ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Απόδοση Σύγχρονων Νεφοϋπολογιστικών Υπηρεσιών

Αικατερίνη Αγαθαγγελίδη

Α.Μ. 711151108

Εισηγητής: Δημήτριος Καλλέργης, Λέκτορας Εφ.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Απόδοση Σύγχρονων Νεφοϋπολογιστικών Υπηρεσιών

Αικατερίνη Αγαθαγγελίδη

Α.Μ. 711151108

Τριμελής εξεταστική επιτροπή

1. **Επιβλέπων καθηγητής: Δημήτριος Καλλέργης,** Λέκτορας Εφ. Πανεπιστημίου Δυτικής Αττικής

   Digitally signed by Dimitrios Kallergis
   Date: 2023.07.10 19:36:09 +03'00'

2. **Μέλος: Κωνσταντίνος Μαυρομμάτης,** Λέκτορας Πανεπιστημίου Δυτικής Αττικής

   Konstantinos Mavrommatis
   Digitally signed by Konstantinos Mavrommatis
   Date: 2023.07.11 11:16:01 +03'00'

3. **Μέλος: Ζαχαρένια Γαροφαλάκη**, ΕΔΙΠ Πανεπιστημίου Δυτικής Αττικής

   Digitally signed by Zacharenia Garofalaki
   Date: 2023.07.11 11:33:42 +03'00'

Αθήνα, Ιούλιος 2022

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Η κάτωθι υπογεγραμμένη Αγαθαγγελίδη Αικατερίνη του Παναγιώτη, με αριθμό μητρώου 711151108, φοιτήτρια του Προπτυχιακού Προγράμματος Σπουδών του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής, δηλώνω ότι:

«Είμαι συγγραφέας αυτής της διπλωματικής εργασίας και κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του διπλώματός μου».

Η Δηλούσα

Digitally signed by Dimitrios Kallergis
Date: 2023.07.10 19:36:32 +03'00'

Περίληψη

Η καθιέρωση της χρήσης υπηρεσιών Υπολογιστικού νέφους από οργανισμούς για τη φιλοξενία της υποδομής και των υπηρεσιών του, έχει επιφέρει πληθώρα από οφέλη, τόσο στη λειτουργία τους όσο και στην εμπειρία του καταναλωτή σε σχέση με τα προϊόντα τους. Παράλληλα, με τη χρήση εργαλείων περιεκτών, οι νεφοϋπολογιστικές υπηρεσίες έχουν τη δυνατότητα να εκτελούν εργασίες, κάθε μία από τις οποίες είναι απομονωμένη, επιτυγχάνοντας με αυτόν τον τρόπο υψηλά επίπεδα διαθεσιμότητας αλλά και ευκολίας στην αντιμετώπιση προβλημάτων. Σκοπός της διπλωματικής αυτής εργασίας είναι να αναδείξει τόσο τη διαδικασία ανάπτυξης και λειτουργίας, μίας υπηρεσίας στο υπολογιστικό νέφος όσο και την πληθώρα εργαλείων που μπορούν να αντληθούν από αποθετήρια κώδικα.

Λέξεις κλειδιά: Υπολογιστική Νέφους, Περιέκτες, Αποθετήρια κώδικα, Διαθεσιμότητα

Abstract

With the establishment of the Cloud services for hosting their infrastructure and services, organizations are witnessing an abundance of advantages regarding their operations, but also regarding the customer experience. Furthermore, using containers, cloud apps have the ability to run services which are isolated, resulting in high availability but also in a more effortless troubleshooting environment. The goal of this thesis, using an example application, is to demonstrate the process of developing and operating a service on cloud, and the abundance of tools that can be drawn from code repositories and be used for an enhanced functionality of the application, whatever its subject is.

Keywords: Cloud, Containers, Repositories, Availability

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Καλλέργη Δημήτριο για την καθοδήγησή που μου προσέφερε για την εργασία αυτή.

Επίσης, θα ήθελα να ευχαριστήσω την οικογένειά μου για την υποστήριξη όλα τα χρόνια των σπουδών μου.

Περιεχόμενα

Κατάλογος Εικόνων

ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ

1.1 Πρόβλημα

Η συνεχής επέκταση της χρήσης εφαρμογών και υπηρεσιών που παρέχονται μέσω διαδικτύου στη σημερινή εποχή έχει φέρει τους επαγγελματίες μπροστά σε ένα πολύ σημαντικό και δύσκολο να παραβλεφθεί πρόβλημα, αυτό της συνεχούς αυξανόμενης ανάγκης για υπολογιστικούς πόρους. Η μέχρι τώρα προσέγγιση της λύσης του ζητήματος δεν είναι πια βιώσιμη. Η κοστοβόρα και χρονοβόρα αγορά και εγκατάσταση νέων πόρων μαζί με τους ήδη υπάρχοντες δεν ανταποκρίνεται πάντα και στο σωστό βαθμό στις ανάγκες ακόμα και τις πιο απλής εφαρμογής που μπορεί να στηρίζουν αυτά τα μέσα. Σε αυτό το ζήτημα λύση έρχεται να δώσει το Υπολογιστικό Νέφος. Το Νέφος, τόσο με τη χρήση του μοντέλου πληρωμής ανάλογα με τη χρήση όσο και με τις λιγοστές απαιτήσεις του σχετικά με την υποδομή του εκάστοτε χρήστη του, αποτελεί ίσως την πιο σημαντική λύση στα χέρια των προγραμματιστών, και όχι μόνο, για να αναπτύξουν τις υπηρεσίες που χρειάζονται χωρίς την αναμονή της αγοράς και της εγκατάστασης νέων εξαρτημάτων στον ήδη υπάρχον εξοπλισμό, και με σημαντικά μειωμένο κόστος

Στα θετικά του Υπολογιστικού Νέφους, έρχονται να προστεθούν και να αυξήσουν σε μεγάλο βαθμό τα προτερήματά του οι μικροϋπηρεσίες. Με τη χρήση των μικροϋπηρεσιών η εκμετάλλευση των πόρων του Νέφους γίνεται όχι μόνο ακόμα πιο αποδοτική αλλά και απλούστερη. Βασικό παράδειγμα αυτού αποτελεί η δέσμευση των απαραίτητων υπολογιστικών πόρων ανάλογα με την ανάγκη της υπηρεσίας εκείνη τη στιγμή. Σημαντικό επίσης πλεονέκτημά τους αποτελεί η μικρή έκταση του κώδικά τους η οποία τις κάνει πιο εύκολα διαχειρίσιμες τόσο από εκείνους που τις αναπτύσσουν όσο και από όσους τις βλέπουν για πρώτη φορά. Τα προτερήματα των μικροϋπηρεσιών έρχονται να αυξήσουν και να συμπληρώσουν οι νέες ανοιχτές τεχνολογίες που υποβοηθούν την ανάπτυξη νέων εφαρμογών που ανταποκρίνονται σε κάθε ανάγκη, ανεξάρτητα από το αντικείμενό της.

1.2. Στόχοι

Στόχος της εργασίας αυτής είναι η εξαγωγή συμπερασμάτων από τη χρήση της τεχνολογίας των μικροϋπηρεσιών για τη δημιουργία μίας νέας εφαρμογής είτε περιορισμένης είτε ευρείας κλίμακας χρήσης. Παράλληλα, ως βασικός στόχος αναγνωρίζεται η συνεισφορά των υπηρεσιών υπολογιστικού νέφους στη λειτουργία υπηρεσιών και εφαρμογών. Η αξιολόγηση αυτή βασίζεται στο κατά πόσο ωφέλησε η χρήση ενός υπολογιστικού νέφους για την ανάπτυξη της υπηρεσίας σε αντίθεση με την χρήση αποκλειστικά των πόρων που προσφέρει ένας ή περισσότεροι υλικοί εξυπηρετητές. Αρχικά η ανάλυση έγινε σε θεωρητικό επίπεδο, αναλύθηκαν τα πλεονεκτήματα και τα μειονεκτήματα των

τεχνολογιών και στη συνέχεια μελετήθηκαν μέσα από την εφαρμογή τους, σε μία υπηρεσία αξιολόγησης και πρότασης ταινιών σε πρακτικό επίπεδο. Τέλος, είναι πολύ σημαντική η αναγνώριση και η αξιολόγηση ανοιχτών τεχνολογιών που μπορούν να ωφελήσουν οποιονδήποτε αναζητά βοήθεια στη δημιουργία της δικής του νέας εφαρμογής, αλλά δεν έχει όλες τις κατάλληλες γνώσεις τα τη δημιουργήσει από την αρχή.

1.3 Συνεισφορά

Για τη βιβλιογραφική μελέτη που παρουσιάζεται σε αυτή την εργασία αναζητήθηκαν πληροφορίες σε σχέση με την προέλευση και τα χαρακτηριστικά του υπολογιστικού νέφους, τα πλεονεκτήματα και τα μειονεκτήματα της μονολιθικής αρχιτεκτονικής και της αρχιτεκτονικής μικροϋπηρεσιών, τα χαρακτηριστικά της τεχνολογίας εικονικοποίησης Λειτουργικών Συστημάτων και πιο συγκεκριμένα της πλατφόρμας Docker, και τέλος οι βασικές λειτουργίες των υπηρεσιών που προσφέρονται από το αποθετήριο Spring Cloud.

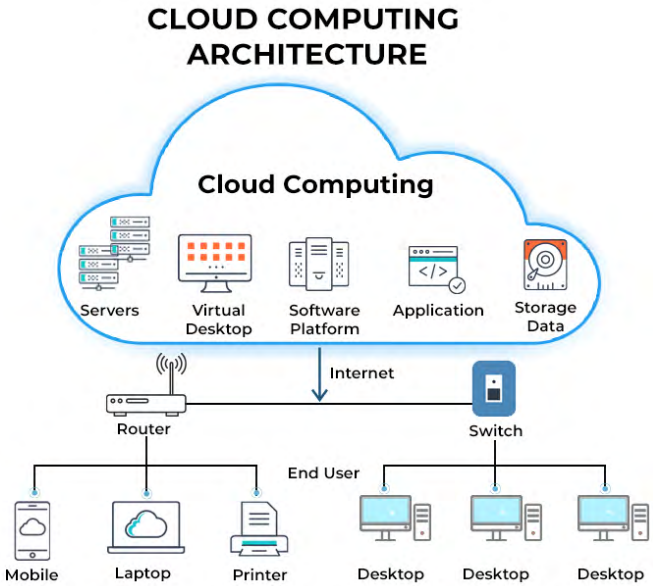
1.4 Διάρθρωση μελέτης

Στο 2^{ο} κεφάλαιο της διπλωματικής αυτής προσεγγίζονται θεωρητικά οι έννοιες που σχετίζονται με την Υπολογιστική Νέφους και την Αρχιτεκτονική της. Στο 3^{ο} κεφάλαιο αναλύονται τα εργαλεία τα οποία χρησιμοποιήθηκαν στην υπηρεσία που μελετάται στην εργασία αυτή. Στο 4^{ο} κεφάλαιο παρουσιάζεται το πειραματικό μέρος της εργασίας και το τελικό αποτέλεσμα της υπηρεσίας. Στο 5^{ο} κεφάλαιο περιγράφονται τα όρια της έρευνας στα πλαίσια της εργασίας, οι δυσκολίες που συναντήθηκαν κατά την υλοποίηση και τα συμπεράσματα που προέκυψαν από αυτή. Τέλος, το 6^{ο} κεφάλαιο πραγματεύεται τις πιθανές μελλοντικές επεκτάσεις της υπηρεσίας που μελετήθηκε στα πλαίσια της διπλωματικής αυτής.

ΚΕΦΑΛΑΙΟ 2: ΘΕΩΡΗΤΙΚΗ ΠΡΟΣΕΓΓΙΣΗ

2.1 Υπολογιστική Νέφους

2.1.1 Ιστορικότητα και ορισμοί



Εικόνα 1 Αρχιτεκτονική υπολογιστικής νέφους, πηγή: www.spiceworks.com

Ο όρος Υπολογιστικό Νέφους εμφανίστηκε πρώτη φορά το 2006. Τότε εμφανίστηκαν οι πρώτες υπηρεσίες cloud από την εταιρία Amazon με όνομα Elastic Compute Cloud (EC2). Η Amazon διέθετε τους υπολογιστικούς πόρους της προς εκμίσθωση από εταιρείες και οργανισμούς με σκοπό να τους χρησιμοποιούν για να «τρέχουν» τις εφαρμογές τους. [1],[4]

Η ιδέα, όμως, για υπολογιστικούς πόρους που θα είναι διαθέσιμοι μέσω δημόσιων υπηρεσιών γίνεται λόγος από τη δεκαετία του 1960, με χαρακτηριστικό παράδειγμα τη δημόσια δήλωση του επιστήμονα John McCarthy «Αν οι υπολογιστές του είδους του οποίου εγώ είμαι υπέρμαχος γίνουν οι υπολογιστές του μέλλοντος, τότε η υπολογιστική κάποια μέρα θα οργανώνεται ως μία δημόσια κοινωφελής υπηρεσία, όπως είναι το τηλεφωνικό σύστημα»[1].

Με τον όρο Υπολογιστική Νέφους, σύμφωνα με το NIST αναφερόμαστε σε: « Ένα μοντέλο για ενεργοποίηση της πανταχού παρούσας, βολικής, κατ' απαίτησης πρόσβασης στο δίκτυο από μια διαμοιρασμένη δεξαμενή διαμορφώσιμων υπολογιστικών πόρων (π.χ. δικτύων, εξυπηρετητών,

αποθήκευσης, εφαρμογών και υπηρεσιών), που μπορούν να παρέχονται και να αποδεσμεύονται γρήγορα, με ελάχιστη προσπάθεια διαχείρισης ή αλληλεπίδρασης από τον πάροχο της υπηρεσίας». Πιο απλοποιημένα, ως Υπολογιστική Νέφους ορίζουμε την κατ' απαίτηση διάθεση υπολογιστικών πόρων μέσω διαδικτύου με σκοπό τη φιλοξενία υπηρεσιών οργανισμών [1],[2],[3].

2.1.2 Χαρακτηριστικά Νέφους

Ένας οργανισμός, για να φιλοξενήσει την κάθε υπολογιστική του ανάγκη, μέχρι πριν δύο δεκαετίες θα έπρεπε να σπαταλήσει υπέρογκα χρηματικά ποσά ώστε να αγοράσει τον απαραίτητο βασικό εξοπλισμό και σε κάθε ανάγκη για επιπλέον πόρους θα έπρεπε να τον αναβαθμίζει. Έναν εξοπλισμό που θα είναι χρήσιμος μόνο από χρήστες που βρίσκονται κοντά σε αυτόν και που σε μικρό χρονικό διάστημα με μία αλλαγή στις ανάγκες του οργανισμού, μπορεί να καταστεί περιττός [1],[4].

Σε αυτά τα προβλήματα δίνει λύση η Υπολογιστική Νέφους, με τα έξι βασικά της χαρακτηριστικά, τα οποία περιγράφουν κάθε υποδομή ενός παρόχου υπηρεσιών νέφους, όπως ορίζονται και από τον NIST.

Κατ' απαίτηση εξυπηρέτηση

Με την κατ' απαίτηση εξυπηρέτηση αναφερόμαστε στην ιδιότητα των υπολογιστικών νεφών, να παρέχουν στον τελικό χρήστη τους πόρους που αναζητά χωρίς να χρειάζεται η συμβολή ενός φυσικού προσώπου [1].

Ευρεία δικτυακή πρόσβαση

Με την ευρεία δικτυακή πρόσβαση περιγράφεται η δυνατότητα που προσφέρουν οι υπηρεσίες υπολογιστικού νέφους, να είναι διαθέσιμες και προσπελάσιμες από οπουδήποτε υπάρχει σύνδεση στο διαδίκτυο και από οποιαδήποτε συσκευή [1].

Συνεκμετάλλευση πόρων

Με την συνεκμετάλλευση ή ομαδοποίηση πόρων, επιτρέπεται η χρήση των εικονικοποιημένων πόρων από πολλαπλούς χρήστες. Τα μοντέλα πολλαπλής μίσθωσης που αξιοποιεί η υπολογιστική νέφους, επιτρέπουν την δυναμική διανομή των υπολογιστών πόρων του νέφους ανάλογα με τις εκάστοτε ανάγκες των καταναλωτών [1],[4].

Ελαστικότητα

Η ελαστικότητα αναφέρεται στην ικανότητα του νέφους να προσφέρει την κλιμάκωση των υπολογιστικών πόρων ανάλογα με τις ανάγκες του εκάστοτε χρήστη, με βάση ένα προσυμφωνημένο, ανάμεσα στον καταναλωτή και τον πάροχο, πρωτόκολλο[1].

Μετρούμενη Υπηρεσία

Η μετρούμενη υπηρεσία είναι η παρακολούθηση της χρήσης των καταναλισκόμενων υπηρεσιών από ένα χρήστη ή έναν οργανισμό ώστε ο πάροχος του υπολογιστικού νέφους να τους αποδώσει τις αντίστοιχες χρεώσεις και τις σχετικές αναφορές που περιγράφουν αυτή την κατανάλωση.

Ένα τελευταίο χαρακτηριστικό, το οποίο δεν ορίζεται από το NIST, αλλά έχει καθιερωθεί λόγω της μεγάλης σημασίας του είναι η ανθεκτικότητα [1].

Ανθεκτικότητα

Ως ανθεκτικότητα περιγράφουμε την δυνατότητα που παρέχουν τα υπολογιστικά νέφη, να παρέχουν επιπλέον πόρους στους καταναλωτές, σε περίπτωση που οι πόροι που είχαν αρχικά δεσμεύσει για εκείνους έχουν κάποιο λειτουργικό πρόβλημα. Οι πόροι αυτοί μπορεί να βρίσκονται στην ίδια τοποθεσία με τους αρχικούς ή σε κάποια διαφορετική, σε ένα ή πολλαπλά αντίγραφα [1].

2.1.3 Μοντέλα Υπηρεσιών Νέφους

Οι υπηρεσίες των υπολογιστικών νεφών προσφέρονται σε τρία μοντέλα:

Υποδομή ως Υπηρεσία ή IaaS

Σε αυτό το μοντέλο υπηρεσιών, ο πάροχος προσφέρει στους καταναλωτές όλη την υποδομή που χρειάζονται για να δημιουργήσουν τα περιβάλλοντα που χρειάζονται σύμφωνα με τις ανάγκες τους. Πιο συγκεκριμένα, το νέφος τους παρέχει το υλικό (hardware), τη δικτυακή υποδομή, τους αποθηκευτικούς χώρους, τα προγράμματα προστασίας και τα προγράμματα διαχείρισης φόρτου κατ' απαίτηση. Με αυτόν τον τρόπο, ο καταναλωτής χρειάζεται μόνο να εγκαταστήσει τα λειτουργικά συστήματα που τον ενδιαφέρουν, και όποια άλλη υπηρεσία κρίνει απαραίτητη. Μέσω αυτού του μοντέλου, ένα βασικό πλεονέκτημα είναι πως ο χρήστης απελευθερώνεται από τις ανάγκες για προστασία της υλικής υποδομής [1],[6].

