```
## ADMIN:   Puppet lecture at Lyon University
## Date:    19/05/2017

# Class: install puppet packges
#
class { 'install::config_management_system':
        evaluate::packages {['config_management_system']:}
        package {
                ['puppet','hiera','facter','puppetdb']:
                        ensure   => installed,
                        version  => '4.10';
                ['chef','ansible']:
                        ensure   => absent;
        }
}
```

puppet

Lyon University

19/05/2017

https://docs.puppet.com/puppet_core_types_cheatsheet.pdf

Christopher Jon Lee

CERN ADC – Central Services Coordinator

University of Cape Town, South Africa

Who am I ?

- Christopher Jon Lee
- XPLOD Systems
  - Diploma in Business Computing (1997)
  - MSCE +I (2000)
  - Computer hardware and software for small companies
- CERN (2012)
  - University of Johannesburg (2012/15)
    - Systems Administrator for the ZA-UJ Tier 3 grid cluster
  - ATLAS – Trigger & Data Acquisition (TDAQ) (2012/17)
    - Systems Administrator High Level Trigger Farm: ~4000 machines
  - University of Cape Town (2015/17)
    - Adjuct Research Office
  - ATLAS – Central Service (ADC) (2017)
    - Coordinator of all ATLAS Offline Central Services

What do I do

- Systems Administration
- Configuring, deploying and maintaining server machines
- Hardware
  - 4 different OEMs
  - Switches
  - HDD vendors / types
  - Virtual Machines
- Software
  - 3 different Operating Systems
  - Custom made RPM's
- Services
  - Apache
  - SSH
  - DNS
  - DHCP
  - NTP
  - MySQL/Maria/PSQL

How do we maintain all of that?

- Bash Scripts
- KickStart files

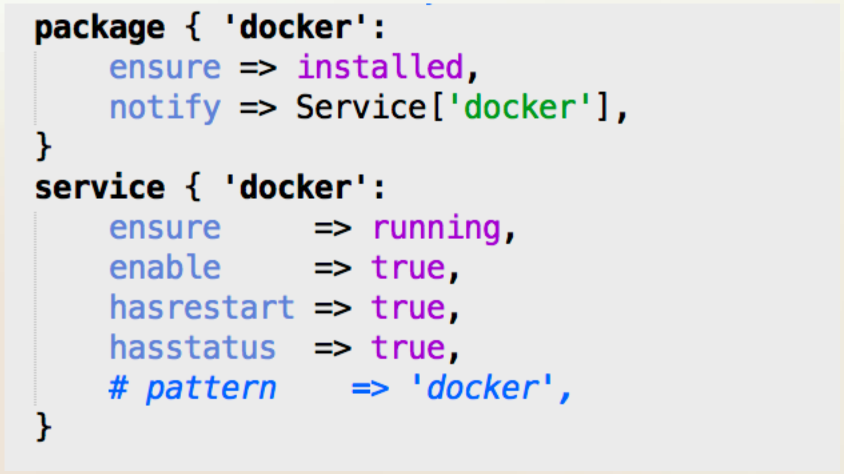
- Time required
- Stability
- Consistency
- Accountability

Puppet

- Puppet is a configuration management tool that is extremely powerful in deploying, configuring, managing, maintaining, a server machine
- https://vimeo.com/57402168:

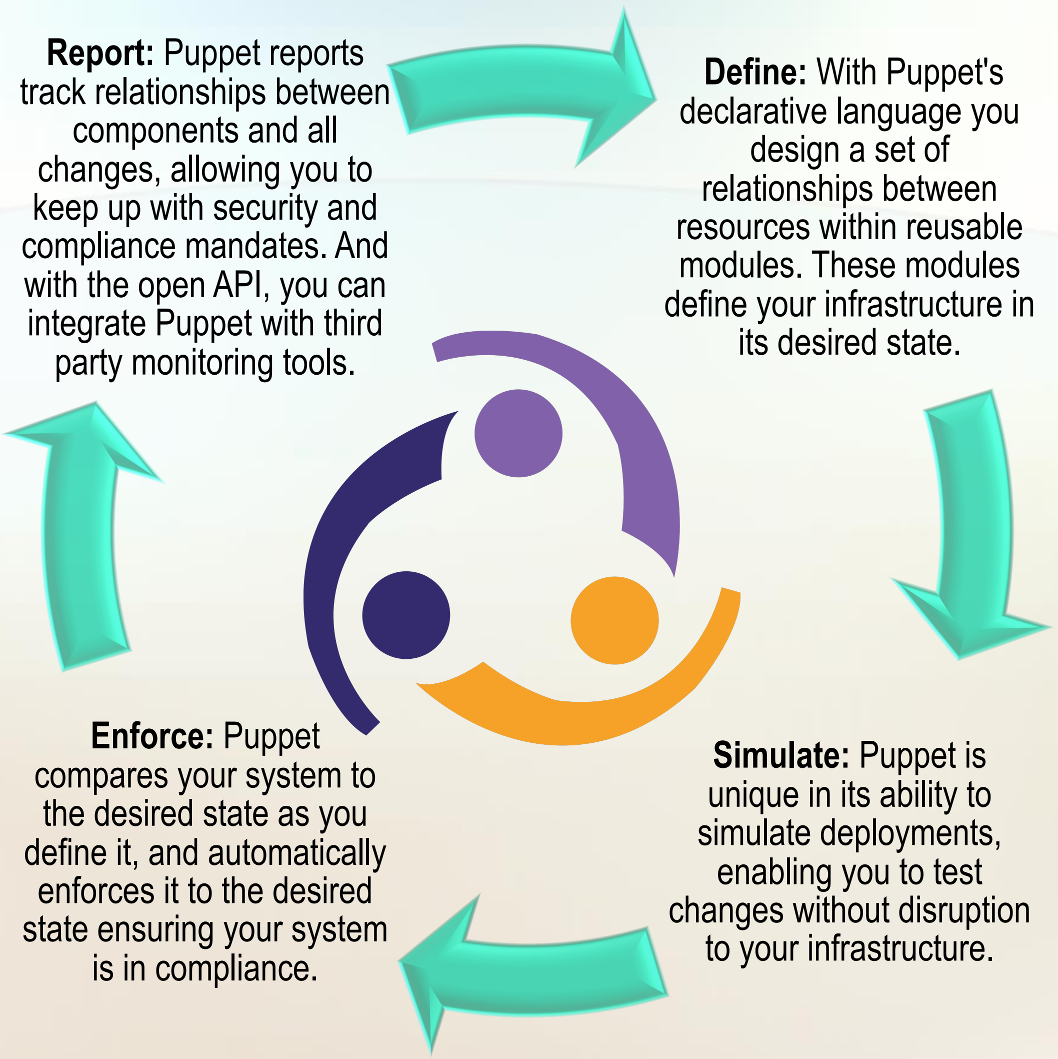
Why Puppet?

