ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Υλοποίηση πλατφόρμας διαχείρισης της κλιμάκωσης διαδικτυακών εφαρμογών, σε περιβάλλοντα υπολογιστικού νέφους, με χρήση εξατομικευμένων κανόνων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Αλέξανδρου Π.
Τσαντήλα

Επιβλέπων: Βαρβαρίγου Θεοδώρα
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2014

Εθνικο Μετσοβιο Πολυτεχνειο
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Υλοποίηση πλατφόρμας διαχείρισης της κλιμάκωσης διαδικτυακών εφαρμογών, σε περιβάλλοντα υπολογιστικού νέφους, με χρήση εξατομικευμένων κανόνων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Αλέξανδρου Π.
Τσαντήλα

Επιβλέπων: Βαρβαρίγου Θεοδώρα
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή στις 11/02/2014.

.....	.....	.....
Βαρβαρίγου Θεοδώρα Καθηγήτρια Ε.Μ.Π.	Λούμος Βασίλειος Καθηγητής Ε.Μ.Π.	Καγιάφας Ελευθέριος Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2014

.....

Αλέξανδρος Π. Τσαντήλας

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

Περίληψη

Σήμερα, το υπολογιστικό νέφος αναδύεται ως ένα ιδιαίτερα δημοφιλές μοντέλο και αναπτύσσεται σαν μία ενδιαφέρουσα νέα τεχνική κι επιχειρηματική λύση. Ανάμεσα στα κυριότερα χαρακτηριστικά του βρίσκεται η ελαστικότητα, η οποία του δίνει τη δυνατότητα να παρέχει ένα δυναμικό περιβάλλον διάθεσης υπηρεσιών. Αυτό το ελαστικό περιβάλλον δίνει τη δυνατότητα σε μία εφαρμογή να προσαρμόζει αυτόματα τους υπολογιστικούς πόρους που χρησιμοποιεί, ανάλογα με φόρτο εργασίας και τις προτεραιότητες που έχουν τεθεί.

Ανάμεσα στους διάφορους παρόχους υπολογιστικού νέφους, η υλοποίηση της ελαστικότητας γίνεται με ποικίλλους τρόπους. Μία προσέγγιση βασίζεται στον ορισμό κανόνων από τον πάροχο της πλατφόρμας, ενώ μία άλλη προβλέπει στατικά τους απαιτούμενους πόρους, ώστε να προσδιορίσει το μέσο φόρτο του συστήματος, κι εν συνεχεία ορίζει κανόνες. Ιδιαίτερο ενδιαφέρον παρουσιάζει η προσέγγιση παροχής της ελαστικότητας ως υπηρεσία (Elasticity-as-a-Service), δίνοντας τη δυνατότητα στους παρόχους να ορίζουν παραμέτρους ελαστικότητας που διαφέρουν από εφαρμογή σε εφαρμογή.

Έτσι, σκοπός αυτής της διπλωματικής εργασίας είναι η ανάπτυξη μιας πλατφόρμας υπολογιστικού νέφους, η οποία διαχειρίζεται αυτόματα την κλιμάκωση των υπολογιστικών πόρων διαδικτυακών εφαρμογών, εγκατεστημένες σε περιβάλλον υπολογιστικού νέφους. Η κλιμάκωση γίνεται με βάση διάφορες παραμέτρους που ορίζει ο πάροχος της κάθε εφαρμογής-υπηρεσίας και με τη βοήθεια αλγορίθμου βασισμένου σε κανόνες.

Λέξεις κλειδιά: Υπολογιστικά νέφη, ελαστικότητα, κλιμάκωση, XaaS, υπηρεσίες διαδικτύου, διακομιστής μεσολάβησης, δαίμονας, NoSQL βάσεις, mongoDB, REST, SOAP, java, apache tomcat

Abstract

Cloud computing has recently emerged as a new paradigm for hosting and delivering services over the Internet and the general interest on clouds as a technical and business solution is growing. Elasticity stands as one of the fundamental features of clouds, providing a dynamic environment of services. This elastic environment gives an application the ability to automatically adjust the infrastructure resources it uses, to accommodate varied workloads and priorities.

The implementation of the elasticity mechanism differs among the various cloud providers. One approach is based on rule criteria, defined by the application provider, while another provisions application resources statically in order to identify the ordinary traffic load and then it specifies rules. There is special interest in an innovative approach that implements elasticity as a SaaS application, which allows an application provider define some application-specific resource requirements and required QoS.

Thus, the main purpose of this thesis project is the development of a cloud platform, that automatically scales the resources of web applications deployed in a cloud computing environment. This is achieved taking into account the parameters given by the provider of each application and uses a rule-based algorithm.

Key words: Cloud computing, elasticity, scalability, XaaS, web services, proxy server, daemon, NoSQL, mongoDB, REST, SOAP, java, apache tomcat

Ευχαριστίες

Θα ήθελα να ευχαριστήσω την καθηγήτρια Θεοδώρα Βαρβαρίγου, που ήταν πάντα πρόθυμη να συζητήσει μαζί μου, που μου ανέθεσε αυτό το πολύ ενδιαφέρον θέμα και η οποία μου έδωσε την ευκαιρία να δουλέψω στο εργαστήριο Distributed Knowledge and Media Systems Group.

Θα ήθελα ιδιαίτερα να ευχαριστήσω τον υποψήφιο διδάκτορα και φίλο Παύλο Κρανά για το χρόνο του, την υπομονή του και την πολυ χρήσιμη βοήθειά του. Χαράσσοντας τις κατευθυντήριες γραμμές αυτής της εργασίας κι επιβλέποντας την εξέλιξη της, συνέβαλε καθοριστικά στην επιτυχή υλοποίηση της.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου που μου έδωσαν την ευκαιρία να σπουδάσω σε αυτό το πανεπιστημιακό ίδρυμα, στα αδέρφια μου για τις πολύτιμες συμβουλές και τη βοήθειά τους και στους φίλους που με στήριξαν σε κρίσιμες αποφάσεις.

Περιεχόμενα

Κατάλογος σχημάτων **13**

Εισαγωγή **15**
Αντικείμενο διπλωματικής 16
Οργάνωση κειμένου 18

I Θεωρητικό υπόβαθρο **19**

1 Υπολογιστικά νέφη **21**
1.1 Εισαγωγή 21
1.2 Εξέλιξη 23
1.2.1 Ιστορική αναδρομή 23
1.2.2 Σχετικές τεχνολογίες 24
1.3 Βασικά χαρακτηριστικά 25
1.3.1 Χαρακτηριστικά κατά NIST 25
1.3.2 Πρόσθετα χαρακτηριστικά 26
1.4 Μοντέλα υπηρεσιών νέφους 27
1.4.1 Υποδομή ως υπηρεσία (IaaS) 27
1.4.2 Πλατφόρμα ως υπηρεσία (PaaS) 30
1.4.3 Λογισμικό ως υπηρεσία (SaaS) 32
1.5 Μοντέλα ανάπτυξης νέφους 33
1.6 Πλεονεκτήματα 34
1.7 Ζητήματα και προκλήσεις 35

2 Η ελαστικότητα στο νέφος **43**
2.1 Ορισμός 43
2.2 Χαρακτηριστικά της ελαστικότητας 44
2.3 Προβλήματα 46
2.4 Η ελαστικότητα σε εμπορικά συστήματα 47
2.4.1 Amazon EC2 47
2.4.2 Google AppEngine 47
2.4.3 Windows Azure 48

3 Αρχιτεκτονικά μοντέλα δικτυακών συστημάτων **49**
3.1 Simple Object Access Protocol (SOAP) 49
3.1.1 Η έννοια υπηρεσιοστρεφούς αρχιτεκτονικής 50
3.1.2 Βασικές αρχές υπηρεσιοστρεφούς αρχιτεκτονικής 52
3.1.3 Υπηρεσίες SOAP 53
3.2 Representational State Transfer (REST) 56
3.2.1 Βασικές αρχές του REST 57
3.2.2 Υπηρεσίες REST 58
3.3 REST vs SOAP 61

4 NoSQL βάσεις δεδομένων **63**
4.1 Ανάγκη ανάπτυξης NoSQL βάσεων δεδομένων 63
4.2 Ταξινόμηση NoSQL βάσεων δεδομένων 65
4.2.1 Document store 66
4.2.2 Graph databases 67
4.2.3 Column-oriented databases 68
4.2.4 Key-value stores 69
4.3 MongoDB 69
4.3.1 Αρχιτεκτονική συστήματος 70
4.3.2 Αποθήκευση δεδομένων 71
4.3.3 Μοντέλο δεδομένων 74
4.3.4 Μοντέλο ερωτημάτων 76
4.3.5 Άλλα χαρακτηριστικά 77

5 Άλλες χρησιμοποιούμενες τεχνολογίες **79**
5.1 Java 79
5.1.1 Στόχοι και χαρακτηριστικά 80
5.1.2 Χαρακτηριστικά της γλώσσας Java 81
5.1.3 Servlets 83
5.1.4 Java Server Pages (JSP) 84
5.2 Apache Tomcat 84
5.3 JSON 85
5.4 XML 86
5.5 Eclipse 87

II Υλοποίηση κι αποτελέσματα εργασίας **89**

6 Περιγραφή προβλήματος **91**
6.1 Ζητήματα στα υπάρχοντα συστήματα ελαστικότητας 91
6.2 Η ελαστικότητα ως υπηρεσία 93
6.2.1 Γενική ιδέα 93
6.2.2 Βασικά χαρακτηριστικά 93
6.3 Πλατφόρμα διαχείρισης της κλιμάκωσης 94

7 Περιγραφή υλοποίησης συστήματος **97**
7.1 Προδιαγραφές σχεδίασης συστήματος 97
7.1.1 Διάγραμμα υλοποίησης 98
7.1.2 Ψηφιδικό διάγραμμα 99
7.2 Σενάρια χρήσης - Ακολουθιακά διαγράμματα 100
7.2.1 Διάγραμμα δραστών-χρήσης 100
7.2.2 Γενική λειτουργία 102
7.2.3 Deploy 104
7.2.4 Undeploy 105
7.2.5 Scale-up 106
7.2.6 Scale-down 106
7.3 Διάγραμμα κλάσεων 107
7.3.1 Βάση δεδομένων (Database) 109
7.3.2 Οδηγός (Driver) 111
7.3.3 Διακομιστής μεσολάβησης (Proxy Server) 112
7.3.4 Μηχανή ελαστικότητας (Elasticity Engine) 114
7.3.5 Αλληλεπιδραστική εφαρμογή χρήστη (Platform GUI) 115

8 Αποτελέσματα εργασίας **119**
8.1 Λειτουργία πλατφόρμας 119
8.2 Γραφικό περιβάλλον πλατφόρμας 124
8.3 Σενάρια λειτουργίας 127

9 Σύνοψη και συμπεράσματα **131**
9.1 Σύνοψη 131
9.2 Συμπεράσματα και μελλοντική δουλειά 132

Βιβλιογραφία **133**

III Παράρτημα **137**

A Σημαντικές κλάσεις **139**
A.1 IaaS Driver 139
A.2 Nginx Proxy Server 143
A.3 Platform GUI 148
A.4 Database 156

B Σημαντικά αρχεία **163**

Κατάλογος Σχημάτων

1.1 Υπολογιστικό νέφος 21
1.2 Ορισμός NIST για την αρχιτεκτονική του υπολογιστικού νέφους . . . 22
1.3 Τεχνικές διαφορές μεταξύ υπολογιστικού πλέγματος και υπολογιστικού νέφους 24
1.4 Μοντέλα υπηρεσιών υπολογιστικού νέφους 27
1.5 Αρχιτεκτονική υπολογιστικού νέφους 32
1.6 Μοντέλα ανάπτυξης νέφους 33
1.7 Σύνοψη: Χαρακτηριστικά του υπολογιστικού νέφους 41

2.1 Υπερτροφοδότηση 44
2.2 Υποτροφοδότηση 44
2.3 Χρέωση στο νέφος 45

3.1 Υπηρεσία SOA και συνδέσεις 50
3.2 Υπηρεσιοστρεφής αρχιτεκτονική 51
3.3 Λειτουργία SOAP 54
3.4 Βασική ιδέα του REST 56
3.5 Λειτουργία του REST 59
3.6 Παράδειγμα RESTful διαδικτυακής υπηρεσίας 60
3.7 REST vs SOAP 61

4.1 Μερίδιο αγοράς βάσεων δεδομένων 64
4.2 Graph database 67
4.3 Παράδειγμα key-value store 69
4.4 MongoDB architecture 70
4.5 MySQL vs MongoDB 73
4.6 Παράδειγμα BSON αρχείου 74
4.7 Παράδειγμα BSON αρχείου με χρήση αναφορών 75
4.8 Παράδειγμα geospatial ερωτήματος 76

7.1 Διάγραμμα υλοποίησης 98
7.2 Ψηφιδικό διάγραμμα 99
7.3 Διάγραμμα δραστών 100
7.4 Διάγραμμα χρήσης 101

7.5 Ακολουθιακό διάγραμμα γενικής λειτουργίας της υπηρεσίας 102
7.6 Ακολουθιακό διάγραμμα γενικής λειτουργίας της πλατφόρμας 103
7.7 Ακολουθιακό διάγραμμα για το deploy 104
7.8 Ακολουθιακό διάγραμμα για το undeploy 105
7.9 Ακολουθιακό διάγραμμα για το scale-up 106
7.10 Ακολουθιακό διάγραμμα για το scale-down 107
7.11 Διάγραμμα κλάσεων 108

8.1 Λειτουργία για instances 1CPU-1024MB, έως 50 χρήστες. 120
8.2 Λειτουργία για instances 1CPU-1024MB, έως 80 χρήστες. 120
8.3 Σενάριο λειτουργία για instances 1CPU-1024MB. 121
8.4 Λειτουργία για instances 2CPU-2048MB, έως 50 χρήστες. 121
8.5 Λειτουργία για instances 2CPU-2048MB, έως 90 χρήστες. 122
8.6 Σενάριο λειτουργία για instances 2CPU-2048MB. 122
8.7 Λειτουργία για instances 4CPU-4096MB, έως 80 χρήστες. 123
8.8 Λειτουργία για instances 4CPU-4096MB, έως 110 χρήστες. 123
8.9 Σενάριο λειτουργία για instances 4CPU-4096MB. 124
8.10 Σελίδα υποδοχής στην πλατφόρμα. 124
8.11 Εγκατάσταση εφαρμογής στην πλατφόρμα. 125
8.12 Απεγκατάσταση εφαρμογής. 125
8.13 Ορισμός κανόνων. 126
8.14 Απαιτούμενοι πόροι για μία εφαρμογή. 126
8.15 Εγκατάσταση εφαρμογής. 127
8.16 Απεγκατάσταση εφαρμογής. 128
8.17 Αποδέσμευση αχρείαστων πόρων. 129
8.18 Τελευταία στατιστικά λειτουργίας της εφαρμογής flipper. 129
8.19 Εφαρμογές που τρέχουν στην πλατφόρμα 130

Εισαγωγή

Ο κόσμος της πληροφορικής γίνεται ολοένα μεγαλύτερος και πιο πολύπλοκος. Το υπολογιστικό νέφος αναδύεται ως ένα ιδιαίτερα δημοφιλές μοντέλο, το οποίο μπορεί να υποστηρίξει τεράστιο όγκο δεδομένων και απαιτήσεις σε τεράστια υπολογιστική ισχύ χρησιμοποιώντας συμπλέγματα υπολογιστών. Έτσι, το ενδιαφέρον για το υπολογιστικό νέφος ως μία τεχνική και επιχειρηματική λύση αυξάνεται διαρκώς. Η ποικιλία των διαθέσιμων εφαρμογών, πλατφόρμας και υπηρεσιών νέφους αυξάνεται, δημιουργώντας νέα χαρακτηριστικά για το νέφος που παρέχονται στον τελικό χρήστη.

Το υπολογιστικό νέφος παρέχει πλέον ένα δυναμικό και ελαστικό περιβάλλον διάθεσης υπηρεσιών το οποίο μπορεί να είναι ανθεκτικό σε ραγδαίες και γιγαντιαίας κλίμακας μεταβολές των συνθηκών. Αυτό επιτυγχάνεται με τα εγγενή χαρακτηριστικά του, που είναι η αυτόματη ανάκαμψη, η αυτό-επιτήρηση, η αυτό-διαχείριση, η αυτόματη επαναδιαμόρφωση, η δυνατότητα καθορισμού ηλεκτρονικών συμβολαίων (SLAs), και οι υψηλές δυνατότητες (αυτό)κλιμάκωσης.

Ένα από τα πιο σημαντικά χαρακτηριστικά του υπολογιστικού νέφους είναι η διαχείριση της ελαστικότητας. Η ελαστικότητα αφορά τη δυνατότητα του υπολογιστικού νέφους να κλιμακώνει μία υποδομή νέφους κατ΄ απαίτηση μέσα σε λίγα λεπτά ή ακόμα και δευτερόλεπτα, αποφεύγοντας έτσι την άσκοπη κατανάλωση πόρων ή την μη εξυπηρέτηση των αναγκών του χρήστη. Η ελαστικότητα δεν μπορεί παρά να είναι αλληλένδετη με τη συνεχή παρακολούθηση (monitoring) και την ανάλογη εξισορρόπηση του φόρτου (load balancing).

Αντικείμενο διπλωματικής

Αντικείμενο αυτής της διπλωματικής εργασίας είναι η ανάπτυξη μιας πλατφόρμας υπολογιστικού νέφους (PaaS), η οποία διαχειρίζεται αυτόματα την κλιμάκωση των υπολογιστικών πόρων (scalability) διαδικτυακών εφαρμογών, εγκατεστημένες σε περιβάλλον υπολογιστικού νέφους. Η κλιμάκωση γίνεται με βάση διάφορες παραμέτρους που ορίζει ο πάροχος της κάθε εφαρμογής-υπηρεσίας. Τέτοιες, μπορούν να είναι ο επιθυμητός χρόνος απόκρισης της υπηρεσίας και ο μέγιστος αριθμός χρηστών που την χρησιμοποιούν σε ένα χρονικό διάστημα.

Ο μηχανισμός ελαστικότητας διαφέρει από πάροχο σε πάροχο. Στη διπλωματική αυτή υλοποιείται μία νέα προσέγγιση της ελαστικότητας [1], σύμφωνα με την οποία η ελαστικότητα παρέχεται σαν μία υπηρεσία στην πλατφόρμα και δεν αποτελεί εγγενές

κομμάτι της. Αυτό συμβαίνει γιατί κάθε εφαρμογή έχει διαφορετικές απαιτήσεις σε πόρους. Έτσι, δίνεται η δυνατότητα στο χρήστη να θέσει τους δικούς του κανόνες και τις δικές του παραμέτρους για την κλιμάκωση της εφαρμογής του, χωρίς να λαμβάνει υπ' όψη εγγενείς περιορισμούς της πλατφόρμας, όπως η δυνατότητα ορισμού παραμέτρων μόνο σε επίπεδο υποδομής (IaaS) και να καθορίζει ο ίδιος την ποιότητα της υπηρεσίας που επιθυμεί.

Το σύστημα που αναπτύχθηκε αποτελείται από διάφορα επιμέρους υποσυστήματα:

- μία αλληλεπιδραστική εφαρμογή (graphical user interface) μέσα από την οποία ο πάροχος μίας εφαρμογής ορίζει τις απαιτήσεις σε χρόνο απόκρισης και τον μέγιστο αριθμό των χρηστών στον οποίο θα πρέπει να ανταποκρίνεται η εφαρμογή του.

- ένας οδηγός (driver) για την επικοινωνία της πλατφόρμας με τον πάροχο υποδομής (IaaS), με σκοπό την εύκολη δέσμευση κι αποδέσμευση πόρων μέσω κώδικα σε java.

- ένα (image) ενός διακομιστή μεσολάβησης (proxy server) ο οποίος δημιουργείται κατά την εκκίνηση κάθε νέας εφαρμογής που εγκαθίσταται στην πλατφόρμα και διανέμει την κίνηση ισάξια στις εικονικές μηχανές, που τρέχουν στον πάροχο υποδομής και στις οποίες τρέχει η εφαρμογή. Σε κάθε τέτοιον διακομιστή τρέχει ένας δαίμονας (daemon), ο οποίος αναλαμβάνει να στέλνει ανά τακτά χρονικά διαστήματα ορισμένα σημαντικά δεδομένα, όπως το χρόνο απόκρισης και τον αριθμό αιτήσεων της εκάστοτε εφαρμογής, σε μία βάση δεδομένων που βρίσκεται στην πλατφόρμα.

- την πλατφόρμα, η οποία αποτελεί το κύριο μέρος του συστήματος. Ανάλογα με τους πραγματικούς χρόνους απόκρισης, τον αριθμό αιτήσεων της εφαρμογής, τον επιθυμητό χρόνο απόκρισης και το μέγιστο αριθμό χρηστών, υπολογίζει τους απαιτούμενους πόρους για την επιθυμητή λειτουργία της εφαρμογής και κλιμακώνει ανάλογα τους υπολογιστικούς πόρους του νέφους υποδομής (IaaS).

Για τον υπολογισμό των απαιτούμενων πόρων (requirement resource calculator) χρησιμοποιήθηκαν διαφορετικοί αλγόριθμοι. Αυτοί συγκρίθηκαν μεταξύ τους με βάση γραφικές παραστάσεις και με κύρια μεταβλητή το χρόνο απόκρισης της εφαρμογής ως προς το χρόνο.

Ο αλγόριθμος που επιλέχθηκε τελικά για τον υπολογισμό των πόρων, βασίζεται στη χρήση κανόνων με βάση τους οποίους γίνεται πιο έξυπνα η κλιμάκωση. Αυτοί οι κανόνες προέκυψαν από μετρήσεις που πραγματοποιήθηκαν για διάφορους αριθμούς χρηστών και για διαφορετικούς πόρους, για μία συγκεκριμένη εφαρμογή (benchmark).

Οργάνωση κειμένου

Η παρούσα εργασία χωρίζεται σε τρία κύρια μέρη. Στο πρώτο ανήκουν τα κεφάλαια 1, 2, 3 και 4, όπου παρουσιάζονται οι έννοιες και οι τεχνολογίες που μελετήθηκαν με στόχο τη αποκόμιση των απαραίτητων γνώσεων για την εξέλιξη της διπλωματικής. Στο δεύτερο μέρος περιλαμβάνονται τα κεφάλαια 5, 6 κι 7, όπου γίνεται εκτενέστερη ανάλυση της περιγραφής του προβλήματος, εξηγείται η υλοποίηση και παρουσιάζονται τα αποτελέσματα. Στο τρίτο μέρος περιλαμβάνεται το παράρτημα με τον κώδικα της υλοποίησης.

Πιο αναλυτικά:

- Στο κεφάλαιο 1 γίνεται μία εισαγωγή στην έννοια του υπολογιστικού νέφους, παρουσιάζονται τα χαρακτηριστικά του, γίνεται ο διαχωρισμός του σε κατηγορίες κι εξηγούνται τα πλεονεκτήματα του. Στη συνέχεια εξηγούνται τα προβλήματα για την ευρεία υιοθέτησή του και γίνεται αναφορά στις μελλοντικές προκλήσεις σχετικά με αυτό. Στο κεφάλαιο 2, γίνεται ιδιαίτερη αναφορά στο γνώρισμα της ελαστικότητας κι αναλύονται οι τρόποι με τους οποίους αυτή επιτυγχάνεται σήμερα σε διάφορα εμπορικά υπολογιστικά νέφη. Ακολούθως, στο κεφάλαιο 3 παρουσιάζονται τα χαρακτηριστικά των αρχιτεκτονικών μοντέλων SOAP και REST και γίνεται μία απλή σύγκριση. Τέλος, στο κεφάλαιο 4 δίνονται τα βασικά χαρακτηριστικά των NoSQL βάσεων δεδομένων και αναλύεται περαιτέρω η βάση mongoDB.

- Στο κεφάλαιο 5 γίνεται μια πιο αναλυτική περιγραφή της διπλωματικής κι εξηγούνται οι λόγοι για τους οποίους αυτή η νέα τεχνική ελαστικότητας υπερτερεί σε σχέση με παλιότερες μεθόδους. Στη συνέχεια, στο κεφάλαιο 6 δίνονται οι λειτουργικές και μη λειτουργικές απαιτήσεις για την ανάπτυξη της πλατφόρμας και παρουσιάζονται τα απαραίτητα διάγραμματα UML που εξηγούν τη λειτουργία της. Έπειτα, στο κεφάλαιο 7 παρουσιάζονται τα αποτελέσματα για τους αλγόριθμους υπολογισμού απαιτούμενων πόρων requirement resource calculator που δοκιμάστηκαν στην πλατφόρμα και γίνεται μία σχετική σύγκριση.

Μέρος Ι

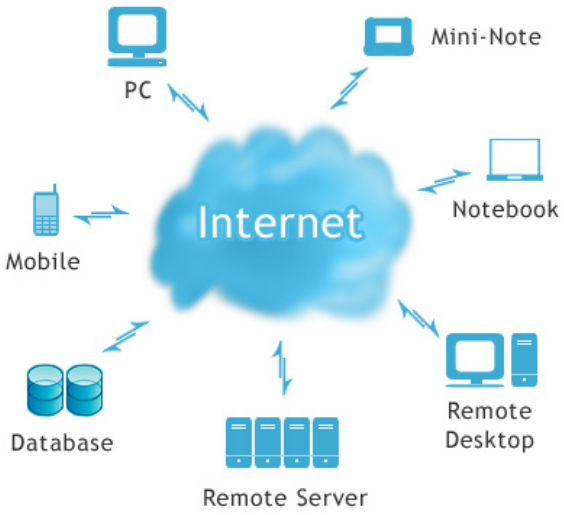
Θεωρητικό υπόβαθρο

Κεφάλαιο 1

Υπολογιστικά νέφη

1.1 Εισαγωγή

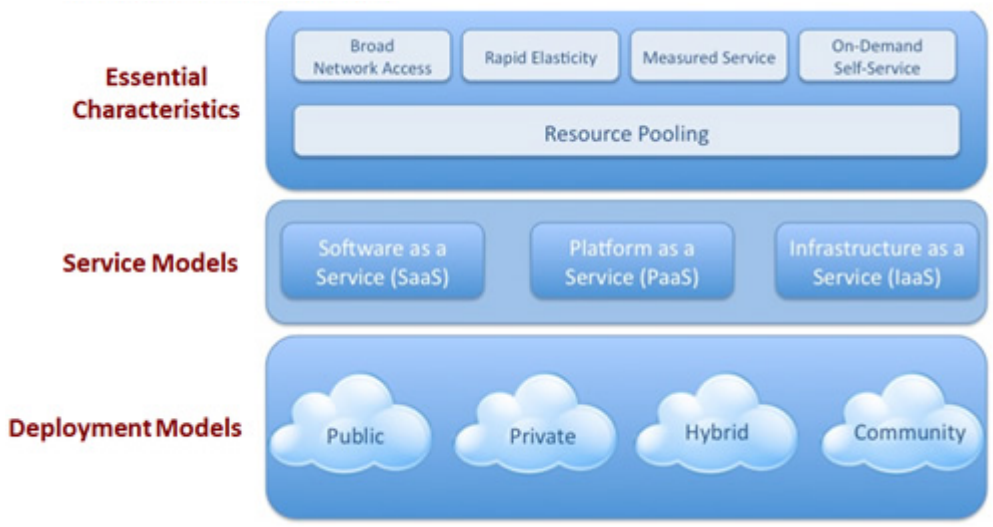
Υπολογιστικό νέφος ονομάζεται η κατ΄ αίτηση διαδικτυακή κεντρική διάθεση υπολογιστικών πόρων (όπως δίκτυο, εξυπηρετητές, εφαρμογές και υπηρεσίες) με υψηλή ευελιξία, ελάχιστη προσπάθεια από τον χρήστη και υψηλή αυτοματοποίηση. Στο υπολογιστικό νέφος η αποθήκευση, η επεξεργασία και η χρήση δεδομένων, λογισμικού και υπηρεσιών γίνεται διαδικτυακά, μέσω απομακρυσμένων υπολογιστών σε κεντρικά datacenter. Υπηρεσίες όπως η κατ΄ αίτηση παροχή εικονικών μηχανών, το ηλεκτρονικό ταχυδρομείο ή τα κοινωνικά δίκτυα συχνά βασίζονται στην τεχνολογία του υπολογιστικού νέφους.



Σχήμα 1.1: Υπολογιστικό νέφος

Σύμφωνα με τον ορισμό του Εθνικού Ινστιτούτου Τυποποιήσεων και Τεχνολογίας (NIST - National Institute of Standards and Technology) [2], το υπολογιστικό νέφος είναι ένα μοντέλο που επιτρέπει την εύκολη, κατ΄ αίτηση (on-demand) πρόσβαση απ΄ οπουδήποτε σε μία κοινόχρηστη τράπεζα παραμετροποιήσιμων υπολογιστικών πόρων (δίκτυα, εξυπηρετητές, μέσα αποθήκευσης, εφαρμογές και υπηρεσίες), οι οποίοι μπορούν να δεσμεύονται και να ελευθερώνονται με την ελάχιστη δυνατή προσπάθεια ή αλληλεπίδραση με τον πάροχο της υπηρεσίας.

Αυτό το μοντέλο αποτελείται από πέντε βασικά χαρακτηριστικά (αυτοεξυπηρέτηση κατ΄ απαίτηση, ευρεία πρόσβαση στο δίκτυο, διάθεση πόρων, ταχεία ελαστικότητα και μετρούμενη υπηρεσία), τρία μοντέλα υπηρεσίας νέφους (λογισμικό ως υπηρεσία, πλατφόρμα ως υπηρεσία, υποδομή ως υπηρεσία) και τέσσερα μοντέλα ανάπτυξης νέφους (δημόσιο νέφος, ιδιωτικό νέφος, νέφος κοινότητας, υβριδικό νέφος).



Σχήμα 1.2: Ορισμός NIST για την αρχιτεκτονική του υπολογιστικού νέφους

Το υπολογιστικό νέφος τα τελευταία χρόνια αναδείχθηκε ως ένα νέο πρότυπο για τη φιλοξενία και την παροχή υπηρεσιών μέσω του διαδικτύου [3] [4] [5]. Γίνεται όλο και περισσότερο ελκυστικό για τους επιχειρηματίες, εφόσον επιτρέπει στις εταιρείες να ξεκινούν με λίγους πόρους και να τους αυξάνουν μόνο όταν υπάρχει αύξηση στην απαίτηση υπηρεσιών. Παρόλα αυτά, η ανάπτυξη του υπολογιστικού νέφους έχει ακόμα πολλά εμπόδια και ουσιαστικά είμαστε στην αρχή μιας μεταβατικής περιόδου κατά την οποία πρέπει να παρθούν πολλές αποφάσεις σχετικά με την υιοθέτηση του από τον επιχειρηματικό κόσμο. Η απόφαση αυτή είναι δύσκολη εξαιτίας ενός εύρους πρακτικών και κοινωνικοπολιτικών αιτιών και ως εκ τούτου είναι ακόμα αδύνατο να αναθέσουν όλες οι εταιρείες τις υπολογιστικές απαιτήσεις τους σε εξωτερικούς παρόχους υπολογιστικού νέφους [6] [7] [8].

1.2 Εξέλιξη

1.2.1 Ιστορική αναδρομή

Η βασική ιδέα του υπολογιστικού νέφους χρονολογείται στη δεκαετία του 1950 [9], όταν μεγάλης κλίμακας υπολογιστές έγιναν διαθέσιμα στα πανεπιστήμια και τις επιχειρήσεις και ήταν προσβάσιμοι μέσω τερματικών. Για την πιο αποτελεσματική χρήση των δαπανηρών κεντρικών υπολογιστών, εξελίχθηκε μια πρακτική ώστε να επιτρέπεται σε πολλούς χρήστες να μοιράζονται τόσο τη φυσική πρόσβαση στον υπολογιστή από πολλά τερματικά, καθώς και να μοιράζονται το χρόνο του επεξεργαστή.

Τη δεκαετία του 1960, ο John McCarthy [10] υποστήριξε ότι κάποια στιγμή, το υπολογιστικό κόστος θα μπορούσε να αποτελέσει μία δημόσια υπηρεσία. Ο Douglas Parkhill [11] ανέλυσε διεξοδικά χαρακτηριστικά όπως η ελαστικότητα, η παροχή υπολογιστικού κόστους ως υπηρεσία, η πρόσβαση μέσω διαδικτύου και η ψευδαίσθηση απεριόριστων πόρων. Παράλληλα, ο Herb Grosch, που έχει γράψει το νόμο του Grosch [12], ισχυρίστηκε ότι ολόκληρος το 1960 ο κόσμος θα μπορούσε να δουλέψει σε τερματικά που τροφοδοτούνται από 15 μεγάλα datacenters.

Τη δεκαετία του 1990, οι εταιρίες τηλεπικοινωνιών που μέχρι πρότερα παρείχαν αποκλειστικά συνδέσεις σημείου-σε-σημείο, άρχισαν να παρέχουν και υπηρεσίες εικονικών ιδιωτικών δικτύων. Με κατάλληλη προσαρμογή της κυκλοφορίας των δεδομένων, ώστε να βελτιστοποιείται η χρήση του κάθε εξυπηρετητή, μπορούσαν να χρησιμοποιήσουν το συνολικό εύρος ζώνης του δικτύου πιο αποτελεσματικά.

Τη δεκαετία του 2000, η Amazon έπαιξε καθοριστικό ρόλο στην ανάπτυξη του υπολογιστικού νέφους, με τον εκσυγχρονισμό των datacenter της, τα οποία χρησιμοποιούσαν μόλις το 10% της μέγιστης δυνατότητάς τους, αφήνοντας περιθώριο για περιστασιακές ανάγκες. Το 2006, ξεκίνησε μια νέα προσπάθεια για την ανάπτυξη και την παροχή νέφους σε εξωτερικούς πελάτες, παρουσιάζοντας το Amazon Web Services (AWS) [13] και το Amazon Elastic Compute Cloud (EC2), το οποίο επιτρέπει σε μικρές επιχειρήσεις και ιδιώτες να νοικιάζουν υπολογιστικούς πόρους.

Η IBM ήταν η πρώτη που άρχισε να χαράσσει μια σαφή στρατηγική για το υπολογιστικό νέφος το 2007, έχοντας ως σκοπό την κατασκευή υπηρεσιών υπολογιστικού νέφους για επιχειρήσεις. Επίσης, ανακοίνωσε συνεργασία με την Google για την προώθηση του υπολογιστικού νέφους στα πανεπιστήμια.

Στις αρχές του 2008, το Eucalyptus [14] έγινε η πρώτη συμβατή πλατφόρμα με την προγραμματιστική διεπαφή του AWS για την ανάπτυξη των ιδιωτικών υπολογιστικών νεφών, ενώ το OpenNebula [15], έγινε το πρώτο λογισμικό ανοιχτού κώδικα για την ανάπτυξη ιδιωτικών και υβριδικών υπολογιστικών νεφών. Το ίδιο έτος, οι προσπάθειες επικεντρώθηκαν στην παροχή εγγυήσεων ποιότητας υπηρεσιών (όπως απαιτείται σε διαδραστικές εφαρμογές πραγματικού χρόνου) σε υποδομές υπολογιστικού νέφους.

Η εξέλιξη του διαδικτύου και η νέα οπτική για την παροχή όλο και περισσότερων υπηρεσιών, σε συνδιασμό με τα ταχύτατα δίκτυα, το χαμηλό κόστος των υπολογιστών και των μέσων αποθήκευσης και την ευρεία υιοθέτηση υπηρεσιοστρεφών αρχιτεκτονικών (Service Oriented Architecture - SOA) [16] και εικονικών μηχανών (hardware virtualization), οδηγεί στην περαιτέρω ανάπτυξη του υπολογιστικού νέφους.

1.2.2 Σχετικές τεχνολογίες

- Το **υπολογιστικό πλέγμα** είναι ένα μοντέλο κατανεμημένου συστήματος, που συντονίζει ένα σύνολο πόρων για την επίτευξη ενός κοινού υπολογιστικού στόχου. Η ανάπτυξη του υπολογιστικού πλέγματος αρχικά δημιουργήθηκε για τις ανάγκες επιστημονικών εφαρμογών μεγάλου υπολογιστικού κόστους.

  Ανέκαθεν υπήρχε διαφωνία για το διαχωρισμό του υπολογιστικού νέφους με το υπολογιστικό πλέγμα (Cloud Computing versus Grid Computing). Η κύρια διαφορά μεταξύ των δύο μοντέλων έγκειται στο γεγονός ότι το υπολογιστικό νέφος αξιοποιεί την εικονικοποίηση (virtualization), κάνοντας παράλληλα σαφή διαχωρισμό των στρωμάτων λογισμικού και ηλικού (IaaS, PaaS, SaaS). Αντίθετα, το υπολογιστικό πλέγμα πετυχαίνει μεγάλο επίπεδο χρησιμοποίησης από την κατανομή μίας διαδικασίας σε πολλαπλούς εξυπηρετητές, όταν ζητείται ανάθεση μιας εκτέλεσης εργασίας.

	Grid Computing	Cloud Computing
Χρησιμοποιούμενα μέσα	Κατανομή σε πολλαπλούς διακομιστές (servers) μιας απλής διαδικασίας.	Εικονικοί servers, ένας server εκτελεί πολλές διαδικασίες ταυτόχρονα.
Τυπικό μοτίβο χρησιμοποίησης	Συνήθως χρησιμοποιείται για εκτέλεση εργασιών λ.χ. εκτέλεση ενός προγράμματος για περιορισμένο χρόνο.	Τακτική χρήση για υπηρεσίας μακρόχρονης υποστήριξης.
Επίπεδο χρήσης εικονικών μέσων	Περιορισμένη χρήση εικονικών μέσων	Εκτεταμένη χρήση εικονικών μέσων

Σχήμα 1.3: Τεχνικές διαφορές μεταξύ υπολογιστικού πλέγματος και υπολογιστικού νέφους

- Η **εικονικοποίηση** είναι μια τεχνολογία που αγνοεί τις λεπτομέρειες του φυσικού υλικού και γενικότερα του χαμηλότερου στρώματος και παρέχει εικονικούς πόρους για εφαρμογές υψηλού επιπέδου. Ένα εικονικός εξυπηρετητής ονομάζεται συνήθως εικονική μηχανή (Virtual Machine - VM). Παρέχει τη δυνατότητα της συγκέντρωσης πραγματικών υπολογιστικών πόρων και την ανάθεση ή την ανακατανομή εικονικών πόρων σε εφαρμογές on-demand.

- Η **αυτόνομη υπολογιστικότητα** αποσκοπεί στη δημιουργία υπολογιστικών συστημάτων που έχουν τη δυνατότητα αυτοδιαχείρισης, δηλαδή αντίδρασης ανάλογα με εσωτερικές και εξωτερικές παρατηρήσεις, χωρίς ανθρώπινη παρέμβαση. Στόχος είναι να ξεπεραστεί η πολυπλοκότητα της διαχείρισης των σημερινών συστημάτων υπολογιστών.

1.3 Βασικά χαρακτηριστικά

1.3.1 Χαρακτηριστικά κατά NIST

Παρακάτω φαίνονται τα πέντε βασικά χαρακτηριστικά σύμφωνα με τον ορισμό του υπολογιστικού νέφους κατά NIST [2]:

- **Αυτοεξυπηρέτηση κατ΄ απαίτηση (On-demand self-service)**
  Ο χρήστης μπορεί όποτε θέλει να χρησιμοποιήσει μια υπηρεσία ή να δεσμεύει υπολογιστικούς πόρους, όπως μονάδες επεξεργασίας και αποθήκευσης, χωρίς την ενδιάμεση αλληλεπίδραση με τον πάροχο των υπηρεσιών.

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

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

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

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

1.3.2 Πρόσθετα χαρακτηριστικά

Στη συνέχεια παραθέτουμε μερικά ακόμα σημαντικά χαρακτηριστικά του υπολογιστικού νέφους:

- **Κλιμάκωση (Scalability)**
  Στο υπολογιστικό νέφος είναι δυνατή η χειροκίνητη ή δυναμική προσθήκη και αφαίρεση κόμβων επεξεργασίας, εκτέλεσης εργασιών ή αποθήκευσης, ανάλογα με την αυξομείωση των απαιτήσεων, χωρίς να αλλοιώνεται η υπηρεσία που παρέχεται στο χρήστη.

- **Εικονικοποίηση (Virtualization)**
  Οι λεπτομέρειες υλοποίησης και κατάστασης στην οποία βρίσκονται οι υπολογιστικοί πόροι αποκρύπτονται από το χρήστη. Οι εικονικές μηχανές (Virtual Machines) δίνουν την εντύπωση στο χρήστη ενός ολοκληρωμένου φυσικού μηχανήματος, ενώ στην πραγματικότητας είναι ένα σύνολο αρχείων και προγραμμάτων που τρέχουν πάνω σε ένα άλλο (ή άλλα) φυσικά μηχανήματα. Οι εικονικές μηχανές μπορεί να έχουν διαφορετικά χαρακτηριστικά από τους φυσικούς πόρους πάνω στους οποίους τρέχουν, τόσο όσον αφορά το λογισμικό, όσο και για το υλικό.

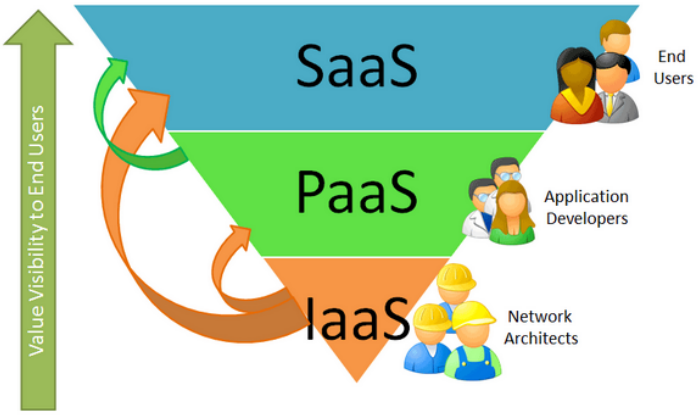
- **Αξιοπιστία (Reliability)**
  Το υπολογιστικό νέφος εγγυάται την ορθή λειτουργία των προγραμμάτων των χρηστών, την αποθήκευση των δεδομένων τους και την αξιόπιστη μεταφορά τους. Για την εξασφάλιση της αξιοπιστίας, τα δεδομένα αποθηκεύονται σε περισσότερους από έναν κόμβους στο νέφος. Ο συνηθισμένος αριθμός ντιγράφων είναι 3 (replication level three). Επίσης, εφόσον τα δεδομένα του χρήστη βρίσκονται σε κάποια μηχανήματα σε ένα datacenter, μία βλάβη στο τερματικό του χρήστη δεν θα έχει καμία επίδραση στα δεδομένα αυτά.

- **Συντήρηση (Maintenance)**
  Η αποσφαλμάτωση των εφαρμογών και η αναβάθμιση των εκδόσεών τους δεν είναι πλέον ευθύνη του χρήστη, αλλά του παρόχου υπολογιστικού νέφους. Είναι στη δικαιοδοσία του παρόχου να προσφέρει συμβατές υπηρεσίες και να τις αναβαθμίζει διαρκώς.

- **Πολυπελατιακότητα (Multitenancy)**
  Κάθε χρήστης μιας υπηρεσίας υπολογιστικού νέφους, δεν χρειάζεται να έχει δικό του αντίγραφο της εφαρμογής. Αρκεί ένα μοναδικό στιγμιότυπο (instance) το οποίο μπορεί να προσαρμοστεί στις ανάγκες του κάθε χρήστη. Αυτό έχει ως αποτέλεσμα την εξοικονόμηση πόρων στο νέφος και την ευκολότερη συντήρηση της εφαρμογής.

1.4 Μοντέλα υπηρεσιών νέφους

Ο διαχωρισμός των υπηρεσιών νέφους γίνεται με βάση το αφαιρετικό επίπεδο ελέγχου των χρηστών στους υπολογιστικούς πόρους που τους παρέχονται. Τα πιο σημαντικά μοντέλα είναι η υποδομή ως υπηρεσία (IaaS), η πλατφόρμα ως υπηρεσία (PaaS) και το λογισμικό ως υπηρεσία (SaaS). Κάθε μοντέλο υψηλότερου επιπέδου αποκρύπτει τα στοιχεία των προγενέστερων μοντέλων. Άλλα μοντέλα μπορεί να είναι το οτιδήποτε ως υπηρεσία (XaaS) και το δίκτυο ως υπηρεσία (NaaS).



Σχήμα 1.4: Μοντέλα υπηρεσιών υπολογιστικού νέφους

1.4.1 Υποδομή ως υπηρεσία (IaaS)

Όταν παρέχεται υποδομή ως υπηρεσία (Infrastructure as a Service - IaaS), ο χρήστης έχει τη δυνατότητα δέσμευσης υπολογιστικών πόρων επεξεργασίας, αποθήκευσης, δικτύων κι άλλων. Ο χρήστης είναι σε θέση να αναπτύξει και να εκτελέσει όποιο λογισμικό θέλει, που μπορεί να είναι τόσο λειτουργικά συστήματα, όσο κι εφαρμογές. Ο χρήστης δεν διαχειρίζεται ή ελέγχει την υποδομή του νέφους, αλλά έχει τον έλεγχο λειτουργίας των συστημάτων, της αποθήκευσης, της ανάπτυξης εφαρμογών κι ενδεχομένως περιορισμένο έλεγχο για την επιλογή στοιχείων του δικτύου (π.χ. firewalls). Μπορεί να ορίζει μερικώς ή πλήρως τα χαρακτηριστικά των μηχανημάτων που χειρίζεται, όπως επεξεργαστή, μνήμη, σκληρό δίσκο, λειτουργικό σύστημα, δίκτυο κι άλλα. Μπορεί να τα χειριστεί ακριβώς όπως θα έκανε σε δικά του μηχανήματα.

Οι υπολογιστικοί πόροι παρέχονται στο χρήστη κυρίως ως εικονικές μηχανές, αλλά μερικές φορές και ως φυσικά μηχανήματα. Ο πάροχος είναι υπεύθυνος για τη συντήρηση των μηχανημάτων στα οποία τρέχουν οι εικονικές μηχανές και φροντίζει για την ασφάλεια, την αξιοπιστία και την επεκτασιμότητά τους.

Amazon EC2

Με το Amazon EC2 [13] οι χρήστες νοικιάζουν υπολογιστικούς πόρους, δημιουργούν εικονικές μηχανές και αναπτύσσουν δικές τους υπηρεσίες. Αρχικά οι χρήστες επιλέγουν μία Amazon Machine Image (AMI) ή δημιουργούν μία δική τους που περιέχει εφαρμογές, δεδομένα και βιβλιοθήκες και δημιουργεί μία εικονική μηχανή στο νέφος. Στη συνέχεια επιλέγονται οι ρυθμίσεις δικτύου, ασφαλείας και λειτουργίας της κάθε εικονικής μηχανής. Υπάρχει μάλιστα η δυνατότητα επιλογής του φυσικού χώρου στον οποίο θα τρέξουν οι εικονικές μηχανές, μεταξύ των περιοχών της Amazon.

Παρακάτω αναλύονται τα κυριότερα χαρακτηριστικά του Amazon EC2:

- Ελαστικότητα: Οι χρήστες μπορούν να έχουν όσους διαθέσιμους πόρους θέλουν χωρίς αναμονή. Υπάρχει επίσης η δυνατότητα αυτόματης κλιμάκωσης των πόρων που χρησιμοποιούν οι εφαρμογές, έτσι ώστε κάθε στιγμή να δεσμεύονται όσοι πόροι χρειάζονται, μειώνοντας το κόστος παροχής υπηρεσιών αλλά και την αποδοτικότητά τους. Έτσι, οι χρήστες μπορούν να ορίζουν συνθήκες κάτω από τις οποίες προστίθενται ή αφαιρούνται πόροι σε μία υπηρεσία, ώστε να διατηρείται η απόδοσή της και παράλληλα να διατηρείται το κόστος χαμηλό.

- Έλεγχος: Οι χρήστες μπορούν να έχουν τον απόλυτο έλεγχο πάνω στις εικονικές μηχανές τους, έχοντας τη δυνατότητα να τα εκκινούν ή να τα τερματίζουν οποιαδήποτε στιγμή επιθυμούν και να τα ελέγχουν με άμεση πρόσβαση σε αυτά ή μέσω μίας διεπαφής ιστού.

- Ευελιξία: Οι χρήστες μπορούν να επιλέξουν μέσα από διάφορους πόρους, λειτουργικά συστήματα και πακέτων λογισμικού. Για παράδειγμα, υπάρχει η δυνατότητα επιλογής χαρακτηριστικών όπως μνήμης, επεξεργαστικής ισχύος, αποθηκευτικού χώρου και λειτουργικού συστήματος.

- Αξιοπιστία και ασφάλεια: Το Amazon EC2 εγγυάται τη διαθεσιμότητα των υπηρεσιών και την ασφάλεια των δεδομένων, κάτι που εξασφαλίζεται με τη διατήρηση πολλαπλών αντιγράφων σε διάφορες περιοχές της υποδομής πόρων. Παράλληλα, καθιστά δυνατή τη ρύθμιση firewalls, αλλά και τη δημιουργία εικονικού ιδιωτικού νέφους (Amazon Virtual Cloud - AVP).

- Συμβατότητα με Amazon Web Services: Το πακέτο AWP συνιστά ένα ολοκληρωμένο σύστημα ανάπτυξης δικτυακών υπηρεσιών. Έτσι, το Amazon Simple Storage Service (S3) παρέχει αποθηκευτικό χώρο, το Amazon Relational Database Service (RDS) παρέχει μία σχεσιακή βάση δεδομένων και το Amazon SimpleDB παρέχει μία NoSQL βάση δεδομένων.

Παράλληλα, το Amazon EC2 παρέχει ορισμένες πρόσθετες ευκολίες στους χρήστες.

- Οι χρήστες μπορούν να δεσμεύσουν εικονικές μηχανές με τρεις διαφορετικούς τρόπους, ώστε να πετυχαίνουν το ελάχιστο δυνατό κόστος για αυτούς, ανάλογα με τις εφαρμογές που θέλουν να τρέξουν. Ο πρώτος τρόπος είναι να επιλέξουν

on-demand χρέωση, κατά την οποία η τιμολόγηση είναι ανάλογη με τη χρήση, χωρίς να απαιτείται έτσι εκ των προτέρων δέσμευση των πόρων. Ο δεύτερος τρόπος είναι η εκ των προτέρων ενοικίαση εικονικών μηχανών (reserved instances) κι ενδείκνυται για χρήστες που δεν ξέρουν πόσους πόρους θα χρειαστούν στο μέλλον, έχοντας σημαντική έκπτωση στην ωριαία χρέωση. Υπάρχει η δυνατότητα δέσμευσης πόρων για ελαφριά, μέτρια και έντονη χρήση (light, medium και heavy utilization), με ανάλογη τιμολόγηση. Τέλος, μπορούν οι χρήστες να ορίζουν μία τιμή, κι όταν αυτή είναι μεγαλύτερη από την Spot price, η οποία αλλάζει ανάλογα με τη διαθεσιμότητα και τη ζήτηση πόρων, να προσφέρεται μία εικονική μηχανή στο χρήστη, αλλιώς να αποδεσμεύεται. Αυτή η τιμολογιακή πολιτική είναι ιδανική είτε για εφαρμογές με ευελιξία ως προς το πότε θα τρέξουν, είτε για fault-tolerant υπηρεσίες.

- Η υποδομή του Amazon συνίσταται από περιοχές (Regions) και ζώνες διαθεσιμότητας (Availability zones). Οι περιοχές βρίσκονται σε διάφορα μέρη του κόσμου, ενώ καθεμιά έχει μία ή περισσότερες ζώνες διαθεσιμότητας. Οι χρήστες μπορούν να επιλέξουν την περιοχή στην οποία θα αποθηκευτούν τα δεδομένα τους και θα τρέξουν οι εικονικές μηχανές τους, πετυχαίνοντας μεγάλη διαθεσιμότητα. Παράλληλα, έχουν τη δυνατότητα αποθήκευσης των δεδομένων σε πολλές ζώνες διαθεσιμότητας, προστατεύοντάς τα από σφάλματα.

- Οι χρήστες μπορούν μέσω μίας διαδικτυακής υπηρεσίας ελέγχου, το Amazon CloudWatch να ελέγχουν τις εφαρμογές και τους πόρους του νέφους. Έτσι, επιλέγουν τις εικονικές μηχανές που θέλουν να παρακολουθήσουν και λαμβάνουν πληροφορίες για αυτά, όπως χρήση του επεξεργαστή, read και writes στο δίσκο, κίνηση δικτύου, αλλά και να συλλέγουν στατιστικά αποτελέσματα και γραφήματα.

- Οι χρήστες μπορούν να εισάγουν images εικονικών μηχανών που έχουν σε δικά τους φυσικά μηχανήματα και να δημιουργούν νέες εικονικές μηχανές με αυτά. Ομοίως, μπορούν να εξάγουν τα images εικονικών μηχανών που τρέχουν στο Amazon EC2.

- Με τη βοήθεια του AWS Marketplace, οι χρήστες μπορούν να πωλούν και να αγοράζουν από την Amazon ή κι από άλλους χρήστες reserved instances που έχουν δημιουργήσει αυτοί.

