ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Διαχείριση και Παρακολούθηση Υποδομής Νέφους

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΝΙΚΟΛΑΟΥ Ι. ΒΛΑΣΤΑΡΑ

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2014

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΈΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Διαχείριση και Παρακολούθηση Υποδομής Νέφους

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΝΙΚΟΛΑΟΥ Ι. ΒΛΑΣΤΑΡΑ

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 17η Σεπτεμβρίου 2014.

(Υπογραφή)	(Υπογραφή)	(Υπογραφή)
.....	.....	.....
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2014

(Υπογραφή)

.....

ΒΛΑΣΤΑΡΑΣ ΝΙΚΟΛΑΟΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά την καθηγήτρια Θεοδώρα Βαρβαρίγου, που μου έδωσε την ευκαιρία να δουλέψω στο εργαστήριο Distributed Knowledge and Media Systems Group, καθώς επίσης και για τις πολύτιμες συμβουλές τις.

Επίσης θα ήθελα να ευχαριστήσω ιδιαίτερα τους Μουλό Βρεττό, Βαφειάδη Γιώργο και Κρανά Παύλο οι οποίοι μου αφιέρωσαν χρόνο και όρεξη, μου έδωσαν πολλές και χρήσιμες συμβουλές για προβλήματα υλοποίησης και όχι μόνο. Η υπομονετική καθοδήγησή τους ήταν καθοριστικής σημασίας για την εκπόνηση αυτής της διπλωματικής εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου που με έκανε τον άνθρωπο που είμαι σήμερα και, μαζί τους, τους φίλους μου και τη Βασιλική για την στήριξη που μου παρείχαν όλα αυτά τα χρόνια.

Περίληψη

Ο σκοπός αυτής της διπλωματικής εργασίας είναι ο σχεδιασμός και η υλοποίηση μιας πλατφόρμας για τη διαχείριση υποδομής cloud computing. Η πλατφόρμα αυτή εκμεταλλεύεται τα οφέλη που προκύπτουν από την παροχή REST API από τους παρόχους υπηρεσιών cloud computing για τον χειρισμό των εικονικών μηχανών καθώς και από τις τεχνικές configuration management για τη γρήγορη και εύκολη εγκατάσταση πακέτων λογισμικού.

Αρχικά έγινε μελέτη των απαιτήσεων για την υλοποίηση της πλατφόρμας, λειτουργικές και μη, με σκοπό τον προσδιορισμό του τρόπου λειτουργίας της. Οι απαιτήσεις αυτές προέκυψαν από μελέτη των ήδη υπαρχουσών λύσεων και το τι προσφέρουν στους χρήστες. Έπειτα, έγινε λεπτομερής σχεδιασμός της πλατφόρμας. Ορίστηκαν με σαφήνεια τα διακριτά τμήματα της πλατφόρμας καθώς επίσης και οι αρμοδιότητές τους. Στη συνέχεια επιλέχθηκε η κατάλληλη αρχιτεκτονική για κάθε τμήμα ενώ επιλέχθηκαν και τα εργαλεία λογισμικού που θεωρήθηκε ότι θα βοηθήσουν στην επίτευξη του επιθυμητού αποτελέσματος.

Το μοντέλο που πρεσβεύει η πλατφόρμα που υλοποιήθηκε στα πλαίσια αυτής της εργασίας στοχεύει στην ενοποίηση της διαχείρισης της υποδομής cloud computing από διαφορετικούς παρόχους. Αυτό δίνει τη δυνατότητα στους χρήστες να μην δεσμεύονται με τις υπηρεσίες μόνο ενός παρόχου αλλά να επιλέγουν και να συνδυάζουν υπηρεσίες από διαφορετικούς ανάλογα με τις ανάγκες των χρηστών αλλά και τις αντίστοιχες τιμολογιακές πολιτικές.

Λέξεις Κλειδιά: <<REST API, Cloud Computing, NoSQL, Chef, Configuration Management, Ruby on Rails, MVC >>

Abstract

The scope of this thesis is the design and implementation of a platform for managing cloud infrastructure. This platform takes advantage of the benefits of the REST API provision from the cloud computing services providers, which is used for handling virtual machines, and configuration management techniques, which are useful for installing software packages easily and quickly.

At first, there a study was made regarding the requirements, operational and non-operational, of the platform's implementation in order to determine the mode of operation. These requirements occurred as a result of studying the already existing solutions and what they offer to users. The next step was the platform's detailed design. Its distinct parts were defined clearly as well as their responsibilities. Subsequently, the most suitable architecture was chosen for each part and the software tools that were considered to be able to help achieve the desired result were selected.

The model which is represented by the platform implemented in the context of this thesis aims in the unification of the management of cloud computing infrastructure of different providers. This enables the users to avoid being bound to the services of a single provider and instead to be able to choose and combine services from different providers depending on their needs and the respective pricing policies.

Keywords: << REST API, Cloud Computing, NoSQL, Chef, Configuration Management, Ruby on Rails, MVC >>

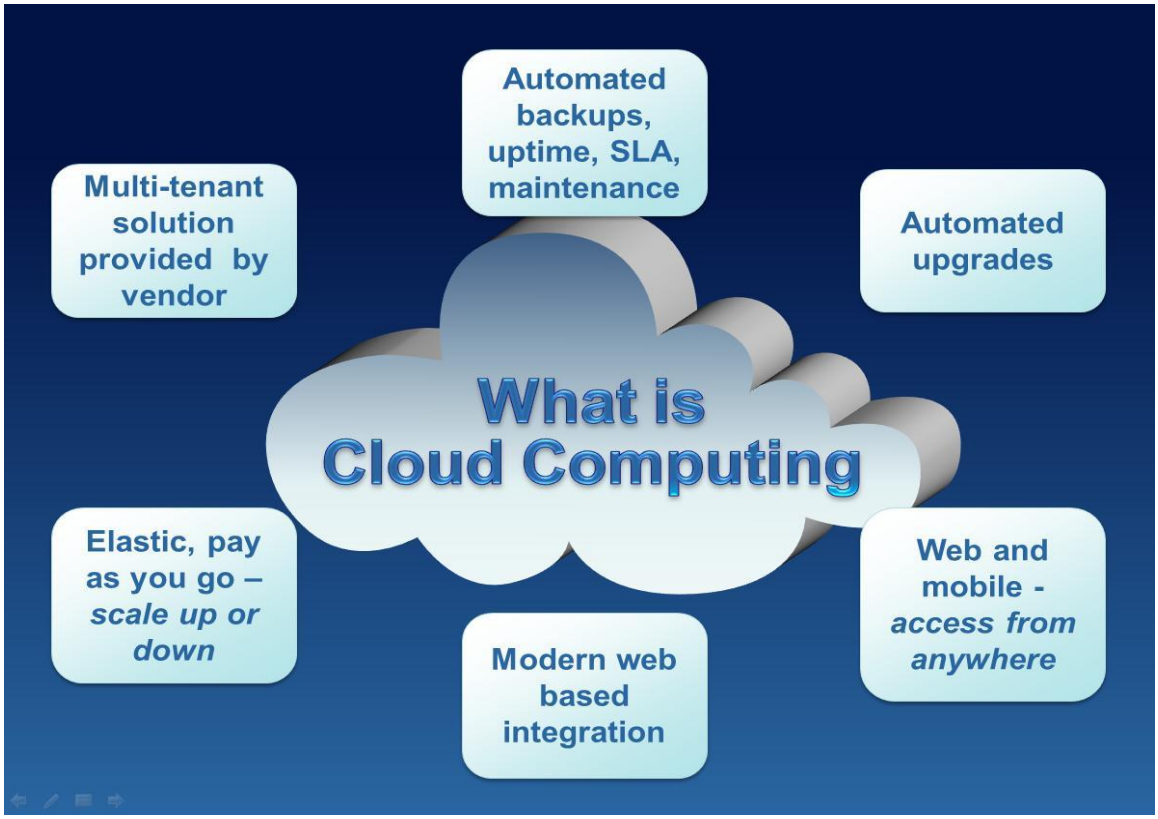
Περιεχόμενα

1
Cloud Computing

1.1 Εισαγωγή

Cloud Computing (ή Υπολογιστικό Νέφος) ονομάζεται η παροχή υπολογιστικών πόρων και υπηρεσιών και βασίζεται στην διανομή διαμοιραζόμενων πόρων ώστε να επιτύχει συνέπεια και κλιμακωσιμότητα σαν μια δημόσια υπηρεσία (όπως το ηλεκτρικό δίκτυο). Η διάθεση των παραπάνω γίνεται με τη βοήθεια του διαδικτύου που επιτρέπει στον χρήστη την πρόσβαση σε απομακρυσμένους υπολογιστές που βρίσκονται σε μεγάλα datacenters. Επιπλέον, υπάρχει μεγάλη ευελιξία και παραμετροποίηση των πόρων και των υπηρεσιών, τέτοια ώστε να μπορούν να ικανοποιηθούν αναγκές και οικιακών χρηστών αλλά και μεγάλων εταιριών.

Επιπλέον μέλημα του cloud computing είναι η βέλτιστη χρησιμοποίηση των διαμοιραζόμενων πόρων καθώς συνήθως οι πόροι, εκτός του ότι μοιράζονται μεταξύ των χρηστών, ανακατανέμονται δυναμικά ανάλογα με τη ζήτηση. Αυτή η τακτική είναι και πιο φιλική προς το περιβάλλον καθώς βοηθάει στη μείωση της κατανάλωσης ηλεκτρικής ισχύος στα datacenters.

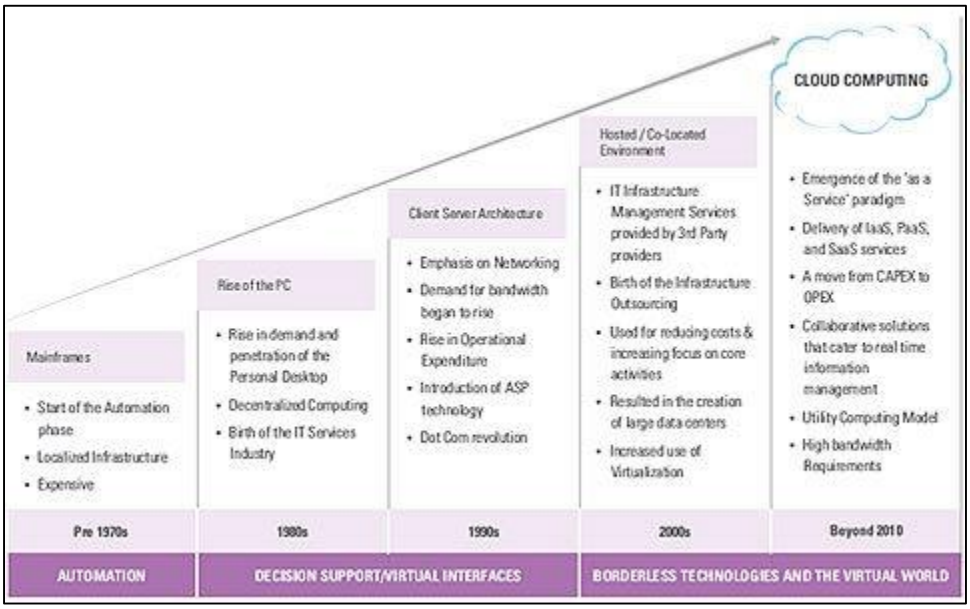


Εικόνα 1.1 – Τι είναι το Cloud Computing

Ακόμα, πολλές εταιρείες σήμερα, αντί να φτιάχνουν τη δική τους υπολογιστική υποδομή αγοράζοντας το απαιτούμενο υλικό και λογισμικό, επιλέγουν να χρησιμοποιήσουν διαμοιραζόμενες υποδομές, μέσω cloud computing, και να πληρώνουν ανάλογα με τη χρήση τους. Με αυτόν τον τρόπο αποφεύγουν τα μεγάλα αρχικά κόστη για τις υποδομές και μπορούν να χρησιμοποίησουν τα αντίστοιχα κεφάλαια στον τομέα δραστηριοποίησής τους ενώ γίνεται και πιο εύκολη η διαχείριση και συντήρηση των υπηρεσιών καθώς και η προσαρμογή σε αύξηση της ζήτησης.

1.2 Ιστορικη αναδρομή

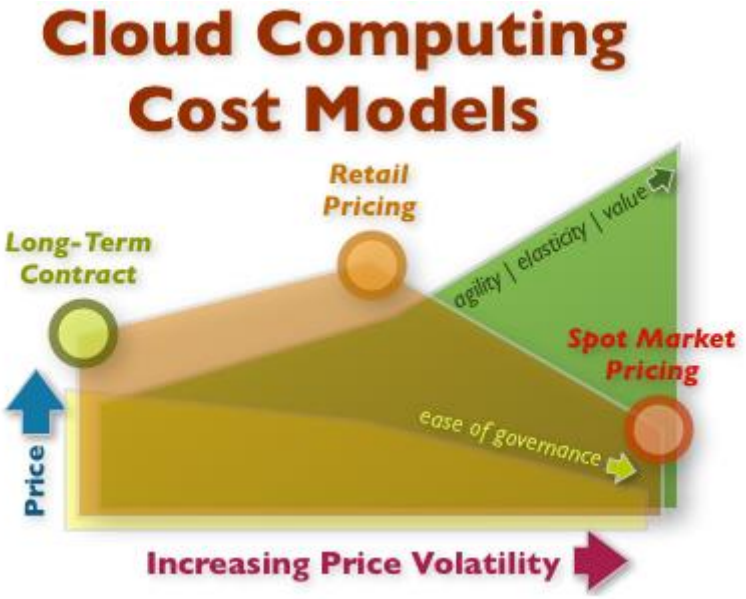
Όταν μιλάμε για cloud computing αναφερόμαστε κυρίως σε καταστάσεις, προϊόντα και ιδέες που ξεκίνησαν τον 21ο αιώνα. Παρόλα αυτά, οι ιδέες πάνω στις οποίες βασίστηκε υπήρχαν πολλά χρόνια πριν.
Η αρχή έγινε τη δεκαετία του 1950 με τη χρήση των mainframes. Πολλαπλοί χρήστες μπορούσαν να έχουν πρόσβαση σε έναν κεντρικό υπολογιστή μέσω απλών τερματικών, των οποίων η μόνη λειτουργία ήταν να παρέχουν προβαση στο mainframe. Λόγω του μεγάλου κόστους αγοράς αλλά και συντήρησης ενός mainframe, δεν ήταν λογικό ένας οργανισμός να παρέχει ένα σε κάθε υπάλληλο. Επιπλέον, ο μέσος χρήστης δεν χρειαζόταν τον μεγάλο αποθηκευτικό χώρο και την μεγάλη υπολογιστική ισχύ ενός τέτοιου υπολογιστή. Έτσι, η βέλτιστη λύση από οικονομικής άποψης, σύμφωνα με τα παραπάνω δεδομένα, ήταν η κοινή χρήση των υπολογιστικών πόρων ενός κεντρικού mainframe από όλους τους υπάλληλους.



Εικόνα 1.2 – Ιστορία του Cloud Computing

Τη δεκαετία του 1960, ο John McCarthy υποστήριξε ότι κάποια στιγμή, το υπολογιστικό κόστος θα μπορούσε να αποτελέσει μία δημόσια υπηρεσία. Ο Douglas Parkhill ανέλυσε διεξοδικά χαρακτηριστικά όπως η ελαστικότητα, η παροχή υπολογιστικού κόστους ως υπηρεσία, η πρόσβαση μέσω διαδικτύου και η ψευδαίσθηση απεριόριστων πόρων. Παράλληλα, ο Herb

Grosch, που έχει γράψει το νόμο του Grosch, ισχυρίστηκε ότι ολόκληρος το 1960 ο κόσμος θα μπορούσε να δουλέψει σε τερματικά που τροφοδοτούνται από 15 μεγάλα datacenters.
Στη συνέχεια, περίπου τη δεκαετία του 1970, με την κυκλοφορία του λειτουργικού συστήματος VM από την IBM, δημιουργήθηκε το concept των virtual machines (εικονικών μηχανών). Με τη χρήση λογισμικού εικονικοποίησης κατέστη δυνατή η ταυτόχρονη εκτέλεση περισσότερων του ενός λειτουργικών συστημάτων σε ένα απομονωμένο περιβάλλον. Με αυτόν τον τρόπο μπορούσαν πλήρεις, εικονικοί, υπολογιστές να εκτελούνται σε υλικό που αντιστοιχούσε σε έναν πραγματικό υπολογιστή. Η εικονικοποίηση έπαιξε καταλυτικό ρόλο στην διαμόρφωση του cloud όπως το γνωρίζουμε σήμερα.
Τη δεκαετία του 1990, οι εταιρίες τηλεπικοινωνιών που μέχρι πρότερα παρείχαν αποκλειστικά συνδέσεις σημείου-σε-σημείο, άρχισαν να παρέχουν και υπηρεσίες εικονικών ιδιωτικών δικτύων. Με κατάλληλη προσαρμογή της κυκλοφορίας των δεδομένων, ώστε να βελτιστοποιείται η χρήση του κάθε εξυπηρετητή, μπορούσαν να χρησιμοποιήσουν το συνολικό εύρος ζώνης του δικτύου πιο αποτελεσματικά.



Εικόνα 1.3 – Το μοντέλο κοστολόγησης στο Cloud Computing

Τη δεκαετία του 2000, η Amazon έπαιξε καθοριστικό ρόλο στην ανάπτυξη του υπολογιστικού νέφους, με τον εκσυγχρονισμό των datacenter της, τα οποία χρησιμοποιούσαν μόλις το 10% της μέγιστης δυνατότητάς τους, αφήνοντας περιθώριο για περιστασιακές ανάγκες. Το 2006, ξεκίνησε μια νέα προσπάθεια για την ανάπτυξη και την παροχή νέφους σε εξωτερικούς πελάτες, παρουσιάζοντας το Amazon Web Services (AWS) και το Amazon Elastic Compute Cloud (EC2), το οποίο επιτρέπει σε μικρές επιχειρήσεις και ιδιώτες να νοικιάζουν υπολογιστικούς πόρους. Η IBM ήταν η πρώτη που άρχισε να χαράσσει μια σαφή στρατηγική για το υπολογιστικό νέφος το 2007, έχοντας ως σκοπό την κατασκευή υπηρεσιών υπολογιστικού νέφους για επιχειρήσεις. Επίσης, ανακοίνωσε συνεργασία με την Google για την προώθηση του υπολογιστικού νέφους στα πανεπιστήμια.

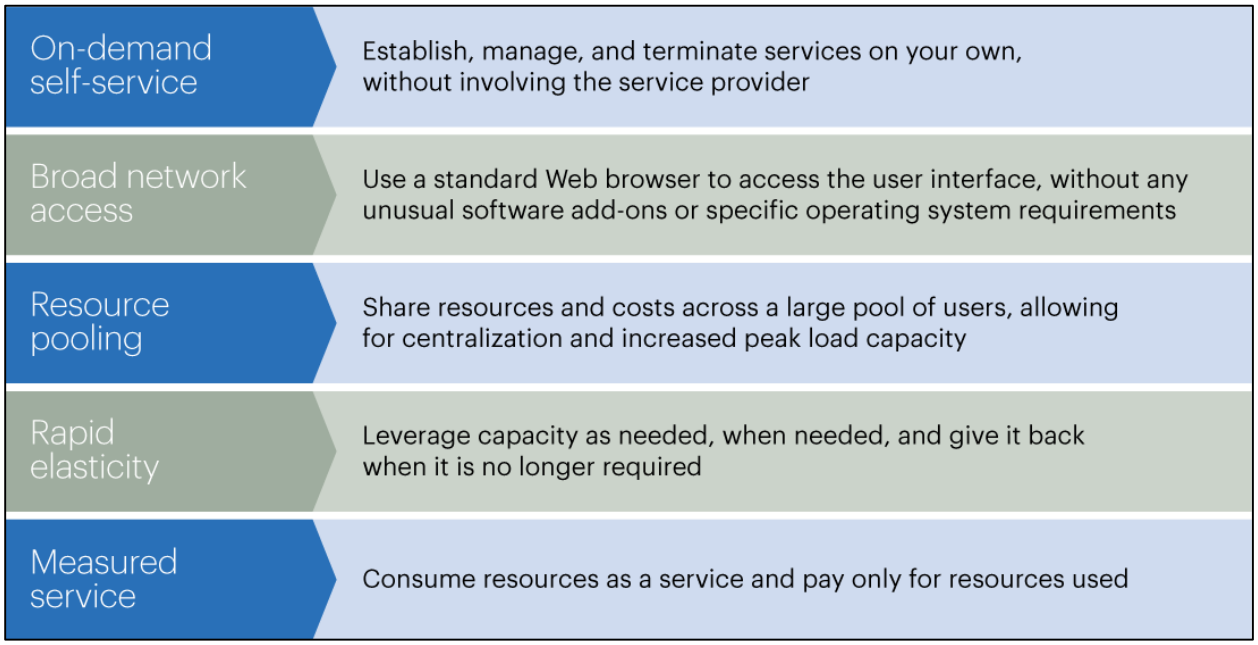
Στις αρχές του 2008, το Eucalyptus έγινε η πρώτη συμβατή πλατφόρμα με την προγραμματιστική διεπαφή του AWS για την ανάπτυξη των ιδιωτικών υπολογιστικών νεφών, ενώ το OpenNebula, έγινε το πρώτο λογισμικό ανοιχτού κώδικα για την ανάπτυξη ιδιωτικών και υβριδικών υπολογιστικών νεφών. Το ίδιο έτος, οι προσπάθειες επικεντρώθηκαν στην παροχή εγγυήσεων ποιότητας υπηρεσιών (όπως απαιτείται σε διαδραστικές εφαρμογές πραγματικού χρόνου) σε υποδομές υπολογιστικού νέφους. Η εξέλιξη του διαδικτύου και η νέα οπτική για την παροχή όλο και περισσότερων υπηρεσιών, σε συνδιασμό με τα ταχύτατα δίκτυα, το χαμηλό κόστος των υπολογιστών και των μέσων αποθήκευσης και την ευρεία υιοθέτηση υπηρεσιοστρεφών αρχιτεκτονικών (Service Oriented Architecture - SOA) και εικονικών μηχανών (hardware virtualization), οδηγεί στην περαιτέρω ανάπτυξη του υπολογιστικού νέφους.

1.3 Χαρακτηριστικά

Σε μια προσπάθεια να περιγραφεί ακόμα πιο συγκεκριμένα το cloud computing, παρουσιάζονται σε αυτή την ενότητα τα χαρακτηριστικά του.

- Αυτοεξυπηρέτηση κατ'απαίτηση

Ο χρήστης μπορεί όποτε θέλει να χρησιμοποιήσει μια υπηρεσία ή να δεσμεύει υπολογιστικούς πόρους, όπως μονάδες επεξεργασίας και αποθήκευσης, χωρίς την ενδιάμεση αλληλεπίδραση με τον πάροχο των υπηρεσιών.



Εικόνα 1.3 – Χαρακτηριστικά του Cloud Computing

- Ευρεία πρόσβαση στο δίκτυο

Οι υπολογιστικοί πόροι είναι διαθέσιμοι μέσω του δικτύου και η πρόσβαση γίνεται μέσω τυποποιημένων μηχανισμών από συσκευές-πελάτες (π.χ. κινητά τηλέφωνα, ταμπλέτες, φορητοί υπολογιστές και σταθμούς εργασίας).

- Διάθεση πόρων

Οι υπολογιστικοί πόροι του παρόχου συγκεντρώνονται για να διατεθούν στους χρήστες, με τη χρήση ενός πολυπελατιακού μοντέλου, σύμφωνα με το οποίο οι διάφοροι φυσικοί και εικονικοί πόροι εκχωρούνται δυναμικά και αναδιανέμονται, ανάλογα με τη ζήτηση των χρηστών. Ο πελάτης δεν έχει γενικά κανένα έλεγχο ή γνώση σχετικά με την ακριβή τοποθεσία των παρεχόμενων πόρων, αλλά μπορεί να είναι σε θέση να προσδιορίζει τη θέση σε ένα υψηλότερο επίπεδο αφαίρεσης (π.χ. χώρα ή δαταςεντερ), προσδίδοντας μία χωρική ανεξαρτησία. Παραδείγματα υπολογιστικών πόρων μπορεί να είναι η αποθήκευση, η επεξεργασία, η μνήμη και το εύρος ζώνης του δικτύου.

- Ταχεία ελαστικότητα

Οι υπολογιστικοί πόροι μπορούν να δεσμεύονται και να αποδεσμεύονται ελαστικά και σε ορισμένες περιπτώσεις, αυτόματα, ώστε το υπολογιστικό νέφος να κλιμακώνεται ανάλογα με τη ζήτηση. Ο χρήστης έχει την αίσθηση ότι οι υπολογιστικοί πόροι είναι απεριόριστοι και μπορούν να διατεθούν σε οποιαδήποτε ποσότητα κι ανά πάσα στιγμή.

- Μετρούμενη υπηρεσία

Τα συστήματα υπολογιστικού νέφους ελέγχονται και βελτιστοποιούνται αυτόματα, αξιοποιώντας ένα σύστημα μέτρησης (συνήθως pay-per-use, charge-per-use). Η χρήση των πόρων μπορεί να παρακολουθείται, να ελέγχεται, και να γίνεται αναφορά, παρέχοντας διαφάνεια τόσο στον πάροχο όσο και στο χρήστη για την υπηρεσία που χρησιμοποιείται.

- Κλιμακωσιμότητα

Στο cloud είναι δυνατή η χειροκίνητη ή δυναμική προσθήκη και αφαίρεση κόμβων επεξεργασίας, εκτέλεσης εργασιών ή αποθήκευσης, ανάλογα με την αυξομείωση των απαιτήσεων, χωρίς να αλλοιώνεται η υπηρεσία που παρέχεται στο χρήστη.

- Εικονικοποίηση

Οι λεπτομέρειες υλοποίησης και κατάστασης στην οποία βρίσκονται οι υπολογιστικοί πόροι αποκρύπτονται από το χρήστη. Οι εικονικές μηχανές (Virtual Machines) δίνουν την εντύπωση στο χρήστη ενός ολοκληρωμένου φυσικού μηχανήματος, ενώ στην πραγματικότητας είναι ένα σύνολο αρχείων και προγραμμάτων που τρέχουν πάνω σε ένα άλλο (ή άλλα) φυσικά μηχανήματα. Οι εικονικές μηχανές μπορεί να έχουν διαφορετικά χαρακτηριστικά από τους φυσικούς πόρους πάνω στους οποίους τρέχουν, τόσο όσον αφορά το λογισμικό, όσο και για το υλικό. Οι cloud υπηρεσίες συχνά αναφέρονται ως επεκτάσιμες σε δεκάδες χιλιάδες, εκατοντάδες χιλιάδες, εκατομμύρια ή περισσότερους ταυτόχρονους χρήστες. Αυτό σημαίνει ότι σε μέγιστη χωρητικότητα (συνήθως θεωρείται το 80%), το σύστημα μπορεί να εξυπηρετήσει τόσους χρήστες χωρίς να παρουσιάσει σφάλμα σε κανένα χρήστη ή να καταρρεύσει συνολικά λόγω εξάντλησης πόρων.

- Αξιοπιστία

Το υπολογιστικό νέφος εγγυάται την ορθή λειτουργία των προγραμμάτων των χρηστών, την αποθήκευση των δεδομένων τους και την αξιόπιστη μεταφορά τους. Για την εξασφάλιση της

αξιοπιστίας, τα δεδομένα αποθηκεύονται σε περισσότερους από έναν κόμβους στο νέφος. Ο συνηθισμένος αριθμός ντιγράφων είναι 3 (replication level three). Επίσης, εφόσον τα δεδομένα του χρήστη βρίσκονται σε κάποια μηχανήματα σε ένα datacenter, μία βλάβη στο τερματικό του χρήστη δεν θα έχει καμία επίδραση στα δεδομένα αυτά.
Για τη μέτρηση της αξιοπιστίας χρησιμοποιούνται δείκτες, όπως ο μέσος χρόνος μεταξύ δύο διαδοχικών αποτυχιών του συστήματος (MTBF - Mean Time Between Failures), ο μέσος χρόνος μέχρι την αποτυχία του συστήματος (MTTF - Mean Time To Failure) και ο μέσος χρόνος μέχρι την επισκευή ενός συστήματος (MTTR - Mean Time To Repair).

- Συντήρηση

Η συντήρηση στο cloud, με την έννοια της αποσφαλμάτωσης των εφαρμογών και της αναβάθμισης των εκδόσεών τους, δεν είναι πλέον ευθύνη του χρήστη. Ο πάροχος φροντίζει για την ορθή και ομαλή λειτουργία των υπηρεσιών που προσφέρονται στο cloud, ο οποίος πλέον δεν χρειάζεται να ασχολείται με κάθε χρήστη ξεχωριστά, αφού είναι δυνατό να βελτιώσει την εμπειρία χρήσης σε όλους τους χρήστες ταυτόχρονα. Επίσης, οφείλει να προσφέρει συμβατό λογισμικό ανεξάρτητα από τη συσκευή-τερματικό (PC, smartphone, tablet κτλ) που θα επιλέξει ο χρήστης για να τρέξει μια εφαρμογή.

- Πολυπελατιακότητα

Κάθε χρήστης μιας υπηρεσίας υπολογιστικού νέφους, δεν χρειάζεται να έχει δικό του αντίγραφο της εφαρμογής. Αρκεί ένα μοναδικό στιγμιότυπο (instance) το οποίο μπορεί να προσαρμοστεί στις ανάγκες του κάθε χρήστη. Αυτό έχει ως αποτέλεσμα την εξοικονόμηση πόρων στο νέφος και την ευκολότερη συντήρηση της εφαρμογής. Οι πληροφορίες των χρηστών αποθηκεύονται σε ένα κεντρικό σημείο και έτσι διευκολύνεται η εξαγωγή στατιστικών στοιχείων σχετικά με την εφαρμογή που βοηθούν τον πάροχο να την βελτιώση και να την προσαρμόσει ακόμα καλύτερα στις ανάγκες των χρηστών.

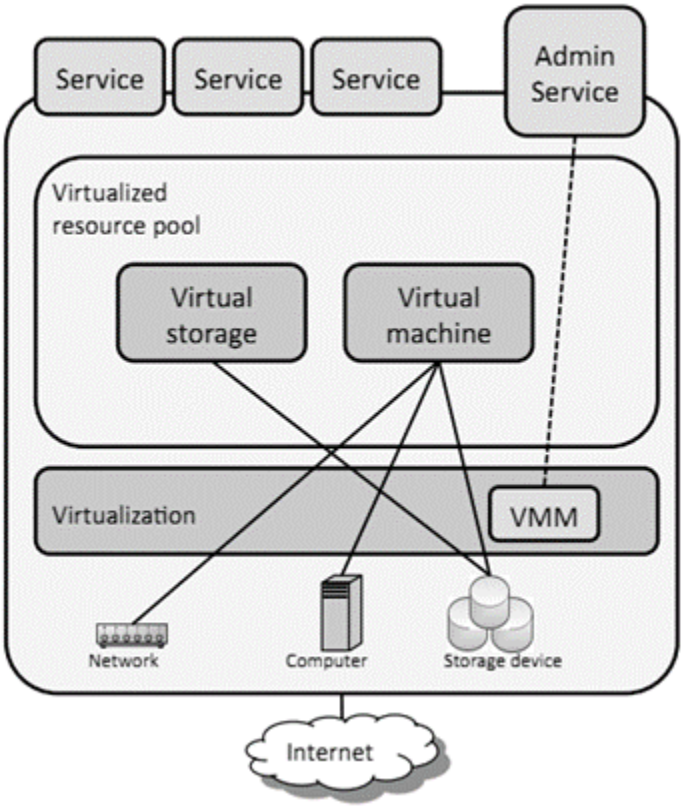
1.4 Υπηρεσίες

Ο διαχωρισμός των υπηρεσιών νέφους γίνεται με βάση το αφαιρετικό επίπεδο ελέγχου των χρηστών στους υπολογιστικούς πόρους που τους παρέχονται. Τα πιο σημαντικά μοντέλα, σε φθίνουσα σειρά με βάση το πόσο έλεγχο επιτρέπουν στον χρήστη, είναι η υποδομή ως υπηρεσία (IaaS), η πλατφόρμα ως υπηρεσία (PaaS) και το λογισμικό ως υπηρεσία (SaaS) ενώ υπάρχουν και άλλα μοντέλα που δεν εντάσσονται στην παραπάνω κατηγοριοποίηση όπως το οτιδήποτε ως υπηρεσία (XaaS) και το δίκτυο ως υπηρεσία (NaaS).

- Υποδομή ως Υπηρεσία (IaaS)

Το IaaS είναι το πρώτο στρώμα του cloud computing και αποτελεί το θεμέλιό του και μέσω αυτού παρέχετεαι πρόσβαση σε ουσιώδεις υπολογιστικούς πόρους. Οι φυσικοί πόροι εικονικοποιούνται, το οποίο σημαίνει ότι μπορούν να μοιραστούν από διαφορετικά λειτουργικά συστήματα και περιβάλλοντα χρηστών - ιδανικά - χωρίς αμοιβαίες παρεμβολές. Χρησιμοποιώντας αυτό το μοντέλο υπηρεσίας, ο χρήστης μπορεί να διαχειριστεί τις εφαρμογές, τα δεδομένα, το λειτουργικό σύστημα, το middleware και τον χρόνο λειτουργίας. Ο πάροχος διαχειρίζεται την εικονικοποίηση, τους servers, τη δικτύωση και την αποθήκευση. Αυτό επιτρέπει στον χρήστη να αποφύγει δαπάνες υλικού, καθώς σε αυτό το μοντέλο η χρέωση αφορά μόνο

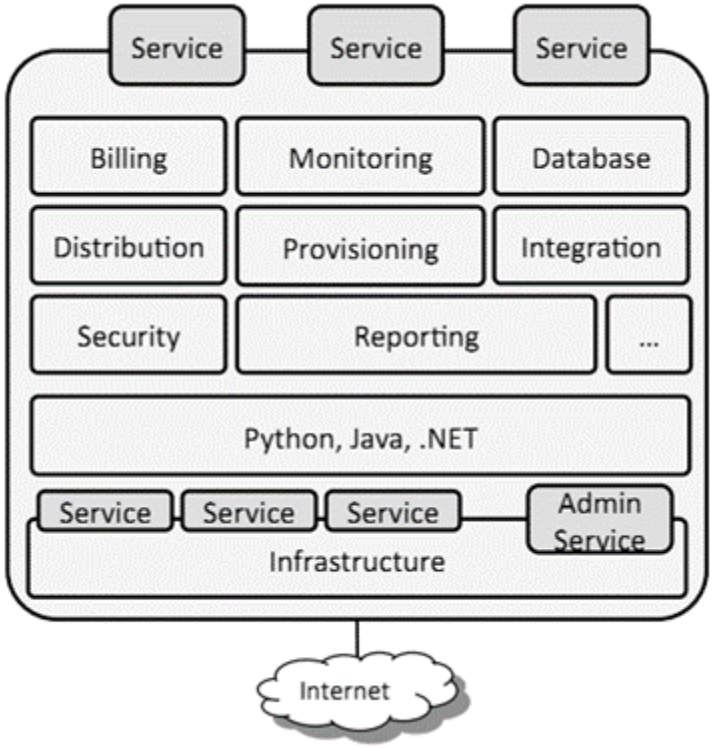
τους πόρους που χρησιμοποιούνται, και να βελτιστοποίησει και αυτοματοποιήσει την κλιμάκωση. Κάποιοι γνωστοί πάροχοι IaaS είναι οι: Amazon, Microsoft, VMWare, Rackspace και Red Hat.



Εικόνα 1.4 - IaaS Cloud Computing

- Πλατφόρμα ως Υπηρεσία (PaaS)

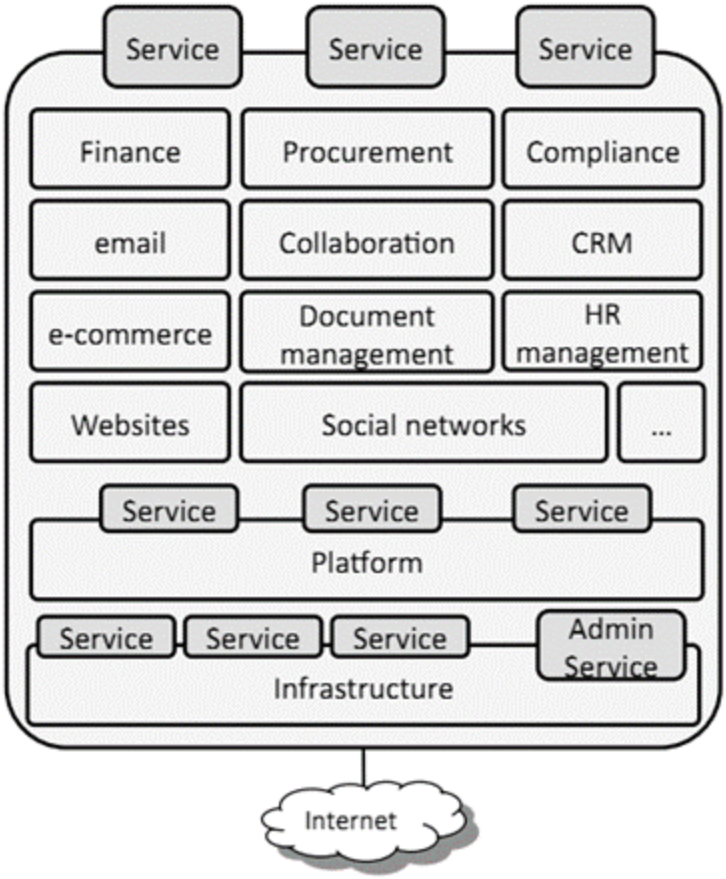
Αυτό το μοντέλο θα μπορούσε να θεωρηθεί το δεύτερο στρώμα και αποτελεί το περιβάλλον για ανάπτυξη cloud εφαρμογών. Ο χρήστης διαχειρίζεται τις εφαρμογές και τα δεδομένα και ο πάροχος αναλαμβάνει και επιβλέπει όλα τα υπόλοιπα (π.χ. διανομή της εφαρμογής στην υποκείμενη υποδομή). Επιπλέον, ο πάροχος υποστηρίζει τον χρήστη με ένα σύνολο βασικών υπηρεσιών για να βοηθήσει την επικοινωνία, την παρακολούθηση και τη χρέωση καθώς και διάφορα άλλα κομμάτια για την διευκόλυνση εκκίνησης μιας εφαρμογής, την εξασφάλιση της κλιμακωσιμότητας και/ή ελαστικότητάς της. Οι υπηρεσίες αυτού του μοντέλου προσφέρουν έναν συμβιβασμό μεταξύ πολυπλοκότητας και ευελιξίας που επιτρέπει την γρήγορη υλοποίηση και διάθεση των εφαρμογών όμως εισάγουν περιορισμούς σχετικά με τις γλώσσες και τα μοντέλα προγραμματισμου που υποστηρίζονται και την πρόσβαση σε πόρους. Μια δημοφιλής PaaS υπηρεσία είναι το Google App Engine.



Εικόνα 1.5 - PaaS Cloud Computing

- Λογισμικό ως Υπηρεσία (SaaS)

Αυτό είναι το τρίτο και τελευταίο στρώμα και προσφέρει πλήρεις εφαρμογές στον τελικό χρήστη του cloud. Σε αυτό το μοντέλο τα πάντα τα διαχειρίζεται ο πάροχος και έτσι προσφέρει εξασφαλισμένη συμβατότητα και ευκολότερη συνεργασία, καθώς όλοι χρησιμοποιούν το ίδιο λογισμικό. Η πρόσβαση σε τέτοιες υπηρεσίες γίνεται μέσω διαδικτύου και υπηρεσιοστραφών αρχιτεκτονικών που βασίζονται σε τεχνολογίες υπηρεσιών διαδικτύου. Οι χρήστες έχουν τη δυνατότητα να χρησιμοποιήσουν μια τέτοια υπηρεσία από μια πλειάδα συσκευών (PC, smartphone, tablet κτλ) το οποίο παρέχει μεγάλη ευελιξία στην επικοινωνία και τη συνεργασία μεταξύ τους. Αυτό το μοντέλο μπορεί να θεωρηθεί επέκταση του ASP (Application Service Provider) μοντέλου όπου μια εφαρμογή τρέχει, συντηρείται και υποστηρίζεται από τον πάροχο της υπηρεσίας. Οι διαφορές με το ASP είναι η ενθυλάκωση της εφαρμογής ως υπηρεσία, η δυναμική παροχή πόρων και η χρέωση σε μονάδες χρήσης (pay as you go). Παρόλα αυτά και τα δύο μοντέλα έχουν ως στόχο το να εστιάσουν στις βασικές αρμοδιότητες με το να αναθέτουν σε τρίτους τις εφαρμογές.



Εικόνα 1.6 - SaaS Cloud Computing

1.5 Είδη

Στη συνέχεια παρουσιάζονται τα τέσσερα κύρια μοντέλα ανάπτυξης υπολογιστικού νέφους:

- Δημόσιο νέφος (Public cloud)

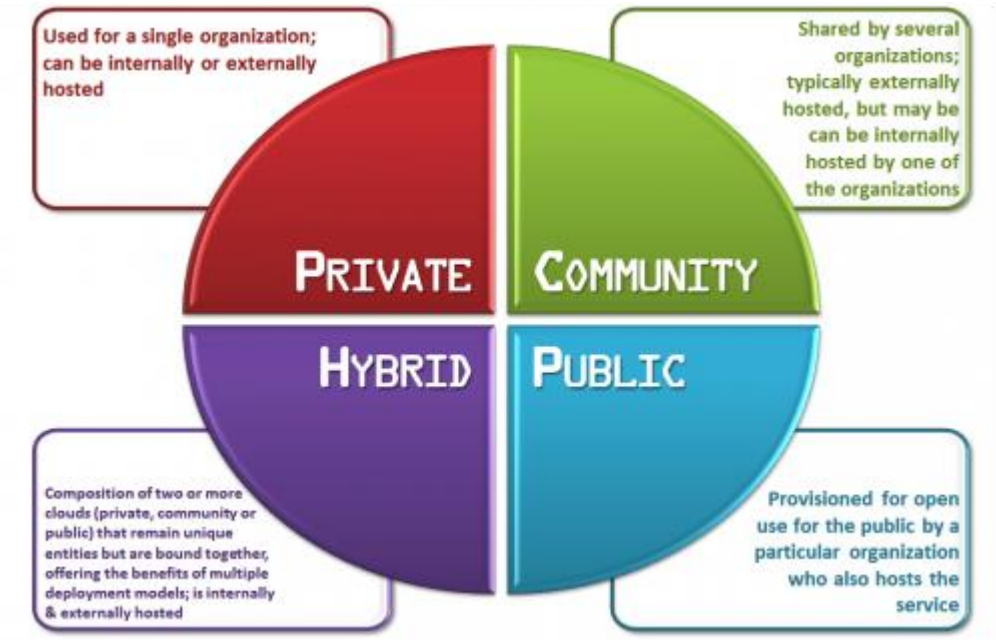
Τα δημόσια νέφη γίνονται διαθέσιμα στο γενικό κοινό από έναν πάροχο που φιλοξενεί την υποδομή. Γενικά, πάροχοι δημόσιων νεφών όπως οι Amazon AWS, Microsoft και Google κατέχουν και χειρίζονται την υποδομή και προσφέρουν πρόσβαση μέσω διαδικτύου. Με αυτό το μοντέλο, οι χρήστες δεν έχουν γνώση ή έλεγχο για το που βρίσκονται τα μηχανήματα που φιλοξενούν το cloud. Στα δημόσια νέφη, όλοι οι χρήστες μοιράζονται τους ίδιους υπολογιστικούς πόρους και έχουν περιορισμένη παραμετροποίηση και ασφάλεια.
Στα δημόσια νέφη, οι χρήστες ωφελούνται όσον αφορά το κόστος γιατί οι δαπάνες συντήρησης των υποδομών μοιράζονται σε όλους και έτσι ισχύει για τον καθένα ένα μοντέλο χαμηλού κόστους pay as you go. Επιπλέον, καθώς τα δημόσια νέφη είναι συνήθως μεγαλύτερα από τα ιδιωτικά, παρέχουν καλύτερη και απρόσκοπτη κλιμάκωση κατ'απαίτηση. Έτσι, αυτά τα νέφη έχουν βέλτιστη αξιοποίηση των κοινών πόρων όμως είναι πιο ευάλωτα από τα ιδιωτικά.

- Ιδιωτικό νέφος (Private cloud)

Ο όρος “ιδιωτικό νέφος” αναφέρεται σε υποδομή νέφους που προορίζεται για έναν συγκεκριμένο οργανισμό. Τα ιδιωτικά νέφη επιτρέπουν σε επιχειρήσεις να φιλοξενούν εφαρμογές στο cloud και παράλληλα να διευθετούν θέματα που αφορούν την ασφάλεια και των έλεγχο δεδομένων, που συνήθως λείπουν από ένα περιβάλλον δημόσιου νέφους. Δεν είναι μοιραζόμενο με άλλους οργανισμούς, είτε γίνεται εσωτερική διαχείριση είτε από τρίτους, και μπορεί να φιλοξενείται εσωτερικά ή εξωτερικά.

Υπάρχουν δύο παραλλαγές ιδιωτικών νεφών:

1. On-Premise ιδιωτικό νέφος: Αυτός ο τύπος cloud φιλοξενείται σε μηχανήματα εντός των εγκαταστάσεων του οργανισμού και χρησιμοποιείται για εφαρμογές στις οποίες απαιτείται πλήρης έλεγχος και παραμετροποίηση της υποδομής καθώς επίσης και ασφάλεια.
2. Externally Hosted ιδιωτικό νέφος: Αυτός ο τύπος cloud φιλοξενείται σε μηχανήματα εκτός των εγκαταστάσεων της επιχείρησης, συνήθως από κάποιον τρίτο που ειδικεύεται στην υποδομή cloud. Ο πάροχος δημιουργεί ένα αποκλειστικό περιβάλλον cloud και εγγυάται πλήρως για την ιδιωτικότητά του.



Εικόνα 1.7 - Είδη Cloud Computing

- Υβριδικό νέφος (Hybrid cloud)

Τα υβριδικά νέφη αποτελούν σύνθεση δύο ή περισσότερων cloud (ιδιωτικών, δημόσιων ή κοινότητας) τα οποία παραμένουν αυτόνομες οντότητες αλλά συνδέονται έτσι ώστε να προσφέρουν τα πλεονεκτήματα που έχει το κάθε είδος. Σε ένα υβριδικό νέφος μπορεί να γίνει αξιοποίηση των πλεονεκτημάτων των “τρίτων“ παρόχων με πλήρη ή μερικό τρόπο, αυξάνοντας την ευελιξία των υπολογιστικών πόρων. Η ενίσχυση ενός κλασικού ιδιωτικού νέφους με τους πόρους ενός δημόσιου μπορεί να χρησιμοποιηθεί για να αντιμετωπιστούν απρόοπτες απότομες αυξήσεις του φόρτου εργασίας. Τα υβριδικά νέφη χρειάζονται υποδομή υλικού και εντός των

εγκαταστάσεων του οργανισμού και από κάποιον “τριτο” πάροχο. Το μειονέκτημα αυτού του είδους είναι ότι πρέπει να παρακολουθούνται πολλαπλές πλατφόρμες ασφάλειας cloud καθώς και το ότι πρέπει να εξασφαλίζεται ότι τα τμήματα που βρίσκονται σε διαφορετικά cloud μπορούν να επικοινωνούν μεταξύ τους.

- Νέφος κοινότητας (Community cloud)

Το νέφος κοινότητας είναι ένα πολυ-πελατιακό μοντέλο υπηρεσιών το οποίο διαμοιράζεται μεταξύ διάφορων οργανισμών στο οποίο ο έλεγχος, η διαχείριση και ασφάλεια γίνεται είτε από κοινού από όλους τους συμμετέχοντες είτε από κάποιο τρίτο. Τα νέφη κοινότητας είναι μια υβριδική μορφή ιδιωτικών νεφών που δημιουργούνται και λειτουργούν συγκεκριμένα για μια ομάδα. Αυτές οι κοινότητες έχουν παρόμοιες απαιτήσεις ως προς το cloud και ο απώτερος σκοπός τους είναι να επιτευχθούν οι επιμέρους στόχοι του καθενός μέσα από τη συνεργασία. Επιδίωξη των νεφών κοινότητας είναι να μπορέσουν οι συμμετέχοντες οργανισμοί να αξιοποιήσουν τα οφέλη ενός δημόσιου νέφους με το επιπρόσθετο επίπεδο ιδιωτικότητας, ασφάλειας και συμμόρφωσης σε μια κοινή πολιτική που συνήθως σχετίζεται με τα ιδιωτικά νέφη.