Πλατφόρμα ως Υπηρεσία ή PaaS

Το μοντέλο Πλατφόρμα ως Υπηρεσία πρόκειται για την προσφορά μίας ολοκληρωμένης υποδομής στον καταναλωτή, με σκοπό τη φιλοξενία ήδη υπαρχουσών εφαρμογών σε αυτό ή την ανάπτυξη νέων. Στα πλεονεκτήματα του περιβάλλοντος αυτού συγκαταλέγεται η δυνατότητα που δίνει στους προγραμματιστές να το παραμετροποιούν με ευκολία όποτε το κρίνουν αναγκαίο και να το μοιράζονται με άλλους ανθρώπους που συμμετέχουν στην δημιουργία της εκάστοτε εφαρμογής [5],[1].

Λογισμικό ως Υπηρεσία ή SaaS

Πρόκειται για μία ποικιλία λογισμικών που διατίθενται στους πελάτες των νεφών για χρήση μέσω του διαδικτύου. Οι καταναλωτές μπορούν να αγοράσουν την εκάστοτε υπηρεσία και να τη χρησιμοποιούν όπως και όποτε έχουν ανάγκη. Η υπηρεσία αυτή μπορεί να προέρχεται από τον πάροχο του νέφους αλλά και από κάποιον τρίτο [1],[6].

2.1.4 Μοντέλα Ανάπτυξης Νέφους

Υπάρχουν τέσσερα μοντέλα ανάπτυξης νεφών

Δημόσιο Νέφος

'Οταν αναφερόμαστε στο δημόσιο νέφος, θέλουμε να περιγράψουμε ένα δημόσια διαθέσιμο νέφος από κάποιο πάροχο, ή από παραπάνω από έναν παρόχους. Η υποδομή αυτή μπορεί να χρησιμοποιηθεί από οποιονδήποτε, με κάποια χρέωση [1],[7].

Κοινοτικό Νέφος

Πρόκειται για ένα τύπο ανάπτυξης νέφους παρόμοιο με το δημόσιο νέφος. Στην περίπτωση αυτή όμως χρήστες του είναι τα μέλη μίας κοινότητας και το νέφος παρέχεται από αυτήν την κοινότητα. Επίσης ως κοινοτικό νέφος μπορεί να ορισθεί το νέφος που παρέχει ένας ή παραπάνω οργανισμοί, αλλά είναι διαθέσιμο σε περιορισμένη ομάδα ανθρώπων. [1],[7].

Ιδιωτικό Νέφος

Το ιδιωτικό νέφος ανήκει σε ένα μόνο πάροχο. Η υποδομή του είναι διαθέσιμη στους χρήστες εμπιστοσύνης του εκάστοτε οργανισμού. Ο έλεγχος και η συντήρηση του ιδιωτικού νέφους γίνεται είτε από τον ίδιο τον οργανισμό είτε την αναλαμβάνει κάποιος εξωτερικός συνεργάτης [1],[7].

Υβριδικό Νέφος

Πρόκειται για ένα μοντέλου νέφους που συνδυάζει ένα ή παραπάνω από τους προαναφερθέντες τύπους νεφών. Χρησιμοποιούνται στην περίπτωση που ένας τύπος νεφών δεν καλύπτει απόλυτα τις ανάγκες του εκάστοτε οργανισμού. Επίσης, πλεονέκτημα του υβριδικού μοντέλου είναι ο διαμοιρασμός των ευθυνών διαχείρισης ανάμεσα στα συμβαλλόμενα μέρη [1],[7].

2.1.5 Πάροχοι Νεφών

Ως απόρροια των παραπάνω περιγραφών μπορούμε να συμπεράνουμε ότι οποιοσδήποτε οργανισμός επιθυμεί και διαθέτει τους χρηματικούς πόρους μπορεί να δημιουργήσει μία υποδομή νέφους. Παρ 'όλα αυτά μεγάλη μερίδα των υποψήφιων χρηστών του νέφους προτιμά την επιλογή του Δημόσιου

Νέφους, καθώς έχει το μεγάλο πλεονέκτημα της χρήσης του νέφους με τη λιγότερη χρηματική δαπάνη σε σχέση με τις υπόλοιπες επιλογές[8].

Όσον αφορά τα δημόσια υπολογιστικά νέφη, στην αγορά του σήμερα, υπάρχουν 10 πάροχοι που μονοπωλούν την αγορά. Σε έρευνα του 2022, αυτοί ορίζονται ως εξής :

Πάροχος νέφους	**Περιοχές**	**Ζώνες Διαθεσιμότητας**
Amazon Web Services (AWS)	26	84
Microsoft Azure	60	116
Google Cloud Platform (GCP)	34	103
Alibaba Cloud	27	84
Oracle Cloud	38	46
IBM Cloud (Kyndryl)	11	29
Tencent Cloud	21	65
OVHcloud	13	33
DigitalOcean	8	14
Linode (Akamai)	11	11

Πίνακας 2.1 *Κατάταξη παρόχων νέφους, πηγή: digitlinfra.com*

Πιο αναλυτικά:

1. Amazon Web Services

Οι υπηρεσίες νέφους που προσφέρει η Amazon έχουν βρεθεί στην κορυφή των προτιμήσεων των χρηστών. Προσφέρει 200 διαφορετικές υπηρεσίες στους καταναλωτές του νέφους της και έχει υποδομές στην Ευρώπη, την Ασία, την Αφρική, την Ωκεανία και τη Βόρειο και Νότιο Αμερική [8].

2. Microsoft Azure

Η Microsoft μέσω των υπηρεσιών του Azure, προσφέρει υπηρεσίες τόσο δημόσιου όσο και υβριδικού νέφους. Επίσης προσφέρει και άλλες υπηρεσίες μέσω του νέφους όπως το Office 365 και το Dynamic 365 [8].

3. Google Cloud Platform

Η υπηρεσίες νέφους της Google απευθύνονται σε επιχειρήσεις και οργανισμούς και επικεντρώνονται στην ανάπτυξη εφαρμογών τόσο σε δοκιμαστικό επίπεδο όσο και σε επίπεδο παραγωγής. Διαθέτει υποδομές σε Ευρώπη, Ασία, Ωκεανία και Βόρειο και Νότιο Αμερική [8].

4. Alibaba Cloud

Το Alibaba Cloud είναι ο κύριος πάροχος υπηρεσιών νέφους στην Ασία. Παρέχει πληθώρα δυνατοτήτων στους χρήστες της, από αποθήκευση και επεξεργασία δεδομένων έως υπηρεσίες τεχνητής νοημοσύνης. Οι υποδομές του βρίσκονται κυρίως στην Κίνα όμως έχει παρουσία και στην υπόλοιπη Ασία καθώς και την Ευρώπη και τη Βόρειο Αμερική [8].

5. Oracle Cloud

Η Oracle προσφέρει τόσο υπηρεσίες τύπου IaaS, για την υλοποίηση υποδομών στο νέφος, όσο και υπηρεσίες τύπου SaaS, αφού προσφέρει τα διάφορα λογισμικά της μέσω του νέφους. Διαθέτει υποδομές που εκτείνονται στις ηπείρους καθώς και υποδομές αποκλειστικά για κυβερνητικούς οργανισμούς, όπως η αμερικανική κυβέρνηση και η κυβέρνηση του Ηνωμένου Βασιλείου [8].

6. Kyndryl (IBM Cloud)

Η IBM μέσω του Kyndryl προσφέρει υπηρεσίες για την υλοποίηση και τη διαχείριση περιβαλλόντων δημοσίου, ιδιωτικού αλλά και υβριδικών νέφους. Επίσης η IBM έχει δημιουργήσει συνεργασίες με την Microsoft, την Amazon και το Google Cloud [8].

7. Tencent Cloud

Μετά το Alibaba Cloud, το Tencent Cloud είναι ο δεύτερος μεγαλύτερος πάροχος υπηρεσιών νέφους στην Ασία. Μέσω του Tencent Cloud παρέχονται στους καταναλωτές πλατφόρμες, για παράδειγμα, για υποστήριξη μουσικής και βίντεο ή ηλεκτρονικών πληρωμών [9].

8. OVHcloud

Το OVHcloud είναι ένας πάροχος νέφους επικεντρωμένος στην περιοχή της Ευρώπης που προσφέρει υπηρεσίες ιδιωτικού και δημόσιου νέφος. Θεωρείται ένας πάροχος πιο οικονομικών λύσεων για τις επιχειρήσεις, σε σχέση με τους μεγαλύτερους παρόχους [8].

9. DigitalOcean

Πρόκειται για ένα πάροχο νέφους, που απευθύνεται περισσότερο σε μικρές, μικρομεσαίες και start-up επιχειρήσεις. Πρόκειται για πάροχο επίσης επικεντρωμένο στην αγορά της Ευρώπης και με επίσης πιο οικονομικές λύσεις όπως το OVHCloud [8].

10. Linode (Akamai)

Και αυτός ο πάροχος επικεντρώνεται για σε υπηρεσίες για προγραμματιστές και μικρές ή νέες επιχειρήσεις. Επίσης προσφέρει και αυτός οικονομικές και ανταγωνιστικές τιμές σε σχέση με τους μεγαλύτερους παρόχους [8].

2.1.6 Κέντρα Δεδομένων

Τα κέντρα δεδομένων είναι οι χώροι στους οποίους στεγάζονται οι υλικές υποδομές των υπολογιστικών νεφών [10], [11].

Τα κέντρα δεδομένων αποτελούνται από τρία διαφορετικά στοιχεία. Αρχικά για να λειτουργεί το κέντρο χρειάζεται να υπάρχει η κατάλληλη δικτυακή υποδομή για να συνδέονται τόσο οι φυσικοί εξυπηρετητές και δομές αποθήκευσης δεδομένων μεταξύ τους, όσο και οι δομές που δημιουργούνται στο νέφος. Επιπλέον, βασικό στοιχείο τους είναι οι εξυπηρετητές, πάνω στους οποίους φιλοξενούνται όλες οι εικονικές υπηρεσίες. Τέλος, τα συστήματα αποθήκευσης δεδομένων, όπου αποθηκεύονται όλα τα δεδομένα των υποδομών που δημιουργούνται στους εξυπηρετητές του κέντρου δεδομένων [10], [11].

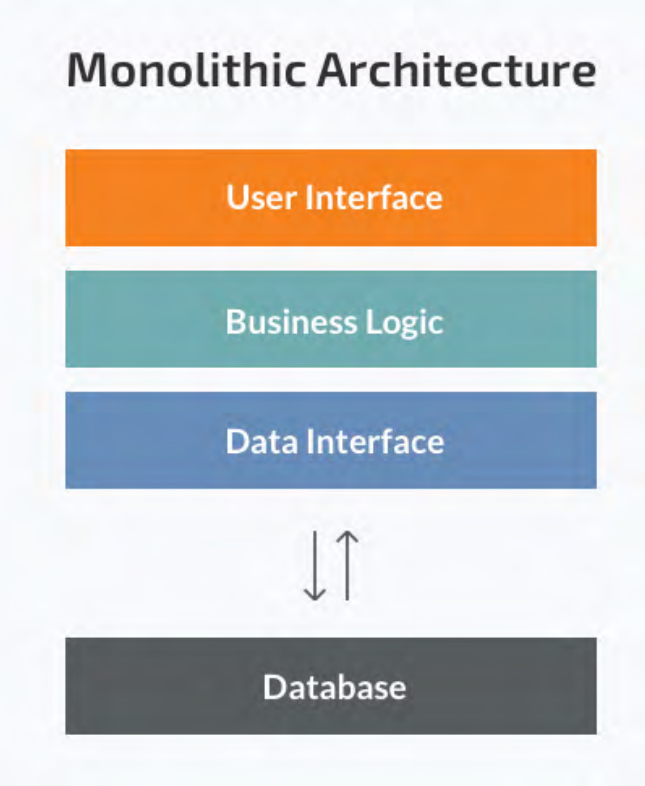
Τα κέντρα δεδομένων για να λειτουργούν αποτελεσματικά πρέπει να πληρώνουν κάποιες προϋποθέσεις. Αρχικά είναι αναγκαίο να έχουν ληφθεί όλα τα απαραίτητα μέτρα για την προστασία της υποδομής τους από φυσικές καταστροφές και από καταστροφές που μπορεί να προκληθούν από ανθρώπους. Αυτό συνήθως επιτυγχάνεται μέσω των πλεοναζουσών υποδομών, ώστε σε περίπτωση κάποιας καταστροφής, να μην υπάρξει διακοπή της λειτουργίας των υπηρεσιών και απώλεια δεδομένων. Επίσης χρειάζεται να έχουν το χαρακτηριστικό της ευκινησίας. Τα νέφη θα πρέπει να υποστηρίζουν κάθε υπηρεσία και κάθε ανάγκη για εξυπηρετητή που μπορεί να προκύψει από κάποιο πελάτη. Ακόμη, τα δίκτυα των κέντρων δεδομένων θα πρέπει να φροντίζουν μία κίνηση στο δίκτυο να μην επηρεάζεται από καμία άλλη, κάθε εξυπηρετητής να μπορεί να έχει διεύθυνση IP, να είναι εύκολη

η διαχείριση του και να μπορεί να διευρυνθεί ανάλογα με τις ανάγκες του κέντρου και των υπηρεσιών που στηρίζει [10], [11].

2.2 Αρχιτεκτονική

2.2.1 Μονολιθική Αρχιτεκτονική

Η μονολιθική αρχιτεκτονική είναι ένα παλιός τρόπος ανάπτυξης εφαρμογών. Πρόκειται για εφαρμογές ανεπτυγμένες πάνω σε μία στοίβα, σε έναν ενιαίο κώδικα [12].



Εικόνα 2 Μονολιθική αρχιτεκτονική, πηγή: www.n-ix.com

Η μονολιθική αρχιτεκτονική αποτελείται από τρεις στρώσεις:

1. Front-End. Το κομμάτι της εφαρμογής με το οποίο αλληλοεπιδρούν οι χρήστες.
2. Business Logic: Αφορά τις διεργασίες που γίνονται από την εφαρμογή για την εξαγωγή των αποτελεσμάτων, ανάλογα με τη λειτουργικότητά της προς τους χρήστες.
3. Back-End: Η στρώση της αρχιτεκτονικής στην οποία οφείλεται η επικοινωνία της εφαρμογής με τη βάση δεδομένων της ή με άλλες υπηρεσίες.

Τα πλεονεκτήματα της μονολιθικής αρχιτεκτονικής είναι:

- Η ευκολία στον εντοπισμό λαθών λόγω του ενιαίου κώδικα.
- Η ευκολία διενέργειας δοκιμαστικών λειτουργιών επίσης λόγω του ενιαίου κώδικα.
- Η μη πολυπλοκότητα της ανάπτυξης μίας τέτοιας εφαρμογής, καθώς δεν χρειάζονται πολλά αρχεία κώδικα.

Τα μειονεκτήματα της μονολιθικής αρχιτεκτονικής είναι:

- Η δυσκολία στην κατανόηση του κώδικα μετά το τέλος της ανάπτυξης της εφαρμογής, αφού πρόκειται για ένα ενιαίο και δυσνόητο από τρίτους κώδικα.
- Η δυσκολία που προκύπτει στην περίπτωση αλλαγής κάποιου σημείου του κώδικα, καθώς θα επηρεαστεί όλος ο κώδικας και όχι μόνο το κομμάτι που αλλάζει.
- Η δυσχέρεια που προκύπτει στην περίπτωση που πρέπει η εφαρμογή να ακολουθήσει κάποια νέα τεχνολογία, καθώς θα πρέπει να γραφεί από την αρχή.
- Η απουσία ευελιξίας, καθώς σε περίπτωση σε περίπτωση που κάποιο σημείο της εφαρμογής χρειαστεί επέκταση, θα χρειαστεί επέκταση ολόκληρου του συστήματος.[14]

Με τον συγκεκριμένο τύπο αρχιτεκτονικής προτείνεται να αναπτύσσονται μικρές και απλές διαδικτυακές εφαρμογές [13],[14],[15].

2.2.2 Αρχιτεκτονική Μικροϋπηρεσιών

Η αρχιτεκτονική μικροϋπηρεσιών αφορά την ανάπτυξη εφαρμογών μέσω της σύνθεσης μικρών διαφορετικών υπηρεσιών ανεξάρτητων η μία από την άλλη. Η επικοινωνία των υπηρεσιών για την εύρυθμη λειτουργία της εφαρμογής γίνεται μέσω του πρωτοκόλλου επικοινωνίας Representational State Transfer (REST) [12].

Τα πλεονεκτήματα της αρχιτεκτονικής μικροϋπηρεσιών είναι:

•Η δυνατότητα ανεξάρτητης επιλογής του κάθε εργαλείου που ταιριάζει στην εκάστοτε εφαρμογή.

•Η ευκολία αντιμετώπισης σφαλμάτων, καθώς ένα λάθος σε μία μικροϋπηρεσία δεν επηρεάζει τις υπόλοιπες.

•Η ευκολία στην κλιμάκωση της κάθε υπηρεσίας που τη χρειάζεται, καθώς είναι ανεξάρτητη μονάδα

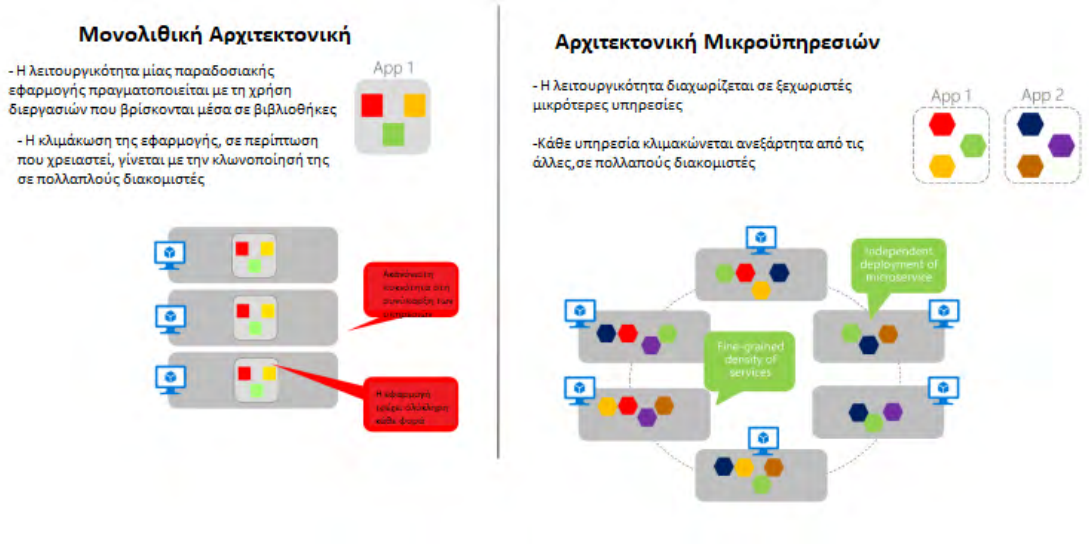
Τα μειονεκτήματα της αρχιτεκτονικής μικροϋπηρεσιών είναι:

•Η πολυπλοκότητα στη διαχείριση λόγω των πολλών συνδεόμενων συστημάτων.

