Ατομική Διπλωματική Εργασία

ΑΝΑΠΤΥΞΗ ΚΑΙ ΕΚΤΙΜΗΣΗ ΕΦΑΡΜΟΓΗΣ ΙΣΤΟΥ ΒΑΣΙΣΜΕΝΗ ΣΤΟ ΑΡΧΙΤΕΚΤΟΝΙΚΟ ΜΟΝΤΕΛΟ ΜΙΚΡΟΥΠΗΡΕΣΙΩΝ

Νικόλας Μίκαλλος

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη και εκτίμηση εφαρμογής ιστού βασισμένη στο αρχιτεκτονικό μοντέλο μικρουπηρεσιών.

Νικόλας Μίκαλλος

Επιβλέπων Καθηγητής
Χάρης Βώλος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2023

Περίληψη

Η αρχιτεκτονική των μικρουπηρεσιών έχει γίνει μια από τις πιο δημοφιλείς επιλογές στον σχεδιασμό και την ανάπτυξη λογισμικού. Αυτή η προσέγγιση αποσπά μια μονολιθική εφαρμογή σε μικρότερες, αυτόνομες υπηρεσίες που συνεργάζονται μεταξύ τους. Κάθε μια από αυτές τις μικρουπηρεσίες εκτελεί μια συγκεκριμένη λειτουργία που μπορεί να αναπτυχθεί (developed), να εγκατασταθεί (deployed) και να κλιμακωθεί (scaled) ανεξάρτητα. Τόσο η αρχιτεκτονική των μικρουπηρεσιών, όσο και η μονολιθική αρχιτεκτονική έχουν τα δικά τους προτερήματα και μειονεκτήματα και η κάθε μια μπορεί να είναι πιο αποδοτική προσέγγιση από την άλλη, ανάλογα με τις ανάγκες μιας εφαρμογής.

Η παρούσα έρευνα περιγράφει τις αρχιτεκτονικές των μικρουπηρεσιών και μονολιθικής ανάπτυξης, τη διαδικασία ανάπτυξης της εφαρμογής FoodTrucks και παρουσιάζει σύγκριση των δυο αυτών αρχιτεκτονικών, με τη πειραματική αξιολόγηση της εφαρμογής η οποία υλοποιήθηκε σαν μονολιθική και με τη χρήση μικρουπηρεσιών.

Οι δυο αυτές αρχιτεκτονικές, αναπτύχθηκαν σε συστάδες (cluster) του cloudlab για να μπορούν να εξεταστούν με το εργαλείο wrk. Τα πειράματα που πραγματοποιήθηκαν, εξετάζουν τη συμπεριφορά της κάθε εφαρμογής κάτω από διαφορετικά φόρτοι εργασίας και πως οι εφαρμογές κλιμακώνονται.

Τα συμπεράσματα αυτής της έρευνας υπογραμμίζουν τη σημασία της επιλογής της κατάλληλης αρχιτεκτονικής λογισμικού, ανάλογα με τις ειδικές ανάγκες και τις απαιτήσεις κάθε συστήματος.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Γενική Εισαγωγή

Το πεδίο των μικρουπηρεσιών έχει αναπτυχθεί γρήγορα τα τελευταία χρόνια και έχει γίνει ένα από τα κύρια μοντέλα ανάπτυξης λογισμικού για πολλές εταιρείες. Οι μικρουπηρεσίες παρέχουν οφέλη όπως τη γρήγορη ανάπτυξη, την ευελιξία και την προσαρμοστικότητα, που είναι απαραίτητα για την ανταπόκριση στις σύγχρονες ανάγκες ανάπτυξης λογισμικού. Παρόλο που οι κύριες ιδέες της αρχιτεκτονικής των μικρουπηρεσιών βασίζονται στην αρχιτεκτονική προσανατολισμένη στις υπηρεσίες (Service Oriented Architecture - SOA), τα σημεία στα οποία δίνει έμφαση είναι αυτά που εξυπηρετούν καλύτερα τις σύγχρονες ανάγκες ανάπτυξης λογισμικού.

Ένα από τα κύρια πλεονεκτήματα των μικρουπηρεσιών είναι η δυνατότητα γρήγορης ανάπτυξης και κυκλοφορίας νέων χαρακτηριστικών στα συστήματα εταιρειών. Αυτό επιτρέπει στις εταιρείες να ανταποκρίνονται γρήγορα στις αλλαγές της αγοράς και στις ανάγκες των χρηστών. Επιπλέον, οι μικρουπηρεσίες επιτρέπουν την παράλληλη ανάπτυξη και επέκταση του λογισμικού, καθώς κάθε υπηρεσία λειτουργεί ανεξάρτητα από τις υπόλοιπες. Αυτό σημαίνει ότι οι ομάδες ανάπτυξης μπορούν να εργάζονται παράλληλα σε διάφορες υπηρεσίες, χωρίς να επηρεάζονται ή να εξαρτώνται η μία από την άλλη. Κάθε μικρουπηρεσία μπορεί να αναπτυχθεί, να ελεγχτεί και να αναβαθμιστεί ανεξάρτητα, επιτρέποντας την ταχύτερη παραγωγή και ενσωμάτωση νέων λειτουργιών στο σύστημα.

Επιπλέον, οι μικρουπηρεσίες προωθούν την ευελιξία και την προσαρμοστικότητα στον τρόπο ανάπτυξης λογισμικού. Κατά την ανάπτυξη μιας υπηρεσίας, η έμφαση δίνεται στην ανεξαρτησία της από το σύστημα. Αυτό σημαίνει ότι μπορεί να υλοποιηθεί χρησιμοποιώντας διαφορετικές τεχνολογίες, γλώσσες προγραμματισμού ή πλατφόρμες ανάπτυξης. Αυτή η ευελιξία επιτρέπει στις εταιρείες να επιλέγουν τις βέλτιστες λύσεις για κάθε μικρουπηρεσία, λαμβάνοντας υπόψη τις ανάγκες και τις απαιτήσεις τους ξεχωριστά. Αυτό επιτρέπει στις εταιρείες να εκμεταλλευτούν τις πλέον καινοτόμες τεχνολογίες και πρακτικές ανάπτυξης, ενώ ταυτόχρονα διατηρούν το υπάρχον σύστημα λειτουργικό και εύχρηστο.

Ωστόσο, παρά τα πλεονεκτήματα των μικρουπηρεσιών, είναι σημαντικό να αναγνωρίζουμε ότι δεν είναι κάθε σύστημα κατάλληλο για την υιοθέτηση αυτής της αρχιτεκτονικής. Υπάρχουν περιπτώσεις όπου ένα σύστημα με πολύ απλή λειτουργικότητα ή χαμηλές απαιτήσεις σε ευελιξία και προσαρμοστικότητα μπορεί να μην επωφεληθεί από τη μετάβαση σε μικρουπηρεσίες. Σε αυτές τις περιπτώσεις, η παραδοσιακή μονολιθική υλοποίηση μπορεί να είναι πιο αποδοτική και οικονομική.

Το πεδίο των μικρουπηρεσιών είναι σχετικά νέο και παρόλο που υπάρχει προηγούμενη έρευνα, υπάρχει η ανάγκη για νέα μετροπρογράμματα που μπορούν να μετρήσουν και να αξιολογήσουν την επίδοση μιας εφαρμογής στην αρχιτεκτονική των μικρουπηρεσιών σε σύγκριση με την αντίστοιχη μονολιθική υλοποίηση. Οι μετρήσεις αυτές είναι σημαντικές για να κατανοήσουμε την επίδραση των μικρουπηρεσιών στη συνολική απόδοση ενός συστήματος. Μέσω αυτών των μετρήσεων, μπορούμε να αναγνωρίσουμε πιθανά σημεία βελτίωσης, προβλήματα απόδοσης ή σημεία αδυναμίας που μπορεί να προκύψουν από την εφαρμογή της αρχιτεκτονικής των μικρουπηρεσιών.

1.2 Στόχος Διπλωματικής Εργασίας

Ο σκοπός αυτής της μελέτης είναι να παρουσιάσει μια εκτενή σύγκριση μεταξύ της σύγχρονης και ευέλικτης αρχιτεκτονικής των μικρουπηρεσιών και της παραδοσιακής μονολιθικής προσέγγισης στην ανάπτυξη λογισμικού. Συγκεκριμένα, ο στόχος είναι να εξεταστεί η συμπεριφορά ενός μονολιθικού συστήματος κάτω από μεγάλο φόρτο εργασίας σε σύγκριση με την αντίστοιχη υλοποίηση χρησιμοποιώντας την αρχιτεκτονική των μικρουπηρεσιών. Επιπλέον, η μελέτη αυτή στοχεύει στο να εξετάσει πώς μεταβάλλεται αυτή η συμπεριφορά κατά την κλιμάκωση των εφαρμογών. Μέσω αυτής της έρευνας, μπορούμε να αναδείξουμε τα πλεονεκτήματα και τους περιορισμούς κάθε αρχιτεκτονικής, προσφέροντας πολύτιμες πληροφορίες για τη λήψη ενημερωμένων αποφάσεων σχετικά με την βέλτιστη επιλογή μεταξύ τους σε διάφορα σενάρια εφαρμογής.

1.3 Συνεισφορά

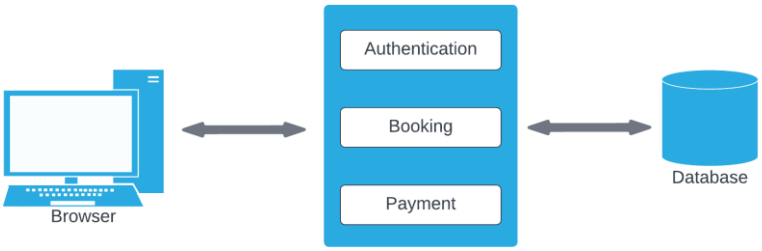
Η κύρια συνεισφορά αυτής της εργασίας είναι η ανάπτυξη του μετροπρογράμματος Foodtrucks, το οποίο διαθέτει υλοποιήσεις σαν μονολιθική εφαρμογή και στην αρχιτεκτονική των μικρουπηρεσιών. Παράλληλα, η εργασία παρουσιάζει τις διαφορές ανάμεσα στις δυο αρχιτεκτονικές, προσφέροντας μια κατανοητή και ολοκληρωμένη εικόνα των δυνατοτήτων και περιορισμών κάθε προσέγγισης.

Κεφάλαιο 2

Υπόβαθρο

2.1 Μονολιθική Αρχιτεκτονική

Η μονολιθική αρχιτεκτονική θεωρείται ο παραδοσιακός τρόπος ανάπτυξης ενός συστήματος λογισμικού. Μια μονολιθική εφαρμογή περιλαμβάνει όλες τις λειτουργίες και τον κώδικα που χρειάζεται για να εξυπηρετεί την επιχειρηματική λογική πίσω από τον σχεδιασμό της. Παρόλο που η εφαρμογή μπορεί να έχει μια τμηματική δομή, συσκευάζεται (packaged) σε ένα εκτελέσιμο αρχείο το οποίο αναπτύσσεται (deployed) σε έναν εξυπηρετητή. Αυτό σημαίνει ότι όλες οι αλλαγές ή προσθήκες στον κώδικα επηρεάζουν ολόκληρη την εφαρμογή και συνοδεύονται από ελέγχους ακόμη και σε λειτουργικά ανεξάρτητα κομμάτια του συστήματος. Στην Εικόνα 2.1 παρουσιάζεται μια αφαιρετική όψη της μονολιθικής αρχιτεκτονικής.



Εικόνα 2.1: Αφαιρετική αναπαράσταση μονολιθικής αρχιτεκτονικής

Μια μονολιθική εφαρμογή είναι σχετικά εύκολη στην υλοποίηση και στην ανάπτυξη (deployment), αλλά τα μειονεκτήματα είναι αρκετά για να κάνουν ένα εναλλακτικό μοντέλο αρχιτεκτονικής να φαίνεται ελκυστικό. Αρχικά οι μονολιθικές εφαρμογές είναι πιο επιρρεπείς σε καταστροφικά λάθη. Ο πηγαίος κώδικας είναι στενά συνδεδεμένος και ένα λάθος σε ένα σημείο του κώδικα μπορεί να ρίξει ολόκληρη την εφαρμογή. Παράλληλα, μια εφαρμογή με πολύπλοκη λειτουργικότητα, και τον μεγάλο όγκο κώδικα που τη συνοδεύει, είναι αρκετά πιο δυσνόητη από ένα προγραμματιστή σε σχέση με πιο μικρά απλά προγράμματα. Μεγάλες και πολύπλοκες εφαρμογές έχουν αυξημένη δυσκολία συντήρησης και η πιθανότητα να γίνει κάποιο καταστροφικό σφάλμα κατά την συντήρηση είναι πολύ πιο μεγάλη.

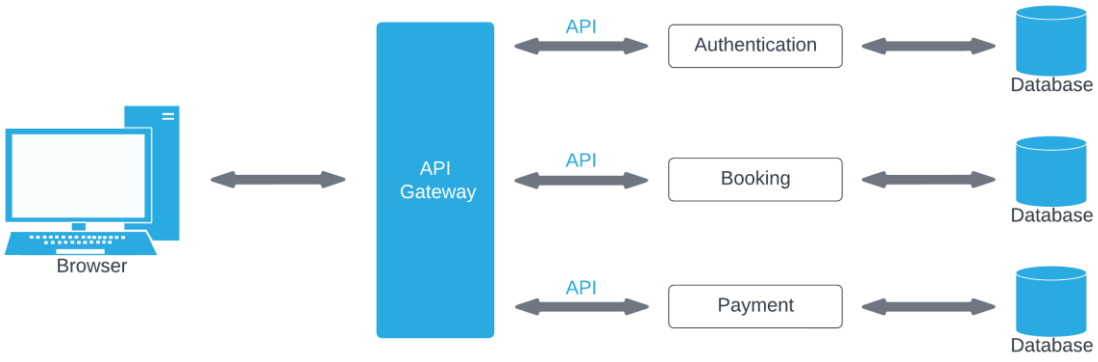
Ένα άλλο σημείο στο οποίο η μονολιθική προσέγγιση υστερεί είναι η διαθεσιμότητα. Σε κάθε αναβάθμιση, πρέπει να αναπτύσσεται ξανά (redeploy) ολόκληρη η εφαρμογή με αποτέλεσμα να μειώνεται η διαθεσιμότητα.

Τέλος, οι μονολιθικές εφαρμογές αντιμετωπίζουν περισσότερες προκλήσεις στην κλιμάκωση. Όλες οι λειτουργίες και οι υπηρεσίες ενσωματώνονται σε ένα ενιαίο πρόγραμμα, επομένως η επέκτασή τους απαιτεί την κλιμάκωση ολόκληρης της εφαρμογής, ακόμη και για την εξυπηρέτηση μεμονωμένων λειτουργιών. Αυτό οδηγεί σε αυξημένο κόστος και πιθανή υπερβολική χρήση πόρων, καθιστώντας τη διαδικασία κλιμάκωσης πιο πολύπλοκη και δαπανηρή.

Λαμβάνοντας υπόψη τα πιο πάνω είναι πιθανό κάποιος να συμπεράνει πως η μονολιθική αρχιτεκτονική δεν είναι καλή επιλογή για την ανάπτυξη μιας εφαρμογής. Παρόλα αυτά, η μονολιθική προσέγγιση μπορεί να αποδειχθεί αποτελεσματική σε αρκετές περιπτώσεις, αφού όπως θα δούμε στη συνέχεια, η αρχιτεκτονική των μικρουπηρεσιών παρουσιάζει νέες προκλήσεις.

2.2 Μικρουπηρεσίες

Παρόλο που οι μικρουπηρεσίες δεν έχουν κάποιο ευρέως αποδεκτό ορισμό, μπορούμε να καθορίσουμε την αρχιτεκτονική προσέγγιση των μικρουπηρεσιών σαν την αποσύνθεση μιας εφαρμογής σε πολλά μικρά μέρη τα οποία έχουν χαμηλή σύζευξη μεταξύ τους και μπορούν να αλλάξουν, να κλιμακωθούν και να εξεταστούν ανεξάρτητα [Αγγελόπουλος2020]. Κάθε ένα από αυτά τα «μέρη» ή πιο σωστά υπηρεσίες, έχει μια λειτουργικότητα και είναι υπεύθυνη για αυτή. Συνήθως, η κάθε υπηρεσία έχει τη δική της τεχνολογική στοίβα (βάση δεδομένων, γλώσσα προγραμματισμού κ.α.) και το δικό της επιχειρησιακό μοντέλο και επικοινωνεί με άλλες υπηρεσίες με ένα συνδυασμό τεχνολογιών. Στην Εικόνα 2.2 παρουσιάζεται μια αφαιρετική όψη της αρχιτεκτονικής των μικρουπηρεσιών. Για τον τρόπο που στήνεται μια υπηρεσία και πως μπορεί να επικοινωνήσει με άλλες υπηρεσίες, θα μιλήσουμε με μεγαλύτερη λεπτομέρεια σε επόμενα κεφάλαια.



