Puppet Configuration Management System
Evaluation Report

4 February 2014

Version 1.0

Table of Contents

A. Document History
B. Document Scope
C. Executive Summary and Recommendation
D. Introduction
E. Prerequisites
F. Evaluation Details
    a. Test run 1
    b. Test run 2
    c. Test run 3
    d. Results
G. Installation Information
H. Usage Information
I. Evaluation Result

Appendix: Suggestions for Evaluation Procedure Revisions

A. Document History

Entire Document	0.1	1/12/2015	Created Document	Peter Enstrom
Usage howto	0.2	1/21/2015	Updated this section	Vana V.
Installation howto	0.2	1/21/2015	Updated this section	Vana V.
Introduction	0.2	1/22/2015	Updated this section	Susan L.
Sections E & I	0.5	1/27/2015	Updated	Peter Enstrom
Installation howto	0.6	1/28/2015	Updated	Vana V.
Entire Document	0.7	2/3/2014	Suggested changes	Peter Enstrom
Entire Document	1.0	2/4/2015	Team Review	Entire team

B. Document Scope

This document is the evaluation report for the functional evaluation of the Puppet Configuration Management System. The results of the evaluation and the collected information here are intended for the benefit of the XSEDE project and the general eScience community.

C. Executive Summary and Recommendation

An evaluation of the Puppet Open Configuration Management System version 3.7.3 was performed by the Technology Investigation Service (TIS). Puppet is a flexible, customizable framework designed to help system administrators automate the many repetitive tasks they regularly perform. The software functioned well and was able to meet all of the tested requirements.

Puppet was selected for evaluation for its ability to provide configuration management and provisioning on multiple distributions of Linux and other OS's. It has a large, flexible configuration language that allows it to perform many administration tasks on a large, diverse set of machines.

The main drawback to the software is that users will need to monitor the puppet log files to detect if any errors are occurring that prevent the configured commands (manifests) from being applied. Some non-fatal errors, such as where one file in a manifest is missing a file it is dependent upon, causes the entire manifest not to be applied even though the rest of the manifest contained valid operations. The enterprise version of the software, which was not tested, does have a reporting feature which eliminates the need to monitor the log files.

We recommend that puppet be used by the XSEDE community to manage their systems. We intend to use it for management tasks on the testing machines used by TIS.

D. Introduction

The Evaluation team was led by Peter Enstrom. The other members of the evaluation team were Susan Litzinger, Vanamala Venkataswamy and Dan Lapine. The evaluation was started on August 30, 2014 and finished on December 12, 2014. There are both commercial and open source versions of the software. This evaluation was performed on the open source version of the Puppet Open Configuration Management System version 3.7.3.

Puppet is a flexible, customizable framework designed to help system administrators automate the many repetitive tasks they regularly perform. It allows the desired state to be defined then automatically enforces that state. Whether a few servers or thousands of physical and virtual machines are being managed, Puppet automates tasks that system administrators often do manually, and ensures consistency, reliability and stability. Puppet's language for specifying how the machines that it manages will be configured is declarative rather than procedural. The configuration is specified but the steps necessary to achieve that configuration are taken care of by Puppet. This makes it possible for people without system administration expertise to set up the configuration without needing the specific knowledge of what is going on in the infrastructure.

Puppet uses a client/server architecture. Every node (physical server, device or virtual machine) being controlled by Puppet has an agent installed on it, while one machine may be used as a server designated to run the Puppet master. Enforcement takes place during regular Puppet runs which follow these steps:

- **Fact collection.** The Puppet agent on each node sends facts about the node's configuration — detailing the hardware, operating system, package versions and other information — to the Puppet master.
- **Catalog compilation.** The Puppet master uses facts provided by the agents to compile detailed data about how each node should be configured — called the catalog — and sends it back to the Puppet agent.
- **Enforcement.** The agent makes any needed changes to enforce the node's desired state. Note: If you choose to run in no-op (simulation) mode, the agent will simply simulate the changes.
- **Report.** Each Puppet agent sends a report back to the Puppet master, indicating any changes that have been made to its node's configuration. (In no-op mode, reports indicate which nodes are out of compliance.)

