ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

PUPPET - ΑΥΤΟΜΑΤΙΣΜΟΣ ΜΑΖΙΚΗΣ ΔΙΑΧΕΙΡΙΣΗΣ ΣΥΣΤΗΜΑΤΩΝ

Του φοιτητή

Χαϊδάρη Δημήτρη

Αρ. Μητρώου: 08/3328

Επιβλέπων καθηγητής

Ψαρράς Νικόλαος

Θεσσαλονίκη 2015

Πίνακας περιεχομένων

ΕΙΣΑΓΩΓΗ

Το Puppet OpenSource είναι ένα framework ανοιχτού κώδικα και προσφέρει τη διαχείριση των ρυθμίσεων (configuration management) ενός ή πολλών υπολογιστικών συστημάτων, σε περιβάλλον Unix, Linux ή Microsoft Windows. Είναι ελεύθερο λογισμικό που διανέμεται υπό τους όρους της άδειας χρήσης Apache 2.0, ενώ μπορεί να λειτουργήσει σε αρχιτεκτονική client-server αλλά και αυτόνομα. Η Puppet Labs είναι η εταιρία που δημιούργησε το Puppet OpenSource και προσφέρει επίσης το εμπορικό προϊόν Puppet Enterprise. Το Puppet Enterprise περιλαμβάνει μεταξύ των άλλων μια γραφική κονσόλα διαχείρισης, προβολή στατιστικών με γραφική αναπαράσταση, όπως επίσης απομακρυσμένη διαχείριση.

Η παρούσα εργασία περιλαμβάνει θεωρητική ανάλυση των στοιχείων που απαρτίζουν ή χρησιμοποιεί το Puppet, η οποία συνοδεύεται από παραδείγματα. Συνήθως χρησιμοποιείται πηγαίος κώδικας για την παρουσίαση των δυνατοτήτων που προσφέρει το Puppet. Αρχικά δίνονται παραδείγματα απλής διαχείρισης των υπηρεσιών ssh, apache, postfix περισσότερο για κατανόηση της γλώσσας Puppet και των βασικών της λειτουργιών. Η ανάλυση συνεχίζεται με παρουσίαση του hiera, ενός τρόπου αποθήκευσης των δεδομένων με ιεραρχική δομή. Εκεί χρησιμοποιούνται πιο σύνθετα παραδείγματα διαχείρισης των υπηρεσιών ntp και vmwaretools. Το hiera χρησιμοποιείται επίσης κατά την δημιουργία και αρχικοποίηση ενός γνωστού συστήματος διαχείρισης περιεχομένου που είναι το wordpress, με την δημιουργία ενός profile. Ακολουθεί η χρήση του git ως version control και σύστημα διαχείρισης πηγαίου κώδικα και η διαδικασία ενσωμάτωσης του με το Puppet. Τέλος, εξετάζονται οι λειτουργίες των γραφικών κονσόλων διαχείρισης foreman και Puppet Enterprise Console.

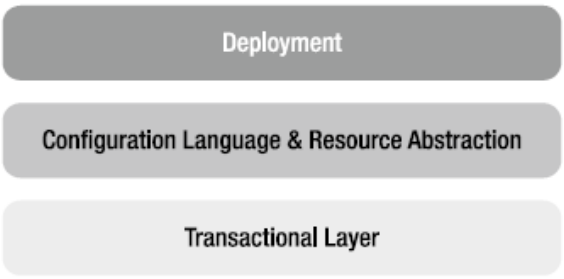
Για τους σκοπούς της πρακτικής εφαρμογής τα παραδείγματα που εξετάζονται, έχουν εφαρμοστεί σε ένα σύνολο από διαφορετικά υπολογιστικά περιβάλλοντα. όπου έχουν ρυθμιστεί:

1. Esxi vSphere Hypervisor, ο οποίος χρησιμοποιείται για την δημιουργία νέων εικονικών υπολογιστικών συστημάτων (virtual machines).
2. Virtual Server που λειτουργεί ως OpenSource Puppet Master και Agent με λειτουργικό CentOS 7 (master2.example.com).
3. Virtual Server που λειτουργεί ως Puppet Master Enterprise με λειτουργικό CentOS 7 (master.example.com).
4. Virtual Server που λειτουργεί ως Puppet Agent με λειτουργικό CentOS 6 (agent1.example.com).

ΚΕΦΑΛΑΙΟ 1

Το λειτουργικό μοντέλο

Το Puppet μπορεί να διαχειριστεί ένα υπολογιστικό σύστημα καθ' όλη την διάρκεια της λειτουργίας του. Από τις αρχικές εγκαταστάσεις λογισμικού που πραγματοποιούνται σε αυτό (apache, postfix κλπ), στις αναβαθμίσεις λογισμικού, στη συντήρηση, και στο στάδιο τέλους του κύκλου ζωής. Στο τελευταίο στάδιο λαμβάνεται η απόφαση να μετακινηθούν οι υπηρεσίες σε διαφορετικό μηχάνημα. Έχει σχεδιαστεί να αλληλεπιδρά συνεχώς με τους hosts, σε αντίθεση με άλλα εργαλεία provisioning που χρησιμοποιούνται για την εγκατάσταση προγραμμάτων, συνήθως με τις προκαθορισμένες ρυθμίσεις αλλά χωρίς να προσφέρουν διαχείριση. Το λειτουργικό μοντέλο του Puppet είναι εύκολο στην κατανόηση, την υλοποίηση και αποτελείται από τρία βασικά στοιχεία (Εικόνα 1.1):

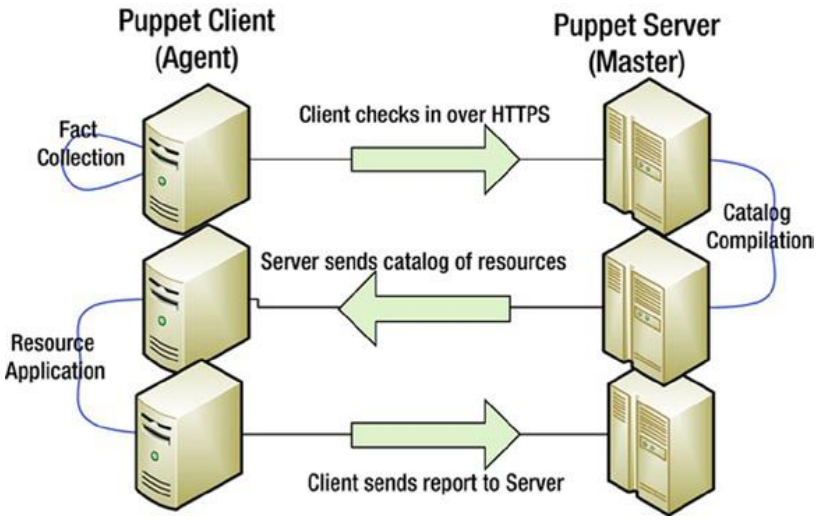


Εικόνα 1.1 – The puppet model

1.1 Ανάπτυξη

Είναι σύνηθες το Puppet να αναπτύσσεται στο μοντέλο πελάτη-εξυπηρετητή (client-server), ενώ κάθε μηχάνημα ανεξαρτήτως σε ποια κατηγορία ανήκει χαρακτηρίζεται ως node.

Ο εξυπηρετητής «Puppet master» εκτελεί την εφαρμογή Puppet master, από την οποία δανείζεται τον χαρακτηρισμό του. Οι υπολογιστές-πελάτες ονομάζονται κατά τον ίδιο τρόπο «Puppet agents», γιατί εκτελείται σε αυτούς η εφαρμογή Puppet agent.



Εικόνα 1.2 - High-level overview of a Puppet configuration run

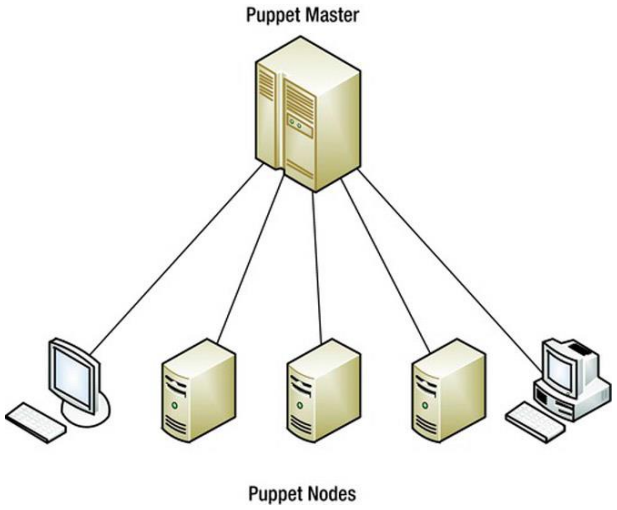
Κατά τη συνήθη διαδικασία ο Puppet master εκτελείται ως υπηρεσία σε έναν εξυπηρετητή και παραμετροποιείται για ένα συγκεκριμένο περιβάλλον. Στη συνέχεια, κάθε Puppet agent ξεχωριστά επικοινωνεί με τον Puppet master μέσω κρυπτογραφημένης και επικυρωμένης σύνδεσης χρησιμοποιώντας το πρωτόκολλο SSL, ώστε να ανακτήσει τις ρυθμίσεις που προορίζονται να εφαρμοστούν στον ίδιο.

Είναι σημαντικό πως όταν δεν υπάρχουν διαθέσιμες ρυθμίσεις προς εφαρμογή για έναν Puppet agent ή όταν έχουν ήδη εφαρμοστεί οι προβλεπόμενες ρυθμίσεις, το Puppet δεν πραγματοποιεί οποιαδήποτε αλλαγή ή ενέργεια. Ένα περιβάλλον υφίσταται αλλαγές από το Puppet μόνο όταν απαιτούνται. Η ιδιότητα αυτή ονομάζεται *idempotency (ταυτοδύναμη συμπεριφορά),* είναι βασικό χαρακτηριστικό του Puppet γιατί παράγει το ίδιο αποτέλεσμα ανεξάρτητα από το αν εκτελείται πολλαπλές φορές. Η παραπάνω διαδικασία χαρακτηρίζεται ως ένα *configuration run*.

Το Puppet μπορεί να εκτελεστεί με διαφορετικούς τρόπους, αναλόγως την περίσταση και το επιδιωκόμενο αποτέλεσμα. Οι επιλογές που υπάρχουν είναι οι παρακάτω τρεις:

- Εκτέλεση του Puppet ως υπηρεσία.
- Εκτέλεση του μέσω ενός βοηθητικού προγράμματος, όπως το cron.
- Με χειροκίνητη εκτέλεση του binary αρχείου.

Η συνήθης πρακτική είναι η πρώτη, δηλαδή η εκτέλεση του Puppet στο παρασκήνιο ως υπηρεσία στους αντίστοιχους nodes. Το Puppet agent ελέγχει περιοδικά και διασταυρώνει ότι οι ρυθμίσεις που προορίζονται για τον node στον οποίο εκτελείται είναι ενημερωμένες. Σε περίπτωση που το παραπάνω δεν ισχύει, πραγματοποιείται ανάκτηση και εφαρμογή των νέων ρυθμίσεων από τον Puppet master. Εξ ορισμού, ο Puppet agent ελέγχει τον Puppet master node για νέες ή τροποποιημένες ρυθμίσεις ανά 30 λεπτά. Η παραπάνω περίοδος μπορεί να αλλάξει ανάλογα με τις απαιτήσεις.



Εικόνα 1.3 - The Puppet client-server model

Επίσης το Puppet μπορεί να αναπτυχθεί και σε αυτόνομο περιβάλλον όπου στην περίπτωση αυτή, ένας node λειτουργεί ταυτόχρονα ως Puppet Master και ως Puppet Agent. Οι ρυθμίσεις δημιουργούνται τοπικά στον Puppet Master και εκτελείται ο Puppet Agent ώστε να τις εφαρμόσει.

Συστήνεται η χρήση της αρχιτεκτονικής client-server καθώς επιτρέπει την κεντρική διαχείριση των ρυθμίσεων των nodes. Επίσης προσφέρει εύκολη πρόσβαση στις αναφορές που λαμβάνονται από τους nodes, συνολικά σε ένα σημείο.

1.2 Η γλώσσα διαχείρισης Puppet και το αφαιρετικό επίπεδο των πόρων

Το Puppet χρησιμοποιεί μια δηλωτική γλώσσα, που ονομάζει *Puppet Language*, για την διαχείριση των πόρων σε ένα σύστημα. Οι πόροι ή αλλιώς *resources* και είναι μια θεμελιώδης μονάδα για την μοντελοποίηση των ρυθμίσεων σε ένα σύστημα. Ως δηλωτική γλώσσα, δημιουργείται μια σημαντική διάκριση μεταξύ του Puppet και πολλών άλλων configuration tools. Η δηλωτική γλώσσα (declarative language) του Puppet κάνει «δηλώσεις» σχετικά με την επιθυμητή κατάσταση του συστήματος, οι οποίες βασίζονται στους πόρους που έχει ή είναι επιθυμητό να έχει ένα σύστημα. Για παράδειγμα δηλώνει ότι ένα συγκεκριμένο πακέτο πρέπει να υπάρχει εγκατεστημένο ή ότι μια υπηρεσία πρέπει να εκτελείται. Δεν ενδιαφέρει τον προγραμματιστή πως το Puppet επιτυγχάνει αυτή τη τελική κατάσταση καθώς οι ενέργειες που πραγματοποιούνται ανήκουν στην αρμοδιότητα του Puppet.

Τα περισσότερα εργαλεία διαχείρισης, όπως το shell ή ένα script γραμμένο σε perl χαρακτηρίζονται *προστακτικά* ή *διαδικαστικά*. Περιγράφουν τα βήματα που εκτελούνται, δηλαδή το πώς, για την πραγματοποίηση μιας εργασίας και όχι την τελική επιθυμητή κατάσταση του συστήματος.

1.2.1 Η γλώσσα διαχείρισης Puppet

Έστω ότι επιθυμούμε την εγκατάσταση του πακέτου vim σε ένα περιβάλλον όπου υπάρχουν υπολογιστικά συστήματα Red Hat Enterprise Linux, Ubuntu, και Solaris. Για την ολοκλήρωση της παραπάνω εργασίας χρειάζεται η συγγραφή ενός script που εκτελεί τα ακόλουθα:

- Σύνδεση σε κάθε host (συμπεριλαμβάνει διαχείριση κωδικών ή κλειδιών).
- Έλεγχος αν το πακέτο vim είναι εγκατεστημένο.
- Αν όχι, χρησιμοποιούνται οι κατάλληλες εντολές για κάθε περιβάλλον ώστε να εγκατασταθεί το πακέτο vim. Για παράδειγμα, εκτελείται η εντολή apt-get για Ubuntu και η εντολή yum για Redhat.
- Αναφορά των αποτελεσμάτων της παραπάνω διαδικασίας για διασφάλιση της ολοκλήρωσης και επιτυχίας.

Στο Puppet η παραπάνω διαδικασία προσεγγίζεται πολύ διαφορετικά, όπου ο χρήστης ορίζει ένα configuration resource για το πακέτο vim. Κάθε resource αποτελείται από *type* (πακέτο, υπηρεσία, cron job), title (το όνομα του resource), και μια σειρά από attributes (ιδιότητες) που καθορίζουν την κατάσταση ενός resource, όπως για το αν μια υπηρεσία θα είναι started ή stopped.

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

Παραπάνω ορίζεται ότι το πακέτο vim πρέπει να εγκατασταθεί, ενώ το resource είναι τύπου package. Η δομή του είναι η εξής:

```
type { title:
attribute => value,
}
```

Μια πλήρη λίστα με τους διαθέσιμους τύπους που μπορεί να διαχειριστεί το puppet βρίσκεται στον Πίνακα 1:

Πίνακας 1 - Resources Supported By Puppet

augeas	nagios_contact	router
computer	nagios_contactgroup	schedule
cron	nagios_host	scheduled_task
exec	nagios_hostdependency	selboolean
file	nagios_hostescalation	selmodule
filebucket	nagios_hostextinfo	service
group	nagios_hostgroup	ssh_authorized_key
host	nagios_service	sshkey
interface	nagios_servicedependency	stage
k5login	nagios_serviceescalation	tidy
macauthorization	nagios_serviceextinfo	user
mailalias	nagios_servicegroup	vlan
maillist	nagios_timeperiod	yumrepo
mcx	notify	zfs
mount	package	zone
nagios_command	resources	zpool

Από τους παραπάνω διακρίνονται ως οι πιο σημαντικοί και συχνά χρησιμοποιούμενοι οι package, file, service, cron, user και group.

Στη συνέχεια ακολουθεί ο τίτλος (title) του resource, στην περίπτωση μας το πακέτο vim που επιθυμούμε την εγκατάσταση του. Ο τίτλος έχει άμεση συσχέτιση και χρησιμοποιείται ως όρισμα στον διαχειριστή πακέτων του κάθε συστήματος. Για παράδειγμα σε περιβάλλον debian: **apt-get install vim**.

Τέλος ορίζεται η ιδιότητα ensure, η οποία περιγράφει την κατάσταση του πακέτου. Στην περίπτωση που επιθυμούσαμε με την απομάκρυνση του πακέτου, θα θέταμε την τιμή absent στην ιδιότητα ensure.

1.2.2 Το αφαιρετικό επίπεδο των πόρων

Μετά τη δημιουργία του resource στο προηγούμενο παράδειγμα, το Puppet αναλαμβάνει όλες τις λεπτομέρειες για την διαχείριση του, όταν ο agent επικοινωνήσει με τον master. Το Puppet διαχειρίζεται τον τρόπο με τον οποίο θα

πραγματοποιήσει μια εργασία, γνωρίζοντας πώς οι διάφορες πλατφόρμες και λειτουργικά συστήματα διαχειρίζονται συγκεκριμένα resources. Κάθε τύπος έχει ένα σύνολο από *providers*, υπεύθυνοι να γνωρίζουν πώς να διαχειριστούν ένα resource χρησιμοποιώντας το κατάλληλο εργαλείο για κάθε σύστημα. Για παράδειγμα ο τύπος package, έχει άνω των είκοσι providers συμπεριλαμβανομένων των εργαλείων yum, aptitude, pkgadd, emerge.

Στον Puppet client εκτελείται η εφαρμογή Facter ώστε να συλλέξει τις πληροφορίες-γεγονότα που σχετίζονται με το συγκεκριμένο σύστημα. Κατά την επικοινωνία του με τον Puppet master, ο Puppet client μεταφέρει τις πληροφορίες που σχετίζονται με τον ίδιο και περισύλλεξε το Facter. Αυτές θα χρησιμοποιηθούν σε επόμενο βήμα από τον master ώστε να επιλέξει τους κατάλληλους providers για τον συγκεκριμένο client. Στην περίπτωση που ο τύπος είναι package, και το λειτουργικό του node είναι RedHat θα χρησιμοποιηθεί ο provider yum για την εγκατάσταση όλων των πακέτων που έχουν οριστεί για το συγκεκριμένο node.

Στη συνέχεια, το Puppet αναφέρει στον master υπό μορφή μηνύματος την επιτυχία ή την αποτυχία της εφαρμογής ενός resource, όπως το πακέτο vim.

1.2.3 Η εφαρμογή Facter και τα γεγονότα

Το Facter είναι ένα εργαλείο ανοιχτού κώδικα και χρησιμοποιείται για την *ευρετηριοποίηση* ενός συστήματος *(system inventory tool)*. Έχει αναπτυχθεί από την Puppet Labs και υπόκειται στους όρους χρήσης της άδειας Apache 2.0. Όταν εκτελείται επιστρέφει γεγονότα (facts) σχετικά με κάθε node, όπως το hostname του, την IP address του, το λειτουργικό σύστημα και την έκδοση του. Τα γεγονότα συλλέγονται όταν ο agent εκτελείται, αποστέλλονται στον master όπου αυτομάτως δημιουργούνται ως μεταβλητές έτοιμες για χρήση.

Κάθε γεγονός επιστρέφεται ως ένα ζεύγος key => value. Για παράδειγμα:

```
$ facter
operatingsystem => Ubuntu
ipaddress => 10.0.0.10
```

Οι παραπάνω τιμές μπορούν να χρησιμοποιηθούν για να ρυθμιστεί κάθε node ξεχωριστά και ανεξάρτητα από τους άλλους.

Το Facter επίσης βοηθάει το Puppet να αντιληφθεί πώς να διαχειριστεί συγκεκριμένα resources σε έναν agent. Έτσι για παράδειγμα σε έναν host με λειτουργικό Ubuntu, το Puppet θα ενημερωθεί πως χρειάζεται να εκτελέσει το εργαλείο apt-get ή το aptitude για την εγκατάσταση νέων πακέτων.

1.3 Το επίπεδο συναλλαγών

Το επίπεδο συναλλαγών του Puppet ταυτίζεται με το engine του. Μια συναλλαγή του Puppet περιγράφει την διαδικασία της ρύθμισης κάθε agent και περιλαμβάνει τα παρακάτω βήματα:

- Την διερμηνεία και μεταγλώττιση του Puppet κώδικα.
- Την αποστολή των μεταγλωττισμένων ρυθμίσεων στον agent.
- Την εφαρμογή των ρυθμίσεων στον agent.
- Την αναφορά των αποτελεσμάτων της συναλλαγής στον master node.

Αρχικά το Puppet συγκεντρώνει τις προς εφαρμογή ρυθμίσεις και υπολογίζει τον τρόπο που θα τις εκτελέσει. Δημιουργεί ένα διάγραμμα συσχέτισης, στο οποίο περιέχονται όλα τα resources με τις μεταξύ τους σχέσεις αλλά και τις συσχετίσεις των resources με το κάθε node ξεχωριστά. Το μοντέλο αυτό θεωρείται από τα ισχυρά χαρακτηρίστηκα του Puppet, γιατί του επιτρέπει να ορίσει την σειρά με την οποία θα εφαρμόσει κάθε resource, βασιζόμενο στα διαγράμματα συσχετίσεων.

Στη συνέχεια το Puppet συλλέγει τα resources που προορίζονται για κάθε node και τα μεταγλωττίζει σε καταλόγους (catalogs), όπου ο καθένας αντιστοιχεί σε ένα node. Οι κατάλογοι αποστέλλοντα στα agent nodes και εφαρμόζονται από το puppet agent binary που βρίσκεται σε κάθε agent node. Τα αποτελέσματα των παραπάνω εργασιών, επιστρέφονται στον puppet master από τους agent, υπό μορφή αναφορών.

Το σημαντικό χαρακτηριστικό του puppet, που ονομάζεται idempotency(*ταυτοδύναμη συμπεριφορά)*, βασίζεται στο επίπεδο συναλλαγών. Όπως ήδη αναφέρθηκε, σημαίνει πως η πολλαπλή εφαρμογή της ίδιας δραστηριότητας, θα αποφέρει το ίδιο αποτέλεσμα κάθε φορά. Δηλαδή προσφέρει συνέπεια των ρυθμίσεων, όσες φορές κι αν εκτελεστούν στους nodes.

ΚΕΦΑΛΑΙΟ 2

Η διαδικασία εγκατάστασης του Puppet

Το Puppet μπορεί να εγκατασταθεί σε μια ποικιλία πλατφόρμων:

- Red Hat Enterprise Linux, CentOS, Fedora, και Oracle Enterprise Linux
- Debian, Ubuntu, Linux mint
- OpenIndiana
- Solaris
- Με μεταγλώττιση από πηγαίο κώδικα
- Microsoft Windows (μόνο clients)
- MacOS X και MacOS X Server
- Άλλα (όπως BSD, Mandrake, και Mandriva)
-

2.1 Εγκατάσταση σε Red Hat Enterprise Linux, fedora ή CentOS

Πρώτο βήμα είναι η εγκατάσταση των απαραίτητων repositories:

- Extra Packages for Enterprise Linux (EPEL)
- Puppet Labs repository

και στη συνέχεια σε εγκατάσταση των απαιτούμενων πακέτων.

Εγκατάσταση του EPEL repository

Enterprise Linux 5:

```
# rpm -Uvh http://dl.fedoraproject.org/pub/epel/5/i386/epel-release-5-4.noarch.rpm
```

Enterprise Linux 6:

```
# rpm -Uvh http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-8.noarch.rpm
```

Εγκατάσταση του Puppet Labs Repository

Enterprise Linux 5:

rpm -ivh http://yum.puppetlabs.com/el/5/products/i386/puppetlabs-release-5-7.noarch.rpm

Enterprise Linux 6:

rpm -ivh http://yum.puppetlabs.com/el/6/products/i386/puppetlabs-release-6-7.noarch.rpm

Εγκατάσταση των απαραίτητων πακέτων

Στον Puppet master πρέπει να υπάρχουν τα πακέτα puppet, puppet-server και facter:

yum install puppet puppet-server facter

Το πακέτο puppet περιέχει τον Puppet agent, το πακέτο puppet-server περιέχει τον master, όπως επίσης το πακέτο facter περιέχει το εργαλείο ευρετηριασμού συστήματος, facter.

Επίσης στον Puppet agent πρέπει να υπάρχουν τα πακέτα puppet και facter:

yum install puppet facter

2.2 Εγκατάσταση σε Debian και Ubuntu

Παρόμοια είναι η διαδικασία εγκατάστασης σε hosts με λειτουργικό Debian ή Ubuntu. Συνοπτικά οι εντολή εγκατάστασης στον Puppet master:

```
# apt-get install puppet puppetmaster
```

Και στον Puppet agent:

```
# apt-get install puppet
```

Για την εγκατάσταση της τελευταίας έκδοσης του Puppet μπορεί να γίνει χρήση του Puppet Lab Repository.