Εικόνα 2.2: Αφαιρετική αναπαράσταση αρχιτεκτονικής μικρουπηρεσιών

Το αρχιτεκτονικό στυλ των μικρουπηρεσιών αντιμετωπίζει πολλά από τα μειονεκτήματα της μονολιθικής προσέγγισης [Gos2020]. Αρχικά, οι μικρουπηρεσίες είναι πιο εύκολο να συντηρηθούν, αφού έχουμε μικρά σχετικά κομμάτια κώδικα, που μπορούν να γίνουν πιο εύκολα κατανοητά από τον προγραμματιστή. Παράλληλα, η αξιοπιστία και η διαθεσιμότητα ενός συστήματος αυξάνεται, αφού κάθε κύρια λειτουργεία της εφαρμογής είναι ανεξάρτητη, επομένως, στην περίπτωση καταστροφικών σφαλμάτων σε μία υπηρεσία, επηρεάζεται μόνο η λειτουργικότητα που προσφέρει η συγκεκριμένη υπηρεσία και όχι ολόκληρη η εφαρμογή. Ταυτόχρονα, κατά την αναβάθμιση της εφαρμογής, αναπτύσσεται ξανά (redeploy) μόνο η υπηρεσία η οποία αναβαθμίστηκε, με αποτέλεσμα να ελαχιστοποιείται ο χρόνος που χρειάζεται μέχρι να είναι ξανά διαθέσιμη η υπηρεσία (down time). Τέλος, αυτή η τμηματική δόμηση της εφαρμογής, επιτρέπει την εύκολη κλιμάκωση, αφού μπορούμε να κλιμακώσουμε μεμονωμένες υπηρεσίες που απαιτούν περισσότερους πόρους. Με αυτό τον τρόπο δεν σπαταλάμε πόρους σε λειτουργίες της εφαρμογής που δεν τους χρειάζονται.

Παρόλο που τα πλεονεκτήματα των μικρουπηρεσιών είναι αρκετά, η δυσκολία υλοποίησης και ανάπτυξης (deployment) της εφαρμογής αυξάνεται σημαντικά. Παράλληλα, η έντονη χρήση του δικτύου για την εσωτερική επικοινωνία των υπηρεσιών αυξάνει την πολυπλοκότητα αποσφαλμάτωσης και δοκιμής της εφαρμογής. Για αυτό το λόγο, υπάρχει η αντίληψη ότι είναι καλύτερα να ξεκινά η υλοποίηση ενός συστήματος σαν μια μονολιθική εφαρμογή και στη συνέχεια, αν υπάρχει η ανάγκη, να μεταφέρεται σε μικρουπηρεσίες. Ο βασικός λόγος είναι πως η μεγάλη πολυπλοκότητα που συνοδεύει την υλοποίηση με μικρουπηρεσίες, επιβραδύνει την ανάπτυξη στα αρχικά στάδια. Συνήθως, κατά την υλοποίηση μιας εφαρμογής στα αρχικά στάδια της σχεδίασης, η ιδέα της εφαρμογής δεν είναι ακόμη ξεκάθαρη. Γι’ αυτό το λόγο υπάρχει η ανάγκη υλοποίησης μιας δοκιμαστικής εφαρμογής, η οποία θα βοηθήσει στην εξακρίβωση των αναγκών αλλά και της βιωσιμότητας της ιδέας της εφαρμογής. Επομένως, ο χρόνος που θα σπαταληθεί στη υλοποίηση των μικρουπηρεσιών ίσως να μην αποσβεσθεί στον μέλλον. Ακόμα όμως και αν οι προδιαγραφές και οι απαιτήσεις της εφαρμογής είναι ξεκάθαρες από την αρχή, οι μικρουπηρεσίες θα έχουν θετικό αντίκτυπο στην υλοποίηση της εφαρμογής μόνο αν γίνει σωστή αποσύνθεση των λειτουργιών της εφαρμογής, που αποδεικνύεται εξαιρετικά δύσκολο επίτευγμα χωρίς να υπάρχει πρακτική εμπειρία που αποκαλύπτει τις ιδιαιτερότητες ενός συστήματος.
Με την υβριδική προσέγγιση, δίνεται χρόνος για βαθύτερη κατανόηση των απαιτήσεων ενός συστήματος και τη σωστή αποδόμηση των λειτουργιών της εφαρμογής, χωρίς να θυσιάζεται η ταχύτητα υλοποίησης.

Στον Πίνακα 2.1 παρουσιάζεται μια συνοπτική σύγκριση των κύριων σημείων που έχουν αναπτυχθεί πιο πάνω για την αρχιτεκτονική των μικρουπηρεσιών και της μονολιθικής αρχιτεκτονικής.

	Μικρουπηρεσίες	Μονολιθικό
Βαθμός Πολυπλοκοτήτας (Complexity)	Μεγάλος	Μικρός
Διαθεσημότητα (Availability)	Μεγάλη	Μικρή
Χρήση Πόρων (Resource Consumption)	Μικρή	Μεγάλη
Κλειμάκωση (Scaling)	Εύκολη	Δύσκολη
Ευελιξία (Agility)	Μεγάλη	Μικρή
Καθυστέρηση Δικτύου (Latency)	Μεγάλη	Μικρή
Ρυθμαπόδοση (Throughput)	Μικρή	Μεγάλη

Πίνακας 2.1 Σύγκριση Μικρουπηρεσιών και Μονολιθικού

2.3 Πρωτόκολλα επικοινωνίας: gRPC και REST APIs

Όπως έχει αναφερθεί στο προηγούμενο κεφάλαιο, για την ορθή λειτουργία μιας εφαρμογής βασισμένης στο αρχιτεκτονικό μοτίβο των μικρουπηρεσιών, χρειάζεται εσωτερική επικοινωνία μεταξύ των υπηρεσιών. Δυο από τα πιο δημοφιλή πρωτόκολλα επικοινωνίας που χρησιμοποιούνται είναι gRPC και REST. Παρόλο που η λειτουργικότητα που προσφέρουν είναι η ίδια, ο σχεδιασμός τους είναι αρκετά διαφορετικός.

Το REST (Representational State Transfer) API είναι μια αρχιτεκτονική προσέγγιση για την ανάπτυξη διαδικτυακών υπηρεσιών, βασισμένη στην αρχή της κατανεμημένης υπολογιστικής αναπαράστασης κατάστασης. Τα REST API χρησιμοποιούν το πρωτόκολλο HTTP για την επικοινωνία μεταξύ του πελάτη και του διακομιστή, χρησιμοποιώντας τυπικές μεθόδους HTTP όπως GET, POST, PUT και DELETE για τη διαχείριση των πόρων. Τα REST API χαρακτηρίζονται από την απλότητα και την επεκτασιμότητα τους.

Το gRPC (gRPC Remote Procedure Call) είναι ένα σύγχρονο, ανοιχτού κώδικα πλαίσιο επικοινωνίας ανάμεσα σε υπηρεσίες, που αναπτύχθηκε από τη Google. Σε αντίθεση με τα REST API, το gRPC χρησιμοποιεί το πρωτόκολλο HTTP/2 για τη μεταφορά δεδομένων, προσφέροντας μεγαλύτερη απόδοση, μικρότερη κατανάλωση πόρων και αυξημένη ταχύτητα επικοινωνίας. Το gRPC χρησιμοποιεί τη γλώσσα περιγραφής διαδικασιών (Protocol Buffers) για τη δήλωση των υπηρεσιών και τη σειριοποίηση των δεδομένων, παρέχοντας μια αυτοματοποιημένη και αποτελεσματική διαδικασία για τη δημιουργία και την επικοινωνία υπηρεσιών.

Στον Πίνακα 2.2 παρουσιάζονται οι κύριες διαφορές στο σχεδιασμό του gRPC σε σχέση με REST οι οποίες αναλύονται και πιο κάτω:

1. Σχεδιασμός API:

   Το gRPC ακολουθεί το μοτίβο Remote Procedure Call (RPC), όπου οι υπηρεσίες ορίζουν μεθόδους και τα αντίστοιχα μηνύματα αιτήματος και απάντησης χρησιμοποιώντας τη γλώσσα περιγραφής διεπαφών protobuf. Οι πελάτες καλούν στη συνέχεια αυτές τις μεθόδους σαν να ήταν τοπικές στο δικό τους σύστημα. Αντίστοιχα, το REST ακολουθεί τις αρχές του Representational State Transfer, χρησιμοποιώντας τις τυπικές μεθόδους HTTP (GET, POST, PUT, DELETE) που αλληλοεπιδρούν με πόρους που αναγνωρίζονται από διευθύνσεις URL.

2. Πρωτόκολλο:

   Το gRPC βασίζεται στο πρωτόκολλο επικοινωνίας HTTP/2 ενώ το REST στο πρωτόκολλο HTTP/1.1. Υπάρχουν αρκετές διαφορές μεταξύ των δυο πρωτόκολλων, αλλά η βασική διαφορά είναι η απόδοση. Ενώ το HTTP/1.1 στέλνει τα πακέτα επικοινωνίας το ένα μετά το άλλο, δημιουργώντας νέα σύνδεση για το κάθε πακέτο, το HTTP/2 χρησιμοποιεί μια TCP σύνδεση μέσω της οποίας μπορεί να στείλει πολλαπλές ροές δεδομένων χωρίς να υπάρχει συμφόρηση.

3. Μορφή δεδομένων:

   Το gRPC χρησιμοποιεί Protocol Buffers (protobuf). Το protobuf είναι ένα πρωτόκολλο στειροποίησης δεδομένων το οποίο συμπιέζει τα δεδομένα και τα κωδικοποιεί σε δυαδική μορφή, καθιστώντας τα δύσκολα να διαβαστούν από ένα άνθρωπο αλλά γρήγορα στη μεταφορά. Τα REST API ανταλλάζουν πληροφορίες σε μορφή JSON, μια μορφή κειμένου η οποία είναι πιο ευανάγνωστη από τον άνθρωπο αλλά δεν πετυχαίνει τον ίδιο βαθμό συμπίεσης με τα protobuf.

4. Υποστήριξη Ροής:

   Το gRPC υποστηρίζει διμερή ροή (bidirectional streaming), επιτρέποντας και στον πελάτη και στον διακομιστή να στείλουν πολλαπλά μηνύματα κατά τη διάρκεια μίας μόνο σύνδεσης. Αυτή η λειτουργία είναι ωφέλιμη για εφαρμογές πραγματικού χρόνου ή σενάρια όπου οι πληροφορίες πρέπει να ανταλλάσσονται συνεχώς. Το REST είναι βασισμένο σε αίτημα-απάντηση και δεν υποστηρίζει εγγενώς ροή δεδομένων (streaming), αν και μπορεί να χρησιμοποιηθεί συνεχείς σάρωση (long polling) για να επιτευχθεί παρόμοια λειτουργικότητα.

5. Διαχείριση Σύνδεσης:

   Το gRPC χρησιμοποιεί μία μοναδική, μακροχρόνια σύνδεση με το HTTP/2, επιτρέποντας την πολυπλεξία πολλαπλών αιτήσεων και απαντήσεων πάνω στην ίδια σύνδεση. Αυτή η λειτουργία μειώνει την καθυστέρηση και βελτιώνει την απόδοση, ιδίως σε περιβάλλοντα δικτύου υψηλής καθυστέρησης. Το REST, χρησιμοποιώντας το HTTP/1.1, δημιουργεί μια νέα σύνδεση για κάθε κύκλο αίτημα-απάντηση, οδηγώντας σε αυξημένη καθυστέρηση και επιβάρυνση.

6. Υποστήριξη Γλώσσας Προγραμματισμού:

   Το gRPC υποστηρίζεται από ποικιλία γλωσσών προγραμματισμού με επίσημες βιβλιοθήκες που παρέχονται από την Google, ενώ το REST είναι ένα πρωτόκολλο που βασίζεται στο πρότυπο HTTP και υποστηρίζεται από σχεδόν κάθε γλώσσα προγραμματισμού.

7. Εργαλεία και Οικοσύστημα:

   Το REST έχει ευρύτερη υιοθέτηση και έχει ένα ώριμο οικοσύστημα με πλούσιο σύνολο εργαλείων και βιβλιοθηκών διαθέσιμο για να υποστηρίξει την ανάπτυξη και τη συντήρησή του. Το gRPC, παρόλο που κερδίζει έδαφος, έχει ένα μικρότερο οικοσύστημα και σχετικά λιγότερα εργαλεία και βιβλιοθήκες διαθέσιμα.

Συνοψίζοντας, το REST API είναι μια απλή, ευέλικτη και ευρέως χρησιμοποιούμενη προσέγγιση για τη δημιουργία διαδικτυακών υπηρεσιών, ενώ το gRPC είναι ένα πιο σύγχρονο, αποτελεσματικό και αποδοτικό πλαίσιο επικοινωνίας που καταλήγει σε υψηλότερη απόδοση και χαμηλότερη καθυστέρηση, καθιστώντας το πιο κατάλληλο για εφαρμογές με αυξημένες απαιτήσεις σε επικοινωνία και απόδοση, όπως οι μικρουπηρεσίες και οι πραγματικού χρόνου εφαρμογές.

	gRPC	REST
Πρωτόκολλο	HTTP/2	HTTP/1.1
Μορφή δεδομένων	Protocol Buffers	JSON
Σχεδιασμός API	Remote Procedure Calls	HTTP Put, Post, Get Delete
Υποστήριξη Ροής	Ναι	Όχι
Διαχείριση Σύνδεσης	Μια μακροχρόνια σύνδεση	Νέα σύνδεση για κάθε κύκλο
Υποστήριξη Γλώσσας Προγραμματισμού	Υποστηρίζεται από πολλές γλώσσες	Υποστηρίζεται από όλες τις γλώσσες
Εργαλεία και Οικοσύστημα	Μικρό οικοσύστημα	Ώριμο οικοσύστημα

Πίνακας 2.2 gRPC και REST APIs

2.4 Οικοσύστημα Docker

Το Docker δημιουργήθηκε για να απλοποιήσει τη δημιουργία, εγκατάσταση και εκτέλεση εφαρμογών με τη χρήση δοχείων (containers). Η εικονοποίηση επιτρέπει στο χρήστη να τρέχει εφαρμογές σε ένα εικονικό περιβάλλον, συσκευάζοντας όλα τα απαραίτητα στοιχεία μιας εφαρμογής, όπως αρχεία και περιβάλλον εκτέλεσης, μαζί.

Αυτά τα συσκευασμένα αρχεία ονομάζονται εικόνες (images). Μία εικόνα Docker περιέχει την εφαρμογή και το περιβάλλον εκτέλεσης της και διανέμεται μέσω του αποθετηρίου Docker (Docker Registry). Αυτή η προσέγγιση επιτρέπει την εύκολη διασύνδεση λογισμικών, καθιστώντας την ιδανική για μια εφαρμογή βασισμένη στην αρχιτεκτονική μικρουπηρεσιών.

Το Docker μας επιτρέπει να ξεπεράσουμε τεχνικά προβλήματα που παρουσιάζονταν από προηγούμενες τεχνολογικές λύσεις. Αρχικά, μπορούμε να διαχειριστούμε τις εξαρτήσεις (dependencies) μιας εφαρμογής, καθώς το περιβάλλον εκτέλεσης είναι ήδη συσκευασμένο με την εφαρμογή και όλες οι απαραίτητες εξαρτήσεις είναι ήδη εγκατεστημένες. Ταυτόχρονα, κερδίζουμε χρόνο από την εγκατάσταση και παραμετροποίηση των προ-απαιτούμενων μιας εφαρμογής, αφού με το docker οι απαιτήσεις αυτές πρέπει να δηλωθούν μόνο μια φορά και μετά εγκαθίστανται αυτόματα.
Τέλος, η ανάπτυξη (deployment) των εικόνων docker απαιτεί πιο λίγους πόρους από πιο παραδοσιακούς τρόπους όπως οι εικονικές μηχανές (Virtual Machines).

