The Puppet Show
Managing Servers with Puppet

A gentle introduction to Puppet;
How it works,
What its benefits are
And how to use it

Robert Harker
harker@harker.com

whoami

UNIX/Linux sysadmin for over 25 years

Known for sendmail classes

My puppet experience

I used puppet to migrate a gaming social site 1up.com to new hardware in a lights-out data center.

Three types of servers:

- Apache/Tomcat front end servers
- Tomcat search servers
- Terracotta caching servers

Two infrastructure servers:

- Puppetmaster, named master, sendmail relay
- Nagios, MRTG, cacti

Puppet is a configuration management tool

Written by Luke Kanies

Supported by Puppet Labs (formerly Reductive Labs)

Written in Ruby but Ruby skills not needed

Client/Server model:

puppetmasterd is the central management daemon

puppetd runs on each managed system (node)

Why this is better than homebrew scripts/ssh loops

Much more testing than a single site can do

Very active user/developer community

Companies are able to hire people that already understand the system

“Puppet, Make It So”

The goal of Puppet is to define the end state of the managed system (node)

This end state is defined by a set of related class data that build up a node specific manifest

Puppetd running on each node inspects itself with factor which defines: OS, network interfaces, IP addresses, file systems, architecture, hostname

The node uploads this information to the puppetmaster

The puppetmaster then uses the factor information to:

Define which resources (classes) need to be applied to the node

Evaluate the resources to make it specific to a node

Build up a node specific manifest and download it to the client

This may include files and templates that are available on the puppetmaster

“Puppet, Make It So” (cont.)

The node then uses this manifest to compare its current configuration to the configuration defined in the manifest

Any differences are logged and corrected:
- Download a file
- Change owner or permissions
- Install or deinstall an application
- Enable or start a service

Because puppet strives to make the current state of the client be the same as the state defined in the manifest, puppetd can be run multiple times without corruption

Puppet is idempotent

Puppet Security Infrastructure

Puppetmasterd listens to port 8140

Puppet uses SSL for security

The puppetmaster includes a SSL certificate server

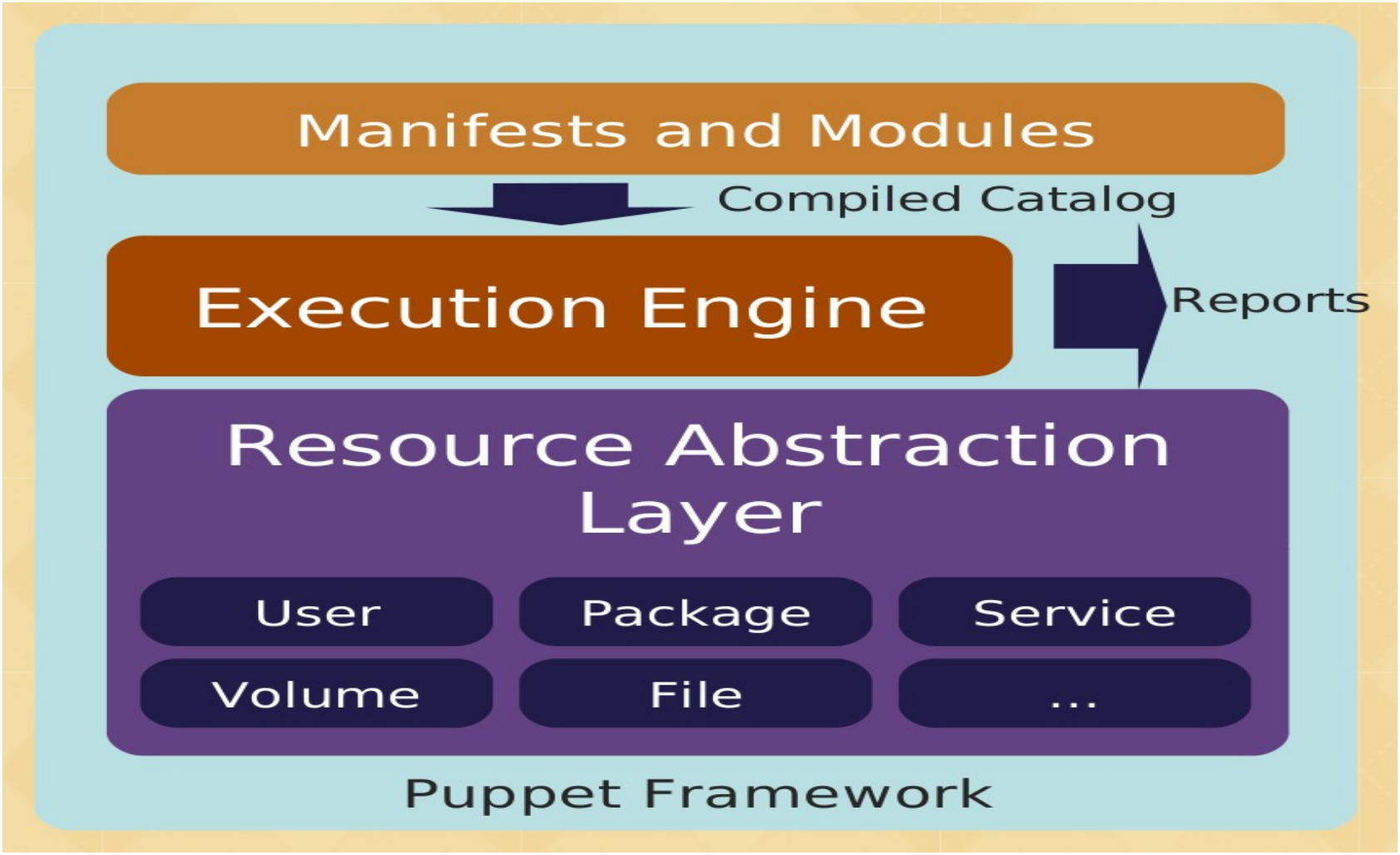
When a node starts up that does not have a certificate,
it sends a certificate request to the puppetmaster