1.6 Πλεονεκτήματα και μειονεκτήματα

Ο βαθμός αποδοχής για οποιοδήποτε προγραμματιστικό μοντέλο αξιολογείται από τα δυνατά και αδύνατα σημεία του. Αν τα πλεονεκτήματα είναι αρκετά και τα μειονεκτήματα τουλάχιστον ανεκτά μέχρι ενός σημείου, ο βαθμός αποδοχής είναι μεγάλος και το μοντέλο θα γίνει αποδεκτό από τους χρήστες. Ακολουθούν μερικά σημαντικά πλεονεκτήματα και μειονεκτήματα του cloud:

A. Πλεονεκτήματα

Το cloud προσφέρει πολλά οφέλη και ευελιξία στους χρήστες του. Ο χρήστης μπορεί να το χρησιμοποιήσει από οπουδήποτε, οποτεδήποτε και με ασφαλή τρόπο. Με τον αυξανόμενο αριθμό των συσκευών με δυνατότητα σύνδεσης στο διαδίκτυο που χρησιμοποιούνται σήμερα (π.χ. tablets, smartphones κτλ), η πρόσβαση κάποιου στα δεδομένα του πρέπει να είναι γρήγορη και εύκολη. Κάποια από τα οφέλη σε σχέση με τη χρήση του cloud είναι τα ακόλουθα:

- Μειώνει την αρχική επένδυση, το συνολικό κόστος ιδιοκτησίας, το συνολικό κόστος λειτουργίας και ελαχιστοποιεί τον επιχειρηματικό ρίσκο.

- Παρέχει δυναμική υποδομή που προσφέρει μειωμένο κόστος και βελτιωμένες υπηρεσίες με λιγότερα κόστη ανάπτυξης και συντήρησης.

- Παρέχει κατ’απαίτηση ευέλικτες, κλιμκώσιμες, βελτιωμένες και προσαρμόσιμες υπηρεσίες με μοντέλο πληρωμής pay-as-you go.

- Παρέχει συνεπή διαθεσιμότητα και απόδοση με αυτόματη αντιμετώπιση των απότομων αυξήσεων στο φόρτο εργασίας.

- Μπορεί να γίνει ανάκτηση της λειτουργίας του ταχύτατα και έχει βελτιωμένες δυνατότητες αποκατάστασης για βελτιωμένη επιχειρηματική ανάκαμψη.

- Παρέχει απεριόριστους υπολογιστικούς, αποθηκευτικούς και δικτυακούς πόρους με ελαστικό τρόπο.

- Προσφέρει αυτόματες αναβαθμίσεις λογισμικού, βελτιωμένη συμβατότητα τύπων εγγράφων και βελτιωμένη συμβατότητα μεταξύ διαφορετικών λειτουργικών συστημάτων.

- Προσφέρει εύκολη συνεργασία ομάδων, δηλαδή παρέχει ευελιξία στους χρήστης σε παγκόσμια κλίμακα για να δουλέψουν στο ίδιο έργο.

Εικόνα 1.8 – Πλεονεκτήματα και μειονεκτήματα του Cloud Computing

- Προσφέρει αυξημένη απόδοση της επένδυσης της υπάρχουσας ακίνητης περιουσίας, ελευθερώνοντας κεφάλαια για στρατηγική ανάπτυξη.

- Είναι φιλικό προς το περιβάλλον καθώς χρησιμοποιεί μόνο όσο χώρο χρειάζεται η εφαρμογή στον εξυπηρετητή το οποίο με τη σειρά του ελαχιστοποιεί το ενεργειακό αποτύπωμα

B. Μεονεκτήματα

Κάθε νόμισμα έχει δύο όψεις. Έτσι και το cloud έχει μειονεκτήματα. Κάποια από αυτά καταγράφονται παρακάτω:

- Απαιτεί δίκτυο υψηλής ταχύτητας και συνεχή σύνδεση.
- Η ιδιωτικότητα και η ασφάλεια δεν είναι βέλτιστες. Τα δεδομένα και οι εφαρμογές μπορεί να μην πολύ ασφαλείς.
- Οι περιπτώσεις σφάλματος είναι αναπόφευκτες και η ανάκαμψη δεν είναι πάντα δυνατή
- Οι χρήστες έχουν εξωτερική εξάρτηση από τον πάροχο του cloud για κρίσιμες για τους στόχους τους εφαρμογές
- Απαιτεί συνεχή παρακολούθηση και επιβολή των SLA (Service Level Agreements)

1.7 Ανοιχτά ζητήματα και προκλήσεις

Τα υπάρχοντα μοντέλα προγραμματισμού (π.χ. κατανεμημένα συστήματα, υπηρεσιοστραφής αρχιτεκτονική κτλ) είναι δομικά στοιχεία του cloud. Υπάρχουν πολυάριθμα ζητήματα που σχετίζονται με αυτά τα μοντέλα και κάποιες καινούργιες προκλήσεις που έχουν αναδειχθεί από το cloud απαιτείται να αντιμετωπιστούν σωστά έτσι ώστε να υλοποιηθεί το cloud στο μέγιστο βαθμό του. Αυτά τα ζητήματα πρέπει να αντιμετωπιστούν με σκοπό να παρέχονται υψηλής ποιότητας υπηρεσίες ενώ παράλληλα να γίνονται σεβαστές οι ανάγκες των παρόχων. Τα ζητήματα μπορούν να οργανωθούν σε διαφορετικές κατηγορίες όπως ασφάλεια, προστασία, διαχείριση ταυτοποίησης, διαχείριση πόρων, διαχείριση ισχύος και ενέργειας, απομόνωση δεδομένων, διαθεσιμότητα πόρων, ανομοιογένεια πόρων. Παρά το ότι υπάρχουν αρκετά θέματα που απαιτούν προσοχή, τα ακόλουθα μπορούν να αντιμετωπιστούν ως πρωταρχικό μέλημα:

A. Ασφάλεια και ιδιωτικότητα

Σύμφωνα με έρευνα του International Data Corporation (IDC), η ασφάλεια, η απόδοση και η διαθεσιμότητα είναι τα τρία μεγαλύτερα ζητήματα στην υλοποίηση του cloud. Η κρίσιμη πρόκληση είναι ο τρόπος αντιμετώπισης των ζητημάτων ασφάλειας και ιδιωτικότητας που προκύπτουν από τη μετακίνηση δεδομένων και εφαρμογών μέσω δικτύων, από απώλεια ελέγχου πάνω στα δεδομένα, από την ανομοιογένεια των πόρων και από διάφορες πολιτικές ασφάλειας. Η αποθήκευση των δεδομένων και η επεξεργασία και μετακίνησή τους εκτός των ορίων ελέγχου ενός οργανισμού θέτει έναν εγγενή κίνδυνο και τα κάνει ευάλωτα σε διάφορες επιθέσεις. Οι απειλές ασφάλειας μπορεί να είναι δύο τύπων, εσωτερικές ή εξωτερικές. Ο εξωτερικός κίνδυνος τίθεται από διάφορα άτομα και οργανισμούς όπως εχθροί ή hackers που

δεν έχουν άμεση πρόσβαση στο cloud. Ο εσωτερικός κίνδυνος είναι ένα γνωστό ζήτημα που μπορεί να τεθεί από θυγατρικές του οργανισμού, εργολάβους, τρέχοντες ή πρώην υπαλλήλους ή άλλους που είχαν κάποια στιγμή πρόσβαση στους εξυπηρετητές, στα δίκτυα και τα δεδομένα ενός οργανισμού για να διευκολύνουν τις δραστηριότητές του. Το cloud θέτει ζητήματα ιδιωτικότητας επειδή οι πάροχοι υπηρεσιών μπορεί να προσπελάσουν τα δεδομένα και, ακούσια ή εκούσια, να τα αλλάξουν ή να τα διαγράψουν με αποτέλεσμα να δημιουργηθούν σαβαρές επιχειρηματικές νομικές συνέπειες.

B. Απόδοση

Σύμφωνα με την έρευνα του IDC, η απόδοση είναι το δεύτερο μεγαλύτερο ζήτημα τις υλοποιήσεις cloud. Το cloud πρέπει να παρέχει βελτιωμένη απόδοση όταν ένας χρήστης επιλέξει να το χρησιμοποιήσει έναντι του κλασικού μοντέλου. Η απόδοση γενικά μετράται από τις δυνατότητες των εφαρμογών που τρέχουν στο cloud. Κακή απόδοση μπορεί να προκύψει από από έλλειψη κατάλληλων πόρων όπως χώρος αποθήκευσης, περιορισμένο εύρος ζώνης, χαμηλότερη ταχύτητα CPU, μνήμη, συνδέσεις δικτύου κτλ. Πολλές φορές οι χρήστες προτιμούν να χρησιμοποιήσουν υπηρεσίες από περισσότερους από έναν παρόχους cloud με τα δεδομένα και τις εφαρμογές να έχουν τη δυνατότητα να βρίσκονται σε δημόσια, ιδιωτικά ή κοινοτικά cloud. Για τις εφαρμογές που χρειάζονται να χειριστούν πολλά δεδομένα είναι πιο δύσκολο να παρασχεθούν κατάλληλοι πόροι. Η κακή απόδοση μπορεί να έχει ως αποτέλεσμα τον τερματισμό της παροχής της υπηρεσίας, την απώλεια πελατών κτλ.

C. Αξιοπιστία και διαθεσιμότητα

Το δυνατό σημείο κάθε τεχνολογίας μετράται από τον βαθμό αξιοπιστίας και διαθεσιμότητας. Η διαθεσιμότητα υποδηλώνει το πόσο συχνά είναι διαθέσιμοι οι πόροι χωρίς να συμβεί κάποιο πρόβλημα (π.χ. απώλεια δεδομένων) και πόσο συχνά παρουσιάζουν σφάλματα. Μια από τις σημαντικές οπτικές που δημιουργεί σοβαρά προβλήματα όσον αφορά την αξιοπιστία του cloud είναι ο χρόνος μη λειτουργίας. Ένας τρόπος για να επιτευχθεί η αξιοπιστία είναι η χρησιμοποίηση πλεοναζόντων πόρων. Η διαθεσιμότητα μπορεί να ερμηνευθεί ως η δυνατότητα απόκτησης πόρων οποτεδήποτε χρειάζεται λαμβάνοντας υπόψιν τον χρόνο που χρειάζεται για να παραχωρηθούν. Ανεξάρτητα από το αν οι χρησιμοποιούμενες αρχιτεκτονικές έχουν στοιχεία για υψηλή αξιοπιστία και διαθεσιμότητα, οι cloud υπηρεσίες μπορούν να υποστούν επιθέσεις DOS (Denial of Service), επιβραδύνσεις απόδοσης και φυσικές καταστροφές. Προκειμένου να αντιμετωπιστούν ο φόβος, η αβεβαιότητα, η αμφιβολία και η λάθος πληροφόρηση, πιθανότατα η αξιοπιστία, η διαθεσιμότητα και η ασφάλεια είναι σημαντικά και πρωταρχικά μελήματα για έναν οργανισμό. Γι’αυτό το επίπεδο αξιοπιστίας και διαθεσιμότητας των πόρων του cloud πρέπει να θεωρείται σημαντικό ζήτημα για τον σχεδιασμό ενός οργανισμού για το στήσιμο των υποδομών του στο cloud με σκοπό να παρέχονται αποτελεσματικές υπηρεσίες στους καταναλωτές.

D. Κλιμακωσιμότητα και ελαστικότητα\

Η κλιμακωσιμότητα και η ελαστικότητα είναι τα πιο μοναδικά και αξιοθαύμαστα χαρακτηριστικά του cloud. Αυτά επιτρέπουν στους χρήστες να χρησιμοποιούν πόρους σαν να ήταν απεριόριστοι. Η κλιμακωσιμότητα μπορεί να οριστεί ως η ικανότητα του συστήματος να αποδόσει καλά ακόμα και όταν οι πόροι κλιμακωθούν προς τα πάνω. Η ελαστικότητα από την άλλη, είναι η ικανότητα να κλιμακωθούν οι πόροι και προς τα πάνω και προς τα κάτω αν και

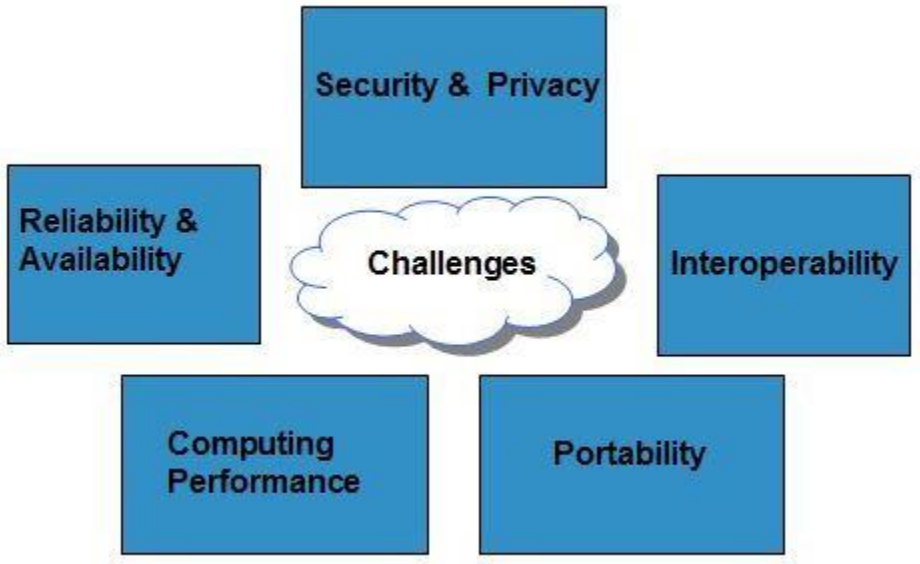
όποτε χρειαστεί. Η ελαστικότητα, επιπλέον, επιτρέπει τη δυναμική ενσωμάτωση και εξαγωγή των φυσικών πόρων στην υποδομή. Η ελαστικότητα επιτρέπει την κλιμακωσιμότητα, το οποίο σημαίνει ότι το σύστημα μπορεί να κλιμακώσει προς τα πάνω ή προς τα κάτω το επίπεδο των υπηρεσιών το οποίο έχει επιλέξει ο χρήστης. Η κλιμακωσιμότητα μπορεί να παρέχεται με δύο τρόπους, οριζόντια ή κατακόρυφα. Οριζόντια κλιμακωσιμότητα είναι η προσθήκη περισσότερων κόμβων στο σύστημα ενώ κατακόρυφη είναι η προσθήκη περισσότερων πόρων σε έναν κόμβο του συστήματος.

E. Διαλειτουργικότητα και φορητότητα

Η διαλειτουργικότητα είναι η δυνατότητα να χρησιμοποιούνται τα ίδια εργαλεία ή εφαρμογές σε κάθε διαφορετική πλατφόρμα υπηρεσίας cloud. Η διαλειτουργικότητα μπορεί να οριστεί σε διάφορα επίπεδα όπως εφαρμογής, υπηρεσίας, διαχείρισης και διαλειτουργικότητας δεδομένων. Οι χρήστες του cloud πρέπει να έχουν την ευελιξία να μετακινούνται μέσα και έξω και να αλλάζουν σε cloud όποτε θέλουν χωρίς να χρειάζεται την περίοδο εξάρτησης απο έναν προμηθευτή. Ένα απο τα εμπόδια υιοθέτησης της διαλειτουργικότητας cloud είναι το ρίσκο εξάρτησης απο έναν προμηθευτή. Τα κύρια προβλήματα για την υλοποίηση του είναι η έλλειψη ανοικτών προτύπων, ανοικτών API, η έλλειψη κανονικών διεπαφών για τους τύπους εικονικών μηχανών και τις υπηρεσίες ανάπτυξης διεπαφών. Η φορητότητα των cloud διαβεβαιώνει ότι μια λύση cloud θα είναι διαθέσιμη να δουλέψει με άλλες πλατφόρμες και εφαρμογές όπως επίσης και με άλλα cloud.

F. Διαχείριση πόρων και δρομολόγηση

Η διαχείριση πόρων μπορεί να αφορά διάφορα επίπεδα όπως το υλικό, το λογισμικό, το επίπεδο εικονικοποίησης, την ασφάλεια και άλλες παραμέτρους που εξαρτώνται από τη διαχείριση και την προμήθεια πόρων. Περιλαμβάνει τη διαχείριση της μνήμης, τον χώρο αποθήκευσης, τους επεξεργαστές, τους πυρήνες, τα νήματα, τα στιγμιότυπα των εικονικών μηχανών, τις συσκευές εισόδου/εξόδου κτλ. Προμήθεια πόρων μπορεί να οριστεί ως η παραχώρηση και διαχείριση πόρων ώστε να παρασχεθούν τα επιθυμητά επίπεδα της υπηρεσίας. Η δρομολόγηση των εργασιών είναι ένας τύπος προμήθειας πόρων όπου η σειρά εκτέλεσης των εργασιών αποφασίζεται με σκοπό να βελτιστοποιηθούν κάποιες παράμετροι όπως ο χρόνος ανάκαμψης, ο χρόνος απόκρισης, ο χρόνος αναμονής, η διεκπεραιωτική ικανότητα και η χρησιμοποίηση των πόρων. Λόγω του ότι το cloud είναι συνδυασμός πολλών υπαρχουσών τεχνολογιών, οι υπάρχουσες στρατηγικές δρομολόγησης μπορούν να εφαρμοστούν. Τα κυριότερα ζητήματα δρομολόγησης στα cloud συστήματα είναι ο διαχωρισμός των εργασιών σε παράλληλες δουλειές, η διασύνδεση δικτύων μεταξύ cloud ή επεξεργαστών, η ανάθεση προτεραιοτήτων σε εργασίες και η επιλογή των επεξεργαστών ή των cloud στα οποία θα παραχωρηθούν οι εργασίες, η ευελιξία των εργασιών, το υποστηριζόμενο επίπεδο παράκαμψης της σειράς εκτέλεσης, τα χαρακτηριστικά του φόρτου εργασίας, η παραχώρηση μνήμης, η παρακολούθηση της εκτέλεσης των επιμέρους δουλειών μιας εργασίας, οι απαιτήσεις της παραχώρησης πόρων, η τοπολογία, η φύση της εργασίας κτλ Η δρομολόγηση εργασιών είναι μια κρίσιμη διαδικασία που πρέπει να αποφασιστεί πολύ προσεκτικά καθώς η λάθος επιλογή στρατηγικής δρομολόγησης μπορεί να οδηγήσει σε καταστροφικά αποτελέσματα όσον αφορά την απόδοση, στην κατασπατάληση πόρων και στην αποτυχία ικανοποίησης των προτύπων Ποιότητας των Υπηρεσιών (Quality of Service).



Εικόνα 1.9 – Προκλήσεις του Cloud Computing

G. Κατανάλωση ενέργειας

Τα datacenters για τα cloud στεγάζουν χιλιάδες εξυπηρετητές και παρέχουν υποδομές ψύξης για να απομακρύνουν τη θερμότητα που παράγεται από αυτούς. Οι εξυπηρετητές και η προαναφερθείσα υποδομή καταναλώνουν ενα τεράστιο ποσό ενέργειας και παράγουν αέρια του θερμοκηπίου (GHGs). Επιπρόσθετα, τα datacenters για τα cloud, τα οποία αποτελούν εγγενές τμήμα της υποδομής των cloud, είναι επίσης πολύ ακριβά όσον αφορά τα λειτουργικά τους κόστη και καταναλώνουν πολλή ενέργεια σε πολύ μεγάλη κλίμακα. Για παράδειγμα, η κατανάλωση ενέργειας των datacenter της Google είναι ισοδύναμη με την πόλη του San Francisco. Από τη στιγμή που ο σκοπός είναι η ανάπτυξη εφαρμογών και διευκολύνσεων για την ανθρώπινη ευημερία, απαιτείται η σχεδίαση τέτοιου υλικού, λογισμικού, πολιτικών δρομολόγησης, δικτύων, και άλλων πρωτοκόλλων τα οποία να καταναλώνουν ενέργεια με πιο φιλικό προς το περιβάλλον και βέλτιστο τρόπο. Ο στόχος είναι, εκτός από τη μείωση κατνάλωσης ενέργειας από τα datacenters και έτσι και του αντίστοιχου κόστους, αλλά και η διατήρηση των περιβαλλοντολογικών προτύπων που είναι αναγκαία για την ανθρώπινη ευημερία.

H. Εικονικοποίηση

Εικονικοποίηση είναι η δημιουργία μιας εικονικής εκδοχής μιας συσκευής αποθήκευσης, ενός λειτουργικού συστήματος, ενός εξυπηρετητή ή ενός δικτύου. Η εικονικοποίηση διαιρεί τους πόρους σε πολλαπλά εικονικά περιβάλλοντα εκτέλεσης και κρύβει τα φυσικά χαρακτηριστικά των υπολογιστικών πόρων για να απλοποιήσει τον τρόπο με τον οποίο άλλα συστήματα, εφαρμογές, ή και χρήστες αλληλεπιδρούν με αυτούς τους πόρους. Η εικονικοποίηση χρησιμοποιείται με δύο τρόπους: τύπου 1 επόπτες / εικονικοποίηση υλικού και τύπου 2 επόπτες / εικονικοποίηση λειτουργικού συστήματος. Η εικονικοποίηση είναι μια από τις κύριες

τεχνολογίες που καθιστούν εφικτή την υλοποίηση του cloud. Συνήθως, η υλοποίησή της επιτρέπει στους καταναλωτές να μεταφέρουν τα δεδομένα και τις εφαρμογές τους σε μια απομακρυσμένη τοποθεσία με κάποιο αντίκτυπο στην απόδοση. Υπάρχουν πολυάριθμα οφέλη από την εικονικοποίηση τα οποία δεν θα μπορούσαν να επιτευχθούν αλλιώς. Κάποια από αυτά περιλαμβάνουν την ελαστικότητα, την κλιμακωσιμότητα, το μειωμένο κόστος, την ανεξαρτησία από την τοποθεσία και από την υποδομή, την παραμετροποίηση, την απλοποιημένη διεπαφή πρόσβασης κτλ. Παρά τα θετικά που προσφέρει, θέτει και και αρκετές προκλήσεις. Έχει πολλά κρίσιμα ζητήματα που πρέπει να αντιμετωπιστούν όπως η εξάπλωση των εικονικών μηχανών, ο χαρακτηρισμός του φόρτου εργασίας των εικονικών μηχανών, ζητήματα ασφαλείας σε επικοινωνία μεταξύ cloud βασισμένη σε επόπτες, η ασφάλεια ζωντανής μετακινησης κτλ.

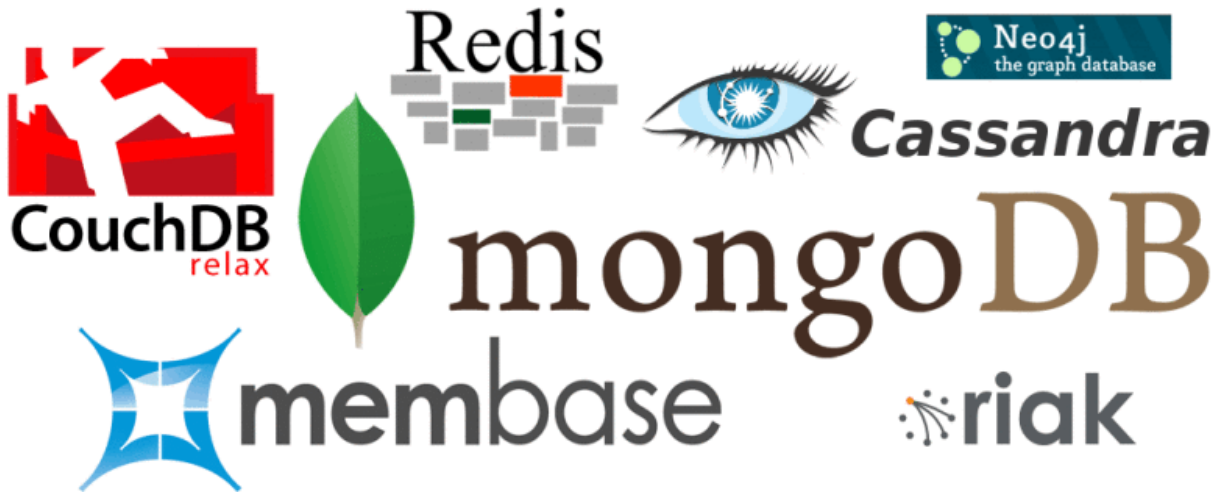
I. Κόστος εύρους ζώνης

Τα υψηλής ταχύτητας κανάλια επικοινωνίας χρησιμεύουν ως η ραχοκοκαλιά του cloud. Με τη χρήση του cloud, η επιχειρήσεις αποκτούν τη δυνατότητα να αποταμιεύσουν χρήματα που θα ξόδευαν σε υλικό και/ή λογισμικό αλλά και πάλι απαιτείται να ξοδεύονται πολλά στο εύρος ζώνης. Είναι σχεδόν αδύνατο, κάποιος, να επωφεληθεί πλήρως απο τις υπηρεσίες cloud χωρίς υψηλής ταχύτητας κανάλια επικοινωνίας. Η μετάβαση σε cloud επίσης αφαιρεί το προκαταβολικό κόστος, ενώ αυξάνει το κόστος της επικοινωνίας δεδομένων στο δίκτυο π.χ. το κόστος που περιλαμβάνεται στη μεταφορά των δεδομένων προς και απο τα ιδιωτικά και άλλα cloud. Αυτό το πρόβλημα είναι εμφανές αν η εφαρμογή του καταναλωτή χειρίζεται πολλά δεδομένα και τα δεδομένα του καταναλωτή είναι μοιρασμένα σε ένα μεγάλο αριθμό απο cloud (ιδιωτικά/δημόσια/κοινωτικά). Το cloud παρέχει μικρότερο κόστος για δουλειές που χρησιμοποιούν κυρίως τον επεξεργαστή απ'ότι για δουλείες που αφορούν κυρίως δεδομένα. Επομένως, οι εφαρμογες που χειρίζονται πολλά δεδομένα αποδίδουν καλύτερα αν τρέχουν σε ιδιωτικό cloud παρά σε δημόσιο ή υβριδικό.

2
NoSQL Βάσεις Δεδομένων

2.1 Εισαγωγή

Μια NoSQL ή Not Only SQL βάση δεδομένων παρέχει ένα μηχανισμό προς αποθήκευση και ανάκτηση δεδομένων που είναι διαμορφωμένα με άλλο τρόπο απο αυτό της μορφής πίνακα που χρησιμοποιείται στις σχεσιακές βάσεις δεδομένων. Κίνητρα για αυτήν την προσέγγιση περιλαμβάνουν την απλότητα του σχεδίου, την οριζόντια κλιμάκωση και τον καλύτερο έλεγχο στην διαθεσιμότητα. Η δομή δεδομένων (π.χ. κλειδιού-τιμής, γραφήματος ή εγγράφου) διαφέρει από τις δομές στα RDBMS, και γι'αυτό κάποιες λειτουργίες είναι πιο γρήγορες σε NoSQL και κάποιες σε RDBMS. Υπάρχουν διαφορές ωστόσο, και η συγκεκριμένη καταλληλότητα μιας δεδομένης NoSQL βάσης δεδομένων εξαρτάται απο το πρόβλημα που καλείται να επιλύσει(π.χ. η λύση χρησιμοποιεί γράφους αλγόριθμου;).



Εικόνα 2.1 – Γνωστές NoSQL βάσεις δεδομένων

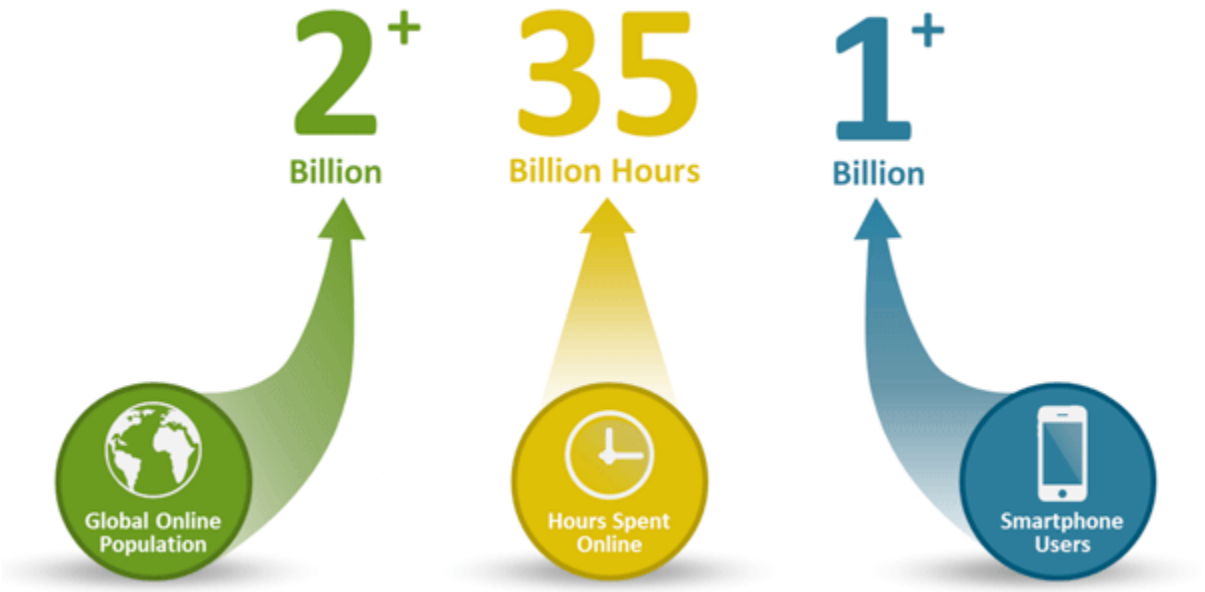
Οι βάσεις δεδομένων NoSQL είναι αυξανόμενα χρησιμοποιούμενες σε εφαρμογές big data και πραγματικού χρόνου. Τα NoSQL συστήματα επίσης αποκαλούνται "Not only SQL" για να δώσουν έμφαση στο οτι μπορεί επίσης να υποστηρίζουν γλώσσες ερωτημάτων σαν την SQL. Πολλές NoSQL αποθηκεύσεις θέτουν σε κίνδυνο τη συνοχή (με την έννοια του θεωρήματος CAP) για χάρη της διαθεσιμότητας και της τμηματικής ανεκτικότητας. Εμπόδια για την ευρύτερη υιοθέτηση των NoSQL αποθηκεύσεων αποτελούν η χρήση χαμηλού επιπέδου γλωσσών ερωτημάτων, η έλλειψη τυποποιημένων διεπαφών και οι τεράστιες επενδύσεις στην υπάρχουσα SQL. Οι περισσότερες NoSQL αποθηκεύσεις στερούνται πραγματικών ACID συναλλαγών, παρόλο που μερικά πρόσφατα συστήματα, όπως το FairCom c-treeACE, το Google Spanner και το FoundationDB, τις έχουν κάνει βασικές στο σχεδιασμό τους.

2.2 Γιατί NoSQL;

Υπάρχουν τρείς τάσεις που διαταράσουν το καθεστώς στις βάσεις δεδομένων. Οι αλληλεπιδραστικές εφαρμογές έχουν αλλάξει δραματικά τα τελευταία 15 χρόνια και το ίδιο ισχύει και για τις ανάγκες διαχείρισης δεδομένων αυτών των εφαρμογών. Σήμερα, τρείς αλληλένδετες μαζικές τάσεις - Big Data, Big Users και Cloud Computing - οδηγούν στην υιοθέτηση της τεχνολογίας NoSQL. Η NoSQL θεωρείται όλο και περισσότερο μια βιώσιμη εναλλακτική των σχεσιακών βάσεων δεδομένων, ιδιαίτερα καθώς περισσότεροι οργανισμοί αναγνωρίζουν ότι καλύτερη κλιμάκωση επιτυγχάνεται σε συστάδες τυπικών εξυπηρετητών του εμπορίου και ότι ένα μοντέλο δεδομένων χωρίς σχήμα (schema-less) είναι συνήθως καλύτερο για την ποικιλία και των τύπο των δεδομένων που καταγράφονται και επεξεργάζονται σήμερα.

- **Big Users**

Όχι πολύ καιρό πριν, 1,000 καθημερινοί χρήστες ήταν πολλοί και 10,000 ήταν ακραία περίπτωση. Σήμερα, με την ανάπτυξη της παγκόσμιας χρήσης του διαδικτύου, τον αυξημένο αριθμό ωρών που ξοδεύουν οι χρήστες όντας συνδεδεμένοι στο διαδίκτυο και την αυξανόμενη δημοτικότητα των smartphones και των tablets, δεν είναι ασυνήθιστο οι εφαρμογές να έχουν ένα εκατομμύριο χρήστες τη μέρα.



Εικόνα 2.2 - Στατιστικά χρήσης εφαρμογών σήμερα

Η υποστήριξη μεγάλου αριθμού ταυτόχρονων χρηστών είναι σημαντική αλλά, επειδή οι απαιτήσεις χρήσης της εφαρμογής είναι δύσκολο να προβλεφθούν, είναι εξίσου σημαντική η δυναμική υποστήριξη της ταχείας αύξησης (ή μείωσης) του αριθμού των ταυτόχρονων χρηστών:

- Μια καινούργια εφαρμογή μπορεί να γίνει ευρέως γνωστή, με αποτέλεσμα να φτάσει από μηδέν σε ένα εκατομμύριο χρήστες κυριολεκτικά εν μια νυκτί.

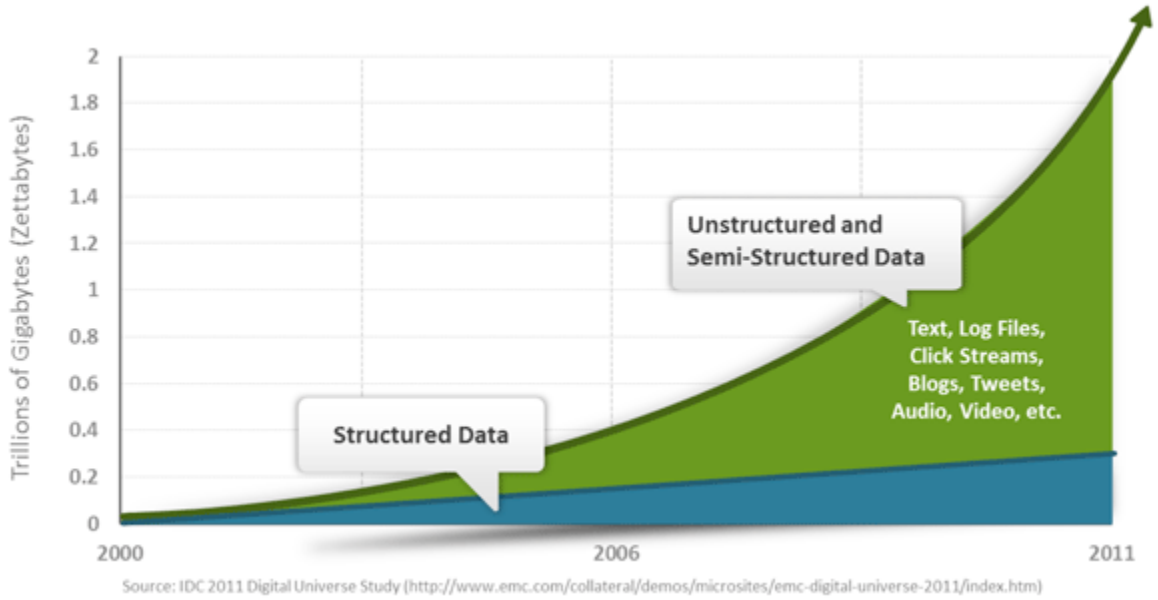
- Κάποιοι χρήστες χρησιμοποιούν μια εφαρμογή συχνά ενώ άλλοι μόνο μερικές φορές και μετά ποτέ ξανα

- Εποχιακές διακυμάνσεις, όπως τα Χριστούγεννα ή τη μέρα του αγίου Βαλεντίνου, δημιουργούν απότομες αυξήσεις για μικρές περιόδους.

Ο μεγάλος αριθμός χρηστών συνδυασμένος με τη δυναμική φύση των προτύπων χρήσης οδηγεί στην ανάγκη για πιο εύκολα κλιμακώσιμες τεχνολογίες βάσεων δεδομένων. Με τις τεχνολογίες σχεσιακών βάσεων δεδομένων, πολλοί προγραμματιστές εφαρμογών δυσκολεύονται (ή δεν το καταφέρνουν καθόλου) να πετύχουν τη δυναμική κλιμακωσιμότητα και το επίπεδο κλιμάκωσης που χρειάζονται διατηρώντας παράλληλα την απόδοση που απαιτούν οι χρήστες και γι'αυτό στρέφονται στις NoSQL τεχνολογίες.

- **Big Data**

Η καταγραφή δεδομένων γίνεται όλο και ευκολότερη μέσω τρίτων όπως οι Facebook, D&B και άλλοι. Προσωπικές πληροφορίες χρηστών, δεδομένα τοποθεσίας, κοινωνικά γραφήματα, περιεχόμενο παραγόμενο από χρήστες, δεδομένα καταγραφής μηχανημάτων και δεδομένα παραγόμενα από αισθητήρες είναι μερικά παραδείγματα του συνεχώς αυξανόμενου πεδίου δεδομένων που καταγράφονται. Είναι αναμενόμενο, λοιπόν, οι προγραμματιστές να θέλουν να εμπλουτίσουν τις υπάρχουσες εφαρμογές και να δημιουργήσουν καινούργιες που να το εκμεταλλεύονται αυτό. Και η χρήση των δεδομένων αλλάζει ραγδαία τη φύση των τηλεπικοινωνιών, των αγορών, της διαφήμισης και της διασκέδασης. Οι εφαρμογές που δεν αφουγκράζονται αυτές τις αλλαγές μένουν πίσω



Εικόνα 2.3 - Αύξηση δομημένων και ημι-δομημένων δεδομένων

Οι προγραμματιστές θέλουν μια πολύ ευέλικτη βάση δεδομένων, η οποία θα πρέπει να υποστηρίζει εύκολα νέους τύπους δεδομένων και μια διαταράσσεται από τις αλλαγές στη δομή

των περιεχομένων από τρίτους παρόχους δεδομένων. Πολλά από τα καινούργια δεδομένα δεν έχουν δομή και οι προγραμματιστές χρειάζονται επίσης μια βάση δεδομένων ικανή να τα αποθηκεύσει αποδοτικά. Δυστυχώς, οι αυστηρά ορισμένες, με σχήμα προσεγγίσεις που χρησιμοποιούνται από τις σχεσιακές βάσεις δεδομένων καθιστούν αδύνατη την γρήγορη ενσωμάτωση νέων τύπων δεδομένων και δεν ταιριάζουν σε δεδομένα χωρίς δομή. Η NoSQL παρέχει ένα μοντέλο δεδομένων που ταιριάζει καλύτερα σε αυτές τις ανάγκες.

- **Cloud Computing**

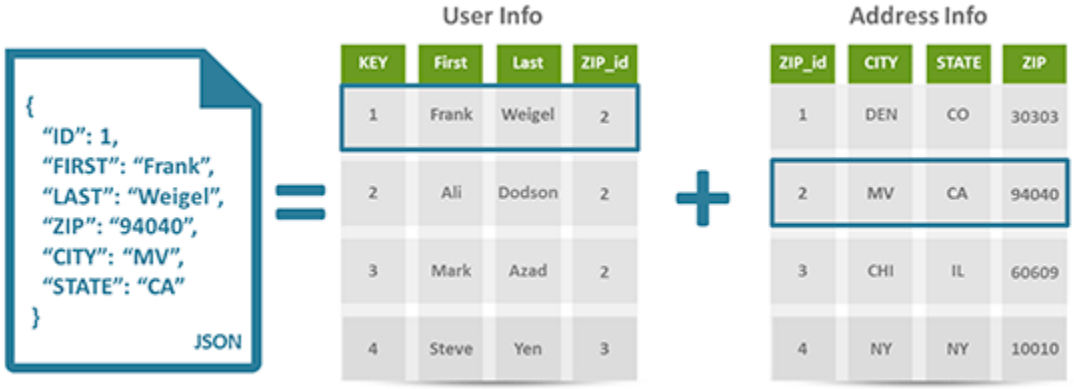
Σήμερα, οι περισσότερες νέες εφαρμογές χρησιμοποιούν μια τριών επιπέδων αρχιτεκτονική διαδικτύου, τρέχουν σε δημόσια ή ιδιωτικά cloud και υποστηρίζουν μεγάλο αριθμό χρηστών. Σε αυτή την αρχιτεκτονική, οι εφαρμογές προσπελαύνονται μέσω ενός περιηγητή διαδικτύου ή εφαρμογής κινητής συσκευής που έχει σύνδεση στο διαδίκτυο. Στο cloud, ένας σταθεροποιητής φόρτου εργασίας κατευθύνει την εισερχόμενη κίνηση σε ένα επίπεδο εξυπηρετητών, με δυνατότητα κλιμάκωσης, που επεξεργάζονται τη λογική της εφαρμογής. Όσον αφορά την κλιμάκωση, για κάθε 10,000 νέους ταυτόχρονους χρήστες, απλά προστίθεται ένας εξυπηρετητής του εμπορίου στην εφαρμογή για να απορροφήσει τον φόρτο εργασίας.
Στο επίπεδο της βάσης δεδομένων, οι σχεσιακές βάσεις ήταν αρχικά η δημοφιλής επιλογή. Παρόλα αυτά η χρήση τους γινόταν αυξανόμενα προβληματική λόγω του ότι είναι μια συγκεντρωτική, διαμοιραζόμενη τεχνολογία η οποία κλιμακώνει κατακόρυφα αντί για οριζόντια. Αυτό τις καθιστά κακή λύση όσον αφορά εφαρμογές που απαιτούν γρήγορη και δυναμική κλιμακωσιμότητα. Οι NoSQL βάσεις δεδομένων δημιουργήθηκαν εξαρχής ως κατανεμημένη και οριζόντια κλιμακούμενη τεχνολογία και για αυτό ταιριάζουν καλύτερα με την υψηλά κατανεμημένη φύση της τριών επιπέδων αρχιτεκτονικής του διαδικτύου.

Γιατί οι προγραμματιστές σκέφτονται την εναλλακτική της NoSQL;

- **Πιο ευέλικτο μοντέλο δεδομένων**

Τα μοντέλα δεδομένων των σχεσιακών και των NoSQL βάσεων δεδομένων είναι πολύ διαφορετικά μεταξύ τους. Το σχεσιακό χωρίζει τα δεδομένα σε πολλούς αλληλένδετους πίνακες οι οποίοι αποτελούνται από γραμμές και στήλες. Οι πίνακες αναφέρονται ο ένας στον άλλο μέσω δευτερευόντων κλειδιών τα οποία αποθηκεύονται επίσης σε στήλες. Κατά την αναζήτηση δεδομένων, οι επιθυμητές πληροφορίες πρέπει να συλλεχθούν από πολλούς πίνακες (συχνά εκατοντάδες στις σημερινές εφαρμογές επιχειρήσεων) και να συνδυαστούν πριν να μπορούν να δοθούν στην εφαρμογή. Παρομοίως, κατά την εγγραφή δεδομένων, αυτή πρέπει να συντονιστεί και να γίνει σε πολλούς πίνακες.
Οι NoSQL βάσεις δεδομένων έχουν ένα πολύ διαφορετικό μοντέλο. Για παράδειγμα, μια document-oriented NoSQL βάση δεδομένων παίρνει τα προς αποθήκευση δεδομένα και τα συγκεντρώνει σε documents χρησιμοποιώντας τη μορφή JSON. Κάθε JSON document μπορεί να θεωρηθεί ως ένα αντικείμενο που θα χρησιμοποιηθεί από την εφαρμογή. Ένα JSON document μπορεί, για παράδειγμα, να πάρει όλα τα δεδομένα που είναι αποθηκευμένα σε μια γραμμή που εκτείνεται σε 20 πίνακες μιας σχεσιακής βάσης δεδομένων και να τα αθροίσει σε ένα μόνο αντικείμενο/document. Η άθροιση των παραπάνω μπορεί να οδηγήσει σε ύπαρξη διπλότυπων, αλλά αφού η αποθήκευση δεν είναι πλέον απαγορευτική όσον αφορά το κόστος, η

ευελιξία του μοντέλου δεδομένων που προκύπτει, η ευκολία της αποδοτικής κατανομής των documents που προκύπτουν και οι βελτιώσεις απόδοσης των εγγραφών και αναγνώσεων το καθιστούν μια εύκολη ανταλλαγή για τις εφαρμογές διαδικτύου.



Εικόνα 2.4 - Document-oriented βάση δεδομένων

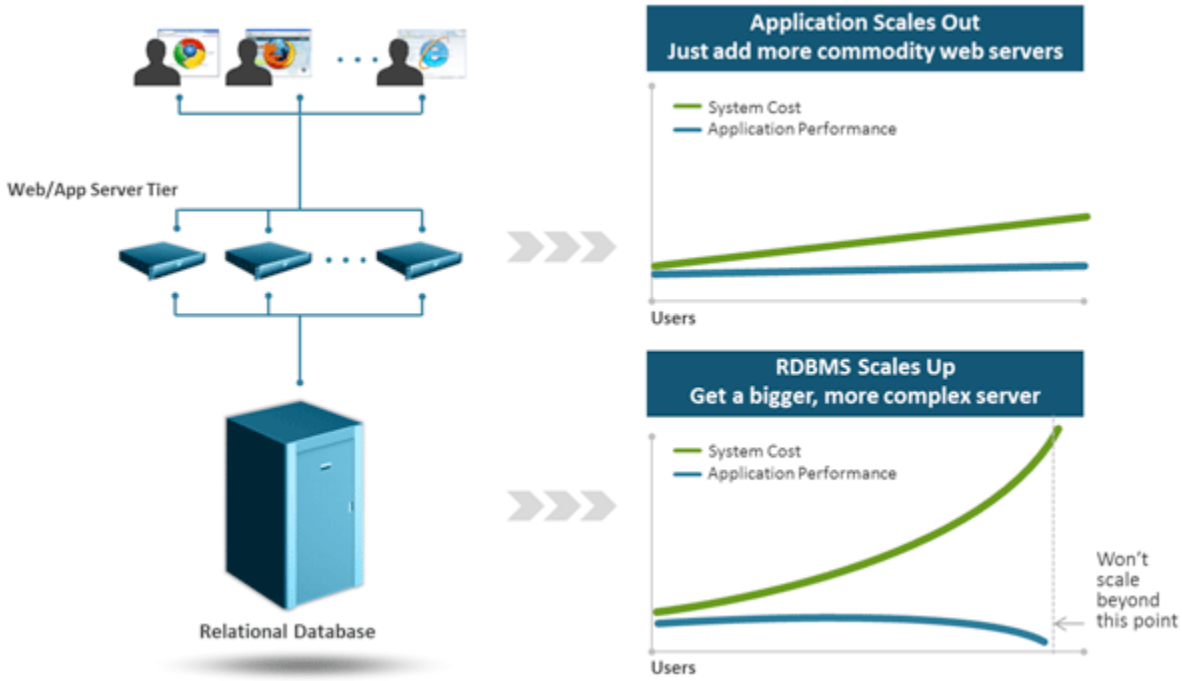
Μια ακόμα μεγάλη διαφορά είναι ότι οι σχεσιακές τεχνολογίες έχουν αυστηρό σχήμα ενώ τα μοντέλα NoSQL δεν έχουν σχήμα. Η σχεσιακή τεχνολογία απαιτεί αυστηρό ορισμό ενός σχήματος πριν να αποθηκευτούν δεδομένα σε μια βάση δεδομένων. Η αλλαγή σχηματος μετά την εισαγωγή δεδομένων είναι δύσκολη και αποφευκταία - το ακριβώς αντίθετο από τη συμπεριφορά που χρειάζεται στην εποχή των Big Data, όπου οι προγραμματιστές χρειάζεται συνεχώς - και ταχέως - να ενσωματώσουν νέους τύπους δεδομένων για να εμπλουτίσουν τις εφαρμογές τους.
Συγκριτικά, οι βάσεις δεδομένων που αποθηκεύουν documents δεν έχουν σχήμα, επιτρέποντας την ελεύθερη προσθήκη πεδίων στα JSON documents. χωρίς να χρειάζεται πρώτα να οριστούν οι αλλαγές. Η μορφή των δεδομένων που εισάγονται μπορεί να αλλάξει οποιαδήποτε στιγμή, χωρίς να διαταραχθεί η εφαρμογή.

- **Πλεονεκτήματα στην απόδοση και την κλιμακωσιμότητα**

Για να αντιμετωπίσουν την αύξηση σε ταυτόχρονους χρήστες (Big Users) και σε ποσότητα δεδομένων (Big Data), οι εφαρμογές και οι υποκείμενες βάσεις δεδομένων χρειάζεται να κλιμακώσουν χρησιμοποιώντας μια από τις δύο επιλογές: κατακόρυφη ή οριζόντια κλιμάκωση. Κατακόρυφη σημαίνει να αυξάνονται συνεχώς οι δυνατότητες των υπάρχοντων εξυπηρετητών ενώ οριζόντια σημαίνει να προστίθενται επιπλέον τυπικοί εξυπηρετητές του εμπορίου στην υποδομή.

➢ **Κατακόρυφη κλιμάκωση με σχεσιακή τεχνολογία: περιορισμοί στο επίπεδο της βάσης δεδομένων**

Στο επίπεδο της εφαρμογής της αρχιτεκτονικής διαδικτύου τριών επιπέδων, η οριζόντια κλιμάκωση ήταν η προεπιλεγμένη λύση για πολλά χρόνια και λειτουργούσε πολύ καλά. Καθώς όλο και περισσότεροι άνθρωποι χρησιμοποιούν μια εφαρμογή, περισσότεροι εξυπηρετητές του εμπορίου προστίθενται στο επίπεδο της εφαρμογής, η απόδοση διατηρείται με την κατανομή του φορτου εργασίας σε ένα αυξημένο αριθμό εξυπηρετητών και το κόστος κλιμακώνεται γραμμικά σε σχέση με τον αριθμό των χρηστών.

