Project Acronym: Open-DAI

Grant Agreement number: 297362

Project Title: Opening Data Architectures and Infrastructures of European Public Administrations

Work Package: System/Platform implementation

Deliverable Number: D4.1

Revisioni History

Revision Date	Author	Organisation	Description
28/9/2012	Luca Gioppo	CSI-Piemonte	Final

Legal Discaimer

Statement of originality:
This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

1 Introduction

This document explains the technical details and installation steps to implement the virtualization platform of the Open-DAI project.
The virtualization platform represent all the middleware described in the D2.2 document, with some update due to actual implementation and last time changes forced by the occurrence of software incompatibility.
The document will explain the general approach to the installation model and will than detail the installation and configuration of the different tools.
It will not be a classical installation manual since much of the detailed operation are within the puppet modules used in the process and these modules will be published by the project to the whole open source community to get feedback.
This document explains the design decisions and the technical rationale behind the installation process.

Using open source technology (and in the project there is a wide amount of technology involved) often causes that some path ends up into a dead end for the tool is not “good enough” for the task at hand or not completed in the parts needed by the project or there are bugs that will be fixed with a time span not compatible with the project work plan, this has caused a few workaround strategy that gives to the installation design a mixed approach since not all tools could be treated at the same way.

The project will keep a task running for improving the installation and configuration process to keep up with the tools evolution and to fix some of the quirkiness present in the installation procedure.

2 Installation model

The need of the project is to have a process that is:

- Automated
- Repeatable

The reason for these constraints are the following:
The model is designed to offer a “private cloud” to all customer in a similar way as a IAAS model, with the basic difference that the client of the cloud provider is not allowed to deploy its own VM on it, but will receive a pre-cooked environment to use for his business; this way the end user is masked the complexity of the cloud environment, and will also be helped in the usage of the tools.

The analogy that can be drawn is with car tracks: the end user will be able to access a big table (the cloud environment) all for his own usage with a pre-mounted track (the middleware), he will manage to understand how the track runs and how to make cars go around (the business applications); he will have the opportunity to make some part of the track bigger (scale the middleware thanks to the Cloud environment) or close some path (switch off some middleware to consume less resources and pay less).

To reach this goal the installation and configuration need to be designed to enable end user the easiness expected without raising too much the operation costs.
In fact another goal of the project is to have a low operation cost and is known that masking complexity to end user often comports to raise operation cost to manage all the tools and integrations needed to implement that ease of use.

In the architecture there are many middleware software tools that have their own peculiarities and installation prerequisites, having a team dedicated to all the aspect of the installation and configuration is a heavy requirement for the platform owner.
The end user of the middleware is not interested in the details of the installation and configuration, he just wants to use the middleware and do business with it.

This is also a project constraint: masking the complexity of the software to the end user considering that the end user will be a public administration od one of its own contractors and that the business of these users will have to be opening data from legacy databases and developing new services with that data.
The middleware HAS to be a tool in the hand of these users and not something that they have to worry about.

Installing all that middleware requires high IT skill since it does not only require the ability do make a proper installation, but is needed all the integration to offer the end user a homogeneous environment where he does not need to know the internals working of the systems.
This complexity cannot mastered by hand and the installation must be guaranteed to be the same in all the different private cloud for all end users.

Installing all the middleware is a time consuming operation (it could take about a week time of hand installation and configuration for the complete environment) and we model calls for a fast time to market: the cloud provider has to be able to let the end user start up in a quick way.

The proposed solution combines two main elements:

- A bootstrap installer for the cloud provider
- A puppet environment with its own custom modules for the single private cloud or as Cloud Stack calls it, “cloud domain” or simply “domain” (in the document it will be used this notation).

The bootstrap installer is used by the cloud provider to solve “the chicken or the egg dilemma” that is how to install the installer (in our case the puppet master) in all domains.
The puppet master will thus be able to install and configure all the middleware virtual machines once in place.

Is not precise to say that the puppet master will “install” because since the virtual machines are inside a cloud environment we need to command the cloud environment to create a new virtual machine to instruct it to become the piece of middleware that we need.

2.1 Declarative approach

The approach used in the project, to simplify the general complexity, is to assign to specific machines a given name.
This declarative approach allows to know beforehand the name of the resource that will be required in a specific installation step; to make an example: the puppet environment is composed of a MySQL database that keeps all the data of the hosts and the puppet master host that do the work.
Having set the names of these hosts that will be the same in all domains allows to implement a more straightforward installation script for the puppet master that knows beforehand the name of the DB to which it needs to connect.

This approach has been adopted for all the middleware components as a first step in the platform implementation.
It will be possible and will be looked after during the project to add a dynamic approach in the installation leaving the end user the power to name the middleware of to internally assign a dynamic not predetermined name so that it could be possible to scale horizontally the clustered systems (right now the approach is to have clusters of two nodes with given names)

2.2 Cloud overview

As stated before the installation phase require to talk with the cloud environment to get information on the offerings available to the single domain and to command the deploy of virtual machine in the domain.
The cloud environment has two approaches to create new virtual machines:

- It create a new machine out of a ISO image, this way the new machine either require a manual installation since most ISO run in an interactive mode or require the integration with tools that

manage an automated installation. This second approach that could appear interesting has the disadvantages that the number of operations that can be done during a normal first ISO installation are limited and there should be in any case the need of a more advanced approach with a tool like puppet.

- It creates a new machine starting from a template. The template is an image of and existing virtual machine frozen in time and configured to be clonable. This approach has the advantage of being able to start from a more complex operative system configuration greatly simplifying the installation process.

The project choose the second approach offered by the cloud platform.
The design decision that had to be mad was, at this point, the amount of software to install in the template.
Having about 16 virtual machine in the project architecture one approach could be the one to create 16 templates and instantiate the corresponding instance when needed.
This approach has the following problems:

- Each template require a storage
- Management of the template in time can be costly since you run the risk of having, even at operative system level, 16 different version of base software raising the cost of change management
- Is much mode difficult to trace modifiers in a template
- Is much more difficult to fix problems in a template

The decision made in the project was to create a very basic template with the essential software:

- CentOS 6.3 (x64)
- SSH server
- iptables rules not set
- sysstat package
- NFS client
- wget package
- SELinux disabled

And a start-up script that will be described afterwards in the document.

There is another feature of the cloud platform that is essential for the project: the ability to pass user data to the virtual machine during its creation.
CloudStack provides API access to attach user data to a deployed VM. Deployed VMs also have access to instance metadata via the virtual router. The virtual router is a type of CloudStack System Virtual Machine. The virtual router is one of the most frequently used service providers in CloudStack. The end user has no direct access to the virtual router. Users can ping the virtual router and take actions that affect it (such as setting up port forwarding), but users do not have SSH access into the virtual router.

The client accesses user data via the virtual router. The deploy call passes the data to the virtual router which then makes it available via HTTP to the guests over the guest network interface. Guests fetch the data whenever needed.
For example, this script can be used to fetch the data:

```
#!/bin/bash
server_ip=$(grep dhcp-server-identifier /var/lib/dhclient-eth0.leases | tail
-1
| awk '{print $NF}' | tr '\;' ' ') wget http://${server_ip}/latest/user-data
```

Using this technique enables a detailed parameterization of the new instance defining a role to be assigned to the created VM (and passed to it).
Basing on this role it will be able to declare it to the puppet environment that will act accordingly.

2.3 Bootstrap installer

The problem is how to install the puppet environment in all the domains.

The puppet environment has to be inside the domain because it will keep all the information of the middleware environment and is important that all the domains have a dedicated resource and access to this management infrastructure.
It could be possible to implement it as an external feature, but it would require a much more complex configuration to keep the different domains information segregated and accessible just to the right owner.

To solve the problem we used an external VM to solve a double task:

- Serving as an rpm and http repository for software to speed up installation
- Hosting a custom made application to start up the domain

2.3.1 Bootstrap application

The project developed a small PHP application to install the puppet environment.
To develop the application we used the YII framework and MySQL database to keep data.
The application is simple, but execute a critical installation: the cloud environment has a template dedicated to management VM and the application has the API keys of all the administrator users of each domain and can thus connect to the platform as the user to execute VM deployments.
It will thus be able to execute a small workflow summed up below:

1. Check how many VM are present in the domain, in case of none it present with the option to create the first VM that will be the management DB (better described in the 2.4 chapter)
2. User command the creation
3. The application issue to the cloud platform the API command to deploy a VM based on the management template with a predefined service offering in the internal network
4. The application wait (and inform the user) for the completion of the operation by the cloud platform
5. As soon that the application receives the ok on the creation of the VM it prompt the user on the possibility to create the puppet master
6. The user command the creation
7. The application issue to the cloud the API commend to create a second VM based on similar parameters, passing to the VM user data that enable it to set all the necessary configuration

The VM when created automatically run a script that recover the user data and place the information in environment variables and than get a dedicated script that install the required software.
In the following chapters we'll explain the operations in detail.

This simple tool enable the cloud provider to activate a new domain with the Open-DAI platform in more or less 30 minutes from the customer request: considering the typology of end users this is a very quick response time, but there could be improvement opportunities dedicating more resources in the provisioning component.

2.3.1.1 Management database installation

The following boxed content is the representation of the script that install the management DB.
As can be seen the first operations are installing the necessary repositories.
The next important operation is to set the correct time for the host: this is an important issue since all the infrastructure make use of certificates and all the hosts need to have a good time synchronization to accept the certificates as valid and not firing out error of invalid/expired certificate; also many operations as scheduled batches for backup need to count on correct time.

The operation count on the presence of a user data parameter that states on which time zone the domain is.
This is important due to the fact that each domain will serve a public administration in a specific European country.

Than the script install facter and augeas that are part of the puppet environment and are tool to get information on the host and to manipulate config files.
This is essential since there will be many files to edit to set up the correct configuration based on dynamic configuration of the host: for example we cannot know beforehand the IP address that the cloud platform will assign to the host and the script has to be able to get it and write it down on every config file that need to have that information.

