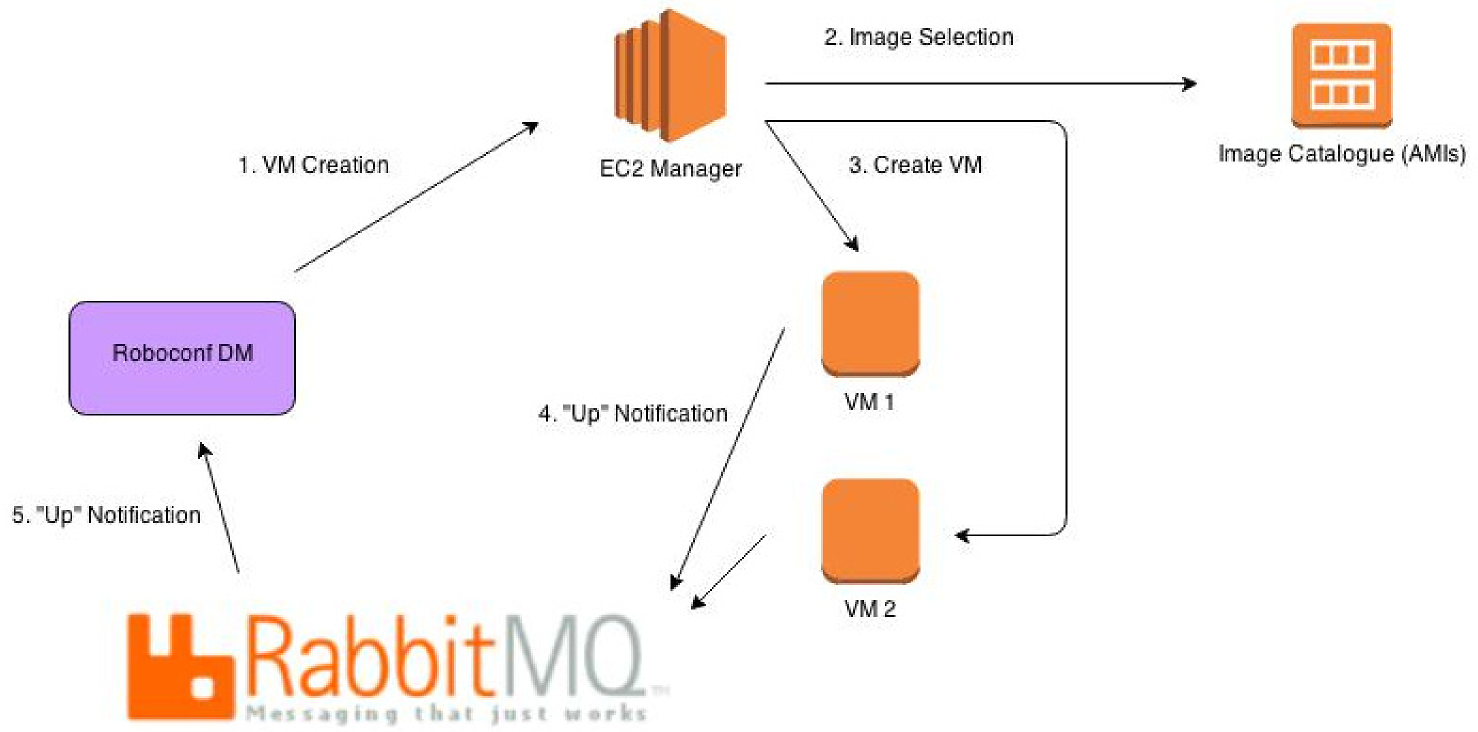
We have our graph.
Let's assume we have the following set of instances in our model.

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

- VM 2 (type: EC2 VM)
    - MySQL (type: MySQL)

This is just a model, something we define in our configuration files.
Let's see what happens when we **deploy all**.

