ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Πρόγραμμα Μεταπτυχιακών Σπουδών
Επιστήμη και Τεχνολογία της Πληροφορικής και των Υπολογιστών

Ειδίκευση Δικτύων Επικοινωνιών και Κατανεμημένων Συστημάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Τεχνικές Αποτίμησης Απόδοσης Σε Σύγχρονα Υπολογιστικά Νέφη

Μιχαήλ Καραβελάκης

Α.Μ. 18045

Εισηγητής: Δημήτριος Καλλέργης, Λέκτορας Εφ.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Πρόγραμμα Μεταπτυχιακών Σπουδών Επιστήμη και Τεχνολογία της Πληροφορικής και των Υπολογιστών

Ειδίκευση Δικτύων Επικοινωνιών και Κατανεμημένων Συστημάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Τεχνικές Αποτίμησης Απόδοσης Σε Σύγχρονα Υπολογιστικά Νέφη

Μιχαήλ Καραβελάκης

Α.Μ. 18045

Τριμελής εξεταστική επιτροπή

1. **<u>Επιβλέπων καθηγητής</u>: Δημήτριος Καλλέργης,** Λέκτορας Εφ. Πανεπιστημίου Δυτικής Αττικής

   Digitally signed by Dimitrios Kallergis
   Date: 2022.03.15 21:26:37 +02'00'

2. **<u>Μέλος</u>: Γραμματή Πάντζιου,** Καθηγήτρια Πανεπιστημίου Δυτικής Αττικής

   Grammati Pantziou
   Digitally signed by Grammati Pantziou
   Date: 2022.03.16 23:07:42 +02'00'

3. **<u>Μέλος</u>: Βασίλειος Μάμαλης,** Καθηγητής Πανεπιστημίου Δυτικής Αττικής

   Vasileios Mamalis
   Digitally signed by Vasileios Mamalis
   Date: 2022.03.16 23:22:19 +02'00'

Αθήνα, Μάρτιος 2022

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος, Καραβελάκης Μιχαήλ του Εμμανουήλ, με αριθμό μητρώου 18045, φοιτητής του Προγράμματος Μεταπτυχιακών Σπουδών Επιστήμης και Τεχνολογίας της Πληροφορικής και των Υπολογιστών του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής, δηλώνω ότι:

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

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

Επιθυμώ την απαγόρευση πρόσβασης στο πλήρες κείμενο της εργασίας μου μέχρι 15-03-2023 και έπειτα από αίτηση μου στη Βιβλιοθήκη και έγκριση του επιβλέποντα καθηγητή».

Ο Δηλών

Digitally signed by Dimitrios Kallergis
Date: 2022.03.15 21:27:07 +02'00'

Περίληψη

Η υπολογιστική νέφους και οι νέες τεχνολογίες που την πλαισιώνουν είναι πλέον απαραίτητα εργαλεία τα τελευταία δέκα έτη. Η ραγδαία αύξηση της χρήσης υπολογιστικών πόρων και η εκτέλεση πολύπλοκων εργασιών, γέννησαν την ανάγκη για άμεση, αυτόματη και ακριβής διαχείριση των νεφοϋπολογιστικών πόρων και υπηρεσιών. Λύση σε αυτά τα σύγχρονα ζητήματα μπορεί να δώσει η χρήση εξειδικευμένων εργαλείων παρακολούθησης και διαχείρισης πόρων. Ο σκοπός της παρούσας διπλωματικής εργασίας είναι η κατανόηση των υπαρχόντων μεθόδων αποτίμησης απόδοσης σε σύγχρονα υπολογιστικά νέφη και η βελτιστοποίηση της απόδοσης αυτής. Αντικείμενο μελέτης είναι η καταγραφή των πλέον διαδεδομένων εργαλείων διαχείρισης περιεκτών και η αποτύπωσή των χαρακτηριστικών τους, δίνοντας ιδιαίτερη βαρύτητα στο συγκεκριμένο ενορχηστρωτή. Επιπλέον, είναι η υλοποίηση εργαλείου αποτίμησης απόδοσής και ελαστικότητας του σε συνθήκες υψηλής ζήτησης, με τη χρήση μεθόδων και εργαλείων έχοντας ως απώτερο σκοπό την πιθανή εύρεση λύσεων, εστιάζοντας σε ένα εργαλείο οριζόντιας αυτόματης κλιμακοθετησιμότητας του ενορχηστρωτή, για μεγαλύτερη απόδοση παράλληλα με τη μείωση χρήσης υπολογιστικών πόρων.

Λέξεις-Κλειδιά: Περιέκτες, Ενορχηστρωτές, Υπολογιστική Νέφους, Αποτίμηση Απόδοσης.

Abstract

Cloud computing and the new technologies that frame it, are indispensable tools in workplaces the past decade. The rapid increase in the use of computer resources and the execution of complex tasks, brought the need for immediate, automatic and accurate management of cloud resources and services. The use of specialized monitoring and resource management tools can provide a solution to these issues. The purpose of this master thesis is the comprehension of the existing performance appraisal methods in modern cloud computing and the optimization of this performance. Objects of study is the listing of the most popular container management tools and the listing of their characteristics, focusing on a specific orchestrator. Additional object, is the implementation of a tool to evaluate its performance and resilience in conditions of high demand, using methods and tools with the ultimate goal of finding possible solutions, focusing on a horizontal autoscaling tool, for greater performance while reducing the use of computing resources.

Keywords: Containers, Orchestrators, Cloud, Performance Assessment.

Ευχαριστίες

Ευχαριστώ τον επιβλέποντα καθηγητή μου Καλλέργη Δημήτριο, για την υποδειγματική βοήθεια, τη συνεχή καθοδήγησή του και την εμπιστοσύνη που μου έδειξε.

Επιπλέον, ευχαριστώ πολύ τους φίλους μου, Καβουσανό Γιώργο και Παπαδογιάννη Νίκο για τη σημαντική βοήθεια που μου προσέφεραν τη στιγμή που τη χρειάστηκα.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου, για την υπομονή τους, καθώς και την υποστήριξη την οποία έλαβα όλα αυτά τα χρόνια.

Πίνακας Περιεχομένων

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κατάλογος Συντομογραφιών

HPA	Horizontal Pod Autoscaler
API	Application Programming Interface
CPU	Central Processing Unit
SaaS	Software as a Service
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
CSV	Comma-Separated Values
APT	Advanced Packaging Tool
REST	Restful
CLI	Command Line Interface
CNCF	Cloud Native Computing Foundation
INC	Incorporated
DNS	Domain Name System
UI	User Interface
CRI	Container Runtime Interface
HTTPS	Hypertext Transfer Protocol Secure
SSH	Secure Shell
VPA	Vertical Pod Autoscaling
CA	Cluster Autoscaling
IP	Internet Protocol
RAM	Random Access Memory
RC	Resource Controller
V	Version
JVM	Java Virtual Mashine
HTTP	Hypertext Transfer Protocol
PHP	Hypertext Preprocessor
ASP.NET	Active Server Pages Network
SOAP	Simple Object Access Protocol
FTP	File Transfer Protocol

JDBC	Java Database Connectivity
LDAP	Lightweight Directory Access Protocol
JMS	Java Message Service
SMTP(S)	Simple Mail Transfer Protocol (Secure)
POP3(S)	Post Office Protocol (Secure)
IMAP(S)	Internet Message Access Protocol (Secure)
XML	Extensible Markup Language
JSON	JavaScript Object Notation
ΕΔΕΤ	Εθνικό Δίκτυο Υποδομών Τεχνολογίας και Έρευνας
GB	Gigabyte
LTS	Long Term Support
iBuC	Internet Bearer Underwriting Corporation
AVG	Average
MIN	Minimum
MAX	Maximum

Κεφάλαιο 1 - Εισαγωγή

1.1 Πρόλογος

Τα τελευταία χρόνια παρατηρείται σημαντική αύξηση του αριθμού των Οργανισμών που καλύπτουν μέρος των εργασιών τους μέσω της υπολογιστικής νέφους, καθώς και με τη χρήση της τεχνολογίας των μικροϋπηρεσιών και των περιεκτών.

Τα υπολογιστικά νέφη (Clouds) αφορούν «*ένα μοντέλο που δίνει τη δυνατότητα της συνεχούς, εύκολης και υψηλών απαιτήσεων πρόσβασης σε μια κοινόχρηστη συλλογή ρυθμιζόμενων υπολογιστικών πόρων, οι οποίοι τροφοδοτούνται και απελευθερώνονται με ελάχιστη προσπάθεια διαχείρισης και αλληλεπίδρασης παροχής υπηρεσιών*»[1]. Τα υπολογιστικά νέφη βασίζονται στην τεχνική της εικονικοποίησης (virtualization). Με την εικονικοποίηση διαχωρίζεται ο εξοπλισμός από το λογισμικό στο οποίο είναι εγκατεστημένες οι εφαρμογές, οι οποίες μπορούν να λειτουργούν πλέον ανεξάρτητα, με διαφορετικά λειτουργικά συστήματα, και να επιτελούν διαφορετικές εργασίες για διαφορετικούς χρήστες διαμοιράζοντας ανάλογα τους πόρους ώστε οι εφαρμογές να προσφέρουν τη μέγιστη δυνατή απόδοση [1]. Η τεχνολογία της εικονικοποίησης συνδυάζοντας τη χρήση των περιεκτών (containers) και των μικροϋπηρεσιών (microservices) πλέον έχει να ξεπεράσει σε απόδοση και δυνατότητες αυτή των εικονικών μηχανών. Οι μικροϋπηρεσίες είναι μια αρχιτεκτονική προσέγγιση στη δημιουργία εφαρμογών ακολουθώντας τον ορισμό της αρχής της μεμονωμένης αρμοδιότητας. Αναφέρονται ως ένας τρόπος σχεδιασμού ενός προϊόντος λογισμικού αποτελούμενο από ομάδες μικρότερων, αυτόνομων υπηρεσιών με κοινά χαρακτηριστικά, όπου η καθεμιά τους είναι αρμόδια για μία συγκεκριμένη δραστηριότητα [2], [3].

Η εκτέλεση μιας εφαρμογής η οποία εμπεριέχεται σε έναν περιέκτη προϋποθέτει την ύπαρξη μηχανισμού διαχείρισης των περιεκτών. Η διαχείριση πολύπλοκων εφαρμογών, αποτελούμενων από ανεξάρτητες μεταξύ τους μικροϋπηρεσίες οι οποίες μπορούν να συνεργάζονται φέρνοντας το επιθυμητό αποτέλεσμα, επιτυγχάνεται μέσω των μηχανισμών ενορχήστρωσης των περιεκτών [4]. Ένας ενορχηστρωτής δίνει τη δυνατότητα διάθεσης, διαμόρφωσης, ελέγχου και

[1] The NIST Definition of Cloud Computing. https://csrc.nist.gov/publications/detail/sp/800-145/final

παρακολούθησης δυναμικά μιας εφαρμογής. Το Kubernetes είναι ένας από τους πλέον διαδεδομένους ενορχηστρωτές παγκοσμίως. Μια από τις σημαντικότερες εφαρμογές τις οποίες διαθέτει το Kubernetes είναι και ο Horizontal Pod Autoscaler (HPA). Ο Horizontal Pod Autoscaler υλοποιείται ως ένα Kubernetes API. Τα κυριότερα ζητήματα που προκύπτουν κατά τη χρήση του HPA είναι η δυσκολία καθορισμού πόρων [π.χ. κεντρική μονάδα επεξεργασίας (CPU), καθορισμός της σήμανσης ορίων στάθμης (watermark), ανεπάρκειες που προκαλούνται κατά την υψηλή ή τη χαμηλή ζήτηση (upswing ή downswing) και γενικότερα η κατανόηση της συμπεριφοράς της εφαρμογής κατά την ανάπτυξή της (deployment), έτσι ώστε να μπορεί να επιτευχθεί μια αξιόπιστη υπηρεσία (service)] [5]. Η απόδοση δύναται να μετρηθεί εξετάζοντας τους περιέκτες, τις υπηρεσίες και τα χαρακτηριστικά της συστοιχίας (cluster) συνολικά, παρέχοντας λεπτομερείς πληροφορίες σχετικά με τη χρήση των πόρων μιας εφαρμογής, σε κάθε ένα από αυτά τα επίπεδα. Αυτές οι πληροφορίες επιτρέπουν να αξιολογηθεί η απόδοση της εφαρμογής ώστε να βρεθούν και να αφαιρεθούν πιθανά σημεία συμφόρησης (bottlenecks) για τη βελτίωση της συνολικής απόδοσης [6].

Η υπάρχουσα βιβλιογραφία προσφέρει πλούσια και εξειδικευμένη γνώση σχετικά με τους ενορχηστρωτές· όμως, δεν υπάρχουν επαρκείς ελληνικές βιβλιογραφικές πηγές που να εξειδικεύονται πάνω στα υπάρχοντα από τους ενορχηστρωτές εργαλεία.

1.2 Σκοπός και Αντικείμενο Μελέτης

Σκοπός της μεταπτυχιακής εργασίας είναι η κατανόηση των υπαρχόντων μεθόδων αποτίμησης απόδοσης σε σύγχρονα υπολογιστικά νέφη και η βελτιστοποίηση της απόδοσης, μέσω των διαθέσιμων εξειδικευμένων εργαλείων τα οποία παρέχονται. Αντικείμενο μελέτης είναι η καταγραφή των πλέον διαδεδομένων εργαλείων διαχείρισης περιεκτών και η αποτύπωσή των χαρακτηριστικών, των δυνατοτήτων και των κυριότερων εργαλείων τους, όπως επίσης και οι διαφορές μεταξύ τους, δίνοντας ιδιαίτερη βαρύτητα στον ενορχηστρωτή Kubernetes. Επιπλέον, είναι η υλοποίηση εργαλείου αποτίμησης απόδοσής και ελαστικότητας του σε συνθήκες υψηλής ζήτησης, με τη χρήση μεθόδων και εργαλείων έχοντας ως απώτερο σκοπό την πιθανή

εύρεση λύσεων για μεγαλύτερη απόδοση παράλληλα με τη μείωση χρήσης υπολογιστικών πόρων.

1.3 Δομή Μεταπτυχιακής Εργασίας

Η παρούσα εργασία αποτελείται από έξι (6) κεφάλαια. Στο **πρώτο κεφάλαιο**, περιλαμβάνεται ο πρόλογος και αναφέρονται ο σκοπός και το αντικείμενο της μελέτης. Το **δεύτερο κεφάλαιο**, ξεκινά με μια σύνοψη γύρω από τις τεχνολογίες νέφους, γίνεται εντοπισμός και προσέγγιση του προβλήματος που θα μας απασχολήσει, καθώς και αναφέρονται οι μέθοδοι παρακολούθησης συστοιχίας και απόδοσης φορτίου. Επιπλέον, αναλύονται δυο μοντέλα κατανομής πιθανοτήτων. Στο **τρίτο κεφάλαιο**, αναφέρονται σε θεωρητικό επίπεδο οι τεχνολογίες που χρησιμοποιήθηκαν και αναλυτικά τα εργαλεία και οι μέθοδοι ώστε να διεκπεραιωθεί το πείραμα και να ληφθούν τα επιθυμητά αποτελέσματα. Στο **τέταρτο κεφάλαιο** διατυπώνονται οι απαιτήσεις της πειραματικής διάταξης, η σχεδίαση και η αρχιτεκτονική της. Επιπρόσθετα αναλύεται η επιλογή παραμέτρων του πειράματος καθώς και ο τρόπος υλοποίησής του. Στη συνέχεια, στο **πέμπτο κεφάλαιο** αποτυπώνονται τα αποτελέσματα καθώς και τα συμπεράσματα στα οποία οδηγηθήκαμε μέσω του πειράματος. Τέλος, το **έκτο κεφάλαιο** ολοκληρώνει την παρούσα εργασία, αναφέροντας προτάσεις για μελλοντικές επεκτάσεις της μεταπτυχιακής εργασίας.

Κεφάλαιο 2 - Βιβλιογραφική Επισκόπηση

2.1 Εισαγωγή

Έχοντας ένα μεγάλο αριθμό τύπων υπολογιστικής νέφους να χρησιμοποιούνται ταυτόχρονα σε έναν οργανισμό ή μια επιχείρηση, δημιουργείται η ανάγκη διαχείρισης των υπολογιστικών νεφών αυτών, συμπεριλαμβανομένων των δεδομένων και των εφαρμογών που φέρουν [7].

Οι ενορχηστρωτές (orchestrators), όπως είναι μεταξύ άλλων ο Kubernetes, έρχονται να δώσουν τη λύση στην πολυπλοκότητα που διέπει την υπολογιστική νέφους. Δημιουργούν, δηλαδή, μια αυτοματοποιημένη διαδικασία διαχείρισης πολλαπλών εργασιών, έχοντας ως κύριο στόχο τη δημιουργία μίας συνεχόμενης και ταυτόχρονα απρόσκοπτης ροής εργασίας. Η διαδικασία αυτή επιτυγχάνεται με ένα μεγάλο αριθμό εργαλείων οι οποίοι διαθέτουν και αναπτύσσονται συνεχώς.

Ένα πολύ χρήσιμο εργαλείο/εφαρμογή κλιμακοθετησιμότητας, η οποία παρέχεται από τον Kubernetes, είναι ο Horizontal Pod Autoscaler. Ο Horizontal Pod Autoscaler κλιμακώνει αυτόματα τον αριθμό των pods με βάση την παρατηρούμενη χρήση των πόρων, με σκοπό να παρέχει τους απαραίτητους πόρους για τη διεκπεραίωση μιας εργασίας με τη μέγιστη απόδοση, κοιτώντας συνολικά τους πόρους της συστοιχίας και κατανέμοντάς τους ανάλογα και ισάξια.

2.2 Εικονικοποιημένες Υπηρεσίες και Απόδοση

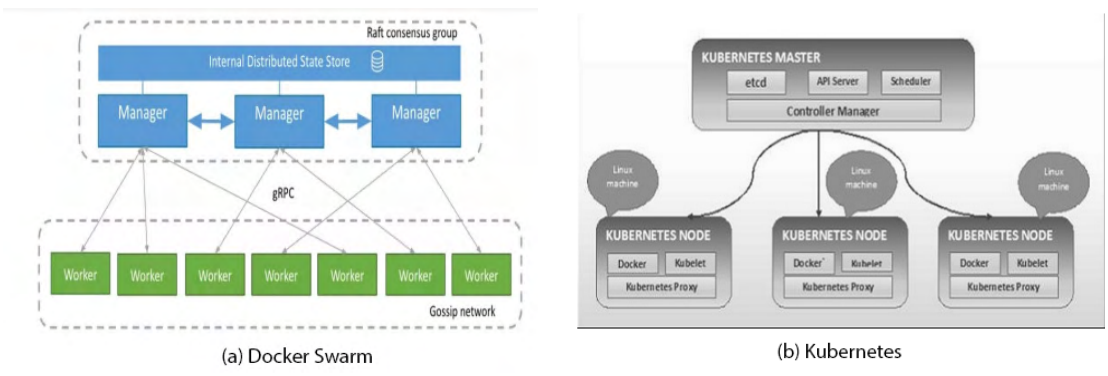
2.2.1 Περιέκτες, Ενορχηστρωτές, Μικροϋπηρεσίες

Οι J. Shah et al. [8] μελέτησαν και ανέλυσαν το μηχανισμό του Docker, τον ενορχηστρωτή Kubernetes καθώς και την πλατφόρμα Google Cloud. Περιέγραψαν τον τρόπο λειτουργείας τους, τί παρέχουν, τα θετικά τους στοιχεία όπως και τις διαφορές μεταξύ των ενορχηστρωτών Kubernetes και Docker Swarm. Κατέληξαν ότι το Docker και το Kubernetes εργάζονται σε διαφορετικά επίπεδα. Το Docker που χρησιμοποιείται ως λογισμικό πληροφορικής και χρησιμοποιεί την τεχνολογία της περιεκτοποίησης. Κύριο πλεονέκτημα της χρήσης του Kubernetes είναι η

ενορχήστρωση μεταξύ πολλαπλών εφαρμογών, όπως επίσης είναι χρήσιμο στην κλιμάκωση εφαρμογών σε ελάχιστο χρόνο. Δίνει τη δυνατότητα διαχείρισης, εξοικονόμηση κόστους καθώς και ορθής χρήσης του συνόλου των πόρων.

Οι Yao Pan et al. [9] σύγκριναν τους γνωστούς ενορχηστρωτές περιεκτών ανοικτού κώδικα Docker Swarm και Kubernetes διεξάγοντας πειραματικές δοκιμές. Συμπέραναν ότι ενώ το Docker Swarm υπερτερεί σε απόδοση του Kubernetes, ο δεύτερος ενορχηστρωτής παρουσιάζει ευελιξία, είναι ωριμότερος όσον αφορά τη λειτουργικότητα σε περιβάλλοντα παραγωγής και είναι αποτελεσματικότερος σε σύνθετες περιπτώσεις.

Οι N. Marathe et al. [10] διερεύνησαν την κατανεμημένη υπολογιστική σε υπολογιστικά νέφη, εστιάζοντας στην εξισορρόπηση φορτίου μεταξύ των κόμβων χρησιμοποιώντας τους ενορχηστρωτές Docker Swarm και Kubernetes, μελετώντας παράλληλα τη δομή τους όπως φαίνεται στην Εικόνα 2.1, με κύρια διαφορά, τον Kubernetes να έχει έναν και μόνο διαχειριστή του συνόλου της συστοιχίας ενώ το Docker Swarm να διαθέτει πολλαπλούς ομαδοποιημένους διαχειριστές οι οποίοι επικοινωνούν μεταξύ τους και με τους εργάτες (workers). Επιπλέον, μελέτησαν τους τρόπους πρόσβασης στις υπηρεσίες από τους κόμβους εντός της συστοιχίας με τη βοήθεια των δύο ενορχηστρωτών (αναλύοντας τα εξαρτήματα (components) τα οποία χρησιμοποιούν για να το πετύχουν αυτό) και κατέγραψαν τις διαφορές μεταξύ τους. Οι ερευνητές κατέληξαν ότι το Kubernetes είναι ανώτερο του Docker Swarm, παρόλη την πολυπλοκότητα που το διακρίνει στη διαδικασία εγκατάστασής του, διότι η αποτελεσματικότητά του καθιστά τη δυσκολία και συνθετότητα της εγκατάσταση αμελητέα.



(a) Docker Swarm

(b) Kubernetes

Εικόνα 2.1: Δομές Docker Swarm - Kubernetes [10]

Οι Q. Lei et al. [11] εξέτασαν τα ζητήματα απόδοσης των μικροϋπηρεσιών, περιέγραψαν την αρχιτεκτονική του Kubernetes για την υλοποίηση του χρονοπρογραμματισμού των πόρων και πρότειναν μια μέθοδο δοκιμής απόδοσης με το Kubemark. Κατέγραψαν μετρικές για την αξιολόγηση της απόδοσης του συστήματος από άκρο σε άκρο: **(α)** καθυστέρηση (latency) της αναστροφής κλήσης των υπηρεσιών API (API callback) (*δηλ.* χρόνος εκκίνησης από άκρο σε άκρο, χρόνος εκκίνησης των ελεγκτών αντιγραφής (replication controllers και χρόνος εκκίνησης των υπηρεσιών API) και **(β)** χρόνος έναρξης των pods.

Οι L. A. Vayghan et al. [3] διεξήγαγαν στην έρευνά τους πειραματικά σενάρια βλάβης σε: **(α)** pods, **(β)** κόμβους και μετρώντας τη διαθεσιμότητα του Kubernetes μέσω των μετρικών: **(α)** χρόνος αντίδρασης, **(β)** χρόνος επιδιόρθωσης, **(γ)** χρόνος ανάκαμψης και **(δ)** χρόνος διακοπής λειτουργίας. Κατέληξαν ότι το Kubernetes επιτρέπει την ίαση μέσω των ενεργειών ανάκτησης βλάβης και συχνά αξιολογούνται μέσω εσωτερικών λειτουργιών, αντιδρώντας καλύτερα σε σύγκριση με την αντίδρασή του σε αστοχίες που προκύπτουν από κάποια εξωτερική σκανδάλιση.

Οι D. Kallergis et al. [12] μελέτησαν τη χρήση -βασισμένων σε πολιτικές- εξουσιοδοτήσεων με ανεξάρτητες μικροϋπηρεσίες που υλοποιούνται σε περιέκτες εντός της υποκείμενης υβριδικής νεφοϋπολογιστικής υποδομής. Οι ερευνητές πρότειναν ένα πλαίσιο (framework) εξουσιοδοτήσεων ασφάλειας και το αντιπαρέθεσαν με υφιστάμενα πλαίσια μικροϋπηρεσιών χρησιμοποιώντας πολλαπλές κατανομές φορτίου με διαφορετικά πλήθη χρηστών. Τα αποτελέσματα των πειραμάτων έδειξαν ότι υπήρξε σημαντική υπεροχή στην απόδοση του προτεινόμενου πλαισίου όσον αφορά στην καθυστέρηση, στην διεκπεραιωτική ικανότητα και στα επιτυχημένα αιτήματα.

2.2.2 Αυτόματη Κλιμακοθετησιμότητα

Οι E. Casalicchio et al. [13] εστίασαν στην επιλογή των καταλληλότερων μετρικών απόδοσης με βάση τη CPU για την ενεργοποίηση ενεργειών αυτόματης κλιμάκωσης στον ενορχηστρωτή Kubernetes, διερευνώντας τη χρήση σχετικών και απόλυτων μετρικών και δημιουργώντας τον δικό τους αλγόριθμο για μεγαλύτερη απόδοση. Τα αποτελέσματα έδειξαν ότι, ενώ ο Horizontal Pod Autoscaler χρησιμοποιεί σχετικές

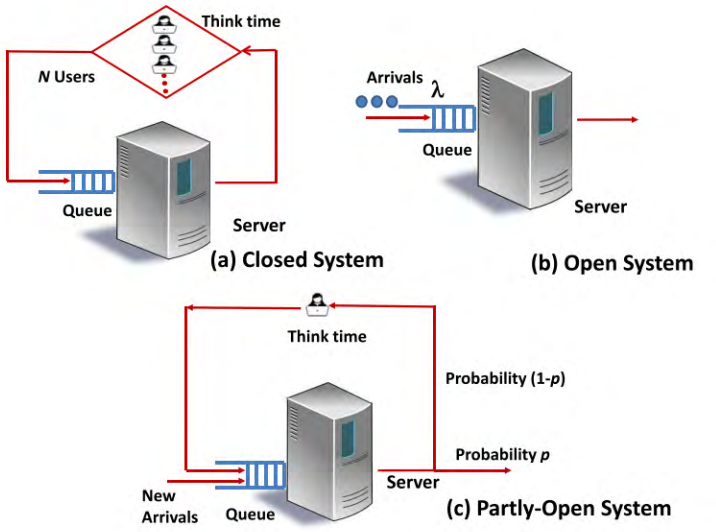
μετρήσεις, για έντονο φόρτο εργασίας της CPU, η χρήση απόλυτων μετρικών επιτρέπει μεγαλύτερη ακρίβεια στις αποφάσεις κλιμάκωσης.

