Πανεπιστήμιο Δυτικής Μακεδονίας
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Κατεύθυνση Δικτύων και Τηλεπικοινωνιών

# Συγκριτική μελέτη συστημάτων ενορχήστρωσης πόρων συμβατά με το περιβάλλον Kubernetes

Διπλωματική Εργασία
του
Εμμανουήλ Σκουλαρίκη

**Επιβλέπων:** Καθηγητής Παναγιώτης Σαρηγιαννίδης

21 Οκτωβρίου 2024

University of Western Macedonia
Faculty of Engineering
Department of Electrical and Computer Engineering
Specialization in Computer Networks and Telecommunications

# A comparative study of Kubernetes compliant resource orchestrators

Diploma Thesis
of
Emmanouil Skoularikis

**Supervisor:** Professor Panagiotis Sarigiannidis

21 Οκτωβρίου 2024

# Δήλωση Πνευματικών Δικαιωμάτων

**Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο**

**“**_____

_____

_____

_____

_____**”**

**καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κ.** _____

**αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.**



**Υπογραφή Φοιτητή:**

**Περίληψη**

Η σύγχρονη εποχή χαρακτιρίζεται από την αυξανόμενη χρήση των containerized εφαρμογών. Αυτές οι εφαρμογές εκτελούνται σε μεμονωμένα πακέτα κώδικα, που ονομάζονται containers. Τα containers περιλαμβάνουν βιβλιοθληκες, δυαδικά (binary) αρχεία, αρχεία διαμόρφωσης, αναγκαία για την εκτέλεση της εφαρμογής σε οποιοδήποτε λειτουργικό σύστημα με αυτά να συσκευάζονται σε ένα ελαφρύ εκτελέσιμο αρχείο. Η χρήση των containerized εφαρμογών αυξάνεται με γεωμτρικούς ρυθμούς, δημιουργώντας την ανάγκη για μία πιο αποτελεσματική, επεκτάσιμη και ασφαλής διαχείριση των εφαρμογών αυτών.

Η λύση στο παραπάνω πρόβλημα δόθηκε με το πρότυπο Kubernetes, μία ανοιχτού κώδικα (open source) τεχνολογία που αρχικά αναπτύχθηκε από την Google και έπειτα υιοθετήθηκε από το CNCF (Cloud Native Computing Foundation). Το Kubernetes αυτοματοποιεί την ανάπτυξη, την κλιμάκωση και την διαχείρηση των containerized εφαρμογών. Αποτελεί πρότυπο για την ενορχήστρωση των containers σε νεφό-τοπικά (cloud-native) περιβάλλοντα, λόγω της στιβαρής αρχιτεκτονικής του, της ευλιξίας του αλλά και του εκτενούς οικοσυστηματός του. Κάθε οργανισμός έχει διαφορετικές ανάγκες, είτε είναι μία μικρή επιχείρηση ή κάποιο τοπικό περιβάλλον δοκιμών είτε μία μεγάλη εταιρεία με σύνθετες υποδομές. Εξαιτίας αυτών των διαφορετικών αναγκών, αναπτύχθηκαν ενορχηστρωτές container συμβατοί με το Kubernetes για να προσφέρουν τις κατάλληλες λύσεις για τις ανάγκες κάθε οργανισμού. Κάποιοι από αυτούς τους ενορχηστρωτές είναι οι K3S, Microk8s και το Vanilla Kubernetes [1].

Στόχος της διπλωματικής εργασίας είναι η συγκριτική μελέτη των ενοχρηστρωτών αυτών και η αξιολόγησή τους βάση των πόρων που καταναλώνουν. Με αποτέλεσμα να βρεθεί ο κατάλληλος για τις ανάγκες του κάθε οργανισμού.

**Abstract**

The modern era is characterized by the increasing use of containerized applications. These applications run in isolated code packages, called containers. Containers include libraries, binary files and configuration files required for an application to run on any operating system, packaged into a lightweight executable file. The use of containerized applications is growing exponentially, creating the need for more efficient, scalable, and secure management of these applications.

The solution to this challenge was provided by the Kubernetes standard, an open-source technology initially developed by Google and later adopted by the CNCF (Cloud Native Computing Foundation). Kubernetes automates the deployment, scaling, and management of containerized applications. We can say it has become the standard for container orchestration in cloud-native environments, due to its robust architecture, flexibility, and extensive ecosystem. Every organization has different needs, either it's a small business or a local testing environment, or a large company with complex infrastructures. Due to these varying needs, Kubernetes-compatible orchestrators were developed to provide tailored solutions for each organization. Some of these orchestrators are K3S, Microk8s, and vanilla Kubernetes.

This thesis aims to conduct a comparative study of these orchestrators and evaluate them, based on the consumed resources and how they manage the available resources on a deployed application, ultimately determining the most suitable for the needs of each organizations.

# Ευχαριστίες

Πρώτα από όλα, θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στην οικογένειά μου για την αδιάκοπη υποστήριξη που μου προσέφεραν καθ' όλη τη διάρκεια της ακαδημαϊκής μου πορείας. Ακολούθως, ευχαριστώ από καρδιάς τους φίλους και τους ανθρώπους που συνάντησα, για τις πολύτιμες εμπειρίες και τις όμορφες στιγμές που μοιραστήκαμε.

Επιπλέον, θα ήθελα να ευχαριστήσω τον κ. Παναγιώτη Σαρηγιαννίδη για την εμπιστοσύνη που μου έδειξε και για την καθοδήγηση κατά την εκπόνηση της διπλωματικής μου εργασίας. Τέλος, θα ήθελα ιδιαιτέρως να εκφράσω την ευγνωμοσύνη μου στους συμφοιτητές μου και στον Υποψήφιο Διδάκτορα του Πανεπιστημίου Δυτικής Μακεδονίας, Αθανάσιο Λιατίφη, καθώς και στον Μεταδιδάκτορα Δημήτριο Πλιάτσιο για τη βοήθεια και την υποστήριξη που μου παρείχαν.

# Συγκριτική μελέτη συστημάτων ενορχήστρωσης πόρων συμβατά με το περιβάλλον Kubernetes

Εμμανουήλ Σκουλαρίκης
ece01393@uowm.gr

21 Οκτωβρίου 2024

# Περιεχόμενα

# Κεφάλαιο 1

# Εισαγωγή

## 1.1 Ενορχήστρωση Πόρων

Η ενορχήστρωση container αυτοματοποιεί και απλοποιεί την εκτέλεση και τη διαχείριση των containerized εφαρμογών. Το Kubernetes είναι το πιο γνωστό εργαλείο για την εκτέλεση και τη διαχειρίση τους [2]. Οι περισσότεροι ενορχηστρωτές υποστηρίζουν το δηλωτικό (declarative) μοντέλο διαμόρφωσης, στο οποίο ο χρήστης δημιουργεί ένα αρχείο διαμόρφωσης είτε σε μορφή yaml είτε σε μορφή json καθορίζοντας την επιθυμητή κατάσταση. Στη συνέχεια ο ενορχηστρωτής εκτελεί αυτό το αρχείο για να πετύχει την επιδιωκόμενη κατάσταση. Το αρχείο αυτό συνήθως αποτελείται από τις εξής παραμέτρους:

- Το image της εφαρμογής και τον τόπο λήψης του.
- Τον αποθηκευτικό χώρο και άλλους πόρους.
- Την δικτύωση μεταξύ των container.
- Τις εκδόσεις των διάφορων στοιχείων που περιέχει το Kubernetes.

Ο ενορχηστρωτής επιλέγει τον κατάλληλο host, αναλόγως τους διαθέσιμους πόρους, είτε την κεντρική μονάδα επεξεργασίας (Central Processing Unit - CPU) είτε την μνήμη τυχαίας προσπέλασης (Random Access Memory - RAM), είτε το δίκτυο (Network). Αφότου εκτελεστούν τα container και είναι λειτουργικά, ο ενορχηστρωτής διαχειρίζεται τον κύκλο ζωής τους (lifecycle) [3]. Αυτο σημαίνει πως, διαχειρίζεται την κλιμάκωση των container, την εξισορρόπηση φορτίου (load balancing) και την κατανομή των πόρων μεταξύ των container. Σε περίπτωση βλάβης, διασφαλίζει την διαθεσιμότητα και την απόδοση των container. Τέλος, συλλέγει και αποθηκεύει τα αρχεία καταγραφών τα οποία χρησιμοποιούνται για την παρακολούθηση της ομαλής λειτουργίας της εφαρμογής.

Το Kubernetes προσφέρει πολλές λειτουργίες, μερικές απο αυτές είναι, η εκτέλεση (deploy) των container σε έναν κόμβο και η διατήρηση της επιθυμητής καταστασής τους [4]. Η επανακυκλοφορία ενός deplpoyment μετά από κάποιες αλλαγές στη διαμορφωποιησή του, ή η παύση και η επανεκτελεσή του. Η εύρεση υπηρεσιών (service discovery), δηλαδή, το Kubernetes εκθέτει ένα container στο διαδύκτιο ή σε άλλα container χρησιμοποιώντας ένα όνομα Συστήματος Ονοματοδοσίας Διαδικτύου (Domain Name System - DNS) ή μία διεύθυνση Διαδικτυακού Πρωτοκόλλου (Internet Proto-

col address - IP address). Παρέχει, χώρο αποθήκευσης στα container, εξισσοροπεί και διανέμει το φορτίο κατάλληλα, αποφεύγοντας την μείωση της απόδοσης και της σταθερότητας του συστήματος. Τέλος, αν ένα container αποτύχει, δηλαδή διακοπεί η λειτουργία του ακούσια, το Kubernetes έχει τη δυνατότητα να το επανεκκινήσει, να το αντικαταστήσει ή ακόμα και να διακόψει τη λειτουργία του αν δεν συναντάει τις απαραίτητες απαιτήσεις.

## 1.2 Κίνητρα και Στόχοι

Στόχος της παρούσας διπλωματικής εργασίας είναι η σύγκριση ενορχηστρωτών contαiner συμβατών με το πρότυπο Kubernetes. Για τη σύγκριση χρησιμοποιήθηκε το Kubeflow, μία υψηλών απαιτήσεων containerized εφαρμογή.

Τα πορίσματα βασίστηκαν σε μετρικές που συλλέχθηκαν κατά τη διάρκεια των δοκιμών. Οι μετρικές αυτές είναι, η χρήση πόρων του επεξεργαστή και μνήμης της υπολογιστικής συστοιχίας (cluster), ένα ή παραπάνω υπολογιστικά μηχανήματα, των πόρων επεξεργαστή και μνήμης του κάθε κόμβου (node) και τέλος, των πόρων επεξεργαστή, μνήμης και δικτυακής κίνησης του χώρου ονοματοδοσίας (namespace της εφαρμογής [5].

Με το πέρας της σύγκρισης καθίσταται σαφές ποιος ενορχηστρωτής είναι ο ιδανικός για κάθε οργανισμό αναλόγως την κλιμακά του, πετυχαίνοντας με αυτόν τον τρόπο την καλύτερη δυνατή διαχείρηση των containerized εφαρμογών και των διαθέσιμων πόρων του.

## 1.3 Δομή και Σύνοψη Έργασίας

Η διπλωματική εργασία ως σύνολο αποτελείται από έξι κεφάλαια. Το κεφάλαιο 2, όπου επικεντρώνεται αρχικά στο πως εκτελούνται παραδοσιακά οι εφαρμογές και ποιοί περιορισμοί υπάρχουν, και μετέπειτα στην εξέλιξη αυτών, τις εικονικές μηχανές (Virtual Machines - VMs). Στη συνέχεια γίνεται μία σύντομη εισαγωγή στα containers και στους ενορχηστρωτές container και τέλος αναφέρονται οι προκλήσεις και τα ζητήματα της ενορχήστρωσης container.

Στο κεφάλαιο 3 παρουσιάζεται η αρχιτεκτονική του ενορχηστρωτή container Kubernetes, η δικτύωση του και ενορχηστρωτές συμβατοί με το Kubernetes.

Στο κεφάλαιο 4 γίνεται η σύγκριση των ενορχηστρωτών αυτών και παρουσιάζεται η πειραματική τοπολογία των πειραμάτων που εκτελέστηκαν. Στη συνέχεια γίνεται αναφορά στην εφαρμογή που χρησιμοποιήθηκε και τέλος πως παραμετροποιήθηκαν οι ενορχηστρωτές.

Στο κεφάλαιο 5 βρίσκονται τα αποτελέσματα των ληφθέντων μετρήσεων, ενώ στο κεφάλαιο 6 προβάλονται τα συμπεράσματα και οι μελλοντικές επεκτάσεις του έργου.

# Κεφάλαιο 2

# Ενορχήστρωση Πόρων σε Νεφο-κεντρικές Εφαρμογές

## 2.1 Ιστορική Αναδρομή

Πολλοί σύγχρονοι υπολιγστικοί φόρτοι εργασίας εκτελούνται σε περιβάλλοντα εκμίσωθσης υπολογιστικών πόρων σε πολλαπλούς χρήστες και επιτρέπουν τη δημιουργία εικονικών μηχανών και container, όπου κάθε φυσική μηχανή χωρίζεται σε εκατοντάδες ή χιλιάδες μικρότερες μονάδες υπολογιστών, που ονομάζονται guests. Το cloud και τα container αποτελούν την πρωταρχική επιλογή των οργανισμών. Οι guests σε ένα cloud περιβάλλον, ονομάζονται συνήθως εικονικές μηχανές, ενώ οι guests σε περιβάλλον εκτέλεσης container ονομάζονται containers. Συνήθως, ένας χρήστης ή ομάδα χρηστών έχει τη δυνατότητα να δημιουργήσει και να χρησιμοποιήσει τους guests με ενορχηστρωμένο τρόπο είτε στο Cloud είτε σε ένα σύμπλεγμα που αποτελείται από εκατοντάδες ή χιλιάδες φυσικές μηχανές που βρίσκονται στο ίδιο κέντρο δεδομένων ή σε πολλά κέντρα δεδομένων, διευκολύνοντας τη λειτουργικότητα σε τομείς όπως ο σχεδιασμός χωρητικότητας, η ανθεκτικότητα και η αξιόπιστη απόδοση υπό μεταβλητό φορτίο. Κάθε guest έχει το δικό του λειτουργικό σύστημα και εκτελεί τις δικές του εφαρμογές έχοντας την ψευδαίσθηση ότι είναι φυσικό μηχάνημα, τόσο για τους τελικούς χρήστες που αλληλεπιδρούν με τις υπηρεσίες της εικονικής μηχανής όσο και για τους προγραμματιστές που είναι σε θέση να δημιουργήσουν αυτές τις υπηρεσίες χρησιμοποιώντας γλώσσες προγραμματισμού, βιβλιοθήκες και λειτουργίες λειτουργικού συστήματος. Η ψευδαίσθηση, ωστόσο, δεν είναι τέλεια, γιατί τελικά οι εικονικές μηχανές μοιράζονται τους πόρους υλικού (CPU, μνήμη, κρυφή μνήμη, συσκευές) του υποκείμενου φυσικού υπολογιστή και, κατά συνέπεια, έχουν πρόσβαση στο λογισμικό του κεντρικού υπολογιστή είτε στο kernel είτε στο λειτουργικό σύστημα σε αντίθεση με μια διαφορετική υπολογιστική μηχανή [6].

Ιδανικά, οι οργανισμοί που προσφέρουν περιβάλλοντα εκμίσθωσης υπολογιστικών πόρων θα παρείχαν ισχυρή απομόνωση του guest από τον host και μεταξύ των guests από τον ίδιο host, αλλά αυτή δεν είναι η πραγματικότητα. Οι προσεγγίσεις που έχουν χρησιμοποιήσει διάφορες υλοποιήσεις για την απομόνωση των guests έχουν διαφορετικά πλεονεκτήματα και μειονεκτήματα. Για παράδειγμα, τα container μοιράζονται το kernel του υπολογιστικού μηχανήματος που εκτελούνται, ενώ οι εικονικές μηχανές εκτελούνται ως διεργασία στο λειτουργικό σύστημα εκθέτοντας διαφορετικές επιφάνειες

επίθεσης μέσω διαφορετικών διαδρομών στο λειτουργικό σύστημα . Ωστόσο, όλες οι υπάρχουσες υλοποιήσεις εικονικών μηχανών και container εκθέτουν περισσότερο το χαμηλότερο επίπεδο λογισμικού και το υλικό πόρων από όσο είναι απαραίτητο.

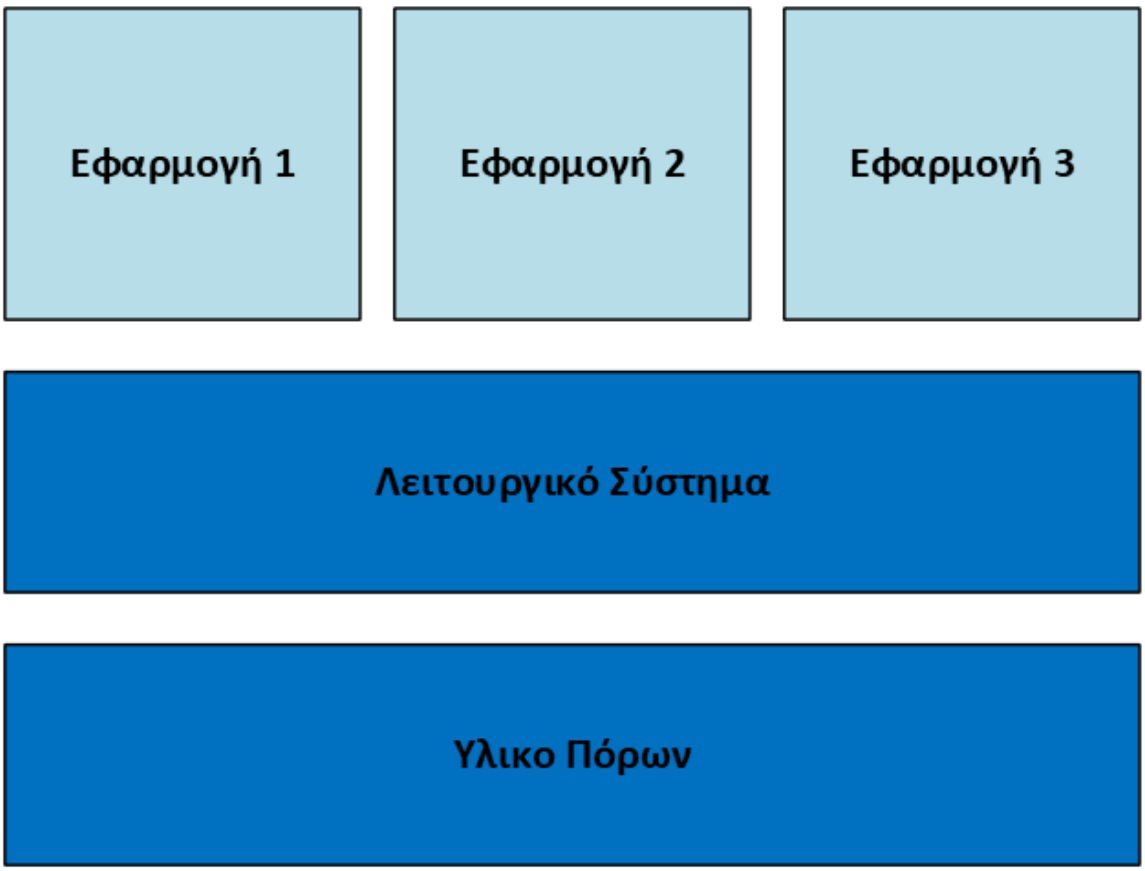
Επειδή η χρήση των περιβάλλοντων αυτών έχει αποδειχθεί χρήσιμη και κερδοφόρα για έναν μεγάλο τομέα της πληροφορικής βιομηχανίας, είναι πιθανό ότι ένα σημαντικό ποσοστό υπολογιστικού φόρτου εργασίας θα συνεχίσει να εκτελείται σε τέτοια περιβάλλοντα για το άμεσο μέλλον. Οι εταιρείες που παρέχουν και χρησιμοποιούν τέτοια περιβάλλοντα έχουν πλήρη επίγνωση των κινδύνων ασφάλειας, αλλά το προτιμούν ούτως ή άλλως επειδή τα οφέλη,όπως η ευελιξία, η ανθεκτικότητα, η αξιοπιστία, η απόδοση, το κόστος ή οποιοσδήποτε άλλος παράγοντας, αντισταθμίζουν τους κινδύνους για τις επιχειρηματικές ανάγκες τους.

Η προέλευση των εικονικών μηχανών και των container εντοπίζεται, σε μια θεμελιώδη αλλαγή των αρχιτεκτονικών υλικού πόρων και λογισμικού, στα τέλη της δεκαετίας του 1950. Το υλικό πόρων εκείνης της εποχής εισήγαγε την έννοια του πολυπρογραμματισμού, που περιλάμβανε τόσο τη βασική πολλαπλή εργασία με τη μορφή απλής εναλλαγής περιβάλλοντος όσο και τη βασική πολυεπεξεργασία με τη μορφή αποκλειστικών επεξεργαστών Input/Output και πολλαπλών επεξεργαστών [7].

Η προέλευση των container αποδίδεται συχνά στην προσθήκη της κλήσης συστήματος chroot [8] στην έβδομη έκδοση του UNIX που κυκλοφόρησε από την Bell Labs το 1979 [9]. Η απλή μορφή απομόνωσης χώρου ονοματοδοσίας συστήματος αρχείων που παρέχει το chroot ήταν σίγουρα μια επιρροή στην ανάπτυξη των container, ωστόσο δεν προσέφερε απομόνωση για χώρους ονοματοδοσίας διεργασιών. Τα container δεν είναι μία ενιαία τεχνολογία. Είναι μια συλλογή τεχνολογιών που συνδυάζονται για να παρέχουν ασφαλή απομόνωση, συμπεριλαμβανομένων χώρων ονοματοδοσίας, cgroups, seccomp και δυνατοτήτων.

Παραδοσιακά οι οργανισμοί εκτελούσαν τις εφαρμογές τους σε φυσικούς διακομιστές (physical servers), οι οποίοι δεν παρείχαν κάποιον τρόπο να οριοθετήσουν τους πόρους που κάθε εφαρμογή ή υποσυστημά της κατανάλωνε, με αποτέλεσμα να δημιουργούνται προβλήματα [10] . Πρώτον, όταν ένας φυσικός διακομιστής (server) φιλοξενούσε μόνο μία εφαρμογή, δεν χρησιμοποιούταν όλοι οι διαθέσιμοι πόροι με αποτέλεσμα να κατέληγαν ανεκμετάλλευτοι. Δεύτερον, όταν φιλουξενούσε περισσότερες από μία εφαρμογές προέκυπταν συχνά προβλήματα στην ανάθεση των πόρων με αποτέλεσμα να μειωνόταν η γενική αποδοσή του. Επειδή οι εφαρμογές εκτελούνταν στο ίδιο υπολογιστικό μηχάνημα, είχαν απευθείας πρόσβαση στους πόρους του με αποτέλεσμα να επηρεάζουν αρνητικά την απόδση. Για παράδειγμα, αν μία εφαρμογή είχε αυξημένες απαιτήσεις επεξεργαστή ή μνήμης, μπορούσε να προκαλέσει συμφόρηση στο μηχάνημα και να μην λειτουργούν σωστά οι υπόλοιπες εφαρμογές εφόσον δεν είχαν την δυνατότητα να χρησιμοποιήσουν τους πόρους που απαιτούσαν με αποτέλεσμα να μειωθεί η συνολική του απόδοση [11]. Επιπλέον, εμφανίστηκε το πρόβλημα της κλιμάκωσης των εφαρμογών, διότι δεν υπήρχαν εργαλεία αυτόματης κλιμάκωσης ή εύκολης καταναμοής των πόρων με αποτέλεσμα όταν αυξανόταν η πολυπλοκότητα των εφαρμογών ή ο αριθμός των εφαρμογών να μειωνόταν η συνολική απόδοση του μηχανήματος.

Εξαιτίας των παραπάνω περιορισμών, καθίσταται αναγκαία η υιοθέτηση σύγχρονων τεχνολογιών, όπως η εικονοποίηση ή το containerization, που προσφέρουν καλύτερη απομόνωση, πιο αποδοτική χρήση πόρων και εύκολες μεθόδους κλιμάκωσης, βελτιώνοντας έτσι την απόδοση και τη σταθερότητα των εφαρμογών και του μηχανήματος.




Σχήμα 2.1: Παραδοσιακή εκτέλεση εφαρμογών

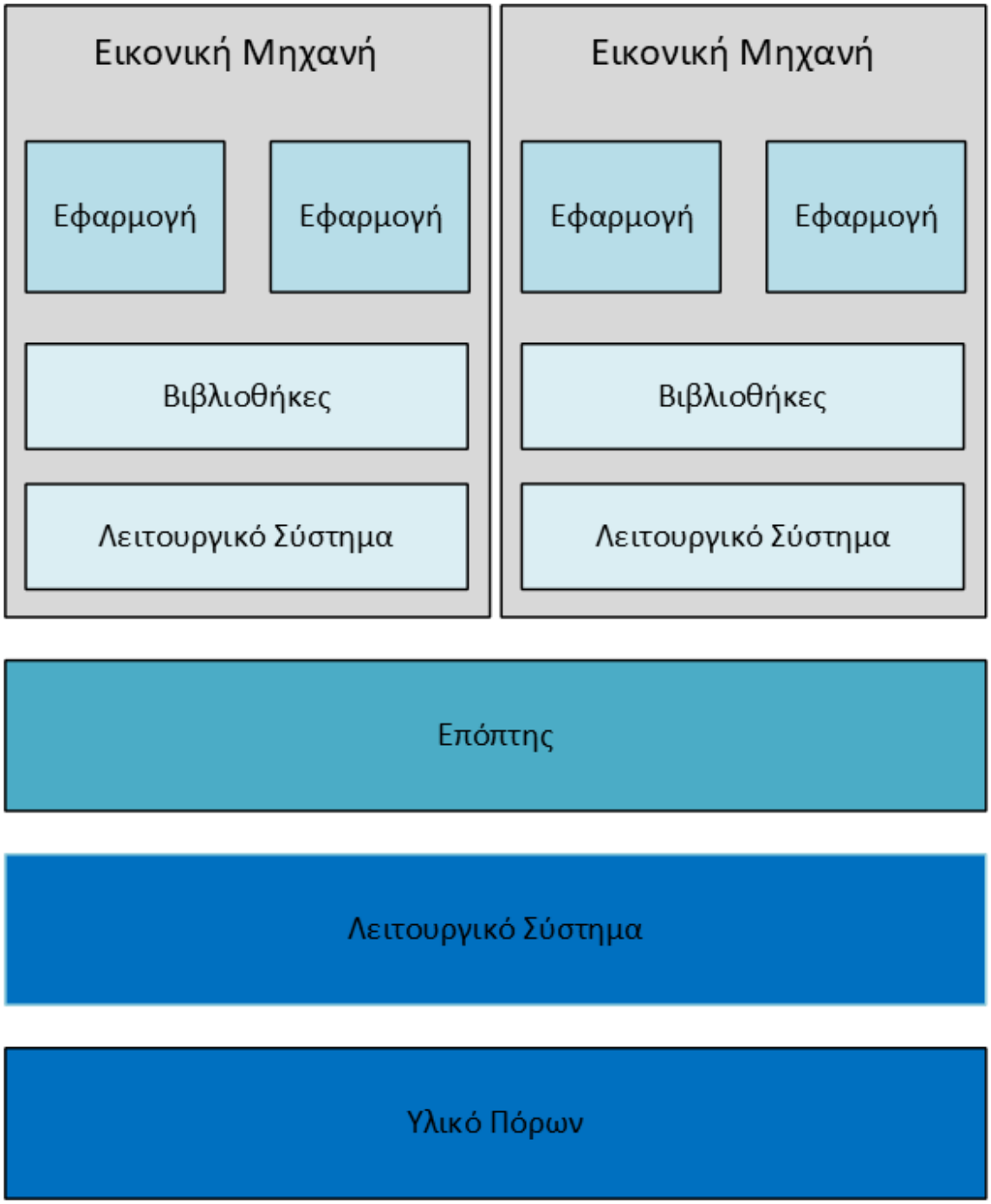
Οι εικονικές μηχανές υπήρχαν στις δεκαετίες του 1980 και του 1990, αλλά δεν είχαν σημαντική απήχηση στους οργανισμούς. Η άνοδος των γλωσσών προγραμματισμού όπως η Smalltalk και η Java έδωσαν νέο σκοπό στον όρο εικονική μηχανή.

