ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Σχεδίαση και ανάπτυξη ευφυούς πλατφόρμας διαχείρισης συνομοσπονδιών νέφους

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

των

Ιωάννη Κ. Στουρνάρα

Ευάγγελου Ι. Οικονόμου

Επιβλέπων: Ιάκωβος Βενιέρης

Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2017

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Σχεδίαση και ανάπτυξη ευφυούς πλατφόρμας διαχείρισης συνομοσπονδιών νέφους

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

των

Ιωάννη Κ. Στουρνάρα

Ευάγγελου Ι. Οικονόμου

Επιβλέπων: Ιάκωβος Βενιέρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την Δευτέρα, 23 Οκτωβρίου 2017

……..	……..	……...
Ιάκωβος Βενιέρης Καθηγητής Ε.Μ.Π.	Δήμητρα Θεοδώρα Κακλαμάνη Καθηγήτρια Ε.Μ.Π.	Γεώργιος Ματσόπουλος Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2017

……..
Ιωάννης Κ. Στουρνάρας
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

……..
Ευάγγελος Ι. Οικονόμου
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Στις μέρες μας, η χρήση των υπολογιστικών νεφών γνωρίζει ιδιαίτερη «άνθηση» και τείνει να γίνει ένα από τα βασικότερα μοντέλα παροχής υλικού και λογισμικού ως υπηρεσία. Η ανάπτυξη που γνώρισε τα τελευταία χρόνια οδήγησε στην ανάγκη δημιουργίας συνομοσπονδιών νέφους μέσω των οποίων οι χρήστες θα μπορούν να διαμοιράζονται πόρους και υπηρεσίες.

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι η σχεδίαση και ανάπτυξη μίας πλατφόρμας η οποία θα υλοποιεί την ιδέα της συνομοσπονδίας νεφών. Στόχος της πλατφόρμας είναι να δίνει τη δυνατότητα στους παρόχους υπηρεσιών νέφους να συνδέσουν την τοπική υποδομή τους με την παγκόσμια αγορά χωρίς την ανάγκη της απευθείας επικοινωνίας μεταξύ των διαφορετικών παρόχων. Επιπλέον, θα δίνεται η δυνατότητα στους χρήστες της πλατφόρμας να αναζητήσουν τον βέλτιστο πάροχο που καλύπτει τις εκάστοτε ανάγκες τους για υπολογιστικούς πόρους προσφέροντας, με αυτόν τον τρόπο, πλουραλισμό επιλογών.

Μία ακόμα πτυχή που θα μελετηθεί είναι ο σχεδιασμός ενός συνολικού συστήματος διαχείρισης των δεδομένων, στο οποίο θα χρησιμοποιηθεί η τεχνολογία των μη σχεσιακών βάσεων δεδομένων που συνδυάζουν χαρακτηριστικά απαραίτητα για την λειτουργία της πλατφόρμας. Ταυτόχρονα, μελετάται η ανάγκη προτυποποίησης των πόρων με βάση συγκεκριμένα χαρακτηριστικά ώστε να μπορεί η πλατφόρμα μας να διαχειριστεί αιτήματα από διαφορετικούς παρόχους χωρίς, όμως, βλάβη της γενικότητας. Επιπροσθέτως, θα μελετηθεί η απόδοση της πλατφόρμας σε μεγάλο όγκο δεδομένων και χρηστών.

Τέλος, θα σχεδιαστεί και θα αναπτυχθεί σύστημα για την δίκαιη αντιμετώπιση των χρηστών που συμμετέχουν σε ένα εθελοντικό περιβάλλον, όπως αυτό της συνομοσπονδίας νεφών. Κρίνεται απαραίτητο για την καλή λειτουργία της πλατφόρμας μας σε πραγματικές συνθήκες να αποτρέπονται συμπεριφορές κατάχρησης του συστήματος από κακόβουλους χρήστες, δημιουργώντας ένα περιβάλλον ασφαλές για την αλληλοσυνεργασία των χρηστών.

Λέξεις κλειδιά: Flask, REST, πλατφόρμα, συνομοσπονδία νεφών, NoSQL βάση δεδομένων, βαθμολογικό σύστημα Glicko, πρωτόκολλο HTTP

Abstract

Nowadays, the use of cloud computing is booming remarkably and tends to become one of the most basic models of provision of infrastructure and software as a service. In recent years, cloud computing has been significantly developed, leading to the need of designing and developing Cloud Federation systems that users will use to share resources and services.

The purpose of the thesis is the design and development of a platform which will implement the concept of Cloud Federation. The platform will provide the ability to cloud computing providers to link their local infrastructure to the world market without the need of establishing direct communication with each other. Furthermore, platform users will be able to search for the optimal provider that can satisfy their needs of computing resources, thereby providing a plurality of choices.

Another aspect that will be studied is the design of a comprehensive data management system, which will use non-relational database technology that combines features essential to the platform's operation. At the same time, we study the imperative need of standardizing resources based on specific features so that our platform will be able to handle requests from different cloud providers in a universal standardized way. In addition, we will measure the performance of our system on a large volume of data and users.

Finally, we will design and develop a system to face the need of treating fairly the users that participate in a voluntary environment, such as the Cloud Federation. It is necessary for the platform's proper operation to prevent abuse of the system by malicious users, creating a secure environment for user interaction.

Keywords: Flask, REST, platform, Cloud Federation, NoSQL database, Glicko rating system, HTTP protocol

Ευχαριστίες

Για την εκπόνηση της παρούσας διπλωματικής εργασίας και την ανάπτυξη της πλατφόρμας αλλά και γενικότερα για την ολοκλήρωση του κύκλου σπουδών μας, νιώθουμε την ανάγκη να ευχαριστήσουμε όλους τους ανθρώπους που μας βοήθησαν.

Αρχικά, θα θέλαμε να ευχαριστήσουμε θερμά τον επιβλέποντα της διπλωματικής εργασίας μας, καθηγητή κ. Ιάκωβο Βενιέρη, ο οποίος μας έδωσε τη δυνατότητα να ασχοληθούμε με ένα αντικείμενο που πραγματικά επιθυμούσαμε και οι δύο. Τον ευχαριστούμε ιδιαίτερα για την πολύτιμη βοήθεια του και τις πολύτιμες γνώσεις και συμβουλές που μας παρείχε κατά την εκπόνηση της εργασίας και καθ΄ όλη τη διάρκεια των σπουδών μας, αλλά κυρίως για την προσωπική ικανοποίηση που εισπράξαμε με το πέρας της παρούσας διπλωματικής εργασίας και την αίσθηση του ότι συμπεριλαμβανόμαστε πλέον κι εμείς στους εν δυνάμει προγραμματιστές.

Στη συνέχεια, θα θέλαμε να ευχαριστήσουμε ξεχωριστά τα μέλη της επιτροπής, την καθηγήτρια κα Δήμητρα Θεοδώρα Κακλαμάνη για την σημαντική συμβολή της στην περαίωση της εργασίας καθώς και τον καθηγητή κ. Γεώργιο Ματσόπουλο για την υποστήριξη του καθ΄ όλη τη διάρκεια εκπόνησης της.

Ιδιαίτερες ευχαριστίες θα θέλαμε να απευθύνουμε στον υποψήφιο Διδάκτορα κ. Ανδρέα Καψάλη, χωρίς τη βοήθεια του οποίου θα ήταν αδύνατη η ολοκλήρωση της διπλωματικής εργασίας, μιας και παρείχε υποστήριξη, καθοδήγηση και συμβουλές σε κάθε δυσκολία που αντιμετωπίσαμε. Επίσης, θα θέλαμε να ευχαριστήσουμε τον υποψήφιο Διδάκτορα κ. Πέτρο Φλώριο Μπάκαλο, αλλά και όλο το προσωπικό του Εργαστηρίου Ευφυών Επικοινωνιών και Δικτύων Ευρείας Ζώνης που μας προσέφεραν την βοήθεια τους, όποτε αυτή ζητήθηκε.

Τέλος, ο καθένας μας προσωπικά θα θέλαμε να ευχαριστήσουμε τις οικογένειες μας.

Εγώ, ο Ιωάννης Στουρνάρας, θα ήθελα να ευχαριστήσω τους γονείς μου και τον αδερφό μου για την υποστήριξη που μου παρείχαν καθ’ όλη τη διάρκεια των σπουδών μου.

Εγώ, ο Ευάγγελος Οικονομού, θα ήθελα να ευχαριστήσω τους γονείς μου και τις αδερφές μου, που με στήριξαν σε όλη τη διάρκεια των σπουδών μου.

Πίνακας Περιεχομένων

1

Εισαγωγή

1.1 Η τεχνολογία των υπολογιστικών νεφών και τα βασικά χαρακτηριστικά τους

Στις μέρες μας, η μεγάλη διαθεσιμότητα σε δίκτυα μεγάλης χωρητικότητας, σε χαμηλού κόστους υπολογιστές και συσκευές αποθήκευσης, σε συνδυασμό με τη μείωση των τιμών σε μεγάλη κλίμακα της ηλεκτρικής ενέργειας, του εύρους ζώνης δικτύου και του λογισμικού οδήγησε στην ανάπτυξη της τεχνολογίας των υπολογιστικών νεφών, γνωστά και με τον όρο cloud computing. Το cloud computing είναι ένα μοντέλο υποδομών και λογισμικού που επιτρέπει την καθολική πρόσβαση σε κοινόχρηστες ομάδες διαρθρώσιμων πόρων, οι οποίοι προμηθεύονται κυρίως μέσω διαδικτύου. Το cloud computing επιτρέπει σε χρήστες και επιχειρήσεις με διάφορες υπολογιστικές ικανότητες να αποθηκεύουν και να επεξεργάζονται δεδομένα είτε σε ιδιωτικά νέφη είτε σε εξυπηρετητές που βρίσκονται σε κέντρα δεδομένων. Άμεσο αποτέλεσμα είναι οι μηχανισμοί πρόσβασης στα δεδομένα να είναι αποδοτικότεροι και πιο αξιόπιστοι. Όπως και ένας οργανισμός κοινής ωφέλειας, το cloud computing βασίζεται στην από κοινού διάθεση των πόρων για να επιτύχει συνοχή και οικονομίες κλίμακας.

Τα τελευταία χρόνια, το cloud computing αποτελεί μία υπηρεσία υψηλής ζήτησης. Ένας βασικός παράγοντας είναι ότι επιτρέπει στις εταιρίες να ελαχιστοποιούν ή να εξαλείφουν το αρχικό κόστος επένδυσης για την ανάπτυξη υποδομών. Επιτρέπει με αυτό τον τρόπο στις επιχειρήσεις να επενδύσουν στους κύριους τομείς της δραστηριότητας τους αντί να δαπανούν σε υπολογιστική υποδομή και συντήρηση της. Επίσης, λόγω της ικανότητας ελαστικής κλιμάκωσης, δίνει τη δυνατότητα στους χρήστες να αυξομειώνουν τους υπολογιστικούς τους πόρους ανάλογα με τις ανάγκες τους ώστε να μπορούν να ανταπεξέλθουν σε κυμαινόμενες και απρόβλεπτες επιχειρησιακές ανάγκες.

Το cloud computing έχει να επιδείξει μία πληθώρα χαρακτηριστικών που το κάνουν να αναπτύσσεται ραγδαία. Επιτρέπει στους χρήστες να έχουν πρόσβαση στα συστήματα χρησιμοποιώντας έναν διαδικτυακό φυλλομετρητή ανεξάρτητα από την τοποθεσία ή τη συσκευή που χρησιμοποιούν. Οι υποδομές που παρέχονται τυπικά από τρίτους, είναι προσβάσιμες μέσω διαδικτύου και οι χρήστες μπορούν να συνδεθούν σε αυτές από παντού.

Ένα ακόμα βασικό χαρακτηριστικό είναι ότι πολλοί χρήστες μπορούν να δουλεύουν παράλληλα στα ίδια δεδομένα αυξάνοντας έτσι την παραγωγικότητα. Η παραγωγικότητα, όμως, αυξάνεται και λόγω του γεγονότος ότι χρονοβόρες διαδικασίες, όπως η εγκατάσταση υλικού και λογισμικού, δεν είναι πλέον απαραίτητες αφού αυτά προσφέρονται από τους παρόχους cloud. Δίνεται έτσι η δυνατότητα στις ομάδες πληροφορικής να ασχοληθούν με βασικότερες για την επιχείρηση δραστηριότητες.

Τα πιο κύρια χαρακτηριστικά, όμως, είναι η απόδοση και η αξιοπιστία που παρέχεται μέσω της τεχνολογίας του cloud computing. Οι υπηρεσίες που παρέχονται, εκτελούνται σε ένα παγκόσμιο δίκτυο από ασφαλή κέντρα δεδομένων, τα οποία αναβαθμίζονται συνεχώς με την τελευταία γενιά του πιο γρήγορου και αποδοτικού υλικού (hardware). Προσφέρει, επίσης, μειωμένο χρόνο μεταφοράς δεδομένων δικτύου για τις εφαρμογές σε σχέση με ένα παραδοσιακό κέντρο δεδομένων, βελτιώνοντας την απόδοση.

Πρόσθετα, κάθε επιχείρηση έχει ανάγκη από αξιόπιστα συστήματα που της επιτρέπουν να λειτουργεί και να ανταπεξέρχεται σε περιπτώσεις καταστροφών. Η δυνατότητα των παρόχων υπηρεσιών cloud computing να δημιουργούν πολλαπλά αντίγραφα ασφαλείας σε πλεονάζοντα σημεία του δικτύου τους, επιτρέπει στις επιχειρήσεις να ανακάμπτουν ευκολότερα μετά από τυχόν καταστροφή επιτυγχάνοντας απρόσκοπτη συνέχεια της επαγγελματικής τους δραστηριότητας.

1.2 Μοντέλα παροχής υπηρεσιών των υπολογιστικών νεφών

Στη σύγχρονη εποχή επικρατεί το μοντέλο των προσανατολισμένων στις υπηρεσίες αρχιτεκτονικών που τάσσεται υπέρ της ιδέας τα πάντα να παρέχονται ως υπηρεσίες, γνωστό και ως “everything as a service”, ή πιο απλά “aaS”. Οι πάροχοι υπηρεσιών υπολογιστικών νεφών (cloud computing) προσφέρουν τις υπηρεσίες τους σε διάφορα μοντέλα. Σύμφωνα με το Εθνικό Ινστιτούτο Τυποποίησης και Τεχνολογίας (NIST) των ΗΠΑ τα τρία τυποποιημένα μοντέλα είναι το Υλικό-ως-Υπηρεσία (Infrastructure-as-a-Service, IaaS), η Πλατφόρμα-ως-Υπηρεσία (Platform-as-a-Service, Paas) και το Λογισμικό-ως-Υπηρεσία (Software-as-a-Service, SaaS). Τα μοντέλα αυτά προσφέρουν υψηλό βαθμό αφαιρετικότητας και γι’ αυτό μπορούν να απεικονιστούν ως διαφορετικά στρώματα μίας στοίβας: υποδομή-, πλατφόρμα-, λογισμικό-ως-υπηρεσία.

- Υλικό-ως-Υπηρεσία: Προσφέρεται η ικανότητα στον καταναλωτή να προμηθεύεται εξυπηρετητές, εικονικές μηχανές, χώρο αποθήκευσης, υπηρεσίες δικτύου και άλλους θεμελιώδεις υπολογιστικούς πόρους, όπου μπορεί να αναπτύξει και να εκτελέσει αυθαίρετα λογισμικό, που μπορεί να περιέχει λειτουργικά συστήματα και εφαρμογές. Ο καταναλωτής δεν μπορεί να διαχειριστεί ή να ελέγξει την υφιστάμενη υποδομή του νέφους, αλλά έχει τη δυνατότητα να ελέγχει τα λειτουργικά συστήματα, τον χώρο αποθήκευσης και τις εγκατεστημένες εφαρμογές, ενώ του παρέχεται και περιορισμένος έλεγχος επιλογής στοιχείων δικτύου. Θεωρείται η πιο βασική κατηγορία υπηρεσιών των υπολογιστικών νεφών και ο καταναλωτής πληρώνει ανάλογα με το χρόνο και την υπηρεσία που χρησιμοποιεί.
- Πλατφόρμα-ως-Υπηρεσία: Δίνεται η δυνατότητα στον καταναλωτή να αναπτύξει ή να αποκτήσει εφαρμογές χρησιμοποιώντας γλώσσες προγραμματισμού, βιβλιοθήκες και εργαλεία τα οποία υποστηρίζονται από τον πάροχο. Η υπηρεσία αυτή είναι σχεδιασμένη ώστε να μπορεί να υποστηρίζει την πλήρη διαδικασία ανάπτυξης μίας διαδικτυακής εφαρμογής, δηλαδή το χτίσιμο, τη δοκιμή, την ανάπτυξη, τη διαχείρισή και την ενημέρωση της. Επιτρέπει, με αυτό τον τρόπο, την αποφυγή δαπανών για την αγορά αδειών λογισμικού, της υποκείμενης υποδομής της εφαρμογής και άλλων αναπτυξιακών εργαλείων. Ο καταναλωτής δεν έχει τη δυνατότητα διαχείρισης της υποκείμενης υποδομής του νέφους, αλλά μπορεί να διαχειριστεί τις εφαρμογές και τις υπηρεσίες που αναπτύσσει καθώς και τη ρύθμιση των παραμέτρων τόσο των εφαρμογών όσο και του περιβάλλοντος υποδοχής.

- Λογισμικό-ως-υπηρεσία: Η υπηρεσία αυτή προσφέρει μία πλήρη λύση αγοράς λογισμικού σε αναδιανεμητική βάση από έναν πάροχο υπηρεσιών νέφους. Ο καταναλωτής νοικιάζει τη χρήση μίας εφαρμογής και μπορεί να συνδεθεί σε αυτή μέσω ελαφρού τερματικού διεπαφής όπως ένας διαδικτυακός φυλλομετρητής ή μέσω ενός προγράμματος διεπαφής. Όλη η υποκείμενη υποδομή, το ενδιάμεσο λογισμικό, το λογισμικό και τα δεδομένα της εφαρμογής βρίσκονται στο κέντρο δεδομένων του παρόχου της υπηρεσίας. Ο πελάτης μπορεί να διαχειριστεί και να ελέγξει μόνο συγκεκριμένες, ειδικές για το χρήστη ρυθμίσεις, συνηθέστερα με την υπογραφή της κατάλληλης συμφωνίας υπηρεσιών.

1.3 Μοντέλα διάρθρωσης και ανάπτυξης πόρων των υπολογιστικών νεφών

Ένα μοντέλο ανάπτυξης πόρων νέφους αντιπροσωπεύει ένα συγκεκριμένο τύπο περιβάλλοντος νέφους, το οποίο ξεχωρίζει πρωτίστως από την ιδιοκτησία, το μέγεθος και την πρόσβαση. Τα κυριότερα μοντέλα ανάπτυξης είναι τα ιδιωτικά νέφη, τα δημόσια νέφη, τα κοινοτικά νέφη και τα υβριδικά νέφη.

- Δημόσιο νέφος: Ένα νέφος λέγεται δημόσιο όταν οι υπηρεσίες του αποδίδονται μέσω ενός δικτύου που είναι ανοιχτό για δημόσια χρήση. Τα δημόσια νέφη ανήκουν και χειρίζονται από τρίτο πάροχο υπηρεσιών νέφους (third-party cloud provider). Όλο το υλικό, το λογισμικό και η υποστηρικτική υποδομή ανήκει και διαχειρίζεται από τον πάροχο και βρίσκεται στα δικά του κέντρα δεδομένων. Τεχνικά, μπορεί να υπάρχει μικρή ή καθόλου διαφορά ανάμεσα στην αρχιτεκτονική ενός ιδιωτικού και ενός δημόσιου νέφους. Αυτό που διαφέρει χαρακτηριστικά είναι οι πτυχές ασφαλείας για τις υπηρεσίες που είναι διαθέσιμες από ένα πάροχο για ένα δημόσιο ακροατήριο και για συνδέσεις που επηρεάζονται από ένα μη-έμπιστο δίκτυο. Ο καταναλωτής μπορεί να έχει πρόσβαση και να διαχειρίζεται το λογαριασμό του μέσω ενός διαδικτυακού φυλλομετρητή.
- Ιδιωτικό νέφος: Ένα νέφος λέγεται ιδιωτικό όταν οι υπηρεσίες και η υποδομή συντηρούνται σε ένα ιδιωτικό δίκτυο. Αναφέρεται σε πόρους υπολογιστικού νέφους οι οποίοι χρησιμοποιούνται αποκλειστικά από μια επιχείρηση ή οργανισμό. Ένα ιδιωτικό νέφος μπορεί να είναι εγκατεστημένο στο κέντρο δεδομένων της εταιρίας ενώ κάποιες εταιρίες μισθώνουν τρίτους παρόχους για να εγκαταστήσουν το νέφος.

Το εγχείρημα σχεδιασμού ιδιωτικού νέφους επιβάλλει τη δέσμευση της εικονοποίησης του περιβάλλοντος της εταιρίας και την ανάγκη ο οργανισμός να επανεξετάσει τις επιλογές για τους ήδη υπάρχοντες πόρους. Μπορεί να οδηγήσει στη βελτίωση της επιχείρησης, αλλά σε κάθε βήμα του σχεδιασμού προκύπτουν σημαντικά θέματα ασφαλείας τα οποία πρέπει να αντιμετωπιστούν για να αποφευχθούν τα τρωτά σημεία.

- Κοινοτικό νέφος: Στα κοινοτικά νέφη η υποδομή διαμοιράζεται μεταξύ πολλών οργανισμών από μία συγκεκριμένη κοινότητα με κοινές ανάγκες, όπως ασφάλεια, αρμοδιότητες, υποχρεωτικότητα, τα οποία διαχειρίζονται είτε εσωτερικά είτε από τρίτους και φιλοξενούνται είτε εσωτερικά είτε εξωτερικά. Το κόστος διαμοιράζεται μεταξύ λιγότερων χρηστών από ότι σε ένα δημόσιο νέφος, αλλά περισσότερο από ένα ιδιωτικό, με αποτέλεσμα μόνο μερικές από τις δυνατότητες μείωσης του κόστους με τη χρήση υπολογιστικών νεφών να είναι εμφανείς.
- Υβριδικό νέφος: Ένα υβριδικό νέφος είναι η σύνθεση δύο ή περισσοτέρων νεφών (ιδιωτικό, δημόσιο ή κοινοτικό) που παραμένουν ξεχωριστές οντότητες αλλά είναι άρρηκτα συνδεδεμένες ώστε να προσφέρουν τα πλεονεκτήματα πολλαπλών μοντέλων. Επιτρέποντας τα δεδομένα και τις εφαρμογές να μετακινούνται μεταξύ ιδιωτικών και δημοσίων (κυρίως) νεφών, τα υβριδικά νέφη προσφέρουν στις επιχειρήσεις μεγαλύτερη ευελιξία και περισσότερες αναπτυξιακές επιλογές. Οι αρχιτεκτονικές για την ανάπτυξη ενός υβριδικού νέφους είναι πολύπλοκες για να αναπτυχθούν και να συντηρηθούν λόγω της πιθανής ανομοιότητάς των διαφορετικών περιβαλλόντων και λόγω του γεγονότος ότι οι διοικητικές αρμοδιότητες μοιράζονται μεταξύ των παρόχων δημοσίου και ιδιωτικού νέφους. Ένα πολύ χαρακτηριστικό παράδειγμα χρήσης των υβριδικών νεφών είναι ένας οργανισμός τεχνολογίας πληροφοριών (IT) να χρησιμοποιεί πόρους ενός δημόσιου νέφους για να καλύψει παροδικές ανάγκες χωρητικότητας που δεν μπορούν να καλυφθούν από το ιδιωτικό νέφος.

Εκτός των παραπάνω πιο διαδεδομένων αναπτυξιακών μοντέλων έχουν αρχίσει να αναπτύσσονται και άλλα μοντέλα, λιγότερο διαδεδομένα, όπως τα κατανεμημένα νέφη (distributed cloud), οι συνομοσπονδίες νεφών (Intercloud ή Cloud Federation) και τα πολλαπλά νέφη (Multicloud). Τα μοντέλα αυτά, κυρίως λόγω τεχνικών δυσκολιών, δεν έχουν αναπτυχθεί αρκετά και δεν έχουν ευρεία χρήση στις μέρες μας.

1.4 Η ανάγκη δημιουργίας και χρήσης συνομοσπονδιών νέφους

Η χρήση της τεχνολογίας των υπολογιστικών νεφών αυξάνεται ολοένα και περισσότερο ειδικά στον τομέα παροχής υλικού και λογισμικού ως υπηρεσία νέφους. Η δημιουργία συνομοσπονδίας νεφών κρίνεται απαραίτητη και έχει ήδη χρησιμοποιηθεί για να περιγράψει τα κέντρα δεδομένων του μέλλοντος. Η συνομοσπονδία νεφών είναι ένα παγκόσμιο διασυνδεδεμένο «νέφος νεφών» και είναι μία επέκταση του διαδικτύου ως «δίκτυο δικτύων» στο οποίο και έχει βασιστεί ως ιδέα.

Η ιδέα δημιουργίας συνομοσπονδιών νέφους έχει βασιστεί στην βασική έννοια ότι ένας μόνο πάροχος δεν έχει άπειρους φυσικούς πόρους ή καθολικό γεωγραφικό ίχνος. Αν επέλθει κορεσμός των υπολογιστικών και αποθηκευτικών πόρων της υποδομής ενός νέφους ή του ζητηθεί να χρησιμοποιήσει πόρους σε περιοχή που δεν διαθέτει υποδομές, δεν θα είναι σε θέση να ικανοποιήσει τα αιτήματα των πελατών του για εκχώρηση υπηρεσιών. Η συνομοσπονδία νέφους θα αντιμετωπίζει τέτοιες καταστάσεις δίνοντας τη δυνατότητα σε ένα πάροχο να χρησιμοποιήσει τους διαθέσιμους πόρους της υποδομής ενός άλλου παρόχου που συμμετέχει στη συγκεκριμένη συνομοσπονδία.

Ένα από τα βασικότερα πλεονεκτήματα της δημιουργίας και χρήσης μίας συνομοσπονδίας νέφους είναι η δυνατότητα που δίνει στους παρόχους να διασυνδέσουν την τοπική υποδομή που διαθέτουν σε μία παγκόσμια αγορά. Οι συμμετέχοντες έχουν τη δυνατότητα να πουλούν και να αγοράζουν πόρους κατ' απαίτηση. Ως αποτέλεσμα ακόμα και οι μικροί πάροχοι μπορούν να προσφέρουν υπηρεσίες παγκοσμίως χωρίς να χρειάζεται να επενδύσουν για να χτίσουν νέες υποδομές. Επιπλέον, εταιρίες που έχουν διαθέσιμους πόρους στα κέντρα δεδομένων τους μπορούν να τους διαθέσουν στην αγορά για να τους αγοράσουν και να τους εκμεταλλευτούν άλλοι πάροχοι, αποκτώντας μία επιπλέον πηγή εισοδήματος.

Επιπρόσθετα, άμεσα οφέλη εμφανίζονται και για τους τελικούς χρήστες. Στη συνομοσπονδία νεφών ένας τελικός χρήστης μπορεί να φιλοξενήσει εφαρμογές από έναν πάροχο της επιλογής του αντί να επιλέξει από έναν περιορισμένο αριθμό παρόχων που προσφέρουν παγκόσμια κάλυψη στην αγορά σήμερα. Οι καταναλωτές μπορούν να επιλέξουν έναν τοπικό πάροχο με την τιμή, την τεχνογνωσία και το πακέτο που επιθυμούν που ταιριάζει στις ανάγκες τους. Ταυτόχρονα, λαμβάνουν άμεση πρόσβαση τόσο σε τοπικούς όσο και σε

παγκόσμιους υπολογιστικούς πόρους, χωρίς περιορισμό επιλογών και χωρίς να χρειάζεται να διαχειρίζονται πολλαπλούς παρόχους.

Κάθε καινοτόμα τεχνολογία έρχεται αντιμέτωπη με διάφορα προβλήματα και ανησυχίες. Στην συγκεκριμένη περίπτωση, οι βασικές ανησυχίες είναι η ασφάλεια των δεδομένων, η διαλειτουργικότητα των εφαρμογών και η εξάρτηση των καταναλωτών από έναν προμηθευτή. Παρόλα αυτά, το μοντέλο της συνομοσπονδίας νεφών δίνει τη δυνατότητα εκδημοκρατικοποίησης της αγοράς και οι καταναλωτές θα αναγνωρίσουν τις πραγματικές δυνατότητες των υπολογιστικών νεφών. Έτσι, οι χειριστές κέντρων δεδομένων και οι μικρότεροι πάροχοι υπηρεσιών νέφους θα μπορέσουν να ανταγωνιστούν τους τωρινούς μεγάλους παγκόσμιους παρόχους υπολογιστικών νεφών.

1.5 Αντικείμενο διπλωματικής εργασίας

Το αντικείμενο της διπλωματικής εργασίας είναι ο σχεδιασμός, η ανάπτυξη και η υλοποίηση πλατφόρμας για τη διαχείριση πόρων των επιμέρους υποδομών των υπολογιστικών νεφών. Η πλατφόρμα θα δέχεται αιτήματα χρηστών για τη δημιουργία εικονικών μηχανημάτων τα οποία θα έχουν συγκεκριμένες απαιτήσεις σε υπολογιστικούς πόρους όπως κεντρική μονάδα επεξεργασίας (CPU), μνήμη και δικτυακούς πόρους και θα αποφασίζει σύμφωνα με την υπάρχουσα κατανομή τη βέλτιστη ανάθεση ανά φυσικό εξυπηρετητή. Επίσης, η πλατφόρμα θα είναι υπεύθυνη για την δίκαιη αντιμετώπιση των χρηστών και για βασικά συστήματα ασφαλείας.

Για την περάτωση της εργασίας αυτής, θα πρέπει αρχικά να εξοικειωθούμε με τη χρήση του διαδικτυακού πλαισίου flask. Γνωρίζοντας το πώς αναπτύσσονται και υλοποιούνται πολλές δημοφιλείς εφαρμογές όπως το LinkedIn και το Pinterest, θα μας βοηθήσει να σχεδιάσουμε την πλατφόρμα μας στα πρότυπα αυτών των εφαρμογών και να αναπτύξουμε το backend του εξυπηρετητή μας.

Για την υλοποίηση της πλατφόρμας μας, χρησιμοποιήσαμε το πρωτόκολλο Representational state transfer (REST). Χρησιμοποιώντας ένα μη καταστατικό πρωτόκολλο στοχεύουμε στην γρήγορη απόδοση, στην αξιοπιστία και στην ικανότητα ανάπτυξης της εφαρμογής μας βασιζόμενοι στην επαναχρησιμοποίηση στοιχείων που μπορούν να διαχειριστούν και να ανανεωθούν χωρίς να επηρεάζουν το σύστημα μας.

Για την διαχείριση των δεδομένων μας θα χρησιμοποιηθεί το σύστημα βάσης δεδομένων MongoDB που προσφέρει ασφαλή και αξιόπιστη μόνιμη αποθήκευση. Δίνεται, επίσης, η δυνατότητα χρήσης μίας εκφραστικής γλώσσας αναζήτησης για να προσπελάσουμε και να διαχειριστούμε τα δεδομένα μας. Παράλληλα, προσφέρει μεγάλη απόδοση σε μεγάλη κλίμακα λόγω της οριζόντιας κλιμάκωσης της εκτελώντας εκατομμύρια λειτουργίες ανά δευτερόλεπτο.

Για την αντιμετώπιση του προβλήματος της δίκαιης αντιμετώπισης των παιχτών θα υλοποιηθεί μία παραλλαγή του αλγόριθμου glicko. Ο αλγόριθμος αυτός θα τροποποιηθεί για να μπορέσει να χρησιμοποιηθεί στο περιβάλλον της εφαρμογής μας, δηλαδή σε ένα μη ανταγωνιστικό περιβάλλον που οι χρήστες συνδέονται και συμμετέχουν εθελοντικά.

Τέλος, θα πραγματοποιηθεί μελέτη της απόδοσης της πλατφόρμας που αναπτύχθηκε για την διαχείριση συνομοσπονδιών νέφους. Σκοπός μας είναι η ανάπτυξη πλατφόρμας που θα μπορεί να ανταπεξέρχεται και να λειτουργεί ικανοποιητικά σε μεγάλο αριθμό χρηστών. Απαιτείται, φυσικά, καλή γνώση της γλώσσας προγραμματισμού Python με τη βοήθεια της οποίας θα αναπτυχθεί η πλατφόρμα μας.

