An Introduction to Puppet Enterprise

Puppet Education
www.puppetlabs.com/education

Course Objective

After completing the course, students will be able to discuss the benefits of the Puppet solution as it applies to their own system automation business needs.

Course Outline

About Puppet

Overview: About Puppet

Objectives

At the end of this lesson, you will be able to:

- Identify the main challenges of scalable and repeatable infrastructure management.
- Identify how Puppet or Puppet Enterprise can be used to overcome such challenges.
- Identify Puppet community resources available to you.
- Identify Puppet Labs' pragmatic and user driven approach towards configuration management.

About Puppet Labs

IT Automation Software for System Administrators:

- Founded in **2005**
- First commercial product release in **2011**
- **6,000** Community Members
- **20,000+** Users
- **50,000+** Nodes managed in the largest deployments
- Support for Red Hat, CentOS, Ubuntu, Debian, SUSE, Solaris 10, Windows, Mac OS X
- Investments from Google Ventures, Cisco, VMware, Kleiner Perkins, and True Ventures

---

Notes:

1. Puppet Labs has over 6 years of product and customer experience in the IT automation market.
2. We estimate that there are in total over 10 million nodes being managed worldwide by Puppet.
3. The number of users of open source software, paying customers, and partners all demonstrate the tremendous market momentum Puppet has - it's becoming the "de facto" standard.

Puppet Customers

FINANCIAL	TELCO/SP	INTERNET	TECH	GOVERNMENT
Bank of America	verizon	Google	CISCO	NASA
NYSE Euronext	MOTOROLA	zynga	McAfee	U.S. DEPARTMENT OF ENERGY
London Stock Exchange	rackspace	PayPal	QUALCOMM	NATIONAL INSTITUTES OF HEALTH
TransUnion		Constant Contact		

DEFENSE	EDUCATION	MANUFACTURING	RETAIL	MEDIA
LOCKHEED MARTIN	HARVARD UNIVERSITY		REI	Disney
NORTHROP GRUMMAN	Cal	BOEING	BARNES&NOBLE BOOKSELLERS	HBO
Los Alamos NATIONAL LABORATORY EST. 1943	University of Phoenix	HERSHEY'S The Hershey Company	NIKE	CONDÉ NAST

Notes:

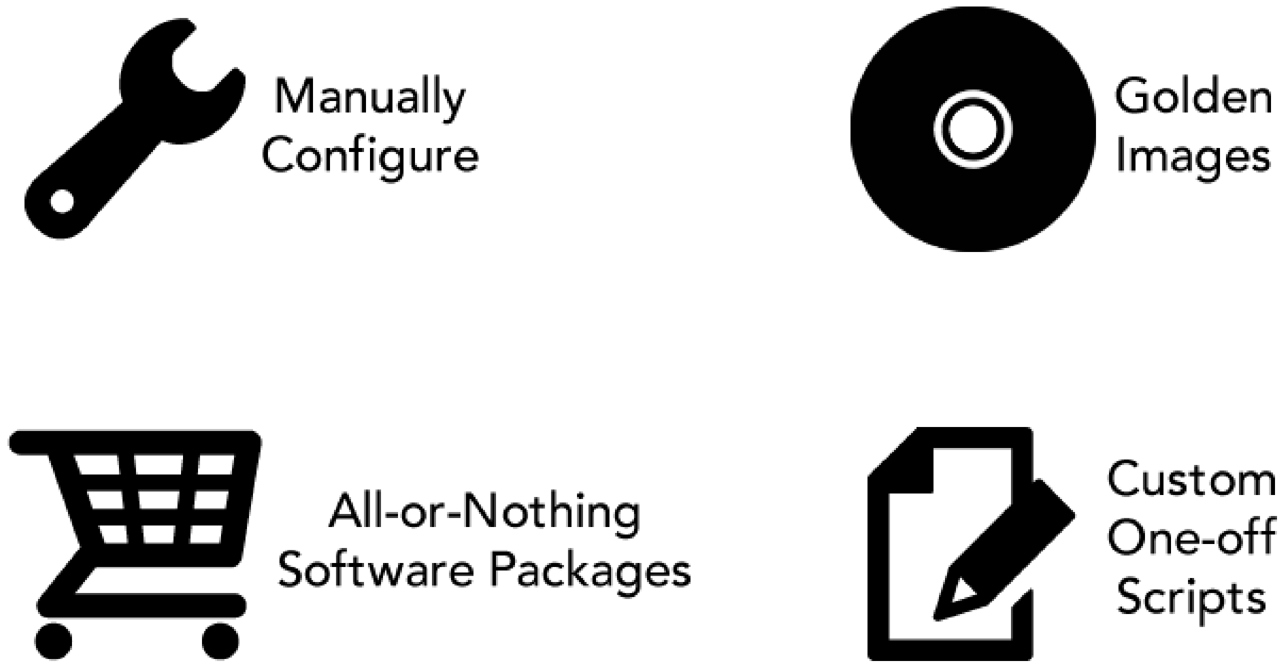
Some of the largest and fastest-growing companies in the world use Puppet, and it has been adopted for IT automation by users in every major vertical market.

- The Puppet Community includes:
  - Redhat (epel)
  - Debian (stable)
  - Ubuntu (main)
  - Solaris (opencsw)
  - Mac OSX (macports)
  - ... and many more
- Activity:
  - active mailing lists
  - puppet-users@googlegroups.com

    - puppet-dev@googlegroups.com
    - IRC channels, including community and Puppet employees
    - #puppet on freenode.net
    - #puppet-dev on freenode.net

- Contributions:
    - hundreds of community members involved in the project
    - hundreds of modules committed to the Puppet Forge

Current State of IT Automation



Notes:

To address infrastructure management challenges, IT automation today offers a number of techniques that often fall short, including the following:

- Manually Configure (literally logging in to every node to configure it)
    - Difficult to scale
    - Impossible, for all intents and purposes, to maintain consistency from node-to-node
- Golden Images (creating a single copy of a node's software and replicating that across nodes)
    - Need separate images for different deployment environments, eg, development, QA, production, or different geo locations
    - As number of images multiply it becomes very difficult to keep track and keep consistent
    - Since they're monolithic copies, golden images are rigid and thus difficult to update as the business needs change
- Custom One-off Scripts (custom code written to address a specific, tactical problem)

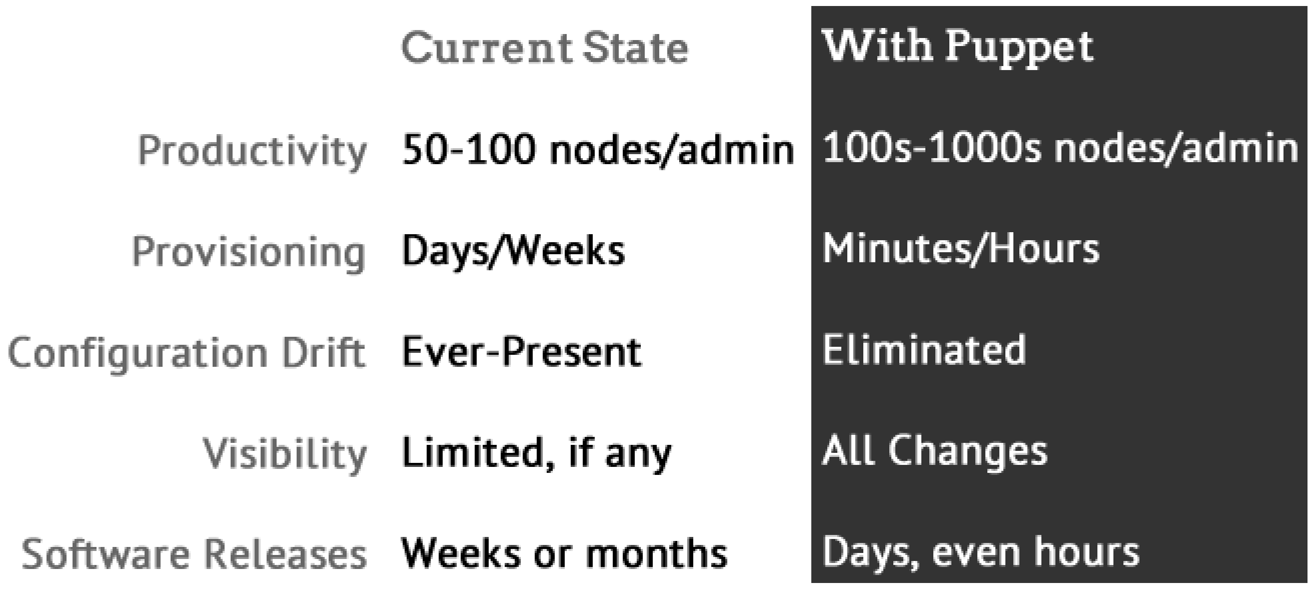
    - No leverage - effort typically cannot be re-used for different applications or different deployments
    - Brittle - as needs change, the entire script must often be re-written
    - Difficult to maintain when the original author leaves the organization

- Software Packages (typically all or nothing approach)
    - These packages typically require that all resources be placed under management - cannot selectively adopt and scale automation
    - As a result, longer deployment times
    - Dated technology developed before virtualization and cloud computing - lacks responsiveness to changing requirements

Introducing Puppet

Configuration Management for systems administrators.

- **DISCOVER**
- **CONFIGURE**
- and **MANAGE** infrastructure.



	Current State	With Puppet
Productivity	50-100 nodes/admin	100s-1000s nodes/admin
Provisioning	Days/Weeks	Minutes/Hours
Configuration Drift	Ever-Present	Eliminated
Visibility	Limited, if any	All Changes
Software Releases	Weeks or months	Days, even hours

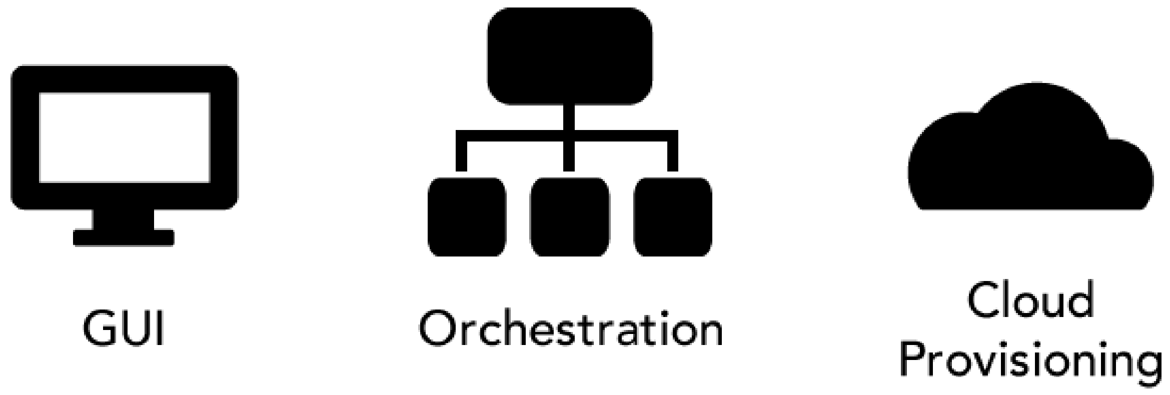
Notes:

Puppet provides a unique approach to IT automation for discovering, configuring, and managing your infrastructure. Puppet provides the following CUSTOMER BENEFITS:

- Productivity / Efficiency
    - Most IT management solutions deliver efficiencies of 20-30 nodes per sysadmin. Puppet enables 100s and even 1000s of nodes per sysadmin! Responsiveness To Business Needs
    - Using Puppet, our customers have dramatically reduced the time it takes them to deliver applications into production, from weeks to days and even hours. Eliminate Configuration Drift
    - With Puppet, your nodes (your servers, desktops, etc.) remain in the state you set for them, dramatically improving service availability, reliability, scalability, and performance.

- Visibility
    - Puppet provides rich data sets not only of infrastructure configuration but also of any changes to that infrastructure, whether under direct control of Puppet or not. You have much more visibility into the changes occurring in your infrastructure over time and their impact to service levels.
- Scalability:
    - Manage thousands of systems from a central point.
    - Reliably distribute system administrative tasks to staff.
- Consistency
    - Ensure that systems are in the intended state.
    - Eliminate inconsistency - QA/Staging/Production
    - Improve velocity of new service delivery.
- Operational efficiency
    - Spend less tracking system drift.

Puppet Enterprise



Notes:

Puppet Labs' commercial product, Puppet Enterprise, offers additional benefits in response to customer requests for the following:

- a Puppet that is easier-to-use, to start solving problems faster.
- tools to get up-and-running quickly in private cloud and public cloud environments.
- tools to make the same changes simultaneously to clusters of nodes.
- additional tools to give them visibility into the state of their infrastructure.

Three of the major capabilities in Puppet Enterprise are shown in the slide above, as follows:

1. Graphical User Interface
    - Right out-of-the-box, you can visually discover and manage resources without a command line interface (CLI) or having to build a Puppet Module.
    - CUSTOMER BENEFIT: Be productive, faster.
2. Cloud Provisioning
    - Respond quickly to business demands by creating and configuring additional capacity for VMware private clouds or Amazon public clouds.

- The same modules you built for internal, physical deployments can be re-used for private and public cloud deployments.
- CUSTOMER BENEFIT: Respond quickly to business demands while maintaining consistency across your deployment environments.

3. Orchestration
    - Simultaneously deploy critical system updates, such as security patches, across clusters of nodes with a single live management command.
    - CUSTOMER BENEFIT: Maintains the model-based integrity and scalability of Puppet while providing more direct "command-and-control".

Puppet Enterprise Stack

Simplifies installation and configuration.

- PE Stack Elements include:
    - Puppet Master and Agent
    - Puppet Module Tool
    - Puppet Enterprise Console
    - Live Management
- Automatically configured to scale, with:
    - Apache
    - Passenger
    - Rack/Rails
- All elements of the PE Stack are fully tested.
- Enterprise support is included.

Notes:

Puppet Enterprise simplifies the installation and configuration of the Puppet Stack, including:

1. Automatically configured to scale with the following supporting technologies and software (not comprehensive):
    - Open JDK
    - ActiveMQ
    - PostgreSQL
    - Ruby 1.9.3
    - Apache
    - Rack/Rails
    - Passenger
2. Puppet Specific Supporting Tools:
    - Facter
    - Hiera

- MCollective
- Puppet Module Tool

3. Puppet Enterprise Installable Roles:
    - Puppet Agent
    - Cloud Provisioner
    - Puppet Master
    - Puppet Console
    - PuppetDB

Puppet Enterprise comes with an Enterprise Service Level Agreement

Resources to Help You Succeed

Services & Support

The Forge

Training

Notes:

Puppet Labs offers a rich ecosystem to enable your success with Puppet Enterprise.

Services & Support

- Experienced team of senior Puppet practitioners provide Professional Services on-site.
- Standard (email 5x8) and Premium (phone 24x7) technical support services available.

The Forge--"This is how you can get started..."

- Hundreds of freely downloadable, customizable modules to help you get up-and-running quickly.
- Includes Puppet code for applications, resources, operating systems, and more.

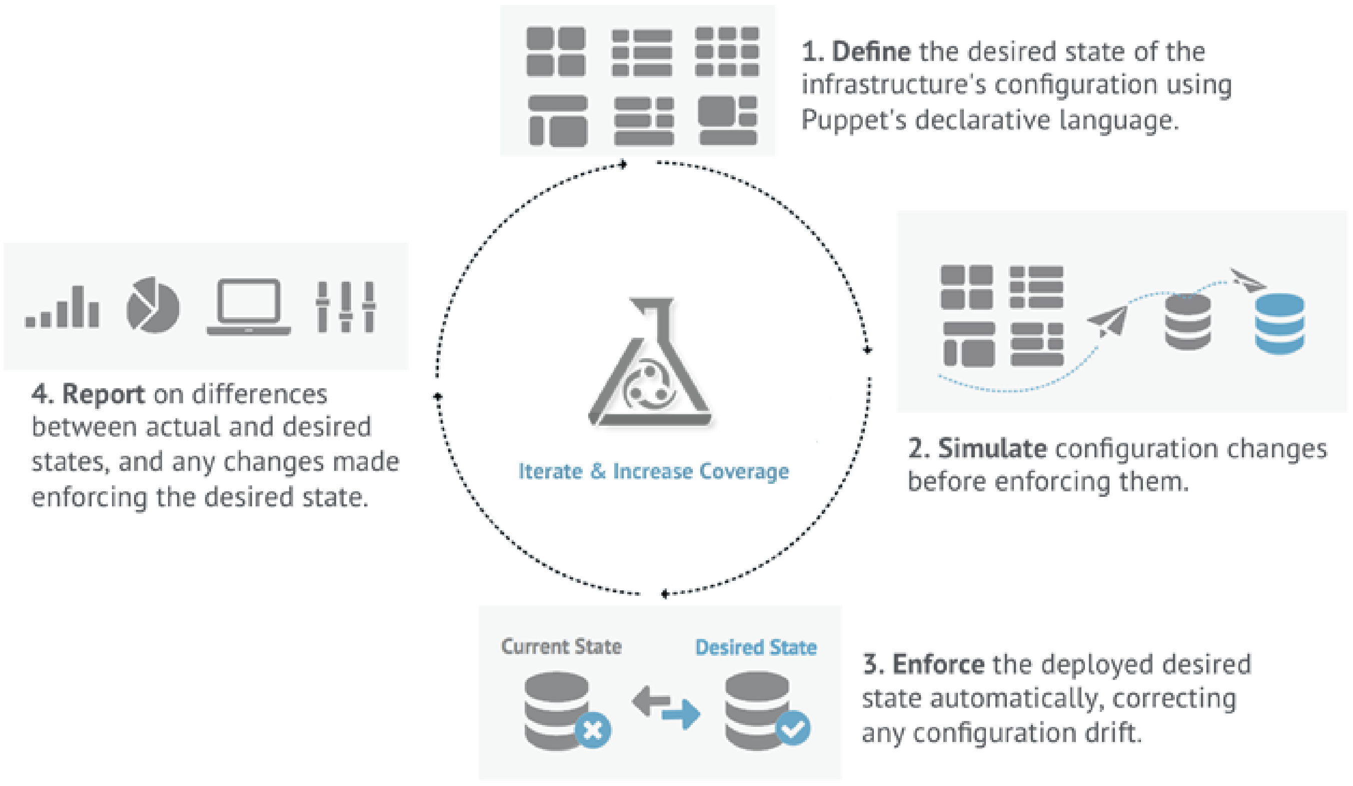
Training

- Introductory and advanced Puppet classes, both publicly available as well as customized on-site, are available to help your team come up-to-speed quickly.

Partners

- Partner products complement Puppet's own capabilities and provide you with a "whole solution" to the challenges you face.

How Puppet Works


1. Define the desired state of the infrastructure's configuration using Puppet's declarative language.
2. Simulate configuration changes before enforcing them.
3. Enforce the deployed desired state automatically, correcting any configuration drift.
Current State
Desired State
4. Report on differences between actual and desired states, and any changes made enforcing the desired state.
Iterate & Increase Coverage

Notes:

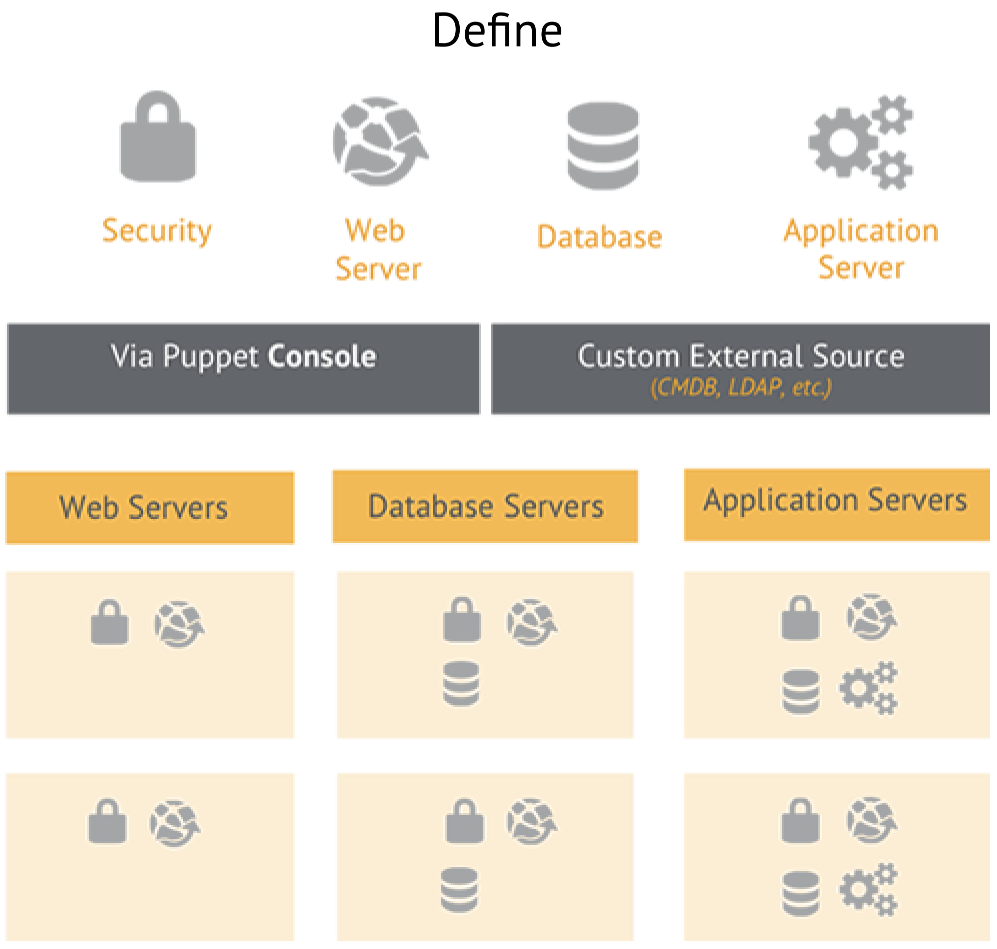
Puppet's unique, model-based approach to automating discovery, configuration, and management follows four steps to automating your infrastructure: 1. Define, 2. Simulate, 3. Enforce, and 4. Report.

