# MOC Puppet System Admin Guide

## *Alpha Version 0.1*

Written by Gefter Mbi for NOAA MOC

## Revision Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Revision Date** | **Author** | **Description** |
| 0.1 | June, 2014 | Gefter Mbi | Document inception. Initial outline, content, some sections written. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Table of Contents

[MOC Puppet System Admin Guide](#h.7o2w5dk3r47r)

Alpha Version 0.1

[Revision Table](#h.cu3ncr9ag7t1)

[Table of Contents](#h.pv086k2j4x3d)

[Purpose](#h.nbokkj7w7urf)

[Background](#h.j6dk7y4pvwq2)

Puppet Versions [Considerations](#sdasdsdsdasds)

Puppet Server [Installation](#sdfdsffddsfsdf)

Puppet Client (2.7.19) installation on [RHEL](#cdfdfdffgfgfgfg) 6

[Puppet Client (2.7.19) installation on RHEL 5](#edddfdfdfdfffd)

[Puppet Client (2.7.1) installation on RHEL 4](#ghhhdfgfgfgfgg)

[Puppet Client (2.7.19) installation on Solaris 9](#ghhhdfgfgfgfgg)

[Puppet Client Configuration](#ghhhdfgfgfgfgg)

[Adding a new Puppet Module](#ghhhdfgfgfgfgg)

## Purpose

This document details administration of the MOC Puppet System(MPS). This will cover installation, configuration, deployment and how updates are done

## Background

Configuration management in the MOC till now has been done using Revision Control System (RCS). This has been used to manage config files like the mappings file in the Sun One Messaging system. But the MOC requires a more robust and scalable system to handle configuration and possibly installations.

Puppet can certainly handle this and some of its features include

* Provisioning
* Configuration management – operating systems and applications
* Re-useable modules to roll out configurations
* Has a dashboard where nodes, classes and groups can be added
* Dashboard has a reports section to track changes in the nodes.

## Puppet version Considerations

The latest version of puppet is 3.6.2. While the puppet master could run on this version, we have an environment with old hardware and old operating systems. Old systems include Solaris 9, Red hat Linux AS4, and Red hat EL 5. These systems would not be able to run the latest versions of puppet clients.

During the initial testing phase, puppet clients (3.6.2) and client 2.7.19 on RHEL 6 were able to connect to the puppet master running 3.6.2. These same clients on RHEL AS4 were not able to connect to the puppet server.

The new version of puppet (3.x.x) uses OpenSSL that creates certificates hashed with SHA256, while RHEL 4 and Solaris use the older version of OpenSSL that can only understand SHA1.

After extensive testing and trouble shooting, here are the acceptable versions that the MOC environment will support

Puppet Server 2.7.19 on RHEL 6

Puppet Client 2.7.19 on RHEL 6

Puppet Client 2.7.1 on RHEL AS 4

Puppet Client 2.7.1 on RHEL AS 5

Puppet Client 2.7.1 on RHEL AS 4

Puppet Client 2.7.19 on Solaris 9

## Puppet Server Installation on RHEL 6.5

* First add the EPEL repo on both the puppet client and server systems:

# rpm -Uvh <http://download.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm>

# rpm -ivh <http://yum.puppetlabs.com/puppetlabs-release-el-6.noarch.rpm>

* Enable RHEL 6 Optional Channel

<https://access.redhat.com/site/documentation/en-US/OpenShift_Enterprise/1/html/Client_Tools_Installation_Guide/Installing_Using_the_Red_Hat_Enterprise_Linux_Optional_Channel.html>

# rhn-channel --add --channel rhel-x86\_64-server-optional-6

* Use the yum-versionlock plugin to lock the package version to what you want:

# yum install yum-versionlock

* Now edit /etc/yum/pluginconf.d/versionlock.list and specify the version you want:

puppet 2.7.19

puppet-server 2.7.19

* Install a specific version of puppet:

# yum install puppet-2.7.19

# yum install puppet-server-2.7.19

* Lock the package version to the currently installed version:

yum versionlock puppet

yum versionlock puppet-server

* On the master, you should have the following directory structure.

etc/puppet/

/etc/puppet/fileserver.conf

/etc/puppet/auth.conf

/etc/puppet/puppet.conf

/etc/puppet/manifests

## Puppet Client (2.7.19) installation of RHEL 6

* First add the EPEL repo on both the puppet client and server systems:

# rpm -Uvh <http://download.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm>

# rpm -ivh <http://yum.puppetlabs.com/puppetlabs-release-el-6.noarch.rpm>

* Enable RHEL 6 Optional Channel

<https://access.redhat.com/site/documentation/en-US/OpenShift_Enterprise/1/html/Client_Tools_Installation_Guide/Installing_Using_the_Red_Hat_Enterprise_Linux_Optional_Channel.html>

# rhn-channel --add --channel rhel-x86\_64-server-optional-6

* Use the yum-versionlock plugin to lock the package version to what you want:

# yum install -y yum-versionlock

* Now edit /etc/yum/pluginconf.d/versionlock.list and specify the version you want:

puppet 2.7.19

* Install a specific version of puppet:

# yum install -y puppet-2.7.19

* Lock the package version to the currently installed version:

yum versionlock puppet

## Puppet Client (2.7.19) installation of RHEL 5

* First add the EPEL repo on both the puppet client and server systems:

# rpm -Uvh <http://mirror.chpc.utah.edu/pub/epel/5/x86_64/epel-release-5-4.noarch.rpm>

# rpm -ivh <http://yum.puppetlabs.com/puppetlabs-release-el-5.noarch.rpm>

* Enable RHEL 6 Optional Channel (This erred out on RHEL 5 but installation still proceeded.)

<https://access.redhat.com/site/documentation/en-US/OpenShift_Enterprise/1/html/Client_Tools_Installation_Guide/Installing_Using_the_Red_Hat_Enterprise_Linux_Optional_Channel.html>

# rhn-channel --add --channel rhel-x86\_64-server-optional-5

* Use the yum-versionlock plugin to lock the package version to what you want:

# yum install yum-versionlock

* Now edit /etc/yum/pluginconf.d/versionlock.list and specify the version you want:

puppet 2.7.19

* Install a specific version of puppet:

# yum install puppet-2.7.19

## Puppet Client (2.7.1) installation of RHEL AS 4

This installation needs a C compiler, gcc works fine.

* autoconf 2.60 is required to build ruby

# wget <http://ftp.gnu.org/gnu/autoconf/autoconf-2.60.tar.gz>

# tar -xzf autoconf-2.60.tar.gz

# cd autoconf-2.60

# ./configure --prefix=/usr && make && make install

* Build ruby-1.8.7-p358

# wget http://ftp.ruby-lang.org/pub/ruby/1.8/ruby-1.8.7-p358.tar.bz2

# tar -xjf ruby-1.8.7-p358.tar.bz2

#cd ruby-1.8.7-p358

#autoconf

#./configure --prefix=/usr && make && make install

If you get compile errors with LX11

Basically you need to add a flag to this link line to tell "gcc" where to find the X11 library (libX11.so). On most Linux systems, this library is found in /usr/X11R6/lib, so you'd add

./configure --prefix=/usr --libdir=/usr/X11R6/lib64/libX11.so && make && make install

./configure --prefix=/usr --with-tclConfig-file=/usr/lib64/tclConfig.sh --with-tkConfig-file=/usr/lib64/tkConfig.sh && make && make install

For successful compiling the exit status should be zero.

# echo $?

Anything other than zero is a failed compilation. Correct errors and run again.

* Make sure the path for ruby is in the PATH variable.
* Install ruby-gems 1.8.10

# wget http://production.cf.rubygems.org/rubygems/rubygems-1.8.10.tgz

# tar -xzf rubygems-1.8.10.tgz

# cd rubygems-1.8.10

# /usr/bin/ruby setup.rb

* Make sure the path for gem is in the PATH variable.
* Install puppet

# gem install puppet --version=2.7.1

Fetching: facter-2.0.2.gem (100%)

Fetching: puppet-2.7.1.gem (100%)

Successfully installed facter-2.0.2

Successfully installed puppet-2.7.1

2 gems installed

Installing ri documentation for facter-2.0.2...

Installing ri documentation for puppet-2.7.1...

Installing RDoc documentation for facter-2.0.2...

Installing RDoc documentation for puppet-2.7.1...

[root@ldap4 mbig]# puppet --version

2.7.1

## Puppet Client (2.7.19) installation of Solaris 9

* Install pkgutil.pkg. This will be used to install puppet. This utility can resolve package dependencies.