At this point the script can silently (that means without any interaction with the user) install MySQL.
Installed the package the script start to configure the installation changing the root password, and setting specific configuration option that the project designed for performance in consideration of the tasks that the particular instance of MySQL will have to do.

```
#!/bin/bash

log "start creating management DB machine"
#add puppet repository
rpm --import https://fedoraproject.org/static/0608B895.txt
rpm      -ivh      http://yum.puppetlabs.com/el/6/products/x86_64/puppetlabs-release-6-
6.noarch.rpm
#rpm -ivh http://yum.puppetlabs.com/el/6/dependencies/x86_64/puppetlabs-release-devel-
6-1.noarch.rpm
#add RPMFORGE repository
rpm       -ivh       http://pkgs.repoforge.org/rpmforge-release/rpmforge-release-0.5.2-
2.el6.rf.x86_64.rpm
#add EPEL repository
rpm -ivh http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-7.noarch.rpm
log "created repositories"

log "create Open-DAI repo"
echo                 -e                "[yum-repo]\nname=yum-repo\nbaseurl=http://yum-
repo.cloudlabcsi.eu/repo/\$releasever/\$basearch/\ngpgcheck=false\nenabled=1"        >
/etc/yum.repos.d/yum-repo.repo

log $(yum check-update)

#install ntp
#NOTA occorre passare tra le uservariables la timezone corretta
yum install  --nogpgcheck -y ntp
rm -f /etc/localtime
ln -s /usr/share/zoneinfo/Europe/$timezone /etc/localtime
ntpdate 1.centos.pool.ntp.org
service ntpd start

log "started ntp" $(service ntpd status)

log "installing facter"
yum install --nogpgcheck -y facter
myHostname=$(facter fqdn)
myHost=$(facter hostname)
myIP=$(facter ipaddress)
myDomain=$(facter domain)
puppet_srv=puppet
puppet_db=puppet
puppet_db_pwd=puppet
puppet_user=puppet
puppet_dash_db_pwd=dashboard
```

```
dashboard_db=dashboard_production
dashboard_user=dashboard
zabbix_srv=zabbix
zabbix_db=zabbix
zabbix_db_pwd=zabbix
zabbix_user=zabbix
zabbix_db_pwd=zabbix

log "installing augeas"
yum install --nogpgcheck -y augeas


#Server MySQL
yum install --nogpgcheck -y mysql-server
log "MySQL Installed"
 #mysql -uroot -p'csi$mgmtdb' -e "SELECT VERSION();SELECT NOW()"
 #mysql -uroot <(echo "create database mydatabase;")

 #  echo  -e  "set  /augeas/load/test/lens  Shellvars.lns\nset  /augeas/load/test/incl
/root/test.log\nload\ndefnode default /files/root/test.log/default 33\nsave"|augtool

res=$(augtool   defnode   default-storage-engine   /files/etc/my.cnf/target[1]/default-
storage-engine InnoDB -s)
res=$(augtool defnode long_query_time /files/etc/my.cnf/target[1]/long_query_time 3 -
s)
res=$(augtool   defnode   slow_query_log   /files/etc/my.cnf/target[1]/slow_query_log
/var/log/mysql/mysql-slow-queries.log -s)
res=$(augtool                        defnode                        innodb_file_per_table
/files/etc/my.cnf/target[1]/innodb_file_per_table -s)
res=$(augtool                       defnode                       innodb_buffer_pool_size
/files/etc/my.cnf/target[1]/innodb_buffer_pool_size 1G -s)
res=$(augtool  defnode  query_cache_size  /files/etc/my.cnf/target[1]/query_cache_size
500M -s)
res=$(augtool                   defnode                   innodb_flush_log_at_trx_commit
/files/etc/my.cnf/target[1]/innodb_flush_log_at_trx_commit 2 -s)
res=$(augtool                     defnode                     innodb_thread_concurrency
/files/etc/my.cnf/target[1]/innodb_thread_concurrency 8 -s)
res=$(augtool                        defnode                        innodb_flush_method
/files/etc/my.cnf/target[1]/innodb_flush_method O_DIRECT -s)
res=$(augtool                        defnode                        max_heap_table_size
/files/etc/my.cnf/target[1]/max_heap_table_size 128M -s)
res=$(augtool defnode table_cache /files/etc/my.cnf/target[1]/table_cache 128 -s)
res=$(augtool  defnode  query_cache_limit  /files/etc/my.cnf/target[1]/query_cache_limit
16M -s)
res=$(augtool  defnode  query_cache_size  /files/etc/my.cnf/target[1]/query_cache_size
128M -s)
res=$(augtool defnode tmp_table_size /files/etc/my.cnf/target[1]/tmp_table_size 128M -
s)
res=$(augtool  defnode  thread_cache_size  /files/etc/my.cnf/target[1]/thread_cache_size
4 -s)
res=$(augtool                        defnode                        max_allowed_packet
/files/etc/my.cnf/target[1]/max_allowed_packet 32M -s)

service mysqld start
log "MySQL working:" $(service mysqld status)
chkconfig --levels 235 mysqld on
root_pwd="csi$mgmtdb"
/usr/bin/mysqladmin -u root password $root_pwd
/usr/bin/mysqladmin -u root -h $myHost password $root_pwd
/usr/bin/mysqladmin -u root -h localhost password $root_pwd
#Password per l'utente root settata al valore: csi$topix
log "changed root password"

#setting innodb default
```

```
mysql -uroot -p$root_pwd -e "INSTALL PLUGIN INNODB_TRX SONAME
'ha_innodb_plugin.so';INSTALL PLUGIN INNODB_LOCKS SONAME 'ha_innodb_plugin.so';INSTALL
PLUGIN INNODB_LOCK_WAITS SONAME 'ha_innodb_plugin.so';INSTALL PLUGIN INNODB_CMP SONAME
'ha_innodb_plugin.so';INSTALL PLUGIN INNODB_CMP_RESET SONAME
'ha_innodb_plugin.so';INSTALL PLUGIN INNODB_CMPMEM SONAME
'ha_innodb_plugin.so';INSTALL PLUGIN INNODB_CMPMEM_RESET SONAME
'ha_innodb_plugin.so';SET storage_engine=InnoDB;"
service mysqld restart
log "MySQL working:" $(service mysqld status)

log $(mysql -uroot -p$root_pwd -e "select plugin_name, plugin_type, plugin_status from
information_schema.plugins;")
log $(mysql -uroot -p$root_pwd -e "show engines;")
log $(mysql -uroot -p$root_pwd -e "show variables;")


#create PUPPET user and DB
mysql -uroot -p$root_pwd -e "CREATE DATABASE $puppet_db CHARACTER SET utf8;CREATE USER
'$puppet_user'@'$puppet_srv.$myDomain' IDENTIFIED BY '$puppet_db_pwd';GRANT ALL
PRIVILEGES ON $puppet_db.* TO '$puppet_user'@'$puppet_srv.$myDomain';"

#CREATE USER '[DB NAME]_admin'@'localhost' IDENTIFIED BY '[ADMIN PASSWORD]';
#GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, ALTER ON [DB NAME].* TO '[DB
NAME]_admin'@'localhost';

# Create dashboard user and DB
mysql -uroot -p$root_pwd -e "CREATE DATABASE $dashboard_db CHARACTER SET utf8;CREATE
USER '$dashboard_user'@'$puppet_srv.$myDomain' IDENTIFIED BY
'$puppet_dash_db_pwd';GRANT ALL PRIVILEGES ON $dashboard_db.* TO
'$dashboard_user'@'$puppet_srv.$myDomain';"

# Create Zabbix user and DB
mysql -uroot -p$root_pwd -e "CREATE DATABASE $zabbix_db CHARACTER SET utf8;CREATE USER
'$zabbix_user'@'$zabbix_srv.$myDomain' IDENTIFIED BY '$zabbix_db_pwd';GRANT ALL
PRIVILEGES ON $zabbix_db.* TO '$zabbix_user'@'$zabbix_srv.$myDomain';"

log $(mysql -uroot -p$root_pwd -e "use mysql;select * from user;")
```

Having set up the DB it will finally create the databases and users for the management and puppet environment

2.3.1.2 Puppet master installation

The script of the installation of the puppet master get more user data to follow up the installation of the basic building pieces of the Open-DAI platform.

As before the first set of operations are the repository update and the time synchronization.

Than it start a much more complex workflow of operations:
Prerequisites for puppet will be installed:

- MySQL client to enable it to connect to the management DB, create the initial set of tables and be able to store the different configurations.
- Apache since it will be required to serve the puppet-dashboard and the management application
- PHP since the puppet-dashboard is a PHP application; with PHP will be installed all the PHP libraries needed
- Subversion client
- Augeas configuration tool

Now is time to install puppet.

The script will than install the mcollective client and the java sdk

Finished the installation it will be the time to configure the various files with the dynamic information.

Puppet needs than to be instructed to store the information in the external storage (that is configure it to use storeconfig option)

Than the script import through subversion the current version of the administration console and the puppet modules.

