Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Επικοινωνιών, Ηλεκτρονικής και
Συστημάτων Πληροφορικής

Μεταφορά εικονικά υλοποιούμενων εφαρμογών στην υπολογιστική αρχιτεκτονική στα άκρα του δικτύου με χρήση Κυβερνήτη

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΣΤΟΥΡΝΑΡΑΣ ΑΛΕΞΙΟΣ

Επιβλέπων : Συμεών Παπαβασιλείου
Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2020

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Επικοινωνιών, Ηλεκτρονικής και
Συστημάτων Πληροφορικής

Μεταφορά εικονικά υλοποιούμενων εφαρμογών στην υπολογιστική αρχιτεκτονική στα άκρα του δικτύου με χρήση Κυβερνήτη

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΣΤΟΥΡΝΑΡΑΣ ΑΛΕΞΙΟΣ

Επιβλέπων : Συμεών Παπαβασιλείου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 11η Μαρτίου 2020.

Συμεών Παπαβασιλείου Καθηγητής Ε.Μ.Π.	Θεοδώρα Βαρβαρίγου Καθηγήτρια Ε.Μ.Π.	Ιωάννα Ρουσσάκη Επίκ. Καθηγήτρια Ε.Μ.Π.

Αθήνα, Μάρτιος 2020

Στουρνάρας Αλέξιος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Τα τελευταία χρόνια παρατηρείται μια ραγδαία αύξηση των συσκευών που ανήκουν στο Διαδίκτυο των Αντικειμένων, οι οποίες, προκειμένου να είναι εύχρηστες και εύκολες στη μεταφορά τους, διαθέτουν περιορισμένους πόρους. Επομένως,η χρήση των εφαρμογών που προσφέρουν οι εν λόγω συσκευές καθιστούν επιτακτική την ανάγκη αξιοποίησης του υπολογιστικού νέφους. Ωστόσο, η ιδέα της αντιμετώπισης του νέφους ως απομονωμένα κέντρα δεδομένων θεωρείται πλέον παρωχημένη,αφού προκαλείται έτσι μεγάλη συμφόρηση στο δίκτυο, που οδηγεί σε μεγάλους χρόνους αναμονής και συχνά καταλήγει σε μη ανταπόκριση των κέντρων αυτών.Προς αποφυγή των δυσκολιών,οι ειδικοί έχουν στρέψει την προσοχή τους στην χρήση της υπολογιστικής αρχιτεκτονικής στα άκρα του δικτύου για την μεταφόρτωση του υπολογιστικού φόρτου των εφαρμογών.Με την τεχνική αυτή, ισομοιράζονται κατάλληλα οι πόροι των κόμβων που ανήκουν στο νέφος και μειώνονται ταυτόχρονα οι χρόνοι αποκρίσης λόγω εγγύτητας των κόμβων στην παραγωγή των δεδομένων.Σκοπός της διπλωματικής εργασίας είναι η βελτιστοποίηση της μεθόδου αυτής με τη χρήση αλγορίθμου που στοχεύει στην μετατόπιση του υπολογιστικού φόρτου της κινητής συσκευής στον κάθε φορά κοντινότερο κόμβο της. Αυτό έχει επιτευχθεί με την υλοποίηση των εφαρμογών σε containers και την αξιοποίηση του Κυβερνήτη, ένα πρόγραμμα Ανοιχτού Πηγαίου Κώδικα για τον αυτόματο χειρισμό των εφαρμογών αυτών. Τέλος, τα πειράματα που διεξάχθηκαν επιβεβαιώνουν τη χρησιμότητα αυτής της υλοποίησης, τόσο ως προς το χρόνο και την ταχύτητα διεκπεραίωσης της εφαρμογής, όσο και ως προς την σωστή αξιοποίηση των πόρων του κάθε κόμβου.

Λέξεις κλειδιά

Υπολογιστικό νέφος, υπολογιστική κινητού νέφους, υπολογιστική στα άκρα του δικτύου, Κυβερνήτης, εικόνες, συσκευασμένες εφαρμογές, μετακίνηση , μετατόπιση υπολογιστικού φόρτου

Abstract

In recent years there has been a rapid increase in Internet of Things devices, which, in order to be easily operated and portable, have intrinsic resource constraints.Hence, the need for proper cloud utilization for the applications these devices offer becomes imperative.However, the idea of treating the cloud as isolated data centers is now considered obsolete, since it causes network congestion, which in turn leads to latency and downtime problems.To avoid these difficulties,experts have turned their attention to the use of computational offloading of the apps to the edge of the cloud, called edge computing.With this technique, the resources of the nodes that belong to the edge of the cloud are properly distributed between the apps and the latency is greatly reduced due to the close proximity of the nodes to the data production.The goal of this diploma thesis is to further improve this technique by using an algorithm which aims to offload the mobile device's computations to the nearest node,and transfer them to another according to the device's location. To achieve this feat, we will develop the apps with docker containers and utilize Kubernetes, an Open Source Code project that automatically orchestrates those containers.In the end, experiments are contacted that prove the usefulness of this implementation, in regards to the time and speed of the app's runtime, as well as the proper exploitation of each of the node's resources.

Key words

cloud computing,mobile cloud computing,edge computing ,Kubernetes,k3s,images,containers,migration, computational offloading

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον καθηγητή κύριο Συμεών Παπαβασιλείου, ο οποίος με την εμπιστοσύνη που μου έδειξε μου έδωσε την κατάλληλη ώθηση ώστε να βρω τι πραγματικά με ενδιαφέρει, καλλιέργώντας μου έτσι το ενδιαφέρον για τον τομέα των Δικτυακών Συστημάτων.

Επίσης, θα ήθελα να ευχαριστήσω τον μεταδιδακτορικό Δημήτρη Δεχουνιώτη και τους διδακτορικούς ερευνητές Μάριο Αυγέρη και Δημήτρη Σπαθαράκη που με την επιμονή, την υπομονή, το χρόνο και τη διάθεσή τους κατάφερα να ολοκληρώσω αυτή την διπλωματική εργασία.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου και τον αδελφό μου για όλα αυτά που μου προσέφεραν και κατάφερα να φτάσω εδώ που είμαι καθώς και τους φίλους μου οι οποίοι όλο αυτόν τον καιρό με στήριξαν και μου βελτίωναν τη διάθεση σε κάθε δύσκολη στιγμή.

Στουρνάρας Αλέξιος,

Αθήνα, 11η Μαρτίου 2020

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

Κεφάλαιο 1

Εισαγωγή

1.1 Διαδίκτυο των Αντικειμένων

Η ραγδαία ανάπτυξη της τεχνολογίας των ασύρματων επικοινωνιών, όπως οι τεχνολογίες 4G/5G, καθώς και των υπολογιστικών συστημάτων, έχει συμβάλλει σημαντικά στην εξέλιξη του Διαδικτύου των Αντικειμένων-Internet of Things (IoT). Ειδικότερα, κατασκευάζονται όλο και περισσότερες "έξυπνες" συσκευές και αντικείμενα που μπορούν να ανταλλάξουν πλέον απευθείας πληροφορίες και δεδομένα μεταξύ τους, χωρίς να χρειάζεται ανθρώπινη παρουσία για την χρήση αυτών. [1]. Αυτός ο νέος τρόπος υλοποίησης επικοινωνίας προσφέρει μία πληθώρα δυνατοτήτων και βελτιστοποιήσεων σε πολλούς τομείς της καθημερινής μας ζωής. Η αυτοματοποίηση της γεωργίας με κατάλληλα γεωργικά μηχανήματα [2], η άμεση επικοινωνία μεταξύ γιατρού και ασθενή με ασφάλη μεταφορά των ιατρικών δεδομένων [3],[4],[5], η δημιουργία έξυπνων δικτύων ενέργειας (smart grids) για τη σωστή διαχείριση της ενέργειας που χρειάζεται ένα κτήριο [6],η δημιουργία οικιακών συσκευών οι οποίες θα διαχειρίζονται αυτόματα όλες τις λειτουργίες ενός σπιτιού [7] και τα έξυπντα αυτοκίνητα [8]είναι μόνο μερικά από τα παραδείγματα που φανερώνουν τη σημασία της ένταξης του IoT στην καθημερινότητά μας. Με στόχο λοιπόν την σταδιακή αυτοματοποίηση των περισσότερων τομέων της και κατ□επέκταση την ύπαρξη τελικά αυτόνομων λειτουργικά πόλεων [9], [10], [11] δημιουργούνται κατάλληλες εφαρμογές οι οποίες ρυθμίζουν όλες τις προαναφερθείσες λειτουργίες, βασιζόμενες σε μετρήσεις και δεδομένα που έχουν λάβει από τις συσκευές όπου λειτουργούν. Ωστόσο, οι συσκευές αυτές (αισθητήρες, κινητά, ενσωματωμένα συστήματα) έχουν πολλά μειονεκτήματα, όπως οι περιορισμένοι διαθέσιμοι πόροι και η πεπερασμένη πηγή ενέργειας (μπαταρία), τα οποία εμποδίζουν την έγκαιρη και σωστή λειτουργία των ολοένα και πιο περίπλοκων και ενεργοβόρων εφαρμογών που δημιουργούνται. Επιπλέον, οι δυσκολίες που περιγράφηκαν είναι σχεδόν αδύνατο να λυθούν καθώς αυτό θα σήμαινε μειωμένη κινητικότητα και δυσχρηστία των συσκευών, λόγω αύξησης του μεγέθους . Προς αποφυγή αυτών, οι ειδικοί έχουν στραφεί στην χρήση του υπολογιστικού νέφους-cloud computing (CC) για την σωστή ολοκλήρωση των εφαρμογών, χρησιμοποιώντας μια τεχνική που ονομάζεται μεταφόρτωση υπολογισμών-computational offloading. (CO). Πιο συγκεκριμένα, αξιοποιούνται οι πόροι και οι υπηρεσίες που προσφέρουν τα κέντρα δεδομένων στο νέφος, μειώνοντας έτσι σε μεγάλο βαθμό την χρήση των πόρων των κινητών συσκευών. Βέβαια, η συνεχής αύξηση των εφαρμογών και συσκευών, προκαλεί νέα προβλήματα, αναγκάζοντας έτσι τους ειδικούς να αλλάξουν τον τρόπο αντιμετώπισης του νέφους και να στραφούν σε νέες πιθανές λύσεις.

1.1.1 Υπολογιστική στα άκρα του δικτύου

Σύμφωνα με προβλέψεις της Cisco, μέχρι το τέλος του 2020 θα υπάρχουν 50 δισεκατομμύρια συσκευές που θα ανήκουν στο IoT [12]. Με τα δεδομένα λοιπόν να αυξάνονται σε πλήθος όλο και περισσότερο, καθίσταται δύσκολη, έως και αδύνατη η γρήγορη και σωστή ανταπόκριση των κέντρων δεδομένων [13]. Επιπροσθέτως, ενώ δημιουργούνται όλο και γρηγορότερες μέθοδοι επεξεργασίας των δεδομένων, οι ταχύτητες των γραμμών μέσω των οποίων μεταφέρονται τα δεδομένα στο νέφος έχουν μείνει σχετικά σταθερές. Άρα, με την ποσότητα των δεδομένων να αυξάνεται, η μεταφορά τους αργεί όλο και περισσότερο, αφού δημιουργείται ένα φαινόμενο συμφόρησης, και οι χρόνοι ανταπόκρισης μεγαλώνουν σε βαθμό που δεν είναι πλέον αποδεκτοί. Τέλος, μέχρι πρόσφατα οι συσκευές που βρίσκονταν στα άκρα του δικτύου ήταν κυρίως καταναλωτές δεδομένων, παραδείγματος χάριν ζήταγαν από έναν εξυπηρετητή ένα αρχείο ή δεδομένα για χρήση. Πλέον, με τα μέσα κοινωνικής δικτύωσης και άλλες υπηρεσίες διαθέσιμες όπως οι πολυμεσικές υπηρεσίες, οι χρήστες μπορούν να παράγουν και να ανεβάζουν δεδομένα, αυξάνοντας ακόμα περισσότερο το πλήθος των δεδομένων και κυρίως, αναγκάζουν τους κόμβους στα άκρα να παρέχουν περισσότερες λειτουργικότητες. Η υπολογιστική στα άκρα του δικτύου αποτελεί την λύση όλων αυτών των δυσκολιών. Με τον όρο υπολογιστική στα άκρα του δικτύου-edge computing (EC) αναφερόμαστε στην τεχνολογία που επιτρέπει υπολογισμούς και εφαρμογές να εκτελούνται σε κόμβους που βρίσκονται στα άκρα του δικτύου, δηλαδή όσο πιο μακριά γίνεται από τα κέντρα δεδομένων και όσο πιο κοντά στον χρήστη που είναι συνδεδεμένος σε αυτό [14]. Τέτοιοι κόμβοι είναι παραδείγματος χάριν οι εξυπηρετητές-servers που ανήκουν στο υπολογιστικό νέφος και ταυτόχρονα ανήκουν στο ίδιο υποδίκτυο με διάφορες IoT συσκευές, έχοντας έτσι τη δυνατότητα να επικοινωνήσουν απευθείας μαζί τους. Η τεχνική αυτή επιτρέπει ουσιαστικά στους υπολογισμούς να γίνονται όσο το δυνατόν πιο κοντά στην παραγωγή των δεδομένων έτσι ώστε να μειώνεται ο χρόνος απόκρισης των εφαρμογών, καθώς δε θα χρειάζεται τα δεδομένα να ταξιδέψουν μεγάλες αποστάσεις, και στο να γίνεται καλύτερη κατανομή των πόρων του νέφους. Μάλιστα, με την χρήση πολλών μικρών EC κέντρων, που είναι γεωγραφικά κατανεμημένα με κατάλληλο τρόπο, καταμερίζεται ο φόρτος εργασίας.

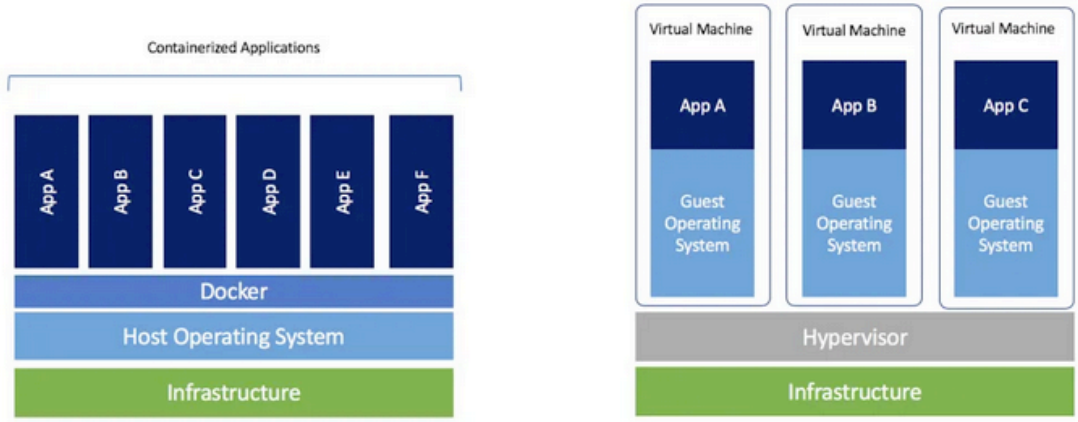
1.1.2 Υπολογιστική αρχιτεκτονική στα άκρα του δικτύου πολλαπλής πρόσβασης

Μία ειδική κατηγορία του edge computing ονομάζεται υπολογιστική αρχιτεκτονική στα άκρα του δικτύου πολλαπλής πρόσβασης-multi access edge computing (MEC). Ειδικότερα, με τον όρο αυτό εννοούμε την πλατφόρμα η οποία χρησιμοποιεί το δίκτυο ασύρματης πρόσβασης-radio access network για 4G και 5G και προσφέρει δυνατότητες για cloud computing σε κινητούς χρήστες [15]. Ουσιαστικά η διαφοροποίηση έγκειται στο γεγονός ότι πλέον συμπεριλαμβάνουμε έτσι και εφαρμογές που δεν σχετίζονται αποκλειστικά με κινητά τηλέφωνα αλλά και με πιο περίπλοκες εφαρμογές, όπως πολυμεσικές υπηρεσίες, ανάλυση βίντεο και εικόνας, μηχανική μάθηση και εφαρμογές επαυξημένης πραγματικότητας [16].Στη διπλωματική εργασία θα ασχοληθούμε κυρίως με αυτό το είδος.

1.1.3 Εικόνες και συσκευασμένες εφαρμογές

Μία ακόμα πρωτοπόρα τεχνολογία που συμβάλλει σε μεγάλο βαθμό στη σωστή αξιοποίηση των πόρων και στην ευκολία χρήσης του νέφους από τις εφαρμογές είναι οι docker εικόνες-images και οι συσκευασμένες εφαρμογές σε δοχεία-containers [17]. Όπως και τα εικονικά μηχανήματα-Virtual Machines (VMs), η τεχνολογία αυτή αποτελεί έναν τρόπο εικονοποίησης των εφαρμογών. Πιο συγκεκριμένα, το container είναι μία τυποποιημένη μονάδα λογισμικού στο επίπεδο εφαρμογής που περιέχει τον κώδικα καθώς και όλα τα απαραίτητα για τη λειτουργία του εργαλεία που δεν υπάρχουν εξ΄αρχής ως βιβλιοθήκες στη γλώσσα προγραμματισμού που χρησιμοποιεί, έτσι ώστε η εφαρμογή να τρέχει γρήγορα και αξιόπιστα από το ένα υπολογιστικό περιβάλλον στο άλλο. Μία docker image είναι αντίστοιχα ένα ελαφρύ, αυτόνομο, εκτελέσιμο πακέτο λογισμικού που περιλαμβάνει όλα όσα απαιτούνται για την εκτέλεση μιας εφαρμογής: κώδικα, εργαλεία συστήματος, βιβλιοθήκες συστήματος και ρυθμίσεις. Η τεχνολογία αυτή είναι Ανοιχτού Πηγαίου Κώδικα και μπορεί να λειτουργήσει

σε όλα τα λειτουργικά συστήματα, σε κέντρα δεδομένων και στο υπολογιστικό νέφος και επιτρέπει στους δημιουργούς εφαρμογών να εστιάσουν μόνο στην εφαρμογή και όχι στην υποδομή που αυτές χρειάζονται.



Σχήμα 1: Διαφοροποίηση μεταξύ containers και VMs [18]

Στην παραπάνω εικόνα φαίνεται η διαφοροποίηση μεταξύ docker containers και VMs. Πιο συγκεκριμένα, παρατηρούμε ότι ενώ το κάθε εικονικό μηχάνημα έχει το δικό του λειτουργικό σύστημα, τα containers μοιράζονται μεταξύ τους το ίδιο λειτουργικό σύστημα που υπάρχει στο μηχάνημα καθώς και τον ίδιο πυρήνα. Από μόνο του αυτό το χαρακτηριστικό καθιστά τα containers πιο γρήγορα στην χρήση τους, αφού χρειάζονται μόνο μερικά δευτερόλεπτα για να λειτουργήσουν και λιγότερο αποθηκευτικό χώρο, σε αντίθεση με τα VMs τα οποία χρειάζονται μεγάλο αποθηκευτικό χώρο και θέλουν αρκετά λεπτά για να εκκινήσουν. Επομένως, τα containers δίνουν στους προγραμματιστές και στους ίδιους τους χρήστες μία μεγαλύτερη ευελιξία, όσον αφορά τη χρήση και δημιουργία των εφαρμογών. Επιπλέον, λόγω του ότι μπορεί να λειτουργήσουν σε πολλά λειτουργικά συστήματα και μηχανήματα, μπορούν εύκολα να μεταφερθούν από το νέφος σε τοπικά μηχανήματα και αντίστροφα, γεγονός που τους δίνει μεγάλη κινητικότητα και κλιμακωσιμότητα, χωρίς να μειώνεται η ποιότητα της λειτουργίας των εφαρμογών [19].Τέλος, επειδή είναι μικρά σε μέγεθος, ένα μηχάνημα μπορεί να έχει μεγαλύτερο αριθμό containers από εικονικά μηχανήματα. Το τελευταίο αυτό πλεονέκτημα μάλιστα επιτρέπει στους προγραμματιστές και σχεδιαστές συστημάτων να χωρίσουν τις εφαρμογές τους σε πολλά μικρά τμήματα, κάθε ένα από τα οποία τρέχει ως μία εικόνα σε ένα container,χωρίς να εξαρτάται πραγματικά από τα υπόλοιπα τμήματα. Αυτή η διαδικασία που ονομάζεται χωρισμός σε μικροϋπηρεσίες-microservices [20],είναι ο πλέον βασικός τρόπος υλοποίησης εφαρμογών. Με βάση όλα αυτά, καταλαβαίνουμε ότι η μεταφόρτωση υπολογισμών και η χρήση του υπολογιστικού κινητού νέφους διευκολύνεται ακόμα περισσότερο και επιτρέπει τη χρήση ακόμα περισσότερων εφαρμογών σε αυτό.

1.2 Σκοπός της Διπλωματικής Εργασίας

Οι εφαρμογές χωρίζονται σε εφαρμογές με ή χωρίς κατάσταση (stateful και stateless αντίστοιχα), ανάλογα με το αν αποθηκεύουν ή όχι δεδομένα που εισάγει ο χρήστης ώστε να τα χρησιμοποιήσουν την επόμενη φορά που θα ξανασυνδεθεί ο ίδιος χρήστης. Για το δεύτερο είδος χρησιμοποιείται κατά κόρον η τεχνική της μεταφόρτωσης υπολογισμού με χρήση containers, καθώς είναι πολύ εύκολο να μεταφερθεί η εφαρμογή στο edge, χωρίς να επηρεαστεί αρνητικά η λειτουργία της. Για το πρώτο είδος όμως, η διαδικασία περιπλέκεται σε μεγάλο βαθμό, καθώς πρέπει να μεταφερθεί και την κατάσταση της, ώστε να συνεχίσει από το σημείο που την είχαμε σταματήσει και να μην χρειαστεί να ξεκινήσει από την αρχή, αφού αυτό θα σήμαινε μεγάλη καθυστέρηση στην συνολική εκτέλεση της.
Σκοπός της διπλωματικής εργασίας είναι η μεταφόρτωση υπολογισμού στα άκρα του υπολογιστικού νέφους για μία εφαρμογή με κατάσταση και ταυτόχρονα, ανάλογα με το που βρίσκεται ο χρήστης και η κινητή συσκευή του, η μεταφορά της εφαρμογής στον πιο κοντινό κόμβο του άκρου του δικτύου ως προς τη συσκευή. Από εκεί, η εφαρμογή έχοντας διατηρήσει την κατάσταση της, συνεχίζει από το σημείο που είχε σταματήσει στον προηγούμενο κόμβο. Έτσι, όταν ο χρήστης θελήσει να λάβει τα αποτελέσματα της επεξεργασίας των δεδομένων, αυτά θα βρίσκονται ήδη στον πιο κοντινό κόμβο σε αυτόν, μειώνοντας έτσι δραστικά τον χρόνο απόκρισης της εφαρμογής. Επίσης, λόγω χρήσης containers, και επειδή αυτό κάθε φορά μεταφέρεται ανάλογα με τη θέση του χρήστη, ο φόρτος εργασίας καταμερίζεται μεταξύ των κόμβων, ειδικά για παραπάνω του ενός χρηστών, γεγονός που μειώνει σημαντικά την πιθανότητα μη ανταπόκρισης του κόμβου. Για την επίτευξη του στόχου αυτού χρησιμοποιείται ο Κυβερνήτης-Kubernetes [21], ένα εργαλείο κατάλληλο για την ενορχήστρωση των containers και την σωστή λειτουργία των docker images. Ειδικότερα, χρησιμοποιείται μία πιο ελαφριά έκδοση αυτού, το k3s [22],το οποίο αφαιρεί τμήματα του Κυβερνήτη τα οποία δεν επηρεάζουν την λειτουργία των εφαρμογών, έτσι ώστε να μπορεί να εκτελείται το ίδιο σε κάθε είδους μηχάνημα ή συσκευή IoT που βρίσκεται στα άκρα του νέφους (πχ Rasberry Pi) ανεξαρτήτως διαθεσιμότητας πόρων,οι οποίοι θα μπορούσαν να είναι ανεπαρκείς για τον Κυβερνήτη. Τέλος, ακολουθούν πειράματα που αξιολογούν τις συνθήκες που προτιμάται η μεταφόρτωση υπολογισμού από την λειτουργία της εφαρμογής απευθείας στη συσκευή και που αποδεικνύουν τους καλύτερους χρόνους ανταπόκρισης στις περιπτώσεις αυτές.