•Η δυσκολία ενσωμάτωσης και διαχείρισης του γραφικού περιβάλλοντος

•Η δυσκολία στο συντονισμό των εξαρτήσεων κάθε διαφορετικού συστήματος που συνθέτει την εφαρμογή.

Η ανάπτυξη εφαρμογών με την αρχιτεκτονική μικροϋπηρεσιών είναι μία λύση για εφαρμογές που χρειάζεται να υλοποιούν πολύπλοκες λειτουργίες, αλλά και εφαρμογές που χρειάζεται να ανανεώνουν συχνά τις τεχνολογίες που χρησιμοποιούν [14],[15].



Εικόνα 3 Διαφορές μεταξύ Μονολιθικής αρχιτεκτονικής και Αρχιτεκτονικής Μικροϋπηρεσιών, πηγή: .NET Microservices Architecture for Containerized .NET Applications, Microsoft

2.2.3 Υπηρεσιοκεντρική Αρχιτεκτονική

Η Υπηρεσιοκεντρική Αρχιτεκτονική (Service Oriented Architecture – SOA), είναι η δόμηση μίας εφαρμογής με τρόπο τέτοιο, ώστε οι επιμέρους υπηρεσίες που την αποτελούν να μπορούν να επικοινωνούν μεταξύ τους μέσω του δικτύου, με τη χρήση πρωτοκόλλων επικοινωνίας [16],[17],[18].

Οι εκάστοτε υπηρεσίες που χρησιμοποιούνται σε αυτή την αρχιτεκτονική έχουν την ικανότητα να επικοινωνούν μεταξύ τους, ανεξαρτήτως της γλώσσας προγραμματισμού με βάση την οποία έχουν αναπτυχθεί. Επίσης, μπορούν να χρησιμοποιηθούν σε πολλά διαφορετικά περιβάλλοντα χωρίς να υπάρχει ανάγκη για την παραμετροποίηση τους [16],[17],[18].

Τα πλεονεκτήματα της αρχιτεκτονικής αυτής είναι:

- Οι υπηρεσίες που έχουν δημιουργηθεί με βάση την Υπηρεσιοκεντρική Αρχιτεκτονική μπορούν να προσαρμοστούν πιο εύκολα όταν χρειαστεί να χρησιμοποιηθούν με νεότερες τεχνολογίες.

- Τα μικρότερα κομμάτια κώδικα που αποτελούν τις εφαρμογές, μπορούν να δημιουργηθούν, και σε περίπτωση που χρειαστεί να ανανεωθούν, πιο γρήγορα σε αντίθεση με εφαρμογές μονολιθικής αρχιτεκτονικής.
- Ο κώδικας που αναπτύσσεται σύμφωνα με την Υπηρεσιοκεντρική Αρχιτεκτονική μπορεί να χρησιμοποιηθεί σε παραπάνω από μία εφαρμογές, ανεξαρτήτως αντικειμένου, κάτι το οποίο μειώνει τα κόστη για την εκάστοτε επιχείρηση, συμβάλλει στη γρήγορη ανάπτυξη νέων υπηρεσιών και στη γρήγορη εξαγωγή τους στην αγορά.

Η Υπηρεσιοκεντρική Αρχιτεκτονική αποτελείται από τέσσερις συνιστώσες:

- Υπηρεσία: Η υπηρεσία αποτελεί το βασικό δομικό στοιχείο της Υπηρεσιοκεντρικής Αρχιτεκτονικής. Μία υπηρεσία μπορεί να είναι προσβάσιμη από το διαδίκτυο, ή όχι. Έχει τρία βασικά χαρακτηριστικά:
  - Τον κώδικα που την υλοποιεί
  - Τη σύμβαση για τη χρήση της, η οποία περιέχει τους όρους χρήσης και τα κόστη που σχετίζονται με τη χρήση της, αν υπάρχουν
  - Τη διεπαφή, που χρησιμοποιείται στη λειτουργία της υπηρεσίας και τη σύνδεσή της με άλλες υπηρεσίες.
- Ο πάροχος της υπηρεσίας: Ο πάροχος δημιουργεί την υπηρεσία, ορίζει τη χρήση και το κόστος της
- Ο καταναλωτής της υπηρεσίας. Ο καταναλωτής παίρνει την υπηρεσία από τον πάροχο και τη χρησιμοποιεί με βάση τους όρους χρήσης.
- Το μητρώο υπηρεσιών. Το μητρώο περιέχει τις περιγραφές των υπηρεσιών. Σε αυτό μπορεί να ψάξει ο εκάστοτε ενδιαφερόμενος καταναλωτής για να βρει την υπηρεσία που καλύπτει τις ανάγκες του.

Οι υπηρεσίες που έχουν αναπτυχθεί με την Υπηρεσιοκεντρική Αρχιτεκτονική, χρησιμοποιούν πρωτόκολλα επικοινωνίας για να επικοινωνούν μεταξύ τους αλλά και για να μεταφέρουν τα δεδομένα που χρειάζονται για τη λειτουργία τους [16],[17],[18].

Για παράδειγμα, τέτοια πρωτόκολλα είναι:

- SOAP
- Apache Thrift
- Java Message Service (JMS)

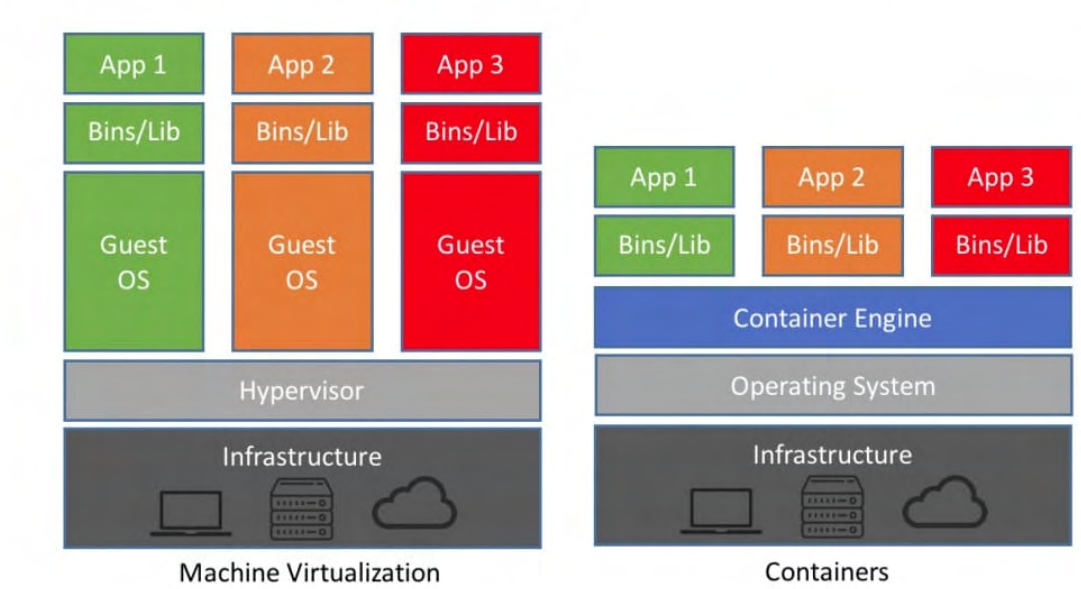
- RESTful API

2.3 Εικονικοποίηση Λειτουργικού Συστήματος

Με την έννοια Εικονικοποίηση Λειτουργικού Συστήματος περιγράφουμε την διαδικασία ανάπτυξης λειτουργικών συστημάτων κατά την οποία σε ένα περιέκτη, περιλαμβάνονται όλες οι συνιστώσες μίας εφαρμογής ή υπηρεσίας καθώς και ο κώδικάς της. Ως περιέκτη ορίζουμε στην νοητή οντότητα που τρέχει πάνω στο εκάστοτε λειτουργικό σύστημα και περιέχει ό,τι είναι απαραίτητο για να λειτουργήσει μία εφαρμογή και προσφέρει την απαραίτητη απομόνωση ώστε αυτή να λειτουργεί χωρίς να επηρεάζει άλλα μέρη του συστήματος [19],[20].

Τα πλεονεκτήματα της εικονικοποίησης είναι:

- Επεκτασιμότητα: Οι πόροι ενός περιέκτη μπορούν να προσαρμόζονται γρήγορα και εύκολα ανάλογα με τις ανάγκες της εφαρμογής.
- Φορητότητα: Οι περιέκτες επιτρέπουν τη μεταφορά μιας εφαρμογής σε διαφορετικό περιβάλλον από αυτό που έχει αναπτυχθεί χωρίς να χρειαστεί κάποια εφαρμογή στον κώδικά της.
- Ευκινησία: Οι αλλαγές στον περιέκτη μπορούν να γίνουν χωρίς να επηρεάζει άλλες εφαρμογές ή το λειτουργικό.
- Ανοχή σε σφάλματα: Σε περίπτωση που υπάρχει υπάρξει κάποιο πρόβλημα στην εφαρμογή αυτό δεν επηρεάζει τις άλλες εφαρμογές ή τους άλλους περιέκτες.



Εικόνα 4 Αρχιτεκτονική εικονικών μηχανών και περιεκτών, πηγή: research.aimultiple.com

2.4 Βασικοί άξονες στην ανάπτυξη ενός project

Η χρήση της Υπηρεσιοκεντρικής Αρχιτεκτονικής και της Αρχιτεκτονικής Μικροϋπηρεσιών λειτουργεί ως μέσο για την επίτευξη της αυτοματοποίησης διεργασιών κατά την ανάπτυξη μίας εφαρμογής ή υπηρεσίας.

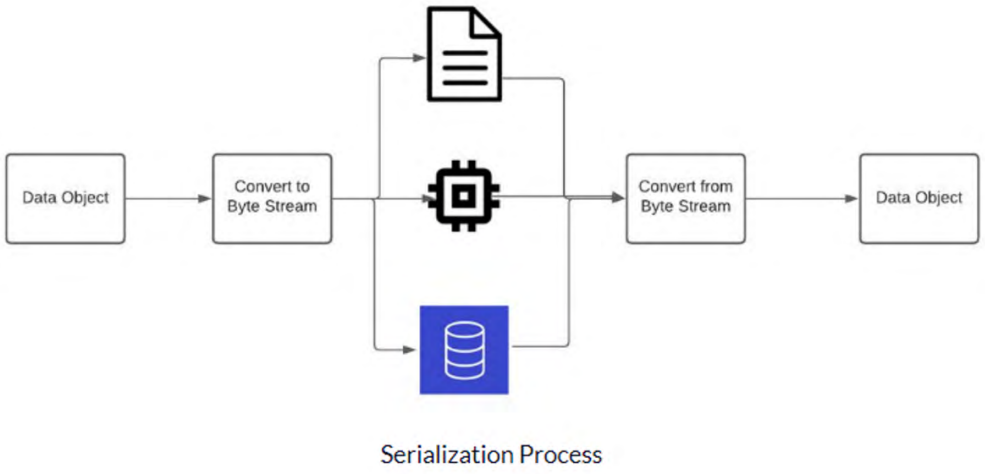
Σύμφωνα με τούς Humble et al. (2010) « Η αυτοματοποίηση διαδικασιών στην ανάπτυξη εφαρμογών αναφέρεται στη χρήση εργαλείων και τεχνικών για την αυτοματοποίηση χειροκίνητων, επαναλαμβανόμενων και ευαίσθητων σε λάθη έργων που σχετίζονται με τη δημιουργία, τη δοκιμή και την ανάπτυξη λογισμικού. Αυτή η αυτοματοποίηση είναι σχεδιασμένη για να μειώσει την πιθανότητα ανθρώπινου λάθους, να αυξήσει τη συνέπεια, και να επιταχύνει τη διαδικασία ανάπτυξης λογισμικού». Η αυτοματοποίηση διαδικασιών προσφέρει πολλά οφέλη στην ανάπτυξη εφαρμογών. Αρχικά, παρατηρείται αυξημένη αποδοτικότητα των εργαζομένων, καθώς μπορούν να κερδίσουν χρόνο για άλλα έργα. Επίσης, λόγω της μείωσης του χρόνου ανάπτυξης, η εκάστοτε εφαρμογή μπορεί να κυκλοφορήσει στην αγορά πολύ πιο γρήγορα. Ακόμη, επιτυγχάνεται και η μείωση των δαπανών, καθώς η μείωση του χρόνου ανάπτυξης εφαρμογών οδηγεί και σε μειωμένα κόστη. Τέλος λόγω της μείωσης της ανθρώπινης παρέμβασης , μειώνεται το πλήθος των λαθών που μπορεί να προκύψουν και επιτυγχάνεται η συνέπεια της παραγωγής των εφαρμογών και η καλύτερη ποιότητα του τελικού αποτελέσματος [21],[22],23],[24],[25],[26].

Η Αρχιτεκτονική Μικροϋπηρεσιών προσφέρει τη δυνατότητα απομόνωσης των υπηρεσιών, επιτρέποντας με αυτόν τον τρόπο την επίτευξη των δυνατοτήτων που προσφέρει η αυτοματοποίηση στο μέγιστο δυνατό βαθμό και ο μηχανισμός ενορχήστρωσης βοηθά στη ροή των εργασιών, συντονίζοντας τις επιμέρους διεργασίες.

Για την ορθή λειτουργία των μικροϋπηρεσιών είναι σημαντική και η σειριοποίηση των δεδομένων. Ως σειριοποίηση δεδομένων ορίζεται « Η διαδικασία μετατροπής της αναπαράστασης ενός αντικειμένου μέσα στη μνήμη σε μία σειρά δυφιοσύλλαβων (byte stream)» (Grus, 1999; Gorelick et al., 2014) [27].

Σε ένα σύστημα που αποτελείται από μικροϋπηρεσίες είναι πολύ σημαντική η δυνατότητα μεταφοράς δεομένων ανάμεσα σε διαφορετικές υπηρεσίες που ζητούν διαφορετικό τύπο δεδομένων ως είσοδο. Λόγω των διαφορετικών γλωσσών προγραμματισμού που χρησιμοποιούνται στην ανάπτυξη των εφαρμογών, απαιτείται η εκάστοτε πληροφορία να είναι αποθηκευμένη με τον κατάλληλο τύπο ώστε να μπορεί να μεταφέρεται από υπηρεσία σε υπηρεσία, χωρίς να είναι η μορφή στην οποία βρίσκεται εμπόδιο για την κατανόησή της. Τέλος, καθώς γίνεται μετατροπή των δεδομένων για τη μεταφορά τους, το μέγεθος της πληροφορίας μειώνεται, με αποτέλεσμα να μειώνονται και οι χρόνοι που απαιτούνται για τη μεταφορά της [28],[29],[30].

Η η σειριοποίηση δεδομένων μπορεί να γίνει με τη χρήση μορφοτύπων όπως XML, JSON, BSON, YAML. Η επιλογή ανάμεσα σε αυτά γίνεται ανάλογα με την πολυπλοκότητα των δεδομένων, την ανάγκη για εύκολη κατανόηση, από ανθρώπου και παραγόντων που επιφέρουν όπως η ταχύτητα και ο αποθηκευτικός χώρος [29],[30].



Εικόνα 5 Διαδικασία Σειριοποίησης, πηγή: https://www.freecodecamp.org/news/what-is-serialization

Για τη συμμόρφωσης ενός έργου με τις δύο αυτές βασικές αρχές χρησιμοποιούνται εργαλεία αυτοματοποίησης. Βασικά παραδείγματα των εργαλείων αυτών αποτελούν το Maven και το Ansible.

Το Maven καθιστά αυτοματοποιημένη τη διαδικασία της δημιουργίας και της δοκιμής της εφαρμογής. Χρησιμοποιεί αποθετήρια από τα οποία αντλεί τις πληροφορίες και τις όποιες μικροϋπηρεσίες χρειάζεται για την κάθε εφαρμογή. Κυρίως επιλέγεται για υπηρεσίες ανεπτυγμένες με τη χρήση της γλώσσας JAVA [31].

Το Ansible αποτελεί άλλο ένα εργαλείο αυτοματοποίησης της διαδικασίας ανάπτυξης και διαχείρισης υπηρεσιών. Με τη χρήση του Ansible απλοποιούνται διαδικασίες όπως η παραμετροποίηση των υπηρεσιών και η εγκατάσταση των απαραίτητων εργαλείων και λογισμικών. Τέλος, δίνει τη δυνατότητα για κλιμακοθέτηση των πόρων σε περίπτωση που αυτό καταστεί αναγκαίο, με τους νέους πόρους να συνεργάζονται απρόσκοπτα και αρμονικά με τους ήδη υπάρχοντες [32].

ΚΕΦΑΛΑΙΟ 3: ΕΡΓΑΛΕΙΑ ΚΑΙ ΜΕΘΟΔΟΙ

3.1 Azure

Το Azure αποτελεί την πλατφόρμα της Microsoft για την παροχή υπηρεσιών υπολογιστικού νέφους.

Το Azure λειτουργεί χάρη στην ύπαρξη κέντρων δεδομένων σε Ευρώπη, Αμερική, Ασία, Αφρική και Ωκεανία.

Εικόνα 6 Χάρτης υποδομών του Azure, πηγή: infrastructuremap.microsoft.com/explore

Η λειτουργία του βασίζεται σε δύο βασικά μοντέλα. Το πρώτο είναι το μοντέλο διαμοιρασμού ευθυνών, στο οποίο με βάση τις ανάγκες του εκάστοτε έργου γίνεται η επιλογή του αντίστοιχου τύπου παροχής υπηρεσιών (IaaS, SaaS, PaaS). Το δεύτερο είναι το μοντέλο με βάση την κατανάλωση, όπου η χρέωση γίνεται με βάση την χρήση των υπηρεσιών [33],[34].

3.2 Maven

Η ανάπτυξη εφαρμογών απαιτεί την εκτέλεση πολλών χρονοβόρων διαδικασιών από την πλευρά των προγραμματιστών. Το Maven σαν εργαλείο έρχεται να λειτουργήσει επικουρικά στη δουλειά τους, προσθέτοντας ένα επίπεδο αυτοματοποίησης στις διαδικασίες ανάπτυξης μειώνοντας ουσιαστικά τους χρόνους.

