Testbed Configuration and Management

Georgi Zlatkov

CERN openlab
5th August 2011

Testbed configuration and management

Student: Georgi Zlatkov
Supervisors: Andrew Elwell & Tomasz Wolak
5th August 2011
Version 1
Distribution: Public

Table of Contents

Abstract

The aim of this openlab summer student project was to evaluate and deploy the IT system management tool “Puppet” in a test environment, which could be used for large-scale system configuration and management at CERN and on the WLCG[1]. The use of this tool would allow centralized and automated configuration and management of production systems and testbeds alike, improved reliability, cross-platform support and better compliance with CERN's policies such as security.

Introduction

In this report an introduction to the IT systems management tool “Puppet” will be given, a brief explanation of its features and ideas on how could be used to manage testbeds at CERN and on the WLCG. The first chapter explains Puppet functionalities, how it works and how to deploy a basic Puppet's system. The second part discusses how and why Puppet would be useful to CERN, a sample iptables module I wrote for CERN and a list of module repositories.

[1] http://lcg.web.cern.ch/LCG/public/overview.htm

1 Puppet

Configuration and management of a large number of systems often comes with a lot of repetitive tasks such as installing the same packages, configuring services to etc. on every machine. Apart from that a system administrator is required regularly ensure that systems stick to the defined configurations and changes are applied in controlled manner. This is not only time consuming but usually hard to achieve. A tool to automate this process is needed.

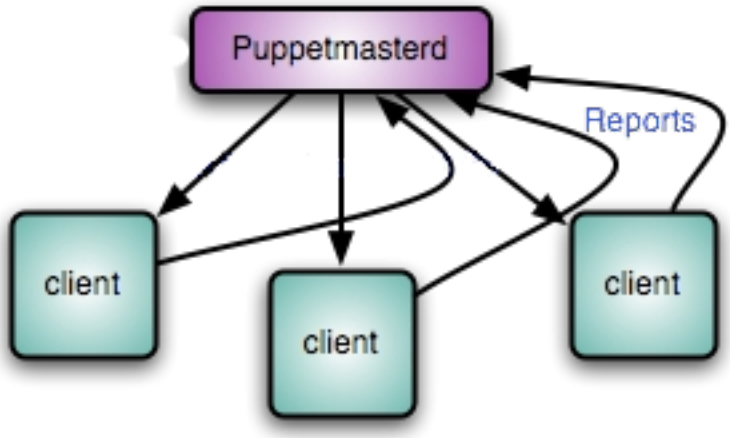
1.1 What is Puppet?

The official description of Puppet states that "Puppet is an open-source next-generation server automation tool. It is composed of a declarative language for expressing system configuration, a client and server for distributing it, and a library for realizing the configuration."[1] Puppet combines both the ability to apply a specific host configurations and to ensure that they will be persistent. It is written in Ruby and released under Apache 2.0 license after version 2.7.0. Released for a first time in 2005 it continues to evolve rapidly with last stable release from 22th July 2011 (2.7.1). Puppet is design to be a cross-platform product. Currently it supports a wide variety of operating systems including most Linux distributions, Unix, Unix-like systems, and after version 2.6 a basic support of Microsoft Windows with the idea to improve and extend it during 2011.[3]

1.2 How Puppet works?

What makes Puppet a powerful system automation tool is its simplicity of adding new configuration policies, applying client specific instructions and making sure that a system complies with the defined end state. A system managed with Puppet is divided into two parts: central server called puppetmaster and the clients called

puppets or nodes. A puppet master can also be a node. The puppetmaster stores the configuration policies and is responsible for applying them and to ensure the end state of the nodes.[1]



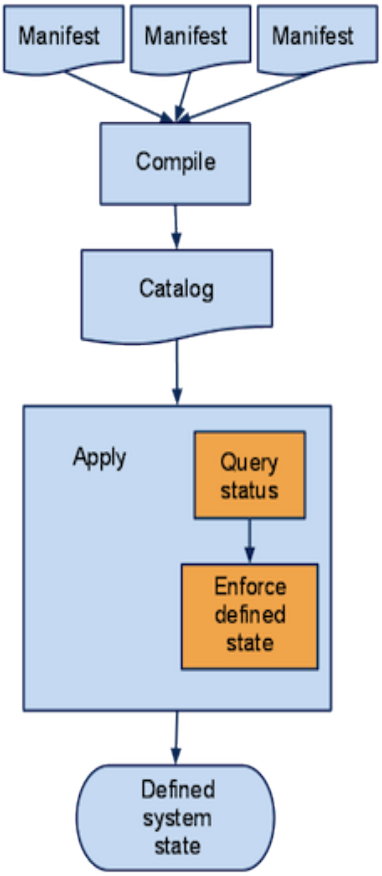
System managed with Puppet
Fig. 1

A system administrator describes the desired end state of a system into *manifests* which are written in Puppet's own declarative language or in Ruby DSL[1].[3] The idea behind the language used for manifests is to be as human readable as possible, without loss of functionality:

```
class sudo {
   file { "/etc/sudoers":
      ensure => present,
      owner => "root",
      group => "root",
      mode  => 440,
       }
}
```

[1] http://projects.puppetlabs.com/projects/1/wiki/Ruby_Dsl

This sample code when applied will ensure that the file is *present* on the client machine, its owner and group are *"root"* and its permissions are set to *"440"*.



