SE464 Lab 3: Architectural Analysis of Puppet

Group 20

Shi Shi, Hong Wen Zhu, Dane Carr, Jiaer Wang, Lucas Wojciechowski

July 20, 2014

1 System Functionality

Puppet is a system which allows a software engineer to describe the desired state of a set of server nodes, automatically configure those nodes to that state, and continuously monitor and correct the nodes' state.

A Ruby-based configuration language specifies the desired state for the nodes as a "manifest". This manifest is uploaded to a "Puppet Master" server which monitors the state of all nodes and pushes any necessary configuration changes.

This system is used in accordance with the following user stories

- As a software engineer, I want a powerful and consistent way to describe my infrastructure deployment plan so that it is thorough and easy to maintain.
- As a software engineer, I want a way to ensure my infrastructure is always in the correct state so that it is robust and reliable.
- As a software engineer, I want an automated way to deploy software to my infrastructure so that deployments are fast and consistent.

2 Quality Attributes and Scenarios

2.1 Reliability

Reliability is a measure of how often a system is available when it is needed.

Puppet improves the reliability of your infrastructure by providing repeatable automatic deployments and continuous monitoring.

As a"metasystem" designed to make other systems reliable, Puppet must itself be reliable. It is important that the Puppet Master be available, both to monitor the state of the nodes in the system and to accept new configuration from system administrators.

2.1.1 Quality Attribute Analysis Scenarios

An Apache webserver instance running on a server crashes. Puppet restarts Apache within a minute.

A software engineer wants to run Sendmail on a particular server node. She modifies the Manifest and pushes it to the Puppet Master. The Puppet Master is available and pushes these changes to the appropriate node within 30 minutes.

2.2 Modifiability / Maintainability

Maintainability is a measure of "fitness-for-future", the ability of a system to be modified to meet future requirements. We can consider Puppet's maintainability from two different perspectives: the effect of using Puppet on the maintainability of server infrastructure and the extendability of the Puppet configuration language to accommodate new types of systems.

Server infrastructure managed by Puppet is more maintainable. Corrective and augmentative maintenance are easy because understanding and adjusting the state of a server is as simple as reading and modifying a text file. The Puppet Master can perform basic preventative maintenance itself and notify system administrators of more serious problems.

The configuration language itself is open to modification to support new types of systems with complex domain-specific configuration requirements.

2.2.1 Quality Attribute Analysis Scenarios

A software engineer wants to use Puppet to deploy new custom genetic analysis software. Different data sets will be deployed to different server nodes. She can easily extend Puppet's configuration language to represent these new types of configuration options.

2.3 Usability

Usability is a measure of how efficiently and effectively a system's user interface allows a user to interact with it.

The Puppet configuration language and command line tools are user interfaces which afford software engineers greater efficiency and effectiveness in managing server infrastructure.

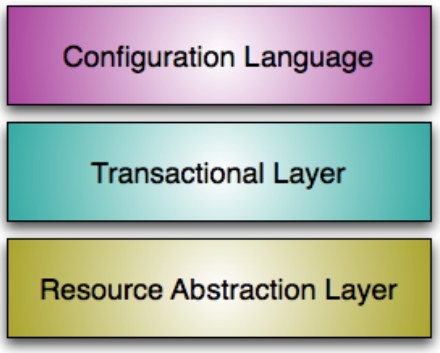
2.3.1 Quality Attribute Analysis Scenarios

A software engineer who just joined the team is tasked with setting up an FTP server on a node in the Company's infrastructure. She is quickly able to understand how Puppet works and perform the task.

3 Architectural Views

3.1 Layered view

Puppet's functionality is divided into three layers: configuration language, transaction layer and resource abstraction layer.



