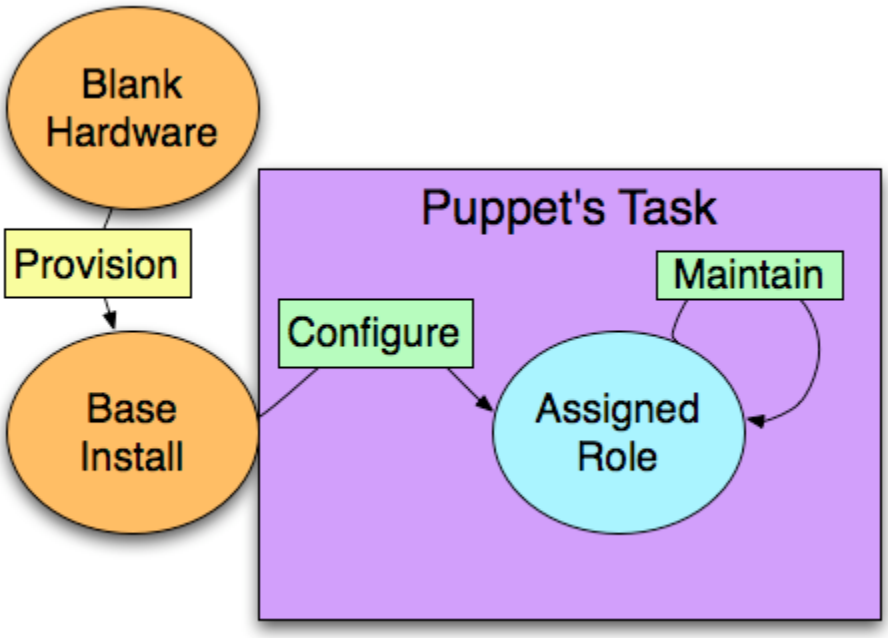
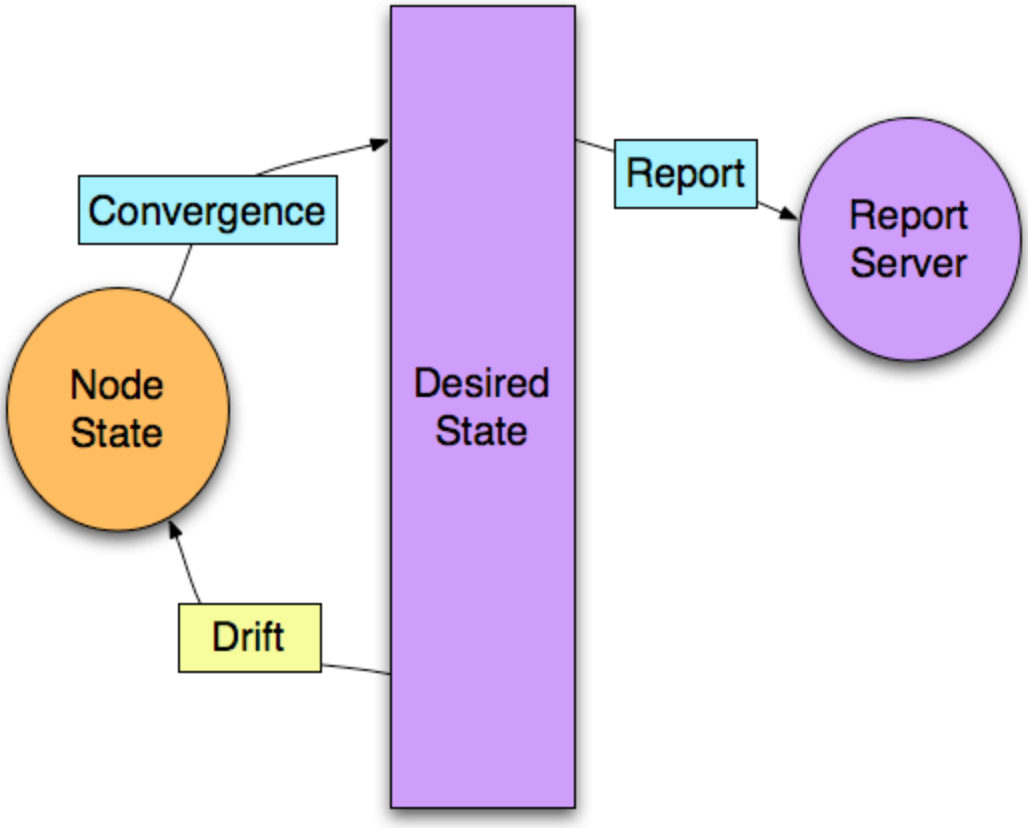
SCaLE Puppet Introduction

Puppet Assigns and Maintains a Machine's Desired Role



Managing Configuration Drift



Puppet Executables that we will employ:

- `ralsh` – The Resource Abstraction Layer Shell.
- `facter` – Executable and library that discovers facts about client systems.
- `puppet` – Executable that interprets Puppet manifests, compiles the catalog, and applies the catalog locally.
- `puppetmasterd` – Centralized daemon that authenticates client connections, serves files, compiles templates, and provides puppet clients with a catalog.
- `puppetd` – Puppet daemon that runs on client machines, makes connections to the puppetmaster, retrieves the catalog, and applies that catalog locally.
- `puppetca` – Puppet's built-in certificate authority.

