Puppet at MWT2

Sarah Williams
Indiana University

Puppet Overview

- "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." –PuppetLabs Puppet FAQ http://docs.puppetlabs.com/guides/faq.html
- Requires an already built node – does not handle OS install

Why Puppet

- Consistent file and data integrity monitoring across the cluster, including change alerts
- Ability to keep revisioned and backed up configuration files in a central repository
- Simplified deployment of new nodes and services
- Role-based configuration profiles
- Ability to share configuration components (“modules”) with other sites
- Integration with provisioning system (Cobbler) and virtual machine creation (KVM)

How it works

- Two components: client & server
  - Server can run as daemon: (Puppetmasterd), or within an existing web application as a rack application (apache+passenger) . MWT2 uses a rack application install, which is more complex to set up, but more stable. Puppetmasterd has a memory leak issue, so if using it is recommended to restart it daily.
  - Client runs on the node to be configured
  - Clients and server have SSL certificates for authentication, and to encrypt communications
- Server stores configurations in files on the server, provides them to client when requested
- Puppet client requires an already-built node (prebuilt with cobbler/rocks/etc or custom installed by hand)
  - Basic requirements must be fulfilled: Network Access, Ruby, Facter
- Client confirms each part of its configuration every time it runs, fixing those pieces which are out of sync
  - Client will not change anything which is not explicitly defined in its configuration
  - Can run as daemon or cron, but daemon has same memory leak issue as puppetmasterd. Cron strongly recommended.
  - Too many clients running at once can overwhelm the server. We stagger the updates across an hour window 9-10am, every weekday.

Configuration

- All configurations are defined on the server
- It is highly recommended to keep the whole puppet tree in a version control system such as CVS or Subversion
- Puppet config files are called 'manifests' and end in .pp
- The puppet config language is declarative and based on Ruby
- When a client checks in, the server matches the client to a node definition and passes that information to the client. The client then executes the objects associated with that node definition.
- Objects are not guaranteed to be executed in a linear fashion. Use require to ensure necessary steps occur in the correct order, but beware creating loops.

Manifests

- Object types include nodes (servers), cron, users, groups, services, packages, and files
- Special objects exec and notify
- Generic object class

```
service { 'cvmfs':
   enable => true,
   ensure => true,
   hasstatus => true,
   hasrestart => true,
   require => [ Package['cvmfs-0.2.68-1'],
                Package['cvmfs-init-scripts'],
                Package['cvmfs-keys-1.1-2'],
                Package['fuse.x86_64'],
                Service['autofs'],
                File['/etc/cvmfs/default.local'],
                File['/etc/auto.master'],
                File['/etc/fuse.conf']  ]
}
```

Modules

- Re-usable components, defined by combining manifests, and files
- Many modules for standard services available in Puppet Forge http://forge.puppetlabs.com/

```
/etc/puppet/modules
        sendmail/
            files/
                sendmail.mc
                submit.cf
            manifests/
                init.pp
```

MWT2 Setup – Organization

- /etc/puppet/manifests/
  - sites.pp: First manifest loaded. All other maniests are loaded form this file.
  - templates.pp: Collection of configurations commonly grouped together (i.e. a worker-node config, a storage-node template)
  - nodes.pp: Defines each client host and assigns them templates

```
import nodes.pp
import templates.pp
```

```
class iu::snode {
  include yum::mwt2::repo
  include baseclass
  include user::osgvo
  include sendmail::x86_64::mwt2
  include snmpd::iu
  include dcache::logrotate
  include rsync::dcachemeta
  include puppet::daily::cron
}
```

```
node 'iut2-s3' {
    include iu::snode
    include dell::delldset
}
```

MWT2 Setup

- /etc/puppet/modules/
  - Directories are the names of the modules

```
class sendmail::x86_64::mwt2 {
package { 'sendmail.x86_64': ensure => present }
package { 'sendmail.i386': ensure => absent }
service { "sendmail":
            enable => false,
            ensure => stopped,
            require => Package["sendmail.x86_64"]
}
file { "/etc/mail/submit.cf":
             source => "puppet:///modules/sendmail/submit.cf",
             owner => root,
             group => root }
}
```

MWT2 Setup (cont.)