1.3 Οργάνωση του Τόμου

Η παρούσα διπλωματική εργασία χωρίζεται σε 6 κεφάλαια. Στο κεφάλαιο 2 παρουσιάζονται εργασίες και έρευνες που σχετίζονται με το παρόν θέμα και έχουν διεξάγει σχετικές υλοποιήσεις. Στο Κεφάλαιο 3 παρουσιάζεται η αρχιτεκτονική του Κυβερνήτη και τα τμήματά του που αξιοποιήθηκαν για την υλοποίηση αυτής της διπλωματικής, ενώ η ίδια η υλοποίηση παρουσιάζεται στο κεφάλαιο 4. Στο Κεφάλαιο 5 παρουσιάζονται τα πειραματικά αποτελέσματα και τέλος, στο Κεφάλαιο 6 αναλύουμε και παρουσιάζουμε τα συμπεράσματα της διπλωματικής, καθώς και προτάσεις για την μελλοντική εξέλιξη της ιδέας μας.

Κεφάλαιο 2

Υπάρχουσα Βιβλιογραφία

2.1 Τμηματοποίηση του Νέφους

Προκειμένου να αξιοποιηθεί βέλτιστα το νέφος, οι ειδικοί το έχουν τμηματοποιήσει ανάλογα με την τοποθεσία των κόμβων του και την εγγύτητα τους στους βασικούς πυρήνες του.Συγκεκριμένα, η γενικότερη κατηγορία που χρησιμοποιείται για το IoT είναι η υπολογιστική αρχιτεκτονική της ομίχλης-fog computing (FC) η οποία γεφυρώνει ουσιαστικά το κενό μεταξύ του νέφους και των τελικών συσκευών και περιλαμβάνει όλες τις διαδικασίες που σχετίζονται με τους υπολογισμούς,τη δικτύωση και τους αποθηκευτικούς χώρους των κοντινών ως προς τις IoT συσκευές κόμβων [16]. Το EC και το MEC που αναφέρθηκαν στις ενότητες 1.1.1 και 1.1.2 αντίστοιχα είναι υποκατηγορίες του FC οι οποίες έχουν να κάνουν ,όπως ήδη αναφέραμε, με τους πιο απομακρυσμένους κόμβους από τους βασικούς πυρήνες του νέφους. Άλλες σημαντικές κατηγορίες είναι οι εξής:

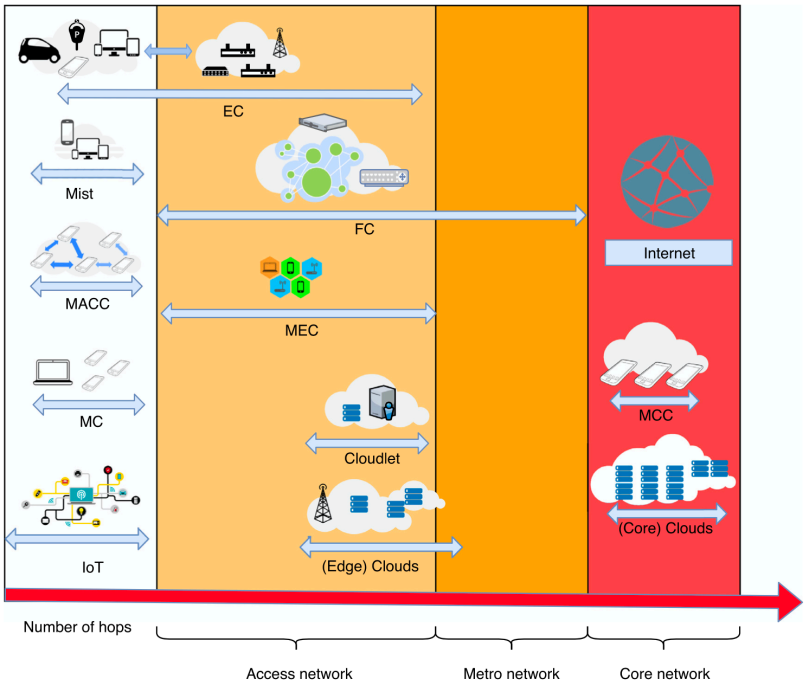
- **υπολογισμός κινητού νέφους-mobile cloud computing (MCC)**: Παρόμοιο με το EC, με τη διαφορά ότι το MCC περιλαμβάνει και τις ίδιες τις φορητές συσκευές οι οποίες μάλιστα μπορεί να βρίσκονται και πιο κοντά στον πυρήνα του νέφους,δίνοντας έτσι τη δυνατότητα για πολύπλοκους υπολογισμούς όχι μόνο σε χρήστες έξυπνων κινητών αλλά σε πολλών ειδών φορητούς χρήστες [23].

- **τοπικό υπολογιστικό νέφος-cloudlet**: Το ενδιάμεσο στάδιο μεταξύ του νέφους και των φορητών συσκευών, που αποτελείται από πολλούς υπολογιστές πολλαπλών πυρήνων οι οποίοι δρουν ως ένα μικρό νέφος [24].

- **υπολογιστική αρχιτεκτονική mist**: υπολογιστική νέφους η οποία συντελείται πάνω στις ίδιες τις φορητές συσκευές που ανήκουν στο νέφος και προτάθηκε ως είδος τελευταία με μελλοντικές βλέψεις σε αυτόνομα συστήματα φορητών συσκευών [25].

Παρατηρούμε ότι υπάρχουν λοιπόν λεπτές αλλά σημαντικές διαφορές μεταξύ των είδων, οι οποίες μάλιστα παρουσιάζονται καλύτερα στο σχήμα 2.

Με βάση λοιπόν αυτή την τμηματοποίηση έχουν διεξαχθεί πολλές έρευνες και πειράματα για να διερευνηθεί η καταλληλότητα του κάθε είδους όσον αφορά διάφορα κριτήρια, όπως η ανάγκη για υποδομές, οι εξαιρετικά μικροί χρόνοι απόκρισης, η γεωγραφική κατανομή των κόμβων, η υποστήριξη των εφαρμογών πραγματικού χρόνου, η υποστήριξη ετερογένειας και η δυνατότητα εικονοποίησης (virtualization). Τα αποτελέσματα αυτών φαίνονται συνοπτικά για το κάθε είδος στον πίνακα 1.



Σχήμα 2: Τμηματοποίηση νέφους με βάση την τοποθεσία και την απόσταση των κόμβων από τους βασικούς πυρήνες του [16]

Features of fog-computing related paradigms.

Feature	CC	MC	FC	EC	MCC	MACC	MEC	cC	mist
Heterogeneity support	√	✗	√	√	√	✗	✗	✗	√
Infrastructure need	√	✗	√	√	√	✗	√	√	√
Geographically distributed	✗	✗	√	√	✗	✗	√	√	√
Location awareness	✗	√	√	√	✗	√	√	√	√
Ultra-low latency	✗	✗	√	√	✗	✗	√	√	√
Mobility support	✗	√	√	√	√	√	√	√	√
Real-time application support	✗	✗	√	√	✗	✗	√	√	√
Large-scale application support	√	✗	√	√	✗	✗	√	✗	√
Standardized	√	√	√	√	✗	✗	√	✗	✗
Multiple IoT Applications	√	✗	√	✗	✗	✗	✗	√	√
Virtualization support	√	✗	√	✗	✗	✗	√	√	✗

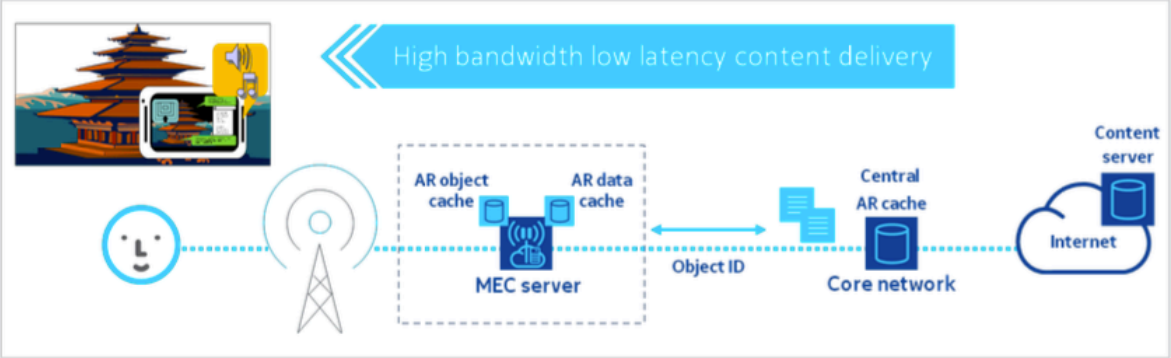
Πίνακας 1: Χαρακτηριστικά του κάθε είδους του νέφους [16]

2.2 Χρήση της υπολογιστικής αρχιτεκτονικής στα άκρα του δικτύου πολλαπλής πρόσβασης

Σύμφωνα με αυτές τις μελέτες παρατηρούμε ότι το MEC είναι συμβατό με τα περισσότερα κριτήρια,με εξαίρεση την υποστήριξη ετερογένειας και των πολλαπλών IoT εφαρμογών, θέτοντας το έτσι το πλέον κατάλληλο για χρήση της μεταφόρτωσης υπολογισμών σε αυτό.Αυτό προκύπτει από την αξιοποίηση του δικτύου ασύρματης πρόσβασης (RAN) το οποίο προσφέρει εγγύτητα και μεγάλο εύρος ζώνης [26]. Ήδη λοιπόν οι προγραμματιστές προσπαθούν να ενσωματώσουν τη χρήση του στις εφαρμογές τους, ώστε να αξιοποιήσουν πλήρως τα οφέλη που προσφέρει. Παρακάτω, παρατίθονται μερικές από τις πιο σημαντικές χρήσεις του MEC μέχρι στιγμής.

2.2.1 Επαυξημένη Πραγματικότητα

Με τον όρο επαυξημένη πραγματικότα-augmented reality (AR) εννοούμε τον συνδυασμό ενός πραγματικού περιβάλλοντος μαζί με συμπληρωματικά στοιχεία τα οποία έχουν παραχθεί σε υπολογιστή, όπως ήχοι,εικόνες, βίντεο και γραφικά. Επομένως, γίνεται αμέσως αντιληπτό ότι οι εφαρμογές επαυξημένης πραγματικότητας χρειάζονται μεγάλη υπολογιστική ισχύ και επιπλέον, επειδή αξιοποιούν πραγματικό περιβάλλον, είναι ευαίσθητες σε τυχόν καθυστερήσεις [27]. Ένα παράδειγμα χρήσης AR μαζί με MEC είναι η επίσκεψη σε ένα μουσείο, όπου ένας επισκέπτης μπορεί να χρησιμοποιήσει μία AR εφαρμογή για να βρίσκει επιπρόσθετα στοιχεία για τα εκθέματα τα οποία τον ενδιαφέρουν [26]. Η εφαρμογή πρέπει να γνωρίζει τη θέση του χρήστη και την κατεύθυνση που αντιμετωπίζει, είτε μέσω τεχνικών τοποθέτησης είτε μέσω της προβολής της κάμερας ή και των δύο. Μετά την ανάλυση αυτών των πληροφοριών, η εφαρμογή μπορεί να παρέχει επιπλέον πληροφορίες σε πραγματικό χρόνο στον χρήστη. Αν ο χρήστης μετακινηθεί, οι πληροφορίες πρέπει να ανανεωθούν. Αυτή η διαδικασία μπορεί να υλοποιηθεί με τη χρήση μιας MEC πλατφόρμας η οποία επιτρέπει στον χρήστη να λαμβάνει σε πραγματικό χρόνο τα δεδομένα,αφού η ανάλυση της πληροφορίας δε γίνεται πλέον στην κινητή συσκευή του, αλλά σε έναν πολύ κοντινό κόμβο του νέφους, όπως φαίνεται στην εικόνα 3.



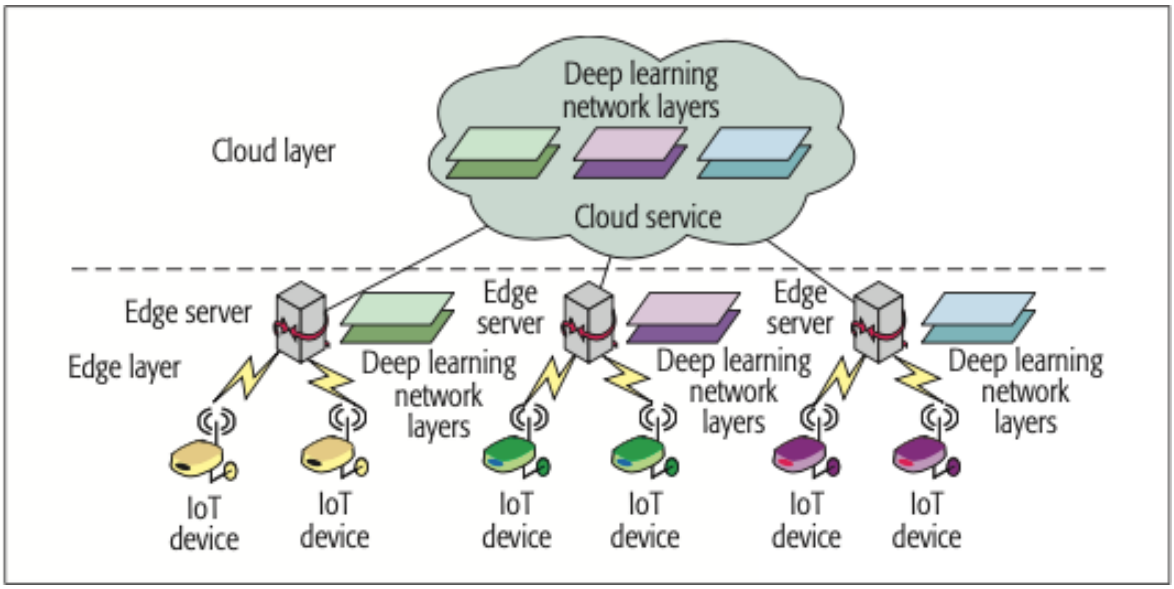
Σχήμα 3: Σενάριο χρήσης επαυξημένης πραγματικότητας [26]

2.2.2 Υπηρεσίες ροών δεδομένων και Παιχνίδια

Οι υπηρεσίες ροών δεδομένων-streaming services και τα παιχνίδια στους υπολογιστές είναι από τα πιο δημοφιλή και μεγαλύτερα σε κατανάλωση εύρους ζώνης μέσα στο Διαδίκτυο [28]. Υπολογίζεται μάλιστα πως τα βίντεο που παράγονται και προσφέρονται από τις υπηρεσίες streaming χρησιμοποιούν περίπου το 70 τα εκατό του διαθέσιμου εύρους ζώνης. Επιπλέον,τα παιχνίδια χρησιμοποιούν όλο και πιο περίπλοκα μέσα, όπως η επαυξημένη πραγματικότητα που ήδη αναφέραμε, και το ίδιο το Διαδίκτυο για την λειτουργία τους, δυσχεραίνοντας ακόμα περισσότερο την κατάσταση. Έτσι, με την παροχή και παραγωγή των βίντεο από πλατφόρμες MEC που βρίσκονται προφανώς κοντά στους χρήστες, καθώς και με τεχνικές όπως το παιχνίδι στο νέφος (Cloud gaming) όλες αυτές οι δυσκολίες περιορίζονται σημαντικά με αποτέλεσμα την καλύτερη ποιότητα εμπειρίας-quality of Experience των χρηστών.

2.2.3 Βαθειά Μάθηση για το Διαδίκτυο των Αντικειμένων

Η βαθειά μάθηση-deep learning γίνεται μία αναδυόμενη τεχνολογία για το IoT. Ειδικότερα, επειδή οι συσκευές IoT παράγουν μία πληθώρα δεδομένων για επεξεργασία, η βαθειά μάθηση προτιμάται σε σύγκριση με την μηχανική μάθηση καθώς έχει γενικά καλύτερη απόδοση όσον αφορά το μέγεθος των δεδομένων. Κατά την επεξεργασία πληροφοριών πολυμέσων μάλιστα, η απόδοση της παραδοσιακής μηχανικής μάθησης εξαρτάται από την ακρίβεια των χαρακτηριστικών που προσδιορίζονται και εξάγονται. Δεδομένου ότι μπορεί να μάθει με ακρίβεια χαρακτηριστικά υψηλού επιπέδου, όπως ανθρώπινα πρόσωπα σε εικόνες και γλώσσες σε φωνές, η βαθιά εκμάθηση μπορεί να βελτιώσει την αποτελεσματικότητα της επεξεργασίας πληροφοριών πολυμέσων [29]. Η χρήση του edge computing λοιπόν σε αυτή την περίπτωση είναι ιδανική, αφού το μεγαλύτερο μέρος των δεδομένων μπορεί να αναλυθεί στους απομακρυσμένους κόμβους, και τα αποτελέσματα αυτών μόνο να αναλυθούν στον πυρήνα του νέφους. Μία τέτοια διαδικασία,η οποία παρουσιάζεται στην εικόνα 4, έχει υλοποιηθεί στην [29], όπου τα πρώτα στρώματα της βαθειάς μάθησης έχουν διεξαχθεί στο edge και τα αποτελέσματα αυτών, τα οποία χρησιμοποιούνται στα επόμενα στρώματα,στέλνονται και επεξεργάζονται στον πυρήνα του νέφους. Έτσι, εξασφαλίζεται ασφάλεια,ταχύτητα και μικρή χρήση εύρους ζώνης για αυτό το πρόβλημα βαθειάς μάθησης.



Σχήμα 4: Χρήση edge computing για πρόβλημα μηχανικής μάθησης στο IoT [29]

2.3 Μέθοδος μετακίνησης των εφαρμογών

Όπως αναφέραμε και στην ενότητα 1.1.3, η χρήση της εικονοποίησης-virtualization βοηθάει σε μεγάλο βαθμό στην αξιοποίηση του MEC, είτε αυτή γίνεται με εικονικά μηχανήματα είτε με docker containers. Βέβαια, ακόμα και έτσι, αν οι χρήστες κινούνται και απομακρύνονται από τους κόμβους που συνδέθηκαν αρχικά, τότε οι χρόνοι απόκρισης μεγαλώνουν και πάλι, αναιρώντας έτσι τα πλεονεκτήματα που προσφέρει το MEC και μειώνοντας ταυτόχρονα την ποιότητα της υπηρεσίας-Quality of Service (QoS) [30]. Παρακάτω λοιπόν παρουσιάζουμε μερικές λύσεις που έχουν προταθεί για αυτό το θέμα,οι οποίες στοχεύουν στη μεταφορά του εικονικού μηχανήματος ή του container στον κάθε φορά πιο κοντινό κόμβο ως προς τον χρήστη, χωρίς βέβαια να επηρεάζεται η λειτουργία της εφαρμογής και χωρίς να έχουμε μεγάλους χρόνους μη απόκρισης-downtime.

2.3.1 Μεταφορά εικονικών μηχανημάτων

Όσον αφορά τα εικονικά μηχανήματα, πρέπει να λάβουμε υπόψην 3 είδη καταστάσεων που πρέπει να αντιμετωπιστούν προκειμένου να γίνει η μεταφορά:

1. Την κατάσταση της εικονικής μηχανής,συμπεριλαμβανομένης της κατάστασης της κεντρικής μονάδας επεξεργασίας, της μητρικής πλακέτας, των προσαρμογέων-adapter δικτύων και αποθηκευτικού χώρου καθώς και τους προσαρμογείς των γραφικών.

2. Τις εξωτερικά συνδεδεμένες συσκευές όπως συσκευές δικτύου, συσκευές USB και μεταθέσιμα μέσα όπως σκληροί δίσκοι.

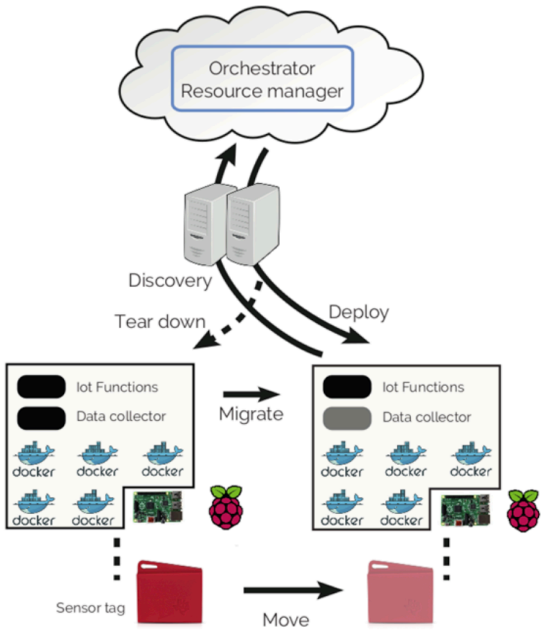
3. Την μνήμη του ίδιου του εικονικού μηχανήματος.

