ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΑΠΟΤΙΜΗΣΗ ΕΠΙΔΟΣΗΣ ΑΥΤΟΜΑΤΗΣ ΚΛΙΜΑΚΩΣΗΣ ΜΙΚΡΟΥΠΗΡΕΣΙΩΝ ΣΕ ΥΠΟΛΟΓΙΣΤΙΚΑ ΝΕΦΗ ΜΕ ΧΡΗΣΗ ΠΕΡΙΕΚΤΩΝ

ΛΕΡΙΟΥ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Κωνσταντίνου Ιωάννης
Αναπληρωτής Καθηγητής

Λαμία, Οκτώβριος 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΑΠΟΤΙΜΗΣΗ ΕΠΙΔΟΣΗΣ ΑΥΤΟΜΑΤΗΣ ΚΛΙΜΑΚΩΣΗΣ ΜΙΚΡΟΥΠΗΡΕΣΙΩΝ ΣΕ ΥΠΟΛΟΓΙΣΤΙΚΑ ΝΕΦΗ ΜΕ ΧΡΗΣΗ ΠΕΡΙΕΚΤΩΝ

ΛΕΡΙΟΥ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Κωνσταντίνου Ιωάννης
Αναπληρωτής Καθηγητής

Λαμία, Οκτώβριος 2025

UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

Performance Evaluation of Autoscaling Microservices in Cloud Computing Using Containers

LERIOU KONSTANTINOS

FINAL THESIS

ADVISOR

Ioannis Konstantinou
Associate Professor

Lamia, October 2025

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

*1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

*2. Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ.), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: 13/10/2025

Ο Δηλ.

Λερίου Κωνσταντίνος

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση
του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων
σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ΠΕΡΙΛΗΨΗ

Η συγκεκριμένη διπλωματική εργασία αναφέρεται στην παρατήρηση της διεργασίας της ελαστικής διαχείρισης πόρων. Πιο συγκεκριμένα χρησιμοποιώντας διάφορα ψηφιακά εργαλεία, όπως το Docker και το Kubernetes. Σε αυτή την εργασία έπειτα από μια σύντομη αναφορά στο θεωρητικό υπόβαθρο, γίνεται ανάλυση των παρατηρήσεων της απόδοσης και συμπεριφοράς των συστημάτων με την χρήση ενός σετ από μικρό-υπηρεσίες για benchmarking, το οποίο ονομάζεται DeathStarBench. Το Jaeger Tracing χρησιμοποιήθηκε επίσης για την ανάλυση των διαφορών καθυστέρησης μεταξύ των pods, επιτρέποντας λεπτομερή παρατήρηση των μεταβολών της απόδοσης σε επίπεδο χιλιοστών του δευτερολέπτου . Τέλος γίνεται σύγκριση μεταξύ της απόδοσης χωρίς κάποιο autoscaler και με την χρήση του HPA, το οποίο είναι ένα feature του Κυβερνήτη που αυξομειώνει αυτόματα τον ρυθμό των pods βάσει τον φόρτο.

ABSTRACT

This thesis focuses on the observation of the process of elastic resource management, utilizing various digital tools such as Docker and Kubernetes. Following a brief overview of the theoretical background, the thesis presents an in-depth analysis of the performance and behavior of systems using a benchmarking microservices suite known as DeathStarBench. Jaeger Tracing was also utilized to examine the differences in response times between individual pods, measured in milliseconds. Finally, a comparison is made between system performance with no autoscaler and with the use of the Horizontal Pod Autoscaler (HPA), a Kubernetes feature that automatically scales the number of pods based on workload.

Πίνακας περιεχομένων

ΚΕΦΑΛΑΙΟ 1

1.1 Εισαγωγή

Στις μέρες μας η ανάγκη για βελτιστοποίηση της χρήσης των υπολογιστικών πόρων (CPU, μνήμη, storage κλπ.) έχει φέρει στο προσκήνιο την χρήση της ελαστικής διαχείρισης πόρων*[1]*. Κυρίοι λόγοι χρήσης της είναι η αντιμετώπιση μεταβαλλόμενου φορτιού, η αποδοτική χρήση υποδομών, η αυξημένη διαθεσιμότητα και σταθερότητα και η βελτίωση της εμπειρίας του χρήστη. Οι εφαρμογές δεν έχουν σταθερό αριθμό χρηστών και έτσι με την ελαστικότητα το σύστημα προσαρμόζει αυτόματα τους πόρους του ανάλογα με τις ανάγκες, δηλαδή χρησιμοποιεί περισσοτέρους πόρους όταν έχει μεγαλύτερο φόρτο και λιγότερους όταν ο φόρτος είναι χαμηλός. Με αυτόν τον τρόπο εξοικονομεί χρήματα και ενέργεια γιατί οι πόροι αυξομειώνονται και προσαρμόζονται αυτόματα. Ταυτόχρονα όταν ο αριθμός των χρηστών αυξάνεται απότομα, το σύστημα αντέχει χωρίς να καταρρέει και αποφεύγει τυχών καθυστερήσεις. Επίσης ο χρήστης έχει σταθερή και γρήγορη απόκριση της εφαρμογής ανεξάρτητα του φόρτου που εκείνη έχει. Ένα χαρακτηριστικό παράδειγμα εφαρμογής ελαστικότητας αποτελεί η χρήση μηχανισμών αυτόματης κλιμάκωσης, όπως το ***Horizontal Pod Autoscaler*** (HPA)*[9]*, ο οποίος μπορεί να αυξάνει ή να μειώνει αυτόματα τα στιγμιότυπα μιας υπηρεσίας, με βάση δείκτες όπως η CPU ή άλλα metrics.

1.2 Πλαίσιο Υλοποίησης

Στη συγκεκριμένη πτυχιακή εργασία προτείνεται ένα πλαίσιο για τη δυναμικότερη και αποδοτικότερη διαχείριση υπολογιστικών πόρων σε εφαρμογές που βασίζονται σε αρχιτεκτονική μικροϋπηρεσιών. Το πλαίσιο αυτό, αξιοποιεί τις δυνατότητες αυτόματης οριζόντιας κλιμάκωσης που προσφέρει το Kubernetes, προσαρμόζοντας τον αριθμό των στιγμιότυπων κάθε υπηρεσίας (pods) σύμφωνα με τις πραγματικές ανάγκες του συστήματος. Στόχος είναι η διατήρηση της βέλτιστης λειτουργικότητας και η αποδοτική χρήση των διαθέσιμων υποδομών.

Τα βασικά χαρακτηριστικά του προτεινόμενου πλαισίου είναι :

- **Αυτόματη Οριζόντια Κλιμάκωση Πόρων**

Το πλαίσιο ενσωματώνει τον μηχανισμό Horizontal Pod Autoscaler (HPA), ο οποίος ρυθμίζει αυτόματα τον αριθμό των pods κάθε μικροϋπηρεσίας με βάση μετρήσεις όπως η χρήση της CPU ή άλλες καθορισμένες μετρικές απόδοσης. Με αυτόν τον τρόπο, το σύστημα προσαρμόζεται άμεσα στις μεταβολές του φορτίου εργασίας.

- **Ενισχυμένη Ελαστικότητα και Απόδοση**

Η δυναμική προσαρμογή του αριθμού των pods επιτρέπει στο σύστημα να ανταποκρίνεται σε αυξημένες απαιτήσεις χωρίς καθυστερήσεις ή υποβάθμιση της ποιότητας υπηρεσίας. Ταυτόχρονα, μειώνει την κατανάλωση πόρων σε περιόδους χαμηλής ζήτησης, προωθώντας την αποδοτικότερη απόκριση.

- **Παρακολούθηση και Ανάλυση Μετρικών**

Το σύστημα συνοδεύεται από μηχανισμούς παρακολούθησης σε πραγματικό χρόνο, οι οποίες βασίζονται σε εργαλεία συλλογής και απεικόνισης μετρικών όπως το Jaeger, για την ανάλυση της απόδοσης των μικροϋπηρεσιών και την έγκαιρη λήψη αποφάσεων.

- **Αποδοτική Χρήση Πόρων και Εξοικονόμηση**

Μέσω της οριζόντιας αυτόματης κλιμάκωσης, αποφεύγεται η υπερεκχώρηση πόρων, εξοικονομούνται υπολογιστικοί πόροι και ενέργεια, και ενισχύεται η σταθερότητα και αξιοπιστία του συστήματος, ακόμα και σε συνθήκες μεταβαλλόμενου φόρτου.

1.3 Θεωρητικό υπόβαθρο

Στο παρόν κεφάλαιο παρέχονται αναλυτικές πληροφορίες σχετικά με τις βασικές έννοιες, τεχνολογίες και αρχές που σχετίζονται με την ελαστική διαχείριση υπολογιστικών πόρων, όπως αυτές αξιοποιούνται και αναλύονται στο πλαίσιο της παρούσας πτυχιακής εργασίας. Το θεωρητικό αυτό υπόβαθρο αποσκοπεί στο να προσφέρει στον αναγνώστη μια πλήρη και στέρεη κατανόηση των εννοιών που συνιστούν το θεμέλιο της παρούσας μελέτης, προκειμένου να διευκολυνθεί η κατανόηση των μεθοδολογιών, των τεχνικών λύσεων και των αποτελεσμάτων που θα παρουσιαστούν στα επόμενα κεφάλαια.

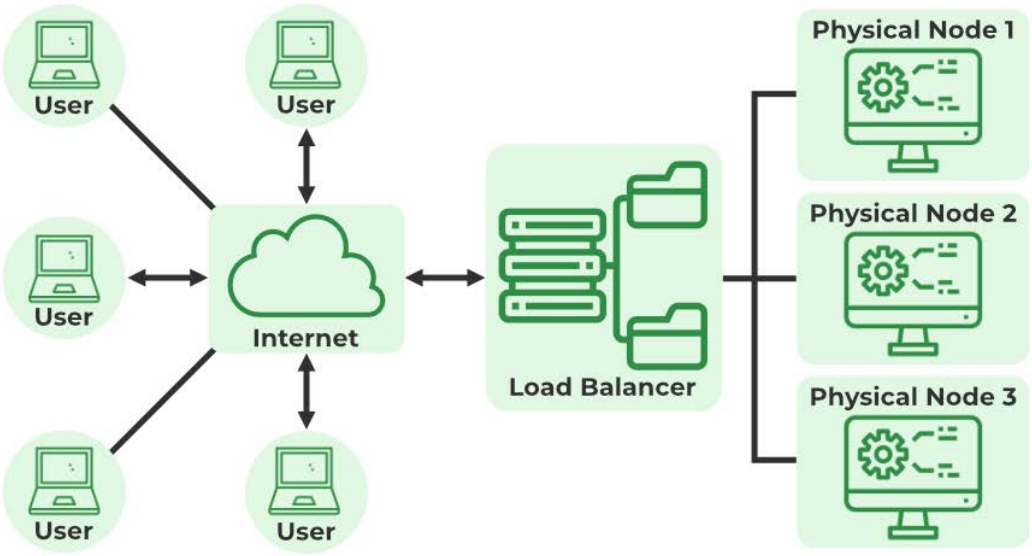
Η δομή του κεφαλαίου περιλαμβάνει επιμέρους ενότητες που καλύπτουν το φάσμα των σχετικών τεχνολογιών, από τη βασική θεωρία των υπολογιστικών πόρων και την έννοια της ελαστικότητας, έως πιο σύγχρονες πρακτικές ενορχήστρωσης μέσω πλατφορμών όπως το Kubernetes, αλλά και την αξιοποίηση εργαλείων παρακολούθησης και προβλεπτικής ανάλυσης. Μέσα από αυτή την επισκόπηση, διασφαλίζεται ότι ο αναγνώστης μπορεί να προσεγγίσει την υπόλοιπη εργασία με ενιαίο εννοιολογικό και τεχνολογικό υπόβαθρο.

1.3.1 Υπολογιστικοί Πόροι

Με τον όρο υπολογιστικοί πόροι αναφερόμαστε στο σύνολο των τεχνικών και φυσικών μέσων που απαιτούνται για την εκτέλεση υπολογιστικών διεργασιών

σε ένα πληροφοριακό σύστημα. Μερικά παραδείγματα υπολογιστικών πόρων είναι η κεντρική μονάδα επεξεργασίας (CPU) η οποία εκτελεί τις εντολές των προγραμμάτων και διαχειρίζεται βασικές λειτουργίες του συστήματος, η μνήμη (RAM) η οποία παρέχει προσωρινή αποθήκευση για τα δεδομένα και τις εντολές που χρησιμοποιούνται κατά την εκτέλεση των προγραμμάτων. Ακόμα ο Αποθηκευτικός χώρος (Disk/HDD/SSD), διατηρεί δεδομένα και εφαρμογές για μακροχρόνια χρήση. Τέλος, το Δίκτυο (Network), είναι ένα παράδειγμα το οποίο χρειάζεται για την επικοινωνία μεταξύ υπηρεσιών και εξωτερικών συστημάτων. Στο παρελθόν, οι πόροι αυτοί ήταν άμεσα δεσμευμένοι σε φυσικούς servers, με στατική κατανομή. Στη σύγχρονη εποχή όμως, με την εξάπλωση των τεχνολογιών εικονικοποίησης (virtualization) και περιεκτών (Containers), η ανάγκη για δυναμική και ελαστική διαχείριση πόρων είναι πιο έντονη από ποτέ.

Η ελαστικότητα (elasticity) αναφέρεται στην ικανότητα ενός υπολογιστικού συστήματος να προσαρμόζει αυτόματα τους πόρους του ανάλογα με τις απαιτήσεις φόρτου, βελτιώνοντας έτσι την αποδοτικότητα και μειώνοντας το κόστος. Η διαχείριση των πόρων γίνεται πλέον δυναμικά, συχνά με αυτοματοποιημένα εργαλεία και πλατφόρμες που επιτρέπουν την κλιμάκωση (scaling) των εφαρμογών ανάλογα με τις ανάγκες.



Εικόνα 1.0 *Πηγή εικόνας:* ***https://www.geeksforgeeks.org/cloud-computing/scalability-and-elasticity-in-cloud-computing***

1.3.2 Μικροϋπηρεσίες (Microservices)

Η αρχιτεκτονική μικροϋπηρεσιών αποτελεί μια σύγχρονη προσέγγιση στη σχεδίαση και ανάπτυξη λογισμικού, όπου μια εφαρμογή διαχωρίζεται σε ένα σύνολο από ανεξάρτητες, μικρές υπηρεσίες. Κάθε μία από αυτές είναι υπεύθυνη

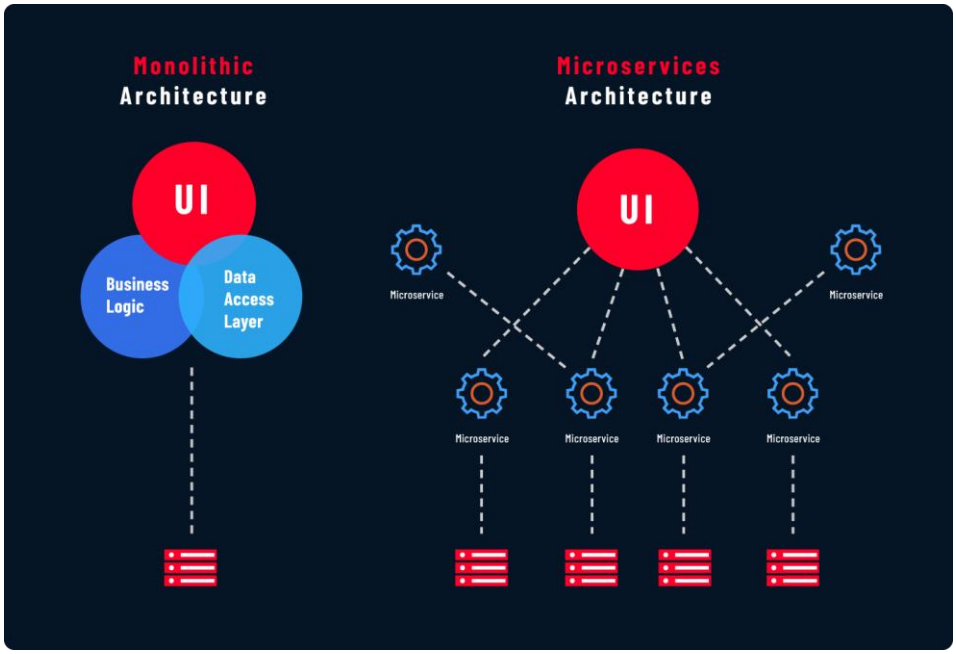
για την υλοποίηση μιας συγκεκριμένης επιχειρησιακής λειτουργίας και μπορεί να αναπτυχθεί, να εξελιχθεί και να διαχειριστεί αυτόνομα. Ο κατακερματισμός της εφαρμογής σε διακριτές μονάδες ενισχύει σημαντικά την ευελιξία και την επεκτασιμότητα του συστήματος, επιτρέποντας στις ομάδες ανάπτυξης να εργάζονται παράλληλα και να υλοποιούν αλλαγές χωρίς να επηρεάζεται το συνολικό οικοσύστημα (Dragoni et al., 2017)[2].