- Declarative language
- Define good KNOWN machine **STATE**
- Saves Time
- Documentation
- Handle Multiple OS



```
package { 'docker':
    ensure => installed,
    notify => Service['docker'],
}
service { 'docker':
    ensure     => running,
    enable     => true,
    hasrestart => true,
    hasstatus  => true,
    # pattern    => 'docker',
}
```

How Puppet Works



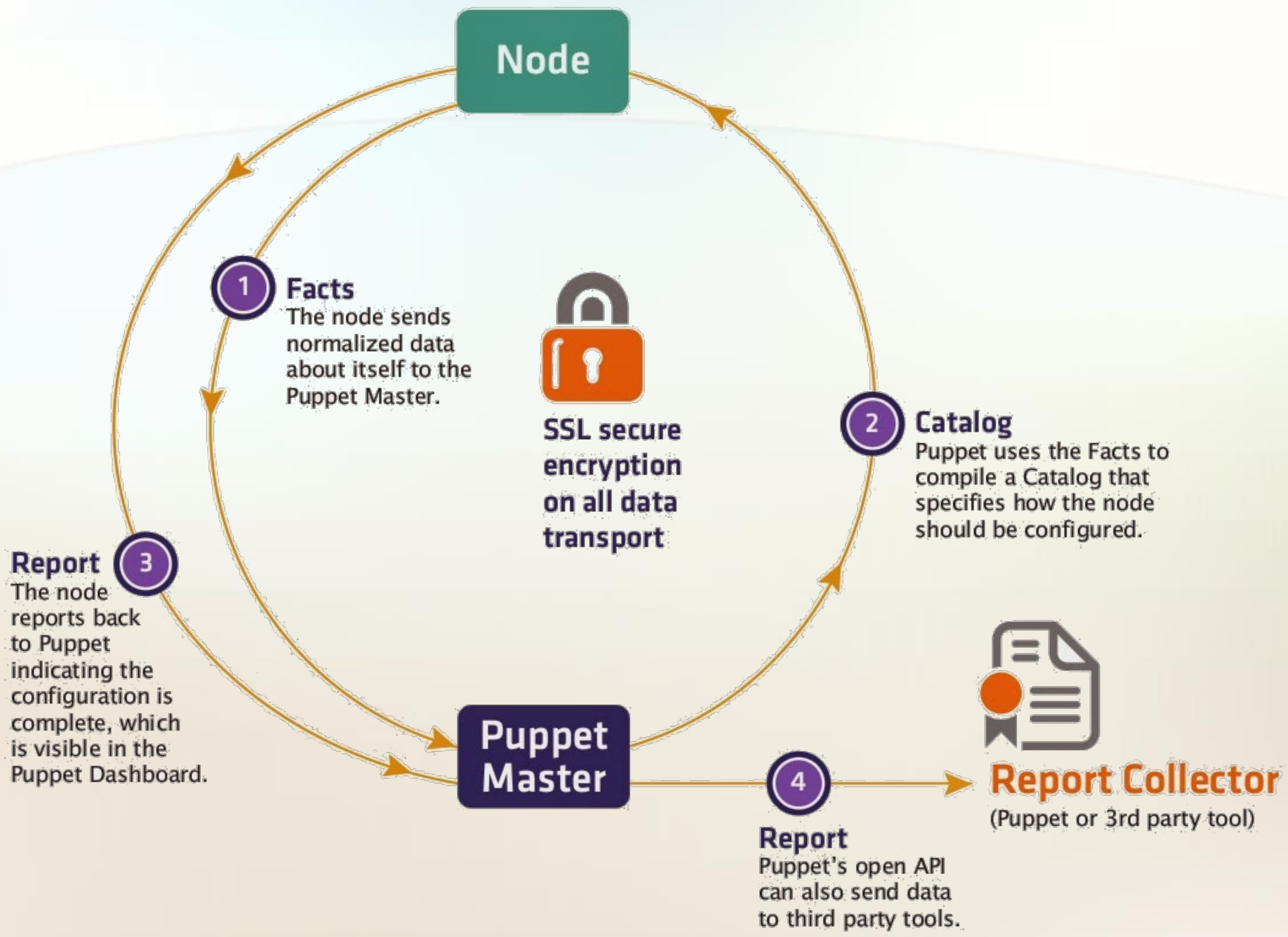
Report: Puppet reports track relationships between components and all changes, allowing you to keep up with security and compliance mandates. And with the open API, you can integrate Puppet with third party monitoring tools.

Define: With Puppet's declarative language you design a set of relationships between resources within reusable modules. These modules define your infrastructure in its desired state.

Simulate: Puppet is unique in its ability to simulate deployments, enabling you to test changes without disruption to your infrastructure.

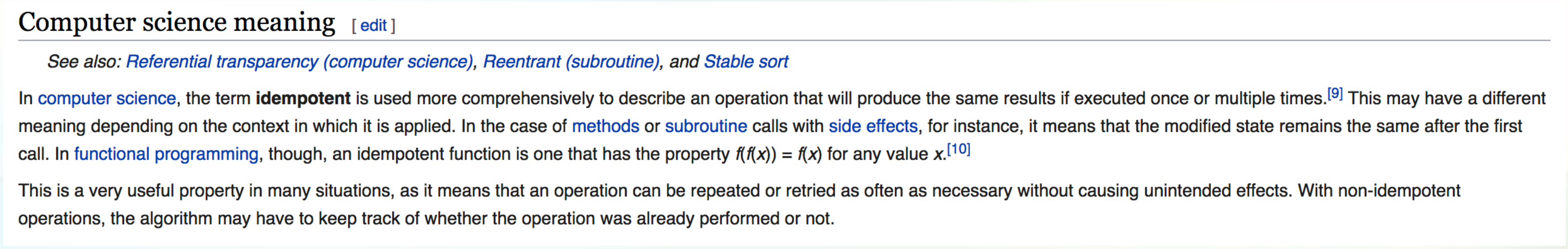
Enforce: Puppet compares your system to the desired state as you define it, and automatically enforces it to the desired state ensuring your system is in compliance.

How Puppet Manages Data Flow for Individual Nodes


Node
1 Facts
The node sends normalized data about itself to the Puppet Master.
SSL secure encryption on all data transport
2 Catalog
Puppet uses the Facts to compile a Catalog that specifies how the node should be configured.
Report 3
The node reports back to Puppet indicating the configuration is complete, which is visible in the Puppet Dashboard.
Puppet Master
4
Report
Puppet's open API can also send data to third party tools.
Report Collector
(Puppet or 3rd party tool)

Idempotent

https://en.wikipedia.org/wiki/Idempotence



Computer science meaning [edit]

See also: Referential transparency (computer science), Reentrant (subroutine), and Stable sort

In computer science, the term **idempotent** is used more comprehensively to describe an operation that will produce the same results if executed once or multiple times.[9] This may have a different meaning depending on the context in which it is applied. In the case of methods or subroutine calls with side effects, for instance, it means that the modified state remains the same after the first call. In functional programming, though, an idempotent function is one that has the property $f(f(x)) = f(x)$ for any value x.[10]

This is a very useful property in many situations, as it means that an operation can be repeated or retried as often as necessary without causing unintended effects. With non-idempotent operations, the algorithm may have to keep track of whether the operation was already performed or not.

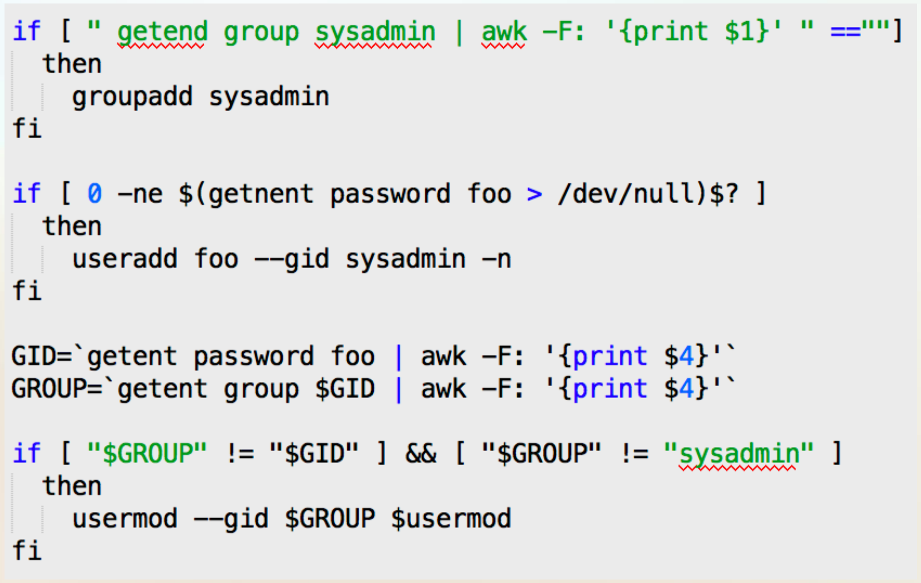
- Puppet only changes resources or attributes that are out of sync
- Produces the same end result, no matter how many times puppet is run.
- Describes the **FINAL state**, rather than a series of steps to follow