Έτσι, οι ερευνητές της [31] αντιμετώπισαν ως εξής τις καταστάσεις αυτές.Όσον αφορά το δίκτυο,όλες οι συνδέσεις που ήταν ανοιχτές πριν την μεταφορά πρέπει να παραμείνουν ανοιχτές και μετά από αυτήν. Χρησιμοποιείται λοιπόν μία κάρτα εικονικής διασύνδεσης δικτύου Ethernet-virtual Ethernet network interface card (VNIC),η οποία έχει δικιά της διεύθυνση MAC που την αναγνωρίζει μοναδικά στο τοπικό της δίκτυο. Επιπλέον, μπορεί να συνδέεται με μία ή περισσότερες πραγματικές κάρτες διασύνδεσης δικτύου, που διαχειρίνονται τον πυρήνα του εικονικού μηχανήματος-vmkernel. Επομένως, οι VNIC πολλών VMs μπορούν να συνδεθούν στην ίδια πραγματική κάρτα διασύνδεσης δικτύου και επειδή έχουν την δικιά τους MAC διεύθυνση,τα εικονικά μηχανήματα μπορούν να κρατούν τις συνδέσεις τους ανοιχτές όσο μετακινούνται,εφόσον βέβαια παραμένουν στο ίδιο υποδίκτυο. Όσον αφορά τη μνήμη του VM, οι ερευνητές την αντιγράφουν στο πραγματικό μηχάνημα όπου θα μεταφερθεί ενώ ταυτόχρονα το VM συνεχίζει να λειτουργεί στο αρχικό. Όταν η μεταφορά αυτή τελειώσει, μεταφέρονται τα κομμάτια μνήμης που τροποποιήθηκαν κατά τη διάρκεια της προηγούμενης μεταφοράς. Αυτή η διαδικασία συνεχίζεται μέχρι τα κομμάτια μνήμης που τροποποιήθηκαν να είναι πολύ μικρά και να επηρεάζουν ελάχιστα τη λειτουργία των εικονικών μηχανημάτων.
Τέλος, για τη λειτουργία των VMs χρησιμοποιούνται δίκτυα αποθήκευσης-storage area networks (SAN), έτσι ώστε όταν χρειαστεί να γίνει η μεταφορά, να συνδεθούν απλά οι δίσκοι του δικτύου αποθήκευσης που χρησιμοποιούνται στο αρχικό μηχάνημα, στο μηχάνημα που θα μεταφερθεί το VM. Με αυτή την μέθοδο, οι χρόνοι μη απόκρισης-downtime μειώνονται σε πολύ μεγάλο βαθμό και παράλληλα βελτιώνεται σημαντικά η απόδοση του VM, αφού βρίσκεται πολύ κοντά στον χρήστη και στην παραγωγή δεδομένων.

Μία παρόμοια τεχνική για τη μεταφορά της μνήμης έχει χρησιμοποιηθεί και στην [32], όπου η μνήμη υφίσταται μία συμπίεση πριν μεταφερθεί. Ειδικότερα, χρησιμοποιείται ένας αλγόριθμος, ο οποίος συμπιέζει τα δεδομένα ανάλογα με την συχνότητα εμφάνισης τους. Άρα, η μνήμη που πρέπει να μεταφερθεί είναι αισθητά μικρότερη σε μέγεθος, χωρίς όμως να επηρεάζεται σημαντικά η λειτουργία των VMs.

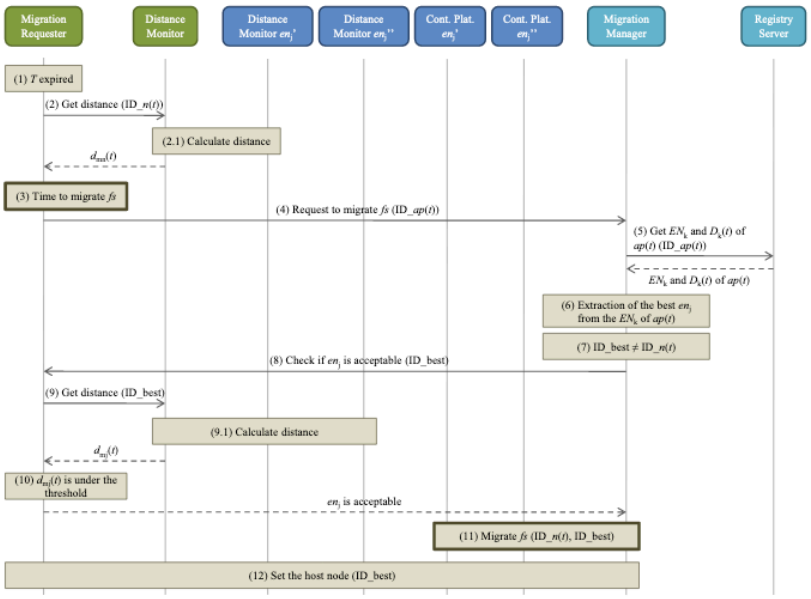
2.3.2 Μεταφορά των συσκευασμένων εφαρμογών

Όσον αφορά τα containers, με τα οποία θα ασχοληθούμε κυρίως στη διπλωματική αυτή, έχουν διεξαχθεί πολλές έρευνες για τη σωστή μεταφορά των εικόνων-εφαρμογών που περιέχονται σε αυτά. Μία από αυτές είναι η [33], η οποία προτείνει τη χρήση της πλατφόρμας Cloud4IoT [34] και του Κυβέρνητη [21] για τη μεταφορά του container. Πιο συγκεκριμένα, όταν η φορητή συσκευή συνδεθεί σε άλλον κόμβο του edge, η πλατφόρμα αυτή θα το εντοπίσει και θα στείλει σήμα στον συντονιστή του νέφους-cloud orchestrator. Αυτός, θα αλλάξει την συνάφεια κόμβου-node affinity [35], που ορίζει το που θα λειτουργεί το container, επιτρέποντας έτσι τη μεταφορά του container σε άλλο κόμβο.Η διαδικασία αυτή φαίνεται στην εικόνα 5, η οποία δυστυχώς λειτουργεί μόνο για stateless (δείτε 1.2) εφαρμογές.



Σχήμα 5: Χρήση cloud4iot πλατφόρμας για μεταφορά container [33]

Μία καλύτερη λύση προσφέρουν οι συγγραφείς της [36], όπου προτείνουν μέθοδο μεταφοράς που συμπεριλαμβάνει και stateful εφαρμογές και ονομάζεται συνοδευτική υπολογιστική νέφους-companion fog computing (CFC). Ειδικότερα, χρησιμοποιείται ένας αλγόριθμος, ο οποίος ανακαλύπτει τον πιο κοντινό κόμβο στην IoT συσκευή, ανάλογα με την πραγματική απόσταση από τη συσκευή ή την δικτυακή απόσταση από τον προηγούμενο κόμβο, και μεταφέρει σε αυτόν το container. Ο αλγόριθμος για την εύρεση του κόμβου παρουσιάζεται στην εικόνα 6.



Σχήμα 6: Αλγόριθμος εύρεσης κόμβου στο CFC [30]

Αφού βρεθεί ο κόμβος, γίνονται τα εξής βήματα για τη μεταφορά του container σε αυτόν.

1. Στον καινούριο κόμβο, κατεβαίνει από το Διαδίκτυο η εικόνα της εφαρμογής, αν δεν υπάρχει ήδη σε αυτόν.

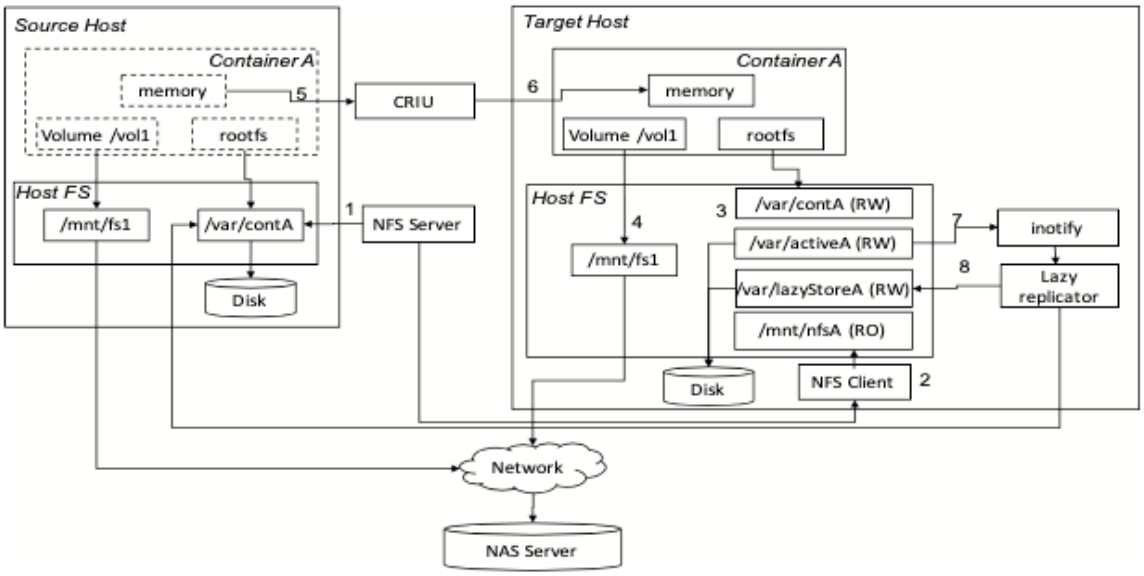
2. Στον προηγούμενο κόμβο, αποθηκεύεται η κατάσταση του container (checkpoint) και σταματά η λειτουργία του.

3. Η κατάσταση αυτή μεταφέρεται στον καινούριο κόμβο.

4. Το container συνεχίζει στον καινούριο κόμβο χρησιμοποιώντας την κατάσταση που μεταφέραμε στο προηγούμενο βήμα.

5. Τροποποιείται κατάλληλα η συσκευή IoT της οποίας τα δεδομένα χειρίζεται το container ώστε να επικοινωνεί με τον καινούριο κόμβο

Είναι σημαντικό να αναφέρουμε ότι στη μέθοδο αυτή δε χρησιμοποιείται ο Κυβερνήτης αλλά το ίδιο το docker [37]. Αυτή η διαφορά είναι σημαντική, καθώς το docker μπορεί να χρησιμοποιήσει το εργαλείο CRIU [38], που επιτρέπει την αποθήκευση της κατάστασης του container. Έτσι, γίνεται δυνατή η μεταφορά και stateful εφαρμογών μεταξύ των κόμβων του edge.
Η ιδέα αυτή βελτιστοποιείται ουσιαστικά στην [39], όπου προτείνεται το Voyager, ένα εργαλείο για πλήρη μεταφορά container. Το Voyager χρησιμοποιεί μία παρόμοια μέθοδο με αυτή που περιγράφηκε στην [31] για τα εικονικά μηχανήματα και φαίνεται στην εικόνα 7.Ειδικότερα, πρωτού αποθηκευτεί και μεταφερθεί η κατάσταση του container με το εργαλείο CRIU, μεταφέρεται ο βασικός κατάλογος-directory του container με χρήση συστήματος αρχείων δικτύου-network file system (NFS) από τον αρχικό κόμβο και φορτώνεται-mounted και στον τελικό κόμβο με δικαιώματα μόνο για διάβασμα. Εκεί,δημιουργούνται 2 καινούρια directories, ένα το οποίο ονομάζεται lazystore και θα αναλυθεί παρακάτω και ένα που ονομάζεται active. Στο δεύτερο αποθηκεύονται τυχόν αλλαγές που γίνονται σε αρχεία του container στον αρχικό κόμβο κατά τη διάρκεια αυτής της διαδικασίας και καινούρια αρχεία τα οποία μπορεί να δημιουργηθούν.



Σχήμα 7: Αλγόριθμος μεταφοράς container με το Voyager [39]

Όταν τελείωσει αυτή η διαδικασία και μεταφέρουμε την κατάσταση του container με το CRIU, το Voyager ξεκινά μία διαδικασίας αντιγραφής. Συγκεκριμένα, διαβάζει όλο το βασικό directory που βρίσκεται πλέον και στον τελικό κόμβο και τον αντιγράφει στο lazystore. Ταυτόχρονα, διαβάζει και το active directory ώστε να βρει τις αλλαγές που έχουν γίνει ήδη και να μη χρειαστεί να τις αντιγράψει και αυτές, μειώνοντας έτσι τα δεδομένα που θα αντιγραφούν και το χρόνο που θα χρειαστεί αυτή η διαδικασία. Με το τέλος αυτής, ό,τι αρχείο θα χρησιμοποιήσει τώρα το container θα το λάβει από το lazystore. Έτσι, η λειτουργία του container συνεχίζει από εκεί που ήταν στον πρώτο κόμβο με όλα τα αρχεία και τα δεδομένα που χρειάζεται.

2.4 Συνεισφορά της Διπλωματικής Εργασίας

Στη διπλωματική αυτή υλοποιούμε μία μέθοδο μεταφοράς container για stateful εφαρμογές στο πλέον χρησιμοποιούμενο εργαλείο διαχείρησης αυτών, τον Κυβερνήτη. Σε αντίθεση με τις μεθόδους που παρουσιάστηκαν παραπάνω λοιπόν, δε χρησιμοποιούμε το εργαλείο CRIU, καθώς αυτό δεν υποστηρίζεται ακόμα από τον Κυβερνήτη και επομένως, εφαρμόζουμε μία άλλη τεχνική για τη διατήρηση της κατάστασης του container. Επιπροσθέτως, εκμεταλλευόμαστε όσο το δυνατόν περισσότερο τις δυνατότητες που μας προσφέρει ο Κυβερνήτης για να υλοποιήσουμε την μεταφορά των αρχείων και των βάσεων δεδομένων που χρησιμοποιεί το container. Δε χρησιμοποιούμε δηλαδή εξωτερικά εργαλεία όπως το Voyager αλλά δημιουργούμε δικές μας εφαρμογές και εικόνες οι οποίες είναι υπεύθυνες για αυτή τη λειτουργία. Τέλος, αυτοματοποιούμε πλήρως τη διαδικασία της μεταφοράς, δηλαδή το σύστημα καταλαβαίνει μόνο του ποιος είναι ο επόμενος κόμβος όπου πρέπει να λειτουργήσει το container και υλοποιεί αμέσως τη μεταφορά. Όλα αυτά αναλύονται εκτενώς στο κεφάλαιο 4.

Κεφάλαιο 3

Κυβερνήτης

Στο παρόν κεφάλαιο παρουσιάζεται εκτενώς το εργαλείο που χρησιμοποιείται σε αυτή τη διπλωματική, ο Κυβερνήτης-Kubernetes [21]. Ειδικότερα αναλύεται η βασική ίδεα στην οποία στηρίζεται η λειτουργία του και κυρίως, επεξηγούνται σε βάθος τα βασικά χαρακτηριστικά και οι δυνατότητες που προσφέρει, τις οποίες χρησιμοποιούμε για την υλοποίηση του συστήματος μας και τη διεξαγωγή των πειραμάτων.

3.1 Βασική ιδέα του Κυβερνήτη

Ο Κυβερνήτης είναι ένα εργαλείο υπεύθυνο για την αυτόματη έναρξη, διαχείρηση και κλίμακωση των συσκευασμένων εφαρμογών σε container και των υπηρεσιών που προσφέρουν. Δημιουργήθηκε από την Google το 2015 και είναι ένα σύστημα που λειτουργεί ως πλατφόρμα σαν υπηρεσία-Platform as a Service (PaaS). Με τον όρο αυτό εννοούμε μία πλατφόρμα η οποία προσφέρει εξοπλισμό συστημάτων πληροφορικής-computer hardware καθώς και λογισμικό-software στους σχεδιαστές εφαρμογών. Επομένως, ο Κυβερνήτης οργανώνει τους αποθηκευτικούς χώρους, τη δικτύωση και τους υπολογιστικούς πόρους ώστε να επιτευχθεί η σωστή λειτουργία της συσκευασμένης εφαρμογής.
Η βασική ιδέα του Κυβερνήτη λοιπόν είναι να αποτρέπει τον σχεδιαστή εφαρμογών από το να αναλωθεί με θέματα που αφορούν το hardware και να ασχοληθεί αποκλειστικά και μόνο με την υλοποίηση του κώδικα της εφαρμογής και τη σωστή μετατροπή του σε container. Έτσι, όταν ο σχεδιαστής επιλέξει να χρησιμοποιήσει τον Κυβερνήτη, χρειάζεται απλά να καθορίσει την επιθυμητή κατάσταση για την εφαρμογή του και ο Κυβερνήτης θα είναι υπεύθυνος να χειριστεί το σύστημα και τις υποδομές του κατάλληλα ώστε να φτάσει σε αυτή την κάτασταση [40].

3.2 Αρχιτεκτονική Κυβερνήτη

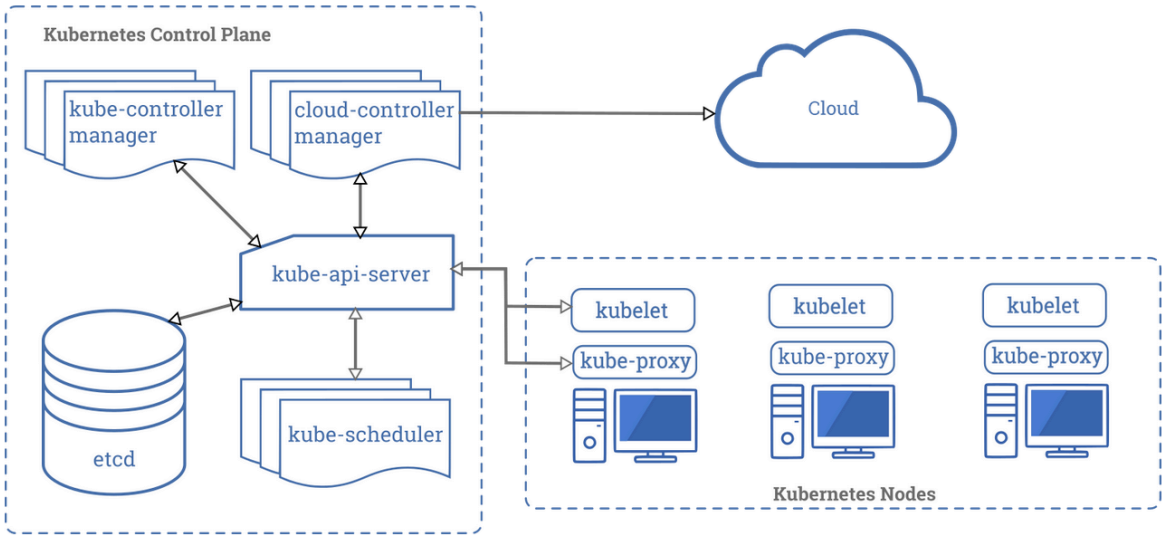
Η κατάσταση που αναφέραμε ορίζεται ως ένα σύνολο JSON αντικειμένων, τα οποία παρουσιάζονται σε ένα αρχείο yaml [41] και είναι αποθηκευμένα σε μία βάση δεδομένων η οποία ονομάζεται etcd [42]. Για να φέρει το container σε αυτή την κατάσταση, ο Κυβερνήτης χρησιμοποιεί συσκευές με τη λογική Αφέντη και Ακόλουθου-Master and Worker nodes, δημιουργώντας έτσι ένα σύστημα το οποίο ονομάζεται ομάδα-cluster. Το κάθε είδος συσκευής στον cluster έχει τα δικά του μέρη τα οποία συντελούν στο σκοπό συτό. Πιο συγκεκριμένα, ο Αφέντης-master έχει τα κύρια στοιχεία-master components, τα οποία είναι υπεύθυνα για την παρακολούθηση των αντικειμένων αυτών στη βάση δεδομένων, έτσι ώστε όταν παρατηρηθεί αλλαγή να προσπαθήσουν να φέρουν το container στην επιθυμητή κατάσταση. Τα στοιχεία αυτά επικοινωνούν με τους Ακολούθους-workers και τροποποιούν το σύστημα τους ανάλογα χρησιμοποιώντας ένα REST API, το οποίο χειρίζεται την βάση δεδομένων που είναι αποθηκευμένη αυτή η κατάσταση. Γίνεται λοιπόν αντιληπτό πως τα στοιχεία αυτά αποτελούν το επίπεδο ελέγχου του Κυβερνήτη. Έτσι, ο master είναι η συσκευή που ρυθμίζει όλο το σύστημα και διαχωρίζεται από τους workers, οι οποίοι είναι ουσιαστικά οι συσκευές που προσφέρουν τους πόρους για να εκτελεστούν τα containers. Βέβαια, και στον ίδιο τον master μπορούν να εκτελεστούν τα containers αλλά αυτή η τακτική δε συνηθίζεται. Τα στοιχεία αυτά, που παρουσιάζονται στην εικόνα 8, είναι οι εξής [43]:

- **kube-apiserver**: Εκθέτει το API του Κυβερνήτη και λειτουργεί ουσιαστικά ως το εμπρόσθιο άκρο-frontend του επιπέδου ελέγχου του Κυβερνήτη. Οποιοσδήποτε θέλει να έχει πρόσβαση στην κατάσταση του Κυβερνήτη πρέπει να επικοινωνήσει με αυτό το στοιχείο. Μπορεί να γίνει πιο ανθεκτικό σε σφάλματα και να μπορεί να χειρίζεται περισσότερους χρήστες αν το τρέχουμε σε περισσότερους από έναν κόμβους.

- **etcd**: Μία κατανεμημένη βάση δεδομένων, υψηλής αντοχής σε σφάλματα και με μεγάλη διαθεσιμότητα, για να αποθηκεύει την κατάσταση του Κυβερνήτη.

- **kube-scheduler**: Το στοιχείο του επιπέδου ελέγχου το οποίο αναμένει τη δημιουργία καινούριων καψουλών-pod (αναλύεται στην ενότητα 3.3.1) και αναζητά τον καλύτερο δυνατό worker για να τις τοποθετήσει. Οι παράγοντες που λαμβάνονται υπόψιν στην αναζήτηση καλύτερου κόμβου αποτελούν οι πόροι που απαιτούνται προς κατανάλωση, οι περιορισμοί υλικού ή λογισμικού, οι προτιμήσεις σε πόρους, η τοπολογία των δεδομένων καθώς και η κατάσταση των ίδιων των κόμβων [44].

- **kube-controller-manager**: Είναι το τμήμα του επιπέδου ελέγχου που συντονίζει και τρέχει τους χειριστές-controllers του Κυβερνήτη για τα διάφορα αντικείμενα που υπάρχουν.



Σχήμα 8: Αρχιτεκτονική master-worker Κυβερνήτη [45]

Όσον αφορα τους workers , έχουν τας εξής στοιχεία τα οποία είναι υπεύθυνα για τη σωστή λειτουργία των εφαρμογών.

- **kubelet**: Ένα μέσο το οποίο τρέχει σε κάθε worker και είναι υπεύθυνος για τη λειτουργία των καψουλών-pod στον κόμβο αυτό, που αποτελούν τις πιο βασικές μονάδες ενθυλάκωσης των container [46]. Αυτό το μέσο ελέγχει μόνο τα pods τα οποία είναι κατασκευασμένα από τον κυβερνήτη. Χρησιμοποιεί σύνολα χαρακτηριστικών των καψουλών-PodSpecs τα οποία λαμβάνει με διάφορους μηχανισμούς και διαβεβαιώνει ότι τα container που περιγράφονται σε αυτά τρέχουν σωστά με βάση τις προδιαγραφές αυτές.

- **kube-proxy**: Λειτουργεί ως ένα πληρεξούσιο του δικτύου που λειτουργεί σε κάθε κόμβο και συμβάλλει στην σωστή λειτουργία των Υπηρεσιών-Services (αναλύεται στην ενότητα 3.5.3). Επιπλέον, παρέχει κανόνες δικτύωσης για τους κόμβους. Το kube-proxy βοηθάει επομένως

στην επικοινωνία των pods τόσο μεταξύ τους όσο και με χρήστες εκτός του Κυβερνήτη. Χρησιμοποιεί το επίπεδο φιλτραρίσματος των πακέτων που έχει το λειτουργικό σύστημα του worker, αν αυτό είναι διαθέσιμο, αλλιώς γίνεται το ίδιο υπεύθυνο για την κίνηση και μεταφορά των πακέτων

- **container-runtime**: Είναι το λογισμικό το οποίο δημιουργεί τα container στους worker nodes. Ο Κυβερνήτης υποστηρίζει αρκετά λογισμικά αυτού του είδους, όπως τα Docker, containerd [47], cri-o [48] και rktlet [49].