Οι μικροϋπηρεσίες επικοινωνούν μεταξύ τους μέσω ελαφριών και καλά ορισμένων διεπαφών, όπως RESTful APIs ή μηνυματοκεντρικά πρωτόκολλα. Αυτός ο τρόπος επικοινωνίας μειώνει τις εξαρτήσεις μεταξύ των υπηρεσιών, ενισχύοντας την απομόνωση των πιθανών σφαλμάτων και συμβάλλοντας στην αυξημένη ανθεκτικότητα του συστήματος (Fowler & Lewis, 2014)[3]. Επιπλέον, κάθε υπηρεσία μπορεί να επιλέξει την κατάλληλη τεχνολογία υλοποίησης με βάση τις δικές της απαιτήσεις, γεγονός που προσδίδει μεγαλύτερη ευελιξία στις τεχνικές αποφάσεις.

Σε σύγκριση με τις μονολιθικές αρχιτεκτονικές, όπου η εφαρμογή υλοποιείται ως μια ενιαία και αδιαίρετη μονάδα, οι μικροϋπηρεσίες ακολουθούν μια περισσότερο κατανεμημένη προσέγγιση. Στις μονολιθικές εφαρμογές, η οποιαδήποτε αλλαγή σε ένα υποσύστημα μπορεί να απαιτεί την επανασύσταση και αναδιάταξη ολόκληρης της εφαρμογής, κάτι που ενδέχεται να επιβραδύνει την ανάπτυξη και να περιορίζει τη δυνατότητα κλιμάκωσης (Richardson, 2018)[4]. Αντίθετα, η φιλοσοφία των μικροϋπηρεσιών επιτρέπει την ανεξάρτητη ανάπτυξη και διάθεση νέων εκδόσεων συγκεκριμένων υπηρεσιών, χωρίς να απαιτείται η αναδιαμόρφωση του συνόλου.

Ωστόσο, η μετάβαση σε μια αρχιτεκτονική μικροϋπηρεσιών δεν είναι απαλλαγμένη προκλήσεων. Η πολυπλοκότητα μεταφέρεται από τον πηγαίο κώδικα στην υποδομή, καθώς απαιτείται προσεκτικός σχεδιασμός της επικοινωνίας μεταξύ των υπηρεσιών, της διαχείρισης της συνέπειας των δεδομένων και της παρακολούθησης της απόδοσης σε ένα κατανεμημένο περιβάλλον (Taibi et al., 2020)[5]. Η υιοθέτηση μηχανισμών όπως η παρακολούθηση (monitoring), η ανίχνευση (tracing) και η συλλογή μετρικών (metrics) καθίσταται κρίσιμη για τη διαχείριση της συνολικής πολυπλοκότητας του συστήματος.

Συνολικά, η επιλογή της αρχιτεκτονικής μικροϋπηρεσιών έναντι της μονολιθικής, εξαρτάται σε μεγάλο βαθμό από τη φύση και την πολυπλοκότητα της εκάστοτε εφαρμογής, τις απαιτήσεις κλιμάκωσης, καθώς και τη δυνατότητα της αναπτυξιακής ομάδας να διαχειριστεί τα χαρακτηριστικά ενός κατανεμημένου συστήματος. Παρά τις προκλήσεις, η αρχιτεκτονική μικροϋπηρεσιών έχει καθιερωθεί τα τελευταία χρόνια ως ένα από τα κυρίαρχα πρότυπα σχεδίασης για την ανάπτυξη σύγχρονων, ευέλικτων και επεκτάσιμων εφαρμογών.



Εικόνα 1.1 *Πηγή εικόνας:* ***https://blog.sparkfabrik.com/hs-fs/hubfs/Blog/monolothic-microservices-image-articles.png?name=monolothic-microservices-image-articles.png&width=1391***

1.3.3 Docker

Το Docker αποτελεί μια τεχνολογία ανοιχτού κώδικα που επιτρέπει την απομόνωση και πακετάρισμα λογισμικού σε containers, προσφέροντας έτσι έναν ελαφρύ και φορητό τρόπο εκτέλεσης εφαρμογών σε οποιοδήποτε περιβάλλον. Η χρήση του Docker είναι πλέον στενά συνδεδεμένη με την αρχιτεκτονική μικροϋπηρεσιών, καθώς κάθε μικροϋπηρεσία μπορεί να εκτελείται σε δικό της απομονωμένο κοντέινερ, διασφαλίζοντας τη συνέπεια και την ευκολία διαχείρισης.

Ένα Docker κοντέινερ περιέχει όλα τα απαραίτητα στοιχεία για την εκτέλεση μιας εφαρμογής: κώδικα, runtime, βιβλιοθήκες, εξαρτήσεις, ακόμα και το λειτουργικό σύστημα σε ελαφριά μορφή, γεγονός που το καθιστά πλήρως φορητό. Η βασική διαφορά με τις εικονικές μηχανές (VMs) είναι πως τα containers μοιράζονται τον πυρήνα του λειτουργικού συστήματος του host, με αποτέλεσμα να είναι ελαφρύτερα και ταχύτερα στην εκκίνηση (Merkel, 2014)***[6]***.

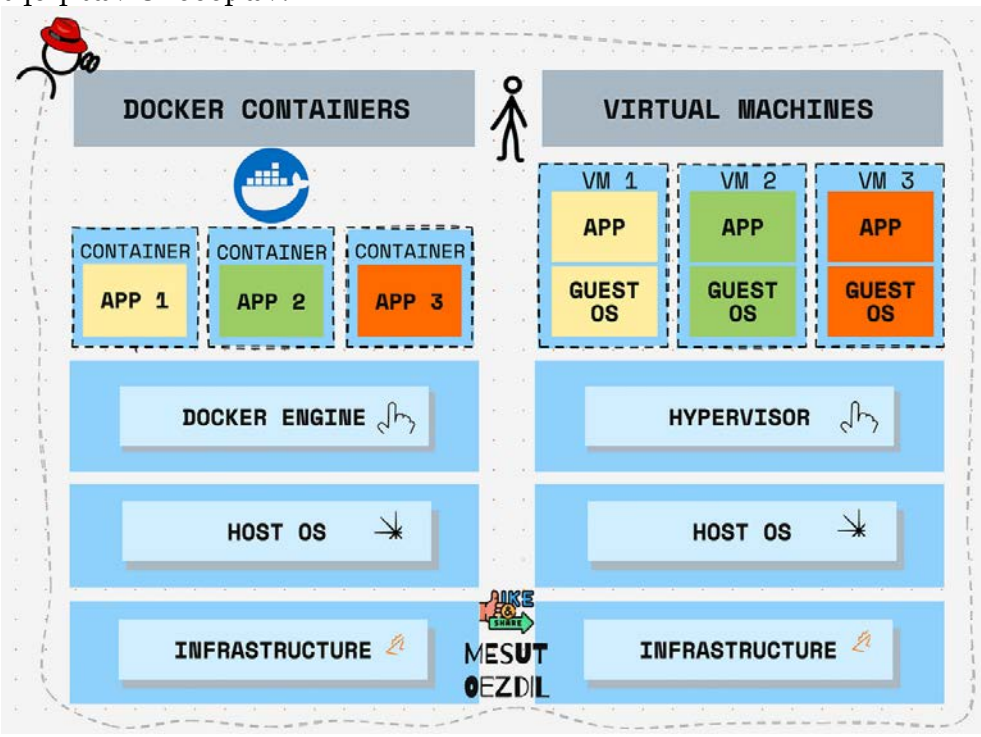
Η χρήση του Docker φέρνει σημαντικά πλεονεκτήματα:

- **Φορητότητα**: Τα containers μπορούν να εκτελεστούν με τον ίδιο τρόπο σε διαφορετικά περιβάλλοντα (development, staging, production).
- **Απομόνωση**: Κάθε κοντέινερ λειτουργεί απομονωμένα, κάτι που ενισχύει την ασφάλεια και τη σταθερότητα.

- **Κλιμάκωση**: Η συμβατότητα του Docker με τα συστήματα ορχήστρωσης (π.χ. Kubernetes) διευκολύνει την αυτόματη κλιμάκωση και διαχείριση εφαρμογών

Μια βασική έννοια στο Docker είναι το Docker Image, ένα «στατικό» στιγμιότυπο με όλα τα απαραίτητα για την εκτέλεση, το οποίο μπορεί να αναπαραχθεί πολλαπλές φορές ως Docker Container. Ο σχεδιασμός των images βασίζεται σε Docker files, που περιγράφουν βήμα-βήμα τις οδηγίες για την κατασκευή του περιβάλλοντος.

Η συμβολή του Docker στην ελαστική διαχείριση πόρων είναι κρίσιμη, καθώς επιτρέπει τη δυναμική εκκίνηση ή τον τερματισμό containers με βάση τις ανάγκες της εφαρμογής, χωρίς τα μεγάλα overheads των εικονικών μηχανών. Παράλληλα, σε περιβάλλοντα με ορχηστρώσεις (όπως το Kubernetes), τα containers μπορούν να προγραμματιστούν από την αρχή, σε διαφορετικούς κόμβους συναρτήσει της διαθεσιμότητας των πόρων, προσφέροντας την βέλτιστη αξιοποίηση των υποδομών.



Εικόνα 1.2 *Πηγή εικόνας:* ***https://mesutoezdil.medium.com/docker-containers-and-virtual-machines-explained-3f8a9bbf5a3b***

1.3.4 Kubernetes

Το Kubernetes είναι μια πλατφόρμα ανοιχτού κώδικα η οποία προσφέρει αυτοματοποιημένη διαχείριση, ανάπτυξη, κλιμάκωση και ορχήστρωση containerized εφαρμογών. Αναπτύχθηκε αρχικά από την Google και σήμερα υποστηρίζεται ενεργά από το Cloud Native Computing Foundation (CNCF),

αποτελώντας τη de facto λύση για τη διαχείριση υποδομών που βασίζονται σε κοντέινερ (Burns et al., 2017)*[7]*.

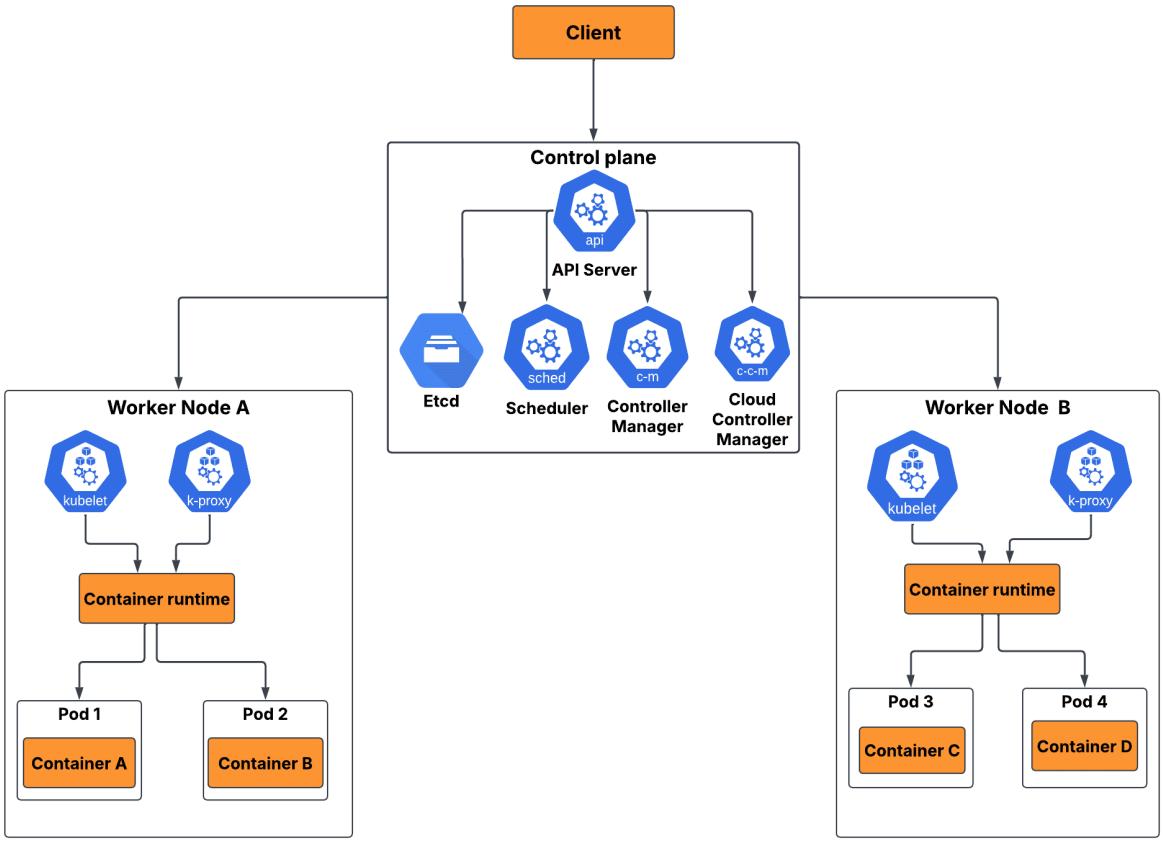
Στον πυρήνα του, το Kubernetes οργανώνει containers σε λογικές ομάδες οι οποίες ονομάζονται pods και επιτρέπει τον δυναμικό προγραμματισμό και την αυτόματη επανεκκίνηση, αντικατάσταση ή ανακατανομή τους σε περίπτωση αποτυχίας. Το σύστημα ακολουθεί το μοντέλο master-node, όπου το control plane λαμβάνει τις αποφάσεις ορχήστρωσης (π.χ. scheduling, scaling, monitoring), ενώ οι worker nodes φιλοξενούν και εκτελούν τα pods.

Μερικές βασικές λειτουργίες και χαρακτηριστικά του Kubernetes περιλαμβάνουν:

- **Αυτόματη κλιμάκωση (Horizontal Pod Autoscaling)**: Προσαρμόζει δυναμικά τον αριθμό των pods βάσει φόρτου εργασίας.
- **Self-healing μηχανισμός**: Τα pods που αποτυγχάνουν είτε τα κάνει επανεκκίνηση είτε τα αντικαθιστά είτε τα μεταφέρει σε διαθέσιμους κόμβους.
- **Load balancing και service discovery**: Ενσωματώνει εσωτερικό DNS και διαχειρίζεται την κατανομή φόρτου μέσω services.
- **Declarative configuration & version control**: Η υποδομή και οι εφαρμογές περιγράφονται μέσω YAML αρχείων, υποστηρίζοντας μέχρι και Infrastructure as Code.
- **Rolling updates & rollbacks**: Επιτρέπει την αναβάθμιση εφαρμογών χωρίς downtime και την επιστροφή σε προηγούμενη έκδοση σε περίπτωση σφαλμάτων.

Η χρήση του Kubernetes καθίσταται κρίσιμη όταν πρόκειται για elastic resource management, αφού προσφέρει μηχανισμούς για τη δυναμική κατανομή υπολογιστικών πόρων, σύμφωνα με τις ανάγκες των workloads. Μέσω εργαλείων όπως το Resource Requests & Limits, μπορεί να επιτευχθεί αποδοτική και ασφαλής κατανομή CPU/Memory πόρων στα pods, αποτρέποντας την υπερκατανάλωση.