Για Debian Wheezy:

```
# wget http://apt.puppetlabs.com/puppetlabs-release-wheezy.deb
# dpkg -i puppetlabs-release-wheezy.deb
# apt-get update

# wget http://apt.puppetlabs.com/puppetlabs-release-precise.deb
# dpkg -i puppetlabs-release-precise.deb
# apt-get update
```

ΚΕΦΑΛΑΙΟ 3

Παραμετροποίηση του Puppet master

Ακολουθεί μια περιεκτική περιγραφή των βασικών ρυθμίσεων που θα πραγματοποιηθούν στον Puppet master, ώστε να συμπεριφέρεται ως master. Θα εξεταστεί η client-server αρχιτεκτονική, που σημαίνει πως ο master node περιέχει τα configuration data που χρειάζεται ο ίδιος, αλλά και οι nodes που διαχειρίζεται. Κάθε agent node συνδέεται μέσω πρωτοκόλλου SSL στον master node και ανακτά τις ρυθμίσεις που προορίζονται για τον ίδιο.

Στις περισσότερες πλατφόρμες το configuration αρχείο του Puppet βρίσκεται στον φάκελο /etc/puppet/ υπό την ονομασία puppet.conf. Σε περίπτωση αδυναμίας εύρεσης του παραπάνω αρχείου μπορούμε να το δημιουργήσουμε εκτελώντας τα παρακάτω:

```
# cd /etc/puppet/
# puppet master --genconfig > puppet.conf
```

Η δομή του configuration αρχείου puppet.conf είναι παρόμοια με ένα .INI αρχείο όπου οι ρυθμίσεις διαιρούνται σε περιοχές (sections). Κάθε περιοχή είναι υπεύθυνη για την ρύθμιση ενός συγκεκριμένου στοιχείου του Puppet. Η περιοχή [agent] είναι υπεύθυνη για τη ρύθμιση του Puppet agent όπως η περιοχή [master] του Puppet master. Επίσης μπορεί να δηλωθεί global configuration περιοχή [main], όπου ορίζονται οι επιλογές των components. Για την αρχική ρύθμιση του Puppet master αρκεί να θέσουμε μια τιμή στην μεταβλητή “server”, με την οποία προσδιορίζεται το hostname του master server.

```
[main]
server=puppet.example.com
```

Προτείνεται η ρύθμιση ενός DNS CNAME στον Puppet master host, είτε μέσω του αρχείου /etc/hosts, είτε στις ρυθμίσεις του DNS server.

```
# /etc/hosts
127.0.0.1 localhost
192.168.0.1 puppet.example.com
```

Έπειτα προχωρούμε σε δημιουργία του αρχείου site.pp, από το οποίο το Puppet αντλεί τις πληροφορίες που χρειάζεται ώστε να γνωρίζει ποιες ρυθμίσεις να εφαρμόσει σε κάθε agent node. Η προκαθορισμένη διαδρομή του αρχείου είναι η /etc/puppet/manifests/site.pp, η οποία μπορεί να αλλάξει τροποποιώντας τις τιμές των μεταβλητών *manifestdir* και *manifest* στην περιοχή [master] στο αρχείο puppet.conf. Αν για οποιοδήποτε λόγο το αρχείο ή ο γονικός φάκελος δεν υπάρχει, εκτελούμε τις παρακάτω εντολές για την δημιουργία τους:

```
# mkdir /etc/puppet/manifests
# touch /etc/puppet/manifests/site.pp
```

Στην συνέχεια ρυθμίζεται το firewall ώστε να επιτρέπεται η κίνηση για τα ports που χρησιμοποιεί ο Puppet master. Για καθολική πρόσβαση στην port 8140, για το Netfilter firewall εκτελείται:

```
# iptables -A INPUT -p tcp -m state --state NEW --dport 8140 -j ACCEPT
```

Προτείνεται ο περιορισμός των δικτύων από τα οποία θα αποδέχεται ο Puppet master συνδέσεις, το οποίο μπορεί να επιτευχθεί με το όρισμα –s στην παραπάνω εντολή, όπου ορίζονται οι source διευθύνσεις, από τις οποίες επιτρέπεται η πρόσβαση:

```
# iptables -A INPUT -p tcp -m state --state NEW -s 192.168.0.0/24 --dport
8140 -j ACCEPT
```

3.1 Εκκίνηση του Puppet master για πρώτη φορά

Στις περισσότερες διανομές, η υπηρεσία του Puppet master μπορεί να εκκινήσει μέσω ενός init script, με το upstart ή το systemd. Στην περίπτωση διανομών που βασίζονται σε Debian, Ubuntu ή RedHat εκτελείται η παρακάτω εντολή για εκκίνηση της υπηρεσίας *puppetmaster.*

```
# service puppetmaster start
```

Κατά την εκκίνηση του daemon, αρχικοποιείται το Puppet Enviroment, δημιουργείται τοπικά μια αρχή έκδοσης πιστοποιητικών (Certificate Authority , CA) μαζί με κλειδιά για τον master. Επίσης δημιουργείται το κατάλληλο network socket το οποίο αναμένει νέες συνδέσεις. Τα πιστοποιητικά και οι συνοδευόμενες πληροφορίες αποθηκεύονται στον κατάλογο /var/lib/puppet/ssl που περιέχει τα παρακάτω:

```
# ls -l /var/lib/puppet/ssl/
drwxrwx--- 5 puppet puppet 4096 Apr 11 04:05 ca
drwxr-xr-x 2 puppet root 4096 Apr 11 04:05 certificate_requests
drwxr-xr-x 2 puppet root 4096 Apr 11 04:05 certs
-rw-r--r-- 1 puppet puppet 918 Apr 11 04:05 crl.pem
drwxr-x--- 2 puppet root 4096 Apr 11 04:05 private
drwxr-x--- 2 puppet root 4096 Apr 11 04:05 private_keys
drwxr-xr-x 2 puppet root 4096 Apr 11 04:05 public_keys
```

Διακρίνεται η αρχή έκδοσης πιστοποιητικών (ca), αιτήσεις για έκδοση πιστοποιητικών από τους clients, το πιστοποιητικό που χρησιμοποιεί ο ίδιος ο master και τα πιστοποιητικά όλων των clients.

Ο Puppet master μπορεί να εκτελεστεί μέσω του τερματικού, και συνήθως χρησιμοποιείται όταν υπάρχει ανάγκη για debug, όπου χρησιμοποιείται η παρακάτω εντολή:

puppet master --verbose --no-daemonize

Χρησιμοποιούνται τα option --verbose για λεπτομερή καταγραφή της εξόδου και το --no-daemonize, ώστε η εντολή να εκτελεστεί στο προσκήνιο με τις εξόδους της ανακατευθυνόμενες στο stdout.

3.2 Σύνδεση του πρώτου Puppet agent στον master

Στο παράδειγμα που θα χρησιμοποιηθεί, ο host που λειτουργεί ως Puppet agent έχει προεγκατεστημένα τα απαιτούμενα πακέτα *puppet* και *facter.* Επίσης το hostname του έχει αλλάξει σε node1.example.com.

```
node1# puppet agent --test --server=puppet.example.com
Info: Creating a new SSL key for node1.example.com
Info: Caching certificate for ca
Info: Creating a new SSL certificate request for node1.example.com
Info: Certificate Request fingerprint (SHA256):
6F:0D:41:14:BD:2D:FC:CE:1C:DC:11:1E:26:07:4C:08:D0:C
7:E8:62:A5:33:E3:4B:8B:C6:28:C5:C8:88:1C:C8
Exiting; no certificate found and waitforcert is disabled
```

Με την επιλογή --server ορίζεται ο εξυπηρετητής που λειτουργεί ως master. Η παραπάνω επιλογή μπορεί επίσης να οριστεί στο /etc/puppet/puppet.conf αρχείο:

```
# /etc/puppet/puppet.conf
[main]
server=puppet.example.com
```

Η επιλογή --test χρησιμοποιείται ώστε η εντολή να εκτελεστεί στο προσκήνιο, με έξοδο στο stdout και να σταματήσει η εκτέλεση της όταν ολοκληρωθεί. Ο agent node1 παράγει ένα ιδιωτικό κλειδί που χρησιμοποιεί για την δημιουργία του CSR (certificate signing request). Το CSR αποστέλλεται μέσω κρυπτογραφημένης σύνδεσης SSL από τον Agent στον Master. Έπειτα ο Agent node αναμένει την υπογραφή και την επιστροφή του CSR από τον Master. Η εντολή τερματίζεται, έπειτα από την αποστολή του CSR καθώς δεν έχει οριστεί η επιλογή waitforcert, η οποία θα διατηρούσε ανοιχτή τη σύνδεση με τον master, μέχρι να υπογραφεί το CSR.

Για την ολοκλήρωση της παραπάνω εργασίας, δηλαδή της σύνδεσης και αυθεντικοποιήσης του agent node, χρειάζεται να υπογραφεί το πιστοποιητικό που εστάλη από τον agent στον master node.

Για προβολή της λίστας των πιστοποιητικών προς υπογραφή, εκτελείται η παρακάτω εντολή στον master:

```
puppet# puppet cert list
"node1.example.com" (SHA256)
6F:0D:41:14:BD:2D:FC:CE:1C:DC:11:1E:26:07:4C:08:D0:C7:E8:62:A5:33:E3
:4B:8B:C6:28:C5:C8:88:1C:C8
```

Υπογράφεται το πιστοποιητικό του νέου agent στον Puppet master :

```
puppet# puppet cert sign node1.example.com
Notice: Signed certificate request for node1.example.com
Notice: Removing file Puppet::SSL::CertificateRequest node1.pro-
puppet.com at
'/var/lib/puppet/ssl/ca/requests/node1.example.com.pem'
```

Τέλος στον agent node1 εκτελείται η εντολή “puppet agent --test” και επιβεβαιώνεται η ολοκλήρωση της αυθεντικοποίησης με τον master node.

```
# puppet agent --test
Info: Retrieving plugin
Info: Caching catalog for node1.example.com
Info: Applying configuration version '1365655737'
Notice: Finished catalog run in 0.13 seconds
```

ΚΕΦΑΛΑΙΟ 4

Περιγραφή βασικών εννοιών

4.1 Αρχεία manifests

Το Puppet χρησιμοποιεί τη δική του domain-specific language (DSL) για να περιγράψει τις ρυθμίσεις για ένα μηχάνημα. Ο κώδικας puppet σε αυτήν την γλώσσα περιέχει δεδομένα ρύθμισης παραμέτρων (configuration data) και αποθηκεύεται σε αρχεία που το Puppet τα ονομάζει manifests. Οι ακόλουθοι κανόνες ισχύουν για τα αρχεία manifest:

- Χρησιμοποιείται UTF8 κωδικοποίηση.
- Για χαρακτήρα αλλαγής γραμμής μπορεί να χρησιμοποιηθεί το line break του Unix (LF), αλλά και το line break των Windows (CRLF).

4.2 Nodes

Κατά την δήλωση ενός node ορίζεται το node name, σε μονά εισαγωγικά, ακολουθούν οι δηλώσεις των ρυθμίσεων παραμετροποίησης, οι οποίες εκλύονται σε curly braces ({}). Το όνομα του client node, μπορεί να είναι το hostname ή το FQDN (Fully Qualified Domain Name).

```
node 'node1.example.com' {
package { 'vim':
ensure => present,
}
}
```

Για την δήλωση πολλαπλών nodes, υπό το ίδιο domain δεν μπορεί να χρησιμοποιηθεί wildcard, αλλά είναι δυνατή η χρήση κανονικών εκφράσεων (regular expression). Η παρακάτω κανονική έκφραση θα ταιριάξει με όλους τους nodes με domain example.com και hostname www1, www11, www13 κτλπ.

```
node /^www\d+\.example\.com/ {
...
}
```

Έπειτα ακολουθεί η δήλωση ενός resource, όπου διασφαλίζεται η εγκατάσταση του πακέτου vim στον “node1.example.com”. Στην έξοδο της εκτέλεσης του Puppet στον agent node, καταλαβαίνουμε ότι το Puppet προχώρησε σε εγκατάσταση του πακέτου vim.

```
root@node1:~# puppet agent --test
Info: Retrieving plugin
Info: Caching catalog for node1.example.com
Info: Applying configuration version '1375079547'
Notice: /Stage[main]//Node[node1]/Package[vim]/ensure: ensure changed
'purged' to 'present'
Notice: Finished catalog run in 4.86 seconds
```

Το παραπάνω είναι ένα απλό παράδειγμα επεξήγησης της δήλωσης ενός node. Δεν είναι καλή πρακτική η δήλωση ενός resource εντός του ορισμού ενός node, όπως στην παραπάνω περίπτωση το resource package. Αντιθέτως, η χρήση του ορίσματος include, ακολουθούμενο από το όνομα μιας ολόκληρης κλάσης προτιμάται καθώς προσφέρει επαναχρησιμοποίηση και εύκολη διαχείριση του Puppet κώδικα.

4.3 Δημιουργία ενός Puppet Module

Τα Puppet modules αποτελούνται από κάποια βασικά components:

- Resources: Μεμονωμένα στοιχεία ρύθμισης παραμέτρων.
- Αρχεία που διαμοιράζονται στους agents.
- Πρότυπα (Templates) που χρησιμοποιούνται για τον εμπλουτισμό των αρχείων.
- Nodes, όπου προσδιορίζονται οι ρυθμίσεις προς εφαρμογή για τον κάθε agent.
- Κλάσεις που είναι συλλογές από resources.
- Ορισμούς (Definitions), δηλαδή σύνθετες συλλογές από resources.

Ένα module αποτελείται από μια συλλογή από manifests, resources, αρχείων, προτύπων, κλάσεων και definitions. Είναι μια αυτόνομη μονάδα, που περιέχει ότι απαιτείται για την διαμόρφωση μιας συγκεκριμένης εφαρμογής. Υπάρχουν πολλά pre-build Puppet modules που εκτελούν σημαντικές εργασίες, ελεύθερα προς λήψη και χρήση μέσω του Puppet Forge (http://forge.puppetlabs.com/). Ακολουθεί η δημιουργία ενός module το οποίο θα διαχειρίζεται την εφαρμογή sudo σε unix/linux συστήματα.

Κατά την δημιουργία ενός module εφαρμόζεται μια συγκεκριμένη δομή καταλόγων, η οποία επιτρέπει στο Puppet την αυτόματη φόρτωση όλων των modules. Επίσης, κάθε module εμπεριέχει το αρχείο init.pp, που λειτουργεί ως το βασικό (main) αρχείο. Το Puppet διενεργεί μια σειρά από ελέγχους για νέα modules στους καταλόγους που εμπεριέχει η μεταβλητή *module path* η οποία ορίζεται στο αρχείο *puppet.conf* στην περιοχή *[master].* Εξ ορισμού το Puppet ερευνά τους καταλόγους /etc/puppet/modules και /usr/share/puppet/modules. Αυτοί μπορούν είτε να αλλάξουν, είτε να προστεθούν νέοι, αναλόγως τις προτιμήσεις του προγραμματιστή.

```
[master]
modulepath = /etc/puppet/modules:/var/lib/puppet/modules:/opt/modules
```

Το όνομα ενός Puppet module διέπεται από περιορισμούς χαρακτήρων και πρέπει να αποτελείται μόνο από γράμματα, αριθμούς, παύλες ή/και κάτω παύλες. Η δομή ενός module ακολουθεί το παρακάτω πρότυπο:

- <MODULE NAME>
    - manifests
    - files
    - templates
    - lib
    - facts.d
    - tests
    - spec

- <MODULE NAME> : Ο γονικός κατάλογος, για παράδειγμα με όνομα "sudo" (*/etc/puppet/modules/sudo/).*
    - manifests/: Περιέχει τα αρχεία manifest του συγκεκριμένου module.
        - init.pp: Περιέχει τον ορισμό μιας κλάσης. **Το όνομα της κλάσης πρέπει να ταιριάζει με το όνομα του module.**
        - other_class.pp: Περιέχει την κλάση sudo::other_class
        - my_defined_type.pp: Περιέχει έναν defined type με το όνομα sudo:: my_defined_type.pp.
        - implementation/: Ένας κατάλογος που επηρεάζει τα ονόματα των κλάσεων που περιέχει.
            - foo.pp: Αρχείο που περιέχει την κλάση sudo::implementation::foo.
            - bar.pp: Αρχείο που περιέχει την κλάση sudo::implementation::bar.
    - files/: Περιέχει στατικά αρχεία τα οποία μπορούν να λάβουν οι διαχειριζόμενοι nodes.
        - etc/: Κατάλογος που περιέχει αρχεία ρυθμίσεων.
            - sudoers: Το συγκεκριμένο αρχείο μπορεί να αποσταλεί σε έναν agent node που θα χρησιμοποιήσει το module sudo μέσω της διεύθυνσης *source => puppet:///modules/sudo/etc/sudoers.*
    - lib/: Περιέχει plugins, όπως custom facts ή custom resource types, τα οποία θα χρησιμοποιηθούν από κοινού από τον Puppet master server και το Puppet agent service.
    - facts.d/: Περιέχει external facts παρόμοια με τα ruby-based custom facts.
    - templates/: Περιέχει πρότυπα που μπορούν να χρησιμοποιηθούν από τα manifest του module.
        - component.erb: Χρησιμοποιείται από ένα αρχείο manifest μέσω της συνάρτησης *template('sudo/component.erb')*
    - test/: Περιέχει παραδείγματα που παρουσιάζουν την δήλωση κλάσεων και defined τύπων.
    - Spec/: Περιέχει δοκιμές των plugins που βρίσκονται στον κατάλογο lib.

Για τη δημιουργία της δομής του module sudo εκτελούμε τις παρακάτω εντολές:

```
# mkdir -p /etc/puppet/modules/sudo/{files,templates,manifests}
# touch /etc/puppet/modules/sudo/manifests/init.pp
```

Τα περιεχόμενα του αρχείου init.pp φαίνονται παρακάτω:

```
class sudo {
package { 'sudo':
ensure => present,
}
if $::osfamily == 'Debian' {
package { 'sudo-ldap':
ensure => present,
require => Package['sudo'],
}
}
file { '/etc/sudoers':
owner => 'root',
group => 'root',
mode => '0440',
source => "puppet://$::server/modules/sudo/etc/sudoers",
require => Package['sudo'],
}
}
```

Το init.pp manifest περιλαμβάνει μια απλή κλάση, με την ονομασία sudo, η οποία περιέχει τρία resources, δύο resources πακέτου και ένα resource αρχείου. Η ιδιότητα require χαρακτηρίζεται ως μεταπαράμετρος (Metaparameter) η οποία είναι μέρος του Puppet Framework. Το require δημιουργεί μια σχέση εξάρτησης μεταξύ του resource sudo-ldap και του resource sudo, όπου το δεύτερο οφείλει να προϋπάρχει.

Οι σχέσεις έχουν σημαντικό ρόλο στο Puppet καθώς επιτρέπουν στον χρήστη να ορίσει την σειρά με την οποία θα εκτελεστούν συγκεκριμένες εργασίες στους hosts. Για παράδειγμα, υπάρχει η δυνατότητα ορισμού σχέσεων ενημέρωσης και εξάρτησης για την αυτόματη ανανέωση ενός service, την στιγμή που θα εντοπιστούν αλλαγές στο configuration αρχείο του συγκεκριμένου service.

Η κλάση sudo ολοκληρώνεται με τη δήλωση ενός file resource, File["/etc/sudoers"], το οποίο διαχειρίζεται τους /etc/sudoers. Οι πρώτες τρεις ιδιότητες προσδιορίζουν τον owner, το group και τα δικαιώματα του αρχείου. Η επόμενη ιδιότητα (source), χρησιμοποιείται για την ανάκτηση και μεταφορά του αρχείου sudoers από τον master στον agent. Η τιμή της ιδιότητας source είναι

εφάμιλλη με την διαδρομή ενός network share, και ενημερώνει το puppet ότι το αρχείο βρίσκεται στο module με την ονομασία sudo καθώς και με τη διαδρομή του αρχείου. Για την δημιουργία του αρχείου προς μεταφορά /etc/sudoers εκτελούνται οι παρακάτω εντολές, όπου το αρχείο αντιγράφεται αυτούσιο από τον master, στον φάκελο /etc/puppet/modules/sudo/files/etc/:

```
Puppet# mkdir p /etc/puppet/modules/sudo/files/etc
Puppet# cp /etc/sudoers /etc/puppet/modules/sudo/files/etc/sudoers
```

Για την εφαρμογή του συγκεκριμένου module στο σύστημα node1.example.com, αρκεί η δήλωση ενός node ορισμού. Η παρακάτω προσθήκη πραγματοποιείται στο αρχείο /etc/puppet/manifests/site.pp στον master agent.

```
node 'node1.example.com' {
include sudo
}
```

Επίσης εκτελείται η εντολή puppet agent --test στον puppet agent ώστε να πραγματοποιηθεί η σύνδεση με τον master και να γίνει αντιληπτό ότι υπάρχει νέο διαθέσιμο configuration. Η έξοδος της εντολής είναι η εξής:

```
Info: Retrieving plugin
Info: Caching catalog for node1.example.com
Info: Applying configuration version '1365735606'
Notice: /Stage[main]/Sudo/Package[sudo]/ensure: created
Notice: /Stage[main]/Sudo/File[/etc/sudoers]/ensure: defined content as
'{md5}1b00ee0a97a1bcf9961e4
76140e2c5c1'
Notice: Finished catalog run in 25.94 seconds
```

Κατά την εκτέλεση της εντολής ο Puppet agent δημιουργεί cache, όπου φυλάσσει τις ρυθμίσεις που πρόκειται να εφαρμόσει. Η cache θα χρησιμοποιηθεί σε περίπτωση αποτυχίας της σύνδεσης του puppet agent με τον master σε μελλοντική εκτέλεση. Έπειτα το Puppet εφαρμόζει τα resources που του έχουμε υποδείξει για τον συγκεκριμένο node, δηλαδή το resource πακέτου sudo και το resource αρχείου /etc/sudoers το οποίο μεταφέρεται από τον master στον agent. Κατά την αντιγραφή του αρχείου και μόνο στην περίπτωση πραγματοποίησης αλλαγών, το Puppet κρατάει αντίγραφο ασφαλείας του παλαιού αρχείου. Η παραπάνω διαδικασία ονομάζεται file bucketing.

Παρόμοια είναι η έξοδος της εντολής “# puppet master --no-daemonize --verbose --debug “ στον master.

```
puppet# puppet master --no-daemonize --verbose --debug
Notice: Starting Puppet master version 3.1.1
[..]
Info: Caching node for node1.example.com
Debug: importing '/etc/puppet/modules/sudo/manifests/init.pp' in
environment production
Debug: Automatically imported sudo from sudo into production
Notice: Compiled catalog for node1.example.com in environment production
in 0.23 seconds
Debug: Finishing transaction 70065298446380
Debug: Received report to process from node1.example.com
Debug: Processing report from node1.example.com with processor
Puppet::Reports::Store
```

Το Puppet φορτώνει αυτομάτως το module sudo, λόγω της κατάλληλης δομής των φακέλων που δημιουργήθηκαν προηγουμένως. Έπειτα δημιουργεί τον catalog, δηλαδή το σύνολο των ενεργειών που θα εκτελεστούν στον node1.example.com και τις μεταδίδει στον agent node, όπου και εφαρμόζονται. Όταν ο Puppet agent εκτελείται σε daemon mode, συνδέεται στον master ανά περιοδικό διάστημα 30 λεπτών για να ελέγξει και να ενημερώσει το configuration του αν χρειαστεί. Το παραπάνω διάστημα μπορεί να αλλάξει, τροποποιώντας την τιμή της επιλογής runinterval που βρίσκεται στο αρχείο /etc/puppet/puppet.conf στον agent node.

