ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ & ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αποθήκευση και Διαχείριση Μεγάλων Δεδομένων σε Κατανεμημένα Συστήματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ευστράτιος Καρυπιάδης

Επιβλέπων: Θεοδώρα Βαρβαρίγου

Καθηγήτρια Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2022

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ & ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αποθήκευση και Διαχείριση Μεγάλων Δεδομένων σε Κατανεμημένα Συστήματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ευστράτιος Καρυπιάδης

Επιβλέπων: Θεοδώρα Βαρβαρίγου

Καθηγήτρια Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την Φεβρουαρίου 2022.

.....	.....	.....
Θεοδώρα Βαρβαρίγου	Εμμανουήλ Βαρβαρίγος	Συμεών Παπαβασιλείου
Καθηγήτρια Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π

Αθήνα, Φεβρουάριος 2022

.....

Ευστράτιος Καρυπιάδης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π

Copyright © Ευστράτιος Καρυπιάδης, 2022

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ' ολοκλήρου ή τμήμα-τος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς το συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν το συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Με τη συνεχή εξέλιξη της τεχνολογίας και την όλο και περισσότερη ψηφιοποίηση της καθημερινότητας έχει αυξηθεί ο παραγόμενος όγκος δεδομένων. Εκτιμάται ότι τα δεδομένα, οποιασδήπτε μορφής και προέλευσης, αποτελούν αγαθό αξίας, δεδομένης της υποστήριξης που μπορούν να προσφέρουν στη λήψη αποφάσεων, στην εκπαίδευση συστημάτων τεχνικής νοημοσύνης και στην εξέλιξη της ήδη υπάρχουσας γνώσης. Δεν είναι τυχαία άλλωστε η φράση: *τα δεδομένα είναι το νέο πετρέλαιο*. Σε μία εποχή, λοιπόν, όπου η αποδοτική αποθήκευση και διαχείριση Μεγάλων Δεδομένων μπορεί να συμβάλλει στην αποτελεσματική αντιμετώπιση ποικίλων προβλημάτων και προκλήσεων είναι επιτακτική η ανάγκη συστημάτων που μπορούν να ανταποκριθούν σε αυτές τις απαιτήσεις.

Σκοπός της διπλωματικής είναι να προτείνει ένα δυναμικό σύστημα αποθήκευσης και διαχείρισης μεγάλου όγκου δεδομένων, το οποίο μεταβάλλει την κλίμακά του, αυτόματα, προκειμένου να προσαρμοστεί στο φόρτο δικτύου προς εξυπηρέτηση εφαρμογών υψηλής διαθεσιμότητας και χαμηλών αποκρίσεων, μέσω της αποδοτικής εκμετάλλευσης των διαθέσιμων πόρων. Για τη δημιουργία του συστήματος χρησιμοποιήθηκε το οικοσύστημα αυτοματοποιημένης διαχείρισης containers Kubernetes, στο οποίο φιλοξενείται βάση μη σχεσιακού σχήματος δεδομένων. Πιο συγκεκριμένα, γίνεται χρήση των λειτουργιών replication και sharding της MongoDB προς αυξομείωση της κλίμακας της βάσης αποθήκευσης. Η μεταβολή της κλίμακας επιτυγχάνεται μέσω πράκτορα-παρατηρητή που αναπτύχθηκε, ο οποίος μεταβάλλει τα αντίγραφα (replicas) της βάσης, όταν ανιχνευθεί υπερβολικός φόρτος στους εξυπηρετητές της, με απώτερο σκοπό τη βελτίωση της απόκρισης του συνολικού συστήματος.

Λέξεις Κλειδιά

Kubernetes, MongoDB, big data, replication, sharding, auto-scaling

Abstract

The constant advancement of technology in combination with the digitalization of everyday life has led to the drastic increase of produced data. This data, irrespective of volume and source, is considered to be an asset of value considering the way it can affect decision making, training of advanced artificial intelligent systems and advancement of already acquired knowledge. It is no wonder that technology leaders claim that Big Data is the new oil. As a result, effective systems for storing and managing data of this volume is needed, in order for the humankind to capitalize on this value.

The goal of this thesis is to propose dynamic system for storing and managing Big Data, which can automatically scale its resources according to network load, aiming at optimizing resource utilization and decreasing response times for applications in need of high availability. Kubernetes was used for the purposes of this system, where a sharded MongoDB instance is deployed, along with a custom agent, who can trigger up- or down-scaling according to cluster workload.

Keywords

Kubernetes, MongoDB, big data, replication, sharding, auto-scaling

... στους γονείς μου

Ευχαριστίες

Θα ήθελα αρχικά να ευχαριστήσω την καθηγήτρια κ. Θεοδώρα Βαρβαρίγου για την εμπιστοσύνη που μου έδειξε με την ανάθεση ενός θέματος σχετικά με μία καινούρια για εμένα τεχνολογία. Οφείλω, επίσης, ένα ειλικρινές ευχαριστώ στους υποψήφιους διδάκτορες Αχιλλέα Μαρινάκη και Αναστάσιο Νικολακόπουλο για τον τρόπο που με υποστήριξαν και σεβάστηκαν τις ιδέες μου κατά την εκπόνηση αυτής της εργασίας.

Τέλος, αυτή η διπλωματική αποτελεί ένα ευχαριστώ στους γονείς μου που έδωσαν τα πάντα για να βρίσκομαι σε θέση να ολοκληρώσω αυτόν τον κύκλο των σπουδών μου. Ευχαριστώ που με στηρίζετε ανελλιπώς, ώστε να ανακαλύπτω καθημερινά τα όριά μου. Ευχαριστώ για την υπομονή που δείξατε τις στιγμές που η δικιά μου είχε εξαντληθεί. Ευχαριστώ που είστε το κύριο ερέθισμα που με έκανε τον άνθρωπο που είμαι σήμερα.

Πίνακας Περιεχομένων

Πίνακας Σχημάτων

1

Εισαγωγή

Είναι γεγονός πως η συνεχής εξέλιξη της τεχνολογίας και η διεύρυνση της χρήση της σε όλο και περισσότερα κομμάτια της καθημερινότητας έχει οδηγήσει στην αύξηση των παραγόμενων δεδομένων, γνωστά ως Μεγάλα Δεδομένα. Κύριες πηγές είναι ο χώρος της οικονομίας, των επιχειρήσεων, της επιστήμης, ακόμα και η προσωπική ζωή. Αν και δεν υπάρχει κάποιος καθολικά αποδεκτός ορισμός, τα Μεγάλα Δεδομένα θα μπορούσαν να οριστούν ως εκείνα τα σύνολα δεδομένων με μέγεθος αρκετά μεγάλο ώστε να μη μπορούν να αποθηκευτούν και να διαχειριστούν εύκολα από τα υπάρχοντα υπολογιστικά συστήματα και εργαλεία λογισμικού.

Βασικά χαρακτηριστικά που ξεχωρίζουν τα Μεγάλα Δεδομένα από άλλα μεγέθη πληροφορίας φέρουν τη συντομογραφία των *5Vs*, περιγράφοντας τα αρχικά των λέξεων volume, variety, velocity, value και veracity δηλαδή μέγεθος, ποικιλομορφία, ταχύτητα, αξία και εγκυρότητα. Ο όρος του μεγέθους αναφέρεται στο μέγεθος των δεδομένων που χρειάζεται να αποθηκευτεί και διαχειριστεί προκειμένου να προκύψει χρήσιμη πληροφορία. Οι παραπάνω ενέργειες απαιτούν υψηλή υπολογιστική ισχύ και πόρους, στοιχεία που καλύπτονται είτε από την άμεση αύξηση των χαρακτηριστικών ενός υπολογιστικού μηχανήματος, είτε από την επιστράτευση παραπάνω μηχανημάτων. Ο όρος της ταχύτητας αναφέρεται αφενός στην ταχύτητα παραγωγής των δεδομένων και αφετέρου στο πόσο γρήγορα τα δεδομένα μεταφέρονται μεταξύ δύο σημείων. Το τελευταίο επηρεάζεται από τους περιορισμούς της σύγχρονης τεχνολογίας δικτύων, με αποτέλεσμα δεδομένα να πρέπει να μεταφέρονται σε υποκομμάτια του αρχικού συνόλου. Το χαρακτηριστικό της ποικιλομορφίας εκφράζει τη διαφορετική μορφοποίηση των δεδομένων, συνδέεται άμεσα με το πλήθος των πηγών παραγωγής, ενώ καθιστά αναγκαία την προεπεξεργασία τους. Τέλος, οι έννοιες της αξίας και της εγκυρότητας αντιστοιχίζονται στο κέρδος που προκύπτει από την αποτελεσματική εκμετάλλευση της υπάρχουσας πληροφορίας και στην ποιότητα που αυτή φέρει. Γίνεται προφανές ότι το τελευταίο είναι δύσκολο να εξασφαλιστεί στο χώρο του διαδικτύου, στοιχείο που καθιστά αναγκαίο τον έλεγχο και τη διασταύρωση των εισερχόμενων δεδομένων πριν αυτά χρησιμοποιηθούν.

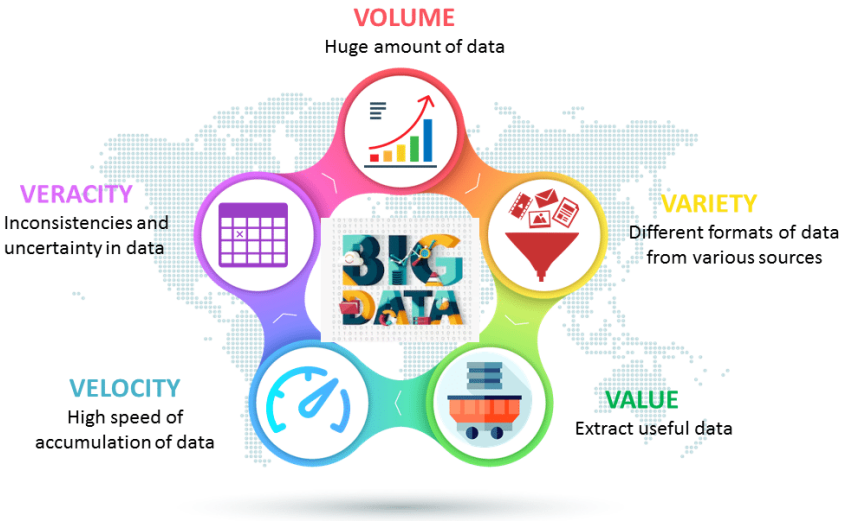
Τα παραπάνω χαρακτηριστικά παρουσιάζουν συγκεκριμένες προκλήσεις στον επιστημονικό και επιχειρησιακό χώρο, οι οποίες πρέπει να αντιμετωπιστούν, για να εκμεταλλευθούν τα πλεονεκτήματα που συνοδεύουν τα Μεγάλα Δεδομένα. Χαρακτηριστική, σχετικά, είναι η φράση: *δε μπορείς να διαχειριστείς ό,τι δε μπορείς να μετρήσεις*. Σε μια προσπάθεια της επιστημονικής κοινότητας να ανταποκριθεί σε αυτές τις απαιτήσεις, έχουν προταθεί ανά τα χρόνια εξειδικεύμενοι τρόποι αποθήκευσης και διαχείρισης. Δύο σχετικά παραδείγματα αποτελούν τα κατανεμημένα συστήματα αποθήκευσης και οι μη σχεσιακές βάσεις δεδομένων. Το πρώτο προβλέπει τη σύνθεση πολλαπλών υπολογιστικών μηχανημάτων σε συστάδες αποθήκευσης, επεξεργασίας και διαχείρισης, σε μία προσπάθεια να καλυφθούν οι αυξημένες απαιτήσεις μέσω της οριζόντιας κλιμάκωσης των διαθέσιμων πόρων. Το δεύτερο σχετίζεται με την αφαίρεση του σχήματος από τα δεδομένα, στοιχείο που διευκολύνει τις λειτουργίες της αποθήκευσης και της αναζήτησης λόγω της άρσης των σχετικών ελέγχων εγκυρότητας που συνοδεύουν άλλα σχεσιακά συστήματα και λύσεις.

1.1 Αντικείμενο Διπλωματικής Εργασίας

Η παρούσα εργασία λαμβάνει υπόψιν τη σπουδαιότητα των Μεγάλων Δεδομένων στο σύγχρονο τεχνολογικό κόσμο, δεδομένης της επίδρασής τους στην επίλυση κοινωνικών, κλιματικών, υγειονομικών και οικονομικών προκλήσεων. Στο πλαίσιο αυτό καλείται να αντιμετωπίσει το πρόβλημα βέλτιστης διαχείρισης υπολογιστικών πόρων και προτείνει ένα κατανεμημένο σύστημα μη σχεσιακής βάσης δεδομένων το οποίο αυτόματα μεταβάλλει την κλίμακά του ανάλογα με το φόρτο δικτύου, προς αποδοτικότερη εξυπηρέτηση των εισερχόμενων αιτημάτων.

1.2 Δομή Διπλωματικής Εργασίας

Η παρούσα εργασία χωρίζεται σε θεωρητικό και πρακτικό μέρος. Το θεωρητικό κομμάτι αποτελείται τρία κεφάλαια τα οποία προσφέρουν μία θεωρητική επισκόπηση των τεχνολογιών που χρησιμοποιήθηκαν στο πρακτικό μέρος. Πιο συγκεκριμένα, το κεφάλαιο 2 πραγματεύεται την τεχνολογία των containers, αρχιτεκτονική εκτέλεσης εφαρμογών λογισμικού. Το κεφάλαιο 3 περιγράφει το οικοσύστημα του Kubernetes, έναν αυτοματοποιημένο περιβάλλον στο οποίο εκτελούνται containerized εφαρμογές, ενώ το κεφάλαιο 4 παρουσιάζονται οι μη σχεσιακές βάσεις δεδομένων και οι διαφορετικές κατηγορίες αυτών. Το πρακτικό κομμάτι αποτελείται από το κεφάλαιο 5, το οποίο αναλύει την αρχιτεκτονική και το λογισμικό που αναπτύχθηκε, ενώ παράλληλα συγκρίνεται το προκύπτων σύστημα με άλλες, μη αυτοματοποιημένες λύσεις στο κεφάλαιο 6.



Σχήμα 1: Τα 5 V's των Μεγάλων Δεδομένων

Θεωρητικό Μέρος

2

Εικονικοποίηση & Containerization

Οι τεχνικές της εικονικοποίησης και του containerization είναι οι πλέον κυρίαρχες στρατηγικές στο χώρο του υπολογιστικού νέφους (cloud computing) για την εξυπηρέτηση ευέλικτων εφαρμογών. Και οι δύο αυτές λύσεις προβλέπουν την αυξομείωση της κλίμακας του υλικού που χρησιμοποιείται με στόχο την εξυπηρέτηση του εισερχόμενου φόρτου εργασίας με τον πιο αποδοτικό και οικονομικό τρόπο.

2.1 Εικονικοποίηση

Η εικονικοποίηση χρησιμοποιεί κομμάτι εξειδικευμένου λογισμικού που επιτρέπει στους πόρους του επεξεργαστή, της μνήμης και της μονάδας αποθήκευσης ενός υπολογιστή να μπορούν να διαχωριστούν σε πολλαπλά εικονικά μηχανήματα (VMs). Κάθε εικονικό μηχάνημα εκτελεί το δικό του λειτουργικό σύστημα και συμπεριφέρεται ως ανεξάρτητη μονάδα, παρόλο που εκτελείται σε ένα, μόνο, υποκομμάτι του πραγματικού υλικού. Γίνεται προφανές, ότι μία τέτοια λειτουργία προωθεί την αποδοτικότερη εκμετάλλευση του υπολογιστικού υλικού, καθώς ελαττώνεται η περίοδος αδράνειας των υπολογιστικών πόρων.

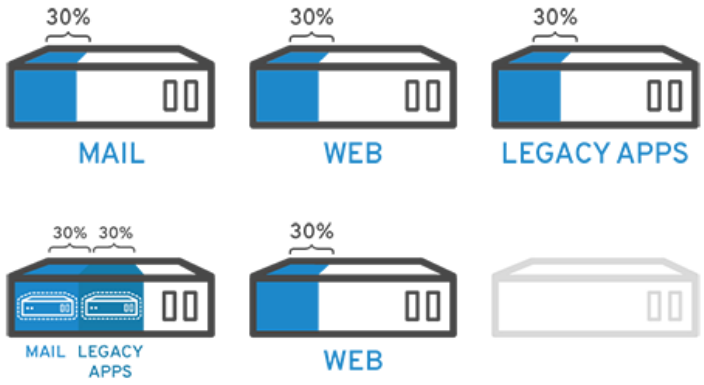
Στον σύγχρονο τεχνολογικό κόσμο η τεχνική της εικονικοποίησης είναι διαδεδομένη λόγω του χώρου του υπολογιστικού νέφους, όπου χρήστες μπορούν να καταναλώνουν μόνο τους υπολογιστικούς πόρους που χρειάζονται και να αυξομειώσουν την κλίμακα των υποδομών τους με εύκολο και οικονομικό τρόπο.

2.1.1 Πλεονεκτήματα

Ένα βασικό πλεονέκτημα της εικονικοποίησης είναι η αποδοτική διαχείριση πόρων. Πριν την εικονικοποίηση, κάθε εξυπηρετητής εφαρμογών χρειαζόταν το δικό του αποκλειστικό επεξεργαστή, στοιχείο που συνεπάγεται την αγορά και διαμόρφωση ξεχωριστών server, για κάθε διαφορετική εφαρμογή. Αναπόφευκτα κάθε μηχάνημα θα χρησιμοποιούσε μόνο ένα υποσύνολο των πόρων του. Αντιθέτως, μέσω της εικονικοποίησης, πολλαπλές εφαρμογές μπορούν να φιλοξενούνται στον ίδιο server, η κάθε μια εκτελώντας το δικό της λειτουργικό σύστημα, στοιχείο που συνεπάγεται τη μέγιστη δυνατή αξιοποίηση των διαθέσιμων πόρων.

Η αντικατάσταση των υλικών μηχανημάτων με ορισμένα από λογισμικό εικονικά μηχανήματα διευκολύνει τη χρήση και τη διαχείριση των πολιτικών που εφαρμόζονται στο σύστημα. Μέσω της αυτοματοποιημένης διαχείρισης υπηρεσιών, για παράδειγμα, οι διαχειριστές του συστήματος μπορούν να εφαρμόσουν συγκεκριμένες πολιτικές ασφαλείας σε συγκεκριμένα μηχανήματα, ανάλογα με το ρόλο τους. Επίσης, πολιτικές μπορούν να αυξήσουν την αποδοτικότητα των μηχανημάτων, μέσω της διακοπής λειτουργίας εικονικών μηχανημάτων που υπολειτουργούν,

προκειμένου να εξοικονομήσουν χώρο, υπολογιστική δύναμη και στην περίπτωση εφαρμογών που φιλοξενούνται στο υπολογιστικό νέφος, χρήματα.



Σχήμα 2: Αποδοτική εκμετάλλευση πόρων μέσω εικονικοποίησης

Εξίσου σημαντικό όφελος που προσφέρει η εικονικοποίηση είναι η ελαχιστοποίηση του χρόνου εκτός λειτουργίας, στοιχείο απαραίτητο για εφαρμογές που απαιτούν υψηλή διαθεσιμότητα. Πιο συγκεκριμένα, λειτουργικά συστήματα και διεργασίες που λόγω βλάβης αποτυγχάνουν μπορούν να διαταράξουν την ακεραιότητα και παραγωγικότητα της φιλοξενούμενης εφαρμογής. Κάτι τέτοιο μπορεί να αποφευχθεί με την παράλληλη εκτέλεση εφεδρικών εικονικών μηχανημάτων, προς αντικατάσταση βασικών εξυπηρετητών κατά την εμφάνιση πιθανής βλάβης. Προφανώς μία τέτοια στρατηγική είναι οικονομικά ακριβότερη.