Okeanos

Ο Okeanos [17] είναι η IaaS υπηρεσία που παρέχεται από το ΕΔΕΤ και χρησιμοποιεί την πλατφόρμα υπολογιστικού νέφους Synnefo. Απευθύνεται στην ακαδημαϊκή κοινότητα και αυτή τη στιγμή διατίθεται δωρεάν και προσφέρει δύο κύριες υπηρεσίες. Η πρώτη είναι η Cyclades, που αφορά τη διάθεση υπολογιστικών και δικτυακών πόρων και η δεύτερη είναι ο Pithos+, που παρέχει αποθηκευτικό χώρο.

- Η υπηρεσία Cyclades Προσφέρει υπολογιστικούς πόρους με τη μορφή εικονικών μηχανών. Οι χρήστες μπορούν να δημιουργούν, να καταστρέφουν, να διαχειρίζονται τις εικονικές μηχανές τους και να συνδέονται σε αυτές, ενώ μπορούν να επιλέξουν ανάμεσα σε διάφορα λειτουργικά συστήματα ή να εγκαταστήσουν κάποιο άλλο της επιλογής τους. Επίσης, έχουν τη δυνατότητα να δημιουργούν τα δικά τους images εικονικών μηχανών και να τα μοιράζονται με άλλους χρήστες. Παράλληλα, μπορούν να επιλέξουν μεταξύ τριών firewalls, ενώ είναι δυνατή και η δημιουργία εικονικών ιδιωτικών δικτύων (VPNs) για μεγαλύτερη απομόνωση των εικονικών μηχανών τους. Τέλος, η διαχείριση των εικονικών μηχανών γίνεται τόσο μέσω του γραφικού περιβάλλοντος δικτυακής εφαρμογής, όσο και με τη βοήθεια ενός εύχρηστου RESTful API που παρέχεται.

- Μέσω της υπηρεσίας Pithos+, οι χρήστες μπορούν να αποθηκεύουν αρχεία στην υποδομή του Okeanos και να έχουν πρόσβαση σε αυτά μέσω του διαδικτύου και των εικονικών τους μηχανών, ενώ είναι δυνατός και ο διαμοιρασμός των αρχείων με άλλους χρήστες ή ομάδες χρηστών. Για την εύκολη διαχείριση και οργάνωση των αρχείων υπάρχει μία διαδικτυακή εφαρμογή, ενώ ο χρήστης μπορεί να συγχρονίσει την υπηρεσία Pithos+ με κάποιον κατάλογο του υπολογιστή του, διαχειρίζοντας έτσι αυτόματα την οργάνωση και την αποθήκευση των αρχείων του.

Flexiscale

Η FlexiScale [18] είναι μία IaaS υπηρεσία που αναπτύχθηκε αρχικά από την εταιρεία XCALIBRE Communications το καλοκαίρι του 2007 κι αργότερα αποκτήθηκε από την Flexiant. Εμφανιζόμενη λίγο μετά την υποδομή Amazon EC2, ήταν η πρώτη υπηρεσία παροχής υποδομής υπολογιστικού νέφους στην Ευρώπη και η δεύτερη στον κόσμο. Οι χρήστες έχουν τη δυνατότητα να δημιουργήσουν, να ξεκινήσουν, να σταματήσουν και να διαχειριστούν εικονικές μηχανές. Υποστηρίζεται τόσο το λειτουργικό σύστημα Windows, όσο και το Linux, ενώ οι χρήστες μπορούν να δημιουργούν τα δικά τους images και να δημιουργούν εικονικά ιδιωτικά δίκτυα. Η διαχείριση των εικονικών μηχανών γίνεται τόσο μέσω του γραφικού περιβάλλοντος δικτυακής εφαρμογής, όσο και με τη βοήθεια ενός εύχρηστου SOAP API που παρέχεται.

1.4.2 Πλατφόρμα ως υπηρεσία (PaaS)

Όταν παρέχεται πλατφόρμα ως υπηρεσία (Platform as a Service - PaaS), ο χρήστης έχει τη δυνατότητα να αναπτύξει πάνω στην υποδομή νέφους, εφαρμογές που δημιουργήθηκαν με τη χρήση γλωσσών προγραμματισμού, βιβλιοθηκών, υπηρεσιών κι εργαλείων που παρέχονται από τον πάροχο. Ο καταναλωτής δεν διαχειρίζεται ή ελέγχει την υποδομή του νέφους, συμπεριλαμβανομένου του δικτύου, των εξυπηρετητών, του λειτουργικού συστήματος ή της αποθήκευσης, αλλά έχει τον έλεγχο των αναπτυχθέντων εφαρμογών κι ενδεχομένως των ρυθμίσεων για το περιβάλλον συντήρησης των εφαρμογών.

Ο χρήστης δεν χρειάζεται να έχει δικά του μηχανήματα και λογισμικό, κάτι που μειώνει σημαντικά το κόστος ανάπτυξης μιας εφαρμογής. Η ποικιλία στους παρόχους βοηθάει στην επιλογή του κατάλληλου για την εκάστοτε εφαρμογή. Παρόλα αυτά, οι περισσότερες πλατφόρμες αναπτύσσουν τις δικές τους προγραμματιστικές διεπαφές και τις δικές τους βιβλιοθήκες, εμποδίζοντας έτσι την ευελιξία των χρηστών του νέφους. Το παραπάνω ζήτημα, το οποίο ονομάζεται vendor lock-in [19], είναι ένα από τα μεγαλύτερα προβλήματα που παρουσιάζονται.

Windows Azure

Η PaaS υπηρεσία της Microsoft ονομάζεται Windows Azure [20]. Δίνει επιλογές χρήσης διάφορων γλωσσών προγραμματισμού, βάσεων δεδομένων και εργαλείων για την ανάπτυξη εφαρμογών, χωρίς να χρειάζεται ενασχόληση των χρηστών με τη συντήρηση της υποδομής. Παράλληλα, η κλιμάκωση των εφαρμογών γίνεται αυτόματα ενώ παρέχονται κι εργαλεία για τον έλεγχο των χρησιμοποιούμενων πόρων.

Οι εικονικές μηχανές επιτρέπουν στους προγραμματιστές την μετεγκατάσταση των εφαρμογών και των υποδομών, χωρίς αλλαγή στον υφιστάμενο κώδικα και μπορούν να τρέξουν είτε λειτουργικό σύστημα Windows, είτε Linux. Υποστηρίζονται σχεσιακές αλλά και NoSQL βάσεις δεδομένων, καθώς και το Hadoop Framework [21] για εφαρμογή τεχνικών εξόρυξης δεδομένων.

Επίσης, δίνεται η δυνατότητα χρήσης υβριδικών λύσεων στην παροχή υπηρεσιών, με ταυτόχρονη χρήση πόρων που παρέχονται μέσω των datacenters του Windows Azure και της ιδιωτικής υποδομής των χρηστών που αναπτύσσουν νέες εφαρμογές. Παρέχονται επίσης δυνατότητες distributed caching, τεχνική που ενισχύει την απόδοση των εφαρμογών. Οι χρήστες επικοινωνούν με τις υπηρεσίες του Windows Azure με τη χρήση του πρωτοκόλλου REST.

Google App Engine

Η Google App Engine είναι μια πλατφόρμα υπολογιστικού νέφους που προσφέρεται από τη Google ως υπηρεσία για την ανάπτυξη και φιλοξενία εφαρμογών σε δικά της datacenters [22]. Προσφέρει αυτόματη κλιμάκωση για διαδικτυακές εφαρμογές, έτσι ώστε καθώς ο αριθμός των αιτήσεων για μια εφαρμογή αυξάνεται, η App Engine κατανέμει αυτόματα περισσότερους πόρους για αυτή, ώστε να χειριστεί την πρόσθετη ζήτηση. Η Google App Engine είναι δωρεάν μέχρι ένα ορισμένο επίπεδο κατανάλωσης πόρων. Ο χρήστης θα πρέπει να πληρώσει για πρόσθετη αποθήκευση, εύρος ζώνης, ή ωρών που απαιτούνται από την εφαρμογή.

Προς το παρόν, οι υποστηριζόμενες γλώσσες προγραμματισμού είναι οι Python, Java (και κατ΄ επέκταση κι άλλες γλώσσες που τρέχουν σε JVM, όπως Groovy, JRuby, Scala, Clojure), Go και PHP. Στο μέλλον σχεδιάζεται να μπορεί να υποστηρίξει περισσότερες γλώσσες, μιας και η Google App Engine έχει αναπτυχθεί για να είναι ανεξάρτητη της γλώσσας προγραμματισμού.

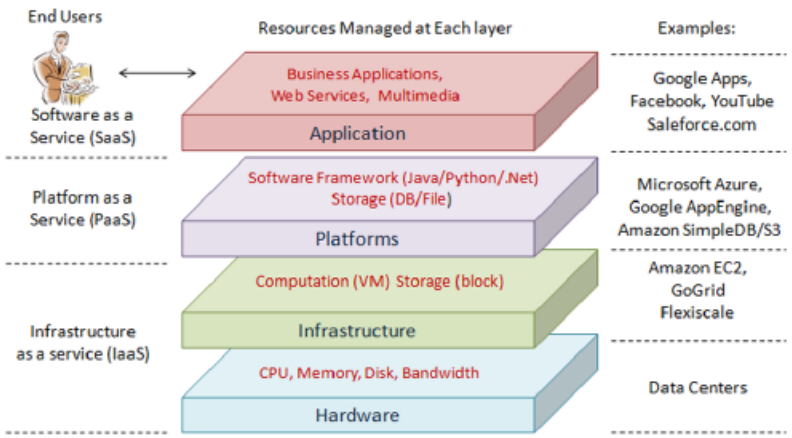
1.4.3 Λογισμικό ως υπηρεσία (SaaS)

Όταν παρέχεται λογισμικό ως υπηρεσία (Software as a Service - SaaS), ο χρήστης έχει τη δυνατότητα χρήσης των εφαρμογών του παρόχου που εκτελούνται σε υποδομή υπολογιστικού νέφους. Οι εφαρμογές αυτές είναι προσβάσιμες από διάφορες συσκευές μέσω ενός απλού γραφικού περιβάλλοντος, όπως ένας φυλλομετρητής ιστού (π.χ. web-based e-mail). Ο χρήστης δεν διαχειρίζεται, ούτε ελέγχει την υποδομή του νέφους, όπως το δίκτυο, τους εξυπηρετητές ή το λειτουργικό σύστημα, με πιθανή την εξαίρεση συγκεκριμένων ρυθμίσεων της εφαρμογής.

Οι εφαρμογές είναι έτοιμες στο νέφος και είναι προσβάσιμες μέσω ενός δικτύου. Ο πάροχος είναι υπεύθυνος για τη διάθεση των εφαρμογών και την ενημέρωση του λογισμικού. Ο χρήστης δεν χρειάζεται να εγκαταστήσει και να ενημερώνει το λογισμικό, παρά μόνο να νοικιάσει τις εφαρμογές που θέλει.

Google Apps

Ένας από τους παρόχους SaaS είναι η Google, μέσω του έτοιμου λογισμικού προς χρήση που παρέχει [23]. Οι εφαρμογές που διατίθενται μέσω του νέφους της Google λέγονται Google Apps και είναι μεταξύ άλλων τα Gmail, Google Talk, Google Docs, Google Calendar και Google Maps. Προκειμένου να χρησιμοποιήσουν οι χρήστες αυτές τις υπηρεσίες απαιτείται σύνδεση στο διαδίκτυο και για κάποιες από αυτές λογαριασμός Google, ενώ αποτελεί υποχρέωση της Google να διαχειρίζεται, να ενημερώνει και να αναβαθμίζει τις υπηρεσίες. Υπάρχει δυνατότητα παραμετροποίησης των εφαρμογών ώστε να ικανοποιούν όσο το δυνατόν καλύτερα τις ανάγκες των χρηστών και η πρόσβαση σε αυτές μπορεί να γίνει από κάθε συσκευή του χρήστη, ανεξάρτητα από το λειτουργικό της σύστημα και τον τύπο της. Παράλληλα, τα δεδομένα αποθηκεύονται σε datacenters της Google, κάτι το οποίο βέβαια εγείρει θέματα ασφάλειας και ιδιωτικότητας των προσωπικών δεδομένων.



Σχήμα 1.5: Αρχιτεκτονική υπολογιστικού νέφους

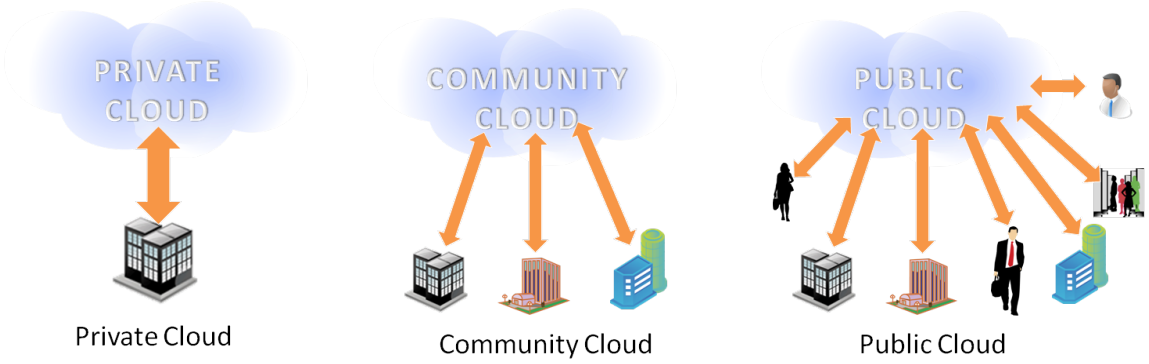
1.5 Μοντέλα ανάπτυξης νέφους

Στη συνέχεια φαίνονται τα τέσσερα κύρια μοντέλα ανάπτυξης υπολογιστικού νέφους:

- **Ιδιωτικό νέφος (Private cloud)**
  Η υποδομή του νέφους παρέχεται για αποκλειστική χρήση από ένα μόνο οργανισμό που περιλαμβάνει πολλαπλούς χρήστες (π.χ. μονάδες της επιχείρησης). Μπορεί να ανήκει, να διαχειρίζεται και να λειτουργεί από τον οργανισμό, από κάποιον τρίτο, ή από συνδυασμό αυτών.

- **Νέφος κοινότητας (Community cloud)**
  Η υποδομή του νέφους παρέχεται για αποκλειστική χρήση από μία συγκεκριμένη κοινότητα χρηστών, από οργανώσεις που έχουν κοινές ανησυχίες (π.χ. το στόχο, τις απαιτήσεις ασφαλείας, την πολιτική και τους παράγοντες συμμόρφωσης). Μπορεί να ανήκει, να διαχειρίζεται και να λειτουργεί από έναν ή περισσότερους οργανισμούς της κοινότητας, από τρίτους, ή από συνδυασμός αυτών.

- **Δημόσιο νέφος (Public cloud)**
  Η υποδομή του νέφους τροφοδοτείται για ελεύθερη χρήση από το ευρύ κοινό. Μπορεί να ανήκει, να διαχειρίζεται και να λειτουργεί από μια επιχείρηση, από ένα ακαδημαϊκό ίδρυμα, μία κυβερνητική οργάνωση, ή κάποιο συνδυασμό αυτών. Βρίσκεται στις εγκαταστάσεις του παρόχου τους νέφους.



Σχήμα 1.6: Μοντέλα ανάπτυξης νέφους

- **Υβριδικό νέφος (Hybrid cloud)**
  Η υποδομή νέφους είναι σύνθεση δύο ή περισσότερων διακριτών υποδομών νέφους (ιδιωτικών, κοινότητας ή δημόσιων) που παραμένουν διακριτές οντότητες, αλλά συνδέονται με τυποποιημένη ή ιδιόκτητη τεχνολογία που επιτρέπει τη φορητότητα των δεδομένων και των εφαρμογών (π.χ. υπολογιστικό νέφος για την εξισορρόπηση φόρτου μεταξύ δύο νεφών).

1.6 Πλεονεκτήματα

- **Ελαστικότητα**
  (Elasticity)
  Πολλές επιχειρήσεις διαθέτουν αρκετά περισσότερους πόρους από τη μέση απαίτηση τους, έτσι ώστε να διασφαλιστεί ότι η ζήτηση είναι σε θέση να ικανοποιείται σε συνθήκες μεγάλης ζήτησης. Ένα από τα βασικά χαρακτηριστικά του υπολογιστικού νέφους, η ελαστικότητα, διασφαλίζει ότι και σε ώρες αιχμής ζήτηση θα ικανοποιείται, με χαμηλό κόστος και χωρίς την επένδυση μεγάλων αρχικών κεφαλαίων για την αγορά εξοπλισμού ή τέλη έκτακτης ανάγκης για τους χρήστες.

- **Βέλτιστη χρήση πόρων**
  (Optimized server utilisation)
  Εφόσον οι περισσότερες επιχειρήσεις συνήθως υποχρησιμοποιούν τους υπολογιστικούς τους πόρους, το υπολογιστικό νέφος μπορεί να διαχειρίζεται τη χρησιμοποίηση των εξυπηρετητών και των πόρων γενικότερα με βέλτιστο τρόπο, χάρη στην ελαστικότητα και την πολιτική pay-per-use. Σύμφωνα με αυτή, μία εφαρμογή χρησιμοποιεί μόνο όσους πόρους χρειάζεται.

- **Οικονομία**
  (Cost saving)
  Τα κόστη υποδομής είναι σχεδόν πάντα σημαντικά κι αντιμετωπίζονται ως κεφαλαιακές δαπάνες (CAPEX). Ωστόσο, με τη χρήση υπολογιστικού νέφους τα κόστη υποδομής γίνονται λειτουργικά έξοδα (OPEX). Σε ορισμένες χώρες, αυτό οδηγεί σε ένα φορολογικό πλεονέκτημα κι επίσης δεν χρειάζεται τόσο μεγάλο αρχικό κεφάλαιο. Επίσης, δεν χρειάζεται διαρκής ανανέωση κι αναβάθμιση του εξοπλισμού, μιας και γι΄ αυτό φροντίζει ο πάροχος του νέφους υποδομής.

- **Μικρός κύκλος ζωής εφαρμογών**
  (Shortened development life cycle)
  Το υπολογιστικό νέφος υιοθετεί αρχιτεκτονικές προσανατολισμένες στις υπηρεσίες (SOA), για την ανάπτυξη λογισμικού, η οποία έχει πολύ μικρότερο κύκλο ζωής από την παραδοσιακή προσέγγιση. Κάθε νέα βιομηχανική εφαρμογή μπορεί να αναπτυχθεί στο διαδίκτυο, συνδέοντας λειτουργικές μονάδες.

- **Μικρός χρόνος υλοποίησης**
  (Reduced time for implementation)
  Το υπολογιστικό νέφος παρέχει υπηρεσίες επεξεργαστικής ισχύος και αποθήκευσης δεδομένων. Αυτό μπορεί να επιτευχθεί σε σχεδόν πραγματικό χρόνο, αντί για εβδομάδες ή μήνες που απαιτούνται όταν μια νέα επιχειρηματική πρωτοβουλία υλοποιείται με τον παραδοσιακό τρόπο.

1.7 Ζητήματα και προκλήσεις

Το υπολογιστικό νέφος έχει πολλά δυνατά σημεία, όπως την ευελιξία των υποδομών, την ταχύτερη ανάπτυξη των εφαρμογών και των δεδομένων, τον έλεγχο του κόστους, την προσαρμογή των πόρων του νέφους στις πραγματικές ανάγκες, τη βελτίωση της παραγωγικότητας, κλπ. Στην αρχή της δεκαετίας του 2010, η αγορά υπολογιστικού νέφους κυριαρχείται από τις υπηρεσίες λογισμικού SaaS και την υποδομή IaaS, ειδικά στο ιδιωτικό νέφος [24].

Παρά το γεγονός ότι το υπολογιστικό νέφος έχει αρχίσει να υιοθετείται από τη βιομηχανία, υπάρχουν ακόμα πολλά που είναι σε σχετικά πρώιμο στάδιο κι έτσι υπάρχουν αρκετοί παράγοντες που αποτρέπουν την ευρεία υιοθέτηση του. Μεταξύ αυτών είναι η αξιοπιστία, η διαθεσιμότητα των υπηρεσιών και των δεδομένων, η ασφάλεια, η πολυπλοκότητα, το κόστος, νομικά ζητήματα, οι επιδόσεις, η μετάβαση, η έλλειψη προτύπων, η περιορισμένη παραμετροποίηση και ζητήματα ασφαλείας.

Παρακάτω αναφέρονται κάποια σημαντικά εμπόδια για την υιοθέτηση του υπολογιστικού νέφους, και τα οποία είναι είτε τεχνικά, είτε πολιτικής κι επιχειρηματικής άποψης [25]. Πολλά από τα υπάρχοντα ζητήματα δεν έχουν αντιμετωπιστεί πλήρως, ενώ προκύπτουν όλο και νέες προκλήσεις [8] [26].

Διαθεσιμότητα υπηρεσιών (Availability of Service)

Οι οργανισμοί ανησυχούν ιδιαίτερα για τη διαθεσιμότητα των υπηρεσιών κι αυτό τους κάνει αρκετά δύσπιστους όσον αφορά το υπολογιστικό νέφος. Από την άλλη πλευρά, τα υπάρχοντα νέφη που προσφέρουν SaaS έχουν πολύ υψηλά κριτήρια στον τομέα αυτό. Για παράδειγμα, το Google Search είναι μία από τις σημαντικότερες υπηρεσίες του διαδικτύου: αν οι χρήστες επισκέπτονταν το Google για αναζήτηση και δεν ήταν διαθέσιμο, θα υπέθεταν ότι το διαδίκτυο έχει πρόβλημα. Ανάλογη διαθεσιμότητα αναμένουν οι χρήστες κι από τις νέες υπηρεσίες.

Οι μεγάλοι παροχείς υπηρεσιών ιστού χρησιμοποιούν πολλαπλούς παρόχους δικτύου, έτσι ώστε ένα πρόβλημα στις υποδομές δεν θα διακόψει την παροχή της υπηρεσίας. Αναλόγως, η μόνη πιθανή λύση για πολύ υψηλή διαθεσιμότητα είναι πολλαπλοί πάροχοι υπολογιστικού νέφους. Ακόμα κι αν μια εταιρεία έχει πολλαπλά datacenter σε διαφορετικές γεωγραφικές περιοχές με τη χρήση διαφορετικών παρόχων δικτύου, μπορεί να έχει κοινή υποδομή λογισμικού και των λογιστικών συστημάτων, ή ακόμα και να χρεωκοπήσει.

Ένα άλλο εμπόδιο όσον αφορά τη διαθεσιμότητα είναι οι Distributed Denial of Service επιθέσεις (DDoS attacks). Στους παρόχους SaaS προσφέρεται η δυνατότητα να αμυνθούν στις επιθέσεις με γρήγορη δέσμευση νέων πόρων. Αφενός ο επιτιθέμενος θα πρέπει να πληρώσει μεγαλύτερο κόστος για τα νέα μηχανήματα που δεσμεύονται για να τον εξυπηρετήσουν. Αφετέρου, ο στόχος της επίθεσης μετατοπίζεται από τον πάροχο SaaS στον πάροχο του νέφους υποδομής, ο οποίος μπορεί να απορροφήσει πιο εύκολα και πιθανώς ήδη έχει προστασία για επιθέσεις DDoS ως βασική ικανότητα.

Ασυμβατότητα μεταξύ παρόχων (Vendor Lock-In)

Οι περισσότεροι πάροχοι υπολογιστικού νέφους αναπτύσσουν δικές τους προγραμματιστικές διεπαφές APIs, που είναι συνήθως καλά τεκμηριωμένες, αλλά αφορούν μόνο τη δική τους εφαρμογή τους και ως εκ τούτου δεν είναι διαλειτουργικά. Έτσι, πολλές πλατφόρμες και υπηρεσίες, που είναι ιδιόκτητες, είναι κατασκευασμένες με συγκεκριμένα πρότυπα, εργαλεία και πρωτόκολλα κι αναπτύχθηκαν από ένα συγκεκριμένο πάροχο για το δικό του υπολογιστικό νέφος. Αυτό μπορεί να κάνει τη μετάβαση από μια ιδιόκτητη πλατφόρμα σε μία άλλη απαγορευτικά πολύπλοκη και δαπανηρή και είναι δύσκολο για τους χρήστες να εξάγουν τα προγράμματα και τα δεδομένα τους από μία πλατφόρμα σε μία άλλη [19]. Οι χρήστες είναι επίσης ευάλωτοι σε αυξήσεις των τιμών, σε προβλήματα αξιοπιστίας, ή ακόμα και σε χρεωκοπία κάπου παρόχου.

Μπορούμε να έχουμε τρεις τύπους vendor lock-in:

- **Platform lock-in:** Οι υπηρεσίες υπολογιστικού νέφους τείνουν να είναι χτισμένες σε μία από τις πολλές πλατφόρμες virtualization, για παράδειγμα στο VMWare ή το Xen. Η μετανάστευση από έναν πάροχο νέφους που χρησιμοποιεί μία πλατφόρμα σε έναν άλλο πάροχο, θα μπορούσε να είναι πολύ περίπλοκη.

- **Data lock-in:** Εφόσον το νέφος εξακολουθεί να είναι νέο, τα πρότυπα της ιδιοκτησίας, δηλαδή, δηλαδή σε ποιον ανήκουν πραγματικά τα δεδομένα εφόσον αυτά βρίσκονται στο υπολογιστικό νέφος, δεν έχουν ακόμη αναπτυχθεί. Αυτό θα μπορούσε να είναι περίπλοκο, εάν κάποιος χρήστης αποφασίσει να μετακινήσει τα δεδομένα από ένα νέφος σε ένα άλλο.

- **Tools lock-in:** Αν τα εργαλεία που δημιουργήθηκαν για τη διαχείριση ενός περιβάλλοντος νέφους δεν είναι συμβατά ανάμεσα σε διαφορετικές φυσικές αλλά και εικονικές υποδομές, αυτά τα εργαλεία θα είναι σε θέση να διαχειρίζονται μόνο δεδομένα ή εφαρμογές που βρίσκονται στο συγκεκριμένο περιβάλλον υπολογιστικού νέφους.

Μερικοί κατασκευαστές έχουν υιοθετήσει APIs άλλων και υπάρχει πλέον μια σειρά από ανοικτά πρότυπα υπό ανάπτυξη, με στόχο την επίτευξη της διαλειτουργικότητας και της φορητότητας.

Επίσης, το **ετερογενές υπολογιστικό νέφος (Heterogeneous cloud computing)** περιγράφεται ως ένα είδος νέφους που έχει τη δυνατότητα να εμποδίζει το vendor lock-in και αφορά υβριδικά μοντέλα νέφους. Η απουσία του vendor lock-in επιτρέπει στους διαχειριστές τους νέφους να επιλέξουν τα δικά τους λογισμικά ελέγχου για συγκεκριμένες εργασίες, ή να αναπτύσσουν εικονικές υποδομές σε άλλες εταιρείες, χωρίς την ανάγκη να εξεταστεί το λογισμικό ελέγχου της άλλης επιχείρησης.

Ένα ετερογενές νέφος περιλαμβάνει ιδιωτικά νέφη, δημόσια νέφη και νέφη SaaS. Το ετερογενές νέφος μπορεί να λειτουργήσει με περιβάλλοντα που δεν είναι virtualized, όπως τα παραδοσιακά datacentes. Επιτρέπει επίσης τη χρήση μονάδων, όπως λογισμικά ελέγχου, εξυπηρετητές και μέσα αποθήκευσης, από πολλούς προμηθευτές.

Ασφάλεια δεδομένων και δυνατότητα ελέγχου (Data Confidentiality and Auditability)

Σήμερα, τα περισσότερα υπολογιστικά νέφη είναι ουσιαστικά δημόσια (και όχι ιδιωτικά) δίκτυα, εκθέτοντας το σύστημα σε περισσότερες επιθέσεις [7]. Δεδομένου ότι το υπάρχει ολοένα μεγαλύτερη αύξηση στη χρήση του υπολογιστικού νέφους, είναι πιθανό ότι όλο και περισσότεροι περισσότεροι hackers βρίσκουν νέους τρόπους να εκμεταλλεύονται τρωτά σημεία του συστήματος. Η απειλή των δεδομένων μεγαλώνει από τις πολλές βασικές προκλήσεις και τους κινδύνους τους νέφους. Για να μετριαστεί η απειλή, οι ενδιαφερόμενοι θα πρέπει να επενδύσουν σε μεγάλο βαθμό στην αξιολόγηση των κινδύνων και να εξασφαλιστεί ότι το σύστημα χρησιμοποιεί κρυπτογράφηση για την προστασία των δεδομένων κι έχει καθορίσει μια αξιόπιστη βάση για την ασφάλεια της πλατφόρμας και των υποδομών. Οι ανησυχίες για την ασφάλεια θα πρέπει να αντιμετωπιστούν για να διατηρηθεί η εμπιστοσύνη στο υπολογιστικό νέφος.

Η εγγύηση της ασφάλειας των δεδομένων αναλαμβάνεται από τους παρόχους των υπηρεσιών, οι οποίοι πρέπει να αναπτύξουν εξελιγμένους μηχανισμούς ασφαλείας για τη διατήρηση των δεδομένων και την απόκρυψη της πληροφορίας από μη εξουσιοδοτημένους χρήστες. Αυτό που εγείρει αμφιβολίες όσον αφορά τα ευαίσθητα δεδομένα είναι η ακριβής χρήση, ο τρόπος αποθήκευσης και επεξεργασίας τους, η ιδιωτικότητα, ο χειρισμός των λαθών, η ανάκτηση δεδομένων, η προστασία από κακόβουλη χρήση και τα θέματα που αφορούν την πολυπελατιακότητα.

Παράλληλα, όπως συμβαίνει και με ιδιωτική αγορά υλικού, οι πελάτες μπορούν να αγοράσουν τις υπηρεσίες του νέφους για φαύλους σκοπούς. Αυτό περιλαμβάνει το σπάσιμο κωδικών και επιθέσεις τη χρήση των μηχανημάτων ή των υπηρεσιών που παρέχονται από το νέφος. Οι λύσεις που δίνουν οι πάροχοι ποικίλουν.

Πολλά από τα εμπόδια μπορούν να ξεπεραστούν άμεσα με ήδη ανεπτυγμένες τεχνολογίες, όπως κρυπτογραφημένη αποθήκευση, εικονικά τοπικά δίκτυα Virtual Local Area Networks, και με ενδιάμεσα συστήματα του δικτύου (π.χ. firewalls, φίλτρα πακέτων). Για παράδειγμα, η κρυπτογράφηση των δεδομένων πριν από την αποθήκευση σε ένα νέφος μπορεί να είναι ακόμη πιο ασφαλής από ό,τι τα μη κρυπτογραφημένα δεδομένα σε ένα τοπικό datacenter.

Ένα σχετικό θέμα είναι ότι πολλές χώρες έχουν νόμους που απαιτούν από τους παρόχους SaaS να κρατούν τα στοιχεία των πελατών και τα πνευματικά δικαιώματα των δεδομένων εντός των εθνικών συνόρων. Ομοίως, ορισμένες επιχειρήσεις μπορεί να μη θέλουν μια χώρα να αποκτάει πρόσβαση στα δεδομένα τους, μέσω του δικαστικού συστήματος. Το υπολογιστικό νέφος δίνει στους παρόχους και τους χρήστες SaaS μεγαλύτερη ελευθερία για την τοποθεσία των δεδομένων τους. Για παράδειγμα, η Amazon S3 παρέχει υπηρεσίες που βρίσκονται στις ΗΠΑ και στην Ευρώπη, επιτρέποντας στους παρόχους να διατηρούν τα δεδομένα τους σε όποιο μέρος επιλέξουν.

Συμφόρηση μεταφοράς δεδομένων (Data Transfer Bottlenecks)

Οι εφαρμογές γίνονται ολοένα και πιο απαιτητικές σε όγκο δεδομένων. Αν υποθέσουμε ότι η μεταφορά ανά μεταφερόμενο terabyte είναι από 100$ έως 150$, οι δαπάνες αυτές μπορούν να αυξηθούν σημαντικά, κάνοντας το κόστος της μεταφορά δεδομένων ένα ουσιώδες ζήτημα. Οι χρήστες και οι πάροχοι υπολογιστικού πρέπει να σκεφτούν τις συνέπειες της κίνησης δεδομένων σε κάθε επίπεδο του συστήματος, αν θέλουν να ελαχιστοποιήσουν το κόστος.

Μία λύση ώστε να ξεπεραστεί το υψηλό κόστος των μεταφορών του διαδικτύου είναι η μεταφορά φυσικών δίσκων. Έχει διαπιστωθεί ότι ο φθηνότερος τρόπος για να σταλούν πολλά δεδομένα είναι να στείλει κάποιος φυσικούς δίσκους ή ακόμη και ολόκληρους υπολογιστές. Αν και δεν υπάρχουν εγγυήσεις από τους κατασκευαστές των δίσκων τα δεδομένα μεταφέρονται πολύ αξιόπιστα με αυτό τον τρόπο.

Μια δεύτερη λύση είναι να βρεθούν άλλοι λόγοι ώστε να είναι ελκυστική η φύλαξη δεδομένων στο νέφος και να ενεργοποιηθούν νέες υπηρεσίες. Ένα παράδειγμα μπορεί να είναι οι υπηρεσίες backup και η φύλαξη δεδομένων έξω από μία ιστοσελίδα. Έτσι, το κόστος εντός του δικτύου θα ήταν μικρότερο, μιας και τα δεδομένα θα υπήρχαν σε περισσότερες τοποθεσίες. Μία λύση δίνει το Vision Cloud, με υιοθέτηση των portlets [27].

Μία τρίτη λύση είναι να μειωθεί γρηγορότερα το κόστος μεταφοράς σε δίκτυα ευρείας ζώνης wide-area networks - WAN. Σύμφωνα με εκτιμήσεις, τα δύο τρίτα του κόστους του εύρος ζώνης σε WAN είναι το κόστος των δρομολογητών, ενώ μόνο το ένα τρίτο είναι το κόστος της οπτικής ίνας.

Αβεβαιότητα επιδόσεων (Performance Unpredictability)

Όπως φαίνεται από τη χρήση του υπολογιστικού νέφους μέχρι σήμερα, οι πολλαπλές εικονικές μηχανές μπορούν να μοιραστούν επεξεργαστική ισχύ και την κύρια μνήμη εκπληκτικά καλά. Παρόλα αυτά, η κοινή χρήση των συστημάτων εισόδου/εξόδου I/O κατανομή είναι πιο προβληματική.

Μια λύση είναι η βελτίωση των αρχιτεκτονικών και των λειτουργικών συστημάτων έτσι ώστε να εικονικοποιεί πιο αποτελεσματικά τις διακοπές interrupts και τα κανάλια I/O. Ένας λόγος για να είμαστε αισιόδοξοι είναι ότι τα λειτουργικά συστήματα ξεπέρασαν σε μεγάλο βαθμό τα προβλήματα αυτά στη δεκαετία του 1980, οπότε έχουμε κάποια επιτυχή παραδείγματα από τα οποία μπορούμε να μάθουμε.

Μια άλλη λύση είναι η μείωση των παρεμβολών I/O, λόγω χρήσης μνήμης flash. Αυτό το είδος αποθήκευσης είναι μνήμη ημιαγωγών που διατηρεί πληροφορίες ακόμα κι όταν απενεργοποιηθεί, όπως και οι μηχανικοί σκληροί δίσκοι. Δεδομένου όμως ότι δεν έχει κινούμενα μέρη, είναι πολύ πιο γρήγορα προσβάσιμη (μs vs ms) και χρησιμοποιεί λιγότερη ενέργεια. Επίσης, μπορεί να αντέξει πολλές περισσότερες αιτήσεις I/O ανά δευτερόλεπτο κι ανά gigabyte αποθήκευσης σε σχέση με τους σκληρούς δίσκους.

Έτσι, εικονικές μηχανές με τυχαία συγκρουόμενες αιτήσεις I/O θα μπορούσαν να συνυπάρξουν καλύτερα στον ίδιο φυσικό υπολογιστή σε σχέση με μηχανικούς δίσκους.

Η εικονικοποίηση μπορεί να προσφέρει σημαντικά οφέλη στο υπολογιστικό νέφος και η δυνατότητα **μετανάστευσης των εικονικών μηχανών (Virtual machine migration)** είναι ιδιαίτερα σημαντική για την εξισορρόπηση του φορτίου στο datacenter. Η μετανάστευση VMs έχει εξελιχθεί αρκετά με τις νέες τεχνικές της μετανάστευσης διεργασιών. Έχει ήδη υλοποιηθεί δυνατότητα μετακίνησης VMs που περιλαμβάνει εξαιρετικά σύντομους χρόνους εκτός λειτουργίας, που κυμαίνονται από δεκάδες χιλιοστά του δευτερολέπτου ως ένα δευτερόλεπτο (live migration) [28]. Παρόλα αυτά, η ανίχνευση hotspots φόρτου εργασίας και η εκκίνηση της μετανάστευσης δεν έχει την δυνατότητα να ανταποκρίνεται στις αιφνίδιες αλλαγές φόρτου εργασίας.

Επεκτασιμότητα μέσων αποθήκευσης (Scalable Storage)

Όπως προαναφέρθηκε, οι τρεις βασικές ιδιότητες των οποίων ο συνδυασμός κάνει το υπολογιστικό νέφος ιδαίτερα ελκυστικό είναι η δέσμευση και αποδέσμευση πόρων ανάλογα με τη ζήτηση, το κόστος ανάλογα με τη χρήση και η αίσθηση άπειρης δυνατότητας επεξεργαστικής ισχύος. Ενώ είναι απλό όταν αναφερόμαστε σε επεξεργαστική ισχύ, είναι λιγότερο προφανές στο πως μπορεί να εφαρμοστεί στη μόνιμη αποθήκευση.

Μία πρόκληση, η οποία εξακολουθεί να είναι ένα ανοικτό ερευνητικό πρόβλημα, είναι να δημιουργηθεί ένα σύστημα αποθήκευσης που όχι μόνο θα ικανοποιεί την πολυπλοκότητα των απαιτούμενων δομών δεδομένων και τις επιδόσεις που εγγυώνται, αλλά και το συνδυασμό με τα πλεονεκτήματα του νέφους, όπως την κλιμάκωση των πόρων νέφους on-demand, την επεκτασιμότητα, τη διατήρηση των δεδομένων και την υψηλή διαθεσιμότητα.

Σφάλματα σε κατανεμημένα συστήματα μεγάλης κλίμακας (Bugs in Large-Scale Distributed Systems)

Μία από τις δύσκολες προκλήσεις στο υπολογιστικό νέφος είναι η διόρθωση λαθών σε κατανεμημένα συστήματα μεγάλης κλίμακας. Ένα συχνό πρόβλημα είναι ότι αυτά τα σφάλματα δεν μπορούν να αναπαραχθούν σε μικρότερα συστήματα κι έτσι η αποσφαλμάτωση (debugging) πρέπει να πραγματοποιείται σε πραγματικά datacenters.

Μία λύση μπορεί να είναι να βασιστούμε στα VMs. Πολλοί παραδοσιακοί πάροχοι SaaS ανέπτυξαν τις υποδομές τους, χωρίς τη χρήση VMs, είτε επειδή τα VMs άρχισαν να χρησιμοποιουνται εκτενώς αργότερα ή επειδή είχαν θεωρήσει ότι δεν τα VMs δεν είχαν την ανάλογη απόδοση.

Γρήγορη κλιμάκωση/ελαστικότητα (Scaling Quickly)

Ένα από τα βασικά χαρακτηριστικά του υπολογιστικού νέφους είναι η δυνατότητα της δέσμευσης και αποδέσμευσης πόρων on-demand. Ο στόχος των παρόχων υπηρεσιών σε αυτή την περίπτωση είναι να δεσμεύει και να αποδεσμεύει πόρους από το νέφος ώστε να ικανοποιήσει τους στόχους για επίπεδο υπηρεσίας (Service Level Objectives - SLOs), ελαχιστοποιώντας παράλληλα το κόστος λειτουργίας. Ωστόσο, δεν είναι προφανές το πώς ένας πάροχος υπηρεσιών μπορεί να πετύχει αυτό το στόχο. Ειδικότερα, δεν είναι εύκολο να προσδιοριστεί η αντιστοίχιση των (SLOs), όπως οι απαιτήσεις για ποιότητα υπηρεσιών Quality of Service - QoS, με τις χαμηλού επιπέδου απαίτησεις για πόρους, όπως απαιτήσεις για επεξεργαστική ισχύ και μνήμη.

Επιπρόσθετα, η πολιτική pay-as-you-go σίγουρα εφαρμόζεται στην αποθήκευση δεδομένων και στη χρήση εύρους ζώνης του δικτύου και βασικά μετρώνται τα byte που χρησιμοποιούνται. Όσον αφορά την υπολογιστική ισχύ, η πολιτική είναι λίγο διαφορετική, ανάλογα με το επίπεδο στο virtualization. Για παράδειγμα, η Google AppEngine αυξομειώνει τους πόρους ανάλογα με τις αυξήσεις και τις μειώσεις του φόρτου, και οι χρήστες χρεώνονται με βάση τους χρησιμοποιούμενους κύκλους. Το AWS χρεώνει με την ώρα για τον αριθμό των στιγμιοτύπων που έχει κάποιος, ακόμη και αν το σύστημα είναι αδρανές.

Μία λύση στα παραπάνω είναι να υπάρχει αυτόματη κλιμάκωση του νέφους, προκειμένου να εξοικονομηθούν χρήματα, αλλά χωρίς να παραβιάζεται και το QoS. Επίσης, για να επιτευχθεί υψηλή ευελιξία και να υπάρχει προσαρμογή σε ταχύτατες διακυμάνσεις της ζήτησης, οι αποφάσεις πρόβλεψης των απαιτούμενων πόρων (resource provisioning) πρέπει να γίνουν κατά τη λειτουργία του συστήματος.

Πέρα απ΄ όλα αυτά, δεδομένου ότι ένα αδρανές υπολογιστής χρησιμοποιεί περίπου τα δύο τρίτα της δύναμης ενός υπολογιστή σε χρήση, η προσεκτική χρήση των πόρων μπορεί να μειώσει την επίδραση των datacenter στο περιβάλλον. Οι πάροχοι υπολογιστικού νέφους ήδη είναι προσεκτικοί όσον αφορά την κατανάλωση πόρων. Με την επιβολή κόστους ανά ώρα κι ανά byte, οι προγραμματιστές ενθαρρύνονται να δώσουν προσοχή στην αποδοτικότητα και τη δέσμευση πόρων μόνο όταν είναι απαραίτητο.

Η δυναμική παροχή πόρων για εφαρμογές στο διαδίκτυο έχει μελετηθεί εκτενώς στο παρελθόν. Οι προσεγγίσεις αυτές συνήθως περιλαμβάνουν:

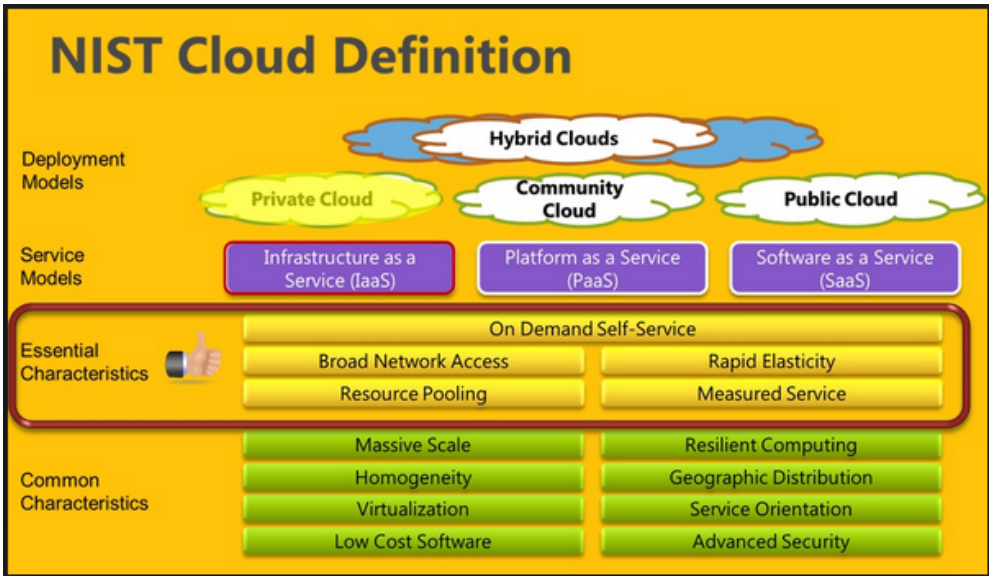
1) την κατασκευή ενός μοντέλου απόδοσης των εφαρμογών, που προβλέπει τον αριθμό των στιγμιοτύπων της εφαρμογής, που απαιτούνται προκειμένου να ικανοποιούνται οι απαιτήσεις για QoS.

2) την περιοδική πρόβλεψη μελλοντικής ζήτησης και τον προσδιορισμό των απαιτούμενων πόρων με βάση το παραπάνω μοντέλο. Το μοντέλο αποδόσεων μπορεί να κατασκευαστεί χρησιμοποιώντας διάφορες τεχνικές, όπως τη θεωρίας αναμονής, τη θεωρία ελέγχου και τη μηχανική μάθηση (machine learning).

3) την αυτόματη κατανομή πόρων χρησιμοποιώντας τις προβλεπόμενες απαιτήσεις.

Άδειες χρήσης λογισμικού (Software Licensing)

Οι τρέχουσες άδειες χρήσης λογισμικού περιορίζουν συνήθως τους υπολογιστές ως προς το λογισμικό μπορούν να τρέξουν. Οι χρήστες πληρώνουν για το λογισμικό και στη συνέχεια να καταβάλουν ένα ετήσιο τέλος συντήρησης. Ως εκ τούτου, πολλοί πάροχοι υπολογιστικού νέφους αρχικά στηρίχθηκαν σε λογισμικό ανοικτού κώδικα (open-source software), εν μέρει γιατί το μοντέλο παραχώρησης άδειας εμπορικού λογισμικού δεν ταιριάζει με την έννοια του δημόσιου νέφους. Άρα, θα πρέπει είτε το λογισμικό ανοιχτού κώδικα να παραμείνει δημοφιλές ή οι εμπορικές επιχειρήσεις λογισμικού να αλλάξουν τη δομή των αδειών τους ώστε να ταιριάζουν καλύτερα με την έννοια του υπολογιστικού νέφους.



Σχήμα 1.7: Σύνοψη: Χαρακτηριστικά του υπολογιστικού νέφους

Κεφάλαιο 2

Η ελαστικότητα στο νέφος

2.1 Ορισμός

Στο υπολογιστικό νέφος, η ελαστικότητα μπορεί να οριστεί ως εξής [29]:

> ''Ελαστικότητα είναι η ικανότητα ένος συστήματος να προσαρμόζεται στις αλλαγές φόρτου που παρουσιάζονται, δεσμεύοντας κι αποδεσμεύοντας πόρους αυτόνομα, έτσι ώστε σε κάθε χρονική στιγμή οι διαθέσιμοι πόροι να ανταποκρίνονται όσο το δυνατόν καλύτερα στην τρέχουσα ζήτηση.''

Σε αυτό το σημείο θα πρέπει να επισημάνουμε ότι η ελαστικότητα είναι μία αρκετά διαφορετική έννοια από την κλιμάκωση (scalability), αν και πολλοί τις συγχέουν λανθασμένα. Η κλιμάκωση αποτελεί προϋπόθεση για την ελαστικότητα, αλλά δεν έχει τις χρονικές διαστάσεις της ελαστικότητας. Πιο αναλυτικά:

Κλιμάκωση είναι η ικανότητα μιας εφαρμογής να ανταποκρίνεται σε αυξημένο φόρτο εργασίας κάνοντας χρήση πρόσθετων πόρων. Έτσι, ένα σύστημα μπορεί να συγκεντρώσει πόρους έτσι ώστε η εφαρμογή να συνεχίζει να λειτουργεί ομαλά. Αυτό μπορεί να γίνει με δύο τρόπους:

- κάνοντας ένα συγκεκριμένο στοιχείο του συστήματος ισχυρότερο ή ασθενέστερο (vertical scalability). Αυτό μπορεί να γίνει για παράδειγμα αυξάνοντας/μειώνοντας τη μνήμη RAM ή προσθέτοντας/αφαιρώντας επεξεργαστές (CPU) από ένα VM.

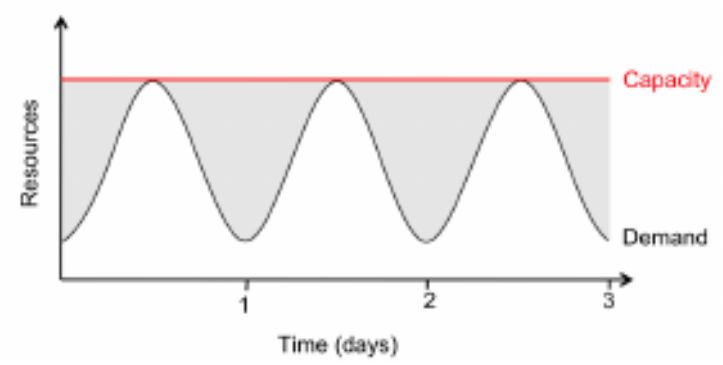
- προσθέτοντας/αφαιρώντας VMs (horizontal scalability).

Η **ελαστικότητα** από την άλλη πλευρά, είναι η ικανότητα να κλιμακώνεται μια υποδομή εντός λεπτών ή ακόμη και δευτερολέπτων (αντί για ημέρες ή εβδομάδες), ανάλογα με τη ζήτηση, αποφεύγοντας έτσι την υποτροφοδότηση (under-provisioning) ή την υπερτροφοδότηση πόρων (over-provisioning). Μία ελαστική πλατφόρμα υπολογιστικού νέφους μπορεί να χειριστεί αιφνίδιες κι απρόβλεπτες αλλαγές στο φόρτο του συστήματος και ουσιαστικά περιγράφει το πόσο καλά μπορεί ένα σύστημα να προσαρμοστεί στο φόρτο εργασίας σε πραγματικό χρόνο.

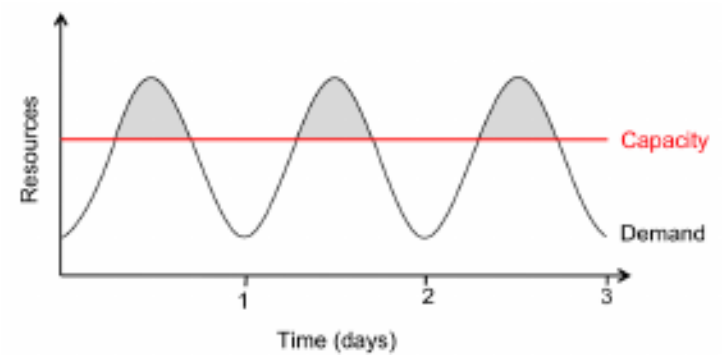
2.2 Χαρακτηριστικά της ελαστικότητας

Η ελαστικότητα ουσιαστικά στοχεύει στο όσο το δυνατόν καλύτερο ταίριασμα των πόρων που διατίθενται με την ποσότητα των πόρων που απαιτούνται στην πραγματικότητα για μια υπηρεσία, αποφεύγοντας την υπερτροφοδότηση ή την υποτροφοδότηση.

Η **υπερτροφοδότηση** (overprovisioning) εμφανίζεται όταν διατίθενται περισσότεροι πόροι από ό,τι απαιτείται. Σε αυτή την περίπτωση, ακόμα κι αν το φορτίο αιχμής έχει προβλεφθεί ορθά, χωρίς ελαστικότητα χάνουμε πόρους κατά το διάστημα που δεν έχουμε υψηλό φόρτο. Αντίθετα, **υποτροφοδότηση** (underprovisioning) έχουμε όταν διατίθενται λιγότεροι πόροι από ό,τι απαιτείται, με αποτέλεσμα να μην εξυπηρετούνται οι χρήστες αποδοτικά.



Σχήμα 2.1: Υπερτροφοδότηση



Σχήμα 2.2: Υποτροφοδότηση

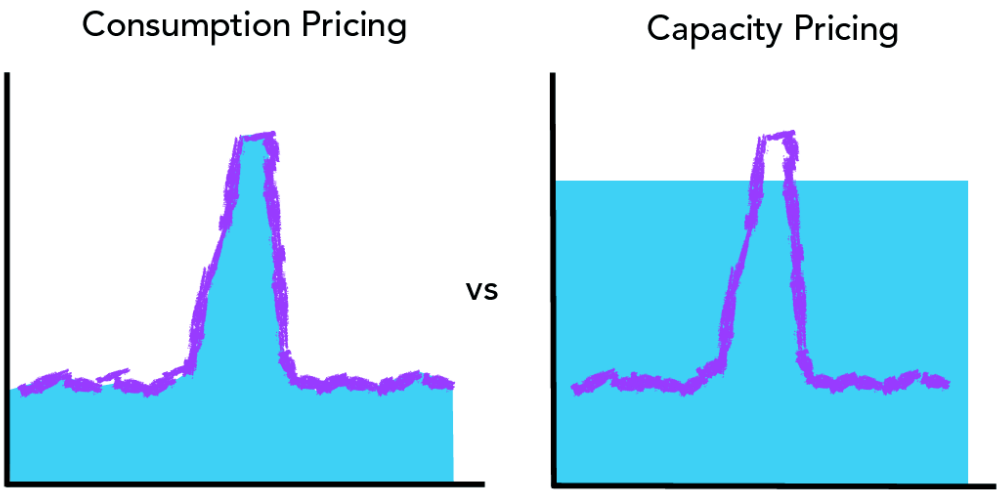
Η ελαστικότητα είναι εκείνο το χαρακτηριστικό του υπολογιστικού νέφους, το οποίο κάνει ακόμα πιο σαφή το διαχωρισμό από άλλες έννοιες, όπως το υπολογιστικό πλέγμα.