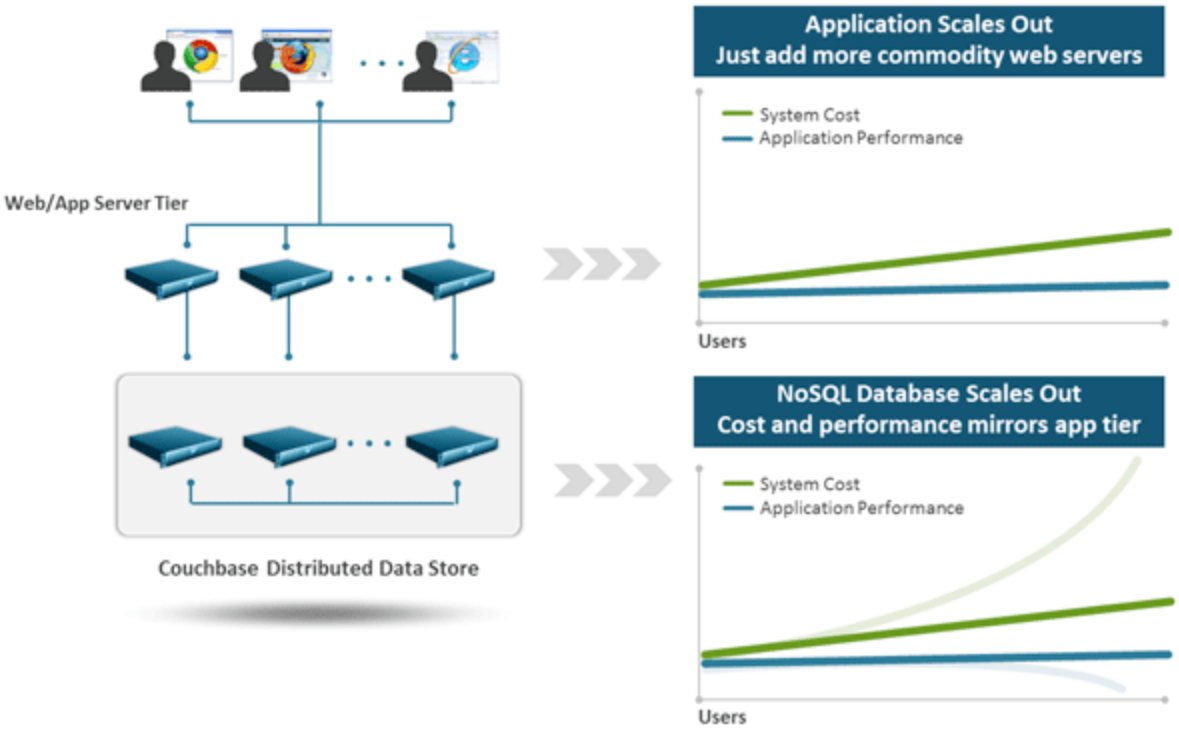


Εικόνα 2.5 - Κλιμακωσιμότητα RDBMS

Πριν από τις NoSQL βάσεις δεδομένων, η προεπιλεγμένη προσέγγιση ως προς την κλιμάκωση στο επίπεδο της βάσης δεδομένων ήταν η κατακόρυφη κλιμάκωση. Αυτό υπαγορευόταν από τη θεμελιωδώς συγκεντρωτική αρχιτεκτονική της σχεσιακής τεχνολογίας βάσεων δεδομένων. Για την υποστήριξη περισσότερων ταυτόχρονων χρηστών και/ή την αποθήκευση περισσότερων δεδομένων, χρειάζονται όλο και υψηλότερων δυνατοτήτων εξυπηρετητές με περισσότερους επεξεργαστές, περισσότερη μνήμη και περισσότερο αποθηκευτικό χώρο για όλους τους πίνακες. Οι μεγάλοι εξυπηρετητές τείνουν να είναι πολύ πολύπλοκοι, ιδιοταγείς, και δυσανάλογα ακριβοί εν αντιθέσει με αυτούς του εμπορίου.

- **Οριζόντια κλιμάκωση με NoSQL τεχνολογία στο επίπεδο της βάσης δεδομένων**

Οι NoSQL βάσεις δεδομένων αναπτύχθηκαν εξ αρχής ώστε να είναι κατανεμημένες και να ευνοούν την οριζόντια κλιμάκωση. Χρησιμοποιούν ένα σύμπλεγμα τυπικών, φυσικών ή εικονικών εξυπηρετητών για να αποθηκεύουν δεδομένα και να υποστηρίζουν τις λειτουργίες της βάσης δεδομένων. Για κλιμάκωση, επιπλέον εξυπηρετητές προστίθενται στο σύμπλεγμα και τα δεδομένα και οι λειτουργίες της βάσης δεδομένων μοιράζονται στο μεγαλύτερο σύμπλεγμα. Επίσης, επειδή οι εξυπηρετητές του εμπορίου αναμένεται να παρουσιάσουν σφάλματα ανά κάποια χρονικά διαστήματα, οι βάσεις δεδομένων NoSQL έχουν δημιουργηθεί έτσι ώστε να είναι ανεκτικές και να ανακάμπτουν από τέτοια προβλήματα και αυτό τις καθιστά πολύ ανθεκτικές.
Οι NoSQL βάσεις δεδομένων παρέχουν μια πολύ ευκολότερη και γραμμική προσέγγιση στην κλιμάκωση των βάσεων δεδομένων. Για κάθε 10,000 νέους χρήστες που αρχίζουν να χρησιμοποιούν μια εφαρμογή, τότε πρέπει απλά να προστεθεί μια ένας νέος εξυπηρετητής βάσης δεδομένων στο σύμπλεγμα. Με αυτόν τον τρόπο η εφαρμογή δεν χρειάζεται τροποποίηση καθώς αυτή βλέπει συνέχεια μόνο μια (κατανεμημένη) βάση δεδομένων.



Εικόνα 2.6 - Κλιμακωσιμότητα NoSQL

Μια κατανεμημένη προσέγγιση οριζόντιας κλιμάκωσης επίσης συνήθως καταλήγει να είναι φθηνότερη από την εναλλακτική κατακόρυφης κλιμάκωσης. Αυτό είναι συνέπεια του ότι οι μεγάλοι, πολύπλοκοι και ανεκτικοί στα σφάλματα εξυπηρετητές έχουν υψηλά κόστη σχεδιασμού, κατασκευής και συντήρησης. Τα κόστη των αδειών των εμπορικών σχεσιακών βάσεων δεδομένων μπορεί επίσης να είναι απαγορευτικό λόγω του ότι κοστολογούνται θεωρώντας ότι θα χρησιμοποιηθεί ένας μόνο εξυπηρετητής. Από την άλλη, οι βάσεις δεδομένων NoSQL είναι γενικά ανοιχτού κώδικα, κοστολογούνται με βάση το ότι θα χρησιμοποιηθούν σε ένα σύμπλεγμα εξυπηρετητών και είναι σχετικά οικονομικές.
Αν και γενικά οι υλοποιήσεις διαφέρουν, οι NoSQL βάσεις δεδομένων έχουν κάποια κοινά χαρακτηριστικά όσον αφορά την κλιμάκωση και την απόδοση:

- Auto-sharding – Μια NoSQL βάση δεδομένων εξαπλώνει αυτόματα τα δεδομένα στους εξυπηρετητές, χωρίς να απαιτεί από τις εφαρμογές να συμμετάσχουν. Εξυπηρετητές μπορούν να προστίθενται και να αφαιρούνται από το στρώμα των δεδομένων χωρίς να σταματάει να λειτουργεί η εφαρμογή με την αυτόματη εξάπλωση των δεδομένων στους εξυπηρετητές. Οι περισσότερες NoSQL βάσεις δεδομένων επίσης υποστηρίζουν αντιγραφή δεδομένων, αποθηκεύοντας πολλαπλά αντίγραφα σε όλο το σύμπλεγμα, ακόμα και σε άλλα datacenters, για να διασφαλίσει υψηλή διαθεσιμότητα και να υποστηρίξει ανάκαμψη από κάποια καταστροφή. Ένα σύστημα NoSQL που υφίσταται σωστή διαχείριση δεν θα χρειαστεί ποτέ να σταματήσει να λειτουργεί, για κανένα λόγο, υποστηρίζοντας 24x365 συνεχή λειτουργία των εφαρμογών.

- Distributed query support – Η χρήση του “Sharding” σε μια σχεσιακή βάση δεδομένων μπορεί να μειώσει, ή να εξαλείψει σε κάποιες περιπτώσεις, την δυνατότητα εκτέλεσης πολύπλοκων ερωτήσεων δεδομένων. Τα NoSQL συστήματα βάσεων δεδομένων διατηρούν πλήρη εκφραστική δύναμη των ερωτημάτων τους ακόμα και όταν είναι κατανεμημένες σε εκατοντάδες εξυπηρετητών.

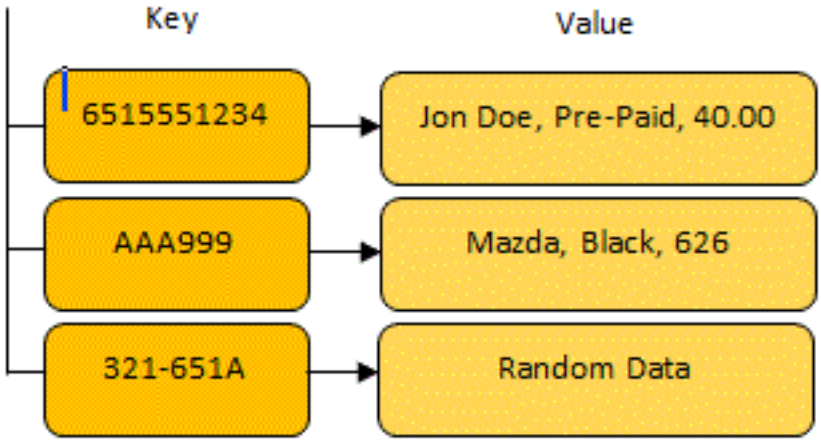
- Integrated caching – Για τη μείωση της καθυστέρησης απόκρισης και την αύξηση της συνεχούς ροής δεδομένων, οι εξελιγμένες τεχνολογίες βάσεων δεδομένων NoSQL, κρατούν δεδομένα στη μνήμη του συστήματος. Αυτή η συμπεριφορά δεν γίνεται αντιληπτή από τον προγραμματιστή της εφαρμογής ενώ στη σχεσιακή τεχνολογία η αντίστοιχη διαδικασία χρειάζεται ξεχωριστή υποδομή και πρέπει να οργανωθεί από τον προγραμματιστή.

2.3 Κατηγορίες

- Key-Value store

Τα key-value stores είναι ένα είδος NoSQL βάσεων δεδομένων που δεν έχουν σχήμα και επίσης οι τιμές αποθηκεύονται ως κλειδί, δηλαδή σε μια στήλη θα υπάρχει ένα κλειδί “Όνομα” και η τιμή θα είναι “Νίκος” και στη δεύτερη στήλη δεν είναι απαραίτητο ότι θα υπάρχει τιμή για το πεδίο “Όνομα” και θα μπορούσε να είναι αποθηκευμένο άλλο είδος δεδομένων στην ίδια στήλη σε διαφορετική γραμμή. Επίσης θα μπορούσαν να υάρχουν περισσότερες ή λιγότερες στήλες σε μια γραμμή από μια άλλη. Αυτό είναι το πιο σύνηθες είδος NoSQL βάσης δεδομένων που κυκλοφορεί στην αγορά ενώ υπάρχουν και άλλα είδη που βασίζονται πάνω στις αρχές των key-value stores και προσθέτουν επιπλέον στοιχεία.
Αυτό το είδος είναι μια πολύ απλή κατασκευή που βασίζεται στο Amazon Dynamo DB. Τα δεδομένα ευρετηριοποιούνται και αναζητώνται με βάση το κλειδί. Τα key-value stores παρέχουν συνεπές hashing έτσι ώστε να μπορούν να κλιμακωθούν σταδιακά παράλληλα με τα δεδομένα. Ανταλάσσουν δομές κόμβων διαμέσου ενός gossip πρωτοκόλλου για να παραμείνουν όλοι οι κόμβοι συγχρονισμένοι. Σε περίπτωση κλιμάκωσης πολύ μεγάλων συνόλων δεδομένων χαμηλής πολυπλοκότητας, τα key-value stores είναι η καλύτερη επιλογή.

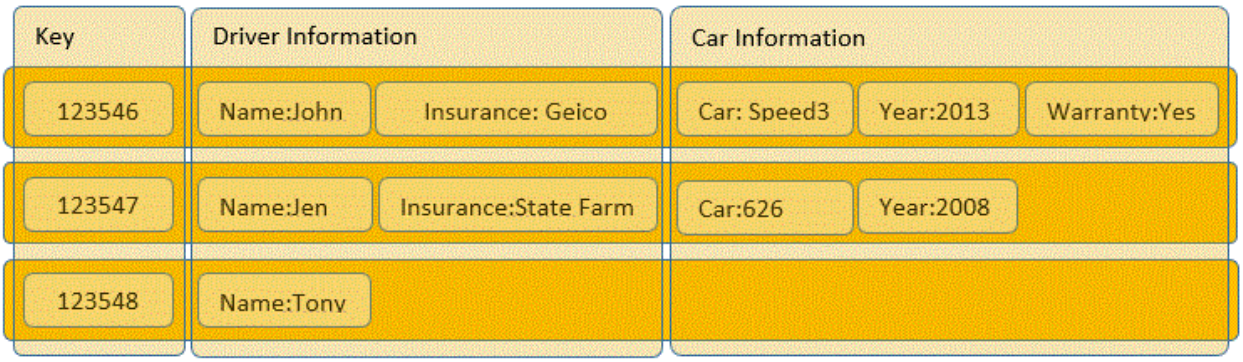
Παραδείγματα: Tokyo Cabinet/Tyrant, Redis, Voldemort, Oracle BDB, Amazon SimpleDB, Riak
Δυνατά σημεία: Γρήγορες αναζητήσεις
Αδύνατα σημεία: Τα αποθηκευμένα δεδομένα δεν έχουν σχήμα



Εικόνα 2.7 - Key-Value store

- Column Family store

Αυτό το είδος δημιουργήθηκε για την αποθήκευση και την επεξεργασία πολύ μεγάλων ποσοτήτων δεδομένων, κατανεμημένων σε πολλά μηχανήματα. Συνεχίζουν να υπάρχουν κλειδιά αλλά δείχνουν σε πολλαπλές στήλες. Οι στήλες οργανώνονται σε οικογένειες.
Τα column family stores βασίζονται στην υλοποίηση BigTable της Google. Μπορεί να μοιάζουν επιφανειακά παρεμφερή με τις σχεσιακές βάσεις δεδομένων αλλά στη βάση της υποδομής υπάρχουν πολλές διαφορές. Μια τέτοια βάση δεδομένων μπορεί να έχει διαφορετικές στήλες σε κάθε γραμμή και γι'αυτό δεν είναι σχεσιακή και δεν έχει αυτό που θεωρείται πίνακας σε ένα RDBMS. Οι μόνες βασικές έννοιες είναι οι στήλες, οι οικογένειες στηλών και οι υπερ-στήλες. Οι οικογένειες στηλών καθορίζουν το πως θα δομηθούν τα δεδομένα στον δίσκο.
Μια στήλη είναι από μόνη της ένα ζευγάρι κλειδιού-τιμής που υπάρχει σε μια οικογένεια στηλών. Μια υπερ-στήλη είναι σαν ένας κατάλογος ή συλλογή από άλλες στήλες, εκτός από υπερ-στήλες.



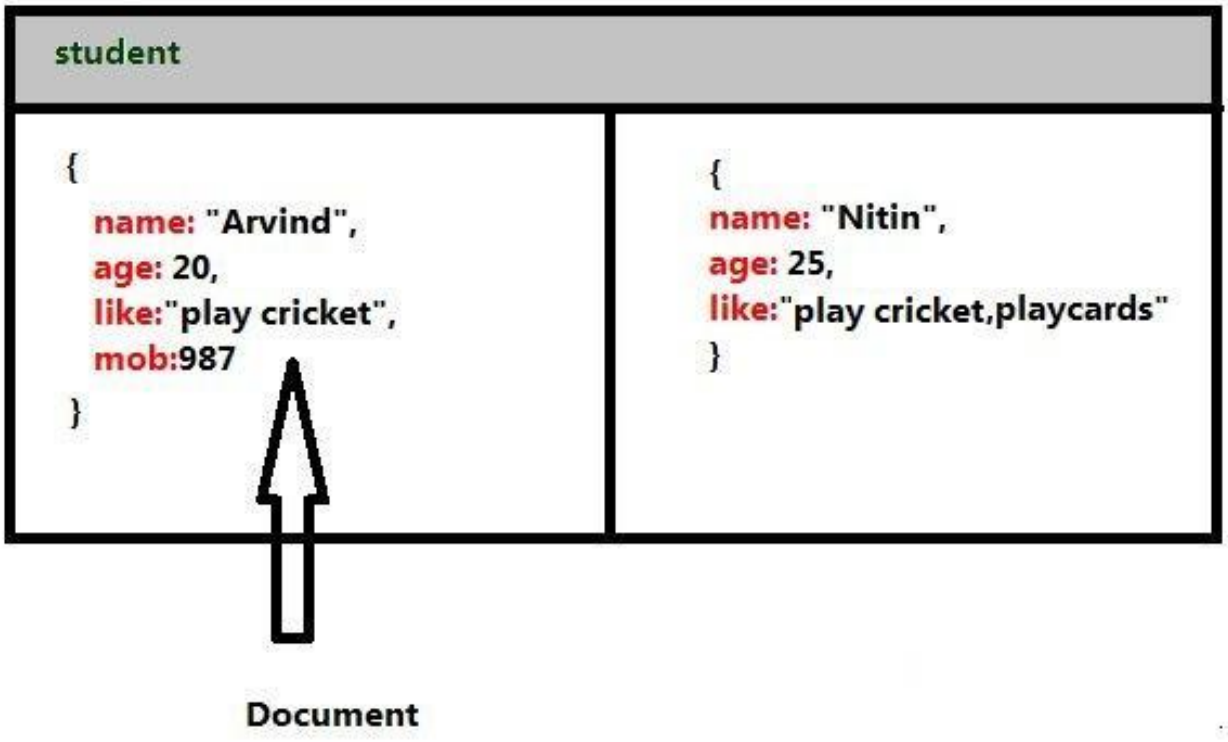
Εικόνα 2.8 - Column Family store

Παραδείγματα: Cassandra, HBase
Δυνατά σημεία: Καλή κλιμάκωση, Αποτελεσματικότητα σε πολύπλοκα σύνολα δεδομένων

Αδύνατα σημεία: Όχι τόσο καλή κλιμάκωση όσο τα key-value stores

- Document store

Αυτό το είδος βασίστηκε στο Lotus Notes και είναι παρεμφερές στα key-value stores. Το μοντελο ουσιαστικά αποτελείται από έγγραφα - των οποίων διατηρούνται οι διαδοχικές εκδοχές - που είναι συλλογές άλλων key-value συλλογών. Τα ημι-δομημένα έγγραφα αποθηκεύονται σε μορφές όπως το JSON.
Οι βάσεις δεδομένων εγγράφων δεν είναι καινούργια ιδέα. Χρησιμοποιήθηκαν σε μια από τις πιο επιφανείς πλατφόρμες επικοινωνίας της δεκαετίας του 90 και επιζεί μέχρι σήμερα, το Lotus Notes που σήμερα ονομάζεται Lotus Domino. Τα API για αυτές τις βάσεις δεδομένων χρησιμοποιούν RESTful web υπηρεσιες και JSON για τη δομή των μηνυμάτων, το οποίο καθιστά εύκολες τις μετακινήσεις δεδομένων.
Μια βάση δεδομένων εγγράφων έχει ένα αρκετά απλό μοντέλο δεδομένων που βασίζεται σε συλλογιές ζευγαριών κλειδιού-τιμής. Μια τυπική εγγραφή θα ήταν κάπως έτσι:

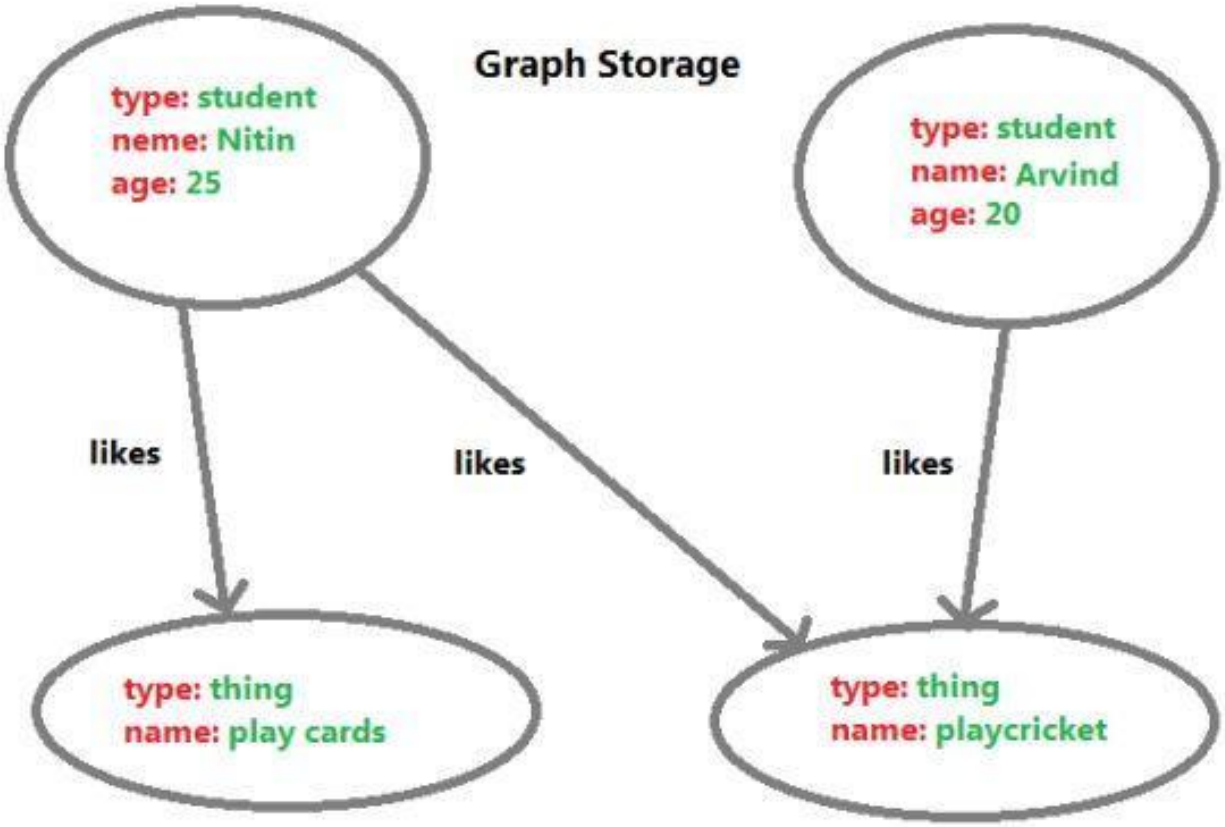


Εικόνα 2.9 - Document store

Παραδείγματα: CouchDB, MongoDb
Δυνατά σημεία*:* Ανοχή σε ελλιπή δεδομένα
Αδύνατα σημεία*:* Απόδοση ερωτημάτων, Μη προτυποποιημένο συντακτικό ερωτημάτων

- Graph store

Αντί για πίνακες με γραμμές και στήλες και την στιβαρή δομή της SQL, χρησιμοποιείται ένα ευέλικτο μοντέλο γραφήματος, το οποίο μπορεί να κλιμακωθεί σε πολλαπλά μηχανήματα. Οι βάσεις δεδομένων NoSQL δεν παρέχουν μια υψηλού επιπέδου δηλωτική γλώσσα ερωτημάτων όπως η SQL για να αποφευχθεί ο επιπλέον χρόνος επεξεργασίας. Αντιθέτως, τα ερωτήματα σε αυτές τις βάσεις γίνονται με ειδικό τρόπο σε κάθε μοντέλο. Πολλές από τις NoSQL πλατφόρμες επιτρέπουν RESTful διεπαφές στα δεδομένα, ενώ άλλες παρέχουν API ερωτημάτων.
Οι βάσεις δεδομένων γραφημάτων επεκτείνουν τις βασικές ιδέες των βάσεων δεδομένων εγγράφων εισάγοντας την έννοια των σχέσεων τύπων μεταξύ δεδομένων ή κόμβων. Το πιο κοινό παράδειγμα είναι η σχέση μεταξύ ανθρώπων σε ένα κοινωνικό δίκτυο όπως το Facebook.
Μια βάση δεδομένων εγγράφων είναι μια μεγάλη πυκνή δομή δικτύου. Ενώ θα χρειαζόταν ώρες σε ένα RDBMS να εξετάσει λεπτομερώς μια τεράστια συνδεδεμένη λίστα ανθρώπων, μια βάση δεδομένων γραφημάτων χρησιμοποιεί πιο εξεζητημένους αλγόριθμους συντομότερων μονοπατιών ώστε να κάνει πιο αποδοτικά τα ερωτήματα δεδομένων. Παρά το ότι είναι πιο αργές από άλλες NoSQL βάσεις, μπορεί να έχουν την πιο πολύπλοκη δομή από όλες και να συνεχίζουν να διατρέχουν δισεκατομμύρια κόμβων και σχέσεων με τεράστια ταχύτητα.



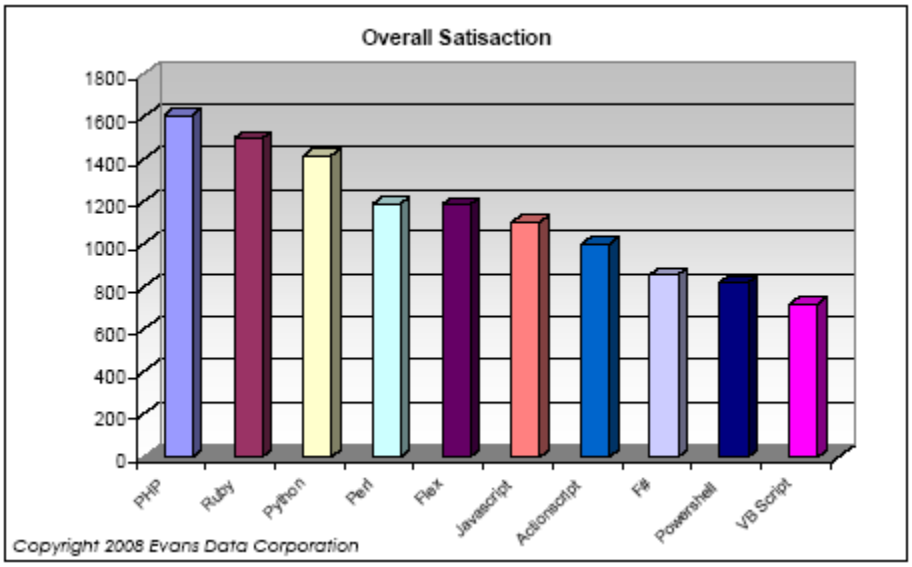
Εικόνα 2.10 - Graph store

Παραδείγματα: Neo4J, InfoGrid, Infinite Graph
Δυνατά σημεία: Αλγόριθμοι γραφημάτων π.χ. shortest path, n degree relationships, κτλ.
Αδύνατα σημεία: Διάσχιση ολόκληρου γράφου για οριστική απάντηση, Δύσκολο να δημιουργηθούν clusters

3
Πλατφόρμες και Εργαλεία

3.1 Ruby

Η Ruby είναι μια γλώσσα προγραμματισμού. Σύχνα, πολλές άλλες λέξεις-κλειδιά συνοδεύουν τη φράση “γλώσσα προγραμματισμού” αλλά αυτές συχνά επιβάλλουν προκατειλημένες αντιλήψεις ως προς το τι σημαίνουν συγκεκριμένα στη Ruby. Πάνω από όλα, η Ruby είναι μια γλώσσα προγραμματισμού.



Εικόνα 3.1 – Οι κορυφαίες scripting γλώσσες προγραμματισμού

Θετικά στοιχεία

Η Ruby είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού. Οτιδήποτε στη Ruby είναι ένα αντικείμενο και για αυτό η Ruby είναι αντικειμενοστραφής με την πιο καθαρή έννοια. Δεν είναι σαν την C++ ή την C όπου υπάρχουν τύποι αντικειμένων και τύποι τιμών ή σαν την Java όπου τα περισσότερα είναι αντικείμενα και λίγοι εγγενής τύποι πρέπει να μετατραπούν για να αντιμετωπστούν σαν αντικείμενα. Στη Ruby όλα είναι αντικείμενα.
Η Ruby είναι μια δυναμική γλώσσα προγραμματισμού. Πολλές γλώσσες χρησιμοποιούν τις έννοιες "compile time" και "run time". Αυτά είναι δύο διαφορετικά σύμπαντα, ο μεταγλωττιστής γνωρίζει εντελώς διαφορετικά πράγματα από το περιβάλλον χρόνου εκτέλεσης. Από την άλλη, η Ruby δεν κάνει διάκριση. Όλα στη Ruby αποφασίζονται την τελευταία στιγμή. Αν πρέπει να κληθεί η μέθοδος foo του αντικειμένου που λέγεται bar, η Ruby όχι μόνο πρέπει να ψάξει που βρίσκεται αυτή η μέθοδος αλλά και που βρίσκεται το αντικείμενο. Αντίθετα, σε γλώσσες όπως η

C, ο μεταγλωττιστής γνωρίζει την ακριβή διεύθυνση και τον τύπο των αντικειμένων καθώς επίσης και τις μεθόδους που έχουν και τις διευθύνσεις τους.
Η Ruby είναι μια γλώσσα προγραμματισμού για το διαδίκτυο. Ενώ αυτό δεν είναι αυστηρά αληθές, μια πλειάδα βιβλιοθηκών όπως οι Sinatra και Rails καθιστούν τη Ruby ιδανική για τον προγραμματισμό εφαρμογών διαδικτύου. Πολλοί γνωρίζουν τη Ruby μόνο λόγω του Ruby on Rails, το οποίο είναι εδώ και καιρό μια από τις πιο σημαντικές, αν όχι η πιο σημαντική, εφαρμογή της Ruby.
Η Ruby είναι μια γλώσσα σεναρίου (scripting). Ο όρος αυτός συχνά χρησιμοποιείται ή ερμηνεύεται λανθασμένα. Πολλοί θεωρούν ότι σημαίνει χρησιμοποιεί διερμηνέα ή ότι δεν είναι μεταγλωττίσιμη. Η Ruby σίγουρα είναι αυτό αλλά είναι επίσης μια κανονική γλώσσα σεναρίου, το οποίο σημαίνει ότι μπορεί να ενσωματωθεί σε άλλα προγράμματα και να χρησιμοποιηθεί για να τα ελέγχει. Ένα καλό παράδειγμα είναι το Google Sketchup, στο οποίο η Ruby χρησιμοποιείται ως πρόσθετη διεπαφή για να ελέγχονται προγραμματιστικά και το πρόγραμμα και τα 3D μοντέλα που σχεδιάζονται από το πρόγραμμα. Ενώ υπάρχουν ευκολότερες γλώσσες σεναρίου που μπορούν να ενσωματωθούν, η Ruby είναι μια συμπαγής και πλήρης, από άποψη χαρακτηριστικών, επιλογή.
Η Ruby είναι ελεύθερη. Αυτό σημαίνει ότι δεν υπάρχει κόστος για την κατεβάσει κανείς και να την χρησιμοποιήσει για οποιοδήποτε σκοπό. Όχι μόνο είναι ελεύθερος ο επίσημος διερμηνέας της Ruby αλλά υπάρχουν και αρκετοί άλλοι ελεύθεροι διερμηνείς για διάφορες πλατφόρμες. Η Ruby είναι επίσης ελεύθερο λογισμικό, το οποίο σημαίνει ότι κάθε χρήστης της είναι μπορεί να δει και να τροποποιήσει τον πηγαίο κώδικα ανάλογα με τις ανάγκες του.

Αρνητικά στοιχεία

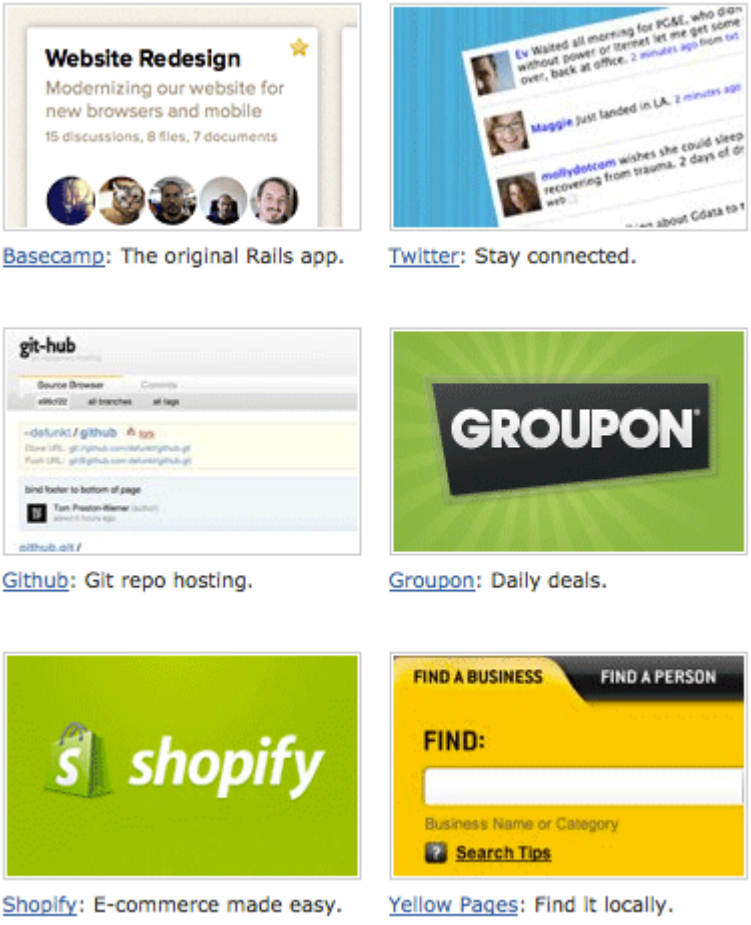
Η Ruby είναι αργή. Μεταγλωττισμένος κώδικας από γλώσσες όπως οι C++ και C# αποδίδουν πολύ καλύτερα. Ακόμα και άλλες γλώσσες στην ίδια κατηγορία όπως οι Python και Perl μπορεί να είναι πολύ ταχύτερες. Παρ'όλα αυτά, αυτή η αντίληψη προέρχεται από μια εποχή που η Ruby ήταν πολύ αργή και συγκεκριμένα πριν από την έκδοση 1.9.x. Οι τωρινές εκδόσεις έχουν βελτιωθεί πολύ και είναι αρκετά ταχύτερες.
Η Ruby είναι αντικειμενοστραφής γλώσσα αλλά μόνο αυτό. Αν κάπου δεν ταιριάζει αυτό το προγραμματιστικό πρότυπο καλό θα ήταν να αποφευχθεί η χρήση της. Άλλες γλώσσες, όπως η Perl, προσπαθούν να ικανοποιήσουν τις ανάγκες όλων των προγραμματιστών.
Η Ruby είναι στη μόδα. Τέτοιες γλώσσες έρχονται και φεύγουν. Η δημοτικότητα της Ruby εκτοξεύθηκε γύρω στο 2005 όταν υπήρχε η θέρμη για το Ruby on Rails. Αντμετωπιζόταν ως παροδικοί και πολλοί υποστήριζαν ότι δεν θα μείνει δημοφιλής για πολύ και, ενώ δεν είναι γνωστό αν σε 10 χρόνια θα είναι δημοφιλής, η ποσότητα κώδικα που γράφεται για το Rubygems και στο Github υποδεικνύει ότι είναι αυθεντικά δημοφιλής.

3.2 Ruby on Rails

Το Ruby on Rails, συχνά αποκαλούμενο και Rails, είναι ένα framework ανοικτού κώδικα για εφαρμογές ιστού το οποίο τρέχει μέσω της γλώσσας προγραμματισμού Ruby. Είναι ένα

framework πλήρους στοίβας: επιτρέπει τη δημιουργία σελιδών και εφαρμογών που συγκεντρώνουν πληροφορία από τον εξυπηρετητή ιστού, μιλάνε με ή στέλνουν ερωτήματα στη βάση δεδομένων και παρέχει πρότυπα έτοιμα για χρήση.

Ως αποτέλεσμα, το Rails παρέχει ένα σύστημα δρομολόγησης που είναι ανεξάρτητο από τον εξυπηρετητή ιστού. Επίσης, δίνει έμφαση στη χρήση πολύ γνωστών μοτίβων και αρχών της τεχνολογίας λογισμικού, όπως το μοτίβο active record, την αρχή convention over configuration (CoC), την αρχή don't repeat yourself (DRY), το σχεδιαστικό πρότυπο model-view-controller (MVC) και τη στρατηγική behavior driven development (BDD).



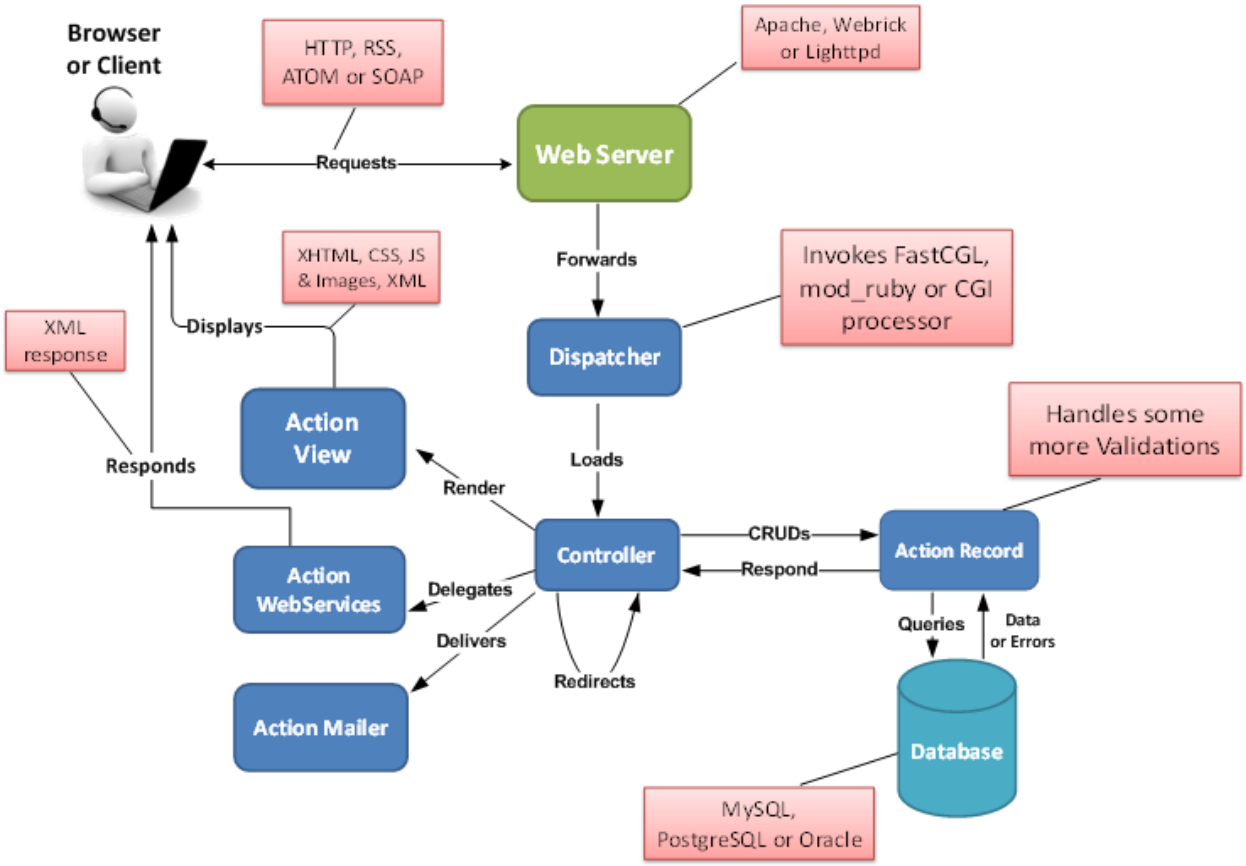
Εικόνα 3.2 – Ιστοσελίδες που χρησιμοποιούν το Ruby on Rails

- Active record

Στην τεχνολογία λογισμικού, το active record είναι ένα αρχιτεκτονικό μοτίβο που συναντάται σε λογισμικό που αποθηκεύει τα δεδομένα του σε σχεσιακές βάσεις δεδομένων. Ονομάστηκε από τον Martin Fowler στο βιβλίο του "Patterns of Enterprise Application Architecture" που εκδόθηκε το 2003. Η διεπαφή ενός αντικειμένου που ακολουθεί αυτό το μοτίβο θα συμπεριλάμβανε λειτουργίες όπως Εισαγωγή, Ενημέρωση και Διαγραφή όπως επίσης και ιδιότητες που αντιστοιχούν περισσότερο ή λιγότερο άμεσα στις στήλες στον υποκείμενο πίνακα της βάσης

δεδομένων.
Το μοτίβο active record είναι μια προσέγγιση της προσπέλασης των δεδομένων μιας βάσης. Ένας πίνακας ή μια όψη της βάσης δεδομένων περικλείεται σε μια κλάση. Έτσι, ένα στιγμιότυπο αντικειμένου αντιστοιχίζεται σε μια γραμμή του πίνακα. Μετά τη δημιουργία ενός αντικειμένου, μια νέα γραμμή προστίθεται στον πίνακα κατά την αποθήκευση. Όποιο αντικείμενο φορτώνεται παίρνει τα δεδομένα του από τη βάση. Όταν ένα αντικείμενο ενημερώνεται, η αντίστοιχη γραμμή του πίνακα ενημερώνεται επίσης. Η περικλείουσα κλάση υλοποιεί μεθόδους προσπέλασης ή ιδιότητες για κάθε στήλη στον πίνακα ή στην όψη.

Αυτό το μοτίβο χρησιμοποιείται συνήθως από εργαλεία αποθήκευσης αντικειμένων και στο object-relational mapping (ORM). Συνήθως, οι σχέσεις δευτερεύοντος κλειδιού εκτίθενται ως ένα στιγμιότυπο αντικειμένου κατάλληλου τύπου μέσω κάποιας ιδιότητας.



Εικόνα 3.3 – Αρχιτεκτονική του Ruby on Rails

- Model–view–controller (MVC)

Το MVC είναι αρχιτεκτονικό μοτίβο λογισμικού για την υλοποίηση διεπαφών χρηστών. Διαιρεί μια εφαρμογή σε τρία διασυνδεδεμένα τμήματα έτσι ώστε να ξεχωρίσει τις εσωτερικές αναπαραστάσεις της πληροφορίας από τους τρόπους που αυτή η πληροφορία παρουσιάζεται ή γίνεται αποδεκτή από τον χρήστη.

Το κεντρικό κομμάτι του MVC, το μοντέλο (model), περιλαμβάνει την συμπεριφορά της εφαρμογής όσον αφορά το πεδίο του προβλήματός της, ανεξάρτητα από τη διεπαφή με τον χρήστη. Το μοντέλο διαχειρίζεται άμεσα τα δεδομένα, τη λογική και τους κανόνες της εφαρμογής. Μια όψη (view) μπορεί να είναι οποιαδήποτε έξοδος αναπαράστασης πληροφορίας, όπως ένα γράφημα ή ένα διάγραμμα. Το τρίτο κομμάτι, ο ελεγκτής (controller), δέχεται είσοδο και τη μετατρέπει σε εντολές για το μοντέλο ή την όψη.

Πέρα από τη διαίρεση της εφαρμογής σε τρία κομμάτια, το MVC ορίζει τις αλληλεπιδράσεις μεταξύ τους. Ένας ελεγκτής μπορεί να στείλει εντολές στο μοντέλο για να ενημερώσει την κατάστασή του. Μπορεί επίσης να στείλει εντολές στην αντίστοιχή του όψη για να αλλάξει το παρουσιάζει η όψη το μοντέλο. Ένα μοντέλο ειδοποιεί τις όψεις και τους ελεγκτές που του αντιστοιχούν όταν γίνεται κάποια αλλαγή στην κατάστασή του. Αυτή η ειδοποίηση επιτρέπει στις όψεις να παράγουν ενημερωμένα αποτελέσματα και στους ελεγκτές να αλλάξουν διαθέσιμο σύνολο εντολών. Σε κάποιες περιπτώσεις μια υλοποίηση του MVC μπορεί αντίθετα να είναι παθητική, με την έννοια ότι θα πρέπει τα άλλα κομμάτια να στέλνουν ερωτήσεις στο μοντέλο για να μάθουν αν άλλαξε η κατάστασή του αντί να περιμένουν να ειδοποιηθούν. Μια όψη ζητά πληροφορίες από το μοντέλο για να παράγει μια αναπαράστασή του για τον χρήστη. Παρόλο που αρχικά αναπτύχθηκε για επιτραπέζιους υπολογιστές, το MVC έχει υιοθετηθεί ευρέως ως αρχιτεκτονική για εφαρμογές διαδικτύου σε μεγάλες γλώσσες προγραμματισμού. Αρκετά εμπορικά και μη εμπορικά frameworks εφαρμογών έχουν δημιουργηθεί, τα οποία επιβάλλουν αυτό το μοτίβο. Αυτά τα frameworks ποικίλουν ως προς τον τρόπο που μοιράζονται οι ευθύνες του MVC μεταξύ πελάτη και εξυπηρετητή.

Τα πρώτα frameworks που ενστερνίστηκαν το MVC επέλεξαν μια προσέγγιση όπου σχεδόν όλη η λογική των τριών κομματιών τοποθετούταν στον εξυπηρετητή. Σε αυτή την προσέγγιση, ο πελάτης στέλνει είτε αιτήματα υπερσυνδέσμου ή δεδομένα από φόρμα στον ελεγκτή και στη συνέχεια λαμβάνει μια πλήρη και ενημερωμένη ιστοσελίδα (ή άλλο έγγραφο) από την όψη. Σε αυτή την περίπτωση το μοντέλο βρίσκεται εξ ολοκλήρου στον εξυπηρετητή. Καθώς οι τεχνολογίες της πλευράς του πελάτη ωρίμασαν, δημιουργήθηκαν frameworks όπως τα JavaScriptMVC και Backbone που επιτρέπουν στα κομμάτια του MVC να εκτελούνται μερικώς στον πελάτη (με τη βοήθεια και του AJAX).

- Convention over Configuration

Η αρχή “Convention over configuration” (ή αλλιώς “coding by convention”) είναι ένα πρότυπο σχεδίασης λογισμικού που επιδιώκει να μειώσει τον αριθμό των αποφάσεων που πρέπει να πάρουν οι προγραμματιστές, κερδίζοντας με αυτόν τον τρόπο σε απλότητα χωρίς απαραίτητα να χάνει σε ευελιξία.

Η φράση ουσιαστικά σημαίνει ότι ένας προγραμματιστής χρειάζεται μόνο να καθορίσει τις μη συμβατικές πλευρές της εφαρμογής. Για παράδειγμα, αν υπάρχει μια κλάση “Sale” στο μοντέλο, ο αντίστοιχος πίνακας στη βάση δεδομένων λέγεται “sales” εξ ορισμού. Μόνο αν υπάρξει παρέκκλιση από αυτή τη σύμβαση, όπως π.χ. να ονομαστεί ο πίνακας “sale”, θα πρέπει να γραφεί επιπλέον κώδικας που να αφορά αυτά τα ονόματα.

Τα κίνητρα για την ανάπτυξη αυτής της αρχής αφορούσαν την πλειάδα ρυθμίσεων που χρειάζεται η ανάπτυξη ενός έργου λογισμικού. Αρκετά frameworks χρειάζονται πολλαπλά αρχεία ρυθμίσεων, καθένα με πολλές επιμέρους ρυθμίσεις. Αυτά παρέχουν πληροφορίες ειδικά για κάθε project, με εύρος από URLs μέχρι αντιστοιχίες μεταξύ κλάσεων και πινάκων της βάσης. Ένας μεγάλος αριθμός αρχείων ρυθμίσεων με πολλές παραμέτρους είναι συχνά δύσκολο να συντηρηθούν.