1. Define your infrastructure and its desired state.
    - Use Puppet's domain specific language (DSL) to define the resources - users, services, packages, anything - you want to manage
    - You can also use one of the hundreds of pre-built, freely downloadable modules from The Forge repository as a starting point
    - You can choose to manage as few resources as a single file, or as many as all resources on the entire node - it's totally up to you.
    - CUSTOMER BENEFIT: These definitions are now both re-useable and portable across operating systems, deployment environments (physical, virtual, public cloud)
2. Simulate these resource definitions
    - The model-based approach is what allows Puppet to simulate configuration changes without impacting anything in production
    - This allows you to understand the impact of these changes on your IT environment before putting them into production

- CUSTOMER BENEFIT: The result is a much more visible and controlled change management of IT automation configuration, resulting in higher service levels

3. Enforce the desired state of your infrastructure
    - Once these definitions are in production, the Puppet Agent checks their actual state against the Puppet Master server every 30 minutes.
    - If the state of an Agent's definitions has drifted or experienced an unauthorized change, Puppet can automatically revert them back into its originally defined, desired state.
    - CUSTOMER BENEFIT: The result is the elimination of configuration drift.
4. Report on the state of your infrastructure
    - The last step is to aggregate all the changes to the desired state of resources across all nodes into a single report.
    - This gives you complete visibility into the types of changes, the rate of those changes, who is making those changes, etc.
    - CUSTOMER BENEFIT: The result is that you're able to understand how changes in your infrastructure impact changes in service levels, including availability, reliability, performance, etc.

How Puppet Works



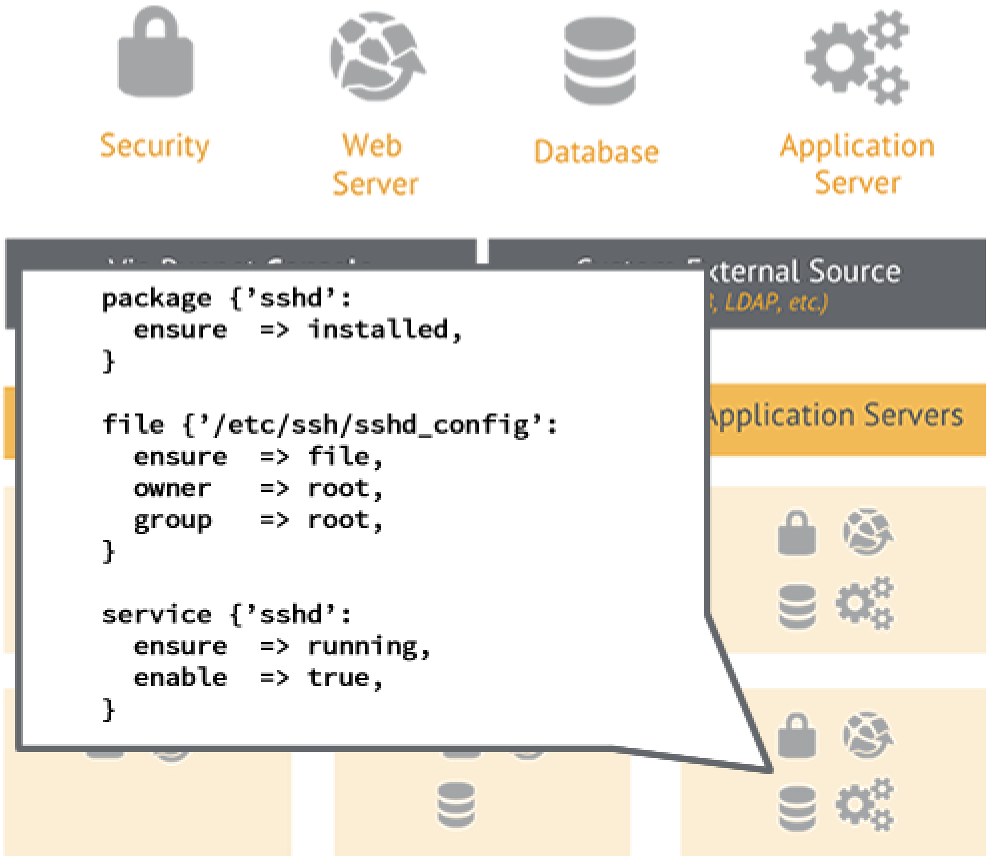
Notes:

As the source of many of the benefits customers receive from Puppet, this slide highlights a few more details about the first step of the process, the **Define** step:

- Puppet's human-readable DSL enables you to specify and manage your infrastructure with defined models of your infrastructure, not procedures.
- Complete services and applications--web servers, database servers, application services--can be built from collections of modules. Thus, Puppet allows you to compose, configure, and manage your infrastructure using re-usable "building block" components.
- Because these models are centrally managed, you can make changes once, test them, and then deploy to multiple nodes.
- This ensures that each resource (your web server, or database server, or application servers) get the right configuration every time.
- And with Puppet's resource abstraction layer the configurations are re-usable and portable across any supported operating system, as well as virtual and cloud environments.

How Puppet Works

Define



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

file {'/etc/ssh/sshd_config':
  ensure  => file,
  owner   => root,
  group   => root,
}

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

Notes:

This is an example of actual Puppet code, designed to be simple and easy to read and write.

With the above lines of code, Puppet will:

- Install the package for the SSH suite
- Install a configuration file for the SSH server
- Make sure the service is up and running

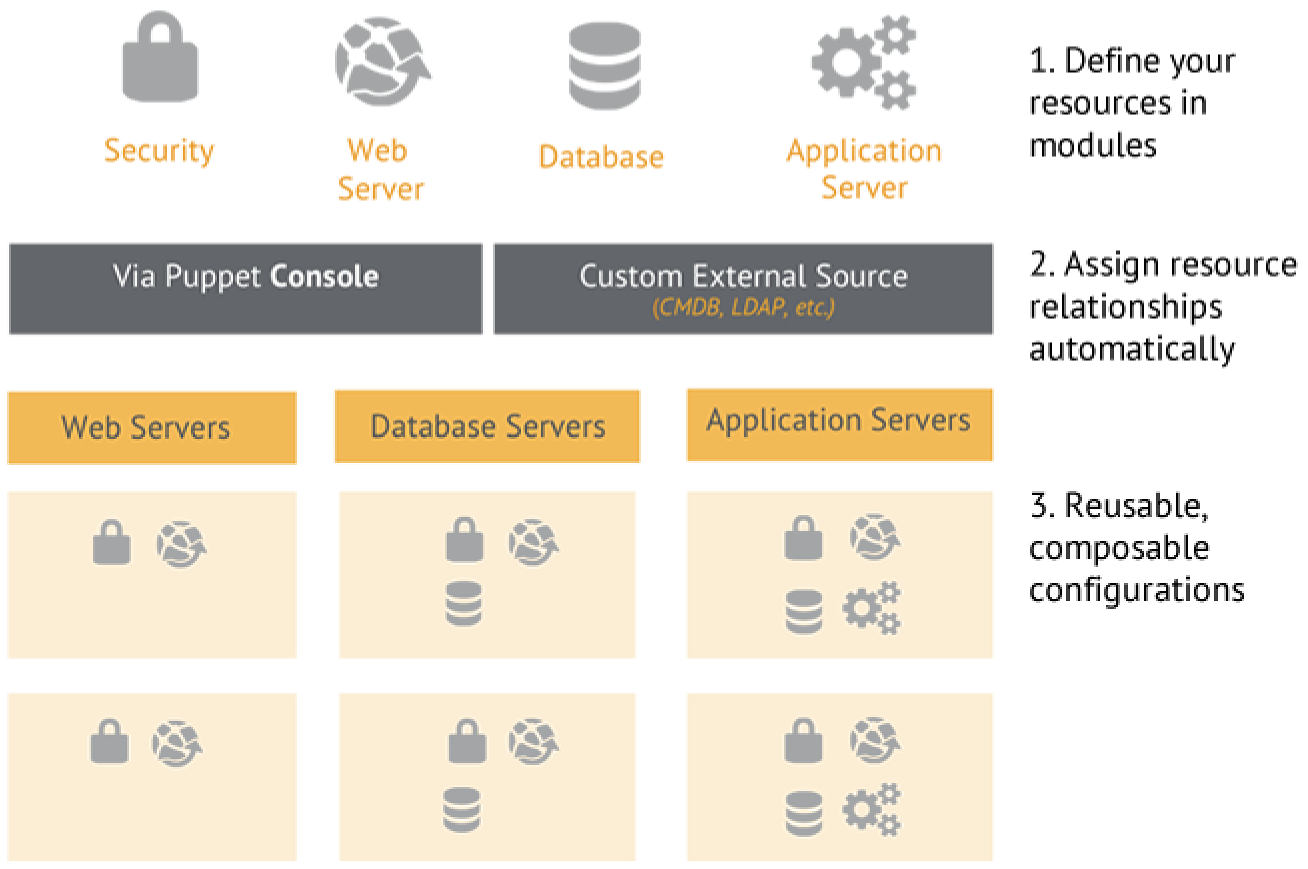
This very simple but useful model of the "SSH service" may now be ported and re-used:

- across multiple operating systems--Red Hat, Solaris, Ubuntu
- across application lifecycle phases--development, QA, production
- across multiple deployment environments--physical, virtual, private cloud, public cloud

If there's a need to change this model in the future, it can be changed in one location and deployed to all the appropriate nodes, maintaining the consistency and integrity of the model.

How Puppet Works

Composable Configurations



Notes:

Complete services and applications--web servers, database servers, application services--can be built from collections of modules.

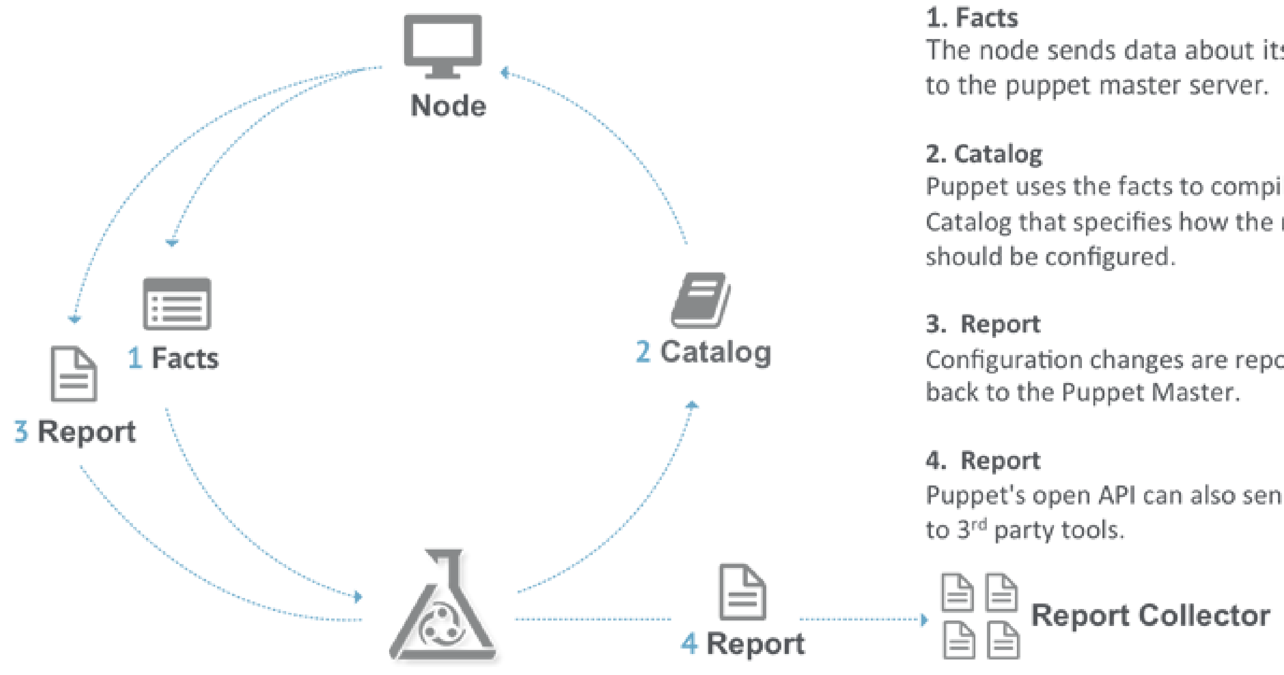
Modules are self-contained bundles of code and data.

Puppet allows system admins to compose, configure, and manage their infrastructure using re-usable "building block" components.

To help users get started, Puppet has hundreds of freely downloadable modules for resources, applications, and services at The Forge (forge.puppetlabs.com).

Data Flow

How Puppet manages data flow from Individual Nodes



1. Facts
The node sends data about its state to the puppet master server.

2. Catalog
Puppet uses the facts to compile a Catalog that specifies how the node should be configured.

3. Report
Configuration changes are reported back to the Puppet Master.

4. Report
Puppet's open API can also send data to 3rd party tools.

Notes:

A look at how definitions are used to automatically configure and manage IT infrastructure:

1. The Puppet Agent on the node tells the Puppet Master information about itself (hostname, node name, operating system, etc.).
2. The Puppet Master looks up the configuration for that node and sends a Catalog representing that intended configuration back to the node.
3. The node reports back any actions that were taken to enforce that configuration.
4. The Puppet Master server aggregates all the reports from all the nodes and provides a single overview on the state of your infrastructure.

Basic Puppet Concepts

Lesson 2: Basic Puppet Concepts

Objectives

At the end of this lesson, you will be able to:

- Identify the core components of the Puppet system management solution.
- Differentiate between declarative and imperative system configuration.
- Explain the main benefits of Puppet IT Automation Software.
- Read the basic syntax of Puppet declarations.

A Use Case

You need to manage a user, Elmo.

You care specifically about:

- his existence
- his primary group
- his home directory

Existing Utilities

Tools built into most distros that can help:

- useradd
- usermod
- groupadd
- groupmod
- mkdir
- chmod
- chgrp
- chown

Notes:

These are just some of the built-in commands that would help you solve this problem. For the purpose of this thought exercise, we're looking at built-in system tools, not dedicated user management solutions.

Command Line Concerns

- Platform idiosyncrasies:
    - Does this box have `useradd` or `adduser`? Oh, `superadduser`. Super.
- What was that flag again?
    - What is the difference between `-l` and `-L`?
    - What does `-r` mean?
        - Recurse
        - Remove read privileges
        - System user
- If I run this command again, what will it do?

Notes:

- If you're tasked with managing multiple platforms, you may have encountered tools that are named differently and whose option flags behave differently.
- Many commands behave correctly when you run them multiple times, but some don't. The proceedural nature does not give you consistent behavior without the need for extra logic.

Do It Yourself

You *could* do something like this:

```
#! /bin/sh

USER=$1; GROUP=$2; HOME=$3

if [ 0 -ne $(getent passwd $USER > /dev/null)$? ]
then useradd $USER --home $HOME --gid $GROUP -n; fi

OLDGID=`getent passwd $USER | awk -F: '{print $4}'`
OLDGROUP=`getent group $OLDGID | awk -F: '{print $1}'`
OLDHOME=`getent passwd $USER | awk -F: '{print $6}'`

if [ "$GROUP" != "$OLDGID" ] && [ "$GROUP" != "$OLDGROUP" ]
then usermod --gid $GROUP $USER; fi

if [ "$HOME" != "$OLDHOME" ]
then usermod --home $HOME $USER; fi
```

But what about...

- Robust error checking?
- Solaris and Windows support?
- Robust logging of changes?
- Readable code?

And managing users is *easy*.

How would you keep **cronjobs**, **packages**, and **services** in a consistent state across your infrastructure?

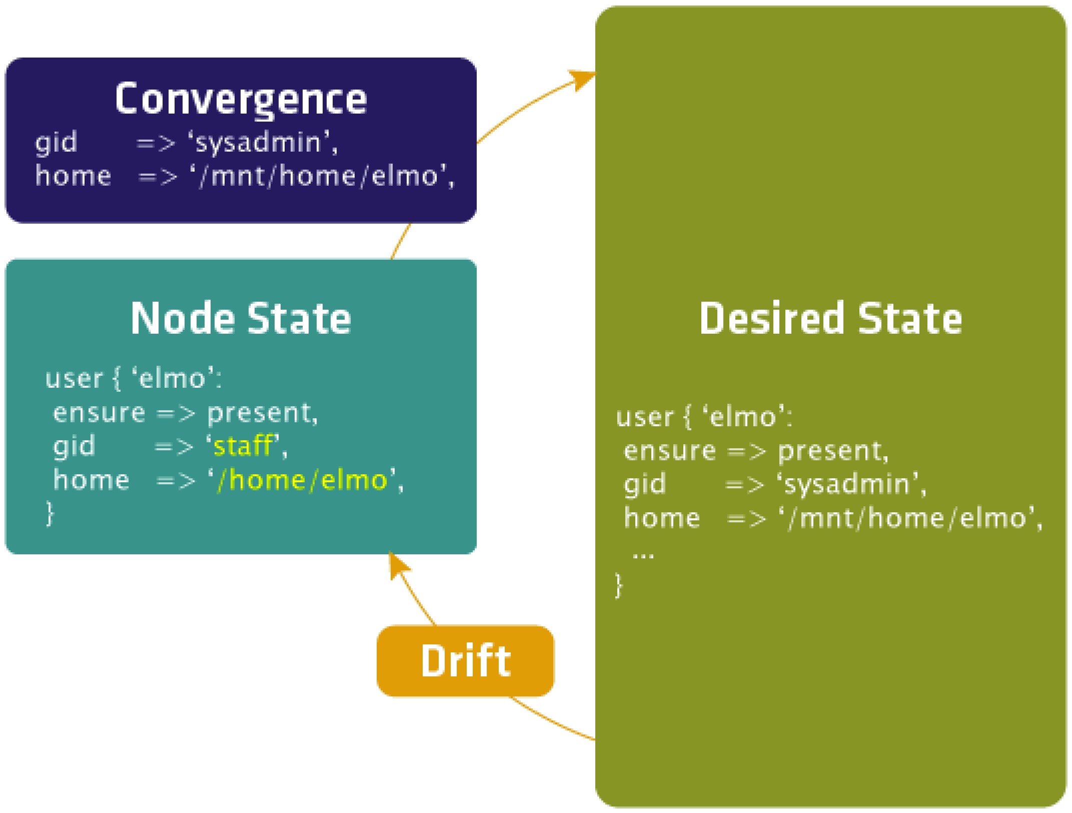
The Puppet Way

A light at the end of the tunnel:

```
user { 'elmo':
  ensure     => present,
  gid        => 'sysadmin',
  home       => '/mnt/home/elmo',
  managehome => true,
}
```

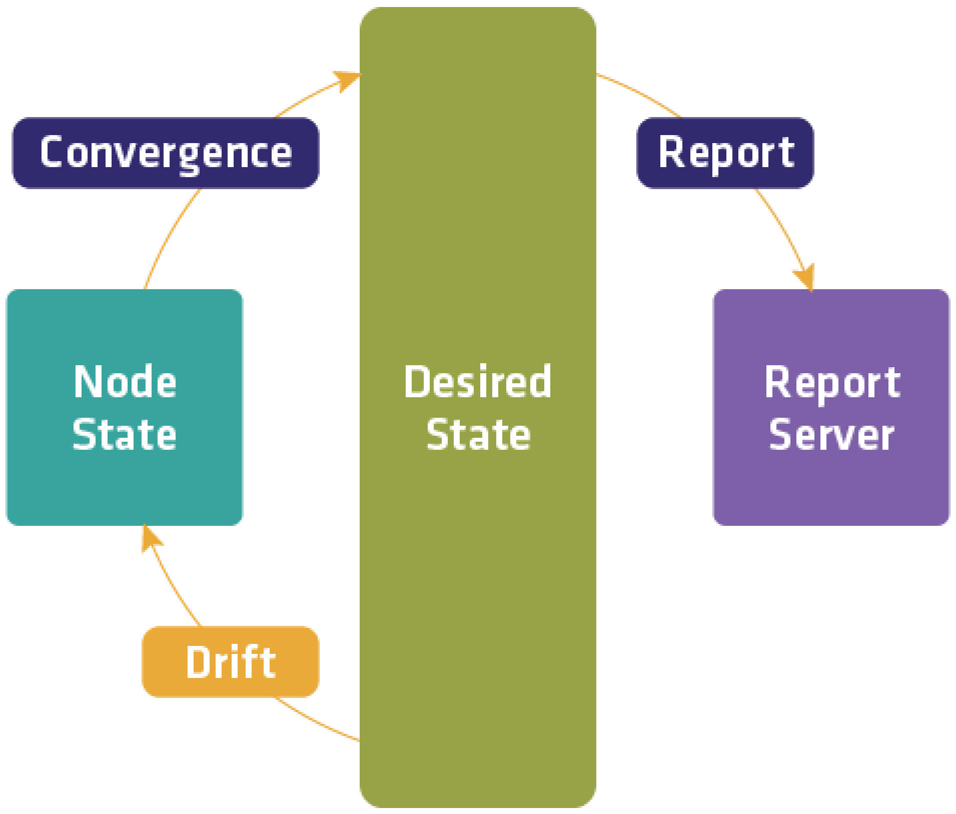
Desired State

Describe the state you want.