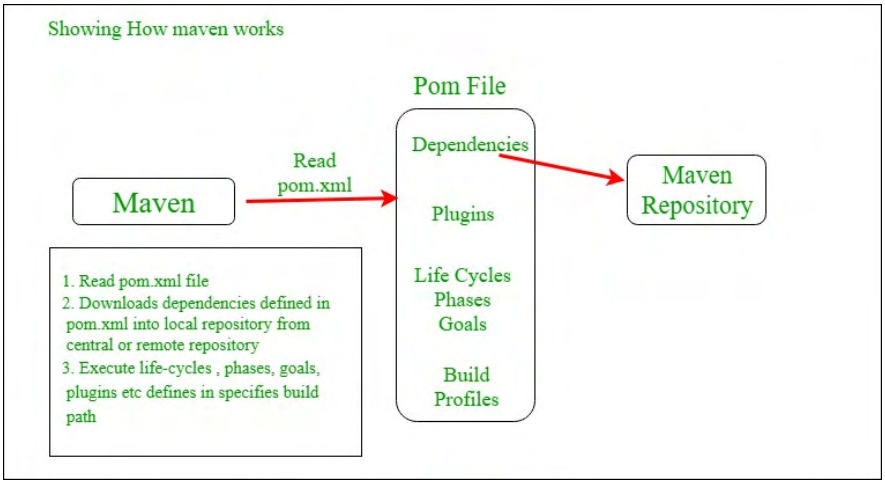
Πιο συγκεκριμένα, το Maven παρέχει απλές και τυποποιημένες μεθόδους για να γίνει η δόμηση μίας εφαρμογής, για την πρόσθεση βιβλιοθηκών σε αυτή, την παραγωγή υπηρεσιοξαρτήσεων (dependencies) από αποθετήρια, κώδικα τα οποία η εφαρμογή αξιοποιεί, τη μεταγλώττιση και τη δοκιμή μίας εφαρμογής καθώς και για πολλές άλλες λειτουργίες.

Για τη λειτουργία του Maven χρησιμοποιούνται αρχεία POM (project Object Model). Πρόκειται για αρχεία τύπου XML τα οποία περιέχουν όλες τις απαραίτητες πληροφορίες για τη δημιουργία και την εκτέλεση της εκάστοτε εφαρμογής. Ένα αρχείο POM μπορεί να περιέχει όλες τις υπηρεσιοξαρτήσεις, τα

αποθετήρια και τις εμβυσματούμενες μονάδες (plugins) που χρειάζεται να αντληθούν ώστε να λειτουργήσει η εφαρμογή [35,[36],[37].

Παρακάτω παρουσιάζονται συνοπτικά τα βήματα του κύκλου ζωής μίας εφαρμογής που έχει αναπτυχθεί με τη χρήση Maven (Maven Build Lifecycle).

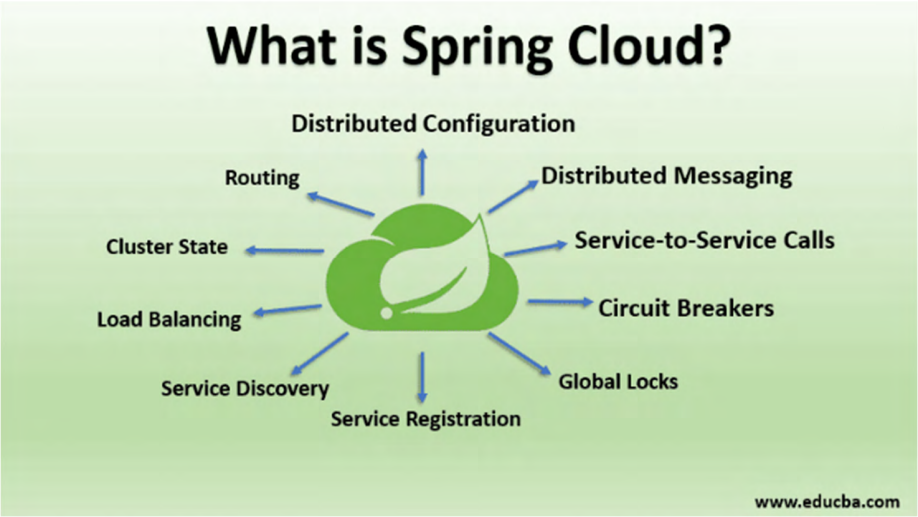
1. Προετοιμασία πηγών, αντιγραφή των απαραίτητων στοιχείων από τις πηγές
2. Επικαιροποίηση των πληροφοριών του έργου
3. Μεταγλώττιση του κώδικα
4. Δοκιμές του κώδικα
5. Εγκατάσταση των αναγκαίων συνιστωσών από τα αποθετήρια
6. Εκτέλεση της εφαρμογής



Εικόνα 7 Maven Build Lifecycle, πηγή: geeksforgeeks.org

Με Maven έχει δημιουργηθεί και η υπηρεσία που θα παρουσιαστεί στη συνέχεια.

3.3 Spring Cloud



Εικόνα 8 Spring Cloud, πηγή: educba.com

Το Spring Cloud είναι μία ανοιχτή βιβλιοθήκη που παρέχει εργαλεία για την ανάπτυξη εφαρμογών. Τα εργαλεία αυτά υλοποιούν διαδικασίες που είναι συνήθεις και απαραίτητες για μία εφαρμογή όπως η κατανεμημένη διάρθρωση (distributed configuration), η καταχώρηση υπηρεσιών (service registration), η δρομολόγηση της κίνησης, η κατανεμημένη επικοινωνία, κ.ά.[38],[39],[41],[41].

3.4 Services της εφαρμογής

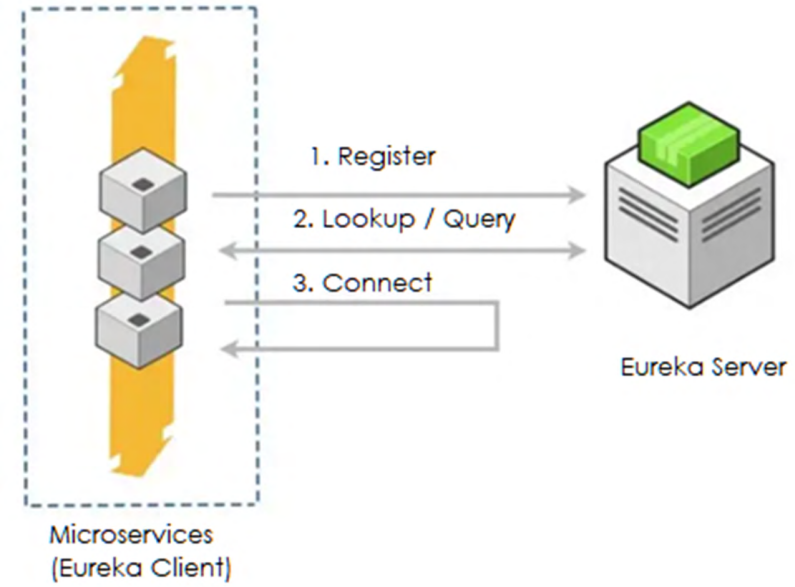
Configuration service

Το Configuration service της εφαρμογής υλοποιείται με τη χρήση του Spring Cloud Config. Πρόκειται για μία υπηρεσία που διαθέτει client-side και server-side υλοποίηση. Μέσω αυτής συγκεντρώνονται σε ένα μέρος όλες οι επιμέρους ρυθμίσεις που αφορούν τις μικροϋπηρεσίες της εκάστοτε εφαρμογής, γεγονός που βοηθάει να γίνει η ανάπτυξη της εφαρμογής εύκολα και σε οποιοδήποτε περιβάλλον, ανεξαρτήτως γλώσσας [42],[43].

Eureka Discovery Service

Πρόκειται για μία μικροϋπηρεσία που υλοποιείται με τη χρήση του Eureka Service, του Spring Cloud. Το Eureka είναι μία REST υπηρεσία αναγνώρισης των μικροϋπηρεσιών που αποτελούν μια εφαρμογή. Εκεί καταγράφεται κάθε μικροϋπηρεσία δίνοντας τα βασικά της στοιχεία όπως πόρτα, IP διεύθυνση, όνομα και κατάσταση με σκοπό οι άλλες μικροϋπηρεσίες να μπορούν να την ανακαλύψουν και να επικοινωνήσουν με αυτή. Κάθε επιμέρους μικροϋπηρεσία, μπορεί να πάρει από το Server μία λίστα που περιλαμβάνει τις υπόλοιπες μικροϋπηρεσίες και να τη χρησιμοποιήσει για να πετύχει τις

επιθυμητές επικοινωνίες. Το Eureka Service αποτελείται και αυτό από δύο μέρη, με το Server να είναι το μέρος που καταγράφονται οι μικροϋπηρεσίες, και το Client η κάθε μικροϋπηρεσία που καταγράφεται. Για να γίνει η καταγραφή μίας μικροϋπηρεσίας, αφού αυτή έχει ξεκινήσει, αποστέλλει τα στοιχεία της στο Eureka Server, και αφού ολοκληρωθεί η διαδικασία αποστέλλει heartbeat σήμα ώστε να παραμένει εγγεγραμμένη. Σε περίπτωση που δε σταλεί το σήμα αυτό, διαγράφεται από τη λίστα [44],[45],[46].



Εικόνα 9 Διαδικασία καταγραφής και αναζήτησης μικροϋπηρεσίας, πηγή: https://medium.com

User Service

Στο User Service βρίσκεται η βάση δεδομένων που έχει αποθηκευμένα τα στοιχεία των χρηστών που χρησιμοποιούνται στην υπηρεσία. Η αποθήκευση γίνεται σε βάση δεδομένων MySQL.

Η MySQL είναι ένα σύστημα διαχείρισης σχεσιακών βάσεων (Relational Database Management System – RDBMS). Για την υλοποίηση των βάσεων χρησιμοποιείται το μοντέλο πελάτη-εξυπηρετητή. Με τη χρήση εντολών-ερωτημάτων (queries) γίνεται η διαχείριση των δεδομένων και η οριοθέτηση του ποιος έχει πρόσβαση σε αυτά [47],[48].

Movie Service

Στο Movie βρίσκεται η βάση δεδομένων που έχει αποθηκευμένες τις ταινίες και τις πληροφορίες σχετικά με αυτές, που χρησιμοποιούνται στην υπηρεσία. Η αποθήκευση γίνεται σε βάσεις δεδομένων MongoDB.

Η MongoDB είναι μία βάση δεδομένων ανοιχτού κώδικα που χρησιμοποιεί αρχεία για να αποθηκεύσει τις πληροφορίες. Η χρήση της προτιμάται σε σχέση με τη χρήση σχεσιακών βάσεων δεδομένων, όπως η MySQL, όταν έχουμε εφαρμογές των οποίων τα δεδομένα αλλάζουν πολύ γρήγορα και δε θέλουμε να υπάρχουν μεγάλες καθυστερήσεις στην ανανέωση της βάσης και την ανάκτηση των δεδομένων από αυτή [49],[50].

Recommendation Service

Στο Recommendation Service υλοποιείται η διαδικασία της πρότασης ταινιών. Στη βάση της μικροϋπηρεσίας αυτής αποθηκεύονται βασικές πληροφορίες για τις ταινίες και τους χρήστες και όταν τρέχει, δείχνει προτιμήσεις για ένα συγκεκριμένο χρήστη μετά από επιλογή αυτού.

Η βάση σε αυτή τη μικροϋπηρεσία υλοποιείται με τη χρήση του εργαλείου Neo4j. Το Neo4j είναι μία βάση που αναπαριστά τα δεδομένα και τις διασυνδέσεις μεταξύ τους με γραφικό τρόπο. Τα δεδομένα είναι αποθηκευμένα ως αντικείμενα και αποτελούν τους κόμβους των γραφημάτων, ενώ οι σχέσεις ανάμεσά τους τις ακμές. Οι βάσεις τέτοιου είδους προσφέρουν ευελιξία, καθώς μπορούν να προσαρμοστούν εύκολα ανάλογα με την εκάστοτε εφαρμογή, και ευκολότερη ανάκτηση των δεδομένων [51],[52].

Recommendation Client

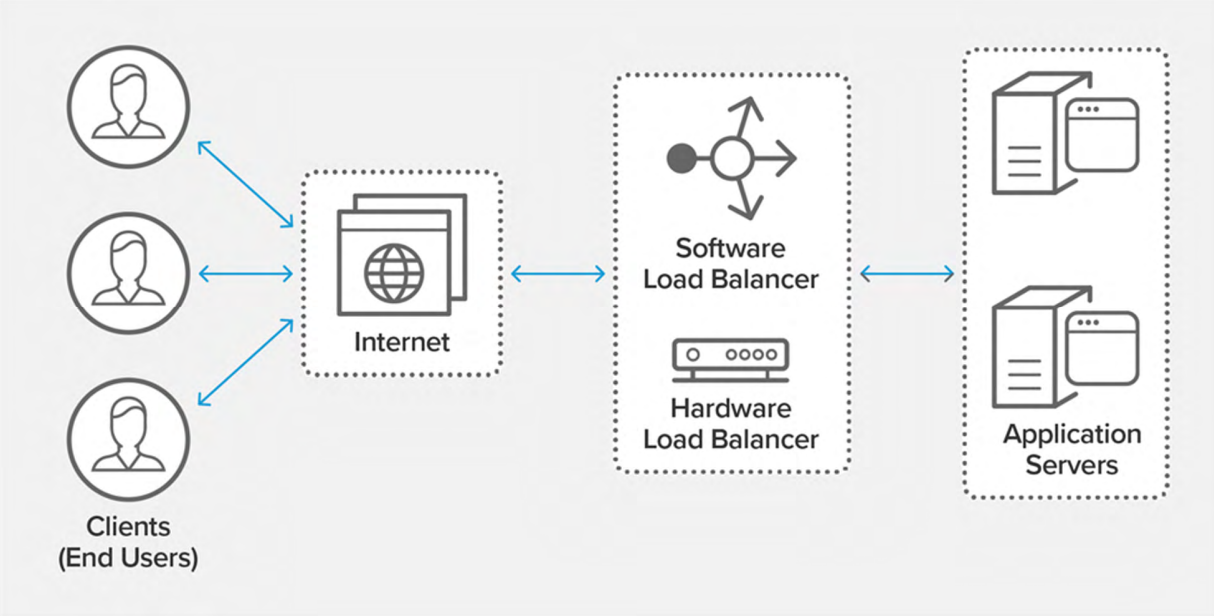
Η τελευταία μικροϋπηρεσία της εφαρμογής είναι το Recommendation Client. Ο ρόλος του είναι κυρίως αυτός του API gateway, δηλαδή του εργαλείου που διαχειρίζεται τα αιτήματα προς στις μικροϋπηρεσίες και βρίσκεται ανάμεσα στο backend και το client μέρος της εφαρμογής. Η μικροϋπηρεσία αυτή χρησιμοποιεί ένα σύνολο εργαλείων για να επιτελέσει διαφορετικές λειτουργίες.

Χρησιμοποιεί ένα circuit breaker – αυτόματο διακόπτη, συγκεκριμένα το Hystrix. Το Hystrix είναι μία βιβλιοθήκη, που απασχολείται με την διαχείριση βλαβών σε μία υπηρεσία. Σε περίπτωση βλάβης, ενεργοποιεί μία προκαθορισμένη διαδικασία. Στην περίπτωση της εφαρμογής πρότασης ταινιών, το Hystrix αν κάποια υπηρεσία παρουσιάζει σφάλμα, δίνει στο χρήστη πέντε συγκεκριμένες ταινίες ως πρόταση. Αυτή είναι η προκαθορισμένη διαδικασία που έχει επιλέξει ο προγραμματιστής [53],[54],[55].

Σε συνεργασία με το Hystrix , χρησιμοποιείται το Zuul Proxy. Πρόκειται για ένα proxy server - διακομιστής μεσολάβησης που επιτρέπει την εξυπηρέτηση hosts, άσχετα με την τεχνολογία που χρησιμοποιούν. Ειδικά, με το Hystrix, το Zuul προσφέρει μεγαλύτερη ανοχή στα σφάλματα [56],[57].

Επίσης χρησιμοποιείται το Feign Client. Πρόκειται για ένα web service client που βοηθάει στη δημιουργία διεπαφών. Το Feign ενσωματώνει ένα εξισορροπητή φορτίου - load balancer, ώστε να διαχειρίζεται τον αριθμό αιτημάτων που παίρνει και να τα εξυπηρετεί χωρίς καθυστέρηση [58],[59].

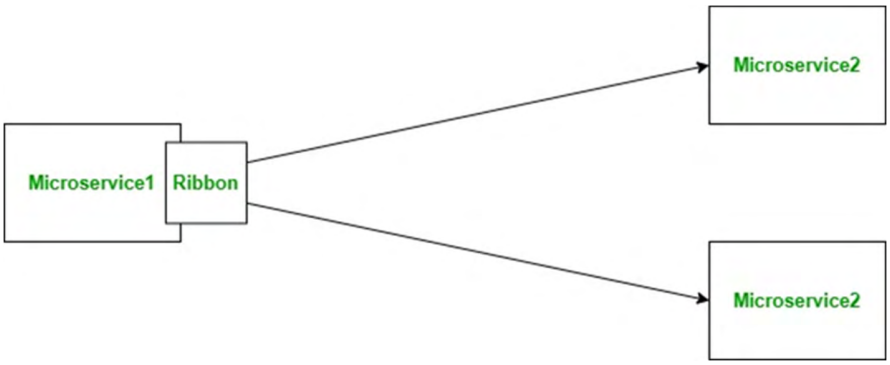
Η εξισορρόπηση φορτίου είναι η διαδικασία διαμοιρασμού της δικτυακής κίνησης ανάμεσα σε δύο πόρους. Καθώς παρουσιάζεται ανάγκη για ταυτόχρονη εξυπηρέτηση πολλών χρηστών η εξισορρόπηση φορτίου επιλύει αυτό το ζήτημα προσφέροντας την εκάστοτε υπηρεσία με μεγάλο ποσοστό διαθεσιμότητας. Η υλοποίηση της γίνεται με τη χρήση κάποιας συσκευής ή υπηρεσίας. Ένας εξισοροπητής φορτίου έχει επίσης τη δυνατότητα να παρακολουθεί την κατάσταση της υγείας ενός πόρου, ώστε σε περίπτωση που δεν είναι διαθέσιμος, ώστε να ανακατευθύνει τα αιτήματα των χρηστών στον λειτουργικό πόρο και να μην υπάρχει διακοπή της υπηρεσίας [60,[61],[62],[63].



Εικόνα 10 Παράδειγμα υποδομής που χρησιμοποιεί εξισορρόπηση φορτίου, πηγή: nginx.com

Ο εξισορροπητής φορτίου που χρησιμοποιείται στη συγκεκριμένη περίπτωση είναι το Ribbon. Το Ribbon είναι ένα client side load balancer, δηλαδή χρησιμοποιεί αλγορίθμους ώστε να επιλέξει τα στιγμιότυπα υπηρεσιών και τον εκάστοτε εξυπηρετητή που θα εξυπηρετήσουν το κάθε αίτημα. Συνήθως η επιλογή γίνεται με τους ήδη υπάρχοντες κανόνες που έχει ρυθμιστεί το Ribbon, υπάρχει όμως δυνατότητα να δημιουργηθούν νέοι κανόνες, ανάλογα με τις ανάγκες της κάθε εφαρμογής.