1.6 Διάρθρωση της εργασίας

Στα πλαίσια της διπλωματικής εργασίας προσπαθήσαμε να προσεγγίσουμε τη δημιουργία της πλατφόρμας βήμα προς βήμα, όπως θα παρουσιαστεί στα επόμενα κεφάλαια. Στο δεύτερο κεφάλαιο παρουσιάζονται όλες οι απαραίτητες τεχνολογίες για την ανάπτυξη ενός τέτοιου project. Στο τρίτο κεφάλαιο, γίνεται η περιγραφή των απαιτήσεων που πρέπει να ικανοποιεί η πλατφόρμα και η ανάλυση των λειτουργιών και των σεναρίων χρήσης. Στη συνέχεια, το τέταρτο κεφάλαιο αναφέρεται στην αρχιτεκτονική του συστήματος μας και γίνεται περιγραφή των εκάστοτε λεπτομερειών της σχεδίασης. Στο πέμπτο κεφάλαιο καταγράφονται οι μέθοδοι υλοποίησης των διαφόρων υποσυστημάτων, και παρουσιάζονται μετρήσεις της απόδοσης των διάφορων λειτουργιών της πλατφόρμας καθώς και ένα πλήρες σενάριο χρήσης. Τέλος, στο έκτο κεφάλαιο, καταγράφονται συμπεράσματα και προτείνονται πιθανές μελλοντικές επεκτάσεις.

2

Τεχνολογίες

2.1 Python

Η Python είναι μια ευρέως διαδεδομένη γλώσσα προγραμματισμού υψηλού επιπέδου, γενικού σκοπού, λειτουργεί με διερμηνευτή (interpreter) και ανήκει στις δυναμικές γλώσσες προγραμματισμού. Η φιλοσοφία σχεδίασης της δίνει έμφαση στην αναγνωσιμότητα του κώδικα και η σύνταξη της επιτρέπει στους προγραμματιστές να εκφράζονται χρησιμοποιώντας λιγότερες γραμμές κώδικα σε σχέση με γλώσσες όπως τη C++ ή τη Java παρέχοντας, έτσι, μειωμένο κόστος συντήρησης των προγραμμάτων. Η γλώσσα προσφέρει τη δυνατότητα για παραγωγή καθαρού και σαφή κώδικα τόσο σε μικρή όσο και μεγάλη κλίμακα.

Η υψηλού επιπέδου σχεδίαση της για δομές δεδομένων συνδυασμένη με τη δυνατότητα δυναμικής σύνδεσης κάνει την Python μια γλώσσα πολύ ελκυστική για ταχεία ανάπτυξη εφαρμογών, για προγράμματα εφαρμογής (γνωστά και ως scripts) και ως συνδετική γλώσσα για τη διασύνδεση ήδη υπαρχουσών στοιχείων. Υποστηρίζει, επίσης, πακέτα και πρότυπα τα οποία ενθαρρύνουν την αρθρωτή δομή και την επαναχρησιμοποίηση κώδικα. Ο διερμηνευτής της Python και οι εκτεταμένες τυποποιημένες βιβλιοθήκες είναι διαθέσιμα είτε σε δυαδική είτε σε πηγαία μορφή χωρίς χρέωση για όλες τις κύριες πλατφόρμες και μπορούν να αναδιανεμηθούν ελεύθερα.

Η Python χρησιμοποιεί δυναμικό σύστημα τύπων και αυτόματη διαχείριση μνήμης και έχει μεγάλη ποικιλία τυποποιημένων βιβλιοθηκών, όπως προαναφέρθηκε. Υποστηρίζει ταυτόχρονα ένα πλήθος προγραμματιστικών μοντέλων, από τα βασικότερα να είναι αντικειμενοστραφή, συναρτησιακό και προστακτικό προγραμματισμό.

Ένα ακόμα βασικό χαρακτηριστικό της Python που την κάνει πολύ δημοφιλή μεταξύ των προγραμματιστών είναι η αυξημένη παραγωγικότητα που παρέχει. Εφόσον δεν υφίσταται το

στάδιο της μεταγλώττισης, η επεξεργασία, το τεστ και το debug γίνονται εξαιρετικά γρήγορα. Το debugging προγραμμάτων Python είναι αρκετά εύκολο αφού ένα bug ή μία λάθος είσοδος δεν θα προκαλέσει ποτέ λάθος κατάτμησης (segmentation fault). Αντ' αυτού, όταν ο διερμηνευτής εντοπίσει ένα λάθος θα προκαλέσει μία εξαίρεση. Ο debugger της Python είναι γραμμένος, επίσης, σε Python επιβεβαιώνοντας την εσωστρεφή δύναμη της γλώσσας.

Οι διερμηνευτές της Python είναι διαθέσιμοι σε πολλά λειτουργικά συστήματα, επιτρέποντας έτσι στον κώδικα να τρέχει σε μεγάλη ποικιλία συστημάτων. Χρησιμοποιώντας διαθέσιμα εργαλεία όπως το Py2exe ή το Pyinstaller, ο κώδικας γραμμένος σε Python μπορεί να μετατραπεί σε ανεξάρτητα εκτελέσιμα αρχεία για κάποια από τα πιο δημοφιλή λειτουργικά συστήματα. Λογισμικό, λοιπόν, γραμμένο στη γλώσσα αυτή μπορεί να αναδιανεμηθεί και να χρησιμοποιηθεί χωρίς την ανάγκη εγκατάστασης κάποιου διερμηνευτή σε διάφορα περιβάλλοντα.

Τα χαρακτηριστικά της Python και τα πλεονεκτήματα της έναντι άλλων γλωσσών προγραμματισμού την κατατάσσουν στις πρώτες δέκα πιο δημοφιλής γλώσσες.

2.2 Flask

Το Flask είναι ένα πλαίσιο ιστού (web framework) γραμμένο σε Python, αδειοδοτημένο με BSD license, που ανήκει στην κατηγορία των ανεκτικών αδειών ελεύθερου λογισμικού. Παρέχει εργαλεία, βιβλιοθήκες και τεχνολογίες που επιτρέπουν της σχεδίαση μιας διαδικτυακής εφαρμογής ή υπηρεσίας, χωρίς ο προγραμματιστής να χρειάζεται να χειριστεί χαμηλού επιπέδου λεπτομέρειες όπως πρωτόκολλα ή διεργασίες και νήματα. Αυτή η εφαρμογή μπορεί να είναι μία απλή ιστοσελίδα, ένα blog ή ακόμα και μία πολύπλοκη εμπορική ιστοσελίδα.

Το flask δεν έχει υπόστρωμα βάσης δεδομένων, μορφή επικύρωσης ή άλλα στοιχεία που βιβλιοθήκες τρίτων προσφέρουν κοινές συναρτήσεις. Παρόλα αυτά, το Flask υποστηρίζει επεκτάσεις που μπορούν να προσθέσουν στην εφαρμογή χαρακτηριστικά σαν αυτή να ήταν υλοποιημένη στο Flask. Επεκτάσεις υπάρχουν για αντικειμενοστραφή προγράμματα αντιστοίχισης, μορφές επικύρωσης, χειρισμό αναφόρτωσης και για διάφορες ελεύθερες τεχνολογίες επαλήθευσης καθώς και πολλά εργαλεία σχετικά με τα κοινώς χρησιμοποιούμενα πλαίσια. Οι επεκτάσεις αυτές ενημερώνονται πιο συχνά από τον πυρήνα του Flask κάνοντάς το ένα από τα πιο δημοφιλή εργαλεία ανάπτυξης ιστοσελίδων.

Πιο συγκεκριμένα, το flask ανήκει στην κατηγορία των μικρο-πλαισίων. Αυτή η κατηγορία πλαισίων συνήθως, έχει μικρή ή καθόλου εξάρτηση σε εξωτερικές βιβλιοθήκες. Αυτό με τη σειρά του έχει κάποια πλεονεκτήματα και μειονεκτήματα. Βασικό πλεονέκτημα είναι ότι το πλαίσιο είναι ελαφρύ, υπάρχουν λίγες εξαρτήσεις για ενημέρωση και δεν χρειάζεται επιπλέον παρατήρηση για bug ασφαλείας. Το μειονέκτημα είναι ότι κάποιες φορές χρειάζεται ο προγραμματιστής να κάνει περισσότερη δουλειά μόνος του ή να αυξήσει τη λίστα των εξαρτήσεων προσθέτοντας συνδεόμενες υπο-μονάδες (plug-ins) και επεκτάσεις (extensions).

Λόγω της απλότητας του Flask, μπορεί να προσφέρει, μέσω των extensions και των plug-ins, μεγάλο αριθμό μηχανισμών που βρίσκουν εφαρμογή για την ανάπτυξη πληθώρας εφαρμογών και υπηρεσιών. Ακολουθεί η παράθεση χαρακτηριστικών του Flask που χρησιμοποιήσαμε για την ανάπτυξη της πλατφόρμας και κάποιων επιπλέον που θεωρούμε απαραίτητα για την ευρύτερη κατανόηση των μηχανισμών που χρησιμοποιήθηκαν.

Το Flask βασίζεται στη Διεπαφή Πύλης Διακομιστή Ιστού (Web Server Gateway Interface ή WSGI) Werkzeug, το οποίο περιέχει ποικίλα εργαλεία διεπαφής (utility modules), προσδίδοντας του λειτουργίες και χαρακτηριστικά, τα βασικότερα από τα οποία είναι:

- Ανάλυση και απόρριψη κεφαλίδων HTTP
- Αντικείμενα (objects) αιτημάτων και απαντήσεων HTTP εύκολης χρήσης
- Υποστήριξη Unicode(διεθνές πρότυπο για την κωδικοποίηση όλων των συστημάτων γραφής) και εργαλεία διαχείρισης Ενιαίων Εντοπιστών Αναγνωριστικών (Uniform Resource Identifier ή URI) και Διεθνοποιημένων Εντοπιστών Αναγνωριστικών (International Resource Identifier ή IRI) βασισμένων σε Unicode. Το URI είναι μία συμβολοσειρά χαρακτήρων που χρησιμοποιείται για την αναγνώριση πόρων συστήματος (system resource), συνήθως πόρων που βρίσκονται σε ένα δίκτυο. Το IRI είναι μία επέκταση του URI.
- Ολοκληρωμένο σύστημα δρομολόγησης για την αντιστοίχιση Ενιαίων Εντοπιστών Πόρων (Uniform Resource Locator ή URL) με άκρα (endpoints) εφαρμογών και αντίστροφα. Το URL είναι ένας συγκεκριμένος τύπος URI, που χρησιμοποιείται, συνήθως, για αναφορές σε πόρους που χρησιμοποιούν το πρωτόκολλο HTTP, αλλά και άλλα πρωτόκολλα που δεν αξίζουν αναφοράς στην παρούσα εργασία.

Οι επεκτάσεις του flask που χρησιμοποιήθηκαν είναι:

- flask_restful: Προσφέρει υποστήριξη για τη γρήγορη ανάπτυξη εφαρμογών REST.
- flask_pymongo: Γεφυρώνει το περιβάλλον του Flask με τη διεπαφή προγραμματισμού εφαρμογών(Application Programming Interface ή API) PyMongo, έτσι ώστε να χρησιμοποιούνται οι κανονικοί μηχανισμοί του Flask για τη σύνδεση με τη βάση δεδομένων MongoDB, καθώς και την διαμόρφωση της.
- flask_httpauth: Μία απλή επέκταση που απλοποιεί τη χρήση της ταυτοποίησης μέσω HTTP των διαδρομών του Flask.

2.3 REST API

Οι Representational state transfer (REST) διαδικτυακές εφαρμογές παρέχουν διαλειτουργικότητα μεταξύ υπολογιστικών συστημάτων στο διαδίκτυο. Οι υπηρεσίες που ακολουθούν το πρωτόκολλο αυτό, μπορούν να έχουν πρόσβαση και να διαχειρίζονται διαδικτυακούς πόρους σε μορφή κειμένου χρησιμοποιώντας ένα ομοιόμορφο και προκαθορισμένο σύνολο από ατομικές λειτουργίες.

Οι διαδικτυακοί πόροι είχαν, αρχικά, οριστεί στον Παγκόσμιο Ιστό ως έγγραφα και αρχεία τα οποία ορίζονταν από μία διεύθυνση URL. Σήμερα, όμως, έχουν μία πιο αφηρημένη μορφή συμπεριλαμβάνοντας οποιαδήποτε οντότητα μπορεί να αναγνωριστεί, ονομαστεί ή χειριστεί στο διαδίκτυο. Σε μία REST υπηρεσία, τα αιτήματα που γίνονται στο αναγνωριστικό ενός πόρου, μπορούν να λάβουν απόκριση της μορφής XML,HTML ή JSON. Η απάντηση μπορεί να επιβεβαιώνει ότι έχει γίνει κάποια αλλαγή στον αποθηκευμένο πόρο, παρέχοντας έτσι συνδέσμους υπερκειμένου με άλλους συναφείς πόρους. Η μετάδοση της περισσότερες φορές γίνεται μέσω του πρωτοκόλλου HTTP, και περιλαμβάνονται οι προκαθορισμένες λειτουργίες του όπως οι GET, PUT, POST, DELETE, PATCH.

Οι περιορισμοί της αρχιτεκτονικής του REST επηρεάζουν κάποιες βασικές αρχιτεκτονικές ιδιότητες:

- Απόδοση: η αλληλεπίδραση των στοιχείων είναι ένας καθοριστικός παράγοντας για την αποδοτικότητα του δικτύου.
- Κλιμακωσιμότητα για να υποστηρίξει μεγάλο αριθμό στοιχείων και αλληλεπίδραση μεταξύ των στοιχείων.
- Απλότητα μιας ομοιόμορφης διεπαφής.

- Επιδεκτικά σε τροποποίηση στοιχεία για να ανταπεξέλθουν στις μεταβαλλόμενες ανάγκες ακόμα και όταν η εφαρμογή εκτελείται.
- Ορατή επικοινωνία μεταξύ των στοιχείων.
- Φορητότητα των στοιχείων μεταφέροντας κώδικα προγράμματος μαζί με τα δεδομένα.
- Αξιοπιστία καθώς παρέχεται ανοχή σε σφάλματα σε συστημικό επίπεδο παρά τα όποια σφάλματα σε στοιχεία, συνδέσμους ή δεδομένα.

Για να οριστεί ένα σύστημα ως RESTful και το σύστημα να κερδίζει τα επιθυμητά χαρακτηριστικά που προαναφέρθηκαν πρέπει να τηρούνται κάποιοι περιορισμοί. Αρχικά, πρέπει να τηρείται η αρχιτεκτονική πελάτη-εξυπηρετητή ώστε να διαχωρίζονται οι ευθύνες του καθενός. Η ατομικότητα των λειτουργιών είναι ένας απαραίτητος περιορισμός. Το αίτημα από κάθε πελάτη πρέπει να περιέχει όλες τις απαραίτητες πληροφορίες για την εξυπηρέτηση του αιτήματος και η κατάσταση της περιόδου να διατηρείται στον πελάτη. Τέλος, το σύστημα θα πρέπει να είναι δομημένο σε στρώματα, να διατηρεί ένα ομοιόμορφο σύστημα διεπαφής και να έχει τη δυνατότητα αποθήκευσης των απαντήσεων σε κρυφές μνήμες για να αποφεύγεται η χρήση ξεπερασμένων ή ακατάλληλων δεδομένων σε απαντήσεις περαιτέρω αιτημάτων.

2.4 MongoDB

Το MongoDB είναι ένα ελεύθερο και ανοιχτού κώδικα σύστημα διαχείρισης βάσεων δεδομένων, το οποίο αποθηκεύει τα δεδομένα σε μορφή εγγράφων και είναι ανεξάρτητο πλατφόρμας. Ανήκει στην κατηγορία των NoSQL βάσεων δεδομένων αποφεύγοντας έτσι την παραδοσιακή δομή των σχεσιακών βάσεων με χρήση πινάκων. Αντ' αυτού, αποθηκεύει τα δεδομένα σε ευέλικτα έγγραφα μορφής JSON (πιο συγκεκριμένα αρχεία BSON), το οποίο σημαίνει ότι τα πεδία μπορούν να διαφέρουν από έγγραφο σε έγγραφο και οι δομές δεδομένων μπορούν να αλλάζουν με το χρόνο. Προσφέρει, έτσι, ευκολότερη και γρηγορότερη ένταξη των δεδομένων στις σύγχρονες εφαρμογές. Αναπτύσσεται από την MongoDB Inc. και τηρεί τους όρους των αδειών GNU Affero General Public License και Apache License.

Κάποια από τα πιο βασικά χαρακτηριστικά της βάσης δεδομένων MongoDB είναι:

- Ad hoc αναζητήσεις: Το MongoDB υποστηρίζει αναζητήσεις ανά πεδίο, ερωτήματα εύρους και κανονικές εκφράσεις. Οι αναζητήσεις μπορούν να επιστρέψουν συγκεκριμένα πεδία των εγγράφων και μπορούν να περιλαμβάνουν ορισμένες από το χρήστη συναρτήσεις σε JavaScript. Οι αναζητήσεις μπορούν, επίσης, να διευθετηθούν ώστε να επιστρέφεται ένα τυχαίο δείγμα αποτελεσμάτων συγκεκριμένου μεγέθους.

- Ευρετηριοποίηση: Τα πεδία σε ένα έγγραφο MongoDB μπορούν να ταξινομηθούν με πρωταρχικούς και δευτερεύοντες δείκτες.
- Replication: Το MongoDB έχει μεγάλη διαθεσιμότητα σε σύνολα αντιγράφων (replica sets). Τα σύνολα αυτά αποτελούνται από δύο ή περισσότερα αντίγραφα των δεδομένων και προσφέρουν υψηλή διαθεσιμότητα και ασφάλεια. Κάθε αρχείο μπορεί να συμπεριφερθεί ως πρωτεύον ή δευτερεύον αντίγραφο οποιαδήποτε στιγμή αλλά όλες οι εγγραφές και οι αναγνώσεις γίνονται στο πρωτεύον αντίγραφο αυτόματα.
- Διαμοιρασμός φόρτου: χρησιμοποιεί shards για να μπορεί να κλιμακώσει οριζόντια, δηλαδή τα δεδομένα κατανέμονται και χωρίζονται σε εύρη, και στη συνέχεια αποθηκεύονται σε διαφορετικά shards τα οποία μπορούν να βρίσκονται σε διαφορετικούς εξυπηρετητές.
- Αποθήκευση αρχείων: μπορεί να χρησιμοποιηθεί ως σύστημα αρχείων με χαρακτηριστικά διαμοιρασμού φόρτου και αναπαραγωγή δεδομένων (data replication) σε πολλαπλά μηχανήματα για αποθήκευση αρχείων.
- Δυνατότητα χρήσης του αλγορίθμου MapReduce για επεξεργασία δεδομένων σε μεγάλα κομμάτια και λειτουργίες σύνοψης.
- Εκτέλεση JavaScript στην πλευρά του εξυπηρετητή: Μπορεί να χρησιμοποιηθεί JavaScript σε αναζητήσεις, συναρτήσεις σύνοψης και να σταλεί κατευθείαν στη βάση δεδομένων για εκτέλεση.

2.5 JSON

Στα υπολογιστικά συστήματα, η μορφή αρχείων JavaScript Object Notation ή JSON είναι ένα πρότυπο ανοιχτού κώδικα που χρησιμοποιεί κείμενο κατανοητό από ανθρώπους για την αποστολή δεδομένων σε ζευγάρια στοιχείων-πληροφορίας (attribute-value) και σε μορφή πινάκων δεδομένων. Είναι μία πολύ κοινή μορφή αρχείων που χρησιμοποιείται κυρίως για την ασύγχρονη επικοινωνία μεταξύ φυλλομετρητών (browser) και εξυπηρετητών, αντικαθιστώντας τον προκάτοχο του, το XML.

Η μορφή αρχείων JSON είναι ανεξάρτητη της γλώσσας προγραμματισμού, παρότι αρχικά ξεκίνησε από τη γλώσσα JavaScript. Πλέον, οι περισσότερες γλώσσες προγραμματισμού περιέχουν κώδικα για την παραγωγή και συντακτική ανάλυση δεδομένων σε μορφή JSON. Τα αρχεία αυτής της μορφής χρησιμοποιούν την επέκταση ".json". Οι βασικές μορφές δεδομένων που υποστηρίζονται από αυτό το πρότυπο αυτού είναι:

- Number: Προσημασμένος δεκαδικός αριθμός που μπορεί να περιέχει δεκαδικό μέρος και να χρησιμοποιεί εκθετικό e μορφής αλλά δεν μπορεί να περιέχει μη αριθμητικά

δεδομένα όπως το NaN. Δεν υπάρχει διάκριση ανάμεσα σε ακεραίους και σε μονάδες κινητής υποδιαστολής.

- String: Ακολουθία από μηδέν ή περισσότερους χαρακτήρες μορφής Unicode. Τα string διαχωρίζονται από διπλά εισαγωγικά.
- Boolean: μπορεί να πάρει τιμή μια εκ των TRUE ή FALSE
- Array: Μια διατεταγμένη λίστα με τιμές που μπορεί να ανήκουν σε οποιαδήποτε μορφή. Οι πίνακες συμβολίζονται με αγκύλες και τα στοιχεία τους διαχωρίζονται με κόμμα.
- Object: Μια μη διατεταγμένη συλλογή από ζεύγη κλειδιών/τιμών που τα κλειδιά είναι της μορφής string. Τα αντικείμενα αυτά έχουν ως σκοπό να αναπαριστούν σχεσιακούς πίνακες και το κάθε κλειδί θα πρέπει να είναι μοναδικό. Τα αντικείμενα συμβολίζονται με αγκιστροειδείς αγκύλες, χρησιμοποιούν κόμμα για διαχωρισμό των ζευγών και τα κλειδιά διαχωρίζονται από τις τιμές με άνω-κάτω τελεία.
- Null: Μία κενή τιμή που ορίζεται με τη χρήση της λέξης null

2.6 HTTP

Το πρωτόκολλο μεταφοράς υπερκειμένου(HTTP) είναι ένα πρωτόκολλο εφαρμογών για κατανεμημένα, συνεργατικά και υπερμέσα επικοινωνιακά συστήματα. Το HTTP είναι η θεμελιώδης δομή για επικοινωνία δεδομένων στον Παγκόσμιο Ιστό. Τα υπερκείμενα είναι δομημένα κείμενα που χρησιμοποιούν λογικούς συνδέσμους μεταξύ κόμβων που περιέχουν κείμενο. Το πρωτόκολλο αυτό χρησιμοποιείται για την ανταλλαγή και μεταφορά υπερκειμένου. Λειτουργεί ως ένα πρωτόκολλο αιτημάτων-απαντήσεων στο υπολογιστικό μοντέλο του πελάτη-εξυπηρετητή και οι πόροι του δεικτοδοτούνται και βρίσκονται στο δίκτυο από τον ενιαίο εντοπιστή πόρων (Uniform Resource Locator-URL), χρησιμοποιώντας την απεικόνιση του Ενιαίου Αναγνωριστικού Πόρων(Uniform Resource Identifier-URI) http και https. Για τον καθορισμό της επιθυμητής πράξης που θα πραγματοποιηθεί σε έναν πόρο το πρωτόκολλο αυτό ορίζει κάποιες συγκεκριμένες μεθόδους. Ορίζονται οχτώ βασικές μέθοδοι απτό τις προδιαγραφές του HTTP/1.1 ενώ δεν υπάρχει περιορισμός στον αριθμό των μεθόδων που μπορούν να οριστούν για μελλοντική χρήση χωρίς βλάβη της υπάρχουσας υποδομής. Οι οχτώ βασικές μέθοδοι είναι:

- GET: Η μέθοδος αυτή ζητάει την απεικόνιση ενός συγκεκριμένου πόρου. Τα αιτήματα που χρησιμοποιούν αυτή τη μέθοδο πρέπει μόνο να ανακτούν τα δεδομένα χωρίς κάποια άλλη επίδραση.

- HEAD: Η μέθοδος αυτή ζητά για μια απάντηση παρόμοια με το GET, αλλά χωρίς το σώμα της απάντησης. Είναι χρήσιμη για την ανάκτηση πληροφοριών που είναι γραμμένες στην επικεφαλίδα των απαντήσεων, χωρίς την μεταφορά ολόκληρου του περιεχομένου.
- POST: Η μέθοδος αυτή ζητά από τον εξυπηρετητή να δεχθεί την οντότητα που εσωκλείεται στο αίτημα ως δευτερεύουσα πληροφορία του διαδικτυακού πόρου που ορίζεται από το URI.
- PUT: Η μέθοδος αυτή ζητά την αποθήκευση της εσωκλειόμενης οντότητας στο παρεχόμενο URI. Αν το URI αναφέρεται σε έναν υπάρχον πόρο, αυτός ο πόρος μεταβάλλεται ενώ αν το URI δεν αναφέρεται σε κάποιο πόρο τότε ο εξυπηρετητής δημιουργεί αυτό τον πόρο με το συγκεκριμένο URI.
- DELETE: Η μέθοδος αυτή διαγράφει τον συγκεκριμένο πόρο.
- TRACE: Με τη μέθοδο αυτή ένας πελάτης μπορεί να δει τις αλλαγές ή τις προσθήκες που έχουν γίνει από ενδιάμεσους εξυπηρετητές.
- OPTIONS: Η μέθοδος αυτή επιστρέφει τις μεθόδους HTTP που υποστηρίζονται από τη συγκεκριμένη URL. Μπορεί να χρησιμοποιηθεί για τον έλεγχο της λειτουργικότητας ενός εξυπηρετητή ζητώντας "*" αντί για συγκεκριμένο πόρο.
- CONNECT: Η μέθοδος αυτή μετατρέπει την ζητούμενη σύνδεση σε ένα διαφανές κανάλι TCP/IP, που συνήθως χρησιμοποιείται για την δημιουργία μιας κρυπτογραφημένης SSL επικοινωνίας μέσω ενός μη κρυπτογραφημένου HTTP proxy.
- PATCH: Η μέθοδος αυτή θέτει συγκεκριμένες και τμηματικές τροποποιήσεις σε έναν συγκεκριμένο πόρο.

Κάποιες από τις μεθόδους αυτές (GET,TRACE,HEAD,OPTIONS) ορίζονται, κατά σύμβαση, ως ασφαλείς το οποίο σημαίνει ότι έχουν σκοπό μόνο την ανάκτηση πληροφοριών και δεν πρέπει να επηρεάζουν την κατάσταση του εξυπηρετητή. Αντίθετα, οι υπόλοιπες μέθοδοι όπως το PUT,POST,PATCH, έχουν ως σκοπό να μεταβάλλουν κάποιους πόρους έχοντας παρενέργειες είτε στην πλευρά του εξυπηρετητή, είτε εξωτερικές όπως πχ στην μετάδοση ενός email. Παρόλα αυτά, ο χειρισμός των μεθόδων που περιγράφονται ως ασφαλείς δεν επιδέχεται περιορισμό με αποτέλεσμα ο απρόσεκτος ή ο εσκεμμένα κακός προγραμματισμός να προκαλέσει σημαντικά προβλήματα στην κατάσταση του εξυπηρετητή.

3

Ανάλυση

3.1 Απαιτήσεις πλατφόρμας συνομοσπονδιών νέφους

Το cloud computing είναι ένα σχετικά καινούργιο πρότυπο λειτουργίας για μεγάλα κατανεμημένα κέντρα δεδομένων. Η δυνατότητα του να προσφέρει στους καταναλωτές υπηρεσίες σε αναδιανεμητική βάση είναι ένα τεράστιο βήμα για τα υπολογιστικά συστήματα. Έδωσε την ώθηση για ένα μοντέλο λειτουργίας ανάλογο του μοντέλου προσφοράς ηλεκτρικής ενέργειας, κινητών τηλεπικοινωνιών με πιο πρόσφατο παράδειγμα όμως να είναι αυτό του διαδικτύου. Η ικανότητα, όμως, των παρόχων να διαθέτουν φυσικούς πόρους και καθολικό γεωγραφικό ίχνος είναι περιορισμένη, ακόμα και για τους πολύ μεγάλους παρόχους. Κρίνεται, λοιπόν, απαραίτητο να δοθεί η δυνατότητα στους παρόχους να ενωθούν και να λειτουργήσουν ως μια ομοσπονδία.

Για να πραγματοποιηθεί ένα τέτοιο σχέδιο που θα δίνει την ευκαιρία στους παρόχους να συνεργαστούν, χρησιμοποιώντας τους διαθέσιμους πόρους ενός άλλου παρόχου για να ικανοποιήσουν αιτήματα εκχώρησης υπηρεσιών που σε άλλη περίπτωση δεν θα μπορούσαν, χρειάζεται η δημιουργία μίας πλατφόρμας που θα λειτουργεί ως διαμεσολαβητής.

Για να είναι χρήσιμη και λειτουργική μία τέτοια πλατφόρμα θα πρέπει να έχει κάποια βασικά χαρακτηριστικά και να πληροί κάποιες βασικές προδιαγραφές. Μερικά από αυτά μπορεί να είναι:

- Δυνατότητα να δέχεται αιτήματα από τους παρόχους-πελάτες οι οποίοι θέλουν να εγγραφούν εθελοντικά στο σύστημα και καταχώρηση αυτών σε μία βάση δεδομένων.
- Δυνατότητα να δέχεται από τα συμμετέχοντα μέλη, αιτήματα για καταχώρηση των πόρων που θέλουν να διαθέσουν στη συνομοσπονδία για χρήση από τα άλλα μέλη. Καταχώρηση των πόρων αυτών σε μια βάση δεδομένων για περαιτέρω χρήση.

- Δυνατότητα να δέχεται αιτήματα από τα ήδη εγγεγραμμένα μέλη για αναζήτηση συγκεκριμένων πόρων, διαθέσιμων στη συνομοσπονδία. Εν συνεχεία, η πλατφόρμα θα πρέπει να μπορεί να επιλέξει τον βέλτιστο-πάροχο μέλος για να ικανοποιήσει το παραπάνω αίτημα.
- Λειτουργώντας ως συνομοσπονδία και ως ένα εθελοντικό σύστημα στο οποίο οι πάροχοι-πελάτες συμμετέχουν κατ' επιλογήν και χωρίς επιπλέον χρέωση, κρίνεται απαραίτητη η δημιουργία και η χρήση αλγορίθμου για τη δίκαιη αντιμετώπιση των χρηστών.
- Θα πρέπει μπορεί να αντιμετωπίζει βασικά ζητήματα ασφαλείας. Η πιστοποίηση ταυτότητας των χρηστών είναι μία πολύ βασική προϋπόθεση που πρέπει να πληρείται ώστε μία οντότητα να μην μπορεί να προσποιηθεί ότι είναι κάποια άλλη.

3.2 Η πλατφόρμα του Intercloud

Η πλατφόρμα που αναπτύσσουμε στην παρούσα διπλωματική εργασία ονομάστηκε, με βάση και τους διεθνείς όρους, Intercloud Exchange Platform. Είναι μία πλατφόρμα που υλοποιεί την ιδέα των συνομοσπονδιών νέφους και δίνει τη δυνατότητα στους χρήστες-παρόχους να διαμοιράζονται πόρους. Θα πρέπει να ικανοποιούνται κάποιες βασικές λειτουργίες, όπως αυτές που αναπτύχθηκαν παραπάνω. Χρειάζεται, επίσης, ο χρόνος απόκρισης της πλατφόρμας σε αιτήματα που δέχεται από χρήστες να είναι όσο το δυνατόν μικρότερος, ώστε να μπορούν να εξυπηρετηθούν ολοένα και περισσότερα αιτήματα.

Αρχικά, οι δραστηριότητες του χρήστη χωρίζονται σε δύο βασικές κατηγορίες, αυτή του πελάτη και αυτή του παρόχου, ανάλογα με την ιδιότητα του. Στο μοντέλο που αναπτύσσουμε ένας χρήστης μπορεί να έχει την ιδιότητα τόσο του παρόχου όσο και του πελάτη. Ανεξαρτήτως ιδιότητας θα πρέπει ο χρήστης να εγγραφεί στην συγκεκριμένη πλατφόρμα, στέλνοντας κατάλληλο αίτημα, δίνοντας το όνομα του ή το όνομα της επιχείρησης και έναν κωδικό πρόσβασης. Η εγγραφή του χρήστη είναι μία απαραίτητη προϋπόθεση που πρέπει να πληρείται για την σωστή λειτουργία της πλατφόρμας.

Η ανάγκη εγγραφής του χρήστη βασίστηκε στο τρίπτυχο εμπιστευτικότητα (confidentiality), ακεραιότητα (integrity), διαθεσιμότητα (availability) γνωστό και ως CIA. Η λήψη διάφορων μέτρων, όπως αυτή του κωδικού πρόσβασης-ταυτότητας χρήστη, εξασφαλίζει την εμπιστευτικότητα ώστε οι ευαίσθητες πληροφορίες να μην πέσουν στα χέρια των λάθος χρηστών και να γίνεται βέβαιη η λήψη των πληροφοριών αυτών από τους κατάλληλους,

εξουσιοδοτημένους χρήστες. Ταυτόχρονα, εξασφαλίζεται η ακεραιότητα του συστήματος καθώς δίνεται πρόσβαση σε συγκεκριμένες λειτουργίες της πλατφόρμας μόνο από τους εξουσιοδοτημένους χρήστες. Άμεσο αποτέλεσμα είναι τα δεδομένα να μην μπορούν να μεταβληθούν από τρίτους προσφέροντας έτσι συνέπεια, ακρίβεια και αξιοπιστία των δεδομένων. Επίσης, για την διαθεσιμότητα του συστήματος προσπαθήσαμε να αναπτύξουμε μία πλατφόρμα που να μπορεί να δέχεται και να επεξεργάζεται πολλά αιτήματα σε μικρό χρονικό διάστημα, δημιουργώντας μία υπηρεσία γρήγορη με δυνατότητα κλιμάκωσης. Όσον αφορά, την απώλεια δεδομένων επιλέξαμε τη χρήση κατάλληλης βάσης δεδομένων που προσφέρει ασφάλεια και διατήρηση των δεδομένων ακόμα και μετά από μία απρόσμενη αποτυχία του συστήματος μας.