At this point all is ready to run puppet and all the other services that are started at the end of the script.

```
#!/bin/bash

log "start creating puppet-master machine"

#### REPOS
#add puppet repository
rpm --import https://fedoraproject.org/static/0608B895.txt
rpm      -ivh      http://yum.puppetlabs.com/el/6/products/x86_64/puppetlabs-release-6-
6.noarch.rpm
#rpm -ivh http://yum.puppetlabs.com/el/6/dependencies/x86_64/puppetlabs-release-devel-
6-1.noarch.rpm
#add RPMFORGE repository
rpm       -ivh       http://pkgs.repoforge.org/rpmforge-release/rpmforge-release-0.5.2-
2.el6.rf.x86_64.rpm
#add EPEL repository
rpm -ivh http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-7.noarch.rpm
log "created repositories"

log "create Open-DAI repo"
echo                  -e                "[yum-repo]\nname=yum-repo\nbaseurl=http://yum-
repo.cloudlabcsi.eu/repo/\$releasever/\$basearch/\ngpgcheck=false\nenabled=1"        >
/etc/yum.repos.d/yum-repo.repo

log $(yum check-update)


#### INSTALL MACHINE
#install ntp
yum install --nogpgcheck -y ntp
rm -f /etc/localtime
ln -s /usr/share/zoneinfo/Europe/$timezone /etc/localtime
ntpdate 1.centos.pool.ntp.org
service ntpd start

log "started ntp" $(service ntpd status)

#install prerequisite packets
# Client MySQL
yum install --nogpgcheck -y mysql
log "MySQL Client Installed"

# Apache
yum install --nogpgcheck -y httpd
chkconfig --levels 235 httpd on
service httpd start
log "started httpd" $(service httpd status)

# PHP
```

```
yum install --nogpgcheck -y php

# Subversion to get puppet modules from repository
yum install --nogpgcheck -y subversion

# Configuration tool Augeas
yum install --nogpgcheck -y augeas

#install Puppet
yum install --nogpgcheck -y puppet-server

#operations to do to connect puppet server to mysql
yum install --nogpgcheck -y ruby-mysql
yum install --nogpgcheck -y rubygems ruby-devel mysql-devel
gem install rails -v 3.0 --no-ri --no-rdoc

#installare mcollective client
yum install --nogpgcheck -y rubygem-stomp
yum install --nogpgcheck -y mcollective mcollective-client mcollective-common

#install java (openjdk will do)
yum install --nogpgcheck -y java-1.6.0-openjdk.x86_64
rpm    -ivh    http://yum.puppetlabs.com/el/6/dependencies/x86_64/tanukiwrapper-3.5.9-
1.el6.x86_64.rpm
rpm       -ivh       http://yum.puppetlabs.com/el/6/dependencies/x86_64/activemq-5.5.0-
1.el6.noarch.rpm
log "installed mcollective and java"

# puppet dashboard
yum install --nogpgcheck -y puppet-dashboard

# Now we can configure
#### CONFIGURATION

myHostname=$(facter fqdn)
myIP=$(facter ipaddress)
myDomain=$(facter domain)
puppetDB=mgmtdb.$myDomain
mc_pwd=mcopwd
mc_stomp_pwd=mcopwd
dash_db_pwd=dbdashpwd


# HAVE TO DEFINE CONFDIR E VARDIR since are not present in the puppet.conf
augtool ins confdir before /files/etc/puppet/puppet.conf/main/logdir -s
augtool set /files/etc/puppet/puppet.conf/main/confdir /etc/puppet -s
augtool ins vardir before /files/etc/puppet/puppet.conf/main/logdir -s
augtool set /files/etc/puppet/puppet.conf/main/vardir /var/lib/puppet -s

res=$(augtool     defnode     certname     /files/etc/puppet/puppet.conf/main/certname
${myHostname,,} -s)
log $res
augtool defnode storeconfigs /files/etc/puppet/puppet.conf/master/storeconfigs true -s
augtool defnode dbadapter /files/etc/puppet/puppet.conf/master/dbadapter mysql -s
augtool defnode dbname /files/etc/puppet/puppet.conf/master/dbname puppet -s
augtool defnode dbuser /files/etc/puppet/puppet.conf/master/dbuser puppet -s
augtool defnode dbpassword /files/etc/puppet/puppet.conf/master/dbpassword puppet -s
augtool      defnode      dbsocket      /files/etc/puppet/puppet.conf/master/dbsocket
/var/lib/mysql/mysql.sock -s
augtool defnode dbserver /files/etc/puppet/puppet.conf/master/dbserver $puppetDB -s
augtool defnode reports /files/etc/puppet/puppet.conf/master/reports 'store, http' -s
augtool      defnode      reporturl      /files/etc/puppet/puppet.conf/master/reporturl
"http://${myIP,,}:3000/reports/upload" -s
```

```
augtool defnode node_terminus /files/etc/puppet/puppet.conf/master/node_terminus exec
-s
echo -e "defnode external_nodes /files/etc/puppet/puppet.conf/master/external_nodes
'/usr/bin/env PUPPET_DASHBOARD_URL=http://${myIP,,}:3000 /usr/share/puppet-
dashboard/bin/external_node'"|augtool -s
augtool defnode fact_terminus /files/etc/puppet/puppet.conf/master/fact_terminus
inventory_active_record -s
augtool defnode modulepath /files/etc/puppet/puppet.conf/master/modulepath
\$confdir/environments/\$environment/modules -s
augtool defnode manifest /files/etc/puppet/puppet.conf/master/manifest
\$confdir/environments/\$environment/manifests/unknown_environment.pp -s

augtool defnode manifest /files/etc/puppet/puppet.conf/production/manifest
\$confdir/environments/\$environment/manifests/site.pp -s

augtool defnode manifest /files/etc/puppet/puppet.conf/dev/manifest
\$confdir/manifests/site.pp -s


#create autosign.conf in /etc/puppet/
echo -e "*.${myDomain,,}" > /etc/puppet/autosign.conf
log "edited autosign.conf"

# appendere nel file /etc/puppet/auth.conf del master
echo -e "path /facts\nauth any\nmethod find, search\nallow *" >> /etc/puppet/auth.conf
log "appended stuff in puppet/auth.conf"


# Checkout the svn repo
svn co http://yum-repo.cloudlabcsi.eu/repos/puppet-modules/trunk /etc/puppet
svn co http://yum-repo.cloudlabcsi.eu/repos/www/trunk/html /var/www/html

#set config.php
sed -i "s/API_KEY/$ak/g" /var/www/html/opendai-consolle/config.php
sed -i "s/SECRET_KEY/$sk/g" /var/www/html/opendai-consolle/config.php
sed -i "s/DOM_ID/$di/g" /var/www/html/opendai-consolle/config.php


#### START PUPPET MASTER NOW
puppet master --verbose

# Modify mcollective/client.cfg

sed -i "s/plugin.psk = unset/plugin.psk = $mc_pwd/g" /etc/mcollective/client.cfg
sed -i "s/plugin.stomp.host = localhost/plugin.stomp.host =
puppet.courtyard.cloudlabcsi.eu/g" /etc/mcollective/client.cfg
sed -i "s/plugin.stomp.port = 61613/plugin.stomp.port = 6163/g"
/etc/mcollective/client.cfg
sed -i "s/plugin.stomp.password = secret/plugin.stomp.password = $mc_stomp_pwd/g"
/etc/mcollective/client.cfg

#Modify /etc/activemq/activemq.xml
echo -e "set /augeas/load/activemq/lens Xml.lns\nset /augeas/load/activemq/incl
/etc/activemq/activemq.xml\nload\nset
/files/etc/activemq/activemq.xml/beans/broker/transportConnectors/transportConnector[2
]/#attribute/uri stomp+nio://0.0.0.0:6163"|augtool -s
echo -e "set /augeas/load/activemq/lens Xml.lns\nset /augeas/load/activemq/incl
/etc/activemq/activemq.xml\nload\nset
/files/etc/activemq/activemq.xml/beans/broker/plugins/simpleAuthenticationPlugin/users
/authenticationUser[2]/#attribute/password $mc_stomp_pwd"|augtool -s

#### Configure puppet-dashboard
#Add the host of the Db and password we use standard DB and user name
```

```
sed -i "s/ database: dashboard_production/ host: $puppetDB\n database:
dashboard_production/g" /usr/share/puppet-dashboard/config/database.yml
sed -i "s/ password:/ password: $dash_db_pwd/g" /usr/share/puppet-
dashboard/config/database.yml

sed -i "s/use_file_bucket_diffs: false/use_file_bucket_diffs: true/g"
/usr/share/puppet-dashboard/config/settings.yml
sed -i "s/file_bucket_server: 'puppet'/file_bucket_server: '$myHostname'/g"
/usr/share/puppet-dashboard/config/settings.yml
sed -i "s/ca_server: 'puppet'/ca_server: '$myHostname'/g" /usr/share/puppet-
dashboard/config/settings.yml
sed -i "s/inventory_server: 'puppet'/inventory_server: '$myHostname'/g"
/usr/share/puppet-dashboard/config/settings.yml
sed -i "s/cn_name: 'dashboard'/cn_name: 'dashboard.$myDomain'/g" /usr/share/puppet-
dashboard/config/settings.yml

# migrate the DB (execute from /usr/share/puppet-dashboard)
cd /usr/share/puppet-dashboard
log $(pwd)
res=$(rake RAILS_ENV=production db:migrate)
log $res

rake cert:create_key_pair
rake cert:request

log "migrated DB"

#You'll need to sign the certificate request on the master by running
cd
log $(pwd)
res=$(puppet cert sign dashboard.$myDomain)
log $res

#Then, from Dashboard's directory again, run:
cd /usr/share/puppet-dashboard
log $(pwd)
res=$(rake cert:retrieve)
log "retrieved certificate"
log $res

#run dashboard (sempre da /usr/share/puppet-dashboard
cd /usr/share/puppet-dashboard
./script/server -e production &
#eseguire i worker (sempre da /usr/share/puppet-dashboard
env RAILS_ENV=production ./script/delayed_job -p dashboard -n 1 -m start
```

2.3.2 Rpm repos host

The puppet environment will need packages that are not distributed in the normal centos repositories and cannot be found in other community repos.
The project had to prepare a small Centos compatible repository to host within the same network environment.
This approach has been followed also for archives like the JBoss and WSO2 tools that does not come in rpm packages but in tar.gz or zip format.
The goal is to quicken up the installation and reduce connection risks: having all the big archives (more or less all the mentioned archives have a size of hundreds of MB) in a local network can assure a high bandwidth and fast installation times.
This approach has also the effect of better controlling what gets installed in the servers: often some open source project has the habit of releasing a "final" version at the same URL and referring to that may cause to have different versions on different domains.

To implement the repos the project had to do the following.

```
#Setup packages
yum install createrepo httpd *yum*

#Activate services at startup
chkconfig httpd on

#Create repos folder
mkdir /var/www/html/repo

#into:
 /etc/httpd/conf/httpd.conf

<Directory /var/www/html/repo>
Options +Indexes
</Directory>

#restart Apache:

/sbin/service httpd restart

create directory for distro and arch:

mkdir -p  /var/www/html/repo/6/i386
mkdir -p  /var/www/html/repo/6/x86_64

#Move packages in the correct folder (32 or 64 bit)

#Generate repolist (x each arch):

cd /var/www/html/repo/6/i386
createrepo

fix permission:
chown -R apache:apache repo

restart Apache:
/sbin/service httpd restart

#crete repo file for yum:

[yum-repo]
name=yum-repo
baseurl=http://yum-repo.cloudlabcsi.eu/repo/$releasever/$basearch/
gpgcheck=false
enabled=1
```

This host serve also another service that is subversion.
This subversion instance is used for keeping the puppet modules stuff.
These modules are the core of the installation procedure and is important to keep versions of them.
This way all the puppet masters can check out the correct revision of the modules and be able to create the hosts basing on the defined specifications.

2.4 Puppet environment

Before entering in detail of the project's puppet environment is important to have a quick overview of the puppet architecture

2.4.1 Puppet overview

Puppet has been built with two modes in mind: A client/server mode with a central server and agents running on separate hosts, or a serverless mode where a single process does all of the work. To ensure consistency between these modes, Puppet has always had network transparency internally, so that the two modes used the same code paths whether they went over the network or not. Each executable can configure local or remote service access as appropriate, but otherwise they behave identically. Note also that you can use the serverless mode in what amounts to a client/server configuration, by pulling all configuration files to each client and having it parse them directly. This section will focus on the client/server mode, because it's more easily understood as separate components, but keep in mind that this is all true of the serverless mode, too.

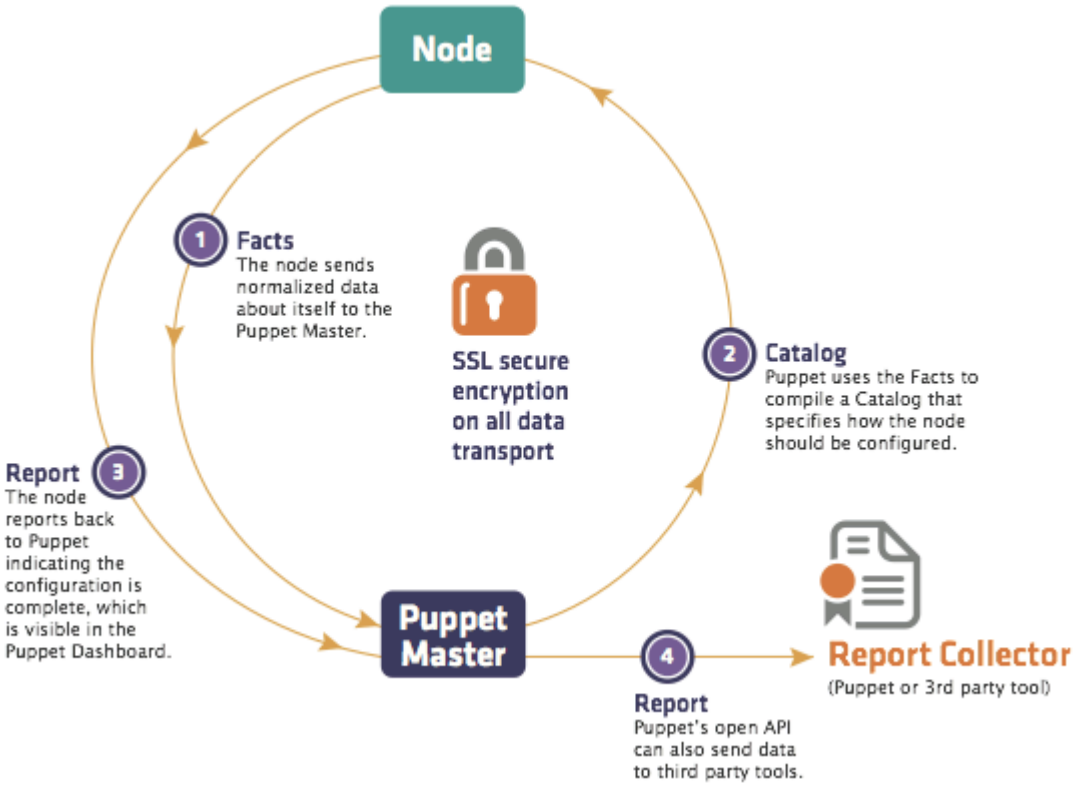
One of the defining choices in Puppet's application architecture is that clients should not get access to raw Puppet modules; instead, they get a configuration compiled just for them. This provides multiple benefits: first, you follow the principle of least privilege, in that each host only knows exactly what it needs to know (how it should be configured), but it does not know how any other servers are configured.

Second, you can completely separate the rights needed to compile a configuration (which might include access to central data stores) from the need to apply that configuration.