Η εικονοποίηση είναι η δημιουργία μίας εικονικής έκδοσης των πόρων υλικού του Ηλεκτρονικού Υπολογιστή (Η/Υ) όπως η κεντρική μονάδα επεξεργασίας (Central Processing Unit - CPU), η μνήμη τυχαίας προσπέλασης (Random Access Memory - RAM), ή ενός Λειτουργικού Συστήματος. Οι επόπτες (Hypervisors) είναι λογισμικά τα οποία αναλμβάνουν την διαχώριση και την εικονοποίηση των υλικών πόρων. Υπάρχουν δύο είδη εποπτών, ο τύπου ένα επόπτης ο οποίος εγκαθίσταται απευθείας στους υλικούς πόρους του Η/Υ και ο τύπου δύο επόπτης ο οποίος εγκαθίσταται απευθείας στο Λειτουργικό Σύστημα [12]. Ένα παράδειγμα εικονοποίησης είναι η εικονική μηχανή (Virtual Machine - VM), ένας εικονικός υπολογιστής που έχει το δικό του Λειτουργικό Σύστημα και εκτελεί τις δικές του εφαρμογές. Μπορούν να φιλοξενηθούν πολλοί εικονικοί ηλεκτρονικοί υπολογιστές σε έναν φυσικό Η/Υ και ο κάθε ένας από αυτούς είναι πλήρως απομωνομένος από τους υπόλοιπους. Το Virtual box, είναι ένας τύπου δύο επόπτης και μπορεί να εγκατασταθεί σε οποιονδήποτε Η/Υ, προσφέροντας την δυνατότητα στον χρήστη να εγκαταστήσει ένα ή και περισσότερα λειτουργικά συστήματα σε αυτόν αναλόγως τους πόρους μνήμης και του αποθηκευτικού χώρου που διαθέτει. Τα επιπλέον εγκαθιστόμενα λειτουργικά συστήματα ονομάζονται φιλοξενούμενα (GuestOS) [13].

Η χρήση της εικονοποίησης προσφέρει αρκετά πλεονεκτήματα και βελτιστοποιήσεις συγκριτικά με την παραδοσιακή μέθοδο εκτέλεσης [14]. Πιο συγκεκριμένα, επιτρέπει την πολλαπλή χρήση εικονικών μηχανών σε έναν διακομιστή. Οι εφαρμογές είναι απο-

μονωμένες μεταξύ των εικονικών μηχανών, παρέχοντας ένα επίπεδο ασφάλειας καθώς στα δεδομένα μίας εφαρμογής δεν μπορούν να έχουν ελεύθερη πρόσβαση οι εφαρμογές των υπόλοιπων εικονικών μηχανών που εκτελούνται στο ίδιο μηχάνημα. Υπάρχει καλύτερη χρήση των πόρων του διακομιστή και προσφέρει καλύτερη επεκτασιμότητα διότι οι εφαρμογές μπορούν να προστεθούν, να ενημερωθούν ή να αφαιρεθούν [15]. Επίσης, λόγω της μείωσης των φυσικών διακομιστών, μειώνεται η κατανάλωση ενέργειας οπότε και το συνολικό κόστος. Τέλος, με τη χρήση της εικονοποίησης ελαχιστοποιείται ο χρόνος εγκατάστασης διότι εγκαθιστάται το λογισμικό και είναι έτοιμο για χρήση, ενώ με τον παραδοσιακό τρόπο έπρεπε πρώτα να αγοραστούν οι πόροι Η/Υ, μετά να συναρμολογηθούν και να εγκατασταθουν τα απαραίτητα λογισμικά [16].

Στο σχήμα 2.3, φαίνεται πως, το τελευταιο επίπεδο είναι το υλικό πόρων στο οποίο στηρίζεται η υποδομή. Για να εκμεταλλευτούν οι πόροι είναι αναγκαία η ύπαρξη ενός Λειτουργικού Συστήματος, ώστε ο χρήστης να αλληλεπιδρά με το σύστημα. Στο Λειτουργικό Σύστημα εγκαθείστατε μία εφαρμογή η οποία εκμεταλλεύεται την εικονοποίηση του επεξεργαστή και μπορεί να δημιουργεί εικονικές μηχανές οι οποίες με το δικό τους Λειτουργικό Σύστημα μπορούν να στηρίξουν τις ανάγκες του χρήστη, εκτελώντας εφαρμογές.




Σχήμα 2.2: Εκτέλεση εφαρμογών σε εικονικές μηχανές

## 2.2 Σύγχρονη προσέγγιση στην ενορχήστρωση πόρων

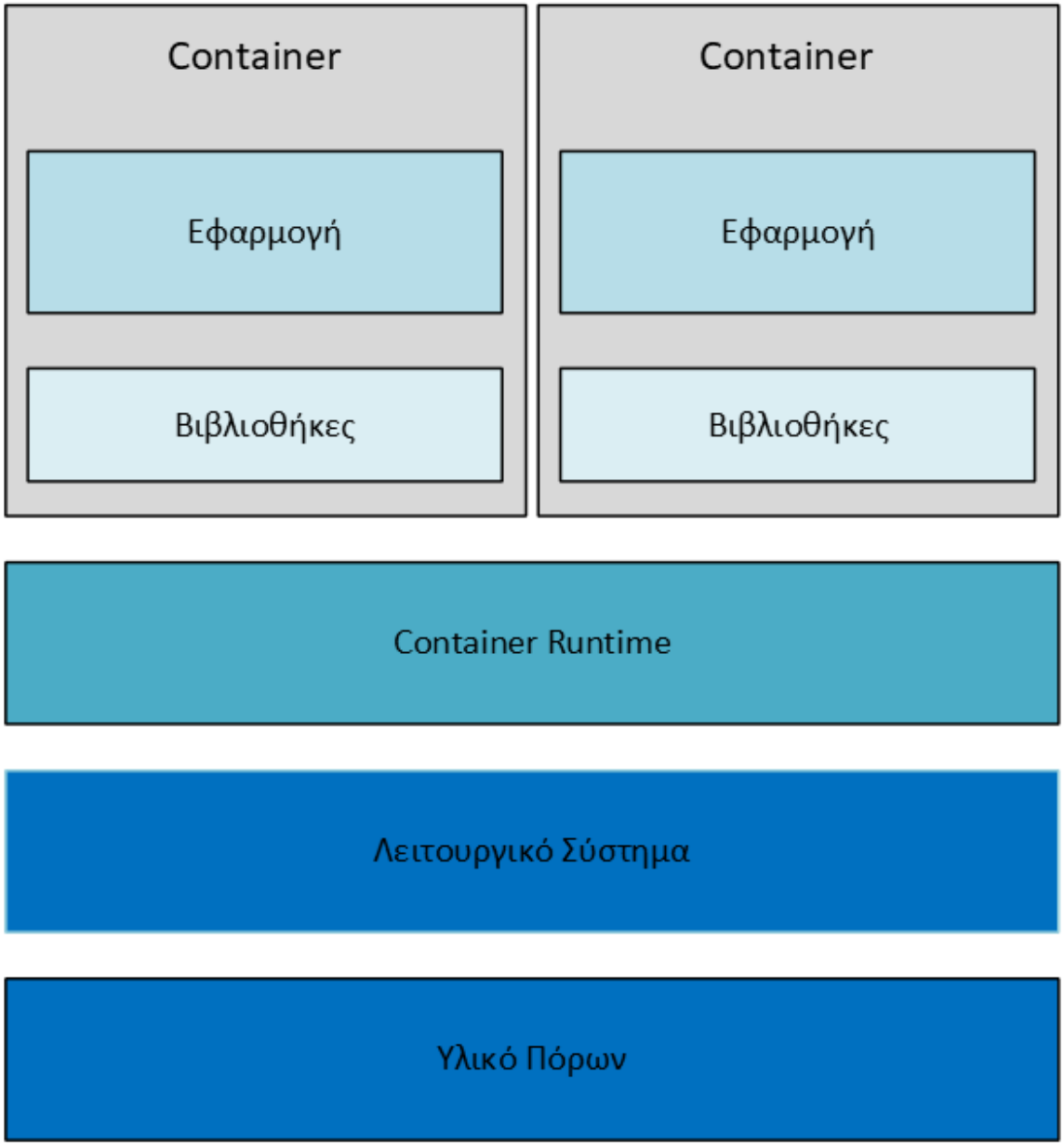
Ένα container είναι μία μονάδα λογισμικού που μπορεί να εκτελέσει κώδικα παρέχοντας τις απαραίτητες πρυποθέσεις, ώστε η εφαρμογή να εκτελείται γρήγορα και να μεταφέρεται από το ένα υπολογιστικό περιβάλλον στο άλλο. Ένα cotnainer image είναι ένα ελαφρύ, αυτόνομο, εκτελέσιμο πακέτο λογισμικού που περιλαμβάνει όλα τα απαραίτητα για την εκτέλεση της εφαρμογής. Αυτά είναι ο κώδικας, το περιβάλλον εκτέλεσης, τα εργαλεία συστήματος, οι βιβλιοθήκες συστήματος και οι ρυθμίσεις. Ένα

container image είναι πολύ πιο ελαφρύ από ένα VM, καταλαμβάνει λιγότερο αποθηκευτικό χώρο και καταναλώνει λιγότερη μνήμη. Επίσης η παύση και η επανεκτέλεσή του γίνεται εντός δευτερολέπτων [17].

Τα container επιτρέπουν στους προγραμματιστές να προωθούν τον κώδικα γρήγορα από την ανάπτυξη στην παραγωγή. Ο διαχωρισμός των εφαρμογών σε μικρότερα στοιχεία βελτιώνει την ταχύτητα ανάπτυξης και τις δοκιμές, με το Kubernetes ως διαχειριστή. Ενισχύοντας την ανθεκτικότητα, καθώς τα containers μπορούν να επανεκκινηθούν ανεξάρτητα, χωρίς να επηρεάζουν ολόκληρο το σύστημα, διασφαλίζοντας έτσι, υψηλή διαθεσιμότητα.

Η λειτουργική συνέπεια επιτυγχάνεται μέσω της ομοιόμορφης διαχείρισης των containerized εφαρμογών, απλοποιώντας τις διαδικασίες παραμετροποίησης και εκτέλεσης σε διάφορα περιβάλλοντα. Επιπλέον βελτιώνεται η επεκτασιμότητα καθώς τα container εκτελούνται γρήγορα, καθιστώντας δυνατή τη δυναμική κλιμάκωση των πόρων. Το Kubernetes βελτιστοποιεί τη χρήση των πόρων καθώς προσαρμόζεται βάσει του φόρτου εργασίας.

Η φορητότητα είναι άλλο ένα πλεονέκτημα, καθώς τα container μπορούν εύκολα να μετακινηθούν σε διαφορετικές πλατφόρμες, υποστηρίζοντας στρατηγικές multicloud. Για να επωφεληθούν πλήρως από αυτές τις δυνατότητες, οι οργανισμοί πρέπει επίσης να ασπαστούν την ευέλικτη ανάπτυξη και την αυτοματοποίηση στις ροές εργασίας τους [18].




Σχήμα 2.3: Εκτέλεση εφαρμογών με containers

## 2.3 Συστήματα Ενορχήστρωσης container βασισμένων στο Kubernetes

Το Containerization έχει καθιερωθεί ως η προτιμώμενη επιλογή για την ταχεία ανάπτυξη εφαρμογών, λόγω της ελαφριάς φύσης της και των δυνατοτήτων άμεσης εκκίνησης. Αυτή η προσέγγιση είναι ιδιαίτερα δημοφιλής σε τομείς όπως τα μεγάλα δεδομένα (Big Data) και το Cloud Computing. Παρά τα πλεονεκτήματα αυτά, η πρόκληση έγκειται στη βέλτιστη κατανομή πόρων σε ετερογενή περιβάλλοντα, διότι οι ενορχηστρωτές container συμβατοί με το πρότυπο Kubernetes παρέχουν μόνο βασικους μηχανισμούς χρονοδρομολόγησης. Η παρούσα μελέτη [19] αντιμετωπίζει αυτούς τους περιορισμούς προτείνοντας μια προηγμένη αρχιτεκτονική τοποθέτησης container με γνώμονα τους

πόρους και παρακολούθηση βασισμένη στο Service Level Agreement - SLA. Αυτή η λύση βελτιστοποιεί την χρονοδρομολόγηση, ελαχιστοποιεί τις μετεγκαταστάσεις container και βελτιώνει τη συνολική απόδοση. Λαμβάνοντας υπόψη παράγοντες όπως η διαθεσιμότητα πόρων, η πλατφόρμα ανάπτυξης και ο τύπος εφαρμογής, η προτεινόμενη λύση βελτιώνει την ενορχήστρωση και διασφαλίζει την αποδοτική διαχείριση των πόρων.

Το Kubernetes είναι ένα ευρέως χρησιμοποιούμενο εργαλείο ενορχήστρωσης container, γνωστό για την επεκτασιμότητα, την αξιοπιστία και την ανοχή σε σφάλματα. Ωστόσο, απαιτεί σημαντικούς πόρους, καθιστώντας το λιγότερο κατάλληλο για περιβάλλοντα με περιορισμένη υπολογιστική ισχύ. Για να αντιμετωπιστεί αυτό, έχουν αναπτυχθεί ελαφριές εναλλακτικές λύσεις όπως το MicroK8s και το K3s. Αυτό το άρθρο [1] παρουσιάζει μια σύγκριση απόδοσης του Kubernetes με αυτές τις ελαφριές εκδόσεις, εστιάζοντας σε μετρήσεις όπως η χρήση πόρων, ο χρόνος εκκίνησης της συστοιχίας υπολογιστών και ο χρόνος που καταναλώνεται σε λειτουργίες κόμβων. Τα αποτελέσματα δείχνουν ότι ενώ το Kubernetes γενικά υπερέχει των υπόλοιπων διανομών στις περισσότερες δοκιμές, το K3S επιδεικνύει καλύτερη χρήση του δίσκου λόγω της χρήσης της ελαφριάς βάσης δεδομένων SQLite. Το MicroK8s, ωστόσο, υστερεί ως προς τη χρήση πόρων και την κατανάλωση χρόνου. Παρά τις διαφορές τους, όλες οι διανομές παρέχουν συγκρίσιμη απόδοση για τις περισσότερες εφαρμογές του πραγματικού κόσμου, με τις ελαφριές εκδόσεις να είναι πιο κατάλληλες για περιβάλλοντα με περιορισμένους πόρους χωρίς σημαντικές απώλειες απόδοσης.

Καθώς το containerization γίνεται μια κοινή προσέγγιση για την ανάπτυξη λογισμικού, υπάρχει αυξανόμενη ζήτηση για ενορχήστρωση container που βασίζεται στο πρότυπο Kubernetes σε συσκευές περιορισμένων πόρων, όπως το Internet of Things - IoT και οι πλατφόρμες υπολογιστών αιχμής Edge Computing. Αρκετές ελαφριές διανομές Kubernetes, όπως το MicroK8s, k3s, k0s και MicroShift, έχουν προκύψει για να αντιμετωπίσουν αυτήν την ανάγκη. Αυτό το άρθρο [20] παρέχει μια συγκριτική ανάλυση απόδοσης αυτών των κατανομών όσον αφορά τη χρήση πόρων και την απόδοση του επιπέδου ελέγχου/δεδομένων υπό σενάρια καταπόνησης. Η μελέτη διαπίστωσε ότι τα k3s και k0s προσφέρουν την υψηλότερη απόδοση επιπέδου ελέγχου, ενώ το MicroShift υπερέχει στο επίπεδο δεδομένων. Τα αποτελέσματα αυτής της μελέτης βοηθούν τους προγραμματιστές και τους διαχειριστές συστημάτων να επιλέξουν την κατάλληλη ελαφριά διανομή Kubernetes για χρήση σε περιβάλλοντα με περιορισμένους πόρους.

Το Kubernetes έχει γίνει το κορυφαίο εργαλείο ενορχήστρωσης container στην υπολογιστική νέφους. Ωστόσο, η χρήση του Kubernetes σε περιβάλλοντα με περιορισμένους πόρους, παρουσιάζει νέες προκλήσεις. Για να αντιμετωπιστεί αυτό, έχουν αναπτυχθεί ελαφριές διανομές όπως K0s και K3s. Αυτή η μελέτη [21] παρέχει μια ολοκληρωμένη σύγκριση των K0s, K3s και Vanilla Kubernetes όσον αφορά την απόδοση και την ασφάλεια. Τα αποτελέσματα δείχνουν ότι το K0s προσφέρει την καλύτερη συνολική απόδοση, ειδικά στη δημιουργία pod και την επεκτασιμότητα, ενώ το Vanilla Kubernetes υπερέχει στην ασφάλεια. Το K3s, αν και ελαφρύ, αντιμετωπίζει προβλήματα σχετικά με την επεκτασιμότητα υπό βαριά φορτία και παρουσιάζει τις περισσότερες ευπάθειες ασφάλειας. Αυτή η έρευνα παρέχει πληροφορίες για την επιλογή της καταλληλότερης διανομής Kubernetes με βάση τις ανάγκες απόδοσης και ασφάλειας σε διαφορετικά περιβάλλοντα.

Οι Amirtha Varshini M, Aruna P, Sanjana V και Chitra A D στην έρευνά τους [22]

παρουσιάζουν μια σύγκριση των Docker container και των εικονικών μηχανών (VM) κατά την εκτέλεση μίας διαδικτυακής εφαρμογής βασισμένης σε React.js. Η αυξανόμενη δημοτικότητα της τεχνολογίας container, λόγω των ελαφριών και αποτελεσματικών δυνατοτήτων ανάπτυξής τους, έχει οδηγήσει σε μια πιο προσεκτική εξέταση της απόδοσής της σε σχέση με τις παραδοσιακές εικονικές μηχανές. Η έρευνα επικεντρώνεται στη χρήση πόρων, την απομόνωση, την πολυπλοκότητα της ανάπτυξης και τις διαφορές απόδοσης μεταξύ Docker container και VM. Τα αποτελέσματα δείχνουν ότι το Docker container προσφέρει σημαντικά πλεονεκτήματα όσον αφορά την αποδοτικότητα των πόρων και την ταχύτερη εκτέλεση εφαρμογών, καθιστώντας το μια ανώτερη επιλογή για τη φιλοξενία σύγχρονων εφαρμογών ιστού. Ωστόσο, οι εικονικές μηχανές παραμένουν πιο κατάλληλες για σενάρια που απαιτούν αυστηρή απομόνωση και υψηλότερη ασφάλεια. Τα ευρήματα υποδηλώνουν ότι τα Docker container είναι ιδανικά για εφαρμογές που απαιτούν επεκτασιμότητα και ελάχιστο κόστος πόρων, ενώ οι εικονικές μηχανές μπορεί να προτιμώνται όταν η ισχυρότερη απομόνωση και η ασφάλεια είναι κρίσιμες.

Οι Indrani Vasireddy, Prathima Kandi και SreeRamya Gandu στην έρευνα τους [23] παρουσιάζουν μια ολοκληρωμένη ανασκόπηση των λύσεων εξισορρόπησης φορτίου στο Kubernetes, μια πλατφόρμα ενορχήστρωσης container ανοιχτού κώδικα που αυτοματοποιεί την ανάπτυξη, την κλιμάκωση και τη διαχείριση των containerized εφαρμογών. Καθώς τα σύγχρονα κατανεμημένα συστήματα (Distributed Systems) βασίζονται όλο και περισσότερο στην αποτελεσματική χρήση πόρων, το Kubernetes παίζει κρίσιμο ρόλο στη δυναμική προσαρμογή του φόρτου εργασίας μεταξύ των κόμβων για τη βελτιστοποίηση της απόδοσης, της επεκτασιμότητας και της διαθεσιμότητας. Η έρευνα εξετάζει διάφορες στρατηγικές εξισορρόπησης φορτίου, συμπεριλαμβανομένων αλγορίθμων που βασίζονται στην κυκλοφορία και με επίγνωση των πόρων, καθώς και προσεγγίσεις που βασίζονται στη συγγένεια, επισημαίνοντας τον αντίκτυπό τους στους φόρτους εργασίας με containers. Βασικές προκλήσεις, όπως οι κυμαινόμενες απαιτήσεις πόρων και η δυναμική φύση των εγγενών εφαρμογών στο cloud, συζητούνται παράλληλα με τις ανταλλαγές όσον αφορά τον χρόνο απόκρισης, την απόδοση και την επεκτασιμότητα. Η ανασκόπηση διερευνά επίσης τις εξελίξεις στην εξισορρόπηση φορτίου Kubernetes για υπολογιστές ακμών, προσφέροντας πληροφορίες για μελλοντικές βελτιώσεις στην ενορχήστρωση πόρων. Μέσω αυτής της ανάλυσης, η εργασία ενοποιεί την υπάρχουσα γνώση και παρουσιάζει κατευθύνσεις για περαιτέρω έρευνα σχετικά με τη βελτίωση της χρήσης πόρων και της εξισορρόπησης φορτίου σε περιβάλλοντα Kubernetes.

Οι Indrani Vasireddy, Prathima Kandi και G Ramya στην έρευνα τους [24] εξετάζουν τις σύγχρονες τεχνικές και τις προκλήσεις στην εξισορρόπηση φορτίου για containerized εφαρμογές που χρησιμοποιούν Kubernetes και Docker. Με το Kubernetes να γίνεται η κορυφαία πλατφόρμα για την εκτέλεση, την κλιμάκωση και τη διαχείριση containerized εφαρμογών, η έρευνα εξετάζει πως η εξισορρόπηση φορτίου παίζει σημαντικό ρόλο στη διασφάλιση της αποτελεσματικής χρήσης των πόρων και της υψηλής διαθεσιμότητας. Συγκρίνει τις τεχνικές εξισορρόπησης φορτίου των Kubernetes και Docker Swarm, επισημαίνοντας τα πλεονεκτήματα του Kubernetes. Αναλύονται σύγχρονοι αλγόριθμοι εξισορρόπησης φορτίου, όπως Round Robin, Least Connections και Consistent Hashing, και η εφαρμογή τους και στα δύο οικοσυστήματα. Επιπλέον, αντιμετωπίζει τις προκλήσεις που σχετίζονται με containerized εφαρμογές, όπως η βελτιστοποίηση απόδοσης σε κατανεμημένα συστήματα.

## 2.4 Προκλήσεις και Ανοιχτά ζητήματα

Παρότι η ενορχήστρωση των container έχει εξαπλωθεί και χρησιμοποιείται από πολλούς οργανισμούς, διέπεται από προκλήσεις και ανοιχτά ζητήματα. Αυτά μπορεί να είναι:

- Η διαχείριση περιβάλλοντων ενορχήστρωσης container απαιτεί πολύ καλές γνώσεις ειδικότερη με την κλιμάκωση του Cluster. Υπάρχει η ανάγκη για την περεταίρω απλοποίηση αυτών των περιβάλλοντων. Αν και το Kubernetes είναι ένα δυνατό εργαλείο, ένα λάθος σε ρύθμιση μπορεί να οδηγήσει προβλήματα επίδοσης και ασφάλειας.
- Η αξιοποίηση των υπολογιστικών πόρων ενός Cluster μεγάλης κλίμακας, που αποτελείται από εκατοντάδες υπολογιστικά μηχανήματα, αποτελεί σημαντική πρόκληση. Οι τωρινοί αλγόριθμοι ταξινόμησης πόρων έχουν αρκετά περιθώρια βελτείωσης ακόμα. Η ανάπτυξη πιο αποδοτικών αλγόριθμων για τη σωστή κατανομή και διαχείριση πόρων είναι ένα ζήτημα που πρέπει να αντιμετωπιστεί στο άμεσο μέλλον.
- Η διαχείριση της εσωτερικής δικτυακής επικοινωνίας του Cluster, της εύρεσης υπηρεσιών και της δρομολόγησης της κίνησης μπορεί να γίνει αρκετά απαιτητική, ειδικά σε περιβάλλοντα μεγαλύτερης κλίμακας. Η επικοινωνία μεταξύ πολλών Cluster είναι ένα θέμα το οποίο ερευνάται και βελτιώνεται συνεχώς.
- Το Kubernetes προσφέρει διάφορες επιλογές αποθηκευτικού χώρου όπως είναι τα Persistent Volume Claims(PVC) και το Container Storage Interface(CSI). Όμως, η ενσωμάτωση των διαφορετικών επιλογών αποθηκευτικού χώρου δεν είναι απρόσκοπτη καθώς υπάρχουν ζητήματα με την διαχείριση των δεδομένων και στην μείωση της επίδοσης, ειδκότερα σε πιο σύνθετα περιβάλλοντα.

Η ενορχήστρωση των containerized εφαρμογών προσφέρει πολλά οφέλη στα περιβάλλοντα που η αποδοτική διαχείριση πόρων και η αυτοματοποίηση είναι κρίσιμη. Μερικές περιπτώσεις τέτοιων εφαρμογών είναι οι παρακάτω.

- Η ενορχήστρωση είναι απαραίτητη για τη διαχείριση υποδομών που εκτελούν απαιτητικές εφαρμογές μηχανικής μάθησης, αυτοματοποιεί την κατανομή των πόρων του συστήματος, την εκτέλεση των pipelines και τη διαχείριση διαδικασιών εκπαίδευσης μοντέλων σε κατανεμημένα περιβάλλοντα. Όμως η ανάγκη για βελτιστοποίηση της απόδοσης κατά την κατανομή πόρων σε απαιτητικά υπολογιστικά φορτία και η διαχείριση αποθήκευσης για μεγάλα σύνολα δεδομένων παραμένουν προκλήσεις που περιορίζουν την αποδοτικότητα των συστημάτων αυτών.
- Σε κατανεμημένα περιβάλλοντα που βασίζονται στο edge computing και το IoT, η ενορχήστρωση επιτρέπει την κλιμάκωση και την αυτόματη ενημέρωση των εφαρμογών που εκτελούνται σε διασκορπισμένες συσκευές. Η διαχείριση της ασταθούς συνδεσιμότητας, οι περιορισμένοι υπολογιστικοί πόροι των edge συσκευών και η διατήρηση της απόδοσης σε πραγματικό χρόνο είναι ζητήματα που χρείζουν βελτιστοποίησης.
- Οι Εφαρμογές Big Data που επεξεργάζονται μεγάλες ποσότητες δεδομένων επωφελούνται από την ενορχήστρωση, μέσω της αυτοματοποίησης της κατανομής υπολογιστικών και αποθηκευτικών πόρων αυξάνοντας την απόδοση. Η διαχείρι-

ση και η βελτιστοποίηση του δικτύου για τη μεταφορά μεγάλων όγκων δεδομένων μεταξύ διακιμιστών, καθώς και η εξασφάλιση της συνέπειας δεδομένων σε κατανεμημένα περιβάλλοντα, απαραίτητα για την αξασφάλιση των χαρακτηριστικών των Big Data.

- Οι εφαρμογές που βασίζονται σε αρχιτεκτονική microservices επωφελούνται από την ενορχήστρωση container, καθώς επιτρέπεται η αυτόματη διαχείριση πολλών μικρών υπηρεσιών που τις συνθέτουν. Η αύξηση αυτών των υπηρεσιών, η ανακατανομή φορτίου και η ανθεκτικότητα σε αποτυχίες είναι θεμελιώδη πλεονεκτήματα. Η ενοποίηση της παρακολούθησης και της διαχείρισης καταγραφών είναι μια πρόκληση, καθώς η πολυπλοκότητα του συστήματος αυξάνεται με την αύξηση των microservices.

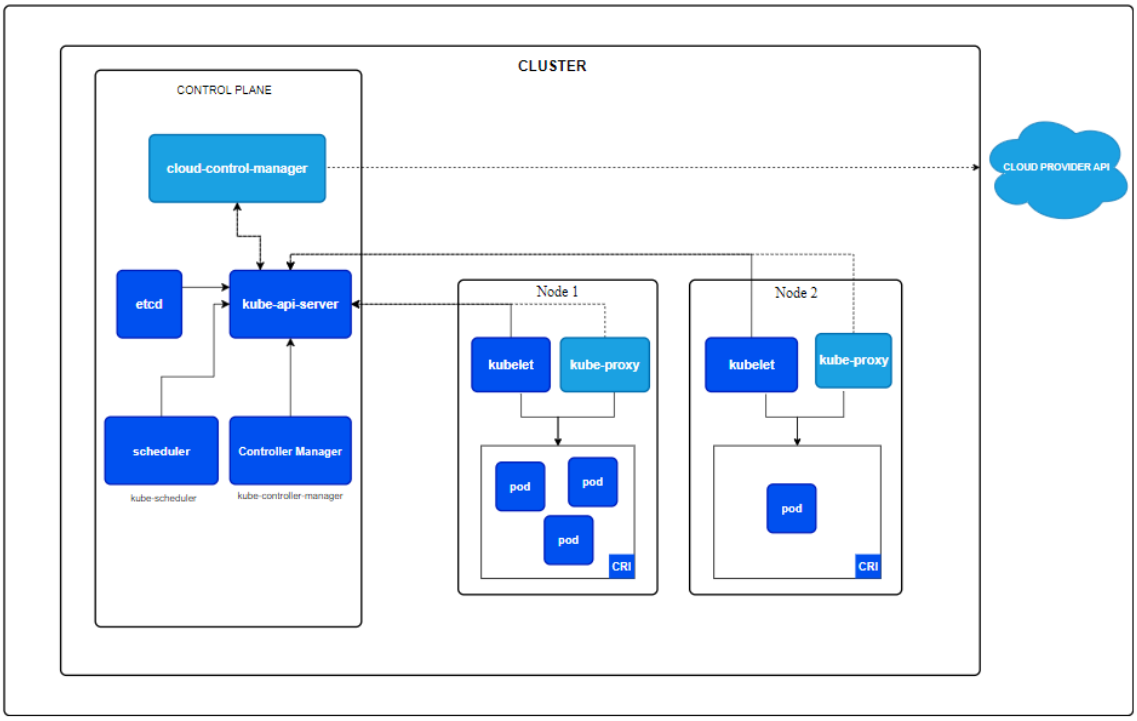
Συμπερασματικά, ενώ η ενορχήστρωση container έχει βελτιώσει σημαντικά τη δυνατότητα διαχείρισης των containerized εφαρμογών σε κλίμακα, εξακολουθεί να αντιμετωπίζει σημαντικές προκλήσεις σε αρκετούς βασικούς τομείς. Η πολυπλοκότητα της διαχείρισης αυτών των περιβάλλοντων, η βελτιστοποίηση της χρήσης πόρων και η αντιμετώπιση προβλημάτων δικτύωσης και αποθήκευσης αποτελούν συνεχή εμπόδια που απαιτούν πιο εκλεπτυσμένες λύσεις. Καθώς οι containerized εφαρμογές γίνονται ολοένα και πιο δημοφιλείς, η ανάγκη για πιο αποτελεσματικά, φιλικά προς τον χρήστη και ολοκληρωμένα συστήματα ενορχήστρωσης γίνεται όλο και πιο σημαντική. Στο πλαίσιο αυτό, η συγκριτική μελέτη ενορχηστρωτών πόρων συμβατών με το πρότυπο Kubernetes μπορεί να βοηθήσει στην κατανόηση της αποδοτικότητας και των διαφορών που παρουσιάζουν αυτοί οι ενορχηστρωτές βάση των αποτελεσμάτων της χρήσης του επεξεργαστή, της μνήμης και του δικτύου.