Εν συνεχεία, έχοντας πραγματοποιήσει το σύστημα ελέγχου ταυτότητας αναλύουμε τις δύο βασικές λειτουργίες που προαναφέρθηκαν, του παρόχου και του πελάτη. Όσον αφορά τις λειτουργίες του παρόχου, η πλατφόρμα χρειάζεται να μπορεί να δέχεται αιτήματα από τον πάροχο. Η συνηθέστερα χρησιμοποιούμενη μορφή είναι αυτή των μηνυμάτων σε μορφή JSON. Αρχικά, λοιπόν, το μήνυμα θα αναλύεται γραμματικά και θα αποφασίζεται ποια λειτουργία της πλατφόρμας θα εκτελεστεί. Ορίζουμε τέσσερις διαφορετικές λειτουργίες της πλατφόρμας για την εξυπηρέτηση των παρόχων και κατ' επέκταση για την λειτουργία της συνομοσπονδίας νεφών. Σε πρώτη φάση, ο πάροχος μέσω κατάλληλου μηνύματος θα διαθέτει στην συνομοσπονδία τους υπολογιστικούς πόρους για αξιοποίηση τους από άλλους χρήστες. Η πλατφόρμα, δεχόμενη ένα αίτημα αυτής της μορφής, θα αποθηκεύει τους πόρους σε μία βάση δεδομένων αντιστοιχίζοντας τους πόρους με τον πάροχο. Φυσικά, ο πάροχος θα πρέπει να μπορεί να μεταβάλλει τους πόρους που διαθέτει. Η πλατφόρμα προσφέρει τη δυνατότητα στον πάροχο να μπορεί να τροποποιήσει τους ήδη υπάρχοντες πόρους, στέλνοντας κατάλληλο αίτημα, το οποίο θα ορίζει μονοσήμαντα τον εκάστοτε πόρο που θέλουμε να τροποποιήσουμε. Τέλος, προσφέρεται η λειτουργία της διαγραφής πόρων όταν ο πάροχος αποφασίσει να μην διαθέτει πλέον συγκεκριμένους πόρους στην συνομοσπονδία. Ο πάροχος θα ορίζει μονοσήμαντα τον πόρο που θέλει να διαγράψει και θα στέλνει κατάλληλο αίτημα, με το οποίο θα μπορεί να διαγράψει έναν πόρο κάθε φορά.

Στην κατηγορία του πελάτη προσφέρεται μία λειτουργία αλλά με μεγαλύτερη πολυπλοκότητα σε σχέση με τις λειτουργίες που προσφέρονται για την ικανοποίηση των αιτημάτων των παρόχων. Η πλατφόρμα πρέπει να μπορεί να δέχεται αιτήματα για την αναζήτηση συγκεκριμένων πόρων που να καλύπτουν τις εκάστοτε ανάγκες του πελάτη. Αρχικά, θα στέλνεται ένα αίτημα από τον πελάτη με τους πόρους που επιθυμεί να αποκτήσει από τη συνομοσπονδία. Η πλατφόρμα θα δέχεται το αίτημα, θα το επεξεργάζεται και θα κάνει

αναζήτηση στη βάση δεδομένων για τον κατάλληλο πάροχο που μπορεί να ικανοποιήσει τις ανάγκες σε υπολογιστικούς πόρους του πελάτη. Τέλος, θα επιστρέφει στον πελάτη ένα μήνυμα με το όνομα του παρόχου δίνοντας του τη δυνατότητα να επικοινωνήσει άμεσα με τον πάροχο για τη σύναψη συμφωνίας και να καταχωρήσει σε μία βάση δεδομένων την συναλλαγή.

Σε αυτό το σημείο χρειάστηκε να προσθέσουμε μία επιπλέον λειτουργία στην πλατφόρμα μας. Λειτουργώντας ως μία συνομοσπονδία νεφών, δηλαδή ως ένα εθελοντικό σύστημα, ερχόμαστε αντιμέτωποι με το πρόβλημα της δίκαιης αντιμετώπισης των χρηστών. Η λειτουργία αυτή θα έχει ως βασικό ρόλο να βαθμολογεί τους παρόχους για τη συνεισφορά τους στη συνομοσπονδία, επιβραβεύοντας αυτούς που συνεισφέρουν στη λειτουργία και χρήση της συνομοσπονδίας. Ταυτόχρονα, θα έχει ως ρόλο να εξαλείψει φαινόμενα εκμετάλλευσης και κατ' επέκταση κατάχρησης, χρησιμοποιώντας μία μορφή βαθμολογίας των χρηστών. Οι χρήστες με χαμηλή βαθμολογία, δηλαδή οι χρήστες που δεν προσφέρουν στη συνομοσπονδία, θα εξυπηρετούνται από αντίστοιχου επιπέδου χρήστες. Άμεσο αποτέλεσμα της παραπάνω λειτουργίας είναι οι «καλοί» χρήστες να ανταμείβονται για την προσφορά τους, λαμβάνοντας υπηρεσίες αντίστοιχου επιπέδου με αυτό που προσφέρουν. Παρακινεί, έτσι, τους πελάτες να λειτουργούν και ως πάροχοι, απορρίπτοντας εμμέσως χρήστες που εκμεταλλεύονται τη συνομοσπονδία για προσωπικό τους όφελος.

3.2.1 Ανάγκη προτυποποίησης αιτημάτων

Στη σύγχρονη εποχή η ανάπτυξη της τεχνολογίας των υπολογιστικών νεφών οδήγησε και στην αύξηση των παρόχων υπηρεσιών νέφους. Κάθε πάροχος, όμως, χρησιμοποιεί διαφορετική ορολογία για την απεικόνιση των πόρων και των υπηρεσιών που διαθέτει. Για την ανάπτυξη και χρήση μίας πλατφόρμας που θα λειτουργεί ως συνομοσπονδία κρίνεται απαραίτητη η υιοθέτηση από όλους τους παρόχους μίας κοινής βάσης επικοινωνίας με αυτή. Εν συνεχεία, οι πόροι που διατίθενται από τους παρόχους θα πρέπει να ακολουθούν μία κοινή μορφή απεικόνισης για την ευκολότερη αποθήκευσή τους και επεξεργασία τους από την πλατφόρμα.

Για την επικοινωνία με την πλατφόρμα επιλέχθηκε η χρήση του πρωτοκόλλου HTTP. Η επικοινωνία, γίνεται με τη χρήση συγκεκριμένων επικεφαλίδων που προσδιορίζουν τη μορφή και τον τύπο των δεδομένων που αναμένονται από την πλατφόρμα μας. Πιο συγκεκριμένα, χρησιμοποιείται επικεφαλίδα που καθορίζει τον τρόπο ταυτοποίησης του αποστολέα και επικεφαλίδα που προσδιορίζει τον τύπο περιεχομένου του μηνύματος. Η πλατφόρμα μας αναπτύχθηκε ως ένας εξυπηρετητής δεδομένων JSON με άμεση συνέπεια να χρησιμοποιείται επικεφαλίδα κατάλληλη για τον προσδιορισμό περιεχομένου αυτής της μορφής.

Επιπροσθέτως, επιλέξαμε την απεικόνιση και αποθήκευση των πόρων στη βάση δεδομένων βασισμένοι στην προτυποποίηση που χρησιμοποιείται από την διαδικτυακή υπηρεσία της Amazon, EC2. Γενικότερα, οι πάροχοι υπηρεσιών υποδομής νέφους διαχωρίζουν τους πόρους τους με βάση την τοποθεσία, το λειτουργικό σύστημα και την βέλτιστη χρήση της εκάστοτε υποδομής. Παρατηρείται, δηλαδή, ένα κοινό μοτίβο μεταξύ των παρόχων το οποίο διαφοροποιείται στην σύμβαση ονομασίας των υποδομών. Για την περάτωση της παρούσας διπλωματικής επιλέξαμε την προτυποποίηση της Amazon, καθώς θεωρείται ένας από τους μεγαλύτερους παρόχους υπηρεσιών νέφους, χωρίς βλάβη της γενικότητας.

Τέλος, για την αναζήτηση των διαθέσιμων πόρων κρίθηκε απαραίτητη η προτυποποίηση μηνυμάτων κατάλληλης μορφής ώστε όλοι οι πάροχοι-πελάτες να χρησιμοποιούν μία κοινή βάση επικοινωνίας. Στο μήνυμα αυτό, θα περιέχονται οι απαιτήσεις του καταναλωτή για πόρους το οποίο θα επεξεργάζεται από την πλατφόρμα για την εξαγωγή αποτελεσμάτων.

3.2.2 Σύστημα βαθμολόγησης χρηστών της πλατφόρμας

Η συμμετοχή στην συνομοσπονδία νεφών, όπως προαναφέρθηκε, είναι εθελοντική. Κάθε χρήστης της πλατφόρμας επιλέγει πότε, πως και πόσο συχνά θα συμμετέχει . Πρέπει, λοιπόν, να αναπτυχθεί ένα σύστημα βαθμολόγησης των χρηστών, κατάλληλο για την εθελοντική φύση της πλατφόρμας. Το σύστημα αυτό θα πρέπει να επιβραβεύει τους χρήστες που συμμετέχουν συχνότερα ως πάροχοι.

Αρχικά, η πλατφόρμα μας είναι υπεύθυνη για τον εντοπισμό στη βάση δεδομένων των παρόχων που μπορούν να ικανοποιήσουν ένα αίτημα για πόρους ενός άλλου χρήστη-πελάτη. Φυσικά, υπάρχει η περίπτωση ένα αίτημα να μπορεί να εξυπηρετηθεί από αρκετούς διαφορετικούς παρόχους. Προκύπτει, λοιπόν, το πρόβλημα της επιλογής του καταλληλότερου παρόχου για την εξυπηρέτηση του αιτήματος.

Κάθε χρήστης με την εγγραφή του στη συνομοσπονδία θα λαμβάνει μια τυποποιημένη βαθμολογία και μια τιμή απόκλισης βαθμολογίας (rating deviation-RD). Η απόκλιση βαθμολογίας (RD) είναι ένα μέτρο αξιολόγησης της εμπιστοσύνης που υπάρχει στη βαθμολογία του κάθε χρήστη. Πιο συγκεκριμένα, χρησιμοποιείται για να δείξει ότι η πραγματική τιμή της βαθμολογίας του χρήστη κυμαίνεται σε ένα συγκεκριμένο εύρος τιμών. Μεγάλη τιμή του RD δείχνει μεγάλη αβεβαιότητα για τις πραγματικές ικανότητες και την απόδοση ενός συγκεκριμένου χρήστη. Η πλατφόρμα θα αξιολογεί τους διαφορετικούς παρόχους που μπορούν να αξιοποιήσουν το αίτημα και θα επιλέγει αυτόν που η βαθμολογία του είναι πιο κοντά στο κατώτατο όριο του εύρους τιμών της βαθμολογίας του χρήστη που πραγματοποίησε το αίτημα

και θα εισάγει την συναλλαγή σε μία βάση δεδομένων. Για να γίνει πιο κατανοητή η παραπάνω λειτουργία, ας υποθέσουμε ότι ο χρήστης που έκανε το αίτημα έχει βαθμολογία 1500 και απόκλιση βαθμολογίας 100. Το σύστημα μας είναι σίγουρο ότι η βαθμολογία του συγκεκριμένου χρήστη κυμαίνεται μεταξύ 1300 και 1700. Θα προσπαθήσει, λοιπόν, να επιλέξει έναν πάροχο με βαθμολογία κοντά στην τιμή 1300. Η λογική του συστήματος είναι να επιβραβεύει όχι μόνο τους παρόχους με καλή απόδοση αλλά και αυτούς με συχνή συμμετοχή στη συνομοσπονδία, κριτήριο πολύ σημαντικό για την καλή λειτουργία του δικτύου νεφών.

Για την ανανέωση της βαθμολογίας των χρηστών επιλέξαμε να χρησιμοποιήσουμε μία παραλλαγή του αλγορίθμου Glicko, από τον οποίο προέκυψε και η παραπάνω ιδέα επιλογής των χρηστών. Ο αλγόριθμος αυτός προτάθηκε από τον Mark Glickman το 1995 και είναι μία βελτίωση του συστήματος ELO που χρησιμοποιείται για να αποδώσει την ικανότητα και να βαθμολογήσει τους παίχτες σε ανταγωνιστικά παιχνίδια, π.χ. το σκάκι. Η παραλλαγή του αλγορίθμου οφείλεται στο γεγονός ότι το σύστημα αυτό αναπτύχθηκε για ανταγωνιστικά παιχνίδια, ενώ εμείς εφαρμόζουμε τον αλγόριθμο αυτό σε ένα εθελοντικό σύστημα όπου οι χρήστες αλληλοεπιδρούν για το δικό τους κοινό συμφέρον.

Ο αλγόριθμος αυτός θα τρέχει ανεξάρτητα από την υπόλοιπη πλατφόρμα για συγκεκριμένες χρονικές περιόδους, που ονομάζονται περίοδοι βαθμολόγησης. Θα χρησιμοποιεί σε αυτές τις περιόδους τις καταχωρήσεις των συναλλαγών της βάσης δεδομένων και βάση αυτών θα υπολογίζει τις νέες βαθμολογίες και τις νέες αποκλίσεις βαθμολογίας. Τέλος, θα ενημερώνει την πλατφόρμα για τις αλλαγές αυτές.

3.2.3 Επιλογή MongoDB για βάση δεδομένων

Το πιο διαδεδομένο και μεγαλύτερο NoSQL σύστημα διαχείρισης δεδομένων-εγγράφων είναι το MongoDB. Προσφέρει διάφορα τεχνικά χαρακτηριστικά τα οποία χρησιμοποιούνται για να υπερνικήσουν τις κλασσικές σχεσιακές βάσεις δεδομένων και τους περιορισμούς άλλων τύπων NoSQL βάσεων. Οι δυνατότητες του MongoDB το κάνουν κατάλληλο για εφαρμογές με αδόμητα, ημι-δομημένα και πολυμορφικά δεδομένα και εφαρμογές με ανάγκη για μεγάλη κλιμακωσιμότητα.

Κάποιες από τις δυνατότητες του MongoDB οι οποίες μας οδήγησαν να το επιλέξουμε ως την κύρια αποθήκη δεδομένων για το σύστημα μας είναι:

- Χρησιμοποιεί τον τύπο BSON που είναι ένας τύπος αποθήκευσης βασισμένος στο JSON. Είναι μία δυαδική σειριακή κωδικοποίηση εγγράφων μορφής JSON που η MongoDB χρησιμοποιεί για να αποθηκεύσει έγγραφα σε συλλογές. Προσφέρει τη δυνατότητα

τύπον δεδομένων όπως ημερομηνίες και δυαδικά δεδομένα που δεν υποστηρίζονται από το JSON. Η μορφή αυτή χρησιμοποιεί το πεδίο _id ως πρωτεύον κλειδί, η τιμή του οποίου θα είναι συνήθως ένας τύπος μοναδικού αναγνωριστικού το οποίου δημιουργείται συνήθως από την ίδια την υπηρεσία. Το πιο σημαντικό πλεονέκτημα της μορφής BSON είναι ότι επιτρέπει στην MongoDB να ταξινομεί εσωτερικά και να απεικονίζει ιδιότητες εγγράφων ακόμα και φωλιασμένα (nested) έγγραφα. Δίνει τη δυνατότητα να είναι αποδοτική τόσο σε μεγάλα μεγέθη όσο και σε ταχύτητα, προσφέροντας μεγάλη ικανότητα διαβίβασης δεδομένων (throughput).

- Προσφέρει πλαίσιο συνάθροισης (aggregation framework). Για επεξεργασία δεσμίδων δεδομένων και για λειτουργίες συνάθροισης μπορεί να χρησιμοποιηθεί MapReduce. Πρόκειται για μια συναφή υλοποίηση για επεξεργασία και δημιουργία μεγάλων συνόλων δεδομένων με παράλληλο, κατανεμημένο αλγόριθμο σε έναν πυρήνα. Το προγραμματιστικό μοντέλο MapReduce αποτελείται από δύο βασικές διαδικασίες. Η Map διαδικασία εκτελεί το φιλτράρισμα και την ταξινόμηση ενώ η διαδικασία Reduce εκτελεί τη διαδικασία σύνοψης. Επιτρέπει έτσι την γρήγορη επεξεργασία τεράστιων συνόλων δεδομένων.
- Ευρετήρια τα οποία δημιουργούνται για βελτίωση της απόδοσης αναζητήσεων. Κάθε έγγραφο MongoDB μπορεί να δεικτοδοτηθεί με πρωτεύοντες και δευτερεύοντες δείκτες το οποίο επιτρέπει στη μηχανή της βάσης δεδομένων να επιλύει ερωτήματα. Η μηχανή μπορεί να χρησιμοποιήσει προκαθορισμένο ευρετήριο, που απεικονίζει τα πεδία των εγγράφων και μπορεί να ξεχωρίσει ποια έγγραφα είναι συμβατά με την εντολή αναζήτησης, βελτιώνοντας έτσι την επίδοση του συστήματος.

3.2.4 Συνοπτικά

Οι λειτουργίες λοιπόν που θα πρέπει να ενσωματώνει η πλατφόρμα είναι:

- Επεξεργασία αιτημάτων εγγραφής χρήστη
- Αποθήκευση ταυτότητας χρήστη
- Δυνατότητα ταυτοποίησης χρηστών
- Επεξεργασία αιτημάτων καταχώρησης και τροποποίησης πόρων
- Αποθήκευση πόρων σε κατάλληλη μορφή και δυνατότητα τροποποίησης τους
- Επεξεργασία αιτημάτων για αναζήτηση διαθέσιμων πόρων
- Αναζήτηση κατάλληλου παρόχου για ικανοποίηση αιτήματος χρήστη
- Αποστολή στο χρήστη το αποτέλεσμα της αναζήτησης
- Αποθήκευση των συναλλαγών που πραγματοποιούνται

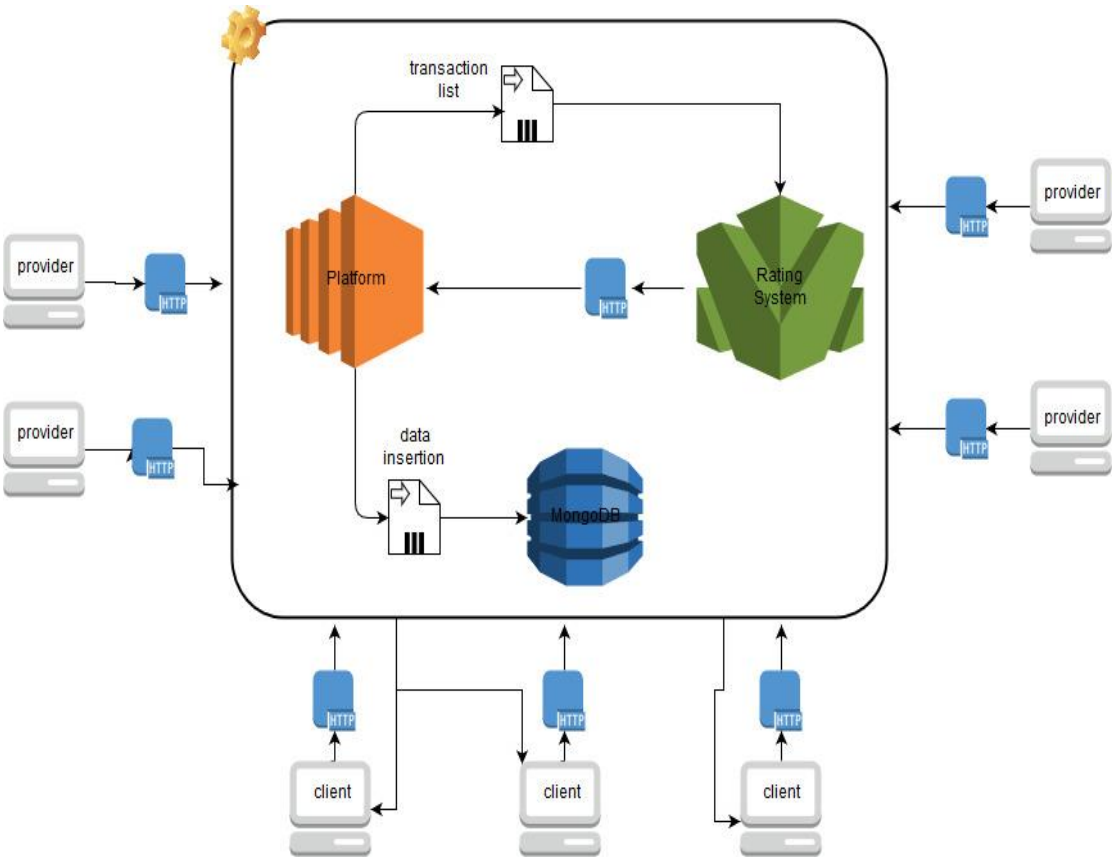
- Δυνατότητα επικοινωνίας με την οντότητα που υλοποιεί το σύστημα βαθμολόγησης

Παράλληλα αναπτύχθηκε και μία οντότητα για το σύστημα βαθμολόγησης με τις παρακάτω λειτουργίες:

- Επικοινωνία με την πλατφόρμα
- Επεξεργασία δεδομένων συναλλαγών και υπολογισμός βαθμολογίας χρηστών
- Ενημέρωση πλατφόρμας για ανανέωση των βαθμολογιών των χρηστών
- Ενημέρωση πλατφόρμας για ανανέωση απόκλισης βαθμολογίας των χρηστών

3.3 Σενάρια χρήσης

Στο παρακάτω σχήμα παρουσιάζεται η βασική αρχιτεκτονική της συνομοσπονδίας νεφών που θα αναπτύξουμε για τις ανάγκες της παρούσας διπλωματικής.



Για την εξαγωγή των απαιτήσεων της πλατφόρμας αναλύθηκαν τα κυριότερα σενάρια χρήσης τα οποία παρουσιάζονται στις παρακάτω υπο-ενότητες.

3.3.1 Εγγραφή χρήστη

Το σενάριο αυτό προδιαγράφει τη διαδικασία που ακολουθείται για να εγγραφεί ένας νέος χρήστης στην πλατφόρμα.

3.3.1.1 Βασική ροή γεγονότων

Χρήστης	Πλατφόρμα
1. Δημιουργία αιτήματος κατάλληλης μορφής με περιεχόμενο: I. Όνομα χρήστη II. Κωδικός πρόσβασης III. URL επικοινωνίας 2. Αποστολή αιτήματος στο κατάλληλο URL	
	3. Γραμματική ανάλυση αιτήματος (parse) 4. Έλεγχος συμπληρωθέντων στοιχείων 5. Επικοινωνία πλατφόρμας με τη βάση δεδομένων 6. Έλεγχος μοναδικότητας στοιχείων I. Όνομα χρήστη 7. Μετατροπή με κατακερματισμό του κωδικού πρόσβασης 8. Δημιουργία εγγραφής χρήστη στη βάση δεδομένων 9. Δημιουργία προσωρινού αδειοδοτικού χρήσης (token) 10. Επιστροφή μηνύματος επιτυχούς εγγραφής και αδειοδοτικού χρήσης
11. Αποθήκευση προσωρινού αδειοδοτικού χρήσης	

3.3.1.2 Ροή διαχείρισης σφάλματος – Ελλιπής συμπλήρωση περιεχομένου αιτήματος

Χρήστης	Πλατφόρμα
	4. Ελλιπές περιεχόμενο αιτήματος 5. Επιστροφή μηνύματος σφάλματος

3.3.1.3 Ροή διαχείρισης σφάλματος – Μη μοναδικό όνομα χρήστη

Χρήστης	Πλατφόρμα
	6. Μη μοναδικό όνομα χρήστη 7. Επιστροφή μηνύματος σφάλματος

3.3.2 Λειτουργία εισαγωγής πόρων παρόχου

Το σενάριο αυτό περιγράφει τη διαδικασία που ακολουθείται για να εισάγει ένας πάροχος τους διαθέσιμους πόρους στη συνομοσπονδία.

3.3.2.1 Βασική ροή γεγονότων

Χρήστης	Πλατφόρμα
1. Δημιουργία επικεφαλίδας με προσθήκη αδειοδοτικού χρήσης 2. Δημιουργία αιτήματος με συμπλήρωση περιεχομένου: I. Όνομα χρήστη II. Τοποθεσία III. Λειτουργικό Σύστημα IV. Τύπος Βελτιστοποίησης V. Οικογένεια Προϊόντος VI. Κατηγορία εικονικής μηχανής VII. Αριθμός εικονικών μονάδων επεξεργασίας VIII. Μέγεθος μνήμης IX. Τύπους μονάδας αποθήκευσης X. Τιμή μηχανής	

XI. Διαθεσιμότητα 3. Αποστολή αιτήματος στο κατάλληλο URL	
	4. Έλεγχος ταυτότητας χρήστη 5. Γραμματική ανάλυση αιτήματος (parse) 6. Έλεγχος συμπληρωθέντων στοιχείων 7. Επιβεβαίωση εγκυρότητας αντιστοίχισης καταχώρησης και ταυτότητας χρήστη 8. Επικοινωνία με τη βάση δεδομένων 9. Έλεγχος μοναδικότητας καταχώρησης 10. Δημιουργία νέας καταχώρησης παρόχου στη βάση δεδομένων 11. Επιστροφή μηνύματος επιτυχούς καταχώρησης

3.3.2.2 Ροή διαχείρισης σφάλματος – Άκυρη αδειοδότηση χρήσης

Χρήστης	Πλατφόρμα
	4. Μη έγκυρη αδειοδότηση χρήσης 5. Επιστροφή μηνύματος μη εγκεκριμένης πράξης

3.3.2.3 Ροή διαχείρισης σφάλματος – Ελλιπής συμπλήρωση περιεχομένου αιτήματος

Χρήστης	Πλατφόρμα
	6. Ελλιπές περιεχόμενο αιτήματος 7. Επιστροφή μηνύματος σφάλματος

3.3.2.4 Ροή διαχείρισης σφάλματος – Απαγορευμένη καταχώρηση δεδομένων

Χρήστης	Πλατφόρμα
	7. Αποτυχία αντιστοίχισης ταυτότητας χρήστη και καταχώρησης 8. Επιστροφή μηνύματος απαγορευμένης πράξης

3.3.2.5 Ροή διαχείρισης σφάλματος – Μη μοναδική καταχώρηση

Χρήστης	Πλατφόρμα
	9. Μη μοναδική καταχώρηση στη βάση δεδομένων 10. Επιστροφή μηνύματος μη επιτυχούς καταχώρησης

3.3.3 Λειτουργία διόρθωσης στοιχείων πόρων παρόχου

Το σενάριο αυτό περιγράφει τη διαδικασία που ακολουθείται για να διορθώσει ένας πάροχος τους διαθέσιμους πόρους στη συνομοσπονδία. Για να το πετύχει αυτό ο πάροχος πρέπει να ορίσει μονοσήμαντα τον πόρο και δίνεται δυνατότητα διόρθωσης μόνο της τιμής και της διαθεσιμότητας μίας μονάδας.

3.3.3.1 Βασική ροή γεγονότων

Χρήστης	Πλατφόρμα
1. Δημιουργία επικεφαλίδας με προσθήκη αδειοδοτικού χρήσης 2. Δημιουργία αιτήματος με συμπλήρωση περιεχομένου: I. Όνομα χρήστη II. Τοποθεσία III. Λειτουργικό Σύστημα IV. Τύπος Βελτιστοποίησης V. Οικογένεια Προϊόντος	

VI. Κατηγορία εικονικής μηχανής VII. Διαθεσιμότητα ή/και τιμή μηχανής 3. Αποστολή αιτήματος στο κατάλληλο URL	
	4. Έλεγχος ταυτότητας χρήστη 5. Γραμματική ανάλυση αιτήματος (parse) 6. Έλεγχος συμπληρωθέντων στοιχείων 7. Επιβεβαίωση εγκυρότητας αντιστοίχισης καταχώρησης και ταυτότητας χρήστη 8. Επικοινωνία με τη βάση δεδομένων 9. Έλεγχος ύπαρξης καταχώρησης 10. Τροποποίηση της υπάρχουσας καταχώρησης παρόχου στη βάση δεδομένων 11. Επιστροφή μηνύματος επιτυχούς διόρθωσης

3.3.3.2 Ροή διαχείρισης σφάλματος – Άκυρη αδειοδότηση χρήσης

Χρήστης	Πλατφόρμα
	4. Μη έγκυρη αδειοδότηση χρήσης 5. Επιστροφή μηνύματος μη εγκεκριμένης πράξης

3.3.3.3 Ροή διαχείρισης σφάλματος – Ελλιπής συμπλήρωση περιεχομένου αιτήματος

Χρήστης	Πλατφόρμα
	6. Ελλιπές περιεχόμενο αιτήματος για προσδιορισμό στοιχείου 7. Επιστροφή μηνύματος σφάλματος

3.3.3.4 Ροή διαχείρισης σφάλματος – Απαγορευμένη διόρθωση δεδομένων καταχώρησης

Χρήστης	Πλατφόρμα
	7. Αποτυχία αντιστοίχισης ταυτότητας χρήστη και καταχώρησης 8. Επιστροφή μηνύματος απαγορευμένης πράξης

3.3.3.5 Ροή διαχείρισης σφάλματος – Μη ύπαρξη καταχώρησης

Χρήστης	Πλατφόρμα
	9. Μη ύπαρξη καταχώρησης στη βάση δεδομένων 10. Επιστροφή μηνύματος αποτυχίας διόρθωσης

3.3.4 Λειτουργία διαγραφής στοιχείων πόρων παρόχου

Το σενάριο αυτό περιγράφει τη διαδικασία που ακολουθείται για να διαγράψει ένας πάροχος τους διαθέσιμους πόρους στη συνομοσπονδία. Για να το πετύχει αυτό ο πάροχος πρέπει να ορίσει μονοσήμαντα τον πόρο και δίνεται δυνατότητα διαγραφής μόνο ενός πόρου ανά αίτημα.

3.3.4.1 Βασική ροή γεγονότων

Χρήστης	Πλατφόρμα
1. Δημιουργία επικεφαλίδας με προσθήκη αδειοδοτικού χρήσης 2. Δημιουργία αιτήματος με συμπλήρωση περιεχομένου: I. Όνομα χρήστη II. Τοποθεσία III. Λειτουργικό Σύστημα IV. Τύπος Βελτιστοποίησης	

V. Οικογένεια Προϊόντος VI. Κατηγορία εικονικής μηχανής 3. Αποστολή αιτήματος στο κατάλληλο URL	
	4. Έλεγχος ταυτότητας χρήστη 5. Γραμματική ανάλυση αιτήματος (parse) 6. Έλεγχος συμπληρωθέντων στοιχείων 7. Επιβεβαίωση εγκυρότητας αντιστοίχισης καταχώρησης και ταυτότητας χρήστη 8. Επικοινωνία με τη βάση δεδομένων 9. Έλεγχος ύπαρξης καταχώρησης 10. Διαγραφή υπάρχουσας καταχώρησης παρόχου στη βάση δεδομένων 11. Επιστροφή μηνύματος επιτυχούς διόρθωσης

3.3.4.2 Ροή διαχείρισης σφάλματος – Άκυρη αδειοδότηση χρήσης

Χρήστης	Πλατφόρμα
	4. Μη έγκυρη αδειοδότηση χρήσης 5. Επιστροφή μηνύματος μη εγκεκριμένης πράξης

3.3.4.3 Ροή διαχείρισης σφάλματος – Ελλιπής συμπλήρωση περιεχομένου αιτήματος

Χρήστης	Πλατφόρμα
	6. Ελλιπές περιεχόμενο αιτήματος για προσδιορισμό στοιχείου 7. Επιστροφή μηνύματος σφάλματος

3.3.4.4 Ροή διαχείρισης σφάλματος – Απαγορευμένη διόρθωση δεδομένων καταχώρησης

Χρήστης	Πλατφόρμα
	7. Αποτυχία αντιστοίχισης ταυτότητας χρήστη και καταχώρησης 8. Επιστροφή μηνύματος απαγορευμένης πράξης

3.3.4.5 Ροή διαχείρισης σφάλματος – Μη ύπαρξη καταχώρησης

Χρήστης	Πλατφόρμα
	9. Μη ύπαρξη καταχώρησης στη βάση δεδομένων 10. Επιστροφή μηνύματος αποτυχίας διαγραφής

3.3.5 Λειτουργία εξυπηρέτησης αιτήματος πελάτη για αναζήτηση πόρων

Το σενάριο αυτό περιγράφει τη διαδικασία που ακολουθείται για την εξυπηρέτηση αιτήματος αναζήτησης για διαθέσιμους πόρους στη συνομοσπονδία από έναν χρήστη με την ιδιότητα του πελάτη. Για να το πετύχει αυτό ο πελάτης πρέπει να ορίσει τα χαρακτηριστικά της μηχανής που ζητάει από τη συνομοσπονδία. Ορίζονται δύο τύποι χαρακτηριστικών, τα υποχρεωτικά χαρακτηριστικά στα οποία θα βασιστεί η αναζήτηση και τα προαιρετικά χαρακτηριστικά τα οποία κρίνει ο πελάτης ότι δεν είναι κρίσιμα για την επιτυχία της αναζήτησης.