3.3 Αντικείμενα και Χειριστές

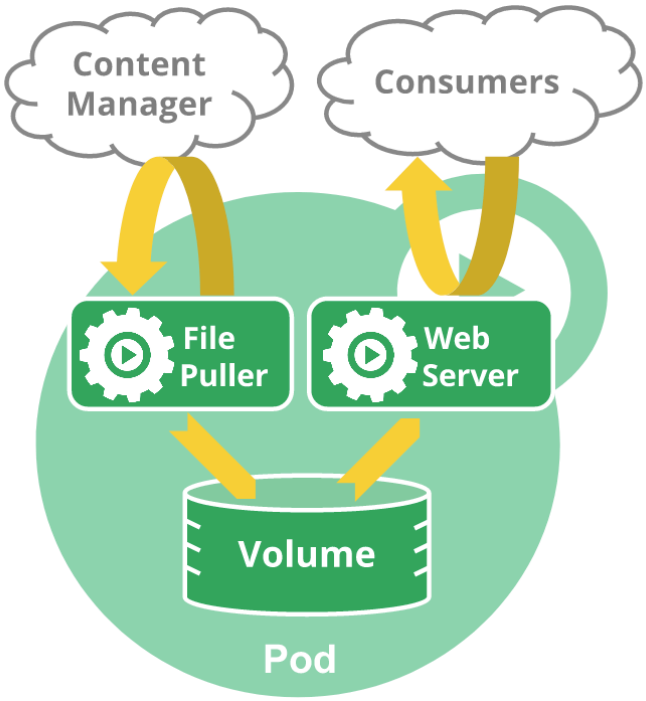
Ο Κυβερνήτης παρέχει ένα σύνολο από αφαιρέσεις-abstractions οι οποίες χρησιμοποιούνται για να αναπαραστήσουν την κατάσταση του συστήματος, η οποία αποτελείται από τις συσκευασμένες εφαρμογές, τους πόρους που αξιοποιούνται καθώς και το σύνολο των JSON αντικειμένων. Όλα αυτά αναπαρίστανται από τα αντικείμενα του Κυβερνήτη που προσφέρει το kubernetes-api. Τα αντικείμενα του Κυβερνήτη είναι μία δήλωση που παρουσιάζουν μία επιθυμητή κατάσταση, την οποία το επίπεδο ελέγχου του Κυβερνήτη προσπαθεί συνεχώς να μετατρέψει σε πραγματική κατάσταση. Ένα αντικείμενο δηλαδή είναι το μέσο με το οποίο ο Κυβερνήτης προσπαθεί να εκτελέσει την βασική ιδέα που αναφέραμε παραπάνω και έχει την εξής συγκεκριμένη δομή [50]:

- **Kind**: Καθορίζει το είδος του αντικειμένου.
- **api-version**: Καθορίζει την εκδοχή του Api όπου περιγράφεται αυτό το αντικείμενο.
- **Metadata**: Δεδομένα που βοηθούν στην αναγνώριση του αντικειμένου.
- **Spec**: Η επιθυμητή κατάσταση του Κυβερνήτη, που περιγράφεται από πολλά εμφωλευμένα πεδία, διαφορετικά για το κάθε αντικείμενο όπως πόροι και αποθηκευτικοί χώροι.
- **Status**: Η πραγματική κατάσταση του αντικειμένου, στην οποία έχει πρόσβαση μόνο ο Κυβερνήτης.

Για τον χειρισμό των αντικειμένων και τη σωστή λειτουργία τους, ο Κυβερνήτης χρησιμοποιεί τους χειριστές-controllers. Πρόκειται για ένα λογισμικό που περιλαμβάνει προγράμματα, τα οποία τρέχουν διαρκώς και παρατηρούν το Api του Κυβερνήτη για τα αντικείμενα τα οποία είναι υπεύθυνοι.

3.3.1 Κάψουλες

Η μικρότερη και πιο βασική μονάδα ενθυλάκωσης των container που μπορεί να δημιουργηθεί από τον Κυβερνήτη ονομάζεται κάψουλα-pod [46]. Ειδικότερα, το pod είναι ένα σύνολο από ένα η περισσότερα container με κοινούς αποθηκευτικούς χώρους και δίκτυο και περιλαμβάνει κατάλληλες οδηγίες για τη λειτουργία αυτών. Μία καλή αναλογία για τη σχέση μεταξύ pod και container είναι τα VMs και οι διεργασίες που αυτές τρέχουν. Εφαρμογές που θα έτρεχαν σε κοινό μηχάνημα όπως ένα VM και θα επικοινωνούσαν μεταξύ τους μέσω αυτού τώρα τρέχουν ως διαφορετικά containers ενθυλακωμένα σε ένα κοινό pod. Αυτές οι συσκευασμένες εφαρμογές στο κοινό pod λοιπόν έχουν τη δυνατότητα να επικοινωνούν μεταξύ τους εύκολα χωρίς να χρειάζεται κάποια περαιτέρω ρύθμιση για το δίκτυο (θα αναλυθεί πλήρως στην ενότητα 3.5) και επιπλέον μπορούν να μοιράζονται και αποθηκευτικούς χώρους. Ένα καλό παράδειγμα για αυτό είναι η εφαρμογή που απεικονίζεται στην εικόνα 9. Πιο συγκεκριμένα, οι χρήστες συνδέονται στην εφαρμογή με τη χρήση ενός frontend εξυπηρετητή-server και ζητάνε συγκεκριμένα αρχεία. Έτσι, ο server επικοινωνεί με μία άλλη εφαρμογή υπεύθυνη για τη διαχείρηση αρχείων, με την οποία μοιράζεται μία βάση δεδομένων. Ο server και αυτή η εφαρμογή είναι λοιπόν μαζί συσκεασμένοι στο ίδιο pod,το οποίο χρησιμοποιεί έναν όγκο-volume (αναλύεται στην ενότητα 3.4) για αποθηκευτικό χώρο.



Σχήμα 9: Παράδειγμα ενός pod [46]

Τα pods θεωρούνται εφήμερες και όχι ανθεκτικές οντότητες. Ειδικότερα, δημιουργούνται, τους ανατίθεται μία μονάδη ταυτότητα-ID και προγραμματίζονται να τρέξουν σε συγκεκριμένους κόμβους στον cluster με βάση τον scheduler, όπου παραμένουν μέχρι την ολοκλήρωση τους, τον τερματισμό τους ή μέχρι να τους μεταφέρει σε άλλο κόμβο ο scheduler. Αν ένας κόμβος τερματιστεί, τα pods που εκτελούνται σε αυτόν θα μεταφερθούν σε άλλον κατάλληλο κόμβο (αν υπάρχει βέβαια). Με τον όρο μεταφέρεται εννοούμε ουσιαστικά ότι το αρχικό pod καταστρέφεται και φτιάχνεται ένα πανομοιότυπο pod με τα ίδια χαρακτηριστικά (και αν επιθυμούμε και με το ίδιο όνομα) αλλά με διαφορετική ID στον καινούριο κόμβο. Είναι σημαντικό να αναφέρουμε ότι οτιδήποτε έχει τον ίδιο χρόνο ζωής με το pod, όπως ένας volume, καταστρέφεται μαζί με το pod και επομένως, σε περίπτωση μεταφοράς του pod δημιουργούνται καινούρια στη θέση τους.

3.3.2 Αντικείμενα διαχείρησης καψουλών

Για τη σωστή διαχείρηση των pods, ο Κυβερνήτης προσφέρει αρκετά αντικείμενα, τα πιο σημαντικά εκ των οποίων είναι τα Deployments και τα DaemonSets. Όσον αφορά τα Deployments [51], είναι το βασικό αντικείμενο με το οποίο λειτουργεί ο Κυβερνήτης. Μέσω αυτού ένας σχεδιαστής εφαρμογών-διαχειριστής του συστήματος του Κυβερνήτη μπορεί να ορίσει πλήρως την κατάσταση που επιθυμεί να φτάσει το σύστημα του. Σε ένα deployment παραδείγματος χάριν, μπορούν να οριστούν πόσα αντίγραφα του pod χρειάζονται με τη χρήση του Replica Set [52], να περάσουμε πληροφορίες από το ίδιο το pod μέσα στα container με τη χρήση μεταβλητών περιβάλλοντος [53], να ορίσουμε τι είδους αποθηκευτικοί χώροι θα χρησιμοποιηθούν, μέχρι και να διευκρινίσουμε πλήρως τους υπολογιστικούς πόρους, όπως τη μνήμη, που θα χρησιμοποιήσουν τα container [54]. Έτσι, με τη χρήση του χειριστή του deployment, τον deployment controller, ο Κυβερνήτης θα προσπαθήσει να φέρει την πραγματική κατάσταση του συστήματος στην επιθυμητή αυτή κατάσταση.
Τα daemonsets [55] είναι παρόμοια με τα deployments, με τη διαφορά ότι με τη χρήση τους δημιουργείται ένα pod σε κάθε κόμβο που ανήκει στον cluster, εκτός βέβαια αν θέλουμε εμείς να εξαιρέσουμε κάποιους από αυτούς. Όλες αυτές τις δυνατότητες τις αξιοποιήσουμε κατάλληλα για τη σωστή λειτουργία του συστήματος μας.

3.4 Αποθηκευτικοί χώροι

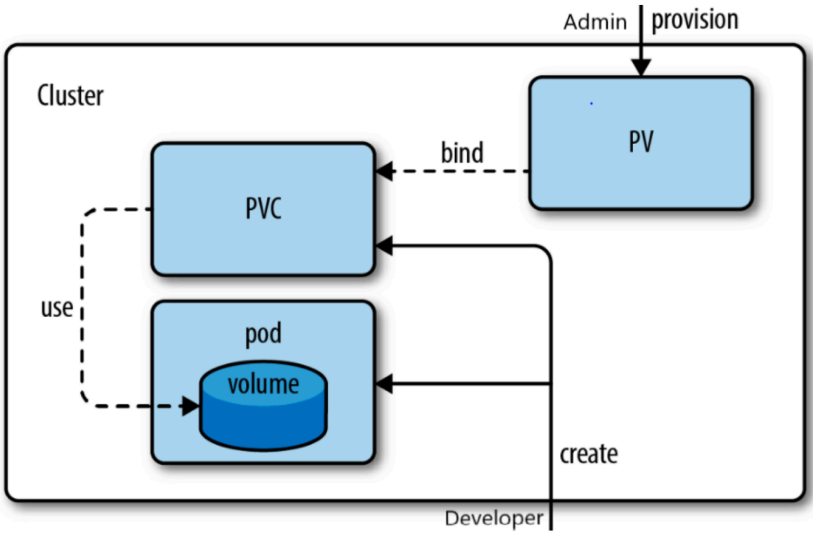
Οι δίσκοι που έχουν τα container για αποθηκευτικούς χώρους έχουν τον ίδιο χρόνο ζωής με το container, γεγονός που προκαλεί προβλήματα για περίπλοκες εφαρμογές. Αρχικά, σε περίπτωση τερματισμού του container λόγω κάποιας δυσλειτουργίας του, το kubelet θα το επανακκινήσει αλλά όλα τα αρχεία που είχε θα έχουν καταστραφεί. Επιπλέον, τα containers τα οποία συνυπάρχουν σε ένα pod χρειάζεται πολλές φορές να μοιράζονται αρχεία. Αυτά τα δύο προβλήματα ο Κυβερνήτης τα αντιμετωπίζει με τη χρήση των όγκων-volumes [56].
Οι volumes έχουν μεγαλύτερη διάρκεια από τα container και συγκεκριμένα, έχουν τον ίδιο χρόνο ζωής με τα pods. Έτσι, σε περίπτωση τερματισμού των container και επανακίννησης τους, τα αρχεία τους που υπάρχουν σε αυτούς διατηρούνται και μπορούν να επαναχρησιμοποιηθούν. Βέβαια, σε περίπτωση τερματισμού των pods οι volumes παύουν να υπάρχουν και τα αρχεία αυτά καταστρέφονται.Ουσιαστικά λοιπόν δεν είναι τίποτα παραπάνω από directories των pods, τα οποία φορτώνονται μέσω αυτών στα container για χρήση. Το πως χρησιμοποιούνται, το πως φορτώνονται και τα περιεχόμενα τους εξαρτώνται από το είδους του volume.
Προκειμένου ένα pod να χρησιμοποιήσει έναν volume, πρέπει να διευκρινήσει στα χαρακτηριστικά του, δηλαδή στην επιθυμητή κατάσταση, το είδος του καθώς και σε ποιο μονοπάτι-path του container θα φορτωθεί. Ειδικότερα, η εικόνα του container βρίσκεται στην ρίζα του συστήματος αρχείων του container, και ο volume φορτώνεται σε συγκεκριμένο μονοπάτι αυτής της εικόνας για χρήση.

3.4.1 Επίμονοι όγκοι

Πολλές φορές χρειάζεται να διατηρήσουμε αρχεία και δεδομένα και μετά τον τερματισμό των pods, είτε για χρήση άλλης εφαρμογής είτε για να κάνουμε μετρήσεις για τη λειτουργία της εφαρμογής που χρησιμοποιούσε τα δεδομένα αυτά. Για αυτό το σκοπό ο κυβερνήτης προσφέρει τους επίμονους όγκους-persistent volumes (PV), ένα αντικείμενο για αποθήκευση δεδομένων το οποίο διατηρείται και υπάρχει και μετά το τέλος των pods [57]. Οι PVs παρέχονται από τους διαχειριστές του συστήματος είτε χειροκίνητα-manually είτε δυναμικά με χρήση άλλων αντικειμένων που προσφέρει ο κυβερνήτης, των αποθηκευτικών κλάσεων-storage classes [58]. Ανάλογα με τον πάροχο νέφους-cloud provider, υπάρχουν πολλά είδη, το καθένα ειδικό για διαφορετικές λειτουργίες. Επιπλέον, μπορούμε να χρησιμοποιήσουμε directories των ίδιων των worker nodes ως PVs αλλά αυτή η τεχνική χρησιμοποιείται μόνο για DaemonSets. Τέλος, η πραγματική χρησιμότητα των PVs έγκειται στην δυνατότητα τους να αξιοποιηθούν από ένα ή και περισσότερα είδη pods. Ειδικότερα, υπάρχουν 3 είδη πρόσβασης στα PVs, το ReadWriteOnce (RWO), που επιτρέπει την ανάγνωση και το γράψιμο από ένα μόνο pod, το ReadOnlyMany (ROM), που επιτρέπει μόνο την ανάγνωση από πολλά διαφορετικά pods και τέλος, το ReadWriteMany (RWM), το πιο σημαντικό από όλα καθώς επιτρέπει τόσο την ανάγνωση όσο και το γράψιμο για πολλά είδη pods.

3.4.2 Αιτήσεις επίμονων όγκων

Για να αξιοποιηθούν οι PVs, οι χρήστες του Κυβερνήτη χρησιμοποιούν τις αιτήσεις επίμονων όγκων-Persistent Volume Claims (PVC), οι οποίες είναι αιτήσεις για χρήσης αποθηκευτικών χώρων. Πιο συγκεκριμένα, όπως τα pods καταναλώνουν υπολογιστικούς πόρους από κόμβους, έτσι και οι PVCs καταναλώνουν PVs. Επίσης, οι αιτήσεις αυτές μπορούν να ζητήσουν συγκεκριμένη ποσότητα αποθηκευτικού χώρου και συγκεκριμένο είδος πρόσβασης. Ο Κυβερνήτης λοιπόν είναι υπεύθυνος να βρει έναν PV ο οποίος ανήκει στο είδος που απαιτεί η PVC και καλύπτει τα προαναφερθέντα κριτήρια που απαιτεί. Είναι σημαντικό να τονιστεί πως ενώ πολλά pods μπορούν να δείχνουν σε μία PVC, αυτή θα συνδέεται αποκλειστεικά και μόνο με έναν PV. Υπάρχει λοιπόν μία ζεύξη μεταξύ PVC και PV, η οποία ισχύει μέχρι να καταστραφούν και τα 2 αντικείμενα αυτά. Αυτή η σύνδεση των δύο και η χρήση τους σε pod φαίνεται στην εικόνα 10



Σχήμα 10: Χρήση PV και PVC

3.5 Δικτύωση

Το δίκτυο του Κυβερνήτη [59] υλοποιείται με τέτοιο τρόπο ώστε σε κάθε cluster να ισχύουν οι εξής κανόνες:

- Κάθε container να μπορεί να επικοινωνήσει με τα υπόλοιπα στο ίδιο pod.
- Κάθε pod σε έναν κόμβο να μπορεί να επικοινωνήσει με όλα τα pods σε κάθε κόμβο χωρίς τη χρήση του πρωτοκόλλου της μετάφρασης διευθύνσεων δικτύου-Network Address Translation (NAT).
- Κάθε παράγοντας του Κυβερνήτη (πχ kubelet) να μπορεί να επικοινωνήσει με όλα τα pods σε αυτόν τον κόμβο.

Για την επίτευξη αυτών των στόχων πρέπει λοιπόν να αντιμετωπιστούν τα εξής 3 προβλήματα:

1. Η επικοινωνία των container μεταξύ τους μέσα σε ένα pod.
2. Η επικοινωνία των διαφορετικών κόμβων μεταξύ τους.
3. Η επικοινωνία μεταξύ των pod που μπορεί να βρίσκονται σε διαφορετικούς κόμβους.

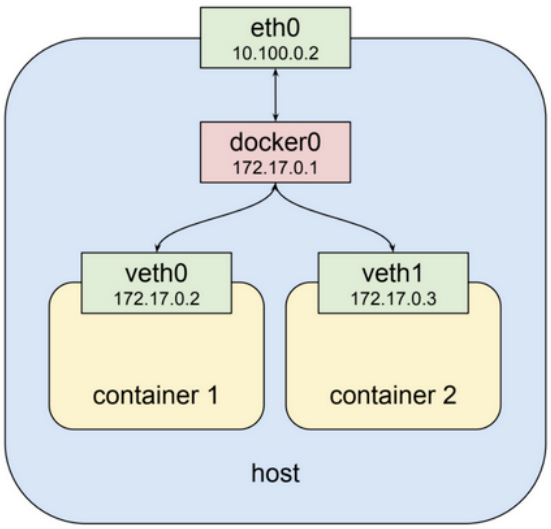
Παρακάτω αναλύονται πλήρως οι τρόποι επίλυσης αυτών των προβλημάτων.

3.5.1 Επικοινωνία μέσα στην ίδια κάψουλα

Όπως αναφέραμε και στην ενότητα 3.3.1, ένα pod μπορεί να περιέχει ένα η περισσότερα container, τα οποία συνήθως θα είναι στενά συνδεδεμένα όσον αφορά τη λειτουργία μιας εφαρμογής. Επομένως, η ανάγκη για την μεταξύ τους επικοινωνία είναι επιτακτική. Για να επιτευχθεί αυτό, κάθε pod έχει και διαφορετική IP διεύθυνση, την οποία μοιράζονται μεταξύ τους τα container που περιέχονται σε αυτά. Επιπλέον, μοιράζονται και το ίδιο διάστημα πυλών-portspace καθώς και τον ίδιο χώρο ονομάτων δικτύου-network namespace. Έτσι, μπορούν να επικοινωνήσουν μεταξύ τους μέσω του localhost με τη χρήση διαφορετικών πυλών. Δίνεται επίσης η δυνατότητα επικοινωνίας μεταξύ τους με τη χρήση κλασσικών μεθόδων επικοινωνίας διεργασιών όπως η χρήση σεμαφόρων.

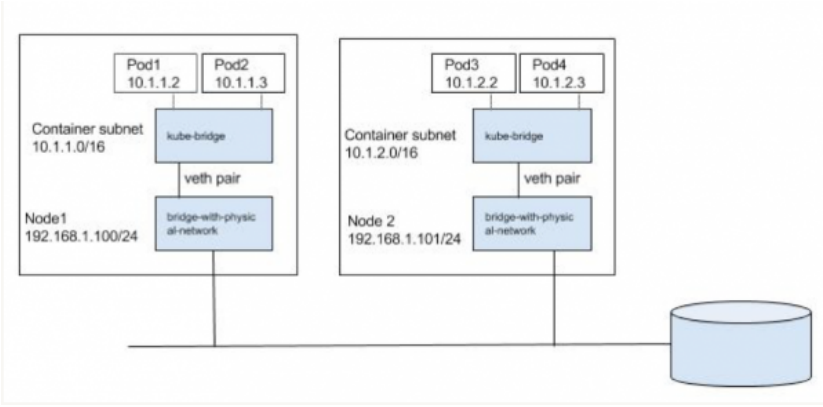
3.5.2 Επικοινωνία μεταξύ κόμβων

Παρ΄όλα αυτά, η υπερφόρτωση ενός pod με πολλά και διαφορετικά container δεν συμβάλλει στη δημιουργία ενός σωστού συστήματος. Ως επί το πλείστον, οι διαχειριστές του Κυβερνήτη χρησιμοποιούν διαφορετικά είδη Pod για τα συστήματα τους, τα οποία μάλιστα δεν είναι αναγκαστικό να εκτελούνται στον ίδιο κόμβο, όπου η επικοινωνία μεταξύ τους θα μπορούσε να λυθεί με τη χρήση μίας απλής γέφυρας-bridge, όπως φαίνεται στην εικόνα 11.



Σχήμα 11: Χρήση γέφυρας docker0 για την επικοινωνία 2 διαφορετικών pod στον ίδιο κομβο

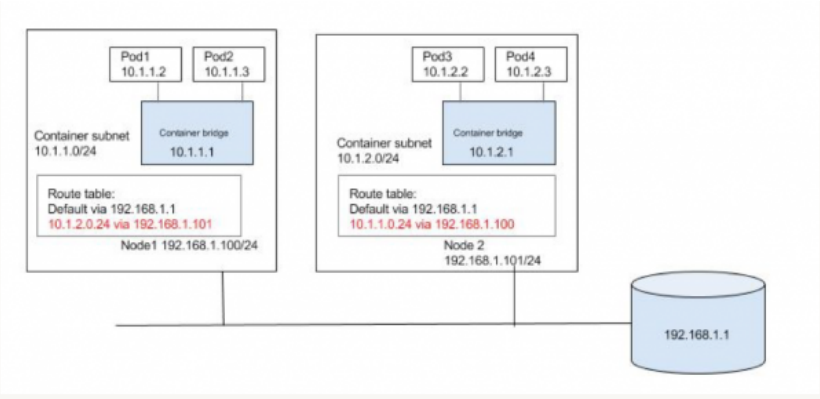
Επομένως, οι διαχειριστές καλούνται να λύσουν το πρόβλημα της επικοινωνίας μεταξύ των διαφορετικών κόμβων στον ίδιο cluster. Ο Κυβερνήτης προσφέρει 3 διαφορετικές επιλογές στους διαχειριστές για τη λύση του ζητήματος αυτού.
Αρχικά, η πιο απλή λύση είναι η χρήση του επιπέδου 2-layer 2 solution. Τα pods και οι κόμβοι μπορούν να δουν το υποδίκτυο που χρησιμοποιείται για τις IP των pods ως ένα τομέα επιπέδου 2. Έτσι, η επικοινωνία μεταξύ των pods, που βρίσκονται είτε στον ίδιο είτε σε διαφορετικούς κόμβους μπορεί να γίνει με τη χρήση του πρωτοκόλλου επίλυσης διευθύνσεων-address resolution protocol (ARP) και τη χρήση μεταγωγέων,όπως φαίνεται στην εικόνα 12.