Τέλος, η αγορά, εγκατάσταση και διαμόρφωση υλικού είναι χρονικά ακριβή διαδικασία, ειδικά εάν ληφθεί υπόψιν το πλήθος των εφαρμογών που δύναται να εξυπηρετηθούν. Με την προϋπόθεση, όμως, ότι το υλικό είναι ήδη διαθέσιμο, η προμήθεια της εφαρμογής με περισσότερα εικονικά μηχανήματα είναι σημαντικά ταχύτερη.

2.1.2 Hypervisor

Οι *Hypervisors* είναι κομμάτια λογισμικού που διαχειρίζονται τα εικονικά μηχανήματα. Λειτουργούν, πρακτικά, ως διεπαφές μεταξύ των εικονικών μηχανημάτων και του υλικού, εξασφαλίζοντας ότι κάθε ένα από αυτά έχει πρόσβαση στους πόρους που απαιτεί και έχει ανάγκη. Συμβάλλει, επίσης, στην απουσία παρεμβολών μεταξύ των εικονικών μηχανημάτων, εμποδίζοντας, για παράδειγμα, την καταπάτηση των εκάστοτε θέσεων μνήμης.

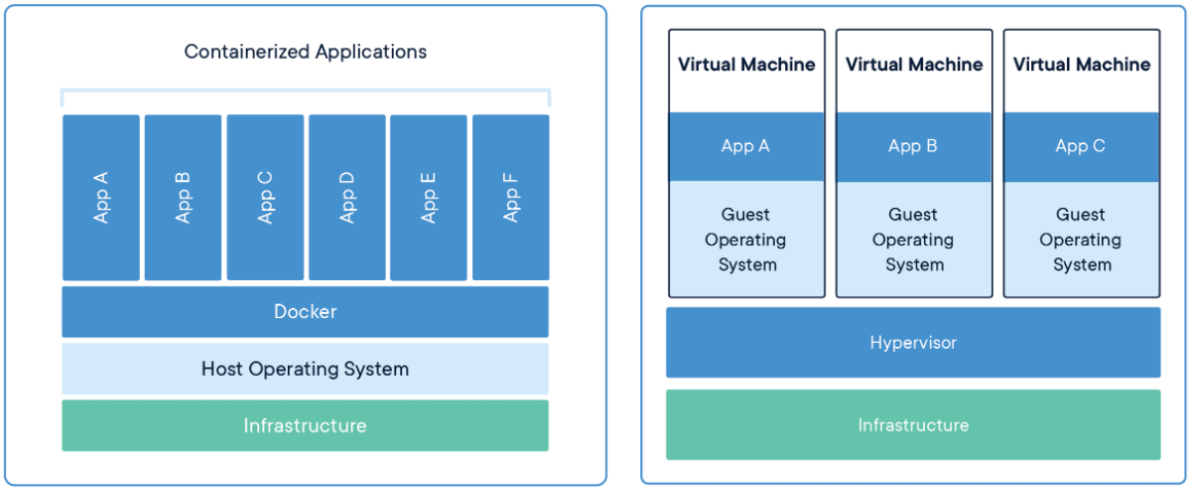
Υπάρχουν δύο τύποι hypervisors, ο τύπος 1 και ο τύπος 2. Ο τύπος 1 αλληλεπιδρά άμεσα με το υλικό αντικαθιστώντας το λειτουργικό σύστημα, ενώ ο τύπος 2 εκτελείται με τη μορφή εφαρμογής στο υπάρχων λειτουργικό σύστημα. Ο τελευταίος συνοδεύεται, συνήθως, από επιπρόσθετο υπολογιστικό κόστος και χρόνους απόκρισης, δεδομένου ότι πρέπει να χρησιμοποιεί το λειτουργικό σύστημα προκειμένου να διαχειριστεί τους διαθέσιμους πόρους. Αντίθετα, ο hypervisor τύπου 1 χαρακτηρίζεται από μεγαλύτερη ασφάλεια, επειδή διαχωρίζεται από το, ευάλωτο σε επιθέσεις, λειτουργικό σύστημα. Αξίζει, τέλος, να σημειωθεί ότι λόγω της ανάδειξης του χώρου του υπολογιστικού νέφους, έχουν αναπτυχθεί εξειδικευμένοι hypervisors για την εποπτεία προγραμμάτων και εφαρμογών που φιλοξενούνται στο cloud.

2.2 Containerization

Η τεχνική του *containerization* προβλέπει τη συγκέντρωση και το πακετάρισμα του κώδικα της εφαρμογής με της απαραίτητες, για την εκτέλεση αυτού, βιβλιοθήκες του λειτουργικού συστήματος, προς δημιουργία ενός εκτελέσιμου αρχείου, ικανό να εκτελεσθεί σε οποιαδήποτε υποδομή υλικού.

Κατά τους παραδοσιακούς τρόπους πλοήγησης εφαρμογών, ο κώδικας αναπτύσσεται σε ένα συγκεκριμένο υπολογιστικό περιβάλλον, ο οποίος εάν μεταφερθεί σε ένα διαφορετικό οικοσύστημα, συνήθως θα πρέπει να περάσει από τη διαδικασία της απασφαλμάτωσης προκειμένου να εκτελεσθεί επιτυχώς σε αυτό. Χαρακτηριστικό παράδειγμα αποτελούν οι εφαρμογές που αναπτύσσονται σε περιβάλλον Windows και παρουσιάζουν σφάλματα όταν μεταφερθούν σε εικονικά μηχανήματα με λειτουργικό σύστημα Linux. Το containerization καταπολεμά αυτό το πρόβλημα με το πακετάρισμα όλων των εξαρτήσεων του κώδικα σε μία μονάδα λογισμικού που φέρει το όνομα του *container*. Το τελευταίο είναι αποκομμένο από το λειτουργικό σύστημα του μηχανήματος στο οποίο δύναται να εκτελεσθεί, στοιχείο που του επιτρέπει την απρόσκοπτη πλοήγησή του σε σύστημα οποιασδήποτε υποδομής και προάγει την ευελιξία και τη μεταφερσιμότητά του.

Αξίζει να σημειωθεί ότι η ιδέα των containers πηγαίνει πίσω δεκαετίες, αλλά αναδείχθηκε περί το 2013 με το πρωτόκολλο ανοιχτού λογισμικού *Docker Engine*, το οποίο προβλέπει καθολικούς κανόνες ανάπτυξης και πακεταρίσματος λογισμικού. Στο σύγχρονο κόσμο τα containers αποτελούν τη βασική στρατηγική πλοήγησης εφαρμογών λογισμικού, ενώ εκτιμάται ότι τουλάχιστον το 50% των εταιρειών που ακολουθούν τη στρατηγική του containerization σκοπεύουν να περάσουν το ήδη υπάρχων λογισμικό τους σε containers μέσα στα επόμενα 2 χρόνια.



Σχήμα 3: Αρχιτεκτονική εικονικοποίησης (αριστερά) και containerization (δεξιά)

2.2.1 Πλεονεκτήματα

Πρώτο πλεονέκτημα του containerization αποτελεί η μεταφερσιμότητα, δεδομένου ότι η πακεταρισμένη εφαρμογή μπορεί να εκτελεστεί σε οποιοδήποτε σύστημα ανεξαρτήτως λειτουργικού συστήματος και υπολογιστικών πόρων. Πιο συγκεκριμένα, μοιράζονται τον πυρήνα του λειτουργικού συστήματος και δε φέρουν το δικό τους, στοιχείο που καθιστά τα containers ελαφρύτερα από τα εικονικά μηχανήματα και κατά συνέπεια πιο γρήγορα, αφού ο χρόνος εκκίνησής τους είναι σημαντικά ταχύτερος. Από το τελευταίο συνεπάγεται ότι και η κλιμάκωση των

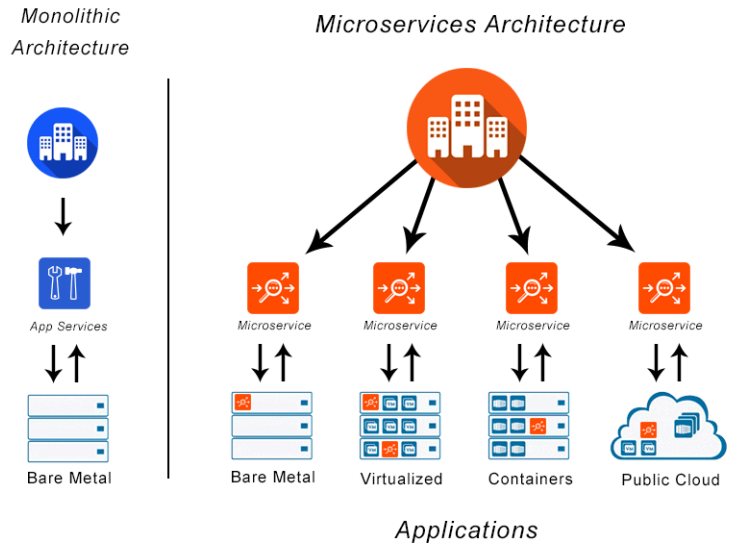
containers καθίσταται απλή, εύκολη και γρήγορη, αφού η πλοήγησή τους ισοδυναμεί με την εκτέλεση ενός εκτελέσιμου αρχείου κώδικα.

Κάθε containerized εφαρμογή αποτελεί ανεξάρτητο στοιχείο του συστήματος, προσφέροντας ανεκτικότητα σε βλάβες και ασφάλεια. Η ενδεχόμενη αποτυχία ενός container, για παράδειγμα, δε θα επηρεάσει την ομαλή λειτουργία άλλων containers που τυχών εκτελούνται στο ίδιο μηχάνημα. Με την ίδια ακριβώς λογική, ένα container δεν έχει πρόσβαση στους πόρους που καλύπτονται από άλλες containerized εφαρμογές.

Τέλος, τα containers προάγουν την αποδοτική εκμετάλλευση των διαθέσιμων πόρων, αφού πολλαπλές containerized εφαρμογές μπορούν να φιλοξενούνται στο ίδιο μηχάνημα, καταναλώνοντας η κάθε μία την απαραίτητη υπολογιστική δύναμη. Το στοιχείο αυτό σε συνδυασμό με τη μικρότερη, σε σχέση με τα εικονικά μηχανήματα χωρητικότητα, καθιστά δυνατή την εκτέλεση πολλαπλών containerized εφαρμογών σε ένα μόνο VM.

2.2.2 Μικροϋπηρεσίες

Η αρχιτεκτονική των μικροϋπηρεσιών αποτελεί μία στρατηγική που υιοθετείται από μικρές και μεγάλες εταιρείες λογισμικού και οφείλεται σε μεγάλο βαθμό στην εξέλιξη της τεχνολογίας των containers. Μέσω αυτής μία πολύπλοκη εφαρμογή χωρίζεται σε μία σειρά εξειδικευμένων υπηρεσιών κάθε μία από τις οποίες διαθέτει τη δική της λογική και το δικό της χώρο αποθήκευσης. Οι υπηρεσίες επικοινωνούν μεταξύ τους μέσω διεπαφών δικτύου και API, ενώ η λογική του μοντέλου αυτού επιτρέπει στις ομάδες προγραμματισμού να ανανεώνουν ή επιδιορθώνουν συγκεκριμένες υπηρεσίες, χωρίς να τεθεί το ολικό σύστημα εκτός λειτουργίας.



Σχήμα 4: Μοντέλο μικροϋπηρεσιών

Η αρχιτεκτονική των μικροϋπηρεσιών και οι containerized εφαρμογές λειτουργούν αποτελεσματικά όταν συνδυάζονται. Αυτό συμβαίνει γιατί τα containers προσφέρουν μία ευέλικτη συγκέντρωση οποιασδήποτε εφαρμογής, είτε πρόκειται για μονολιθικό πρόγραμμα, είτε για τμήμα αυτού. Μία μικροϋπηρεσία που έχει περάσει από το στάδιο του containerization κερδίζει όλα τα ευεργετικά στοιχεία των containers, δηλαδή μεταφερσιμότητα σε σχέση με τη διαδικα-

σία ανάπτυξης, συμβατότητα εκτέλεσης, ταχύτητα ανάπτυξης, απομόνωση βλαβών, αυτοματοποίηση εγκατάστασης και διαμόρφωσης.

Συνοπτικά, οι τεχνολογίες του υπολογιστικού νέφους, των containers, και των μικροϋπηρεσιών συνδυάζονται στο σύγχρονο τεχνολογικό κόσμο, για να προσφέρουν εξελιγμένα επίπεδα ανάπτυξης και πλοήγησης εφαρμογών σε σχέση με τα ήδη υπάρχοντα περιβάλλοντα και μεθοδολογίες. Αυτές οι προσεγγίσεις προσθέτουν ταχύτητα, αποδοτικότητα, αξιοπιστία και ασφάλεια στον κύκλο ανάπτυξης λογισμικού, στοιχεία που καθιστούν εφικτή την ταχύτερη παράδοση εφαρμογών και τη βελτίωση της εμπειρίας του χρήστη.

3

Kubernetes

Ο Kubernetes είναι μία φορητή, επεκτάσιμη πλατφόρμα ανοικτού λογισμικού, υπεύθυνη για τη διαχείριση containerized εφαρμογών και υπηρεσιών, που υποστηρίζει δηλωτική και αυτόματη διαμόρφωση. Το όνομά του προέρχεται από τον ελληνικό όρο *Κυβερνήτης*, που περιγράφει τον υπεύθυνο ή πιλότο, ενός πλοίου. Ευρέως χρησιμοποιούμενη είναι και η συντόμευση του ονόματός του, *K8s*, που προκύπτει από το αρχικό και τελικό γράμμα αυτού, με το άθροισμα του πλήθους των υπολοίπων χαρακτήρων να παρεμβάλλεται μεταξύ τους.

Με τα containers να προτιμώνται όλο κα περισσότερο από τις σύγχρονες εταιρείες λογισμικού και μη, ο Kubernetes προσφέρει ένα ανθεκτικό πλαίσιο στο οποίο τρέχουν κατανεμημένα συστήματα, ενώ παράλληλα φροντίζει για την κλίμακα και την ανακατεύθυνση της εφαρμογής σε περίπτωση βλάβης, υποστηρίζοντας διαφορετικά μοτίβα ανάπτυξης και αυτοματισμών. Πιο συγκεκριμένα πρόκειται για ένα υπολογιστικό σύστημα, το οποίο δύναται να αντικαταστήσει τον υπεύθυνο του περιβάλλοντος παραγωγής, στοχεύοντας στην ελαχιστοποίηση του χρόνου που η εφαρμογή θα βρίσκεται εκτός λειτουργίας.

3.1 Ιστορική Αναδρομή

Ο Kubernetes αποτελεί δημιούργημα της Google και αποτελεί το τρίτο σε σειρά σύστημα που αναπτύχθηκε από την εταιρεία, σε διάστημα 10 χρόνων, για τη διαχείριση containers. Κάθε σύστημα είναι έντονα επηρεασμένο από το προηγούμενο, αν και κάθε ένα από αυτά αναπτύχθηκε για να εξυπηρετήσει διαφορετικούς σκοπούς.

Το πρώτο ενοποιημένο σύστημα διαχείρισης containers είναι γνωστό με το όνομα *Borg*. Κατασκευάστηκε για να διαχειρίζεται τόσο υπηρεσίες μακράς διαρκείας, όσο και πακέτα εργασιών, εφαρμογές οι οποίες χρησιμοποιούν και μοιράζονται μία ομάδα μηχανημάτων προς αποδοτικότερη εκμετάλλευση των διαθέσιμων πόρων. Κάτι τέτοιο ήταν δυνατό, επειδή η υποστήριξη containers από τον πυρήνα των Linux είχε ξεκινήσει την εποχή εκείνη να είναι διαθέσιμη, κάτι το οποίο επέτρεπε καλύτερη απομόνωση μεταξύ εφαρμογών υψηλής διαθεσιμότητας ή απόκρισης και εργασιών υψηλών υπολογιστικών απαιτήσεων.

Όσο έτρεχαν όλο και περισσότερες εφαρμογές στο σύστημα Borg, οι ομάδες υποδομών και εφαρμογών της εταιρείας ανέπτυξαν ένα διευρυμένο οικοσύστημα εργαλείων και υπηρεσιών, για να το υποστηρίξουν. Αυτά τα συστήματα προσέφεραν επιλογές διαμόρφωσης και ανανέωσης εργασιών, πρόβλεψη υπολογιστικών απαιτήσεων, δυναμική εφαρμογή αρχείων διαμόρφωσης σε ήδη εξελισσόμενες εργασίες, αυτόματη μεταβολή της κλίμακας των εφαρμογών, διαχείριση του κύκλου ζωής των μηχανημάτων, διαχείριση ποσοστών και μεριδίων εργασιών και πολλά ακόμη. Η ανάπτυξη αυτού του οικοσυστήματος πήγαζε από τις διαφορετικές ανάγκες μεταξύ των ομάδων της Google, με αποτέλεσμα τη δημιουργία μιας ετερογενούς συλλογής συστημάτων, με τα οποία ο χρήστης έπρεπε να αλληλεπιδράσει σε διαφορετικές γλώσσες και δι-

εργασίες διαμόρφωσης, προκειμένου να εξυπηρετήσουν τις εκάστοτε ανάγκες. Σημειώνεται πως το Borg αποτελεί ακόμα και σήμερα το κυριότερο σύστημα διαχείρισης containers που χρησιμοποιείται από τη Google, λόγω της έκτασής, των χαρακτηριστικών και της ανθεκτικότητας που το χαρακτηρίζουν.

Απόγονος του Borg, αποτελεί το σύστημα *Omega* που αναπτύχθηκε, για να καλύψει την επιθυμία βελτίωσης και αναβάθμισης του λογισμικού του οικοσυστήματος του πρώτου. Εφαρμόζει αρκετά από τα μοτίβα που αποδείχθηκαν χρήσιμα και αποτελεσματικά στο Borg, αλλά κατασκευάστηκε από το μηδέν, για να έχει μία συνεπή αρχιτεκτονική. Το Omega έχε τη δυνατότητα να αποθηκεύει την κατάσταση της ομάδας μηχανημάτων, η οποία στη συνέχεια προσπελαύνεται από διαφορετικά μέρη του πίνακα ελέγχου αυτής μέσω μηχανισμών εξασφάλισης συνέπειας, προς αντιμετώπιση πιθανών συγκρούσεων. Αυτή η αποσύνδεση επιτρέπει στη λειτουργικότητα του συστήματος να διαχωριστεί σε όμοια κομμάτια, παρακάμπτοντας μία μονολιθική αρχιτεκτονική, στην οποία όλες οι αλλαγές περνούν από ένα κεντροποιημένο μηχάνημα-αφέντη. Πολλές από τις καινοτομίες του Omega, ενσωματώθηκαν στη συνέχεια στο Borg.

Το τρίτο σύστημα διαχείρισης container που αναπτύχθηκε από τη Google είναι ο Kubernetes. Σε αντίθεση με τα προηγούμενα δύο, ο Κυβερνήτης είναι πλατφόρμα ανοικτού λογισμικού που αναπτύχθηκε ως πλήρως εσωτερικό σύστημα. Όπως το Omega, έχει στον πυρήνα του μία κεντρική μόνιμη αποθήκη που μοιράζεται μεταξύ των μηχανημάτων και παρατηρητές αλλαγών σε σχετικά αντικείμενα. Σε αντίθεση με το Omega που εκθέτει την αποθήκη ευθέως σε εξαρτήματα του πίνακα ελέγχου, η κατάσταση του Κυβερνήτη προσπελαύνεται αποκλειστικά μέσω συγκεκριμένου REST API που εφαρμόζει έκδοση λογισμικού υψηλού επιπέδου, επικύρωση στοιχείων, σημασιολογία και πολιτική που υποστηρίζονται από ένα ευρύ φάσμα διαφορετικών μηχανημάτων. Αξίζει να σημειωθεί ότι ο Kubernetes αναπτύχθηκε με πρωταρχικό σκοπό τη βελτίωσης της εμπειρίας των προγραμματιστών που αναπτύσσουν λογισμικό, σχεδιασμένο να τρέξει σε ομάδα μηχανημάτων. Πιο συγκεκριμένα στοχεύει να διευκολύνει τη διαχείριση και την ανάπτυξη πολύπλοκων κατανεμημένων συστημάτων, ενώ παράλληλα εκμεταλλεύεται την αποδοτικότερη χρήση των πόρων που επιτρέπουν τα containers.