Running puppet

- Puppet can run in two different ways
- Apply: https://docs.puppet.com/puppet/4.10/services_apply.html
- Agent: https://docs.puppet.com/puppet/4.10/services_agent_unix.html
- Differences:
- Principle of least privilege
- Ease of centralised reporting and inventory
- Ease of updating configurations
- CPU and memory usage on managed machines
- Need for a dedicated master server
- Need for good network connectivity
- Security overhead

Declarative Language (DSL)

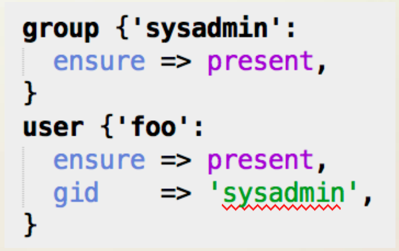
- The user models the desired state
- Puppet figures out how to enforce it.



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

if [ 0 -ne $(getnent password foo > /dev/null)$? ]
  then
    useradd foo --gid sysadmin -n
fi

GID=`getent password foo | awk -F: '{print $4}'`
GROUP=`getent group $GID | awk -F: '{print $4}'`

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

BASH

Puppet



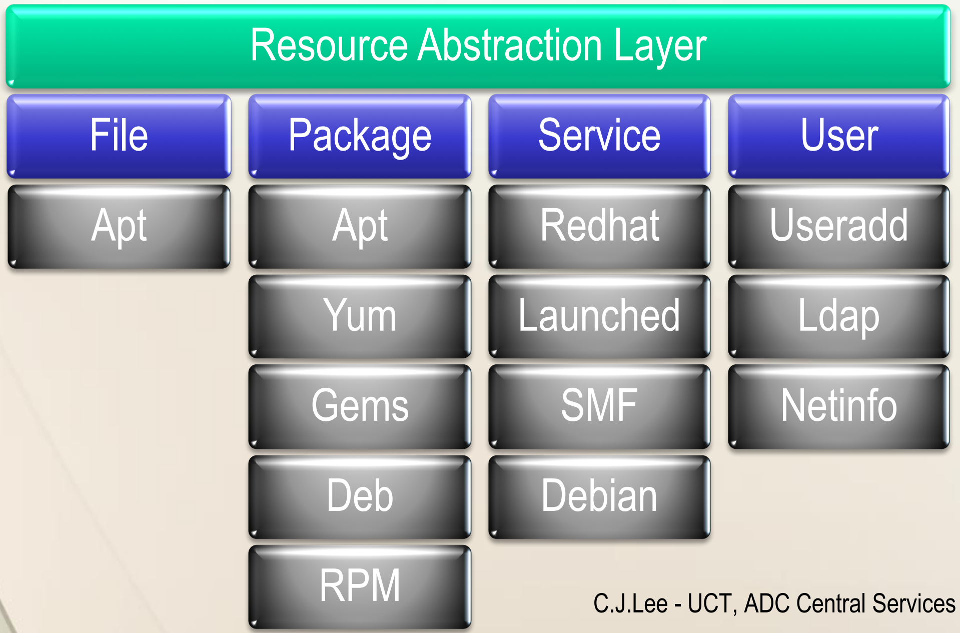
```
group {'sysadmin':
  ensure => present,
}
user {'foo':
  ensure => present,
  gid    => 'sysadmin',
}
```

*Spelling error in bash code ;)

Puppet Resources

- https://docs.puppet.com/puppet/4.10/lang_resources.html
- Resources are building blocks
  - Resource Abstraction layer
  - Providers
- They can be combined to create larger components
- They model the **expected state** of a your system
- *# puppet resource --types*

```
file { '/build':
    ensure => directory,
    owner  => 'root',
    group  => 'root',
    mode   => '1777',
}
```

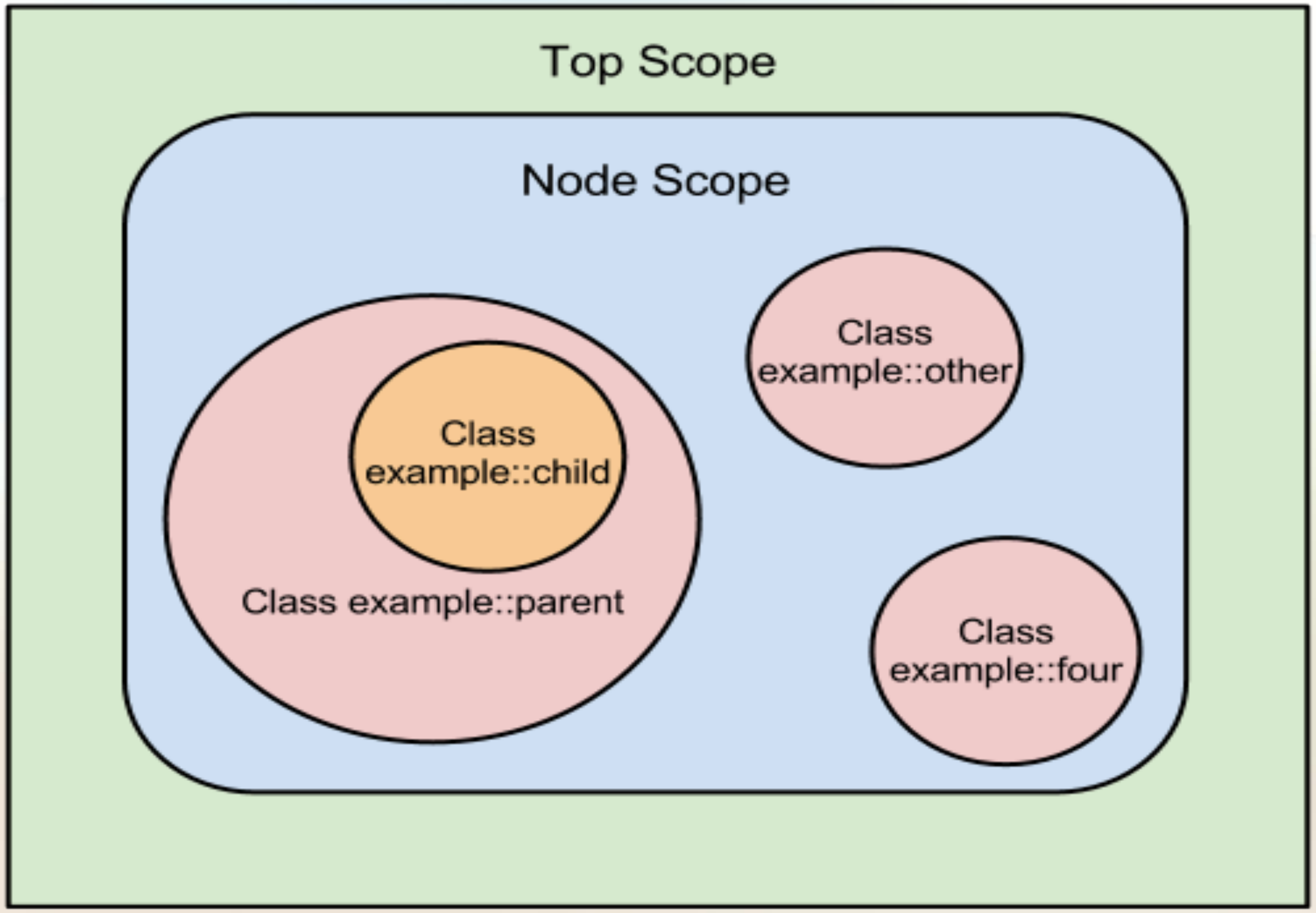


State Configuration

ONLY configures the desired state!