2.4.1 Εικόνες και Δοχεία Docker

Μια εικόνα Docker είναι μια αρχειοθετημένη και ανεξάρτητη μονάδα λογισμικού που περιλαμβάνει όλα τα απαραίτητα στοιχεία για τη λειτουργία ενός προγράμματος, όπως το λειτουργικό σύστημα, τις βιβλιοθήκες, τις εξαρτήσεις και το ίδιο το λογισμικό. Οι εικόνες Docker δημιουργούνται από ένα αρχείο Dockerfile, το οποίο περιγράφει τα βήματα και τις εντολές που πρέπει να εκτελεστούν για τη δημιουργία της εικόνας. Τα αρχεία Dockerfile είναι απλά αρχεία κειμένου και έχουν τη δομή που παρουσιάζεται στη Εικόνα 2.3.

```
FROM ubuntu                          # Start with Ubuntu Linux
RUN apt-get update                   # Update package sources list
RUN apt-get install nginx -y         # Install nginx web server
COPY index.html /var/www/html        # Copy static site files inside the container
EXPOSE 80                            # Open TCP port 80
CMD ['nginx', '-g', 'daemon off']    # Run the web server application
```

Εικόνα 2.3: Δομή αρχείου Dockerfile

Αρχικά ορίζεται μια υπάρχουσα εικόνα Docker που θα χρησιμοποιηθεί σαν βάση, με την εντολή FROM. Οι εικόνες βάσης περιέχουν ένα λειτουργικό σύστημα και προ-εγκατεστημένο λογισμικό και χρησιμοποιούνται σαν θεμέλιο για τη δημιουργία νέων εικόνων. Στη συνέχεια ορίζονται εντολές που θέλουμε να εκτελεστούν κατά τη διάρκεια κατασκευής της εικόνας με την εντολή RUN. Για παράδειγμα, ορίζεται η εγκατάσταση εξαρτήσεων και πακέτων που απαιτούνται από την εφαρμογή. Η εντολή COPY, αντιγράφει αρχεία και καταλόγους όπως τον κώδικα της εφαρμογής, από τον υπολογιστή μας στην αντίστοιχη θέση τους στην εικόνα. Με την εντολή EXPOSE ορίζουμε τη θύρα που θα χρησιμοποιηθεί από την εφαρμογή για επικοινωνία με το εξωτερικό περιβάλλον. Τέλος, με την εντολή CMD ορίζουμε τις εντολές που θα εκτελεστούν όταν ξεκινήσει το δοχείο.

Οι εικόνες Docker μπορούν να αποθηκευτούν σε αποθετήρια όπως το Docker Hub ή σε ιδιωτικά αποθετήρια, για να καταστούν εύκολα διαθέσιμες για κοινή χρήση και ανακατανομή.

Ένα δοχείο Docker είναι μια εκτελέσιμη και ενεργή έκδοση μιας εικόνας Docker. Όταν μια εικόνα Docker εκτελείται στο Docker Engine, δημιουργείται ένα νέο δοχείο. Τα δοχεία παρέχουν ένα απομονωμένο και ελαφρύ περιβάλλον για την εκτέλεση των εφαρμογών, καθιστώντας το πιο εύκολο να διαχειριστείτε τις εξαρτήσεις και να εξασφαλίσετε ότι η εφαρμογή λειτουργεί όπως αναμενόταν σε διαφορετικά περιβάλλοντα [Swani2017]. Τα δοχεία Docker μπορούν να εκτελεστούν, να τροποποιηθούν, να σταματήσουν και να διαγραφούν ανεξάρτητα από άλλα δοχεία ή το υποκείμενο λειτουργικό σύστημα.

Η βασική διαφορά μεταξύ μιας εικόνας Docker και ενός δοχείου Docker είναι ότι η εικόνα είναι ένα αρχειοθετημένο και ανεξάρτητο σύνολο αρχείων που περιγράφει την εφαρμογή και τις εξαρτήσεις της, ενώ το δοχείο είναι μια ενεργή και εκτελέσιμη υπόσταση αυτής της εικόνας. Μια εικόνα Docker αποτελεί το πρότυπο από το οποίο δημιουργούνται τα δοχεία, ενώ τα δοχεία είναι τα πραγματικά εκτελούμενα στιγμιότυπα των εφαρμογών που βασίζονται σε αυτό το πρότυπο.

2.4.2 Docker Engine

Το Docker Engine είναι η προεπιλεγμένη πλατφόρμα εκτέλεσης δοχείων Docker και λειτουργεί σε διάφορα λειτουργικά συστήματα Linux και Windows.
Το Docker δημιουργεί απλά εργαλεία και μια καθολική προσέγγιση συσκευασίας που συμπεριλαμβάνει όλες τις εξαρτήσεις μιας εφαρμογής μέσα σε ένα δοχείο (container), το οποίο στη συνέχεια εκτελείται στο Docker Engine. Το Docker Engine επιτρέπει στις εφαρμογές που είναι σε δοχείο (containerized) να εκτελούνται ομοιόμορφα οπουδήποτε, σε οποιαδήποτε υποδομή, λύνοντας το πρόβλημα των εξαρτήσεων.

2.4.3 Docker Compose

Το Docker Compose είναι ένα εργαλείο που αναπτύχθηκε για να βοηθήσει στον καθορισμό και την κοινή χρήση εφαρμογών που αποτελούνται από πολλαπλά δοχεία (containers). Με το Compose, μπορούμε να δημιουργήσουμε ένα αρχείο YAML για να καθορίσουμε τις υπηρεσίες και με μια μόνο εντολή, μπορούμε να εκκινήσουμε όλα τα δοχεία ή να τα καταργήσουμε. Η δομή του αρχείου αυτού είναι τυποποιημένη. Αρχικά ορίζουμε την έκδοση του Compose που θα χρησιμοποιηθεί. Στη συνέχεια ορίζουμε τις υπηρεσίες της εφαρμογής, Για κάθε υπηρεσία ορίζουμε την εικόνα που θα χρησιμοποιηθεί, τις θύρες που θα εκτίθενται από το δοχείο, τοίχων τοπικούς φακέλους που θέλουμε να προσαρτήσουμε στο δοχείο και τέλος μεταβλητές περιβάλλοντος (αν υπάρχουν). Παράδειγμα ενός τέτοιου αρχείου παρουσιάζεται στην Εικόνα 2.4.

```
version: “3”.        # Specify docker-compose version
services:            # Define the services/containers to be run
  my-application:    # Name of the first service
    build: ./        # Specify the directory of the Dockerfile
    ports:
      - “80:8080”    # Assign container port (80) to given host port (8080)
    environment:
      - CONF
  db:                # Name of the second service
    image: postgres
  redis:             # Name of the third service
    image: redis
    command: redis-server –sage “” --appendonly no
    ports:
      - “6379”       # Assign container port (6379) to random host port
```

Εικόνα 2.4: Δομή αρχείου docker-copose.yml

Όπως φαίνεται στην Εικόνα 2.4, στη πρώτη γραμμή του αρχείου docker-compose.yml ορίζεται η έκδοση της σύνταξης του docker-compose που θα χρησιμοποιηθεί. Στη συνέχεια, διακρίνεται η ενότητα υπηρεσιών (services). Οι ενότητες στο αρχείο ξεχωρίζουν με τη χρήση εσοχής. Στο παράδειγμα ορίζονται τρεις υπηρεσίες, my-application, db και redis. Για κάθε υπηρεσία ορίζεται αρχικά το όνομα της και στη συνέχεια, με τη χρήση νέας εσοχής, ορίζονται κάποιες παράμετροι. Η παράμετρος build καθορίζει τον κατάλογο στον οποίο βρίσκεται το Dockerfile για τη κατασκευή της εικόνας. Στη περίπτωση που χρησιμοποιείται μια προ-κατασκευασμένη εικόνα, μπορεί να χρησιμοποιηθεί η παράμετρος image στη θέση του build, η οποία ορίζει την εικόνα Docker που θα χρησιμοποιηθεί. Η παράμετρος ports, ορίζει τη σύνδεση θυρών μεταξύ του δοχείου και του διακομιστή, η παράμετρος environment ορίζει τις μεταβλητές περιβάλλοντος που θα χρησιμοποιηθούν στο δοχείο και τέλος, η παράμετρος command ορίζει κάποια εντολή που θα εκτελεστεί κατά την εκκίνηση του δοχείου.

Το μεγάλο πλεονέκτημα της χρήσης του Compose είναι ότι επιτρέπει τον καθορισμό της στοίβας μιας εφαρμογής σε ένα αρχείο, το οποίο μπορεί να διατηρηθεί στον κορμό του αποθετηρίου του έργου υπό έλεγχο εκδόσεων και να διευκολύνει τη συνεισφορά σε ένα έργο. Με την κλωνοποίηση ενός αποθετηρίου πλέον η εφαρμογή μπορεί να εκτελεστεί με τη χρήση μιας εντολής.

2.4.4 Docker-swarm

Ένα σμήνος (swarm) αποτελείται από πολλαπλούς υπολογιστές υποδοχής Docker (Docker hosts) που τρέχουν σε λειτουργεία σμήνους (swarm mode). Αυτοί οι υπολογιστές χωρίζονται σε διαχειριστές και εργάτες. Οι διαχειριστές αναθέτουν εργασίες στου εργάτες οι οποίοι εκτελούν υπηρεσίες. Παράλληλα οι διαχειριστές διαχειρίζονται τη συμμετοχή άλλων κόμβων στο σμήνος. Κάθε κόμβος μπορεί να είναι διαχειριστής, εργάτης ή να εκτελεί και τους δυο ρόλους.

Κατά τη δημιουργία μιας υπηρεσίας καθορίζεται η βέλτιστη κατάσταση της (αριθμός αντιγράφων, πόρους δικτύου και αποθήκευσης, θύρες η υπηρεσία εκθέτει στον εξωτερικό κόσμο και άλλα). Το Docker εργάζεται για τη διατήρηση αυτής της επιθυμητής, βέλτιστης κατάστασης. Για παράδειγμα, αν ένας κόμβος εργάτης (worker node) δεν είναι διαθέσιμος, το Docker προγραμματίζει τις εργασίες του κόμβου αυτού σε άλλους κόμβους. Μια εργασία θεωρείται ένα τρέχον δοχείο που είναι μέρος μιας υπηρεσίας σμήνους και διαχειρίζεται από ένα διαχειριστή, σε αντίθεση με ένα αυτόνομο δοχείο.

Ένα από τα κύρια πλεονεκτήματα των υπηρεσιών σμήνους σε σύγκριση με τα αυτόνομα δοχεία είναι ότι μπορούν να τροποποιηθούν οι παράμετροι της υπηρεσίας (configuration), συμπεριλαμβανομένου και του δικτύου και των όγκων μνήμης που είναι συνδεδεμένοι σε αυτή, χωρίς την ανάγκη να γίνει χειροκίνητη επανεκκίνηση της υπηρεσίας. Το Docker θα ενημερώσει τις παραμέτρους, θα σταματήσει τις εργασίες της υπηρεσίας με τις παλιές παραμέτρους και θα δημιουργήσει νέες εργασίες που ταιριάζουν με την νέα παραμετροποίηση.

Οι στοίβες υπηρεσιών σμήνους μπορούν να οριστούν και να τρέξουν με τον ίδιο τρόπο που μπορεί να χρησιμοποιηθεί το Docker Compose για να οριστούν και να τρέξουν δοχεία.

2.5 Σχετικές Εργασίες

Σκοπός αυτής της εργασίας είναι να παρουσιάσει μια σύγκριση μεταξύ της αρχιτεκτονικής των μικρουπηρεσιών καθώς και της μονολιθικής προσέγγισης για ανάπτυξη ενός συστήματος. Καθώς η αρχιτεκτονική των μικρουπηρεσιών έχει αρχίσει να γίνεται αρκετά δημοφιλής τα τελευταία χρόνια, έχουν αναπτυχθεί και αρκετά μετροπρογράμματα που εξετάζουν τις διαφορές μεταξύ των δυο αρχιτεκτονικών. Μετροπρόγραμμα μπορεί να οριστεί ως ένα λογισμικό το οποίο εκτελείται για να μετρηθεί η επίδοση ενός υπολογιστικού περιβάλλοντος.

Στη συνέχεια αυτού του κεφαλαίου παρουσιάζονται μερικά από τα υπάρχων μετροπρογράμματα για εφαρμογές στην αρχιτεκτονική των μικρουπηρεσιών. Παρόλο που αυτά τα μετροπρογράμματα παρουσιάζουν τις διαφορές των δυο αρχιτεκτονικών, δεν υπάρχει διαθέσιμη έρευνα που να παρέχει τον πυγαίο κώδικα μιας εφαρμογής και στις δυο αρχιτεκτονικές. Η διάθεση της μονολιθικής υλοποίησης ενός μετροπρογράμματος μαζί με την αντίστοιχη υλοποίηση του ίδιου μετροπρογράμματος στην αρχιτεκτονική των μικρουπηρεσιών, βοηθά στην σύγκριση μεταξύ των δυο αρχιτεκτονικών από τους ενδιαφερόμενους.

Ταυτόχρονα, τα μετροπρογράμματα που αναφέρονται πιο κάτω είναι αρκετά σύνθετα με μεγάλο όγκο πηγαίου κώδικα. Η ανάπτυξη μιας σχετικά απλής εφαρμογής στην αρχιτεκτονική των μικρουπηρεσιών με την αντίστοιχη μονολιθική υλοποίηση της, μπορεί να χρησιμοποιηθεί σε ένα προπτυχιακό μάθημα το οποίο εστιάζει στη παρουσίαση των μικρουπηρεσιών, προσφέροντας μία βάση για τους φοιτητές έτσι ώστε να εξοικειωθούν με το προγραμματιστικό μοντέλο των μικρουπηρεσιών.

2.5.1 DeathStarBench

Το DeathStarBench [Delimitrou, Gan2019] αποτελεί μια σουίτα προγραμμάτων συγκριτικής αξιολόγησης εφαρμογών νέφους που είναι βασισμένες στο αρχιτεκτονικό μοτίβο μικρουπηρεσιών το οποίο αναπτύχθηκε στα πλαίσια μιας έρευνας η οποία στόχευε στη μελέτη των δυσκολιών που μπορεί να προκύψουν κατά την υλοποίηση μικρουπηρεσιών σε διαφορετικά υπολογιστικά περιβάλλοντα.

Στη σουίτα προγραμμάτων του DeathStarBench έχουν υλοποιηθεί τρεις ξεχωριστές υπηρεσίες. Οι υπηρεσίες αυτές είναι υπηρεσία κοινωνικής δικτύωσης (Social Network), υπηρεσία πολυμέσων (Media Service) και υπηρεσία κρατήσεων σε ξενοδοχεία (Hotel Reservation).

Πέραν των πιο πάνω διαθέσιμων υπηρεσιών, η σουίτα του DeathDtarBench εξακολουθεί να αναπτύσσεται. Κατά τη περίοδο αυτής της έρευνας υπάρχουν ακόμη τρεις υπηρεσίες οι οποίες υλοποιούνται με σκοπό να προστεθούν στη σουίτα υπηρεσιών. Οι νέες υπηρεσίες είναι ηλεκτρονικό εμπόριο (e-commerce), τραπεζικό σύστημα (Banking System) και σύστημα για συγχρονισμό αυτόματων αεροσκαφών (Drone Coordination System).

Σην έρευνα που συνόδευσε την υλοποίηση του DeathStarBench έχουν μελετηθεί οι επιπτώσεις που έχει η χρίση των μικρουπηρεσιών σε όλη τη στοίβα συστήματος νέφους (cloud system stack), από τον σχεδιασμό διακομιστή σε ένα κέντρο δεδομένων μέχρι τα επιπρόσθετα έξοδα του λειτουργικού συστήματος και του δικτύου. Παράλληλα, μετρήθηκαν οι επιδώσεις κλίμακας των μικρουπηρεσιών καθώς η συστάδα υπολογιστών (cluster) μεγαλώνει σε μέγεθος και οι υπηρεσίες γίνονται πιο πολύπλοκες. Μέσα από αυτή την έρευνα έχει προκύψει ότι οι μικρουπηρεσίες ασκούν αυξημένη πίεση στην καθυστέρηση ουράς (tail latency) καθώς και στην προβλεψιμότητα της απόδοσης [Gan2019] .

2.5.2 Train Ticket: A Benchmark Microservice System

Το Train Ticket [FudanSELab, Zhou2019] αποτελεί ένα σύστημα για κλείσιμο εισιτηρίων για τρένα, το οποίο βασίζεται στην αρχιτεκτονική των μικρουπηρεσιών και αποτελείται από 41 μικρουπηρεσίες. Οι υπηρεσίες αυτές είναι ανεπτυγμένες σε διάφορες γλώσσες προγραμματισμού (Java – Spring Boot, Spring Cloud, Node.js – Express, Python – Django, Go – Webgo) και χρησιμοποιούνται δυο διαφορετικές βάσεις δεδομένων (MongoDB και MySQL).