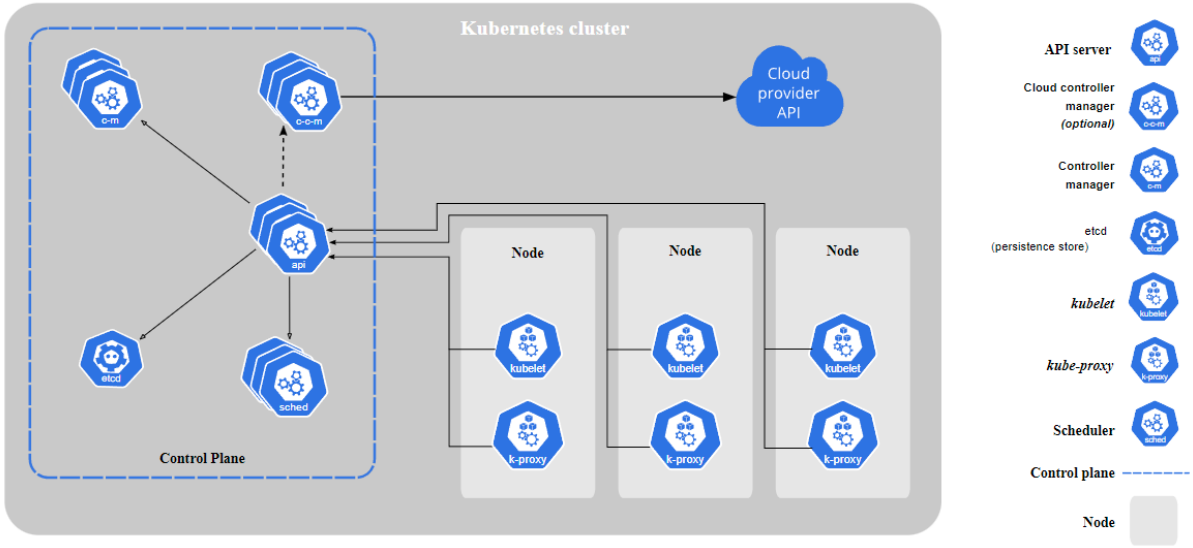
3.2 Δομικά Εξαρτήματα

Μία συστάδα στην οποία εκτελείται ο Κυβερνήτης αποτελείται από ένα σύνολο μηχανημάτων-εργατών που ονομάζονται *κόμβοι* (*Nodes*), κάθε ένα από το οποίο είναι ικανό να τρέξει containerized εφαρμογές. Κάθε συστάδα διαθέτει τουλάχιστον ένα μηχάνημα-εργάτη. Κάθε μηχάνημα-εργάτης φιλοξενεί *ακάτους* (*Pods*) που είναι εξαρτήματα της εφαρμογής. Ο *πίνακας ελέγχου* (*control plane*) διαχειρίζεται τα μηχανήματα-κόμβους και τις εκάστοτε ακάτους που τρέχουν σε αυτά. Σε περιβάλλοντα παραγωγής, ο πίνακας ελέγχου εκτελείται σε παραπάνω του ενός μηχανήματος και η συστάδα αποτελείται από πολλαπλά μηχανήματα προς εξασφάλιση υψηλής διαθεσιμότητας και ανεκτικότητας σε πιθανές βλάβες.

3.2.1 Εξαρτήματα Πίνακα Ελέγχου

Τα συστατικά του πίνακα ελέγχου λαμβάνουν αποφάσεις για τη συστάδα, για παράδειγμα τη δρομολόγηση εργασιών, ενώ παράλληλα ανιχνεύουν και αποκρίνονται σε συμβάντα που αφο-

ρούν τη συστάδα. Ενώ τα συστατικά αυτά μπορούν να εκτελεστούν σε οποιοδήποτε κόμβο της συστάδας, για λόγους απλότητας συνήθως παρατάσσονται στο ίδιο μηχάνημα, στο οποίο δεν εκτελείται κάποιο container του χρήστη και της εφαρμογής που πρόκειται να φιλοξενηθεί. Το συγκεκριμένο μηχάνημα συνήθως φέρει το όνομα του *κόμβου-αφέντη* (*master node*).



Σχήμα 5: Αρχιτεκτονική δομικών στοιχείων του Kubernetes

3.2.1.1 kube-apiserver

Ο API server του πίνακα ελέγχου του Κυβερνήτη είναι υπεύθυνος για την έκθεση του API του. Αποτελεί ουσιαστικά την εξωτερική διεπαφή του πίνακα με το υπόλοιπο σύστημα. Είναι σχεδιασμένος να ακολουθεί μία οριζόντια κλιμάκωση, που σημαίνει ότι κλιμακώνεται μέσω της δημιουργίας περισσοτέρων αντιγράφων του εαυτού του. Ο χρήστης μπορεί να εκμεταλλευθεί πολλαπλά αντίγραφά του ώστε να εξισορροπήσει την κίνηση του δικτύου μεταξύ τους.

3.2.1.2 etcd

Αποτελεί μία συνεπή και υψηλής διαθεσιμότητας αποθήκη που περιέχει πληροφορίες σχετικά με την κατάσταση της συστάδας του Kubernetes, σε μορφή κλειδιού-τιμής.

3.2.1.3 kube-scheduler

Το εξάρτημα αυτό είναι υπεύθυνο για την αντιστοίχιση ακάτων που έχουν δημιουργηθεί αλλά δεν εκτελούνται ακόμα σε κάποιο μηχάνημα της συστάδας. Στους παράγοντες που λαμβάνονται υπόψιν περιλαμβάνονται οι απαιτήσεις υπολογιστικής ισχύος, οι περιορισμοί υλικού, λογισμικού και πολιτικής, οι προδιαγραφές προτίμησης του χρήστη, η τοπικότητα των δεδομένων και οι διορίες.

3.2.1.4 kube-controller-manager

Εξάρτημα που εκτελεί διεργασίες ελέγχου. Λογικά κάθε ελεγκτής αποτελεί μία ξεχωριστή διεργασία, αλλά για λόγους απλότητας το σύνολο αυτών συγκεντρώνεται σε ένα δυαδικό αρχείο εκτέλεσης και εκτελείται ως μία διεργασία. Κάποιοι από τους τύπους αυτών των ελεγκτών είναι ο *ελεγκτής κόμβων* (*node controller*) που ανιχνεύει και ανταποκρίνεται σε περίπτωση που κάποιο μηχάνημα βγει εκτός λειτουργίας, ο *ελεγκτής διεργασιών* (*job controller*) που παρατη-

ρεί για αντικείμενα εργασιών και δημιουργεί άκατους προς εκτέλεση αυτών, ο *ελεγκτής διαθέσιμων άκρων* (*endpoints controller*) που συνδέει άκατους μέσω υπηρεσιών προς επικοινωνία αυτών και ο *ελεγκτής λογαριασμού υπηρεσίας* (*service account & token controller*) που δημιουργεί και διαχειρίζεται λογαριασμούς αδειών για τους ονοματοχώρους της συστάδας.

3.2.1.5 cloud-controller-manager

Εξάρτημα το οποίο ενσωματώνει λογική ελέγχου νέφους. Πιο συγκεκριμένα επιτρέπει τη σύνδεση της συστάδας με το API του παρόχου του νέφους και διαχωρίζει τα συστατικά στοιχεία που αλληλεπιδρούν μόνο με εκείνο από αυτά που επικοινωνούν αποκλειστικά με τη συστάδα. Ο συγκεκριμένος διαχειριστής εκτελεί συγκεκριμένους ελεγκτές για κάθε πάροχο υπηρεσιών νέφους, ενώ σε περίπτωση που ο χρήστης διαθέτει δικό του περιβάλλον και μηχανήματα, ο διαχειριστής δεν εγκαθίσταται. Όπως και ο kube-controller-manager, συγκεντρώνει όλες τις λογικές διεργασίες σε ένα δυαδικό εκτελέσιμο αρχείο. Μπορεί να επεκταθεί με οριζόντια κλιμάκωση προς βελτίωση της απόδοσης και διευκόλυνση στην ανοχή βλαβών.

3.2.2 Εξαρτήματα Κόμβων

Τα εξαρτήματα κόμβων τρέχουν σε κάθε κόμβο της συστάδας, διατηρώντας τη λειτουργία εκτελούμενων Pod.

3.2.2.1 kubelet

Πρόκειται για πράκτορα που εξασφαλίζει ότι τα containers της εφαρμογής εκτελούνται σε άκατους. Πιο συγκεκριμένα λαμβάνει ένα σύνολο προδιαγραφών που χαρακτηρίζουν τα Pods και ελέγχει εάν τα containers στο εσωτερικό αυτών εκτελούνται επιτυχώς και είναι υγιή. Ο kubelet δεν παρακολουθεί containers που έχουν δημιουργηθεί εκτός του οικοσυστήματος του Κυβερνήτη.

3.2.2.2 kubeproxy

Αποτελεί ένα διακομιστή μεσολάβησης που τρέχει σε κάθε κόμβο συνθέτοντας ένα κομμάτι των Υπηρεσιών του Kubernetes. Διατηρεί ένα σύνολο κανόνων δικτύου για όλα τα Nodes, οι οποίοι επιτρέπουν την επικοινωνία των άκατων με στοιχεία, τόσο εσωτερικά όσο και εξωτερικά της συστάδας του Κυβερνήτη.

3.2.2.3 container-runtime

Απαραίτητο λογισμικό που είναι υπεύθυνο για την εκτέλεση των containers. Ο Κυβερνήτης υποστηρίζει διάφορα τέτοια λογισμικά, όπως *Docker*, *containerd*, *CRI-O*.

3.3 Αρχιτεκτονική Συστάδας

3.3.1 Κόμβοι (Nodes)

Όπως έχει ήδη σημειωθεί, ο Κυβερνήτης εκτελεί τον όγκο εργασίας του χρήστη δημιουργώντας containers που τοποθετεί μέσα σε Pods, τα οποία με τη σειρά τους τρέχουν σε κόμβους.

Ένας κόμβος μπορεί να είναι είτε φυσικό, είτε εικονικό μηχάνημα, χαρακτηριστικό το οποίο εξαρτάται από τη συστάδα. Κάθε κόμβος διαχειρίζεται από τον αντίστοιχο πίνακα ελέγχου και περιλαμβάνει όλες εκείνες τις υπηρεσίες που είναι απαραίτητες για να εκτελούνται αποτελεσματικά οι άκατοι της εφαρμογής.

3.3.1.1 Διαχείριση

Υπάρχουν δύο διαφορετικοί τρόποι για να προστεθούν κόμβοι στον API server του Κυβερνήτη: ο πράκτορας kubelet του Node εγγράφει ο ίδιος τον κόμβο στον πίνακα ελέγχου ή ο διαχειριστής-χρήστης του Kubernetes προσθέτει χειροκίνητα ένα αντικείμενο τύπου Node. Σημειώνεται ότι η αυτόματη προσθήκη του κόμβου είναι ο προτιμώμενος τρόπος εγγραφής κόμβου και χρησιμοποιείται στην πλειονότητα των κατανομών. Μετά την παραπάνω προσθήκη ο πίνακας ελέγχου εξετάζει εάν το καινούριο αντικείμενο είναι έγκυρο και υγιές, εάν δηλαδή ανταποκρίνετααι σε ερεθίσματα του πίνακα ελέγχου και εκτελεί τις απαραίτητες υπηρεσίες. Στην περίπτωση θετικού αποτελέσματος, ο κόμβος αναγνωρίζεται και είναι στη θέση να εκτελέσει Pods εφαρμογών, ενώ σε διαφορετική περίπτωση αγνοείται από τη συστάδα μέχρι να φτάσει σε υγιή κατάσταση.

3.3.1.2 Ονομασία και μοναδικότητα

Το όνομα ενός κόμβου είναι το στοιχείο που το χαρακτηρίζει. Για αυτό το λόγο δύο Nodes δε μπορούν να έχουν το ίδιο όνομα ταυτόχρονα, γιατί σε διαφορετική περίπτωση ο Kubernetes τα αντιμετωπίζει ως το ίδιο αντικείμενο. Στην περίπτωση των κόμβων συμπεραίνεται έμμεσα ότι αντίγραφα που χρησιμοποιούν το ίδιο όνομα έχουν και ίδια κατάσταση λειτουργίας και χαρακτηριστικά, γεγονός το οποίο μπορεί να οδηγήσει σε ασυνέπεια, εάν ένα αντικείμενο επεξεργαστεί χωρίς να μεταβληθεί το όνομά του. Εάν ένας κόμβος χρειαστεί να αντικατασταθεί ή ενημερωθεί σημαντικά, είναι απαραίτητο να αφαιρεθεί από τον API server και να επαναπροσθεθεί με τις επιθυμητές ιδιότητες.

3.3.1.3 Κατάσταση Κόμβων (node-state)

Η κατάσταση ενός κόμβου περιλαμβάνει τα χαρακτηριστικά των διευθύνσεων, της ακεραιότητας, της χωρητικότητας και της κατανομής, καθώς και γενικές πληροφορίες.

3.3.1.3.1 Διευθύνσεις

Πληροφορίες σχετικά με το domain όνομα, όπως αυτό δηλώθηκε από το πυρήνα του κόμβου, εξωτερικές και εσωτερικές IP διευθύνσεις. Σημειώνεται ότι τελευταίες αφορούν διευθύνσεις που δρομολογούνται εσωτερικά της συστάδας, ενώ οι πρώτες εκείνες που είναι διαθέσιμες για δρομολόγηση εξωτερικά αυτής.

3.3.1.3.2 Ακεραιότητα

Πληροφορίες σχετικά με την υγεία και ετοιμότητα ενός κόμβου. Μερικά παραδείγματα του συγκεκριμένου πεδίου είναι *Ready*, που εκφράζει ότι το Node είναι έτοιμο να δεχθεί Pods εφαρμογών, *DiskPressure*, που αντικατοπτρίζει περιορισμένη χωρητικότητα δίσκου, *MemoryPressure*, που περιγράφει περιορισμένη χωρητικότητα μνήμης, *PIDPressure*, που εκφράζει ότι μεγάλο πλήθος διεργασιών φιλοξενούνται στον κόμβο, και *NetworkUnavailable*, σε περίπτωση που το δίκτυο του κόμβου δεν έχει διαμορφωθεί σωστά. Σε περίπτωση που ένας κόμβος παραμείνει στην κατάσταση *Unavailable* περισσότερο από προκαθορισμένο διάστημα

(eviction-timeout), ο ελεγκτής κόμβων αφαιρεί από τον apiserver, όλες τις ακάτους που έχουν αποδοθεί στο δεδομένο Node, προκειμένου να δρομολογηθούν σε άλλο διαθέσιμο κόμβο.

3.3.1.3.3 Χωρητικότητα και κατανομή

Περιγράφουν τους πόρους που είναι διαθέσιμοι σε ένα Node. Πιο συγκεκριμένα δίνει πληροφορίες σχετικά με τους πόρους επεξεργαστή, μνήμης, καθώς και το μέγιστο πλήθος ακάτων που μπορούν να φιλοξενηθούν στον κόμβο. Δίνονται πληροφορίες τόσο για τους συνολικούς πόρους, όσο και για τους διαθέσιμους που μπορούν να εκμεταλλευθούν από εφαρμογές.

3.3.1.3.4 Γενικές πληροφορίες

Πληροφορίες σχετικά με τον κόμβο, όπως η έκδοση πυρήνα, η έκδοση Kubernetes, λεπτομέρειες σχετικά με το περιβάλλον εκτέλεσης των containers και το λογισμικό του Node. Οι πληροφορίες αυτές συλλέγονται από τον πράκτορα kubelet και δημοσιεύονται στον API server του Κυβερνήτη.

3.3.1.4 Σήματα ελέγχου (heartbeats)

Οι κόμβοι της συστάδας στέλνουν ανά τακτά χρονικά διαστήματα σήματα ελέγχου προς προσδιορσμό της διαθεσιμότητας του Node και επίλυσης τυχών ασυνέπειας, εάν αυτή προκύψει. Για τους κόμβους υπάρχουν δύο ειδών σημάτων: είτε ενημέρωση της κατάστασης του Node, είτε μέσω *Lease* αντικειμένων. Ο τελευταίος τύπος σημάτων πρόκειται για μία οικονομικότερη λύση, από άποψη κλιμάκωσης και απόδοσης καθώς χρησιμοποιεί εξιδεικευμένο API. Ο πράκτορας kubelet είναι υπεύθυνος για την διαχείριση των σημάτων ελέγχου των κόμβων.

3.3.1.5 Ελεγκτής κόμβων (node-controller)

Πρόκειται για στοιχείο του πίνακα ελέγχου που διαχειρίζεται διάφορες ενέργειες των κόμβων και έχει πολλαπλούς ρόλους κατά τη διάρκεια ζωής αυτών. Ο πρώτος είναι να αναθέτει ένα CIDR block διευθύνσεων στο Node, όταν αυτό δημιουργηθεί για πρώτη φορά. Ο δεύτερος είναι να διατηρεί την εσωτερική λίστα της συστάδας του κόμβου επικαιροποιημένη και ο τρίτος αφορά τον έλεγχο της υγείας και διαθεσιμότητας του Node.

3.3.2 Επικοινωνία Πίνακα Ελέγχου - Κόμβων

Υπάρχουν δύο μονοπάτια επικοινωνίας μεταξύ του πίνακα ελέγχου και δει του API server του Κυβερνήτη και της συστάδας.

3.3.2.1 Κόμβος σε πίνακα ελέγχου

Ο Kubernetes ακολουθεί προσέγγιση *hub-and-spoke*, που σημαίνει ότι η χρήση του API από τα Nodes καταλήγει πάντα στον API server. Κανένα άλλο εξάρτημα του πίνακα ελέγχου δεν έχει κατασκευαστεί, για να εκθέτει απόμακρες υπηρεσίες. Ο apiserver έχει διαμορφωθεί ώστε να αποκρίνεται σε συνδέσεις που ακολουθούν πρωτόκολλο HTTPS με τουλάχιστον μία μέθοδο επαλήθευσης χρήστη. Κάτι τέτοιο είναι απαραίτητο, ειδικά εάν τα ανώνυμα αιτήματα είναι ενεργοποιημένα.

Οι κόμβοι θα πρέπει να είναι προμηθευμένοι με το δημόσιο πιστοποιητικό root του cluster, ώστε να μπορούν να συνδέονται με ασφαλή τρόπο στον apiserver με έγκυρα στοιχεία χρήστη.

Η προτεινόμενη προσέγγιση προβλέπει τα στοιχεία χρήστη που παρέχονται στον kubelet να είναι σε μορφή πιστοποιητικού χρήστη. Άκατοι που επιθυμούν να συνδεθούν με τον apiserver μπορούν να το κάνουν με ασφαλή τρόπο, μέσω της μόχλευσης λογαριασμών υπηρεσίας, έτσι ώστε ο Κυβερνήτης να εισάγει το δημόσιο root certificate και την αντίστοιχη λεξικογραφική μονάδα σε αυτές με αυτόματο τρόπο κατά τη δημιουργία τους.

3.3.2.2 Πίνακας ελέγχου σε κόμβο

Υπάρχουν δύο βασικά μονοπάτια επικοινωνίας από τον πίνακα ελέγχου στους κόμβους. Η επικοινωνία ξεκινά από τον apiserver και κατά τον πρώτο τρόπο καταλήγει στον kubelet που εκτελείται σε κάθε Node της συστάδας, ενώ κατά το δεύτερο χρησιμοποιείται η λειτουργικότητα του διακομιστή διαμεσολάβησης του ίδιου του apiserver.

3.3.3 Ελεγκτές

Στους τομείς της ρομποτικής και των αυτοματισμών οι βρόχοι ελέγχου είναι ατέρμονες βρόχοι που ρυθμίζουν την κατάσταση ενός συστήματος. Ο θερμοστάτης ενός δωματίου, αποτελεί, για παράδειγμα, έναν τέτοιο βρόχο ελέγχου. Στον Κυβερνήτη οι ελεγκτές είναι βρόχοι ελέγχου που παρακολουθούν την κατάσταση της συστάδας και κάνουν αιτήσεις για αλλαγές όταν χρειάζεται. Κάθε ελεγκτής αποσκοπεί στο να τοποθετήσει το σύνολο της συστάδας πιο κοντά στην επιθυμητή κατάσταση.

