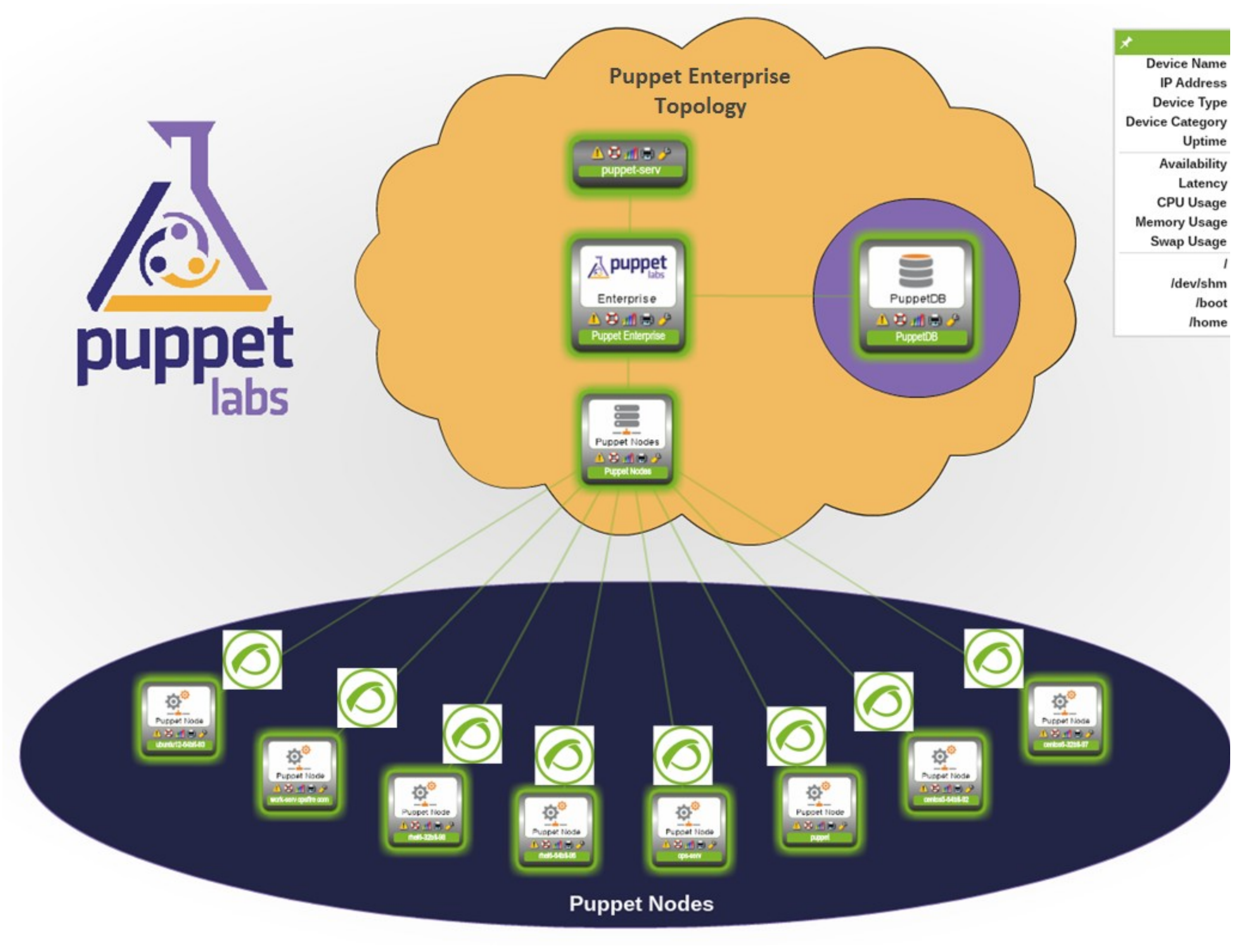
PandoraFMS' Agents automated installation with Puppet.

