ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
UNIVERSITY OF PIRAEUS

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Διατμηματικό Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένες Τεχνολογίες Πληροφορικής και Υπηρεσίες»

*Κατανεμημένα Συστήματα και Υπολογιστικά Νέφη*

# Η συμπεριφορά της υπηρεσίας υπολογιστικής νέφους σε συγκεκριμένο φορτίο και υπό την ανάπτυξη υπηρεσίας με την αρχιτεκτονική των microservices εντός σμήνους από containers

28/4/2024

# Contents

# Εισαγωγή

Στην παρούσα μελέτη θα αναλυθεί το φορτίο (επισκεψιμότητα) που δέχεται μία εφαρμογή η οποία βρίσκεται σε νεφοϋπολογιστική υποδομή, συγκεκριμένα μια εφαρμογή τύπου ηλεκτρονικού καταστήματος, και θα γίνει αναφορά στους λόγους για τους οποίους η αξιοποίηση υπηρεσίας αρχιτεκτονικής microservices σε containers μπορεί να βελτιστοποιήσει την απόδοση της.

# Γιατί Microservices;

Μέχρι πριν μερικά χρόνια, ο τρόπος που οι προγραμματιστές δημιουργούσαν τις εφαρμογές τους ήταν σαν ένα σύνολο, μία ολότητα όπου τα διαφορετικά κομμάτια, το frontend, το backend, η βάση δεδομένων κλπ. ανήκουν στο ίδιο codebase, οι λεγόμενες μονολιθικές υπηρεσίες. Μία διαφορετική προσέγγιση από αυτή την αρχιτεκτονική είναι τα microservices, όπου η κάθε υπηρεσία είναι απομονωμένη από τις υπόλοιπες. Κάποια μόνο από τα θετικά αυτής είναι πως πλέον η κάθε υπηρεσία, όντας απομονωμένη από τις υπόλοιπες, μπορεί να τρέξει σε ξεχωριστό και καταλληλότερο hardware, είναι ευκολότερο για τους software engineers να συντηρήσουν ή/και να κάνουν αλλαγές τον κώδικα χωρίς να επηρεαστούν αρνητικά οι υπόλοιπες καθώς και, μεταξύ άλλων, αυξάνεται το fault tolerance[1]. Στην περίπτωση για παράδειγμα μιας εφαρμογής e-shop, η αυθεντικοποίηση του χρήστη, το καλάθι, το σύστημα πληρωμής καθώς οι υπόλοιπες υπηρεσίες μπορούν να είναι απομονωμένες. Γιατί όμως οδηγηθήκαμε σε αυτήν την εξέλιξη; Ένας από τους σημαντικότερους λόγους είναι η αύξηση της πολυπλοκότητας των εφαρμογών, και κατ' επέκταση η ανάγκη περεταίρω βελτίωση τους. Η κάθε υπηρεσία μπορεί να αναπτυχθεί, να δεχτεί αλλαγές και να γίνει deploy ανεξαρτήτως από τις υπόλοιπες[2] ενώ όντας απομονωμένη, είναι λιγότερο πιθανό η οποιαδήποτε αλλαγή σε αυτήν να επιφέρει αρνητικές συνέπειες και να δημιουργήσει bugs στις υπόλοιπες.

