ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΙΚΡΟΫΠΗΡΕΣΙΕΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΟ ΝΕΦΟΣ: ΜΙΑ ΤΕΧΝΟΟΙΚΟΝΟΜΙΚΗ ΠΡΟΣΕΓΓΙΣΗ

Διπλωματική Εργασία

του

Ιωάννη Παπακωνσταντίνου

Θεσσαλονίκη, Ιούνιος 2020

ΜΙΚΡΟΫΠΗΡΕΣΙΕΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΟ ΝΕΦΟΣ: ΜΙΑ ΤΕΧΝΟΟΙΚΟΝΟΜΙΚΗ ΠΡΟΣΕΓΓΙΣΗ

Ιωάννη Παπακωνσταντίνου

Πτυχίο Λογιστική & Χρηματοοικονομικής, Πα.Μακ., 2017

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Μαμάτας Ελευθέριος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 30/06/2020

Χατζηγεωργίου Αλέξανδρος	Φούσκας Κωνσταντίνος
.....	.....

Ιωάννης Παπακωνσταντίνου

.....

Περίληψη

Η συνεχόμενη αύξηση της εξάρτησης των ανθρώπων γύρω από τις ηλεκτρονικές συσκευές έχει οδηγήσει στην εμφάνιση ολοένα και περισσότερο απαιτητικών εφαρμογών, με όρους απαιτούμενων υπολογιστικών πόρων και προσαρμογής σε δυναμικές δικτυακές συνθήκες, λειτουργία σε μεγάλη κλίμακα. Χαρακτηριστικό παράδειγμα αποτελούν οι εφαρμογές αναμετάδοσης βίντεο (streaming) αλλά και άλλου είδους υπηρεσιών, όπως είναι το Netflix και το Uber. Η αύξηση των αιτημάτων από τους χρήστες προς τις διάφορες υπηρεσίες, αλλά και οι απαιτητικές εφαρμογές, έχουν δημιουργήσει την ανάγκη καλύτερου σχεδιασμού αυτών με σκοπό την πλήρη αξιοποίηση των υπολογιστικών πόρων.

Το παραπάνω πρόβλημα προσπαθεί να λύσει η αρχιτεκτονική των μικροϋπηρεσιών (microservices). Η διπλωματική εργασία εξετάζει πως η αρχιτεκτονική των μικροϋπηρεσιών μπορεί να αξιοποιηθεί κατάλληλα, ώστε να υπάρξει σωστή χρήση πόρων, τόσο σε τεχνικό όσο και σε οικονομικό επίπεδο. Η κύρια προσέγγιση της παραπάνω αρχιτεκτονικής είναι η δημιουργία μιας εφαρμογής, η οποία θα αποτελείται από πλήθος άλλων μικρότερων υπηρεσιών. Αυτές οι μεμονωμένες μικροϋπηρεσίες αναπτύσσονται ανεξάρτητα η μία από την άλλη και επικοινωνούν μεταξύ τους μέσω μιας διεπαφής προγράμματος εφαρμογής (API).

Μελετήθηκαν οι μικροϋπηρεσίες από θεωρητικής σκοπιάς ως προς το τι έχουν να προσφέρουν τόσο στο χρήστη όσο και στον δημιουργό της εφαρμογής, αλλά και στο που χωλαίνει αυτού του είδους η προσέγγιση. Για να υποστηριχθεί αυτή η θεωρητική μελέτη αναπτύχθηκε μια εφαρμογή η οποία αποτελείται από μικρότερες υπηρεσίες, δηλαδή αναπτύχθηκε με την λογική των μικροϋπηρεσιών, αλλά δημιουργήθηκε και με τη μονολιθική προσέγγιση με σκοπό να πραγματοποιηθεί η καλύτερη δυνατή σύγκριση ανάμεσα στις δύο αρχιτεκτονικές προσεγγίσεις.

Η ανάπτυξη των εφαρμογών έλαβε χώρα στο περιβάλλον Docker. Κάθε υπηρεσία αναπτύσσεται στον δικό της υποδοχέα (container) και είναι πλήρως αυτόνομη και ανεξάρτητη από τις υπόλοιπες. Οι υποδοχείς μπορούν εύκολα να δημιουργήσουν αντίγραφα του εαυτού τους, μέσω μια εικόνας που τους έχει οριστεί, με σκοπό την εξυπηρέτηση περισσότερων αιτημάτων. Για την δημιουργία και την ενορχήστρωση αυτών χρησιμοποιήθηκε το σμήνος docker (docker swarm).

Η έρευνα προχωράει ένα βήμα παρακάτω και προτείνει μια διαφορετική προσέγγιση ως προς το πως να επέρχεται αύξηση των πόρων στις μικροϋπηρεσίες, δηλαδή προτείνεται ένας διαφορετικός τρόπος επέκτασης των υποδοχέων. Η λογική που εξετάζεται είναι ότι η δημιουργία νέου υποδοχέα δε πρέπει να συμβαίνει με βάση την χρήση του επεξεργαστή αλλά με βάση το χρόνο των απαντήσεων των αιτημάτων.

Πραγματοποιήθηκαν συγκρίσεις ανάμεσα στην μονολιθική προσέγγιση και στην αρχιτεκτονική των μικροϋπηρεσιών, σε συνδυασμό με τον νέο τρόπο κλιμάκωσης, με σκοπό τη μέτρηση των πόρων που μπορούν να εξοικονομηθούν ή να χρησιμοποιηθούν διαφορετικά. Τα αποτελέσματα που προέκυψαν ήταν πολύ ενθαρρυντικά, διότι χωρίς να μειώνεται σημαντικά η μέση ποιότητα εξυπηρέτησης, κατέστη εφικτό να εξοικονομηθεί μεγάλο ποσοστό υπολογιστικών πόρων. Τέλος, υπάρχει μια οικονομική ανάλυση με σκοπό να αποτυπωθεί και από οικονομικής σκοπιάς το αντίκτυπο της χρήσης των μικροϋπηρεσιών σε συνδυασμό με τη διαφορετική λογική στην επεκτασιμότητα των υποδοχέων.

Λέξεις Κλειδιά: Μικροϋπηρεσίες, Μονολιθική αρχιτεκτονική, Τεχνοοικονομική ανάλυση, Υπολογιστικό νέφος, Μέθοδοι κλιμάκωσης υποδοχέων

Πρόλογος – Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Ελευθέριο Μαμάτα, ο οποίος μέσω των μαθημάτων του, μου όξυνε την περιέργεια στο να ασχοληθώ και να μάθω για την αρχιτεκτονική των μικροϋπηρεσιών και των υποψήφιο διδάκτωρ κ. Σαράντη Καλαφατίδη για τη πολύτιμη βοήθεια του στην εκπόνηση αυτής της διατριβής.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κατάλογος Διαγραμμάτων

1 Εισαγωγή

1.1 Πρόβλημα – Σημαντικότητα του θέματος

Με τα χρόνια η συνεχής ανάπτυξη μεγάλων μονολιθικών εφαρμογών έχει δημιουργήσει την ανάγκη της αποδόμησης αυτών σε μικρότερες μονάδες, οι οποίες θα συνεργάζονται μεταξύ τους και θα προσφέρουν ένα ενιαίο αποτέλεσμα. Αυτή η προσέγγιση ονομάζεται αρχιτεκτονική των μικροϋπηρεσιών και τα δύο κύρια πλεονεκτήματα που προσφέρει είναι η επεκτασιμότητα, η οποία οδηγεί σε αποδοτικότερη χρήση των πόρων και η γρηγορότερη ανάπτυξη καινούριων χαρακτηριστικών, σε μεγάλης κλίμακας εφαρμογές. Η μελέτη των μικροϋπηρεσιών ελκύει το ενδιαφέρον στους προγραμματιστές, διότι τους ίδιους ή λιγότερους υπολογιστικούς πόρους, μπορούν να τους εκμεταλλευτούν πιο αποδοτικά σε σχέση με μια μονολιθική προσέγγιση. Επιπρόσθετα, η γρήγορη και μη κοστοβόρα δημιουργία αντιγράφων υποδοχέων καθιστά εύκολη την διαχείριση της ελαστικότητας των υποδοχέων, με αποτέλεσμα να προσφέρεται η δυνατότητα αυξομείωσης υπολογιστικών πόρων, με σκοπό την αποτελεσματικότερη διαχείρισή τους.

Φυσικά, οι μικροϋπηρεσίες δεν αποτελούν πανάκεια στη ανάπτυξη κάθε πιθανής εφαρμογής. Ένα μεγάλο πρόβλημα των μικροϋπηρεσιών αποτελεί το γεγονός ότι η δημιουργία και η ανάπτυξη εφαρμογών, μέσω της αρχιτεκτονικής των μικροϋπηρεσιών, είναι κατά κύριο λόγο πολύπλοκη και χρονοβόρα. Επιπλέον, υπάρχει σημαντική καθυστέρηση στο συγχρονισμό των δεδομένων μεταξύ των υποδοχέων, καθώς το κάθε αντίγραφο διατηρεί τη δική του βάση δεδομένων. Αυτό, καθιστά την αρχιτεκτονική των μικροϋπηρεσιών σχεδόν απαγορευτική για ανάπτυξη υπηρεσιών, οι οποίες έχουν να κάνουν με συναλλαγές και απαιτούν χαμηλό χρόνο ανταπόκρισης. Παραδείγματα τέτοιων υπηρεσιών υπάρχουν σε τραπεζικά συστήματα ή σε ανταγωνιστικά παιχνίδια, όπως είναι το e-banking της Εθνικής Τράπεζας ή το Counter Strike: Global Offensive αντίστοιχα.

1.2 Σκοπός – Στόχοι

Ο σκοπός αυτής της μελέτης είναι να εξάγει τεχνικά και οικονομικά συμπεράσματα ως προς την χρήση των πόρων στην αρχιτεκτονική των μικροϋπηρεσιών σε σύγκριση με την μονολιθική προσέγγιση. Αρχικά, πραγματοποιήθηκε έρευνα γύρω από το θεωρητικό υπόβαθρο των δύο αρχιτεκτονικών με σκοπό την εύρεση πλεονεκτημάτων και μειονεκτημάτων. Έπειτα, αναπτύχθηκε η μονολιθική εφαρμογή με θέμα την αποθήκευση, την εύρεση και την παρουσίαση τίτλων από ταινίες και βιβλία εισαγόμενα από το χρήστη, ενώ παράλληλα η ίδια εφαρμογή αναπτύχθηκε και μέσω της αρχιτεκτονικής των μικροϋπηρεσιών. Ωστόσο, η έρευνα

επικεντρώνεται σε μια τεχνική και οικονομική ανάλυση μέσω δοκιμών και συγκρίσεων ανάμεσα στις δύο παραπάνω εφαρμογές-αρχιτεκτονικές. Στόχος αποτελεί, η εξαγωγή συμπερασμάτων ως προς την εξοικονόμηση υπολογιστικών πόρων μιας εφαρμογής, η οποία λειτουργεί, υπό το πρίσμα της αρχιτεκτονικής των μικροϋπηρεσιών, αλλά και η πρόταση μιας νέας μεθόδου κλιμάκωσης των υποδοχέων, με σκοπό την μείωση των υπολογιστικών πόρων που βρίσκονται σε χρήση. Αυτό το νέο μοντέλο επεκτασιμότητας των υποδοχέων βασίζεται στον χρόνο ανταπόκρισης των αιτημάτων και όχι στο ποσοστό χρήσης της επεξεργαστικής δύναμης από τους υποδοχείς. Τέλος, η οικονομική ανάλυση λαμβάνει υπόψιν τις τιμολογήσεις από παρόχους του υπολογιστικού νέφους και σε συνδυασμό με τα παραπάνω εξάγει ποσοτικά και ποιοτικά συμπεράσματα ως προς την μείωση κόστους λειτουργίας της υπηρεσίας και την ανεπαίσθητη μείωση ποιότητάς της.

1.3 Συνεισφορά

Κατά τη βιβλιογραφική μελέτη πραγματοποιήθηκαν επιγραμματικά οι παρακάτω ενέργειες:

1. Εύρεση και κατανόηση πληροφοριών σχετικά με το υπολογιστικό νέφος.
2. Αναζήτηση πληροφοριών και όρων ως προς τις συμφωνίες επιπέδου υπηρεσιών (SLA).
3. Συγκέντρωση και ανάλυση πληροφοριών ως προς την μονολιθική αρχιτεκτονική.
4. Μελέτη και ανάλυση της αρχιτεκτονικής των μικροϋπηρεσιών.
5. Έρευνα και κατανόηση της πλατφόρμας docker.

1.4 Βασική Ορολογία

1.4.1 Υπολογιστικό Νέφους

Το Υπολογιστικό Νέφος (Cloud Computing) είναι το σύνολο των διαθέσιμων μηχανημάτων, δικτύων, αποθηκευτικών χώρων, υπηρεσιών και διεπαφών, που διανέμουν κατά παραγγελία από κοινού τον κάθε επιθυμητό υπολογισμό ως υπηρεσία. Οι υπηρεσίες νέφους περιλαμβάνουν την παροχή λογισμικού, υποδομών και αποθήκευσης μέσω του διαδικτύου (είτε ως ξεχωριστά στοιχεία, είτε ως ολοκληρωμένη πλατφόρμα) βασισμένη σε αυτό που έχει ζητήσει ο χρήστη [19].

1.4.2 Μονολιθική Αρχιτεκτονική

Για τη μονολιθική αρχιτεκτονική μπορούν να δοθούν δύο ορισμοί. Ο μη τεχνικός ορισμός είναι ότι, κάτι κατασκευάζεται πάνω σε αυστηρά ένα μόνο υλικό, δηλαδή σε έναν μόνο λίθο, ενώ από τεχνική σκοπιά ο ορισμός που μπορεί να δοθεί είναι ότι μια μονολιθική εφαρμογή έχει ενιαία

βάση κώδικα με πολλαπλές ενότητες. Οι ενότητες χωρίζονται είτε ως χαρακτηριστικά των επιχειρήσεων είτε ως τεχνικά χαρακτηριστικά. Περιέχει ένα ενιαίο σύστημα κατασκευής το οποίο δημιουργεί ολόκληρη την εφαρμογή ή και τις εξαρτήσεις και τέλος ένα εκτελέσιμο ή αναπτυσσόμενο δυαδικό.

1.4.3 SLI - SLO - SLA

Μια συμφωνία σε επίπεδο υπηρεσίας (SLA) είναι συνήθως μια δέσμευση μεταξύ ενός παρόχου υπηρεσιών και ενός πελάτη. Αυτή η συμφωνία, περιλαμβάνει συνέπειες από τη μη τήρηση των στόχων σε επίπεδο υπηρεσιών (SLO) που έχουν τεθεί, οι οποίοι στόχοι μπορούν να γίνουν μετρήσιμοι μέσω των δεικτών επιπέδου υπηρεσιών (SLI).

1.4.4 Αρχιτεκτονική Μικροϋπηρεσιών

Η αρχιτεκτονική μικροϋπηρεσιών είναι μια προσέγγιση δημιουργίας μιας μεγάλης επιχειρηματικής εφαρμογής με πολλαπλές μικρές μονάδες που ονομάζονται μικροϋπηρεσίες, που αναπτύσσονται και δοκιμάζονται ξεχωριστά η κάθε μονάδα από την άλλη. Κάθε μικροϋπηρεσία διασυνδέεται με ένα κοινό πρωτόκολλο επικοινωνίας όπως είναι το Representational State Transfer (REST) και με χρήση JavaScript Object Notation (JSON) ανταλλάσσονται πληροφορίες. Κάθε μικροϋπηρεσία κατασκευάζεται ξεχωριστά είτε σε ένα μόνο μηχάνημα είτε σε διαφορετικό μηχάνημα, αλλά εκτελεί τη δική του μοναδική διαδικασία. Κάθε μικροϋπηρεσία μπορεί να έχει είτε τη δική της βάση δεδομένων ή σύστημα αποθήκευσης είτε μπορεί να μοιράζεται κοινή βάση δεδομένων ή σύστημα αποθήκευσης. Η αρχιτεκτονική των μικροϋπηρεσιών έχει ως στόχο να διανείμει ή να διαχωρίσει μια μεγάλη εφαρμογή σε μικρότερα κομμάτια-εφαρμογές.

1.4.5 Docker

Το Docker παρέχει ορισμένες εγκαταστάσεις, οι οποίες είναι χρήσιμες για προγραμματιστές και διαχειριστές. Είναι μια ανοιχτή πλατφόρμα που μπορεί να χρησιμοποιηθεί για την κατασκευή, τη διανομή και την εκτέλεση εφαρμογών σε ένα φορητό, μικρού χρόνου εκτέλεσης και συσκευασίας εργαλείο, γνωστό και ως Docker Engine. Παρέχει επίσης το Docker Hub, το οποίο είναι υπηρεσία νέφους για κοινή χρήση εικόνων-εφαρμογών. Το κόστος ανακατασκευής σε πλατφόρμα υπολογιστικού νέφους μειώνεται σε μεγάλο βαθμό, μέσω της αντικατάστασης της παραδοσιακής εικονικής μηχανής με τον υποδοχέα docker, το οποίο ποτελεί και το κεντρικό θέμα της διατριβής [1].

1.5 Διάρθρωση της Μελέτης

Η βιβλιογραφική μελέτη της διατριβής βρίσκεται στο 2ο κεφάλαιο, το οποίο έχει σα κύριο στόχο να αναλυθούν όλες οι βασικές ορολογίες που χρησιμοποιήθηκαν. Περεταίρω στο 3ο κεφάλαιο αναλύονται τα εργαλεία και οι εφαρμογές που αναπτύχθηκαν, στα πλαίσια της διπλωματικής, τόσο με την αρχιτεκτονική των μικροϋπηρεσιών όσο και με την μονολιθική προσέγγιση. Στο 4ο κεφάλαιο, παρουσιάζεται η μεθοδολογία της πειραματικής ανάπτυξης, τίθενται όρια ως προς τις δοκιμές, που διεξήχθησαν στις παραπάνω εφαρμογές και παρουσιάζονται τα αποτελέσματα αυτών , ενώ παράλληλα αναπτύσσεται μια μελέτη περίπτωσης. Τέλος, στο 5ο κεφάλαιο αναλύονται τα συμπεράσματα της διπλωματικής τα οποία αποτελούν απόρροια των παραπάνω αποτελεσμάτων, αναφέρονται δυσκολίες που προέκυψαν κατά την διάρκεια της εκπόνησης της διατριβής και επεκτάσεις για μελλοντική έρευνα.

2 Βιβλιογραφική Επισκόπηση – Θεωρητικό Υπόβαθρο

Στο 2ο κεφάλαιο θα αναπτυχθεί όλη η βιβλιογραφική ανάλυση-μελέτη που απαιτήθηκε προκειμένου να γίνει κατανοητό στον αναγνώστη το αντικείμενο της έρευνας, αλλά και να αποτελέσει βάση της παρούσας διατριβής. Πραγματοποιήθηκε προσπάθεια να αναλυθούν σε βάθος όλες οι απαραίτητες έννοιες που χρησιμοποιήθηκαν. Βασικές ορολογίες που θα συναντήσει ο αναγνώστης στο 2ο κεφάλαιο είναι επιγραμματικά το υπολογιστικό νέφος, όλες οι απαραίτητες έννοιες επί των συμφωνιών επιπέδου υπηρεσιών, η μονολιθική αρχιτεκτονική, η αρχιτεκτονική των μικροϋπηρεσιών και η πλατφόρμα docker.

2.1 Υπολογιστικό Νέφος

Εν έτει 2020, οι επιχειρήσεις αντιμετωπίζουν σοβαρά προβλήματα ρευστότητας, διότι είτε δυσκολεύονται να πραγματοποιήσουν μεγάλες πληρωμές, σε μικρό χρονικό διάστημα, το οποίο μπορεί να οδηγήσει σε πτώχευση, είτε εξ' αρχής κατέστη ανέφικτη η ίδρυσή τους λόγω έλλειψης οικονομικών πόρων. Σε αυτό το πρόβλημα, έρχεται να προσφέρει μια λύση το υπολογιστικό νέφος, το οποίο μπορεί να προσφέρει τους αναγκαίους υπολογιστικούς πόρους με αντάλλαγμα τακτικές πληρωμές σε βάθος χρόνου. Έχοντας το παραπάνω υπόψιν, επιδιώκεται να υπάρξει σύγκριση των δύο αρχιτεκτονικών με σκοπό την εξαγωγή συμπερασμάτων ως προς την αποταμίευση χρηματικών πόρων στο υπολογιστικό νέφος.

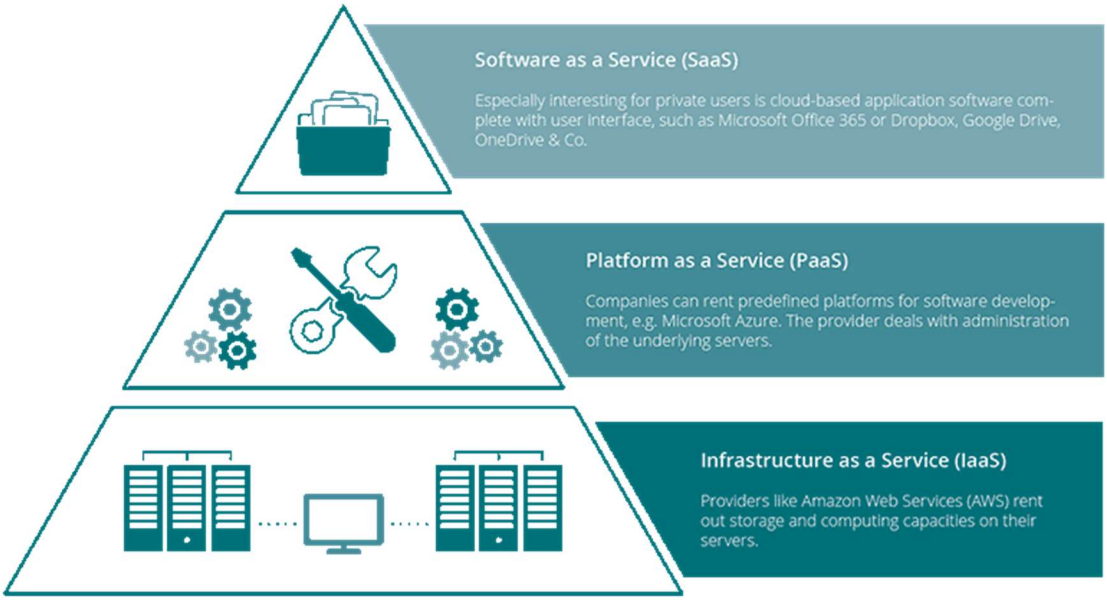
Το υπολογιστικό νέφος είναι ένα αναδυόμενο μοντέλο για την παροχή των υπολογιστικών πόρων μέσω δικτύου. Πραγματοποιούνται αρκετές προσπάθειες για τον ορισμό του υπολογιστικό νέφους. Ο Vaquero [12] ορίζει ότι: «Τα σύννεφα είναι μεγάλη πισίνα των ευκόλως χρησιμοποιήσιμων και προσβάσιμων εικονικοποιημένων πόρων (όπως είναι οι υλικοί πόροι-επεξεργαστές υπολογιστών, οι αναπτυξιακές πλατφόρμες και οι δικτυακές εφαρμογές). Αυτοί οι πόροι μπορούν να αναπροσαρμοστούν δυναμικά ώστε να προσαρμοστούν σε ένα μεταβλητό φορτίο (κλίμακα), επιτρέποντας επίσης τη βέλτιστη χρήση των πόρων. Αυτή η ομάδα πόρων είναι που συνήθως χρησιμοποιείται από ένα μοντέλο αμοιβής ανά χρήση, στο οποίο προσφέρονται εγγυήσεις από τον πάροχο της υποδομής μέσω προσαρμοσμένων συμφωνιών σε επίπεδο υπηρεσιών, τα λεγόμενα SLA (Service Level Agreement). Ένας πιο λεπτομερής ορισμός παρέχεται από το Εθνικό Ινστιτούτο Προτύπων και Τεχνολογίας (NIST). Σύμφωνα λοιπόν με το NIST «Το υπολογιστικό νέφος είναι ένα μοντέλο που επιτρέπει την πανταχού παρούσα, βολική, κατά απαίτηση πρόσβαση δικτύου σε κοινόχρηστους διαμορφωμένους υπολογιστικούς πόρους,

όπως δίκτυα, διακομιστές, αποθηκευτικούς χώρους, εφαρμογές και υπηρεσίες, που μπορούν να διατεθούν γρήγορα με ελάχιστη προσπάθεια διαχείρισης [8].

Σύμφωνα με το NIST, το μοντέλο σύννεφο αποτελείται από πέντε βασικά χαρακτηριστικά:

1. Αυτοεξυπηρέτηση κατ' απαίτηση: ο καταναλωτής μπορεί να χρησιμοποιήσει υπολογιστικούς πόρους αυτόματα χωρίς καμία ανθρώπινη παρέμβαση.
2. Ευρεία πρόσβαση στο δίκτυο: Οι πόροι του υπολογιστικού νέφους είναι διαθέσιμοι μέσω του δικτύου, υποστηρίζοντας πολλές διαφορετικές συσκευές όπως κινητά, ταμπλέτες και σταθερούς ή φορητούς υπολογιστές.
3. Συγκέντρωση πόρων: Εξυπηρέτηση πολλών πελατών χρησιμοποιώντας μοντέλο μισθωτών που παρέχει την αίσθηση της ανεξαρτησίας της τοποθεσίας σε κάθε πελάτη. Ωστόσο, ο πελάτης μπορεί να εξυπηρετηθεί από τον ίδιο φυσικό πόρο ο οποίος θεωρητικά παρέχει μεγαλύτερη ασφάλεια.
4. Ταχεία ελαστικότητα: Οι πόροι διατίθενται και χρησιμοποιούνται κατ' απαίτηση ή και αυτοματοποιημένα με σκοπό να καθίσταται σίγουρο ότι η εφαρμογή έχει ακριβώς την απαιτούμενη υπολογιστική δύναμη ανά πάσα στιγμή.
5. Μετρούμενη υπηρεσία: Η χρήση των πόρων παρακολουθείται, υπολογίζεται και χρεώνεται με βάση τη χρήση δημιουργώντας ένα μοντέλο αμοιβής ανά χρήση προσαρμοσμένο στις ανάγκες του εκάστοτε πελάτη.