Τα κυριότερα χαρακτηριστικά γνωρίσματα της ελαστικότητας είναι τα παρακάτω:

- **Γρήγορη εγκατάσταση εφαρμογών κι εύκολη κλιμάκωση:** Ο κόσμος των επιχειρήσεων απαιτεί γρήγορη παράδοση των νέων δυνατοτήτων και συχνές ενημερώσεις σε εφαρμογές, διεργασίες, διεπαφές επικοινωνίας, κλπ. Με το υπολογιστικό νέφος, υπάρχει η δυνατότητα δέσμευσης κι αποδέσμευσης πόρων υποδομής, όπως εικονικές μηχανές, μέσων αποθήκευσης και εύρους ζώνης, μέσα σε λίγα λεπτά και με ελάχιστες κινήσεις, αντί να απαιτεί μήνες για την προμήθεια, εγκατάσταση και διαμόρφωση υλικού και λογισμικού (που επιτείνεται από πιθανή γραφειοκρατία) που μπορεί να εμποδίσει περαιτέρω την καινοτομία.

  Όταν υπάρχει ανάγκη για πόρους, η υποδομή τους δεσμεύει για να εξυπηρετήσει το φόρτο, ενώ όταν η κίνηση υποχωρεί στα φυσιολογικά επίπεδα, μπορεί εξίσου εύκολα να τους αποδεσμεύσει και να κρατήσει την ελάχιστη ποσότητα πόρων που χρειάζεται μία εφαρμογής.

- **Χρέωση μόνο για τους πόρους που απαιτούνται:** Με το νέφος, ο καθένας πληρώνει για όσους πόρους χρησιμοποιεί. Ο πάροχος μιας εφαρμογής μπορεί να αντιμετωπίσει απρόβλεπτα φορτία αιχμής, ενώ παράλληλα θα συνεχίσει να πληρώνει για τους ακριβείς πόρους που χρησιμοποιεί, χωρίς να χρειάζεται να επενδύσει σε υλικό που θα παραμένει σε αχρηστία κατά το μεγαλύτερο μέρος του κύκλου ζωής της εφαρμογής. Ένα δευτερεύον όφελος για τους επαγγελματίες της ανάπτυξης εφαρμογών είναι ότι μπορούν να δημιουργούν αντίγραφα τους συστήματος για την ανάπτυξη και τη δοκιμή των εφαρμογών τους, χωρίς να χρειάζεται να επενδύσουν σε επιπλέον υλικό.



Σχήμα 2.3: Χρέωση στο νέφος

- **Διαφύλαξη κεφαλαίου:** Η αγορά της κατάλληλης ποσότητας υλικού, όπως εξυπηρετητές, μέσα αποθήκευσης και συσκευές δικτύου είναι συχνά μία επένδυση που οδηγεί σε υπερβολικές δαπάνες και χρησιμεύει μόνο σε περιπτώσεις που ο φόρτος του συστήματος είναι αυξημένος. Τα δημόσια νέφη απαιτούν μηδενική αρχική επένδυση σε υλικό, ενώ ακόμα και τα ιδιωτικά νέφη που αρχικά χρειάζονται σημαντικές επενδύσεις, μπορούν να αποδώσουν πολλά στο μέλλον. Πλέον, το υλικό του υπολογιστή χάνει την αξία του πολύ γρήγορα. Έτσι, είναι φανερό πως η ελαστικότητα είναι αυτή που μετατρέπει τις κεφαλαιακές δαπάνες μιας σύγχρονης επιχείρισης σε λειτουργικές δαπάνες, μιας και δεν χρειάζεται πλέον η αγορά υπολογιστικού υλικού, αλλά η δέσμευσή του όταν ο φόρτος του συστήματος είναι μεγάλος.
- **Ψευδαίσθηση των άπειρων πόρων:** Η ψευδαίσθηση αυτή οφείλεται και πάλι στην ελαστικότητα. Οι πάροχοι υπηρεσιών νομίζουν ότι έχουν ανεξάντλητους πόρους υποδομής, χάρη στην ικανότητα αυτή του νέφους να αυξομειώνει τους πόρους όσο χρειάζεται, ώστε να ανταποκρίνεται στο φόρτο του συστήματος, ακόμα και σε ακραίες καταστάσεις.

2.3 Προβλήματα

Παρόλα αυτά τα πλεονεκτήματα, υπάρχουν ορισμένα προβλήματα, λόγω των οποίων οι περισσότερες εμπορικές πλατφόρμες δεν μπορούν να εκμεταλλευτούν τα θετικά στοιχεία της ελαστικότητας και να επωφεληθούν την υψηλή διαθεσιμότητα, την απόδοση και την κλιμάκωση [30]. Αντ΄ αυτού, οι περισσότερες επιχειρήσεις δεσμεύουν τους πόρους στατικά βάσει της μέγιστης και της αναμενόμενης ζήτησης κι όχι παρακολουθώντας σε πραγματικό χρόνο το φόρτο του συστήματος κι αυτοματοποιώντας των πόρων υποδομής. Τέτοια προβλήματα είναι:

- Η αδυναμία γρήγορης δέσμευσης πόρων για την εξυπηρέτηση του φόρτου.
- Η αδυναμία γρήγορης αποδέσμευσης πόρων, με αποτέλεσμα άσκοπη χρέωση για πόρους που δεν χρησιμοποιούνται.
- Η αδυναμία των σχεσιακών βάσεων δεδομένων να επωφεληθούν από περισσότερες από μία εικονικές μηχανές.
- Δεν υπάρχουν γενικής χρήσης προγραμματιστικές διεπαφές (APIs) για πλατφόρμες υπολογιστικού νέφους, οι οποίες να βοηθούν την κατασκευή εφαρμογών ώστε να τρέχουν ελαστικά σε ένα νέφος.
- Η χρήση συμβατικών αρχιτεκτονικών με κάποιες μικρές τροποποιήσεις.

Η πιο ουσιαστική λύση στα παραπάνω προβλήματα, είναι ο σχεδιασμός μιας πλατφόρμας ελαστικών εφαρμογών που αξιοποιούν πλήρως τα πλεονεκτήματα της ελαστικότητας του υπολογιστικού νέφους, με τη βοήθεια εργαλείων, υπηρεσιών, προγραμματιστικών διεπαφών και συγκεκριμένων κανόνων.

2.4 Η ελαστικότητα σε εμπορικά συστήματα

Παρακάτω παρουσιάζονται κάποιοι πάροχοι υπηρεσιών υπολογιστικού νέφους, ενώ γίνεται και ιδιαίτερη αναφορά στις δυνατότητες ελαστικότητάς τους.

2.4.1 Amazon EC2

Το Amazon EC2 είναι μία υπηρεσία υπολογιστικού νέφους, που παρέχεται από την Amazon και παρέχει IaaS στους χρήστες. Οι χρήστες, μπορούν να δημιουργούν εικονικές μηχανές και να αναπτύσσουν δικές τους υπηρεσίες. Έτσι, επιλέγουν μία Amazon Machine Image, με τη βοήθεια της οποίας μπορούν να δημιουργήσουν μία εικονική μηχανή στο Amazon EC2 και στη συνέχεια επιλέγουν ρυθμίσεις δικτύου, ασφαλείας και λειτουργίας της κάθε εικονικής μηχανής. Παρέχεται η δυνατότητα επιλογής του φυσικού χώρου στον οποίο θα τρέξουν οι εικονικές μηχανές.

Ένα στιγμιότυπο του Amazon EC2 μοιάζει πολύ με φυσικό υλικό, μιας και οι χρήστες μπορούν να ελέγχουν σχεδόν ολόκληρη την στοίβα λογισμικού, από τον πυρήνα και πάνω. Ως εκ τούτου, το επίπεδο της αφαιρετικότητας που παρουσιάζεται στον προγραμματιστή είναι πολύ χαμηλό και ο χρήστης έχει πλήρη διαχείριση των πόρων. Το API που έχει είναι πολύ απλό και αρκούν μερικές δεκάδες κλήσεις του API ώστε να δεσμευτούν και να ρυθμιστούν οι εικονικοί πόροι. Δεν υπάρχει περιορισμός σχετικά με τις εφαρμογές που μπορούν να χρησιμοποιηθούν. Το χαμηλό επίπεδο εικονικοποίησης (συχνότητα CPU, μέσα αποθήκευσης, συνδεσιμότητα σε επίπεδο IP) επιτρέπει στους προγραμματιστές να γράψουν κώδικα για ό,τι θέλουν.

Από την άλλη πλευρά, αυτό καθιστά δύσκολο για την Amazon να προσφέρει ελαστικότητα, γιατί η σημασιολογία που συνδέεται με τα διάφορα θέματα διαχείρισης της κατάστασης του συστήματος είναι ιδιαίτερα εξαρτημένη από την εκάστοτε εφαρμογή. Έτσι, μπορεί οι χρήστες να έχουν άμεσα διαθέσιμους όσους πόρους θέλουν χωρίς αναμονή, αλλά αυτό αποτελεί μια διαδικασία χειροκίνητη. Για την επίτευξη οριζόντιας κλιμάκωσης κι ελαστικότητας πρέπει να φτιάξει κάποιος χρήστης ένα σύμπλεγμα από VMs και να τα παραμετροποιήσει ο ίδιος ανάλογα με τις ανάγκες του.

2.4.2 Google AppEngine

Το πακέτο Google AppEngine της Google είναι ένα σύστημα που παρέχει SaaS. Οι εφαρμογές που παρέχονται λέγονται Google Apps, μερικές από τις οποίες είναι οι Gmail, Google Talk, Google Docs και Google Maps. Οι χρήστες χρειάζεται να είναι απλά συνδεδεμένοι στο διαδίκτυο και να έχουν πρόσβαση με κάποιο Google account, ενώ η διαχείριση, ενημέρωση κι αναβάθμιση αυτών των εφαρμογών είναι υποχρέωση της Google. Παρέχει τις ίδιες υπηρεσίες προστασίας της ιδιωτικής ζωής και ασφάλειας των δεδομένων που εφαρμόζονται σε όλες τις εφαρμογές της Google.

Η Google AppEngine βρίσκεται στο άλλο άκρο σε σχέση με το Amazon EC2, από την άποψη της αφαιρετικότητας και τις διαχείρισης των πόρων. Η AppEngine α-

πευθύνεται αποκλειστικά σε παραδοσιακές διαδικτυακές εφαρμογές, επιβάλλοντας μιας δομή που διαχωρίζει ξεκάθαρα το επίπεδο υπολογισμού και το επίπεδο αποθήκευσης. Προφανώς όμως, η AppEngine δεν είναι κατάλληλη για γενική υπολογιστική χρήση, παρά μόνο φιλοξενεί κάποιες υπηρεσίες ιστού.

Επιπλέον, οι AppEngine εφαρμογές βασίζονται σε αιτήσεις κι αποκρίσεις και ως εκ τούτου είναι πολύ προβλέψιμο το πόσος υπολογιστικός χρόνος απαιτείται για την εξυπηρέτηση ενός συγκεκριμένου αιτήματος. Έτσι, διαθέτει εντυπωσιακή αυτόματη κλιμάκωση και υψηλή διαθεσιμότητα των υπηρεσιών. Όμως, αυτή η αυτόματη κλιμάκωση είναι διαφανής για τους παρόχους εφαρμογών. Δεν υπάρχει δυνατότητα για έναν προγραμματιστή να γράψει τους δικούς του κανόνες για την κλιμάκωση, ανάλογα με τις ειδικές ανάγκες της δικής του εφαρμογής.

2.4.3 Windows Azure

Η πλατφόρμα Windows Azure της Microsoft είναι η PaaS υπηρεσία της Microsoft. Δίνει επιλογές χρήσης πολλών γλωσσών προγραμματισμού, βάσεων δεδομένων κι εργαλείων ανάπτυξης εφαρμογών. Αποτελείται από τρία κύρια στοιχεία που παρέχουν ένα συγκεκριμένο σύνολο υπηρεσιών στους χρήστες του νέφους. Αυτά είναι ένα περιβάλλον βασισμένο στα Windows για την εκτέλεση εφαρμογών και την αποθήκευση δεδομένων, υπηρεσίες βάσης δεδομένων που βασίζονται στον SQL server και υπηρεσίες κατανεμημένης υποδομής που παρέχουν μία και μόνο επαλήθευση της ταυτότητας σε όλες τις εφαρμογές.

Η πλατφόρμα αυτή είναι κάτι ενδιάμεσο σημείο σε αυτό το φάσμα της ευελιξίας έναντι της άνεσης για τον προγραμματιστή. Οι Azure εφαρμογές γραμμένες χρησιμοποιώντας τις βιβλιοθήκες .NET, και μετατρέπονται σε Common Language Runtime, μια γλώσσα ανεξάρτητη από περιβάλλον στο οποίο εκτελείται. Το σύστημα υποστηρίζει γενική υπολογιστική χρήση, αντί για μία μόνο κατηγορία εφαρμογών. Οι χρήστες έχουν τη δυνατότητα επιλογής της γλώσσας, αλλά δεν μπορούν να επιλέξουν λειτουργικό σύστημα ή περιβάλλον εκτέλεσης. Οι βιβλιοθήκες παρέχουν ένα βαθμό αυτόματης ρύθμισης των παραμέτρων του δικτύου, κλιμάκωσης κι ελαστικότητας, αλλά απαιτούν από τον προγραμματιστή να καθορίσει κάποιες ιδιότητες της εφαρμογής του.

Έτσι, η πλατφόρμα αυτή παρέχει δυνατότητες ελαστικότητας, αλλά αυτή προσφέρεται με τη βοήθεια κανόνων μέσω ενός αρχείου ρυθμίσεων που ορίζεται από τους χρήστες.

Κεφάλαιο 3

Αρχιτεκτονικά μοντέλα δικτυακών συστημάτων

3.1 Simple Object Access Protocol (SOAP)

Το SOAP, είναι μια προδιαγραφή πρωτοκόλλου για την ανταλλαγή δομημένων πληροφοριών ανάμεσα σε δικτυακές υπηρεσίες δικτύων υπολογιστών [31]. Στηρίζεται στο XML Information Set για την μορφή του μηνύματος με τις πληροφορίες. Συνήθως βασίζεται σε άλλα πρωτόκολλα του στρώματος εφαρμογής, και κυρίως στο HTTP ή το SMTP, για τη διαπραγμάτευση και τη μετάδοση μηνυμάτων.

Ουσιαστικά είναι ένας τρόπος επικοινωνίας κι ένας μηχανισμός για την ανταλλαγή πληροφοριών, μεταξύ ενός προγράμματος που εκτελείται σε ένα λειτουργικό σύστημα κι ενός άλλου προγράμματος που εκτελείται στο ίδιο ή σε άλλο λειτουργικό σύστημα, χρησιμοποιώντας το πρωτόκολλο HTTP του διαδικτύου και της γλώσσας σήμανσης XML. Δεδομένου ότι τα πρωτόκολλα ιστού εγκατασταθεί και είναι διαθέσιμα για χρήση από όλες τις μεγάλες πλατφόρμες λειτουργικών συστημάτων, το HTTP και το XML παρέχουν ήδη μία εύκολη λύση στο πρόβλημα της επικοινωνίας μεταξύ προγραμμάτων που τρέχουν σε διαφορετικά λειτουργικά συστήματα σε ένα δίκτυο. Το πρωτόκολλο SOAP καθορίζει ακριβώς πως να κωδικοποιήσει μια κεφαλίδα HTTP και ένα αρχείο XML, έτσι ώστε ένα πρόγραμμα σε έναν υπολογιστή μπορεί να καλέσει ένα πρόγραμμα σε έναν άλλο υπολογιστή και να μεταδώσει πληροφορίες. Διευκρινίζει επίσης πως το πρόγραμμα που καλείται μπορεί να επιστρέψει μια απάντηση.

Το SOAP είναι αρκετά ευέλικτο ώστε να επιτρέπει την χρήση διαφορετικών πρωτοκόλλων μεταφοράς. Πιο συνιθισμένη είναι η χρήση του HTTP ως πρωτοκόλλου μεταφοράς, αλλά μπορούν να χρησιμοποιηθούν και άλλα πρωτόκολλα, όπως το SMTP.

Επίσης, δεδομένου ότι το SOAP ταιριάζει πολύ καλά με το μοντέλο μηνύματος/απάντησης του HTTP, μπορεί εύκολα να χρησιμοποιηθεί με τα υπάρχοντα τείχη προστασίας (firewalls) και τους διακομιστές μεσολάβησης (proxy servers).

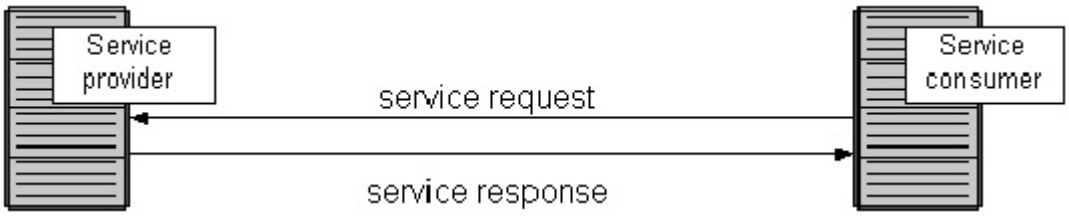
Το πρωτόκολλο SOAP ουσιαστικά βασίζεται στην υπηρεσιοστρεφή αρχιτεκτονική (Service Oriented Architecture - SOA), που περιγράφεται στη συνέχεια.

3.1.1 Η έννοια υπηρεσιοστρεφούς αρχιτεκτονικής

Η υπηρεσιοστρεφής αρχιτεκτονική (Service-oriented architecture - SOA) είναι ένα σχεδιαστικό πρότυπο σχεδίασης και αρχιτεκτονικής λογισμικού, που βασίζεται σε διακριτά τμήματα του λογισμικού που παρέχουν κάποιες λειτουργίες ως υπηρεσίες [16]. Οι υπηρεσίες δεν είναι συσχετιζόμενες, είναι χαλαρά συνδεδεμένες μονάδες, που λειτουργούν αυτόνομα και υλοποιούν μια ενέργεια. Έτσι, μια υπηρεσία μπορεί να εκληφθεί ως μία αυτοτελή λογική αναπαράσταση μιας επαναλαμβανόμενης λειτουργίας ή δραστηριότητας. Οι υπηρεσίες μπορούν να συνδυαστούν με άλλες εφαρμογές και μαζί να παρέχουν την πλήρη λειτουργικότητα μίας μεγάλης εφαρμογής λογισμικού. Ο σκοπός της SOA είναι να επιτρέπει την εύκολη συνεργασία ενός μεγάλου αριθμού υπολογιστών που είναι συνδεδεμένοι μέσω ενός δικτύου. Κάθε υπολογιστής μπορεί να τρέξει έναν αυθαίρετο αριθμό υπηρεσιών, καθεμιά από τις οποίες είναι φτιαγμένη με τέτοιο τρόπο, ώστε να εξασφαλίζεται ότι μπορεί να ανταλλάσσει πληροφορίες με οποιαδήποτε άλλη υπηρεσία εντός του δικτύου χωρίς ανθρώπινη παρέμβαση και χωρίς να γίνουν σε αλλαγές στο ίδιο το πρόγραμμα.

Πρακτικά, η υπηρεσιοστρεφής αρχιτεκτονική αποτελεί έναν τρόπο σχεδίασης ενός συστήματος λογισμικού, έτσι ώστε αυτό να παρέχει υπηρεσίες είτε σε εφαρμογές (οι οποίες θεωρούμε ότι αποτελούν τον τελικό χρήστη) ή σε άλλες υπηρεσίες μέσω δημοσιευμένων και γνωστών διεπαφών. Δίνεται έτσι ένας καλύτερος τρόπος να εκτεθούν κάποιες λειτουργίες μιας επιχείρησης και να αναπτυχθούν εφαρμογές οι οποίες βασίζονται σε αυτές ακριβώς τις λειτουργίες. Τελικά, η SOA δεν είναι απλά μια αυθαίρετη έννοια, αλλά μια σημαντική αρχιτεκτονική, λόγω της ολοένα αναδυόμενης χρήστης των υπηρεσιών ιστού.

Η κεντρική ιδέα της SOA φαίνεται παρακάτω:



Σχήμα 3.1: Υπηρεσία SOA και συνδέσεις

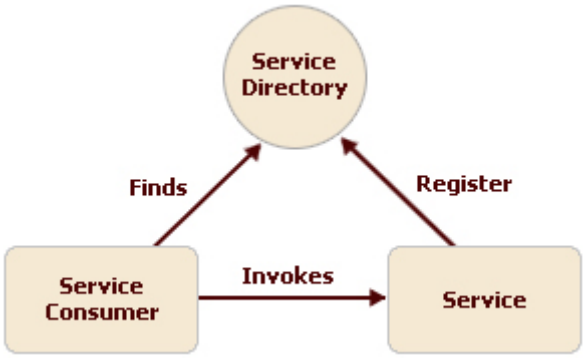
- Υπηρεσία: Μια λογική οντότητα, ο ρόλος της οποίας προσδιορίζεται από μία ή περισσότερες δημοσιευμένες διεπαφές.
- Πάροχος υπηρεσίας: Η οντότητα λογισμικού που υλοποιεί μια προδιαγραφήτης υπηρεσίας.
- Καταναλωτής υπηρεσίας: Η οντότητα λογισμικού, η οποία καλεί έναν πάροχο υπηρεσίας.

Πολλές υπηρεσίες ιστού υλοποιούν την υπηρεσιοστρεφή αρχιτεκτονική. Οι υπηρεσίες ιστού καθιστούν τα λειτουργικά δομικά στοιχεία προσβάσιμα μέσω τυποποιημένων πρωτοκόλλων του διαδικτύου, ανεξαρτήτου πλατφόρμας ή γλωσσας προγραμματισμού. Οι υπηρεσίες αυτές μπορεί να αντιπροσωπεύουν είτε νέες εφαρμογές ή απλά τροποποιημένα υπάρχοντα συστήματα ώστε να γίνουν διαθέσιμα στο διαδίκτυο.

Κάθε δομικό στοιχείο SOA μπορεί να παίξει τους παρακάτω ρόλους:

- Πάροχος υπηρεσιών: Ο πάροχος υπηρεσιών δημιουργεί μια διαδικτυακή υπηρεσία κι ενδεχομένως δημοσιεύει τη διεπαφή του και εισάγει πληροφορίες στο μητρώο υπηρεσιών (service registry). Ο πάροχος πρέπει να αποφασίσει ποιες υπηρεσίες θα δημοσιεύσει, να κάνει συμβιβασμούς μεταξύ της ασφάλειας και της προσβασιμότητας και να τιμολογήσει τις υπηρεσίες ή (αν δεν υπάρχουν χρεώσεις) να βρει πως να τις εκμεταλλευτεί. Ο πάροχος πρέπει επίσης να αποφασίσει σε ποια κατηγορία θα πρέπει να καταγραφεί η υπηρεσία και τι είδους συμφωνίες με τους εμπορικούς εταίρους απαιτούνται για τη χρήση της υπηρεσίας. Καταγράφει ποιες υπηρεσίες είναι διαθέσιμες στο πλαίσιο αυτό και παραθέτει όλους τους πιθανούς αποδέκτες των υπηρεσιών.

- Χρήστης υπηρεσίας: Οι χρήστες υπηρεσιών ιστού, εντοπίζουν τις εγγραφές στο μητρώο υπηρεσιών και στη συνέχεια συνδέονται με τον αντίστοιχο πάροχο, προκειμένου να καλέσουν μία από τις υπηρεσίες. Μπορούν να έχουν πρόσβαση σε πολλαπλές υπηρεσίες, εάν ο πάροχος παρέχει πολλαπλές υπηρεσίες.



Σχήμα 3.2: Υπηρεσιοστρεφής αρχιτεκτονική

Οι προγραμματιστές κατασκευάζουν SOA υπηρεσίες με τη χρήση διαδικτυακών πρότυπων υπηρεσιών (όπως είναι για παράδειγμα το SOAP) που έχουν αποκτήσει ευρεία αποδοχή από τις επιχειρήσεις. Τα πρότυπα αυτά (που αναφέρονται επίσης ως προδιαγραφές της υπηρεσίας) παρέχουν επίσης μεγάλη διαλειτουργικότητα και προστασία από κλείδωμα σε ιδιόκτητο λογισμικό πωλητή.

3.1.2 Βασικές αρχές υπηρεσιοστρεφούς αρχιτεκτονικής

Η SOA βασίζεται στην έννοια της υπηρεσίας. Κάθε υπηρεσία από την οποία αποτελείται μια SOA εφαρμογή, έχει σχεδιαστεί για να εκτελεί μια δραστηριότητα. Ως αποτέλεσμα, κάθε υπηρεσία είναι χτισμένη ως ένα διακριτό κομμάτι κώδικα. Αυτό καθιστά δυνατή την επαναχρησιμοποίηση του κώδικα με διαφορετικούς τρόπους στην εφαρμογή, αλλάζοντας μόνο τη διαλειτουργικότητα μιας υπηρεσίας με τις υπόλοιπες που συνθέτουν την εφαρμογή, αντί να γίνονται αλλαγές στον κώδικα της ίδιας της υπηρεσίας. Οι SOA αρχές σχεδιασμού χρησιμοποιούνται κατά τη διάρκεια τόσο της ανάπτυξης, όσο και της ενσωμάτωσης του λογισμικού.

Η SOA γενικά παρέχει έναν τρόπο για τους χρήστες των υπηρεσιών, (όπως web-based εφαρμογές), να έχουν επίγνωση των διαθέσιμων SOA-based υπηρεσίες. Για παράδειγμα, διαφορετικά τμήματα της ίδιας επιχείρησης μπορεί να αναπτύξουν SOA υπηρεσίες σε διαφορετικές γλώσσες. Οι αντίστοιχοι πελάτες θα επωφεληθούν από μια καλά καθορισμένη διεπαφή για να έχουν πρόσβαση σε αυτές τις υπηρεσίες. Η SOA ορίζει τη διεπαφή από την άποψη των πρωτοκόλλων και της λειτουργικότητας.

Η ιδιότητα της υπηρεσιοστρέφειας απαιτεί χαλαρή σύνδεση των υπηρεσιών με τα λειτουργικά συστήματα και άλλες τεχνολογίες που κρύβονται κάτω από τις εφαρμογές. Η SOA διαχωρίζει τις λειτουργίες σε διακριτές μονάδες ή υπηρεσίες, τις οποίες οι προγραμματιστές κάνουν προσβάσιμες μέσω ενός δικτύου, ώστε να επιτρέπουν στους χρήστες να τις συνδυάζουν και να τις επαναχρησιμοποιούν για τις εφαρμογές τους.

Για την ανάπτυξη, τη συντήρηση και τη χρήση SOA λογισμικού, τηρούνται οι ακόλουθες βασικές αρχές [16]:

- Τυποποιημένη σύμβαση παροχής υπηρεσιών (Standardized service contract): Οι υπηρεσίες εμπίπτουν σε μια συμφωνία επικοινωνιών, όπως ορίζεται από κοινού από ένα ή περισσότερα έγγραφα περιγραφής των υπηρεσιών.

- Χαλαρή σύζευξη υπηρεσιών (Service loose coupling): Οι υπηρεσίες διατηρούν μια σχέση που ελαχιστοποιεί τις εξαρτήσεις και απλά διατηρεί τη γνώση της ύπαρξης τους. Η χαλαρή σύζευξη εξασφαλίζει ότι η σύμβαση παροχής υπηρεσιών δεν είναι στενά συνδεδεμένη με τους καταναλωτές των υπηρεσιών και με την υποκείμενη λογική των υπηρεσιών και της εφαρμογής. Η έννοια αυτή έχει άμεση επιρροή από το αντικειμενοστρεφές μοντέλο, σύμφωνα με το οποίο ο στόχος είναι να μειωθεί η σύζευξη μεταξύ των κλάσεων, ώστε κάποιες αλλαγές σε αυτές να μη σπάει την υπάρχουσα σχέση μεταξύ τους.

- Αφαιρετικότητα υπηρεσιών (Service abstraction): Πέρα από τις περιγραφές της σύμβασης παροχής υπηρεσιών, υπηρεσίες κρύβουν τη λογική τους από τον έξω κόσμο. Οι πληροφορίες που δημοσιεύονται σε μια σύμβαση παροχής υπηρεσιών περιορίζεται μόνο σε ό,τι απαιτείται για να χρησιμοποιηθεί αποτελεσματικά η υπηρεσία και στην εξυπηρετούμενη σύμβαση (τεχνική σύμβαση και SLA).

- Επαναχρησιμοποιησιμότητα υπηρεσιών (Service reusability): Βασικό χαρακτηριστικό της SOA αρχιτεκτονικής είναι η δημιουργία υπηρεσιών που έχουν τη δυνατότητα να επαναχρησιμοποιηθούν. Οι επαναχρησιμοποιήσιμες υπηρεσίες έχουν σχεδιαστεί με τέτοιο τρόπο, έτσι ώστε η λογική τους να είναι ανεξάρτητη από οποιαδήποτε συγκεκριμένη τεχνολογία.

- Αυτονομία υπηρεσιών (Service autonomy): Οι υπηρεσίες έχουν τον έλεγχο της λογικής που ενσωματώνουν, ώστε να έχουν ενισχυμένη ανεξαρτησία από το περιβάλλον εκτέλεσής τους. Αυτό έχει ως αποτέλεσμα μεγαλύτερη αξιοπιστία, καθώς οι υπηρεσίες μπορούν να λειτουργούν με λιγότερη εξάρτηση από τους πόρους για τους οποίους υπάρχει μικρός ή καθόλου έλεγχος.

- Ανιθαγένεια υπηρεσιών (Service statelessness): Οι υπηρεσίες ελαχιστοποιούν την κατανάλωση πόρων, καθυστερώντας τη διαχείρισης των πληροφοριών κατάστασης, όποτε είναι δυνατόν. Αυτό βοηθάει στο σχεδιασμό κλιμακούμενων υπηρεσιών, διαχωρίζοντας τις από τα δεδομένα κατάστασής τους, εφόσον η διαχείριση των δεδομένων κατάστασής τους ανατίθεται σε εξωτερική μονάδα. Ως αποτέλεσμα υπάρχει μείωση των πόρων που καταναλώνουν.

- Γνωστοποίηση υπηρεσιών (Service discoverability): Οι υπηρεσίες συμπληρώνονται με επικοινωνιακά μετα-δεδομένα, με τη βοήθεια των οποίων μπορούν να ανακαλυφθούν και να ερμηνευθούν.

- Συνθεσιμότητα υπηρεσιών (Service composability): Τελευταία, κερδίζει ολοένα έδαφος η ανάπτυξη λογισμικού από ανεξάρτητα στοιχεία, που πιθανώς προϋπάρχουν. Αυτή είναι και η βασική έννοια στον αντικειμενοστρεφή προσανατολισμό, όπου το τελικό προϊόν αποτελείται από πολλά αλληλένδετα αντικείμενα. Οι υπηρεσίες είναι αποτελεσματικοί συμμετέχοντες μιας σύνθεσης-εφαρμογής, ανεξάρτητα από το μέγεθος και την πολυπλοκότητα της σύνθεσης.

3.1.3 Υπηρεσίες SOAP

Μια SOAP υπηρεσία πρέπει να ικανοποιεί τις ακόλουθες απαιτήσεις:

- Όλα τα μυνήματα μεταφέρονται μέσω του SOAP πρωτοκόλλου, εκτός από τα δυαδικά επισυναπτόμενα δεδομένα. Αξίζει να αποσαφηνιστεί ότι τα SOAP μυνήματα μεταφέρονται μέσω διαδικτυακών πρωτοκόλλων, όπως απαιτεί ο ορισμός των υπηρεσιών ιστού.

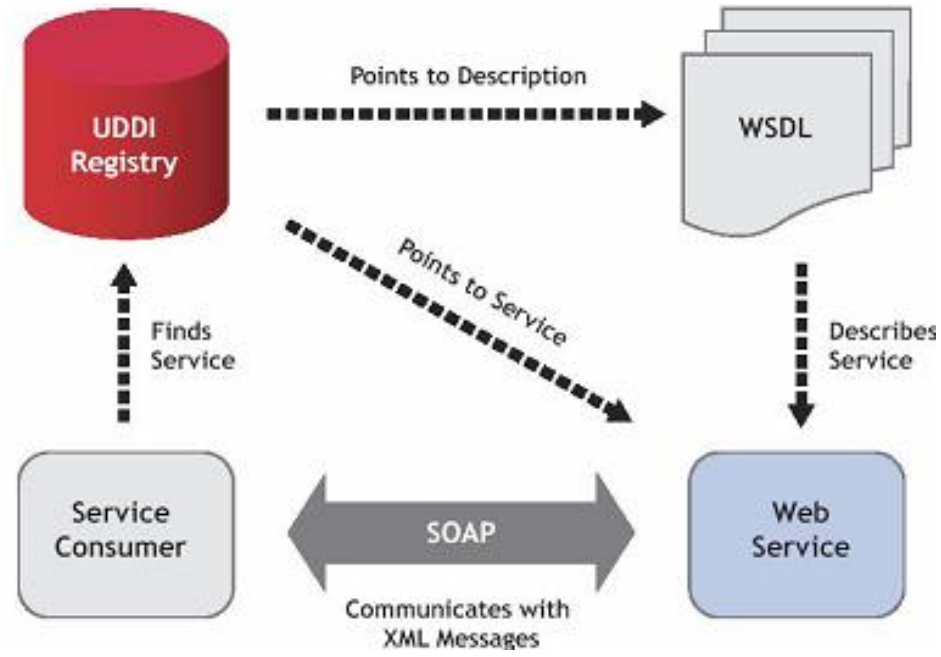
- Η περιγραφή της υπηρεσίας γίνεται με τη γλώσσα περιγραφής WSDL.

Οι SOAP υπηρεσίες είναι οι πιο διαδεδομένες στη βιομηχανία λογισμικού. Το κύριο πλεονέκτημά τους είναι η εξασφάλιση της μεταφοράς του μυνήματος μέσω ετερογενών δικτύων, χωρίς να ενδιαφέρει ποιο πρωτόκολλο κάνει τη μεταφορά. Το SOAP επίσης μπορεί να μεταφέρει μυνήματα με διάφορους τρόπους, από την απλή περίπτωση ερώτησης-απάντησης μέχρι περιπτώσεις εκπομπής (broadcast) αλλά και πιο σύνθετες.

Η γλώσσα περιγραφής υπηρεσιών ιστού (Web Services Description Language - WSDL) είναι μία γλώσσα περιγραφής διεπαφών, βασισμένη σε XML, που χρησιμοποιείται για την περιγραφή της λειτουργικότητας που προσφέρεται από μια υπηρεσία ιστού. Μια περιγραφή WSDL μιας υπηρεσίας ιστού (που αναφέρεται επίσης ως ένα αρχείο WSDL) παρέχει μια περιγραφή αναγνώσιμη από μηχανή, για το πως μπορεί να ονομάζεται η υπηρεσία, ποιες παραμέτρους αναμένει και τι δομές δεδομένων επιστρέφει. Έτσι, συνήθως παρέχεται ένα αρχείο WSDL από τον πάροχο μιας υπηρεσίας, το οποίο παίρνει ο χρήστης και με τη βοήθεια κάποιου εργαλείου μετατρέπει την περιγραφή αυτή σε κώδικα για τη γλώσσα προγραμματισμού που θέλει.

Το Universal Description, Discovery and Integration (UDDI) είναι μια υπηρεσία καταλόγου, όπου οι επιχειρήσεις μπορούν να εγγράψουν και να αναζητήσουν για υπηρεσίες ιστού. Είναι σχεδιασμένο ώστε να ερωτάται από τα μηνύματα SOAP και να παρέχει πρόσβαση στα αρχεία WSDL. Πριν το UDDI, δεν υπήρχε κάποια τυποποίηση στο διαδίκτυο ώστε οι επιχειρήσεις να προσεγγίζουν τους πελάτες και τους συνεργάτες τους και να δίνουν πληροφορίες σχετικά με τα προϊόντα και τις υπηρεσίες τους. Επίσης, δεν υπήρχε μέθοδος ένταξης σε συστήματα και διαδικασίες άλλων.

Η προδιαγραφή UDDI μπορεί να βοηθήσει στην επίλυση διαφόρων προβλημάτων. Αρχικά, δίνει τη δυνατότητα ανακάλυψης της κατάλληλης υπηρεσίας, από εκατομμύρια που είναι διαθέσιμες. Φτάνει σε νέους πελάτες και αυξάνει την πρόσβαση στους υπάρχοντες, ενώ επεκτείνει την αγορά με νέες προσφορές. Επίσης, περιγράφει τις υπηρεσίες και τις επιχειρηματικές διαδικασίες σε ένα ενιαίο, ανοικτό και ασφαλές περιβάλλον.



Σχήμα 3.3: Λειτουργία SOAP

Παρακάτω φαίνεται ένα παράδειγμα SOAP, στο οποίο αποστέλλεται σε ένα διακομιστή μία αίτηση GetStockPrice. Η αίτηση έχει μία παράμετρο StockName, και μια παράμετρο Price που θα επιστραφεί στην απάντηση. Ο χώρος ονομάτων για τη συνάρτηση ορίζεται στο "http://www.example.org/stock".

Η SOAP αίτηση είναι η εξής:

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

Η αντίστοιχη SOAP απόκριση είναι:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

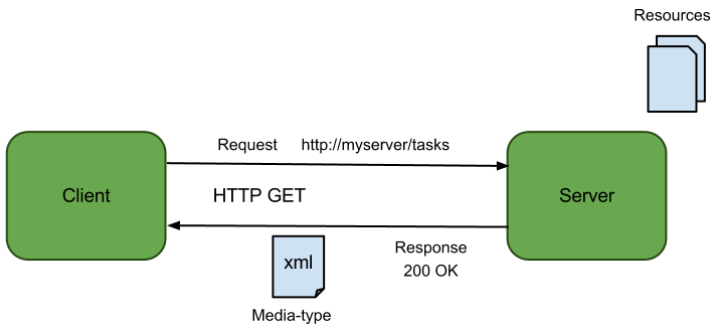
3.2 Representational State Transfer (REST)

Το REST είναι ένα αρχιτεκτονικό μοντέλο που αγνοεί τις λεπτομέρειες της υλοποίησης των επιμέρους συνιστωσών και τη σύνταξη του πρωτοκόλλου, προκειμένου να επικεντρωθεί στους ρόλους των στοιχείων και τους περιορισμούς κατά την αλληλεπίδρασή τους με άλλες συνιστώσες. Το REST έχει αναδειχθεί ως ένα κυρίαρχο μοντέλο σχεδιασμού διεπαφής προγραμματισμού εφαρμογών ιστού (web API).

Ένα σύστημα το οποίο εφαρμόζει τους κανόνες και περιορισμούς της αρχιτεκτονικής REST, λέμε ότι είναι ''RESTful''. Ένα τέτοιο σύστημα αποτελείται από πελάτες (clients) κι εξυπηρετητές (servers). Η διαδικτυακή υπηρεσία αποτελείται από πόρους (Resource Oriented Service). Ο πελάτης αποστέλλει στον εξυπηρετητή ένα αίτημα (request) και ο εξυπηρετητής το επεξεργάζεται κατάλληλα, επιστρέφοντας μία απάντηση (response). Οι αιτήσεις και οι απαντήσεις είναι σχεδιασμένες γύρω από τη μεταφορά αναπαράστασης πόρων. Ένας πόρος μπορεί να είναι οποιαδήποτε έννοια (κάποιος χρήστης, ένα προϊόν κ.α) στον οποίο δίνουμε μια μοναδική διεύθυνση. Για το σκοπό αυτό χρησιμοποιούμε καθολικούς ταυτοποιητές (URIs). Η αναπαράσταση (representation) ενός πόρου (resource) είναι τυπικά ένα έγγραφο το οποίο απεικονίζει τη τρέχουσα κατάσταση (state) ενός πόρου.

Ο όρος ''REST'' προέρχεται από τη έννοια της μεταφοράς κατάστασης στον πελάτη. Με κάθε νέο αίτημα του πελάτη, επιστρέφεται από τον εξυπηρετητή μια απάντηση, με αποτέλεσμα να μεταβαίνει σε μια νέα κατάσταση.

Το μεγαλύτερο αλλά και πιο γνωστό σύστημα, το οποίο εφαρμόζει το αρχιτεκτονικό μοντέλο REST, είναι ο παγκόσμιος ιστός. Όπως ο παγκόσμιος ιστός, έτσι και κάθε καλο-σχεδιασμένη RESTful διαδικτυακή υπηρεσία, θα πρέπει να συμπεριφέρεται ως ένα δίκτυο από ιστοσελίδες (virtual state machine), όπου ο κάθε χρήστης περιηγείται μέσα στη εφαρμογή επιλέγοντας συνδέσμους (μεταφορείς κατάστασης), με αποτέλεσμα μια νέα ιστοσελίδα (η αναπαράσταση της νέας κατάστασης) η οποία μεταφέρθηκε ως απάντηση από τον εξυπηρετητή.



Σχήμα 3.4: Βασική ιδέα του REST

Η βασική ιδέα του REST είναι η ύπαρξη πόρων (πηγές συγκεκριμένων πληροφοριών), καθένας από τους οποίους αναφέρεται με ένα παγκόσμιο αναγνωριστικό (π.χ. ένα URI στο HTTP). Για να χειριστούν τους πόρους αυτούς οι συνιστώσες του δικτύου

(user agents και origin servers) επικοινωνούν μέσω τυποποιημένης διασύνδεσης (π.χ. HTTP) και ανταλλάσσουν αναπαραστάσεις αυτών των πόρων.

Μια εφαρμογή μπορεί να αλληλεπιδράσει με έναν πόρο, γνωρίζοντας δύο πράγματα: το αναγνωριστικό του πόρου και τις απαιτούμενες ενέργειες, χωρίς να χρειάζεται να γνωρίζει εάν μεσολαβούν ενδιάμεσοι (διακομιστές, πύλες, τείχη προστασίας, ή οτιδήποτε άλλο). Η εφαρμογή, ωστόσο, πρέπει να μπορεί να κατανοεί τη μορφή των επιστρεφόμενων πληροφοριών (αναπαράσταση), οι οποίες είναι συνήθως ένα HTML, XML ή JSON έγγραφο, αλλά και οποιοδήποτε άλλο περιεχόμενο.

3.2.1 Βασικές αρχές του REST

Η αρχιτεκτονική REST περιγράφει τους ακόλουθους έξι περιορισμούς που εφαρμόζονται, αφήνοντας την υλοποίηση των επιμέρους τμημάτων ελεύθερη για σχεδιασμό από τον κάθε προγραμματιστή:

- **Μοντέλο πελάτη-εξυπηρετητή (Client-server)**
  Οι πελάτες χωρίζονται από τους εξυπηρετητές με μια ενιαία διεπαφή. Αυτός ο διαχωρισμός των ρόλων που σημαίνει ότι οι χρήστες ενδιαφέρονται για διαφορετικά πράγματα από τους εξυπηρετητες. Για παράδειγμα οι χρήστες δεν ασχολούνται με την αποθήκευση των δεδομένων, η οποία παραμένει εσωτερικό θέμα κάθε εξυπηρετητή, ενώ οι εξυπηρετητές δεν ασχολούνται με το περιβάλλον εργασίας ή την κατάσταση του χρήστη. Έτσι, ο κώδικας από την πλευρά του χρήστη έχει μεγαλύτερη φορητότητα, ενώ οι εξυπηρετητές είναι απλούστεροι κι επεκτάσιμοι. Οι εφαρμογές πελάτη κι εξυπηρετητή μπορούν επίσης να αναπτυχθούν ανεξάρτητα, εφ΄ όσον η διασύνδεση μεταξύ τους δεν μεταβάλλεται.

- **Stateless σχεδίαση**
  Η επικοινωνία πελάτη-εξυπηρετητή δεν περιορίζεται περαιτέρω από δεδομένα του πελάτη που τυχόν αποθηκεύονται στον εξυπηρετητή μεταξύ των αιτημάτων. Κάθε αίτηση από οποιονδήποτε πελάτη περιέχει όλες τις αναγκαίες πληροφορίες για την εξυπηρέτηση του αιτήματος και η κατάσταση της συνεδρίας κρατείται στην πλευρά του πελάτη.

- **Επαναχρησιμοποίηση αποκρίσεων (Cacheable responses)**
  Όπως και στο διαδίκτυο, οι πελάτες μπορούν να αποθηκεύουν τις αποκρίσεις. Οι αποκρίσεις πρέπει επομένως να αυτοπροσδιορίζονται, σιωπηρά ή ρητά, ως επαναχρησιμοποιήσιμες ή όχι, έτσι ώστε να αποτρέπουν τους πελάτες να επαναχρησιμοποιήσουν ακατάλληλα δεδομένα ως απόκριση σε περαιτέρω αιτήματα.

- **Πολυεπίπεδο σύστημα (layered system)**
  Ο πελάτης δεν ξέρει αν είναι συνδεδεμένος απευθείας με τον τελικό εξυπηρετητή (end server) ή σε κάποιον ενδιάμεσο. Οι ενδιάμεσοι εξυπηρετητές βοηθούν στη βελτίωση της κλιμάκωσης εφαρμόζοντας ισολογισμό του φόρτου εργασίας και παρέχοντας μοιραζόμενες μνήμες. Παράλληλα, εφαρμόζουν πολιτικές και ελέγχους ασφαλείας.

- **Ενιαία διεπαφή (uniform interface)** Η ενιαία διεπαφή μεταξύ των πελατών και των εξυπηρετητών απλοποιεί την αρχιτεκτονική και αποσυνδέει τα διάφορα τμήματα, επιτρέποντάς τους να εξελιχθούν ανεξάρτητα.

- **Κώδικας on-demand** Αυτός είναι ουσιαστικά ένας προαιρετικός περιορισμός. Οι εξυπηρετητές μπορούν να επεκτείνουν τη λειτουργικότητα του πελάτη μεταφέροντας σε αυτόν εκτελέσιμο κώδικα. Τέτοια παραδείγματα μπορούν να μετταγλωτισμένα τμήματα κώδικα, όπως Java Applets ή JavaScript scripts.

Η ενιαία διεπαφή που πρέπει να παρέχεται από κάθε REST υπηρεσία, πρέπει να τηρεί τις εξής θεμελιώδεις αρχές:

- **Προσδιορισμός των πόρων**
  Οι πόροι προσδιορίζονται στο αίτημα, χρησιμοποιώντας για παράδειγμα URIs σε δικτυακά συστήματα REST. Οι ίδιοι οι πόροι είναι εννοιολογικά διαφορετικοί από τις αναπαραστάσεις που επιστρέφονται στον πελάτη. Για παράδειγμα, ένας διακομιστής δεν αποστέλλει τη βάση δεδομένων του, αλλά ίσως κάποιο αρχείο HTML, XML ή JSON που αντιπροσωπεύει κάποιες εγγραφές της βάσης δεδομένων.

- **Χειρισμός των πόρων μέσω αναπαραστάσεων**
  Όταν ένας πελάτης έχει μια αναπαράσταση ενός πόρου, συμπεριλαμβανομένων οποιωνδήποτε μεταδεδομένων, έχει αρκετές πληροφορίες ώστε να τροποποιήσει ή να διαγράψει τον πόρο στον διακομιστή, με την προϋπόθεση ότι έχει την άδεια να το πράξει.

- **Αυτο-περιγραφικά μηνύματα (Self-descriptive messages)**
  Κάθε μήνυμα περιλαμβάνει αρκετές πληροφορίες για να περιγράψει πως μπορεί να υποστεί επεξεργασία. Οι αποκρίσεις δηλώνουν επίσης ρητά τη δυνατότητα επαναχρησιμοποίησής τους.

- **Τα υπερμέσα ως κινητήρια δύναμη της κατάστασης των εφαρμογών (HATEOAS)**
  Οι πελάτες κάνουν μεταβάσεις μόνο μέσω ενεργειών που προσδιορίζονται δυναμικά μέσα από υπερμέσα του διακομιστή (π.χ. μέσω hyperlinks).

3.2.2 Υπηρεσίες REST

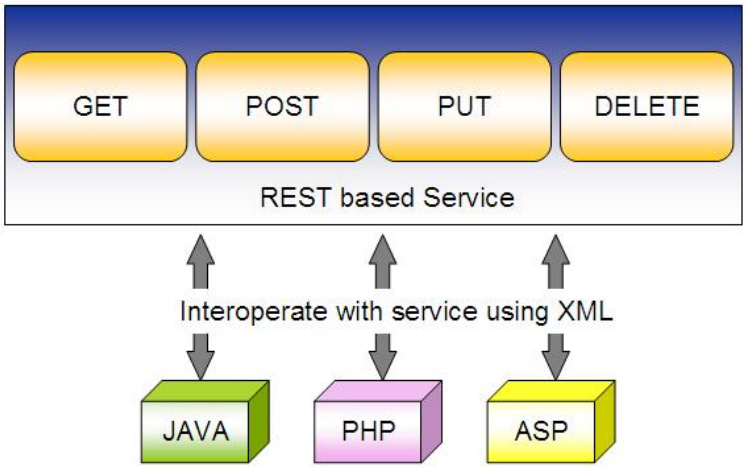
Το REST έχει ως στόχο τη συμπεριφορά μίας καλά σχεδιασμένης εφαρμογής ιστού: σε ένα δίκτυο ιστοσελίδων, ο χρήστης προχωρά μέσα σε μία εφαρμογή επιλέγοντας συνδέσμους (μεταβάση κατάστασης), με αποτέλεσμα τη μεταφορά της επόμενης σελίδας στο χρήστη (που αντιπροσωπεύει την επόμενη κατάσταση του αιτήματος).

Το REST περιγράφηκε αρχικά στα πλαίσια του πρωτοκόλλου HTTP, αλλά δεν περιορίζεται σε αυτό. Μπορεί να βασίζεται επίσης σε άλλα πρωτόκολλα επιπέδου εφαρμογών, εφόσον παρέχουν ένα πλούσιο και ομοιόμορφο λεξιλόγιο για εφαρμογές που βασίζονται στην αναπαράσταση και τη μετάβαση καταστάσεων.

Σήμερα, το REST κάνει ευρεία χρήση των μεθόδων του HTTP πρωτοκόλλου. Με τον τρόπο αυτό πετυχαίνεται μία-προς-μία αντιστοίχιση ανάμεσα στις CRUD λειτουργίες, δηλαδή τη δημιουργία(Create), την ανάγνωση(Read), την ενημέρωση(Update) και τη διαγραφή(Delete) και των HTTP μεθόδων.

Σύμφωνα με τη αντιστοίχιση αυτή έχουμε:

- Για τη δημιουργία ενός πόρου στο εξυπηρετητή, χρησιμοποιούμε τη μέθοδο POST ή PUT.
- Για τη ανάκτηση πόρου από τον εξυπηρετητή, χρησιμοποιούμε τη μέθοδο GET.
- Για τη αλλαγή κατάστασης πόρου ή για τη ενημέρωση του, χρησιμοποιούμε τη μέθοδο POST ή PUT.
- Για τη διαγραφή πόρου από τον εξυπηρετητή, χρησιμοποιούμε τη DELETE.

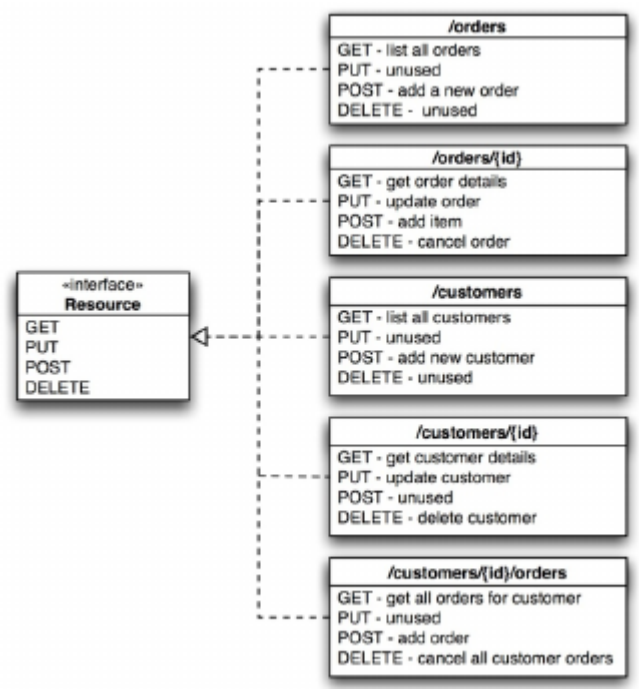


Σχήμα 3.5: Λειτουργία του REST

Μια RESTful υπηρεσία ιστού, είναι μία προγραμματιστική διεπαφή ιστού (web API) που έχει υλοποιηθεί με τη χρήση HTTP πρωτοκόλλου και τις αρχές της αρχιτεκτονικής REST. Πρόκειται για μια συλλογή των πόρων, με τέσσερα βασικά χαρακτηριστικά:

- το URI βάσης για το web API, όπως για παράδειγμα http://example.com/resources/.
- το διαδικτυακό τύπο δεδομένων που υποστηρίζεται από το web API. Αυτός είναι συχνά JSON, αλλά μπορεί να είναι κι οποιοδήποτε άλλος τύπος.
- το σύνολο των ενεργειών που υποστηρίζονται από το web API μέσω των HTTP μεθόδων (GET, PUT, POST, DELETE).
- Το API θα πρέπει να χειρίζεται μέσω υπερκειμένου.