Οι T. Ye al. [14] πρότειναν ένα εργαλείο αυτόματης κλιμακοθετησιμότητας. Σχεδίασαν μια υβριδική στρατηγική κλιμακοθετησιμότητας, λαμβάνοντας υπ' όψη τους δύο βασικούς τύπους κλιμακοθετησιμότητας: **(α)** οριζόντια και **(β)** κάθετη. Η υβριδική κλιμακοθετησιμότητα βασίζεται σε ένα μοντέλο πρόβλεψης ζήτησης πόρων σε ταχέως μεταβαλλόμενους φόρτους εργασίας. Τα αποτελέσματα βασίστηκαν στη χρήση πόρων της CPU και έδειξαν ότι το προτεινόμενο σύστημα αυτόματης κλιμακοθετησιμότητας ξεπερνά τις υπάρχουσες προσεγγίσεις.

Όσον αφορά τους διαθέσιμους πόρους οι οποίοι παρέχονται προς χρήση στον Horizontal Pod Autoscaler, οι M. Amaral et al. [15] ανέλυσαν δύο μοντέλα τα οποία μπορούν να χρησιμοποιηθούν για την υλοποίηση αρχιτεκτονικής μικροϋπηρεσιών χρησιμοποιώντας περιέκτες και σύγκριναν την απόδοση με σημεία αναφοράς τη λειτουργία της CPU και του δικτύου στα δύο προαναφερθέντα μοντέλα αρχιτεκτονικής μικροϋπηρεσιών. Τα αποτελέσματα της έρευνας έδειξαν ότι η προσέγγιση εμπερικλειόμενων (nested) περιεκτών είναι ένα αποδοτικό μοντέλο, κυρίως χάρη στη βελτιωμένη κοινή χρήση πόρων (ίδια μνήμη και αποθηκευτικό μέσο). Επιπλέον, οι εμπερικλειόμενοι περιέκτες δεν έχουν σημαντικό αντίκτυπο στην απόδοση της CPU, ωστόσο, υπάρχουν αντισταθμίσεις όσον αφορά στην απόδοση του δικτύου.

2.2.3 Γεννήτριες Φορτίου

Οι M. Curiel και A. Pont [16] μελέτησαν τα επικρατέστερα χαρακτηριστικά γεννητριών φορτίου για διαδικτυακά συστήματα. Για τον σκοπό αυτό, οι ερευνητές κατηγοριοποιούν τα εργαλεία παραγωγής φορτίου σε τρεις ομάδες: **(α)** γεννήτριες ίχνους (trace generators), **(β)** γεννήτριες κίνησης (traffic generators) και **(γ)** γεννήτριες φόρτου (workload generators). Επιπλέον κατηγοριοποίηση των εργαλείων γίνεται βάσει των παραμέτρων που καθορίζουν την ένταση φορτίου (load intensity) και τις απαιτήσεις πόρων (resource demands) σε τρία μοντέλα συστήματος (Εικόνα 2.2): **(α)** κλειστό (closed), **(β)** ανοικτό (open) και **(γ)** μερικώς ανοικτό (partly-open).



Εικόνα 2.2: Μοντέλα Συστήματος [16]

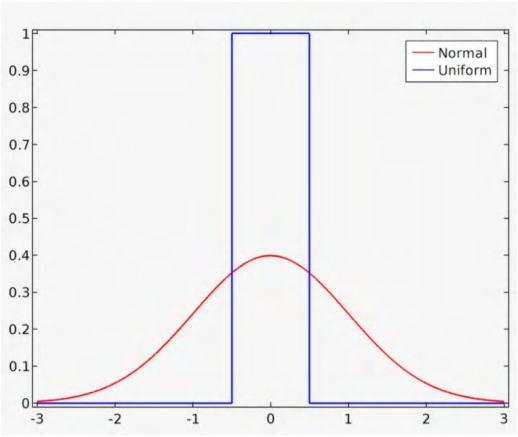
Τέλος, οι ερευνητές [16] εντοπίζουν ότι η επιλογή των εργαλείων μπορεί να γίνει με βάση **(α)** τη διαδικασία γέννησης αιτήματος (*δηλ.* βάσει αιτήματος ή βάσει συνεδρίας) και **(β)** την υλοποίηση (*δηλ.* σύγχρονα ή ασύγχρονα).

2.3 Κατανομές Πιθανοτήτων

Υπάρχουν διάφορες διαφορετικές κατανομές πιθανότητας. Κάθε μία από αυτές τις διανομές έχει μια συγκεκριμένη εφαρμογή και χρήση που είναι κατάλληλη για μια συγκεκριμένη ρύθμιση.

2.3.1 Ομοιόμορφη Κατανομή

Η ομοιόμορφη κατανομή (uniform distribution) ανήκει στην οικογένεια των συμμετρικών κατανομών πιθανοτήτων, κατά την οποία όλα τα αποτελέσματα δύναται να συμβούν με ακριβώς την ίδια πιθανότητα. Η πιθανότητα είναι σταθερή (Εικόνα 2.3) με κάθε μεταβλητή να έχει ίσες πιθανότητες να είναι το αποτέλεσμα [17].



Εικόνα 2.3: Ομοιόμορφη Κατανομή (πηγή: https://www.comsol.com)

Η ομοιόμορφη κατανομή περιλαμβάνει δύο υποκατηγορίες, με βάση τα πιθανά αποτελέσματα [17]:

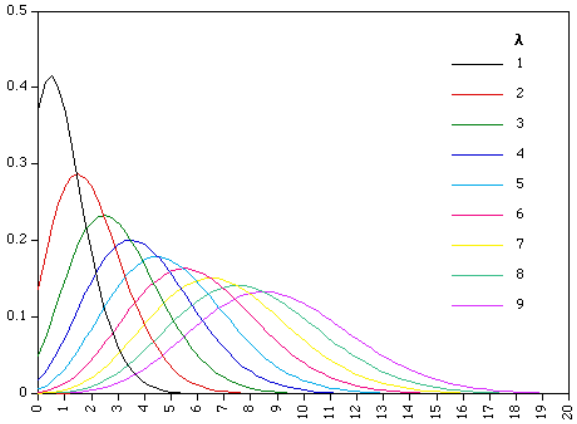
- **Διακριτή ομοιόμορφη κατανομή** - Στη διακριτή ομοιόμορφη κατανομή η πιθανότητα των αποτελεσμάτων είναι ισόποσα πιθανή έχοντας πεπερασμένες τιμές.
- **Συνεχής ομοιόμορφη κατανομή** – Γνωστή και ως ορθογώνια κατανομή, διαθέτει άπειρο αριθμό μετρήσιμων σημείων τα οποία είναι ισόποσα πιθανό να εμφανιστούν. Μια συνεχής τυχαία μεταβλητή μπορεί να πάρει οποιαδήποτε πραγματική τιμή σε ένα προκαθορισμένο εύρος τιμών.

2.3.2 Κατανομή Poisson

Η κατανομή Poisson (Poisson distribution), είναι μία διακριτή συνάρτηση κατανομής που εκφράζει την πιθανότητα ενός δεδομένου αριθμού γεγονότων που συμβαίνουν σε ένα σταθερό διάστημα χρόνου ή/και χώρου αν αυτά τα γεγονότα συμβαίνουν με ένα γνωστό μέσο ρυθμό και είναι ανεξάρτητα από το χρονικό διάστημα από την τελευταία περίπτωση. Η κατανομή Poisson μπορεί επίσης να χρησιμοποιηθεί για τον αριθμό γεγονότων σε άλλα καθορισμένα διαστήματα όπως η απόσταση, η επιφάνεια ή ο όγκος [18].

Δεδομένου μόνο του μέσου ρυθμού, για ένα ορισμένο διάστημα παρακολούθησης (αριθμό αλληλογραφίας ανά μέρα, τηλεφωνήματα ανά ώρα, κτλ.), και υποθέτοντας ότι η διαδικασία, ή ο συνδυασμός των διαδικασιών , που παράγουν τα γεγονότα είναι ουσιαστικά τυχαίος, η κατανομή Poisson καθορίζει πόσο πιθανό είναι ότι ο

δεδομένος αριθμός, κατά την διάρκεια μίας περιόδου παρατήρησης. Αυτό σημαίνει ότι, προβλέπει τον αριθμό διάδοσης γύρω από ένα γνωστό ρυθμό εξάπλωσης [19].



Εικόνα 2.4: Κατανομή Poisson [19]

Η κατανομή Poisson (Εικόνα 2.4) έχει την παράμετρο λ που δηλώνει τη μέση τιμή αριθμού εμφανίσεων ενός γεγονότος, η οποία είναι ανεξάρτητη της τελευταίας χρονικής στιγμής εμφάνισης του γεγονότος (Εικόνα 2.5) [19].

$$P_\lambda(X = k) = \frac{\lambda^k}{k!}\,\mathrm{e}^{-\lambda}$$

Εικόνα 2.5: Τύπος Κατανομής Poisson (πηγή: https://www.ml-science.com)

2.4 Εντοπισμός και Προσέγγιση Προβλήματος

Ήδη από την προηγούμενη δεκαετία, παρατηρείται ολοένα αυξανόμενη μετακίνηση σε υποδομές υπολογιστικού νέφους με σκοπό, μεταξύ άλλων, την εκτέλεση σύνθετων και πολύπλοκων εργασιών. Επομένως, προέκυψε η ανάγκη για άμεση, αυτόματη και ακριβή διαχείριση των νεφοϋπολογιστικών πόρων και υπηρεσιών. Απώτερος στόχος είναι η βέλτιστη απόδοση των παροχών ενός οργανισμού σε συνδυασμό με την ελάχιστη δυνατή χρήση των πόρων που έχει στη διάθεσή του. Λύση σε αυτά τα σύγχρονα ζητήματα μπορεί να δώσει η χρήση εξειδικευμένων λογισμικών και εργαλείων τους, με σκοπό τη διαχείριση περιεκτών και την παρακολούθηση/διαχείριση πόρων καθώς και η παρακολούθηση απόδοσης φορτίου [20], [21]. Η επιλογή του εργαλείου διαχείρισης (*δηλ.* του ενορχηστρωτή) έγινε λαμβάνοντας υπ' όψη τις ερευνητικές εργασίες των J. Shah et al. [8], Yao Pan et al. [9] και N. Marathe et al. [10], όπου όλες οι ερευνητικές ομάδες, μετά από σύγκριση των

δύο διασημότερων εργαλείων ενορχήστρωσης, κατέληξαν σε ένα κοινό συμπέρασμα: διέκριναν την απλότητα που διέπει τον Docker Swarm, όμως αναγνώρισαν την ανωτερότητα του Kubernetes στην ενορχήστρωση, ειδικά στην περίπτωση σύνθετων διαδικασιών.

Με τη συγκριτική έρευνα των M. Amaral et al. [15] μεταξύ των δυο μοντέλων αρχιτεκτονικής μικροϋπηρεσιών, όσον αφορά την απόδοση του συστήματος βάσει της CPU και της ταχύτητας του δικτύου, παρείχαν μια καθοδήγηση ανάλυσης συγκριτικής αξιολόγησης για τους σχεδιαστές συστημάτων και κατ' επέκταση στο σχεδιασμό της παρούσας διπλωματικής εργασίας.

Στη βιβλιογραφία παρατηρήθηκε ότι παρ' ότι υπάρχουν πολλαπλές αναφορές σχετικά με την εξισορρόπηση φορτίου, αλλά είναι ελάχιστες εκείνες που αναφέρονται στην οριζόντια κλιμακοθετησιμότητα ενός συστήματος με σταθερούς πόρους, στον τρόπο λειτουργίας της και στην απόδοσή της. Από τα ευρήματα της έρευνας των T. Ye et al. [14] πάνω στην αυτόματη κλιμακοθετησιμότητα, συνδυαστικά με την παρακολούθηση της χρήσης της CPU στην παρούσα διπλωματική εργασία, η CPU επιλέχθηκε με βάση την ανάγκη για υπέρμετρη χρήση πόρων από το σύστημα ώστε να μετρηθεί συγκριτικά η απόδοση διαφορετικών σεναρίων μέσω του συνολικού σφάλματος, της διεκπεραιωτική ικανότητας και της διακύμανσης της καθυστέρηση. Επιπρόσθετα, με γνώμονα την έρευνα των E. Casalicchio et al. [13], η παρούσα διπλωματική εργασία διερευνά την απόδοση ενός συστήματος με χρήση του Horizontal Pod Autoscaler, επομένως και τη χρήση μόνο των σχετικών μετρικών και αξιοποιώντας το σύνολο των διαθέσιμων πόρων.

Από τη μελέτη των M. Curiel και A. Pont [16], καθοδηγήθηκε η παρούσα μεταπτυχιακή εργασία όσον αφορά την κατανόηση των γεννητριών φορτίου, καθώς και η επιλογή της κατάλληλης γεννήτριας για την πειραματική διαδικασία που θα ακολουθούσε.

Η επιλογή των πειραματικών παραμέτρων που αφορά την κατανομή και τη δοκιμή φορτίου στην παρούσα διπλωματική εργασία έγινε βάσει της ερευνητικής εργασίας των D. Kallergis et al. [12].

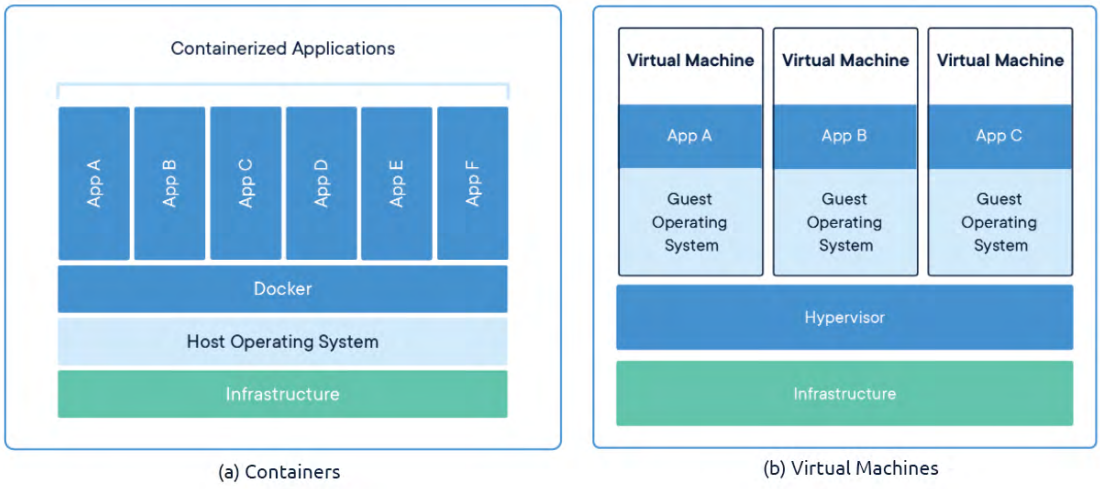
Κεφάλαιο 3 - Εργαλεία και Μέθοδοι

3.1 Περιέκτες

3.1.1 Εισαγωγή

Ένας περιέκτης (container) είναι μια πρότυπη μονάδα λογισμικού η οποία συσκευάζει τον κώδικα κατά τέτοιο τρόπο, επιτρέποντας στην εφαρμογή να εκτελείται ταχύτατα και με αξιοπιστία μεταξύ των υπολογιστικών περιβαλλόντων και εγγυάται ότι οι εφαρμογές λειτουργούν με τον ίδιο τρόπο οπουδήποτε. Είναι, δηλαδή, μια ιδανική μονάδα ανάπτυξης εφαρμογών που παρέχει εικονικοποίηση σε επίπεδο λειτουργικού συστήματος, καθώς μπορούν να απομονώσουν και να ελέγξουν πόρους για ένα σύνολο διαδικασιών [22], [23].

Οι περιέκτες φέρουν πολλές ομοιότητες με τα εικονικά μηχανήματα, αλλά έχουν ιδιότητες απομόνωσης (isolation properties) ώστε να μοιράζονται το λειτουργικό σύστημα μεταξύ των εφαρμογών, όπως φαίνεται στην Εικόνα 3.1. Παρόμοια με μια εικονική μηχανή, ένας περιέκτης έχει το δικό του σύστημα αρχείων, μερίδιο στην CPU, μνήμη, χώρο διεργασίας και πολλά άλλα [24].



Εικόνα 3.1: Στάδια Εξέλιξης της Εικονικοποίησης [22]

3.1.2 Πλεονεκτήματα των Περιεκτών

Ο περιέκτης δεν μιμείται το φυσικό υλικό, όπως κάνει μια εικονική μηχανή, είναι ελαφρύς καταλαμβάνοντας λιγότερο χώρο, ενώ πολλαπλοί περιέκτες μπορούν να διαχειριστούν παράλληλα περισσότερες εφαρμογές απαιτώντας λιγότερα εικονικά

μηχανήματα ή λειτουργικά συστήματα και έχοντας λιγότερο επίβαρο (overhead) [22], [25].

Οι περιέκτες έχουν γίνει δημοφιλής διότι παρέχουν πολλαπλά επιπλέον οφέλη [24]:

Πίνακας 3.1: Πλεονεκτήματα Περιεκτών

Πλεονεκτήματα Περιεκτών	**Σημειώσεις**
Δημιουργία και ανάπτυξη μιας ευέλικτης εφαρμογής	Αυξημένη ευκολία και αποτελεσματικότητα στη δημιουργία μιας εικόνας περιέκτη σε σύγκριση με τη χρήση μιας εικόνας εικονικής μηχανής
Συνεχής ανάπτυξη (development and deployment)	Παρέχουν αξιόπιστη και συχνή δημιουργία και ανάπτυξη εικόνων με γρήγορες και αποτελεσματικές επαναφορές (rollback)
Διαχωρισμός των μηχανικών ανάπτυξης και των μηχανικών λειτουργίας εκμετ άλλευσης (Dev and Ops)	Η δημιουργία των εικόνων των περιεκτών γίνεται σε χρόνο «build/release» αντί σε χρόνο «deployment» διαχωρίζοντας έτσι τις εφαρμογές από την υποδομή
Πληροφορίες και μετρήσεις σε όλα τα επίπεδα	Οι πληροφορίες και οι μετρήσεις που παρέχονται, δεν έγκεινται μόνο στο επίπεδο του λειτουργικού συστήματος, αλλά και στο επίπεδο της εφαρμογής (υγεία εφαρμογής κ.α.)
Περιβαλλοντική συνοχή	Αφορά την ανάπτυξη, τη δοκιμή και την παραγωγή. Λειτουργεί το ίδιο σε ένα φορητό υπολογιστή όπως και στο υπολογιστικό νέφος
Φορητότητα	Υπάρχει φορητότητα κατανομής στα υπολογιστικά νέφη και τα λειτουργικά συστήματα
Διαχείριση	Διαχείριση με επίκεντρο την εφαρμογή
Ανεξάρτητη διαχείριση	Οι εφαρμογές κατακερματίζονται σε κομμάτια. Είναι ανεξάρτητες μεταξύ τους και μπορούν να αναπτυχθούν καθώς και να διαχειριστούν δυναμικά – όχι σαν μια μονολιθική στοίβα που λειτουργεί σε ένα μεγάλο μηχάνημα μοναδικού σκοπού
Απομόνωση πόρων	Υπάρχει απομόνωση πόρων, κατά συνέπεια προβλέψιμη απόδοση της εφαρμογής
Αξιοποίηση πόρων	Γίνεται αξιοποίηση πόρων για υψηλή απόδοση και πυκνότητα

3.1.3 Λειτουργία Περιέκτη

Οι περιέκτες δημιουργούνται χρησιμοποιώντας «base images». Μια εικόνα (image) μπορεί είτε να περιλαμβάνει μόνο τις βασικές αρχές του λειτουργικού συστήματος, είτε να αποτελείται από μια εξελιγμένη προ-εγκατεστημένη στοίβα εφαρμογών (application stack) έτοιμη για εκκίνηση. Κατά τη δημιουργία εικόνων, κάθε ενέργεια που εκτελείται (δηλαδή εκτέλεση εντολών, όπως το apt-get install) σχηματίζει ένα νέο

επίπεδο πάνω από το προηγούμενο. Οι εντολές μπορούν να εκτελεστούν χειροκίνητα ή αυτόματα χρησιμοποιώντας το Dockerfiles [22], [25].

Μια εφαρμογή, για να εισαχθεί εντός ενός περιέκτη πρέπει να εκτελεστεί η διαδικασία της περιεκτοποίησης (application containerization), η συλλογή δηλαδή των στοιχείων που απαιτούνται για την εκτέλεση μιας εφαρμογής δημιουργώντας μια εικόνας της εφαρμογής, το σύνολο δηλαδή των πληροφοριών έχοντας τη μορφή ενός εκτελέσιμου αρχείου. Αποτελείται από στρώματα, στα οποία περιέχεται ο κώδικας, το αρχείο εκτέλεσης, οι βιβλιοθήκες, οι μεταβλητές περιβάλλοντος καθώς και τα αρχεία διαμόρφωσης [26].

3.2 Docker

3.2.1 Περιγραφή και Πλεονεκτήματα

Η εκτέλεση της εφαρμογής του εκάστοτε περιέκτη επιτυγχάνεται μέσω του μηχανισμού Docker [27]. Το Docker είναι μια πλατφόρμα λογισμικού ανοιχτού κώδικα που επιτρέπει στους προγραμματιστές τη δημιουργία και την ανάπτυξη εφαρμογών. Επιπρόσθετα, επιτρέπει τη διαχείριση εφαρμογών, με μεγάλη ταχύτητα, συσκευάζοντας αυτές εντός των περιεκτών, όπως φαίνεται στην Εικόνα 3.2 [28]. Μέσω του Docker, μπορούμε να χρησιμοποιούμε και να επεκτείνουμε γρήγορα εφαρμογές σε οποιοδήποτε περιβάλλον και γνωρίζοντας ότι ο κώδικας θα εκτελεστεί [29].