3.3.5.1 Βασική ροή γεγονότων

Χρήστης	Πλατφόρμα
1. Δημιουργία επικεφαλίδας με προσθήκη αδειοδοτικού χρήσης	

2. Δημιουργία αιτήματος με συμπλήρωση χαρακτηριστικών αναζήτησης: I. Υποχρεωτικά (mandatory) χαρακτηριστικά II. Προαιρετικά (optional) χαρακτηριστικά 3. Αποστολή αιτήματος στο κατάλληλο URL	
	4. Έλεγχος ταυτότητας χρήστη 5. Γραμματική ανάλυση αιτήματος (parse) 6. Έλεγχος συμπληρωθέντων στοιχείων 7. Επικοινωνία με τη βάση δεδομένων 8. Αρχικό φιλτράρισμα παρόχων με βάση τη βαθμολογία τους 9. Αναζήτηση παρόχων που ικανοποιούν τα δοθέντα χαρακτηριστικά 10. Τελικό φιλτράρισμα παρόχων και επιλογή ενός μοναδικού παρόχου 11. Καταχώρηση στη βάση δεδομένων της συναλλαγής με στοιχεία: I. Όνομα πελάτη II. Όνομα παρόχου III. Αποτέλεσμα συναλλαγής 12. Δημιουργία απάντησης με: I. Όνομα παρόχου II. URL επικοινωνίας με πάροχο 13. Αποστολή απάντησης στον χρήστη
14. Επικοινωνία με πάροχο για σύναψη συμφωνίας	

3.3.5.2 Εναλλακτική ροή γεγονότων

Στο σενάριο αυτό περιγράφεται η περίπτωση κατά την οποία κανένας από τους παρόχους που έχουν εγγραφεί στο σύστημα δεν μπορεί να ικανοποιήσει τα δοθέντα χαρακτηριστικά.

Χρήστης	Πλατφόρμα
	9. Αδυναμία εύρεσης παρόχων με τα δοθέντα χαρακτηριστικά 10. Επιστροφή μηνύματος μη αντιστοίχισης αποτελεσμάτων

3.3.5.3 Ροή διαχείρισης σφάλματος – Άκυρη αδειοδότηση χρήσης

Χρήστης	Πλατφόρμα
	11. Μη έγκυρη αδειοδότηση χρήσης 12. Επιστροφή μηνύματος μη εγκεκριμένης πράξης

3.3.5.4 Ροή διαχείρισης σφάλματος – Ελλιπής συμπλήρωση περιεχομένου αιτήματος

Χρήστης	Πλατφόρμα
	8. Ελλιπές περιεχόμενο αιτήματος για υποχρεωτικό πεδίο 9. Επιστροφή μηνύματος σφάλματος

3.3.6 Λειτουργία επικοινωνίας πλατφόρμας και συστήματος βαθμολόγησης

Παρόλο που το συγκεκριμένο σενάριο δεν αφορά τις δυνατότητες που έχει ο χρήστης για την αξιοποίηση της πλατφόρμας η αναγκαιότητα ύπαρξης του συστήματος βαθμολόγησης για τη σωστή λειτουργία της συνομοσπονδίας, μας παροτρύνει να παρουσιάσουμε το συγκεκριμένο σενάριο. Παρακάτω, λοιπόν, παρουσιάζεται ο τρόπος με τον οποίο το σύστημα αυτό επικοινωνεί με την πλατφόρμα μας και οι λειτουργίες τις οποίες εκτελεί ώστε οι χρήστες να αντιμετωπίζονται δίκαια.

Σύστημα Βαθμολόγησης	Πλατφόρμα
1. Εκκίνηση περιόδου βαθμολόγησης 2. Δημιουργία επικεφαλίδας με προσθήκη αδειοδοτικού χρήσης 3. Δημιουργία μηνύματος κατάλληλης μορφής με περιεχόμενο: I. Σταθερά c 4. Αποστολή αιτήματος στο κατάλληλο URL	
	6. Έλεγχος ταυτότητας 7. Γραμματική ανάλυση αιτήματος (parse) 8. Επικοινωνία με τη βάση δεδομένων 9. Ανανέωση βαθμολογίας απόκλισης για όλους τους εγγεγραμμένους χρήστες 10. Αποστολή απάντησης
11. Δημιουργία επικεφαλίδας με προσθήκη αδειοδοτικού χρήσης 12. Αποστολή αιτήματος λήψης λίστας συναλλαγών στο κατάλληλο URL	
	12. Έλεγχος ταυτότητας 13. Επικοινωνία με τη βάση δεδομένων 14. Ανάκτηση λίστας συναλλαγών 15. Δημιουργία καινούργιας λίστας συναλλαγών με περιεχόμενο κάθε συναλλαγής: I. Όνομα παρόχου II. Όνομα πελάτη III. Βαθμολογία παρόχου IV. Βαθμολογία πελάτη

	V. Απόκλιση βαθμολογίας παρόχου VI. Απόκλιση βαθμολογίας πελάτη VII. Αποτέλεσμα συναλλαγής 16. Αποστολή λίστας
17. Γραμματική ανάλυση λίστας συναλλαγών 18. Υπολογισμός νέας βαθμολογίας για κάθε χρήστη στη λίστα 19. Υπολογισμός νέας απόκλισης βαθμολογίας για κάθε χρήστη στη λίστα 20. Δημιουργία λίστας με περιεχόμενο: I. Όνομα χρήστη II. Νέα βαθμολογία III. Νέα απόκλιση βαθμολογίας 21. Δημιουργία επικεφαλίδας με προσθήκη αδειοδοτικού χρήσης 22. Αποστολή κάθε στοιχείου της λίστας στο κατάλληλο URL	
	23. Έλεγχος ταυτότητας 24. Γραμματική ανάλυση λίστας 25. Επικοινωνία με τη βάση δεδομένων 26. Ανανέωση βαθμολογίας χρηστών που περιέχονται στη λίστα 27. Ανανέωση βαθμολογίας απόκλισης χρηστών που περιέχονται στη λίστα 28. Αποστολή μηνύματος επιτυχούς ανανέωσης
29. Αναμονή για εκκίνηση νέας βαθμολογικής περιόδου	

4

Αρχιτεκτονική του συστήματος και σχεδίαση

Η φιλοσοφία σχεδίασης του συστήματος που αναπτύξαμε βασίζεται στις αρχές της αρχιτεκτονικής συστημάτων λογισμικού και πραγματοποιήθηκε με γνώμονα τις ανάγκες των χρηστών αλλά και την αποδοτικότητα, ευελιξία και ασφάλεια του συστήματος. Σε αυτό το κεφάλαιο θα περιγράψουμε τη διαδικασία ορισμού της αρχιτεκτονικής, των υποσυστημάτων, των μοντέλων, των διεπαφών και των δεδομένων με σκοπό την ικανοποίηση των απαιτήσεων του συστήματος.

Προτού συνεχίσουμε την περαιτέρω περιγραφή της σχεδίασης, είναι σημαντικό να αναφερθεί ότι το σύστημα μας δεν δημιουργείται «εν κενώ», δηλαδή επηρεάζεται από το περιβάλλον στο οποίο δημιουργείται. Οφείλει λοιπόν να ανταπεξέρχεται, όχι μόνο στις αρχικές απαιτήσεις των σχεδιαστών, αλλά και στις ανάγκες των χρηστών. Επίσης, θα πρέπει να λαμβάνεται πρόβλεψη για την επίλυση προβλημάτων που μπορούν να προκύψουν στο μέλλον, επιπλέον των τωρινών. Η συνολική διαδικασία από την αρχική σχεδίαση ως την τελική υλοποίηση περιλαμβάνει την λεπτομερή εξέταση όλων των σχετικών παραγόντων και τον συνυπολογισμό όλων των απαιτούμενων προδιαγραφών για την δημιουργία ενός συστήματος χρήσιμου βασισμένο σε τεχνική και θεωρητική ανάλυση.

Η σχεδίαση αποτελεί ένα σημαντικό στάδιο της ανάπτυξης συστήματος και ξεκινάει αφότου η φάση της ανάλυσης έχει ολοκληρωθεί. Είναι σημαντικό να αναφέρουμε ότι το αποτέλεσμα που προκύπτει από την διαδικασία της ανάλυσης και οι προδιαγραφές που προκύπτουν από αυτή, χρησιμοποιούνται ως «είσοδος» στο στάδιο της σχεδίασης, το οποίο κατόπιν οδηγεί στην αποτελεσματική υλοποίηση του συστήματος.

4.1 Περιγραφή συνολικής λειτουργίας συστήματος και σχηματική δομή

Η φιλοσοφία σχεδίασης που ακολουθήσαμε είναι αυτή της αρχιτεκτονικής λογισμικού βασισμένου σε υποσυστήματα (component based software architecture). Σε αυτή τη φιλοσοφία επικεντρωνόμαστε στο διαχωρισμό της σχεδίασης σε ξεχωριστά λειτουργικά και υποσυστήματα, τα οποία ορίζουν συγκεκριμένα πρότυπα επικοινωνίας και αλληλεπίδρασης μεταξύ τους. Προσφέρει έτσι ένα υψηλότερο επίπεδο αφαίρεσης και χωρίζει το πρόβλημα σε υπο-προβλήματα, το κάθε ένα από τα οποία αντιστοιχίζεται με ένα από τα υποσυστήματα.

Η επικοινωνία και η αλληλεπίδραση του χρήστη με την πλατφόρμα πραγματοποιείται μέσω αιτημάτων HTTP. Παρεμβάλλεται, όμως, ένα ενδιάμεσο στάδιο και πιο συγκεκριμένα ο Flask Web Server, ο οποίος δέχεται τα αιτήματα που πρέπει να ακολουθούν τις προδιαγραφές ενός RESTful API το οποίο έχουμε υλοποιήσει. Τα αιτήματα διαμοιράζονται στο κατάλληλο υποσύστημα με βάση το URL. Εν συνεχεία, απαντώνται από τον Server αφού πρώτα έχει πραγματοποιηθεί επεξεργασία τους από το υποσύστημα που είναι υπεύθυνο για την ανάλυση και εξαγωγή αποτελεσμάτων για το εκάστοτε αίτημα.

Η πλατφόρμα που σχεδιάζουμε προκειμένου να καλυφθούν οι διάφορες λειτουργίες που περιεγράφηκαν στο στάδιο της ανάλυσης θα χωριστεί σε τέσσερα βασικά υποσυστήματα, πέραν του Flask Web Server,τα οποία διαθέτουν ξεχωριστή λειτουργικότητα και μπορούν να θεωρηθούν σχετικώς ανεξάρτητα, στοχεύοντας το καθένα στην επίλυση ενός τύπου προβλήματος.

Αρχικά, ορίζεται το υποσύστημα που πραγματοποιεί την εγγραφή των χρηστών στη συνομοσπονδία νεφών (root). Σε αυτό περιλαμβάνεται ένα από τα βασικότερα κομμάτια της ανάπτυξης της πλατφόρμας μας, το οποίο είναι η ταυτοποίηση των χρηστών με τη δημιουργία αδειοδοτικού χρήσης για κάθε χρήστη. Επίσης, επικοινωνεί με μία συγκεκριμένη συλλογή της βάσης δεδομένων (Providers) για την αποθήκευση των βασικών χαρακτηριστικών των χρηστών, τα οποία προσδιορίζουν επαρκώς τους συμμετέχοντες στην συνομοσπονδία και χαρακτηρίζουν τη συμπεριφορά τους. Η επικοινωνία και η αλληλεπίδραση μεταξύ συστήματος και βάσης δεδομένων γίνεται με αιτήματα του πρωτοκόλλου HTTP.

Στη συνέχεια, ορίζεται το υποσύστημα που είναι υπεύθυνο για την διαχείριση των δεδομένων των χρηστών (Exchange). Βασική λειτουργία του συγκεκριμένου υποσυστήματος είναι η

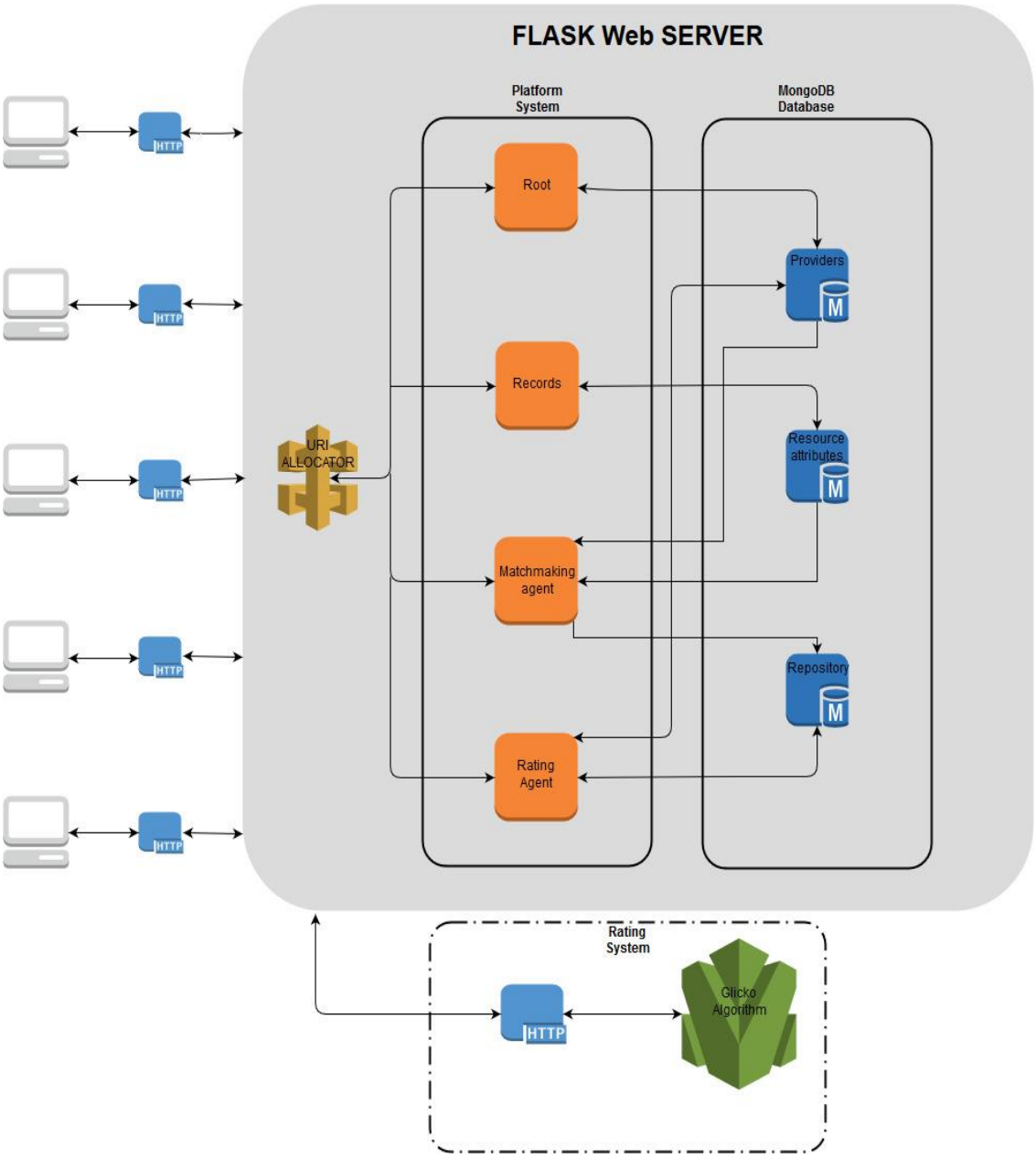
επικοινωνία με το υποσύστημα root για έλεγχο της ταυτότητας του χρήστη. Επιπροσθέτως, είναι υπεύθυνο για την επικοινωνία με μία συγκεκριμένη συλλογή της βάσης δεδομένων (Resource Attributes) για την εισαγωγή των διαθέσιμων πόρων των χρηστών σε αυτή, καθώς και τη διόρθωση και διαγραφή πόρων. Η επιθυμητή λειτουργία καθορίζεται από τη μορφή του αιτήματος που δέχεται η πλατφόρμα. Τα αιτήματα, όπως και προηγουμένως, ακολουθούν το πρωτόκολλο HTTP για την επικοινωνία συστήματος-χρήστη και συστήματος-βάσης δεδομένων.

Για την εξυπηρέτηση των χρηστών και την κάλυψη των αναγκών τους με διαθέσιμους πόρους της συνομοσπονδίας αναπτύσσεται ένα ακόμα υποσύστημα. Το συγκεκριμένο υποσύστημα θα δέχεται τα αιτήματα των χρηστών με τις εκάστοτε ανάγκες τους για πόρους, επικοινωνώντας αρχικά με το υποσύστημα root για τον έλεγχο της ταυτότητας του χρήστη που έστειλε το αίτημα. Βασική του λειτουργία είναι η αναζήτηση στη βάση δεδομένων για τους παρόχους που μπορούν να καλύψουν τις συγκεκριμένες ανάγκες. Για την επίτευξη του παραπάνω στόχου θα επικοινωνεί με τη συλλογή που είναι αποθηκευμένα τα δεδομένα και θα επιλέγει ένα σύνολο παρόχων που είναι ικανοί για την εξυπηρέτηση των αναγκών του πελάτη. Στη συνέχεια, θα κάνει μία τελική επιλογή ενός και μόνο παρόχου η οποία θα βασίζεται στη συμπεριφορά του παρόχου στη συνομοσπονδία. Θα επικοινωνεί λοιπόν με τη συλλογή που είναι αποθηκευμένα τα χαρακτηριστικά των χρηστών επιλέγοντας αυτόν που κυμαίνεται στο ίδιο εύρος αξιοπιστίας με τον χρήστη που έκανε το αίτημα. Στη συνέχεια, θα αποθηκεύει τα δεδομένα της συναλλαγής στη συλλογή της βάσης δεδομένων με όνομα Repository. Η επικοινωνία συστήματος-χρήστη και συστήματος-βάσης δεδομένων ακολουθεί το πρωτόκολλο HTTP.

Τέλος, αναπτύσσεται το υποσύστημα που είναι υπεύθυνο για την επικοινωνία με το σύστημα βαθμολόγησης. Το σύστημα αυτό δεν είναι ορατό εξωτερικά από τους χρήστες και απαγορεύεται η οποιαδήποτε πρόσβαση τους σε αυτό. Με αυτό τον τρόπο διασφαλίζεται ότι η συμπεριφορά των χρηστών παραμένει ανεπηρέαστη από τους ενδογενείς μηχανισμούς βαθμολόγησης και αποτρέπεται ο αθέμιτος ανταγωνισμός και η γενικότερη κακόβουλη χρήση της πλατφόρμας. Επομένως, λοιπόν, η αρχική λειτουργία του είναι η ο έλεγχος της ταυτότητας του βαθμολογικού συστήματος απορρίπτοντας τα αιτήματα από άλλους χρήστες. Στη συνέχεια, δεχόμενο αιτήματα, που ακολουθούν το πρωτόκολλο HTTP, από το βαθμολογικό σύστημα έχει τη δυνατότητα να επιστρέφει σε αυτό τη λίστα με τις συναλλαγές που έχουν πραγματοποιηθεί για περαιτέρω επεξεργασία. Για την αποστολή της λίστας επικοινωνεί τόσο με τη συλλογή της βάσης δεδομένων Repository όσο και με την συλλογή Providers μέσω αιτημάτων HTTP. Επίσης, το υποσύστημα αυτό είναι υπεύθυνο για την ανανέωση των βαθμολογιών των χρηστών που αναφέρθηκαν στο στάδιο της ανάλυσης για την δίκαιη αντιμετώπιση των χρηστών.

Ανεξάρτητα της πλατφόρμας έχει υλοποιηθεί και το σύστημα βαθμολόγησης, το οποίο επικοινωνεί με την πλατφόρμα με αιτήματα HTTP. Το υποσύστημα που έχει αναπτυχθεί είναι υπεύθυνο για την αποστολή της λίστας συναλλαγών και των χαρακτηριστικών των χρηστών. Σκοπός του είναι η επεξεργασία της λίστας για τον υπολογισμό των νέων βαθμολογιών των χρηστών και στη συνέχεια η ενημέρωση της πλατφόρμας με τα νέα αυτά δεδομένα για αποθήκευση τους στην κατάλληλη συλλογή της βάση δεδομένων.

Στο παρακάτω σχήμα παρουσιάζεται σχηματικά η περιγραφείσα αρχιτεκτονική του συστήματος:



4.2 Η βάση δεδομένων MongoDB

Για την εξυπηρέτηση των αναγκών της πλατφόρμας μας, χρειάζεται η χρήση μίας βάσης δεδομένων. Όπως παρουσιάστηκε και παραπάνω στην αρχιτεκτονική του συστήματος μας, η βάση μας θα αποτελείται από τρεις ανεξάρτητες οντότητες-συλλογές:

- Συλλογή providers
- Συλλογή Resource_attributes
- Συλλογή Transaction_list

Κάθε μία από αυτές τις οντότητες έχει τα δικά της χαρακτηριστικά, τα οποία έχουν προκύψει από τις ανάγκες της παρούσας διπλωματικής. Στη συνέχεια παρουσιάζεται η δομή της κάθε συλλογής ξεχωριστά.

4.2.1 Η δομή της συλλογής providers

Η συλλογή *providers* περιέχει δεδομένα που ταυτοποιούν τον κάθε πάροχο και χαρακτηρίζουν τη συμπεριφορά του στη συνομοσπονδία. Αποτελείται από έγγραφα που έχουν μοναδικό δείκτη το πεδίο του όνομα χρήστη. Τα πεδία που περιλαμβάνει ένα έγγραφο είναι:

- *_id*: Το όνομα χρήστη του παρόχου που συμμετέχει στις υπηρεσίες της πλατφόρμας.
- *Active*: Το πεδίο αυτό παίρνει τιμή Αληθές (True) όταν ο χρήστης συμμετέχει ενεργά στην συνομοσπονδία νεφών, και τιμή Ψευδές (False) όταν ο χρήστης έχει αποχωρήσει από την συνομοσπονδία.
- *rating*: Η βαθμολογία του παρόχου που εκφράζει το πόσο ενεργή και αποτελεσματική είναι η συμμετοχή του στη συνομοσπονδία.
- *RD*: Η απόκλιση βαθμολογίας (rating deviation), όπως ορίζεται από το σύστημα glicko.
- *url:* Το URL επικοινωνίας του παρόχου
- *pwd*: Ο κατακερματισμένος κωδικός πρόσβασης του χρήστη.

Η συλλογή αυτή χρησιμοποιείται από τα υποσυστήματα-κλάσεις *Root*, Matchmaking-*agent* και *Rating-agent*.

Παρακάτω παρουσιάζεται ένα έγγραφο-παράδειγμα της προαναφερθείσας συλλογής για τον πάροχο με όνομα user1, που είναι ενεργός και έχει μόλις εγγραφεί στο σύστημα καθώς έχει τις τυποποιημένες τιμές βαθμολογίας και απόκλισης βαθμολογίας.

```
/* 1 */
{
    "_id" : "user1",
    "active" : true,
    "rating" : 1500,
    "RD" : 350,
    "pwd" : "$6$rounds=656000$MKevrlUEYv5PjjM9$dph6XL3Z4DlKYth5tbXie0/VyzBXjrKd3e5KDvu3ciy8cQYt.K.Fy1
}
```

4.2.2 Η δομή της συλλογής Resource_attributes

Η δομή της συλλογής *res_attributes* στηρίζεται στον τρόπο με τον οποίο κατηγοριοποιούνται οι υποδομές ως υπηρεσίες στην αγορά. Οι υποδομές συνδυάζονται με εικόνες μηχανών (Machine Images) και προσφέρονται για κατανάλωση ως μία οντότητα. Αυτή η οντότητα που συνδυάζει υποδομές και εικόνες μηχανών ονομάζεται εικονική μηχανή (Virtual Machine ή VM). Τα χαρακτηριστικά που ορίζουν τη μοναδικότητα μιας εικονικής μηχανής, καθώς και άλλα χαρακτηριστικά που καθορίζουν τη ζήτηση και προσφορά της στην αγορά, είναι αυτά που καταγράφονται και χρησιμοποιούνται από τη βάση. Η δομή της βάσης λοιπόν στηρίχτηκε στη βέλτιστη κατηγοριοποίηση των χαρακτηριστικών αυτών και συνεπώς στη μεταξύ τους σχέση.

Αξίζει να σημειωθεί εδώ για ποιο λόγο τα χαρακτηριστικά που ορίζουν τις δυνατότητες μιας εικονικής μηχανής δεν είναι τελείως τεχνικά, δηλαδή για παράδειγμα η χρήση από τις υποδομές μιας συγκεκριμένης μάρκας Κεντρικών Μονάδων Επεξεργασίας με συγκεκριμένα χαρακτηριστικά (συχνότητα ρολογιού κλπ.), μιας συγκεκριμένης μάρκας και αρχιτεκτονικής μνήμη κοκ. Στην αγορά των υποδομών ως υπηρεσίες, οι υπηρεσίες κατηγοριοποιούνται, σε τεχνικό επίπεδο, κυρίως ως προς την χρήση για την οποία προορίζονται. Πιο συγκεκριμένα, μια υποδομή ως υπηρεσία μπορεί να χρησιμοποιηθεί για την υποστήριξη μιας εφαρμογής, οι ανάγκες της οποίας για επεξεργαστική ισχύ να είναι συγκεκριμένες και να έχουν μεγαλύτερη προτεραιότητα από τις ανάγκες της για αποδοτική μνήμη. Σε κάποια άλλη περίπτωση, μπορεί να χρησιμοποιηθεί για την ανάπτυξη μιας καινούριας εφαρμογής, για την οποία οι ανάγκες της σε πόρους δεν είναι ξεκάθαρες, οπότε η ελαστικότητα στους παρεχόμενους πόρους να είναι χρήσιμη. Οι πάροχοι, λοιπόν, κατηγοριοποιούν και αναπτύσσουν τις υπηρεσίες τους με βάση τις ανάγκες που θα πρέπει να καλύψουν, οπότε και οι υποδομές που θα χρησιμοποιήσουν θα είναι και αυτές προσαρμοσμένες στις αντίστοιχες ανάγκες. Συνεπώς, η σύγκριση παρόμοιων υπηρεσιών μεταξύ διαφορετικών παρόχων με βάση τα ακριβή τεχνικά χαρακτηριστικά των υποδομών είναι αόριστη, ασαφής

και δεν έχει κάποιο όφελος. Αντίθετα, η σύγκριση παρόμοιων υπηρεσιών με βάση τα κριτήρια που θέτει η αγορά, όπως είναι το κόστος της υπηρεσίας, η διαθεσιμότητα και η φερεγγυότητα, είναι σαφώς πιο χρήσιμη και ωφέλιμη.

Τα χαρακτηριστικά που μας ενδιαφέρουν, λοιπόν, παραθέτονται παρακάτω μαζί με την αντιστοίχιση τους στα πεδία της βάσης δεδομένων:

Χαρακτηριστικό	**Πεδίο**
Τοποθεσία υποδομής	region
Εικόνα μηχανής (με βάση το λειτουργικό σύστημα)	OS
Τύπος βελτιστοποίησης εικονικής μηχανής (ως προς τη χρήση για την οποία προορίζεται)	optimization
Οικογένεια εικονικής μηχανής (ως προς τα τεχνικά χαρακτηριστικά των υποδομών)	productFamily
Κατηγορία εικονικής μηχανής (ως προς το μέγεθος)	instanceSize
Μέγεθος εικονικής μνήμης (σε GiB = 2^{30} Bytes)	memory
Πλήθος εικονικών Κεντρικών Μονάδων Επεξεργασίας	vCPU
Είδος και μέγεθος υποδομής αποθήκευσης	storageInstance
Τιμή υπηρεσίας	price
Διαθεσιμότητα εικονικής μηχανής	availability

Η σχέση των χαρακτηριστικών αυτών είναι η εξής:

- Μία υποδομή μπορεί να προσφέρεται σε πολλές τοποθεσίες.
- Μία εικόνα μηχανής (λειτουργικό σύστημα) μπορεί να προσφέρει πολλές διαφορετικές υπηρεσίες ως προς τον τύπο βελτιστοποίησης της εικονικής μηχανής.
- Ένας τύπος βελτιστοποίησης εικονικής μηχανής μπορεί προσφέρει πολλές διαφορετικές υπηρεσίες ως προς τα τεχνικά χαρακτηριστικά της υποδομής (οικογένεια εικονικής μηχανής) και το μέγεθος της (κατηγορία εικονικής μηχανής).

- Κάθε οικογένεια και κατηγορία εικονικής μηχανής έχουν διαφορετικά ποσοτικά χαρακτηριστικά, δηλαδή πλήθος εικονικών κεντρικών μονάδων επεξεργασίας, μέγεθος εικονικής μνήμης, είδος και μέγεθος υποδομής αποθήκευσης.
- Η τιμή και διαθεσιμότητα μιας υπηρεσίας καθορίζονται μοναδικά από την τοποθεσία, την εικόνα μηχανής, τον τύπο βελτιστοποίησης, την οικογένεια και την κατηγορία της εικονικής μηχανής.
- Τέλος, φυσικά, ο κάθε χρήστης της πλατφόρμας προσφέρει υπηρεσίες που έχουν υποδομές σε πολλές τοποθεσίες.

Έτσι προκύπτει η δομή της συλλογής *resource_attributes* η οποία παρουσιάζεται στο παρακάτω έγγραφο-παράδειγμα:

Κάθε έγγραφο της συλλογής έχει μοναδικό δείκτη το πεδίο του όνομα χρήστη. Το επόμενο πεδίο είναι ο πίνακας *locations.* Ο πίνακας *locations* περιλαμβάνει τα πεδία *region* και *instances.* Το πεδίο *instances* είναι πίνακας και περιλαμβάνει τα πεδία *OS* και *instanceType.* Το πεδίο *instanceType* είναι και αυτό πίνακας και περιλαμβάνει τα πεδία *optimization, productFamily, instanceSize* και *details.* Τέλος το πεδίο *details* περιλαμβάνει τα «φωλιασμένα» ή «ενσωματωμένα» (embedded) πεδία *vCPU, memory, storageInstance, price* και *availability*.

Έχει χρησιμοποιηθεί δηλαδή το μοτίβο των «ενσωματωμένων» εγγράφων για την οργάνωση των δεδομένων. Κάθε «ενσωματωμένος» πίνακας έχει δικό του τοπικό μοναδικό δείκτη ένα άλλο πεδίο, όπως παρουσιάζεται παρακάτω:

Πίνακας	**Πεδίο μοναδικού δείκτη**
locations	*Region*
instances	*OS*
instanceType	*Optimization*

Ο λόγος που προτιμήθηκε αυτή η δομή από την δομή της δημιουργίας πολλών εγγράφων με χρήση αναφορών μεταξύ τους για να εκφραστούν οι σχέσεις μεταξύ των χαρακτηριστικών, είναι επειδή οι σχέσεις μεταξύ των χαρακτηριστικών των εικονικών μηχανών είναι σταθερές και δεν αναμένεται να τροποποιηθούν σημαντικά στο μέλλον και επίσης για να αποφύγουμε μεγάλο πλήθος εγγράφων. Επιπλέον, προσφέρει πλεονεκτήματα για τον μηχανισμό αναζήτησης που χρησιμοποιούνται από την πλατφόρμα, τα οποία θα εξεταστούν στο υποσύστημα-κλάση του *Matchmaking_agent*. Μάλιστα, το μέγεθος του εγγράφου αυξάνεται όλο και λιγότερο όσο αυξάνονται οι υπηρεσίες που εισέρχονται σε ένα έγγραφο, λόγω των «ενσωματωμένων» πινάκων. Τέλος, είναι απολύτως ρεαλιστικό να υποτεθεί ότι δεν αναμένεται ποτέ να ξεπεραστεί το τεχνικό όριο των 16MB ανά έγγραφο (δηλαδή ανά χρήστη), κάτι που θα επηρέαζε σημαντικά την απόδοση της βάσης δεδομένων.