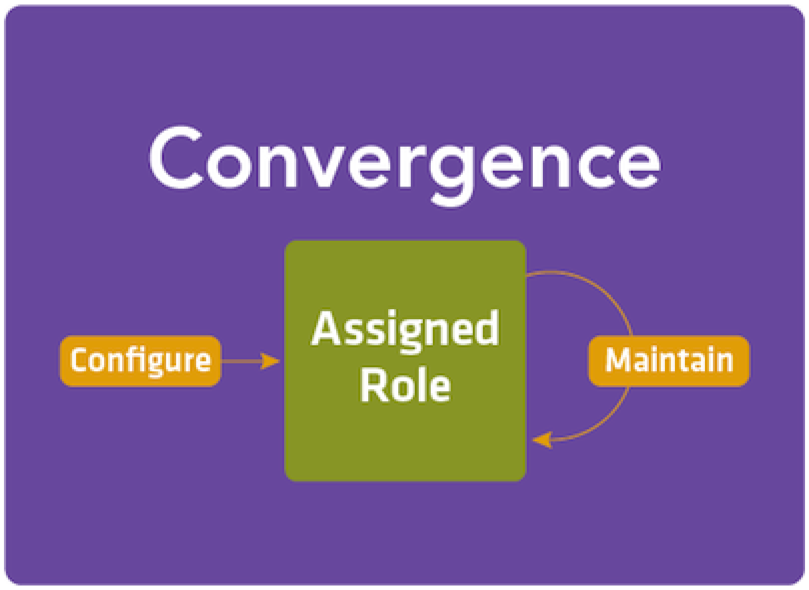
Robust Logging

Any convergence actions are reported.



Maintaining State

- You provision a node.
- Puppet configures it.
- Puppet maintains the desired state.



Infrastructure as Code

or *Executable Documentation*

```
class sysadmins {

  user { 'elmo':
    ensure     => present,
    gid        => 'sysadmin',
    home       => '/home/sysadmins/elmo',
    managehome => true,
  }

  group { 'sysadmin':
    ensure => present,
  }

}
```

- Descriptive
- Straightforward
- Transparent

Idempotency

Puppet enforces in an idempotent way.

```
# First Puppet Run
notice: /Group[sysadmin]/ensure: created
notice: /User[elmo]/ensure: created
notice: Finished catalog run in 0.08 seconds

# Second Puppet Run
notice: Finished catalog run in 0.03 seconds
```

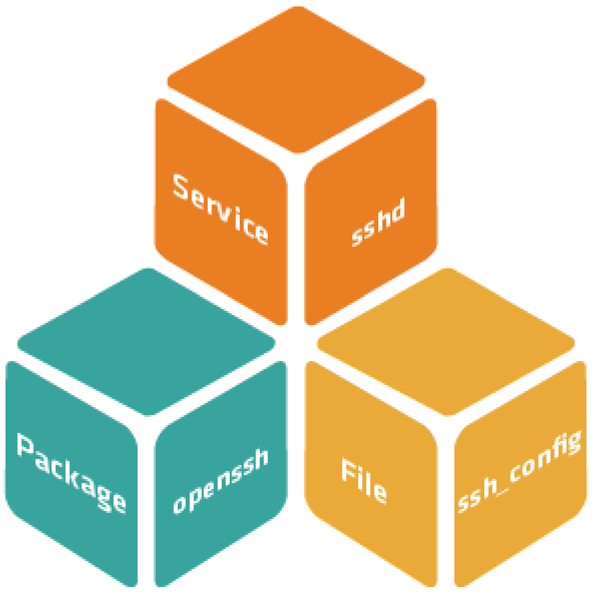
Idempotence:
The property of certain operations in mathematics or computer science in that they can be applied multiple times without further changing the result beyond the initial application.

Notes:

- Idempotent - able to be applied multiple times with the same outcome.
- Puppet resources are **idempotent**, since they describe a desired final state rather than a series of steps to follow.
- Source: Puppet Docs - http://docs.puppetlabs.com/references/glossary.html#idempotent

Puppet Resources

- Resources are building blocks.
- They can be combined to make larger components.
- Together they can model the expected state of your system.



Resource Declarations

Resources are managed in terms of *attributes*.

- Instruct Puppet to manage a **package**:

```
package { 'openssh':
  ensure => present,
}
```

- Instruct Puppet to manage a **user**:

```
user { 'elvis':
  ensure => absent,
}
```

Attributes describe the state that Puppet should converge the resource to.

User Resource

Sample Attributes

- `uid`: The user's uid number.
- `gid`: The user's primary group. Numeric or name.
- `home`: The user's home directory.
- `shell`: The user's login shell.

Want to know more?

```
$ puppet describe user

- **comment**
   A description of the user.  Generally the user's full name.

- **ensure**
   The basic state that the object should be in.  Valid values are
   `present`, `absent`, `role`.

......
......
```

Notes:

`puppet describe` takes a resource type as an argument. It returns detailed documentation on that specific resource type.

Resource Declarations

```
# Type is 'user'
# Title is 'elmo'
user { 'elmo':
  ensure => present,    # The attribute 'ensure' is set to 'present'
  gid    => 'sysadmin',  # The attribute `gid` is set to `sysadmin`
}
```

- Type and title pairs must be unique for a node.

Notes:

- Declarations start with the resource type in lowercase.
- Curly braces define the resource block.
- Separate the title from body with a colon.
- Body consists of a list of attributes and values.
- Use alphanumerics & quote strings.
- Best practice suggestions:
    - You should quote strings, even when not strictly required.
    - You should include a comma after the last attribute in a block because it reduces maintenance errors.

Declarative Modeling Language

- Model the desired state.
- Let Puppet figure out how to enforce it.

Comparison

Imperative

```
if [ 0 -ne $(getent passwd elmo > /dev/null)$? ]
then
    useradd elmo --gid sysadmin -n
fi

GID=`getent passwd elmo | awk -F: '{print $4}'`
GROUP=`getent group $GID | awk -F: '{print $1}'`

if [ "$GROUP" != "$GID" ] && [ "$GROUP" != "sysadmin" ]
then
    usermod --gid $GROUP $USER
fi
```

Declarative

```
user { 'elmo':
  ensure => present,
  gid    => 'sysadmin',
}
```

Imperative

```
if [ "`getent group sysadmin | awk -F: '{print $1}'`" == "" ]
then
    groupadd sysadmin
fi
```

Declarative

```
group { 'sysadmin':
  ensure => present,
}
```

Abstraction

Resources in Puppet are abstracted from underlying providers.

```
package { 'ssh':
  ensure => present,
  name   => $::operatingsystem ? {
    'RedHat' => 'openssh',
    'Ubuntu' => 'ssh',
  },
}
```

```
yum install mysql-server
```

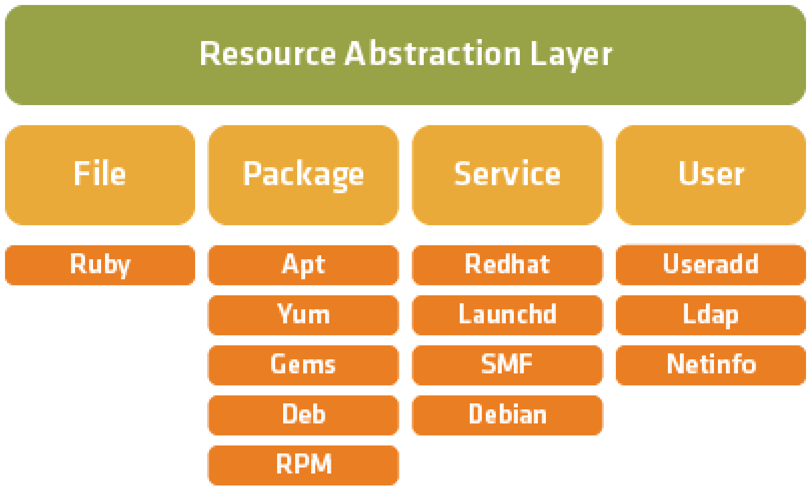
```
apt-get install mysql-server
```

Notes:

Specification in the Puppet DSL translates to *Implementation* via the provider chosen for the platform the agent is running on.

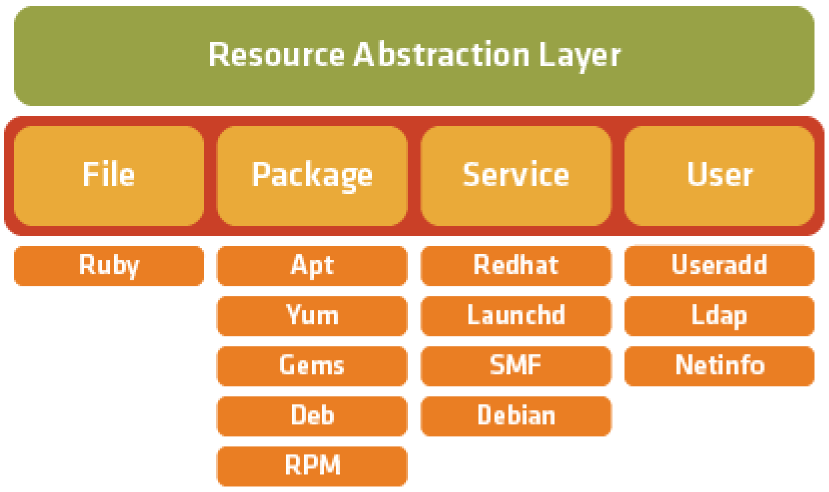
Resource Abstraction Layer

Provides a consistent model for resources across supported platforms.



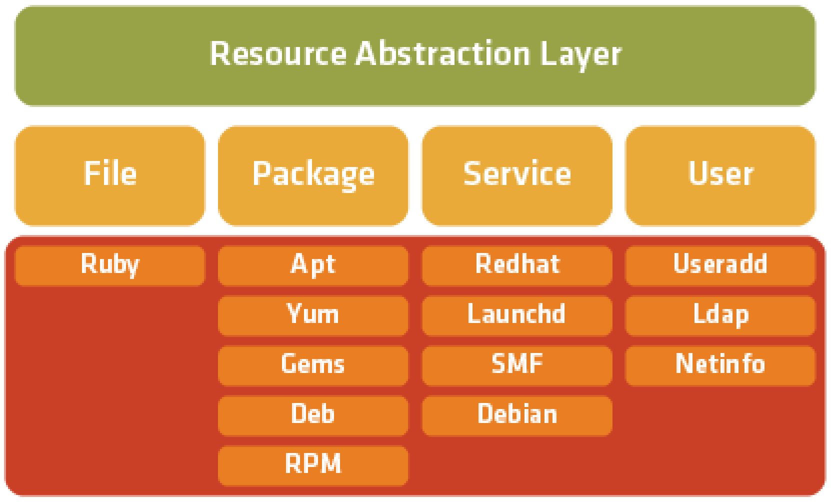
Types

Similar resources are grouped into resource types.



Providers

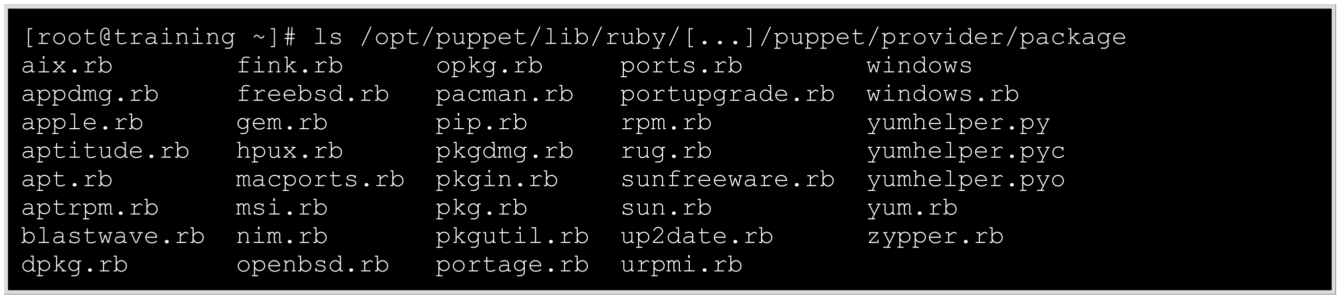
Each resource type has one or more providers.



The interface between the underlying OS and the resource types.

Many Providers

Providers for the `package` type:



```
[root@training ~]# ls /opt/puppet/lib/ruby/[...]/puppet/provider/package
aix.rb         fink.rb       opkg.rb     ports.rb        windows
appdmg.rb      freebsd.rb    pacman.rb   portupgrade.rb  windows.rb
apple.rb       gem.rb        pip.rb      rpm.rb          yumhelper.py
aptitude.rb    hpux.rb       pkgdmg.rb   rug.rb          yumhelper.pyc
apt.rb         macports.rb   pkgin.rb    sunfreeware.rb  yumhelper.pyo
aptrpm.rb      msi.rb        pkg.rb      sun.rb          yum.rb
blastwave.rb   nim.rb        pkgutil.rb  up2date.rb      zypper.rb
dpkg.rb        openbsd.rb    portage.rb  urpmi.rb
```

Notes:

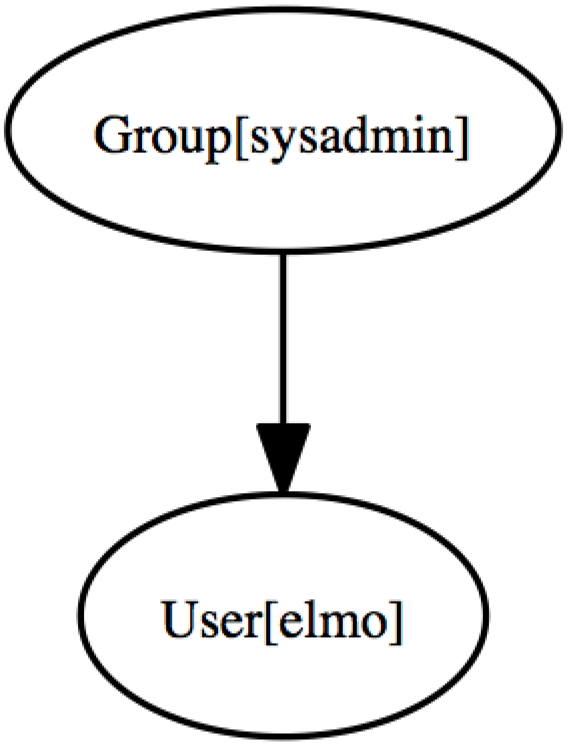
Some package types can retrieve their own package files, while others cannot. For those package formats that cannot retrieve their own package files, you can use the `source` parameter to point to the correct file or URI.

For the most current information regarding Provider packages please see:

- http://docs.puppetlabs.com/references/latest/type.html#package

Enforcement Order

Puppet enforces resources based on dependencies between them.



Notes:

Sometimes certain resources must be enforced in a certain order. For example, a *group* must exist before a *user* can be added to that group. As Puppet is enforcing the catalog, it should enforce the group before the user. It uses dependencies to determine this.

The order that we type resources into our manifests is irrelevant. Puppet will construct a dependency graph to determine the enforcement order. In a later lesson, we will learn how to describe these dependencies to Puppet.

- `Directed Acyclic Graph`:
    - A hierarchical graph with directed arrows between nodes.
    - Each node represents a resource and the arrows represent dependencies between the resources.
    - There can be no cycles or circles in the dependencies.
- Puppet starts at the top and "walks" the graph to turn it into a serial list of resources to enforce.

Classroom Environment

Lesson 3: The Classroom Environment

Objectives

At the end of this lesson, you will be able to:

- Complete the steps of setting up a Puppet Master and Agent on your local machine.
- Use the Puppet Facter tool to display system facts in the classroom setup.
- Explain the concepts behind Puppet resources and use the Puppet Resource tool.

Lab 3.1: Pre-installation

- **Objective:**
  - Assign a hostname to your agent and make that name persist across reboots.
- **Steps:**
  - Edit your systems host file (i.e. `/etc/hosts`).
  - Configure your system's hostname.
  - Ensure your system's time is synced. `ntpdate pool.ntp.org`

Lab 3.2: Installation

- **Objective:**
  - Install the Puppet Master and Agent on your virtual machine and explore some of the basic functionality of Puppet Enterprise.
- **Steps:**
  - Create an answer file for your puppet agent installation.
  - Install your Puppet Master and Agent using the answer file you created.
  - Explore some basic command line functions in Puppet as outlined in the Exercise and Lab Guide.

Notes:

This course uses Puppet Enterprise for all labs and exercises, so we are installing the Enterprise of our Software at this point. However, the principles and concepts taught in this course apply equally to Puppet Open Source, unless specifically designated as Puppet Enterprise only in the course materials.

Because we are creating a classroom environment where all students connect to a central master, we are installing only the agent in this exercise and explicitly responding "no" to the question whether to install the master. A central installation of Puppet Master for the classroom is being demo'd by the instructor.

If you wanted to install a Puppet Master server, you would instead answer "yes" to the same question and would be presented with additional options to configure your master as well as your agent while executing the installation script.

If you are using Puppet Open Source, see http://docs.puppetlabs.com/guides/installation.html for installation documentation.

For further documentation on installing Puppet Enterprise, see http://docs.puppetlabs.com/pe/latest/install_basic.html and http://docs.puppetlabs.com/guides/troubleshooting.html.

Facter

- Puppet uses `facter` to gather information about the host system.
- Executing the `facter` command returns a list of key value pairs.

```
[root@training ~]# facter
architecture => x86_64
domain => puppetlabs.com
facterversion => 1.5.2
fqdn => training.puppetlabs.lan
hardwaremodel => x86_64
hostname => aku
interfaces => eth0
ipaddress => 172.16.10.1
kernel => Linux
operatingsystem => Ubuntu
...
```

- The returned key value pairs are `facts`.

Notes:

- Facter is Puppet's system inventory tool. Facter reads facts about a node (such as its hostname, IP address, operating system, etc.) and makes them available to Puppet.
- Facter includes a large number of built-in facts. You can view their names and values for the local system by running `facter` at the command line.
- In agent/master Puppet arrangements, agent nodes send their facts to the master, and the master compiles the catalog using these facts.

Exercise 3.3: Facter

- **Objective:**
  - Become familiar with the use of `facter`.
- **Steps:**
  - Execute `facter ipaddress`.
    - What is returned?
  - Execute `facter operatingsystem`.
    - What is returned?
  - Execute `facter`.
    - What is returned?

Puppet Resource

- A command line tool for inspecting puppet resources on the system.
- It interacts directly with the Resource Abstraction Layer (RAL).

Puppet Resource Query

Executing `puppet resource` and providing a *resource type* and *title* returns the state of a resource.

```
[root@training ~]# puppet resource user elvis

user { 'elvis':
  ensure => absent,
}
```

Puppet Resource Query

Executing `puppet resource` and providing a *resource type* queries all known instances of that resource on the system.

```
[root@training ~]# puppet resource user

....
user { 'vcsa':
    ensure  => present,
    uid     => '69',
    gid     => '69',
    shell   => '/sbin/nologin',
    comment => 'virtual console memory owner',
    home    => '/dev',
}
user { 'willywonka':
    ensure => present,
    uid    => '1006',
    gid    => '1008',
    shell  => '/bin/bash',
    home   => '/home/willywonka',
}
```

Exercise 3.4: Puppet Resource

- **Objective:**
  - Use `puppet resource` to inspect user accounts.

- **Steps:**
  - Create your user account manually with useradd (ex. `useradd elmo`).
  - Using `puppet resource`, inspect your user account on your system.
  - Add a password to your account and see how it affects the output.

Puppet Roles

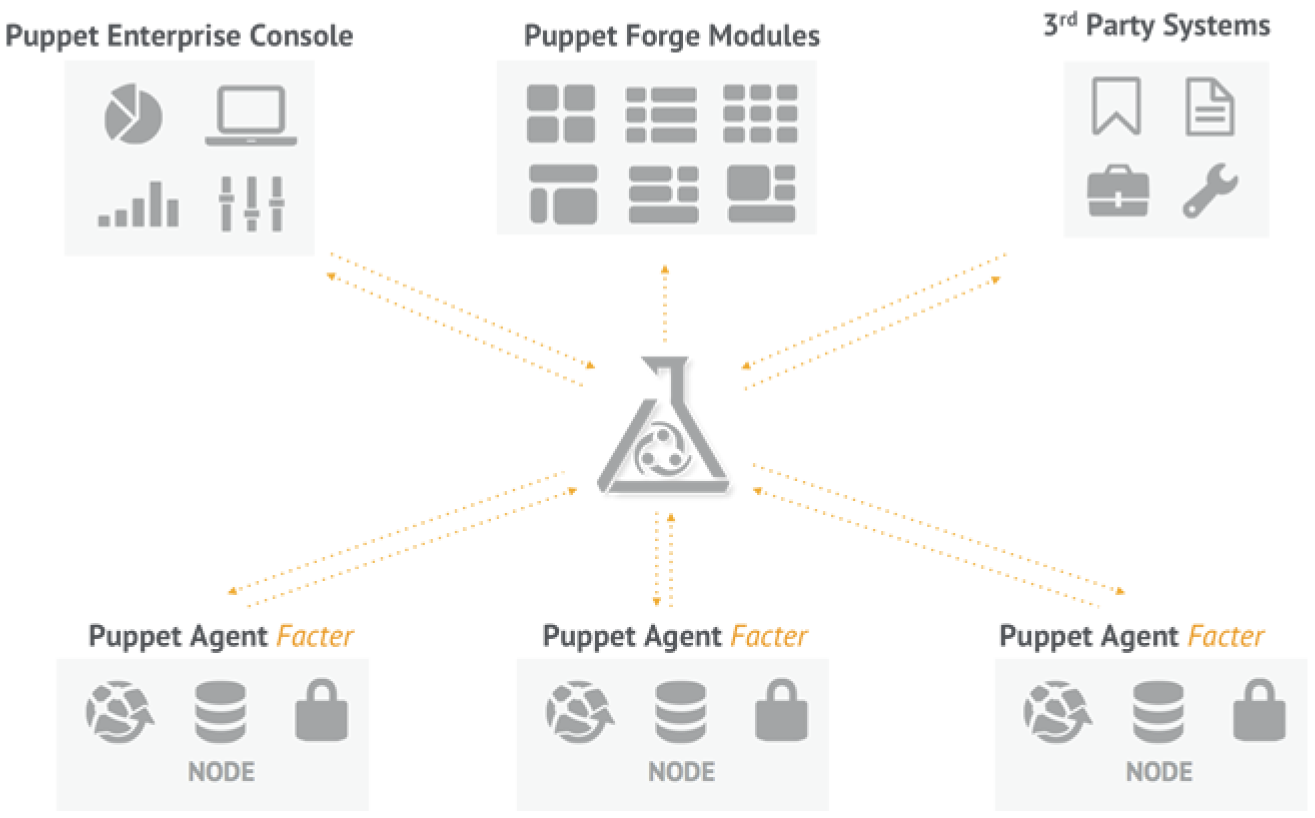
Lesson 4: Puppet Roles

Objectives

At the end of this lesson, you will be able to:

- Describe the roles of the agent and the master.
- Identify how the Puppet Catalog fits into Puppet workflow.
- Recognize standard module paths.
- Describe node classification.
- Describe the SSL relationship between agent and master.