Puppet runs on many distributions of Linux, as well as Windows, Mac OS X, BSD, AIX, Solaris and HP-UX. There is no GUI in the open source version, but one is available in the commercial version. It has an API that allows its backend database to be queried. The API was not evaluated.

E. Prerequisites

The puppet agent service has no particular hardware requirements and can run on nearly anything. However, the puppet master service is fairly resource intensive, and should be installed on a robust dedicated server.

Software requirements for Puppet are detailed in their documentation here. Ruby is required. If using Puppet master/ agent setup, you may need to install Apache. We aren't aware of any OS or service requirements.

F. Evaluation Details

Puppet was evaluated to verify that it meets the following requirements:

- The ability to specify a server's configuration - It should allow packages, user accounts, groups, etc. to be specified and stored as a specification that can later be applied to a server.
- The ability to apply the same configuration to multiple servers - It should allow a previously defined specification to be used to configure multiple servers.
- The ability to configure servers with different OS types/versions - It should work with most of the recent, common versions of Linux.
- Adequate notification of failures - Users need 100% notification for any failure when configuring a server.
- Understandable messages - The software should provide user error messages that are appropriate, useful and non-cryptic.

Puppet's configuration management functions were tested in 3 separate test runs. In the first one Puppet was installed and run locally on physical test nodes. In the second, Puppet was installed and run in client/server mode on virtual machines using VirtualBox. In the last one, Puppet was installed and run locally on machines in a cloud using OpenStack. During the tests, Puppet's behavior was configured by changing the contents of its manifest files. The manifests dictate what state Puppet enforces on the machines that it manages. After the manifests are applied, the person conducting the test verifies that the configuration specified by the manifest has been applied to the appropriate machines.

Test run 1

In the first test run, Puppet was installed, configured and run on a single server. The servers used for this test were pancake1.psc.edu and pancake3.psc.edu which ran CentOS. Puppet was then invoked from the command line which applies its manifest once on the local machine. This is not how Puppet would normally be run but it does allow a manifest to be applied immediately instead of waiting for the next cycle of the Puppet master daemon. This shows that Puppet is installed correctly and can manage the machine. Next, some errors were introduced to determine how Puppet handled and reported them. For account creation we tested how Puppet reacts when an account already exists, when the parent directory of the home directory we specified doesn't exist, when the UID is already in use by another user, and when the Puppet process didn't have the access needed to create the account.

No issues were encountered during the first test run. Messages were displayed in the terminal where Puppet was invoked. The same messages were logged in /var/lib/puppet/state/last_run_report.yaml as well as being written to the system log. The messages were informative. Here is an example from the test attempting to add a user with a UID that is already in use:

Error: Could not create user t2susan: Execution of '/usr/sbin/useradd -d /home/t2susan -s /bin/bash -u 20034 -m t2susan'

returned 4: useradd: UID 20034 is not unique

Error: /Stage[main]/Localusers/User[t2susan]/ensure: change from absent to present failed:

Could not create user t2susan: Execution of '/usr/sbin/useradd -d /home/t2susan -s /bin/bash -u 20034 -m t2susan' returned 4: useradd: UID 20034 is not unique