# Κεφάλαιο 3

# Εργαλεία Ενορχήστρωσης Πόρων

## 3.1 Αρχιτεκτονική του Ενορχηστρωτή container Kubernetes

Το kubernetes είναι μία πλατφόρμα διαχείρησης container, με στόχο να διευκολύνει τόσο τη δηλωτική διαμόρφωση όσο και τον αυτοματισμό. Οι υπηρεσίες, η υποστήριξη και τα εργαλεία του Kubernetes είναι ευρέως διαθέσιμα. Τα container είναι ένας καλός τρόπος για την ομαδοποίηση και την εκτέλεση των εφαρμογών. Σε οργανισμούς που χαρακτηρίζονται από περίπλοκα υπολογστικά συστήματα με πολλές εξαρτήσεις η χρήση των container επιφέρει πολλά πλεονεκτήματα όπως, εάν ένα container σταματήσει να λειτουργεί πρέπει ο διαχειριστής του συστήματος να το επανεκτελέσει χειροκίνητα, το Kubernetes αναλαμβάνει αυτή την εργασία. Παρέχει ανακάλυψη υπηρεσιών και εξισορρόπηση φορτίου, εκθέτει τα container χρησιμοποιώντας το DNS όνομα τους ή την IP διεύθυνση τους και αν υπάρχει μεγάλο δικτυακό φορτίο, τότε το εξισορροπεί και το διαμοιράζει, ώστε η εφαρμογή να παραμείνει σταθερή [25]. Προσφέρει αυτοματοποιήμενες εκδόσεις και επαναφορά σε προηγούμενες, ο διαχειριστής μπορεί να περιγράψει την επιθυμητή κατάσταση των container και να αλλάξει την επιθμητή καταστασή τους, για παράδειγμα, μπορεί να αυτοματοποιήσει το Kubernetes να δημιουργεί νέα container για την εφαρμογή αφαιρώντας τα ήδη υπάρχωντα και οι πόροι που έχουν δεσμευτεί να χρησιμοποιηθούν από τα νέα container. Ο διαχειριστής έχει τη δυνατότητα να ορίσει την επεξεργαστική ισχύ και την μνήμη RAM που μπορεί να χρησιμοποιήσει το κάθε container παρέχοντας την βέλτιστη λειτουργία της εφαρμογής. Αν κάποιο container αποτύχει, το Kubernetes θα το επανεκκινήσει ή θα το αντικαταστήσει, επίσης αν κάποιο container δεν ανταποκρίνεται τότε θα το τερματίσει χωρίς να το αναφέρει στον διαχειριστή μέχρι να είναι έτοιμο για χρήση. Επιτρέπει την αποθήκευση και την διαχείρηση ευαίσθητων πληροφοριών όπως κωδικοί πρόσβασης, ανανγωστιστηκά πιστοποίησης OAuth και SSH κλειδιά. Μπορεί να εγκαταστήσει και να ενημερώσει τα διαπιστευτήρια και τις διαμορφώσεις των εφαρμογών χωρίς να χρειαστεί η επανεκτέλεση των εφαρμογών και χωρίς την έκθεση των διαπιστευτηρίων. Υπάρχει η δυνατότητα της επέκτασης των υπολογιστικών πόρων που έχουν ανατεθεί στην εφαρμογή με τη χρήση μίας εντολής τερματικού ή μέσω γραφικού περιβάλλοντος ή και αυτόματα. Εκχωρεί διευθύνσεις IPv4 και IPv6 σε pods και υπηρεσίες. Ενώ τέλος, προσφέρει τη δυνατότητα επέκτασης λειτουργιών χωρίς την ανάγκη αλλαγής του πηγαίου κώδικα της εφαρμογής [26].




Σχήμα 3.1: Αρχιτεκτονική Kubernetes

Το kubernetes δεν είναι ένα παραδοσιακό σύστημα Πλατφόρμα ως Υπηρεσία (PaaS) καθώς λειτουργεί σε επίπεδο container και όχι σε επίπεδο υλικού. Ενώ προσφέρει κάποια κοινά χαρακτηριστικά με το PaaS όπως ανάπτυξη, επέκταση, εξισορρόπηση φορτίου και επιτρέπει στον διαχειρηστή να ενσωματώσει τις δικές του λύσεις καταγραφής, παρακολούθησης και ειδοποίησης. Παρέχει τα δομικά στοιχεία για τη δημιουργία πλατφόρμων για προγραμματιστές ώστε να αναπτύσουν και να δοκιμάζουν εφαρμογές διατηρώντας την ευελιξία και την επιλογή των χρηστών όπου είναι αναγκαίο.

Το kubenetes API επιτρέπει στον διαχειριστή να κάνει ερωτήματα (queries) και να χειρίζεται την κατάσταση των αντικειμένων στο Kubernetes. Ο πυρήνας του επιπέδου ελέγχου (Control plane) είναι το API server το οποίο παρέχει το HTTP API . Οι χρήστες και όλα τα εσωτερικά και εξωτερικά στοιχεία του Kubernetes επικοινωνούν μεταξύ τους μέσω του API server. Οι περισσότερες λειτουργίες μπορούν να εκτελεστούν μέσω της διεπαφής kubectl η οποία χρησιμοποιεί το API.

Ένα kubernetes cluster αποτελείται από ένα ή και περισσότερα control plane και από ένα σύνολο υπολογιστικών μηχανημάτων που ονομάζονται κόμβοι (nodes) στους οποίους εκτελούνται οι εφαρμογές. Αυτοί οι κόμβοι είναι οι worker nodes και φιλοξενούν τα pods, στοιχεία του φόρτου εργασίας της εφαρμογής.Το control plane διαχειρίζεται τα worker nodes και τα pods του cluster. Σε έναν οργανισμό, το control plane συνήθως εκτελείται σε πολλούς υπολογιστές και σε ένα cluster εκτελούνται πολλά pods παρέχοντας ανοχή σε σφάλματα και υψηλή διαθεσιμότητα. Τα στοιχεία του control plane λαμβάνουν καθολικές αποφάσεις σχετικά με το cluster, καθώς και την ανιχνεύση και την απόκριση σε διάφορα συμβάντα όπως για παράδειγμα, την εκκίνηση ενός καινούριου pod όταν τα replicas του deployment δεν είναι επαρκή. Τα στοιχεία του control plane μπορούν να εκτελεστούν σε οποιοδήποτε μηχάνημα του cluster, αλλά για λόγους απλότητας εκτελούνται στο ίδιο μηχάνημα χωρίς τα container της εφαρμογής [27]. Τα στοιχεία του control plane είναι τα εξής:

- kube-apiserver: Παρέχει το Kubernetes API. Ο API server είναι εκείνο το κομμάτι του προτύπου που παρέχει ένα API υψηλού επιπέδου σε τρίτες εφαρμογές διευκολύνοντας την αλληλεπίδραση με το cluster. Επικυρώνει και διαμορφώνει δεδομένα για τα αντικείμενα API που περιλαμβάνουν pods, services, replication controllers και άλλα. Εξυπηρετεί τις λειτουργίες REST και παρέχει στο front end την κοινή κατάσταση του cluster, μέσω της οποίας αλληλεπιδρούν όλα τα υπόλοιπα στοιχεία. Έχει σχεδιαστεί για οριζοντια κλιμάκωση, δηλαδή να δημιουργούνται παραπάνω instances. Ο διαχειριστής έχει τη δυνατότητα να εκτελεί πολλά instances και να εξισορροπεί την κίνηση μεταξύ τους [28].
- etcd: Είναι ένας συνεπής, αξιόπιστος και εξαιρετικά διαθέσιμος χώρος αποθήκευσης ζεύγους τιμής κλειδιού (key value pair) για τα δεδομένα στα οποία πρέπει να έχει πρόσβαση το cluster [29].
- kube-scheduler: Αναθέτει τα pods που δημιουργούνται στα worker nodes του cluster. Χρησιμοποιεί το API του Kubernetes για να εντοπίσει τα pods που δεν έχουν ανατεθεί σε κάποιο node. Μπορεί να τοποθετήσει πολλά pods σε ένα node αρκεί να υπάρχουν επαρκείς πόροι. Αν το node αποτύχει και δεν είναι διαθέσιμο, τότε η εφαρμογή θα αντιμετωπίσει προβλήματα, για αυτό ο Scheduler τοποθετεί τα pods σε πολλά nodes για να μην αντιμετοπίσει κάποιο πρόβλημα στο μέλλον. Η ανάθεση των pods σε nodes εξαρτάται από κάποιους παράγοντες. Οι παράγοντες που λαμβάνονται υπόψη για την ανάθεση είναι οι απαιτήσεις πόρων, οι περιορισμοί υλικού/λογισμικού, οι προδιαγραφές συνάφειας (affinity) και αντισυνάφειας (anti-affinity), οι παρεμβολές του φόρτου εργασίας και οι προθεσμίες [30].
- kube-controller-manager: Εκτελεί διαδικασίες ελεγκτή. Αναμένει από τον API server να τον ειδοποιήσει ότι δημιουργήθηκε ένα νέο αντικείμενο και έπειτα εκτελεί τις κατάλληλες διαδικασίες για να το ενεργοποιήσει. Κάθε ελεγκτής είναι μία ξεχωριστή διαδικασία, για την μείωση της πολυπλοκότητας, όλοι οι ελεγκτές μεταγλωτίζονται σε ένα ενιαίο binary αρχείο και εκτελενούνται ως μία διεργασία. Υπάρχουν πολλοί διαφορετικοί τύποι ελεγκτών. Μερικοί από αυτούς είναι οι εξής [31]:
  - node controller: Υπεύθυνος για την παρατήρηση και την απόκριση όταν ένα ή περισσότερα node αποτύχουν.
  - Job controller: Παρακολουθεί αντικείμενα εργασίας που αντιπροσωπεύουν μεμονωμένες εργασίες και στη συνέχεια δημιουργεί pods για να εκτελέσει αυτές τις εργασίες μέχρι την ολοκλήρωση.
  - ServiceAccount controller: Δημιουργεί προεπιλεγμένα ServiceAccounts για καινούρια (namespaces).
  - Cloud-controller-manager: Επιτρέπει στον διαχειριστή να συνδέσει το cluster στο API του cloud παρόχου και διαχωρίζει τα στοιχεία που αλληλεπιδρούν με την πλατφόρμα στο cloud με αυτά που αλληλεπιδρούν μόνο με το cluster. Εκτελεί ειδικούς ελεγκτές για τον πάροχο Cloud. Αν το περιβάλλον εργασίας είναι τοπικό ή εκμάθησης, τότε δεν υπάρχει αυτός ο ελεγκτής. Συνδυάζει πολλούς λογικά ανεξάρτητους βρόχους ελέγχου σε ένα binary αρχείο που εκτελείται ως μία ενιαία διεργασία [32].
- DNS: Χρησιμοποιείται ώστε το φόρτο εργασίας (workload) να εντοπίζει υπηρεσίες

(services) του cluster. Δημιουργεί εγγραφές DNS για pods και services, δίνοντας τη δυνατότητα επικοινωνίας με τα services με όνομα DNS αντί για διευθύνσεις IP. Το Kubernetes δημοσιεύει πληροφορίες σχετικά με τα pods και τα services που χρησιμοποιούνται για τη διαμόρφωση του DNS. Στη συνέχεια το Kubelet επεξεργάζεται το DNS των pods ώστε τα container να εντοπίζουν τα services χρησιμοποιώντας το όνομα και όχι την διεύθυνση IP.

Τα στοιχεία των worker nodes είναι τα παρακάτω:

- kubelet: Είναι ο πρωταρχικός agent που εκτελείται σε κάθε worker node. Επικοινωνεί με τον API server για να τον ενημερώσει για την κατάσταση του node και να αναγνωρίσει τι φόρτο εργασίας μπορεί να ανατεθεί σε αυτό. Επίσης επικοινωνεί με το container runtime, όπως είναι το Docker, για να διασφαλίσει ότι τα φορτία εργασίας έχουν εκτελεστεί και είναι υγιή.
- Kube-proxy: Είναι ένας διακομιστής μεσολάβησης δικτύου που εκτελείται σε κάθε node. Διατηρεί κανόνες δικτύου, οι οποίοι επιτρέπουν την δικτυακή επικοινωνία μεταξύ των nodes αλλά και μεταξύ των pods και συνδέσεων εκτός του cluster. Χρησιμοποιεί το επίπεδο φιλτραρίσματος πακέτων του λειτουργικού συστήματος, αν υπάρχει, αλλίως προωθεί το ίδιο την κίνηση.
- Container runtime: Είναι βασικό στοιχείο που επιρέπει το Kubernetes να διαχειρίζεται και να εκτελεί containers αποτελεσματικά. Είναι υπεύθυνο για τον έλεγχο της εκτέλεσης και της διαχείρισης του κύκλου ζωής των container. Μέσω του container runtime το Kubernetes μπορεί να εκτελεί και να διαγράφει contains διασφαλίζοντας την ομαλή λειτουργία τους. Τα πιο γνωστά container runtimes που υποστηρίζονται και χρησιμοποιούνται από το Kubernetes είναι τα, docker [33], containerd [34], CRI-O [35].

Τα αντικείμενα (objects) είναι σταθερές οντότητες του Kubernetes. Χρησιμοποιούνται για την αναπαρασταση της κατάστασης του cluster. Τα objects μπορούν να περιγράψουν τα εξής:

- Ποιες containerized εφαρμογές εκτελούνται και σε ποια nodes.
- Τους διαθέσιμους πόρους για αυτές τις εφαρμογές.
- Τις πολιτικές συμπεριφοράς των εφαρμογών, για παράδειγμα, την πολιτική επανεκκίνησης, αναβαθμίσεις και ανοχής λαθών.

Μόλις δημιουργηθεί ένα object το Kubernetes συνεχώς εργάζεται για να διασφαλίσει ότι αυτό θα υπάρχει. Με τη δημιουργία του, ο διαχειριστής ορίζει στο Kubernetes την επιθυμητή κατάσταση του cluster. Αξιοποιώντας το API μέσω της εντολής kubectl ο χρήστης μπορεί να εργαστεί με τα objects, να τα δημιουργήσει, να τα τροποποιήσει ακόμα και να τα διαγράψει. Objects ή resources μπορεί να είναι, pod, deployment, ReplicaSets, StatefulSet, DaemonSets, PersistentVolume, Service, Namespaces, ConfigMaps, Secrets, Job.

Το pod είναι η μικρότερη εκτελέσιμη υπολογιστική μονάδα που υπάρχει στο Kubernetes. Είναι ένα περιβάλλον το οποίο στεγάζει μία ομάδα από containers με κοινό χώρο αποθήκευσης και δικτυακούς πόρους αλλά και με οδηγίες/προδιαγραφές για το πως να εκτελέσουν τα containers. Τα pods μπορούν να τρέχουν είτε ένα container είτε πολλά μαζί. Ομαδοποιώντας δύο ή παραπάνω containers μαζί, επιτυγχάνεται

η ταχύτερη επικοινωνία τους και τα δεδομένα μοιράζονται πιο εύκολα. Η φύση τους είναι εφήμερη, αυτό σημαίνει ότι δεν έχουν εγγυημένη η μακροπρόθεσμη διάρκεια ζωης. Μπορούν να δημιουργηθούν, να καταστραφούν και να αναδημιουργηθούν ανά πάσα στιγμή. Τέλος χρησιμοποιούνται για την εκτέλεση, την κλιμάκωση και την διαχείρηση των containerized εφαρμογών του cluster. Ο διαχειριστής μπορεί να χρησιμοποιήσει πόρους φόρτου εργασίας είτε είναι deployment είτε είναι StatefulSet είτε έιναι DaemonSet για να δημιουργήσει και να διαχειριστεί πολλά pods. Ένας controller χειρίζεται την αντιγραφή, την κυκλοφορία αλλά και την αυτόματη επιδιόρθωση σε περίπτωση που το pod αποτύχει. Για παράδειγμα, αν ένα node σταματήσει να λειτουργεί, τότε ένας controller δημιουργεί έναν αντικαταστάτη pod και ο scheduler το τοποθετεί σε ένα node που λειτουργεί. Οι controllers χρησιμοποιούν τα pod template για να δημιουργήσουν τα pods. Τα pod template είναι οι προδιαγραφές για τη δημιουργία των pods και συμπεριλαμβάνονται στους πόρους φόρτου εργασίας (deployments, jobs, DaemonSets). Όταν γίνουν αλλαγές σε ένα template τότε ο controller δημιουργεί νέα pods βάση το ενημερωμένο template διαγράφοντας τα παλιά. Όπως προαναφέρθηκε, τα pods επιτρέπουν την κοινή χρήση δεδομένων και την επικοινωνία μεταξύ των container που στεγάζουν. Ένα pod μπορεί να καθορίσει ένα σύνολο από κοινούς χώρους αποθήκευσης ώστε όλα τα containers να έχουν πρόσβαση σε αυτούς, δίνοντας τους την δυνατότητα να μοιράζονται δεδομένα. Επίσης οι χώροι αυτοί επιτρέπουν την διατήρηση των μόνιμων δεδομένων (Persistent Data) σε περίπτωση που κάποιο ζονταινερ επανεκκινηθεί.

Σε κάθε pod ανατίθεται μία διεύθυνση IP. Κάθε container μέσα σε ένα pod χρησιμοποιεί το δικτυακό χώρο ονομάτων συμπεριλαμβανομένης της IP διεύθυνση και τις δικτυακές πόρτες(network ports). Μέσα στο pod και μόνο τότε, τα containers επικονωνούν μεταξύ τους χρησιμοποιώντας το localhost, με containers από διαφορετικά pods επικοινωνούν με την IP του pod που στεγάζει τα συγκεκριμένα container [36].

Τα στατικά pods (Static pods) διαχειρίζονται απευθείας από το kubelet. Είναι ορατά στον API server αλλά δεν μπορούν να ελεγχθούν από αυτόν. Δεν μπορεί ο διαχειριστής να τα τροποποιήσει χρησιμοποιώντας το kubectl, για να επιτευχθεί κάτι τέτοιο πρέπει να τροποποιήσει το ανάλογο αρχείο διαμόρφωσης του static pod. Το kubelet παρατηρεί τον φάκελο όπου βρίσκεται το αρχείο αυτό και εκτελεί τις κατάλληλες αλλαγές βασιζόμενο στο αρχείο διαμόρφωσης (config file). Αυτού του είδους pods είναι χρήσιμα για πολύ σημαντικά στοιχεία του cluster τα οποία πρέπει να τρέχουν πάντα στο node [37].

Τα pods έχουν έναν κύκλο ζωής (lifecycle). Ξεκινώντας από το στάδιο της αναμονής (pending state), στη συνέχεια προχωράνε στο στάδιο εκτέλεσης (running) αν έστω ένα από τα container εκκινήθεί. Στο στάδιο επιτυχίας (succeeded state), εφόσον όλα εξελιχθούν ομαλά, ή στο στάδιο αποτυχίας (failed state) σε περίπτωση που παρουσιαστεί κάποιο πρόβλημα με κάποιο container. Μόλις χρονοδρομολογηθεί ένα pod και έχει δεσμευθεί σε ένα node, το Kubernetes προσπαθεί να το εκτελέσει, αν αποτύχει να το εκτελέσει,για παράδειγμα, αν ο node σταματήσει να λειτουργεί πριν την εκτέλεση του pod, τότε δεν εκτελείται ποτέ [38].

Το Deployment είναι ένα αντικείμενο το οποίο χρησιμοποιείται για τη διαχείριση του κύκλου ζωής ενός ή και περισσότερων παρόμοιων pods. Επιτρέπει τη δηλωτική διαχείρηση της επιθυμητής κατάστασης της εφαρμογής, όπως ο αριθμός των αντίγραφων και οι πόροι που θα χρησιμοποιηθούν. Το δηλωτικά σημαίνει πως ο διαχειριστής περιγράφει την επιθυμυτή κατάσταση της εφαρμογής και το Kubernetes βρίσκει και εκτελεί τα βήματα για να φτάσει την εφαρμογή στην κατάσταση αυτή. Οταν δημιουργείται ένα deployment, ο χρήστης παρέχει ένα pod template, το οποίο καθορίζει τη διαμόρφωση

(congirutaion των pods τα οποία θα διαχειρίζεται το deployment. Έπειτα δημιουργεί τα pods αυτά, χρησιμοποιώντας ένα ReplicaSet. Το ReplicaSet είναι υπεύθυνο για τη δημιουργία και την κλιμάκωση των pods και διασφαλίζει ότι αν ένα pod αποτύχει, θα ανιτκατασταθεί. Όταν ενημερώνεται ένα deployment, ενημερώνεται και το ReplicaSet και στη συνέχεια τα ανάλογα pods [39].

Ένα StatefulSet χρησιμοποιείται για την διαχειρήση των εφαρμογών με διατήρηση της καταστασής τους (stateful applications). Οι stateful applications ειναι εφαρμογές οι οποίες αποθηκεύουν και παρακολουθούν τα δεδομένα. Όλες οι βάσεις δεδομένων είναι παραδείγματα τέτοιων εφαρμογών. Ενώ οι εφαρμογές χωρίς διατήρηση κατάστασης (stateless applications) δεν διατηρούν τα δεδομένα τους. Το StatefulSet διασφαλίζει ότι κάθε pod αναγνωρίζεται μοναδικά με τη χρήση ενός αριθμού, ξεκινώντας από το μηδέν. Όταν ένα pod πρέπει να αντικτασταθεί, τότε ένα νέο pod με τον ίδιο αριθμό προστείθεται. Αυτό διασφαλίζει ότι το νέο pod θα έχει την ίδια μοναδική ταυτότητα και θα είναι προσαρτημένο στον ίδιο μόνιμο αποθηκευτηκό χώρο με το προηγούμενο pod, έτσι διατηρούνται τα δεδομένα και η κατάσταση των pods κατα τη διάρκεια των αλλαγών [40].

Ένα DaemonSet επιβεβαιώνει ότι ένα αντίγραφο ενός pod εκτελείται σε έναν ή περισσότερους nodes. Είναι χρήσιμα για χαμηλού επιπέδου services όπως είναι οι agents καταγραφής και παρακολούθησης. Ο DaemonSet controller δημιουργεί το pod και προσθέτει στο αρχείο διαμόρφωσης του pod μία γραμμή κώδικα ώστε το pod να ταιριάξει με το σωστό node. Αφότου δημιουργηθεί, ο scheduler το αναθέτει στο node. Σε περίπτωση που δεν μπορεί να ταιριάξει στο node, ο scheduler ίσως αφαιρέσει κάποια από τα ήδη υπάρχοντα pods, βασιζόμενος στην προτεραιότητα του καινούριου [41].

Ο μόνιμος αποθηκευτικός χώρος (Persistent volume) προσαρτάται στα pods. Όταν διαγραφεί ένα pod, ο χώρος αυτός παραμένει με τα δεδομένα που είναι αποθηκευμένα σε αυτόν. Υπάρχουν διάφορα είδη τέτοιου χώρου, όπως οι τοπικοί δίσκοι, η δικτυακή αποθήκευση (Network Attached Storage - NAS) και το cloud. Συνήθως παρέχεται χειροκίνητα από τον διαχειριστή ή χρησιμοποιώντας Storage Classes. Το Storage Class παρέχει στους διαχειριστές έναν τρόπο για να περιγράψουν τι είδους χώρο αποθήκευσης παρέχουν στον cluster [42].

Ένα service, είναι ένας τρόπος για τον διαχειριστή να έχει πρόσβαση σε ένα σύνολο από pods τα οποία παρέχουν κάποια λειτουργικότητα. Εκθέτει μία εφαρμογή στο δίκτυο, ώστε να έχουν πρόσβαση σε αυτή οι χρήστες, ακόμη και αν το φόρτο εργασίας της είναι κατανεμημένο σε πολλά nodes. Επίσης ένα service μπορεί να είναι και εξισορροπιστής φορτίου (load balancer) για μία ομάδα από pods. Υπάρχουν τέσσερις τύποι service [43].

- clusterIP: Επιτρέπει την επικοινωνία των pods εσωτερικά του cluster και είναι ο προεπιλεγμένος τύπος service.
- nodePort: Επιτρέπει την εξωτερική πρόσβαση σε μία εφαρμογή που εκτελείται στο cluster. Εκθέτει μία δικτυακή θύρα σε επίπεδο node, η κίνηση που διέρχεται από αυτή τη θύρα, προωθείτε στην αντίστοιχη υπηρεσία η οποία στη συνέχεια τη δρομολογεί στην εφαρμογή ιστού.
- Loadbalancer: Επιτρέπει την εξωτερική κίνηση να διανέμεται αποτελεσματικά εσωτερικά του cluster. Αναθέτει μια δημόσια IP διεύθυνση ή ένα DNS όνομα στο service κάνοντας το προσβάσιμο εξωτερικά του cluster. Οδηγεί την εξωτερική κίνηση στη θύρα του service το οποίο στη συνέχεια τι διανέμει στα ανάλογα pods.

Στη συνέχεια, τα namespaces παρέχουν έναν μηχανισμό για την απομόνωση τον πόρων στο cluster. Σε κάθε namespace μπορούν εκτελούνται διάφορα objects τα οποία θα υπάρχουν μόνο σε αυτό και σε κανένα άλλο. Με τη χρήση των namespace εξαλείφεται το ρίσκο επηρεασμού από πόρους σε διαφορετικά namespaces [44].

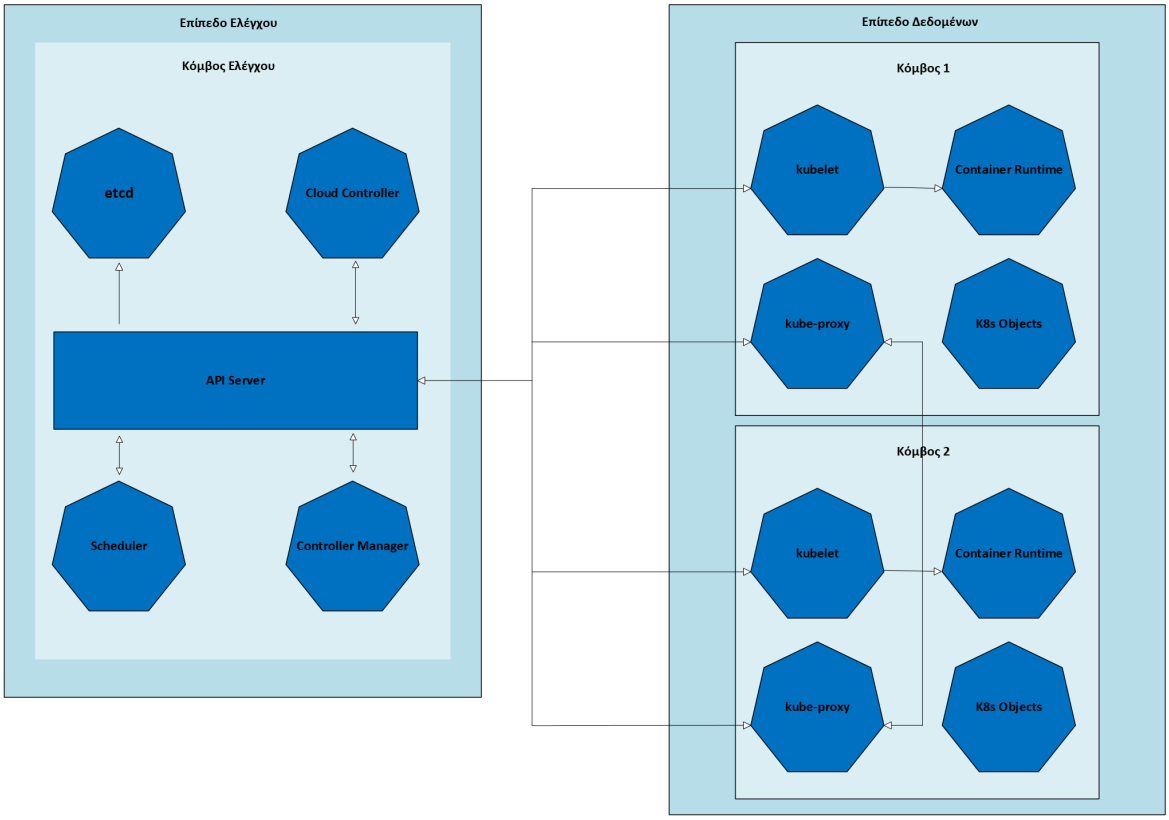
Το configmap είναι ένα API object για την αποθήκευση πληροφοριών, που δεν είναι ευαίσθητες, σε key value pair [45].