Puppet Configuration Management



The Agent Daemon

`puppet agent` runs on all managed nodes.

- It is responsible for:
    - Initiating a secure and authenticated connection to the Puppet Master.
    - Sending information about its current state.
    - Enforcing a retrieved **catalog**.

Notes:

The **catalog** is an object that represents the desired end-state of a node.

Example Configuration

`/etc/puppetlabs/puppet/puppet.conf`

```
[main]
    vardir = /var/opt/lib/pe-puppet
    logdir = /var/log/pe-puppet
    rundir = /var/run/pe-puppet
[agent]
    certname = webdb1
    server = puppet
```

Notes:

Other configuration variables of interest:

- `vardir`: location where Puppet stores dynamically growing information.
- `rundir`: location where Puppet PID files are stored.
- `ssldir`: location where SSL certificates are stored.
- `ca_server`: the server to use for certificate authority requests.
- `certname`: the certificate name to use when communicating with the master.
- `server`: the host name of the puppetmaster.

Useful Command Line Arguments

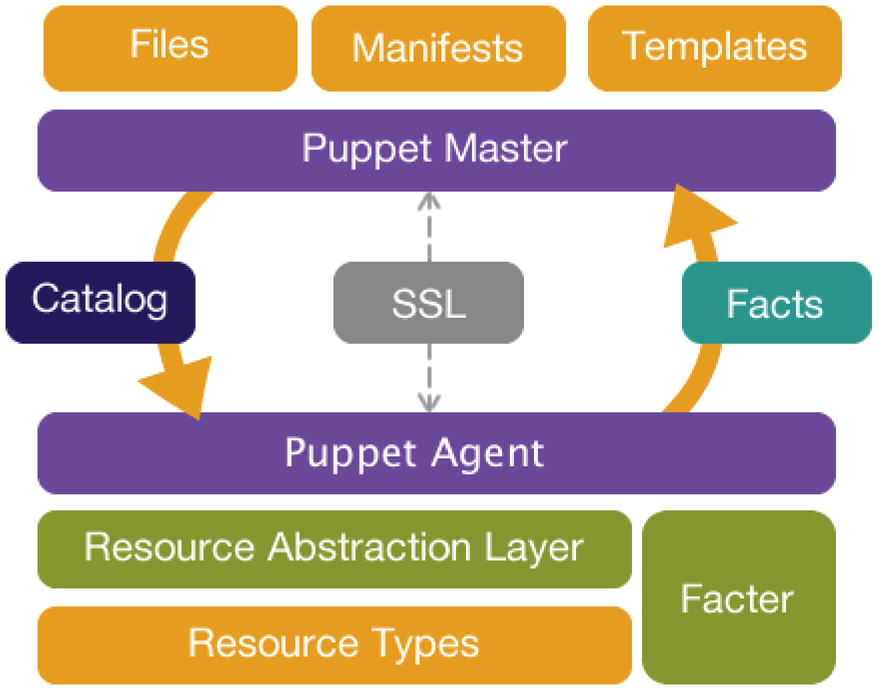
- `--test`
  - `--no-daemonize`
  - `--verbose`
  - `--onetime`
- `--noop`
- `--debug`
- `--tags`
- `--environment <env>`
- `--configprint`
- `--genconfig`

The Master Daemon

`puppet master` runs on the central server.

- It is responsible for:
    - authenticating agent connections.
    - signing certificates.
    - compiling manifests into a catalog.
    - serving that catalog to the agent.
    - serving files.

Agent/Master Architecture



Notes:

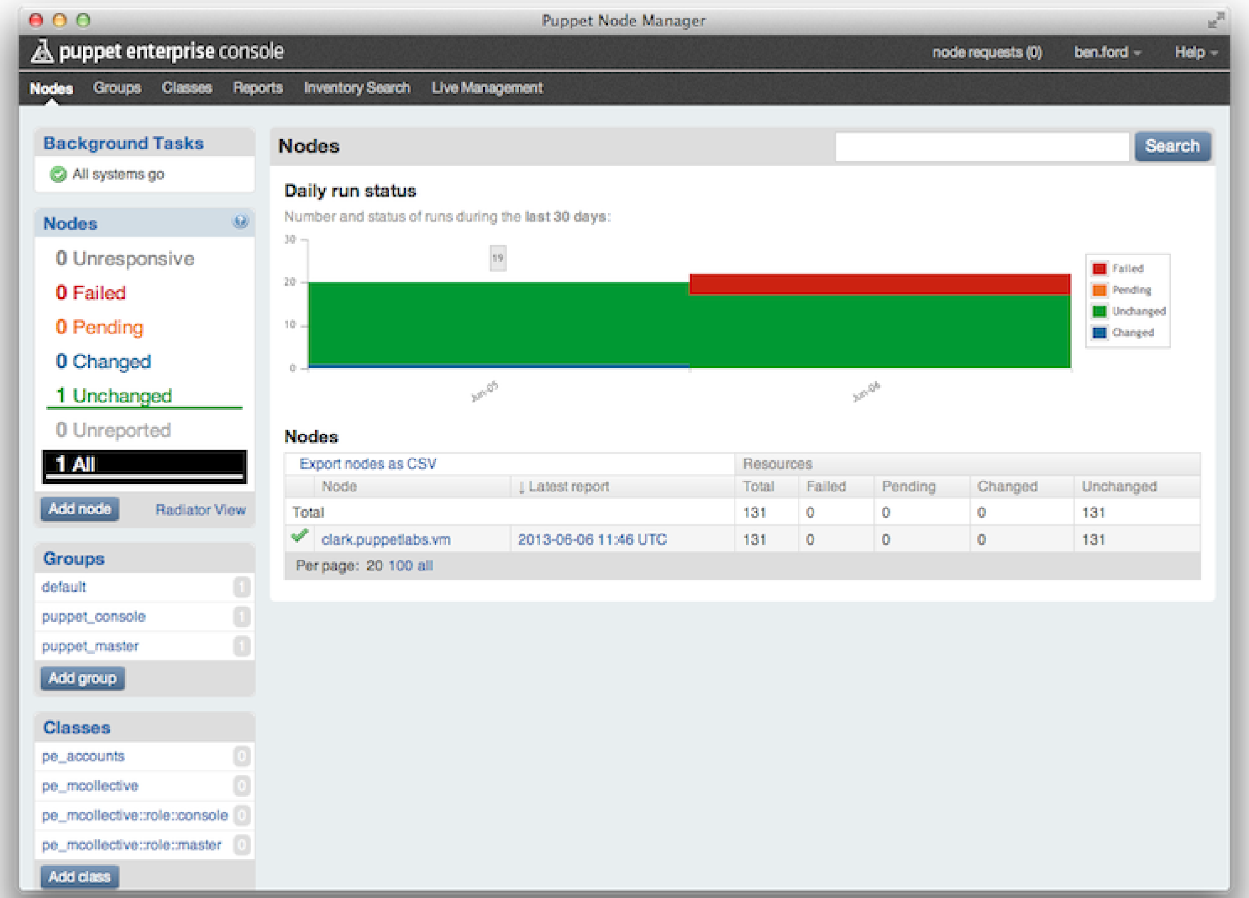
The only information transmitted between the Master and Agent is the *Facts* submitted by the Agent and the *Catalog* returned by the Master. This means that the Master has no inherent knowledge of any other state on the Agent and the Agent sees none of the Puppet source code used to generate the catalog.

Puppet Enterprise Console

Provides a graphical interface to your Puppet infrastructure.

- It is responsible for:
    - presenting an overview of your systems.
    - providing detailed information about each node.
    - collating and displaying statistics.
    - providing an interface for node classification.
    - enabling report browsing and viewing.

Infrastructure Overview



Background Tasks

All systems go

Nodes

0 Unresponsive
0 Failed
0 Pending
0 Changed
1 Unchanged
0 Unreported
1 All

Groups

default	1
puppet_console	1
puppet_master	1

Classes

pe_accounts	0
pe_mcollective	0
pe_mcollective::role::console	0
pe_mcollective::role::master	0

Nodes

Daily run status

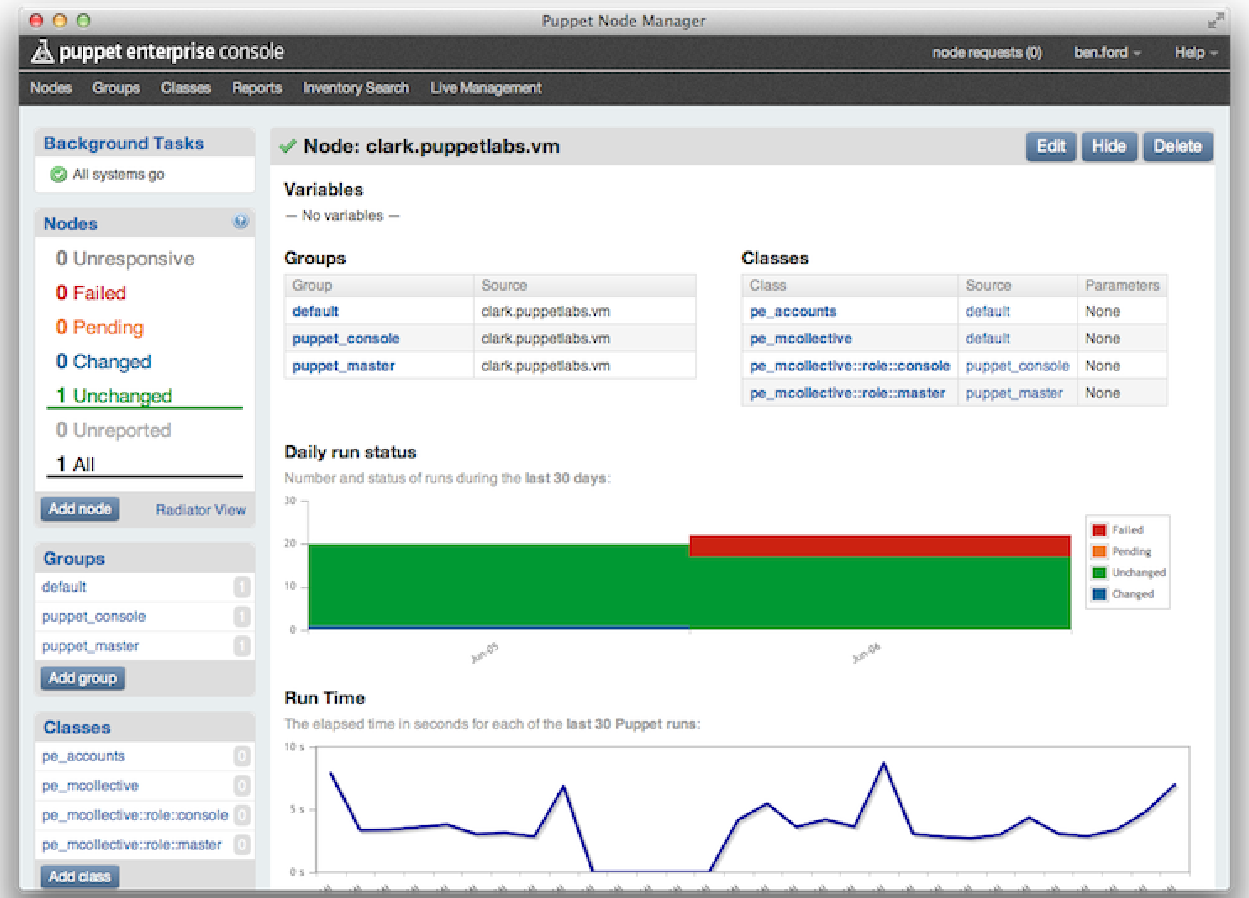
Number and status of runs during the **last 30 days**:

Nodes

Export nodes as CSV		Resources				
Node	↓ Latest report	Total	Failed	Pending	Changed	Unchanged
Total		131	0	0	0	131
✓ clark.puppetlabs.vm	2013-06-06 11:46 UTC	131	0	0	0	131

Per page: 20 100 all

Node Details and Statistics



Puppet Node Manager

puppet enterprise console — node requests (0) — ben.ford — Help

Nodes | Groups | Classes | Reports | Inventory Search | Live Management

Background Tasks

All systems go

Nodes

- 0 Unresponsive
- 0 Failed
- 0 Pending
- 0 Changed
- 1 Unchanged
- 0 Unreported
- 1 All

Add node — Radiator View

Groups

default	1
puppet_console	1
puppet_master	1

Add group

Classes

pe_accounts	0
pe_mcollective	0
pe_mcollective::role::console	0
pe_mcollective::role::master	0

Add class

Node: clark.puppetlabs.vm — Edit — Hide — Delete

Variables

— No variables —

Groups

Group	Source
default	clark.puppetlabs.vm
puppet_console	clark.puppetlabs.vm
puppet_master	clark.puppetlabs.vm

Classes

Class	Source	Parameters
pe_accounts	default	None
pe_mcollective	default	None
pe_mcollective::role::console	puppet_console	None
pe_mcollective::role::master	puppet_master	None

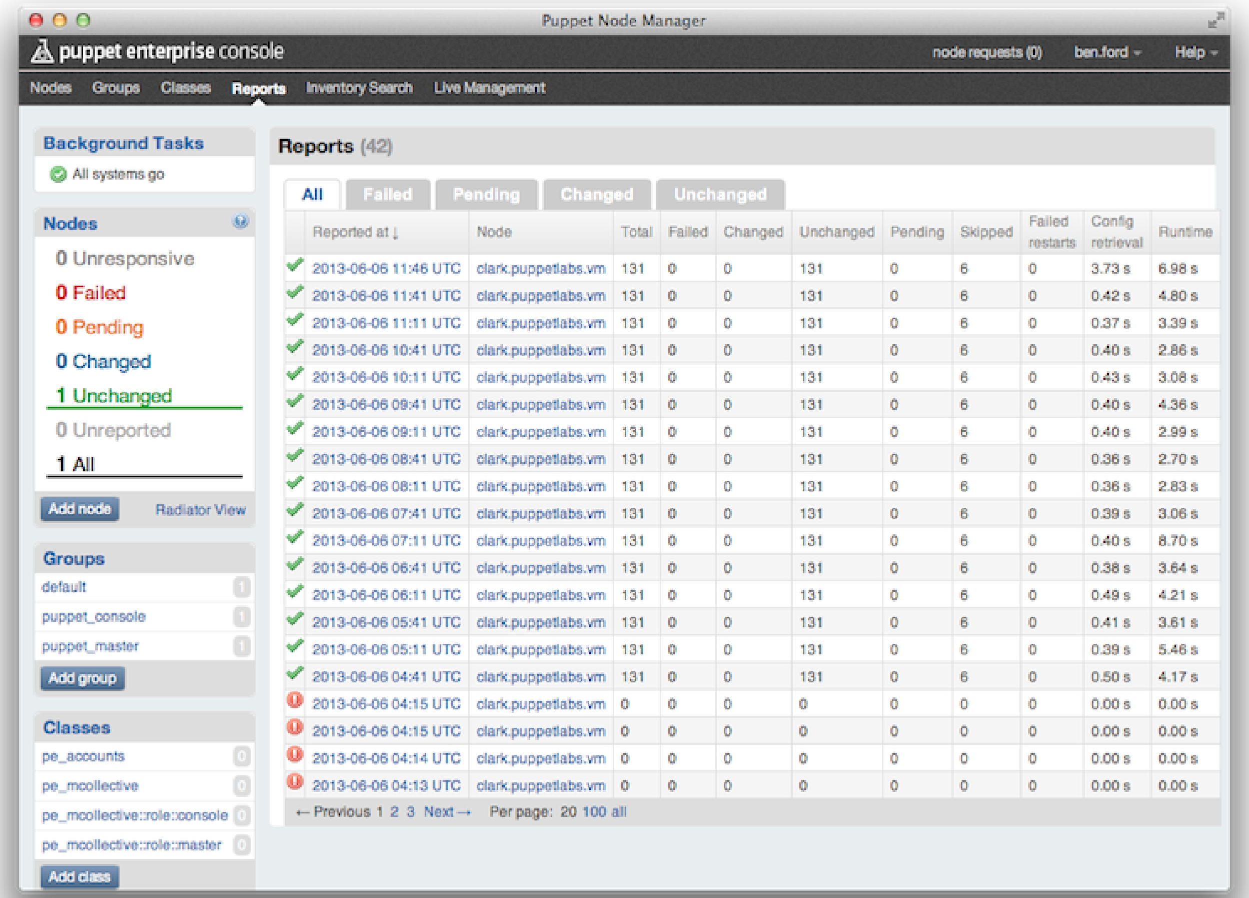
Daily run status

Number and status of runs during the **last 30 days**:

Run Time

The elapsed time in seconds for each of the **last 30 Puppet runs**:

Browsing Latest Reports

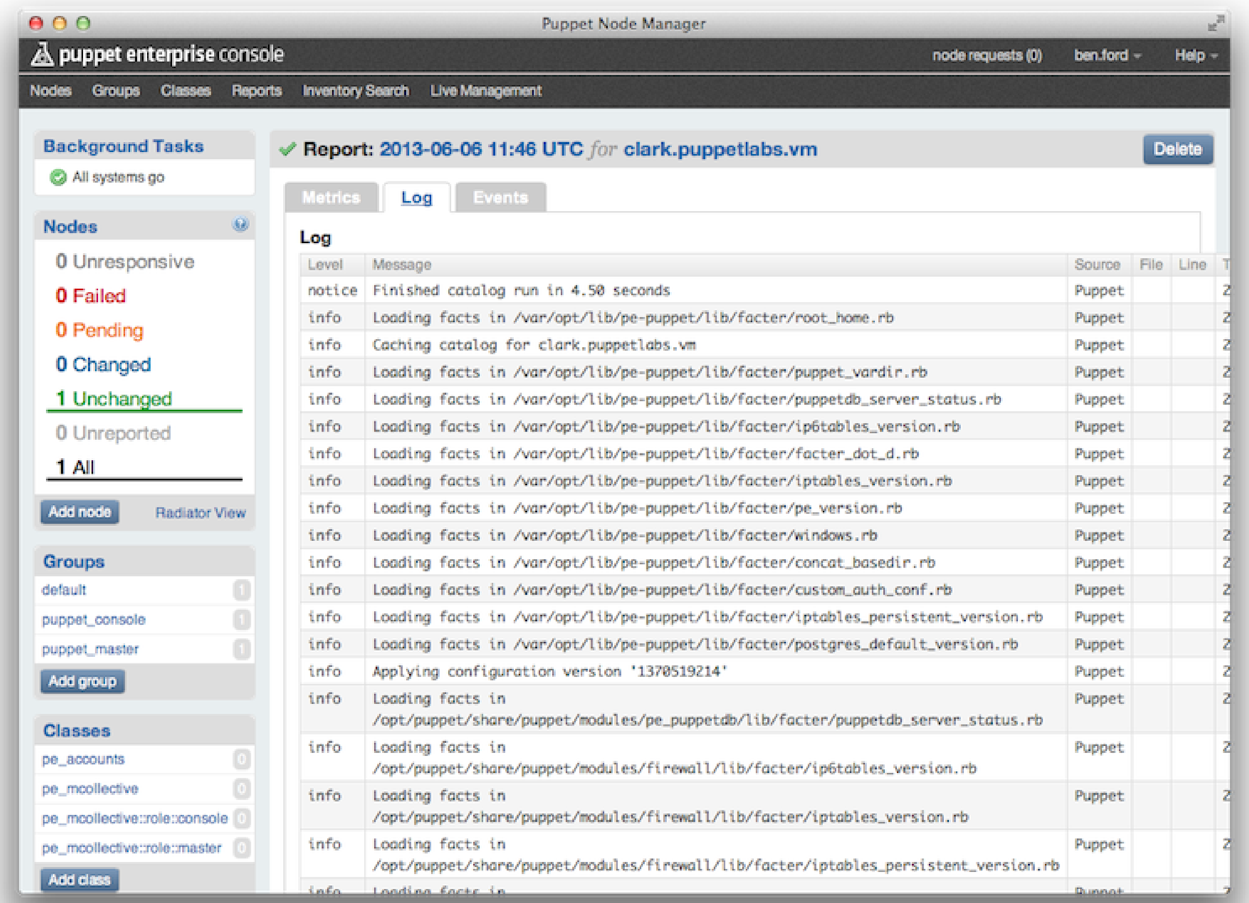


Reports (42)