Παρακάτω φαίνεται ένα παράδειγμα δικτυακής υπηρεσίας, που έχει αναπτυχθεί με αρχιτεκτονική REST. Όπως φαίνεται, χρησιμοποιούμε ένα σύνολο από HTTP μεθόδους και τις εφαρμόζουμε σε ένα σύνολο διεθύνσεων URI.



Σχήμα 3.6: Παράδειγμα RESTful διαδικτυακής υπηρεσίας

Παράδειγμα αίτησης λήψης λίστας πελατών με HTTP GET:

```
GET /customers HTTP/1.1
Host: mydomain.com
Accept: application/xml
```

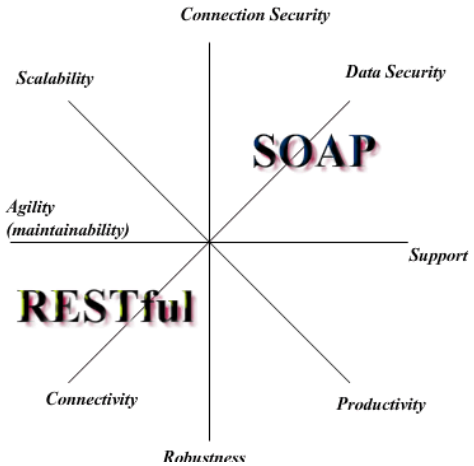
Παράδειγμα απάντησης λήψης λίστας πελατών σε XML με HTTP GET:

```
<customers>
  <customer>
    <id>1233</id>
    <name>Pavlos Leggos</name>
    <address>25, Othonos, Athens, Greece</address>
  </customer>
  <customer>
    <id>1234</id>
    <name>Giorgos Aggelou</name>
    <address>12, Akropoleos, Athens, Greece</address>
  </customer>
</customers>
```

3.3 REST vs SOAP

Αν και πολλοί θεωρούν πως το μοντέλο REST δεν διαφέρει πολύ από το SOAP, στην πραγματικότητα αυτές τις διαφορές είναι αρκετά σημαντικές:

- Καταρχήν, το REST δεν προσανατολίζεται καθόλου στις υπηρεσίες, παρόλο που πολλοί θεωρούν ότι βασίζεται στην υπηρεσιοστρεφή αρχιτεκτονική. Αντίθετα, βασίζεται στην έννοια των πόρων. Άρα, το ζήτημα REST vs SOAP ανάγεται σε ζήτημα resource-oriented vs service-oriented αρχιτεκτονικής. Σε αντίθεση με μια υπηρεσία, όπου οι μέθοδοι είναι εντελώς ανεξάρτητες και μπορούν να υλοποιηθούν ξεχωριστά, οι μεθόδοι για έναν πόρο είναι συγκεκριμένες.
- Το SOAP είναι ένα πρωτόκολλο ανταλλαγής μηνυμάτων, ενώ το REST δεν είναι πρωτόκολλο, αλλά μία αρχιτεκτονική τεχνοτροπία.
- Το SOAP έχει τυπικές και καθορισμένες προδιαγραφές, αντίθετα από το REST.
- Το SOAP είναι πιο ασφαλές από το REST.
- Το REST δεν επιβάλλει κάποια μορφή μηνύματος, όπως XML, JSON ή κάποια άλλη. Το SOAP βασίζεται στη γλώσσα XML για την ανταλλαγή μυνημάτων.
- Το REST είναι ένα πλήρως stateless μοντέλο. Το SOAP έχει προδιαγραφές ακόμα και για stateful υλοποίηση.
- Το SOAP έχει αυστηρές προδιαγραφές για κάθε τμήμα της υλοποίησης. Αντίθετα, το REST δίνει τη γενική έννοια και είναι λιγότερο περιοριστικό.
- Το SOAP είναι σχεδιασμένο για τη διαχείριση κατανεμημένων συστημάτων. Αντίθετα, το REST είναι πιο χρήσιμο για επικοινωνία από-σημείο-σε-σημείο.



Σχήμα 3.7: REST vs SOAP

Κεφάλαιο 4

NoSQL βάσεις δεδομένων

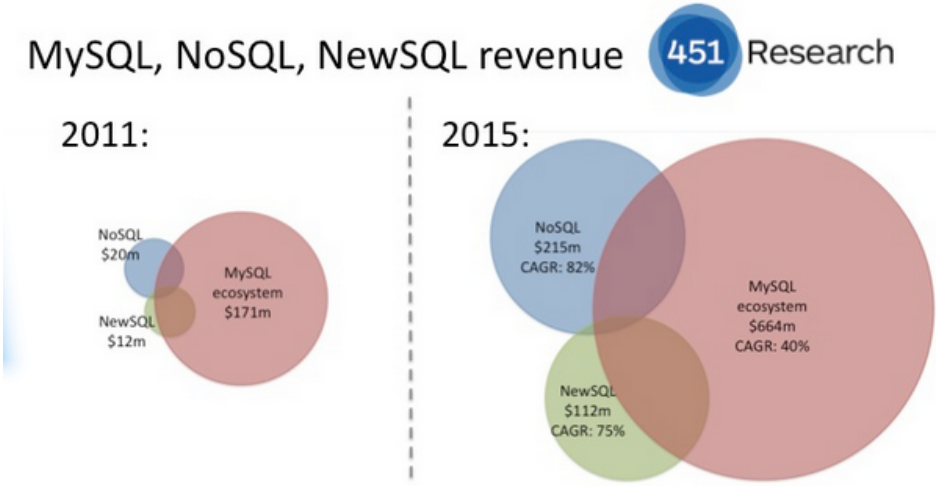
4.1 Ανάγκη ανάπτυξης NoSQL βάσεων δεδομένων

Μια βάση δεδομένων είναι μια συλλογή από δεδομένα που οργανώνονται σε τέτοιες δομές ώστε να τα αναπαριστούν, να τα ιεραρχούν και να τα συνδέουν με τον καλύτερο δυνατό τρόπο. Η βάση δεδομένων έχει κι ένα σύστημα διαχείρισης, το Σύστημα Διαχείρισης Βάσεων Δεδομένων (Database Management System - DBMS). Υπάρχουν διάφορα μοντέλα στα οποία μπορεί να βασίζεται ένα DBMS και η σχετική γλώσσα προγραμματισμού για το χειρισμό του. Οι παραδοσιακές βάσεις δεδομένων χρησιμοποιούν το σχεσιακό μοντέλο (σχεσιακές βάσεις δεδομένων - relational DBMS) και τη γλώσσα προγραμματισμού SQL (Structured Query Language).

Ένα νέο μοντέλο δόμησης και διαχείρισης δεδομένων είναι οι NoSQL βάσεις δεδομένων. Υπάρχουν διάφορα είδη NoSQ βάσεων, ανάλογα με τον τρόπο που αποθηκεύονται τα δεδομένα και καθένα έχει τη δική του γλώσσα υποβολής ερωτημάτων. Μια NoSQL βάση δεδομένων παρέχει ένα μηχανισμό για την αποθήκευση και την ανάκτηση δεδομένων, ο οποίος χρησιμοποιεί λιγότερο περιοριστικά μοντέλα συνέπειας από τις παραδοσιακές σχεσιακές βάσεις δεδομένων. Το NoSQL ερμηνεύεται ως ‘‘όχι μόνο SQL’’, που στην πραγματικότητα σημαίνει ότι επιτρέπεται και η χρήση SQL ερωτημάτων.

Οι NoSQL βάσεις δεδομένων δημιουργήθηκαν ως ένα μέσο για την προσφορά υψηλής απόδοσης (όσον αφορά τόσο την ταχύτητα όσο και το μέγεθος) και υψηλής διαθεσιμότητας, θυσιάζοντας την ACID προσέγγιση (ατομικότητα, συνέπεια, απομόνωση, μονιμότητα) των παραδοσιακών βάσεων δεδομένων και χρησιμοποιώντας λιγότερο περιοριστικά BASE χαρακτηριστικά (βασική διαθεσιμότητα, χαλαρή κατάσταση, τελική συνέπεια).

Μερικοί λόγοι για αυτή την ολοένα και πιο διαδεδομένη προσέγγιση είναι η απλότητα του σχεδιασμού, η οριζόντια κλιμάκωση και ο καλύτερος έλεγχος διαθεσιμότητας. Ταυτόχρονα, παρατηρείται όλο και περισσότερο η ανάγκη για ταυτόχρονη αποθήκευση και διαχείριση μεγάλου όγκου κατανεμημένων δεδομένων, τα οποία εμφανίζουν



Σχήμα 4.1: Μερίδιο αγοράς βάσεων δεδομένων

πολυμορφία, όπως και η ανάγκη μεταφοράς τους πάνω από ένα δίκτυο ή μέσα από συστήματα P2P (peer-to-peer). Επίσης, παρατηρείται η χρήση υπηρεσιών πάνω από διαφορετικές πλατφόρμες, το λογισμικό και τα εργαλεία των οποίων μπορεί να διαφέρουν. Για την υλοποίηση τέτοιων σύνθετων, κατανεμημένων υπηρεσιών δεν αρκούν παλαιότερα μοντέλα ανάπτυξης. Οι σχεσιακές βάσεις δεδομένων δεν έχουν δημιουργηθεί για τέτοιες υπηρεσίες ή για ευέλικτες περιπτώσεις όπου χρειάζεται γρήγορα scale-up ή scale-down στους πόρους που χρησιμοποιούνται. Οι NoSQL βάσεις χρησιμοποιούνται ευρέως για εφαρμογές με μεγάλο όγκο δεδομένων και σε εφαρμογές ιστού.

Γι΄ αυτό το λόγο, αναπτύχθηκαν οι NoSQL βάσεις δεδομένων, όπου τα δεδομένα δεν είναι απαραίτητα δομημένα σε πίνακες, όπως στο σχεσιακό μοντέλο, αλλά με διάφορους τρόπους [32]. Αυτό δίνει μεγάλη ελευθερία στους χρήστες, οι οποίοι μπορούν να επιλέξουν το μοντέλο που ταιριάζει καλύτερα στις ανάγκες της εφαρμογής τους. Επίσης, κάθε τέτοια βάση δεδομένων παρέχει δικά της εργαλεία για τη διαχείριση των δεδομένων, καθώς και δικής της γλώσσα υποβολής ερωτημάτων (query language), εφόσον δεν υπάρχει συγκεκριμένη γλώσσα προγραμματισμού για το χειρισμό τους.

Οι NoSQL βάσεις δεδομένων παρέχουν επεκτασιμότητα, πολυμορφία κι αύξηση στις επιδόσεις των εφαρμογών. Εφαρμόζονται σε clusters, υποστηρίζουν διαχείριση κατανεμημένων δεδομένων και παρέχουν μηχανισμούς ανάκτησης δεδομένων μετά από σφάλματα. Οι πιο συνηθισμένες είναι βελτιστοποιημένες key–value stores, που έχουν ως στόχο απλές λειτουργίες ανάκτησης και προσθήκης δεδομένων, με αποτέλεσμα σημαντικά οφέλη στην απόδοση.

4.2 Ταξινόμηση NoSQL βάσεων δεδομένων

Ανάλογα με τον τρόπο που αναπαριστώνται κι αποθηκεύονται τα δεδομένα, οι NoSQL βάσεις χωρίζεται σε διάφορες κατηγορίες. Αν και δεν υπάρχει επίσημη ταξινόμηση, τα συστήματα NoSQL βάσεων μπορούν να διαχωριστούν ως εξής:

A) Core NoSQL Systems, τα περισσότερα από τα οποία δημιουργήθηκαν σαν συστατικά στοιχεία για υπηρεσίες Web 2.0, με την ακόλουθη κατηγοριοποίηση:

- Column Store (Hadoop/HBase, Cassandra, Hypertable, Cloudata, Amazon SimpleDB, SciDB)
- Document Store (CouchDB, MongoDB, Terrastore, ThruDB, OrientDB, RavenDB, Citrusleaf, SisoDB, CloudKit, Perservere, Jackrabbit)
- Key Value Store
  - KV Store - Eventually consistent (Amazon Dynamo, Voldemort, Dynomite, SubRecord, Mo8onDb, Dovetaildb)
  - KV Store - Hierarchical (GT.m, Cache)
  - KV Store - Ordered (TokyoTyrant, Lightcloud, NMDB, Luxio, MemcacheDB, Actord)
  - KV Cache (Memcached, Repcached, Coherence, Hazelcast, Infinispan, EXtremeScale, JBossCache, Velocity, Terracoqua)
  - Tuple Store (Gigaspaces, Coord, Apache River)
- Graph Databases (Neo4J, Infinite Graph, Sones, InfoGrid, HyperGraphDB, Trinity, AllegroGraph, Bigdata, DEX, OpenLink Virtuoso, VertexDB, FlockDB, Java Universal Network / Graph Framework, Sesame, Filament, OWLim, NetworkX, iGraph)

B) Soft NoSQL Systems, τα περισσότερα από τα οποία είναι συστήματα που δεν συνδέονται με κάποια υπηρεσία Web 2.0, αλλά μοιράζονται τα χαρακτηριστικά των NoSQL συστημάτων. Αυτά χωρίζοντα στα:

- Object Databases (db4o, Versant, Objectivity, Gemstone, Progress, Starcounter, Perst, ZODB, NEO, PicoLisp, Sterling, StupidDB, KiokuDB, Durus)
- Grid Cloud Database Solutions (GigaSpaces, Queplix, Hazelcast, Joafip, GridGain, Infinispan, Coherence, eXtremeScale)
- XML Databases (Mark Logic Server, EMC Documentum xDB, Tamino, eXist, Sedna, BaseX, Xindice, Qizx, Berkeley DB XML)
- Multivalue Databases (U2, OpenInsight, OpenQM, Globals)
- άλλες NoSQL-related databases (IBM Lotus/Domino, Intersystems Cache, eXtremeDB, ISIS Family, Prevayler, Yserial)

4.2.1 Document store

Στις document-oriented databases, τα δεδομένα αποθηκεύονται σε έγγραφα (documents) με κάποιο συγκεκριμένο μοτίβο κι όχι σε συσχετιζόμενους πίνακες, όπως στις σχεσιακές βάσεις δεδομένων. Τα έγγραφα μπορεί να είναι XML, JSON, BSON, YAML, PDF αρχεία και άλλα. Οι διάφορες υλοποιήσεις δίνουν διαφορετικούς τρόπους ανάγνωσης και διαχείρισης των δεδομένων. Τα δεδομένα μπορεί να είναι οργανωμένα σε συλλογές (collections), σε ιεραρχίες καταλόγου (directory-hierarchies), σε μεταδεδομένα που δε συμμετέχουν σε ερωτήματα στη βάση (non-visible metadata) ή ετικέτες (tags). Για παράδειγμα, σε σύγκριση με σχεσιακές βάσεις δεδομένων, οι συλλογές θα μπορούσαν να θεωρηθούν ως πίνακες.

Ανάλογα με τις εκάστοτε ανάγκες, έγγραφα της ίδιας συλλογής θα μπορούσαν να έχουν διαφορετικά πεδία. Η προσθήκη ενός νέου πεδίου σε ένα έγγραφο δεν επηρεάζει τα υπόλοιπα έγγραφα της βάσης.

Για παράδειγμα, ένα έγγραφο μπορεί να είναι το παρακάτω:

```
{
        FirstName: "Bob",
        Address: "5 Oak St.",
        Hobby: "sailing"
}
```

Ένα άλλο θα μπορούσε να ήταν το εξής:

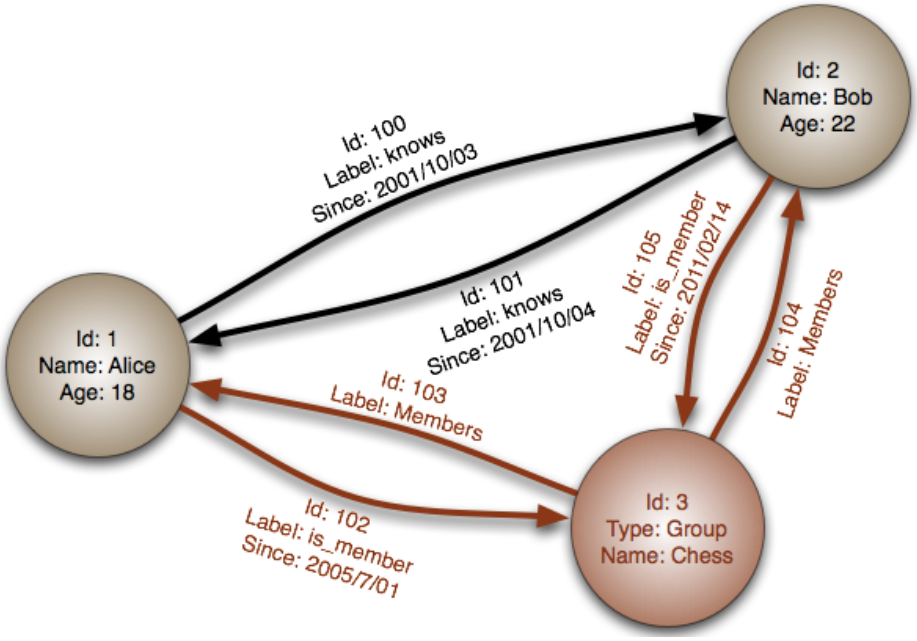
```
{
        FirstName: "Jonathan",
        Address: "15 Wanamassa Point Road",
        Children: [
                {Name: "Michael", Age: 10},
                {Name: "Jennifer", Age: 8},
                {Name: "Samantha", Age: 5},
                {Name: "Elena", Age: 2}
        ]
}
```

Τα έγγραφα αντιπροσωπεύονται μέσω ενός μοναδικού κλειδιού (Universally Unique Identifier - UUID), κάτι που καθιστά σχεδόν αδύνατο να έχουν δύο έγγραφα το ίδιο κλειδί. Αντίθετα από τις σχεσιακές βάσεις δεδομένων, για τον μονοσήμαντο προσδιορισμό της κάθε εγγραφής σε έναν πίνακα, χρησιμοποιούνται τα πρωτεύοντα κλειδιά μαζί με κάποιο μηχανισμό παραγωγής τους (auto-increment). Ωστόσο, σε περιπτώσεις κατανεμημένων σχεσιακών βάσεων, όπου τα δεδομένα δεν αποθηκεύονται σε ένα μοναδικό σημείο, η προσθήκη νέων εγγραφών σε δύο σημεία της βάσης μπορεί να οδηγήσει σε διαφορετικές εγγραφές με το ίδιο πρωτεύον κλειδί. Αυτό εξαλείφεται με τη χρήση του μοναδικού κλειδιού στις document store βάσεις.

4.2.2 Graph databases

Οι graph databases σχεδιάστηκαν για δεδομένα που αναπαρίστανται καλύτερα με τη μορφή γράφου, όπως αναπαράσταση κοινωνικών σχέσεων, μεταφορές, χάρτες, τοπολογίες δικτύου και άλλα. Οποιοδήποτε σύστημα αποθήκευσης που παρέχει διασύνδεση αντικειμένων μέσω δεικτών μπορεί να χρησιμοποιηθεί σαν graph database. Μία τέτοια βάση δεδομένων αποτελείται από κόμβους, ακμές και τις ιδιότητες των κόμβων. Οι κόμβοι αναπαριστούν οντότητες και οι ακμές τις σχέσεις που τις συνδέουν. Οι πληροφορίες αποθηκεύονται στις ακμές.

Παρακάτω φαίνεται ένα παράδειγμα μίας graph database:



Σχήμα 4.2: Graph database

Οι graph databases είναι ένα ισχυρό εργαλείο για graph-like ερωτήματα, όπως για παράδειγμα τον υπολογισμό της συντομότερης διαδρομής μεταξύ δύο κόμβων στο γράφημα. Άλλα graph-like ερωτήματα μπορούν να πραγματοποιηθούν σε μια graph database με φυσικό τρόπο (για παράδειγμα ο υπολογισμός της διαμέτρου τους γραφήματος ή η ανίχνευση ομάδων).

Συγκριτικά με τις σχεσιακές βάσεις δεδομένων, οι graph databases είναι γρηγορότερες για δεδομένα που συσχετίζονται μεταξύ τους και ταιριάζουν καλύτερα στις αντικειμενοστρεφείς φιλοσοφίες. Συνήθως δεν απαιτούν πολύπλοκες συνενώσεις, είναι καλύτερες στην κλιμάκωση και καθώς δεν απαιτούν κάποιο αυστηρό schema, είναι πιο αποδοτικές για δεδομένα που αλλάζει συνεχώς το schema τους.

4.2.3 Column-oriented databases

Στις column-oriented databases, τα δεδομένα στους πίνακες αποθηκεύονται κατά στήλες, σε αντίθεση με τις σχεσιακές βάσεις δεδομένων που αποθηκεύονται κατά γραμμές.

Έστω ο παρακάτω πίνακας ενός σχεσιακού συστήματος βάσης δεδομένων:

EmpId	Lastname	Firstname	Salary
10	Smith	Joe	40000
12	Jones	Mary	50000
11	Johnson	Cathy	44000
22	Jones	Bob	55000

Μία column-oriented database σειριοποιεί όλες τις τιμές μιας στήλης μαζί, μετά τις τιμές της επόμενης στήλης, και ούτω καθεξής. Οπότε, το παραπάνω παράδειγμα, σε μία column-oriented database θα ήταν:

```
10:001,12:002,11:003,22:004;Smith:001,Jones:002,Johnson:003,
Jones:004;Joe:001,Mary:002,Cathy:003,Bob:004;40000:001,50000:002,
44000:003,55000:004;
```

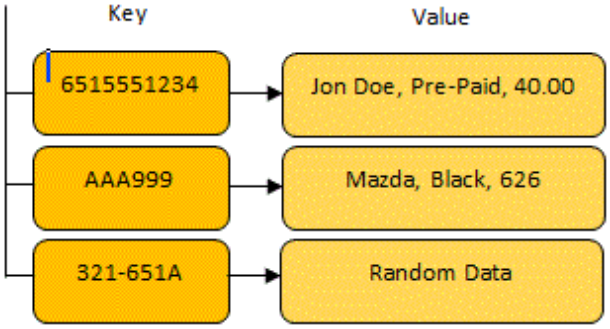
Οι column-oriented databases είναι πιο αποτελεσματικές όταν μία συνάθροιση πρέπει να υπολογίζεται για πολλές σειρές, αλλά μόνο για ένα αρκετά μικρό υποσύνολο των στηλών, επειδή η ανάγνωση ενός μικρού τέτοιου υποσυνόλου των δεδομένων μπορεί να είναι αρκετά ταχύτερη από την ανάγνωση όλων των δεδομένων. Επίσης, τέτοιες βάσεις δεδομένων είναι πιο αποτελεσματικές όταν οι νέες τιμές μιας στήλης παρέχονται για όλες τις γραμμές ταυτόχρονα, επειδή τα δεδομένα αυτής της στήλης μπορούν να γραφτούν αποδοτικά και να αντικαταστήσουν τα παλιά δεδομένα χωρίς να τροποποιήσουν τυχόν άλλες στήλες.

Αντίθετα, μία οργάνωση προσανατολισμένη στις σειρές (row-oriented organisation) είναι πιο αποτελεσματική όταν απαιτούνται την ίδια στιγμή πολλές στήλες μίας μόνο σειράς, καθώς ολόκληρη η σειρά μπορεί να ανακτηθεί με μία μόνο αναζήτηση στο δίσκο. Μία τέτοια οργάνωση είναι πιο αποτελεσματική όταν γράφεται μια νέα σειρά, αν όλα τα δεδομένα των στηλών αυτής της σειράς παρέχονται ταυτόχρονα, αφού ολόκληρη η σειρά μπορεί να γραφτεί με μία μόνο αναζηση στο δίσκο.

Πρακτικά, μία οργάνωση σε column-oriented database είναι πιο αποδοτική σε συστήματα OLAP (online analytical processing), όπου απαιτούνται συνενώσεις μεγάλου όγκου παρόμοιων δεδομένων, όπως σε αποθήκες δεδομένων (data warehouses), συστήματα διαχείρισης πελατειακών σχέσεων (customer relationship management - CRM) και παρόμοια συστήματα, όπου απαιτούνται λιγότερα πολύπλοκα ερωτήματα.

4.2.4 Key-value stores

Τα key-value stores είναι ένας τρόπος αποθήκευσης σε μορφή ζευγαριών κλειδιού-τιμής key-value. Στο value αποθηκεύονται τα δεδομένα, είτε σε μορφή αντικειμένων, είτε σε άλλο τύπο και το key είναι το κλειδί που ανατίθεται στο value. Η αναζήτηση των δεδομένων γίνεται σε πολλά συστήματα με αυτό το κλειδί. Οι key-value store βάσεις επιτρέπουν στις εφαρμογές να αποθηκεύουν τα δεδομένα με schema-less τρόπο.



Σχήμα 4.3: Παράδειγμα key-value store

4.3 MongoDB

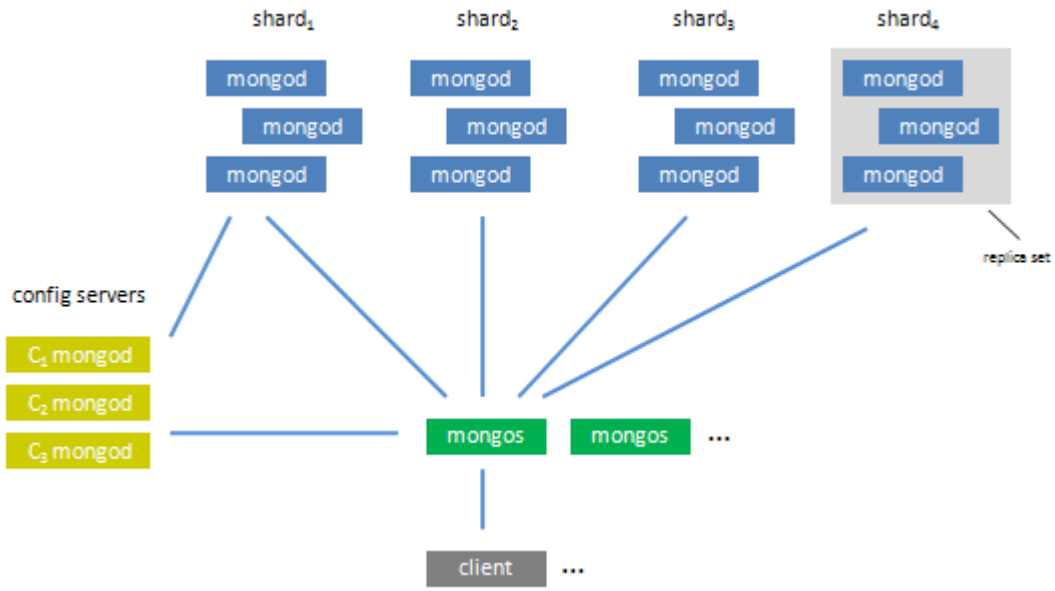
Η MongoDB (από το ''humongous'') είναι ένα διαπλατφορμικό σύστημα βάσης δεδομένων που βασίζεται σε έγγραφα (cross-platform document-oriented database system) και κατηγοριοποιείται ως NoSQL βάση δεδομένων [33]. Η MongoDB δεν έχει την παραδοσιακή δομή μίας σχεσιακής βάσης δεδομένων με πίνακες, αλλά χρησιμοποιεί JSON-like έγγραφα με δυναμικά schemas (η MongoDB καλεί αυτή τη μορφοποίηση BSON), καθιστώντας την ενσωμάτωση των δεδομένων σε ορισμένους τύπους εφαρμογών ευκολότερη και ταχύτερη. Κυκλοφορεί με άδεια χρήσης που είναι ένας συνδυασμός των GNU Affero General Public License και Apache License και αποτελεί ελεύθερο και ανοικτού κώδικα λογισμικό.

Αρχικά, αναπτύχθηκε από την εταιρεία 10gen, με βάση τη Νέας Υόρκη (πλέον ονομάζεται MongoDB Inc.) τον Οκτώβριο του 2007 ως συστατικό μιας πλατφόρμας PaaS. Στη συνέχεια, η εταιρεία στράφηκε σε ένα μοντέλο ανάπτυξης ανοιχτού κώδικα το 2009, με την 10gen να προσφέρει εμπορική υποστήριξη και άλλες υπηρεσίες. Από τότε, η MongoDB έχει υιοθετηθεί ως λογισμικό backend από μια σειρά σημαντικών ιστοσελίδων και υπηρεσιών, όπως οι Craigslist, eBay, Foursquare, SourceForge και The New York Times, μεταξύ άλλων. Η MongoDB είναι το πιο δημοφιλές NoSQL σύστημα βάσεων δεδομένων.

Η MongoDB είναι σχεδιασμένη να προσφέρει υψηλή απόδοση στις εφαρμογές, επεκτασιμότητα, υψηλή διαθεσιμότητα και δυνατότητα υποβολής σύνθετων ερωτημάτων. Για τα δεδομένα της εγγυάται τελική συνέπεια (eventual consistency).

4.3.1 Αρχιτεκτονική συστήματος

Με τη MongoDB είναι δυνατή η ανάπτυξη κατανεμημένων εφαρμογών που τρέχουν σε clusters υπολογιστών, όπου με την προσθήκη νέων κόμβων πραγματοποιείται αυτόματη κατανομή του φόρτου εργασίας. Σε ένα cluster υπολογιστών, διακρίνονται τρεις τύποι κόμβων, οι shards, οι configuration servers και οι routers. Αυτή η δομή είναι δυνατή λόγω του sharding που εμφανίζεται στα δεδομένα και θα περιγραφεί παρακάτω.



Σχήμα 4.4: MongoDB architecture

Shard nodes

Οι shard nodes είναι υπεύθυνοι για την αποθήκευη των δεδομένων. Κάθε shard αποτελείται από ένα replica set, δηλαδή έναν ή περισσότερους κόμβους που διατηρούν αντίγραφα των ίδιων δεδομένων. Από τους κόμβους που δομούν το shard, ένας θεωρείται πρωτεύων (primary) και οι υπόλοιποι δευτερεύοντες (secondary). Σε περίπτωση αποτυχίας του πρωτεύοντος, αναλαμβάνει ως πρωτεύων ένας από τους δευτερεύοντες. Τα writes και τα reads δρομολογούνται στον πρωτεύων κόμβο, ενώ τα eventual consistent reads κατανέμονται στους δευτερεύοντες. Η mongoDB προκειμένου να είναι γρήγορη κι ευέλικτη χρησιμοποιεί απλούς μηχανισμούς κλειδώματος (locking) για την εξυπηρέτηση ταυτόχρονων reads και writes από πολλούς πελάτες (clients) κι όχι πολύπλοκους μηχανισμούς. Το λογισμικό που χρησιμοποιούν οι shards ονομάζεται mongod.

Configuration servers

Οι configuration servers αποθηκεύουν μεταδεδομένα και πληροφορίες δρομολόγησης, που υποδεικνύουν ποια δεδομένα διατηρούνται στο κάθε shard. Η πρόσβαση σε αυτούς γίνεται μέσω των shards, οι οποίοι ενημερώνουν ποια δεδομένα διαθέτουν, και μέσω των routers, οι οποίοι ζητούν πληροφορίες δρομολόγησης των αιτημάτων που υποβάλλουν οι χρήστες. Οι configuration servers χρησιμοποιούν κι αυτοί το λογισμικό mongod.

Routers

Οι routers χρησιμοποιούνται για την επικοινωνία ενός ή περισσότερων πελατών με τη βάση. Δέχονται αιτήματα για reads και writes από τους χρήστες με τη μορφή ερωτημάτων (queries) και τροποποιήσεων (updates) της βάσης, συμβουλεύονται τους configuration servers για το ποιο shard έχει τα ζητούμενα δεδομένα και δρομολογούν τις εργασίες στα αντίστοιχα shards. Αφού ολοκληρωθούν τα αιτήματα, οι routers ομαδοποιούν τα αποτελέσματα και τα στέλνουν πίσω στο χρήστη. Το λογισμικό που χρησιμοποιούν οι routers ονομάζεται mongos.

Clients

Με τον όρο clients εννοούμε ολόκληρη την εφαρμογή ή ένα κομμάτι της. Η επικοινωνία τους με τη βάση γίνεται μέσω του κατάλληλου mongo driver, ο οποίος επιλέγεται ανάλογα με τη γλώσσα που είναι γραμμένη η εφαρμογή.

4.3.2 Αποθήκευση δεδομένων

Στη συνέχεια, θα παρουσιαστούν τα εργαλεία που χρησιμοποιούνται από τη mongoDB για την αποθήκευση και την ανάκτηση δεδομένων. Είναι ιδιαίτερα σημαντική τόσο η γνώση του τρόπου αποθήκευσης των δεδομένων όσο και η κατανόηση των μηχανισμών αξιοπιστίας κι ασφαλείας που παρέχονται. Παρακάτω αναλύονται οι διαδικασίες και οι τεχνικές αποθήκευσης των δεδομένων, καθώς και οι τεχνικές για τον γρήγορο εντοπισμό και την ανάκτησή τους.

Preallocated files

Η mongoDB χρησιμοποιεί preallocation, με σκοπό την καλύτερη επικοινωνία μιας εφαρμογής με το δίσκο. Κατά το preallocation όταν ένα έγγραφο φτάσει ένα ορισμένο μέγεθος δεσμεύεται αυτόματα χώρος για το επόμενο, πριν η εφαρμογή ζητήσει τη δημιουργία του. Με αυτό τον τρόπο αποφεύγεται και ο θρυμματισμός των εγγράφων (fragmentation), που έχει ως αποτέλεσμα πιο αργή πρόσβαση στα έγγραφα και άρα μείωση της συνολικής επίδοσης των εφαρμογών. Τα έγγραφα που δημιουργούνται με preallocation έχουν εκ των προτέρον γνωστό και σταθερό μέγεθος. Έτσι αποφεύγεται, τόσο το ο θρυμματισμός, όσο και η ανεξέλεγκτη αύξηση του μεγέθους των

εγγράφων. Τα έγγραφα αποθηκευόνται σε διαδοχικές θέσεις μνήμης, κάνοντας πιο εύκολη και γρήγορη την πρόσβαση σε αυτά.

Για παράδειγμα, γενικά όταν μια εφαρμογή χρειάζεται να κάνει write δεδομένα σε ένα έγγραφο που δεν έχει χώρο να τα αποθηκεύσει, τότε πρέπει να ζητηθεί η δημιουργία νέου εγγράφου κι στη συνέχεια να γίνει write σε αυτό, διαδικασία που είναι ιδιαίτερα χρονοβόρα. Αντίθετα, με το preallocation, που γίνεται πριν χρειαστεί να γίνουν write τα δεδομένα, το έγγραφο δημιουργείται εκ των προτέρων, με αποτέλεσμα τα writes να γίνονται άμεσα, χωρίς να χρειάζεται να περιμένουν μέχρι να δημιουργηθεί νέο έγγραφο.

Πρακτικά, για κάθε βάση δεδομένων, η mongoDB ξεκινά το preallocation με ένα έγγραφο μεγέθους 64MB. Κάθε φορά που απαιτείται νέο preallocation, το μέγεθος του εγγράφου διπλασιάζεται μέχρι να φτάσει τα 2GB. Είναι φανερό λοιπόν πως κάθε βάση στη mongoDB μπορεί να έχει χρησιμοποίητο χώρο έως 2GB, όμως στην πράξη ο αναξιοποίητος χώρος είναι ελάχιστος.

Padding

Η mongoDB διαθέτει ένα μηχανισμό εφαρμογής atomic upserts στα έγγραφα. Τα upserts είναι μια σημαία των λειτουργιών update, που μεταβάλλουν τη συμπεριφορά τους από απλή ενημέρωση των εγγράφων σε εισαγωγή νέων δεδομένων. Πιο συγκεκριμένα, όταν οι χρήστες ορίζουν ένα ερώτημα με βάση το οποίο θα εκτελεστεί μία λειτουργία update σε κάποια έγγραφα, τότε αν βρεθεί τέτοιο έγγραφο πραγματοποιείται το update, αλλιώς εκτελείται ένα insert με τα δεδομένα που έχουν οριστεί στο ερώτημα.

Αυτή η λειτουργία μπορεί να οδηγήσει σε προβλήματα στην απόδοση και την αποθήκευση. Μπορεί για παράδειγμα να χρειαστεί η μεταφορά ενός εγγράφου σε σημείο του δίσκου που να μπορεί να αποθηκευτεί σε συνεχόμενες θέσεις μνήμης, ενώ θα πρέπει να ενημερωθούν οι δείκτες και οι shards, διαδικασία ιδαίτερα χρονοβόρα. Έτσι, η mongoDB εκτελεί ευριστικούς αλγορίθμους για τον εντοπισμό εγγράφων που χρειάζεται να αυξηθούν σε μέγεθος πιο συχνά Σε αυτά τα έγγραφα εφαρμόζεται padding, δηλαδή προστίθεται στο τέλος του εγγράφου χώρος, ώστε να μην χρειαστεί η μεταφορά του αργότερα.

Memory mapped files

Για τη διαχείριση των δεδομένων, η mongoDB χρησιμοποιεί memory-mapped files. Το memory mapping προβλέπεται από το λειτουργικό σύστημα κι έτσι δεν χρειάζεται η ανάπτυξη πολύπλοκου κώδικα από την πλευρά της βάσης δεδομένων. Η mongoDB ζητάει από το λειτουργικό σύστημα να κάνει map όλα τα αρχεία από το δίσκο στη μνήμη RAM του μηχανήματος που τα διατηρεί. Τα αρχεία αντιστοιχίζονται σε blocks εικονικής μνήμης και μπορούν να τα χειριστούν σαν να ήταν αποθηκευμένα σε φυσική μνήμη. Η πρόσβαση στα δεδομένα γίνεται μέσω buffers της RAM στις εικόνες των δεδομένων που έχουν προκύψει από το memory mapping. Η mongoDB

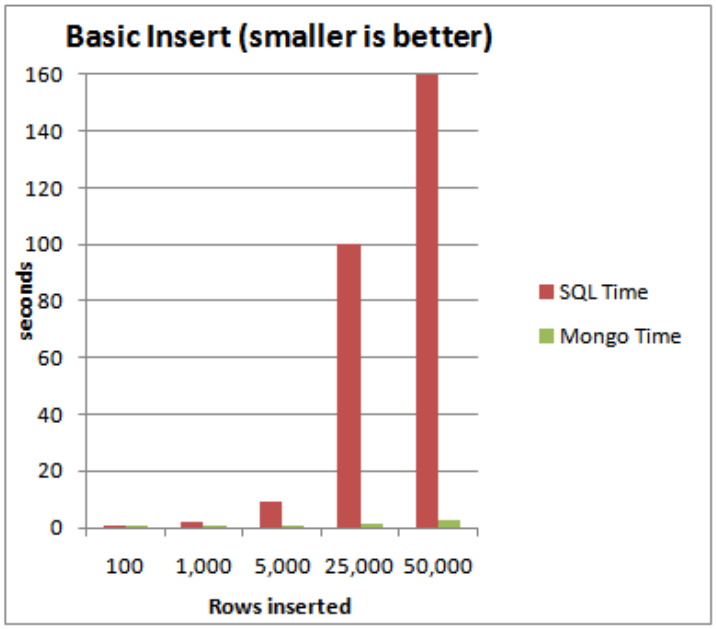
περιοδικά συγχρονίζεται με το δίσκο, αντιγράφοντας όλα τα writes που βρίσκονται στους buffers της RAM.

Παρόλα αυτά, υπάρχουν κι ορισμένα μειονεκτήματα στη χρήση του. Ένα τέτοιο παρατηρείται κατά το RAM read-ahead. Όταν ζητείται η μεταφορά δεδομένων από το δίσκο στη RAM, δεν αντιγράφονται μόνο οι ζητούμενες σελίδες, αλλά και οι επόμενες γειτονικές, τα οποία θεωρούνται ότι θα ζητηθούν αργότερα. Έτσι, αυτή γεμίζει με δεδομένα που πιθανώς δεν χρειάζονται, απομακρύνοντας δεδομένα που πρόκειται να αναζητηθούν. Αυτό μπορεί να προκύψει κι από RAM fragmentation που προκύπτει από τη μεταφορά θρυμμαρισμένων αρχείων του δίσκου στη RAM.

Journal

Για την εξασφάλιση της συνέπειας των δεδομένων στη βάση, χρησιμοποιείται write ahead logging σε ένα journal. Οι λειτουργίες write γράφονται πρώτα στο journal κι έπειτα εφαρμόζονται περιοδικά στα δεδομένα. Αν κάθε φορά που γινόταν ένα write αυτό γραφόταν στο δίσκο, τότε η επίδοση θα ήταν πολύ χειρότερη. Έτσι, το journal μαζεύει πολλά updates και inserts, τα προωθεί μαζικά στο δίσκο ανά τακτά διαστήματα και τα journal files διαγράφονται.

Ακόμα, σε περίπτωση αστοχίας υλικού, η βάση συμβουλεύεται το journal και επανεκτελεί τις λειτουργίες write στα δεδομένα. Με αυτό τον τρόπο εξασφαλίζεται η συνέπεια. Οι περισσότερες εφαρμογές διαθέτουν δύο ή τρία journal files, κάθε ένα από τα οποία κάνει preallocation 1GB, αν και υπάρχει η δυνατότητα χρήσης μικρών journal files με μέγεθος 128MB, που ονομάζονται smallfiles.



Σχήμα 4.5: MySQL vs MongoDB

4.3.3 Μοντέλο δεδομένων

Στη mongoDB τα δεδομένα αποθηκεύονται με schema-free τρόπο σε αρχεία τύπου BSON (Binary JSON), τα οποία ονομάζονται έγγραφα (documents). Κάθε έγγραφο αντιπροσωπεύει μία εγγραφή στην ορολογία των σχεσιακών βάσεων δεδομένων και περιέχει πεδία σε μορφή ζευγαριών κλειδιού-τιμής (key-value). Στο κλειδί αποθηκεύεται σε μία συμβολοσειρά το όνομα του πεδίου και στην τιμή μπορούν να αποθηκευτούν τύποι δεδομένων που υποστηρίζονται από αρχεία JSON, αλλά και άλλοι όπως ημερομηνίες. Τα έγγραφα έχουν μέγιστο μέγεθος 16MB, ενώ για την αποθήκευση μεγαλύτερων δεδομένων σε μέγεθος, όπως εικόνες, χρησιμοποιείται το GridFS [;]. Δίνεται ακόμα η δυνατότητα ενσωματωμένης αποθήκευσης αντικειμένων και πινάκων στο εσωτερικό των εγγράφων, κάτι που μειώνει την ανάγκη χρήσης ενώσεων (joins) στην πλευρά του χρήστη. Ένα σύνολο εγράφων ορίζει μία συλλογή (collection), που είναι το ανάλογο των πινάκων στις σχεσιακές βάσεις δεδομένων. Ένα σύνολο συλλογών συνιστά μία βάση δεδομένων.

Κάθε έγγραφο υποχρεωτικά πρέπει να διαθέτει ένα πεδίο όπου αποθηκεύεται το μοναδικό αναγνωριστικό id του εγγράφου και χρησιμοποιείται σαν πρωτεύον κλειδί (primary key) για τον εντοπισμό του. Το κλειδί αυτού του πεδίου ονομάζεται *_id* και η τιμή του είναι συνηθέστερα τύπου *ObjectID*, ή οποιουδήποτε άλλου τύπου εκτός από *array*, κι έχει μοναδική τιμή για κάθε έγγραφο. Σε όλες τις βάσεις δημιουργείται αυτόματα.

Παρακάτω φαίνεται το παράδειγμα ενός BSON αρχείου, στο οποίο υπάρχει και ενσωματωμένο αντικείμενο:

```
{
    "_id" : ObjectID("document_id"),
    "name" : "Jack",
    "birth" : new Date("Apr 25, 1995"),
    "phone" : {
                "type" : "mobile",
                "number" : "some_number"
              },
    "languages" : ["English", "German", "French"]
}
```

Σχήμα 4.6: Παράδειγμα BSON αρχείου

Η mongoDB παρέχει αρκετές ελευθερίες στον τρόπο αποθήκευσης των δεδομένων. Εφόσον η βάση δομείται με schema-free τρόπο, δεν χρειάζεται τα έγγραφα μίας συλλογής να έχουν τα ίδια πεδία, την ίδια δομή και τους ίδιους τύπους δεδομένων και άρα μπορεί το καθένα να διαθέτει δικό του schema. Για την αποτελεσματικότερη εκτέλεση των εφαρμογών και για την επιλογή του κατάλληλου schema από τους σχεδιαστές πρέπει να λαμβάνονται υπ' όψη τα ερωτήματα που θα υποβάλλονται στη βάση, η συχνότητα των updates και ο ρυθμός αύξησης του μεγέθους των εγγράφων.

Βλέπουμε πως η μοντελοποίηση πρέπει να γίνεται με βάση τη χρήση και το περιεχόμενο των βάσεων κι όχι με βάση τον τρόπο αποθήκευσής τους, όπως γίνεται στις σχεσιακές βάσεις δεδομένων όπου η εκτέλεση των ερωτημάτων βασίζεται στις συνενώσεις.

Για την εξαγωγή αποτελεσμάτων μέσω ερωτημάτων είναι απαραίτηση η αναπαράσταση των σχέσων μεταξύ των εγγράφων. Η συσχέτιση των δεδομένων στη mongoDB γίνεται με δύο τρόπους, με τη χρήση ενσωματωμένων αντικειμένων (embedded objects) και τη χρήση αναφορών (references).

Embedded documents
Η τεχνική της ενσωματωμένης αποθήκευσης αντικειμένων και πινάκων εντός των εγγράφων χρησιμοποιείται όταν τα ενσωματωμένα δεδομένα εξετάζονται κυρίως σε συνδιασμό με τα έγγραφα που τα περιέχουν. Καθώς η επεξεργασία πληροφοριών του ίδιου εγγράφου απαιτεί λιγότερο φόρτο εργασίας στους κόμβους συγκριτικά με τη συλλογή δεδομένων από διαφορετικά έγγραφα, η τεχνική της ενσωματωμένης αποθήκευσης εγγράφων προτιμάται όταν είναι δυνατό. Ένα μειονέκτημα της μεθόδου αυτής είναι το γεγονός ότι μπορεί να αυξηθεί ανεξέλεγκτα το μέγεθος ενός εγγράφου.

References
Στις περιπτώσεις που το παραπάνω μοντέλο δεν βολεύει τους σχεδιαστές, η mongoDB δίνει την εναλλακτική επιλογή των αναφορών. Τα έγγραφα που συσχετίζονται μπορεί να ανήκουν σε διαφορετικές συλλογές ή ακόμα και σε διαφορετικές βάσεις δεδομένων. Οι αναφορές λειτουργούν σαν τα εξωτερικά κλειδιά (foreign keys) των σχεσιακών βάσεων δεδομένων κι έχουν ως σκοπό τον ορισμό σχέσεων μεταξύ των εγγράφων. Με τη χρήση αυτής της τεχνικής μπορεί να ελεγχθεί το μέγεθος των εγγράφων, αλλά απαιτείται μεταφορά περισσότερων δεδομένων για τη διενέργεια ερωτημάτων.

Παρακάτω φαίνεται το παράδειγμα ενός BSON αρχείου με αναφορές:

```
original_id = ObjectId()

db.places.insert({
    "_id": original_id
    "name": "Broadway Center"
    "url": "bc.example.net"
})

db.people.insert({
    "name": "Erin"
    "places_id": original_id
    "url":  "bc.example.net/Erin"
})
```

Σχήμα 4.7: Παράδειγμα BSON αρχείου με χρήση αναφορών

4.3.4 Μοντέλο ερωτημάτων

Για την υποβολή ερωτημάτων χρησιμοποιούνται JSON-like αρχεία, τα οποία στέλνονται σαν BSON αντικείμενα μέσω του driver που χρησιμοποιεί η εφαρμογή, ανάλογα με τη γλώσσα προγραμματισμού που είναι γραμμένη. Στα αρχεία αυτά εμπεριέχονται οι τιμές και οι συνθήκες που πρέπει να ικανοποιούν τα έγγραφα για να συμπεριληφθούν στο αποτέλεσμα που θα επιστραφεί. Τα ερωτήματα υποβάλλονται σε όλα τα έγγραφα μίας συλλογής κάθε φορά και συμπεριλαμβάνονται όλα τα ενσωματωμένα αντικείμενα και οι πίνακες που περιέχουν. Για την υποβολή ερωτημάτων σε έγγραφα περισσότερων συλλογών, θα πρέπει το ερώτημα να εφαρμόζεται σε όλες τις συλλογές που διαθέτουν τα ζητούμενα έγγραφα.

Παρακάτω φαίνεται ένα παράδειγμα ενός ερωτήματος στη mongoDB:

```
{"name": "Jack", "phone.type": "mobile", "languages": "German"}
```

Το μοντέλο ερωτημάτων της mongoDB υποστηρίζει μεταξύ άλλων τα ακόλουθα χαρακτηριστικά:

- Υποβολή ερωτημάτων σε έγγραφα και σε ενσωματωμένα αντικείμενα.
- Υποβολή geospatial ερωτημάτων. Παρακάτω φαίνεται παράδειγμα ενός τέτοιου ερωτήματος:

```
{ _id : ObjectId(...),
  locs : [ [ 55.5 , 42.3 ] ,
           [ -74 , 44.74 ] ,
           { lng : 55.5 , lat : 42.3 } ]
}
```

Σχήμα 4.8: Παράδειγμα geospatial ερωτήματος

- Χρήση τελεστών σύγκρισης ($<, >, \leq, \geq, =$).
- Χρήση λογικών τελεστών (*and, or, not, nor*).
- Χρήση τελεστών ικανοποιήσης συνθηκών (*equals, exists, in, mod, type, ...*).
- Χρήση συναρτήσεων ομαδοποίησης (*count, sum, ...*).

Για την υλοποίηση πολύπλοκων συναρτήσεων ομαδοποίησης, η mongoDB έχει υλοποιήσει μία παραλλαγή του MapReduce. Η map function επεξεργάζεται την είσοδο, παρέχει ζευγάρια κλειδιών-τιμέων, τα ομαδοποιεί με βάση το κλειδί και για κάθε ένα προκύπτει ένας πίνακας από τιμές. Τα αποτελέσματα προωθούνται στη reduce function, η οποία εφαρμόζεται σε όλα τα στοιχεία του πίνακα τιμών κι επιστρέφει μία μοναδική τιμή για το κάθε κλειδί. Τα αποτελέσματα αποθηκεύονται σε μία συλλογή (είτε αντικαθιστώντας τα δεδομένα της, είτε δημιουργώντας την αν δεν υπάρχει). Οι

χρήστες μπορούν να εφαρμόζουν ταξινόμηση στα δεδομένα εισόδου, ώστε να γίνονται λιγότερες πράξεις.

Ο χρήστης μπορεί προαιρετικά να ορίσει μία ακόμα συνάρτηση, τη finalize function, η οποία εφαρμόζεται στα αποτελέσματα της reduce function και δίνει ένα μοναδικό αποτέλεσμα. Χρησιμοποιείται για την ομαδοποίηση αποτελεσμάτων, όπως τον υπολογισμό ενός μέσου όρου.

4.3.5 Άλλα χαρακτηριστικά

Indexing

Για γρηγορότερη κι αποδοτικότερη εκτέλεσητων ερωτημάτων, χρησιμοποιούνται ευρετήρια (indexes), που είναι δομές δεδομένων που εντοπίζουν γρήγορα κάποια έγγραφα με βάση ένα ή περισσότερα συγκεκριμένα πεδία. Τα ευρετήρια αποθηκεύονται σε μορφή B-δέντρων και είναι όμοια με αυτά που συναντώνται και σε άλλες βάσεις δεδομένων. Εφαρμόζονται στα κλειδιά των πεδίων ενός εγγράφου μίας συλλογής. Η mongoDB δημιουργεί αυτόματα ευρετήριο με βάση το id του κάθε εγγράφου, αλλά μπορούν να οριστούν κι άλλα ευρετήρια για τα άλλα πεδία ενός εγγράφου.

Η δημιουργία των κατάλληλων ευρετηρίων γίνεται από τους σχεδιαστές ανάλογα με τη συχνότητα που εκτελούνται τα ερωτήματα και την αναλογία των reads και writes που γίνονται στα έγγραφα. Αν και η διατήρηση πολλών ευρετηρίων μπορεί να δημιουργεί καθυστέρηση στα updates, η προσεκτική δημιουργία τους μπορεί να κάνει την εκτέλεση ερωτημάτων ταχύτατη. Μία καλή πρακτική για γρήγορη εκτέλεση των ερωτημάτων είναι να διατηρούνται τα ευρετήρια στη RAM. Σε κάθε ερώτημα, χρησιμοποιείται μόνο ένα ευρετήριο, η επιλογή του οποίου γίνεται από τον βελτιστοποιητή ερωτημάτων (query optimizer).