Resources

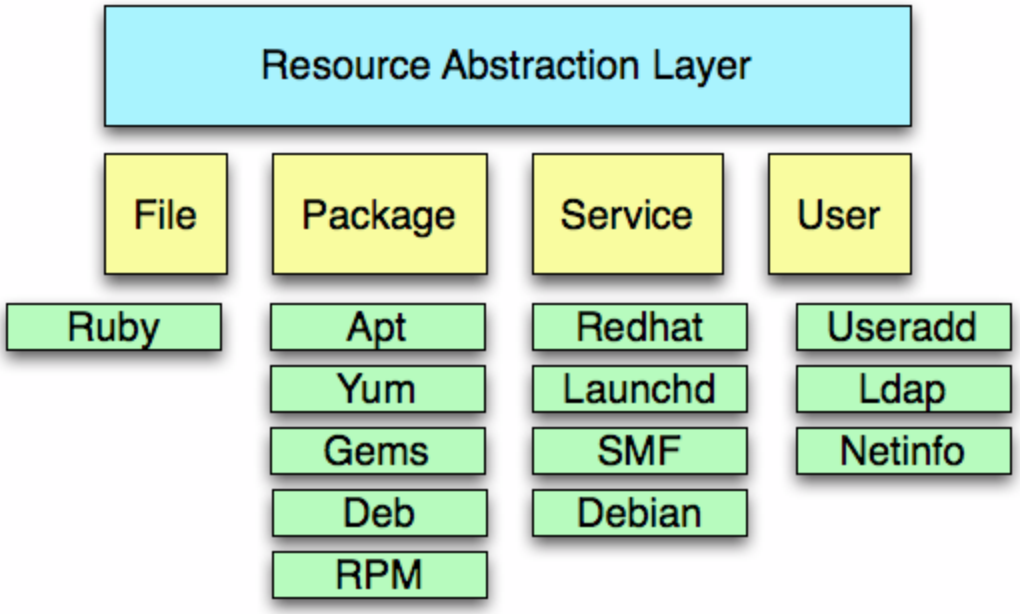
Resources are the building blocks Puppet uses to model system configurations.

Simple user resource declaration.

```
user{'redmine':
  ensure => present,
  shell  => '/usr/sbin/nologin',
}
```

RAL: Resource Abstraction Layer

The RAL provides a consistent model for resources across supported platforms.



RAL: Resource Abstraction Layer

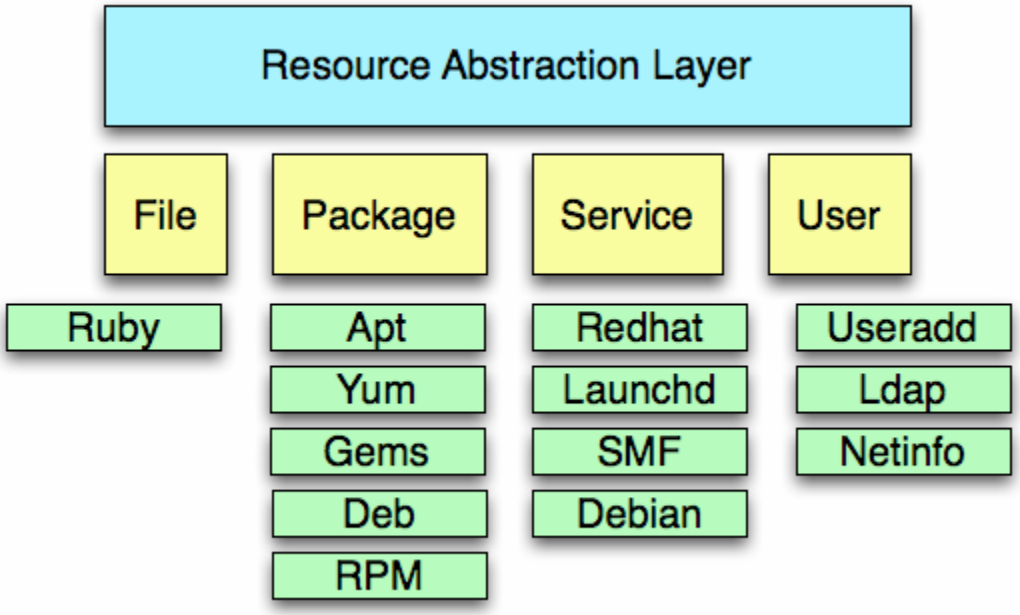
Resource types depend on providers to translate specification into implementation.

```
package{'rubygems':
  ensure => installed,
}
```

Package is just one of the many native Puppet resource types.

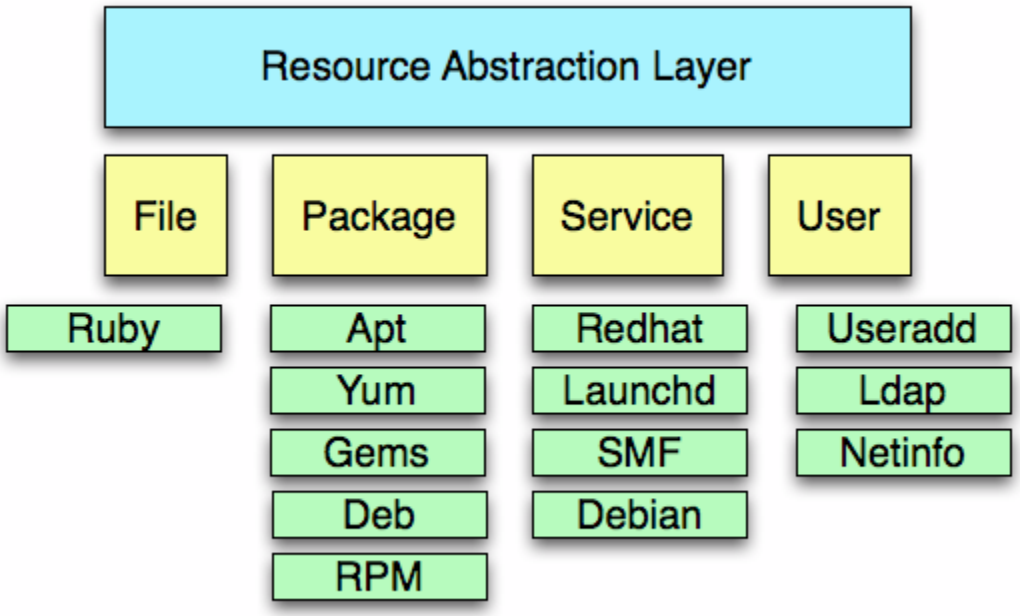
RAL: Resource Abstraction Layer

Each resource type has one or more providers.