Η πλατφόρμα υποστηρίζει πολλαπλούς τρόπους ανάπτυξης (bare-metal, cloud, hybrid), καθώς και plugins για παρακολούθηση, ασφάλεια, δικτύωση και αποθήκευση. Σε περιβάλλοντα με μεγάλες απαιτήσεις διαθεσιμότητας και συνεχούς ανάπτυξης (CI/CD), το Kubernetes αποτελεί βασικό δομικό λίθο για cloud-native αρχιτεκτονικές.



Εικόνα 1.3 *Πηγή εικόνας:* ***https://www.cherryservers.com/blog/kubernetes-architecture***

1.3.5 wrk2

Το wrk2 αποτελεί ένα εργαλείο μέτρησης απόδοσης (benchmarking tool) για HTTP εφαρμογές, σχεδιασμένο να δημιουργεί ιδιαίτερα υψηλό και σταθερό φόρτο εργασίας. Πρόκειται για μια βελτιωμένη έκδοση του εργαλείου wrk, η οποία δίνει έμφαση στη διατήρηση ενός σταθερού ρυθμού αιτημάτων ανά δευτερόλεπτο (requests per second – RPS), γεγονός που το καθιστά ιδιαίτερα κατάλληλο για την αξιολόγηση συστημάτων που υλοποιούν μηχανισμούς αυτόματης κλιμάκωσης και απαιτούν έναν ακριβή έλεγχο του φορτίου (Wang et al., 2017)*[8]*.

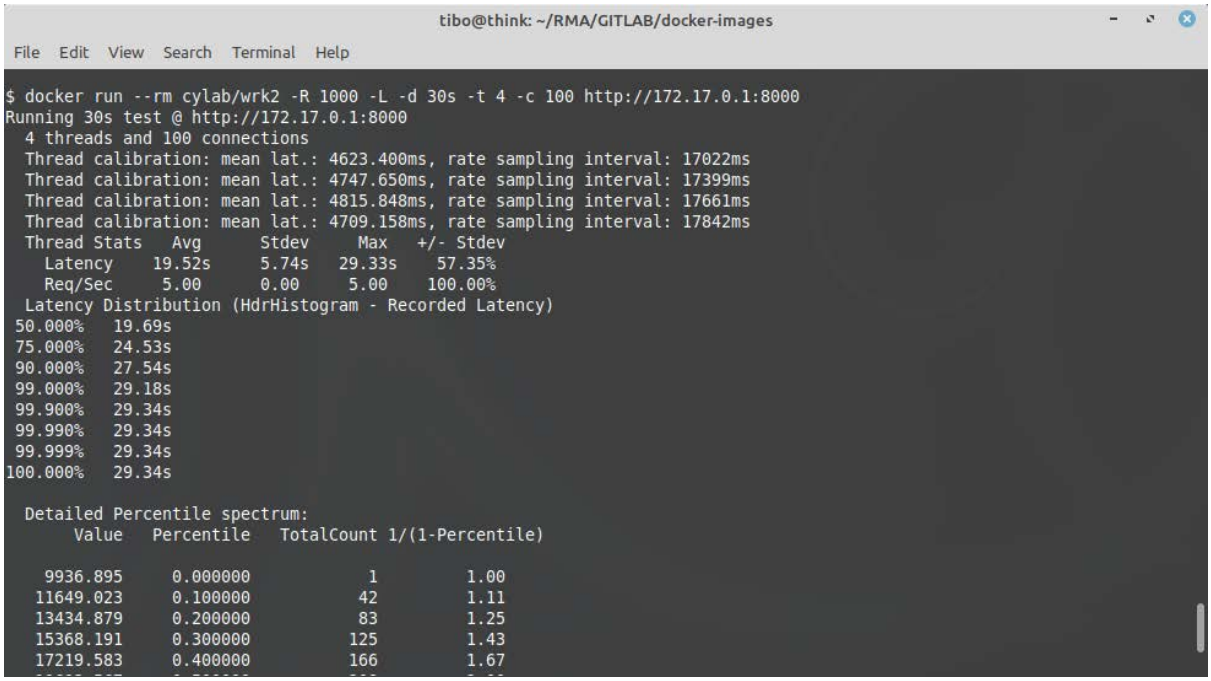
Στο πλαίσιο της παρούσας εργασίας, το wrk2 χρησιμοποιείται εκτενώς για την εκτέλεση stress tests σε μικροϋπηρεσίες, με στόχο τη δημιουργία συνθηκών αυξημένης κατανάλωσης πόρων και πιθανών παραβιάσεων των συμφωνημένων στόχων επιπέδου υπηρεσίας (SLAs). Η προσέγγιση αυτή μας επιτρέπει να παρατηρήσουμε την απόκριση του συστήματος σε συνθήκες έντονου φόρτου και να αξιολογήσουμε την αποτελεσματικότητα των μεθόδων κλιμάκωσης σε ένα ρεαλιστικό περιβάλλον.

Η χρήση του εργαλείου βασίζεται σε μια σειρά παραμέτρων οι οποίες ορίζουν τη διάρκεια, τον αριθμό των νημάτων και των ταυτόχρονων συνδέσεων. Για παράδειγμα, μια τυπική εντολή:

wrk -t12 -c400 -d30s http://127.0.0.1:8080/index.html

εκτελεί ένα πείραμα διάρκειας 30 δευτερολέπτων σε 400 ταυτόχρονες συνδέσεις, οι οποίες κατανέμονται σε 12 νήματα. Η έξοδος που παράγεται περιλαμβάνει μετρικές όπως ο μέσος χρόνος απόκρισης, η τυπική απόκλιση, η κατανομή καθυστερήσεων (π.χ. 50%, 75%, 99% percentiles), καθώς και ο συνολικός ρυθμός αιτημάτων ανά δευτερόλεπτο. Η πληροφορία αυτή είναι κρίσιμη για τον εντοπισμό bottlenecks, την παρακολούθηση της tail latency και την εκτίμηση της συμπεριφοράς του συστήματος υπό πίεση.

Σε αντίθεση με εργαλεία που υιοθετούν "closed-loop" μοντέλο, όπου ο ρυθμός αιτημάτων εξαρτάται ευθέως από τον χρόνο απόκρισης του συστήματος, το wrk2 χρησιμοποιεί "open-loop" προσέγγιση και επιτρέπει τον ακριβή καθορισμό του input load. Αυτό το χαρακτηριστικό το καθιστά ιδανικό για πειράματα που σχετίζονται με την ελαστική διαχείριση πόρων, στα οποία είναι απαραίτητη η αναπαραγωγή σταθερών και μετρήσιμων συνθηκών του φόρτου.



Εικόνα 1.4 *Πηγή εικόνας:* ***https://cylab.be/blog/150/http-benchmarking-with-wrk2***

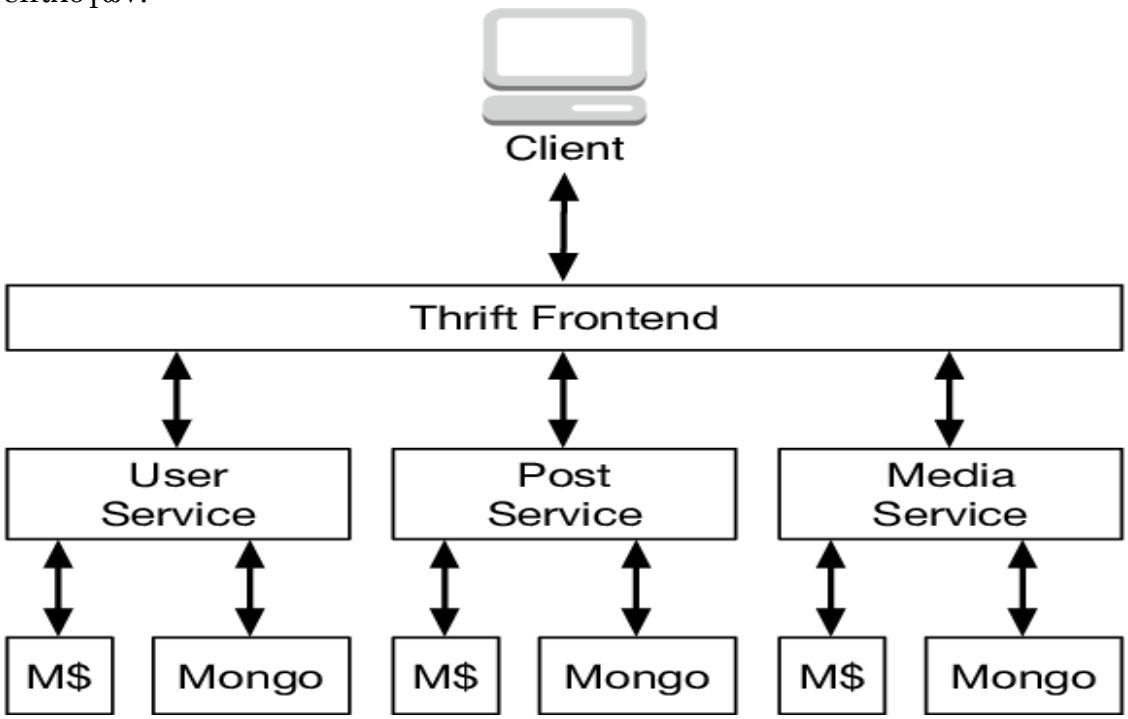
1.3.6 DeathStarBench

Το DeathStarBench είναι μια ανοιχτού κώδικα σουίτα αξιολόγησης (benchmarking suite), σχεδιασμένη για τη μέτρηση της απόδοσης και της επεκτασιμότητας εφαρμογών μικροϋπηρεσιών που βασίζονται σε υποδομές cloud. Πρόκειται για ένα από τα πιο ολοκληρωμένα εργαλεία στον τομέα της αξιολόγησης microservices, καθώς περιλαμβάνει διαφορετικά workloads τα οποία καλύπτουν

ποικίλα σενάρια χρήσης. Συγκεκριμένα, παρέχει έξι εφαρμογές μικροϋπηρεσιων, τρεις ολοκληρωμένες και ακόμα άλλες τρεις in progress.

Στην παρούσα εργασία, έμφαση δίνεται στην αρχιτεκτονική μικροϋπηρεσιών Social Network, η οποία προσομοιώνει μια σύγχρονη πλατφόρμα κοινωνικής δικτύωσης. Η εφαρμογή αυτή είναι σχεδιασμένη ως σύνολο ανεξάρτητων μικροϋπηρεσιών οι οποίες καλύπτουν διαφορετικές λειτουργίες της πλατφόρμας, όπως διαχείριση χρηστών, σχέσεις φίλων, αλληλεπιδράσεις (likes, comments), αποθήκευση και προβολή περιεχομένου. Αυτή η αρθρωτή σχεδίαση (modular design), σε αντίθεση με την παραδοσιακή μονολιθική προσέγγιση, διευκολύνει τόσο τη μελέτη της απόδοσης κάθε επιμέρους στοιχείου όσο και την εφαρμογή στοχευμένων μεθόδων κλιμάκωσης.

Χάρη στην αρχιτεκτονική αυτή, καθίσταται εφικτός ο εντοπισμός της μικροϋπηρεσίας η οποία αποτελεί το «στενό σημείο» (bottleneck) σε δεδομένο φόρτο εργασίας. Αυτό παρέχει πολύτιμη πληροφορία για την αξιολόγηση των στρατηγικών ελαστικής διαχείρισης πόρων, καθώς επιτρέπει την παρακολούθηση της αλληλεπίδρασης των διαφορετικών υπηρεσιών και την κατανόηση του τρόπου με τον οποίο η κατανομή πόρων επηρεάζει τη συνολική απόδοση του συστήματος. Έτσι, το DeathStarBench δεν αποτελεί απλώς μια σουίτα πειραμάτων, αλλά ένα ουσιαστικό πλαίσιο το οποίο υποστηρίζει την αναπαραγωγή ρεαλιστικών συνθηκών σε περιβάλλοντα. Τέλος, καθίσταται ιδιαίτερα χρήσιμο εργαλείο για την ερευνά στον χώρο του cloud computing, λόγω της μεγάλης ποικιλίας πειραματικών επίλογων.



Εικόνα 1.5 *Πηγή εικόνας:* ***https://www.researchgate.net/figure/Microservice-architecture-of-the-Social-Network-benchmark-Each-node-is-a-service-or_fig1_392716821***

1.3.7 Jaeger Tracing

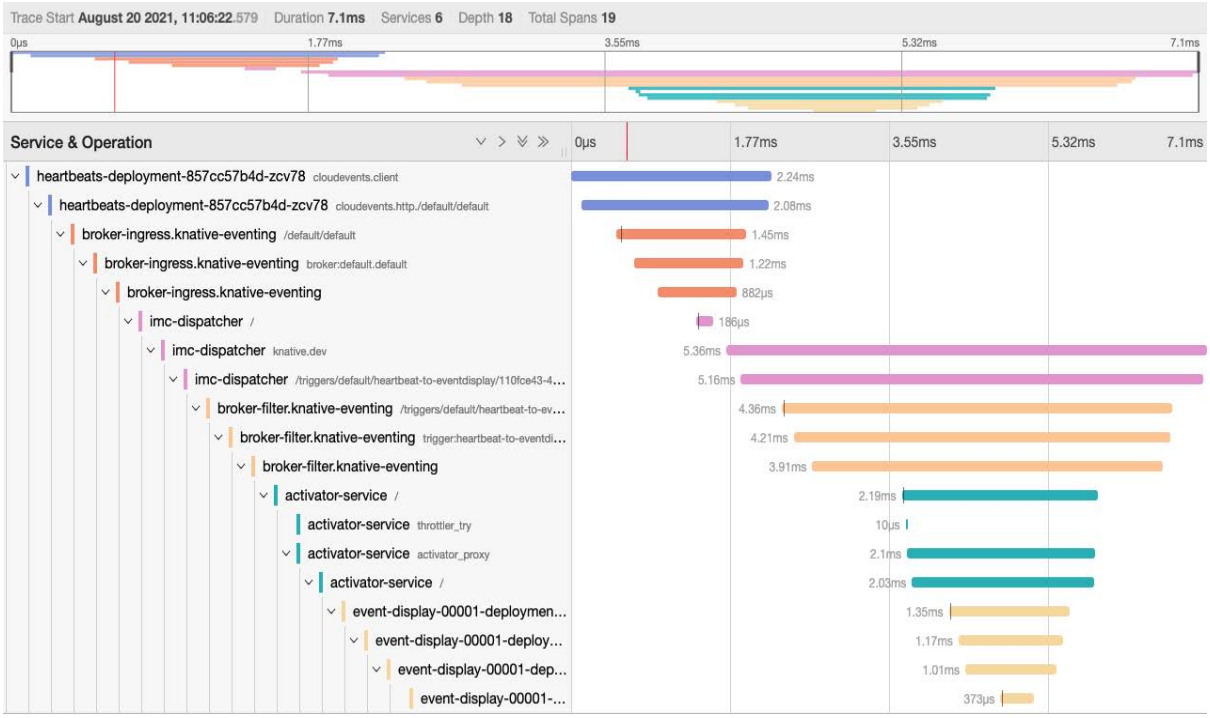
Το Jaeger είναι ένα εργαλείο ανοικτού κώδικα για την ανάλυση και παρακολούθηση κατανεμημένων συστημάτων. Αναπτύχθηκε αρχικά από την Uber και σήμερα υποστηρίζεται από το CNCF. Ο βασικός του σκοπός είναι να επιτρέπει την παρακολούθηση των αιτημάτων καθώς αυτά «ταξιδεύουν» μέσα από διαφορετικές μικροϋπηρεσίες σε περιβάλλοντα cloud.

Η χρήση των microservices έχει αυξήσει σημαντικά την πολυπλοκότητα των εφαρμογών. Ενώ σε μια μονολιθική αρχιτεκτονική είναι σχετικά εύκολο να καταλάβει κανείς πως εκτελείται ένα αίτημα, σε ένα σύστημα μικροϋπηρεσιων η ιδιά λειτουργία μπορεί να χρειαστεί συνεργασία πολλών διαφορετικών υπηρεσιών. Αυτό καθιστά δύσκολο τον εντοπισμό του σημείου που προκαλεί καθυστερήσεις ή σφάλματα. Το Jaeger προσφέρει end-to-end tracing, δηλαδή την πλήρη ιχνηλατηση της πορείας ενός αιτήματος. Με αυτόν τον τρόπο, μπορεί να καταγράφει κάθε στάδιο που περνά το αίτημα, ο χρόνος απόκρισης σε κάθε υπηρεσία και οι πιθανές καθυστερήσεις. Το εργαλείο αναθέτει σε κάθε αίτημα ένα μοναδικό αναγνωριστικό (trace ID), το οποίο το συνοδεύει σε όλη τη διαδρομή του.