Replication

Για την εξασφάλιση συνέπειας και υψηλής διαθεσιμότητας, οι βάσεις δεδομένων υλοποιούν μηχανισμούς φύλαξης αντιγράφων (replication). Η mongoDB χρησιμοποιεί τα replica sets, κάθε ένα από τα οποία είναι συνήθως ένας shard node, που αποτελείται από έναν πρωτεύων και κάποιους δευτερεύοντες κόμβους. Ο πρωτεύων κόμβος είναι υπεύθυνος για όλα τα writes και τα consistent reads και διατηρεί αρχείο log με τις εργασίες που έχουν γίνει σε αυτόν. Οι δευτερεύοντες κόμβοι ενημερώνονται ασύγχρονα από το log του πρωτεύοντος κόμβου κι εφαρμόζουν τις αλλαγές που τους αφορούν.

Με τη χρήση replica sets διασφαλίζεται αυτόματη εναλλακτική σύνδεση σε περίπτωση σφάλματος. Αν ο πρωτεύων κόμβος αποτύχει, τότε εκλέγεται αυτόματα ένας νέος από τους δευτερεύοντες κόμβους τους replica set. Όταν οι κόμβοι που έχουν αστοχήσει επανέλθουν, συγχρονίζονται με τον πρωτεύων και συνεχίζουν να λειτουργούν ως δευτερεύοντες.

Auto-sharding

Με τον όρο sharding εννοούμε τον καταμερισμό μιας βάσης δεδομένων σε ένα cluster. Τα cluster στη mongoDB αποτελούνται από shard nodes, στους οποίους αποθηκεύονται τα δεδομένα, από configuration servers κι από routers. Η mongoDB υποστηρίζει αυτόματο sharding, ισοκατανέμοντας τον όγκο των δεδομένων και το φόρτο εργασίας σε όλους τους κόμβους ενός cluster. Με την προσθήκη νέων κόμβων, τα δεδομένα καταμερίζονται αυτόματα, παρέχοντας πλήρη επεκτασιμότητα στη βάση.

Οι σχεδιαστές επιλέγουν σε ποιες βάσεις και σε ποιες συλλογές τους θα ενεργοποιήσουν το sharding. Κάθε τέτοια συλλογή διαμερίζει τα έγγραφά της σύμφωνα με ένα ορισμένο shard key, έτσι ώστε κάθε shard να πάρει έγγραφα με ένα συγκεκριμένο εύρος τιμών. Έτσι, κάθε shard διατηρεί ταξινομημένα σύμφωνα με το shard key τα έγγραφα που του έχουν ανατεθεί. Τα έγγραφα στα shards χωρίζονται σε chunks, που το καθένα διαθέτει ταξινομημένα έγγραφα από ένα start shard key, μέχρι ένα end shard key. Αν ένα chunk μεγαλώσει πολύ, τότε χωρίζεται, επιτρέποντας την ισοκατανομή των δεδομένων στο cluster με την αυτόματη μετακίνησή τους.

Κεφάλαιο 5

Άλλες χρησιμοποιούμενες τεχνολογίες

Για την ανάπτυξη, τη λειτουργία και τον έλεγχο του συστήματος που υλοποιήθηκε, έγινε χρήση διάφορων σύγχρονων τεχνολογιών, πλατφορμών και προγραμματιστικών εργαλείων. Σαν κύρια γλώσσα προγραμματισμού των κλάσεων χρησιμοποιήθηκε η Java 7, ενώ αναπτύχθηκαν Java Server Pages (JSP) και Servlets για το γραφικό περιβάλλον της εφαρμογής. Σαν εξυπηρετητής δυαδικτυακών υπηρεσιών χρησιμοποιήθηκε ο Apache Tomcat server 7 και σαν πλατφόρμα προγραμματισμού το προγραμματιστικό περιβάλλον Eclipse. Επίσης, χρησιμοποιήθηκαν αρκετά και οι γλώσσες σήμανσης XML και JSON. Όλες οι παραπάνω τεχνολογίες αναλύονται εκτενέστερα παρακάτω.

5.1 Java

Η Java είναι μια από τις πιο διαδεδομένες αντικειμενοστρεφείς γλώσσες προγραμματιστού. Σχεδιάστηκε αρχικά από τη Sun Microsystems και κυκλοφόρησε πρώτη φορά το 1995 σαν ένα στοιχείο του πυρήνα της πλατφόρμας Java. Η σύνταξή της προέρχεται κατά κύριο μέρος από τις γλώσσες προγραμματισμού C και C++, αλλά έχει πιο απλό αντικειμενοστρεφές μοντέλο και λιγότερες ευκολίες χαμηλού επιπέδου. Οι εφαρμογές Java μεταγλωττίζονται τυπικά σε δυαδικό κώδικα, ο οποίος μπορεί να εκτελεστεί σε οποιαδήποτε εικονική μηχανή της Java (Java Virtual Machine - JVM) ανεξαρτήτως της αρχιτεκτονικής του υπολογιστικού συστήματος που εκτελείται. Ο ίδιος ο δημιουργός της Java, James Gosling, είχε υποσχεθεί μία γλώσσα "Write once, run anywhere", που σημαίνει ότι οποιοδήποτε λογισμικό που έχει αναπτυχθεί με Java μπορεί να εκτελεστεί οπουδήποτε χωρίς την παραμικρή αλλαγή.

Οι αρχικοί μεταγλωττιστές της Java, οι εικονικές μηχανές και οι βιβλιοθήκες κλάσεων αναπτύχθηκαν από τη Sun το 1995. Από το 2007 και σε συνεργασία με τις προδιαγραφές της Java Community Process η Sun έκανε διαθέσιμο το μεγαλύτερο μέρος της τεχνολογίας της ως ελεύθερο λογισμικό κάτω από την άδεια General Public Licence (GNU).

5.1.1 Στόχοι και χαρακτηριστικά

Οι πέντε πρωτεύοντες στόχοι κατά τη δημιουργία της Java ήταν ότι η γλώσσα Java θα έπρεπε:

1) να είναι απλή, αντικειμενοστρεφής και φιλική προς τον προγραμματιστή.

2) να είναι σταθερή και ασφαλής, επιτρέποντας την εκτέλεση εφαρμογών από απομακρυσμένες πηγές με ασφάλεια.

3) να είναι αρχιτεκτονικά ουδέτερη και μεταφέρσιμη, ώστε να εκτελείται η ίδια εφαρμογή σε οποιαδήποτε αρχιτεκτονική υπολογιστικού συστήματος.

4) να εκτελείται με υψηλή απόδοση.

5) να είναι διερμηνεύσιμη, να έχει νήματα και να είναι δυναμική.

Οι παραπάνω προδιαγραφές δηλώνουν τα πλεονεκτήματα που απορρέουν από τη χρήση της Java σαν κύρια γλώσσα ανάπτυξης λογισμικού. Πιο αναλυτικά, εξαιτίας της κατασκευής της ως αντικειμενοστρεφής γλώσσα παρέχει απλότητα στο σχεδιασμό, διαχωρίζει τον κώδικα σε ξεχωριστές και αυτόνομες μονάδες και καθιστά εφικτές μικρές αλλαγές στον κώδικα σε πολύ σύντομο χρονικό διάστημα, χωρίς να επηρεάζει τη συνολική λειτουργία του. Ανάλογα εύκολη είναι και η προσθήκη νέων λειτουργιών, ο εντοπισμός και η διόρθωση σφαλμάτων, η επαναχρησιμοποίηση κλάσεων και μεθόδων σε διαφορετικά προγράμματα και η ανάπτυξη και συντήρηση των εφαρμογών.

Ιδιαίτερα σημαντική είναι η δυνατότητα της Java να μπορεί να εκτελεστεί πάντα και σε οποιαδήποτε πλατφόρμα υλικού ή λειτουργικού συστήματος, με αποτέλεσμα να είναι ανεξάρτητη από την πλατφόρμα εκτέλεσης. Με αυτό τον τρόπο είναι δυνατή η διασύνδεση ετερογενών εφαρμογών, αλλά και δικτύων που αποτελούνται από διαφορετικά υπολογιστικά συστήματα.

Τα σημαντικότερα χαρακτηριστικά της πλατφόρμας της Java είναι τα παρακάτω:

- Απλότητα. Έχει ένα σύντομο και συνεκτικό σύνολο χαρακτηριστικών που την καθιστά εύκολη στην εκμάθηση και στη χρήση. Τα περισσότερα χαρακτηριστικά τα έχει αντλήσει από την C++, κάνοντας την οικεία στους προγραμματιστές.

- Ασφάλεια. Ένα πρόγραμμα γραμμένο σε Java δεν μπορεί να επηρεάσει ένα άλλο σύστημα. Παρέχει ένα ασφαλή τρόπο για την ανάπτυξη εφαρμογών ιστού, ενώ παράλληλα παρέχει και ασφαλή τρόπο πρόσβασης σε αυτές.

- Μεταφερσιμότητα. Τα προγράμματα σε Java μπορούν να εκτελεστούν σε οποιοδήποτε περιβάλλον, στο οποίο είναι εγκατεστημένη μία εικονική μηχανή της Java και άρα σε οποιοδήποτε λειτουργικό σύστημα (Linux, Microsoft Windows, Macintosh). Έτσι, τα προγραμμάτα σε Java μπορούν να μεταφερθούν μέσω του internet.

- Αντικειμενοστρέφεια. Η Java είναι ξεκάθαρα αντικειμενοστρεφής γλώσσα, που παρέχει όλα τα χαρακτηριστικά αντικειμενοστρέφειας.

- Στιβαρότητα. Η Java ενθαρρύνει τον προγραμματισμό χωρίς σφάλματα, έχοντας αυστηρό σύστημα τύπων και κάνοντας ελέγχους κατά την εκτέλεση του προγράμματος.

- Πολυνηματικότητα. Η Java παρέχει ενσωματωμένη υποστήριξη για πολυνηματικό προγραμματισμό.

- Αρχιτεκτονική ουδετερότητα. Η Java δεν δεσμεύεται με καμία αρχιτεκτονική μηχανής ή λειτουργικού συστήματος. Έτσι, είναι ανεξάρτητη από το υλικό, αλλά και το λογισμικό του υπολογιστή στον οποίο τρέχει.

- Διερμηνευσιμότητα. Η Java υποστηρίζει διαπλατφορμικό κώδικα, μέσω του Java bytecode, το οποίο διερμηνεύεται σε οποιαδήποτε πλατφόρμα από την εικονική μηχανή της Java.

- Υψηλή αποδοτικότητα. Ο Bytecode είναι πολύ βελτιστοποιήσιμος και εκτελείται πολύ γρήγορα από την εικονική μηχανή της Java.

5.1.2 Χαρακτηριστικά της γλώσσας Java

Μερικά σημαντικά χαρακτηριστικά υλοποίησης της γλώσσας Java είναι τα εξής:

Αυτόματη διαχείριση μνήμης

Η Java χρησιμοποιεί έναν αυτόματο συλλέκτη σκουπιδιών για τη διαχείριση της μνήμης κατά τον κύκλο ζωής ενός αντικειμένου. Ο προγραμματιστής καθορίζει πότε δημιουργούνται τα αντικείμενα και η εικονική μηχανή εκτέλεσης της Java είναι υπεύθυνη για την ανάκτηση της μνήμης, όταν τα αντικείμενα δεν είναι πλέον σε χρήση. Όταν δεν υπάρχουν αναφορές σε ένα αντικείμενο, η μη προσπελάσιμη μνήμη απελευθερώνεται αυτόματα από το συλλέκτη σκουπιδιών. Επιπλέον, μπορεί ακόμη να εμφανιστεί κάτι παρόμοιο με διαρροή μνήμης, αν ο κώδικας ενός προγραμματιστή διατηρεί μια αναφορά σε ένα αντικείμενο που δεν είναι πλέον απαραίτητο. Εάν καλούνται μέθοδοι για ένα ανύπαρκτο αντικείμενο, τότε ρίχνεται μία εξαίρεση "null pointer exception".

Μία από τις ιδέες πίσω από το μοντέλο αυτόματης διαχείρισης μνήμης, είναι ότι οι προγραμματιστές μπορούν να γλιτώσουν το βάρος της χειροκίνητης διαχείρισης μνήμης. Σε ορισμένες γλώσσες, η μνήμη για τη δημιουργία αντικειμένων δεσμεύεται στη στοίβα, ενώ σε άλλες η μνήμη δεσμεύεται κι αποδεσμεύεται από το σωρό. Στην τελευταία περίπτωση, η ευθύνη της διαχείρισης μνήμης ανατίθεται στον προγραμματιστή. Αν το πρόγραμμα δεν αποδεσμεύει ένα αντικείμενο, τότε παρουσιάζεται διαρροή μνήμης.

Κληρονομικότητα

Διαφορετικά είδη των αντικειμένων έχουν συχνά ένα ορισμένο σύνολο χαρακτηριστικών κοινών μεταξύ τους. Για παράδειγμα, ο κύκλος, το τετράγωνο και το τρίγωνο είναι όλα σχήματα. Παρόλα αυτά, καθένα από αυτά έχουν επιπρόσθετα χαρακτηριστικά και μπορεί να χρειάζονται διαφορετικές λειτουργίες για την επεξεργασία τους.

Ο αντικειμενοστρεφής προγραμματισμός επιτρέπει σε ορισμένες κλάσεις να κληρονομούν την κατάσταση και τη συμπεριφορά από άλλες κλάσεις. Σε αυτό το παράδειγμα, η κλάση ¨Σχήμα' μπορεί να είναι υπερκλάση των κλάσεων ¨κύκλος¨, τετράγωνο' και ¨τρίγωνο¨. Στη γλώσσα προγραμματισμού Java, κάθε κλάση έχει το δικαίωμα να έχει μία άμεση υπερκλάση και κάθε υπερκλάση έχει τη δυνατότητα για απεριόριστο αριθμό υποκλάσεων.

Εξαιρέσεις

Μια εξαίρεση είναι ένα γεγονός που συμβαίνει κατά την εκτέλεση ενός προγράμματος και διαταράσσει τη φυσιολογική ροή των εντολών. Υπάρχουν τρεις κατηγορίες των εξαιρέσεων:

- Ελεγμένες εξαιρέσεις (Checked exceptions): Μια ελεγμένη εξαίρεση αποτελεί εξαίρεση, που συνήθως αποτελεί σφάλμα χρήστη ή πρόβλημα που δεν μπορεί να προβλεφθεί από τον προγραμματιστή. Για παράδειγμα, εάν να ανοίξει ένα αρχείο αλλά δεν μπορεί να βρεθεί, εγείρεται μία εξαίρεση. Οι εξαιρέσεις αυτές δεν είναι δυνατόν να αγνοηθούν κατά τη στιγμή της μεταγλώττισης.

- Εξαιρέσεις κατά τη διάρκεια εκτέλεσης (Runtime exceptions): Ένα σφάλμα χρόνου εξαίρεσης είναι μία εξαίρεση που ίσως θα μπορούσε να είχε αποφευχθεί από τον προγραμματιστή. Σε αντίθεση με τις ελεγμένες εξαιρέσεις, οι εξαιρέσεις χρόνου εκτέλεσης αγνοούνται κατά τη στιγμή της μεταγλώττισης.

- Λάθη (Errors): Αυτά δεν είναι εξαιρέσεις, αλλά προβλήματα που προκύπτουν πέρα από τον έλεγχο του χρήστη ή τον προγραμματιστή. Τα λάθη κατά κανόνα αγνοούνται στον κώδικα, γιατί σπάνια μπορεί να γίνει κάτι για την αντιμετώπιση ενός λάθους. Για παράδειγμα, αν συμβεί μια υπερχείλιση στοίβας, θα εγερθεί ένα σφάλμα. Αυτά τα σφάλμα επίσης αγνοούνται κατά τη διαδικασία της μεταγλώττισης.

Generics

Το 2004, προστέθηκαν στη γλώσσα Java τα generics, ως μέρος της J2SE 5.0. Πριν από την εισαγωγή των generics, κάθε δήλωση μεταβλητής πρέπει να είναι ενός συγκεκριμένου τύπου. Για τις container κλάσεις, για παράδειγμα, αυτό είναι ένα πρόβλημα διότι δεν υπάρχει κανένας εύκολος τρόπος για να δημιουργηθεί ένας container που δέχεται μόνο συγκεκριμένους τύπους αντικειμένων. Είτε ο container

λειτουργεί σε όλους τους υποτύπους της κλάσης ή της διεπαφής, συνήθως Object, ή μια διαφορετική κλάση container θα πρέπει να δημιουργηθεί για κάθε κατηγορία που περιλαμβάνεται. Τα generics επιτρέπουν τον ελέγχου τύπων κατά το χρόνο μεταγλώττισης, χωρίς να χρειάζεται να δημιουργηθεί ένας μεγάλος αριθμός container κλάσεων, που καθεμιά περιέχει σχεδόν πανομοιότυπο κώδικα. Εκτός από την δυνατότητα δημιουργίας πιο αποδοτικού κώδικα, ορισμένες εξαιρέσεις χρόνου εκτέλεσης (runtime exceptions) μετατρέπεται σε σφάλματα χρόνου μεταγλώττισης (compile-time errors), χαρακτηριστικό γνωστό ως ασφάλεια τύπων.

5.1.3 Servlets

Το servlet είναι μια κλάση της γλώσσας προγραμματισμού Java, που χρησιμοποιείται για να επεκτείνει τις δυνατότητες του εξυπηρετητή. Παρά το γεγονός ότι τα servlets μπορούν να ανταποκριθούν σε κάθε είδους αιτήματα, χρησιμοποιούνται συνήθως για την επέκταση των δυνατοτήτων ορισμένων εφαρμογών που φιλοξενούνται σε εξυπηρετητές ιστού. Έτσι, αυτές μπορούν να θεωρηθούν ως Java Applets και τρέχουν στους εξυπηρετητές αντί για τους περιηγητές ιστού. Αυτά τα είδη servlets είναι τα αντίστοιχα της Java με άλλες τεχνολογίες δυναμικού περιεχομένου στον παγκόσμιο ιστό, όπως η PHP και η ASP.NET.

Τα servlets πιο συχνά χρησιμοποιούνται για:

- Την επεξεργασία και την αποθήκευση των δεδομένων που υποβλήθηκαν από μια φόρμα HTML.

- Την παροχή δυναμικού περιεχομένου, όπως τα αποτελέσματα ενός ερωτήματος σε μία βάση δεδομένων.

- Τη διαχείριση πληροφοριών κατάστασης που δεν υπάρχουν στο stateless πρωτόκολλο HTTP.

Από τεχνική άποψη, ένα servlet είναι μια κλάση Java που συμμορφώνεται με το Java Servlet API, ένα πρότυπο για την υλοποίηση Java κλάσεων που ανταποκρίνεται σε αιτήματα. Τα servlets θα μπορούσαν να επικοινωνούν πάνω από οποιοδήποτε πρωτόκολλο πελάτη-εξυπηρετητή, αλλά χρησιμοποιούνται πιο συχνά με το πρωτόκολλο HTTP κι έτσι συχνά ονομάζονται HTTP servlets. Έτσι, ένας προγραμματιστής λογισμικού μπορεί να χρησιμοποιήσει ένα servlet για να προσθέσει δυναμικό περιεχόμενο σε μία ιστοσελίδα, χρησιμοποιώντας την πλατφόρμα της Java. Το περιεχόμενο που δημιουργείται είναι συνήθως HTML, αλλά μπορεί να είναι άλλος τύπος δεδομένων, όπως XML.

Για την ανάπτυξη και την εκτέλεση ενός servlet, πρέπει να χρησιμοποιείται ένας web container, ο οποίος (επίσης γνωστός ως servlet container) είναι ουσιαστικά το συστατικό ενός εξυπηρετητή που αλληλεπιδρά με τα servlets.

5.1.4 Java Server Pages (JSP)

Τα JSP είναι μια τεχνολογία που βοηθά τους προγραμματιστές να δημιουργήσουν ιστοσελίδες που κατασκευάζονται δυναμικά, βασιζόμενες σε αρχεία HTML, XML, ή άλλους τύπους εγγράφων. Η τεχνολογία JSP, η οποία κυκλοφόρησε το 1999 από την Sun Microsystems, είναι παρόμοια με τα αρχεία PHP, αλλά χρησιμοποιεί τη γλώσσα προγραμματισμού Java. Για την ανάπτυξη και την εκτέλεση JavaServer Pages, απαιτείται ένας συμβατός εξυπηρετητής ιστού, όπως ο Apache Tomcat ή ο Jetty.

Τα JSP επιτρέπουν την παρεμβολή κώδικα Java κι ορισμένων προκαθορισμένων ενεργειών, μέσα στατικό περιεχόμενο ιστοσελίδων, με τη σελίδα που προκύπτει να μεταγλωττίζεται και να εκτελείται στον εξυπηρετητή για να παράξει το τελικό έγγραφο. Οι μεταγλωττισμένες σελίδες, καθώς και τυχόν εξαρτώμενες βιβλιοθήκες της Java, χρησιμοποιούν Java bytecode κι όπως κάθε άλλο πρόγραμμα γραμμένο σε Java πρέπει να εκτελούνται σε μία εικονική μηχανή της Java (JVM). Τα JSPs συνήθως χρησιμοποιούνται για την παραγωγή HTML και XML αρχείων, αλλά και για άλλες μορφές δεδομένων μέσω της χρήσης του OutputStream.

Οι JSP σελίδες χρησιμοποιούν διάφορα διαχωριστικά για τις λειτουργίες scripting. Τα πιο βασικά είναι τα ''<% ... %>'', τα οποία περικλείουν μια δέσμη ενεργειών JSP (Scriptlet). Αυτή η δέσμη ενεργειών είναι ένα τμήμα κώδικα γραμμένου σε Java, το οποίο εκτελείται όταν ο χρήστης ζητά τη σελίδα. Άλλα συνήθη διαχωριστικά είναι τα ''<% = ... %>'' για εκφράσεις, όπου η δέσμη ενεργειών και τα διαχωριστικά αντικαθίστανται με το αποτέλεσμα της αποτίμησης της έκφρασης, καθώς και τα διαχωριστικά ''<%˜ ... %>'' για εντολές του μεταγλωττιστή.

Από Java Server Pages μπορούν να παραχθούν αυτόματα από το μεταγλωττιστή Servlets. Η διαφορά μεταξύ των servlets και των JSP είναι ότι στα servlets ενσωματώνεται κώδικας HTML μέσα σε κώδικα γραμμένο σε Java, ενώ στα JSP ενσωματώνεται κώδικας Java μέσα σε HTML.

5.2 Apache Tomcat

Ο Apache Tomcat container είναι ένας δικτυακός container ανοιχτού λογισμικού (εξυπηρετητής εφαρμογών). Περιλαμβάνει τον Apache Tomcat server, που είναι ένας εξυπηρετητής ιστού ανοικτού κώδικα και Servlet container, που αναπτύχθηκε από την Apache Software Foundation (ASF). Ο Tomcat υλοποιεί τα πρότυπα Servlet της Java και JavaServer Pages (JSP) από τη Sun Microsystems, και παρέχει ένα περιβάλλον εξυπηρετητή ιστού Java HTTP, ώστε να τρέχει κώδικας Java.

Σε αντιστοιχεία με τον Apache Tomcat server, ο οποίος είναι ένας Servlet και JSP server που εξυπηρετεί τεχνολογίες Java, υπάρχει και ο Apache HTTP server που εξυπηρετεί αιτήματα HTTP. Αυτός είναι γρηγορότερος όταν έχουμε να κάνουμε με στατικές σελίδες, είναι περισσότερο παραμετροποίησιμος από τον Tomcat, είναι πιο σταθερός και υποστηρίζει CGI scripts, Server API modules, Perl, PHP και άλλα, όμως δεν υποστηρίζει JSP και Servlets.

Έχει αναπτυχθεί σε ένα περιβάλλον ανοιχτό, με τη συνεργασία πολλών κορυφαίων

προγραμματιστών κι εταιρειών και έχει εκδοθεί υπό την άδεια λογισμικού Apache License, v2. Πάνω σε Apache Tomcat τρέχουν πολλές μεγάλης κλίμακας, κρίσιμες δικτυακές εφαρμογές από ένα ευρύ φάσμα βιομηχανιών και οργανισμών.

Ο Apache Tomcat περιλαμβάνει εργαλεία για τη διαμόρφωση και τη διαχείριση του, αλλά μπορεί επίσης να ρυθμιστεί κι από την επεξεργασία αρχείων XML. Μερικοί από τους σημαντικότερους καταλόγους του Tomcat είναι οι εξής:

- /bin: Περιέχει script εκκίνησης, τερματισμού και άλλα. Τα αρχεία *.sh (για συστήματα Unix) είναι λειτουργικά αντίγραφα των *.bat αρχεία (για Windows).

- /conf: Αρχεία διαμόρφωσης και ρυθμίσεων. Το πιο σημαντικό αρχείο εδώ είναι το server.xml, το οποίο είναι το κύριο αρχείο ρυθμίσεων για τον container.

- /logs: Τα αρχεία καταγραφής γεγονότων (log files) βρίσκονται εδώ.

- /webapps: Σε αυτό τον κατάλογο τοποθετούνται οι δικτυακές εφαρμογές που υλοποιούνται.

5.3 JSON

Το JSON (JavaScript Object Notation), είναι ένα ανοιχτό πρότυπο μορφοποίησης που χρησιμοποιεί κείμενο αναγνώσιμο από τον άνθρωπο για τη μετάδοση αντικειμένων δεδομένων, που αποτελούνται από ζεύγη γνωρίσματος-τιμής. Χρησιμοποιείται κυρίως για τη μετάδοση δεδομένων μεταξύ ενός εξυπηρετητή και μίας εφαρμογής ιστού, ως εναλλακτική λύση στο XML. Ο κυριότερος λόγος που οδήγησε στην ανακάλυψη και την ευρεία υιοθέτηση του JSON ήταν η ανάγκη για stateful, πραγματικού χρόνου επικοινωνία μεταξύ των εξυπηρετητών και των φυλλομετρητών ιστού, χωρίς τη χρήση plugin του φυλλομετρητή, όπως το Flash ή Java applets, που ήταν τότε η κυρίαρχη σχεδίαση.

Παρά το γεγονός ότι προέρχεται από την γλώσσα σεναρίων JavaScript, το JSON είναι μια μορφοποίηση δεδομένων ανεξάρτητη από τη γλώσσα και είναι άμεσα διαθέσιμος ο κώδικας για την ανάλυση και την παραγωγή δεδομένων σε μορφή JSON, σε μια μεγάλη ποικιλία γλωσσών προγραμματισμού. Οι βασικοί τύποι του JSON είναι οι Number, String, Boolean, Array, Object, null

Παρακάτω φαίνεται ένα παράδειγμα αρχείου JSON:

```
{
        "IP": "109.231.185.88"
        "timeStamp": "17/01/2014:11:34:14"
        "averageResponseTimes": 0.234
        "numOfResponseTimes": 26
        "bytes": 1024
        "calling": "nginxDaemon"
}
```

5.4 XML

Η Extensible Markup Language (XML) είναι μια γλώσσα σήμανσης, που καθορίζει ένα σύνολο κανόνων για την κωδικοποίηση των εγγράφων σε μορφή που να είναι αναγνώσιμη τόσο από τον άνθρωπο, όσο και από τον υπολογιστή. Ορίζεται στο πρότυπο XML 1.0 κατά το W3C, και σε διάφορα άλλα συναφή πρότυπα, ελεύθερου κι ανοιχτού κώδικα.

Οι σχεδιαστικοί στόχοι της XML εστιάζουν στην απλότητα, τη γενικότητα και την ευχρηστία μέσω του Διαδικτύου. Πρόκειται για δεδομένα σε μορφή κειμένου, με ισχυρή υποστήριξη για όλες τις γλώσσες του κόσμου, μέσω των χαρακτήρων Unicode. Αν και ο σχεδιασμός της XML εστιάζει σε αρχεία, χρησιμοποιείται ευρέως για την αναπαράσταση αυθαίρετων δομών δεδομένων, για παράδειγμα στον τομέα των υπηρεσιών ιστού. Πολλές προγραμματιστικές διεπαφές εφαρμογών (APIs) έχουν αναπτυχθεί για να βοηθήσουν τους προγραμματιστές λογισμικού στην επεξεργασία XML δεδομένων, όπως και διάφορα συστήματα σχήματος για να βοηθήσουν στον ορισμό της XML-based γλωσσών.

Έχει χαρακτηριστεί ως επεκτάσιμη γλώσσα, διότι επιτρέπει στους χρήστες να καθορίσουν τα δικά τους στοιχεία. Πρωταρχικός σκοπός της είναι να διευκολύνει την ανταλλαγή δομημένων δεδομένων μεταξύ των διαφόρων συστημάτων πληροφοριών, ιδίως μέσω του Διαδικτύου. Χρησιμοποιείται τόσο για την κωδικοποίηση εγγράφων, όσο και για τη σειριοποίηση δεδομένων. Η XML είναι ένα πλαίσιο για τον ορισμό γλωσσών σήμανσης: δεν υπάρχει δεδομένη συλλογή ετικετών σήμανσης. Κάθε XML γλώσσα στοχεύει στο δικό της τομέα εφαρμογής, αλλά οι διάφορες γλώσσες μοιράζονται και πολλά κοινά χαρακτηριστικά.

Τα βασικά δομικά στοιχεία ενός εγγράφου XML είναι τα παρακάτω:

- Οι χαρακτήρες (characters): Εξ ορισμού, ένα έγγραφο XML είναι μια σειρά χαρακτήρων Unicode.
- Ο επεξεργαστής (XML processor-parser): Ο επεξεργαστής αναλύει την σήμανση και περνάει δομημένες πληροφορίες σε μία εφαρμογή. Η προδιαγραφή θέτει τις απαιτήσεις για το τι πρέπει να κάνει ο επεξεργαστής XML, αλλά η εφαρμογή είναι εκτός του της σκοπιάς του. Ο επεξεργαστής συχνά αναφέρεται κοινώς ως XML parser.
- Σήμανση και περιεχόμενο (markup and content): Οι χαρακτήρες που συνθέτουν ένα έγγραφο XML διαιρούνται σε χαρακτήρες σήμανσης και περιεχομένου, κάτι το οποίο διακρίνεται από απλούς συνταχτικούς κανόνες. Γενικά, οι συμβολοσειρές που αποτελούν τη σήμανση αρχίζουν με τον χαρακτήρα ‘‘<’’ και τελειώνουν με τον ‘‘>’’, ή αρχίζουν με τον χαρακτήρα ‘‘&’’ και τελειώνουν με τον ‘‘;’’. Οι συμβολοσειρές που δεν είναι σήμανση αποτελούν το περιεχόμενο.
- Ετικέτες (tags): Ένα στοιχείο σήμανσης ξεκινά με ‘‘<’’ και τελειώνει με ‘‘>’’. Οι ετικέτες είναι είτε ετικέτες έναρξης (<section>), είτε ετικέτες τέλους (</section>), είτε ετικέτες χωρίς στοιχείο (<line-break />).

- Στοιχεία (elements): Ένα συστατικό του εγγράφου, είτε ξεκινάει με μία ετικέτα έναρξης και τελειώνει με μία ετικέτα τέλους ή αποτελείται μόνο από μία ετικέτα χωρίς στοιχείο. Οι χαρακτήρες μεταξύ της ετικέτας έναρξης και λήξης, εάν υπάρχουν, είναι το περιεχόμενο του στοιχείου και μπορούν να περιέχουν σήμανση, συμπεριλαμβανομένων και άλλων στοιχείων (child elements). Παράδειγμα ενός τέτοιου στοιχείου είναι το "<Greeting>Hello, world.</Greeting>".

- Γνωρίσματα (attributes): Γνώρισμα είναι ένα στοιχείο σήμανσης που αποτελείται από ένα ζευγάρι ονόματος/τιμής που υπάρχει μέσα σε μία ετικέτα έναρξης ή σε μία ετικέτα χωρίς στοιχείο. Ένα γνώρισμα μπορεί να έχει μόνο μία τιμή και κάθε γνώρισμα μπορεί να εμφανίζεται το πολύ μία φορά σε κάθε στοιχείο.

Παρακάτω φαίνεται ένα παράδειγμα αρχείου XML:

```
<bp:deployment_artefact_section>
  <bp:deployment_artefact>
    <bp:artefact_id>WarrantManagement_app_war</bp:artefact_id>
    <bp:artefact_name>flipper.war</bp:artefact_name>
    <bp:artefact_type>war-file</bp:artefact_type>
    <!-- TODO: Check location-->
    <bp:artefact_location>https://svn.forge.morfeo-project.org/
      4caast/trunk/WP8/T8-3_VirtualPrivateCloud/RP2-Demo-Artefacts
      /WAR/tomcatAllInOne/flipper.war</bp:artefact_location>
    <bp:artefact_horizontalScaleable>true
    </bp:artefact_horizontalScaleable>
  </bp:deployment_artefact>
</bp:deployment_artefact_section>
```

5.5 Eclipse

Το λογισμικό Eclipse SDK είναι ένα πολυγλωσσικό ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού (IDE), που περιλαμβάνει ένα βασικό χώρο εργασίας κι ένα επεκτάσιμο σύστημα για την προσαρμογή του περιβάλλοντος. Γραμμένο υπό την άδεια Eclipse Public License, το περιβάλλον Eclipse είναι ελεύθερο και ανοικτού κώδικα λογισμικό. Είναι γραμμένο κυρίως σε Java και μπορεί να χρησιμοποιηθεί για την ανάπτυξη εφαρμογών τόσο σε Java αλλά και σε άλλες γλώσσες προγραμματισμού όπως Ada, C, C++, Haskell, JavaScript, Perl, PHP, Python, Ruby, Scala και Erlang, με τη βοήθεια διαφόρων plug-in.

Το αρχικό κομμάτι του κώδικα προήλθε από το IBM VisualAge. Το περιβάλλον ανάπτυξης του Eclipse (Eclipse Software Develepment Kit - SDK), το οποίο περιλαμβάνει τα εργαλεία ανάπτυξης της Java, προορίζεται κυρίως για προγραμματιστές Java. Οι χρήστες μπορούν να επεκτείνουν τις ικανότητές τουω με την εγκατάσταση plug-in που γράφτηκαν για την πλατφόρμα αυτή, αλλά κι εργαλείων ανάπτυξης για άλλες γλώσσες προγραμματισμού και μπορούν να γράψουν και να συνεισφέρουν τα δική τους plug-in modules.

Μέρος II

Υλοποίηση κι αποτελέσματα εργασίας

Κεφάλαιο 6

Περιγραφή προβλήματος

6.1 Ζητήματα στα υπάρχοντα συστήματα ελαστικότητας

Η ελαστικότητα είναι μια κεντρική έννοια στα περιβάλλοντα υπολογιστικού νέφους και μπορεί να εφαρμοστεί από τους παρόχους με διάφορους τρόπους και σε διαφορετικό βαθμό. Σήμερα, ένα από τα μεγαλύτερα εμπόδια όσον αφορά την ελαστικότητα στο νέφος είναι το γεγονός ότι οι τρέχουσες εφαρμογές δεν μπορούν να επωφεληθούν πλήρως από τις ελαστικές δυνατότητες της υποδομής.

Υπάρχουν δύο κύριες προσεγγίσεις ελαστικότητας. Η μία βασίζεται σε κανόνες οι οποίοι ορίζονται από τον πάροχο της πλατφόρμας. Η άλλη δεσμεύει τους πόρους στατικά, ώστε να δει πότε υπάρχουν μέγιστα και ελάχιστα στο φόρτο του συστήματος και εν συνεχεία δημιουργεί κανόνες οι οποίοι ενσωματώνονται στην πλατφόρμα και διαχειρίζονται την κλιμάκωση. Το βασικό μειονέκτημα αυτών των προσεγγίσεων είναι ότι εφαρμόζουν τις δυνατότητες κλιμάκωσης ανάλογα με το τι μπορεί να προσφέρει η πλατφόρμα και δεν λαμβάνουν υπόψη τις συγκεκριμένες ανάγκες της εφαρμογής.

Μια άλλη προσέγγιση είναι να στηριχθεί στο ίδιο το σύστημα κλιμάκωσης της πλατφόρμας, χρησιμοποιώντας τα γενικού σκοπού APIs που διατίθενται από τον πάροχο του νέφους. Το σύστημα αυτό μπορεί να βασίζεται σε μοντέλο πρόβλεψης φόρτου, σε κανόνες ή ακόμα και σε νευρωνικό δίκτυο. Ωστόσο, η κλιμάκωση έχει σχεδιαστεί για να καλύπτει τις συγκεκριμένες ανάγκες της πλατφόρμας, οι οποίες δεν είναι πάντα κατάλληλες για κάθε είδους εφαρμογή.

Ας αναλύσουμε τώρα τα προβλήματα που παρουσιάζονται στα συστήματα ελαστικότητας υπάρχοντων παρόχων υπολογιστικού νέφους:

- Το **Amazon EC2** επιτρέπει κάθετη κλιμάκωση ενός VM από τον πελάτη, ώστε να ανταποκριθεί στις ανάγκες τους για πόρους, κάτι που δεν γίνεται όμως αυτόματα. Για την επίτευξη οριζόντιας κατάτμησης απαιτείται η δημιουργία συμπλέγματος VMs και η ρύθμιση των παραμέτρων ανάλογα με τις ανάγκες του χρήστη. Ως εκ τούτου, μπορεί να παρέχεται μηχανισμός για εύκολη δέσμευση

των πόρων, αλλά δεν χρησιμοποιείται κάποιος αυτόματος μηχανισμός κλιμάκωσης, για τις εφαρμογές που τρέχουν πάνω στα VMs.

- Η **Google AppEngine** φιλοξενεί διάφορες υπηρεσίες και χειρίζεται την ανάπτυξη και την παρακολούθηση των διάφορων υπηρεσιών, κάνοντας χρήση του πυρήνα μηχανής της Google. Αυτό σημαίνει ότι οι κανόνες ελαστικότητας είναι εντελώς διαφανείς στους παρόχους των εφαρμογών. Έτσι, δεν δίνεται η δυνατότητα σε έναν χρήστη της πλατφόρμας να γράψει τους δικούς τους κανόνες κλιμάκωσης, ανάλογα με τις ειδικές απαιτήσεις της εφαρμογής του.

- Το **Windows Azure** της Microsoft όπως έχουμε αναφέρει σε προηγούμενη ενότητα αποτελείται από τρεις κύριες υπηρεσίες, μία εκ των οποίων είναι οι υπηρεσίες κατανεμημένης υποδομής με μία επαλήθευσης ταυτότητας του χρήστη. Αυτό βοηθά μια εφαρμογή να καταγράψει τις υπηρεσίες της σε ένα διάδρομο υπηρεσιών κι επιτρέποντας έτσι την πρόσβαση από άλλες εφαρμογές. Το Azure προσφέρει συγκεκριμένα VMs με προκαθορισμένους επεξεργαστικούς πυρήνες και μέγεθος μνήμης. Η αυτόματη κλιμάκωση προσφέρεται με τη βοήθεια κανόνων, μέσω ενός αρχείου ρυθμίσεων που ορίζεται από τους χρήστες. Έτσι, το παρακολουθεί τα VMs και κάθε φορά που υπάρχει ο φόρτος του συστήματος μειώνεται, αυτά μπορούν να αλλάζουν με μικρότερα, κλιμακώνοντας έτσι την εφαρμογή.

- Το **Cloud Foundry** είναι ένα ανοιχτό project PaaS που ξεκίνησε από VMware. Η κλιμάκωση υποστηρίζεται από τον αριθμό των VMs για μία εφαρμογή. Ο χρήστης μπορεί να καθορίσει πόσα VMs θα πρέπει να δημιουργηθούν αρχικά κι έχει επίσης το δικαίωμα να τροποποιήσει τον αριθμό αυτό κατά τη διάρκεια της εκτέλεσης. Οι πόροι που διατίθενται για για ένα VM δεν μπορούν να τροποποιηθούν κι αυτό αποδίδεται στο γεγονός ότι δεν περιλαμβάνονται κανόνες κλιμάκωσης και τιμολόγησης. Άρα, οι εφαρμογές δεν είναι αυτόματα ελαστικές.

- Η **RightScale** είναι μία πλατφόρμα διαχείρισης εφαρμογών για IaaS νέφη. Η αυτόματη κλιμάκωση υλοποιείται μέσω της παρακολούθησης του συστήματος. Αρχικά, η RightScale δίνει τη δυνατότητα παρακολούθησης των επιδόσεων και στη συνέχεια οι χρήστες μπορούν να καθορίσουν ειδοποιήσεις για κάθε μετρική που παρακολουθείται. Αυτές οι ειδοποιήσεις συνήθως δεν δίνουν το έναυσμα για μια δράση κλιμάκωση, αλλά υποστηρίζεται ένα σύστημα ψηφοφορίας όπου κάθε ‘‘οντότητα’’ ψηφίζει για οριζόντια κλιμάκωση. Κάθε μετρική απόδοσης μπορεί να στείλει ειδοποιήσεις, ανάλογα με τις ρυθμίσεις του χρήστη.

Σε όλα τα παραπάνω συστήματα, ένα κοινό πρόβλημα είναι ότι η κλιμάκωση δεν γίνεται με βάση τις ειδικές ανάγκες της εκάστοτε εφαρμογής που τρέχει στην πλατφόρμα, αλλά με γενικούς κανόνες της πλατφόρμας.

6.2 Η ελαστικότητα ως υπηρεσία (ElaaS: Elasticity as a Service)

Στη συνέχεια, παρουσιάζεται μία καινοτόμα προσέγγιση, η οποία παρέχει την ελαστικότητα σαν μία υπηρεσία, η οποία τρέχει πάνω σε μία πλατφόρμα, αλλά είναι γενικά ανεξάρτητη από αυτήν [1]. Ονομάζεται Elasticity-as-a-Service (ElaaS) επειδή η προσφερόμενη λειτουργικότητα παρέχεται ως υπηρεσία σε κάθε υποδομή νέφους ή πλατφόρμας, παρέχοντας τη δυνατότητα δυναμικής διαχείρισης σε όλα τα επίπεδα του νέφους.

6.2.1 Γενική ιδέα

Υπό ιδανικές συνθήκες, κάθε εφαρμογή που τρέχει σε μία πλατφόρμα υπολογιστικού νέφους θα πρέπει να κλιμακώνεται με δικούς της κανόνες. Αυτό είναι αρκετά δύσκολο βέβαια και η σύνθεση μιας στρατηγικής ελαστικότητας από διάφορους κανόνες κλιμάκωσης δεν είναι απαραίτητα η βέλτιστη για κάθε εφαρμογή. Για το λόγο αυτό, μερικές πλατφόρμες προσπαθούν να λύσουν αυτό το πρόβλημα, είτε με την ανάθεση της στρατηγικής της ελαστικότητας στον πελάτη ή με τον περιορισμό των πιθανών εφαρμογών που μπορούν να τρέξουν στο PaaS επίπεδο. Ωστόσο, σε μια τέτοια λύση η δυναμική κλιμάκωση μιας εφαρμογής παραμένει αόρατη στο χρήστη.

Για το λόγο αυτό, η ελαστικότητα είναι πιο κατάλληλο να υλοποιείται ως μια εφαρμογή SaaS με βαθιά γνώση για το περιβάλλον εκτέλεσης. Έτσι, μπορεί να είναι ένα βοηθητικό σύστημα που μπορεί παρέχεται ως υπηρεσία με πολλές εναλλακτικές λύσεις που προσφέρονται από διάφορους παρόχους, ανάλογα με την εκάστοτε εφαρμογή.

Αυτό το προτεινόμενο πλαίσιο μπορεί εύκολα να αναπτυχθεί σε οποιαδήποτε πλατφόρμα υπολογιστικού νέφους που μπορεί να αντιδρά δυναμικά στις απαιτήσεις του συστήματος ή του χρήστη. Μπορεί να θεωρηθεί ως μία εφαρμογή του νέφους, που μπορεί να χρησιμοποιηθεί από άλλες εφαρμογές για να παρέχει τη λειτουργικότητα της ελαστικότητας, αξιολογώντας τη λειτουργική κατάσταση της εφαρμογής και κάνοντας διορθωτικές ενέργειες, όταν αυτό απαιτείται σύμφωνα με τις σχετικές ρυθμίσεις. Πρόκειται για μια γενικευμένη λύση, επιτρέποντας σε οποιονδήποτε να κάνει χρήση των δυνατοτήτων της, προκειμένου να υλοποιήσει τη δική του μηχανή ελαστικότητας. Είναι επίσης κατανεμημένη, ενώ έχει και την ικανότητα να προσαρμόζεται στις ειδικές απαιτήσεις του κάθε χρήστη, εφαρμογής ή πλατφόρμας.

6.2.2 Βασικά χαρακτηριστικά

Τα κυριότερα χαρακτηριστικά αυτού του προτεινόμενου πλαισίου είναι τα εξής:

- Είναι εντελώς γενικευμένο και στηρίζεται τόσο σε αντιδραστικές (reactive) όσο και προβλεπτικές (predictive) προσεγγίσεις για το πως ενεργοποιούνται οι διορθωτικές κινήσεις ελαστικότητας. Παράλληλα, μπορεί να βασιστεί είτε σε κανόνες, είτε σε συστήματα μηχανικής μάθησης (machine-learning) ή νευρωνικά δίκτυα (neural networks) προκειμένου να κατευθύνεται η κλιμάκωση

της πλατφόρμας, επιτρέποντας στο χρήστη να επιλέξει τη λύση που ταιριάζει καλύτερα στις ανάγκες του.

- Είναι αποσυνδεδεμένο από το υπόλοιπο σύστημα κι ανεξάρτητο από την υποδομή του νέφους έτσι ώστε ο χρήστης να μπορεί να εστιάζει μόνο στη συγκεκριμένη εφαρμογή του, χωρίς να χρειάζεται να λάβει υπόψη τους περιορισμούς της πλατφόρμας. Επιπλέον, το πλαίσιο αυτό χρησιμοποιεί τις υπηρεσίες της πλατφόρμας, η οποία εξασφαλίζει την κατάλληλη εξισορρόπηση του φόρτου των εισερχόμενων αιτήσεων, των υπηρεσιών και των δεδομένων, προκειμένου να κλιμακωθεί αποτελεσματικά, χωρίς να περιορίζεται όμως σε μια συγκεκριμένη πλατφόρμα. Χρησιμοποιεί επίσης τα συστήματα παρακολούθησης της πλατφόρμας και τις υπηρεσίες τιμολόγησης, για να ελέγξει αν οι προτεινόμενες δράσεις είναι εφικτές και αποδεκτές σε σχέση με τους προκαθορισμένους όρους επιπέδου υπηρεσιών (SLAs).

- Η ElaaS σχεδιάζεται και υλοποιείται ως ένα σύνολο υπηρεσιών που έχουν αναπτυχθεί στο νέφος, και βασιζόμενη στην έννοια της πολυπελατιακότητας, ρυθμίζεται ανεξάρτητα για κάθε αίτηση ελαστικότητας από ένα χρήστη ή μια εφαρμογή.

- Η ElaaS μπορεί να είναι μία υπηρεσία που εισάγεται σε μία συγκεκριμένη πλατφόρμα όπου τρέχει ένα SaaS. Για το λόγο αυτό μπορεί να τρέξει σε μία άλλη πλατφόρμα ή σε ένα ιδιωτικό νέφος, αλλά να πετυχαίνει το στόχο της με τη χρήση γενικευμένων διεπαφών της πλατφόρμας.

6.3 Πλατφόρμα διαχείρισης της κλιμάκωσης

Για την ανάπτυξη ενός συστήματος που μπορεί να εκμεταλλεύεται την παραπάνω προσέγγιση, κατασκευάσαμε μία ολοκληρωμένη πλατφόρμα υπολογιστικού νέφους, PaaS, στην οποία μπορούν να εγκατασταθούν διάφορες εφαρμογές SaaS. Η πλατφόρμα μπορεί να διαχειρίζεται τους πόρους της υποδομής IaaS, να επιβλέπει τις εφαρμογές που εκτελούνται και να παρακολουθεί τη λειτουργία του σύστηματος και βασίζεται στη λογική της ελαστικότητας-ως-υπηρεσία (ElaaS), που αναλύθηκε στην προηγούμενη ενότητα.

Το κύριο μέρος της πλατφόρμας είναι η μηχανή κλιμάκωσης (Scalability Engine). Ο ρόλος της είναι να κλιμακώνει τους χρησιμοποιούμενους πόρους του νέφους υποδομής (IaaS), που στην περίπτωσή μας είναι ο εμπορικός πάροχος Flexiscale, ανάλογα με τους απαιτούμενους πόρους για την επιθυμητή λειτουργία του συστήματος, που υπολογίζει ο Resource Requirement Calculator. Ο υπολογισμός αυτός γίνεται με βάση κανόνων που λαμβάνουν υπ' όψη τους πραγματικούς χρόνους απόκρισης, τον αριθμό των πελατών που χρησιμοποιούν την εφαρμογή, τον επιθυμητό χρόνο απόκρισης και το μέγιστο αριθμό χρηστών.

Στην πλατφόρμα υπάρχει επίσης μία βάση δεδομένων η οποία περιέχει τα στοιχεία κάθε εφαρμογής που τρέχει. Τέτοια στοιχεία είναι ο κωδικός της κάθε εφαρμογής, η διεύθυνση ip του proxy server της, οι διευθύνσεις IP των εικονικών μηχανών στις οποίες τρέχει η εφαρμογή, καθώς και το μέγεθός τους. Επίσης, περιλαμβάνει για κάθε εφαρμογή τους μέσους χρόνους απόκρισης της υπηρεσίας, το μέγεθος των δεδομένων και το πλήθος των αιτημάτων, οργανωμένα ανά δέκα δευτερόλεπτα και για τα τελευταία 300 λεπτά που τρέχει.

Μία σημαντική μονάδα που επίσης χρησιμοποιείται από την πλατφόρμα, είναι ένας οδηγός (driver) για την επικοινωνία της με τον πάροχο της υποδομής (IaaS), δηλαδή την υποδομή Flexiscale. Ο οδηγός έχει σκοπό την εύκολη δέσμευση κι αποδέσμευση πόρων μέσω κώδικα γραμμένο σε γλώσσα java, χρησιμοποιώντας την προγραμματιστική διεπαφή (API) της Flexiscale. Με τη βοήθεια αυτής της διεπαφής, επικοινωνεί με τους πόρους της Flexiscale και στέλνει μυνήματα μέσω του πρωτοκόλλου SOAP, δεσμεύοντας κι αποδεσμεύοντάς τους.

Μία ακόμα σημαντική μονάδα είναι ο διακομιστής μεσολάβησης (proxy server). Κατά την εκκίνησή κάθε εφαρμογής που εγκαθίσταται στην πλατφόρμα, δημιουργείται ένας τέτοιος διακομιστής, ο οποίος διανέμει την κίνηση ισάξια στις εικονικές μηχανές που δημιουργούνται στην υποδομή Flexiscale. Σε κάθε τέτοιον διακομιστή τρέχει ένας δαίμονας (daemon), ο οποίος αναλαμβάνει να στέλνει ανά τακτά χρονικά διαστήματα στην πλατφόρμα (και να αποθηκεύει στη βάση δεδομένων της), με τη βοήθεια του πρωτοκόλλου REST, ορισμένα σημαντικά δεδομένα, όπως το χρόνο απόκρισης και τον αριθμό χρηστών της εκάστοτε εφαρμογής. Όπως προαναφέραμε, ο διακομιστής μεσολάβησης δεν είναι μοναδικός, αλλά δημιουργείται ένας για κάθε εφαρμογή που στήνεται στο νέφος. Το αρχείο ρυθμίσεων που περιλαμβάνει τις IP διευθύνσεις των εικονικών μηχανών στις οποίες τρέχει η εφαρμογή ανανεώνεται από τον οδηγό της πλατφόρμας, ο οποίος στέλνει στον proxy τις διευθύνσεις με τη βοήθεια του πρωτοκόλλου REST.

Παράλληλα, η κλιμάκωση οδηγείται από τον χρήστη, με τη βοήθεια μίας αλληλεπιδραστικής εφαρμογής (graphical user interface), που αναλαμβάνει την εγκατάσταση και την απεγκατάστασή των εφαρμογών. Μέσα από αυτήν, ο πάροχος μίας εφαρμογής ορίζει τις απαιτήσεις σε χρόνο απόκρισης και τον μέγιστο αριθμό των χρηστών στον οποίο θα πρέπει να ανταποκρίνεται η υπηρεσία του. Ουσιαστικά, ο χρήστης τροποποιεί ένα αρχείο xml, που περιέχει τους κανόνες κλιμάκωσης για τη συγκεκριμένη εφαρμογή. Αυτοί οι κανόνες αναλύονται κι εφαρμόζονται από την ίδια την πλατφόρμα και προφανώς είναι διαφορετικοί για κάθε εφαρμογή. Το αρχείο αυτό μετέπειτα το επεξεργάζεται ο Resource Requirement Calculator για τον υπολογισμό των απαιτούμενων πόρων.