and **ENFORCES** it

Scope



https://docs.puppet.com/puppet/4.10/lang_scope.html

Variables

- Variables store values so they can be accessed later.
- In the Puppet language, variables are actually constants, since they can't be reassigned. But since "variable" is more comfortable and familiar to most people, the name has stuck.
- Puppet only allows a given variable to be assigned once within a given scope.
- Assigning variables
  - `$content = "some content\n"`
- Arrays
  - `[$a, $b, $c] = [1,2,3] # $a = 1, $b = 2, $c = 3`
- Hashes
  - `[$a, $b] = {a => 10, b => 20} # $a = 10, $b = 20`
- Accessing out-of-scope variables
  - You can access out-of-scope variables from named scopes by using their qualified names: `$vhostdir = $apache::params::vhostdir`
  - Note that the top scope's name is the empty string — thus, the qualified name of a top scope variable would be, e.g., `$::osfamily`.

Variables examples

```
class system {
  $operatingsystem = "MyOS"
  notify { "The operating system is : ${operatingsystem}": }
}
```

```
Notice: The operating system is MyOS
```

```
class truesystem {
  $operatingsystem = "MyOS"
  notify { "The operating system is : ${::operatingsystem}": }
}
```

```
Notice: The operating system is Darwin
```

```
[chlee@Christophers-MacBook-Pro ~]# facter operatingsystem
Darwin
```

- Before requesting a catalog (or compiling one with puppet apply), Puppet will collect system information with Facter. Puppet receives this information as facts, which are pre-set variables you can use anywhere in your manifests.
- Facts are sent to the master
- Facts are the only thing the master knows about the agent
- Exposed as global variables (top scope) during the catalog compilation

- The custom facts walkthrough explains in detail how to write and distribute your own custom or external facts.

Directory structure

```
[root@master spec]# pwd
/etc/puppetlabs/code/environments/production/modules/azure
[root@master spec]# tree -aL 1
.
├── .fixtures.yml        # Dependencies config for unit testing
├── .gitignore           # List of files for git to ignore during commits
├── .rspec           # Rspec configuration
├── .rspec_parallel      # Rspec parallel configuration
├── .rubocop.yml         # Rubocop configuration
├── .travis.yml          # Travis configuration
├── CHANGELOG.md         # Markdown file of changes
├── CONTRIBUTING.md      # Markdown file of how to contribute
├── Gemfile          # List of gem dependencies for testing
├── Guardfile            # Configuration file for Guard
├── LICENSE          # Module license
├── README.md            # Readme
├── Rakefile             # Unit testing configuration
├── data                 # Data in code (Hiera 5) directory
├── examples             # Examples directory
├── hiera.yaml           # Data in code (Hiera 5) configuration file
├── lib              # Lib directory
├── manifests            # Manifests directory
├── metadata.json        # Metadata for Puppet Forge
├── rakelib          # Extensions for unit testing
├── spec                 # Spec directory
└── templates            # Template directory
```

https://puppet.com/blog/magic-directories-guide-to-puppet-directory-structure
https://docs.puppet.com/puppet/4.10/modules_fundamentals.html

Puppet "files"

- site.pp
- init.pp
- Puppet supports two templating languages:
  - Embedded Puppet (EPP) uses Puppet expressions in special tags. It's easy for any Puppet user to read, but only works with newer Puppet versions. (≥ 4.0, or late 3.x versions with future parser enabled.)
  - Embedded Ruby (ERB) uses Ruby code in tags. You need to know a small bit of Ruby to read it, but it works with all Puppet versions.

Classes

- The core of the Puppet language is declaring resources.
- Every other part of the language exists to add flexibility and convenience to the way resources are declared.
- Groups of resources can be organized into classes, which are larger units of configuration.
- While a resource might describe a single file or package, a class can describe everything needed to configure an entire service or application (including any number of packages, config files, service daemons, and maintenance tasks).
- Smaller classes can then be combined into larger classes which describe entire custom system roles, such as “database server” or “web application worker.”

Class Example

```
class ntp {
 case $::operatingsystem {
  centos, redhat:  { $service_name = 'ntpd' }
  debian, ubuntu: { $service_name = 'ntp' }
 }
 package { 'ntp':
  ensure => installed,
 }
 service { 'ntp':
  name => $service_name,
  ensure => running,
  enable => true,
  subscribe => File['ntp.conf'],
 }
 file { 'ntp.conf':
  path => '/etc/ntp.conf',
  ensure => file,
  require => Package['ntp'],
  source => "puppet:///modules/ntp/ntp.conf",
  # This source file would be located on the Puppet master at
  # /etc/puppetlabs/code/modules/ntp/files/ntp.conf }
}
```

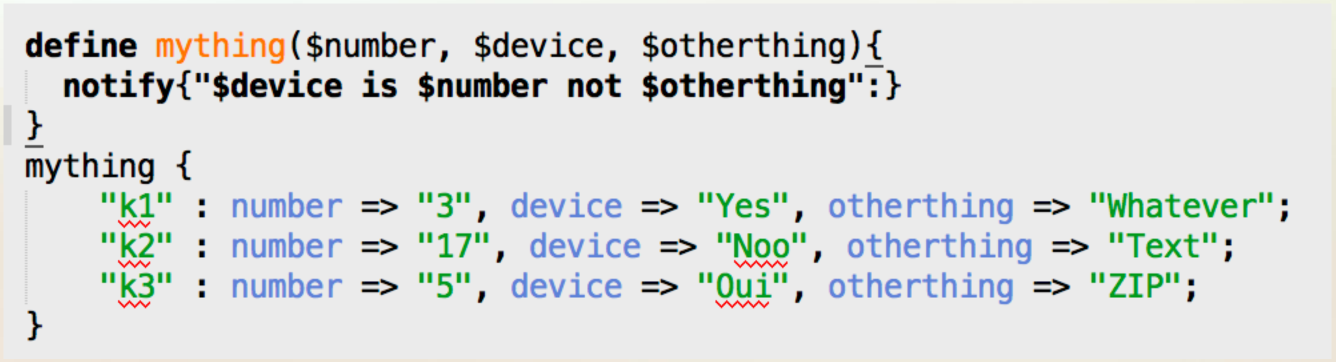
Nodes

- A node definition or node statement is a block of Puppet code that will only be included in matching nodes' catalogs.
- Node statements only match nodes by name. By default, the name of a node is its certname.

```
# /etc/puppetlabs/puppet/manifests/site.pp
node 'www1.example.com' {
  include common
  include apache
  include squid
}
node 'db1.example.com' {
  include common
  include mysql
}
```