Σχήμα 12: Λύση με τη χρήση μεταγωγέων και ARP

Μία καλύτερη λύση, η οποία μπορεί να χρησιμοποιηθεί πιο εύκολα για την κλιμάκωση του συστήματος είναι η χρήση του επιπέδου 3-layer 3 solution. Ειδικότερα, χρησιμοποιούμε τους κανόνες δρομολόγησης των κόμβων, αντί να χρησιμοποιούμε μεταγωγείς που συνδέονται απευθείας με τα

pods. Έτσι, πακέτα που προορίζονται για τα υποδίκτυα των κόμβων που ανήκουν στα pods δρομολογούνται μέσω των αντίστοιχων κόμβων. Ένα καλό παράδειγμα απεικονίζεται στην εικόνα 13.



Σχήμα 13: Λύση με τη χρήση κανόνων δρομολόγησης

Τέλος, μπορούν να χρησιμοποιηθούν δίκτυα επικαλλύματος-overlay networks, τα οποία χρησιμοποιούν σήραγγες-tunnels και ενθυλάκωση για τη μετακίνηση των πακέτων. Ειδικότερα, ενθυλακώνουν τα πακέτα από τον αρχικό κόμβο και με τη χρήση σηράγγων-tunnel που έχουν κατασκευάσει τα ίδια, μεταφέρουν τα ενθυλακωμένα πακέτα στον προορισμό τους, όπου λαμβάνουν την αρχική τους μορφή. Το εργαλείο που χρησιμοποιούμε εμείς στη διπλωματική, το k3s [22], χρησιμοποιεί για την επικοινωνία των κόμβων ένα τέτοιο δίκτυο, το flannel [60].

3.5.3 Υπηρεσίες

Με έναν από αυτούς τους τρόπους λοιπόν οι διαχειριστές επιλύουν το πρόβλημα της επικοινωνίας μεταξύ των κόμβων. Βέβαια, αυτό δεν συνεπάγεται αυτόματα και τη σωστή επικοινωνία των pods στους κόμβους αυτούς. Πιο συγκεκριμένα, όπως αναφέραμε τα pods δεν είναι ανθεκτικές οντότητες. Με τη χρήση των Deployments δημιουργούνται και καταστρέφονται δυναμικά και κάθε φορά παίρνουν και από μία IP, η οποία κατά πάσα πιθανότητα θα είναι διαφορετική από την προηγούμενη που είχαν πριν καταστραφούν. Επίσης, μπορεί να βρίσκονται και σε διαφορετικό κόμβο σε περίπτωση μεταφοράς τους. Επομένως, αν αυτά τα pods προσφέρουν λειτουργίες (ας τα ονομάσουμε backend-pods) σε pods με τα οποία επικοινωνούν χρήστες (ας τα ονομάσουμε frontend-pods), πρέπει τα frontend-pods να μπορούν να συνεχίσουν να συνδέονται με τα backend-pods, ανεξαρτήτως του αν άλλαξαν IP ή όχι. Για αυτό το σκοπό, ο Κυβερνήτης προσφέρει τις Υπηρεσίες-Services [61].
Οι Services, είναι αφαιρέσεις που καθορίζουν ένα λογικό σύνολο από Pods και μία πολιτική με την οποία έχουμε πρόσβαση σε αυτά. Ειδικότερα, μία Service χρησιμοποιεί έναν επιλογέα-selector, με τον οποίο περιλαμβάνει όλα τα pods τα οποία έχουν στα χαρακτηριστικά τους τον ίδιο επιλογέα. Έτσι, όταν ένας χρήστης ή ένα άλλο pod θέλει να έχει πρόσβαση σε αυτά τα pods, επικοινωνεί με την Service, η οποία γνωρίζει τις IP των pod που περιλαμβάνει και τις ανανεώνει σε περίπτωση που αυτές αλλάξουν, και όχι με τα ίδια τα pods. Το πακέτο λοιπόν που στέλνεται έχει ως στόχο την Service και αυτή επιλέγει με τη χρήση ενός αλγορίθμου, συνήθως του Round-Robin, μία από τις IP των pod που περιλαμβάνει για να στείλει το πακέτο.
Υπάρχουν 3 βασικά είδη services που καθορίζουν την πρόσβαση στα pods τα οποία είναι η clusterip service,η nodeport service και η loadbalancer service.Εμείς εδώ θα αναλύσουμε μόνο τα 2 πρώτα μιας και είναι αυτά που χρησιμοποιούμε στην υλοποίησή μας.

- **clusterip**: Είναι το βασικό είδος service που υπάρχει στον Κυβερνήτη. Υλοποιεί τη λογική που περιγράφθηκε παραπάνω και χρησιμοποιείται μόνο για την επικοινωνία των pods μέσα στον cluster. Ο διαχειριστής ορίζει ένα port, το οποίο είναι η πύλη-port που ακούει το container, και ένα targetPort. Έτσι, αντί να επικοινωνούν με τις IP των pods στην πύλη port, επικοινωνούν με την cluster ip service στο targetPort και αυτή μεταφέρει το πακέτο σε ένα από τα pods που ακούνε στην πύλη port. Για λόγους ευκολίας, συνήθως το port και targetPort ταυτίζονται.

- **nodeport**: Επιτρέπει την επικοινωνία των pods με κόμβους και χρήστες εκτός του cluster. Ειδικότερα, μαζί με το port και το targetport ο διαχειριστής του συστήματος ορίζει και το nodePort. Με αυτόν τον τρόπο, ένας χρήστης μπορεί να συνδεθεί στην IP ενός **κόμβου** στην πύλη nodeport και η service μεταφέρει όπως πριν το πακέτο στην ip:<port> ενός από τα pod τα οποία περιλαμβάνει.

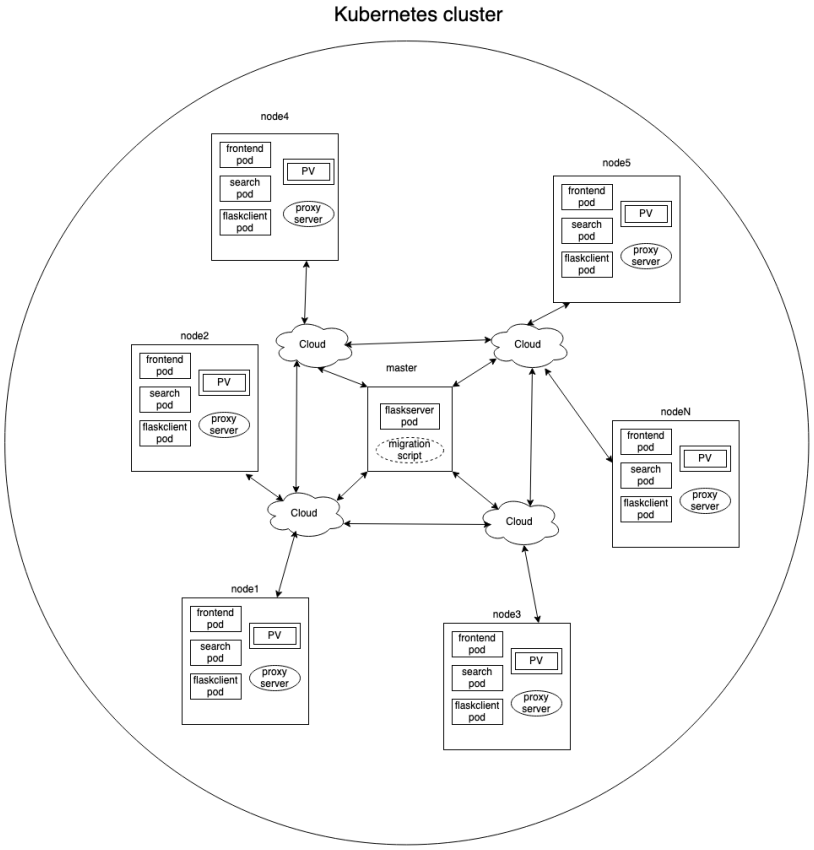
Με αυτόν τον τρόπο υλοποιείται πλήρως η επικοινωνία μεταξύ των διαφορετικών εφαρμογών που εκτελούνται σε έναν cluster του Κυβερνήτη, χωρίς να χρειάζεται ο σχεδιαστής των εφαρμογών να ανησυχεί για τον τρόπο επικοινωνίας μεταξύ αυτών, σε επίπεδο ρύθμισης δικτύου του Κυβερνήτη. Βέβαια, είναι σημαντικό να αναφέρουμε πως το api του Κυβερνήτη δε μας δίνει τη δυνατότητα να επιλέξουμε συγκεκριμένο pod να στείλουμε το πακέτο, όταν χρησιμοποιούμε το ανάλογο service ούτε να στείλουμε ένα πακέτα σε όλα τα pods που περιλαμβάνονται στο ανάλογο service.

Κεφάλαιο 4

Αρχιτεκτονική Συστήματος

Σε αυτό το κεφάλαιο αναλύουμε την υλοποίηση του συστήματος για την μεταφορά stateful εφαρμογών από τον ένα κόμβο του Κυβερνήτη στον άλλο, ανάλογα με το που βρίσκεται ο χρήστης. Αναλύουμε ποια από τα τμήματα του Κυβερνήτη που αναφέραμε στο προηγούμενο κεφάλαιο αξιοποιήσαμε και επεξηγούμε τη λειτουργία όλων των εφαρμογών που δημιουργήσαμε εμείς για την εκτέλεση του αλγορίθμου. Αναφέρονται οι βασικές τεχνικές δυσκολίες που αντιμετωπίσαμε και τέλος, περιγράφεται αναλυτικά η λειτουργία του αλγορίθμου.

4.1 Εφαρμογές για υλοποίηση του συστήματος



Σχήμα 14: Αρχιτεκτονική του συστήματος

Αρχικά, πρέπει ένα κατάλληλο σύστημα για να περιγράψουμε τον αλγόριθμό μας είναι αυτό που απεικονίζεται στο σχήμα 14 και αποτελείται από N κόμβους, έναν master και N-1 worker nodes. Οι worker nodes ανήκουν στο edge ενώ ο master μπορεί να ανήκει και σε άλλο τμήμα του cloud. Παρ΄ όλα αυτά, όλοι οι κόμβοι ανήκουν προφανώς στο ίδιο cluster του Κυβερνήτη και μπορούν να επικοι-

νωνήσουν μεταξύ τους χρησιμοποιώντας μία από τις μεθόδους που αναφέρθηκαν στην ενότητα 3.5.2. Όσον αφορά αποκλειστικά τους worker nodes, ο καθένας έχει ένα ειδικό είδος PV που θα χρησιμοποιεί ως αποθηκευτικό χώρο, τον τοπικό επίμονο δίσκο-local persistent disk (LPD) [62], ο οποίος θα αναλυθεί στην επόμενη ενότητα. Επιπροσθέτως, θεωρούμε ότι κάθε worker node συνδέεται με ένα σημείο πρόσβασης-access point και όταν συνδεθεί ο χρήστης σε αυτό, η εφαρμογή θα μεταφέρεται στο αντίστοιχο node. Επομένως, σχεδιάσαμε μία δική μας εφαρμογή προς μεταφορά μεταξύ αυτών των κόμβων, την hashapp, η οποία εκτελεί ουσιαστικά τη διαδικασία της εξόρυξης-mining του εικονικού νομίσματος bitcoin. Διαβάζει ένα αρχείο, το database.txt, γραμμή προς γραμμή και προσθέτει στο τέλος ένα νούμερο, ονομαζόμενο nonce, ξεκινώντας από το 0. Έπειτα, περνάει τη γραμμή μαζί με το νούμερο από μία συνάρτηση κατακερματισμού-hash function και προσπαθεί να βρει αποτέλεσμα το οποίο να ξεκινάει απο 5 μηδενικά. Αν δε βρει τέτοιο αποτέλεσμα, αυξάνει το nonce σειριακά, μέχρι να βρει το ζητούμενο. Όταν το βρει, γράφει στο αρχείο result.txt την γραμμή στην οποία αναφέρεται, το αποτέλεσμα της συνάρτησης κατακερματισμού και το nonce για το οποίο αυτό το αποτέλεσμα ξεκινάει με 5 μηδενικά. Μετά συνεχίζει στην επόμενη γραμμή μέχρι το τέλος του αρχείου database.txt. Προκειμένου να μεταφέρουμε την εφαρμογή μεταξύ των κόμβων, δημιουργήσαμε τις εξής επιπλέον εφαρμογές σε container, οι οποίες, σε συνεργασία με τη χρήση του api του Κυβερνήτη (θα αναλυθεί διεξοδικά στην επόμενη ενότητα), αποτελούν το σύστημα μας. Παρακάτω παρουσιάζεται συνοπτικά η λειτουργία της καθεμίας:

- **flaskserver**: Είναι υπεύθυνη για την επικοινωνία μεταξύ του master και των υπόλοιπων εφαρμογών. Όποτε μία εφαρμογή θέλει να στείλει ένα μήνυμα στον master, επικοινωνεί με το flaskserver-pod το οποίο ειδοποιεί τον master. Προφανώς, το pod αυτό τρέχει μόνο στον master για να επικοινωνεί κατευθείαν μαζί του. Επιπλέον, στέλνει δεδομένα από τον master πίσω στις εφαρμογές, τα οποία θα αναλυθούν στην ενότητα 4.4 όπου επεξηγείται συνολικά ο αλγόριθμος.

- **searching-client**: Όπως φαίνεται και από το όνομα, αυτή η εφαρμογή είναι υπεύθυνη για την αναζήτηση του χρήστη. Τρέχει ως daemonset σε κάθε worker και διαβάζει συνεχώς ένα αρχείο, το deamon.txt, το οποίο προσομοιάζει το αρχείο iptables που υπάρχει στους routers και περιέχει όλους τους χρήστες συνδεδεμένους σε αυτόν. Όταν ο χρήστης συνδέεται στον ανάλογο worker, το searching-client-pod ειδοποιεί μέσω του flaskserver-pod τον master για το που συνδέθηκε ο χρήστης.

- **migration-server**: Ένα pod που τρέχει πάντα στον κόμβο όπου βρίσκεται και η hashapp. Εξυπηρετεί το directory όπου υπάρχουν το database.txt και result.txt έτσι όταν χρειαστεί να μεταφερθούν μαζί με το hashapp-pod, να συνδεθεί η υπεύθυνη για τη μεταφορά εφαρμογή σε αυτόν και να τα μεταφέρει.

- **flaskclient**: Είναι η εφαρμογή υπεύθυνη για τη μεταφορά του database.txt και result.txt που αναφέρθηκε παραπάνω. Τρέχει και αυτή σε όλους τους workers σαν daemonset. Κάθε pod της περιμένει μήνυμα από το flaskserver-pod και όταν το λάβει, συνδέεται στο migration-server pod και μεταφέρει τα δύο αυτά αρχεία στο μηχάνημα όπου βρίσκεται.

- **frontend**: Μία εφαρμογή με την οποία επικοινωνεί ο χρήστης για να ανεβάσει το database.txt αρχείο του και να λάβει όποτε αυτός επιθυμεί το result.txt. Τρέχει παντού ως daemonset και κάθε pod ανεβάζει στο μηχάνημα όπου τρέχει, το database.txt στο ίδιο directory που θα χρησιμοποιήσει η hashapp εφαρμογή.

Όλες οι docker images που δημιουργήθηκαν για αυτές τις εφαρμογές υπάρχουν σε μία δημόσια αποθήκη-public repository του docker hub [63], διαθέσιμες για όλους προς χρήση.
Για την επικοινωνία μεταξύ των εφαρμογών χρησιμοποιήθηκε το εργαλείο Flask [64], το οποίο χρησιμοποιεί τη γλώσσα προγραμματισμού python. Με τη χρήση αυτού, κατασκευάζουμε τις εφαρμογές ως server για επικοινωνία, που ακούνε σε συγκεκριμένη IP και Port, και υλοποιούν τις προαναφερθείσες λειτουργίες. Από αυτή τη διαδικασία εξαιρείται η searching-client, η οποία ουσιαστικά είναι

μία εφαρμογή για ανάγνωση αρχείου και δε χρειάζεται να δημιουργηθεί ως server. Όπως αναφέραμε όμως στην ενότητα 3.5.3, δε μπορούμε να χρησιμοποιήσουμε τις IP των pod για επικοινωνία, καθώς τα pod είναι εφήμερες οντότητες και κάθε φορά που αλλάζουν, επανεκκινούνται ή μεταφέρονται αλλάζουν και οι IP τους. Επομένως, κατασκευάσαμε μία service για κάθε είδους εφαρμογή ώστε τα pods να επικοινωνούν μαζί τους, και αυτές να στείλουν τα μηνύματα στις ανάλογες ip, όπου ακούν οι servers μας. Παρακάτω παρατίθεται ο πίνακας 2, όπου παρουσιάζεται κάθε εφαρμογή που λειτουργεί ως server με το άναλογο service που δημιουργήθηκε για την επικοινωνία με αυτή, καθώς και το είδος του service.

app image	service	service type
flaskserver	flaskserver-service	clusterip
migration server	migration-service	clusterip
flaskclient	flaskclient-service	clusterip
frontend	frontend-service	nodeport

Πίνακας 2: Εφαρμογές-server για την υλοποιήση του συστήματος μαζί με τα ανάλογα services

Παρατηρούμε ότι για τις 3 πρώτες εφαρμογές έχουμε δημιουργήσει services τύπου clusterip. Αυτό σημαίνει ότι ο χρήστης δεν έχει πρόσβαση σε αυτές και ότι επιτρέπουν μόνο την επικοινωνία μεταξύ των εφαρμογών μεταξύ τους. Αντιθέτως, προκειμένου να καταφέρει ο χρήστης να ανεβάσει τη βάση δεδομένων του και να λάβει το αποτέλεσμα της εφαρμογής hashapp, πρέπει να έχει πρόσβαση στα pod του frontend και γι΄αυτό δημιουργήσαμε ένα service τύπου nodeport.

4.2 Αυτοματοποίηση μεταφοράς

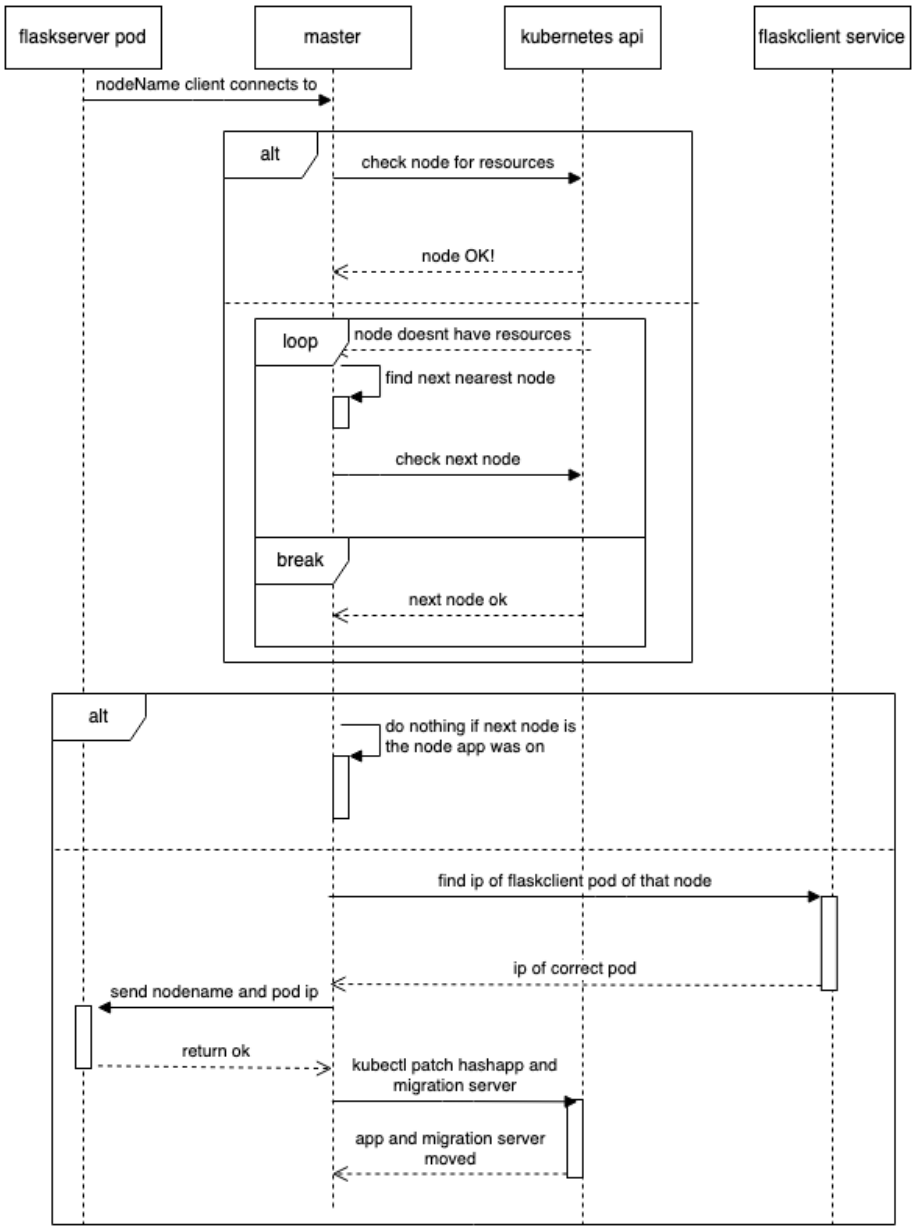
Για να αξιοποιήσουμε πραγματικά τα οφέλη του edge computing, πρέπει η hashapp να τρέχει αποκλειστικά σε κόμβους του edge. Επομένως, δε μπορεί να χρησιμοποιήσει αποθηκευτικούς χώρους που προσφέρουν οι πάροχοι νέφους και βρίσκονται σε άλλους κόμβους του cloud αλλά directories των ίδιων των κόμβων όπου τρέχει. Όπως αναφέρθηκε όμως στην ενότητα 3.4.1, τα hostpath pv που το επιτρέπουν αυτό λειτουργούν μόνο για daemonset-pods, γεγονός που σημαίνει ότι η εφαρμογή θα έπρεπε να τρέχει ταυτόχρονα σε κάθε κόμβο. Προκειμένου να αντιμετωπιστεί το πρόβλημα αυτό, χρησιμοποιήσαμε ένα ειδικό είδος PV, τον τοπικό επίμονο δίσκο-local persistent disk (LPD) [62], το οποίο δημιουργείται σε συγκεκριμένο κόμβο και χρησιμοποιεί directories του για αποθηκευτικό χώρο. Έτσι, δημιουργούμε ένα PV με ReadWriteMany δικαιώματα σε κάθε worker node, για να έχουν πρόσβαση σε αυτό όλες οι εφαρμογές που αναφέρθηκαν, καθώς και ένα PVC για κάθε PV.Ειδικότερα, το pvc1 αναφέρεται στο PV του node1, το pvc2 στο pv του node2 κ.ο.κ. Όποτε λοιπόν θέλουμε να μεταφέρουμε την εφαρμογή σε συγκεκριμένο κόμβο, αναφερόμαστε στο ανάλογο pvc, αναγκάζοντας τον Κυβερνήτη να μεταφέρει το pod σε εκείνο τον κόμβο, ώστε να φέρει την πραγματική κατάσταση στην επιθυμητή κατάσατση. Με αυτό τον τρόπο ωστόσο, ορίζουμε εμείς κάθε φορά τον κόμβο που θα πρέπει να τρέχει η εφαρμογή, μέθοδος που αντιτίθεται στη βασική λογική του Κυβερνήτη, δηλαδή να βρίσκει κάθε φορά ο ίδιος τον βέλτιστο κόμβο για την εκτέλεση της εφαρμογής.