3.3.3.1 Μοτίβα ελέγχου

Κάθε ελεγκτής παρακολουθεί τουλάχιστον έναν πόρο του Kubernetes. Αυτά τα αντικείμενα χαρακτηρίζονται από ένα πεδίο τιμής που αντιπροσωπεύει την επιθυμητή κατάσταση. Οι ελεγκτές του δεδομένου πόρου προσπαθούν να φέρουν την τωρινή κατάσταση πιο κοντά στην επιθυμητή. Κάτι τέτοιο μπορεί να συμβεί με τον ελεγκτή να εκτελεί μόνος του τις απαραίτητες ενέργειες και αλλαγές. Πιο συχνά, ωστόσο, οι ελεγκτές στέλνουν στον apiserver μηνύματα ή σήματα, τα οποία πυροδοτούν με τη σειρά τους επιθυμητές και χρήσιμες παρενέργειες.

3.3.3.1.1 Έλεγχος μέσω apiserver

Ο ελεγκτής Job είναι ένα παράδειγμα ενσωματωμένου ελεγκτή του Κυβερνήτη, ο οποίος διαχειρίζεται την κατάσταση της συστάδας μέσω της αλληλεπίδρασής του με τον apiserver. Αντικείμενα τύπου Job εκτελούνται σε Pods προκειμένου να εκτελέσουν μία ενέργεια και τερματίζουν με την ολοκλήρωση αυτής.

Όταν ένας ελεγκτής Job ανιχνεύσει μία νέα εργασία προς εκτέλεση, εξακριβώνει πως κάπου μέσα στη συστάδα οι πράκτορες kubelet σε ένα σύνολο κόμβων εκτελούν το σωστό αριθμό ακάτων για να φέρουν εις πέρας. Ο ελεγκτής δεν εκτελεί Pods ή containers μόνος του. Αντίθετα, ειδοποιεί κατάλληλα τον apiserver.

Όταν ο χρήστης δημιουργήσει ένα αντικείμενο τύπου Job, η επιθυμητή κατάσταση είναι η αντίστοιχη εργασία να ολοκληρωθεί. Ο ελεγκτής Job κάνει την τωρινή κατάσταση να έρθει πιο κοντά στην επιθυμητή με το να δημιουργήσει τα απαραίτητα Pods για την εργασία, τα οποία θα την εκτελέσουν, ώστε εκείνη να είναι πιο κοντά στην ολοκλήρωση. Σημειώνεται ότι οι ελεγκτές ενημερώνουν ακόμα και τα αντικείμενα που τους διαμόρφωσαν. Για παράδειγμα, όταν το

αντικείμενο τύπου Job ολοκληρωθεί, ο αντίστοιχος ελεγκτής ενημερώνει την κατάστασή του σε *Finished*.

3.3.3.1.2 Άμεσος έλεγχος

Σε αντίθεση με το προηγούμενο παράδειγμα, υπάρχει η περίπτωση ελεγκτές να πρέπει να μεταβάλλουν αντικείμενα εξωτερικά της συστάδας του Kubernetes. Για παράδειγμα, ένας βρόχος ελέγχου που παρακολουθεί το πλήθος των διαθέσιμων Nodes στη συστάδα, χρειάζεται να επικοινωνήσει με εξωτερικά στοιχεία του cluster, για να δημιουργήσει νέους κόμβους, εάν αυτό χρειαστεί. Τέτοιοι ελεγκτές που αλληλεπιδρούν με εξωτερικές καταστάσεις βρίσκουν την επιθυμητή κατάσταση από τον apiserver και έπειτα επικοινωνούν άμεσα με το εξωτερικό σύστημα, για να το φέρουν στην επιθυμητή κατάσταση.

Σημειώνεται ότι ο ελεγκτής κάνει τις απαραίτητες ενέργειες, για να φέρει το σύστημα στην επιθυμητή κατάσταση και έπειτα δίνει αναφορά στον apiserver της συστάδας. Άλλοι βρόχοι ελέγχου μπορούν να παρατηρήσουν τα δεδομένα που αναφέρθησαν και να προβούν στο δικό τους σύνολο ενεργειών. Στο παράδειγμα του θερμοστάτη, εάν το δωμάτιο είναι πολύ κρύο, κάποιος άλλος ελεγκτής μπορεί να ενεργοποιήσει το θερμαντικό σώμα που προστατεύει από τον παγετό.

3.3.3.2 Τωρινή και επιθυμητή κατάσταση

Αξίζει να παρατηρηθεί ότι η συστάδα μπορεί να αλλάζει σε οποιοδήποτε σημείο όσο εκτελούνται εργασίες και βρόχοι ελέγχουν διορθώνουν αυτόματα βλάβες και ασυνέπειες. Αυτο πρακτικά σημαίνει ότι η συστάδα δε θα φτάσει ποτέ, εν δυνάμει, κάποια σταθερή κατάσταση. Παρόλα αυτά, όσο οι ελεγκτές εκτελούνται και είναι ικανοί να πραγματοποιήσουν χρήσιμες αλλαγές, δεν έχει σημασία ένα η συστάδα φτάσει ή όχι σε μόνιμη κατάσταση. Εξάλλου, βασικό χαρακτηριστικό του Κυβερνήτη είναι η ικανότητά του να αντεπεξέρχεται σε τυχόν συνεχείς αλλαγές.

3.3.3.3 Σχεδιασμός

Τιμώντας την αρχή του σχεδιασμού του, ο Κυβερνήτης χρησιμοποιεί πολλούς ελεγκτές, καθένας από τους οποίους διαχειρίζεται ένα συγκεκριμένο στοιχείο του cluster. Το πιο σύνηθες είναι ένας συγκεκριμένος βρόχος ελέγχου να χρησιμοποιεί ένα είδος πόρου ως την επιθυμητή κατάσταση και να έχει ένα διαφορετικό είδος πόρου που θα φέρει την φέρει εις πέρας. Είναι αποτελεσματικότερο να υπάρχουν πολλαπλοί απλούστεροι ελεγκτές, παρά ένας, που θα αποτελείται από ένα μονολιθικό σύστημα διασυνδεδεμένων βρόχων ελέγχου. Προφανώς οι ελεγκτές μπορούν να παρουσιάσουν βλάβη κατά τη διάρκεια της ζωής τους, γεγονός στο οποίο ο Kubernetes είναι ανεκτικός.

3.3.4 Διεπαφή εκτέλεσης container (Container Runtime Interface)

Πρόκειται για μία πρόσθετη διεπαφή της συστάδας που επιτρέπει στον πράκτορα kubelet του κάθε μηχανήματος να χρησιμοποιεί μία ποικιλία διαφορετικών λογισμικών εκτέλεσης container, χωρίς να χρειάζεται να επαναμεταγλωττίσει τα συστατικά στοιχεία της συστάδας. Η εν λόγω διεπαφή είναι το βασικό πρωτόκολλο επικοινωνίας μεταξύ του kubelet και του λογισμικού εκτέλεσης container.

3.3.5 Συλλογή Σκουπιδιών (Garbage Collection)

Η συλλογή σκουπιδιών είναι ένας συλλογικός όρος για τους διάφορους μηχανισμούς του Κυβερνήτη να καθαρίζει πόρους της συστάδας. Οι μηχανισμοί αυτοί του επιτρέπουν να συλλέγει αποτυχημένα Pods, ολοκληρωμένες εργασίες, αντικείμενα χωρίς αναφορές ιδιοκτήτη, αχρησιμοποίητα containers, δυναμικά παρεχόμενες χώρους αποθήκευσης, αντικείμενα τύπου Lease.

Όταν λαμβάνει χώρα η διαγραφή ενός αντικειμένου, ο Kubernetes διαγράφει αυτόματα όλα τα παρελκόμενα στοιχεία αυτού μέσω της διαδικασίας διαγραφής του καταρράκτη (cascading-deletion). Υπάρχουν δύο τύποι αυτής της διαδικασίας: η εμπρόσθια διαγραφή και η οπίσθια διαγραφή. Κατά την πρώτη, το αντικείμενο προς διαγραφή παραμένει ορατό μέσω του API του Kubernetes μέχρις ότου ολοκληρωθεί η διαγραφή των συστατικών στοιχείων που λαμβάνει χώρα στο παρασκήνιο. Κατά την τελευταία, το αντικείμενο αφαιρείται από το API αμέσως, ενώ η διαγραφή των στοιχείων του βρίσκεται σε εξέλιξη. Ο Κυβερνήτης χρησιμοποιεί οπίσθια διαγραφή από προεπιλογή, με το χρήστη να έχει τη δυνατότητα να αλλάξει αυτή τη συμπεριφορά.

3.4 Φόρτοι Εργασίας – Workloads

Ένας φόρτος εργασίας είναι μία εφαρμογή που τρέχει, μέσα σε Pods, στον Κυβερνήτη. Οι άκατοι έχουν καθορισμένο κύκλο ζωής. Εάν για παράδειγμα ένας κόμβος της συστάδας αποτύχει λόγω βλάβης, αποτυγχάνουν και όλα τα Pods που φιλοξενούνται σε αυτόν. Ο Kubernetes θεωρεί το συγκεκριμένο στάδιο αποτυχίας ως τελικό, που σημαίνει ότι ο χρήστης χρειάζεται να δημιουργήσει νέες ακάτους, αντικαθιστώντας εκείνες που απέτυχαν, προκειμένου να επιστρέψει στην προηγούμενη, επιθυμητή, κατάσταση.

Προς απλούστευση της παραπάνω διαδικασίας, ο χρήστης μπορεί να χρησιμοποιήσει πόρους τύπου workload, οι οποίοι διαχειρίζονται ένα σύνολο από Pods, μόνοι τους, χωρίς να χρειάζεται ανθρώπινη παρεμβολή. Οι δεδομένοι πόροι δημιουργούν ελεγκτές, οι οποίοι εξασφαλίζουν ότι τρέχει ο σωστός αριθμός ακάτων, αντιστοιχίζοντας την κατάσταση που έχει δηλώσει ο χρήστης.

Οι φόρτοι εργασίας μπορούν να εφαρμοστούν μέσω εργαλείου της γραμμής εντολών που φέρει το όνομα *kubectl* και επιτρέπει την επικοινωνία του χρήστη με συστάδες Κυβερνήτη. Οι πόροι προς εφαρμογή περνάνε στο εργαλείο ως ορίσματα σε μορφή αρχείων *yaml*.

Ο Κυβερνήτης παρέχει διάφορους ενσωματωμένους πόρους φόρτου εργασίας, όπως *Deployment*, *ReplicaSet*, *StatefulSet*, *DaemonSet* και *Job*.

3.4.1 Deployments

Πόροι τύπου Deployment, παρέχουν δηλωτικές ενημερώσεις για Pods και ReplicaSets. Ο χρήστης περιγράφει την επιθυμητή κατάσταση σε έναν τέτοιο πόρο και ο αντίστοιχος ελεγκτής μεταβάλλει την τωρινή κατάσταση με ελεγχόμενη ρυθμό, προκειμένου να φτάσει στην επιθυμητή.

3.4.2 ReplicaSets

Βασικός σκοπός ενός τέτοιου πόρου είναι να διατηρεί ένα σταθερό σύνολο από Pods, ώστε να εκτελούνται επιτυχώς σε κάθε χρονική στιγμή. Για αυτό το λόγο είναι το κύριο εργαλείο που χρησιμοποιείται προκειμένου να εξασφαλιστεί η διαθεσιμότητα ενός συγκεκριμένου αριθμού ακάτων.

Ένας φόρτος εργασίας τύπου ReplicaSet ορίζεται από πεδία που περιλαμβάνουν έναν επιλογέα, προκειμένου να προσδιοριστούν τα Pods που μπορεί να αποκτήσει, το πλήθος των αντιγράφων των ακάτων που πρέπει να διατηρεί και το πρότυπο των Pods, στο οποίο θα πρέπει αυτά να υπακούν. Στη συνέχεια το ReplicaSet πραγματοποιεί τον σκοπό του με το να δημιουργεί και να καταστρέφει Pods, ανάλογα με τις απαιτήσεις που του προσδιορίστηκαν.

Σημειώνεται πως τα Deployments χαρακτηρίζονται από υψηλότερη ιεραρχία και διαχειρίζονται πόρους τύπου ReplicaSet. Για αυτό το λόγο προτιμάται η χρήση Deployments έναντι των τελευταίων, εάν δεν υπάρχουν ιδιαίτερες απαιτήσεις από την εφαρμογή.

3.4.3 StatefulSets

Όπως και τα Deployments, έτσι και τα StatefulSets διαχειρίζονται ακάτους που χαρακτηρίζονται από όμοιες προδιαγραφές. Η διαφορά τους είναι ότι για κάθε Pod διατηρούν μία μοναδική ταυτότητα, την οποία δε χάνουν, αλλά διατηρούν ανεξάρτητα από τυχόν αναπρογραμματιστιμούς που μπορεί να λάβουν χώρα. Εάν για παράδειγμα ένα Pod αποτύχει κατά τη διάρκεια ζωής του, το Stateful θα φροντίσει να το επαναδημιουργήσει διατηρώντας στοιχεία όπως αναγνωριστές δικτύου και μόνιμους αποθηκευτικούς χώρους.

3.4.4 DaemonSets

Τα DaemonSets εξασφαλίζουν ότι όλοι οι κόμβοι της συστάδας εκτελούν ένα αντίγραφο ενός συγκεκριμένου Pod. Στην περίπτωση που το πλήθος των κόμβων του cluster αυξηθεί, οι αντίστοιχες άκατοι θα προστεθούν σε αυτούς, με αντίστοιχη σειρά γεγονότων να λαμβάνει χώρα στην περίπτωση της διαγραφής Node.

Μερικά παραδείγματα DaemonSets που μπορεί να τρέχουν από προεπιλογή στον Κυβερνήτη περιλαμβάνουν δαίμονες που κρατάνε αρχεία καταγραφής για τον εκάστοτε κόμβο, δαίμονες υπεύθυνοι για μετρήσεις των κόμβων και δαίμονες διαχείρισης του διαθέσιμου αποθηκευτικού χώρου.

3.4.5 Jobs

Οι δεδομένοι τύποι εργασίας δημιουργούν ένα ή περισσότερα Pods που επιχειρούν να εκτελέσουν επ’ άπειρον, έως ότου ολοκληρωθεί επιτυχώς η εκτέλεση ενός συγκεκριμένου πλήθους από αυτά. Τα Jobs μπορούν να χρησιμοποιηθούν για να εκτελούνται πολλαπλές άκατοι παράλληλα.

3.5 Υπηρεσίες – Services και Δικτύωση

Κάθε Pod έχει τη δικιά του διεύθυνση IP. Αυτό σημαίνει ότι τόσο η αναλυτική διασύνδεση α-κάτων, όσο και η αντιστοίχιση θυρών containers σε θύρες εξυπηρετητή μπορεί να αποφευχθεί, δημιουργώντας ένα απλό και κομψό μοντέλο επικοινωνίας, στο οποίο τα Pods πρακτικά συ-μπεριφέρονται ως φυσικά ή εικονικά μηχανήματα.

Οι διευθύνσεις IP υπάρχουν στην εμβέλεια των ακάτων, δηλαδή containers που βρίσκονται στο ίδιο Pod μοιράζονται τις διευθύνσεις IP και MAC, γεγονός το οποίο επιτρέπει την επικοι-νωνία τους μέσω του localhost.

3.5.1 Service

Ο πόρος τύπου *Service* είναι ένας αφηρημένος τρόπος για να εκτεθεί μία εφαρμογή, που εκτε-λείται σε ένα σύνολο Pod, ως υπηρεσία δικτύου. Ο Κυβερνήτης αποδίδει σε καθένα από αυτά τη δική του διεύθυνση IP και εξισορροπεί αυτόματα το φόρτο δικτύου μεταξύ τους. Πιο συγκε-κριμένα, τα Services ορίζουν ένα σύνολο από ακάτους και μία πολιτική σύμφωνα με την οποία επιτυγχάνεται η προσπέλασή τους. Το σύνολο αυτό καθορίζεται συνήθως από έναν επιλογέα που δηλώνεται στο αντίστοιχο αρχείο διαμόρφωσης.

Ένα παράδειγμα θα ήταν μία εφαρμογή που αποτελείται από 3 backend replicas. Οι frontend καταναλωτές της δε χρειάζεται ούτε να ξέρουν την αρχιτεκτονική συστοιχία των ακάτων του backend, ούτε ποιο Pod καταναλώνουν. Αυτή η απόζευξη καθίσταται δυνατή με τα Services.

3.5.2 Ingress

Ο Ingress είναι υπεύθυνος να εκθέτει διευθύνεις, που ακολουθούν το πρωτόκολλο HTTP και HTTPS από συστήματα εξωτερικά της συστάδας σε Services εντός εκείνης. Η δρομολόγηση της κίνησης πραγματοποιείται σύμφωνα με τους κανόνες διαμόρφωσης του πόρου Ingress.

Σχήμα 6: Η λειτουργία Ingress

Μία υπηρεσία Ingress μπορεί να προγραμματιστεί να δίνει σε services εξωτερικά προσπελά-σιμα URLs, να εξισορροπεί την κίνηση του δικτύου και να προσφέρει εξυπηρέτηση μέσω ει-κονικών διευθύνσεων.

3.6 Αποθήκευση

Τα αποθηκευμένα αρχεία σε ένα container είναι εφήμερα, γεγονός που προκαλεί σειρά προβλημάτων για τις εφαρμογές. Ένα από αυτά είναι η απώλεια των αρχείων όταν το container αποτύχει λόγω βλάβης. Ο kubelet θα επανεκκινήσει το container χωρίς να διατηρήσει την τελευταία του κατάσταση. Ένα δεύτερο πρόβλημα προκύπτει όταν τα αρχεία διαμοιράζονται στα πλαίσια ενός Pod. Τα προβλήματα αυτά καταπολεμώνται με τις χωρητικότητες – *Volumes* του Kubernetes.

3.6.1 PersistentVolumes

Ένας πόρος τύπου *PersistentVolume* είναι ένα κομμάτι αποθηκευτικού χώρου μέσα στη συστάδα, το οποίο παρέχεται από το διαχειριστή αυτής ή από κάποιο σύστημα δυναμικής παροχής. Τέτοιοι πόροι είναι πρόσθετα στοιχεία του cluster, τα οποία, όμως, είναι ανεξάρτητα του κύκλου ζωής των ακάτων που τα χρησιμοποιούν. Αυτό σημαίνει ότι και να αποτύχει κάποιο Pod, τα αρχεία του PV διατηρούνται ακέραια.

Πόροι αυτής της κατηγορίας παρέχονται έπειτα από αίτημα του χρήστη ή της εφαρμογής μέσω των *PersistentVolumeClaims*, τα οποία μπορούν τα αντιστοιχιστούν στα Pods της συστάδας. Όπως τα Pods καταναλώνουν πόρους των Nodes, έτσι και τα PersistentVolumesClaims, καταναλώνουν PVs. Τα αιτήματα μπορούν να προσδιορίσουν μέγεθος αποθηκευτικού χώρου και μεθόδους πρόσβασης, όπως ReadWriteOnce και ReadWriteMany.

Όπως αναφέρθηκε υπάρχουν δύο τρόποι παροχής των δεδομένων πόρων. Κατά τον στατικό τρόπο ο διαχειριστής του cluster θα πρέπει να δημιουργήσει χειροκίνητα ένα πλήθος από PVs, τα οποία θα καταναλωθούν από τυχόν PVCs, εφόσον ικανοποιούν τις προδιαγραφές τους. Αξίζει να σημειωθεί ότι σε αυτήν την περίπτωση ελλοχεύει ο κίνδυνος είτε να μη δημιουργηθούν αρκετά PVs, για να εξυπηρετήσουν την εφαρμογή, είτε να δημιουργηθεί ένα μεγάλο πλήθος αυτών, γεγονός που ισοδυναμεί με μη αποδοτική εκμετάλλευση των διαθέσιμων πόρων. Κατά το δυναμικό τρόπο παροχής, αναπτύσσεται στη συστάδα ένα Pod δυναμικού παρόχου, ο οποίος παρακολουθεί για PVCs. Όταν και μόνο όταν ανιχνεύσει κάποιο, δημιουργεί επί τόπου ένα PV που εξυπηρετεί το αίτημα.