- Don't repeat yourself (DRY)

Στην τεχνολογία λογισμικού, το DRY είναι μια αρχή ανάπτυξης λογισμικού που στόχος της οποίας είναι η μείωση της επανάληψης κάθε είδους, το οποίο είναι ιδιαίτερα χρήσιμο σε πολυεπίπεδες αρχιτεκτονικές. Η αρχή αυτή επεξηγείται ως εξής: Κάθε κομμάτι γνώσης πρέπει να έχει μια μοναδική, ξεκάθαρη και έγκυρη αναπαράσταση μέσα σε ένα σύστημα. Η αρχή αυτή διατυπώθηκε από τους Andy Hunt και Dave Thomas στο βιβλίο τους “The Pragmatic Programmer”. Την εφαρμόζουν αρκετά ευρέως για να συμπεριλάβει σχήματα βάσεων δεδομένων, σχέδια ελέγχου, το σύστημα ανάπτυξης και ακόμα και την τεκμηρίωση. Όταν εφαρμόζεται επιτυχώς, η μετατροπή ενός στοιχείου ενός συστήματος δεν απαιτεί αλλαγές σε άλλα στοιχεία που δεν σχετίζονται λογικά με αυτό. Επιπροσθέτως, τα στοιχεία που είναι λογικά συσχετισμένα αλλάζουν όλα μαζί με τρόπο προβλέψιμο και ομοιόμορφο και έτσι παραμένουν συγχρονισμένα. Πέρα από τη χρήση μεθόδων και υπο-ρουτινών στον κώδικα, οι Thomas και Hunt βασίζονται σε γεννήτριες κώδικα, αυτόματα συστήματα ανάπτυξης και γλώσσες σεναρίου για να διασφαλίσουν την εφαρμογή της αρχής σε όλα τα στρώματα.

Αυτή η φιλοσοφία είναι κυρίαρχη σε αρχιτεκτονικές στις οποίες τα τεχνουργήματα λογισμικού εξάγονται από ένα κεντρικό μοντέλο αντικειμένου που εκφράζεται σε μια μορφή όπως η UML. Ο κώδικας που ακολουθεί την αρχή DRY και παράγεται από μετασχηματισμό δεδομένων και γεννήτριες κώδικα επιτρέπει στον προγραμματιστή να αποφύγει καταστάσεις “copy-paste”. Τέτοιος κώδικας συνήθως βοηθάει στο να είναι τα μεγάλα συστήματα λογισμικού πιο εύκολα στη συντήρηση, εφόσον οι μετασχηματισμοί δεδομένων είναι εύκολο να δημιουργηθούν και να συντηρηθούν. Μερικά παραδείγματα τεχνικών προγραμματισμού DRY είναι εργαλεία όπως τα XDoclet και XSLT.

- Behavior Driven Development (BDD)

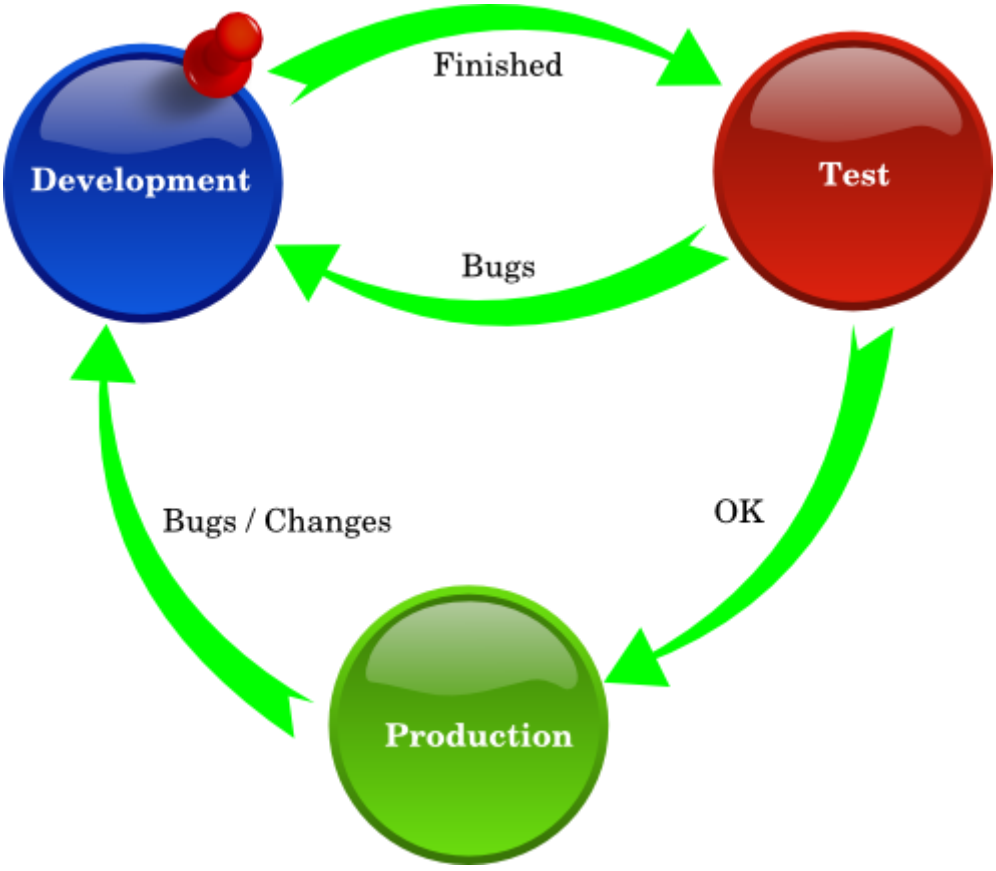
Στην τεχνολογία λογισμικού, το BDD είναι μια διαδικασία ανάπτυξης λογισμικού που βασίζεται στο TDD (Test Driven Development). Το BDD συνδυάζει τις γενικές τεχνικές και αρχές του TDD με ιδέες από domain driven σχεδιασμό και αντικειμενοστραφή ανάλυση και σχεδιασμό για να παρέχει στις ομάδες ανάπτυξης λογισμικού και διαχείρισης κοινά εργαλεία και κοινή διαδικασία για να συνεργάζονται στην ανάπτυξη λογισμικού.

Παρόλο που το BDD είναι καταρχήν μια ιδέα σχετικά με το πως θα πρέπει να γίνεται η διαχείριση της ανάπτυξης λογισμικού από πλευράς επιχειρηματικών συμφερόντων αλλά και τεχνικής γνώσης, η εφαρμογή του προϋποθέτει τη χρήση εξειδικευμένων εργαλειών λογισμικού για να υποστηριχθεί η διαδικασία ανάπτυξης. Παρά το ότι τα εργαλεία αυτά συχνά αναπτύσσονται ειδικά για χρήση σε BDD projects, μπορούν να αντιμετωπιστούν ως εξειδικευμένη μορφή των εργαλείων που υποστηρίζουν το TDD.

Στον πυρήνα του, το BDD είναι μια εξειδικευμένη εκδοχή του TDD που εστιάζει στις συμπεριφορικές προδιαγραφές των μονάδων λογισμικού. Το TDD είναι μια μεθοδολογία ανάπτυξης λογισμικού που δηλώνει ότι για κάθε τμήμα λογισμικού ένας προγραμματιστής πρέπει να:

- ορίσει αρχικά ένα σύνολο τεστ για αυτό το τμήμα
- υλοποιήσει στη συνέχεια το τμήμα
- επικυρώσει τελικά ότι η υλοποίηση του τμήματος κάνει το τεστ να επιτυγχάνει

Αυτός ο ορισμός είναι αρκετά ασαφής, με την έννοια του ότι επιτρέπει τα τεστ να είναι υψηλού επιπέδου απαιτήσεων λογισμικού, χαμηλού επιπέδου τεχνικών λεπτομερειών ή οτιδήποτε ανάμεσα σε αυτά. Ο πρώτος που ανέπτυξε το BDD (Dan North) το επινόησε επειδή ήταν δυσαρεστημένος με έλλειψη προδιαγραφών μέσα στο TDD ως προς το τι θα πρέπει να ελεγχθεί και πως. Έτσι, ένας τροπος ερμηνείας του BDD είναι ότι αποτελεί μια συνεχιζόμενη ανάπτυξη του TDD που κάνει πιο συγκεκριμένες επιλογές από αυτό.



Εικόνα 3.4 – Behavior Driven Development

Το BDD προσδιορίζει ότι τα τεστ για κάθε τμήμα λογισμικού θα πρέπει να είναι καθορισμένα όσον αφορά την επιθυμητή συμπεριφορά τους. Η επιθυμητή συμπεριφορά αποτελείται από τις απαιτήσεις που θέτει η επιχείρηση. Αυτό αφορά την επιθυμητή συμπεριφορά που έχει επιχειρηματική αξία για την οντότητα που ζήτησε την κατασκευή του τμήματος λογισμικού. Με

όρους BDD αυτό αναφέρεται ως ότι το BDD είναι μια “outside-in” δραστηριότητα. Ακολουθώντας αυτή τη βασική επιλογή, μια δεύτερη που έγινε από το BDD σχετίζεται με το πως θα πρέπει να καθορίζεται η επιθυμητή συμπεριφορά. Σε αυτή την περιοχή το BDD επιλέγει να χρησιμοποιήσει μια ημι-επίσημη μορφή για τις προδιαγραφές συμπεριφοράς την οποία δανείζεται από τις προδιαγραφές τύπου “user story” από τον τομέα αντικειμενοστραφούς ανάλυσης και σχεδιασμού. Το BDD ορίζει ότι οι επιχειρηματικοί αναλυτές και οι προγραμματιστές πρέπει να συνεργάζονται σε αυτόν τον τομέα και θα πρέπει να καθορίζουν τη συμπεριφορά σε user stories, τα οποία θα πρέπει να είναι καταγραμμένα με εκτενή περιγραφή σε ένα έγγραφο. Κάθε user story θα πρέπει να ακολουθεί την παρακάτω δομή:

Τίτλος: Η ιστορία θα πρέπει να έχει έναν ξεκάθαρο τίτλο

Αφήγηση

Ένα μικρό εισαγωγικό κομμάτι που θα πρέπει να καθορίζει:

- ποιός είναι ο κύριος δράστης της ιστορίας
- ποιό αποτέλεσμα θέλει ο δράστης να έχει η ιστορία
- τι θα κερδίσει ο δράστης από αυτό το αποτέλεσμα

Κριτήρια αποδοχής ή σενάρια

μια περιγραφή κάθε ξεχωριστής περίπτωσης της αφήγησης. Ένα τέτοιο σενάριο έχει την ακόλουθη δομή:

- Ξεκινάει με τον ορισμό της αρχικής κατάστασης που θεωρείται αληθής στην αρχή του σεναρίου. Αυτό μπορεί να αποτελείται από μια ή περισσότερες συνθήκες
- Στη συνέχεια δηλώνει ποιά γεγονότα προκαλούν την αρχή του σεναρίου
- Τέλος, δηλώνει το αναμενόμενο αποτέλεσμα, σε μια ή περισσότερες συνθήκες

Το BDD δεν έχει τυπικές απαιτήσεις για το πως ακριβώς πρέπει να γραφούν αυτές οι ιστορίες, αλλά επιμένει στο ότι κάθε ομάδα που το χρησιμοποιεί θα πρέπει να βρει μια απλή προτυποποιημένη μορφή για να καταγράφει τις ιστορίες που θα περιέχουν τα παραπάνω στοιχεία. Παρ’όλα αυτά, το 2007 ο Dan North πρότεινε ένα πρότυπο για μια μορφή που υιοθετήθηκε από πολλά εργαλεία για BDD. Ένα σύντομο παράδειγμα είναι το παρακάτω:

Story: Returns go to stock

In order to keep track of stock
As a store owner
I want to add items back to stock when they're returned

Scenario 1: Refunded items should be returned to stock
Given a customer previously bought a black sweater from me
And I currently have three black sweaters left in stock
When he returns the sweater for a refund
Then I should have four black sweaters in stock

Scenario 2: Replaced items should be returned to stock
Given that a customer buys a blue garment

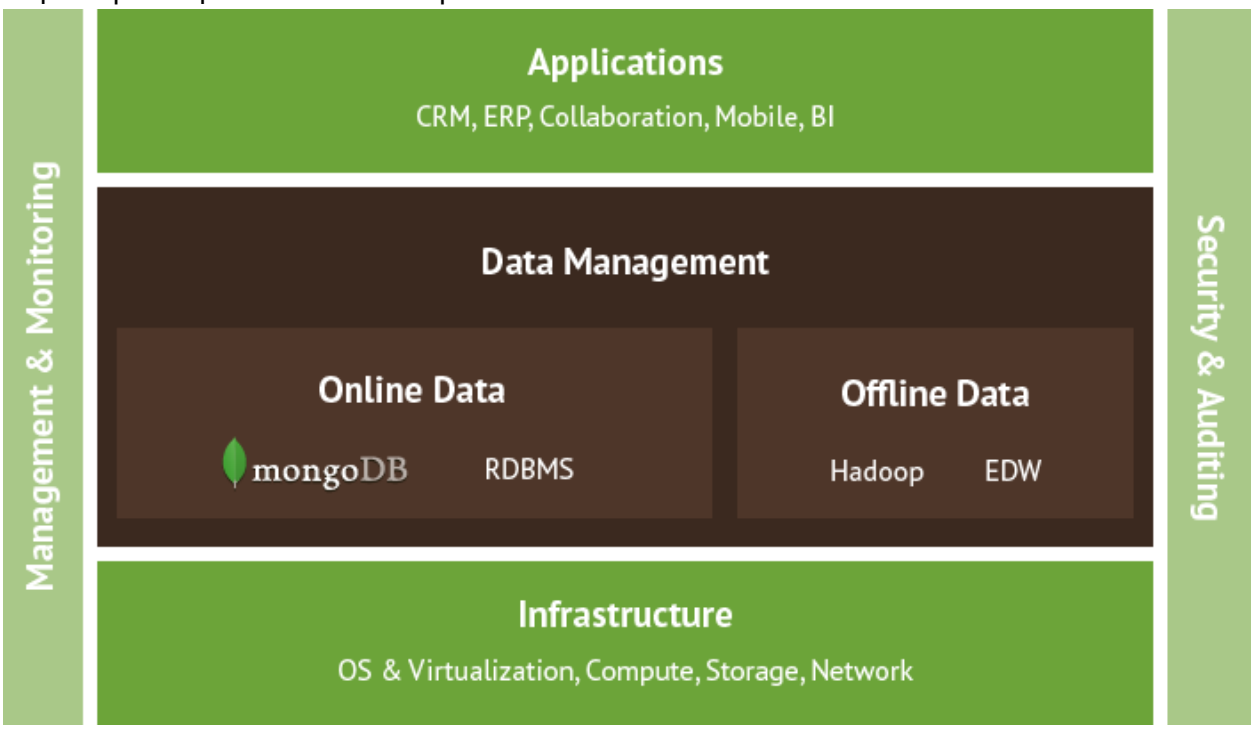
And I have two blue garments in stock
And three black garments in stock.
When he returns the garment for a replacement in black,
Then I should have three blue garments in stock
And two black garments in stock

Τα σενάρια θα ήταν ιδανικό να διατυπώνονται δηλωτικά και όχι προστακτικά, σε επιχειρηματική γλώσσα, χωρίς αναφορές στα στοιχεία του γραφικού περιβάλλοντος μέσω των οποίων γίνονται οι αλληλεπιδράσεις.
Αυτή η μορφή ονομάζεται γλώσσα Gherkin, η οποία έχει συντακτικό παρόμοιο με το παραπάνω παράδειγμα. Παρ'όλα αυτά, ο όρος Gherkin αφορά συγκεκριμένα τα εργαλεία Cucumber και JBehave.

3.3 MongoDB

Τι είναι η MongoDB?

Η MongoDB είναι μια NoSQL βάση δεδομένων σχεδιασμένη για τον τρόπο που φτιάχνονται και τρέχουν οι εφαρμογές σήμερα χρησιμοποιώντας σύγχρονες τεχνικές ανάπτυξης, προγραμματιστικά μοντέλα και υπολογιστικούς πόρους. Ως αποτέλεσμα, δίνει τη δυνατότητα στις επιχειρήσεις να είναι πιο ευέλικτες και κλιμακώσιμες, να δημιουργούν νέες εφαρμογές, να βελτιώσουν την εμπειρία των πελατών και να επιταχύνουν το χρόνο μέχρι την αγορά ενώ παράλληλα να μειώσουν τα κόστη.



Εικόνα 3.5 – Επισκόπηση της MongoDB

Νέες προκλήσεις

Πως φτιάχνουμε εφαρμογές

- Νέοι και πολύπλοκοι τύποι δεδομένων. Πλούσιες δομές δεδομένων με δυναμικά χαρακτηριστικά, μικτή δομή, κείμενο, πολυμέσα, πίνακες και άλλοι πολύπλοκοι τύποι είναι συνηθισμένοι στις σημερινές εφαρμογές.
- Σύγχρονες γλώσσες προγραμματισμού. Οι αντικειμενοστραφείς γλώσσες προγραμματισμού αλληλεπιδρούν με δεδομένα σε δομές που είναι δραματικά διαφορετικές από τον τρόπο που αποθηκεύονται τα δεδομένα σε μια σχεσιακή βάση δεδομένων.
- Ταχύτερη ανάπτυξη. Οι ομάδες μηχανικών λογισμικού σήμερα ασπάζονται σύντομους επαναληπτικούς κύκλους ανάπτυξης.

Πως τρέχουμε εφαρμογές

- Νέα κλιμακωσιμότητα για τα Big Data. New Scalability for Big Data. Ο λειτουργικός και αναλυτικός φόρτος εργασίας δοκιμάζει τις παραδοσιακές δυνατότητες σε μια ή περισσότερες διαστάσεις κλιμάκωσης, διαθεσιμότητας, απόδοσης και αποδοτικότητας κόστους.
- Γρήγορη, πραγματικού χρόνου απόδοση. Οι χρήστες αναμένουν συνεπή και αλληλεπιδραστική εμπειρία από τις εφαρμογές σε πολλούς τύπους διεπαφών.
- Νέα υπολογιστικά περιβάλλοντα. Οι απαιτήσεις της υποδομής για εφαρμογές μπορούν πολύ εύκολα να ξεπεράσουν τους πόρους ενός μοναδικού υπολογιστή και το cloud πλέον παρέχει μαζική, ελαστική και αποδοτική ως προς το κόστος υπολογιστική χωρητικότητα σε ένα μοντέλο καταμετρούμενου κόστους.

Περίληψη χαρακτηριστικών MongoDB

Η MongoDB ενστερνίζεται αυτές τις νέες πραγματικότητες μέσω βασικών καινοτομιών.

- Μοντέλο δεδομένων εγγράφου. Τα δεδομένα αποθηκεύονται σε μια δομή που αντιστοιχίζεται σε αντικείμενα στις σύγχρονες γλώσσες προγραμματισμού και είναι εύκολο να το καταλάβουν οι προγραμματιστές.
- Πλούσιο μοντέλο ερωτημάτων. Η MongoDB είναι κατάλληλη για μια ευρεία γκάμα εφαρμογών. Παρέχει πλούσια υποστήριξη ευρετηρίων και ερωτημάτων, συμπεριλαμβανομένων και δευτερευόντων, γεωχωρικών και αναζήτησης κειμένου καθώς και το Aggregation Framework και εγγενές MapReduce.
- Ιδιωματικοί οδηγοί. Οι προγραμματιστές αλληλεπιδρούν με τη βάση δεδομένων μέσω εγγενών βιβλιοθηκών που είναι ενσωματωμένες με το αντίστοιχο περιβάλλον τους και τα αποθετήρια κώδικά τους, κάνοντας τη MongoDB απλή και φυσική στη χρήση.
- Οριζόντια κλιμακωσιμότητα. Καθώς ο όγκος και η ροή των δεδομένων αυξάνονται, οι προγραμματιστές μπορούν να εκμεταλλευθουν το υλικό του εμπορίου και του cloud για να αυξήσουν τη χωρητικότητα των MongoDB συστημάτων.

- Υψηλή διαθεσιμότητα. Πολλαπλά αντίγραφα διατηρούνται με εγγενή αντιγραφή. Η αυτόματη μετάβαση στους δευτερεύοντες κόμβους, εξυπηρετητές και data centers καθιστά δυνατό να επιτευχθεί περίοδος λειτουργίας επιπέδου επιχείρησης χωρίς ειδικό κώδικα και πολύπλοκη ρύθμιση.
- Απόδοση εντός μνήμης. Τα δεδομένα διαβάζονται από και γράφονται στη RAM ενώ μετάφερονται και στον δίσκο για ανθεκτικότητα, παρέχοντας γρήγορη απόδοση και εξαλείφονταςτην ανάγκη για ξεχωριστό στρώμα caching.
- Ευελιξία. Από το μοντέλο δεδομένων εγγράφου στην ανάπτυξη εφαρμογών σε πολλά data centers, στη ρυθμίσιμη συνέπεια στις επιλογές διαθεσιμότητας επιπέδου λειτουργίας, η MongoDB παρέχει τεράστια ευελιξία στις ομάδες ανάπτυξης και λειτουργίας και για αυτούς τους λόγους είναι κατάλληλη για μια ευρεία γκάμα εφαρμογών σε πολλές βιομηχανίες.

Το μοντέλο δεδομένων της MongoDB

Δεδομένα ως έγγραφα

Η MongoDB αποθηκεύει τα δεδομένα ως έγγραφα σε δυαδική αναπαράσταση που ονομάζεται BSON (Binary JSON). Τα έγγραφα που τείνουν να έχουν παρόμοια δομή οργανώνονται ως συλλογές. Για λόγους σύγκρισης, θα μπορούσαν να θεωρηθούν οι συλλογές αντίστοιχες των πινάκων σε μια σχεσιακή βάση δεδομένων, τα έγγραφα παρόμοια με τις γραμμες και τα πεδία παρόμοια με τις στήλες.

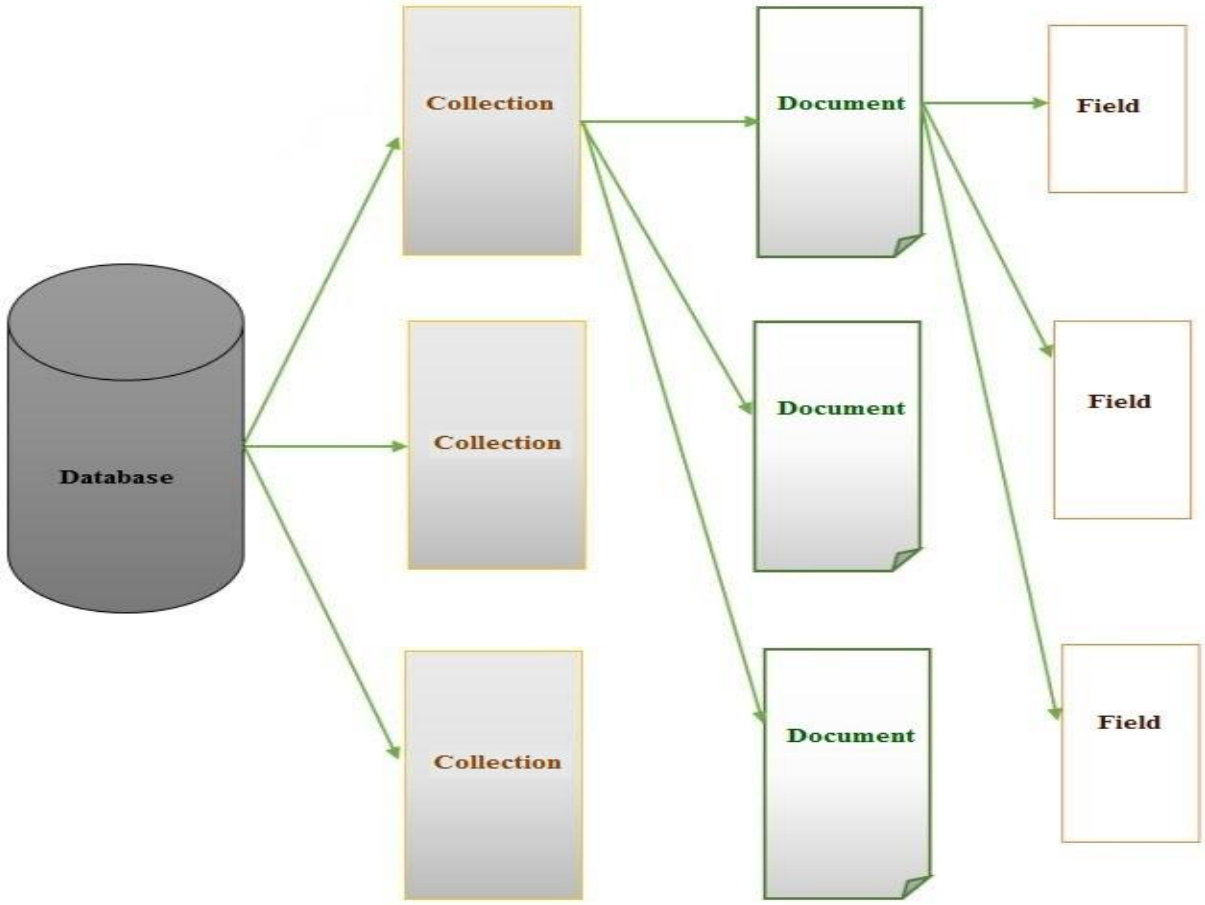
Δυναμικό σχήμα

Τα έγγραφα της MongoDB μπορεί να διαφέρουν σε δομή. Για παράδειγμα, όλα τα έγγραφα που περιγράφουν χρήστες μπορεί να περιέχουν το user id και την τελευταία ημερομηνία που μπήκαν στο σύστημα αλλά μόνο μερικά από αυτά μπορεί να περιέχουν την ταυτότητα του χρήστη για μια ή περισσότερες third-party εφαρμογές. Τα πεδία μπορεί να ποικίλουν από έγγραφο σε έγγραφο. Δεν υπάρχει ανάγκη να δηλωθεί η δομή των εγγράφων στο σύστημα. Αν ένα νέο πεδίο χρειαστεί να προστεθεί σε ένα έγγραφο τότε το πεδίο μπορεί να δημιουργηθεί χωρίς να επηρεάσει όλα τα άλλα έγγραφα στο σύστημα, χωρίς να ενημερωθεί κάποιος κεντρικός κατάλογος συστήματος και χωρίς να χρειαστεί να σταματήσει να είναι διαθέσιμο το σύστημα.

Το μοντέλο ερωτημάτων της MongoDB

Ιδιωματικοί οδηγοί

Η MongoDB παρέχει εγγενείς οδηγούς για όλες τις δημοφιλείς γλώσσες προγραμματισμού και τα frameworks έτσι ώστε να κάνει τον προγραμματισμό φυσικό. Οι υποστηριζόμενοι οδηγοί περιλαμβάνουν Java, .NET, Ruby, PHP, JavaScript, node.js, Python, Perl, PHP, Scala και άλλα. Οι οδηγοί της MongoDB είναι σχεδιασμένοι να είναι ιδιωματικοί για την κάθε γλώσσα.



Εικόνα 3.6 – Δομή δεδομένων της MongoDB

Τύποι ερωτημάτων

Η MongoDB υποστηρίζει πολλούς τύπους ερωτημάτων. Ένα ερώτημα μπορεί να επιστρέψει ένα έγγραφο ή ένα υποσύνολο συγκεκριμένων πεδίων μέσα σε ένα έγγραφο:

- Τα ερωτήματα κλειδιού-ζεύγους επιστρέφουν αποτελέσματα βασισμένα σε οποιοδήποτε πεδίο εντός του εγγράφου, συχνά στο πρωτεύον κλειδί.
- Τα ερωτήματα εύρους επιστρέφουν αποτελέσματα βασισμένα σε τιμές που ορίζονται ως ανισότητες (π.χ. μεγαλύτερο από, λιγότερο από ή ίσο με).
- Τα γεωχωρικά ερωτήματα επιστρέφουν αποτελέσματα βασισμένα σε κριτήρια εγγύτητας, τομής και ένωσης όπως προσδιορίζονται από ένα σημείο, γραμμή, κύκλο ή πολύγωνο.
- Τα ερωτήματα αναζήτησης κειμένου επιστρέφουν αποτελέσματα βασισμένα σε ορίσματα κειμένου χρησιμοποιώντας τελεστές Boolean (π.χ. AND, OR, NOT).
- Τα ερωτήματα Aggregation Framework επιστρέφουν αθροίσματα τιμών που επιστρέφονται από το ερώτημα (π.χ. count, min, max, average, παρόμοιο με το GROUP BY της SQL).
- Τα ερωτήματα MapReduce εκτελούν πολύπλοκες επεξεργασίες δεδομένων οι οποίες εκφράζονται σε JavaScript και εκτελούνται πάνω στα δεδομένα της βάσης.

Ευρετηριοποίηση

Όπως στα περισσότερα συστήματα διαχείρισης βάσεων δεδομένων, τα ευρετήρια είναι ζωτικής σημασίας μηχανισμός για τη βελτιστοποίηση της απόδοσης των συστημάτων MongoDB. Και ενώ τα ευρετήρια θα βελτιώσουν την απόδοση κάποιων λειτουργιών κατά τάξεις μεγέθους, έχουν αντίστοιχα κόστη στη μορφή πιο αργών εγγραφών, χρήσης δίσκου και μνήμης. Η MongoDB συμπεριλαμβάνει υποστήριξη για πολλούς τύπους ευρετηρίων σε κάθε πεδίο στο έγγραφο.

Traditional Database	MongoDB
Relational	Document-Orientated
Server	Server
Database	Database
Table	Collection
Row	Document
Column	Attribute
SQL Query	BSON Query
Index	Index

Εικόνα 3.7 – Βασικές έννοιες της MongoDB

Η διαχείριση δεδομένων της MongoDB

Auto-sharding

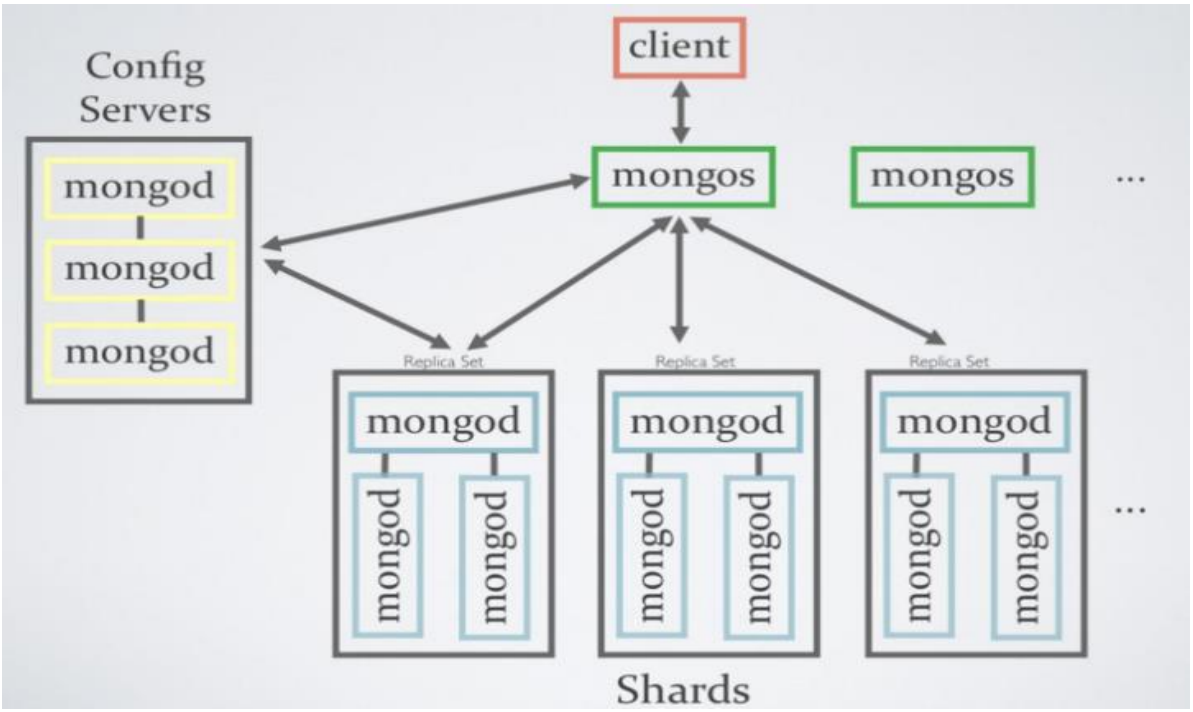
Η MongoDB παρέχει οριζόντια κλιμάκωση χρησιμοποιώντας μια τεχνική που ονομάζεται sharding, η οποία είναι διαφανής στις εφαρμογές. Το sharding διανέμει τα δεδομένα σε πολλαπλές φυσικές κατατμήσεις που ονομάζονται shards. Το sharding επιτρέπει στα συστήματα MongoDB να αντιμετωπίσουν τους περιορισμούς υλικού ενός μοναδικού εξυπηρετητή, όπως τα bottlenecks στη RAM ή στο I/O των δίσκων, χωρίς να προστίθεται επιπλέον πολυπλοκότητα στην εφαρμογή.
Το sharding είναι διαφανές στις εφαρμογές. Είτε υπάρχει ένα είτε εκατό shards, ο κώδικας της εφαρμογής για τα ερωτήματα στη MongoDB είναι ο ίδιος. Οι εφαρμογές στελνουν ερωτήματα σε έναν δρομολογητή ερωτημάτων που στέλνει τα ερωτήματα στα κατάλληλα shards.

Η συνέπεια και η ανθεκτικότητα στη MongoDB

Μοντέλο συναλλαγών

Η MongoDB είναι συμβατή με το ACID στο επίπεδο του εγγράφου. ‘Ενα ή περισσότερα πεδία μπορεί να γραφούν σε μια λειτουργία, συμπεριλαμβάνοντας ενημερώσεις σε πολλαπλά υπο-έγγραφα και στοιχεία ενός πίνακα. Οι ACID εγγυήσεις που παρέχονται από τη MongoDB διασφαλίζουν πλήρη απομόνωση καθώς ένα έγγραφο ενημερώνεται. Οποιοδήποτε σφάλμα προκαλεί τη λειτουργία να αντιστραφεί και έτσι υπάρχει πάντα μια συνεπής όψη του εγγράφου.

Οι προγραμματιστές μπορούν να χρησιμοποιήσουν τα Write Concerns της MongoDB για να ρυθμίσουν τις λειτουργίες έτσι ώστε να επιβεβαιώνονται στην εφαρμογή μόνο αφού έχουν περαστεί σε ένα αρχείο journal στο δίσκο. Αυτό είναι το ίδιο μοντέλο που χρησιμοποιείται από πολλές παραδοσιακές σχεσιακές βάσεις δεδομένων για παρέχονται εγγυήσεις ανθεκτικότητας. Ως ένα κατανεμημένο σύστημα, η MongoDB παρουσιάζει επιπρόσθετη ευελιξία στο να δίνει στους χρήστες τη δυνατότητα να επιτύχουν τους επιθυμητους στόχους ανθεκτικότητας με το να ελέγχουν πως οι λειτουργίες εγγραφής περνούνται σε όλα τα αντίγραφα.



Εικόνα 3.8 – Sharding στη MongoDB

Σύνολα αντιγράφων

Η MongoDB διατηρεί πολλαπλά αντίγραφα των δεδομένων χρησιμοποιώντας εγγενή αντιγραφή. Ένα σύνολο αντιγράφων είναι ένα πλήρως αυτο-επιδιορθώμενο shard το οποίο βοηθάει να αποτραπεί το η μη διαθεσιμότητα της βάσης. Η μετάβαση σε εφεδρικά αντίγραφα είναι πλήρως αυτοματοποιημένη, εξαλείφοντας την ανάγκη να υπάρχουν διαχειριστές οι οποίοι θα πρέπει να παρεμβαίνουν χειροκίνητα.

Ο αριθμός των αντιγράφων σε ένα σύνολο αντιγράφων MongoDB είναι ρυθμιζόμενος, και ένας μεγαλύτερος αριθμός αντιγράφων παρέχει αυξημένη ανθεκτικότητα δεδομένων και προστασία από περιόδους μη διαθεσιμότητας της βάσης (π.χ. σε περίπτωση που πολλά μηχανήματα παρουσιάσουν βλάβη, βλάβη σε επίπεδο data center ή δικτύου). Προαιρετικά, μπορεί να ρυθμιστεί οι λειτουργίες να κάνουν εγγραφές σε πολλαπλά αντίγραφα πριν να γίνει η επιστροφή στην εφαρμογή και ως εκ τούτου να παρέχεται μια λειτουργικότητα παρόμοια με τη σύγχρονη αντιγραφή.

Τα σύνολα αντιγράφων επίσης παρέχουν λειτουργική ευελιξία προσφέροντας έναν τρόπο να αναβαθμιστεί το υλικό και το λογισμικό χωρίς να απαιτείται να σταματήσει η διαθεσιμότητα της βάσης.

Απόδοση εντός μνήμης με χωρητικότητα δίσκου

Η MongoDB κάνει εκτενή χρήση της RAM για να επιταχύνει τις λειτουργίες της βάσης δεδομένων. Η ανάγνωση δεδομένων από τη μνήμη μετράται σε νανοσεκόντ, ενώ η ανάγνωση από τον δίσκο μετράται σε μιλισεκόντ. Η ανάγνωση από τη μνήμη είναι περίπου 100,000 φορές γρηγορότερη από την ανάγνωση από τον δίσκο. Στη MongoDB όλα τα δεδομένα διαβάζονται και διαχειρίζονται μέσω αρχείων αντιστοιχιζόμενων στη μνήμη. Τα δεδομένα που δεν προσπελαύνονται δεν φορτώνονται στη RAM. Ενώ δεν απαιτείται όλα τα δεδομένα να χωράνε στη RAM, ο στόχος της ομάδας ανάπτυξης θα πρέπει να είναι τα δεδομένα που προσπελαύνονται συχνά να χωράνε στη RAM.
Για παράδειγμα, μπορεί σε μια περίπτωση ένα μέρος ολόκληρης της βάσης να είναι το πιο συχνά προσπελάσιμο από την εφαρμογή, όπως δεδομένα που σχετίζονται με πρόσφατα γεγονότα ή δημοφιλή προϊόντα. Αν ο όγκος των δεδομένων που προσπελαύνονται συχνά υπερβαίνει την χωρητικότητα ενός μηχανήματος, η MongoDB μπορεί να κλιμακωθεί οριζόντια κατά μήκος πολλαπλών εξυπηρετητών χρησιμοποιώντας αυτόματο sharding. Επειδή η MongoDB παρέχει απόδοση εντός μνήμης για τις περισσότερες εφαρμογές δεν υπάρχει ανάγκη για ξεχωριστό στρώμα caching.

3.4 Chef

Σύντομη επισκόπηση

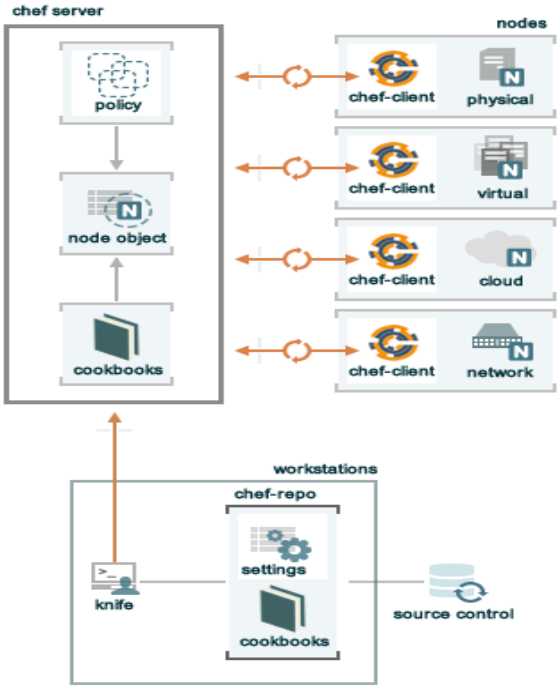
Το Chef είναι μια ισχυρή πλατφόρμα αυτοματοποίησης η οποία μετατρέπει την πολύπλοκη υποδομή σε κώδικα, ζωντανεύοντας τους εξυπηρετητές και τις υπηρεσίες των χρηστών. Ανεξάρτητα από το αν χρησιμοποιούνται cloud υπηρεσίες, τοπική υποδομή ή κάποια υβριδική επιλογή, το Chef αυτοματοποιεί τον τρόπο που ρυθμίζονται, διατίθενται και διαχειρίζονται οι εφαρμογές μέσα από το δίκτυο, όσο μεγάλο και αν είναι.

Το Chef βασίζεται σε απλές ιδέες: την επίτευξη της επιθυμητής κατάστασης, την συγκεντρωτική μοντελοποίηση της IT υποδομής και τους βασικούς πόρους που λειτουργούν ως δομικά στοιχεία. Αυτές οι ιδέες δίνουν τη δυνατότητα για γρήγορη διαχείριση οποιασδήποτε υποδομής με το Chef. Επίσης επιτρέπουν στο Chef να χειριστεί και τις πιο δύσκολες προκλήσεις σχετικά με τις υποδομές.

Οτιδήποτε μπορεί να τρέξει το chef-client μπορεί να το διαχειριστεί το Chef. Για παράδειγμα, είναι δυνατή η διαχείριση πραγματικών μηχανών, φυσικών μηχανών ή στιγμιοτύπων βασισμένων σε cloud. Το chef-client τρέχει σε ένα κόμβο και εκτελεί τις εργασίες που το ρυθμίζουν. Το Chef server είναι το κεντρικό αποθετήριο για όλα τα δεδομένα ρυθμίσεων. Το chef-client και το Chef server επικοινωνούν μεταξύ τους. Για ασφαλείς επικοινωνίες, χρησιμοποιούν έναν συνδυασμό δημόσιων και ιδιωτικών κλειδιών που διασφαλίζουν ότι το Chef server απαντά μόνο στα αιτήματα που γίνονται από το chef-client.

Τα τμήματα του Chef

Το ακόλουθο διάγραμμα δείχνει τις σχέσεις μεταξύ των διάφορων στοιχείων ενός πολύ απλού οργανισμού, συμπεριλαμβανομένου του Enterprise Chef server, των σταθμών εργασίας, του chef-repo και κάποιων απλών κόμβων που υπάρχουν είτε σε VirtualBox ή σε Amazon Web Services.



Εικόνα 3.9 – Επισκόπηση του Chef

Οι ακόλουθες ενότητες δίνουν λίγο περισσότερες λεπτομέρειες για αυτά τα στοιχεία.

Κόμβοι

Κόμβος είναι κάθε πραγματικό, εικονικό ή cloud μηχάνημα που ρυθμίζεται να συντηρείται από ένα chef-client.

Σταθμοί εργασίας

Σταθμός εργασίας είναι ένας υπολογιστής που ρυθμίζεται να τρέχει το Knife, για να συγχρονίζεται με το chef-repo και να αλληλεπιδρά με έναν Chef server. Ο σταθμός εργασίας είναι το μέρος από το οποίο οι περισσότεροι χρήστες θα κάνουν την περισσότερη δουλειά, συμπεριλαμβανομένων των παρακάτω:

- Ανάπτυξη cookbooks και recipes
- Διατήρηση του chef-repo σε συγχρονισμό χρησιμοποιώντας version source control
- Χρήση του Knife για μεταφόρτωση αντικειμένων από το chef-repo στον Chef server

- Ρύθμιση της πολιτικής του οργανισμού, συμπεριλαμβανομένου του ορισμού ρόλων και περιβάλλοντων and και της διασφάλισης ότι τα κρίσιμα δεδομένα αποθηκεύονται σε data bags
- Αλληλεπίδραση με τους κόμβους, όπως και όποτε απαιτείται, όπως η εκτέλεση μιας λειτουργίας bootstrap.

Knife

Το Knife είναι ένα εργαλείο γραμμής εντολών που παρέχει μια διεπαφή μεταξύ ενός τοπικού chef-repo και του Chef server. Το Knife βοηθάει τους χρήστες να διαχειριστούν:

- Κόμβους
- Cookbooks και recipes
- Ρόλους
- Αποθήκες JSON δεδομένων (data bags), συμπεριλαμβανομένων και κρυπτογραφημένων δεδομένων
- Περιβάλλοντα
- Cloud πόρους
- Την εγκατάσταση του chef-client στους διαχειριστικούς σταθμούς εργασίας
- Την αναζήτηση ευρετηριοποιημένων δεδομένων στον Chef server

Αποθετήριο

Το chef-repo είναι η δομή αποθετηρίου στην οποία τα cookbooks γράφονται, ελέγχονται και συντηρούνται. Τα cookbooks περιέχουν recipes - το πιο βασικό στοιχείο ρύθμισης μέσα στο Chef - γνωρίσματα, πόρους, παρόχους βιβλιοθήκες, αρχεία, πρότυπα κ.α. Το chef-repo πρέπει να είναι συγχρωνισμένο χρησιμοποιώντας ένα σύστημα version control, όπως το git και να υφίσταται διαχείριση αντίστοιχη με πηγαίο κώδικα.

Η δομή καταλόγου μέσα στο chef-repo ποικίλει. Κάποιοι οργανισμοί προτιμούν να έχουν όλα τα cookbooks τους σε ένα μόνο chef-repo, ενώ άλλοι οργανισμοί προτιμούν να χρησιμοποιούν ένα chef-repo για κάθε cookbook.

Το git είναι το πιο συχνά χρησιμοποιούμενο μέρος για την αποθήκευση ενός chef-repo που χρησιμοποιείται με ένα φιλοξενούμενο λογαριασμό Enterprise Chef, όμως το git δεν είναι απαραίτητο.

Ο φιλοξενούμενος εξυπηρετητής

Το Chef server λειτουργεί ως κεντρικό σημείο για τα δεδομένα ρυθμίσεων. Το Chef server αποθηκεύει cookbooks, τις πολιτικές που εφαρμόζονται στους κόμβους, και τα μεταδεδομένα που περιγράφουν κάθε καταγεγραμμένο κόμβο που διαχειρίζεται το chef-client. Οι κόμβοι χρησιμοποιούν το chef-client για να ζητήσουν από το Chef server λεπτομέρειες ρύθμισης, όπως recipes, πρότυπα και κατανομή αρχείων. Το chef-client τότε κάνει όσο περισσότερη γίνεται από τη ρύθμιση στους ίδιους τους κόμβους (και όχι στο Chef server). Αυτή η κλιμακώσιμη προσέγγιση κατανέμει την προσπάθεια για ρύθμιση σε όλο το τον οργανισμό.

Το φιλοξενούμενο Enterprise Chef είναι μια έκδοση του Chef server που φιλοξενείται από το Chef. Το φιλοξενούμενο Enterprise Chef βασίζεται στο cloud, είναι κλιμακώσιμο και διαθέσιμο (24x7/365) με έλεγχο πρόσβασης βασισμένο στους πόρους. Έχει τις ίδιες δυνατότητες αυτοματοποίησης με κάθε Chef server αλλά χωρίς να απαιτείται να στηθεί και να διαχειριστεί πίσω από το firewall.

Εικόνα 3.10 – Πελάτες και συνεργάτες του Chef

Cookbooks

Ένα cookbook είναι η δομική μονάδα ρύθμισης και κατανομής πολιτικών. Κάθε cookbook ορίζει ένα σενάριο, όπως όλα όσα χρειάζονται για την εγκατάσταση και τη ρύθμιση του MySQL, και τότε περιέχει όλα τα κομμάτια που απαιτούνται για να υποστηρίξουν το σενάριο, συμπεριλαμβανμένων των παρακάτω:

- Τιμές γνωρισμάτων που τίθενται στους κόμβους
- Ορισμοί που επιτρέπουν την δημιουργία επαναχρησιμοποιήσιμων συλλογών πόρων
- Διανομές αρχείων
- Βιβλιοθήκες που επεκτείνουν το chef-client και/ή παρέχουν βοήθειες στον κώδικα Ruby
- Recipes που προσδιορίζουν τους πόρους προς διαχείριση και τη σειρά στην οποία αυτή οι πόροι θα εφαρμοστούν
- Προσαρμοσμένοι πόροι και πάροχοι
- Πρότυπα
- Εκδόσεις
- Μεταδεδομένα για recipes (συμπεριλαμβανομένων εξαρτήσεων), περιορισμοί εκδόσεων, υποστηριζόμενες πλατφόρμες κ.α.

Το chef-client χρησιμοποιεί τη Ruby ως γλώσσα αναφοράς για δημιουργία cookbooks και ορισμό recipes, με μια εκτεταμένη DSL για εξειδικευμένους πόρους. Το chef-client παρέχει ένα λογικό σύνολο πόρων, αρκετό για να υποστηρίξει πολλά από τα πιο συχνά σενάρια αυτοματοποίησης υποδομής. Παρ'όλα αυτά, αυτή η DSL μπορεί επίσης να επεκταθεί όταν απαιτούνται επιπλέον πόροι και δυνατότητες.

Συμπέρασμα

Η κύρια λανθάνουσα αρχή του Chef είναι ότι ο χρήστης ξέρει καλύτερα σχετικά με το ποιό είναι το περιβάλλον, τι πρέπει αυτό να κάνει και πως πρέπει να συντηρηθεί. Το chef-client είναι σχεδιασμένο να μην κάνει υποθέσεις για κανένα από αυτά. Μόνο ο χρήστης και η ομάδα του κατανοούν τα τεχικά προβλήματα και τι απαιτείται για να λυθούν. Μόνο η ομάδα μπορεί να καταναοήσει τα ανθρώπινα προβλήματα που είναι μοναδικά σε κάθε οργανισμό και το αν καποια τεχνική λύση είναι βιώσιμη.