Η χρησιμότητα του Jaeger είναι σημαντική, καθώς:

- βοηθά στον εντοπισμό της ρίζας ενός προβλήματος
- παρέχει εικόνα των εξαρτήσεων ανάμεσα στις υπηρεσίες, μέσω γραφικών αναπαραστάσεων,
- δίνει την δυνατότητα να παρακολουθεί κανείς την κατανομή των καθυστερήσεων, πράγμα που το κάνει επίσης πολύ σημαντικό για την αξιολόγηση της συνολικής απόδοσης ενός συστήματος.

Στην παρούσα εργασία, το Jaeger αξιοποιείται για την παρακολούθηση της αρχιτεκτονικής μικροϋπηρεσιων του DeathStarBench Social Network benchmark. Με αυτόν τον τρόπο, καθίσταται δυνατή η ανάλυση της συμπεριφοράς του συστήματος υπό φορτίο καθώς και η αξιολόγηση στρατηγικών αυτόματης κλιμάκωσης μέσα από πραγματικά δεδομένα παρακολούθησης.



Εικόνα 1.6 *Πηγή εικόνας:* ***https://knative.dev/blog/articles/distributed-tracing/***

1.3.8 Horizontal Pod Autoscaler (HPA)

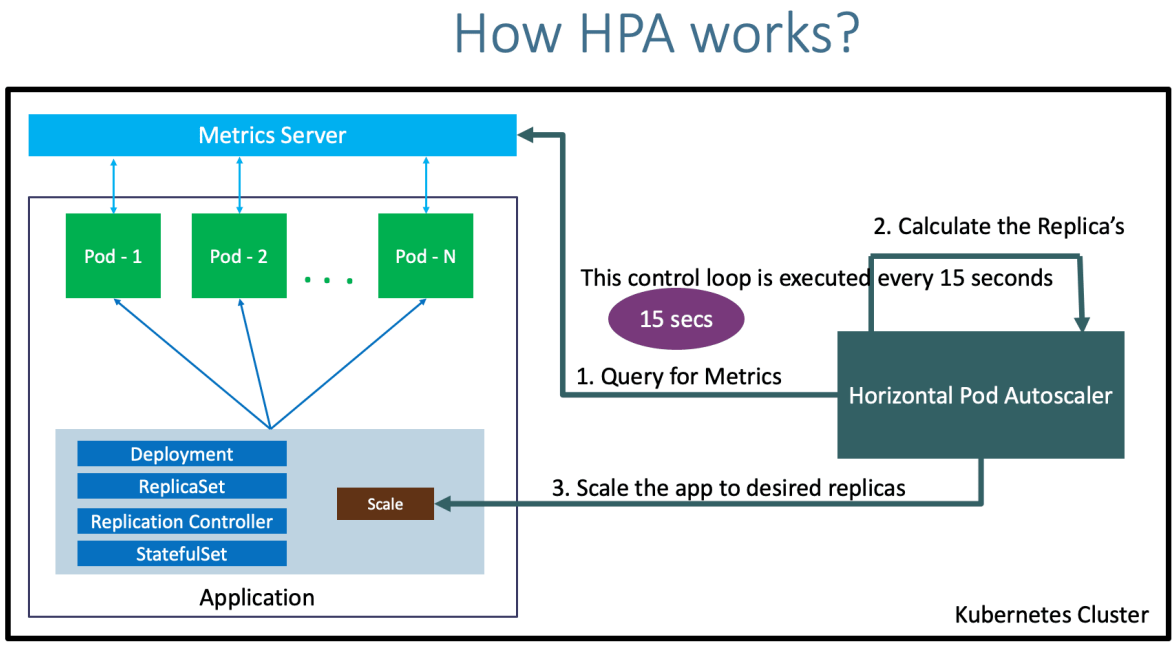
Το Horizontal Pod Autoscaler (HPA) αποτελεί έναν από τους βασικότερους μηχανισμούς του Kubernetes για την αυτόματη προσαρμογή της κλίμακας εφαρμογών. Η λειτουργία του έγκειται στην αυτόματη αύξηση ή μείωση του αριθμού των pods τα οποία τρέχουν μια εφαρμογή, βάσει προκαθορισμένων μετρικών απόδοσης. Με αυτόν τον τρόπο, το σύστημα μπορεί να ανταποκρίνεται δυναμικά σε όλες μεταβολές του φορτιού εργασίας, εξασφαλίζοντας αφενός υψηλή διαθεσιμότητα και αφετέρου αποδοτική χρήση των υπολογιστικών πόρων.

Η πιο συνηθισμένη μετρική που χρησιμοποιείται είναι η χρήση CPU ανά pod, ωστόσο το HPA υποστηρίζει και άλλες μετρικές, όπως η χρήση μνήμης, προσαρμοσμένα metrics μέσω του Metrics API ή ακόμα και application level μετρικές (π.χ. requests per second). Το HPA παρακολουθεί συνεχώς αυτές τις τιμές και, όταν ανιχνεύσει ότι το τρέχον φορτίο αποκλίνει από τις καθορισμένες πολίτικες, ενεργοποιεί διαδικασίες κλιμάκωσης.

Για παράδειγμα, ένας διαχειριστής μπορεί να ορίσει ότι μια εφαρμογή πρέπει να διατηρεί μέση χρήση CPU στο 50%. Αν η πραγματική χρήση υπερβεί αυτό το οροί, ο HPA θα αυξήσει τον αριθμό των pods ενώ αν μειωθεί, τότε θα μειώσει. Η διαδικασία αυτή εκτελείται περιοδικά (συνήθως κάθε 15 δευτερόλεπτα, ανάλογα με το sync period), επιτρέποντας γρήγορη ανταπόκριση στις αλλαγές. Αξίζει να σημειωθεί ότι η οριζόντια κλιμάκωση μέσω HPA διαφοροποιείται από την κάθετη κλιμάκωση (Vertical Pod Autoscaler, VPA), η οποία αυξάνει ή μειώνει

τους πόρους που κατανέμονται σε κάθε pod. Στην πράξη, το HPA προσφέρει καλύτερη ευελιξία σε εφαρμογές μικροϋπηρεσιών, καθώς επιτρέπει την ταυτόχρονη εκτέλεση πολλών instances, ενισχύοντας την ανθεκτικότητα και την κατανομή του φόρτου.

Στην συγκεκριμένη εργασία, ο HPA αξιοποιείται ως βασικό εργαλείο για την ελαστική διαχείριση πόρων, καθώς επιτρέπει την προσαρμογή του αριθμού των pods ανάλογα με τις συνθήκες φορτιού που δημιουργούνται από τα stress test. Με αυτόν τον τρόπο, αξιολογούνται οι πολίτικες αυτόματης κλιμάκωσης σε ρεαλιστικά σενάρια μικροϋπηρεσιών.



Εικόνα 1.7 *Πηγή εικόνας:* ***https://www.stacksimplify.com/aws-eks/aws-eks-kubernetes-autoscaling/learn-to-master-horizontal-pod-autoscaling-on-aws-eks/***

ΚΕΦΑΛΑΙΟ 2 Υλοποίηση

2.1 Δομή

Στο παρόν κεφάλαιο παρουσιάζεται η διαδικασία υλοποίησης της προτεινομένης λύσης για την ελαστική και έξυπνη διαχείριση πόρων σε περιβάλλοντα μικροϋπηρεσιών. Αρχικά, περιγράφεται η συνολική αρχιτεκτονική του συστήματος και τα βασικά δομικά του στοιχεία, έτσι ώστε να καταστεί σαφές πως συνδυάζονται τα επιμέρους υποσυστήματα. Στη συνέχεια, αναλύεται η μεθοδολογία συλλογής και επεξεργασίας δεδομένων σχετικά με την κατανάλωση πόρων και την παρακολούθηση των υπηρεσιών μέσω ιχνηλάτησης (tracing). Η

πληροφορία αυτή αξιοποιείται για τον εντοπισμό κρίσιμων μικροϋπηρεσιών, οι οποίες αποτελούν πιθανά σημεία συμφόρησης υπό υψηλό φόρτο εργασίας.

Ακολούθως, εξετάζεται η προσέγγιση που υιοθετήθηκε για την υλοποίηση του μηχανισμού αυτόματης κλιμάκωσης, με έμφαση στον ορισμό των μετρικών και των πολιτικών που καθοδηγούν τη λήψη αποφάσεων. Ιδιαίτερη προσοχή δίνεται στη διαμόρφωση του περιβάλλοντος ενισχυτικής μάθησης, όπου περιγράφονται οι χώροι καταστάσεων και ενεργειών, καθώς και η συνάρτηση ανταμοιβής που αξιοποιείται για τη βελτιστοποίηση της στρατηγικής κλιμάκωσης.

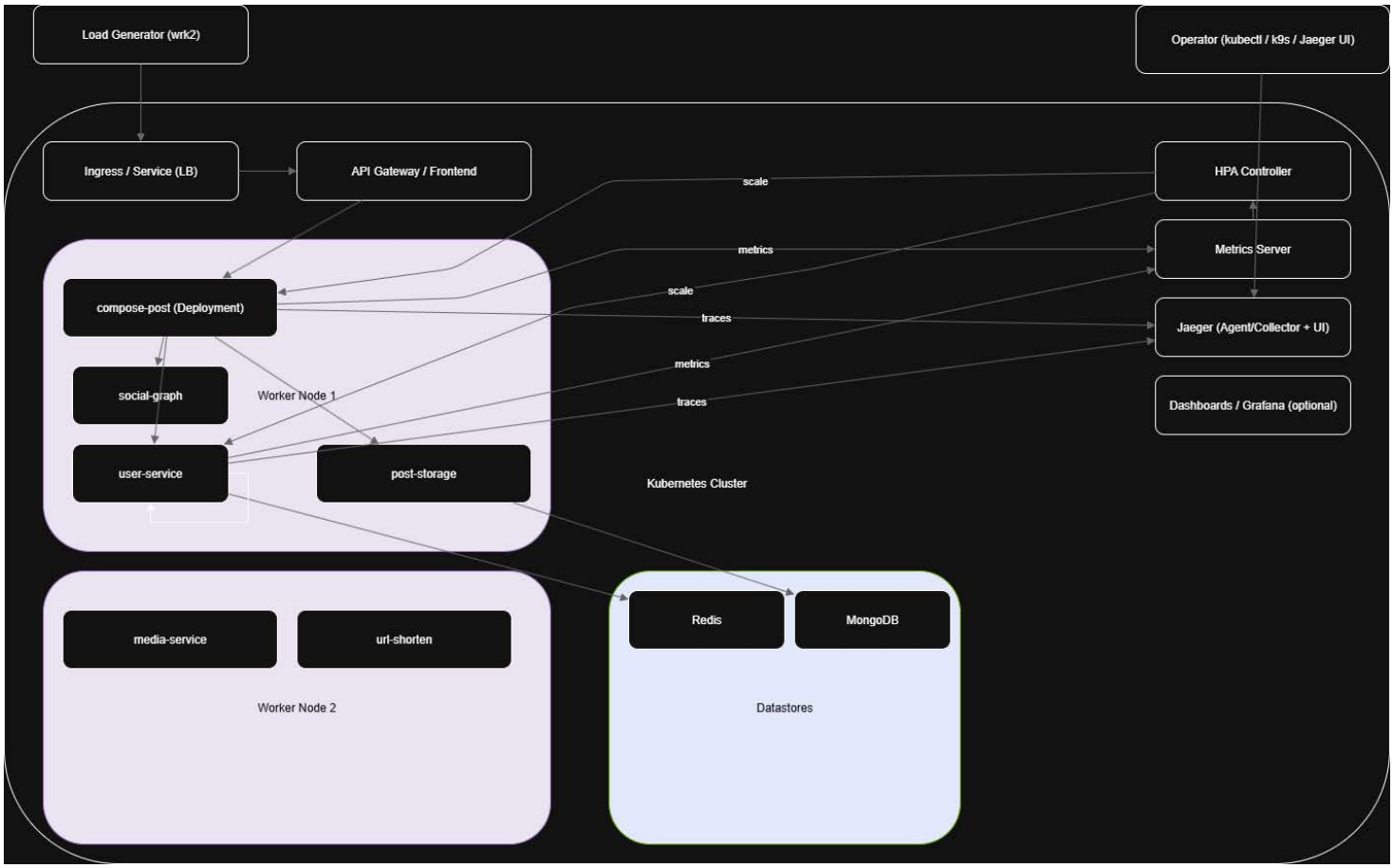
Επιπλέον, παρουσιάζεται η διαδικασία εκπαίδευσης του πράκτορα, η αρχιτεκτονική που υιοθετήθηκε και οι υπερπαράμετροι που χρησιμοποιήθηκαν για την προσαρμογή του αλγορίθμου. Παράλληλα, συζητείται ο τρόπος με τον οποίο αντιμετωπίζεται το πρόβλημα του μεγάλου χρόνου προσομοίωσης, μέσω τεχνικών παράλληλης εκπαίδευσης και κανονικοποίησης των δεδομένων. Τέλος, γίνεται αναφορά στα πρακτικά ζητήματα που προέκυψαν κατά τη διαδικασία υλοποίησης, καθώς και στις βελτιώσεις που εισήχθησαν ώστε το σύστημα να μπορεί να λειτουργήσει αποδοτικά σε πραγματικά περιβάλλοντα μικροϋπηρεσιών.

Στην εργασία θα υλοποιηθεί και θα αξιολογηθεί το workload Social Network (Death star Bench) πάνω σε Kubernetes. Η εφαρμογή εκτίθεται μέσω Ingress/Service, δέχεται συνθετικό φορτίο από wrk2 (open-loop) και αλληλοεπιδρά με MongoDB/Redis για την αποθήκευση των δεδομένων. Για παρατηρητικότητα ενσωματώνεται Jaeger (distributed tracing), ενώ οι μετρικές πόρων συλλέγονται από τον Metrics Server ώστε ο Horizontal Pod Autoscaler (HPA) να κλιμακώνει δυναμικά τα Deployments.

Οι βασικές ροές είναι:

- **Traffic:** wrk2 → Ingress → API/Gateway → Μικροϋπηρεσίες → Βάσεις.
- **Tracing:** spans από κάθε υπηρεσία → Jaeger (Agent/Collector + UI).
- **Metrics → Autoscaling:** Metrics Server → HPA → Scale Up/Down σε compose-post, user-service και αρκετά άλλα.
- **Έλεγχος:** kubectl/k9s/Jaeger UI για επιβεβαίωση κατάστασης και εντοπισμό bottlenecks.

Παρατίθεται το επόμενο διάγραμμα το οποίο εξηγεί τι συνδέεται με ποια υπηρεσία.