Η συλλογή χρησιμοποιείται από τα υποσυστήματα-κλάσεις *Records* και *Matchmaking-agent.*

4.2.3 Η συλλογή Transaction_list

Η συλλογή *agent* αποτελείται από έγγραφα που περιγράφουν τα απαραίτητα στοιχεία των συναλλαγών. Κάθε έγγραφο έχει μοναδικό δείκτη το αναγνωριστικό της συναλλαγής. Τα πεδία του εγγράφου είναι:

- *_id*: Το αναγνωριστικό της συναλλαγής. Είναι μοναδικό για κάθε συναλλαγή που γίνεται στη συνομοσπονδία νεφών.
- *Provider_name*: Το *provider_name* είναι μεταβλητή που αντικαθίσταται κάθε φορά από το όνομα χρήστη του παρόχου που αναλαμβάνει να εξυπηρετήσει ένα αίτημα για παροχή υπηρεσιών. Περιέχει ως “ενσωματωμένα” πεδία τα πεδία *client* και *outcome*.
- *client*: Το όνομα χρήστη του παρόχου που ζητά υπηρεσίεςαπό τη συνομοσπονδία.

- *Outcome*: Η έκβαση της συναλλαγής. Παίρνει τιμές *0* ή *1* αν η έκβαση της συναλλαγής είναι επιτυχημένη ή αποτυχημένη αντίστοιχα.

Παρακάτω παρουσιάζεται ένα έγγραφο-παράδειγμα της συλλογής στο οποίο, η μεταβλητή έχει πάρει ως τιμή το όνομα του παρόχου *user5* που ικανοποίησε το αίτημα για πόρους του χρήστη *user1.*

4.2.4 Χρήση Python API

Για την επικοινωνία με την MongoDB τόσο από την πλευρά της πλατφόρμας, όσο και από την πλευρά του *flask_web_server,* χρησιμοποιήθηκε το επίσημο Python API (*pymongo* και *flask_pymongo*) που παρέχεται από την MongoDB, και ο κώδικας γράφτηκε στη γλώσσα προγραμματισμού Python. Στο παρακάτω σχεδιάγραμμα φαίνονται οι μέθοδοι του API που χρησιμοποιήσαμε σε διάφορα σημεία του συστήματος.

Μέθοδος	**Περιγραφή**
find()	Επιστρέφει όλα τα έγγραφα από μία συγκεκριμένη συλλογή, τα οποία πληρούν τα κριτήρια που δίνονται ως όρισμα.
find_one()	Επιστρέφει το πρώτο έγγραφο από μία συγκεκριμένη συλλογή, το οποίο πληροί τα κριτήρια που δίνονται ως όρισμα.
insert_one()	Καταχωρεί ένα καινούριο έγγραφο σε μία συγκεκριμένη συλλογή.
update_one()	Ενημερώνει πολλαπλά πεδία ενός εγγράφου σε μία συγκεκριμένη συλλογή.
aggregate()	Συγκεντρώνει, αναλύει και επεξεργάζεται έγγραφα μίας συγκεκριμένης συλλογής και επιστρέφει οργανωμένα.

	αποτελέσματα όπως καθορίζεται από τα ορίσματα. Η παραπάνω διαδικασία χρησιμοποιεί μηχανισμό σωλήνωσης (pipeline), όπου το κάθε στάδιο επεξεργασίας τροφοδοτεί το επόμενο.

4.3 Το σύστημα της πλατφόρμας

Για την εξυπηρέτηση των αναγκών των χρηστών, αναπτύσσεται η πλατφόρμα Intercloud Exchange Platform.

Η ανάπτυξη της πλατφόρμας στηρίχτηκε στην επέκταση *flask_restful* του *flask_web_server*. Η συγκεκριμένη επέκταση δίνει τη δυνατότητα ανάπτυξης Διεπαφής Προγραμματισμού Εφαρμογών (Application Programming Interface ή API) ως ένα αντικείμενο (object) της Python, με ιδιότητες που ικανοποιούν τις προδιαγραφές της αρχιτεκτονικής REST. Πιο συγκεκριμένα, το αντικείμενο (που αποτελεί το API) δίνει τη δυνατότητα να αντιστοιχίζονται μία προς μία οι μέθοδοι του πρωτοκόλλου HTTP με τις μεθόδους των υποσυστημάτων-κλάσεων της πλατφόρμας. Δηλαδή, κάθε προορισμός URL αντιστοιχίζεται με ένα υποσύστημα-κλάση της πλατφόρμας και η αντίστοιχη μέθοδος (POST, GET, DELETE, PATCH, PUT) του εκάστοτε HTTP αιτήματος δρομολογείται κατευθείαν από τον εξυπηρετητή *flask_web_server* στην κατάλληλη μέθοδο του υποσυστήματος-κλάσης προορισμού. Ο μηχανισμός αυτός παρουσιάζεται στο σχεδιάγραμμα της αρχιτεκτονικής ως το σύστημα URI Allocator.

Κάθε υποσύστημα της πλατφόρμας, λοιπόν, υλοποιείται ως μία υποκλάση της κλάσης *Resource*. Η κλάση *Resource* είναι η γονική κλάση (parent class) του *flask_restful* API, που προσφέρει τις δυνατότητες που αναφέρθηκαν από πάνω.

Όπως παρουσιάστηκε και στην αρχιτεκτονική του συστήματος μας, η πλατφόρμα θα αποτελείται από τέσσερα υποσυστήματα ανεξάρτητα μεταξύ τους, το καθένα με στόχο την εξυπηρέτηση διαφορετικών αιτημάτων. Τα υποσυστήματα αυτά είναι τα εξής:

- Υποσύστημα Root
- Υποσύστημα Records
- Υποσύστημα Matchmaking_agent
- Υποσύστημα Rating_agent

Κάθε ένα από τα παραπάνω υποσυστήματα έχει διαφορετικά χαρακτηριστικά και υλοποιεί διαφορετικές μεθόδους για την σωστή λειτουργία της συνομοσπονδίας. Στη συνέχεια παρουσιάζεται κάθε υποσύστημα ξεχωριστά.

4.3.1 Το υποσύστημα Root

Η χρήση του υποσυστήματος *Root* ενεργοποιείται όταν ο *Flask_Web_Server* λάβει αίτημα HTTP με προορισμό το URL /root/. Τα αιτήματα που αποστέλλονται από το χρήστη στο συγκεκριμένο URL μπορούν να είναι :

- POST: Για εγγραφή ή για επανεγγραφή του χρήστη στην πλατφόρμα.
- GET: Για να λάβει αδειοδοτικό χρήσης για να μπορεί να χρησιμοποιήσει τις υπηρεσίες της πλατφόρμας.
- DELETE: Για να διαγραφεί από τη πλατφόρμα.
- PATCH: Για να αλλάξει το κωδικό πρόσβασης.

Οι μέθοδοι *hash_pashword* και *generate_auth_token* καλούν με τη σειρά τους μεθόδους από βιβλιοθήκες που είναι υπεύθυνες για την ασφάλεια της πλατφόρμας. Οι βιβλιοθήκες και οι μέθοδοι αυτές θα περιγράφουν αναλυτικά στο κεφάλαιο της υλοποίησης. Το συγκεκριμένο υποσύστημα-κλάση επικοινωνεί μόνο με την συλλογή *providers* της βάσης δεδομένων.

Ακολουθεί αναλυτική περιγραφή του υποσυστήματος:

<table>
<tr><th>Όνομα κλάσης</th><th colspan="3">Root</th><th>Resource</th></tr>
<tr><td rowspan="3">Μεταβλητές</td><td>_name</td><td>_rating</td><td colspan="2">Active</td></tr>
<tr><td>_pwd</td><td>_RD</td><td colspan="2">Result</td></tr>
<tr><td>h_pwd</td><td>url</td><td colspan="2"></td></tr>
<tr><td>hash_password</td><td colspan="4">Κατακερματισμός κωδικού πρόσβασης χρήστη για να καταχωρηθεί στη βάση.</td></tr>
<tr><td>generate_auth_token</td><td colspan="4">Παραγωγή περιορισμένης χρονικής διάρκειας αδειοδοτικού χρήσης με βάση τα στοιχεία του χρήστη και ενός «κρυφού» κλειδιού.</td></tr>
</table>

POST	Γίνεται γραμματική ανάλυση αιτήματος POST και έλεγχος των συμπληρωθέντων στοιχείων. Στη συνέχεια πραγματοποιείται επικοινωνία με τη βάση δεδομένων για τον έλεγχο μοναδικότητας του χρήστη. Εφόσον επαληθευτούν τα παραπάνω και ο χρήστης κάνει εγγραφή πρώτη φορά, δημιουργείται εγγραφή στη συλλογή *providers* με πεδία: • όνομα χρήστη • κατακερματισμένος κωδικός πρόσβασης • βαθμολογία • απόκλιση βαθμολογίας • ενεργός (boolean μεταβλητή active με αρχική τιμή *True*) • URL επικοινωνίας Η αρχική τιμή της βαθμολογίας χρήστη είναι *1500* και της απόκλισης βαθμολογίας *350* για πρώτη εγγραφή. Σε περίπτωση που ο χρήστης κάνει επανεγγραφή, τότε υπάρχει ήδη εγγραφή, στην οποία αλλάζουν (update) μόνο τα πεδία του κωδικού πρόσβασης, του URL επικοινωνίας και από ανενεργός γίνεται ενεργός (η μεταβλητή *active* γίνεται *True).* Τέλος, γίνεται κλήση της μεθόδου *generate_auth_token* για τη δημιουργία αδειοδοτικού χρήσης. Η μέθοδος επιστρέφει κατάλληλο μήνυμα ανάλογα με την έκβαση του αιτήματος.
GET	Γίνεται γραμματική ανάλυση αιτήματος GET και έλεγχος των συμπληρωθέντων στοιχείων. Στη συνέχεια πραγμοτοποιείται επικοινωνία με τη βάση δεδομένων για την εύρεση ενεργού χρήστη. Εφόσον βρεθεί ο χρήστης και επιβεβαιωθούν τα στοιχεία του με κλήση της μεθόδου *pwd_context.verify*, δημιουργείται κατάλληλο αδειοδοτικό χρήσης με κλήση της μεθόδου *generate_auth_token.* Η μέθοδος επιστρέφει κατάλληλο μήνυμα ανάλογα με την έκβαση του αιτήματος.

DELETE	Γίνεται γραμματική ανάλυση αιτήματος DELETE και έλεγχος των συμπληρωθέντων στοιχείων. Στη συνέχεια πραγματοποιείται επικοινωνία με τη βάση δεδομένων για την εύρεση και ταυτοποίηση ενεργού χρήστη με τρόπο πανομοιότυπο της μεθόδου *get* που περιεγράφηκε παραπάνω. Αν βρεθεί ενεργός χρήστης και ταυτοποιηθεί, τότε χαρακτηρίζεται ως ανενεργός (η μεταβλητή *active* γίνται *False)* και διαγράφονται όλα τα πεδία στο έγγραφο του εκτός από τη βαθμολογία και την απόκλιση βαθμολογίας σε περίπτωση που θελήσει να επανεγγραφεί.
PATCH	Γίνεται γραμματική ανάλυση αιτήματος PATCH και έλεγχος των συμπληρωθέντων στοιχείων. Στη συνέχεια, πραγματοποιείται επικοινωνία με τη βάση δεδομένων για εύρεση και ταυτοποίηση ενεργού χρήστη, όπως περιγράφεται και στις παραπάνω μεθόδους. Ο καινούριος κωδικός πρόσβασης κατακερματίζεται με κλήση της μεθόδου *hash_password* και αντικαθιστά το υπάρχον πεδίο του κωδικού πρόσβασης στο έγγραφο του χρήστη.

4.3.2 Το υποσύστημα Records

Η χρήση του υποσυστήματος *Records* ενεργοποιείται όταν ο *Flask_Web_Server* λάβει αίτημα HTTP με προορισμό το URL /update/. Τα αιτήματα που αποστέλλονται από το χρήστη στο συγκεκριμένο URL μπορούν να είναι:

- POST: Για να καταχωρήσει καινούρια υπηρεσία και τα χαρακτηριστικά της στην πλατφόρμα.
- DELETE: Για να διαγράψει μια υπηρεσία από τη πλατφόρμα.
- PATCH: Για να αλλάξει συγκεκριμένα χαρακτηριστικά, μιας συγκεκριμένης υπηρεσίας, από την πλατφόρμα.

Κατά την κλήση των μεθόδων στο συγκεκριμένο υποσύστημα-κλάση, ενεργοποιείται μία συνάρτηση «περιτυλίγματος» (decorator function) από την επέκταση *flask_httpauth* που είναι υπεύθυνη για την ασφαλή χρήση της πλατφόρμας. Η συγκεκριμένη συνάρτηση

(*verify_token*) διαβάζει την επικεφαλίδα *Authorization* του αιτήματος HTTP, από όπου λαμβάνει το αδειοδοτικό χρήσης για να ταυτοποιήσει τον αποστολέα. Θα ακολουθήσει αναλυτική περιγραφή του παραπάνω μηχανισμού στο κεφάλαιο της υλοποίησης. Το υποσύστημα επικοινωνεί μόνο με τη συλλογή *Resource_attributes* της βάσης δεδομένων.

Ακολουθεί αναλυτική περιγραφή του υποσυστήματος:

<table>
<tr><th>Όνομα κλάσης</th><th colspan="3">Records</th><th>Resource</th></tr>
<tr><td rowspan="5">Μεταβλητές</td><td>reqparse</td><td>_optimization</td><td colspan="2">_storageInstance</td></tr>
<tr><td>args</td><td>_productFamily</td><td colspan="2">_price</td></tr>
<tr><td>_name</td><td>_instanceSize</td><td colspan="2">_availability</td></tr>
<tr><td>Region</td><td>_Vcpu</td><td colspan="2">Results</td></tr>
<tr><td>_OS</td><td>_memory</td><td colspan="2">g.user</td></tr>
<tr><td>POST</td><td colspan="4">Γίνεται γραμματική ανάλυση αιτήματος POST και έλεγχος των συμπληρωθέντων στοιχείων. Η διαδικασία συνεχίζει στην περίπτωση που η ταυτότητα του αποστολέα και το περιεχόμενο του αιτήματος είναι συμβατά. Στη συνέχεια πραγμοτοποιείται επικοινωνία με τη βάση δεδομένων για έλεγχο μοναδικότητας της υπηρεσίας. Εφόσον η υπηρεσία δεν υπάρχει στη βάση, τότε καταχωρούνται τα κατάλληλα χαρακτηριστικά σε νέα πεδία, ώστε να οριστεί καινούρια υπηρεσία. Αν κάποια χαρακτηριστικά ορίζουν πίνακες που δεν υπάρχουν, τότε αυτοί δημιουργούνται κατά περίπτωση. Η παραπάνω διαδικασία γίνεται με πολλαπλά, και κατά περίπτωση, αιτήματα στη βάση.</td></tr>
<tr><td>DELETE</td><td colspan="4">Γίνεται γραμματική ανάλυση αιτήματος DELETE και έλεγχος των συμπληρωθέντων στοιχείων. Η διαδικασία συνεχίζει στην περίπτωση που η ταυτότητα του αποστολέα και το περιεχόμενο του αιτήματος είναι συμβατά. Στη συνέχεια πραγμοτοποιείται επικοινωνία με τη βάση δεδομένων για τον εντοπισμό της υπηρεσίας προς διαγραφή. Διαγράφονται μόνο τα πεδία που ορίζουν την μοναδικότητα της υπηρεσίας (οικογένεια προϊόντος, μέγεθος της οντότητας κλπ) και τα ποσοτικά πεδία που της αντιστοιχούν (τιμή, διαθεσιμότητα κλπ).</td></tr>
</table>

PATCH	Γίνεται γραμματική ανάλυση αιτήματος PATCH και έλεγχος των συμπληρωθέντων στοιχείων. Η διαδικασία συνεχίζει στην περίπτωση που η ταυτότητα του αποστολέα και το περιεχόμενο του αιτήματος είναι συμβατά. Στη συνέχεια πραγμοτοποιείται επικοινωνία με τη βάση δεδομένων για τον εντοπισμό της υπηρεσίας προς τροποποίηση. Μπορούν να τροποποιηθούν μόνο τα ποσοτικά πεδία της τιμής και της διαθεσιμότητας.

4.3.3 Το υποσύστημα Matchmaking_agent

Η χρήση του υποσυστήματος *Matchmaking_agent* ενεργοποιείται όταν ο *Flask_Web_Server* λάβει αίτημα HTTP με προορισμό το URL */search/.* Το αίτημα που αποστέλλεται από το χρήστη στο συγκεκριμένο URL μπορεί να είναι μόνο τύπου GET. Το υποσύστημα επεξεργάζεται το αίτημα κάνοντας δυναμική αναζήτηση στη συλλογή *resource_attr,* χρησιμοποιώντας παράλληλα δεδομένα από τη συλλογή *providers,* προκειμένου να κάνει τη βέλτιστη αντιστοίχιση του αιτήματος με τον κατάλληλο πάροχο που είναι ικανός για την εξυπηρέτηση του.

Σε περίπτωση επιτυχούς αντιστοίχισης, καλείται η μέθοδος *import_data* που κάνει καταχώρηση καινούριας συναλλαγής στη συλλογή *transaction_list* και επιστρέφεται κατάλληλο μήνυμα στον αποστολέα που τον ενημερώνει για την επιτυχή αντιστοίχιση και του κοινοποιείται το URL του παρόχου που ικανοποιεί τα κριτήρια καθώς και το αναγνωριστικό της συναλλαγής.

Κατά την κλήση της μεθόδου GET, ενεργοποιείται η συνάρτηση “περιτυλίγματος” *verify_token* της επέκτασης *flask_httpauth*, όπως ακριβώς περιγράφηκε και για τις μεθόδους του υποσυστήματος-κλάσης *Records.*

Όνομα κλάσης	**Matchmaking_agent**		**Resource**
Μεταβλητές	*reqparse*	*vm*	*region_match*
	outcome	*_id_lst*	*os_match*

	contract	*_loc_lst*	*match_dict*
	prov_rating	*_os_lst*	*final_id_list*
	prov_RD	*vm_values*	*g.user*
	lower_bound	*_mandatory*	*match*
	upper_bound	*_optional*	
import_data	Εισαγωγή καινούριας συναλλαγής στη συλλογή *transaction_list*.		
match_making	Σε περίπτωση αντιστοίχισης περισσότερων από έναν παρόχων με τις απαιτήσεις του αιτήματος, καλείται η συγκεκριμένη μέθοδος, έτσι ώστε να βρεθεί ο καταλληλότερος πάροχος μέσω του μηχανισμού glicko.		
GET	Γίνεται γραμματική ανάλυση αιτήματος GET και έλεγχος των συμπληρωθέντων στοιχείων. Στη συνέχεια πραγματοποιείται επικοινωνία με τη βάση δεδομένων για την ταυτοποίηση του αποστολέα και τη λήψη των στοιχείων της κατάταξης του στη συνομοσπονδία νεφών. Ακολουθεί επιπλέον γραμματική ανάλυση του αιτήματος, και κατασκευάζονται κατάλληλες δομές από τα κριτήρια που έχει θέσει ο αποστολέας στο αίτημα, για την αναζήτηση κατάλληλου παρόχου. Πιο συγκεκριμένα, για κάθε εικονική μηχανή που ζητά ο αποστολέας, προσδιορίζεται ποια από τα κριτήρια που δίνονται είναι αναγκαία για την εύρεση κατάλληλου παρόχου και ποια από τα κριτήρια είναι προαιρετικά. Αυτά αντιστοιχίζονται στα λεξικά (dictionaries) *_mandatory* και *_optional*. Για τα προαιρετικά κριτήρια προσδιορίζεται και η προτεραιότητα με την οποία θα γίνει η αναζήτηση για κάθε κριτήριο, έτσι ώστε να ικανοποιηθούν πρώτα τα πιο σημαντικά από αυτά. Εφόσον βρεθούν αντιστοιχίες με παρόχους		

	για κάθε εικονική μηχανή που ζητείται από το αίτημα, τότε επιλέγονται οι πάροχοι που μπορούν να παρέχουν όλες τις εικονικές μηχανές. Στην περίπτωση που προκύψουν περισσότεροι από ένας πάροχοι καλείται η μέθοδος *match_making* για να επιλεχθεί ένας μόνο πάροχος. Τέλος, καλείται η μέθοδος *import_data* για την καταχώρηση καινούριας συναλλαγής και επιστρέφεται κατάλληλο μήνυμα στον αποστολέα ενημερώνοντας τον για την έκβαση του αιτήματος.

4.3.4 Το υποσύστημα Rating_agent

Η χρήση του υποσυστήματος Rating_agent ενεργοποιείται όταν ο *Flask_Web_Server* λάβει αίτημα HTTP με προορισμό το URL /rating_agent/. Τα αιτήματα που αποστέλλονται μπορούν να προέρχονται μόνο από το βαθμολογικό σύστημα. Στο συγκεκριμένο URL μπορούν να σταλούν αιτήματα της μορφής:

- GET: Για την αποστολή της λίστας των συναλλαγών στο σύστημα βαθμολόγησης.
- PUT: Για ενημέρωση της βαθμολογίας απόκλισης στην αρχή της βαθμολογικής περιόδου.
- PATCH: Για την ενημέρωση των βαθμολογιών και των αποκλίσεων βαθμολογίας των χρηστών στο τέλος της βαθμολογικής περιόδου

Κατά την κλήση των παραπάνω μεθόδων GET, PUT, PATCH ενεργοποιείται η συνάρτηση "περιτυλίγματος" *verify_token* της επέκτασης *flask_httpauth*, όπως ακριβώς περιγράφηκε και για τις μεθόδους των παραπάνω υποσυστημάτων. Ο μοναδικός, όμως, που έχει δικαίωμα χρήσης του συγκεκριμένου υποσυστήματος είναι το σύστημα βαθμολόγησης το οποίο και επικοινωνεί μόνο με τις συλλογές *Transaction_list* και *providers* της βάσης δεδομένων.

Ακολουθεί αναλυτική περιγραφή του υποσυστήματος:

Όνομα κλάσης	**Rating_agent**		**Resource**
Μεταβλητές	*reqparse*	*g.user*	*provider_list*
	args	*c*	*RD_*
	provider	*rating*	*patched*

GET	Γίνεται γραμματική ανάλυση αιτήματος GET και η διαδικασία συνεχίζει στην περίπτωση που η ταυτότητα του αποστολέα ταιριάζει με αυτή του συστήματος βαθμολόγησης. Στη συνέχεια πραγματοποιείται αναζήτηση στη συλλογή transaction_list για τις συναλλαγές που ολοκληρώθηκαν στη συγκεκριμένη περίοδο βαθμολόγησης και δημιουργείται η λίστα providers_list σε μορφή κατάλληλη για επεξεργασία από το σύστημα βαθμολόγησης. Τέλος, διαγράφονται από τη συλλογή οι συναλλαγές που έχουν ολοκληρωθεί και αποστέλλεται η λίστα providers ως απάντηση στο αίτημα.
PUT	Γίνεται γραμματική ανάλυση αιτήματος PUT και η διαδικασία συνεχίζει στην περίπτωση που η ταυτότητα του αποστολέα ταιριάζει με αυτή του συστήματος βαθμολόγησης. Στη συνέχεια πραγματοποιείται επικοινωνία με τη βάση δεδομένων για την ανανέωση των αποκλίσεων βαθμολογίας όλων των χρηστών με βάση τη σταθερά *c* που έχει ληφθεί από το σύστημα βαθμολόγησης. Τέλος, ενημερώνει τον αποστολέα ότι το αίτημα ολοκληρώθηκε για να συνεχιστεί η λειτουργία του συστήματος βαθμολόγησης.
PATCH	Γίνεται γραμματική ανάλυση αιτήματος PATCH και η διαδικασία συνεχίζει στην περίπτωση που η ταυτότητα του αποστολέα ταιριάζει με αυτή του συστήματος βαθμολόγησης. Στη συνέχεια πραγματοποιείται επικοινωνία με τη βάση δεδομένων για τον εντοπισμό του χρήστη προς τροποποίηση. Μπορούν να τροποποιηθούν μόνο τα πεδία *rating* και R*D* με τις ανανεωμένες βαθμολογίες που έχουν αποσταλεί από το σύστημα βαθμολόγησης. Τέλος, ενημερώνει το σύστημα βαθμολόγησης ότι η διαδικασία έχει ολοκληρωθεί επιτυχώς. Η διαδικασία επαναλαμβάνεται για κάθε χρήστη που έχει συμμετάσχει στην τρέχουσα βαθμολογική περίοδο.

4.4 Το σύστημα βαθμολόγησης

Στο στάδιο της ανάλυσης, αναλύσαμε την επιλογή των χρηστών με βάση την βαθμολογία τους και την απόκλιση βαθμολογίας τους, ιδέα που στηρίχτηκε στον αλγόριθμο Glicko. Στο κομμάτι αυτό, αναλύεται το υποσύστημα το οποίο είναι υπεύθυνο για την ανανέωση των βαθμολογιών και της απόκλισης βαθμολογίας των χρηστών. Για να εφαρμόσουμε τον συγκεκριμένο αλγόριθμο αρχικά ορίζουμε μια περίοδο βαθμολόγησης η οποία διαμορφώνεται ανάλογα με τις ανάγκες μας. Στη συνέχεια, το υποσύστημα υπολογίζει τις νέες βαθμολογίες και τις αποκλίσεις βαθμολογίας για τους χρήστες που συμμετείχαν στη συνομοσπονδία για την συγκεκριμένη περίοδο βαθμολογίας έχοντας πραγματοποιήσει κάποια συναλλαγή. Ακολουθούνται δύο βασικά βήματα για την επίτευξη του παραπάνω στόχου:

- *ΒΗΜΑ 1*

  Στο πρώτο βήμα καθορίζεται η τιμή της βαθμολογίας (r) και της απόκλισης βαθμολογίας (RD) στην αρχή της περιόδου βαθμολόγησης. Αν ένας χρήστης συμμετέχει για πρώτη φορά στη συνομοσπονδία θέτουμε r=1500 και RD=150. Σε κάθε άλλη περίπτωση το σύστημα ανανεώνει το RD του κάθε χρήστη με βάση τον τύπο:

$$RD = \min\left(\sqrt{{RD_{old}}^2 + c^2}, RD_{max}\right) \quad (1)$$

  Το RD_{max} καθορίζει το μέγιστο επίπεδο αβεβαιότητας που μπορεί να υπάρχει για κάθε χρήστη ενώ η σταθερά c προσδιορίζει την αύξηση του επιπέδου αβεβαιότητας σε κάθε βαθμολογική περίοδο. Η σταθερά c μπορεί να υπολογιστεί ως ο χρόνος που χρειάζεται να περάσει, σε μονάδες βαθμολογικών περιόδων, πριν η βαθμολογία ενός τυπικού χρήστη γίνει το ίδιο αβέβαιη με αυτή ενός μη-βαθμολογημένου (μη-εγγεγραμμένου) χρήστη.

- *ΒΗΜΑ 2*

Στο τέλος κάθε βαθμολογικής περιόδου το σύστημα επεξεργάζεται τη λίστα με τις συναλλαγές και τα αποτελέσματα τους για κάθε χρήστη που συμμετείχε. Υποθέτοντας ότι ένας χρήστης ολοκλήρωσε ως πάροχος m συναλλαγές με βαθμολογίες πελατών r_1, r_2, r_3….r_m και αποκλίσεις βαθμολογίας RD_1, RD_2, RD_3….RD_m οι νέες τιμές υπολογίζονται με βάση τους τύπους:

$$r_{new} = \mathrm{r} + \frac{q\,g}{{}^{1}/_{RD^2} + {}^{1}/_{d^2}} \sum_{j=1}^{m} (s_j - E(s|r, r_j, RD_j)) \quad (2)$$

$$RD_{new} = \sqrt{({}^{1}/_{RD^2} + {}^{1}/_{d^2})^{-1}} \quad (3)$$

$$q = \frac{\ln 10}{400} = 0.0057565 \quad (4)$$

$$g(RD) = \frac{1}{\sqrt{1 + 3q^2(RD^2)/\pi^2}} \quad (5)$$

$$E\left(s\middle|r, r_j, RD_j\right) = \frac{1}{1 + 10^{-q(RD_j)(r - r_j)/400}} \quad (6)$$

$$d^2 = [q^2 \sum_{j=1}^{m} \left(g\left(RD_j\right)\right)^2 E\left(s\middle|r, r_j, RD_j\right)(1 - E\left(s\middle|r, r_j, RD_j\right))]^{-1} \quad (7)$$

Αυτοί οι υπολογισμοί εκτελούνται για κάθε πάροχο που συμμετείχε στη βαθμολογική περίοδο. Η παραλλαγή του συστήματος μας έγκειται στο γεγονός ότι το σύστημα Glicko δέχεται ως αποτέλεσμα και την ισοπαλία ενώ το δικό μας σύστημα έχει μόνο δύο πιθανά αποτελέσματα (s_j) για κάθε συναλλαγή, νίκη και ήττα, που κωδικοποιούνται με 0 και 1. Επίσης, στο δικό μας σύστημα το αποτέλεσμα μπορεί να είναι 0 και 1 μόνο για τους παρόχους, το οποίο προσδιορίζει αν η συμφωνία παρόχου-πελάτη ικανοποιήθηκε επαρκώς. Όσον αφορά τον υπολογισμό της νέας βαθμολογίας του πελάτη που ζήτησε πόρους από τη συνομοσπονδία, δεχόμαστε μόνο ένα δυνατό αποτέλεσμα ανεξάρτητα το αποτέλεσμα της συναλλαγής του με τον πάροχο και αυτό είναι το 0. Το κίνητρο μας για τις παραπάνω αλλαγές στο σύστημα, είναι η ανταμοιβή των παρόχων που όχι μόνο συμμετέχουν συχνότερα στη συνομοσπονδία αλλά και που παρέχουν καλές υπηρεσίες. Ταυτόχρονα, δεχόμενοι μόνο το αποτέλεσμα της ήττας για τους πελάτες της πλατφόρμας, προσπαθούμε να αποτρέψουμε φαινόμενα κατάχρησης της πλατφόρμας, κακόβουλης και αλόγιστης χρήσης της.

Η χρήση του υποσυστήματος υπολογισμού των νέων βαθμολογιών Glicko_algorithm ενεργοποιείται περιοδικά και διαρκεί ένα προκαθορισμένο χρονικό διάστημα, την περίοδο βαθμολόγησης, την οποία μπορούμε να ορίσουμε και να μεταβάλλουμε ανάλογα με τις εκάστοτε ανάγκες.

Όνομα Αρχείου	**Glicko_algorithm.py**		
Μεταβλητές	*Providers_list*	*q*	*G*
	E	d^2	*RD_new*
	rating_new	*ratings_list*	*C*
g_RD	Μέθοδος για τον υπολογισμό της παραμέτρου *g* με βάση τη σχέση (5).		
param_E	Μέθοδος για τον υπολογισμό της παραμέτρου E με βάση τη σχέση (6).		
Param_d	Μέθοδος για τον υπολογισμό της παραμέτρου d^2 με βάση τη σχέση (7).		
new_rating	Μέθοδος για τον υπολογισμό της νέας βαθμολογίας *rating_new* ενός χρήστη με βάση τη σχέση (2).		
New_RD	Μέθοδος για τον υπολογισμό της νέας απόκλισης βαθμολογίας *RD_new* ενός χρήστη με βάση τη σχέση (3).		
convert_dict	Η μέθοδος αυτή χρησιμοποιείται για να μετατρέψει σε κατάλληλη μορφή την λίστα με τις συναλλαγές που λαμβάνεται από το υποσύστημα *Rating Agent*. Οι συναλλαγές αποθηκεύονται στη βάση δεδομένων με τη μορφή πάροχος-πελάτης-αποτέλεσμα. Η συνάρτηση αυτή δημιουργεί την αντίστροφη συναλλαγή, δηλαδή πελάτης-πάροχος-αποτέλεσμα με το αποτέλεσμα να είναι πάντα 0		

	στην προκειμένη περίπτωση ανεξάρτητα από το αποτέλεσμα της συναλλαγής για τους λόγους που αναφέρθηκαν παραπάνω. Στη συνέχεια, επισυνάπτει όλες τις διαφορετικές συναλλαγές που προϋπήρχαν και που προέκυψαν στην *providers_list*.
count_providers	Η συνάρτηση αυτή καλεί τις μεθόδους που είναι υπεύθυνες για τον υπολογισμό των παραμέτρων του συστήματος Glicko. Αφού υπολογιστούν η νέα βαθμολογία και η νέα απόκλιση ενός χρήστη ελέγχει τις τιμές αυτές ώστε να κυμαίνονται στα επιθυμητά πλαίσια.
main_prog	Η μέθοδος αυτή εκτελεί τη συνολική λειτουργία του προγράμματος μας εκτελώντας τις παραπάνω βοηθητικές συναρτήσεις. Αρχικά, δημιουργεί κατάλληλο αίτημα για την επικοινωνία με το υποσύστημα *Rating Agent* και στέλνει μήνυμα εκκίνησης της βαθμολογικής περιόδου επισυνάπτοντας στο μήνυμα τη σταθερά *c* που προαναφέρθηκε (αίτημα PUT). Στη συνέχεια, αφού λάβει απάντηση, στέλνει αίτημα στον *Rating Agent* για αποστολή της λίστα συναλλαγών (αίτημα GET). Έπειτα, μετατρέπει τη λίστα σε κατάλληλη μορφή. Η νέα αυτή λίστα χρησιμοποιείται για τον υπολογισμό των βαθμολογιών και των αποκλίσεων βαθμολογίας όλων των χρηστών που ολοκλήρωσαν κάποια συναλλαγή στην τρέχουσα βαθμολογική περίοδο και τα επισυνάπτει τα αποτελέσματα στην *rating_list*. Τέλος, στέλνει διαδοχικά αιτήματα στον *Rating Agent* με τα νέα δεδομένα των χρηστών για αποθήκευση τους στη βάση δεδομένων του συστήματος μας (αίτημα PATCH).
__main__	Περιοδική εκτέλεση της συνάρτησης *main_prog*.