Το Docker ενίσχυσε τις δυνατότητες της περιεκτοποίησης Linux με τεχνολογίες που επιτρέπουν [28]:

Πίνακας 3.2: Πλεονεκτήματα Docker

Πλεονεκτήματα Docker	Σημειώσεις
Ελαφρύτεροι Περιέκτες με Λεπτομερείς Ενημερώσεις	Μόνο μία διαδικασία μπορεί να εκτελεστεί σε κάθε περιέκτη
Αυτόματη Δημιουργία περιέκτη	Μπορεί να δημιουργήσει αυτόματα ένα κοντέινερ με βάση τον πηγαίο κώδικα της εφαρμογής
Εκδόσεις Περιέκτη	Μπορεί να παρακολουθεί τις εκδόσεις μιας εικόνας ενός περιέκτη, να επιστρέφει σε προηγούμενες εκδόσεις και να εντοπίζει ποιος δημιούργησε μια έκδοση και πώς
Επαναχρησιμοποίηση	Οι υπάρχοντες περιέκτες μπορούν να χρησιμοποιηθούν

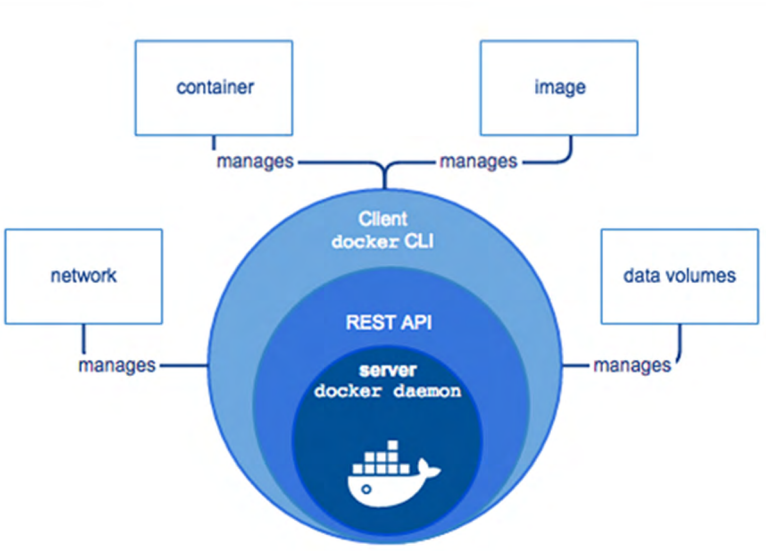
Περιέκτη	ως βασικές εικόνες - ουσιαστικά σαν πρότυπα για την κατασκευή νέων
Κοινόχρηστες Βιβλιοθήκες Περιεκτών	Παρέχει πρόσβαση σε ένα μητρώο ανοιχτού κώδικα που περιέχει χιλιάδες περιέκτες

3.2.2 Μηχανισμός Docker

Ο Μηχανισμός του Docker (Docker Engine) επιτρέπει να αναπτυχθούν, να συναρμολογηθούν, να σταλούν και τελικά να εκτελεστούν εφαρμογές. Για να επιτευχθούν οι παραπάνω ενέργειες χρησιμοποιεί τα ακόλουθα στοιχεία [30]:

Πίνακας 3.3: Μηχανισμός Docker

Πλεονεκτήματα Docker	Σημειώσεις
Docker Daemon	Διαδικασία παρασκηνίου (Εικόνα 3.2). Διαχειρίζεται εικόνες Docker, περιέκτες, δίκτυα και τόμους αποθήκευσης (storage volumes). Ο Docker daemon «ακούει» και επεξεργάζεται αιτήματα API του Docker
Docker Engine REST API	Docker API. Χρησιμοποιείται από εφαρμογές για αλληλεπίδραση με τον Docker Daemon. Είναι προσβάσιμο μέσω ενός http πελάτη
Docker CLI	Γραμμή εντολών για την αλληλεπίδραση με τον Docker Daemon



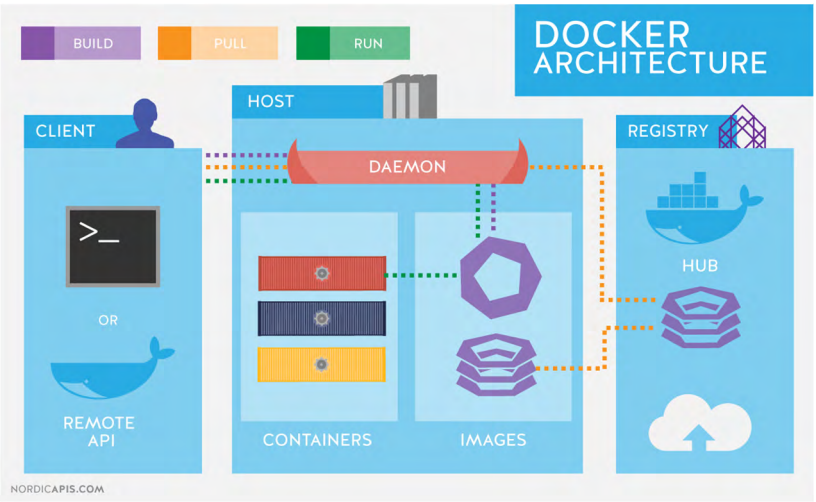
Εικόνα 3.2: Μηχανισμός του Docker [30]

3.2.3 Αρχιτεκτονική του Docker

Το Docker ακολουθεί το μοντέλο πελάτη-εξυπηρετητή (client – server). Περιλαμβάνει τα παρακάτω στοιχεία [30]:

Πίνακας 3.4: Στοιχεία Docker

Στοιχεία Docker	Σημειώσεις
Docker Client	Επιτρέπει την αλληλεπίδραση μεταξύ Docker - χρηστών. Υπάρχει δυνατότητα επικοινωνίας με πολλαπλούς Daemons. Παρέχει γραμμή εντολών (CLI) για τη δημιουργία, την εκτέλεση και την παύση εφαρμογών σε έναν Docker Daemon
Docker Host	Παρέχει ένα πλήρες περιβάλλον εκτέλεσης (execute and run) των εφαρμογών. Περιλαμβάνει τον Docker Daemon, Εικόνες, περιέκτες, δίκτυα και αποθήκευση
Αντικείμενα Docker (Docker objects)	Εικόνες (images): Δυαδικό πρότυπο για ανάγνωση προς την κατασκευή περιεκτών. Περιέχουν επίσης μετα-δεδομένα (metadata) των δυνατοτήτων και των αναγκών ενός περιέκτη Περιέκτες: Ως αναλύθηκαν Δικτύωση: Υπάρχουν δύο βασικοί τύποι διαθέσιμων δικτύων. Το προεπιλεγμένο δίκτυο Docker (none, bridge και host) και τα δίκτυα που καθορίζονται από τον χρήστη (δίκτυο γέφυρας, δίκτυο επικάλυψης και δίκτυο Macvlan) Αποθήκευση (storage): Τα δεδομένα μπορούν να αποθηκευτούν μέσα στο εγγράψιμο επίπεδο (layer) ενός περιέκτη ή σε έναν από τους μόνιμους (persistent) χώρου αποθήκευσης, όπως τους τόμους δεδομένων (data volumes)
Μητρώα Docker (Docker registries)	Υπηρεσία παροχής τοποθεσίας, διάφορης των Docker εξυπηρετητή και πελάτη. Περιέχει αποθετήρια που φιλοξενούν μία ή περισσότερες εικόνες Docker



Εικόνα 3.3: Αρχιτεκτονική του Docker [30]

Για την εκτέλεση εντολών εντός του μηχανισμού του Docker, εκτελούνται εντολές στη γραμμή εντολών, όπου με τη σειρά της επικοινωνεί μέσω του REST API με τον Docker Daemon. Ο Docker Daemon, εκτελεί αναζήτηση της εικόνας στα αρχεία του τοπικού

συστήματος ή εάν η εικόνα δεν βρεθεί εκεί, τότε γίνεται αναζήτηση στα Μητρώα του Docker. Έπειτα η εικόνα λαμβάνεται τοπικά εντός του συστήματος και εκτελείται, σύμφωνα με τις οδηγίες που υπάρχουν στο Dockerfile. Ο Docker Daemon αναλαμβάνει να εκτελέσει τις εργασίες οι οποίες περιλαμβάνουν τη δημιουργία, εκτέλεση και διανομής του περιέκτη.

3.3 Ενορχηστρωτές Περιεκτών

3.3.1 Εισαγωγή

Στην περίπτωση όπου η χρήση περιεκτών είναι περιορισμένη, η διαδικασία διαχείρισής τους γίνεται εύκολα μέσω της μηχανής (Docker engine). Όμως καθώς αυξάνονται οι εφαρμογές, δημιουργείται η ανάγκη για διαχειριστικά εργαλεία που θα διασφαλίσουν ότι δεν θα υπάρχει διακοπή λειτουργίας των εφαρμογών (downtime) και θα βοηθήσουν στην αυτοματοποίηση της συντήρησής τους. Επιπρόσθετα, θα μπορούν να αντικαθιστούν αυτόματα τους αποτυχημένους περιέκτες και να διαχειρίζονται τις ενημερώσεις τους.

Τα λογισμικά που σχεδιάστηκαν για τη διάθεση, τη διαχείριση αλλά και την κλιμακοθετησιμότητα, τον έλεγχο και τη συντήρηση των εφαρμογών δυναμικά που περιέχονται σε ένα περιέκτη, ονομάζονται «Ενορχηστρωτές» (Orchestrators) [31].

3.3.2 Χρήση και Πλεονεκτήματα Ενορχηστρωτών Περιεκτών

Ένας ενορχηστρωτής περιεκτών μπορεί να χρησιμοποιηθεί σε οποιοδήποτε περιβάλλον όπου χρησιμοποιούνται περιέκτες.

Παρέχει βοήθεια κατά την ανάπτυξη της ίδιας της εφαρμογής σε διάφορα μεταξύ τους υπολογιστικά περιβάλλοντα, χωρίς να απαιτείται επανασχεδιασμός. Οι μικροϋπηρεσίες εντός των περιεκτών διευκολύνουν την ενορχήστρωση υπηρεσιών, συμπεριλαμβανομένης της αποθήκευσης, της δικτύωσης και της ασφάλειας.

Παγκόσμια, υπάρχει μια στροφή πολλών οργανισμών προς την τεχνολογία των περιεκτών και της ενορχήστρωσής τους, με σκοπό την αυτοματοποίηση και διαχείριση πολύπλοκων εργασιών όπως [32]:

- Την παροχή και ανάπτυξη (provisioning and deployment).

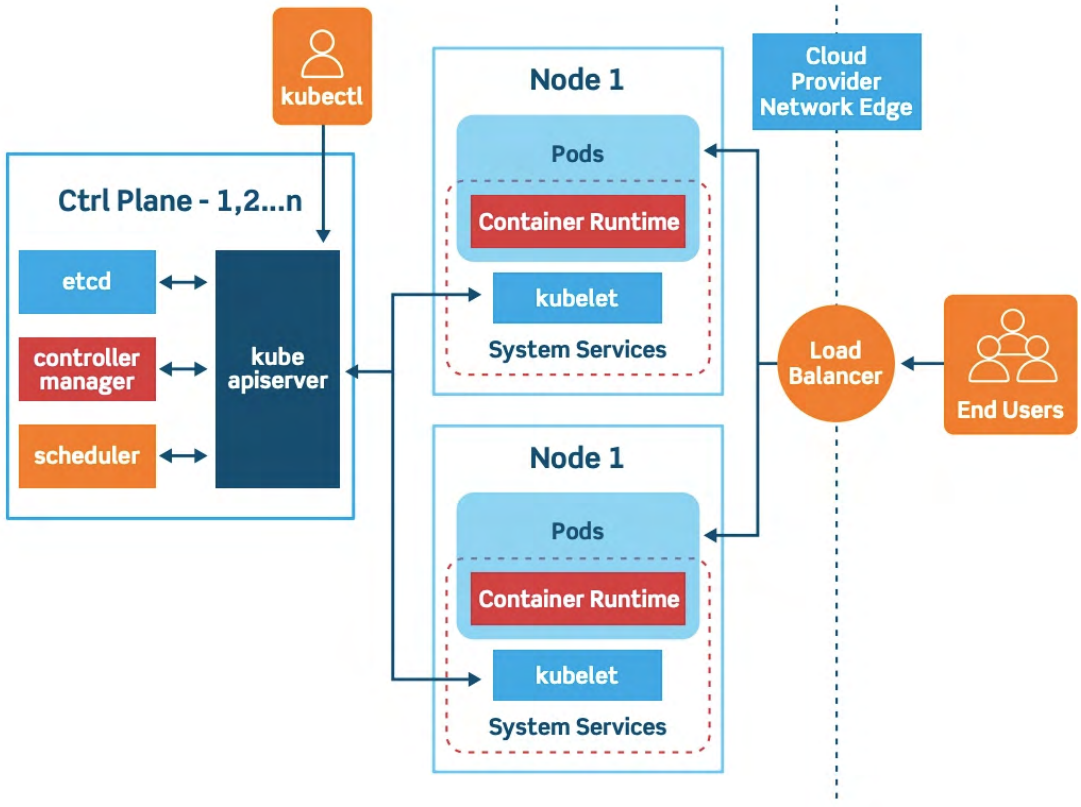
- Τη ρύθμιση παραμέτρων και τον προγραμματισμό.
- Την κατανομή των πόρων και τη διαθεσιμότητα περιεκτών.
- Την κλιμακοθετησιμότητα ή αφαίρεση περιεκτών με βάση την εξισορρόπηση του φόρτου εργασίας σε όλη την υποδομή.
- Την εξισορρόπηση φορτίου και δρομολόγηση κυκλοφορίας.
- Την παρακολούθηση της υγείας των περιεκτών και τη μέτρηση αποδοτικότητάς τους.
- Τη διαμόρφωση εφαρμογών με βάση τον περιέκτη στον οποίο θα εκτελούνται.
- Διατηρούν ασφαλείς τις αλληλεπιδράσεις μεταξύ των περιεκτών.
- Τη δυνατότητα για αυτόματη επιδιόρθωση της εφαρμογής.

3.3.3 Λογισμικά Διαχείρισης Περιεκτών

Τα λογισμικά ενορχήστρωσης παρέχουν ένα πλαίσιο (framework) για τη διαχείριση περιεκτών και την αρχιτεκτονική μικροϋπηρεσιών σε κλίμακα.
Υπάρχουν πολλά εργαλεία ενορχήστρωσης που μπορούν να χρησιμοποιηθούν για τη διαχείριση του κύκλου ζωής των περιεκτών. Οι πλέον δημοφιλείς επιλογές είναι το Kubernetes, το Docker Swarm και το Apache Mesos.

Kubernetes:

Το Kubernetes (γνωστό ως k8s ή «kube») είναι ένα λογισμικό/πλατφόρμα το οποίο λειτουργεί ως ενορχηστρωτής περιεκτών. Είναι ανοιχτού κώδικα (open source) το οποίο αρχικά δημιουργήθηκε από την Google και έγινε έπειτα δωρεά στο CNCF [5].
Με τον ενορχηστρωτή Kubernetes δύναται να δημιουργηθούν υπηρεσίες εφαρμογών σε πολλαπλούς περιέκτες, διαμορφώνοντάς τους σε συστοιχίες και κλιμακώνοντάς τους και παρακολουθώντας παράλληλα την υγεία τους [8].



Εικόνα 3.4: Αρχιτεκτονική του Kubernetes (πηγή: https://platform9.com)

Το Kubernetes αυτοματοποιεί τις διαδικασίες κατά την ανάπτυξη και την κλιμακοθετησιμότητα των εφαρμογών που εκτελούνται στους περιέκτες.

Έχει τη δυνατότητα να ομαδοποιήσει σε μια συστοιχία, διαφορετικές ομάδες από hosts, είτε φυσικών, είτε εικονικών μηχανών και παρέχει την πλατφόρμα για την εύκολη και αποτελεσματική διαχείριση αυτών των συστοιχιών όπως φαίνεται στην Εικόνα 3.4. Αυτές οι συστοιχίες μπορούν να περιλαμβάνουν ξενιστές (hosts) σε δημόσια, ιδιωτικά ή υβριδικά υπολογιστικά νέφη. Ο Ενορχηστρωτής Kubernetes καθίσταται κορυφαία επιλογή για τη φιλοξενία νεφοϋπολογιστικών εφαρμογών. Τέλος, ο ενορχηστρωτής Kubernetes βοηθά επίσης στην εξισορρόπηση φορτίου, επιτρέποντάς τη μετακίνηση εφαρμογών χωρίς να απαιτείται επανασχεδιασμός τους [32].

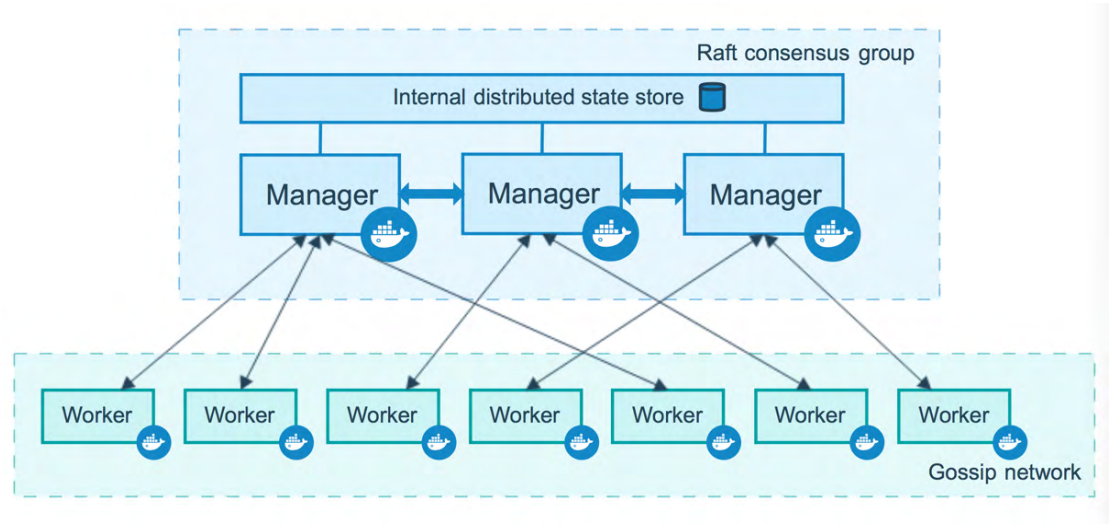
Το Kubernetes παρέχει [32], [8]:

- Ανακάλυψη υπηρεσιών (service discovery) και εξισορρόπηση φορτίου
- Ενορχήστρωση αποθήκευσης (storage orchestration)
- Αυτοματοποιημένα rollouts και rollbacks

- Αυτόματο bin packing
- Αυτοθεραπεία (self-healing)
- Διαχείριση secret και διαμόρφωσης (configuration)

Docker Swarm:

Το Docker Swarm είναι το εργαλείο του Docker για τη διαχείριση και ενορχήστρωση συστοιχιών, το οποίο κατασκευάστηκε και διαχειρίζεται η Docker Inc. Εισήχθη στο Docker Engine ως «Swarm Mode» με την ενημέρωση Docker 1.12, ως υποστήριξη στο Docker Engine, για την ενορχήστρωση πολλαπλών host και πολλαπλών περιεκτών όπως φαίνεται στην Εικόνα 3.5. Οι διαχειριστές και οι προγραμματιστές λογισμικού μπορούν να δημιουργήσουν και να διαχειριστούν ένα εικονικό σύστημα γνωστό ως Σμήνος (Swarm). Κάθε κόμβος του Docker Swarm είναι ένας Docker Daemon που αποτελείται από έναν ή περισσότερους κόμβους Docker [33].



Εικόνα 3.5: Αρχιτεκτονική του Docker Swarm [10]

Μέσω της απευθείας σύνδεσης με το Docker API, παρέχεται η πρόσβαση σε μια σειρά εργαλείων, όπως το Docker Compose. Οι αναπτύξεις των περιεκτών συνήθως χειρίζονται μέσω Docker Compose ή της γραμμής εντολών του Docker [30]. Κάθε περιέκτης εντός του Docker Swarm μπορεί να αναπτυχθεί και να είναι προσβάσιμος από κόμβους της ίδιας συστοιχίας [34].

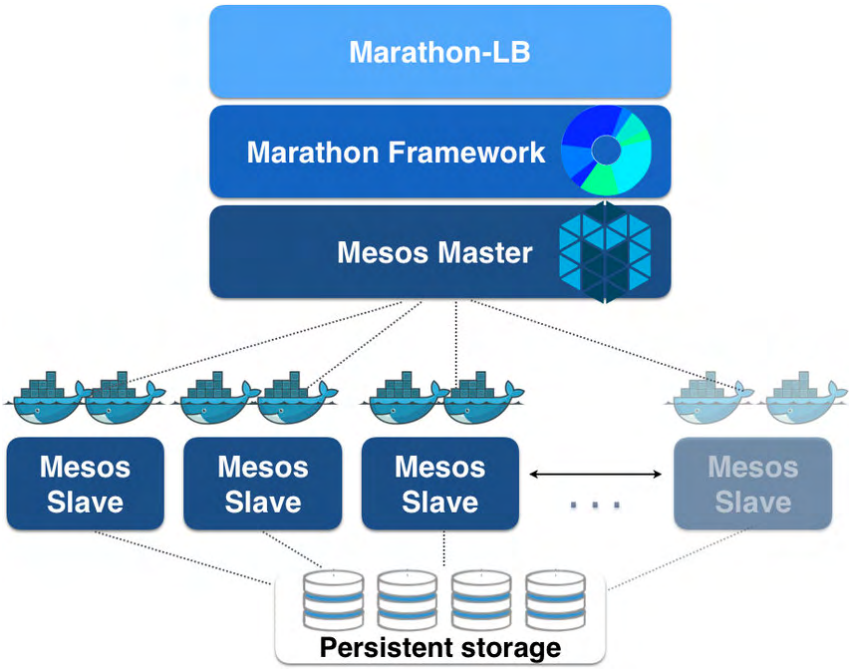
Το Docker Swarm παρέχει [34]:

- Αποκεντρωμένη πρόσβαση διευκολύνοντας τις ομάδες στην πρόσβαση και στη διαχείριση του περιβάλλοντός του.

- Υψηλή ασφάλεια καθώς κάθε επικοινωνία μεταξύ των διαχειριστικών κόμβων και των κόμβων πελάτη (client) εντός του Swarm είναι εξαιρετικά ασφαλή.
- Αυτόματη εξισορρόπηση φορτίου.
- Υψηλή κλιμακοθετησιμότητα αφού η εξισορρόπηση φορτίου μετατρέπει το περιβάλλον Swarm σε μια επεκτάσιμη υποδομή.
- Επαναφορά εργασιών.

Marathon:

Το Marathon είναι ένα πλαίσιο ανοιχτού κώδικα με σκοπό τη διαχείριση και την ενορχήστρωση περιεκτών το οποίο βασίζεται στο Apache Mesos και προορίζεται να εργαστεί πάνω σε εφαρμογές ή υπηρεσίες που θα εκτελούνται για πολύ μεγάλο χρονικό διάστημα.



Εικόνα 3.6: Αρχιτεκτονική του Marathon (πηγή: https://www.epj-conferences.org)

Το Marathon είναι μια -πλήρως βασισμένη σε REST- λύση και για τη διαχείρισή του μπορεί να χρησιμοποιηθεί μια διεπαφή διαδικτυακού χρήστη. Μπορεί να εκτελέσει πολλαπλούς χρονοπρογραμματιστές ταυτόχρονα, έτσι ώστε το σύστημα να μπορεί να συνεχίσει εάν ένας χρονοπρογραμματιστής κολλάει, με απώτερο σκοπό την αποφυγή κάποιου σφάλματος. Όπως και το Kubernetes, έτσι και το Marathon επιτρέπει την εκτέλεση τακτικών ελέγχων υγείας, ώστε να είμαστε ενήμεροι για την κατάσταση των εφαρμογών. Ένα άλλο πλεονέκτημα του Marathon είναι η

σταθερότητά του, διαθέτοντας μια ποικιλία χρήσιμων λειτουργιών, όπως ελέγχους υγείας, συνδρομές συμβάντων (event subscriptions) και μετρικές [34], [35].

3.4 Kubernetes

3.4.1 Αρχιτεκτονική Kubernetes

Προτού αναλυθεί το πώς λειτουργούν μαζί και μεταξύ τους τα διάφορα στοιχεία του Kubernetes, είναι σκόπιμο να αναφερθούν οι θεμελιώδεις αρχές σχεδιασμού του.

3.4.1.1 Αρχές Σχεδιασμού

Ο σχεδιασμός μιας Kubernetes συστοιχίας βασίζεται σε 3 αρχές. Μια συστοιχία Kubernetes πρέπει να είναι [32]:

- Ασφαλής, ακολουθώντας τις τελευταίες βέλτιστες πρακτικές ασφάλειας.
- Εύχρηστη, δηλαδή λειτουργική χρησιμοποιώντας μερικές απλές εντολές.
- Επεκτάσιμη, χωρίς να ευνοεί έναν πάροχο και να προσαρμόζεται από ένα αρχείο διαμόρφωσης.

3.4.1.2 Δομικά Στοιχεία και Λειτουργία

Το Kubernetes όταν γίνεται εγκατάσταση, έχει ως αποτέλεσμα τη δημιουργία μιας συστοιχίας. Η συστοιχία αυτή αποτελείται από δυο μέρη. Το πρώτο μέρος αποτελείται από τα μηχανήματα που επιτελούν τις εργασίες και εκτελούν τις περιεκτοποιημένες εφαρμογές. Τα μηχανήματα αυτά ονομάζονται «Kubernetes nodes» ή «worker nodes». Το δεύτερο μέρος της συστοιχίας αποτελεί το επίπεδο ελέγχου, γνωστό ως control plane [5].

Οι Kubernetes nodes αποτελούνται από φυσικά είτε εικονικά μηχανήματα. Φιλοξενούν τις υπηρεσίες που είναι απαραίτητες για να μπορούν να λειτουργήσουν τα pods. Τα pods αποτελούν τη μικρότερη οντότητα που μπορεί να εκτελέσει μια Kubernetes συστοιχία και αντιπροσωπεύει μια διεργασία της [36].

Το επίπεδο ελέγχου διατηρεί τη συστοιχία στην επιθυμητή κατάσταση, π.χ. ποιες εφαρμογές θα εκτελούνται και ποιες εικόνες οι εφαρμογές αυτές θα χρησιμοποιήσουν. Διαχειρίζεται τους worker nodes και τα pods στη συστοιχία και

λαμβάνει αποφάσεις σχετικά με αυτήν, καθώς είναι υπεύθυνο για την ανίχνευση και απόκριση στα γεγονότα της συστοιχίας (cluster events).

Λαμβάνει τις εντολές από έναν διαχειριστή (ή μια ομάδα διαχειριστών) και μεταβιβάζει αυτές τις οδηγίες στα υπολογιστικά μηχανήματα. [24], [37], [38]. Αυτή η ανάθεση εργασιών λειτουργεί με ένα πλήθος υπηρεσιών, ώστε να αποφασίζει αυτομάτως, ποιος κόμβος ταιριάζει καλύτερα για την εργασία. Κατόπιν κατανέμει πόρους και εκχωρεί τα pod στον εκάστοτε κόμβο για να εκπληρώσει το ζητούμενο έργο [37].

Ο έλεγχος των Περιεκτών συμβαίνει σε υψηλό επίπεδο (high level), προσφέροντας καλύτερο έλεγχο χωρίς την ανάγκη διαχείρισης κάθε περιέκτη ή κόμβου χωριστά [37].

Δομικά στοιχεία του επιπέδου ελέγχου [8], [38]:

Πίνακας 3.5: Δομικά Στοιχεία του Επιπέδου Ελέγχου

Δομικά Στοιχεία του Επιπέδου Ελέγχου	**Σημειώσεις**
Kube-APIserver	Μοναδικό σημείο πρόσβασης στη διαχείριση και τις υπηρεσίες της συστοιχίας. Μέσω του Kube-APIserver μπορεί να προσπελαστεί η βάση δεδομένων (etcd)
etcd	Κατανεμημένη βάση δεδομένων της συστοιχίας τύπου κλειδιού-τιμής (key-value). Διατηρείται σε πολλαπλά αντίγραφα και προσφέρει προστασία σε περιπτώσεις αποτυχίας και επανεκκίνησης του Kube-APIserver
Kube-Scheduler	Παρακολούθηση των νεοσύστατων pods και ανάθεση στις συστοιχίες προς εκτέλεση. Για κάθε pod, καθίσταται υπεύθυνος για την εύρεση του ιδανικότερου κόμβου ως προς την εκτέλεση αυτού του pod
Kube-controller-manager	Κάθε ελεγκτής (controller) αποτελεί μια διαφορετική διεργασία στη λειτουργία της συστοιχίας. Οι διεργασίες συγκεντρώνονται και μεταγλωττίζονται (compile) ως ένα ενιαίο αρχείο και εκτελούνται ως μια μοναδική διεργασία. Οι βασικότεροι ελεγκτές είναι ο ελεγκτής κόμβου (node controller), ο ελεγκτής αντιγραφής (replication controller), οι ελεγκτές λογαριασμού υπηρεσίας και αδειοδοτικού (service account & token controllers) και ο ελεγκτής ακροσημείων (endpoints controller)
Cloud-controller-manager	Σύνδεση της συστοιχίας στο API της νεφοϋπολογιστικής υπηρεσίας του παρόχου και διαχωρισμός των στοιχείων που αλληλοεπιδρούν με την νεφοϋπολογιστική πλατφόρμα από τα στοιχεία που αλληλοεπιδρούν μόνο με τη συστοιχία

Δομικά Στοιχεία του κόμβου Kubernetes [8], [38], [39]:

Πίνακας 3.6: Δομικά Στοιχεία του Κόμβου Kubernetes

Δομικά Στοιχεία του κόμβου Kubernetes	Σημειώσεις
Kube-Proxy	Διακομιστής μεσολάβησης δικτύου (network proxy). Εξασφαλίζει τη σύνδεση των χρηστών στις υπηρεσίες μέσω της διεύθυνσης IP και της θύρας (port) που αντιστοιχούν στα αντίστοιχα pod, διαμέσου του kube-APIserver
Container Runtime Interface (CRI)	Λογισμικό υπεύθυνο για την εκτέλεση των περιεκτών

Addons [38]:

Τα Πρόσθετα (Addons) χρησιμοποιούν πόρους του Kubernetes (daemonSet, deployment, κ.λπ.) για τη δημιουργία και εκτέλεση των χαρακτηριστικών (features) της συστοιχίας.