	Reported at ↓	Node	Total	Failed	Changed	Unchanged	Pending	Skipped	Failed restarts	Config retrieval	Runtime
✓	2013-06-06 11:46 UTC	clark.puppetlabs.vm	131	0	0	131	0	6	0	3.73 s	6.98 s
✓	2013-06-06 11:41 UTC	clark.puppetlabs.vm	131	0	0	131	0	6	0	0.42 s	4.80 s
✓	2013-06-06 11:11 UTC	clark.puppetlabs.vm	131	0	0	131	0	6	0	0.37 s	3.39 s
✓	2013-06-06 10:41 UTC	clark.puppetlabs.vm	131	0	0	131	0	6	0	0.40 s	2.86 s
✓	2013-06-06 10:11 UTC	clark.puppetlabs.vm	131	0	0	131	0	6	0	0.43 s	3.08 s
✓	2013-06-06 09:41 UTC	clark.puppetlabs.vm	131	0	0	131	0	6	0	0.40 s	4.36 s
✓	2013-06-06 09:11 UTC	clark.puppetlabs.vm	131	0	0	131	0	6	0	0.40 s	2.99 s
✓	2013-06-06 08:41 UTC	clark.puppetlabs.vm	131	0	0	131	0	6	0	0.36 s	2.70 s
✓	2013-06-06 08:11 UTC	clark.puppetlabs.vm	131	0	0	131	0	6	0	0.36 s	2.83 s
✓	2013-06-06 07:41 UTC	clark.puppetlabs.vm	131	0	0	131	0	6	0	0.39 s	3.06 s
✓	2013-06-06 07:11 UTC	clark.puppetlabs.vm	131	0	0	131	0	6	0	0.40 s	8.70 s
✓	2013-06-06 06:41 UTC	clark.puppetlabs.vm	131	0	0	131	0	6	0	0.38 s	3.64 s
✓	2013-06-06 06:11 UTC	clark.puppetlabs.vm	131	0	0	131	0	6	0	0.49 s	4.21 s
✓	2013-06-06 05:41 UTC	clark.puppetlabs.vm	131	0	0	131	0	6	0	0.41 s	3.61 s
✓	2013-06-06 05:11 UTC	clark.puppetlabs.vm	131	0	0	131	0	6	0	0.39 s	5.46 s
✓	2013-06-06 04:41 UTC	clark.puppetlabs.vm	131	0	0	131	0	6	0	0.50 s	4.17 s
!	2013-06-06 04:15 UTC	clark.puppetlabs.vm	0	0	0	0	0	0	0	0.00 s	0.00 s
!	2013-06-06 04:15 UTC	clark.puppetlabs.vm	0	0	0	0	0	0	0	0.00 s	0.00 s
!	2013-06-06 04:14 UTC	clark.puppetlabs.vm	0	0	0	0	0	0	0	0.00 s	0.00 s
!	2013-06-06 04:13 UTC	clark.puppetlabs.vm	0	0	0	0	0	0	0	0.00 s	0.00 s

← Previous 1 2 3 Next → Per page: 20 100 all

Viewing a Report



Background Tasks

All systems go

Nodes

0 Unresponsive
0 Failed
0 Pending
0 Changed
1 Unchanged
0 Unreported
1 All

Groups

default	1
puppet_console	1
puppet_master	1

Classes

pe_accounts	0
pe_mcollective	0
pe_mcollective::role::console	0
pe_mcollective::role::master	0

Report: 2013-06-06 11:46 UTC *for* clark.puppetlabs.vm

Log

Level	Message	Source	File	Line	T
notice	Finished catalog run in 4.50 seconds	Puppet			2
info	Loading facts in /var/opt/lib/pe-puppet/lib/facter/root_home.rb	Puppet			2
info	Caching catalog for clark.puppetlabs.vm	Puppet			2
info	Loading facts in /var/opt/lib/pe-puppet/lib/facter/puppet_vardir.rb	Puppet			2
info	Loading facts in /var/opt/lib/pe-puppet/lib/facter/puppetdb_server_status.rb	Puppet			2
info	Loading facts in /var/opt/lib/pe-puppet/lib/facter/ip6tables_version.rb	Puppet			2
info	Loading facts in /var/opt/lib/pe-puppet/lib/facter/facter_dot_d.rb	Puppet			2
info	Loading facts in /var/opt/lib/pe-puppet/lib/facter/iptables_version.rb	Puppet			2
info	Loading facts in /var/opt/lib/pe-puppet/lib/facter/pe_version.rb	Puppet			2
info	Loading facts in /var/opt/lib/pe-puppet/lib/facter/windows.rb	Puppet			2
info	Loading facts in /var/opt/lib/pe-puppet/lib/facter/concat_basedir.rb	Puppet			2
info	Loading facts in /var/opt/lib/pe-puppet/lib/facter/custom_auth_conf.rb	Puppet			2
info	Loading facts in /var/opt/lib/pe-puppet/lib/facter/iptables_persistent_version.rb	Puppet			2
info	Loading facts in /var/opt/lib/pe-puppet/lib/facter/postgres_default_version.rb	Puppet			2
info	Applying configuration version '1370519214'	Puppet			2
info	Loading facts in /opt/puppet/share/puppet/modules/pe_puppetdb/lib/facter/puppetdb_server_status.rb	Puppet			2
info	Loading facts in /opt/puppet/share/puppet/modules/firewall/lib/facter/ip6tables_version.rb	Puppet			2
info	Loading facts in /opt/puppet/share/puppet/modules/firewall/lib/facter/iptables_version.rb	Puppet			2
info	Loading facts in /opt/puppet/share/puppet/modules/firewall/lib/facter/iptables_persistent_version.rb	Puppet			2
info	Loading facts in	Puppet			

Demo

Viewing report details in Console

Certificate Management

- The Puppet Master serves as a certificate authority for all connected agents.
- All agents must have valid signed certificates to request a catalog.
- The `puppet cert` command allows you to manage client and server certificates.

Puppet Agent Bootstrap

- When a puppet agent runs for the first time, it:
    - generates a new certificate.
    - sends a CSR to the server to be signed.
    - checks for a signed cert every two minutes (by default).

Unless autosigning is enabled, each new certificate must be signed explicitly.

Notes:

Autosigning is not considered to be best practice as there is reduced visibility into which agents exist on your network. It should only be enabled when the network layer is fully trusted and there is no concern of agent impersonation.

Working With Certificates

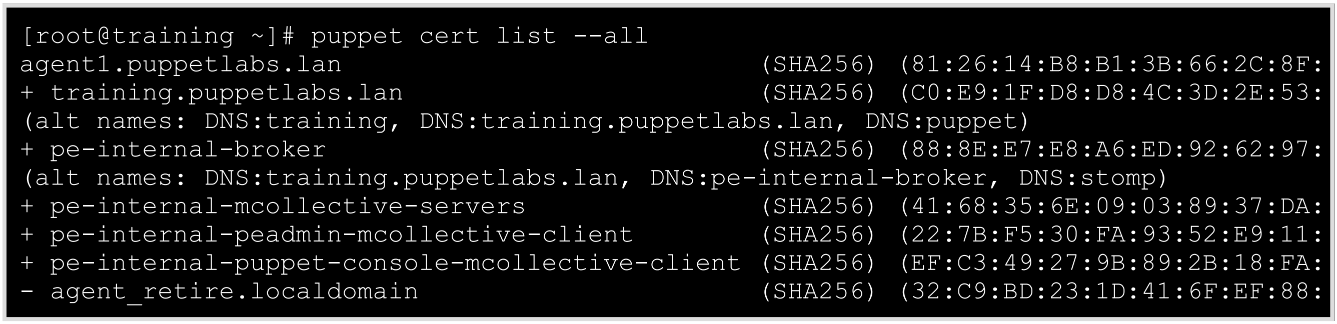
List Outstanding Certificates

```
[root@training ~]# puppet cert list
puppetagent.localdomain
```

- Certificates waiting to be signed.
- The PE installer signs the certificates it generates.

Working With Certificates

List All Certificates



```
[root@training ~]# puppet cert list --all
agent1.puppetlabs.lan                            (SHA256) (81:26:14:B8:B1:3B:66:2C:8F:
+ training.puppetlabs.lan                        (SHA256) (C0:E9:1F:D8:D8:4C:3D:2E:53:
(alt names: DNS:training, DNS:training.puppetlabs.lan, DNS:puppet)
+ pe-internal-broker                             (SHA256) (88:8E:E7:E8:A6:ED:92:62:97:
(alt names: DNS:training.puppetlabs.lan, DNS:pe-internal-broker, DNS:stomp)
+ pe-internal-mcollective-servers                (SHA256) (41:68:35:6E:09:03:89:37:DA:
+ pe-internal-peadmin-mcollective-client         (SHA256) (22:7B:F5:30:FA:93:52:E9:11:
+ pe-internal-puppet-console-mcollective-client (SHA256) (EF:C3:49:27:9B:89:2B:18:FA:
- agent_retire.localdomain                       (SHA256) (32:C9:BD:23:1D:41:6F:EF:88:
```

- '+' indicates an active & signed certificate
- '-' indicates a revoked certificate
- '' (empty) indicates a pending certificate

Working With Certificates

Sign a certificate

```
[root@training ~]# puppet cert sign agent1.puppetlabs.lan
notice: Signed certificate request for agent1.puppetlabs.lan
notice: Removing file Puppet::SSL::CertificateRequest agent1.puppetlabs.lan ...
```

Revoke a certificate

```
[root@training ~]# puppet cert revoke agent_retire.puppetlabs.lan
notice: Revoked certificate with serial 8
```

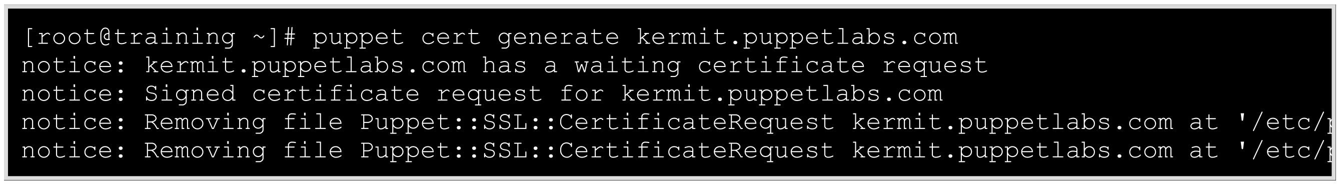
Remove a certificate

```
[root@training ~]# puppet cert clean agent_retire.puppetlabs.lan
notice: Revoked certificate with serial 8
notice: Removing file Puppet::SSL::Certificate agent_retire.puppetlabs.lan ...
notice: Removing file Puppet::SSL::Certificate agent_retire.puppetlabs.lan ...
notice: Removing file Puppet::SSL::Key agent_retire.puppetlabs.lan at '/etc...
```

(output trimmed for slide)

Working With Certificates

Pre-create certificates on the master.



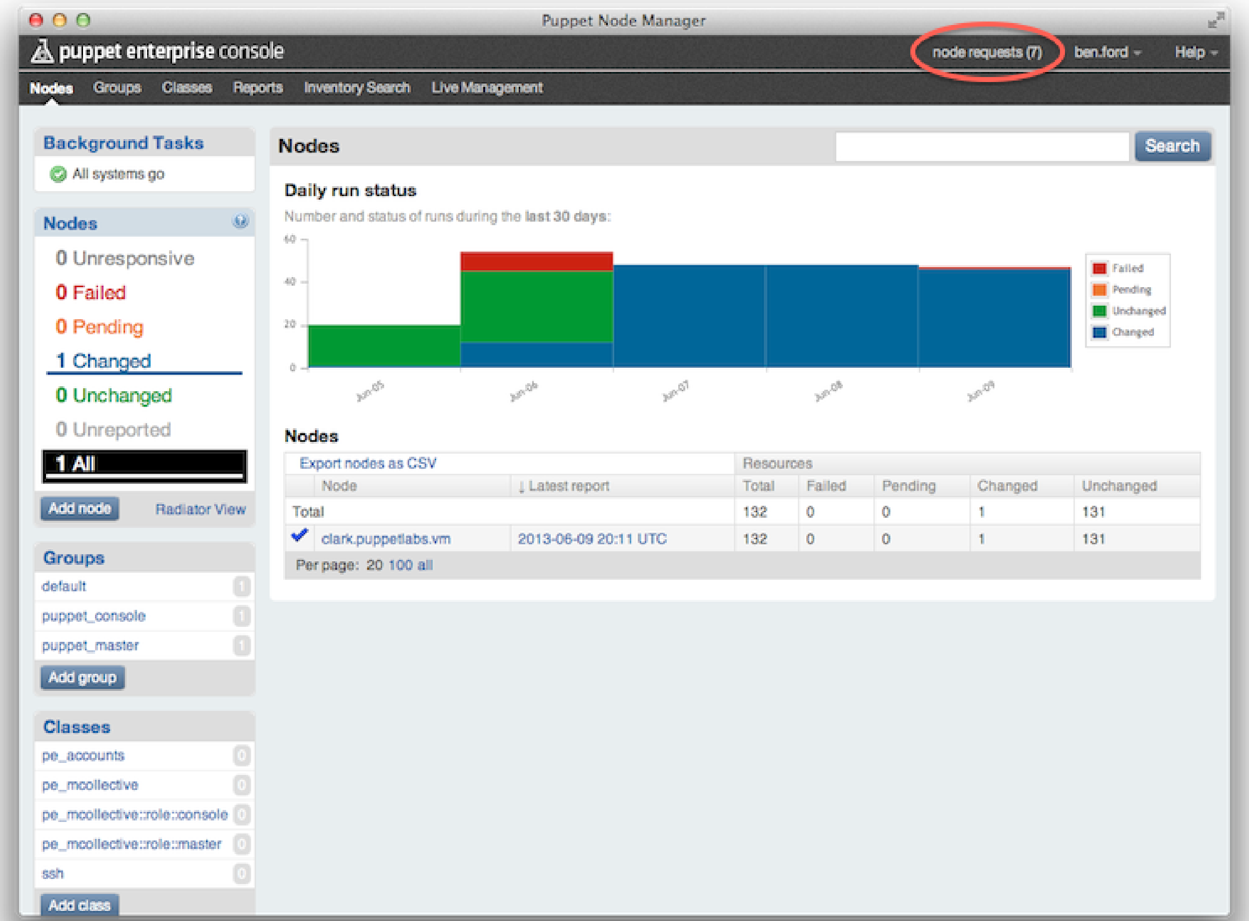
```
[root@training ~]# puppet cert generate kermit.puppetlabs.com
notice: kermit.puppetlabs.com has a waiting certificate request
notice: Signed certificate request for kermit.puppetlabs.com
notice: Removing file Puppet::SSL::CertificateRequest kermit.puppetlabs.com at '/etc/
notice: Removing file Puppet::SSL::CertificateRequest kermit.puppetlabs.com at '/etc/
```

- Can be used as part of your provisioning process.
- Simply transfer the generated certificates onto the freshly built system.
- When the agent starts for the first time, it's already authorized to request a catalog from the master.

Signing Certificates with PE Console

- Before PE 2.7
    - Certificate signing had to be done manually via command line.
- Starting with PE 2.7
    - PE Console added the capability to view and respond to Node requests graphically.
    - Node requests can be approved or rejected in the PE Console without login access to the Puppet Master.
    - Users with read/write privileges in the PE Console can take action on node requests.

Viewing Node Requests



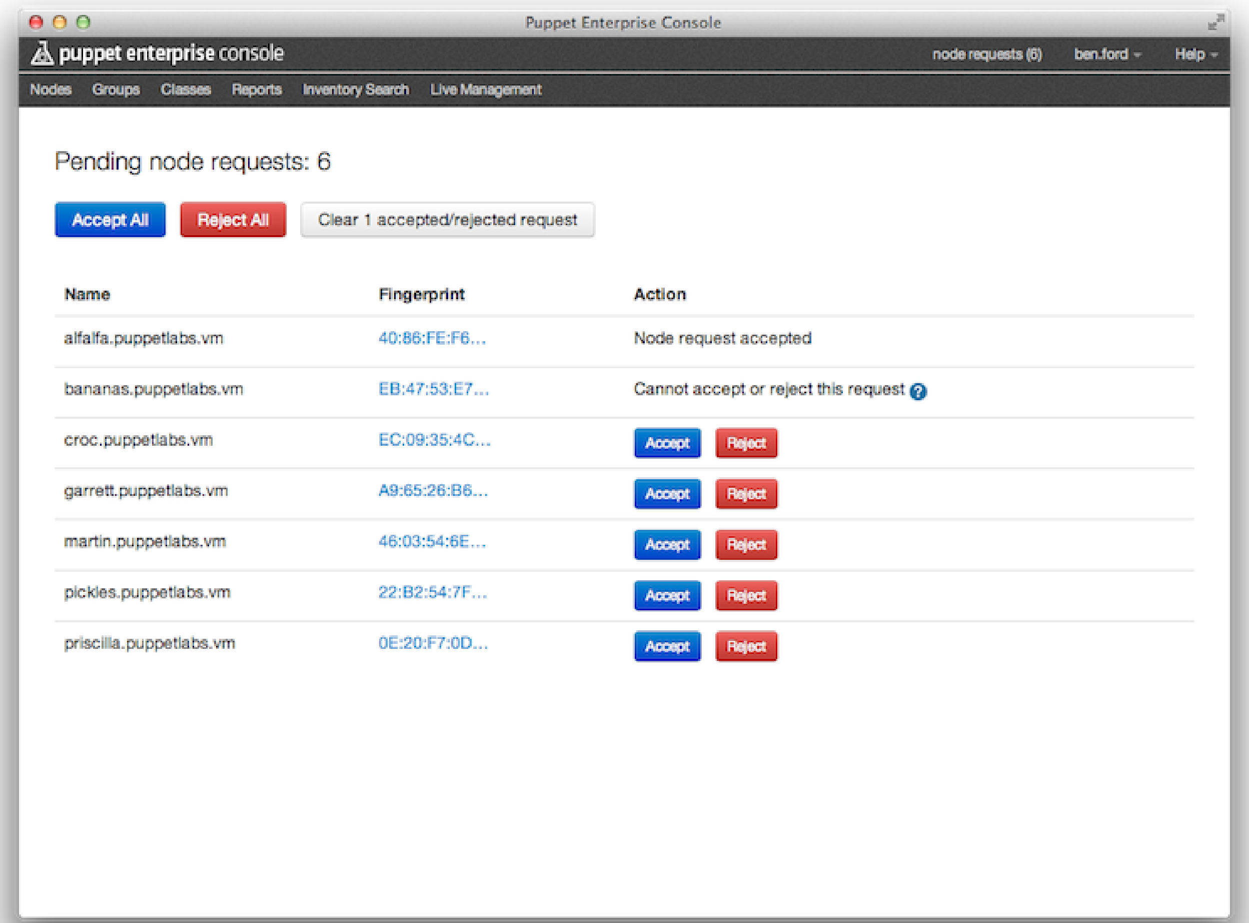
Nodes

		Resources				
Node	↓ Latest report	Total	Failed	Pending	Changed	Unchanged
Total		132	0	0	1	131
✓ clark.puppetlabs.vm	2013-06-09 20:11 UTC	132	0	0	1	131

Notes:

- Click on the pending nodes indicator to view and manage current requests. This will bring up a view containing a list of all the pending node requests.
- Each item on the list shows the node's name and its corresponding CSR's fingerprint.
- Click on any truncated fingerprint to view the entire fingerprint in a pop-up.

Rejecting and Approving Node Requests

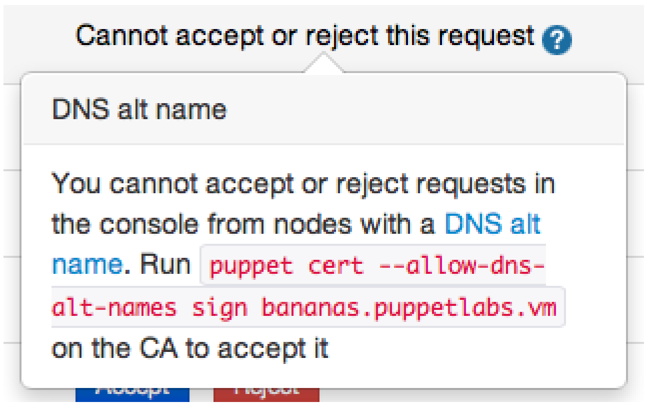


Pending node requests: 6

Name	Fingerprint	Action
alfalfa.puppetlabs.vm	40:86:FE:F6...	Node request accepted
bananas.puppetlabs.vm	EB:47:53:E7...	Cannot accept or reject this request
croc.puppetlabs.vm	EC:09:35:4C...	Accept Reject
garrett.puppetlabs.vm	A9:65:26:B6...	Accept Reject
martin.puppetlabs.vm	46:03:54:6E...	Accept Reject
pickles.puppetlabs.vm	22:B2:54:7F...	Accept Reject
priscilla.puppetlabs.vm	0E:20:F7:0D...	Accept Reject

Notes:

- In some cases, you may see the message *Cannot accept or reject this request*.