Το υπολογιστικό νέφος διαθέτει τρία μοντέλα υπηρεσιών, συγκεκριμένα: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) και Software-as-a-Service (SaaS). Κάθε μοντέλο υπηρεσίας παρέχει ένα διαφορετικό επίπεδο. Το παρακάτω σχήμα δείχνει την πολυεπίπεδη προβολή των μοντέλων υπηρεσίας στο υπολογιστικό νέφος.



Εικόνα 2-1: Μοντέλα Υπηρεσιών στο υπολογιστικό νέφος

Υποδομή-ως-Υπηρεσία: Το IaaS είναι το χαμηλότερο επίπεδο των συστημάτων υπολογιστικού νέφους. Το IaaS προσφέρει υπολογιστική δύναμη, αποθήκευση και δικτύωση σε μορφή Εικονικών Μηχανών (VM). Το NIST ορίζει το IaaS ως «Τη παροχή υπηρεσιών επεξεργασίας, αποθήκευσης, δικτύων και άλλων θεμελιωδών υπολογιστών πόρων. Ο καταναλωτής είναι σε θέση να αναπτύξει και να εκτελέσει ανενόχλητα λογισμικό, το οποίο μπορεί να περιλαμβάνει λειτουργικά συστήματα και εφαρμογές». Το IaaS στρώμα διαχειρίζεται το υποκείμενο υλικό και παρέχει στον καταναλωτή μια ενιαία πολιτική για τους ετερογενείς πόρους. Η εικονικοποίηση αποτελεί ένα κυρίαρχο στοιχείο στο IaaS, διότι με τη βοήθειά της μπορεί να εξυπηρετεί, σε περιβάλλον πολλαπλών μισθωτών, πολλαπλούς καταναλωτές από τον ίδιο φυσικό πόρο, επιτυγχάνοντας έτσι αποδοτικότητα πόρων. Οι καταναλωτές αν και δεν έχουν κανένα έλεγχο επί της υποκείμενης υποδομής, έχουν τον πλήρη έλεγχο της εικονικής μηχανής.

Η υποδομή του υπολογιστικού νέφους πρέπει να διαχειρίζεται τους φυσικούς και εικονικούς πόρους, τον κύκλο ζωής των εικονικών μηχανών, τη δημιουργία δικτύων κ.α. Τα εργαλεία λογισμικού που παρέχουν τις παραπάνω δυνατότητες αποτελούν έναν εικονικό διαχειριστή υποδομής, τον λεγόμενο «Virtual Infrastructure Manager, VIM». Το VIM παρέχει επίσης τη διαχείριση των διεπαφών προγραμματισμού εφαρμογών καθώς και μια διεπαφή μπροστινού άκρου μέσω της οποίας οι καταναλωτές μπορούν να κάνουν αιτήματα στις εικονικές μηχανές και σε άλλους πόρους. Οι εικονικές μηχανές προσφέρονται σε διάφορες διαμορφώσεις

επεξεργαστικής ισχύος, μνήμης και χωρητικότητας. Το Openstack και το VMware vSphere είναι παραδείγματα του VIM.

Το IaaS παρέχει πολλά πλεονεκτήματα στον καταναλωτή. Παρέχει εξοικονόμηση κόστους, επειδή ο καταναλωτής δεν χρειάζεται να επενδύσει στους φυσικούς πόρους. Επιπλέον, ο καταναλωτής πληρώνει τώρα μόνο για τους πόρους που χρησιμοποιήθηκαν. Η παροχή εικονικών πόρων αντί των φυσικών πόρων βελτιώνει επίσης το χρόνο στην αγορά και στη σύνθεση των πόρων, καθώς η παροχή είναι σχεδόν στιγμιαία. Οι επιχειρήσεις μπορούν να επικεντρωθούν πλήρως στην βασική δραστηριότητά τους χωρίς να χρειάζεται να σπαταλήσουν χρόνο, χώρο και χρήματα στη δημιουργία και συντήρηση της υποδομής.

Η Πλατφόρμα-ως-Υπηρεσία (PaaS) παρέχει ένα μέρος της υποκείμενης υποδομής, του λογισμικού και των πόρων της εφαρμογής. Το NIST ορίζει το PaaS ως «Η ικανότητα που παρέχεται στον καταναλωτή να αναπτύξει τις υπηρεσίες του στην υποδομή του υπολογιστικού νέφους, η οποία δημιουργείται από καταναλωτές, χρησιμοποιώντας τις γλώσσες προγραμματισμού, τις βιβλιοθήκες, τις υπηρεσίες και τα εργαλεία που υποστηρίζει ο πάροχος.» Η PaaS παρέχει μια πλατφόρμα για την κατασκευή, την ανάπτυξη, την εκτέλεση και τη διαχείριση των εφαρμογών λογισμικού. Ωστόσο, ο καταναλωτής δεν διαχειρίζεται την υποκείμενη υποδομή, όπως ένα λειτουργικό σύστημα, δίκτυο ή διακομιστές, αλλά έχει τον έλεγχο της αναπτυχθείσας εφαρμογής και του περιβάλλοντος στο οποίο φιλοξενείται [8]. Αξιοσημείωτα παραδείγματα πλατφορμών PaaS είναι IBM Bluemix, Microsoft Azure και Google App Engine.

Η εφαρμογή που αναπτύσσεται στο PaaS προσφέρεται στους καταναλωτές μέσω του δικτύου σαν Λογισμικό ως υπηρεσία (Software-as-a-Service SaaS). Το SaaS έχει το υψηλότερο επίπεδο αφαίρεσης. Οι καταναλωτές του SaaS δεν έχουν κανέναν έλεγχο επί της υποκείμενης υποδομής και των δυνατοτήτων της εφαρμογής. Η εφαρμογή μπορεί να χρησιμοποιηθεί είτε από τους τελικούς χρήστες απευθείας μέσω των πελατών, όπως ένα πρόγραμμα περιήγησης ή από εφαρμογές τρίτων μέσω APIs. Το SaaS προσφέρει μια εναλλακτική λύση στις εφαρμογές που είναι εγκατεστημένες στους τοπικούς υπολογιστές. Παραδείγματα των SaaS περιλαμβάνουν το Fiori του SAP, Έγγραφα Google και πολλά άλλα.

Υπάρχουν, κυρίως, τέσσερα μοντέλα ανάπτυξης για το υπολογιστικό νέφος τα οποία επιγραμματικά είναι το ιδιωτικό, το δημόσιο, το κοινό και το υβριδικό.

Το ιδιωτικό υπολογιστικό νέφος είναι μια υποδομή για υπολογιστικό νέφος που δεν είναι διαθέσιμο στο ευρύ κοινό και είναι αποκλειστικό μόνο για τον πελάτη που το χρησιμοποιεί. Ένα ιδιωτικό υπολογιστικό νέφος μπορεί να υπάρχει στις εγκαταστάσεις ή εκτός των εγκαταστάσεων και μπορεί να ανήκει, να διαχειρίζεται και να το λειτουργεί ο οργανισμός ή μια τρίτη οργάνωση ή

οποιοσδήποτε συνδυασμός παραπάνω. Το κίνητρο για ιδιωτικό υπολογιστικό νέφος είναι να αξιοποιήσει τους εσωτερικούς πόρους και να μειώσουν τις ανησυχίες για την ασφάλεια, όπως είναι η ιδιωτικότητα των δεδομένων και η εμπιστοσύνη ως προς τη διαχείριση από τρίτους. Εταιρείες όπως το Redhat και το VMWare ανήκουν στους ιδιωτικούς παρόχους του υπολογιστικού νέφους.

Η υποδομή υπολογιστικού νέφους που διατίθεται στο ευρύ κοινό αναφέρεται ως δημόσιο σύννεφο. Οι πάροχοι αυτού του είδους του υπολογιστικού νέφους έχουν τον έλεγχο της υποδομής. Κάθε δημόσιος προμηθευτής υπολογιστικού νέφους μπορεί να έχει το δικό του σύνολο πολιτικών και μοντέλου χρέωσης. Τα δημόσια οφέλη υπολογιστικού νέφους περιλαμβάνουν εξοικονόμηση του προκαταβολικού κόστους του υλικού, την επεκτασιμότητα κατά παραγγελία, την εύκολη ρύθμιση στη χρήση και πολλά άλλα. Το Pivotal, το Amazon EC2 είναι μερικά παραδείγματα του δημόσιου υπολογιστικού νέφους.

Αρκετοί οργανισμοί που συμμερίζονται τις ίδιες ανησυχίες μπορούν να παρέχουν αποκλειστικά μια υποδομή υπολογιστικού νέφους για τη χρήση τους, το οποίο ονομάζεται κοινό υπολογιστικό νέφος. Η υποδομή του υπολογιστικού νέφους μπορεί να ανήκει ή να διαχειρίζεται από έναν ή περισσότερους οργανισμούς ή από ένα τρίτο μέρος.

Τέλος, το υβριδικό υπολογιστικό νέφος είναι ένας συνδυασμός δύο ή περισσότερων ξεχωριστών υποδομών υπολογιστικού νέφους. Οι οντότητες, ωστόσο, παραμένουν μοναδικές και συνδέονται μεταξύ τους με τυποποιημένες τεχνολογίες που επιτρέπουν τη μεταφορά δεδομένων και εφαρμογών (π.χ., έκρηξη υπολογιστικού νέφους για εξισορρόπηση φορτίου μεταξύ υπολογιστικών νεφών).

2.2 SLI – SLO -SLA

Καθώς ο αυξανόμενος αριθμός καταναλωτών υπολογιστικού νέφους αποστέλλει το φόρτο εργασίας του σε παρόχους υπολογιστικού νέφους, η συμφωνία επιπέδου υπηρεσίας (SLA) μεταξύ καταναλωτών και παρόχων καθίσταται υψίστης σημασίας ώστε να διασφαλιστεί ότι, η ποιότητα των υπηρεσιών διατηρείται σε ικανοποιητικά επίπεδα ανεξάρτητα από τη δυναμική φύση του περιβάλλοντος του υπολογιστικού νέφους. Το SLA λοιπόν, είναι μια συμφωνία, τυπική ή άτυπη, μεταξύ πελάτη και παρόχου υπηρεσίας, η οποία περιέχει μια επεξήγηση της συμφωνημένης υπηρεσίας, τις παραμέτρους του επιπέδου υπηρεσίας, τις εγγυήσεις σχετικά με την ποιότητα της υπηρεσίας, τις ρυθμίσεις και τις θεραπείες για όλες τις περιπτώσεις παραβίασης [25]. Το SLA και οι παρελκόμενες έννοιες μελετώνται, με στόχο την εύρεση ενός διαφορετικού τρόπου κλιμάκωσης των υποδοχέων, ο οποίος βασίζεται στην συμφωνία μεταξύ πελάτη και παρόχου.

Για να υπάρξει σωστή διαχείριση μιας υπηρεσίας πρέπει αρχικά να γίνουν κατανοητά ποια αποτελέσματα της υπηρεσίας έχουν πραγματικά σημασία και έπειτα πώς μπορούν να μετρηθούν και να αξιολογηθούν.

Για να επιτευχθεί ο παραπάνω στόχος και σύμφωνα με τα δεδομένα ως προς την ικανοποίηση των χρηστών, ορίστηκαν δείκτες επιπέδου υπηρεσιών (Service Level Indicators), στόχοι (Service Level Objectives) και συμφωνίες (Service Level Agreements). Αυτές οι μετρήσεις περιγράφουν τους κυριότερους συμφωνηθέντες όρους, δηλαδή ποιες τιμές είναι αποδεκτές, και ποιες θα είναι οι επιπτώσεις, σε περίπτωση αθέτησης των ανωτέρω όρων. Τελικά, η επιλογή των κατάλληλων μετρήσεων συμμετέχει στην επιλογή της σωστής αντίδρασης σε περίπτωση αθέτησης των όρων και προσφέρει στους διαχειριστές της υπηρεσίας την αυτοπεποίθηση ότι η υπηρεσία τους είναι υγιής.

2.2.1 Δείκτες Επιπέδου Υπηρεσιών (SLI)

Το SLI είναι ένας δείκτης επιπέδου υπηρεσιών - ένα προσεκτικά καθορισμένο ποσοτικό μέτρο κάποιας πτυχής του επιπέδου υπηρεσίας που παρέχεται [21].

Αρχικά, ένας δείκτης επιπέδου υπηρεσιών σημαντικός για τους διαχειριστές είναι η διαθεσιμότητα, ή το ποσοστό του χρόνου που μια υπηρεσία είναι διαθέσιμη. Συχνά ορίζεται με βάση το κλάσμα, καλά εξαπλωμένων στο χρόνο, αιτημάτων που επιτυγχάνουν προς τα συνολικά αιτήματα. Εάν και η διαθεσιμότητα 100% είναι πρακτικά αδύνατη, η διαθεσιμότητα της υπηρεσίας σε ποσοστό 95% είναι συχνά εφικτή και συνηθίζεται να περιλαμβάνεται στα διάφορα πακέτα υπηρεσιών υπολογιστικού νέφους.

Οι περισσότερες υπηρεσίες λαμβάνουν σημαντικά υπόψιν το χρόνο που χρειάζεται για να επιστρέψει μια απάντηση από ένα αίτημα, για αυτό άλλωστε και θεωρείται ο σημαντικότερος δείκτης. Επιπλέον, σε συνδυασμό με τον παραπάνω, ένας δείκτης που λαμβάνεται υπόψιν είναι αυτός που εκφράζει το ποσοστό σφάλματος, δηλαδή το κλάσμα των σωστών απαντήσεων προς όλων των αιτημάτων που δημιουργήθηκαν.

Τέλος, σημαντικό είναι να σημειωθεί ότι το SLI μετρά άμεσα ένα επίπεδο εξυπηρέτησης ενδιαφέροντος, αλλά πολλές φορές υπάρχουν διαθέσιμες πληροφορίες μόνο για τον υπολογισμό ενός παρόμοιου δείκτη και όχι του πιο σχετικού, διότι μπορεί είναι δύσκολο να αποκτηθεί ή να ερμηνευτεί ο τελευταίος. Για παράδειγμα, η καθυστέρηση απόκρισης στην πλευρά του πελάτη είναι συχνά η πιο σχετική μέτρηση για το χρήστη, αλλά μπορεί να είναι δυνατή μόνο η μέτρηση χρόνου επιστροφής της απάντησης του αιτήματος στο διακομιστή.

2.2.2 Στόχοι Επιπέδου Υπηρεσιών (SLO)

Ένας στόχος επιπέδου υπηρεσίας είναι μια τιμή στόχου ή ένα εύρος τιμών για ένα επίπεδο υπηρεσίας που μετριέται από ένα SLI. Επομένως, μια φυσική δομή για SLOs είναι SLI ≤ στόχος ή κατώτερο όριο ≤ SLI ≤ ανώτερο όριο.

Η επιλογή ενός κατάλληλου SLO είναι πολύπλοκη. Αρχικά, δεν μπορεί πάντα να γίνει επιλογής της τιμής του, εάν ο δείκτης καθορίζεται από τις επιθυμίες των χρηστών. Ο δείκτης QPS, ερωτήματα ανά δευτερόλεπτο, αποτελεί χαρακτηριστικό παράδειγμα των παραπάνω.

Από την άλλη πλευρά, ένας στόχος που συνήθως θέτουν οι διαχειριστές των υπηρεσιών είναι ότι η μέση καθυστέρηση των αιτημάτων πρέπει να παραμένει κάτω από 3 δευτερόλεπτα. Ο καθορισμός ενός τέτοιου στόχου μπορεί να παρακινήσει την δημιουργία του κώδικα με τις κατάλληλες μεθόδους ή την αγορά κατάλληλου εξοπλισμού με σκοπό την επίτευξη ελαχιστοποίησης του χρόνου ανταπόκρισης των απαντήσεων των αιτημάτων των χρηστών.

Η επιλογή και η δημοσίευση των SLO στους χρήστες θέτει προσδοκίες για τον τρόπο με τον οποίο θα λειτουργεί μια υπηρεσία. Αυτή η στρατηγική μπορεί να μειώσει τις αβάσιμες καταγγελίες στους κατόχους υπηρεσιών, για παράδειγμα, ότι η υπηρεσία είναι αργή. Χωρίς ένα ξεκάθαρο SLO, οι χρήστες συχνά αναπτύσσουν τις δικές τους πεποιθήσεις σχετικά με τις επιθυμητές επιδόσεις, οι οποίες μπορεί να μην σχετίζονται με τις πεποιθήσεις των ανθρώπων που σχεδιάζουν και λειτουργούν την υπηρεσία. Αυτή η στρατηγική μπορεί να οδηγήσει σε υπερβολική εμπιστοσύνη για την υπηρεσία, όταν οι χρήστες πιστεύουν λανθασμένα ότι η υπηρεσία θα είναι πιο διαθέσιμη από ότι στην πραγματικότητα είναι και από την άλλη σε έλλειψη εμπιστοσύνης όταν πιστεύουν ότι το σύστημα είναι πιο αδύναμο και λιγότερο αξιόπιστο από ότι στην πραγματικότητα.

2.2.3 Συμφωνίες Επιπέδου Υπηρεσιών (SLA)

Τέλος, οι συμφωνίες επιπέδου υπηρεσιών αποτελούν μια ρητή ή σιωπηρή σύμβαση με τους χρήστες της εφαρμογής που περιλαμβάνει συνέπειες από τη μη τήρηση των SLO που έχουν τεθεί. Οι συνέπειες αναγνωρίζονται πιο εύκολα όταν είναι οικονομικές, όπως μια έκπτωση ή μια ποινή για την παραβίαση των συμφωνιών, αλλά μπορούν να έχουν και άλλες μορφές. Ένας εύκολος τρόπος διάκρισης ανάμεσα στο στόχο και στη συμφωνία είναι η απάντηση στο ερώτημα «τι συμβαίνει εάν οι στόχοι δεν πληρούνται;». Εάν δεν υπάρχει σαφής συνέπεια, τότε κατά πάσα πιθανότητα αποτελεί ένα SLO.

Οι διαχειριστές της εφαρμογής δεν συμμετέχουν συνήθως στην κατασκευή των SLA, επειδή οι SLA συνδέονται στενά με τις αποφάσεις των επιχειρήσεων και των προϊόντων.

Εντούτοις, οι διαχειριστές συμμετέχουν στην αποφυγή των συνεπειών των χαμένων SLO. Μπορούν, επίσης, να βοηθήσουν στον ορισμό των κατάλληλων SLI και προφανώς πρέπει να υπάρχει ένας αντικειμενικός τρόπος μέτρησης των SLO στη συμφωνία με σκοπό να μη προκύψουν διαφωνίες από τις δύο πλευρές.

Χαρακτηριστικό παράδειγμα σημαντικής υπηρεσίας που δεν διαθέτει SLA αποτελεί η Αναζήτηση του Google. Το κοινό θέλει να χρησιμοποιεί την υπηρεσία όσο το δυνατόν πιο αποτελεσματικά, αλλά δεν έχει υπογράψει καμία σύμβαση με τη Google και τους χρήστες . Παρόλα αυτά, εξακολουθούν να υπάρχουν συνέπειες εάν η αναζήτηση δεν είναι διαθέσιμη ή η μη διαθεσιμότητα έχει ως αποτέλεσμα τη δυσφήμιση, καθώς και πτώση των εσόδων από τις διαφημίσεις. Πολλές άλλες υπηρεσίες Google, όπως το Google Cloud, περιλαμβάνει σαφή SLAs με τους χρήστες τους. Ανεξάρτητα εάν μια συγκεκριμένη υπηρεσία έχει SLA ή όχι, είναι σημαντικό να οριστούν SLIs και SLOs, με σκοπό να χρησιμοποιούνται για την κατάλληλη διαχείριση της υπηρεσίας.

2.3 Μονολιθική Αρχιτεκτονική

Ο κλασικός τρόπος ανάπτυξης εφαρμογών πραγματοποιείται μέσω της μονολιθικής αρχιτεκτονικής. Αυτό είχε σαν αποτέλεσμα, στις αρχές του υπολογιστικού νέφους, την εμφάνιση μονολιθικών εφαρμογών σε αυτό, καθιστώντας την μονολιθική αρχιτεκτονική και το υπολογιστικό νέφος στενά συνδεδεμένες έννοιες. Άλλωστε, η έρευνα για την μονολιθική αρχιτεκτονική παίζει σημαντικό ρόλο στην κατανόηση της σύγκρισης του 4ου κεφαλαίου ανάμεσα στην μονολιθική εφαρμογή και στην εφαρμογή μέσω της αρχιτεκτονικής των μικροϋπηρεσιών.

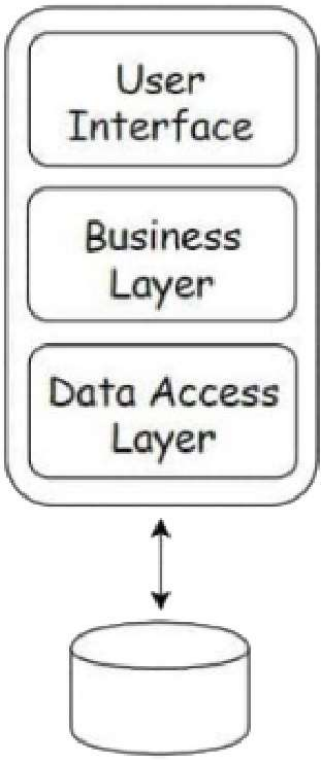
Μια μονολιθική εφαρμογή είναι μια εφαρμογή, η οποία χρησιμοποιεί ένα κύριο κώδικα για να εξυπηρετήσει πολλές διαφορετικές υπηρεσίες και διεπαφές όπως REST πρωτόκολλα, APIs και HTML σελίδες. Η μονολιθική αρχιτεκτονική θεωρείται η τυπική προσέγγιση ανάπτυξης μιας μικρής εφαρμογής, διότι η δημιουργία κώδικα σε μια κοινή βάση απαλλάσσει τον δημιουργό από περιττή πολυπλοκότητα.

Μια μονολιθική εφαρμογή χρησιμοποιεί συνήθως μία ενιαία βάση δεδομένων για να χειριστεί όλα τα δεδομένα. Η βάση δεδομένων μπορεί να κλιμακωθεί σε διαφορετικά μέρη, αλλά χρησιμοποιώντας πάντα το ίδιο μοντέλο. Με μια ενιαία βάση δεδομένων, οι συναλλαγές είναι συνήθως εύκολα διαχειρίσιμες. καθώς τα περισσότερα συστήματα βάσεων δεδομένων παρέχουν το ACID (Atomicity, Consistency, Isolation, Durability), δηλαδή ατομικότητα, συνέπεια, απομόνωση και αντοχή. Οι προγραμματιστές μπορούν εύκολα να ορίσουν τις συναλλαγές και να

εστιάσουν περισσότερο στην παροχή νέων δυνατοτήτων στους τελικούς χρήστες. Βέβαια, η ενιαία βάση δεδομένων έχει τους περιορισμούς της. Η μονολιθική εφαρμογή μπορεί να έχει πολλά διαφορετικά είδη δεδομένων. Μερικά από τα δεδομένα θα μπορούσαν να είναι πιο κατάλληλα για να αποθηκευτούν σε μια βάση δεδομένων NoSQL και μερικά από τα υπόλοιπα σε μια σχεσιακή βάση δεδομένων. Ωστόσο, με την μονολιθική προσέγγιση ο προγραμματιστής πρέπει να επιλέξει συνήθως μόνο μία μηχανή βάσης δεδομένων και να την χρησιμοποιήσει για όλα τα είδη δεδομένων.

Οι περισσότερες εφαρμογές μπορεί να είναι αρκετά απλές στην αρχή, αλλά όσο η εφαρμογή αναπτύσσεται περεταίρω τόσο και η πολυπλοκότητα της αυξάνεται. Ένας τυπικός τρόπος αντιμετώπισης της πολυπλοκότητας μιας εφαρμογής που έχει μια μονολιθική αρχιτεκτονική είναι να χωριστεί η εφαρμογή σε διαφορετικά στρώματα. Μια πολυεπίπεδη προσέγγιση η οποία χρησιμοποιείται ευρέως σε δίκτυα δικτύωσης και λειτουργικών συστημάτων.

Η πολυεπίπεδη μονολιθική αρχιτεκτονική που φαίνεται στο σχήμα 1 είναι η πιο κοινότοπη. Η εφαρμογή χωρίζεται σε ένα επίπεδο διεπαφής χρήστη UI (User Interface), σε ένα στρώμα υπηρεσιών και σε ένα στρώμα πρόσβασης δεδομένων. Στη συνέχεια, το επίπεδο πρόσβασης δεδομένων έχει συνήθως πρόσβαση σε μία βάση δεδομένων που χειρίζεται όλα τα δεδομένα που σχετίζονται με την εφαρμογή.