# Τεχνολογίες και η εφαρμογή που χρησιμοποιήθηκε

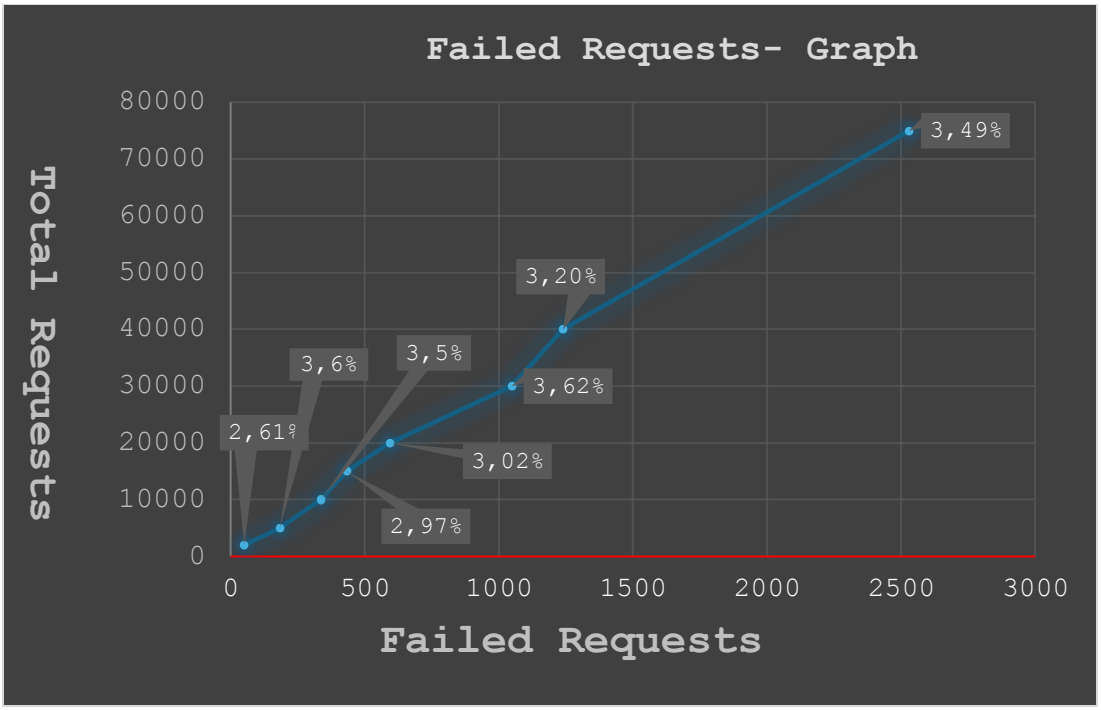
Μία από τις τεχνολογίες η οποία μας επιτρέπει να προχωρήσουμε σε αυτήν την απομόνωση μεταξύ των υπηρεσιών που συζητήθηκε, και η οποία χρησιμοποιήθηκε στην παρούσα μελέτη, είναι το Docker[3] και Docker Compose[3], μια open-source πλατφόρμα μέσω της οποίας μπορεί ο προγραμματιστής να δημιουργήσει και να τρέξει "containers" – απομονωμένες δηλαδή υπηρεσίες. Αυτά έχουν ικανότητες δικτύωσης[4], και έτσι επιτυγχάνεται η μεταξύ τους επικοινωνία.

Η εφαρμογή που χρησιμοποιήθηκε και στην οποία έγιναν μετρήσεις φόρτου είναι ένα ηλεκτρονικό κατάστημα[5] με υπηρεσίες καταλόγου, σύνδεσης χρηστών, πληρωμής, παραγγελιών,

καλαθιού κ.α. γραμμένα σε αρκετές διαφορετικές γλώσσες όπως GO!, Shell Script, JavaScript, Java κ.α. Παράλληλα, η εφαρμογή μας προσφέρει το load test, ένα python script γραμμένο με Locust framework το οποίο προσομοιώνει το traffic στον ιστότοπο του καταστήματος. Για παράδειγμα μπορεί να προσομοιώσει 100 επισκέπτες που κάνουν 20000 requests με τους αριθμούς αυτούς να είναι παραμετροποιήσιμοι, ενώ το περιβάλλουν στο οποίο τρέχει η εφαρμογή είναι Ubuntu Linux 16.06 LTS Server εντός της νεφοϋπολογιστικής υποδομή Okeanos[6]