[agent]
runinterval=3600

4.4 Κληρονομικότητα των nodes

Το Puppet υποστηρίζει την κληρονομικότητα μεταξύ nodes, αλλά είναι προτεινόμενο να αποφεύγεται καθώς δεν θεωρείται καλή πρακτική. Στο Puppet 3.7 ο μεταγλωττιστής επισημάνει τον κώδικα που περιέχει κληρονομικότητα ως deprecated, ενώ στο μέλλον πρόκειται να καταργηθεί συνολικά ως έννοια (Puppet 4). Η κληρονομικότητα μεταξύ των nodes, συνήθως προκαλεί επιπλοκές και ασάφειες στον κώδικα. Η επαναχρησιμοποίηση του Puppet κώδικα μπορεί να επιτευχθεί με διαφορετικούς τρόπους όπως χρησιμοποιώντας κλάσεις ή defined τύπους. Η ιεραρχική δομή που προσφέρει η κληρονομικότητα, μπορεί να αντικατασταθεί με την χρήση εξωτερικών πηγών δεδομένων (external sources). Εκεί αποθηκεύονται ξεχωριστά τα δεδομένα που χρειάζεται κάθε node και ακολουθεί η ανάκτησή τους από τα αρχεία manifest. Η βέλτιστη λύση

πραγματοποίησης μιας ιεραρχικής δομής αυτήν τη στιγμή, είναι το **hiera,** που είναι διαθέσιμο από την έκδοση Puppet 3.0 και μετά. Ένα παράδειγμα κληρονομικότητας φαίνεται παρακάτω.

```
node basenode {
include sudo
include mailx
}
node
'web.example.com'
inherits basenode {
include apache
}
```

4.5 Πεδίο ισχύς μιας μεταβλητής

Το scope ή αλλιώς πεδίο ισχύς μιας μεταβλητής ορίζεται από το Puppet κατά την δημιουργία μιας κλάσης, ενός definition ή ενός node. Καθώς υπάρχει και το top scope, για ότι ορίζεται εκτός αυτών των δομών. Κάθε δεδομένη στιγμή, τέσσερις περιοχές ισχύς των μεταβλητών είναι διαθέσιμες για χρήση από το Puppet, οι οποίες είναι: top scope, node scope, parent scope, και local scope. Το top scope περιλαμβάνει τις μεταβλητές που έχουν οριστεί στο site.pp, δηλαδή στο κυρίως αρχείο όπου πραγματοποιούνται οι ορισμοί των node. Υπάρχει καθολική πρόσβαση(explicit) στις μεταβλητές του top scope, από όλα τα αρχεία manifest. Οι μεταβλητές του top score γίνονται διαθέσιμες χρησιμοποιώντας το "::" πριν από το όνομα της μεταβλητής. Για παράδειγμα η μεταβλητή-γεγονός $::osfamily προσφέρει το λειτουργικό σύστημα ενός agent σε ένα manifest που θέλει να το αξιοποιήσει. Το node scope είναι το πεδίο που δημιουργείται κατά τον ορισμό ενός node και έχει ισχύ στην περιοχή μεταξύ του ανοίγματος και του κλεισίματος των curly braces ({}). Το node scope είναι ανώνυμο και δεν υπάρχει η δυνατότητα explicit πρόσβασης στις μεταβλητές που περιέχει. Το local scope ορίζεται κατά την δημιουργία μιας κλάσης ή ενός definition. Στο Puppet 3, το parent scope ορίζεται ως την περιοχή της δήλωση μιας κλάσης, η οποία κληρονομείται από κάποια άλλη κλάση χρησιμοποιώντας τη λέξη-κλειδί *inherits.*

```
class ssh::params {
case $::osfamily {
'Debian': { $sshd_package = 'ssh' }
'RedHat': { $sshd_package = 'openssh-server' }
default: {fail("Login class does not work on osfamily: ${::osfamily}")}
}
}

class ssh inherits ssh::params {
package { $::ssh::params::sshd_package:
ensure => installed,
}
}
include ssh
```

Στην προκειμένη περίπτωση η κλάση ssh::params κληρονομείται και εμπεριέχεται στο local scope της κλάσης ssh. Η parental μεταβλητή $::ssh::params::sshd_package χρησιμοποιείται για τον καθορισμό του πακέτου προς εγκατάσταση και λαμβάνει τιμή ανάλογα την οικογένεια του λειτουργικού συστήματος που χρησιμοποιεί ο client.

Για την αποφυγή της κληρονομικότητας, υπάρχει η δυνατότητα χρησιμοποίησης του include, ώστε η κλάση ssh να έχει πρόσβαση στις μεταβλητές της κλάσης ssh:params.

```
class ssh::params {
case $::osfamily {
'Debian': { $sshd_package = 'ssh' }
'RedHat': { $sshd_package = 'openssh-server' }
default: {fail("Login class does not work on osfamily: ${::osfamily}")}}}

class ssh {
include ssh::params
package { $::ssh::params::sshd_package:
ensure => installed,
}}
include ssh
```

4.6 Παραμετροποιήσιμες κλάσεις

Η παραπάνω μέθοδος δεν λειτουργεί με τις parameterized κλάσεις, δηλαδή τις κλάσεις που δέχονται παραμέτρους. Στο επόμενο παράδειγμα πραγματοποιείται κλήση της κλάσης ssh, που αποφέρει το ίδιο αποτέλεσμα.

```
class ssh::params {
case $::osfamily {
'Debian': {
$sshd_package = "ssh"
$sshd_service = "ssh"
}
'RedHat': {
$sshd_package = "openssh-server"
$sshd_service = "sshd"
}
default: {fail("Login class does not work on osfamily: ${::osfamily}")}
}
}
class ssh (
manage_package = false,
manage_service = false,
package_name = $::ssh::params::sshd_package
) inherits ssh::params {
if manage_package == true {
package { $package_name:
ensure => installed,
}
}
if manage_service == true {
service { $::ssh::params::sshd_service:
ensure => running,
}
}
}

class { 'ssh':
manage_package => true,
manage_service => true,
}
```

Με αυτόν τον τρόπο έχει αυξηθεί η πολυπλοκότητα του κώδικα, αλλά παράλληλα η ευελιξία και η ισχύ του. Είναι κοινή πρακτική και σύμβαση μεταξύ των Puppet developer η χρήση της κλάσης <module_name>::params, όπου ορίζονται οι μεταβλητές που χρησιμοποιούνται από τις κλάσεις του module. Η ιδέα είναι να λειτουργεί ως το μοναδικό μέρος σε ένα module όπου συγκεντρώνονται οι δηλώσεις των μεταβλητών και συμβάλει στην επαναχρησιμοποίηση και ευαναγνωσιμότητα του κώδικα. Η κληρονόμηση της συγκεκριμένης κλάσης, επιτρέπει την πρόσβαση στην μεταβλητή ::ssh::params::sshd_package από την κλάση ssh. **Είναι αποδεκτή η χρήση της κληρονομικότητας αποκλειστικά και μόνο σε μια τέτοια περίπτωση.** Κατά την κλήση της κλάσης ssh, ορίζουμε τις μεταβλητές manage_package, manage_service ως true. Ο λόγος που είναι δυνατή η αντικατάσταση των συγκεκριμένων μεταβλητών, είναι ότι έχουν προηγουμένως δηλωθεί στην κλάση ssh, στο default πεδίο δήλωσης μεταβλητών.

```
class ssh (
manage_package = false,
manage_service = false,
package_name = $::ssh::params::sshd_package
) inherits ssh::params {

…}
```

4.7 Οι κλάσεις στο Puppet

Μια κλάση του Puppet είναι μια συλλογή από Puppet κώδικά, ο οποίος συνήθως εκτελεί μια συγκεκριμένη εργασία. Το όνομα μιας κλάσης λειτουργεί ως το χαρακτηριστικό της γνώρισμα. Οι κλάσεις αποθηκεύονται στα modules για μελλοντική χρήση, και δεν εφαρμόζονται μέχρις ότου να κληθούν με το όνομά τους. Η προσθήκη μιας κλάσης σε έναν κατάλογο ενός node μπορεί να επιτευχθεί είτε με την δήλωση της σε ένα αρχείο manifest, είτε με την εκχώρηση της από κάποιο εργαλείο ENC (External Node Classifier). Μια κλάση είναι συνήθως υπεύθυνη για την παραμετροποίηση ενός μεγάλου αριθμού από μέρη μιας λειτουργίας, όπως για παράδειγμα τα πακέτα, τα αρχεία ρυθμίσεων και τις υπηρεσίες μιας εφαρμογή για την εύρυθμη λειτουργία της.

4.7.1 Ορισμός μιας κλάσης

Ακολουθούν δύο παραδείγματα ορισμού κλάσεων. Στο πρώτο ορίζεται η κλάση base::linux η οποία δεν δέχεται παραμέτρους ενώ στο δεύτερο η κλάση apache, η οποία δέχεται.

```
# Μια κλάση δίχως παραμέτρους
class base::linux {
  file { '/etc/passwd':
    owner => 'root',
    group => 'root',
    mode  => '0644',
  }
  file { '/etc/shadow':
    owner => 'root',
    group => 'root',
    mode  => '0440',
  }
}
```

```
# Μια κλάση με παραμέτρους. Τίθεται προκαθορισμένη τιμή στην παράμετρο
$version.
    class apache ($version = 'latest') {
      package {'httpd':
        ensure => $version, # Γίνεται χρήση της παραμέτρου
        before => File['/etc/httpd.conf'],
      }
      file {'/etc/httpd.conf':
        ensure  => file,
        owner   => 'httpd',
        content => template('apache/httpd.conf.erb'), # Αρχείο Template
από ένα module
      }
      service {'httpd':
        ensure    => running,
        enable    => true,
        subscribe => File['/etc/httpd.conf'],
      }
    }
```

Μια κλάση αποτελείται από τα παρακάτω στοιχεία:

- Ξεκινάει με τη λέξη-κλειδί class.
- Ακολουθεί το όνομα της κλάσης.
- Μια προαιρετική λίστα από παραμέτρους, που εμπεριέχονται σε παρένθεση. Οι παράμετροι διαχωρίζονται με κόμμα και δηλώνονται ως μεταβλητές ($version). Στις παραμέτρους μπορούν να οριστούν προκαθορισμένες τιμές. ($version = latest).
- Η λίστα των παραμέτρων ολοκληρώνεται με προαιρετικό κόμμα στο τέλος και με κλείσιμο της παρένθεσης.
- Υπάρχει η δυνατότητα κληρονομικότητας δηλώνοντας τη λέξη-κλειδί inherits, η οποία ακολουθείται από το όνομα κλάσης.
- Η αρχή δήλωσης της κλάσης ξεκινάει με άνοιγμα curly braces.
- Περιοχή με κώδικα Puppet, ο οποίος συνήθως περιέχει τουλάχιστον μια δήλωση ενός resource.
- Ο ορισμός της κλάσης ολοκληρώνεται με κλείσιμο των curly braces..

4.7.2 Παράμετροι και μεταβλητές

Οι παράμετροι επιτρέπουν στην κλάση να δεχτεί εξωτερικά δεδομένα, δηλαδή μεταβλητές οι οποίες παίρνουν τιμή κατά την κλήση της κλάσης. Αρκετές κλάσεις χρειάζονται εξωτερικές τιμές μεταβλητών εκτός των γεγονότων (facts), για τη ρύθμιση τους. Κάθε παράμετρος μιας κλάση παίρνει τη μορφή μεταβλητής και μπορεί να χρησιμοποιηθεί από την κλάση σαν κάθε άλλη μεταβλητή του Puppet. Οι τιμές των παραμέτρων μπορούν να προκαθοριστούν κατά τη δήλωση μιας κλάσης και στη συνέχεια να αλλάξουν κατά την κλήση της κλάσης. Η έλλειψη των προκαθορισμένων τιμών στις παραμέτρους μιας κλάσης μπορεί να έχει αρνητικά αποτελέσματα αν συνοδεύεται με μη ορισμό των παραμέτρων κατά την κλήση της συγκεκριμένης κλάσης. Το Puppet παράγει έξοδο σφάλματος όταν υπάρχει η παραπάνω περίσταση κατά την μεταγλώττιση του κώδικα. Γι’ αυτό κάποιοι προγραμματιστές προτείνουν τη δήλωση προκαθορισμένων τιμών στις παραμέτρους μιας κλάσης.

Από την άλλη μεριά, η συγκεκριμένη πρακτική μπορεί να έχει αρνητικά, μη επιθυμητά αποτελέσματα κατά την συγγραφή ή αποσφαλμάτωση μιας κλάσης. Η ύπαρξη προκαθορισμένων τιμών σε μεταβλητές παραμέτρων συμβάλλει στην επιτυχημένη εκτέλεση του Puppet κώδικα, από πλευρά μεταγλώττισης. Όμως το αποτέλεσμα της εκτέλεσης, οι ενέργειες που εκτελεί δηλαδή το Puppet μπορεί είναι εσφαλμένο. Είναι κάτι που μπορεί να προκύψει από την ύπαρξη μιας ή πολλών προκαθορισμένων τιμών που έχουν δηλωθεί σε μια κλάση, για τις οποίες δεν πραγματοποιήθηκε αντικατάσταση με τις επιθυμητές τιμές κατά την κλήση της κλάσης. Κατά την παραπάνω περίπτωση παράγεται λανθασμένο αποτέλεσμα σε έναν ή παραπάνω agent, δεν υπάρχει σφάλμα εξόδου από το Puppet κατά την μεταγλώττιση και η αποσφαλμάτωση για την εύρεση της αιτίας του προβλήματος είναι δύσκολη και χρονοβόρα.

Η παραπάνω συμπεριφορά μπορεί να αποφευχθεί μη ορίζοντας προκαθορισμένες τιμές στις παραμετροποιήσιμες μεταβλητές μιας κλάσης. Στην περίπτωση που ο προγραμματιστής ξεχάσει να δηλώσει τιμή σε κάποια από αυτές τις παραμέτρους, το Puppet θα τον ειδοποιήσει με σφάλμα εκτέλεσης, και έτσι θα μπορέσει εύκολα να εντοπίσει το σημείο όπου χρειάζεται διόρθωση.

4.7.3 Διαδρομή ορισμού στον δίσκο

Οι κλάσεις αποθηκεύονται κατά κύριο λόγο στα modules. Όταν υπάρχει η κατάλληλη δομή των αρχείων στον δίσκο, το Puppet το αντιλαμβάνεται και φορτώνει αυτομάτως τις κλάσεις στο σύνολό τους χρησιμοποιώντας τα ονόματά τους. Οι κλάσεις πρέπει να αποθηκεύονται υπό τον φάκελο manifests/ του module στο οποίο ανήκουν. Επίσης να ορίζεται μια κλάση ανά αρχείο, καθώς το όνομα του αρχείου πρέπει να είναι ταυτόσημο με το όνομα κλάσης. Τέλος, υπάρχει η δυνατότητα μια κλάση ή πολλές να οριστούν στο main manifest αρχείο (/etc/puppetlabs/puppet/manifests/site.pp), αλλά δεν είναι συνήθη πρακτική.

4.7.4 Δήλωση κλάσεων

Κατά τη κλήση μιας κλάσης σε ένα αρχείο manifest, το Puppet προσθέτει τα resources που περιέχονται για αυτήν τη κλάση σε έναν κατάλογο (catalog). Ο κατάλογος περιέχει τις προς εκτέλεση ενέργειες από το puppet agent. Η δήλωση/κλήση μιας κλάσης μπορεί να πραγματοποιηθεί στο πεδίο ορισμού ενός node, σε top level επίπεδο στο site manifest, σε άλλες κλάσεις ή defined types. Η δήλωση μιας κλάσης δεν είναι ο μοναδικός τρόπος για την προσθήκη της σε έναν κατάλογο, καθώς η ανάθεση κλάσεων σε nodes μπορεί να πραγματοποιηθεί με κάποιον ENC(External Node Classifier).

Ο ENC είναι πρόγραμμα γραμμένο σε οποιαδήποτε γλώσσα προγραμματισμού, εκτελείται από το Puppet και δέχεται σαν όρισμα το όνομα ενός agent node. Έπειτα επιστρέφει στο Puppet, σε μορφή yaml, διάφορες πληροφορίες που σχετίζονται με τον συγκεκριμένο node, συμπεριλαμβανομένων των κλάσεις που αντιστοιχούν στον ίδιο. Το ENC μπορεί να ρυθμιστεί με εξωτερικές πηγές δεδομένων, δηλαδή αρχεία που εμπεριέχουν δεδομένα για κάποιον node, αλλά και με μεταβλητές που υπάρχουν διαθέσιμες στο Puppet ($::osfamily), για την κατηγοριοποίηση των διαφόρων nodes σε ένα περιβάλλον.

Οι κλάσεις χαρακτηρίζονται ως singletons, δηλαδή τα resources που περιέχει μια κλάση εξετάζονται μια φορά ανά μεταγλώττιση. Είναι ανεξάρτητο το αν η κλάση παρουσιάζει διαφορετική συμπεριφορά λόγω των αλλαγών στις τιμές των παραμέτρων.

Το Puppet προσφέρει δύο τρόπους για την δήλωση/κλήση μιας κλάση, οι οποίοι κατηγοριοποιούνται σε include-like και resource-like.

4.7.4.1 Include-like συμπεριφορά

Οι συναρτήσεις include, require, contain και hiera_include επιτρέπουν την ασφαλή δήλωση/κλήση μιας κλάσης επαναλαμβανόμενα, ανεξάρτητα από τον αριθμό των δηλώσεων. Η δήλωση της ίδιας κλάσης πάνω από μια φορά θα αποφέρει το ίδιο αποτέλεσμα, την προσθήκη της δηλαδή στον κατάλογο ενός agent μια φορά και μόνο. Αυτό επιτρέπει στις κλάσεις ή στους defined types να διαχειρίζονται οι ίδιοι τις εξαρτήσεις τους. Επίσης επιτρέπει στον προγραμματιστή του Puppet, την δημιουργία επικαλυπτόμενων ρόλων για έναν node.

Η include-like συμπεριφορά βασίζεται σε δεδομένα από εξωτερικές πηγές, όπως επίσης στις προκαθορισμένες τιμές των παραμέτρων των κλάσεων. Όταν πραγματοποιείται η κλήση/δήλωση μιας κλάσης, το Puppet εκτελεί τις παρακάτω ενέργειες για να προσδιορίσει μια τιμή για κάθε παράμετρο της κλάσης:

1. Αναζητεί την τιμή της παραμέτρου σε μια εξωτερική πηγή δεδομένων, χρησιμοποιώντας το μοντέλο <class name>::<parameter name>. Για παράδειγμα, για την ανάκτηση της τιμής της παραμέτρου version, η οποία βρίσκεται στην κλάση apache, το Puppet αναζητά την τιμή της μεταβλητής apache::version.
2. Σε περίπτωση που δεν έχει οριστεί μια συγκεκριμένη τιμή στην παραπάνω παράμετρο από μια εξωτερική πηγή δεδομένων, το Puppet θα χρησιμοποιήσει την προκαθορισμένη τιμή, η οποία έχει δηλωθεί κατά τον ορισμό της κλάσης.
3. Αποτυχία της μεταγλώττισης με μήνυμα σφάλματος σε περίπτωση αδυναμίας εύρεσης της τιμής της παραμέτρου.

Στις περισσότερες περιπτώσεις, προτείνεται η include-like δήλωση των κλάσεων, οι οποίες να περιέχουν μόνο Puppet κώδικα χωρίς δεδομένα. Οι παράμετροι των κλάσεων συμβάλλουν στην εξειδίκευση των κλάσεων για τους nodes. Ως εκ τούτου, θα πρέπει κατά το δυνατόν να φυλάσσονται ξεχωριστά από τον Puppet κώδικα μιας κλάσης. **Με αυτόν τον τρόπο επιτυγχάνεται η επαναχρησιμοποίηση του Puppet κώδικα, ενώ διαχωρίζονται τα δεδομένα και οι παράμετροι μιας κλάσης από το μοντέλο.**

Ωστόσο, η παραπάνω οδηγία μπορεί να προκαλέσει προβλήματα συμβατότητας σε παλαιότερες εκδόσεις του Puppet και εκεί να χρειαστούν συμβιβασμοί.

4.7.4.1.1 Η συνάρτηση include

Η include συνάρτηση είναι ο βασικός τρόπος δήλωσης/κλήσης μιας κλάσης. Όπως υποδεικνύει το όνομά της, η συνάρτηση include έχει include-like συμπεριφορά και μπορεί να δεχτεί:

- Μια κλάση
- Μια λίστα από κλάσεις.
- Έναν πίνακα κλάσεων.

```
    include base::linux
    include base::linux # Το αποτέλεσμα είναι το ίδιο αφού η κλάση
εκτελείται μια φορά από τον μεταγλωττιστή.
    include base::linux, apache # Καλείται λίστα κλάσεων
    $my_classes = ['base::linux', 'apache']
    include $my_classes # Καλείται πίνακας που περιέχει κλάσεις
```

4.7.4.1.2 Η συνάρτηση require

Με τη συνάρτηση require δηλώνονται κλάσεις οι οποίες χαρακτηρίζονται από το Puppet ως προαπαιτούμενες. Δηλαδή, δημιουργείται μια σχέση εξάρτησης ανάμεσα στις κλάσεις που δηλώνονται με την συνάρτηση require(apache) και στη δομή, που περιλαμβάνει τη συνάρτηση (apache::vhost).

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

Στο παραπάνω παράδειγμα το Puppet θα διασφαλίσει ότι τα resources που περιέχονται στην κλάση apache, θα εφαρμοστούν πριν από κάθε resource του instance apache::vhost. Η συνάρτηση require έχει include-like συμπεριφορά και μπορεί να δεχτεί:

- Μια κλάση
- Μια λίστα από κλάσεις.
- Έναν πίνακα κλάσεων.

4.7.4.1.3 Η συνάρτηση hiera_include

Η συνάρτηση δέχεται ως παράμετρο μια λίστα ονομάτων, κλάσεων του hiera και έπειτα προχωρά στην δήλωση αυτών. Μπορεί να χρησιμοποιηθεί για την δημιουργία μιας ιεραρχικής δομής, έτσι απαλλάσσει τον Puppet κώδικα από την επαναλαμβανόμενη δήλωση των nodes. Με αυτόν τον τρόπο το hiera χρησιμοποιείται ως ENC.

```
# /etc/puppetlabs/puppet/hiera.yaml
...
hierarchy:
  - "%{::clientcert}"
  - common

# /etc/puppetlabs/puppet/hieradata/web01.example.com.yaml
---
classes:
  - apache
  - memcached
  - wordpress

# /etc/puppetlabs/puppet/hieradata/common.yaml
---
classes:
  - base::linux


# /etc/puppetlabs/puppet/manifests/site.pp
    hiera_include(classes)
```

Στο παραπάνω παράδειγμα, στον node “web01.example.com” θα δηλωθούν/κληθούν οι κλάσεις apache, memcached, wordpress, και η base::linux. Ενώ στους υπόλοιπους nodes με διαφορετικό hostname, θα δηλωθεί μόνο η κλάση base::linux. Η συνάρτηση hiera-include έχει include-like συμπεριφορά.

4.7.4.2 Resource-like συμπεριφορά

Στην resource-like συμπεριφορά, η δήλωση/κλήση μιας κλάσης υποχρεωτικά **δεν** επαναλαμβάνεται. Κατ’ αυτόν τον τρόπο μια κλάση δηλώνεται μια και μόνο φορά, διαφορετικά προκαλείται σφάλμα στην μεταγλώττιση του Puppet κώδικα. Κατά την κλήση μιας κλάσης με την παραπάνω μέθοδο, υπάρχει η δυνατότητα αντικατάστασης των παραμέτρων της κλάσης κατά την μεταγλώττιση. Όσοι παράμετροι δεν αντικατασταθούν, θα λάβουν τιμή από την εξωτερική πηγή δεδομένων αν έχει δηλωθεί. Το Puppet εκτελεί τις παρακάτω ενέργειες για να προσδιορίσει τις τιμές των παραμέτρων μιας κλάσης:

1. Χρησιμοποιεί τα ορίσματα της κλάσης για να αντικαταστήσει την τιμή της παραμέτρου.
2. Αναζητάει την τιμή της παραμέτρου από μια εξωτερική πηγή δεδομένων, χρησιμοποιώντας το μοντέλο <class name>::<parameter name>. Για παράδειγμα, για την ανάκτηση της τιμής της παραμέτρου version, η οποία βρίσκεται στην κλάση apache, το Puppet αναζητά την τιμή της μεταβλητής apache::version.
3. Σε περίπτωση που δεν έχει οριστεί μια συγκεκριμένη τιμή στην παραπάνω παράμετρο από μια εξωτερική πηγή δεδομένων, το Puppet θα χρησιμοποιήσει την προκαθορισμένη τιμή, η οποία έχει δηλωθεί κατά τον ορισμό της κλάσης.
4. Αποτυχία την μεταγλώττισης με μήνυμα σφάλματος στη περίπτωση αδυναμίας εύρεσης της τιμής της.

Η δήλωση/κλήση μιας κλάσης, με resource-like συμπεριφορά, μοιάζει με την δήλωση ενός απλού resource και χρησιμοποιείται ο pseudo-resource τύπος “class”.

```
# Κλήση με χρήση παραμέτρου:
class {'apache':
  version => '2.2.21',
}
# Κλήση μιας κλάσης δίχως παραμέτρους:
class {'base::linux':}
```

Στη resource-like συμπεριφορά οι πολλαπλές δηλώσεις/κλήσεις της ίδιας κλάσης δεν επιτρέπονται. Επίσης οι παράμετροι που δέχεται μια κλάση μπορεί να αντικατασταθούν κατά τη μεταγλώττιση, όπου στη δήλωση της κλάσης ορίζονται υπό μορφή ιδιοτήτων (parameter => ‘value’). Για όσες παραμέτρους δεν λάβουν τιμή με τον παραπάνω τρόπο, το Puppet θα ερευνήσει αν έχουν δηλωθεί σε εξωτερικές πηγές δεδομένων. Οι παράμετροι που δεν αντικατασταθούν, λαμβάνουν τις προκαθορισμένες τους τιμές, αν δηλώθηκαν κατά τον ορισμό της κλάσης. Τέλος, αν κάποια παράμετρος δεν λάβει τιμή με κανέναν από τους παραπάνω τρόπους, θα προκληθεί σφάλμα κατά την μεταγλώττιση.