RAL: Resource Abstraction Layer

Providers are the interface between the underlying OS and the resource types.



RAL: Resource Abstraction Layer

The package resource type has 23 providers:

```
ls -1 /usr/lib/ruby/site_ruby/1.8/puppet/provider/package

appdmg.rb       freebsd.rb  sunfreeware.rb
apple.rb        gem.rb      sun.rb
aptitude.rb     hpux.rb     up2date.rb
apt.rb          openbsd.rb  urpmi.rb
aptrpm.rb       pkgdmg.rb   yumhelper.py
blastwave.rb    portage.rb  yumhelper.pyc
darwinport.rb   ports.rb    yumhelper.pyo
dpkg.rb         rpm.rb      yum.rb
fink.rb         rug.rb
```

Ralsh: The Resource Abstraction Layer Shell

Executing `ralsh` and providing a resource and a title returns the state of a resource.

```
root@puppetclient:~$ ralsh user redmine

user { 'redmine':
  ensure => 'absent'
}
```

Ralsh: The Resource Abstraction Layer Shell

Executing `ralsh` and providing a resource, a title, and specifying an attribute alters the resource.

```
root@puppetclient:~$ ralsh user redmine ensure=present

notice: /User[redmine]/ensure: created
user { 'redmine':
    uid => '500',
    password => '!!',
    gid => '500',
    home => '/home/redmine',
    shell => '/bin/bash',
    ensure => 'present'
}
```

Core Resource types:

- user
- group
- host
- cron
- exec
- file
- package
- service
- mount
- tidy

Core Resource types:

The type reference documentation can also be found on the Reductive Labs website.

http://docs.reductivelabs.com/guides/types/index.html

The User Resource Type

Some basic attributes for the user resource type:

- name: OS specified limits apply. (namevar)
- ensure: Sets the basic state of the user resource. Valid values are `absent`, `present`.
- gid: The user's primary group. Can be specified numerically or by name.
- groups: The secondary group or groups to which the user is assigned. The primary group should not be listed. Multiple groups should be specified as an array.
- home: The users home directory.
- managehome: Whether to manage the home directory when managing the user. Valid values are `true`, `false`.

The File Resource Type

Basic Attributes:

- path: Specifies the target location for file. (namevar)
- ensure: Accepts absent, present, file, and directory. Any other value will be treated as a symlink.
- owner: Owner of file.

- group: Group of file.
- mode: Mode of file
- content: Specifies the content of file as a string.
- source: Specifies the source of file.
- force: Force replacement of directories with a link. Valid values (true, false).
- ignore: Omits files matching specified patterns during recursion (Ex: .svn, .git).
- recurse: Whether or not directories should be managed recursively. Valid values (true, false)
- purge: Whether or not to purge unmanaged file resources within a directory. Valid values (true, false)

The File Resource Type

Simple file resource declaration with a local source.

```
file {'/etc/sudoers':
  ensure => file,
  group  => 'root',
  owner  => 'root',
  mode   => '440',
  source => '/etc/puppet/files/sudoers',
}
```

The File Resource Type

Directory example.

```
file {'/tmp/src':
  ensure => directory,
  mode   => '755',
}
```

The File Resource Type

Symlink example.

```
file {'/tmp/testfile':
  source => '/tmp/src/testfile',
}

file {'/tmp/testlink':
  ensure => '/tmp/testfile',
}
```

Facter and Facts

Puppet uses `facter` to gather information about the host system.

Facter and Facts

Executing the `facter` command returns a list of key value pairs.

```
root@puppetclient:~$ facter
architecture => x86_64
domain => reductivelabs.com
facterversion => 1.5.2
fqdn => puppetclient.reductivelabs.com
hardwaremodel => x86_64
hostname => aku
interfaces => eth0
ipaddress => 172.16.10.1
kernel => Linux
operatingsystem => Ubuntu
...
```

The returned key value pairs are `facts`.

The Puppet Executable

The standalone `puppet` executable:

- interprets puppet code

The Puppet Executable

The standalone `puppet` executable:

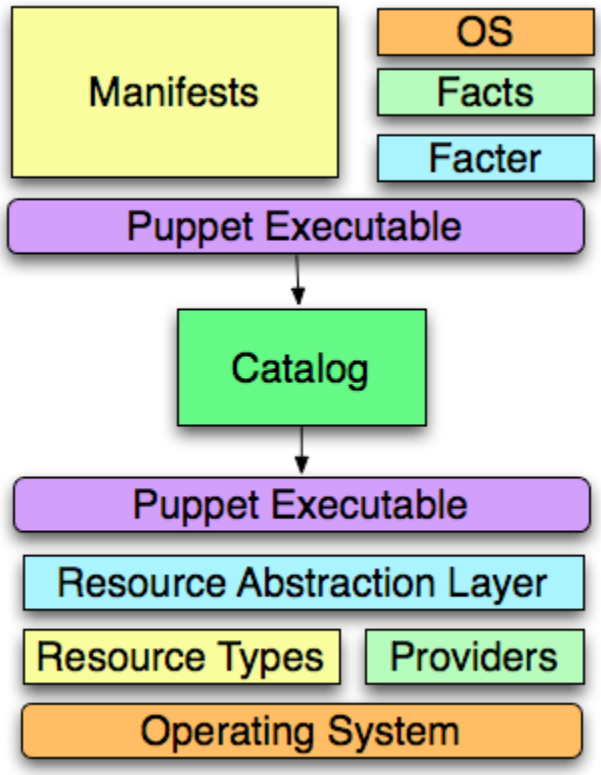
- interprets puppet code
- compiles a catalog

The Puppet Executable

The standalone `puppet` executable:

- interprets puppet code
- compiles a catalog
- uses the RAL to apply the catalog locally.