Πίνακας 3.7: Λίστα Addons

Λίστα Addons	Σημειώσεις
Web UI (Dashboard)	Διαδικτυακά βασισμένο γραφικό περιβάλλον γενικού σκοπού των συστοιχιών. Απεικόνιση των λειτουργιών της συστοιχίας, διαχείριση και αντιμετώπιση προβλημάτων των εφαρμογών
Container Resource Monitoring	Καταγράφει μετρήσεις δεδομένων των περιεκτών με βάση το χρόνο (time-series) σε μια κεντρική βάση δεδομένων. Παρέχει γραφικό περιβάλλον για ευκολότερη ανάγνωση των δεδομένων
Cluster-level Logging	Μηχανισμός καταγραφής σε επίπεδο συστοιχίας. Είναι υπεύθυνος για την αποθήκευση αρχείων καταγραφής (logs) των περιεκτών σε ένα κεντρικό χώρο αποθήκευσης

Επικοινωνία Επιπέδου Ελέγχου – Κόμβου Kubernetes [40]:

Σκοπός είναι να επιτραπεί στον χρήστη να προσαρμόσει την εγκατάστασή του, έτσι ώστε να ενδυναμώσει τις παραμέτρους ασφαλείας του δικτύου με αποτέλεσμα η συστοιχία να μπορεί να εκτελεστεί σε ένα αναξιόπιστο δίκτυο.

Πίνακας 3.8: Επικοινωνία Επιπέδου Ελέγχου – Κόμβου Kubernetes

Επικοινωνία Επιπέδου Ελέγχου – Κόμβου Kubernetes	Σημειώσεις
Επικοινωνία Κόμβου	Ο APIserver ανιχνεύει απομακρυσμένες συνδέσεις σε

Kubernetes - Επιπέδου Ελέγχου	μια θύρα HTTPS κάνοντας έλεγχο ταυτοποίησης του πελάτη (client). Μία ή περισσότερες μορφές εξουσιοδότησης θα πρέπει να είναι ενεργοποιημένες, ειδικά εάν επιτρέπονται ανώνυμα αιτήματα. Οι κόμβοι πρέπει να έχουν δημόσιο ριζικό πιστοποιητικό για τη συστοιχία, ώστε να συνδεθούν ασφαλώς με τον APIserver. Τα pods για να συνδεθούν ασφαλώς στον APIserver, χρησιμοποιούν λογαριασμό υπηρεσίας (service account), ώστε το Kubernetes να κάνει αυτομάτως εισαγωγή του δημόσιου ριζικού πιστοποιητικού και ενός έγκυρου διακριτικού φορέα (bearer token) κατά τη δημιουργία του pod. Το Kubernetes έχει μια εικονική διεύθυνση IP και μέσω του kube-proxy πραγματοποιείται ανακατεύθυνση στο τελικό σημείο HTTPS (HTTPS end point) του APIserver. Ο τρόπος διασύνδεσης των κόμβων και των ομάδων των pods προς το επίπεδο ελέγχου, είναι προεπιλεγμένα ασφαλισμένος και δύναται να εκτελεστεί σε αναξιόπιστα ή/και δημόσια δίκτυα
Επικοινωνία Επιπέδου Ελέγχου στον Κόμβο	APIserver – kubelet: Εκτελείται σε όλους τους κόμβους της συστοιχίας. Χρησιμοποιείται για τη λήψη αρχείων καταγραφής (logs) των pods, την τοποθέτησή τους μέσω του kubectl στα ενεργά pod και την παροχή της «port-forwarding» λειτουργίας του kubelet. Τερματίζονται στο ακροσημείο HTTPS του kubelet. APIserver - κόμβος/pod/υπηρεσία: HTTP συνδέσεις από προεπιλογή. Δεν είναι πιστοποιημένες/ κρυπτογραφημένες. Σήραγγες SSH: προστασία της επικοινωνίας μεταξύ του επιπέδου ελέγχου και των κόμβων. Ο APIserver δημιουργεί σήραγγα SSH για τον εκάστοτε κόμβο της συστοιχίας διοχετεύοντας την κίνηση μέσα από αυτή και διασφαλίζοντας ότι η κίνηση δεν εκτίθεται εκτός του δικτύου των κόμβων. Υπηρεσία «Konnectivity»: παρέχει εξουσιοδότηση (proxy) για την προστασία της επικοινωνίας μεταξύ του επιπέδου ελέγχου και των κόμβων

3.4.2 Αυτόματη Κλιμακοθετησιμότητα

3.4.2.1 Εισαγωγή

Ένα από τα σημαντικότερα πλεονεκτήματα του Kubernetes είναι η δυνατότητα υψηλού βαθμού αυτοματοποίησης και συγκεκριμένα, η δυνατότητα αυτόματης κλιμακοθετησιμότητας (autoscaling) των εργασιών (workloads) και των περιβαλλόντων στα οποία εκτελούνται. Θέτοντας ορισμένες παραμέτρους που

καθοδηγούν την κλιμακοθετησιμότητα, επιτρέπει στο Kubernetes να εκτελεί την δουλειά αυτόματα χωρίς συνεχόμενη ανθρώπινη επίβλεψη. Χωρίς τον αυτοματισμό αυτό, θα έπρεπε να εκτελούνται αρκετά αντίγραφα εργασίας (replicas) για να αντιμετωπιστεί η ζήτηση σε περίοδο αιχμής. Επιπλέον θα έπρεπε να παρακολουθείται συνεχώς η κυμαινόμενη ζήτηση στις υπηρεσίες εφαρμογών (application services) ώστε να αυξάνεται ή να μειώνεται ανάλογα όπως και χειροκίνητα ο αριθμός των αντιγράφων εργασίας [5]. Επιπλέον, τα εργαλεία αυτόματης κλιμακοθετησιμότητας για μεγάλα νεφοϋπολογιστικά συστήματα αξιοποιούν τα μέγιστα όσον αφορά τους διαθέσιμους πόρους με αποτελεσματικότητα, λειτουργώντας ως κριτήριο για την επιλογή του Kubernetes ως τον ενορχηστρωτή της παρούσας διπλωματικής εργασίας [8].

3.4.2.2 Τύποι Αυτόματης Κλιμακοθετησιμότητας

Οι τύποι αυτόματης κλιμακοθετησιμότητα ς που είναι διαθέσιμοι μέσω των API είναι [5], [38]:

Horizontal Pod Autoscaling (HPA) - όταν λαμβάνουμε μια απότομη αύξηση ή πτώση της ζήτησης για μια εργασία, το Kubernetes μπορεί αυτόματα να αυξομειώσει τον αριθμό των αντιγράφων των pods που εξυπηρετούν την εκάστοτε εργασία.

Vertical Pod Autoscaling (VPA) – Καθώς είναι πολύ δύσκολο να επιτευχθεί ο ακριβής προσδιορισμός των υπολογιστικών πόρων που απαιτείται για την αντιμετώπιση ενός κυμαινόμενου φόρτου εργασίας, το Kubernetes παρακολουθεί την απόδοση του φόρτου εργασίας με την πάροδο του χρόνου και προτείνει τις βέλτιστες απαιτήσεις πόρων για την εργασία ή τις προσαρμόζει αυτόματα εάν του επιτραπεί.

Cluster Autoscaling (CA) – Στην ελληνική γλώσσα θα μπορούσε να ονομαστεί «Η Αυτόματη Κλιμακοθετησιμότητα Συστοιχίας». Οι εργασίες μπορούν να εκτελεστούν μόνο εάν υπάρχει επαρκής χωρητικότητα στους κόμβους που ανήκουν σε μία συστοιχία. Αντίθετα, εάν υπάρχει ένας μεγάλος αριθμός κόμβων σε αχρησία, ουσιαστικά χρεωνόμαστε την πλεονάζουσα υπολογιστική δύναμη. Το Kubernetes έχει τη δυνατότητα να αυξάνει ή να μειώνει δυναμικά τον αριθμό των κόμβων που απαρτίζουν τη συστοιχία, ώστε να αντικατοπτρίζει τη υπάρχουσα ζήτηση.

3.4.2.3 Pod

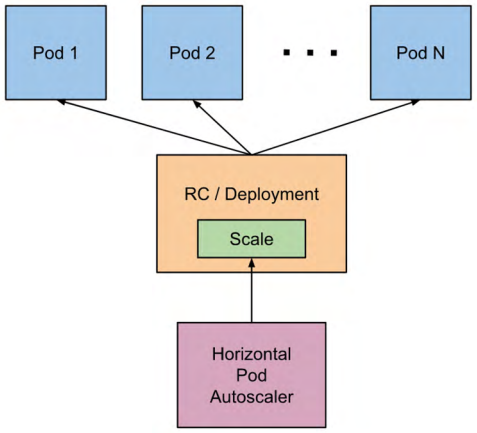
Πριν προχωρήσουμε στην επεξήγηση του Horizontal Pod Autoscaler και την ανάλυση της λειτουργίας του, θα πρέπει να κατανοήσουμε τον τρόπο λειτουργίας των pods.
Ένα pod αποτελείται από ένα ή περισσότερους περιέκτες στενά συνδεδεμένους μεταξύ τους, με κοινό αποθηκευτικό χώρο, κοινούς πόρους δικτύου και με προδιαγραφές για τον τρόπο με τον οποίο λειτουργούν οι περιέκτες μέσα σε αυτό. Επιπλέον, μπορεί να περιέχει περιέκτες οι οποίοι να εκτελούνται αυτόματα μαζί με την εκκίνηση ενός pod, πριν την εκτέλεση των περιεκτών που φέρουν τις εφαρμογές. Επομένως, ένα pod μοιάζει όσο αφορά τον τρόπο λειτουργίας του με μια ομάδα Docker περιεκτών, έχοντας κοινόχρηστα namespaces και κοινόχρηστους τόμους αρχείων συστήματος [36], [41].
Τα pods έχουν έναν καθορισμένο κύκλο ζωής. Όταν ένα pod εκτελείται σε μια συστοιχία και παρουσιαστεί ένα κρίσιμο σφάλμα στον κόμβο όπου εκτελείται το pod αυτό, τότε όλα τα pods σε αυτόν τον κόμβο αποτυγχάνουν. Το Kubernetes αντιμετωπίζει αυτό το επίπεδο αποτυχίας ως τελικό. Θα χρειαστεί δηλαδή να δημιουργηθεί ένα νέο pod, ακόμα κι αν επανέλθει ο κόμβος σε υγιή κατάσταση αργότερα [42], [43].

Σε κάθε pod εκχωρείται μια μοναδική διεύθυνση IP. Κάθε περιέκτης εντός ενός pod μοιράζεται τον ονοματοχώρο δικτύου (network namespace), συμπεριλαμβανομένων της διεύθυνσης IP και των θυρών (ports) του δικτύου. Όταν οι περιέκτες εντός του pod έρχονται σε επικοινωνία με εξωτερικές οντότητες, οφείλουν να συντονίζονται όσον αφορά τη χρήση των κοινών πόρων δικτύωσης (για παράδειγμα τις θύρες). Οι εντός pod περιέκτες διαθέτουν κοινή διεύθυνση IP και χώρο στη θύρα (port space). Μπορούν να βρουν ο ένας τον άλλο μέσω του localhost. Οι περιέκτες που θα χρειαστεί να αλληλοεπιδράσουν με περιέκτες οι οποίοι εκτελούνται σε κάποιο άλλο pod εντός της συστοιχίας, δύναται να επικοινωνήσουν χρησιμοποιώντας τη δικτύωση IP [36].

3.4.2.4 Horizontal Pod Autoscaler

Η εργασία που αναλαμβάνει να διεκπεραιώσει ο Horizontal Pod Autoscaler, όπως προαναφέρθηκε, είναι να κλιμακώνει αυτόματα τον αριθμό των pods σε ένα ελεγκτή

αντιγραφής (replication controller) ή σε μια ανάπτυξη. Η κλιμακοθετησιμότητα αυτή γίνεται κυρίως με βάση την παρατηρούμενη χρήση της CPU ή της μνήμης RAM ή τον συνδυασμό και των δύο στην τελευταία δοκιμαστική έκδοση του HPA. Υλοποιείται ως ένα Kubernetes API πόρων και διαχειριστή. Οι πόροι καθορίζουν τη συμπεριφορά του διαχειριστή, ο οποίος προσαρμόζει ανά περιόδους τον αριθμό των replicas (Εικόνα 3.7) σε έναν ελεγκτή αντιγραφής ή σε μια ανάπτυξη για να αντιστοιχεί την παρατηρούμενη μέση χρήση της CPU με αυτήν που είναι προκαθορισμένη από τον χρήστη [21].



Εικόνα 3.7: Λειτουργία Horizontal Pod Autoscaler [21]

Ο Horizontal Pod Autoscaler υλοποιείται ως «βρόχος ελέγχου» (control loop), με μια «σημαία» (horizontal-pod-autoscaler-sync-period flag) η οποία ανήκει στον ελεγκτή διαχειριστή (controller manager) και ελέγχει με περιόδους έχοντας ως προεπιλεγμένη τιμή τα 15 δευτερόλεπτα.

Κατά τη διάρκεια κάθε περιόδου, ο ελεγκτής διαχειριστής θέτει ερωτήματα για τη χρήση των πόρων σε σχέση με τις μετρικές που καθορίζονται από τον Horizontal Pod Autoscaler. Ο ελεγκτής διαχειριστής λαμβάνει τις μετρήσεις είτε από το «API μετρικών πόρων» (resource metrics API) για μετρήσεις πόρων ανά pod, είτε από το «API προσαρμοσμένων μετρήσεων» (custom metrics API), για όλες τις άλλες μετρήσεις.

Για μετρικές πόρων ανά pod (per-pod resource metrics), όπως είναι η CPU, ο ελεγκτής ανακτά τις μετρήσεις από το API μετρικών πόρων από κάθε pod που έχει ανατεθεί στον Horizontal Pod Autoscaler. Εάν έχει οριστεί μια «τιμή αξιοποίησης» (target utilization value), ο ελεγκτής διαχειριστής την υπολογίζει ως ένα ποσοστό του «ισοδύναμου αιτήματος πόρων» (equivalent resource request) των περιεκτών σε

κάθε pod [9]. Με άλλα λόγια, όταν το κατώφλι της τιμής αξιοποίησης που ορίστηκε ξεπεραστεί από ένα ή περισσότερα pods, τότε δημιουργούνται καινούρια pods ώστε να ρίξουν την τιμή αυτή στα θεμιτά επίπεδα.

Για προσαρμοσμένες μετρικές ανά pod (per-pod custom metrics), ο ελεγκτής λειτουργεί παρόμοια με τις μετρικές πόρων ανά pod, με ειδοποιό διαφορά το ότι λειτουργεί με μη επεξεργασμένες τιμές (raw values) και όχι με τιμές αξιοποίησης utilization values).

Για τις μετρήσεις αντικειμένων (objects) και τις εξωτερικές μετρήσεις, λαμβάνεται μια μεμονωμένη μέτρηση, η οποία περιγράφει το εν λόγω αντικείμενο. Αυτή η μέτρηση συγκρίνεται με την τιμή αξιοποίησης, για να παραχθεί μια αναλογία όπως και παραπάνω. Στην έκδοση autoscaling/v2beta2 API, αυτή η τιμή μπορεί προαιρετικά να διαιρεθεί με τον αριθμό των pods πριν γίνει η σύγκριση [21].

Η ανάλυση της αρχιτεκτονικής, καθώς και τα μεταδεδομένα που παρέχονται διαμέσου του Horizontal Pod Autoscaler και τα οποία περιγράφουν τη συμπεριφορά των πόρων εντός μιας συστοιχίας, στην έρευνα των Q. Lei et al. [11] συντέλεσαν στην κατανόηση και εξέλιξη της παρούσας διπλωματικής εργασίας.

3.4.3 Παρακολούθηση

3.4.3.1 Παρακολούθηση Συστοιχίας

Σκοπός της παρακολούθησης της συστοιχίας (cluster monitoring) είναι η παρακολούθηση της υγείας του συνόλου της συστοιχίας. Μερικές από τις μετρικές παρακολούθησης συστοιχίας είναι [44]:

- Χρήση πόρων των κόμβων (node resource utilization)
- Αριθμός των κόμβων (number of nodes)
- Τρέχοντα pods (running pods)

3.4.3.2 Παρακολούθηση των Pods

Η παρακολούθηση των pods (pod monitoring) μπορεί να χωριστεί σε τρεις κατηγορίες [6][44]:

Πίνακας 3.9: Παρακολούθηση των Pods

Παρακολούθηση των Pods	Σημειώσεις
Kubernetes Μετρικές (Kubernetes Metrics)	Παρακολούθηση χειρισμού μεμονωμένων pods και των αναπτύξεών τους από τον ενορχηστρωτή. Λαμβάνονται πληροφορίες όπως: ο αριθμός των υποστάσεων (instances) που έχει ένα pod μια χρονική στιγμή και πόσα ήταν αναμενόμενο να υπάρχουν (εάν ο αριθμός είναι χαμηλός, είναι πιθανό να χρησιμοποιούνται όλοι οι πόροι της συστοιχίας), η κατά την ανάπτυξη πρόοδος (πόσες υποστάσεις άλλαξαν σε νεότερη έκδοση), ο έλεγχος υγείας καθώς και ορισμένα δεδομένα δικτύου
Μετρικές Περιεκτών (Container Metrics)	Λαμβάνονται κυρίως μέσω του cAdvisor, ο οποίος ρωτάει κάθε κόμβο για τους περιέκτες που εκτελούνται. Οι μετρήσεις που λαμβάνονται είναι η CPU και η χρήση μνήμης σε σύγκριση με το μέγιστο επιτρεπόμενο όριό τους
Μετρικές Εφαρμογών (Application Metrics)	Αναπτύσσονται από την ίδια την εφαρμογή και σχετίζονται με τους επιχειρηματικούς κανόνες στους οποίους απευθύνεται η εφαρμογή

Χρησιμοποιώντας τις μετρικές αυτές, μπορούμε να παρακολουθούμε πώς ο ενορχηστρωτής χειρίζεται ένα συγκεκριμένο pod και την ανάπτυξή του από τον ενορχηστρωτή. Μπορούν να παρακολουθούνται οι ακόλουθες πληροφορίες:

- Ο αριθμός των instances που έχει ένα pod αυτήν τη στιγμή και πόσα ήταν αναμενόμενα (εάν ο αριθμός είναι χαμηλός, η συστοιχία μπορεί να είναι χωρίς διαθέσιμους πόρους).
- Την κατάσταση της ανάπτυξης που βρίσκεται σε εξέλιξη.
- Ο έλεγχος υγείας
- Ορισμένα δεδομένα δικτύου διαθέσιμα μέσω υπηρεσιών δικτύου.

Το Kubernetes, έχει πολλούς τρόπους για την παρακολούθηση μιας εφαρμογής. Είναι εφικτό να χρησιμοποιηθούν για τη συλλογή στατιστικών [6]:

Πίνακας 3.10: Παρακολούθηση Εφαρμογών

Παρακολούθηση Εφαρμογών	Σημειώσεις
Resource metrics pipeline	Παρέχει περιορισμένο σύνολο μετρήσεων που σχετίζονται με τη συστοιχία
Full metrics pipeline	Παρέχει πρόσβαση σε περισσότερες μετρήσεις σε σύγκριση με το Resource metrics pipeline. Το Kubernetes μπορεί να ανταποκριθεί σε αυτές τις

	μετρήσεις κλιμακώνοντας αυτόματα ή προσαρμόζοντας τη συστοιχία βάσει της τρέχουσας κατάστασής του

3.5 NGINX

Το NGINX είναι λογισμικό ανοιχτού κώδικα, δημιουργία του Igor Sysoev, για υπηρεσίες διαδικτύου, εξισορρόπηση φορτίου (load balancing), reverse proxy, caching και πολλά άλλα. Πέραν των δυνατοτήτων του ως HTTP διακομιστής, το NGINX δύναται να λειτουργήσει ως πληρεξούσιος εξυπηρετητής (proxy server) για email (IMAP, POP3 και SMTP) και για εξισορρόπηση φορτίου. Τέλος, τοποθετείται συχνά μεταξύ των πελατών και ενός δεύτερου διακομιστή, για να χρησιμεύσει ως τερματιστής SSL/TLS ή επιταχυντής Ιστού [45].

3.6 Apache JMeter

Η εφαρμογή Apache JMeter είναι ένα λογισμικό ανοιχτού κώδικα και λειτουργεί ως ένα γεννήτρια κίνησης (traffic generator). Είναι λογισμικό εξ ολοκλήρου γραμμένο σε προγραμματιστική γλώσσα Java. Το Apache JMeter μπορεί επίσης να προσομοιώσει μεγάλα φορτία δημιουργώντας πολλαπλούς ταυτόχρονους ή με κάποια μαθηματική κατανομή, εικονικούς χρήστες, ώστε να αναλύσει τη συνολική απόδοση κάτω από διαφορετικούς τύπους φορτίου [20]. Τα χαρακτηριστικά του Apache JMeter περιλαμβάνουν [20]:

Πίνακας 3.11: Χαρακτηριστικά του Apache JMeter

Επικοινωνία Επιπέδου Ελέγχου – Κόμβου Kubernetes	**Σημειώσεις**
Φόρτωση και Δοκιμή Απόδοσης	Δυνατότητα φόρτωσης και δοκιμής απόδοσης πολλών διαφορετικών τύπων πρωτοκόλλων
Πλήρης Δυνατότητα Δοκιμής IDE	Επιτρέπει γρήγορη εγγραφή ενός «Test Plan» (από προγράμματα περιήγησης ή εγγενείς εφαρμογές) για εντοπισμό σφαλμάτων
Γραμμή Εντολών (CLI)	Λειτουργία γραμμής εντολών για δοκιμές φορτίου από οποιοδήποτε λειτουργικό σύστημα συμβατό με Java
Αναφορά HTML	Πλήρης και έτοιμη για παρουσίαση δυναμική αναφορά HTML

Εξαγωγή Δεδομένων	Δυνατότητα εξαγωγής δεδομένων από τις πιο δημοφιλείς εκδόσεις απόκρισης, HTML, JSON, XML ή οποιαδήποτε μορφή κειμένου
Φορητότητα	Πλήρης φορητότητα
Πλαίσιο Πολλαπλών Νημάτων (Multi-threading Framework)	Επιτρέπει την ταυτόχρονη δειγματοληψία από πολλά νήματα και την ταυτόχρονη δειγματοληψία διαφορετικών συναρτήσεων από ξεχωριστές ομάδες νημάτων
Αποθήκευση και Ανάλυση Αποτελεσμάτων Εκτός Σύνδεσης	Αποθήκευση και ανάλυση/αναπαραγωγή εκτός σύνδεσης των αποτελεσμάτων των δοκιμών
Ευρεία Δυνατότητα Κλιμακοθετησιμότητας	Περιλαμβάνει προεγκατεστημένα δειγματολόγια (samplers). Παρέχει χρονοδιακόπτες με δυνατότητα σύνδεσης και μπορούν να επιλεγούν διάφορες μορφές κατανομής φορτίου. Τα πρόσθετα (plug-ins), τα οποία αφορούν ανάλυσης δεδομένων και οπτικοποίησης, επιτρέπουν μεγάλη κλιμακοθετησιμότητα καθώς και εξατομίκευση

Η παρακολούθηση της απόδοσης φορτίου από το JMeter, γίνεται κατά κύριο λόγο από τους ακροατές (listeners). Ο ακροατής είναι ένα στοιχείο που δείχνει τα αποτελέσματα των απεσταλμένων δειγμάτων, τα οποία μπορούν να εμφανιστούν σε δέντρο, πίνακες, γραφήματα ή να αποθηκευτούν σε ένα αρχείο καταγραφής. Η ποικιλία σε ακροατές επιτρέπει στους μηχανικούς απόδοσης, να παρακολουθούν τα αιτήματα που αποστέλλονται καθώς και να αναλύουν τις απαντήσεις που λαμβάνονται από το υπό δοκιμή σύστημα. Οι ακροατές συγκεντρώνουν πληροφορίες σχετικά με τον χρόνο και τη χωρητικότητα των αιτημάτων που συλλέγονται από τα αιτήματα και τις απαντήσεις ή διαχειρίζονται στατιστικές πληροφορίες όπως η διανομή και το ποσοστό σφάλματος. Σχεδόν όλοι οι ακροατές έχουν τη δυνατότητα να καταγράφουν τα αποτελέσματα σε αρχεία (όπως csv), επιτρέποντας στους μηχανικούς απόδοσης να τα μετατρέψουν στη μορφή που επιθυμούν ώστε να τα αναλύσουν [46].

Όπως προέκυψε από τη μελέτη της βιβλιογραφίας [16], το εργαλείο που επιλέγω ως γεννήτρια κίνησης (traffic generator) είναι το JMeter, λόγω: **(α)** ευκολίας χρήσης, **(β)** παραγωγής αιτημάτων με ρεαλιστικό τρόπο, **(γ)** εφαρμογής σε ανοικτά συστήματα, **(δ)** υποστήριξης κατανεμημένης παραγωγής φορτίου και **(ε)** δυνατότητας

παραγωγής αιτημάτων χωρίς εξάρτηση από προϋπάρχοντα (request-based *αντί* session-based).

3.7 Νεφοϋπολογιστική Υπηρεσία ~okeanos

Το ~okeanos [47] είναι η νεφοϋπολογιστική υπηρεσία τύπου IaaS, η οποία προσφέρεται από το ΕΔΕΤ για την Ελληνική Ερευνητική και Ακαδημαϊκή Κοινότητα. Αποτελείται από τις υπηρεσίες cyclades και pithos+.