Defined state of a client
Fig. 2

On a regular basis the Puppet daemon on the client system will try to connect to its puppetmaster and download a catalog of the current active manifests. The catalog is compared to the system's configuration and the system is reconfigured to match the desired end state.

1.3 Deploying Puppet

Deploying Puppet on a system is a fast and straightforward process. Most of the operating systems offer Puppet in their package repositories but installation from source is also possible[1].
Once installed, Puppet needs to be configured before manifests can be applied.

First identity verification between the master and node machine needs to be performed. An identity verification in Puppet is performed using SSL certificates. Thus each client needs to request a personal certificate issued by the puppetmaster:

puppet agent --test --server puppet.example.com --waitforcert 10

Back on the puppetmaster we check for certificate request using "puppetca", which is shipped with the Puppet package and is responsible for certificate management

puppetca –list

and accept the request.

puppetca --sign client.example.com

This gives us a server and a client which have mutually trusted connection based on Puppet's own certificate authority. The last step is to start Puppet's client daemon "puppetd" on the client

/usr/sbin/puppetd

and it will report to the master on a regular basis.[4]

1.4 Basics of Puppet's language

Puppet's system configuration includes a collection of manifests written in Puppet's own declarative language. This language is based on a resource abstraction layer (RAL) which is used by Puppet to read and modify the state of resources on a client system in the following order: (1) Puppet checks what state a resource should be in, (2) compares it to the current state and (3) makes any necessary changes to reach the described state.[5]

Each resource has a name and a list of attributes which hold information about its state. In the example from the previous chapter "*/etc/sudoers*" was the resource's name and from its attributes we used "*ensure*", "*owner*", "*group*" and "*mode*". Most of the resources have one attribute whose value defaults to its name, in our case that is the "*path*" attribute. More information about Puppet's built-on resources and the available attributes can be found in the reference section of the PuppetLab's webite[1] or on the Puppet core types cheat sheet[2].

[1] http://docs.puppetlabs.com/references/2.7.0/type.html

[2] http://projects.puppetlabs.com/projects/puppet/wiki/Core_Types_Cheat_Sheet/

Applying a configuration on a system must be done in a specific order. For example before starting a service it first needs to be installed and configured. In Puppet's manifests we need to specify a relation between two resources so they can be executed in a logical order. To do this we use the "*before*" and "*require*" attributes for ordering and "*notify*" and "*subscribe*" when we need a resource to react to a change in another resource. In such a case "*before*" is used in the earlier resource and "*require*" in the later. The same dependency applies for "*notify*" and "*subscribe*". An example is a simple manifest to configure SSH on a client system[4]:

```
package { 'openssh':
        ensure => present,
}
file { '/etc/ssh/sshd_config':
        ensure => file,
        source  => '/etc/puppet/files/sshd_config',
        require  => Package['openssh'],
}
service { 'sshd':
        ensure     => running,
        subscribe  => File['/etc/ssh/sshd_config'],
}
```

Puppet's language also consists of variable declaration and *if-else* and *case* statements which are similar to the one used in any other programming languages. More information about them could be found on PuppetLab's website[1].

Another important feature of Puppet is the stash of pre-assigned variables with host specific information by Facter which could be extended with custom facts[1] if needed. The custom facts are written in Ruby and added as plugins to Puppet.

[1] http://docs.puppetlabs.com/guides/language_guide.html

1.5 Advanced configuration with Puppet

Classes[1]

A class is a collection of resources which are applied together and could be specific to a host or operating system.[1]

Modules[1]

A module is a directory which combines classes and any additional files that may be needed. Modules help us to collect configurations for a single application in a tree structure and later implement them as one.[1]

Parameterized classes[2]

Parameterized classes are declared the same way as the classical classes but a list of parameters is pasted to them when called. This allows us to change the behavior of a class depending on its use.[1]

File serving capability[3]

Quite frequently we will need to transfer static files to the client system. For that purpose Puppet has a file serving capability which allows us to use "*puppet:///*" as a prefix in the source attribute of a resource to tell the client to retrieve a file from its master's file server. By default the file server path is set to "/etc/puppet/module/" but could be changed if needed.[1]

[1] http://docs.puppetlabs.com/learning/modules1.html

[2] http://docs.puppetlabs.com/guides/parameterized_classes.html

[3] http://docs.puppetlabs.com/guides/file_serving.html

Templates[1]

Templates allow us to manage files when their content depend on an external factor like the host or operating system. They offer a convenient way to create files with host specific information.[1]

Virtual Resources[2]

A Virtual Resource is not by default sent and executed on the client but it needs to be called instead. It is marked as virtual by adding “@” infront of its specification. This is useful when we need to realize a resource more than one time which is not allowed when using classical resources.[1]

Puppet Version Control[3]

A good practice is to use a version control system to store manifests and other configuration files.[1]

[1] http://docs.puppetlabs.com/guides/templating.html

[2] http://docs.puppetlabs.com/guides/virtual_resources.html

[3] http://projects.puppetlabs.com/projects/1/wiki/Puppet_Version_Control

2 Using Puppet at CERN

2.1 Puppet for configuration and management of testbeds

The standard CERN tool (Quattor) for hosting and managing testbeds lacks the ability to manage different operating systems than SLC. On the other hand Puppet is build around the idea of cross-platform compatability. Equipped with its own declarative programming language and the ability to configure and manage a large number of hosts it corresponds to the needs of CERN.