Εικόνα 2-2: Μια τυπική μονολιθική εφαρμογή

Αυτή η μονολιθική προσέγγιση καθιστά εύκολη την δημιουργία, την ανάπτυξη και την επεκτασιμότητα όταν το μέγεθος της εφαρμογής είναι σχετικά μικρό [11]. Οι προγραμματιστές νιώθουν οικεία με αυτού του είδους την αρχιτεκτονική καθώς για πολλά χρόνια αποτελούσε τη βασική προσέγγιση ανάπτυξης μια εφαρμογής.

Βέβαια, σε ένα μονολιθικό κώδικα βάσης, η ανάπτυξη περιλαμβάνει πάντα κάθε κομμάτι της εφαρμογής, καθώς κάθε φορά που αλλάζει ένα μόνο στοιχείο, η εφαρμογή πρέπει να επανατοποθετηθεί ολόκληρη στο αντίστοιχο μηχάνημα. Αυτό καθιστά τη συνεχή ανάπτυξη δύσκολα πραγματοποιήσιμη, χρονοβόρα, κοστοβόρα αλλά και στη πραγματικότητα αποτυγχάνει να αξιολογεί τις αντιδράσεις των χρηστών και να προσαρμόζεται ανάλογα.

Η κλιμάκωση μιας μονολιθικής εφαρμογής γίνεται προσθέτοντας νέους κόμβους στο ίδιο τεχνούργημα. Η επεκτασιμότητα καθίσταται απλή, καθώς μειώνονται αισθητά οι επιλογές της κλιμάκωσης. Τα στοιχεία της εφαρμογής που χρειάζονται περισσότερους υπολογιστικούς πόρους πρέπει να κλιμακωθούν μαζί με όλα τα υπόλοιπα στοιχεία που θα μπορούσαν να καλύψουν το φόρτο εργασίας τους με λιγότερους πόρους, με αποτέλεσμα να υπάρχει σπατάλη πόρων και πρόσθετες δαπάνες χωρίς κάποια χρησιμότητα.

Η δημιουργία μιας εφαρμογής μέσω της μονολιθικής προσέγγισης αποτελεί κλασική επιλογή για τον προγραμματιστή, διότι προσφέρεται ευκολία στη ανάπτυξη, τοποθέτηση και επεκτασιμότητα, όταν ο βασικός κώδικας είναι μικρός. Ωστόσο, καθώς το μέγεθος της εφαρμογής και της οργάνωσης αυξάνεται, το ίδιο συμβαίνει και στα διαφορετικά στρώματα της εφαρμογής. Αν δεν δοθεί μεγάλη προσοχή στην αρχιτεκτονική και στη ποιότητα του κώδικα, είναι πολύ πιθανό ότι η ποιότητα των διαφορετικών στρωμάτων να επιδεινωθεί. Αυτή η επιδείνωση μπορεί να συμβαίνει λόγω των επιχειρησιακών απαιτήσεων και του μειωμένου διαθέσιμου χρόνου, το οποίο οδηγεί τους προγραμματιστές να δίνουν λύσεις που δεν είναι οι βέλτιστες. Αυτές οι μη βέλτιστες λύσεις θα πρέπει να αναπροσαρμοστούν αλλά ο χρόνος για αναπροσαρμογή δεν είναι πάντα διαθέσιμος, το οποίο οδηγεί τις βραχυπρόθεσμες κακές λύσεις να γίνονται μακροπρόθεσμες μόνιμες λύσεις.

Καθώς το μέγεθος του κώδικα αυξάνεται και η ποιότητα αυτού επιδεινώνεται, γίνεται όλο δυσκολότερο να προστεθούν νέα χαρακτηριστικά και να τροποποιηθούν-βελτιωθούν οι παλιότερες λειτουργίες, επειδή ο προγραμματιστής πρέπει να βρει τη σωστή θέση να εφαρμόσει όλες αυτές τις αλλαγές. Αυτό έχει ως αποτέλεσμα βραδύτερους κύκλους ανάπτυξης. Ο κύκλος ανάπτυξης ενός νέου χαρακτηριστικού μπορεί να επιβραδυνθεί ακόμα περισσότερο καθώς οι αλλαγές μπορούν επηρεάζουν πολλά σημεία. Το αντίκτυπο μιας αλλαγής μπορεί να είναι δύσκολο να κατανοηθεί [5]. Ο προγραμματιστής μπορεί να πιστεύει ότι η αλλαγή είναι μικρή, αλλά στην

πραγματικότητα μπορεί να επηρεάσει πολλαπλά σημεία. Αυτό οδηγεί σε μια κατάσταση όπου απαιτείται εκτενής χειρωνακτική δοκιμή και οι κύκλοι δοκιμών παλινδρόμησης μπορούν έτσι να γίνουν μεγάλοι. Όλα αυτά συσσωρεύονται και κάνουν τη διαδικασία απελευθέρωσης νέων λειτουργιών αργή.

Ένα από τα μειονεκτήματα μιας μεγάλης μονολιθικής εφαρμογής είναι ότι χρειάζεται πολύς χρόνος να εξοικειωθεί ο προγραμματισμός με τον μεγάλο κώδικα. Χρειάζεται χρόνος για νέους προγραμματιστές να κατανοήσουν το μεγάλο κώδικα και δεν βρίσκουν εύκολα το σωστό μέρος για να εφαρμόσουν τις αλλαγές τους. Αυτό μπορεί να αποφευχθεί διατηρώντας τη σπονδυλωτότητα στα στρώματα και συνεχώς να διαγράφουν σημεία του κώδικα, τα οποία δεν έχουν λειτουργικότητα πλέον λόγω νέων χαρακτηριστικών, με σκοπό να τον κρατήσουν καθαρό. Ωστόσο, αν οι συνεχές δοκιμές δεν είναι εφικτές λόγω χρόνου, οι προγραμματιστές μπορεί να φοβούνται να επαναπροσδιορίζουν τον κώδικα, καθώς οι αλλαγές τους μπορούν να επηρεάσουν πολλά σημεία. Αυτό έχει σαν αποτέλεσμα ο κώδικας, εν τέλει, να περιέχει άχρηστες γραμμές και κατ' επέκταση να μεγαλώνει άσκοπα, οι νέοι προγραμματιστές να μη μπορούν να τον κατανοήσουν και να συνεχίζεται ατέρμονα ένας φαύλος κύκλος.

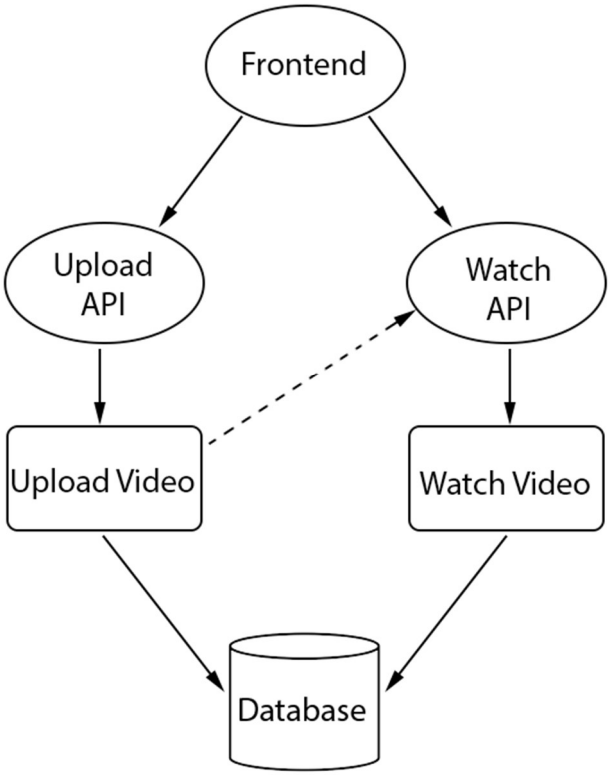
Τα καθαρά σύνορα των αρθρωμάτων εντός του μονολιθικού κώδικα μπορεί να είναι δύσκολο να επιτευχθούν κατά την ανάπτυξη. Οι γλώσσες προγραμματισμού παρέχουν κάποια εργαλεία για προγραμματιστές για να διασφαλιστεί η αρθρωτότητα και η χαλαρή σύζευξη στον μονολιθικό κώδικα βάσης. Για παράδειγμα, στη Java είναι δυνατή η διασφάλιση κάποιων ορίων χρησιμοποιώντας τα πακέτα και την ορατότητα των κατηγοριών και των μεθόδων. Ωστόσο, η παραβίαση αυτών των ορίων μπορεί να συμβεί εύκολα, διότι αυτά τα σύνορα είναι προσβάσιμα από τους προγραμματιστές.

2.4 Αρχιτεκτονική Μικροϋπηρεσιών

Σε αντίθεση με την μονολιθική αρχιτεκτονική έρχεται η αρχιτεκτονική των μικροϋπηρεσιών να προτείνει μια τελείως διαφορετική λογική ως προς τον σχεδιασμό ανάπτυξης εφαρμογών. Η διατριβή προσπαθεί να συγκρίνει τις δύο αρχιτεκτονικές στην ίδια βάση, για αυτό το σκοπό αναπτύσσεται η ίδια εφαρμογή και με τις δύο προσεγγίσεις. Αυτή η σύγκριση, έχει ως στόχο, την εξαγωγή συμπερασμάτων ως προς τους υπολογιστικούς πόρους που μπορούν να εξοικονομηθούν μέσω της αρχιτεκτονικής των μικροϋπηρεσιών και κατ' επέκταση τη μείωση του κόστους.

Η αρχιτεκτονική των μικροϋπηρεσιών είναι μια καινούρια αρχιτεκτονική, η οποία έχει γίνει δημοφιλής τα τελευταία χρόνια. Οι μικροϋπηρεσίες είναι μικρές υπηρεσίες που συμβάλλουν

στη δημιουργία μια μεγαλύτερης εφαρμογής. Ένας κανόνας ο οποίος θα μπορούσε να οριστεί για το μέγεθος μιας μικροϋπηρεσίας είναι ότι αυτή η εφαρμογή θα μπορούσε να ξαναγραφτεί μέσα σε δύο εβδομάδες [9]. Ο μικρός κύκλος ανάπτυξης της εφαρμογής αποτελεί βασικό στοιχείο των μικροϋπηρεσιών. Για να γίνει κατανοητή η αρχιτεκτονική των μικροϋπηρεσιών, μια μικροϋπηρεσία θα μπορούσε να διαχειριστεί το ανέβασμα βίντεο και μια άλλη μικροϋπηρεσία τη αναπαραγωγή των παραπάνω βίντεο. Η αρχική σελίδα της εφαρμογής (Frontend) καλεί ανάλογα με το αίτημα του χρήστη τη κάθε μικροϋπηρεσία ξεχωριστά. Επιπρόσθετα η μικροϋπηρεσία που ανεβάζει τα βίντεο μπορεί να καλέσει την μικροϋπηρεσία που παρακολουθεί τα βίντεο για να ελέγξει αν το βίντεο που ανεβαίνει είναι κατάλληλο για αναπαραγωγή. Το παράδειγμα αυτό απεικονίζεται και στο παρακάτω σχήμα για την καλύτερη κατανόηση. Αυτές οι δύο υπηρεσίες έχουν ξεχωριστό και αυτόνομο κώδικα, και η επικοινωνία μεταξύ τους γίνεται πάντα μέσω API.



Εικόνα 2-3: Μια τυπική αρχιτεκτονική μικροϋπηρεσιών

Αυτές οι υπηρεσίες διατηρούν το μέγεθος μικρό με σκοπό η λειτουργία τους να είναι συγκεκριμένη και σαφής. Επιπλέον, οι μικροϋπηρεσίες πρέπει πάντα να συμμορφώνονται με την αρχή της ενιαίας ευθύνης (SRP). Το SRP σημαίνει «συγκέντρωση των πράξεων που αλλάζουν για τον ίδιο λόγο και διαχωρισμός των πράξεων που αλλάζουν για διάφορους λόγους» [7]. Λόγω των παραπάνω, μέσω της αρχιτεκτονικής προσφέρεται ευελιξία στην ομάδα των προγραμματιστών, οι

οποίοι έχουν τη δυνατότητα να χωρίζουν την μεγάλη εφαρμογή σε μικρότερες, με σκοπό τη καλύτερη κατανόηση του κώδικα και την ευκολότερη ανάπτυξη νέων δυνατοτήτων χωρίς να κινδυνεύουν να επηρεάσουν άθελά τους άλλα χαρακτηριστικά της εφαρμογής.

Ωστόσο, η διάλυση μια μεγάλης εφαρμογής σε πολλές μικρότερες υπηρεσίες δημιουργεί επίσης πολλές διασυνδέσεις REST οι οποίες με τη σειρά τους μπορούν να εισάγουν προβλήματα επιδόσεων εάν οι υπηρεσίες είναι υπερβολικά αραιωμένες. Εάν μία μικροϋπηρεσία πρέπει να περάσει από πολλές κλήσεις υπηρεσιών, ο χρόνος εκτέλεσης των απομακρυσμένων κλήσεων αυξάνεται με αποτέλεσμα μια απλή διαδικασία να διαρκεί περισσότερο χρόνο από ότι είναι αναγκαίο.

Κατά την ανάπτυξη μικροϋπηρεσιών, ο στόχος είναι να υπάρχουν υπηρεσίες που είναι χαλαρά συνδεδεμένες και εξαιρετικά συναινετικές [9]. Χαλαρή σύζευξη σημαίνει ότι οι υπηρεσίες δεν πρέπει να γνωρίζουν οτιδήποτε για τα εσωτερικά των άλλων υπηρεσιών. Αυτό επιτυγχάνεται με τις μικροϋπηρεσίες καθώς έχουν σαφή όρια από τη φύση και επικοινωνούν μόνο μέσω διεπαφής που εκδίδει κάθε μικροϋπηρεσία. Η συνοχή μπορεί να περιγραφεί ως στεγανότητα σχετικών χαρακτηριστικών σε διαφορετικές ενότητες. Όταν χωρίζονται οι μικροϋπηρεσίες σωστά συμμορφώνονται με το SRP και έχουν ένα ενιαίο σκοπό που λειτουργούν. Εάν αυτές οι ιδιότητες εφαρμόζονται στις μικροϋπηρεσίας, αυτό σημαίνει ότι οι μικροϋπηρεσίας είναι επίσης πολύ συνεκτικές.

Προκειμένου να επιτευχθεί χαλαρή σύζευξη με την αρχιτεκτονική μικροϋπηρεσιών, μπορεί να έχει νόημα να αντιγράψουμε κομμάτια από τον κώδικα μιας μικροϋπηρεσίας σε μια άλλη. Συνήθως, οι προγραμματιστές έχουν διδαχθεί την αρχή DRY (μην επαναλάβετε τον εαυτό σας). Η DRY δηλώνει ότι ο ίδιος κώδικας δεν πρέπει να επαναλαμβάνεται σε διαφορετικά σημεία και πρέπει αυτός να επαναχρησιμοποιηθεί. Αυτή η αρχή αποτελεί μια καλή συμβουλή για τις μικροϋπηρεσίες, αλλά όταν πολλαπλές μικροϋπηρεσίες χρησιμοποιούν κοινόχρηστο κώδικα, μπορούν να προκύψουν κοινά προβλήματα. Εάν μια υπηρεσία απαιτεί αλλαγή σε κοινό κώδικα σημαίνει ότι όλες οι υπηρεσίες που χρησιμοποιούν την ίδια κοινόχρηστη βιβλιοθήκη πρέπει να ενημερωθούν και να αναπτυχθούν. Αυτό σημαίνει ότι οι υπηρεσίες είναι τώρα σφιχτά συζευγμένες. Η κατάσταση μπορεί να λυθεί με την επανάληψη του κώδικα. Δίνει τη ελευθερία για κάθε υπηρεσία να είναι ανεξάρτητη .

Η χαλαρή σύζευξη, η υψηλή συνοχή, η SRP και η αρθρωτότητα θεωρούνται καλές αρχές για ανάπτυξη λογισμικού με αρχιτεκτονική μικροϋπηρεσιών. Αυτές οι αρχές και τα πρότυπα μπορούν να χρησιμοποιηθούν από οποιαδήποτε καλά σχεδιασμένη εφαρμογή, αλλά με την αρχιτεκτονική των μικροϋπηρεσιών, είναι πιο πιθανό αυτές οι αρχές να παραμείνουν, κατά την

εξέλιξη ανάπτυξης του κώδικα, λόγω της φύσης των μικροϋπηρεσιών. Η αρχιτεκτονική των μικροϋπηρεσιών έχει δύο ιδιότητες που καθιστούν τη παραπάνω επίτευξη εφικτή, πρώτον ξεκάθαρη ιδιοκτησία του κώδικα και δεύτερον μικρές βάσεις κώδικα. Η κυριότητα σημαίνει ότι συγκεκριμένη ομάδα είναι υπεύθυνη για την μικροϋπηρεσία και οι μικρές βάσεις κώδικα είναι ευκολότερα διαχειρίσιμες.

Λόγω της χαλαρής σύζευξης, υψηλής συνοχής, SRP και αρθρωτότητας, οι κύκλοι ανάπτυξης μπορούν να είναι σημαντικά πιο γρήγοροι από μια παραδοσιακή μονολιθική ανάπτυξη εφαρμογής [7]. Τροποποίηση λειτουργιών και προσθήκη νέων χαρακτηριστικών μέσα σε μια εφαρμογή με την αρχιτεκτονική των μικροϋπηρεσιών, γίνεται σχετικά γρήγορα καθώς οι ίδιες οι υπηρεσίες είναι μικρές και πιο κατανοητές για τους προγραμματιστές. Η πολυπλοκότητα υπάρχει κυρίως στην επικοινωνία από το εσωτερικών των μικροϋπηρεσιών προς το περιβάλλον τους.

Η πολυπλοκότητα των μικροϋπηρεσιών έγκειται στις διασυνδέσεις μεταξύ τους [4]. Οι ίδιες οι υπηρεσίες λόγω του μικρού μεγέθους γίνονται εύκολα αντιληπτές από τον προγραμματιστή ποιος είναι ο σκοπός τους. Η δυσκολία έγκειται στο γεγονός όταν αυτές οι υπηρεσίες πρέπει να συνδυαστούν και να επικοινωνήσουν με σκοπό την δημιουργία ενός άλλου αποτελέσματος. Το παραπάνω αποτελεί τεράστιο εμπόδιο στον εντοπισμό σφάλματος σε κλήσεις που περνούν από πολλές υπηρεσίες. Αυτή η λειτουργική πολυπλοκότητα είναι δύσκολο να διαχειριστεί και απαιτεί πολλές τεχνικές ικανότητες από τις ομάδες ανάπτυξης ακόμα και με τα εργαλεία που υπάρχουν στη διάθεσή τους.

Οι μικροϋπηρεσίες παρέχουν πολύ λεπτή και ευέλικτη κλίμακα [2]. Διαφορετικές υπηρεσίες μπορούν να χρειαστούν διαφορετική κλιμάκωση. Μια υπηρεσία ίσως χρειαστεί οριζόντια κλίμακα ενώ μια άλλη απαιτεί κατακόρυφη κλιμάκωση. Η οριζόντια κλιμάκωση σημαίνει προσθήκη περισσότερων μηχανημάτων που εξυπηρετούν την μικροϋπηρεσία. Η κατακόρυφη κλιμάκωση σημαίνει ότι προστίθενται περισσότεροι πόροι στα ίδια υπάρχοντα μηχανήματα. Και οι δύο κλιμακώσεις μπορούν να αυτοματοποιηθούν ανάλογα με το φόρτο εργασίας, αλλά η κάθετη κλιμάκωση δεν είναι τόσο ελαστική όσο η οριζόντια επειδή η προσθήκη ή η αφαίρεση της χωρητικότητας σε μια υπάρχουσα εγκατάσταση απαιτεί χρόνο αναμονής [10]. Με την αρχιτεκτονική μικροϋπηρεσιών υπάρχει η δυνατότητα κλιμάκωσης κάθε υπηρεσίας ξεχωριστά και ανάλογα με τις ανάγκες τους, δηλαδή είτε με οριζόντια είτε με κάθετη κλιμάκωση .

Ένα από τα οφέλη που προσφέρουν οι μικροϋπηρεσίες ως προς την κλιμάκωση είναι η διασπορά του κινδύνου μέσω αυτής. Οι πιο κρίσιμες υπηρεσίες μπορούν να μοιραστούν σε πολλούς κεντρικούς υπολογιστές, σε φυσικά μηχανήματα ή ακόμη και σε κέντρα δεδομένων. Με

αυτό το είδος προσέγγισης, ο χρόνος διακοπής των κρίσιμων υπηρεσιών μπορεί να ελαχιστοποιηθεί.

Η αρχιτεκτονική μικροϋπηρεσιών επιτρέπει την υλοποίηση πολύγλωσσου κώδικα. Οι ομάδες ανάπτυξης μικροϋπηρεσιών μπορούν να κάνουν ανεξάρτητες επιλογές η μια από την άλλη, αναλόγως πάντα με τις τεχνολογικές προκλήσεις που πρέπει να αντιμετωπίσουν και το σκοπό δημιουργίας της μικροϋπηρεσίας τους. Φυσικά, πρέπει να υπάρχουν ορισμένοι περιορισμοί, προκειμένου να περιοριστεί ο αριθμός των γλωσσών στην εφαρμογή. Μια δυνατότητα είναι να χρησιμοποιηθεί ένα υποσύνολο πολυγλωσσικής προσέγγισης, όπου μόνο οι γλώσσες που εκτελούν την ίδια εικονική μηχανή να επιτρέπονται [15]. Ακόμη και αν ο αριθμός των γλωσσών προγραμματισμού είναι περιορισμένος, η πολύγλωσση προσέγγιση δίνει στις ομάδες καλύτερα εργαλεία για την αντιμετώπιση των τεχνικών προβλημάτων της εφαρμογής τους.

Το σημαντικότερο πλεονέκτημα που προσφέρει η πολυγλωσσική προσέγγιση στην αρχιτεκτονική των μικροϋπηρεσιών είναι η χρήση διαφορετικών βάσεων δεδομένων ανάλογα με τις ανάγκες κάθε μικροϋπηρεσίας [11]. Τα δεδομένα μιας υπηρεσίας μπορεί να διαφέρουν πολύ από τα δεδομένα μιας άλλης υπηρεσίας. Για παράδειγμα, μία υπηρεσία μπορεί να έχει δεδομένα που ταιριάζουν απόλυτα σε ένα σύστημα διαχείρισης σχεσιακής βάσης δεδομένων (RDBMS), ενώ η άλλη υπηρεσία έχει δεδομένα που ταιριάζουν καλύτερα σε μια βάση δεδομένων NoSQL.

Οι μικροϋπηρεσίες εκθέτουν τα όρια μέσα στην εφαρμογή, έχοντας πολλές υπηρεσίες με σαφείς διεπαφές. Αυτό διευκολύνει τη δοκιμή επειδή υπάρχουν περισσότερες επιλογές για το πού να γίνει η δοκιμή και πώς να την δοκιμάσουν. Οι μικρές και αρθρωτές υπηρεσίες είναι εύκολο να δοκιμαστούν επειδή δεν υπάρχουν πολλές εξαρτήσεις που πρέπει να αρχικοποιηθούν. Επίσης, συνεκτικές υπηρεσίες που εφαρμόζουν οριοθετημένα περιβάλλοντα σημαίνουν ότι η ανάγκη για επαναλαμβανόμενες δοκιμές μειώνονται, διότι οι επιπτώσεις των αλλαγών περιορίζονται κυρίως στο εσωτερικό της υπηρεσίας [3].

Οι μικροϋπηρεσίες είναι μικρές εξ ορισμού, παρόλο που δεν υπάρχει σαφής κανόνας για το πόσο μικρές θα πρέπει να είναι [6]. Ωστόσο, θα πρέπει να είναι αρκετά μικρές, ώστε μια ομάδα να μπορεί να έχει την ευθύνη ολόκληρης της μικροϋπηρεσίας. Τα μεγέθη ομάδας μπορεί να διαφέρουν, αλλά ένας μέγιστος αριθμός μελών της ομάδας θεωρείται ότι είναι περίπου δώδεκα άνθρωποι με σκοπό τη ορθή επικοινωνία μεταξύ τους [9]. Όταν το μέγεθος της ομάδας ή το μέγεθος της υπηρεσίας είναι πάρα πολύ μεγάλο, τότε σπάσιμο της υπηρεσίας σε δύο ή περισσότερα μικρότερες πρέπει να ληφθεί υπόψιν. Όταν η υπηρεσία σπάσει σε δύο υπηρεσίες, θα μπορούσε να σημαίνει επίσης να χωρίσουμε την ομάδα σε μικρότερες ομάδες.

