Notes on Puppet - CSE 408222 – Cloud Computing

Caution: These are high level notes that I use to organize my lecture. You may find them useful for reviewing main points, but they aren't a substitute for participating in class.

Excellent introduction to Puppet is here:
https://docs.puppetlabs.com/learning/introduction.html

Introduction

Challenge of managing many machines; manual modification does not scale up!

Problem of synchronizing at scale: how to deal with machines online/offline, machines newly added, machines with different operating systems or roles.

Key Idea: Declare the goal state of system configuration outside of the system itself.

A good idea found in Make, cfengine, Chef, Amazon EC2, and many other systems.

Puppet High Level Architecture

Record system state in a manifest in source control.

Puppetmasterd compiles the manfest into a catalog.

Agents at each machine pull the catalog, observe the local state, and update as necessary.

Unauthorized local changes are not prevented, but they are reversed!

Example: User Info

A **resource** is an abstract description of something:

```
user { 'dthain':
        ensure => present,
        uid => '382',
        gid => 'profs',
        shell => '/bin/bash',
        home => '/home/dthain’
}
```

A **provider** is the implementation detail of that resource:

Stored in /etc/passwd, LDAP service, NetInfo database, etc…

Query the local provider:

```
puppet resource user dthain
```

Update the local provider:

```
puppet resource user dthain shell=”/bin/tcsh”
```

Example: Files

Important: Things not mentioned are left unchanged!
An item with details but no ensure value will only be modified if it exists.
Examples:

```
file {'/tmp/test1': ensure  => file,  content => "Hi.\n", }
file {'/tmp/test2': ensure => directory, mode   => 0644 }
file {'/tmp/test3': ensure => link, target => '/tmp/test1', }
user {‘dthain’: ensure => absent, }
```

Ordering of Dependencies

Ordering is not significant unless specifically stated.
Explicit keywords can be given:

Requires means the named resource must come first.
Before means the named resource must come later.
Notify means the named resource must refresh after an update.
Subscribe means the named resource will cause this one to refresh.

(Refresh asks it to look again at the environment, typically SIGHUP.)

```
File[‘/etc/passwd’] might say:
    Before => Service[‘login’]
Service[‘login’] might say:
    Require => File[‘/etc/passwd’]
File[‘sshd_config’] might say:
    Notify => Service[‘sshd’]
Service[‘sshd’] might say:
    Subscribe => File[‘sshd_config’]
```

Shortcut syntax:

```
File[‘passwd’] -> Service[‘login’]
File[‘sshd_config’] ~-> Service[‘sshd’]
```

There are some implicit autorequire rules, such as parent directories come before children.

Package-File-Service Idiom:

```
package { 'openssh-server':
    ensure => present,
        before => File['sshd_config'],
    }

    file { 'sshd_config':
        path => '/etc/ssh/sshd_config',
        ensure => file, mode => 600,
        source =>'/root/examples/sshd_config',
    }

    service { 'sshd':
        ensure => running,
        enable => true,
        subscribe => File['sshd_config'],
    }
```

Variables (Constants)

$name can be string, number, boolean, array, hash, or undef

Facts are predefined variables: ipaddress, fqdn, os, osfamily, memory, etc.

Facter gleans facts from the environ and makes them available to the compiler.

```
        file {'motd':
          ensure  => file,
          path    => '/etc/motd',
          mode    => 0644,
          content => "This Learning Puppet VM's IP address is ${ipaddress}. It thinks its
         hostname is ${fqdn}…"
        }
```

Expressions

if(expr) { } elsif(expr) { } else { }

expr can be anything that evaluates true/false, however, facts are always strings, and must be explicitly converted to Boolean: e.g. str2bool("$is_virtual")

case $variable { value1: {code} value2: {code} ... default { code }

```
        case $operatingsystem {
          centos, redhat: { $apache = "httpd" }
          debian, ubuntu: { $apache = "apache2" }
          default: { fail("Unrecognized operating system for webserver") }
        }
```

Note that this is all declarative rather than procedural. Variables (constants) cannot change: there is no looping, all arithmetic is constant at compile-time, etc.

Overall Structure

Define a bunch of related things, without taking effect:

```
class condor {   /* packages, files, services */  }
```

Cause everything in the class to take effect:

```
include condor
```

Many system configurations are simply a list of include commands.

A module is a directory with an init.pp file, containing the class definition. It also contains any other elements of the configuration that must be relayed to the individual machines. For example, the condor config file might be this:

```
file { ‘condor.config’:
        path => ‘/etc/condor.config’,
        ensure => file,
        source => “puppet:///modules/condor/condor.config”
}
```

The Puppet Forge (forge.puppetlabs.com) is a repository of modules

Specializing Configurations

Ways of indicating file content:

```
Source => “/some/local/path”,
Source => “puppet:///some/master/path”,
Source => template(“/some/template/path”)
```

A template allows variables in the puppet configuration to be substituted into the content of a given configuration file. For example:

If the puppet template contains:

```
class condor ( $matchmaker = “condor.cse.nd.edu” ) {
        file { “condor.config”:
                content = template(“condor.config.template”)
        }
}
```

And condor.config.template contains:

```
CONDOR_HOST = <%= matchmaker %>
```

The result is:

```
CONDOR_HOST = condor.cse.nd.edu
```

In short, the contents of the machine config files can be cleanly specialized directly from the puppet configuration.

Configuring Groups of Machines

```
node 'www.cse.nd.edu' { include common; include apache }
node 'ws1.nd.edu', 'ws2.nd.edu' { include condor; include develop }
```

Putting it All Together

In principle, never modify a machine directly!
Instead, create a VM image, modify it once to install a puppet agent.
For each machine role, write a puppet template that describes everything.
Put the templates in version control -> “Undo” for system administrators.