VM Creation

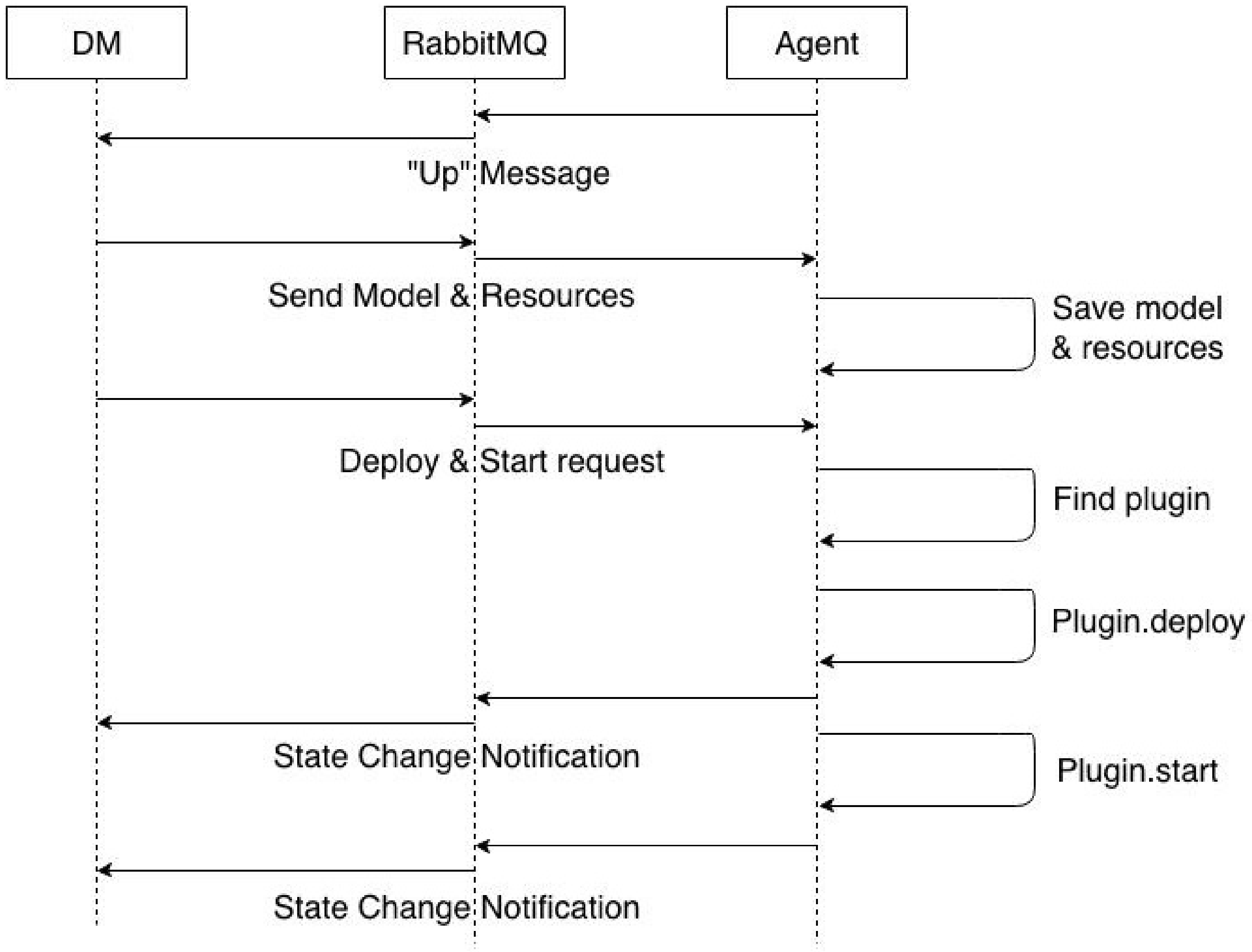


The DM sends one creation request per VM.
The request contains the ID of the image template to instantiate, as well as the expected *flavour* (CPU, RAM...) for the VM.

The virtual image contains an OS and a Roboconf agent with its configuration (RabbitMQ's IP...). Once the VM is up, the agent notifies the DM it is running.

DM and Agent Interactions

Here is what happens when an agent is « up ».



The special case of « Start »

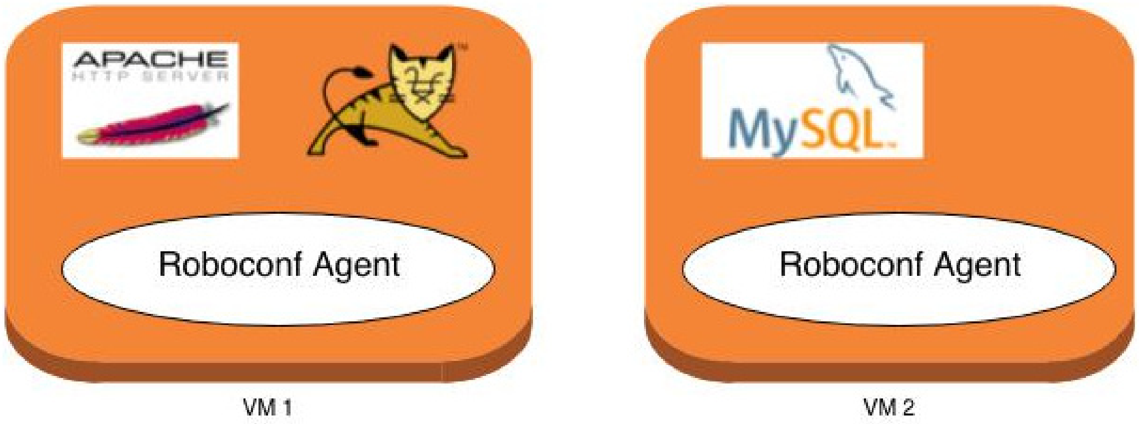
An agent manipulates instances (associated to components).
An agent uses plug-ins to update the life cycle of an instance.

But « plugin.start » does not start an instance.
It **tries** to start it. Before executing it, Roboconf checks all the runtime dependencies. If they are all satisfied, the plug-in can start.

Otherwise, the instance will remain in **starting** until its dependencies are all resolved.

Before Synchronization

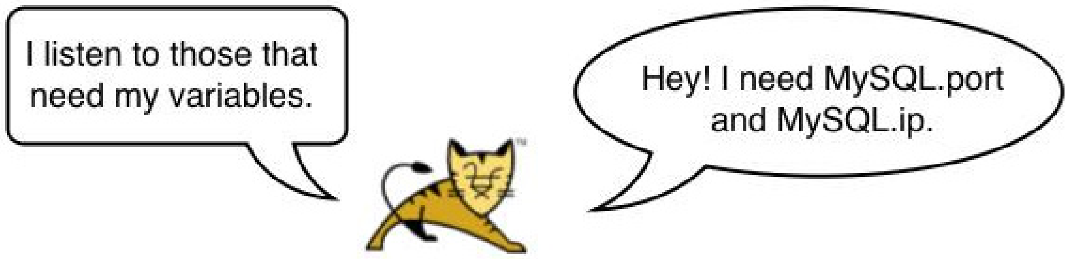
Let's get back to our example. So far, the VM have been created. Apache, Tomcat and MySQL have been deployed. But Tomcat does not know where MySQL runs. And Apache does not know where web applications are deployed.



In fact, while deployments progressively complete here and there, agents exchange information between them through RabbitMQ.