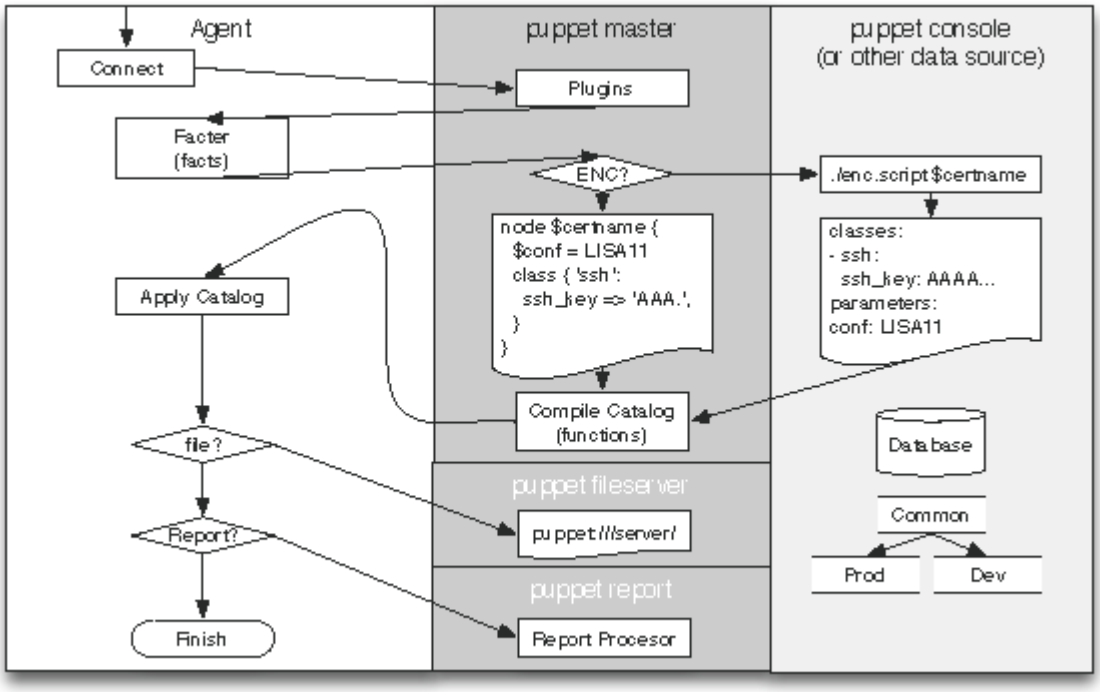
Third, you can run hosts in a disconnected mode where they repeatedly apply a configuration with no contact to a central server, which means you remain in compliance even if the server is down or the client is disconnected (such as would be the case in a mobile installation, or when the clients are in a DMZ).

Given this choice, the workflow becomes relatively straightforward:

1. The Puppet agent process collects information about the host it is running on, which it passes to the server.
2. The parser uses that system information and Puppet modules on local disk to compile a configuration for that particular host and returns it to the agent.
3. The agent applies that configuration locally, thus affecting the local state of the host, and files the resulting report with the server.
4. The server can generate reports on the activity



Thus, the agent has access to its own system information, its configuration, and each report it generates. The server has copies of all of this data, plus access to all of the Puppet modules, and any back-end databases and services that might be needed to compile the configuration.



Beyond the components that go into this workflow, which we'll address next, there are many data types that Puppet uses for internal communication. These data types are critical, because they're how all communication is done and they're public types which any other tools can consume or produce.
The most important data types are:

- **Facts**: System data collected on each machine and used to compile configurations.
- **Manifest**: Files containing Puppet code, generally organized into collections called "modules".
- **Catalog**: A graph of a given host's resources to be managed and the dependencies between them.
- **Report**: The collection of all events generated during application of a given Catalog.

Beyond Facts, Manifests, Catalogs, and Reports, Puppet supports data types for files, certificates (which it uses for authentication), and others.

A characteristic of puppet configurations is the concept of idempotency, meaning they can safely be run multiple times. Once you develop your configuration, your machines will apply the configuration often — by default, every 30 minutes — and Puppet will only make any changes to the system if the system state does not match the configured state.

If you tell the system to operate in no-op (“aka dry-run”), mode, using the --noop argument to one of the Puppet tools, puppet will guarantee that no work happens on your system. Similarly, if any changes do happen when running without that flag, puppet will ensure those changes are logged.
Because of this, you can use Puppet to manage a machine throughout its lifecycle — from initial installation, to ongoing upgrades, and finally to end-of-life, where you move services elsewhere.

Unlike system install tools like Sun’s Jumpstart or Red Hat’s Kickstart, Puppet configurations can keep machines up to date for years, rather than just building them correctly only the first time and then necessitating a rebuild. Puppet users usually do just enough with their host install tools to boostrap Puppet, then they use Puppet to do everything else.

This is has been the project’s design choice.

To look in detail into the flow of a puppet execution

2.4.1.1 Agent

The first component encountered in a Puppet run is the agent process. This was traditionally a separate executable called puppetd, but in version 2.6 it has been reduced down to one executable so now it is invoked with puppet agent, akin to how Git works. The agent has little functionality of its own; it is primarily configuration and code that implements the client-side aspects of the above-described workflow.

2.4.1.2 Facter

The next component after the agent is an external tool called Facter, which is a very simple tool used to discover information about the host it is running on. This is data like the operating system, IP address, and host name, but Facter is easily extensible so many organizations add their own plugins to discover custom data. The agent sends the data discovered by Facter to the server, at which point it takes over the workflow.

2.4.1.3 External Node Classifier

On the server, the first component encountered is what is called the External Node Classifier, or ENC. The ENC accepts the host name and returns a simple data structure containing the high-level configuration for that host. The ENC is generally a separate service or application: either another open source project, such as Puppet Dashboard or Foreman (in the case of Open-DAI the ENC will be the Puppet Dashboard), or integration with existing data stores, such as LDAP. The purpose of the ENC is to specify what functional classes a given host belongs to, and what parameters should be used to configure those classes.
Note that as of Puppet 2.7, the ENC is not a required component; users can instead directly specify node configurations in Puppet code. Support for an ENC was added about 2 years after Puppet was launched

because it was realized that classifying hosts is fundamentally different than configuring them, and it made more sense to split these problems into separate tools than to extend the language to support both facilities. The ENC is always recommended, and at some point will become a required component.
Once the server receives classification information from the ENC and system information from Facter (via the agent), it bundles all of the information into a Node object and passes it on to the Compiler.

2.4.1.4 Compiler

Puppet has a custom language built for specifying system configurations. Its compiler is really three chunks: A Yacc-style parser generator and a custom lexer; a group of classes used to create an Abstract Syntax Tree (AST); and the Compiler class that handles the interactions of all of these classes and also functions as the API to this part of the system.

The most complicated thing about the compiler is the fact that most Puppet configuration code is lazily loaded on first reference (to reduce both load times and irrelevant logging about missing-but-unneeded dependencies), which means there aren't really explicit calls to load and parse the code.
Puppet's parser uses a normal Yacc-style parser generator built using the open source Racc tool. Unfortunately, there were no open source lexer generators when Puppet was begun, so it uses a custom lexer.
Given the AST and a Node object (from the ENC), the compiler takes the classes specified in the node object (if there are any), looks them up and evaluates them. In the course of this evaluation, the compiler is building up a tree of variable scopes; every class gets its own scope which is attached to the creating scope. This amounts to dynamic scoping in Puppet: if one class includes another class, then the included class can look up variables directly in the including class.

The Scope tree is temporary and is discarded once compiling is done, but the artifact of compiling is also built up gradually over the course of the compilation. This artefact is called a Catalog, but it is just a graph of resources and their relationships. Nothing of the variables, control structures, or function calls survive into the catalog; it's plain data, and can be trivially converted to JSON, YAML, or just about anything else.
During compilation, containment relationships are created; a class "contains" all of the resources that come with that class. A class might contain a definition, which itself contains either yet more definitions, or individual resources. A catalog tends to be a very horizontal, disconnected graph: many classes, each no more than a couple of levels deep.

One of the awkward aspects of this graph is that it also contains "dependency" relationships, such as a service requiring a package (maybe because the package installation actually creates the service), but these dependency relationships are actually specified as parameter values on the resources, rather than as edges in the structure of the graph.

2.4.1.5 Transaction

Once the catalog is entirely constructed (assuming there is no failure), it is passed on to the Transaction. In a system with a separate client and server, the Transaction runs on the client, which pulls the Catalog down via HTTP as in Figure 18.2.
Puppet's transaction class provides the framework for actually affecting the system, whereas everything else we've discussed just builds up and passes around objects. Unlike transactions in more common systems such as databases, Puppet transactions do not have behaviors like atomicity.

The transaction performs a relatively straightforward task: walk the graph in the order specified by the various relationships, and make sure each resource is in sync. As mentioned above, it has to convert the graph from containment edges (e.g., Class[ssh] contains Package[ssh] and Service[sshd]) to dependency edges (e.g., Service[sshd] depends on Package[ssh]), and then it does a standard topological sort of the graph, selecting each resource in turn.

For a given resource, is performed a simple three-step process: retrieve the current state of that resource, compare it to the desired state, and make any changes necessary to fix discrepancies. For instance, given this code:

```
file { "/etc/motd":
    ensure => file,
    content => "Welcome to the machine",
    mode => 644
}
```

the transaction checks the content and mode of /etc/motd, and if they don't match the specified state, it will fix either or both of them. If /etc/motd is somehow a directory, then it will back up all of the files in that directory, remove it, and replace it with a file that has the appropriate content and mode.

2.4.1.6 Resource Abstraction Layer

The Transaction class is the heart of getting work done with Puppet, but all of the work is actually done by the Resource Abstraction Layer (RAL), which also happens to be the most interesting component in Puppet, architecturally speaking.

The RAL was the first component created in Puppet and, other than the language, it most clearly defines what the user can do. The job of the RAL is to define what it means to be a resource and how resources can get work done on the system, and Puppet's language is specifically built to specify resources as modeled by the RAL.

The Transaction doesn't actually affect the system directly, and it instead relies on the RAL for that. The providers are a RAL component that do the actual work. In fact, in general the providers are the only part of Puppet that actually touch the system. The transaction asks for a file's content, and the provider collects it; the transaction specifies that a file's content should be changed, and the provider changes it.

Note, however, that the provider never decides to affect the system—the Transaction owns the decisions, and the provider does the work. This gives the Transaction complete control without requiring that it understand anything about files, users, or packages, and this separation is what enables Puppet to have a full simulation mode where it’s largely guaranteed the system won't be affected.

2.4.1.7 Reporting

As the transaction walks the graph and uses the RAL to change the system's configuration, it progressively builds a report. This report largely consists of the events generated by changes to the system. These events, in turn, are comprehensive reflections of what work was done: they retain a timestamp the resource changed, the previous value, the new value, any message generated, and whether the change succeeded or failed (or was in simulation mode).

The events are wrapped in a ResourceStatus object that maps to each resource. Thus, for a given Transaction, you know all of the resources that are run, and you know any changes that happen, along with all of the metadata you might need about those changes.
Once the transaction is complete, some basic metrics are calculated and stored in the report, and then it is sent off to the server (if configured). With the report sent, the configuration process is complete, and the agent goes back to sleep or the process just ends

2.4.2 Hiera

Hiera makes Puppet better by **keeping site-specific data out of file manifests.** Puppet classes can request whatever data they need, and your Hiera data will act like a site-wide config file.
This makes it:

- Easier to configure your own nodes: default data with multiple levels of overrides is finally easy.
- Easier to re-use public Puppet modules: don’t edit the code, just put the necessary data in Hiera.
- Easier to publish your own modules for collaboration: no need to worry about cleaning out your data before showing it around, and no more clashing variable names.

With Hiera, is possible can:

- Write common data for *most* nodes
- Override *some* values for machines located at a particular facility…
- …and override some of *those* values for one or two unique nodes.

This way, the user only have to write down the *differences* between nodes. When each node asks for a piece of data, it will get the specific value it needs.
To decide which data sources can override which, Hiera uses a **configurable hierarchy.** This ordered list can include both **static** data sources (with names like “common”) and **dynamic** ones (which can switch between data sources based on the node’s name, operating system, and more).

This component is installed in the puppet manager host ant serves as the project common data repository.