3.1.1 Configuration Language Layer

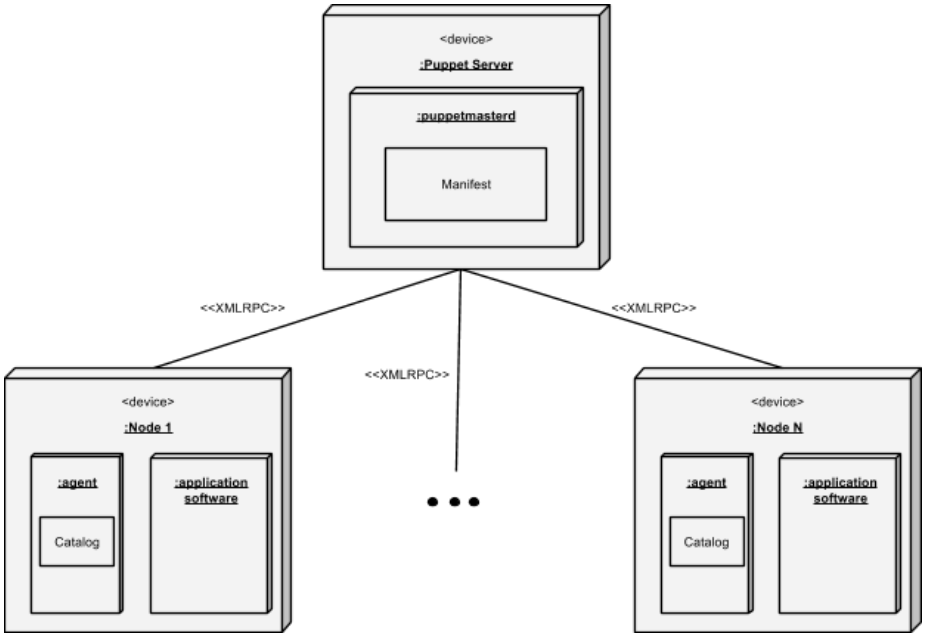
The Configuration Language Layer is the Ruby API used to describe the desired state of the nodes. This layer exposes resources as Ruby methods. Because configurations are described in Ruby, it is possible to use the full power of Ruby metaprogramming to eliminate redundant code and integrate with other systems.

```
user { 'dave':
  ensure     => present,
  uid        => '507',
  gid        => 'admin',
  shell      => '/bin/zsh'
}
```

The configuration language is platform independent, so that it can be used on any type of server that can be configured with puppet, and so that one configuration can be applied across heterogeneous systems.

This layer only concerns itself with the desired state of the nodes, it does not know how to communicate with other nodes in the system or realize the desired state.

3.1.2 Transactional Layer



The Transactional Layer handles the communication between nodes in Puppet. This is crucial because the Puppet infrastructure contains many different agents which need not be on the same physical machine.

Puppet Master The master coordinates the Puppet infrastructure. It has the canonical Manifests, queries facts from Puppet Nodes, processes these facts to build a catalog, sends that catalog to the nodes, and routes reports to the Report Collector. It can be thought of as an instance of the Mediator Pattern.

Puppet Node A node is a server who's software is managed by Puppet.

Puppet Report Collector A report collector receives information from one or more Puppet masters about the state of the system, which it makes available to software engineers.

Development Machine Software engineers may run the Puppet command line tool on their development machine to talk to the Puppet Master, among other tasks.

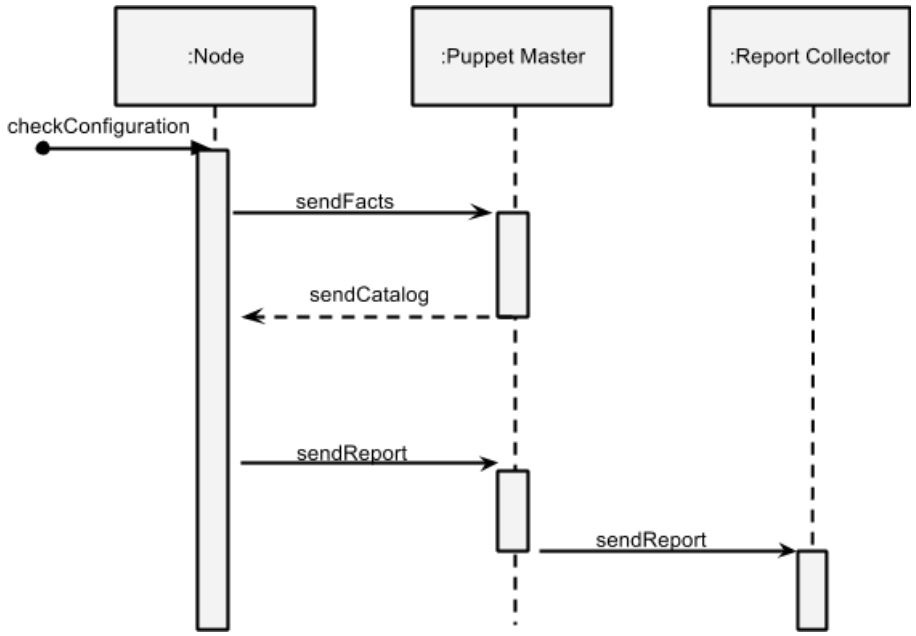
This layer may treat the configuration layer as a "black box", it doesn't need to know what a particular configuration source file means, it just needs to transfer the information between agents. This layer does not know how to realize the desired state specified in the configuration layer; that is left to the Resource Abstraction Layer.