Το secret είναι ένα object που περιέχει ευσαίσθητα δεδομένα, όπως είναι κωδικοί ή ένα διακριτικό (token). Αυτές οι πληροφορίες, διαφορετικά θα μπορούσαν να τοποθετηθούν σε ένα pod ή σε ένα container image. Οπότε με τη χρήση του secret δεν χρειάζεται να γίνει κάτι τέτοιο, καθώς δημιουργούνται ξεχωριστά από τα pods τα οποία τα χρησιμοποιούν, υπάρχει μικρότερος κίνδυνος της έκθεσης του. Είναι παρόμοια με τα configmaps με τη διαφορά ότι περιέχουν ευαίσθητες πληροφορίες [46].

Το job δημιουργεί ένα ή παραπάνω pod, τα οποία χρησιμοποιούνται για να εκτελέσουν εργασίες στο cluster. Εξασφαλίζει ότι ένας αριθμός από pods ολοκληρώνει επιτυχώς μία εργασία πριν καταγράψει το job ως ολοκληρωμένο. Διαγράφοντας ένα job, θα διαγραφούν και τα pods που του αναλογούν, συνήθως είναι ένα μόνο. Αναβάλλοντας ένα job, θα διαγραφούν τα ενεργά pods μέχρι αυτό να συνεχιστεί. Υπάρχουν τρία είδη jobs:

- Τα non-parallel jobs σχεδιάζονται για να εκτελέσουν μία εργασία μέχρι την ολοκληρωσή της. Είναι το πιο απλό είδος αλλά και το πιο αξιόπιστο. Ένα non-parallel job μπορεί να είναι ένα script που ο διαχειριστής εκτελεί μία φορά τη μέρα. Το non-parallel job θα χρονοδρομολογήσει το pod, θα εκτελέσει το script και θα εξασφαλίσει την σωστή ολοκλήρωση της εργασίας [47].
- Το Parallel Jobs with a Fixed Completion Count, δημιουργεί πολλά pod τα οποία εκτελούνται συγχρόνως για να ολοκλρώσουν μία εργασία. Όταν τα pods φέρουν επιτυχώς εις πέρας την εργασία τότε το job θεωρείται ολοκληρωμένο. Για παράδειγμα, στην επεξεργασία μεγάλων datasets, όπου τα δεδομένα διασπόνται σε τμήματα και επεξεργάζονται συγχρόνως από πολλά pods.
- Τα Parallel Jobs with a Work Queue αναλόγως του φόρτου εργασίας, κλιμακώνουν δυναμικά τα pods. Για παράδειγμα, εισέροχμενα requests από ένα web application όπου τα pods κλιμακώνονται δυναμικά βάση της απαίτησης που υπάρχει.

Τέλος, οι ετικέτες (labels) και οι επιλογείς (selectors) είναι η μέθοδος κατά την οποία ομαδοποιούνται τα objects στο Kubernetes. Τα lables είναι key value ζεύγοι που υποδεικνύουν τις ιδιότητες που έχει το κάθε object. Ο selector βοηθάει στην εύρεση του ή στην ομαδοποιήση των objects, επιτρέποντας έτσι μία πιο αποδοτική διαχείριση του συστήματος [48].




Σχήμα 3.2: Στοιχεία του Kubernetes cluster

## 3.2 Δικτύωση σε Περιβάλλον Kubernetes

Ένα από τα ζητήματα ενός cluster είναι η δικτυωσή των containers, δηλαδή η επικονωνία μεταξύ τους. Η Διεπαφή Δικτύωσης Container (Container Networking Interface - CNI) αναλαμβάνει αυτη τη δουλειά. Είναι ένα project το οποίο δημιουργήθηκε από το CNCF με σκοπό τη δικτύωση των container στο λειτουργικό σύστημα Linux [49]. Τα container όπως και τα pods είναι εφήμερα, κάνοντας έτσι τη διαδικασία δικτυωσής τους περίπλοκη. Το CNI απλουστεύει αυτή τη διαδικασία και την κάνει αυτόματη μέσω κάποιου framework. Αποτελείται πρώτον, από μία βιβλιοθήκη η οποία διαχειρίζεται τους πόρους δικτύου και τις διακτυακές διεπαφές των container, καθιστώντας έτσι πιθανή την επικονωνία μεταξύ τους εσωτερικά αλλά και εξωτερικά του cluster. Δεύτερον, από κανόνες,πρότυπα και πρωτόκολλα που ορίζουν πως τα container runtime συνεργάζονται με κάποια δικτυακά plugins για τη διαχείρηση της δικτύωσης των container. Το CNI διασφαλίζει ότι η δικτύωση θα είναι δυναμική, ασφαλής και ευπροσάρμοστη με την εφήμερη φύση των container [50].

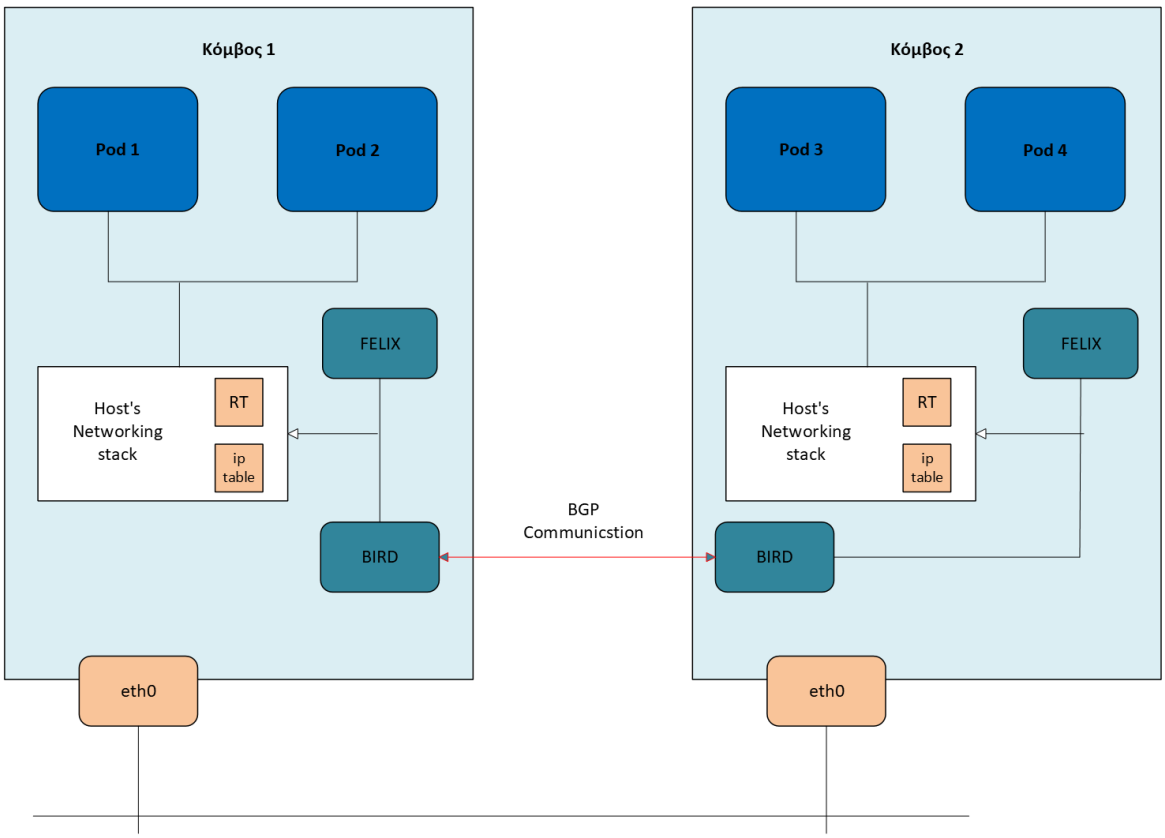
Ο σκοπός του CNI είναι να αφαιρέσει την διαδικασία δικτύωσης από τις ευθύνες του διαχειριστή. Μέσω των plugins αναθέτει διευθύνσεις IP στα container και δρομολογεί την δικτυακή κίνηση τόσο μεταξύ τους όσο και με εξωτερικά δίκτυα. Ακολουθεί τους παρακάτω κανόνες. Κάθε pod πρέπει να επικοινωνεί με ένα άλλο pod μέσα στο cluster χωρίς τη χρήση της Μετάφρασης Διεύθυνσης Δικτύου (Network Address Translation - NAT). Κάθε node πρέπει να επικονωνεί με κάθε pod χωρίς πάλι τη χρήση του NAT, ώστε να παρακολουθούνται ή να ελέγχονται για τυχόν προβλήματα [51]. Αποφεύγοντας την χρήση του NAT, διότι προκαλεί περαιτέρω επιβάρυνση στους πόρους του μηχανήμα-

τος, το Kubernetes παρέχει άμεση και αποδοτική επικοινωνία μεταξύ των στοιχείων του cluster.

Προσφέρει Διαχείριση Διεύθυνσης IP (IP Address Management - IPAM), δηλαδή, αναθέτει και απαλευθερώνει διευθύνσεις IP για τα pods, εξασφαλίζοντας έτσι, ότι κάθε pod θα λάβει ξεχωριστή IP ώστε να επιτευχθεί σωστή επικονωνία μεταξύ των containers. Επιτρέπει στον διαχειριστή να ορίσει Πολιτικές Δικτύου (Network Policies) για την επικοινωνία των pods μεταξύ τους ή και με το περιβάλλον εξωτερικά του cluster. Τα network policies δίνουν τη δυνατότητα στον διαχειριστή να ορίσει κανόνες για το ποια δικτυακή κίνηση επιτρέπεται να κινείται μεταξύ των pods. Να διασφαλίσει ότι τα ευαίσθητα σημεία της εφαρμογής είναι απομωνομένα από άλλα κομμάτια της. Τέλος να εφαρμόσει ελέγχους ασφάλειας στο επίπεδο δικτύου [52].

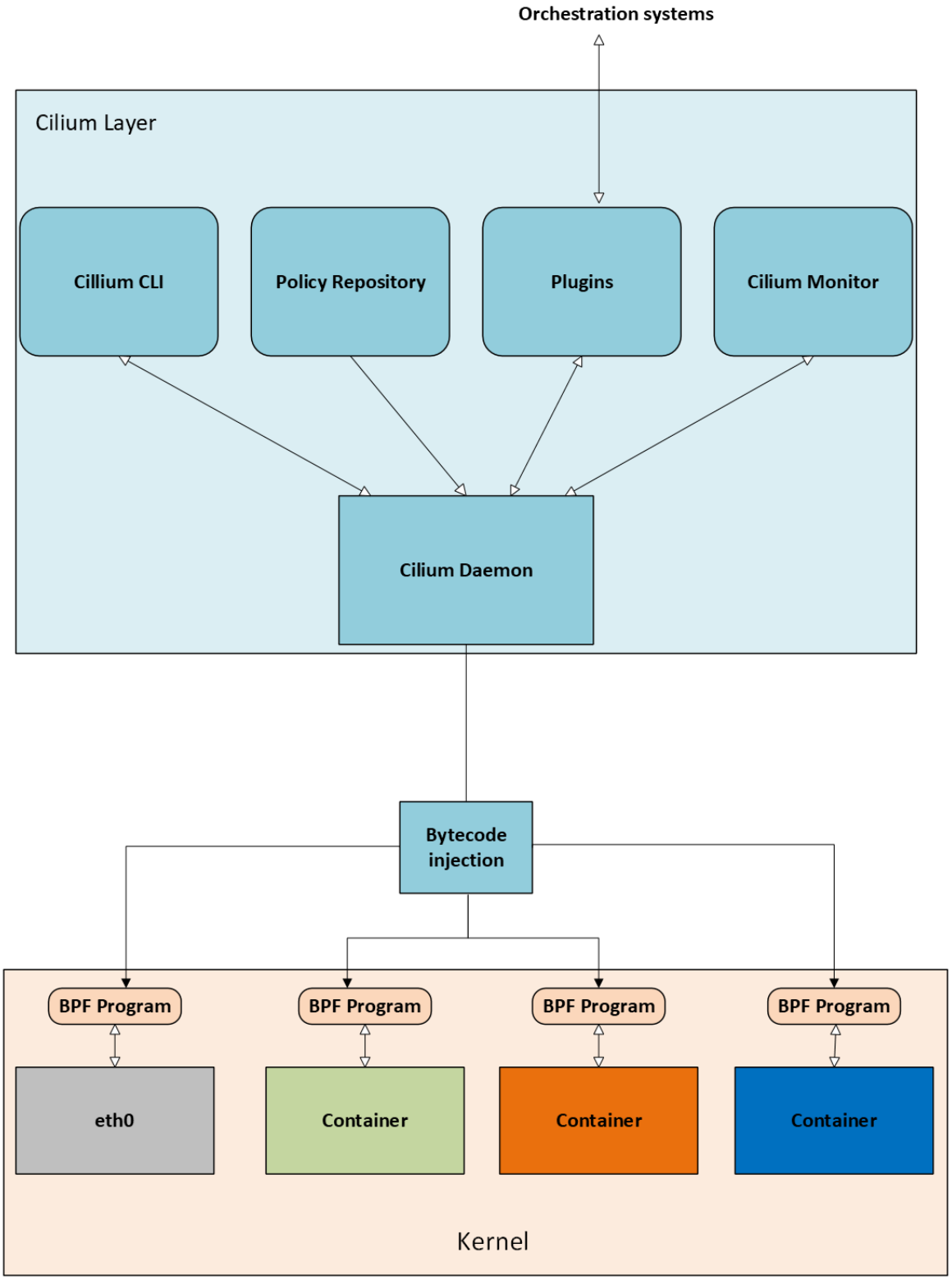
ινδεντ Τα CNI plugins που υποστηρίζουν network policies, συνήθως εκμεταλλεύονται λειτουεγίες του Linux Kernel όπως είναι το iptables και το ebpf. Τέλος, με τη δημιουργία παραπάνω Pod, υπάρχει αύξηση στη δικτυακή κίνηση, οπότε το plugin πρέπει να διαχειριστεί αυτή την κατάσταση για να μη δημιουργηθεί κάποιο bottleneck. Για την αντιμετώπιση του παραπάνω προβλήματος, τα plugins έχουν τη δυνατότηα να κλιμακώνονται αναλόγως του μεγέθους του cluster. Κάθε περιβάλλον που χρησιμοποιεί το Kubernetes έχει τις δικές του ανάγκες, ο διαχειριστής έχει την δυνατότητα επιλογής από μία πληθώρα CNI plugins για την ικανοποίηση των αναγκών αυτών [53]. Κάποια από αυτά τα plugins είναι τα, Calico [54], Cilium [55], Flannel [56], Antrea [57], KubeOVN [58].

Το Calico προσφέρει μία αξιόπιστη και επεκτάσιμη λύση για τη δικτύωση των container. Είναι ένα από τα πιο γνωστά και πιο χρησιμοποιημένα CNI plugin για το Kubernetes. Λειτουργεί σε ένα flat Layer 3 δίκτυο, αναθέτοντας μοναδικές διευθύνσεις IP σε κάθε pod. Με αυτό τον τρόπο επιτρέπει την εύκολη αντιμετώπιση προβλημάτων. Χρησιμοποιεί το Πρωτόκολλο Πύλης Διασύνδεσης (Border Gateway Protocol - BGP) για τη δρομολόγηση της δικτυακής κίνησης. Υποστηρίζει τα Network Policies του Kubernetes ώστε να υπάρχει έλεγχος στην επικοινωνία μεταξύ των pods. Προσφέρει διάφορες μεθόδους ενθυλάκωσης όπως είναι, IP-in-IP και VXLAN για να διευκολύνει την επικοινωνία μεταξύ των pod. Επίσης, επιτρέπει στα pods του cluster να επικοινωνούν με τον έξω κόσμο χρησιμοποιώντας NAT, δηλαδή με μία κοινή δημόσια IP διεύθυνση. Με την προσθήκη του Calico στο cluster, εγκαθηστώνται daemons, μεταξύ άλλων τα δύο βασικά στοιχεία είναι τα Bird και Felix και χρησιμοποιούνται για την προώθηση και λήψη δικτυακής πληροφορίας μεταξύ των nodes και τροποποιούν τους πίνακες δρομολόγησης (routing tables). Τέλος το Felix εγκαθιστά Λίστες Ελέγχου Πρόσβασης (Access Control Lists - ACLs) στο Kernel του Linux για να επιτρέψει μόνο έγκυρη κίνηση να φτάσει τα pods [59].




Σχήμα 3.3: Αρχιτεκτονική Calico

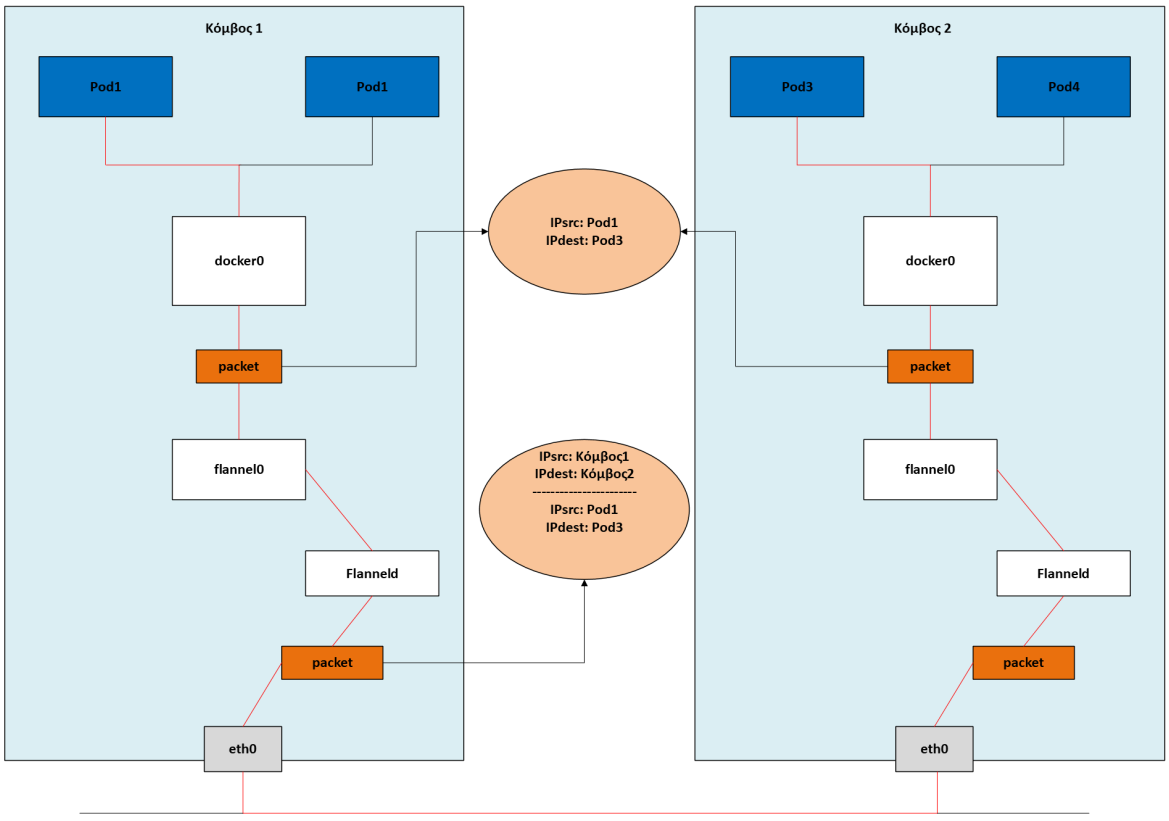
Το Cilium είναι ένα open source project το οποίο παρέχει δικτύωση, ασφάλεια και παρακολούθηση στο Kubernetes και άλλες πλατφόρμες ενορχήστρωσης πόρων. Στη βάση του Cilium, βρίσκεται η τεχνολογία eBPF, μιας τεχνολογίας που επιτρέπει την δυναμική εκφόρτωση προγραμμάτων C σε επίπεδο πυρήνα λειτουργικού συστήματος με ασφαλή τρόπο. Τα προγράμματα αυτά μπορούν να εκτελούνται μετά από συγκεκριμένες κλήσεις συναρτήσεων συστήματος (system calls). Χάρη σε αυτό, το Cilium είναι σε θέση να παρέχει υψηλής απόδοσης δίκτυα, προχωρημένο Load Balancing, διαφανή κρυπτογράφηση και πολλά άλλα. Περιλαμβάνει τέσσερα στοιχεία κλειδιά, το Cilium agent, το Cilium Client, το Cilium Operator και το Cilium CNI plugin. Το agent εκτελείται σε όλα τα nodes του cluster και είναι υπεύθυνο για την εγκαθίδρυση συνδέσεων με το Kubernetes API και συντυρεί πολιτικές δικτύου και ασφάλειας. Το client, το οποίο είναι ομαδοποιημένο με το agent, επιθεωρεί και διαχειρίζεται την κατάσταση του agent του και προσφέρει απευθείας σύνδεση με τους eBPF πίνακες. Το operator διαχειρίζεται κεντρικά τις εργασίες του cluster. Το CNI plugin ενεργοποιείται απο το Kubernetes κατά τη διαδικασία χρονοδρομολόγησης των pod ή κατά την απενεργοποιήση κάποιου node. Αλληλεπιδρά με το Cilium API του κάθε node για να διαμορφωποιήσει τα απαραίτητα datapaths για τη δικτύωση, το load balancing και τα network policies [60].




Σχήμα 3.4: Αρχιτεκτονική Cilium

Το Flannel είναι ένα από τα πιο απλά CNI που μπορούν να εγκατασταθούν στο cluster. Εκτελεί ένα μικρό εκτελέσιμο διαδικό αρχείο εν ονόματι flanneld σε κάθε host και είναι υπεύθυνο για την εκχώρηση ενός υποδικτύου για τον κάθε host. Χρησιμοποιεί είτε το Kubernetes API είτε απευθείας το etcd για την αποθήκευση της δικτυακής διαμόρ-

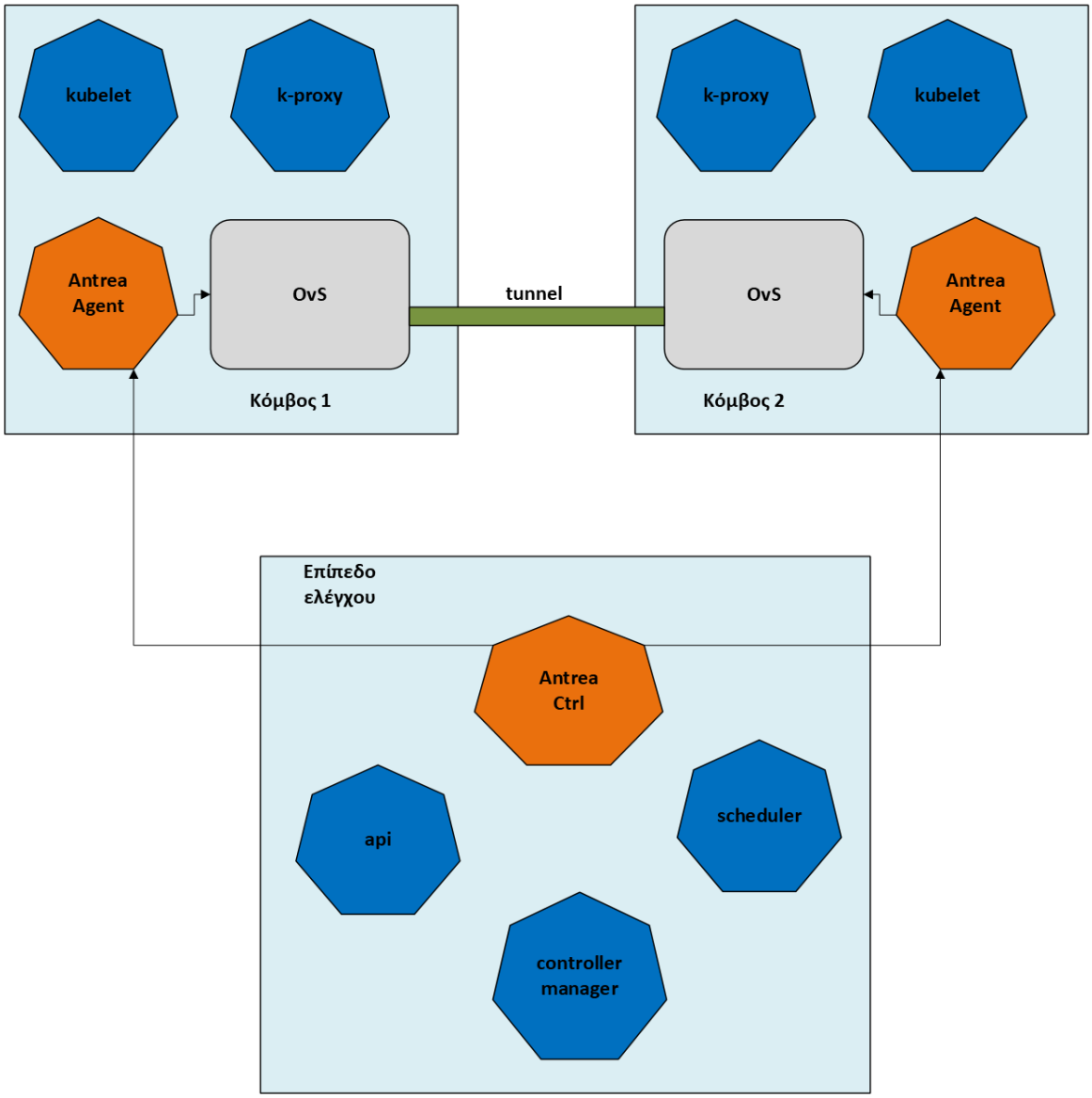
φωσης, των υποδικτύων και δευτερεύοντα δεδομένα όπως είναι η δημόσια διεύθυνση IP του host. Για να υπάρχει επικοινωνία μεταξύ των pod, αξιοποιείται η ενθυλάκωση VXLAN η οποία είναι η πιο συνηθισμένη. Δημιουργείται ένα δίκτυο το οποίο βρίσκεται πάνω από αυτό του host και ανατέθεται σε κάθε node μια ομάδα διευθύνσεων IP. Τα pod που βρίσκονται σε κάθε node λαμβάνουν μία IP από αυτη την ομάδα. Τέλος χρησιμοποιεί το προκαθορισμένο IPAM που παρέχεται από το framework. Τα containers επικοινωνούν μεταξύ τους μέσω της γέφυρας docker0 η οποία διαμορφωποιείται από το flannel. Αν ένα πακέτο πρέπει να δρομολογηθεί σε κάποιο άλλο node, το kernel προωθεί το πακέτο στην διεργασία flanneld η οποία με τη σειρά της, επικοινωνεί με το etcd για να εντοπίσει σε ποιό node πρέπει να αποσταλλεί το πακέτο [61].




Σχήμα 3.5: Αρχιτεκτονική Flannel

Το Antrea έχει σχεδιαστεί για το Kubernetes. Επικεντρώνεται στη δικτύωση και στην ασφάλεια του cluster. Αξιοποιεί το Open vSwitch ως το επίπεδο προώθησης δεδομένων. Το Open vSwitch είναι ένας υψηλής απόδοσης προγραμματιζόμενος εικονικός μεταγωγέας πακέτων που υποστηρίζει και Linux και Windows. Το Antrea δημιουργεί ένα Deployment το οποίο εκτελεί το Antrea Controller και ένα DaemonSet το οποίο περιέχει δύο container για να εκτελέσει τα Antrea Agent και OVS daemons. Επίσης το DaemonSet περιέχει και ένα container εκκίνησης το οποίο εγκαθιστά το CNI plugin εν ονόματι antrea-cni στο node και είναι υπεύθυνο για την εκφόρτωση του OVS ως μικροεφαρμογή kernel και την διασύνδεσή του ως CNI plugin. Με την εγκατασταση του, δημιουργεί μία OVS γέφυρα για τη επικοινωνία μεταξύ των pods. Για την επικοινωνία τους εντός του node, τα πακέτα δεδομένων προωθούνται απευθείας μέσω της OVS γέφυρας. Για την επικοινωνία μεταξύ των nodes, τα πακέτα δεδομένων εν-

θυλακώνονται και αποστέλλονται μέσω ενός τούνελ και απόενθυλακώνονται στο node προορισμού και τέλος καταφτάνουν στο ανάλογο pod. Το Antrea Controller είναι υπεύθυνο για την επίβλεψη των πόρων που καταναλώνει το Kubernetes, τον υπολογισμό των πολιτικών δικτύου και τα διανέμει στα Antrea Agents. Το Antrea Agent διαχειρίζεται την γέφυρα του OVS σε κάθε node, τη δικτύωση των pods και ενισχύει τις πολιτικές δικτύου [62].