The Puppet Executable



The Puppet Executable

Files containing Puppet code are known as manifests and by convention have a .pp suffix.

The Puppet Executable

Example Puppet Manifest:

```
user {'elvis':
  ensure     => present,
```

```
  home       => '/home/elvis',
  gid        => 'elvis',
  shell      => '/bin/bash',
  managehome => true,
}

# A group resource definition
group {'foo':
  ensure => present,
}
```

Specifying Dependencies

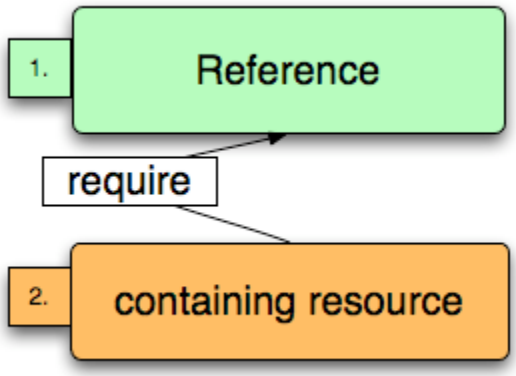
- Puppet is not a procedural language, it is declarative.
- All ordering dependencies between resources must be explicity specified.

Specifying Dependencies

The `require` and `before` metaparameters establish dependencies between resources.

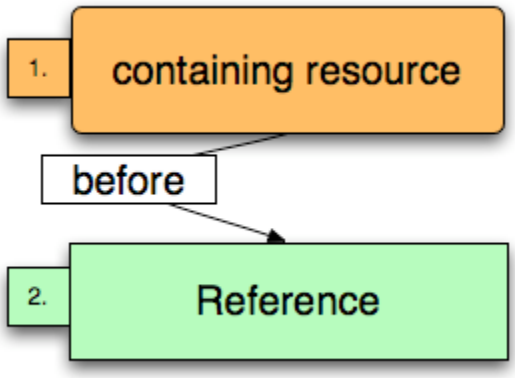
Specifying Dependencies

`require`



Specifying Dependencies

before

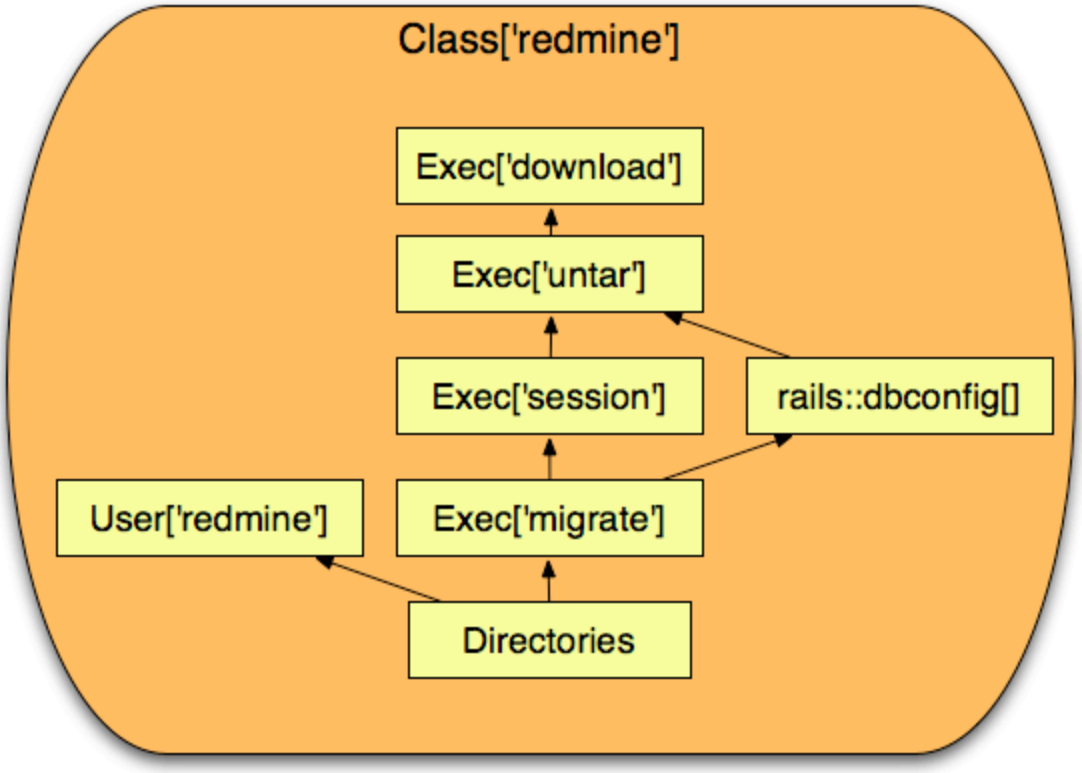


Specifying Dependencies

This ensures that permissions of these directories are managed only after the db migration task is run.

```
exec{'migrate':
  command => '/usr/bin/rake db:migrate',
  cwd     => $reddir,
  environment => 'RAILS_ENV=production',
  require => Exec['session'],
  creates => "${reddir}/db/schema.rb"
}
file{
  [ "${reddir}/public",
    "${reddir}/files",
    "${reddir}/log",
    "${reddir}/tmp",
    "${reddir}/public/plugin_assets"
  ]:
  ensure  => directory,
  recurse => true,
  owner   => 'redmine', group   => 'redmine', mode    => '0755',
  require  => Exec['migrate'],
}
```