3.1.3 Resource Abstraction Layer

The Resource Abstraction Layer knows how to translate the instructions from the Configuration Language Layer into architecture-specific commands for nodes, Catalogs.

It relies upon the transactional layer to receive information from the Configuration Language and transfer Catalogs to their correct nodes.

3.2 Logical View



The communication between the components is shown in the above sequence diagram. Each component is responsible for compiling the information required by the next step, so that clients are not accessing raw Puppet modules.

Periodically, (every 30 minutes by default) puppet nodes check their configuration with the puppet master. A module called "Facter" gathers up information about the node, such as the operating system, IP address, or other custom information gathered by various Facter plugins. The node then sends this information on to the puppet master.

The puppet master uses the facts about the node together with the manifest files to compile a configuration specific to that node. The compiled configuration is called a catalog. The puppet master sends the catalog back to the node, which then applies the configuration. After the configuration has been applied, it generates a report of the configuration and sends it back to the puppet master. The puppet master then sends the report on to the report collector, which can either be puppet itself or a third party implementation based on puppet's open API.

4 Architectural Analysis

4.1 Reliability

An Apache webserver instance running on a server crashes. Puppet restarts Apache within a minute.

Puppet Nodes periodically send "facts", a snapshot of their state, to their Puppet Master. The Puppet Master compares these facts to the desired system state, as defined in the manifest, and creates a "catalog" of changes to be made on the Puppet Node in order to bring it into the desired state.

In this case, the fact that Apache wasn't running would be reported to the Puppet Master. The Puppet Master would then send the node a catalog which would include commands to restart Apache.

The catalog can be thought of as a use of the command pattern.

A software engineer wants to run Sendmail on a particular server node. She modifies the Manifest and pushes it to the Puppet Master. The Puppet Master is available and pushes these changes to the appropriate node within 30 minutes.

The Puppet Master daemon is a single process that can be isolated from the rest of the infrastructure and carefully monitored. As long as the Puppet Master daemon is running and not reporting any errors, the rest of your infrastructure can be assumed to be working correctly.

In addition, it is possible to run multiple Puppet Masters, each collecting facts from and sending catalogs to a number of Puppet Nodes. This provides redundancy in case one Puppet Master fails.

4.2 Modifiability / Maintainability

A software engineer wants to use Puppet to deploy new custom genetic analysis software. Different data sets will be deployed to different server nodes. She can easily extend Puppet's configuration language to represent these new types of configuration options.

Puppet's Resource Abstraction Layer allows users to customize the configuration system used by Puppet to suit domain specific needs. This means that complex infrastructure deployed by Puppet will not be constrained by the expressiveness of Puppet's configuration language.

4.3 Usability

A software engineer who just joined the team is tasked with setting up an FTP server on a node in the Company's infrastructure. She is quickly able to understand how Puppet works and perform the task.

The Puppet Configuration language is based on Ruby, a well-known programming language widely considered to be easy to use and powerful. Using Ruby as a user interface makes Puppet easy to learn, especially by people already framiliar with Ruby.

5 Key Weaknesses

Puppet is a complex system, often eschewing ease of use for modularity. Although the configuration language itself is easy to use, there is a steep learning curve for understanding the system as a whole. This is mitigated through good documentation and tutorials. Perhaps a graphical Puppet management interface would make the system even easier to use.

6 References

- http://docs.puppetlabs.com/guides/introduction.html
- http://docs.puppetlabs.com/learning/ral.html
- http://puppetlabs.com/puppet/what-is-puppet
- http://www.aosabook.org/en/puppet.html
- http://www.slideshare.net/lkanies/portable-infrastructure-with-puppet