Σχήμα 3.6: Αρχιτεκτονική Antrea

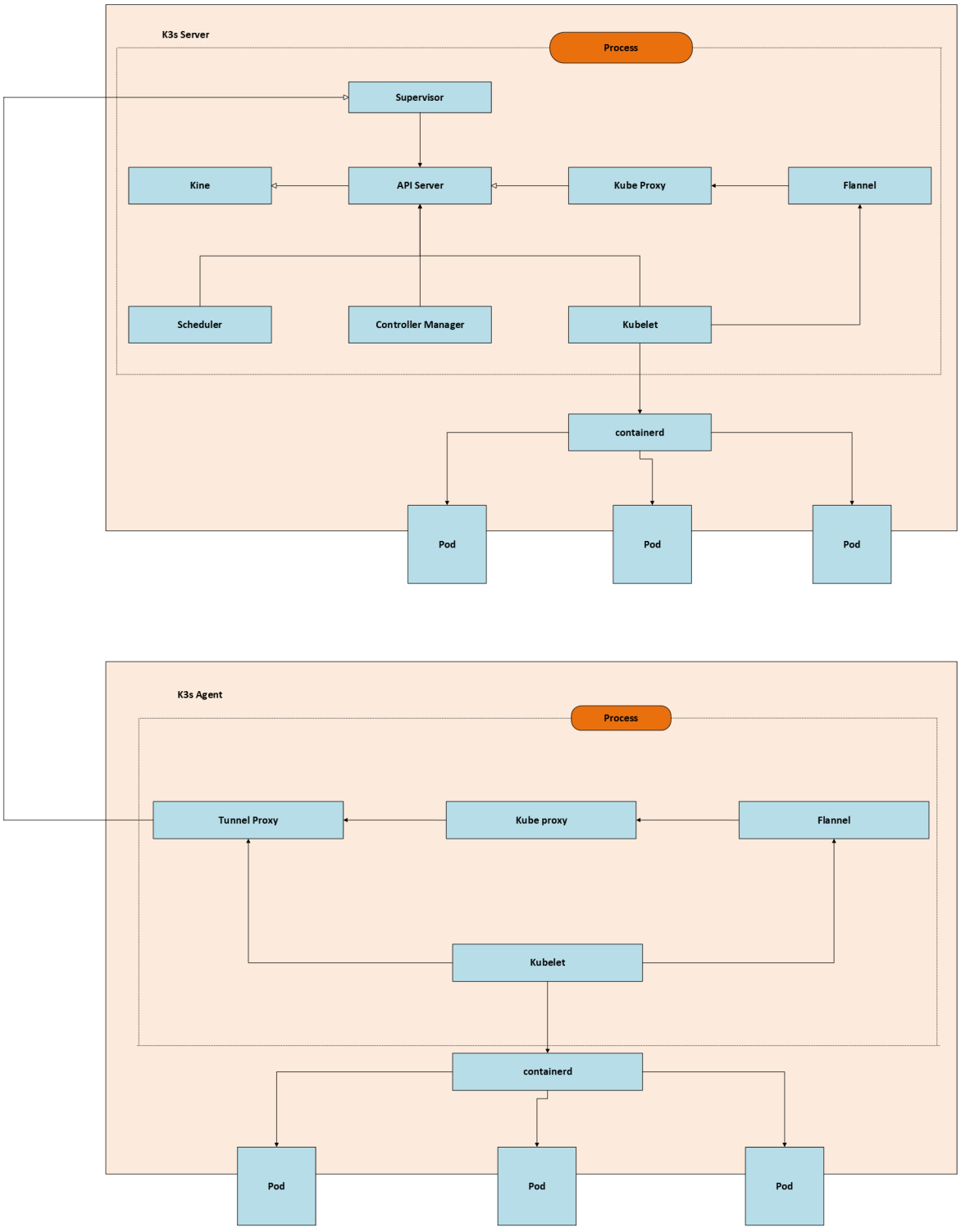
## 3.3 Ενορχηστρωτές container συμβατοί με το πρότυπο Kubernetes

Αναλόγως με τις ανάγκες του κάθε περιβάλλοντος υπάρχουν διάφοροι ενορχηστρωτέςcontainer συμβατοί με το Kubernetes. Μέρικοι από αυτούς είναι ο K3s, ο MicroK8s και το Vanilla Kubernetes.

Το K3s είναι ένας μη απαιτητικός ενορχηστρωτής container που δημιουργήθηκε

από το Rancher Labs και είναι πιστοποιημένο από το CNCF. Διανέμεται ως ένα δυαδικό αρχείο και περιέχει μόνο τα βασικά στοιχεία του Kubernetes. Τα στοιχεία αυτά είναι τα, kube-apiserver, kube-scheduler, kubelet, kube-controller-manager και kube-proxy. Αυτά τα στοιχεία ο διαχειριστής μπορεί να τα αντικαταστήσει με άλλα της επιλογής του. Χρσηιμποιεί την SQLite αντί για etcd για την αποθήκευση των δεδομένων του cluster, παρόλα αυτά υποστηρίζει το etcd, MySQL και PostgreSQL για μεγαλύτερης κλίμακας cluster. Επίσης χρησιμοποιεί το containerd ως container runtime διότι είναι πιο ελαφρύ από το συνηθισμένο runtime, το Docker. Η δικτυωσή του επιτυγχάνεται με τη χρήση ενός CNI plugin το Flannel. Το K3s συμπεριλαμβάνει το Traefik ως ingress controller. Το ingress επιτρέπει την HTTP και HTTPS κίνηση εξωτερικά του cluster να δρομολογείται στα services, για να είναι εφικτό αυτό πρέπει να υπάρχουν οι ingress controllers. Πρέπει στο cluster να υπάρχει τουλάχιστον ένας.

Η αρχιτεκτονική του είναι σχετικά απλή. Αποτελείται από το server node το οποίο περιέχει τα στοιχεία του control plane και τον χώρο αποθήκευσης και από το agent node χωρίς τα προαναφερθέντα στοιχεία. Και τα δύο nodes περιέχουν τα, kubelet, container runtime και το CNI. Επίσης, τα στοιχεία του server ή και του agent συμπεριλαμβάνονται σε μία διεργασία το καθένα. Τέλος, αν υπάρχει ένα single node cluster, ο server και ο agent εκτελούνται ως μία διεργασία [63].




Σχήμα 3.7: Αρχιτεκτονική K3s

Το MicroK8s δημιουργήθηκε από την Canonical και είναι ένας ελαφρύς ενορχηστρωτής container συμβατός με το Kubernetes. Περιέχει όλα τα βασικά στοιχεία του Kubernetes, δηλαδή τα, Kubernetes API server, scheduler, controller manager, kubelet, container runtime (containerd) και kube-proxy, έτσι επιτρέπει την γρήγορη εκκίνηση του και χαμηλή κατανάλωση πόρων. Τα βασικά στοιχεία και οι υπηρεσίες συσκευάζονται σε ένα snap πακέτο, το snap είναι πακέτο μίας εφαρμογής και των ε-

ξαρτησεών της που μπορεί να εγκατασταθεί και να εκτελεστεί σε διάφορες διανομές του Linux, με αυτό τον τρόπο ο διαχειριστής μπορεί να εγκαταστήσει το MicroK8s με μία εντολή. Η δικτύωση του επιτυγχάνεται με ένα CNI plugin, το flannel. Ένα από το βασικά χαρακτηριστικά του είναι ότι επιτρέπει στον διαχειριστή να ενεργοποιεί ή να απενεργοποιεί κάποια στοιχεία αναλόγως τις ανάγκες, με αυτόν τον τρόπο μειώνεται η χρήση πόρων διότι χρησιμοποιούνται μόνο τα αναγκαία στοιχεία στον cluster. Το MicroK8s υποστηρίζει παραπάνω από ένα node με την χρήση μίας απλής εντολής. Έπειτα αναλαμβάνει την διαμοίραση των pods στα nodes, διατηρώντας το επίπεδο ελέγχου στο master node [20].

Το Vanilla K8s είναι η έκδοση του Kubernetes που διανέμει το CNCF και περιέχει τα βασικά στοιχεία και τις λειτουργίες. Είναι η πιο ενημερωμένη έκδοση του Kubernetes συγκριτικά με τις υπόλοιπες διανομές. Αποτελείται από το επίπεδο ελέγχου (control plane) το οποίο περιέχει τα API server, Scheduler, etcd και Controller manager και από τους κόμβους εργάτες οι οποίοι περιέχουν τα kubelet, Container Runtime και kube-proxy. Μπορεί να εγκατασταθεί σε διάφορα περιβάλλοντα, είτε είναι τοπικά είτε είναι μεγαλύτερης κλίμακας. Δίνει τη δυνατότητα κάποια στοιχεία του να εγκαταστηθούν εξωτερικά του cluster και να υπάρχει επικοινωνία με αυτά. Προσφέρει ευελιξία και εξατομίκευση αναλόγως τις ανάγκες του περιβάλλοντος και του διαχειριστή [64].

## 3.4 Ενορχηστρωτής container OpenShift

Το OpenShift είναι μια οικογένεια λογισμικών που αναπτύχθηκαν από τη Red Hat για το containerization. Το OpenShift Container Framework είναι το βασικό της προϊόν, μια πλατφόρμα σχεδιασμένη γύρω από τα Docker containers, τα οποία οργανώνονται και εκτελούνται από το Kubernetes με βάση το Red Hat Enterprise Linux . Το Red Hat OpenShift Container Platform παρέχει μια πλατφόρμα cloud για προγραμματιστές και οργανισμούς IT ώστε να εκτελούν εφαρμογές σε μια σταθερή, κλιμακούμενη υποδομή με ελάχιστη διαμόρφωση και διαχείριση φόρτου εργασίας. Προσφέρει διάφορες γλώσσες προγραμματισμού και προγραμματιστικά πλαίσια (framewroks, όπως Java, JavaScript, Python, Ruby και PHP.

Το OpenShift Container Platform συνδυάζει το Docker και το Kubernetes και παρέχει τη διαχείριση των container μέσω ενός API. Βασισμένο στο Red Hat Enterprise Linux και το Kubernetes, προσφέρει σύγχρονες εφαρμογές επιπέδου επιχείρησης με ένα πιο ασφαλές και κλιμακούμενο λειτουργικό σύστημα, ενώ παρέχει ενσωματωμένα περιβάλλοντα εκτέλεσης εφαρμογών και βιβλιοθήκες. Η πλατφόρμα OpenShift Container βοηθά τους οργανισμούς να ικανοποιούν τις απαιτήσεις ασφαλείας, ιδιωτικότητας, συμμόρφωσης και διακυβέρνησης.

Η βασική λειτουργικότητα ενορχήστρωσης container του OpenShift υποστηρίζεται από το Kubernetes, συμπεριλαμβανομένων χαρακτηριστικών όπως η ανθεκτικότητα σε αποτυχία στοιχείων, η παρακολούθηση, η αυτόματη επαναχρονοδρομολόγηση container και η οριζόντια κλιμάκωση.

Η ομπρέλα του OpenShift αποτελείται από τρία έργα που συνυπάρχουν. Τα τρία αυτά έργα αντιπροσωπεύουν τους χρήστες, τους προγραμματιστές και την κοινότητα με διάφορους τρόπους, συμβάλλουν στην παροχή της τεχνολογίας OpenShift στους τελικούς χρήστες, τους προγραμματιστές και τα μέλη της κοινότητας. Τα τρία αυτά έργα είναι τα εξής:

- OpenShift Origin: Το OpenShift Origin είναι η έκδοση ανοικτού κώδικα του

OpenShift που παρέχεται από την κοινότητα υπό την άδεια Apache License 2.0.

- OpenShift Online: Το OpenShift Online είναι η δημόσια ελεγχόμενη έκδοση του OpenShift. Αυτή η υπηρεσία βασίζεται στο Amazon EC2 και εκτελεί μια σταθεροποιημένη έκδοση του OpenShift Origin.
- OpenShift Enterprise: Το OpenShift Enterprise είναι μια πλήρως χρηματοδοτούμενη ιδιωτική λύση Πλατφόρμα ως Υπηρεσία (PaaS) από τη Red Hat που μπορεί να εκτελεστεί στο υλικό πόρων της κάθε επιχείρησης. Με την εφαρμογή του OpenShift Enterprise, οι επιχειρήσεις μπορούν να βελτιώσουν την ευελιξία τους στην κάλυψη των επιχειρηματικών τους αναγκών.

Επίσης οι τρόποι αλληλεπίδρασης με το OpenShift είναι οι εξής:

- Γραμμή εντολώνRHC: Το εργαλείο γραμμής εντολών RHC εγκαθίσταται ως διεπαφή γραμμής εντολών σε έναν τοπικό υπολογιστή για επικοινωνία με το OpenShift. Είναι ανεπτυγμένο σε γλώσσα προγραμματισμού Ruby. Με τη χρήση αυτού του εργαλείου ο χρήστης έχει πρόσβαση σε όλες τις λειτουργίες του OpenShift.
- Κονσόλα ιστού: Δεν χρειάζεται να εγκατασταθεί κανένα πρόγραμμα στον υπολογιστή. Η κονσόλα ιστού επιτρέπει την είσοδο και την έναρξη ανάπτυξης εφαρμογών.

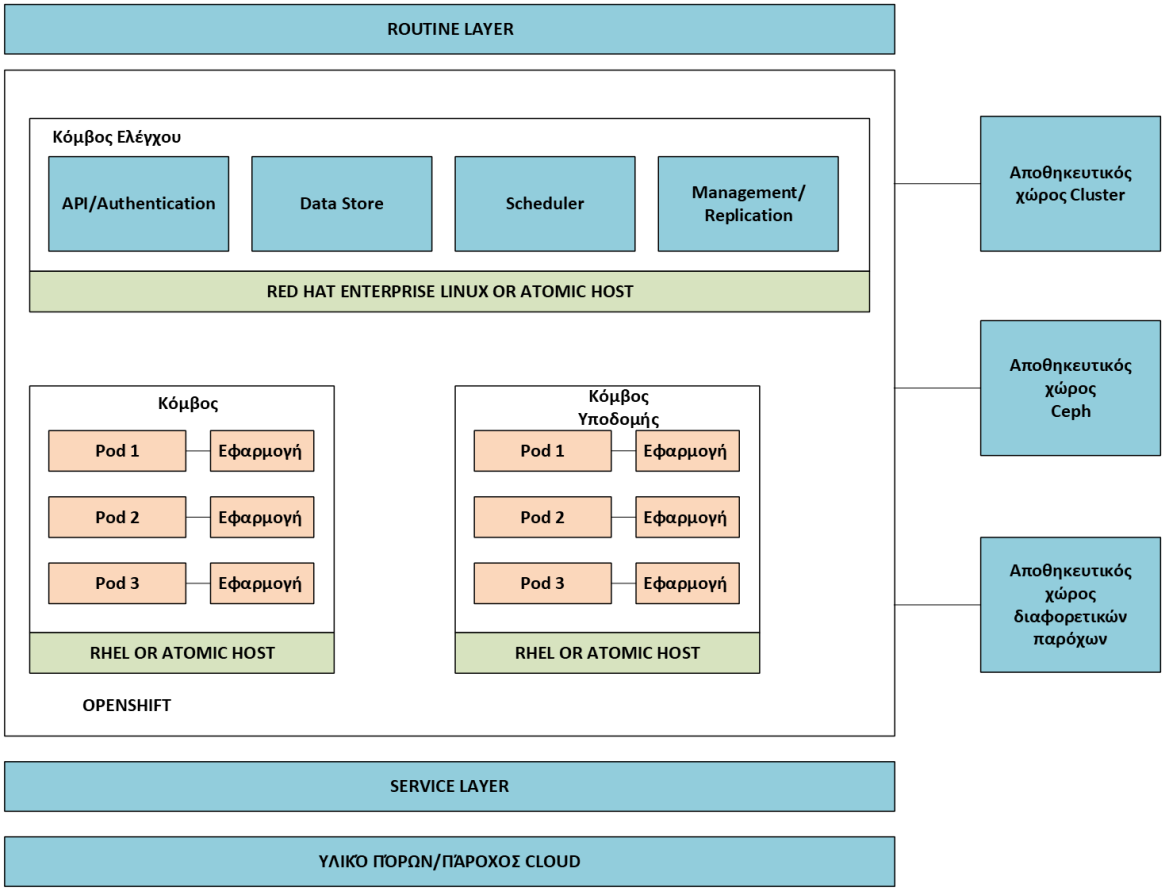
Η αρχιτεκτονική της πλατφόρμας διαχείρισης contianer, OpenShift, σχήμα 3.8, περιλαμβάνει μικρότερες, απομονωμένες μονάδες, βασισμένες σε μικροϋπηρεσίες που λειτουργούν μαζί. Αυτές οι υπηρεσίες χωρίζονται ανάλογα με τη λειτουργία τους:

- Τα REST APIs που παρέχουν κάθε αντικείμενο στον πυρήνα.
- Τα Controllers που διαβάζουν αυτά τα APIs και εφαρμόζουν τροποποιήσεις στα αντικείμενα.

Οι χρήστες καλούν το REST API για να ενημερώσουν την κατάσταση του συστήματος. Οι controllers χρησιμοποιούν το REST API για να διαβάσουν και να συγχρονίσουν τα υπόλοιπα μέρη του συστήματος με την επιθυμητή κατάσταση των χρηστών. Για παράδειγμα, εάν ένας χρήστης ζητήσει μια κατασκευή (build), δημιουργείται ένα αντικείμενο κατασκευής. Ο build controller παρατηρεί τη δημιουργία αυτή και εκτελεί μια διεργασία στο σύμπλεγμα κατασκευής. Όταν ολοκληρωθεί η κατασκευή, ο controller χρησιμοποιεί το REST API για να ενημερώσει το αντικείμενο κατασκευής, και ο χρήστης ενημερώνεται ότι η κατασκευή έχει ολοκληρωθεί.

Ο controller διασφαλίζει ότι μεγάλο μέρος της λειτουργικότητας του OpenShift Contiainer Framework είναι επεκτάσιμο. Οι controllers μετατρέπουν τις ενέργειες του χρήστη σε πραγματικότητα. Διάφορες συμπεριφορές μπορούν να επιβληθούν προσαρμόζοντας αυτούς τους controllers.

Από την πλευρά της διαχείρισης συστήματος, αυτό σημαίνει επίσης ότι το API μπορεί να χρησιμοποιηθεί για να αναπτυχθεί ένα αρχείο εντολών για προγραμματισμένες επαναλαμβανόμενες ενέργειες. Αυτά τα αρχεία είναι επίσης controllers που παρακολουθούν και λαμβάνουν ενέργειες όταν υπάρχουν αλλαγές.



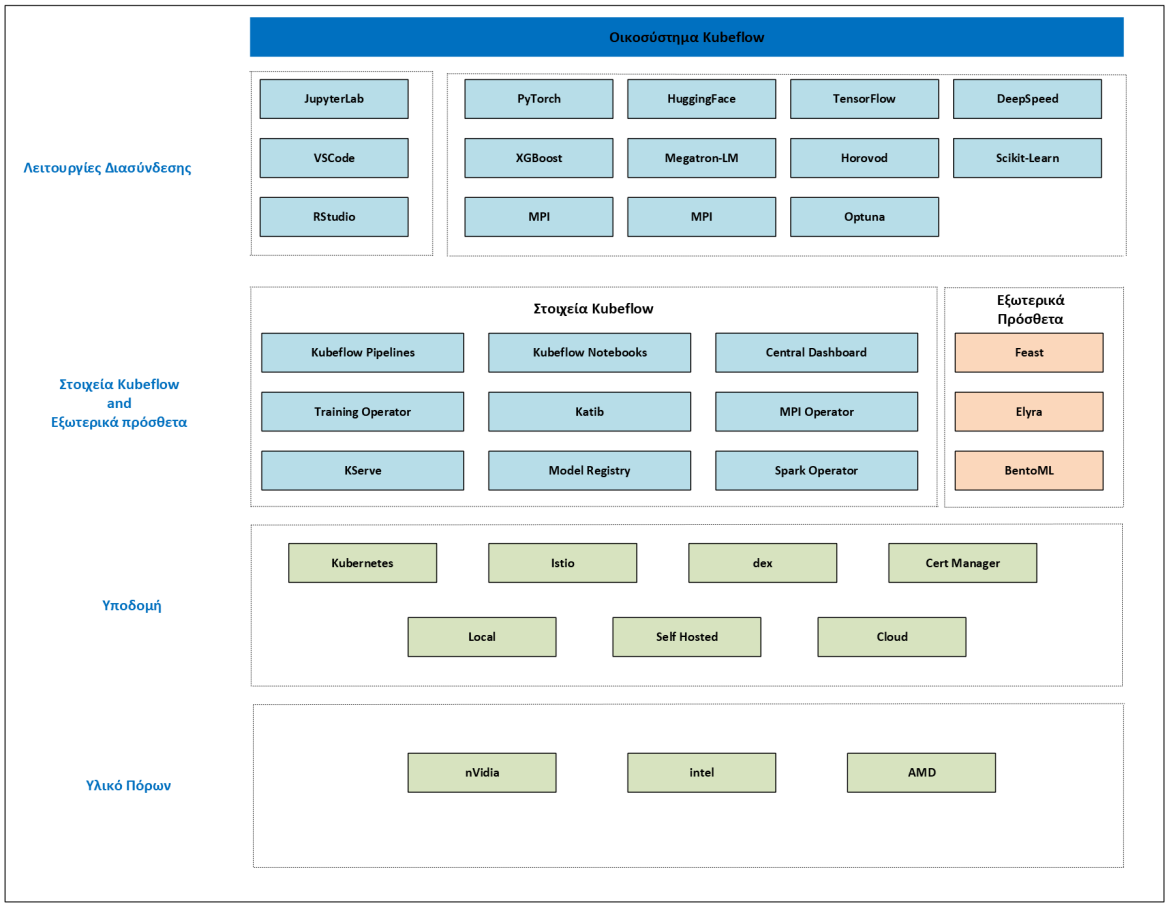

Σχήμα 3.8: Αρχιτεκτονική Openshift

# Κεφάλαιο 4

# Σύγκριση Ενορχηστρωτών συμβατών με το πρότυπο Kubernetes

## 4.1 Νεφο-κεντρική Εφαρμογή

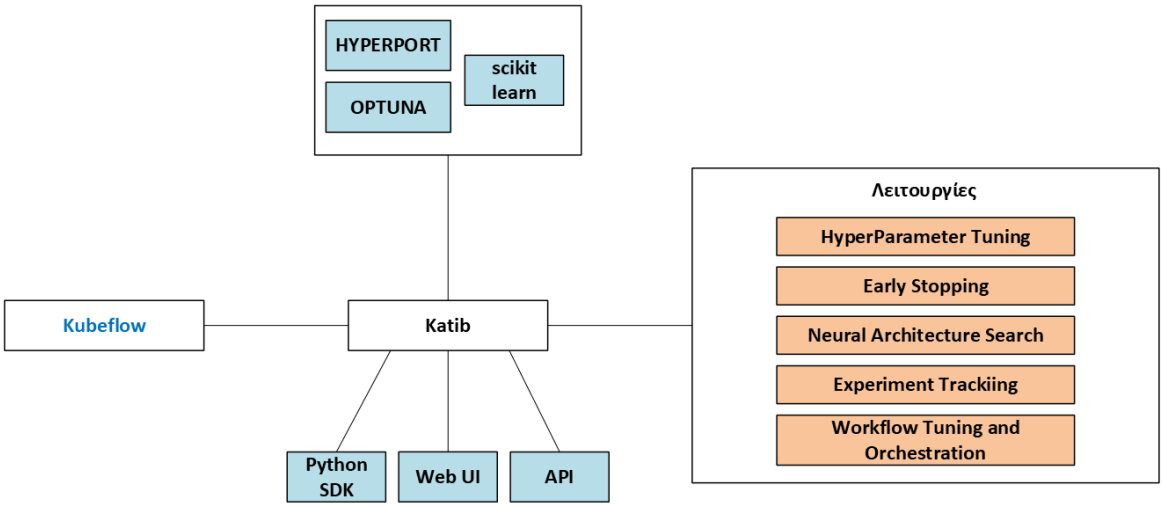
Για τα πειράματα χρησιμοποιήθηκε η containerized εφαρμογή Kubeflow. Είναι ένα ανοιχτού κώδικα project για το κάθε στάδιο του κύκλου ζωής μηχανικής μάθησης (Machine Learning - ML) που υποστηρίζει εργαλεία και προγραμματιστικά πλαίσια (frameworks). Επιτρέπει την δημιουργία μοντέλων μηχανικής μάθησης και την εκτελεσή τους για χρήση σε εφαρμογές τεχνητής νοημοσύνης (Artificial Intelligence - AI) Το Kubeflow κάνει το AI/ML στο Kubernetes απλό, φορητό και επεκτάσιμο. Αποτελείται από αρκετά στοιχεία όπως φαίνεται στο σχήμα 4.1.




Σχήμα 4.1: Οικοσύστημα Kubeflow

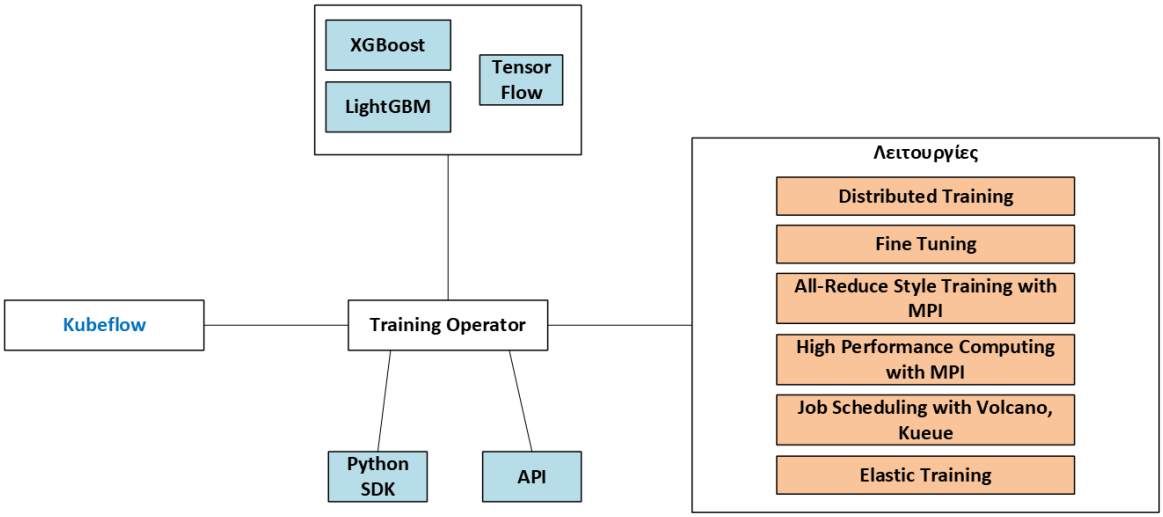
Τα στοιχεία του Kubeflow είναι:

- Το Kubeflow Spark Operator απλοποιεί και εκτελεί Spark εφαρμογές τόσο εύκολα όσο η εκτέλεση άλλων εφαρμογών στο Kubernetes.
- Τα Kubeflow Notebooks παρέχουν έναν τρόπο εκτέλεσης περιβάλλοντων ανάπτυξης που βασίζονται στον ιστό, εκτελώντας τα ως pods.
- Το Kubeflow Katib όπως φαίνεται στο σχήμα 4.2 χρησιμοποείται για την αυτοματοποίηση της μηχανικής μάθησης.




Σχήμα 4.2: Αρχιτεκτονική του Kubeflow Katib

- Το Kubeflow Training Operator, όπως φαίνεται στο σχήμα 4.3, χρησιμοποιείται για λεπτομερή ρύθμιση και επεκτάσιμη κατανεμημένη εκπαίδευση μοντέλων μηχανικής μάθησης.