2.4.3 Mcollective

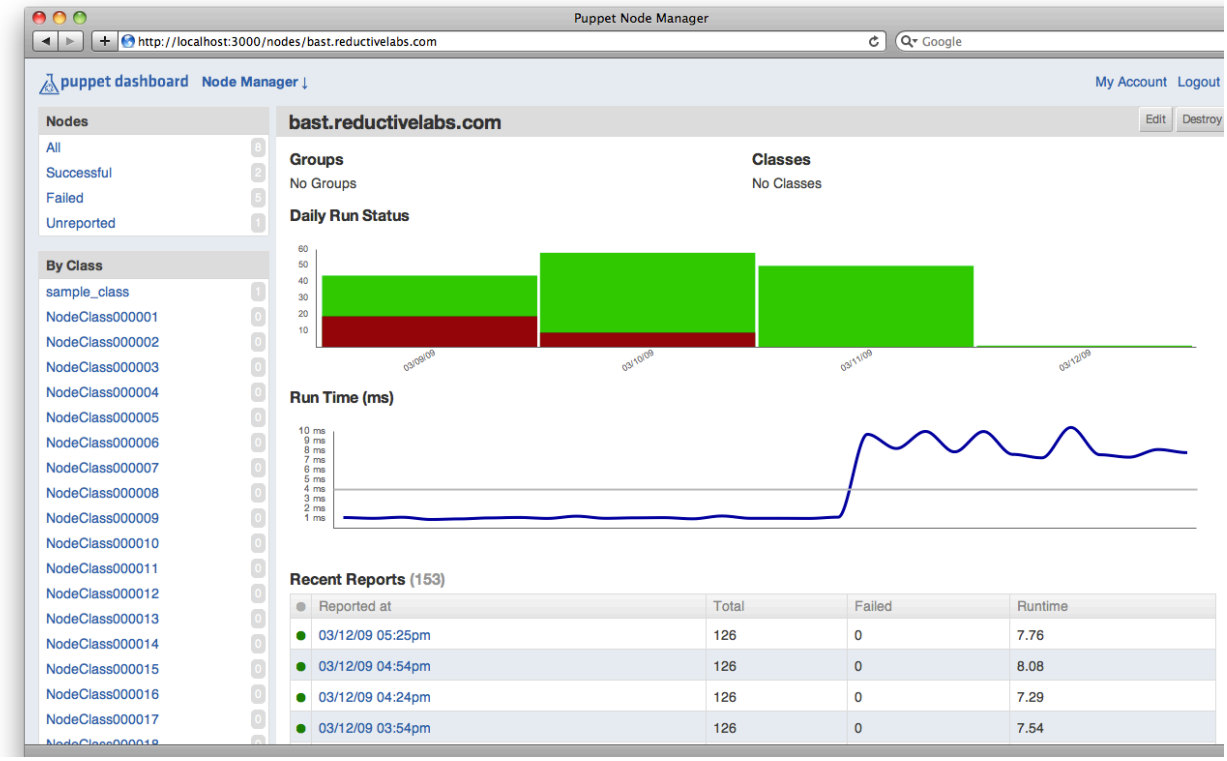
The Marionette Collective AKA mcollective is a framework to build server orchestration or parallel job execution systems.
Primarily is used as a means of programmatic execution of Systems Administration actions on clusters of servers.
MCollective and what does it allow you to do

- Interact with small to very large clusters of servers
- Use a broadcast paradigm for request distribution. All servers get all requests at the same time, requests have filters attached and only servers matching the filter will act on requests. There is no central asset database to go out of sync, the network is the only source of truth.
- Break free from ever more complex naming conventions for hostnames as a means of identity. Use a very rich set of meta data provided by each machine to address them. Meta data comes from Puppet.
- Comes with simple to use command line tools to call remote agents.
- Ability to write custom reports about your infrastructure.
- A number of agents to manage packages, services and other common components are available from the community.

The mcollective operates in a reversed logical approach the servers are installed on the hosta and the client is installed in central point that will issue the control so for the Open-DAI project the client is installed in the puppet master and the server is automatically installed and configured on each new VM created.

2.4.4 Puppet Dashboard

Puppet Dashboard is an open source web console for Puppet, which can analyze reports, browse inventory data, and assign classes to nodes.
Puppet Dashboard provides a visual snapshot of important information. The dashboard shows you status of recent puppet runs, including a chart of recent run failure percentages, and an activity feed of recent changes to nodes in the system.

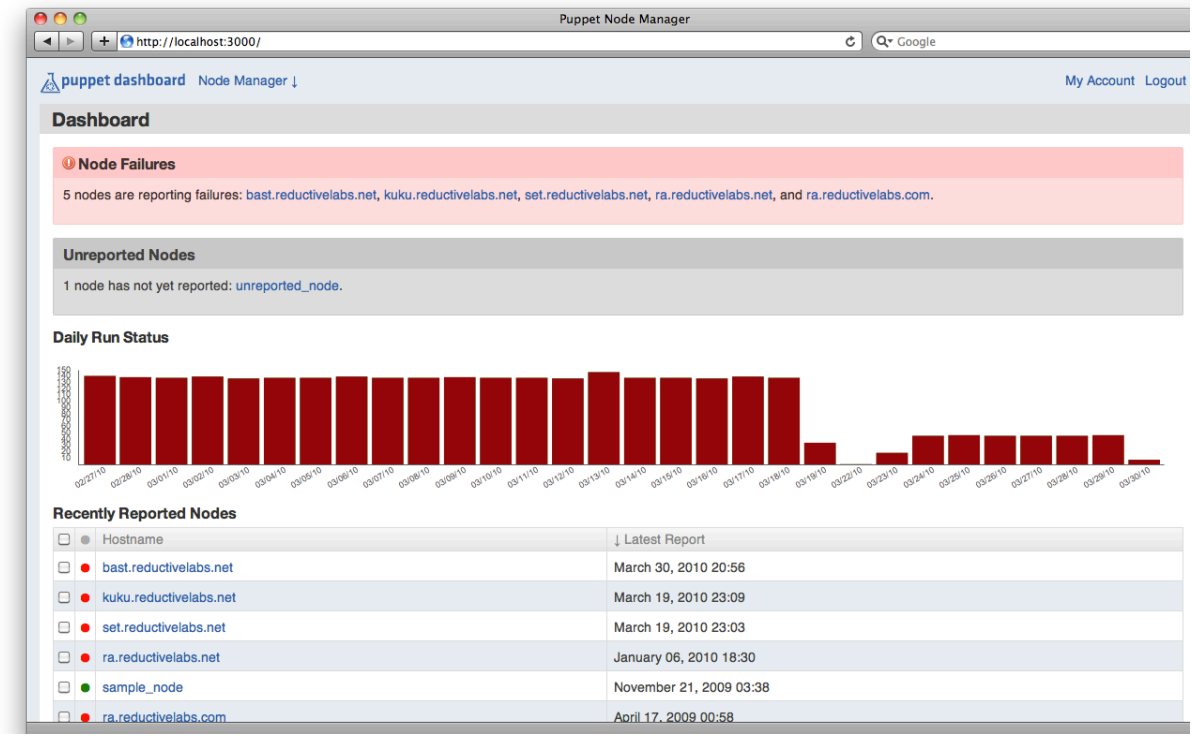


The node view of the Puppet dashboard provides status information and reporting capabilities for installed nodes, including:

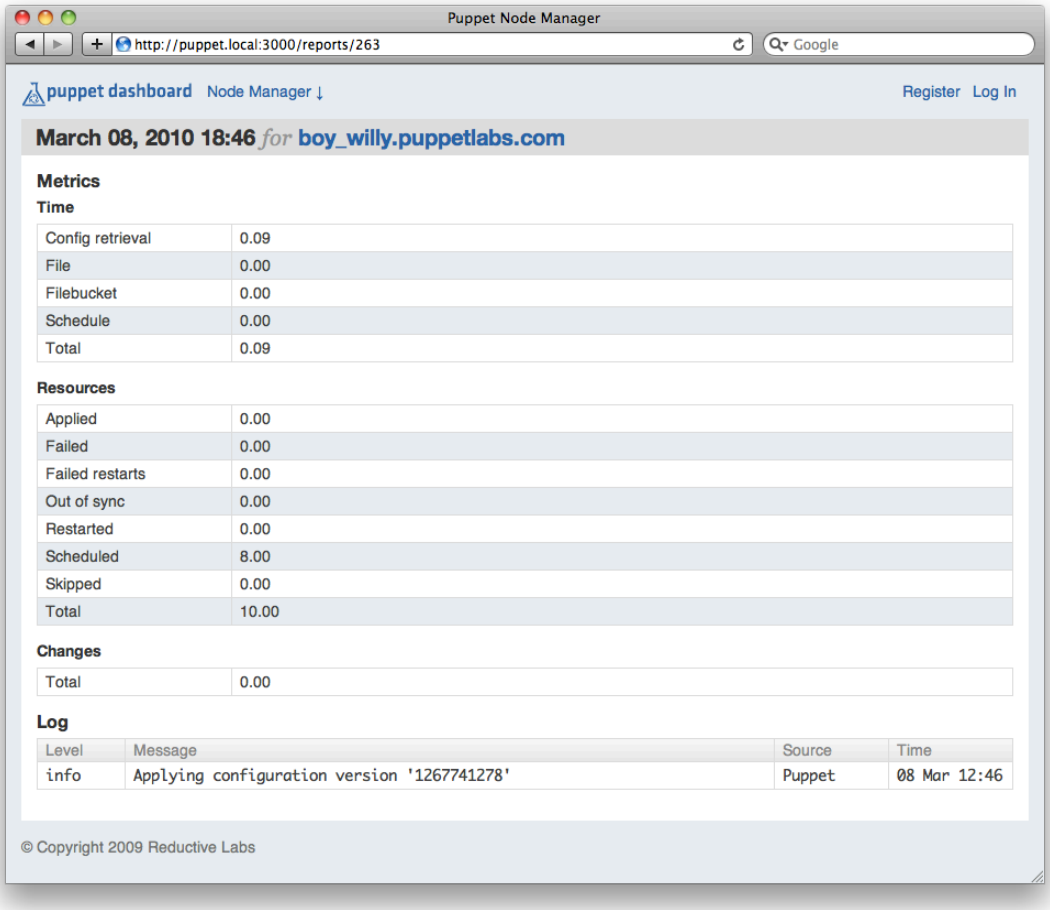
- The current configuration of a node
- Status information for recent Puppet runs
- Report information for recent Puppet runs
- A graph of run time for recent Puppet runs
- A list of configuration changes made

The node view also provides the ability to do basic node management tasks:

- Nodes can be updated or removed
- Classes and parameters can be applied to nodes directly or inherited from groups

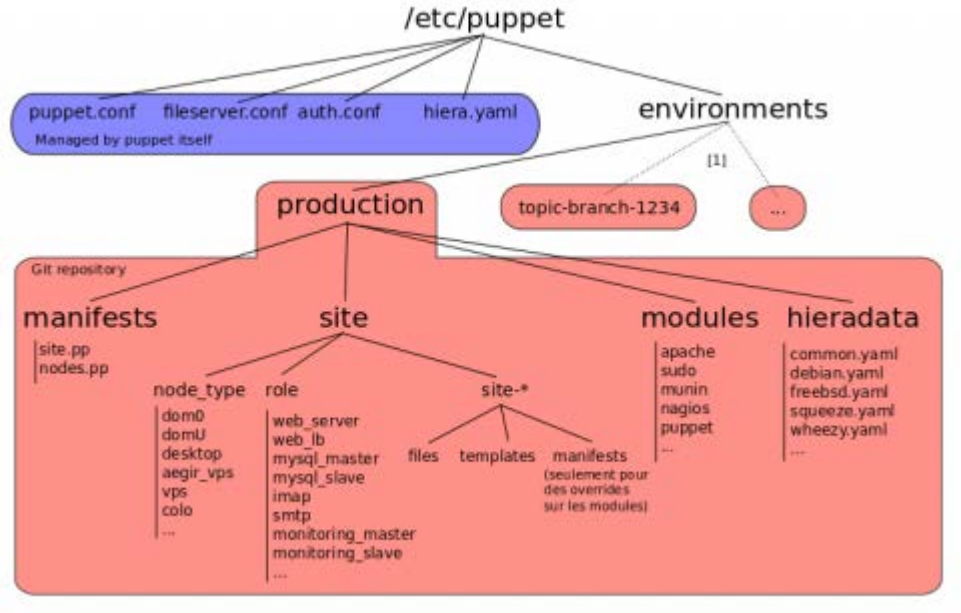


The Report View provides report information on a given Puppet run, on a node by node basis. The reports show benchmark information, statistics about the number of resources the run affected, and full report logs. As the reports generated by Puppet grow to provide even more detailed information, so too will the displays here grow to show that information.