Η υπηρεσία cyclades δίνει τη δυνατότητα στους χρήστες να δημιουργούν εικονικές μηχανές, συνδεδεμένες στο διαδίκτυο και διαθέτοντας τις επιθυμητές ανά τον χρήστη τεχνικές προδιαγραφές. Η υπηρεσία pithos+ παρέχει αποθήκευση στο υπολογιστικό νέφος με άμεση πρόσβαση του χρήστη στα δεδομένα από παντού. Επιπρόσθετα, λειτουργεί ως χώρος αποθήκευσης για τις ανάγκες που μπορεί να έχουν οι εικονικές μηχανές.

3.8 Νεφοϋπολογιστική Υπηρεσία Hetzner

Η Hetzner Online GmbH [48] είναι εταιρία παροχής νεφοϋπολογιστικών πόρων, εικονικών μηχανών και διαχείρισης κέντρων δεδομένων. Από το 1997, η εταιρεία παρέχει σε ιδιώτες προϊόντα φιλοξενίας υψηλής απόδοσης, καθώς και την απαραίτητη υποδομή για την αποτελεσματική λειτουργία τους. Διαθέτει πολλά πάρκα κέντρων δεδομένων στη Γερμανία και στη Φινλανδία.

Κεφάλαιο 4 - Πειραματική Διάταξη

4.1 Ανάλυση και Σχεδίαση Πειραματικής Διάταξης

4.1.1 Απαιτήσεις

Κατά τον σχεδιασμό του πειράματος, δημιουργήθηκε αρχικά δοκιμαστική εγκατάσταση του Docker και του Kubernetes στη νεφοϋπολογιστική υπηρεσία ~okeanos, ώστε να υπολογιστούν οι πόροι οι οποίοι χρειάζονταν για να δημιουργηθεί η πειραματική διάταξη και να έρθει εις πέρας το πείραμα.

Οι πόροι που ήταν διαθέσιμοι προς εκμετάλλευση στη νεφοϋπολογιστική υπηρεσία ~okeanos φαίνονται στον Πίνακα 4.1.

Πίνακας 4.1: Διαθέσιμοι Πόροι και Απαιτήσεις Διάταξης

	~okeanos	**Kubernetes Cluster**	**Kybernetes Master**	**Kybernetes Node**
CPU (Cores)	8	4	3	1
RAM (GB)	8	8	4	4
Disk Space (GB)	40	40	20	20

Οι παραπάνω πόροι, χωρίστηκαν με γνώμονα τις προδιαγραφές που ορίζουν το Docker και το Kubernetes ώστε να έχουμε δυο (2) χωριστές εικονικές μηχανές. Επιπλέον, αυτές οι προδιαγραφές ήταν απαραίτητες για την μετέπειτα επιλογή των εικονικών μηχανών από την εταιρία Hetzner, όπως θα αναλυθεί στη «Σχεδίαση Πειραματικής Διάταξης» που ακολουθεί.

Οι ελάχιστες προδιαγραφές για μια Kubernetes master node – worker node διάταξη η οποία είναι εγκατεστημένη να τρέχει πάνω σε Docker version 20.10.2, φαίνονται στον πίνακα 4.1.

Οι παραπάνω προδιαγραφές αφορούν όλη τη συστοιχία. Από το σύνολο των πόρων, ο Kubernetes master πρέπει να καταλαμβάνει το ελάχιστο 3 CPU cores, 4 GB RAM και 15 GB χώρο στο σκληρό δίσκο. Αντίστοιχα, ο worker node χρειάζεται το ελάχιστο 1 CPU cores, 4 GB RAM και 15 GB χώρο στο σκληρό δίσκο.

Στην αρχική, δοκιμαστική διάταξη για τον καθορισμό των πόρων ο διαχωρισμός των πόρων φαίνεται στον πίνακα 4.1.

Η διάταξη αυτή μετά από πολλαπλές δοκιμές δούλεψε ικανοποιητικά.

Με το πέρας των δοκιμών πάνω στην προαναφερθείσα διάταξη, οι εικονικές μηχανές διαγράφηκαν και αντικαταστάθηκαν με μία εικονική μηχανή, στην οποία ανατέθηκε το σύνολο των πόρων που προαναφέραμε και τους οποίους παρείχε νεφοϋπολογιστική υπηρεσία ~okenaos για το πείραμα.

Η εικονική μηχανή, πλέον θα στεγάσει αποκλειστικά το Apache JMeter, αρχικά δοκιμαστικά σε λογισμικό Windows Server για να επιβεβαιωθεί ότι οι πόροι ήταν επαρκείς και έπειτα σε λογισμικό Ubuntu Server για την διεξαγωγή του πειράματος. Οι πόροι κρίθηκαν επαρκείς, καθώς μετά από πολυήμερες δοκιμές, το Apache JMeter σε γραφικό περιβάλλον (GUI Mode) κατάφερε να λειτουργήσει σε ικανοποιητικό βαθμό χωρίς την παρουσία προβλήματος με έως 1000 χρήστες και έχοντας πολλαπλούς listeners για την καταγραφή των αποτελεσμάτων.

4.1.2 Σχεδίαση Πειραματικής Διάταξης

Για τη διεξαγωγή του πειράματος αρχικά ορίστηκε η διάταξη που θα έπρεπε να υλοποιηθεί. Η πειραματική διάταξη αποτελείται από τρείς (3) εικονικές μηχανές με προδιαγραφές που συναντούν τις απαιτήσεις που αναφέρθηκαν προηγουμένως. Οι δυο εκ των τριών εικονικών μηχανών δημιουργήθηκαν σε διακομιστές της Hetzner Cloud στην Φινλανδία. Οι εικονικές μηχανές της Hetzner, επιλέγονται από μια λίστα εικονικών μηχανών με προκαθορισμένους πόρους, ώστε να μπορεί να επιλεγεί η βέλτιστη μηχανή για την εκάστοτε εργασία.

Η τρίτη μηχανή που χρησιμοποιήθηκε ήταν της νεφοϋπολογιστικής υπηρεσίας ~okeanos.

Τα χαρακτηριστικά των εικονικών μηχανών που επιλέχθηκαν και χρησιμοποιήθηκαν, καθώς και τα λειτουργικά συστήματα τα οποία εγκαταστάθηκαν σε αυτές, περιγράφονται στον παρακάτω πίνακα.

Πίνακας 4.2: Χαρακτηριστικά Εικονικών Μηχανών

	Kubernetes Master	**Kubernetes Node**	**Apache JMeter**
Πάροχος	Hetzner	Hetzner	~okeanos
CPU (Cores)	3	2	8
RAM (GB)	4	4	8
Disk Space (GB)	80	40	40
Distributor ID	Ubuntu	Ubuntu	Ubuntu

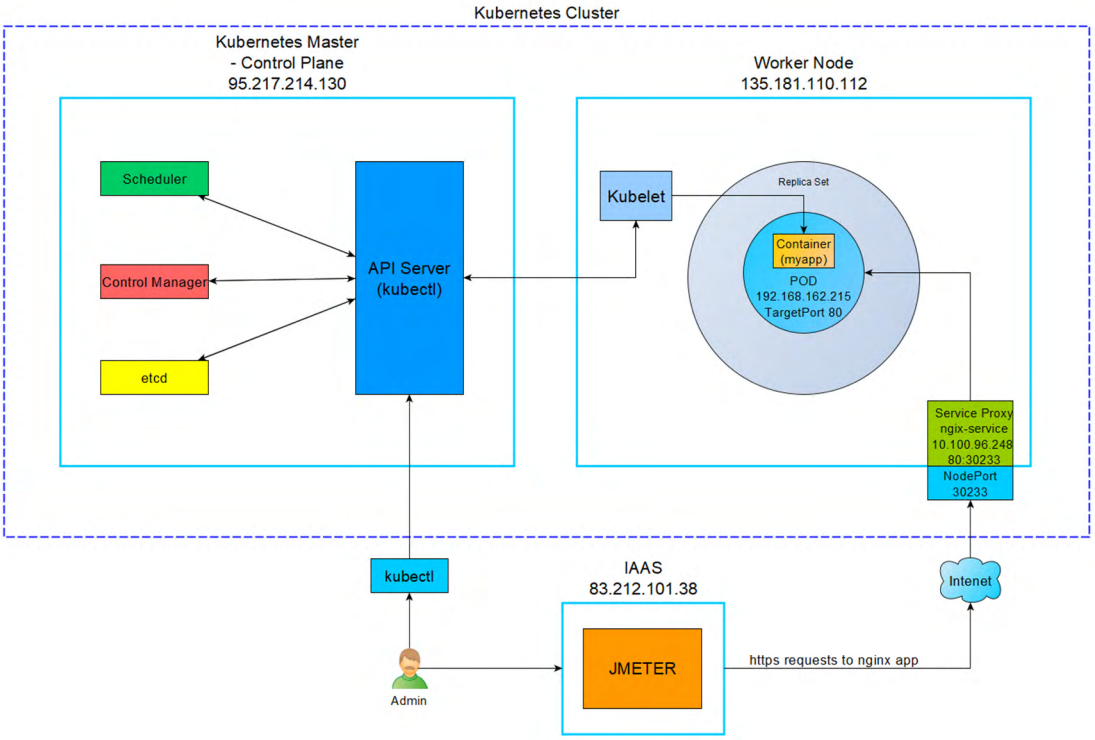
Description	Ubuntu 20.04.1 LTS	Ubuntu 20.04.1 LTS	Ubuntu 16.04.3 LTS
Release	20.04	20.04	16.04
Codename	focal	focal	xenial

Με το πέρας της εγκατάστασης των λειτουργικών συστημάτων, ως επόμενο βήμα ορίστηκε η εγκατάσταση του ενορχηστρωτή Kubernetes και η δημιουργία συστοιχίας του κυρίου κόμβου (master node) (*δηλ.* του επιπέδου ελέγχου με τον worker node). Τέλος, έγινε εγκατάσταση του JMeter στην νεφοϋπολογιστική υπηρεσία ~okeanos. Το JMeter θα λειτουργήσει ως γεννήτρια κίνησης προς τον Kubernetes worker. Μέσω αυτού, ταυτοχρόνως θα συλλέγονται μετρήσεις για την απόδοση του συστήματος.

Να σημειωθεί ότι επιλογή αποτελεί η λειτουργία του συστήματος ως ανοικτό (*δηλ.* τα νέα αιτήματα που καταφθάνουν είναι ανεξάρτητα από την ολοκλήρωση προηγούμενων αιτημάτων) [16].

4.1.3 Αρχιτεκτονική Πειραματικής Διάταξης

Η αρχιτεκτονική της πειραματικής διάταξης φαίνεται στις Εικόνες 4.1 και 4.2, όπως διαμορφώθηκε εγκαθιστώντας τον Horizontal Pod Autoscaler. Αντιπροσωπεύει ένα βασικό και απλοποιημένο μοντέλο μια Kubernetes συστοιχίας, έχοντας τον Kubernetes master και ένα μοναδικό worker node.

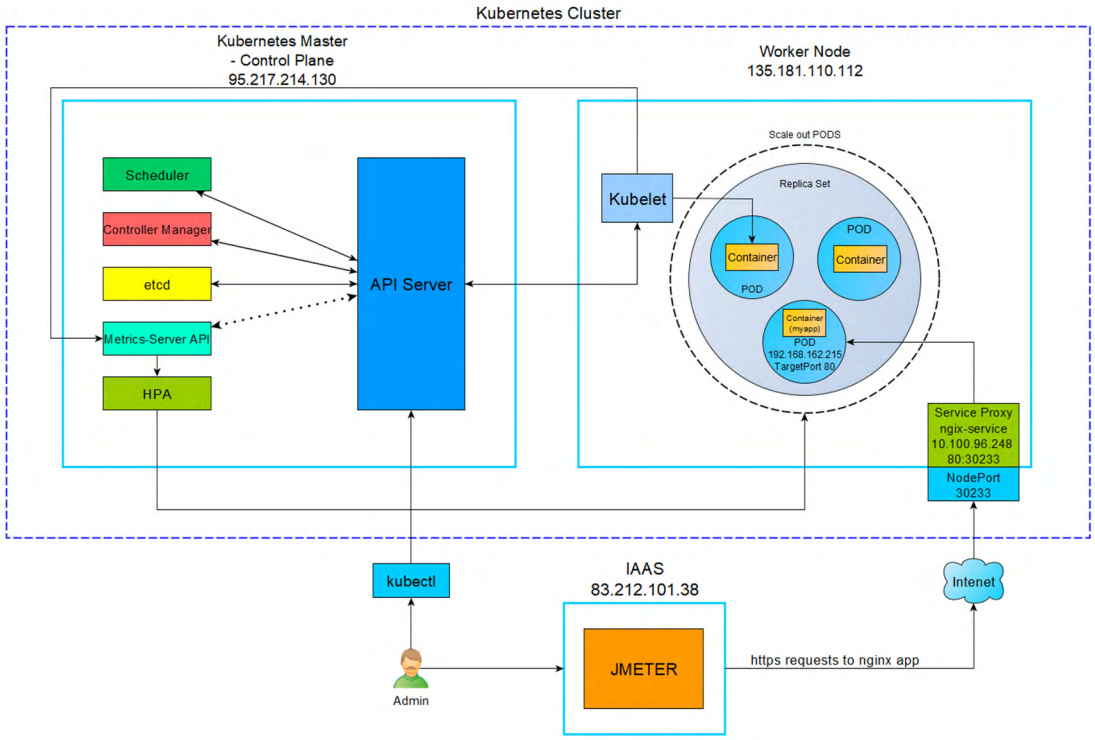


Εικόνα 4.1: Πειραματική Διάταξη Ενός Pod

Ο διαχειριστής, μέσω του JMeter στέλνει αιτήματα στον περιέκτη εντός του pod μέσω της NodePort 30233. Από εκεί γίνεται αντιστοιχία με την Port 80 της υπηρεσίας του Nginx. Κατόπιν, η Port 80 αντιστοιχείται με την TargetPort 80 του pod στην οποία τρέχει καθ' αυτή η εφαρμογή.

Στην παρούσα διάταξη (Εικόνα 4.1) η συστοιχία αποτελείται αποκλειστικά από ένα pod με προκαθορισμένο το μέγιστο των πόρων του, με αποτέλεσα όλα τα αιτήματα να εξυπηρετούνται από αυτό, λειτουργώντας έτσι ως αφιερωμένη (dedicated) εικονική μηχανή που εξυπηρετεί αποκλειστικά μια εργασία.

Η δεύτερη εκδοχή της πειραματικής διάταξης (Εικόνα 4.2) περιγράφει την προαναφερθείσα διαδικασία, έχοντας πλέον εγκαταστήσει τον Horizontal Pod Autoscaler και κατ' επέκταση λειτουργεί πλέον με πολλαπλά pod τα οποία μοιράζονται τους αρχικά δοθέντες πόρους.



Εικόνα 4.2: Πειραματική Διάταξη με Horizontal Pod Autoscaler

Έχοντας θέσει ένα μέγιστο όριο πόρων και ένα μέγιστο όριο στο ποσοστό χρήσης των πόρων ανά pod, ο metrics server μέσω του kubelet παρακολουθεί το εκάστοτε pod. Την στιγμή που το pod αυτό τείνει να ξεπεράσει το προαναφερθέν ποσοστό, ο Metrics Server δίνει εντολή στον Horizontal Pod Autoscaler να δημιουργήσει ένα νέο pod, πανομοιότυπο με το προηγούμενο.

Ο μέγιστος αριθμός των pod που μπορούν να δημιουργηθούν ορίζεται από τον διαχειριστή στον Horizontal Pod Autoscaler.

4.2 Επιλογή Παραμέτρων για το Πείραμα

4.2.1 Επιλογή Πόρων στα Pods

Το Kubernetes είναι δομημένο με σκοπό την απλοποίηση των διαδικασιών και τη διευκόλυνση της παραγωγής. Έχοντας αυτό υπ' όψη, έχει θέσει ως προεπιλογή, το ένα pod να μην ξεπερνάει σε χρήση το 50% των συνολικών πόρων του worker στον οποίο στεγάζεται.

Με αυτή τη λογική ως γνώμονα, εξετάζοντας την απόδοση του συστήματος και εστιάζοντας κατά κόρων στην χρήση της CPU συνδυαστικά με τον αριθμό των pods, θέσαμε ως ανώτατο όριο σε κάθε υποπερίπτωση τον ένα πυρήνα (1000 millicores), θεωρητικά υποθέτοντας ότι οι εναπομείναντες πόροι έχουν ως στόχο να είναι διαθέσιμοι για διαφορετικές εργασίες εντός του worker.

Επομένως, όταν στην περίπτωση του ενός pod έχουμε ως όριο τα 1000 millicores, στην περίπτωση περισσότερων pods, τα 1000 millicores διαιρούνται με τον μέγιστο αριθμό pods που έχουμε θέσει. Για παράδειγμα στα maximum τέσσερα pods, θα έχουμε 250 millicores ανά pod.

Με τον παραπάνω τρόπο, μπορεί να προσδιοριστεί με ακρίβεια η απόδοση ενός συστήματος που χρησιμοποιεί προκαθορισμένο αριθμό πόρων έχοντας είτε συγκεντρωμένους πόρους σε ένα pod και άμεσα διαθέσιμους για μια συγκεκριμένη εργασία, είτε χωρισμένους σε πολλαπλά pod, διαθέτοντας όμως ένα ποσοστό άμεσα διαθέσιμο και τους υπολειπόμενους πόρους ελεύθερους έως ότου χρειαστούν. Αξίζει να τονιστεί ότι η επιλογή των παραμέτρων αυτών, έγινε ώστε να υπάρχει κοινή σταθερά για μέτρηση και τη σύγκριση της απόδοσης των πειραματικών διατάξεων με σταθερούς πόρους και δεν είναι η ιδανική επιλογή για τη μέγιστη απόδοση ενός συστήματος.

4.2.2 Εργασία εντός του Περιέκτη

Αρχικά οι δοκιμές έγιναν χωρίς κάποια εργασία εντός του περιέκτη, απαντώντας απλά στα http αιτήματα που δημιουργούσε ο JMeter. Παρατηρήθηκε ότι όλα τα αιτήματα εξυπηρετούνταν ανεξάρτητα από τον αριθμό ή τη συχνότητα που εμφανίζονται.

Αποφασίστηκε να δημιουργηθεί μια εργασία εντός του περιέκτη ώστε ο worker να αναγκαστεί να χρησιμοποιήσει επιπλέον πόρους και να έρθει αντιμέτωπος με τα όριά του όσον αφορά τη χρήση των διαθέσιμων σε αυτόν πόρων.

Η εργασία με τίτλο «kolastirion-app» εντός του περιέκτη, δημιουργήθηκε σε προγραμματιστική γλώσσα Python από τον Γεώργιο Καβουσανό, μεταπτυχιακό φοιτητή του Πολυτεχνείου της Δανίας και αναρτήθηκε στο Github[2], απ' όπου και χρησιμοποιήθηκε ως εικόνα για τους περιέκτες.

Εργασία «kolastirion-app»:

```
from flask import Flask
app = Flask(__name__)
baseline_range = 100
@app.route('/')
def baseline():
    for i in range(baseline_range):
        eval(str(i)+' + '+str(i*i))
    return "Result:"+ str(baseline_range)

@app.route('/heavy')
def heavy():
    max_range=baseline_range*100
    for i in range(max_range):
        eval(str(i)+' + '+str(i*i))
    return "Result:"+ str(max_range)

@app.route('/kolasis')
def kolastirion():
```

[2] https://hub.docker.com/r/georgsatyros/kolastirion-app

```
    max_range=baseline_range*1000
    for i in range(max_range):
        eval(str(i)+' + '+str(i*i))
    return "Operations:"+ str(max_range)
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80)
```

Η εργασία δημιουργήθηκε έχοντας εντός τρία διαφορετικά σενάρια. Το κάθε σενάριο ξεκινώντας από το βασικό (`baseline_range = 100`) πραγματοποιεί υπολογισμούς (`eval(str(i)+' + '+str(i*i))`) και φέρνει ένα αποτέλεσμα (`return "Result:"+ str(baseline_range)`). Το δεύτερο σενάριο με όνομα «`Heavy`», υπολογίζει το εκατονταπλάσιο του βασικού σεναρίου (`max_range=baseline_range*100`) και τέλος το σενάριο με όνομα «`kolasis`», υπολογίζει το χιλιαπλάσιο του βασικού σεναρίου (`max_range=baseline_range*1000`).

Τα τρία αυτά σενάρια, δημιουργήθηκαν αρχικά για δοκιμές κόπωσης του συστήματος με σκοπό την επιλογή του σεναρίου που εξυπηρετούσε ικανοποιητικά το σκοπό του πειράματος συνδυαστικά με τις παραμέτρους που ορίστηκαν στην γεννήτρια κίνησης.

Το τελικό σενάριο το οποίο επιλέχθηκε μετά από πληθώρα δοκιμών και συγκριτικών αποτελεσμάτων ήταν το «kolasis». Το σενάριο αυτό, συνδυαστικά με τον μεγάλο αριθμό χρηστών, οδηγούσε σε εξαιρετικά μεγάλο ποσοστό σφάλματος εξυπηρέτησης των αιτημάτων στο σενάριο του ενός pod και κρίθηκε κατάλληλο για να εξετάσουμε τη βελτίωση στην εξυπηρέτηση αυξάνοντας τον αριθμό των pods στον worker node.

4.2.3 Επιλογή Αριθμού Pods

Αρχικά, το πείραμα πραγματοποιήθηκε έχοντας ενεργό ένα μοναδικό καθ' όλη τη διάρκεια του πειράματος, παίρνοντας τις μετρήσεις και θέτοντας τη βάση για τη σύγκριση των αποτελεσμάτων στη συνέχεια. Η CPU η οποία ζητήθηκε να είναι άμεσα διαθέσιμη για το pod, επιλέχθηκε να είναι 450 millicores, θέτοντας με αυτό τον τρόπο τη μέγιστη τιμή των πόρων κατά την εκκίνηση ενός pod, με βάση το πλάνο που είχε τεθεί για τη συνέχεια του πειράματος.

Πίνακας 4.3: Πειραματικά Σενάρια Βάση του Αριθμού των Pods

Minimum No. of Pods	Maximum No. of Pods	CPU Utilization Percentage (%)	Requested CPU	CPU Limits (Millicores)
1	1	-	450	1000
1	2	99	450	500
1	4	99	225	250

Μετά το πέρας των μετρήσεων με το ένα pod, επιλέχθηκαν τρία διαφορετικά σενάρια που θα έτρεχε το πείραμα. Σε όλα τα σενάρια, κοινός παρονομαστής αποτελεί η εκκίνηση του πειράματος πάντοτε με ένα pod, όπως επίσης και η συνολική χρήση της CPU του worker να μην ξεπερνά τα 1000 millicores. Ξεκινώντας με το ένα pod σε κάθε νέο σενάριο διπλασιαζόταν ο μέγιστος αριθμός των pods (Πίνακας 4.3). Το πείραμα σταμάτησε με το μέγιστο αριθμό να είναι τα τέσσερα (4) pods, καθώς μετά οι πόροι κρίθηκαν ανεπαρκείς για να υποδιπλασιαστούν ώστε να εξυπηρετήσουν ικανοποιητικά την εργασία εντός του περιέκτη και να γίνει λήψη ορθών αποτελεσμάτων.

Η ελάχιστη ζητούμενη CPU για την εκκίνηση στο σενάριο των δυο (2) pod ήταν τα 450 millicores, έτσι ώστε να έχουμε απευθείας σύγκριση των αποτελεσμάτων με αυτών του ενός pod, Θέτοντας ως μέγιστο όριο τα 500 millicores ώστε να μην υπερβούμε το όριο που τέθηκε αρχικά όσον αφορά τους συνολικούς πόρους.

Αντίστοιχα, στο σενάριο με μέγιστο αριθμό τέσσερα (4) pods, η ελάχιστη ζητούμενη CPU για την εκκίνησή τους υποδιπλασιάστηκε στα 225 millicores, όπως υποδιπλασιασμό υπέστη και το μέγιστο όριο χρήσης στα 250 millicores.

Το ποσοστό χρησιμοποίησης της CPU χρησιμοποιείται για να ορίσουμε το χρόνο δημιουργίας ενός επιπλέον pod από τον Horizontal Pod Autoscaler. Το ποσοστό αυτό υπολογίζεται με βάση το μέγιστο όριο χρήσης της CPU του ενός pod (CPU limits). Όταν ο metrics server διαβάσει ότι η χρήση της CPU φτάσει ή ξεπεράσει το ορισμένο από εμάς ποσοστό, ενημερώνει τον Horizontal Pod Autoscaler για να δημιουργήσει ένα καινούριο pod. Η λογική γύρω από το ποσοστό χρησιμοποίησης της CPU μας λέει ότι, όταν παρατηρηθεί αύξηση χρήσης της CPU από ένα pod, να υπάρχει έτοιμο ή να προετοιμάζεται το επόμενο pod πριν χρειαστούν άμεσα οι πόροι του.

Στο σύνολο του πειράματος, το ποσοστό χρησιμοποίησης της CPU (CPU utilization percentage) τέθηκε στο 99% με σκοπό να αξιοποιηθούν στο σύνολό τους οι πόροι του κάθε ενεργού pod πριν τη δημιουργία του επόμενου. Δεν επιλέχθηκε το 100% σαν

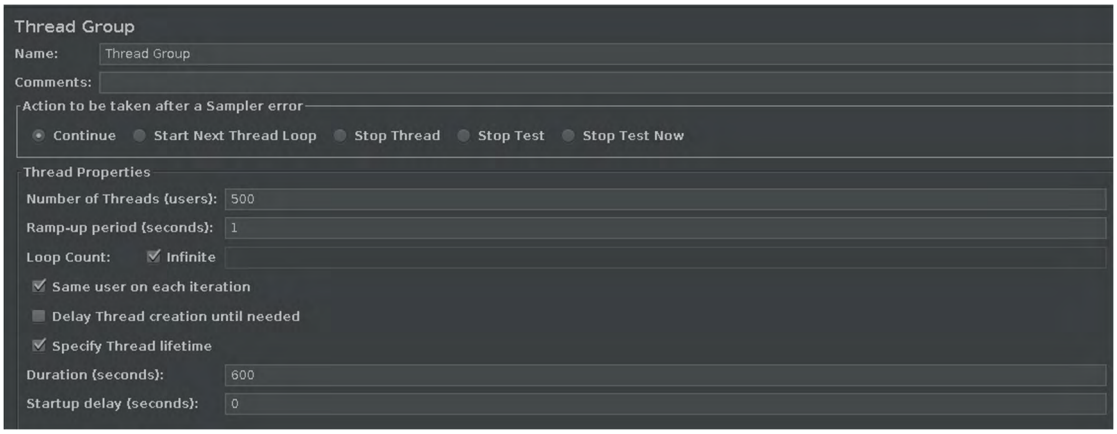
ποσοστό για την αποφυγή πιθανών σφαλμάτων, καθώς έχει δημιουργηθεί με τέτοιο τρόπο ώστε το ποσοστό που χρησιμοποιείται στην παραγωγή (όχι σε ερευνητικό επίπεδο) και έχει παρατηρηθεί ότι αποδίδει τα μέγιστα, είναι κάτω του 50%.