3.6.2 EphemeralVolumes

Κάποιες εφαρμογές δεν έχουν ανάγκη από μόνιμους χώρους αποθήκευσης. Ένα παράδειγμα είναι υπηρεσίες τύπου cache, οι οποίες μετακινούν δεδομένα, που δεν είναι συνεχώς απαραίτητα, από τη μνήμη RAM στο δίσκο. Άλλες εφαρμογές απαιτούν μόνο την ανάγνωση αρχείων διαμόρφωσης, τα οποία συνήθως προσπελαύνουν μία φορά στον κύκλο ζωής τους. Πόροι τύπου *EphemeralVolumes*, εξυπηρετούν τέτοιες απαιτήσεις λειτουργίας. Τα EVs είναι προφανώς πιο ευέλικτα από τα PVs, καθώς ακόμα και αν μία άκατος αποτύχει, δεν απαραίτητο να αντιστοιχισθεί σε συγκεκριμένο χώρο αποθήκευσης και δει κόμβο.

4

Μη – Σχεσιακές Βάσεις Δεδομένων

Ο όρος NoSQL χρησιμοποιήθηκε για πρώτη φορά το 1998 για μία βάση δεδομένων η οποία δε διέθετε διεπαφή SQL και απέκτησε μεγαλύτερη βαρύτητα τη δεκαετία του 2000, λόγω της ανάπτυξης του διαδικτύου. Η αυξημένη δημοτικότητα του παγκοσμίου ιστού δημιούργησε την ανάγκη για βάσεις δεδομένων που μπορούν να ανταποκρίνονται στην κλίμακα του μεγέθους αυτού και να διαχειρίζονται αποτελεσματικά τα τεράστια μεγέθη δεδομένων που παράγει.

4.1 Κίνητρα

Μερικά από τα χαρακτηριστικά των σχεσιακών βάσεων δεδομένων περιλαμβάνουν ισχυρές γλώσσες δηλωτικού προγραμματισμού, σχήμα και metadata δεδομένων, διασφάλιση συνοχής και συνέπειας, συγχρονισμό λειτουργιών για πολλαπλούς χρήστες. Από τα παραπάνω προκύπτει ότι οι σχεσιακές βάσεις είναι σχεδιασμένες προς συνοχή και ασφάλεια δεδομένων, χαρακτηρστικό απαραίτητο για ασφαλιστικό ή τραπεζικό λογισμικό, λόγου χάριν. Παρόλα αυτά, ο έλεγχος της ανθεκτικότητας των δεδομένων απαιτεί υψηλά ποσοστά υπολογιστικής ισχύος, γεγονός το οποίο φέρνει ένα σύστημα διαχείρισης σχεσιακών βάσεων στα όριά του, όταν καλείται να επεξεργαστεί μεγάλο όγκο δεδομένων.

Σε ένα πρακτικό σενάριο, η επεξεργασία δεδομένων που είναι προσανατολισμένη στη διασφάλιση της συνοχής αποτελεί τροχοπέδη για μεγάλο όγκο αυτών, ειδικά σε περιπτώσεις και εφαρμογές που χαρακτηρίζονται από απαιτήσεις υψηλής διαθεσιμότητας, όπως είναι τα κοινωνικά δίκτυα. Προς κάλυψη της συγκεκριμένης απαίτησης, εντάθηκαν οι προσπάθειες, κυρίως από την κοινότητα ανοιχτού λογισμικού, για δημιουργία κατανεμημένων βάσεων NoSQL.

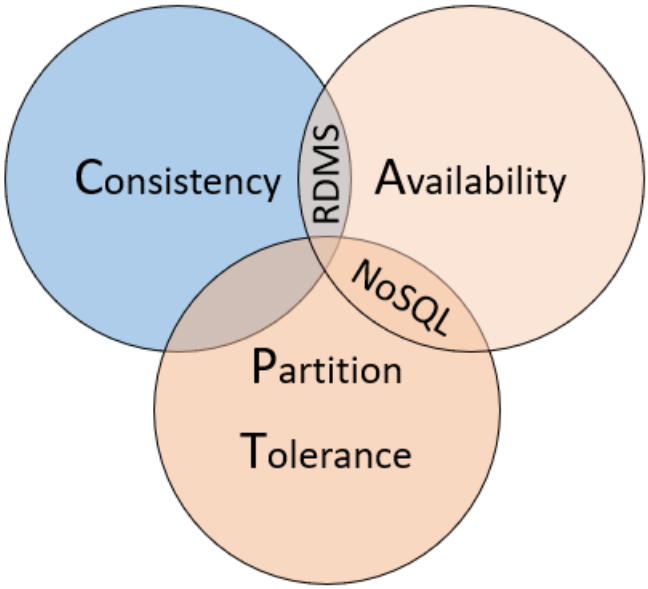
4.2 Χαρακτηριστικά

Ένα από τα κύρια στοιχεία που ξεχωρίζει τις μη σχεσιακές βάσεις δεδομένων είναι η απλούστευση της πολυπλοκότητας. Πιο συγκεκριμένα, οι ιδιότητες ACID (atomicity, consistency, isolation, durability) που δύναται να χαρακτηρίζουν τα συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων αυξάνουν τις υπολογιστικές απαιτήσεις, γεγονός που για δεδομένες εφαρμογές δεν είναι απαραίτητο. Επίσης, αποφεύγεται και η αντιστοίχιση εννοιών δεδομένων σε σχεσιακά αντικείμενα, υπολογισμός που δομές δεδομένων χαμηλής πολυπλοκότητας δε χρειάζεται. Αποτέλεσμα των παραπάνω είναι οι βάσεις NoSQL να χαρακτηρίζονται από υψηλή διεκπαιρεωτική ικανότητα και καλύτερες επιδόσεις για εφαρμογές που διαχειρίζονται μεγάλο όγκο δεδομένων.

Χαρακτηριστική είναι και η ιδιότητα της οριζόντιας κλιμάκωσης πόρων και της δυνατότητάς τους να εκτελούνται σε συστήματα με ήπια υπολογιστική ικανότητα. Είναι σχεδιασμένες δη-

λαδή να μην εξαρτώνται από υλικό υψηλής διαθεσιμότητας και να προσαρμόζονται στις απαιτήσεις μέσω κλιμάκωσης και κατακερμάτισης, λειτουργίες οι οποίες μπορούν να προγραμματιστούν ώστε να έχουν και αυτόματο χαρακτήρα.

Από τα παραπάνω γίνεται προφανές ότι θυσιάζεται η αξιοπιστία για καλύτερη απόδοση, γεγονός το οποίο δεν είναι απαραίτητα αρνητικό για τύπους εφαρμογών που στοχεύουν σε γρήγορες αποκρίσεις και θετική εμπειρία χρήστη. Το θεώρημα CAP ερμηνεύει αυτόν το συμβιβασμό μέσω των εννοιών της συνοχής, της διαθεσιμότητας και της ανοχής σε βλάβη server της βάσης. Η συνοχή προβλέπει ότι όλοι οι κόμβοι της συστάδας της βάσης θα επιστρέψουν το ίδιο αποτέλεσμα δεδομένων, η διαθεσιμότητα ότι θα υπάρχουν γρήγορες ανταποκρίσεις σε κάθε αίτημα, ενώ η ανοχή σε βλάβες ότι ακόμα και αν κάποιος κόμβος αποτύχει, το σύστημα θα είναι σε θέση να ανταποκριθεί επιτυχώς σε τυχόν αιτήματα.



Σχήμα 7: Το θεώρημα CAP

Οι σχεσιακές βάσεις δεδομένων συνήθως προσφέρουν συνοχή και διαθεσιμότητα, σε αντίθεση με τις μη σχεσιακές, χαρακτηριστικά των οποίων είναι η διαθεσιμότητα και ανοχή σε βλάβες κόμβων.

4.3 Τύποι Βάσεων Δεδομένων

Οι μη σχεσιακές βάσεις μπορούν να διακριθούν στις βάσεις κλειδιού-τιμής, βάσεις προσανατολισμένες στις στήλες, βάσεις αρχείων, βάσεις γράφων και αντικειμενοστραφείς βάσεις.

4.3.1 Βάσεις Κλειδιού – Τιμής (Key – Value stores)

Το μοντέλο των συγκεκριμένων βάσεων είναι απλό, αλλά ισχυρό και αποδοτικό. Χαρακτηρίζεται από μία εύχρηστη διεπαφή προγραμματισμού εφαρμογών (API) και επιτρέπει στο χρήστη να αποθηκεύει δεδομένα με μη-σχηματικό τρόπο. Συνήθως τα δεδομένα είναι κάποιου είδος τύπου δεδομένου γλώσσας προγραμματισμού ή αντικείμενο και αποτελούνται από δύο μέρη:

μία γραμματοσειρά που αντιπροσωπεύει το κλειδί και τα δεδομένα προς αποθήκευση, συνθέτοντας έτσι ένα ζευγάρι κλειδιού και τιμής. Αυτός ο τρόπος αποθήκευσης είναι παρόμοιος με τους πίνακες κατακερματισμού, όπου κάθε κλειδί λειτουργεί σαν πίνακας περιεχομένου. Το συγκεκριμένο μοντέλο επικεντρώνεται στη δυνατότητα κλιμάκωσης, παρά στη συνοχή και για αυτό λειτουργίες ανάλυσης, ενώσεις και αθροιστικές διεργασίες παραλείπονται. Προσφέρονται ωστόσο γρήγορες αναζητήσεις και επιλογές για μαζική αποθήκευση δεδομένων. Στα αρνητικά ανήκει η έλλειψη σχήματος δεδομένων, γεγονός που κάνει πιο δύσκολη τη δημιουργία όψεων για αυτά.

Τέτοιου τύπου βάσεις μπορούν να χρησιμοποιηθούν σε περιπτώσεις που χρειάζεται να αποθηκευτεί η συνεδρία, το καλάθι αγορών ή τα αγαπημένα προϊόντα του χρήστη. Για το λόγο αυτό συναντώνται κυρίως σε εφαρμογές ηλεκτρονικών καταστημάτων και forum. Παραδείγματα βάσεων που ακολουθούν το μοντέλο κλειδιού και τιμής αποτελούν οι *Amazon DynamoDB* και *RIAK*.

4.3.1.1 Amazon DynamoDB

Η Dynamo δημιουργήθηκε από την Amazon το 2012 ως μία αποθήκη κλειδιού-τιμής υψηλής διαθεσιμότητας που χρησιμοποιήθηκε, αρχικά, για εσωτερική χρήση. Κατασκευάστηκε σαν μια προσπάθεια κανονικοποίησης των σχεσιακών μοντέλων βάσεων που βελτιστοποιεί την αποθήκευση μειώνοντας τα περιττά δεδομένα. Προσφέρει χαμηλούς και προβλέψιμους χρόνους απόκρισης για οποιαδήποτε κλίμακα δεδομένων και χρησιμοποιεί SSD για την αποθήκευση αυτών. Τα δεδομένα αντιγράφονται με σύγχρονο τρόπο σε cloud servers διαφορετικών γεωγραφικών ζωνών της Amazon, προσφέροντας υψηλή διαθεσιμότητα και ανθεκτικότητα.

4.3.1.2 RIAK

Η RIAK είναι μία κατανεμημένη βάση ανοιχτού λογισμικού που αναπτύχθηκε από τη Basho με γλώσσες προγραμματισμού C, Erlang και JavaScript. Είναι εμπνευσμένη από τη Dynamo της Amazon και ακολουθεί αρκετές από τις αρχές τις. Προσφέρει υψηλή διαθεσιμότητα, ανεκτικότητα σε βλάβη κόμβων και ανθεκτικότητα δεδομένων. Διακρίνεται στις εκδόσεις *RIAK-KV* και *RIAK-TS* που ενδείκνυται για αποθήκευση δεδομένων χρονοσειράς.

4.3.2 Αποθήκες Στηλών (Column – Oriented stores)

Οι αποθήκες στηλών αποτελούν ένα υβριδικό σχήμα βάσεων που αποτελούνται από στήλες και γραμμές σε αντίθετη με τις αμιγώς σχεσιακές βάσεις. Αν και ακολουθούν το ίδιο μοντέλο και προσθετικά χαρακτηριστικά με τις τελευταίες, δεν αποθηκεύουν τα δεδομένα τους σε τραπέζια, αλλά σε κατανεμημένες αρχιτεκτονικές. Στο συγκεκριμένο μοντέλο κάθε κλειδί σχετίζεται με ένα ή περισσότερα χαρακτηριστικά-στήλες. Τα δεδομένα αποθηκεύονται με τέτοιο τρόπο που διευκολύνονται οι αθροιστικές διεργασίες και ελαχιστοποιείται η δραστηριότητα I/O. Επίσης προσφέρεται εύκολη κλιμάκωση στην αποθήκευση, ενώ τα δεδομένα που αποθηκεύονται στη βάση βασίζονται στη σειρά ταξινόμησης της στήλης.

Ο συγκεκριμένος τύπος βάσης είναι κατάλληλος για εφαρμογές εξόρυξης δεδομένων και ανάλυσης, όπου ο τρόπος με τον οποίο αποθηκεύονται τα δεδομένα ενδείκνυται για την επεξεργασία που δύναται να υποστούν. Στα παραδείγματα τέτοιου τύπου βάσεων ανήκουν οι *Big Table* και *Cassandra*.

4.3.2.1 Big Table

Η *Big Table* κατασκευάστηκε από τη Google ως μία συμπιεσμένη βάση αποθήκευσης υψηλής απόδοσης που κυκλοφόρησε το 2005 και βασίζεται στο σύστημα αρχείων της Google (GFS). Οι γλώσσες προγραμματισμού που χρησιμοποιήθηκαν είναι οι C και C++. Προσφέρει συνέπεια, ανεκτικότητα σε βλάβες και ανθεκτικότητα. Είναι σχεδιασμένη να κλιμακώνεται κατά μήκος χιλιάδων μηχανημάτων, ενώ η αύξηση του πλήθους της συστάδας είναι εύκολη και απλη. Αποτελείται από τρία βασικά μέρη: μία βιβλιοθήκη που συνδέεται με κάθε κόμβο, έναν server-αφέντη και πολλούς εξυπηρετητές ταμπλέτας. Οι τελευταίοι χρησιμεύουν στη διαχείριση ενός συνόλου από πίνακες δεδομένων, ενώ ο master-server είναι υπεύθυνος για αλλαγές στο σχήμα δεδομένων και για την αντιστοίχιση πινάκων στους εξυπηρετητές ταμπλέτας, αντισταθμίζοντας το φόρτο του δικτύου. Σημειώνεται ότι η Big Table δε μπορεί να κατανεμηθεί εξωτερικά της Google και είναι διαθέσιμη μόνο μέσω της εφαρμογής της.

4.3.2.2 Cassandra

Η Cassandra αναπτύχθηκε από την Apache Software και κυκλοφόρησε το 2008. Η γλώσσα προγραμματισμού στη οποία βασίστηκε είναι η Java, ενώ είναι εμπνευσμένη τόσο από τη Dynamo της Amazon, όσο και από τη Big Table της Amazon, καθώς υλοποιεί έννοιες κλειδιουτιμής, αλλά και στηλών αποθήκευσης. Προσφέρει υψηλή διαθεσιμότητα, ανεκτικότητα σε βλάβες κόμβων, ανθεκτικότητα και συνέπεια. Σημειώνεται ότι διαθέτει δυναμικό σχήμα δεδομένων, ενώ οι αναγνώσεις από τη βάση είναι σημαντικά πιο αργές από τις εγγραφές σε αυτή.

4.3.3 Βάσεις Αρχείων (Document stores)

Ο συγκεκριμένος τύπος βάσεων αποθηκεύει τα δεδομένα σε μορφή αρχείων, τα οποία προσφέρουν συχνά υψηλές αποδόσεις και ικανότητα οριζόντιας κλιμάκωσης. Υπάρχει η δυνατότητα εμφώλευσης αρχείου μέσα σε αρχείου, το οποίο είναι παρόμοιο με τις καταχωρήσεις των σχεσιακών βάσεων δεδομένων (records), αλλά αισθητά πιο ευέλικτο, καθώς δε χαρακτηρίζεται από σχήμα. Το στοιχείο αυτό καθιστά την υλοποίησή τους αισθητά πιο πολύπλοκη από τις βάσεις κλειδιού-τιμής. Τα αρχεία έχουν καθορισμένη μορφοποίηση, όπως XML, PDF, JSON. Σε αντίθεση με τις σχεσιακές βάσεις που όλες οι καταχωρήσεις έχουν ακριβώς τα ίδια πεδία πληροφοριών, τα αρχεία μπορούν να έχουν εντελώς διαφορετική μορφή και πεδία στο εσωτερικό μιας αντίστοιχης συλλογής. Τα αρχεία προσπελαύνονται από τη βάση μέσω ενός μοναδικού κλειδιού, τα οποία μπορεί να είναι μια απλή γραμματοσειρά.

Οι βάσεις αρχείων ενδείκνυνται για εφαρμογές στις οποίες η μορφή των δεδομένων δεν είναι καθορισμένη και κάθε εγγραφή μπορεί να αποτελείται από διαφορετικά χαρακτηριστικά και πεδία. Προφανώς θα πρέπει να αποφεύγονται εάν η βάση πρόκειται να έχει πολλές σχέσεις και κανονικοποιήσεις μεταξύ των δεδομένων. Παραδείγματα τέτοιου τύπου βάσεων αποτελούν οι *MongoDB* και *CouchDB*, οι οποίες προσφέρονται ως υπηρεσία βάσεων (DBaaS).

4.3.3.1 MongoDB

Η MongoDB αναπτύχθηκε από την 10gen και κυκλοφόρησε το 2009. Βασίζεται στη γλώσσα προγραμματισμού C++, χαρακτηρίζεται από υψηλές αποδόσεις και προσφέρει ανθεκτικότητα, ανοχή σε βλάβες κόμβων και συνοχή δεδομένων. Μεταξύ άλλων διαθέτει λειτουργίες όπως αθροιστικά αιτήματα, πίνακες περιεχομένων (indexes) και αυτοματοποιημένο κατακερματι-

σμό. Τα δεδομένα αποθηκεύονται κυρίως σε BSON (Binary JSON) αρχεία που αποτελούνται από όνομα πεδίου, τιμή και τύπο δεδομένου. Σε σύγκριση με τα παραδοσιακά JSON αρχεία, χαρακτηρίζονται από αποδοτικότερη αποθήκευση και ταχύτερη απόκριση σε αιτήματα. Στα αρνητικά της συγκαταλέγονται οι υψηλές υπολογιστικές απαιτήσεις για δημιουργία πινάκων περιεχομένου και η αφερεγγυότητα, σε περιπτώσεις, των δεδομένων.

4.3.3.2 CouchDB

Η CouchDB αναπτύχθηκε από την Apache Software, κυκλοφόρησε το 2005 και βασίζεται στη γλώσσα προγραμματισμού C++. Χρησιμοποιεί JSON αρχεία προς αποθήκευση δεδομένων και προσφέρει RESTful API, για να δημιουργήσει και να ενημερώσει δεδομένα της βάσης. Τα αιτήματα γράφονται σε γλώσσα προγραμματισμού Javascript και η βάση διαχειρίζεται μέσω ενσωματωμένης διαδικτυακής εφαρμογής με όνομα *FULTON*. Χαρακτηρίζεται από υψηλή διαθεσιμότητα, ανεκτικότητα σε βλάβες κόμβων και ανθεκτικότητα, ενώ παράλληλα προσφέρει ταυτόχρονη πρόσβαση σε χρήστες. Η χρήση της ενδείκνυται για εφαρμογές στις οποίες εκτελούνται προγραμματισμένα αιτήματα σε διαφορετικής μορφής δεδομένων και για προγράμματα τα οποία μπορεί να μην έχουν συνεχή σύνδεση διαδικτύου, όπως οι εφαρμογές κινητών τηλεφώνων.