Variable Exchange - 1/6

Let's **assume** that Tomcat is the first deployed.


I listen to those that need my variables.
Hey! I need MySQL.port and MySQL.ip.

First, the local agent starts listening those that need Tomcat variables. Then, it requests the variables it needs (MySQL). It will not be able to start until it has these variables. Since it is alone, nobody will answer its request.

Variable Exchange - 2/6

Let's **assume** Apache is deployed right after.

I listen to those that need my variables.
APACHE HTTP SERVER
Hey! I need Tomcat.port and Tomcat.ip.

First, the local agent starts listening those that need Apache variables. Then, it requests the variables it needs (Tomcat). It will not be able to start until it has these variables.

Variable Exchange - 3/6

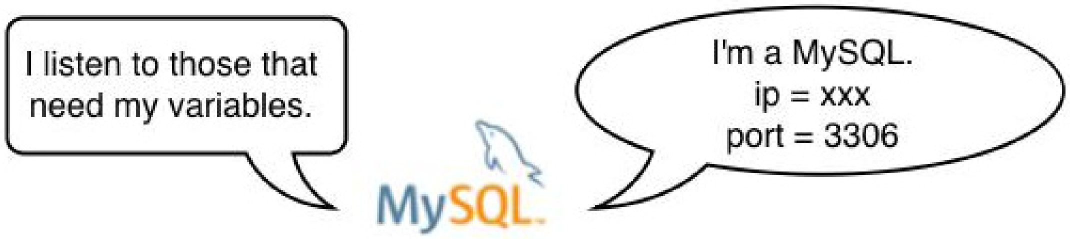
The agent in charge of Tomcat receives the request sent by the Apache agent.

Need Tomcat.port and Tomcat.ip?

But it is not started, it is missing variables.
So, it cannot publish its variables.

Variable Exchange - 4/6

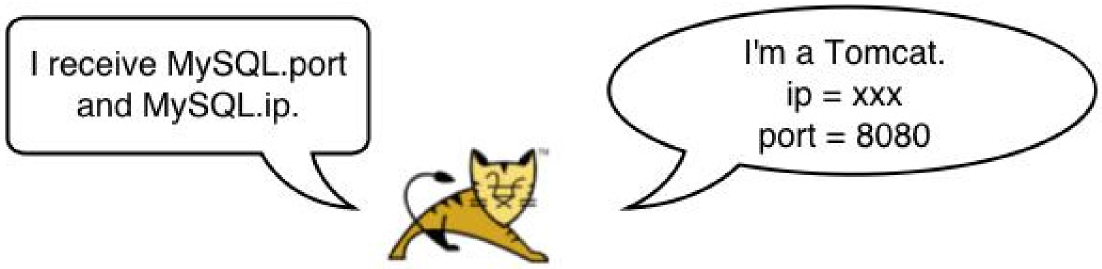
Now, MySQL is up.
The graph states that the MySQL component / type does not have any dependency. So, it can directly start.



When it starts, it listens to those that could need it. And it publishes its variables.

Variable Exchange - 5/6

Tomcat (its agent) gets notified about MySQL.
All its dependencies are resolved and it can start.


I receive MySQL.port and MySQL.ip.
I'm a Tomcat. ip = xxx port = 8080

So, it can publish its variables.

Variable Exchange - 6/6

Eventually, Apache is notified about the Tomcat variables.
It can now start.

I receive Tomcat.port and Tomcat.ip.
APACHE
HTTP SERVER

Since it does not export any variable, it publishes nothing.

Parallel Deployments

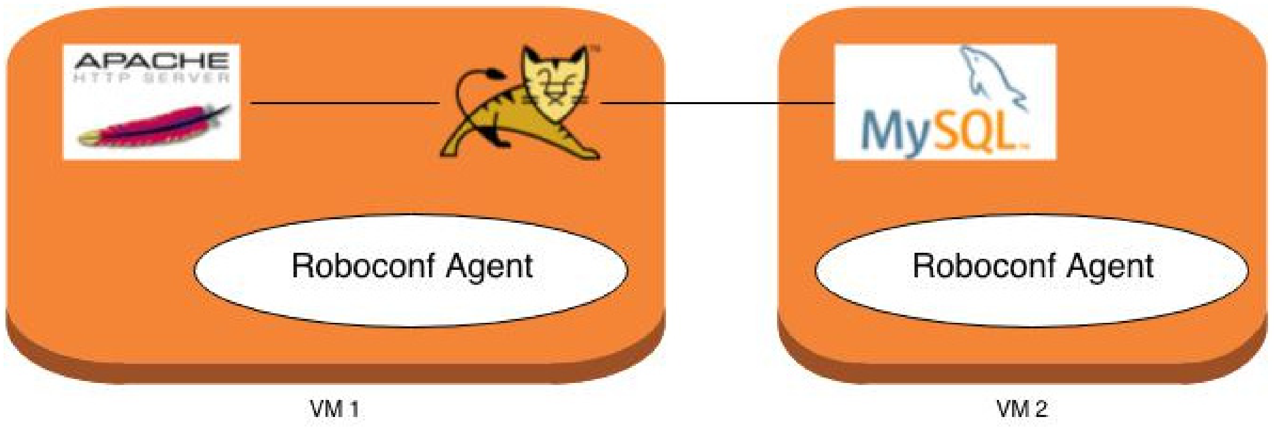
Roboconf parallelizes everything it can.
Here, two flows were executed in parallel.

- Create VM 1, deploy Apache and Tomcat.
- Create VM 2 and deploy MySQL.

While things were progressing, information was exchanged between agents, so that every instance could resolve its dependencies to start.

No matter in which order deployments are performed, Roboconf guarantees everything will get synchronized and running at the end. This is achieved through asynchronous exchanges.