Redmine Dependencies



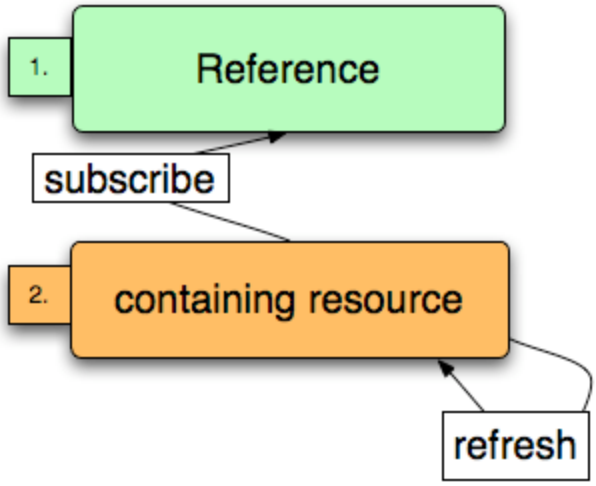
Specifying Dependencies

Resources can be refreshed by other resources.

The `subscribe` and `notify` metaparameters establish refresh relationships between resources.

Specifying Dependencies

The `subscribe` metaparameter establishes a refresh relationship from the containing resource to a change in the referenced resource.



Specifying Dependencies

This manifest ensures that mysqld is restarted only if /root/.my.cnf changes.

```
service{'mysqld-restart':
  restart => '/usr/sbin/service mysqld restart'
}
file{'/root/.my.cnf':
  content => template('mysql/my.cnf.erb'),
  notify    => Service['mysqld-restart'],
}
```

The Resources Resource Type

Using the `host` resource type we can specify specific host entries.

```
host {'kermit.reductivelabs.com':
  ensure        => present,
  host_aliases  => 'aku',
  ip            => '172.16.238.131',
}
host {'piggy.reductivelabs.com':
  ensure       => present,
  host_aliases => ['piggy', 'missy'],
  ip           => '172.16.238.132',
}
host {'oscar.reductivelabs.com':
  ensure => absent,
}
```

What if we only want to have explicitly declared entries in the /etc/hosts file?

The Resources Resource Type

If a resource is `ensurable` then the `resources` resource type can be used to enable purging of unmanaged resources.

The Resources Resource Type

This will purge all unspecified host resources.

```
resources {'host':
  purge => true,
}
```

The Resources Resource Type

Attributes:

- name: the name of the resource type that is to be managed. (namevar)
- purge: true or false
- unless_system_user: true, false, or some upper uid limit specified as an integer.

The Resources Resource Type

Exercise: Purging unmanage resources.

- Use `ralsh` to generate a manifest named hosts.pp in /etc/puppet/manifests.
- Edit hosts.pp to include a `resources` type that enables purging for the `host` resource type.
- Manually add a host entry to /etc/hosts.
- Use `puppet` to interpret the hosts.pp manifest and ensure that the unmanaged resource is purged.

The Service Resource Type:

Attributes:

- name: The name of the service as understood on the underlying services subsystem. (namevar)

- enable: If a service should be started at boot. Can be true or false.
- ensure: If the resource should currently be running. Can be true, false, running, or stopped.
- hasrestart: Specifies that your service has a restart command. Can be true or false.
- hasstatus: Specifies that your service has a status command. Can be true or false.
- pattern: The pattern to search for in the process table.
- restart: Specify a restart command.
- start: Specify a start command.
- status: Specify a status command.
- stop: Specify a stop command.

The Service Resource Type:

Example of a `service` resource type:

```
service {'sshd':
  enable     => true,
  ensure     => running,
  hasstatus  => true,
  hasrestart => true,
}
```

The Service Resource Type:

Exercise:

- Use `ralsh` to stop the sshd service.
- What happens if you execute the same ralsh command again?
- Set the parameter `hasstatus=true` and use ralsh to ensure sshd is stopped.
- Use `ralsh` to ensure that the sshd service is started. Be sure to use `hasstatus=true`.

Classes

Classes in Puppet are used to model fundamental aspects of nodes.

Classes

Example: ruby::dev class.

```
class ruby::dev {
  require ruby
```

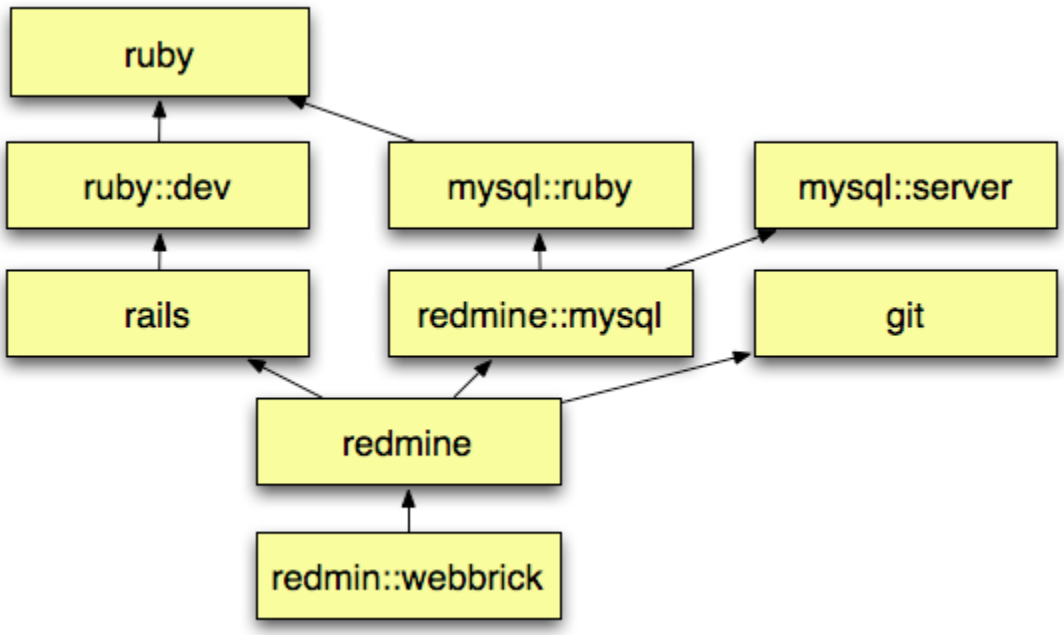
```
  package {['ruby-rdoc', 'ruby-irb', 'rubygem-rake']:
    ensure => installed,
  }
}
```