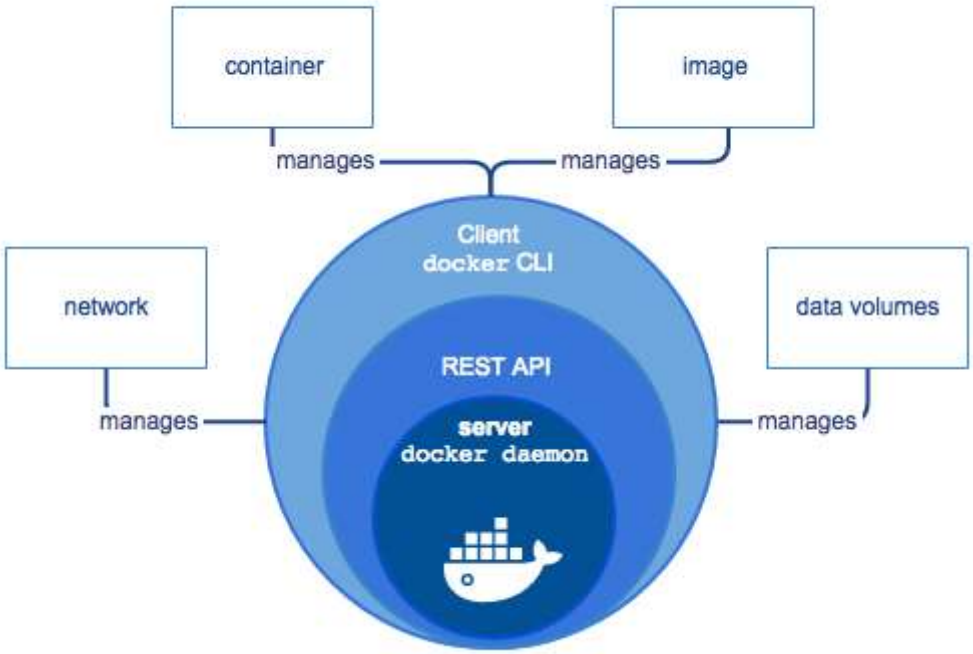
2.5 Docker

Κατά την διάρκεια της έρευνας, κατέστη γρήγορα αντιληπτό, ότι οι έννοιες αρχιτεκτονική των μικροϋπηρεσιών και πλατφόρμα docker είναι αλληλένδετα συνδεδεμένες, με αποτέλεσμα να αποτελεί μονόδρομο η ανάπτυξη της εφαρμογής στην παραπάνω πλατφόρμα. Αυτή η ανάπτυξη δεν αποτέλεσε πρόβλημα, αντιθέτως το docker παρέχει διάφορα εργαλεία, όπως είναι το docker hub και το docker stats, ως προς την ανάπτυξη και την μέτρηση των υποδοχέων (container) αντίστοιχα, με σκοπό την διευκόλυνση του χρήστη ή και του προγραμματιστή.

Το docker είναι μια πλατφόρμα ανοικτού κώδικα που εκτελεί εφαρμογές και καθιστά ευκολότερη την ανάπτυξη και τη διανομή τους. Οι εφαρμογές που είναι ενσωματωμένες στο docker είναι συσκευασμένες με όλες τις υποστηρικτικές εξαρτήσεις σε μια τυποποιημένη μορφή που ονομάζεται υποδοχέας. Αυτοί οι υποδοχείς συνεχίζουν να τρέχουν με ένα απομονωμένο τρόπο πάνω από τον πυρήνα του λειτουργικού συστήματος αυτόνομα και ανεξάρτητα. Το επιπλέον στρώμα πάνω από το λειτουργικό μπορεί να επηρεάσει την απόδοση. Ωστόσο, η τεχνολογία των υποδοχέων υπάρχει τα τελευταία δέκα χρόνια, αλλά το docker είναι γενικά μια καινούρια τεχνολογία, η οποία αποτελεί μια καινοτομία μεταξύ των σημαντικότερων, καθώς συνδυάζει νέες δυνατότητες και τεχνολογίες, που έχουν προηγηθεί. Παρέχει τη δυνατότητα δημιουργίας και ελέγχου των υποδοχέων, στους οποίους οι εφαρμογές μπορούν να συσκευάζονται εύκολα σε υποδοχείς με ελάχιστους υπολογιστικούς πόρους σε σύγκριση με τους πόρους που θα χρησιμοποιούσε η ίδια εφαρμογή εάν αναπτυσσόταν με τον παραδοσιακό τρόπο ανάπτυξης, π.χ. σε μονολιθική αρχιτεκτονική. Αυτές οι εικονικές εφαρμογές μπορούν εύκολα να δουλέψουν σχεδόν σε οποιαδήποτε συσκευή χωρίς καμία αλλαγή. Επιπλέον, το docker μπορεί να προσφέρει περισσότερες εικονικές καταστάσεις από άλλες καινοτομίες, στον ίδιο εξοπλισμό. Επιπρόσθετα, το docker μπορεί εύκολα να συνεργαστεί με εργαλεία τρίτων, που βοηθούν στην εύκολη ανάπτυξη και διαχείριση των υποδοχέων, και έχουν την ικανότητα να αναπτυχθούν εύκολα στο υπολογιστικό νέφος. Συμπερασματικά, όλα τα παραπάνω τοποθετούν την τεχνολογία docker στη κορυφή των καινοτομιών.

Υπάρχουν τέσσερα κύρια συστατικά του docker, ο docker πελάτης και διακομιστής, οι εικόνες και οι υποδοχές. Αυτές οι έννοιες θα εξηγηθούν λεπτομερώς παρακάτω. Το docker μπορεί να εξηγηθεί ως πελάτης και διακομιστής εφαρμογής, όπως απεικονίζεται στην παρακάτω εικόνα. Ο διακομιστής παίρνει το αίτημα από τον πελάτη με σκοπό να το επεξεργαστείτε αναλόγως. Το πλήρες πρωτόκολλο μεταφορά κατάστασης (RESTful API) και μια δυαδική γραμμή εντολών πελάτη παρέχονται από το docker. Docker daemon/ διακομιστής και ο πελάτης μπορούν να

αναπτυχθούν στο ίδιο μηχάνημα ή ένας πελάτης μπορεί να συνδεθεί με έναν απομακρυσμένο διακομιστή, που τρέχει σε άλλο μηχάνημα.



Εικόνα 2-4: Μοντέλα Υπηρεσιών στο υπολογιστικό νέφος

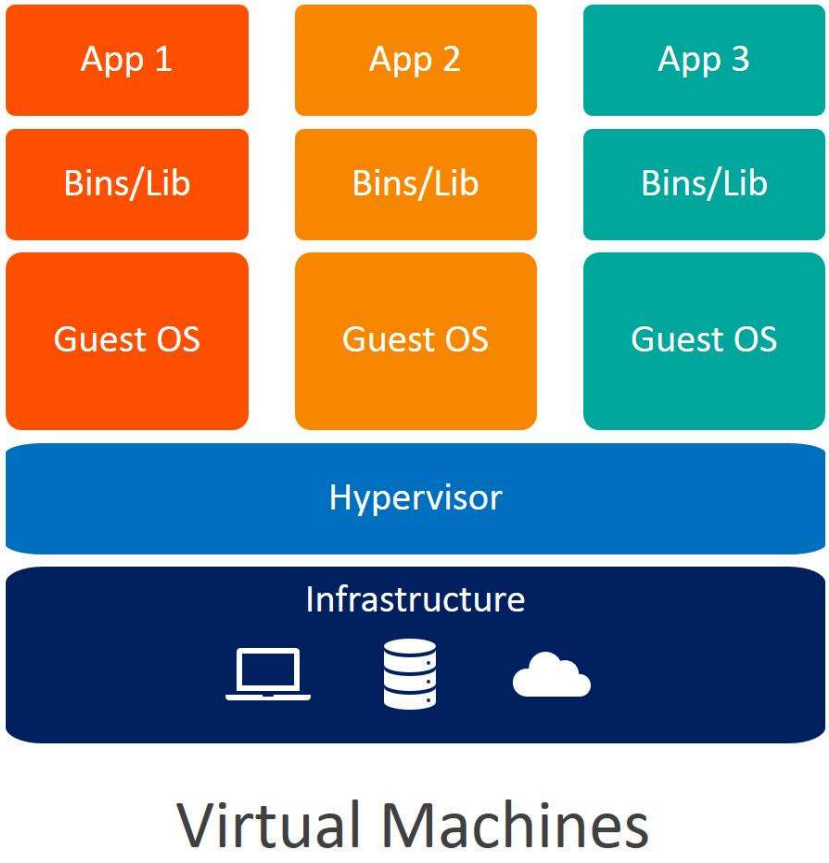
Υπάρχουν δύο μέθοδοι για την δημιουργία μιας εικόνας docker. Η πρώτη είναι να δημιουργηθεί μια εικόνα χρησιμοποιώντας ένα πρότυπο μόνο για ανάγνωση. Η θεμελίωση κάθε εικόνας αποτελεί τη βασική εικόνα. Οι εικόνες λειτουργικού συστήματος αποτελούν βασικές εικόνες, όπως είναι και η εικόνα Ubuntu. Οι παραπάνω εικόνες δημιουργούν ένα υποδοχέα με τη δυνατότητα να τρέχουν πλήρως ένα λειτουργικό σύστημα. Η βασική εικόνα μπορεί επίσης να δημιουργηθεί από το μηδέν. Οι απαιτούμενες εφαρμογές μπορούν να προστεθούν στη βασική εικόνα, αλλά είναι απαραίτητο να δημιουργηθεί μια νέα εικόνα. Η διαδικασία δημιουργίας μιας νέας εικόνας ονομάζεται «Πραγματοποιώντας μια αλλαγή». Η δεύτερη μέθοδος είναι η δημιουργία ενός αρχείο docker. Το αρχείο docker περιέχει μια λίστα με οδηγίες και όταν εκτελείται η εντολή «docker build» από το τερματικό ακολουθεί όλες τις οδηγίες που δίνονται από το παραπάνω αρχείο και δημιουργεί την εικόνα. Αυτός είναι ο αυτοματοποιημένος τρόπος δημιουργίας μια εικόνα.

Οι εικόνες του docker τοποθετούνται στα μητρώα docker, και λειτουργούν όπως οι αποθήκες πηγαίου κώδικα όπου η εικόνα μπορεί να ωθηθεί ή να τραβηχτεί από μία μόνο πηγή. Υπάρχουν δύο είδη μητρώων, τα δημόσια και τα ιδιωτικά. Docker Hub καλείται ένα δημόσιο

μητρώο όπου ο καθένας μπορεί να κατεβάσει τις διαθέσιμες εικόνες και να ανεβάσει τις δικές τους εικόνες χωρίς να χρειάζεται να δημιουργήσει μια εικόνα από το μηδέν. Οι εικόνες μπορούν να διανεμηθούν σε μια συγκεκριμένη περιοχή (δημόσιο ή ιδιωτικό) χρησιμοποιώντας τη λειτουργία του docker hub.

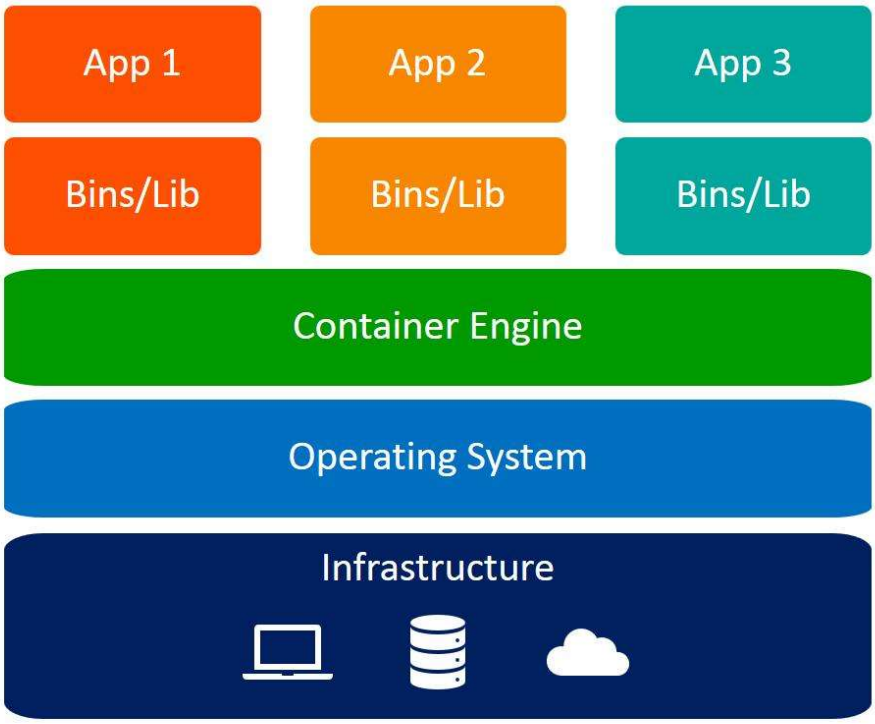
Η εικόνα του docker δημιουργεί έναν υποδοχέα. Οι υποδοχείς περιέχουν το σύνολο των εργαλείων που απαιτούνται για μια εφαρμογή, έτσι ώστε η εφαρμογή να μπορεί να λειτουργήσει απολύτως μεμονωμένα. Για παράδειγμα, υποθέστε ότι υπάρχει μια εικόνα του Ubuntu OS με SQL SERVER. Όταν αυτή η εικόνα εκτελείται με εντολή τρέχουσας λειτουργίας docker, δημιουργεί έναν υποδοχέα και θα τρέξει SQL SERVER σε λειτουργικό σύστημα Ubuntu.

Η εικονικοποίηση είναι μια παλιά έννοια, η οποία έχει χρησιμοποιηθεί και χρησιμοποιείται στο υπολογιστικό νέφος, εφόσον η Υποδομή-ως-Υπηρεσία (IaaS) έγινε αποδεκτή σα μια κύρια τεχνική για τη σύσταση του συστήματος, την παροχή πόρων και την μίσθωση διαφορετικών μονάδων. Οι εικονικοί πόροι διαδραματίζουν κυρίαρχο ρόλο στην επίλυση των προβλημάτων χρησιμοποιώντας την τεχνική πυρήνα του υπολογιστικού νέφους. Το παρακάτω σχήμα δείχνει την αρχιτεκτονική της εικονικής μηχανής.



Εικόνα 2-5: Αρχιτεκτονική εικονικών μηχανών

Το Hypervisor, το οποίο είναι υπεύθυνο για την εικονικοποίηση, βρίσκεται μεταξύ οικοδεσπότη και φιλοξενούμενου συστήματος. Είναι μια εικονική πλατφόρμα και χειρίζεται περισσότερα από ένα λειτουργικά συστήματα στο διακομιστή. Λειτουργεί μεταξύ του λειτουργικού συστήματος και του επεξεργαστή-πόρων.



Εικόνα 2-6: Αρχιτεκτονική υποδοχέων docker

Η παραπάνω εικόνα απεικονίζει την αρχιτεκτονική υποδοχέων docker. Οι υποδοχείς των Linux διαχειρίζονται από το εργαλείο docker και αποτελεί μία μέθοδο εικονικοποίηση επιπέδου λειτουργικού συστήματος. Το παραπάνω σχήμα δείχνει ότι σε έναν κεντρικό υπολογιστή ελέγχου υπάρχουν πολλοί υποδοχείς Linux, τα οποία είναι απομονωμένα. Πόροι όπως το δίκτυο, η μνήμη, ο επεξεργαστής, διατίθενται και διαμοιράζονται από το Linux πυρήνα.

Σύμφωνα με τον Waldspurger (2002), στους υποδοχείς του Linux, μια αρχιτεκτονική είναι η διαχείριση της CPU και η διανομή των πόρων πιο αποτελεσματικά. Σε οποιοδήποτε παράδειγμα Hyper-V ή VMWare, λόγω των γενικών επιβαρύνσεων, δεν είναι εύκολο να τρέξει περισσότερες από δέκα εικονικές μηχανές [14]. Σε ένα μεγάλο βαθμό, αυτό το πρόβλημα επιλύθηκε από τους υποδοχείς. Οι υποδοχείς χρησιμοποιούν μόνο τους πόρους που χρειάζονται για τις υπηρεσίες ή

εφαρμογές. Ακόμη και σε ένα αδύναμο μηχάνημα, μπορούν να εκτελεστούν πάνω από πενήντα αιτήσεις δημιουργίας υποδοχέων. Για παράδειγμα, ας υποθέσουμε ότι ένας οργανισμός παρέχει μηνύματα ηλεκτρονικού ταχυδρομείου υπηρεσιών ασφαλείας. Οι κύριες λειτουργίες αυτών των υπηρεσιών είναι να ελέγχουν τα μηνύματα ηλεκτρονικού ταχυδρομείου για ιούς, ανεπιθύμητα μηνύματα και κακόβουλο λογισμικό. Επιπλέον, θα μπορούσε να μεταφέρει μηνύματα στον πράκτορα, τα αρχεία καταγραφής κ.α.. Κυρίως σε αυτές τις περιπτώσεις, δεν υπάρχει χρήση οποιωνδήποτε συνδεδεμένων εξαρτήσεων ή βιβλιοθηκών επιπέδου λειτουργικού συστήματος. Ως εκ τούτου, αξίζει τον κόπο να δημιουργήσει για κάθε λειτουργία ένα ξεχωριστό υποδοχέα ή και παραπάνω εάν οι ανάγκες το απαιτούν αντί για εικονικές μηχανές.

Σε πολλές επιχειρήσεις χρησιμοποιούνται εικονικές μηχανές για την εκτέλεση δοκιμών διάφορων στοιχείων. Σε αυτή τη διαδικασία, πολλοί πόροι επεξεργαστικής δύναμης και χώρος μνήμης καταναλώνεται. Ωστόσο, η τεχνολογία των υποδοχέων παρέχει εγγύηση στους χρήστες τους ότι η υπέρβαση του φόρτου εργασίας δεν θα επηρέαζε την αποτελεσματικότητα των πόρων. Η εγκατάσταση του υποδοχέα διαρκεί λιγότερο χρόνο σε σύγκριση με τις εικονικές μηχανές, έτσι ώστε η προσαρμοστικότητα των υποδοχέων να είναι πολύ υψηλότερη από αυτή των εικονικών μηχανημάτων.

Τόσο το Docker όσο και το OpenVZ βρίσκονται υπό εξέταση όσον αφορά τις πτυχές ασφαλείας τους. Όταν η απομόνωση μειώνεται, επηρεάζεται άμεσα η ασφάλεια, η οποία μειώνεται εξίσου. Οι διαχειριστές των Linux μπορούν εύκολα να αποκτήσουν πρόσβαση στους υποδοχείς καθώς οι υποδοχείς χρησιμοποιούν τον ίδιο πυρήνα και το λειτουργικό σύστημα. Η απομόνωση του docker δεν είναι τόσο ισχυρή όσο μιας εικονικής μηχανής, παρόλο που το docker απομονώνει τη εφαρμογή, η οποία εκτελείται στον υποδοχέα. Επιπλέον, είναι πιθανό ότι μερικές από τις οι εφαρμογές δεν θα είναι σε θέση να τρέχουν σε τεχνολογία υποδοχέα και πρέπει να λειτουργούν σε διαφορετικό λειτουργικό σύστημα.

Έχει υπάρξει έντονη ζήτηση και εξέλιξη των υποδοχέων τα τελευταία χρόνια. Το docker έχει γίνει δημοφιλές πολύ γρήγορα, λόγω των δυνατοτήτων που παρέχει. Τα βασικά πλεονεκτήματα του docker επιγραμματικά είναι η ταχύτητα, η φορητότητα, η επεκτασιμότητα, η ταχεία παράδοση/ανάπτυξη και η αποτελεσματικότερη χρήση των πόρων. Παρακάτω αναλύονται το κάθε ένα ξεχωριστά.

1. Η ταχύτητα είναι ένα από τα πιο σημαντικά πλεονεκτήματα που μπορούν να προσφέρουν οι υποδοχείς. Ο χρόνος που απαιτείται για την κατασκευή ενός υποδοχέα είναι λίγος επειδή είναι πολύ μικρός. Ανάπτυξη και δοκιμές μπορούν να γίνουν γρηγορότερα, καθώς οι υποδοχείς είναι μικροί. Οι υποδοχείς μπορούν να

ωθηθούν για έλεγχο μόλις δημιουργηθούν και στη συνέχεια από εκεί, στο περιβάλλον παραγωγής [13].

2. Αυτές οι εφαρμογές είναι κατασκευασμένες μέσα σε υποδοχείς με αποτέλεσμα να είναι εξαιρετικά φορητές. Αυτές οι φορητές εφαρμογές μπορούν να μεταφερθούν εύκολα σα μία μονάδα και η απόδοση τους παραμένει ίδια.
3. Το docker έχει την ικανότητα να αναπτυχθεί σε πολλαπλούς φυσικούς διακομιστές, διακομιστές δεδομένων και πλατφόρμες υπολογιστικού νέφους. Μπορεί επίσης να τρέχει σε κάθε μηχανή Linux. Οι υποδοχείς μπορούν εύκολα να μετακινηθούν από περιβάλλον υπολογιστικού νέφους σε τοπικό οικοδεσπότη και πίσω πάλι στο υπολογιστικό νέφος, σε μηδενικό χρόνο. Οι ρυθμίσεις μπορούν να πραγματοποιηθούν με σχετική ευκολία και η κλίμακα μπορεί απλά να ρυθμιστεί από τον χρήστη σύμφωνα με τις ανάγκες της εφαρμογής.
4. Η μορφή ενός υποδοχέα είναι τυποποιημένη με σκοπό οι προγραμματιστές και οι διαχειριστές να έχουν λιγότερο άγχος. Η ευθύνη του διαχειριστή είναι να αναπτύξει και να διατηρήσει σταθερό τον αριθμό των υποδοχέων στο διακομιστή, ενώ η ευθύνη του προγραμματιστή είναι να φροντίσει τις εφαρμογές στο εσωτερικό του υποδοχέα. Οι υποδοχείς μπορούν και λειτουργούν σε κάθε περιβάλλον καθώς έχουν όλες τις απαιτούμενες βιβλιοθήκες που είναι αναγκαίες για να λειτουργήσει η εφαρμογή. Το docker παρέχει ένα αξιόπιστο, σταθερό και βελτιωμένο περιβάλλον, με σκοπό να υπάρξουν τα αναμενόμενα αποτελέσματα, όταν θα υπάρξει μεταφορά λειτουργιών από το δοκιμαστικό σύστημα στο παραγωγικό.
5. Το docker χρησιμοποιεί τους πόρους που είναι διαθέσιμοι πιο αποτελεσματικά, επειδή δεν χρησιμοποιεί έναν hypervisor. Αυτός είναι ο λόγος, ο οποίος περισσότεροι υποδοχείς μπορούν να τρέξουν σε έναν μόνο κεντρικό υπολογιστή σε σύγκριση με τις εικονικές μηχανές. Η απόδοση ενός υποδοχέα είναι υψηλότερη λόγω της υψηλότερης πυκνότητας και της αποτελεσματικότερης χρήσης των πόρων.

Το Docker Compose είναι ένα εργαλείο ενορχήστρωσης για το docker που επιτρέπει να οριστεί ένα σύνολο υποδοχέων και τις αλληλεξαρτήσεις τους στη μορφή ενός αρχείου YAML. Στη συνέχεια, μπορεί να χρησιμοποιηθεί το docker compose για τη δημιουργία μέρους ή συνόλου της στοίβας των εφαρμογών κ.λπ.

Το σμήνος (swarm) docker αποτελείται από μια ομάδα φυσικών ή εικονικών μηχανών που εκτελούν την εφαρμογή docker και έχουν ρυθμιστεί με σκοπό να ενώνονται και να συνεργάζονται

σαν ένα ενιαίο σύμπλεγμα. Μόλις μια ομάδα μηχανών συγκροτηθεί, μπορούν ακόμα να εκτελεστούν οι συνηθισμένες εντολές docker, αλλά θα εκτελεστούν από τις μηχανές σαν σύμπλεγμα. Οι δραστηριότητες του συμπλέγματος ελέγχονται από έναν διαχειριστή σμήνους και οι μηχανές που έχουν ενταχθεί στο σύμπλεγμα αναφέρονται ως κόμβοι.

Το σμήνος είναι ένα εργαλείο ενορχήστρωσης των υποδοχέων, που σημαίνει ότι επιτρέπει στο χρήστη να διαχειρίζεται πολλαπλούς υποδοχείς που αναπτύσσονται σε πολλαπλές μηχανές. Ένα από τα βασικά πλεονεκτήματα που συνδέονται με τη λειτουργία ενός σμήνους είναι το υψηλό επίπεδο διαθεσιμότητας που προσφέρεται για εφαρμογές. Σε ένα σμήνος, υπάρχουν συνήθως αρκετοί κόμβοι εργαζομένων και τουλάχιστον ένας κόμβος διαχειριστή ο οποίος είναι υπεύθυνος για την αποτελεσματική διαχείριση των πόρων των κόμβων των εργαζομένων και την εξασφάλιση της αποτελεσματικής λειτουργίας του συμπλέγματος.

Τέλος ίσως το σημαντικότερο πλεονέκτημα που προσφέρει το σμήνος είναι η αυτόματη εξισορρόπηση φορτίου. Το σμήνος προγραμματίζει εργασίες χρησιμοποιώντας μια ποικιλία μεθοδολογιών για να εξασφαλίσει ότι υπάρχουν αρκετοί πόροι διαθέσιμοι για όλους τους υποδοχείς. Μέσω μιας διαδικασίας που περιγράφεται ως αυτοματοποιημένη εξισορρόπηση φορτίου, ο διαχειριστής σμήνους εξασφαλίζει ότι τα φορτία εργασίας των υποδοχέων έχουν εκχωρηθεί για να τρέχουν στον πιο κατάλληλο κεντρικό υπολογιστή για βέλτιστη απόδοση.

2.6 Συμπεράσματα θεωρητικού υπόβαθρου

Στο 2ο κεφάλαιο πραγματοποιήθηκε μια λεπτομερής έρευνα όλων των εννοιών, που χρησιμοποιούνται στην διατριβή, με στόχο την εξαγωγή συμπερασμάτων βάσει μόνο του θεωρητικού υπόβαθρου. Τα κύρια συμπεράσματα από την βιβλιογραφική επισκόπηση παρουσιάζονται παρακάτω.