The puppetmaster generates an SSL certificate and signs it

The puppetmaster returns the signed certificate to the node
The puppetmaster's public certificate is also returned

Certificates can be viewed and manipulated with puppetca

Puppet Architecture



Puppet Language

Puppet decouples the syntax of the configuration management tool from the syntax of the underlying OS and applications

This is done with Puppet's Resource Abstraction Layer (RAL)

The decoupling allows Puppet to define a high level idea like user, application, or service

The Puppet RAL will translate that in to the commands required by the client OS

Resources describe some aspect of a system

Each resource has a type, a title, and a list of attributes

Puppet Is A Declarative Language

Resource Types - Each resource is modeled as a type

Resources include services, packages, files, users, permissions, run state

Resource Types are independent of the underlying OS semantics

Providers how to implement a resource

Providers hide the underlying differences between OSes

Providers for the type 'package' include apt, yum, pkgadd

Resource providers actually perform the management functions

Puppet Language Structure

Puppet creates a graph based structure to define the relationships between resources:

What resources depend on another

What order the resources should be evaluated

Whether a change in one resource requires an action by another resource
I.e. restarting a service if a config file changes

Puppet has a tool to print out these dependency graphs

Puppet Syntax

Resources are made up of a type, title and a series of attributes:

```
  file { 'sshd_config':
    owner => 'root',
    group => 'root',
  }
```

Type is `file` Title is `sshd_config`

Attributes define the owner and group as `root`

Puppet allows you to specify a local name in addition to the title:

```
file { 'sshd_config':
  name => $operatingsystem ? {
    solaris => '/usr/local/etc/ssh/sshd_config',
    default => '/etc/ssh/sshd_config',
  },
...
```

Puppet Syntax, `Type[title]`

The title of a resource can be used to refer to file resource
The resource will include its attributes and OS specific logic

You use the syntax:

```
Type[title]
```

For example to define a service that depends on the file:

```
service { 'sshd':
  subscribe -> File[sshd_config],
}
```

`subscribe` tells Puppet to restart the service if its dependent resource is changed

Puppet Syntax, `Type[title]` (cont.)