A more complex module example:
http://repo.mwt2.org/viewvc/puppet/modules/xrootd/manifests/init.pp?revision=1.6&view=markup

- Most of our system configuration happens in Puppet. Cobbler provisions nodes with a minimum set of RPMs, including the puppet client, which auto-runs after install. We could do more configuration in Cobbler (RedHat kickstart files), but find it easier to maintain just one set of config files.
- All servers (worker and head node) are configured via Puppet. Some server software and configuration for head nodes is done by hand. We are working towards full automation. We have not tried automating a Pacman install.
- It is helpful to keep track of and limit the layers of abstraction built into the configuration. Too many layers of abstraction make the configuration difficult to understand & update. We are currently over-abstracted, working on combining redundant system classes. One way to accomplish this is using 'define'.

Define Statements

- The CVMFS module is a good example of a define which takes variables, assigns defaults and deploys a set of configuration statements related to a single topic.
- This one line configures all of CVMFS for a worker node.
  - *cvmfs::mount { cvmfsuc: cvmfs_http_proxy=>" http://uct2-grid1.uchicago.edu:3128;http://iut2-grid1.iu.edu:3128;DIRECT" }*
- Then the cvmfs::mount is as follows:

```
define cvmfs::mount(
  $cvmfs_http_proxy="http://uct2-grid1.uchicago.edu:3128;http://iut2-grid1.iu.edu:3128;DIRECT",
  $cvmfs_quota_limit=25000,
  $cvmfs_server_url="http://cvmfs.racf.bnl.gov:8000/opt/@org@,http://cernvm-webfs.cern.ch/opt/@org@",
  $cvmfs_repositories='atlas',
  $cvmfs_cache_base='/scratch/cvmfs')
}
```

- Note that the above example only overrides the cvmfs_http_proxy field, but leaves the rest of the fields alone. Another node or template might choose to override a different setting, or not override any of the defaults.

http://repo.mwt2.org/viewvc/puppet/modules/cvmfs/manifests/init.pp?revision=1.10&view=markup

Common tasks

- Add a new server:
  - Add host definition and assign templates in nodes.pp

  OR
  - Define in Cobbler and use the Management Classes field to assign templates.
- Remove a server from puppet control:
  - In nodes.pp, comment out the templates assigned to the host.
- Test configuration changes:
  - Time the test to not coincide with daily Puppet client run window
  - Apply changes to code, test synax with puppet –parseonly
  - Run on a test workernode with puppetd –test
- Force immediate update of client hosts
  - Use SSH to run puppetd –test on all nodes
  - Verify no errors by looking at SSH output or logs on syslog collector

Encryption

- The first run of puppet will generate the client certificate. If client certificate is lost, it must be deleted from the server side with 'puppetca –clean <hostname>' for a new one to be generated
  - When reloading a node, we usually delete and regenerate the cert rather than try to preserve it
  - If the cert on the server isn't cleaned before reload, the new cert will be rejected. To recover, clear the cert on the server, delete /var/lib/puppet on the client, and re-run the puppet client.
- We have autosign turned on for the domains we control. Otherwise you must sign each worker's cert on the server side. If you turn on autosign, make sure you define the domain to be only those machines you control.

Services Currently Configured

- Atlas WN Requirements (Compilers, libraries)
- Hardware monitoring with Ganglia, Dellsrvadmin, Smartd, Lmsensors, customized for each hardware configuration
- Hardware optimization with hdparm, blockdev, kernel & ext4 tunings
- Database & system disk backups
- CVMFS
- dCache
- Ldap
- Nagios probes
- NFS
- PBS
- Condor
- Sendmail
- Snmpd
- Ssh
- Syslog
- Xrootd
- Yum

Future Plans

- Integration with Cobbler, DNS and DHCP via the External Nodes feature in puppet, would allow us to define a node in Cobbler and have DNS, DHCP and Puppet automatically configured.
- Move the above management services to single KVM
- Generate daily report from Syslog-ng collector on how many nodes updated vs failed to update, with errors

References

- Puppet style guide
  http://docs.puppetlabs.com/guides/style_guide
- Puppet best practices
  http://projects.puppetlabs.com/projects/1/wiki/Puppet_Best_Practice