Το υπολογιστικό νέφος προσφέρει εύκολη χρήση σχεδόν απεριόριστων υπολογιστικών πόρων, ανεξαρτήτως τοποθεσίας και ανθρώπινης παρέμβασης και οι πληρωμές γίνονται μόνο για τους πόρους που χρησιμοποιούνται. Επιπλέον, προσφέρεται πλήθος επιλογών, ως προς το είδος του υπολογιστικού νέφος (IaaS, Paas, SaaS), με σκοπό την ικανοποίηση κάθε πιθανής ανάγκης.

Μεγάλο κομμάτι του υπολογιστικού νέφους αποτελεί η συμφωνία σε επίπεδο υπηρεσίας (SLA) και οι σχετικές έννοιες ,δηλαδή τα SLO και SLI. Η ρητή ή σιωπηρή συμφωνία ανάμεσα στον πάροχο και στον πελάτη καθορίζεται από την λεπτή γραμμή ανάμεσα στο ποιες είναι οι προσδοκίες του πελάτη από τον πάροχο και στο κατά πόσο ο πάροχος μπορεί να μειώσει το κόστος λειτουργίας της υπηρεσίας που προσφέρει, χωρίς να μειώσει αισθητά την ικανοποίηση του πελάτη.

Η μονολιθική αρχιτεκτονική αποτελεί την κλασική μέθοδο ανάπτυξης εφαρμογών. Αρκετοί προγραμματιστές συνεχίζουν να την προτιμούν, για μικρές εφαρμογές, λόγω της εύκολης ανάπτυξης που προσφέρει. Ωστόσο, το παραπάνω μπορεί να αποτελέσει παγίδα, καθώς η συνεχής ανάπτυξη μπορεί να δημιουργήσει μεγάλο κώδικα, ο οποίος μπορεί να απαιτεί αρκετό χρόνο για την κατανόηση του από έναν άλλο προγραμματιστή, πέρα του δημιουργού του, αλλά και ο νέος κώδικας, που αναπτύσσεται, να προκαλέσει άθελά του αλλαγές σε άλλες λειτουργίες. Αυτό αναγκάζει τον κύκλο ανάπτυξης να μεγαλώνει με αποτέλεσμα η πιθανότητα λάθους να μεγαλώνει ακόμα περισσότερο. Τέλος, τα καθαρά σύνορα των αρθρωμάτων εντός του μονολιθικού κώδικα μπορεί να είναι δύσκολο να επιτευχθούν κατά την ανάπτυξη, έχοντας σαν αποτέλεσμα να μειώνονται αισθητά οι επιλογές της κλιμάκωσης και πολλές φορές να πρέπει να κλιμακωθούν και αχρείαστες λειτουργίες του προγράμματος.

Στον αντίποδα, υπάρχει η αρχιτεκτονική των μικροϋπηρεσιών. Μπορεί το επίπεδο πολυπλοκότητας να είναι υψηλότερο από την προηγούμενη αρχιτεκτονική, αλλά λόγου του μικρού μεγέθους τον υπηρεσιών, στο σύνολο είναι ευκολότερα ελεγχόμενη, αλλά και προσφέρει καθαρή διάκριση των λειτουργιών, δηλαδή γίνεται γρήγορα αντιληπτή η λειτουργικότητα του κάθε μέρους του κώδικα. Αυτό οδηγεί σε ευέλικτη κλίμακα, δίνοντας την δυνατότητα στον διαχειριστή να αυξομειώνει τους πόρους ανάλογα με τις ανάγκες την εκάστοτε στιγμή, με αποτέλεσμα την αποδοτικότερη χρήση των υπολογιστικών πόρων. Επιπρόσθετα, προσφέρεται η δυνατότητα υλοποίησης πολύγλωσσου κώδικα, και χρήση διαφορετικών βάσεων δεδομένων, κάτι που δε μπορεί να συμβεί στην μονολιθική προσέγγιση.

Το docker αποτελεί την κύρια πλατφόρμα ανάπτυξης εφαρμογών με την αρχιτεκτονικής των μικροϋπηρεσιών. Ο υποδοχέας είναι τυποποιημένος και μπορεί να τρέχει αυτόνομα και ανεξάρτητα πάνω από τον πυρήνα του λειτουργικού συστήματος, καθιστώντας τον εξαιρετικά φορητό και κατάλληλο να μπορεί να αναπτύσσεται σε πολλαπλούς φυσικούς διακομιστές. Σε αυτή την πλατφόρμα υπάρχουν πολλά χρήσιμα εργαλεία όπως είναι το docker compose, το docker hub και το σμήνος docker.

Βάσει των παραπάνω, εξάγονται συμπεράσματα ως προς το ποια είδη εφαρμογών είναι κατάλληλα για ανάπτυξη με την αρχιτεκτονική των μικροϋπηρεσιών και ποια είναι προτιμότερο να αναπτυχθούν με τον κλασικό τρόπο, δηλαδή με την μονολιθική αρχιτεκτονική. Δε συνίσταται η ανάπτυξη εφαρμογών με την αρχιτεκτονική των μικροϋπηρεσιών, στις περιπτώσεις που στην εφαρμογή απαιτούνται συναλλαγές χρονικά άμεσες και ακριβείς ή η ασφάλεια αποτελεί προτεραιότητα. Χαρακτηριστικό παράδειγμα εφαρμογών που αποφεύγει την ανάπτυξη μέσω της αρχιτεκτονικής των μικροϋπηρεσιών, αποτελούν οι τραπεζικές-χρηματιστηριακές. Επίσης, στα

ανταγωνιστικά ηλεκτρονικά παιχνίδια, λόγω της ανάγκης άμεσης ανταπόκρισης της εφαρμογής στα αντανακλαστικά των παικτών, αποφεύγεται η ανάπτυξή τους μέσω της αρχιτεκτονικής των μικροϋπηρεσιών. Αντιθέτως, εφαρμογές αναμετάδοσης βίντεο (streaming), εφαρμογές ιστού με πλήθος δυνατοτήτων και γενικότερα εφαρμογές μεγάλης κλίμακας, αλλά ανεξάρτητες χρόνου ανταπόκρισης, όπως είναι το Netflix και το Uber, μπορούν να επωφεληθούν σε μεγάλο βαθμό από την αρχιτεκτονική των μικροϋπηρεσιών. Η παρακάτω εφαρμογή που αναπτύχθηκε χάρη της διατριβής, δεν αποτελεί μεγάλης κλίμακας εφαρμογή, λόγω περιορισμένου χρόνου, αλλά είναι ανεξάρτητη χρόνου και μπορεί στο μέλλον να αναπτυχθούν επιπλέον δυνατότητες, χωρίς να υπάρχει κίνδυνος αλλοίωσης των προηγούμενων.

3 Πειραματική Ανάπτυξη

Η έρευνα και η μελέτη όλων των προηγούμενων εννοιών αποτελεί τη βάση στη σωστή κατανόηση των κεφαλαίων που ακολουθούν. Τελικός στόχος της έρευνας είναι να εξάγει τεχνοοικονομικά συμπεράσματα, μέσω των συγκρίσεων ανάμεσα στη παραδοσιακή ανάπτυξη εφαρμογών, δηλαδή την μονολιθική προσέγγιση, και στην αρχιτεκτονική των μικροϋπηρεσιών. Επιπρόσθετα, σκοπός της έρευνας είναι να ασχοληθεί με το πως μπορεί να γίνει ακόμα αποδοτικότερη η χρήση των υπολογιστικών πόρων στις μικροϋπηρεσίες.

Για να επιτευχθούν τα παραπάνω, κρίθηκε αναγκαία η ανάπτυξη της ίδιας εφαρμογής και με τις δύο αρχιτεκτονικές που μελετούνται. Πριν την ανάλυση της πειραματικής ανάπτυξης, παρουσιάζονται τα διάφορα επιπλέον εργαλεία – βιβλιοθήκες που χρησιμοποιήθηκαν κατά την διάρκεια της ανάπτυξης των εφαρμογών.

Λόγω της έλλειψης εμπειρίας σε ανάπτυξη «Full Stack», χρειάστηκε να βρεθεί ένα περιβάλλον που θα διευκολύνει την ανάπτυξη της εφαρμογής. Αυτό ήταν το Node.js, το οποίο προσφέρει την ανάπτυξη κώδικα στην πλευρά του διακομιστή, στην ίδια γλώσσα με την πλευρά του πελάτη, δηλαδή Javascript. Η βιβλιοθήκη Express συνεργάζεται καλά με το Node.js, με κύριο στόχο την δημιουργία αιτημάτων HTTP σε διαφορετικές διαδρομές URL και σε συνδυασμό με το NPM, το οποίο χρησιμοποιείται ως διαχειριστής πακέτων, αποτελεί συνήθη συνδυασμό ανάπτυξης τέτοιου είδους εφαρμογών. Η χρήση του διακομιστή NGINX, αποτέλεσε μονόδρομο στην διαχείριση των διαδρομών ανάμεσα στις μικροϋπηρεσίες, των αιτημάτων προς την βάση δεδομένων MongoDB και την επιστροφή των απαντήσεων μέσω της μορφής JSON. Οι επιλογές της βάσης και του JSON έγιναν λόγω της ευκολίας που προσφέρεται κατά την χρήση τους. Τέλος, χρησιμοποιήθηκε το εργαλείο Docker Stats με σκοπό την μέτρηση των υπολογιστικών πόρων που καταναλώνονται από τους υποδοχείς κατά την διάρκεια των δοκιμών.

3.1 Διάφορα εργαλεία – Βιβλιοθήκες

3.1.1 Node.js – Express

Το Node.js είναι ένα περιβάλλον χρόνου εκτέλεσης JavaScript ανοικτού κώδικα και διαδικτύου. Είναι ένα δημοφιλές εργαλείο για σχεδόν οποιοδήποτε είδος έργου. Το Node.js εκτελεί τη μηχανή JavaScript V8 και τον πυρήνα του Google Chrome, έξω από το πρόγραμμα περιήγησης. Αυτό επιτρέπει στο Node.js να είναι πολύ αποδοτικό.

Μια εφαρμογή Node.js εκτελείται σε μια ενιαία διαδικασία, χωρίς να δημιουργείται ένα νέο thread για κάθε αίτημα. Το Node.js παρέχει ένα σύνολο ασύγχρονων πρωτογενών εισόδων / εξόδων στην τυπική βιβλιοθήκη του, τα οποία εμποδίζουν τον αποκλεισμό του κώδικα JavaScript και γενικά, οι βιβλιοθήκες στο Node.js γράφονται χρησιμοποιώντας παραδείγματα μη αποκλεισμού, καθιστώντας τη συμπεριφορά αποκλεισμού την εξαίρεση και όχι τον κανόνα.

Όταν το Node.js χρειάζεται να εκτελέσει μια λειτουργία εισόδου / εξόδου, όπως ανάγνωση από το δίκτυο, πρόσβαση σε μια βάση δεδομένων ή το σύστημα αρχείων, αντί να αποκλείσει το thread και να χάσει τις περιόδους του CPU που περιμένουν, το Node.js θα επαναλάβει τις λειτουργίες όταν επανέλθει η απάντηση. Αυτό επιτρέπει στο Node.js να χειρίζεται χιλιάδες ταυτόχρονες συνδέσεις με ένα μόνο διακομιστή χωρίς να εισάγει το βάρος της διαχείρισης του ανταγωνισμού νήματος, το οποίο θα μπορούσε να αποτελέσει σημαντική πηγή σφαλμάτων.

Το Node.js έχει ένα μοναδικό πλεονέκτημα οι προγραμματιστές frontend, που γράφουν JavaScript για το πρόγραμμα περιήγησης, είναι πλέον σε θέση να γράψουν τον κώδικα της πλευράς του διακομιστή στην ίδια γλώσσα, χωρίς δηλαδή να χρειάζεται να μάθουν μια εντελώς διαφορετική γλώσσα [17].

Το Express είναι το πιο δημοφιλές πλαίσιο κόμβων ιστού, και είναι η υποκείμενη βιβλιοθήκη για μια σειρά άλλων δημοφιλών πλαισίων κόμβων ιστού. Παρέχει μηχανισμούς για:

- Τη δημιουργία αιτημάτων HTTP σε διαφορετικές διαδρομές URL
- Την ενσωμάτωση των μηχανισμών εμφάνισης και τη δημιουργία απαντήσεων με την εισαγωγή δεδομένων σε πρότυπα.
- Τον ορισμό των κοινών ρυθμίσεων εφαρμογών ιστού όπως τη θύρα που θα χρησιμοποιηθεί για τη σύνδεση και τη θέση των προτύπων που χρησιμοποιούνται για την απόδοση της απάντησης.
- Την προσθήκη πρόσθετης ενδιάμεσης εφαρμογής επεξεργασίας αιτημάτων σε οποιοδήποτε σημείο του αγωγού διαχείρισης αιτημάτων.

Ενώ η ίδια η Express είναι αρκετά μινιμαλιστική, οι προγραμματιστές έχουν δημιουργήσει συμβατά πακέτα ενδιάμεσου λογισμικού για να αντιμετωπίσουν σχεδόν οποιοδήποτε πρόβλημα ανάπτυξης εφαρμογών ιστού.

3.1.2 NPM

Το npm σημαίνει διαχειριστής πακέτων κόμβων. Επιτρέπει τη συνεχή διαχείριση των πακέτων node.js. Προσφέρει την ικανότητα να εγκαταστήσει, να μοιράζεται και να διαχειρίζεται τα πακέτα node.js.

Η χρήση του npm είναι ένας ακρογωνιαίος λίθος της σύγχρονης ανάπτυξης ιστού. Ωστόσο, σε συνδυασμό με το Node.js, χρησιμοποιείται ως διαχειριστής πακέτων ή εργαλείο δημιουργίας για το front-end.

3.1.3 NGINX

Το Nginx είναι ένας διακομιστής ιστού ο οποίος μπορεί επίσης να χρησιμοποιηθεί ως αντίστροφος διακομιστής μεσολάβησης-κρύπτη [18]. Αποτελεί έναν ισχυρό διακομιστή ιστού και χρησιμοποιεί μια αρχιτεκτονική βασισμένη σε γεγονότα.

3.1.4 JSON

Το JSON - συντομογραφία για το JavaScript Αντικειμενογραφική - είναι μια μορφή για την κοινή χρήση δεδομένων. Όπως υποδηλώνει το όνομά του, το JSON προέρχεται από τη γλώσσα προγραμματισμού JavaScript, αλλά είναι διαθέσιμο για χρήση από πολλές γλώσσες όπως Python, Ruby, PHP και Java.

Το JSON χρησιμοποιεί την επέκταση .json όταν βρίσκεται μόνο του. Όταν ορίζεται σε άλλη μορφή αρχείου (όπως στο .html), μπορεί να εμφανιστεί μέσα σε εισαγωγικά ως συμβολοσειρά JSON ή μπορεί να είναι ένα αντικείμενο που έχει εκχωρηθεί σε μια μεταβλητή. Αυτή η μορφή είναι εύκολη στη μετάδοση μεταξύ του διακομιστή ιστού και του προγράμματος-πελάτη ή του προγράμματος περιήγησης. Αναγνώσιμο και ελαφρύ, το JSON προσφέρει μια καλή εναλλακτική λύση αντί για την XML και απαιτεί πολύ λιγότερη μορφοποίηση [16].

3.1.5 MongoDB

Η MongoDB είναι μια σχεσιακή βάση δεδομένων NoSQL που αποθηκεύει τα δεδομένα με τη μορφή ζευγών κλειδιών-τιμών. Πρόκειται για μια βάση δεδομένων εγγράφων ανοικτού κώδικα, η οποία παρέχει υψηλή απόδοση και επεκτασιμότητα, καθώς και μοντελοποίηση δεδομένων και διαχείριση δεδομένων τεράστιων συνόλων δεδομένων σε μια επιχειρησιακή εφαρμογή.

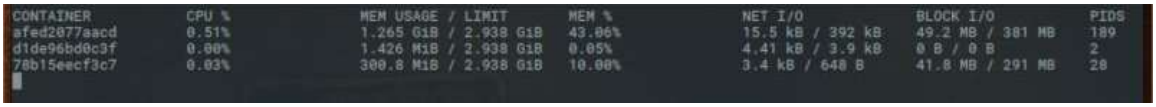
Η MongoDB παρέχει επίσης τη δυνατότητα της αυτόματης κλιμάκωσης καθώς αποτελεί μια βάση δεδομένων πολλαπλών πλατφορμών που μπορεί να εγκατασταθεί σε διαφορετικές πλατφόρμες όπως Windows, Linux κ.λπ.

3.1.6 Docker Stats

Τέλος, πρέπει να αναφερθεί ότι, για την συλλογή των πληροφοριών των δοκιμών χρησιμοποιήθηκε το docker stats. Οι δοκιμές επικεντρώθηκαν μόνο στο δείκτη «CPU %», ωστόσο, επιγραμματικά στον παρακάτω πίνακα, αναγράφεται η λειτουργικότητα του κάθε δείκτη όπως φαίνεται στην εικόνα 3-1:

Δείκτης	**Περιγραφή**
Container	Το όνομα του υποδοχέα
CPU %	Το ποσοστό χρήσης του κεντρικού επεξεργαστή που χρησιμοποιεί ο υποδοχέας
Mem Usage / Limit	Η συνολική χρήση μνήμης που χρησιμοποιεί ο υποδοχέας και η συνολική ποσότητα που επιτρέπεται να χρησιμοποιεί
Net I/O	Ο όγκος των δεδομένων που έχει στείλει και λάβει ο υποδοχέας μέσω της διεπαφής δικτύου του
Block I/O	Ο όγκος των δεδομένων που έχει διαβάσει και γράψει ο υποδοχέας από συσκευές block στον κεντρικό υπολογιστή
PIDs	Τον αριθμό των διεργασιών ή των threads που έχει δημιουργήσει ο υποδοχέας

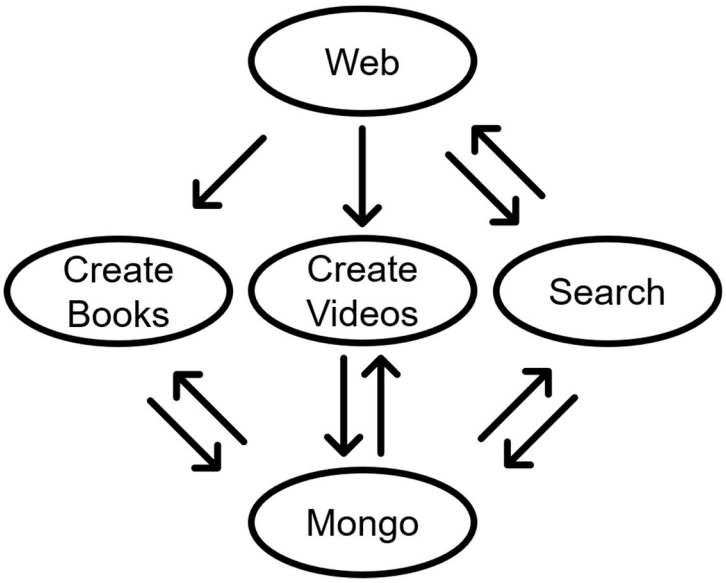
Πίνακας 3-1: Δείκτες Docker Stats



Εικόνα 3-1: Παράδειγμα χρήσης Docker stats

3.2 Ανάπτυξη εφαρμογής με την Αρχιτεκτονική των Μικροϋπηρεσιών

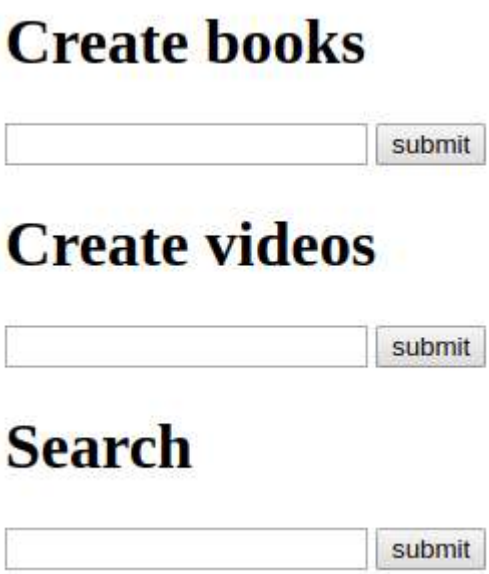
Σε αυτό το κεφάλαιο θα αναλυθούν οι μικροϋπηρεσίες που αναπτύχθηκαν χάρη της εφαρμογής. Για την ορθότερη κατανόηση της εφαρμογής και του κώδικά της, πριν την ανάλυσή του, παρουσιάζεται η κάτοψή της. Όπως γίνεται εμφανές στο σχήμα, η υπηρεσία Web δέχεται πληροφορίες μόνο από την υπηρεσία «Search», και παρέχει τις υπηρεσίες «Create Books» και «Create Videos», ενώ και οι τρεις τελευταίες αναφερθείσες αλληλεπιδρούν αμφίδρομα με την υπηρεσία «Mongo».



Εικόνα 3-2: Αρχιτεκτονική Εφαρμογής

Στη παρούσα διπλωματική αναπτύχθηκε μια εφαρμογή ιστού η οποία είχε τα εξής χαρακτηριστικά, κάποια λιγότερα εμφανή στο χρήστη και κάποια περισσότερο:

1. Μια διεπαφή για την αλληλεπίδραση με το χρήστη και εμφανίζει τα αποτελέσματα της υπηρεσίας Search
2. Έναν ρυθμιστή Nginx για την διαχείριση των αιτημάτων από τους χρήστες
3. Μια υπηρεσία για τη δημιουργία βιβλίου
4. Μια υπηρεσία για τη δημιουργία ταινίας
5. Μια υπηρεσία για τη αναζήτηση
6. Μια βάση για την αποθήκευση των παραπάνω



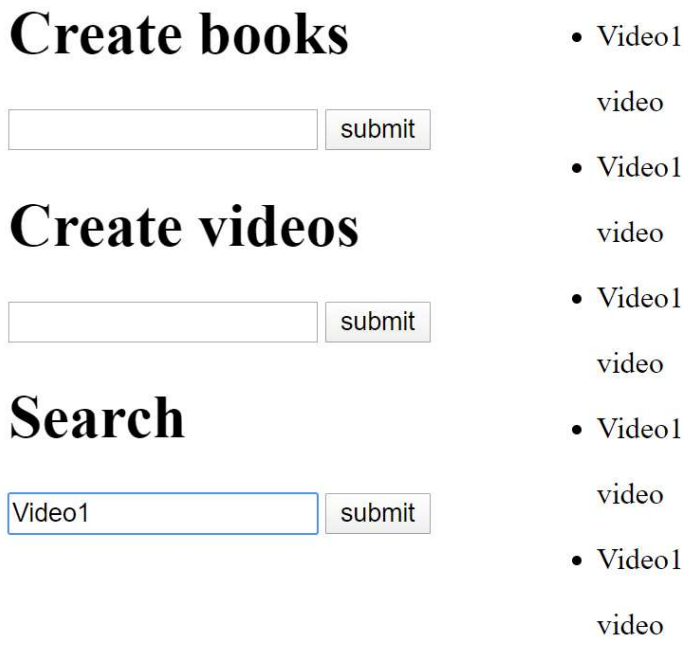
Εικόνα 3-3: Πρόσοψη της εφαρμογής

Αυτή λοιπόν η εφαρμογή αναπτύχθηκε και σε μονολιθική αλλά και σε αρχιτεκτονική μικροϋπηρεσιών με σκοπό τη καλύτερη δυνατή σύγκριση.

Παρακάτω παρουσιάζονται οι μικροϋπηρεσίες που αναπτύχθηκαν στο πλαίσιο της παραπάνω εφαρμογής με την λογική της αρχιτεκτονικής των μικροϋπηρεσιών.

3.2.1 Μικροϋπηρεσία «Web»

Η υπηρεσία Web είναι υπεύθυνη για τη παρουσίαση της ιστοσελίδας στο χρήστη, την αλληλεπίδραση μαζί του, και τη παρουσίαση των αποτελεσμάτων. Η υπηρεσία Web αποτελείται από ένα απλό αρχείο html, το οποίο έχει τη δυνατότητα να δημιουργεί τα αιτήματα προς τις υπόλοιπες μικροϋπηρεσίες, αλλά να μεταφράζει τις απαντήσεις αυτών, με σκοπό την παρουσίασή τους στο τελικό χρήστη.



Εικόνα 3-4: Παράδειγμα αποτελέσματος της εφαρμογής

3.2.2 Μικροϋπηρεσία «Create Books» και «Create Videos»

Αυτές οι δύο εφαρμογές στη βάση τους είναι πανομοιότυπες. Ο μοναδικός τους σκοπός είναι να αποθηκεύουν στη βάση δεδομένων τα ονόματα των βίντεο ή των βιβλίων και να αναζητούν, η κάθε μικροϋπηρεσία ξεχωριστά, τις αντίστοιχες αξίες.

```
// Search Book
app.get("/api/v1/books", async (req, res) => {
  const books = await Book.find({});
  res.json(books);
});

// Create Book
app.post("/api/v1/books", async (req, res) => {
  const book = new Book({ name: req.body.name });
  const savedBook = await book.save();
  res.json(savedBook);
});
```

Εικόνα 3-5: Κώδικας μικροϋπηρεσίας «Create Books»

Τα αιτήματα του χρήστη έχουν σκοπό είτε να εκμαιεύσουν το όνομα ενός βιβλίου είτε να εισάγουν ένα νέο βιβλίο στη βάση δεδομένων με τις αντίστοιχες παραπάνω εντολές. Η εφαρμογή αυτή δίνει απάντηση μέσω JSON.