Resulting Deployment

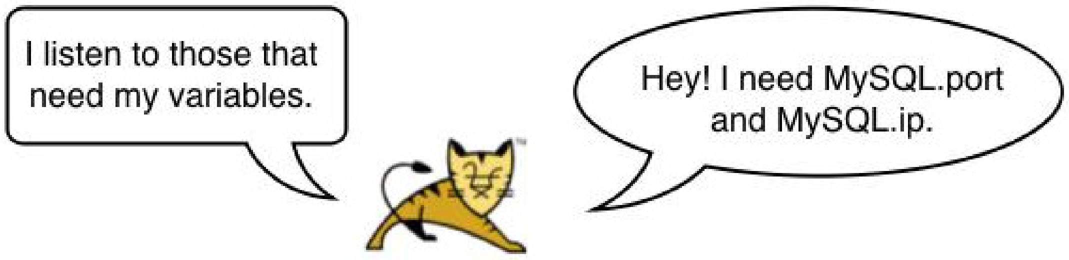


Here is what we get at the end.
Let's now try to add a new Tomcat server.

In our model, we define a new set of instances that we will deploy and start.

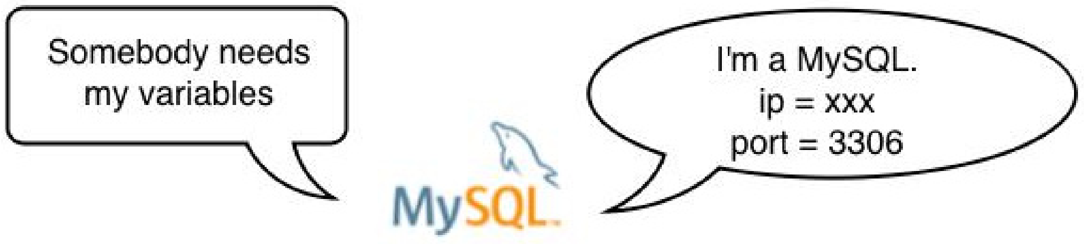
- VM 3 (type: EC2 VM)
  - Tomcat 2 (type: Tomcat)

Reconfiguration - 1/4


I listen to those that need my variables.
Hey! I need MySQL.port and MySQL.ip.

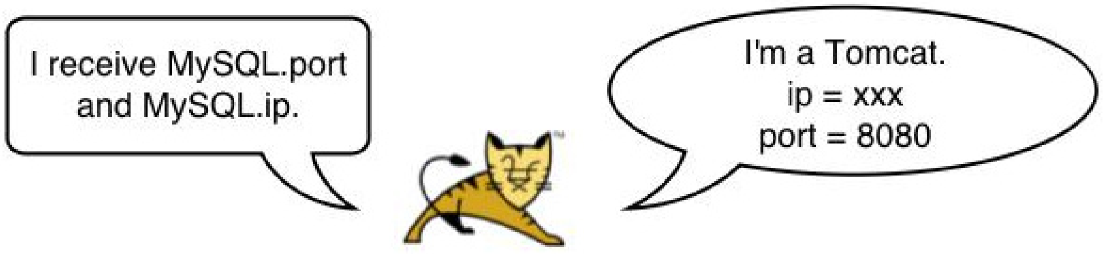
Once the new VM is created, the agent gets its model and deploys Tomcat. Before starting it, it needs its dependencies.

Reconfiguration - 2/4


Somebody needs my variables
I'm a MySQL. ip = xxx port = 3306
MySQL

An instance of MySQL is already up and running.
It receives the request from the new Tomcat and publishes its variables.