# wget <http://mirror.opencsw.org/opencsw/pkgutil.pkg>

# pkgadd -d pkgutil.pkg

* Add /opt/csw/bin to PATH variable (/etc/profile)
* Install puppet

# pkgutil --install puppet

This will install puppet with all the dependencies ruby, rubygems etc…

## Puppet Client configuration

* Make sure the hosts file contains a reference to puppet server

140.90.121.178 puppet dora01.nems.noaa.gov

* Make sure the /etc/puppet/puppet.conf file exists
* Add server line to the /etc/puppet/puppet.conf file
* Set report to true

[main]

# The Puppet log directory.

# The default value is '$vardir/log'.

logdir = /var/log/puppet

# Where Puppet PID files are kept.

# The default value is '$vardir/run'.

rundir = /var/run/puppet

# Where SSL certificates are kept.

# The default value is '$confdir/ssl'.

ssldir = $vardir/ssl

[agent]

# The file in which puppetd stores a list of the classes

# associated with the retrieved configuratiion. Can be loaded in

# the separate ``puppet`` executable using the ``--loadclasses``

# option.

# The default value is '$confdir/classes.txt'.

classfile = $vardir/classes.txt

# Where puppetd caches the local configuration. An

# extension indicating the cache format is added automatically.

# The default value is '$confdir/localconfig'.

localconfig = $vardir/localconfig

**server = dora01.nems.noaa.gov**

**report = true**

* One the client run puppet test

# puppet agent --test –verbose

* On the puppet server sign the certificate from the client

# puppet cert sign server-name

* On the client run test again

# puppet agent --test –verbose

## Adding a new Puppet Module.

There are 2 environments in the MOC puppet configuration, the DEV and the PROD environments.

* All development files are located in /etc/puppet/dev
* All production files are located in /etc/puppet/prod

The procedure below is applicable to both environments.

To create a new Module in the DEV environment,

* Change to /etc/puppet/dev/modules
* Create a directory to represent the module. This directory would contain all the module files. The directory name is the module name. eg LDAP, MTA
* Change into the module directory
* Create the following sub directories

## Module Layout

On disk, a module is simply **a directory tree with a specific, predictable structure:**

* <MODULE NAME>
  + manifests
  + files
  + templates
  + lib
  + facts.d
  + tests
  + spec

### Example

This example module, named “ldap,” shows the standard module layout in more detail:

* ldap — This outermost directory’s name matches the name of the module.
  + manifests/ — Contains all of the manifests in the module.
    - init.pp — Contains a class definition. **This class’s name must match the module’s name.**
    - other\_class.pp — Contains a class named **ldap::other\_class.**
    - my\_defined\_type.pp — Contains a defined type named **ldap::my\_defined\_type.**
    - implementation/ — This directory’s name affects the class names beneath it.
      * foo.pp — Contains a class named **ldap::implementation::foo.**
      * bar.pp — Contains a class named **ldap::implementation::bar.**
    - nodes – contains nodes.pp. All the clients in the environment
  + files/ — Contains static files, which managed nodes can download.
    - service.conf — This file’s URL would be **puppet:///modules/ldap/service.conf.**
  + lib/ — Contains plugins, like [custom facts](http://docs.puppetlabs.com/facter/latest/custom_facts.html) and custom resource types. These will be used by both the puppet master server and the puppet agent service, and they’ll be synced to all agent nodes whenever they request their configurations. See [“Using Plugins”](http://docs.puppetlabs.com/guides/plugins_in_modules.html) for more details.
  + facts.d/ — Contains [external facts](http://docs.puppetlabs.com/facter/latest/custom_facts.html#external-facts), which are an alternative to Ruby-based [custom facts](http://docs.puppetlabs.com/facter/latest/custom_facts.html). These will be synced to all agent nodes, so they can submit values for those facts to the puppet master. (Requires Facter 2.0.1 or later.)
  + templates/ — Contains templates, which the module’s manifests can use. See [“Templates”](http://docs.puppetlabs.com/guides/templating.html) for more details.
    - component.erb — A manifest can render this template with template('ldap/component.erb').
  + tests/ — Contains examples showing how to declare the module’s classes and defined types.
    - init.pp
    - other\_class.pp — Each class or type should have an example in the tests directory.
  + spec/ — Contains spec tests for any plugins in the lib directory.
  + rcs/ - contains file revisions.