2.2 Sample module

This is a sample module I wrote as part of my project on request from my supervisor. Its purpose is to configure the “iptables” service on a client hosts by combining chunks of iptables rules and applying the configuration on the host.

The first part of the manifest is

```
class iptables($iptables_rules) {
    case $operatingsystem {
        "Scientific","Fedora": { class { "iptables::redhat": iptables_rules =>
$iptables_rules, } }
        "Debian","Ubuntu": { class { "iptables::debian": iptables_rules =>
$iptables_rules, } }
        default: { notify { "Unrecognised distribution": } }
    }
}
```

I define a parameterized class which takes as a parameter list of files containing the iptables rules. Then depending on the host's operating system another class is called and the parameters are transferred to it.

In case the pre-assigned variable "*$operatingsystem*" is equivalent to Scientific or Fedora the *"iptables::redhat"* class is called. After ensuring that the "*iptables*" package is installed it define a service resource. Because "*iptables*" is a "no running" process we need to trick Puppet that the service is always running so when a change on the configuration file occurs it will apply it right away.

```
service { "iptables":
            require => Package["iptables"],
            hasstatus => true,
            status => "true",
            hasrestart => false,
      }
```

Last we need to write the configuration file. The chucks of iptables rules are transmitted to the template which write them on the file "*/etc/sysconfig/iptables*" and notifies the service.

```
file { "/etc/sysconfig/iptables":
            ensure  => present,
            owner   => "root",
            group   => "root",
            mode    => "0640",
           content =>
template("/etc/puppet/modules/iptables/templates/iptables.erb"),
            require => Package["iptables"],
           notify  => Service["iptables"],
      }
```

In case the pre-assigned variable "*$operatingsystem*" is equivalent to Debian or Ubuntu we again ensure that the "iptables" package is installed and write the "*iptables*" instructions to "*/etc/default/iptables*". Debian based systems apply the "*iptables*" rules using the command "*/sbin/iptables-restore"* so we define an "*exec" resource which execute the command*

```
exec { "iptables_apply":
        command => "/sbin/iptables-restore < /etc/default/iptables",
    }
```

2.3 Resources

Combining multiple files which are related to a single application in modules is a good practice. There are many repositories where one find modules written by skillful Puppet users. These are some of them.

Puppet Forge: http://forge.puppetlabs.com/

This is the official repository of PuppetLabs called PuppetForge. It consists of all types of modules which are separated into categories for easy searching.

Ricardo Brito Da Rocha's github repository: https://github.com/rochaporto/repositories

A lot of useful CERN specific modules written by a member of the IT-GT group at CERN.

David Schmitt's git repository: http://git.black.co.at/

David Schmitt's repository offers webhosting automation modules.

Eshao: https://github.com/eshao/puppet

Eshao is a github repository for modules for FreeBSD. Its main idea is for writing manifests in a simple and clean way.

Example42: http://www.example42.com/

Dedicated to Puppet documentation, tools, tutorials and sample instructions, it also consists of a large number of modules including the “puppi” tool for deployment automation.

Summary

The IT-GT group is responsible for hosting and managing testbeds for internal users and EMI. The need of a cross-platform automating system administration tasks system suggests the use of tool like Puppet. It has a large developer base, a language specifically designed for configuration of servers, good documentation, multiple platform support and a fast evolving product. For the needs of the IT-GT group Puppet could be extended with CERN specific modules which to replace the current system.

Bibliography

[1] “Puppet Documentation”, *Puppet Labs,* 13 May 2011,
<http://downloads.puppetlabs.com/puppet/puppet.pdf> [accessed 4 August 2011]

[2] “Puppet versus Chef: 10 reasons why Puppet wins”, John Arundel, 1 December 2010,
<http://bitfieldconsulting.com/puppet-vs-chef> [accessed 4 August 2011]

[3] “Puppet (software)”, Wikipedia, The Free Encyclopedia, 8 July 2011,
<http://en.wikipedia.org/wiki/Puppet_%28software%29>
[accessed 4 August 2011]

[4] “Automate System Administration Tasks with Puppet”, Sean Walberg, 1 December 2008,
<http://www.linuxjournal.com/magazine/automate-system-administration-tasks-puppet?page=0,0> [accessed 4 August 2011]

[5] “Zero to puppet in one day”,
<http://finninday.net/wiki/index.php/Zero_to_puppet_in_one_day>
[accessed 4 August 2011]