Επιπροσθέτως, η χρήση των LPD επιφέρει και ένα δεύτερο πρόβλημα που σχετίζεται με τη λειτουργία του kube-scheduler και το πως χειρίζεται την έλλειψη πόρων. Ειδικότερα, τα κριτήρια για πόρους είναι οι κεντρικές μονάδες επεξεργασίας-central processing units (CPUs) καθώς και η μνήμη που έχουμε στη διάθεση μας σε έναν κόμβο. Ο Κυβερνήτης μας δίνει τη δυνατότητα να ορίσουμε στην επιθυμητή κατάσταση μέσω των yaml αρχείων τόσο τις CPUs όσο και την μνήμη που θέλουμε να αξιοποιήσει η εφαρμογή μας. Με βάση αυτά, ο kube-scheduler κρίνει πως θα τρέχουν τα pods στους κόμβους. Άμα ένας κόμβος δεν έχει αρκετές CPUs για να εκτελεστούν όλα τα pods, τότε θα

μειώσει την απόδοση των pods που απαιτούν το μεγαλύτερο ποσοστό χρήσης των CPUs. Αντιθέτως, αν δεν έχει αρκετή μνήμη, τότε ο scheduler καταστρέφει το pod που απαιτεί το μεγαλύτερο ποσοστό μνήμης για να τοποθετήσει το καινούριο pod και το μεταφέρει σε άλλο κατάλληλο κόμβο. Ορίζοντας εμείς τον κόμβο όπου θα εκτελεστεί η εφαρμογή, αναγκάζουμε τον scheduler να προσπαθήσει να εκτελέσει το container εκεί και δεν επιτρέπουμε την μεταφορά του σε άλλο κόμβο. Αυτό σημαίνει πως αν ο κόμβος αυτός δεν έχει την απαραίτητη μνήμη διαθέσιμη, επειδή τα pod των υπόλοιπων εφαρμογών που δημιουργήσαμε και τρέχουν εκεί την αξιοποιούν, θα προσπαθήσει να ελευθερώσει αρκετή μνήμη προκειμένου να τρέξει η hashapp, καταστρέφοντας ένα από τα προαναφερθέντα pods. Στην περίπτωση μας, αυτό θα ήταν καταστροφικό, καθώς κάθε pod είναι απαραίτητο για τη σωστή λειτουργία του συστήματος μας.

Με στόχο την λύση αυτών των προβλημάτων, δημιουργήσαμε μία αυτόματη διαδικασία-script, η οποία θα προσπερνάει ουσιαστικά τη λειτουργία του scheduler. Ειδικότερα, περιμένει να λάβει μήνυμα από το flaskserver-pod, που θα ενημερώνει τον master σε ποιον κόμβο συνδέθηκε ο χρήστης. Έπειτα, το script αυτό θα χρησιμοποιεί κατάλληλα το api του Κυβερνήτη, για να υπολογίσει αν οι πόροι του συγκεκριμένου κόμβου είναι αρκετοί ώστε να μεταφερθεί εκεί η εφαρμογή. Αν δεν είναι, βρίσκει τον κόμβο που είναι ποιο κοντά στον κόμβο που συνδέθηκε ο χρήστης και κάνει τους ίδιους υπολογισμούς για αυτόν. Η διαδικασία συνεχίζεται μέχρι να βρεθεί ένας κατάλληλος κόμβος. Έτσι, ο scheduler δε θα χρειαστεί να καταστρέψει κάποιο pod, καθώς έχουν γίνει ήδη οι κατάλληλοι υπολογισμοί ώστε να χρησιμοποιθεί κόμβος που έχει σίγουρα διαθέσιμους πόρους. Όσον αφορά τη μεταφορά του hashapp-pod, χρησιμοποιούμε τη μέθοδο συρραφής-patch του Κυβερνήτη [65]. Πιο συγκεκριμένα, με τη χρήση ενός ειδικού αρχείου που ονομάζεται patch-file, μπορούμε να τροποποιήσουμε την επιθυμητή κατάσταση ενός deployment, χωρίς να τροποποιούμε το αρχικό yaml αρχείο του deployment αυτού. Όταν λοιπόν βρεθεί κατάλληλος κόμβος, το script αναζητά το pv που υπάρχει σε αυτόν, βρίσκει το pvc που αναφέρεται σε αυτό και χρησιμοποιεί το patch-file για να αλλάξει το pvc που χρησιμοποιεί η hashapp μέσω του deployment αρχείου της, αλλάζοντας έτσι και τη επιθυμητή κατάσταση και αναγκάζοντας τον Κυβερνήτη να μεταφέρει το pod. Επομένως,αυτοματοποιούμε πλήρως τη διαδικασία εύρεσης του βέλτιστου κόμβου και της μεταφοράς, χωρίς να χρειαστεί να ανησυχούμε για το πως θα λειτουργήσει ο scheduler, αφού προσπερνάμε πλήρως αυτό το τμήμα της λειτουργίας του. Όλα αυτά φαίνονται πλήρως στο σχήμα 15 .

Πρέπει εδώ να αναφέρουμε τα εξής. Αρχικά, η μέθοδος patch λειτουργεί μόνο εφόσον έχει εκτελεστεί ήδη κάπου το deployment για την hashapp και πως δε μπορεί να εκκινήσει την εφαρμογή. Άρα, την πρώτη φορά πρέπει πράγματι να τροποποιήσουμε το deployment αρχείο ώστε να εκτελεστεί σε συγκεκριμένο κόμβο η εφαρμογή και όταν θέλουμε έπειτα να την μεταφέρουμε, τότε θα χρησιμοποιήσουμε τη μέθοδο patch. Επιπλέον, ο master γνωρίζει εξ΄αρχής τις αποστάσεις μεταξύ των worker nodes, τις οποίες έχει αποθηκευμένες σε ένα αρχείο ονομαζόμενο apostaseis.txt. Οι αποστάσεις αυτές μετριούνται σε βήματα-hops που χρειάζεται ένα πακέτο να κάνει για να μεταφερθεί από τον ένα κόμβο στον άλλο. Τέλος, το κριτήριο το οποίο ελέγχει ο master είναι η διαθέσιμη μνήμη, καθώς όπως αναφέραμε παραπάνω είναι το κριτήριο υπεύθυνο για την καταστροφή των pods, σε περίπτωση έλλειψής της.



Σχήμα 15: Λειτουργία Script στον master

4.3 Διαμεσολαβητές

Οι worker nodes προσομοιάζουν servers που ανήκουν στο edge και θα εκτελούν την εφαρμογή μας. Λόγω όμως της τοπολογίας που χρησιμοποιήθηκε στα πειράματα, έχουμε πρόσβαση μόνο σε 4 εικονικά μηχανήματα τα οποία έχουν μία εξωτερική και μία εσωτερική IP. Το σύστημα του Κυβερνήτη που έχουμε κατασκευάσει είναι ρυθμισμένο με βάση τις εσωτερικές IP και τα σημεία πρόσβασης-access points στα οποία θα συνδέεται ο χρήστης δεν έχουν απευθείας πρόσβαση σε αυτές, αλλά μπορούν να επικοινωνήσουν μόνο με τις εξωτερικές. Για να λύσουμε το πρόβλημα της επικοινωνίας χρήστη-συστήματος λοιπόν, δημιουργήσαμε σε κάθε worker node και από έναν εξυπηρετητή που λειτουργεί ως διαμεσολαβητής-proxy server και ο οποίος επικοινωνεί απευθείας με το pod του frontend που βρίσκεται σε εκείνον τον worker node. Έτσι, αντιστοιχίζοντας έναν proxy server και άρα έναν worker node σε κάθε access point, ικανοποιούμε τις εξής 4 συνθήκες:

- **Είσοδο χρήστη**:Τα access point είναι ρυθμισμένα έτσι ώστε κάθε φορά που συνδέεται ο χρήστης σε ένα από αυτά, να στέλνουν μία εντολή http στον ανάλογο proxy server. Αυτός, λαμβάνοντας αυτή την εντολή τροποποιεί κατάλληλα το αρχείο του worker node που προσομοιάζει το iptables αρχείο των routers. Επομένως, η searching-client εφαρμογή αναγνωρίζει την αλλαγή του αρχείου και στέλνει μήνυμα στον flask-server.

- **Έξοδο χρήστη**: Ομοίως με την είσοδο του χρήστη, όταν αυτός αποσυνδέεται από το access point, αυτό στέλνει πάλι μία http εντολή στον proxy. Ο server αυτός τροποποιεί πάλι το αρχείο που αναφέραμε και το επιστρέφει στην προηγούμενη του κατάσταση, γεγονός που αναγνωρίζει η searching-client.

- **Ανέβασμα βάσης δεδομένων**: Όταν ο χρήστης το επιθυμήσει, μπορεί να συνδεθεί απευθείας με τον proxy και να ανεβάσει σε αυτόν την βάση δεδομένων του. Ο proxy θα αποθηκεύσει τη βάση αυτή σε ένα τυχαίο directory του worker node και μετά θα την ανεβάσει απευθείας στο pod του frontend, το οποίο θα την αποθηκεύσει κατάλληλα στο PV όπου χρησιμοποιεί η hashapp εφαρμογή.

- **Επιστροφή αποτελέσματος**: Ο χρήστης τέλος όταν θελήσει να λάβει το αποτέλεσμα του, το ζητάει από τον proxy server. Αυτός ειδοποιεί το frontend pod του ανάλογου worker node, το οποίο του επιστρέφει το πιο πρόσφατο result.txt αρχείο. Έπειτα, ο proxy server το αποθηκεύει προσωρινά και το επιστρέφει και αυτός με τη σειρά του στον χρήστη.

Με αυτόν τον τρόπο λύνουμε το θέμα επικοινωνίας του χρήστη με τους servers και της εισόδου του χρήστη σε αυτούς, επιτρέποντας μας έτσι να υλοποιήσουμε τον αλγόριθμο μεταφοράς του container.

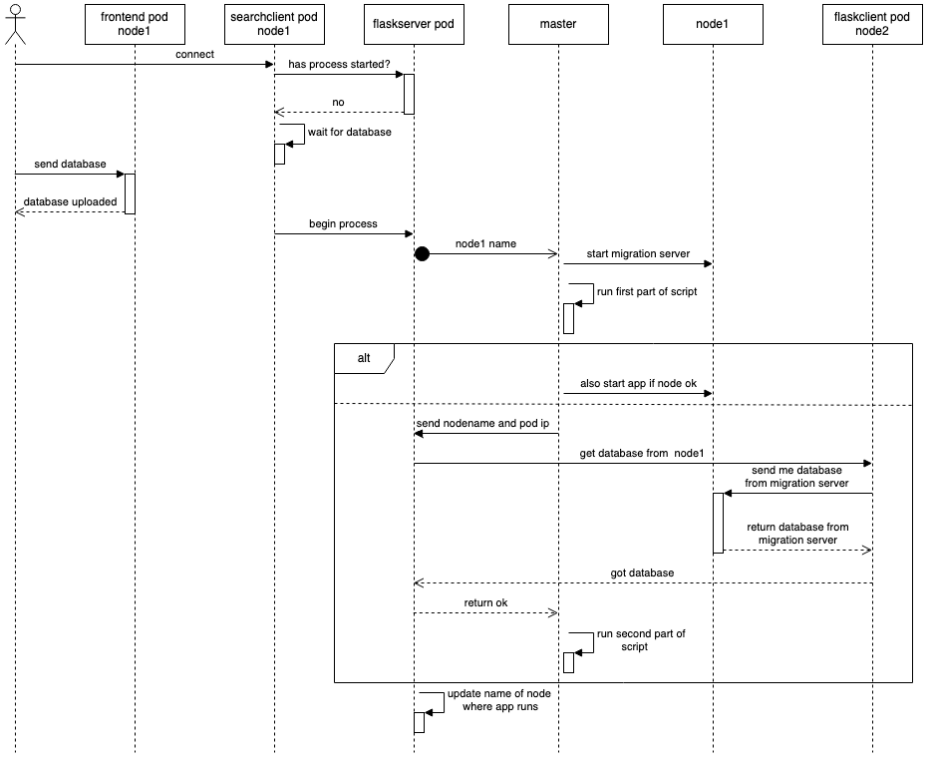
4.4 Αλγόριθμος μεταφοράς για εφαρμογή με κατάσταση

Αφού εξηγήσαμε αναλυτικά το κάθε τμήμα του συστήματος μας, θα παρουσιάσουμε τώρα αναλυτικά το πως συνεργάζονται αυτά μεταξύ τους προκειμένου να μεταφέρεται αυτόματα κάθε φορά στο ανάλογο εικονικό μηχάνημα (που προσομοιάζει έναν server) η εφαρμογή και να συνεχίζει από το σημείο που βρισκόταν. Παρουσιάζουμε λοιπόν τον αλγόριθμο για όλες τις διαφορετικές περιπτώσεις που υπάρχουν, οι οποίες είναι η έναρξη της εφαρμογής, μία τυχαία χρονική στιγμή όπου θα εκκινήσει η διαδικασία της μεταφοράς καθώς και το τέλος της διαδικασίας, δηλαδή όταν επιθυμήσει ο χρήστης να λάβει το αποτέλεσμα του. Να διευκρινίσουμε πως όταν λέμε ότι επικοινωνούμε με ένα pod, εννοούμε με το αντίστοιχο service που αναφέραμε στον πίνακα 2.

4.4.1 Πρώτη φορά

Στο σχήμα 16, παρουσίαζεται ο αλγόριθμος για την εκκίνηση της εφαρμογής στον σωστό κόμβο. Θεωρούμε ότι έχουμε εκκινήσει τα searching-client,frontend και flaskclient daemonset καθώς και το

flaskserver deployment και περιμένουμε τον πελάτη να ανεβάσει τη βάση δεδομένων που επιθυμεί να χρησιμοποιήσει η hashapp εφαρμογή. Για λόγους ευκολίας, δεν αναφέρουμε τη χρήση του proxy server καθώς αναλύθηκε στην προηγούμενη ενότητα και θεωρούμε ότι ο πελάτης επικοινωνεί κατευθείαν με το frontend. Επιπλέον, θεωρούμε ως πρώτο τμήμα το script όλα τα βήματα του σχήματος 15 μέχρι και την εύρεση του σωστού κόμβου ενώ ως δεύτερο τμήμα την τροποποίηση του deployment αρχείου για την hashapp και τη χρήση της μεθόδου patch για τον migration server. Τα βήματα λοιπόν για αυτή τη διαδικασία είναι τα εξής:



Σχήμα 16: Ο αλγόριθμος για την έναρξη της εφαρμογής

1. Ο χρήστης συνδέεται στον worker node1, γεγονός το οποίο εντοπίζει η searching-client και η οποία ρωτάει τον flaskserver αν έχει εκκινήσει η διαδικασία.

2. Αφού είναι η πρώτη φορά, ο flaskserver απαντάει αρνητικά στην searching-client.

3. Με την αρνητική απάντηση, η searching-client περιμένει είτε να ανεβεί στον worker αυτόν η βάση δεδομένων είτε ο χρήστης να αποσυνδεθεί από αυτόν και να συνδεθεί σε άλλον worker, όπου θα επαναληφθεί αυτή η διαδικασία.

4. Έστω ότι ο χρήστης δε συνδέεται αλλού και ανεβάζει σε αυτόν τον worker τη βάση. Η searching client, έχοντας και αυτή πρόσβαση στο PV που αποθηκεύεται η βάση, αναγνωρίζει αυτή την αλλαγή και στέλνει στον flaskserver το όνομα του worker όπου ανέβηκε η βάση.

5. Ο flaskserver ενημερώνεται για την έναρξη της διαδικασίας και μεταφέρει στον master το όνομα του worker.

6. Ο master, χρησιμοποιώντας το script που αναφέραμε, ενεργοποιεί αρχικά τον migration server στον worker αυτόν. Έπειτα, υπολογίζει αν οι πόροι που διαθέτει είναι αρκετοί για να εκιννήσει εκεί την hashapp εφαρμογή. Αν δεν είναι, βρίσκει τον αμέσως επόμενο δυνατό και πιο κοντινό στον node1, δηλαδή στην περίπτωσή μας τον node2. Επιπλέον, μέσω της χρήσης του api και του flaskclient-service, βρίσκει την ip του flaskclient-pod που ανήκει στον node όπου μπορεί να

ξεκινήσει τελικά η hashapp. Αυτό γίνεται επειδή ο flaskserver θα χρειαστεί να επικοινωνήσει με το συγκεκριμένο flaskclient-pod που λειτουργεί σε αυτόν τον κόμβο, κάτι που δε μας επιτρέπει απευθείας το api του Κυβερνήτη. Άρα, εδώ πρέπει να επικοινωνήσουμε απευθείας με IP και όχι μέσω service.

7. Αν ο node1 έχει αρκετούς πόρους, τότε το script ενεργοποιεί εκεί και το hashapp pod. Αν όχι, τότε ερχόμαστε σε αυτό το βήμα, όπου στέλνει με μία http post εντολή το όνομα του επόμενου κόμβου που έχει πόρους, έστω τον node2, και την ip του flaskclient-pod που βρίσκεται στον node2 στο flaskserver pod. Ο flaskserver χρησιμοποιεί αυτή την ip για να ειδοποιήσει το flaskclient-pod του node2 με μία άλλη http post εντολή να πάρει την database από τον migration-server που τρέχει στον node1. Αποθηκεύει επίσης το όνομα του κόμβου όπου θα εκτελεστεί η hashapp.

8. Ο flaskclient λαμβάνει την database και επιστρέφει στον flaskserver την επιβεβαίωση της μεταφοράς του database. Αυτός επίσης επιστρέφει στον master την επιβεβαίωση αυτή και ο master μεταφέρει εκεί τον migration-server και ενεργοποιεί εκεί την hashapp εφαρμογή.

Έτσι, είτε ο αρχικός κόμβος είτε ο αμέσως επόμενος έχουν ταυτόχρονα το pod του hashapp καθώς και το pod του migration-server. Από εκεί και πέρα, τρέχουν κάθε φορά σε έναν worker ταυτόχρονα 5 pods και όταν ο χρήστης αποσυνδεθεί από έναν κόμβο και και συνδεθεί σε έναν διαφορετικό, θα ενεργοποιηθεί η επόμενη περίπτωση του αλγορίθμου, δηλαδή η διαδικασία της μεταφοράς του pod.

4.4.2 Μεταφορά της εφαρμογής

Έστω λοιπόν ότι ο πρώτος κόμβος όπου συνδέθηκε ο χρήστης είχε αρκετούς πόρους για να εκκινήσει η hashapp και δε χρειάστηκε να μεταφερθεί η βάση. Επομένως, τρέχουν τώρα στον κόμβο αυτό τα 5 pods που αναφέραμε και ο χρήστης μετακινείται από αυτόν σε έναν άλλο κόμβο. Θεωρούμε τώρα τα εξής. Πρωτίστως, ο κόμβος που μεταφέρθηκε είναι ο πιο κόντινος στον προηγούμενο κόμβο και το αντίστροφο. Επίσης, θεωρούμε πάλι ως πρώτο τμήμα του script όλα τα βήματα του σχήματος 15 μέχρι και την εύρεση του σωστού κόμβου ενώ ως δεύτερο τμήμα τη χρήση της μεθόδου patch για τον migration server και την hashapp. Η διαδικασία που ακολουθείται λοιπόν και φαίνεται στο σχήμα 17 είναι η εξής:

1. Η searching-client εντοπίζει τον χρήστη στον node2 και ρωτάει τον flaskserver αν έχει εκκινήσει η διαδικασία.

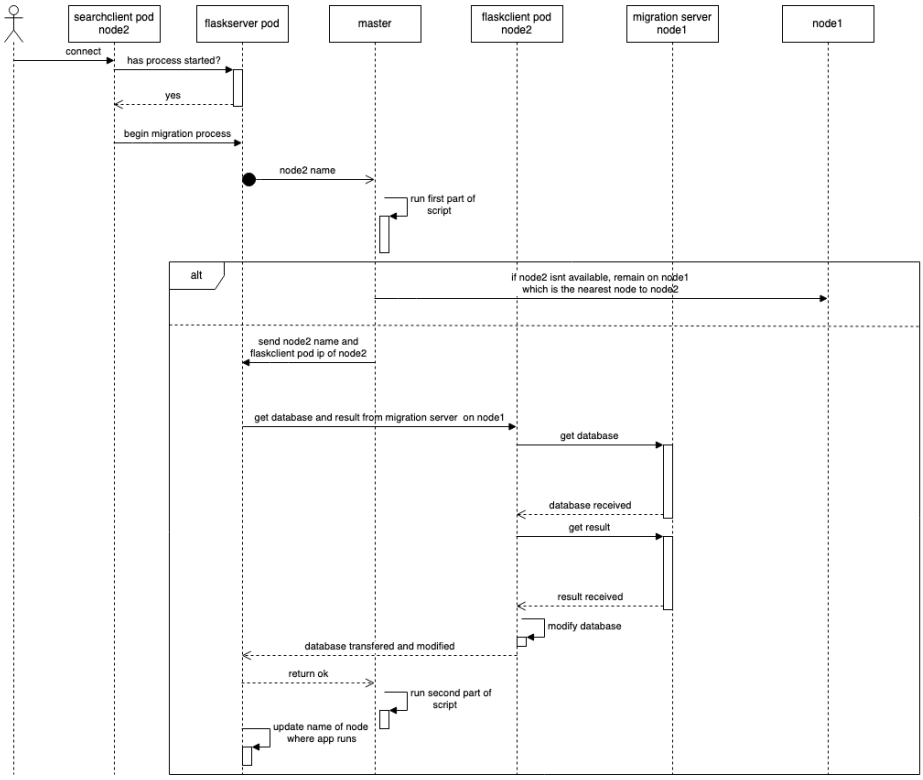
2. Ο flaskserver απαντάει θετικά στην searching-client.

3. Η searching-client καταλαβαίνει ότι δε πρέπει να περιμένει να ανεβεί η βάση δεδομένων από τον χρήστη και στέλνει στον flaskserver το όνομα του κόμβου, προκειμένου να ελέγξει ο master αν μπορεί η εφαρμογή να μεταφερθεί εκεί.

4. Ο flaskserver μεταφέρει το όνομα στον master.