3.2.3 Μικροϋπηρεσία «Search»

Η μικροϋπηρεσία Search είναι υπεύθυνη για την αναζήτηση της λέξης που αιτείται ο χρήστης να αναζητήσει στη βάση δεδομένων.

```
app.get("/api/v1/search", async (req, res) => {
  // we don't want to await we want both request to run a
  const videosPromise = Video.find({});
  const booksPromise = Book.find({});
  const promises = [videosPromise, booksPromise];
  const [videos, books] = await Promise.all(promises);

  res.json(videos.concat(books));
});
```

Εικόνα 3-6: Κώδικας μικροϋπηρεσίας «Search»

3.2.4 Μικροϋπηρεσία «Nginx»

Η μικροϋπηρεσία Nginx λειτουργεί σαν ένας αντίστροφος μεσολαβητής και είναι υπεύθυνη για τη δρομολόγηση των αιτημάτων στις κατάλληλες πόρτες-διεπαφές. Σε αυτή τη μικροϋπηρεσία υπάρχει ένας χάρτης ο οποίος κατέχει τις πληροφορίες για τη σωστή δρομολόγηση των αιτημάτων στις κατάλληλες υπηρεσίες.

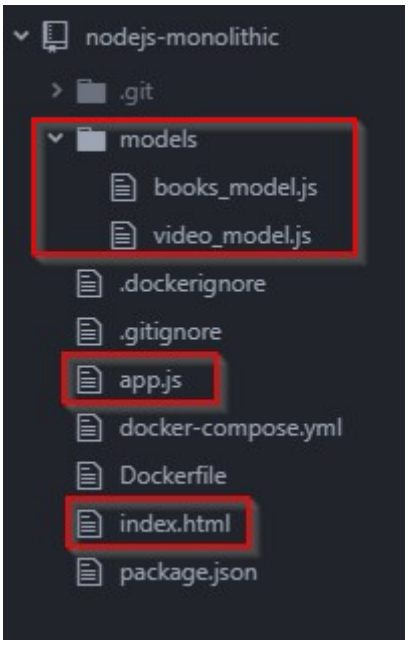
3.2.5 Μικροϋπηρεσία «Mongo»

Η μόνη λειτουργία της μικροϋπηρεσίας Mongo είναι να δημιουργεί τη βάση δεδομένων και να τη κρατάει ενεργή.

3.3 Ανάπτυξη εφαρμογής με την Μονολιθική Αρχιτεκτονική

Η ανάπτυξη της εφαρμογής διεξήχθη και με την μονολιθική αρχιτεκτονική. Το προηγούμενο κεφάλαιο, δηλαδή το «3.2.1 - Ανάπτυξη εφαρμογής με την Αρχιτεκτονική των Μικροϋπηρεσιών», αποτελεί τη βάση για την μονολιθική αρχιτεκτονική, διότι οι μεμονωμένες μικροϋπηρεσίες συνθέτουν το μοναδικό αρχείο που εκτελεί όλες τις δυνατότητες της εφαρμογής. Η εφαρμογή αποτελείται από τα παρακάτω αρχεία:

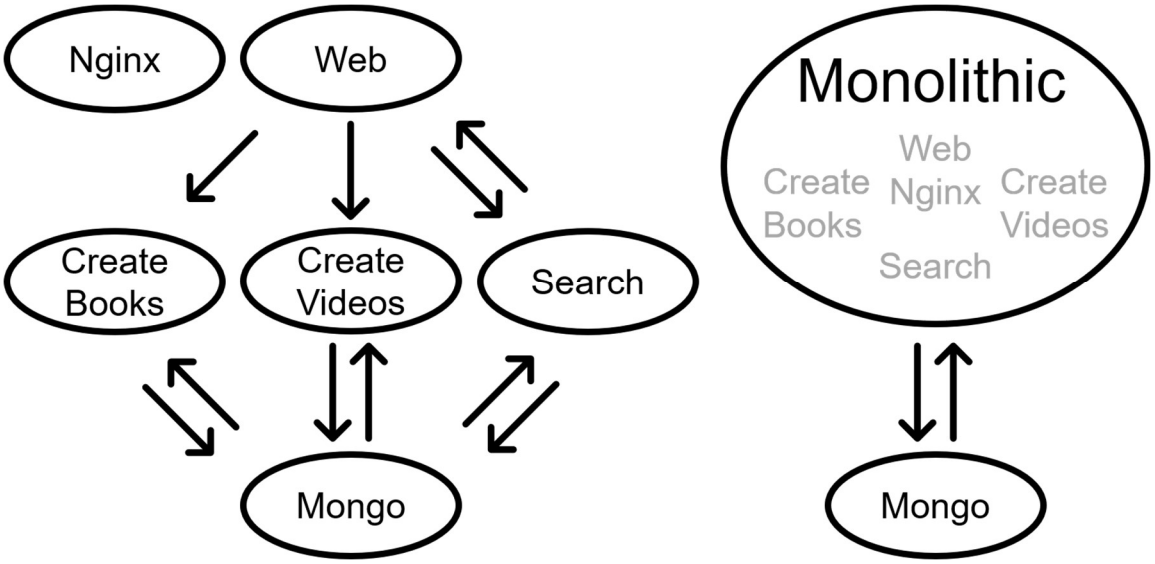
1. Τα μοντέλα της βάσης δεδομένων
2. Το «app.js» που περιέχει όλες τις λειτουργίες της εφαρμογής
3. Το «index.html» που αποτελεί την πρόσοψη της εφαρμογής



Εικόνα 3-7: Δέντρο αρχείων Μονολιθικής Εφαρμογής

3.4 Σύγκριση Μονολιθικής Αρχιτεκτονικής με Μικροϋπηρεσιών

Η μεγάλη διαφορά ανάμεσα στην ανάπτυξη της εφαρμογής με τις δύο παραπάνω αρχιτεκτονικές αποτελεί το γεγονός ότι ο κώδικας, αλλά και κατ' επέκταση οι δυνατότητες της εφαρμογής, στην αρχιτεκτονική των μικροϋπηρεσιών, χωρίζονται σε μεμονωμένες μικροϋπηρεσίες, σε αντίθεση με την μονολιθική αρχιτεκτονική. Αυτό δίνει το πλεονέκτημα στους διαχειριστές, μέσω της χρήσης του Nginx, να κατανέμουν τους πόρους, αλλά και τα αιτήματα, στις κατάλληλες υποδοχές-μικροϋπηρεσίες με τη μέγιστη αποδοτικότητα και χωρίς να γίνεται χρήση όλων των πόρων άνευ λόγου, όπως γίνεται στη μονολιθική αρχιτεκτονική. Αυτή η σύγκριση γίνεται ξεκάθαρη στην παρακάτω εικόνα, η οποία παρουσιάζει την διάταξη των υποδοχέων, για την εξεταζόμενη εφαρμογή, στις δύο διαφορετικές αρχιτεκτονικές.



Εικόνα 3-8: Σύγκριση διάταξης υποδοχέων των αρχιτεκτονικών

Στο αριστερό μέρος της εικόνας διακρίνονται οι υποδοχείς της εφαρμογής με τη λογική της αρχιτεκτονικής των μικροϋπηρεσιών, ενώ στο δεξί, η ίδια εφαρμογή, παρουσιάζεται με την μονολιθική μορφή της. Όπως γίνεται κατανοητό στην αρχιτεκτονική των μικροϋπηρεσιών οι πόροι μπορούν να διανεμηθούν αποδοτικότερα εφόσον υπάρχει σαφής διαχωρισμός των υπηρεσιών, και κατ' επέκταση των υποδοχέων. κάτι που δε συμβαίνει στη μονολιθική αρχιτεκτονική, καθώς υπάρχει μόνο ο υποδοχέας «Monolithic», ο οποίος περιέχει φυσικά όλες τις παραπάνω μικροϋπηρεσίες.

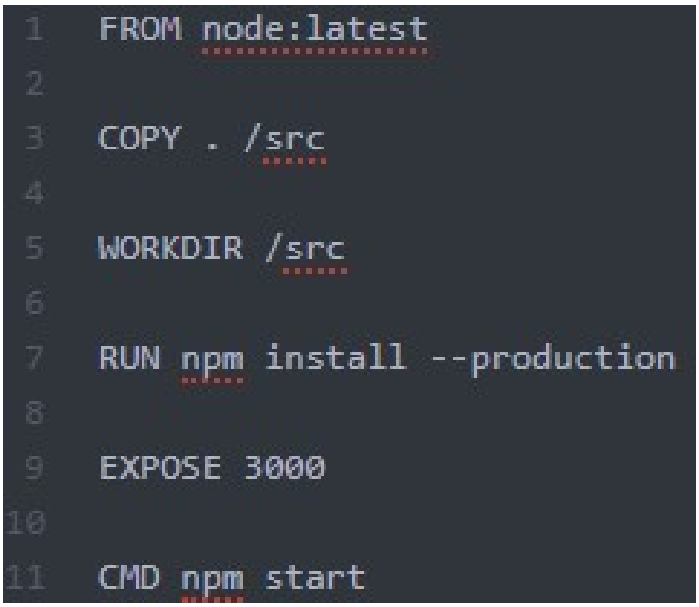
3.5 Δημιουργία Εφαρμογής μέσω Υποδοχέα

Η εφαρμογή αποτελείται από τις παραπάνω μικροϋπηρεσίες, κάθε μία από αυτές, έχει τη δικιά της μοναδική χρησιμότητα. Σε συνεργασία μεταξύ τους προσφέρουν τη τελική εφαρμογή στο χρήστη. Αυτό συμβαίνει μέσω της χρήσης του Docker και των υποδοχέων. Για όλες τις μικροϋπηρεσίες δημιουργείται ένας ή παραπάνω υποδοχείς μέσω του αρχείου «docker-compose.yml».

Η ίδια λογική ακολουθήθηκε και για τη μονολιθική έκδοση της εφαρμογής. Όλες οι μικροϋπηρεσίες, πλην της βάσης δεδομένων-δηλαδή της μικροϋπηρεσίας Mongo-, δημιουργήθηκαν και αναπτύχθηκαν με τη τεχνική του docker-compose με τη μόνη και ειδοποιό διαφορά ότι όλες οι μικροϋπηρεσίας υπάρχουν σε έναν μόνο υποδοχέα με σκοπό να τηρηθεί η μονολιθική λογική.

3.5.1 Ανάλυση Dockerfile

Το dockerfile όπως έχει αναλυθεί και στο δεύτερο κεφάλαιο είναι μια σειρά εντολών, οι οποίες είναι υπεύθυνες για τη δημιουργία ενός υποδοχέα. Παρακάτω ακολουθεί η ανάλυση ενός απλού dockerfile το οποίο ήταν υπεύθυνο για τη δημιουργία του υποδοχέα για την υπηρεσία books.



Εικόνα 3-9: Κώδικας Dockerfile

Η πρώτη εντολή είναι ότι ο υποδοχέας πρέπει κατεβάσει τη τελευταία έκδοση μιας υπάρχουσας εικόνας και σε συνδυασμό με την 3η εντολή να την αντιγράψει στην υπάρχουσα θέση στο μονοπάτι «/src»

Η εντολή «WORKDIR» προστάζει ότι οι παρακάτω εντολές θα τρέξουν στο φάκελο «/src»

Οι τρεις τελευταίες εντολές, με τη βοήθεια του εργαλείου npm, είναι υπεύθυνες για τη δημιουργία όλων των απαραίτητων εξαρτήσεων, την έκθεση της θύρας 3000 στο εξωτερικό περιβάλλον του υποδοχέας και τέλος τη λειτουργία του.

3.5.2 Ανάλυση αρχείου docker-compose.yml

Το αρχείου docker-compose.yml αποτελείται από μια σειρά εντολών, οι οποίες είναι υπεύθυνες για τη δημιουργία ενός ή περισσότερων υπηρεσιών, οι οποίες αποτελούνται από υποδοχείς, και τις εξαρτήσεις μεταξύ τους. Παρακάτω ακολουθεί η ανάλυση ενός τμήματος του docker-compose αρχείου το οποίο ήταν υπεύθυνο για τη δημιουργία του υποδοχέα για την υπηρεσία web.

```yaml
version: '3.3'
services:
  web:
    ports:
      - "3000:3000"
    image: ioannispapakonstantinou/micro_web
    deploy:
      resources:
        limits:
          cpus: '0.10'
```

Εικόνα 3-10: Κώδικας docker-compose.yml

Παρόμοια λογική με το dockerfile αρχείο υπάρχει και στο docker-compose.yml. Αρχικά το παραπάνω αρχείο προστάζει να χρησιμοποιηθεί η έκδοση «3.3» του docker compose. Έπειτα, να συνδέσει την πόρτα 3000 του οικοδεσπότη, στην πόρτα 3000 του υποδοχέα και να χρησιμοποιήσει την εικόνα «ioannispapakonstantinou/micro_web» η οποία βρίσκεται στο docker hub. Τέλος, να βάλει όριο 10% στην χρήση του κεντρικού επεξεργαστή της ανάπτυξης.

3.5.3 Ανάλυση χρήσης σμήνους docker

Για να χρησιμοποιηθεί το σμήνος docker πρέπει αρχικά να τρέξει η εντολή «docker swarm init». Έπειτα με την βοήθεια του αρχείου docker-compose.yml δημιουργούνται και όλες οι απαραίτητες υπηρεσίες, με σκοπό τη διαχείριση των υποδοχέων. Τέλος η κλιμάκωσή είτε προς τα πάνω είτε προς τα κάτω αποτελεί εύκολη διαδικασία, καθώς με τη χρήση μόνο μιας εντολής μπορεί να υπάρξει το επιθυμητό αποτέλεσμα, με σκοπό την αποτελεσματικότερη χρήση των διαθέσιμων πόρων.

4 Πειραματική και Θεωρητική Συγκριτική Ανάλυση

Ο σκοπός αυτού του κεφαλαίου είναι να υπάρξει όσο το δυνατότερο καλύτερη σύγκριση ανάμεσα στις δύο αρχιτεκτονικές, και τους τρόπους κλιμάκωσης των υποδοχέων, με στόχο την εξαγωγή συμπερασμάτων στο 5ο κεφάλαιο. Αυτός ο στόχος έχει τόσο τεχνικές προεκτάσεις όσο και οικονομικές. Από τεχνικής σκοπιάς, οι συγκρίσεις των δύο εφαρμογών-αρχιτεκτονικών αποσκοπούν, στην εύρεση της ποσοτικής διαφοράς στη χρήση των υπολογιστικών πόρων, αλλά και της ποιοτικής, όσον αφορά την ικανοποίηση των πελατών βάσει των χρόνων ανταπόκρισης των αιτημάτων. Από την άλλη πλευρά, δηλαδή την οικονομική σκοπιά, οι συγκρίσεις πραγματοποιούνται με σκοπό τον υπολογισμό των χρηματικών πόρων που μπορούν να εξοικονομηθούν, λόγω της χρήσης λιγότερων υπολογιστικών πόρων στο υπολογιστικό νέφος και κατ' επέκταση την μείωση του κόστους της εφαρμογής, χωρίς αισθητή επιδείνωση της ικανοποίησης των χρηστών.

Τα παραπάνω, καθίστανται εφικτά μέσω της δυναμικής κατανομής τους στην αρχιτεκτονική των μικροϋπηρεσιών, αλλά και μέσω του νέου τρόπο κλιμάκωσης των υποδοχέων, ο οποίος βασίζεται στο χρόνο ανταπόκρισης απαντήσεων από τα αιτήματα των χρηστών προς τις μικροϋπηρεσίες και όχι στη χρήση του επεξεργαστή από τους υποδοχείς. Επιπλέον, πραγματοποιήθηκε έρευνα εύρεσης τιμολογιακών πλάνων από διάφορους παρόχους του υπολογιστικού νέφους με σκοπό υπολογισμό την οικονομικής ελάφρυνσης χρησιμοποιώντας την αρχιτεκτονική των μικροϋπηρεσιών αλλά και την κλιμάκωση βάσει του χρόνου ανταπόκρισης των αιτημάτων. Στόχο αποτελεί η μετάφραση της εξοικονόμησης των υπολογιστικών πόρων, σε χρηματικά ποσά στο υπολογιστικό νέφος.

Σημειώνεται ότι οι παραπάνω συγκρίσεις, αλλά και η οικονομική ανάλυση, πραγματοποιήθηκαν σε θεωρητικό επίπεδο, εξαιτίας τεχνικών περιορισμών. Το πρακτικό επίπεδο, έρχεται να το καλύψει η μελέτη περίπτωση, έχοντας σαν αποτέλεσμα, την επίτευξη των αρχικών στόχων, δηλαδή τη μείωση της χρήσης των αναγκαίων υπολογιστικών πόρων βάσει της αρχιτεκτονικής των μικροϋπηρεσιών, αλλά και την περαιτέρω μείωση των παραπάνω πόρων βάσει του νέου τρόπου κλιμάκωσης. Ωστόσο, πρέπει να σημειωθεί, ότι αυτή η αποδοτικότερη χρήση των πόρων και η σημαντική μείωση του κόστους, επηρέασε το μέσο χρόνο ανταπόκρισης κατά 200 χιλιοστά του δευτερολέπτου, καθιστώντας τη αύξηση ανεπαίσθητη, εφόσον και το 95% των παρατηρήσεων παραμένει κάτω από τα 3 δευτερόλεπτα.

4.1 Πειραματική Ανάλυση

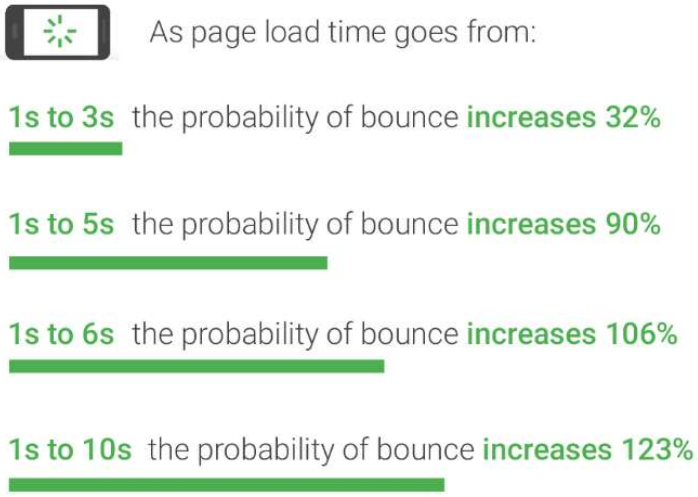
Για την υλοποίηση δοκιμών χρησιμοποιήθηκε το πρόγραμμα «Apache JMeter». Το εργαλείο αυτό αποτελεί ένα καθαρό λογισμικό ανοιχτού κώδικα Java το οποίο αναπτύχθηκε για πρώτη φορά από τον Stefano Mazzocchi του ιδρύματος Apache Software, το οποίο σχεδιάστηκε για να φορτώσει τη δοκιμαστική λειτουργική συμπεριφορά και να μετρήσει την απόδοση. Μπορεί να χρησιμοποιηθεί για να αναλύσει και να μετρήσει την απόδοση της εφαρμογής web ή μια ποικιλία άλλων υπηρεσιών. Η δοκιμή της απόδοσης μιας εφαρμογής σημαίνει δοκιμή μιας εφαρμογής ιστού με μεγάλο φορτίο σε συνδυασμό με την πολλαπλή και ταυτόχρονη κυκλοφορία χρηστών. Το JMeter αρχικά χρησιμοποιήθηκε για δοκιμές εφαρμογών Web ή εφαρμογών FTP. Στις μέρες μας, χρησιμοποιείται για μια μεγάλη ποικιλία δοκιμών [20].

Πριν γίνει αναφορά στα σενάρια σύγκρισης, με σκοπό να υπάρξουν κάποια όρια ως προς τις συγκρίσεις που θα γίνουν ανάμεσα στις αρχιτεκτονικές αλλά και στις μεθόδους κλιμάκωσης, ορίστηκαν κάποιες προϋποθέσεις. Επιπρόσθετα, τα παρακάτω όρια επιλέχθηκαν ώστε να αναδείξουν τις αρχιτεκτονικές.

Στο πρώτο σενάριο σύγκρισης ανάμεσα στην μονολιθική και στην αρχιτεκτονική των μικροϋπηρεσιών, τέθηκε όριο 10% στην επεξεργαστική ισχύ των υποδοχέων που ήταν υπεύθυνοι για την μικροϋπηρεσία «search», για την οποία παρουσιάζονται τα πειράματα. Το όριο αυτό τέθηκε στην αρχιτεκτονική των μικροϋπηρεσιών για να αναδείξει την χρησιμότητα αυτής της προσέγγισης. Από την άλλη πλευρά στη μονολιθική αρχιτεκτονική το όριο που τέθηκε ήταν το 80% χρήσης της επεξεργαστικής δύναμης από τον εκάστοτε υποδοχέα. Επιχειρήθηκε να τεθεί ίσο όριο με αυτό στην αρχιτεκτονική των μικροϋπηρεσιών, αλλά λόγω τεχνικών περιορισμών από το σμήνος docker, δεν κατέστη εφικτό.

Επιπρόσθετα, τέθηκε ανώτατο όριο χρόνου ανταπόκρισης των αιτημάτων, διότι οι χρήστες προσδοκούν συγκεκριμένο χρόνο απάντησης των αιτημάτων τους, με βάση πάντα το είδος του αιτήματος και την ανεκτικότητά τους ως προς την αναμονή της απάντησης. Ωστόσο, η διατριβή επικεντρώνεται μόνο στο χρόνο ανταπόκρισης και όχι στο θέμα του αιτήματος, διότι το θέμα παραμένει ίδιο σε όλα τα παρακάτω σενάρια. Το παρακάτω διάγραμμα, το οποίο βασίζεται σε έρευνα που διεξήγαγε η Google [24], υποδεικνύει ότι η πιθανότητα με την οποία θα διακόψουν το αίτημά τους οι χρήστες ως προς μια υπηρεσία λόγου αυξημένου χρόνου ανταπόκρισης, αυξάνεται δραματικά από τα 3 έως τα 5 δευτερόλεπτα, έχοντας την πιο σημαντική αύξηση κατά 58%(=90%-32%). Τέλος οι περισσότερες συμφωνίες σε επίπεδο υπηρεσιών (SLA) απαιτούν διαθεσιμότητα της εφαρμογής ίσης με το 95% του συνολικού χρόνο που λειτουργεί η υπηρεσία. Σαν αποτέλεσμα

των παραπάνω, τέθηκε μέγιστο όριο στο χρόνο ανταπόκρισης των αιτημάτων τα 3 δευτερόλεπτα για τουλάχιστον το 95% των παρατηρήσεων.



Διάγραμμα 4-1: Χρόνοι ανταπόκρισης Vs Bounce Rates

Επιπλέον, πρέπει να γνωστοποιηθούν οι παράμετροι των δοκιμών. Για τα πειράματα που αναλύονται παρακάτω, οι παράμετροι, μετά από πολλές δοκιμές, κατέληξαν να είναι 17,500 αιτήματα από 175 χρήστες, οι οποίοι με τη σειρά τους πραγματοποιούσαν τα αιτήματα σχεδόν συγχρονισμένα. Αξίζει να σημειωθεί ότι σε όλα τα σενάρια δε χάθηκαν αιτήματα, το οποίο γίνεται φανερό από το ποσοστό λάθους 0% στους παρακάτω πίνακες και ότι όλα τα μεγέθη χρόνου μετριούνται σε χιλιοστά του δευτερολέπτου.

Τρεις ήταν οι μετρικές που δόθηκαν μεγαλύτερη προσοχή. Η χρήση των πυρήνων από τους υποδοχείς-μικροϋπηρεσίες, ο μέσος χρόνος ανταπόκρισης των απαντήσεων και ο δείκτης «95% Line». Οι δύο πρώτοι δείκτες είναι αυτοεξηγούμενοι, ωστόσο ο τελευταίος παρουσιάζει την τιμή κάτω από την οποία βρίσκεται το 95% των παρατηρήσεων. Αυτός ο δείκτης είναι ο σημαντικότερος δείκτης, καθώς μέσω αυτού καθορίζεται ο στόχος σε επίπεδο υπηρεσίας και κατ' επέκταση, η τήρηση ή παραβίαση της συμφωνίας σε επίπεδο υπηρεσιών.

Τέλος, βάσει των παραπάνω, επιδίωξη της διατριβής, μέσω των δοκιμών, αποτελεί να βρεθεί η χρυσή τομή ανάμεσα στους ελάχιστους υπολογιστικούς πόρους που πρέπει να χρησιμοποιηθούν έτσι ώστε τουλάχιστον το 95% των χρηστών να μείνουν ικανοποιημένοι, δηλαδή να τηρηθεί η συμφωνία σε επίπεδο υπηρεσιών με το μικρότερο κόστος.