4.3.4 Βάσεις Γράφων (Graph stores)

Οι βάσεις γράφων αποθηκεύουν δεδομένα σε μορφή γράφων, οι οποίοι αποτελούνται από κόμβους και ακμές, όπου οι κόμβοι λειτουργούν ως αντικείμενα και οι ακμές ως σχέσεις αντικειμένων. Ο γράφος αποτελείται από ιδιότητες που σχετίζονται με τους κόμβους και χρησιμοποιεί την τεχνική *index-free-adjacency*, που σημαίνει ότι κάθε κόμβος διαθέτει έναν άμεσο δείκτη που οδηγεί σε γειτονικό κόμβο. Με την τεχνική αυτή μπορούν να προσπελαυθούν εκατομμύρια εγγραφές, δεδομένου ότι το μοντέλο δίνει έμφαση στον τρόπο που αυτές συνδέονται μεταξύ τους. Οι βάσεις γράφων προσφέρουν μία μη-σχηματική αποθήκευση ημιδομημένων δεδομένων, ενώ τα αιτήματα εκφράζονται ως διάβαση του γράφου, στοιχείο το οποίο τις καθιστά ταχύτερες από τις σχεσιακές βάσεις δεδομένων. Σημειώνεται ότι ο συγκεκριμένος τύπος βάσης συμμορφώνεται στις αρχές ACID, προσφέρει υποστήριξη απόσυρσης συναλλαγών, αλλά ο κατακερματισμός του μπορεί να παρουσιάσει πολλές δυσκολίες. Η συσταδοποίηση παρόμοιων τύπων βάσεων είναι δύσκολο να επιτευχθεί, σε αντίθεση με την αύξηση της κλίμακάς τους η οποία χαρακτηρίζεται από απλότητα και ευκολία. Αξίζει να αναφερθεί ότι οι βάσεις γράφων θα πρέπει να αποφεύγονται όταν δεν υφίστανται σχέσεις μεταξύ των δεδομένων.

Τέτοιου τύπου βάσεις μπορούν να χρησιμοποιηθούν σε μία ποικιλία εφαρμογών, όπως κοινωνικά δίκτυα, συστήματα προτάσεων, βιοπληροφορική, ασφάλεια και διαχείριση δικτύων. Ο *Neo4j* είναι ένα παράδειγμα παρόχου που προσφέρει υπηρεσίες βάσεων γράφων.

4.3.4.1 Neo4j

Η συγκεκριμένη βάση αναπτύχθηκε από τη Neo Technology, κυκλοφόρησε το 2007 και βασίζεται στη γλώσσα προγραμματισμού Java. Πρόκειται για μία βάση υψηλών αποδόσεων που προσφέρει ένα αντικειμενοστραφές και ευέλικτο δίκτυο αποθήκευσης δεδομένων. Είναι αξιόπιστη, υψηλής διαθεσιμότητας και εύκολα κλιμακούμενη. Η γλώσσα αιτημάτων που υποστηρίζει ονομάζεται *CYPHER*.

4.3.5 Αντικειμενοστραφείς Βάσεις (Object – Oriented stores)

Οι αντικειμενοστραφείς βάσεις αποθηκεύουν τα δεδομένα σε μορφή αντικειμένων, παρόμοια με τα αντικείμενα που χρησιμοποιούνται στις γλώσσες προγραμματισμού. Για αυτό το λόγο μπορούν να θεωρηθούν σαν ένας συνδυασμός αρχών αντικειμενοστραφών γλωσσών προγραμματισμού και αρχών βάσεων δεδομένων. Τέτοιου τύπου βάσεις προσφέρουν όλα τα χαρακτηριστικά των πρώτων, όπως ενθηλάκωση δεδομένων, πολυμορφισμό και κληρονομικότητα. Οι κλάσεις, αντικείμενα και τα χαρακτηριστικά τω κλάσεων θα μπορούσαν να αντιστοιχιστούν στους πίνακες, στις τούπλες και στις στήλες των σχεσιακών βάσεων δεδομένων, αντίστοιχα. Κάθε αντικείμενο χαρακτηρίζεται από ένα μοναδικό αναγνωριστικό προς αναπαράστασή του. Οι πρόσβαση σε εγγραφές καθίσταται γρήγορη λόγω της χρήσης δεικτών που χρησιμοποιούνται για την προσπέλασή τους. Αρνητικό των δεδομένων βάσεων είναι ο περιορισμός τους σε συγκεκριμένες γλώσσες προγραμματισμού και η δυσκολία κλιμάκωσης σε περίπτωση που καλυφθούν οι πόροι φυσικής μνήμης. Για αυτό το σκοπό θα πρέπει να αποφεύγονται όταν οι σχέσεις μεταξύ των δεδομένων είναι απλές.

Περιπτώσεις στις οποίες ενδείκνυται να χρησιμοποιούνται περιλαμβάνουν εφαρμογές που χαρακτηρίζονται από σύνθετες σχέσεις μεταξύ των δεδομένων και μεταβαλλόμενα σχήματα αντικειμένων. Στην πράξη, χρησιμοποιούνται κυρίως σε επιστημονικές έρευνες και στις τηλεπικοινωνίες. Η *Db4o* είναι ένα παράδειγμα βάσης που ακολουθεί το αντικειμενοστραφές μοντέλο.

4.3.5.1 Db4o

Η Db40 δημιουργήθηκε από τον Carl Rosenberger το 2000, κυκλοφόρησε το 2001 και βασίζεται στις γλώσσες προγραμματισμού Java και C#. Διαθέτει γραφική διεπαφή χρήστη με όνομα OME (Object Manager Enterprise), η οποία μπορεί να χρησιμοποιηθεί για σκοπούς διαχείρισης, οργάνωσης και προσπέλασης της βάσης. Προσφέρει τη δυνατότητα στο χρήστη να χρησιμοποιήσει αντικειμενοστραφείς γλώσσες προγραμματισμού για να αιτηθεί δεδομένα, έναντι SQL. Βασικό μειονέκτημα της είναι η απουσία μηχανισμού που επιτρέπει την εξαγωγή και εισαγωγή δεδομένων σε μορφές αρχείων, όπως JSON, XML και text, που μπορεί να προέρχονται από διαφορετικές αποθήκες.

4.4 Βελτιστοποίηση

Στην προσπάθεια οι βάσεις δεδομένων να ανταποκριθούν στις απαιτήσεις που έθετε ο μεγαλός όγκος δεδομένων, δεν δημιουργήθηκαν μόνο διαφορετικά μοντέλα αποθήκευσης, όπως αναφέρθηκε παραπάνω, αλλά αναπτύχθηκαν και τεχνικές ή αρχιτεκτονικές βελτιστοποίησης. Συστήματα βάσεων που φιλοξενούν μεγάλο όγκο δεδομένων ή χαρακτηρίζονται από απαιτήσεις γρήγορης απόκρισης συνήθως εξαντλούν τους υπολογιστικούς πόρους ενός μόνο server. Προς αντιμετώπιση του συγκεκριμένου προβλήματος υπάρχουν δύο τρόποι επίλυσης: η κατακόρυφη και η οριζόντια κλιμάκωση.

Κατά την κατακόρυφη κλιμάκωση αυξάνονται οι υπολογιστικοί πόροι του εξυπηρετητή στον οποίο τρέχει η βάση δεδομένων. Προφανώς αυτή η προσέγγιση περιορίζεται από τη διαθέσιμη τεχνολογία, καθώς είναι πιθανό ένας μονάχα server να μην καταφέρει να είναι αρκετά ισχυρός

για ένα δεδομένο σύνολο δεδομένων και τις λειτουργίες που πρέπει να εκτελεσθούν σε αυτό. Συνεπώς υπάρχει ένα άνω φράγμα στην κατακόρυφη κλιμάκωση.

Κατά την οριζόντια κλιμάκωση το σύνολο των δεδομένων διαιρείται σε πολλαπλούς εξυπηρετητές. Αν και η υπολογιστική δύναμη του εκάστοτε μηχανήματος δεν είναι πολύ υψηλή, καθένα από αυτά διαχειρίζεται μόνο ένα υποσύνολο των δεδομένων, γεγονός που καθιστά το όλο σύστημα αποδοτικότερο από έναν εξυπηρετητή με τα μέγιστα δυνατά υπολογιστικά χαρακτηριστικά. Προκείμενου να αυξηθούν οι πόροι του συστήματος αρκεί να προστεθούν παραπάνω servers στη συστάδα, μία ενέργεια η οποία είναι συνήθως φθηνότερη από την απόκτηση επεξεργαστή, μνήμης ή δίσκο τελευταίας τεχνολογίας. Μειονέκτημα της συγκεκριμένης προσέγγισης είναι η αύξηση της πολυπλοκότητας του συστήματος υποδομής και η αντίστοιχη συντήρηση που απαιτείται.

Δύο τεχνικές οριζόντιας κλιμάκωσης που χρησιμοποιούνται ευρέως στο χώρο των βάσεων δεδομένων αποτελούν εκείνες της αντιγραφής και του κατακερματισμού. Αξίζει να αναφερθεί ότι αυτές οι τεχνικές δεν αφορούν μόνο τις μη-σχεσιακές βάσεις, αλλά βρίσκουν εφαρμογές και σε σχεσιακές αποθήκες, καθώς οι αρχές λειτουργίας τους δεν περιορίζονται από την ύπαρξη σχεσιακού σχήματος που να συνδέει τα δεδομένα προς αποθήκευση.

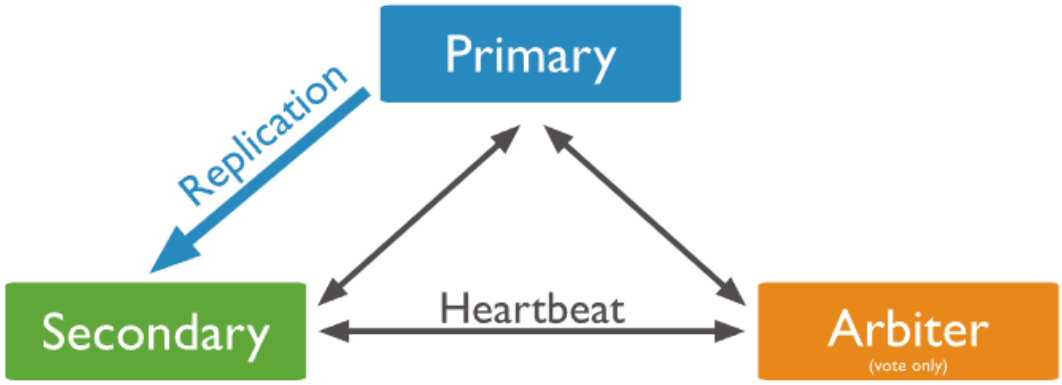
Σημειώνεται ότι οι έννοιες που θα αναπτυχθούν στη συνέχεια του κεφαλαίου προσεγγίζονται από την οπτική γωνία της MongoDB, δεδομένης της χρήσης της στο πρακτικό μέρος της παρούσας εργασίας.

4.4.1 Αντιγραφή (Replication)

Η υλοποίηση της αντιγραφής στη Mongo επιτυγχάνεται μέσω του *replica set*. Πρόκειται για μία ομάδα κόμβων, καθένας από τους οποίους διατηρεί το ίδιο σύνολο δεδομένων. Στο σύνολο μπορεί να υπάρχει και ένας κόμβος-διαιτητής (arbiter). Από τους κόμβους που φέρουν τα δεδομένα ακριβώς ένας καθίσταται κύριος (primary) κόμβος και όλοι οι υπόλοιποι δευτερεύοντες (secondary). Ο κύριος κόμβος εκτελεί όλες τις διεργασίες εγγραφής. Οι δευτερεύοντες κόμβοι αντιγράφουν ασύγχρονα τα περιεχόμενα του κυρίου και εκτελούν κατάλληλη επεξεργασία στα δεδομένα τους ώστε να φέρουν ακριβή αντίγραφα του primary node.

Κάθε χρονική στιγμή μόνο ένας κόμβος μπορεί να είναι κύριος στα πλαίσια τους συνόλου των κόμβων. Σε περίπτωση που αυτός τεθεί εκτός λειτουργίας, τότε λαμβάνει χώρα μία ψηφοφορία μεταξύ των κόμβων ώστε να προσδιοριστεί νέος κύριος κόμβος. Σε ένα τέτοιο σενάριο είναι χρήσιμος ο arbiter, ο οποίος συμβάλλει στον προσδιορισμό του νέου κύριου κόμβου όταν το πλήθος των ψήφων είναι περιττό και υπάρχει ισοπαλία. Ο κόμβος-κριτής δε φέρει δεδομένα και δε μπορεί να γίνει κύριος κόμβος.

Η αντιγραφή προσφέρει διαθεσιμότητα δεδομένων μέσω του προφανή πλεονασμού αυτών. Με πολλαπλά αντίγραφα των δεδομένων σε διαφορετικούς εξυπηρετητές η τεχνική αυτή συμβάλλει στην ανοχή σε τυχόν σφάλματα, ενώ βελτιώνει και την απόδοσης των αναγνώσεων της βάσης, αφού διαφορετικά αιτήματα μπορούν να κατανέμονται στους διαθέσιμους servers και κατά συνέπεια να αποφεύγεται η επιβάρυνση και δει η επιβράδυνση της απόκρισης ενός μονάχα κόμβου.



Σχήμα 8: Αρχιτεκτονική replication στη MongoDB

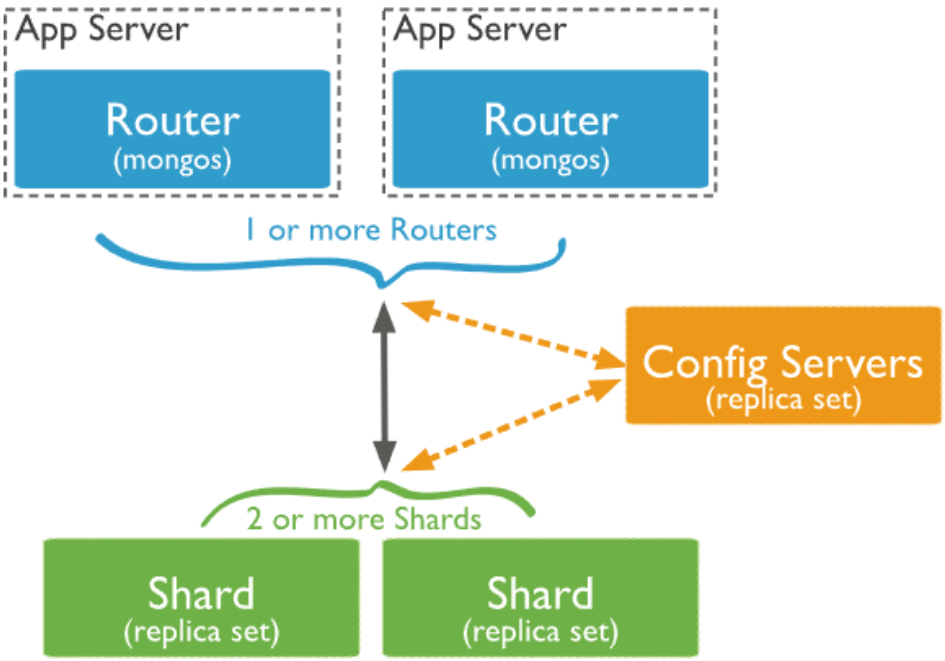
Η Mongo υποστηρίζει και αναγνώσεις καθρέφτη (mirror reads). Πιο συγκεκριμένα, μετά από βλάβη του κύριου κόμβου και εκλογή νέου, η γρήγορη μνήμη (cache memory) του νέου βασικού κόμβου δε θα ανταποκρίνεται βέλτιστα στα αρχικά αιτήματα που θα δεχθεί το σύνολο. Τα mirror reads είναι ένας τρόπος να προετοιμαστούν οι μνήμες cache των εν δυνάμει βασικών κόμβων που πρόκειται να εκλεχθούν.

4.4.2 Κατακερματισμός (Sharding)

Ο κατακερματισμός είναι επίσης ένας τρόπος κατανομής των δεδομένων σε πολλαπλά μηχανήματα και ο τρόπος με τον οποίο αποτυπώνεται στη MongoDB είναι μέσω της συστάδας κατακερματισμού. Μία τέτοια συστάδα αποτελείται από τρία βασικά μέρη: τα *shards*, τα οποία περιέχουν ένα κομμάτι των δεδομένων, το δρομολογητή mongos, ο οποίος είναι υπεύθυνος για τη διαχείριση των αιτημάτων, προσφέροντας μία διεπαφή μεταξύ της εφαρμογής και της βάσης και τους εξυπηρετητές διαμόρφωσης *config servers*, οι οποίοι κρατάνε πληροφορίες για τη διαμόρφωση της συστάδας, καθώς και metadata δεδομένων. Σημειώνεται ότι και τα τρία αυτά μέρη μπορούν να δημιουργηθούν ως replica sets, δηλαδή να ακολουθούν την τεχνική της αντιγραφής που αναπτύχθηκε στην προηγούμενη παράγραφο και πολλαπλά αντίγραφά τους να είναι διαθέσιμα προς κατανάλωση.

Η τεχνική του sharding επιτυγχάνεται μέσω του κλειδιού κατακερματισμού. Ποιο συγκεκριμένα, ένα πεδίο από τα δεδομένα επιλέγεται ως κλειδί κατακερματισμού, με βάση το οποίο θα διαιρεθεί το σύνολο σε κομμάτια. Υπάρχουν δύο τρόποι χρήσης αυτού του κλειδιού: είτε διαιρούνται τα δεδομένα σε εύρη τιμών ανάλογα με το κλειδί (ranged sharding), είτε υπολογίζεται μια γραμματοσειρά μέσω γεννήτριας συνάρτησης από το κλειδί και στη συνέχεια με βάση αυτην διαιρούνται τα δεδομένα σε εύρη τιμών (hashed sharding). Σημειώνεται ότι το πεδίο των δεδομένων που θα χρησιμοποιηθεί ως κλειδί, θα πρέπει να ανήκει σε πίνακα περιεχομένου, δηλαδή να έχει δημιουργηθεί το αντίστοιχο index.Δεδομένου ότι το sharding χρησιμοποιείται για βελτίωση των αποδόσεων ενός συστήματος αποθήκευσης δεδομένων, θα πρέπει να μελετηθεί προσεκτικά το σύνολο αποθήκευσης προτού πραγματοποιηθεί ο κατακερματισμός. Κάτι τέτοιο είναι απαραίτητο, γιατί τόσο το κλειδί, όσο και η μέθοδος κατακερματισμού επηρεάζουν άμεσα το τρόπο με τον οποίο θα διαχωριστεί το σύνολο των δεδομένων, άρα και την απόδοση του συστήματος. Μία λάθος επιλογή κλειδιού, για παράδειγμα, θα μπορούσε να

συγκεντρώσει όλο το σύνολο δεδομένων σε ένα μόνο εξυπηρετητή και συνεπώς η όλη μέθοδος να ήταν αναποτελεσματική.



Σχήμα 9: Αρχιτεκτονική sharding στη MongoDB

Πρακτικό Μέρος

5

Δυναμικά Μεταβαλλόμενο Σύστημα Αποθήκευσης

Το σύστημα, που περιγράφεται στη συνέχεια του κεφαλαίου, προβλέπει την αυτόματη μεταβολή της κλίμακας μη σχεσιακής βάσης δεδομένων, ανάλογα με το φόρτο εργασίας των εξυπηρετητών αυτής. Σκοπός του συστήματος είναι να βελτιστοποιήσει τη χρήση των διαθέσιμων υπολογιστικών πόρων και να βελτιώσει την απόκριση της βάσης δεδομένων σε εισερχόμενα αιτήματα, χωρίς να είναι απαραίτητη η ανθρώπινη παρέμβαση. Για το σκοπό αυτό σχεδιάσθηκε εξειδικευμένος πράκτορας, ο οποίος λειτουργεί σαν κομμάτι του ολικού συστήματος βάσης-εξυπηρετητή.