4.2.4 Επιλογή Παραμέτρων στο Εργαλείο Δοκιμής Φορτίου

Η επιλογή των παραμέτρων που αφορούν το εργαλείο δοκιμής φορτίου (JMeter), έγινε με κύριο γνώμονα τις παραμέτρους που χρησιμοποιήθηκαν στην εργασία των D. Kallergis et. al [12].

Οι παράμετροι οι οποίες λήφθηκαν υπ' όψη (Εικόνα 4.3) ήταν:

- Ο αριθμός των νημάτων (number of threads)
- Η περίοδος ανάκλισης (ramp-up period)
- Το απαρίθμημα βρόγχου (loop count)
- Η διάρκεια (duration)
- Η καθυστέρηση εκκίνησης (startup delay)
- Οι χρονομετρητές κατανομής φορτίου



Εικόνα 4.3: Επιλογή Παραμέτρων στο Εργαλείο Δοκιμής Φορτίου

4.2.4.1 Αριθμός Νημάτων

Ο αριθμός νημάτων αντιπροσωπεύει τον αριθμό των χρηστών οι οποίοι πραγματοποιούν τα http αιτήματα. Κάθε νήμα θα πραγματοποιήσει το αίτημα στο σύνολό του, ανεξάρτητα από τους υπόλοιπους. Κατά τη διάρκεια του πειράματος, χρησιμοποιήθηκαν πολλαπλά νήματα για την προσομοίωση ταυτόχρονων συνδέσεων.

Κατά την ομοιόμορφη κατανομή, για κάθε σενάριο μέγιστου αριθμού pods ξεχωριστά (1, 2, 4, 8), λήφθηκαν μετρήσεις για πέντε διαφορετικές ομάδες χρηστών. Οι ομάδες αυτές αποτελούνταν από 250, 500, 1000, 2000, 4000 νήματα.

4.2.4.2 Περίοδος Ανάκλισης

Η περίοδος ανάκλησης περιγράφει στο JMeter το χρόνο ο οποίος χρειάζεται για να "ανακλιθεί" ο πλήρης αριθμός των νημάτων που έχουν επιλεγεί. Κάθε νήμα υπολογίζεται να ξεκινά σε χρονικό διάστημα (δευτερόλεπτα) ίσο με το πηλίκο της περιόδου ανάκλισης (σε δευτερόλεπτα) προς τον συνολικό αριθμό των νημάτων, μετά την έναρξη του προηγούμενου.
Σε όλη τη διάρκεια του πειράματος, η περίοδος ανάκλισης ορίστηκε σε 1 δευτερόλεπτο (Εικόνα 4.3).

4.2.4.3 Απαρίθμημα Βρόγχου

Το απαρίθμημα βρόγχου ελέγχει τον αριθμό που θα επαναληφθεί το πείραμα. Εάν εισαχθεί μια τιμή απαρίθμησης βρόχου 1, το JMeter θα εκτελέσει το πείραμα μόνο μία φορά. Ως τιμή για το πείραμά μας ορίστηκε το άπειρο (infinite) (Εικόνα 4.3) με σκοπό το πείραμα να εκτελείται επανειλημμένα έως ότου σταματήσει στην ορισμένη από εμάς διάρκεια.

4.2.4.4 Διάρκεια και Καθυστέρηση Εκκίνησης

Ο όρος «διάρκεια» περιγράφει την χρονική διάρκεια σε δευτερόλεπτα στην οποία εκτελείται το πείραμα. Ξεκινάει να μετράει τη χρονική στιγμή που δόθηκε η εντολή εκτέλεσης και συνεχίζει μέχρι να ολοκληρωθεί το ορισμένο από εμάς χρονικό διάστημα, όπου και τερματίζεται η επαναδημιουργία των νημάτων.
Έπειτα από πολλαπλές δοκιμές με πληθώρα χρόνων εκτέλεσης (από 150 έως 1800 δευτερόλεπτα), καταλήξαμε ότι μετά τη χρονική διάρκεια των 600 δευτερολέπτων, σε συνδυασμό με τις υπόλοιπες παραμέτρους που ορίστηκαν, οι μεταβολές στις τιμές των αποτελεσμάτων ήταν αρκετά μικρές για να ληφθούν υπ’ όψη. Αυτό είχε ως αποτέλεσμα, η διάρκεια διεξαγωγής του πειράματος να οριστεί σταθερά στα 600 δευτερόλεπτα (Εικόνα 4.3).

Η καθυστέρηση κατά την εκκίνηση του πειράματος ορίστηκε στα 0 δευτερόλεπτα (Εικόνα 4.3), καθώς δεν επηρεάζει το πείραμα η όποια διαφοροποίηση της τιμής υπό την προϋπόθεση να είναι σταθερή για όλα τα σενάρια της πειραματικής διαδικασίας.

4.2.4.5 Χρονομετρητές Κατανομής Φορτίου

Το JMeter στέλνει αιτήματα χωρίς καθυστέρηση μεταξύ κάθε δειγματολήπτη/αιτήματος με αποτέλεσμα να εκτελούνται όλα μαζί.

Οι χρονομετρητές (timers) του JMeter έχουν δημιουργηθεί με σκοπό την καθυστέρηση, για ένα συγκεκριμένο χρονικό διάστημα, πριν από κάθε δειγματολήπτη να εκτελέσει το αίτημά του.

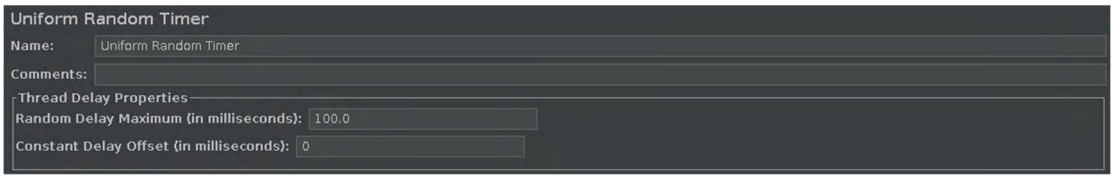
Θέλοντας να προσομοιώσουμε μια κίνηση χρηστών στο πείραμα και όχι μια πλημμύρα αιτημάτων επιλέξαμε να χρησιμοποιήσουμε δυο διαφορετικούς χρονομετρητές, όμοιους με αυτούς που χρησιμοποιήθηκαν στην εργασία των D. Kallergis et. al [12].

Οι χρονομετρητές που χρησιμοποιήθηκαν ήταν:

- Χρονομετρητής ομοιόμορφης κατανομής - Uniform random timer
- Χρονομετρητής κατανομής Poisson - Poisson random timer

4.2.4.5.1 Χρονομετρητής Ομοιόμορφης Κατανομής

Ο χρονομετρητής ομοιόμορφης κατανομής του JMeter καθυστερεί κάθε αίτημα για ένα τυχαίο χρονικό διάστημα. Το χρονικό διάστημα αυτό επιλέγεται μέσα από την παράμετρο «Random Delay Maximum», και θέτει το μέγιστο διάστημα μεταξύ των αιτημάτων. Είναι προκαθορισμένη στα 100 χιλιοστοδευτερόλεπτα και παρέμεινε η ίδια σε όλα τα σενάρια του πειράματος (Εικόνα 4.4).

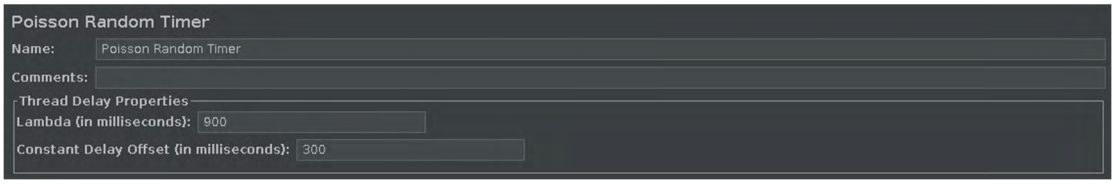


Εικόνα 4.4: Χρονομετρητής Ομοιόμορφης κατανομής

4.2.4.5.2 Χρονομετρητής Κατανομής Poisson

Ο χρονομετρητής κατανομής Poisson (Εικόνα 4.5) χρησιμοποιεί τον επόμενο τυχαίο αριθμό από την κατανομή Poisson χρησιμοποιώντας το άθροισμα της καθορισμένης από τον χρήστη παράμετρο «λ» ή «Λάμδα» (σε χιλιοστοδευτερόλεπτα) και τη «Σταθερή Μετατόπιση Καθυστέρησης» (Constant Delay Offset). Όταν

χρησιμοποιηθούν οι προεπιλεγμένες παράμετροι του JMeter (100 ms ως λάμδα και 300 ms ως μετατόπιση σταθερής καθυστέρησης), ο χρονομετρητής κατανομής Poisson θα δημιουργήσει μια καθυστέρηση στην κατά προσέγγιση περιοχή μεταξύ 375 έως 425 χιλιοστοδευτερολέπτων.



Εικόνα 4.5: Χρονομετρητής Κατανομής Poisson

Κατά τη διάρκεια του πειράματος στα σενάρια με τη χρήση του χρονομετρητή κατανομής Poisson, ο αριθμός των νημάτων επιλέχθηκε να παραμείνει σταθερός στους 1000 και να μεταβάλλεται μόνο η τιμή «λ». Εστιάσαμε στο σύνολο των χιλίων χρηστών διότι είναι η διάμεσος τιμή (median) του συνόλου των χρηστών. Σταθερή παρέμεινε και η τιμή της μεταβλητής Σταθερής Μετατόπισης Καθυστέρησης στα 300 χιλιοστοδευτερόλεπτα.

Επιλέχθηκε να γίνουν πολλαπλές δοκιμές εισάγοντας διαφορετικές τιμές λάμδα για το φορτίο εισόδου ώστε να παρατηρηθεί η συμπεριφορά κάθε σεναρίου. Οι τιμές αυτές ήταν λ = 0.0011, 0.0022, 0.2, 1. Το λάμδα σε χιλιοστοδευτερόλεπτα υπολογίζεται από τον τύπο:

$$\lambda\ (ms) = 1 / \lambda$$

Επομένως οι τιμές των λάμδα που καταχωρήθηκαν στον JMeter και διεξήχθησαν τα σενάρια ήταν λ (ms) = 900, 455, 5, 1 αντίστοιχα.

4.3 Υλοποίηση Πειραματικής Διάταξης

Λεπτομερειακή καταγραφή της υλοποίησης βρίσκεται στο Παράρτημα.

Κεφάλαιο 5 – Αποτελέσματα και Συζήτηση

5.1 Αποτελέσματα

Καθ' όλη τη διάρκεια των πειραμάτων, η χρήση της CPU στο/στα pods του worker ήταν πάντα στο 100%, λόγο του υψηλού φόρτου εργασίας που αναθέσαμε, επομένως δεν κρίθηκε απαραίτητη η παρακολούθηση και καταγραφή της απόδοσης από την πλευρά του Kubernetes.
Οι μετρικές που κρατήσαμε ήταν:

- Συνολικός χρόνος πειράματος
- Συνολικός αριθμός δειγμάτων
- Διεκπεραιωτική ικανότητα (throughput)
- Avg/min/max διαρρεύσαντος χρόνου
- Ποσοστό σφάλματος του πειράματος
- Χρονοσφραγίδα (time stamp) ανά δείγμα
- Διαρρεύσας χρόνος (elapsed time) ανά δείγμα
- Επιτυχία ανά δείγμα
- Απεσταλμένα Bytes ανά δείγμα
- Ενεργά δείγματα (active thread counts) ανά δείγμα
- Καθυστέρηση (latency) ανά δείγμα
- Χρόνος σύνδεσης ανά δείγμα

Παρατηρήθηκε επιπλέον μια απόκλιση στον χρόνο ολοκλήρωσης του πειράματος, διαφορετική από αυτή που είχαμε ορίσει κατά την επιλογή των παραμέτρων. Όσο ο αριθμός των αιτημάτων-χρηστών αυξανόταν, τόσο αυξανόταν και ο χρόνος ολοκλήρωσης. Αυτό οφείλεται στην ουρά που έχει δημιουργηθεί για την εξυπηρέτηση των αιτημάτων. Ενώ η γεννήτρια κίνησης έχει σταματήσει να παράγει αιτήματα, αναμένει έως ότου εξυπηρετηθεί ή αποτύχει το σύνολο των αιτημάτων τα οποία έχουν ήδη αποσταλεί πριν το πέρας του πειράματος.

5.1.1 Αποτελέσματα σε Ομοιόμορφη Κατανομή

Το πείραμα αρχικά έτρεξε με ενεργό το χρονομετρητή ομοιόμορφης κατανομής για τα σενάρια του ενός (1), δυο (2) και τεσσάρων (4) pods.

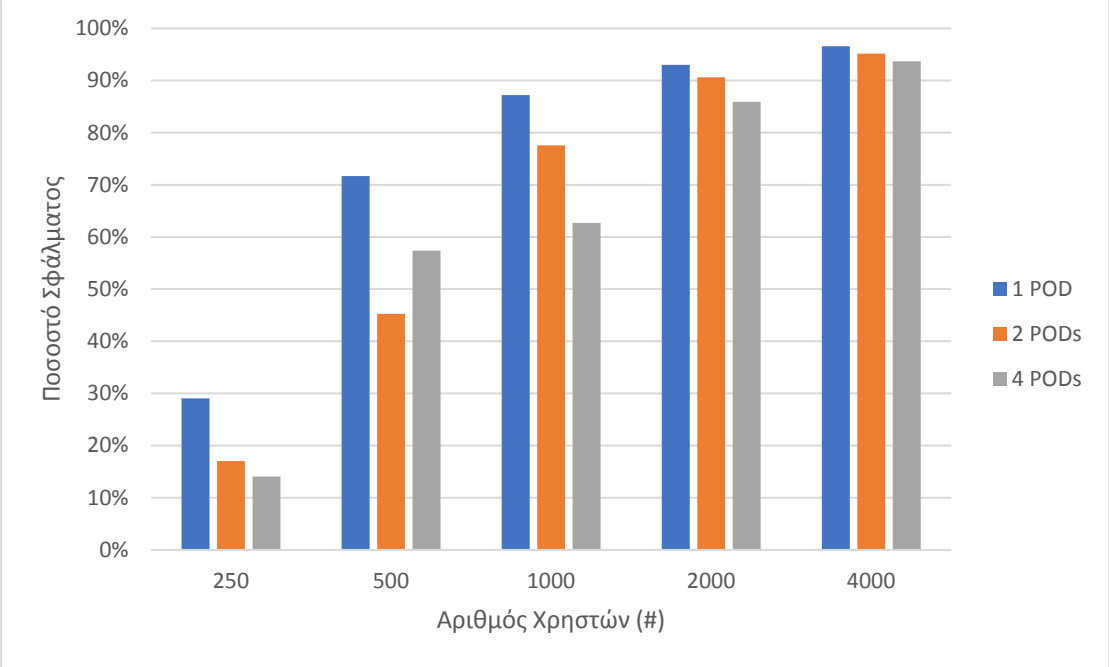
5.1.1.1 Ποσοστό Σφάλματος

Τα αποτελέσματα του ποσοστού σφάλματος ανά σενάριο φαίνονται στον πίνακα 5.1.

Πίνακας 5.1: Ποσοστό Σφάλματος Ομοιόμορφης Κατανομής

	Ομοιόμορφη Κατανομή		
Αριθμός POD	**1**	**2**	**4**
Αριθμός Χρηστών	**Ποσοστό Σφάλματος**		
250	29,06%	17,01%	14,06%
500	71,69%	45,27%	57,37%
1000	87,20%	77,58%	62,69%
2000	93,01%	90,61%	85,91%
4000	96,58%	95,17%	93,70%

Τα αποτελέσματα όσον αφορά τις μετρήσεις, ήταν τα αναμενόμενα, καθώς με την αύξηση των χρηστών ανά πείραμα, παρατηρήθηκε ότι το ποσοστό σφάλματος μειωνόταν αντιδιαμετρικά με την αύξηση του αριθμού των pods (Εικόνα 5.1).
Επιπροσθέτως, παρατηρήθηκε σημαντική μείωση στο ποσοστό σφάλματος κατά τον διπλασιασμό τον pods. Με μεγαλύτερη αποτελεσματικότητα στο σενάριο των χιλίων (1000) χρηστών-αιτημάτων, αφού αρχικά το σφάλμα έφτασε το 87,20%, στη συνέχεια μειώθηκε κατά 10%, δηλαδή 77,58% με τα δυο pods. Κατέληξε να είναι 62,69% με τα τέσσερα (4) ενεργά pods, επιπλέον μείωση δηλαδή, κατά περίπου 15 ποσοστιαίες μονάδες και συνολική μείωση κατά 25%, γεγονός που καθιστά βέβαιο ότι η χρήση σταθερών πόρων και έχοντας πολλαπλή επεξεργασία επηρεάζει θετικά την ταχύτητα εξυπηρέτησης.



Εικόνα 5.1: Ποσοστό Σφάλματος Ομοιόμορφης Κατανομής

Σημαντική παρατήρηση αποτελεί και η αύξηση σφάλματος μετά τους χίλιους (1000) χρήστες-νήματα. Όσο διπλασιαζόταν ο αριθμός των χρηστών, παρατηρήθηκε αισθητή μείωση στη διαφορά του σφάλματος ανά αριθμό pods. Εάν συγκρίνουμε τα αποτελέσματα θα παρατηρήσουμε ότι, ενώ στους χίλιους (1000) χρήστες-νήματα η διαφορά στο σφάλμα ανάμεσα στο ένα (1) και τέσσερα pods ήταν περίπου 25%, η διαφορά στους 4000 χρήστες-νήματα ανήλθε, παρά τη μείωσή του, στο μόλις στο 3%. Αυτό μας οδηγεί στο συμπέρασμα ότι όσο ανεβαίνει ο αριθμός των χρηστών-αιτημάτων, τόσο δυσκολότερη καθίσταται η εξυπηρέτηση, τείνοντας το σφάλμα να κυμανθεί στα ίδια επίπεδα καθώς πλησιάζουμε το 100% του ποσοστού του σφάλματος.

Ακόμα μια παρατήρηση έγινε στα σενάρια των πεντακοσίων (500) χρηστών-αιτημάτων, όπου κατά το σενάριο των δυο (2) pods υπήρξε σημαντική μείωση του σφάλματος, κατά περίπου 26.50%. Αυτή είναι και η μεγαλύτερη μείωση σφάλματος που παρατηρήθηκε καθ' όλη τη διάρκεια του πειράματος. Σε αντίθεση με το σύνολο των υπολοίπων αποτελεσμάτων, παρατηρήθηκε αύξηση του σφάλματος στα τέσσερα (4) pods σε σύγκριση με τα αποτελέσματα του σεναρίου των δυο (2) pods. Αξίζει να αναφερθεί σε αυτό το σημείο ότι η διαφορά σφάλματος κατά το σενάριο των τεσσάρων (4) pods στους πεντακόσιους (500) και χίλιους (1000) χρήστες-νήματα ήταν η ελάχιστη και ανέρχεται στο ~5.3%.

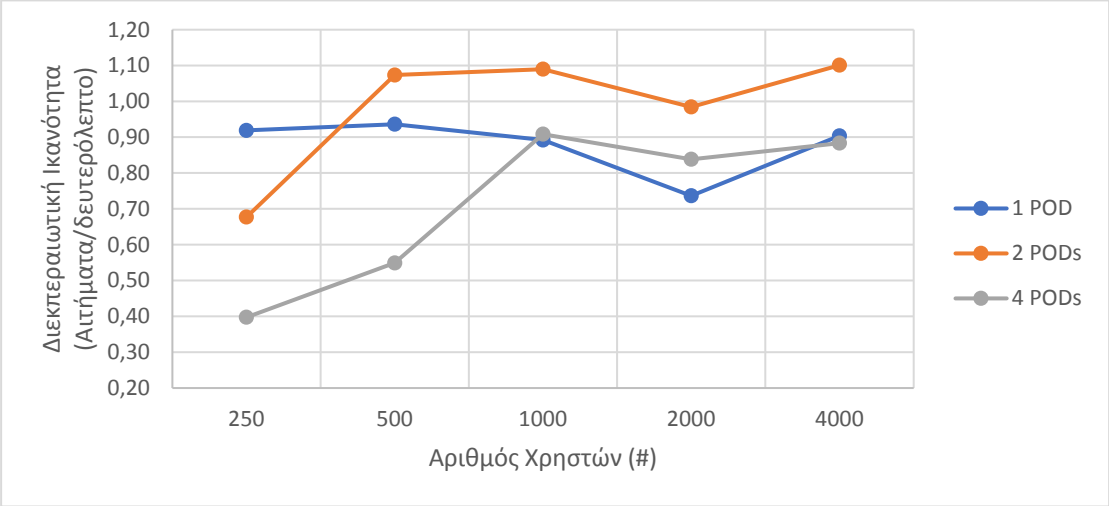
5.1.1.2 Διεκπεραιωτική Ικανότητα (Throughput)

Επιπλέον πληροφορίες για την απόδοση της συστοιχίας ανά σενάριο, πήραμε μέσω της διεκπεραιωτικής ικανότητας, η οποία περιγράφει τα επιτυχημένα αιτήματα ανά δευτερόλεπτο (Πίνακας 5.2).

Πίνακας 5.2: Διεκπεραιωτική Ικανότητα των Επιτυχημένων Αιτημάτων

	Ομοιόμορφη Κατανομή		
Αριθμός POD	**1**	**2**	**4**
Αριθμός Χρηστών	**Διεκπεραιωτική Ικανότητα**		
250	0,92	0,68	0,40
500	0,94	1,07	0,55
1000	0,89	1,09	0,91
2000	0,74	0,98	0,84
4000	0,90	1,10	0,88

Ενώ στους διακόσιους πενήντα (250) χρήστες η διεκπεραιωτική ικανότητα ήταν το μέγιστο κατά τις μετρήσεις με το ένα (1) pod και ανέρχεται στα 0.92 αιτήματα ανά δευτερόλεπτο, τα αποτελέσματα όλων των υπολοίπων μετρήσεων δείχνουν ότι η μέγιστη διεκπεραιωτική ικανότητα υπήρξε κατά την εκτέλεση του σεναρίου με τα 2 pods για κάθε μέτρηση, ανεξάρτητα τον αριθμό χρηστών (Εικόνα 5.2). Η μέγιστη διεκπεραιωτική ικανότητα ήταν τα 1,1 αιτήματα ανά δευτερόλεπτο και σημειώθηκε κατά την εκτέλεση του πειράματος με τους τέσσερις χιλιάδες (4000) χρήστες.



Εικόνα 5.2: Διεκπεραιωτική Ικανότητα ανά Αριθμό Χρηστών

Στο σενάριο με τα τέσσερα (4) pods τα αποτελέσματα ποικίλουν. Στους διακόσιους πενήντα (250) και στους πεντακόσιους (500) χρήστες, η διεκπεραιωτική ικανότητα είναι σημαντικά χαμηλότερη σε σύγκριση με τα υπόλοιπα σενάρια. Αντίθετα, στο σενάριο των δύο χιλιάδων (2000) χρηστών-νημάτων, η διεκπεραιωτική ικανότητα ξεπέρασε αρκετά αυτή του ενός pod, κατά 0,1 αίτημα ανά δευτερόλεπτο. Στα σενάρια των χίλιων (1000) και τεσσάρων χιλιάδων (4000) χρηστών-αιτημάτων, η διαφορά της διεκπεραιωτικής ικανότητας ανάμεσα στο ένα (1) pod και τα τέσσερα (4) pods είναι ελάχιστη (0,02 αιτήματα ανά δευτερόλεπτο).

Παρατηρήθηκε ότι η ταχύτητα εξυπηρέτησης αιτημάτων σε συνάρτηση με το χρόνο αυξάνεται στο σενάριο των δύο (2) pods όμως μειώνεται ξανά με το περαιτέρω διπλασιασμό τους.

Η συστοιχία φαίνεται να αποδίδει τα μέγιστα στα σενάρια των δύο (2) pods, από τους 500 χρήστες και άνω, αφού παρατηρήθηκε μείωση σφάλματος ταυτοχρόνως με αύξηση της ταχύτητας εξυπηρέτησης αιτημάτων σε συνάρτηση με το χρόνο.

5.1.2 Αποτελέσματα σε Κατανομή Poisson

Οι μετρήσεις του πειράματος συνεχίστηκαν με ενεργό τον χρονομετρητή κατανομής Poisson για τα σενάρια των 1, 2 και 4 pods και με σταθερό τον αριθμό χρηστών τους χίλιους (1000) χρήστες, μεταβάλλοντας την τιμή του λάμδα.

5.1.2.1 Ποσοστό Σφάλματος

Τα αποτελέσματα του ποσοστού σφάλματος ανά σενάριο φαίνονται στον πίνακα 5.3, κάνοντας σύγκριση με το σφάλμα της ομοιόμορφης κατανομής για τους χίλιους (1000) χρήστες.