Κεφάλαιο 7

Περιγραφή υλοποίησης συστήματος

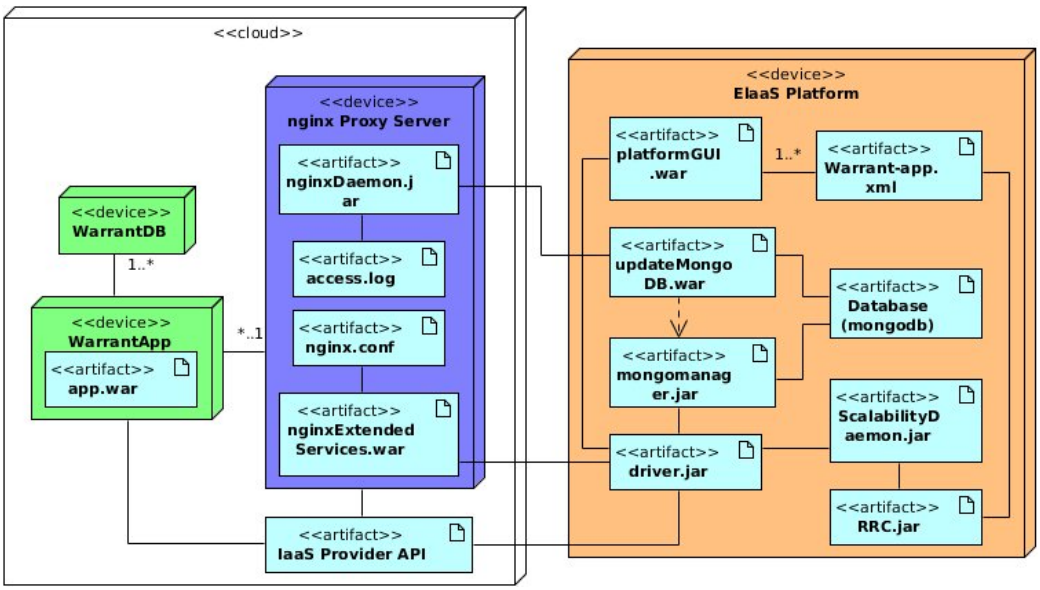
7.1 Προδιαγραφές σχεδίασης συστήματος

Για την επιτυχή υλοποίηση του συστήματος, πάρθηκαν κάποιες σημαντικές σχεδιαστικές αποφάσεις. Αυτές αφορούν τις επιμέρους μονάδες της πλατφόρμας, τον τρόπο επικοινωνίας κι αλληλεπίδρασης μεταξύ τους, το σχεδιασμό του τελικού συστήματος, τη βάση δεδομένων και τα αρχεία που ήταν απαραίτητα για τη ρύθμιση του συστήματος.

Η κύρια γλώσσα προγραμματισμού που χρησιμοποιήθηκε είναι η Java, ενώ το γραφικό περιβάλλον της υπηρεσίας είναι κατασκευασμένο με τη βοήθεια JSP και Servlets. Τα services που υλοποιήθηκαν τρέχουν πάνω σε έναν Apache Tomcat Server. Το αρχείο που περιέχει τις πληροφορίες και τους κανόνες κλιμάκωσης της κάθε εφαρμογής και χρησιμοποιείται από την πλατφόρμα για την κλιμάκωση είναι γραμμένο σε XML, ενώ η επεξεργασία του έγινε με τη βοήθεια της αρχιτεκτονικής Jaxb. Σαν proxy server που διαμοιράζει τον φόρτο στα επιμέρους μηχανήματα επιλέχθηκε ο nginx proxy server, ενώ για την επικοινωνία τους με την πλατφόρμα, χρησιμοποιήθηκε το μοντέλο REST. Όλες οι επιμέρους μονάδες και βιβλιοθήκες υλοποιήθηκαν στο προγραμματιστικό περιβάλλον Eclipse.

Η αλληλεπίδραση μεταξύ των μονάδων που υλοποιήθηκαν, αναλύεται πιο λεπτομερώς στη συνέχεια, με τη βοήθεια διαγραμμάτων.

7.1.1 Διάγραμμα υλοποίησης



Σχήμα 7.1: Διάγραμμα υλοποίησης

Παραπάνω φαίνεται το διάγραμμα υλοποίησης του συστήματος. Σε αυτό γίνονται φανερά τα εξής:

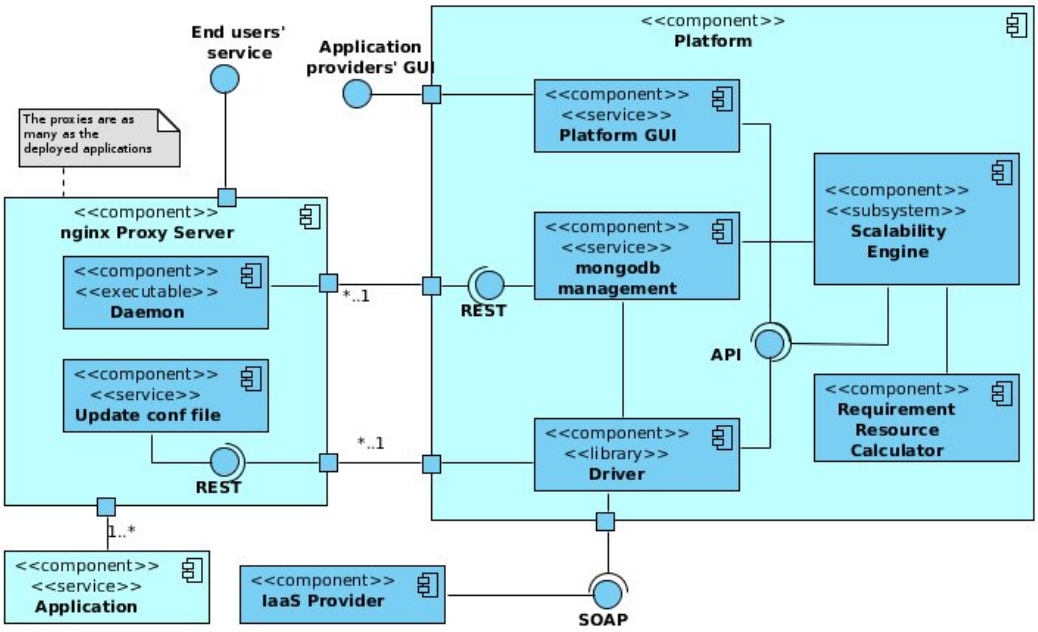
- Η σημαντικότερη μονάδα της πλατφόρμας είναι ο ScalabilityDaemon, ένα εκτελέσιμο .jar αρχείο. Χρησιμοποιεί τη μονάδα RRC.jar, που υπολογίζει τους απαιτούμενους πόρους της εφαρμογής ανάλογα με το φόρτο, με βάση τους κανόνες κλιμάκωσης που υπάρχουν στο αρχείο Warrant-app.xml.

- Μία ακόμα σημαντική μονάδα είναι το γραφικό περιβάλλον platformGUI.war για την εγκατάσταση κι απεγκατάσταση εφαρμογών στην πλατφόρμα, όπως και για την εισαγωγή νέων κανόνων κλιμάκωσης. Τρέχει πάνω σε έναν Tomcat Server και ουσιαστικά επεξεργάζεται το αρχείο Warrant-app.xml.

- Για τη βάση δεδομένων mongoDB έχει δημιουργηθεί ένα service, το updateMongoDB.war, το οποίο χρησιμοποιούν οι proxy servers για να στείλουν τους χρόνους απόκρισης και τον αριθμό των χρηστών μίας εφαρμογής στην πλατφόρμα και να τα αποθηκεύσουν στη βάση δεδομένων. Επίσης, έχει αναπτυχθεί μία βιβλιοθήκη, η mongomanager.jar, η οποία παρέχει συναρτήσεις για εγγραφή κι επεξεργασία δεδομένων της βάσης.

- Κατασκευάστηκε επίσης ο driver.jar, που στην ουσία είναι μία βιβλιοθήκη για την επεξεργασία των πόρων υποδομής και πιο συγκεκριμένα για deploy, undeploy, scale-up και scale-down.

7.1.2 Ψηφιδικό διάγραμμα



Σχήμα 7.2: Ψηφιδικό διάγραμμα

Παραπάνω φαίνεται το ψηφιδικό διάγραμμα του συστήματος. Σε αυτό φαίνονται τα εξής:

- Η Flexiscale παρέχει ένα SOAP API με τη βοήθεια του οποίου μπορεί κανείς να χρησιμοποιήσει τις δυνατότητες της εφαρμογής. Έτσι, ο driver της πλατφόρμας μπορεί μέσω αυτού του API να κάνει deploy, undeploy, scale-up και scale-down στους πόρους που χρησιμοποιεί η πλατφόρμα.

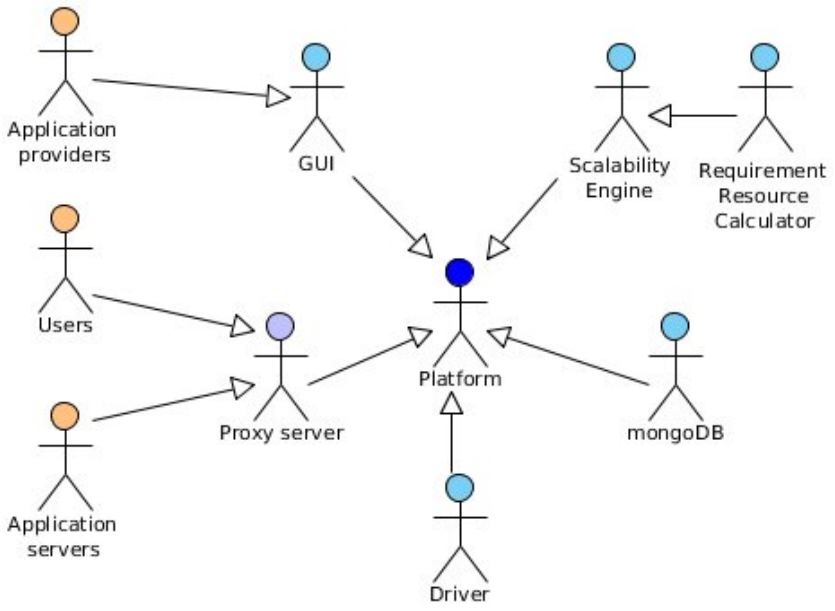
- Στον proxy server τρέχει ένα REST service, το οποίο αναβαθμίζει το αρχείο ρυθμίσεων nginx.conf του nginx και στο οποίο γράφει τις καινούριες διευθύνσεις IP στις οποίες θα μοιράζει τον φόρτο ο proxy server. Έτσι, κάθε φορά που σηκώνεται ή κλείνει ένα μηχάνημα στο οποίο τρέχει η εφαρμογή, ο driver στέλνει στον proxy τις νέες ενεργές διευθύνσεις, μέσω αυτού του REST service.

- Στην πλατφόρμα τρέχει ένα REST service, το οποίο προσθέτει δεδομένα στη βάση δεδομένων. Συγκεκριμένα, με τη βοήθεια αυτού του service, ο proxy server στέλνει ανά τακτά χρονικά διαστήματα τους χρόνους απόκρισης της εφαρμογής στους χρήστες. Τα δεδομένα αυτά επεξεργάζεται μετέπειτα ο Resource Requirement Calculator για τον νέο υπολογισμό των απαιτούμενων πόρων για την εκάστοτε εφαρμογή.

7.2 Σενάρια χρήσης - Ακολουθιακά διαγράμματα

Στη συνέχεια θα παρουσιάσουμε την ακριβή λειτουργία του συστήματος με τη βοήθεια σεναρίων χρήσης, ακολουθιακών διαγραμμάτων που παρουσιάζουν λεπτομερώς τις ενέργειες που πραγματοποιούνται κι ενός διαγράμματος χρήσης για τους χρήστες της πλατφόρμας και των εφαρμογών που τρέχουν σε αυτή.

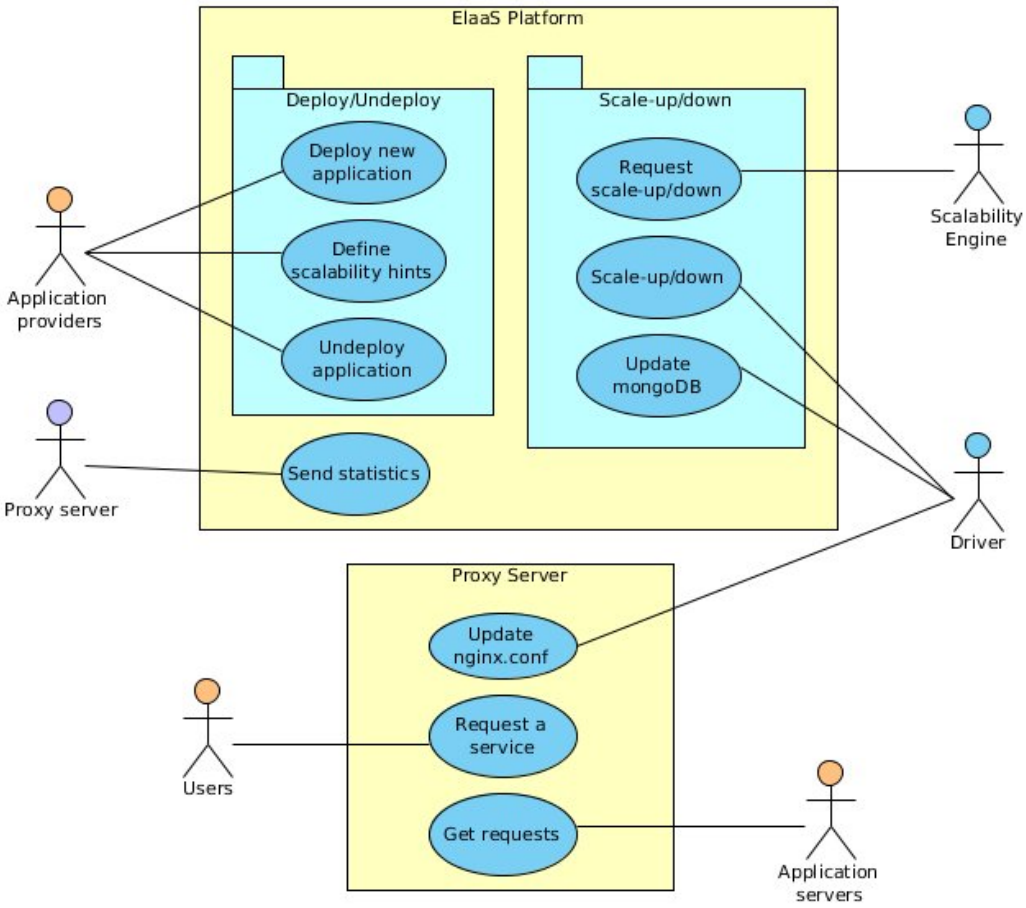
7.2.1 Διάγραμμα δραστών-χρήσης



Σχήμα 7.3: Διάγραμμα δραστών

Παραπάνω φαίνεται το διάγραμμα δραστών. Από αυτό συμπεραίνουμε ότι:

- Η πλατφόρμα αποτελείται από τα υποσυστήματα GUI, Scalability Engine, mongoDB και Driver, με τα οποία αλληλεπιδρά. Επίσης, επικοινωνεί με τον Proxy Server.
- Το υποσύστημα Scalability Engine επικοινωνεί με τον Requirement Resource Calculator.
- Οι πάροχοι χρησιμοποιούν το GUI για τη χρήση των υπηρεσιών της πλατφόρμας. Οι χρήστες των υπηρεσιών στέλνουν αιτήματα στον Proxy Server, ο οποίος τα προωθεί στους Application Servers.



Σχήμα 7.4: Διάγραμμα χρήσης

Στο παραπάνω διάγραμμα βλέπουμε ότι:

- Οι πάροχοι των εφαρμογών μπορούν:

  1) να εγκαταστήσουν μία καινούρια εφαρμογή.
  2) να ορίσουν κανόνες κλιμάκωσης για μία υπάρχουσα εφαρμογή.
  3) να απεγκαταστήσουν μία υπάρχουσα εφαρμογή.

- Οι χρήστες των εφαρμογών μπορούν να ζητήσουν μία υπηρεσία. Ο proxy server προωθεί αυτό το αίτημα στους Application servers.

- Η Scalability engine αιτείται δέσμευση νέων πόρων ή αποδέσμευση των υπάρχοντων, ανάλογα με το φόρτο του συστήματος.

- Ο Driver πραγματοποιεί τη δέσμευση κι αποδέσμευση πόρων. Παράλληλα, ανανεώνει τη βάση δεδομένων και το αρχείο ρυθμίσεων του Proxy server.

7.2.2 Γενική λειτουργία

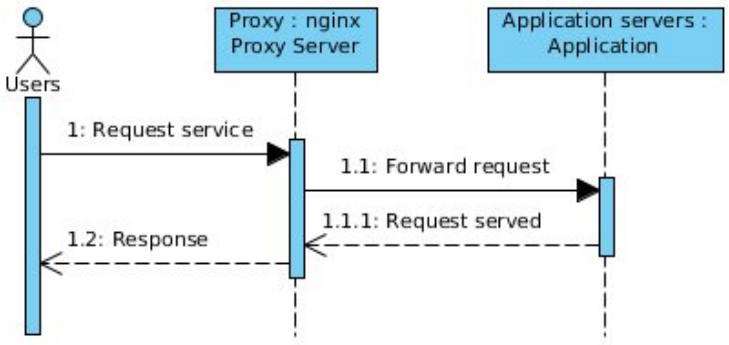
Το σενάριο λειτουργίας της εφαρμογής που παρέχεται στον τελικό χρήστη είναι το εξής:

1) Ο χρήστης στέλνει αίτημα για τη χρήση μίας υπηρεσίας.

2) Ο proxy server προωθεί το αίτημα αυτό σε έναν από τους επιμέρους εξυπηρετητές (application servers), που διατίθενται για αυτή την υπηρεσία.

3) Ο εξυπηρετητής αυτός απαντάει στον proxy server σχετικά με το αίτημα του χρήστη.

4) Ο proxy server στέλνει το αποτέλεσμα στο χρήστη και κρατάει τα απαραίτητα στατιστικά, τα οποία αποθηκεύει στο αρχείο access.log.



Σχήμα 7.5: Ακολουθιακό διάγραμμα γενικής λειτουργίας της υπηρεσίας

Το σενάριο λειτουργίας της πλατφόρμας είναι το εξής:

1) Η πλατφόρμα ζητάει τις εγκατεστημένες εφαρμογές από τη βάση δεδομένων.

2) Για κάθε εφαρμογή παίρνει τους τελευταίους χρόνους απόκρισης και τους χρήστες από τη βάση δεδομένων.

3) Υπολογίζει τους μέσους χρόνους απόκρισης της εφαρμογής και τους μέσους χρήστες.

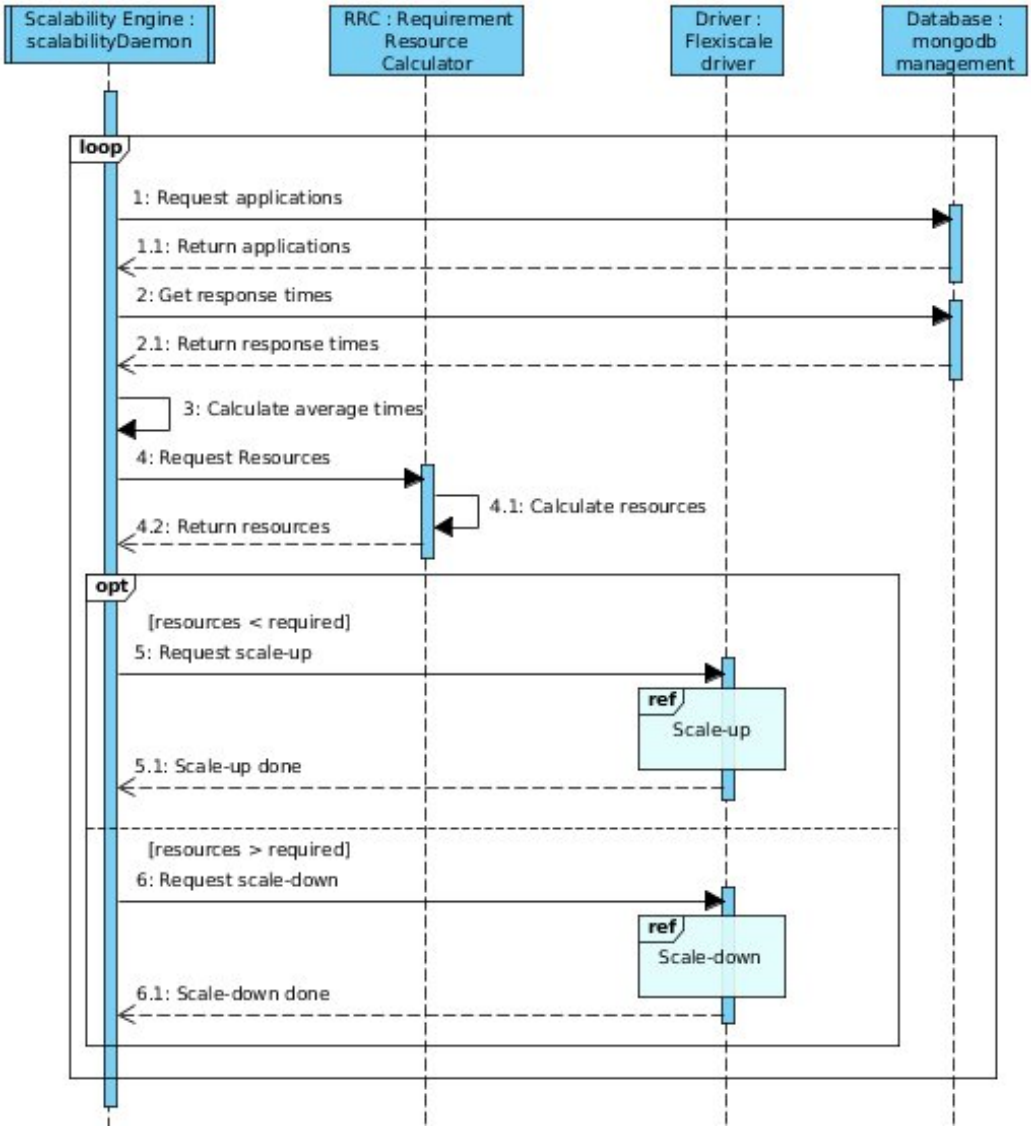
4) Αιτείται από τον Requirement Resource Calculator τους απαιτούμενους πόρους για αυτούς τους χρόνους απόκρισης και αυτό τον αριθμό χρηστών.

5) Ο RRC επιστρέφει τους απαιτούμενους πόρους.

6) Αν οι απαιτούμενοι πόροι είναι λιγότεροι από αυτούς που χρησιμοποιούνται τώρα, τότε ζητείται από τον Driver να προχωρήσει σε δέσμευση νέων πόρων.

7) Αν είναι περισσότεροι, τότε ζητείται από τον Driver να προχωρήσει σε αποδέσμευση πόρων.

8) Η διαδικασία επαναλαμβάνεται επ΄ άπειρον, κάθε δέκα δευτερόλεπτα.



Σχήμα 7.6: Ακολουθιακό διάγραμμα γενικής λειτουργίας της πλατφόρμας

7.2.3 Deploy

Όταν ένας πάροχος θέλει να εγκαταστήσει στο νέφος μία νέα εφαρμογή, το σενάριο λειτουργίας είναι το εξής:

1) Ο πάροχος κάνει αίτημα για εγκατάσταση νέας εφαρμογής.

2) Η πλατφόρμα δημιουργεί ένα νέο αρχείο Warrant-app.xml για τη νέα εφαρμογή και της δίνει το προσδιοριστικό id της.

3) Η πλατφόρμα ζητάει από τον driver τη δημιουργία των αναγκαίων VMs.

4) Ο driver ζητάει τη δημιουργία ενός VM που τρέχει την εφαρμογή κι ενός proxy.

5) Ο driver ενημερώνει τη βάση δεδομένων με τη νέα εφαρμογή.

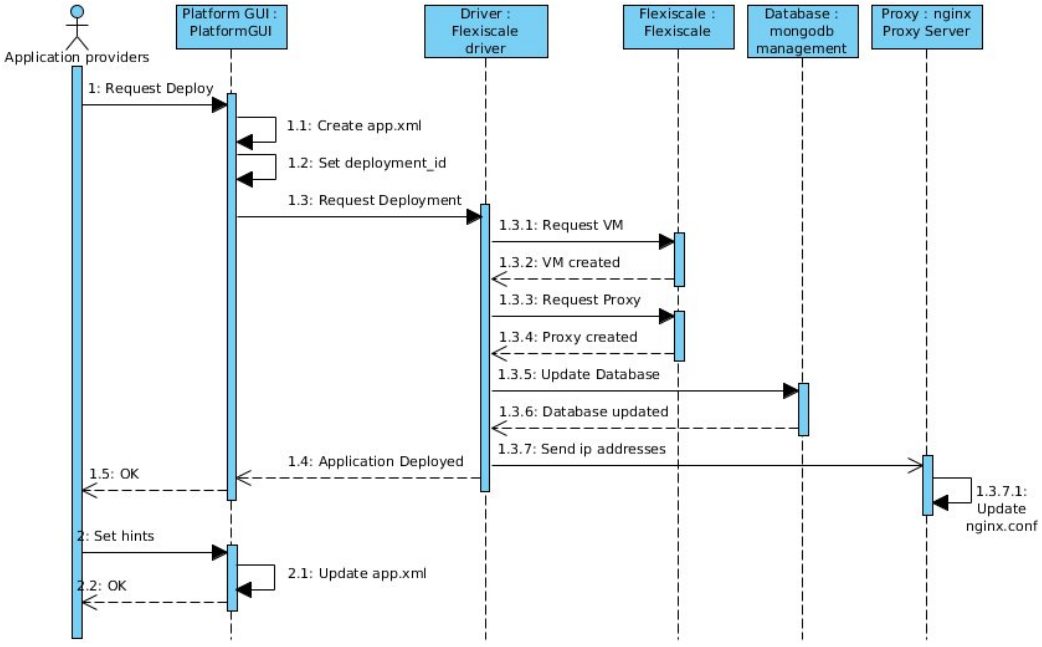
6) Στη συνέχεια στέλνει την IP address του VM στον proxy server.

7) Η εφαρμογή έχει πλέον εγκατασταθεί στο νέφος.

Κατά την εισαγωγή κανόνων κλιμάκωσης για μία εφαρμογή, συμβαίνουν τα εξής:

1) Ο πάροχος κάνει αίτημα για προσθήκη νέων κανόνων.

2) Η πλατφόρμα προσθέτει τους νέους κανόνες, τροποποιώντας το Warrant-app.xml.



Σχήμα 7.7: Ακολουθιακό διάγραμμα για το deploy

7.2.4 Undeploy

Όταν ένας πάροχος θέλει να απεγκαταστήσει μία υπάρχουσα εφαρμογή, το σενάριο λειτουργίας είναι το εξής:

1) Ο πάροχος κάνει αίτημα για απεγκατάσταση νέας εφαρμογής.

2) Η πλατφόρμα βρίσκει το id της εφαρμογής, δεδομένουν του ονόματος που έβαλε ο πάροχος.

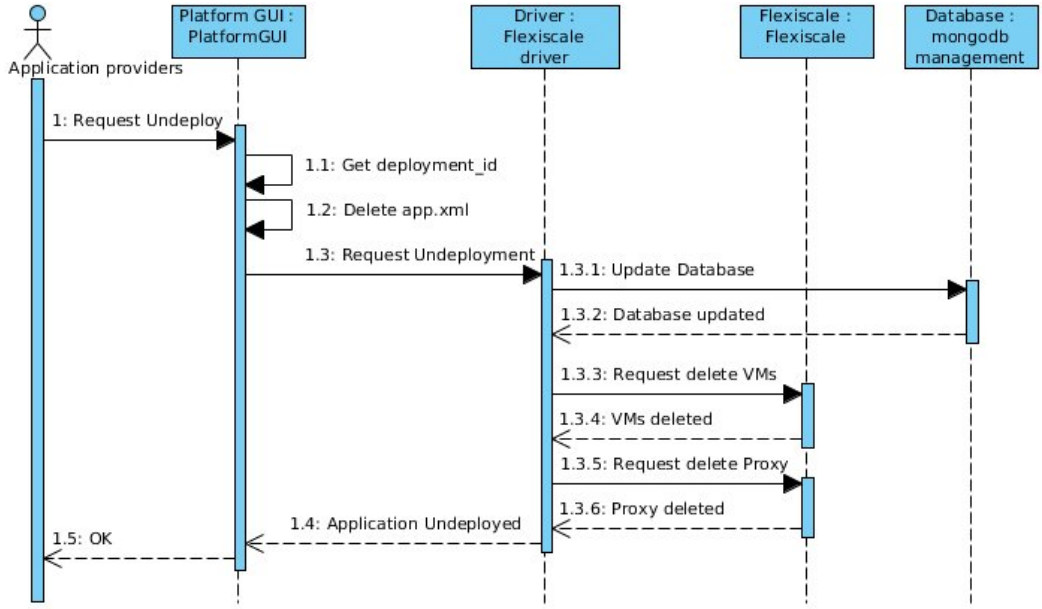
3) Η πλατφόρμα διαγράφει το αρχείο Warrant-app.xml της εφαρμογής.

4) Η πλατφόρμα ζητάει από τον driver τη διαγραφή των VMs.

5) Ο driver διαγράφει την εφαρμογή από τη βάση δεδομένων.

6) Ο driver ζητάει τη διαγραφή των VMs που τρέχουν την εφαρμογή, καθώς και του proxy server.

7) Η εφαρμογή έχει πλέον απεγκατασταθεί από το νέφος.



Σχήμα 7.8: Ακολουθιακό διάγραμμα για το undeploy

7.2.5 Scale-up

Όταν η πλατφόρμα αιτείται από τον Driver τη δέσμευση νέων πόρων, το σενάριο λειτουργίας είναι το εξής:

1) Ο driver παίρνει τη λίστα με τις διευθύνσεις IP των VMs που εξυπηρετούν τώρα την εφαρμογή.

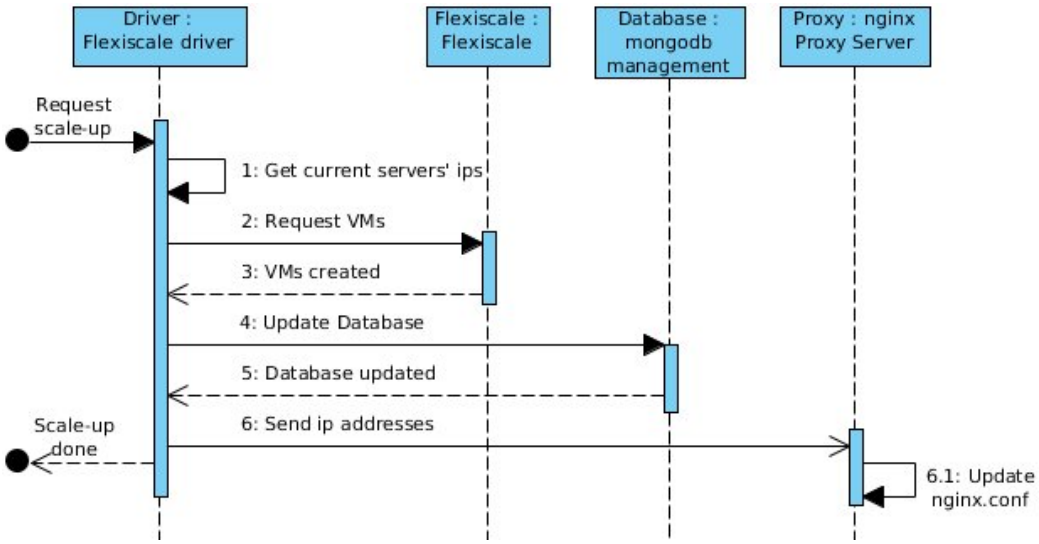
2) Αιτείται τη δημιουργία νέων VMs.

3) Ενημερώνει τη βάση δεδομένων με όλα τα VMs που εξυπηρετούν πλέον την εφαρμογή.

4) Προσθέτει στην υπάρχουσα λίστα τις νέες διευθύνσεις IP και στέλνει τη νέα λίστα στον proxy server.

5) Ο proxy server ενημερώνει το αρχείο ρυθμίσεων nginx.conf.

6) Η δέσμευση νέων πόρων έχει ολοκληρωθεί.



Σχήμα 7.9: Ακολουθιακό διάγραμμα για το scale-up

7.2.6 Scale-down

Όταν η πλατφόρμα αιτείται από τον Driver την αποδέσμευση πόρων, το σενάριο λειτουργίας είναι το εξής:

1) Ο driver παίρνει τη λίστα με τις διευθύνσεις IP των VMs που εξυπηρετούν τώρα την εφαρμογή.

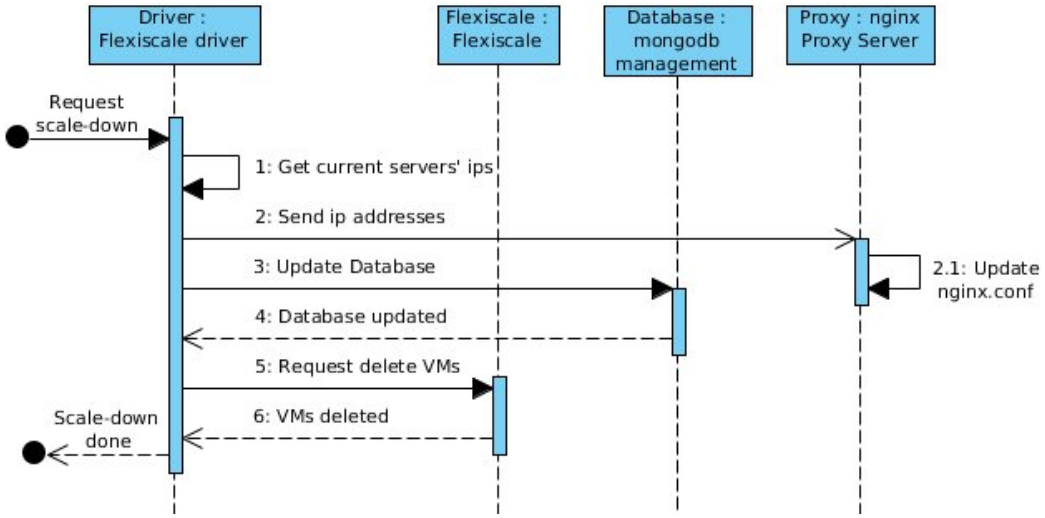
2) Αφαιρεί από την υπάρχουσα λίστα τις διευθύνσεις IP των μηχανημάτων που θα αφαιρεθούν και στέλνει τη νέα λίστα στον proxy server.

3) Ο proxy server ενημερώνει το αρχείο ρυθμίσεων nginx.conf.

4) Ο driver ενημερώνει τη βάση δεδομένων με τα VMs που εξυπηρετούν πλέον την εφαρμογή.

5) Ο driver αιτείται τη διαγραφή των απαιτούμενων VMs.

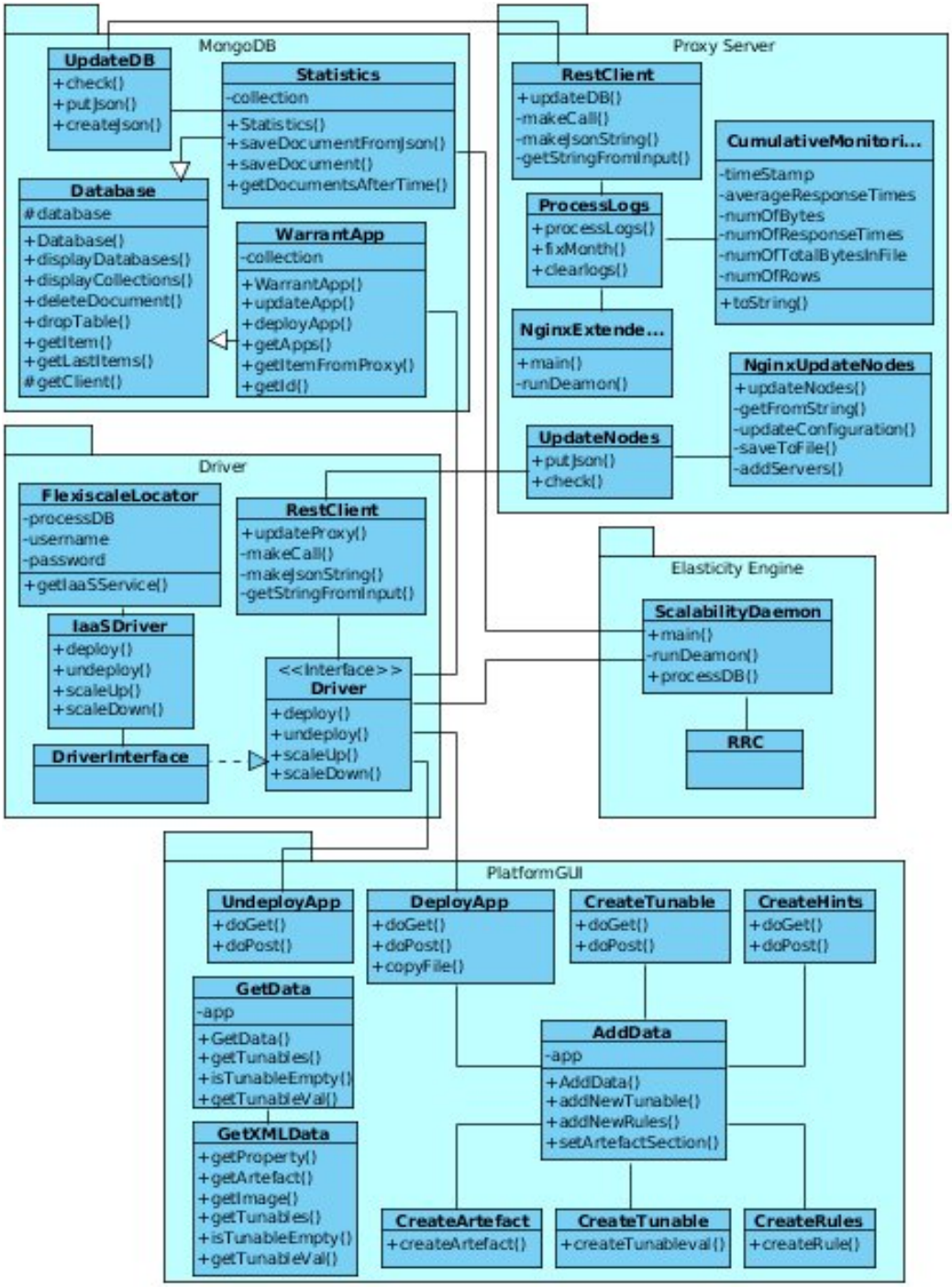
6) Η αποδέσμευση πόρων έχει ολοκληρωθεί.



Σχήμα 7.10: Ακολουθιακό διάγραμμα για το scale-down

7.3 Διάγραμμα κλάσεων

Στη συνέχεια παρουσιάζουμε το διάγραμμα κλάσεων του συστήματος, με σκοπό την καλύτερη κατανόηση του κώδικα που γράφτηκε. Στο διάγραμμα αυτό φαίνονται οι διάφορες υπομονάδες του συστήματος, με τις κυριότερες κλάσεις που υλοποιήθηκαν. Έπειτα γίνεται μία ανάλυση των κλάσεων της κάθε υπομονάδας και αναλύονται οι λειτουργίες καθεμιάς.



Σχήμα 7.11: Διάγραμμα κλάσεων

7.3.1 Βάση δεδομένων (Database)

Κλάση Database

Περιγραφή	Μία γενική κλάση με μεθόδους για τη διαχείριση, προσθήκη, διαγραφή και τροποποίηση εγγραφών στη βάση δεδομένων. Περιέχει επίσης μεθόδους για την άντληση στοιχείων από τη βάση δεδομένων.
Γνωρίσματα	database
Λειτουργίες	*-Database(database)*: Ο constructor της κλάσης. *-displayDatabases()*: Επιστρέφει όλες τις βάσεις δεδομένων. *-displayCollections()*: Επιστρέφει όλες τις συλλογές της βάσης. *-deleteDocument(key, key_value, collection)*: Διαγράφει ένα αρχείο από μία συλλογή της βάσης. *-dropTable(collection)*: Διαγράφει μία συλλογή της βάσης δεδομένων. *-getItem(key, key_value, item, collection)*: Επιστρέφει το πεδίο ενός αρχείου, μιας συλλογής της βάσης, ανάλογα με ένα πεδίο-κλειδί. *-getLastItems(number, item, sort, collection)*: Επιστρέφει ένα πεδίο των τελευταίων αρχείων, μιας συλλογής της βάσης, ταξινομημένα σύμφωνα με ένα συγκεκριμένο πεδίο. Αν ο αριθμός number είναι μηδέν, τότε επιστρέφει ένα πεδίο όλων των αρχείων της συλλογής. *-getClient()*: Δημιουργεί την επικοινωνία με τη βάση δεδομένων.

Κλάση Statistics

Περιγραφή	Επεκτείνει την κλάση Database, για τη διαχείριση των συλλογών με τα στατιστικά στοιχεία των proxy servers, όπως τους χρόνους απόκρισης, τον αριθμό των χρηστών, το μέγεθος των δεδομένων, κτλ, για κάθε εφαρμογή που τρέχει στην πλατφόρμα.
Γνωρίσματα	*collection*
Λειτουργίες	*-Statistics(database, collection)*: Ο κατασκευαστής της κλάσης. *-saveDocumentFromJson(json)*: Αποθηκεύει στη συλλογή της βάσης δεδομένων ένα έγγραφο, τα δεδομένα του οποίου δίνονται στη μορφή json αρχείου. *-saveDocument(calling, ip, timeStamp, numOfResponseTimes, numOfBytes, averageResponseTimes)*: Αποθηκεύει στη συλλογή της βάσης δεδομένων ένα αρχείο, με τα πεδία που δίνονται. *-getDocumentsAfterTimestamp(timeStamp)*: Επιστρέφει όλα τα αρχεία που δημιουργήθηκαν μετά από μία χρονική στιγμή. *-getLastItems(number, item)*: Επιστρέφει το πεδίο των πιο πρόσφατων αρχείων της συλλογής. *-deleteDocument(key_value)*: Διαγράφει ένα αρχείο της συλλογής. *-dropTable()*: Διαγράφει τη συλλογή.

Κλάση WarrantApp

Περιγραφή	Επεκτείνει την κλάση Database, για τη διαχείριση μίας συλλογής με τα στοιχεία των εφαρμογών που είναι εγκατεστημένες στην πλατφόρμα, όπως τον proxy server τους, της διευθύνσεις IP που τρέχουν την εφαρμογή, τις δυνατότητες και το πλήθος των υπολογιστικών πόρων, κτλ.
Γνωρίσματα	*collection*
Λειτουργίες	*-WarrantApp(database, collection)*: Ο constructor της κλάσης. *-updateApp(deployment_id, instances, cpu, ram, ips, scalePoint)*: Τροποποιεί το αρχείο της βάσης που αφορά μια εφαρμογή που τρέχει στην πλατφόρμα. *-deployApp(name, deployment_id, proxyIp, min_cpu, min_ram, image, ips)*: Δημιουργεί ένα αρχείο για μία νέα εφαρμογή που εγκαθίσταται στην πλατφόρμα. *-getApps()*: Επιστρέφει όλες τις εφαρμογές που τρέχουν. *-getLastItems(number, item)*: Επιστρέφει το πεδίο των τελευταίων αρχείων της συλλογής, ταξινομημένων με βάση το id. *-getItem(key, item)*: Επιστρέφει το πεδίο μίας εφαρμογής, με βάση το id της. *-getItemFromProxy(key, item)*: Επιστρέφει το πεδίο μίας εφαρμογής, με βάση τον proxy server της. *-getId(key)*: Επιστρέφει το id μίας εφαρμογής, με βάση το ονομά της. *-deleteDocument(key_value)*: Διαγράφει ένα αρχείο της συλλογής. *-dropTable()*: Διαγράφει τη συλλογή.

Κλάση UpdateDB

Περιγραφή	Μία κλάση που υλοποιεί ένα REST service, το οποίο προσθέτει στη βάση δεδομένων τους χρόνους απόκρισης μίας εφαρμογής που είναι εγκατεστημένη στην πλατφόρμα.
Γνωρίσματα	*context*
Λειτουργίες	*-check()*: Ελέγχει αν λειτουργεί το REST service. *-putJson(ip, content)*: Εισάγει στην κατάλληλη συλλογή στη βάση δεδομένων τα στατιστικά στοιχεία της εφαρμογής, τα οποία έρχονται από τον proxy server με τη μορφή json αρχείου. *-createJson(result)*: Δημιουργεί ένα αρχείο json που αναφέρει αν η υπηρεσία κλήθηκε επιτυχώς ή όχι.

7.3.2 Οδηγός (Driver)

Κλάση FlexiscaleServiceCustomLocator

Περιγραφή	Δημιουργεί την επικοινωνία με την υποδομή της Flexiscale.
Γνωρίσματα	*serviceURL, username, password*
Λειτουργίες	*-getFlexiscaleService*: Δημιουργεί τη σύνδεση με τη Flexiscale.

Κλάση FlexiantDriver

Περιγραφή	Υλοποιεί το interface "DriverInterface" κι αποτελεί μια βιβλιοθήκη με μεθόδους για τη διαχείριση των πόρων της υποδομής Flexiscale.
Γνωρίσματα	
Λειτουργίες	*-deploy(name, vdc, vlan, image, cpu, ram)*: Δημιουργεί και ξεκινάει έναν server με τις ελάχιστες δυνατές απαιτήσεις της εφαρμογής κι επιστρέφει τη διεύθυνση IP του. *-undeploy(vdc, ips)*: Σταματάει όλους τους servers μιας εφαρμογής και ύστερα τους καταστρέφει. *-scaleUp(name, vdc, vlan, baseImage, cpu, ram)*: Δημιουργεί έναν νέο server με την υπολογιστική ισχύ που ζητείται κι επιστρέφει τη διεύθυνση IP του. *-scaleDown(vdc, ip)*: Καταστρέφει έναν υπάρχων server.

Κλάση FlexiantManager

Περιγραφή	Υλοποιεί το interface "ManagerInterface" κι αποτελεί μια βιβλιοθήκη με μεθόδους για εύκολη κι αποδοτική διαχείριση των πόρων υποδομής. Καλεί τις μεθόδους της κλάσης FlexiantDriver, αλλά κάνει κι επιπλέον ελέγχους για τον έλεγχο των πόρων και την ομαλή λειτουργία των εφαρμογών.
Γνωρίσματα	
Λειτουργίες	*-deploy(name, deployment_id, image, cpu, ram)*: Δημιουργεί έναν proxy server για τη νέα εφαρμογή. Στη συνέχεια δημιουργεί ακόμα ένα server με τις ελάχιστες απαιτήσεις για την εφαρμογή κι επιστρέφει τη διεύθυνση IP του. *-undeploy(deployment_id)*: Καταστρέφει όλους τους servers που τρέχουν, καθώς και τον proxy server της εφαρμογής. Αφαιρεί το αρχείο για αυτή την εφαρμογή από τη βάση δεδομένων. *-scaleUp(deployment_id, number, name, baseImage, cpu, ram)*: Δημιουργεί νέους servers για την εφαρμογή, για κάθε έναν από τους οποίους τρέχει ξεχωριστό thread που αναλαμβάνει τη δημιουργία τους, κι επιστρέφει τις διευθύνσεις IP τους. Επίσης, ανανεώνει το αρχείο ρυθμίσεων του proxy server και τη βάση δεδομένων. *-scaleDown(deployment_id, number)*: Καταστρέφει ορισμένους servers της εφαρμογής, για κάθε έναν από τους οποίους τρέχει ένα thread, που αναλαμβάνει την αποδέσμευσή τους. Επίσης, ανανεώνει και το αρχείο ρυθμίσεων του proxy server, καθώς και τη βάση δεδομένων.

Κλάση ScaleDownRunnable

Περιγραφή	Υλοποίηση ενός thread που καταστρέφει έναν server. Έχει ως σκοπό την ταυτόχρονη αποδέσμευση πολλών servers, με παράλληλη λειτουργία threads.
Γνωρίσματα	*vdc, ip, driver*
Λειτουργίες	*-ScaleDownRunnable(vdc, ip, driver)*: Ο κατασκευαστής της κλάσης, ο οποίος θέτει τα γνωρίσματά της. *-run()*: Καλεί τη συνάρτηση του driver, που είναι υπεύθυνη για την αποδέσμευση ενός server.

Κλάση ScaleUpCallable

Περιγραφή	Υλοποίηση ενός thread που δημιουργεί έναν server. Έχει ως σκοπό την ταυτόχρονη δέσμευση πολλών servers, με παράλληλη λειτουργία των threads.
Γνωρίσματα	*name, vdc, vlan, image, image, ram, driver*
Λειτουργίες	*-ScaleUpCallable(name, vdc, vlan, image, cpu, ram, driver)*: Ο κατασκευαστής της κλάσης, ο οποίος θέτει τα γνωρίσματά της. *-call()*: Καλεί τη συνάρτηση του driver, που είναι υπεύθυνη για την δέσμευση ενός server κι ελέγχει την επιτυχή δημιουργία του.

Κλάση RestClient

Περιγραφή	Καλεί το REST service ενός proxy server, το οποίο ανανεώνει το αρχείο ρυθμίσεων nginx.conf.
Γνωρίσματα	
Λειτουργίες	*-checkConnection(ip)*: Ελέγχει την επικοινωνία με τον proxy server. *-updateProxy(proxy, ips)*: Καλεί τη μέθοδο για τη σύνδεση με τον proxy. *-makeCall(proxy, ips)*: Καλεί το REST service του proxy. *-makeJsonString(ips)*: Δημιουργεί το json αρχείο που θα σταλεί στον proxy. *-getStringFromInputStream(is)*: Επεξεργάζεται την απάντηση του REST service που καλείται.

7.3.3 Διακομιστής μεσολάβησης (Proxy Server)

Κλάση CumulativeMonitoringData

Περιγραφή	Κλάση με setters και getters των στατιστικών στοιχείων του proxy server.
Γνωρίσματα	*timeStamp, numOfBytes, averageResponseTimes, numOfResponseTimes, numOfTotalBytesInFile, numOfRows*
Λειτουργίες	Setters και Getters για όλα τα γνωρίσματα.

Κλάση NginxExtendedDeamonMain

Περιγραφή	Ένας δαίμονας που τρέχει σε κάθε proxy server. Επεξεργάζεται τα στατιστικά και τα στέλνει στη βάση δεδομένων της πλατφόρμας.
Γνωρίσματα	
Λειτουργίες	*-main()*: Η κύρια μέθοδος της κλάσης, η οποία καλεί το δαίμονα. *-runDeamon()*: Ο δαίμονας, ο οποίος εκτελείται συνεχώς κι ο οποίος καλεί τη συνάρτηση που επεξεργάζεται το log file του proxy. *-clearLogs()*: Αδειάζει το log file του proxy, αν ο χρόνος επεξεργασίας του είναι πολύ μεγάλος.

Κλάση ProcessLogs

Περιγραφή	Επεξεργάζεται το log file του proxy και στέλνει τα στατιστικά που αντλεί στη βάση δεδομένων της πλατφόρμας.
Γνωρίσματα	
Λειτουργίες	*-processLogs(filename, position, timestamp)*: Επεξεργάζεται τα στατιστικά του αρχείου access.log για τα τελευταία δέκα δευτερόλεπτα. Στη συνέχεια, καλεί το REST service της πλατφόρμας, για την προσθήκη των στατιστικών στη βάση δεδομένων. *-fixMonth(dateString)*: Μετατρέπει το μήνα από τη μορφή που είναι στο log file, σε αριθμητική μορφή.

Κλάση RestClient

Περιγραφή	Καλεί το REST service της πλατφόρμας, το οποίο προσθέτει στη βάση δεδομένων τα νέα στατιστικά στοιχεία της εφαρμογής.
Γνωρίσματα	
Λειτουργίες	*-updateDB(ip, date, numOfResponseTimes, numOfBytes, averageResponseTimes)*: Καλεί τη μέθοδο για τη σύνδεση με τη βάση δεδομένων. *-makeCall(json, ip)*: Καλεί το REST service της πλατφόρμας, για την εισαγωγή στοιχείων στη βάση δεδομένων. *-makeJsonString(json)*: Δημιουργεί το json αρχείο που θα σταλεί στην πλατφόρμα. *-getStringFromInputStream(is)*: Επεξεργάζεται την απάντηση του REST service που καλείται.

Κλάση UpdateNodes

Περιγραφή	Μία κλάση που υλοποιεί ένα REST service, για την τροποποίηση του αρχείου ρυθμίσεων nginx.conf.
Γνωρίσματα	*context*
Λειτουργίες	*-check()*: Ελέγχει αν το service είναι σε λειτουργία. *-putJson(content)*: Καλεί τη μέθοδο για την εισαγωγή των νέων διευθύνσεων IP στο αρχείο ρυθμίσεων κι επιστρέφει το αποτέλεσμα επιτυχίας του service.

Κλάση NginxUpdateNodes

Περιγραφή	Κλάση για την εισαγωγή των νέων διευθύνσεων IP στο αρχείο ρυθμίσεων του proxy server.
Γνωρίσματα	
Λειτουργίες	*-updateNodes(strJson)*: Καλεί τη μέθοδο για την τροποποίηση του αρχείου ρυθμίσεων κι επιστρέφει το αποτέλεσμα. *-getFromString(str)*: Παίρνει τις διευθύνσεις IP από μία συμβολοσειρά και τις αποθηκεύει σε μία λίστα. *-updateConfiguration(ips)*: Εισάγει τις νέες διευθύνσεις IP στο αρχείο ρυθμίσεων του proxy server. Αντιγράφει το βασικό αρχείο και προσθέτει τις διευθύνσεις IP των μηχανημάτων που τρέχουν την εφαρμογή, τις οποίες παίρνει από τον driver μέσω του REST service. *-saveToFile(str, filename)*: Αποθηκεύει μία συμβολοσειρά σε ένα αρχείο. *-addServers(ips, sb)*: Κατασκευάζει μία συμβολοσειρά με συγκεκριμένη μορφή, η οποία περιέχει τις διευθύνσεις IP των μηχανημάτων που τρέχουν την εφαρμογή. *-createJson(result)*: Αποθηκεύει σε ένα json αρχείο το αποτέλεσμα της τροποποίησης του αρχείου ρυθμίσεων.