Reconfiguration - 3/4


I receive MySQL.port and MySQL.ip.
I'm a Tomcat. ip = xxx port = 8080

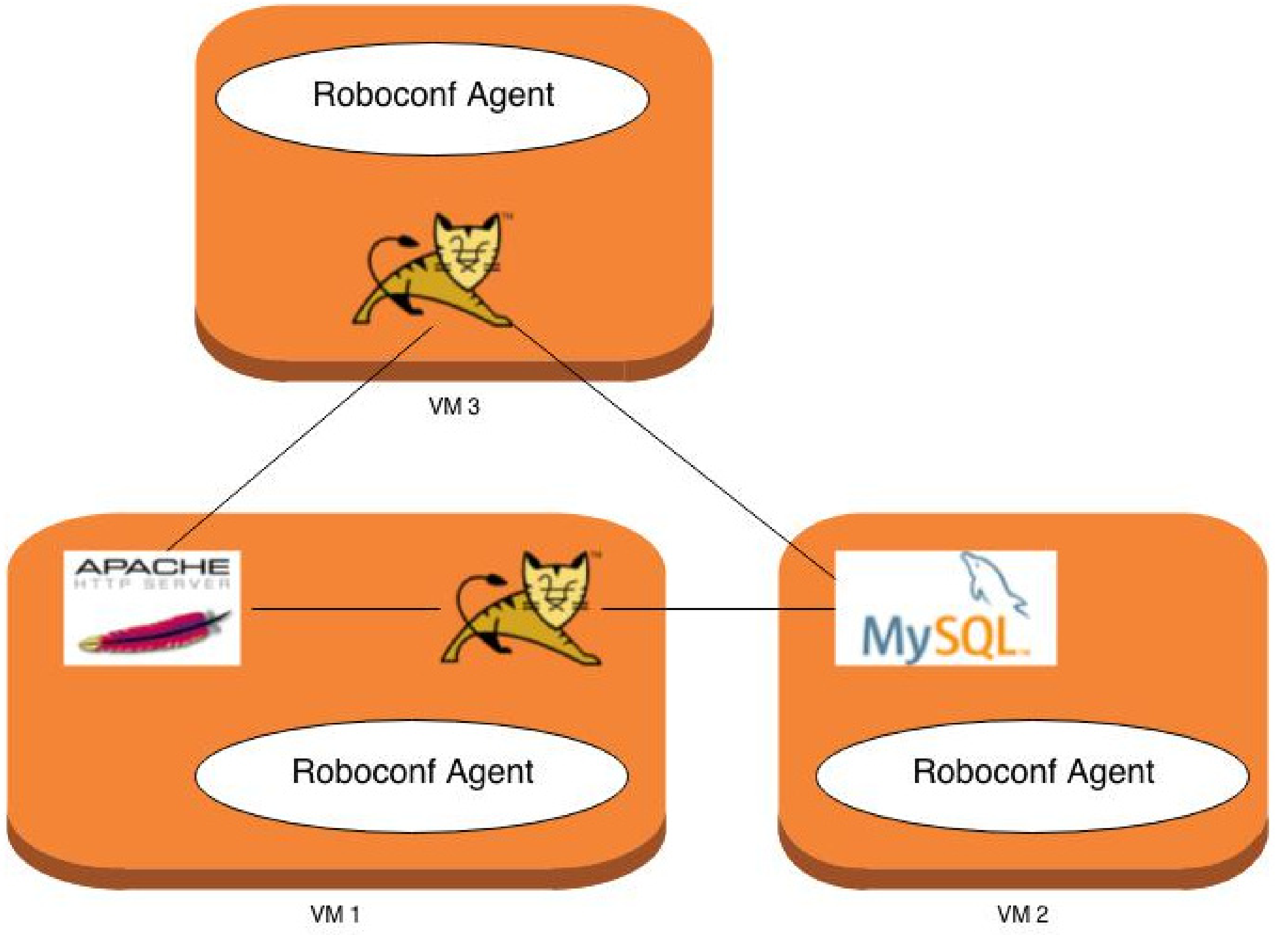
The new Tomcat gets the MySQL variables.
It can thus start and publish its own variables.

Reconfiguration - 4/4

An Apache load balancer is also up and running.
It is notified a new Tomcat is here. The associated agent executes the **update** recipe, which adds the new Tomcat in the load balancer configuration.

Using one or all the Tomcat instances is specified in the recipes.
For a load balancer, we need to know ALL the instances. For a relational database, we need to ONE instance (others are ignored).

Resulting Deployment



The Apache load balancer can now dispatch requests to 2 Tomcat servers.
Both rely on the same database.

Conclusion

Roboconf can deploy a distributed application **in any order**.

Parallelism is pushed to the limits. The asynchronous exchanges between agents guarantee that no matter what, the application will *land on its feet*.

This mecanism is used at deployment time as well as during reconfiguration phasis (addition or removal of application parts).

LAMP is the basic example

If one has to deploy a real LAMP stack, Roboconf fits.
However, a solution like Juju (from Ubuntu) does it too, and probably in a more affordable way.

But for complex applications or architectures, Roboconf brings a real value to teams. From this point of view, the LAMP example helps to understand the way it works.

INDUSTRIAL USE CASE

OpenPaaS

OpenPaaS is an open source PaaS (Platform as a Service) that aims at improving collaboration within and between organizations. It joins the ESN trend (**Enterprise Social Network**).

It covers the creation of communities around professional topics or projects, live messaging, email, agendas, document sharing and so on. All of this is structured around the notion of professional communities.