2.4.5 Project environment

The following figure explains the approach used in managing the modules in regard of the environment using the versioning system on the red area.



And the project tried to follow these guidelines:

- Files directly in /etc/puppet should be managed by Puppet itself. This way, you can easily bootstrap a new puppet master if/when needed.
- Modules should extract values for variables from an external store, instead of expecting to use global variables. Use Hiera for this. The abstraction of data from modules makes code a lot easier to understand and to maintain.
- Hiera data files should be inside your repository, and part of environments. Changing data, especially in a hierachichal data store, must be tested out. Having data files in the repository will carry all files with your modifications over to a new environment so that is possible to test that change out without it having an impact on production. The hiera.yaml file will decide where it gets information from. Is possible to use the $environment variable in the path.
- Main manifest, site.pp, and node definitions (if kept inside a manifest) should also be part of environments. Is possible to place the site.pp file in the environment directory by adding a line to your puppet.conf file that looks like this: `manifest=$confdir/environments/$environment/manifests/site.pp`.
- modules should be split in two directories: "modules" should contain generic modules that could very well be pushed to a public repository, while "site" should contain site-specific modules and module overrides. This will possibly not apply to every organisation, but splitting things in two makes it a lot easier to contribute modifications to open sourced modules. It also helps in creating a psychological barrier between "general tools for managing different services" and "how do *we* manage things".

2.4.5.1 Base puppet's node

The project defined for the puppet site, all nodes as default nodes. This is a stripped down version of the site.pp (omitting defaults and other stuff irrelevant to this example):

```
import "base"

node default {
    include base
}
```

There is so defined a new manifest called base.pp. This class is applied to all nodes and is where the dynamic configuration happens. The stripped down version of base.pp looks something like this:

```
class base {
    ...
    # Includes that apply to all machines
    ...

    # role-specific includes
    case $role {
        'somerole': {
            include somerole
        }
        'otherrole': {
            include otherrole
        }
    }
}
```

At the top of the base class, are included configurations that apply to all machines. Stuff like ntp, sshd, yum, etc. The case statement then includes any additional modules on a per-role basis.

To recap, we have a CloudStack template that we can deploy that will automatically check into the puppetmaster and receive the configuration for whatever $role we specify in the user-data.

One of the big downsides to this approach is that the only way to set user-data for an instance is by calling the deployVirtualMachine or updateVirtualMachine commands via CloudStack's API.
This forced the development of an alternative interface not being able to give the end user the CloudStack one where the creation of the VM is already implemented.

The default configuration for all machines is described below

```
    # Includes that apply to all machines
    package {
      curl:       ensure => present;
      wget:       ensure => present;
      augeas:           ensure => present;
    }
    class { 'mcollective':
      stomp_server         => "${puppet_master}",
      server               => true,
      client               => false,
      mc_security_provider => 'psk',
      mc_security_psk      => 'xxxx',
      stomp_port => 6163,
      stomp_passwd => 'xxx',
        fact_source => 'yaml',
    }

      group {
            zabbix :
                  ensure => present
      }
      user {
            zabbix :
                  ensure => present,
```

```
                  managehome => true,
                  gid => 'zabbix',
                  require => Group['zabbix'],
                  comment => 'Zabbix user'
      }
      package { 'zabbix':
                        ensure   => present,
                        source      =>  "http://${puppet_master}/zabbix-2.0.2-
1.el6.x86_64.rpm",
                        provider => 'rpm',
                        before => Anchor['zabbix::base'],
                  }

      anchor { 'zabbix::base': }
      package { 'zabbix-agent':
                        ensure   => present,
                        source      =>  "http://${puppet_master}/zabbix-agent-
2.0.2-1.el6.x86_64.rpm",
                        provider => 'rpm',
                        require => Package['zabbix'],
                  }
      file {"/etc/zabbix/zabbix_agentd.conf":
            ensure  => "present",
            content => template( "base/zabbix_agentd.conf.erb" ),
      }
      file {"/etc/zabbix/zabbix_agent.conf":
            ensure  => "present",
            content => template( "base/zabbix_agent.conf.erb" ),
      }

      augeas { "zabbix-agentd-conf":
                  lens    => "Shellvars.lns",
                  incl    => "/etc/zabbix/zabbix_agentd.conf",
                  changes                        =>                          "set
/files/etc/zabbix/zabbix_agentd.conf/Server 10.1.1.102",
                  }

      class {'backup':}
    notice ("class base included")
```

To better explain each host will:

- Install CURL, WGET and Augeas
- Configure mcollective accordingly to the stated values
- Install the zabbix client to enable automatic monitoring of the host
- Configure the backup server data

With these operations we are guaranteed that each host created in the environment is:

- Connected to the puppet master
- All its data are stored in the puppet dashboard
- Is connected to the monitoring server
- Can connect to the backup server

After this depending on the role assigned puppet will morph the host into its own architectural role. Some of the most relevant roles are described in the following chapters.

All these operations are done automatically without human intervention.

2.4.5.2 JBoss

JBoss is distributed by RedHat as a ZIP archive.
Puppet will have to download it from the common repository host, unpack in a standard folder (tipically /opt/jbossas) and configure it.

JBoss has the peculiarity that has to be used in clustered configuration and this require to have a brief explanation on how the clustering is implemented.
In the project JBoss AS7 will be configured in domain mode and enable clustering so we could get HA and session replication among the nodes.
A managed domain spans over multiple hosts with centralized administration and management policies. Each host that runs in domain mode can contain several server instances that belong to one domain. The domain mode is one of two possible operating modes of the EAP 6 respectively the JBoss AS 7.

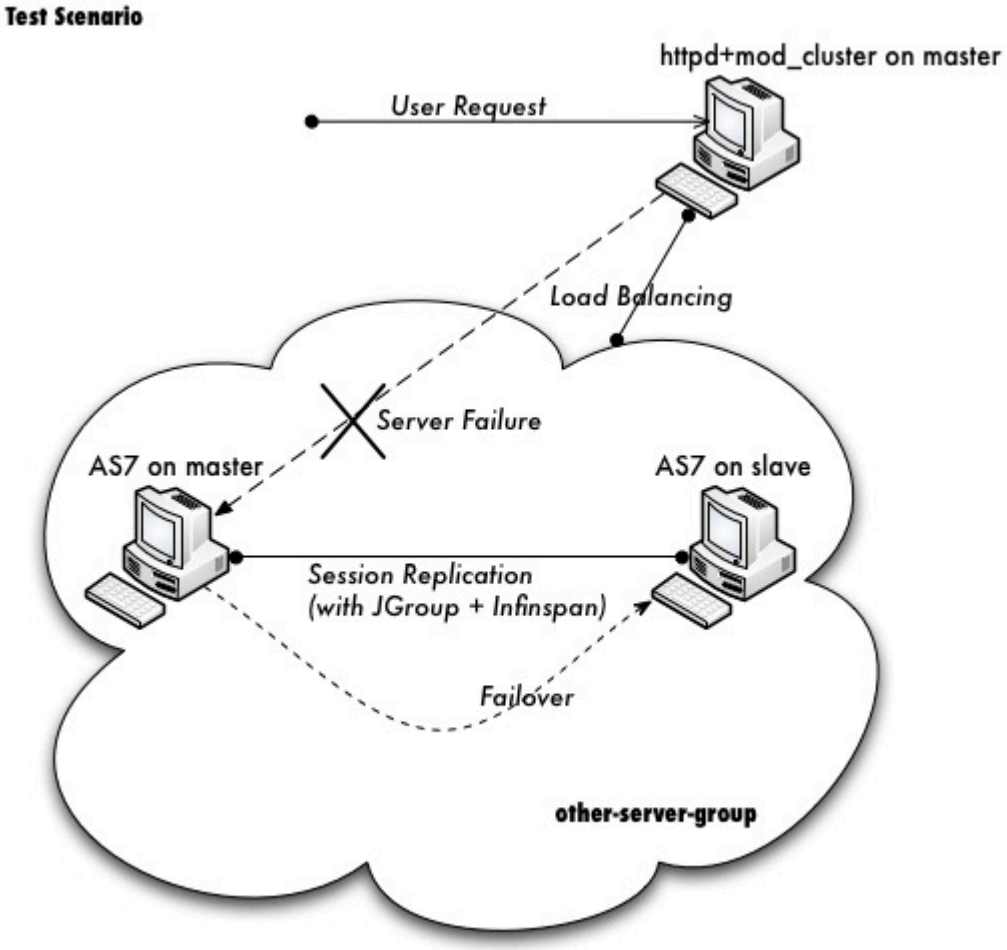
The other mode is the standalone mode. A JBoss application server, which operates in the standalone mode, is an independent process. This mode is pretty similar to the previous versions of the JBoss application server. The different operating modes have no influence to the capabilities of the application server, but rather how to manage one or multiple servers. So, the domain mode is completely independent from clustering. But it provides many useful features for managing clustered environments such as handling of deployments for multiple instances.

For example the JBoss AS 7 running in standalone mode does not support any more the previous farming deployment capabilities. Now, this can be done with the domain mode capabilities and of course there are more features such as starting and stopping servers from one single console that makes it easier to handle clustered environments.

A domain can consist of multiple physical or virtual hosts. Each host requires a host controller that is responsible for managing the lifecycle of the servers. Every domain needs one domain controller for centralized management tasks. One of the host controllers or a dedicated instance can act as domain controller. In the project we use one node to be at the same time the domain controller and a host even if a more modular architecture advises to have a dedicated host as domain controller.

This project choice is motivated by having a reduced amount of VM in the environment since each VM represent a potential cost element in the general project offering.
In the domain mode it is possible to manage multiple server instances on one host. That is why every host needs a separate host controller, which controls the server instances on that host. The domain controller interacts with the host controllers to manage the server instances. Every server instance of a domain belongs to a virtual server group. The idea is, that all servers of the same server group, perform the same tasks. When you deploy an application you will not deploy it to a server, you will deploy it to a whole server group. It is also possible to mange different server groups in one domain, e.g. production, staging or a test sever group.

The controller and the server instances of each host are separate JVM processes, which will be monitored by a process controller. The process controller is also a separate JVM process that is responsible for spawning the other processes and monitoring their lifecycle. If, the host controller process crashed, the process controller will start up the host controller and each server that is configured with the auto-start parameter. However, if the server process of a server instance is terminated unexpectedly, the process controller will not restart the server instance automatically. For this, an external monitoring is necessary.