This article is going to serve as a foundation for automating the automatic process of installing Pandora FMS' agents on multiple systems simultaneously: either Windows, Linux or Unix. The tool that has been used for this process is Puppet.

Puppet is a new generation OpenSource solution for server automation.
It features a declarative language used to express the system configuration, a client-server scheme for distribution, and a library to run that configuration.
Among its various activities, it allows us to control the execution of different services on a machine, centralized control of configuration files, installation package management, management of presence and availability of mount points, user and group management and deployment of machines.



In this article we will show the declarative language that should be configured in Puppet (Puppet's modules) with the prerequisites that in the net where you will deploy agents, has already installed the Puppet's server and clients on the different computers.

As indicated, Puppet makes the deployment through “manifests”. This manifests are written in Puppet's own code and tell Puppet clients the process that must be performed to accomplish the ultimate goal, which in our case will be to install, configure and start the agents.

Puppet's main folder is by default /etc/puppet. This folder may vary depending on Puppet's installation path.

Inside /etc/pupept/files, where all the static files are placed to be sent to the clients we are going to introduce the necessary files for be deployed in the clients, we are setting the necessary files for the agent deployment:

_ Software agent linux and/or windows (agent in tar.gz format)

_ Agent's configuration file that contains the configuration. One of the most important things is the IP address of the server, and the modules definition.

If we are going to deploy agents to both Windows and Linux platforms, we need to have different configuration files for each OS, because the module definition could not be different.

Inside /etc/puppet/manifests folder we will create the resources, which are the steps that Puppet will perform in the agent deployment.

First, create a init.pp file inside /etc/puppet/manifests. This file contains the resources of the module.

For a Unix agent deployment, this would be the content of the init.pp file:

```
class install_agent_unix {

File ['pandorafms_agent_unix-5.1.tar.gz file'] → Exec ['Extract tar'] → Exec [ 'Instalar Pandora
Agent'] → File ['pandora_agent.conf file'] → Service ['pandora_agent_daemon service']

file { 'pandorafms_agent_unix-5.1.tar.gz file':
owner   => root,
group   => root,
mode    => 777,
source  => "puppet:///files/pandorafms_agent_unix-5.1.tar.gz",
path => "/tmp/pandorafms_agent_unix-5.1.tar.gz"
}

exec { 'Extract tar':
 path => "/bin:/usr/bin",
 Unless => "find /tmp/pandorafms_agent_unix-5.1",
 Command => "tar -xzf /tmp/pandorafms_agent_unix-5.1.tar.gz",
 }

exec { 'Instalar Pandora Agent':
      Command => "/tmp/pandorafms_agent_unix-5.1/pandora_agent_installer –install,
}

file { 'pandora_agent.conf file':
owner   => root,
group   => root,
mode    => 644,
source  => "puppet:///files/pandora_agent.conf",
path => "/etc/pandora/pandora_agent.conf",
}

service { 'pandora_agent_daemon service':
       ensure => running,
       name => “pandora_agent_daemon”,
       enable => true,
    }
}
```

As a summary, what we are going to do is send this file, unzip it, run the installation, send the configuration file and run Pandora's agent.

If we are deploying Windows agents, the file would be instead:

```
class install_agent_wdos {

File ['PandoraFMS_windows_agent_v5.1.setup.exe file'] → Exec [ 'Instalar Pandora Agent'] → File
['pandora_agent.conf file'] → Service ['pandora_agent service']

file { 'pandorafms_agent_unix-5.1.setup.exe file':
owner   => root,
group   => root,
mode    => 777,
source  => "puppet:///modules/pandorafms/pandorafms_agent_unix-5.1.setup.exe",
path => "C:\PandoraFMS_windows_agent_v5.1.setup.exe"
}

exec { 'Instalar Pandora Agent':
      Command => "C:\PandoraFMS_windows_agent_v5.1.setup.exe /mode Silent /prefix
c:\agente_pandora,
}

file { 'pandora_agent.conf file':
owner   => root,
group   => root,
mode    => 644,
source  => "puppet:///modules/pandorafms/pandora_agent.conf",
path => "c:\agente_pandora\pandora_agent.conf",
}

service { 'pandora_agent_daemon service':
       ensure => running,
       name => “pandora_agent_daemon”,
       enable => true,
    }
}
```

Bear in mind that Windows Puppet agents are oficially supported on Windows Server version 2003, 2008. They are not on other version, but it should work.