Επιπλέον, σε περίπτωση που κάποιος εξυπηρετητής δεν είναι λειτουργικός, το Ribbon τον αγνοεί στη διαδικασία της επιλογής [64],[65].



Εικόνα 11 Παράδειγμα λειτουργίας Ribbon, πηγή: studytonight.com

POSTMAN

Το Postman είναι μία REST API πλατφόρμα που χρησιμοποιείται κυρίως για περιβάλλοντα δοκιμών.

Ως API (Application Programming Interface – Διεπαφή Προγραμματισμού Εφαρμογών) ορίζουμε το μέσο το οποίο υλοποιεί την επικοινωνία δεδομένων ανάμεσα στο χρήστη και στην υπηρεσία που χρησιμοποιεί. Ένα API αποτελείται από διάφορα πρωτόκολλα τα οποία ρυθμίζουν τη λειτουργία του αλλά προσφέρουν και την ασφάλεια που χρειάζεται η εκάστοτε υπηρεσία μέσω της χρήσης ελέγχων και την αυθεντικοποίησης των χρηστών.

Ιδιάιτερα τα API είναι γνωστά για τη χρήση τους σε επικοινωνίες ανάμεσα σε χρήστες και web services. Συγκεκριμένα, το REST API, ή RESTful API, υλοποιεί αιτήματα πρόσβασης προς βάσεις για να παρουσιάσει δεδομένα στο χρήστη.

Τα αιτήματα προς τη βάση αφορούν τέσσερις λειτουργίες:

- GET: για την ανάκτηση εγγραφών
- POST: για την δημιουργία μίας εγγραφής
- PUT για την ανανέωση μίας εγγραφής
- DELETE: για τη διαγραφή μιας εγγραφής

Πιο συγκεκριμένα, μέσω του Postman μπορούν να γίνουν σε μία υπηρεσία κλήσεις τύπου HTTP, όπως GET και POST και να εμφανίσει τα αντίστοιχα αποτελέσματα στο περιβάλλον του [66],[67],[68],[69].

Ένα παράδειγμα της γενικής κλήσης GET του τύπου HTTP είναι το παρακάτω:

```
GET / HTTP/1.1
Host: reqbin.com
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/79.0.3945.88 Safari/537.36
Upgrade-Insecure-Requests: 1
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/
signed-exchange;v=b3;q=0.9
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate
```

Ένα παράδειγμα της γενικής κλήσης POST του τύπου HTTP είναι το παρακάτω:

```
POST /echo/post/json HTTP/1.1
Authorization: Bearer mt0dgHmLJMVQhvjpNXDyA83vA_Pxh33Y
Accept: application/json
Content-Type: application/json
Content-Length: 85
Host: reqbin.com
{
  "Id": 12345,
  "Customer": "John Smith",
  "Quantity": 1,
  "Price": 10.00
}
```

3.5 Docker

Το Docker είναι μία πλατφόρμα που δίνει τη δυνατότητα ανάπτυξης και εκτέλεσης εφαρμογών μέσα σε μονάδες που ονομάζονται περιέκτες (containers).

Για να επιτευχθεί η ομαλή λειτουργία και συνεργασία των περιεκτών, το Docker χρησιμοποιεί αρχιτεκτονική τύπου Client-Server. Ο Docker Client, επικοινωνεί με το Docker Daemon. Ως Docker Daemon αναφέρουμε την υπηρεσία που διαχειρίζεται και υλοποιεί την «ενορχήστρωση» όλων των Docker Images. Η έννοια του Docker Image θα αναλυθεί παρακάτω. Ο Client και ο Daemon μπορούν είτε να βρίσκονται στο ίδιο σύστημα είτε σε διαφορετικά.

Για την ενσωμάτωση του Docker και την εκμετάλλευση των πλεονεκτημάτων που προσφέρει η εικονικοποίηση λειτουργικού συστήματος σε μία υπηρεσία είναι απαραίτητη η υλοποίηση κάποιων εργαλείων.

Ως Docker Image περιγράφουμε τη δομή που περιέχει τον κώδικα για τη δημιουργία του Docker Container – Περιέκτη. Τα Docker Images έχουν επίπεδα, τα οποία δημιουργούνται μετά από κάθε εντολή του dockerfile. Η έννοια του dockerfile θα αναλυθεί παρακάτω. Κάθε αλλαγή στο image δημιουργεί ένα νέο επίπεδο και το επίπεδο πριν από αυτό αποθηκεύεται για περιπτώσεις βλαβών ή για χρήση σε άλλες υλοποιήσεις.

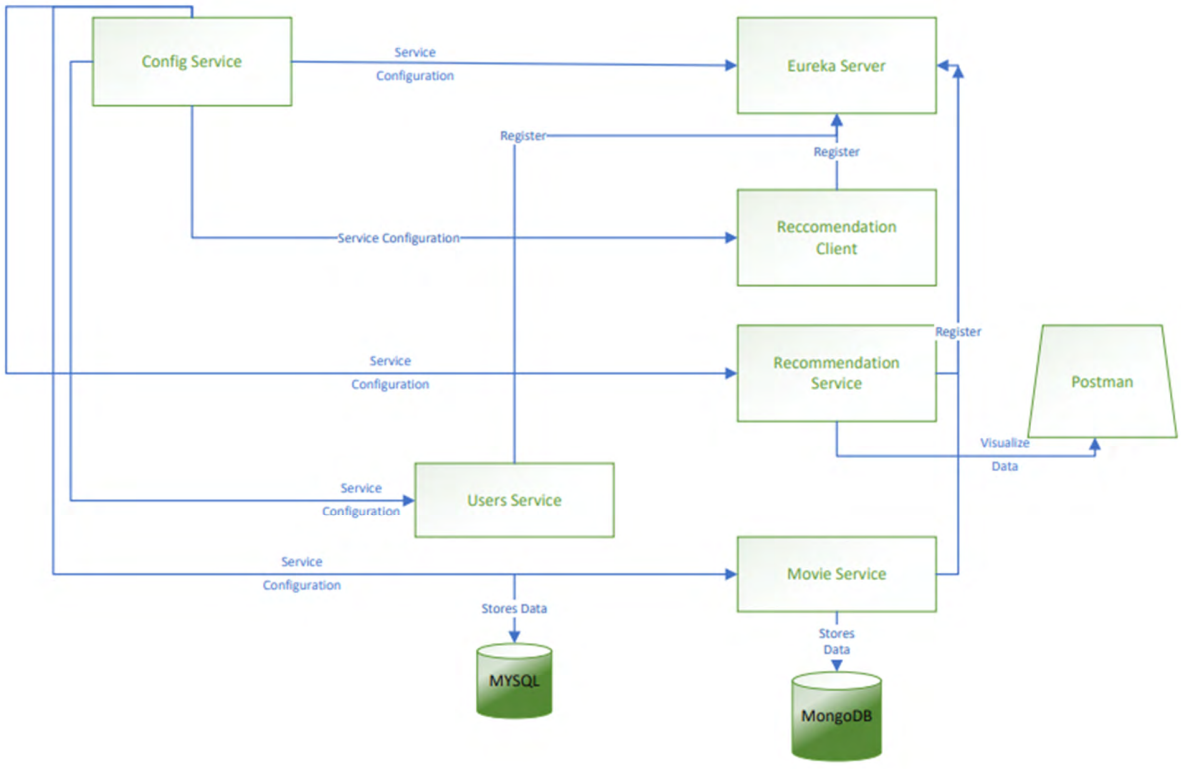
Το dockerfile είναι ένα αρχείο το οποίο περιέχει τις βασικές οδηγίες για τη δημιουργία του Docker Image. Αποτελείται από εντολές που θα χρησιμοποιούνταν συνήθως στη γραμμή εντολών και εκτελούνται από το Docker Engine.

Τέλος, από το Docker Image δημιουργούνται και εκτελούνται τα Docker Containers. Τα Containers είναι αυτά τα οποία εκτελεί και αλληλοεπιδρά ο χρήστης. Τα Containers μεταξύ τους επικοινωνούν μέσω της δικτύωσης που δημιουργεί το Docker.

Το Docker Engine είναι η τεχνολογία πάνω στην οποία βασίζεται όλη η πλατφόρμα του Docker. Χρησιμοποιεί το μοντέλο πελάτη-εξυπηρετητή για να εγκαθιδρύσει την επικοινωνία, τόσο ανάμεσα στο χρήστη και στους περιέκτες, όσο και μεταξύ των περιεκτών [70],[71],[72],[73],[74].

ΚΕΦΑΛΑΙΟ 4: ΠΡΑΚΤΙΚΗ ΠΡΟΣΕΓΓΙΣΗ

Με σκοπό την περαιτέρω διερεύνηση των παραπάνω τεχνολογιών, στα πλαίσια αυτής της διπλωματικής εργασίας, υλοποιήθηκε μία υπηρεσία πρότασης ταινιών, σχεδιασμένη και υλοποιημένη με αρχιτεκτονική μικροϋπηρεσιών.Η ιδέα και η βασική υλοποίηση της υπηρεσίας αυτής προέρχεται από το αποθετήριο https://github.com/mdeket/spring-cloud-movie-recommendation.



Εικόνα 12 High Level Design εφαρμογής αξιολογήσεων ταινιών με χρήση Visio

Πρόκειται για μία απλή υπηρεσία, η οποία διατηρεί βάσεις με ταινίες, χρήστες και αξιολογήσεις ταινιών, οι οποίες έχουν ήδη μέσα δεδομένα, και δίνει στον εκάστοτε χρήστη προτάσεις για ταινίες. Για την υλοποίηση αυτής της υπηρεσίας και των επιμέρους λειτουργιών της, έχουν χρησιμοποιηθεί οι μικροϋπηρεσίες που προσφέρονται από το Spring Cloud.

Παρακάτω θα αναλυθεί ο κώδικας της εφαρμογής.

Configuration service

Παρακάτω ακολουθεί ο κώδικας που έχει χρησιμοποιηθεί σε αυτή την εφαρμογή για την υλοποίηση του Spring Cloud Config:

```
package com.example;

//χρήση των repositories του config server και του Spring Cloud
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@EnableConfigServer
@SpringBootApplication
public class ConfigServiceApplication {

	public static void main(String[] args) {
		SpringApplication.run(ConfigServiceApplication.class, args);
	}
}
```

Η υπηρεσία αυτή τρέχει στην πόρτα 8888. Μετά την εκτέλεσή της, θα ανατρέξει σε μία βιβλιοθήκη με YAML αρχεία, στα οποία θα βρει τις απαραίτητες ρυθμίσεις για κάθε μία από τις επιμέρους εφαρμογές. Η βιβλιοθήκη αυτή βρίσκεται στο ακόλουθο αποθετήριο: https://github.com/mdeket/spring-cloud-example-config-repo

Eureka Discovery Service

Ο κώδικας που υλοποιεί το Eureka Server στην εφαρμογή είναι ο εξής:

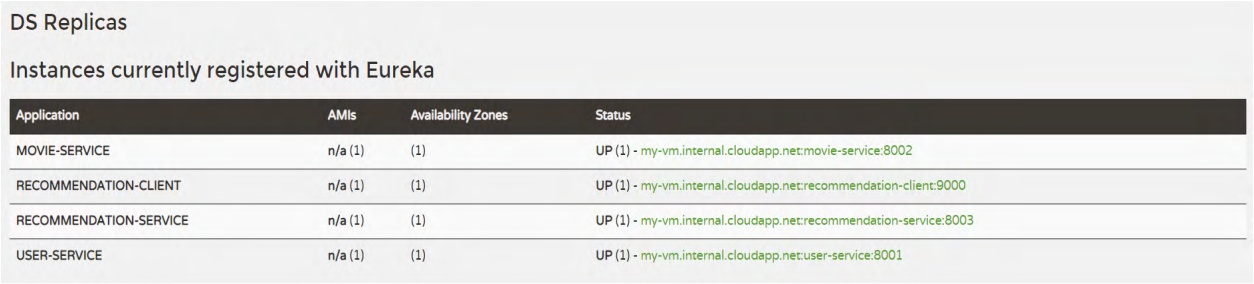
```
package com.example;

//χρήση των repositories του Eureka Server και του Spring Cloud
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@EnableEurekaServer
@SpringBootApplication
public class EurekaServiceApplication {

	public static void main(String[] args) {
		SpringApplication.run(EurekaServiceApplication.class, args);
	}
}
```

Οι ρυθμίσεις του καταγράφονται στο eureka-service-default.yml:

```
server:
  port: 8761 //πόρτα στην οποία τρέχει το Eureka Server
eureka:
  instance:
    hostname: localhost
```

```
client:  //Δημιουργούμε Eureka Server δε μπορεί να κάνει register με τον εαυτό του
  registerWithEureka: false
  fetchRegistry: false
```

Οι υπηρεσίες που έχουν εγγραφεί στο Eureka εμφανίζονται στο Eureka Dashboard. Όσον αφορά τη συγκεκριμένη εφαρμογή, η εικόνα που παίρνουμε μετά από την εγγραφή όλων των υπηρεσιών είναι η εξής:


DS Replicas

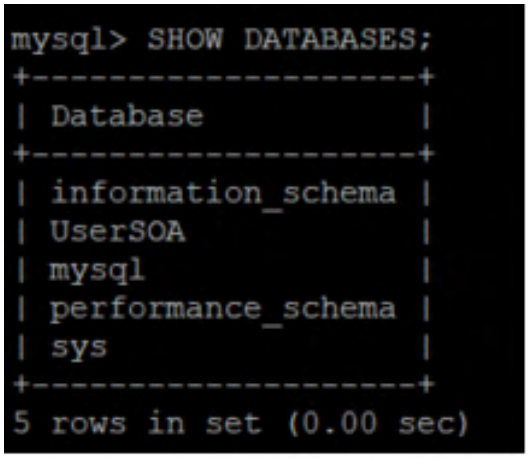
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
MOVIE-SERVICE	n/a (1)	(1)	UP (1) - my-vm.internal.cloudapp.net:movie-service:8002
RECOMMENDATION-CLIENT	n/a (1)	(1)	UP (1) - my-vm.internal.cloudapp.net:recommendation-client:9000
RECOMMENDATION-SERVICE	n/a (1)	(1)	UP (1) - my-vm.internal.cloudapp.net:recommendation-service:8003
USER-SERVICE	n/a (1)	(1)	UP (1) - my-vm.internal.cloudapp.net:user-service:8001

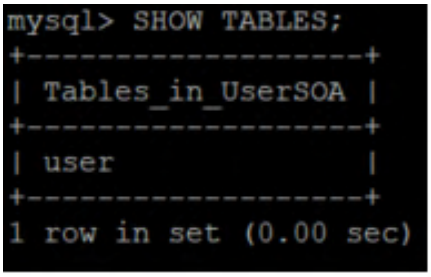
Εικόνα 13 Eureka Dashboard

User Service

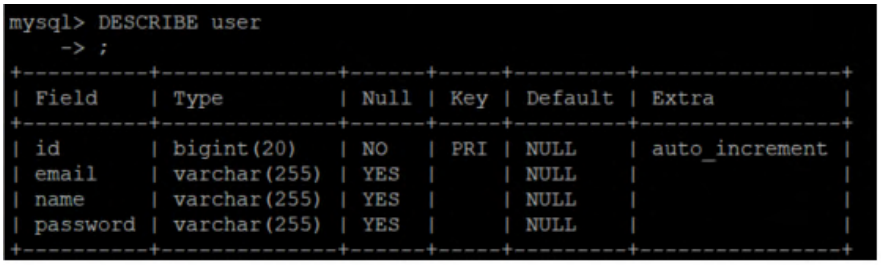
Πρόκειται για το κομμάτι της υπηρεσίας στο οποίο βρίσκονται τα δεδομένα των χρηστών. Τα δεδομένα αυτά αποθηκεύονται σε μία βάση MySQL με όνομα UserSOA.Η βάση αυτή περιέχει ένα πίνακα user, με τα στοιχεία των χρηστών, id (το οποίο είναι και το primary key),email, name, password. Όταν τρέχει η υπηρεσία αυτή, τρέχει και ένα script το οποίο τοποθετεί μέσα σε αυτή τη βάση 50 χρήστες.



Εικόνα 14 Βάση Δεδομένων χρηστών UserSOA



Εικόνα 15 Πίνακας user της βάσης UserSOA



Εικόνα 16 Πεδία πίνακα user

Οι ρυθμίσεις του User Service περιλαμβάνονται στο αρχείο user-service-default.yml:

```
server:
  port: 8001 //πόρτα στην οποία τρέχει το Eureka Server

spring:
  profiles: default
  datasource:
    url: jdbc:mysql://localhost:3306/UserSOA
    username: root //username και password
    password: root
    useSSL: false
    driver-class-name: com.mysql.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update
      show-sql: true
      useSSL: false
      naming-strategy: org.hibernate.cfg.ImprovedNamingStrategy
      dialect: org.hibernate.dialect.MySQL5Dialect
```

Παρακάτω βρίσκεται ο κώδικας που υλοποιεί τη συγκεκριμένη υπηρεσία:

User.java

```
package com.example;
```

```
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import org.hibernate.validator.constraints.Email;

@Entity
public class User implements Serializable {

   @Id
   @GeneratedValue
   private Long Id;

   @Email
   private String email;  //email του χρήστη

   private String password; //password  χρήστη
   private String name; //όνομα χρήστη

   public User() {
   }

   public Long getId() {
      return Id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

}
```

UserController.java

```
package com.example;

import java.util.Collection;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
```

```
@RequestMapping("/user")
public class UserController {

   @Autowired
   private UserRepo customerRepo;

   @RequestMapping(method = RequestMethod.GET)
   public Collection getAllUsers(){ //κλήση της λίστας χρηστών από το repository
      return this.customerRepo.findAll();
   }

   @RequestMapping(value="/{userId}", method = RequestMethod.GET)
   public User getUsers(@PathVariable(name = "userId") Long userId){
      return this.customerRepo.findOne(userId);
   }


}
```

UserRepo.java

```
package com.example;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepo extends JpaRepository<User, Long>{

}
```

UserServiceApplication.java

```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
//εκτέλεση της υπηρεσίας και καταγραφή της στο Eureka
@EnableDiscoveryClient
@SpringBootApplication
public class UserServiceApplication {

	public static void main(String[] args) {
		SpringApplication.run(UserServiceApplication.class, args);
	}
}
```

Movie Service

Όπως το User Service, το Movie Service υλοποιεί τη βάση που περιέχει τις ταινίες και τα στοιχεία τους.