Classes

You can also create relationships to classes using `require` and `before`.

```
class redmine::webbrick {
  include redmine
  $redmine_port='3000'
  exec{'start-redmine':
    command => 'ruby script/server webrick -e production &',
    unless  => "netstat -ltn | grep ${redmine_port}",
    cwd     => $redmine::params::reddir,
    user    => 'redmine',
    path    => '/bin:/usr/bin',
    require => Class['redmine'],
  }
}
```

Redmine Class Relationships



Defined Resource Types

Defined resource types behave like custom resource types.

- Accepts Metaparameters
- Can be used multiple times

Defined Resource Types

Vhost example

```
define rails::db_config(
  $adapter,
  $database,
  $host='localhost',
  $username,
  $password,
  $encoding='utf8',
  $environment='production',
  $socket='/tmp/mysql.sock'
){
  file{"${name}/config/database.yml":
    content => template('rails/database.yml.erb'),
  }
}
```

Defined Resource Types

Using a defined resource example.

```
$reddir='/opt/redmine'
rails::db_config{$reddir:
  adapter  => 'mysql',
  username => 'redmine',
  password => 'password',
  database => 'redmine',
  socket   => '/var/run/mysqld/mysqld.sock',
}
```

Templates

Puppet uses Ruby's builtin templating, ERB.

http://www.ruby-doc.org/stdlib/libdoc/erb/rdoc/classes/ERB.html

Templates

Basic ERB for productions rails db configuration.

```
<%= environment %>:
  adapter: <%= adapter %>
  database: <%= database %>
  host: <%= host %>
  username: <%= username %>
  password: <%= password %>
  encoding: <%= encoding %>
  socket: <%= socket %>
```