Πίνακας 5.3: Ποσοστό Σφάλματος Κατανομής Poisson

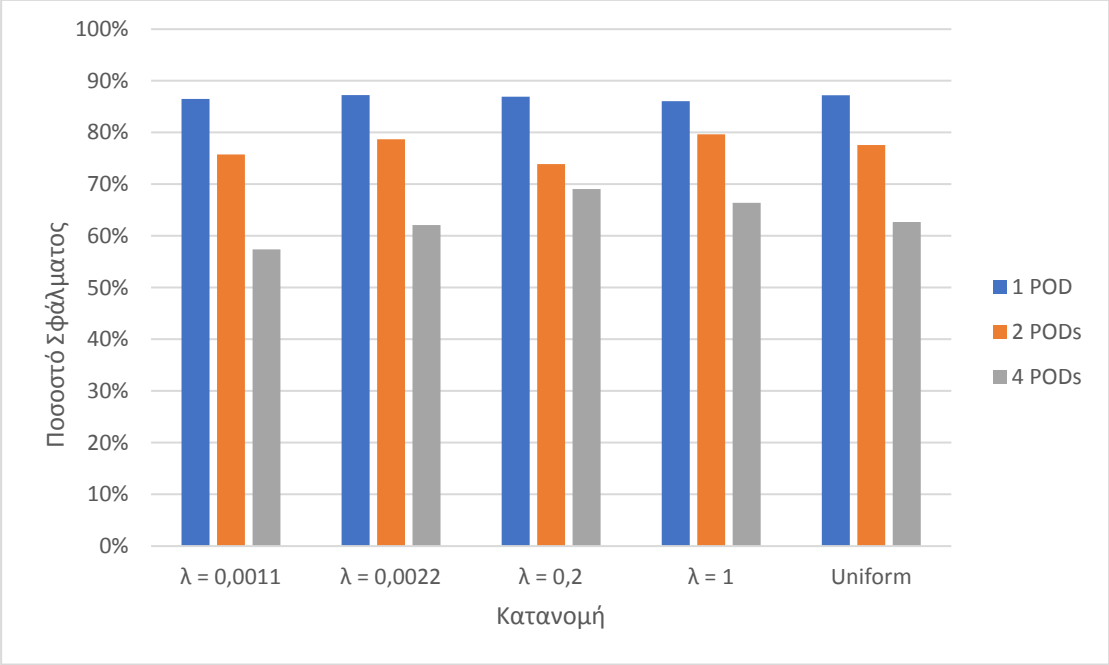
	Κατανομή Poisson		
Αριθμός POD	**1**	**2**	**4**
	Ποσοστό Σφάλματος		
λ = 0,0011	86.48%	75.74%	57.37%
λ = 0,0022	87.23%	78.69%	62.08%
λ = 0,2	86.91%	73.89%	69.07%

λ = 1	86.05%	79.63%	66.39%
Uniform	87,20%	77,58%	62,69%

Παρατηρούμε από τα αποτελέσματα τα οποία συλλέχθηκαν, ότι το μοτίβο όσον αφορά το ποσοστό σφάλματος ανά αριθμό pod είναι σχετικά ίδιο ανάμεσα στις μετρήσεις με διαφορετικά λάμδα, όπως και με τη σύγκριση του ποσοστού σφάλματος από την ομοιόμορφη κατανομή (Εικόνα 5.3).

Τα αποτελέσματα των μετρήσεων για όλες τις τιμές του λάμδα με το ένα (1) pod, κρατήθηκαν οριακά σταθερές και κυμάνθηκαν μεταξύ του 86,05% - 87,23%, δηλαδή διακύμανση οριακά πάνω από το 1%. Το ίδιο ίσχυσε για τα σενάρια των δυο (2) pods, με ελαφρώς μεγαλύτερη διακύμανση στις τιμές του σφάλματος οι οποίες βρέθηκαν ανάμεσα στο 73,89% με 79,63%.

Η μεγαλύτερη διαφοροποίηση στις τιμές του σφάλματος, παρατηρήθηκε στα σενάρια για τα τέσσερα (4) pods. Οι τιμές είχαν διαφορά η οποία ανέρχεται περίπου στο 12%, δηλαδή από 57,37% έως 69,07%, έχοντας το ελάχιστο ποσοστό σφάλματος, όταν το λάμδα ήταν ορισμένο σε 0,0011 (900ms).



Εικόνα 5.3: Ποσοστό Σφάλματος Κατανομής Poisson

5.1.2.2 Διεκπεραιωτική Ικανότητα (Throughput)

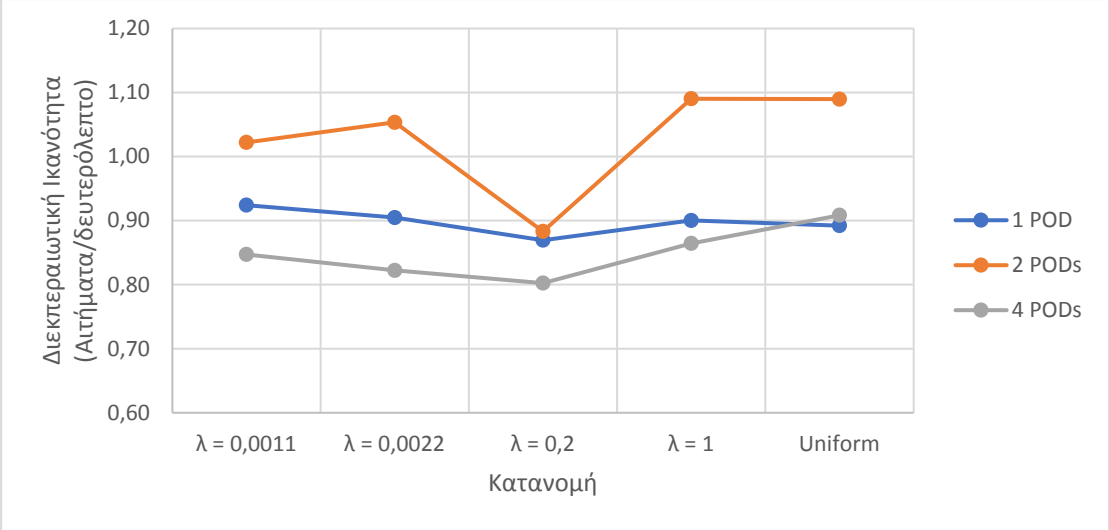
Όπως και στην ομοιόμορφη κατανομή, αντίστοιχα στην κατανομή Poisson μετρήσαμε τη διεκπεραιωτική ικανότητα με σταθερά τους χίλιους (1000) χρήστες διαφοροποιώντας τις τιμές του λάμδα και συγκρίνοντας τα αποτελέσματα με αυτά της ομοιόμορφης κατανομής (Πίνακας 5.4).

Πίνακας 5.4: Διεκπεραιωτική Ικανότητα των Επιτυχημένων Αιτημάτων

	Κατανομή Poisson		
Αριθμός POD	1	2	4
	Διεκπεραιωτική Ικανότητα		
λ = 0,0011	0,92	1,02	0,85
λ = 0,0022	0,90	1,05	0,82
λ = 0,2	0,87	0,88	0,80
λ = 1	0,90	1,09	0,86
Uniform	0,89	1,09	0,91

Παρατηρήθηκαν μικρές αυξομειώσεις στις διαφορές ανάμεσα στις τιμές των δειγμάτων. Τα διαγράμματα (Εικόνας 5.4) παρουσιάζουν μια τάση προς τη δημιουργία καμπύλης. Παρατηρήθηκε συνολική μείωση της εξυπηρέτησης αιτημάτων ανά δευτερόλεπτο σε όλα τα σενάρια που η τιμή του λάμδα είχε οριστεί ως 0,2 (5 ms), με την μεγαλύτερη να είναι αυτή του σεναρίου των δυο (2) pods, δηλαδή 0.88 εξυπηρετούμενα αιτήματα ανά δευτερόλεπτο, το οποίο και ήταν το χαμηλότερο σημείο στην προαναφερθείσα καμπύλη.

Σε όλες τις υπόλοιπες περιπτώσεις, τα σενάρια των δυο (2) pods ήταν τα μοναδικά τα οποία υπερέβησαν το ένα (1) αίτημα ανά δευτερόλεπτο, με μέγιστη εξυπηρέτηση στην περίπτωση του λάμδα να ισούται με 1 (1 ms) και να είναι όμοια η εξυπηρέτηση με το σενάριο της ομοιόμορφης κατανομής (1,09 αιτήματα ανά δευτερόλεπτο).

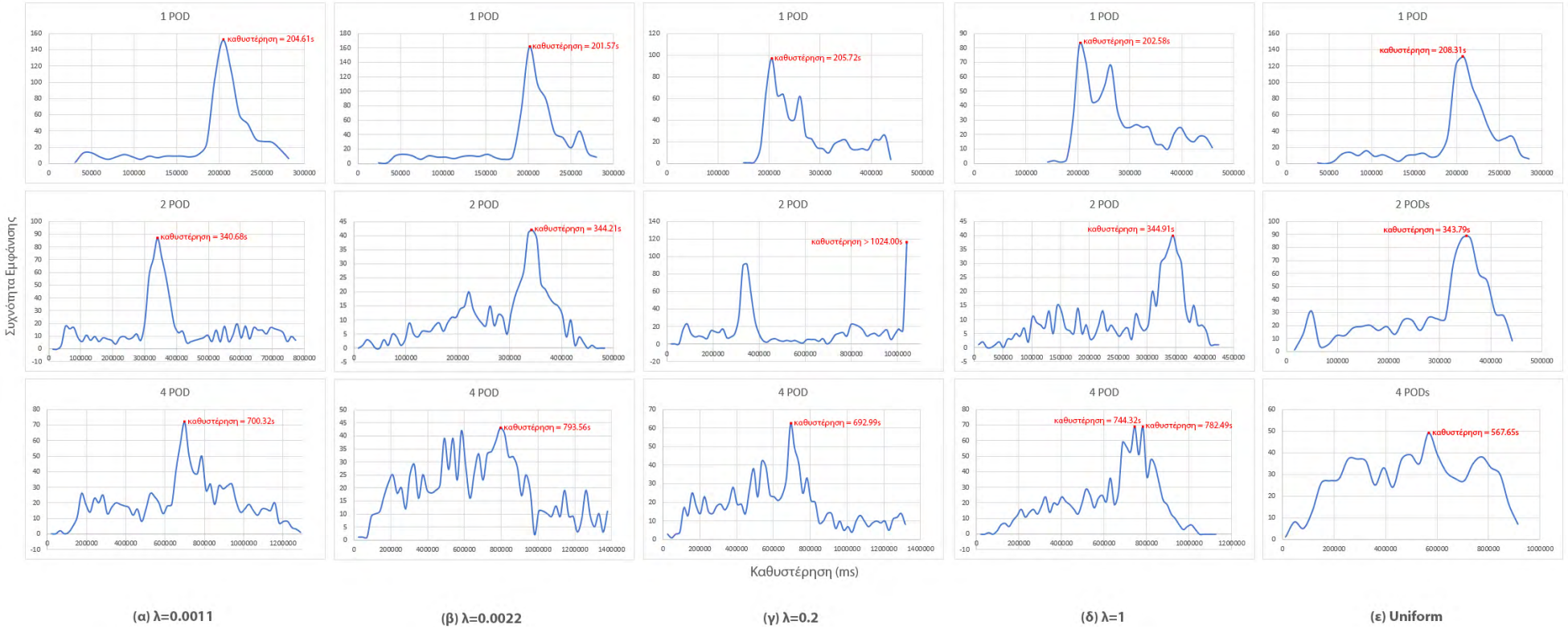


Εικόνα 5.4: Διεκπεραιωτική Ικανότητα ανά Κατανομή

5.1.2.3 Διακύμανση της Καθυστέρησης

Ένα ακόμα αποτέλεσμα το οποίο προέκυψε είναι η διακύμανση της καθυστέρησης (fluctuation of latency). Η διακύμανση της καθυστέρηση περιγράφει τον αριθμό (ποσοτικά) εμφάνισης των τιμών προς καθυστέρηση κατά τη διάρκεια του πειράματος. Στην κορυφή των διαγραμμάτων (Εικόνα 5.5) μπορούμε να δούμε ανά σενάριο τον μέγιστο αριθμό εμφάνισης της καθυστέρησης (άξονας y) καθώς και την τιμή της ίδιας της καθυστέρησης (άξονας x).

Παρατηρήθηκε αύξηση της καθυστέρησης και αντίστοιχα του αριθμού της συχνότητας εμφάνισής της, κατά τον διπλασιασμό των pods ανά σενάριο. Αυτό γίνεται διότι κατά την διαδικασία της δοκιμής, εάν υπάρχουν περισσότερα από ένα pods, τα αιτήματα δεν απορρίπτονται, αλλά περιμένουν να εξυπηρετηθούν σε πάνω από μια ουρές, με αποτέλεσμα, ναι μεν να υπάρχει επιπλέον καθυστέρηση εξυπηρέτησης, αλλά ταυτόχρονα σημαντική μείωση του σφάλματος, το οποίο επιβεβαιώνεται με τα αποτελέσματα που φαίνονται στην Εικόνα 5.3. Αξίζει να παρατηρηθεί η περίπτωση (γ) για λ=0,2 και ειδικότερα το σενάριο των δυο (2) pods. Παρατηρήθηκε κορύφωση προς τον αριθμό εμφάνισης της καθυστέρησης για τιμές καθυστέρησης πολύ μεγαλύτερες του μέσου όρου των υπόλοιπων περιπτώσεων, το οποίο δικαιολογείται από το αποτέλεσμα που περιεγράφηκε παραπάνω κατά το σενάριο των δυο (2) pods για λ=0,2, όπου παρατηρήθηκε σημαντική πτώση της διεκπεραιωτική ικανότητας από την αναμενόμενη.



Εικόνα 5.5: Διακύμανση της Καθυστέρησης

Επιπλέον, παρατηρήθηκε ότι στο σενάριο των τεσσάρων (4) pods για λ=1, τα ανώτατα σημεία ήταν δύο (2) καθώς ο αριθμός της συχνότητας εμφάνισης ήταν ακριβώς ο ίδιος (69 φορές) για τιμές καθυστέρησης ίσες με 744,32 και 782,49 δευτερόλεπτα. Ενώ σε όλες τις υπόλοιπες περιπτώσεις (εξαιρουμένης της περίπτωσης του σεναρίου των δύο (2) pods για λ=0,2) η διαφορά ήταν αναλογικά μικρή, στο σενάριο των

τεσσάρων (4) pods, η μέγιστη διαφορά έφτασε τα 225,91 δευτερόλεπτα. Παρατηρείται λοιπόν εξαιρετική συμπεριφορά στην απόκριση του συστήματος κατά την ομοιόμορφη κατανομή στο σενάριο των τεσσάρων (4) pods.
Μεταξύ των τιμών του λ σε κατανομή Poisson ανά σενάριο αριθμού pods, το σύστημα φάνηκε να αποκρίνεται καλύτερα στο σενάριο των τεσσάρων (4) pods για λ=0,2 όπου η κορύφωση της εμφάνισης προς καθυστέρηση ήταν τα 692,99 δευτερόλεπτα.

5.2 Συμπεράσματα

Οι αυτοματισμοί που διαθέτει και παρέχει ο ενορχηστρωτής Kubernetes καθιστούν ευκολότερη τη διαχείριση και την επίβλεψη μιας ή περισσότερων συστοιχιών, αυξάνοντας παράλληλα την απόδοσή τους με τη χρήση τον ελάχιστων δυνατών πόρων. Μπορεί να χρησιμοποιηθεί ως λύση σε μικρότερα ή μεγαλύτερα συστήματα, προσφέροντας αυτονομία και ταυτόχρονα δυνατότητα κλιμακοθετησιμότητας.
Ο Horizontal Pod Autoscaler αποδεικνύεται από μόνο του, ένα πανίσχυρο εργαλείο διαχείρισης των pods. Σε συνθήκες παραγωγής και με τις σωστές ρυθμίσεις, μπορεί να αποδειχθεί σωτήριο σε μια εταιρία ή έναν οργανισμό. Αυτό βασίζεται κυρίως στην άμεση κατ' απαίτηση διάθεση των πόρων, αφού επεκτείνει αυτόματα και άμεσα τον αριθμό των pods ανάλογα την ζήτηση, ενώ ταυτόχρονα μειώνει τον αριθμό τους εάν δεν χρειάζονται ώστε να τους διαθέσει αλλού για να μην προκληθεί συμφόρηση στο σύστημα. Συγκριτικά με την εργασία των T. Ye et al. [14], δεν εστιάσαμε στον ιδανικό χρόνο απόκρισης όπως προαναφέρθηκε, αλλά στη συνολική βελτίωση της απόδοσης του συστήματος.
Ένα ακόμα συμπέρασμα είναι ότι σε ένα σύστημα και πιο συγκεκριμένα σε σύστημα το οποίο επιδέχεται φόρτο εργασίας χρησιμοποιώντας όλους τους πόρους που του παρέχονται, υφίσταται μια χρυσή τομή ανάμεσα στον αριθμό των pods που εξυπηρετούν τα αιτήματα και στον αριθμό των πόρων που το κάθε pod καταλαμβάνει σε συνάρτηση με το σύνολο των πόρων του κόμβου στον οποίο στεγάζεται. Είναι προφανές ότι περισσότερα pods εξυπηρετούν γρηγορότερα τα αιτήματα που λαμβάνει ο κόμβος. Υπάρχει, όμως, ένα κατώφλι ελάχιστων πόρων προς χρήση ανά pod, το οποίο εάν ξεπεραστεί και οι πόροι υποδιαιρεθούν περαιτέρω, οδηγούμαστε

σε μικρότερη απόδοση όσο αφορά το χρόνο εξυπηρέτησης, παρά το γεγονός ότι υπάρχει μείωση του σφάλματος, η οποία σε αριθμούς μερικών χιλιάδων χρηστών είναι μη υπολογίσιμη.

Τέλος, κατά τη δημιουργία των pods από τον Horizontal Pod Autoscaler, παρατηρήθηκε, λόγω των ρυθμίσεών μας, μια μικρή καθυστέρηση στην προετοιμασία και στη δημιουργία τους. Αυτό οφειλόταν στον τρόπο με τον οποίο είχαμε ορίσει τη δημιουργία τους και αφορά το ποσοστό χρησιμοποίησης της CPU (CPU utilization percentage), το οποίο ήταν 99%. Εάν το ποσοστό αυτό ήταν αρκετά χαμηλότερο τα αποτελέσματα θα διαφοροποιούνταν φέρνοντας ικανοποιητικότερες μετρήσεις και ενδεχομένως να μπορούσε να γίνει σύγκριση με τα αποτελέσματα της έρευνας των M. Amaral et al. [15] όσον αφορά και συγκρίνοντας την απόδοση με σημεία αναφοράς τη λειτουργία της CPU.

Κεφάλαιο 6 - Επίλογος και Μελλοντική Εργασία

6.1 Επίλογος

Στόχος της μεταπτυχιακής εργασίας αυτής ήταν η κατανόηση των υπαρχόντων μεθόδων αποτίμησης απόδοσης σε σύγχρονα υπολογιστικά νέφη και η βελτιστοποίηση της απόδοσης αυτής, μέσω των διαθέσιμων εργαλείων τα οποία παρέχονται, έχοντας ως απώτερο σκοπό την εις βάθος κατανόηση του ενορχηστρωτή Kubernetes και την πιθανή εύρεση λύσεων για μεγαλύτερη απόδοση παράλληλα με τη μείωση χρήσης υπολογιστικών πόρων.

Πραγματοποιήθηκε η υλοποίηση της συστοιχίας Kubernetes, καθώς και η υλοποίηση του εργαλείου δοκιμής φορτίου JMeter, και λήφθηκαν μετρήσεις για την απόδοση της συστοιχίας. Έγινε σύγκριση της προαναφερθείσας απόδοσης ανάλογα τον αριθμό των pods, τα οποία πολλαπλασιάζονταν μέσω της χρήσης του εργαλείου Horizontal Pod Autoscaler.

Η χρήση ενός ενορχηστρωτή, καταλήξαμε ότι βοηθάει στην αυτοματοποίηση και τη δυνατότητα κλιμακοθετησιμότητας ενός συστήματος, ενώ τα εργαλεία του αποτελούν μεγάλο πλεονέκτημα στη διαχείριση ενός ή περισσότερων, μικρών και ειδικά μεγαλύτερων συστημάτων.

6.2 Μελλοντική Εργασία

Η παρούσα διπλωματική εργασία, ανοίγει το δρόμο για μελλοντικές επεκτάσεις και περαιτέρω έρευνα γύρω από τον ενορχηστρωτή Kubernetes αλλά και άλλους ενορχηστρωτές.

Ως προτάσεις για την περαιτέρω συνέχιση της έρευνας, είναι η δημιουργία επιπλέον κόμβου ο οποίος να εκτελεί ίδια ή διαφορετική εργασία και σε συνδυασμό με τον Horizontal Pod Autoscaler να παρατηρηθεί η απόδοσή τους. Έπειτα, μελετώντας την έρευνα των Marathe N. et al.[10], δύναται να προστεθεί ένας εξισορροπητής φορτίου μεταξύ των συστοιχιών ώστε να δούμε πιθανές διαφορές στην απόδοση [16]. Επιπροσθέτως, θα μπορούσε να γίνει, συνδυαστικά με τα παραπάνω, εφαρμογή του Vertical Pod Autoscaler, υπολογίζοντας πλέον αυτόματα τους πόρους που ένα pod χρειάζεται για να φέρει εις πέρας την εργασία, μη ορίζοντας χειροκίνητα τα όρια

των πόρων ανά ομάδα pods. Ακόμα, θα μπορούσε να γίνει σύγκριση απόδοσης μεταξύ ενορχηστρωτών χρησιμοποιώντας τα ανάλογα εργαλεία και μεθόδους. Τέλος, σύμφωνα με τα αποτελέσματα που συζητήθηκαν στο κεφ. 5, εκτιμώ ότι έχει ενδιαφέρον να εξεταστεί η επέκταση της νεφοϋπολογιστικής υποδομής των D. Kallergis et al. [12] ως προς τη χρήση συστοιχίας (cluster) αντί σμήνους (swarm) περιεκτών.

Βιβλιογραφία

[1] Xing, Y., & Zhan, Y. (2012). Virtualization and cloud computing. In Future wireless networks and information systems (pp. 305-312). Springer, Berlin, Heidelberg.

[2] Lewis L., & Fowler, M. (2014). “Microservices” [Online]. Available: https://martinfowler.com/articles/microservices.html. [Accessed: 15-Feb-2022].

[3] Vayghan, L. A., Saied, M. A., Toeroe, M., & Khendek, F. (2018). Deploying microservice based applications with Kubernetes: experiments and lessons learned. In 2018 IEEE 11th international conference on cloud computing (CLOUD), pp. 970-973. IEEE.

[4] Richardson, C. (2020). “What are microservices?” [Online]. Available: https://microservices.io. [Accessed: 15-Feb-2022].

[5] Nguyen, T. T., Yeom, Y. J., Kim, T., Park, D. H., & Kim, S. (2020). Horizontal pod autoscaling in kubernetes for elastic container orchestration. *Sensors*, *20*(16), 4621.

[6] The Kubernetes Authors (2018). “Tools for Monitoring Resources” [Online]. Available: https://kubernetes.io/docs/tasks/debug-application-cluster/resource-usage-monitoring. [Accessed: 15-Feb-2022].

[7] Accenture (2021). “Cloud computing” [Online]. Available: https://www.accenture.com/nz-en/insights/cloud-computing-index. [Accessed: 15-Feb-2022].

[8] Shah, J., & Dubaria, D. (2019). Building modern clouds: using docker, kubernetes & Google cloud platform. In 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), pp. 0184-0189. IEEE.

[9] Pan, Y., Chen, I., Brasileiro, F., Jayaputera, G., & Sinnott, R. (2019). A performance comparison of cloud-based container orchestration tools. In 2019 IEEE International Conference on Big Knowledge (ICBK) (pp. 191-198). IEEE.

[10] Marathe, N., Gandhi, A., & Shah, J. M. (2019). Docker swarm and kubernetes in cloud computing environment. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)* (pp. 179-184). IEEE.

[11] Lei, Q., Liao, W., Jiang, Y., Yang, M., & Li, H. (2019). Performance and scalability testing strategy based on kubemark. In 2019 IEEE 4th International

Conference on Cloud Computing and Big Data Analysis (ICCCBDA) (pp. 511-516). IEEE.

[12] Kallergis, D., Garofalaki, Z., Katsikogiannis, G. and Douligeris, C. (2020). CAPODAZ: A containerised authorisation and policy-driven architecture using microservices. Ad Hoc Networks, 104, p.102153.

[13] Casalicchio, E., & Perciballi, V. (2017). Auto-scaling of containers: The impact of relative and absolute metrics. In *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS* W)* (pp. 207-214). IEEE.

[14] Ye, T., Guangtao, X., Shiyou, Q., & Minglu, L. (2017). An auto-scaling framework for containerized elastic applications. In 2017 3rd international conference on big data computing and communications (BIGCOM) (pp. 422-430). IEEE.

[15] Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M., & Steinder, M. (2015). Performance evaluation of microservices architectures using containers. In *2015 IEEE 14th International Symposium on Network Computing and Applications* (pp. 27-34). IEEE.

[16] Curiel, M., & Pont, A. (2018). Workload generators for web-based systems: Characteristics, current status, and challenges. IEEE Communications Surveys & Tutorials, 20(2), 1526-1546.

[17] Kuipers, L., & Niederreiter, H. (2012). *Uniform distribution of sequences*. Courier Corporation..

[18] Haight, F. A., (1967). Handbook of the Poisson distribution (No. 519.23 H3).

[19] Statistics, I., (2007). "The Poisson Distribution. UMass Amherst" [Online]. Available: https://web.archive.org/web/20140419014445/http://www.umass.edu/wsp/resources/poisson/index.html. [Accessed: 15-Feb-2022].

[20] Apache Software Foundation (2021). "Apache JMeter" [Online]. Available: https://jmeter.apache.org. [Accessed: 15-Feb-2022].

[21] The Kubernetes Authors (2018). "Horizontal Pod Autoscaler" [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale. [Accessed: 15-Feb-2022].

[22] Docker Inc. (2020). “What is a Container?- A standardized unit software” [Online]. Available: https://www.docker.com/resources/what-container. [Accessed: 15-Feb-2022].

[23] Pan, Y., Chen, I., Brasileiro, F., Jayaputera, G., & Sinnott, R. (2019). A Performance Comparison of Cloud-Based Container Orchestration Tools. In 2019 IEEE International Conference on Big Knowledge (ICBK), pp. 191-198. IEEE.