Στο παρακάτω παράδειγμα παρουσιάζεται ένας πιο εξεζητημένος τρόπος δήλωσης των παραμέτρων που χρειάζεται μια κλάση. Ακολουθώντας παρόμοια

πρότυπα υπάρχει η δυνατότητα δήλωσης των προκαθορισμένων παραμέτρων με σημαντικά λιγότερο και “καθαρότερο” κώδικα.

```
# /etc/puppet/modules/webserver/manifests/params.pp

  class webserver::params {
   $packages = $operatingsystem ? {
     /(?i-mx:ubuntu|debian)/        => 'apache2',
     /(?i-mx:centos|fedora|redhat)/ => 'httpd',
   }
   $vhost_dir = $operatingsystem ? {
     /(?i-mx:ubuntu|debian)/        => '/etc/apache2/sites-enabled',
     /(?i-mx:centos|fedora|redhat)/ => '/etc/httpd/conf.d',
   }
  }

  # /etc/puppet/modules/webserver/manifests/init.pp

  class webserver(
   $packages  = $webserver::params::packages,
   $vhost_dir = $webserver::params::vhost_dir
  ) inherits webserver::params {

   package { $packages: ensure => present }

   file { 'vhost_dir':
     path   => $vhost_dir,
     ensure => directory,
     mode   => '0750',
     owner  => 'www-data',
     group  => 'root',
   }
  }
```

ΚΕΦΑΛΑΙΟ 5

Παραδείγματα δημιουργίας module διαχείρισης των υπηρεσιών SSH, POSTFIX, APACHE

5.1 Διαχείριση της υπηρεσίας SSH

Η εντολή “puppet module generate module_name” επιτρέπει την αυτόματη δημιουργία μιας κενής δομής με τα βασικά αρχεία.

```
# cd /etc/puppet/modules
# puppet module generate ssh
Notice: Generating module at /etc/puppet/ssh
ssh
ssh/Modulefile
ssh/README
ssh/manifests
ssh/manifests/init.pp
ssh/spec
ssh/spec/spec_helper.rb
ssh/tests
ssh/tests/init.pp
```

Ακολουθεί η δημιουργία και συγγραφή των απαραίτητων κλάσεων.

5.1.1 Η κλάση ssh::params

Στη κλάση ssh::params δηλώνονται οι μεταβλητές που θα χρησιμοποιηθούν από τις διάφορες κλάσεις του module. Οι μεταβλητές γίνονται διαθέσιμες στις υπόλοιπες κλάσεις χρησιμοποιώντας την συνάρτηση include σε κάθε κλάση. Στην συγκεκριμένη κλάση πραγματοποιείται ένας έλεγχος case της μεταβλητής-γεγονός $::osfamily για την εξακρίβωση του λειτουργικού που εκτελείται στον agent, ώστε να είναι στη συνέχεια δυνατή η ρύθμιση των ανάλογων παραμέτρων. Η μεταβλητή $ssh_package_name περιέχει το όνομα του πακέτου που πρόκειται να εγκατασταθεί, όπως η $ssh_service_config περιέχει την διαδρομή του αρχείου ρυθμίσεων της υπηρεσίας ssh, καθώς η μεταβλητή $ssh_service_name περιέχει το όνομα της υπηρεσίας.

ssh/manifests/params.pp

```
class ssh::params {
case $::osfamily {
Solaris: {
$ssh_package_name = 'cswopenssh'
$ssh_service_config = '/etc/opt/csw/ssh/sshd_config'
$ssh_service_name = 'cswopensshd'
}
Debian: {
$ssh_package_name = 'openssh-server'
$ssh_service_config = '/etc/ssh/sshd_config'
$ssh_service_name = 'sshd'
}
RedHat: {
$ssh_package_name = 'openssh-server'
$ssh_service_config = '/etc/ssh/sshd_config'
$ssh_service_name = 'sshd'
}
default: {
fail("Module ssh does not support osfamily: ${::osfamily}")
}
}
}
```

5.1.2 Η κλάση ssh::install

Στην κλάση ssh::install δηλώνεται ένα resource πακέτου (package), το οποίο διασφαλίζει την εγκατάσταση του πακέτου ssh. Ορίζεται η ιδιότητα name η οποία παίρνει τιμή ανάλογα με το λειτουργικό του agent από την κλάση ssh::params. Η ανάλογη τιμή θα χρησιμοποιηθεί από τον διαχειριστή πακέτων στο λειτουργικό του κάθε agent για την εγκατάσταση του επιθυμητού πακέτου (apt-get install openssh-server). Η ιδιότητα name θα λάβει την τιμή του τίτλου του resource, στην περίπτωση που δεν οριστεί ξεχωριστά στην περιοχή δηλώσεων των ιδιοτήτων.

ssh/manifests/install.pp

```
class ssh::install {
include ssh::params
package { 'ssh':
ensure => present,
name => $::ssh::params::ssh_package_name,
}
}
```

5.1.3 Η κλάση ssh::config

Ενώ στην κλάση ssh::config δηλώνεται ένα resource αρχείου (file) το οποίο διαχειρίζεται το αρχείο ρυθμίσεων της υπηρεσίας ssh. Εκεί ορίζονται ο owner, το group, τα δικαιώματα του αρχείου, όπως επίσης η source διαδρομή από την οποία θα μεταμορφωθεί το συγκεκριμένο αρχείο. Η διαδρομή δηλώνει ότι το αρχείο θα ανακτηθεί από τον master server με hostname puppet. Επίσης ακολουθεί η δήλωση δύο μεταπαραμέτρων, require και notify, οι οποίες χρησιμοποιούνται για την δημιουργία σχέσεων (relationships) ανάμεσα σε κλάσεις ή resources. Στην συγκεκριμένη περίπτωση ενημερώνεται το Puppet για την δημιουργία σχέσεων ανάμεσα στο resource αρχείου με όλα τα resources που εμπεριέχονται στις κλάσεις ssh::install και ssh::service. Που σημαίνει ότι πριν από την εφαρμογή του resource αρχείου, με το require δηλώνεται πως θα εφαρμοστούν πρώτα όλα τα resources της κλάσης ssh::install. Με τη δήλωση notice, το Puppet θα ενημερώσει όλα τα resources της κλάσης ssh::service, σε περίπτωση οποιαδήποτε αλλαγής στο resource αρχείου. Αυτό έχει σαν αποτέλεσμα οι αλλαγές που πραγματοποιούνται στο αρχείο ρυθμίσεων της υπηρεσίας ssh να συνοδεύονται από επανεκκίνηση της υπηρεσίας ssh, κάτι που είναι επιθυμητό.

ssh/manifests/config.pp

```
class ssh::config {
include ssh::params
file { ::$ssh::params::ssh_service_config:
ensure => present,
owner => 'root',
group => 'root',
mode => 0600,
source => "puppet:///modules/ssh/sshd_config",
require => Class["ssh::install"],
notify => Class["ssh::service"],
}
}
```

5.1.4 Η κλάση ssh::service

Η κλάση ssh::service περιέχει ένα resource υπηρεσίας (service), το οποίο ορίζει την διαχείριση της υπηρεσίας ssh, ενώ η ονομασία της λαμβάνεται μέσω της μεταβλητής $::ssh::params::ssh_service_name από την κλάση ssh::params. Η τιμή της ιδιότητας ensure διασφαλίζει ότι η υπηρεσία θα εκτελείται, καθώς με την ιδιότητα enable ενημερώνεται το λειτουργικό του agent, ώστε να προστεθεί η εν λόγω υπηρεσία στη λίστα υπηρεσιών που εκτελούνται κατά την έναρξη του συστήματος.

ssh/manifests/service.pp

```
class ssh::service {
include ssh::params
service { $::ssh::params::ssh_service_name:
ensure => running,
hasstatus => true,
hasrestart => true,
enable => true,
require => Class["ssh::config"],
}
}
```

5.1.5 Η κλάση ssh και τροποποίηση του αρχείου site.pp

Στο αρχείο init.pp, το βασικό αρχείο του module ssh, ορίζονται οι κλάσεις που αποτελούν μέρος του module ssh και οι μεταξύ τους εξαρτήσεις.

Τέλος στο main site.pp ορίζεται το νέο module στην κλάση base που δημιουργήθηκε προηγουμένως, η οποία εφαρμόζεται χρησιμοποιώντας το include στη δήλωση του κάθε agent node.

ssh/manifests/init.pp

```
class ssh {
class { '::ssh::install: } ->
class { '::ssh::config': } ->
class { '::ssh::service':} ->
Class['ssh']
}
/etc/puppetlabs/puppet/manifests/site.pp

class base {
include sudo
include ssh
```

```
}
node 'puppet.example.com' {
include base
}
node 'web.example.com' {
include base
}
node 'db.example.com' {
include base
}
node 'mail.example.com' {
include base
}
```

Ακολουθεί η δομή των αρχείων που δημιουργήθηκαν στον φάκελο /etc/puppet/modules:

```
root@puppet:/etc/puppet/modules# find.
.
./ssh
./ssh/Modulefile
./ssh/README
./ssh/manifests
./ssh/manifests/init.pp
./ssh/manifests/service.pp
./ssh/manifests/config.pp
./ssh/manifests/install.pp
./ssh/spec
./ssh/spec/spec_helper.rb
./ssh/tests
./ssh/tests/init.pp
./ssh/files
./ssh/files/sshd_config
```

5.2 Διαχείριση της υπηρεσίας postfix

```
root@puppet:/etc/puppet/modules# puppet module generate postfix
Notice: Generating module at /etc/puppet/modules/postfix
postfix
postfix/Modulefile
postfix/README
postfix/manifests
postfix/manifests/init.pp
postfix/spec
postfix/spec/spec_helper.rb
postfix/tests
postfix/tests/init.pp
root@puppet:/etc/puppet/modules# cd postfix
root@puppet:/etc/puppet/modules# mkdir files
root@puppet:/etc/puppet/modules# mkdir templates
root@puppet:/etc/puppet/modules# touch files/master.cf
root@puppet:/etc/puppet/modules# touch manifests/package.pp
root@puppet:/etc/puppet/modules# touch manifests/config.pp
root@puppet:/etc/puppet/modules# touch manifests/service.pp
root@puppet:/etc/puppet/modules# touch templates/main.cf.erb
```

5.2.1 Η κλάση postfix::install

Ο παρακάτω ορισμός διασφαλίζει την εγκατάσταση των πακέτων postfix και mailx.

postfix/manifests/install.pp

```
class postfix::install {
package { [ "postfix", "mailx" ]:
ensure => present,
}
}
```

5.2.2 Η κλάση postfix::config

Στην παρακάτω κλάση ορίζονται οι προκαθορισμένες ιδιότητες owner, group και mode για κάθε resource τύπου File σε ένα πεδίο, για αποφυγή της επανάληψης. Η ιδιότητα content σε συνδυασμό με την κλήση της συνάρτησης template επιτρέπει τη δημιουργία του αρχείου ρυθμίσεων main.cf από ένα πρότυπο, το οποίο περιγράφεται στο αρχείο main.cf.erb. Το συγκεκριμένο αρχείο χρησιμοποιεί ERB (perl) μεταβλητές για την ανάθεση τιμών στις μεταβλητές από τα Puppet γεγονότα. Κάθε μεταβλητή περικλείεται σε <%= %>, ενώ παίρνουν τις τιμές των μεταβλητών-γεγονότων όταν εκτελεστεί το puppet. Όσες μεταβλητές δεν περικλείονται σε <%= %>, όπως είναι το mydomain, αποτελούν μεταβλητές του postfix και δεν αλλάζουν κατά την μεταγλώττιση του puppet.

postfix/manifests/config.pp

```
class postfix::config {
File {
owner => 'postfix',
group => 'postfix',
mode => 0644,
}
file { '/etc/postfix/master.cf':
ensure => present,
source => 'puppet:///modules/postfix/master.cf',
require => Class['postfix::install'],
notify => Class['postfix::service'],
}
file { '/etc/postfix/main.cf':
ensure => present,
content => template('postfix/main.cf.erb'),
```

```
require => Class['postfix::install'],
notify => Class['postfix::service'],
}
}
postfix/templates/main.cf.erb

soft_bounce = no
command_directory = /usr/sbin
daemon_directory = /usr/libexec/postfix
mail_owner = postfix
myhostname = <%= @hostname %>
mydomain = <%= @domain %>
myorigin = $mydomain
mydestination = $myhostname, localhost.$mydomain, localhost, $mydomain
unknown_local_recipient_reject_code = 550
relay_domains = $mydestination
smtpd_reject_unlisted_recipient = yes
unverified_recipient_reject_code = 550
smtpd_banner = $myhostname ESMTP
setgid_group = postdrop
```

5.2.3 Η κλάση postfix::service

Δηλώνεται η κλάση postfix::service που είναι υπεύθυνη για τη διαχείριση της υπηρεσίας postfix.

```
class postfix::service {
service { 'postfix':
lensure => running,
hasstatus => true,
hasrestart => true,
enable => true,
require => Class['postfix::config'],
}
}
```

5.2.4 Η κλάση postfix και τροποποίηση του αρχείου site.pp

Παρακάτω ορίζεται η κλάση postfix, που είναι η κυρίως κλάση του module και περιέχει όλες τις κλάσεις που δηλώθηκαν προηγουμένως. Η δήλωση include postfix, έχει σαν αποτέλεσμα την εγκατάσταση και διαχείριση της υπηρεσίας postfix στον node “mail.example.com”

postfix/manifests/init.pp

```
class postfix {
include postfix::install
include postfix::config
include postfix::service
}
```

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

```
node 'mail.example.com' {
include base
include postfix
}
```

5.3 Διαχείριση της υπηρεσίας Apache και δημιουργία των virtual host

Με την εντολή puppet module generate δημιουργείται μια κενή δομή για το νέο module apache. Επίσης δημιουργούμε χειροκίνητα τους επιπλέον φακέλους και τα αρχεία που χρειάζονται, ενώ ακολουθεί η δήλωση των απαραίτητων κλάσεων και resources. Το module δημιουργείται για να λειτουργεί σε debian-based συστήματα.

```
root@puppet:/etc/puppet/modules# puppet module generate apache
Notice: Generating module at /etc/puppet/modules/apache
apache
apache/Modulefile
apache/README
apache/manifests
apache/manifests/init.pp
apache/spec
apache/spec/spec_helper.rb
apache/tests
apache/tests/init.pp
root@puppet:/etc/puppet/modules# mkdir apache/files
root@puppet:/etc/puppet/modules# mkdir apache/templates
root@puppet:/etc/puppet/modules# touch apache/manifests/install.pp
root@puppet:/etc/puppet/modules# touch apache/manifests/service.pp
root@puppet:/etc/puppet/modules# touch apache/manifests/vhost.pp
root@puppet:/etc/puppet/modules# touch apache/templates/vhost.conf.erb
```

```
class apache::install {
package { [ 'apache2' ]:
ensure => present,
}
}
```

```
class apache::service {
service { "apache2":
ensure => running,
hasstatus => true,
hasrestart => true,
enable => true,
require => Class['apache::install'],
}
}
```

Για να είναι δυνατή η δήλωση ξεχωριστών virtualhost για κάθε website του apache server θα χρησιμοποιηθεί ένας Puppet definition (ορισμός). Οι definitions είναι συλλογές από resources, όπως και οι κλάσεις αλλά με μια σημαντική διαφορά. Μπορούν να δηλωθούν/κληθούν πολλαπλές φορές για τον ίδιο host, ενώ δέχονται παραμέτρους.

```
define apache::vhost(
$docroot,
$port,
$priority,
$ssl=true,
$serveraliases = '',
$template='apache/vhost.conf.erb',
){
include apache
file {"/etc/apache2/sites-enabled/${priority}-${name}":
content => template($template),
owner => 'root',
group => 'root',
mode => '0640',
require => Class['apache::install'],
notify => Class['apache::service'],
}
}
```

Κατά τον ορισμό του definition δηλώνεται μια λίστα παραμέτρων και κάποιες από αυτές έχουν προκαθορισμένη τιμή. Οι τιμές όλων των παραμέτρων μπορούν να αλλάξουν κατά την κλήση/δήλωση του ορισμού. Για όσες παραμέτρους δεν έχει οριστεί τιμή, θα προκληθεί σφάλμα κατά τη μεταγλώττιση. Ο παραπάνω definition ξεκινάει με τη κλήση της κλάσης apache χρησιμοποιώντας το include. Με αυτόν τον τρόπο διασφαλίζεται η εφαρμογή του module apache πριν από κάθε δήλωση ενός vhost. Έτσι αποτρέπεται η περίπτωση να δηλωθεί ένας νέος apache vhost, χωρίς ο apache webserver να είναι εγκατεστημένος στο node ή να μην διαχειρίζεται από το Puppet.

Επιπλέον, δηλώνεται το resource τύπου αρχείου file {"/etc/apache2/sites-enabled/${priority}-${name}":}. Ο τίτλος του συγκεκριμένου resource κατασκευάζεται χρησιμοποιώντας τις μεταβλητές ${priority} και ${name}. Οι δύο μεταβλητές λαμβάνουν τιμή κατά την κλήση/δήλωση του definition. Συγκεκριμένα η μεταβλητή ${name} λαμβάνει τον τίτλο που ορίζουμε κατά την κλήση του νέου definition. Ακολουθεί ο Puppet κώδικας, ο οποίος εκτελείται για να δηλώσει έναν νέο definition τύπου apache::vhost.

```
apache::vhost { 'www.example.com':
port => '80',
docroot => '/var/www/www.example.com',
ssl => false,
priority => '10',
serveraliases => 'home.example.com',
}
```

Στο παραπάνω παράδειγμα η μεταβλητή ${name} λαμβάνει την τιμή ‘www.example.com’, καθώς η μεταβλητή ${priority} την τιμή ‘10’. Συνεπώς η διαδρομή του αρχείου που θα περιέχει την δήλωση του νέου vhost διαμορφώνεται ως εξής: “/etc/apache2/sites-enabled/10-www.example.com”.

Το Puppet επιτρέπει την ύπαρξη μόνο ενός resource(ίδιου τύπου) με το ίδιο όνομα σε ένα σύστημα. Στην περίπτωση που έχουν δηλωθεί δυο ή παραπάνω resources με το ίδιο όνομα για τον ίδιο node, προκαλείται σφάλμα κατά την μεταγλώττιση του καταλόγου και η εκτέλεση του Puppet θα διακοπεί. Το παραπάνω δεν συμβαίνει στην include-like περίπτωση καθώς η πολλαπλή δήλωση μιας κλάσης έχει ως αποτέλεσμα την εφαρμογή των resources που περιέχει μια και μόνο φορά. Για αυτόν τον λόγο οι κλάσεις θεωρούνται singletons, κάτι που δεν ισχύει με τους definitions. Επίσης η κλήση ενός definition με τις ίδιες παραμέτρους θα προκαλέσει σφάλμα. Κατά τον ορισμό του file resource χρησιμοποιείται η μεταβλητή $name ώστε να αποτραπεί μια τέτοια συμπεριφορά που θα οδηγούσε σε σφάλμα.

Παρακάτω παρουσιάζονται τα περιεχόμενα του ERB template.

apache/templates/vhost.conf.erb

```
NameVirtualHost *:<%= @port %>
<VirtualHost *:<%= @port %>>
ServerName <%= @name %>
<%if @serveraliases.is_a? Array -%>
<% @serveraliases.each do |name| -%><%= " ServerAlias #{@name}\n" %><%
end
-%>
<% elsif @serveraliases != '' -%>
<%= " ServerAlias #{@serveraliases}" -%>
<% end -%>
DocumentRoot <%= @docroot %>
<Directory <%= @docroot %>>
Options Indexes FollowSymLinks MultiViews
AllowOverride None
Order allow,deny
allow from all
</Directory>
```

```
ErrorLog /var/log/apache2/<%= @name %>_error.log
LogLevel warn
CustomLog /var/log/apache2/<%= @name %>_access.log combined
ServerSignature On
</VirtualHost>
```

Κάθε παράμετρος που ορίζεται κατά τη δήλωση του definition apache::vhost, συμπεριλαμβανομένου της μεταβλητής $name, χρησιμοποιείται στο ERB template. Επίσης χρησιμοποιείται και κώδικας γραμμένος σε Ruby ο οποίος φαίνεται παρακάτω:

```
<%if @serveraliases.is_a? Array -%>
<% @serveraliases.each do |name| -%><%= " ServerAlias #{@name}\n" %><%
end
-%>
<% elsif @serveraliases != '' -%>
<%= " ServerAlias #{@serveraliases}" -%>
<% end -%>
```

Η συμπλήρωση του template κατά την κλήση του παραπάνω definition με τις ακόλουθες παραμέτρους θα έχει ως αποτέλεσμα τη δημιουργία του αρχείου που θα διαβαστεί από τον apache, για να οριστεί ο νέος vhost “www.example.com”:

```
port => '80',
docroot => '/var/www/www.example.com',
ssl => false,
priority => '10',
serveraliases => 'home.example.com',
```

/etc/apache2/sites-enabled/10-www.example.com

```
NameVirtualHost *:80
<VirtualHost *:80>
ServerName www.example.com
ServerAlias home.example.com
DocumentRoot /var/www/www.example.com
<Directory /var/www/www.example.com>
Options Indexes FollowSymLinks MultiViews
AllowOverride None
Order allow,deny
allow from all
</Directory>
ErrorLog /var/log/apache2/www.example.com_error.log
LogLevel warn
CustomLog /var/log/apache2/www.example.com_access.log combined
ServerSignature On
</VirtualHost>
```

Έπειτα ορίζεται η τελευταία κλάση του module apache, στο init.pp αρχείο, η οποία περιλαμβάνει την κλάση εγκατάστασης της υπηρεσίας apache και την κλάση που διαχειρίζεται την υπηρεσία.

apache/manifests/init.pp

```
class apache {
include apache::install
include apache::service
}
```

ΚΕΦΑΛΑΙΟ 6

Hiera

Σε εκδόσεις του Puppet μεγαλύτερες του 3.0 περιλαμβάνεται το εργαλείο *Hiera*, το οποίο βρίσκεται build-in στον πυρήνα του Puppet. Η λέξη Hiera είναι συντομογραφία του Hierarchal data store, δηλαδή ενός τρόπου αποθήκευσης δεδομένων με ιεραρχική δομή. Η αρχική έκδοση του hiera δημιουργήθηκε το 2011 και ο σκοπός ανάπτυξής του ήταν ο διαχωρισμός του Puppet κώδικα, από τις παραμέτρους που αφορούσαν την ρύθμιση ενός Puppet host. Το πρόβλημα, δηλαδή η συνύπαρξη των configuration data με τον κώδικα Puppet, λύνεται όταν διαχωρίζονται τα δεδομένα ρυθμίσεων ενός host (configuration data) και αποθηκεύονται σε μια εξωτερική πηγή δεδομένων. Το Hiera παίζει τον ρόλο της εξωτερικής πηγής δεδομένων και αποθηκεύει σε ξεχωριστό μέρος τον Puppet κώδικα, τα δεδομένα και τις παραμέτρους. Η διαδικασία περιλαμβάνει:

- Εγκατάσταση του Hiera στον Puppet master.
- Ρύθμιση του Hiera (αρχείο hiera.yaml), με τη δημιουργία μιας ιεραρχικής δομής που θα καλύπτει στο σύνολο τους Puppet agent nodes.
- Δημιουργία και εμπλουτισμός ξεχωριστά για τον κάθε agent node αρχείου όπου θα δηλώνονται οι τιμές των παραμέτρων των module που χρησιμοποιεί.
- Δημιουργία ενός γενικού αρχείου ρυθμίσεων που θα λειτουργεί ως προκαθορισμένο στην περίπτωση που δεν οριστεί συγκεκριμένο αρχείο ρυθμίσεων για έναν node.

Η διαδικασία ανάκτησης από το Puppet μιας μεταβλητής που έχει δηλωθεί στο hiera, χαρακτηρίζεται ως ένα external lookup και πραγματοποιείται με τη χρήση συναρτήσεων που προσφέρει το hiera στο Puppet για αυτήν τη λειτουργία. Κατά την κλήση τους, το hiera μεταβιβάζει στον μεταγλωττιστή του Puppet τις τιμές των μεταβλητών που του ζητήθηκαν. Χαρακτηρίζεται ως external lookup γιατί “ψάχνει” την τιμή μιας μεταβλητής όχι εντός ενός manifest αρχείου, αλλά σε μια εξωτερική πηγή δεδομένων.

Το παρακάτω κεφάλαιο περιλαμβάνει την περιγραφή της διαδικασίας εγκατάστασης του hiera, όπως και τη ρύθμιση του, μέσω του αρχείου hiera.yaml. Θα αναλυθεί η διαδικασία δημιουργίας μιας ιεραρχικής δομής και των στοιχείων που την απαρτίζουν. Τέλος θα δημιουργηθεί ένα data store, όπου θα περιλαμβάνει τον ορισμό των παραμέτρων που χρειάζονται τα modules ενός agent node. Αρχικά να σημειωθεί γιατί ένα εργαλείο σαν το hiera είναι τόσο σημαντικό.

Όπως αναφέρθηκε προηγουμένως το hiera επιτρέπει τον διαχωρισμό των παραμέτρων που χρειάζονται για την ρύθμιση ενός host από τα manifest αρχεία, δηλαδή από τον Puppet κώδικα. Οι κλάσεις του Puppet μπορούν να ζητήσουν τα δεδομένα που χρειάζονται και το hiera να τους τα παρέχει, ενεργώντας ως ένα αρχείο ρυθμίσεων προσβάσιμο από όλες τις κλάσεις (site-wide config file).