Το σύστημα αυτό έχει αναπτυχθεί σαν μέρος μιας έρευνας η οποία επικεντρώνεται στην ανάλυση σφαλμάτων και την αποσφαλμάτωσης συστημάτων στην αρχιτεκτονική των μικρουπηρεσιών.

Τα αποτελέσματα της μελέτης δείχνουν ότι οι περισσότερες περιπτώσεις σφαλμάτων, εκτός από αυτές που προκαλούνται από περιβαλλοντικές ρυθμίσεις, μπορούν να επωφεληθούν από την οπτικοποίηση των αποτελεσμάτων, ειδικά στις περιπτώσεις που σχετίζονται με τις αλληλεπιδράσεις των μικρουπηρεσιών. Θεωρώντας τις μικρουπηρεσίες ή τις καταστάσεις των μικρουπηρεσιών ως κόμβους, μπορούμε να βελτιώσουμε περαιτέρω την αποτελεσματικότητα της αποσφαλμάτωσης των μικρουπηρεσιών χρησιμοποιώντας εργαλεία οπτικοποίησης κορυφαίας τεχνολογίας για διανεμημένα συστήματα [Zhou2019].

2.5.3 MicroSuite

Το MicroSuite [Wenischlab, Sriraman2018] αποτελεί μια σειρά από διαδικτυακές υπηρεσίες δεδομένων (Online, Data-Intensive Services – OLDI) που αποτελούνται από τρία επίπεδα, εμπρόσθιο (front-tier), ενδιάμεσο (mid-tier) και ακραίο (leaf).

Το MicroSuite περιλαμβάνει τέσσερις υπηρεσίες OLDI που ενσωματώνουν λογισμικό ανοιχτού κώδικα. Η πρώτη είναι μια υπηρεσία αναζήτησης υψηλών διαστάσεων, βασισμένη στις ομοιότητες εικόνων (HDSearch). Τη δεύτερη υπηρεσία αποτελεί ένας δρομολογητής βασισμένος στην κλιμάκωση αποθηκεύσεων ζευγών κλειδιού-τιμής (key-value stores), ανθεκτικός σε σφάλματα (Router). Τρίτη είναι μια υπηρεσία για την εκτέλεση αλγεβρικών συνόλων σε λίστες αποστολής για την ανάκτηση εγγράφων (Set Algebra). Τέλος, περιέχει ένα σύστημα προτάσεων αντικειμένων, βασισμένο στον χρήστη, για την πρόβλεψη αξιολογήσεων των χρηστών (Recommend).

Το MicroSuite γράφτηκε αρχικά για την αξιολόγηση των επιπτώσεων που αντιμετωπίζουν οι μικρουπηρεσίες λόγο του λειτουργικού συστήματος και του δικτύου. Μέσα από αυτή τη μελέτη έχει αναδειχθεί ότι ο διαχειριστής εργασιών του λειτουργικού συστήματος (OS sheduler) μπορεί να επηρεάσει τη καθυστέρηση ουράς (tail latency) μιας μικρουπηρεσίες μέχρι και 87% [Sriraman2018].

2.5.4 vSwarm – Serverless Benchmarking Suite

Το vSwarm [vhive-serverless, Schall2022] είναι μια συλλογή από μετροπρογράμματα (benchmarks) ανεξάρτητα από υποδομή διακομιστών (serverless) έτοιμα για εκτέλεση. Το κάθε μετροπρόγραμμα αποτελείται από ένα αριθμό διασυνδεδεμένων συναρτήσεων και έχουν σαν γενικό επίκεντρο το φόρτος εργασίας από μεγάλους όγκους δεδομένων. Σε συνδυασμό με τα μετροπρογράμματα, η σουίτα του vSwarm περιέχει ένα σύνολο από ανεξάρτητες συναρτήσεις, που υποστηρίζουν x86 και arm64 αρχιτεκτονικές.

Η σουίτα αποτελείται από 10 μετροπρογράμματα και 25 ανεξάρτητες συναρτήσεις. Μεταξύ άλλων, περιλαμβάνει υλοποίηση του αλγορίθμου MapReduce στη γλώσσα προγραμματισμού golang, αναγνώριση αντικειμένων σε βίντεο σε πραγματικό χρόνο στις γλώσσες προγραμματισμού golang και python και υλοποιήσεις διάφορων αλγορίθμων όπως AES και Fibonacci.

Η σουίτα του vSwarm έχει χρησιμοποιηθεί σε μια μελέτη η οποία εστιάζει στις συναρτήσεις χωρίς διακομιστή (serverless functions) και πως μπορούν να βελτιστοποιηθούν. Στη μελέτη αυτή έχει ταυτοποιηθεί η ανάγκη να διατηρούνται οι συχνά χρησιμοποιημένες συναρτήσεις σε μια κατάσταση αναμονής στη προσωρινή μνήμη, για να αποφεύγονται μεγάλες καθυστερήσεις από την εκκίνηση μιας συνάρτησης (cold start delays). Παρατηρήθηκε ότι υπάρχει μεγάλος βαθμός διασύνδεσης μεταξύ των συναρτήσεων που βρίσκονται σε αναμονή σε ένα διακομιστή και ότι οι συναρτήσεις αυτές καλούνται σχετικά αραιά. Η ανάλυση απόδοσης σε ένα πραγματικό διακομιστή έχει δήξει ότι το εμπρόσθιο άκρο (front-end) αποτελεί σημείο φραγμού (bottleneck) για τις συναρτήσεις που βρίσκονται σε αναμονή, καθώς πρέπει να ανακτήσουν πληροφορίες από την κύρια μνήμη [Schall2022].

2.5.5 Alibaba Trace Analysis

Η έρευνα της Alibaba [Shutian2021] αποτελεί μια λεπτομερή μελέτη μεγάλης κλίμακας για εγκαταστάσεις που εξυπηρετούν εφαρμογές μικρουπηρεσιών στις συστάδες (clusters) της Alibaba. Η έρευνα αυτή στοχεύει στην καλύτερη κατανόηση των μικρουπηρεσιών, εξετάζοντας τα χαρακτηριστικά τους και την απόδοση τους κατά τη διάρκεια της λειτουργίας των εφαρμογών.

Συγκεκριμένα, παρατηρείται ότι τα γραφήματα κλήσεων των μικρουπηρεσιών κατανέμονται με βαριά ουρά (heavy tail distribution) και η τοπολογία τους μοιάζει με δέντρο, με πολλές μικρουπηρεσίες να εξυπηρετούν μεγάλο όγκο κλήσεων και να αποτελούν καυτά σημεία (hot spots) . Επιπλέον, αποκαλύπτονται τρείς τύποι σημαντικής εξάρτησης κλήσεων που μπορούν να χρησιμοποιηθούν για τη βελτιστοποίηση του σχεδιασμού μικρουπηρεσιών.

Τέλος, η μελέτη συμπεραίνει ότι οι περισσότερες μικρουπηρεσίες είναι πιο ευαίσθητες στην παρέμβαση της κεντρικής μονάδας επεξεργασίας (CPU) παρά στην παρέμβαση της μνήμης, πράγμα που είναι σημαντικό για την απλούστευση του προβλήματος χρονοπρογραμματισμού και την καλύτερη διαχείριση των πόρων της συστάδας (cluster). Ως μέρος των συμπερασμάτων, υπάρχουν επίσης προτάσεις για μελλοντικές κατευθύνσεις έρευνας στον τομέα.

Κεφάλαιο 3

Σχεδίαση

3.1 Η Εφαρμογή FoodTrucks

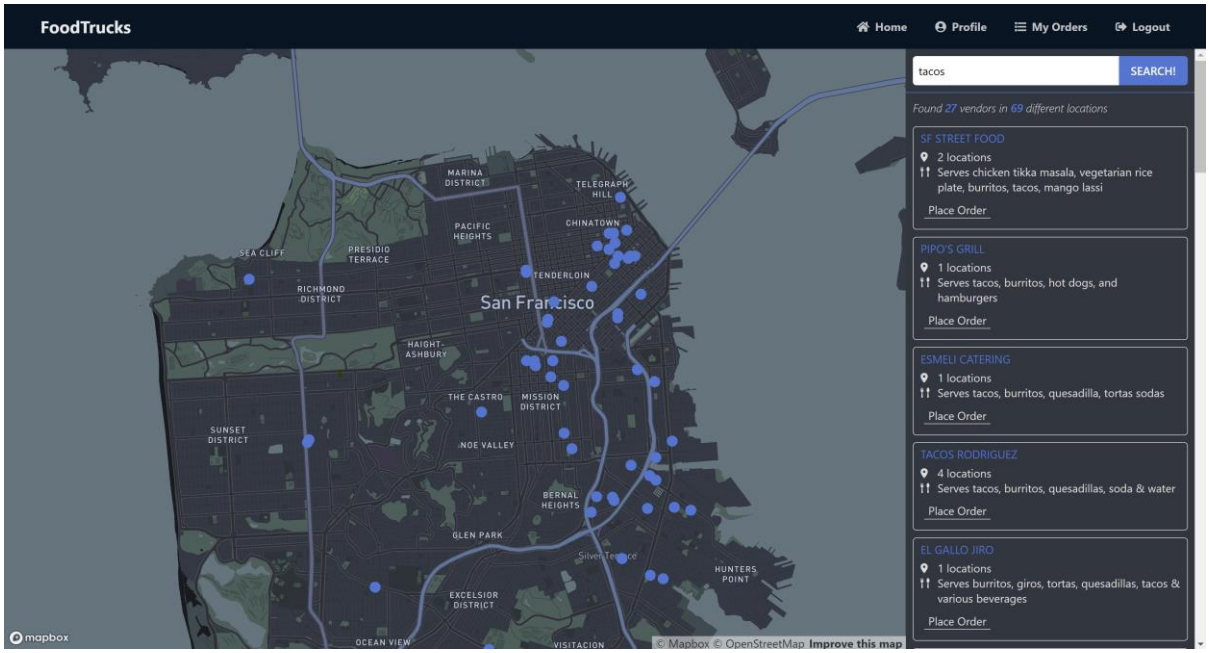
Στο προηγούμενο κεφάλαιο παρουσιάστηκε η θεωρητική σκοπιά της αρχιτεκτονικής των μικρουπηρεσιών, της μονολιθικής προσέγγισης και άλλων σχετικών τεχνολογιών καθώς και αποτελέσματα σχετικών ερευνών.

Σε αυτό το κεφάλαιο θα αναπτυχθεί η σχεδίαση της εφαρμογής FoodTrucks, η οποία έχει αναπτυχθεί σαν μονολιθική εφαρμογή αλλά και στην αρχιτεκτονική των μικρουπηρεσιών. Σκοπός της ανάπτυξης της εφαρμογής είναι η συγκριτική αξιολόγηση των δυο αρχιτεκτονικών (μικρουπηρεσίες και μονολιθικό).

Η εφαρμογή FoodTrucks έχει βασιστεί σε μια μονολιθική εφαρμογή ανοιχτού πηγαίου κώδικα, SF Food Trucks [prakhar1989]. Η εφαρμογή αυτή αξιοποιεί τις ακόλουθες τεχνολογίες: για τον σχεδιασμό της διεπαφής (front-end) της εφαρμογής έχει χρησιμοποιηθεί React. Η React είναι πλαίσιο (framework) για τη γλώσσα προγραμματισμού javascript. Στην αρχική οθόνη της εφαρμογής φορτώνεται ένας χάρτης, από το Mapbox της OpenStreetMap. Στη δεξιά μεριά της οθόνης φορτώνεται μια μπάρα αναζήτησης, όπου ο χρήστης μπορεί να αναζητήσει φαγητά. Με την επιτυχημένη αναζήτηση, εμφανίζονται στον χάρτη οι τοποθεσίες των κινητών καντινών που διαθέτουν το συγκεκριμένο προϊόν, καθώς και ένα συνοπτικό αποτέλεσμα όλων των καντινών κάτω από την μπάρα αναζήτησης, όπως φαίνεται στην Εικόνα 3.1.

Για το υποστηρικτικό σύστημα (back-end) της εφαρμογής έχει χρησιμοποιηθεί Flask. Η Flask είναι μια ελαφριά και ευέλικτη βιβλιοθήκη της Python που χρησιμοποιείται για τη δημιουργία εφαρμογών ιστού. Χαρακτηρίζεται από την απλότητα, την ευκολία χρήσης και τη μικρή καμπύλη μάθησης που προσφέρει. Η Flask χρησιμοποιεί το μοντέλο WSGI (Web Server Gateway Interface) για την επικοινωνία μεταξύ της εφαρμογής ιστού και του διακομιστή, επιτρέποντας τον καθορισμό των διαδρομών URL και των συναρτήσεων που θα κληθούν όταν ένα αίτημα φτάσει στο URL. Παράλληλα, υποστηρίζει τη χρήση προτύπων (templates) για τη δυναμική δημιουργία HTML και τη διαχείριση της παρουσίασης των δεδομένων στον πελάτη. Για τον χειρισμό των προτύπων αυτών χρησιμοποιεί τη βιβλιοθήκη Jinja2, η οποία επιτρέπει τη δυναμική μεταφορά δεδομένων στην HTML. Η Flask είναι κατάλληλη για την κατασκευή εφαρμογών ιστού μικρού ή μεσαίου μεγέθους και παρέχει μια απλή και ευέλικτη βάση για την ανάπτυξη. Μπορεί να χρησιμοποιηθεί σε συνδυασμό με την αρχιτεκτονική μικρουπηρεσιών για τη δημιουργία εφαρμογών που είναι εύκολο να κλιμακωθούν και να συντηρηθούν.

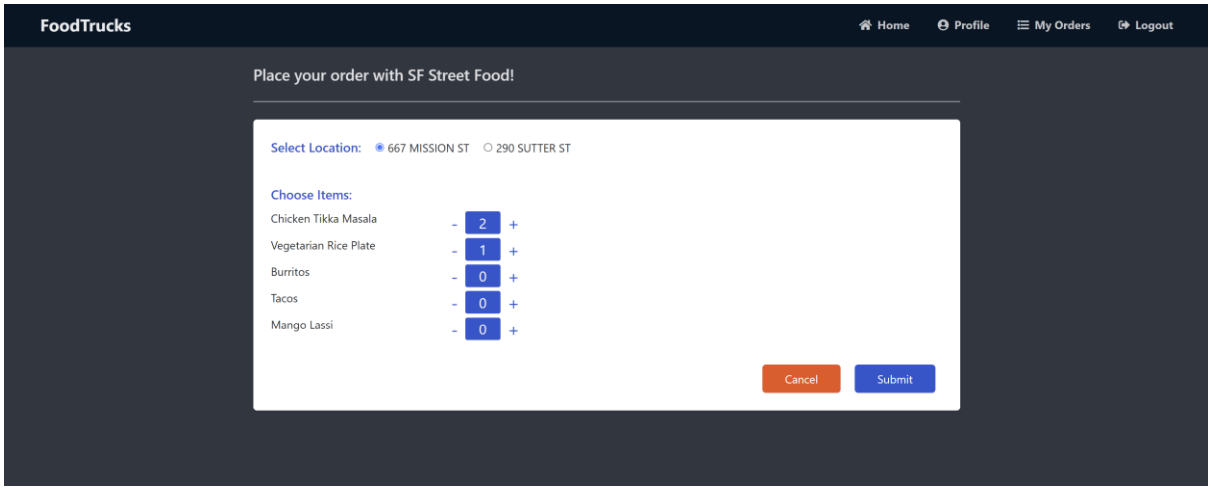
Η αναζήτηση, γίνεται με τη βοήθεια του Elasticsearch. Το Elasticsearch είναι μια υπηρεσία αναζήτησης (search engine) ανοιχτού κώδικα, στην οποίο φορτώνεται ένα σύνολο δεδομένων με πληροφορίες για τα διαθέσιμα φορτηγά. Τα δεδομένα αυτά αποθηκεύονται σε μορφή JSON και οργανώνονται σε ευρετήρια (indices). Το Elasticsearch παρέχει το δικό του API, μέσα από το οποίο η εφαρμογή Foodtrucks μπορεί να αναζητήσει το συγκεκριμένο ευρετήριο στο οποίο αποθηκευτήκαν τα δεδομένα για τις κινητές καντίνες, για το κείμενο αναζήτησης που εισάγει ο χρήστης. Συγκεκριμένα, η κάθε καντίνα περιέχει ένα πεδίο με τα φαγητά που προσφέρει. Αν αυτό το πεδίο περιέχει το αντικείμενο που αναζητά ο χρήστης, τότε ολόκληρο το αντικείμενο της καντίνας επιστρέφεται από το Elasticsearch σαν αποτέλεσμα. Η δυνατότητα κατανεμημένης αποθήκευσης και αναζήτησης εξασφαλίζει υψηλή διαθεσιμότητα και αντοχή σε σφάλματα, καθιστώντας το Elasticsearch ιδανικό για αναζήτηση κειμένου.