Cannot accept or reject this request

DNS alt name

You cannot accept or reject requests in the console from nodes with a DNS alt name. Run `puppet cert --allow-dns-alt-names sign bananas.puppetlabs.vm` on the CA to accept it

- This is often because DNS altnames have been used. You cannot use the console to approve/reject such node requests. The CSR for those nodes must be accepted or rejected using puppet cert on the CA. For more information, see the DNS altnames entry in the reference guide.
- In some cases, attempting to accept or reject a node request will result in an error. This is typically because the request has been modified somehow, usually by being accepted or rejected elsewhere (e.g. by another user or from the CLI) since the request was first generated.
- Accepted/rejected nodes will remain displayed in the console for 24 hours after the action is taken. This interval cannot be modified. However, you can use the "Clear accepted/rejected requests" button to clean up the display at any time.

Demo

Triggering a Puppet Run

Modules and Classes

Lesson 5: Modules and Classes

Objectives

At the end of this lesson, you will be able to:

- Describe Puppet modules and classes.
- Describe the structure of a Puppet module.
- Identify the benefits of using a module to contain configuration.
- Identify the concept of autoloading content in modules.
- Differentiate between defining and declaring classes.

Puppet Classes

Classes define a collection of resources that are managed together as a single unit.

```
# /etc/puppetlabs/puppet/modules/ssh/manifests/init.pp

class ssh {

  package  { 'openssh-clients':
    ensure => present,
  }

  file { '/etc/ssh/ssh_config':
    owner   => 'root',
    group   => 'root',
    mode    => '0644',
    require => Package['openssh-clients'],
    source  => 'puppet:///modules/ssh/ssh_config',
  }

  service { 'sshd':
    ensure => stopped,
    enable => false,
  }

}
```

Notes:

- Stated another way, `package`, `file`, and `service` are individual Puppet resources bundled together to define a single idea, or class.
- Class definitions are contained in manifests. The `init.pp` file above is an example of a manifest written in Puppet DSL.
- Note that there is a trailing comma after the last attribute in each resource above. This is not required, but is best practices because it reduces the chances of errors throughout the lifetime of the manifest file.

Node Definitions

Multiple classes are declared together to represent a role.

For example, to build a web application from Puppet classes on `oscar.example.com`:

```
node 'oscar.example.com' {
  include ssh
  include apache
  include mysql
  include web-app
}
```

Notes:

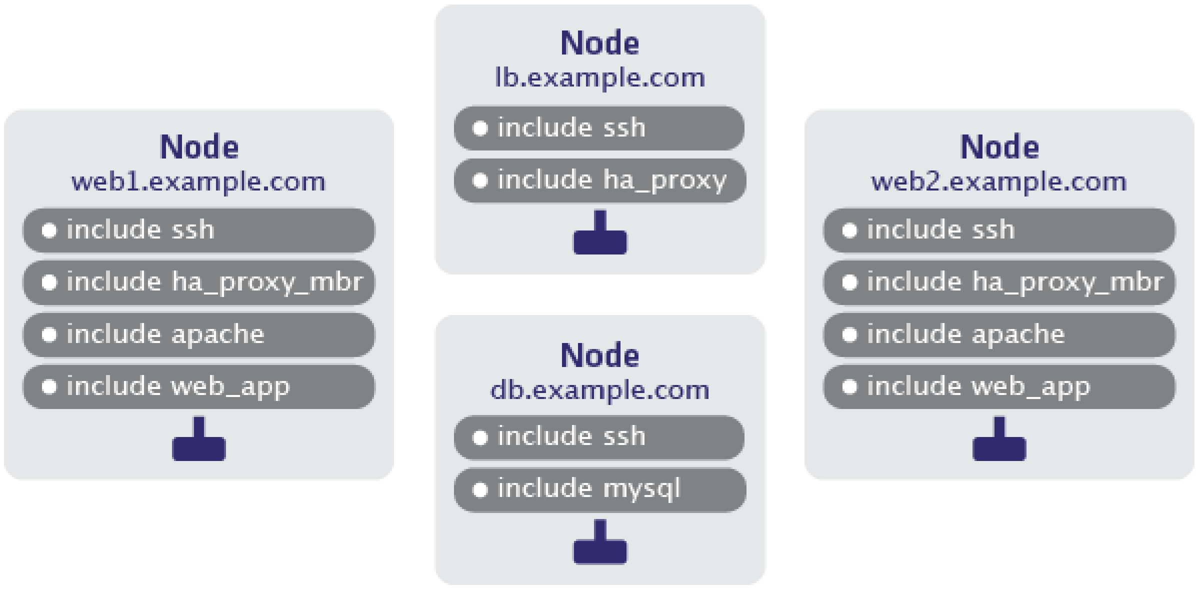
This is a node definition which represents the agent machine and the classes that compose its Puppet configuration. When the node `oscar.example.com` requests a catalog from the master, these classes will be used to build it.

Node definitions can match based on simple strings, like above, or they can match based on regular expressions. Regular expressions are only used when no exact match is found, and they are compared in order until a regex matches, regardless of specificity.

Best practices are to avoid any complex logic in node definitions and simply include the required classes. This leads to a configuration model that is more readable and more composable. It also makes the transition to an External Node Classifier like the Enterprise Console a painless process.

Classes are Reusable

If a more complex deployment is needed, reusing existing classes saves effort and reduces error.



Classes are Singleton

Classes must be unique and can only used once on a given node.

```
class ssh {
  package { 'ssh':
    ensure => present,
  }
  file { '/etc/ssh/sshd_config':
    ensure => file,
    owner  => 'root',
    group  => 'root',
  }
  service { 'sshd':
    ensure => running,
    enable => true,
  }
}

include ssh
include ssh
```

You only ever get one instance of the class in the catalog.

Notes:

In most cases, classes can be included multiple times. Puppet is smart enough to only declare the class once. This means that best practices are to include a class when it's going to be referenced, even if that means that it may be included many times.

Modules

Modules are directories that contain your configuration. They are designed to encapsulate all of the components related to a given configuration in a single folder hierarchy.

- They have a pre-defined structure that enable the following:
    - auto-loading of classes
    - file-serving for templates and files
    - auto-delivery of custom Puppet extensions
    - easy sharing with others

Auto-loading of Classes

`import` is considered harmful.

Previous versions of Puppet required you to manually import source code files.

```
import '/etc/puppet/path/to/your/manifest.pp'
```

- This isn't manageable past a dozen or so manifests.
- Doesn't encourage reusable code.
- Not best practice and mostly obsolete.

Notes:

Puppet source code files are commonly referred to as *manifests*.

You should generally avoid the `import` keyword. It was introduced to the language before modules existed, and was rendered mostly obsolete once Puppet could autoload classes and defined types from modules.

The one modern use for importing is to allow node definitions to be stored in several files. However, note that this requires you to restart the puppet master or touch `site.pp` whenever you edit your nodes. This practice is also mostly obsoleted by modern node classification schemes.

Auto-loading of Classes

Modules enable class auto-discovery.

- First, Puppet needs to know where to find your modules.

```
# puppet.conf on puppet master
[master]
  modulepath=/etc/puppetlabs/puppet/modules
```

- Then, your classes are placed in this predictable structure.

```
[root@training ~]# tree /etc/puppetlabs/puppet/modules/ssh/
...
├── manifests
    ├── init.pp              ## class ssh { ... }
    └── server.pp            ## class ssh::server { ... }
```

- Puppet expects to find classes in the `manifests` directory of your module.

Auto-loading of Classes

Class names can be broken into namespaces.

Class names map directly to where Puppet expects to find them.

- The first segment in a name identifies the module.
- The final segment in a name identifies the filename.
- Any intermediary segments are evaluated as subdirectories of the module's `manifests` directory.
- The module's default class is located in the `manifests/init.pp` file and has the same name as the module itself.

```
[root@training ~]# tree /etc/puppetlabs/puppet/modules/apache/
...
├── manifests
    ├── init.pp              ## class apache { ... }
    ├── mod
    │   └── php.pp           ## class apache::mod::php { ... }
    └── mod.pp               ## class apache::mod { ... }
```

Where would we expect to find the class `foo::bar::baz`?

Lab 5.1: Build Your First Module

- **Objective:**
  - Construct and test a Puppet Module to manage a user account `introduction`.
- **Steps:**
  - Create the module directory structure & support files.
  - Validate the syntax of your user class.

Notes:

What happens if you run `puppet apply` against your init.pp manifest?

Define and Declare

Now that we have built our class, how do we use it?

define:
To specify the contents and behavior of a class. Defining a class doesn't automatically include it in a configuration; it simply makes it available to be declared.

declare:
To direct Puppet to include or instantiate a given class. To add classes, use the `include` keyword or the `class {'foo':}` syntax.

Notes:

From Puppet Docs:

- define -- http://docs.puppetlabs.com/references/glossary.html#define
- declare -- http://docs.puppetlabs.com/references/glossary.html#declare

Defining vs Declaring

When you build a class like the following, you are defining it.

```
class ssh {
  package  { 'openssh-clients':
    ensure => present,
  }
  file { '/etc/ssh/ssh_config':
    owner   => 'root',
    group   => 'root',
    mode    => '0644',
    require => Package['openssh-clients'],
    source  => 'puppet:///modules/ssh/ssh_config',
  }
  service { 'sshd':
    ensure => stopped,
    enable => false,
  }
}
```

To use it, you need to declare the class.

```
include ssh
```

Declaring a class instructs Puppet to enforce the class.

Notes:

You may have seen the *resource-like* class declaration syntax:

```
class classname { 'title': }
```

This is designed for Parameterized Classes and will be covered in the "Advanced Classes" lesson and in the *Advanced Puppet* training course.

Declaration Testing

Preparing to test our declarations:

- Save example usage (class declarations) with the module.
    - ad hoc testing during development
    - example usage when sharing with others

```
[root@training ~]# tree /etc/puppetlabs/puppet/modules/ssh
├── manifests
│   ├── init.pp              ## class ssh { ... }
│   └── server.pp            ## class ssh::server { ... }
└── tests
    ├── init.pp              ## include ssh
    └── server.pp            ## include ssh::server
```

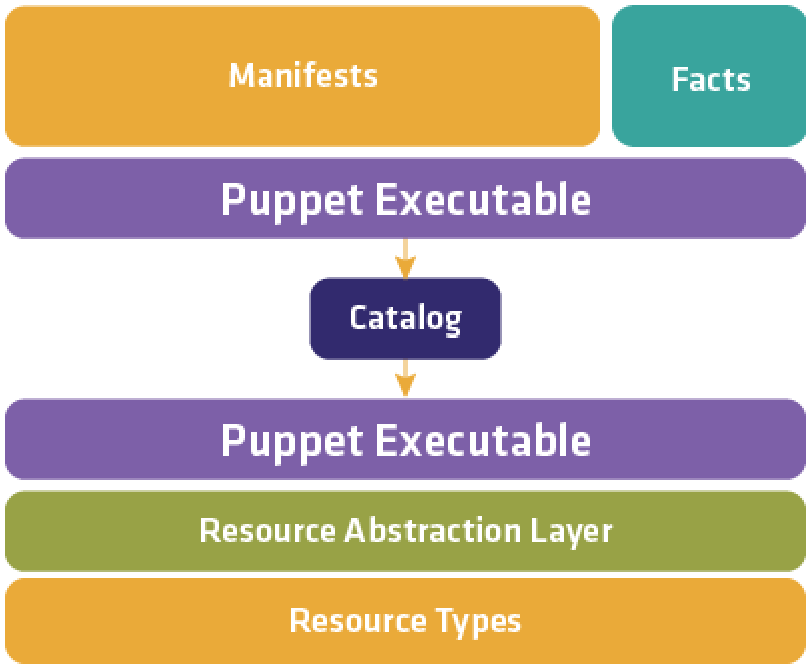
Each smoke test should declare the class it is testing.

```
# /etc/puppetlabs/puppet/modules/ssh/tests/init.pp

include ssh
```

The `puppet apply` Executable

- Compiles puppet manifest into a resource catalog.
- Uses the Resource Abstraction Layer to simulate or enforce the catalog locally.



Notes:

- In agent/master Puppet arrangements, agent nodes send their facts to the master, and the master compiles the catalog using these facts. When using `puppet apply`, local facts are used to build the catalog.
- When using `puppet apply`, remember to apply against files in the `tests` directory, not in the manifests directory.
- Files in the manifests directory contain the resource definitions, but to implement, defined resources need to be declared and the files in the `tests` directory contain the declaration, which will actually initiate action.
- There is no harm in running `puppet apply` against files in the manifests directory, but this will not apply any changes.
- Running `puppet apply` against files in the `tests` directory can be used as an ad hoc verification or proof of concept to see how the module will manage the system once implemented.

Applying a Smoke Test

One-off manifest enforcement.

- Validate your code.
- Enforce a class locally one time only.
- Temporary changes that may be overridden on the next Agent run.
- `puppet apply` compiles a manifest file and enforces it immediately.

```
[root@training ssh]# puppet apply tests/init.pp
notice: /Stage[main]/Ssh/Service[sshd]/ensure: ensure changed 'stopped' to 'run...
notice: Finished catalog run in 0.14 seconds
```

Simulating Change with Puppet

`--noop` mode simulates without enforcing.

- Resource Abstraction Layer can simulate events rather than taking action.
- Inform you of system drift and expected convergence actions.

```
[root@training sudoers]# puppet apply --noop tests/init.pp
notice: //File[/etc/sudoers]/mode: current_value 0646, should be 0440 (noop)
notice: Finished catalog run in 0.03 seconds
```

Simulating Change with Puppet

`--noop` mode simulates without enforcing.

Once convergence actions are verified, Puppet can be run without `--noop` to enforce the change in state.

```
[root@training sudoers]# puppet apply --noop tests/init.pp
notice: //File[/etc/sudoers]/mode: current_value 0646, should be 0440 (noop)
notice: Finished catalog run in 0.03 seconds
[root@training sudoers]# puppet apply tests/init.pp
notice: //File[/etc/sudoers]/mode: mode changed '0646' to '0440'
notice: Finished catalog run in 0.03 seconds
```

Individual resources may also be placed in `noop` mode.

```
package { 'kernel':
  ensure => latest,
  noop   => true,
}
```

Notes:

Because Puppet can inspect the current state of your system and knows how to declare your resource to be present or absent statefully, it can inspect what the current state of your system is and give you meaningful information about what it would take to configure your system from its running state to the state you have declared in your Puppet manifests.

The `--noop` flag can be used in both the apply and agent roles. It can also be applied to individual resources in the manifest itself. For example, just like `--noop` as the parameter for `puppet apply`, you can enable simulation for individual resources when you want to monitor what would happen for a given resource, should it be enforced.

Having simulation capabilities built into every Puppet type without additional effort from the systems administrator is part of what separates Puppet from other configuration management tools.

Lab 5.2: Use Your Module

- **Objective:**
  - Enforce your `users` class on your local agent.

- **Steps:**
  - Create a smoke test that includes your class.
  - Apply your smoke test with `puppet apply`.

A Simple Group Resource Declaration

```
group { 'sysadmin':
    ensure => present,
    gid    => '5000',
}
```

Additional Attributes

- `name`: The group name.
- `ensure`: Group resource state. Valid values are `present`, `absent`.
- `gid`: The numerical group ID.
- `members`: Members of the group.

Puppet Describe

Want to know more?

```
$ puppet describe group

- **allowdupe**
   Whether to allow duplicate GIDs.  This option does not work on
   FreeBSD (contract to the `pw` man page).  Valid values are `true`,
   `false`.

- **attribute_membership**
   Whether specified attribute value pairs should be treated as the only
   attributes of the user or whether they should merely be treated as the
   minimum list.  Valid values are `inclusive`, `minimum`.
......
......
```

Lab 5.3: Expand Your Module

- **Objective:**
  - Extend your module to manage multiple resource types.
- **Steps:**
  - Modify your `users` class to manage a `group` resource.
  - Test & apply your class locally.

Classification

Lesson 6: Classification

Objectives

At the end of this lesson, you will be able to:

- Describe node classification.
- Write node declarations in your site manifest.
- Use the Puppet Enterprise Console to classify nodes.

`site.pp`

The main entry point for entire Puppet network.

- The standard manifest file for the Puppet Master.
- Compiled any time an agent connects and requests a catalog.
- Can contain global resources and classes that apply to all nodes equally.
- Puppet Enterprise uses it to configure file backups.
- PE defaults to `/etc/puppetlabs/puppet/manifests/site.pp`.

Notes:

Puppet Enterprise default location of `site.pp` is `/etc/puppetlabs/puppet/manifests/site.pp`. The location of `site.pp` can be configured with the manifest setting.

If you're using separate node definition files and import them into `site.pp` (with an `import nodes/*.pp`, for example), any new files added won't get noticed until you restart the Puppet Master.

To ensure new files with node definitions are actually read, `touch site.pp` (or the importing file).

Node Definitions

Include node specific configuration.

- Puppet node definitions look similar to classes.
- The node definition corresponding to the Agent's name is declared automatically.
- Only one node definition is ever declared.
- By default, the Agent node’s name is its `certname`.

```
node 'foo.puppetlabs.com' {
  include ssh
}
```

When the node `foo.puppetlabs.com` connects to the Puppet Master, it will be assigned the `ssh` class.

Notes:

An agent node's `certname` is how it is identified in the Puppet network. It is set at install time but can be changed later. The `certname` is usually (but not always) the node's fully qualified domain name.

Best practices are to avoid any complex logic in node definitions and simply include the required classes. This leads to a configuration model that is more readable and more composable. It also makes the transition to an External Node Classifier like the Enterprise Console a painless process.

Regular Expressions

Configure nodes by nodename patterns.

- Regular expressions can be used to define nodes.
- The first match found is declared, regardless of specificity.
- Regular expressions are only evaluated if no exact match is found.

```
node /^web\d{3}.puppetlabs.com$/ {
  include ssh
  include apache
  include mysql
  include mywebapp
}
```

When a web application server, identified by a nodename of `webXXX`, connects to the Puppet Master, it will be assigned the classes above.

Notes:

Remember that regular expressions are not as readable as simple strings are. As such, best practices are to, when possible, minimize the use of regular expressions to make it more clear which node definition will be enforced. See http://docs.puppetlabs.com/puppet/3/reference/lang*node*definitions.html for more information.

Default Node

When no other node declaration matches.

```
node default {
  notify { "${::fqdn} has no node definition": }
}
```

- You can specify a node named `default`.
- This will be used if no directly matching node is found.
- Sometimes used when many of only a single type of system are on a network.

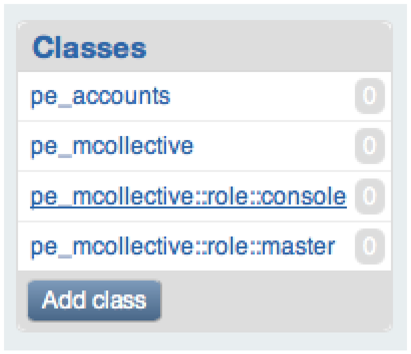
Demo

`/etc/puppetlabs/puppet/manifests/site.pp`

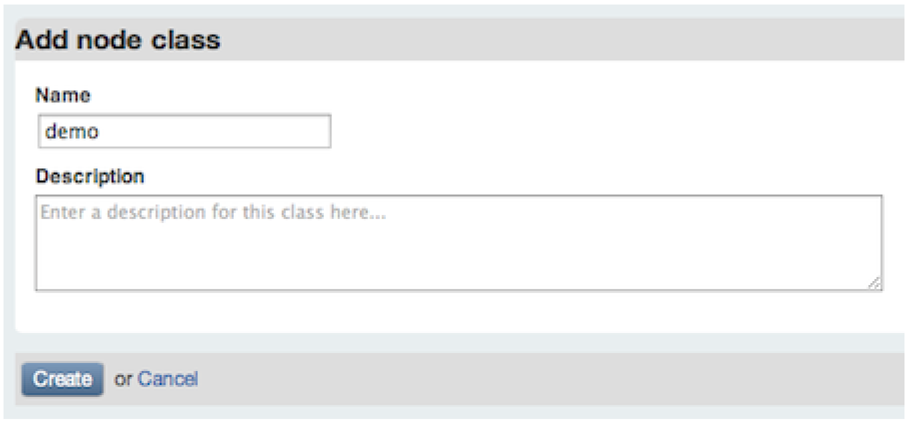
Console Classification

Adding a class to the Console.

- "Add class" button in the console's sidebar:


Classes
pe_accounts 0
pe_mcollective 0
pe_mcollective::role::console 0
pe_mcollective::role::master 0
Add class

- Type the new class's name in the text field and click "Create":



Notes:

Students often ask why you must inform the Console of the classes that you can use. There are three major reasons.

First, the Console is designed to be a standalone role. It is often installed on the same machine as the Master, but not always. Due to this, it cannot simply parse source code to discover the classes. In that case, why don't we have a REST API to query the Master for a list of all known classes?

The reason for that is that Puppet modules are often designed with internal classes that are never meant to be called by end users. These might be `params` classes or internal composition classes. Puppet currently does not have a way of marking the visibility of classes, so simply parsing source code for a list of available classes works for the compiler, but it cannot work for a classifier, which needs a list of public classes only.