Επιτρέπει:

- Την εύκολη διαχείριση νέων nodes, αποτρέποντας την επανάληψη των ίδιων παραμέτρων με την χρήση προκαθορισμένων τιμών για τους περισσότερους nodes. Για όσους nodes χρειάζονται διαφορετικές τιμές από τις προκαθορισμένες πραγματοποιείται στοχευόμενη και δομημένη αντικατάσταση των παραμέτρων.
- Ευκολία στη χρήση και εφαρμογή νέων module που βρίσκονται διαθέσιμα στο διαδίκτυο για λήψη. Δεν τροποποιείται ο Puppet κώδικας και απλά προσθέτονται τα δεδομένα που χρειάζονται οι nodes στο hiera.
- Ευκολία στην δημιουργία και διαμοίραση στο διαδίκτυο νέων modules καθώς περιλαμβάνεται μόνο κώδικας Puppet. Δεν χρειάζεται ο προγραμματιστής να αφαιρέσει από τον κώδικα τιμές παραμέτρων πριν τον δημοσιεύσει στο διαδίκτυο.

6.1 Εγκατάσταση του hiera

Για την χρήση του hiera με το Puppet προτείνεται η εγκατάσταση του μόνο στον Puppet master εξυπηρετητή. Η εγκατάσταση του στους agent nodes είναι προαιρετική και μη απαραίτητη.

Προαπαιτήσεις:

- Σύστημα που βασίζεται σε unix, linux ή windows.
- Ruby έκδοσης 1.8.5 ή μεταγενέστερη.
- Για την συνεργασία του hiera με το Puppet, έκδοση Puppet 2.7 ή μεταγενέστερη.
- Σε χρήση με την έκδοση Puppet 2.7, εγκατάσταση του πακέτου hiera-puppet.

Εκτελείται η παρακάτω εντολή για εγκατάσταση του hiera.

Για εκδόσεις του Puppet > 2.7:

```
$ sudo puppet resource package hiera ensure=installed
```

Για την έκδοση Puppet 2.7 εκτελείται επιπλέον η εντολή:

```
$ sudo puppet resource package hiera-puppet ensure=installed
```

6.2 Το configuration αρχείο hiera.yaml

Το αρχείο hiera.yaml περιέχει διάφορες ρυθμίσεις σχετικά με την ιεραρχία των πηγών δεδομένων, το backend που θα χρησιμοποιηθεί (yaml ή json) και προσαρμοσμένες ρυθμίσεις που αφορούν το κάθε backend αντίστοιχα. Με τον όρο backend, ορίζεται το format που θα χρησιμοποιηθεί για την περιγραφή των δεδομένων και οι επιλογές είναι δύο, yaml ή json. Το αρχείο ρυθμίσεων hiera.yaml πρέπει πάντοτε να υπάρχει έστω και κενό, γιατί η εκτέλεση του hiera χωρίς αυτό θα έχει ως αποτέλεσμα την δημιουργία σφάλματος. Το config αρχείο βρίσκεται σε διαφορετικές διαδρομές στον δίσκο, αναλόγως με τον τρόπο κλήσης του hiera. Όταν εκτελείται μέσω του Puppet η προκαθορισμένη διαδρομή του αρχείου ρυθμίσεων είναι μια από τις ακόλουθες:

- /etc/puppet/hiera.yaml σε περιβάλλον *nix (unix ή linux) με εγκατεστημένο το Puppet open source
- /etc/puppetlabs/puppet/hiera.yaml σε περιβάλλον *nix (unix ή linux) με εγκατεστημένο το Puppet Enterprise
- COMMON_APPDATA\PuppetLabs\puppet\etc\hiera.yaml σε περιβάλλον Windows

Σε μεταγενέστερες εκδόσεις του Puppet 3 υπάρχει η επιλογή δήλωσης διαφορετικής διαδρομής του αρχείου config με την τροποποίηση της τιμής hiera_config στο γενικό αρχείο ρυθμίσεων puppet.conf.

Το hiera μπορεί επίσης να εκτελεστεί μέσω τερματικού σε περιβάλλοντα *nix ή μέσω του cmd στα windows. Εκεί οι αντίστοιχες διαδρομές του αρχείου είναι:

- /etc/hiera.yaml σε περιβάλλον *nix
- COMMON_APPDATA\PuppetLabs\hiera\etc\hiera.yaml σε περιβάλλον Windows

Τέλος, υπάρχει η δυνατότητα κλήσης του hiera μέσω κώδικα ruby όπου οι διαδρομές του αρχείου ρυθμίσεων παραμένουν ίδιες με αυτές κατά την εκτέλεση από το τερματικό.

6.3 Η δομή του αρχείου ρυθμίσεων hiera.yaml

Το αρχείο ρυθμίσεων του hiera είναι γραμμένο σε format yaml. Το yaml είναι ένας δομημένος τρόπος σύνταξης δεδομένων με σειριακή μορφή, κατανοητός από τον άνθρωπο.

Όταν το αρχείο ρυθμίσεων βρίσκεται στην διαδρομή που πρέπει όμως είναι κενό, το hiera το αντιλαμβάνεται και φορτώνει τις προκαθορισμένες ρυθμίσεις οι οποίες είναι οι ακόλουθες:

```
---
:backends: yaml
:yaml:
  :datadir: /var/lib/hiera
:hierarchy: common
:logger: console
```

Ένα άλλο πολύ κοινό παράδειγμα είναι το παρακάτω:

```
---
:backends:
  - yaml
  - json
:yaml:
  :datadir: /etc/puppet/hieradata
:json:
  :datadir: /etc/puppet/hieradata
:hierarchy:
  - "%{::clientcert}"
  - "%{::custom_location}"
  - common
```

Ακολουθεί η επεξήγηση των καθολικών ρυθμίσεων που μπορεί να περιέχει το αρχείο ρυθμίσεων hiera.yaml. Αν μια ρύθμιση από τις παρακάτω απουσιάζει, λαμβάνει την προκαθορισμένη της τιμή. Κάθε ρύθμιση ξεκινά υποχρεωτικά με το σύμβολο της Ruby “:”.

6.3.1 :hierarchy

Δηλώνεται μια συμβολοσειρά ή ένας πίνακας συμβολοσειρών, όπου κάθε στοιχείο του πίνακα είναι το όνομα μιας πηγής δεδομένων. Οι πηγές δεδομένων διαχωρίζονται σε στατικές και δυναμικές. Όποιες δηλώνονται στην ιεραρχία, ελέγχονται από την αρχή προς το τέλος. Ως προκαθορισμένη τιμή έχει οριστεί το “common”. Όπως δηλώνει η λέξη, είναι μια κοινή πηγή δεδομένων για όλους τους nodes.

```
# /etc/puppet/hiera.yaml
---
:hierarchy:
  - "%{::clientcert}"
  - "%{::environment}"
  - "virtual_%{::is_virtual}"
  - common
```

6.3.1.1 Στατική πηγή δεδομένων

Είναι στοιχείο της ιεραρχίας που δεν περιέχει *interpolation tokens*. Μια στατική πηγή δεδομένων χρησιμοποιείται καθολικά από όλους τους nodes. Στο παραπάνω παράδειγμα το “common” είναι μια στατική πηγή δεδομένων η οποία θα χρησιμοποιηθεί στο σύνολο των nodes. Η πηγή δεδομένων common θα χρησιμοποιηθεί και από έναν node με hostname “webserver1” και από έναν node με hostname “dbserver1”.

6.3.1.2 Δυναμική πηγή δεδομένων

Είναι στοιχείο της ιεραρχίας που περιέχει τουλάχιστον ένα *interpolation token*. Ως *interpolation token* ορίζεται η έκλυση μιας μεταβλητής ή μιας συνάρτησης σε “%{}”, δηλαδή %{variable} ή %{function(“input”)}. Το hiera θα αντικαταστήσει κατά την εκτέλεση του, το token με την τιμή της μεταβλητής ή την έξοδο της συνάρτησης. Έχει ως αποτέλεσμα την χρήση διαφορετικών πηγών δεδομένων από τους nodes, όταν οι τιμή της μεταβλητής διαφέρει ανάμεσα στους nodes. Η μεταβλητή $::clientcert που χρησιμοποιείται στο παραπάνω παράδειγμα είναι μοναδική σε κάθε node. Περιέχει την τιμή του hostname που χρησιμοποιήθηκε στην υπογραφή του πιστοποιητικού κατά την σύνδεση του agent node με τον master . Για παράδειγμα, ένας agent node με hostname “web01”, έχει ως πρώτη προτεραιότητα την πηγή δεδομένων “web01”, όπως ένας άλλος node με hostname “db01” έχει την πηγή δεδομένων “db01”.

6.3.2 :backends

Δηλώνεται μια συμβολοσειρά ή ένας πίνακας συμβολοσειρών, όπου κάθε στοιχείο του πίνακα είναι το όνομα ενός διαθέσιμου Hiera backend. Το Hiera προσφέρει δύο backends που παρέχονται μαζί με την εγκατάσταση του, το yaml και το json αλλά υπάρχει η δυνατότητα προσθήκης νέων backends υπό τη μορφή add-ons. Η λίστα των backends επεξεργάζεται από το hiera σειριακά. Αυτό σημαίνει ότι στο προηγούμενο παράδειγμα όπου δηλώνονται τα yaml και json ως backends, το hiera ελέγχει πρώτα όλα τα αρχεία με κατάληξη yaml και έπειτα τα αρχεία json. Ως προκαθορισμένη τιμή έχει οριστεί το “yaml” backend.

6.3.3 :logger

Δηλώνεται το όνομα ενός logger ως συμβολοσειρά. Οι Loggers διαχειρίζονται τη διαδρομή που θα ακολουθήσουν τα μηνύματα ελέγχου, προειδοποίησης ή σφάλματος. Το hiera παρέχει με την εγκατάσταση του, τους παρακάτω loggers:

- console - Τα μηνύματα κατευθύνονται στο STDERR
- puppet - Τα μηνύματα κατευθύνονται στο σύστημα καταγραφών του Puppet.
- noop - Τα μηνύματα σιωπούν, δεν κατευθύνονται πουθενά.

Ως προκαθορισμένη τιμή ορίζεται το “console”. Στην περίπτωση που το hiera εκτελείται μέσω του Puppet, το Puppet αλλάζει την προκαθορισμένη τιμή του :logger σε “puppet” ακόμη κι αν έχει οριστεί διαφορετική τιμή στο αρχείο ρυθμίσεων.

6.3.4 :merge_behavior

Είναι μια καθολική ρύθμιση για όλους τους nodes, όπου ορίζεται ο τρόπος με τον οποίο το hiera θα συγχωνεύσει τις τιμές των δηλωμένων παραμέτρων ανάλογα με την ιεραρχία που ορίστηκε (:hierarchy). Χρησιμοποιείται στην περίπτωση ορισμού διαφορετικών τιμών των ίδιων παραμέτρων (κλειδιά), σε διαφορετικά επίπεδα της ιεραρχίας. Μπορεί να έχει μια από τις παρακάτω τιμές:

- native - Συγχώνευση των top-level κλειδιών μόνο. Πραγματοποιείται έλεγχος στην ιεραρχία με βάση την προτεραιότητα και επιλέγεται το πρώτο κλειδί.
- deep – Αναδρομική συγχώνευση. Στην περίπτωση αλληλοσυγκρουόμενων κλειδιών, επιλέγονται οι τιμές με τη **χαμηλότερη προτεραιότητα**. Είναι μια στρατηγική που δεν συνιστάται γιατί αυξάνει την πολυπλοκότητα και μπορεί να μην έχει τα επιδιωκόμενα αποτελέσματα.
- deeper - Αναδρομική συγχώνευση. Στην περίπτωση αλληλοσυγκρουόμενων κλειδιών, επιλέγονται οι τιμές με την **υψηλότερη προτεραιότητα**.

Προκαθορισμένη τιμή είναι το “native”. Για την χρήση οποιασδήποτε άλλης τιμής, απαιτείται η εγκατάσταση του Ruby gem “deep_merge”.

6.3.4.1 Παράδειγμα

1. Η χρήστης ‘jen’ ορίζεται μόνο στον host με όνομα ‘officepc’.
2. Ο χρήστης ‘bob’ ορίζεται σε όλους τους hosts, αλλά έχει διαφορετικό uid στον υπολογιστή ‘officepc’, από τους υπόλοιπους hosts.
3. Ο χρήσης ‘ash’ ορίζεται παντού με συγκεκριμένο, σταθερό ‘uid’ και ‘shell’.

Στο αρχείο hiera.yaml δηλώνεται μια ιεραρχία δύο επιπέδων, όπως φαίνεται παρακάτω:

```
# /etc/puppet/hiera.yaml
    ---
    :backends:
      - yaml
    :logger: puppet
    :hierarchy:
      - "%{hostname}"
      - common
    :yaml:
      :datadir: /etc/puppet/hieradata
    # options are native, deep, deeper
    :merge_behavior: deeper
```

Στο αρχείο common.yaml δηλώνονται οι προκαθορισμένοι χρήστες για όλους τους nodes.

```
# /etc/puppet/hieradata/common.yaml
 ---
    site_users:
      bob:
        uid: 501
        shell: /bin/bash
      ash:
        uid: 502
        shell: /bin/zsh
        group: common
```

Ενώ στο αρχείο officepc.yaml, προσδιορίζονται οι πληροφορίες των χρηστών που είναι συγκεκριμένες για αυτόν τον node.

```
# /etc/puppet/hieradata/officepc.yaml
    ---
    site_users:
      jen:
        uid: 503
        shell: /bin/zsh
        group: officepc
      bob:
        uid: 1000
        group: officepc
```

Με τον τρόπο συγχώνευσης *native* παράγεται το παρακάτω αποτέλεσμα. Λόγω του τρόπου συγχώνευσης και της ιεραρχίας που δηλώθηκε λείπει η παράμετρος shell από τον χρήστη bob, καθώς η τιμή του top-level κλειδιού bob από το common.yaml, αντικαταστάθηκε εξ ολοκλήρου από το officepc.yaml.

```
{
  "bob"=>{
    group=>"officepc",
    uid=>1000,
  },
  "jen"=>{
    group=>"officepc",
    uid=>503
    shell=>"/bin/zsh",
  },
  "ash"=>{
    group=>"common",
    uid=>502,
    shell=>"/bin/zsh"
  }
}
```

Με τον τρόπο συγχώνευσης *deeper* παράγεται το παρακάτω αποτέλεσμα. Στην περίπτωση αυτή, η παράμετρος shell παραμένει από το common.yaml, αλλά επιτρέπεται στο officepc.yaml να αντικαταστήσει τις τιμές των παραμέτρων uid και group, λόγω της υψηλότερης ιεραρχίας. Με αυτόν τον τρόπο μειώνονται οι διπλότυπες δηλώσεις παραμέτρων ανάμεσα στα yaml αρχεία, ενώ βασικό ρόλο έχει η ιεραρχία.

```
{
  "bob"=>{
    group=>"officepc",
    uid=>1000,
    shell=>"/bin/bash"
  },
  "jen"=>{
    group=>"officepc",
    uid=>503
    shell=>"/bin/zsh",
  },
  "ash"=>{
    group=>"common",
    uid=>502,
    shell=>"/bin/zsh"
  }
}
```

Τέλος, με την συγχώνευση *deep* παράγεται το παρακάτω αποτέλεσμα. Στην παρακάτω περίπτωση το officepc.yaml καταφέρνει να θέσει την τιμή της παραμέτρου group επειδή δεν είχε δηλωθεί στο common.yaml. Αλλά όπου υπήρχε σύγκρουση, όπως στη παράμετρο uid, το common.yaml νίκησε. Αυτή είναι μια ανεπιθύμητη συμπεριφορά από τους περισσότερους χρήστες και γι' αυτόν το λόγο **προτείνεται να αποφεύγεται ο τρόπος συγχώνευσης *deep***.

```
{
  "bob"=>{
    group=>"officepc",
    uid=>501,
    shell=>"/bin/bash"
  },
  "jen"=>{
    group=>"officepc",
    shell=>"/bin/zsh",
    uid=>503
  },
  "ash"=>{
    group=>"common",
    uid=>502,
    shell=>"/bin/zsh"
  }
}
```

6.4 Ιεραρχία μεταξύ των πηγών δεδομένων

6.4.1 Ταξινόμηση

Κάθε στοιχείο της ιεραρχίας αντιστοιχεί με μια πηγή δεδομένων. Το hiera ελέγχει κάθε πηγή δεδομένων με την σειρά που έχει οριστεί στο hiera.yaml, ξεκινώντας από την αρχή.

- Όταν μια πηγή δεδομένων έχει οριστεί στο αρχείο ρυθμίσεων, αλλά δεν υπάρχει ως φυσικό αρχείο, το hiera ελέγχει την αμέσως επόμενη πηγή δεδομένων.
- Όταν η πηγή δεδομένων υπάρχει, αλλά δεν περιέχει τα ζητούμενα δεδομένα, το hiera προχωράει σε έλεγχο της αμέσως επόμενης πηγής δεδομένων.
- Όταν βρεθεί η ζητούμενη μεταβλητή/κλειδί, το hiera πραγματοποιεί ανάκτηση της τιμής του κλειδιού. Οι τρόποι ανάκτησης που προσφέρει το hiera είναι οι παρακάτω τρεις:
    - **Normal priority lookup.** Το hiera σταματάει την αναζήτηση στην πρώτη πηγή δεδομένων που περιέχει τα ζητούμενα δεδομένα, και επιστρέφει την τιμή τους.
    - **Array lookup.** Το hiera συνεχίζει την αναζήτηση ενός κλειδιού σε όλες τις πηγές δεδομένων και επιστρέφει τις τιμές που βρίσκει σε μορφή πίνακα. Οι τιμές που ανακτήθηκαν, δηλαδή τα στοιχεία του πίνακα είναι ιεραρχικά ταξινομημένα. Αυτό σημαίνει, πως οι τιμές με

μεγαλύτερη προτεραιότητα στην αναζήτηση βρίσκονται στα πρώτα στοιχεία του πίνακα.

- **Hash lookup.** Το hiera συνεχίζει την αναζήτηση ενός κλειδιού σε όλες τις πηγές δεδομένων και προσδοκά οι τιμές των κλειδιών να είναι **hashes.** Στη περίπτωση που κάποια από τις τιμές δεν είναι hash, παράγεται σφάλμα εξόδου. Έπειτα συγχωνεύει τις τιμές hash με ιεραρχική σειρά βάσει προτεραιότητας, όπου παράγεται μια τιμή hash η οποία επιστρέφεται ως το αποτέλεσμα.

6.4.2 Πολλαπλά Backends

Το hiera παρέχει την δυνατότητα δήλωσης πολλαπλών backends στο αρχείο ρυθμίσεων hiera.yaml. Κάθε νέο backend λειτουργεί ως μια καινούρια ιεραρχία. Το hiera δίνει προτεραιότητα στο πρώτο backend, όπου ελέγχεται κάθε επίπεδο της ιεραρχία του. Έπειτα προχωράει στο δεύτερο backend. Έστω ότι το αρχείο hiera.yaml περιέχει τα παρακάτω:

```
---
:backends:
  - yaml
  - json
:hierarchy:
  - one
  - two
  - three
```

Το hiera θα ελέγξει τις πηγές δεδομένων με την ακόλουθη σειρά:

- one.yaml
- two.yaml
- three.yaml
- one.json
- two.json
- three.json

6.4.3 Παράδειγμα βάσει μιας δεδομένης ιεραρχίας

Έστω ότι το αρχείο ρυθμίσεων hiera.yaml περιέχει τα παρακάτω. Η ιεραρχία βασίζεται σε τρεις μεταβλητές-γεγονότα που διαθέτει κάθε agent node, ::clientcert, ::environment, ::is_virtual και στην στατική πηγή δεδομένων common που είναι κοινή για όλους τους nodes:

```
# /etc/puppet/hiera.yaml
---
:hierarchy:
  - "%{::clientcert}"
  - "%{::environment}"
  - "virtual_%{::is_virtual}"
  - common
```

Επίσης υποθέτουμε ότι τα αρχεία που λειτουργούν ως πηγές δεδομένων είναι τα παρακάτω, τα οποία βρίσκονται στον φάκελο “/etc/puppet/hieradata”:

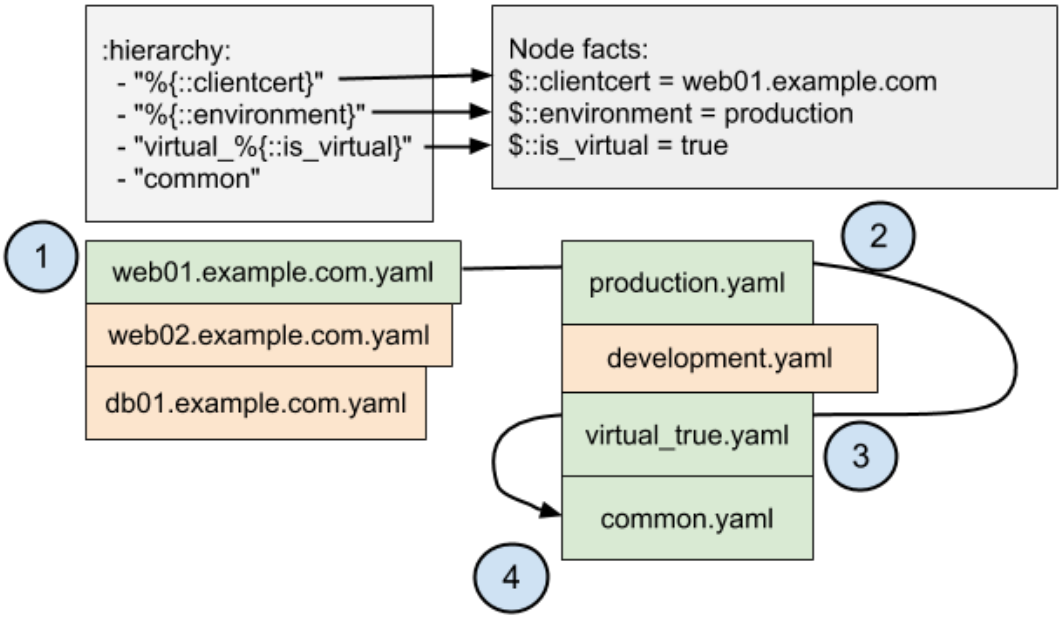
- web01.example.com
- web02.example.com
- db01.example.com
- production.yaml
- development.yaml
- virtual_true.yaml
- common.yaml

Για τον node με hostname “web01.example.com” ορίζονται οι παρακάτω μεταβλητές-γεγονότα:

- ::clientcert = web01.example.com
- ::environment = production
- ::is_virtual = true

Στην Εικόνα 6.1 φαίνεται η σειρά αναζήτησης των κλειδιών με βάση την ιεραρχία που βασίζεται στα γεγονότα που έχουν τεθεί, η οποία διαμορφώνεται ως εξής:

- web01.example.com.yaml
- production.yaml
- virtual_true.yaml
- common.yaml



Εικόνα 6.1

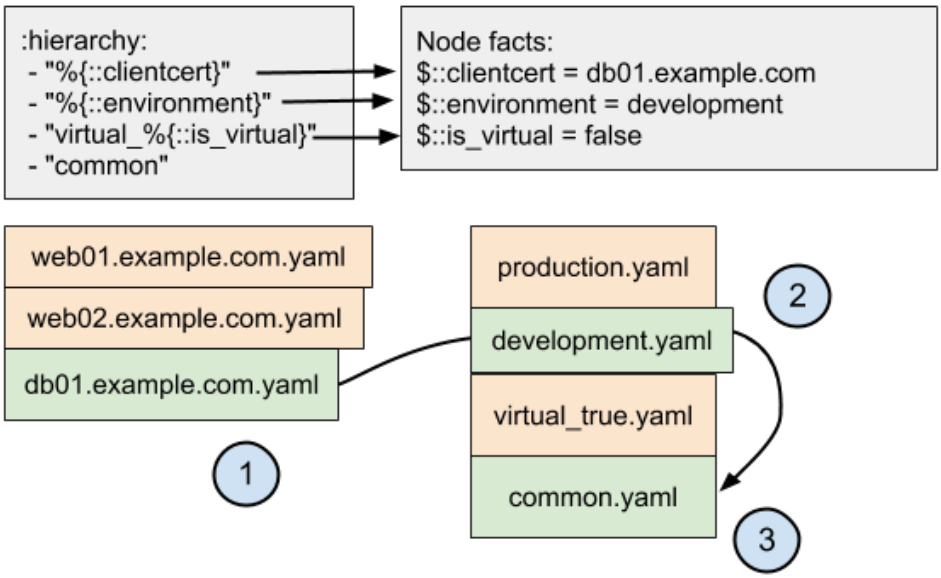
Για τον node με hostname “db01.example.com” ορίζονται οι παρακάτω τα παρακάτω μεταβλητές-γεγονότα:

1. ::clientcert = db01.example.com
2. ::environment = development
3. ::is_virtual = false

Αντίστοιχα, στην εικόνα 6.2 φαίνεται η σειρά αναζήτησης των κλειδιών με βάση την ιεραρχία που βασίζεται στα γεγονότα που έχουν τεθεί, η οποία διαμορφώνεται ως εξής:

1. db01.example.com.yaml
2. development.yaml
3. common.yaml

Το αρχείο virtual_false.yaml δεν υπάρχει στις πηγές δεδομένων οπότε παρακάμπτεται.