Test run 2

In the second test run, Puppet was installed, configured and run on a set of virtual servers running Ubuntu. All of the virtual servers were set up on a single test machine using VirtualBox. The test machine used was hexapuma.ncsa.illinois.edu which runs OpenSuSE. Puppet was installed on one server as the puppet master. Puppet was configured and started up such that both machines had agents that were in communication with the master to control their virtual machines. Manifests were built with configurations specific to each of the server and the client machines. The manifests for this test are in a tarball on the wiki here. The Puppet processes were run as daemons in this test. For each section of this test, a version of the manifest was copied over the /etc/puppet/manifests/site.pp file. After the next cycle of the puppet master daemon executed, the /var/lib/puppet/state/last_run_report.yaml file was checked on each machine for messages from Puppet to verify the changes took effect and to look for any error messages. Since it was running continuously as client/server daemons rather than from the command line, status messages were not sent to the terminal. During the test, manifests were put into place so that Puppet managed both the master and agent virtual machines. During the test, Puppet functions were exercised to create user accounts and groups, to create a multilevel directory and to later update it. A file was copied from a source file. The Puppet function to create a file only if a required file exists was tested. Puppet was used to ensure that a service was running. In order to view an error message in the master/agent architecture an attempt was made to create files with a cyclical dependency which failed as expected.

Puppet performed the actions that were possible and gave informative error messages for actions that weren't. Messages were logged in /var/lib/puppet/state/last_run_report.yaml as well as being written to the system log. The one issue that was encountered may be a design decision rather than a bug but it isn't clear. In that portion of the test the manifest specified three independent files that should exist on the system. Two of them were specified correctly but the third was dependent upon a non-existent file. None of the actions specified in the manifest file were performed and the log contained this message:

Failed to apply catalog: Could not find dependency File[/tmp/TIS_S_bogus] for File[/tmp/TIS_S_fileC] at /etc/puppet/manifests/site.pp:29

Test run 3

The third test run used OpenStack to test Puppet in a cloud environment. It was executed on NCSA's OpenStack test machine. A virtual server was created using CentOS and Puppet was installed on it. As with the first test run, Puppet was invoked from the command line for each part of this test. The manifests used in this test are in a tarball on the wiki here. A manifest that creates users, groups,

installs a package and executes OS specific code was applied to the machine. The same manifest was reapplied to verify that Puppet deals with already existing elements correctly. The manifest also contained a call to a script that appended a line to a test file. It was executed by Puppet every time it ran. The test then introduced error conditions to see how Puppet would deal with them. Manifests were applied with unrecognizable syntax and correct syntax but unrecognized keywords. Puppet was told to look for a non-existent manifest. A manifest tried to have Puppet create a file from a non-existent source file, to create a link in a non-existent parent directory, and to execute a bad command to verify a file. A manifest was applied to make Puppet try to create a user with a UID that was already in use by another user, to create a user with a home directory under a non-existent parent directory, and then to create a group duplicating a GID already in use by another group on the system. The test then attempted to load a non-existent package. As a non-privileged user, Puppet was run with a manifest to create user accounts, groups, and install a package. Three separate virtual servers were then created with different Linux OS types (CentOS, SUSE and UBUNTU) and Puppet was installed on all of them. The same manifest was executed on all of them to perform the same actions: create user accounts and groups, install a package and execute OS specific code.

Puppet performed the actions that were possible and gave informative error messages for actions that weren't. Messages were logged in /var/lib/puppet/state/last_run_report.yaml as well as being written to the system log.

In this test, Puppet was able to remove an installed software package when running the manifest on an Ubuntu and a CentOS Linux system. However, on OpenSuSE some additional configuration information is needed in order to remove a package. The Puppet documentation does contain "uninstall_options" for package management that is probably necessary when using OpenSuSE. We don't think this is a problem with Puppet.

Results

Overall the software worked very well. No serious errors were encountered and the only issue found is arguably not a bug but a design decision. The software handled errors well and logged informative messages. There is a learning curve for new users creating the manifest files but the logged information helps get past this. Once everything is configured correctly with correct manifest files Puppet runs with no problems.

If Puppet is to be used on a production system we recommend that its logs be monitored to ensure that the manifests are being fully applied. Even if everything is working properly failures can occur in the future since some changes on the system can result in problems. For example, deleting a required file results in the entire manifest file no longer being applied. There is no indication of this outside of the log files if running in master/agent mode.

Puppet messages are logged in the syslog and to the /var/lib/puppet/state/last_run_report.yaml file.