Εικόνα 3.1: Αρχική οθόνη εφαρμογής με λειτουργία αναζήτησης.

Η εφαρμογή αυτή έχει δώσει τη βάση για την εφαρμογή που χρησιμοποιήθηκε σε αυτή την έρευνα. Η λειτουργικότητα που μας προσφέρει είναι αρκετά περιορισμένη και για αυτό τον λόγο χρειάστηκε να προστεθούν ακόμη μερικές λειτουργίες.

Η εφαρμογή έχει επεκταθεί για να μπορεί να εξυπηρετεί παραγγελίες. Αρχικά, προστέθηκε στη εφαρμογή ταυτοποίηση χρηστών (user authentication) για να μπορεί να υποστηρίζει ξεχωριστούς λογαριασμούς. Η υποστήριξη λογαριασμών χριστών (user accounts), επιτρέπει στην υλοποίηση δυο νέων λειτουργειών. Η πρώτη είναι η τοποθέτηση παραγγελίας. Ο κάθε χρήστης μπορεί να συνδεθεί στον λογαριασμό του, να αναζητήσει ένα προϊόν και να τοποθετήσει μια παραγγελία στην επιθυμητή κινητή καντίνα, όπως φαίνεται στην Εικόνα 3.2. Η δεύτερη λειτουργεία που υλοποιήθηκε είναι η παρακολούθηση και ολοκλήρωση των παραγγελιών από τον ιδιοκτήτη της καντίνας, ο οποίος μπορεί να συνδεθεί στην εφαρμογή, να δει τις παραγγελίες που εκκρεμούν και να επισημάνει ως ολοκληρωμένες αυτές που έχουν παραδοθεί.



Εικόνα 3.2: Αρχική οθόνη εφαρμογής με λειτουργία αναζήτησης.

Για να μπορεί η εφαρμογή να υποστηρίζει τη λειτουργικότητα των παραγγελιών, χρειάστηκε να προστεθεί στην τεχνολογική στοίβα της εφαρμογή μια βάση δεδομένων. Η τελική τεχνολογική στοίβα της εφαρμογής περιέχει React, Flask, Elasticsearch και MySQL. Λεπτομέρειες για την MySQL αναγράφονται στο υπο-κεφάλαιο 3.4.

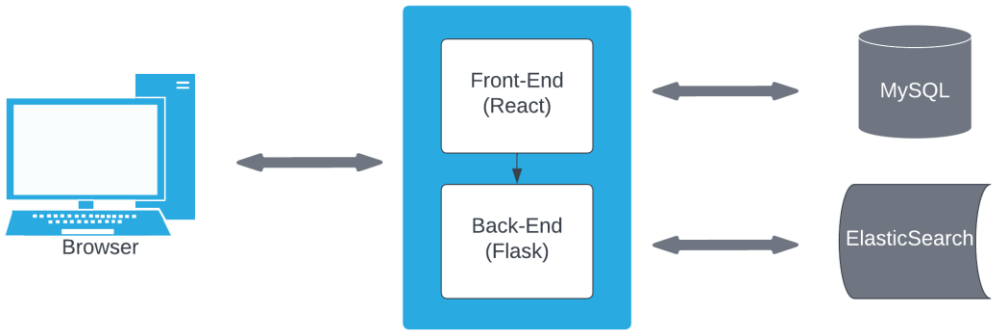
Η Flask επιτρέπει τη σχεδίαση μιας διεσπάρης προγραμματισμού (API), μέσω της οποίας δηλώνονται όλες οι διαδρομές της εφαρμογής (URL paths) και η λειτουργικότητα που υποστηρίζουν. Πιο κάτω παρουσιάζονται όλα τα άκρα της διεπαφής προγραμματισμού της εφαρμογής (API endpoints), με την αντίστοιχή λειτουργικότητα που παρέχουν:

1. search: Αναζήτηση προϊόντος στο ευρετήριο του elasticsearch.
2. searchStore: Αναζήτηση ευρετηρίου elasticsearch, με βάση το όνομα της κινητής καντίνας.
3. login: Επαλήθευση και σύνδεση χρήστη.
4. logout: Αποσύνδεση χρήστη και τερματισμός συνεδρίας (session).
5. register: Εγγραφή νέου χρήστη στην εφαρμογή.
6. home: Παρουσίαση αρχικής οθόνης της εφαρμογής, για τελικούς χρήστες.
7. workerHome: Παρουσίαση αρχικής οθόνης της εφαρμογής, για εργαζόμενους στις κινητές καντίνες.
8. profile: Παρουσίαση προσωπικών στοιχείων χρήστη.
9. myOrders: Προβολή ιστορικού παραγγελιών χρήστη.
10. placeOrder: Προβολή οθόνης για τοποθέτηση παραγγελίας.

11. cancel: Ακύρωση παραγγελίας.
12. saveOrder: Αποστολή παραγγελίας.
13. completeOrder: Επισήμανση παραγγελίας ως ολοκληρωμένη.

3.2 Αρχιτεκτονική Μονολιθικής Εφαρμογής

Η μονολιθική έκδοση της εφαρμογής [nmikal98A], έχει βασιστεί στο κλασικό μοντέλο μιας μονολιθικής εφαρμογής. Όλη η λειτουργικότητα της εφαρμογής συσκευάζεται σε ένα μόνο εκτελέσιμο αρχείο, όπως φαίνεται στην Εικόνα 3.3.



Εικόνα 3.3: Εικονική αναπαράσταση μονολιθικής αρχιτεκτονικής

Το κύριο αρχείο του πηγαίου κώδικα της εφαρμογής (app.py), περιέχει τους ορισμούς για όλες τις διαδρομές (URLs) της εφαρμογής, καθώς και τις υλοποιήσεις των αντίστοιχων συναρτήσεων που πρέπει να εκτελεστούν. Πιο κάτω αναλύονται οι κύριες λειτουργίες της εφαρμογής.

1. Εγγραφή νέου χρήστη

   Με τη λειτουργία εγγραφής, ένας χρήστης της εφαρμογής μπορεί να δημιουργήσει ένα νέο λογαριασμό. Η δημιουργία νέου λογαριασμού απαιτεί την εισαγωγή των στοιχείων του χρήστη (όνομα, ηλεκτρονική διεύθυνση, τηλέφωνο

επικοινωνίας, διεύθυνση διαμονής) και την επιλογή ενός κωδικού πρόσβασης. Για τη διαχείριση των κωδικών πρόσβασης χρησιμοποιείται η βιβλιοθήκη Bcrypt.
Το Bcrypt είναι μια βιβλιοθήκη της γλώσσας προγραμματισμού Python, που χρησιμοποιείται για τη δημιουργία ασφαλών κατακερματισμένων κωδικών πρόσβασης. Ο κατακερματισμός είναι μια τεχνική που χρησιμοποιείται για τη μετατροπή των κωδικών πρόσβασης σε μια μοναδική, αναγνωρίσιμη μορφή, γνωστή ως hash, που δεν μπορεί να αντιστραφεί για να αποκαλύψει τον πρωτότυπο κωδικό πρόσβασης.

Όταν ένας χρήστης δημιουργεί τον κωδικό πρόσβασης του, η εφαρμογή θα χρησιμοποιήσει το Bcrypt για να δημιουργήσει μια κατακερματισμένη εκδοχή του κωδικού πρόσβασης και θα την αποθηκεύσει, μαζί με τα υπόλοιπα στοιχεία του χρήστη στη βάση δεδομένων.

2. Σύνδεση

Μετά τη δημιουργία ενός λογαριασμού, ο χρήστης μπορεί να συνδεθεί στον λογαριασμό του. Όταν ο χρήστης προσπαθήσει να συνδεθεί στο λογαριασμό του, η εφαρμογή, με τη βοήθεια της βιβλιοθήκης Bcrypt, θα κατακερματίσει τον κωδικό πρόσβασης που εισάγει και θα τον συγκρίνει με τον αποθηκευμένο κατακερματισμένο κωδικό. Αν τα δύο hashes ταιριάζουν, ο κωδικός πρόσβασης είναι σωστός και η σύνδεση επιτρέπεται.

Με την επιτυχημένη σύνδεση στον λογαριασμό του, ο χρήστης έχει τη δυνατότητα να τοποθετήσει μια παραγγελία σε κάποια κινητή καντίνα, να δει τις παραγγελίες που έχει τοποθετήσει στο παρελθόν και να δει τα προσωπικά του στοιχεία.

3. Αναζήτηση προϊόντος

Από την αρχική οθόνη της εφαρμογής, δίνεται η δυνατότητα αναζήτησης ενός προϊόντος. Η αναζήτηση γίνεται με τη βοήθεια του Elasticsearch. Όταν ο χρήστης εισάγει ένα κείμενο στη μπάρα αναζητήσεις, αυτό το κείμενο μεταφράζεται σε μορφή JSON.

Το νέο JSON που δημιουργείται έχει την ακόλουθη μορφή: "query": {"match": {"fooditems": key}}, όπου `key` είναι το κείμενο που εισήγαγε ο χρήστης. Αυτό το JSON στη συνέχεια χρησιμοποιείται από τη συνάρτηση αναζήτησης του Elasticsearch, όπου αντιστοιχίζει το κείμενο αναζήτησης με τα διαθέσιμα προϊόντα που είναι αποθηκευμένα και επιστρέφει τις πληροφορίες των καντινών που έχουν το συγκεκριμένο προϊόν.

4. Τοποθέτηση Παραγγελίας

Αφού συνδεθεί στον λογαριασμό του, ο χρήστης μπορεί να τοποθετήσει μια παραγγελία. Με την αναζήτηση ενός προϊόντος, παρουσιάζονται στον χρήστη οι καντίνες που έχουν αυτό το προϊόν και δίνεται στο χρήστη η επιλογή να τοποθετήσει παραγγελία σε μια καντίνα.

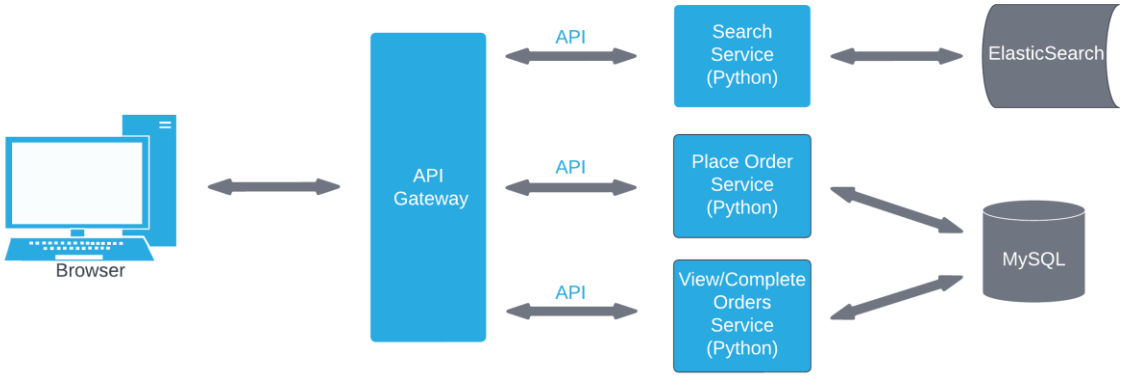
Η τοποθέτηση της παραγγελίας γίνεται με τη φόρτωση μια φόρμας η οποία περιέχει όλα τα διαθέσιμα προϊόντα της καντίνας. Στη συνέχεια, με την υποβολή της φόρμας, η παραγγελία αποθηκεύεται στη βάση δεδομένων.

5. Προβολή / Εξυπηρέτηση Παραγγελιών

Η προβολή παραγγελιών εξυπηρετεί δύο σκοπούς. Αρχικά, ένας χρήστης μπορεί να δει το ιστορικό παραγγελιών που έχει καταχωρίσει μέσω της εφαρμογής.
Παράλληλα, ο ιδιοκτήτης/υπάλληλος μιας κινητής καντίνας, μπορεί να συνδεθεί στην εφαρμογή και να δει όλες τις παραγγελίες που έχουν τοποθετηθεί στην καντίνα του. Στη συνέχεια, μπορεί να επισημάνει μια παραγγελία ως ολοκληρωμένη. Με αυτό τον τρόπο μπορούν να διαχειριστούν καλύτερα τις τρέχουσες παραγγελίες και ταυτόχρονα ενημερώνεται ο χρήστης για την κατάσταση της παραγγελίας του.

3.3 Αρχιτεκτονική Μικρουπηρεσιών

Η αρχιτεκτονική της εκδοχής που βασίζεται σε μικρουπηρεσίες [nmikal98B], ακολουθεί τη κλασική μορφή μιας εφαρμογής μικρουπηρεσιών, όπως αυτή έχει παρουσιαστεί σε προηγούμενα κεφάλαια. Συγκεκριμένα, το εμπρόσθιο άκρο (front-end) τις εφαρμογής λειτουργεί σαν μια πύλη API (API Gateway). Αυτή η πύλη διαχειρίζεται τις εισερχόμενες αιτήσεις, τις οποίες προωθεί στις κατάλληλες υπηρεσίες για να εξυπηρετηθούν. Η αρχιτεκτονική αυτής της εκδοχής της εφαρμογής παρουσιάζεται στην Εικόνα 3.4.



Εικόνα 3.4: Εικονική αναπαράσταση αρχιτεκτονικής μικρουπηρεσιών

Σε αντίθεση με την μονολιθική υλοποίηση της εφαρμογής, η εκδοχή των μικρουπηρεσιών, παρόλο που ορίζει όλες τις διαδρομές (URLs) της εφαρμογής σε ένα αρχείο, οι υλοποιήσεις τους βρίσκονται σε διαφορετικά, ανεξάρτητα προγράμματα.

Η λειτουργικότητα της εφαρμογής είναι η ίδια με τη μονολιθική εκδοχή. Παρουσιάζεται διαφοροποίηση στον τρόπο που εξυπηρετούνται οι πιο κάτω λειτουργίες:

1. Αναζήτηση προϊόντος

   Στη μονολιθική εκδοχή της εφαρμογής περιγράφεται η λειτουργικότητα της αναζήτησης προϊόντων και πως η αναζήτηση εξυπηρετείται από το Elasticsearch.

Στην αρχιτεκτονική των μικρουπηρεσιών, το κείμενο αναζήτησής που εισάγει ο χρήστης, αποστέλλεται στην υπηρεσία που είναι υπεύθυνη για την εξυπηρέτηση των αιτημάτων αναζήτησης. Στη συνέχεια, η υπηρεσία αυτή είναι υπεύθυνη να ολοκληρώσει την αναζήτηση με τη βοήθεια του Elasticsearch και να επιστρέψει τα αποτελέσματα στο εμπρόσθιο άκρο (front-end) της εφαρμογής που δείχνει τα αποτελέσματα στο χρήστη.

Η επικοινωνία μεταξύ του εμπρόσθιου άκρου (front-end) και της υπηρεσίας αναζήτησης, επιτυγχάνεται με το πρωτόκολλο επικοινωνίας gRPC. Στο protobuf της υπηρεσίας αναζήτησης ορίζεται η δομή των μνημάτων που ανταλλάζονται μεταξύ των υπηρεσιών. Το εμπρόσθιο άκρο της εφαρμογής στέλνει ένα μήνυμα το οποίο περιέχει μόνο το κείμενο αναζήτησης του χρήστη. Η υπηρεσία αναζήτησης επιστρέφει ένα μήνυμα το οποίο περιέχει τα αντικείμενα των καντινών όπως αυτά είναι αποθηκευμένα στο Elasticsearch, τον αριθμό τον αποτελεσμάτων, τις τοποθεσίες και το αποτέλεσμα της αναζήτησής (αν ήταν επιτυχής ή όχι). Στην εικόνα 3.5 παρουσιάζεται ο ορισμός του protobuf για την υπηρεσία αναζήτησης.