Each of the module’s subdirectories has a specific function, as follows.

### Manifests

**Each manifest in a module’s manifests folder should contain one class or defined type.** The file names of manifests **map predictably** to the names of the classes and defined types they contain.

init.pp is special and **always contains a class with the same name as the module.**

Every other manifest contains a class or defined type named as follows:

| **Name of module** | **::** | **Other directories:: (if any)** | **Name of file (no extension)** |
| --- | --- | --- | --- |
| ldap | :: |  | other\_class |
| ldap | :: | implementation:: | foo |

Thus:

* ldap::other\_class would be in the file ldap/manifests/other\_class.pp
* ldap::implementation::foo would be in the file ldap/manifests/implementation/foo.pp

The double colon that divides the sections of a class’s name is called the **namespace separator.**

### Allowed Module Names

Module names should only contain **lowercase letters, numbers, and underscores,** and should **begin with a lowercase letter;** that is, they should match the expression [a-z][a-z0-9\_]\*. Note that these are the same restrictions that apply to class names, but with the added restriction that module names cannot contain the namespace separator (::) as modules cannot be nested.

Although some names that violate these restrictions currently work, using them is not recommended.

Certain module names are disallowed:

* main
* settings

### Files

Files in a module’s files directory are automatically served to agent nodes. They can be downloaded by using **puppet:/// URLs** in the source attribute of a [file](http://docs.puppetlabs.com/references/stable/type.html#file) resource.

Puppet URLs work transparently in both agent/master mode and standalone mode; in either case, they will retrieve the correct file from a module.

Puppet URLs are formatted as follows:

| **Protocol** | **3 slashes** | **“Modules”/** | **Name of module/** | **Name of file** |
| --- | --- | --- | --- | --- |
| puppet: | /// | modules/ | ldap/ | service.conf |

So puppet:///modules/ldap/service.conf would map to ldap/files/service.conf.

### Templates

Any ERB template (see [“Templates”](http://docs.puppetlabs.com/guides/templating.html) for more details) can be rendered in a manifest with the template function. The output of the template is a simple string, which can be used as the content attribute of a [file](http://docs.puppetlabs.com/references/stable/type.html#file) resource or as the value of a variable.

**The template function can look up templates identified by shorthand:**

| **Template function** | **(‘** | **Name of module/** | **Name of template** | **’)** |
| --- | --- | --- | --- | --- |
| template | (' | ldap/ | component.erb | ') |

So template('ldap/component.erb') would render the template ldap/templates/component.erb.

* Copy the file you want to centrally manage to the files directory
* Change to the manifests directory and create the init.pp file. This file contains the class whose name should match the module name.
* See content of init.pp file below

class ldap {

file { '/etc/hosts.allow':

owner => root,

group => root,

mode => 0644,

source=> "puppet:///modules/ldap/hosts.allow",

}

file { '/etc/hosts.deny':

owner => root,

group => root,

mode => 0644,

source=> "puppet:///modules/ldap/hosts.deny",

}

}

* Add the module to the site manifest in /etc/puppet/dev/manifest/site.pp

The site manifest is a primary data source. It can:

* Assign classes
* Configure classes with parameters
* Configure classes with top-scope (and node-scope) variables
* Convert arbitrary data from other sources into class parameters or variables
* Declare lone resources, outside any class

//site.pp

#import "templates"

#import "modules"

import "nodes"

import "pupp"

import "fixes"

import "ldap"

* This module can now be assigned to ldap nodes.
* In the manifest directory edit the nodes file and add the new module to the desired servers.

//node.pp

node 'ldap0.nems.noaa.gov' {

include pupp

include fixes

include ldap

}

node 'ldap4.nems.noaa.gov' {

include pupp

include fixes

include ldap

}

node 'ldap5.nems.noaa.gov' {

include fixes

include ldap

}

node 'ldap6.nems.noaa.gov' {

include fixes

include ldap

}

* Run puppet test on client
* puppetd --test --noop –evaltrace : Test Production env
* puppet agent --test --noop --environment=development : Test Development env
* Apply new config: puppet agent –test --environment=production

How to update a puppet managed file

Puppet dashboard installation and configuration be