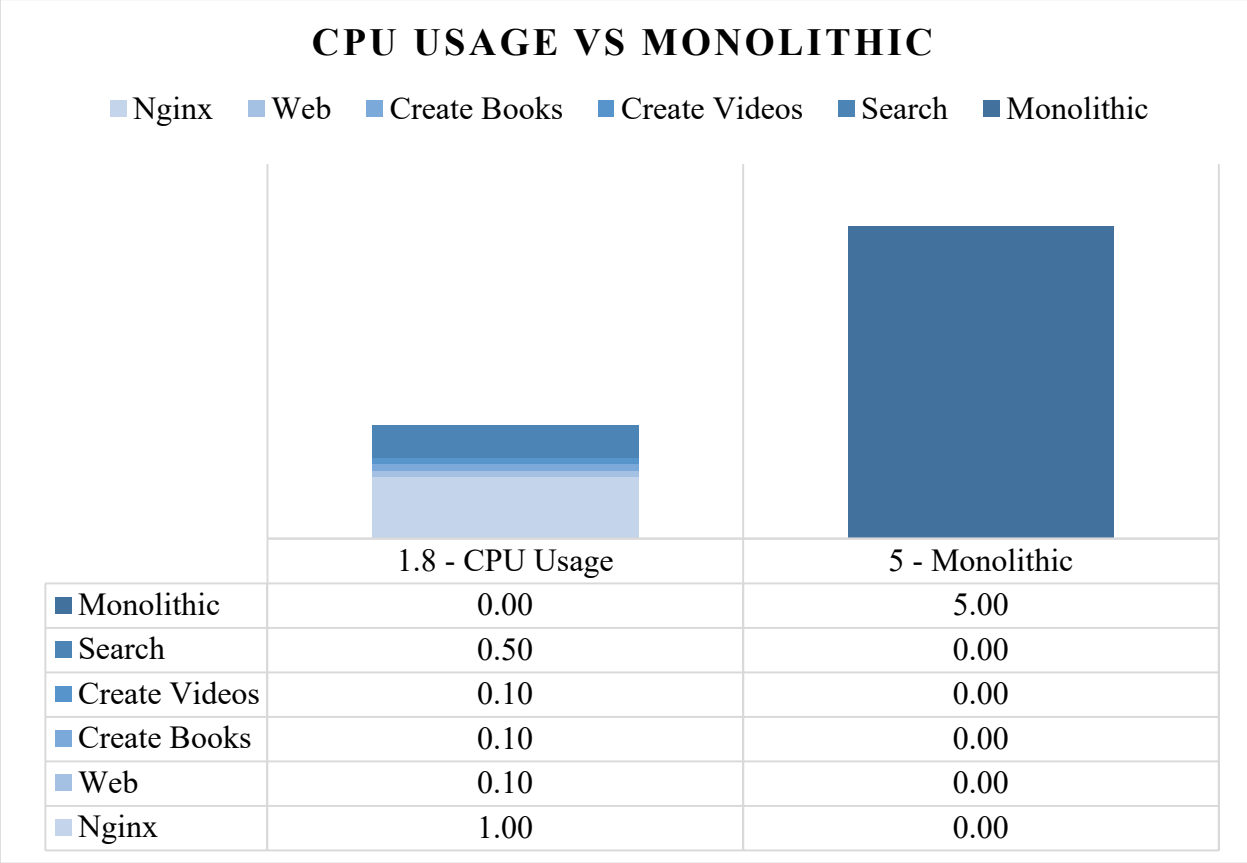
4.1.1 Μονολιθική σε σύγκριση με Μικροϋπηρεσίες

Αρχικά, η έρευνα εξετάζει τους πόρους που μπορεί να εξοικονομήσει η επιχείρηση, κατανέμοντας τους δυναμικά και κατάλληλα στις μεμονωμένες υπηρεσίες και όχι ενιαία σε μια

μονολιθική προσέγγιση. Για το κάθε σενάριο, διατέθηκαν 8 πυρήνες επεξεργαστικής ισχύς. Τα αποτελέσματα που προέκυψαν από την έρευνα είναι εκπληκτικά, διότι η αρχιτεκτονική των μικροϋπηρεσιών κατάφερε να εξυπηρετήσει τα αιτήματα με χρήση 1.8 από το σύνολο των 8 πυρήνων, σε αντίθεση με τη μονολιθική προσέγγιση, στην οποία κρίθηκε απαραίτητη η χρήση 5 πυρήνων.

Στο παρακάτω διάγραμμα παρουσιάζεται η μέγιστη δυνατή χρήση των πυρήνων, ανά υπηρεσία, σε κάθε ένα από τα δύο σενάρια, ήτοι 1.8 στο «CPU Usage» σενάριο και 5 στο Monolithic σενάριο. Όπως έχει αναφερθεί στην αρχή του κεφαλαίου, τέθηκαν όρια στα δύο παρακάτω σενάρια. Η ένδειξη 0.5 στην υπηρεσία Search μεταφράζεται ως χρήση των 5 υποδοχέων Search, οι οποίοι διατηρούσαν τη χρήση επεξεργαστικής δύναμης μικρότερη από 0.1, με αποτέλεσμα να πρέπει να δεσμευτεί το 0.50 από το σύνολο των πυρήνων. Αντίστοιχα, στο σενάριο Monolithic, χρησιμοποιήθηκαν 5 υποδοχείς Monolithic, οι οποίοι διατηρούσαν τη χρήση επεξεργαστικής δύναμης κάτω από το 1, με αποτέλεσμα να πρέπει να δεσμευτούν 5.00 από το σύνολο των πυρήνων.

Όσον αφορά την παραπάνω κλιμάκωση των υποδοχέων, πρέπει να αναφερθεί ότι, όταν η χρήση του κεντρικού επεξεργαστή κάποιου υποδοχέα φτάνει για περισσότερο από το μισό χρόνο της δοκιμής παραπάνω από το 80% του ορίου που του είχε τεθεί, πολλαπλασιάζεται ο υποδοχέας. Μετά από επανειλημμένες δοκιμές, η μικροϋπηρεσία Search για να μπορέσει να ανταποκριθεί στα αιτήματα, χρειάστηκε 5 υποδοχείς στο σενάριο SLA, των οποίων ο δείκτης CPU %, από το εργαλείο docker stats, παρέμενε σταθερά κάτω από 0.08. Αντίστοιχα, με την ίδια λογική κατανεμήθηκαν πόροι και στις υπόλοιπες υπηρεσίας-σενάρια. Σημειώνεται ότι οι εν λόγω υπηρεσίες αναλύθηκαν στο κεφάλαιο 3.2.



	1.8 - CPU Usage	5 - Monolithic
Monolithic	0.00	5.00
Search	0.50	0.00
Create Videos	0.10	0.00
Create Books	0.10	0.00
Web	0.10	0.00
Nginx	1.00	0.00

Διάγραμμα 4-2: CPU Usage Vs Μονολιθική

Τα παραπάνω συμπεράσματα προέκυψαν χρησιμοποιώντας τους υποδοχείς. Στην αρχιτεκτονική των μικροϋπηρεσιών υπάρχει η δυνατότητα να κατανεμηθούν οι πόροι με σκοπό την καλύτερη εξυπηρέτηση, στη συγκεκριμένη στιγμή, των αιτημάτων από τους χρήστες προς συγκεκριμένη μικροϋπηρεσία. Στην μονολιθική αρχιτεκτονική, η παραπάνω δυνατότητα, δεν είναι διαθέσιμη, με αποτέλεσμα να πρέπει να κατανέμονται ισόποσα οι πόροι σε όλες τις υπηρεσίες, το οποίο οδηγεί σε μη ορθή-αποτελεσματική χρήση των υπολογιστικών πόρων.

Για να γίνει σωστή εξυπηρέτηση των αιτημάτων πρέπει να υπάρχουν για κάθε αίτημα, ανά πάσα στιγμή, αρκετοί διαθέσιμοι υπολογιστικοί πόροι. Για να επιτευχθεί το παραπάνω και επειδή ο κλασικός τρόπος κλιμάκωσης των υποδοχέων γίνεται με βάση την χρήση του κεντρικού επεξεργαστή, τα πειράματα που διεξήχθησαν με τη μονολιθική προσέγγιση διατηρούσαν τη χρήση του κεντρικού επεξεργαστή του κάθε υποδοχέα κάτω από το 80%.

Σενάρια	Χρήση	Average	Median	95% Line	Error %
CPU Usage	1.8	648	606	1,233	0
Monolithic	5	76	70	138	0

Πίνακας 4-1: Μικροϋπηρεσίες Vs Μονολιθική

Από τον παραπάνω πίνακα γίνεται φανερό ότι στη μονολιθική προσέγγιση ο μέσος όρος των παρατηρήσεων των μετρήσεων του χρόνου εξυπηρέτησης των αιτημάτων είναι εμφανώς πολύ

καλύτερος. Στη μονολιθική οι χρήστες εξυπηρετούνται κατά μέσο όρο σε 76ms, ενώ στη αρχιτεκτονική των μικροϋπηρεσιών ο μέσος χρόνος εξυπηρέτησης είναι σχεδόν εξαπλάσιος, δηλαδή 648ms.

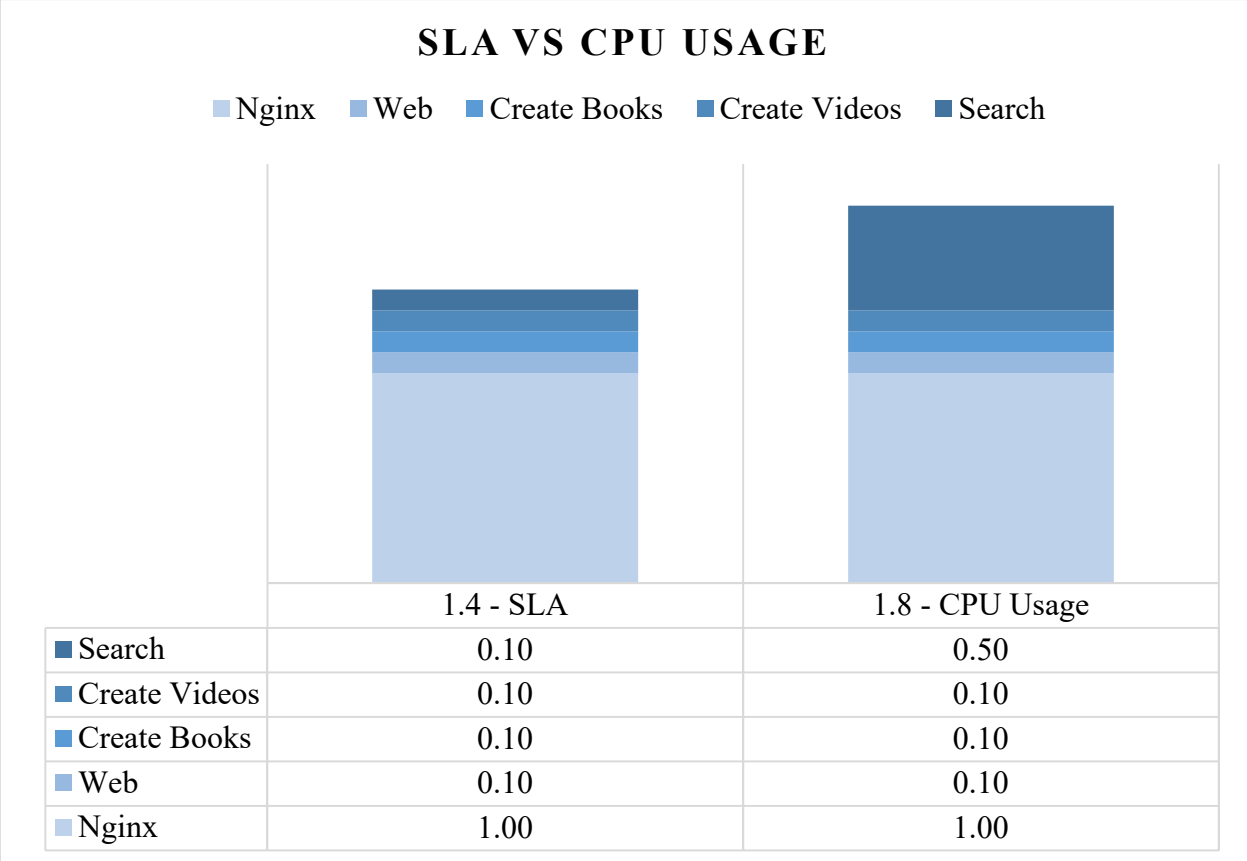
Το παραπάνω δεν αποτελεί πρόβλημα, διότι ο επιπλέον χρόνος ανταπόκρισης στην αρχιτεκτονική των μικροϋπηρεσιών σε σύγκριση με την μονολιθική, δεν κρίνεται σημαντικός. Επιπλέον, δεδομένου ότι το 95% των παρατηρήσεων παραμένουν κάτω από τα 3 δευτερόλεπτα αποτελεί πολύ σημαντικό επιχείρημα ως προς τη χρήση της αρχιτεκτονικής των μικροϋπηρεσιών. Επιπλέον, η ικανοποίηση των πελατών δεν αλλάζει αισθητά, διότι είναι κατά περίπου μισό δευτερόλεπτο πιο αργός ο μέσος όρος ανταπόκρισης. Ωστόσο, το κόστος της εφαρμογής μειώνεται σε μεγάλο βαθμό, καθώς, στο σενάριο με χρήση των μικροϋπηρεσιών, η εφαρμογή χρησιμοποιεί κατά $\frac{5-1.8}{5} = 64\%$ λιγότερους πόρους.

4.1.2 Χρήση κεντρικού επεξεργαστή σε σύγκριση με Χρόνους Ανταπόκρισης

Μετά από τα αισιόδοξα αποτελέσματα από τη παραπάνω σύγκριση ανάμεσα στην μονολιθική αρχιτεκτονική και στην προσέγγιση των μικροϋπηρεσιών, η έρευνα επιθυμεί να εμβαθύνει και να εξετάσει εάν μπορεί να καταφέρει ακόμα καλύτερη αποτελεσματικότητα των υπάρχων πόρων.

Για να επιτευχθεί το παραπάνω χρειάζεται να προσαρμοστεί ο τρόπος με τον οποίο πολλαπλασιάζονται οι υποδοχείς. Ο παραδοσιακός τρόπος κλιμάκωσης των υποδοχέων, όπως συνέβη στο προηγούμενο κεφάλαιο, γίνεται σύμφωνα με το ποσοστό χρήσης του κεντρικού επεξεργαστή. Η προσέγγιση που ακολουθήθηκε όσον αφορά την κλιμάκωση των υποδοχέων είναι ότι πολλαπλασιάζονταν σύμφωνα με το χρόνο ανταπόκρισης των αιτημάτων. Αυτό είχε σαν αποτέλεσμα η εφαρμογή να δουλεύει ανεπηρέαστα, δηλαδή τα αιτήματα να εξυπηρετούνται στον επιτρεπτό χρόνο, αλλά και να χρησιμοποιούνται στο εκατό τοις εκατό οι διαθέσιμοι πόροι.

Όπως αναφέρθηκε και στην αρχή του κεφαλαίου, στις δοκιμές που διεξήχθησαν, τουλάχιστον το 95% των απαντήσεων στα αιτήματα από τους χρήστες, έκαναν χρόνο ανταπόκρισης λιγότερο των 3 δευτερολέπτων. Επιπρόσθετα, χρησιμοποιώντας το νέο τρόπο κλιμάκωσης κατέστη εφικτό η εξεταζόμενη μικροϋπηρεσία Search να χρησιμοποιεί 0.10 από τους 8 πυρήνες, σε αντίθεση με το «CPU Usage» σενάριο, που χρησιμοποιεί το 0.50 των 8 πυρήνων, με αποτέλεσμα τη συνολική μείωση κατά 0.40 πυρήνες. Με βάση τα παραπάνω προκύπτουν οι εξής συνδυασμοί κατανομής των διαθέσιμων πόρων:



	1.4 - SLA	1.8 - CPU Usage
Search	0.10	0.50
Create Videos	0.10	0.10
Create Books	0.10	0.10
Web	0.10	0.10
Nginx	1.00	1.00

Διάγραμμα 4-3: SLA Vs CPU Usage

Όλα τα παραπάνω απασχολούν ιδιαίτερα τις επιχειρήσεις, διότι η έρευνα αποδεικνύει ότι μπορεί να γίνει αποτελεσματικότερη χρήση των πόρων, το οποίο έχει σαν αποτέλεσμα μικρότερη χρήση πόρων, χαμηλότερο κόστος και τέλος μεγαλύτερο κέρδος, χωρίς ουσιαστικά να μειώνεται η μέση ποιότητα και ικανοποίηση των χρηστών. Ωστόσο, η οικονομική σκοπιά των παραπάνω συμπερασμάτων θα αναπτυχθεί περισσότερο στο επόμενο κεφάλαιο.

Σενάρια	Χρήση	Average	Median	95% Line	Error %
CPU Usage	1.8	648	606	1,233	0
SLA	1.4	1,529	1,575	1,814	0

Πίνακας 4-1: SLA Vs CPU

Σε αυτό το σενάριο γίνεται προσπάθεια εύρεσης καλύτερου τρόπου κλιμάκωσης των υποδοχέων στην αρχιτεκτονική των μικροϋπηρεσιών. Παρότι ο υποδοχέας Search λειτουργούσε συνέχεια στο μέγιστο των δυνατοτήτων του, δηλαδή στο 0.1, αυτό ευτυχώς δεν είχε σαν αποτέλεσμα να χαθούν αιτήματα, καθώς ο δείκτης «Error %» έχει τιμή 0. Επιπλέον, από τον παραπάνω πίνακα γίνεται αντιληπτό το γεγονός ότι ο μέσος όρος στο SLA σενάριο είναι κατά περίπου 1 δευτερόλεπτο μεγαλύτερος από τον κλασικό τρόπο κλιμάκωσης. Το παραπάνω δε

στέκεται εμπόδιο διότι το 95% των χρόνων ανταπόκρισης των παρατηρήσεων, παραμένει κάτω από 3 δευτερόλεπτα.

Τέλος, αυτή η σύγκριση υποδεικνύει ότι η μέθοδος κλιμάκωσης στην αρχιτεκτονική των μικροϋπηρεσιών επιδέχεται βελτίωσης. Με το νέο τρόπο κλιμάκωσης, δηλαδή βάσει του χρόνου ανταπόκρισης των αιτημάτων, η εξοικονόμηση που συμβαίνει στους πόρους είναι $\frac{1.8-1.4}{1.8} = 20\%$ στο σύνολο της εφαρμογής, και $\frac{0.5-0.1}{0.5} = 80\%$ στην μικροϋπηρεσία Search. Ο τελευταίος δείκτης κατέχει την μεγαλύτερη βαρύτητα για το κατά πόσο ο νέος τρόπος κλιμάκωσης είναι αποτελεσματικός, ωστόσο ο πρώτος αποτελεί τον κυρίαρχο δείκτη στην απόφαση μιας επιχείρησης εάν αυτή η εξοικονόμηση των πόρων στο σύνολο της εφαρμογής αποτελεί σημαντική μείωση ώστε να υιοθετηθεί ο νέος τρόπος κλιμάκωσης.

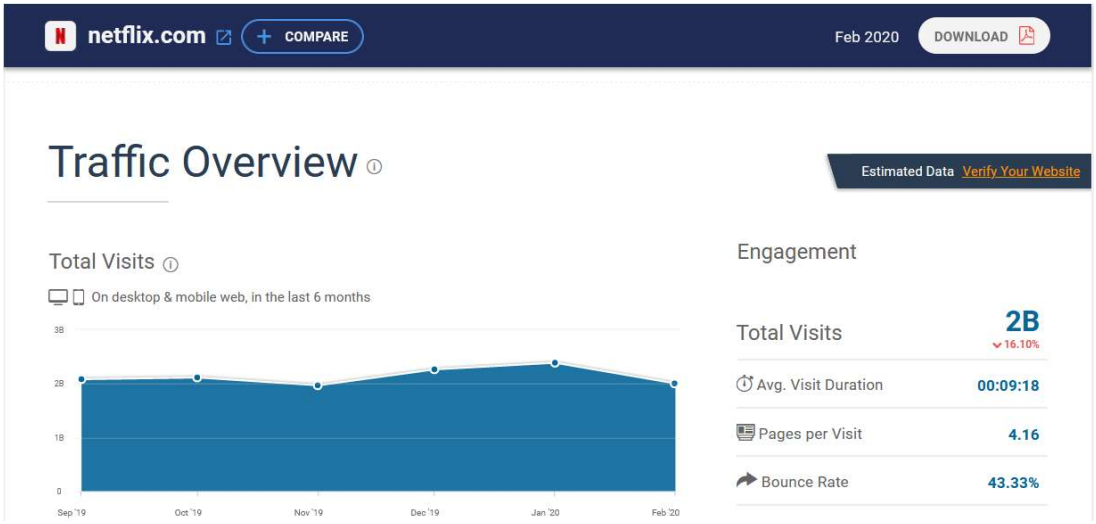
4.2 Οικονομική Ανάλυση

Στόχος της διατριβής αποτελεί η μετάφραση της εξοικονόμησης που συνέβει στους υπολογιστικούς πόρους, σε χρηματικά ποσά, στο υπολογιστικό νέφος. Για αυτό το λόγο, πραγματοποιήθηκε έρευνα με σκοπό την εύρεση τιμολογιακής πολιτικής στο υπολογιστικό νέφος στη ιστοσελίδα «www.cloudorado.com» [26], η οποία παρέχει άμεσες συγκρίσεις μεταξύ των παρόχων. Η επιλογή των παρόχων του υπολογιστικού νέφους έγινε σύμφωνα με το πόση υπολογιστική δύναμη μπορούν να διαθέσουν στο κάθε τιμολογιακό πλάνο. Αναζητήθηκαν πάροχοι με τα παρακάτω χαρακτηριστικά:

- RAM: 32 GB
- Αποθηκευτικός χώρος: 30 GB SSD
- Πυρήνες επεξεργαστή: 8
- Λειτουργικό σύστημα: Linux

Από τους διαθέσιμους παρόχους επιλέχθηκαν η Amazon, λόγω ποικιλίας επιλογών στα τιμολογιακά πλάνα και βάση της ιστοσελίδας «https://calculator.aws/#/» [27] το μηνιαίο ποσό ανέρχεται στα $747.

Η οικονομική ανάλυση της διατριβής βασίστηκε στην ιστοσελίδα «www.similarweb.com» [23], η οποία παρουσιάζει στατιστικά στοιχεία για την επισκεψιμότητα άλλων ιστοσελίδων. Επιλέχθηκε η ιστοσελίδα της εταιρίας Netflix για το μήνα Φεβρουάριο 2020. Οι δείκτες στους οποίους θα επικεντρωθεί η έρευνα είναι το σύνολο των επισκέψεων στην προαναφερθείσα ιστοσελίδα και ο δείκτης «Pages per Visit». Ο τελευταίος δείχνει πόσους συνδέσμους επισκέπτεται ένας χρήστης κατά μέσο όρο σε κάθε επίσκεψη.



Εικόνα 4-1: Στατιστικά Netflix

Σύμφωνα με τα παραπάνω στοιχεία, κατά το μήνα Φεβρουάριο, πραγματοποιήθηκαν 2 δισεκατομμύρια επισκέψεις, από τις οποίες προέκυψαν 4 περίπου επιπλέον επισκέψεις συνδέσμων για την εκάστοτε. Εάν λοιπόν, παρθεί σαν υπόθεση ότι η κάθε επίσκεψη αντιστοιχεί σε ένα αίτημα, εξάγεται σαν αποτέλεσμα ότι το μήνα Φεβρουάριο, η ιστοσελίδα Netflix, δέχθηκε 8 δισεκατομμύρια αιτήματα, οπότε για 29 ημέρες, δεχόταν $\frac{8,000,000,000}{29*24*4} = 2,873,563$ αιτήματα ανά 15 λεπτά.

Σύμφωνα με τα παραπάνω, κάνοντας μια απλή αναλογία των αιτημάτων που μπορούν να εξυπηρετήσουν οι συνδεσμολογίες από τα σενάρια του του προηγούμενου κεφαλαίου, σε σχέση με τα υπολογισμένα αιτήματα που γίνονται στην υπηρεσία Netflix, προκύπτει σαν αποτέλεσμα ότι χρειάζονται περίπου 165 φορές η ίδια συνδεσμολογία των παραπάνω σεναρίων με σκοπό να εξυπηρετηθούν τα υποθετικά αιτήματα.

Λαμβάνοντας τα παραπάνω υπόψιν και αγνοώντας τεχνικά προβλήματα, όπως είναι το κόστος και οι τεχνολογικοί περιορισμοί ως προς τη συνδεσμολογία, εξάγονται τα παρακάτω αποτελέσματα:

Σενάρια	Χρήση CPU	Ανάγκες CPU	Νέφος	Συνολικό Κόστος
SLA	1.4	231	29	$21,663.00
CPU Usage	1.8	297	38	$28,386.00
Monolithic	5	825	104	$77,688.00

Πίνακας 4-2: Κόστος Υπολογιστικού νέφους

Η στήλη «Ανάγκες CPU» υπολογίζονται πολλαπλασιάζοντας το 165 με τη στήλη «Χρήση CPU» . Έπειτα διαιρείται η προαναφερθείσα στήλη με τους 8 πυρήνες, όπως στις αρχικές δοκιμές, του τιμολογιακού πλάνου του υπολογιστικού νέφους για να προκύψει το πόσες φορές χρειάζεται το κάθε σενάριο το πλάνο του υπολογιστικού νέφους, στρογγυλοποιώντας προς τα πάνω όπου

χρειάζεται. Τέλος πολλαπλασιάζοντας την στήλη «Νέφος»» με τη τιμή $747 για να προκύψει το συνολικό υπολογιστικό κόστος για το μήνα Φεβρουάριο.

Από τον παραπάνω πίνακα γίνεται αντιληπτό το γεγονός ότι το κόστος του μονολιθικού σεναρίου είναι κατά 2.74 φορές μεγαλύτερο από το CPU σενάριο και 3.59 από το SLA σενάριο.