Όπως φαίνεται στην πιο κάτω εικόνα, ο ορισμός ενός protobuf περιέχει τις δηλώσεις των εφαρμογών που καλούνται μέσω της απομακρυσμένης κλήσης εργασίας (remote procedure call) και η δομή των μνημάτων που περνούν και επιστρέφονται από την εργασία. Η δομή των μηνυμάτων ορίζεται με τη χρήση της εντολής ‘message’, το όνομα του μηνύματος και στη συνέχεια τις μεταβλητές που περιέχονται στο μήνυμα μαζί με τον τύπο δεδομένων τους.

```
syntax = "proto3";

service searchService {
    rpc Search (searchRequest) returns (searchResponse) {}
    rpc SearchStore (searchRequest) returns (searchStoreResponse) {}
}

message searchRequest {
    string req = 1;
}

message searchResponse {
    repeated Trucks trucks = 1;
    int32 hits = 2;
    int32 locations = 3;
    string status = 4;
}

message searchStoreResponse {
    repeated Trucks trucks = 1;
}

message Trucks {
    string name = 1;
    repeated string fooditems = 2;
    repeated Branches branches = 3;
    bool drinks = 4;
}

message Branches {
    string hours = 1;
    string schedule = 2;
    string address = 3;
    Location location = 4;
}

message Location {
    string longitude = 1;
    string latitude = 2;
    string human_address = 3;
}

message HumanAddress {
    string address = 1;
    string city = 2;
    string state = 3;
    string zip = 4;
}
```

Εικόνα 3.5: Ορισμός protobuf για υπηρεσία αναζήτησης.

2. Τοποθέτηση Παραγγελίας

   Η λειτουργία τοποθέτησης παραγγελίας υλοποιείται σε μια ξεχωριστή υπηρεσία, με παρόμοιο τρόπο όπως η υπηρεσία αναζήτησης. Σε αυτή τη περίπτωση, τα μνήματα που ανταλλάζονται μέσω του πρωτοκόλλου επικοινωνίας gRPC έχουν την ακόλουθη δομή: το εμπρόσθιο άκρο (front-end) της εφαρμογής αποστέλλει ένα μήνυμα το οποίο περιέχει τον μοναδικό αριθμό ταυτοποίησης του χρήστη που είναι συνδεδεμένος, το όνομα της καντίνας που επιθυμεί να τοποθετήσει την παραγγελία, την τοποθεσία της καντίνας (στη περίπτωση που η ίδια καντίνα μπορεί να βρίσκεται σε πολλαπλές τοποθεσίες), τα αντικείμενα που επιθυμεί να παραγγείλει και τέλος την ημερομηνία και ώρα που τοποθέτησε την παραγγελία. Μετά την ολοκλήρωση της διαδικασίας τοποθέτησης παραγγελίας, η υπηρεσία που είναι υπεύθυνη για την παραγγελία επιστρέφει στο εμπρόσθιο άκρο της εφαρμογής ένα μήνυμα επιτυχίας/αποτυχίας.

3. Προβολή / Εξυπηρέτηση Παραγγελιών

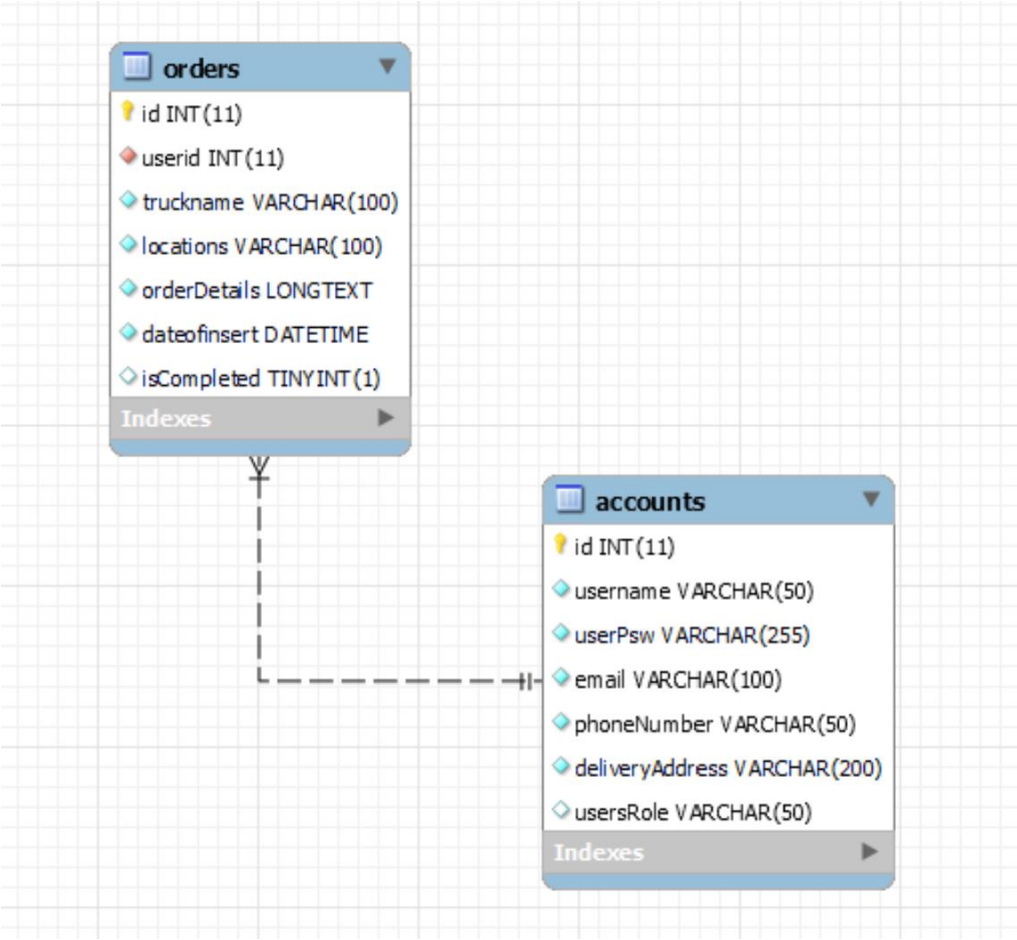
   Η τελευταία λειτουργεία που υλοποιήθηκε σαν ξεχωριστή υπηρεσία είναι η υπηρεσία προβολής και εξυπηρέτησης παραγγελιών. Σε αυτή τη περίπτωση, τα μηνύματα που ανταλλάζονται μεταξύ του εμπρόσθιου άκρου της εφαρμογής και της υπηρεσίας περιέχουν τον μοναδικό αριθμό ταυτοποίησης μιας υπηρεσίας και το αποτέλεσμα της λειτουργίας (επιτυχία/αποτυχία).

3.4 Σχεσιακή Βάση Δεδομένων

MySQL είναι ένα σύστημα διαχείρισης βάσεων δεδομένων σχεσιακού τύπου (RDBMS) και είναι βασισμένο στη γλώσσα SQL (Structured Query Language). Η επιλογή της MySQL έναντι μιας κατανεμημένης βάσης δεδομένων ή κάποιου αποθηκευτικού συστήματος κλειδιού-τιμής (key-value stores) έγινε για τους ακόλουθους λόγους.

1. Η MySQL είναι ιδανική για εφαρμογές που απαιτούν δομημένα δεδομένα με συγκεκριμένες σχέσεις μεταξύ των πινάκων, όπως συμβαίνει στην υποστήριξη δυνατότητας τοποθέτησης παραγγελιών, όπου η παραγγελία και ο χρήστης πρέπει να συνδεθούν με συγκεκριμένους κανόνες.

2. Η MySQL επιτρέπει σύνθετες ερωτήσεις αναζήτησης, αυτό είναι ιδιαίτερα χρήσιμο σε μια εφαρμογή που υποστηρίζει τοποθέτηση παραγγελιών, αφού υπάρχει η ανάγκη αναζήτησης μιας παραγγελίας βάση διαφόρων παραμέτρων.



Εικόνα 3.6: Σχήμα βάσης δεδομένων.

Η δομή της βάσης δεδομένων της εφαρμογής είναι αρκετά απλή. Η μόνη λειτουργικότητα που υποστηρίζεται από τη βάση είναι η ταυτοποίηση χριστών και η τοποθέτηση παραγγελίας. Επομένως, το σχήμα της βάσης αποτελείται από δυο πίνακες. Ο ένας περιέχει πληροφορίες για τον λογαριασμό κάθε χρήστη. Ο άλλος περιέχει πληροφορίες για κάθε παραγγελία και αναφέρει τον πίνακα των λογαριασμών σαν εξωτερικό κλειδί (foreign key). Στην εικόνα 3.6 φαίνεται εικονική απεικόνιση του σχήματος της βάσης.

Ακολουθούν οι λειτουργίες της εφαρμογής που απαιτούν διασύνδεση με τη βάση δεδομένων και τα αντίστοιχα ερωτήματα (queries):

1. Ανάκτηση πληροφορίες λογαριασμού με βάση το όνομα χρήστη:

   'SELECT * FROM accounts WHERE username = %s ', (username,))

2. Δημιουργία νέου λογαριασμού χρήστη:

   'INSERT INTO accounts VALUES (NULL, %s, %s, %s,%s ,%s)', (username, hashedPassword, email, phoneNum, deliveryAddr))

3. Ανάκτηση παραγγελιών ενός χρήστη:

   'SELECT * FROM orders WHERE userid = %s', (session['id'],)

4. Τοποθέτηση παραγγελίας:

   'INSERT INTO orders VALUES (NULL, %s, %s, %s,%s, %s, %s)', (session['id'], truckname, location, orderDetails, date, 0))

5. Επισύναψη παραγγελίας ως ολοκληρωμένη:

   'UPDATE orders SET isCompleted = 1 WHERE id =%s;',(id)

Κεφάλαιο 4

Αξιολόγηση

4.1 Πειραματική Αξιολόγηση

Για να είναι εφικτή η σύγκριση μεταξύ της μονολιθικής υλοποίησης της εφαρμογής και της υλοποίησης με μικρουπηρεσίες, έχουν διεξαχθεί κάποια πειράματα τα οποία παρουσιάζουν την απόδοση της κάθε εφαρμογής σε διάφορες συνθήκες.
Για τη πειραματική αξιολόγηση των δυο υλοποιήσεων έχει χρησιμοποιηθεί το CloudLab. Το CloudLab είναι μια πλατφόρμα για πειραματική έρευνα και παρέχει υπηρεσίες για τη διεξαγωγή επιστημονικών πειραμάτων σε περιβάλλον νέφους (cloud). Αυτό επιτρέπει στους ερευνητές να δοκιμάζουν νέες ιδέες σε ένα περιβάλλον που προσομοιώνει το υπολογιστικό νέφος σε μεγάλη κλίμακα.

Το CloudLab προσφέρει πρόσβαση σε μια ποικιλία υλικού και λογισμικού, συμπεριλαμβανομένων διάφορων τύπων μηχανημάτων, δικτύων και αποθηκευτικών συστημάτων. Επιπλέον, παρέχει εργαλεία για την αυτοματοποίηση και τον έλεγχο των πειραμάτων, καθιστώντας την επιστημονική έρευνα πιο αποτελεσματική και αξιόπιστη.

Για την αξιολόγηση της εφαρμογής Foodtrucks, στήθηκαν δυο σμήνη (swarm) Docker στα οποία αναπτύχτηκαν (deployed) οι εφαρμογές στις δυο αρχιτεκτονικές. Κάθε σμήνος αποτελείται από πέντε κόμβους, εκ των οποίων ο ένας είναι ο διαχειριστής (manager) και οι υπόλοιποι τέσσερεις εργάτες (workers). Ο αριθμός τον κόμβων επιλέχτηκε με βάση τον αριθμό των υπηρεσιών στην αρχιτεκτονική των μικρουπηρεσιών.

Για τη διατήρηση συνοχής στα πειράματα και για την βελτίωση της αξιοπιστίας των πειραμάτων, τα δυο σμήνη οργανώθηκαν στο ίδιο υπολογιστικό κέντρο και τα πειράματα εκτελέστηκαν την ίδια χρονική περίοδο. Συγκεκριμένα, τα πειράματα αυτά έχουν διεξαχθεί με το εργαλείο wrk.

4.1.1 Εργαλείο Αξιολόγησης WRK

Το wrk αποτελεί ένα σύγχρονο εργαλείο αξιολόγησης των επιδόσεων HTTP, το οποίο χρησιμοποιείται για την εκτίμηση και την ανάλυση της απόδοσης ενός διακομιστή ιστού (web server). Αυτό το εργαλείο παρέχει σημαντικές πληροφορίες σχετικά με την ικανότητα ενός διακομιστή να διαχειρίζεται συγκεκριμένα επίπεδα φόρτου και κίνησης, πριν αρχίσει να παρουσιάζει προβλήματα ή να υπερβαίνει τα προκαθορισμένα όρια απόδοσης.

Το wrk προσφέρει τη δυνατότητα δημιουργίας πολλαπλών νημάτων (threads), επιτρέποντας την παραμετροποίηση του αριθμού των ανοιχτών συνδέσεων ανά thread, ενώ παράλληλα καθορίζει τη διάρκεια της διαδικασίας δοκιμής. Επιπρόσθετα, το wrk υποστηρίζει την εκπόνηση σεναρίων δοκιμής μέσω της γλώσσας προγραμματισμού Lua, παρέχοντας έτσι μεγάλη ευελιξία και προσαρμογή στις ανάγκες των διάφορων δοκιμών.

4.1.2 Γλώσσα Προγραμματισμού LUA

Η Lua αποτελεί μια δυναμική, ερμηνευμένη (interpreted) γλώσσα προγραμματισμού υψηλού επιπέδου. Αρχικά σχεδιάστηκε για ενσωμάτωση σε συστήματα πραγματικού χρόνου (real-time systems) και έχει αποκτήσει μεγάλη δημοτικότητα για τη χρήση της σε βιντεοπαιχνίδια, ενσωματωμένα συστήματα (embedded systems), και άλλες εφαρμογές όπου η επεκτασιμότητα και η ευελιξία είναι παράγοντες ζωτικής σημασίας.

Η Lua χαρακτηρίζεται από μια απλοποιημένη σύνταξη που διευκολύνει την ταχεία ανάπτυξη, ενώ παρέχει πλούσιες δυνατότητες διαχείρισης δεδομένων, όπως λίστες (lists), σύνολα (sets), και πολύπλοκες δομές δεδομένων (complex data structures). Παράλληλα, είναι μια ελαφριά γλώσσα προγραμματισμού, η οποία έχει σχεδιαστεί για να είναι γρήγορη και αποδοτική. Αυτό την καθιστά ιδανική για την δημιουργία μικρών προγραμμάτων (scripts) που θα

εκτελούνται σε περιβάλλοντα όπου οι πόροι είναι περιορισμένοι ή η απόδοση είναι κρίσιμη, όπως σε δοκιμές επιδόσεων με το wrk.

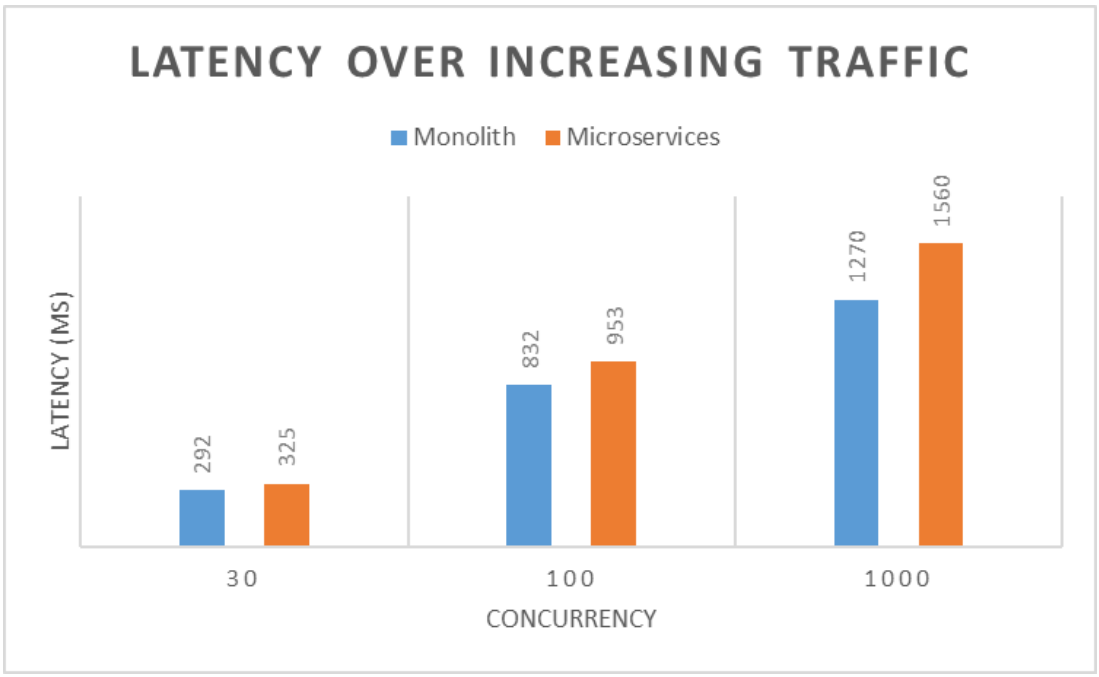
Για την αξιολόγηση της εφαρμογής FoodTrucks, έχει δημιουργηθεί ένα πρόγραμμα σε Lua, το οποίο προσδιορίζει τις παραμέτρους των αιτημάτων του εργαλείου wrk. Στο πρόγραμμα αυτό δηλώνεται μια προκαθορισμένη λίστα από αντικείμενα και καταστήματα που υπάρχουν στο ευρετήριο (index) του Elasticsearch. Στη συνέχεια δηλώνονται δυο συναρτήσεις οι οποίες καθορίζουν τη μορφή του αιτήματος. Η μια συνάρτηση καθορίζει αιτήματα αναζήτησης και η άλλη αιτήματα τοποθέτησης παραγγελιών. Τα περιεχόμενα της αναζήτησης και της παραγγελίας κατασκευάζονται δυναμικά, αντλώντας τυχαία πληροφορίες από τη λίστα με αντικείμενα. Τέλος, το αίτημα επιστρέφεται σαν είσοδος στο εργαλείο wrk. Τα αιτήματα εναλλάσσονται τυχαία μεταξύ αίτημα αναζήτησης και αίτημα παραγγελίας, διατηρώντας τον αριθμό των αιτημάτων της κάθε κατηγορίας ίσο.