Η ιδέα ότι ο χρήστης ξέρει καλύτερα τι πρέπει να συμβεί μέσα στον οργανισμό συνδυάζεται με την αντίληψη του ότι σε κάθε περίπτωση χρειάζεται για να καταφέρει να εξασφαλίσει την ορθή λειτουργία. Είναι σπάνιο ένα άτομο να ξέρει τα πάντα για ένα πολύπλοκο πρόβλημα, πόσο μάλλον να ξέρει όλα τα βήματα που μπορεί να χρειάζονται για να λυθεί. Το ίδιο ισχύει και με τα εργαλεία. Το Chef προσφέρει βοήθεια με την διαχείριση της υποδομής και μπορεί να βοηθήσει στην επίλυση πολύ πολύπλοκων προβλημάτων. Το Chef έχει επίσης μεγάλη κοινότητα χρηστών που έχουν πολλή εμπειρία στην επίλυση πολύ πολύπλοκων προβλημάτων. Η κοινότητα μπορεί να παρέχει γνώση και υποστήριξη σε τομείς που ένας οργανισμός μπορεί να μην έχει και μπορεί, μαζί με το Chef, να τον βοηθήσει να λύσει κάθε πρόβλημα.

4
Ανάλυση Απαιτήσεων και Σχεδίαση

4.1 Σκοπός

Οι πάροχοι υπηρεσιών cloud computing, και πιο συγκεκριμένα οι πάροχοι υπηρεσιών τύπου IaaS, παρέχουν (συνήθως) τρείς διαφορετικούς τρόπους χρήσης των υπηρεσιών τους: γραφικό περιβάλλον διαθέσιμο μέσω διαδικτύου, πρόγραμμα πελάτη γραμμής εντολών και REST API. Σε κάποιες περιπτώσεις, οι τρείς παραπάνω τρόποι χρήσης των υπηρεσιών δεν προσφέρουν συνεπή εικόνα στους χρήστες, με την έννοια του ότι δεν παρέχονται και από τους τρείς όλες/οι ίδιες δυνατότητες. Αυτό μπορεί να γίνεται για διάφορους λόγους, όπως η ασφάλεια των λογαριασμών των χρηστών (κάποιες μέθοδοι είναι πιο εύκολα παραβιάσιμες) ή η διευκόλυνση των χρηστών (π.χ. πιο εύκολη η χρήση ενός καλοσχεδιασμένου γραφικού περιβάλλοντος παρά ενός κρυπτικού προγράμματος πελάτη γραμμής εντολών).

Iaas Provider	amazon web services	Windows Azure	Google Compute Engine	rackspace the open cloud company	IBM SmartCloud Enterprise	hp Converged Infrastructure
Key Features	Rich set of services and integrated monitoring tools; competitive pricing model. AWS can also be used as a PaaS.	Easy-to-use administration tool, especially for Windows admins. Windows Azure can also be used as a PaaS.	With the Google infrastructure backing it up, this IaaS is designed to scale.	Easy to use control panel, especially for non-system administrators, and strong customer service.	A good combination of management, software and security features for enterprise cloud administrators.	A good solution for enterprises wanting to integrate their existing IT infrastructure with public cloud services and investing in a hybrid cloud.
Limitations	AWS is a complex mixture of services. As your workflows become more complex and you use more services it can be difficult to project expenses. However, Amazon offers a monthly calculator to help estimate your costs.	Minimal, easy-to-use portal interface may not be so appealing to command line gurus.	Lacks ease of administration features. Running Hadoop on Google Compute Engine, for example, requires more from users; because it's not integrated you have to download the Hadoop package, a patch for Hadoop and a set of JDK packages along with several other steps (outlined here) to deploy a Hadoop cluster.	No messaging or specialized services (like Amazon Simple Queue Service and DynamoDB), although there are alternatives (like RabbitMQ and MongoDB or CouchDB) that you can run, you'll just need to manage them yourself.	May find difficulty distinguishing itself from other OpenStack providers, at least among non-IBM customers.	HP is still relatively new in the IaaS space with a limited track record and feature set compared to more seasoned providers.
Pricing	Instances range from $0.113/hour to $6.82/hour, with volume discounts available for reserved instances. Storage prices range from $0.095/GB/month to $0.125/GB/month. Additional charges for application services and data egress may apply.	m $0.02 to $1.60 per hour. Storage prices range from $0.07/GB/month to $0.12/GB/month, depending on level of redundancy.	Instances range from $0.019/hour to $1.659/hour. Provisioned storage is $0.04/GB/month; snapshot storage is $0.125/GB/month.	Instances start at $0.04/hour and go up to $5.44/hour. File storage starts at $0.10/GB/month and block storage is $0.12/GB/month.	Hourly and monthly pricing available, however numbers are not disclosed. Contact IBM for details.	Instances range from $0.03/hour to $3.40/hour. Block storage is $0.10/GB/month while object storage costs $0.09/GB/month.
Bonus	New users can get 750 hours, 30GB storage and 15GB bandwidth for free with AWS's Free Usage Tier.	Free 30-day trial with a limit of up to $200 is available for new users.	Google charges by the minute after a minimum of 10 minutes in an hour.	Rackspace is currently offering a $100 credit on your first month bill.	Free cloud server for one month from IBM's SoftLayer.	Free 90-day trial with a $100 credit for each of your first three HP Public Cloud monthly invoices.

Εικόνα 4.1 - Σύγκριση παρόχων υπηρεσιών Cloud Computing

Επίσης, είναι σύνηθες πλέον οι χρήστες, είτε εταιρείες είτε ιδιώτες, να διατηρούν λογαριασμούς σε περισσότερους από έναν παρόχους υπηρεσιών cloud computing. Αυτό γίνεται γιατί έτσι τους δίνεται η δυνατότητα να εκμεταλλεύονται τα καλύτερα χαρακτηριστικά που διαφέρουν μεταξύ των παρόχων, για καλύτερη ικανοποίηση των αναγκών τους. Άλλοι προσφέρουν πιο συμφέρουσα τιμολογιακή πολιτική, καλύτερη κλιμάκωση (οριζόντια ή κατακόρυφη), καλύτερη και πιο πλήρη υποστήριξη κ.α. Αυτό όμως προσθέτει αρκετή πολυπλοκότητα στη διαχείριση καθώς ανάλογα με τον πάροχο αλλάζει ο τρόπος που επιτελούνται οι βασικές λειτουργίες πάνω στις υπηρεσίες που παρέχει.

Στις περισσότερες περιπτώσεις, οι εικονικές μηχανές που δημιουργούνται στις υποδομές των παρόχων πρόκειται να χρησιμοποιηθούν για κάποιο συγκεκριμένο ρόλο, όπως για παράδειγμα web server, database server κτλ. Αυτό συνήθως προϋποθέτει την εγκατάσταση συγκεκριμένου λογισμικού, ανάλογα με τον ρόλο, η εγκατάσταση του οποίου πρέπει να γίνει από τον χρήστη/διαχειριστή αφού ολοκληρωθεί η δημιουργία του εικονικού μηχανήματος. Ο πιο απλός τρόπος να γίνει αυτό είναι μέσω ssh, το οποίο σημαίνει αφενός τη στέρηση της ευκολίας ενός γραφικού περιβάλλοντος και αφετέρου υψηλή πολυπλοκότητα αφού θα πρέπει να εγκατασταθούν μια προς μια οι απαιτούμενες εφαρμογές και να επιλυθούν τα όποια ενδεχόμενα θα προκύψουν από αυτή τη διαδικασία.

Σε αυτή την εργασία γίνεται μια προσπάθεια για ανάπτυξη λογισμικού που έχει σκοπό να παρέχει μια ενοποιημένη εμπειρία χρήσης υπηρεσιών cloud computing. Για τον σκοπό αυτό χρησιμοποιείται το REST API που προσφέρουν οι πάροχοι, το οποίο δίνει τη δυνατότητα να δημιουργηθεί μια υποδομή που θα μπορεί να επεκταθεί και να υποστηρίξει τις υπηρεσίες κάθε παρόχου. Επι της ουσίας, πρόκειται για την δημιουργία ενός διαμεσολαβητή μεταξύ των REST API των παρόχων και ενός συνεπούς και ομοιογενούς γραφικού περιβάλλοντος διαχείρισης. Επιπλέον, γίνεται και μια προσπάθεια αυτοματοποίησης της εγκατάστασης συγκεκριμένων εφαρμογών επιλογής του χρήστη ώστε να αποφευχθεί η προαναφερθείσα δυσκολία στην χειροκίνητη εγκατάστασή τους. Λόγω της μεγάλης πολυπλοκότητας, αρχικός στόχος είναι να παρέχονται οι βασικές δυνατότητες και όχι οι πιο εξεζητημένες έτσι ώστε να δοθεί βάση και στην εξασφάλιση ορισμένων παθητικών χαρακτηριστικών υψηλής σημασίας.

Στη συνέχεια γίνεται μια ανάλυση του προσανατολισμού του λογισμικού όσον αφορά τις λειτουργικές αλλά και τις μη λειτουργικές απαιτήσεις.

4.2 Λειτουργικές Απαιτήσεις

Στα πλαίσια της ανάλυσης των λειτουργικών απαιτήσεων μπορεί να γίνει διαχωρισμός τους ως προς την οντότητα που αφορούν. Εδώ οντότητες θεωρούνται οι χρήστες, οι λογαριασμοί σε κάποιο πάροχο υπηρεσιών cloud computing και οι εικονικές μηχανές. Σημειώνεται ότι τα παρακάτω συνιστούν ένα θεμελιώδες σύνολο λειτουργιών ικανό να εξασφαλίσει ένα βασικό επίπεδο λειτουργικότητας.

Χρήστες

Οι χρήστες είναι η βασική οντότητα. Αποτελούν το σημείο εισόδου στην εφαρμογή υπό την έννοια ότι για να γίνει οτιδήποτε άλλο πρέπει ο χρήστης πρώτα να εγγραφεί έτσι ώστε να υπάρχει ένα σημείο αναφοράς. Οι ενέργειες που αφορούν τους χρήστες είναι οι παρακάτω:

- Εγγραφή

Όπως αναφέρθηκε, οι χρήστες πρέπει να μπορούν να εγγραφούν για να υπάρχει αποθήκευση της δραστηριότητάς τους. Για την εγγραφή, απαιτούνται ένα όνομα, ένα email και ένα password.

Το password, για λόγους εξασφάλισης ενός ελάχιστου επιπέδου ασφάλειας απαιτείται να έχει μήκος 6 ή περισσότερους χαρακτήρες.

- Σύνδεση

Οι χρήστες πρέπει να μπορούν να συνδεθούν όταν επιθυμούν να χρησιμοποιήσουν την υπηρεσία μετά από καιρό. Με αυτόν τον τρόπο έχουν πρόσβαση στους λογαριασμούς των παρόχων που έχουν αποθηκεύσει καθώς και στα αντίστοιχα εικονικά μηχανήματα. Για την σύνδεση απαιτείται το email και το password.

- Αποσύνδεση

Οι χρήστες πρέπει να μπορούν να αποσυνδεθούν. Όταν ολοκληρώσουν την εργασία τους έχουν τη δυνατότητα να κλείσουν τη συνεδρία (session) έτσι ώστε να αποφευχθεί μη εξουσιοδοτημένη πρόσβαση. Αυτό είναι ιδιαίτερα χρήσιμο για περιβάλλοντα στα οποία ένας υπολογιστής μπορεί να χρησιμοποιηθεί από περισσότερα από ένα άτομα.

Λογαριασμοί

Οι λογαριασμοί ως οντότητα αναφέρονται σε λογαριασμούς στις υπηρεσίες διάφορων παρόχων. Κάθε λογαριασμός ανήκει σε έναν χρήστη και ανάλογα με το σε ποιόν πάροχο αντιστοιχεί μπορεί να υπάρχουν μικρο-διαφορές στον τρόπο που αντιμετωπίζεται από την υπηρεσία όμως σε κάθε περίπτωση αυτό δεν γίνεται αντιληπτό από τον χρήστη. Οι ενέργειες που αφορούν τους λογαριασμούς είναι οι παρακάτω:

- Δημιουργία

Για κάθε χρήστη θα πρέπει να υπάρχει η δυνατότητα αποθήκευσης ενός λογαριασμού σε πάροχο υπηρεσιών cloud computing που του ανήκει. Για κάθε λογαριασμό που προστίθεται απαιτείται το username που συσχετίζεται με αυτόν τον λογαριασμό, το όνομα του παρόχου και το token. Το token είναι μια συμβολοσειρά που χρησιμοποιείται ως μέτρο ασφαλείας κατά αντιστοιχία με ένα password κατά τη χρήση του REST API.

- Καταστροφή

Για κάθε λογαριασμό που έχει προστεθεί θα πρέπει να υπάρχει η δυνατότητα διαγραφής του από το σύστημα. Αυτό έχει ως αποτέλεσμα και τη διαγραφή των αντίστοιχων εικονικών μηχανών από το σύστημα αλλά όχι την καταστροφή τους.

- Επεξεργασία

Για κάθε λογαριασμό που έχει προστεθεί θα πρέπει να υπάρχει η δυνατότητα επεξεργασίας των στοιχείων του. Πέρα από την τροποποίηση του username, η οποία είναι μικρής σημασίας στην παρούσα φάση, είναι πολύ σημαντική η τροποποίηση του token. Τα token έχουν συγκεκριμένη χρονική διάρκεια ισχύος και έπειτα εκδίδονται καινούργια. Έτσι, αυτή η λειτουργία είναι πολύ

σημαντική γιατί θα πρέπει ο χρήστης να ανανεώσει χειροκίνητα μέσω αυτής το token για να μπορεί να συνεχίσει να χειρίζεται τον κάθε λογαριασμό. Ο λόγος που αυτό δεν γίνεται αυτόματα θα αναφερθεί στη συνέχεια.

Εικονικές μηχανές

Οι εικονικές μηχανές ως οντότητα αναφέρονται στις εικονικές μηχανές που μπορεί να δημιουργήσει ο χρήστης σε κάθε λογαριασμό του σε κάποιον πάροχο. Επίσης για κάθε μια από τις υποστηριζόμενες λειτουργίες για τις εικονικές μηχανές προϋποτίθεται ότι για να ολοκληρωθούν θα πρέπει να επιτρέπεται από τον αντίστοιχο πάροχο. Οι ενέργειες που αφορούν τις εικονικές μηχανές είναι οι παρακάτω:

- Δημιουργία

Θα πρέπει να δίνεται η δυνατότητα δημιουργίας μιας εικονικής μηχανής. Αυτό περιλαμβάνει την καταγραφή της στο σύστημα αλλά και την αίτηση δημιουργίας της στον πάροχο. Ανάλογα με το τι επιτρέπει ο κάθε πάροχος οι εικονικές μηχανές έχουν διαφορετικές ποικιλίες αριθμού επεξεργαστών, μεγέθους μνήμης, μεγέθους δίσκου, λειτουργικού συστήματος και IP διευθύνσεων και παρουσιάζονται στον χρήστη για να επιλέξει. Επίσης ο χρήστης πρέπει να παρέχει και ένα όνομα για την εικονική μηχανή.

- Καταστροφή

Για κάθε εικονική μηχανή που έχει δημιουργηθεί μέσα από το σύστημα πρέπει να υπάρχει η δυνατότητα καταστροφής, δηλαδή διαγραφή της από το σύστημα και αίτηση καταστροφής της στον πάροχο.

- Εκκίνηση

Για κάθε εικονική μηχανή που έχει δημιουργηθεί μέσα από το σύστημα πρέπει να υπάρχει η δυνατότητα εκκίνησής της, αν αυτή έχει τερματίσει την λειτουργία της.

- Τερματισμός

Για κάθε εικονική μηχανή που έχει δημιουργηθεί μέσα από το σύστημα πρέπει να υπάρχει η δυνατότητα τερματισμού της, αν αυτή έχει εκκινήσει την λειτουργία της.

- Τερματικό

Για κάθε εικονική μηχανή που έχει δημιουργηθεί μέσα από το σύστημα πρέπει να υπάρχει η δυνατότητα παροχής εικονικού τερματικού μέσω ssh με σκοπό τον άμεσο χειρισμό της. Ο χρήστης έχει τη δυνατότητα να εκτελέσει σχεδόν οποιαδήποτε εντολή θέλει σε αυτό το τερματικό όμως λόγω περιορισμών δεν υπάρχει αμεσότητα πραγματικού χρόνου και μπορεί να δει τα αποτελέσματα μόνο αφού ολοκληρωθεί η εκτέλεση.

- Κατάσταση

Για κάθε εικονική μηχανή που έχει δημιουργηθεί μέσα από το σύστημα πρέπει να υπάρχει ανά πάσα στιγμή ενημερωμένη καταγραφή της κατάστασής της, δηλαδή αν βρίσκεται σε λειτουργία ή όχι, σε κατάσταση δημιουργίας ή διαγραφής και οποιαδήποτε άλλη προβλέπει ο πάροχος. Αυτή τη λειτουργία ουσιαστικά την επωμίζονται όλες οι άλλες οι οποίες φροντίζουν να ενημερώνουν την κατάσταση κάθε φορά που αλλάζει υπ'ευθύνη τους.

4.3 Μη Λειτουργικές Απαιτήσεις

Σε αυτή την ενότητα παρουσιάζονται οι μη λειτουργικές απαιτήσεις του συστήματος. Οι απαιτήσεις αυτές διέπουν το σύστημα σε όλη τη λειτουργία του και όντας βασικά κομμάτια της τελικής εμπειρίας χρήσης δεν παραβιάζονται σε καμία περίπτωση.

Μοναδικότητα χρηστών

Κάθε χρήστης πρέπει να είναι μοναδικός. Αυτό το χαρακτηριστικό αποδίδεται στο email που δίνει ο χρήστης κατά την εγγραφή του, το οποίο ελέγχεται για τη μοναδικότητά του στη βάση δεδομένων του συστήματος.

Αποκρισιμοτητα

Το σύστημα πρέπει να έχει υψηλή αποκρισιμότητα. Πρέπει η εμπειρία που προσφέρεται στους χρήστες να τους κάνει να νιώθουν ότι το σύστημα αποκρίνεται σε πραγματικό χρόνο και ότι δεν "κολλάει". Στην προκειμένη περίπτωση, αυτό είναι μια μεγάλη πρόκληση διότι η λειτουργία του συστήματος εξαρτάται ως επί το πλείστον από ενέργειες που αφενός χρειάζονται αρκετό χρόνο για να ολοκληρωθούν και αφετέρου εκτελούνται στους εξυπηρετητές του παρόχου και έτσι δεν είναι άμεσα παρατηρήσιμες. Θα πρέπει βέβαια να σημειωθεί πως σε αυτές τις περιπτώσεις ως αποκρισιμότητα δεν υπονοείται η ταχεία ολοκλήρωση μιας ενέργειας (κάτι τέτοιο δεν γίνεται) αλλά το να μην περιμένει όλο το σύστημα παγωμένο την ολοκλήρωσή της. Σε επόμενο κεφάλαιο αναλύεται πως επιτυγχάνεται αυτή η απαίτηση παρά τις δυσκολίες.

Συνέπεια δεδομένων

Τα δεδομένα της βάσης δεδομένων του συστήματος πρέπει να είναι συνεπή με αυτά στις βάσεις δεδομένων των παρόχων. Πιο απλά η βάση του συστήματος θα πρέπει να είναι ενημερωμένη για τα εικονικά μηχανήματα που έχουν δημιουργηθεί ή διαγραφεί και για την κατάστασή τους (αν είναι ενεργοποιημένα ή όχι κτλ). Στην παρούσα, αρχική, υλοποίηση του συστήματος για να επιτευχθεί αυτό θα πρέπει οι όποιο λογαριασμοί παρόχων συνδεθούν στο σύστημα να μην

έχουν καθόλου εικονικά μηχανήματα από πριν ενώ θα πρέπει και όλες οι καταστροφές εικονικών μηχανημάτων να γίνονται μέσω του συστήματος και όχι εκτός αυτού.

Ασφάλεια

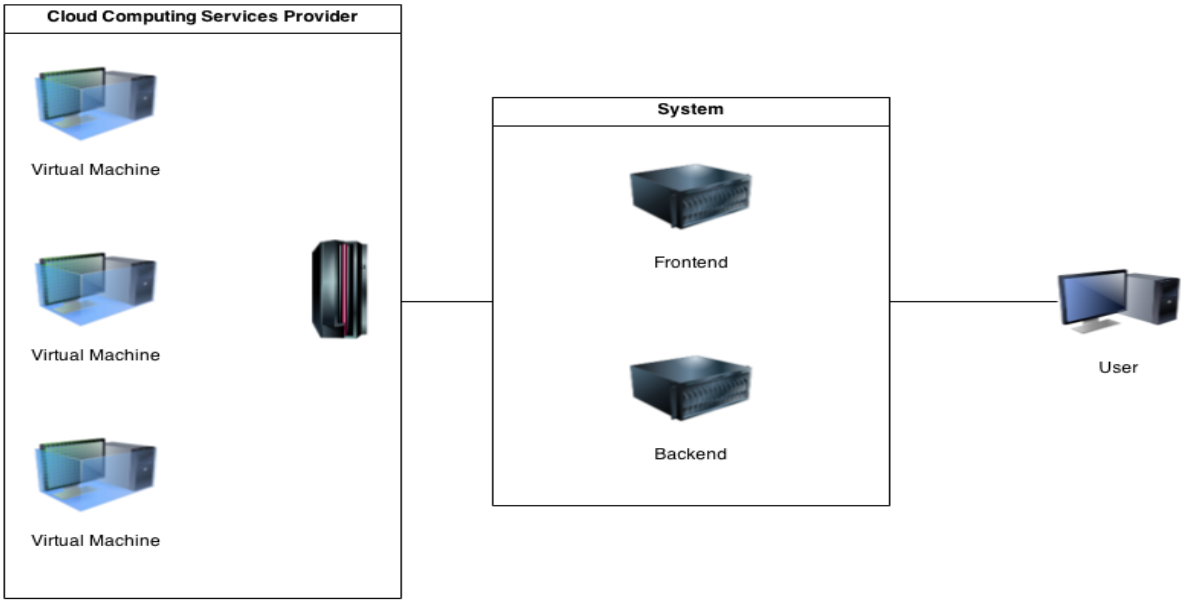
Οι λογαριασμοί των χρηστών στο σύστημα (όχι στους παρόχους) θα πρέπει να είναι ασφαλείς. Για αυτό τον σκοπό χρησιμοποιείται το password που ζητείται από τον χρήστη κατά την εγγραφή του. Όμως και εκεί χρειάζεται επιπλέον προσοχή όσον αφορά τον τρόπο που αυτό αποθηκεύεται αλλά και τον τρόπο που ταυτοποιείται κατά την είσοδο του χρήστη.

Ευχρηστία

Το γραφικό περιβάλλον θα πρέπει να είναι εύχρηστο. Αυτό σημαίνει ότι οι χρήστες θα πρέπει να μπορούν να πλοηγηθούν εύκολα και να μπορούν άμεσα να επιτύχουν το αποτέλεσμα που θέλουν. Θα πρέπει να υπάρχουν απλά μενού και επεξηγηματικές επιγραφές στα κουμπιά και τους συνδέσμους καθώς επίσης και εύστοχες επιλογές συνδέσμων κατεύθυνσης έτσι ώστε να μην χάνονται μέσα στο γραφικό περιβάλλον.

4.4 Αρχιτεκτονική

Σε αυτή την ενότητα περιγράφεται η αρχιτεκτονική του συστήματος. Η περιγραφή φτάνει μέχρι και την παρουσίαση συγκεκριμένων πρωτοκόλλων και τεχνολογιών που επιλέχθηκαν χωρίς να γίνεται αναφορά σε εργαλεία λογισμικού.



Εικόνα 4.2 - Σχεδιάγραμμα συστήματος

Το σύστημα αποτελείται από δύο κύρια τμήματα: το frontend και το backend. Αυτά τα τμήματα έχουν σαφώς καθορισμένες ευθύνες και είναι αυστηρά διαχωρισμένα μεταξύ τους καθώς (πρέπει να) μπορούν να λειτουργήσουν σε διαφορετικά μηχανήματα. Για αυτόν τον λόγο έχει δοθεί ιδιαίτερη σημασία και στον σχεδιασμό της μεταξύ τους επικοινωνίας έτσι ώστε να εξασφαλίζεται η συνέπεια των δεδομένων και να επιτυγχάνεται υψηλή απόδοση.

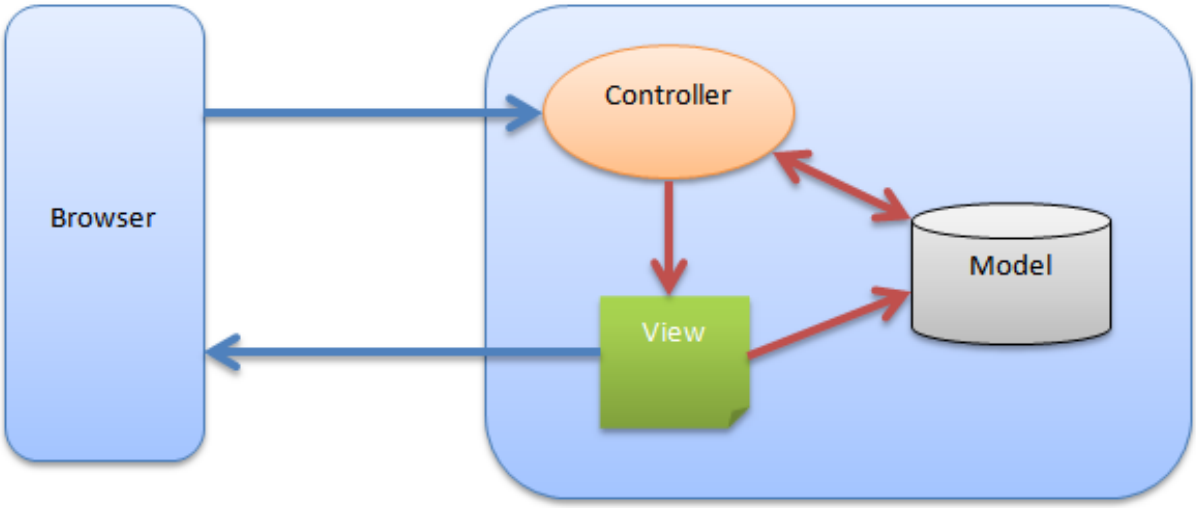
Frontend

- Αρμοδιότητες

Αυτό το τμήμα είναι το πρώτο κομμάτι του συστήματος. Κύριες αρμοδιότητές του είναι η αλληλεπίδραση με τον χρήστη, η διαχείριση της βάσης δεδομένων καθως και η διεκπεραίωση βασικών εργασιών που δεν απαιτούν πολύ χρόνο. Ουσιαστικά αποτελεί τον πυρήνα του συστήματος καθώς εχει την ευθύνη για όλες τις βασικές λειτουργίες ενώ το δεύτερο κομμάτι, το backend, έχει πολύ πιο συγκεκριμένο εύρος ευθύνης, όπως περιγράφεται στην συνέχεια.

- Αρχιτεκτονική

Η οργάνωση του frontend έγινε σύμφωνα με το αρχιτεκτονικό μοτίβο Model-View-Controller (MVC). Όπως και άλλα μοτίβα, το MVC εκφράζει τον πυρήνα της λύσης ενός προβλήματος ενώ παράλληλα επιτρέπει την προσαρμογή ανάλογα με το σύστημα. Το κεντρικό κομμάτι του MVC είναι το μοντέλο (model) το οποίο αποτυπώνει την συμπεριφορά της εφαρμογής σε σχέση με το πεδίο του προβλήματος που καλείται να λύσει, ανεξάρτητα από την αλληλεπίδραση με τον χρήστη. Το μοντέλο διαχειρίζεται απευθείας τα δεδομένα, τη λογική και τους κανόνες της εφαρμογής. Μια όψη (view) μπορεί να είναι οποιαδήποτε έξοδος αναπαράστασης πληροφορίας, όπως ένα γράφημα ή ένα διάγραμμα. Το τρίτο κομμάτι, ο ελεγκτής (controller), δέχεται είσοδο και τη μετατρέπει σε εντολές για το μοντέλο ή την όψη.



Εικόνα 4.3 - Αρχιτεκτονική frontend (MVC)

Πέρα από τη διαίρεση της εφαρμογής σε τρία κομμάτια, το MVC ορίζει τις αλληλεπιδράσεις μεταξύ τους. Ένας ελεγκτής μπορεί να στείλει εντολές στο μοντέλο για να ενημερώσει την κατάστασή του. Μπορεί επίσης να στείλει εντολές στην αντίστοιχή του όψη για να αλλάξει το παρουσιάζει η όψη το μοντέλο. Ένα μοντέλο ειδοποιεί τις όψεις και τους ελεγκτές που του αντιστοιχούν όταν γίνεται κάποια αλλαγή στην κατάστασή του. Αυτή η ειδοποίηση επιτρέπει στις όψεις να παράγουν ενημερωμένα αποτελέσματα και στους ελεγκτές να αλλάξουν διαθέσιμο σύνολο εντολών. Σε κάποιες περιπτώσεις μια υλοποίηση του MVC μπορεί αντίθετα να είναι παθητική, με την έννοια ότι θα πρέπει τα άλλα κομμάτια να στέλνουν ερωτήσεις στο μοντέλο για να μάθουν αν άλλαξε η κατάστασή του αντί να περιμένουν να ειδοποιηθούν. Μια όψη ζητά πληροφορίες από το μοντέλο για να παράγει μια αναπαράστασή του για τον χρήστη.

Backend

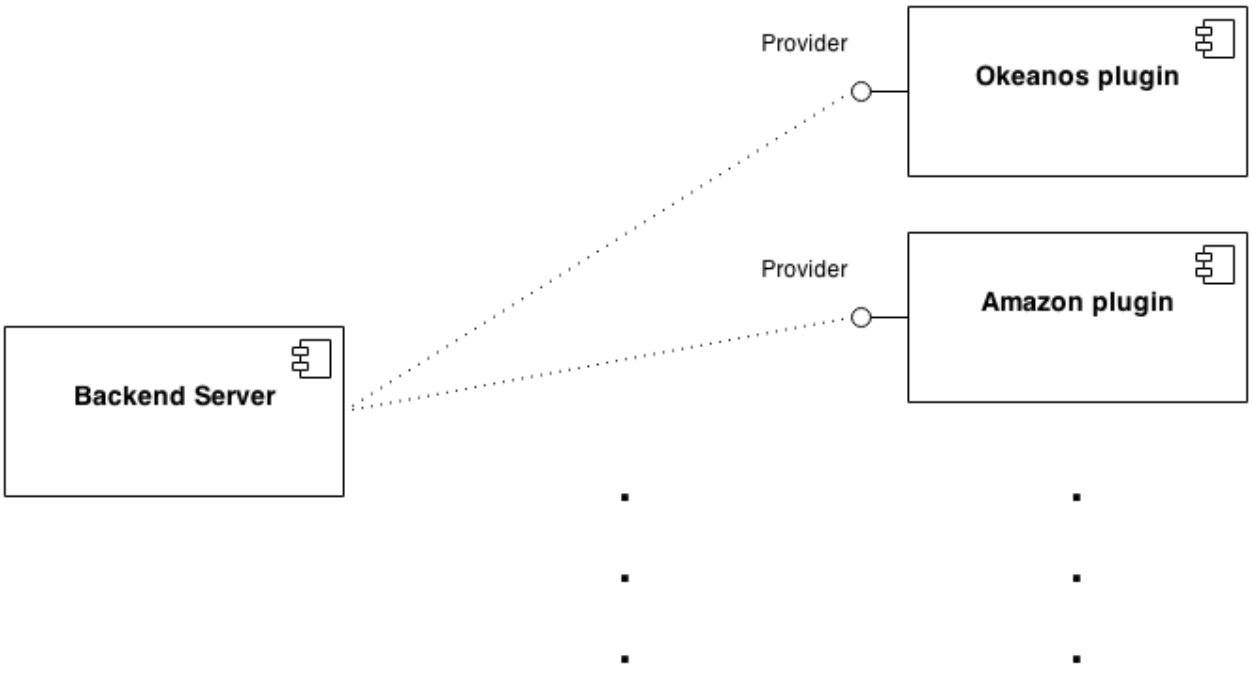
- Αρμοδιότητες

Αυτό το τμήμα είναι το δεύτερο κομμάτι του συστήματος. Οι αρμοδιότητες που του ανατίθενται είναι οι λειτουργίες που έχουν μεγάλους χρόνους αναμονής και αν αποδίδονταν στο frontend θα δημιουργούσαν πρόβλημα στην αποκρισιμότητα του συνολικού συστήματος. Κύρια αρμοδιότητά του, λοιπόν, είναι η αλληλεπίδραση με τους παρόχους cloud υπηρεσιών. Αυτή είναι η βασική ευθύνη του και είναι αποκλειστική με την έννοια ότι κανένα άλλο μέρος του συστήματος δεν επικοινωνεί με τους παρόχους. Επί της ουσίας, έχει ρόλο διαμεσολαβητή μεταξύ του frontend και των παρόχων. Το frontend ετοιμάζει ένα αίτημα προς έναν πάροχο, σύμφωνα πάντα με το τι ζητάει ο χρήστης, και το στέλνει στο backend ζητώντας του να το διεκπεραιώσει. Το backend προωθεί το αίτημα, περιμένει να ολοκληρωθεί και όταν συμβεί αυτό ενημερώνει το frontend για τα αποτελέσματα. Μια ακόμα ευθύνη του backend είναι το να φροντίσει για την εγκατάσταση των επιπλέων εφαρμογών που θα ζητήσει ο χρήστης κατά τη δημιουργία μιας εικονικής μηχανής. Αμέσως μόλις ολοκληρωθεί η δημιουργία μιας εικονικής μηχανής και ενημερωθεί το backend από τον πάροχο, πριν ειδοποιήσει το frontend για αυτό το γεγονός, επικοινωνεί με την μηχανή για να επιβλέψει την εγκατάσταση των επιλεχθέντων πακέτων λογισμικού.

- Αρχιτεκτονική

Το backend ακολουθεί αρχιτεκτονική βασισμένη στη χρήση plugins. Πρόκειται για ένα ευέλικτο και επεκτάσιμο μοτίβο το οποίο δίνει τη δυνατότητα στο σύστημα να υποστηρίξει σχεδόν όλους τους παρόχους με ένα ελάχιστο προγραμματιστικό κόστος, τη συγγραφή του αντίστοιχου plugin. Το backend αποτελείται από ένα βασικό κομμάτι στο οποίο προσαρτώνται τα προαναφερθέντα plugins τα οποία πρέπει να τηρούν ορισμένες σχεδιαστικές προϋποθέσεις για να χρησιμοποιηθούν σωστά. Πιο συγκεκριμένα ορίζεται μια διεπαφή μεταξύ αυτών και του βασικού μέρους στην οποία πρέπει να προσαρμόζονται έτσι ώστε το τελευταίο να μην επιβαρύνεται με επιμέρους λεπτομέρειες υλοποίησης και ιδιαιτερότητες που μπορεί να παρουσιάζουν τα REST APIs των παρόχων. Αυτό στην πραγματικότητα επιτυγχάνεται με την υλοποίηση συγκεκριμένων συναρτήσεων από την πλευρά του plugin τις οποίες αναμένει να βρει το βασικό τμήμα του backend. Οι συναρτήσεις αυτές αποτελούν προσαρμογή των δεδομένων με τέτοιο τρόπο ώστε

να υποβάλλονται έγκυρα αιτήματα στους παρόχους τα οποία αφορούν βασικές λειτουργίες όπως εκκίνηση ή τερματισμός της λειτουργίας μιας εικονικής μηχανής, δημιουργία ή καταστροφή της κτλ.



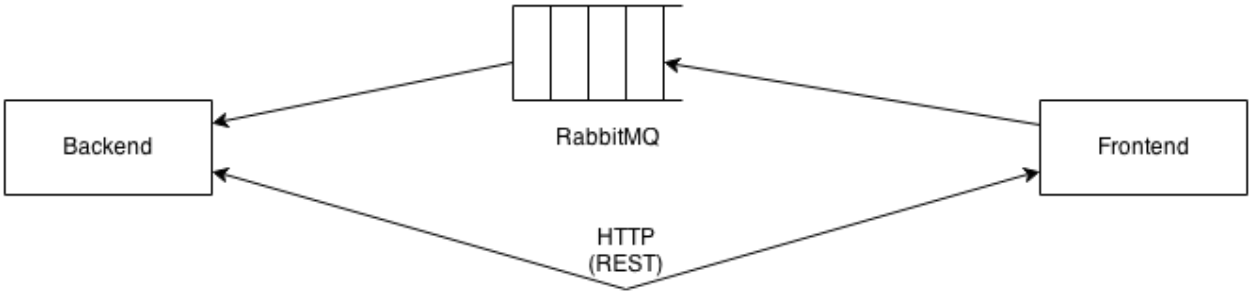
Εικόνα 4.4 - Αρχιτεκτονική backend (plugins)

Διασύνδεση - Επικοινωνία

Μετά την περιγραφή των αρμοδιοτήτων και των αρχιτεκτονικών των δύο τμημάτων του συστήματος, έχει ιδιαίτερη σημασία η περιγραφή του μοντέλου επικοινωνίας μεταξύ τους. Για το συγκεκριμένο σκοπό σχεδιάστηκε ένα πρότυπο δύο καναλιών. Το πρώτο κανάλι επικοινωνίας αποτελείται από μια ουρά μηνυμάτων που βασίζεται στο πρωτόκολλο AMQP (Advanced Message Queuing Protocol). Σε αυτή την ουρά το frontend βάζει μηνύματα και το backend τα παραλαμβάνει. Το δεύτερο κανάλι ουσιαστικά αφορά ένα REST API του συστήματος (όχι των παρόχων για τα οποία έχει γίνει αναφορά σε προηγούμενες ενότητες) το οποίο εξυπηρετεί την ανταλλαγή δεδομένων μεταξύ frontend και backend.

Τα δύο κανάλια λειτουργούν ως εξής. Όταν ένας χρήστης ζητάει μια ενέργεια πάνω σε μια εικονική μηχανή, το frontend στέλνει ένα μήνυμα στην ουρά. Μόλις το παραλάβει το backend, χρησιμοποιεί τις πληροφορίες που περιέχει για να πάρει μέω του REST API του συστήματος τα δεδομένα για την ενέργεια που ζήτησε ο χρήστης. Όταν η ενέργεια ολοκληρωθεί, το backend, χρησιμοποιώντας και πάλι το REST API του συστήματος, ενημερώνει το frontend για την έκβαση και τα αποτελέσματα της ενέργειας. Ενώ η παραπάνω διαδικασία θα μπορούσε να καλυφθεί εξολοκλήρου από το δεύτερο κανάλι, η ύπαρξη του πρώτου προσφέρει δυνατότητα κλιμακωσιμότητας, καθώς αν υπάρχει ανάγκη μπορούν να δημιουργηθούν πολλά στιγμιότυπα

του backend τα οποία θα παρακολουθούν την ουρά περιμένωντας για μηνύματα από το frontend.



Εικόνα 4.5 - Μοντέλο επικοινωνίας

4.5 Μοντέλο δεδομένων

Σε αυτή την ενότητα παρουσιάζεται το μοντέλο δεδομένων της βάσης δεδομένων του συστήματος. Η σχεδίαση αφορά NoSQL βάση δεδομένων και για αυτό γίνεται αναφορά σε συλλογές (collections) και έγγραφα (documents). Επίσης, γίνεται αξιοποίηση της δυνατότητας να ενσωματωθεί ένα έγγραφο σε ένα άλλο ως αντίστοιχη της δυνατότητας αναφοράς ενός πίνακα σε άλλον μέσω δευτερεύοντος κλειδιού (foreign key) στις σχεσιακές βάσεις δεδομένων. Ακολουθεί αναλυτική περιγραφή όλων των συλλογών που περιέχει η βάση. Σημειώνεται ότι σε κάποια μοντέλα υπάρχουν γνωρίσματα που δεν αναφέρονται παρακάτω γιατί δεν χρησιμοποιούνται στην παρούσα φάση και αφορούν μελλοντικές επεκτάσεις.

Χρηστες (Users)

Η συλλογή “Χρήστες” περιέχει έγγραφα που αφορούν κάθε χρήστη που έχει εγγραφεί στο σύστημα. Κάθε έγγραφο σε αυτή τη συλλογή περιέχει ως ενσωματωμένα έγγραφα που ανήκουν στη συλλογή “Λογαριασμοί”. Τα γνωρίσματα αυτής της συλλογής είναι τα παρακάτω.

- name

Το γνώρισμα αυτό αφορά το όνομα που επιλέγει ο χρήστης κατά την εγγραφή του.

- email

Το γνώρισμα αυτό αφορά το email που επιλέγει ο χρήστης κατά την εγγραφή του.

- password_digest

Το γνώρισμα αυτό αφορά το password που επιλέγει ο χρήστης κατά την εγγραφή του. Εδώ δεν αποθηκεύεται το ίδιο το password αλλά το αποτέλεσμα που προκύπτει όταν δοθεί ως είσοδος σε μια hash function.

- remember_token

Το γνώρισμα αυτό αφορά ένα token που δημιουργείται κατά την εγγραφή του χρήστη και χρησιμεύει στο να παρακολοθεί το σύστημα την πορεία του χρήστη καθώς πλοηγείται. Πρέπει να σημειωθεί ότι για λόγους ασφάλειας δεν αποθηκεύεται το ίδιο το token αλλά το αποτέλεσμα που προκύπτει όταν δοθεί ως είσοδος σε μια hash function.

Λογαριασμοί (Accounts)

Η συλλογή “Λογαριασμοί” περιέχει έγγραφα που αφορούν λογαριασμούς σε παρόχους υπηρεσιών cloud computing. Τέτοια έγγραφα δεν μπορούν να υπάρξουν αυτοφυώς αλλά μόνο ενσωματωμένα σε κάποιο έγγραφο από τη συλλογή “Χρήστες” το οποίο θα συμβολίζει και το σε ποιόν ανήκει αυτός ο λογαριασμός. Επίσης, τα έγγραφα αυτής της συλλογής περιέχουν έγγραφα από τη συλλογή “Εικονικές Μηχανές”.

- username

Το γνώρισμα αυτό αφορά το username που επιλέγει ο χρήστης κατά την προσθήκη του λογαριασμού.

- provider

Το γνώρισμα αυτό αφορά τον πάροχο στον οποίο αντιστοιχεί ο λογαριασμός.

- token

Το γνώρισμα αυτό αφορά το token που ορίζεται από τον πάροχο στον οποίο ανήκει ο λογαριασμός και πιστοποιεί την ταυτότητα του χρήστη κατά την χρήση του REST API.

Εικονικές Μηχανές (VMs)

Η συλλογή “Εικονικές Μηχανές” περιέχει έγγραφα που αφορούν εικονικές μηχανές που ανήκουν στον χρήστη. Τα έγγραφα αυτά δεν μπορούν να υπάρχουν ξεχωριστά αλλά μόνο ως ενσωματωμένα σε κάποιο έγγραφο που ανήκει στη συλλογή “Λογαριασμοί”. Αυτό συμβολίζει σε ποιόν λογαριασμό παρόχου ανήκουν αυτές οι εικονικές μηχανές.

- image

Το γνώρισμα αυτό αφορά το λειτουργικό σύστημα που είναι εγκατεστημένο στην εικονική μηχανή. Μπορεί να πρόκειται για μια βασική έκδοση είτε για κάποια customised από τον πάροχο εκδοχή κάποιου λειτουργικού συστήματος.

- cpu

Το γνώρισμα αυτό αφορά τους εικονικούς πυρήνες επεξεργαστικής ισχύος που έχει διαθέσιμους η εικονική μηχανή.

- memory

Το γνώρισμα αυτό αφορά το μέγεθος της εικονικής μνήμης που έχει διαθέσιμη η εικονική μηχανή.

- disk

Το γνώρισμα αυτό αφορά τη χωρητικότητα του εικονικού δίσκου που έχει διαθέσιμη η εικονική μηχανή.

- ips

Το γνώρισμα αυτό αφορά τις διευθύνσεις ip που έχουν αποδοθεί στην εικονική μηχανή.

- name

Το γνώρισμα αυτό αφορά το όνομα που έχει δοθεί στην εικονική μηχανή.

- username

Το γνώρισμα αυτό αφορά το όνομα του λογαριασμού του χρήστη στην εικονική μηχανή.

- password

Το γνώρισμα αυτό αφορά το password του παραπάνω λογαριασμού.

- providerID

Το γνώρισμα αυτό αφορά το id που έχει αποδοθεί στην εικονική μηχανή από τον πάροχο, δηλαδή το αναγνωριστικό με το οποίο αναφέρεται στην εικονική μηχανή.

- hostname

Το γνώρισμα αυτό αφορά το δικτυακό όνομα (fqdn) που έχει αποδοθεί στην εικονική μηχανή.

- status

Το γνώρισμα αυτό αφορά την κατάσταση (ενεργοποιημένη ή όχι, κτλ) της εικονικής μηχανής.

- messages

Το γνώρισμα αυτό αφορά μηνύματα που έχουν να κάνουν με την εξέλιξη των διάφορων ενεργειών πάνω στην εικονική μηχανή όπως π.χ. μηνύματα σφάλματος ή επιτυχίας κ.α. Αυτά μπορεί να παράγονται από τον πάροχο είτε από το backend.

- recipes

Το γνώρισμα αυτό αφορά τις εφαρμογές που εγκαταστάθηκαν (έπειτα από αίτημα του χρήστη) στην εικονική μηχανή. Η επιλογή του ονόματος έχει να κάνει με την ορολογία συγκεκριμένου εργαλείου που χρησιμοποιήθηκε για αυτόν τον σκοπό.

Βιβλία Μαγειρικής (Cookbooks)

Η συλλογή “Βιβλία Μαγειρικής” περιέχει έγγραφα που αφορούν τις διαθέσιμες εφαρμογές προς εγκατάσταση κατά τη δημιουργία μιας εικονικής μηχανής από τις οποίες μπορεί να διαλέξει ο χρήστης. Η επιλογή του ονόματος έχει να κάνει με την ορολογία συγκεκριμένου εργαλείου που χρησιμοποιήθηκε για αυτόν τον σκοπό. Ουσιαστικά ένα cookbook ομαδοποιεί κάποια recipes τα οποία είναι scripts που αναλαμβάνουν την εγκατάσταση συγκεκριμένων εφαρμογών.

- name

Το γνώρισμα αυτό αφορά το όνομα του cookbook.

- recipes

Το γνώρισμα αυτό αφορά τα διαθέσιμα recipes που προσφέρει το cookbook.

Αιτήματα (Requests)

Η συλλογή “Αιτήματα” περιέχει έγγραφα που αφορούν αιτηματα του χρήστη για ενέργειες πάνω σε εικονικές μηχανές. Κάθε αίτημα αντιπροσωπεύει μια ενέργεια και για την διεκπεραίωσή του πρέπει να προωθηθεί στο backend. Σημειώνεται ότι σε ένα έγγραφο αυτής της συλλογής μπορεί να τροποποιεί τα πεδία και το frontend (π.χ. όταν το δημιουργεί) και το backend καθώς σε κάποιες περιπτώσεις καποια στοιχεία γίνονται διαθέσιμα μόνο αφού ολοκληρωθεί η σχετική ενέργεια.

- vm_id

Το γνώρισμα αυτό αφορά το id που έχει μέσα στο σύστημα η εικονική μηχανή στην οποία αναφέρεται το αίτημα.

- account_id

Το γνώρισμα αυτό αφορά το id που έχει μέσα στο σύστημα ο λογαριασμός στον οποίο ανήκει η εικονική μηχανή.

- user_id

Το γνώρισμα αυτό αφορά το id που έχει μέσα στο σύστημα ο χρήστης στον οποίο ανήκει ο λογαριασμός.

- vm

Το γνώρισμα αυτό αφορά το έγγραφο που αντιπροσωπεύει την εικονική μηχανή στη βάση δεδομένων του συστήματος. Ουσιαστικά είναι ένα έγγραφο της συλλογής “Εικονικές Μηχανές”.

- command

Το γνώρισμα αυτό αφορά τη λειτουργία που έχει ζητήθει από τον χρήστη.

- token

Το γνώρισμα αυτό αφορά το token του παρόχου.

- provider

Το γνώρισμα αυτό αφορά τον πάροχο στον οποίο αντιστοιχεί ο λογαριασμός στον οποίο ανήκει η εικονική μηχανή.

- messages

Το γνώρισμα αυτό αφορά μηνύματα που έχουν να κάνουν με την εξέλιξη των διάφορων ενεργειών πάνω στην εικονική μηχανή.

- status

Το γνώρισμα αυτό αφορά την κατάσταση του αιτήματος (π.χ. επιτυχές ή όχι κτλ).

- ssh

Το γνώρισμα αυτό αφορά την επιλεγμένη εντολή που ζητείται να εκτελεστεί μέσω ssh. Αυτό το γνώρισμα χρειάζεται μόνο όταν η τιμή του πεδίου command είναι “ssh”.