The server configuration of the domain will be according to the policies of the domain configuration. These configurations are centralized in the domain.xml file of the domain controller. The domain.xml is located at domain/configuration/. It includes the main configuration for all server instances. This file is only required for the domain controller. A physical host in a managed domain will be configured with the host.xml file which is located at domain/configuration too. This file includes host specific configuration such network interface bindings.
In the domain.xml file we will define a cluster-ha server group for the server instances that will belong to this group. The default configuration includes already two server groups. We will replace the configuration with the following server group configuration for our domain:

```
<domain xmlns="urn:jboss:domain:1.3">
    ...
    <server-groups>
        <server-group name="cluster-ha" profile="ha">
            <jvm name="default"/>
            <socket-binding-group ref="ha-sockets" />
        </server-group>
    </server-groups>
</domain>
```

Each server group needs a unique name and a reference to one profile of the domain configuration. The default configuration includes four preconfigured profiles:

- default – Support of Java EE Web-Profile plus some extensions like RESTFul Web Services or support for EJB3 remote invocations
- full – Support of Java EE Full-Profile and all server capabilities without clustering
- ha – default profile with clustering capabilities
- full-ha – full profile with clustering capabilities

A profile contains the configuration of the supported subsystems that is added by an extension. We choose the ha profile as our application does not require additional technologies except the Java EE web profile technologies and cluster capabilities.
The referenced profile will be assigned by the server group to one socket-binding group. A socket-binding group references to logical interface names instead direct to the interfaces of a host. These logical interfaces are defined in the <interfaces> section of the domain.xml configuration file. The logical interfaces and the socket-binding group represent the socket addresses of each host.

In addition to the profile and the socket-binding group configuration of a server group, the domain controller can provide a set of system-wide properties and a reference to a JVM configuration, which is located in the host.xml configuration file.

For the domain controller we use the preconfigured host-master.xml configuration file, that includes the necessary configuration for the host, e.g. the physical network binding of the management interface or the JVM configuration. With the following configuration in the host-master.xml file, the host becomes the domain controller.

```
<host name="master" xmlns="urn:jboss:domain:1.3">
    ...
    <domain-controller>
       <local/>
    </domain-controller>
    ...
</host>
```

The last step before we start the domain controller is to create a management user for the domain controller. This user is necessary when the host controller needs to establish a connection to the domain controller. For this there is an add-user.sh script in the bin directory of the JBoss AS distribution.

All these configuration are done by puppet that uses the CLI to execute the operations.
All the configuration files of JBoss are XML files that require adding new nodes and writing specific data and there aren't in the open source ecosystem tools that can manage these files with the needed accuracy (not even Augeas that should be the one used for this task: the project dedicated lot of time in evaluation its usage without good results)

We will set the physical network bind address to the host configuration with the `jboss.bind.address.management` property. The management interface must be reachable for all hosts in the domain in order to establish a connection with the domain controller.

```
./domain.sh \
   --host-config=host-master.xml \
   -Djboss.bind.address.management=192.168.0.1
```

After the domain controller is configured and started, the next step is to setup the hosts. On each host we need also a fresh copy of the JBoss AS distribution.

This calls for the implementation of two different puppet class for the domain master node and for the slaves ones.

The main configuration for the hosts is the host.xml file which is, as already mentioned, located at domain/configuration/. The starting point will be using the preconfigured slave-host.xml file for the host controller. This file is also located inside the domain configuration directory of the JBoss AS 7 distribution. The first thing is to choose a unique name for each host in our domain to avoid name conflicts. Otherwise, the default is the host name of the server.

```
<host name="slave" xmlns="urn:jboss:domain:1.3">
    ...
</host>
```

As already mentioned before, the host controller needs to know how to establish a connection to the domain controller. The following configuration on the host specifies where the domain controller is located. Thus, the host controller can register to the domain controller itself. The `remote` tag must include the username and the security realm of the domain controller.

```
<host name="master" xmlns="urn:jboss:domain:1.3">
    ...
    <domain-controller>
       <remote                              host="${jboss.domain.master.address}"
port="${jboss.domain.master.port:9999}" security-realm="ManagementRealm"/>
    </domain-controller>
    ...
</host>
```

It will be needed to create a user also for the slave host. Notice that the username must be equal to the name given in the slaves host element. That mean for each additional host you need a user.
`./add-user.sh`

```
Enter the details of the new user to add.
Realm (ManagementRealm) :
Username : slave
Password : 123123
Re-enter Password : 123123
About to add user 'slave' for realm 'ManagementRealm'
Is this correct yes/no? yes
```

In slave we need to configure host.xml for authentication. We should change the security-realms section as following:

```
<security-realms>
   <security-realm name="ManagementRealm">
       <server-identities>
           <secret value="MTIzMTIz="/>
       </server-identities>
       <authentication>
           <properties          path="mgmt-users.properties"          relative-
to="jboss.domain.config.dir"/>
       </authentication>
   </security-realm>
</security-realms>
```

We've added server-identities into security-realm, which is used for authentication host when slave tries to connect to master. Because the slave's host name is set as 'slave', so we should use the 'slave' user's password on master. In secret value property we have 'MTIzMTIz=', which is the base64 code for '123123'.

All these steps are done with the help of external resources of puppet to synch data from different hosts without human intervention.

The steps of the cluster creation are the following:

1. Create the master node
2. Creating the slave node
3. Adding the external resource of the slave user
4. Running puppet a second time on the master to apply the external resources
5. Running puppet a second time on the slave to enable the host and link it to the cluster

2.4.5.3 MySQL

The installation of MySQL is quite straightforward since is a package present in the distributions repository
The complexity of the installation id that it require a mechanism to get it in synch with the other hosts installation.
The problem of puppet is that:
It describe the final state of the machine and not the process to reach that state
Is not thought to be completely automatic in a distributed environment.

To better explain many software can have a DB on an external host; to do this on puppet the sys op can add some classes to the nodes definition (adding on the DB host a class that define the new DB and than using the DB data to complete the current node installation) and run the puppet agent in the correct order, but if you want to get rid of the human intervention to create the installation of an application with the specification that it can describe that it will need, things start to become complex for puppet is lacking the orchestration aspect.
The human implicitly execute a workflow:

1. He install the application host
2. Add the database on the DB host
3. Finishes the installation on the application host only once the DB is present

To overcome these problems the strategy that the project used is the following:
Using the puppet feature of external resources that is a feature that enable one class to define a resource giving it a tag and the possibility for another class to collect all resources of a specific type with a precise tag.
Another feature is to condition the execution of an installation step to a check.

Doing so at one step during the application installation the class will define a resources with the data needed to create the DB and the next step will be conditioned to the ability to connect to the DB.

So all the puppet run in the application host will stop failing in the DB connection thus suspending the installation. This will give time for puppet to run in the DB node where it will have a step that collect all the DB creation resources (at that time zero so no operation) and it will execute that operation (unless the DB is present).

Another installation step of MySQL are the tuning aspect.

2.4.5.4 WSO2 API-Manager

This piece of software from WSO2 is one of the latest tools available in the WSO2 suite and has been added in the Open-DAI architecture as a possible integration/substitution of the governance registry.
This tool cover an important aspect of the architecture: publishing and managing the API produced by the project this tool could contribute deeply to the exploitation phase of the Open-DAI model and for these reason it has been included as a last minute technology update.
The installation follows the usual pattern defined for the WSO2 products, that come in zip archive.

Puppet have to download the archive, uncompress it and configure the various dynamic variables.

2.4.5.5 WSO2 Governance Registry

These are the steps/ instructions given to puppet to install the governance registry
1. Download Governance Registry binary file from the local repository: wso2greg-4.1.1.zip
2. Extract files

```
unzip wso2greg-4.1.1.zip
```

3. If working on a x64 system, change default x32 wrapper:

```
mv  wso2greg-4.1.1/bin/native/wrapper-linux-x86-32  wso2greg-4.1.1/bin/native/wrapper-linux-x86-32.bak
```

Configuring MySQL for Governance Registry

Using the puppet method of the external resources create the registry database and the user

```
create database regdb;
GRANT ALL ON regdb.* TO regadmin@"%" IDENTIFIED BY "regadmin";
```

Setting up configuration files for Governance Registry editing the registry.xml file located in the wso2greg-4.1.1/repository/conf directory of the deployed registry instance as shown below. Replace “mysql-server-ip”.

```
<currentDBConfig>mysql-db</currentDBConfig>

<dbConfig name="mysql-db">
<url>jdbc:mysql://mysql-server-ip:3306/regdb</url>
<userName>regadmin</userName>
<password>regadmin</password>
<driverName>com.mysql.jdbc.Driver</driverName>
<maxActive>80</maxActive>
<maxWait>60000</maxWait>
<minIdle>5</minIdle>
</dbConfig>
```

Edit the user-mgt.xml file (inside the wso2greg-4.1.1/repository/conf directory) of the deployed instance as below. Replace “mysql-server-ip”.

```
<Configuration>
...
<Property name="url">jdbc:mysql://mysql-server-ip:3306/regdb</Property>
<Property name="userName">regadmin</Property>
<Property name="password">regadmin</Property>
<Property name="driverName">com.mysql.jdbc.Driver</Property>
<Property name="maxActive">50</Property>
<Property name="maxWait">60000</Property>
<Property name="minIdle">5</Property>
</Configuration>
```

Use puppet to place the MySQL Java connector JAR in the wso2greg-4.1.1/repository/components/lib directory.

Edit file carbon.xml in wso2greg-4.1.1/repository/conf/ to include domain name as follows:

```
<ServerURL>https://cos-7-dai.dev-cloud-open-dai.eu:${carbon.management.port}${carbon.context}/services/</ServerURL>
```

To complete the installation WSO2 can be run in setup mode

```
export JAVA_HOME=/usr/java/jdk1.6.0_32
export PATH=$JAVA_HOME/bin:$PATH:.
export CLASSPATH=$JAVA_HOME/lib

./wso2greg-4.1.1/bin/wso2server.sh -Dsetup
```

2.4.5.6 WSO2 BPS

This WSO2 tool comes in a zip format; since is complex to manage zip format in puppet in consideration of versioning and updates the project decided to generare a simple rpm out of the official distribution so that it will be easier to proceed with the installation.

The puppet class will thus ensure the presence of the BPS package.
Afterwards will declare an external resources so that the SOA MySQL db will be able to create the database and the user.

The equivalent commend are:

```
create database bpsdb;
GRANT ALL ON bpsdb.* TO bpsadmin@"%" IDENTIFIED BY "bpsadmin";
```

From the wso2bps-2.1.2/dbscripts/bps/bpel folder puppet will then execute the loading of the mysql.sql script to configure the database:

Setting up configuration files for Business Process Server

Edit file 'datasources.properties' inside wso2bps-2.1.2/repository/conf directory with the following properties. Replace “mysql-server-ip”.

```
synapse.datasources.bpsds.driverClassName=com.mysql.jdbc.Driver
synapse.datasources.bpsds.url=jdbc:mysql://mysql-server-ip:3306/bpsdb
synapse.datasources.bpsds.username=bpsadmin
synapse.datasources.bpsds.password=bpsadmin
```

Edit the registry.xml file located in the wso2bps-2.1.2/repository/conf directory of the deployed registry instance as shown below. Replace “mysql-server-ip”.

```
<currentDBConfig>mysql-db</currentDBConfig>

<dbConfig name="mysql-db">
<url>jdbc:mysql://mysql-server-ip:3306/bpsdb</url>
<userName>bpsadmin</userName>
<password>bpsadmin</password>
<driverName>com.mysql.jdbc.Driver</driverName>
<maxActive>80</maxActive>
<maxWait>60000</maxWait>
<minIdle>5</minIdle>
</dbConfig>
```

Edit the user-mgt.xml file (inside the wso2bps-2.1.2/repository/conf directory) of the deployed instance as below. Replace “mysql-server-ip”.

```
<Configuration>
...
<Property name="url">jdbc:mysql://mysql-server-ip:3306/bpsdb</Property>
```

```
<Property name="userName">bpsadmin</Property>
<Property name="password">bpsadmin</Property>
<Property name="driverName">com.mysql.jdbc.Driver</Property>
<Property name="maxActive">50</Property>
<Property name="maxWait">60000</Property>
<Property name="minIdle">5</Property>
</Configuration>
```