5

Υλοποίηση

5.1 Flask Web Server

Σε αυτό το κεφάλαιο θα αναλύσουμε την λειτουργία του εξυπηρετητή που αναπτύξαμε σε Flask, ένα διαδικτυακό πλαίσιο γραμμένο σε Python, και ο οποίος είναι υπεύθυνος για την διαχείριση όλων των αιτημάτων στην πλατφόρμα μας, είτε αυτά προέρχονται από τους χρήστες είτε από το σύστημα βαθμολόγησης. Σε αυτό το σημείο αξίζει να σημειωθεί πως υλοποιούμε ένα RESTful interface για τις μεθόδους που απαντάνε στα αιτήματα, το οποίο μπορεί να χρησιμοποιηθεί για οποιαδήποτε άλλη πλατφόρμα που θα θελήσει να χρησιμοποιήσει τη συνομοσπονδία νεφών, καθώς με απλά HTTP requests είναι δυνατό να επικοινωνήσει και να λάβει απαντήσεις μέσω του interface που δημιουργήσαμε.

Η πλατφόρμα μας υλοποιείται ως ένα RESTful API μέσω της επέκτασης Flask RESTful και όλα τα βασικά της υποσυστήματα μαζί με τους RESTful μηχανισμούς για την επικοινωνία μέσω HTTP υλοποιούνται όπως αναλύεται παρακάτω:

__init__.py

```
from flask import Flask, current_app
from flask_restful import Api, Resource, reqparse
from flask_pymongo import PyMongo

app = Flask(__name__)
app.config['MONGO_DBNAME'] = 'restdb'
app.config['MONGO_URI'] = 'mongodb://localhost:27017/restdb'
app.config['SECRET_KEY'] = 'Server_key'
mongo = PyMongo(app)
api = Api(app)

from app import res

api.add_resource(Intercloud.Root, '/root')
api.add_resource(Intercloud.Records, '/update')
api.add_resource(Intercloud.Matchmaking_agent, '/search')
api.add_resource(Intercloud.Rating_Agent, '/rate')
```

Στο αρχείο Intercloud.py περιέχεται η υλοποίηση όλων των υποσυστημάτων και οι μέθοδοι τους που αναλύθηκαν στο κεφάλαιο της σχεδίασης. Παρακάτω παρουσιάζονται οι εισαγωγές βιβλιοθηκών που είναι απαραίτητες για τη λειτουργία της πλατφόρμας, καθώς και η αρχικοποίηση της επέκτασης *flask_httpauth.*

Intercloud.py

```
from flask import jsonify, request, make_response, g
from flask_restful import Api, Resource, reqparse
from flask_pymongo import PyMongo
from app import app, mongo
import math
import operator
import json
from bson import json_util
from collections import defaultdict
from random import randint
from itertools import chain
#for authentication
from flask_httpauth import HTTPTokenAuth
from passlib.apps import custom_app_context as pwd_context
from itsdangerous import (TimedJSONWebSignatureSerializer
                  as Serializer, BadSignature, SignatureExpired)

auth = HTTPTokenAuth(scheme='Token')

@auth.verify_token
def verify_token(token, pwd=None):
      s = Serializer(app.config['SECRET_KEY'])
      try: data = s.loads(token)
      except SignatureExpired:
            return False
      except BadSignature:
            return False
      g.user = data
      return True
```

Η μέθοδος *verify_token* πυροδοτείται από τα «περιτυλίγματα» (decorators) *@auth.login_required* των μεθόδων που απαιτούν ταυτοποίηση χρήστη, καθώς και έλεγχο εγκυρότητας αιτήματος με βάση τη ταυτότητα του χρήστη. Η συγκεκριμένη μέθοδος, μέσα από τους μηχανισμούς του *auth*, λαμβάνει το αδειοδοτικό χρήσης από την επικεφαλίδα *Authorization* του αιτήματος HTTP, και προσπαθεί να ανακτήσει το κρυπτογραφημένο περιεχόμενο του αδειοδοτικού χρήσης μέσω του μηχανισμού *TimedJSONWebSignatureSerializer*. Ο μηχανισμός αυτός χρησιμοποιεί ένα «κρυφό κλειδί» που είναι αποθηκευμένο στο περιβάλλον της πλατφόρμας, το όνομα χρήστη και μία χρονοσφραγίδα για τη δημιουργία και επικύρωση αδειοδοτικών χρήσης. Σε περίπτωση ορθής ανάκτησης, επιστρέφει *True*, ενώ σε περίπτωση μη έγκυρου αδειοδοτικού ή ληγμένου αδειοδοτικού επιστρέφει *False*. Ταυτόχρονα γίνεται ανάθεση του ανακτημένου περιεχομένου *data*, το οποίο

είναι το όνομα χρήστη του αποστολέα, στη καθολική (global) μεταβλητή *g.user*, ώστε να χρησιμοποιηθεί η ταυτότητα του αποστολέα για τον έλεγχο εγκυρότητας αιτήματος μέσα στην καλούσα μέθοδο.

Παρακάτω θα παρουσιαστεί το υποσύστημα-κλάση *Root* το οποίο διαθέτει ποικίλους μηχανισμούς για την εξυπηρέτηση αιτημάτων των χρηστών. Για αυτό το λόγο, αναλύονται οι μέθοδοι που καλούνται για την εξυπηρέτηση των αιτημάτων αυτών.

```
class Root(Resource):
	def __init__(self):
		pass
	def hash_password(self, pwd):
		return pwd_context.encrypt(pwd)
	def generate_auth_token(self, key, expiration=18000):
		s = Serializer(app.config['SECRET_KEY'],\
		expires_in=expiration)
		return s.dumps(key)
	#insert user in database and generate token
	def post(self):
		try: name_ = request.authorization.username
		except: return make_response(jsonify({'message': \
			'No name provided.'}), 400)
		try: pwd_ = request.authorization.password
		except: return make_response(jsonify({'message': \
			'No password provided.'}), 400)
		result = mongo.db['providers'].find_one({'_id': name_})
		if result is not None:
			if result['active'] == True:
				return make_response(jsonify({'Message': \
				'Provider already exists'}), 409)
			else:
				h_pwd = self.hash_password(pwd_)
				mongo.db.providers.update_one({'_id':
name_},
				{'$set':{
					'pwd': h_pwd,
					'active': True
					}
				})
				token = self.generate_auth_token(name_)
				return make_response(jsonify({\
				'token':token.decode(), 'message':'Provider
' + name_ + ' rejoined successfully.'}), 201)

		h_pwd = self.hash_password(pwd_)
		rating_ = 1500
		RD_ = 350
		url = 'http://intercloud/'+name_+'/'
		mongo.db.providers.insert_one(
			{
				'_id': name_,
				#insert with hash
				'pwd': h_pwd,
				'rating': rating_,
				'RD': RD_,
				'active': True,
				'url': url
```

```
                  })
            token = self.generate_auth_token(name_)
            return make_response(jsonify({'token':token.decode(),
                  'message':'Provider ' + name_ + ' joined.'}),
201)
      #generate new token after request
      def get(self):
            try: name_ = request.authorization.username
            except: return make_response(jsonify({'message': \
                  'No name provided.'}), 400)
            try: pwd_ = request.authorization.password
            except: return make_response(jsonify({'message': \
                  'No password provided.'}), 400)
            result = mongo.db['providers'].find_one({'_id': name_,
'active': True})
            if result == None:
                  return make_response(jsonify({'Message': \
                  'You are not registered in the federation.'}),
404)
            h_pwd = result['pwd']
            if pwd_context.verify(pwd_, h_pwd):
                  token = self.generate_auth_token(name_)
                  return make_response(jsonify({ \
                  'token':token.decode(),'message':'New token
generated successfully.'}), 200)
            else:
                  return make_response(jsonify({'message': \
                  'Authentication failed.'}), 401)
      #delete user updating active field
      def delete(self):
            try: name_ = request.authorization.username
            except: return make_response(jsonify({'message': \
                  'No name provided.'}), 400)
            try: pwd_ = request.authorization.password
            except: return make_response(jsonify({'message': \
                  'No password provided.'}), 400)
            result = mongo.db['providers'].find_one({'_id': name_})
            if result == None:
                  response = make_response(jsonify({'Message': \
                  'Provider not found'}), 404)
            elif result['active'] == False:
                  response = make_response(jsonify({'Message': \
                  'Provider not found'}), 404)
            else:
                  h_pwd = result['pwd']
                  if pwd_context.verify(pwd_, h_pwd):
                        mongo.db['providers'].update_one({'_id':
name_},
                              {'$unset': {'pwd':""}, '$set':
{'active': False}})
                        response = make_response(jsonify({\
                        'Message': 'Provider deregistered
successfully.'}), 200)
                  else: response = make_response(jsonify({\
                  'Message': 'Forbidden action.'}), 403)
            return response
      #change password after request
      def patch(self):
            try: name_ = request.authorization.username
            except: return make_response(jsonify({'message': \
                  'No name provided.'}), 400)
```

```
            try: pwd_ = request.authorization.password
            except: return make_response(jsonify({'message': \
                  'No password provided.'}), 400)
            results = mongo.db['providers'].find_one({'_id': name_,
'active': True})
            if results == None:
                  return make_response(jsonify({'Message': \
                  'You are not registered in the federation.'}),
404)
            h_pwd = results['pwd']
            if pwd_context.verify(pwd_, h_pwd):
                  content = request.json
                  try: new_pwd = content['new_password']
                  except: return make_response(jsonify({'message':\
                  'New password not provided.'}), 400)
                  h_pwd = self.hash_password(new_pwd)
                  mongo.db['providers'].update_one(
                        {'_id' : name_},
                        {
                        '$set' : { 'pwd' : h_pwd }
                        }
                        )
                  return make_response(jsonify({'message': \
                  'New password set successfully.'}), 201)
            else: return make_response(jsonify({'Message':\
            'Forbidden action.'}), 403)
```

Υλοποιούνται τέσσερις μέθοδοι για την εξυπηρέτηση των HTTP αιτημάτων που διαβάζουν τα δεδομένα από τα αιτήματα που παίρνουν σαν όρισμα, καλούν την κατάλληλη μέθοδο και τέλος επιστρέφουν ένα μήνυμα μορφής JSON. Η πρώτη μέθοδος post υλοποιεί την εγγραφή του χρήστη στη βάση. Λαμβάνει τα ορίσματα απευθείας από το αίτημα μέσω των μεθόδων request.authorization.username και request.authorization.paassword το οποίο κατακερματίζεται μέσω της μεθόδου hash_password() πριν αποθηκευτεί στη συλλογή providers. Εν συνεχεία, παράγει κατάλληλο αδειοδοτικό χρήσης με χρήση της μεθόδου generate_auth_token(). Η δεύτερη μέθοδος είναι υπεύθυνη για τη εξυπηρέτηση του αιτήματος επιστροφής νέου αδειοδοτικού χρήσης ενώ η τρίτη μέθοδος delete αλλάζει την κατάσταση του χρήστη σε ανενεργή κατ' απαίτηση του. Με την τελευταία μέθοδο patch ο χρήστης μπορεί να αλλάξει τον κωδικό πρόσβασης του. Τέλος, υπάρχουν δύο βοηθητικές μέθοδοι για τον κατακερματισμό του κωδικού πρόσβασης και την παραγωγή του αδειοδοτικού χρήσης.

Παρακάτω θα παρουσιαστεί ο υποσύστημα-κλάση *Records* το οποίο διαθέτει ποικίλους μηχανισμούς για την εξυπηρέτηση αιτημάτων που μπορεί να είναι αρκετά σύνθετα. Για αυτό το λόγο, αναλύεται πρώτα η μορφή των αιτημάτων για να γίνει κατανοητή η υλοποίησή του και στη συνέχεια αναλύονται οι μέθοδοι που καλούνται για την εξυπηρέτηση των πολύπλοκων αυτών αιτημάτων.

Τα αιτήματα που δέχεται το υποσύστημα είναι της μορφής POST, DELETE και PATCH, για την εισαγωγή, διαγραφή και επεξεργασία στοιχείων των υπηρεσιών που προσφέρει στη συνομοσπονδία ένας πάροχος αντίστοιχα. Οι πληροφορίες των στοιχείων των υπηρεσιών κωδικοποιούνται σε μορφή JSON αρχείου, που περνά ως περιεχόμενο στο αίτημα HTTP. Ακολουθούν παραδείγματα των αρχείων JSON για κάθε τύπο αιτήματος που δέχεται το υποσύστημα *Records*:

- POST:

```
{"res_attr":
      {"name": "amazon",
      "location": "Asia Pasific(Tokyo)",
      "OS": "Red Hat Enterprise Linux",
      "optimization": "Compute Optimized",
      "productFamily": "c4",
      "instanceSize": "xlarge",
      "vCPU": "4",
      "memory": "7.5",
      "storageInstance": "7.5 EBS",
      "price": "0.312",
      "availability": "63"
      }
}
```

- DELETE:

```
{"res_attr":
      {"name": "amazon",
      "location": "Asia Pasific(Tokyo)",
      "OS": "Red Hat Enterprise Linux",
      "optimization": "Compute Optimized",
      "productFamily": "c4",
      "instanceSize": "xlarge"
      }
}
```

- PATCH:

```
{"res_attr":
      {"name": "amazon",
      "location": "Asia Pasific(Tokyo)",
      "OS": "Red Hat Enterprise Linux",
      "optimization": "Compute Optimized",
      "productFamily": "c4",
      "instanceSize": "xlarge",
      "availability": "63"
      }
}
```

Το αίτημα POST απαιτεί το σύνολο των χαρακτηριστικών με τα οποία περιγράφεται μία υπηρεσία, ενώ τα DELETE και PATCH απαιτούν μόνο τα χαρακτηριστικά που μπορούν να

περιγράψουν μονοσήμαντα μια υπηρεσία. Επιπλέον, με το αίτημα PATCH, ο πάροχος μπορεί να επεξεργαστεί μόνο τα πεδία της τιμής και της διαθεσιμότητας μιας υπηρεσίας, και συνεπώς απαιτείται να υπάρχουν στο JSON αρχείο τα πεδία *price* ή *availability* ή και τα δύο μαζί.

```
class Records(Resource):
      def __init__(self):
            self.reqparse = reqparse.RequestParser()
            self.reqparse.add_argument('res_attr', type=dict,
required=True,
            help='No proper attribute provided.', location='json')
            self.trans_table=dict.fromkeys(map(ord, ')'), None)

      @auth.login_required
      def post(self):
            args = self.reqparse.parse_args()['res_attr']
            try: name_ = args['name']
            except: return make_response(jsonify({'message': \
                  'No name provided.'}), 400)
            #comment-out next line to disable auth
            if name_ != g.user: return make_response(jsonify({\
            'message': 'Forbidden action.'}), 403)
            try: Region = args['location']
            except: return make_response(jsonify({'message': \
                  'No location provided.'}), 400)

            try: productFamily_ = args['productFamily']
            except: return make_response(jsonify({'message': \
                  'No productFamily provided.'}), 400)

            try: instanceSize_ = args['instanceSize']
            except: return make_response(jsonify({'message': \
                  'No instanceSize provided.'}), 400)

            try: vCPU_ = args['vCPU']
            except: return make_response(jsonify({'message': \
                  'No vCPU provided.'}), 400)

            try: memory_ = args['memory']
            except: return make_response(jsonify({'message': \
                  'No memory provided.'}), 400)

            try: storageInstance_ = args['storageInstance']
            except: return make_response(jsonify({'message': \
                  'No storageInstance provided.'}), 400)

            try: optimization_ = args['optimization']
            except: return make_response(jsonify({'message': \
                  'No optimization provided.'}), 400)

            try: OS_ = args['OS']
            except: return make_response(jsonify({'message': \
                  'No OS provided.'}), 400)

            try: price_ = args['price']
```

```
            except: return make_response(jsonify({'message': \
                  'No price provided.'}), 400)

            try: availability_ = args['availability']
            except: return make_response(jsonify({'message': \
                  'No availability provided.'}), 400)

            results = mongo.db['resource_attr'].find_one(
                  {'_id': name_, 'locations': {
                  '$elemMatch':{  'region': Region,
                        'instances': {'$elemMatch':{
                              'OS': OS_, 'instanceType':\
                              {'$elemMatch':{
                                    'optimization': optimization_,
                                    'productFamily':
productFamily_,
                                    'instanceSize':
instanceSize_}}}
                              }
                        }}
                  })

            if results != None:
                  response = make_response(jsonify({'Message': \
                  'Entry already exists'}), 409)
                  return response

            results =
mongo.db['resource_attr'].find_one({'_id':name_})
            if results != None:
                  mongo.db.resource_attr.update_one(
                        {'_id':name_, 'locations':{'$not':\
                        {'$elemMatch':{'region':Region}}}},\
                        {'$addToSet': \
                        {'locations':{'region':Region,
'instances':[]}}})
                  mongo.db.resource_attr.update_one(
                        {'_id':name_, 'locations':
                              {'$elemMatch':{
                                    'region': Region,
                                    'instances':

      {'$not':{'$elemMatch':{'OS': OS_}}}}}
                        },
                        {'$addToSet': {'locations.$.instances':{
                                    'OS':OS_,
                                    'instanceType':[]}}
                        })
                  mongo.db.resource_attr.update_one(
                        {'_id': name_,
                        'locations.region':Region,
                        'locations.instances.OS':OS_,
                        },
                        {'$push':
{'locations.$[i].instances.$[j].instanceType':
                              {
                              'optimization': optimization_,
                              'productFamily': productFamily_,
                              'instanceSize': instanceSize_,
                              'details':{
                                    'price': float(price_),
```

```
                                        'availability':
int(availability_),
                                        'vCPU': int(vCPU_),
                                        'memory' : float(memory_),
                                        'storageInstance':
storageInstance_}}}
                        },
                        array_filters=[{'i.region':Region},
{'j.OS':OS_}]
                        )
            else:
                  mongo.db.resource_attr.insert_one(
                  {'_id': name_,
                   'locations': [{'region': Region,
                  'instances': [{ 'OS': OS_,
                        'instanceType': [{
                              'optimization':optimization_,
                              'productFamily':productFamily_,
                              'instanceSize':instanceSize_,
                              'details':{
                                    'price': float(price_),
                                    'availability':
int(availability_),
                                    'vCPU': int(vCPU_),
                                    'memory': float(memory_),
                                    'storageInstance':
storageInstance_,
                                                }
                                    }]}]}]})
            return make_response(jsonify({'message':\
            'Resource attributes from provider ' + \
            name_ + ' registered.'}), 201)

      @auth.login_required
      def delete(self):
            args = self.reqparse.parse_args()['res_attr']
            try: _name = args['name']
            except: return make_response(jsonify({'message': \
            'No name provided.'}), 400)
            #comment-out next line to disable auth
            if _name != g.user: return make_response(jsonify({\
            'message':'Forbidden action.'}), 403)
            result = mongo.db['resource_attr'].find_one({'_id':
_name})
            if result is None:
                  response = make_response(jsonify({'Message': \
                  'Provider not found'}), 404)
                  return response
            try: _region = args['location']
            except: return make_response(jsonify({'message': \
            'Attribute location not specified.'}), 400)
            try: _OS = args['OS']
            except: return make_response(jsonify({'message': \
            'Attribute OS image not specified.'}), 400)
            try: _optimization = args['optimization']
            except: return make_response(jsonify({'message': \
            'Attribute optimization not specified.'}), 400)
            try: _productFamily = args['productFamily']
            except: return make_response(jsonify({'message': \
            'Attribute productFamily not specified.'}), 400)
            try: _instanceSize = args['instanceSize']
```

```
            except: return make_response(jsonify({'message': \
            'Attribute instanceSize not specified.'}), 400)
            result = mongo.db['resource_attr'].update_one(
                  {'_id': _name, 'locations': {
                  '$elemMatch':{  'region': _region,
                        'instances': {'$elemMatch':{
                              'OS': _OS, 'instanceType':{\
                              '$elemMatch':{
                                    'optimization': _optimization,
                                    'productFamily':_productFamily,
                                    'instanceSize':_instanceSize}}}
                                    }
                        }}
                  },
                  {'$pull':
{'locations.$[i].instances.$[j].instanceType':\
                  {'$and':[{'optimization': _optimization},
                        {'productFamily': _productFamily},
                        {'instanceSize': _instanceSize}]}}
                  },
                  array_filters=[{'i.region':_region},\
                  {'j.OS':_OS}]
                  )
            if result is not None:
                  return  make_response(jsonify({'Message':\
                  'Item Deleted'}), 200)
            else:
                  return make_response(jsonify({'Message':\
                  'Item not found'}), 400)

def patch(self):
            args = self.reqparse.parse_args()['res_attr']
            try: _name = args['name']
            except: return make_response(jsonify({'message':\
            'No name provided.'}), 400)
            #comment-out next line to disable auth
            if _name != g.user: return make_response(jsonify({\
            'message':'Forbidden action.'}), 403)
            result = mongo.db['resource_attr'].find_one({'_id':
_name})
            if result is None:
                  response = make_response(jsonify({'Message': \
                  'Provider not found'}), 404)
                  return response
            try: _region = args['location']
            except: return make_response(jsonify({'message': \
            'Attribute location not specified.'}), 400)
            try: _OS = args['OS']
            except: return make_response(jsonify({'message':\
            'Attribute OS image not specified.'}), 400)
            try: _optimization = args['optimization']
            except: return make_response(jsonify({'message': \
            'Attribute optimization not specified.'}), 400)
            try: _productFamily = args['productFamily']
            except: return make_response(jsonify({'message': \
            'Attribute productFamily not specified.'}), 400)
            try: _instanceSize = args['instanceSize']
            except: return make_response(jsonify({'message': \
            'Attribute instanceSize not specified.'}), 400)
            price_missing = False
            set_dict = {}
```

```
            try:
                  _price = args['price']

      set_dict['locations.$[i].instances.$[j].instanceType.$[k].det
ails.price'] = float(_price)
            except: price_missing = True
            try:
                  _availability = args['availability']

      set_dict['locations.$[i].instances.$[j].instanceType.$[k].det
ails.availability'] = int(_availability)
            except:
                  if price_missing:
                        return make_response(jsonify({'message':\
                        'Attributes price or availability not
specified.'}), 400)
            result = mongo.db['resource_attr'].update_one(
            {'_id': _name, 'locations': {
            '$elemMatch':{  'region': _region,
                  'instances': {'$elemMatch':{
                        'OS': _OS, 'instanceType':{'$elemMatch':{
                              'optimization': _optimization,
                              'productFamily': _productFamily,
                              'instanceSize': _instanceSize}}}
                        }
                  }}
            },
            {'$set': set_dict
            },
            array_filters=[{'i.region':_region}, {'j.OS':_OS}, \
            {'$and': [{'k.productFamily': _productFamily},\
            {'k.instanceSize': _instanceSize}]}]
            )
            if result.matched_count != 0:
                  if result.modified_count != 0:
                        return  make_response(jsonify({'Message': \
                        'Item Patched', 'result':\
                        result.modified_count}), 200)
                  else:
                        return make_response(jsonify({'Message': \
                        'No changes to be made.'}), 200)
            else:
                  return make_response(jsonify({'Message': \
                  'Item not found'}), 400)
```

Υλοποιούνται τρεις μέθοδοι για την εξυπηρέτηση των HTTP αιτημάτων που διαβάζουν τα δεδομένα από τα αιτήματα που παίρνουν σαν όρισμα με χρήση της μεθόδου reqparse.parse_args(), καλούν την κατάλληλη μέθοδο και τέλος επιστρέφουν ένα μήνυμα μορφής JSON. Δέχεται αίτημα HTTP από το χρήστη με περιεχόμενο JSON το οποίο αναλύει γραμματικά για να αποθηκεύσει σε κατάλληλη μορφή, για περαιτέρω χρήση, τους πόρους που διαθέτει ο χρήστης στη συνομοσπονδία. Η πρώτη μέθοδος post υλοποιεί την εγγραφή των πόρων στη βάση δεδομένων μέσω της συνάρτησης insert_one() αν δεν προϋπάρχει εγγραφή με αντίστοιχο όνομα χρήστη. Στην περίπτωση που υπάρχει, καλεί την συνάρτηση

update_one() διαδοχικά για να εισάγει στη βάση δεδομένων τα χαρακτηριστικά του πόρου, στα σωστά ενσωματωμένα έγγραφα. Εν συνεχεία, επιστρέφει μήνυμα το οποίο προσδιορίζει την έκβαση του αιτήματος σε μορφή JSON με χρήση της μεθόδου make_response(). Η δεύτερη μέθοδος delete είναι υπεύθυνη για τη εξυπηρέτηση του αιτήματος διαγραφής πόρων από τη βάση δεδομένων. Αντίστοιχα, με την προηγουμένη μέθοδο δέχεται αίτημα HTTP με περιεχόμενο JSON το οποίο αναλύει γραμματικά με τη μέθοδο reqparse.parse_args() για τον αυστηρό προσδιορισμό του πόρου προς διαγραφή. Αφού γίνει έλεγχος των συμπληρωθέντων στοιχείων καλεί τη μέθοδο update_one() με όρισμα pull για να διαγράψει το στοιχείο από τη συλλογή Resource_attributes. Τέλος, υλοποιείται η μέθοδος patch η οποία είναι υπεύθυνη για την ανανέωση συγκεκριμένων στοιχείων ενός πόρου που υπάρχει ήδη στη βάση δεδομένων. Το περιεχόμενο σε μορφή JSON περικλείεται στο αίτημα το όποιο επεξεργάζεται η προαναφερθείσα μέθοδος με τη reqparse.parse_args() και γίνεται έλεγχος των συμπληρωθέντων στοιχείων. Στη συνέχεια, με τη μέθοδο update_one() τροποποιείται ο επιθυμητός πόρος με μοναδικά πεδία που επηρεάζονται να είναι η τιμή (price) και η διαθεσιμότητα (availability). Πριν την κλήση κάθε μεθόδου, καλείται η συνάρτηση περιτυλίγματος auth.login_required που αναφέρθηκε παραπάνω, η οποία είναι υπεύθυνη για την ταυτοποίηση που κάνει το αίτημα. Σε περίπτωση αποτυχίας ταυτοποίησης του χρήστη, ή σε περίπτωση που ο συγκεκριμένος χρήστης δεν έχει δικαίωμα πρόσβασης στον πόρο (g.user διάφορος της μεταβλητής _name) τον οποίο επιθυμεί να εισάγει, τροποποιήσει ή διαγράψει επιστρέφεται μήνυμα στον χρήστη μη εγκεκριμένης πρόσβασης.

Παρακάτω θα παρουσιαστεί το υποσύστημα-κλάση *Matchmaking-agent* το οποίο διαθέτει ποικίλους μηχανισμούς για την εξυπηρέτηση αιτημάτων που μπορεί να είναι αρκετά σύνθετα. Για αυτό το λόγο, αναλύεται πρώτα η μορφή των αιτημάτων για να γίνει κατανοητή η υλοποίησή του και στη συνέχεια αναλύονται τμηματικά οι μέθοδοι που καλούνται για την εξυπηρέτηση των πολύπλοκων αυτών αιτημάτων.

Τα αιτήματα που δέχεται το συγκεκριμένο υποσύστημα είναι της μορφής GET και αφορούν την εύρεση του καταλληλότερου παρόχου υπηρεσιών, όπως προσδιορίζονται από τα αιτήματα, καθώς και με βάση τη ποιότητα υπηρεσιών των παρόχων, όπως προκύπτει από το σύστημα glicko.

Το αίτημα λοιπόν, πρέπει να έχει μία αυστηρή, αλλά ταυτόχρονα, ευέλικτη μορφή. Αρχικά, πρέπει να προσδιορίζονται ο αριθμός και τα τεχνικά χαρακτηριστικά των υπηρεσιών που ζητούνται. Έπειτα, πρέπει να δίνονται τα χαρακτηριστικά που θεωρούνται, από τον αποστολέα, αναγκαία για την καταλληλότητα του παρόχου. Τέλος, δίνεται η δυνατότητα

προσδιορισμού χαρακτηριστικών που είναι προαιρετικά για την επιτυχή έκβαση της αναζήτησης, αλλά μπορούν να φιλτράρουν τυχόν πολλαπλούς κατάλληλους παρόχους με βάση τα επιθυμητά αυτά χαρακτηριστικά. Η δυνατότητα αυτή υλοποιείται μέσω δευτερεύουσας αναζήτησης προτεραιότητας.

Το αίτημα της αναζήτησης έχει την εξής μορφή JSON αρχείου:

```
{"search":
  {"1":
    {"optimization":"General Purpose", "vCPU":"8", "memory":"30.5",
        "storageInstance":"EBS", "VMnum":"50",
      "mandatory":{"location":"EU(London)"},
      "optional":{"1.OS":"Windows", "2.price":"2.085"}
    },
  "2":
    {"productFamily":"t4", "instanceSize":"xlarge", "VMnum": "45",
      "mandatory":{"price":"1.005"},
      "optional":{"1.location":"EU(Frankfurt)", "2.OS":"RHEL"}
    }
  }
}
```

Για κάθε εικονική μηχανή που ζητείται προσδιορίζονται αρχικά τα τεχνικά χαρακτηριστικά της. Για την πρώτη εικονική μηχανή προσδιορίζεται ο τύπος βελτιστοποίησης *optimization*, το πλήθος των εικονικών Κεντρικών Μονάδων Επεξεργασίας *vCPU*, το μέγεθος της εικονικής μνήμης *memory* και είδος και μέγεθος υποδομής αποθήκευσης *storageInstance*. Για την δεύτερη εικονική μηχανή προσδιορίζεται η οικογένεια *productFamily* και η κατηγορία μεγέθους *instanceSize* της εικονικής μηχανής. Κάθε υπο-αίτημα που αφορά μια εικονική μηχανή, προσδιορίζεται το πλήθος των εικονικών μηχανών που ζητούνται και έπειτα προσδιορίζονται τα αναγκαία *mandatory* και προαιρετικά *optional* χαρακτηριστικά για την επιτυχία της αναζήτησης. Για κάθε προαιρετικό χαρακτηριστικό δίνεται ένας αριθμός προτεραιότητας που καθορίζει το «φιλτράρισμα» με την επιθυμητή σειρά.