Παρακάτω βρίσκεται ο κώδικας που υλοποιεί τη συγκεκριμένη υπηρεσία:

MongoConfig.java

```
package com.example;

import com.mongodb.Mongo;
import com.mongodb.MongoClient;
import org.springframework.data.mongodb.config.AbstractMongoConfiguration;
import org.springframework.data.mongodb.repository.config.EnableMongoRepositories;

@EnableMongoRepositories(basePackages = "com.example")
public class MongoConfig extends AbstractMongoConfiguration{

   @Override
   protected String getDatabaseName() { //καταγραφή των ταινιών στη βάση MongoDB
      return "movie";
   }

   @Override
   public Mongo mongo() throws Exception {
      return new MongoClient();
   }

}
```

Movie.java

```
 package com.example;

import java.util.Date;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Document;

@Document
public class Movie {
  //πώς είναι καταγεγραμμένη μία ταινία στη βάση δεδομένων
   @Id
   private String id;

   @Indexed
   private Long movieId;

   @Indexed
   private String name;

   @CreatedDate
   private Date createdDate;
```

```
    public Movie() {
    }

    public MovieDTO toDTO(){
        return new MovieDTO(this);
    }

    public Movie(String name, Long movieId) {
        this.name = name;
        this.createdDate = new Date();
        this.movieId = movieId;
    }

    public Long getMovieId() {
        return movieId;
    }

    public void setMovieId(Long movieId) {
        this.movieId = movieId;
    }

    public Date getCreatedDate() {
        return createdDate;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getId() {
        return id;
    }

    public void setCreatedDate(Date createdDate) {
        this.createdDate = createdDate;
    }

}
```

MovieController.java

```
package com.example;

import com.google.gson.Gson;
```

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.stream.Collectors;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
// κλάση για την εμφάνιση όλων των ταινιών στο χρήστη
@RestController
@RequestMapping("/movie")
public class MovieController {

   @Autowired
   private MovieRepo;

   @RequestMapping(method = RequestMethod.GET)
   public Collection<MovieDTO> getAllMovies(){
      return this.movieRepo.findAll()
            .stream()
            .map(Movie::toDTO)
            .collect(Collectors.toList());
   }

   @RequestMapping(value = "/list", method = RequestMethod.GET)
   public Collection<MovieDTO> getListMovies(@RequestParam String ids){
      System.out.println("Somebody found me!");
      Gson = new Gson();
      List<Double> idsList = (ArrayList<Double>)gson.fromJson(ids, ArrayList.class);
      List<MovieDTO> retVal = new ArrayList();
      idsList.stream()
            .forEach(id -> {
               MovieDTO dto = new MovieDTO(this.movieRepo.findByMovieId(id.longValue()));
               retVal.add(dto);
            });
      return retVal;
   }

   @RequestMapping(method = RequestMethod.GET, value = "/{movieId}")
   public Movie getMovie(@PathVariable(value = "movieId") Long movieId){
      return this.movieRepo.findByMovieId(movieId);
   }

   @RequestMapping(method = RequestMethod.GET, value = "/dummyData")
   public void insertDummyData(){
      this.movieRepo.deleteAll();
```

```
        this.movieRepo.save(new Movie("The Shawshank Redemption", 1l));
        this.movieRepo.save(new Movie("The Godfather", 2l));
        this.movieRepo.save(new Movie("The Dark Knight", 3l));
        this.movieRepo.save(new Movie("12 Angry Men", 4l));
        this.movieRepo.save(new Movie("Schindler's List", 5l));
        this.movieRepo.save(new Movie("Pulp Fiction", 6l));
        this.movieRepo.save(new Movie("The Lord of the Rings: The Return of the King", 7l));
        this.movieRepo.save(new Movie("The Good, the Bad and the Ugly", 8l));
        this.movieRepo.save(new Movie("Fight Club", 9l));
        this.movieRepo.save(new Movie("The Lord of the Rings: The Fellowship of the Ring", 10l));
        this.movieRepo.save(new Movie("Star Wars: Episode V - The Empire Strikes Back", 11l));
        this.movieRepo.save(new Movie("Forrest Gump", 12l));
        this.movieRepo.save(new Movie("The Lord of the Rings: The Two Towers", 13l));
        this.movieRepo.save(new Movie("One Flew Over the Cuckoo's Nest", 14l));
        this.movieRepo.save(new Movie("Goodfellas", 15l));
        this.movieRepo.save(new Movie("The Matrix", 16l));
        this.movieRepo.save(new Movie("Goodfellas", 15l));
        this.movieRepo.save(new Movie("The Matrix", 16l));
        this.movieRepo.save(new Movie("Rocky", 17l));
        this.movieRepo.save(new Movie("Old boy", 18l));
    }

}
```

MovieDTO.java

```
package com.example;

public class MovieDTO {

    private Long movieId;
    private String name;

    public MovieDTO() {
    }

    public MovieDTO(Movie movie){
        this.movieId = movie.getMovieId();
        this.name = movie.getName();
    }

    public Long getMovieId() {
        return movieId;
    }

    public String getName() {
        return name;
    }
```

```
    public void setMovieId(Long movieId) {
        this.movieId = movieId;
    }

    public void setName(String name) {
        this.name = name;
    }

}
```

MovieRepo.java

```
package com.example;

import org.springframework.data.mongodb.repository.MongoRepository;

public interface MovieRepo extends MongoRepository<Movie, String>{
    public Movie findByMovieId(Long movieId);
}
```

MovieServiceApplication.java

```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@EnableEurekaClient
@SpringBootApplication
public class MovieServiceApplication{

        public static void main(String[] args) {
                SpringApplication.run(MovieServiceApplication.class, args);
        }
```

Οι ρυθμίσεις της υπηρεσίας καταγράφονται στο movie-service-default.yml:

```
server:
  port: 8002
```

Recommendation Service

Στο Recommendation Service υλοποιείται η διαδικασία της πρότασης ταινιών. Στη βάση της μικροϋπηρεσίας αυτής αποθηκεύονται βασικές πληροφορίες για τις ταινίες και τους χρήστες και όταν τρέχει, δείχνει προτιμήσεις για ένα συγκεκριμένο χρήστη μετά από επιλογή αυτού.

Το Recommendation Service υλοποιείται με τον παρακάτω κώδικα:

RecommendationController.java

```
package com.example;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.Random;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
// κλάση για την εμφάνιση ταινιών ανάλογα με το χρήστη για τον οποίο γίνεται η αναζήτηση
@RestController
@RequestMapping("/recommendation")
public class RecommendationController {


   @Autowired
   private UserRepo userRepo;

   @Autowired
   private MovieRepo movieRepo;

   @RequestMapping(value = "/user", method = RequestMethod.GET)
   public Collection<User> getAllUsers(){
      List<User> users = new ArrayList<>();
      this.userRepo.findAll().forEach(u -> {
         users.add(u);
      });
      return users;
   }

   @RequestMapping(value = "/movie", method = RequestMethod.GET)
   public Collection<Movie> getAllMovies(){
      List<Movie> movies = new ArrayList<>();
      this.movieRepo.findAll().forEach(m -> {
         movies.add(m);
      });
      return movies;
   }

   @RequestMapping(value = "/user/{userId}", method = RequestMethod.GET)
   public User getUser(@PathVariable(name = "userId") Long userId){
      return this.userRepo.findOne(userId);
   }
```

```
  @RequestMapping(value = "/recommend/user/{userId}", method = RequestMethod.GET)
  public Collection<RecommendationData> getRecommendationForUser(@PathVariable(name =
"userId") Long userId){
    Collection<RecommendationData> data = this.movieRepo.getRecommendationForUser(userId);

    // Οι 5 καλύτερες ταινίες εμφανίζονται αν ο χρήστης προς αναζήτηση δεν υπάρχει
    if(data == null || data.isEmpty()){
      data = this.movieRepo.getTopFiveMovies();
    }
    return data;
  }

  @RequestMapping(value = "/movie/{movieId}", method = RequestMethod.GET)
  public Movie getMovie(@PathVariable(name = "movieId") Long movieId){
    return this.movieRepo.findOne(movieId);
  }

  @RequestMapping(value = "/dummyData", method = RequestMethod.GET)
  public void addDummyData(){

    this.userRepo.deleteAll();
    this.movieRepo.deleteAll();

    for(int i = 0; i < 50; i++){
      this.userRepo.save(new User(new Long(i)));
    }

    for(int i = 0; i < 18; i++){
      this.movieRepo.save(new Movie(new Long(i)));
    }

    List<User> users = new ArrayList();
    this.userRepo.findAll().forEach(u -> {
      users.add(u);
    });

    List<Movie> movies = new ArrayList();
    this.movieRepo.findAll().forEach(m -> movies.add(m));

    users.stream()
        .forEach(user -> {
          for(int i = 1; i < 10; i++){
            Movie = movies.get(new Random().nextInt(18));
            if(user.getLikes().contains(movie)){
              continue;
            }
            user.getLikes().add(movie);
            movie.getLikes().add(user);
```

```
                userRepo.save(user);
                movieRepo.save(movie);
            }
        });

    users.stream()
        .forEach(user -> {
            for(int i = 0; i < 25; i++){
                User userToFollow = users.get(new Random().nextInt(50));
                if(user.getFollowing().contains(userToFollow)
                        || user.getId().longValue() == userToFollow.getId().longValue()){
                    continue;
                }

                user.getFollowing().add(userToFollow);
                userRepo.save(user);
            }
        });


  }
}
```

MyConfiguration.java

```
package com.example;

import org.neo4j.ogm.session.Session;
import org.neo4j.ogm.session.SessionFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Scope;
import org.springframework.context.annotation.ScopedProxyMode;
import org.springframework.data.neo4j.config.Neo4jConfiguration;
import org.springframework.data.neo4j.repository.config.EnableNeo4jRepositories;
import org.springframework.transaction.annotation.EnableTransactionManagement;

author milandeket
 */
@Configuration
@EnableNeo4jRepositories(basePackages = "com.example")
@EnableTransactionManagement
public class MyConfiguration extends Neo4jConfiguration {

  @Bean
  public org.neo4j.ogm.config.Configuration getConfiguration() {
    org.neo4j.ogm.config.Configuration config = new org.neo4j.ogm.config.Configuration();
    config
      .driverConfiguration()
```

```
        .setDriverClassName("org.neo4j.ogm.drivers.http.driver.HttpDriver")
        .setURI("http://neo4j:root@localhost:7474");
      return config;
   }


   @Bean
   @Override
   public SessionFactory getSessionFactory() {
      // with domain entity base package(s)
      return new SessionFactory(getConfiguration(), "com.example");
   }

   @Bean
   @Scope(value = "session", proxyMode = ScopedProxyMode.TARGET_CLASS)
   @Override
   public Session getSession() throws Exception {
      return super.getSession();
   }

}
```

RecommendationData.java

```
package com.example;

import org.springframework.data.neo4j.annotation.QueryResult;
@QueryResult
public class RecommendationData {
  // κλάση που πραγματέυεται τα likes στις ταινίες της βάσης
   private Integer id;
   private Integer likes;

   public RecommendationData() {
   }

   public RecommendationData(Integer id, Integer likes) {
      this.id = id;
      this.likes = likes;
   }

   public Integer getId() {
      return id;
   }

   public Integer getLikes() {
      return likes;
   }
```

```
   public void setId(Integer id) {
      this.id = id;
   }

   public void setLikes(Integer likes) {
      this.likes = likes;
   }

}
```

RecommendationServiceApplication.java

```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@EnableDiscoveryClient
@SpringBootApplication
public class RecommendationServiceApplication{

   public static void main(String[] args) {
         SpringApplication.run(RecommendationServiceApplication.class, args);
   }
}
```

Μαζί και με δύο κλάσεις για τις ταινίες και τους χρήστες, όπως οι παραπάνω Movie.java και User.java ολοκληρώνεται η υλοποίηση της υπηρεσίας. Παρακάτω οι ρυθμίσεις της μικροϋπηρεσίας :

```
server:
  port: 8003
neo4j:
  uri: http://localhost:7474/
  username: neo4j
  password: root
```

Recommendation Client

Παρακάτω ακολουθεί ο κώδικας για το Recommendation Client

Config.java

```
package com.example;

import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;
@Configuration
```

```
public class Config {

   @LoadBalanced
   @Bean
   RestTemplate restTemplate(){
    return new RestTemplate();
   }

}
```

MainController.java

```
package com.example;

import com.example.dto.MovieDTO;
import com.example.dto.RecommendationDTO;
import com.example.dto.UserDTO;
import com.example.service.MovieService;
import com.example.service.RecommendationClientService;
import com.example.service.RecommendationService;
import com.example.service.UserService;
import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;
import com.netflix.ribbon.proxy.annotation.Http;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ExecutionException;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
//διάχείρηση των αιτημάτων προς την υπηρεσία.π΄.΄π.
@RestController
@RequestMapping("/api")
public class MainController {

   @Autowired
   private RecommendationClientService;

   @Autowired
   private UserService;

   @HystrixCommand(fallbackMethod = "recoveryRecommendation")
```

```
   @GetMapping(value ="/recommendation/user/{userId}")
   public List<MovieDTO> getRecommendation(@PathVariable(value = "userId") Long userId) throws
InterruptedException{
      RecommendationDTO dto = null;
      try {
         dto = this.recommendationClientService.getRecommendationData(userId).get();
      } catch (ExecutionException ex) {
         Logger.getLogger(MainController.class.getName()).log(Level.SEVERE, null, ex);
      }
      return dto.getMovies();
   }

   public List<MovieDTO> recoveryRecommendation(Long id){
      List<MovieDTO> movies = new ArrayList();
      movies.add(new MovieDTO("10", "The Lord of the Rings: The Fellowship of the Ring"));
      movies.add(new MovieDTO("14", "One Flew Over the Cuckoo's Nest"));
      movies.add(new MovieDTO("4", "12 Angry Men"));
      movies.add(new MovieDTO("9", "Fight Club"));
      movies.add(new MovieDTO("6", "Pulp Fiction"));
      return movies;
   }

   @GetMapping(value = "/userDetails/{userId}")
   public UserDTO getUserDetails(@PathVariable(name = "userId") Long userId){
      return userService.getUser(userId);
   }

}
```

RecommendationClientApplication.java

```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.netflix.feign.EnableFeignClients;
import org.springframework.cloud.netflix.hystrix.dashboard.EnableHystrixDashboard;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

@SpringBootApplication
@EnableZuulProxy
@EnableFeignClients(basePackages = "com.example")
@EnableDiscoveryClient
@EnableCircuitBreaker
@EnableHystrixDashboard
public class RecommendationClientApplication {
```

```
	public static void main(String[] args) {
		SpringApplication.run(RecommendationClientApplication.class, args);
	}
}
```

Docker

Για να γίνει χρήση της τεχνολογίας Docker, έχουν δημιουργηθεί τα παρακάτω αρχεία:

Docker-compose.yml :

```
version: '3'

services:

 config-service:
  image: config-service
  ports:
   - "8888:8888"
  volumes:
   - ./config:/config


 eureka-service:
  image: eureka-service
  ports:
   - "8761:8761"

 user-service:
  image: user-service
  ports:
   - "8001:8001"
  environment:
   - eureka.client.service-url.defaultZone=http://eureka-service:8761/eureka
   - spring.cloud.config.uri=http://config-service:8888


 movie-service:
  image: movie-service
  ports:
   - "8002:8002"
  environment:
   - eureka.client.service-url.defaultZone=http://eureka-service:8761/eureka
   - spring.cloud.config.uri=http://config-service:8888
   - user-service.url=http://user-service:8001


 recommendation-service:
```

```
  image: recommendation-service
  ports:
   - "8003:8003"
  environment:
   - eureka.client.service-url.defaultZone=http://eureka-service:8761/eureka
   - spring.cloud.config.uri=http://config-service:8888
   - movie-service.url=http://movie-service:8002
   - user-service.url=http://user-service:8001

 recommendation-client:
  image: recommendation-client
  ports:
   - "9000:9000"
  environment:
   - recommendation-service.url=http://recommendation-service:8003
```

Dockerfile σε κάθε μικροϋπηρεσία :

```
FROM openjdk:8-jdk-alpine

ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar

ENTRYPOINT ["java","-jar","/app.jar"]
```

POSTMAN

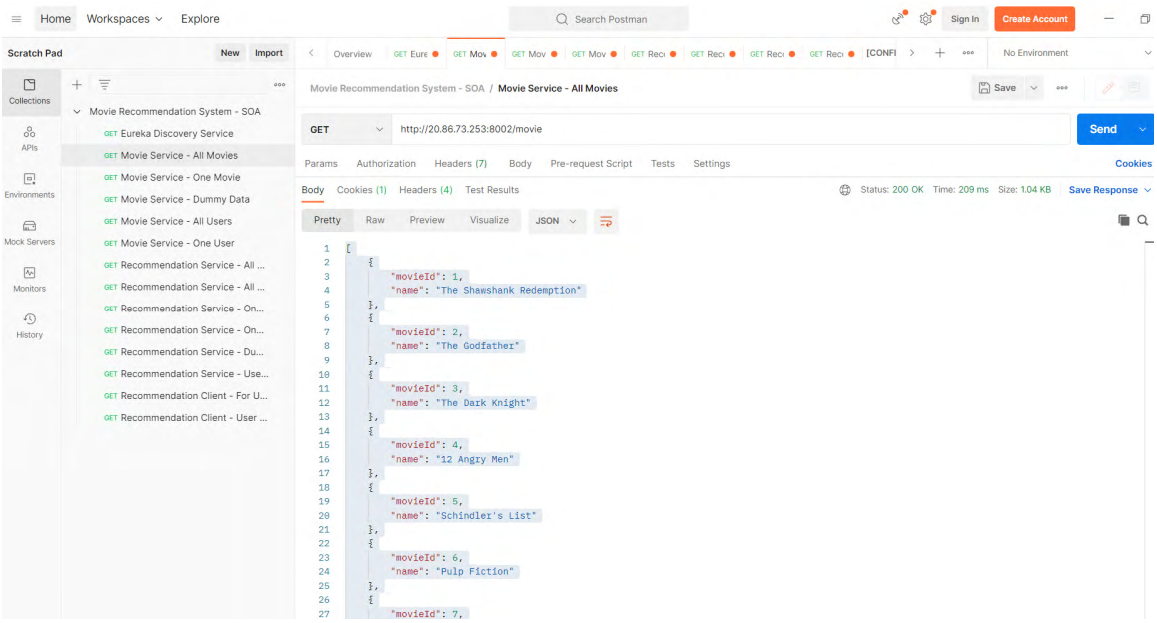
Στο συγκεκριμένο παράδειγμα, αφού συνδέσουμε την υπηρεσία με το Postman μπορούμε να δούμε τα αποτελέσματα από διάφορες κλήσεις στην υπηρεσία. Επιλέγοντας το επιθυμητό είδος αποτελέσματος από το μενού στα αριστερά του παραθύρου, εμφανίζονται οι αντίστοιχες πληροφορίες.

Για παράδειγμα, παρακάτω το αποτέλεσμα της κλήσης στο Postman για να φέρει τη λίστα με όλες τις ταινίες που υπάρχουν αποθηκευμένες. Φαίνεται το id της κάθε ταινίας και ο τίτλος της.