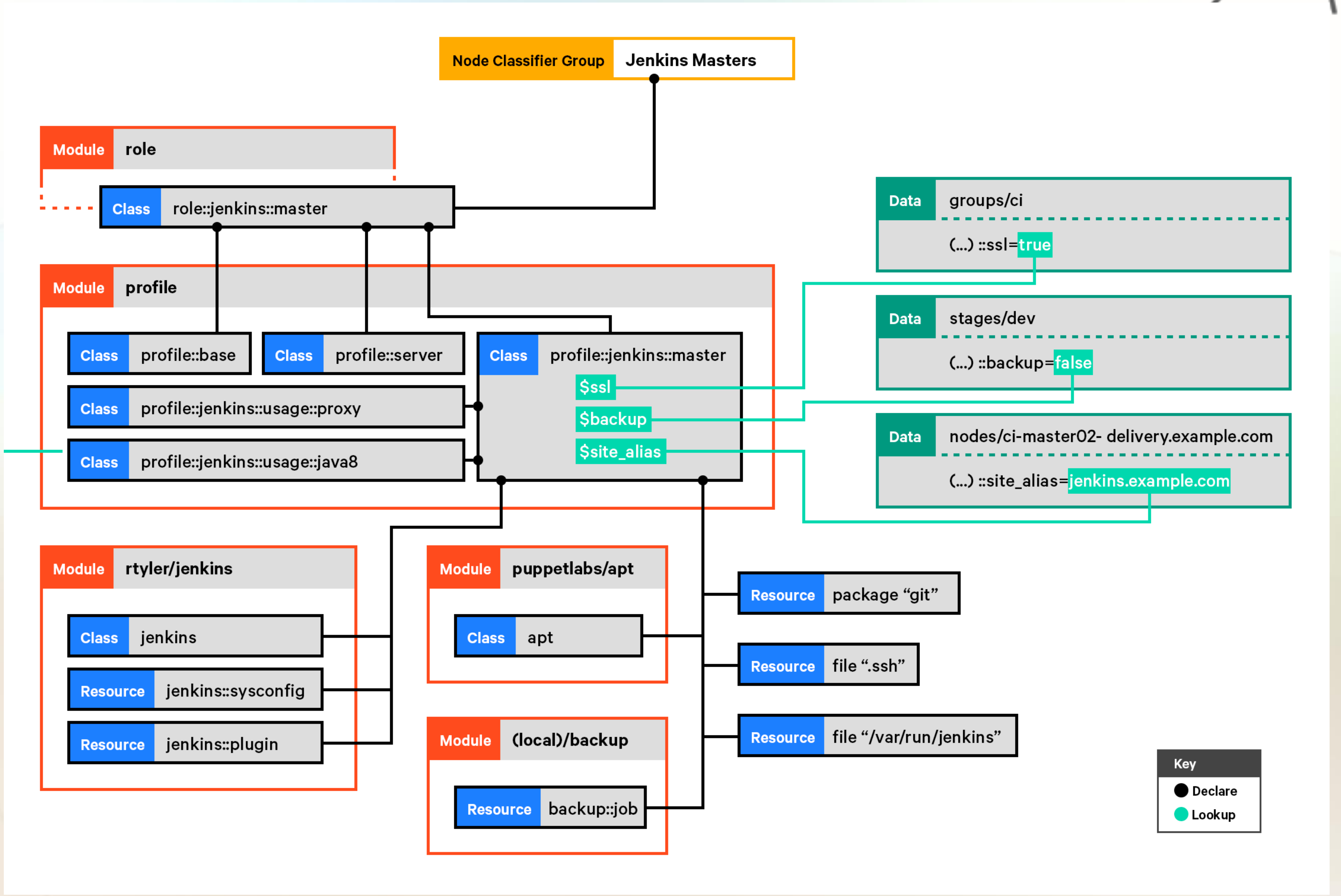
Defines

- Defined resource types (also called defined types or defines) are blocks of Puppet code that can be evaluated multiple times with different parameters.
- Once defined, they act like a new resource type: you can cause the block to be evaluated by declaring a resource of that new resource type.
- Defines can be used as simple macros or as a lightweight way to develop fairly sophisticated resource types.



```
define mything($number, $device, $otherthing){
  notify{"$device is $number not $otherthing":}
}
mything {
    "k1" : number => "3", device => "Yes", otherthing => "Whatever";
    "k2" : number => "17", device => "Noo", otherthing => "Text";
    "k3" : number => "5", device => "Oui", otherthing => "ZIP";
}
```

Roles and Profiles

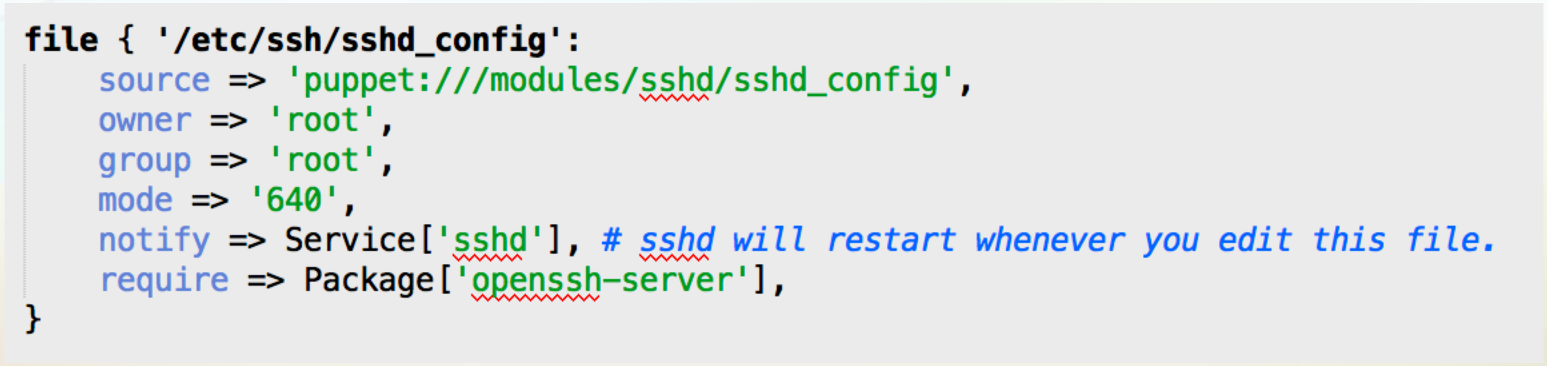


https://docs.puppet.com/pe/2017.1/r_n_p_intro.html

Manipulation of files: Direct

```
file { "/var/tmp/testfile":
    ensure  => "present",
    owner   => "root",
    group   => "root",
    mode    => "664",
    content => "This is a test file created using puppet.
                Puppet is really cool",
}
```

Manipulation of files: fileserver



```
file { '/etc/ssh/sshd_config':
    source => 'puppet:///modules/sshd/sshd_config',
    owner => 'root',
    group => 'root',
    mode => '640',
    notify => Service['sshd'], # sshd will restart whenever you edit this file.
    require => Package['openssh-server'],
}
```

Relationships: Metaparameters

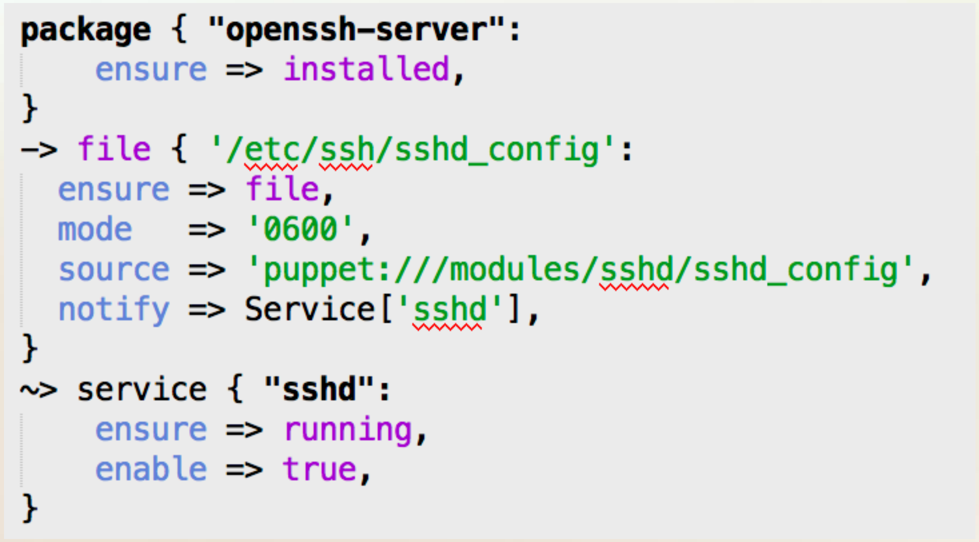
- Puppet uses four metaparameters to establish relationships, and you can set each of them as an attribute in any resource.
- The value of any relationship metaparameter should be a resource reference (or array of references).