Once we have configured the manifest, the only thing left to do is associate this manifest to every node or systems that we want this agent to be installed.

```
/etc/puppet/manifests/site.pp

An example of this configuration to apply to every node would be:

node unix_agent {

       include install_agent_unix

}

node windows_agent {

       include install_agent_wdows

}
```

There are various ways to associate the classes to Puppet nodes, in this example we have chose the following, but Puppet's administrator can do it in a different thiway, for instance to making it compatible with the actual installation of Puppet in its environment. For doing that you can use this manifests to use in your configuration.

Puppet's agents monitorization with Pandora FMS

For monitor Puppet's agents, we have a plugin in our module library of pandorafms.com http://pandorafms.com/index.php?sec=Library&sec2=repository&lng=en&action=view_PUI&id_PUI=600 that gathers the following information from a Puppet agent:

- Last run
- Time since last run
- Is out of sync? → If it's out of synchronization
- Resources changed/failed/failed to restart/restarted/scheduled/skipped
- Events failed/succeeded
- Total changes

The plugin is written in Ruby, so you will need Ruby to be installed on the system.

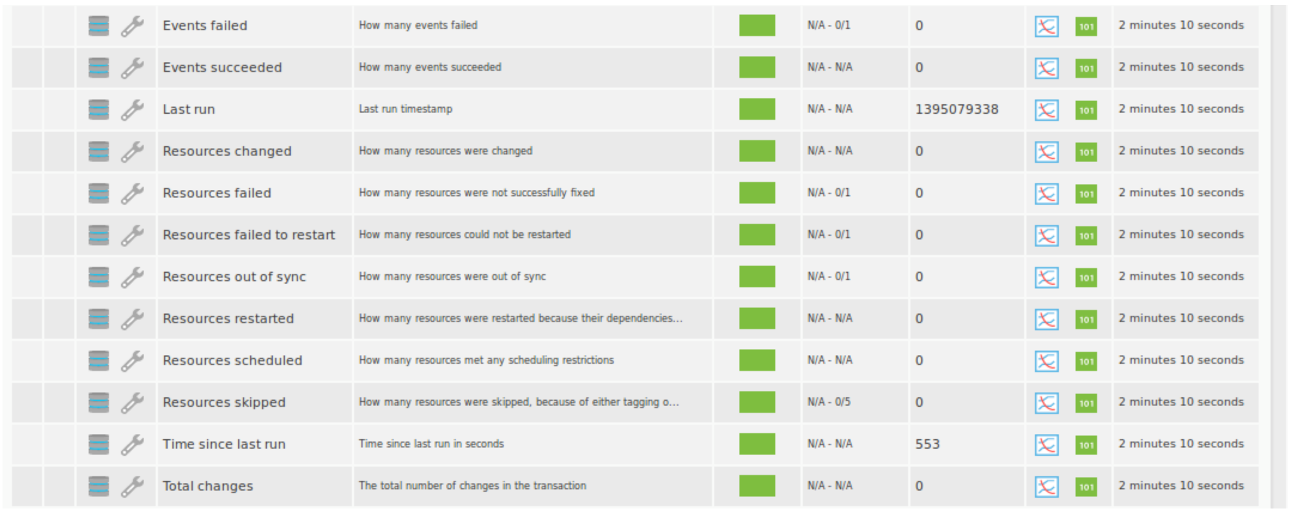
For executing the plugin we must only find the *last_run_summary.yaml* file, where the Puppet agent stores the last execution time it was ran. The file structure is like this:

```
time:
   group: 0.001692
   last_run: 1317804488
   class: 0.003929
   yumrepo: 0.020103
   service: 9.017434
   schedule: 0.004151
   cron: 0.010546
   config_retrieval: 15.0572321414948
   total: 34.9742621414947
   package: 0.588751
   filebucket: 0.000687
   file: 8.895422
   exec: 1.361625
   user: 0.01269
 resources:
   total: 194
   changed: 0
   failed: 0
   failed_to_restart: 0
   out_of_sync: 0
   scheduled: 0
   skipped: 0
 events:
   total: 0
 changes:
   total: 0
```