7.3.4 Μηχανή ελαστικότητας (Elasticity Engine)

Κλάση ScalabilityDaemon

Περιγραφή	Ο δαίμονας της πλατφόρμας που έχει ως σκοπό την ομαλή λειτουργία κάθε εφαρμογής, ελέγχοντας τους χρόνους απόκρισης και το πλήθος των χρηστών και δεσμεύοντας-αποδεσμεύοντας πόρους αναλόγως.
Γνωρίσματα	
Λειτουργίες	*-main()*: Η κύρια μέθοδος της κλάσης. Τρέχει το δαίμονα που αναλαμβάνει τον υπολογισμό των απαιτούμενων πόρων και τη δέσμευση-/αποδέσμευσή τους. *-runDeamon()*: Ο δαίμονας της πλατφόρμας. Εκτελείται συνέχεια κι ελέγχει τη λειτουργία κάθε εφαρμογής. Καλεί τον Requirement Resource Calculator για τον υπολογισμό των απαιτούμενων πόρων κάθε εφαρμογής. *-processDB(Statistics stats)*: Επεξεργάζεται τους τελευταίους χρόνους απόκρισης από τη βάση δεδομένων και βρίσκει το μέσο χρόνο απόκρισης.

Κλάση Monitoring

Περιγραφή	Σύστημα παρακολούθησης το οποίο εκτελεί έναν απλό κανόνα για την κλιμάκωση των εφαρμογών, ανάλογα με το χρόνο απόκρισής τους.
Γνωρίσματα	
Λειτουργίες	*-scale(averageResponseTime, deployment_id, morethan, lessthan)*: Αν ο χρόνος απόκρισης είναι μεγαλύτερος από μία συγκεκριμένη τιμή, τότε δεσμεύει νέους πόρους. Αν είναι μικρότερος από κάποια ορισμένη τιμή, τότε αποδεσμεύει πόρους.

Κλάση ResourceRequirementCalculator

Περιγραφή	Η μηχανή η οποία υπολογίζει τους απαιτούμενους πόρους για κάθε εφαρμογή, δεδομένου του χρόνου απόκρισης και του πλήθους των χρηστών της εφαρμογής.
Γνωρίσματα	
Λειτουργίες	

7.3.5 Αλληλεπιδραστική εφαρμογή χρήστη (Platform GUI)

Κλάση DeployApplication

Περιγραφή	Servlet για την εγκατάσταση μίας νέας εφαρμογής.
Γνωρίσματα	
Λειτουργίες	*-DeployApplication()*: Κατασκευαστής της κλάσης. *-doGet()*: Καλεί την doPost(). *-doPost()*: Εγκαθιστά μια καινούρια εφαρμογή. Δημιουργεί ένα νέο αρχείο ρυθμίσεων Warrant-app.xml για την εφαρμογή και καλεί τον Driver για τη δημιουργία του proxy server και του αρχικού VM που τρέχει την εφαρμογή.

Κλάση UndeployApplication

Περιγραφή	Servlet για την απεγκατάσταση μίας υπάρχουσας εφαρμογής.
Γνωρίσματα	
Λειτουργίες	*-UndeployApplication()*: Κατασκευαστής της κλάσης. *-doGet()*: Καλεί την doPost(). *-doPost()*: Απεγκαθιστά μια υπαρχουσα εφαρμογή. Διαγράφει το νέο αρχείο ρυθμίσεων Warrant-app.xml της συγκεκριμένης εφαρμογής και καλεί τον Driver για την καταστροφή του proxy server και των VMs που τρέχουν την εφαρμογή.

Κλάση CreateHints

Περιγραφή	Servlet για την προσθήκη κανόνων κλιμάκωσης για μία εφαρμογή.
Γνωρίσματα	
Λειτουργίες	*-CreateHints()*: Κατασκευαστής της κλάσης. *-doGet()*: Καλεί την doPost(). *-doPost()*: Παίρνει τους νέους κανόνες κλιμάκωσης που δόθηκαν από τον πάροχο μέσω της web σελίδας και καλεί μία μέθοδο που αναλαμβάνει την εισαγωγή τους στο αρχείο ρυθμίσεων Warrant-app.xml.

Κλάση CreateTunables

Περιγραφή	Servlet για την προσθήκη συνθήκης κλιμάκωσης για μία εφαρμογή.
Γνωρίσματα	
Λειτουργίες	*-CreateTunables()*: Κατασκευαστής της κλάσης. *-doGet()*: Καλεί την doPost(). *-doPost()*: Παίρνει τη νέα συνθήκη κλιμάκωσης που δόθηκε από τον πάροχο μέσω της web σελίδας και καλεί μία μέθοδο που αναλαμβάνει την εισαγωγή της στο αρχείο ρυθμίσεων Warrant-app.xml της εφαρμογής. Ανακατευθύνει τον πάροχο στη web σελίδα για εισαγωγή νέων κανόνων για αυτή τη νέα συνθήκη.

Κλάση AddData

Περιγραφή	Κλάση για την προσθήκη δεδομένων στο αρχείο ρυθμίσεων Warrant-app.xml μιας εφαρμογής, καλώντας το αντίστοιχο REST service.
Γνωρίσματα	*app*
Λειτουργίες	*-AddData(app)*: Ο κατασκευαστής της κλάσης. *-addNewTunable(id, value)*: Καλεί το service για την προσθήκη νέας συνθήκης κλιμάκωσης. *-addNewRules(resource_name, product_name, instances, cpu, ram, up, resp)*: Καλεί το service για την προσθήκη νέου κανόνα. *-setArtefactSection(image, artefact_name, artefact_location, dbartefact_name, dbartefact_location)*: Καλεί το service για την προσθήκη του τμήματος με τα στοιχεία της εφαρμογής (όνομα, βάση, image).

Κλάση GetData

Περιγραφή	Κλάση για την άντληση δεδομένων από το αρχείο ρυθμίσεων Warrant-app.xml μιας εφαρμογής, καλώντας το αντίστοιχο REST service.
Γνωρίσματα	*app*
Λειτουργίες	*-GetData(app)*: Ο κατασκευαστής της κλάσης. *-getTunables()*: Καλεί το service για την άντληση των συνθηκών κλιμάκωσης της εφαρμογής. *-getTunable(i)*: Καλεί το service για την άντληση μίας συνθήκης. *-isTunableEmpty(i)*: Καλεί το service για να ελέγξει αν έχουν εισαχθεί συνθήκες κλιμάκωσης της εφαρμογής. *-getTunableVal(i, j)*: Καλεί το service για να πάρει την τιμή μίας συνθήκης κλιμάκωσης.

Κλάση Calculate

Περιγραφή	Κλάση για τον υπολογισμό των απαιτούμενων πόρων, με βάση τους κανόνες που δίνονται στο αρχείο ρυθμίσεων Warrant-app.xml μιας εφαρμογής, καλώντας το αντίστοιχο REST service.
Γνωρίσματα	*app*
Λειτουργίες	*-Calculate(app)*: Ο κατασκευαστής της κλάσης. *-calculate(uptousers, resptime)*: Καλεί το service για τον υπολογισμό των απαιτούμενων πόρων, με βάση τον αριθμό χρηστών και το χρόνο απόκρισης που δίνονται.

Κλάση CalculateResources

Περιγραφή	Μία κλάση που υλοποιεί ένα REST service, για τον υπολογισμό των απαιτούμενων πόρων, με βάση τους κανόνες που δίνονται στο αρχείο ρυθμίσεων Warrant-app.xml μιας εφαρμογής, καλώντας το αντίστοιχο REST service.
Γνωρίσματα	*uriInfo, request*
Λειτουργίες	*-getResources(uptousers, resptime, app)*: Υπολογίζει τους απαιτούμενους πόρους, με βάση τον αριθμό χρηστών και το χρόνο απόκρισης που δίνονται. Ο υπολογισμός γίνεται με επεξεργασία των κανόνωνπου δίνονται στο αρχείο Warrant-app.xml της εφαρμογής.

Κλάση CreateArtefact

Περιγραφή	Μία κλάση που υλοποιεί ένα REST service, για την προσθήκη στοιχείων σχετικά με την εφαρμογή, στο αρχείο ρυθμίσεων Warrant-app.xml.
Γνωρίσματα	*uriInfo, request*
Λειτουργίες	*-createArtefact(image, art_name, art_location, db_name, db_loc, app)* Προσθέτει στο κατάλληλο σημείο του xml αρχείου, πληροφορίες σχετικά με το image της εφαρμογής, το όνομά της, την τοποθεσία στην οποία βρίσκεται, καθώς και το αρχείο της βάσης δεδομένων, το όνομα και την τοποθεσία της.

Κλάση CreateRules

Περιγραφή	Μία κλάση που υλοποιεί ένα REST service, για την προσθήκη κανόνων στο αρχείο ρυθμίσεων Warrant-app.xml μιας εφαρμογής.
Γνωρίσματα	*uriInfo, request*
Λειτουργίες	*-createRule(uval, rval, resource, product, inst, cpu_req, ram_req, app)* Εισάγει στο κατάλληλο σημείο του xml αρχείου έναν νέο κανόνα, αντιστοιχίζοντας στις συνθήκες αριθμού χρηστών και χρόνου απόκρισης τους κατάλληλους πόρους. Προσθέτει πληροφορίες για το απαιτούμενο πλήθος VMs, τη μνήμη και την υπολογιστική ισχύ τους.

Κλάση CreateTunable

Περιγραφή	Μία κλάση που υλοποιεί ένα REST service, για την προσθήκη συνθηκών κλιμάκωσης στο αρχείο ρυθμίσεων Warrant-app.xml μιας εφαρμογής.
Γνωρίσματα	*uriInfo, request*
Λειτουργίες	*-createTunableval(id, value, app)*: Εισάγει στο κατάλληλο σημείο του xml αρχείου μία νέα συνθήκη κλιμάκωσης. Όταν ικανοποιούνται κάποιες συνθήκες, τότε εφαρμόζονται οι αντίστοιχοι κανόνες κλιμάκωσης για αυτές τις συνθήκες.

Κλάση GetXmlData

Περιγραφή	Μία κλάση που υλοποιεί ένα REST service, για την άντληση δεδομένων από το αρχείο ρυθμίσεων Warrant-app.xml μιας εφαρμογής.
Γνωρίσματα	*uriInfo, request*
Λειτουργίες	*-getProperty(app)*: Επιστρέφει όλη την ενότητα του αρχείου ρυθμίσεων Warrant-app.xml της εφαρμογής, η οποία αφορά τις συνθήκες και τους κανόνες κλιμάκωσης της εφαρμογής. *-getArtefact(app)*: Επιστρέφει όλη την ενότητα του αρχείου ρυθμίσεων Warrant-app.xml της εφαρμογής, η οποία αφορά την υπηρεσία βάση δεδομένων της. Περιέχει το image που θα τρέχουν οι application servers και το αρχείο δημιουργίας της βάσης δεδομένων. *-getTunables(app)*: Επιστρέφει το τμήμα του xml αρχείου με τις συνθήκες κλιμάκωσης της εφαρμογής. Μία συνθήκη είναι ένα ζευγάρι χρόνου απόκρισης και αριθμού χρηστών. *-getTunable(app)*: Επιστρέφει μία συγκεκριμένη συνθήκη κλιμάκωσης. *-isTunableEmpty(app)*: Ελέγχει αν έχουν εισαχθεί συνθήκες κλιμάκωσης. *-getTunableVal(i, j, app)*: Επιστρέφει την τιμή ενός πεδίου μιας συνθήκης κλιμάκωσης.

Κεφάλαιο 8

Αποτελέσματα εργασίας

8.1 Λειτουργία πλατφόρμας

Για τη μελέτη της λειτουργίας της πλατφόρμας, ελήφθησαν μετρήσεις για μία δοκιμαστική διαδικτυακή εφαρμογή που χρησιμοποιήθηκε για benchmarking κι ονομάζεται flipper. Οι κανόνες κλιμάκωσης ορίστηκαν με βάση τον επιθυμητό χρόνο απόκρισης της εφαρμογής, ενώ δοκιμάστηκε κυρίως οριζόντια κλιμάκωση για πιο γρήγορη και άμεση κλιμάκωση του συστήματος. Έτσι, πάρθηκαν μετρήσεις για application servers με χαρακτηριστικά 1CPU-1024MB RAM, 2CPU-2048MB RAM και 4CPU-4096MB RAM. Ο database server της εφαρμογής είχε σε όλες τις περιπτώσεις χαρακτηριστικά 4CPU-8192MB RAM, ώστε να μην παρατηρείται καθυστέρηση στην άντληση των δεδομένων από τη βάση. Έτσι, ανάλογα με το χρόνο απόκρισης, δημιουργούνταν νέα instances από application servers η καταστρέφονταν κάποια από τα υπάρχοντα.

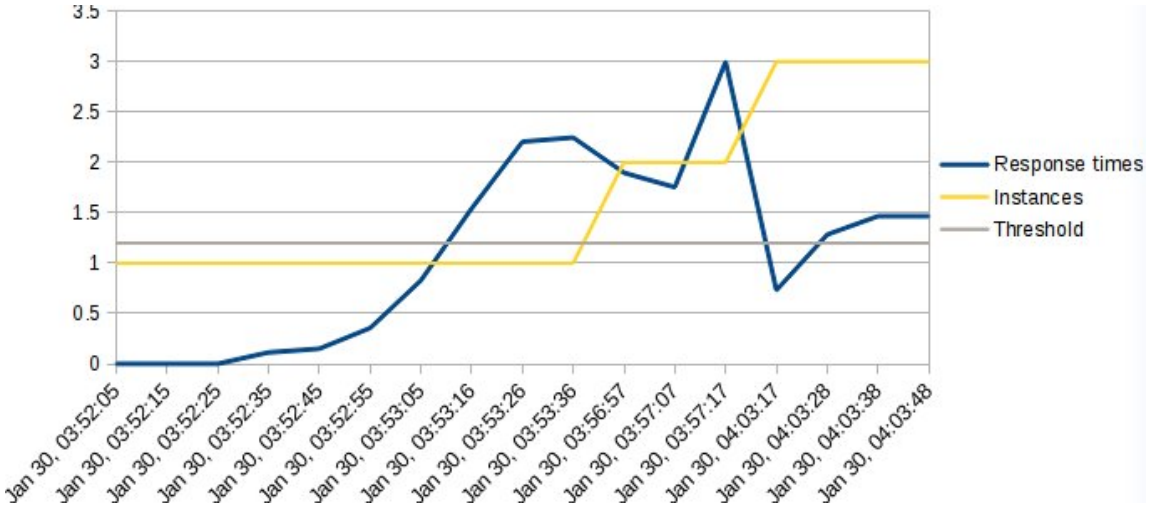
Πιο συγκεκριμένα, επιλέχθηκε μία τιμή ως άνω όριο για τον αποδεκτό χρόνο απόκρισης της εφαρμογής. Όταν αυτός ξεπερνιέται, τότε γίνεται αυτόματα scale-up από την πλατφόρμα. Αντίθετα, όταν ο μέσος χρόνος απόκρισης πέφτει κάτω από μία άλλη τιμή, τότε γίνεται αυτόματα scale-down από την πλατφόρμα. Η τιμή αυτή είναι μικρότερη (περίπου κατά 1/3) από την τιμή άνω ορίου, έτσι ώστε να μην γίνονται συνέχεια scale-up και scale-down και να υπάρχει μία σχετική σταθερότητα στο σύστημα, διατηρώντας παράλληλα το χρόνο απόκρισης σε επιθυμητά επίπεδα.

Για τον ίδιο λόγο, ο μέσος χρόνος απόκρισης της εφαρμογής υπολογίζεται ως ο μέσος όρος των χρόνων απόκρισης για το τελευταίο μισό λεπτό λειτουργίας της πλατφόρμας, ενώ για να γίνει κλιμάκωση θα πρέπει να είναι μεγαλύτεροι του άνω (ή αντίστοιχα του κάτω) ορίου οι 6 τελευταίοι μέσοι χρόνοι απόκρισης. Επίσης, μετά από κάθε ενέργεια κλιμάκωσης η πλατφόρμα αναμένει για ένα λεπτό μέχρι να σταθεροποιηθεί το σύστημα.

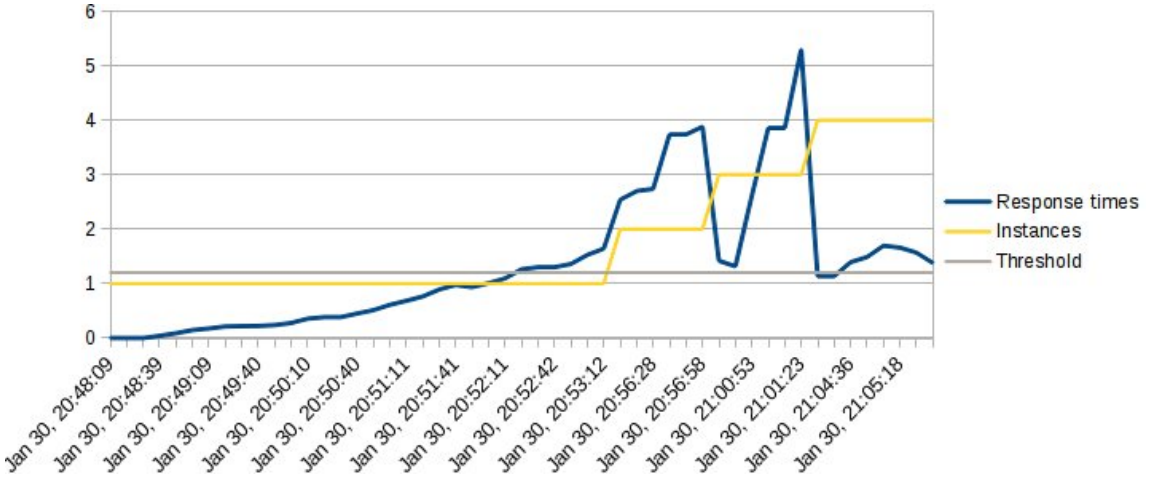
Στη συνέχεια παρατίθεται οι μετρήσεις που ελήφθησαν για τα διάφορα μεγέθη instances και με τους χρήστες να αυξομοιώνονται κοντά στις οριακές τιμές κλιμάκωσης, ώστε να δοκιμαστεί η σταθερότητα του συστήματος.

Instances 1CPU-1024MB RAM

Εδώ παραθέτουμε τις μετρήσεις για application servers με 1CPU, 1024MB RAM. Αρχικά, δημιουργήθηκαν χρήστες με το εργαλείο Apache jmeter. Όπως είναι προφανές, με την αύξηση του πλήθους των χρηστών παρατηρείται αύξηση και στους χρόνους απόκρισης της εφαρμογής. Όταν ο μέσος χρόνος απόκρισης ξεπερνάει για κάποιο χρονικό διάστημα το κατώφλι που επιλέχθηκε, τότε γίνεται scale-up, με αποτέλεσμα ο χρόνος απόκρισης να έρχεται σε επιθυμητά επίπεδα. Η παραπάνω διαδικασία επαναλαμβάνεται, μέχρι να σταματήσουν να αυξάνονται περαιτέρω οι χρήστες, με αποτέλεσμα το σύστημα να σταθεροποιείται.

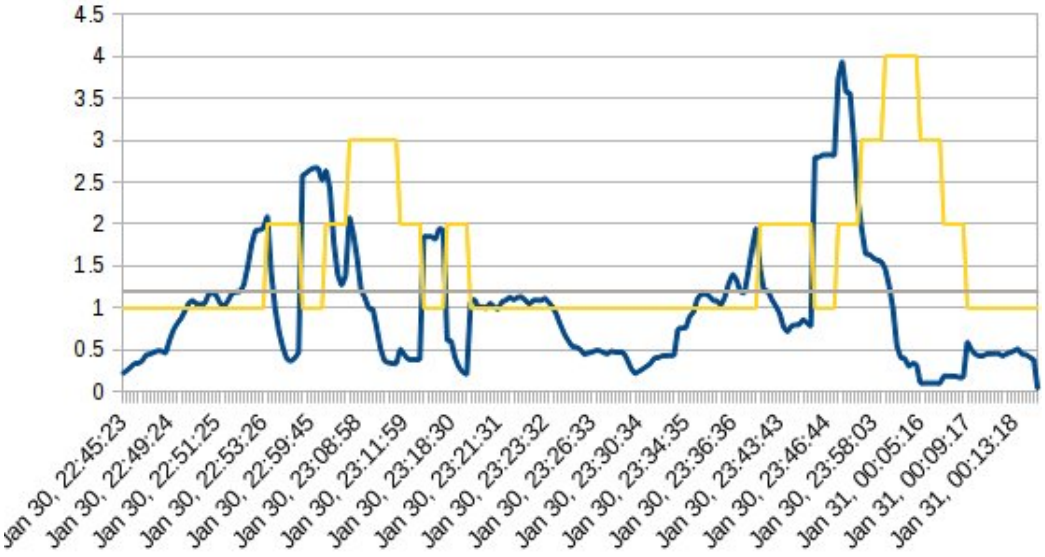


Σχήμα 8.1: Λειτουργία για instances 1CPU-1024MB, έως 50 χρήστες.



Σχήμα 8.2: Λειτουργία για instances 1CPU-1024MB, έως 80 χρήστες.

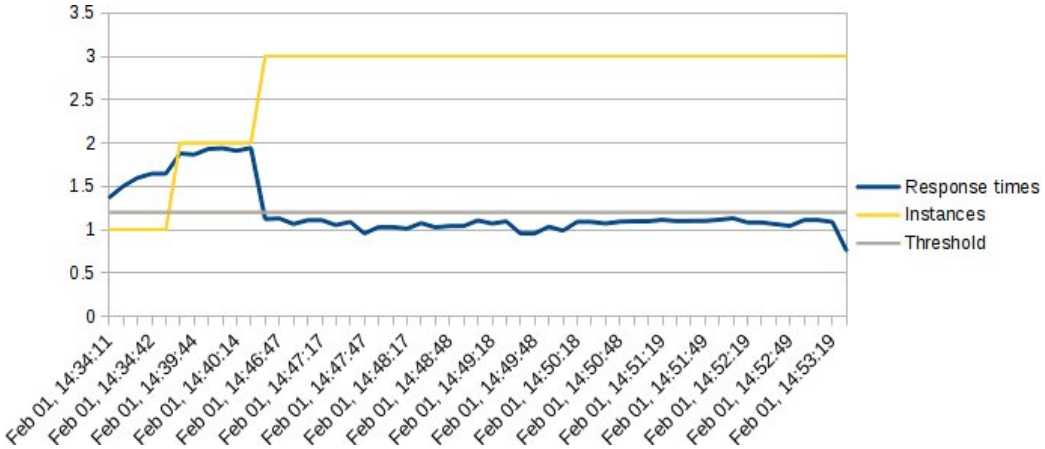
Στη συνέχεια, παραθέτουμε τη συμπεριφορά του συστήματος για ένα υποθετικό σενάριο λειτουργίας, με αυξομοίωση του πλήθους των χρηστών γύρω από τα σημεία κλιμάκωσης.



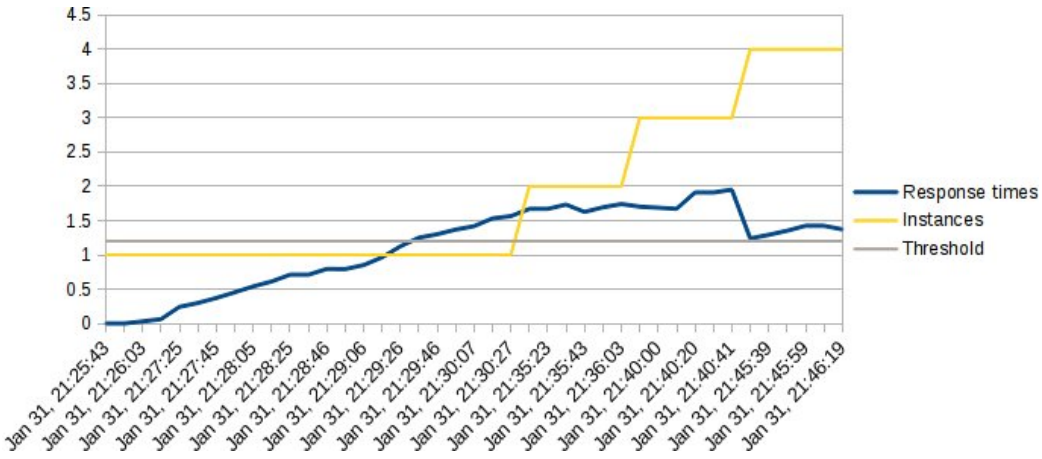
Σχήμα 8.3: Σενάριο λειτουργία για instances 1CPU-1024MB.

Instances 2CPU-2048MB RAM

Ακολούθως, φαίνονται οι μετρήσεις για application servers με 2CPU, 2048MB RAM.

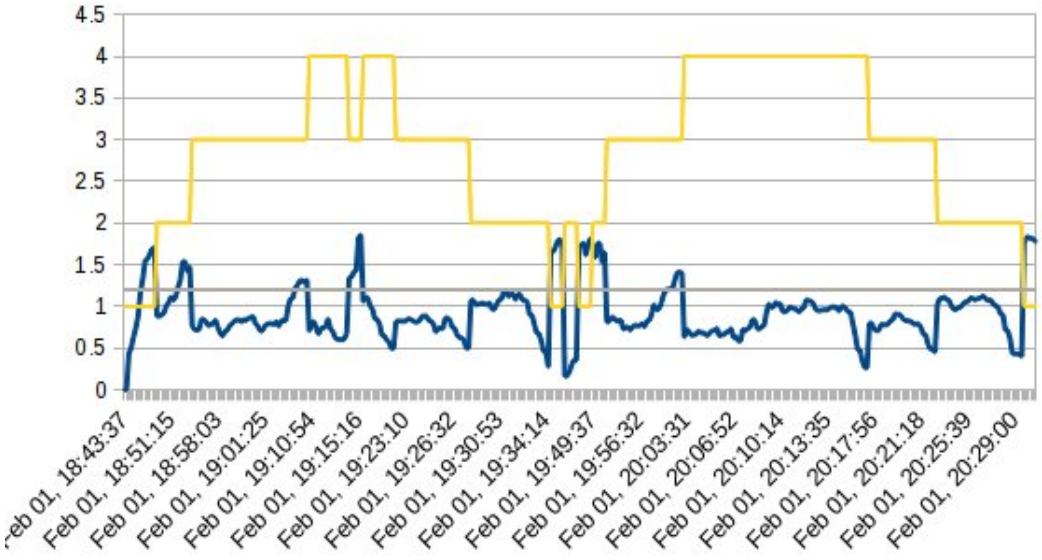


Σχήμα 8.4: Λειτουργία για instances 2CPU-2048MB, έως 50 χρήστες.



Σχήμα 8.5: Λειτουργία για instances 2CPU-2048MB, έως 90 χρήστες.

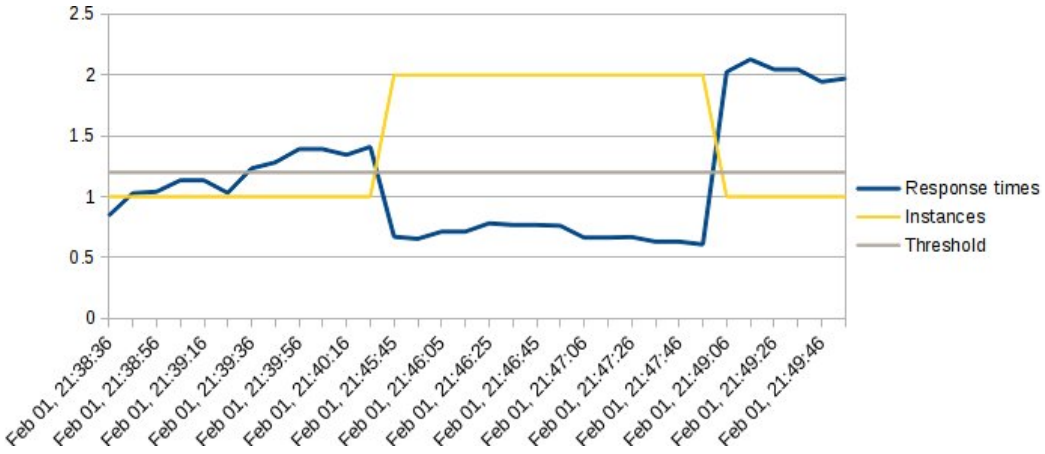
Η συμπεριφορά του συστήματος για ένα υποθετικό σενάριο λειτουργίας, φαίνεται παρακάτω. Όπως και πριν, έτσι και τώρα το σενάριο περιλαμβάνει αυξομοίωση του πλήθους των χρηστών γύρω από τα σημεία κλιμάκωσης, με σκοπό τη δοκιμή της σταθερότητάς του. Αν και το σημαντικότερο κριτήριο είναι η διατήρηση του χρόνου απόκρισης κάτω από ένα άνω όριο, δεν είναι επιθυμητή συνεχής κλιμάκωση του συστήματος.



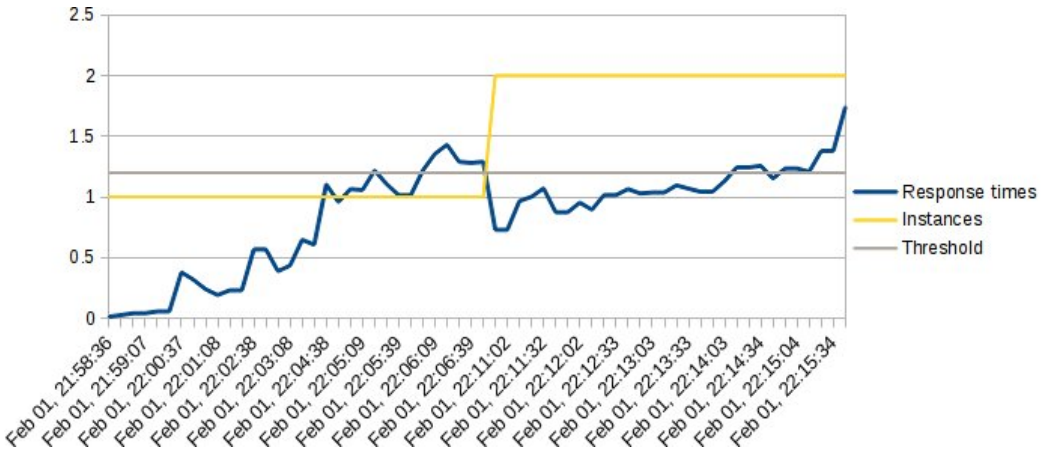
Σχήμα 8.6: Σενάριο λειτουργία για instances 2CPU-2048MB.

Instances 4CPU-4096MB RAM

Τέλος, παρατίθενται οι μετρήσεις για application servers με 4CPU, 4096MB RAM. Για τέτοιο μέγεθος instances, όπως είναι φυσικό παρατηρήθηκε ότι ο χρόνος απόκρισης μειώνεται αρκετά. Ως εκ τούτου, οι μετρήσεις πάρθηκαν για αρκετά μεγαλύτερο πλήθος χρηστών της εφαρμογής.

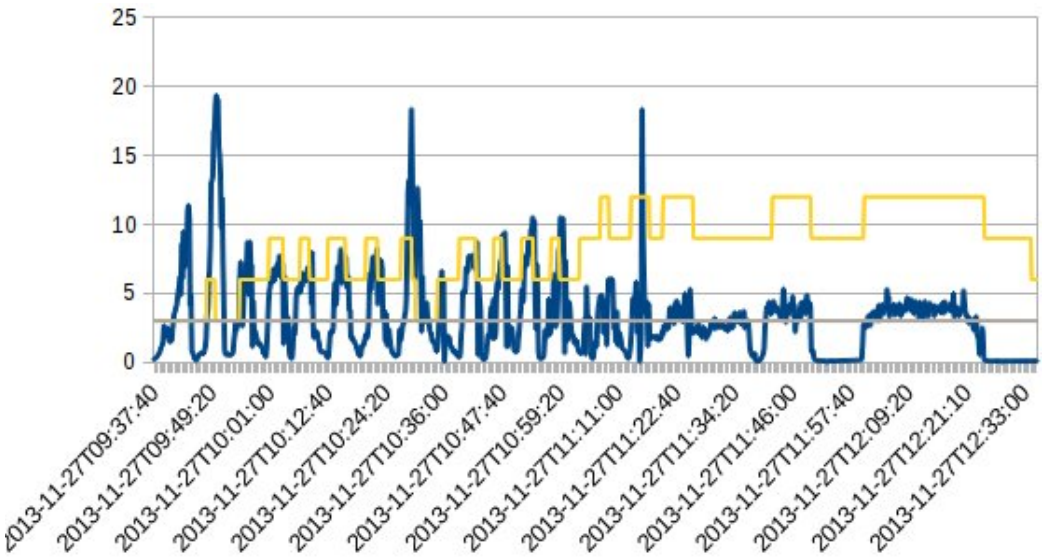


Σχήμα 8.7: Λειτουργία για instances 4CPU-4096MB, έως 80 χρήστες.



Σχήμα 8.8: Λειτουργία για instances 4CPU-4096MB, έως 110 χρήστες.

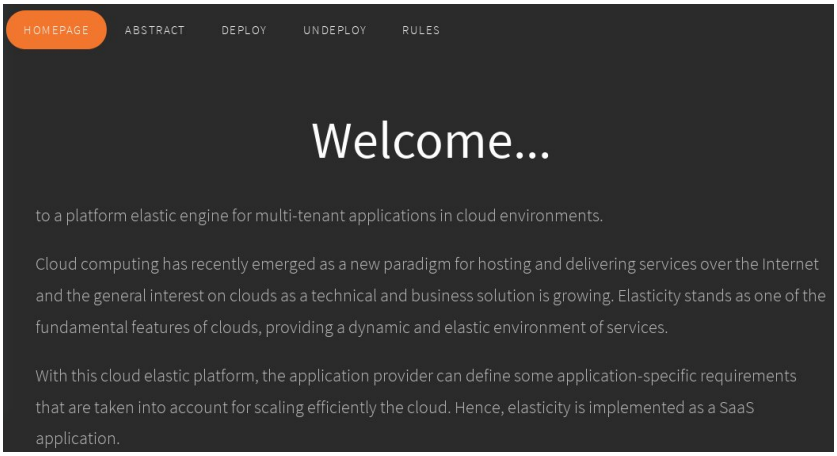
Η συμπεριφορά του συστήματος για ένα υποθετικό σενάριο λειτουργίας, φαίνεται στη συνέχεια:



Σχήμα 8.9: Σενάριο λειτουργία για instances 4CPU-4096MB.

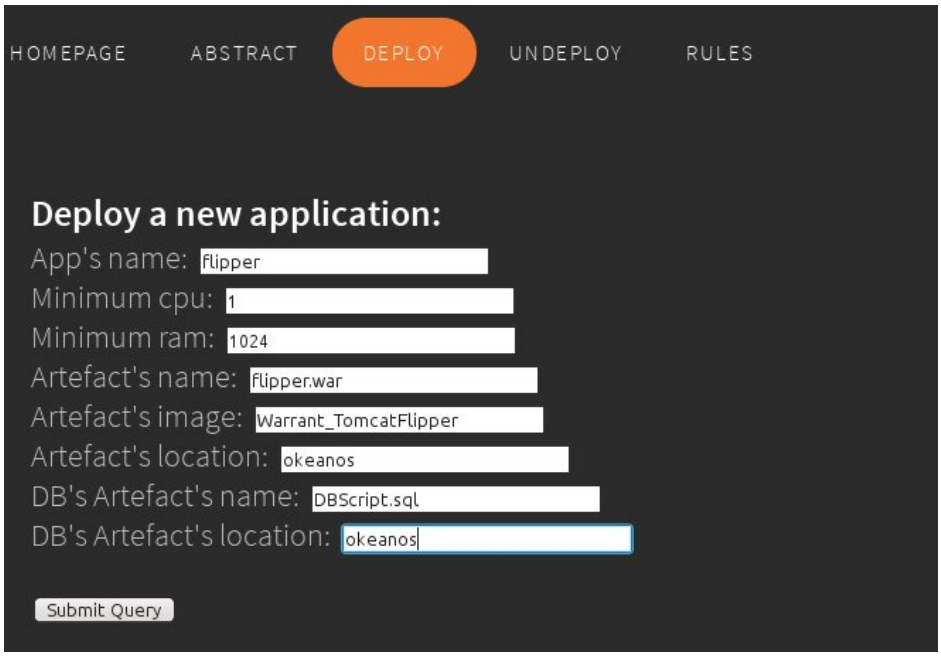
8.2 Γραφικό περιβάλλον πλατφόρμας

Στη συνέχεια παρατίθεται στιγμιότυπα από το γραφικό περιβάλλον της διαδικτυακής εφαρμογής της πλατφόρμας. Η σελίδα υποδοχής φαίνεται παρακάτω:



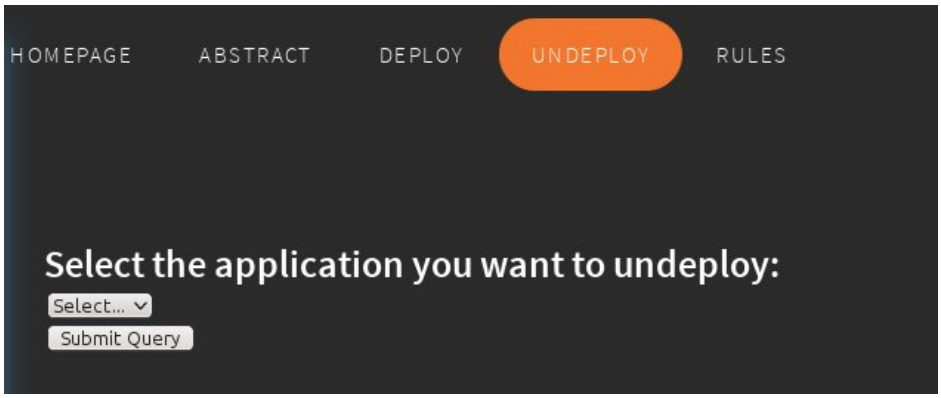
Σχήμα 8.10: Σελίδα υποδοχής στην πλατφόρμα.

Σε περίπτωση που ένας πάροχος επιλέξει να εγκαταστήσει μία νέα εφαρμογή, θα πρέπει να συμπληρώσει ορισμένα πεδία, ώστε να καταχωρηθεί η εφαρμογή στη βάση δεδομένων και να δημιουργηθεί το αρχείο ρυθμίσεών της. Στη συνέχεια, του ζητείται να προσθέσει κανόνες κλιμάκωσης, οι οποίοι αποθηκεύονται στο αρχείο ρυθμίσεων.



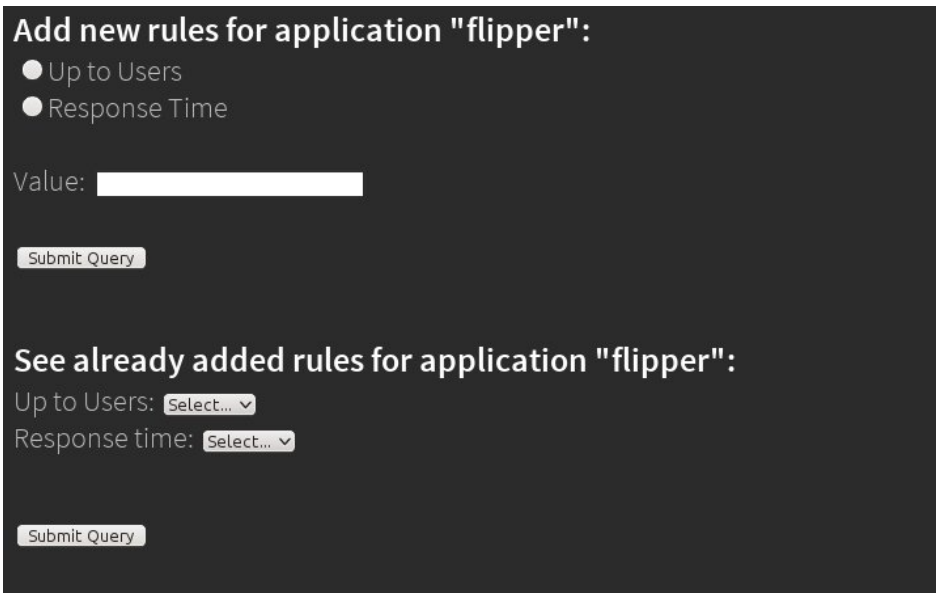
Σχήμα 8.11: Εγκατάσταση εφαρμογής στην πλατφόρμα.

Ένας πάροχος μπορεί να να απεγκαταστήσει μία εφαρμογή από την εξής σελίδα:



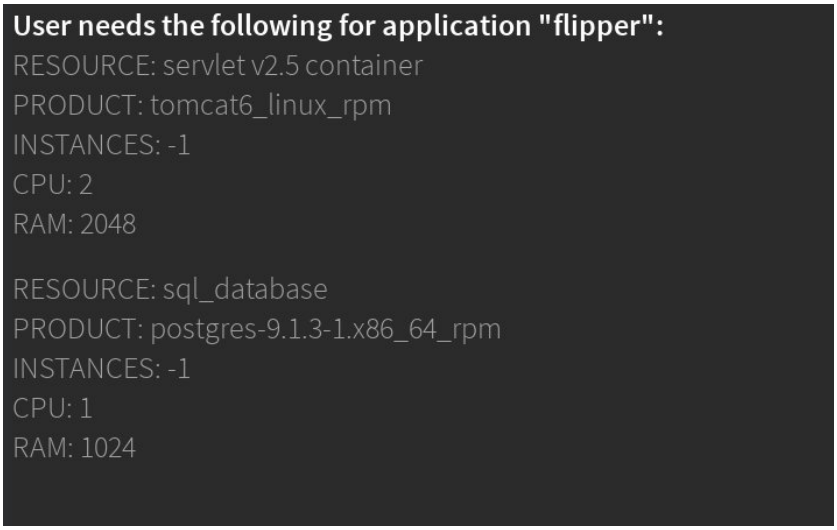
Σχήμα 8.12: Απεγκατάσταση εφαρμογής.

Ανά πάσα στιγμή, κάποιος πάροχος μπορεί να προσθέσει νέους κανόνες κλιμάκωσης για μία εφαρμογή, ή να δει τους ήδη υπάρχοντες:



Σχήμα 8.13: Ορισμός κανόνων.

Για παράδειγμα, οι απαιτούμενοι πόροι για την εφαρμογή flipper, για 50 χρήστες και χρόνο απόκρισης 1 δευτερόλεπτο φαίνονται παρακάτω:



Σχήμα 8.14: Απαιτούμενοι πόροι για μία εφαρμογή.

8.3 Σενάρια λειτουργίας

Στη συνέχεια, παραθέτουμε κάποια σενάρια λειτουργίας της εφαρμογής flipper, παραθέτοντας στιγμιότυπα από τη βάση δεδομένων κι από το γραφικό περιβάλλον που παρέχει ο πάροχος υποδομής. Αυτό το γραφικό περιβάλλον βοηθάει στη δημιουργία και τη διαχείριση των εικονικών μηχανών.

Εγκατάσταση εφαρμογής

Αρχικά, δεν είναι εγκατεστημένη καμία εφαρμογή στην πλατφόρμα:

```
> db.apps.find();
>
```

Κατά την εγκατάσταση μίας εφαρμογής, δημιουργείται ο proxy server της εφαρμογής κι ένας application server με τις ελάχιστες δυνατότητες ώστε να λειτουργήσει η εφαρμογή.

Σχήμα 8.15: Εγκατάσταση εφαρμογής.

Παράλληλα, η εφαρμογή καταχωρείται στη βάση δεδομένων:

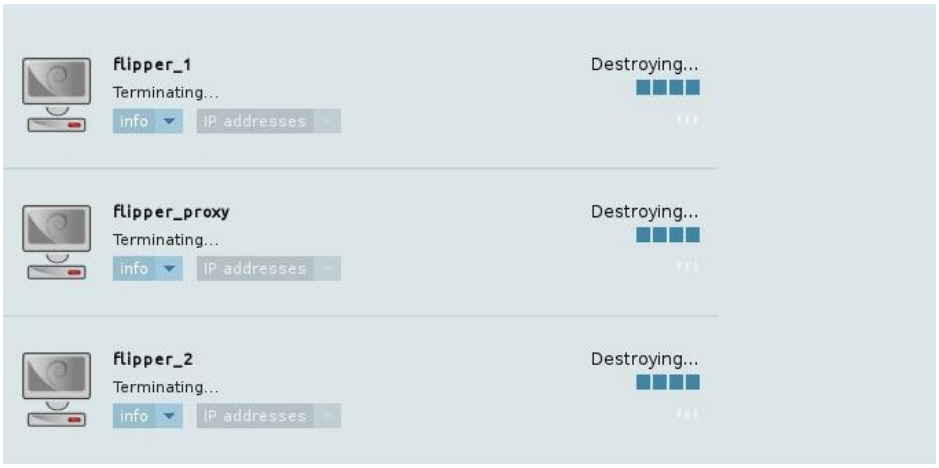
```
> db.apps.find();
{ "_id" : ObjectId("52e58f8de4b084ce80f20d1d"), "name" : "flipper", "deployment_id" : 1, "proxyIP" : "109.231.85.188", "
instances" : 1, "cpu" : 1, "ram" : 1024, "image" : "Warrant_TomcatFlipper", "ips" : [  "109.231.87.2" ], "scalepoint" :
ISODate("2014-01-26T22:43:25.131Z") }
```

Απεγκατάσταση εφαρμογής

Έστω ότι υπάρχει μία εφαρμογή στην πλατφόρμα, για την οποία λειτουργούν δύο application servers:

```
> db.apps.find();
{ "_id" : ObjectId("52e58f8de4b084ce80f20d1d"), "name" : "flipper", "deployment_id" : 1, "proxyIP" : "109.231.85.188", "
instances" : 2, "cpu" : 1, "ram" : 1024, "image" : "Warrant_TomcatFlipper", "ips" : [  "109.231.87.2",  "109.231.87.3" ]
, "scalepoint" : ISODate("2014-01-26T22:43:25.131Z") }
```

Κατά την απεγκατάσταση μίας εφαρμογής, καταστρέφεται ο proxy server της εφαρμογής και όλοι οι application servers που τρέχουν εκείνη τη στιγμή.



Σχήμα 8.16: Απεγκατάσταση εφαρμογής.

Πλέον, στη βάση δεδομένων δεν υπάρχει καμία καταχωρημένη εφαρμογή:

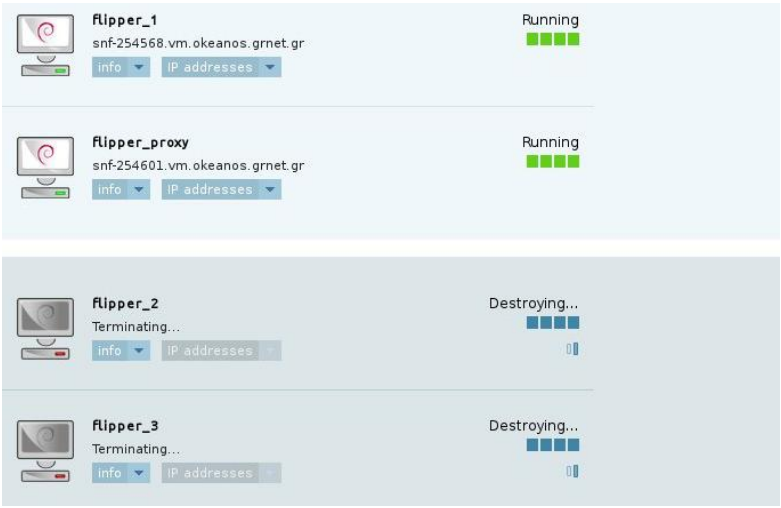
```
> db.apps.find();
>
```

Scale-down

Έστω τώρα ότι υπάρχει μία εφαρμογή στην πλατφόρμα, για την οποία λειτουργούν τρεις application servers:

```
> db.apps.find();
{ "_id" : ObjectId("52e58f8de4b084ce80f20d1d"), "name" : "flipper", "deployment_id" : 1, "proxyIP" : "109.231.85.188", "
instances" : 3, "cpu" : 1, "ram" : 1024, "image" : "Warrant_TomcatFlipper", "ips" : [  "109.231.87.2",  "109.231.87.3",
 "109.231.87.4" ], "scalepoint" : ISODate("2014-01-26T22:43:25.131Z") }
```

Κατά την αποκλιμάκωση της εφαρμογής, καταστρέφονται δύο application servers.



Σχήμα 8.17: Αποδέσμευση αχρείαστων πόρων.

Όπως φαίνεται κι από τη βάση δεδομένων, για την εφαρμογή τρέχει πλέον ένας application server:

```
> db.apps.find();
{ "_id" : ObjectId("52e58f8de4b084ce80f20d1d"), "name" : "flipper", "deployment_id" : 1, "proxyIP" : "109.231.85.188", "
instances" : 1, "cpu" : 1, "ram" : 1024, "image" : "Warrant_TomcatFlipper", "ips" : [  "109.231.87.2" ], "scalepoint" :
ISODate("2014-01-26T22:43:25.131Z") }
```

Βάση δεδομένων

Στη συνέχεια φαίνονται οι τελευταίες εγγραφές της συλλογής flipper.stats της βάσης δεδομένων. Πιο συγκεκριμένα φαίνονται τα στατιστικά για το τελευταίο λεπτό λειτουργίας της δοκιμαστικής εφαρμογής flipper:

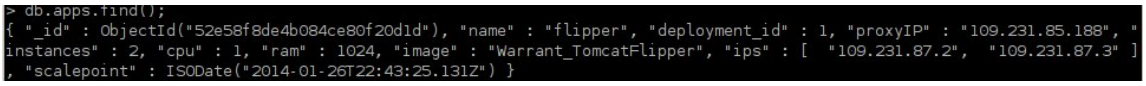
```
> db["flipper.stats"].find().sort({_id:-1}).limit(6);
{ "_id" : ObjectId("52f218320cf2c872b3df639c"), "timeStamp" : ISODate("2014-02-05T10:53:38Z"), "averageResponseTimes" :
"1.9893437500000009", "numOfResponseTimes" : 96, "bytes" : 353880, "calling" : "nginxDaemon", "IP" : "83.212.123.11" }
{ "_id" : ObjectId("52f218270cf2c872b3df639b"), "timeStamp" : ISODate("2014-02-05T10:53:27Z"), "averageResponseTimes" :
"1.9657244897959185", "numOfResponseTimes" : 98, "bytes" : 362457, "calling" : "nginxDaemon", "IP" : "83.212.123.11" }
{ "_id" : ObjectId("52f2181d0cf2c872b3df639a"), "timeStamp" : ISODate("2014-02-05T10:53:16Z"), "averageResponseTimes" :
"1.9705096153846144", "numOfResponseTimes" : 104, "bytes" : 382510, "calling" : "nginxDaemon", "IP" : "83.212.123.11" }
{ "_id" : ObjectId("52f218120cf2c872b3df6399"), "timeStamp" : ISODate("2014-02-05T10:53:06Z"), "averageResponseTimes" :
"2.0031067961165054", "numOfResponseTimes" : 103, "bytes" : 380222, "calling" : "nginxDaemon", "IP" : "83.212.123.11" }
{ "_id" : ObjectId("52f218070cf2c872b3df6398"), "timeStamp" : ISODate("2014-02-05T10:52:55Z"), "averageResponseTimes" :
"1.9433469387755111", "numOfResponseTimes" : 98, "bytes" : 361305, "calling" : "nginxDaemon", "IP" : "83.212.123.11" }
{ "_id" : ObjectId("52f217fd0cf2c872b3df6397"), "timeStamp" : ISODate("2014-02-05T10:52:44Z"), "averageResponseTimes" :
"2.0541300000000002", "numOfResponseTimes" : 100, "bytes" : 368326, "calling" : "nginxDaemon", "IP" : "83.212.123.11" }
```

Σχήμα 8.18: Τελευταία στατιστικά λειτουργίας της εφαρμογής flipper.

Τα στοιχεία που φαίνονται στις παραπάνω εγγραφές είναι τα εξής:

- _id: Το μοναδικό αναγνωριστικό της εγγραφής.
- timeStamp: Η χρονική στιγμή που στάλθηκε η εγγραφή στη βάση δεδομένων της πλατφόρμας.
- averageResponseTimes: Ο μέσος όρος των χρόνων απόκρισης για τα τελευταία δέκα δευτερόλεπτα λειτουργίας της εφαρμογής.
- numOfResponseTimes: Το πλήθος των παραπάνω χρόνων απόκρισης της εφαρμογής.
- bytes: Το συνολικό πλήθος bytes των δεδομένων που στάλθηκαν.
- calling: Η μονάδα που καλεί το REST service της πλατφόρμας για την εισαγωγή στη βάση δεδομένων. Στην περίπτωσή μας είναι ο δαίμονας που τρέχει στον proxy server.
- IP: Η διεύθυνση IP του proxy server στον οποίο τρέχει η παραπάνω μονάδα.