http://open-paas.org

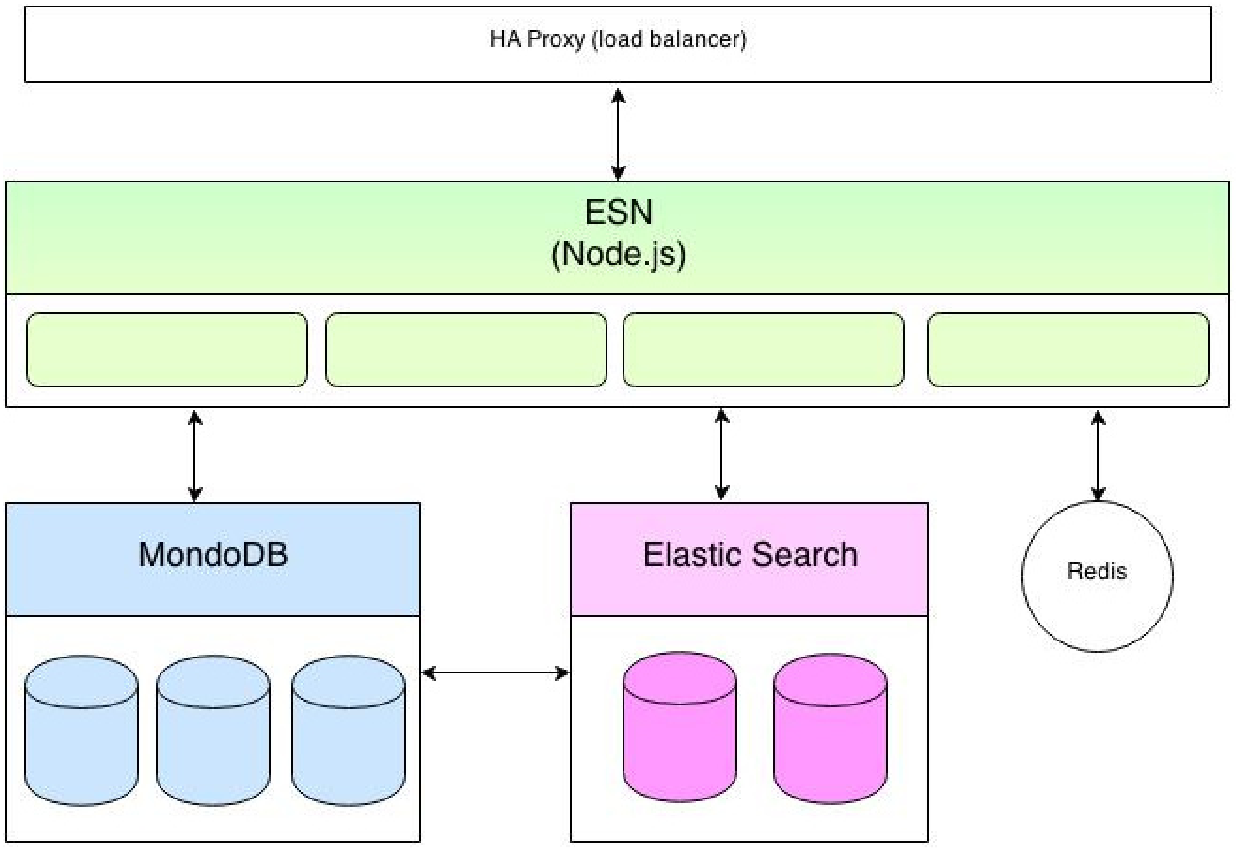
Technical Constraints

The platform relies on Web technologies.
The back-end is implemented with Node.js. The front-end is implemented with AngularJS and related projects.

The platform must be cloud-ready.
It must adapt to intense-load periods.
And it must scale for more and more users. This led to use no-SQL databases.

Various other programs are or will be required in addition.
So, OpenPaaS is a complex distributed platform that must run in the cloud.

Overview



For the moment, OpenPaaS focuses on the web stack.
Scalable parts include the ESN application (Node.js), MongoDB and ElasticSearch.

Planned Upgrade

OpenPaaS should soon integrate other Software.

- Apache James, a Java-server for e-mailing.
- Cassandra, another no-SQL database.
- OpenLDAP and LinID.

Cassandra is used to store e-mails in a distributed database.
LDAP is used for authentication and permissions management.

This will add new scalable parts in the platform.
Hopefully, Roboconf will help to make its management as simple as possible.

CONFIGURATION FILES

Project Structure

- descriptor/
  - application.properties
- graph/
  - *.graph
  - 1 directory per component (for its recipes)
- instances/
  - *.instances

The graph is defined in ***.graph** files.
The description can be split in several files, for modularity.
The same thing applies to ***.instances** files.

This structure can also be ported into a Maven structure, for a combined use with the Roboconf Maven plugin.

Graph Files - 1/5

These files use a custom syntax, inspired from CSS.
They are readable and editable easily through any text editor. They are less verbose than XML, and less error-prone than YAML.

```
# A component called 'MyComponentName'
MyComponentName {
    installer: bash;
}
```

Comments and properties must fit on a single line.
installer is the extension Roboconf will use to deploy, start, stop and undeploy instances of this component.

Every component has an associated directory that contains recipes or resources. These resources are used by the Roboconf extension identified by the **installer** name.

Graph Files - 2/5

An important aspect of components are imports and exports. They are variables that a component exposes or requires.

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

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

If a component imports a variable, then another component in the graph must export it. A component instance cannot start unless all its imports are resolved.

Every time a component instance appears or leaves, it notifies those that import its variables. The notified instances can then take proper actions through their recipes.

Graph Files - 3/5

It is possible to define optional imports.
Optional imports mean a component can start, even if these imports are not resolved. But it will be notified if they appear.

```
# A cluster node.
ClusterNode {
        exports: ip, port = 18741;
        imports: ClusterNode.ip (optional), ClusterNode.port (optional);
}
```

In this example, the cluster node can start if it is alone. And it will be notified when other nodes appear or disappear.

When a component instance is started, and that one of its dependencies disappear, Roboconf stops it and puts it in the **starting** state. If the dependency comes back, it will automatically be started.

ip is a special variable whose value is set automatically by Roboconf.
Any other exported variable must have a default value.

Graph Files - 4/5

Imports and exports deal with runtime dependencies.
Containment relations are handled through the **children** key word.

```
# Tomcat and MySQL are other graph components
VM_Tomcat {
	children: Tomcat;
}

VM_MySQL {
	children: MySQL;
}

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

children lists components that can be deployed over an instance of this component. It can be very constrained or quite flexible.

Graph Files - 5/5

Sometimes, components share properties.
It is possible to group properties into a **facet**.

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

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

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

VM_Azure {
        facets: VM;
}

Tomcat {
        facets: deployable;
}
```

Recipes - 1/3

Recipes are resources used by a Roboconf plug-in to manage the life cycle of component instances.

```
VM_EC2 {
	installer: target;
}

VM_Openstack {
	installer: target;
}
```

This graph implies there will be two directories in the graph, called **VM_EC2** and **VM_Openstack**. Because these components use the **target** installer, these directories will contain a **target.properties** file.

One will have parameters to handle VM on EC2.
The other one will contain parameters related to Openstack.

Recipes - 2/3

Here is an example of **target.properties**.

```
# These properties are specific to 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
```

Recipes - 3/3

The **target** installer is in charge of managing machines (VM, server, device...).
The configuration depends on the target type.

The **bash** installer manages the life cycle of a component through bash scripts (deploy, start, stop, undeploy and update). These scripts must be idempotent.

The **puppet** installer manages the life cycle of a component through a Puppet module. The module name must start with **roboconf_**.

These are the only available extensions for now.
But new ones can be easily created (Chef, ANT, custom one...).

Instances - 1/2

Components define rules and relations (how).
Instances define what really runs (what and where).