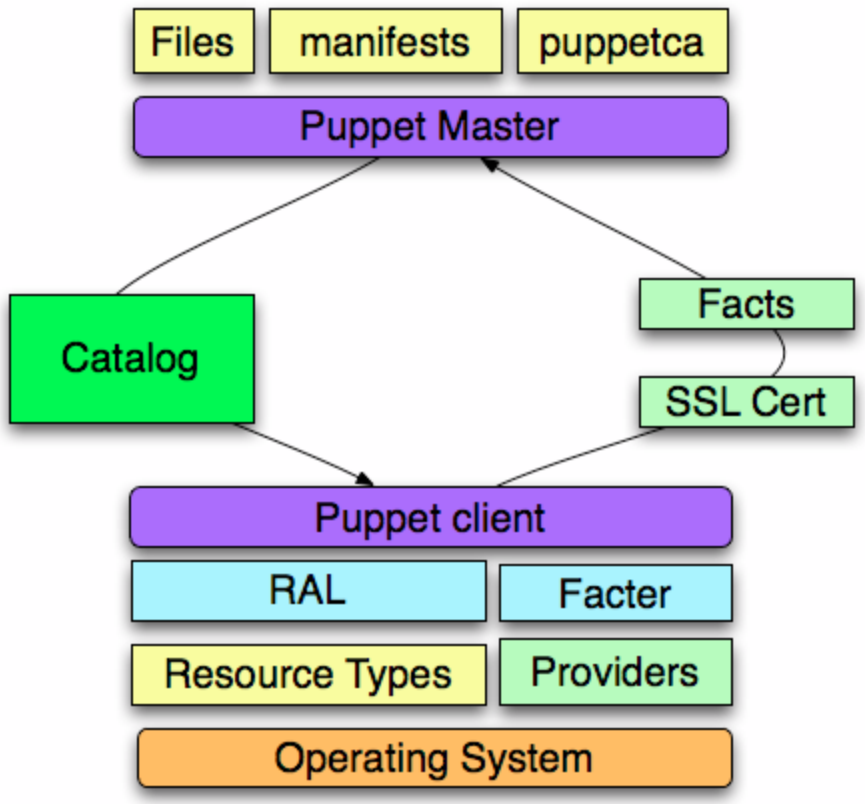
Templates

Basic ERB syntax: Iteration

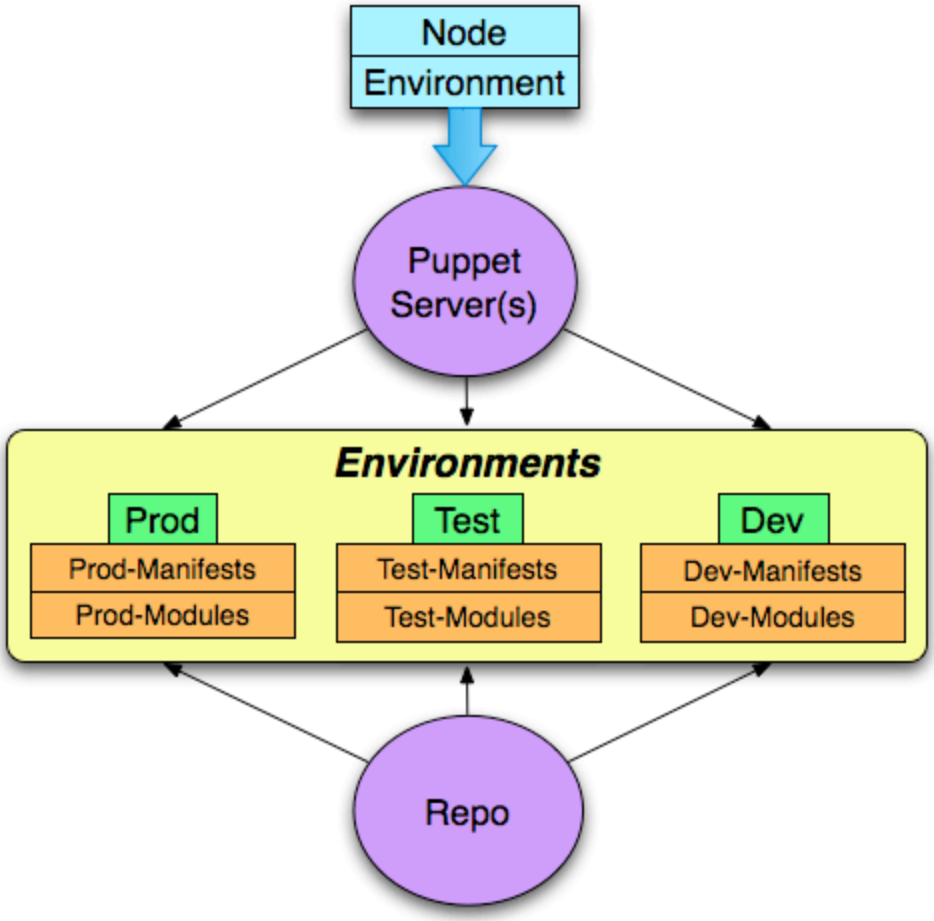
```
# We can also iterate over arrays

<% fooarray.each do |val| %>
   Foo array has a value of <%= val %>
<% end %>
```