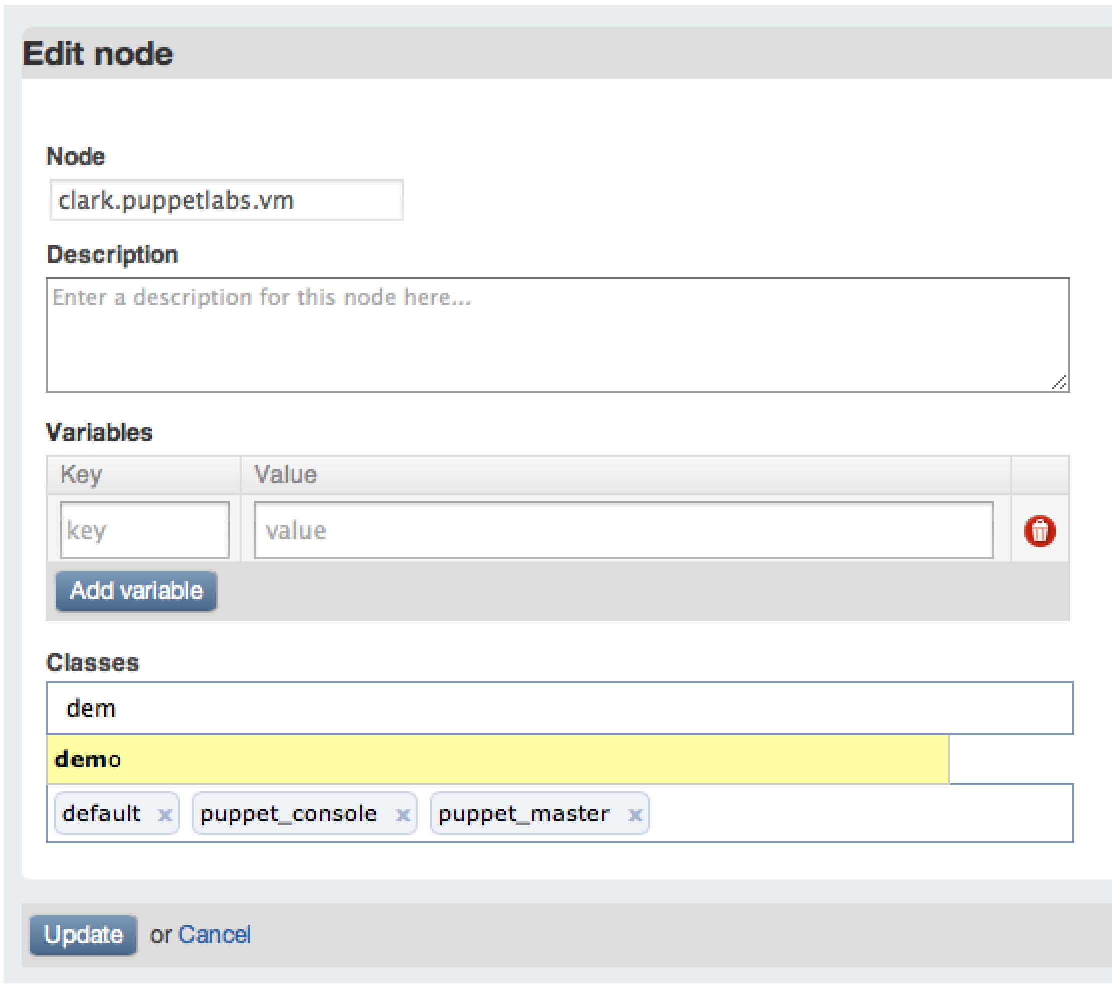
The third major reason is that often system administrators do not want to expose their end users to all the complexities of their system and will only list a smaller subset of classes for users to choose from. This works well with the *roles & profiles* pattern that is becoming common practice.

As a workaround to avoid manual data entry, rake tasks are available that can programmatically add classes to the console.

See http://docs.puppetlabs.com/pe/latest/console<em>classes</em>groups.html#rake-api for more information.

Console Classification

Assigning a Class to a Node.



Node
clark.puppetlabs.vm

Description
Enter a description for this node here...

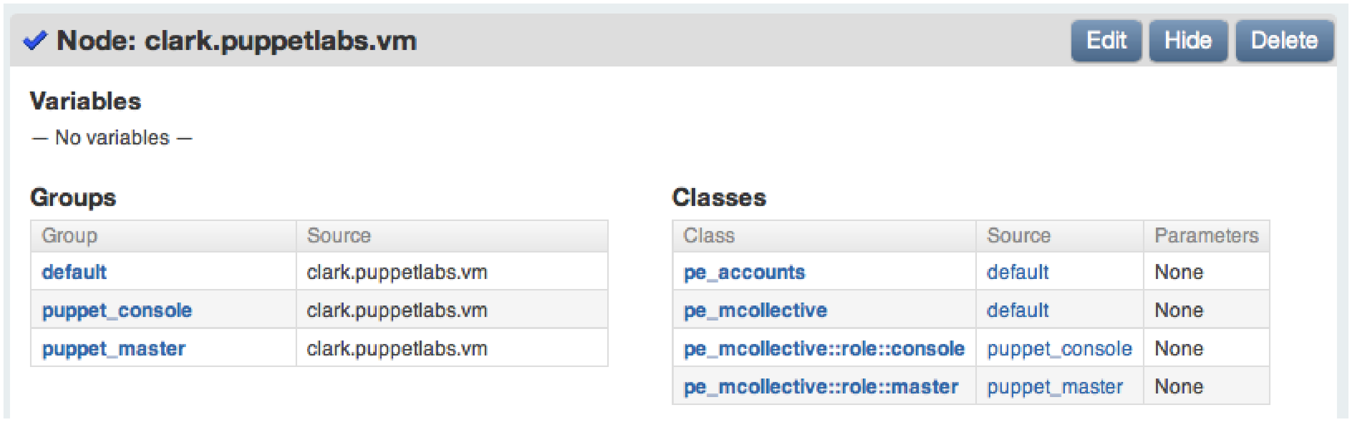
Variables

Key	Value
key	value

Classes
dem
demo
default x puppet_console x puppet_master x

Console Classification

Node definition



Variables

— No variables —

Groups

Group	Source
default	clark.puppetlabs.vm
puppet_console	clark.puppetlabs.vm
puppet_master	clark.puppetlabs.vm

Classes

Class	Source	Parameters
pe_accounts	default	None
pe_mcollective	default	None
pe_mcollective::role::console	puppet_console	None
pe_mcollective::role::master	puppet_master	None

Equivalent to:

```
node 'clark.puppetlabs.vm' {
  include pe_accounts
  include pe_mcollective
  include pe_mcollective::roles::console
  include pe_mcollective::roles::master
}
```

Demo

Classification of nodes with the Console.

Puppet Forge

Lesson 7: Puppet Forge

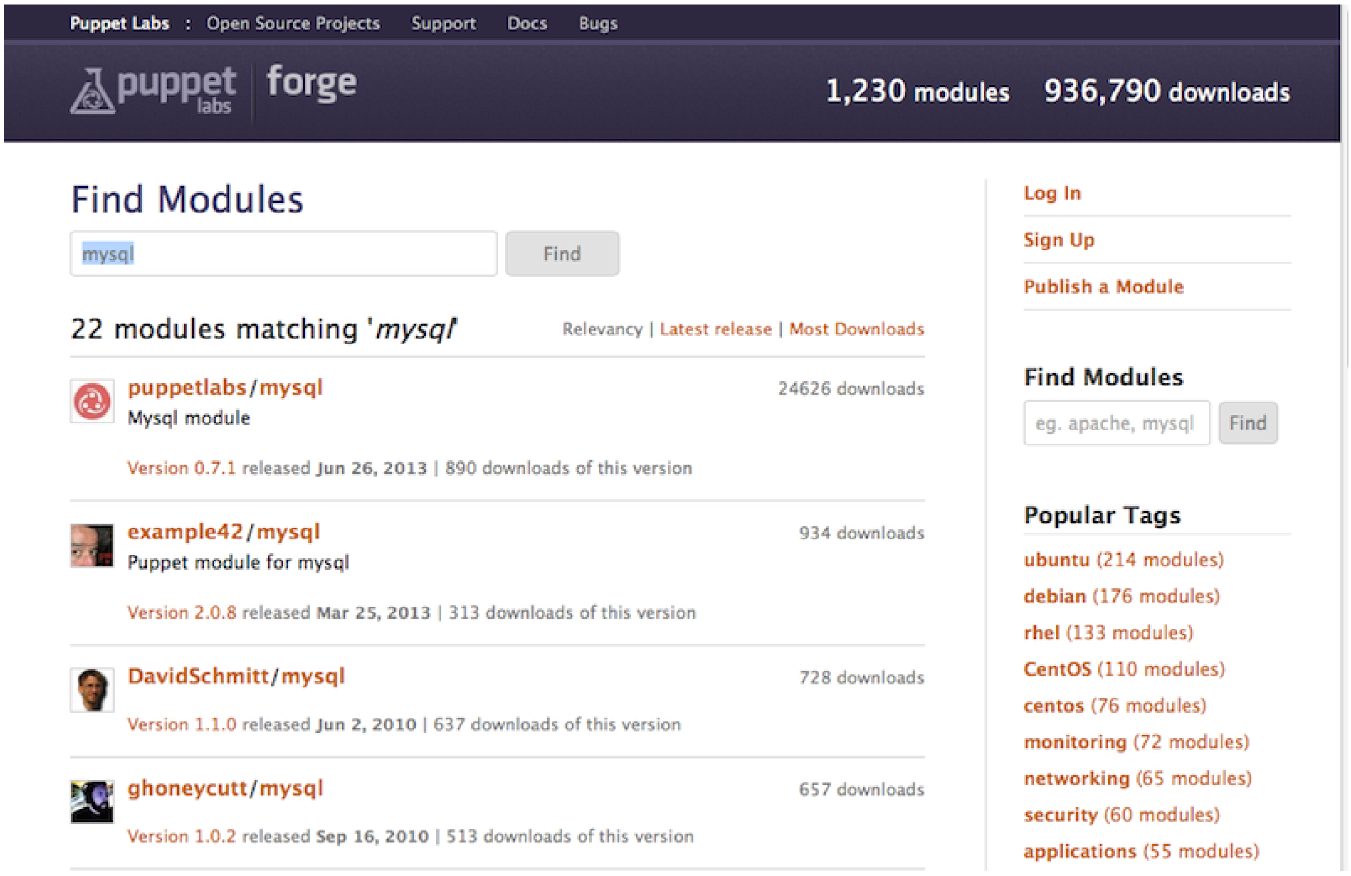
Objectives

At the end of this lesson, you will be able to:

- Use the Puppet Module Tool to list installed modules.
- Find and install Puppet modules from the Forge.
- Explain how to share modules with others using the Forge.

Puppet Forge

Share & Download Puppet Modules.


Puppet Labs : Open Source Projects Support Docs Bugs
puppet labs forge
1,230 modules 936,790 downloads
Log In
Sign Up
Publish a Module
Find Modules
eg. apache, mysql
Find

Find Modules

mysql [Find]

22 modules matching '*mysql*' Relevancy | Latest release | Most Downloads

puppetlabs/mysql 24626 downloads
Mysql module
Version 0.7.1 released Jun 26, 2013 | 890 downloads of this version

example42/mysql 934 downloads
Puppet module for mysql
Version 2.0.8 released Mar 25, 2013 | 313 downloads of this version

DavidSchmitt/mysql 728 downloads
Version 1.1.0 released Jun 2, 2010 | 637 downloads of this version

ghoneycutt/mysql 657 downloads
Version 1.0.2 released Sep 16, 2010 | 513 downloads of this version

Popular Tags

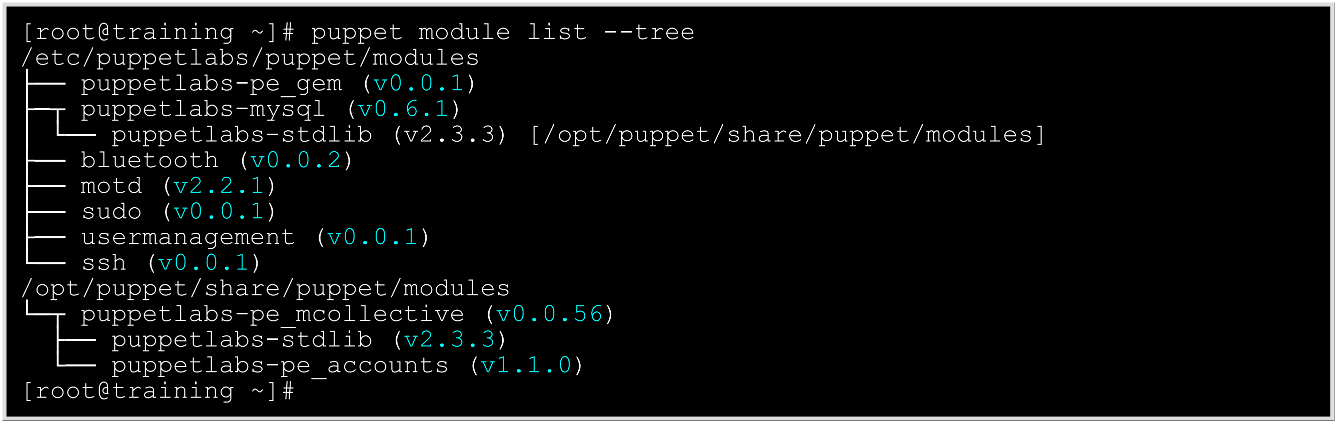
ubuntu (214 modules)
debian (176 modules)
rhel (133 modules)
CentOS (110 modules)
centos (76 modules)
monitoring (72 modules)
networking (65 modules)
security (60 modules)
applications (55 modules)

Puppet Module Tool

From the command line, you can:

- Search for Modules.
- Install Modules (with dependencies).
- List installed Modules.

Puppet Module List



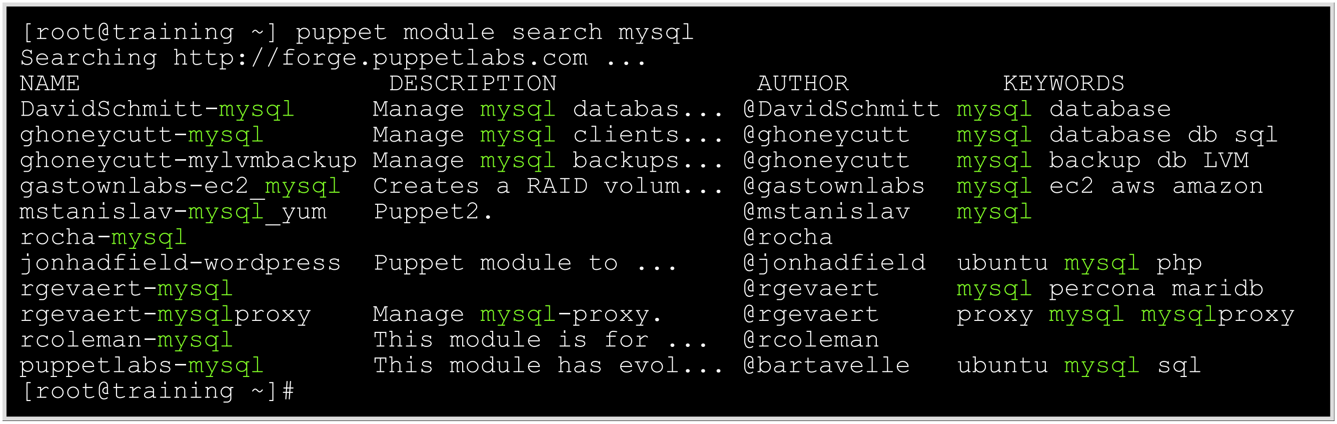
```
[root@training ~]# puppet module list --tree
/etc/puppetlabs/puppet/modules
├── puppetlabs-pe_gem (v0.0.1)
├─┬ puppetlabs-mysql (v0.6.1)
│ └── puppetlabs-stdlib (v2.3.3) [/opt/puppet/share/puppet/modules]
├── bluetooth (v0.0.2)
├── motd (v2.2.1)
├── sudo (v0.0.1)
├── usermanagement (v0.0.1)
└── ssh (v0.0.1)
/opt/puppet/share/puppet/modules
└─┬ puppetlabs-pe_mcollective (v0.0.56)
  ├── puppetlabs-stdlib (v2.3.3)
  └── puppetlabs-pe_accounts (v1.1.0)
[root@training ~]#
```

Notes:

The version information comes out of the module's metadata files that are required for posting modules to the Forge. Since we haven't written our modules for sharing, they have no metadata and no versioning or dependency information.

More information on publishing modules can be found at http://docs.puppetlabs.com/puppet/latest/reference/modules_publishing.html.

Puppet Module Search



```
[root@training ~] puppet module search mysql
Searching http://forge.puppetlabs.com ...
NAME                        DESCRIPTION               AUTHOR          KEYWORDS
DavidSchmitt-mysql         Manage mysql databas... @DavidSchmitt mysql database
ghoneycutt-mysql           Manage mysql clients... @ghoneycutt   mysql database db sql
ghoneycutt-mylvmbackup Manage mysql backups... @ghoneycutt   mysql backup db LVM
gastownlabs-ec2_mysql     Creates a RAID volum... @gastownlabs  mysql ec2 aws amazon
mstanislav-mysql_yum      Puppet2.                  @mstanislav   mysql
rocha-mysql                                         @rocha
jonhadfield-wordpress     Puppet module to ...     @jonhadfield  ubuntu mysql php
rgevaert-mysql                                      @rgevaert     mysql percona maridb
rgevaert-mysqlproxy       Manage mysql-proxy.      @rgevaert     proxy mysql mysqlproxy
rcoleman-mysql            This module is for ...   @rcoleman
puppetlabs-mysql          This module has evol... @bartavelle   ubuntu mysql sql
[root@training ~]#
```

Demo

Using the Forge

Additional Puppet Concepts

Lesson 8: Additional Puppet Concepts

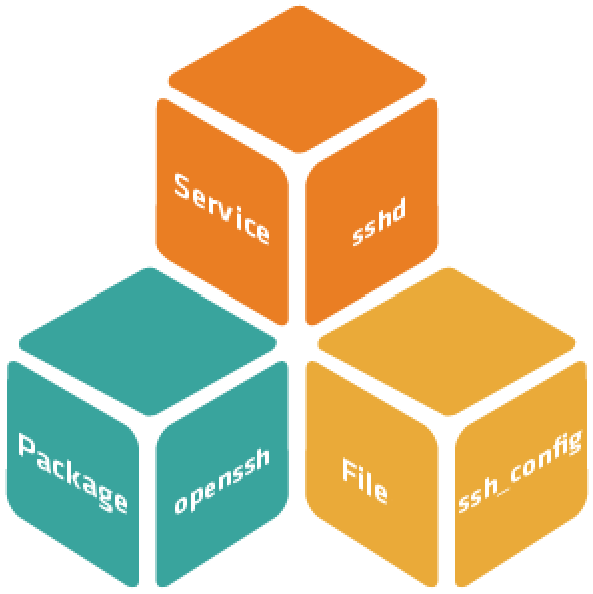
Objectives

At the end of this lesson, you will be able to:

- Discuss the following concepts as they pertain the Puppet IT Automation Software Solution
    - Puppet Resources
    - Puppet Resource Relationships
    - Puppet Variables
    - Puppet Templates
    - Puppet Report Handlers

Puppet Resources

- Resources are building blocks.
- They can be combined to make larger components.
- Together they can model the expected state of your system.



Core Resource Types

- `user`
- `group`
- `host`
- `cron`
- `exec`
- `file`
- `package`
- `service`

Meta Resource Types

- **`notify`**
- `filebucket`
- `resources`
- `schedule`

Platform Specific Resource Types

Core

- `mcx` - OS X Client Management
- `yumrepo` - RedHat ::osfamily software repo
- `zfs` - ZFS filesystems
- `selmodule` - SELinux module

Via Forge modules

- `registry_value` - Windows registry key values
- `netdev_l2_interface` - Layer 2 interface on Cisco or Juniper device
- `f5` - F5 device config
- `netapp` - NetApp network storage devices

Component Specific Resource Types

- `augeas`
- `k5login`
- `selboolean`
- `selmodule`
- `mailalias`
- `maillist`
- `sshkey`
- `ssh_authorized_key`

Resource Limits

Providers are limited to functionality exposed by the OS.

Example: the `user` Resource Type

Provider	Allow Duplicates	Manage Homedir	Manage Passwords	Manage Solaris RBAC
directoryservice			√	
hpxuseradd	√	√		
ldap			√	
netinfo			√	
pw	√	√		
user_role_add	√	√	√	√
useradd	√	√	√	
windows_adsi		√	√	

Notes:

For example, only the Solaris `user`*`role`*`add` provider is able to manage Solaris user roles.

Dependency Management

How does Puppet prioritize the enforcement of resources?

- Puppet does **not** enforce resources top down, based on their position in the manifest.
- Instead, Puppet checks for applicable dependencies between resources in the manifest code.
- Puppet then reorders resource enforcement to meet the determined relationship requirements.

Manifests are parsed in source order when compiling,

but the resource enforcement order is driven by the dependency graph.

Implicit Dependencies

- Specifying each and every dependency is tedious.
- Puppet is smart enough to recognize that certain resources always depend on one another.
- For these related resources Puppet automatically creates implicit dependencies.

Notes:

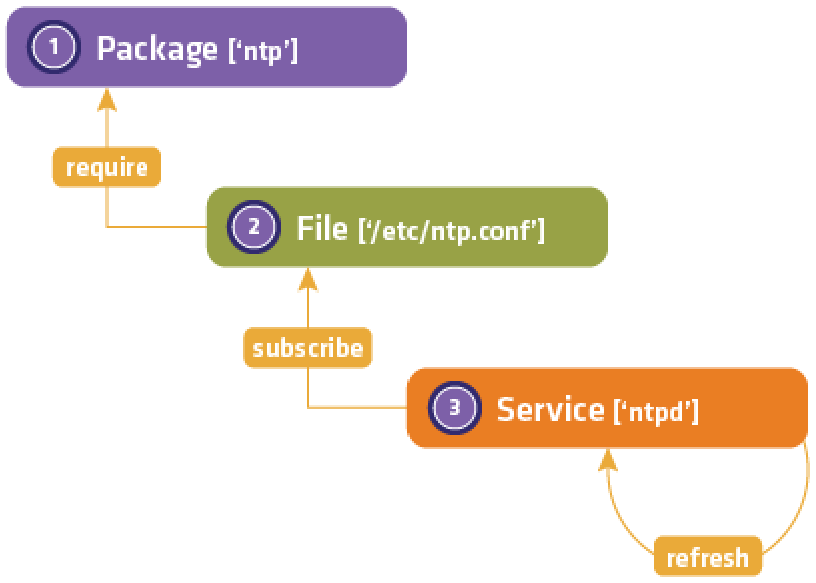
Implicit Resource relationships are superseded by explicit Resource relationship declarations.