Εικόνα 2.0 *Αρχιτεκτονική υλοποίησης και ροές*

2.1.1 Περιβάλλον και προ απαιτούμενα

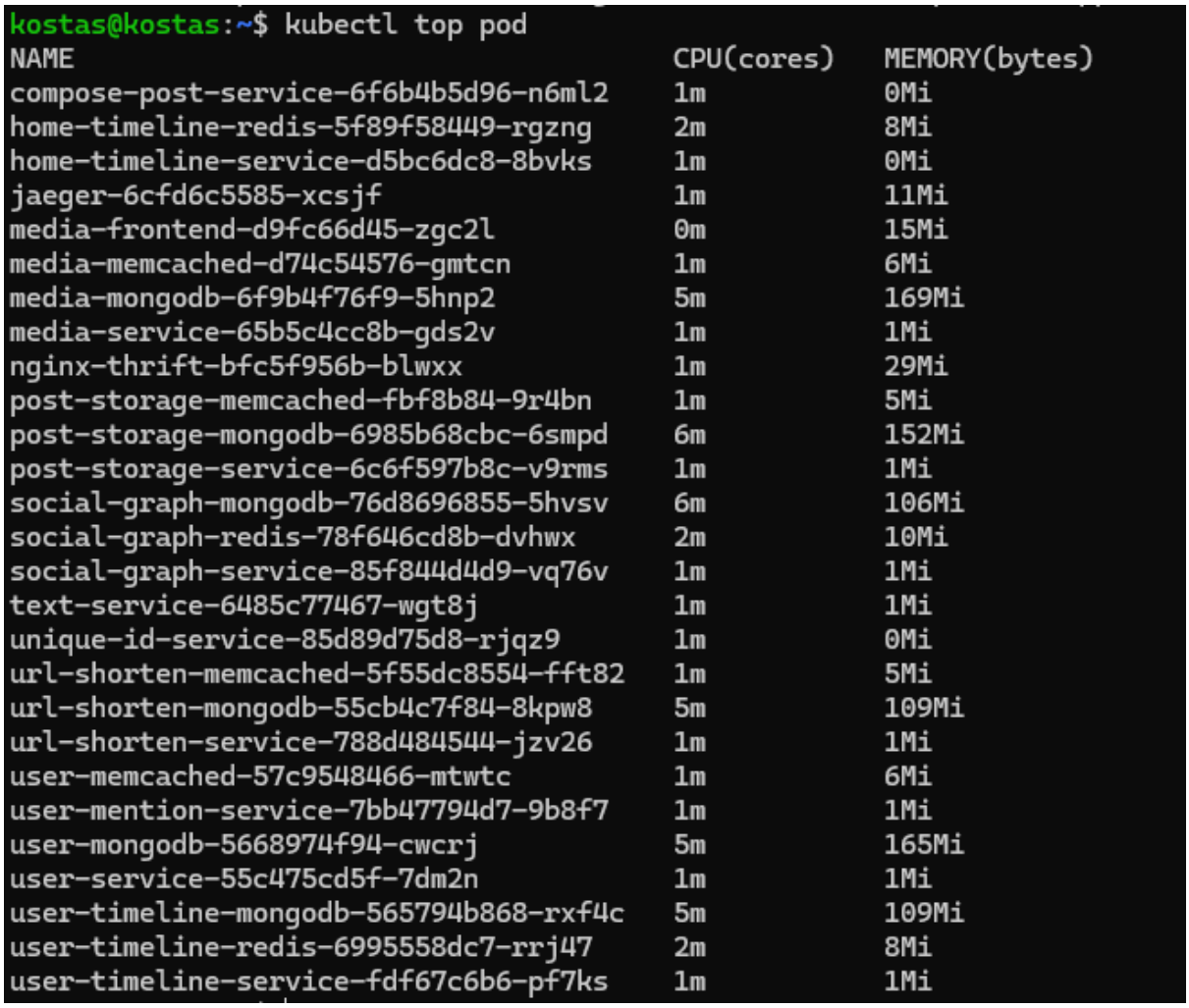
Αρχικά, η υλοποίηση πραγματοποιήθηκε σε τοπικό περιβάλλον, όπου εγκαταστάθηκε και ρυθμίστηκε ένα Kubernetes cluster μέσω Minikube στον προσωπικό υπολογιστή. Το στάδιο αυτό επέτρεψε την εξοικείωση με τα βασικά εργαλεία διαχείρισης του Kubernetes (kubectl, helm, k9s) και την κατανόηση της διαδικασίας ανάπτυξης και παρακολούθησης μικροϋπηρεσιών σε περιορισμένο, αλλά απολύτως ελεγχόμενο περιβάλλον.

Έλεγχος έκδοσης του Kubernetes client:

```
kostas@kostas:~$ kubectl version --client
Client Version: v1.32.4
Kustomize Version: v5.5.0
```

Εικόνα 2.1 *Έξοδος της εντολής* ***kubectl version –client***

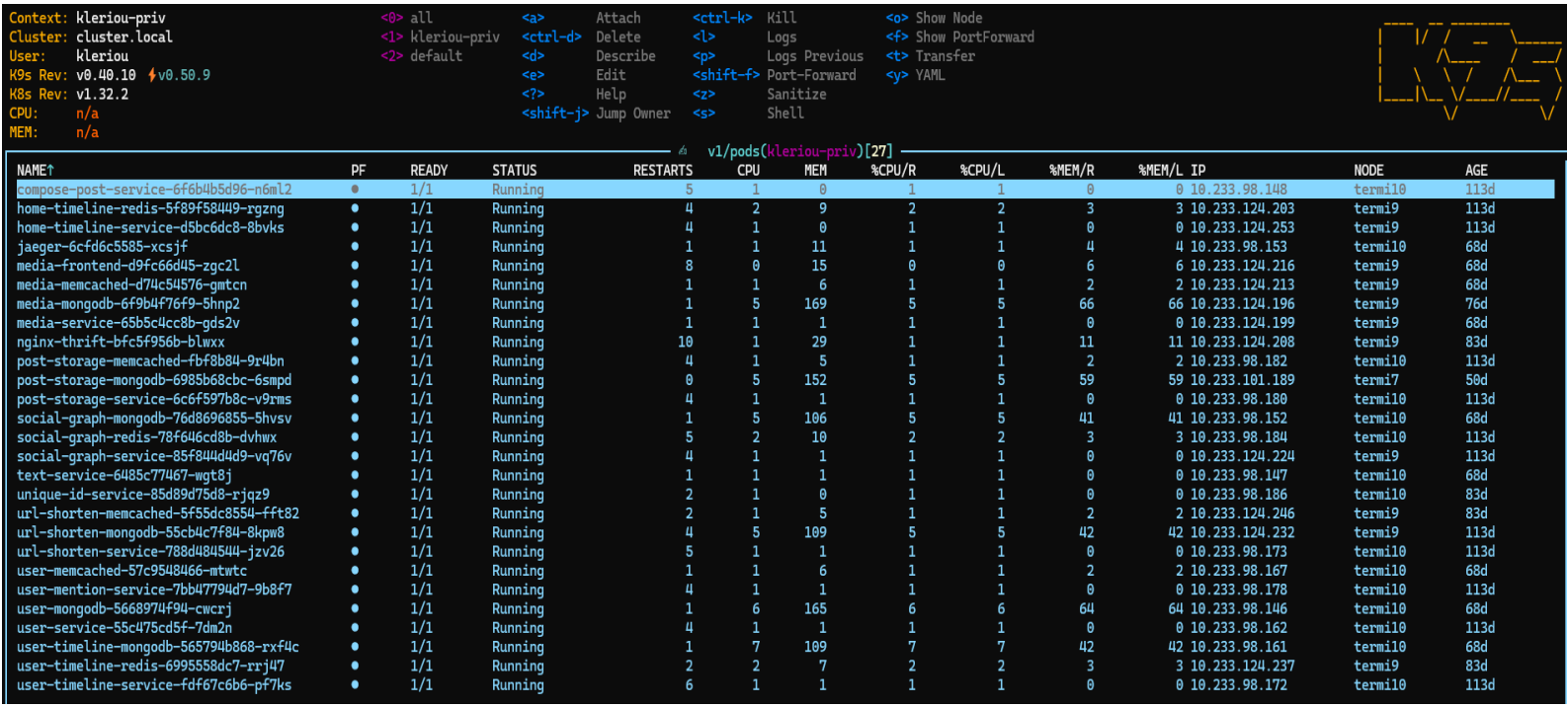
Παρακάτω βλέπουμε την κατανάλωση CPU και μνήμης στο cluster



```
kostas@kostas:~$ kubectl top pod
NAME                                      CPU(cores)   MEMORY(bytes)
compose-post-service-6f6b4b5d96-n6ml2     1m           0Mi
home-timeline-redis-5f89f58449-rgzng      2m           8Mi
home-timeline-service-d5bc6dc8-8bvks      1m           0Mi
jaeger-6cfd6c5585-xcsjf                   1m           11Mi
media-frontend-d9fc66d45-zgc2l            0m           15Mi
media-memcached-d74c54576-gmtcn           1m           6Mi
media-mongodb-6f9b4f76f9-5hnp2            5m           169Mi
media-service-65b5c4cc8b-gds2v            1m           1Mi
nginx-thrift-bfc5f956b-blwxx              1m           29Mi
post-storage-memcached-fbf8b84-9r4bn      1m           5Mi
post-storage-mongodb-6985b68cbc-6smpd     6m           152Mi
post-storage-service-6c6f597b8c-v9rms     1m           1Mi
social-graph-mongodb-76d8696855-5hvsv     6m           106Mi
social-graph-redis-78f646cd8b-dvhwx       2m           10Mi
social-graph-service-85f844d4d9-vq76v     1m           1Mi
text-service-6485c77467-wgt8j             1m           1Mi
unique-id-service-85d89d75d8-rjqz9        1m           0Mi
url-shorten-memcached-5f55dc8554-fft82    1m           5Mi
url-shorten-mongodb-55cb4c7f84-8kpw8      5m           109Mi
url-shorten-service-788d484544-jzv26      1m           1Mi
user-memcached-57c9548466-mtwtc           1m           6Mi
user-mention-service-7bb47794d7-9b8f7     1m           1Mi
user-mongodb-5668974f94-cwcrj             5m           165Mi
user-service-55c475cd5f-7dm2n             1m           1Mi
user-timeline-mongodb-565794b868-rxf4c    5m           109Mi
user-timeline-redis-6995558dc7-rrj47      2m           8Mi
user-timeline-service-fdf67c6b6-pf7ks     1m           1Mi
```

Εικόνα 2.2 *χρήση* ***kubectl top pod***

Με χρήση του εργαλείου k9s παρακολουθείται σε πραγματικό χρόνο η κατάσταση των pods στο απομακρυσμένο Kubernetes cluster του εργαστηρίου, όπου φαίνονται οι υπηρεσίες του Social Network Benchmark να εκτελούνται κανονικά σε πολλούς κόμβους.



Εικόνα 2.3 *χρήση* ***k9s***

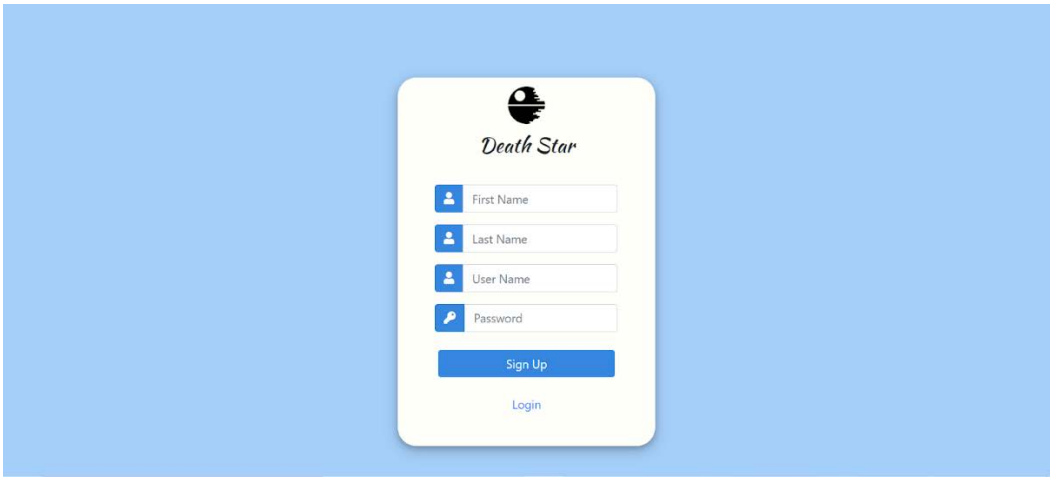
Παρακάτω βλέπουμε πως υλοποιείται το Deployment YAML, στο οποίο από την 3η σειρά ενεργοποιείται ή απενεργοποιείται το Horizontal Pod Autoscale, και επίσης έχει οριστεί ο Ελάχιστος αριθμός των replicas, ο Μέγιστος αριθμός καθώς και τα percentages από τα οποία, αν για αρκετή ώρα το συνολικό utilization βρίσκεται κάτω από τα όρια, η αντίστοιχα πάνω μειώνει τα pods ή αντίθετα τα αυξάνει.

```
kostas@kostas:~/DeatstarNew/DeathStarBench/socialNetwork/helm-chart/socialnetwork$ cat values.yaml
global:
  hpa:
    enabled: true
    minReplicas: 1
    maxReplicas: 10
    targetMemoryUtilizationPercentage: '85'
    targetCPUUtilizationPercentage: '60'
  resources:
    limits:
      cpu: 100m
      memory: 256Mi
    requests:
      cpu: 100m
      memory: 256Mi
  replicas: 1
  imagePullPolicy: "IfNotPresent"
  restartPolicy: Always
  serviceType: ClusterIP
  dockerRegistry: docker.io
```

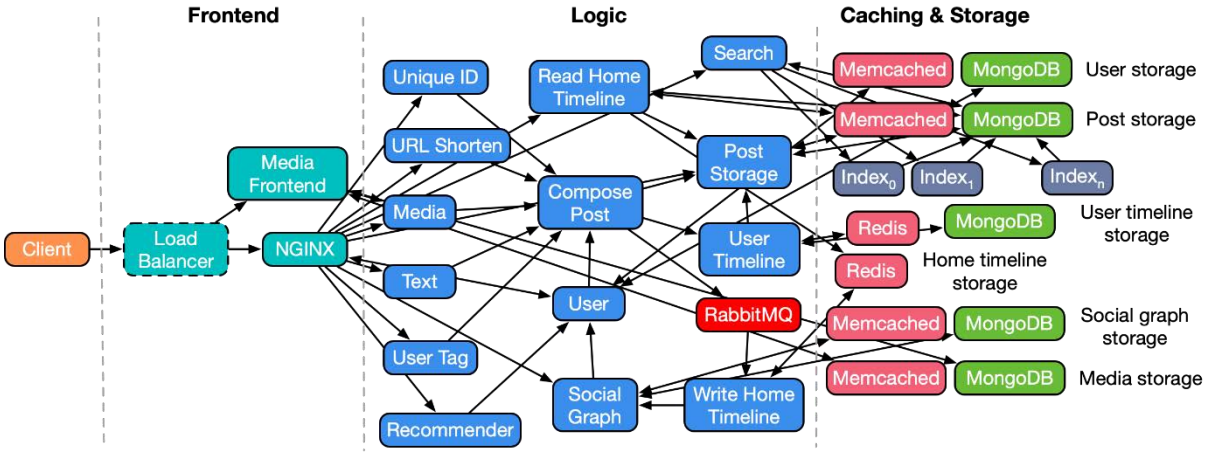
Εικόνα 2.4 *Ένα μικρό YAML snippet*

2.1.2 Ανάπτυξη DeathStarBench SocialNetwork στο Kubernetes

Για την προσομοίωση ρεαλιστικών συνθηκών μικροϋπηρεσιών χρησιμοποιήθηκε η σουίτα DeathStarBench, συγκεκριμένα το Social Network workload. Η εφαρμογή αναπτύχθηκε στο Kubernetes μέσω των παρεχόμενων manifests, δημιουργώντας ένα σύνολο ανεξάρτητων υπηρεσιών με διακριτές διεπαφές. Κάθε μικροϋπηρεσία αναλαμβάνει συγκεκριμένη λειτουργικότητα της πλατφόρμας κοινωνικής δικτύωσης, επιτρέποντας την παρακολούθηση και αξιολόγηση της απόδοσης σε επίπεδο υπηρεσίας. Η ανάπτυξη σε Kubernetes επιτρέπει την απομόνωση της κάθε υπηρεσίας, τη δυναμική κλιμάκωση και την εύκολη παραμετροποίηση των resource requests και limits.