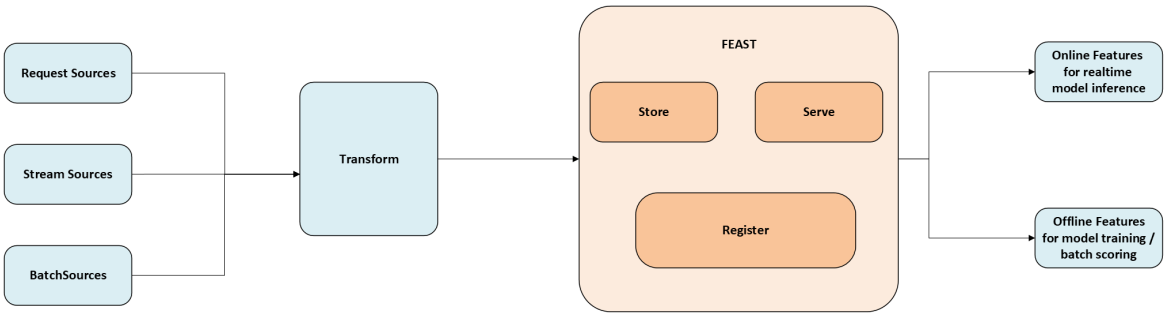



Σχήμα 4.3: Αρχιτεκτονική του Kubeflow Training Operator

- Το Model Registry παρέχει ένα κεντρικό ευρετήριο στους μηχανικούς για την διαχείριση των μοντέλων, των εκδόσεων και διαφόρων μεταδεδομένων.
- Το KServe επιτρέπει την εξαγωγή συμπερασμάτων χωρίς διακομιστή Serverless στο kubernetes. Το KServe είναι υπεύθυνο για την κλιμάκωση των μοντέλων ανάλογα με το workload. Αυτό σημαίνει, ότι δεν χρειάζεται να ανατεθεί από σε συγκεκριμένους διακομιστές ή συγκεκριμένοι υπολογιστική πόροι, αφού η διαδικασία είναι πλήρως αυτοματοποιημένη. Από την άλλη η Serverless αρχιτεκτονική δεν σημαίνει ότι δεν υπάρχουν διακομιστές, αλλά ότι ο χρήστης δεν ασχολείται με την διαχειρησή τους αφού το kubernetes σε συνδιασμό με το KServe αναλαμβάνουν την διαδικασία κλιμάκωσης και διαχείρισης των πόρων ανάλογα με την

ανάγκη των μοντέλων. Είτε να ενεργοποιηθούν οι πόροι αν αυτό είναι αναγκαίο, είτε να απενεργοποιηθούν όταν η ζήτηση μειώνεται.

- Το Feast, όπως φαίνεται στο σχήμα 4.4 είναι μία ομάδα από λειτουργίες που επιτρέπουν σε ομάδες προγραμματιστών να ορίζουν, να διαχειρίζονται, να επικειρώνουν και να προβάλλουν χαρακτηριστικά σε μοντέλα μηχανικής μάθησης.




Σχήμα 4.4: Αρχιτεκτονική του Feast

- Τα Kubeflow Pipelines χρησιμοποιούνται για τη δημιουργία, την εκτέλεση και τη διαχείριση κάθε σταδίου του κύκλου ζωής της μηχανικής μάθησης.

Ο κύκλος ζωής μηχανικής μάθησης αποτελείται από πολλά στάδια. Πρέπει να υπάρχει αξιολόγηση του αποτελέσματος του κάθε σταδίου και αν είναι αναγκαίο να γίνονται αλλαγές στις παραμέτρους ώστε να παράγεται το σωστό αποτέλεσμα. Τα στάδια είναι πέντε, το Data Preperation, το Model Developmnet, το Model Optimization, το Model training και το Model Serving.

Στο στάδιο του Data Preperation, γίνεται μια αρχική προεπεξεργασία των δεδομένων κατάλληλη ούτως ώστε να είναι συμβατά με τους αλγόριθμους μηχανικής μάθησης που πρόκειται να χρησιμοποιηθούν. Στο στάδιο του Model Development, διαλέγει ένα ML framework και αναπτύσσει την αρχιτεκτονική του μοντέλου. Στο στάδιο του Model Optimization, ελέγχονται οι διάφορες παράμετροι του μοντέλου μηχανικής μάθησης για την εύρεση, μέσω διαφόρων τεχνικών, των πιο αποδοτικών. Στο στάδιο του Model Training, εκπαιδεύεται το μοντέλο σε ένα υπολογιστικό περιβάλλον μεγάλης κλίμακας. Τέλος, στο στάδιο του Model Serving, είναι ένα υπολογιστικό περιβάλλον μεγάλης κλίμακας για την διάθεση μοντέλων μηχανικής μάθησης, προσφέροντας αυτοματοποιημένη υλοποίηση, παρακολούθηση, και διαχείριση μοντέλων μέσω εργαλείων όπως το KServe. Έτσι δύναται η εύκολη προσαρμογή των μοντέλων σε εφαρμογές και υπηρεσίες.

Ένα pipeline είναι μία περιγραφή της ροής εργασίας (workflow) μηχανικής μάθησης και συμπεριλαμβάνει όλα τα απαραίτητα στοιχεία της ροής εργασίας και πως αυτά συνδέονται μεταξύ τους σε μορφή διαγράμματος. Η διαμόρφωση του pipeline περιέχει τις απαραίτητες παραμέτρους για την εκτέλεση του pipeline και τις εισόδους και εξόδους του κάθε στοιχείου. Με την εκτέλεση του pipeline, ξεκινάνε ένα ή περισσότερα ποδς για το κάθε στάδιο.

## 4.2 Παραμετροποίηση Εργαλείων και Πειραματική τοπολογία

Η υποδομή που βασίστηκαν οι τοπολογίες για τις τρεις διανομές των ενορχηστρωτών πόρων συμβατών με το Kubernetes:

- Περιέχει 64 επεξεργαστές Intel(R) Xeon(R) 2.10GHZ εξασφαλίζοντας ότι οι κόμβοι του Cluster θα έχουν διαθέσιμους πόρους για να εκτελέσουν τα pods και να διαχειριστούν μεγάλο φόρτο εργασίας.
- Η μνήμη του Cluster είναι 251.50GB και είναι επαρκής για να υποστηρίξει μεγάλες εφαρμογές ή μεγάλο πλήθος από pods χωρίς να υπάρξουν προβλήματα στέρησης μνήμης.
- Περιέχει 24 TB αποθηκευτικού χώρου, αρκετός για αποθήκευση μεγάλου φόρτου δεδομένων, χωρίς τον φόβο εξαντλησής του.

Λόγω των παραπάνω διασφαλίζεται ότι δεν θα δημιουργηθεί συμφόρηση κατά την εκτέλεση και τη διαχείριση μεγάλου φόρτου εργασίας.

Στον πίνακα 4.1 παρουσιάζονται τα χαρακτηριστικά των τριών διανομών που χρησιμοποιήθηκαν στη μελέτη. Για την κάθε διανομή αναγράφεται ο αριθμός των κόμβων του Cluster, το λειτουργικό σύστημα του host, τους πυρήνες του επεξεργαστή του host, την μνήμη του host, τον αποθηκευτικό χώρο του host και την έκδοση του Kubernetes.

Πίνακας 4.1: Πίνακας Τοπολογιών

| Διανομή Kubernetes | Κόμβος | Λειτουργικό Σύστημα | CPU | Μνήμη RAM (GB) | Αποθηκευτικός Χώρος (GB) | Έκδοση Kubernetes |
|---|---|---|---|---|---|---|
| VANILLA | node1 | Ubuntu 22.04 | 4 | 8 | 150 | 1.30 |
| | node2 | Ubuntu 22.04 | 4 | 8 | 150 | 1.30 |
| | node3 | Ubuntu 22.04 | 4 | 8 | 150 | 1.30 |
| K3s | node1 | Ubuntu 22.04 | 4 | 8 | 150 | 1.30 |
| | node2 | Ubuntu 22.04 | 4 | 8 | 150 | 1.30 |
| | node3 | Ubuntu 22.04 | 4 | 8 | 150 | 1.30 |
| MicroK8s | node1 | Ubuntu 22.04 | 4 | 8 | 150 | 1.31 |
| | node2 | Ubuntu 22.04 | 4 | 8 | 150 | 1.31 |
| | node3 | Ubuntu 22.04 | 4 | 8 | 150 | 1.31 |

# Κεφάλαιο 5

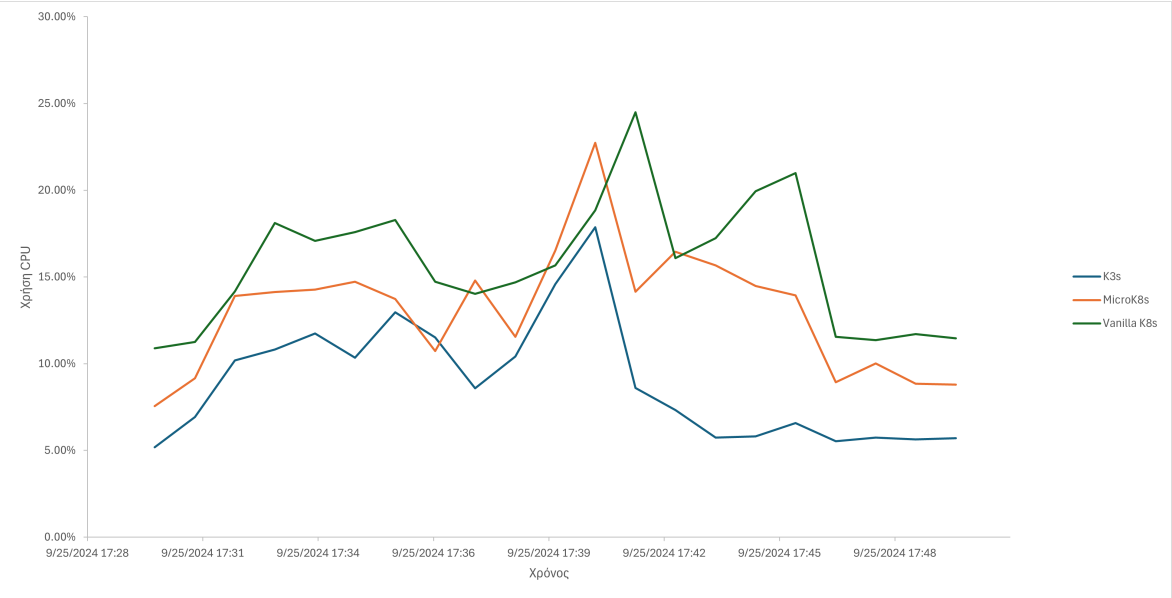
# Αποτελέσματα

Σε αυτό το κεφάλαιο παρουσιάζονται τα γραφήματα των τριών διανομών συμβατών με το Kubernetes εκτελώντας ένα pipeline στο Kubeflow. Οι μετρικές που συλλέχθηκαν αφορούν τη χρήση του επεξεργαστή, της μνήμης και του δικτύου της κάθε διανομής, των κόμβων και του namespace του Kubeflow της κάθε διανομής. Αρχικά, παρουσιάζονται συγκεντρωτικά τα γραφήματα επεξεργαστή και μνήμης του κάθε cluster. Στη συνέχεια, παρουσιάζονται τα γραφήματα επεξεργαστή και μνήμης των τριών κόμβων του κάθε cluster. Τέλος, παρουσιάζονται τα γραφήματα επεξεργαστή, μνήμης και δικτυακής κίνησης του namespace του Kubeflow.

Με την παραπάνω προσέγγιση, γίνεται πιο σαφής ο τρόπος χρήσης με τον οποίο η κάθε διανομή χειρίζεται την εκτέλεση μίας πολύπλοκης εφαρμογής και συγκεκριμένα, πολύπλοκες εργασίες μηχανικής μάθησης, αποκαλύπτοντας τα πλεονεκτήματα και τα μειονεκτήματα τους.

## 5.1 Αποτελέσματα Χρήσης Επεξεραγαστή και των τρίων διανομών

Με το πέρας των πειραμάτων αναδείχθηκαν τα παρακάτω αποτελέσμετα για την κάθε διανομή. Πρώτα παρουσιάζονται τα αποτελέσματα για την χρήση του επεξεργαστή σε ένα γενικό σχήμα.



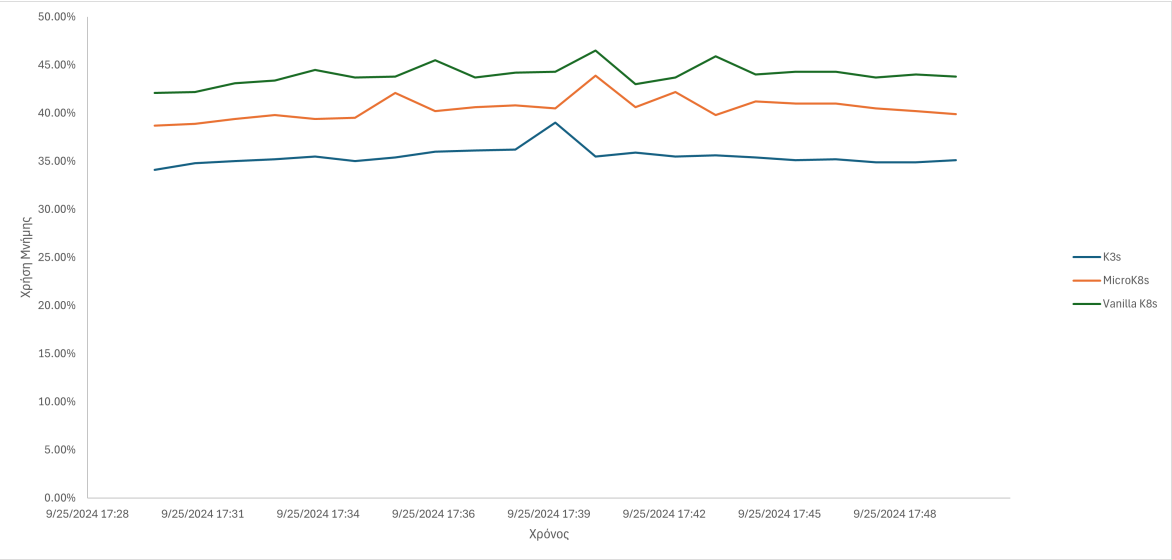

Σχήμα 5.1: Χρήση CPU του cluster.

Το σχήμα 5.1 παρουσιάζει την χρήση των πόρων του επεξεργαστή για τις τρεις διανομές, K3s (μπλε), MicroK8s (πορτοκαλί) και το Vanilla K8s (πράσινο) καθώς εκτελείται ένα Kubeflow pipeline. Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος άξονας την ποσοστιαία χρήση επεξεργαστή.

Αρχικά, μέχρι τη στιγμή 17:30 όλες οι διανομές παρουσιάζουν σχετικά χαμηλή χρήση επεξεργαστή με ποσοστό ανάμεσα στο 5-10%. Αυτό υποδεικνύει μία ομαλή εκκίνηση. Στα επόμενα λεπτά και μέχρι την χρονική στιγμή 17:35, η χρήση επεξεργαστή και στις τρεις διανομές αυξάνεται λόγω του φόρτου εργασίας που παράγεται από το pipeline. Το Vanilla K8s παρουσιάζει την μεγαλύτερη αύξηση με ποσοστό κοντά στο 20%, ακολουθεί το MicroK8s με 15% και το K3s με 13%. Το Vanilla K8s την χρονική περίοδο 17:39 μέχρι 17:47 έχει την μεγαλύτερη αύξηση στην κατανάλωση επεξεργαστή φτάνοντας το 25%. Σταθεροποιείται κοντά στο 12-13%. Το MicroK8s ακολουθεί παρόμοιο μοτίβο φτάνοντας το 23-24% και σταθεροποιείται στο 10%. Το K3s τη στιγμή 17:39 έχει μία μικρή αύξηση στο 18% και στη συνέχεια μειώνεται και σταθεροποιείται στο αρχικό ποσοστό κατανάλωσης, δηλαδή στο 5%.

Από τα παραπάνω συμπερένεται ότι το Vanilla K8s καταναλώνει τους περισσότερους πόρους επεξεργαστή, ειδικότερα την περίοδο με το μεγαλύτερο φόρτο εργασίας λόγω των απαιτητικών εργασιών του pipeline. Το K3s είναι το πιο αποδοτικό στην κατανάλωση και το MicroK8s βρίσκεται ανάμεσα τους, παρουσιάζοντας μία ισορροπία στην απόδοση και στην επίδοση.

Το σχήμα 5.2 παρουσιάζει την χρήση των πόρων της μνήμης των τριών διανομών, K3s (μπλε μπάρα), MicroK8s (πορτοκαλί μπάρα) και το Vanilla (πράσινη μπάρα). Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος άξονας την ποσοστιαία χρήση επεξεργαστή.



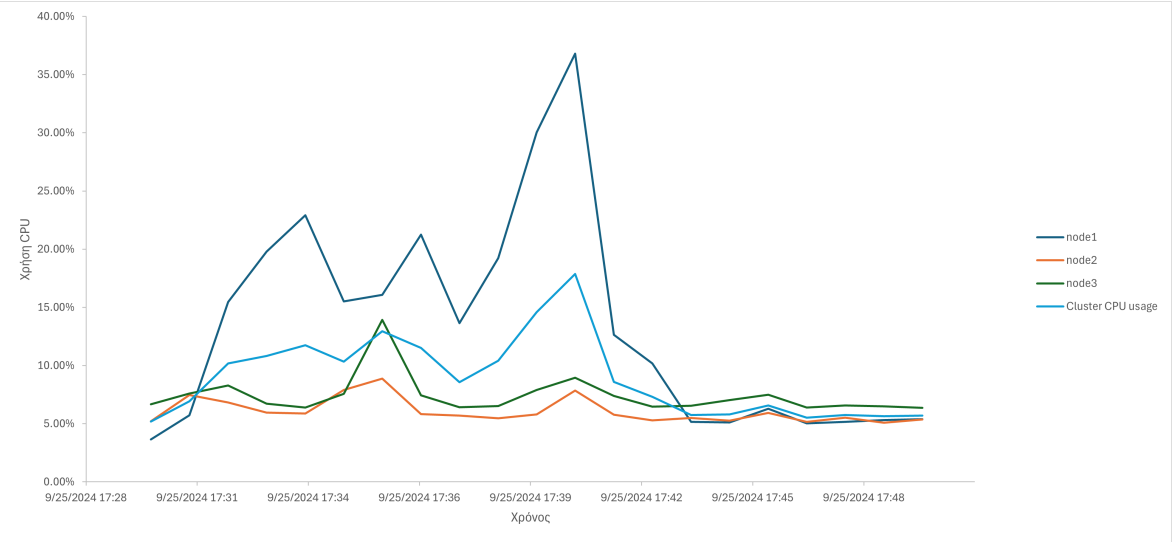

Σχήμα 5.2: Χρήση Μνήμης του cluster.

Αρχικά, και οι τρεις διανομές παρουσιάζουν σχεδόν σταθερή χρήση της μνήμης. Το Vanilla K8s (πράσινο) βρίσκεται ανάμεσα στο 40-45% με τρείς στιγμιαίες αυξήσεις στο 45%. Το K3s (μπλε) βρίσκεται κοντά στο 35% με μία στιγμιαία αύξηση να πλησιάζει το 40%. Το MicroK8s (πορτοκαλί) βρίσκεται κοντά στο 40% με τρεις αυξήσεις να πλησιάζουν το 42%.

Από τα παραπάνω φαίνεται πως το Vanilla K8s καταναλώνει τους περισσότερους πόρους μνήμης και έχει τις μεγαλύτερες αυξήσεις. Το K3s έχει την πιο σταθερή χρήση των πόρων με μία μόνο αύξηση λόγω του φόρτου εργασίας που παράγεται από το pipeline. Το MicroK8s βρίκσεται ανάμεσα τους και παρουσιάζει τρεις ήπιες αυξήσεις.

## 5.2 Αποτελέσματα CPU, MEM, NETWORK για την κάθε διανομή ξεχωριστά

Αρχικά θα αναλυθούν τα σχήματα για το K3s.
Το σχήμα 5.3 παρουσιάζει την χρήση του επεξεργαστή των node1, node2, node3 και του cluster. Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος το ποσοστό χρήσης του επεξεργαστή.



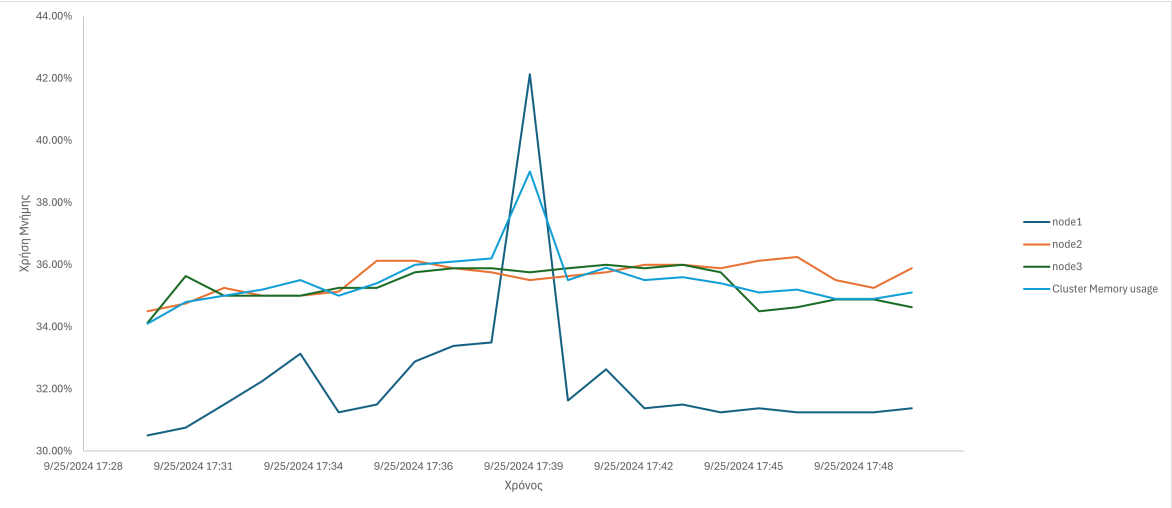

Σχήμα 5.3: Χρήση CPU των κόμβων της διανομής K3s

.

Αρχικά, μέχρι την στιγμή 17:31 η χρήση επεξεργαστή στους τρεις nodes και του cluster είναι σχετικά χαμηλή και βρίσκεται ανάμεσα στο 5-10%. Την χρονική περίοδο 17:31-17:41, το node1 (μπλε) παρουσιάζει μια σταδιακή αύξηση και βρίσκεται ανάμεσα στο 15-25% και παρουσιάζει την μεγαγλύτερη στιγμιαία αύξηση στην κατανάλωση πόρων του επεξεραστή σχετικά με τους υπόλοιπους δύο, φτάνοντας το 40% την στιγμή 17:41. Αυτό δείχνει πως έχει αναλάβει το περισσότερο φόρτο εργασίας. Η χρήση επεξεργαστή του cluster (γαλάζιο) φαίνεται να ακολουθεί αυτή του node1, υποδεικνύοντας ότι το node1 είναι το βασικό στοιχείο κατανάλωσης των πόρων επεξεργαστή σε αυτή την χρονική περίοδο. Το node2 (πορτοκαλί) βρίσκεται ανάμεσα στο 5-10% χωρίς κάποιες ιδιαίτερες αυξήσεις στη χρήση πόρων. Αυτό δείχνει πως έχει αναλάβει κάποιο ελαφρύ φόρτο εργασίας. Το node3 (πράσινο) έχει επίσης σταθερή χρήση και βρίσκεται ανάμεσα στο 5-10% παρουσιάζοντας μία στιγμιαία κοντά στο 15%. Αυτό δείχνει πως δεν επεξεργάζεται μεγάλο φόρτο εργασίας.

Από τα παραπάνω φαίνεται ότι το node1 χειρίζεται το μεγαλύτερο φόρτο εργασίας κατά την διάρκεια εκτέλεσης του pipeline ενώ τα node2, node3 παρουσιάζουν μία πιο σταθερή χρήση του επεξεργαστή υποδεικνύοντας ότι έχουν αναλάβει μικρότερο φόρτο εργασίας ή δεν αξιοποιούνται σωστά ώστε να υπάρχει καλύτερη εξισορρόπησει φόρτου εργασίας και πόρων.

Το σχήμα 5.4 παρουσιάζει την χρήση μνήμης των node1, node2, node3 και του cluster. Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος άξονας την ποσοστιαία χρήση επεξεργαστή.

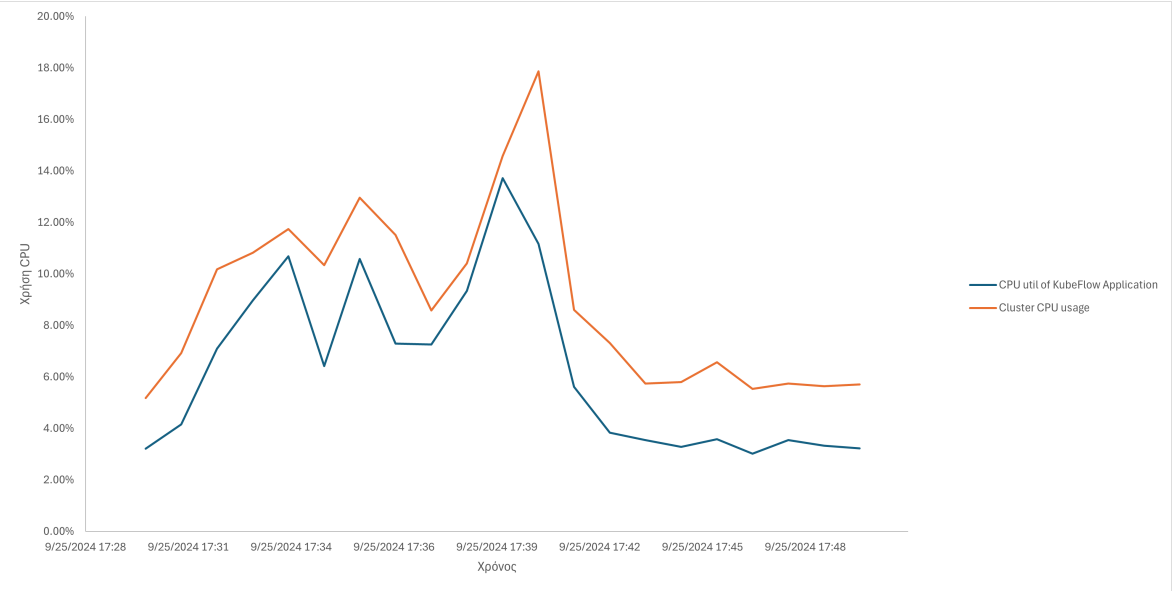



Σχήμα 5.4: Χρήση Μνήμης των κόμβων της διανομής K3s

Αρχικά, μέχρι την χρονική στιγμή 17:31 η χρήση μνήμης στους τρεις nodes και του cluster είναι σχετικά σταθερή. Το node1 (μπλε) κυμαίνεται στο 32-34%, το node2 (πορτοκαλί) στο 36%, το node3 (πράσινο) στο 34-36% και το cluster (γαλάζιο) κοντά στο 36%. Το node1 από τη στιγμή 17:34 μέχρι και την 17:41 έχει τη χαμηλότερη χρήση πόρων και βρίσκεται ανάμεσα στο 32-34%, παρουσιάζοντας μία στιγμιαία αύξηση, τη στιγμή 17:39 με ποσοστό 41%. Αυτή η αύξηση υποδεικνύει ότι εκείνη την χρονική στιγμή εκτελέστηκαν εργασίες που είχαν σχετικά υψηλές απαιτήσεις πόρων μνήμης. Το node2 έχει περισσότερη χρήση σχετικά με το node1 και βρίσκεται κοντά στο 36%, αλλά δεν παρουσιάζει κάποια σημαντική στιγμιαία αύξηση. Το node3 βρίσκεται κοντά στο 36% χωρίς καμία αύξηση στην χρήση μνήμης, αυτό δείχνει πως δεν χρησιμοποιείται ιδιαίτερα από την εφαρμογή. Το cluster βρίσκεται και αυτό κοντά στο 36% με μία στιγμιαία αύξηση παρόμοια με το node1 κοντά στο 40%. Αυτό δείχνει πως το node1 επηρέαση την συνολική μνήμε του cluster. Από τη στιγμή 17:41 μέχρι την λήξη του πειράματος, το node1 έχει ποσοστό χρήσης κοντά στο 32%, το node2,node3 κοντά στο 36% και το cluster επίσης κοντά στο 36%.

Από τα παραπάνω φαίνεται πως το node1 παρουσίασε μία στιγμιαία αύξηση λόγω εργασιών που απαιτούσαν παραπάνω μνήμη και αυτό επηρέασε και την χρήση μνήμης του cluster. Το node2 είχε παρόμοια χρήση πόρων με το node3 που ήταν υψηλότερες του node1, που υποδεικνύει πως εκτελούσαν περισσότερες εργασίες λόγω του pipeline και περισσότερες εσωτερικές διεργασίες.

Το σχήμα5.5 παρουσιάζει την χρήση επεξεργαστή του namespace της εφαρμογής Kubeflow και του cluster. Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος άξονας την ποσοστιαία χρήση επεξεργαστή.