4.2 Παρουσίαση Αποτελεσμάτων

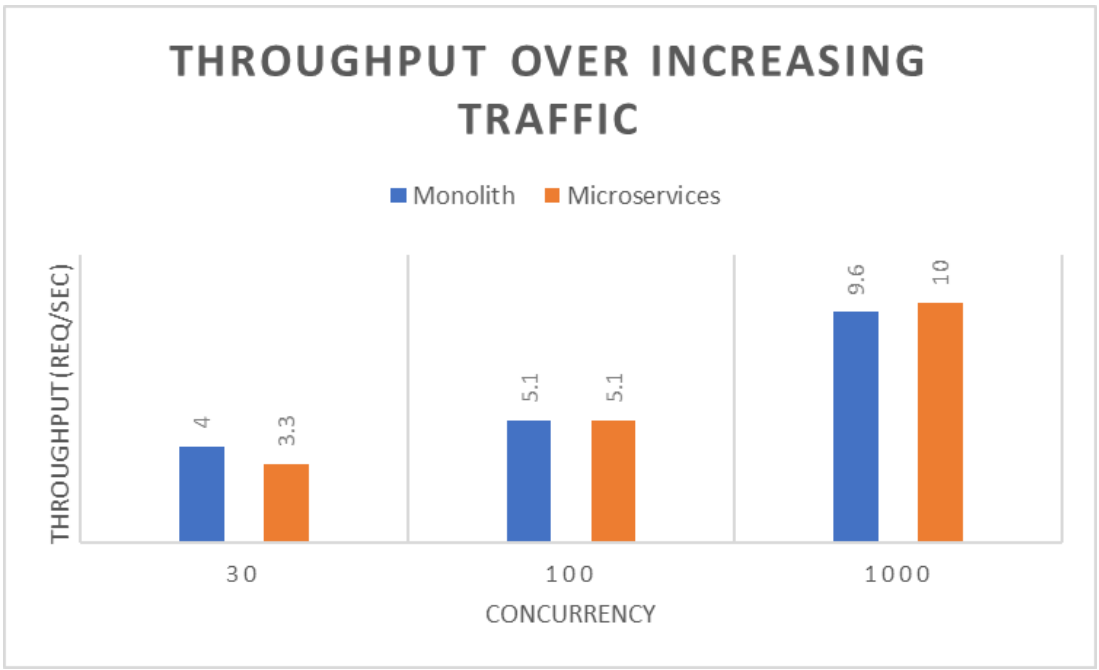
Για τη διεξαγωγή του πρώτου πειράματος ισχύουν τα ακόλουθα:

1. Οι ρυθμίσεις του wrk για τον αριθμό των νημάτων και ο χρόνος του πειράματος παραμένουν σταθερά. Ο αριθμός τον νημάτων είναι 28. Αυτή η απόφαση έχει παρθεί με βάση τον αριθμό των φυσικών υπολογιστικών μονάδων (CPUs) του συστήματος που αναπτύχθηκαν οι εφαρμογές. Με τη βοήθεια της εντολής lscpu διακριβώθηκε ότι το σύστημα αποτελείται από 2 υποδοχές (sockets) με 14 πυρήνες (cores) ανά υποδοχή και δυο νήματα (threads) ανά πυρήνα. Ο χρόνος εκτέλεσης του κάθε πειράματος είναι 30 δευτερόλεπτα.
2. Όλες οι τιμές που παρουσιάζονται πιο κάτω αποτελούν τον μέσω όρο των τιμών που πάρθηκαν μετά από 3 εκτελέσεις του κάθε πειράματος.
3. Μικρός φόρτος εργασίας θεωρείται ένα πείραμα με 30 παράλληλες συνδέσεις, μεσαίος φόρτος εργασίας οι 300 παράλληλες συνδέσεις και μεγάλος φόρτος εργασίας οι 1000 παράλληλες συνδέσεις.

Το πρώτο πείραμα που έχει διεξαχθεί είναι η μέτρηση της καθυστέρησης (latency) και της ρυθμαπόδοσης (throughput) των δυο εφαρμογών κάτω από μικρό, μεσαίο και μεγάλο φόρτο εργασίας. Τα αποτελέσματα του πειράματος φαίνονται πιο κάτω, στις εικόνες 4.1 και 4.3 αντίστοιχα.



Εικόνα 4.1: Παρουσίαση καθυστέρησης μονολιθικής αρχιτεκτονικής και μικρουπηρεσιών.



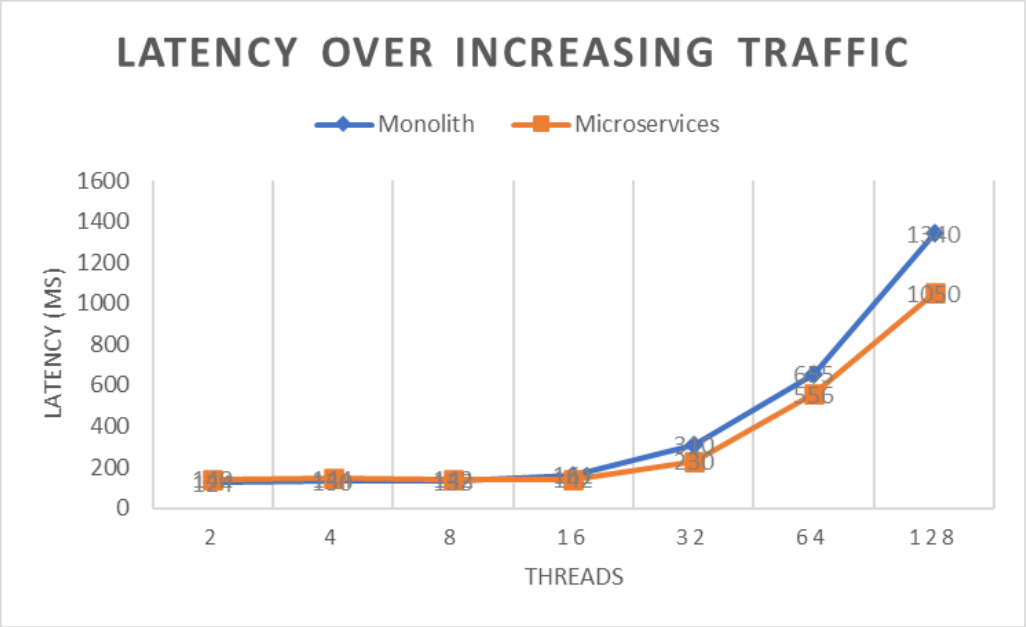
Εικόνα 4.2: Παρουσίαση ρυθμαπόδοσης μονολιθικής αρχιτεκτονικής και μικρουπηρεσιών.

Λαμβάνοντας υπόψη τη τυπική απόκλιση των αποτελεσμάτων, από το πιο πάνω πείραμα μπορούμε να δούμε ότι οι δυο υλοποιήσεις ανταποκρίνονται περίπου το ίδιο, με τη μονολιθική εφαρμογή να επιδεικνύει μικρότερα επίπεδα καθυστέρησης σε μεγαλύτερο φόρτο εργασίας. Αυτό είναι αναμενόμενο λόγο της επιπλέων επικοινωνίας που απαιτείται μεταξύ των υπηρεσιών στην αρχιτεκτονική των μικρουπηρεσιών.

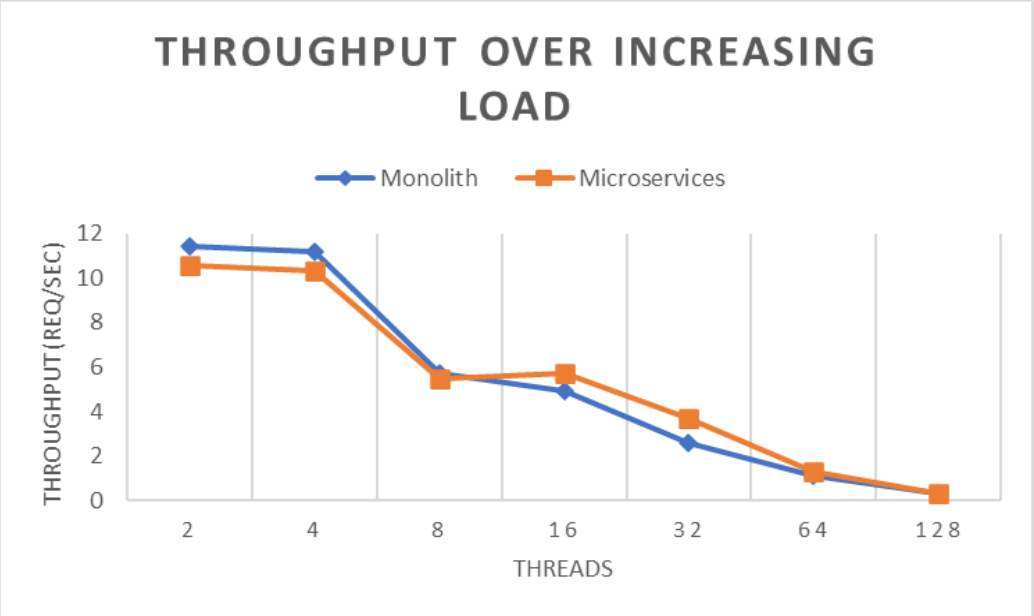
Για την καλύτερη διερεύνηση των πιο πάνω αποτελεσμάτων, έχει εκτελεστεί το ίδιο πείραμα με διαφορετικές παραμέτρους στο εργαλείο wrk. Τα επόμενα πειράματα έχουν διεξαχθεί για τη διακρίβωση της συμπεριφοράς κάθε εφαρμογής κάτω από διαφορετικούς φόρτους εργασίας και διαφορετικό αριθμό νημάτων. Τα αποτελέσματα των πειραμάτων φαίνονται στις εικόνες 4.3 και 4.4.

Για τη διεξαγωγή του δεύτερου πειράματος ισχύουν τα ακόλουθα:

1. Η ρύθμιση του wrk για τον χρόνο του πειράματος παραμένει σταθερή στα 30 δευτερόλεπτα.
2. Κάθε μέτρηση γίνεται με αυξανόμενο αριθμό νημάτων και παράλληλων συνδέσεων, συγκεκριμένα για 2 νήματα χρησιμοποιήθηκαν 5 συνδέσεις, για 4 νήματα χρησιμοποιήθηκαν 10 συνδέσεις, για 8 νήματα χρησιμοποιήθηκαν 15 συνδέσεις, για 16 νήματα χρησιμοποιήθηκαν 20 συνδέσεις, για 32 νήματα χρησιμοποιήθηκαν 50 συνδέσεις, για 64 νήματα χρησιμοποιήθηκαν 100 συνδέσεις, για 128 νήματα χρησιμοποιήθηκαν 200 συνδέσεις,
3. Όλες οι τιμές που παρουσιάζονται πιο κάτω αποτελούν τον μέσω όρο των τιμών που πάρθηκαν μετά από 3 εκτελέσεις του κάθε πειράματος.



Εικόνα 4.3: Παρουσίαση καθυστέρησης με αυξανόμενο φόρτο εργασίας.



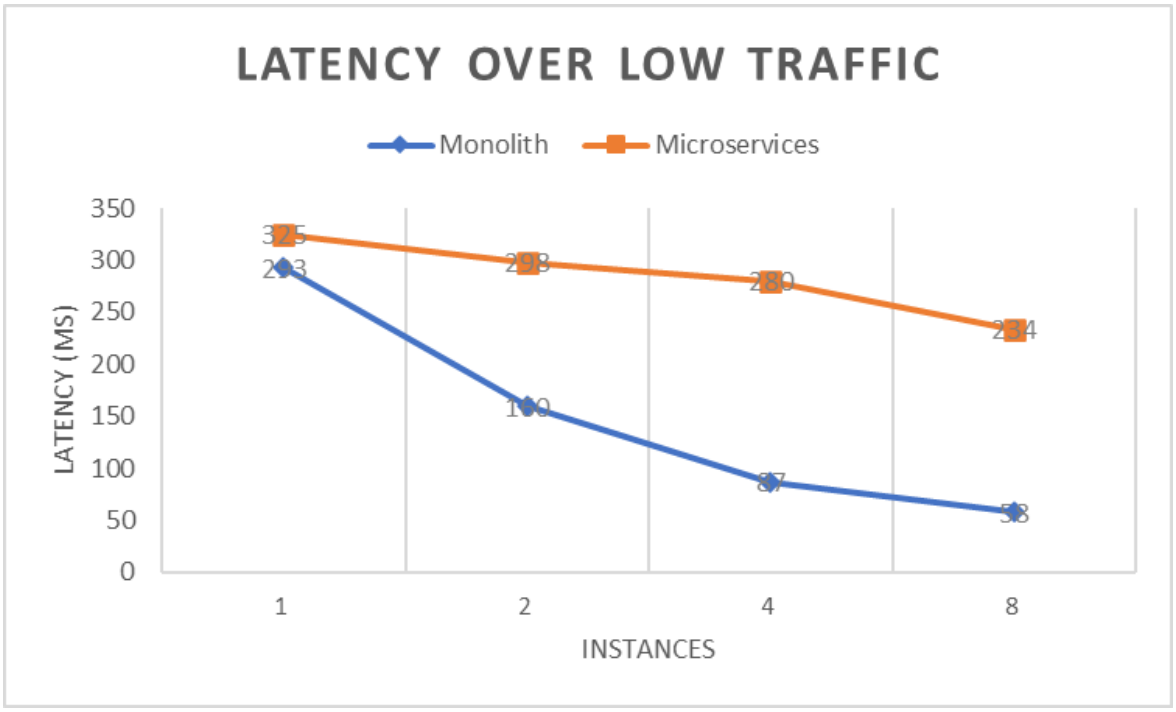
Εικόνα 4.4: Παρουσίαση ρυθμαπόδοσης με αυξανόμενο φόρτο εργασίας.

Από τα πιο πάνω πειράματα μπορούμε να διακρίνουμε ότι η μονολιθική υλοποίηση της εφαρμογής είναι πιο αποδοτική με πιο χαμηλό αριθμό νημάτων, ενώ η υλοποίηση στην αρχιτεκτονική των μικρουπηρεσιών είναι πιο αποδοτική από τη μονολιθική στις περιπτώσεις που έχουμε μεγαλύτερο αριθμό νημάτων. Αυτό μπορεί να ευθύνεται στην απομόνωση μεταξύ των υπηρεσιών. Αφού κάθε υπηρεσία εκτελείται ανεξάρτητα, ο φόρτος εργασίας σε μια υπηρεσία δεν επηρεάζει άμεσα τις άλλες.