5
Υλοποίηση και Έλεγχος

5.1 Οδηγός Εγκατάστασης

Σε αυτή την ενότητα περιγράφεται η διαδικασία εγκατάστασης του συστήματος. Όπως περιγράφηκε σε προηγούμενο κεφάλαιο, το σύστημα χωρίζεται σε δύο τμήματα, frontend και backend, τα οποία μπορούν να λειτουργήσουν σε διαφορετικά μηχανήματα. Η εγκατάσταση του frontend είναι πιο πολύπλοκη από αυτή του backend και για αυτό θα δοθεί μεγαλύτερη βάση σε αυτό, ενώ θα γίνει και μια μικρή αναφορά στο backend στο τέλος. Η διαδικασία που περιγράφεται γίνεται σε μεγάλο βαθμό γίνεται με τη χρήση τερματικού και εκτυλίσσεται σε ένα σύστημα με Ubuntu 14.04.1 αμέσως μετά την ολοκλήρωση της εγκατάστασης του λειτουργικού συστήματος (και πιθανότατα είναι έγκυρη για τις περισσότερες debian-based διανομές). Γενικά, όταν παρατίθεται μια εντολή εγκατάστασης της μορφής sudo apt-get install <πακέτο>, μπορεί πέρα από το <πακέτο> να χρειαστεί να εγκατασταθούν και άλλα πακέτα ως προαπαιτούμενα, τα οποία όμως προτείνονται αυτόματα. Αρχικά εγκαθιστούμε κάποια βασικά εργαλεία με την εντολή:

```
$ sudo apt-get install git curl nodejs expect
```

Εγκατάσταση Ruby

Το πρώτο βήμα είναι η εγκατάσταση της Ruby. Για να αποφευχθούν τα όποια προβλήματα σχετίζονται με την εγκατάσταση μιας ή περισσότερων (για λόγους συμβατότητας) εκδόσεων της γλώσσας Ruby προτείνεται η χρήση του Ruby Version Manager (RVM) που επιτρέπει την εγκατάσταση και διαχείριση πολλαπλών εκδόσεων της Ruby στο ίδιο μηχάνημα.

Έτσι, αρχικά πρέπει να γίνει η εγκατάσταση του RVM γράφοντας σε ένα τερματικό την εντολή:

```
$ curl -sSL https://get.rvm.io | bash -s stable
```

Στη συνέχεια για να εγκατασταθεί η Ruby ελέγχονται οι απαιτήσεις:

```
$ rvm requirements
```

Σε περίπτωση που εμφανιστεί το μήνυμα “command not found”, θα πρέπει να χρησιμοποιηθεί η παρακάτω εντολή:

```
$ source ~/.rvm/scripts/rvm
```

Ενδέχεται να χρειαστούν και τα παρακάτω:

```
$ sudo apt-get install libyaml-dev
```

Τελικά, γίνεται εγκατάσταση της έκδοσης 2.0.0 της γλώσσας Ruby:

```
$ rvm install 2.0.0
```

Υπάρχει περίπτωση να προκύψει κάποιο πρόβλημα με την τοποθεσία του OpenSSL. Τότε θα πρέπει να χρησιμοποιηθεί η παρακάτω εντολή:

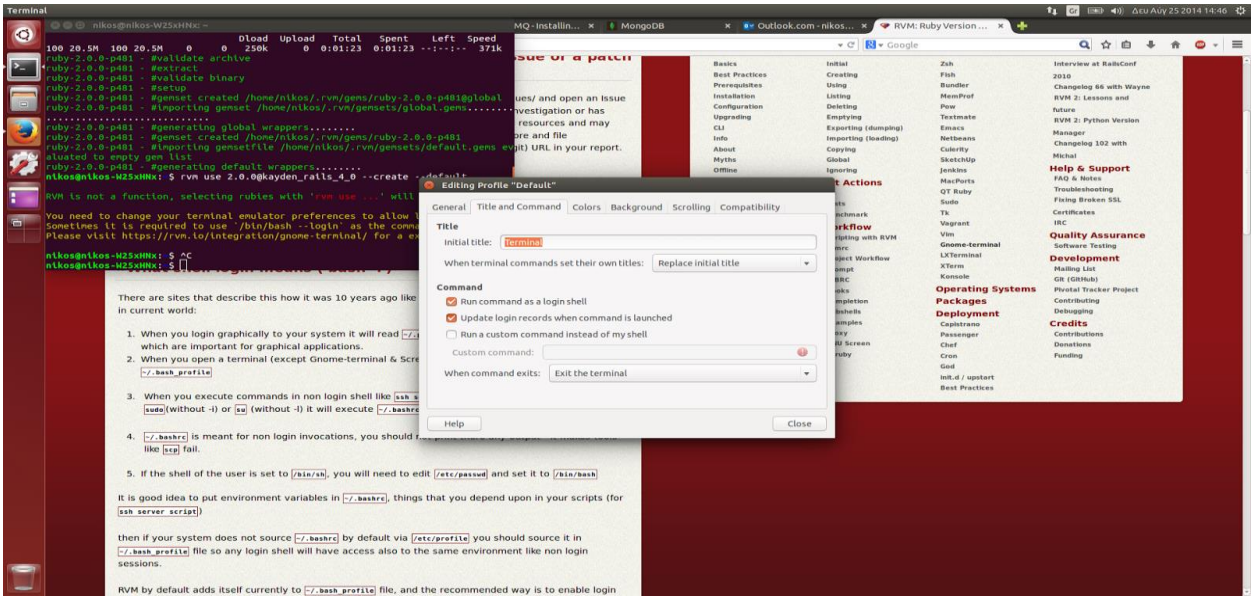
```
$ rvm install 2.0.0 --with-openssl-dir=$HOME/.rvm/usr
```

όπου στο όρισμα "`--with-openssl-dir`" πρέπει να μπει η τοποθεσία του OpenSSL.

Μετά την εγκατάσταση της Ruby, πρέπει να γίνουν ρυθμίσεις για άλλα λογισμικά που χρειάζονται για να τρέξουν εφαρμογές του Rails. Αυτό συνήθως περιλαμβάνει την εγκατάσταση κάποιων gems, τα οποία είναι αυτοτελή πακέτα κώδικα Ruby. Λόγω του ότι τα gems με διαφορετικούς αριθμούς έκδοσης κάποιες φορές δημιουργούν προβλήματα, είναι βολικό να δημιουργηθούν ξεχωριστά gemsets, τα οποία είναι αυτοτελή σύνολα από gems. Για αυτό, λοιπόν, προτείνεται η δημιουργία ενός gemset με όνομα kayden_rails_4_0 (το όνομα μπορεί να είναι οτιδήποτε):

```
$ rvm use 2.0.0@kayden_rails_4_0 --create –default
```

Σε αυτό το σημείο μπορεί να εμφανιστεί το μήνυμα "RVM is not a function, selecting rubies with 'rvm use ...' will not work.". Τότε θα πρέπει να πατήσουμε δεξί κλικ πάνω στο τερματικό, να επιλέξουμε "Profile Preferences" και στην καρτέλα "Title and Command" να τσεκάρουμε το "Run command as a login shell". Αυτά φαίνονται και στην παρακάτω εικόνα:



Εικόνα 5.1 - Ρύθμιση login shell

Αυτή η εντολή (`rvm use 2.0.0@kayden_rails_4_0 --create --default`) δημιουργεί (--create) το gemset kayden_rails_4_0 και το συσχετίζει με την έκδοση 2.0.0 της Ruby ενώ κανονίζει την άμεση έναρξη της χρήσης του (use) και το ορίζει ως προεπιλεγμένο (--default) gemset, έτσι ώστε κάθε φορά που ανοίγει ένα καινούργιο παράθυρο τερματικού ο συνδυασμός Ruby/gemset 2.0.0@kayden_rails_4_0 να είναι αυτομάτως επιλεγμένος. Το RVM παρέχει μια μεγάλη ποικιλία εντολών για τη διαχείριση των gemsets και αν υπάρξει κάποιο πρόβλημα οι παρακάτω εντολές μπορεί να βοηθήσουν:

```
$ rvm help
$ rvm gemset help
```

Εγκατάσταση RubyGems

Το RubyGems είναι ένας διαχειριστής πακέτων για έργα σε Ruby, και υπάρχουν πολλές χρήσιμες βιβλιοθήκες (συμπεριλαμβανομένου του Rails) διαθέσιμες ως πακέτα Ruby, ή αλλιώς gems. Σε περίπτωση που έχει γίνει εγκατάσταση του RVM, όπως προτάθηκε προηγουμένως, το RubyGems είναι ήδη εγκατεστημένο, καθώς το RVM το εγκαθιστά αυτόματα. Ειδάλλως, μπορεί να εγκατασταθεί κατεβάζοντας το από το http://rubygems.org/pages/download και χρησιμοποιώντας την εντολή:

```
$ ruby setup.rb
```

Κατά την εγκατάσταση gems, το RubyGems εξ ορισμού παράγει δύο διαφορετικά είδη τεκμηριώσεων (το ri και το rdoc), αλλά πολλοί προγραμματιστές Ruby και Rails θεωρούν ότι τα οφέλη που προσφέρουν δεν αξίζουν τον χρόνο που χρειάζεται για τη δημιουργία τους και για

αυτό βασίζονται στη διαδικτυακή τεκμηρίωση. Για να ακυρωθεί η αυτόματη παραγωγή των παραπάνω προτείνεται η δημιουργία ενός αρχείου με όνομα .gemrc στο home directory το οποίο θα πρέπει να περιέχει τις παρακάτω γραμμές:

```
install: --no-rdoc --no-ri
update:  --no-rdoc --no-ri
```

Εγκατάσταση Rails

Μόλις ολοκληρωθεί και η εγκατάσταση του RubyGems, η εγκατάσταση του Rails είναι εύκολη. Εδώ χρησιμοποιείται η έκδοση 4.0, η οποία εγκαθίσταται ως εξής:

```
$ gem install rails --version 4.0.8
```

Για να επιβεβαιωθεί η ορθή εγκατάσταση του Rails, χρησιμοποιείται η παρακάτω εντολή που τυπώνει τον αριθμό έκδοσης:

```
$ rails –v
```

Επίσης, ενδέχεται να χρειαστεί η εγκατάσταση των παρακάτω πακέτων:

```
$ sudo apt-get install libxslt-dev libxml2-dev libsqlite3-dev
```

Εγκατάσταση RabbitMQ

Για την εγκατάσταση του RabbitMQ πρέπει να χρησιμοποιηθεί η παρακάτω εντολή:

```
$ sudo apt-get install rabbitmq-server
```

Εγκατάσταση MongoDB

Για την εγκατάσταση της MongoDB χρησιμοποιείται η εντολή:

```
$ sudo apt-get install mongodb
```

Αυτή εγκαθιστά τη MongoDB από τα πακέτα της διανομής (εδώ Ubuntu 14.04.1). Υπάρχει και η επιλογή απευθείας εγκατάστασης από τα επίσημα αποθετήρια που είναι πιο ενημερωμένα αλλά για τις ανάγκες αυτής της εργασίας δεν χρησιμοποιήθηκε αυτή η επιλογή.

Εγκατάσταση κώδικα και επιπλέον gems

Σε αυτό το σημείο χρειάζεται να κατεβάσουμε τον κώδικα της εφαρμογής. Πρώτα όμως πρέπει να φτιάξουμε ένα directory στο οποίο θα τοποθετηθεί, το οποίο εδώ θα είναι το ~/rails_projects/. Τα παραπάνω γίνονται με τις ακόλουθες εντολές:

```
$ mkdir ~/rails_projects/
$ cd ~/rails_projects/
$ git clone https://dkmsgroup@bitbucket.org/dkmsgroup/kayden.git
```

Τώρα θα πρέπει να έχει δημιουργηθεί το directory ~/rails_projects/kayden/ το οποίο θα περιέχει τον κώδικα της εφαρμογής. Σε αυτό το σημείο, και με δεδομένη την εγκατάσταση του RubyGems θα τρέξουμε τις παρακάτω εντολές:

```
$ cd kayden/
$ bundle install
```

Αυτό θα αναλάβει να κοιτάξει ένα ειδικό αρχείο στον κώδικα της εφαρμογής, που λέγεται Gemfile και στο οποίο καταγράφονται τα gems που πρέπει να είναι εγκατεστημένα, και να εγκαταστήσει αυτά που λείπουν.

Επιμέρους ρυθμίσεις

Σε αυτό το σημείο το frontend λειτουργεί πλήρως. Χρειάζεται μια επιπλέον ρύθμιση σε επίπεδο βάσης δεδομένων, καθώς θα πρέπει να καταγραφούν στη βάση δεδομένων τα cookbooks που είναι διαθέσιμα. Αυτό γίνεται με την παρακάτω εντολή:

```
$ mongorestore -db kayden_development -c cookbooks --dir\
> dump/kayden_development/cookbooks.bson
```

Αυτή η εντολή παίρνει το αρχείο cookbooks.bson, που παρέχεται μαζί με τον κώδικα της εφαρμογής, και το εισάγει στη βάση δεδομένων kayden_development στη συλλογή cookbooks.

Εγκατάσταση Chef

Από εδώ και πέρα ότι εγκαθίσταται αφορά τη λειτουργία του backend. Αν πρόκειται και τα δύο μέρη να λειτουργήσουν στο ίδιο μηχάνημα τότε συνεχίζουμε τη διαδικασία από το αμέσως παραπάνω βήμα. Αν το backend προορίζεται να τρέξει σε διαφορετικό μηχάνημα μπορούμε να παραλείψουμε την εγκατάσταση του Rails και της MongoDB (προφανώς μαζί με τη ρύθμιση για τα cookbooks). Σημειώνεται ότι σε αυτή την περίπτωση θα πρέπει να δοθεί μεγάλη προσοχή στις λεπτομέρειες που αναφέρονται στο τέλος της ενότητας και αφορούν την επικοινωνία μεταξύ frontend και backend όταν αυτά τρέχουν σε διαφορετικούς υπολογιστές.

Πρώτο βήμα, λοιπόν, για τις ανάγκες του backend είναι η εγκατάσταση του Chef. Επειδή τη στιγμή που γράφεται αυτή η εργασία η τελευταία έκδοση του Chef είναι alpha και δεν λειτουργεί σωστά θα εγκαταστήσουμε αυτή με την οποία έγιναν οι δοκιμές.

```
$ gem install chef --version 11.14.6
```

Στη συνέχεια πρέπει να εγκατασταθούν και κάποια επιπλέον gems.

```
$ gem install knife-solo librarian-chef net-ping bunny json httparty
```

Επικοινωνία μεταξύ frontend και backend

Σε περίπτωση που τα δύο τμήματα τρέχουν σε διαφορετικούς υπολογιστές θα πρέπει να γίνουν οι παρακάτω τροποποιήσεις. Συμβολίζουμε με $FIP και $BIP τις ip διευθύνσεις των frontend και backend αντίστοιχα.

- Αλλαγές στο frontend

Αρχείο: `~/rails_projects/kayden/app/helpers/vms_helper.rb`
Γραμμή 4: `conn = Bunny.new(:hostname => "$BIP")`
Γραμμή 43: `conn = Bunny.new(:hostname => "$BIP")`

- Αλλαγές στο backend

Αρχείο: `~/rails_projects/kayden/backend/backend_init.rb`
Γραμμή 10: `server_url = "http://$FIP:3000/"`

Έναρξη λειτουργίας

- frontend

Ανοίγουμε ένα τερματικό και εκτελούμε:

```
$ cd ~/rails_projects/kayden
$ rails s
```

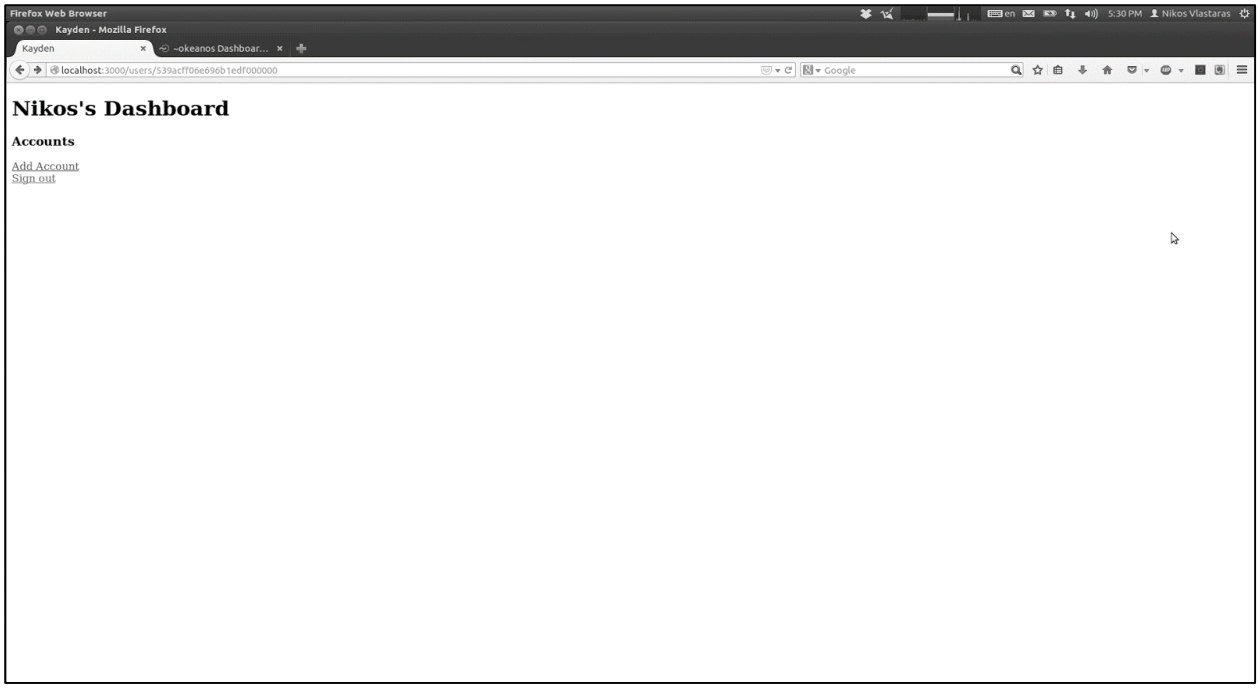
- backend

Ανοίγουμε ένα τερματικό και εκτελούμε:

```
$ cd ~/rails_projects/kayden/backend
$ ./backend_init.rb
```

5.2 Εκτέλεση

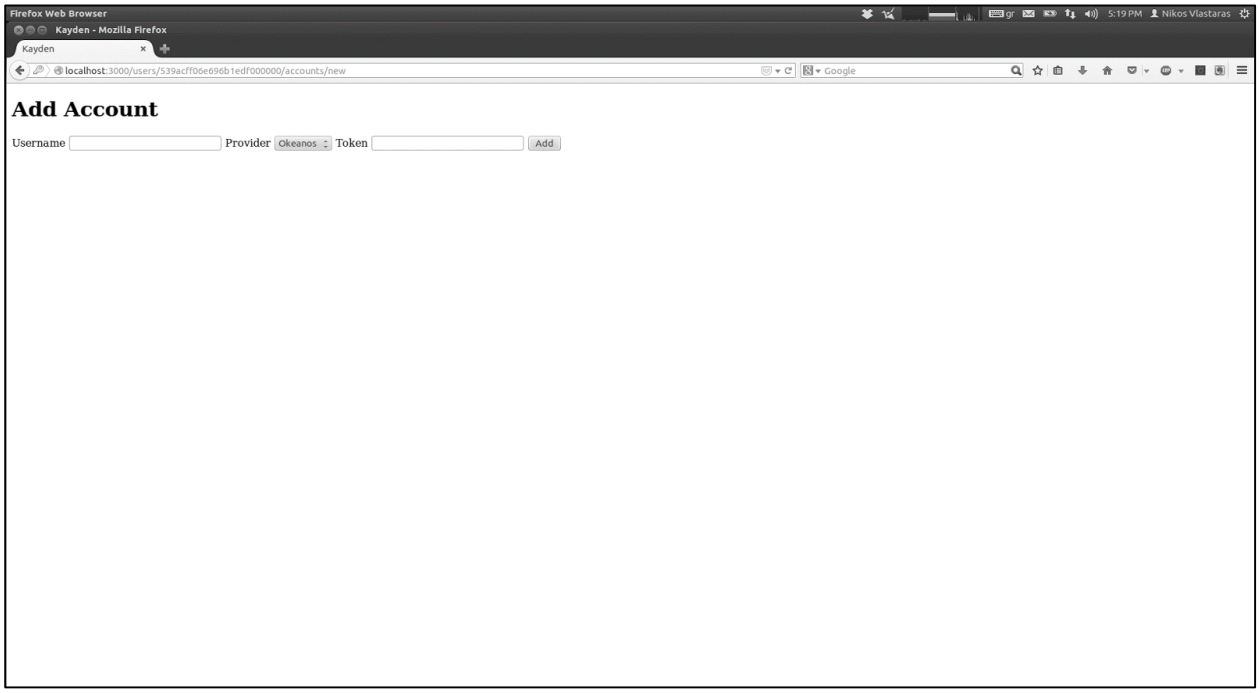
Σε αυτή την ενότητα θα περιγραφεί η διαδικασία εκτέλεσης του συστήματος. Για αυτό τον σκοπό θα χρησιμοποιηθεί ένας λογαριασμός στην υπηρεσία IaaS cloud computing okeanos. Δεν θα περιγραφούν οι λειτουργίες εγγραφής, σύνδεσης και αποσύνδεσης χρήστη καθώς θεωρούνται τετριμμένες. Όλες οι παρακάτω περιγραφές θα ξεκινούν από το dashboard του χρήστη που είναι η οθόνη στην οποία βρίσκεται αμέσως μόλις εγγραφεί ή συνδεθεί.



Εικόνα 5.2 - Dashboard χρήστη

Προσθήκη λογαριασμού

Από το Dashboard διαλέγουμε την επιλογή “Add Account”.

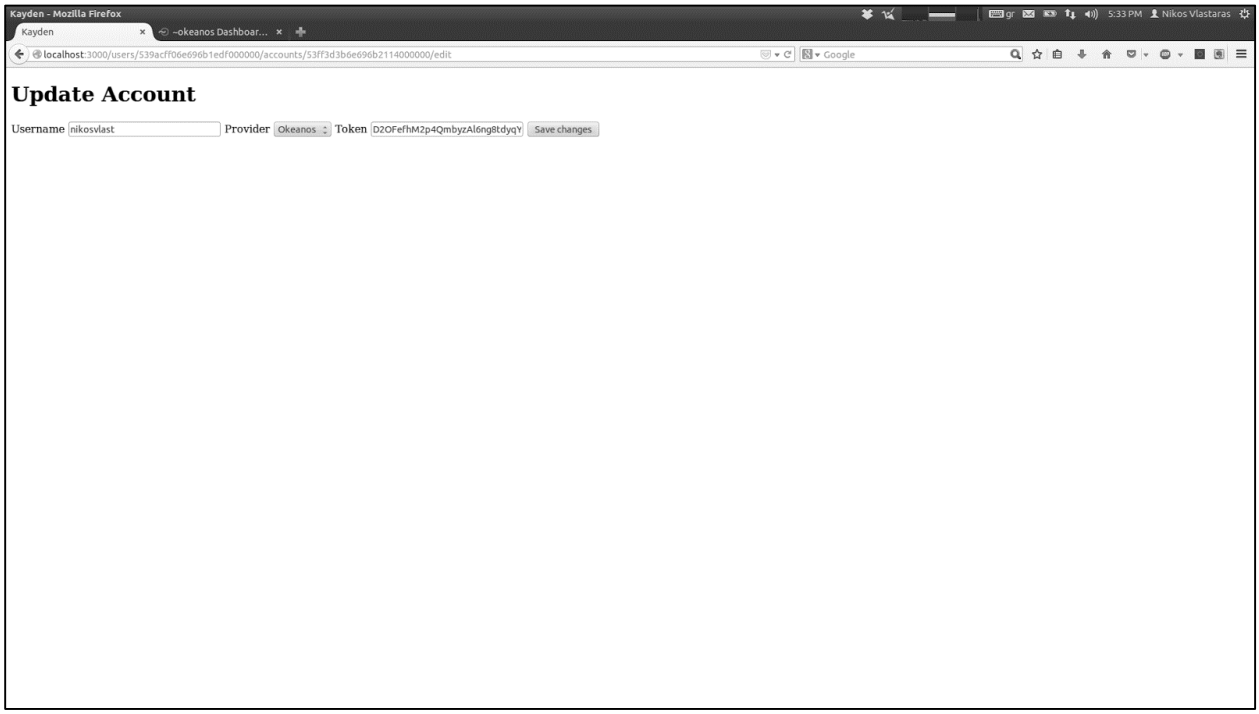


Εικόνα 5.3 - Προσθήκη λογαριασμού

Στη συνέχεια μεταβαίνουμε στην οθόνη προσθήκης λογαριασμού στην οποία συμπληρώνουμε τα απαραίτητα στοιχεία. Έπειτα πατάμε “Add”, ο λογαριασμός προστίθεται και βρισκόμαστε και πάλι στο Dashboard.

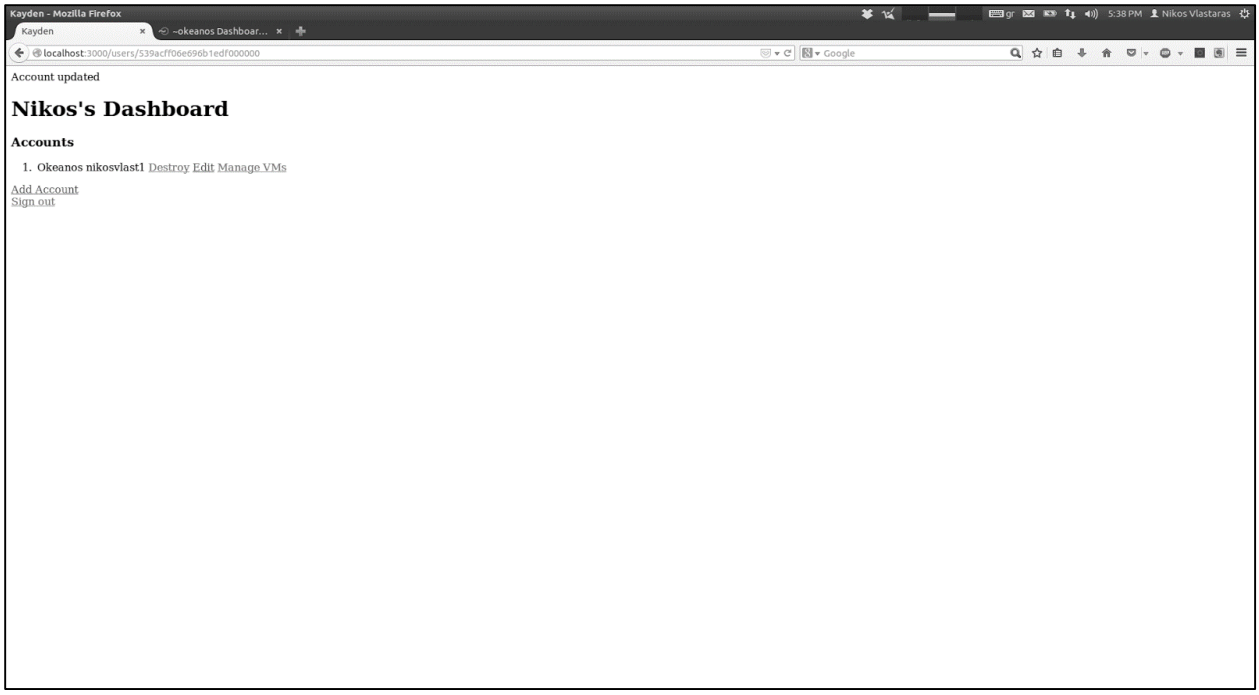
Επεξεργασία λογαριασμού

Από το Dashboard διαλέγουμε την επιλογή “Edit” που βρίσκεται δίπλα από τον λογαριασμό του οποίου τις λεπτομέρειες θέλουμε να επεξεργαστούμε.



Εικόνα 5.4 - Τροποποίηση λογαριασμού

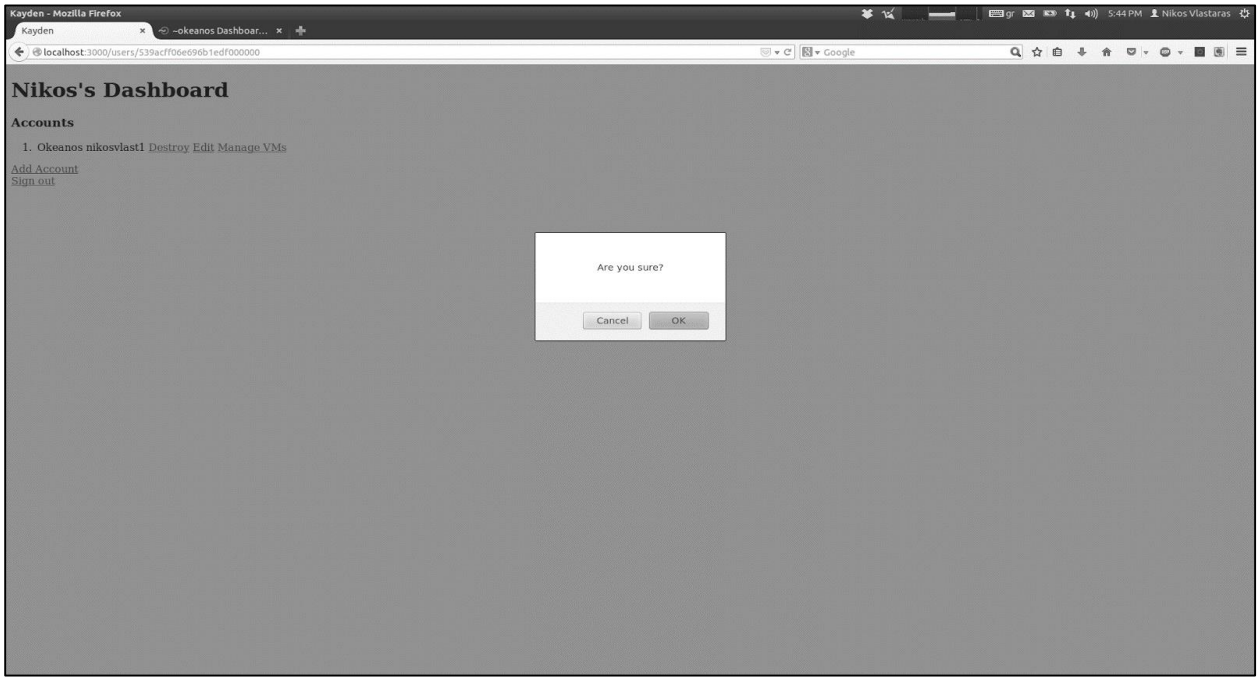
Μετά τροποποιούμε κάποια πληροφορία, επιλέγουμε “Save Changes”, οι πληροφορίες τροποποιούνται και βρισκόμαστε και πάλι στο Dashboard.



Εικόνα 5.5 - Ολοκλήρωση τροποποίησης λογαριασμού

Διαγραφή λογαριασμού

Από το Dashboard διαλέγουμε την επιλογή “Destroy” που βρίσκεται δίπλα από τον λογαριασμό τον οποίο θέλουμε να διαγράψουμε.

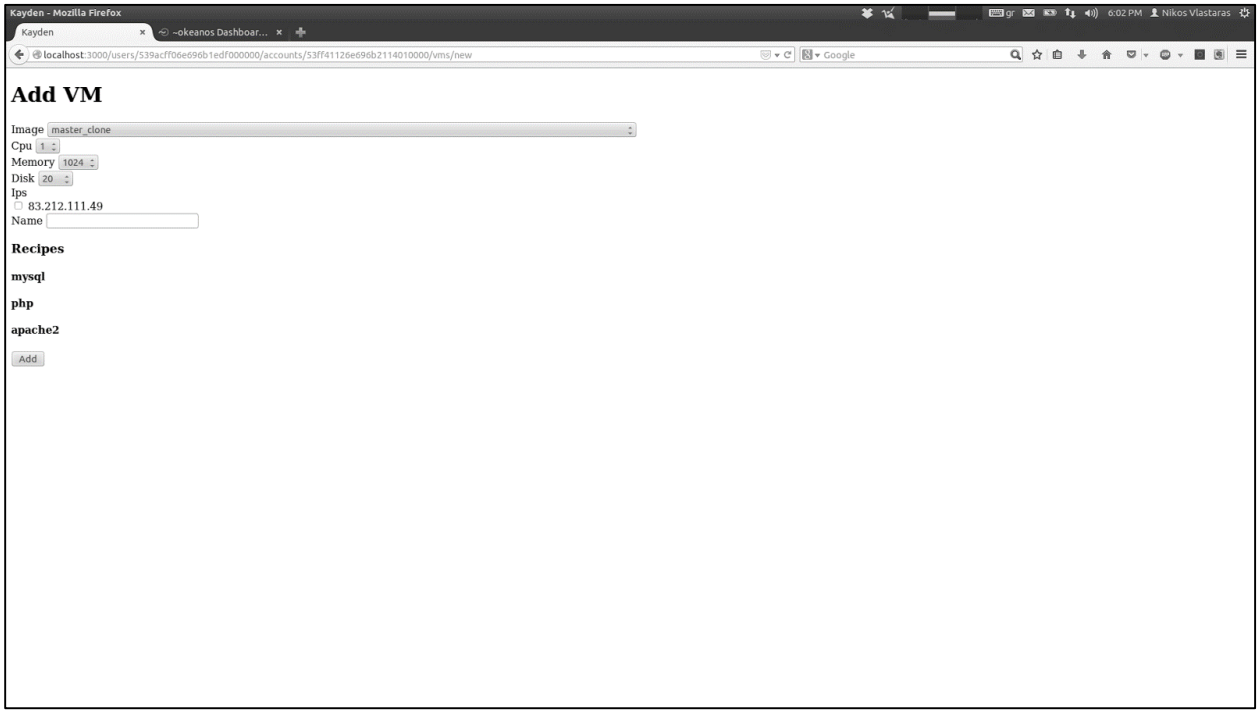


Εικόνα 5.6 - Διαγραφή λογαριασμού

Τότε εμφανίζεται ένα παράθυρο διαλόγου επιβεβαίωσης για τη διαγραφή και αν επιλέξουμε “OK” τότε ο λογαριασμός διαγράφεται και βρισκόμαστε και πάλι στο Dashboard.

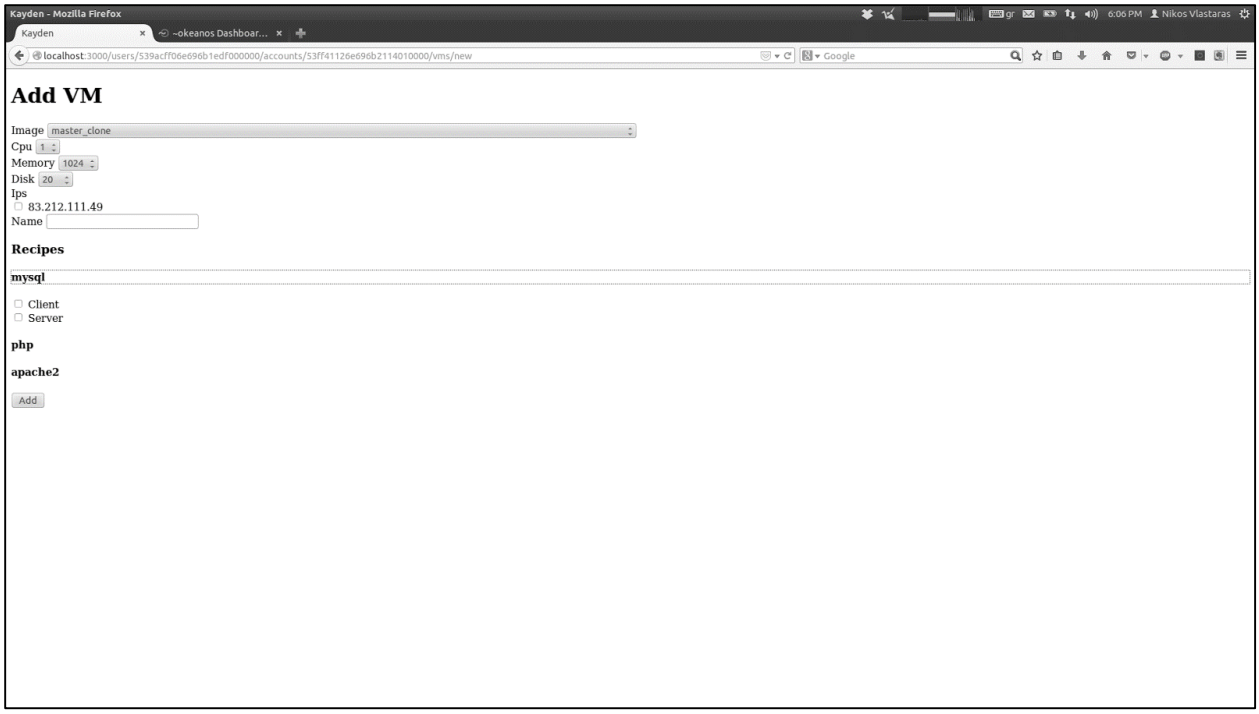
Δημιουργία εικονικής μηχανής

Από το Dashboard διαλέγουμε την επιλογή “Manage VMs” που βρίσκεται δίπλα από τον λογαριασμό στον οποίο θέλουμε να δημιουργήσουμε μια εικονική μηχανή.



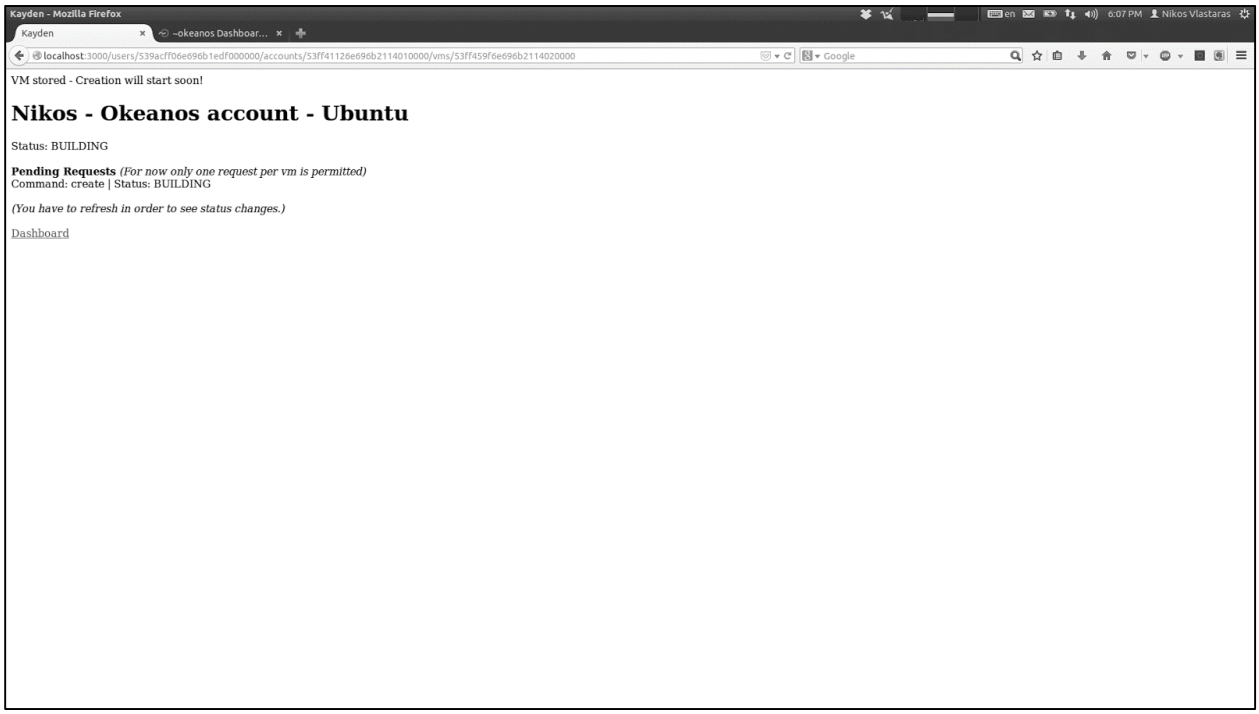
Εικόνα 5.7 - Δημιουργία εικονικής μηχανής

Τότε επιλέγουμε το “Add VM” και βρισκόμαστε στην οθόνη που διαλέγουμε τις λεπτομέρειες της εικονικής μηχανής που πρόκειται να δημιουργηθεί και συμπληρώνουμε τα βασικά στοιχεία.



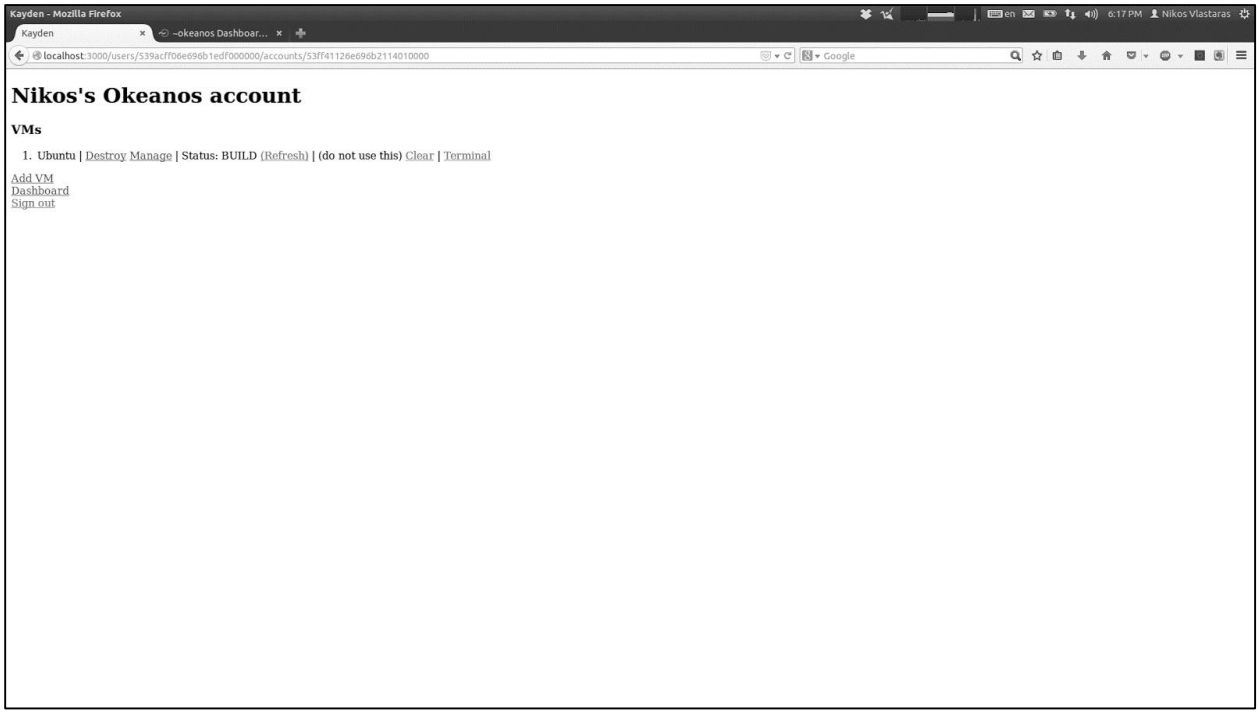
Εικόνα 5.8 - Επιλογή recipes

Κάτω από το “Recipes” εμφανίζονται τα ονόματα των διαθέσιμων cookbooks και πατώντας πάνω σε κάποιο από αυτά εμφανίζονται τα διαθέσιμα recipes του συγκεκριμένου cookbook.



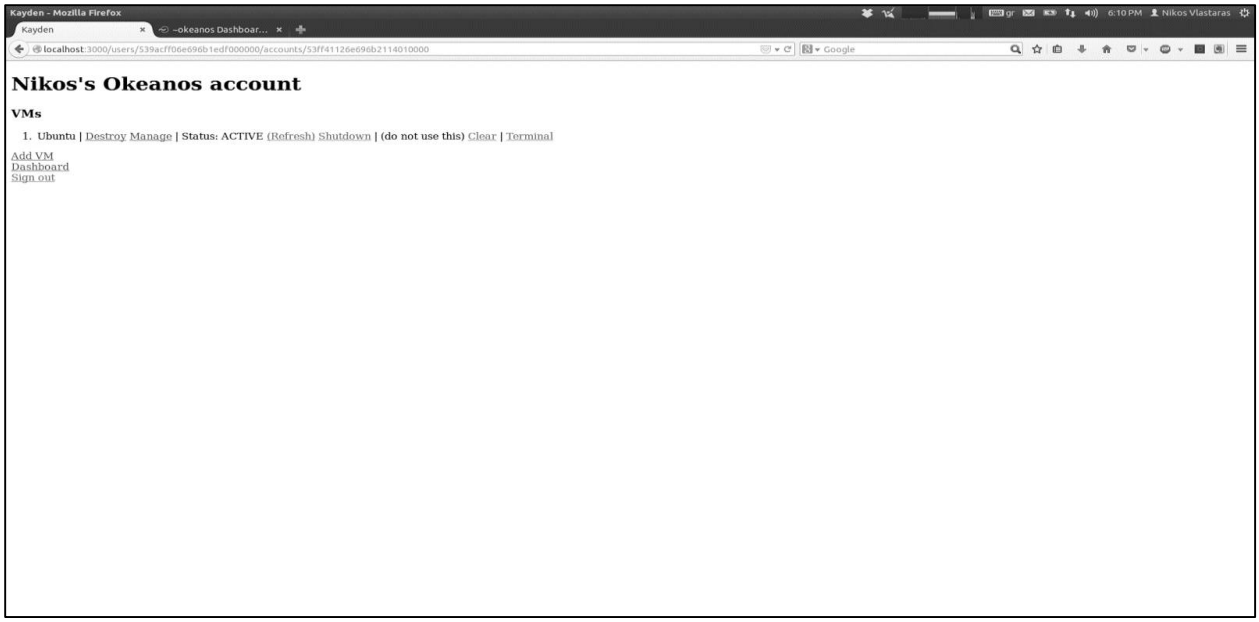
Εικόνα 5.9 - Επιβεβαίωση αιτήματος δημιουργίας

Στη συνέχεια πατάμε “Add”, ξεκινάει η διαδικασία δημιουργίας της εικονικής μηχανής και βρισκόμαστε σε μια οθόνη που μας δίνει κάποιες πληροφορίες για την εικονική μηχανή όπως την κατάσταση και τα αιτήματα που εκκρεμούν για αυτή. Προς το παρόν υποστηρίζεται μόνο ένα αίτημα για κάθε εικονική μηχανή οπότε η λίστα θα έχει μόνο το αίτημα για τη δημιουργία της.



Εικόνα 5.10 - Σελίδα λογαριασμού χρήστη – εικονική μηχανή υπό κατασκευή

Έπειτα διαλέγουμε την επιλογή Dashboard και στη συνέχεια την επιλογή “Manage VMs” δίπλα από τον λογαριασμό στον οποίο προσθέσαμε την εικονική μηχανή. Τότε βρισκόμαστε σε μια οθόνη όπου καταγράφονται σε μια λίστα όλες οι εικονικές μηχανές του λογαριασμού και δίνεται η δυνατότητα για κάποιες λειτουργίες πάνω σε αυτές ενώ μπορούμε να πληροφορηθούμε και για την κατάστασή τους (αν λειτουργεί ή όχι κτλ).

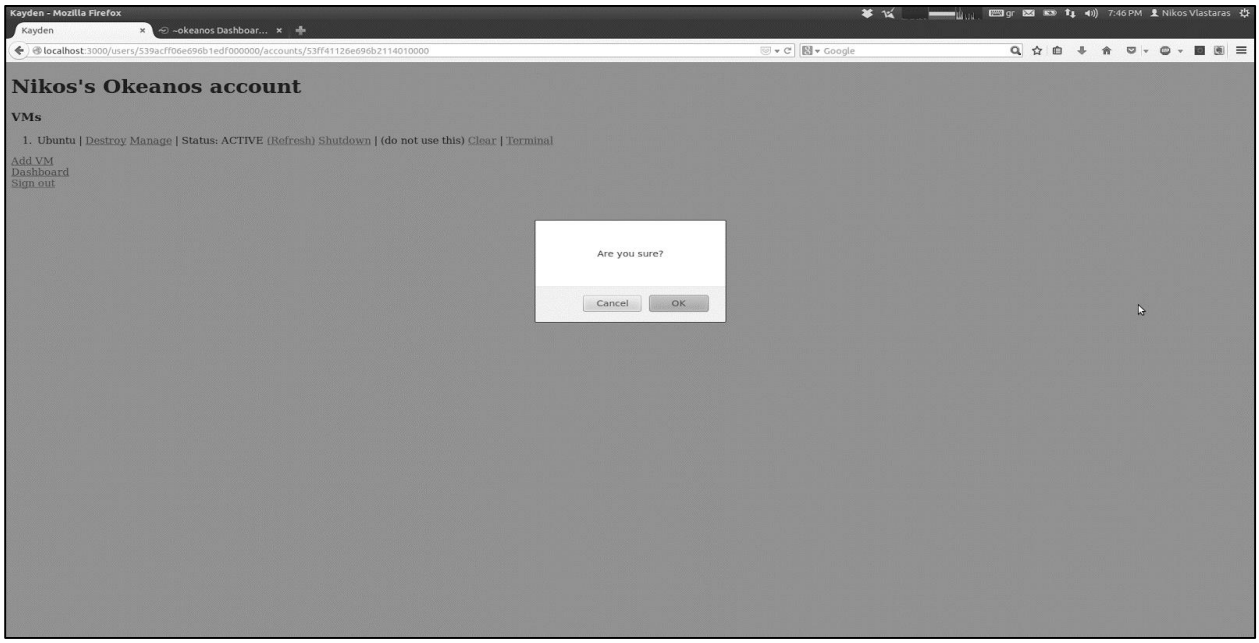


Εικόνα 5.11 - Ολοκλήρωση κατασκευής εικονικής μηχανής

Σε αυτή την περίπτωση στην κατάσταση της εικονικής μηχανής φαίνεται ότι η δημιουργία της βρίσκεται σε εξέλιξη. Μόλις ολοκληρωθεί η δημιουργία, μετά από ανανέωση της σελίδας, η εικονική μηχανή φαίνεται να είναι ενεργή.