5. Ο master ελέγχει πάλι τον κόμβο node2 αν έχει αρκετούς διαθέσιμους πόρους, αλλιώς ελέγχει τον αμέσως πιο κοντινό, στην περίπτωση μας το node1. Αν ο node2 δεν έχει αρκετούς πόρους, τότε η διαδικασία σταματάει εδώ, αφού η εφαρμογή τρέχει ήδη στον node1, που είναι ο πιο κοντινός κόμβος στο node2. Έστω εδώ ότι ο node2 έχει αρκετούς πόρους, άρα μπορεί να μεταφερθεί η εφαρμογή εκεί. Ο master στέλνει όπως πριν με μία εντολή http post το όνομα του κόμβου που θα μεταφερθεί η εφαρμογή και την IP του flaskclient-pod που τρέχει στον κόμβο αυτό.

6. Ο flaskserver χρησιμοποιεί την IP αυτή για να στείλει μήνυμα στο αντίστοιχο flaskclient-pod να μεταφέρει τη βάση δεδομένων και το αποτέλεσμα, χρησιμοποιώντας μία http get εντολή.



Σχήμα 17: Περίπτωση μεταφοράς του pod σε άλλον κόμβο

7. Το flaskclient-pod μεταφέρει τη βάση δεδομένων και το αρχείο με τα αποτελέσματα. Επίσης, διαβάζει την τελευταία γραμμή του αρχείου αποτελεσμάτων, αναγνωρίζει την τελευταία γραμμή της βάσης δεδομένων που πέρασε επιτυχώς από τη hash function και αφαιρεί όλες τις γραμμές της βάσης δεδομένων μέχρι και αυτή τη γραμμή. Έτσι, όταν επανέλθει η hashapp σε λειτουργία, θα συνεχίσει από το σημείο που ήταν, διατηρώντας έτσι την κατάσταση της.

8. Το flaskclient-pod ειδοποιεί τον flaskserver ότι ολοκληρώθηκε η μεταφορά. Ο flaskserver ανανεώνει πλέον το όνομα του κόμβου που τρέχει η εφαρμογή που είχε αποθηκεύσει προηγουμένως.

9. Ο flaskserver ειδοποιεί και αυτός τον master ότι η μεταφορά του database ολοκληρώθηκε και έτσι ο master μεταφέρει τώρα τον migration-server και την hashapp με τη χρήση της μεθόδου patch, όπως περιγράφθηκε στην ενότητα 4.2.

Για λόγους καλύτερης οπτικοποίησης του αλγορίθμου, θεωρήσαμε ότι ο πιο κοντινός κόμβος σε αυτόν που μεταφέρθηκε ο χρήστης ήταν ο κόμβος που έτρεχε ήδη η εφαρμογή, ώστε αν ο καινούριος κόμβος δεν είχε την απαραίτητη μνήμη, να παραμείνει εκεί το hashapp pod και ο migration server. Αν ο πιο κοντινός κόμβος ήταν άλλος, τότε θα ακολουθείτο η ίδια ακριβώς διαδικασία με αυτήν που ακολουθήθηκε για τον καινούριο κόμβο.

Είναι σημαντικό επίσης να αναφέρουμε το εξής. Όταν ο master στέλνει το όνομα του node και την IP στον flaskserver, το κάνει με ένα http post. Αυτό σημαίνει ότι για να μπορέσει να συνεχίσει το script που εκτελεί και να συνεχίσει στην επόμενη εντολή, πρέπει να πάρει μία απάντηση από τον flaskserver. Επειδή όμως και αυτός εκτελεί μία http get εντολή σε ένα flaskclient-pod, περιμένει και αυτός απάντηση, η οποία έρχεται όταν γίνεται η μεταφορά των αρχείων. Έτσι, επιστρέφει απάντηση στον master όταν έχει γίνει η μεταφορά αυτή και άρα μόνο τότε μπορεί να συνεχίστει η εκτέλεση του script. Το σύστημα μας λοιπόν είναι απόλυτα συγχρονισμένο και δεν υπάρχει περίπτωση να γίνει πιο γρήγορα η μεταφορά των ανάλογων pod από ότι των δύο αρχείων και να μη μπορέσει έτσι να συνεχιστεί σωστά η λειτουργία του συστήματος.

Καταφέραμε λοιπόν έτσι να μεταφέρουμε πλήρως τα απαραίτητα αρχεία για την εκτέλεση της εφαρμογής, την ίδια την εφαρμογή και κυρίως, την κατάσταση της εφαρμογής στον πιο κοντινό στον χρήστη κόμβο. Πρέπει τώρα να επιτρέψουμε στον χρήστη να μπορεί να λάβει το πιο πρόσφατο αποτέλεσμα όποτε το επιθυμήσει.

4.4.3 Τέλος διαδικασίας

Όπως εξηγήσαμε, ο χρήστης μπορεί να είναι συνδεδεμένος στον πιο κοντινό σε αυτόν server (για την ακρίβεια στο VM που αντιστοιχεί σε αυτόν τον server) αλλά λόγω έλλειψης πόρων, η εφαρμογή να μην εκτελείται σε αυτόν τον κόμβο. Επομένως, όταν θελήσει να λάβει το αρχείο αποτελεσμάτων, πρέπει να μπορεί να πάρει το πιο πρόσφατο αρχείο, που δεν είναι αναγκαστικά αυτό που βρίσκεται στον κόμβο που είναι συνδεδεμένος. Το πρόβλημα αυτό το λύνουμε ως εξής. Όταν ο χρήστης θελήσει, ζητάει από το pod του frontend που υπάρχει στον server που είναι συνδεδεμένος το αρχείο αποτελεσμάτων (υπενθυμίζουμε μέσω του proxy). Το pod αυτό ρωτάει τον flaskserver που εκτελείται η hashapp εκείνη τη στιγμή. Αν τρέχει στον κόμβο αυτό, τότε του στέλνει κατευθείαν το αρχείο, αφού αυτό είναι το πιο πρόσφατο. Αλλιώς, αν τρέχει σε άλλο κόμβο, σημαίνει ότι το πιο πρόσφατο αρχείο αποτελεσμάτων υπάρχει σε εκείνον τον κόμβο. Επομένως, το frontend-pod συνδέεται στον migration-server που όπως αναφέραμε, βρίσκεται πάντα στον κόμβο όπου τρέχει η εφαρμογή, και λαμβάνει μόνο το result.txt, το οποίο επιστρέφει στον χρήστη.

Πρέπει να αναφέρουμε επιπλέον ότι αν ο χρήστης επιθυμήσει να λάβει το αρχείο κατά τη διάρκεια της μεταφοράς, θα λάβει το αρχείο από τον κόμβο που βρισκόταν η εφαρμογή πριν την έναρξη της. Αυτό συμβαίνει επειδή ο flaskserver ανανεώνει το όνομα του κόμβου αφού ολοκληρωθεί πλήρως η μεταφορά. Έτσι, αν κατά τη διάρκεια της μεταφοράς η hashapp έχει αποθηκεύσει καινούρια αποτελέσματα στο result.txt, ο χρήστης θα λάβει και αυτά τα αποτελέσματα. Βέβαια, ότι έχει παράξει η hashapp κατά τη διάρκεια της μεταφοράς θα πρέπει να υπολογιστεί ξανά στον καινούριο κόμβο, αφού η κατάσταση της εφαρμογής που είχαμε αποθηκεύσει ήταν σε προηγούμενο στάδιο.

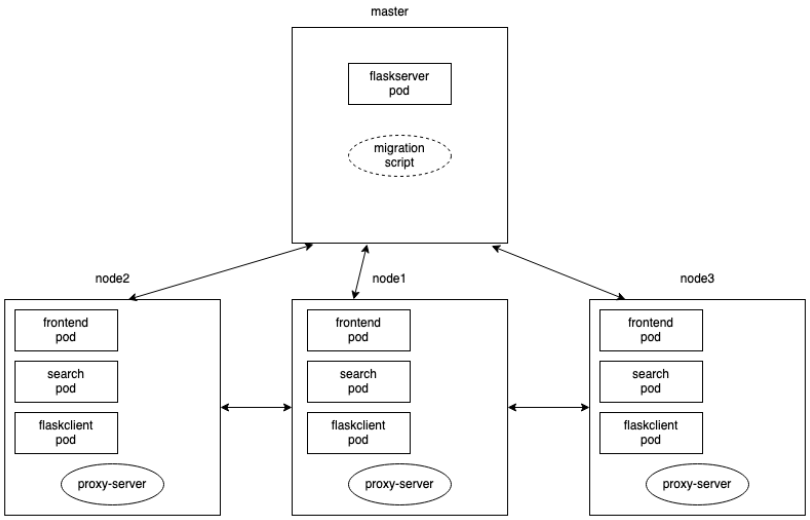
Παρ΄όλα αυτά, καταφέραμε να μεταφέρουμε πλήρως τόσο την stateful εφαρμογή όσο και τα αποτελέσματα της για να τα λάβει ο χρήστης όποτε επιθυμήσει, χάνοντας για ένα μικρό διάστημα, όσο διαρκεί η μεταφορά του database, ένα ελάχιστο ποσοστό των αποτελεσμάτων της εφαρμογής.

Κεφάλαιο 5

Πειραματική Αξιολόγηση

Στο κεφάλαιο αυτό παρουσιάζονται οι τεχνικές λεπτομέρειες και η πειραματική αξιολόγηση του αλγορίθμου μεταφοράς για εφαρμογή με κατάσταση που περιγράφθηκε στο προηγούμενο κεφάλαιο.

5.1 Λεπτομέρειες υλοποίησης



Σχήμα 18: Η τοπολογία του πειράματός μας

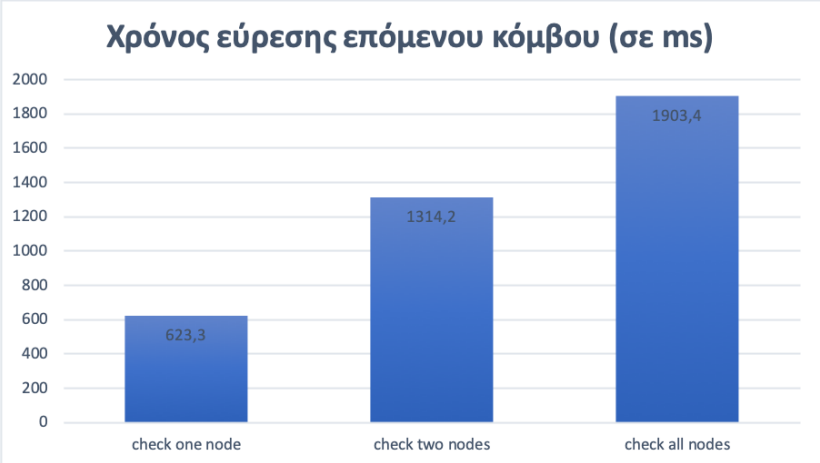
Αρχικά, όπως αναφέραμε και στην ενότητα 4.3, μας δίνονται 4 εικονικά μηχανήματα, που προσομοιάζουν 4 servers, έναν master node και 3 worker nodes, ονομαζόμενοι node1,node2 και node3 αντίστοιχα. Ο node1 είναι ο πιο κοντινός κόμβος τόσο για τον node2 όσο και για τον node3 και απέχει ένα βήμα από τον καθένα ενώ ο node3 είναι ο πιο απομακρυσμένος. Η διάταξη του σχήματος 18 παρουσιάζει αυτές τις συνθήκες. Κάθε worker node χρησιμοποιεί και έναν εφήμερο αποθηκευτικό χώρο-ephemeral storage ως LPD, που θα θεωρούμε δεδομένο ότι θα έχει πάντα αρκετό χώρο προκειμένου να μεταφερθούν τα αρχεία της εφαρμογής. Επιπλέον, θεωρούμε ότι σίγουρα κάποιος κόμβος θα έχει αρκετούς υπολογιστικούς πόρους, δηλαδή μνήμη, για να εκτελεστεί η εφαρμογή. Όσον αφορά τον ίδιο το χρήστη, θεωρούμε ότι θα συνδεθεί σίγουρα στο access point με το πιο ισχυρό σήμα και άρα η μεταφορά των αρχείων θα είναι η γρηγορότερα δυνατή. Τέλος, θεωρούμε ότι πριν αρχίσουν τα πειράματα έχουμε ήδη εκκινήσει, όπως και στην περιγραφή του αλγορίθμου, τα searching-client,frontend και flaskclient daemonset, το flaskserver deployment και τον proxy server του κάθε εικονικού μηχανήματος.

5.2 Διεξαγωγή πειραμάτων

Αφού αναφέραμε τις τεχνικές λεπτομέρειες της υλοποίησης, είμαστε έτοιμοι να διεξάγουμε τα πειράματα μας. Χρησιμοποιήσαμε 3 αρχεία διαφορετικού μεγέθους το καθένα, ένα των 10 KB, ένα των 10 MB και ένα του 1GB ως βάση δεδομένων. Έπειτα μετρήσαμε τον χρόνο που κάνει να μεταφερθεί σε κάθε περίπτωση η βάση αυτή, καθώς και τον χρόνο που χρειάζεται ο master να βρει τον καινούριο κόμβο όπου θα μεταφερθεί η βάση και τον χρόνο για να μεταφέρει το ίδιο το pod. Τέλος, μετρήσαμε για κάθε βάση τον μέγιστο χρόνο που θα χρειαστεί να περάσει προκειμένου να λάβει το αρχείο αποτελεσμάτων ο χρήστης.

5.2.1 Χρόνος εύρεσης επόμενου κόμβου

Αρχικά, παρουσιάζουμε τους χρόνους που χρειάστηκαν για την εύρεση του κόμβου όπου μπορεί να μεταφερθεί η hashapp εφαρμογή. Οι χρόνοι αυτοί παρουσιάζονται στην εικόνα 19 και όπως είναι λογικό, δεν έχουν να κάνουν με το μέγεθος του database αλλά με τους ίδιους τους κόμβους.



Σχήμα 19: Χρόνοι για εύρεση επόμενου κόμβου

Ειδικότερα, υπάρχουν οι εξής 3 γενικές περιπτώσεις:

1. Η περίπτωση στην οποία ο κόμβος όπου θα συνδεθεί ο χρήστης έχει αρκετούς πόρους για να μεταφερθεί εκεί η εφαρμογή. Σε αυτήν την περίπτωση, ο master κάνει κατα μέσο όρο 623,3 χιλιοστά του δευτερολέπτου-milliseconds (ms) να υπολογίσει ότι πράγματι υπάρχουν οι διαθέσιμοι πόροι, ανεξαρτήτως του κόμβου όπου συνδέθηκε ο χρήστης.
2. Η περίπτωση στην οποία ο master χρειάζεται να ελέγξει τους πόρους για 2 κόμβους. Εδώ υπάρχουν δύο υποκατηγορίες, που διαφοροποιούνται με βάση την τοποθεσία του hashapp-pod πριν ξεκινήσει η διαδικασία της μεταφοράς και το που θα συνδεθεί ο χρήστης. Αυτές είναι οι εξής:
   - Ο κόμβος που έχει συνδεθεί ο χρήστης να μην έχει αρκετούς πόρους αλλά ο αμέσως πιο κοντινός να έχει.
   - Να μην έχει κανείς άλλος κόμβος αρκετούς πόρους, αλλά ο πιο κοντινός κόμβος σε αυτόν που έχει συνδεθεί ο χρήστης να είναι ο κόμβος που τρέχει ήδη η εφαρμογή.

   Σε κάθε περίπτωση από αυτές, μετρήσαμε ότι ο χρόνος εύρεσης του κόμβου είναι περίπου 1314,2 ms.

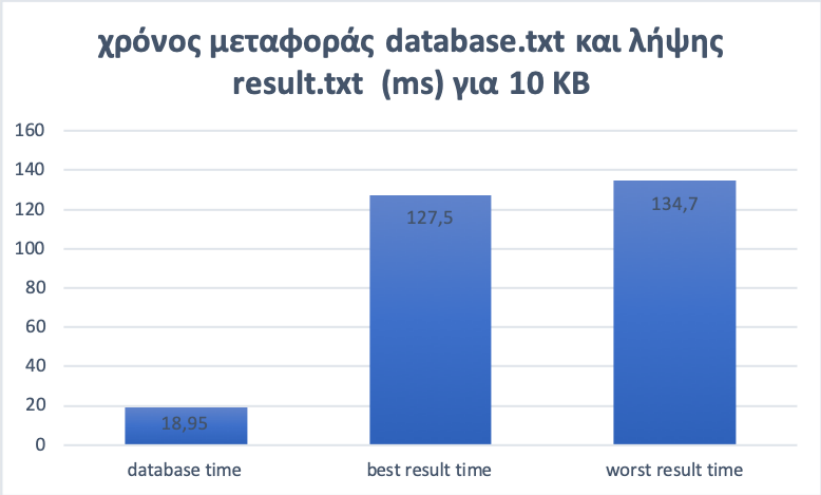
3. Η πιο χρονοβόρα περίπτωση, στην οποία ο master πρέπει να ελέγξει όλους τους άλλους κόμβους, οι οποίοι δε θα έχουν αρκετή μνήμη και έτσι να πρέπει το pod να παραμείνει στον προηγούμενο κόμβο.Ένα παράδειγμα για αυτή την περίπτωση είναι το pod να βρίσκεται στον node2 και ο χρήστης να συνδεθεί στον node3. Ο master, θα ελέγξει πρώτα τον node3, μετά τον node1 και τέλος θα ελέγξει τον node2, όπου και θα παραμείνει το pod. Σε αυτή την περίπτωση, ο master θα χρειαστεί περίπου 1903,4 ms για τους υπολογισμούς.

Μία σημαντική παρατήρηση εδώ είναι το γεγονός πως η τελευταία περίπτωση δεν είναι δυνατή αν το hashapp-pod βρίσκεται στον node1. Αυτό συμβαίνει γιατί ο node1 είναι ο πιο κόντινος κόμβος και στον node2 και στον node3 όποτε θα γίνουν στη χείριστη περίπτωση μόνο 2 έλεγχοι.

5.2.2 Χρόνοι μεταφοράς βάσης δεδομένων και λήψης αποτελεσμάτων

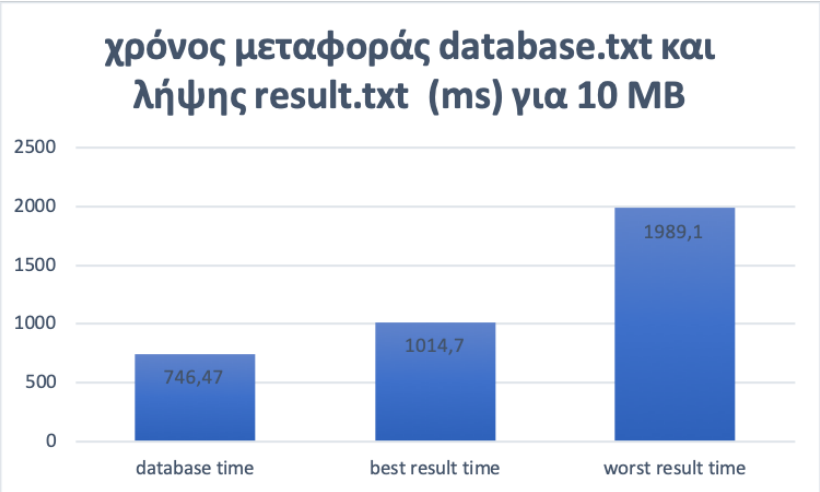
Το επόμενο στάδιο των πειραμάτων μας αποτελεί η μεταφορά της βάσης δεδομένων, για τα 3 διαφορετικά μεγέθη αρχείων. Να αναφέρουμε πως για την καλύτερη οπτικοποίηση των αποτελεσμάτων, συμπεριλαμβάνουμε στα διαγράμματα μας για κάθε μέγεθος και τους χρόνους που θα χρειαστεί για να λάβει ο χρήστης τα αποτελέσματα, παρ΄όλο που πρακτικά είναι το τελευταίο τμήμα του αλγορίθμου. Έτσι, έχουμε τις 3 εξής περιπτώσεις:

1. Για αρχεία μεγέθους 10 KiloBytes:



Σχήμα 20: Χρόνοι για αρχεία 10KB

2. Για αρχεία μεγέθους 10 MegaBytes:



Σχήμα 21: Χρόνοι για αρχεία 10MB

3. Για αρχεία μεγέθους 1 GigaByte:

Σχήμα 22: Χρόνοι για αρχεία 1GB

Αρχικά, παρατηρούμε πως όσο το μέγεθος του αρχείου αυξάνεται, τόσο αυξάνεται και ο χρόνος που χρειάζεται για τη μεταφορά της βάσης από τον ένα κόμβο στον άλλο. Ειδικότερα, για τη βάση των 10 KB απαιτούνται περίπου 19 ms για τη μεταφορά, για τη βάση των 10 MB περίπου 746 ms, δηλαδή σχεδόν ένα δευτερόλεπτο, ενώ για την βάση του 1 GB ο χρόνος αυξάνεται σημαντικά, δηλαδή περίπου στα 110 δευτερόλεπτα. Αυτό φυσικά ήταν αναμενόμενο και λογικό, αλλά δεν επηρεάζει πραγματικά την απόδοση του συστήματος μας. Αυτό συμβαίνει επειδή όσο μεταφέρεται η βάση, το hashapp-pod συνεχίζει να λειτουργεί στον προηγούμενο κόμβο, όπως αναφέραμε και στην ενότητα 4.4.3. Έτσι, κάθε στιγμή ο χρήστης μπορεί να λάβει το result.txt από το frontend-pod αφού το σύστημα μας συνεχίζει να ανταποκρίνεται. Έχουμε δηλαδή μηδενικό χρόνο μη απόκρισης. Βέβαια, αυτοί οι χρόνοι

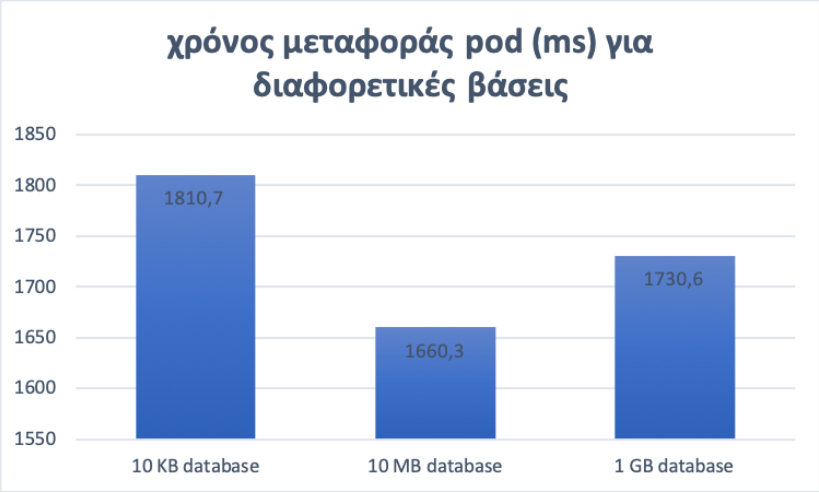
ταυτίζονται με τους χρόνους για τους οποίους η hashapp εκτελείται άσκοπα, αφού ότι έχει παραχθεί στο result.txt σε αυτή τη διάρκεια θα πρέπει να επαναϋπολογιστεί μετά τη μεταφορά.