- **before** — Applies a resource before the target resource.
- **require** — Applies a resource after the target resource.
- **notify** — Applies a resource before the target resource.
  - The target resource refreshes if the notifying resource changes.
- **subscribe** — Applies a resource after the target resource.
  - The subscribing resource refreshes if the target resource changes.

- https://docs.puppet.com/puppet/4.10/lang_relationships.html

Relationships: Chaining

- You can create relationships between two resources or groups of resources using the **->** and **~>** operators.
- **->** (ordering arrow; a hyphen and a greater-than sign)
  - Applies the resource on the left before the resource on the right.
- **~>** (notifying arrow; a tilde and a greater-than sign)
  - Applies the resource on the left first. If the left-hand resource changes, the right-hand resource will refresh



```
package { "openssh-server":
    ensure => installed,
}
-> file { '/etc/ssh/sshd_config':
  ensure => file,
  mode   => '0600',
  source => 'puppet:///modules/sshd/sshd_config',
  notify => Service['sshd'],
}
~> service { "sshd":
    ensure => running,
    enable => true,
}
```

Single source of truth: Hiera

- Hiera is a key/value lookup tool for configuration data, built to make Puppet better and let you set node-specific data without repeating yourself.
- Keeps site-specific data out of your manifests.
- Avoids repetition
- Hiera uses a configurable hierarchy
  - static data sources (with names like “common”)
  - dynamic ones (which can switch between data sources)

Hiera

```
## hiera.conf
---
:backends:
  - yaml
:yaml:
  :datadir: "/etc/puppet/hieradata/"
:hierarchy:
  - "1.nodes/%{::hostname}"
  - "2.os/%{::slv}"
  - "3.nt/%{nt::type}"
  - "4.hw/%{boardproductname}"
  - "6.env/%{environment}"
  - "8.site/%{::SITE}"
  - "common"
```

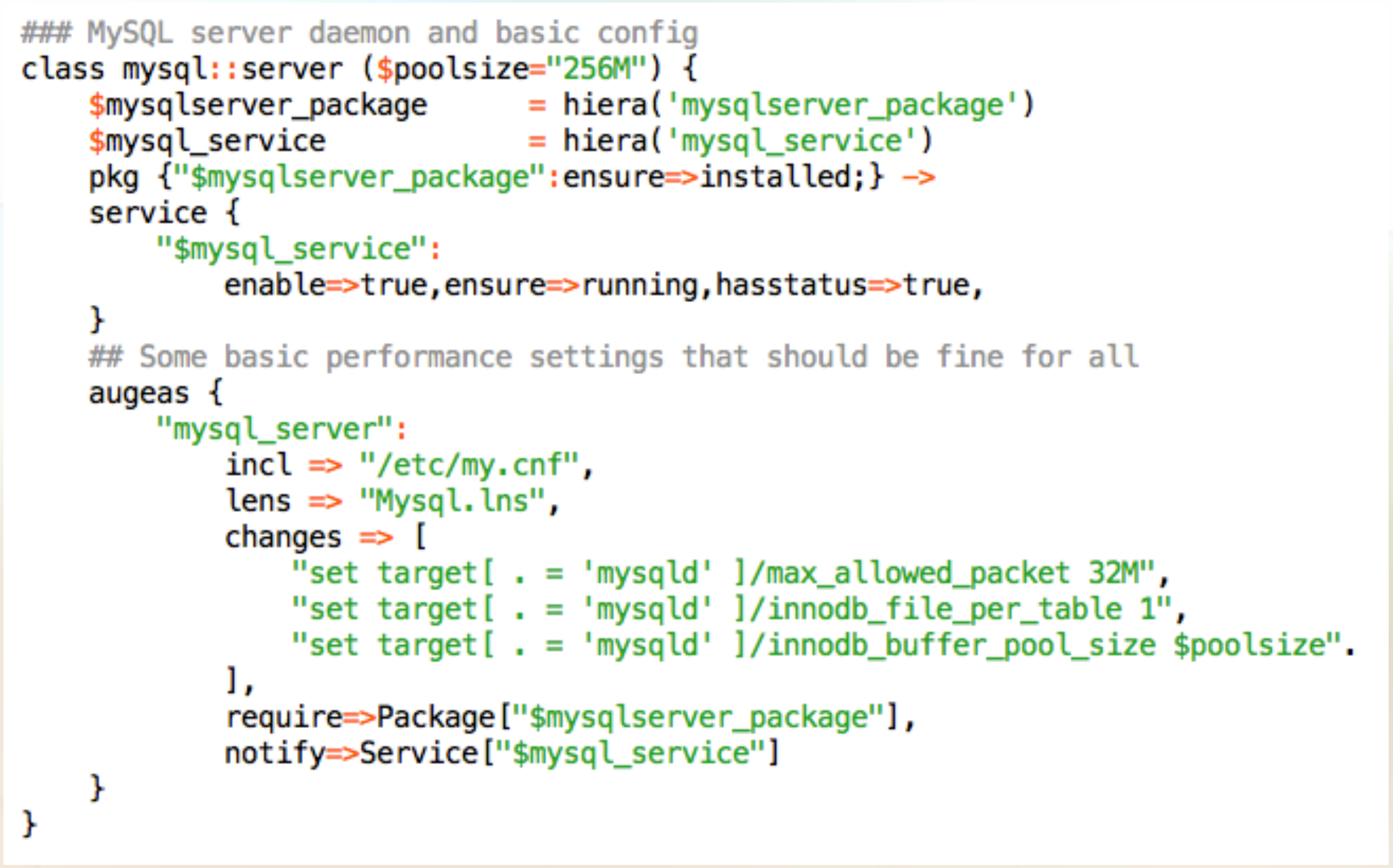
```
class web::shibboleth ($urlname) {
    $shibboleth_packages      = hiera('shibboleth_packages','')

    case $slv {
        5: { package {
                'shibboleth': ensure => present;
                'shibboleth-selinux': ensure => present;
                'log4shib': ensure => present;
                'mod_authz_ldap': ensure => present;
                'mod_auth_kerb': ensure => present;
                }
        }
        6: { package {[
                 'shibboleth',
                 'shibboleth-selinux',
                 'liblog4shib1',
                 'xmltooling-schemas',
                 'opensaml-schemas',
                 ]:ensure=>present; }
                }
        7: { package {$shibboleth_packages:ensure=>present }}
    }
}
```

PuppetDB

- PuppetDB stores:
  - The most recent facts from every node
  - The most recent catalog for every node
- Optionally, 14 days (configurable) of event reports for every node
- Together, these give you a huge inventory of metadata about every node in your infrastructure and a searchable database of every single resource being managed on any node.
- Puppet itself can search a subset of this data using exported resources, which allow nodes to manage resources on other nodes.
- The remaining data is available through PuppetDB's query APIs (see the navigation sidebar for details).