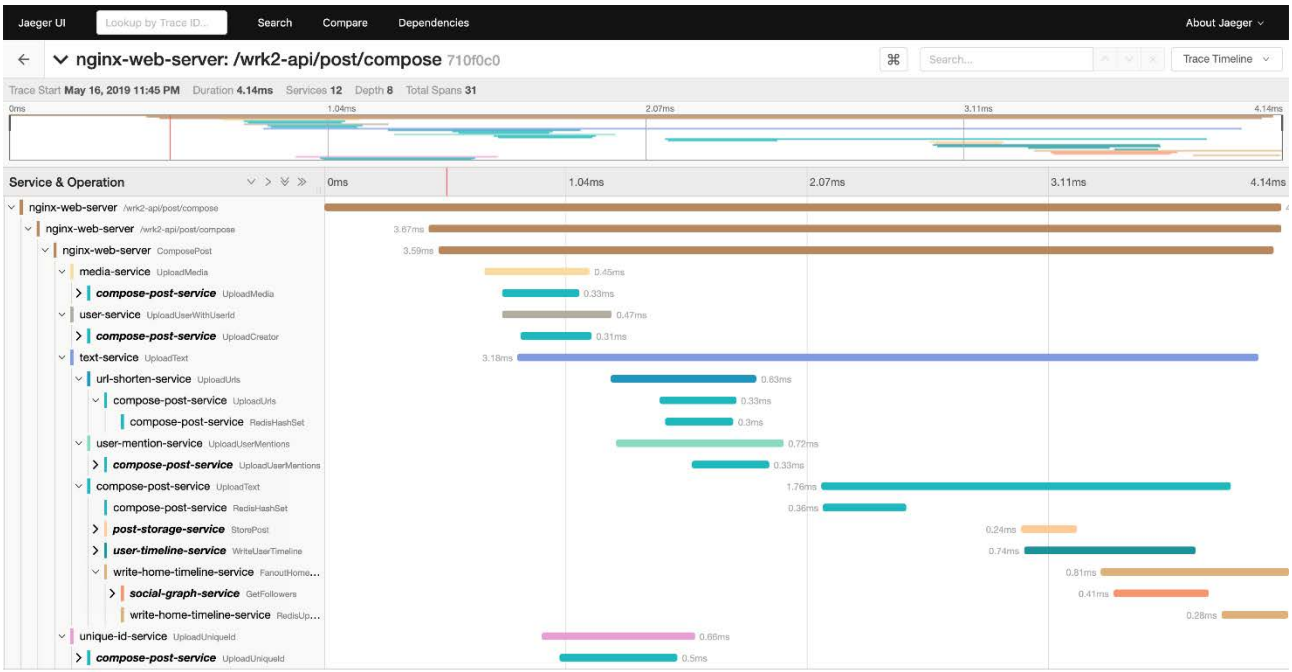
Εικόνα 2.5 *Πηγή εικόνας:*
https://github.com/delimitrou/DeathStarBench/blob/master/socialNetwork/figures/signup.png



Εικόνα 2.6 *Πηγή εικόνας:*
https://github.com/delimitrou/DeathStarBench/blob/master/socialNetwork/figures/socialNet_arch.png

2.1.3 Παρατηρησιμότητα (Tracing & Metrics)

Για την παρατήρηση της κατάστασης της εφαρμογής την ώρα που δέχεται αιτήματα, εγκαταστάθηκε το Jaeger. Με τον τρόπο αυτό, κάθε αίτημα αποκτά ένα trace_id και η διαδρομή του μέσα στις μικρουπηρεσίες είναι ορατή. Έτσι εντοπίζεται ποιο κομμάτι καθυστερεί (π.χ. μια κλήση σε βάση ή ένα service που αργεί. Παράλληλα παίρνονται βασικές μετρήσεις (CPU, μνήμη, ρυθμός αιτημάτων, ποσοστό σφαλμάτων) ανά pod. Το σημαντικό σε όλο αυτό είναι ότι καθώς βρίσκονται όλα στο ίδιο χρονικό πλαίσιο, όταν ανεβαίνουν τα replicas ή αλλάζει η χρήση της CPU, ταυτόχρονα επηρεάζεται το latency.



Εικόνα 2.7 *Πηγή εικόνας:* ***https://github.com/delimitrou/DeathStarBench/blob/master/socialNetwork/figures/socialNet_jaeger.png***

2.1.4 Παραγωγή Φορτίου (wrk2)

Στόχος εδώ είναι να παραχθεί μετρήσιμο και επαναλήψιμο φορτίο, ώστε να συγκριθούν ίδιες καταστάσεις πριν και μετά τις ρυθμίσεις κλιμάκωσης. Χρησιμοποιείται wrk2 σε open-loop λειτουργία. Δίνεται ένας στοχευμένος ρυθμός αιτημάτων (RPS) και ο πελάτης συνεχίζει να στέλνει αιτήματα με αυτόν τον ρυθμό, ανεξάρτητα από το αν η εφαρμογή καθυστερεί. Έτσι αποφεύγεται το feedback που έχουν τα κλασικά κλειστά loops και φαίνεται καθαρά πότε στο σύστημα αρχίζει ο κορεσμός (σφάλματα και μεγάλα latency).

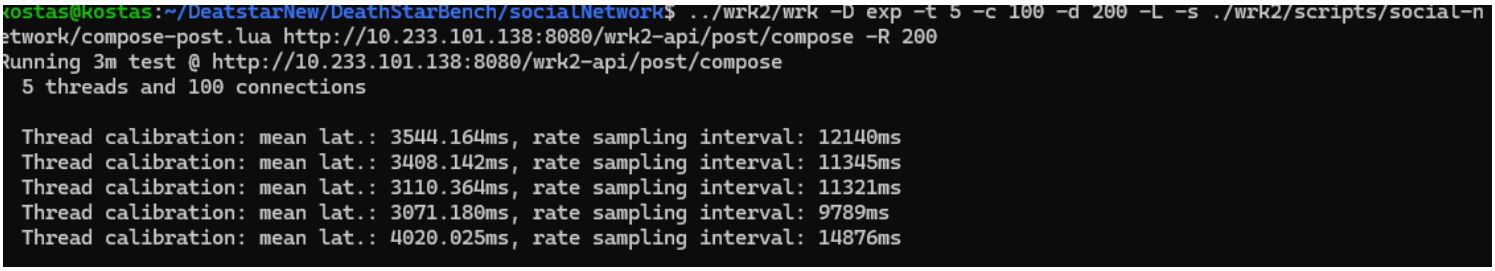
Ρύθμιση περιβάλλοντος

- **Απομόνωση πελάτη:** τρέχω το wrk2 σε ξεχωριστό μηχάνημα από το cluster, στο ίδιο δίκτυο, για να μην καταναλώνει πόρους από τα pods.
- **Χρονικός συγχρονισμός:** τα timestamps του wrk2 να ταιριάζουν χρονικά με τα traces-metrics

Παράμετροι εκτέλεσης

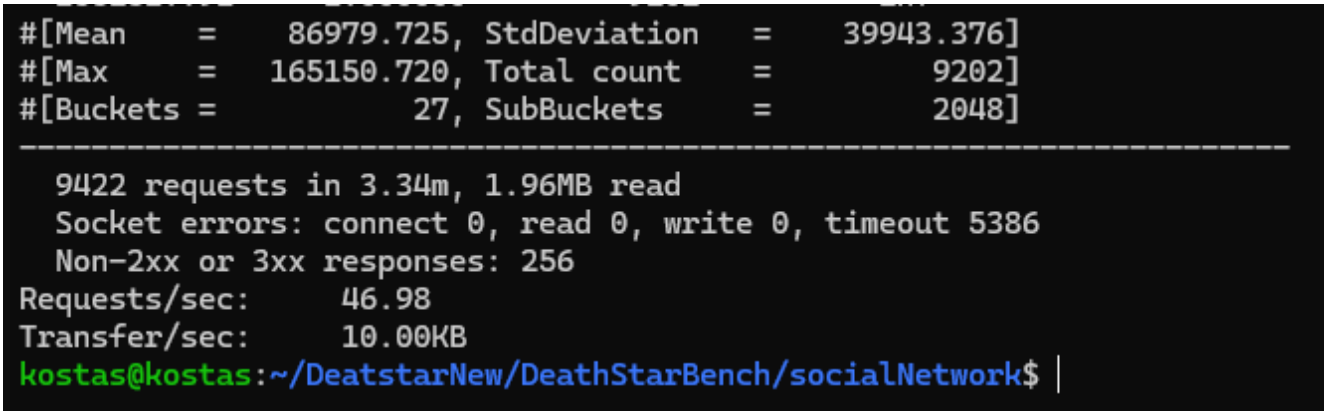
- **RPS (-R):** Η αλλιώς Requests Per Second, ουσιαστικά είναι το πιο σημαντικό κομμάτι στις δοκιμές καθώς ανεβαίνουν ανά 100 τα requests, μέχρι να εμφανιστεί μια σταθερή απόκλιση από τον στόχο ή να παρατηρηθεί αύξηση στο latency
- **Threads (-t) / Connections (-c):** Επιλέγονται threads (λογικούς πυρήνες) κοντά στο σύστημα του πελάτη, καθώς μένουν και αρκετές συνδέσεις γύρω στις 120 για να κρατιέται γεμάτος ο αγωγός.
- **Διάρκεια (-d):** κάθε φορά που δοκιμάζουμε, ορίζουμε ένα χρονικό όριο έως ότου θα σταματήσει το τσεκάρισμα. Συνήθως ορίζεται περίπου στα 2 λεπτά.
- **Επαναλήψεις :** Προκειμένου να μην ληφθούν πιθανόν κάποια τυχαία αποτελέσματα, οι δοκιμές επαναλαμβάνονται για να επιβεβαιωθεί η ομοιότητα των αποτελεσμάτων.

Μια ενδεικτική κλήση είναι η:
../wrk2/wrk -D exp -t 12 -c 400 -d 300 -L -s ./wrk2/scripts/social-network/compose-post.lua http://10.109.126.103:8080/wrk2-api/post/compose -R 10



Εικόνα 2.8 *εκτέλεση εντολής wrk*

Από την παραπάνω εικόνα παρατηρούμε πως εκτελέστηκε η εντολή wrk θέτοντας 5 λογικούς πυρήνες (threads), καθώς 100 συνεχόμενων συνδέσεων, διάρκειας 200 δευτερολέπτων και με Requests per seconds 200. Στην αρχή εκτελεί κάποια calibration στους πυρήνες και έπειτα ξεκινάει το stress test.



Εικόνα 2.9 *αποτελέσματα χρήσης εντολής wrk*

Επίσης από την εικόνα **1.15** βλέπουμε ότι η εκτέλεση του stress test ολοκληρώθηκε με επιτυχία, στέλνοντας 9422 requests σε μόλις 3 λεπτά και 34 δευτερόλεπτα. Παράλληλα παρατηρούμε κάποια timeouts καθώς και άλλα non 2xx – 3xx responses.

2.1.5 Μηχανισμός Αυτόματης Κλιμάκωσης

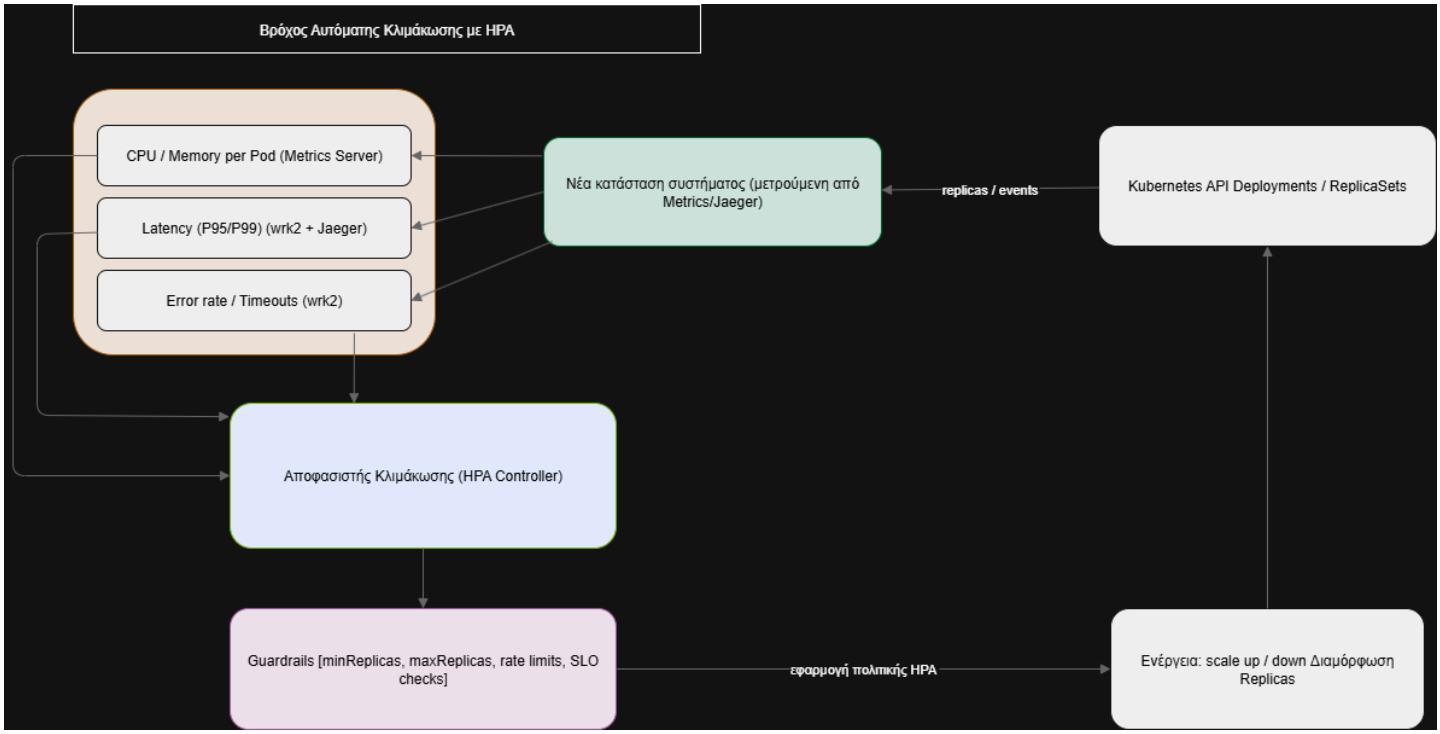
Σε αυτή την ενότητα περιγράφεται ο μηχανισμός αυτόματης κλιμάκωσης που εφαρμόζεται αποκλειστικά με Horizontal Pod Autoscaler (HPA). Ο HPA παρακολουθεί μετρικές πόρων ανά pod από τον Metrics Server (κυρίως CPU και, συμπληρωματικά, Memory) και προσαρμόζει δυναμικά τον αριθμό των replicas των στοχευμένων Deployments, με στόχο να κρατά τη χρήση κοντά σε μια επιθυμητή τιμή (π.χ. CPU 60%). Η κλιμάκωση εφαρμόζεται κυρίως στις υπηρεσίες που ανήκουν στο κρίσιμο μονοπάτι αιτήματος, όπως compose-post και user-service.

Παραμετροποίηση HPA.

- **Στόχος (Target):** CPU Utilization = 60% (ενδεικτικά).
- **Όρια:** minReplicas = 2, maxReplicas = 16, ώστε να αποφεύγονται ακραίες καταστάσεις.
- **Ρυθμός αλλαγών:** Χρησιμοποιούνται οι default σταθεροποιήσεις του Kubernetes (stabilization window / cooldown) ώστε να μη μεταβάλλονται υπερβολικά συχνά τα replicas.
- **Έκθεση μετρικών:** Ο Metrics Server εκθέτει cpu.utilization ανά pod · τα αποτελέσματα του HPA συσχετίζονται χρονικά με τα Jaeger traces και τα αποτελέσματα του wrk2 (P95/P99 latency, error rate) για να τεκμηριώνεται ο αντίκτυπος της κλιμάκωσης.

Guardrails.
Εφαρμόζονται πρακτικοί περιορισμοί ώστε η κλιμάκωση να είναι ασφαλής και σταθερή:

- **replicas:** [minReplicas, maxReplicas] ανά υπηρεσία.
- **Rate-limits αλλαγών**: αποφυγή πολλαπλών αυξομειώσεων σε σύντομο χρονικό διάστημα.