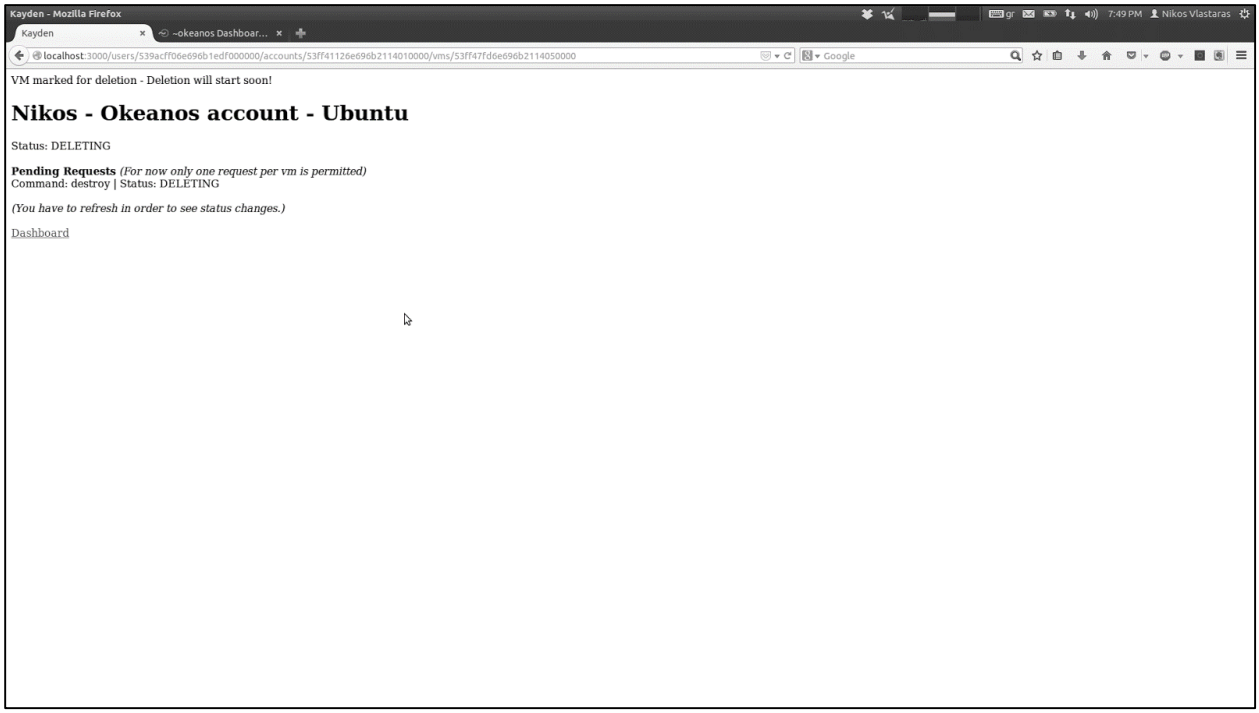
Καταστροφή εικονικής μηχανής

Από το Dashboard διαλέγουμε την επιλογή “Manage VMs” που βρίσκεται δίπλα από τον λογαριασμό από τον οποίο θέλουμε να καταστρέψουμε μια εικονική μηχανή.



Εικόνα 5.12 - Καταστροφή εικονικής μηχανής

Τότε επιλέγουμε το “Destroy” δίπλα από την εικονική μηχανή που θέλουμε να καταστρέψουμε. Στη συνέχεια εμφανίζεται ένα παράθυρο διαλόγου επιβεβαίωσης για τη διαγραφή και αν επιλέξουμε “OK” τότε ξεκινάει η καταστροφή της εικονικής μηχανής και βρισκόμαστε στην οθόνη με τις πληροφορίες της εικονικής μηχανής όπου βλέπουμε να εκκρεμεί το αίτημα καταστροφής της.



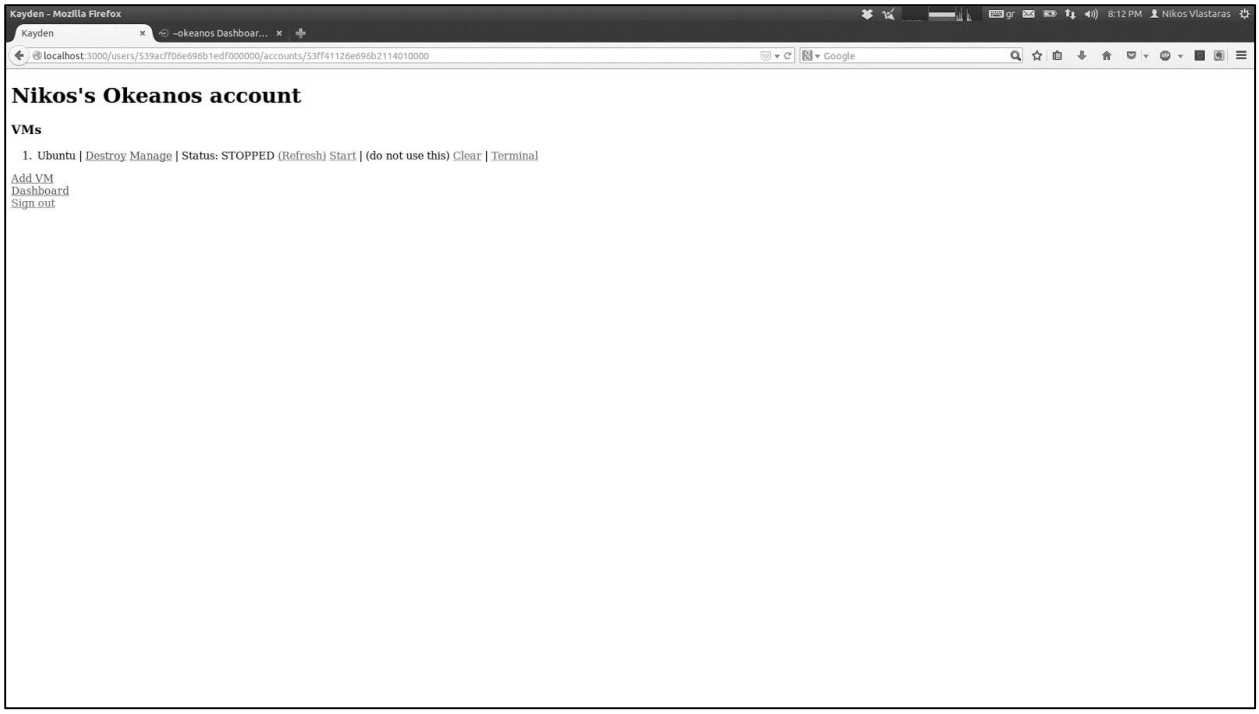
Εικόνα 5.13 - Επιβεβαίωση αιτήματος καταστροφής

Έπειτα διαλέγουμε την επιλογή Dashboard και στη συνέχεια την επιλογή “Manage VMs” δίπλα από τον λογαριασμό από τον οποίο καταστρέψαμε την εικονική μηχανή. Τότε βρισκόμαστε στην οθόνη που φαίνονται όλες οι εικονικές μηχανές του λογαριασμού και βλέπουμε στην κατάσταση της εικονικής μηχανής ότι η καταστροφή της βρίσκεται σε εξέλιξη.

Μόλις ολοκληρωθεί, μετά από ανανέωση της σελίδας, η εικονική μηχανή δεν φαίνεται πια στη λίστα.

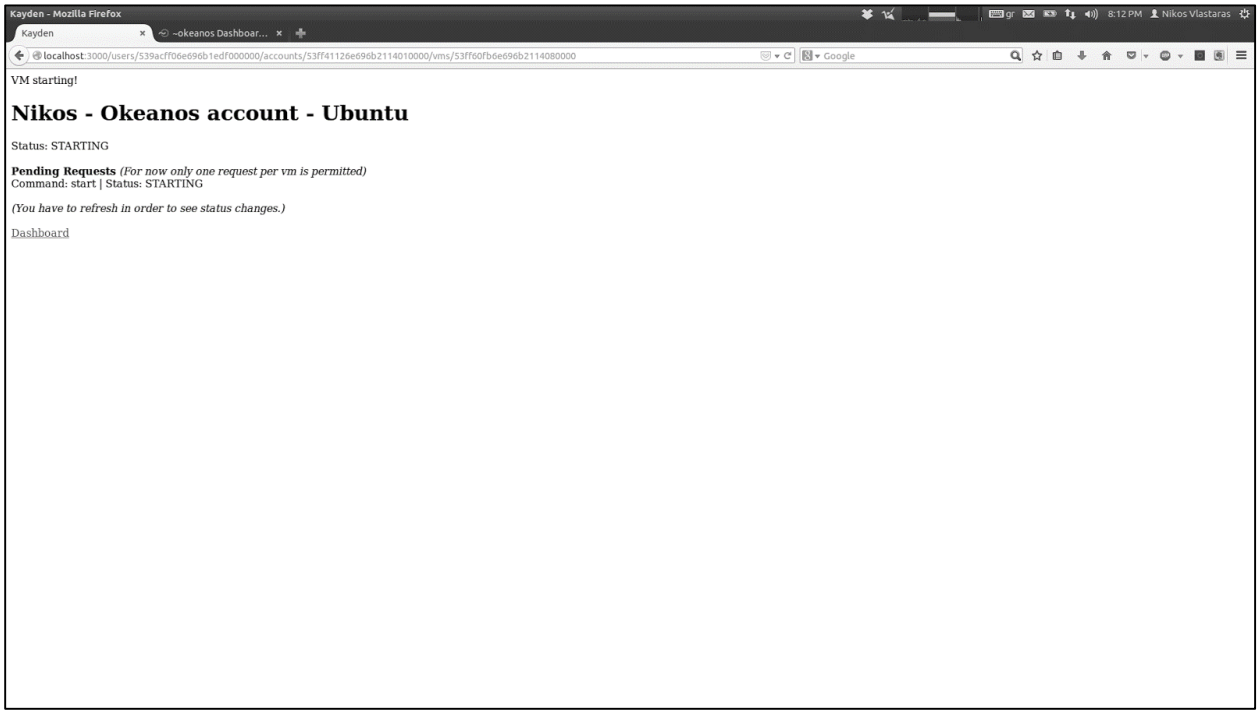
Έναρξη εικονικής μηχανής

Από το Dashboard διαλέγουμε την επιλογή “Manage VMs” που βρίσκεται δίπλα από τον λογαριασμό στον οποίο θέλουμε να εκκινήσουμε μια εικονική μηχανή.



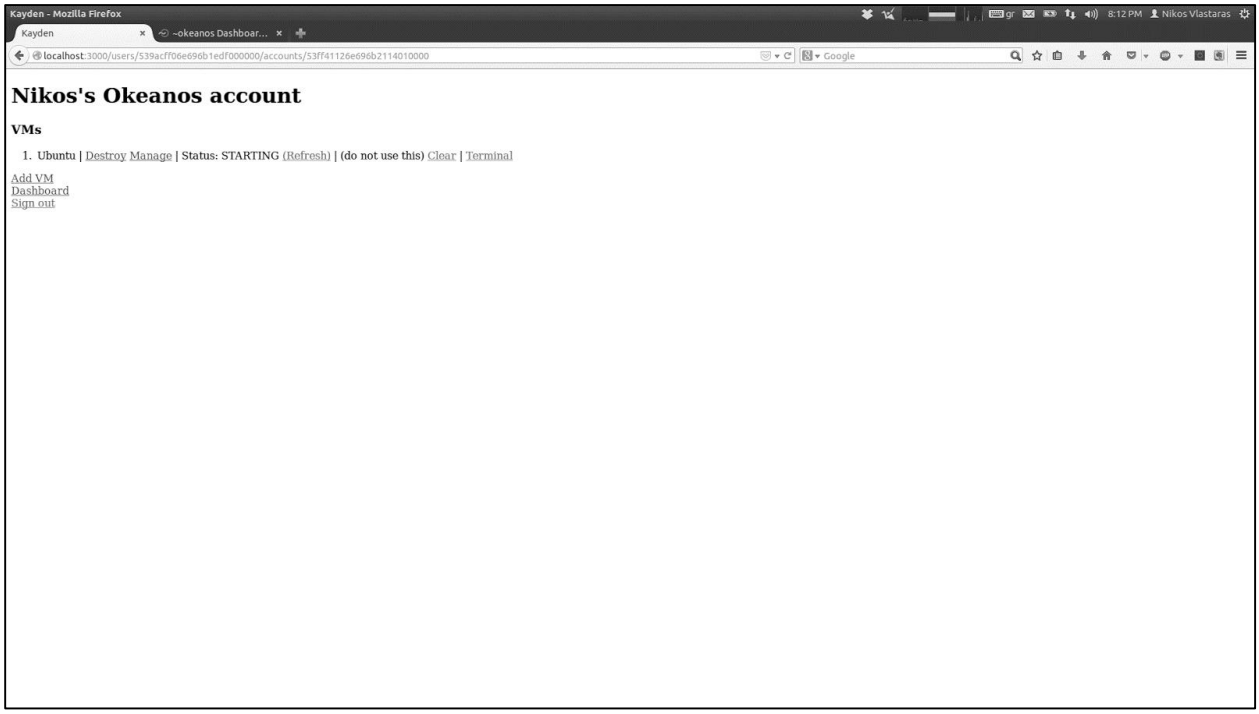
Εικόνα 5.14 - Εικονική μηχανή σταματημένη

Τότε επιλέγουμε το “Start” δίπλα από την εικονική μηχανή που θέλουμε να εκκινήσουμε και βρισκόμαστε στην οθόνη με τις πληροφορίες της εικονικής μηχανής όπου βλέπουμε να εκκρεμεί το αίτημα εκκίνησής της. Πρέπει να σημειωθεί ότι για να υπάρχει αυτή η δυνατότητα πρέπει η κατάσταση της εικονικής μηχανής να είναι STOPPED.



Εικόνα 5.15 - Επιβεβαίωση αιτήματος έναρξης εικονικής μηχανής

Έπειτα διαλέγουμε την επιλογή Dashboard και στη συνέχεια την επιλογή “Manage VMs” δίπλα από τον λογαριασμό στον οποίο εκκινήσαμε την εικονική μηχανή. Τότε βρισκόμαστε στην οθόνη που φαίνονται όλες οι εικονικές μηχανές του λογαριασμού και βλέπουμε στην κατάσταση της εικονικής μηχανής ότι η εκκίνηση της βρίσκεται σε εξέλιξη.

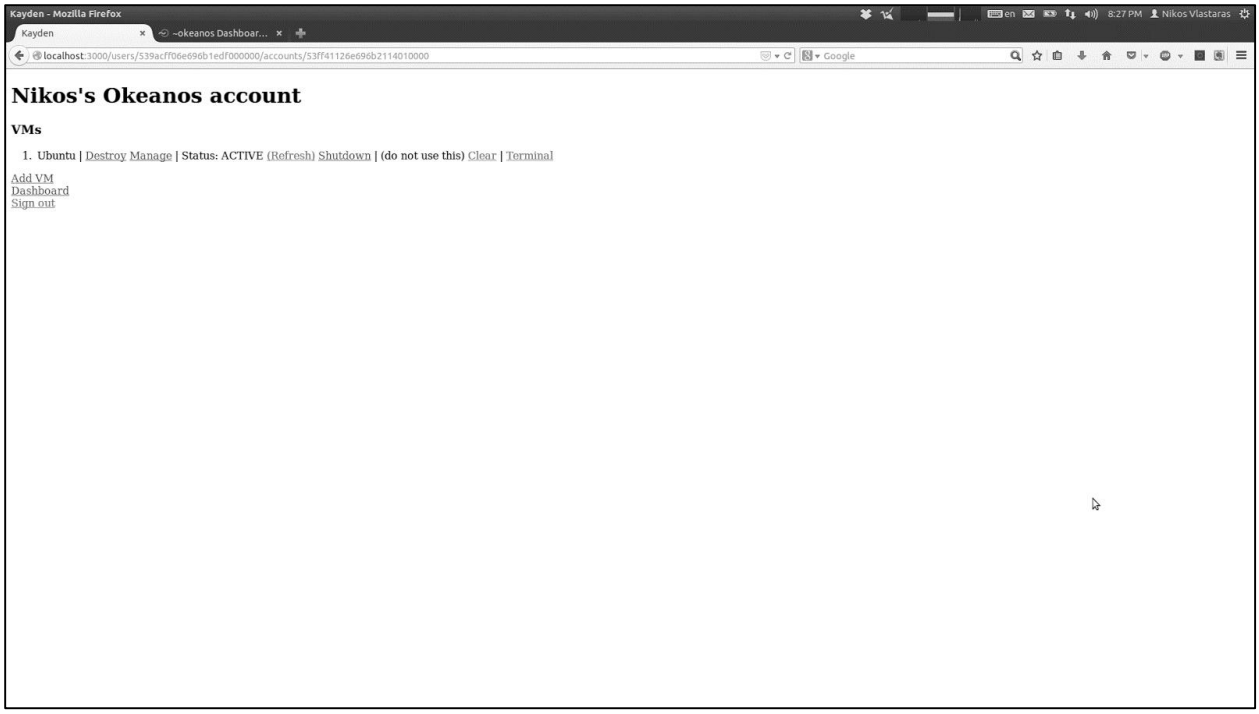


Εικόνα 5.16 - Εικονική μηχανή ενεργοποιείται

Μόλις ολοκληρωθεί η διαδικασία, μετά από ανανέωση της σελίδας, η εικονική μηχανή έχει κατάσταση ACTIVE.

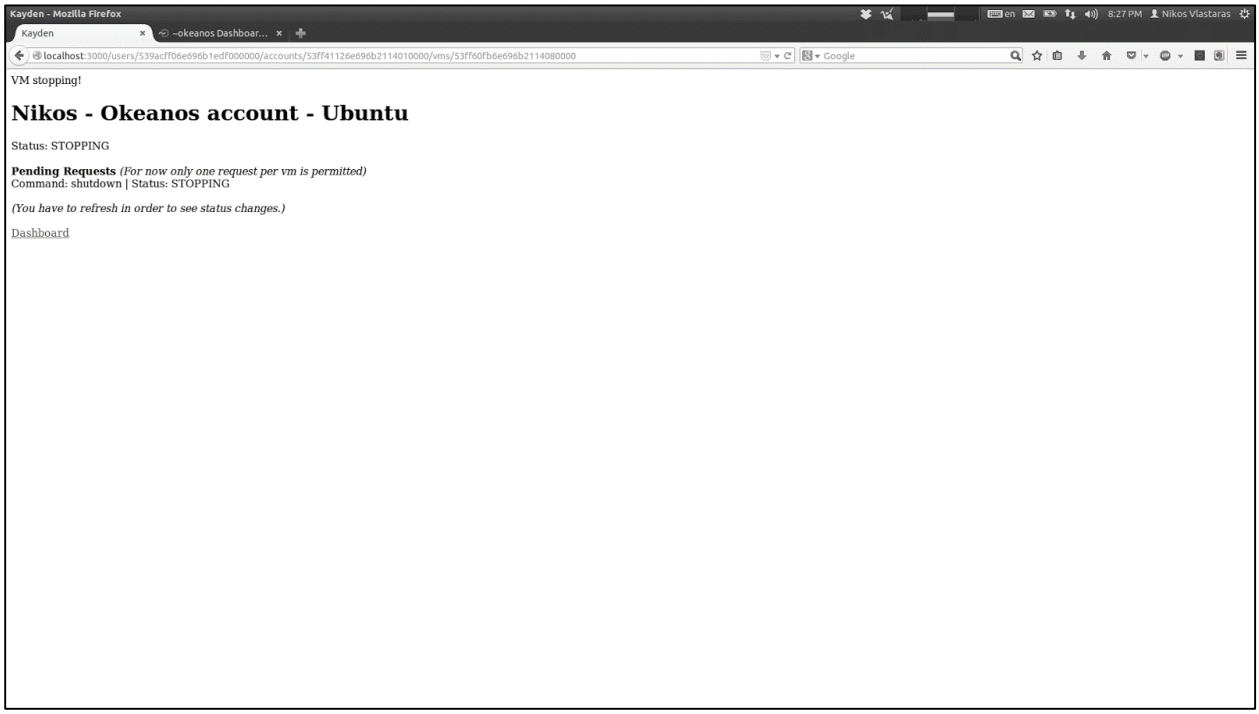
Τερματισμός εικονικής μηχανής

Από το Dashboard διαλέγουμε την επιλογή “Manage VMs” που βρίσκεται δίπλα από τον λογαριασμό στον οποίο θέλουμε να τερματίσουμε μια εικονική μηχανή.



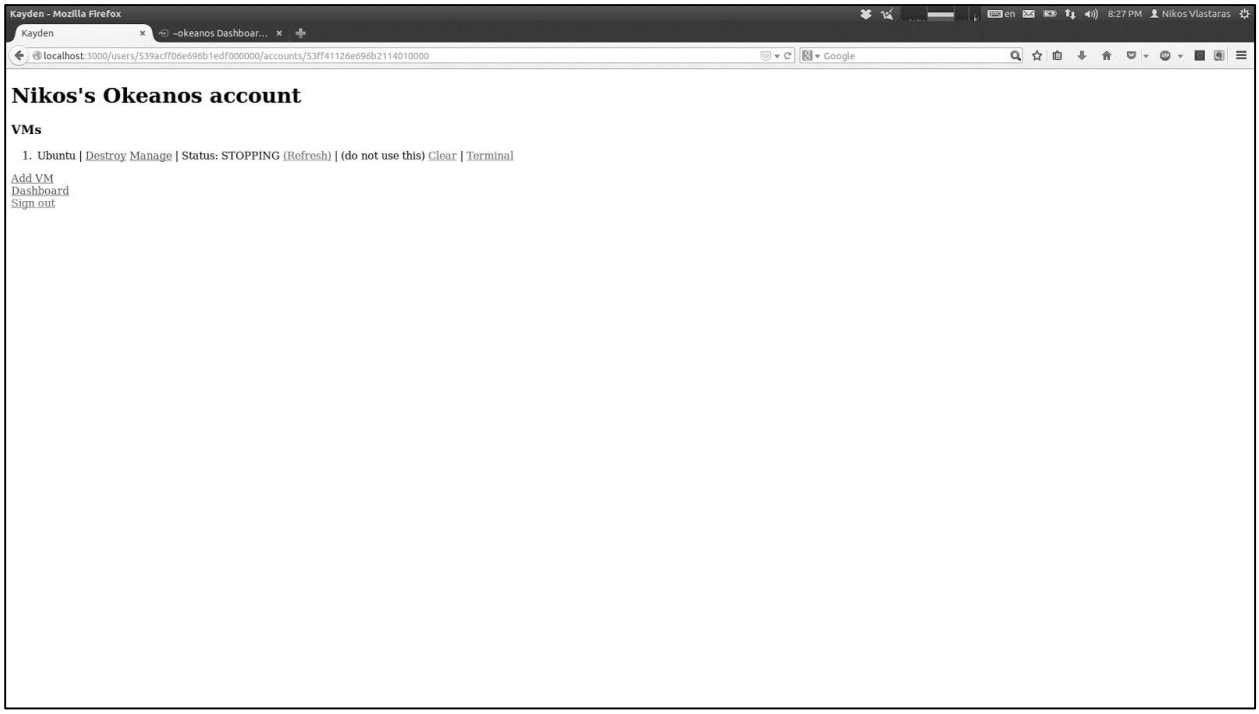
Εικόνα 5.17 - Εικονική μηχανή ενεργοποιημένη

Τότε επιλέγουμε το “Shutdown” δίπλα από την εικονική μηχανή που θέλουμε να τερματίσουμε και βρισκόμαστε στην οθόνη με τις πληροφορίες της εικονικής μηχανής όπου βλέπουμε να εκκρεμεί το αίτημα τερματισμού της. Πρέπει να σημειωθεί ότι για να υπάρχει αυτή η δυνατότητα πρέπει η κατάσταση της εικονικής μηχανής να είναι ACTIVE.



Εικόνα 5.18 - Επιβεβαίωση αιτήματος τερματισμού εικονικής μηχανής

Έπειτα διαλέγουμε την επιλογή Dashboard και στη συνέχεια την επιλογή “Manage VMs” δίπλα από τον λογαριασμό στον οποίο τερματίσαμε την εικονική μηχανή. Τότε βρισκόμαστε στην οθόνη που φαίνονται όλες οι εικονικές μηχανές του λογαριασμού και βλέπουμε στην κατάσταση της εικονικής μηχανής ότι ο τερματισμός της βρίσκεται σε εξέλιξη.

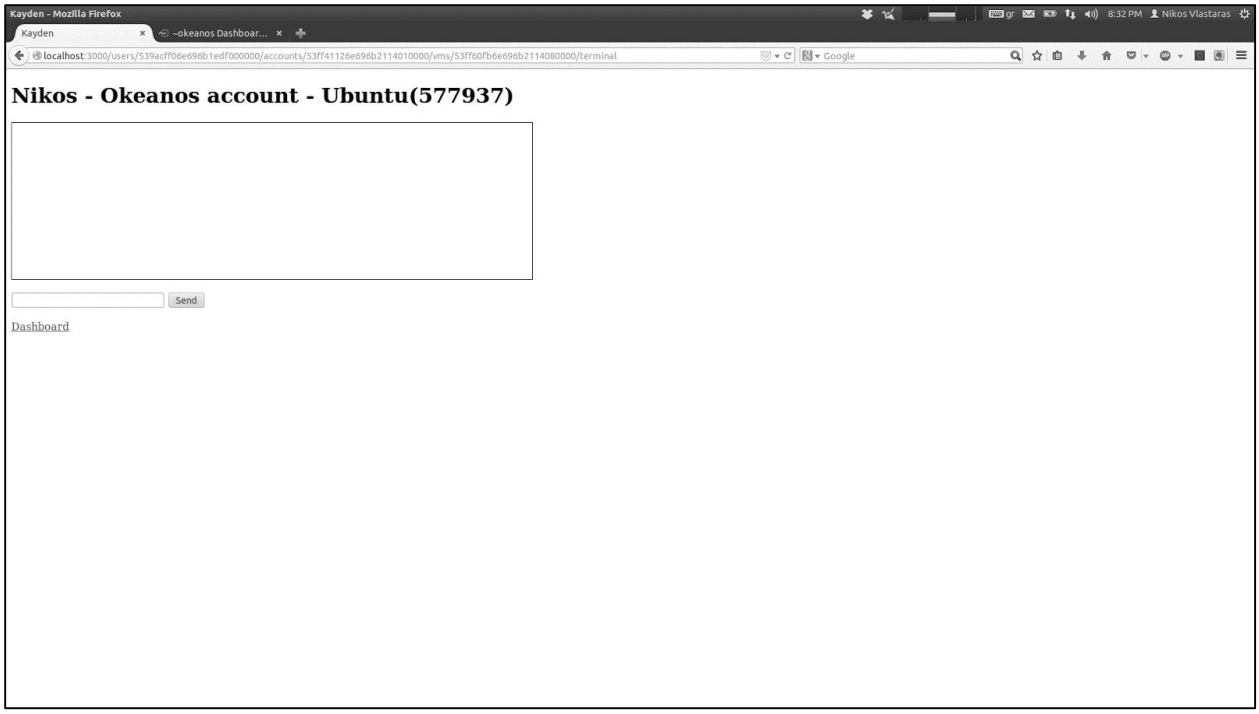


Εικόνα 5.19 - Εικονική μηχανή τερματίζει

Μόλις ολοκληρωθεί η διαδικασία, μετά από ανανέωση της σελίδας, η εικονική μηχανή έχει κατάσταση STOPPED.

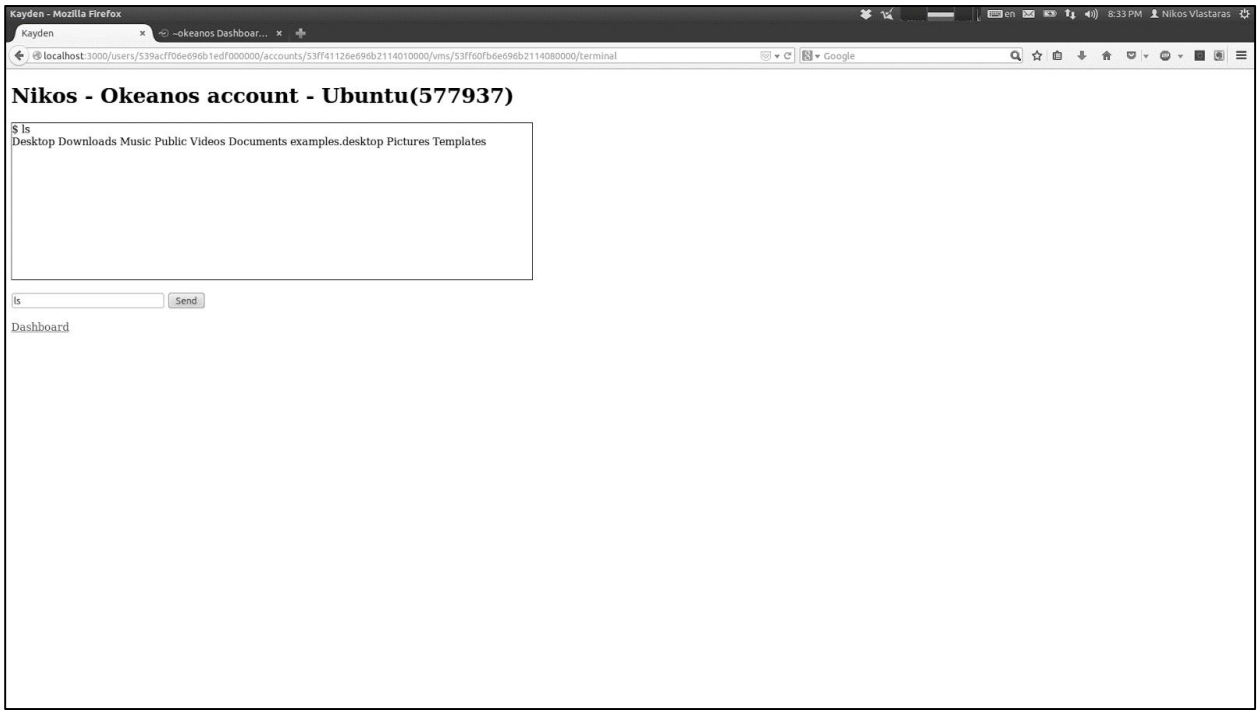
Τερματικό σε εικονική μηχανή

Από το Dashboard διαλέγουμε την επιλογή “Manage VMs” που βρίσκεται δίπλα από τον λογαριασμό στον οποίο θέλουμε να τερματίσουμε μια εικονική μηχανή.



Εικόνα 5.20 - Τερματικό σε εικονική μηχανή

Τότε επιλέγουμε το “Terminal” δίπλα από την εικονική μηχανή στην οποία θέλουμε να ανοίξουμε τερματικό και βρισκόμαστε στην οθόνη που προσομοιώνει το τερματικό.



Εικόνα 5.21 - Τερματικό σε εικονική μηχανή με αποτέλεσμα εντολής

Εδώ μπορούμε να βάλουμε μια εντολή, να πατήσουμε “Send” και μετά από λίγη ώρα να εμφανιστεί το αποτέλεσμα της εντολής

6
Επίλογος

6.1 Σύνοψη και Συμπεράσματα

Σύνοψη

Στην παρούσα διπλωματική εργασία μελετήθηκε η υποδομή υπηρεσιών cloud computing και αναπτύχθηκε μια πλατφόρμα για την διαχείρισή της.

Αρχικά εξετάστηκαν με ιδιαίτερη προσοχή οι αντίστοιχες πλατφόρμες των παρόχων υπηρεσιών cloud computing. Με αυτόν τον τρόπο καθορίστηκαν σε πρώτη φάση οι λειτουργικές απαιτήσεις της πλατφόρμας, δηλαδή ποιες λειτουργίες θα μπορεί να εκτελέσει. Σε πρώτη φάση επιλέχθηκε ένα πολύ βασικό σύνολο λειτουργιών για λόγους συμβατότητας με όλους τους παρόχους, το οποίο όμως επιτρέπει πλήρη έλεγχο της υποδομής. Επιπλέον, έπειτα από μια πιο διεξοδική εξέταση προέκυψε η ανάγκη για τον προσδιορισμό κάποιων μη λειτουργικών απαιτήσεων, οι οποίες αποτελούν κυρίως ποιοτικά χαρακτηριστικά της πλατφόρμας που επηρεάζουν την εμπειρία χρήσης. Τέτοιες είναι, μεταξύ άλλων, η συνέπεια και η ασφάλεια των δεδομένων που αποσκοπούν στη διαφάνεια της λειτουργίας της πλατφόρμας.

Στη συνέχεια, μετά από περαιτέρω ανάλυση των απαιτήσεων, λειτουργικών και μη, ξεκίνησε η διαδικασία σχεδιασμού της πλατφόρμας. Σε αυτό το σημείο χρειάστηκε να ληφθούν υπόψη και αρκετές παράμετροι υλοποίησης όπως η κλιμακωσιμότητα του συστήματος. Η πλατφόρμα χωρίστηκε σε δύο μέρη, το frontend και το backend, τα οποία ακολούθησαν ξεχωριστή πορεία ως προς την αρχιτεκτονική τους. Το frontend ακολούθησε το μοντέλο MVC ενώ το backend βασίστηκε σε μια αρχιτεκτονική βασισμένη σε plugins. Στο backend προσαρτήθηκε επίσης ένα τμήμα υπεύθυνο για configuration management με σκοπό την διευκόλυνση των χρηστών.

Λόγω του διαχωρισμού της πλατφόρμας σε δύο τμήματα, δόθηκε ιδιαίτερη βάση στον τρόπο επικοινωνίας τους. Σχεδιάστηκε, λοιπόν, ένα μοντέλο ανταλλαγής δεδομένων μεταξύ τους, το οποίο αποτελείται από δύο κανάλια. Το πρώτο επί της ουσίας είναι μια ουρά στην οποία μπαίνουν μηνύματα από το frontend προς το backend και το άλλο αφορά ένα REST API το οποίο χρησιμοποιείται από το backend για να παίρνει αλλά και να δίνει δεδομένα στο frontend. Ακόμα, σχεδιάστηκε ένα μοντέλο δεδομένων που αντικατοπτρίζει με σαφή και παραστατικό τρόπο τη σχέση εγκλεισμού μεταξύ εικονικών μηχανών και λογαριασμών σε παρόχους υπηρεσιών cloud computing και μεταξύ λογαριασμών και χρηστών.

Με την ολοκλήρωση του σχεδιασμού, διερευνήθηκαν όλες οι πιθανές προτάσεις υλοποίησης και επιλέχθηκαν οι πλατφόρμες και οι τεχνολογίες που ταίριαζαν καλύτερα. Για την υλοποίηση του frontend χρησιμοποιήθηκε το Ruby on Rails. Το Ruby on Rails είναι ένα πλαίσιο ανάπτυξης λογισμικού Ιστού ανοιχτού κώδικα για τη γλώσσα προγραμματισμού Ruby. Το Ruby on Rails για να οργανώσει τον προγραμματισμό των εφαρμογών χρησιμοποιεί αρχιτεκτονική Model-View-Controller (MVC), και για αυτό τον λόγο επελέγη ουσιαστικά. Ακόμα, για τη βάση δεδομένων

προτιμήθηκε NoSQL βάση δεδομένων, αντί σχεσιακής, και πιο συγκεκριμένα η MongoDB. Με αυτόν τον τρόπο εξασφαλίζεται ευελιξία και οριζόντια κλιμακωσιμότητα ενώ παράλληλα το γεγονός ότι είναι schema-free παρέχει στο σύστημα τη δυνατότητα προσαρμογής σε σχεδόν όλους τους παρόχους υπηρεσιών cloud computing. Ολοκληρώνοντας τις επιλογές εργαλείων λογισμικού πρέπει να αναφέρουμε το Chef. Το Chef είναι ένα εργαλείο configuration management γραμμένο σε Ruby και Erlang. Εδώ χρησιμοποιείται για να εγκαθίστανται εφαρμογές σε μια εικονική μηχανή κατά τη δημιουργία της.

Τέλος, παρέχεται ένας ενδελεχής και πλήρης οδηγός εγκατάστασης του συστήματος. Γίνεται ξεχωριστή αναφορά για την εγκατάσταση του frontend και του backend και για το καθένα αναφέρονται λεπτομερώς τα πακέτα λογισμικού που απαιτείται να εγκατασταθούν για την λειτουργία τους καθώς και αναλυτικά βήματα για το πως θα γίνει αυτό. Επιπλέον, δίνονται και λεπτομερείς οδηγίες για την εγκατάσταση των δύο τμημάτων (frontend και backend) σε ξεχωριστά μηχανήματα καθώς και για το πως πρέπει να ρυθμιστεί η μεταξύ τους επικοινωνία. Στη συνέχεια, παρουσιάζεται και η ροή εκτέλεσης του συστήματος. Για κάθε σενάριο χρήσης/λειτουργίας παρουσιάζεται όλη η διεπαφή χρήστη-συστήματος αναλυτικά ενώ εξηγείται πως πρέπει να ενεργήσει ο χρήστης για να έχει τα αποτελέσματα που περιγράφονται. Για καλύτερη κατανόηση παρέχονται και screenshots με τις οθόνες του συστήματος για κάθε βήμα.

Συμπεράσματα

Με την ολοκλήρωση του πρώτου κύκλου ανάπτυξης της πλατφόρμας είναι χρήσιμο να εξαχθούν κάποια συμπεράσματα. Το κυριότερο συστατικό της είναι το REST API που προσφέρουν οι πάροχοι υπηρεσιών cloud computing και ουσιαστικά σε αυτό οφείλει την ύπαρξή της. Το REST API καθιστά πολύ εύκολη τη διαχείριση των εικονικών μηχανών καθώς παρέχει πολύ απλές και κυρίως περιγραφικές εντολές για τις διάφορες λειτουργίες που μπορούν να γίνουν πάνω στις εικονικές μηχανές.

Ακόμα, το REST API αποτελεί την πιο εύκολη λύση διαχείρισης εικονικών μηχανών όσον αφορά τις ανάγκες αυτού του εγχειρήματος. Οι πάροχοι δίνουν την δυνατότητα αλληλεπίδρασης με τις εικονικές μηχανές και μέσω κάποιου web gui όμως αυτό δεν εξυπηρετεί τις απαιτήσεις της πλατφόρμας που υλοποιήθηκε σε αυτή την εργασία.

Ένα ακόμα θετικό του REST API είναι το ότι είναι ανεξάρτητο από το λειτουργικό σύστημα. Αυτό σημαίνει ότι με την υλοποίηση του κατάλληλου client μπορεί να γίνει αξιοποίηση του από κάθε σύστημα.

Επιπλέον, αν και στη συγκεκριμένη υλοποίηση χρησιμοποιήθηκε το API της υπηρεσίας IaaS Cloud Computing Okeanos, προβλέπεται και η υποστήριξη και άλλων παρόχων. Αυτή η δυνατότητα οφείλεται στις σχετικές ομοιότητες που παρουσιάζουν μεταξύ τους τα API των παρόχων από άποψη δυνατοτήτων αλλά και από άποψη εντολών.

Τέλος, πέραν του REST API, πρέπει να σημειωθεί η ευκολία εγκατάστασης λογισμικού με τη χρήση του Chef. Το Chef δίνει τη δυνατότητα, αφού αποκτηθούν τα κατάλληλα cookbooks από την κοινότητά του, να εγκατασταθούν τα επιθυμητά πακέτα λογισμικού αφού πρώτα καταγραφούν σε μια λίστα. Επιπλέον, προσφέρεται και ένα εργαλείο, το knife, το οποίο αναλαμβάνει την εγκατάσταση του Chef στην εικονική μηχανή προτού εγκαταστήσει τα πακέτα.

Η πλατφόρμα που υλοποιήθηκε εκμεταλλεύεται αυτά τα πλεονεκτήματα και παρέχει στον χρήστη τη δυνατότητα να επιλέγει από μια λίστα με όλα τα διαθέσιμα πακέτα αυτά που θέλει και να αναλαμβάνει αυτή να χρησιμοποιήσει το knife για να εγκαταστήσει Chef και τα πακέτα.

6.2 Μελλοντικές Επεκτάσεις

Η πλατφόρμα διαχείρισης υποδομής cloud computing υπηρεσιών που υλοποιήθηκε σε αυτή την εργασία αποτελεί μια λύση που λειτουργεί ολοκληρωμένα και ανεξάρτητα, ωστόσο επιδέχεται επεκτάσεις και βελτιώσεις. Συγκεκριμένα θα μπορούσαν να γίνουν δύο βελτιώσεις που θα αύξαναν πάρα πολύ τη λειτουργικότητα της πλατφόρμας.

Πρώτη επέκταση θα ήταν η υλοποίηση ενός συστήματος άμεσου monitoring των εικονικών μηχανών. Αυτή η επέκταση θα είχε ως σκοπό να καταγράφει στατιστικά στοιχεία για τη μηχανή όπως η χρησιμοποίηση του cpu, της μνήμης, του δίσκου και του δικτύου. Αυτά τα στοιχεία θα μπορούσαν να χρησιμεύσουν σε πρώτη φάση για να προφυλάσσεται ο χρήστης, από υπερβολικές χρεώσεις. Σε αυτό θα μπορούσε να βοηθήσει η υλοποίηση ενός τμήματος στο οποίο να τίθενται κανόνες που να αφορούν συγκεκριμένα όρια χρήσης στους πόρους και αν αυτά ξεπεραστούν να ενημερώνεται ο χρήστης (mail, sms κα) ή ακόμα και να διακόπτεται η λειτουργία της μηχανής αυτόματα. Επιπλέον, τα στατιστικά αυτά στοιχεία, θα μπορούσαν να χρησιμοποιηθούν και για να διασταυρωθούν με τα αντίστοιχα που μετράει ο πάροχος έτσι ώστε να επικυρωθεί η σωστή χρέωση των παρεχόμενων υπηρεσιών.

Η συλλογή των στατιστικών στοιχείων με υψηλή ακρίβεια που περιγράφηκαν παραπάνω είναι δύσκολη. Η πρώτη προσέγγιση, που είναι η αποστολή ερωτημάτων από το σύστημα στην εικονική μηχανή για αυτά τα στοιχεία και καταγραφή τους στη βάση δεδομένων του συστήματος, δεν παρέχει αρκετές εγγυήσεις ακρίβειας. Επιπλέον, δεν είναι ρεαλιστικό να αποστέλλει το σύστημα ερωτήματα σε όλες τις εγγεγραμμένες σε αυτό εικονικές μηχανές και μόνο από άποψη φόρτου. Επομένως, στα πλαίσια αυτής της λειτουργίας, θα ήταν χρήσιμη και η υλοποίηση ενός agent ο οποίος θα εγκαθίσταται στην εικονική μηχανή αμέσως μετά τη δημιουργία της. Κύρια ευθύνη αυτού του λογισμικού θα ήταν η συλλογή των στατιστικών στοιχείων με υψηλή ακρίβεια, η τοπική αποθήκευσή τους καθώς και η διάθεσή τους στον χρήστη όταν αυτός τα ζητήσει μέσω καινούργιας λειτουργίας του συστήματος.

Δεύτερη επέκταση θα ήταν η παροχή δυνατότητας συνεργασίας των εικονικών μηχανών μεταξύ τους, υπό την επίβλεψη του συστήματος. Αυτό σημαίνει ότι κάποιος, άνθρωπος ή οργανισμός, θα μπορούσε να “συναρμολογήσει” ένα ολόκληρο πληροφοριακό σύστημα έτοιμο να εξυπηρετήσει πραγματικές ανάγκες. Προς αυτή την κατεύθυνση θα μπορούσε να βοηθήσει και το τμήμα configuration management που έχει ήδη υλοποιηθεί. Με κατάλληλη επέκτασή του θα μπορούσαν να συνδυαστούν πακέτα λογισμικού με ρόλους, όπως Web Server, Database Server κτλ. Έτσι θα δημιουργούνταν έτοιμα blueprints και ο χρήστης θα μπορούσε να ζητήσει τη κατευθείαν τη δημιουργία ενός μηχανήματος με συγκεκριμένο configuration για έναν συγκεκριμένο ρόλο. Τα παραπάνω δεν αφορούν μόνο εικονικές μηχανές που υπάγονται στον ίδιο λογαριασμό, ούτε καν μόνο μηχανές που ανήκουν στον ίδιο πάροχο, αλλά όλες όσες έχουν καταγραφεί σε έναν χρήστη του συστήματος. Αυτό ευνοεί και τη ανάπτυξη στρατηγικής από

πλευράς χρηστών όσον αφορά την επιλογή παρόχων, με βάση την τιμολογιακή πολιτική τους σε σχέση με τις προσφερόμενες υπηρεσίες.

Για την διευκόλυνση του παραπάνω θα ήταν χρήσιμη και η ανάπτυξη ενός αντίστοιχου γραφικού περιβάλλοντος για πιο εύκολη και φιλική προς τον χρήστη αλληλεπίδρασή. Φυσικά, θα μπορούσαν όλα να γίνονται με εντολές, αντίστοιχες με εντολές τερματικού, αλλά αυτό θα χρειαζόταν μεγάλη εξοικείωση ενώ υστερεί και στην εποπτεία που προσφέρει. Αντίθετα η χρήση ενός γραφικού περιβάλλοντος, αναπτυγμένο έτσι ώστε να έχει ομοιότητες με προγράμματα σχεδιασμού (όπως το MS Visio) λόγω του άμεσου χειρισμού που προσφέρουν, θα ήταν πιο διαισθητική καθώς όλα θα αντιστοιχούσαν σε εικόνες. Οι εικονικές μηχανές θα είχαν συγκεκριμένα εικονίδια, πάνω στα οποία θα αναγράφονταν κάποια στοιχεία όπως π.χ. ο ρόλος τους, ενώ και συνδέσεις μεταξύ τους θα συμβολίζονταν με γραμμές τις οποίες θα μπορούσε να τραβήξει ο χρήστης από το ένα μηχάνημα στο άλλο.

Τέλος, η πλατφόρμα θα μπορούσε να επωφεληθεί σημαντικά, κυρίως ως προς την εμπειρία χρήσης, και από κάποιες πιο μικρές βελτιώσεις. Μια τέτοια θα ήταν η αυτόματη ανανέωση του token για κάθε λογαριασμό. Τα token ασφαλείας ανανεώνονται από τον πάροχο ανά κάποιο χρονικό διάστημα, τα παλιά παύουν να ισχύουν και αυτό, αν δεν γίνει αντιληπτό άμεσα από τον χρήστη προκαλεί προβλήματα λειτουργίας. Στην παρούσα κατάσταση η ευθύνη ανήκει στον χρήστη να το αντιληφθεί και να το ανανεώσει αλλά θα μπορούσε να υλοποιηθεί ένας μηχανισμός που να διατηρεί τον χρόνο που απομένει μέχρι να λήξει το token και όταν αυτό συμβεί να πηγαίνει και να διαβάζει το καινούργιο. Αυτό βέβαια περιέχει και πολλά ζητήματα ασφάλεια που απαιτούν λεπτό χειρισμό και για αυτό για να γίνει οποιαδήποτε απόπειρα υλοποίησης θα έπρεπε να δοθεί μεγάλη βάση σε αυτόν τον τομέα.

Μια ακόμα αντίστοιχου βελησνεκούς μικρο-βελτίωση θα ήταν και η υλοποίηση ενός μηχανισμού συγχρονισμού μεταξύ της πλατφόρμας και του κάθε λογαριασμού σε πάροχο υπηρεσιών cloud computing. Στο παρόν στάδιο, έχει γίνει η παραδοχή ότι κάθε λογαριασμός που έχει προστεθεί στο σύστημα δεν επιτρέπεται να τροποποιηθεί εκτός της πλατφόρμας (π.χ. από το γραφικό περιβάλλον που προσφέρει ο πάροχος). Αυτό έδωσε τη δυνατότητα να προχωρήσει η ανάπτυξη της πλατφόρμας όμως δεν είναι ρεαλιστικό. Έτσι, ένας τέτοιος μηχανισμός θα μπορούσε κάθε φορά που συνδέεται ένας χρήστης να διερευνά για κάθε λογαριασμό του κατά πόσο η εικόνα που έχει για αυτόν, που βρίσκεται αποθηκευμένη στη βάση δεδομένων του, είναι αληθής και αν εντοπίσει διαφορές να τροποποιεί την προαναφερθείσα βάση αναλόγως.

Βιβλιογραφία

[1] Gerard Conway and Edward Curry. “Managing cloud computing: A life cycle approach.”