Εικόνα 6.2

6.5 Ολοκληρωμένο παράδειγμα τροποποίησης υπάρχοντος module για την αξιοποίηση του με το Hiera

Στο παρακάτω παράδειγμα θα χρησιμοποιηθεί το Puppet Labs ntp module (http://forge.puppetlabs.com/puppetlabs/ntp). Η κλάση ntp δέχεται πέντε παραμέτρους, οι περισσότερες από τις οποίες αντικατοπτρίζουν αποφάσεις που πρέπει να ληφθούν για κάθε agent node που θα χρησιμοποιήσει την υπηρεσία ntp.

- servers
- restrict
- autoupdate
- enable
- template

Η παράμετρος restrict αποτρέπει ή επιτρέπει την δραστηριοποίηση ενός node ως time server για άλλους hosts. Με την παράμετρο servers ορίζονται οι εξυπηρετητές με τους οποίους θα συγχρονίσει η ntp υπηρεσία. Ενώ η παράμετρος

autoupdate επιτρέπει την αυτόματη ενημέρωση της υπηρεσίας από το Puppet, αν αυτό είναι επιθυμητό.

Χωρίς τη χρήση του hiera ίσως χρειαστεί η προσθήκη προκαθορισμένων τιμών για τις παραμέτρους αυτές, κάτι που έχει επιπτώσεις στην επαναχρησιμοποίηση και στην δυνατότητα διαμοιρασμού του κώδικα. Επίσης, όταν υπάρχουν μικρές διαφορές στις παραμέτρους ανάμεσα στους nodes, αυξάνεται η επανάληψη κατά την δήλωση των κλάσεων στο site manifest αρχείο.

6.5.1 Περιγραφή του περιβάλλοντος

Ας υποθέσουμε ότι μια επιχείρηση έχει έναν αριθμό από ntp server οι οποίοι έχουν συγκεκριμένες ιδιαιτερότητες:

- Δύο ntp servers έχουν δικαίωμα να επικοινωνούν με time servers εκτός της επιχείρησης και χαρακτηρίζονται primary.
- Ένας εκ των δύο primary, είναι υψηλού ρίσκου και δεν του επιτρέπεται να πραγματοποιεί αυτόματες ενημερώσεις του πακέτου ntp, χωρίς να προηγηθούν δοκιμές.
- Οι υπόλοιποι ntp servers της επιχείρησης λαμβάνουν τα δεδομένα που χρειάζονται από τους δύο primary.

Η κατάσταση του περιβάλλοντος πριν το hiera, περιλαμβάνει δήλωση των παραμέτρων της κλάσης ntp για κάθε agent node στο κυρίως site.pp manifest (/etc/puppet/manifests/site.pp). Το site.pp αρχείο περιέχει τα παρακάτω:

```
node "kermit.example.com" {
      class { "ntp":
        servers    => [ '0.us.pool.ntp.org iburst','1.us.pool.ntp.org
iburst','2.us.pool.ntp.org iburst','3.us.pool.ntp.org iburst'],
        autoupdate => false,
        restrict   => [],
        enable     => true,
      }
    }

    node "grover.example.com" {
      class { "ntp":
        servers    => [ 'kermit.example.com','0.us.pool.ntp.org
iburst','1.us.pool.ntp.org iburst','2.us.pool.ntp.org iburst'],
        autoupdate => true,
```

```
        restrict   => [],
        enable     => true,
      }
    }

    node "snuffie.example.com", "bigbird.example.com",
"hooper.example.com" {
      class { "ntp":
        servers    => [ 'grover.example.com', 'kermit.example.com'],
        autoupdate => true,
        enable     => true,
      }
    }
```

6.5.2 Δημιουργία ιεραρχίας σχετική με το περιβάλλον

Τροποποιείται το αρχείο ρυθμίσεων hiera.yaml ώστε να περιέχει τα παρακάτω:

```
---
:backends:
  - yaml
:yaml:
  :datadir: /etc/puppet/hieradata
:hierarchy:
  - "node/%{::fqdn}"
  - common
```

Ως backend ορίζεται το yaml και οι πηγές δεδομένων αποθηκεύονται στον φάκελο /etc/puppet/hieradata. Έπειτα δηλώνεται μια δυναμική πηγή δεδομένων με βάση το fully qualified domain name (fqdn) των agent node, υπό τον φάκελο node/. Όταν για παράδειγμα είναι επιθυμητό να δημιουργηθεί μια πηγή δεδομένων, πρώτη στην ιεραρχία για τον node με $::fqdn=grover.example.com, δημιουργείται και εμπλουτίζεται το αρχείο /etc/puppet/hieradata/node/grover.example.com.yaml. Για όσους nodes δεν έχουν οριστεί πηγές δεδομένων με βάση το fqdn τους, το hiera απλώς θα προχωρήσει στο επόμενο επίπεδο της ιεραρχίας, που στο παράδειγμα είναι το common.

Η πηγή δεδομένων common (/etc/puppet/hieradata/common.yaml), παρέχει τις κοινές ή προκαθορισμένες τιμές των παραμέτρων. Όσες παράμετροι οριστούν σε υψηλότερο επίπεδο της ιεραρχίας, θα αντικατασταθούν ενώ όσες δεν οριστούν θα έχουν τις προκαθορισμένες τιμές που ορίζονται στο common.yaml, της τελευταίας σε ιεραρχία πηγής δεδομένων.

6.5.3 Δημιουργία των πηγών δεδομένων

Οι πέντε παράμετροι της κλάσης ntp εξηγούνται παρακάτω:

- **servers** - Πίνακας που περιέχει time servers, που είναι αρχικά μη ορισμένος. Στη συνέχεια λαμβάνει τιμή έπειτα από ελέγχους που πραγματοποιούνται στον agent node.
- **restrict** - Πίνακας που περιέχει ορίσματα διαφορετικών περιορισμών, αναλόγως το λειτουργικό σύστημα των agent nodes.
- **autoupdate** - Τιμή boolean που δηλώνει αν επιτρέπεται η αυτόματη ενημέρωση του πακέτου ntp. Εξ ορισμού έχει οριστεί σε false.
- **enable** - Τιμή Boolean που δηλώνει αν η υπηρεσία ntp θα ξεκινάει με την εκκίνηση του συστήματος. Εξ ορισμού έχει οριστεί σε true.
- **template** - Η διαδρομή του προτύπου που θα χρησιμοποιήσει το Puppet για την ρύθμιση της υπηρεσίας ntp.

Γνωρίζοντας τις παραμέτρους που δέχεται η κλάση ntp αρκεί να παρθούν αποφάσεις σχετικές με το περιβάλλον που περιγράφηκε στην αρχή του υποκεφαλαίου. Έστω ότι οι δύο nodes που τους επιτρέπεται η επικοινωνία με time servers εκτός της επιχείρησης είναι οι kermit.example.com και grover.example.com. Επιθυμούμε ο kermit να έχει επικοινωνία με servers εκτός περιβάλλοντος, να μην πραγματοποιείται αυτόματη αναβάθμιση της υπηρεσίας ntp και η υπηρεσία να ξεκινά κατά την εκκίνηση του συστήματος.

Παρακάτω φαίνονται τα περιεχόμενα του αρχείου που λειτουργεί ως πηγή δεδομένων για τον kermit node.

```
#/etc/puppet/hieradata/node/kermit.example.com.yaml
---
ntp::restrict:
  -
ntp::autoupdate: false
ntp::enable: true
ntp::servers:
  - 0.us.pool.ntp.org iburst
  - 1.us.pool.ntp.org iburst
  - 2.us.pool.ntp.org iburst
  - 3.us.pool.ntp.org iburst
```

Στον grover node επιτρέπεται η αυτόματη ενημέρωση του πακέτου ntp, ενώ δηλώνεται ο kermit στην λίστα των εξυπηρετητών. Τα περιεχόμενα του αρχείου που έχει τον ρόλο της πηγής δεδομένων φαίνονται παρακάτω:

```
#/etc/puppet/hieradata/node/grover.example.com.yaml
---
ntp::restrict:
  -
ntp::autoupdate: true
ntp::enable: true
ntp::servers:
  - kermit.example.com iburst
  - 0.us.pool.ntp.org iburst
  - 1.us.pool.ntp.org iburst
  - 2.us.pool.ntp.org iburst
```

Τέλος, για τους υπόλοιπους nodes της επιχείρησης είναι επιθυμητή η δήλωση των kermit και grover ως εξυπηρετητές συγχρονισμού. Η ιεραρχία που δημιουργήσαμε προηγουμένως μας επιτρέπει την χρήση της πηγής δεδομένων common.yaml για τον σκοπό αυτό. Η συγκεκριμένη πηγή βρίσκεται δεύτερη στην ιεραρχία, που σημαίνει ότι χρησιμοποιείται από τους nodes, για τους οποίους δεν έχει δημιουργηθεί πηγή δεδομένων με βάση το fqdn τους. Τα περιεχόμενα του αρχείου common.yaml φαίνονται παρακάτω:

```
#/etc/puppet/hieradata/node/common.yaml
---
ntp::autoupdate: true
ntp::enable: true
ntp::servers:
  - grover.example.com iburst
  - kermit.example.com iburst
```

6.5.4 Ρύθμιση του hiera για χρήση του σε τερματικό και έλεγχος των πηγών δεδομένων

Για την ρύθμιση του command-line εργαλείου που προσφέρει το hiera δημιουργείται ένα softlink μεταξύ του αρχείου /etc/puppet/hiera.yaml και του αρχείου /etc/hiera.yaml εκτελώντας την παρακάτω εντολή:

```
# ln -s /etc/puppet/hiera.yaml /etc/hiera.yaml
```

Στη συνέχεια πραγματοποιείται έλεγχος της μεταβλητής ntp::servers για τους δύο nodes kermit και grover, δίνοντας ως όρισμα τη μεταβλητή-γεγονός ::fqdn. Με αυτόν τον τρόπο προσομοιώνεται και επιβεβαιώνεται η ορθή εκτέλεση των lookup που πραγματοποιεί το hiera ώστε να πραγματοποιήσει ανάκτηση των ζητούμενων μεταβλητών. Οπότε, για τον kermit node εκτελείται η παρακάτω εντολή:

```
$ hiera ntp::servers ::fqdn=kermit.example.com
```

Η οποία έχει ως αποτέλεσμα την τιμή της μεταβλητής ntp::servers για τον node με ::fqdn=kermit.example.com:

```
["0.us.pool.ntp.org iburst", "1.us.pool.ntp.org iburst",
"2.us.pool.ntp.org iburst", "3.us.pool.ntp.org iburst"]
```

Για τον node grover:

```
$ hiera ntp::servers ::fqdn=grover.example.com
["kermit.example.com iburst", "0.us.pool.ntp.org iburst",
"1.us.pool.ntp.org iburst", "2.us.pool.ntp.org iburst"]
```

Για οποιοδήποτε άλλον node το αποτέλεσμα πρέπει να είναι το παρακάτω καθώς οι τιμή της μεταβλητής ntp::servers προέρχεται από το common.yaml:

```
$ hiera ntp::servers ::fqdn=snuffie.example.com
["kermit.example.com iburst", "grover.example.com iburst"]
```

6.5.5 Τροποποίηση του site.pp

Πριν από την ρύθμιση του hiera και την δημιουργία των πηγών δεδομένων, το site.pp αρχείο περιείχε δηλώσεις όπως την παρακάτω για κάθε node:

```
    node "kermit.example.com" {
      class { "ntp":
        servers    => [ '0.us.pool.ntp.org iburst','1.us.pool.ntp.org
iburst','2.us.pool.ntp.org iburst','3.us.pool.ntp.org iburst'],
        autoupdate => false,
        restrict   => [],
        enable     => true,
      }
    }
```

Τροποποιείται το αρχείο site.pp και τα περιεχόμενα του αντικαθίστανται με μια δήλωση πολλαπλών nodes, στην οποία περιέχεται ένα include του ntp module:

```
node "kermit.example.com", "grover.example.com", "snuffie.example.com" {
      include ntp
    }
```

Το hiera αναλαμβάνει τον εμπλουτισμό των τιμών των παραμέτρων που δέχεται η κλάση ntp αυτομάτως. Λαμβάνει υπόψη του την ιεραρχία που έχει δηλωθεί και πραγματοποιεί lookup των μεταβλητών στις αντίστοιχες πηγές δεδομένων που έχουν οριστεί. Σε αυτό το σημείο η κλάση ntp δέχεται τις παραμέτρους που χρειάζεται από το hiera. Επετεύχθη ο διαχωρισμός των δεδομένων που ορίζονται για τον κάθε node, από τον Puppet κώδικα και αυτό ήταν το ζητούμενο.

Με την παραπάνω μέθοδο το hiera παρέχει τα δεδομένα που χρειάζεται η parameterized κλάση ntp. Η κλάση ntp γίνεται include από όποιον node επιθυμεί να χρησιμοποιήσει την υπηρεσία ntp και αυτό δηλώνεται στο site.pp αρχείο με την προκαθορισμένη μορφή που προσφέρει το Puppet.

6.5.6 Χρήση του hiera_include

Με τον ίδιο τρόπο που το hiera εκχωρεί τις τιμές των παραμέτρων μιας κλάσης, υπάρχει η δυνατότητα εκχώρησης nodes σε μια κλάση, μέσω του hiera. Το hiera παρέχει στο Puppet τη συνάρτηση hiera_include για αυτήν την λειτουργία.

Η συνάρτηση hiera_include() δέχεται ως όρισμα μια λέξη-κλειδί. Προτείνεται η χρήση της λέξης “classes” γιατί είναι συναφή με το αντικείμενο που γίνεται include, αλλά μπορεί να οριστεί οτιδήποτε. Η τιμή της συγκεκριμένης παραμέτρου μπορεί να είναι το όνομα μιας κλάσης ή ενός πίνακα κλάσεων και ορίζεται στις πηγές δεδομένων.

Για την χρήση του hiera_include με το προηγούμενο παράδειγμα, πραγματοποιούνται οι αλλαγές που απαιτούνται στις πηγές δεδομένων όπως και στο site.pp manifest αρχείο. Το αρχείο site.pp τροποποιείται ώστε να περιέχει μόνο την παρακάτω δήλωση:

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

hiera_include('classes')
```

Για την εκχώρηση της κλάσης ntp σε όσους nodes ταιριάζουν με τις πηγές δεδομένων που έχουμε ορίσει, προστίθεται η παρακάτω γραμμή στις πηγές δεδομένων που επιθυμούμε:

```
"classes" : "ntp",
```

Για παράδειγμα, η πηγή δεδομένων με πρώτη προτεραιότητα που αντιστοιχεί στον node kermit τροποποιείται ώστε να περιέχει τα παρακάτω:

```
#/etc/puppet/hieradata/node/kermit.example.com.yaml
---
classes: ntp
ntp::restrict:
  -
ntp::autoupdate: false
ntp::enable: true
ntp::servers:
  - 0.us.pool.ntp.org iburst
  - 1.us.pool.ntp.org iburst
  - 2.us.pool.ntp.org iburst
  - 3.us.pool.ntp.org iburst
```

Με αυτόν τον τρόπο δηλώθηκε η κλάση ntp στους nodes που ήταν επιθυμητό χωρίς την δήλωση τους μέσα στον κώδικα Puppet. Το hiera ανέλαβε την εφαρμογή των κλάσεων, όπου αυτές ορίστηκαν, με βάση την ιεραρχία που υπάρχει.

6.5.6.1 Διαχείριση της κλάσης vmwaretools

Στο προηγούμενο παράδειγμα, κατά την δημιουργία της ιεραρχίας, χρησιμοποιήθηκε η μεταβλητή-γεγονός ::fqdn για την καθοδηγημένη δήλωση τιμών των παραμέτρων με βάση το fully qualified domain name των nodes. Έτσι ο node kermit.example.com, λαμβάνει διαφορετικές τιμές παραμέτρων από τον grove.example.com και διαφορετικές από κάθε άλλο node με άλλο fqdn. Η παραπάνω δυνατότητα υπάρχει λόγω του hiera και της ιεραρχίας που επιτρέπει τον ορισμό μιας δυναμικής πηγή δεδομένων βάσει του γεγονότος ::fqdn.

Όμως το γεγονός ::fqdn δεν είναι το μόνο που επιτρέπει την καθοδηγημένη αντιστοίχιση των κλάσεων στους nodes. Για αυτόν τον λόγο μπορεί να χρησιμοποιηθεί οποιοδήποτε γεγονός και να είναι κάποιο χαρακτηριστικό όχι τόσο εύκολα διακριτό όπως το fqdn. Ακολουθεί ένα παράδειγμα καθοδηγημένης μαζικής διαχείρισης με χρήση του hiera_include. Έστω ότι στο περιβάλλον της επιχείρησης υπάρχει ένας αριθμός από εικονικές μηχανές, βασιζόμενες στον VMware ESXi Hypervisor. Οι εικονικές μηχανές έχουν ρυθμιστεί ώστε να είναι agent nodes στον master node του περιβάλλοντος που περιγράφηκε και είναι επιθυμητή η εγκατάσταση των vmware tools.

Θα χρησιμοποιηθεί το hiera για την οργάνωση και την δόμηση των παραμέτρων των κλάσεων με τέτοιο τρόπο ώστε να εξασφαλιστεί η καθολική εφαρμογή του module σε όλους τους VMWare virtual host. Επίσης, η διαδρομή εγκατάστασης του πακέτου θα καθοριστεί ανάλογα με το λειτουργικό σύστημα κάθε virtual host.

Για το συγκεκριμένο παράδειγμα θα χρησιμοποιηθεί το puppet-vmwaretools module από το Puppet Forge (https://github.com/craigwatson/puppet-vmwaretools). Η κλάση vmwaretools λαμβάνει δύο παραμέτρους, το version και το working_dir. Η πρώτη παράμετρος δηλώνει την έκδοση των vmwaretools και η δεύτερη τον φάκελο εγκατάστασης του πακέτου.

Τροποποιείται το hiera.yaml αρχείο και προστίθενται δύο νέες πηγές δεδομένων στην ιεραρχία. Μια με βάση το γεγονός ::virtual που επιστρέφει την τιμή vmware όταν ο virtual host βασίζεται σε VMWare και μια με το γεγονός ::osfamily που επιστρέφει την οικογένεια του λειτουργικού συστήματος που ανήκει ένας node. Τα περιεχόμενα του αρχείου βρίσκονται παρακάτω:

```
---
:backends:
  - yaml
:yaml:
  :datadir: /etc/puppet/hieradata
:hierarchy:
  - "node/%{::fqdn}"
  - "virtual/%{::virtual}"
  - "osfamily/%{osfamily}"
  - common
```

Στη συνέχεια δημιουργούνται οι φάκελοι που θα αποθηκευτούν οι νέες πηγές δεδομένων:

```
`mkdir /etc/puppet/hieradata/virtual; mkdir
/etc/puppet/hieradata/osfamily`
```

Έπειτα δημιουργείται η πηγή δεδομένων vmware.yaml που περιέχει απλώς τον ορισμό της κλάσης vmwaretools:

```
#/etc/puppet/hieradata/virtual/vmware.yaml
---
classes: vmwaretools
```

Επίσης, χρειάζεται να δηλωθούν τα δεδομένα που χρειάζεται η κλάση vmwaretools, δηλαδή οι παράμετροι version και working_dir. Η παράμετρος working_dir λαμβάνει την τιμή /opt/vmware όταν το λειτουργικό ενός agent βασίζεται σε RedHat, και /usr/local/vmware όταν βασίζεται σε Debian. Αυτό το πρόβλημα που λυνόταν με προγραμματιστική λογική σε προηγούμενα κεφάλαια (κλάση ::params), στην συγκεκριμένη περίπτωση λύνεται με την ιεραρχία των πηγών δεδομένων. Οπότε δημιουργούνται τα παρακάτω αρχεία:

```
#/etc/puppet/hieradata/osfamily/RedHat.yaml
---
vmwaretools::working_dir: /opt/vmware
```

```
#/etc/puppet/hieradata/osfamily/Debian.yaml
---
vmwaretools::working_dir: /usr/local/vmware
```

Η τελευταία παράμετρος version, θα δηλωθεί στο αρχείο common.yaml γιατί δεν επιθυμούμε διαφοροποιήσεις στις εκδόσεις των vmwaretools ανάμεσα στους virtual hosts. Το αρχείο common.yaml τροποποιείται και περιέχει τα παρακάτω:

```
---
vmwaretools::version: 8.6.5-621624
ntp::autoupdate: true
ntp::enable: true
ntp::servers:
  - grover.example.com iburst
  - kermit.example.com iburst
```

Και τέλος προχωράμε σε έλεγχο των παραπάνω δηλώσεων μέσω του hiera command-line εργαλείου.

```
$ hiera vmwaretools::working_dir osfamily=RedHat
/opt/vmware

$ hiera vmwaretools::working_dir osfamily=Debian
/usr/local/vmware

$ hiera vmwaretools::version
8.6.5-621624

$ hiera classes ::virtual=vmware
vmwaretools
```

Για την ολοκλήρωση της εφαρμογής της κλάσης vmwaretools απαιτείται επανεκκίνηση της υπηρεσίας puppetmaster.

6.6 Profiles: technology-specific wrapper classes

Ένα puppet profile ορίζεται ως μια wrapper-κλάση, δηλαδή μια κλάση που συγκεντρώνει κλήσεις προς το hiera, μαζί με δηλώσεις κλάσεων για την δόμηση μιας λειτουργικής ενότητας. Ένα profile χρησιμοποιείται για την διαχείριση μιας εφαρμογής στο σύνολο της. Θα εξεταστεί η δημιουργία ενός profile για την διαχείριση της εγκατάστασης του wordpress σε έναν agent node. Η διαδικασία περιλαμβάνει, εγκατάσταση του apache και της mysql. δημιουργία vhost και χρήστη συστήματος στο λειτουργικό σύστημα του agent, δημιουργία βάσης mysql όπως και δημιουργία χρήστη στη βάση mysql. Για τον σκοπό αυτό θα χρησιμοποιηθούν τα παρακάτω modules από το Puppet Forge:

- https://forge.puppetlabs.com/hunner/wordpress
- https://forge.puppetlabs.com/puppetlabs/apache
- https://forge.puppetlabs.com/puppetlabs/mysql

Η εγκατάσταση τους μπορεί να πραγματοποιηθεί με την εκτέλεση των παρακάτω εντολών στον Puppet Master (master2.example.com):

```
#puppet module install hunner-wordpress
#puppet module install puppetlabs-apache
#puppet module install puppetlabs-mysql
```

Έπειτα δημιουργούνται και εμπλουτίζονται τα παρακάτω αρχεία:

Δημιουργία ιεραρχίας (node, location, role, common).

```
#/etc/puppet/hiera.yaml
---
:backends:
  - yaml

:hierarchy:
  - "node/%{::fqdn}"
  - "location/%{::location}"
  - "role/%{::role}"
  - common

:yaml:
  :datadir: /etc/puppet/hieradata

:merge_behavior: deeper
```

Δημιουργία του module που θα περιέχει το νέο profile wordpress.

```
#mkdir -p /etc/puppet/modules/profiles/manifests/
#touch /etc/puppet/modules/profiles/manifests/wordpress.pp

#/etc/puppet/modules/profiles/manifests/wordpress.pp

class profiles::wordpress {

  ## Hiera lookups
  $site_name               = hiera('profiles::wordpress::site_name')
  $wordpress_user_password =
hiera('profiles::wordpress::wordpress_user_password')
  $wordpress_root_password =
hiera('profiles::wordpress::mysql_root_password')
  $wordpress_db_host       =
hiera('profiles::wordpress::wordpress_db_host')
  $wordpress_db_name       =
hiera('profiles::wordpress::wordpress_db_name')
  $wordpress_db_password   =
hiera('profiles::wordpress::wordpress_db_password')
  $wordpress_user          = hiera('profiles::wordpress::wordpress_user')
  $wordpress_group         =
hiera('profiles::wordpress::wordpress_group')
  $wordpress_docroot       =
hiera('profiles::wordpress::wordpress_docroot')
  $wordpress_port          = hiera('profiles::wordpress::wordpress_port')

  ## Create user
  group { 'wordpress':
    ensure => present,
    name   => $wordpress_group,
  }
  user { 'wordpress':
    ensure   => present,
    gid      => $wordpress_group,
    password => $wordpress_user_password,
    name     => $wordpress_user,
```

```
    home     => $wordpress_docroot,
  }

  ## Configure mysql
  class { 'mysql::server':
    root_password => $wordpress_root_password,
  }

  class { 'mysql::bindings':
    php_enable => true,
  }

  ## Configure apache
  include apache
  include apache::mod::php
  apache::vhost { $site_name:
    port    => $wordpress_port,
    docroot => $wordpress_docroot,
  }

  ## Configure wordpress
  class { '::wordpress':
    install_dir => $wordpress_docroot,
    db_name     => $wordpress_db_name,
    db_host     => $wordpress_db_host,
    db_password => $wordpress_db_password,
  }
}
```

Οι προκαθορισμένες τιμές των παραμέτρων της κλάσης wordpress δηλώνονται στο common.yaml

```
#/etc/puppet/hieradata/common.yaml

profiles::wordpress::site_name: 'wordpress.example.com'
profiles::wordpress::wordpress_user_password: 'wordpress'
profiles::wordpress::mysql_root_password: 'password'
profiles::wordpress::wordpress_db_host: 'localhost'
profiles::wordpress::wordpress_db_name: 'wordpress'
profiles::wordpress::wordpress_db_password: 'wordpress'
profiles::wordpress::wordpress_user: 'wordpress'
profiles::wordpress::wordpress_group: 'wordpress'
profiles::wordpress::wordpress_docroot: '/var/www/wordpress'
profiles::wordpress::wordpress_port: '80'
```

Αν επιθυμούμε την τροποποίηση των παραμέτρων για την εγκατάσταση του wordpress για παράδειγμα στον Puppet master master2.example.com δημιουργείται και εμπλουτίζεται το παρακάτω αρχείο:

```
#/etc/puppet/hieradata/node/master2.example.com.yaml

profiles::wordpress::site_name: 'wordpress2.example.com'
profiles::wordpress::wordpress_user_password: 'wordpress2'
profiles::wordpress::mysql_root_password: 'password'
profiles::wordpress::wordpress_db_host: 'localhost'
profiles::wordpress::wordpress_db_name: 'wordpress2'
profiles::wordpress::wordpress_db_password: 'wordpress2'
profiles::wordpress::wordpress_user: 'wordpress2'
profiles::wordpress::wordpress_group: 'wordpress2'
profiles::wordpress::wordpress_docroot: '/var/www/wordpress2'
profiles::wordpress::wordpress_port: '80'
```

Εδώ μπορούμε να θέσουμε συγκεκριμένες παραμέτρους για τον node master2.example.com, που λόγω της ιεραρχίας που ορίστηκε στο hiera.yaml έχουν μεγαλύτερη προτεραιότητα από την πηγή δεδομένων common.yaml και άρα υπερισχύουν.

Τέλος αρκεί η παρακάτω δήλωση στο main site.pp αρχείο, ώστε η εφαρμογή wordpress να γίνει διαθέσιμη για τον node master2.example.com, στη διεύθυνση url που έχει οριστεί.