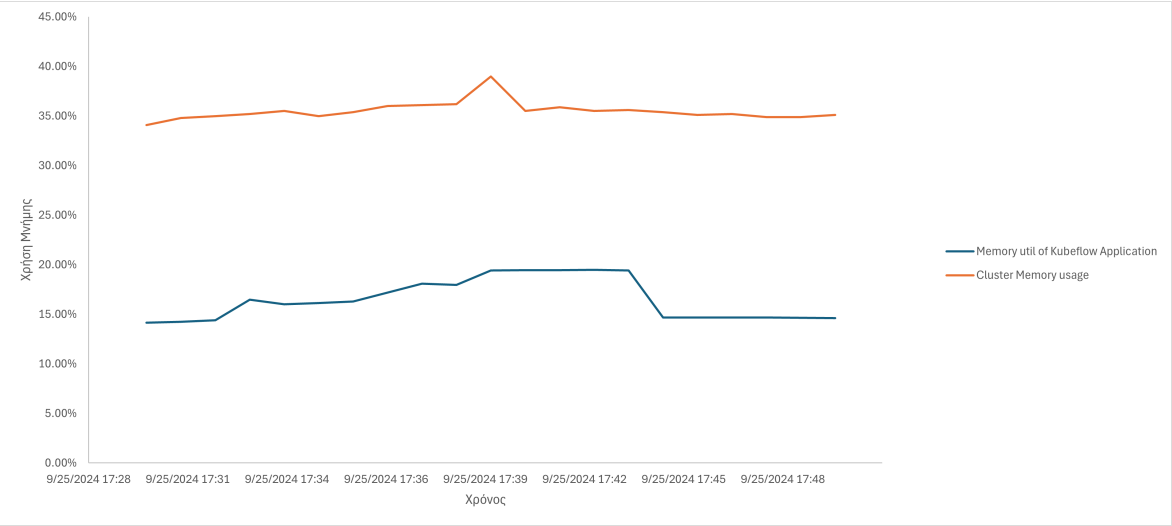



Σχήμα 5.5: Χρήση CPU του Kubeflow namespace της διανομής K3s

.

Αρχικά, μέχρι την στιγμή 17:30 η χρήση επεξεργαστή τόσο του Kubeflow (μπλε) όσο και του cluster (πορτοκαλί) παραμένει χαμηλή. Από τη στιγμή 17:31 μέχρι την 17:42 η χρήση επεξεργαστή του kubeflow φαίνεται να επηρεάζει σημαντικά το cluster με ποσοστό ανάμεσα στο 6-12% με τρεις στιγμιαίες αυξήσεις, οι πρώτες δύο κοντά στο 10% και η τρίτη κοντά στο 14%. Η χρήση πόρων του cluster αυτή την περίοδο βρίσκεται ανάμεσα στο 10-18% που υποδεικνύει ότι εκτελεί σημαντικές εργασίες του pipeline. Από την στιγμή 17:41 και μετά η χρήση πόρων επεξεργαστή μειώνεται και σταθεροποιείται για το Kubeflow στο 4% και για το cluster στο 6%. Από τα παραπάνω φαίνεται ότι, η χρήση πόρων επεξεργαστή του Kubeflow συμβάλλει σημαντικά στην ολική χρήση επεξεργαστή του cluster.

Το σχήμα 5.6 παρουσιάζει την χρήση μνήμης του namespace της εφαρμογής Kubeflow και του cluster. Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος άξονας την ποσοστιαία χρήση επεξεργαστή.

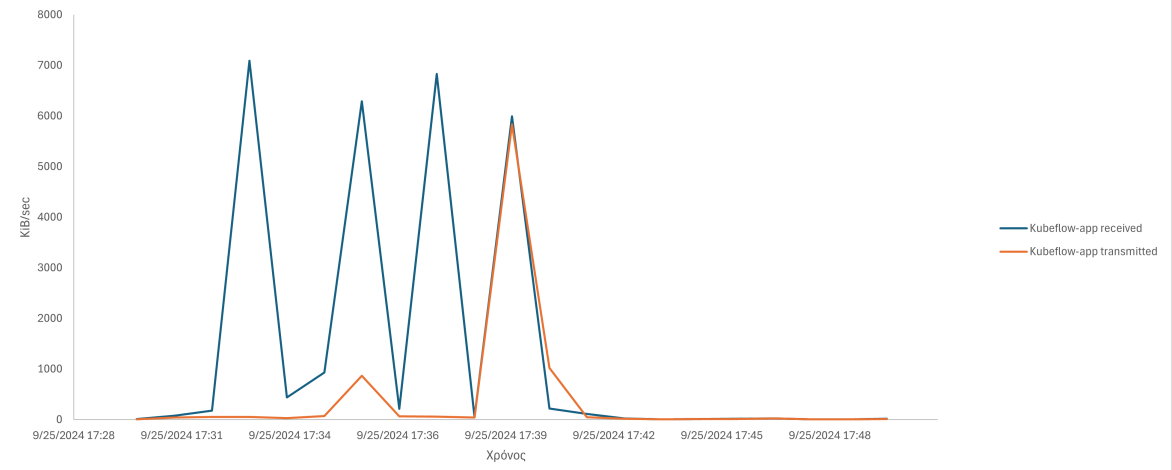



Σχήμα 5.6: Χρήση Μνήμης του Kubeflow namespace της διανομής K3s

.

Η χρήση μνήμης του Kubeflow (μπλε) παραμένει σχετικά σταθερή και κοντά στο 15-20% και μειώνεται στο 15% μετά την χρονική στιγμή 17:45. Η χρήση μνήμης του cluster (πορτοκαλί) παραμένει επίσης σταθερή στο 36% με μία στιγμιαία αύξηση στο 40% λόγω του φόρτου εργασίας του pipeline. Φαίνεται πως το Kubeflow δεν επηρεάζει σημαντικά την κατανάλωση πόρων μνήμης του cluster.

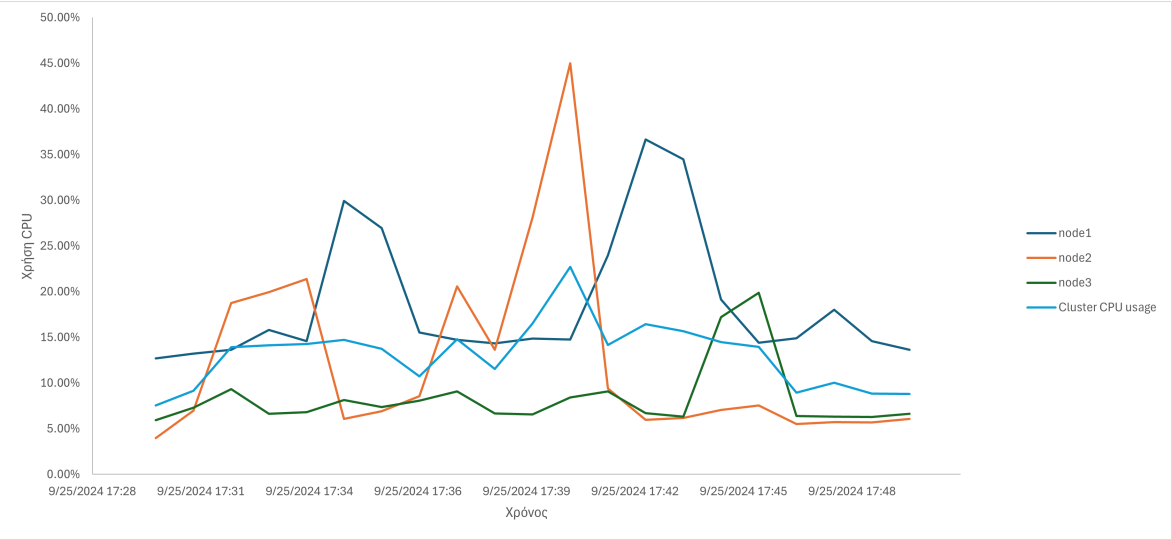
Τέλος, το σχήμα 5.7 παρουσιάζει τη δικτυακή κίνηση που δέχεται και μεταδίδει το Kubeflow namespace. Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος την δικτυακή κίνηση σε KiB/sec.




Σχήμα 5.7: Διακτυακή κίνηση του Kubeflow namespace της διανομής K3s

Μέχρι τη στιγμή 17:31 η δικτυακή κίνηση βρίσκεται στο μηδέν. Από τη στιγμή 17:31 μέχρι τη στιγμή 17:40 υπάρχουν τέσσερις στιγμιαίες αυξήσεις στα δεδομένα που λαμβάνονται από το Kubeflow, αυτό σημαίνει πως η εφαρμογή δέχεται αρκετά δεδομένα από κάποια εξωτερική πηγή. Τη χρονική 17:39 φαίνεται να μεταδίδει δεδομένα.

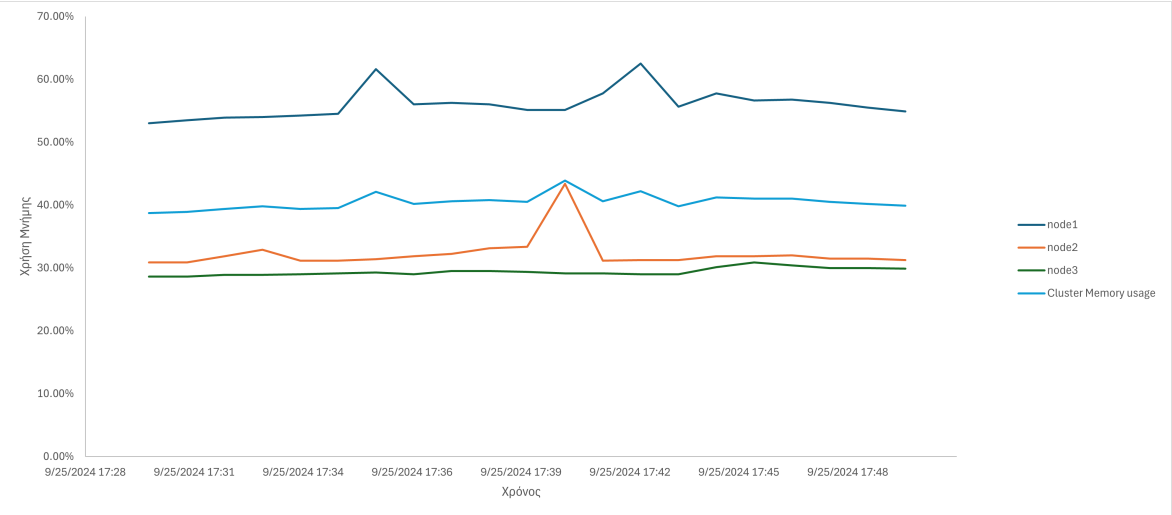
Στη συνέχεια παρουσιάζονται τα γραφήματα για το MicroK8s. Το σχήμα 5.8 παρουσιάζει την χρήση επεξεργαστή του κάθε node και του cluster. Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος το ποσοστό χρήσης του επεξεργαστή.




Σχήμα 5.8: Χρήση CPU των κόμβων της διανομής MicroK8s

Μέχρι τη στιγμή 17:30 η χρήση των τριών κόμβων και του cluster βρίσκεται ανάμεσα στο 5-15%. Τη χρονική στιγμή 17:39 το node2 (πορτοκαλί) παρουσιάζει μία στιγμαία αύξηση στο 45% ενώ οι υπόλοιπες οντότητες βρίσκονται ανάμεσα στο 5-15%, αυτό υποδεικνύει πως το node2 έλαβε μεγαλύτερο φόρτο εργασίας λόγω του pipeline. Το node1 (μπλε) από τη χρονική στιγμή 17:31 μέχρι και την 17:43 παρουσιάζει δύο στιγμιαίες αυξήσεις μεταξύ του 25-30%, αυτό υποδηλώνει πως λαμβάνει διάφορους φόρτους εργασίας αναλόγως των σταδίων του pipeline. Το node3 (πράσινο) παραμένει σταθερό στη χρήση επεξεργαστή μεταξύ 5-10% με μία μικρή αύξηση στο 15% τη χρονική στιγμή 17:45. Η χρήση επεξεργαστή του cluster (γαλάζιο) φαίνεται πως επηρεάζεται αρκετά από τα node1, node2.Όταν αυτά τα δύο nodes έχουν στιγμιαίες αυξήσεις, το cluster παρουσιάζει και αυτό. Από τη χρονική στιγμή 17:43 και μετά οι οντότητες φαίνεται πως σταθεροποιούνται στα ποσοστά πριν την εκτέλεση του πειράματος, δηλαδή στο 5-15%, εκτός της στιγμιαίας αύξησης του node3.

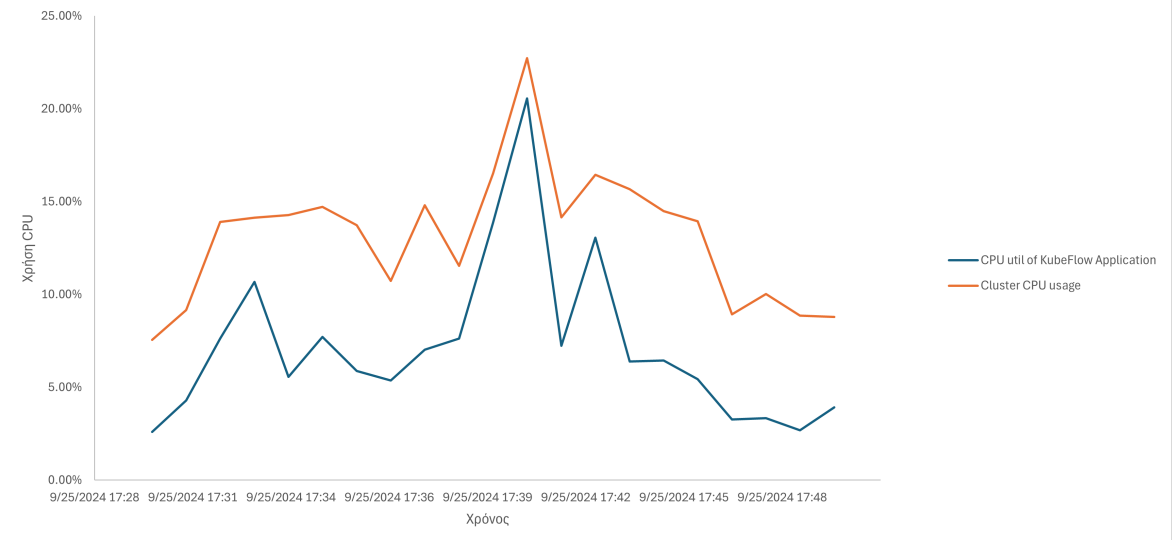
Το σχήμα 5.9παρουσιάζει την χρήση μνήμης του κάθε node και του cluster. Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος το ποσοστό χρήσης της μνήμης.




Σχήμα 5.9: Χρήση Μνήμης των κόμβων της διανομής MicroK8s

Όπως φαίνεται, το node1 (μπλε) έχει σταθερή χρήση ανάμεσα στο 50-60% , τα node2 (πορτοκαλί) και node3 (πράσινο) παραμένουν κοντά στο 30% με το node2 να έχει μία μικρή αύξηση στο 40%. Το cluster (γαλάζιο) βρίσκεται σταθερά στο 40%. Από τα παραπάνω φαίνεται ότι το node1 έχει αναλάβει το μεγαλύτερο φόρτο εργασίας που προέρχεται από την εκτέλεση του pipeline του Kubeflow, ενώ το node2 φαίνεται να μην έχει μεγάλο φόρτο εργασίας που να προέρχεται από το pipeline. Το node3 έχει σταθερή χρήση πόρων μνήμης και δεν λαμβάνει εργασίες του pipeline.

Το σχήμα 5.10 παρουσιάζει την χρήση επεξεργαστή του κάθε node και του cluster. Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος το ποσοστό χρήσης του επεξεργαστή.

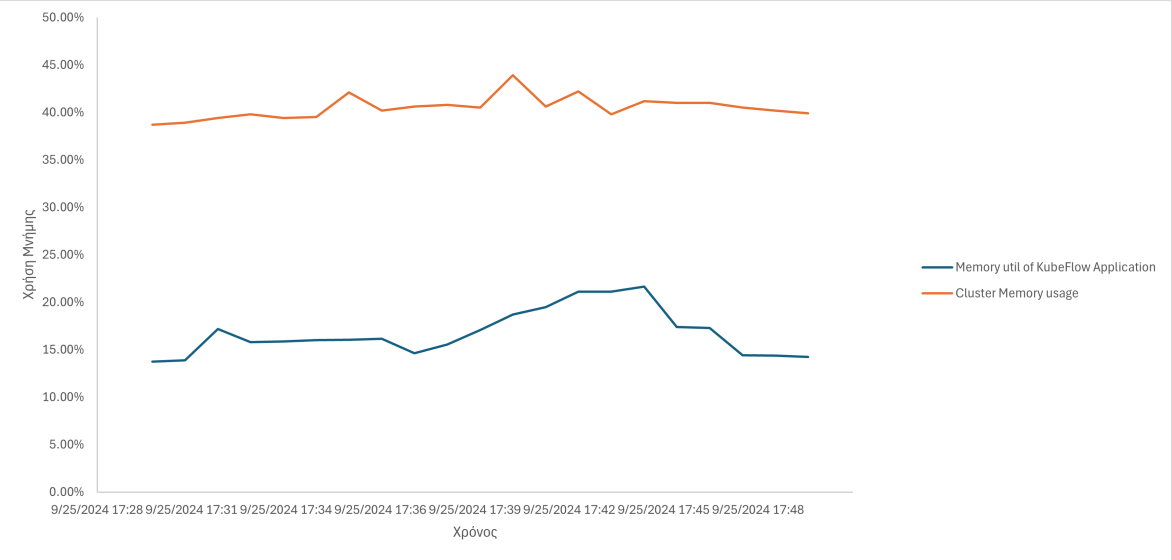



Σχήμα 5.10: Χρήση CPU του Kubeflow namespace της διανομής MicroK8s

Αρχικά, από τη στιγμή 17:30 μέχρι τη στιγμή 17:37 το Kubeflow (μπλε) καταναλώνει λίγους πόρους επεξεργαστή, βρίσκεται ανάμεσα στο 5-10% ,συγκριτικά του cluster (πορτοκαλί) που κυμαίνεται στο 10-15% με εξαίρεση μία μικρή αύξηση τη στιγμή 17:32

με ποσοστό κοντά στο 10%. Την περίοδο εκτέλεσης του Pipeline, το kubeflow καταλαμβάνει σχεδόν το μεγαλύτερο μέρος των πόρων του επεξεργαστή με ποσοστό να φτάνει το 20%, ενώ του cluster να φτάνει το 23%. Αυτή τη χρονική περίοδο, 17:39-17:40, υπάρχει μεγάλο φόρτο εργασίας λόγω του pipeline που μάλλον επεξεργάζεται δεδομένα ή εκπαιδεύει το μοντέλο. Τέλος, από τη χρονική στιγμή 17:42, αρχίζει να μειώνεται και να σταθεροποιείται η χρήση του Kubeflow κοντα στο 5%, ενώ το cluster σταθεροποείται στο 10%, αυτό υποδεικνύει πως εκτελούνται ακόμα κάποιες εργασίες.

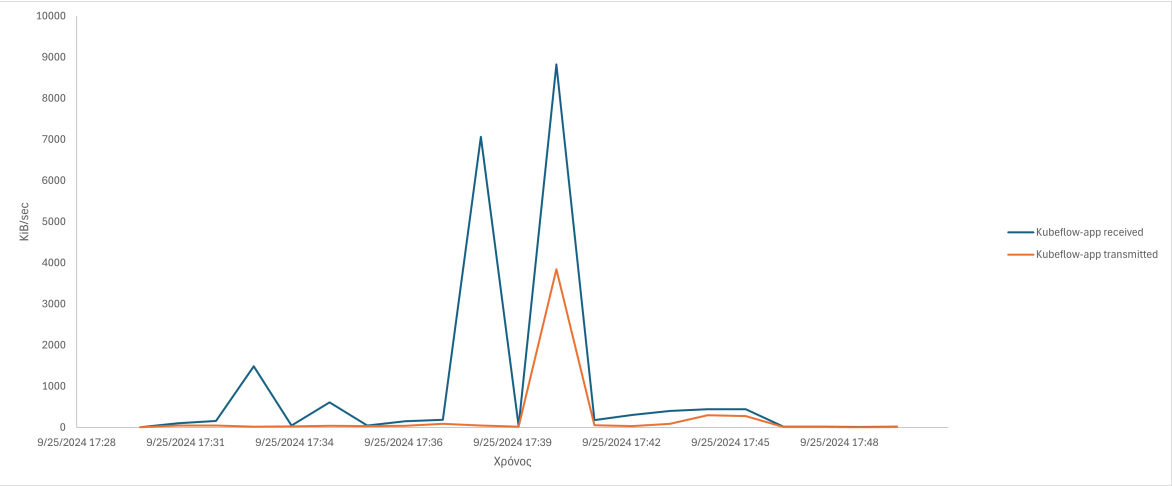
Το σχήμα 5.11 παρουσιάζει την χρήση μνήμης του namespace της εφαρμογής Kubeflow και του cluster. Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος άξονας παριστάνει το ποσοστό χρήσης μνήμης.




Σχήμα 5.11: Χρήση Μνήμης του Kubeflow namespace της διανομής MicroK8s.

Παρατειρείται ότι, πέρα από την περίοδο που εκτελείται το Pipeline όπου η χρήση μνήμης αυξάνεται ελάχιστα τόσο του cluster (πορτοκαλί) όσο και του Kubeflow (μπλε), η χρήση μνήμης του Kubeflow παραμένει σχετικά σταθερή και σε χαμηλά επίπεδα κοντά στο 15-20% και του cluster να παραμένει επίσης σχετικά σταθερή κοντά στο 40-45% πράγμα που υποδεικνύει ότι το Kubeflow δεν καταναλώνει πολλούς πόρους μνήμης και δεν έχει κάποια επίπτωση στη συνολική μνήμη που καταναλώνει το cluster.

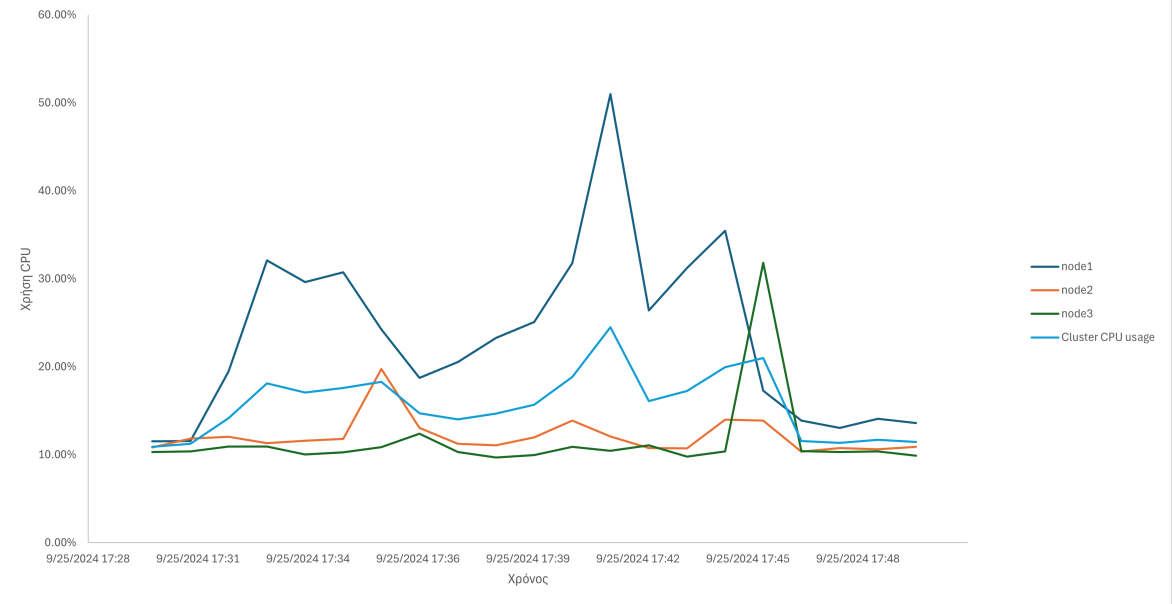
Τέλος, το σχήμα 5.12 παρουσιάζει τη δικτυακή κίνηση που δέχεται και μεταδίδει το Kubeflow namespace. Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος την δικτυακή κίνηση σε KiB/sec.




Σχήμα 5.12: Διακτυακή κίνηση του Kubeflow namespace της διανομής MicroK8s.

Η εφαρμογή φαίνεται να κατεβάζει δεδομένα και ειδικότερα έχει κάποιες αυξήσεις τις χρονικές στιγμές, 17:37, 17:40 με τιμές ανάμεσα στα 6000-9000 KiB/sec για ανάλυση ή για εκτέλεση εργασιών μηχανικής μάθησης και μεταδίδει δεδομένα τη χρονική στιγμή 17:40 σε κάποιο σεριρε του cluster.

Στη συνέχεια παρουσιάζονται τα γραφήματα για το Vanilla Kubernetes. Το σχήμα 5.13 παρουσιάζει την χρήση του επεξεργαστή των node1, node2, node3 και του cluster. Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος άξονας την ποσοστιαία χρήση επεξεργαστή.




Σχήμα 5.13: Χρήση CPU των κόμβων της διανομής Vanilla Kubernetes

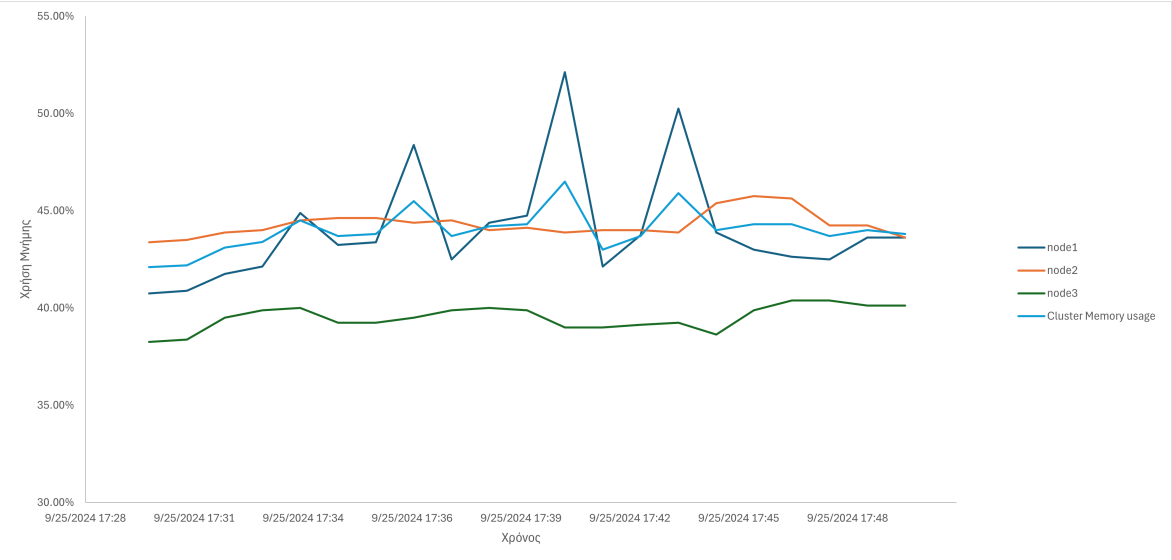
.

Αρχικά, μέχρι την στιγμή 17:31 η χρήση επεξεργαστή στους τρεις nodes και του cluster (γαλάζιο) είναι σχετικά χαμηλή και βρίσκεται ανάμεσα στο 10-15%. Την χρονική περίοδο 17:31-17:45, το node1 (μπλε) βρίσκεται ανάμεσα στο 20-35% και παρουσιάζει

την μεγαγλύτερη στιγμιαία αύξηση στην κατανάλωση πόρων του επεξεραστή σχετικά με τους υπόλοιπους δύο, φτάνοντας το 50% την στιγμή 17:42. Αυτό δείχνει πως έχει αναλάβει το περισσότερο φόρτο εργασίας. Η χρήση επεξεργαστή του cluster φαίνεται να ακολουθεί αυτή του node1, υποδεικνύοντας ότι το node1 είναι το βασικό στοιχείο κατανάλωσης των πόρων επεξεργαστή σε αυτή την χρονική περίοδο. Το node2 (πορτοκαλί) βρίσκεται ανάμεσα στο 10-20% με μία στιγμιαία αύξηση στο 20%. Αυτό δείχνει πως έχει αναλάβει κάποιο ελαφρύ φόρτο εργασίας. Το node3 (πράσινο) έχει την πιο σταθερή χρήση και βρίσκεται ανάμεσα κοντά στο 10% παρουσιάζοντας μία στιγμιαία αύξηση στο 30%. Αυτό δείχνει πως δεν επεξεργάζεται μεγάλο φόρτο εργασίας.

Από τα παραπάνω φαίνεται ότι το node1 χειρίζεται το μεγαλύτερο φόρτο εργασίας κατά την διάρκεια εκτέλεσης του pipeline ενώ τα node2, node3 παρουσιάζουν μία πιο σταθερή χρήση του επεξεργαστή υποδεικνύοντας ότι έχουν αναλάβει μικρότερο φόρτο εργασίας ή δεν αξιοποιούνται σωστά ώστε να υπάρχει καλύτερη εξισορρόπησει φόρτου εργασίας και πόρων.

Το σχήμα 5.14 παρουσιάζει την χρήση μνήμης των node1, node2, node3 και του cluster.Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος άξονας την ποσοστιαία χρήση επεξεργαστή.