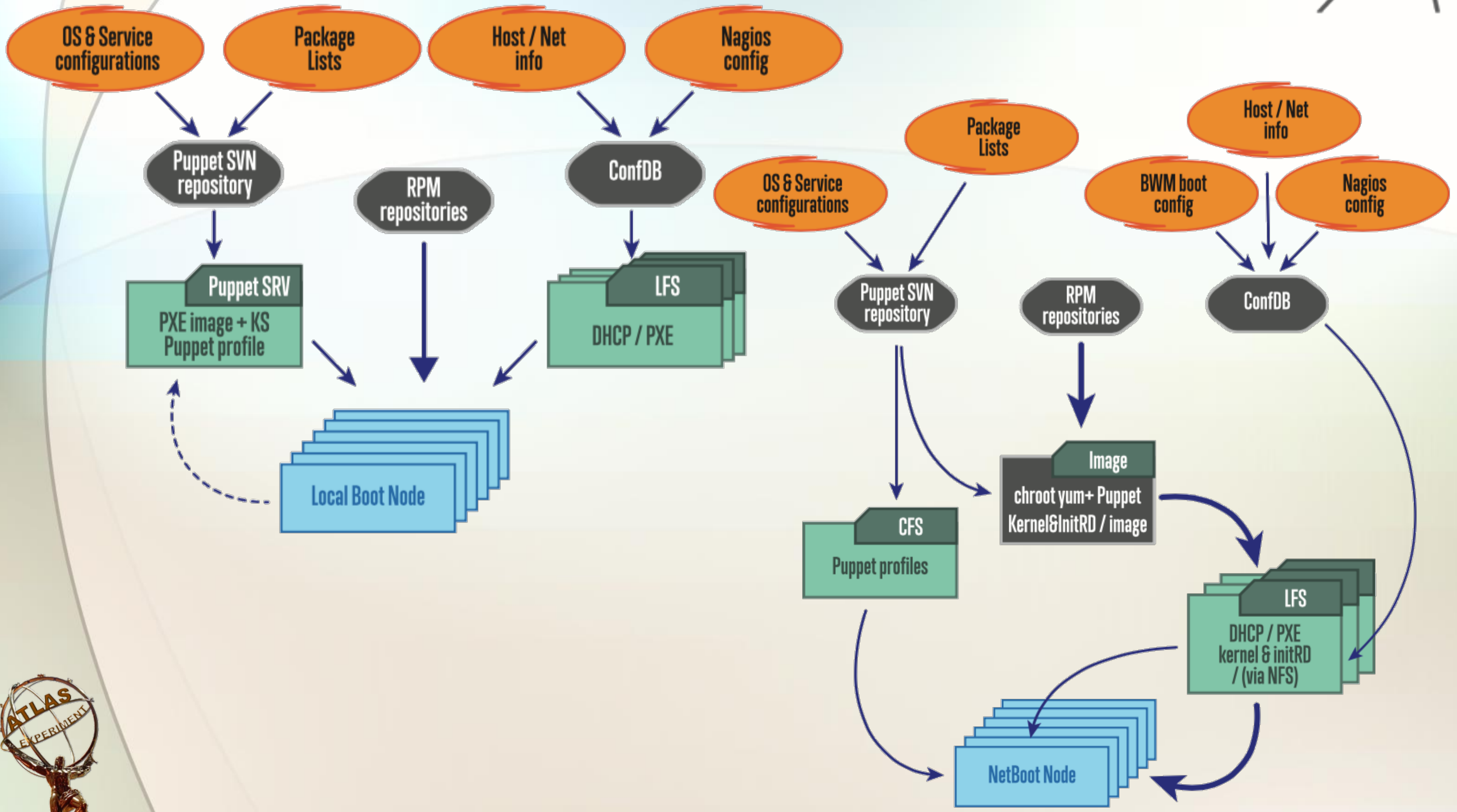
- https://docs.puppet.com/puppetdb/latest/index.html

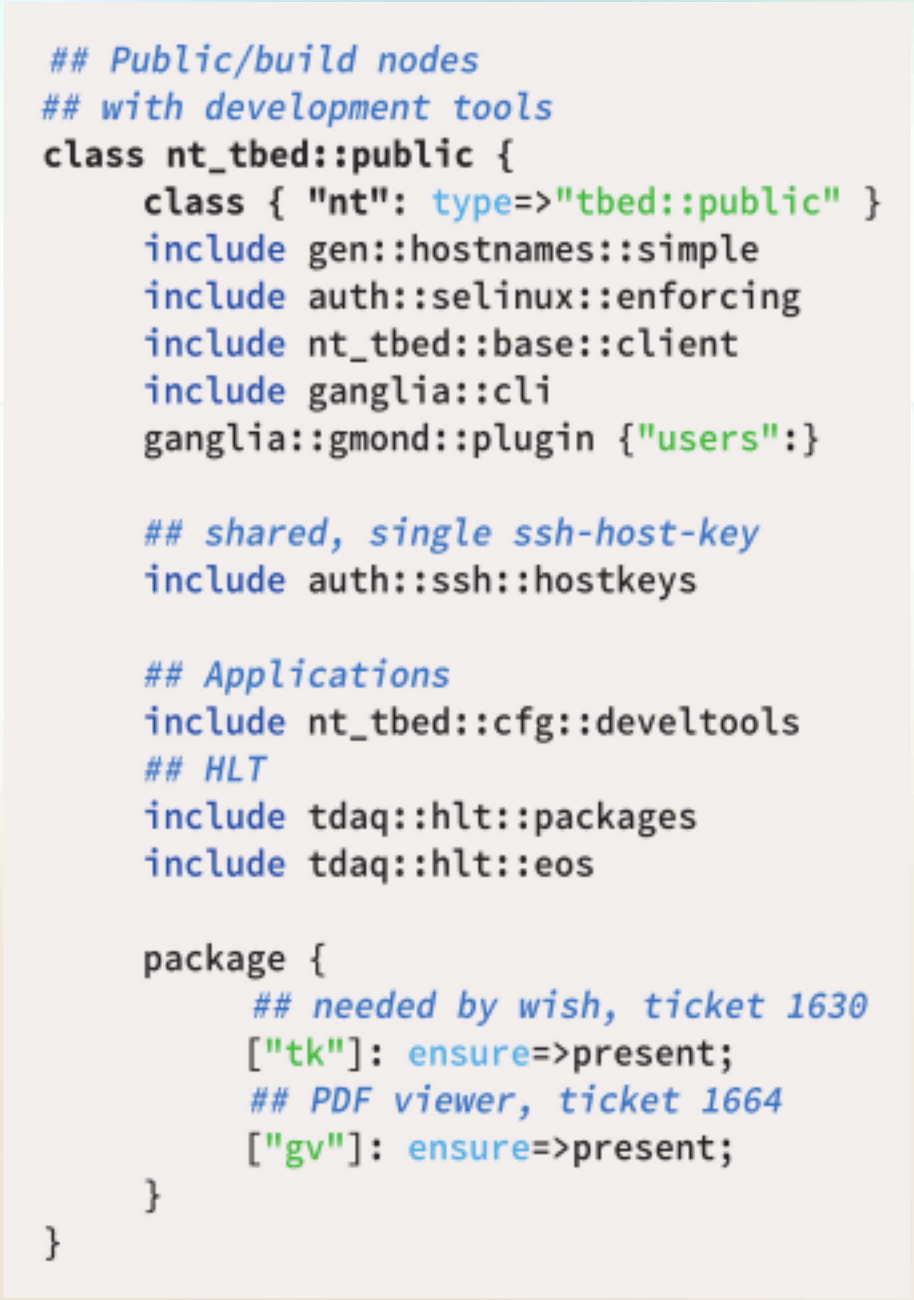
How we use puppet



```
### MySQL server daemon and basic config
class mysql::server ($poolsize="256M") {
    $mysqlserver_package      = hiera('mysqlserver_package')
    $mysql_service            = hiera('mysql_service')
    pkg {"$mysqlserver_package":ensure=>installed;} ->
    service {
        "$mysql_service":
            enable=>true,ensure=>running,hasstatus=>true,
    }
    ## Some basic performance settings that should be fine for all
    augeas {
        "mysql_server":
            incl => "/etc/my.cnf",
            lens => "Mysql.lns",
            changes => [
                "set target[ . = 'mysqld' ]/max_allowed_packet 32M",
                "set target[ . = 'mysqld' ]/innodb_file_per_table 1",
                "set target[ . = 'mysqld' ]/innodb_buffer_pool_size $poolsize".
            ],
            require=>Package["$mysqlserver_package"],
            notify=>Service["$mysql_service"]
    }
}
```