Εικόνα 2.10 *Βρόχος Αυτόματης Κλιμάκωσης με HPA*

ΚΕΦΑΛΑΙΟ 3 Συναφή Έργα

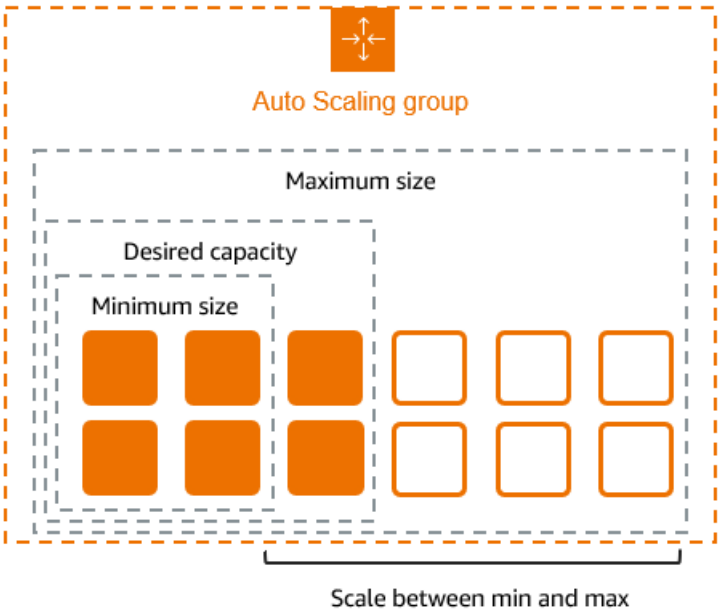
3.1 Cloud Autoscalers

Η αυτόματη κλιμάκωση (autoscaling) αποτελεί βασική υπηρεσία των μεγάλων παροχών cloud και επιτρέπει τη δυναμική προσαρμογή των πόρων που χρησιμοποιεί μια εφαρμογή ανάλογα με το εκάστοτε φορτίο. Οι πιο διαδεδομένες υλοποιήσεις προσφέρονται από την Amazon Web Services (AWS) και τη Microsoft Azure, οι οποίες διαθέτουν ώριμα και ευρέως χρησιμοποιούμενα εργαλεία.

AWS Auto Scaling[10]: Η Amazon παρέχει ένα ολοκληρωμένο πλαίσιο για αυτόματη κλιμάκωση. Το εργαλείο αυτό μπορεί να εφαρμοστεί τόσο σε EC2 Auto Scaling Groups (ομάδες εικονικών μηχανών) όσο και σε υπηρεσίες υψηλοτέρου επιπέδου όπως DynamoDB και ECS.
Η κλιμάκωση μπορεί να οριστεί με διάφορες στρατηγικές:

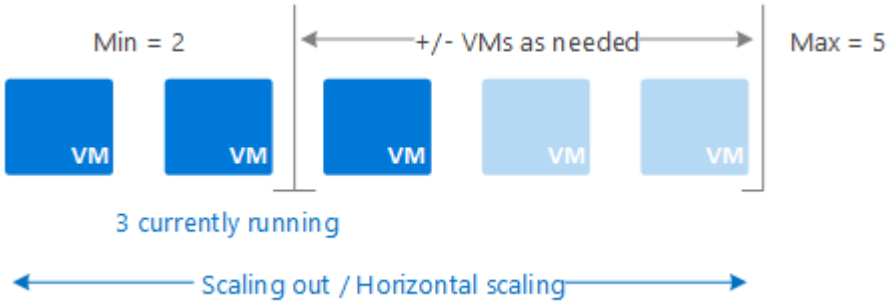
- Target Tracking Policies, οπού ο χρήστης θέτει έναν στόχο (π.χ. διατήρηση CPU στο 50%), το σύστημα αυξομειώνει αυτόματα τον αριθμό των instances για να τον πέτυχει.
- Step Scaling Policies, που βασίζονται σε κανόνες ανίχνευσης thresholds.
- Predictive Scaling, το οποίο χρησιμοποιεί machine learning για να προβλέπει μελλοντικές ανάγκες βάσει ιστορικών δεδομένων.

Η λύση της AWS προσφέρει υψηλή ωριμότητα, μεγάλη παραμετροποίηση και βαθιά ενσωμάτωση με τις υπόλοιπες υπηρεσίες του οικοσυστήματος AWS.



Εικόνα 3.0 *Πηγή εικόνας:* ***https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html***

Azure Autoscale[11]: Η Microsoft, προσφέρει αντίστοιχες δυνατότητες. Η υπηρεσία επιτρέπει την αυτόματη κλιμάκωση σε Virtual Machine Scale Sets, App services και Azure Kubernetes Service (AKS). Η κλιμάκωση καθορίζεται με βάση κανόνες, οι οποίοι μπορούν να οριστούν σε μετρικές όπως CPU, μνήμη, χρήση δίσκου η ακόμα και custom application metrics που δημοσιεύονται στο Azure Monitor. Ένα βασικό πλεονέκτημα του Azure είναι η δυνατότητα δημιουργίας προσαρμοσμένων χρονοδιαγραμμάτων κλιμάκωσης (scheduled autoscaling), ώστε οι εφαρμογές να μπορούν να κλιμακώνονται ανάλογα με προβλεπόμενα patterns ζήτησης (π.χ. ώρες αιχμής).



Εικόνα 3.1 *Πηγή εικόνας:* ***https://learn.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-overview***

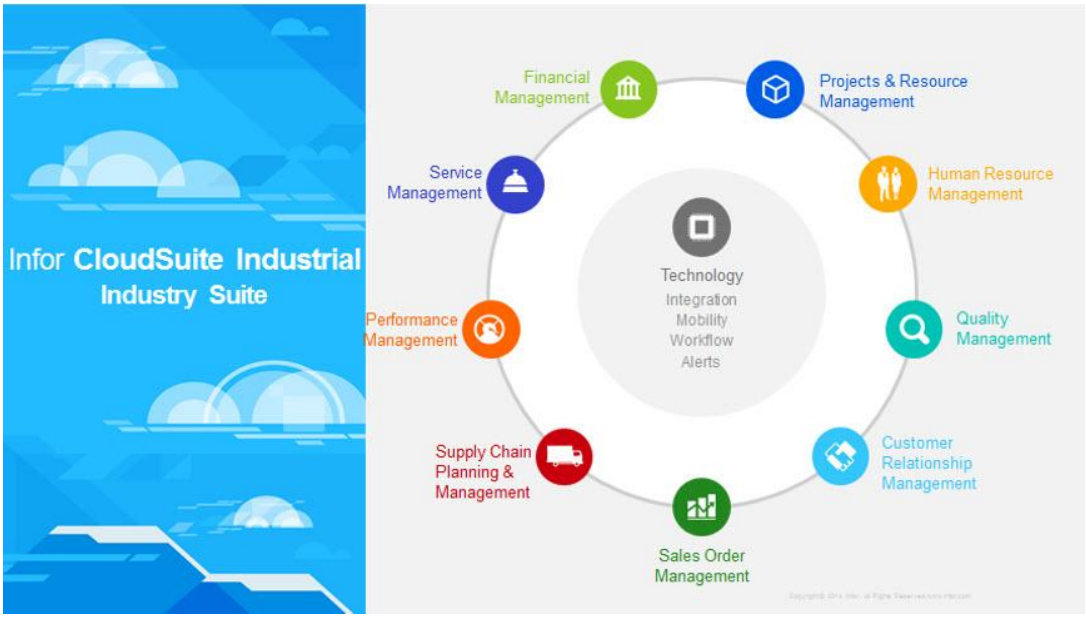
Συγκριτικά και οι δυο λύσεις προσφέρουν ώριμους μηχανισμούς για αυτόματη κλιμάκωση, με την AWS να δίνει έμφαση σε πιο προηγμένες προβλεπτικές τεχνικές και την Azure να διευκολύνει την ενοποίηση με monitoring εργαλεία και προσαρμοσμένα σενάρια.

3.2 Benchmarks

Πέρα από το DeathStarBench, το οποίο επικεντρώνεται κυρίως στην αξιολόγηση εφαρμογών μικροϋπηρεσιών, ένα ακόμη σημαντικό benchmark που χρησιμοποιείται ευρέως στην ερευνητική κοινότητα είναι το CloudSuite.

Το CloudSuite αναπτύχθηκε από το EPFL(École Polytechnique Fédérale de Lausanne) και αποτελεί μια συλλογή από workloads που αντικατοπτρίζουν ρεαλιστικά σενάρια χρήσης σε περιβάλλοντα cloud. Περιλαμβάνει εφαρμογές που σχετίζονται με web serving, data analytics, graph processing, media streaming και machine learning, προσφέροντας έτσι μια ποικιλία σεναρίων τα οποία καλύπτουν διαφορετικές κατηγορίες υπηρεσιών. Ένα βασικό πλεονέκτημα του CloudSuite είναι ότι τα workloads προέρχονται από πραγματικές cloud εφαρμογές και datasets, κάτι που το καθιστά ιδιαίτερα αντιπροσωπευτικό για την αξιολόγηση της απόδοσης. Ωστόσο, σε αντίθεση με το DeathStarBench, το οποίο δίνει έμφαση στην αρχιτεκτονική μικροϋπηρεσιών και τις εξαρτήσεις μεταξύ τους, το CloudSuite εστιάζει περισσότερο στην καθαρή απόδοση υποδομών για συγκεκριμένους τύπους εφαρμογών.

Συνεπώς, το CloudSuite μπορεί να θεωρηθεί συμπληρωματικό προς το DeathStarBench: ενώ το δεύτερο παρέχει βαθιά μελέτη στη συμπεριφορά των microservices, το πρώτο δίνει μια πιο γενική αποτίμηση για workloads που συναντώνται σε σύγχρονα cloud περιβάλλοντα.



Εικόνα 3.2 *Πηγή εικόνας:* ***https://www.izertis.com/en/-/blog/infor-cloudsuite-industrial-advantages***

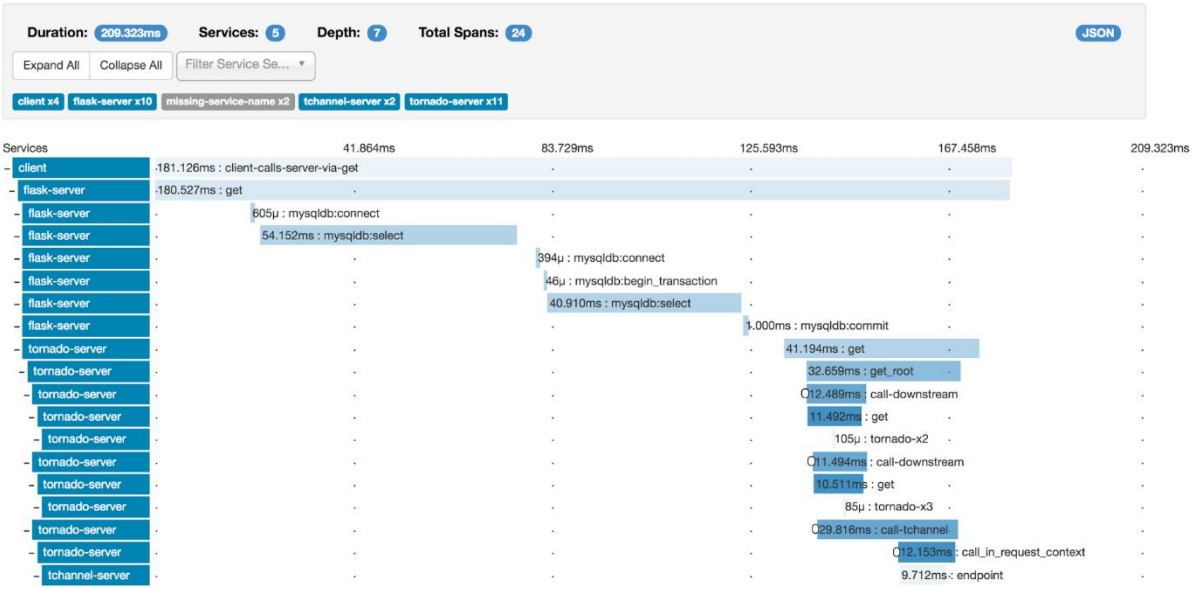
3.3 Distributed Tracing Tools

Η πολυπλοκότητα που χαρακτηρίζει τις εφαρμογές μικροϋπηρεσιών καθιστά αναγκαία την ύπαρξη μηχανισμών παρακολούθησης της ροής των αιτημάτων, ώστε να εντοπίζονται σημεία συμφόρησης και να επιτυγχάνεται αποδοτική διάγνωση προβλημάτων. Στο πλαίσιο αυτό, τα εργαλεία distributed tracing αποτελούν κρίσιμο στοιχείο της σύγχρονης παρατηρητικότητας (observability), επιτρέποντας την αναλυτική καταγραφή και οπτικοποίηση των αλληλεπιδράσεων μεταξύ υπηρεσιών.

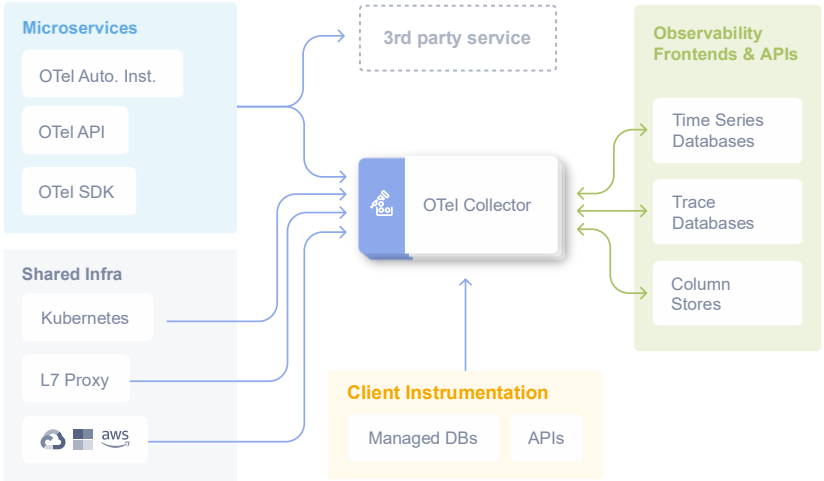
Ανάμεσα στα πρώτα συστήματα tracing που εισήχθησαν σε μεγάλη κλίμακα συγκαταλέγεται το Dapper της Google, το οποίο αποτέλεσε ορόσημο για τη μελέτη και την υλοποίηση μεταγενέστερων εργαλείων. Στην ίδια λογική κινείται και το Zipkin, το οποίο αναπτύχθηκε από το Twitter και παραμένει μια από τις πιο διαδεδομένες ανοιχτού κώδικα υλοποιήσεις, παρέχοντας δυνατότητες συλλογής, αποθήκευσης και απεικόνισης traces με σχετικά απλή και ελαφριά αρχιτεκτονική. Πιο πρόσφατα, το OpenTelemetry, έργο του Cloud Native Computing Foundation (CNCF), έχει καθιερωθεί ως πρότυπο για την ενοποιημένη συλλογή traces, metrics και logs. Σε αντίθεση με τα παραδοσιακά εργαλεία, τα οποία εστιάζουν αποκλειστικά στην ιχνηλάτηση αιτημάτων, το OpenTelemetry προσφέρει μια συνολική προσέγγιση στην παρατηρητικότητα, διευκολύνοντας την ενσωμάτωση με πληθώρα backends (π.χ. Jaeger, Prometheus, Grafana).

Παράλληλα, οι μεγάλοι πάροχοι υπολογιστικού νέφους προσφέρουν τις δικές τους ολοκληρωμένες λύσεις. Ενδεικτικά, το AWS X-Ray, το Google Cloud Trace και το Azure Application Insights παρέχουν ενσωμάτωση tracing στις αντίστοιχες πλατφόρμες, προσφέροντας αυξημένη ευχρηστία σε οργανισμούς που βασίζονται σε συγκεκριμένα οικοσυστήματα cloud. Ωστόσο, η στενή εξάρτηση από τις υπηρεσίες κάθε παρόχου περιορίζει τη φορητότητα και αυξάνει την πιθανότητα vendor lock-in.

Συμπερασματικά, ενώ το Jaeger έχει αναδειχθεί σε δημοφιλή επιλογή ανοικτού κώδικα για distributed tracing, η ύπαρξη πληθώρας εναλλακτικών – όπως το Zipkin, το OpenTelemetry και οι εμπορικές λύσεις των παρόχων cloud – καταδεικνύει την αυξανόμενη σημασία της παρατηρητικότητας σε περιβάλλοντα μικροϋπηρεσιών και την ποικιλία προσεγγίσεων που έχουν αναπτυχθεί για την υποστήριξή της.