Unlike the graph, which was really a graph, instance definitions are trees.

```
# Deploy on EC2 VM (called VM1)
instance of VM_EC2 {
    name: VM1;
    instance of MyApp {
            name: MyApp_on_EC2;
    }
}

# Deploy on Openstack VM (called VM2)
instance of VM_Openstack {
    name: VM2;
    instance of MyApp {
            name: MyApp_on_Openstack;
    }
}
```

Instances - 2/2

Beyond specifying a name and its components, an instance can override the value of exported component properties.

It can even define new properties that were not defined in the component. This is useless with respect to exports, but it can be very useful in recipes.

```
# The Component
Tomcat {
        exports: ip, port = 8080;
}
```

```
# An instance
instance of Tomcat {
        name: Tomcat;
        port: 8080;
        log_level: INFO; # used in the recipe
}
```

Application Descriptor

The application descriptor is a simple properties file with meta-data about the project itself.

```
# Application Descriptor for Roboconf
application-name = MyApp
application-qualifier = sample
application-description = A sample application

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

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

TOOLS

Tooling

Beyond the way Roboconf is implemented, some choices have been made to make things simple. The syntax of configuration files was chosen in this sense.

However, tools are always helpful.
They help new users, and more generally, they make gain a lot of time.

eclipse

- Creation Wizards
- Text Editor

- Web Administration
- CLI Administration?

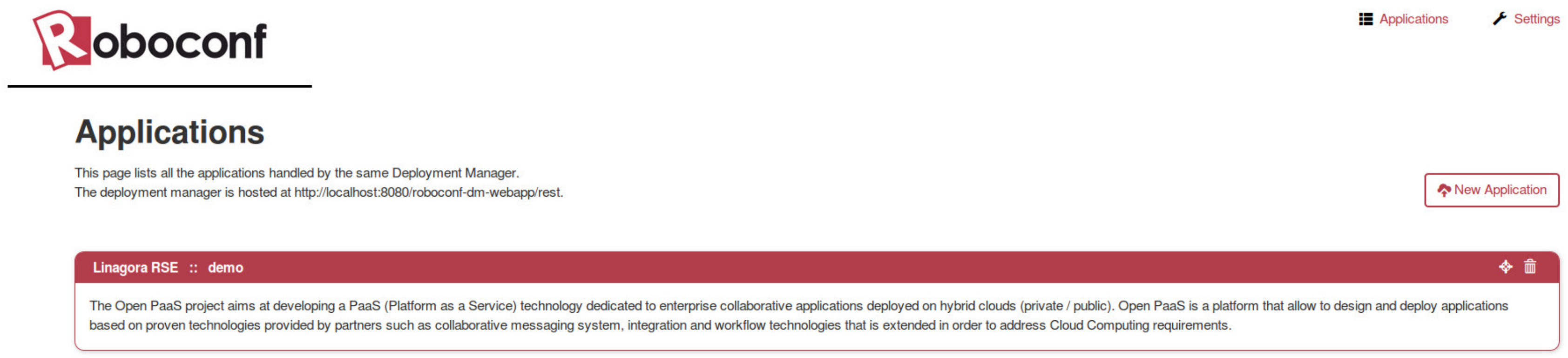
maven

- Validation
- Packaging
- Testing

- Report Generation
For Project & Support

The Web Administration - 1/2

The web administration is web application (AngularJS) that provides a user interface for the DM's REST API.