Ακολουθεί η υλοποίηση του υποσυστήματος ανά μέθοδο της κλάσης:

Η μέθοδος *import_data* καλείται μέσα από τη μέθοδο *get* με ορίσματα τη ταυτότητα του αποστολέα του αιτήματος και τη ταυτότητα του κατάλληλου παρόχου που προέκυψε μετά την αναζήτηση. Εισάγει νέα καταχώρηση στη λίστα των συναλλαγών της βάσης δεδομένων και επιστρέφει το αναγνωριστικό της νέας συναλλαγής.

import_data

```
class Matchmaking_agent(Resource):
      def __init__(self):
            self.reqparse = reqparse.RequestParser()
            self.reqparse.add_argument('search', type=dict,
required=True,
```

```
            help='No proper search.', location='json')

      #create new entry in the transaction list
      def import_data(self, pr, cl):
            #simulate random transaction result
            outcome_=randint(0,1)
            contract ={pr:{'client': cl, 'outcome':outcome_}}
            result = mongo.db['agent'].insert_one(contract)
            trans_id = result.inserted_id
            return trans_id
```

Η μέθοδος *match_making* καλείται από τη μέθοδο *get* με ορίσματα τη λίστα των παρόχων που προέκυψαν από την αναζήτηση και τη βαθμολογία του παρόχου που υπέβαλε το αίτημα. Γίνεται αναζήτηση στη συλλογή *providers* με διαδοχικές κλήσεις της *find_one* για κάθε πάροχο για τη συλλογή των απαραίτητων χαρακτηριστικών του κάθε παρόχου. Στη συνέχεια, με κριτήρια τη βαθμολογία και την απόκλιση βαθμολογίας, αν προκύψουν πάροχοι με ίδια βαθμολογία, επιλέγεται ο πάροχος με τη χαμηλότερη βαθμολογία και χαμηλότερη απόκλιση βαθμολογίας αντίστοιχα. Αν προκύψουν πάροχοι με ίδια βαθμολογία και ίδια απόκλιση βαθμολογίας, επιλέγεται ένας τυχαίος πάροχος. Επιστρέφει το όνομα χρήστη του παρόχου και το URL του.

match_making

```
      #use glicko system to reduce matched providers down to one
      def match_making(self,data,low_c):
            my_dict={}
            #get matching providers' characteristics from the db
            urls = {}
            for match in data:
                  results = mongo.db['providers'].find_one(
                        {'_id': match},
                        {'rating':1, 'RD':1, 'url':1, '_id':0}
                  )
                  urls[match] = results['url']
                  dif= results['rating'] - low_c
                  dif=abs(dif)
                  matching=match+'_'+str(results['RD'])
                  my_dict[matching]=dif
            #fetch the provider(s) with the lowest rating
difference from
            #the requesting provider's rating
            min_value=min(my_dict.values())
            result=[key for key,value in my_dict.items() if
value==min_value]
            if len(result)==1:
                  result,RD=result[0].split("_")
                  url = [value for key,value in urls.items() if
key==result]
                  return result, url
            #fetch the provider(s) with the lowest RD
            my_dict={}
            for res in result:
                  res,RD=res.split("_")
                  my_dict[res]=float(RD)
```

```
            min_value=min(my_dict.values())
            result=[key for key,value in my_dict.items() \
                        if value==min_value]
            #just fetch a random provider
            if len(result)==1:
                  url = [value for key,value in urls.items() \
                        if key==result[0]]
                  return result[0], url[0]
            else:
                  j=len(result)-1
                  i=randint(0,j)
                  url = [value for key,value in urls.items() \
                        if key==result[i]]
                  return result[i], url[0]
```

Ακολουθεί η υλοποίηση της μεθόδου *get* τμηματικά:

Αρχικά, αφού ταυτοποιηθεί ο αποστολέας μέσω του *flask_httpauth*, γίνεται αναζήτηση στη συλλογή *providers* της βάσης δεδομένων, ώστε να γίνει ένα αρχικό φιλτράρισμα των παρόχων με βάση το βαθμολογικό τους εύρος στο σύστημα glicko και τη βαθμολογία του αποστολέα, ώστε να περιοριστεί το εύρος της αναζήτησης στην τελική λίστα παρόχων *_id_list* που μπορούν να ικανοποιήσουν το αίτημα.

Get

```
      @auth.login_required
      def get(self):
            args = self.reqparse.parse_args()['search']
            #find providers which have the proper rating
            prov = mongo.db.providers.find_one(
                  {'_id': g.user},
                  {'rating': 1, 'RD':1}
            )
            prov_rating = prov['rating']
            prov_RD = prov['RD']
            lower_bound = prov_rating - 2*prov_RD
            upper_bound = prov_rating + 2*prov_RD
            provs = mongo.db.providers.find(
                  {'rating': {'$lte':upper_bound}, 'active': True},
                  {'_id':1}
            )
            _id_lst = []
            for prov in provs:
                  _id_lst.append(prov['_id'])
```

Για κάθε ζητούμενη εικονική μηχανή γίνεται γραμματική ανάλυση των χαρακτηριστικών της και του πλήθους των εικονικών μηχανών που ζητούνται. Αν το πεδίο *VMnum* που προσδιορίζει το πλήθος των εικονικών μηχανών λείπει ή τα χαρακτηριστικά που δίνονται δεν είναι επαρκή για την αναζήτηση, επιστρέφεται κατάλληλο μήνυμα. Στη συνέχεια γίνεται γραμματική ανάλυση του πεδίου *mandatory* που περιέχει τα αναγκαία χαρακτηριστικά για την επιτυχή αναζήτηση και παράλληλα κατασκευάζονται κατάλληλα λεξικά (*vm_values,*

region_match, os_match) για την αναζήτηση. Ακολουθεί η κλήση της μεθόδου *aggregate* που με μηχανισμό σωλήνωσης (pipeline) αντιστοιχεί έγγραφα με πεδία που προσδιορίζονται από τα λεξικά και τις λίστες που δίνονται (*$match*) και δημιουργεί υπο-έγγραφα από τα έγγραφα που προέκυψαν από την αντιστοίχιση (*$unwind*). Τα διαδοχικά *$match* και *$unwind* κάνουν την αναζήτηση πιο γρήγορη και αποδοτική, αφού το πλήθος των επιμέρους υπο-εγγράφων που αναζητούνται περιορίζεται σταδιακά και τα συνολικό πλήθος των πεδίων της βάσης που εξετάζονται είναι δραματικά μειωμένο. Η *aggregate* επιστρέφει λεξικά με τα ονόματα χρηστών των παρόχων, τις περιοχές ανά πάροχο και εικόνες λειτουργικών συστημάτων ανά πάροχο, με σκοπό να χρησιμοποιηθούν αργότερα σε πιθανές αναζητήσης προαιρετικής φύσεως. Αν δε βρεθούν κατάλληλοι πάροχοι από την *aggregate* επιστρέφεται μήνυμα ότι η αναζήτηση απέτυχε.

```
            #search for each virtual machine
            final_results = {}
            #final_results holds results for both machines
            for _vm in args:
                  vm = args[_vm]
                  vm_values = {}
                  tmp_id_lst = _id_lst
                  tmp_loc_lst = []
                  tmp_os_lst = []
                  #check if number of vms requested is given
                  try:
                        vm_values['locations'\
                        '.instances.instanceType'\
                        '.details.availability'] = \
                        {'$gte': int(vm['VMnum'])}
                  except: return make_response(jsonify({'Message':\
                  'VMnum key missing.'}), 401)
                  #check if vm characteristics define the vm
properly
                  try:
                        vm_values['locations'\
                        '.instances.instanceType'\
                        '.optimization'] =\
                        vm['optimization']
                  except:
                        try:
                              vm_values['locations'\
                              '.instances.instanceType'
                              '.productFamily'] =\
                              vm['productFamily']
                              vm_values['locations'\
                              '.instances.instanceType'\
                              '.instanceSize'] = \
                              vm['instanceSize']
                        except: return \
                        make_response(jsonify({'Message': \
                        'productFamily and instanceSize keys '\
                        'are missing or optimization key '\
                        'is missing.'}), 401)
                  #check if mandatory characteristics are given
```

```
                    try: _mandatory = vm['mandatory']
                    except: return make_response(jsonify\
                    ({'Message':'Mandatory field missing.'}), 401)
                    #construct proper dictionaries to issue the query
                    for key in _mandatory:
                          if key == 'location':
                                tmp_loc_lst = [_mandatory[key]]
                          elif key == 'OS':
                                tmp_os_lst = [_mandatory[key]]
                          elif key == 'price':

      vm_values['locations.instances.instanceType'\
                                '.details.price'] = {'$lte':\
                                float(_mandatory[key])}
                          elif key == 'vCPU':

      vm_values['locations.instances.instanceType'\
                                '.details.vCPU'] = {'$gte': \
                                float(_mandatory[key])}
                          elif key == 'memory':

      vm_values['locations.instances.instanceType'\
                                '.details.memory'] = {'$gte':\
                                float(_mandatory[key])}
                          elif key == 'storageInstance':

      vm_values['locations.instances.instanceType'\
                                '.details.storage'
                                'Instance'] = _mandatory[key]

                    region_match = {}
                    if tmp_loc_lst != []:
                          region_match['locations.region'] = {'$in':\
                                tmp_loc_lst}
                    os_match = {}
                    if tmp_os_lst != []:
                          os_match['locations.instances.OS'] =
{'$in': tmp_os_lst}

                    results = mongo.db.resource_attr.aggregate([
                          {'$match': {'_id': {'$in': tmp_id_lst}}},
                          {'$unwind':'$locations'},
                          {'$match': region_match},
                          {'$unwind':'$locations.instances'},
                          {'$match': os_match},

      {'$unwind':'$locations.instances.instanceType'},
                          {'$match': vm_values},
                          {'$project': {'_id': 1, \
                          'locations.region': 1, \
                          'locations.instances.OS': 1}}
                          ])

                    match_dict = {}
                    for doc in results:
                          try:
                                if doc['locations']['region'] not in\
                                match_dict[doc['_id']]['loc_match']:
                                      match_dict[doc['_id']]\
                                      ['loc_match'].append(\
                                      doc['locations']['region'])
```

```
                        if doc['locations']['instances']\
                        ['OS'] not in match_dict[doc['_id']]\
                        ['os_match']:
                              match_dict[doc['_id']]\
                              ['os_match'].append(\
                              doc['locations']['instances']\
                              ['OS'])
                  except:
                        match_dict[doc['_id']] =\
                        {'loc_match': [doc['locations']\
                        ['region']],'os_match': \
                        [doc['locations']['instances']\
                        ['OS']]}

            #check if proper providers exist
            if match_dict == {}:
                  return make_response(jsonify({'Message':\
                  'Could not ''find a match.'}), 200)
```

Η προαιρετική αναζήτηση υλοποιείται με τον ίδιο τρόπο που υλοποιήθηκε και η βασική αναζήτηση, με τη διαφορά ότι γίνονται διαδοχικές αναζητήσεις τηρώντας την προταιρεότητα που έχει δοθεί από το αίτημα και ότι τα λεξικά που τροφοδοτούν τη μέθοδο *aggregate* είναι πιο σύνθετα, αφού λαμβάνονται υπ'όψη και τα αποτελέσματα της προηγούμενης αναζήτησης. Είτε προκύψουν νέα αποτελέσματα από την προαιρετική αναζήτηση είτε δεν προκύψουν, καλείται η μέθοδος *match_making*, σε περίπτωση που προκύψουν παραπάνω του ενός παρόχου, για να μειώσουμε τους παρόχους που προέκυψαν σε έναν. Στη συνέχεια, καλείται η μέθοδος *import_data* για να ενημερωθεί η λίστα συναλλαγών της βάσης και να λάβουμε το αναγνωριστικό της νέας συναλλαγής. Τέλος, επιστρέφεται μήνυμα που ενημερώνει τον αποστολέα ότι η αναζήτηση πέτυχε, συμπεριλαμβάνοντας το όνομα χρήστη του παρόχου που επιλέχθηκε, το URL του, και το αναγνωριστικού της νέας συναλλαγής.

```
            #check for optional characteristics
            try: _optional = vm['optional']
            except: _optional = {}
            #if there is only one matched provider abort
            #optional search
            if len(match_dict) == 1:
                  _optional = {}
            #proceed optional search based on the
            #prioriterised sorted characteristics
            for _key, value in sorted(_optional.items()):
                  key = _key.split('.')[1]
                  region_match = {}
                  os_match = {}
                  if key == 'location':
                        region_match = \
                        {'locations.region': value}
                  elif key == 'OS':
                        os_match = \
                        {'locations.instances.OS': value}
```

```
                        elif key == 'price':
                              vm_values['locations.instances'
                              '.instanceType.details.price'] \
                              = {'$lte': float(value)}
                        elif key == 'vCPU':
                              vm_values['locations.instances'
                              '.instanceType.details.vCPU'] \
                              = {'$gte': float(value)}
                        elif key == 'memory':
                              vm_values['locations.instances'
                              '.instanceType.details.memory'] \
                              = {'$gte': float(value)}
                        elif key == 'storageInstance':
                              vm_values['locations.instances'
                              '.instanceType.details.'
                              'storageInstance']= value
                        if region_match == {}:
                              or_lst = []
                              for key in match_dict:
                                    or_lst.append({'$and':\
                                          [{'_id': key},
                                          {'locations.region':\
                                                {'$in':\

      match_dict[key]['loc_match']}
                                          }]
                                    })
                              region_match = {'$or': or_lst}
                        if os_match == {}:
                              or_lst = []
                              for key in match_dict:
                                    or_lst.append({'$and':\
                                          [{'_id': key},
                                          {'locations.instances'
                                          '.OS':{'$in':\

      match_dict[key]['os_match']}
                                          }]
                                    })
                              os_match = {'$or': or_lst}
                        tmp_id_list = []
                        for result in match_dict:
                              tmp_id_list.append(result)
                        results =
mongo.db.resource_attr.aggregate([
                              {'$match': {'_id': {'$in': \
                              tmp_id_lst}}},
                              {'$unwind':'$locations'},
                              {'$match': region_match},
                              {'$unwind':'$locations.instances'},
                              {'$match': os_match},
                              {'$unwind':'$locations.instances'
                                    '.instanceType'},
                              {'$match': vm_values},
                              {'$project': {'_id': 1, \
                                    'locations.region': 1,\
                                    'locations.instances.OS': 1}}
                              ])
                        new_match_dict = {}
                        for doc in results:
                              try:
```

```
                                    if doc['locations']['region'] \
                                    not in new_match_dict[doc\
                                    ['_id']]['loc_match']:
                                          new_match_dict\

      [doc['_id']]['loc_match']\
                                          .append(doc['locations']\
                                          ['region'])
                                    if doc['locations']\
                                    ['instances']['OS'] not in \
                                    new_match_dict[doc['_id']]\
                                    ['os_match']:
                                          new_match_dict\
                                          [doc['_id']]['os_match']\
                                          .append(doc['locations']\
                                          ['instances']['OS'])
                              except:
                                          new_match_dict[doc['_id']]\
                                          = {'loc_match':[doc[\
                                          'locations']['region']],\
                                          'os_match':\
                                          [doc['locations']\
                                          ['instances']['OS']]}
                        #if there are no results, keep previous
                        #results and abort optional search
                        if new_match_dict != {}:
                              match_dict = new_match_dict
                        else: break
                  #hold results per virtual machine requested
                  res_lst = []
                  for result in match_dict:
                        res_lst.append(result)
                  final_results[_vm] =  res_lst
            #find the providers that can provide all the vms
            init_key = next(iter(final_results))
            final_id_list = final_results[init_key]
            for vm in final_results:
                  final_id_list = list(\

      set(final_id_list).intersection(final_results[vm]))
            if final_results == []:
                  return make_response(jsonify({'Message': \
                        'No match found.'}), 200)
            #reduce to one matched provider with the help of
            #glicko rating system
            if len(final_id_list) != 1:
                  match, url = self.match_making\
                  (final_id_list, lower_bound)
            else:
                  match = final_id_list[0]
                  result = mongo.db['providers'].find_one(\
                  {'_id': match},{'url':1, '_id':0})
                  url = result['url']
            #create entry for the new impending transaction in the
            #transaction log
            trans_id = self.import_data(match, g.user)
            #return message informing the sender about the new
            #transaction and identity-url of the matched provider
            return make_response(jsonify({'Message': \
            'Match found.','Provider name': match, \
            'URL': url,'Transaction id': trans_id}), 200)
```

Παρακάτω θα παρουσιαστεί η υλοποίηση του υποσυστήματος-κλάσης Rating_agent που υλοποιεί μεθόδους για την επικοινωνία με το σύστημα βαθμολόγησης:

```
class Rating_Agent(Resource): #Rating agent

      def __init__(self):
            self.reqparse = reqparse.RequestParser()
            self.reqparse.add_argument('rating', type=dict,
required=True,
            help='No proper data from agent.', location='json')
            self.provider=defaultdict(dict)

      #creating provider dict, sending it to glicko system
      @auth.login_required
      def get(self):
            args = self.reqparse.parse_args()['rating']
            #comment-out next line to disable auth
            if g.user != 'agent': return make_response(jsonify(\
            {'message': 'Forbidden action.'}), 403)
            json=[]
            results=mongo.db['agent'].find({},{'_id':0})
            for result in results:
                  key=list(result.keys())
                  key=key[0]
                  prov=mongo.db['providers'].find_one({'_id':key},\
                  {'_id':0, 'rating':1, 'RD':1})
                  rating= prov['rating']
                  RD= prov['RD']
                  key_prov=key+'_'+str(rating)+'_'+str(RD)
                  client=result[key]['client']
                  prov=mongo.db['providers'].find_one(\
                  {'_id':client},{'_id':0, \
                  'rating':1, 'RD':1})
                  result[key]['rating']=prov['rating']
                  result[key]['RD']=prov['RD']
                  self.provider.setdefault(key_prov,
[]).append(result[key])
            response=self.provider
            mongo.db['agent'].remove({})
            return response

      #updates rating and RD for each provider
      #data received from glicko_system
      @auth.login_required
      def patch(self):
            args = self.reqparse.parse_args()['rating']
            #comment-out next line to disable auth
            if g.user != 'agent': return make_response(jsonify(\
            {'message':'Forbidden action.'}), 403)
            args=args['update']
            provider=args['provider']
            rating=args['rating']
            RD=args['RD']
            results = mongo.db['providers'].find_one({'name':
provider})
            if results == {}:
                  response = make_response(jsonify({'Message': \
                  'Provider not found'}), 404)
                  return response
```

```
            patched = False
            if rating != None:
                  mongo.db['providers'].update(
                        {'name' : provider},
                        {
                        '$set' : { 'rating' : rating }
                        }
                        )
                  patched = True
            if RD != None:
                  mongo.db['providers'].update(
                        {'name' : provider},
                        {
                        '$set' : { 'RD' : RD }
                        }
                        )
                  patched = True

            if patched != True:
                  response = make_response(jsonify({'Message': \
                  'No proper patch content provided.'}), 400)
                  return response
            return make_response(jsonify({'Message' : 'Provider '+\
            provider + ' patched successfully.'}), 200)

      #updates RD for each provider at the start of rating period
      #constant c received from glicko_system
      @auth.login_required
      def put(self):
            args = self.reqparse.parse_args()['rating']
            #comment-out next line to disable auth
            if g.user != 'agent': return make_response(jsonify(\
            {'message': 'Forbidden action.'}), 403)
            c= float(args['constant'])
            results=mongo.db['providers'].find({},{'_id':0})
            for result in results:
                  RD_=min(math.sqrt(result['RD']**2+c**2),350)
                  mongo.db['providers'].update(
                        {'name':result['name']},
                        {'$set': {'RD': RD_}})
            return make_response(jsonify({'Message' : 'Providers '\
            ' patched successfully.'}), 200)
```

Υλοποιούνται τρεις μέθοδοι για την εξυπηρέτηση των HTTP αιτημάτων που διαβάζουν τα δεδομένα από τα αιτήματα που παίρνουν σαν όρισμα με χρήση της μεθόδου reqparse.parse_args(), καλούν την κατάλληλη μέθοδο και τέλος επιστρέφουν ένα μήνυμα μορφής JSON. Τα αιτήματα αυτά μπορούν να προέρχονται μόνο από το σύστημα βαθμολόγησης το οποίο ταυτοποιείται πριν την κλήση κάθε μεθόδου με χρήση της συνάρτησης περιτυλίγματος auth.login_required ώστε να μην μπορεί ένας χρήστης να επηρεάσει τις βαθμολογίες αποσκοπώντας σε καλύτερη και συχνότερη χρήση του από τη συνομοσπονδία. Η πρώτη μέθοδος GET χρησιμοποιείται από το βαθμολογικό σύστημα για

να λάβει την λίστα με τις συναλλαγές. Αρχικά λοιπόν, γίνεται μία αναζήτηση στη συλλογή Transaction_list όλων των ολοκληρωμένων συναλλαγών με κλήση της συνάρτησης find() και δημιουργείται ένα λεξικό (defaultdict) το οποίο περιλαμβάνει τις βαθμολογίες και τις αποκλίσεις βαθμολογίας των παρόχων σε συνδυασμό με αυτές των πελάτων τους καθώς και το αποτέλεσμα κάθε συναλλαγής. Τέλος, αδειάζει τη συλλογή με τη λίστα των ολοκληρωμένων συναλλαγών και επιστρέφει σαν απάντηση το λεξικό αυτό στο σύστημα βαθμολόγησης. Στη συνέχεια, υλοποιείται η μέθοδος patch η οποία αναλύει γραμματικά το αίτημα με την reqparse.parse_args() και ανανεώνει στη συλλογή providers την βαθμολογία και την απόκλιση βαθμολογίας του χρήστη που δηλώνεται από τη μεταβλητή name. Τέλος, η μέθοδος put δέχεται από το σύστημα βαθμολόγησης την σταθερά c, την οποία χρησιμοποιεί για τον υπολογισμό των νέων αποκλίσεων βαθμολογίας για όλους τους χρήστες της πλατφόρμας και την ενημέρωση της συλλογής providers μέσω της συνάρτησης update().

5.2 Το σύστημα βαθμολόγησης glicko_algorithm

Το σύστημα αυτό προκαλεί την εκκίνηση της βαθμολογικής περιόδου και στη συνέχεια υπολογίζει τις νέες βαθμολογίες και τις αποκλίσεις βαθμολογίας για όλους τους χρήστες που συμμετείχαν στην συνομοσπονδία την τρέχουσα περίοδο πραγματοποιώντας κάποια συναλλαγή. Ο υπολογισμός των νέων δεδομένων γίνεται με βάση τις συναλλαγές που έχουν πραγματοποιηθεί, το αποτέλεσμα της συναλλαγής και με βάση τις προηγούμενες βαθμολογίες των χρηστών. Η βασική λειτουργία του συστήματος παρουσιάζεται παρακάτω:

```
import requests
import json
from bson import json_util
import math
from itertools import chain
from collections import defaultdict
from twisted.internet import task
from twisted.internet import reactor

def main_prog():
      headers={"Content-Type": "application/json"}
      url = 'http://localhost:5000/rate'
      #NEW RATING PERIOD, DEFINE constant c
      #send PUT request to server
      rating_period=60*60 #1 hour
      time_RDmax=48 #48 hours until RD_average=RD_max
      RD_average=50
      RD_max=350
      c=math.sqrt((RD_max**2-RD_average**2)/time_RDmax)
      c=str(round(c,1))
      payload={"rating":{"constant":c}}
      r=requests.put(url, data=json.dumps(payload),
headers=headers)
      print(r.content)
```

```
      #Send GET request to server, receive data
      #executing glicko_algo
      #calculate new rating and RD
      payload={"rating":{"agent":"key"}}

      r=requests.get(url, data=json.dumps(payload),
headers=headers)
      providers=r.content
      s=providers.decode('utf-8')
      providers=json.loads(s)

      ratings={}
      client_temp={}
      for provider in providers:
            data_c=providers[provider]
            provider,r_p,RD_p=provider.split("_")
            RD_p=float(RD_p)
            r_p=int(r_p)
            client_temp=convert_dict(data_c,RD_p,r_p,provider)
            for key in client_temp:
                  if key in my_new:
                        my_new[key].append(client_temp[key][0])
                  else:
                        my_new[key]=client_temp[key]

      for key in my_new:
            if key in providers:
                  providers[key].append(my_new[key][0])
            else:
                  providers[key]=my_new[key]

      for provider in providers:
            data_c=providers[provider]
            provider,r_p,RD_p=provider.split("_")
            RD_p=float(RD_p)
            r_p=int(r_p)
            #ratings dict has new ratings and RDs for each provider
            ratings[provider]=count_providers(data_c,RD_p,r_p)
      print(ratings)

      #Send Patch request to server to update database
      for provider in ratings:
            payload={}
            payload={"rating":{"update":{"provider":provider,\
            "rating":ratings[provider]['rating'],\
            "RD":ratings[provider]['RD']}}}
            r=requests.patch(url, data=json.dumps(payload),
headers=headers)
            print(r.content)

      #executing rating system periodically
if __name__ == "__main__":
      timeout=3600 #3600 sec
      l=task.LoopingCall(main_prog)
      l.start(timeout)
      reactor.run()
```

Στη μέθοδο main_prog η επικοινωνία με το server γίνεται με χρήση της βιβλιοθήκης requests και συγκεκριμένα με τη μέθοδο requests.() ανάλογα με τη μορφή του HTTP αιτήματος που θέλουμε να στείλουμε. Στη συνέχεια, εισάγουμε στη συνάρτηση τις μεταβλητές HEADERS που καθορίζουν το περιεχόμενο του αιτήματος και payload που ορίζει τα δεδομένα προς αποστολή. Αρχικά, γίνεται εκκίνηση τις περιόδου, καθορίζοντας την παράμετρο c με αποστολή αιτήματος request.put(). Έπειτα, στέλνεται αίτημα request.get() για λήψη του λεξικού με όλες τις συναλλαγές της περιόδου και μετατροπή του λεξικού αυτού με την convert_dict() η οποία δημιουργεί και τα ζεύγη πελάτης-πάροχος για υπολογισμό των νέων βαθμολογιών όχι μόνο των παρόχων αλλά και των χρηστών με βάση τα αιτήματα για πόρους που έχουν πραγματοποιήσει. Αφού έχει δημιουργηθεί το τελικό λεξικό για επεξεργασία καλείται η συνάρτηση count_providers() που υπολογίζει τις νέες βαθμολογίες με βάση τις σχέσεις που έχουν περιγραφεί στο κεφάλαιο της σχεδίασης. Τέλος, στέλνονται επαναληπτικά αιτήματα request.patch() για ανανέωση όλων των βαθμολογιών και αποκλίσεων βαθμολογίων όλων των χρηστών. Το σύστημα Rating_agent εκτελείται περιοδικά ανά 1 ώρα που όπως έχει οριστεί για τις ανάγκες της διπλωματικής είναι η διάρκεια μίας περιόδου βαθμολόγησης.

5.3 Μετρήσεις

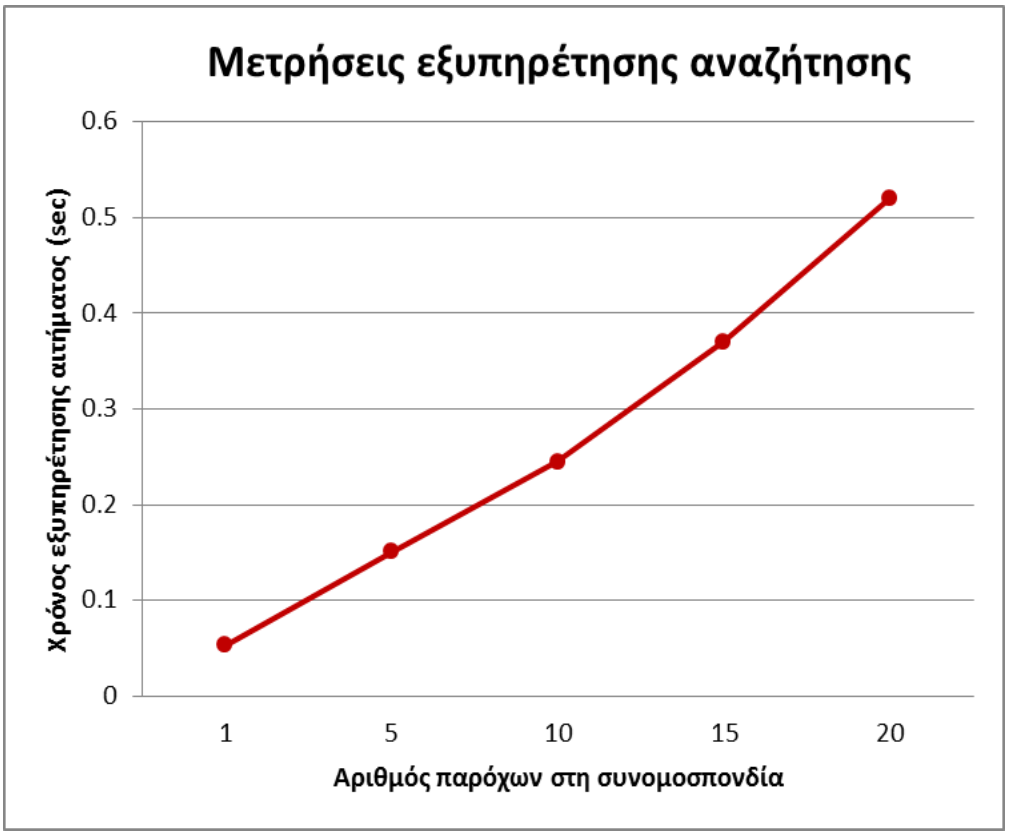
Θεωρούμε ότι η μεγάλη πλειοψηφία των αιτημάτων που δέχεται η πλατφόρμα είναι αιτήματα για αναζήτηση υπηρεσιών. Πραγματοποιήσαμε, συνεπώς, μελέτη της απόδοσης της πλατφόρμας μας ως προς το χρόνο ολοκλήρωσης αιτημάτων αναζήτησης, για μεγάλο αριθμό δεδομένων και για διαφορετικό αριθμό παρόχων σε κάθε περίπτωση για να μπορέσουμε να ελέγξουμε την κλιμακωσιμότητα της πλατφόρμας μας και την δυνατότητα της να ανταπεξέλθει σε πραγματικές συνθήκες λειτουργίας.

Οι μετρήσεις πραγματοποιήθηκαν με το εξής σκεπτικό:

Αρχικά, γνωρίζουμε ότι ένας μέσος πάροχος προσφέρει στην αγορά των υπηρεσιών ως υποδομών μερικές χιλιάδες εικονικές μηχανές. Θεωρούμε τη χειρότερη περίπτωση, στην οποία διαθέτει το σύνολο των εικονικών μηχανών του για χρήση στη συνομοσπονδία νεφών. Για τη πραγματοποίηση των μετρήσεων χρησιμοποιήθηκε το σύνολο των υπηρεσιών που προσφέρει η Amazon Web Services (AWS) σε ιδιώτες. Προέκυψε, λοιπόν, ένας πάροχος με δεδομένα που περιγράφουν μονοσήμαντα 3318 εικονικές μηχανές, στον οποίο γίνεται και το αρχικό αίτημα για αναζήτηση. Το αίτημα αφορά μία συγκεκριμένη εικονική μηχανή, με όλα τα δυνατά πεδία που μπορούν να συμπληρωθούν (αναγκαία και προεραιτικά πεδία).

Ακολουθούν μετά μετρήσεις με 5 εγγεγραμμένους παρόχους στη συνομοσπονδία, οι οποίοι προσφέρουν ακριβώς τις ίδιες υπηρεσίες, υλοποιώντας το χειρότερο δυνατό σενάριο για αναζήτηση, αφού όλοι οι πάροχοι ικανοποιούν το αίτημα. Στο ίδιο μοτίβο πραγματοποιούνται μετρήσεις για 10, 15 και 20 παρόχους. Προέκυψαν οι παρακάτω μετρήσεις:

Αριθμός Παρόχων	Συνολικός αριθμός μοναδικών Εικονικών Μηχανών	Μέτρηση 1	Μέτρηση 2	Μέτρηση 3	Μέτρηση 4	Μέτρηση 5	Μέσος όρος μετρήσεων
1	3318	0.044 sec	0.055 sec	0.055 sec	0.058 sec	0.054 sec	0.0532 sec
5	16590	0.161 sec	0.146 sec	0.145 sec	0.156 sec	0.156 sec	0.1508 sec
10	33180	0.261 sec	0.249 sec	0.251 sec	0.233 sec	0.233 sec	0.2448 sec
15	49770	0.361 sec	0.384 sec	0.36 sec	0.351 sec	0.351 sec	0.37 sec
20	66360	0.49 sec	0.568 sec	0.564 sec	0.47 sec	0.47 sec	0.5204 sec



Παρατηρούμε ότι ο χρόνος που χρειάζεται για την εξυπηρέτηση ενός αιτήματος αναζήτησης έχει σχεδόν γραμμική εξάρτηση με τον αριθμό των παρόχων, και κατ' επέκταση με τον αριθμό των μοναδικών εικονικών μηχανών, που υπάρχουν στη συνομοσπονδία.

Διευκρινίζεται ότι κάθε εικονική μηχανή περιγράφεται από 12 πεδία, από τα οποία μόνο τα 7 είναι μοναδικά για κάθε εικονική μηχανή, λόγω της χρήσης «φωλιασμένων» εγγράφων. Επομένως, ο αριθμός των πεδίων που προσπελάσονται ανά εικονική μηχανή είναι μεταβλητός και ένα υποσύνολο αυτών εξαρτάται άμεσα από το περιεχόμενο του αιτήματος. Επίσης, σε πραγματικές συνθήκες, είναι σπάνιο ένα αίτημα να ικανοποιείται από μεγάλο αριθμό παρόχων ταυτόχρονα, καθώς οι πάροχοι δεν προσφέρουν πανομοιότυπες υπηρεσίες, μειώνοντας επιπλέον, με αυτόν τον τρόπο, το εύρος των δεδομένων που προσπελάσονται.

Οι παραπάνω μετρήσεις, συνεπώς, προσομοιώνουν τις χειρότερες δυνατές συγκυρίες που μπορούν να προκύψουν για την εξυπηρέτηση ενός αιτήματος αναζήτησης. Ακόμα και σε αυτές τις συγκυρίες, ο χρόνος εξυπηρέτησης κρίνεται ικανοποιητικός και έχει καλά περιθώρια κλιμακωσιμότητας ως προς το εύρος των παρόχων και των εικονικών μηχανών που προσφέρουν. Τέλος, σε πραγματικές συνθήκες, αναμένεται ότι ο χρόνος εξυπηρέτησης αναζήτησης θα έχει βελτιωμένη συμπεριφορά σε σχέση με τα αποτελέσματα που προέκυψαν παραπάνω.