Copy the MySQL driver to corresponding directory: $HOME/wso2bps-2.1.2/repository/components/lib

Edit file carbon.xml in wso2bps-2.1.2/repository/conf/ to include domain name as follows:

```
<ServerURL>https://cos-6-dai.dev-cloud-open-
dai.eu:${carbon.management.port}${carbon.context}/services/</ServerURL>
```

The tool has an option to automatically set up the DB

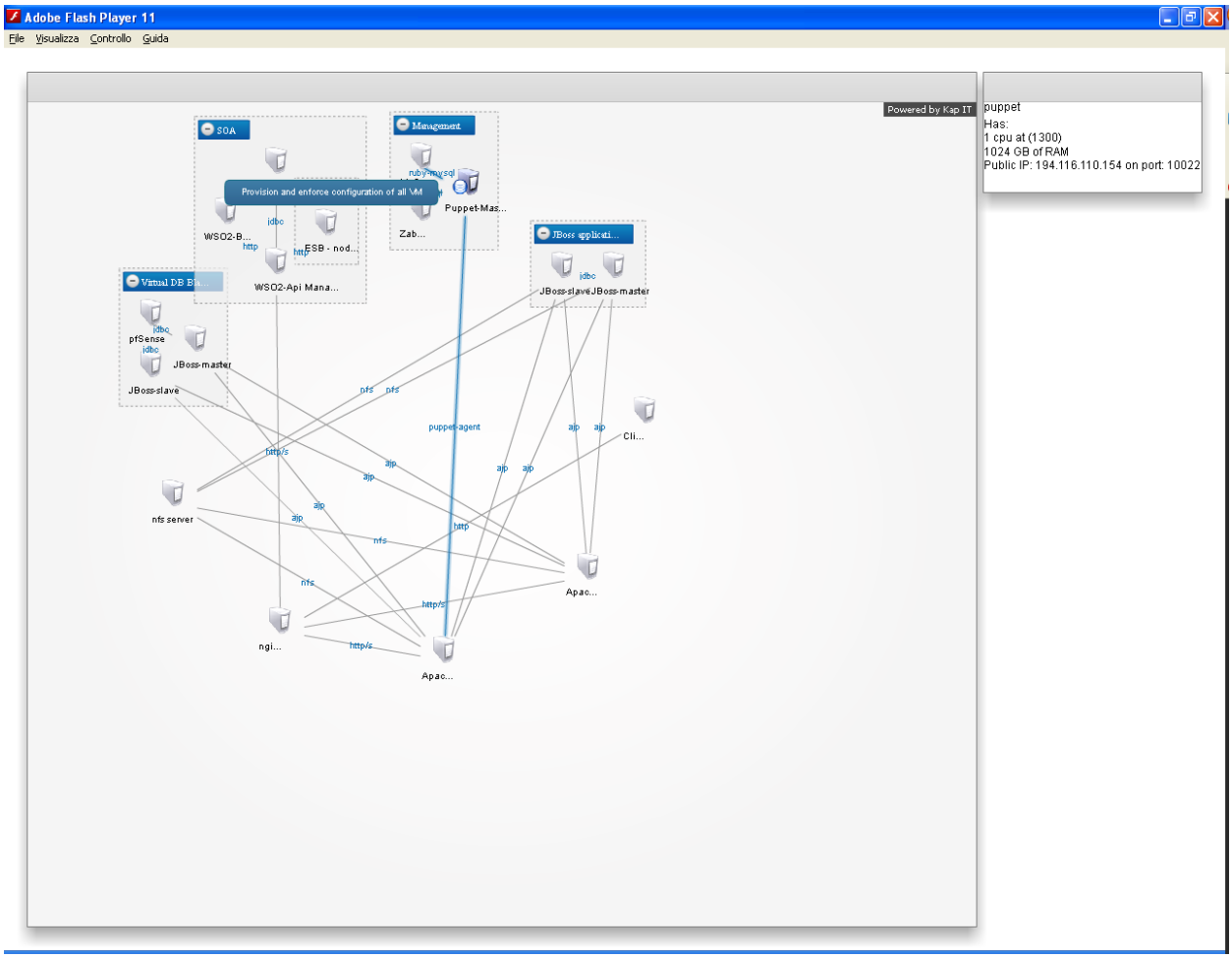
```
export JAVA_HOME=/usr/java/jdk1.6.0_32
export PATH=$JAVA_HOME/bin:$PATH:.
export CLASSPATH=$JAVA_HOME/lib

./wso2bps-2.1.2/bin/wso2server.sh -Dsetup
```

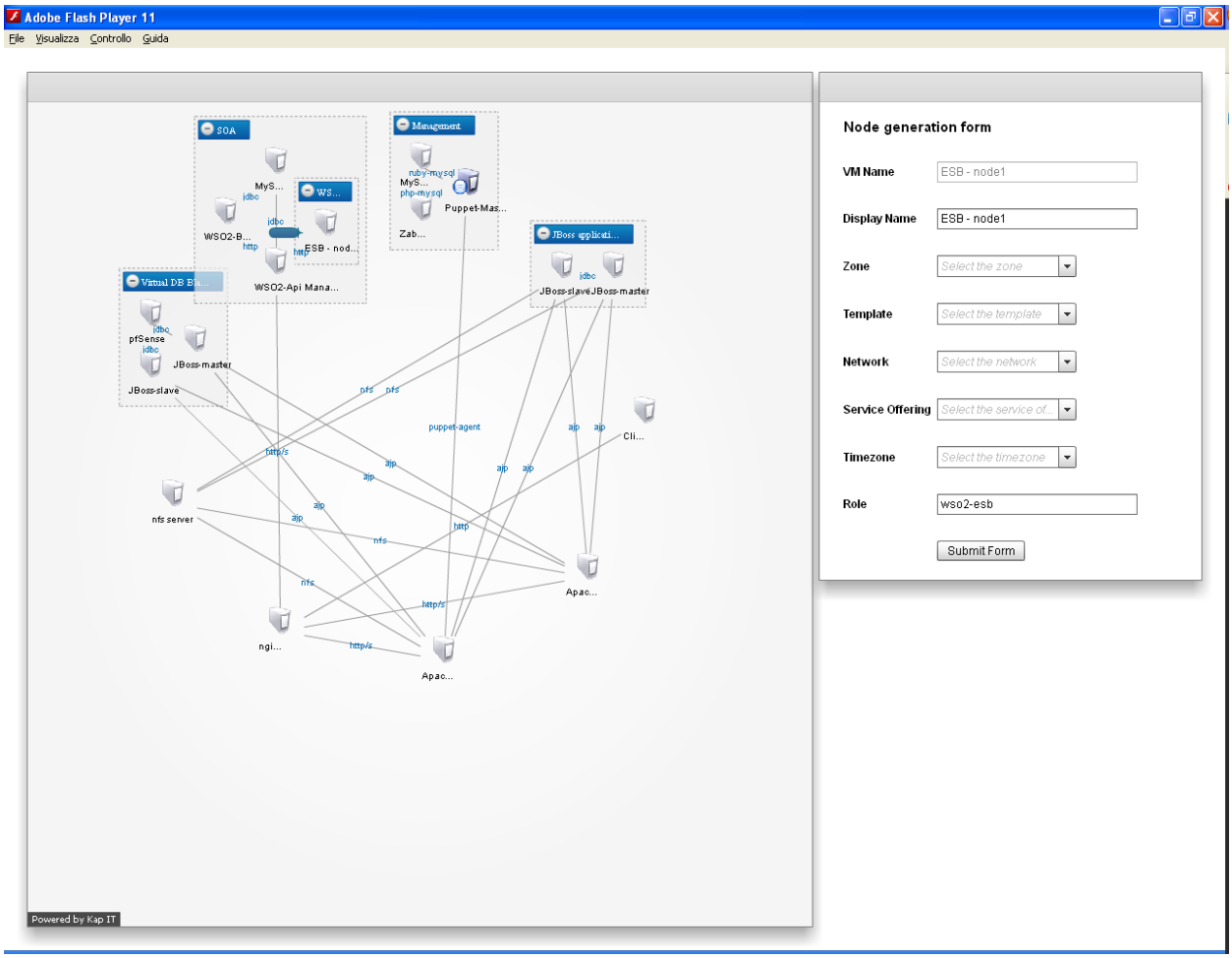
2.5 Open-DAI management console

The goal of the project is, on one end to leave the end user the power to configure the architecture according to his own needs, on the other to give him a wrapped and easy tool to use that mask the complexity of the infrastructure.
To reach this goal the project developed a web/Flash interface to enable the user in managing the platform.
With this interface the user is masked from the cloud characteristics and he just have the architecture graph of the VM shown in a way that immediately clear to understand and that expose the relationships between the hosts.
The version of the application released as the first deliverable implement the ability to:

- be able to create the host that the application knows that does not yet exist in the environment since it query the cloud platform for its status
- get the information on the status of the node/host

The figures below show these two functionalities.


Adobe Flash Player 11
File Visualizza Controllo Guida
Powered by Kap IT
puppet
Has:
1 cpu at (1300)
1024 GB of RAM
Public IP: 194.116.110.154 on port: 10022
SOA
Management
Provision and enforce configuration of all VM
Puppet-Mas...
Zab...
WSO2-B...
ESB - nod...
WSO2-Api Mana...
JBoss applicati...
JBoss-slaveJBoss-master
Virtual DB Ba...
pfSense
JBoss-master
JBoss-slave
nfs server
Apac...
ngi...
Apac...
Cli...
idbc
http
nfs
puppet-agent
ajp
http/s



In future versions of the application it will be possible to:

- Start or stop the host
- Invoke operations on the host depending on the nature/role of it

2.6 Open-DAI network detail

The architecture states that each domain will make use of 3 public IP addresses:
One dedicated to nginx proxy (used since CloudStack proxy is not able to manage SSL connections as needed in the project)
One dedicated to the PfSense for creating the VPN connection towards the legacy database
One dedicated to the management consoles
All the addresses will be resolved by the DNS managed in the CSI Piemonte environment dedicated to the project to allow for the maximum freedom for the pilots.

2.6.1 Nginx

The nginx will be the proxy for all the services offered by the platform.
This independently on which will be the tool exposing the services.
Considering that the domain will be assigned a DNS domain like www.country.cloud-open-dai.eu the starting policy (that could be changed as a normal operation task) given out of the box will be summed up by the table below.

Internally each domain has its won name resolution based on a local naming service.

Public URL	Proxyed to
www.country.cloud-open-dai.eu	apache.domain.local:80
vdb.country.cloud-open-dai.eu	apache.domain.local:8080 this will in turn balance to jboss.domain.local:8080
api.country.cloud-open-dai.eu	api-manager.domain.local:9443 (the API store)
esb.country.cloud-open-dai.eu	esb.domain.local:9443
bps.country.cloud-open-dai.eu	bps.domain.local:9443

Nginx will also be configured to make a reverse proxy SSL towards the services that need to interact using that protocol with the end user

2.6.2 Management console

For the management console we'll use a single IP to place under the same URL all the consoles that the different tools publish to allow the end user to access them.
The strategy is to port map all the request to the different hosts and to use a different set of port to point to all the services that could use the same port (being on different hosts).
The table below sum up the port mapping adopted.

Public URL	Mapping
management.country.cloud-open-dai.eu	puppet.domain.local:80
management.country.cloud-open-dai.eu:9999	jboss.domain.local:9999
management.country.cloud-open-dai.eu:19999	api-manager.domain.local:9999
management.country.cloud-open-dai.eu:29999	esb.domain.local:9999
management.country.cloud-open-dai.eu:39999	bps.domain.local:9999
management.country.cloud-open-dai.eu:3000	puppet.domain.local:3000 (puppet dashboard)
management.country.cloud-open-dai.eu:81	zabbix.domain.local:80