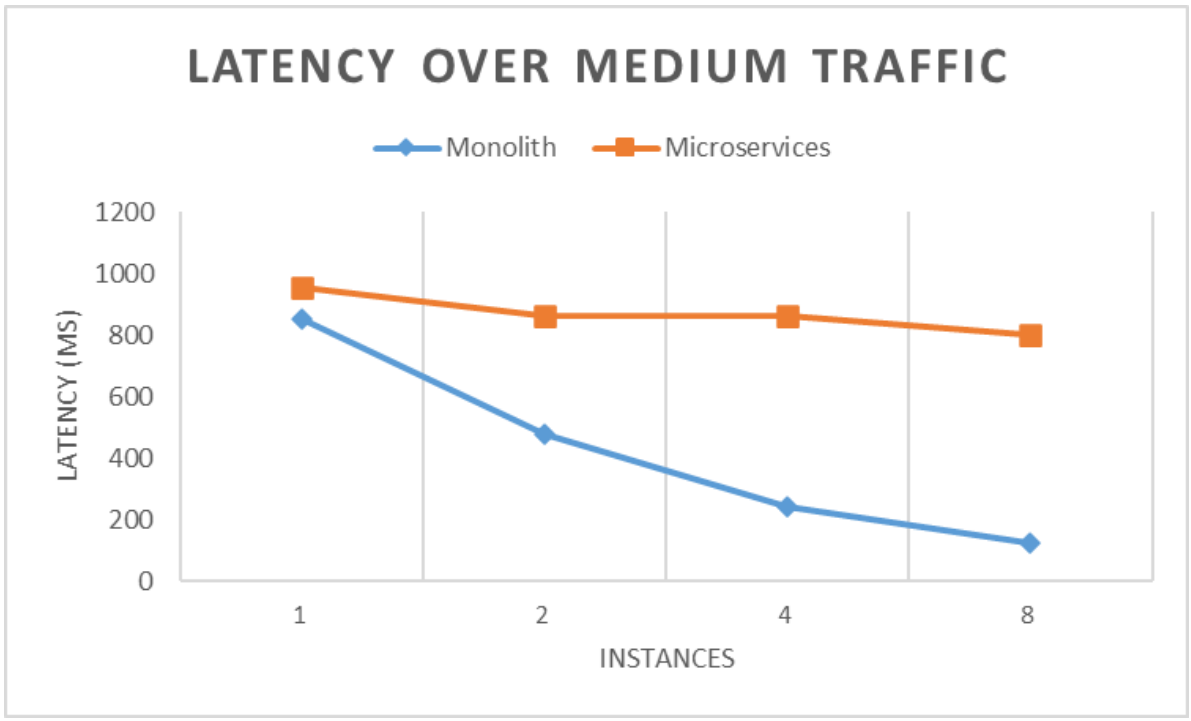
Τέλος, εξετάστηκε η συμπεριφορά κάθε εφαρμογής κατά τη διαδικασία κλιμάκωσης. Η κάθε εφαρμογή εξετάστηκε στον ελάχιστο αριθμό στιγμιότυπων (instances), δηλαδή ένα στιγμιότυπο κάθε υπηρεσίας, μέχρι και με 8 στιγμιότυπα κάθε υπηρεσίας. Τα πειράματα επαναλήφθηκαν για μικρό, μεγάλο και μεσαίο φόρτο εργασίας. Στη περίπτωση της μονολιθικές εφαρμογής, κλιμακώνεται ολόκληρη η εφαρμογή, ενώ στην υλοποίηση με μικρουπηρεσίες, κλιμακώνονται μόνο οι υπηρεσίες αναζήτησης και τοποθέτησης παραγγελιών. Τα αποτελέσματα των πειραμάτων κλιμάκωσης φαίνονται στις εικόνες 4.5, 4.6 και 4.7.

Για τη διεξαγωγή του τρίτου πειράματος ισχύουν τα ακόλουθα:

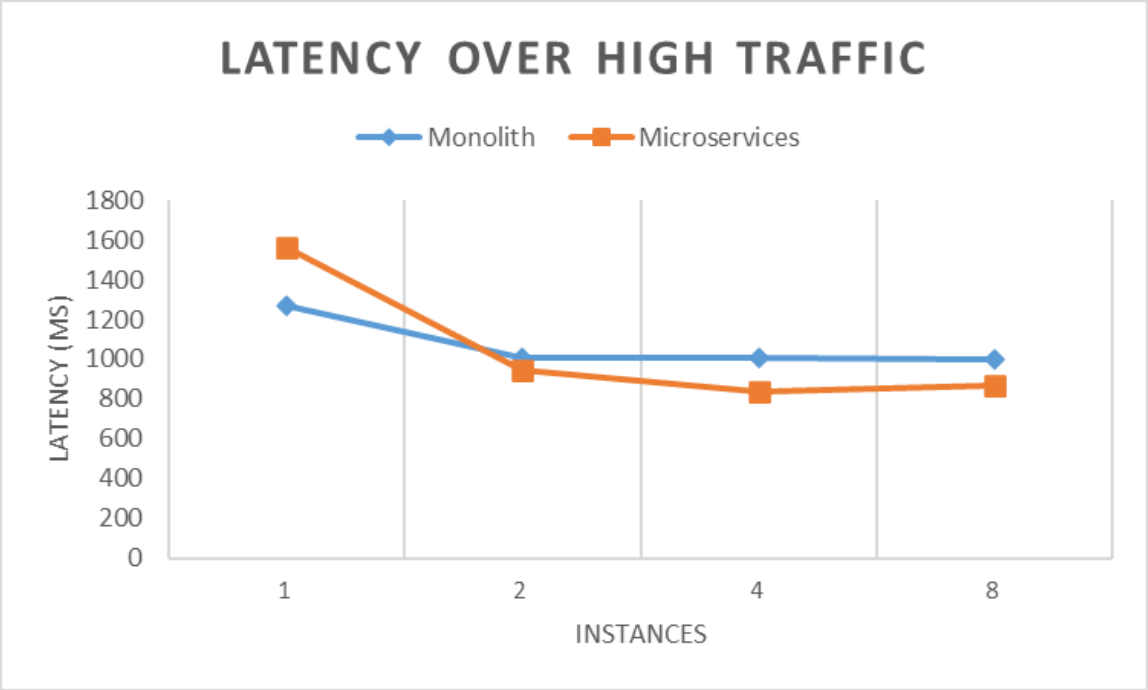
1. Οι ρυθμίσεις του wrk για τον αριθμό των νημάτων και ο χρόνος του πειράματος παραμένουν σταθερά. Ο αριθμός τον νημάτων είναι 28. Ο χρόνος εκτέλεσης του κάθε πειράματος είναι 30 δευτερόλεπτα.
2. Όλες οι τιμές που παρουσιάζονται πιο κάτω αποτελούν τον μέσω όρο των τιμών που πάρθηκαν μετά από 3 εκτελέσεις του κάθε πειράματος.
3. Μικρός φόρτος εργασίας θεωρείται ένα πείραμα με 30 παράλληλες συνδέσεις, μεσαίος φόρτος εργασίας οι 300 παράλληλες συνδέσεις και μεγάλος φόρτος εργασίας οι 1000 παράλληλες συνδέσεις.



Εικόνα 4.5: Παρουσίαση καθυστέρησης μονολιθικής αρχιτεκτονικής και μικρουπηρεσιών κατά τη διαδικασία κλιμάκωσης με χαμηλό φόρτο εργασίας.



Εικόνα 4.6: Παρουσίαση καθυστέρησης μονολιθικής αρχιτεκτονικής και μικρουπηρεσιών κατά τη διαδικασία κλιμάκωσης με μεσαίο φόρτο εργασίας.



Εικόνα 4.7: Παρουσίαση καθυστέρησης μονολιθικής αρχιτεκτονικής και μικρουπηρεσιών κατά τη διαδικασία κλιμάκωσης με μεγάλο φόρτο εργασίας.

Τα πιο πάνω πειράματα δείχνουν ότι η μονολιθική εφαρμογή αποκρίνεται καλυτέρα στη κλιμάκωση σε περιπτώσεις μικρού φόρτου εργασίας. Στο συγκεκριμένο παράδειγμα φαίνεται ότι η μονολιθική εφαρμογή εξυπηρετεί πιο πολλά αιτήματα με πιο μικρή καθυστέρηση από την εφαρμογή στην αρχιτεκτονική μικρουπηρεσιών. Η συμπεριφορά αυτή είναι αναμενόμενη αφού οι εργασίες των ανεξάρτητων υπηρεσιών δεν είναι ιδιαίτερα απαιτητικές σε υπολογιστικό χρόνο, επομένως το επιπρόσθετο κόστος (overhead) επικοινωνίας μεταξύ των υπηρεσιών είναι αρκετά μεγάλο για να επηρεάζει αρνητικά την απόδοση της εφαρμογής. Ταυτόχρονα, η αρχιτεκτονική των μικρουπηρεσιών κλιμακώνεται καλύτερα σε περιπτώσεις μεγάλου φόρτου εργασίας όπου οι απαιτήσεις σε υλικούς πόρους των υπηρεσιών αυξάνονται και το όφελος της τμηματοποιημένης επεξεργασίας επικαλύπτει το κόστος επικοινωνίας μεταξύ των υπηρεσιών.

Κεφάλαιο 5

Συμπεράσματα

5.1 Συμπεράσματα

Η αρχιτεκτονική των μικρουπηρεσιών έχει γίνει μια από τις πιο δημοφιλείς επιλογές στον σχεδιασμό και την ανάπτυξη λογισμικού. Αυτή η προσέγγιση αποσπά μια μονολιθική εφαρμογή σε μικρότερες, αυτόνομες υπηρεσίες που συνεργάζονται μεταξύ τους. Κάθε μια από αυτές τις μικρουπηρεσίες εκτελεί μια συγκεκριμένη λειτουργία που μπορεί να αναπτυχθεί (developed), να εγκατασταθεί (deployed) και να κλιμακωθεί (scaled) ανεξάρτητα. Τόσο η αρχιτεκτονική των μικρουπηρεσιών, όσο και η μονολιθική αρχιτεκτονική έχουν τα δικά τους προτερήματα και μειονεκτήματα και η κάθε μια μπορεί να είναι πιο αποδοτική προσέγγιση από την άλλη, ανάλογα με τις ανάγκες μιας εφαρμογής.

Η παρούσα έρευνα παρουσιάζει σύγκριση των δυο αυτών αρχιτεκτονικών, με τη πειραματική αξιολόγηση μιας εφαρμογής η οποία υλοποιήθηκε και σαν μονολιθική και με τη χρήση μικρουπηρεσιών. Τα αποτελέσματα της έρευνας έδειξαν ότι η αρχιτεκτονική μικρουπηρεσιών ανταποκρίνεται καλύτερα σε πολλαπλά παράλληλα αιτήματα. Αντιθέτως, η μονολιθική εφαρμογή δείχνει καλύτερη απόδοση κατά την κλιμάκωση.

Τα συμπεράσματα αυτά ενισχύονται από την κατανόηση των βασικών ιδιοτήτων και των σχετικών επιπτώσεων κάθε αρχιτεκτονικής. Συγκεκριμένα, μέσα από μια σειρά πειραμάτων έχει γίνει εμφανές η δυνατότητα των μικρουπηρεσιών να χειριστούν πιο αποτελεσματικά τα εισερχόμενα αιτήματα στις περιπτώσεις που έχουμε έντονους υπολογισμούς και μεγάλο αριθμό νημάτων και παράλληλων συνδέσεων και απαιτείται κλιμάκωση της εφαρμογής. Η

αρχιτεκτονική των μικρουπηρεσιών, με την αποκεντρωμένη και αυτόνομη φύση της, μπορεί να οδηγήσει σε χαμηλότερη καθυστέρηση, ειδικά όταν απαιτείται κλιμάκωση.

Ταυτόχρονα, η μονολιθική υλοποίηση είναι πιο αποτελεσματική, με μικρότερη καθυστέρηση στις περιπτώσεις που έχουμε μικρό αριθμό παράλληλων συνδέσεων. Η μονολιθική εφαρμογή, με την ενιαία της φύση, μπορεί να βελτιώσει τη ροή δεδομένων και να μειώσει την καθυστέρηση κατά την κλιμάκωση, χάρη στην απλότητα της δομής της και την καλύτερη χρήση των πόρων.

Τα συμπεράσματα αυτής της έρευνας υπογραμμίζουν τη σημασία της επιλογής της κατάλληλης αρχιτεκτονικής λογισμικού, ανάλογα με τις ειδικές ανάγκες και τις απαιτήσεις κάθε συστήματος.

5.2 Μελλοντικές Επεκτάσεις

Η εφαρμογή που έχει χρησιμοποιηθεί για αυτή την έρευνα αποτελεί ένα ολοκληρωμένο σύστημα αναζήτησης προϊόντων σε κινητές καντίνες και διαχείρισης παραγγελιών. Σε μεταγενέστερο στάδιο μπορούν να προκύψουν διάφορες επεκτάσεις μικρού ή μεγάλου βαθμού. Μερικές από αυτές παρατίθενται πιο κάτω:

1. Δημιουργία συνθετικού συνόλου δεδομένων (dataset) για δοκιμές αντοχής (stress testing) της εφαρμογής. Το σύνολο δεδομένων που χρησιμοποιείται από την εφαρμογή είναι περιορισμένο στην πολιτεία του Σαν Φρανσίσκο. Η δημιουργία ενός συνθετικού συνόλου δεδομένων με πληροφορίες για κινητές καντίνες σε τοποθεσίες σε ολόκληρο τον κόσμο μπορεί να δώσει μια καλύτερη κατανόηση για τη συμπεριφορά της εφαρμογής σε μεγάλη κλίμακα, καθώς και για τους περιορισμούς του elasticsearch.

   Ένας τρόπος με τον οποίο θα μπορούσε κάποιος να προσεγγίσει την κατασκευή του συνθετικού συνόλου δεδομένων είναι ο ακόλουθος. Αρχικά πρέπει να οριστούν κάποια γεωγραφικά όρια για τις περιοχές στις οποίες θα τοποθετηθούν οι καντίνες. Στη συνέχεια, με τη βοήθεια κάποιου αλγορίθμου τυχαίας κατανομής, να κατασκευαστεί ένα σύνολο με γεωγραφικές συντεταγμένες μέσα σε αυτά τα όρια.

Τέλος, με τη βοήθεια κάποιας βιβλιοθήκης κατασκευής εικονικών δεδομένων, όπως η faker.js, να κατασκευαστούν τα στοιχεία της κάθε καντίνας, όπως όνομα και τα αντικείμενα που προσφέρει. Το σύνολο δεδομένων μπορεί να φορτωθεί στο elasticsearch και να χρησιμοποιηθεί από την εφαρμογή.

2. Ενσωμάτωση συστήματος για τη παρακολούθηση συναλλαγών μεταξύ των μικρουπηρεσιών. Με τη παρακολούθηση των συναλλαγών μεταξύ των υπηρεσιών μπορούμε να κατανοήσουμε καλύτερα τον τρόπο λειτουργίας του συστήματος και να εντοπίσουμε πιθανά σημεία συμφόρησης (bottleneck) στην εφαρμογή. Ένα παράδειγμα ενός τέτοιου εργαλείου είναι το Jaeger, το οποίο αποτελεί ένα σύστημα ανοικτού κώδικα για την παρακολούθηση συναλλαγών μεταξύ διανεμημένων συστημάτων.

3. Δοκιμή κάποιας άλλης βάσης δεδομένων, όπως κατανεμημένες βάσης δεδομένων ή αποθηκεύσεις ζευγών κλειδιού-τιμής (key-value stores). Οι κατανεμημένες βάσεις δεδομένων μπορούν να κλιμακωθούν και επιτρέπουν να εξεταστεί κατά πόσο η βάση δεδομένων αποτελεί σημείο συμφόρησης για την εφαρμογή. Ταυτόχρονα, τα αποθηκευτικά συστήματα κλειδιού-τιμής τείνουν να είναι πιο γρήγορα, αφού δεν απαιτούν πολύπλοκα ερωτήματα και μπορούν να διαχειριστούν μεγάλο όγκο δεδομένων.

Βιβλιογραφία

[Αγγελόπουλος2020] Αγγελόπουλος, Σ. (2020). Τεχνικές για ανάπτυξη εφαρμογής με χρήση Microservices (Doctoral dissertation, University of Piraeus (Greece)).

[Delimitrou] https://github.com/delimitrou/DeathStarBench

[Docker] docs.docker.com

[FudanSELab] https://github.com/FudanSELab/train-ticket

[Gan2019] Y. Gan., ASPLOS (2019). An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems.

[Gos2020] Gos, K., & Zabierowski, W. (2020, April). The comparison of microservice and monolithic architecture. In 2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH) (pp. 150-153). IEEE.

[nmikal98A] https://github.com/nmikal98/FoodTruckApp_Monolith.git

[nmikal98B] https://github.com/nmikal98/FoodTruckApp_Microservices_gRPC.git

[Prakhar1989] https://github.com/prakhar1989/FoodTrucks

[Schall2022] Schall D (2022). Lukewarn Serverless Functions: Characterization and Optimization

[Shutian2021] Shutian L (2021). Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis

[Sriraman2018] Sriraman Akshita & Wensich F. Thomas (2018). μSuite: A Benchmark Suite for Microservices.

[Swani2017] Swani, L., & Tyagi, P. (2017). Dockerization (Replacement of VMs).

[vhive-serverless] https://github.com/vhive-serverless/vSwarm

[Wenischlab] https://github.com/wenischlab/MicroSuite

[Zhou2019] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding (2019). Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study.