5.4 Παράδειγμα χρήσης

Στο κεφάλαιο αυτό θα περιγραφεί ένα σενάριο χρήσης της πλατφόρμας, μέσω του οποίου θα γίνει ευκολότερη η κατανόηση των λειτουργιών και της χρησιμότητας της. Θα αναλυθεί ο τρόπος με τον οποίο γίνεται η επικοινωνία των διαφόρων χρηστών με την πλατφόρμα αλλά και ο τρόπος με τον οποίο κάθε χρήστης μπορεί να την χρησιμοποιήσει στα πλαίσια των δικών του αναγκών. Σε αυτό το σενάριο λοιπόν θα εμπλακούν πέντε χρήστες από τους οποίους οι τέσσερις θα έχουν την ιδιότητα του παρόχου ενώ ο πέμπτος θα έχει την ιδιότητα του πελάτη-καταναλωτή. Ο ρόλος των χρηστών που θα συμμετάσχουν ως πάροχοι στη συνομοσπονδία είναι η παροχή υπολογιστικών πόρων στο σύστημα ενώ ο ρόλος του πελάτη θα είναι να στείλει αίτημα για αναζήτηση διαθέσιμων πόρων στην συνομοσπονδία με χαρακτηριστικά που θα ορίζονται μέσω του αιτήματος.

Αρχικά, θα πρέπει όλοι οι χρήστες να είναι εγγεγραμμένοι στην πλατφόρμα για να αποκτήσουν δικαίωμα χρήσης της. Θεωρούμε ότι οι τρεις πάροχοι είναι συνδεδεμένοι ήδη στην πλατφόρμα και έχουν εισάγει ήδη τους πόρους τους στην βάση δεδομένων. Ο τέταρτος πάροχος και ο πελάτης δεν έχουν ξανασυμμετάσχει στην πλατφόρμα με συνέπεια να

χρειάζεται η εγγραφή τους στην πλατφόρμα. Αφού γίνει αυτό και τους επιστραφεί αδειοδοτικό χρήσης (token), ο νέος χρήστης που θα εκτελέσει και αυτός χρέη παρόχου, στο συγκεκριμένο σενάριο, θα εισάγει τα χαρακτηριστικά των πόρων που θα διαθέσει στην συνομοσπονδία. Για τις ανάγκες του συγκεκριμένου σεναρίου θεωρούμε ότι οι τρεις εγγεγραμμένοι χρήστες έχουν πραγματοποιήσει παλαιότερες συναλλαγές, οι οποίες χρησιμοποιήθηκαν για τον προσδιορισμό των καινούργιων τιμών των βαθμολογιών τους και των αποκλίσεων τους. Σημειώνεται ότι, ο τέταρτος πάροχος και ο πελάτης, με την εγγραφή τους στο σύστημα ως νέοι χρήστες, απέκτησαν τις τυποποιημένες βαθμολογίες για χρήστες που εγγράφονται πρώτη φορά.

Στη συνέχεια, ο πελάτης διαμορφώνει το αίτημα με τους πόρους που αναζητεί στην συνομοσπονδία και επικοινωνεί με την πλατφόρμα. Η πλατφόρμα είναι υπεύθυνη για τον εντοπισμό στη βάση δεδομένων της των παρόχων που μπορούν να εξυπηρετήσουν το αίτημα του πελάτη. Ταυτόχρονα, στην περίπτωση εύρεσης πολλαπλών παρόχων, θα εφαρμοστεί μηχανισμός επιλογής ενός μοναδικού παρόχου, ο οποίος θα βρίσκεται πιο κοντά στην ζώνη αξιοπιστίας του πελάτη.

Τέλος, αφού επιτευχθεί συμφωνία πελάτη-παρόχου και ολοκληρωθεί η συναλλαγή θα εκτελεστεί το βαθμολογικό σύστημα για την ανανέωση των βαθμολογιών των παρόχων με βάση τις συναλλαγές που εκτέλεσαν στην τρέχουσα βαθμολογική περίοδο.

5.4.1 Εκτέλεση σεναρίου

Το σενάριο του οποίου έγινε περιγραφή πιο πάνω αφορά πέντε χρήστες της εφαρμογής. Υποθέτουμε ότι τα ονόματα τους είναι prov1, prov2, prov3 prov4 και client. Εκτελώντας το σενάριο με αυτούς τους χρήστες θα μία προσπάθεια καλύτερης κατανόησης και περιγραφής των λειτουργιών και της χρησιμότητας της πλατφόρμας.

Αρχικά λοιπόν, θα στείλει αίτημα εγγραφής στην πλατφόρμα ο πάροχος prov4, ο οποίος δεν είχε χρησιμοποιήσει την πλατφόρμα στο παρελθόν. Το αίτημα που στέλνεται στην πλατφόρμα είναι τύπου HTTP POST και έχει τα δεδομένα το όνομα χρήστη prov4 και τον κωδικό πρόσβασης pass4, κωδικοποιημένα σε Base64 στην επικεφαλίδα Authorization. Έχει προορισμό το υποσύστημα *Root* που διαχειρίζεται την καταχώρηση νέων χρηστών στη συνομοσπονδία. Το αίτημα παρουσιάζεται παρακάτω:

```
POST /root HTTP/1.1
Host: 127.0.0.1:5000
Authorization: Basic cHJvdjE6cGFzczE=
User-Agent: curl/7.47.0
```

```
Accept: */*
```

Το συγκεκριμένο αίτημα δημιουργήθηκε από τη γραμμή εντολών μέσω του εργαλείου curl ως εξής:

```
curl -u prov1:pass1 -X POST http://127.0.0.1:5000/root
```

Όλα τα αιτήματα από τη πλευρά του χρήστη που ακολουθούν, μπορούν να δημιουργηθούν από γραμμή εντολών με παρόμοιο τρόπο.

Ο νέος χρήστης, λοιπόν, λαμβάνει απάντηση από το υποσύστημα *Root* για την επιτυχή εγγραφή του, μαζί με το αδειοδοτικό χρήσης (token), που είναι απαραίτητο για να έχει πρόσβαση στις υπηρεσίες της πλατφόρμας.

```
HTTP/1.0 201 CREATED
Content-Type: application/json
Content-Length: 179
Server: Werkzeug/0.12.2 Python/3.5.2
Date: Wed, 11 Oct 2017 13:59:03 GMT

{
  "message": "Provider prov4 joined.",
  "token":
"eyJpYXQiOjE1MDc3MzAzNDMsImV4cCI6MTUwNzc0ODM0MywiYWxnIjoiSFMyNTYif
Q.InByb3YzIg._r-BzaWKbb8wFYnkZHeN45Dyg-STpOcFg8L5UtvSadc"
}
```

Στη συνέχεια, ο χρήστης prov4 θα στείλει αιτήματα POST στο υποσύστημα *Records* για να καταχωρήσει τις υπηρεσίες που διαθέτει για χρήση από τη συνομοσπονδία. Τα αιτήματα έχουν τη παρακάτω μορφή:

```
POST /update HTTP/1.1
Host: 127.0.0.1:5000
User-Agent: curl/7.47.0
Accept: */*
Authorization: Token
eyJpYXQiOjE1MDc3MzAzNDMsImV4cCI6MTUwNzc0ODM0MywiYWxnIjoiSFMyNTYifQ
.InByb3YzIg._r-BzaWKbb8wFYnkZHeN45Dyg-STpOcFg8L5UtvSadc
Content-type: application/json
Content-Length: 275

{
  "res_attr": {
    "name": "prov4",
    "location": "EU(London)",
    "OS": "Red Hat Enterprise Linux",
```

```
    "optimization": "Compute Optimized",
    "productFamily": "c4",
    "instanceSize": "xlarge",
    "vCPU": "4",
    "memory": "7.5",
    "storageInstance": "EBS",
    "price": "0.312",
    "availability": "63"
  }
}
```

Η δημιουργία του παραπάνω αιτήματος από τη γραμμή εντολών γίνεται πάλι μέσω του εργαλείου curl, όπου δίνονται ως όρισμα ο τύπος του HTTP αιτήματος (-X POST), ως ορίσματα επικεφαλίδων τα πεδία Authorization (-H "Authorization: Token *token*") και Content-type (-H "Content-type: application/json"), ως όρισμα περιεχομένου το αρχείο JSON (-d '*json_file*') που περιγράφει τα χαρακτηριστικά της εικονικής μηχανής που καταχωρείται στην πλατφόρμα ως διαθέσιμο για χρήση από τη συνομοσπονδία και τέλος ως όρισμα προορισμού το URL του υποσυστήματος *Records* http://127.0.0.1:5000/update.

Για κάθε υπηρεσία που καταχωρεί ο χρήστης στη πλατφόρμα, το υποσύστημα απαντά κατάλληλα στο χρήστη:

```
HTTP/1.0 201 CREATED
Content-Type: application/json
Content-Length: 72
Server: Werkzeug/0.12.2 Python/3.5.2
Date: Wed, 11 Oct 2017 14:23:02 GMT

{
  "message": "Resource attributes from provider prov4 registered."
}
```

Ο νέος χρήστης client, κάνει και αυτός εγγραφή στη πλατφόρμα και καταχωρεί τους δικούς του πόρους προς χρήση από τη συνομοσπονδία. Έπειτα, θέλει να εμπλουτίσει τις υπηρεσίες που μπορεί να προσφέρει, ως πάροχος στους πελάτες του, χρησιμοποιώντας πόρους που είναι διαθέσιμοι από τη συνομοσπονδία. Συνεπώς, στέλνει αίτημα στη πλατφόρμα για αναζήτηση κατάλληλου παρόχου που θα μπορέσει να του προσφέρει τους πόρους που χρειάζεται.

Θέλει να εμπλουτίσει τους πόρους του, λοιπόν, με δύο εικονικές μηχανές που έχουν τα εξής χαρακτηριστικά:

Εικονική μηχανή 1:

- Τοποθεσία υποδομών: Ευρώπη (Λονδίνο)
- Πλήθος στιγμιοτύπων: 50
- Λειτουργικό σύστημα εικόνας: Windows
- Πρότυπο βελτιστοποίησης: Βελτιστοποίηση ως προς τη μνήμη
- Αριθμός εικονικών ΚΜΕ: 4
- Μέγεθος εικονικής μνήμης: 15.25
- Τιμή σε € ανά ώρα χρήσης: 0,4

Εικονική μηχανή 2:

- Τοποθεσία υποδομών: Ευρώπη (Λονδίνο)
- Πλήθος στιγμιοτύπων: 50
- Λειτουργικό σύστημα εικόνας: Red Hat Enterprise Linux
- Οικογένεια προϊόντος: t2
- Κατηγορία εικονικής μηχανής: xlarge
- Τιμή σε € ανά ώρα χρήσης: 0,3

Ο χρήστης θεωρεί ότι για την πρώτη εικονική μηχανή τα απαραίτητα χαρακτηριστικά για την επιτυχία της αναζήτησης είναι η τοποθεσία υποδομών, το μέγεθος της εικονικής μνήμης και το λειτουργικό σύστημα εικόνας, ενώ θεωρεί προεραιτικά με προτεραιότητα την τιμή και έπειτα τον αριθμό των εικονικών ΚΜΕ. Για τη δεύτερη εικονική μηχανή θεωρεί ότι όλα τα χαρακτηριστικά είναι απαραίτητα για την εύρεση κατάλληλου παρόχου.

Συνεπώς, διαμορφώνει και στέλνει στο υποσύστημα *Matchmaking-agent* το παρακάτω HTTP αίτημα:

```
GET /search HTTP/1.1
Host: 127.0.0.1:5000
User-Agent: curl/7.47.0
Accept: */*
Authorization: Token
eyJpYXQiOjE1MDc3MzE3NTgsImV4cCI6MTUwNzc0OTc1OCwiYWxnIjoiSFMyNTYifQ
.InByb3YxIg.3kLS-SxRVl_scar_sMXWfZa_R3dvTmK0_sK0oU10bV0
Content-type: application/json
Content-Length: 340

{
  "search": {
    "1": {
      "optimization": "Memory Optimized",
      "VMnum": "50",
      "mandatory": {
        "location": "EU(London)",
```

```
        "OS": "Windows",
        "memory": "8"
      },
      "optional": {
        "1.price": "0.4",
        "2.vCPU": "4"
      }
    },
    "2": {
      "productFamily": "t2",
      "instanceSize": "xlarge",
      "VMnum": "50",
      "mandatory": {
        "location": "EU(London)",
        "OS": "Red Hat Enterprise Linux",
        "memory": "2",
        "price": "0.3"
      }
    }
  }
}
```


Ας υποθέσουμε τώρα ότι οι υπόλοιποι χρήστες έχουν καταχωρήσει στη βάση δεδομένων της πλατφόρμας τις εξής υπηρεσίες:

prov1


```
{
"_id" : "prov1",
  "locations" : [
    {
      "region" : "US East(N. Virginia)",
      "instances" : [
        {
          "OS" : "Red Hat Enterprise Linux",
          "instanceType" : [
            {
              "optimization" : "General Purpose",
              "productFamily" : "t2",
              "instanceSize" : "large",
              "details" : {
                          "price" : 0.1528,
                          "vCPU" : 2,
                          "memory" : 8.0,
                          "availability" : 73,
                          "storageInstance" : "EBS"
            }
          }
        ]
      }
    ]
  }
]
}
```

prov2


```
{
"_id" : "prov2",
  "locations" : [
    {
      "region" : "EU(London)",
      "instances" : [
        {
          "OS" : "Red Hat Enterprise Linux",
          "instanceType" : [
            {
              "optimization" : "General Purpose",
              "productFamily" : "t2",
              "instanceSize" : "xlarge",
              "details" : {
                            "price" : 0.2712,
                            "vCPU" : 4,
                            "memory" : 16.0,
                            "availability" : 68,
                            "storageInstance" : "EBS"
              }
            }
          ]
        },
          "OS" : "Windows",
          "instanceType" : [
            {
              "optimization" : "Memory Optimized",
              "productFamily" : "r4",
              "instanceSize" : "large",
              "details" : {
                            "price" : 0.248,
                            "vCPU" : 2,
                            "memory" : 15.25,
                            "availability" : 73,
                            "storageInstance" : "EBS"
              }
            }
          ]
        }
      ]
    }
  ]
}
```


prov3


```
{
"_id" : "prov3",
  "locations" : [
    {
      "region" : "US East(N. Virginia)",
      "instances" : [
        {
          "OS" : "Windows",
          "instanceType" : [
            {
              "optimization" : "Memory Optimized",
              "productFamily" : "r4",
```

```
                "instanceSize" : "xlarge",
                "details" : {
                                "price" : 0.378,
                                "vCPU" : 4,
                                "memory" : 30.5,
                                "availability" : 52,
                                "storageInstance" : "EBS"
                }
              }
            ]
          }
        ]
      }
    ]
}
```


prov4


```
{
"_id" : "prov4",
  "locations" : [
    {
      "region" : "EU(London)",
      "instances" : [
        {
          "OS" : "Red Hat Enterprise Linux",
          "instanceType" : [
            {
              "optimization" : "General Purpose",
              "productFamily" : "t2",
              "instanceSize" : "xlarge",
              "details" : {
                              "price" : 0.2966,
                              "vCPU" : 4,
                              "memory" : 16.0,
                              "availability" : 94,
                              "storageInstance" : "EBS"
              }
            }
          ]
        },
          "OS" : "Windows",
          "instanceType" : [
            {
              "optimization" : "Memory Optimized",
              "productFamily" : "r4",
              "instanceSize" : "large",
              "details" : {
                              "price" : 0.235,
                              "vCPU" : 2,
                              "memory" : 15.25,
                              "availability" : 121,
                              "storageInstance" : "EBS"
              }
            }
          ]
        }
      ]
    }
```

```
  ]
}
```

Επίσης, οι χρήστες έχουν τις εξής βαθμολογίες στο σύστημα glicko:

Χρήστης	Rating	RD
prov1	1600	80
prov2	1200	60
prov3	1450	100
prov4	1500	150
client	1500	150

Αρχικά, ο *Matchmaking_agent* επιβεβαιώνει ότι όλοι οι χρήστες βρίσκονται στη ζώνη αξιοπιστίας του χρήστη client μέσω των βαθμολογιών τους. Προχωρώντας στην αναζήτηση στη βάση δεδομένων, απορρίπτει κατευθείαν το χρήστης prov1, καθώς δεν έχει καταχωρήσεις στη τοποθεσία του Λονδίνου. Επίσης, ο prov3 ικανοποιεί μόνο το αίτημα για την πρώτη εικονική μηχανή, οπότε απορρίπτεται και αυτός. Οι χρήστες prov2 και prov4 μπορούν να καλύψουν τα υποχρεωτικά χαρακτηριστικά για την πρώτη εικονική μηχανή και το πρώτο προεραιτικό (τιμή) και δεν καλύπτουν το δεύτερο (αριθμός εικονικών ΚΜΕ), ενώ καλύπτουν το σύνολο των χαρακτηριστικών για τη δεύτερη εικονική μηχανή. Συνεπώς, οι prov2 και prov4 ικανοποιούν και οι δυο τους το αίτημα του χρήστη client. Για να μειωθούν οι κατάλληλοι πάροχοι σε έναν, ο *Matchmaking-agent* χρησιμοποιεί το αλγόριθμο glicko, με αποτέλεσμα να επιλεχθεί τελικά ο prov2. Το αποτέλεσμα του αλγορίθμου Glicko φαντάζει λανθασμένο με μία πρώτη ματιά καθώς η βαθμολογία τόσο του client όσο και του prov4 είναι 1500. Όμως, για να επιλεγεί ο κατάλληλος πάροχος για την εξυπηρέτηση του αιτήματος χρησιμοποιείται και η απόκλιση βαθμολογίας του πελάτη που μας παρέχει πληροφορίες για την αξιοπιστία της βαθμολογίας του. Γνωρίζοντας, λοιπόν, ότι η πραγματική βαθμολογία του πελάτη κυμαίνεται στα όρια [1200, 1800] επιλέγεται ο πάροχος με βαθμολογία πιο κοντά στο κατώτερο άκρο

Το υποσύστημα, λοιπόν, αφού βρήκε κατάλληλη αντιστοιχία παρόχου ικανού να ικανοποιήσει το αίτημα του χρήστη client, δημιουργεί νέα καταχώρηση συναλλαγής στη λίστα συναλλαγών και απαντά στο χρήστη με κατάλληλο μήνυμα που περιλαμβάνει την ταυτότητα του παρόχου που μπορεί να του προσφέρει τις υπηρεσίες που ζήτησε, το URL επικοινωνίας με το πάροχο και το αναγνωριστικό της συναλλαγής:

```
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 148
Server: Werkzeug/0.12.2 Python/3.5.2
```

```
Date: Wed, 11 Oct 2017 14:38:06 GMT

{
  "Message": "Match found.",
  "Provider name": "prov2",
  "Transaction id": "59de2cce1bbcc71ec813570c",
  "URL": "http://intercloud/prov2/"
}
```

Αξίζει να σημειωθεί τέλος, ότι τα σενάρια χρήσης της πλατφόρμας ως προς την αναζήτηση πόρων ποικίλουν. Για παράδειγμα, θα μπορούσε ο χρήστης prov4 να ικανοποιούσε και το δεύτερο προαιρετικό χαρακτηριστικό (αριθμός εικονικών ΚΜΕ 4) για την πρώτη εικονική μηχανή, με αποτέλεσμα να κριθεί καταλληλότερος του χρήστη prov2, χωρίς να χρησιμοποιηθεί ο αλγόριθμος glicko. Όσο αυξάνεται ο αριθμός των παρόχων και των υπηρεσιών που προσφέρουν τόσο αυξάνονται και τα πιθανά σενάρια αναζήτησης. Για λόγους χώρου και ευανάγνωσης της εργασίας, όμως, επιλέχθηκε να παρουσιαστεί ένα απλό και αρκετά αντιπροσωπευτικό παράδειγμα χρήσης, δίνοντας μία καλή εικόνα για τα δυνατά σενάρια που μπορούν να προκύψουν.

6

Επίλογος

6.1 Συμπεράσματα

Κατά τη συγγραφή της διπλωματικής εργασίας και την υλοποίηση της πλατφόρμας ασχοληθήκαμε με πολλές τεχνολογίες και μέσω της ενασχόλησης μας με αυτές παρήχθησαν κάποια χρήσιμα συμπεράσματα.

Η διαδικασία ανάπτυξης μιας πλατφόρμας απαιτεί την ανάλυση πολλών διαφορετικών διαδικασιών και πρέπει να πληροί πολλές προδιαγραφές για να θεωρηθεί επιτυχημένη. Ένα από τα σημαντικότερα κριτήρια είναι η λειτουργικότητα της. Οι χρήστες έχουν την απαίτηση όχι μόνο να προσφέρονται αρκετές διαφορετικές λειτουργίες αλλά και αυτές οι λειτουργίες να είναι αποδοτικές. Κρίνεται απαραίτητο λοιπόν, η πλατφόρμα να μπορεί να δέχεται πολλά διαφορετικά αιτήματα αλλά και να τα επεξεργάζεται γρήγορα για την καλή εξυπηρέτηση των χρηστών. Επίσης, θα πρέπει να προσφέρεται στους χρήστες ασφάλεια των δεδομένων και των ευαίσθητων πληροφοριών τους. Όλα τα παραπάνω απαιτούν ανάλυση και είναι κρίσιμα κομμάτια της ανάπτυξης μιας εφαρμογής.

Επίσης, αναπτύσσοντας μία πλατφόρμα υπεύθυνη για την εξυπηρέτηση παρόχων υπηρεσιών νέφους ήρθαμε αντιμέτωποι με το πρόβλημα της διαφορετικής απεικόνισης των προσφερόμενων πόρων. Για την περάτωση της διπλωματικής εργασίας κρίθηκε απαραίτητη η προτυποποίηση της απεικόνισης των πόρων αυτών, ώστε οι χρήστες να μπορούν να χρησιμοποιούν μία κοινή βάση επικοινωνίας με την πλατφόρμα χωρίς, όμως, βλάβη της γενικότητας. Με αυτό τον τρόπο επιτυγχάνεται η δημιουργία μίας πλατφόρμας κατάλληλης για χρήση από πολλούς διαφορετικούς παρόχους με μοναδική προϋπόθεση την συμμόρφωση των χρηστών με τα συγκεκριμένα πρότυπα επικοινωνίας για καλύτερη εξυπηρέτηση τους.

Κατά τη διαδικασία σχεδίασης της πλατφόρμας, ασχοληθήκαμε και με το θέμα δίκαιης αντιμετώπισης των χρηστών. Μετά από διαφορετικές αναζητήσεις στη βάση δεδομένων της πλατφόρμας μας και μελέτη των υπηρεσιών που παρέχονται από τους μεγαλύτερους παρόχους υπηρεσιών νέφους, διαπιστώσαμε ότι υπάρχει μια πληθώρα παρόχων που μπορούν να εξυπηρετήσουν ένα συγκεκριμένο αίτημα για πόρους. Γίνεται αντιληπτό, ότι για την καλύτερη εξυπηρέτηση των χρηστών, λύνοντας και το πρόβλημα επικοινωνίας ενός πελάτη με πολλούς διαφορετικούς παρόχους, πρέπει να γίνει επιλογή ενός και μόνο παρόχου για την κάλυψη ενός αιτήματος. Η επιλογή αυτή πρέπει να βασίζεται σε διάφορα κριτήρια ώστε να υπάρχει δίκαιη επιλογή του τελικού παρόχου, ειδικά σε ένα περιβάλλον εθελοντικό όπως αυτό της συνομοσπονδίας νεφών.

Τέλος, ασχοληθήκαμε με πολλές διαφορετικές τεχνολογίες στα πλαίσια αυτής της διπλωματικής, και ειδικά με μερικές από αυτές για πρώτη φορά. Καταλήξαμε έτσι σε κάποια χρήσιμα, κατά τη γνώμη μας, συμπεράσματα. Αρχικά, στη σημερινή εποχή με τον μεγάλο όγκο δεδομένων οι μη σχεσιακές βάσεις δεδομένων αποκτούν ολοένα και μεγαλύτερη εφαρμογή καθώς εμφανίζουν πλεονεκτήματα σε σχέση με τις κλασσικές SQL βάσεις ειδικά στον τομέα της οριζόντιας κλιμάκωσης και στον τομέα της επεξεργασίας δεδομένων κατά δέσμες, προσφέροντας έτσι νέες καλύτερες και ταχύτερες μεθόδους για την αποθήκευση, αναζήτηση και επεξεργασία δεδομένων . Ταυτόχρονα, σε μία εποχή που η ανταλλαγή δεδομένων μεταξύ συστημάτων γίνεται με χρήση μηνυμάτων συνήθως σε μορφή δυαδική, XML ή JSON, η δυνατότητα διαχείρισης αυτών των μορφών εσωτερικά από τις NoSQL βάσεις, μειώνει το πλήθος του κώδικα για μετατροπή σε μορφή κατάλληλη για αποθήκευση ενώ ταυτόχρονα μειώνει τα έξοδα συντήρησης του. Επιπροσθέτως, διαπιστώσαμε ότι η χρήση του πλαισίου ιστού Flask έχει αρκετά πλεονεκτήματα σε σχέση με άλλα μεγάλα πλαίσια ιστού. Το βασικότερο πλεονέκτημα του είναι η προσαρμοστικότητα του καθώς μπορεί εύκολα να προσαρμοστεί σε νέες ανάγκες και τεχνολογίες με τη χρήση των επεκτάσεων που προσφέρονται. Σε σύγκριση με άλλα μεγαλύτερα πλαίσια, αυτές οι επεκτάσεις δεν είναι μέρος του πυρήνα του Flask και ο χρήστης είναι ελεύθερος να διαλέξει μεταξύ αυτών που χρειάζεται. Ταυτοχρόνως, προσφέρει ένα πολύ καλό σύνολο τεκμηρίωσης κάνοντας το ένα πλαίσιο εύκολο στη χρήση για γρήγορη ανάπτυξη εφαρμογών ενώ ο κάθε χρήστης μπορεί να συνεισφέρει στην ανάπτυξη του ίδιου του πλαισίου αφού πλέον χρησιμοποιείται το GitHub. Καταλήγοντας, παρόλο που το Flask είναι ένα αρκετά νέο εργαλείο για το οποίο δεν υπάρχουν πολλά παραδείγματα σε πραγματικές συνθήκες, προτείνεται η χρήση του καθώς ο προγραμματιστής μπορεί να κατανοήσει καλύτερα πως τα διαφορετικά κομμάτια συνδέονται μεταξύ τους, αποφεύγοντας την ύπαρξη αχρησιμοποίητων λειτουργιών.

6.2 Μελλοντικές επεκτάσεις

Η πλατφόρμα που αναπτύχθηκε στα πλαίσια της διπλωματικής εργασίας κρίνεται ως χρήσιμη και εύχρηστη καθώς μπορεί να χρησιμοποιηθεί από παρόχους για το διαμοιρασμό των πόρων που θέλουν να προσφέρουν στη συνομοσπονδία. Για να συνεχίσει να είναι λειτουργική και να εξυπηρετεί καλύτερα τους χρήστες της, υπάρχουν νέα χαρακτηριστικά τα οποία θα μπορούσαν να προστεθούν. Επίσης, υπάρχει η ανάγκη διόρθωσης τυχόν σφαλμάτων που προκύπτουν κατά τη χρήση της πλατφόρμας και να παρέχονται οι απαραίτητες αναβαθμίσεις όποτε αυτό είναι απαραίτητο. Θα αναφερθούμε παρακάτω σε ορισμένες μελλοντικές επεκτάσεις και διορθώσεις που θα μπορούσαν να γίνουν.

Αρχικά, όντας σε δοκιμαστικό στάδιο η πλατφόρμα μας δεν προσφέρει στο χρήστη ένα γραφικό περιβάλλον επιτρέποντας την επικοινωνία μόνο μέσω απευθείας HTTP αιτημάτων. Για την ευκολότερη χρήση της λοιπόν, μία μελλοντική επέκταση θα μπορούσα να είναι η ανάπτυξη γραφικού περιβάλλοντος που θα χρησιμοποιείται από τους χρήστες για εγγραφή και αναζήτηση στην συνομοσπονδία νεφών.

Μια ακόμα προσθήκη χρήσιμη για την καλύτερη εξυπηρέτηση των χρηστών είναι η προσθήκη ακριβών γεωγραφικών δεδομένων των παρόχων. Παρόλο που η τοποθεσία είναι ένα δεδομένο το οποίο χρησιμοποιείται ήδη στην πλατφόρμα μας μπορούμε να προεκταθούμε στο συγκεκριμένο κομμάτι αναζητώντας όχι μόνο συγκεκριμένες τοποθεσίες αλλά να γίνεται αναζήτηση παρόχων σε ένα εύρος τοποθεσιών κοντά στην αρχική αναζήτηση του χρήστη. Βέβαια, θα πρέπει να υπάρξει και συσχέτιση της γεωγραφικής τοποθεσίας με το χρόνο καθυστέρησης επικοινωνίας μεταξύ των παρόχων και για να προκύψουν ορθά αποτελέσματα.

Επίσης, το σύστημα επιλογής του κατάλληλου χρήστη μέσω της βαθμολογίας του εξετάζει τόσο την βαθμολογία του παρόχου όσο και τη βαθμολογία του πελάτη. Η μέθοδος αυτή είναι αποδοτική αλλά δεν λαμβάνει υπόψιν της την πολυπλοκότητα του αιτήματος θεωρώντας πως κάθε αίτημα για πόρους έχει την ίδια δυσκολία εξυπηρέτησης. Στη μελλοντική επέκταση, θα μπορούσε να εξεταστεί η πολυπλοκότητα του αιτήματος, βαθμολογώντας κατάλληλα το αίτημα, ενώ εν συνεχεία η επιλογή του κατάλληλου παρόχου θα γίνεται με βάση την βαθμολογία του αιτήματος, αντιστοιχίζοντας ένα πολύπλοκο αίτημα σε ένα πάροχο με υψηλή βαθμολογία, δηλαδή με καλή ποιότητα υπηρεσιών και συχνή συμμετοχή στη συνομοσπονδία.

Μία ακόμα επέκταση που θα μπορούσε να γίνει είναι προς την κατεύθυνση της διεύρυνσης των υπηρεσιών που προσφέρει η πλατφόρμα. Η πλατφόρμα έχει διαμορφωθεί για την εξυπηρέτηση αναγκών σε υποδομές ως υπηρεσία (IaaS), αλλά υπάρχουν ακόμα αρκετές λειτουργίες που μπορούν να προστεθούν. Η βασικότερη από αυτές είναι η ενσωμάτωση λογισμικού το οποίο ο χρήστης να μπορεί να εκτελεί κατ' απαίτηση, επεκτείνοντας έτσι την συνομοσπονδία στο διαμοιρασμό όχι μόνο υπολογιστικών πόρων αλλά και υπηρεσιών λογισμικού (SaaS), ένα από τα βασικότερα μοντέλα παροχής υπηρεσιών νέφους.

Τέλος, καθώς το μέγεθος των συνόλων δεδομένων που χρησιμοποιούμε στις μέρες μας αυξάνεται εκθετικά, παρουσιάζεται η ανάγκη για κατανεμημένα συστήματα και κλιμάκωση των λειτουργιών, δυνατότητα που παρέχεται ήδη από την MongoDB. Συνεπώς, θα πρέπει να τεθούν στο μέλλον σε λειτουργία απομακρυσμένοι εξυπηρετητές, σε μορφή Cluster, για την διαχείριση των πόρων και των αιτημάτων.

Ευελπιστούμε η παρούσα εργασία να ανταποκρίθηκε στον αρχικό σκοπό της και να αποτελέσει χρήσιμη βάση για μελλοντική έρευνα στον τομέα της δημιουργίας και διαχείρισης συνομοσπονδιών νέφους.

7

Βιβλιογραφία

[1] Python, https://www.python.org/

[2] MongoDB, https://www.mongodb.com/

[3] Flask, http://flask.pocoo.org/

[4] JSON (JavaScript Object Notation), http://www.json.com/

[5] Amazon EC2 Instance Types, https://aws.amazon.com/ec2/instance-types/

[6] Amazon EC2 Pricing, https://aws.amazon.com/ec2/pricing/

[7] M. E. Glickman, *The Glicko system,* http://www.glicko.net/glicko/glicko.pdf

[8] Jemal H. Abawajy: *Determining Service Trustworthiness in Intercloud Computing Environments. 10th International Symposium on Pervasive Systems, Algorithms, and Networks,* 2009

[9] *Standard for Intercloud Interoperability and Federation (SIIF),* IEEE Std. P2302, 2012

[10] Abraham Silberschatz, Henry F. Korth, S. Sudarshan, *Συστήματα Βάσεων Δεδομένων,* 2011

[11] A. Li, X. Yang, S. Kandula, and M. Zhang, *CloudCmp: comparing public cloud providers, in Proceedings of the 10th ACM SIGCOMM conference on Internet measurement (IMC '10), 2010, ACM, New York, NY, USA 1-14.*

[12] S. K. Garg, S. Versteeg and R. Buyya, "SMICloud: *A Framework for Comparing and Ranking Cloud Services," 2011 Fourth IEEE International Conference on Utility and Cloud Computing, Victoria, NSW, 2011, pp. 210-218*.

[13] Stackoverflow, http://stackoverflow.com/