This syntax can also refer to several related resources of the same type

```
service { 'sshd':
  require -> File['sshd_config', authorized_keys'],
}
```

Require defines the dependency order:

Nothing can happen to the service `sshd` until the `sshd_config` and `authorized_keys` files are correct

Puppet Can Manage Users And Groups

```
group { "harker":
  ensure => present,
  gid => 1318
}

user { "harker":
  ensure => present,
  gid => "harker",
  groups => ["adm", "staff", "root"],
  membership => minimum,
  shell => "/bin/bash",
  require => Group["harker"]
}
```

Puppet Nagios Types

Puppet can generate nagios configuration files based on node configuration

Nagios types supported:

```
nagios_servicedependency
nagios_hostescalation
nagios_serviceextinfo
nagios_hostgroup
nagios_hostextinfo
nagios_hostdependency
nagios_service
nagios_contactgroup
nagios_contact
nagios_command
nagios_timeperiod
nagios_servicegroup
nagios_host
nagios_serviceescalation
```

Resource Collections

Aggregation combines multiple resources into a new resource

Two ways to do this: Classes and definitions.

Classes model fundamental aspects of nodes

Classes define the resources that define an aspect of a node

Classes are are singletons are only evaluated once per node

Definitions can be reused many times on the same node

They work as custom created Puppet types

They can be evaluated multiple times with different inputs each time

You pass variable values into the defines.

Puppet Classes

Puppet classes define how to install and configure files, applications, services, etc...

A class is defined with:

```
class Title {
}
```

Resources are then added to the class

Puppet Classes (cont.)

A class can have multiple resources:

```
class sshd {
  package { openssh-server: ensure => present }
  file { 'sshd_config':
    name => $operatingsystem ? {
      solaris => '/usr/local/etc/ssh/sshd_config',
      default => '/etc/ssh/sshd_config',
    },
    owner => root,
    group => root,
    mode => 444,
    backup => false,
    source => "puppet:///files/etc/ssh/sshd_config",
    require => Package["openssh-server"],
  }
  service { "sshd":
    enable => true ,
    ensure => running,
    subscribe => [Package[openssh-server], File["sshd_config"],],
  }
```

Installs the package

Installs a configuration file

Starts the service

```
}
```

Puppet Modules

A Puppet Module is a reusable collection of resources, classes, files, definitions and templates

A module by nature should be self-contained

A Puppet module has a specific directory structure:

```
MODULE_PATH/
        downcased_module_name/
                files/
                manifests/
                        init.pp
                lib/
                        puppet/
                                parser/
                                        functions
                                provider/
                                type/
                        facter/
                templates/
```

README

Puppet Modules (cont.)

Each module must contain a `init.pp` manifest file

```
class ntpd {
        package { ntp: ensure => latest }
        service { ntp: ensure => running }
        file { "/etc/ntp/ntpservers":
                source => "puppet://
  $servername/modules/ntp/ntpservers"
        }
        file { "/etc/ntp.conf":
                content => template("ntp/ntp.conf.erb")
        }
}
```

Module Structure For `ntpd`

```
MODULE_PATH/
        ntp/
                manifests/
                        init.pp
                files/
                        ntpservers
                templates/
                        ntp.conf.erb
```

Puppet Templates

Modules can also edit files on the fly

In `ntpd/templates/ntp.conf.erb`:

```
# /etc/ntp.conf, configuration for ntpd
. . .
fudge 127.127.1.0 stratum <%= local_stratum %>
. . .
```

The ntpd module's `init.pp` recipe file:

```
. . .
    $local_stratum = $ntp_local_stratum ? {
        '' => 13,
        default => $ntp_local_stratum,
}
. . .
```

Puppet Manifests

Puppet has three types of manifests files:

Nodes: define which classes each managed node should use

Classes: action files that define what to do

Modules: reusable classes

Nodes define what packages, files and services should be installed

Classes and modules define what need to be done to install it

Classes and modules are added to a node with an include statement

```
include sshd.pp
```