Package | File | Service

One of the most useful and common design patterns used in production.

- We commonly specify several resources together to model a complete configuration.
- A reasonable workflow when installing a service is to:
    - Install a package.
    - Configure one or more config files.
    - Enable the service.
- To model this in puppet, we use the package-file-service design pattern.

Workflow recap:



1. Install package.
2. Configure file.
3. Enable service.
4. Restart service when config file is updated.

Variables

Variables are prefixed with '$':

```
$httpd_dir = '/etc/httpd/conf.d'
```

Variables can be used as resource titles:

```
file { $httpd_dir:
  ensure => directory,
}
```

Variables can be used as attribute values:

```
file { '/etc/httpd/conf.d/README':
  ensure  => file,
  content => $readme_content,
}
```

ERB Templates

Ruby's built-in templating language.

- Templates are mostly plain text files.
- Inserting ERB tags allows you to:
    - Display or act on the contents of variables.
    - Alter the flow of logic.
    - Include Ruby code to perform calculations or iterate.

Arrays

The Puppet language supports simple arrays:

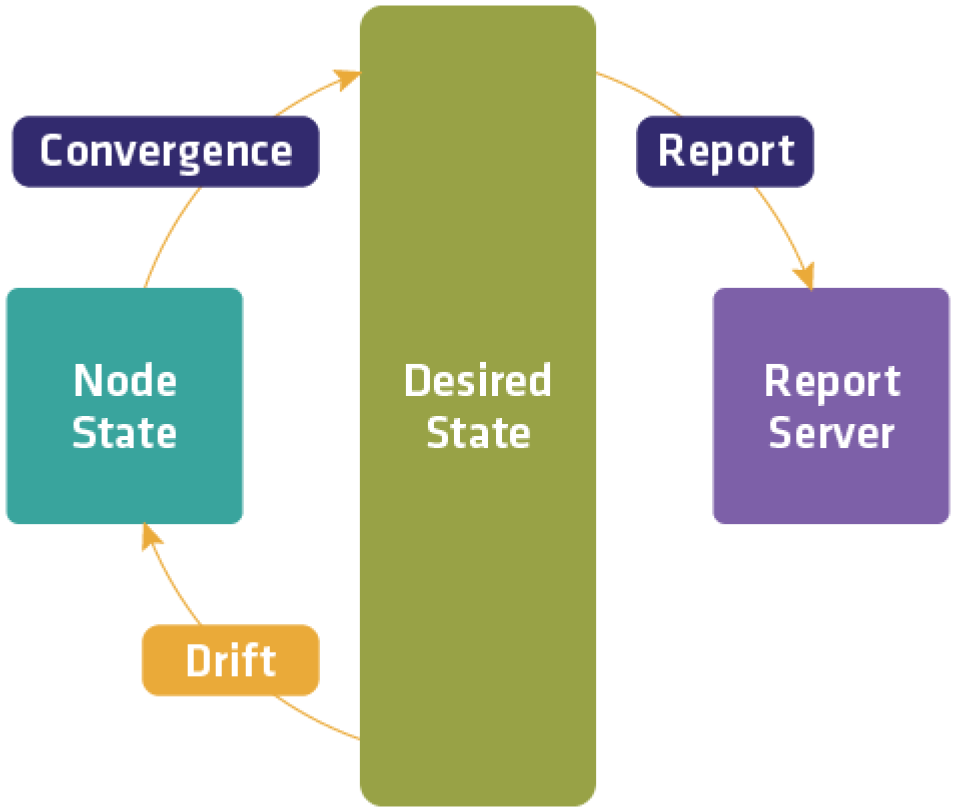
```
$somearray = [ 'one', 'two', 'three' ]
```

Arrays can be used as an argument to some resource parameters:

```
user { 'elvis':
  ensure => present,
  home   => '/home/elvis',
  uid    => '5000',
  gid    => 'hounddog',
  shell  => '/bin/bash',
  groups => ['jailhouse', 'surfer', 'legend'],
}
```

Agent Reports

The Puppet Agent can be configured to generate and send a report to the Puppet Master after every puppet run.



What's in a Report?

Basic Example

```
info: Applying configuration version '1328975856'
notice: Hello World!
notice: /Notify[example]/message: defined 'message' as 'Hello World!'
notice: Finished catalog run in 0.03 seconds
```

Transaction Data

```
notice: /Notify[example]/message: defined 'message' as 'Hello World!'
```

Metric Data

```
notice: Finished catalog run in 0.03 seconds
```

Report Handlers

Process reports on the Master

Built-in Report Handlers:

- `http/https`
- `log`
- `puppetdb`
- `tagmail`
- `rrdgraph`
- `store`

Documentation

Built in `type` documentation.

```
[root@training ~]# puppet describe <type> [-s]

[root@training ~]# puppet describe --list

[root@training ~]# puppet doc -r [type|report|providers|...]
```

- Use the same docstrings used to generate documentation pages.
- The `-s` flag provides a type summary only.
- The `--list` argument will list all types known to Puppet.
- `puppet doc` can output Markdown or PDF files.
  - We use it to generate docs.puppetlabs.com.

Notes:

For more information regarding Resource Types: http://docs.puppetlabs.com/references/latest/type.html

Course Conclusion

Questions & Next Steps

Still Learning:

- Learning Puppet Tutorials - http://docs.puppetlabs.com/learning/
- Download the Learning Puppet VM - http://puppetlabs.com/learning
- Puppet Labs Workshop - https://puppetlabs.com/learn

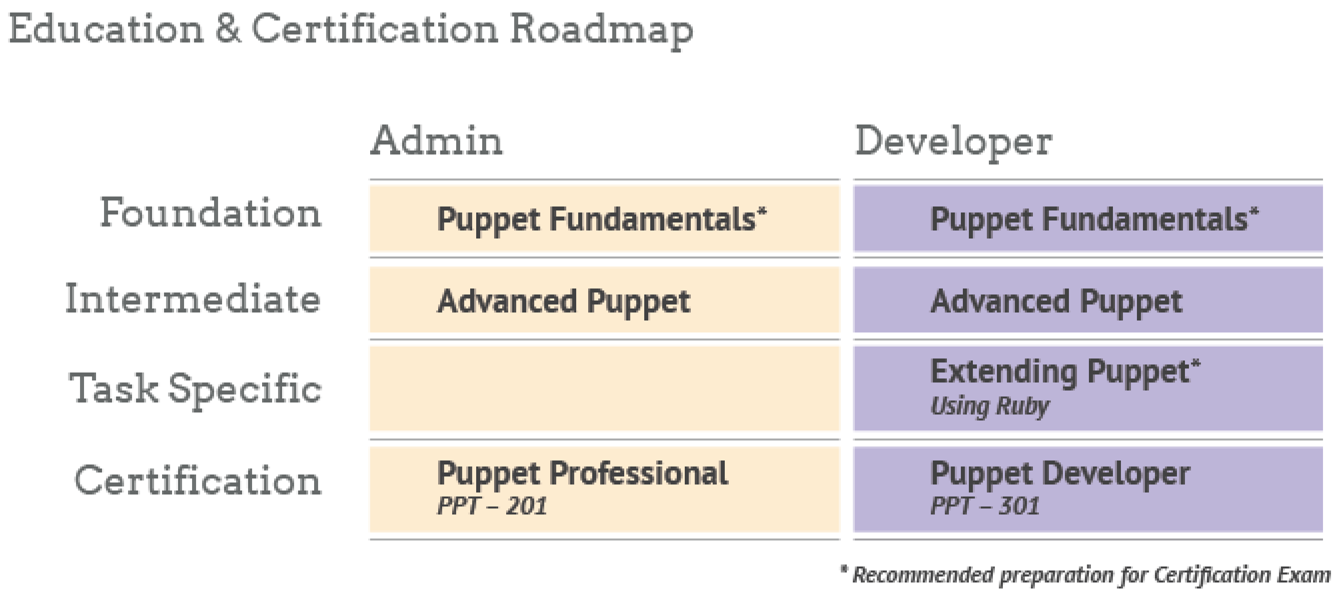
Extending Puppet:

- Download Puppet Enterprise - manage 10 nodes for free
    - http://info.puppetlabs.com/download-pe.html
- Puppet Docs - http://docs.puppetlabs.com/
- Puppet Forge - http://forge.puppetlabs.com/
- Puppet Training - http://puppetlabs.com/category/events/upcoming/

Notes:

Need more technical detail or product drill down?
Schedule a follow-up call with a Puppet Labs Professional Services Engineer.

Puppet Education Roadmap



Education & Certification Roadmap

	Admin	Developer
Foundation	Puppet Fundamentals*	Puppet Fundamentals*
Intermediate	Advanced Puppet	Advanced Puppet
Task Specific		Extending Puppet* *Using Ruby*
Certification	Puppet Professional *PPT – 201*	Puppet Developer *PPT – 301*

** Recommended preparation for Certification Exam*

Appendix

Glossary

module
Self-contained bundles of code and data.

idempotent
Able to be applied multiple times with the same outcome.

define
To specify the contents and behavior of a class or a defined resource type. Defining a class or type doesn't automatically include it in a configuration; it simply makes it available to be declared.

declare
To direct Puppet to include a given class or resource in a given configuration. To declare resources, use the lowercase file {"/tmp/bar":} syntax. To declare classes, use the include keyword or the class {"foo":} syntax. (Note that Puppet will automatically declare any classes it receives from an external node classifier.)

You can configure a resource or class when you declare it by including attribute/value pairs.

Facter
Puppet's system inventory tool. Facter reads facts about a node (such as its hostname, IP address, operating system, etc.) and makes them available to Puppet.

Facter includes a large number of built-in facts; you can view their names and values for the local system by running facter at the command line.

In agent/master Puppet arrangements, agent nodes send their facts to the master.

Puppet Module Cheat Sheet

- Modules are directories with a predictable structure.
- Puppet can automatically load manifests, files, and plugins from modules in its modulepath.
- Use `puppet --configprint modulepath` to see where Puppet expects to find modules on your system.

Example Module: /etc/puppetlabs/puppet/modules/apache

manifests

This directory holds the module's Puppet code.

- Each .pp file should contain one and only one class or defined type.
- Filenames and class/defined type names are related; see the examples below.
- Within a module, the special `$module_name` variable always contains the module's name.

apache/manifests/init.pp

```
class apache {
  ...
}
```

Init.pp is special; it should contain a class (or define) with the same name as the module.

apache/manifests/vhost.pp

```
define apache::vhost ($port, $docroot) {
  ...
}
```

Other classes (and defines) should be named **`modulename::filename`** (without the .pp extension).

apache/manifests/config/ssl.pp

```
class apache::config::ssl {
  ...
}
```

Subdirectories add intermediate namespaces.

lib

This directory holds Ruby plugins, which can add features to Puppet and Facter.

apache/lib/puppet/type/apache_setting.rb

A custom type.

apache/lib/puppet/parser/functions/htpasswd.rb

A custom function.

apache/lib/facter/apache_confdir.rb

A custom fact.

files

Nodes can download any files in this directory from Puppet's built-in file server.

- Use the `source` attribute to download file contents from the server.
- Use `puppet:///` URIs to specify which file to fetch.
- Files in this directory are served at **`puppet:///modules/modulename/filename`**.

apache/files/httpd.conf

To fetch this file:

```
file {'/etc/apache2/httpd.conf':
  ensure => file,
  source => 'puppet:///modules/apache/httpd.conf',
}
```

apache/files/extra/ssl

Puppet's file server can navigate any subdirectories:

```
file {'/etc/apache2/httpd-ssl.conf':
  ensure => file,
  source => 'puppet:///modules/apache/extra/ssl',
}
```

templates

This directory holds ERB templates.

- Use the `template` function to create a string by rendering a template.
- Use the `content` attribute to fill file contents with a string.
- Template files are referenced as **`modulename/filename.erb`**.

apache/templates/vhost.erb

To use this template:

```
file {'/etc/apache2/sites-enabled/wordpress.conf':
  ensure  => file,
  content => template('apache/vhost.erb'),
}
```

Configuration Management as Lego

by Adrien Thebo[1]

Configuration management is hard. Configuring systems properly is a lot of hard work, and trying to manage services and automate system configuration is a serious undertaking.

Even when you've managed to get your infrastructure organized in Puppet manifests or Chef cookbooks, organizing your code can get ugly, fast. All too often a new tool has to be managed under a short deadline, so any sort of code written to manage it solves the immediate problem and no more. Quick fixes and temporary code can build up, and before you know it, your configuration management becomes a tangled mess. Nobody intends for their configuration management tool to get out of hand, but without guidelines for development, all it takes is a few instances of `git commit -a -m 'Good enough'` for the rot to set in.

Organizing configuration management code is clearly a good idea, but how do you do it? For normal development, there are many of design patterns for laying out and organizing programs and libraries. Traditional software development has had around 40 years to mature, and config management is fairly young by comparison and hasn't had the time to have formal best practices.

This is a proposal for an organizational pattern that I'm calling the "Lego pattern." Admittedly, there's nothing revolutionary about these ideas. To be honest, all the ideas espoused in this article are simply applications of the unix philosophy[2]. This pattern can be used to organize code for any configuration management tool, but for the sake of brevity, I'll be using Puppet to provide examples.

The Base Blocks

Fundamental behavior is provided by a set of base modules. These are akin to the rectangular Lego blocks - they're generic, they're reusable, and you can swap them out for similar pieces. Modules like this should be focused on three tenets of the Unix philosophy: the **Rule of Modularity, the Rule of Composition, and the Rule of Separation**[3].

[1] https://twitter.com/nullfinch

[2] http://en.wikipedia.org/wiki/Unix_philosophy

[3] http://www.faqs.org/docs/artu/ch01s06.html

When writing base modules, they should be, well, modular. They should do one thing and do it well. For instance, a module for installing a web application should not manage a database service, neither should it configure logging. while these are valid concerns, they're not directly related. Managing only one service in one module makes that module more reusable and more maintainable.

Base block modules should also be built to be composed with other modules. If a module only handles one service, then it can also safely interact with similar modules. For instance, that web app module only handles installing and running the web app, another module can handle backing up files, and they can be used together to solve the whole of a business problem. If people want to use your module and also back up related files, they won't be forced to use your backup tool - they can use your module to provide the service and use their module to handle backups.

Lastly, base block modules should be built to hide the underlying implementation, and provide a fairly complete interface to the service that they're managing. Modules like this only need to be manipulated via parameters that they expose (much like software libararies), so you can see what options you can tune and configure without having to have complete mastery of the service that its managing. The advantage of this is that you have a clean separation between how the core elements of the service work, and how you're implementing them.

The puppetlabs/apache[4] module is a good example of this. The apache module is designed to give you the set of tools you'll need to manage almost any apache configuration regardless of the underlying system. It hides the system-specific configuration and presents you with a simpler interface to configure vhosts, apache modules, and further to ensure that the necessary packages are installed and the service is running. When using this module you could have a vhost defined like this:

```
apache::vhost { 'www.example.com':
  vhost_name      => '192.126.100.1',
  port            => '80',
  docroot         => '/home/www.example.com/docroot/',
  logroot         => '/srv/www.example.com/logroot/',
  serveradmin     => 'webmaster@example.com',
  serveraliases   => ['example.com',],
}
```

The `apache::vhost` provides all the options that you could tune, and you set them as needed. You don't need to have to touch the underlying templates used, or know the

[4] http://forge.puppetlabs.com/puppetlabs/apache

syntax of apache configuration, or really anything about how the module works, aside from the options presented by the vhost.

Fundamentally, the apache module does one thing, and does one thing well. It doesn't handle things like monitoring, backups, and it doesn't try to run back end services. You can use this module to run apache, and combine it with other modules to build the rest of your configuration.

The Weird Blocks and Code Layout

Of course, every site has their own internal services and applications, and this is where the weird blocks come in. Weird blocks are analogous to the Lego blocks that have axles or hinges sticking out: they're designed to do something very specific and can't really be reused anywhere else. In turn, nothing else can provide the behavior that they provide.

Generally, these generally should be written like base blocks but with a couple of twists. One twist is that since these modules cannot be reused elsewhere, it can make sense to embed site specific data in templates and manifests. Secondly, these modules are located in a different place on the filesystem. Using the Puppet `modulepath` setting or chef `cookbook_path` setting, you can specify a list of locations to check for modules. You can take advantage of this to locate reusable base blocks in one place, and weird blocks in another place.

```
├── base-blocks
|   └── apache
|       ├── manifests
|       |   ├── init.pp
|       |   ├── ssl.pp
|       |   └── vhost.pp
|       └── templates
|
├── weird-blocks
|   └── boardie
|       ├── manifests
|       |   └── init.pp
|       └── templates
|           └── config.yml.erb
```

Differentiating between base blocks and weird blocks is surprisingly powerful. The distinction makes publishing your base-blocks easier, and allows you to easily tell what sort of work a module is expected to do.

This separation can also be used to control access - perhaps one team manages an internal service, so they can handle the configuration management for that service. However this team won't be administering the rest of your infrastructure. Giving them access to the weird-blocks directory means they'll be able to do their job, but they'll be bound to respecting the interfaces of the base-blocks instead of taking shortcuts and putting site specific changes in your base blocks.

Composing Blocks into Services (like Lego kits)

So we have all of these well defined modules and classes, but without assembling them you have a pile of Lego - something that's not useful and mainly exists to cause searing pain when you step on one. Therefore, we need some sort of concept, like a site configuration, where you take these individual parts and snap them into configurations that work for you.

Building on top of the multiple module-path idea outline, assembled modules go in a site-services directory, like so:

```
├── site-services
|   └── infrastructure
|       └── manifests
|           ├── dhcp.pp
|           ├── mrepo.pp
|           ├── webserver.pp
|           └── postgresql.pp
```

Within this site-services directory, you build out modules that provide a complete solution. For instance, the `infrastructure::postgresql` module would do things like use the postgresql module to install and run the postgres service, use the nagios module for monitoring postgresql, use the backupexec module to back it up, and so forth. In addition, this is where you inject the site-specific configuration into the modules, so this is where you make the underlying modules work for your infrastructure.

Things in site-services generally won't directly include resources and will only include other classes. Put another way, they exist almost entirely to aggregate classes into usable units and configure their settings. The following example would be an example of everything you would need to bring up the mrepo[5] infrastructure on a node:

[5] http://dag.wieers.com/home-made/mrepo/

```
class infrastructure::mrepo {

  motd::register {'mrepo': }

  class { 'staging':
    path  => '/opt/staging',
    owner => 'root',
    group => 'root',
    mode  => '0755',
  }

  $mirror_root = '/srv/mrepo'

  class { 'mrepo::params':
    src_root     => $mirror_root,
    www_root     => "${mirror_root}/www",
    user         => "root",
    group        => "root",
  }

  class { 'mrepo::exports':
    clients => '192.168.100.0/23',
  }

  # Bring in a list of the actual repositories to instantiate
  include infrastructure::mrepo::centos
}
```

Using this model anyone can use the mrepo module, and our own implementation can be used with `include infrastructure::mrepo`. We have a clear separation of the mrepo implementation and how we're using it.

Roles: They're Like Lego Cities

At this point, we have the modules built in site-services that configure our environment the way we need it. The final step is taking these services and grouping them into configurations that we'll apply to machines. For instance, bringing up a new webserver could involve including modules from site-services to set up our configurations SSH, Apache, and Postgres. Bringing up a new host for building packages would mean bringing in our site-specific configurations for Tomcat, Jenkins, and compilers and such. This would give us a hierarchy like this:

```
├── site-roles
|   ├── buildhosts
|   |   └── manifests
|   |       ├── init.pp
|   |       ├── jenkins.pp
|   |       └── compilers.pp
|   |
|   └── webservices
|       ├── manifests
|       |   ├── redmine.pp
|       |   └── wordpress.pp
```

Each manifest in here would be a further abstraction on top of the site-services module. They would look something like this:

```
class webservices::redmine {

  include infrastructure::apache::passenger
  include infrastructure::mysql

  class { 'custom_redmine':
    vhost_name    => $fqdn,
    serveraliases => "redmine.${domain} redmine-${hostname}.$
{domain}",
    www_root      => '/srv/passenger/redmine',
  }

  pam::allowgroup { 'redmine-devs': }
  pam::allowgroup { 'redmine-admins': }

  sudo::allowgroup { 'redmine-admins': }
}
```

This final layer takes all our implementations of apache and mysql and applies them, controls system access, and provides for a complete redmine stack. Including this one class, `webservicse::redmine`, is all it takes to provide for every requirement of a redmine instance, so deploying more machines for a specific role means including a single self contained class.

This gives us the following hierarchy:

- base-blocks and weird-blocks provide basic functionality
- site-services assemble blocks into functional services
- site-roles assemble services into fully functional and independent roles

If you use this pattern, in no time, you could have configuration management code that is about as awesome as a seven foot replica of Serenity.

(image credit - brickfrenzy[6])

Article Source:
http://sysadvent.blogspot.com/2012/12/day-13-configuration-management-as-legos.html

[6] http://www.flickr.com/photos/brickfrenzy/

Certificate of Completion

________________________________ attended ___ hours of

Puppet Labs training and completed the

Introduction to Puppet Enterprise course.

____________________________ __________________

(Instructor Signature) (Date)