```
  {
    "movieId": 1,
    "name": "The Shawshank Redemption"
  },
  {
    "movieId": 2,
    "name": "The Godfather"
  },
  {
    "movieId": 3,
    "name": "The Dark Knight"
  },
  {
    "movieId": 4,
```

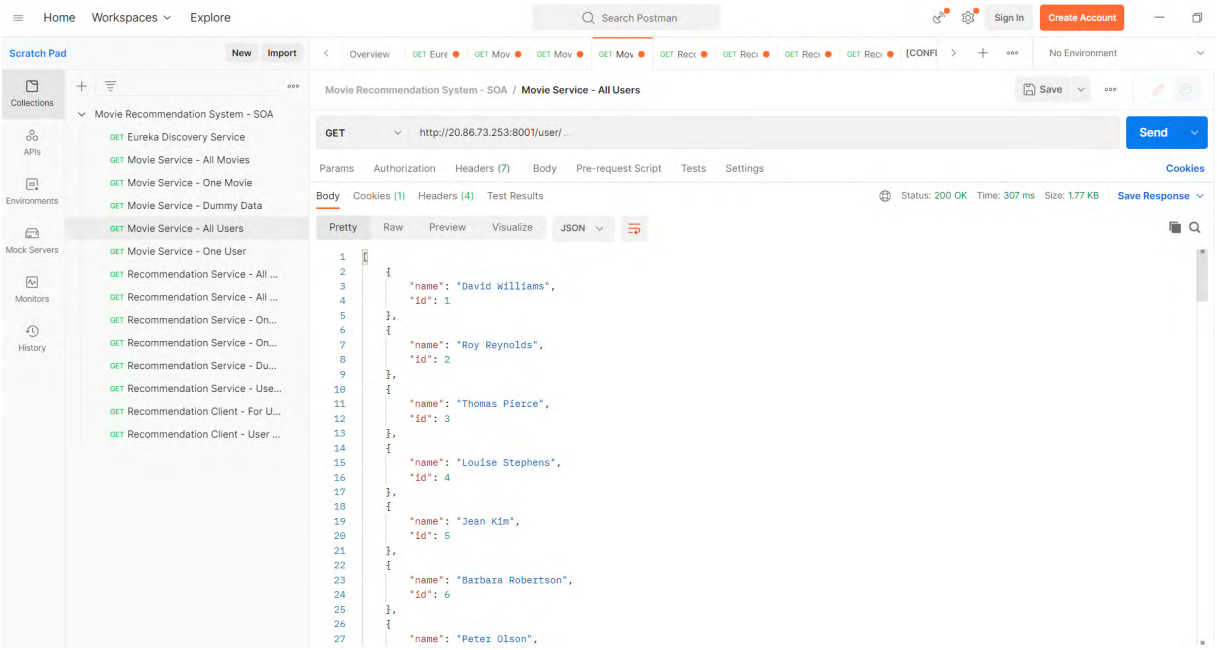
```
    "name": "12 Angry Men"
  },
  {
    "movieId": 5,
    "name": "Schindler's List"
  },
  {
    "movieId": 6,
    "name": "Pulp Fiction"
  },
  {
    "movieId": 7,
    "name": "The Lord of the Rings: The Return of the King"
  },
  {
    "movieId": 8,
    "name": "The Good, the Bad and the Ugly"
  },
  {
    "movieId": 9,
    "name": "Fight Club"
  },
  {
    "movieId": 10,
    "name": "The Lord of the Rings: The Fellowship of the Ring"
  },
  {
    "movieId": 11,
    "name": "Star Wars: Episode V - The Empire Strikes Back"
  },
  {
    "movieId": 12,
    "name": "Forrest Gump"
  },
  {
    "movieId": 13,
    "name": "The Lord of the Rings: The Two Towers"
  },
  {
    "movieId": 14,
    "name": "One Flew Over the Cuckoo's Nest"
  },
  {
    "movieId": 15,
    "name": "Goodfellas"
  },
  {
    "movieId": 16,
```

```
    "name": "The Matrix"
  },
  {
    "movieId": 15,
    "name": "Goodfellas"
  },
  {
    "movieId": 16,
    "name": "The Matrix"
  },
  {
    "movieId": 17,
    "name": "Rocky"
  },
  {
    "movieId": 18,
    "name": "Old boy"
  }
```

Εικόνα 17 Ενδεικτικό της λίστας ταινιών από το Movie Service από το περιβάλλον του Postman

Ακολουθεί ένα δείγμα του αποτελέσματος της κλήσης της λίστας των χρηστών που είναι αποθηκευμένοι στη βάση της υπηρεσίας. Φαίνεται το id κάθε χρήστη και το ονοματεπώνυμό τους.



Εικόνα 18 Ενδεικτικό της λίστας χρηστών από το User Service από το περιβάλλον του Postman

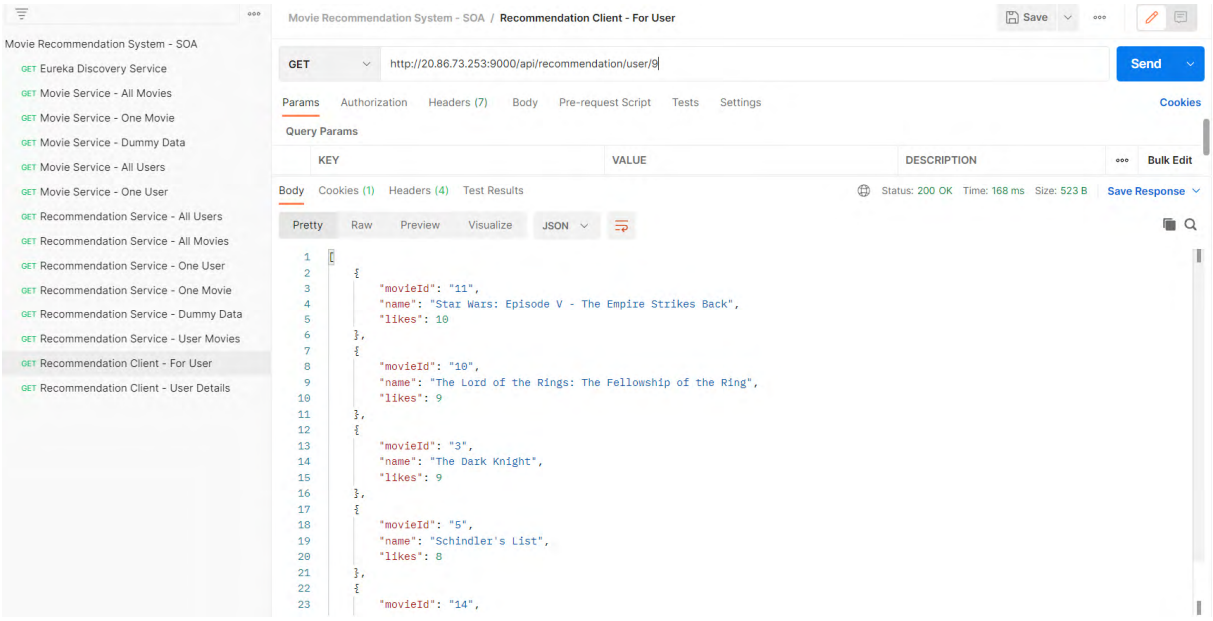
Για τη βασική λειτουργία της υπηρεσίας, που είναι η πρόταση ταινιών σε κάποιο χρήστη, παρακάτω ακολουθούν δύο παραδείγματα.

Συμπληρώνοντας στη γραμμή GET ένα τυχαίο id χρήστη, εδώ το 9 και το 32, παίρνουμε αντίστοιχα δύο διαφορετικές προτάσεις ταινιών.

Προτάσεις για το χρήστη 9:


```
{
    "movieId": "11",
    "name": "Star Wars: Episode V - The Empire Strikes Back",
    "likes": 10
},
{
    "movieId": "10",
    "name": "The Lord of the Rings: The Fellowship of the Ring",
    "likes": 9
},
{
    "movieId": "3",
    "name": "The Dark Knight",
    "likes": 9
},
{
    "movieId": "5",
    "name": "Schindler's List",
```

```
    "likes": 8
  },
  {
    "movieId": "14",
    "name": "One Flew Over the Cuckoo's Nest",
    "likes": 8
  }
```

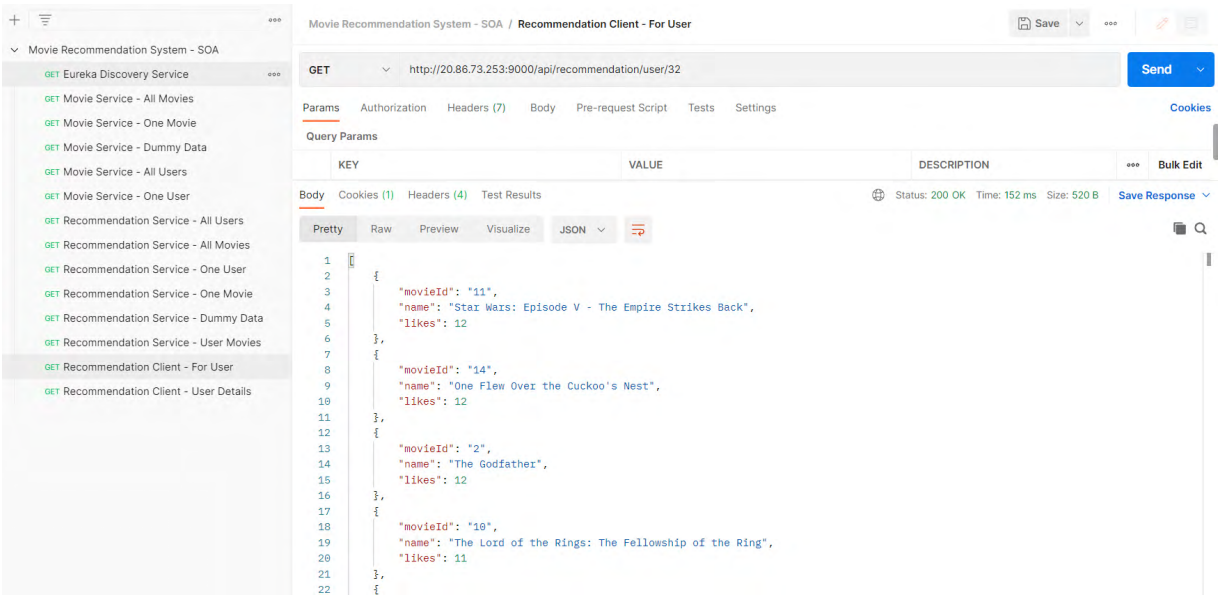



Εικόνα 19 Προτάσεις του Service για το χρήστη με το id 9 από το περιβάλλον του Postman

Προτάσεις για το χρήστη 32:


```
  {
    "movieId": "11",
    "name": "Star Wars: Episode V - The Empire Strikes Back",
    "likes": 12
  },
  {
    "movieId": "14",
    "name": "One Flew Over the Cuckoo's Nest",
    "likes": 12
  },
  {
    "movieId": "2",
    "name": "The Godfather",
    "likes": 12
  },
  {
    "movieId": "10",
```

```
    "name": "The Lord of the Rings: The Fellowship of the Ring",
    "likes": 11
  },
  {
    "movieId": "16",
    "name": "The Matrix",
    "likes": 10
  }
```

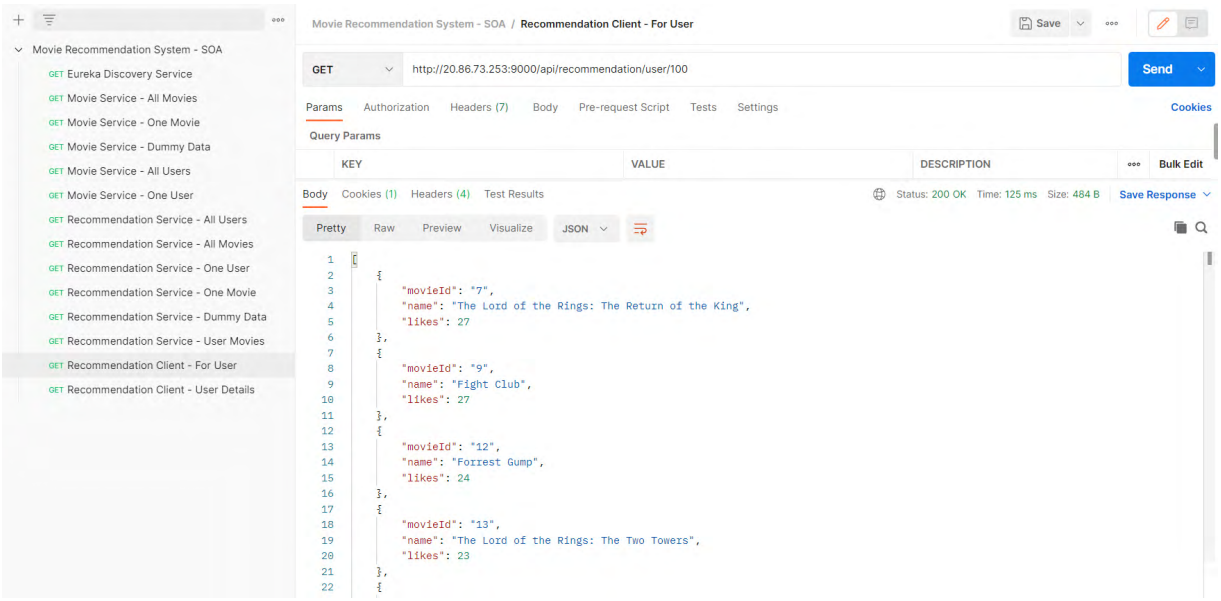
Εικόνα 20 Προτάσεις του Service για τον χρήστη με το id 32 από το περιβάλλον του Postman

Η υπηρεσία, σε περίπτωση που το id του χρήστη που δοθεί δεν υπάρχει στη λίστα χρηστών, ή το Postman χάσει την σύνδεση με τη βάση, ή εμφανιστεί κάποιο πρόβλημα στην υπηρεσία, εμφανίζει μία συγκεκριμένη λίστα με 5 ταινίες.

Προτάσεις για το χρήστη 100:

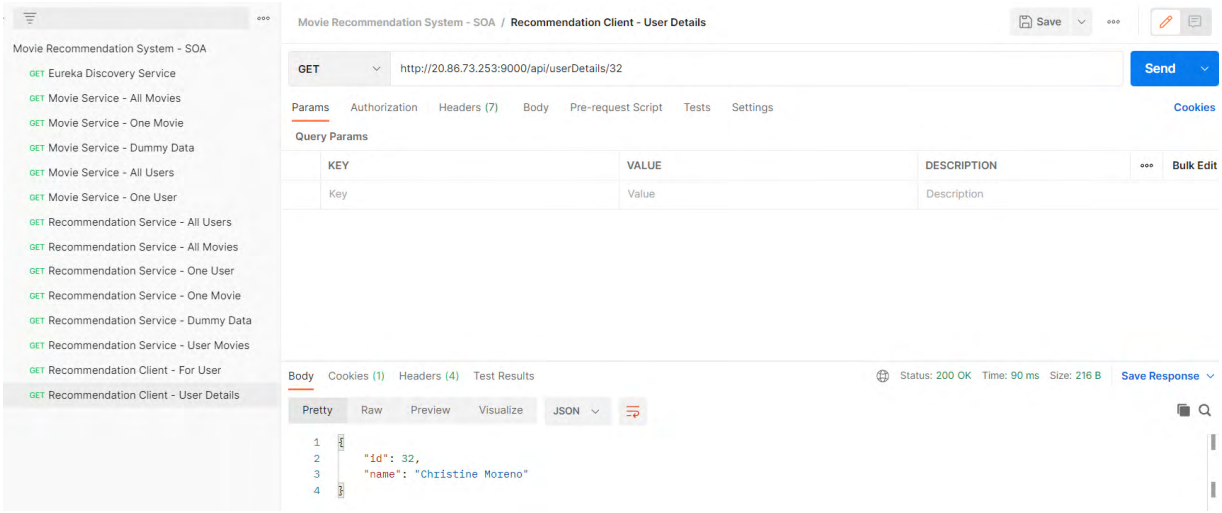

```
  {
    "movieId": "7",
    "name": "The Lord of the Rings: The Return of the King",
    "likes": 27
  },
  {
    "movieId": "9",
    "name": "Fight Club",
    "likes": 27
  },
  {
    "movieId": "12",
```