[2] Mell, Peter, and Timothy Grance. ”The NIST definition of cloud computing (draft).” NIST special publication 800.145 (2011): 7.

[3] Marston, Sean, et al. ”Cloud computing—The business perspective.” Decision Support Systems 51.1 (2011): 176-189.

[4] Pearson, Siani, and Azzedine Benameur. ”Privacy, security and trust issues arising from cloud computing.” Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on. IEEE, 2010.

[5] Η ιστορία του υπολογιστικού νέφους, http://en.wikipedia.org/wiki/Cloud computing#History

[6] Okeanos IaaS, https://okeanos.grnet.gr/home/

[7] Η βάση δεδομένων mongoDB, http://www.mongodb.org/

[8] Το web framework Ruby on Rails, http://rubyonrails.org/

[9] Το μεσισμικό μηνυμάτων RabbitMQ, http://www.rabbitmq.com/

[10] Το λογισμικό configuration management Chef, http://www.getchef.com/

[11] REST from Wikipedia http://en.wikipedia.org/wiki/Representational_State_Transfer

[12] Jeffery K, Schubert H and Neidecker-Lutz B, \The Future for Cloud Computing: Opportunities for European Cloud Computing Beyond 2010", Expert Group report, public version 1.0, January 2010.

[13] Vaquero, Luis M., et al. "A break in the clouds: towards a cloud definition." ACM SIGCOMM Computer Communication Review 39.1 (2008): 50-55.

[14] Pearson, Siani, and Azzedine Benameur. "Privacy, security and trust issues arising from cloud computing." Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on. IEEE, 2010.

[15] Dillon, Tharam, Chen Wu, and Elizabeth Chang. "Cloud computing: Issues and challenges." Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on. Ieee, 2010.

[16] Zhang, Qi, Lu Cheng, and Raouf Boutaba. "Cloud computing: state-of-the-art and research challenges." Journal of Internet Services and Applications 1.1 (2010): 7-18.

[17] Ye, Kejiang, et al. "Live migration of multiple virtual machines with resource reservation in cloud computing environments." Cloud Computing (CLOUD), 2011 IEEE International Conference on. IEEE, 2011.

[18] Tudorica, Bodgan George, and Cristian Bucur. "A comparison between several NoSQL databases with comments and notes." Roedunet International Conference (RoEduNet), 2011 10th. IEEE, 2011.

[19] Ru Iosup and Simon Ostermann and Nezih Yigitbasi and Radu Prodan and Thomas Fahringer and Dick Epema. An early performance analysis of cloud computing services for scientific computing. TU Delft, Tech. Rep., Dec 2008, [Online] Available.

[20] Peter Mell, NIST. Big Data Tradeoffs: What Agencies Need To Know. http://breakinggov.com/2012/11/12/big-data-tradeoffs-what-agencies-need-to-know-nists-peter-me/

[21] Markus BÖhm, Stefanie Leimeister, Christoph Riedl, Helmut Krcmar. Cloud Computing and Computing Evolution.

[22] G. Burd. NoSQL: An Overview of NoSQL Databases. April 2012.

[23] Berriman, G. Bruce. "The Application of Cloud Computing to Scientific Workflows: A Study of Cost and Performance." *Philosophical Transactions: Mathematical* Physical and Engineering Sciences.Vol. 371, No. 1983, E-Science-towards the Cloud: Infrastructures, Applications and Research (2013): 1-14. *JSTOR*. Web. 03 Sept. 2014.

[24] Reese, George. *Cloud Application Architectures*. Sebastopol, CA: O'Reilly, 2009. Print.

[25] Copeland, Rick. *MongoDB Applied Design Patterns*. Sebastopol, CA: O'Reilly Media, 2013. Print.

[26] Chodorow, Kristina. *MongoDB: The Definitive Guide:*. Beijing: O'Reilly, 2013. Print.

[27] Keyes, Jessica. *Software Configuration Management*. Boca Raton, FL: Auerbach Publications, 2004. Print.

[28] Sabharwal, N. *Automation through Chef Opscode: A Hand-on Approach to Infrastructure Automation, Devops Automation, and Reporting through Chef*. Berkeley: Apress, 2014. Print.

[29] *Learning Chef*. N.p.: Oreilly & Associates, 2014. Print.

[30] Hartl, Michael. *Ruby on Rails Tutorial: Learn Web Development with Rails*. N.p.: n.p., n.d. Print.

[31] Fernandez, Obie, and Kevin Faustino. *The Rails 4 Way*. N.p.: n.p., n.d. Print.

[32] Ruby, Sam, David Thomas, David Heinemeier Hansson, and Davidson Pfalzer. *Agile Web Development with Rails 4*. Raleigh, N.C.: Pragmatic helf, 2013. Print.

[33] Wintermeyer, Stefan. *Ruby on Rails 4.0 Guide: A Step by Step Guide to Learn Ruby on Rails 4.0 and Ruby 2.0*. Lexington, KY: CreateSpace Independent Platform, 2013. Print.

[34] Gamma, Erich. *Design Patterns: Elements of Reusable Object-oriented Software*. Reading, Massachusetts: Addison-Wesley, 1997. Print.

[35] Burris, Eddie. *Programming in the Large with Design Patterns*. Leawood, Kan: Pretty Print, 2012. Print.

[36] Bass, Len, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Upper Saddle River, NJ: Addison-Wesley, 2013. Print.

Παράρτημα Α - Κώδικας Backend

backend_init.rb

```
#!/usr/bin/env ruby
# encoding: utf-8

require 'bunny'
load "backend_server.rb"

conn = Bunny.new
conn.start

server_url = "http://localhost:3000/"
queue = "from_server"

ch = conn.create_channel
q = ch.queue(queue)

begin
    server = BackendServer.new(server_url,q)
    server.init
rescue Interrupt, StandardError => e
    puts "\n"
    puts e.inspect
    ch.close
    conn.close
    exit
end
```

backend_server.rb

```
require "bunny"
require "net/ping"
require "pp"
require "json"
require 'socket'
require 'timeout'
load "okeanos plugin.rb"

class BackendServer

    def initialize(server_url,q)
        @server_url = server_url
        @q = q
        @ping_timeout = 120
        @kitchen_dir = "kitchen/"
        @knife_dir = "kitchen/"
    end

    def init
        puts " [x] Awaiting requests"
        @q.subscribe(:block => true) do |delivery_info, properties, payload|
            reqID = payload
            puts " [x] Received request #{reqID}"
            # Retrieve request
            request = get_request(reqID)
            # Instantiate cloud provider object. Pass the request and it will use what it needs.
            @s = Object::const_get(request["provider"].capitalize).new(request)
            @user = "user"
            eval "#{request["command"]}(request,reqID)"
            puts " [x] Awaiting requests"
        end
    end
```

```
private

    ## Helper functions

    def get_request(reqID)
        resp = HTTParty.get(@server_url + "requests/#{reqID}")
        request = JSON.parse(resp.body)
        return request
    end

    def update_request(reqID,msgs=nil,stat=nil,vm=nil)
        options = { :headers => { "Content-Type" => "application/json" },
        :body    => { "request" => {} }}
        if msgs
            options[:body]["request"]["messages"] = msgs
        end
        if stat
            options[:body]["request"]["status"] = stat
        end
        if vm
            options[:body]["request"]["vm"] = vm
        end
        options[:body] = JSON.generate(options[:body])
        resp = HTTParty.patch(@server_url+"requests/#{reqID}",options)

        return resp
    end

    def destroy_request(reqID)
        resp = HTTParty.delete(@server_url+"requests/#{reqID}")
        return resp
    end

    def is_port_open?(ip, port)
        begin
            Timeout::timeout(1) do
                begin
                    s = TCPSocket.new(ip, port)
                    s.close
                    return true
                rescue Errno::ECONNREFUSED, Errno::EHOSTUNREACH
                    return false
                end
            end
        rescue Timeout::Error
        end

        return false
    end

    ## VM actions

    def start(request,reqID)
        vm = request["vm"]
        resp = @s.start(vm["providerID"])
        puts " [x] Starting vm #{vm["providerID"]}"
        update_request(reqID,(request["messages"] << "Starting vm #{vm["providerID"]}"))

        resp = @s.status(vm["providerID"])
        while (resp["server"]["status"] != "ACTIVE")
            print "."
            sleep 10
            resp = @s.status(vm["providerID"])
        end
        print "\n"

        # Second request update
        puts " [x] VM started"
        update_request(reqID,(request["messages"] << "VM started"),resp["server"]["status"])
        puts " [x] Checking VM availability"
```

```
            p = Net::Ping::External.new(vm["hostname"])
            time = 0
            while (time < @ping_timeout && !p.ping)
                print "."
                sleep 10
                time = time + 10
            end
            print "\n"

            if time >= @ping_timeout
                msg = "Ping timed out. Check manually"
                puts msg
                # stat = "ACTIVE-ONLINE(?)"
            else
                msg = "VM available"
                puts msg
                # stat = "ACTIVE-ONLINE"
            end
            update_request(reqID,(request["messages"] << msg))
            destroy_request(reqID)
            return 0
        end

        def shutdown(request,reqID)
            vm = request["vm"]
            resp = @s.shutdown(vm["providerID"])
            puts " [x] Stopping vm #{vm["providerID"]}"
            update_request(reqID,(request["messages"] << "Starting vm #{vm["providerID"]}"))

            resp = @s.status(vm["providerID"])
            while (resp["server"]["status"] != "STOPPED")
                print "."
                sleep 10
                resp = @s.status(vm["providerID"])
            end
            print "\n"

            puts " [x] VM stopped"
            update_request(reqID,(request["messages"] << "VM stopped"),resp["server"]["status"])
            destroy_request(reqID)
            return 0
        end

    def create(request,reqID)
        vm = request["vm"]
        resp = @s.create(vm)
        # HANDLE ERROR: create response (202 -> Success)
        # puts "CODE: #{resp.header.code}"
        # puts "MESSAGE: #{resp.header.message}"
        vm["providerID"] = resp["server"]["id"]
        vm["password"] = resp["server"]["adminPass"]
        vm["hostname"] = resp["server"]["SNF:fqdn"]
        vm["username"] = "user"
        request["vm"] = vm

        # First request update
        puts " [x] Waiting for vm to be created"
        update_request(reqID,(request["messages"] << "Waiting for vm to be
created"),resp["server"]["status"],vm)

        resp = @s.status(vm["providerID"])
        while (resp["server"]["status"] != "ACTIVE")
            print "."
            sleep 10
            resp = @s.status(vm["providerID"])
        end
        print "\n"

        # Second request update
        puts " [x] VM created"
```

```
        update_request(reqID,(request["messages"] << "VM created"),resp["server"]["status"])

        puts " [x] Checking vm availability"

        p = Net::Ping::External.new(vm["hostname"])
        time = 0
        while (time < @ping_timeout && !p.ping)
            print "."
            sleep 10
            time = time + 10
        end
        print "\n"

        # Third request update
        if time >= @ping_timeout
            msg = "Ping timed out. Check manually"
            puts msg
            # stat = "ACTIVE-ONLINE(?)"
        else
            msg = "VM available"
            puts msg
            # stat = "ACTIVE-ONLINE"
        end

        #FIXME
        if vm["recipes"]
            bootstrap(request,reqID)
        end

        update_request(reqID,(request["messages"] << msg))
        destroy_request(reqID)
        return 0
    end

    def destroy(request,reqID)
        vm = request["vm"]
        resp = @s.destroy(vm["providerID"])
        puts "Waiting for vm to be deleted"
        update_request(reqID,(request["messages"] << "Waiting for vm to be deleted"),"DELETING")

        resp = @s.status(vm["providerID"])
        while (resp["server"]["status"] != "DELETED")
            print "."
            sleep 10
            resp = @s.status(vm["providerID"])
        end
        print "\n"

        # Second request update
        puts "VM deleted"
        update_request(reqID,(request["messages"] << "VM deleted"),resp["server"]["status"])
        destroy_request(reqID)
        return 0
    end

    def status(request,reqID)
        vm = request["vm"]
        resp = @s.status(vm["providerID"])
        puts " [x] Got vm status"
        update_request(reqID,(request["messages"] << "Got vm status"),resp["server"]["status"])
        destroy_request(reqID)
        return 0
    end

    def ssh(request,reqID)
        vm = request["vm"]
        puts " [x] Checking port 22 availability"
        time = 0
        while (time < @ping_timeout && !is_port_open?(vm["hostname"],22))
            print "."
            sleep 10
```

```
            time = time + 10
        end
        print "\n"

        if time >= @ping_timeout
            puts "Check timed out. Chef aborted"
            return 1
        else
            resp = @s.ssh_cmd(vm["hostname"],vm["username"],vm["password"],request["ssh"])
            puts " [x] Got ssh response"
            # p resp
            update_request(reqID,(request["messages"] << resp))
            destroy_request(reqID)
            return 0
        end
    end

    def bootstrap(request,reqID)
            vm = request["vm"]
            puts " [x] Checking port 22 availability"
            time = 0
      while (time < @ping_timeout && !is_port_open?(vm["hostname"],22))
        print "."
        sleep 10
        time = time + 10
      end
      print "\n"

      if time >= @ping_timeout
        puts "Check timed out. Chef aborted"
      else
        puts " [x] Configuring node file for #{vm["hostname"]}"
          node = { "\"run_list\"" => vm["recipes"].keys.map { |k| "\"recipe[#{k}]\"" } }
          # puts "echo #{node.to json} > #{@kitchen dir}nodes/#{vm["hostname"]}.json"
          value = %x[ echo #{node.to_json} > #{@kitchen_dir}nodes/#{vm["hostname"]}.json ]
          puts " [x] Running knife"
          #Dir.chdir(@knife_dir) {
            # puts " [x] ./knife.exp #{@user} #{vm["hostname"]} #{vm["password"]} #{@knife dir}"
            system("./knife.exp #{@user} #{vm["hostname"]} #{vm["password"]} #{@knife_dir}")
          #}
      end
    end
end
```

okeanos_plugin.rb

```
require 'httparty'
require 'net/ssh'

class Okeanos

    def initialize(request)
        @token = request["token"]
        @url = "https://cyclades.okeanos.grnet.gr/compute/v2.0"
    end

    def start(id)
        query = "/servers/#{id}/action"
        options = {:headers => {"X-Auth-Token"   => @token,
                                "Content-Type"   => "application/json",
                                "Content-Length" => "32"},
                   :body    => "{\"start\": {}}"}
        resp = HTTParty.post(@url + query, options)
        return resp
    end

    def shutdown(id)
        query = "/servers/#{id}/action"
        options = {:headers => {"X-Auth-Token"   => @token,
```

```
                                "Content-Type"   => "application/json",
                                "Content-Length" => "32"},
                :body    => "{\"shutdown\": {}}"}
        resp = HTTParty.post(@url + query, options)
        return resp
    end

    def create(vm)
        flavor = get_flavor(vm["cpu"],vm["memory"],vm["disk"])
        n = []
        vm["ips"].each do | ip |
            ip =~ /^(.*)::(.*)$/
            n << { "uuid" => $2.to_i, "fixed_ip" => $1 }
        end
        networks = JSON.generate(n)
        query = "/servers"
        options = {:headers => {"X-Auth-Token"   => @token,
                                "Content-Type"   => "application/json",
                                "Content-Length" => "735"},
                   :body    => "{
                                  \"server\": {
                                     \"name\"     : \"#{vm["name"]}\",
                                     \"imageRef\" : \"#{vm["image"]}\",
                                     \"flavorRef\": #{flavor.to_i},
                                     \"metadata\" : {
                                        \"EloquentDescription\": \"\",
                                        \"ShortDescription\"   : \"\"
                                     },
                                     \"networks\" : " + networks + "}}"}
        resp = HTTParty.post(@url + query, options)
        return resp
    end

    def destroy(id)
        query = "/servers/#{id}"
        options = {:headers => {"X-Auth-Token"   => @token}}
        resp = HTTParty.delete(@url + query, options)
        return resp
    end

    def status(id)
        query = "/servers/#{id}"
        options = {:headers => {"X-Auth-Token"   => @token}}
        resp = HTTParty.get(@url + query, options)
        return resp
    end

    def list
        query = "/servers/detail"
        options = {:headers => {"X-Auth-Token" => @token}}
        resp = HTTParty.get(@url + query, options)
        return resp
    end

    def get_ip(id)
        query = "/servers/#{id}/ips"
        options = {:headers => {"X-Auth-Token" => @token}}
        resp = HTTParty.get(@url + query, options)
        str = ""
        resp["addresses"].each { |k, v| str = str + "IPv#{v[0]["version"]}: #{v[0]["addr"]}\n" }
        return resp
    end

    def ssh_cmd(host,user,password,cmd)
        dt = ""
        Net::SSH.start(host, user, { :password => password, :paranoid => false }) do |ssh|
            ssh.open_channel do |ch|
                ch.request_pty do |c, success|
                    raise "could not request pty" unless success
                    ch.exec cmd
                    ch.on_data do |c, data|
```

```
                        if data =~ /\[sudo\]/ || data =~ /Password/i
                            ch.send_data "#{password}\n"
                        else
                            #$stdout.print data
                            dt = dt + data.to_s
                        end
                    end
                    ch.on_extended_data do |c, type, data|
                        #stderr.print data
                        dt = dt + data.to_s
                    end
                end
            end
        end
        return dt
    end

    private

        def get_flavor(cpu,memory,disk)
            resp = HTTParty.get("https://cyclades.okeanos.grnet.gr/compute/v2.0/flavors/detail",
                                { :headers => { "X-Auth-Token" => @token } })
            fl = JSON.parse(resp.body)
            fl["flavors"].each do | f |
                if f["name"] == "C#{cpu}R#{memory}D#{disk}ext_vlmc" # Add support for storage
system
                    return f["id"]
                end
            end
            return -1 # Control flow should not reach this point
        end

end
```

knife.exp

```
#!/usr/bin/expect -f


set user [lindex $argv 0];
set host [lindex $argv 1];
set pass [lindex $argv 2];
set kdir [lindex $argv 3];

set timeout -1
set err 0

cd $kdir
spawn knife solo bootstrap $user@$host
expect {
    yes/no  { send yes\n ; exp_continue }
    assword { send $pass\n ; exp_continue }
    ERROR:  { set err 1 ; exp_continue }
    "$ "    {}
}
exit
```

Παράρτημα Β - Κώδικας Frontend

Models

account.rb

```
class Account
  include Mongoid::Document
  include Mongoid::Timestamps

  validates :username, presence: true
  validates :provider, presence: true
  validates :token,    presence: true

  embedded_in :user, inverse_of: :accounts
  embeds_many :vms

  field :username, type: String
  field :provider, type: String
  field :token,    type: String

end
```

cookbook.rb

```
class Cookbook
  include Mongoid::Document

  field :name   , type: String
  field :recipes, type: Array
end
```

prototype.rb

```
class Prototype
  include Mongoid::Document

  field :provider,   type: String
  field :parameters, type: Array
end
```

request.rb

```
class Request
  include Mongoid::Document
  include Mongoid::Timestamps

  before_save :check_unique, :exclude_id

  field :vm_id,      type: String
  field :account_id, type: String
  field :user_id,    type: String
  field :vm,         type: Hash
  field :command,    type: String
  field :token,      type: String
  field :provider,   type: String
  field :messages,   type: Array, default: []
  field :status,     type: String
  field :ssh,        type: String
```

```
  private

        def check_unique
                req = Request.find_by(vm_id: self.vm_id)
                if (req && req["id"].to_s != self["id"].to_s)
                        self.status = "FAILURE"
                        self.messages << "Cannot make more than one request for each vm."
                        return false
                else
                        return true
                end
        end

        def exclude_id
                self.vm = self.vm.except("_id")
        end
end
```

user.rb

```
class User
  include Mongoid::Document
  include Mongoid::Timestamps
  include ActiveModel::SecurePassword

  before_save { self.email = email.downcase }
  before_create :create_remember_token

  validates :name,  presence: true, length: { maximum: 50 }
  VALID_EMAIL_REGEX = /\A[\w+\-.]+@[a-z\d\-.]+\.[a-z]+\z/i
  validates :email, presence: true, format: { with: VALID_EMAIL_REGEX },
                    uniqueness: { case_sensitive: false }

  embeds_many :accounts
  field :name, type: String
  field :email, type: String
  field :password_digest, type: String
  field :remember_token, type: String

  index({ email: 1, remember_token: 1 }, { unique: true })

  has_secure_password
  validates :password, length: { minimum: 6 }

  def User.new_remember_token
    SecureRandom.urlsafe_base64
  end

  def User.digest(token)
    Digest::SHA1.hexdigest(token.to_s)
  end

  private

    def create_remember_token
      self.remember_token = User.digest(User.new_remember_token)
    end
end
```

vm.rb

```
class Vm
  include Mongoid::Document
  include Mongoid::Timestamps

  embedded_in :account, inverse_of: :vms

  field :image,      type: String
  field :cpu,        type: String
  field :memory,     type: String
  field :disk,       type: String
```

```
  field :ips,        type: Array, default: []
  field :name,       type: String
  field :username,   type: String
  field :password,   type: String
  field :providerID, type: String
  field :hostname,   type: String
  field :status,     type: String
  field :messages,   type: Array, default: []
  field :parameters, type: Hash,  default: {}
  field :recipes,    type: Hash,  default: {}
  field :attribs,    type: Hash,  default: {}
end
```

Controllers

accounts_controller.rb

```
class AccountsController < ApplicationController
  before_action :signed_in_user
  before_action :correct_user,   only: [:destroy, :edit, :update, :show]

  def new
    @account = current_user.accounts.new
  end

  def create
    @account = current_user.accounts.new(account_params)
    if @account.save
      flash[:success] = "Account created!"
      redirect_to root_url
    else
      render 'static_pages/home'
    end
  end

  def show
    @account = current_user.accounts.find(params[:id])
    @account.vms.all.each do | vm |
      if vm.status == "DELETED"
        vm.destroy
      end
    end
    @vms = @account.vms.all
  end

  def update
    if @account.update_attributes(account_params)
      flash[:success] = "Account updated"
      redirect_to root_url
    else
      render 'edit'
    end
  end

  def edit
  end

  def destroy
    @account.destroy
    redirect_to root_url
  end

  private

    def account_params
      params.require(:account).permit(:username, :provider, :token)
    end
```

```
    def correct_user
      @account = current_user.accounts.find_by(id: params[:id])
      redirect_to root_url if @account.nil?
    end
end
```

requests_controller.rb

```
class RequestsController < ApplicationController
        skip_before_filter :verify_authenticity_token

  def show
    @request = Request.find(params[:id])
    render :json => @request
  end

  def update
    @request = Request.find(params[:id])
    @user = User.find(@request["user_id"])
    @account = @user.accounts.find(@request["account_id"])
    @vm = @account.vms.find(@request["vm_id"])

    if params["request"]["vm"]
      @request["vm"] = params["request"]["vm"]
      @vm["providerID"] = params["request"]["vm"]["providerID"]
      @vm["password"] = params["request"]["vm"]["password"]
      @vm["hostname"] = params["request"]["vm"]["hostname"]
      @vm["username"] = params["request"]["vm"]["username"]
    end
    if params["request"]["messages"]
      @request["messages"] = params["request"]["messages"]
      @vm["messages"] = params["request"]["messages"]
    end
    if params["request"]["status"]
      @request["status"] = params["request"]["status"]
      @vm["status"] = params["request"]["status"]
    end

    if @request.save && @vm.save
      puts "SUCCESS"
      render :json => { "result" => "success" }
    else
      puts "FAILURE"
      render :json => { "result" => "failure" }
    end
  end

  def destroy
    @request = Request.find(params[:id])
    if @request.destroy
      puts "SUCCESS"
      render :json => { "result" => "success" }
    else
      puts "FAILURE"
      render :json => { "result" => "failure" }
    end
  end
end
```

sessions_controller.rb

```
class SessionsController < ApplicationController

  def new
  end

  def create
    user = User.find_by(email: params[:session][:email].downcase)
    if user && user.authenticate(params[:session][:password])
      sign_in user
```

```
      redirect_back_or user
    else
      flash.now[:error] = 'Invalid email/password combination'
      render 'new'
    end
  end

  def destroy
    sign out
    redirect_to root_url
  end
end
```

static_pages_controller.rb

```
class StaticPagesController < ApplicationController
  def home
    if signed_in?
      redirect_to current_user
    end
  end
end
```

users_controller.rb

```
class UsersController < ApplicationController
  before_action :signed_in_user, only: [:edit, :update, :show]
  before_action :correct_user,   only: [:edit, :update, :show]

  def new
    @user = User.new
  end

  def create
    @user = User.new(user_params)
    if @user.save
      sign_in @user
      flash[:success] = "Welcome!"
      redirect_to @user
    else
      render 'new'
    end
  end

  def show
    @user = User.find(params[:id])
    @accounts = @user.accounts.all
  end

  def edit
  end

  def update
    if @user.update_attributes(user_params)
      flash[:success] = "Profile updated"
      redirect_to @user
    else
      render 'edit'
    end
  end

  private

    def user_params
      params.require(:user).permit(:name, :email, :password,
                                   :password_confirmation)
    end

    # Before filters
```

```
    def correct_user
      @user = User.find(params[:id])
      redirect_to(root_url) unless current_user?(@user)
    end
end
```

vms_controller.rb

```
class VmsController < ApplicationController
  include VmsHelper
  before action :signed in user
  before_action :correct_user,   only: [:destroy, :edit, :update, :show]

  def new
    @account = current_user.accounts.find(params[:account_id])
    @vm = @account.vms.new
    @image = eval "#{@account.provider.downcase}_resource(\"image\")"
    proc = eval "#{@account.provider.downcase}_resource(\"proc\")"
    @cpu = proc["cpu"]
    @memory = proc["memory"]
    @disk = proc["disk"]
    @ips = eval "#{@account.provider.downcase}_resource(\"ips\")"
    @cookbooks = Cookbook.all
  end

  def create
    @account = current_user.accounts.find(params[:account_id])
    @vm = @account.vms.new(vm_params)
    # @vm[:parameters] = params[:vm][:parameters]
    @vm[:recipes] = params[:vm][:recipes]
    # @vm[:attribs] = params[:vm][:attribs]
    if params[:vm][:ips]
      params[:vm][:ips].each { | k,v | @vm[:ips] << k }
    end
    @vm[:status] = "BUILDING"
    @request = Request.new( :vm_id => @vm[:id].to_s,
                            :account_id => @account[:id].to_s,
                            :user_id => current_user.id.to_s,
                            :vm => @vm.as_document,
                            :status => "BUILDING",
                            :command => "create",
                            :token => @account.token,
                            :provider => @account.provider)
    if @vm.save
      if @request.save
        send_request(@request[:id].to_s)
        flash[:success] = "VM stored - Creation will start soon!"
        redirect_to user_account_vm_path(current_user,@account,@vm)
        # redirect_to rquest_user_account_vm_path(current_user,@account,@vm,:cmd => "create")
      else
        @vm.destroy
        flash[:failure] = "Request not created! - Database issue"
        redirect_to user_account_path(current_user,@account)
      end
    else
      flash[:failure] = "VM not created! - Database issue"
      redirect_to user_account_path(current_user,@account)
    end
  end

  def show
       @account = current_user.accounts.find(params[:account_id])
       @vm = @account.vms.find(params[:id])
    @requests = Request.find_by(vm_id: params[:id]).to_a
  end

  def update
  end

  def edit
```

```
  end

  def status
    @account = current_user.accounts.find(params[:account_id])
    @vm = @account.vms.find(params[:id])
    @request = Request.new( :vm_id => @vm[:id].to_s,
                            :account_id => @account[:id].to_s,
                            :user_id => current_user.id.to_s,
                            :vm => @vm.as_document,
                            :command => "status",
                            :token => @account.token,
                            :provider => @account.provider)
    if @request.save
      send_request(@request[:id].to_s)
      redirect_to user_account_path(current_user,@account)
    else
      flash[:failure] = @request.messages.pop
      redirect_to user_account_path(current_user,@account)
    end
  end

  def destroy
    @account = current_user.accounts.find(params[:account_id])
    @vm = @account.vms.find(params[:id])
    @vm[:status] = "DELETING"
    @request = Request.new( :vm_id => @vm[:id].to_s,
                            :account_id => @account[:id].to_s,
                            :user_id => current_user.id.to_s,
                            :vm => @vm.as_document,
                            :status => "DELETING",
                            :command => "destroy",
                            :token => @account.token,
                            :provider => @account.provider)
    if @request.save
      if @vm.save
        send_request(@request[:id].to_s)
        flash[:success] = "VM marked for deletion - Deletion will start soon!"
        redirect_to user_account_vm_path(current_user,@account,@vm)
        # redirect_to rquest_user_account_vm_path(current_user,@account,@vm,:cmd => "create")
      else
        flash[:failure] = "VM not marked for deletion! - Database issue"
        redirect_to user_account_vm_path(current_user,@account,@vm)
      end
    else
      flash[:failure] = @request.messages.pop
      redirect_to user_account_vm_path(current_user,@account,@vm)
    end
  end

  def clear
    @account = current_user.accounts.find(params[:account_id])
    @vm = @account.vms.find(params[:id])
    #@vm.destroy
    @request = Request.find_by(vm_id: @vm[:id].to_s)
    @request.destroy
    redirect_to user_account_path(current_user,@account)
  end

  def start
    @account = current_user.accounts.find(params[:account_id])
    @vm = @account.vms.find(params[:id])
    @vm[:status] = "STARTING"
    @request = Request.new( :vm_id => @vm[:id].to_s,
                            :account_id => @account[:id].to_s,
                            :user_id => current_user.id.to_s,
                            :vm => @vm.as_document,
                            :status => "STARTING",
                            :command => "start",
                            :token => @account.token,
                            :provider => @account.provider)
    if @request.save
```

```
      if @vm.save
        send_request(@request[:id].to_s)
        flash[:success] = "VM starting!"
        redirect_to user_account_vm_path(current_user,@account,@vm)
        # redirect_to rquest_user_account_vm_path(current_user,@account,@vm,:cmd => "create")
      else
        flash[:failure] = "VM could not be started! - Database issue"
        redirect_to user_account_vm_path(current_user,@account,@vm)
      end
    else
      flash[:failure] = @request.messages.pop
      redirect_to user_account_vm_path(current_user,@account,@vm)
    end
  end

  def shutdown
    @account = current_user.accounts.find(params[:account_id])
    @vm = @account.vms.find(params[:id])
    @vm[:status] = "STOPPING"
    @request = Request.new( :vm_id => @vm[:id].to_s,
                            :account_id => @account[:id].to_s,
                            :user_id => current_user.id.to_s,
                            :vm => @vm.as_document,
                            :status => "STOPPING",
                            :command => "shutdown",
                            :token => @account.token,
                            :provider => @account.provider)
    if @request.save
      if @vm.save
        send_request(@request[:id].to_s)
        flash[:success] = "VM stopping!"
        redirect_to user_account_vm_path(current_user,@account,@vm)
        # redirect_to rquest_user_account_vm_path(current_user,@account,@vm,:cmd => "create")
      else
        flash[:failure] = "VM could not be stopped! - Database issue"
        redirect_to user_account_vm_path(current_user,@account,@vm)
      end
    else
      flash[:failure] = @request.messages.pop
      redirect_to user_account_vm_path(current_user,@account,@vm)
    end
  end

  def terminal
    @account = current_user.accounts.find(params[:account_id])
    @vm = @account.vms.find(params[:id])
  end

  def ssh
      @account = current_user.accounts.find(params[:account_id])
      @vm = @account.vms.find(params[:id])
      resp = ssh_cmd(@vm["hostname"],@vm["username"],@vm["password"],params["command"])
      render :json => { "response" => resp }
  end

  private

    def vm_params
      params.require(:vm).permit(:image, :cpu, :memory, :disk, :ips, :name)
    end

    def correct_user
      @account = current_user.accounts.find_by(id: params[:account_id])
      redirect_to root_url if @account.nil?
      @vm = @account.vms.find_by(id: params[:id])
      redirect_to root_url if @vm.nil?
    end
end
```

Helpers

requests_helper.rb

```
module RequestsHelper

        def send_request(reqId)
        conn = Bunny.new
                conn.start

                ch = conn.create_channel
    q  = ch.queue("from_server")

                puts " [x] VM command: #{action[:cmd]}"

    q.publish(reqId, :persistent => true)

                conn.close
  end

  def okeanos_receive
        conn = Bunny.new
                conn.start

                ch = conn.create_channel
    q = ch.queue("to_server", :durable => true)

        resp = nil
                vm_json_ext = {}
    q.subscribe(:manual_ack => true, :block => true) do |delivery_info, properties, payload|
        puts " [.] Got #{properties[:correlation_id]}"
        resp =
                { :corid => properties[:correlation_id],
                        :payload => payload }
        ch.ack(delivery_info.delivery_tag)
      delivery_info.consumer.cancel
    end

                conn.close

                return resp
        end


end
```

sessions_helper.rb

```
module SessionsHelper

  def sign_in(user)
    remember_token = User.new_remember_token
    cookies.permanent[:remember_token] = remember_token
    user.update_attribute(:remember_token, User.digest(remember_token))
    self.current_user = user
  end

  def signed_in?
    !current_user.nil?
  end

  def current_user=(user)
    @current_user = user
  end

  def current_user
    remember_token = User.digest(cookies[:remember_token])
```

```
    @current_user ||= User.find_by(remember_token: remember_token)
  end

  def current_user?(user)
    user == current_user
  end

  def signed_in_user
    unless signed_in?
      store_location
      redirect_to signin_url, notice: "Please sign in."
    end
  end

  def sign_out
    current_user.update_attribute(:remember_token,
                                  User.digest(User.new_remember_token))
    cookies.delete(:remember_token)
    self.current_user = nil
  end

  def redirect_back_or(default)
    redirect_to(session[:return_to] || default)
    session.delete(:return_to)
  end

  def store_location
    session[:return_to] = request.url if request.get?
  end
end
```

vms_helper.rb

```
module VmsHelper

  def send_request(reqId)
    conn = Bunny.new
    conn.start

    ch = conn.create_channel
    q  = ch.queue("from_server")

    puts " [x] Sent request #{reqId}"

    q.publish(reqId, :persistent => true)

    conn.close
  end

  def ssh_cmd(host,user,password,cmd)
    dt = ""
    Net::SSH.start(host, user, { :password => password, :paranoid => false }) do |ssh|
      ssh.open_channel do |ch|
        ch.request_pty do |c, success|
          raise "could not request pty" unless success
          ch.exec cmd
          ch.on_data do |c, data|
            if data =~ /\[sudo\]/ || data =~ /Password/i
              ch.send_data "#{password}\n"
            else
              #$stdout.print data
              dt = dt + data.to_s
            end
          end
          ch.on_extended_data do |c, type, data|
            #stderr.print data
            dt = dt + data.to_s
          end
        end
```

```
      end
    end
    return dt
  end

  def okeanos_receive
        conn = Bunny.new
                conn.start

                ch = conn.create_channel
    q = ch.queue("to_server", :durable => true)
        resp = nil
                vm_json_ext = {}
    q.subscribe(:manual_ack => true, :block => true) do |delivery_info, properties, payload|
        puts " [.] Got #{properties[:correlation_id]}"
        resp = { :corid => properties[:correlation_id],
                            :payload => payload }
        ch.ack(delivery_info.delivery_tag)
      delivery_info.consumer.cancel
    end
                conn.close
                return resp
        end

  def okeanos_resource(type)
    case type
    when "image"
      resp = HTTParty.get("https://cyclades.okeanos.grnet.gr/compute/v2.0/images",
                          { :headers => { "X-Auth-Token" => @account.token } })
      im = JSON.parse(resp.body)
      images = []
      im["images"].each { | i | images << [i["name"],i["id"]] }
      return images
    when "proc"
      resp = HTTParty.get("https://cyclades.okeanos.grnet.gr/compute/v2.0/flavors/detail",
                          { :headers => { "X-Auth-Token" => @account.token } })
      fl = JSON.parse(resp.body)
      cpu = Set.new
      memory = Set.new
      disk = Set.new
      fl["flavors"].each do | f |
        # if (f["SNF:allow_create"] == true)
          f["name"] =~ /^C(\d+)R(\d+)D(\d+)(\D+)$/
          cpu.add($1.to_i)
          memory.add($2.to_i)
          disk.add($3.to_i)
        # end
      end
      return { "cpu"    => cpu.to_a,
               "memory" => memory.to_a,
               "disk"   => disk.to_a }
    when "ips"
      resp = HTTParty.get("https://cyclades.okeanos.grnet.gr/network/v2.0/floatingips",
                          { :headers => { "X-Auth-Token" => @account.token } })
      ip = JSON.parse(resp.body)
      ips = []
      ip["floatingips"].each do | i |
        if (i["instance_id"] == nil) && (i["port_id"] == nil)
           ips << { "uuid" => i["floating_network_id"],
                    "ip" => i["floating_ip_address"] }
        end
      end
      return ips
    else
      puts "WRONG OPTION"
    end
  end

end
```

Λοιπά αρχεία ρυθμίσεων

routes.eb

```
Kayden::Application.routes.draw do
  resources :users do
    resources :accounts do
      resources :vms do
        get  'log',      on: :member
        get  'rquest',   on: :member
        get  'status',   on: :member
        get  'clear',    on: :member
        get  'start',    on: :member
        get  'shutdown', on: :member
        get  'terminal', on: :member
        post 'ssh',     on: :member
      end
    end
  end
  resources :sessions, only: [:new, :create, :destroy]
  resources :requests
  root  'static_pages#home'
  match '/signup',  to: 'users#new',            via: 'get'
  match '/signin',  to: 'sessions#new',         via: 'get'
  match '/signout', to: 'sessions#destroy',     via: 'delete'
  # The priority is based upon order of creation: first created -> highest priority.
  # See how all your routes lay out with "rake routes".

  # You can have the root of your site routed with "root"
  # root 'welcome#index'

  # Example of regular route:
  #   get 'products/:id' => 'catalog#view'

  # Example of named route that can be invoked with purchase url(id: product.id)
  #   get 'products/:id/purchase' => 'catalog#purchase', as: :purchase

  # Example resource route (maps HTTP verbs to controller actions automatically):
  #   resources :products

  # Example resource route with options:
  #   resources :products do
  #     member do
  #       get 'short'
  #       post 'toggle'
  #     end
  #
  #     collection do
  #       get 'sold'
  #     end
  #   end

  # Example resource route with sub-resources:
  #   resources :products do
  #     resources :comments, :sales
  #     resource :seller
  #   end

  # Example resource route with more complex sub-resources:
  #   resources :products do
  #     resources :comments
  #     resources :sales do
  #       get 'recent', on: :collection
  #     end
  #   end

  # Example resource route with concerns:
  #   concern :toggleable do
  #     post 'toggle'
  #   end
```

```
  #   resources :posts, concerns: :toggleable
  #   resources :photos, concerns: :toggleable

  # Example resource route within a namespace:
  #   namespace :admin do
  #     # Directs /admin/products/* to Admin::ProductsController
  #     # (app/controllers/admin/products_controller.rb)
  #     resources :products
  #   end
end
```

Gemfile

```
source 'https://rubygems.org'

# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '4.0.4'


# Use SCSS for stylesheets
gem 'sass-rails', '~> 4.0.2'

# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'

# Use CoffeeScript for .js.coffee assets and views
gem 'coffee-rails', '~> 4.0.0'

# See https://github.com/sstephenson/execjs#readme for more supported runtimes
# gem 'therubyracer', platforms: :ruby

# Use jquery as the JavaScript library
gem 'jquery-rails'

# Turbolinks makes following links in your web application faster. Read more:
https://github.com/rails/turbolinks
gem 'turbolinks'

# Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
gem 'jbuilder', '~> 1.2'

group :doc do
  # bundle exec rake doc:rails generates the API under doc/api.
  gem 'sdoc', require: false
end

# Use ActiveModel has_secure_password
# gem 'bcrypt', '~> 3.1.7'

# Use unicorn as the app server
# gem 'unicorn'

# Use Capistrano for deployment
# gem 'capistrano', group: :development

# Use debugger
# gem 'debugger', group: [:development, :test]

# MongoDB database
gem 'mongoid', '~> 4.0.0.beta1', github: 'mongoid/mongoid'
gem 'bson_ext'

# bcrypt function (used to hash user passwords)
gem 'bcrypt'

# jquery ui
gem 'jquery-ui-rails'

# rabbitmq implementation
```

```
gem 'bunny'

gem 'json'

gem 'httparty'

gem 'rename'

gem 'net-ssh'
```

Παράρτημα Γ - Recipes

apache2

default.rb

```
#
# Cookbook Name:: apache2
# Recipe:: default
#
# Copyright 2008-2013, Opscode, Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

package 'apache2' do
  package_name node['apache']['package']
end

if platform_family?('rhel', 'fedora', 'arch', 'suse', 'freebsd')
  directory node['apache']['log_dir'] do
    mode '0755'
  end

  package node['apache']['perl_pkg']

  cookbook_file '/usr/local/bin/apache2_module_conf_generate.pl' do
    source 'apache2_module_conf_generate.pl'
    mode   '0755'
    owner  'root'
    group  node['apache']['root_group']
  end

  %w[sites-available sites-enabled mods-available mods-enabled].each do |dir|
    directory "#{node['apache']['dir']}/#{dir}" do
      mode  '0755'
      owner 'root'
      group node['apache']['root_group']
    end
  end

  execute 'generate-module-list' do
    command "/usr/local/bin/apache2_module_conf_generate.pl #{node['apache']['lib_dir']}
#{node['apache']['dir']}/mods-available"
    action  :nothing
  end

  %w[a2ensite a2dissite a2enmod a2dismod].each do |modscript|
    template "/usr/sbin/#{modscript}" do
      source "#{modscript}.erb"
      mode  '0700'
      owner 'root'
      group node['apache']['root_group']
    end
  end

  # installed by default on centos/rhel, remove in favour of mods-enabled
```

```
  %w[proxy_ajp auth_pam authz_ldap webalizer ssl welcome].each do |f|
    file "#{node['apache']['dir']}/conf.d/#{f}.conf" do
      action :delete
      backup false
    end
  end

  # installed by default on centos/rhel, remove in favour of mods-enabled
  file "#{node['apache']['dir']}/conf.d/README" do
    action :delete
    backup false
  end

  # enable mod_deflate for consistency across distributions
  include_recipe 'apache2::mod_deflate'
end

if platform_family?('freebsd')
  file "#{node['apache']['dir']}/Includes/no-accf.conf" do
    action :delete
    backup false
  end

  directory "#{node['apache']['dir']}/Includes" do
    action :delete
  end

  %w[
      httpd-autoindex.conf httpd-dav.conf httpd-default.conf httpd-info.conf
      httpd-languages.conf httpd-manual.conf httpd-mpm.conf
      httpd-multilang-errordoc.conf httpd-ssl.conf httpd-userdir.conf
      httpd-vhosts.conf
  ].each do |f|
    file "#{node['apache']['dir']}/extra/#{f}" do
      action :delete
      backup false
    end
  end

  directory "#{node['apache']['dir']}/extra" do
    action :delete
  end
end

%W[
  #{node['apache']['dir']}/ssl
  #{node['apache']['dir']}/conf.d
  #{node['apache']['cache_dir']}
].each do |path|
  directory path do
    mode  '0755'
    owner 'root'
    group node['apache']['root_group']
  end
end

# Set the preferred execution binary - prefork or worker
template '/etc/sysconfig/httpd' do
  source   'etc-sysconfig-httpd.erb'
  owner    'root'
  group    node['apache']['root_group']
  mode     '0644'
  notifies :restart, 'service[apache2]'
  only_if  { platform_family?('rhel', 'fedora') }
end

template 'apache2.conf' do
  case node['platform_family']
  when 'rhel', 'fedora', 'arch'
    path "#{node['apache']['dir']}/conf/httpd.conf"
  when 'debian'
```

```
    path "#{node['apache']['dir']}/apache2.conf"
  when 'freebsd'
    path "#{node['apache']['dir']}/httpd.conf"
  end
  source   'apache2.conf.erb'
  owner    'root'
  group    node['apache']['root_group']
  mode     '0644'
  notifies :reload, 'service[apache2]'
end

template 'apache2-conf-security' do
  path     "#{node['apache']['dir']}/conf.d/security.conf"
  source   'security.erb'
  owner    'root'
  group    node['apache']['root_group']
  mode     '0644'
  backup   false
  notifies :reload, 'service[apache2]'
end

template 'apache2-conf-charset' do
  path      "#{node['apache']['dir']}/conf.d/charset.conf"
  source   'charset.erb'
  owner    'root'
  group    node['apache']['root_group']
  mode     '0644'
  backup   false
  notifies :reload, 'service[apache2]'
end

template "#{node['apache']['dir']}/ports.conf" do
  source   'ports.conf.erb'
  owner    'root'
  group    node['apache']['root_group']
  mode     '0644'
  notifies :reload, 'service[apache2]'
end

template "#{node['apache']['dir']}/sites-available/default" do
  source   'default-site.erb'
  owner    'root'
  group    node['apache']['root_group']
  mode     '0644'
  notifies :reload, 'service[apache2]'
end

node['apache']['default_modules'].each do |mod|
  module_recipe_name = mod =~ /^mod_/ ? mod : "mod_#{mod}"
  include_recipe "apache2::#{module_recipe_name}"
end

apache_site 'default' do
  enable node['apache']['default_site_enabled']
end

service 'apache2' do
  case node['platform_family']
  when 'rhel', 'fedora', 'suse'
    service_name 'httpd'
    # If restarted/reloaded too quickly httpd has a habit of failing.
    # This may happen with multiple recipes notifying apache to restart - like
    # during the initial bootstrap.
    restart_command '/sbin/service httpd restart && sleep 1'
    reload_command '/sbin/service httpd reload && sleep 1'
  when 'debian'
    service name 'apache2'
    restart_command '/usr/sbin/invoke-rc.d apache2 restart && sleep 1'
    reload command '/usr/sbin/invoke-rc.d apache2 reload && sleep 1'
  when 'arch'
    service_name 'httpd'
```

```
  when 'freebsd'
    service_name 'apache22'
  end
  supports [:restart, :reload, :status]
  action [:enable, :start]
end
```

mod_php5.rb

```
#
# Cookbook Name:: apache2
# Recipe:: php5
#
# Copyright 2008-2013, Opscode, Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

case node['platform_family']
when 'debian'
  package 'libapache2-mod-php5'
when 'arch'
  package 'php-apache' do
    notifies :run, 'execute[generate-module-list]', :immediately
  end
when 'rhel'
  package 'which'

  package 'php package' do
    if node['platform_version'].to_f < 6.0
      package_name 'php53'
    else
      package_name 'php'
    end
    notifies :run, 'execute[generate-module-list]', :immediately
    not_if 'which php'
  end
when 'fedora'
  package 'php package' do
    package_name 'php'
    notifies :run, 'execute[generate-module-list]', :immediately
    not_if 'which php'
  end
when 'freebsd'
  freebsd_port_options 'php5' do
    options 'APACHE' => true
    action :create
  end

  package 'php package' do
    package name 'php5'
    source 'ports'
    notifies :run, 'execute[generate-module-list]', :immediately
  end
end

file "#{node['apache']['dir']}/conf.d/php.conf" do
  action :delete
  backup false
```

```
end

apache_module 'php5' do
  case node['platform_family']
  when 'rhel', 'fedora', 'freebsd'
    conf true
    filename 'libphp5.so'
  end
end
```

mysql

client.rb

```
#
# Cookbook Name:: mysql
# Recipe:: client
#
# Copyright 2008-2013, Chef Software, Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

mysql_client 'default' do
  action :create
end

end
```

server.rb

```
#
# Cookbook Name:: mysql
# Recipe:: server
#
# Copyright 2008-2013, Chef Software, Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

mysql_service node['mysql']['service_name'] do
```

```
  version node['mysql']['version']
  port node['mysql']['port']
  data_dir node['mysql']['data_dir']
  server_root_password node['mysql']['server_root_password']
  server_debian_password node['mysql']['server_debian_password']
  server_repl_password node['mysql']['server_repl_password']
  allow_remote_root node['mysql']['allow_remote_root']
  remove_anonymous_users node['mysql']['remove_anonymous_users']
  remove_test_database node['mysql']['remove_test_database']
  root_network_acl node['mysql']['root_network_acl']
  version node['mysql']['version']
  action :create
end
```

php

default.rb

```
#
# Author::  Joshua Timberman (<joshua@opscode.com>)
# Author::  Seth Chisamore (<schisamo@opscode.com>)
# Cookbook Name:: php
# Recipe:: default
#
# Copyright 2009-2011, Opscode, Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

include_recipe "php::#{node['php']['install_method']}"

# update the main channels
php_pear_channel 'pear.php.net' do
  action :update
end

php_pear_channel 'pecl.php.net' do
  action :update
end

include_recipe "php::ini"
```

module_apc.rb

```
#
# Author::  Joshua Timberman (<joshua@opscode.com>)
# Author::  Seth Chisamore (<schisamo@opscode.com>)
# Cookbook Name:: php
# Recipe:: module_apc
#
# Copyright 2009-2011, Opscode, Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
```

```
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

case node['platform_family']
when 'rhel', 'fedora'
  %w{ httpd-devel pcre pcre-devel }.each do |pkg|
    package pkg do
      action :install
    end
  end
  php_pear 'APC' do
    action :install
    directives(:shm_size => '128M', :enable_cli => 0)
  end
when 'debian'
  package 'php-apc' do
    action :install
  end
end
```