5.1 Απαιτήσεις Συστήματος

Βασική προϋπόθεση του συστήματος που προτείνεται είναι η ύπαρξη τουλάχιστον 3 μηχανημάτων με 6 Gb RAM το καθένα. Αυτό συμβαίνει, επειδή τα δομικά στοιχεία των πρακτόρων της MongoDB που χρησιμοποιήθηκαν απαιτούν την ύπαρξη τουλάχιστον 3 βάσεων αποκατάστασης, καθεμιά από τις οποίες δεσμεύει 5 Gb RAM. Σημειώνεται ότι οι βάσεις αυτές θα πρέπει να φιλοξενούνται σε διαφορετικά μηχανήματα. Γίνεται σαφές ότι, δεδομένου ότι το σύστημα προτείνεται ως μέσο διαχείρισης μεγάλων δεδομένων, οι ελάχιστες απαιτήσεις που αναφέρονται παραπάνω, μάλλον οριακά θα εξυπηρετήσουν το σύστημα, ειδικά εάν οι επεξεργαστές των μηχανημάτων δεν είναι υψηλών αποδόσεων.

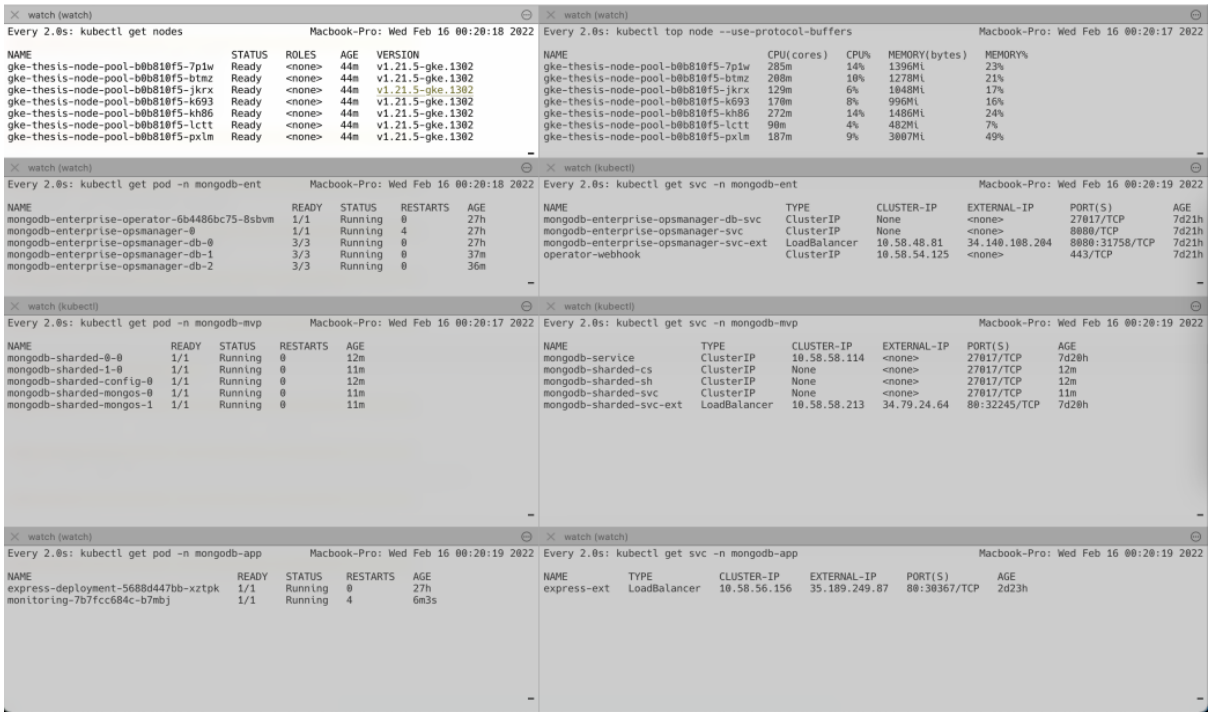
5.2 Αρχιτεκτονική

Το σύστημα αποτελείται από μία συστάδα εφτά μηχανημάτων που φιλοξενείται στην πλατφόρμα της Google, *Google Kubenretes Environment (GKE)*. Ο λόγος για τον οποίο επιλέχθηκε πάροχος υπολογιστικού νέφους είναι η κάλυψη των απαιτήσεων του συστήματος και η ευκολία προσθαφαίρεσης μηχανημάτων στα πλαίσια της συστάδας. Τα μηχανήματα είναι τύπου preemptible VM και χαρακτηρίζονται από διπύρηνους επεξεργαστές και μνήμη RAM μεγέθους 8 Gb. Σημειώνεται ότι ο συγκεκριμένος τύπος μηχανημάτων σταματούν τη λειτουργία τους κάθε 24 ώρες και αποτελούν τον οικονομικότερο τύπο πόρου που προσφέρει η πλατφόρμα.

Στα πλαίσια της εφαρμογής, πρακτικά, χρησιμοποιούνται μόνο τα 6 μηχανήματα, καθώς το τελευταίο χαρακτηρίζεται ως κόμβος-αφέντης και φιλοξενεί Ακάτους, οι οποίες είναι απαραίτητες για την εκτέλεση της εφαρμογής, δε συνθέτουν, όμως, συστατικά στοιχεία του συστήματος και του πράκτορα που αναπτύχθηκαν. Αυτός ο διαχωρισμός γίνεται, ώστε να απομονωθούν, όσο το δυνατόν περισσότερο, τα υπόλοιπα μηχανήματα και να μην επιβαρυνθούν με κάλυψη πόρων, στοιχείο που θα υπονομεύσει τη βέλτιστη λειτουργία της βάσης και του πρά-

κτορα. Ο διαχωρισμός των Ακάτων μεταξύ των μηχανημάτων έγινε με χρήση ετικετών και επιλογέων πλοήγησης.

Βασικό στοιχείο του συστήματος είναι η βάση δεδομένων. Στα πλαίσια της εργασίας χρησιμοποιήθηκε ένα sharded cluster της MongoDB. Η αρχική κατάσταση του συστήματος προβλέπει την ύπαρξη 2 shards, με μία replica δεδομένων το καθ' ένα. Υπενθυμίζεται ότι ένα sharded cluster περιλαμβάνει και από τουλάχιστον έναν config server και ένα mongos router. Στα πλαίσια της εφαρμογής έχουν πλοηγηθεί δύο mongos ως replica set, για να εξασφαλιστεί η υψηλή διαθεσιμότητα της βάσης. Σημειώνεται, επίσης, ότι απαραίτητα στοιχεία για την ομαλή εκτέλεση της βάσης, είναι ο *Enterprise Operator* και ο *Enterprise Ops Manager* της MongoDB, οι οποίοι είναι υπεύθυνοι για τη διαχείριση της συστάδας της βάσης και την παροχή ελέγχων υγείας και μετρήσεων. Τα αντίστοιχα Pods φιλοξενούνται στο master μηχάνημα της συστάδας, με εξαίρεση 3 replicas του Ops Manager που αποτελούν backup της βάσης και τοποθετούνται σε 3 τυχαία μηχανήματα από τα υπολειπόμενα 6.



Σχήμα 10: Dashboard ονοματοχώρων του συστήματος σε αρχική κατάσταση

Τα δεδομένα που χρησιμοποιήθηκαν προέρχονται από την πλατφόρμα μηχανικής μάθησης και ανάλυσης δεδομένων, *Kaggle*. Πιο συγκεκριμένα, αποτελούν δεδομένα υγειονομικού χαρακτήρα και αντιστοιχούν σε μετρήσεις ποδηλατικών διαδρομών που προέρχονται από αντίστοιχη εφαρμογή smartphone που διαχειρίζεται η πολιτεία του Chicago στα πλαίσια προώθησης εναλλακτικών τρόπων μετακίνησης. Κάθε διαδρομή χαρακτηρίζεται, μεταξύ άλλων, και από ένα πεδίο μοναδικού κωδικού διαδρομής, το οποίο αποτελεί προϊόν τυχαία σειράς αλφαριθμητικών χαρακτήρων. Το συγκεκριμένο πεδίο χρησιμοποιήθηκε ως κλειδί κατακερματισμού, αφού πρώτα πέρασε από τη διαδικασία του hashing, στοιχείο που διευκολύνει την εξασφάλιση ισότιμης κατανομής των δεδομένων στα δύο shards.

Εκτός των παραπάνω, σχεδιάστηκε εξ' ολοκλήρου εξυπηρετητής ως διεπαφή μεταξύ χρήστη και βάσης δεδομένων. Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε είναι η *Javascript*, ενώ για το περιβάλλον backend ανάπτυξης η *Node.js*. Αναπτύχθηκε ένα στοιχειώδες endpoint API, το οποίο υπακούει στο μοντέλο και σχήμα των δεδομένων που χρησιμοποιήθηκαν. Σημει-

ώνεται ότι ο server έχει συνδεθεί με κατάλληλο τρόπο με τη βάση, ώστε να υποστηρίζεται το διάβασμα αρχείων από κόμβο που χαρακτηρίζεται από την ταχύτερη δυνατή απόκριση, ανεξάρτητα της κυριότητάς του, εάν δηλαδή είναι primary ή secondary. Στην υλοποίηση που θα μελετηθεί ο χρήστης έχει τη δυνατότητα να φιλτράρει τα δεδομένα ως προς τις ημερομηνίες έναρξης και λήξης της μέτρησης της ποδηλατικής διαδρομής. Προφανώς η υλοποίηση του server πέρασε από το στάδιο του containerization, προκειμένου να φιλοξενηθεί στον Kubernetes. Φιλοξενείται στο μηχάνημα-αφέντη του cluster.

Τελευταίο συστατικό στοιχείο του συστήματος αποτελεί ο Πράκτορας Αυτόματης Μεταβολής Κλίμακας της βάσης δεδομένων. Αναπτύχθηκε σε γλώσσα προγραμματισμού Python, και πέρασε και εκείνος από το στάδιο του containerization. Φιλοξενείται στο master κόμβο της συστάδας.

5.3 Σχεδιασμός

Για λόγους οργάνωσης δημιουργήθηκαν 3 διαφορετικοί ονοματοχώροι στο περιβάλλον του Κυβερνήτη, οι *mongodb-ent*, *mongodb-mvp* και *mongodb-app*.

Στο dashboard σου σχήματος 10 αντικατοπτρίζεται ο βασικός σχεδιασμός του συστήματος. Έχει χωριστεί σε 4 γραμμές με 2 πεδία η κάθε μία. Στην πρώτη γραμμή φαίνονται τα μηχανήματα της συστάδας και οι αντίστοιχοι πόροι επεξεργαστή και μνήμης που καταναλώνουν. Η δεύτερη γραμμή αντιστοιχίζεται στον mongodb-ent, όπου έχουν πλοηγηθεί Pods του Operator και του Ops Manager της MongoDB. Σημειώνεται ότι στο συγκεκριμένο ονοματοχώρο υπάρχει μία διεύθυνση που μπορεί να προσπελαστεί εξωτερικά από τη συστάδα του Kubernetes και φέρει το όνομα *mongodb-enterprise-opsmanager-svc-ext*. Μέσω αυτής είναι προσπελάσιμος ο Ops Manager της εφαρμογής, ο οποίος προσφέρει μετρήσεις σχετικά με τη φιλοξενούμενη βαση και επιλογές διαμόρφωσης της συστάδας της MongoDB. Η τρίτη γραμμή αφορά τον mongodb-mvp, όπου και έχει πλοηγηθεί η βάση δεδομένων της εφαρμογής. Η εξωτερικά προσπαλάσιμη διεύθυνση δεν αποτελεί προεπιλεγμένο κομμάτι του συστήματος, αλλά δημιουργήθηκε, για να φορτωθούν τα δεδομένα στη βάση. Σημειώνεται ότι η σύνδεση ουσιαστικά πραγματοποιείται μέσω των router της βάσης, mongos. Αξίζει, επίσης, να αναφερθεί ότι εξωτερικές διευθύνσεις μπορούν να δημιουργηθούν, τόσο για τους εξυπηρετητές διαμόρφωσης, όσο και για τα shards της βάσης, ώστε να τροποποιηθούν καταλλήλως από το διαχειριστή. Δεδομένης, ωστόσο, της ύπαρξης του Ops Manager, ο οποίος φέρει και γραφική διεπαφή, κάτι τέτοιο δεν ενδείκνυται. Τέλος, η τελευταία γραμμή αφορά τον ονοματοχώρο mongodb-app, όπου φιλοξενείται ο server, μέσω του API του οποίου χρήστες μπορούν να αιτηθούν στη βάση και ο Πράκτορας Αυτόματης Μεταβολής Κλίμακας.

5.4 Πράκτορας Αυτόματης Μεταβολής Κλίμακας

Το συγκεκριμένο σύστημα φέρει το όνομα του πράκτορα, αντικατοπτρίζοντας την ομοιότητά του με τους πράκτορες (agents) του Κυβερνήτη. Πιο συγκεκριμένα έχει το ρόλο παρατηρητή και χρησιμοποιεί το εσωτερικό API ώστε να λαμβάνει κάθε 15 δευτερόλεπτα μετρήσεις σχετικά με τους πόρους του επεξεργαστή που χρησιμοποιεί κάθε κόμβος. Το χρονικό διάστημα που επιλέχθηκε δεν είναι τυχαίο, αλλά αντιστοιχεί στο διάστημα που λαμβάνει ο kubelet τις α-

ντίστοιχες μετρήσεις. Ένα μικρότερο χρονικό διάστημα λήψης των μετρήσεων θα ήταν αχρείαστο και θα επιβάρυνε χωρίς ουσία το master μηχάνημα της συστάδας. Γίνεται, λοιπόν, προφανές ότι οποιαδήποτε απόφαση πάρει ο Πράκτορας για μεταβολή της κλίμακας της μπορεί να είναι, στη χειρότερη περίπτωση, κατά 15 δευτερόλεπτα καθυστερημένη, εάν ο ρυθμός του δεν ευθυγραμμιστεί με εκείνον του kubelet.

Για τη λειτουργία του Πράκτορα και δεδομένου ότι χρησιμοποιεί το εσωτερικό API, τόσο, για να λαμβάνει πληροφορίες για την κατάσταση του cluster, όσο και για να τροποποιεί τους πόρους που έχουν πλοηγηθεί, είναι απαραίτητος τουλάχιστον ένας *Service Account*, ο οποίος δίνει τα αντίστοιχα δικαιώματα πρόσβασης. Κατά την πλοήγηση του Deployment του Πράκτορα δηλώνεται ρητά ότι η αντίστοιχη Άκατός του θα εκτελεσθεί κάτω από τα δεδομένα δικαιώματα.

Στα πλαίσια της εργασίας, ο Πράκτορας ανιχνεύει κατάσταση όπου τουλάχιστον δύο μηχανήματα-εργάτες της συστάδας είναι υπερφορτωμένα, δηλαδή χρησιμοποιούν το 50% της υπολογιστικής δύναμης του επεξεργαστή τους. Το ποσοστό αυτό δεν είναι τυχαίο αλλά προκύπτει σύμφωνα με τα χαρακτηριστικά των μηχανημάτων της συστάδας, τα οποία καλύπτουν όλη την υπολογιστική τους δύναμη, όταν 2 χρήστες αιτούνται ταυτόχρονα στο σύστημα. Το κατώφλι αυτό επιλέχθηκε στα πλαίσια της διπλωματικής εργασίας για την προσομοίωση ενός συγκεκριμένου σεναρίου. Προφανώς, ο πράκτορας μπορεί να μεταβληθεί αναλόγως, για να εξυπηρέτησει διαφορετικών ορίων. Σε ένα τέτοιο σενάριο πυροδοτεί άμεσα την αύξηση των replicas των shards της βάσης κατά 1. Σημειώνεται ότι μετά την εντολή που θα δώσει μέσω του εσωτερικού API, ο πράκτορας μένει αδρανής για 5 λεπτά. Αυτό γίνεται, για να υλοποιηθούν οι απαραίτητες αλλαγές στη βάση ομαλά και να μην προκύψει ασταθής κατάσταση από τυχόν συνεχή αιτήματα μεταβολής της κλίμακας. Κάτι τέτοιο θα επιβάρυνε, επίσης, τους δρομολογητές δικτύου της βάσης, οι οποίοι θα έβγαιναν εκτός λειτουργίας, γεγονός που σημαίνει ότι για ένα χρονικό διάστημα δε θα εξυπηρετούνταν αιτήματα.

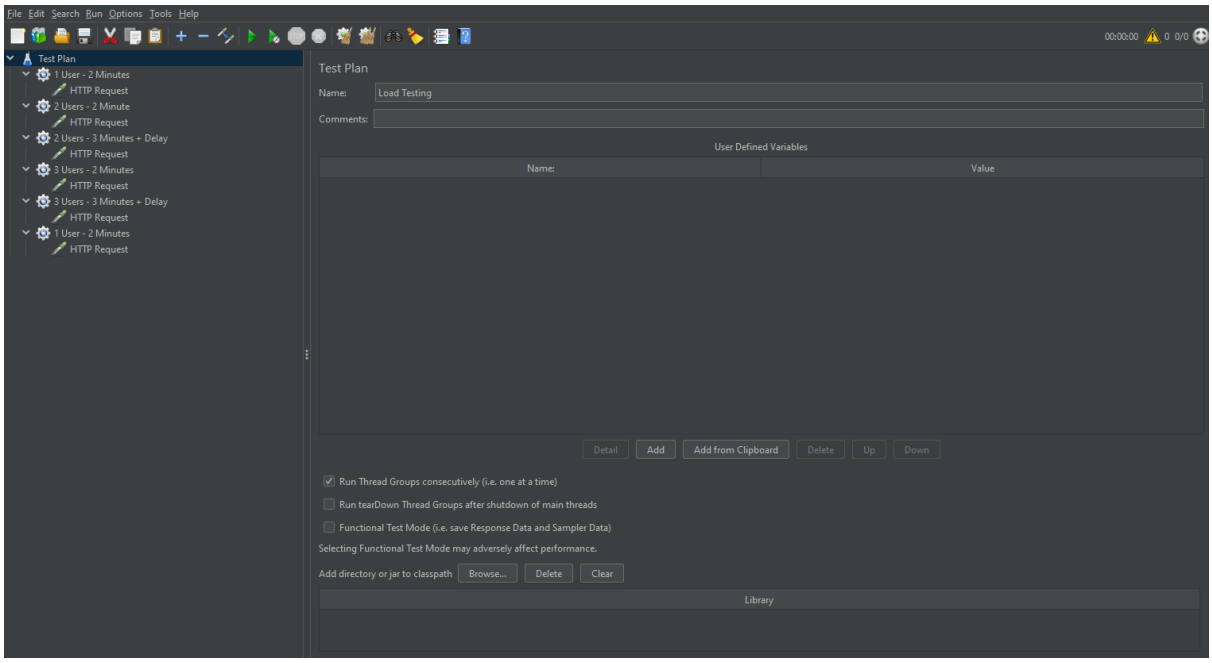
Εντελώς αντίστοιχα με το σενάριο αύξησης, ο πράκτορας μπορεί να δώσει εντολή για μείωση των replicas. Πιο συγκεκριμένα, εάν όλα τα nodes χρησιμοποιούν λιγότερο του 30% της υπολογιστικής τους δύναμης, ο πράκτορας πυροδοτεί τη μείωση της κλίμακας της βάσης και πάλι κατά ένα replica, μεταβαίνοντας στη συνέχεια στην κατάσταση αδράνειας. Και εδώ τα ποσοστά μπορούν να προσαρμοστούν σε διαφορετικές ανάγκες. Προφανώς και στα δύο σενάρια λαμβάνουν χώρα οι απαραίτητοι έλεγχοι ώστε να μην παραβιάζονται τα κατώτερα και ανώτερα επίπεδα των replicas της βάσης.