[24] The Kubernetes Authors (2018). “What is Kubernetes? - Kubernetes.” [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes. [Accessed: 15-Feb-2022].

[25] Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M., & Steinder, M. (2015). Performance evaluation of microservices architectures using containers. In 2015 IEEE 14th International Symposium on Network Computing and Applications, pp. 27-34. IEEE.

[26] Rouse, M. (2020). “Application containerization (app containerization)” [Online]. Available: https://searchitoperations.techtarget.com/definition/application-containerization-app-containerization. [Accessed: 15-Feb-2022].

[27] Bernstein, D. (2014). Containers and cloud: From lxc to docker to kubernetes. IEEE Cloud Computing, 1(3), pp. 81-84.

[28] IBM (2021). “Docker” [Online]. Available: https://www.ibm.com/cloud/learn/docker. [Accessed: 15-Feb-2022].

[29] Marathe, N., Gandhi, A., & Shah, J. M. (2019). Docker Swarm and Kubernetes in Cloud Computing Environment. In 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), pp. 179-184. IEEE.

[30] Arvind (2021). “Docker Architecture” [Online]. Available: https://www.aquasec.com/cloud-native-academy/docker-container/docker-architecture. [Accessed: 15-Feb-2022].

[31] Hewlett Packard Enterprise Development LP (2020). “WHAT IS CONTAINER ORCHESTRATION?” [Online]. Available: https://www.hpe.com/us/en/what-is/container-orchestration.html. [Accessed: 15-Feb-2022].

[32] Red Hat Inc. (2019). “What is container orchestration?” [Online]. Available: https://www.redhat.com/en/topics/containers/what-is-container-orchestration. [Accessed: 15-Feb-2022].

[33] Heidari, P., Lemieux, Y., & Shami, A. (2016). Qos assurance with light virtualization-a survey. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)* (pp. 558-563). IEEE.

[34] Simplilearn (2021). “What is Docker Swarm: Modes, Example & Working” [Online]. Available: https://www.simplilearn.com/tutorials/docker-tutorial/docker-swarm. [Accessed: 15-Feb-2022].

[35] Mesosphere, Inc. (2018). “Marathon -A container orchestration platform for Mesos and DC/OS” [Online]. Available: https://mesosphere.github.io/marathon. [Accessed: 15-Feb-2022].

[36] The Kubernetes Authors (2018). “Pods” [Online]. Available: https://kubernetes.io/docs/concepts/workloads/pods. [Accessed: 15-Feb-2022].

[37] Red Hat Inc. (2019). “What is container Kubernetes?” [Online]. Available: https://www.redhat.com/en/topics/containers/what-is-kubernetes. [Accessed: Accessed: 15-Feb-2022].

[38] The Kubernetes Authors (2018). “Kubernetes Components” [Online]. Available: https://kubernetes.io/docs/concepts/overview/components. [Accessed: 15-Feb-2022].

[39] Lei, Q., Liao, W., Jiang, Y., Yang, M., & Li, H. (2019). Performance and Scalability Testing Strategy Based on Kubemark. In 2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), pp. 511-516. IEEE.

[40] The Kubernetes Authors (2018). “Control Plane-Node Communication” [Online]. Available: https://kubernetes.io/docs/concepts/architecture/control-plane-node-communication. [Accessed: 15-Feb-2022].

[41] Medel, V., Rana, O., Bañares, J. Á., & Arronategui, U. (2016). Modelling performance & resource management in Kubernetes. In Proceedings of the 9th International Conference on Utility and Cloud Computing (pp. 257-262).

[42] The Kubernetes Authors (2018). “Workloads” [Online]. Available: https://kubernetes.io/docs/concepts/workloads. [Accessed: 15-Feb-2022].

[43] Casalicchio, E., & Perciballi, V. (2017). Auto-scaling of containers: The impact of relative and absolute metrics. In 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS* W), pp. 207-214. IEEE.

[44] Daniel Berman (2020) . “Kubernetes Monitoring: Best Practices, Methods, and Existing Solutions” [Online]. Available: https://logz.io/blog/kubernetes-monitoring. [Accessed: 15-Feb-2022].

[45] F5 Networks, Inc. (2021). “What is NGINX?” [Online]. Available: https://www.nginx.com/resources/glossary/nginx. [Accessed: 15-Feb-2022].

[46] Apache Software Foundation (2021). “Introduction to listeners” [Online]. Available: https://jmeter.apache.org/usermanual/listeners.html. [Accessed: 15-Feb-2022].

[47] GRNET (2021). “WHAT IS ~OKEANOS?” [Online]. Available: https://okeanos.grnet.gr/support/faq/okeanos-what-is-okeanos. [Accessed: 15-Feb-2022].

[48] Hetzner Online GmbH (2021). “About Us” [Online]. Available: https://www.hetzner.com/unternehmen/ueber-uns. [Accessed: 15-Feb-2022].

Παράρτημα

Υλοποίηση Πειραματικής Διάταξης

Εγκατάσταση Λογισμικού Docker εντός των Διακομιστών

Οι εντολές που παρουσιάζονται παρακάτω, έτρεξαν και στα δύο εικονικά μηχανήματα τα οποία θα στέγαζαν αργότερα την Kubernetes συστοιχία καθώς και στο λογισμικό του JMeter.

Πριν προχωρήσει η εγκατάσταση του Docker, ενημερώθηκε την υπάρχουσα λίστα πακέτων:

```
sudo apt update
```

Για να επιτρέψουμε στο σύστημα να έχει πρόσβαση στα αποθετήρια Docker μέσω HTTPS εκτελέσθηκε η εντολή:

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

Στη συνέχεια προστέθηκε το κλειδί GPG για να διασφαλισθεί η αυθεντικότητα του πακέτου λογισμικού και εγκαταστάθηκε το αποθετήριο Docker:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu  $(lsb_release -cs) stable"
```

Η παραπάνω εντολή εγκαθιστά το πιο πρόσφατο αποθετήριο για τη συγκεκριμένη έκδοση του Ubuntu.

Έπειτα προχωρήσαμε σε μια ακόμα ενημέρωση και εγκαταστήσαμε το Docker, ενεργοποιώντας το να τρέχει κατά την εκκίνηση της μηχανής:

```
sudo apt-get install docker-ce
```

```
sudo systemctl enable docker
```

Τέλος ελέγξαμε εάν έχει εγκατασταθεί και τρέχει ομαλά (Εικόνα 4.6) μέσω της εντολής:

```
sudo systemctl status docker
```

```
root@ubuntu-4gb-hel1-4:~# sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2021-01-07 17:23:20 CET; 1 years 0 months ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 588 (dockerd)
      Tasks: 22
     Memory: 501.2M
     CGroup: /system.slice/docker.service
             ├─   588 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
             └─808729 runc --version
```

Εικόνα 4.6: Έλεγχος Ορθής Λειτουργίας Docker

Εγκατάσταση Λογισμικού Kubernetes

Μετά την εγκατάσταση του Docker στις εικονικές μηχανές, έγινε η εγκατάσταση του λογισμικού Kubernetes όπως και η δημιουργία συστοιχίας μεταξύ αυτών. Η εγκαταστάσεις διαφοροποιούνται ανάμεσα στον Kubernetes master και στον Kubernetes worker.

Πριν προχωρήσει η εγκατάσταση του Kubernetes, ενημερώθηκε την υπάρχουσα λίστα πακέτων και έγινε επανεκκίνηση της μηχανής που θα στεγάσει τους διακομιστές μας:

```
sudo apt update
```

```
sudo systemctl reboot
```

Μετά την ολοκλήρωση της επανεκκίνησης των διακομιστών, έγινε μια επιπλέον ενημέρωση και κατόπιν προστέθηκε το αποθετήριο Kubernetes για τον Ubuntu 20.04 σε όλους τους διακομιστές:

```
sudo apt update

sudo apt -y install curl apt-transport-https

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg |
sudo apt-key add -

echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" |
sudo tee /etc/apt/sources.list.d/kubernetes.list
```

Στη συνέχεια έγινε η εγκατάσταση των πακέτων για τους kubelet, kubeadm και kubectl, επιλέχθηκε να μην γίνονται αυτόματα ενημερώσεις ώστε να αποφύγουμε κάποιο μελλοντικό πρόβλημα σχετικό με τη συμβατότητα και ολοκληρώθηκε με τον έλεγχο της έκδοσης του kubectl ώστε να γίνει βέβαιο το ότι έγινε ορθά η παραπάνω εγκατάσταση (Εικόνα 4.7). Περιμένουμε σαν έξοδο να μας εμφανίσει τις εκδόσεις του Client και του kubeadm:

```
sudo apt update
sudo apt -y install vim git curl wget kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

```
$ kubectl version --client && kubeadm version
```

```
Client Version: version.Info{Major:"1", Minor:"20", GitVersion:"v1.20.1", GitCommit:"c4d752765b3b
bac2237bf87cf0b1c2e307844666", GitTreeState:"clean", BuildDate:"2020-12-18T12:09:25Z", GoVersion:
"go1.15.5", Compiler:"gc", Platform:"linux/amd64"}
kubeadm version: &version.Info{Major:"1", Minor:"20", GitVersion:"v1.20.1", GitCommit:"c4d752765b
3bbac2237bf87cf0b1c2e307844666", GitTreeState:"clean", BuildDate:"2020-12-18T12:07:13Z", GoVersio
n:"go1.15.5", Compiler:"gc", Platform:"linux/amd64"}
```

Εικόνα 4.7: Έλεγχος Έκδοσης του kubectl

Μετά το πέρας των παραπάνω διαδικασιών η εγκατάσταση συνεχίστηκε στον κάθε διακομιστή ξεχωριστά.

Εγκατάσταση στον Kubernetes Master

Μετά τη σύνδεση στον διακομιστή ο οποίος θα αποτελούσε τον Kubernetes master, ενεργοποιήσαμε την υπηρεσία kubectl και έπειτα αρχικοποιήσαμε την μηχανή η οποία θα εκτελεί τα δομικά στοιχεία (Εικόνα 4.9) του επιπέδου ελέγχου όπως τον API-Server και τη βάση δεδομένων της συστοιχίας (etcd):

```
sudo systemctl enable kubelet
```

```
sudo kubeadm config images pull
```

```
[config/images] Pulled k8s.gcr.io/kube-apiserver:v1.20.14
[config/images] Pulled k8s.gcr.io/kube-controller-manager:v1.20.14
[config/images] Pulled k8s.gcr.io/kube-scheduler:v1.20.14
[config/images] Pulled k8s.gcr.io/kube-proxy:v1.20.14
[config/images] Pulled k8s.gcr.io/pause:3.2
[config/images] Pulled k8s.gcr.io/etcd:3.4.13-0
[config/images] Pulled k8s.gcr.io/coredns:1.7.0
```

Εικόνα 4.8: Στοιχεία Εξόδου Εικόνων Δομικών Στοιχείων Συστοιχίας

Για τη δημιουργία της συστοιχίας και τη διάρθρωση του kubectl, καθώς και για να μπορεί ο τοπικός χρήστης να εκτελέσει τα αρχεία διαμόρφωσης (configuration files) με σκοπό να μπορεί να αλληλεπιδρά με τη συστοιχία, εκτελέστηκαν οι εντολές:

```
sudo kubeadm init \
  --pod-network-cidr=192.168.0.0/16 \
  --upload-certs
```

```
mkdir -p $HOME/.kube
sudo cp -f /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Όσον αφορά την εγκατάσταση του δικτυακού πρόσθετου (plugin) χρησιμοποιήθηκε το Calico με σκοπό να έχουμε ένα λειτουργικό δίκτυο στη συστοιχία:

```
kubectl create -f https://docs.projectcalico.org/manifests/tigera-operator.yaml
kubectl create -f https://docs.projectcalico.org/manifests/custom-resources.yaml
```

Η συστοιχία μας σε αυτό το σημείο είναι έτοιμη.

Για τη δημιουργία λογαριασμού διαχειριστή, αρχικά δημιουργήθηκε ένα αρχείο δήλωσης λογαριασμού υπηρεσιών (Service Account manifest file):

```
vim admin-sa.yml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: mike
  namespace: kube-system
```

```
kubectl apply -f admin-sa.yml
serviceaccount/mike created
clusterrolebinding.rbac.authorization.k8s.io/mike created
```

Ορίστηκε ο λογαριασμός αυτός ως διαχειριστής της συστοιχίας δημιουργώντας και εκτελώντας το αρχείο, παράλληλα δίνοντας τη δυνατότητα να μπορεί ο διαχειριστής αυτός να αποκτήσει τα δικαιώματα του υπερχρήστη στη συστοιχία αλλά και να δημιουργεί token για την πρόσβαση στον πίνακα ελέγχου του Kubernetes:

```
vim admin-rbac.yml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: mike
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: mike
    namespace: kube-system
```

```
kubectl apply -f admin-rbac.yml
```

Αν και ο πίνακας ελέγχου δημιουργήθηκε, δεν θα αναλυθεί η διαδικασία δημιουργίας του καθώς δεν αποτέλεσε σημείο-κλειδί για την πειραματική διαδικασία πέρα από την οπτικοποίηση της λειτουργίας της συστοιχίας.

Σύνδεση του Kubernetes Worker με τη Συστοιχία

Αρχικά, δημιουργήθηκε ένα token το οποίο παρέμεινε ενεργό για 24 ώρες και χρησιμοποιήθηκε για την ένωση του Kubernetes worker με τη συστοιχία:

```
sudo kubeadm token create
```

```
kubeadm token list
```

Για να βρεθεί το token, η εντολή join επικυρώνει το δημόσιο κλειδί ρίζας CA, αντιστοιχίζοντας τον κατακερματισμό (hash) του με αυτό που παρέχεται. Η εντολή αυτή εκτελέστηκε στο επίπεδο ελέγχου:

```
openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -
pubin -outform der 2>/dev/null | openssl dgst -sha256 -hex | sed
's/^.* //'
```

```
kubectl cluster-info
```

Τρέχοντας την παραπάνω εντολή, βλέπουμε ότι Kubernetes master τρέχει στην διεύθυνση https://95.217.214.130:6443. Επομένως για να ενωθεί ο worker και έχοντας ότι χρειαζόταν εκτελέστηκε η εντολή με την μορφή:

```
kubeadm join \
  https://95.217.214.130:6443\
  --token <token> \
  --discovery-token-ca-cert-hash sha256:<hash>
```

Εγκατάσταση NGIX

Αρχικά, δημιουργήθηκε μια ανάπτυξη του NGINX χρησιμοποιώντας την εικόνα (image) του NGINX:

```
kubectl create deployment nginx --image=nginx
```

Επειδή η συστοιχία διαθέτει δημόσια και ιδιωτική διεύθυνση IP, το Kubernetes θα εκχωρήσει αυτήν την υπηρεσία σε θύρα στην περιοχή άνω των 30000 ώστε να είναι πλέον εφικτή η πρόσβαση στην υπηρεσία μέσω διαδικτύου (Εικόνα 4.10):

```
kubectl create service nodeport nginx --tcp=80:80
```

```
mike@ubuntu-4gb-hel1-4:/tmp/kubernetes-ingress/deployments$ kubectl get service
-n nginx-ingress
NAME            TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)
       AGE
nginx-ingress   NodePort   10.111.204.174   <none>        80:30658/TCP,443:30304
/TCP   3m36s
mike@ubuntu-4gb-hel1-4:/tmp/kubernetes-ingress/deployments$
```

Εικόνα 4.9: Τυπωμένα Στοιχεία του NGINX-Ingress

Έχοντας πλέον έτοιμη τη διεύθυνση που θα τρέξει η εργασία, με IP αυτή του Kubernetes master και πόρτα αυτή του NGINX: http://95.217.214.130:30233.

Τέλος, για να «τρέξει» η εργασία που δημιουργήσαμε ώστε όταν ένας χρήστης στείλει αίτημα στην προαναφερθείσα διεύθυνση, παραμετροποιήσαμε το αρχείο nginx-deployment.yaml. Το αρχείο αυτό είναι υπεύθυνο να παραπέμπει το χρήστη στην σελίδα καλωσορίσματος της εφαρμογής του NGINX εντός της συστοιχίας.

Παρακάτω βλέπουμε πως διαμορφώθηκε μετά την μορφοποίηση:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx
 labels:
   app: nginx
spec:
 replicas: 1
 selector:
   matchLabels:
     app: nginx
 template:
   metadata:
     labels:
       app: nginx
   spec:
     containers:
     - name: nginx
       image: georgsatyros/kolastirion-app
       ports:
       - containerPort: 80
```

Οι ρέπλικες παρέμειναν μία σε αριθμό, και εντός του περιέκτη προστέθηκε η εικόνα της εργασίας καθώς και η πόρτα που τη συνδέει με το pod.

Εγκατάσταση του Metrics Server και Δημιουργία του Horizontal Pod Autoscaler

Για να μπορεί να λειτουργήσει ο Horizontal Pod Autoscaler αρχικά θα πρέπει να είναι εγκατεστημένος ο Metrics Server.

Έγινε λήψη του manifest αρχείου μέσω της εντολής wget και αφ' εταίρου εκτελέσθηκε και ελέγχθηκε ώστε να βεβαιωθεί ότι λειτουργεί σωστά:

```
wget https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml -O metrics-server-components.yaml
```

```
kubectl apply -f metrics-server-components.yaml
```

```
kubectl get deployment metrics-server -n kube-system
```

```
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
metrics-server   1/1     1            1           7m23s
```

Εικόνα 4.10: Έλεγχος Ορθής Λειτουργίας του Metrics Server

Αφού ολοκληρώθηκε η εγκατάσταση του Metrics Server, σειρά πήρε η δημιουργία του Horizontal Pod Autoscaler. Για να επιτευχθεί αυτό όπως και η σωστή λειτουργία του, αρχικά έπρεπε να γίνει μορφοποίηση του αρχείου nginx-deployment.yaml προσθέτοντας τα όρια γύρω από τη χρήση των πόρων ανά pod. Χωρίς την αποτύπωση των κατώτατων και ανώτερων ορίων όσον αφορά τη διαθεσιμότητα των πόρων στο pod, ο Horizontal Pod Autoscaler δεν μπορεί να δώσει εντολή ανάπτυξης νέων pods:

```
cd /tmp/kubernetes-ingress/deployments
```

```
nano nginx-deployment.yaml
```

Το αρχείο πήρε τη μορφή, όπως φαίνεται παρακάτω, αρχικά για τα δυο pods και στη συνέχεια μορφοποιήθηκε εκ νέου όταν εκτελέσαμε το πείραμα για τα τέσσερα pods με τα αντίστοιχα όρια (Πίνακας: 4.1):

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: georgsatyros/kolastirion-app
        resources:
          requests:
            cpu: "250m"
          limits:
            cpu: "500m"
        ports:
        - containerPort: 80
```

Μετά την αποθήκευση του αρχείου εκτελέσθηκε ώστε να καταχωρηθούν οι αλλαγές και τέλος ορίστηκε ο ελάχιστος και μέγιστος αριθμός των pods τα οποία θα υπάρχουν διαθέσιμα για τη συγκεκριμένη εργασία (Πίνακας 4.1).

```
kubectl apply -f nginx-deployment.yaml
```

```
kubectl autoscale deployment nginx --cpu-percent=99 --min=1 --max=2
```

```
kubectl autoscale deployment nginx --cpu-percent=99 --min=1 --max=4
```

```
kubectl get hpa
```

Με τη χρήση διάφορων εντολών παρακολουθήθηκε η συμπεριφορά των pods κατά τη διάρκεια του πειράματος, όπως το πόσα ενεργά pods υπάρχουν τη συγκεκριμένη χρονική στιγμή και τι πόρους καταναλώνουν (Εικόνα 4.11):

```
kubectl get pods
```

```
kubectl top pods
```

```
mike@ubuntu-4gb-hel1-4:/tmp/kubernetes-ingress/deployments$ kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
nginx-86f85954b8-mq2m7   1/1     Running   0          21m
nginx-86f85954b8-wxvpl   1/1     Running   0          14m
mike@ubuntu-4gb-hel1-4:/tmp/kubernetes-ingress/deployments$ kubectl top pods
NAME                     CPU(cores)   MEMORY(bytes)
nginx-86f85954b8-mq2m7   500m         30Mi
nginx-86f85954b8-wxvpl   1m           20Mi
mike@ubuntu-4gb-hel1-4:/tmp/kubernetes-ingress/deployments$
```

```
mike@ubuntu-4gb-hel1-4:/tmp/kubernetes-ingress/deployments$ kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
nginx-86f85954b8-mq2m7   1/1     Running   0          8m48s
nginx-86f85954b8-wxvpl   1/1     Running   0          116s
mike@ubuntu-4gb-hel1-4:/tmp/kubernetes-ingress/deployments$ kubectl top pods
NAME                     CPU(cores)   MEMORY(bytes)
nginx-86f85954b8-mq2m7   500m         28Mi
nginx-86f85954b8-wxvpl   500m         20Mi
mike@ubuntu-4gb-hel1-4:/tmp/kubernetes-ingress/deployments$
```

```
mike@ubuntu-4gb-hel1-4:/root$ kubectl get pods
NAME                     READY   STATUS        RESTARTS   AGE
nginx-5cf975445b-84tfh   1/1     Running       0          16h
nginx-5cf975445b-c89bd   1/1     Terminating   0          26m
nginx-5cf975445b-mcmw4   1/1     Running       0          27m
mike@ubuntu-4gb-hel1-4:/root$
```

Εικόνα 4.11: Παρακολούθηση των Pods κατά τη Διάρκεια Πειράματος

Εγκατάσταση Λογισμικού JMeter

Κατόπιν της εγκατάστασης του Ubuntu Server όπως περιεγράφηκε παραπάνω, ξεκίνησε η διαδικασία εγκατάστασης του JMeter.

Αρχικά, έγινε λήψη και εγκατάσταση των αρχείων του OpenJDK (έκδοση 8). Το OpenJDK (Open Java Development Kit) είναι μια δωρεάν εφαρμογή ανοιχτού κώδικα της Java πλατφόρμας, Standard Edition (Java SE):

```
sudo apt-get install openjdk-8-jdk
```

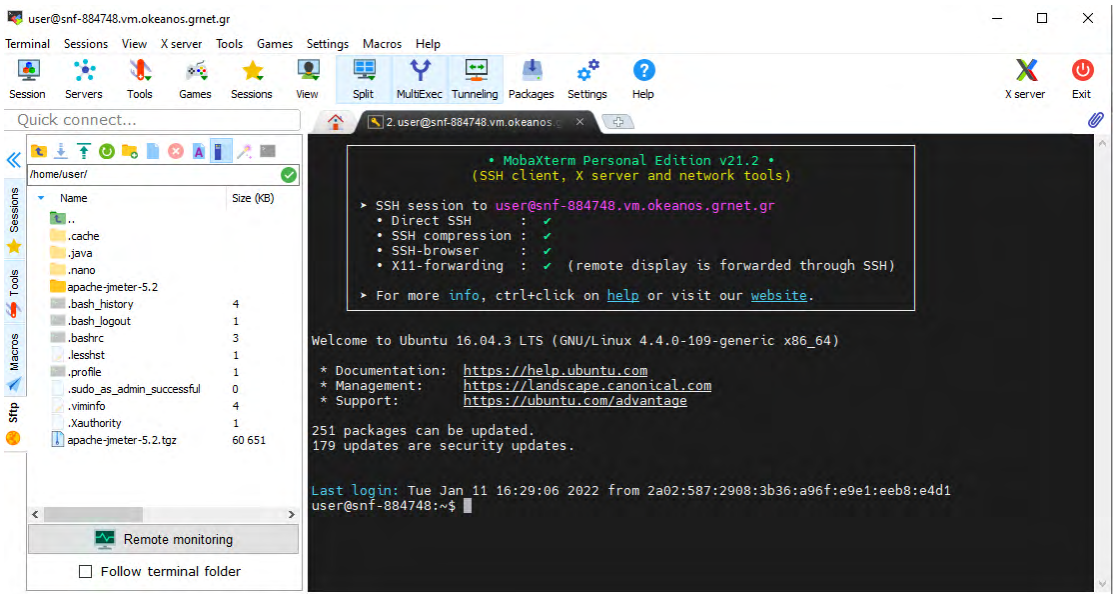
Συνεχίσαμε με τη λήψη και εγκατάσταση των αρχείων του JMeter (έκδοση 5.2), τοπικά στον server:

```
wget https://www.apache.org/dist//jmeter/binaries/apache-jmeter-5.2.tgz
```

Έπειτα δημιουργήθηκε κατάλογος με τα αρχεία (directory) εντός της τοποθεσίας που επιλέχθηκε:

```
tar -xvzf  apache-jmeter-5.2.tgz
```

Για να οριστούν οι παράμετροι του πειράματος, προτιμήθηκε η χρήση γραφικού περιβάλλοντος μέσω του πρωτόκολλου «Windows X System», γνωστό και ως «X11», το οποίο είναι υπεύθυνο για τη δημιουργία «παραθύρων απεικόνισης» στην οθόνη ενός υπολογιστή. Η λύση που προτάθηκε και κατ' επέκταση χρησιμοποιήθηκε, ήταν η εγκατάσταση του προγράμματος «MobaXterm» τοπικά στον υπολογιστή του διαχειριστή (Εικόνα 4.12).



Εικόνα 4.12: Απεικόνιση του MobaXterm

Μετά την ολοκλήρωση στο σύνολο των εγκαταστάσεων, σειρά πήρε η δημιουργία αρχείου «.jmx», με σκοπό την επιλογή των παραμέτρων για τη δοκιμή φορτίου.

Το JMeter εκτελείται εντός του «bin», μέσω της εντολής:

```
./jmeter
```

Στο σημείο αυτό δημιουργήθηκε και αποθηκεύτηκε το εκτελέσιμο αρχείο το οποίο θα έκανε τη δοκιμή φορτίου στη συστοιχία.

Για να εκτελεστεί το προαναφερθέν αρχείο, τρέξαμε την εντολή:

```
./jmeter -n -t /home/user/apache-jmeter-5.2/loadtesting.jmx -l
/home/user/apache-jmeter-5.2/results.csv
```

Η παραπάνω εντολή εκτελεί το αρχείο «loadtesting.jmx» και αποθηκεύει τα αποτελέσματα στο αρχείο «results.csv», από όπου και πήραμε τα τελικά αποτελέσματα του πειράματος ανά σενάριο.