G. Installation Information

Standard open source Puppet installation includes puppet master and puppet agent software as a base. That said, Puppet can be configured to be updated on the client nodes in different ways. We tested both a puppet master/agent setup for automated configuration changes and multiple independent puppet agents on machines and with GIT used to synchronize manifests. For this evaluation, we tested 3 types of setup. 1. Master and Agent setup using VMs (using Virtualbox for VMs) 2. Puppet agents on cloud infrastructure and 3. Puppet agents using GIT for synchronization between different nodes.

How to decide what kind of configuration you will need:

If you are maintaining many VMs and need to automatically configure/maintain those VMs then setting up Puppet Master on one VM and puppet agents on the rest of the VMs is recommended. This is true for cloud infrastructure as well where resources come and go on the fly and whenever a new resource comes up just install agent on it and sync it with puppet master. Setting up Master/Agent configuration may be tedious and may take more effort than setting up multiple puppet agents and syncing them using GIT or manually copying manifests. It needs some knowledge of networking and requires that agents and master can talk to each other on the network.

On the other hand, if your configuration does not change very frequently and updates are propagated only as needed then setting up a central puppet node with GIT is suitable. The new (or existing) nodes can then sync with central puppet node as and when needed. One drawback is with file copy. If the central puppet node and GIT are on different machines, then file copy may be duplicated. This is especially inefficient if the file sizes are too large and files need to be copied from central puppet node to GIT machine to another puppet agent machine.

Puppet Master: Main server that describes the manifests, modules and hosts/systems information on which manifests and modules can to be applied. Sometimes puppet master can be referred as puppet server depending on what OS distro one is using but puppet master and puppet server are same thing and can be used interchangeably.

Puppet Agent: Other systems that uses system specific providers to enforce the resources specified in the manifests.

Modules: Code chunks concerning a service/package set to install.

Instructions for installing Puppet on different flavors of machines

RHEL/CentOS

Install the latest repo and then install puppet-server:

Download the latest repo from the puppetlab web site by selecting the correct OS version for your machine. For the RHEL/CentOS testing, we chose the i386 EL 6.5 version. This install includes the

software necessary for both the server and agent.

```
$ rpm -ivh http://yum.puppetlabs.com/el/6.5/products/i386/puppetlabs-release-6-10.noarch.rpm

To install a Puppet master:

$ yum install puppet-server

To only install a Puppet agent:

$ yum install puppet
```

SLES/OpenSuse

Download the zypp's .repo file from http://download.opensuse.org/repositories/systemsmanagement:/puppet/

Select the correct OS version and copy the systemsmanagement:puppet.repo file to /etc/zypp/repos.d. You can use wget or cut/paste. Update the repo database and accept the new gpg key for the new repository

```
$ zypper refresh
```

To install a Puppet master:

```
$ zypper install puppet-server

To install a Puppet agent:

zypper install puppet
```

Ubuntu/Debian

```
$ wget https://apt.puppetlabs.com/puppetlabs-<ubuntu-release-name>.deb

$ sudo dpkg -i puppetlabs-<ubuntu-release-name>.deb

$ sudo apt-get update

To install a Puppet master:
```

`$ sudo apt-get install puppetmaster-passenger` # We recommend this one, as it will save you a step in the post-install tasks. It will install Puppet and its prerequisites, and automatically configure a production-capacity web server.

`$ sudo apt-get install puppetmaster` # This will install Puppet, its prerequisites, and an init script (`/etc/init.d/puppetmaster`) for running a test-quality puppet master server.