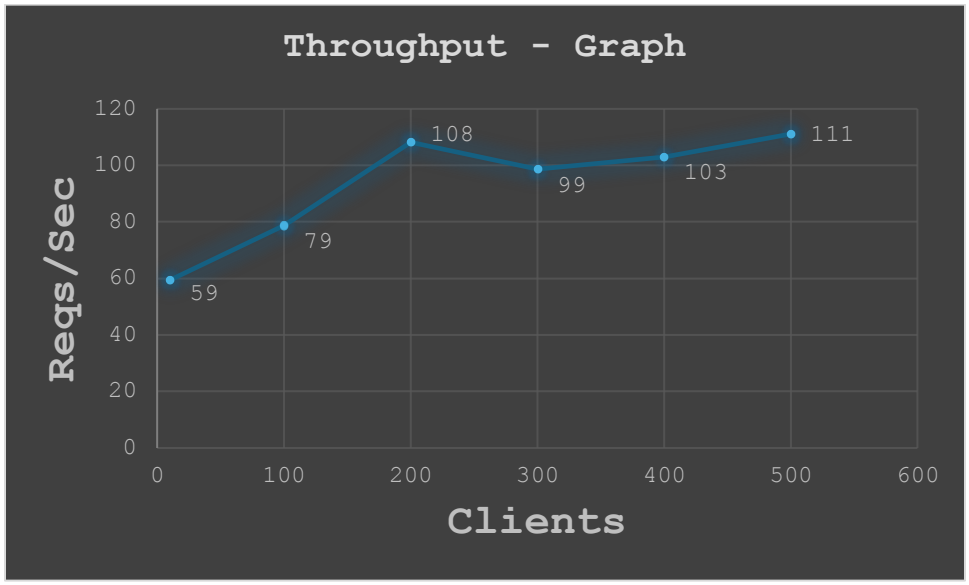
## Μετρήσεις φορτίου

Αφού λοιπόν δημιουργήσαμε μια εικονική μηχανή στην παραπάνω υποδομή και η οποία φιλοξενεί τις υπηρεσίες του ηλεκτρονικού καταστήματος με την αρχιτεκτονική microservices, προχωράμε στην πραγματοποίηση του load test. Συγκεκριμένα, στο πρώτο test θα μελετήσουμε το failure rate, δηλαδή πόσα από τα 2000, 5000, 10000…., 75000 αιτήματα πραγματοποιούνται με επιτυχία σε περίπτωση που έχουμε 100 επισκέπτες ταυτόχρονα.




Στο παραπάνω γράφημα παρατηρούμε στον κάθετο άξονα τα συνολικά αιτήματα των χρηστών, ενώ στον οριζόντιο τα αιτήματα στα οποία ο εξυπηρετητής δεν ανταποκρίθηκε. Παράλληλα σημειώνεται η αποτυχία σε ποσοστό για κάθε περίπτωση. Φαίνεται λοιπόν ότι το ο αριθμός σφαλμάτων μένει σχετικά σταθερός στο 3% με μία απόκλιση περίπου +- 0.6% είτε έχουμε λίγα αιτήματα (10000), είτε έχουμε αρκετά περισσότερα (75000).

Το επόμενο γράφημα πραγματεύεται την απόδοση της εφαρμογής μας. Συγκεκριμένα βλέπουμε τις περιπτώσεις που ο εξυπηρετητής δέχεται 25000 αιτήματα από 10, 100, 200, 300, 400 και 500 χρήστες και πόσα από αυτά τα αιτήματα μπορεί να επεξεργαστεί το δευτερόλεπτο. Βλέπουμε ότι όσο οι χρήστες είναι πάνω από 200, ο εξυπηρετητής μπορεί να εξυπηρετήσει περίπου 100-110 αιτήματα το δευτερόλεπτο, φαίνεται αυτό να είναι ένα όριο το οποίο δεν μπορεί να ξεπεραστεί. Όσο όμως οι χρήστες είναι λιγότεροι, δηλαδή κάτω από 200, τότε ο αριθμός των αιτημάτων που εξυπηρετούνται μειώνεται στα 79 και 59 για 100 και 10 χρήστες αντίστοιχα. Φαίνεται λοιπόν ότι όσο οι χρήστες είναι λίγοι, το hardware της εικονικής μηχανής δεν αξιοποιείται στο έπακρο. Αυτό μπορούμε να το εκμεταλλευτούμε, σε συνδυασμό με το scalability που μας προσφέρει μια νεφοϋπολογιστική υπηρεσία, ώστε πάντα να χρησιμοποιούμε, και άρα να πληρώνουμε, όσους ακριβώς πόρους χρειαζόμαστε, όχι περισσοτέρους. Πόρους όπως πυρήνες επεξεργαστών, μνήμη, network bandwidth και χώρου αποθήκευσης.



# Βιβλιογραφικές Αναφορές


[1] https://www.sciencedirect.com/science/article/pii/S0950584921000793

[2] https://link.springer.com/chapter/10.1007/978-3-319-74433-9_3

[3] https://www.docker.com/

[4] https://docs.docker.com/network/

[5] https://web.archive.org/web/20210925113541/https://microservices-demo.github.io/docs/

[6] https://okeanos.grnet.gr/home/