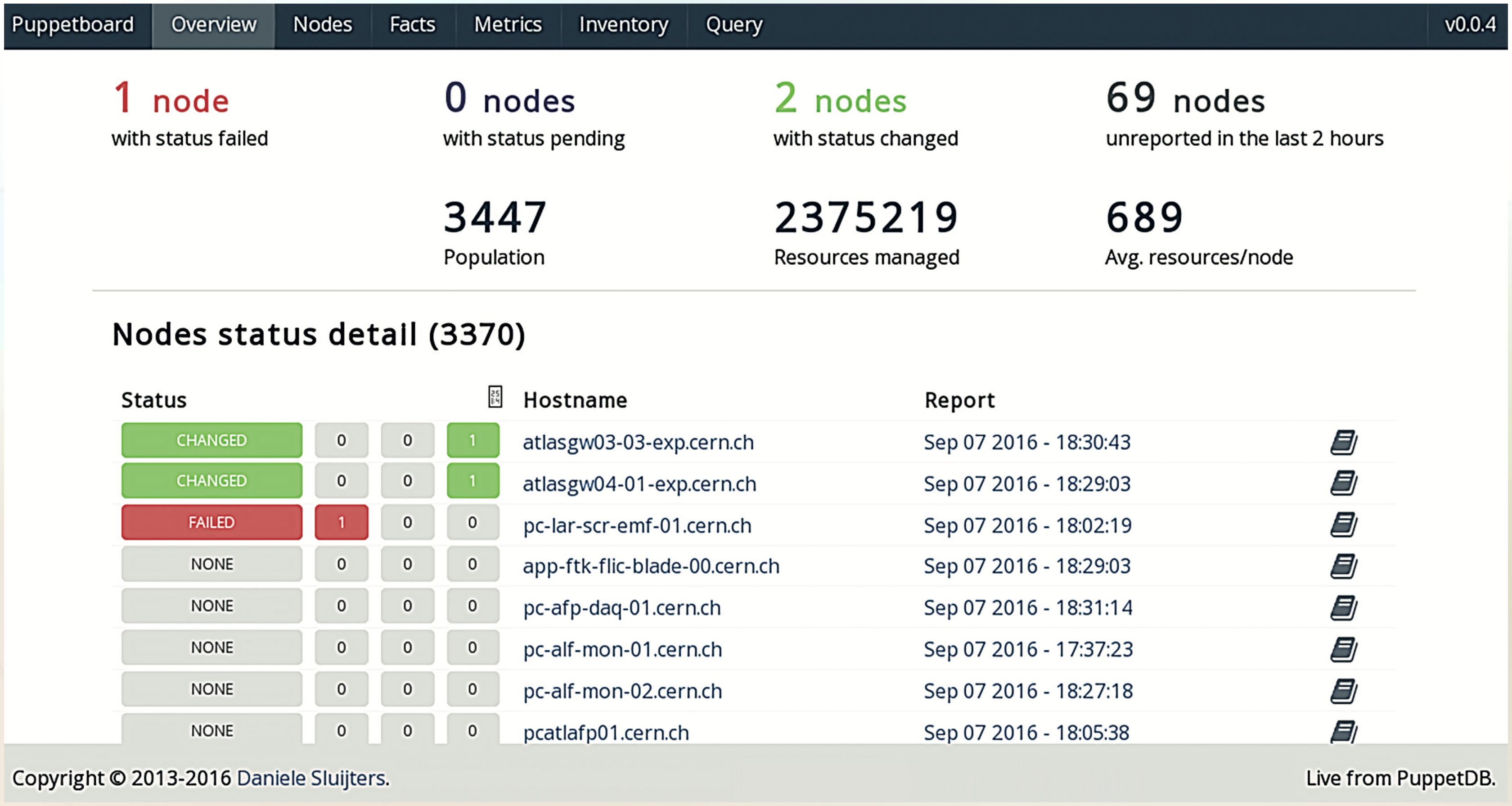
How we use puppet





```
## Public/build nodes
## with development tools
class nt_tbed::public {
    class { "nt": type=>"tbed::public" }
    include gen::hostnames::simple
    include auth::selinux::enforcing
    include nt_tbed::base::client
    include ganglia::cli
    ganglia::gmond::plugin {"users":}

    ## shared, single ssh-host-key
    include auth::ssh::hostkeys

    ## Applications
    include nt_tbed::cfg::develtools
    ## HLT
    include tdaq::hlt::packages
    include tdaq::hlt::eos

    package {
        ## needed by wish, ticket 1630
        ["tk"]: ensure=>present;
        ## PDF viewer, ticket 1664
        ["gv"]: ensure=>present;
    }
}
```

Monitoring



1 node with status failed

0 nodes with status pending

2 nodes with status changed

69 nodes unreported in the last 2 hours

3447 Population

2375219 Resources managed

689 Avg. resources/node

Nodes status detail (3370)

Status				Hostname	Report
CHANGED	0	0	1	atlasgw03-03-exp.cern.ch	Sep 07 2016 - 18:30:43
CHANGED	0	0	1	atlasgw04-01-exp.cern.ch	Sep 07 2016 - 18:29:03
FAILED	1	0	0	pc-lar-scr-emf-01.cern.ch	Sep 07 2016 - 18:02:19
NONE	0	0	0	app-ftk-flic-blade-00.cern.ch	Sep 07 2016 - 18:29:03
NONE	0	0	0	pc-afp-daq-01.cern.ch	Sep 07 2016 - 18:31:14
NONE	0	0	0	pc-alf-mon-01.cern.ch	Sep 07 2016 - 17:37:23
NONE	0	0	0	pc-alf-mon-02.cern.ch	Sep 07 2016 - 18:27:18
NONE	0	0	0	pcatlafp01.cern.ch	Sep 07 2016 - 18:05:38

Copyright © 2013-2016 Daniele Sluijters. Live from PuppetDB.

Questions

WE TOOK THE HOSTAGES, SECURED THE BUILDING, AND CUT THE COMMUNICATION LINES LIKE YOU SAID.
EXCELLENT.
BUT THEN THIS GUY CLIMBED UP THE VENTILATION DUCTS AND WALKED ACROSS BROKEN GLASS, KILLING ANYONE WE SENT TO STOP HIM.
AND HE RESCUED THE HOSTAGES?
NO, HE IGNORED THEM. HE JUST RECONNECTED THE CABLES WE CUT, MUTTERING SOMETHING ABOUT "UPTIME".
SHIT, WE'RE DEALING WITH A *SYSADMIN*.