To install a Puppet agent:

```
$ sudo apt-get install puppet
```

To upgrade to the latest version of Puppet, you can run:

```
$ sudo apt-get update

$ sudo puppet resource package puppetmaster ensure=latest # On Master

$ sudo puppet resource package puppet ensure=latest # On Agents
```

Starting Puppet

Start the puppet server on the master server.

```
master$ sudo puppet master
```

Connect from the agent machine.

```
agent$ puppet agent –t
```

Sign the certificate for the agent on the master server.

```
master$ puppet cert sign –all
```

If you are running an agent on the master server, puppet can manage it as well.

```
master$ puppet agent
```

Start the puppet agent on the agent machine.

```
agent$ puppet agent
```

At this point puppet is up and running and will apply the contents of the /etc/puppet/manifests/site.pp file

Usage Information

In standard installation, Puppet manifests and modules live in /etc/puppet and in our testing we used this as the default for all node configuration. /etc/puppet/puppet.conf holds the config information for the puppet services. In our testing environment, we built a directory tree under /etc/puppet for the node/site configs.

Sample Site information file: /etc/puppet/manifests/site.pp

```
node 'copacetic' {
      include mc
      include localusers
      file { '/tmp/hello':content => "Hello dan on copacetic\n",
      }
}
```

```
node 'stroit' {
      include mc
      file { '/tmp/hello':content => "Hello dan on stroit\n",
      }
}
```

Sample manifest for recursive directory creation. /etc/puppet/manifests/directory-recursive.pp

```
# create a directory tree, list the directories in order, using variable
whisper_dirs for list of directories.
$whisper_dirs = [ "/tmp/whisper/", "/tmp/whisper/2.0",
                  "/tmp/whisper/2.0/bin", "/tmp/whisper/2.0/log",
                  ]
file { $whisper_dirs:
      ensure => "directory",
      owner  => "vv3xu",
      group  => "staff",
      mode   => 750,
}
```

Sample manifest to copy a file from source (/etc/puppet/files/cat-pictures.txt) to destination (/tmp/cat-pictures.txt). /etc/puppet/manifests/file-source.pp

```
file { "/tmp/cat-pictures.txt":
      source => "puppet:///files/cat-pictures.txt",
}
```

Sample manifest for installing “screen” software. /etc/puppet/manifests/screen-install.pp

```
package { "screen":
      ensure => "installed"
}
```

Sample modules for creating users, user home etc.
/etc/puppet/modules/localusers/manifests/init.pp

```
# Manage local users
# this assumes no local passwds
# i.e kerberos or other
class localusers {
      user { "test_dan":
            ensure => present,
            uid => '804',
            gid => 'users',
            shell => '/bin/bash',
            home => '/home/test_dan',
```

```
            managehome => true,
        }
        user { "test_peter":
            ensure => present,
            uid => '805',
            gid => 'users',
            shell => '/bin/bash',
            home => '/home/test_peter',
            managehome => true,
      }
}
```

Once the manifests and modules are created on master node, use git (or other rcs) to maintain control of changes. Use rsync to keep the master tree synced to the various nodes. Use puppet apply to run the deployed tree

```
$ puppet apply /etc/puppet/manifests/site.pp --modulepath=/etc/puppet/modules
```

By running this command, all the modules in /etc/puppet/modules (in this case creating users) will be executed based on the site information in site.pp. Change the node names to match the FQDN of your nodes. You can also create other user types with other modules per function (admin, remote, etc), install packages, create files or define other tasks that needs to be accomplished on a given site/node.

H. Evaluation Result

We recommend Puppet for use in XSEDE. It ensures that systems are configured correctly and that they stay that way. It runs on most modern operating systems. Its configuration language is flexible and powerful yet it is simple to use for most tasks. The software is well documented. It handles errors well and gives informative messages.

We tested on a small number of machines, not at scale. The enterprise version specifically addresses issues of running at a large scale.

The commercial version of Puppet provides reporting which is lacking in the open version. Because of this it will be necessary to periodically monitor Puppet's log files to ensure that it is still managing the system as desired. For example, if Puppet is managing a file that depends upon another file, it will stop applying all of the commands in its manifest if the required file is deleted. You have to monitor to catch this since it could work well for a long while but if something deletes the required file, the entire manifest would no longer be updated.

I. Appendix: Suggestions for Evaluation Procedure Revisions

Using google documents for collaborative editing sped up the report writing process.