Όσον αφορά τώρα τη λήψη του αρχείου, μετράμε τους χρόνους για την περίπτωση όπου όλο το database έχει επεξεργαστεί επιτυχώς από την hashapp εφαρμογή, γεγονός που σημαίνει ότι το αρχείο αποτελεσμάτων έχει περίπου ίδιο μέγεθος με αυτό του database, δηλαδή 10 KB, 10 MB και 1 GB. Επιπλέον, μετράμε 2 χρόνους για τη λήψη των αρχείων αυτών. Η καλύτερη περίπτωση (best result time) είναι το hashapp-pod να εκτελείται εκεί όπου είναι συνδεδεμένος ο χρήστης και επομένως, όταν θελήσει να λάβει το result.txt, θα το λάβει απευθείας από τον κόμβο αυτό, όπως αναλύθηκε στην ενότητα 4.4.3. Η χειρότερη περίπτωση (worst result time) είναι το hashapp-pod να τρέχει σε άλλο κόμβο από αυτόν που είναι συνδεδεμένος ο χρήστης. Άρα προκειμένου να λάβει το πιο πρόσφατο result.txt, θα πρέπει το frontend-pod που τρέχει στον κόμβο που συνδέθηκε ο χρήστης να λάβει από τον migration-server το result.txt. Καταλαβαίνουμε λοιπόν ότι σε αυτή την περίπτωση χρειάζεται μία επιπλέον μεταφορά του αρχείου, γεγονός που αιτιολογεί τη διαφορά μεταξύ των 2 αυτών χρόνων. Επίσης, παρατηρούμε ότι αυτή η διαφορά μεγαλώνει ανάλογα με το μέγεθος του αρχείου, το οποίο είναι επίσης λογικό.

Να σημειώσουμε επιπλέον ότι λόγω χρήσης του proxy server έχουμε μία παραπάνω μεταφορά του αρχείου από ότι θα χρειαζόταν κανονικά. Ο χρήστης ζητάει ουσιαστικά από αυτόν το αρχείο και ο proxy συνδέεται στο frontend-pod. Όταν το λάβει από αυτό, το αποθηκεύει σε ένα προσωρινό directory όπως εξηγήσαμε στην ενότητα 4.3 και έπειτα το στέλνει στον χρήστη. Άρα στην καλύτερη περίπτωση γίνονται δύο μεταφορές αντί για μία, ενώ στην χειρότερη γίνονται τρεις αντί για δύο, με τους χρόνους να τροποποιούνται ανάλογα.

5.2.3 Χρόνος μεταφοράς της κάψουλας

Αφού υπολογίσαμε το χρόνο εύρεσης του νέου κόμβου και το χρόνο που θα χρειαστεί για να μεταφερθεί η βάση και να σταλεί το αποτέλεσμα, μένει μόνο να μετρήσουμε το χρόνο που θα χρειαστεί για να μεταφερθεί σε κάθε περίπτωση το pod από τον ένα κόμβο στον άλλο, ώστε να συνεχίσει η λειτουργία της εφαρμογής. Στην εικόνα 23, παρουσιάζεται ο χρόνος που χρειάζεται για να επανεκκινήσει ο Κυβερνήτης στον καινούριο κόμβο το pod για κάθε μέγεθος αρχείων.



Σχήμα 23: Χρόνος για μεταφορά του pod

Παρατηρούμε πως ο χρόνος που χρειάζεται σε κάθε περίπτωση είναι περίπου ο ίδιος και ίσος με σχεδόν 2 δευτερόλεπτα, δηλαδή περίπου 1740 ms κατά μέσο όρο. Καταλαβαίνουμε λοιπόν ότι το μέγεθος των αρχείων δεν επηρεάζει το χρόνο για τη μεταφορά του pod. Αυτό είναι αναμενόμενο, καθώς η μεταφορά αυτή έχει να κάνει αποκλειστικά με τη λειτουργία του Κυβερνήτη και με το control plane του, που προσπαθεί να φέρει την πραγματική κατάσταση στη νέα επιθυμητή κατάσταση. Αυτός ο χρόνος των δύο δευτερολέπτων είναι επίσης ο χρόνος για τον οποίο η hashapp εφαρμογή μας

δεν αποκρίνεται και επειδή σχετίζεται με τον ίδιο τον Κυβερνήτη, δε μπορούμε δυστυχώς να τον βελτιστοποιήσουμε.

5.3 Συνολικοί χρόνοι

Εν κατακλείδι, διεξάγαμε συνολικά τα εξής πειράματα. Όσον αφορά τη διαδικασία της μεταφοράς, χρονομετρήσαμε όλες τις πιθανές περιπτώσεις για την εύρεση του επόμενου κόμβου, τη μεταφορά των αρχείων που σχετίζονται με την εφαρμογή για τρία διαφορετικά μεγέθη αρχείων και τέλος, τη μεταφορά του ίδιου του pod. Επομένως, μπορούμε τώρα να υπολογίσουμε τον συνολικό χρόνο για τη διαδικασία της μεταφοράς για κάθε περίπτωση. Αυτοί οι χρόνοι φαίνονται στον πίνακα 3. Διευκρινίζουμε πως για τη μεταφορά του pod, βάλαμε ως χρόνο τον μέσο όρο των χρόνων για τα 3 είδη αρχείων που φαίνονται στην εικόνα 23.Παρατηρούμε ότι για μικρά αρχεία το ποσοστό του χρόνου που αντιστοιχεί στη μη απόκριση της εφαρμογής είναι αρκετά μεγάλο, ενώ για μεγάλα αρχεία το μεγαλύτερο ποσοστό του χρόνου το καταλαμβάνει η μεταφορά της βάσης δεδομένων.

file	file size	database transfer time	time for pod	time for new node	total time
database.txt	10 KB	18.95 ms	1740 ms	623.3 ms	2.382 sec
				1314.2 ms	3.072 sec
				1903.4 ms	3.662 sec
	10 MB	746.47 ms		623.3 ms	3.109 sec
				1314.2 ms	3.800 sec
				1903.4 ms	4.389 sec
	1 GB	108.205 sec		623.3 ms	110.568 sec
				1314.2 ms	111.259 sec
				1903.4 ms	111.848 sec

Πίνακας 3: Χρόνοι για λήψη αρχείου αποτελεσμάτων

Όσον αφορά τη λήψη αποτελεσμάτων, παρουσιάζονται στον πίνακα 4 συνολικά όλες οι πιθανές περιπτώσεις που αναφέρθηκαν στην ενότητα 5.2.2.

getting result.txt from	10 KB	10 MB	1 GB
node where client connects	127.5 ms	1014.7 ms	102 sec
different node	134.7 ms	1989.1 ms	185 sec

Πίνακας 4: Συνολικοί χρόνοι για πλήρη μεταφορά stateful containerized app

Κεφάλαιο 6

Συμπεράσματα και Μελλοντική Εργασία

6.1 Σύνοψη συμπερασμάτων αξιολόγησης

Στα πλαίσια αυτής της διπλωματικής, υλοποιήσαμε ένα σύστημα για τη μεταφορά stateful containerized εφαρμογών στο edge computing με τη χρήση του Κυβερνήτη. Κάθε φορά που ένας χρήστης μετακινείται, η εφαρμογή θα μεταφέρεται στο μηχάνημα που ανήκει στο edge και βρίσκεται πίσω από το access point που έχει συνδεθεί ο χρήστης. Προς επίτευξη του στόχου αυτού, δημιουργήσαμε δικές μας εφαρμογές, μία αυτοματοποιημένη διαδικασία-script στον master του Κυβερνήτη καθώς και δικούς μας εξυπηρετητές που λειτουργούν ως διαμεσολαβητές μεταξύ χρήστη και worker nodes του Κυβερνήτη. Με τη χρήση όλων αυτών, υλοποιούμε έναν αλγόριθμο ο οποίος εκτελεί τις εξής λειτουργίες. Αρχικά, εντοπίζει που έχει συνδεθεί ο χρήστης. Έπειτα, υπολογίζει αν ο κόμβος που βρίσκεται πίσω από το access point που συνδέθηκε ο χρήστης έχει τους απαραίτητους διαθέσιμους πόρους προκειμένου να μεταφερθεί εκεί η εφαρμογή. Αν τους έχει, τότε μεταφέρονται εκεί η ίδια η εφαρμογή καθώς και τα απαραίτητα για τη λειτουργία της αρχεία. Αν οι πόροι που διαθέτει δεν είναι αρκετοί, τότε βρίσκεται ο πιο κοντινός σε αυτόν κόμβος που έχει τους απαραίτητους πόρους και συνεχίζει εκεί η εφαρμογή. Τέλος, αν δεν υπάρχει τέτοιος κόμβος, η εφαρμογή παραμένει στον κόμβο που βρίσκεται ήδη. Στη συνέχεια της διπλωματικής, διεξήχθησαν πειράματα για να κρίνουμε την αποδοτικότητα αυτού του αλγορίθμου.Συμπαιρένουμε λοιπόν πως ο αλγόριθμος για τη μεταφορά των stateful εφαρμογών είναι αρκετά αποδοτικός, ακόμα και για αρχεία μεγέθους 1 GB. Επιπλέον, ακόμα και αν οι χρόνοι για τη μεταφορά αρχείων αυτής της κλίμακας προσεγγίζουν μερικά λεπτά, ο χρόνος μη απόκρισης της εφαρμογής είναι σταθερός στα 2 δεύτερόλεπτα, με τον χρήστη να έχει τη δυνατότητα να λάβει οποιαδήποτε άλλη στιγμή τα αποτελέσματα του. Ακόμα, πρέπει να αναφέρουμε ότι ο αλγόριθμος μας δεν προσθέτει επιπλέον καθυστερήσεις στο σύστημα προκειμένου να εκτελεστεί ορθά, δηλαδή οι χρόνοι που υπολογίσαμε είναι οι βέλτιστοι δυνατοί για τα μεγέθη αρχείων τα οποία χρησιμοποιήσαμε. Τέλος, το μόνο σημείο που δεν αποκρίνεται η εφαρμογή κατά τη διάρκεια του αλγορίθμου οφείλεται αποκλειστικά στο control plane του Κυβερνήτη, που σημαίνει ότι δε μπορούμε εμείς να βελτιστοποιήσουμε περαιτέρω τον αλγόριθμο μας.
Εν κατακλείδι, υλοποιήσαμε αποδοτικά τη μεταφορά stateful εφαρμογών σε container χωρίς τη χρήση έτοιμων εξωτερικών εργαλείων παρά μόνο με τη χρήση δικών μας απλών εφαρμογών και των δυνατοτήτων που μας προσφέρει το api του Κυβερνήτη.

6.2 Μελλοντική Εργασία

Το έργο της παρούσας διπλωματικής εργασίας μπορεί να επεκταθεί και στις εξής ακόλουθες ενδεικτικές, αλλά συγκεκριμένες, κατευθύνσεις:

- Να προσπαθήσουμε να ενσωματώσουμε το εργαλείο CRIU [38] στον Κυβερνήτη, που προς το παρόν δε το υποστηρίζει, προκειμένου να αποθηκεύουμε με πιο μεθοδευμένο τρόπο την κατάσταση της κάθε εφαρμογής.

- Να αξιοποιήσουμε όσο το δυνατόν καλύτερα το api του Κυβερνήτη, το οποίο εξελίσσεται συνεχώς, προκειμένου να κάνουμε το σύστημα μας κλιμακώσιμο, δηλαδή να μπορεί να εξυπηρε-

τήσει πολλούς χρήστες. Επιπλέον, να εκμεταλλευτούμε τις εξελίξεις αυτές ώστε να προσπαθήσουμε να βελτιώσουμε τους χρόνους του αλγορίθμου μας.

- Να πειραματιστούμε με πραγματικές stateful εφαρμογές και όχι με εφαρμογές δικής μας δημιουργίας (hashapp). Έτσι, θα μπορέσουμε να αξιοποιήσουμε τις μεθόδους που προσφέρουν για να βελτιώσουμε το σύστημά μας, παραδείγματος χάριν να αποθηκεύουμε και να μεταφέρουμε και τα αποτελέσματα που παράγονται κατά τη διάρκεια της μεταφοράς ώστε να μη χρειάζεται να επαναϋπολογίζονται.

Βιβλιογραφία

[1] Lu Tan and Neng Wang, "Future internet: The Internet of Things," in *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, Chengdu, China: IEEE, Aug. 2010, pp. V5–376–V5–380, isbn: 978-1-4244-6539-2. doi: 10.1109/ICACTE.2010.5579543. [Online]. Available: http://ieeexplore.ieee.org/document/5579543/ (visited on 01/28/2020).

[2] J.-c. Zhao, J.-f. Zhang, Y. Feng, and J.-x. Guo, "The study and application of the IOT technology in agriculture," in *2010 3rd International Conference on Computer Science and Information Technology*, vol. 2, IEEE, 2010, pp. 462–465.

[3] P. Gope and T. Hwang, "BSN-Care: A secure IoT-based modern healthcare system using body sensor network," *IEEE sensors journal*, vol. 16, no. 5, pp. 1368–1376, 2015.

[4] S. Rahimi Moosavi, T. Nguyen Gia, A.-M. Rahmani, E. Nigussie, S. Virtanen, J. Isoaho, and H. Tenhunen, "SEA: a secure and efficient authentication and authorization architecture for IoT-based healthcare using smart gateways," in *Procedia Computer Science*, vol. 52, Elsevier, 2015, pp. 452–459.

[5] M. Elhoseny, G. Ramírez-González, O. M. Abu-Elnasr, S. A. Shawkat, N. Arunkumar, and A. Farouk, "Secure medical data transmission model for IoT-based healthcare systems," *Ieee Access*, vol. 6, pp. 20 596–20 608, 2018.

[6] B. L. R. Stojkoska and K. V. Trivodaliev, "A review of Internet of Things for smart home: Challenges and solutions," *Journal of Cleaner Production*, vol. 140, pp. 1454–1464, 2017.

[7] M. Soliman, T. Abiodun, T. Hamouda, J. Zhou, and C.-H. Lung, "Smart home: Integrating internet of things with web services and cloud computing," in *2013 IEEE 5th international conference on cloud computing technology and science*, vol. 2, IEEE, 2013, pp. 317–320.

[8] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds," in *2014 IEEE world forum on internet of things (WF-IoT)*, IEEE, 2014, pp. 241–246.

[9] S. P. Mohanty, U. Choppali, and E. Kougianos, "Everything you wanted to know about smart cities: The internet of things is the backbone," *IEEE Consumer Electronics Magazine*, vol. 5, no. 3, pp. 60–70, 2016.

[10] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things journal*, vol. 1, no. 1, pp. 22–32, 2014.

[11] N. Kalatzis, M. Avgeris, D. Dechouniotis, K. Papadakis-Vlachopapadopoulos, I. Roussaki, and S. Papavassiliou, "Edge computing in IoT ecosystems for UAV-enabled early fire detection," in *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*, IEEE, 2018, pp. 106–114.

[12] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *CISCO white paper*, vol. 1, no. 2011, pp. 1–11, 2011.

[13] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[14] M. Avgeris, D. Dechouniotis, N. Athanasopoulos, and S. Papavassiliou, "Adaptive Resource Allocation for Computation Offloading: A Control-Theoretic Approach," English, *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 2, p. 23, Apr. 2019, issn: 1533-5399. doi: 10.1145/3284553. [Online]. Available: https://pure.qub.ac.uk/en/publications/adaptive-resource-allocation-for-computation-offloading-a-control (visited on 02/04/2020).

[15] F. Giust, G. Verin, K. Antevski, J. Chou, Y. Fang, W. Featherstone, F. Fontes, D. Frydman, A. Li, and A. Manzalini, "MEC deployments in 4G and evolution towards 5G," *ETSI White Paper*, vol. 24, pp. 1–24, 2018.

[16] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," en, *Journal of Systems Architecture*, vol. 98, pp. 289–330, Sep. 2019, issn: 13837621. doi: 10.1016/j.sysarc.2019.02.009. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1383762118306349 (visited on 01/29/2020).

[17] *What is a Container?* en. [Online]. Available: https://www.docker.com/resources/what-container (visited on 01/28/2020).

[18] *Are Containers Replacing Virtual Machines?* en-US, Aug. 2018. [Online]. Available: https://www.docker.com/blog/containers-replacing-virtual-machines/ (visited on 01/29/2020).

[19] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An introduction to docker and analysis of its performance," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 17, no. 3, p. 228, 2017.

[20] D. Jaramillo, D. V. Nguyen, and R. Smart, "Leveraging microservices architecture by using Docker technology," in *SoutheastCon 2016*, IEEE, 2016, pp. 1–5.

[21] *Production-Grade Container Orchestration - Kubernetes*. [Online]. Available: https://kubernetes.io/ (visited on 02/03/2020).

[22] *K3s: Lightweight Kubernetes*. [Online]. Available: https://k3s.io/ (visited on 01/29/2020).

[23] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches: A survey of mobile cloud computing," en, *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, Dec. 2013, issn: 15308669. doi: 10.1002/wcm.1203. [Online]. Available: http://doi.wiley.com/10.1002/wcm.1203 (visited on 01/28/2020).

[24] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[25] J. S. Preden, K. Tammemäe, A. Jantsch, M. Leier, A. Riid, and E. Calis, "The benefits of self-awareness and attention in fog and mist computing," *Computer*, vol. 48, no. 7, pp. 37–45, 2015.

[26] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[27] A. Al-Shuwaili and O. Simeone, "Energy-Efficient Resource Allocation for Mobile Edge Computing-Based Augmented Reality Applications," *IEEE Wireless Communications Letters*, vol. 6, no. 3, pp. 398–401, Jun. 2017, issn: 2162-2337, 2162-2345. doi: 10.1109/LWC.2017.2696539. [Online]. Available: http://ieeexplore.ieee.org/document/7906521/ (visited on 02/02/2020).

[28] K. Bilal and A. Erbad, "Edge computing for interactive media and video streaming," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, IEEE, 2017, pp. 68–73.

[29] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the Internet of Things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.

[30] C. Puliafito, E. Mingozzi, and G. Anastasi, "Fog computing for the internet of mobile things: issues and challenges," in *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*, IEEE, 2017, pp. 1–6.

[31] M. Nelson, B.-H. Lim, and G. Hutchins, "Fast Transparent Migration for Virtual Machines," en, p. 4, 2005.

[32] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan, "Live virtual machine migration with adaptive, memory compression," in *2009 IEEE International Conference on Cluster Computing and Workshops*, IEEE, 2009, pp. 1–10.

[33] C. Dupont, R. Giaffreda, and L. Capra, "Edge computing in IoT context: Horizontal and vertical Linux container migration," in *2017 Global Internet of Things Summit (GIoTS)*, IEEE, 2017, pp. 1–4.

[34] D. Pizzolli, G. Cossu, D. Santoro, L. Capra, C. Dupont, D. Charalampos, F. De Pellegrini, F. Antonelli, and S. Cretti, "Cloud4IoT: A Heterogeneous, Distributed and Autonomic Cloud Platform for the IoT," in *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Luxembourg, Luxembourg: IEEE, Dec. 2016, pp. 476–479, isbn: 978-1-5090-1445-3. doi: 10.1109/CloudCom.2016.0082. [Online]. Available: http://ieeexplore.ieee.org/document/7830723/ (visited on 02/03/2020).

[35] *Assigning Pods to Nodes*, en. [Online]. Available: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/ (visited on 02/03/2020).

[36] C. Puliafito, E. Mingozzi, C. Vallati, F. Longo, and G. Merlino, "Companion fog computing: Supporting things mobility through container migration at the edge," in *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*, IEEE, 2018, pp. 97–105.

[37] *Empowering App Development for Developers*, en. [Online]. Available: https://www.docker.com/ (visited on 02/03/2020).

[38] *CRIU*. [Online]. Available: https://www.criu.org/Main_Page (visited on 02/03/2020).

[39] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, "Voyager: Complete container state migration," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 2017, pp. 2137–2142.

[40] *Concepts*, en. [Online]. Available: https://kubernetes.io/docs/concepts/ (visited on 02/06/2020).

[41] *The Official YAML Web Site*. [Online]. Available: https://yaml.org/ (visited on 02/06/2020).

[42] *etcd*, en-us. [Online]. Available: https://etcd.io/ (visited on 02/06/2020).

[43] *Kubernetes Components*, en. [Online]. Available: https://kubernetes.io/docs/concepts/overview/components/ (visited on 02/06/2020).

[44] *Kubernetes Scheduler*, en. [Online]. Available: https://kubernetes.io/docs/concepts/scheduling/kube-scheduler/ (visited on 02/06/2020).

[45] *Kubernetes 101 - IBM Cloud Architecture Center*, en. [Online]. Available: https://www.ibm.com/cloud/architecture/content/course/kubernetes-101/volumes-part2/ (visited on 02/06/2020).

[46] *Pods*, en. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/pods/pod/ (visited on 02/07/2020).

[47] *containerd*, en-us. [Online]. Available: https://containerd.io/ (visited on 02/08/2020).

[48] *cri-o*. [Online]. Available: https://cri-o.io/ (visited on 02/08/2020).

[49] *kubernetes-retired/rktlet*, original-date: 2016-08-10T20:15:40Z, Jan. 2020. [Online]. Available: https://github.com/kubernetes-retired/rktlet (visited on 02/08/2020).

[50] *Understanding Kubernetes Objects*, en. [Online]. Available: https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/ (visited on 02/08/2020).

[51] *Deployments*, en. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/ (visited on 02/07/2020).

[52] *ReplicationController*, en. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller/ (visited on 02/08/2020).

[53] *Container Environment Variables*, en. [Online]. Available: https://kubernetes.io/docs/concepts/containers/container-environment-variables/ (visited on 02/07/2020).

[54] *Managing Compute Resources for Containers*, en. [Online]. Available: https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/ (visited on 02/06/2020).

[55] *DaemonSet*, en. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/ (visited on 02/07/2020).

[56] *Volumes*, en. [Online]. Available: https://kubernetes.io/docs/concepts/storage/volumes/ (visited on 02/06/2020).

[57] *Persistent Volumes*, en. [Online]. Available: https://kubernetes.io/docs/concepts/storage/persistent-volumes/ (visited on 02/06/2020).

[58] *Dynamic Volume Provisioning*, en. [Online]. Available: https://kubernetes.io/docs/concepts/storage/dynamic-provisioning/ (visited on 02/06/2020).

[59] *Cluster Networking*, en. [Online]. Available: https://kubernetes.io/docs/concepts/cluster-administration/networking/ (visited on 02/06/2020).

[60] *coreos/flannel*, original-date: 2014-07-10T17:45:29Z, Feb. 2020. [Online]. Available: https://github.com/coreos/flannel (visited on 02/08/2020).

[61] *Service*, en. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/service/ (visited on 02/10/2020).

[62] *Kubernetes 1.14: Local Persistent Volumes GA*, en. [Online]. Available: https://kubernetes.io/blog/2019/04/04/kubernetes-1.14-local-persistent-volumes-ga/ (visited on 02/15/2020).

[63] *Docker Hub*. [Online]. Available: https://hub.docker.com/ (visited on 02/12/2020).

[64] *Welcome to Flask — Flask Documentation (1.1.x)*. [Online]. Available: https://flask.palletsprojects.com/en/1.1.x/ (visited on 02/08/2020).

[65] *Update API Objects in Place Using kubectl patch*, en. [Online]. Available: https://kubernetes.io/docs/tasks/run-application/update-api-object-kubectl-patch/ (visited on 02/15/2020).