Τέλος, φαίνονται οι εγγραφές της συλλογής apps της βάσης δεδομένων. Πιο συγκεκριμένα φαίνονται τα στοιχεία των εφαρμογών που τρέχουν αυτή τη στιγμή στην πλατφόρμα:



Σχήμα 8.19: Εφαρμογές που τρέχουν στην πλατφόρμα

Τα στοιχεία που φαίνονται στις παραπάνω εγγραφές είναι τα εξής:

- _id: Το μοναδικό αναγνωριστικό της εγγραφής.
- name: Το όνομα της εφαρμογής.
- deployment_id: Το αναγνωριστικό της εφαρμογής, που δίνεται από τον driver.
- proxyIP: Η διεύθυνση IP του proxy server.
- instances: Το πλήθος των application servers που τρέχουν την εφαρμογή.
- cpu: Η υπολογιστική ισχύς των application servers.
- ram: Η μνήμη RAM των application servers.
- image: Το image των application servers της συγκεκριμένης εφαρμογής.
- ips: Οι διευθύνσεις IP των application servers που τρέχουν την εφαρμογή.
- scalepoint: Η χρονική στιγμή της τελευταίας κλιμάκωσης της εφαρμογής.

Κεφάλαιο 9

Σύνοψη και συμπεράσματα

9.1 Σύνοψη

Στην παρούσα διπλωματική εργασία παρουσιάστηκαν εκτενώς η αρχιτεκτονική και η υλοποίηση μίας πλατφόρμας υπολογιστικού νέφους, η οποία διαχειρίζεται την κλιμάκωση διαδικτυακών εφαρμογών, ανάλογα με τον αριθμό χρηστών και τους χρόνους απόκρισής τους και με τη χρήση εξατομικευμένων κανόνων.

Πιο συγκεκριμένα, υλοποιήθηκαν διάφορες μονάδες όπως μία αλληλεπιδραστική εφαρμογή για τον ορισμό των απαιτήσεων σε χρόνο απόκρισης και μέγιστο αριθμό χρηστών της εφαρμογής, ένας οδηγός για τη διαχείριση των πόρων υποδομής, το (image) ενός διακομιστή μεσολάβησης, η πλατφόρμα κλιμάκωσης και διάφορα REST services. Επίσης, εξηγήθηκαν οι λειτουργικές και μη λειτουργικές απαιτήσεις για την ανάπτυξη της πλατφόρμας

Για την αποτελεσματική λειτουργία και διεκπεραίωση της πλατφόρμας, χρησιμοποιήθηκαν διάφορες τεχνολογίες αιχμής, όπως τα αρχιτεκτονικά μοντέλα διακτυακών συστημάτων REST και SOAP, η mongoDB που είναι μία NoSQL βάση δεδομένων, η γλώσσα Java και η δημιουργία JSP και Servlets, αρχεία JSON και XML και η πλατφόρμα ανάπτυξης λογισμικού Eclipse.

Παράλληλα, εξηγήθηκαν τα πλεονεκτήματα της χρήσης εξατομικευμένων κανόνων κλιμάκωσης εφαρμογών για την ορθή και σταθερή λειτουργία τους και οι λόγοι για τους οποίους αυτή η νέα προσέγγιση ελαστικότητας των εφαρμογών υπερτερεί σε σχέση με άλλες μεθόδους.

Στο παρόν κεφάλαιο έγινε η παρουσίαση των μετρήσεων που αφορούσαν κάποια σενάρια λειτουργίας της δοκιμαστικής εφαρμογής flipper. Παράλληλα, παρουσιάστηκε η κλιμάκωση της εφαρμογής αυτής, ανάλογα με τους χρόνους απόκρισης για διάφορους αριθμούς χρηστών.

9.2 Συμπεράσματα και μελλοντική δουλειά

Η ανάγκη των εφαρμογών να ανταποκρίνονται όσο το δυνατόν καλύτερα στην τρέχουσα ζήτηση και να προσαρμόζονται στις αλλαγές φόρτου που παρουσιάζονται, δεσμεύοντας κι αποδεσμεύοντας πόρους αυτόνομα, οδήγησαν στην έννοια της ελαστικότητας. Μέχρι στιγμής, υπάρχουν δύο κύριες προσεγγίσεις ελαστικότητας. Η μία βασίζεται σε κανόνες οι οποίοι ορίζονται από τον πάροχο της πλατφόρμας, ενώ η άλλη δεσμεύει τους πόρους στατικά, ώστε να δει πότε υπάρχουν μέγιστα και ελάχιστα στο φόρτο του συστήματος και εν συνεχεία δημιουργεί κανόνες οι οποίοι ενσωματώνονται στην πλατφόρμα. Άλλες προσεγγίσεις περιλαμβάνουν τη χρήση νευρωνικών δικτύων και μοντέλων πρόβλεψης, αλλά σε κάθε περίπτωση με τις υπάρχουσες μεθόδους η κλιμάκωση έχει σχεδιαστεί για να καλύπτει τις συγκεκριμένες ανάγκες της πλατφόρμας.

Στην προσέγγιση που παρουσιάζεται στην παρούσα εργασία, η ελαστικότητα υλοποιείται ως μία υπηρεσία της πλατφόρμας, που είναι όμως ανεξάρτητη από αυτήν και ρυθμίζεται με βάση τις ανάγκες της εκάστοτε εφαρμογής. Έτσι, ενώ χρησιμοποιεί τα συστήματα παρακολούθησης της πλατφόρμας, οι κανόνες κλιμάκωσης κάθε εφαρμογής διαφέρουν, ανάλογα με τις απαιτήσεις καθεμιάς.

Από τη μελέτη της εφαρμογής flipper, είδαμε πως η χρήση τέτοιων εξατομικευμένων κανόνων είναι αποδοτική, μιας κι εξισορροπούνται αποτελεσματικά οι χρόνοι απόκρισης, ανάλογα με το φόρτο της εφαρμογής. Όταν ο μέσος χρόνος απόκρισης (ή ο αριθμός των χρηστών) περνάει ένα άνω όριο, τότε η εφαρμογή κλιμακώνεται κι έτσι ο χρόνος απόκρισης εξισορροπείται.

Παρόλα αυτά, υπάρχουν αρκετά περιθώρια βελτίωσης και θέματα για μελλοντική έρευνα. Θα μπορούσε να γίνει μελέτη και χρήση τόσο αντιδραστικών (reactive) όσο και προβλεπτικών (predictive) προσεγγίσεων για την ενεργοποίηση διορθωτικών κινήσεων, αλλά και χρήση συστημάτων μηχανικής μάθησης ή νευρωνικών δικτύων. Έτσι, σε μία μελλοντική εργασία θα μπορούσαν να γίνουν μετρήσεις για όλες τις μεθόδους κλιμάκωσης ή και για συνδιασμό τους. Ιδανικά, θα πρέπει να μπορεί ο χρήστης να επιλέξει τη λύση που ταιριάζει καλύτερα στις ανάγκες του ή και να μπορεί να εφαρμόσει συνδιασμό των παραπάνω για την εφαρμογή του.

Βιβλιογραφία

[1] Kranas P., et al. "ElaaS: An innovative Elasticity as a Service framework for dynamic management across the cloud stack layers." Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on. IEEE, 2012.

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

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

[4] Cheng, D.: PaaS-onomics: A CIO's Guide to using Platform-as-a-Service to Lower Costs of Application Initiatives While Improving the Business Value of IT. Technical report, Long Jump (2008).

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

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

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

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

[9] Η ιστορία του υπολογιστικού νέφους, http://en.wikipedia.org/wiki/Cloud_computing#History

[10] McCarthy, John. "Recursive functions of symbolic expressions and their computation by machine, Part I." Communications of the ACM 3.4 (1960): 184-195.

[11] Parkhill, Douglas F. The challenge of the computer utility. Vol. 2. Reading: Addison-Wesley Publishing Company, 1966.

[12] Ein-Dor, Phillip. "Grosch's law re-revisited: CPU power and the cost of computation." Communications of the ACM 28.2 (1985): 142-151.

[13] Amazon Web Services, http://aws.amazon.com/

[14] Το λογισμικό Eucalyptus, https://www.eucalyptus.com/

[15] Το μεσισμικό OpenNebula, http://opennebula.org/

[16] Valipour, M. Hadi, et al. "A brief survey of software architecture concepts and service oriented architecture." Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on. IEEE, 2009.

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

[18] Flexiscale IaaS, http://www.flexiscale.com/

[19] Kurze, Tobias, et al. "Cloud federation." CLOUD COMPUTING 2011, The Second International Conference on Cloud Computing, GRIDs, and Virtualization. 2011.

[20] Windows Azure PaaS, http://www.windowsazure.com/en-us/

[21] Apache Hadoop Framework, http://hadoop.apache.org/

[22] Google App Engine PaaS, https://cloud.google.com/products/app-engine/

[23] Google Apps SaaS, http://www.google.gr/intx/el/enterprise/apps/business/

[24] Pallis, George. "Cloud computing: the new frontier of internet computing." Internet Computing, IEEE 14.5 (2010): 70-73.

[25] Armbrust, Michael, et al. "A view of cloud computing." Communications of the ACM 53.4 (2010): 50-58.

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

[27] Kolodner, Elliot K., et al. "A cloud environment for data-intensive storage services." Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on. IEEE, 2011.

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

[29] Herbst, Nikolas Roman, Samuel Kounev, and Ralf Reussner. "Elasticity in Cloud Computing: What It Is, and What It Is Not."

[30] Rymer J., Gualtieri M., et al, "Cloud Computing Brings Demand For Elastic Application Platforms", Forrester, April 2011.

[31] Η υπηρεσιοστρεφής αρχιτεκτονική, http://en.wikipedia.org/wiki/Service-oriented_architecture#Web_services_approach

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

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

Μέρος III

Παράρτημα

Σημαντικές κλάσεις

IaaS Driver

Κυρίως κλάση του οδηγού

```
/**
 * The Class FlexiantManager.
 */
public class Driver implements DriverInterface {

   /* (non-Javadoc)
    * @see gr.ntua.flexiantManage.ManagerInterface#deploy(java.lang.String,
      java.lang.String)
    */
   public String deploy(String name, String image, int cpu, int ram) {

      PlatformHandler handler = new PlatformHandler();
      WarrantApp apps = new WarrantApp("ElaaS_platform", "apps");
      List<String> ips = new ArrayList<String>();
      String proxy = null;
      String ip = null;

      // Create deployment id
      List<Integer> s = apps.getLastItems(1, "deployment_id");
      int deployment_id = 1;
      if (!s.isEmpty())
         deployment_id = s.get(0) + 1;

      // Start threads that create the application and the proxy server
      final ExecutorService workers = Executors.newCachedThreadPool();
      Future<String> createProxy, createAppServer;

      createAppServer = workers.submit(new ScaleUpCallable(name + "_1",
          image, cpu, ram, handler));
      createProxy = workers.submit(new ScaleUpCallable(name + "_proxy", "
          Warrant_nginx", 2, 2048, handler));

      // Wait until all service requests are complete and get results
      try {
         ip = createAppServer.get();
         proxy = createProxy.get();
      } catch (InterruptedException | ExecutionException e) {
         e.printStackTrace();
      }

      // Set new ip and update database and proxy's configuration file
```

```
        ips.add(ip);
        RestClient client = new RestClient();
        client.updateProxy(proxy, ips);
        apps.deployApp(name, deployment_id, proxy, cpu, ram, image, ips);

        return ip;
    }


    /* (non-Javadoc)
     * @see gr.ntua.flexiantManage.ManagerInterface#undeploy()
     */
    public void undeploy(int deployment_id) {

        // Delete document from database
        WarrantApp apps = new WarrantApp("ElaaS_platform", "apps");
        String name = apps.getItem(deployment_id, "name");
        List<String> ips = apps.getItem(deployment_id, "ips");
        apps.deleteDocument(deployment_id);

        // Delete all application servers and proxy server
        PlatformHandler driver = new PlatformHandler();
        for (int i = 0; i < ips.size(); i++) {
            int code = i+1;
            Runnable task = new ScaleDownRunnable(name + "_" + code, driver);
            Thread worker = new Thread(task);
            worker.start();
        }
        Runnable task = new ScaleDownRunnable(name + "_proxy", driver);
        Thread worker = new Thread(task);
        worker.start();
    }


    /* (non-Javadoc)
     * @see gr.ntua.flexiantManage.ManagerInterface#scaleUp(java.lang.String,
         java.lang.String, int, int)
     */
    public List<String> scaleUp(int deployment_id, int number, String image,
        int cpu, int ram) {

        // Get current ips
        WarrantApp apps = new WarrantApp("ElaaS_platform", "apps");
        String name = apps.getItem(deployment_id, "name");
        String proxy = apps.getItem(deployment_id, "proxyIP");
        List<String> ips = apps.getItem(deployment_id, "ips");
        List<String> new_ips = new ArrayList<String>();

        PlatformHandler driver = new PlatformHandler();

        // Start threads that create new servers
        final ExecutorService workers = Executors.newCachedThreadPool();

        Collection<Callable<String>> tasks = new ArrayList<Callable<String>>()
            ;
        for (int i = 0; i < number; i++) {
            int code = ips.size() + i+1;
            tasks.add(new ScaleUpCallable(name + "_" + code, image, cpu, ram,
                driver));
        }

        // Wait until all service requests are complete and get results
```

```
        List<Future<String>> results;
        try {
            results = workers.invokeAll(tasks);
            for (Future<String> f : results)
                new_ips.add(f.get());
        } catch (InterruptedException | ExecutionException e) {
            e.printStackTrace();
        }

        // Update ips in the database
        // and proxy's configuration file calling a rest service
        ips.addAll(new_ips);
        int instances = apps.getItem(deployment_id, "instances");
        apps.updateApp(deployment_id, instances+number, cpu, ram, ips, new
            Date());

        RestClient client = new RestClient();
        client.updateProxy(proxy, ips);

        return new_ips;
    }


    /* (non-Javadoc)
     * @see gr.ntua.flexiantManage.ManagerInterface#scaleDown()
     */
    public void scaleDown(int deployment_id, int number) {

        // Get current ips
        WarrantApp apps = new WarrantApp("ElaaS_platform", "apps");
        List<String> ips = apps.getItem(deployment_id, "ips");
        String name = apps.getItem(deployment_id, "name");
        String proxy = apps.getItem(deployment_id, "proxyIP");

        // Check if possible to scale down (instances > number)
        int instances = apps.getItem(deployment_id, "instances");
        if (number >= instances)
            number = instances - 1;

        // Update proxy's configuration file
        List<String> new_ips = ips.subList(0, ips.size()-number);
        RestClient client = new RestClient();
        client.updateProxy(proxy, new_ips);

        // Update database
        int cpu = apps.getItem(deployment_id, "cpu");
        int ram = apps.getItem(deployment_id, "ram");
        apps.updateApp(deployment_id, instances - number, cpu, ram, new_ips,
            new Date());

        // Start threads that delete servers
        PlatformHandler driver = new PlatformHandler();
        for (int i = ips.size()-number; i < ips.size(); i++) {
            int code = i+1;
            Runnable task = new ScaleDownRunnable(name + "_" + code, driver);
            Thread worker = new Thread(task);
            worker.start();
        }
    }
}
```

Κλάση που στέλνει στον proxy server τη νέα λίστα διευθύνσεων IP

```
/**
 * The Class RestClient.
 */
public class RestClient {


   /**
    * Check connection.
    *
    * @param ip the ip
    * @return the int
    */
   public int checkConnection(String ip) {

     HttpURLConnection conn = null;
     int result = 0;

     try {
         URL url = new URL("http://" + ip + ":8080");
         conn = (HttpURLConnection) url.openConnection();
         conn.setRequestMethod("GET");
         conn.setDoOutput(true);
         conn.addRequestProperty("Content-Type", "text/html");
         result = conn.getResponseCode();
     } catch (MalformedURLException ex) {
         System.out.println("MalformedURLException: " + ex.getMessage());
     } catch (IOException ex) {
         System.out.println("IOException: " + ex.getMessage());
     } finally {
         if(conn != null)
        conn.disconnect();
     }
     return result;
   }


    /**
     * Update proxy.
     *
     * @param proxy the proxy
     * @param ips the ips
     */
    public void updateProxy(String proxy, List<String> ips) {

        System.out.println("Starting client...");
        System.out.println("IPs registered.\nNow making the call");
        String result = makeCall(proxy, ips);
        System.out.println(result);
        System.out.println("end");
    }


    /**
     * Make call.
     *
     * @param proxy the proxy
     * @param ips the ips
```

```
     * @return the string
     */
    private static String makeCall(String proxy, List<String> ips) {

        HttpURLConnection conn = null;
        String result = "";

        try {
=            URL url = new URL("http://" + proxy + ":8080/
    NginxExtendedServices/rest/updatenodes/");

            conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("POST");
            conn.setDoOutput(true);
            conn.addRequestProperty("Content-Type", "application/json");

            OutputStreamWriter out = new OutputStreamWriter(conn.
                getOutputStream());
            out.write(makeJsonString(ips));
            out.close();

            if ( conn.getResponseCode() != 200 ) {
                throw new RuntimeException("Failed. Http Code: " + conn.
                    getResponseCode());
            }

            InputStream inputstream = conn.getInputStream();
            result = getStringFromInputStream(inputstream);
        } catch (MalformedURLException ex) {
            System.out.println("MalformedURLException: " + ex.getMessage());
        } catch (IOException ex) {
            System.out.println("IOException: " + ex.getMessage());
        } finally {
            if(conn != null)
                conn.disconnect();
        }
        return result;
    }
}
```

Nginx Proxy Server

Κλάση που επεξεργάζεται τα log αρχεία του proxy

```
/**
 * The Class ProcessLogs.
 */
public class ProcessLogs {

    /**
     * Process logs.
     *
     * @param filename the filename
     * @param position the position
     * @param timestamp the timestamp
     * @return the cumulative monitoring data
     * @throws FileNotFoundException the file not found exception
```

```
     * @throws IOException Signals that an I/O exception has occurred.
     */
    public static CumulativeMonitoringData processLogs(String filename, long
         position, long timestamp) throws FileNotFoundException, IOException
         {

        RandomAccessFile raf = new RandomAccessFile(filename, "r");

        raf.seek(position);
        String line;
        Integer numOfBytes = 0;
        String dateString = "";
        Double averageResponseTimes = 0.0;
        int totalRows = 0;
        int numOfResponseTimes = 0;

        while((line = raf.readLine()) != null) {
         totalRows++;

            // Get date
            int index = line.indexOf("]", 1);
            dateString = line.substring(1, index);
            int index_tmp = dateString.indexOf(" ");
            dateString = dateString.substring(0, index_tmp);
            dateString = fixMonth(dateString);

            try {
                Date currentLineDate = new SimpleDateFormat("dd/MM/yyyy:HH:
                    mm:ss").parse(dateString);

                // Take care only last logs
                if(currentLineDate.getTime()<=timestamp)
                    continue;
            } catch (ParseException ex) {
                Logger.getLogger(ProcessLogs.class.getName()).log(Level.
                    SEVERE, null, ex);
                continue;
            }

            // Get total bytes sent
            line = line.substring(index + 2);
            index = line.indexOf(" ");
            String strnuByte = line.substring(0, index);
            Integer nuByte = Integer.valueOf(strnuByte);

            // If no new response time is detected
            if(nuByte == 0)
                continue;
            numOfBytes += nuByte;

            // Get average response time
            numOfResponseTimes++;
            line = line.substring(index + 1);
            index = line.indexOf(" ");
            String strResponseTime = line.substring(0, index);
            if("-".equals(strResponseTime))
                continue;
            Double currentResponseTime = Double.valueOf(strResponseTime);
            averageResponseTimes += currentResponseTime;
        }

        averageResponseTimes = averageResponseTimes/numOfResponseTimes;
```

```
        CumulativeMonitoringData result =  new CumulativeMonitoringData();
        result.setAverageResponseTimes(averageResponseTimes);
        result.setNumOfBytes(numOfBytes);
        result.setNumOfResponseTimes(numOfResponseTimes);
        result.setTimeStamp(new Date());
        result.setNumOfTotalBytesInFile(raf.length());
        result.setNumOfRows(totalRows);

        // Find public IP
        URL whatismyip = new URL("http://agentgatech.appspot.com");
        URLConnection connection = whatismyip.openConnection();
        connection.addRequestProperty("Protocol", "Http/1.1");
        connection.addRequestProperty("Connection", "keep-alive");
        connection.addRequestProperty("Keep-Alive", "1000");
        connection.addRequestProperty("User-Agent", "Web-Agent");
        BufferedReader in =  new BufferedReader(new InputStreamReader(
            connection.getInputStream()));
        String ip = in.readLine();

        // Call rest service to update the platform's mongoDB
        RestClient client = new RestClient();
        client.updateDB(ip, result.getTimeStamp(), numOfResponseTimes,
            numOfBytes, averageResponseTimes+"");

        // Print statistics to file
    PrintWriter out = new PrintWriter(new FileWriter("/var/log/nginx/
        daemon_out"), true);
        out.println(result.toString());
        out.close();
       raf.close();
        return result;
    }

}
```

Κλάση που καλεί το REST service της πλατφόρμας, για προσθήκη των στατιστικών στη βάση δεδομένων

```
/**
 * The Class RestClient.
 */
public class RestClient {

   /**
    * Update db.
    *
    * @param ip the ip
    * @param date the date
    * @param numOfResponseTimes the num of response times
    * @param numOfBytes the num of bytes
    * @param averageResponseTimes the average response times
    */
   public void updateDB(String ip, Date date, int numOfResponseTimes, int
       numOfBytes, String averageResponseTimes) {

      JSONObject json = new JSONObject();
      try {
         json.put("IP", ip);
```

```
            json.put("timeStamp", date);
            json.put("averageResponseTimes", averageResponseTimes);
            json.put("numOfResponseTimes", numOfResponseTimes);
            json.put("bytes", numOfBytes);

        } catch (JSONException e) {
            e.printStackTrace();
        }
        String result = makeCall(json, ip);
    }


    /**
     * Make call.
     *
     * @param json the json
     * @param ip the ip
     * @return the string
     */
     private static String makeCall(JSONObject json, String ip) {

         HttpURLConnection conn = null;
         String result = "";

         try{
             URL url = new URL("http://83.212.116.79:8080/updateMongoDB/rest/
                 updateDB/"+ip);

             conn = (HttpURLConnection) url.openConnection();
             conn.setRequestMethod("POST");
             conn.setDoOutput(true);
             conn.addRequestProperty("Content-Type", "application/json");

             OutputStreamWriter out = new OutputStreamWriter(conn.
                 getOutputStream());
             out.write(makeJsonString(json));
             out.close();

             if ( conn.getResponseCode() != 200 ) {
                 System.out.println("Failed. Http Code: " + conn.
                     getResponseCode());
             }

             InputStream inputstream = conn.getInputStream();
             result = getStringFromInputStream(inputstream);
         } catch (MalformedURLException ex) {
             System.out.println("MalformedURLException: " + ex.getMessage());
         } catch (IOException ex) {
             System.out.println("IOException: " + ex.getMessage());
         } finally {
             if(conn != null)
                 conn.disconnect();
         }
         return result;
     }

}
```

REST service που ανανεώνει τη λίστα διευθύνσεων IP του proxy

```
/**
 * The Class NginxUpdateNodes.
 */
public class NginxUpdateNodes {

    /**
     * Update nodes.
     *
     * @param strJson the str json
     * @return the int
     */
    public static int updateNodes(String strJson) {

        List<String> ips = getFromString(strJson);

        int result = -1;

        try {
            result = updateConfiguration(ips);
        } catch (FileNotFoundException | IOException ex) {
            Logger.getLogger(NginxUpdateNodes.class.getName()).log(Level.
                SEVERE, null, ex);
        }
        return result;
    }


    /**
     * Gets the from string.
     *
     * @param str the str
     * @return the from string
     */
    private static List<String> getFromString(String str) {

        JSONObject json;
        List<String> ips = new ArrayList<String>();

        try {
      json = new JSONObject(str);
      JSONArray jsonArray = json.getJSONArray("ips");
      for(int i=0; i<jsonArray.length(); i++) {
          ips.add(jsonArray.get(i).toString());
      }
   } catch (JSONException e) {
      e.printStackTrace();
   }
        return ips;
    }

    /**
     * Update configuration.
     *
     * @param ips the ips
     * @return the int
     * @throws FileNotFoundException the file not found exception
     * @throws IOException Signals that an I/O exception has occurred.
```

```
     */
    private static int updateConfiguration(List<String> ips) throws
        FileNotFoundException, IOException {

        BufferedReader br = new BufferedReader(new FileReader("/etc/nginx/
            nginx.conf.template"));

        try {
            StringBuilder sb = new StringBuilder();
            String line = br.readLine();

            while (line != null) {
                if(line.contains("[placeholder]")){
                    sb = addServers(ips, sb);
                } else
                    sb.append(line);
                sb.append("\n");
                line = br.readLine();
            }
            String everything = sb.toString();

            saveToFile(everything, "/etc/nginx/nginx.conf");
        } finally {
            br.close();
        }

        try {
         Runtime.getRuntime().exec("sudo /etc/init.d/nginx reload");
      } catch (IOException e) {
         e.printStackTrace();
      }

        return 0;
    }


    /**
     * Adds the servers.
     *
     * @param ips the ips
     * @param sb the sb
     * @return the string builder
     */
    private static StringBuilder addServers(List<String> ips, StringBuilder
        sb) {

        for(String ip : ips) {
            sb.append("        server  ");
            sb.append(ip);
            sb.append(":8080;");
            sb.append("\n");
        }
        return sb;
    }

}
```

Platform GUI

REST service που υπολογίζει τους απαιτούμενους πόρους για μία εφαρμογή

```
package eu._4caast.services;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.List;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
/**
 * The Class CalculateResources.
 */
@Path("/calculate")
public class CalculateResources {

   /** The uri info. */
   @Context
   UriInfo uriInfo;

   /** The request. */
   @Context
   Request request;


   /**
    * Gets the resources.
    *
    * @param uptousers the uptousers
    * @param resptime the resptime
    * @param app the app
    * @return the resources
    * @throws JAXBException the jAXB exception
    * @throws FileNotFoundException the file not found exception
    */
   @GET
   @Produces({ MediaType.APPLICATION_XML })
   @Path("/{i}/{j}/{app}")
   public List<Resolution> getResources(@PathParam("i") String uptousers,
       @PathParam("j") String resptime, @PathParam("app") String app) throws
        JAXBException, FileNotFoundException {

      String HINTS_XML = new String(HelpMethods.getHintsFile(app));
      Blueprint blueprint = null;

      // Create JAXB context and instantiate marshaller
      JAXBContext context = JAXBContext.newInstance(Blueprint.class);
      Unmarshaller um;
      um = context.createUnmarshaller();

      // Get root element from the xml file
      blueprint = (Blueprint) um.unmarshal(new FileInputStream(HINTS_XML));
```

```
        // Get the desired objects created
        List<Object> offers2 = blueprint.
            getBasicPropertiesSectionOrDeploymentArtefactSectionOrOfferingSection
            ();
        OfferingSection offers = (OfferingSection) offers2.get(1);
        Offering offering = offers.getOffering();
        List<ExtProperty> property = offering.getExtProperty();
        HintsSection hints = ((ExtProperty) property.get(0)).getHintsSection()
            ;
        Initializationrules initrules = hints.getInitializationrules();
        List<Initialization> initialization = initrules.getInitialization();

        int i = 0;
        while (i < initialization.size()) {
           List<Tunableselection> tunableselection = initialization.get(i).
               getTunableselection();
           if ( (tunableselection.get(0).getVal().intValue() == Integer.
               parseInt(uptousers)) &&
                 (tunableselection.get(1).getVal().intValue() == Integer.
                     parseInt(resptime)) )
              break;
           i++;
        }
        Resolutions resolutions = initialization.get(i).getResolutions();
        List<Resolution> resolution = resolutions.getResolution();
        return resolution;
    }

}
```

REST service που προσθέτει κανόνες στο αρχείο ρυθμίσεων μίας εφαρμογής

```
/**
 * The Class CreateRules.
 *
 */
@Path("/hints")
public class CreateRules {

    /** The uri info. */
    @Context
    UriInfo uriInfo;

    /** The request. */
    @Context
    Request request;


    /**
     * Creates the rule.
     *
     * @param uval the uval
     * @param rval the rval
     * @param resource the resource
     * @param product the product
     * @param inst the inst
     * @param cpu_req the cpu_req
```

```
     * @param ram_req the ram_req
     * @param app the app
     * @return the resolutions
     * @throws ParserConfigurationException the parser configuration
         exception
     * @throws SAXException the sAX exception
     * @throws IOException Signals that an I/O exception has occurred.
     * @throws JAXBException the jAXB exception
     */
    @GET
    @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
    @Path("/{uval}/{rval}/{resource}/{product}/{inst}/{cpu_req}/{ram_req}/{
        app}")
    public Resolutions createRule(@PathParam("uval") String uval, @PathParam(
        "rval") String rval, @PathParam("resource") String resource,
        @PathParam("product") String product, @PathParam("inst") String inst,
         @PathParam("cpu_req") String cpu_req, @PathParam("ram_req") String
        ram_req, @PathParam("app") String app) throws
        ParserConfigurationException, SAXException, IOException,
        JAXBException {

        String HINTS_XML = new String(HelpMethods.getHintsFile(app));
        String delims = ",";
        String[] resource_name = resource.split(delims);
        String[] product_name = product.split(delims);
        String[] instances = inst.split(delims);
        String[] cpu = cpu_req.split(delims);
        String[] ram = ram_req.split(delims);

        // Create JAXB context and instantiate unmarshaller
        JAXBContext context = JAXBContext.newInstance(Blueprint.class);
        Unmarshaller um;
        um = context.createUnmarshaller();

        // Get root element from the xml file
        Blueprint blueprint = (Blueprint) um.unmarshal(new FileInputStream(
            HINTS_XML));

        // Get the desired objects
        List<Object> offersList = blueprint.
            getBasicPropertiesSectionOrDeploymentArtefactSectionOrOfferingSection
            ();
        OfferingSection offers = (OfferingSection) offersList.get(1);
        Offering offering = offers.getOffering();
        List<ExtProperty> property = offering.getExtProperty();
        HintsSection hints = property.get(0).getHintsSection();
        Initializationrules initializationRules = hints.getInitializationrules
            ();

        // If initializationRules is null, then no rule has been created
        if (initializationRules == null ) {
           initializationRules = new Initializationrules();
        }
        List<Initialization> ruleList = initializationRules.getInitialization
            ();

        // Create new rule
        Initialization rule = new Initialization();

        // Create tunableselections
        List<Tunableselection> tunableselection = rule.getTunableselection();
        Tunableselection tunable = new Tunableselection();
```

```
        tunable.setId(new BigInteger("1"));
        tunable.setVal(new BigInteger(uval));
        tunableselection.add(tunable);

        tunable = new Tunableselection();
        tunable.setId(new BigInteger("2"));
        tunable.setVal(new BigInteger(rval));
        tunableselection.add(tunable);

        // Create the resolutions
        Resolutions resolutions = new Resolutions();
        for (int i = 0; i < 2; i++) {
            List<Resolution> resolutionList = resolutions.getResolution();
            Resolution res = new Resolution();
            res.setResourceName(resource_name[i]);
            res.setProductName(product_name[i]);
            res.setInstances(new BigInteger(instances[i]));
            res.setCpu(new BigDecimal(cpu[i]));
            res.setRam(new BigInteger(ram[i]));
            resolutionList.add(res);
        }

        // Add new rule to the ruleList
        rule.setResolutions(resolutions);
        ruleList.add(rule);

        // Assign the initialization rules to the HintsSection
        hints.setInitializationrules(initializationRules);

        // Instantiate marshaller and write to file
         Marshaller m = context.createMarshaller();
         m.marshal(blueprint, new File(HINTS_XML));

         return resolutions;
    }

}
```

REST service που επιστρέφει δεδομένα από το αρχείο ρυθμίσεων μίας εφαρμογής

```
/**
 * The Class GetXmlData.
 */
@Path("/getXml")
public class GetXmlData {

    /** The uri info. */
    @Context
    UriInfo uriInfo;

    /** The request. */
    @Context
    Request request;


    /**
     * Gets the property.
```

```
     *
     * @param app the app
     * @return the property
     * @throws ParserConfigurationException the parser configuration
         exception
     * @throws SAXException the sAX exception
     * @throws IOException Signals that an I/O exception has occurred.
     * @throws JAXBException the jAXB exception
     */
    @GET
    @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
    @Path("/{app}")
    public List<ExtProperty> getProperty(@PathParam("app") String app) throws
         ParserConfigurationException, SAXException, IOException,
        JAXBException {

        String HINTS_XML = HelpMethods.getHintsFile(app);

        // Create JAXB context and instantiate marshaller
        JAXBContext context = JAXBContext.newInstance(Blueprint.class);
        Unmarshaller um;
        um = context.createUnmarshaller();

        // Get root element from the xml file
        Blueprint blueprint = (Blueprint) um.unmarshal(new FileInputStream(
            HINTS_XML));

        // Get the exterior property
        List<Object> offersList = blueprint.
            getBasicPropertiesSectionOrDeploymentArtefactSectionOrOfferingSection
            ();
        OfferingSection offers = (OfferingSection) offersList.get(1);
        Offering offering = offers.getOffering();
        return offering.getExtProperty();
    }


    /**
     * Gets the artefact.
     *
     * @param app the app
     * @return the artefact
     * @throws ParserConfigurationException the parser configuration
         exception
     * @throws SAXException the sAX exception
     * @throws IOException Signals that an I/O exception has occurred.
     * @throws JAXBException the jAXB exception
     */
    @GET
    @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
    @Path("/artefact/{app}")
    public DeploymentArtefactSection getArtefact(@PathParam("app") String app
        ) throws ParserConfigurationException, SAXException, IOException,
        JAXBException {

        String HINTS_XML = HelpMethods.getHintsFile(app);

        // Create JAXB context and instantiate marshaller
        JAXBContext context = JAXBContext.newInstance(Blueprint.class);
        Unmarshaller um;
        um = context.createUnmarshaller();
```

```
        // Get root element from the xml file
        Blueprint blueprint = (Blueprint) um.unmarshal(new FileInputStream(
            HINTS_XML));

        // Get deployment artefact section
        List<Object> offersList = blueprint.
            getBasicPropertiesSectionOrDeploymentArtefactSectionOrOfferingSection
            ();
        DeploymentArtefactSection artefacts = (DeploymentArtefactSection)
            offersList.get(2);

        return artefacts;
    }


    /**
     * Gets the tunables.
     *
     * @param app the app
     * @return the tunables
     * @throws ParserConfigurationException the parser configuration
       exception
     * @throws SAXException the sAX exception
     * @throws IOException Signals that an I/O exception has occurred.
     * @throws JAXBException the jAXB exception
     */
    @GET
    @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
    @Path("/tunables/{app}")
    public List<Tunable> getTunables(@PathParam("app") String app) throws
        ParserConfigurationException, SAXException, IOException,
        JAXBException {

        List<ExtProperty> property = getProperty(app);
        HintsSection hints = ((ExtProperty) property.get(0)).getHintsSection()
            ;
        Tunables tunables = hints.getTunables();
        return tunables.getTunable();
    }


    /**
     * Checks if is tunables empty.
     *
     * @param app the app
     * @return the string
     * @throws ParserConfigurationException the parser configuration
       exception
     * @throws SAXException the sAX exception
     * @throws IOException Signals that an I/O exception has occurred.
     * @throws JAXBException the jAXB exception
     */
    @GET
    @Produces({ MediaType.TEXT_PLAIN })
    @Path("/size/{app}")
    public String isTunablesEmpty(@PathParam("app") String app) throws
        ParserConfigurationException, SAXException, IOException,
        JAXBException {

        List<ExtProperty> property = getProperty(app);
        HintsSection hints = ((ExtProperty) property.get(0)).getHintsSection()
            ;
```

```
    Tunables tunables = hints.getTunables();

    if ( (tunables.getTunable().isEmpty()) || (getTunables(app).size() ==
        0) )
       return "true";
    else
       return "false";
}


/**
 * Gets the tunable.
 *
 * @param i the i
 * @param app the app
 * @return the tunable
 * @throws ParserConfigurationException the parser configuration
     exception
 * @throws SAXException the sAX exception
 * @throws IOException Signals that an I/O exception has occurred.
 * @throws JAXBException the jAXB exception
 */
@GET
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
@Path("/tunables/{i}/{app}")
public List<Tunableval> getTunable(@PathParam("i") int i, @PathParam("app
    ") String app) throws ParserConfigurationException, SAXException,
    IOException, JAXBException {

   return getTunables(app).get(i).getTunableval();
}


/**
 * Checks if is tunable empty.
 *
 * @param i the i
 * @param app the app
 * @return the string
 * @throws ParserConfigurationException the parser configuration
     exception
 * @throws SAXException the sAX exception
 * @throws IOException Signals that an I/O exception has occurred.
 * @throws JAXBException the jAXB exception
 */
@GET
@Produces({ MediaType.TEXT_PLAIN })
@Path("/size/{i}/{app}")
public String isTunableEmpty(@PathParam("i") int i, @PathParam("app")
    String app) throws ParserConfigurationException, SAXException,
    IOException, JAXBException {

   if ( (isTunablesEmpty(app).equals("true")) || (getTunable(i, app).size
       () == 0) )
      return "true";
   else
      return "false";
}

/**
 * Gets the tunable val.
 *
```

```
     * @param i the i
     * @param j the j
     * @param app the app
     * @return the tunable val
     * @throws ParserConfigurationException the parser configuration
          exception
     * @throws SAXException the sAX exception
     * @throws IOException Signals that an I/O exception has occurred.
     * @throws JAXBException the jAXB exception
     */
    @GET
    @Produces({ MediaType.TEXT_PLAIN })
    @Path("/tunables/{i}/{j}/{app}")
    public String getTunableVal(@PathParam("i") int i, @PathParam("j") int j,
         @PathParam("app") String app) throws ParserConfigurationException,
        SAXException, IOException, JAXBException {

       return ""+getTunable(i, app).get(j).getDesc();
    }

}
```

Database

Κλάση που διαχειρίζεται τις συλλογές με τα στατιστικά των εφαρμογών

```
/**
 * The Class Statistics.
 */
public class Statistics extends Database {


    /** The collection. */
    private String collection;


    /**
     * Instantiates a new statistics.
     *
     * @param database the database
     * @param collection the collection
     */
    public Statistics(String database, String collection) {

       super(database);
       this.collection = collection;
    }


    /**
     * Save document from json.
     *
     * @param json the json
     * @return the int
     */
```

```
public int saveDocumentFromJson(String json) {

    // Save new document
    JSONObject dbObject = null;
    try {
        dbObject = new JSONObject(json);
    } catch (JSONException e1) {
        e1.printStackTrace();
    }

    // Get items and
    // convert timeStamp from String type to Date
    SimpleDateFormat df = new SimpleDateFormat("EEE MMM dd HH:mm:ss z yyyy
        ");

    String ip = null;
    String calling = null;
    Date timeStamp = null;
    int numOfResponseTimes = 0;
    int numOfBytes = 0;
    String averageResponseTimes = null;
    try {
        timeStamp = df.parse((String) dbObject.get("timeStamp"));
        ip = (String) dbObject.get("IP");
        calling = (String) dbObject.get("calling");
        numOfResponseTimes = (Integer) dbObject.get("numOfResponseTimes");
        numOfBytes = (Integer) dbObject.get("bytes");
        averageResponseTimes = (String) dbObject.get("averageResponseTimes"
            );
    } catch (JSONException e1) {
        e1.printStackTrace();
    } catch (ParseException e) {
        e.printStackTrace();
    }

    // Insert the document in the database
    return saveDocument(calling, ip, timeStamp, numOfResponseTimes,
        numOfBytes, averageResponseTimes);
}


/**
 * Save document.
 *
 * @param calling the calling
 * @param ip the ip
 * @param timeStamp the time stamp
 * @param numOfResponseTimes the num of response times
 * @param numOfBytes the num of bytes
 * @param averageResponseTimes the average response times
 * @return the int
 */
public int saveDocument(String calling, String ip, Date timeStamp, int
    numOfResponseTimes, int numOfBytes, String averageResponseTimes) {

    // Get collection
    MongoClient client = getClient();
    DB db = client.getDB(database);
    DBCollection table = db.getCollection(collection);

    // Save
    BasicDBObject document = new BasicDBObject();
```

```
        document.put("timeStamp", timeStamp);
        document.put("averageResponseTimes", averageResponseTimes);
        document.put("numOfResponseTimes", numOfResponseTimes);
        document.put("bytes", numOfBytes);
        document.put("calling", calling);
        document.put("IP", ip);
        WriteResult result = table.insert(document);
        table.ensureIndex(new BasicDBObject("timeStamp", 1), new BasicDBObject
            ("expireAfterSeconds", 10000));
        client.close();

        // Get action result
        CommandResult rs = result.getLastError();
          if (rs.getErrorMessage() == null)
           return 0;
          else
           return 1;
    }


    /**
     * Gets the documents after timestamp.
     *
     * @param timeStamp the time stamp
     * @return the documents after timestamp
     */
    public DBCursor getDocumentsAfterTimestamp(Date timeStamp) {

        // Get collection
        MongoClient client = getClient();
        DB db = client.getDB(database);
        DBCollection table = db.getCollection(collection);

        // Make query and get Documents
        BasicDBObject query = new BasicDBObject();
        query.put("timeStamp", BasicDBObjectBuilder.start("$gte", timeStamp).
            get());
        DBCursor cursor = table.find(query).sort(new BasicDBObject("timeStamp"
            , -1));

        client.close();
        return cursor;
    }


    /**
     * Gets the last items.
     *
     * @param <T> the generic type
     * @param number the number
     * @param item the item
     * @return the last items
     */
    public <T> List<T> getLastItems(int number, String item) {

        return super.getLastItems(number, item, "timeStamp", collection);
    }


    /**
     * Drop table.
     */
```

```
    public void dropTable() {

        super.dropTable(collection);
    }


    /**
     * Delete document.
     *
     * @param key_value the key_value
     */
    public void deleteDocument(String key_value) {

        super.deleteDocument("timeStamp", key_value, collection);
    }
}
```

Κλάση που διαχειρίζεται τη συλλογή με τα στοιχεία των εφαρμογών

```
/**
 * The Class WarrantApp.
 */
public class WarrantApp extends Database {

    /** The collection. */
    private String collection;


    /**
     * Instantiates a new warrant app.
     *
     * @param database the database
     * @param collection the collection
     */
    public WarrantApp(String database, String collection) {

        super(database);
        this.collection = collection;
    }


    /**
     * Update app.
     *
     * @param deployment_id the deployment_id
     * @param instances the instances
     * @param cpu the cpu
     * @param ram the ram
     * @param ips the ips
     * @param scalePoint the scalepoint
     * @return the int
     */
    public int updateApp(int deployment_id, int instances, int cpu, int ram,
        List<String> ips, Date scalepoint) {

        // Get collection
        MongoClient client = getClient();
```

```
        DB db = client.getDB(database);
        DBCollection table = db.getCollection(collection);

        // Update document
        BasicDBObject newDocument = new BasicDBObject();
        BasicDBObject fieldSets = new BasicDBObject();
        fieldSets.put("instances", instances);
        fieldSets.put("cpu", cpu);
        fieldSets.put("ram", ram);
        fieldSets.put("ips", ips);
        fieldSets.put("scalepoint", scalepoint);
        newDocument.put("$set", fieldSets);

        BasicDBObject searchQuery = new BasicDBObject().append("deployment_id"
            , deployment_id);

        WriteResult result = table.update(searchQuery, newDocument);
        client.close();

        // Get action result
        CommandResult rs = result.getLastError();
        if (rs.getErrorMessage() == null)
            return 0;
        else
            return 1;
    }


    /**
     * Deploy app.
     *
     * @param name the name
     * @param deployment_id the deployment_id
     * @param proxyIp the proxy ip
     * @param min_cpu the minimum cpu
     * @param min_ram the minimum ram
     * @param image the image
     * @param ips the ips
     * @return the int
     */
    public int deployApp(String name, int deployment_id, String proxyIp, int
        min_cpu, int min_ram, String image, List<String> ips) {

        // Get collection
        MongoClient client = getClient();
        DB db = client.getDB(database);
        DBCollection table = db.getCollection(collection);

        // Save
        BasicDBObject document = new BasicDBObject();
        document.put("name", name);
        document.put("deployment_id", deployment_id);
        document.put("proxyIP", proxyIp);
        document.put("instances", 1);
        document.put("cpu", min_cpu);
        document.put("ram", min_ram);
        document.put("image", image);
        document.put("ips", ips);
        document.put("scalepoint", new Date());

        WriteResult result = table.insert(document);
        client.close();
```

```
        // Get action result
        CommandResult rs = result.getLastError();
        if (rs.getErrorMessage() == null)
            return 0;
        else
            return 1;
    }


    /**
     * Gets the apps.
     *
     * @return the apps
     */
    public List<String> getApps() {

        return super.getLastItems(0, "name", "name", collection);
    }


    /**
     * Gets the item.
     *
     * @param <T> the generic type
     * @param key the key
     * @param item the item
     * @return the item
     */
    public <T> T getItem(int key, String item) {

        return super.getItem("deployment_id", key, item, collection);
    }


    /**
     * Gets the item.
     *
     * @param <T> the generic type
     * @param key the key
     * @param item the item
     * @return the item
     */
    public <T> T getItemFromProxy(String key, String item) {

        return super.getItem("proxyIP", key, item, collection);
    }


    /**
     * Gets the id.
     *
     * @param key the key
     * @return the id
     */
    public int getId(String key) {

        return super.getItem("name", key, "deployment_id", collection);
    }


    /**
```

```
     * Drop table.
     */
    public void dropTable() {

        super.dropTable(collection);
    }


    /**
     * Delete document.
     *
     * @param key_value the key_value
     */
    public void deleteDocument(int key_value) {

        super.deleteDocument("deployment_id", key_value, collection);
    }

    /**
     * Gets the last items.
     *
     * @param <T> the generic type
     * @param number the number
     * @param item the item
     * @return the last items
     */
    public <T> List<T> getLastItems(int number, String item) {

        return super.getLastItems(number, item, "deployment_id", collection);
    }

}
```

REST service που προσθέτει στη βάση δεδομένων τα νέα στατιστικά των εφαρμογών

```
/**
 * REST Web Service.
 */
@Path("updateDB/{ip}")
public class UpdateDB {

    /** The context. */
     @Context
     private UriInfo context;


     /**
      * Put json.
      *
      * @param ip the ip
      * @param content the content
      * @return the string
      */
     @POST
     @Consumes("application/json")
     @Produces(MediaType.APPLICATION_JSON)
     public String putJson(@PathParam("ip") String ip, String content) {
```

```
        String jsonResponse = null;

        // Save document to database
        WarrantApp apps = new WarrantApp("ElaaS_platform", "apps");
        String name = apps.getItemFromProxy(ip, "name");

        Statistics DB = new Statistics("ElaaS_platform", name + ".stats");
        int result = DB.saveDocumentFromJson(content);

        // Get output json
          if (result == 0)
              jsonResponse = createJson("success");
          else
              jsonResponse = createJson("fail");

          return jsonResponse;
      }

}
```

Σημαντικά αρχεία

Αρχείο ρυθμίσεων του proxy server

```
# For more information on configuration, see:
#   * Official English Documentation: http://nginx.org/en/docs/
#   * Official Russian Documentation: http://nginx.org/ru/docs/

user              root;
worker_processes  1;

error_log  /var/log/nginx/error.log;
#error_log  /var/log/nginx/error.log  notice;
#error_log  /var/log/nginx/error.log  info;

pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    log_format timed_combined '$remote_addr - $remote_user [$time_local] '
                              '"$request" $status $body_bytes_sent '
                              '"$http_referer" "$http_user_agent" '
                              '$request_time $upstream_response_time $pipe';

    log_format custom_monitoring '[$time_local] $body_bytes_sent $
        upstream_response_time "$remote_addr" "$request" "$http_referer" "$
        http_user_agent" "$http_referer" $request_time $pipe';


    access_log  /var/log/nginx/access.log  custom_monitoring;

    sendfile        on;
    #tcp_nopush     on;

    #keepalive_timeout  0;
    keepalive_timeout  65;
```

```
    #gzip  on;

    # Load config files from the /etc/nginx/conf.d directory
    # The default server is in conf.d/default.conf
    include /etc/nginx/conf.d/*.conf;


    upstream warrantApplication {
       [placeholder]
    }


    server {
       listen 80;
       server_name www.loadbalancer.com;

       location / {
          proxy_pass http://warrantApplication;
       }

    }


}
```

Τμήμα του αρχείου ρυθμίσεων της εφαρμογής flipper, με τους κανόνες και τους απαιτούμενους πόρους

```
    <bp:offering_section>
    <bp:offering>
      <bp:offering_id>Resolved-WarrantManagement-Public</bp:offering_id>
      <bp:resource_name>WarrantManagement Application Public Part</
          bp:resource_name>
      <bp:structural_interface>http://localhost:8080/WarrantManagement.../?
          wsdl</bp:structural_interface>
      <bp:endpoint_location>http://localhost:8080/WarrantManagement.../</
          bp:endpoint_location>
      <bp:range_of_instance>
         <bp:minimum>1</bp:minimum>
         <bp:maximum>9999</bp:maximum>
      </bp:range_of_instance>
     <!-- EXTENSION with hints and initialization rules (provided by NTUA)
         -->
      <bp:ext_property>
         <bp:p_name>hints and initialization rules</bp:p_name>
         <hi:hints_section xmlns:hi="http://www.4caast.eu/hints">
            <hi:tunables>
               <hi:tunable id="1" desc="uptousers">
                  <hi:tunableval desc="100"/>
                  <hi:tunableval desc="500"/>
               </hi:tunable>
               <hi:tunable id="2" desc="resptime">
                  <hi:tunableval desc="1.0"/>
                  <hi:tunableval desc="0.5"/>
               </hi:tunable>
            </hi:tunables>
```

```
<hi:initializationrules>
   <hi:initialization id="1">
      <hi:tunableselection id="1" val="1"/>
      <hi:tunableselection id="2" val="1"/>
      <hi:resolutions>
         <hi:resolution>
            <hi:resource_name>servlet v2.5 container</
                hi:resource_name>
            <hi:product_name>tomcat6_linux_rpm</hi:product_name>
            <hi:instances>-1</hi:instances>
            <hi:cpu>0.5</hi:cpu>
            <hi:ram>1024</hi:ram>
         </hi:resolution>
         <hi:resolution>
            <hi:resource_name>sql_database</hi:resource_name>
            <hi:product_name>postgres-9.1.3-1.x86_64_rpm</
                hi:product_name>
            <hi:instances>-1</hi:instances>
            <hi:cpu>0.5</hi:cpu>
            <hi:ram>1024</hi:ram>
         </hi:resolution>
      </hi:resolutions>
   </hi:initialization>
   <hi:initialization id="2">
      <hi:tunableselection id="1" val="1"/>
      <hi:tunableselection id="2" val="2"/>
      <hi:resolutions>
         <hi:resolution>
            <hi:resource_name>servlet v2.5 container</
                hi:resource_name>
            <hi:product_name>tomcat6_linux_rpm</hi:product_name>
            <hi:instances>-1</hi:instances>
            <hi:cpu>2</hi:cpu>
            <hi:ram>4096</hi:ram>
         </hi:resolution>
         <hi:resolution>
            <hi:resource_name>sql_database</hi:resource_name>
            <hi:product_name>postgres-9.1.3-1.x86_64_rpm</
                hi:product_name>
            <hi:instances>-1</hi:instances>
            <hi:cpu>2</hi:cpu>
            <hi:ram>4096</hi:ram>
         </hi:resolution>
      </hi:resolutions>
   </hi:initialization>
   <hi:initialization id="3">
      <hi:tunableselection id="1" val="2"/>
      <hi:tunableselection id="2" val="1"/>
      <hi:resolutions>
         <hi:resolution>
            <hi:resource_name>servlet v2.5 container</
                hi:resource_name>
            <hi:product_name>tomcat6_linux_rpm</hi:product_name>
            <hi:instances>-1</hi:instances>
            <hi:cpu>2</hi:cpu>
            <hi:ram>2048</hi:ram>
         </hi:resolution>
         <hi:resolution>
            <hi:resource_name>sql_database</hi:resource_name>
            <hi:product_name>postgres-9.1.3-1.x86_64_rpm</
                hi:product_name>
            <hi:instances>-1</hi:instances>
```

```
                    <hi:cpu>1</hi:cpu>
                    <hi:ram>1024</hi:ram>
                 </hi:resolution>
              </hi:resolutions>
           </hi:initialization>
           <hi:initialization id="4">
              <hi:tunableselection id="1" val="2"/>
              <hi:tunableselection id="2" val="2"/>
              <hi:resolutions>
                 <hi:resolution>
                    <hi:resource_name>servlet v2.5 container</
                        hi:resource_name>
                    <hi:product_name>tomcat6_linux_rpm</hi:product_name>
                    <hi:instances>-1</hi:instances>
                    <hi:cpu>3</hi:cpu>
                    <hi:ram>6144</hi:ram>
                 </hi:resolution>
                 <hi:resolution>
                    <hi:resource_name>sql_database</hi:resource_name>
                    <hi:product_name>postgres-9.1.3-1.x86_64_rpm</
                        hi:product_name>
                    <hi:instances>-1</hi:instances>
                    <hi:cpu>3</hi:cpu>
                    <hi:ram>6144</hi:ram>
                 </hi:resolution>
              </hi:resolutions>
           </hi:initialization>
        </hi:initializationrules>

     </hi:hints_section>
  </bp:ext_property>
  <!-- end of EXTENSION -->
 </bp:offering>
 </bp:offering_section>
```