By default, this file can be found at */var/lib/puppet/state/last_run_summary.yaml*, so the plugin execution would be like this:

```
module_plugin <ruby_path> /etc/pandora/plugins/pandora_puppet.rb --summary-file
/var/lib/puppet/state/last_run_summary.yaml
```

In Windows, the *last_run_summary.yaml* can be found at C:\ProgramData\PuppetLabs\puppet\var\state



Events failed	How many events failed	🟩	N/A - 0/1	0	2 minutes 10 seconds
Events succeeded	How many events succeeded	🟩	N/A - N/A	0	2 minutes 10 seconds
Last run	Last run timestamp	🟩	N/A - N/A	1395079338	2 minutes 10 seconds
Resources changed	How many resources were changed	🟩	N/A - N/A	0	2 minutes 10 seconds
Resources failed	How many resources were not successfully fixed	🟩	N/A - 0/1	0	2 minutes 10 seconds
Resources failed to restart	How many resources could not be restarted	🟩	N/A - 0/1	0	2 minutes 10 seconds
Resources out of sync	How many resources were out of sync	🟩	N/A - 0/1	0	2 minutes 10 seconds
Resources restarted	How many resources were restarted because their dependencies...	🟩	N/A - N/A	0	2 minutes 10 seconds
Resources scheduled	How many resources met any scheduling restrictions	🟩	N/A - N/A	0	2 minutes 10 seconds
Resources skipped	How many resources were skipped, because of either tagging o...	🟩	N/A - 0/5	0	2 minutes 10 seconds
Time since last run	Time since last run in seconds	🟩	N/A - N/A	553	2 minutes 10 seconds
Total changes	The total number of changes in the transaction	🟩	N/A - N/A	0	2 minutes 10 seconds

Puppet's server monitoring with Pandora FMS.

For monitor Puppet's server, the main requisite is having a Pandora's agent installed in the Puppet server.

Below we illustrate how the configuration of the different modules that would have to be set in the Pandora's agent configuration file:

1.- Puppetmaster and puppetdb status

Boolean type module that will show if the puppet server is running or not. It will return 1 if possitive, or 0 if it's not.

```
module_begin
module_name Status puppetmaster
module_type generic_proc
module_exec /etc/init.d/puppetmaster status | grep running | wc -l
module_description Status puppermaster
module_end

module_begin
module_name Status puppetdb
module_type generic_proc
module_exec /etc/init.d/puppetdb status | grep running | wc -l
module_description Status puppetdb
module_end
```

2.- Puppet's manifestos configuration status

The Puppet server itself has a tool that allows us checking the status of puppet's manifestos. If we select the directory where these manifestos are stored, we can check them all.

We have two ways to monitor this. The first one, is showing us if there is an anomaly or not, and the other one is gathering an error when it is present.

```
module_begin
module_name Status manifests
module_type generic_proc
module_exec puppet parser validate /etc/puppet/manifests/ | wc -l
module_critical_inverse 1
module_description Status manifest
module_end

module_begin
module_name Manifests errors
module_type async_string
module_exec puppet parser validate /etc/puppet/manifests/
module_description Status manifest
module_end
```

3.- Check if all the Puppet agents are signed

With the next module we can check if all the agents are signed in the Puppet server, therefore working without an issue.

```
module_begin
module_name Puppet agents certs status
module_type generic_proc
module_exec puppet cert –list | wc -l
module_critical_inverse 1
module_description Status Puppet agents certs
module_end
```

4.- Puppetmaster and puppetdb CPU and RAM usage

Shows the CPU % usage and RAM of puppet services:

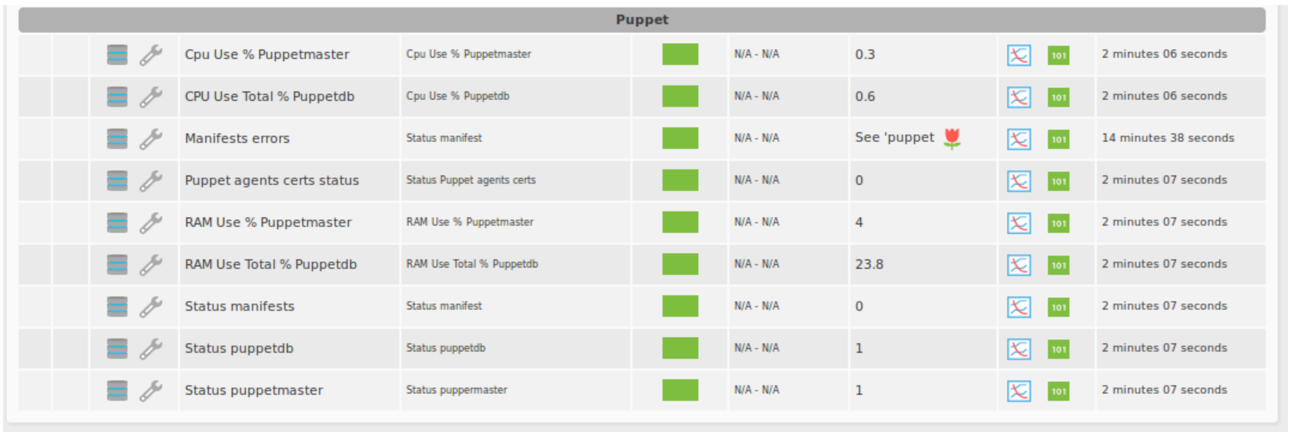
```
module_begin
module_name Cpu Use % Puppetmaster
module_type generic_data
module_exec ps aux | grep puppetmaster | grep -v grep | gawk '{print $3}'
module_description Cpu Use % Puppetmaster
module_end

module_begin
module_name RAM Use % Puppetmaster
module_type generic_data
module_exec ps aux | grep puppetmaster | grep -v grep | gawk '{print $4}'
module_description RAM Use % Puppetmaster
module_end

module_begin
module_name Cpu Use Tot % puppetdb
module_type generic_data
module_exec ps aux | grep puppetdb | grep -v grep | awk '{sum += $3} END {print sum}'
module_description Cpu Use % Puppetdb
module_end

module_begin
module_name RAM Use Total % Puppetdb
```

```
module_type generic_data
module_exec ps aux | grep puppetdb | grep -v grep | awk '{sum += $4} END {print sum}'
module_description RAM Use Total % Puppetdb
module_end
```



Puppet						
Cpu Use % Puppetmaster	Cpu Use % Puppetmaster	🟩	N/A - N/A	0.3		2 minutes 06 seconds
CPU Use Total % Puppetdb	Cpu Use % Puppetdb	🟩	N/A - N/A	0.6		2 minutes 06 seconds
Manifests errors	Status manifest	🟩	N/A - N/A	See 'puppet		14 minutes 38 seconds
Puppet agents certs status	Status Puppet agents certs	🟩	N/A - N/A	0		2 minutes 07 seconds
RAM Use % Puppetmaster	RAM Use % Puppetmaster	🟩	N/A - N/A	4		2 minutes 07 seconds
RAM Use Total % Puppetdb	RAM Use Total % Puppetdb	🟩	N/A - N/A	23.8		2 minutes 07 seconds
Status manifests	Status manifest	🟩	N/A - N/A	0		2 minutes 07 seconds
Status puppetdb	Status puppetdb	🟩	N/A - N/A	1		2 minutes 07 seconds
Status puppetmaster	Status puppermaster	🟩	N/A - N/A	1		2 minutes 07 seconds