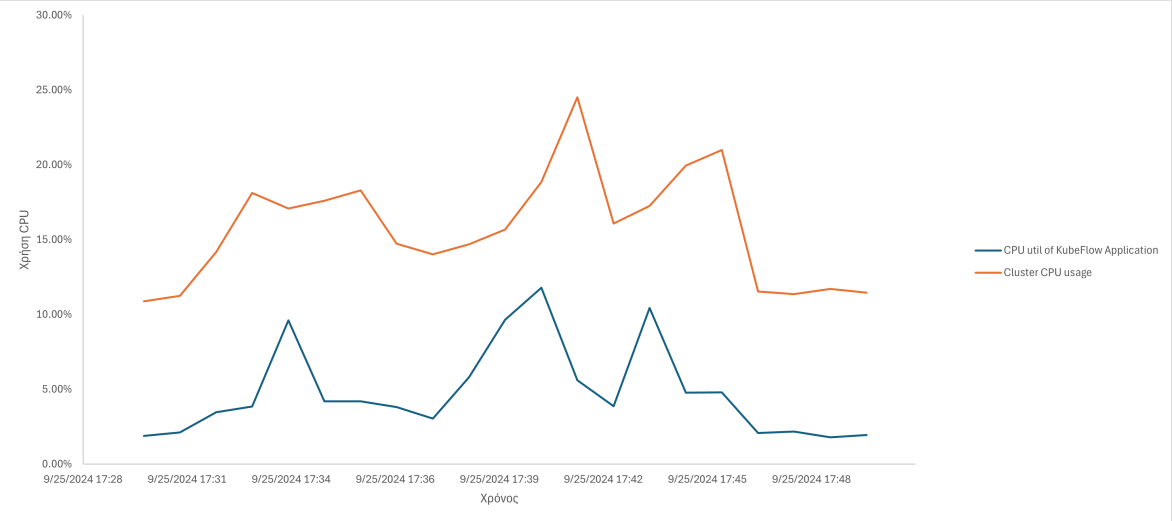

Σχήμα 5.14: Χρήση Μνήμης των κόμβων της διανομής Vanilla Kubernetes

Αρχικά, μέχρι την χρονική στιγμή 17:33 η χρήση μνήμης στους τρεις nodes και του cluster (γαλάζιο) είναι σχετικά σταθερή. Το node1 (μπλε) κυμαίνεται στο 40-42%, το node2 (πορτοκαλί) στο 44%, το node3 (πράσινο) στο 39-40% και το cluster 42-44%. Το node1 από τη στιγμή 17:34 μέχρι και την 17:45 βρίσκεται ανάμεσα στο 40-52%, παρουσιάζοντας τρεις στιγμιαίες αυξήσεις, τη στιγμή 17:36 με ποσοστό 47%, την 17:40 με ποσοστό 52% και την 17:43 με ποσοστό 50%. Αυτές οι αυξήσεις υποδεικνύουν ότι εκείνες τις χρονικές στιγμές εκτελούνταν εργασίες που είχαν σχετικά υψηλές απαιτήσεις πόρων μνήμης. Η χρήση μνήμης του cluster ακολουθεί παρόμοιο μοτίβο με το node1 με τρεις στιγμιαίες αυξήσεις στο 45%, αυτό δείχνει πως το node1 επηρεάζει σημαντικά την γενική χρήση μνήμης του cluster. Το node2 έχει παρόμοια χρήση πόρων με το node1 και βρίσκεται κοντά στο 45%, δεν παρουσιάζει κάποια σημαντική αύξηση. Το node3 βρίσκεται κοντά στο 40% χωρίς καμία αύξηση στην χρήση μνήμης, αυτό δείχνει

πως δεν χρησιμοποιείται ιδιαίτερα από την εφαρμογή. Από τη στιγμή 17:45 μέχρι την λήξη του πειράματος, το node1 έχει ποσοστό χρήσης ανάμεσα στο 40-45%, το node2 κοντά στο 45%, το node3 κοντά στο 40% και το cluster κοντά στο 45%.

Από τα παραπάνω φαίνεται πως το node1 παρουσίασε τρεις στιγμιαίες αυξήσεις λόγω εργασιών που απαιτούσαν παραπάνω μνήμη και αυτό επηρέασε και την χρήση μνήμης του cluster. Το node2 είχε παρόμοια χρήση πόρων με το node1 χωρίς κάποιες σημαντικές αυξήσεις και το node3 φαίνεται πως δεν ανέλαβε σημαντικό φόρτο εργασίας.

Το σχήμα 5.15 παρουσιάζει την χρήση επεξεργαστή του namespace της εφαρμογής Kubeflow και του cluster. Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος άξονας την ποσοστιαία χρήση επεξεργαστή.

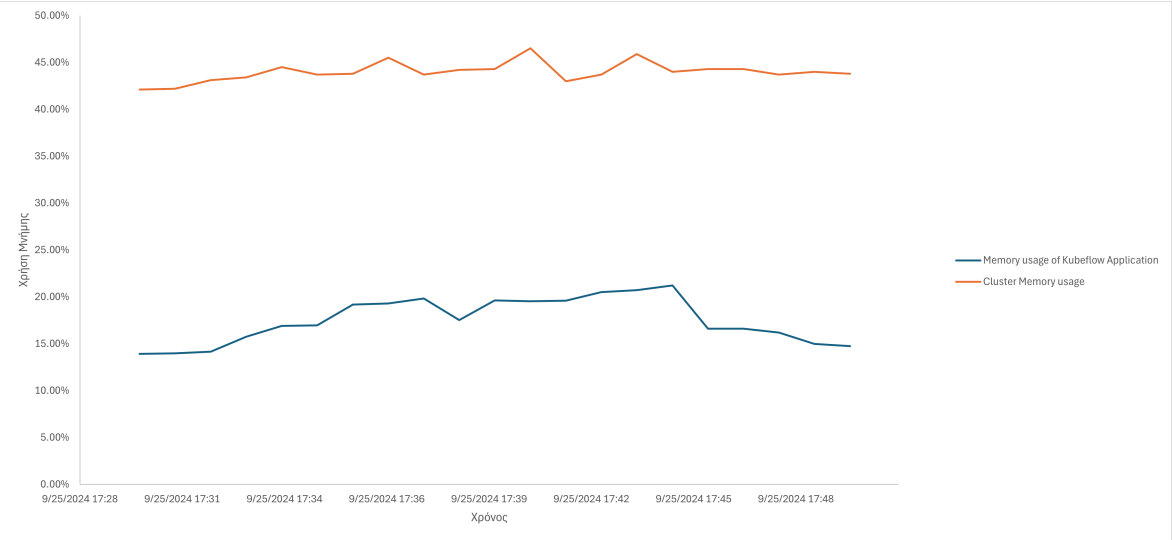



Σχήμα 5.15: Χρήση CPU του Kubeflow namespace της διανομής Vanilla Kubernetes

.

Αρχικά, μέχρι την στιγμή 17:30 η χρήση επεξεργαστή τόσο του Kubeflow (μπλε) όσο και του cluster (πορτοκαλί) παραμένει χαμηλή. Από τη στιγμή 17:31 μέχρι την 17:37 η χρήση επεξεργαστή του kubeflow φαίνεται να επηρεάζει μέχρι κάποιο βαθμό το cluster με ποσοστό κοντά στο 5% με μία στιγμιαία αύξηση στο 10%. Η χρήση πόρων του cluster αυτή την περίοδο βρίσκεται ανάμεσα στο 15-20% που υποδεικνύει ότι εκτελεί σημαντικές εργασίες του pipeline. Από την στιγμή 17:37-17:45 και οι δύο οντότητες παρουσιάζουν παρόμοιες αυξήσεις στην χρήση του επεξεργαστή, με το Kubeflow να έχει δύο στιγμιαίες αυξήσεις στο 10% και το cluster δύο στιγμιαίες αυξήσεις στο 25% και στο 23%. Αυτές οι αυξήσεις υποδεικνύουν ότι εκείνες τις στιγμές το pipeline εκτελούσε αρκετά απαιτητικές εργασίες και χρειαζόταν επεξεργαστική δύναμη. Από την στιγμή 17:45 και μετά η χρήση πόρων επεξεργαστή μειώνεται και σταθεροποιείται για το Kubeflow στο 3-4% και για το cluster στο 12%. Από τα παραπάνω φαίνεται ότι, η χρήση πόρων επεξεργαστή του Kubeflow συμβάλλει σημαντικά στην ολική χρήση επεξεργαστή του cluster. Παρόλα αυτά, η χρήση επεξεργαστή του cluster παραμένει αρκετά πιο υψηλή σχετικά με του Kubeflow που υποδεικνύει ότι εκτελούνται και άλλες εσωτερικές διεργασίες.

Το σχήμα 5.16 παρουσιάζει την χρήση μνήμης του namespace της εφαρμογής Kubeflow και του cluster. Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος άξονας την ποσοστιαία χρήση επεξεργαστή.

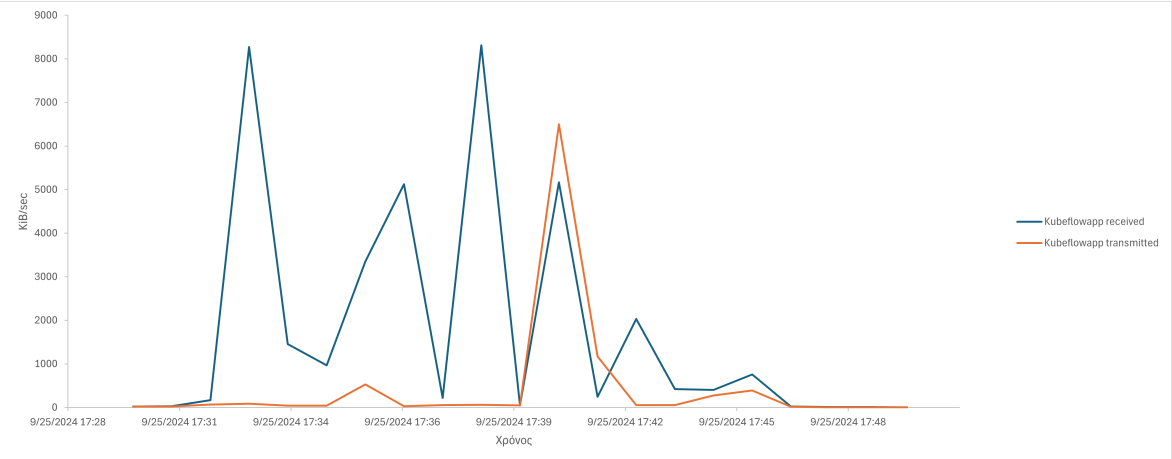



Σχήμα 5.16: Χρήση Μνήμης του Kubeflow namespace της διανομής Vanilla Kubernetes

.

Παρατειρείται ότι, πέρα από την περίοδο που εκτελείται το Pipeline όπου η χρήση μνήμης αυξάνεται ελάχιστα τόσο του cluster (πορτοκαλί) όσο και του Kubeflow (μπλε), η χρήση μνήμης του Kubeflow παραμένει σχετικά σταθερή και σε χαμηλά επίπεδα κοντά στο 15-20% και του cluster να παραμένει επίσης σχετικά σταθερή κοντά στο 40-45% πράγμα που υποδεικνύει ότι το Kubeflow δεν καταναλώνει πολλούς πόρους μνήμης και δεν έχει κάποια επίπτωση στη συνολική μνήμη που καταναλώνει το cluster.

Το σχήμα 5.17 παρουσιάζει τη δικτυακή κίνηση που δέχεται και μεταδίδει το Kubeflow namespace. Ο οριζόντιος άξονας παριστάνει τον χρόνο και ο κατακόρυφος την δικτυακή κίνηση σε KiB/sec.




Σχήμα 5.17: Διακτυακή κίνηση του Kubeflow namespace της διανομής Vanilla Kubernetes

Φαίνεται ότι η εφαρμογή κατεβάζει δεδομένα και ειδικότερα έχει κάποιες αυξήσεις τις χρονικές στιγμές, 17:33, 17:35, 17:39, 17:40 με τιμές ανάμεσα στα 5000-8000

KiB/sec για ανάλυση ή για εκτέλεση εργασιών μηχανικής μάθησης και μεταδίδει δεδομένα τη χρονική στιγμή 17:39 σε κάποιο service του cluster.

# Κεφάλαιο 6

# Συμπεράσματα και Μελλοντικές Επεκτάσεις

Στο κεφάλαιο 5, παρουσιάστηκαν τα αποτελέσματα χρήσης πόρων επεξεργαστή, μνήμης και δικτύου του κάθε ενορχηστρωτή. Συγκεκριμένα οι μετρικές που λήφθηκαν είναι η κατανάλωση πόρων του κάθε Cluster ως μία οντότητα, των κόμβων συγκριτικά με την γενική κατανάλωση του αντίστοιχου Cluster, του namespace του Kubeflow σε σχέση με του Cluster και της δικτυακής εισερχόμενης, εξερχόμενης κίνησης του namespace του Kubeflow. Παρακάτω παρουσιάζονται τα συμπεράσματα της σύγκρισης των τριών ενορχηστρωτών:

- Το K3s είναι το πιο αποδοτικό distribution τόσο στη χρήση του επεξεργαστή όσο και στη χρήση στης μνήμης. Λόγω της χαμηλής και σταθερής κατανάλωσης πόρων ακόμα και κατά τη διάρκεια της εκτέλεσης του Kubeflow pipeline, το καθιστά ιδανικό για περιβάλλοντα μικρότερης κλίμακας και για IoT εφαρμογές. Συνιστάται για περιπώσεις όπου η σοφή κατανάλωση πόρων είναι πιο σημαντική από την απόδοση.
- Το MicroK8s βρίσκεται ανάμεσα στο K3s και Vanilla Kubernetes, καθώς χρησιμοποιεί περισσότερους πόρους από το k3s και παραμένει πιο αποδοτικό από το Vanilla. Είναι κατάλληλο για περιβάλλοντα μεσαίας κλίμακας όπου απαιτείται η καλύτερη χρήση πόρων χωρίς να θυσιάζεται η επίδοση.
- Το Vanilla Kubernetes καταναλώνει τους περισσότερους πόρους και του επεξεργαστή και της μνήμης. Είναι καταλληλότερο για περιβάλλοντα μεγαλύτερης κλίμακας, διότι διαχειρίζεται καλύτερα μεγαλύτερο φόρτο εργασίας από τις υπόλοιπες δύο διανομές και προσφέρει καλύτερη επίδοση. Συνιστάται για περιπτώσεις όπου η επίδοση είναι πιο σημαντική από τη σοφή κατανάλωση πόρων.

Συνοπτικά, το K3s συνίσταται για περιβάλλοντα με περιορισμένους πόρους λόγω της χαμηλής κατανάλωσης επεξργαστή και μνήμης, ενώ το Vanilla λόγω της επιδοσής του είναι ιδανικό για περιβάλλοντα μεγαλύτερης κλίμακας. Το MicroK8s ενδείκνυται για περιβάλλοντα που χρειάζονται και επίδοση αλλά και απόδοση. Τέλος, στη δικτυακή κίνηση δεν υπήρχε κάποια διαφορά μεταξύ των τριών διανομών.

Η μελέτη που παρουσιάζεται σε αυτή τη διπλωματική αποτελεί μία βάση για περαιτέρω έρευνες στον τομέα των ενορχηστρωτών container συμβατών με το Kubernetes.

Μπορεί να επεκταθεί και να χρησιμοποιηθεί ως αφετηρία για βαθύτερη ανάλυση, με τις μελλοντικές μελέτες να επικεντρώνονται στα εξής:

- Στην αξιολόγηση στο πως διάφοροι αποθηκευτικοί χώροι, όπως Persistent Volume Claims, Container Storage Interface drivers, επηρεάζουν την χρήση πόρων και την απόδοση του Cluster. Η αποθήκευση αποτελεί σημαντικό κομμάτι της υποδομής ενός Cluster διότι επηρεάζει τη διαθεσιμότητα και την ταχύτητα ανάκτησης δεδομένων. Η μελέτη αυτών των αποθηκευτικών χώρων και πως επιδρούν στην κατανάλωση των πόρων του συστήματος, μπορούν να βοηθήσουν τους ερευνητές να τους βελτιστοποιήσουν ώστε να υπάρχει καλύτερη απόδοση και διαχείριση τόσο των αποθηκευτικών χώρων όσο και των πόρων των συστημάτων.
- Στην ανάλυση της κατανάλωσης ενέργειας και του κόστους των κόμβων του κάθε ενορχηστρωτή κατά την διάρκεια εκτέλεσης των πειραμάτων. Η ανάλυση των απαιτήσεων ενέργειας των κόμβων του Cluster και η σύγκριση με διαφορετικούς ενορχηστρωτές container, μπορεί να οδηγήσει σε στρατηγικές μείωσης της κατανάλωσης ενέργειας, προσφέροντας πιο αποδοτικές λύσεις για μεγάλες κλίμακες εγκαταστάσεων.
- Στη Δοκιμή διαφορετικών εφαρμογών, για παράδειγμα βάσεις δεδομένων ή εργαλειών επεξεργασείας μεγάλων δεδομένων, για την παρατήρηση της χρήσης πόρων και συμπεριφοράς των ενορχηστρωτών.
- Στην επέκταση του χρόνου δοκιμών ώστε να παρατηρηθεί η συμπεριφορά των ενορχηστρωτών στη χρήση του επεξεργαστή, της μνήμης και του δικτύου υπό παρατεταμένο φόρτο εργασίας. Σε παρατεταμένες χρονικές δοκιμές, υπάρχει περίπτωση εμφάνισης διάφορων προβλημάτων όπως, η διαρροή μνήμης, δηλαδή μία διεργασία αφότου ολοκληρωθεί να μην αποδεσμεύσει το κομμάτι μνήμης που χρησιμοποιούσε, ή η μη αποδοτική κλιμάκωση της εφαρμογής.
- Καθώς πολλές διανομές εγκαθιστούνται σε εικονικές μηχανές, αξίζει να μελετηθεί πως διάφοροι επόπτες επηρεάζουν την χρονοδρομολόγηση εφαρμογών και τις πολιτικές διαχείρησης πόρων.

# Παράρτημα Α΄

# Ακρωνύμια και συντομογραφίες

**LAN** Local Area Network

**ACL** Access Control List

**AI** Artificial Intelligence

**BGP** Border Gateway Protocol

**CPU** Central Processing Unit

**CNCF** Cloud Native Computing Foundation

**CNI** Container Networking Interface

**DNS** Domain Name System

**IoT** Internet of Things

**IP** Internet Protocol

**IPAM** IP Address Management

**ML** Machine Learning

**NAT** Network Address Translation

**NAS** Network Attached Storage

**RAM** Random Access Memory

**SLA** Service Level Agreement

**VM** Virtual Machine

**Η/Υ** Ηλεκτρονικός Υπολογιστής

# Βιβλιογραφία


[1] S. Telenyk, O. Sopov, E. Zharikov, and G. Nowakowski, "A comparison of kubernetes and kubernetes-compatible platforms," in *2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, vol. 1, 2021, pp. 313–317.

[2] G. Sayfan, *Mastering Kubernetes: Master the art of container management by using the power of Kubernetes*. Packt Publishing Ltd, 2018.

[3] M.-N. Tran, D.-D. Vu, and Y. Kim, "A survey of autoscaling in kubernetes," in *2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2022, pp. 263–265.

[4] Z. Rejiba and J. Chamanara, "Custom scheduling in kubernetes: A survey on common problems and solution approaches," *ACM Comput. Surv.*, vol. 55, no. 7, Dec. 2022. [Online]. Available: https://doi.org/10.1145/3544788

[5] L. Mercl and J. Pavlik, "The comparison of container orchestrators," in *Third International Congress on Information and Communication Technology*, X.-S. Yang, S. Sherratt, N. Dey, and A. Joshi, Eds. Singapore: Springer Singapore, 2019, pp. 677–685.

[6] J. Kabbedijk, M. Pors, S. Jansen, and S. Brinkkemper, "Multi-tenant architecture comparison," in *Software Architecture: 8th European Conference, ECSA 2014, Vienna, Austria, August 25-29, 2014. Proceedings 8*. Springer, 2014, pp. 202–209.

[7] A. Randal, "The ideal versus the real: Revisiting the history of virtual machines and containers," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–31, 2020.

[8] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE cloud computing*, vol. 1, no. 3, pp. 81–84, 2014.

[9] B. W. Kernighan, "Unix time-sharing system," *Document Preparation, Bell Syst. Tech. J.*, vol. 57, no. 6, pp. 2115–2135, 1978.

[10] A. Khan, "Key characteristics of a container orchestration platform to enable a modern application," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 42–48, 2017.

[11] Kubernetes overview. [Online]. Available: https://kubernetes.io/docs/concepts/overview/

[12] S. N. T.-c. Chiueh and S. Brook, "A survey on virtualization technologies," *Rpe Report*, vol. 142, 2005.

[13] M. Pearce, S. Zeadally, and R. Hunt, "Virtualization: Issues, security threats, and solutions," *ACM Computing Surveys (CSUR)*, vol. 45, no. 2, pp. 1–39, 2013.

[14] J. Surbiryala and C. Rong, "Cloud computing: History and overview," in *2019 IEEE Cloud Summit*, 2019, pp. 1–7.

[15] C. Yström and A. Stenborg, "Performance comparison between a kubernetes cluster and an embedded system," 2021.

[16] M. Chebiyyam, R. Malviya, S. K. Bose, and S. Sundarrajan, "Server consolidation: Leveraging the benefits of virtualization," *Infosys Research, SETLabs Briefings*, vol. 7, no. 1, pp. 65–75, 2009.

[17] A. Tosatto, P. Ruiu, and A. Attanasio, "Container-based orchestration in cloud: State of the art and challenges," in *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*, 2015, pp. 70–75.

[18] I. M. A. Jawarneh, P. Bellavista, F. Bosi, L. Foschini, G. Martuscelli, R. Montanari, and A. Palopoli, "Container orchestration engines: A thorough functional and performance comparison," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[19] R. Kandan, M. F. Khalid, B. I. Ismail, and O. H. Hoe, "Advanced resource allocation and service level monitoring for container orchestration platform," in *2019 IEEE International Conference on Sensors and Nanotechnology*. IEEE, 2019, pp. 1–4.

[20] H. Koziolek and N. Eskandani, "Lightweight kubernetes distributions: A performance comparison of microk8s, k3s, k0s, and microshift," in *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 17–29. [Online]. Available: https://doi.org/10.1145/3578244.3583737

[21] P. Ascensão, L. F. Neto, K. Velasquez, and D. P. Abreu, "Assessing kubernetes distributions: A comparative study," in *2024 IEEE 22nd Mediterranean Electrotechnical Conference (MELECON)*. IEEE, 2024, pp. 832–837.

[22] V. Sanjana, P. Aruna, A. Chitra *et al.*, "An empirical study on dockers and virtual machines for hosting react. js web application," in *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. IEEE, 2023, pp. 1–5.

[23] I. Vasireddy, P. Kandi, and S. Gandu, "Efficient resource utilization in kubernetes: A review of load balancing solutions," *International Journal of Innovative Research in Engineering & Management*, vol. 10, no. 6, pp. 44–48, 2023.

[24] I. Vasireddy, G. Ramya, and P. Kandi, "Kubernetes and docker load balancing: State-of-the-art techniques and challenges," *International Journal of Innovative Research in Engineering & Management*, vol. 10, no. 6, pp. 49–54, 2023.

[25] E. Truyen, N. Kratzke, D. Van Landuyt, B. Lagaisse, and W. Joosen, "Managing feature compatibility in kubernetes: Vendor comparison and analysis," *IEEE Access*, vol. 8, pp. 228 420–228 439, 2020.

[26] A. E. Nocentino and B. Weissman, *Kubernetes Architecture*. Berkeley, CA: Apress, 2021, pp. 53–70. [Online]. Available: https://doi.org/10.1007/978-1-4842-7192-6_3

[27] T. Kubernetes, "Kubernetes," *Kubernetes. Retrieved May*, vol. 24, p. 2019, 2019.

[28] C.-C. Chen, M.-H. Hung, K.-C. Lai, and Y.-C. Lin, *Docker and Kubernetes*. John Wiley & Sons, Ltd, 2021, ch. 5, pp. 169–213. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119739920.ch5

[29] A. Jeffery, H. Howard, and R. Mortier, "Rearchitecting kubernetes for the edge," in *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*, ser. EdgeSys '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 7–12. [Online]. Available: https://doi.org/10.1145/3434770.3459730

[30] K. Manaouil and A. Lebre, "Kubernetes and the Edge?" Inria Rennes - Bretagne Atlantique, Research Report RR-9370, Oct. 2020. [Online]. Available: https://inria.hal.science/hal-02972686

[31] L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "Kubernetes as an availability manager for microservice applications," 2019. [Online]. Available: https://arxiv.org/abs/1901.04946

[32] N. Poulton, *The kubernetes book*. NIGEL POULTON LTD, 2023.

[33] "docker — docker.com," https://www.docker.com/, [Accessed 20-10-2024].

[34] "containerd — containerd.io," https://containerd.io/, [Accessed 20-10-2024].

[35] "cri-o — cri-o.io," https://cri-o.io/, [Accessed 20-10-2024].

[36] Kubernetes pods. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/pods/

[37] Create static Pods. [Online]. Available: https://kubernetes.io/docs/tasks/configure-pod-container/static-pod/

[38] Kubernetes pod lifecycle. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/

[39] Kubernetes deployments. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

[40] Kubernetes statefulsets. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

[41] Kubernetes daemonset. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/

[42] Kubernetes persistent volumes. [Online]. Available: https://kubernetes.io/docs/concepts/storage/persistent-volumes/

[43] Kubernetes service. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/service/

[44] Kubernetes namespaces. [Online]. Available: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

[45] Kubernetes ConfigMaps. [Online]. Available: https://kubernetes.io/docs/concepts/configuration/configmap/

[46] Kubernetes secrets. [Online]. Available: https://kubernetes.io/docs/concepts/configuration/secret/

[47] Kubernetesjobs. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/job/

[48] Kubernetes labels and Selectors. [Online]. Available: https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

[49] Z. Kang, K. An, A. Gokhale, and P. Pazandak, "A comprehensive performance evaluation of different kubernetes cni plugins for edge-based and containerized publish/subscribe applications," in *2021 IEEE International Conference on Cloud Engineering (IC2E)*, 2021, pp. 31–42.

[50] S. Qi, S. G. Kulkarni, and K. K. Ramakrishnan, "Assessing container network interface plugins: Functionality, performance, and scalability," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 656–671, 2021.

[51] R. Kumar and M. C. Trivedi, "Networking analysis and performance comparison of kubernetes cni plugins," in *Advances in Computer, Communication and Computational Sciences*, S. K. Bhatia, S. Tiwari, S. Ruidan, M. C. Trivedi, and K. K. Mishra, Eds. Singapore: Springer Singapore, 2021, pp. 99–109.

[52] G. Koukis, S. Skaperas, I. A. Kapetanidou, L. Mamatas, and V. Tsaoussidis, "Performance evaluation of kubernetes networking approaches across constraint edge environments," 2024. [Online]. Available: https://arxiv.org/abs/2401.07674

[53] S. Novianti and A. Basuki, "The performance analysis of container networking interface plugins in kubernetes," in *Proceedings of the 6th International Conference on Sustainable Information Engineering and Technology*, ser. SIET '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 231–234. [Online]. Available: https://doi.org/10.1145/3479645.3479700

[54] "About Calico | Calico Documentation — docs.tigera.io," https://docs.tigera.io/calico/latest/about, [Accessed 20-10-2024].

[55] "Cilium - Cloud Native, eBPF-based Networking, Observability, and Security — cilium.io," https://cilium.io/, [Accessed 20-10-2024].

[56] "GitHub - flannel-io/flannel: flannel is a network fabric for containers, designed for Kubernetes — github.com," https://github.com/flannel-io/flannel, [Accessed 20-10-2024].

[57] "Antrea — antrea.io," https://antrea.io/, [Accessed 20-10-2024].

[58] "Kube-OVN | The Most Advanced Kubernetes Network Fabric for Enterprises — kube-ovn.io," https://www.kube-ovn.io/, [Accessed 20-10-2024].

[59] S. Qi, S. G. Kulkarni, and K. K. Ramakrishnan, "Understanding container network interface plugins: Design considerations and performance," in *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN*, 2020, pp. 1–6.

[60] S. Qi, S. G. Kulkarni, and K. Ramakrishnan, "Assessing container network interface plugins: Functionality, performance, and scalability," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 656–671, 2020.

[61] Y. YANG, X. DONG, and Z. DONG, "Technical analysis of network plug-in flannel for containers," *ZTE Communications*, vol. 15, no. 4, pp. 43–46, 2019.

[62] V. Dakić, J. Redžepagić, M. Bašić, and L. Žgrablić, "Performance and latency efficiency evaluation of kubernetes container network interfaces for built-in and custom tuned profiles," 2024.

[63] S. Böhm and G. Wirtz, "Profiling lightweight container platforms: Microk8s and k3s in comparison to kubernetes." in *ZEUS*, 2021, pp. 65–73.

[64] M. Moravcik, M. Kontsek, P. Segec, and D. Cymbalak, "Kubernetes-evolution of virtualization," in *2022 20th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. IEEE, 2022, pp. 454–459.