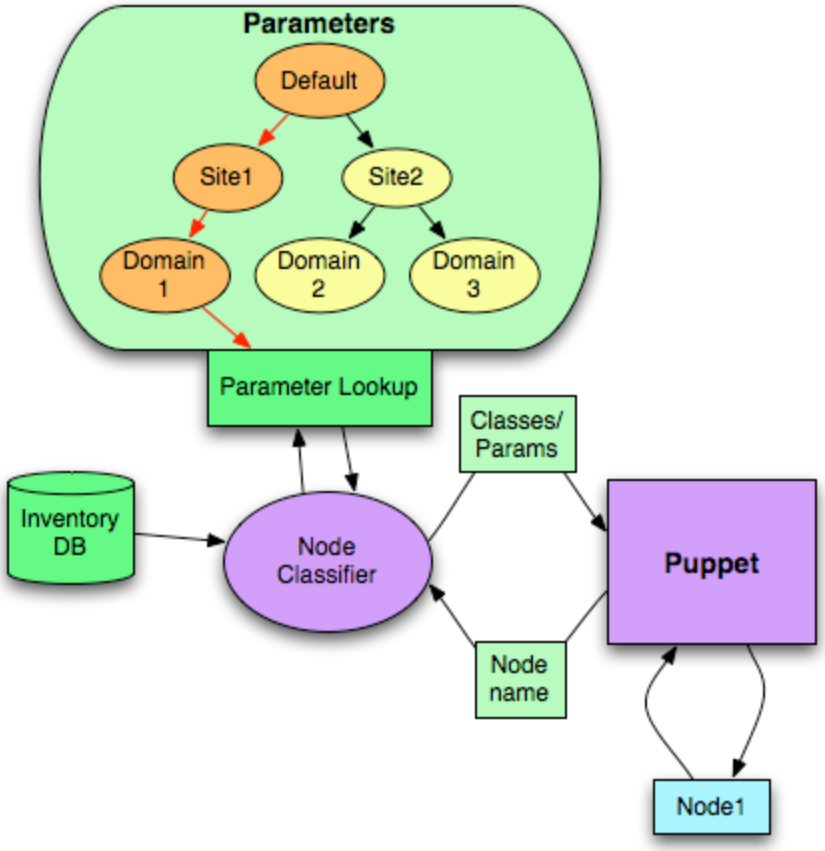
Client/Server Diagram



Environments



External Nodes Diagram

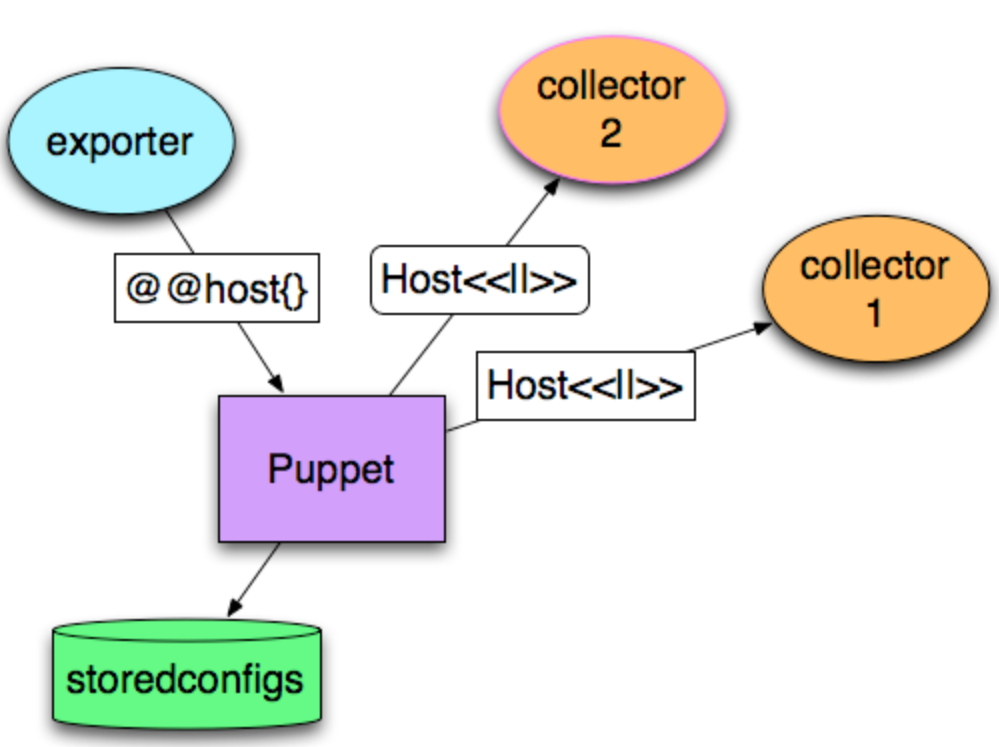


Exported Resources

Puppet has the ability to export resources to a database so that they can be collected and used on other hosts.

```
class hosts {
 @@host { $hostname: ip => $ipaddress, alias => $fqdn }
 Host <<||>>
}
```

Exported Resources



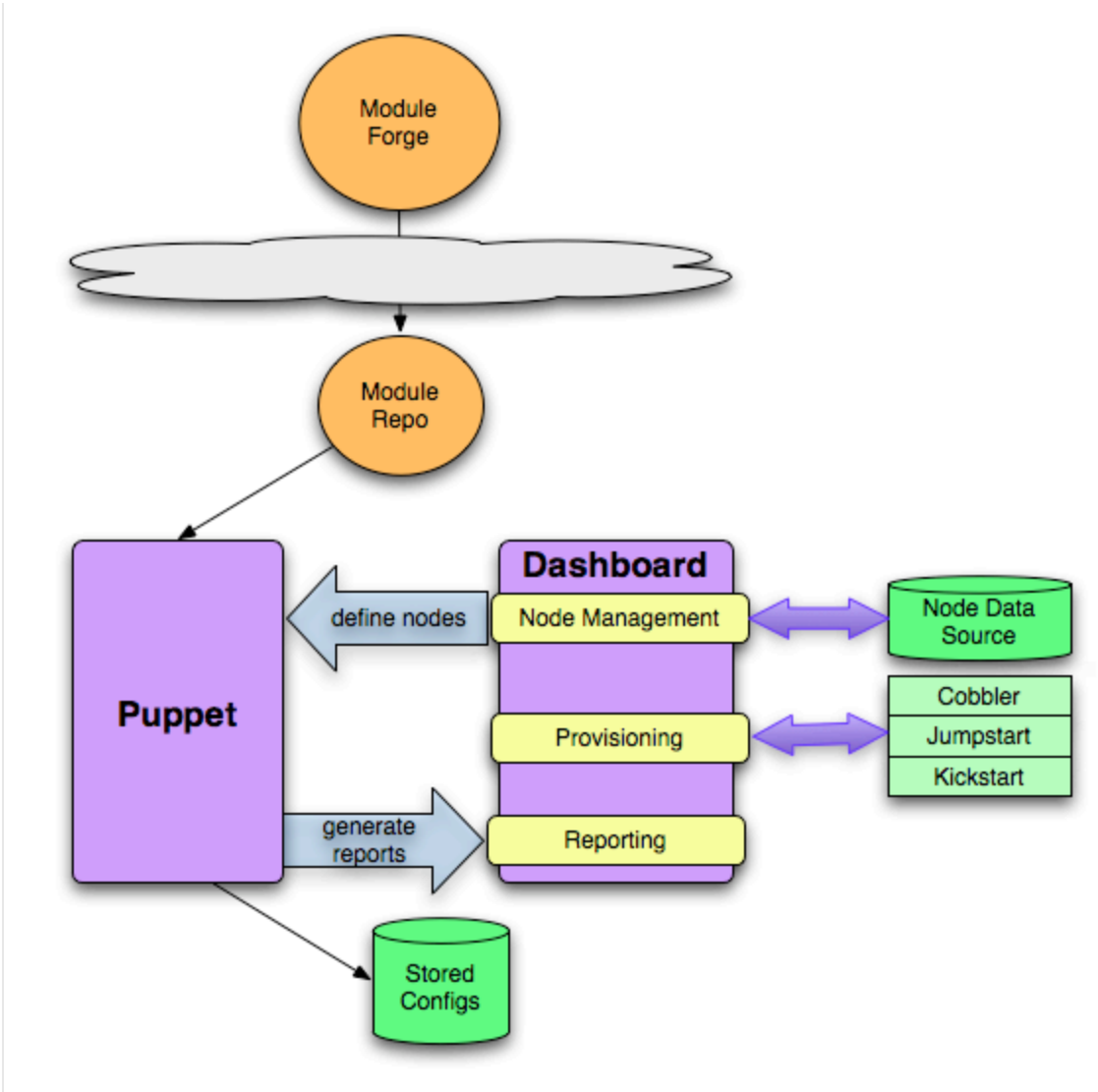
Exporting resources requires configuration of storeconfigs.

Exported Resources

But what if there is a host entry that we don't want to have in the /etc/host files.

- We can use the `resources` resource to purge rogue entries.
- Exported resources for decommissioned servers must be purged from the database using a script.

Future Architecture



I am known by many names

Dan Bode

dan <at> reductivelabs.com

bodepd <on> irc://chat.freenode.net/#puppet

Want to learn more?

http://reductivelabs.com/

http://docs.reductivelabs.com/

http://reductivelabs.com/trac/puppet