Εικόνα 3.3 *Πηγή εικόνας:* ***https://logz.io/blog/zipkin-vs-jaeger/***



Εικόνα 3.4 *Πηγή εικόνας:* ***https://opentelemetry.io/docs/***

3.4 Stress Testing Tools

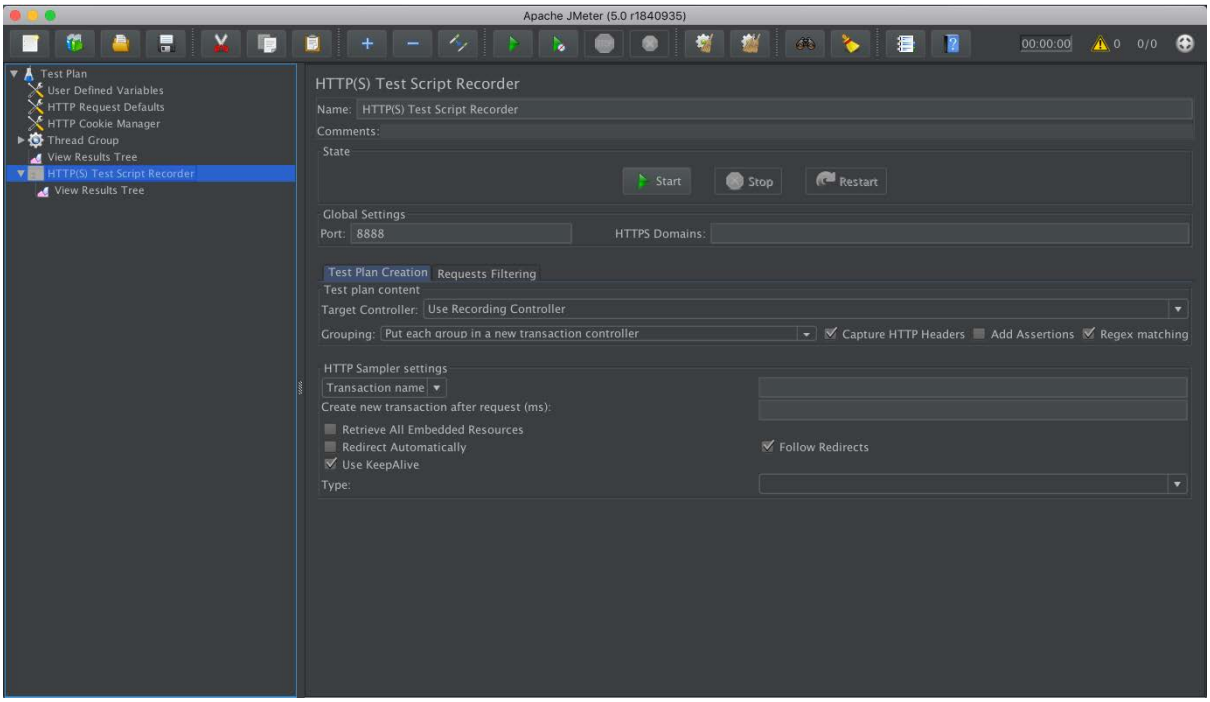
Η αξιολόγηση της απόδοσης κατανεμημένων συστημάτων και εφαρμογών μικροϋπηρεσιών απαιτεί τη χρήση εργαλείων ελέγχου φόρτωσης (load testing) και αντοχής (stress testing). Τα συγκεκριμένα εργαλεία επιτρέπουν την προσομοίωση πραγματικών συνθηκών χρήσης, δημιουργώντας ελεγχόμενα σενάρια υψηλής ζήτησης με στόχο τον εντοπισμό σημείων συμφόρησης και την αποτίμηση της αποτελεσματικότητας μηχανισμών αυτόματης κλιμάκωσης.

Πέραν του wrk2, το οποίο χρησιμοποιείται σε αυτή την εργασία για τη δημιουργία ρεαλιστικών συνθηκών φόρτου σε εφαρμογές μικροϋπηρεσιών, έχουν αναπτυχθεί και άλλες σημαντικές λύσεις. Ενδεικτικά:

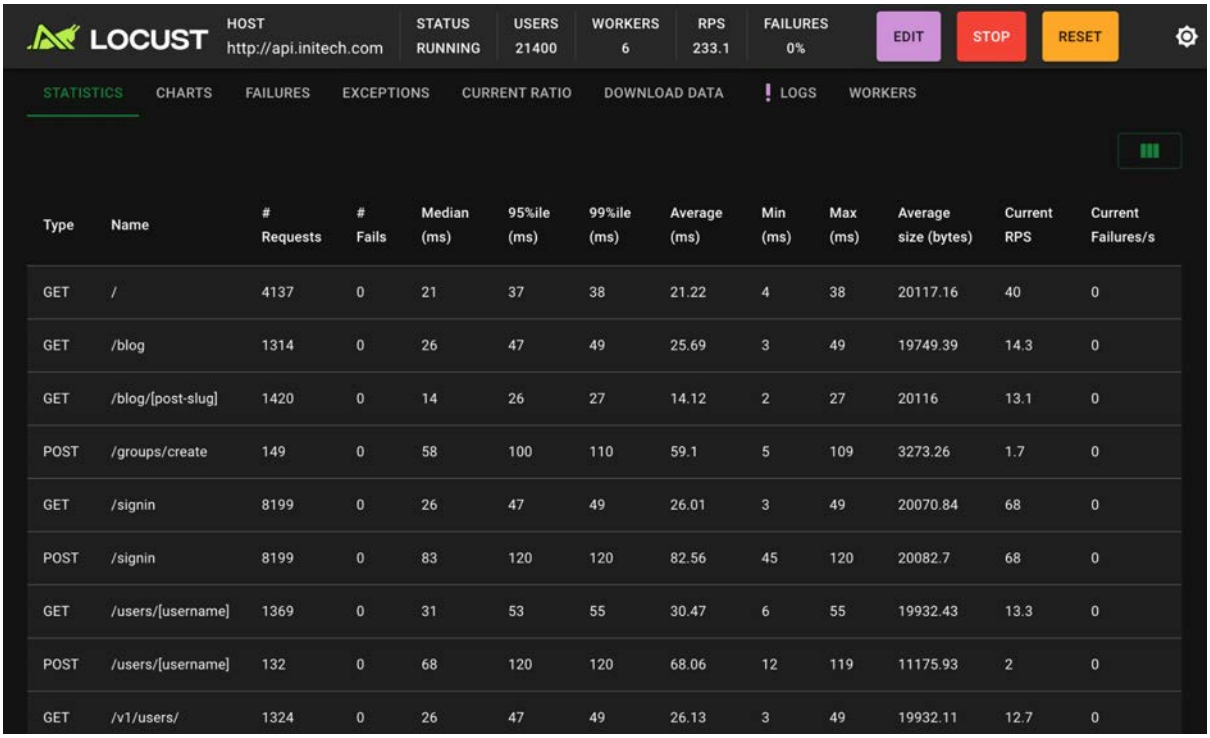
- Το Apache JMeter αποτελεί ένα από τα πιο διαδεδομένα εργαλεία ανοικτού κώδικα για τη δημιουργία σύνθετων σεναρίων φόρτου σε επίπεδο HTTP, SOAP, REST και βάσεων δεδομένων. Διακρίνεται για την ευκολία επεκτασιμότητας μέσω plugins και τη δυνατότητα χρήσης γραφικού περιβάλλοντος για τον ορισμό πολύπλοκων δοκιμών.
- Το Locust είναι ένα σύγχρονο εργαλείο load testing γραμμένο σε Python, το οποίο επιτρέπει τον ορισμό σεναρίων χρηστών με κώδικα, παρέχοντας έτσι μεγάλη ευελιξία και ευκολία αυτοματοποίησης. Διαθέτει επίσης, δυνατότητες κατανεμημένης εκτέλεσης, καθιστώντας το κατάλληλο για μεγάλης κλίμακας δοκιμές.
- Το YCSB (Yahoo! Cloud Serving Benchmark), παρόλο που εστιάζει πρωτίστως στην αποτίμηση της απόδοσης βάσεων δεδομένων NoSQL, έχει αξιοποιηθεί εκτενώς στην ερευνητική κοινότητα για την προσομοίωση υψηλών φορτίων σε υποδομές cloud.

- Επιπλέον, σε επίπεδο παρόχων cloud, παρέχονται διαχειριζόμενες λύσεις όπως το AWS Distributed Load Testing και το Azure Load Testing, οι οποίες επιτρέπουν τη δημιουργία stress tests απευθείας στο οικοσύστημα των αντίστοιχων πλατφορμών. Αν και διευκολύνουν την ενοποίηση με άλλες υπηρεσίες, συχνά εμποδίζονται από τους περιορισμούς φορητότητας.

Σε σύγκριση με αυτά τα εργαλεία, το wrk2 υπερτερεί στην ακρίβεια δημιουργίας σταθερού ρυθμού αιτημάτων (constant throughput), γεγονός που το καθιστά ιδιαίτερα χρήσιμο σε πειράματα όπου είναι κρίσιμη η μελέτη του tail latency. Ωστόσο, εργαλεία όπως το JMeter και το Locust προσφέρουν μεγαλύτερη απόδοση στη δημιουργία σύνθετων σεναρίων και μεγαλύτερη προσαρμοστικότητα σε διαφορετικά πρωτόκολλα και workloads.

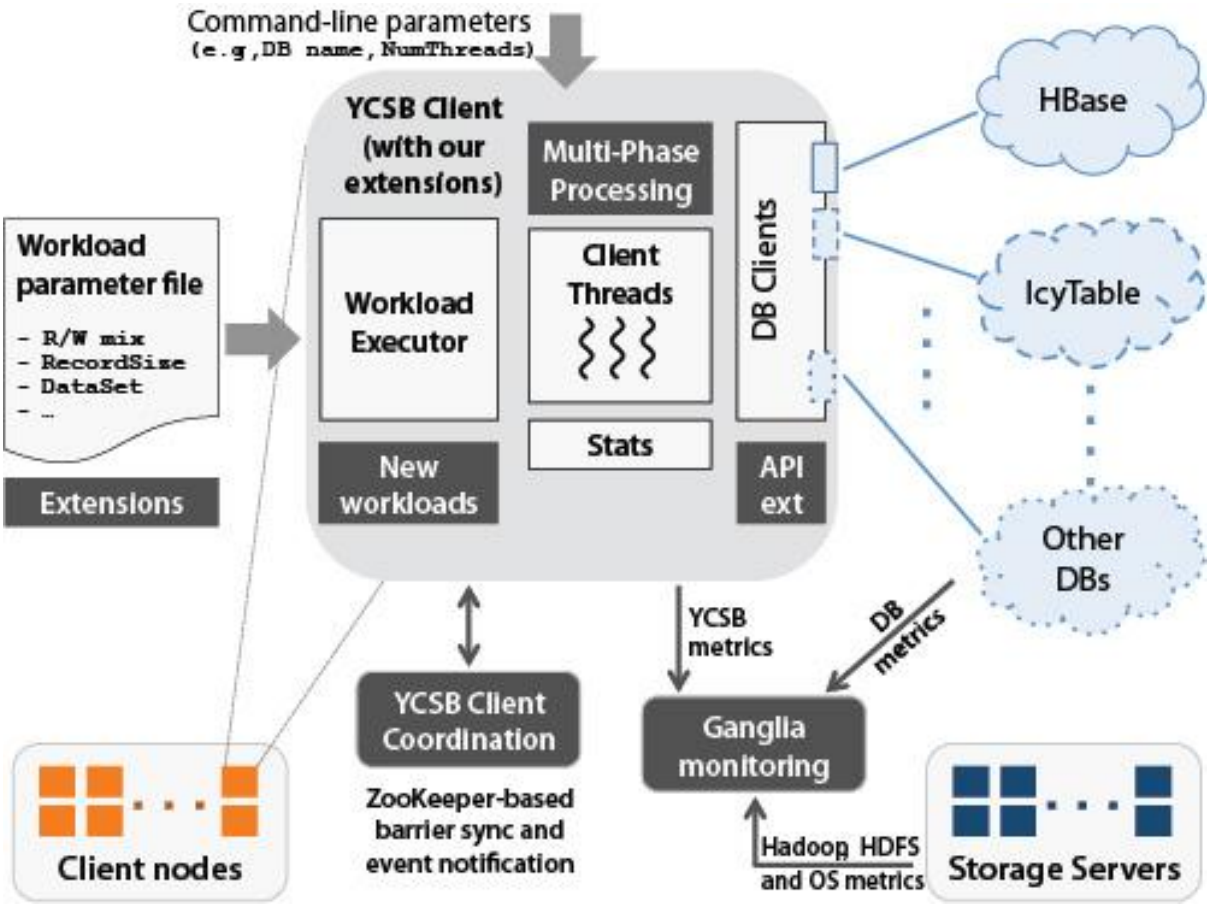


Εικόνα 3.5 *Πηγή εικόνας:* ***https://en.wikipedia.org/wiki/Apache_JMeter***



Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	4137	0	21	37	38	21.22	4	38	20117.16	40	0
GET	/blog	1314	0	26	47	49	25.69	3	49	19749.39	14.3	0
GET	/blog/[post-slug]	1420	0	14	26	27	14.12	2	27	20116	13.1	0
POST	/groups/create	149	0	58	100	110	59.1	5	109	3273.26	1.7	0
GET	/signin	8199	0	26	47	49	26.01	3	49	20070.84	68	0
POST	/signin	8199	0	83	120	120	82.56	45	120	20082.7	68	0
GET	/users/[username]	1369	0	31	53	55	30.47	6	55	19932.43	13.3	0
POST	/users/[username]	132	0	68	120	120	68.06	12	119	11175.93	2	0
GET	/v1/users/	1324	0	26	47	49	26.13	3	49	19932.11	12.7	0

Εικόνα 3.6 *Πηγή εικόνας:* ***https://locust.io/***



Εικόνα 3.7 *Πηγή εικόνας:* ***https://www.pdl.cmu.edu/ycsb++/index.shtml***

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Microsoft Azure. https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-elastic-computing

[2] Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., Mustafin, R., & Safina, L. (2017). ***Microservices: How to make your application scale. Computer Science – Research and Development***, *32(3-4), 267-271. https://doi.org/10.1007/s00450-016-0337-0*

[3] Fowler, M., & Lewis, J. (2014). ***Microservices: a definition of this new architectural term***. *Retrieved from https://martinfowler.com/articles/microservices.html*

[4] Newman, S. (2015). ***Building Microservices: Designing Fine-Grained Systems***. *O'Reilly Media.*

[5] Richardson, C. (2018). ***Microservices Patterns: With examples in Java***. *Manning Publications.*

[6] Taibi, D., Lenarduzzi, V., & Pahl, C. (2020). *Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation.* ***IEEE Cloud Computing***, *7(2), 22-32. https://doi.org/10.1109/MCC.2020.2971768*

[7] Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. ***Linux Journal***, *2014(239), 2.*

[8] Hightower, K., Burns, B., & Beda, J. (2017). ***Kubernetes: Up and Running.*** *O'Reilly Media*.

[9] Wang, G., Chen, T., & Xu, W. (2017). ***Wrk2: A constant throughput load generator.*** *ACM Symposium on Cloud Computing (SoCC).*

[10] Kubernetes Documentation. ***Horizontal Pod Autoscaling.*** https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale

[11] Amazon Web Services. *What is AWS Auto Scaling*: https://aws.amazon.com/autoscaling/

[12] Microsoft Azure. *Autoscale overview*: https://learn.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-overview

ΑΚΡΩΝΥΜΙΑ

CPU: Central Processing Unit (Κεντρική Μονάδα Επεξεργασίας)
RAM: Random Access Memory (Μνήμη Τυχαίας Προσπέλασης)
HPA: Horizontal Pod Autoscaler
SLA: Service Level Agreement
API: Application Programming Interface
REST: Representational State Transfer
VM: Virtual Machine (Εικονική Μηχανή)
CNCF: Cloud Native Computing Foundation

YAML: Μορφή Αρχείων διαμόρφωσης
CI/CD: Continuous Integration / Continuous Deployment
RPS: Requests Per Second (Αιτήματα ανά δευτερόλεπτο)