```
#/etc/puppet/manifests/site.pp
node 'master2.example.com' {

include profiles::wordpress
}
```

ΚΕΦΑΛΑΙΟ 7

Puppet Enviroment και Version-Control

Μια από τις λειτουργίες που προσφέρει το Puppet είναι η απομονωμένη ομαδοποίηση των Agent nodes με την χρήση των Puppet Enviroment. Ο Puppet master μπορεί να ρυθμιστεί έτσι ώστε να εξυπηρετεί κάθε Enviroment με διαφορετικά modules και ξεχωριστό main manifest αρχείο. Έτσι, επιτρέπεται η δημιουργία διαφορετικών εκδόσεων των ίδιων modules και η εφαρμογή τους σε ξεχωριστούς nodes. Μπορεί για παράδειγμα να χρησιμοποιηθεί το μοντέλο ανάπτυξης “development, testing και production” όπου τα περιβάλλοντα ανάπτυξης είναι απομονωμένα και οι αλλαγές που πραγματοποιούνται σε ένα, δεν επηρεάζει τα υπόλοιπα. Το συγκεκριμένο μοντέλο επιτρέπει την ασφαλή μετάβαση από ένα μη κρίσιμο περιβάλλον, όπως είναι το development ή το testing, στο κυρίως περιβάλλον μιας επιχείρησης (production). Η αρχική έκδοση ενός νέου module δημιουργείται στο περιβάλλον development και η ανάπτυξη του συνεχίζεται μέχρις ότου να κριθεί ασφαλή η μετάβαση του στο περιβάλλον testing. Εκεί πραγματοποιούνται ενδελεχείς έλεγχοι και διορθώσεις για την επίτευξη της επιθυμητής λειτουργικότητας του module. Τέλος, πραγματοποιείται η εφαρμογή του module στο περιβάλλον production. Τα παραπάνω βήματα είναι σημαντικά και απαραίτητα κατά την ανάπτυξη ενός module και είναι καλή πρακτική να εφαρμόζονται.

Στο συγκεκριμένο παράδειγμα θα χρησιμοποιηθεί το Git ως σύστημα ελέγχου εκδόσεων των Puppet modules. Το Git είναι ένα σύστημα ελέγχου διανεμόμενης έκδοσης και διαχείρισης πηγαίου κώδικα (SCM) με έμφαση στην ταχύτητα, στην ακεραιότητα των δεδομένων και στην υποστήριξη για κατανεμημένες μη γραμμικές ροές εργασίας. Το Git σχεδιάστηκε και αναπτύχθηκε αρχικά από τον linus torvalds για τη ανάπτυξη του πυρήνα Linux το 2005 και έχει γίνει από τότε το πιο πλατιά διαδεδομένο σύστημα ελέγχου εκδόσεων για ανάπτυξη λογισμικού. Όπως τα περισσότερα άλλα διανεμόμενα συστήματα ελέγχου εκδόσεων αναθεώρησης και αντίθετα με τα περισσότερα συστήματα πελάτη-διακομιστή, κάθε κατάλογος εργασίας του Git είναι ένα ολοκληρωμένο αποθετήριο με πλήρες ιστορικό και

δυνατότητες πλήρους παρακολούθησης της έκδοσης, ανεξάρτητα από την πρόσβαση δικτύου ή ενός κεντρικού διακομιστή. Το Git είναι Ελεύθερο λογισμικό που διανέμεται κάτω από τους όρους της έκδοσης 2 της Γενικής Άδειας Δημόσιας Χρήσης GNU.

7.1 Δημιουργία των στατικών Enviroment: development, testing, production

Για την εφαρμογή του μοντέλου development, testing, production στο Puppet μπορούν να δημιουργηθούν στατικά τρία περιβάλλοντα, που ορίζονται στο αρχείο puppet.conf. Τα περιεχόμενα του αρχείου φαίνονται παρακάτω:

```
#/etc/puppet/puppet.conf
[main]
  server = puppet.example.com
  environment = production
  confdir = /etc/puppet
[agent]
  report = true
  show_diff = true
[production]
  manifest = /etc/puppet/environments/production/manifests/site.pp
  modulepath = /etc/puppet/environments/production/modules
[testing]
  manifest = /etc/puppet/environments/testing/manifests/site.pp
  modulepath = /etc/puppet/environments/testing/modules
[development]
  manifest = /etc/puppet/environments/development/manifests/site.pp
  modulepath = /etc/puppet/environments/development/modules
```

Ακολουθεί η ρύθμιση του Git, με δημιουργία branches για κάθε ένα από τα παραπάνω enviroments στους φακέλους των module. Έχει προηγηθεί η δημιουργία του παρακάτω δημόσιου αποθετηρίου με χρήση ssh key στον host master2.example.com: https://github.com/go2jimmys/puppet.

```
[root@master2 ~]# cd /etc/puppet/environments/development/
[root@master2 development]# git init
Initialized empty Git repository in /etc/puppet/environments/development/
[root@master2 development]# git checkout -b devel
Switched to a new branch 'devel'
[root@master2 development]# git add *
```

```
[root@master2 development]# git commit -m "Initialize Development
Enviroment"
[root@master2 development]# git remote add origin
git@github.com:go2jimmys/puppet.git
[root@master2 development]# git push origin devel
Warning: Permanently added the RSA host key for IP address
'192.30.252.128' to the list of known hosts.
Enter passphrase for key '/root/.ssh/id rsa':
Writing objects: 100% (3/3), 332 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:go2jimmys/puppet.git
 * [new branch]      devel -> devel

[root@master2 environments]# git clone
/etc/puppet/environments/development/ /etc/puppet/environments/testing
[root@master2 environments]# cd /etc/puppet/environments/testing
[root@master2 testing]# git remote remove origin
[root@master2 testing]# git remote add origin
git@github.com:go2jimmys/puppet.git
[root@master2 testing]# git checkout -b devel
[root@master2 testing]# git push origin testing


[root@master2 environments]# git clone
/etc/puppet/environments/development/ /etc/puppet/environments/production
[root@master2 environments]# cd /etc/puppet/environments/production
[root@master2 production]# git remote remove origin
[root@master2 production]# git remote add origin
git@github.com:go2jimmys/puppet.git
[root@master2 production]# git checkout -b devel
[root@master2 production]# git push origin production
```

7.2 Δυναμικό περιβάλλον

Τα στατικά περιβάλλοντα όπως στο προηγούμενο παράδειγμα θέτουν περιορισμό στις ονομασίες και στο πλήθος των περιβαλλόντων διαθέσιμα για ανάπτυξη. Αυτό το πρόβλημα μπορεί να λυθεί με την ρύθμιση του δυναμικού περιβάλλοντος στο puppet.conf. Η διαδικασία περιλαμβάνει τον ορισμό μιας μεταβλητής $enviroment, η οποία μπορεί να λάβει διαφορετική τιμή σε κάθε agent node. Προκαθορισμένη τιμή ορίζεται το περιβάλλον production στην περιοχή main. Έτσι το puppet.conf αρχείο διαμορφώνεται ως εξής:

```
#/etc/puppet/puppet.conf
[main]
  server = puppet.example.com
  environment = production
  confdir = /etc/puppet
[master]
  environment = production
  manifest    = $confdir/environments/$environment/manifests/site.pp
```

```
  modulepath  = $confdir/environments/$environment/modules
[agent]
  report = true
  show_diff = true
  environment = production
```

Με αυτόν τον τρόπο αρκεί να δημιουργηθεί ένα νέο περιβάλλον υπό τον φάκελο /etc/puppet/enviroments, και το Puppet master θα προχωρήσει στην αυτόματη φόρτωση του χωρίς να χρειαστούν άλλες αλλαγές στο puppet.conf αρχείο.

ΚΕΦΑΛΑΙΟ 8

Γραφικές κονσόλες διαχείρισης

Το Puppet παρέχει τρία εργαλεία με λειτουργικότητα κονσόλας σε γραφικό περιβάλλον και είναι τα: foreman, Puppet Enterprise Console, και Puppetboard. Το foreman είναι ένα ολοκληρωμένο εργαλείο διαχείρισης του κύκλου ζωής ενός node. Προσφέρει provisioning, configuration management και προβολή των αναφορών. Το Puppet Enterprise Console είναι εμπορική εφαρμογή της Puppet Labs, ενώ η λειτουργικότητα του περιορίζεται καθώς υπάρχει ελεύθερη χρήση του για μέχρι και δέκα agent nodes. Το εργαλείο Puppetboard προσφέρει διαχείριση των αναφορών που λαμβάνει ο Puppet Master από τους agents. Στο συγκεκριμένο κεφάλαιο θα παρουσιαστεί η διαδικασία εγκατάστασης του foreman όπως και του Puppet Enterprise σε διαφορετικούς Puppet Master nodes. Επίσης θα εξεταστεί η λειτουργικότητα της κάθε κονσόλας.

8.1 Foreman

Εγκατάσταση του πακέτου σε RedHat/Fedora/CentOS

```
$ yum -y install
http://yum.theforeman.org/releases/1.2/el6/x86_64/foreman-release.rpm
$ yum -y install foreman-installer
```

Εγκατάσταση του πακέτου σε Debian/Ubuntu

```
$ echo "deb http://deb.theforeman.org/ precise stable" >
/etc/apt/sources.list.d/foreman.list
$ wget -q http://deb.theforeman.org/foreman.asc -O- | apt-key add -
$ apt-get update && apt-get install foreman-installer
```

8.1.1 Ρύθμιση της εγκατάστασης

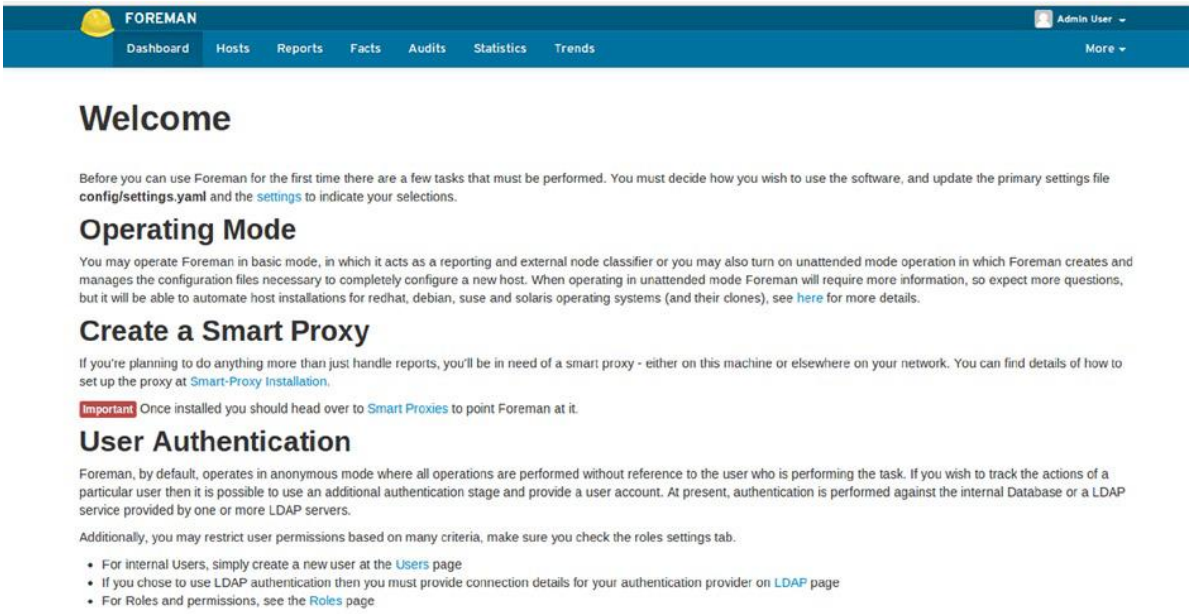
Ο installer του foreman είναι μια συλλογή από Puppet modules τα οποία μπορούν να διαχειριστούν την εγκατάσταση του foreman. Κατά την κλήση του οδηγού απενεργοποιείται η λειτουργία της υπηρεσίας tftp.

```
$ ruby /usr/share/foreman-installer/generate_answers.rb
Welcome to the Foreman Installer!
---------------------------------
This installer will help you set up Foreman and the associated extra
configuration necessary to get you up and running. There is an
interactive shell
which will ask you questions, but if you just want to get up and running
as fast
as possible, answer 'yes' to the all-in-one install at the beginning
Ready to start? (y/n)
y
Do you want to use the default all-in-one setup?
This will configure Foreman, Foreman-Proxy, Puppet (including a
puppetmaster),
several puppet environments, TFTP (for provisioning) and sudo (for puppet
certificate management) (y/n)
n
Main Config Menu
1. Configure Foreman settings
2. Configure Foreman_proxy settings
3. Configure Puppet settings
4. Configure Puppetmaster settings
5. Display current config
6. Save and Exit
7. Exit without Saving
Choose an option from the menu... 2
Current config is:
Foreman_proxy is enabled with defaults
Foreman_proxy Config Menu
1. Enable foreman_proxy with all defaults
2. Disable foreman_proxy completely
3. Should Foreman_proxy manage Puppet (needed for puppet classes)?
(default:
true)
4. Should Foreman_proxy manage DNS? (default: false)
5. Should Foreman_proxy be installed from the stable,rc, or nightly repo?
(default: stable)
6. Should Foreman_proxy manage DHCP? (default: false)
7. Should Foreman_proxy manage TFTP? (default: true)
8. Should Foreman_proxy manage PuppetCA (needed for certificates)?
(default:
true)
9. Add other key/value pair to the config
10. Go up to main menu
Choose an option from the menu... 7
y/n?
n
Current config is:
Foreman_proxy is enabled with overrides:
---
tftp: false
Do you want to run Puppet now with these settings? (y/n)
y
```

Η παραπάνω διαδικασία παράγει ένα αρχείο ρυθμίσεων δομημένο σε yaml:

```
# cat /usr/share/foreman-installer/foreman_installer/answers.yaml
---
foreman_proxy:
tftp: false
puppet: true
foreman: true
puppetmaster: true
```

Η κονσόλα foreman είναι διαθέσιμη στην διεύθυνση https://fqdn/, όπου fqdn το πλήρες hostname του node όπου πραγματοποιήθηκε η εγκατάσταση. Τα προκαθορισμένα στοιχεία πρόσβασης username/password είναι admin/admin.



Εικόνα 8.1 - Foreman Dashboard

Η εγκατάσταση ολοκληρώνεται με την προσθήκη ενός proxy από το μενού more → configuration → Smart Proxies για την πόρτα 8443.

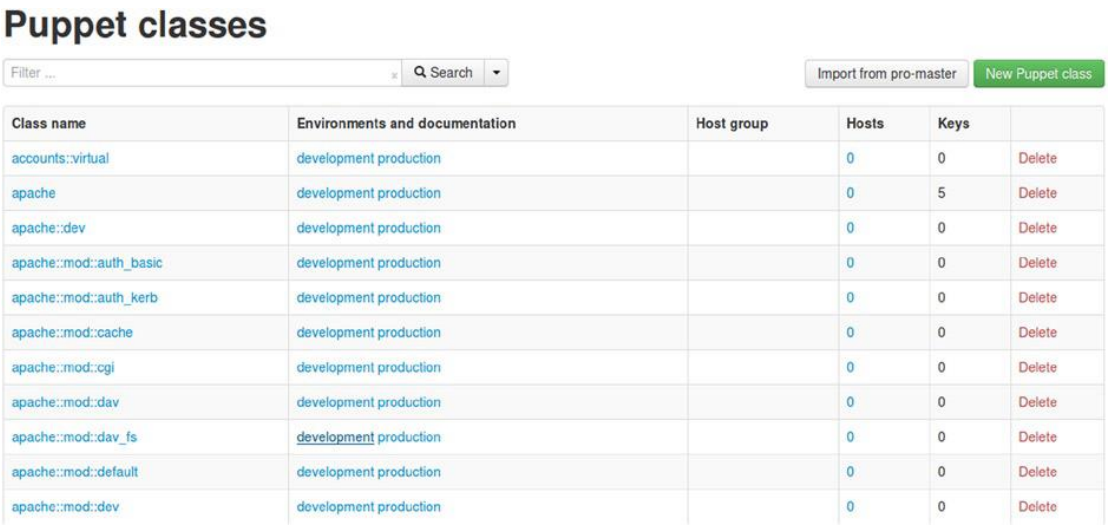
Εικόνα 8.2 - Smart Proxies list

8.1.2 Εισαγωγή δεδομένων από το Puppet

Για την εισαγωγή υπάρχων Puppet module στο Foreman χρησιμοποιείται το μενού more → Configuration → Environments και επιλέγεται το Import From Puppetmaster, όπου Puppetmaster το hostname του Puppet master server.

Εικόνα 8.3 - Puppet Classes List

Έπειτα επιλέγονται και προστίθενται τα επιθυμητά Puppet environments. Μετά την εισαγωγή οι διαθέσιμες κλάσεις εμφανίζονται σε λίστα.



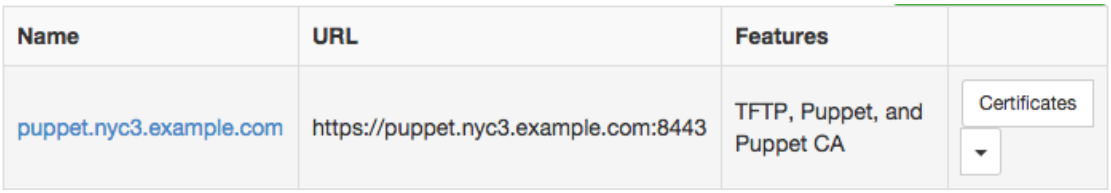
Εικόνα 8.4 - Puppet Classes List

8.1.3 Προσθήκη νέων Client

Μετά την εισαγωγή των environments από τον Puppet master, ακολουθεί η προσθήκη νέων Agent nodes με την αποστολή SSL Certificate Request και επιβεβαίωσης τους από τον Master.

```
node1# puppet agent --test --server=puppet.example.com
Info: Creating a new SSL key for node1.example.com
Info: Caching certificate for ca
Info: Creating a new SSL certificate request for node1.example.com
Info: Certificate Request fingerprint (SHA256):
6F:0D:41:14:BD:2D:FC:CE:1C:DC:11:1E:26:07:4C:08:D0:C
7:E8:62:A5:33:E3:4B:8B:C6:28:C5:C8:88:1C:C8
Exiting; no certificate found and waitforcert is disabled
```

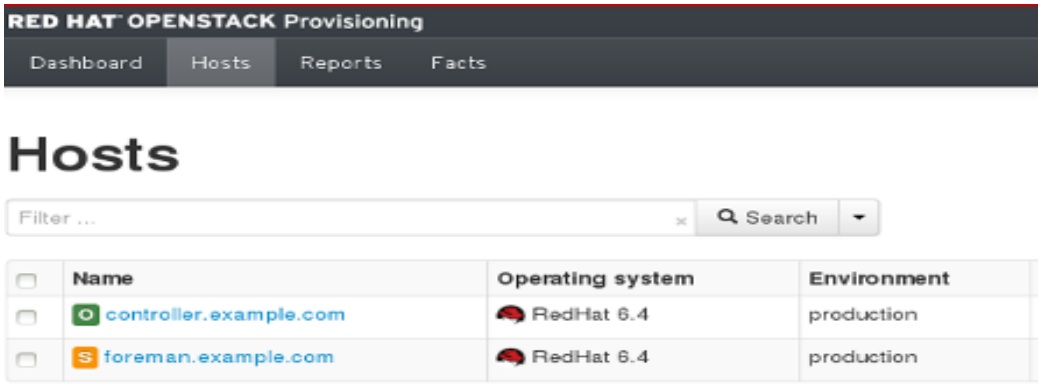
Η διαδικασία επιβεβαίωσης μπορεί να πραγματοποιηθεί εντός του Foreman από το μενού more → Configuration → Smart Proxies → Certificates, αλλά και με τον συνήθη τρόπο μέσω τερματικού στον Puppet master.



Εικόνα 8.5 - List of Certificates