```
    "name": "Forrest Gump",
    "likes": 24
  },
  {
    "movieId": "13",
    "name": "The Lord of the Rings: The Two Towers",
    "likes": 23
  },
  {
    "movieId": "16",
    "name": "The Matrix",
    "likes": 23
  }
```

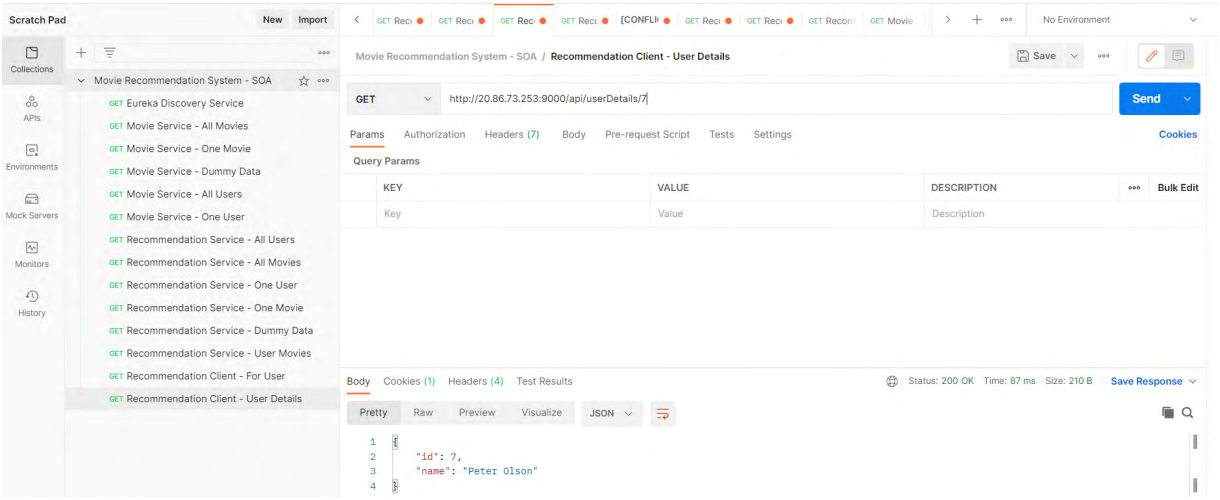



Εικόνα 21 Προτάσεις του Service για τον ανύπαρκτο χρήστη με το id 100 από το περιβάλλον του Postman

Επίσης μπορούμε να βρούμε το ονοματεπώνυμο των χρηστών με βάση το id τους. Παρακάτω βλέπουμε τα αποτελέσματα της αναζήτησης για τους χρήστες με τα id 32 και 9.



Εικόνα 22 Εύρεση του χρήστη με id 32 από το περιβάλλον του Postman



Εικόνα 23 Εύρεση του χρήστη με id 9 από το περιβάλλον του Postman

ΚΕΦΑΛΑΙΟ 5: ΣΥΜΠΕΡΑΣΜΑΤΑ

5.1 Όρια έρευνας

Αρχικά, σκοπός της διπλωματικής εργασίας αυτής ήταν για το κομμάτι της εικονικοποίησης της υπηρεσίας να γίνει και χρήση Kubernetes. Δεδομένου του μεγάλου αριθμού νέων τεχνολογιών που χρησιμοποιήθηκαν, λήφθηκε η απόφαση για τη χρήση μόνο της πιο οικείας πλατφόρμας Docker.

Επίσης, δεδομένης της πολυπλοκότητας της ανάπτυξης ενός γραφικού περιβάλλοντος, προτιμήθηκε η χρήση της αρχικής υπηρεσίας Postman για την εμφάνιση των αποτελεσμάτων της εφαρμογής.

Περιορισμοί προκύπτουν και από τη γλώσσα JAVA που χρησιμοποιήθηκε.Οι εφαρμογές που αναπτύσσονται με τη χρήση της γλώσσας JAVA καταναλώνουν μεγαλύτερο ποσοστό μνήμης, κάτι το οποίο ως αντικειμενοστραφής γλώσσα προγραμματισμού δε μπορεί να ελέγξει, καθώς δεν έχει τη δυνατότητα να διαχειριστεί πόρους σε επίπεδο υλικού. Αυτό τις καθιστά πιο αργές σε σχέση με εφαρμογές με κώδικα σε άλλες γλώσσες. Επίσης ο κώδικας σε JAVA είναι αρκετά πιο περίπλοκος και μακροσκελής σε σχέση με αυτόν σε άλλες γλώσσες.

5.2 Δυσκολίες που προέκυψαν

Από τις δυσκολίες που προέκυψαν κατά τη διεξαγωγή αυτής της διπλωματικής αναγνωρίζεται η εύρεση των σωστών εκδόσεων Ubuntu, JAVA και Neo4j ώστε να λειτουργεί σωστά η υπηρεσία, καθώς ο κώδικας του αποθετηρίου είναι αρκετά παλιός.

Επίσης, στις δυσκολίες της υλοποίησης, περιλαμβάνονται και όλες οι προηγούμενες δοκιμές για τη δημιουργία μίας αποτελούμενης από μικροϋπηρεσίες εφαρμογής και η προσπάθεια διόρθωσης ήδη υπαρχόντων αντίστοιχων υπηρεσιών με την επικαιροποίηση και την αλλαγή του κώδικά τους.

5.3 Συμπεράσματα και συζήτηση

Η υπηρεσία που υλοποιήθηκε στα πλαίσια αυτής της διπλωματικής εργασίας, θα μπορούσε δυνητικά να αποτελέσει μία υπηρεσία αντίστοιχη ήδη υπαρχόντων όπως το IMDb ή το Rotten Tomatoes. Επίσης ο συνδυασμός της με μία υπηρεσία θέασης κατ’ απαίτηση όπως το ευρέως διαδεδομένο, πια, Netflix.

Μέσω της υλοποίησης αυτής της υπηρεσίας, διακρίνουμε τη δυνατότητα της απομόνωσης που προσφέρουν οι μικροϋπηρεσίες και το όφελος αυτής, καθώς τυχόν αλλαγές στον κώδικα ή η εμφάνιση κάποιου προβλήματος δε μπορεί να επηρεάσει τις υπόλοιπες μικροϋπηρεσίες. Αντίστοιχα είναι και τα πλεονεκτήματα της εικονικοποίησης λογισμικού, όπου το απομονωμένο περιβάλλον για την ανάπτυξη

της εφαρμογής προστατεύει τις υπόλοιπες μικροϋπηρεσίες από λάθη που μπορεί να εμφανιστούν σε μία.

Παρά τα πλεονεκτήματα της χρήσης της αρχιτεκτονικής μικροϋπηρεσιών, η χρήση της σε σχέση με τη χρήση της μονολιθικής αρχιτεκτονικής, πρέπει να επιλέγεται ανάλογα με τις απαιτήσεις της εκάστοτε προς ανάπτυξη εφαρμογής.

ΚΕΦΑΛΑΙΟ 6: ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ ΤΗΣ ΕΡΓΑΣΙΑΣ

Καθώς ο χρόνος διεξαγωγής της διπλωματικής εργασίας είναι περιορισμένος, σε μελλοντική ενασχόληση με αυτή θα ήταν ενδιαφέρουσα η χρήση του εργαλείου Ansible,το οποίο έχει αντίστοιχη λειτουργικότητα με το Maven, αλλά και η προσπάθεια συνδυασμού των επιμέρους μικροεφαρμογών από την αρχή, είτε με τη χρήση Ansible είτε με τη χρήση Maven, κάτι το οποίο ξεκίνησα να υλοποιώ αλλά από νωρίς φάνηκε πως είναι ένας περίπλοκος στόχος, πέρα από τα στενά πλαίσια του θέματος της διπλωματικής.

Σε συνέχεια αυτού, μία μελλοντική έκδοση της εφαρμογής θα μπορούσε να χρησιμοποιεί κάποιες πιο σύγχρονες μικροϋπηρεσίες, που προσφέρουν καλύτερες δυνατότητες, όπως αυτές που χρησιμοποιούν μεγάλες πλατφόρμες αντίστοιχης λειτουργίας. Για παράδειγμα το Netflix χρησιμοποιεί το μορφότυπο Apache Kafka για την ασύγχρονη μετάδοση περιεχομένου και το μορφότυπο Spinnaker για τη συνεχή χωρίς διακοπές απόδοση περιεχομένου. Η πλατφόρμα HBO χρησιμοποιεί το εργαλείο Hadoop για την αποθήκευση και την επεξεργασία μεγάλου όγκου δεδομένων. Με τις προσθήκες αυτές, η εφαρμογή δυνητικά θα μπορούσε να υπάρξει και σε ένα παραγωγικό περιβάλλον.

Ακόμη μία προσθήκη για τη βελτίωσή της, θα ήταν η χρήση Kubernetes, καθώς οι δυνατότητες κλιμακοθέτησης, ενορχήστρωσης των περιεκτών, φορητότητας και η διαθεσιμότητα που προσφέρουν θα ήταν ωφέλιμες τόσο για ένα περιβάλλον δοκιμών όσο και για ένα περιβάλλον παραγωγής.

Επίσης μία μελλοντική αλλαγή που θα μπορούσε να γίνει στην εφαρμογή είναι η προσθήκη της δυνατότητας αξιολόγησης των ταινιών από τους χρήστες με κάποιο σύστημα βαθμολόγησης, όπως η βαθμολόγηση στα δέκα ή η βαθμολόγηση με αστεράκια,

Μελλοντικά επίσης τα αποτελέσματα της εφαρμογής θα μπορούσαν να εμφανίζονται σε μία οθόνη με ένα γραφικό περιβάλλον και όχι στο περιβάλλον Postman όπως τώρα. Χρήσιμη επέκταση αυτού, θα ήταν η προσθήκη οθονών για την σύνδεση των χρηστών αλλά και την εγγραφή νέων χρηστών.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Thomas Erl. Zaigham Mahmood, Ricardo Puttini – Cloud Computing Αρχές, Τεχνολογία και Αρχιτεκτονική

[2] Ορισμός του όρου Υπολογιστικό Νέφος, διαθέσιμο στο « https://www.nist.gov/news-events/news/2011/10/final-version-nist-cloud-computing-definition-published»

[3] Ορισμός του όρου Υπολογιστικό Νέφος, διαθέσιμο στο «http://www.epset.gr/el/content/ypologistiko-nefos-cloud-computing»

[4] Πληροφορίες για τα Χαρακτηριστικά Νέφους, διαθέσιμες στο «https://dione.lib.unipi.gr/xmlui/bitstream/handle/unipi/4931/Michopoulos.pdf?sequence=3»

[5] Πληροφορίες για το PaaS, διαθέσιμες στο «https://cited.gr/platform-as-a-service-paas/»

[6] Διαφάνειες μαθήματος «Υπολογιστική Νέφους», Lecture-A, καθ. Δημήτριος Καλλέργης, α.ε. 2019-2020

[7] Πληροφορίες για τα Μοντέλα Ανάπτυξης Νέφους «https://dione.lib.unipi.gr/xmlui/bitstream/handle/unipi/5733/Kostantos.pdf?sequence=2&isAllowed=y»

[8] Πληροφορίες για τους παρόχους Νέφους, διαθέσιμες στο «https://dgtlinfra.com/top-10-cloud-service-providers-2022/»

[9] Πληροφορίες για τον πάροχο Tencent, διαθέσιμες στο «https://www.gartner.com/reviews/market/cloud-infrastructure-and-platform-services/vendor/tencent-cloud»

[10] Πληροφορίες για τα κέντρα δεδομένων, διαθέσιμες στο «https://www.cisco.com/c/en/us/solutions/data-center-virtualization/what-is-a-data-center.html»

[11] Διαφάνειες μαθήματος «Υπολογιστική Νέφους», Lecture-C, καθ. Δημήτριος Καλλέργης, α.ε. 2019-2020

[12] Χρήστος Γιβανούδης – Ανάπτυξη Μικροϋπηρεσιών σε Docker, 2014

[13] Παναγιώτης Σπανιάς – Υλοποίηση ενός πληροφοριακού συστήματος παρουσιολογίου, 2021

[14] Πληροφορίες για την αρχιτεκτονική , διαθέσιμες στο «https://dspace.lib.uom.gr/bitstream/2159/24618/2/BagiasNikolaosMSc2020present.pdf»

[15] Ιωάννης Παπακωνσταντίνου – Μικροϋπηρεσίες και υπολογιστικό νέφος: Μια τεχνοοικονομική προσέγγιση, 2020

[16] Πληροφορίες για την Υπηρεσιοκεντρική Αρχιτεκτονική, διαθέσιμες στο «https://aws.amazon.com/what-is/service-oriented-architecture/»

[17] Πληροφορίες για την Υπηρεσιοκεντρική Αρχιτεκτονική, διαθέσιμες στο «https://www.ibm.com/topics/soa»

[18] Πληροφορίες για την Υπηρεσιοκεντρική Αρχιτεκτονική, διαθέσιμες στο «https://medium.com/@SoftwareDevelopmentCommunity/what-is-service-oriented-architecture-fa894d11a7ec»

[19] Πληροφορίες για την εικονικοποίηση ΛΣ, διαθέσιμες στο «https://aws.amazon.com/what-is/containerization/https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/»

[20] Πληροφορίες για την εικονικοποίηση ΛΣ, διαθέσιμες στο «https://learn.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/https://research.aimultiple.com/containerization/»

[21] Ορισμός του όρου Αυτοματοποίηση διαδικασιών, διαθέσιμος στο «Humble J., & Farley D. (2010) *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional»

[22] Πληροφορίες για την Αυτοματοποίηση διαδικασιών, διαθέσιμες στο «https://puppet.com/resources/whitepaper/state-of-devops-report/»

[23] Πληροφορίες για την Αυτοματοποίηση διαδικασιών, διαθέσιμες στο «https://www.cio.com/article/3248985/5-reasons-to-automate-your-software-development-lifecycle.html»

[24] Πληροφορίες για την Αυτοματοποίηση διαδικασιών, διαθέσιμες στο «https://www.researchgate.net/publication/319063572_Automation_in_Software_Development»

[25] Πληροφορίες για την Αυτοματοποίηση διαδικασιών, διαθέσιμες στο «https://dzone.com/refcardz/microservices-and-workflow-engines»

[26] Πληροφορίες για την Αυτοματοποίηση διαδικασιών, διαθέσιμες στο «https://thenewstack.io/true-success-in-process-automation-requires-microservices»

[27] Ορισμός του όρου Σειριοποίηση δεδομένων «Grus, J. (2015) *Data Science from Scratch: First Principles with Python.* O'Reilly Media, Inc.

[28] Πληροφορίες για την Σειριοποίηση δεδομένων, διαθέσιμες στο «Gorelick M., & Ozsvald I.(2014) *High Performance Python* (2nd ed.). O'Reilly Media

[29] Πληροφορίες για την Σειριοποίηση δεδομένων, διαθέσιμες στο «https://devopedia.org/data-serialization»

[30] Πληροφορίες για την Σειριοποίηση δεδομένων, διαθέσιμες στο «https://www.freecodecamp.org/news/what-is-serialization/»

[31] Πληροφορίες για το Maven, διαθέσιμες στο «https://maven.apache.org/»

[32] Πληροφορίες για το Ansible, διαθέσιμες στο «https://www.ansible.com/what-is-ansible»

[33] Πληροφορίες για το Azure, διαθέσιμες στο «https://learn.microsoft.com/en-us/training/paths/microsoft-azure-fundamentals-describe-cloud-concepts/»

[34] Πληροφορίες για το Azure, διαθέσιμες στο «https://infrastructuremap.microsoft.com/explore»

[35] Πληροφορίες για το Maven, διαθέσιμες στο «https://www.tutorialspoint.com/maven/maven_overview.htm»

[36] Πληροφορίες για το Maven, διαθέσιμες στο «https://www.simplilearn.com/tutorials/maven-tutorial/what-is-maven»

[37] Πληροφορίες για το Maven, διαθέσιμες στο «https://www.geeksforgeeks.org/introduction-apache-maven-build-automation-tool-java-projects/»

[38] Πληροφορίες για το Spring Cloud διαθέσιμες στο «https://spring.io/blog/2014/06/03/introducing-spring-cloud»

[39] Πληροφορίες για το Spring Cloud διαθέσιμες στο «https://www.baeldung.com/spring-cloud-bootstrapping

[40] Πληροφορίες για το Spring Cloud διαθέσιμες στο https://medium.com/javarevisited/what-is-spring-cloud-and-how-it-is-different-from-spring-and-spring-boot-128d276a1432

[41] Πληροφορίες για το Spring Cloud διαθέσιμες στο https://www.educba.com/what-is-spring-cloud/

[42] Πληροφορίες για το Config Service διαθέσιμες στο «https://www.baeldung.com/spring-cloud-configuration»

[43] Πληροφορίες για το Config Service διαθέσιμες στο «https://docs.spring.io/spring-cloud-config/docs/current/reference/html/»

[44] Πληροφορίες για το Eureka Service διαθέσιμες στο «https://medium.com/@ijayakantha/microservices-service-registration-and-discovery-with-netflix-eureka-9a2aa729da96»

[45] Πληροφορίες για το Eureka Service διαθέσιμες στο «https://moviecultists.com/how-eureka-server-works-internally»

[46] Πληροφορίες για το Eureka Service διαθέσιμες στο «https://dzone.com/articles/spring-cloud-amp-spring-bootimplementing-eureka-se»

[47] Πληροφορίες για τη MySQL διαθέσιμες στο https://www.hostinger.com/tutorials/what-is-mysql#So_What_is_MySQL](https://www.hostinger.com/tutorials/what-is-mysql#So_What_is_MySQL)

[48] Πληροφορίες για τη MySQL διαθέσιμες στο «https://www.w3schools.com/MySQL/default.asp»

[49] Πληροφορίες για τη MongoDB διαθέσιμες στο «https://www.mongodb.com/why-use-mongodb»

[50] Πληροφορίες για τη MongoDB διαθέσιμες στο «https://vontikakis.com/el/blog/nosql-baseis-dedomenon-mongodb-ta-prota-bemata»

[51] Πληροφορίες για το Neo4j διαθέσιμες στο «https://neo4j.com/»

[52] Πληροφορίες για το Neo4j διαθέσιμες στο «https://www.tutorialspoint.com/neo4j/neo4j_overview.htm»

[53] Πληροφορίες για το Hystrix διαθέσιμες στο «https://www.baeldung.com/spring-cloud-netflix-hystrix»

[54] Πληροφορίες για το Hystrix διαθέσιμες στο «https://medium.com/techno101/hystrix-spring-boot-implementation-34dafab2c819»

[55] Πληροφορίες για το Hystrix διαθέσιμες στο «https://howtodoinjava.com/spring-cloud/spring-hystrix-circuit-breaker-tutorial/»

[56] Πληροφορίες για το Zuul διαθέσιμες στο «https://blog.heroku.com/using_netflix_zuul_to_proxy_your_microservices»

[57] Πληροφορίες για το Zuul διαθέσιμες στο «https://netflixtechblog.com/announcing-zuul-edge-service-in-the-cloud-ab3af5be08ee»

[58] Πληροφορίες για το Feign διαθέσιμες στο «https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-feign.html»

[59] Πληροφορίες για το Feign διαθέσιμες στο «https://www.javatpoint.com/using-feign-rest-client-for-service-invocation»

[60] Πληροφορίες για την εξισορρόπηση φορτίου διαθέσιμες στο «https://www.nginx.com/resources/glossary/load-balancing/»

[61] Πληροφορίες για την εξισορρόπηση φορτίου διαθέσιμες στο «https://www.nginx.com/resources/glossary/load-balancing/»

[62] Πληροφορίες για την εξισορρόπηση φορτίου διαθέσιμες στο «https://aws.amazon.com/what-is/load-balancing/»

[63] Πληροφορίες για την εξισορρόπηση φορτίου διαθέσιμες στο «https://www.citrix.com/solutions/app-delivery-and-security/load-balancing/what-is-load-balancing.html»

[64] Πληροφορίες για το Ribbon διαθέσιμες στο «https://dzone.com/articles/microservices-tutorial-ribbon-as-a-load-balancer-1»

[65] Πληροφορίες για το Ribbon διαθέσιμες στο «https://www.studytonight.com/post/load-balancing-spring-boot-microservices-using-netflixs-ribbon»

[66] Πληροφορίες για το Rest API διαθέσιμες στο «https://www.ibm.com/topics/rest-apis»

[67] Πληροφορίες για το Rest API διαθέσιμες στο «https://www.redhat.com/en/topics/api/what-is-a-rest-api»

[68] Πληροφορίες για το Postman διαθέσιμες στο «https://www.postman.com/company/about-postman/»

[69] Πληροφορίες για το Postman διαθέσιμες στο «https://hevodata.com/learn/postman-rest-client/»

[70]Πληροφορίες για το Docker διαθέσιμες στο «https://takacsmark.com/dockerfile-tutorial-by-example-dockerfile-best-practices-2018/»

[71]Πληροφορίες για το Docker διαθέσιμες στο «https://docs.docker.com/get-started/overview/»

[72]Πληροφορίες για το Docker διαθέσιμες στο «https://www.ibm.com/topics/docker»

[73]Πληροφορίες για το Docker διαθέσιμες στο «https://docs.docker.com/engine/»

[74]Πληροφορίες για το Docker διαθέσιμες στο «https://www.aquasec.com/cloud-native-academy/docker-container/docker-architecture/»