Διάγραμμα 4-4: Κόστος Υπολογιστικού Νέφους

4.3 Μελέτη Περίπτωσης

Σε αυτό το κεφάλαιο χρησιμοποιήθηκαν όλες οι παραπάνω πληροφορίες και η γνώση που αποκτήθηκε από την έρευνα των προηγούμενων κεφαλαίων με σκοπό να πραγματοποιηθεί μια πραγματική μελέτη περίπτωσης. Πραγματοποιήθηκε προσπάθεια δοκιμών με τα στοιχεία που παρουσιάστηκαν στο κεφάλαιο «4.2 - Οικονομική Ανάλυση», αλλά λόγω περιορισμού της υπολογιστικής δύναμης, κάτι τέτοιο δεν κατέστη εφικτό.

4.3.1 Εφικτό Σενάριο

Το «google.gr» το μήνα Φεβρουάριο είχε 44,9 εκατομμύρια αιτήματα χρηστών και άλλους 10 συνδέσμους, από την αρχική τους αναζήτηση, σύμφωνα με το ιστότοπο «similarweb.com» [22]. Εάν θεωρηθεί ότι κάθε νέα ιστοσελίδα αντιστοιχεί σε ένα αίτημα μπορούμε να εξάγουμε τα

παρακάτω αποτελέσματα, όπως συνέβει στα προηγούμενα κεφάλαια. Για το μήνα Φεβρουάριο πραγματοποιήθηκαν: $\frac{44,900,000*10}{29*24*4} = 161,278$ αιτήματα ανά 15 λεπτά.

Ο σκοπός, λοιπόν, αυτού του κεφαλαίου είναι να πραγματοποιηθεί, όσο είναι δυνατόν, μια σύγκριση με πραγματικά δεδομένα ανάμεσα στην μονολιθική αρχιτεκτονική και την προοπτική των μικροϋπηρεσιών, αλλά και να φτάσει στα όριά της η τελευταία. Για αυτό το λόγο, για την αρχιτεκτονική των μικροϋπηρεσιών χρησιμοποιήθηκε ο τρόπος κλιμάκωσης που αναλύθηκε στο κεφάλαιο 4.1.2, δηλαδή κλιμάκωση σύμφωνα με το χρόνο ανταπόκρισης, αλλά και χρησιμοποιήθηκε ο διπλάσιος όγκος αιτημάτων από αυτών του ιστότοπου «Google.gr».

Οι παράμετροι των δοκιμών που πραγματοποιήθηκαν είναι επιγραμματικά οι παρακάτω:

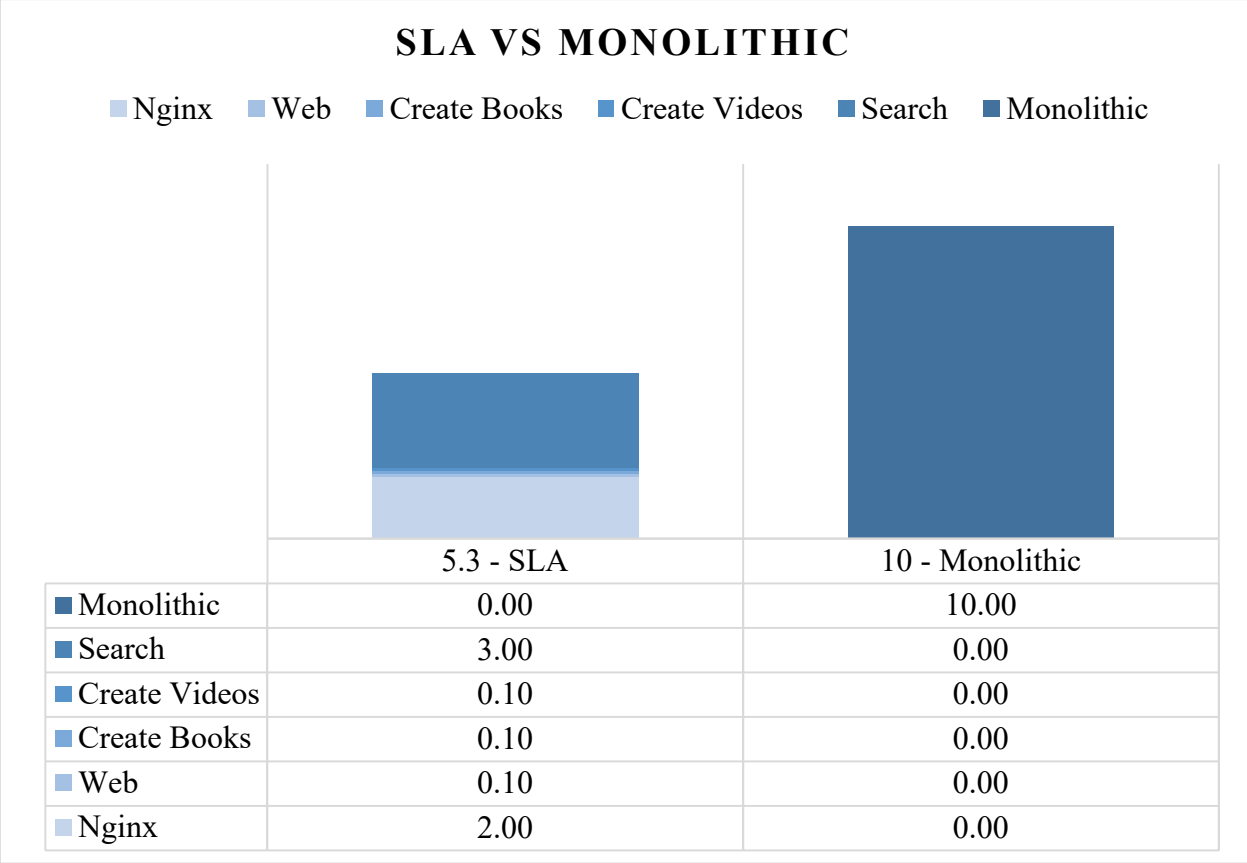
Σύνολο Videos/Books	150
Threads	750
Loop Count	500
Παρατηρήσεις	375,000
Υπηρεσία	Search
Χρόνος	~15 λεπτά

Πίνακας 4-3: Εφικτό Σενάριο- Παράμετροι

Μετά από επανειλημμένες δοκιμές διεξήχθησαν τα παρακάτω αποτελέσματα:

Σενάρια	Υποδοχείς	Αριθμός Πυρήνων	Μέσος όρος	95% Line
SLA	3 Search – 2 Nginx	5.3	1,931	2,309
Monolithic	10 Monolithic	10	1,737	2,611

Πίνακας 4-4: Εφικτό Σενάριο – Σύγκριση



	5.3 - SLA	10 - Monolithic
Monolithic	0.00	10.00
Search	3.00	0.00
Create Videos	0.10	0.00
Create Books	0.10	0.00
Web	0.10	0.00
Nginx	2.00	0.00

Διάγραμμα 4-5: SLA Vs Monolithic

Στο πίνακα 4-4 στη στήλη «Υποδοχείς» παρουσιάζονται οι υποδοχείς, οι οποίοι είναι απαραίτητοι για την εκτέλεση των δοκιμών. Στο σενάριο με την αρχιτεκτονική των μικροϋπηρεσιών χρειάστηκαν 3 υποδοχείς για την υπηρεσία search και 2 για την υπηρεσία Nginx, όπως απεικονίζεται και στο διάγραμμα 4-5. Ωστόσο, για τη λειτουργία ολόκληρης της εφαρμογής θα χρειαστούν επιπλέον πυρήνες για το λειτουργικό σύστημα αλλά και για τις περαιτέρω μικροϋπηρεσίες. Επιπλέον, λαμβάνοντας υπόψιν τα τιμολογιακά πλάνα που προσφέρει η Amazon Web Services , προκύπτει ο παρακάτω πίνακας:

Σενάριο	Αριθμός Πυρήνων	Ανάγκες στο Νέφος	Συνολικό ποσό
SLA	5.3	8	$747
Monolithic	10	16	$1494

Πίνακας 4-5: Εφικτό σενάριο - Οικονομικά στοιχεία

Από τους παραπάνω πίνακες γίνεται εμφανές ότι με τους μισούς πόρους είναι εφικτό να εξυπηρετηθούν τα ίδια αιτήματα με καλύτερους χρόνους ανταπόκρισης, εφόσον κατανεμηθούν αποτελεσματικά οι πόροι. Επιπρόσθετα, γίνεται κατανοητό ότι τόσο το χρηματικό κόστος της μονολιθικής αρχιτεκτονικής, στο υπολογιστικό νέφος, όσο και οι πυρήνες κεντρικού επεξεργαστή που βρίσκονται σε χρήση, είναι κατά 150% μεγαλύτερα, από αυτά των μικροϋπηρεσιών, σε μηνιαία βάση.

4.3.2 Ανέφικτο Σενάριο

Παράλληλα με το παραπάνω σενάριο έγινε προσπάθεια διεξαγωγής δοκιμών με παραπάνω από 375,000 αιτήματα μέσα σε 15 λεπτά. Αυτές οι δοκιμές οδηγούσαν σε ένα ποσοστό λάθους μεγαλύτερο του 10% στις απαντήσεις των παρατηρήσεων, καθιστώντας τα αποτελέσματα αναξιόπιστα. Τα δεδομένα του συγκεκριμένου σεναρίου παρουσιάζονται στους πίνακες που ακολουθούν.

Σύνολο Videos/Books	150
Threads	2,000
Loop Count	500
Παρατηρήσεις	1,000,000
Υπηρεσία	Search
Χρόνος	~15 λεπτά

Πίνακας 4-6: Ανέφικτο Σενάριο - Πληροφορίες

Σενάριο	Υποδοχείς	Χρήση Πυρήνων	Μέσος όρος	95% Line	Error %
SLA	7 Search – 5 Nginx	12.3	1,498	4,142	1.76%
Monolithic	15 Monolithic	15	1,322	2,927	20.56%

Πίνακας 4-7: Ανέφικτο Σενάριο - Σύγκριση

Από τον παραπάνω πίνακα γίνεται κατανοητό το γεγονός ότι το σενάριο SLA με λιγότερους πόρους διεξάγει καλύτερα αποτελέσματα εφόσον έχει πολύ μικρότερο ποσοστό λάθους από ότι το μονολιθικό και ότι, οι δείκτες, μέσος όρος και 95% των παρατηρήσεων, είναι υψηλότεροι, διότι υπάρχουν πολλά λιγότερα λάθη με αποτέλεσμα να αυξάνονται αισθητά.

4.4 Συμπεράσματα Συγκριτικής Ανάλυσης

Από την πειραματική ανάλυση και την σύγκριση των σεναρίων, σε θεωρητικό αλλά και σε πρακτικό επίπεδο, επιτεύχθηκε η εξαγωγή μετρήσιμων συμπερασμάτων τόσο για την μονολιθική σε σύγκριση με τη αρχιτεκτονική των μικροϋπηρεσιών όσο και για τον συνήθη τρόπο κλιμάκωσης των μικροϋπηρεσιών σε σύγκριση με τον προτεινόμενο.

Η μελέτη καταλήγει ότι, στο σενάριο Monolithic χρησιμοποιήθηκαν 5 πυρήνες από τους 8, ενώ στα σενάρια «CPU Usage» και SLA, 1.8 και 1.4 από τους 8 πυρήνες, αντίστοιχα. Εάν η σύγκριση γίνει στο σύνολο της εφαρμογής και μεταξύ των δύο αρχιτεκτονικών υπάρχει μείωση χρήσης πόρων κατά $\frac{5-1.8}{5} = 64\%$ και κατά $\frac{5-1.4}{5} = 72\%$ εάν ληφθεί υπόψιν και η προτεινόμενη μέθοδος κλιμάκωσης με βάση το χρόνο ανταπόκρισης των αιτημάτων. Επιπρόσθετα, εάν οι διαφορές υπολογιστούν αποκλειστικά για την μικροϋπηρεσία Search τότε υπάρχει εξοικονόμηση πόρων κατά $\frac{5-0.5}{5} = 90\%$ και $\frac{5-0.1}{5} = 98\%$. Φυσικά, η τελευταία σύγκριση δε μπορεί να έχει υπόσταση χωρίς και τις υπόλοιπες μικροϋπηρεσίες, πέρα της Search, αλλά ο σκοπός που

εξυπηρετεί αυτή η σύγκριση είναι, να τονιστεί η εξοικονόμηση των πόρων που μπορεί να υπάρξει μέσω του διαχωρισμού των λειτουργιών μιας εφαρμογής, που προσφέρει η αρχιτεκτονική των μικροϋπηρεσιών, αλλά και η εύκολη δυνατότητα κλιμάκωσης των υποδοχέων, όπως φάνηκε και στις δοκιμές.

Επιπρόσθετα, όσον αφορά τα οικονομικά οφέλη που έχει να προσφέρει η αρχιτεκτονική των μικροϋπηρεσιών, τουλάχιστον σε θεωρητικό επίπεδο, είναι εκπληκτικά. Η έρευνα έδειξε ότι το κόστος του μονολιθικού σεναρίου είναι κατά 2.74 φορές μεγαλύτερο από το CPU σενάριο και 3.59 από το SLA σενάριο. Η αρχιτεκτονική των μικροϋπηρεσιών σε συνδυασμό με το τρόπο κλιμάκωσης βάση του χρόνου ανταπόκρισης, εκμεταλλεύεται δύο πολύ σημαντικά στοιχεία με σκοπό την όσο το δυνατόν μεγαλύτερη μείωση του κόστους και κατ' επέκταση αύξηση των κερδών της επιχείρησης. Η εκμετάλλευση υφίσταται, πρώτον, στην δυνατότητα που παρέχει η αρχιτεκτονική στο να κατανέμει ξεχωριστά όλες τις λειτουργίες, σε μεμονωμένα μηχανήματα, με σκοπό την αποτελεσματικότερη χρήση των διαθέσιμων υπολογιστικών πόρων και δεύτερον, στην δυνατότητα μέγιστης χρήσης των παραπάνω πόρων, αρκεί το 95% των περιπτώσεων να μην υπερβούν τα 3 δευτερόλεπτα, βάσει της συμφωνίας σε επίπεδο υπηρεσίας.

Τέλος, κατά την μελέτη περίπτωσης στο μονολιθικό σενάριο χρησιμοποιήθηκαν οι διπλάσιοι πόροι από ότι στο σενάριο που χρησιμοποιήθηκε η αρχιτεκτονική των μικροϋπηρεσιών. Ο σκοπός αυτής της μελέτης περίπτωσης επιτεύχθηκε καθώς, σε ρεαλιστικό σενάριο, υπάρχει μια ισόποση μείωση χρήσης υπολογιστικών πόρων και χρηματικού κόστους κατά $\frac{16-8}{16} = 50\%$ και $\frac{1494-747}{1494} = 50\%$, αντίστοιχα.

5 Επίλογος

Οι δύο βασικοί στόχοι της διατριβής ήταν, πρώτον, η σύγκριση ανάμεσα στην μονολιθική αρχιτεκτονική και στην προοπτική των μικροϋπηρεσιών και, δεύτερον, η εύρεση μιας αποτελεσματικότερης μεθόδου κλιμάκωσης με γνώμονα πάντα την εξοικονόμηση υπολογιστικών και χρηματικών πόρων. Και οι δύο προαναφερόμενοι στόχοι επιτεύχθηκαν, εφόσον υπήρξε όσο το δυνατό καλύτερη σύγκριση ανάμεσα στις αρχιτεκτονικές, με σκοπό την εξαγωγή συμπερασμάτων, αλλά και προτάθηκε αποτελεσματικότερος τρόπος κλιμάκωσης των υποδοχέων, με δοκιμές οι οποίες υποστηρίζουν τα παραπάνω.

5.1 Σύνοψη και συμπεράσματα

Όπως αναφέρθηκε και στη βιβλιογραφική μελέτη τα θετικά που προκύπτουν από την αρχιτεκτονική των μικροϋπηρεσιών επικρατούν έναντι της μονολιθικής, και αυτό υποστηρίζεται κυρίως από τα κεφάλαια 2 και 3. Το κεφάλαιο «Πειραματική Ανάπτυξη» δείχνει τον κώδικα να είναι κατανεμημένος σε κάθε μία μικροϋπηρεσία, το οποίο τον καθιστά εύκολα κατανοητό από τον προγραμματιστή. Επιπρόσθετα, η ανάπτυξη και η ένταξη καινούριου κώδικα με τη χρήση του docker είναι λιγότερο χρονοβόρα και μειώνει αισθητά τον κίνδυνο λάθους.

Το χαμηλό κόστος και η εξοικονόμηση πόρων παίζει σημαντικό ρόλο στην απόφαση μιας επιχείρησης να επιλέξει τη σωστή αρχιτεκτονική για την ανάπτυξη των εφαρμογών της. Στο 4ο κεφάλαιο έγινε ξεκάθαρη η επικράτηση των μικροϋπηρεσιών επί της μονολιθικής αρχιτεκτονικής μέσω των συγκρίσεων. Οι συγκρίσεις αυτές έδειξαν ότι, μέσω της αρχιτεκτονικής των μικροϋπηρεσιών και της αποδοτικότερης μεθόδου κλιμάκωσης των υποδοχέων, δηλαδή της κλιμάκωσης με βάση το χρόνο ανταπόκρισης, μπορεί να εξοικονομηθεί σημαντικό χρηματικό ποσό. Ωστόσο, το σημαντικότερο όλων είναι η εξοικονόμηση που πραγματοποιήθηκε στους πόρους, όπως έδειξαν και τα συμπεράσματα του 4ου κεφαλαίου.

Από την άλλη πλευρά, τα κύρια μειονεκτήματα των μικροϋπηρεσιών επικεντρώνονται γύρω από τη πολυπλοκότητα ανάπτυξης εφαρμογών με μικροϋπηρεσίες και στο χρόνο που χρειάζεται η επιλογή της συνδεσμολογίας και οι δοκιμές που απαιτούνται για την επιλογή του κατάλληλου αριθμού των υποδοχέων που κρίνονται απαραίτητοι. Η αντιμετώπιση της πρώτης πρόκλησης επήλθε με τη χρήση διάφορων χρήσιμων εργαλείων όπως είναι το nginx, ενώ η πολυπλοκότητα της συνδεσμολογίας και η εύρεση της αποτελεσματικότερης, μέσω μελέτης των εγχειριδίων και επανάληψης των δοκιμών.

5.2 Όρια και περιορισμοί της έρευνας

Στην αρχική φάση σχεδιασμού της εφαρμογής με την αρχιτεκτονική των μικροϋπηρεσιών έγινε προσπάθεια ανάπτυξης της εφαρμογής στην πλατφόρμα Kubernetes. Αυτή η απόφαση λήφθηκε μετά από έρευνα και εξαγωγή συμπερασμάτων, σύμφωνά με τα οποία, αυτή η πλατφόρμα προσφέρει περισσότερες επιλογές-εργαλεία παραμετροποίησης, ενορχήστρωσης και ανάπτυξης εφαρμογών με την αρχιτεκτονική των μικροϋπηρεσιών. Εν τέλει, κάτι τέτοιο δεν κατέστη εφικτό, διότι η ανάπτυξη μιας μικρού μεγέθους εφαρμογής στην πλατφόρμα Kubernetes δημιουργεί παραπάνω πολυπλοκότητα από ότι ήταν διαχειρίσιμη.

Επιπρόσθετα, κατά τη διάρκεια των δοκιμών αντιμετωπίστηκαν περιορισμοί ως προς τους διαθέσιμους υπολογιστικούς πόρους. Έγινε προσπάθεια για περισσότερες δοκιμές με μεγαλύτερο αριθμό αιτημάτων όπως φάνηκε στο κεφάλαιο «4.3.1 - Ανέφικτο Σενάριο», των οποίων τα αποτελέσματα δε μπορούν να ληφθούν υπόψιν εφόσον το ποσοστό λάθους είναι αρκετά υψηλό.

Τέλος, όσον αφορά την οικονομική ανάλυση, οι περιορισμοί που αντιμετωπίστηκαν ήταν η εύρεση των τιμολογιακών πλάνων, αλλά γρήγορα επήλθε λύση μέσω της ιστοσελίδας «www.cloudorado.com». Ωστόσο, στην πραγματικότητα, μπορεί να υπήρχαν επιπρόσθετα κόστη. Οι οικονομική πόροι που διατέθηκαν στην διατριβή δεν επαρκούσαν ώστε να πραγματοποιηθεί μια μελέτη περίπτωσης στο υπολογιστικό νέφος.

5.3 Μελλοντικές Επεκτάσεις

Διεξήχθησαν επιπρόσθετες δοκιμές με μεγαλύτερο αριθμό παρατηρήσεων τα οποία δεν συμπεριλήφθηκαν στην διατριβή διότι το ποσοστό λάθους ήταν σε ποσοστό μεγαλύτερο του 50%. Πιθανοί λόγοι δυσκολίας των παραπάνω παρατηρήσεων είναι ότι είτε η βάση δέχεται ορισμένο αριθμό συνδέσεων είτε οι ίδιοι οι υποδοχείς δεν μπορούν να ανταποκριθούν. Παρατηρήθηκε ότι αυξάνοντας τους υποδοχείς μειώνεται το ποσοστό λάθους αλλά πάλι μέχρι τον αριθμό των 15 υποδοχέων. Πέραν αυτού του αριθμού δεν παρατηρήθηκε κάποια αλλαγή. Το παραπάνω θα μπορούσε να υπάρξει αντικείμενο μελλοντικής μελέτης.

Επιπλέον, λόγω του περιορισμένου χρόνου, αναπτύχθηκε μόνο ένα είδος εφαρμογής. Αντικείμενο μελλοντικής επέκτασης θα μπορούσε να αποτελούν οι εφαρμογές των οποίων η ανάπτυξη με την αρχιτεκτονική των μικροϋπηρεσιών, δε θεωρείται η βέλτιστη επιλογή. Σε συνδυασμό με το παραπάνω, οι δοκιμές θα μπορούσαν να πραγματοποιηθούν και σε παραπάνω από ένα μηχανήματα με σκοπό την εύρεση και την λύση διάφορων προβλημάτων που θα μπορούσαν να προκύπτουν, εξαιτίας της συνδεσμολογίας.

Τέλος, η κλιμάκωση των υποδοχέων, στη διάρκεια των δοκιμών της διατριβής, συνέβαινε χειροκίνητα. Αντικείμενο μελλοντικής έρευνας θα μπορούσε να αποτελεί η αυτοματοποίηση της κλιμάκωσης, βάσει του χρόνου ανταπόκρισης των αιτημάτων και σύμφωνα πάντα με τη συμφωνία σε επίπεδο υπηρεσιών.

Βιβλιογραφία

[1] Babak Bashari Rad, Harrison John Bhatti, Mohammad Ahmadi - An Introduction to Docker and Analysis of its Performance

[2] Carneiro Jr, C. and Schmelmer, T., Polyglot services. In Microservices From Day One, Springer, 2016

[3] Clemson, T., Testing Strategies in a Microservice Architecture, 2014

[4] Fowler, M., Microservice trade offs, 2015

[5] Kharenko, A., Monolithic vs Microservices Architecture, 2015

[6] Lewis, J. and Fowler, M., Microservices a definition of this new term, 2014

[7] Martin, R. C., The Single Responsibility Principle, 2009

[8] Mell, Peter M., and Timothy Grance, Sp 800-145. the nist definition of cloud computing, 2011

[9] Newman, S., Building microservices. O'Reilly Media, Inc., United States of America, 2015.

[10] Ranchal, R., Mohindra, A., Manweiler, J. G. and Bhargava, B., Radical strategies for engineering web-scale cloud solutions

[11] Richardson, C., Monolithic architecture, 2015

[12] Vaquero, Luis M., Luis Rodero-Merino, Juan Caceres, and Maik Lindner, A break in the clouds: towards a cloud definition, 2015

[13] Vase, T. Advantages of Docker, 2015

[14] Waldspurger, C. A, Memory resource management in VMware ESX server, 2002

[15] Wampler, D. and Clark, T., Guest editors' introduction: multiparadigm programming, 2010

[16] Introduction to JSON, διαθέσιμο στο «https://www.digitalocean.com/community/tutorials/an-introduction-to-json»

[17] Introduction to Node.js, διαθέσιμο στο «https://nodejs.dev/»

[18] NGINX – Wiki, διαθέσιμο στο «https://en.wikipedia.org/wiki/Nginx»

[19] Ορισμός υπολογιστικού νέφους, διαθέσιμο στο «https://en.wikipedia.org/wiki/Cloud_computing»

[20] Introduction to Jmeter, διαθέσιμο στο «https://www.guru99.com/introduction-to-jmeter.html»

[21] Service Level Objectives, διαθέσιμο στο «https://landing.google.com/»

[22] Στοιχεία «google.gr», διαθέσιμο στο «https://www.similarweb.com/website/google.gr»

[23] Στοιχεία «netflix.gr», διαθέσιμο στο «https://www.similarweb.com/website/netflix.com»

[24] Find out how you stack up to new industry benchmarks for mobile page speed, διαθέσιμο στο https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks/

[25] Ορισμός SLA , διαθέσιμο στο «https://en.wikipedia.org/wiki/Service-level_agreement»

[26] Στοιχεία συγκρίσεων παρόχων υπολογιστικού νέφους, διαθέσιμο στο «https://www.cloudorado.com/cloud_server_comparison»

[27] Στοιχεία τιμολογιακών πλάνων υπολογιστικού νέφους, διαθέσιμο στο «https://aws.amazon.com/»