Nodes Have Inheritance

A complex node can be configured by inheriting a simpler node

I start with a basenode that all hosts inherit

This includes things I want done on all nodes:

Applications installed or removed

Services enabled or disabled

Site wide configuration files

You can then make a more complex node based on this inheritance

Webserver = basenode + apache

MySQLserver = basenode + MySQL

You can then make a specific node or host:

fooMysql = MySQLserver + foo specific additions

barMysql = MySQLserver + bar specific additions

/etc/puppet/manifests/nodes.pp

```
node 'basenode' {
  # nodefiles contain host specific files such as host ssh keys
  include nodefiles
  # Custom puppet configuration for puppetd nodes, not on puppetmaster
  include puppet-configs
  # Only download rpms from our private repos
  include yumrepos
  # Install and remove packages from the core OS install
  include baseapps
  # Enable and disable system services
  include basesrvcs
  # Classes that have custom configurations
  include iptables
  include hosts
  include nrpe
  include ntp
  include snmp
  include subversion
  include dell
  include sysfiles
  include sshd
  include rootfiles
  include java
}
```

```
#########################################################
# Foo Domain and role nodes
#########################################################
node 'foodomain' inherits basenode {
  # Things that should be in all Foo servers that do not provide
  #    datacenter infrstructure services like named, sendmail, etc

  # set the local servers we point to
  # Lets use FQDNs
  $my_puppet_server = "opssrv.bil.foo.com"
  $my_syslog_server = "nagios.bil.foo.com"
  $my_ntp_server = "opssrv.bil.foo.com"

  $my_local_network = "10.212.62.0/24"

  # These include statements are including classes
  # Order can be important, so be careful
  include named
  include sudo
  include passwd
  include homedirs
  include sendmail
```

```
}
# Roles for different types of servers
node 'web-role' inherits foodomain {
  # Things that are specific to the apache servers
  include appsrv-role
  include appsrv-apache2
}

node 'tomcat-role' inherits foodomain {
  # Things that are specific to the tomcat servers
  include appsrv-tomcat
}

node 'web-tomcat-role' inherits foodomain {
  # A role that inherits apache
  # It needs the tomcat stuff repeated here as well
  include appsrv-role
  include appsrv-apache2
  include appsrv-tomcat
  include terracotta
}

node 'terracotta-role' inherits foodomain {
  # Things that are specific to the terracotta servers
  include terracotta
```

```
}
#########################################################
# nodes that are actually hosts
#########################################################

node 'opssrv.bil.foo.com', 'mgmt.bil.foo.com' inherits basenode {
  # Things that are specific to the management server
  include opssrv
  include puppet-master-configs
}

node 'nagios-01.bil.foo.com' inherits basenode {
  # Things that are specific to the montoring server
  include httpd
  include nagios-server
}

node 'appsrv-01.bil.foo.com', 'appsrv-02.bil.foo.com',
    'appsrv-03.bil.foo.com', 'appsrv-04.bil.foo.com',
    'appsrv-05.bil.foo.com', 'appsrv-06.bil.foo.com',
    'appsrv-07.bil.foo.com', 'appsrv-08.bil.foo.com'
  inherits web-tomcat-role {
  # Things specific to the apache/tomcat applications servers
```

```
}

node 'tcsrv-01.bil.foo.com', 'tcsrv-02.bil.foo.com'
  inherits terracotta-role {
  # Things specific to the terracotta servers
  include tcsrv-role
}

node 'search-01.bil.foo.com' inherits tomcat-role {
  # Things specific to the tomcat search servers
  include search-role
  include search-tomcat
}

node 'download-01.bil.foo.com', 'download-02.bil.foo.com'
   inherits web-role {
  # Things specific to the apache download servers
  include download-role
  include download-apache2
}
```

Pros and Cons of Puppet

Pros:

Buzz in the Linux systems management community

OpenSource

Active development community

New features and modules added weekly

Cons:

Scalability

- Factor running on the client is expensive
- Manifest generation on the puppetmaster is expensive

Many desirable features missing

- Dashboard is alpha, likely to be commercial product
- Many systems administration tasks missing