```
node1# puppet agent -t
info: Caching certificate for node1.example.com
info: Caching certificate_revocation_list for ca
info: Caching catalog for node1.example.com
info: Applying configuration version '1379704950'
info: Creating state file /var/lib/puppet/state/state.yaml
notice: Finished catalog run in 0.02 seconds
```

Η σελίδα Hosts, περιλαμβάνει μια λίστα με τους διαθέσιμους Puppet agent. Επιλέγεται ένας node για λεπτομερή περιγραφή του συγκεκριμένου host.



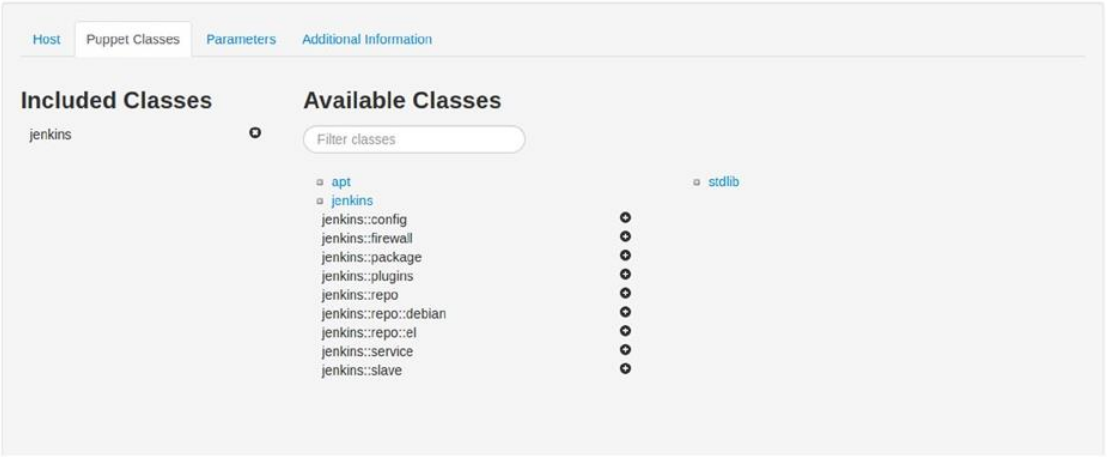
Εικόνα 8.6 - Host List

Από αυτήν την σελίδα ο χρήσης μπορεί να έχει πρόσβαση σε αναφορές, σε γεγονότα, και μετρικές για τις αλλαγές που πραγματοποιήθηκαν από Puppet. Η επιλογή edit επιτρέπει μεταξύ άλλων, την ανάθεση κλάσεων στον node.

Εικόνα 8.7 - Detailed View

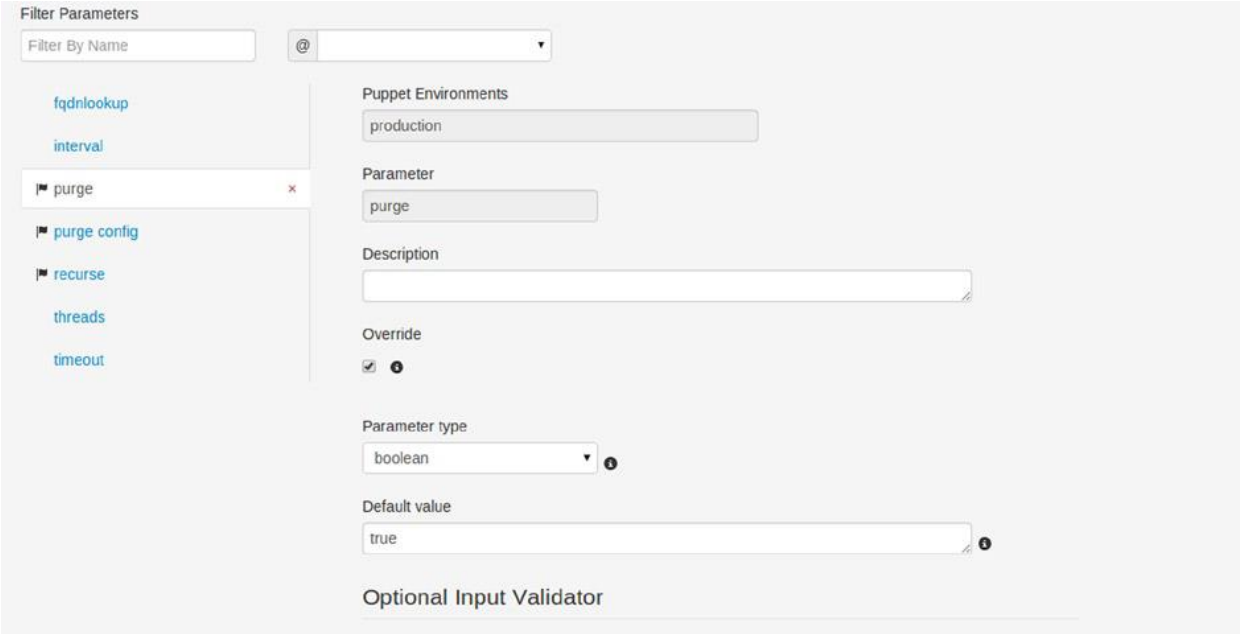
8.1.4 Χρήση του Foreman ως ENC (External Node Classifier)

Το foreman μέσω του web interface του μπορεί να χρησιμοποιηθεί ως ENC, για την εφαρμογή κλάσεων στους nodes με προσαρμοσμένες παραμέτρους. Ενώ μπορεί να χρησιμοποιηθεί παράλληλα με το main site.pp manifest για την ρύθμιση ενός περιβάλλοντος. Στο παρακάτω παράδειγμα προσθέτουμε την κλάση jerkins σε έναν agent node. Αυτό έχει ως αποτέλεσμα την εγκατάσταση του πακέτου και την διαχείριση της υπηρεσίας jerkins από το puppet.



Εικόνα 8.8 - Class Assignment

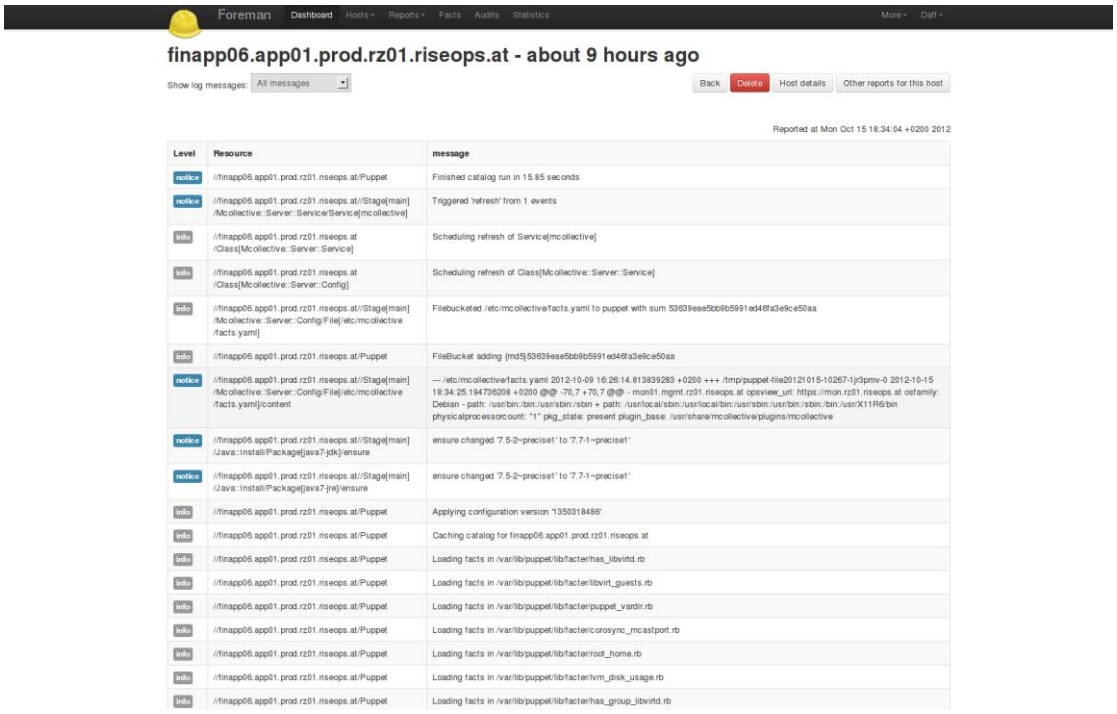
Η προσθήκη προσαρμοσμένων παραμέτρων σε μιας κλάση γίνεται δυνατή μέσω της καρτέλας Parameters με την επιλογή των παραμέτρων στην αριστερή στήλη, και τροποποίησης της τιμής τους στο κεντρικό μέρος της σελίδας. Στις παραμέτρους που έχουν αλλαχτεί, εμφανίζεται το εικονίδιο σημαίας αριστερά από κάθε παράμετρο.



Εικόνα 8.9 - Customize Parameters

8.1.5 Προβολή αναφορών μέσω του Foreman

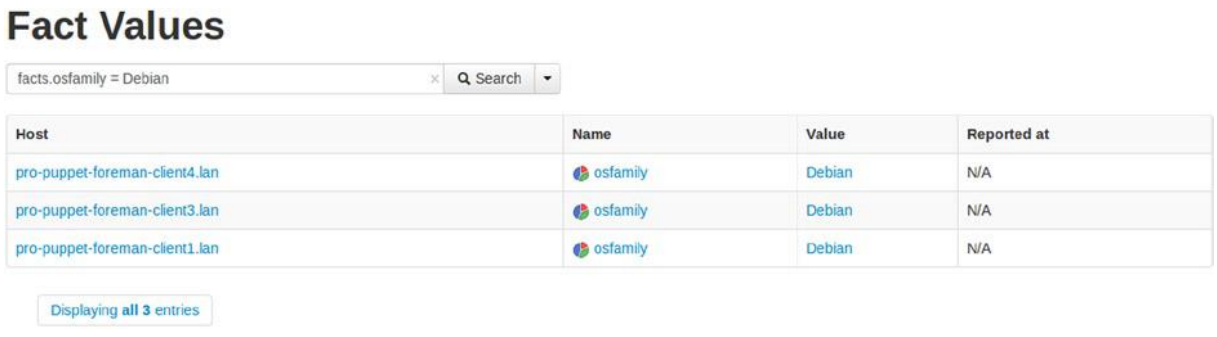
Το Foreman έχει την δυνατότητα να συλλέγει και να προβάλλει τις αναφορές που λαμβάνει ο Puppet master από τους agent nodes, οι οποίες μπορούν να εμφανιστούν με την επιλογή reports είτε από το detailed view ενός host, είτε από το κεντρικό μενού.



Εικόνα 8.10 - Reports

8.1.6 Αναζήτηση γεγονότων

Το Foreman συλλέγει τα γεγονότα από κάθε agent node, στα οποία υπάρχει η δυνατότητα εκτέλεσης queries για την εμφάνιση προσαρμοσμένων αποτελεσμάτων. Για παράδειγμα η προβολή όλων των node που το λειτουργικό τους ταιριάζει με την ευρύτερη οικογένεια “debian” μπορεί να ανακτηθεί με το query osfamily = Debian.



Εικόνα 8.11 - Search Hosts Based On Facts

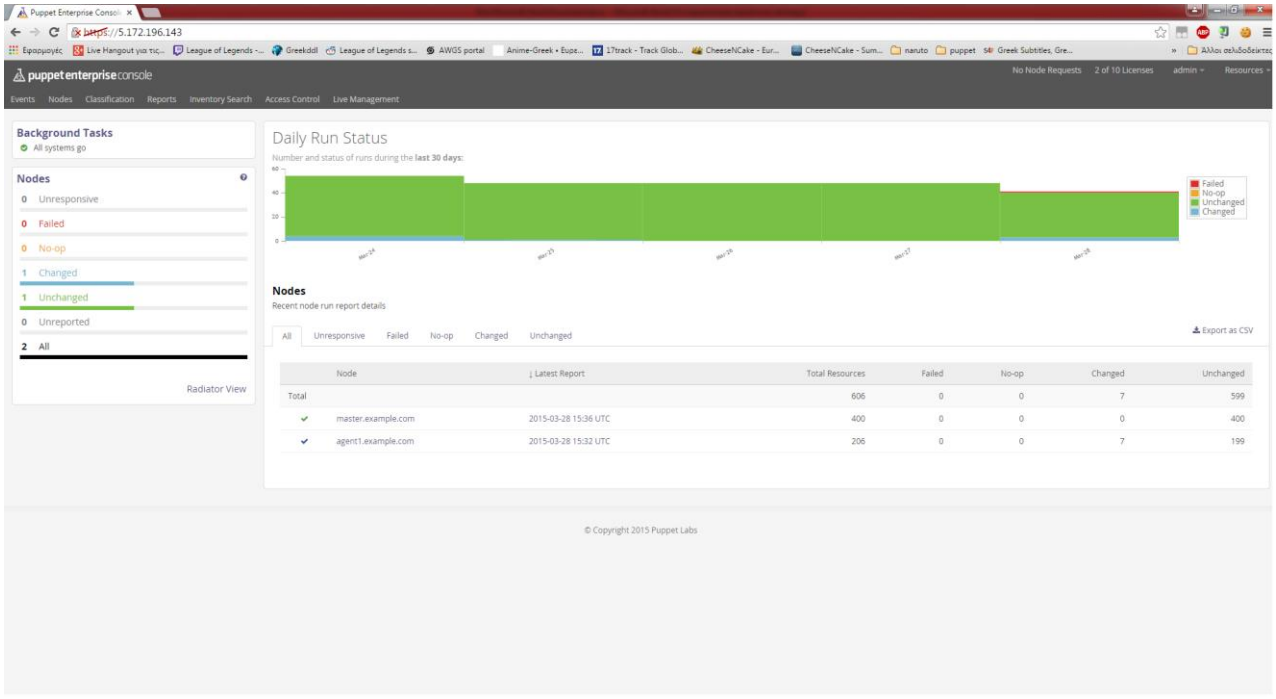
8.2 Puppet Enterprise Console

Το Puppet Enterprise αναπτύσσεται από την Puppet Labs και διατίθεται ως εμπορική εφαρμογή με γραμμή υποστήριξης, ενώ έχει ελεύθερη άδεια χρήσης για έως και δέκα agent nodes. Προσφέρει προβολή αναφορών, κατηγοριοποίηση των nodes με την ανάθεση κλάσεων, όπως επίσης την ικανότητα Live Management που βασίζεται στην υπηρεσία mcollective.

Εγκατάσταση του Puppet Enterprise σε CentOS 7:

```
#wget --trust-server-names "https://pm.puppetlabs.com/cgi-
#bin/download.cgi?ver=latest&dist=el&arch=x86_64&rel=7"
#tar -xzvf puppet-enterprise-3.7.2-el-7-x86_64.tar.gz
#cd ./puppet-enterprise-3.7.2-el-7-x86_64
```

Η εγκατάσταση ολοκληρώνεται από την σελίδα https://<fqdn>:3000, όπου ορίζονται διάφορες πληροφορίες σχετικές με τις υπηρεσίες που πρόκειται να εγκατασταθούν. Στο παράδειγμα, το Puppet Enterprise έχει εγκατασταθεί στον node με hostname=master.example.com, που διαθέτει την IP 5.172.196.143. Η εφαρμογή Puppet Enterprise Console γίνεται διαθέσιμη από τη διεύθυνση http://master.example.com ή http://5.172.196.143.

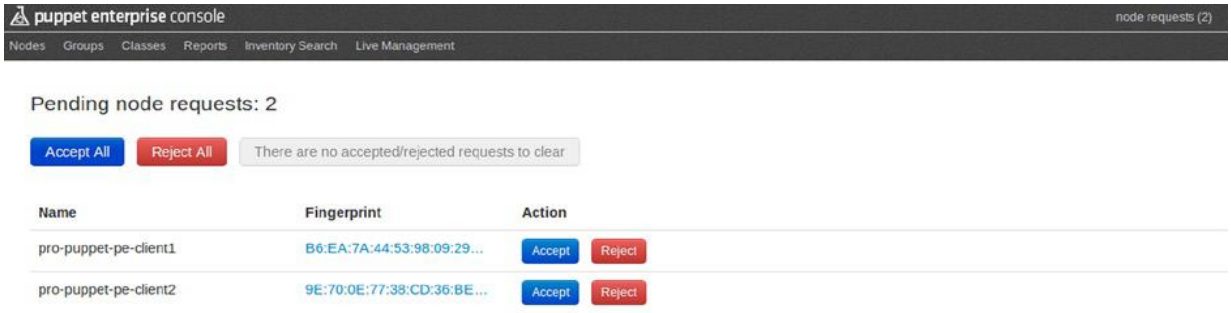


Εικόνα 8.12 - Puppet Enterprise Dashboard

8.2.1 Προσθήκη νέων Client

Η συγκεκριμένη διαδικασία είναι η ίδια κατά την σύνδεση ενός node στον Puppet master με την αποστολή SSL Certificate Request, απλώς το πιστοποιητικό μπορεί να επιβεβαιωθεί από τον Puppet master και μέσω του Puppet Enterprise Console. Επίσης υπάρχει η δυνατότητα αυτόματης εγκατάστασης και ρύθμισης ενός host και της μετατροπής του σε Puppet agent με την εκτέλεση ενός script που παρέχεται από τον Puppet master. Το script βρίσκεται στην διεύθυνση: https://master.example.com:8140/packages/current/install.bash και εκτελείται από τον host με την παρακάτω εντολή:

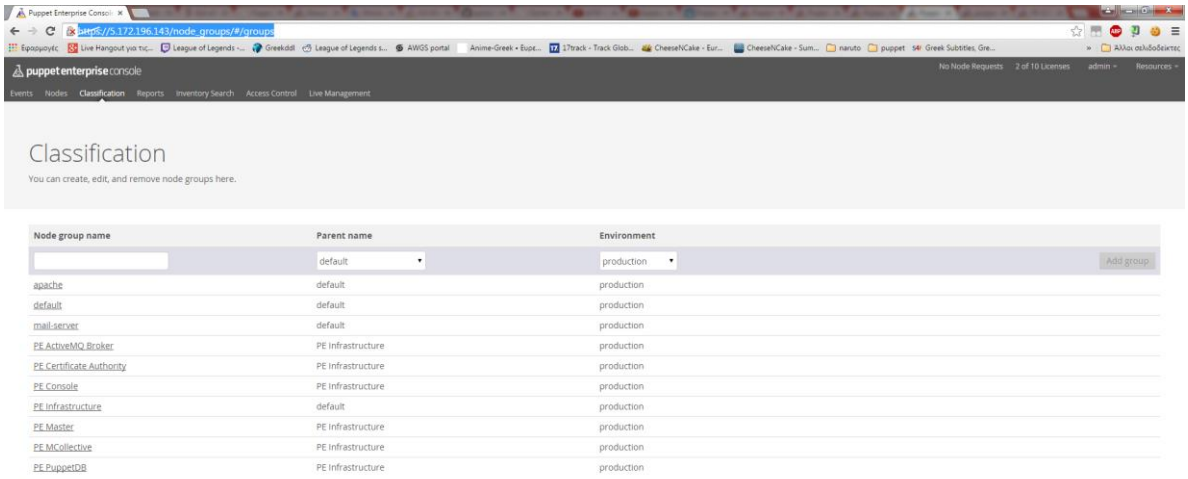
```
curl -k https://master.example.com:8140/packages/current/install.bash | sudo bash
```



Εικόνα 8.13 - View Certificates

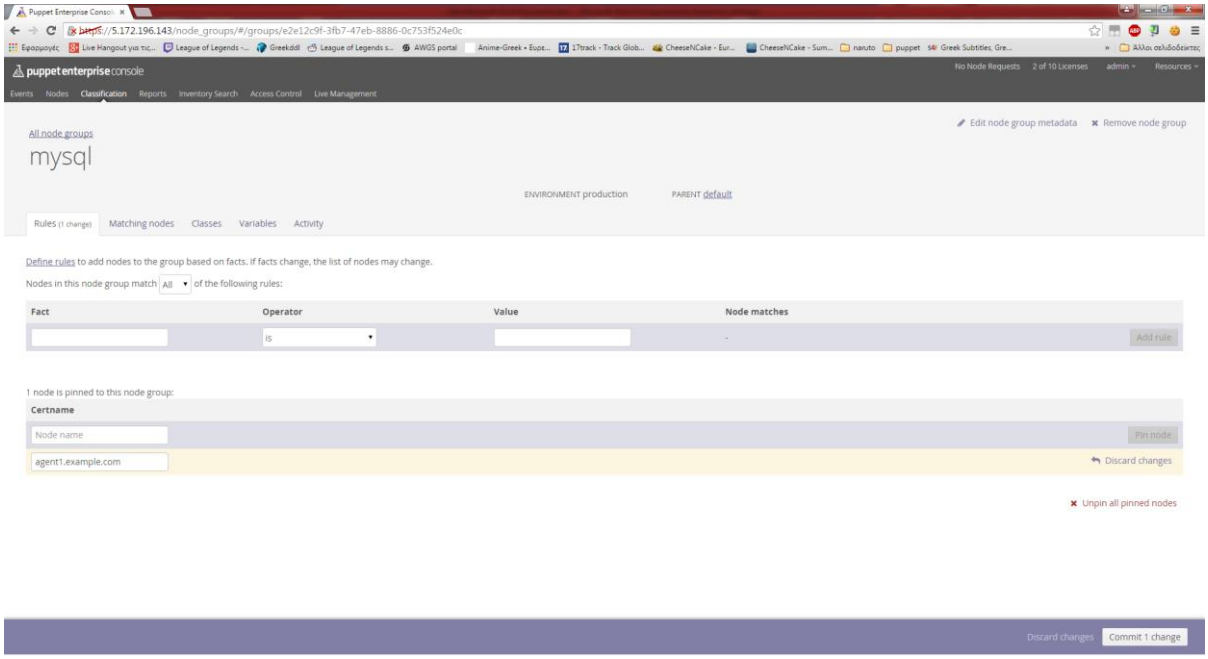
8.2.2 Χρήση του Puppet Enterprise ως ENC (External Node Classifier)

Ακολουθεί ένα παράδειγμα χρήσης του Puppet Enterprise για την προσθήκη και παραμετροποίηση της κλάσης mysql στον host agent1.example.com. Από το μενού Classification δημιουργείται ένα νέο group, θέτοντας το όνομα mysql, με parent κλάση default και environment το production.



Εικόνα 8.14 - Classification

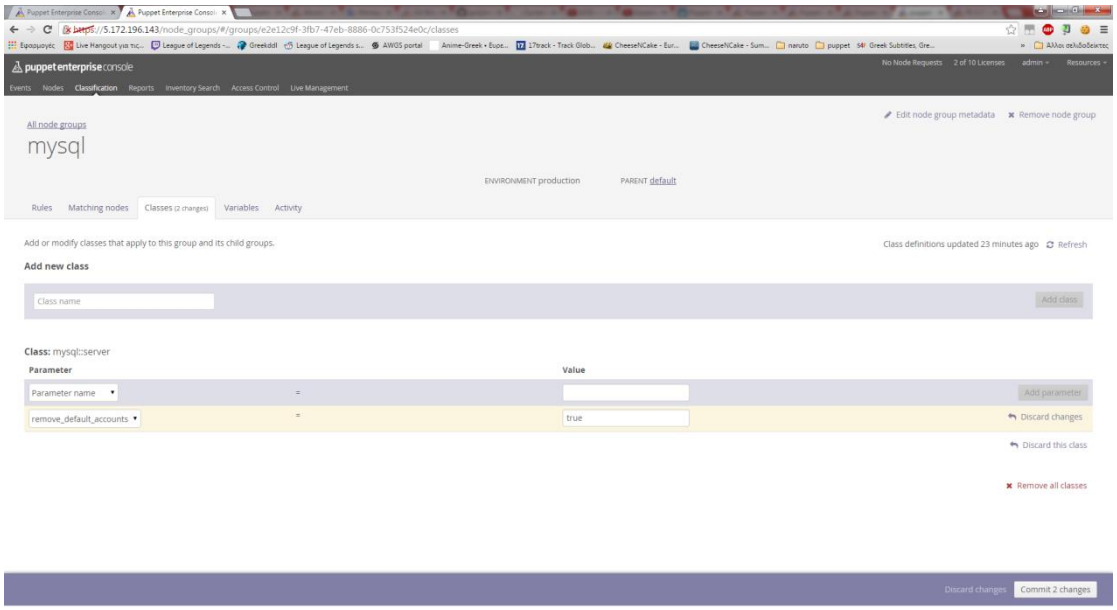
Στο αρχικό tab rules δημιουργείται ένας νέος κανόνας για την αντιστοίχιση του node agent1.example.com και επιλέγεται στα δεξιά το pin node και το commit change.



Εικόνα 8.15 - Set New Rule

Ακολουθεί η προσθήκη της κλάσης mysql::server, όπως επίσης η αλλαγή της προκαθορισμένης τιμής της παραμέτρου remove_default_accounts από false σε true. Η διαδικασία ολοκληρώνεται με την επιλογή commit κάτω δεξιά. Στην

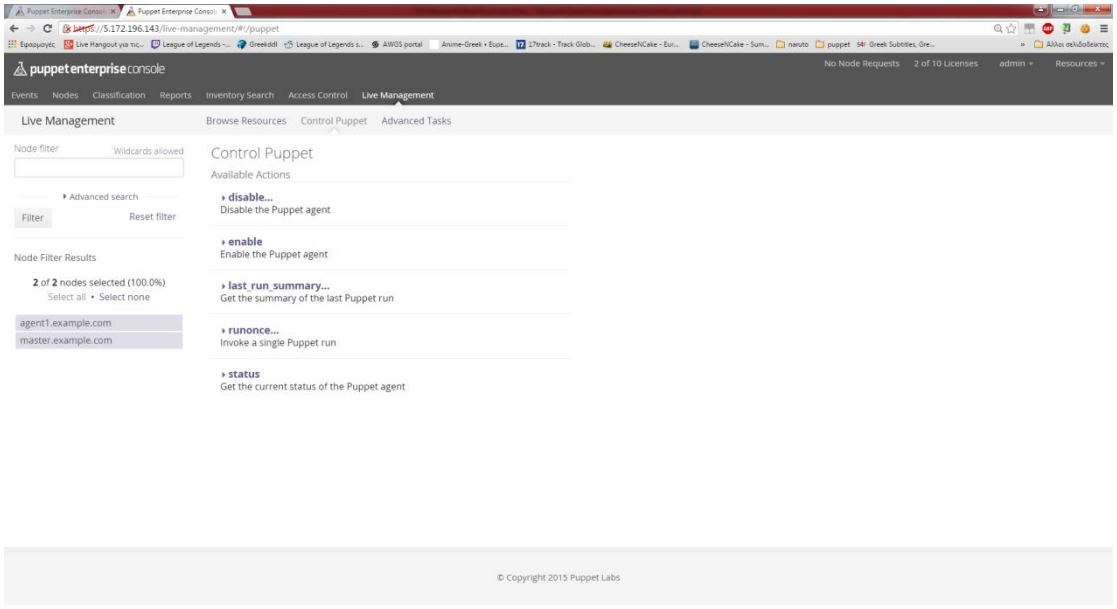
επόμενη εκτέλεση του Puppet agent, το πακέτο mysql θα βρίσκεται διαθέσιμο στο node με hostname agent1.example.com.



Εικόνα 8.16 - Add A Parametrized Class

8.2.3 Live Management

Η υπηρεσία live management προσφέρει απευθείας πρόσβαση των agent nodes μέσω του Puppet master. Δίνει την δυνατότητα περιήγησης, αποστολή ερωτημάτων, εκτέλεσης εντολών παράλληλα σε διαφορετικούς nodes.



Εικόνα 8.17 - Live Managment

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. http://docs.puppetlabs.com/puppet/3.7/reference/
2. https://docs.puppetlabs.com/puppet/latest/reference/modules_fundamentals.html
3. https://docs.puppetlabs.com/references/3.7.latest/type.html
4. https://docs.puppetlabs.com/puppet/latest/reference/lang_resources.html
5. http://docs.puppetlabs.com/hiera/latest/index.html
6. https://docs.puppetlabs.com/puppet/3.6/reference/lang_facts_and_builtin_vars.html
7. https://docs.puppetlabs.com/learning/manifests.html
8. http://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging
9. http://docs.puppetlabs.com/hiera/latest/hierarchy.html
10. https://docs.puppetlabs.com/pe/latest/quick_writing_nix.html
11. http://garylarizza.com/blog/2014/02/17/puppet-workflow-part-1/
12. Pro Puppet by Spencer Krum - εκδόσεις Apress
13. https://puppetlabs.com/blog/git-workflow-and-puppet-environments
14. http://theforeman.org/manuals/1.7/index.html#Releasenotesfor1.7