Applications

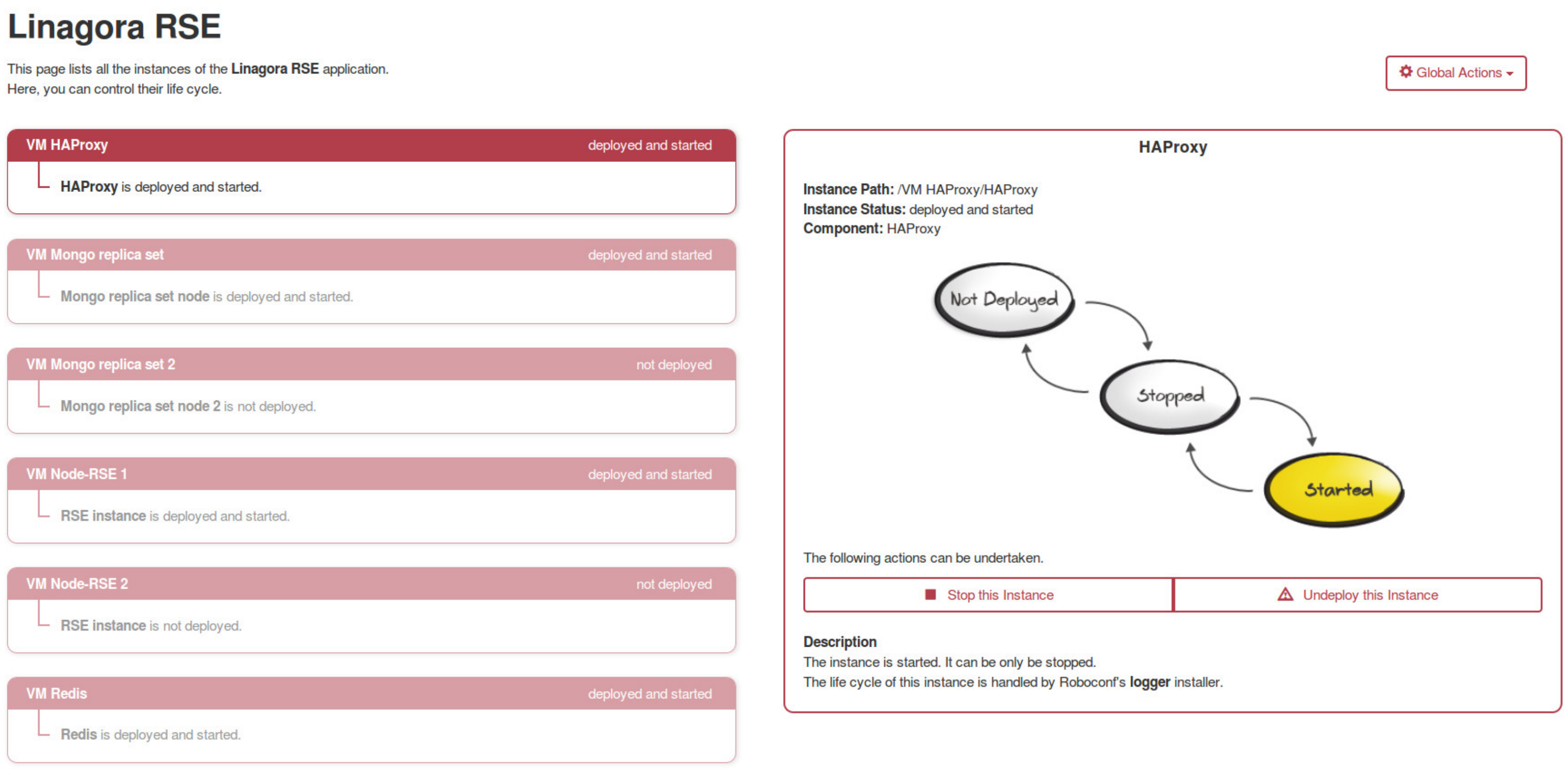
This page lists all the applications handled by the same Deployment Manager.
The deployment manager is hosted at http://localhost:8080/roboconf-dm-webapp/rest.

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.

The Web Administration - 2/2

This console helps to perform actions and watch the application parts.



Linagora RSE

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

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

The following actions can be undertaken.

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

Maven Plug-in

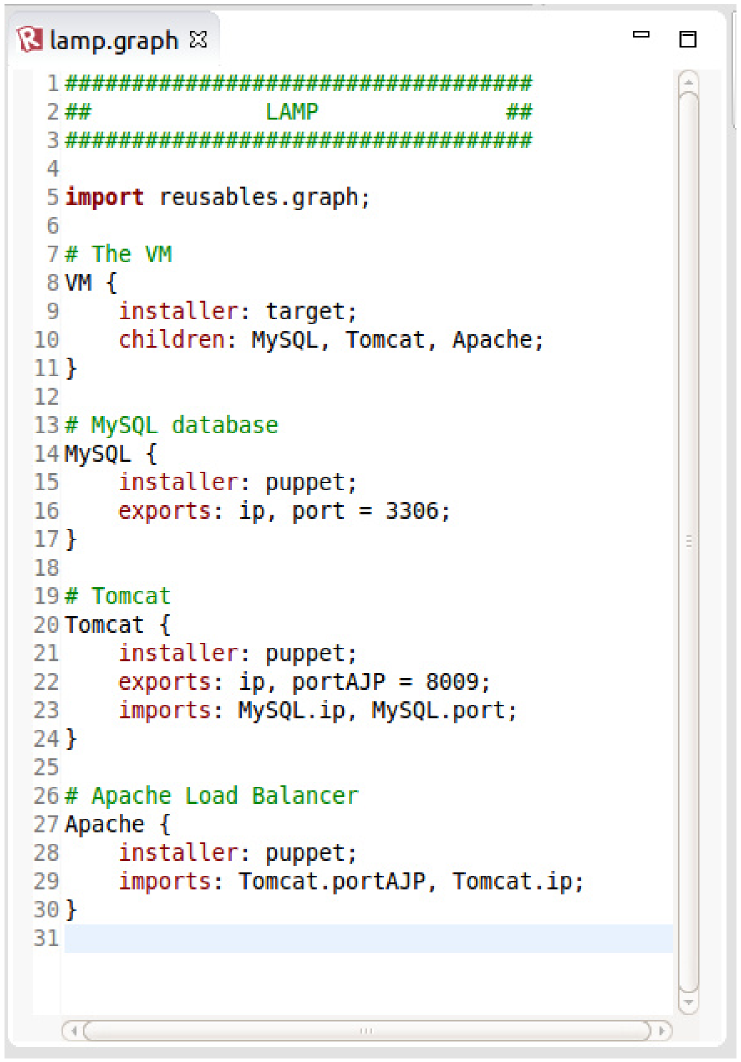
Roboconf has its own Maven plug-in.
It helps to...

- ... validate the configuration files (graph, instances)...
- ... package the project in a deployable archive...
- ... insert Maven properties in configuration files.

Ongoing work will allow testing and sharing reusable recipes.
Other tools will come later.

Eclipse Plug-in

Roboconf has also its own Eclipse plug-in.
It provides a creation wizard for new projects, and dedicated editors.



```
###################################
##               LAMP            ##
###################################

import reusables.graph;

# The VM
VM {
    installer: target;
    children: MySQL, Tomcat, Apache;
}

# MySQL database
MySQL {
    installer: puppet;
    exports: ip, port = 3306;
}

# Tomcat
Tomcat {
    installer: puppet;
    exports: ip, portAJP = 8009;
    imports: MySQL.ip, MySQL.port;
}

# Apache Load Balancer
Apache {
    installer: puppet;
    imports: Tomcat.portAJP, Tomcat.ip;
}

```

RESOURCES

Download & Documentation

roboconf.net

Source Code

github.com/roboconf

Contact & Questions

twitter.com/roboconf
github.com/roboconf/roboconf/issues