Σημειώνεται πως είναι ωφέλιμο το σύστημα να εξελιχθεί στη μέγιστη κλίμακα που μπορεί να υποστηρίξει η συστάδα, εκτός του πλαισίου της εφαρμογής και στη συνέχεια να επιστρέψει στην επιθυμητή αρχική κατάσταση. Αυτό συμβαίνει ώστε όλα τα μηχανήματα να αποκτήσουν εκτός από τα απαραίτητα αρχεία για την εκτέλεση των Ακάτων και τα αντίγραφα δεδομένα των βάσεων. Μία τέτοια αρχικοποίηση θα επιταχύνει την πλοήγηση των αντιγράφων της βάσης, όταν αυτό χρειαστεί.

5.5 Μελέτη

Προς μελέτη της απόδοσης του συστήματος χρησιμοποιήθηκε το λογισμικό προσομοίωσης φόρτου εργασίας *JMeter*. Πρόκειται για λογισμικό που στέλνει αιτήματα στην επιθυμητή διεύ-

θυνση IP ακολουθώντας το μοντέλο και τις προτιμήσεις του χρήστη. Υπενθυμίζεται ότι τα δεδομένα μπορούν να φιλτραριστούν, μέσω του API που κατασκευάστηκε, με τη βοήθεια των παραμέτρων ημερομηνίας έναρξης και λήξης μέτρησης ποδηλατικής διαδρομής. Προς προσομοίωση ενός αντιπροσωπευτικού σεναρίου χρησιμοποιήθηκαν οι τυχαίες μεταβλητές του εν λόγω framework, μέσω των οποίων κάθε αίτημα είχε τυχαίες και διαφορετικές μεταξύ τους ημερομηνίες. Για την εξασφάλιση αποδοτικών queries, ικανοποιήθηκε ο περιορισμός κατά τον οποίο η ημερομηνία έναρξης είναι πάντα μικρότερη της ημερομηνίας λήξης.



Σχήμα 11: Σενάριο προσομοίωσης χρηστών στο περιβάλλον του JMeter

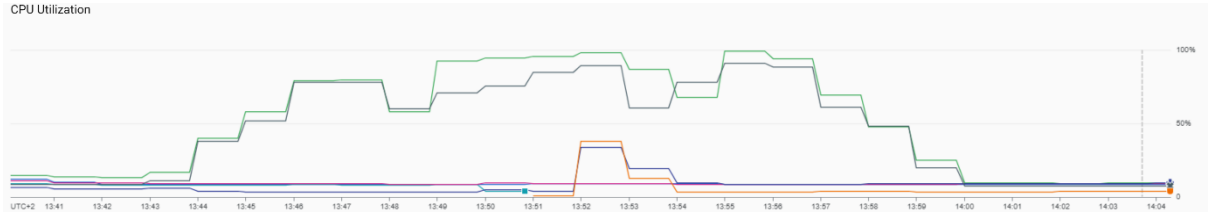
Το σενάριο προσομοίωσης διαρκεί 15 λεπτά και είναι δομημένο ως εξής:

- για 2 λεπτά 1 χρήστης στέλνει συνεχόμενα αιτήματα στο σύστημα
- για 2 λεπτά 2 χρήστες στέλνουν συνεχόμενα αιτήματα στο σύστημα
  - παύση 30 δευτερολέπτων
- για 3 λεπτά 2 χρήστες στέλνουν συνεχόμενα αιτήματα στο σύστημα
- για 2 λεπτά 3 χρήστες στέλνουν συνεχόμενα αιτήματα στο σύστημα
  - παύση 30 δευτερολέπτων
- για 3 λεπτά 3 χρήστες στέλνουν συνεχόμενα αιτήματα στο σύστημα
- για 2 λεπτά 1 χρήστης στέλνει συνεχόμενα αιτήματα στο σύστημα

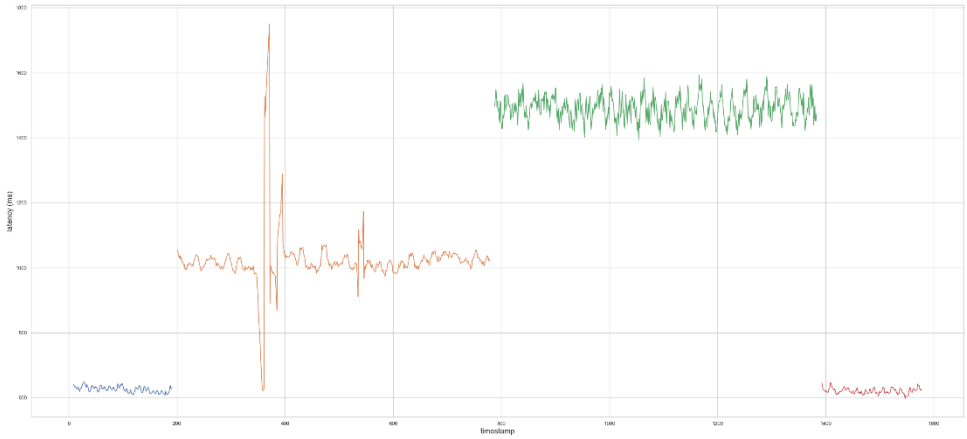
Αξίζει να παρατηρηθεί ότι η παύση των 30 δευτερολέπτων δύναται να προσομοιώσει ένα πραγματικό σενάριο, ενώ παράλληλα μπορεί να αποδειχθεί ευεργετική για την απόδοσης της βάσης. Υπενθυμίζεται ότι είναι ενεργοποιημένη η λειτουργία των *nearest reads*, δηλαδή η Mongo εξυπηρετεί τα αιτήματα από εκείνους τους κόμβους που μπορούν να απαντήσουν πιο γρήγορα, ανεξάρτητα από την κυριότητα τους, εάν δηλαδή είναι primary ή secondary. Εάν η παύση, αυτή, δεν παρεμβαλλόταν μεταξύ των σεναρίων, τα υπερφορτωμένα μηχανήματα μπορεί να οδηγούσαν σε μία λανθασμένη εκτίμηση από τη Mongo, η οποία δε θα ήταν αντιπροσωπευτική της πραγματικής απόκρισης που χαρακτηρίζουν τα εκάστοτε μηχανήματα.

5.5.1 Προσομοίωση απουσία του Πράκτορα

Όπως αναφέρθηκε η αρχική κατάσταση του συστήματος αποτελείται από 2 shards, κάθε ένα από τα οποία έχουν 1 replica. Συνολικά, δηλαδή χρησιμοποιούνται 2 από τα 6 μηχανήματα-εργάτες που είναι διαθέσιμα. Στα πλαίσια της συγκεκριμένης προσομοίωσης δε χρησιμοποιείται ο Πράκτορας Αυτόματης Μεταβολής, επομένως η κλίμακα του συστήματος δε θα μεταβληθεί.



Σχήμα 12: Διάθεση πόρων στο σύστημα απουσία του Πράκτορα



Σχήμα 13: Απόκριση συστήματος απουσία του Πράκτορα

Στο αντίστοιχο διάγραμμα διάθεσης πόρων γίνεται εμφανές ότι μόνο 2 από τα 6 μηχανήματα εκμεταλλεύονται από το σύστημα. Όταν δύο χρήστες αιτούνται στη βάση οι επεξεργαστές των αντίστοιχων κόμβων χρησιμοποιούνται στο 65%-85%. Με την αύξηση των χρηστών από 2 σε 3, η κάλυψη της υπολογιστικής ισχύς αυξάνεται στο 75%-100%. Σημειώνεται ότι οι παύσεις των αιτημάτων που λαμβάνουν χώρα κατά την προσομοίωση δε φαίνεται να επιδρούν ευεργετικά στην απόδοση των μηχανημάτων, τα οποία αγγίζουν και πάλι τα υψηλά ποσοστά κάλυψης πόρων όταν τα αιτήματα επανεκκινήσουν. Αξίζει, επίσης να αναφερθεί ότι η αύξηση κατανά λωσης που παρατηρείται σε ακόμα δύο μηχανήματα, οφείλεται σε παύση λειτουργίας των εκάστοτε κόμβων και δε συνδέονται με το σενάριο προσομοίωσης.

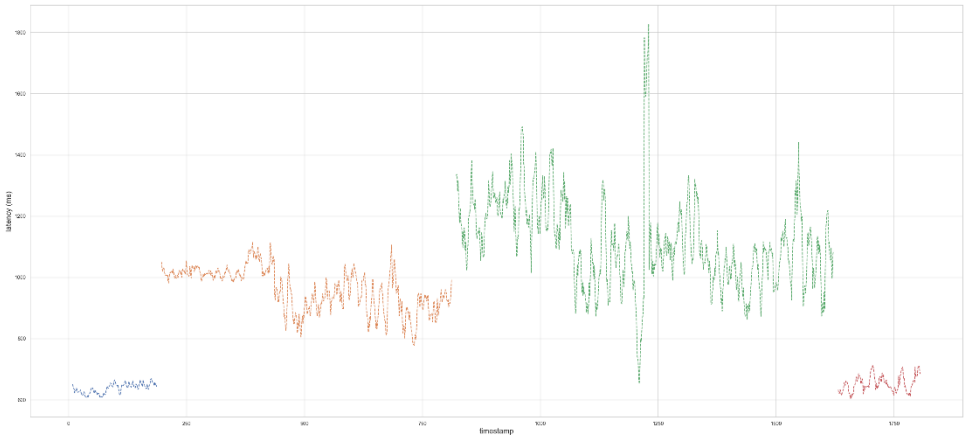
Η υπερφόρτωση των εξυπηρετητών των αντίστοιχων κόμβων αποτυπώνεται και στο διάγραμμα απόκρισης του συστήματος. Τα στάδια προσομοίωσης έχουν διαχωριστεί χρωματικά σύμφωνα με το πλήθος των χρηστών, ενώ προς καλύτερη οπτικοποίηση των αποτελεσμάτων έχει χρησιμοποιηθεί ο μέσος όρος 10 δειγμάτων της χρονοσειράς. Είναι εμφανές ότι όσο αυξάνεται ο όγκος εργασίας, η ταχύτητα απόκρισης του συστήματος ελαττώνεται. Πιο συγκεκριμένα, ο μέσος χρόνος απόκρισης για ένα χρήστη ισούται με 639.74 ms, για 2 χρήστες με 1024.35 ms, ενώ για 3 χρήστες αγγίζει τα 1521.43 ms. Προφανώς, όταν μετά το βασικό φόρτο εργασίας οι

χρήστες που αιτούνται στο σύστημα μειωθούν σε ένα, παρατηρείται ίδιας τάξης απόκριση με το αρχικό κομμάτι της προσομοίωσης.

5.5.2 Προσομοίωση παρουσία του Πράκτορα

Ακριβώς το ίδιο σενάριο προσομοίωσης εκτελέσθηκε για το ίδιο σύστημα μετά την πλοήγηση του Πράκτορα Αυτόματης Μεταβολής Κλίμακας. Από το αντίστοιχο διάγραμμα διάθεσης πόρων του συστήματος φαίνεται η αποδοτικότερη εκμετάλλευση των διαθέσιμων πόρων, ενώ μπορούν να γίνουν αντιληπτά και τα στάδια της προσομοίωσης που ακολουθήθηκαν.

Σχήμα 14: Διάθεση πόρων στο σύστημα παρουσία του Πράκτορα

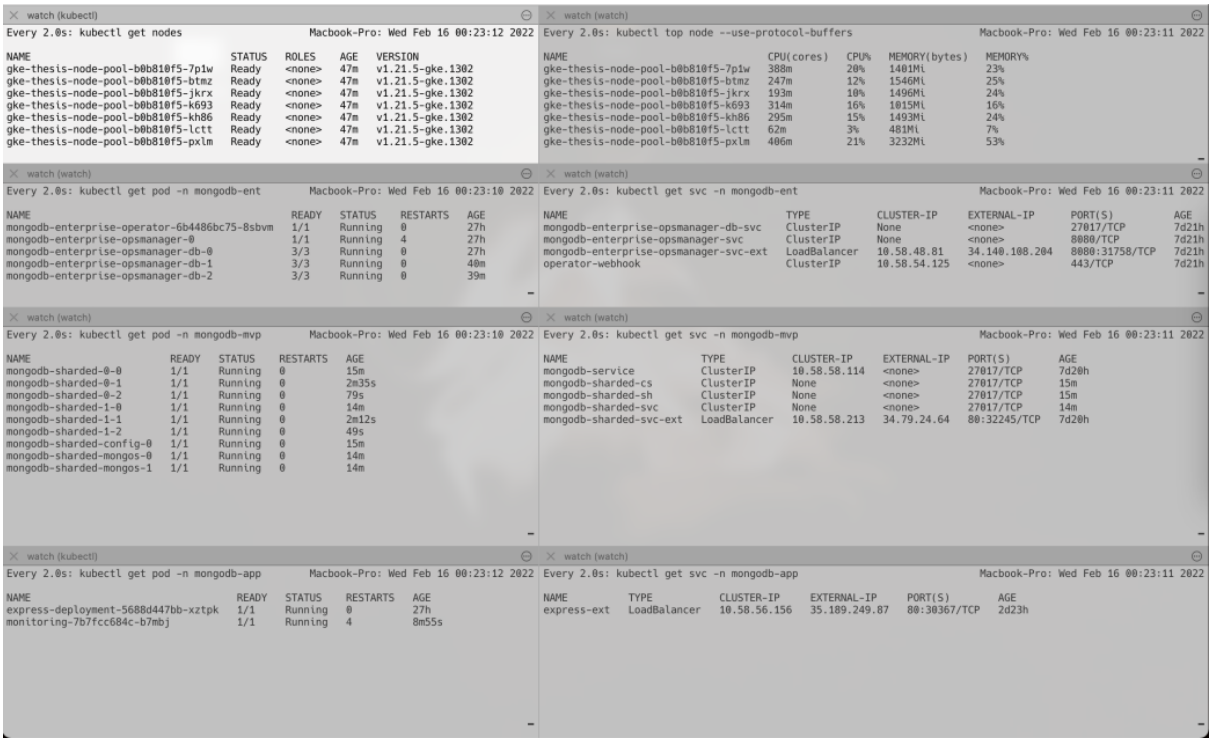


Σχήμα 15: Απόκριση συστήματος παρουσία του Πράκτορα

Πιο συγκεκριμένα, με την αύξηση των χρηστών από 1 σε 2, η κάλυψη της υπολογιστικής ισχύος των αντίστοιχων κόμβων έφτασε στο 65%-90%. Ο Πράκτορας ανίχνευσε το παραπάνω συμβάν και πυροδότησε την αύξηση των replicas της συστάδας της βάσης κατά ένα. Συνεπώς τα ενεργά μηχανήματα είναι πλέον 4 και καταναλώνουν το 50%-65% των υπολογιστικών τους πόρων. Δεδομένου ότι τουλάχιστον 2 από αυτά ξεπερνούν το 50% και μετά το πέρασμα των 3 λεπτών αδράνειας του Πράκτορα, πυροδοτείται ακόμα μία αύξηση των replicas που ισούνται τελικά με 3. Το αποτέλεσμα είναι όλα τα μηχανήματα της συστάδας να συμβάλλουν στην εξυπηρέτηση του φόρτου εργασίας με τα ποσοστά κάλυψης των επεξεργαστών τους να κυμαίνονται από 35% έως 55%. Τέλος, όσο το πλήθος των χρηστών που αιτούνται στο σύστημα μειώνεται, ο Πράκτορας πυροδοτεί και την αντίστοιχη πτώση των replicas, εξοικονομώντας αχρείαστη υπολογιστική ισχύ. Με τη λήξη των αιτημάτων το σύστημα επιστρέφει στην αρχική του κατάσταση που αποτελείται από 2 shards, με ένα replica το καθ’ ένα.

Σημειώνεται ότι η αύξηση των replicas από 2 σε 3 έλαβε χώρα πριν το σενάριο των 3 χρηστών, αφού τα μηχανήματα στο οποία πλοηγήθηκαν τα νεότερα αντίγραφα της βάσης ήταν υπερφορτωμένα από την αρχικοποίηση των απαραίτητων Ακάτων. Αν και το σενάριο δημιουργήθηκε για να εξετάσει κλιμακωτά τα διάφορα σενάρια, γίνεται αντιληπτό ότι το σύστημα ακόμα ανταποκρίνεται ιδανικά στο φόρτο εργασίας.

Η αποδοτικότερη εκμετάλλευση των πόρων αποτυπώνεται και πάλι στο διάγραμμα απόκρισης του δυναμικού, πλέον, συστήματος. Γίνεται αντιληπτό ότι το σύστημα ανταποκρίνεται καλύτερα στο φόρτο των 2 και 3 χρηστών όταν αυξηθεί η κλίμακά του, έπειτα από την ενέργεια του Πράκτορα. Οι χρονικές στιγμές που λαμβάνει χώρα η αύξηση των replicas είναι μάλιστα εμφανής στις μεταπτώσεις των πορτοκαλί και πράσινων χρονοσειρών. Συνοπτικά, ο μέσος χρόνος απόκρισης για ένα χρήστη ισούται με 638.07 ms, για 2 χρήστες με 943.48 ms, ενώ για 3 χρήστες με 1116.03 ms. Σημειώνεται ότι για το χρονικό διάστημα όπου ένας χρήστης αιτείται στο σύστημα με 3 αντίγραφα βάσης, ο μέσος χρόνος απόκρισης πριν την ελάττωση της κλίμακας φτάνει στα 629 ms. Ο Πράκτορας εκτιμά, ωστόσο, ότι χρησιμοποιούνται περισσότεροι υπολογιστική πόροι από τις απαιτήσεις και για αυτό δίνει εντολή για μείωση των replicas.

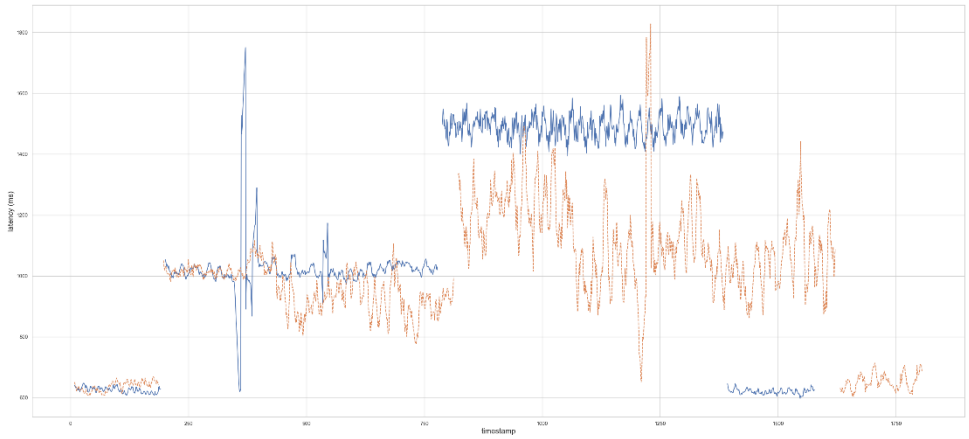


Σχήμα 16: Dashboard ονοματοχώρων του συστήματος στην τελική κατάσταση

6

Συμπεράσματα

Γίνεται σαφές ότι το σύστημα ανταποκρίνεται αποδοτικά στον αυξημένο φόρτο εργασίας, μεταβάλλοντας την κλίμακα της βάσης. Δεδομένης της χρήσης των περισσοτέρων μηχανημάτων η μείωση του χρόνου απόκρισης φαίνεται λογική, ωστόσο αυτή γίνεται με αυτόματο τρόπο και χωρίς ανθρώπινη παρέμβαση. Πρακτικά, η απόφαση του Πράκτορα ανάλογα με την κάλυψη υπολογιστικών πόρων ορίζει ένα άνω και κάτω φράγμα αποδεκτών χρόνων απόκρισης του συστήματος. Όταν το σύστημα ανταποκρίνεται στο πάνω αποδεκτό όριο, τα replicas της βάσης αυξάνονται με αυτόματο τρόπο, για να ρίξουν το χρόνο απόκρισης σε αποδεκτά όρια, με αντάλλαγμα, όμως τη χρήση παραπάνω μηχανημάτων. Αντίστοιχα, όταν πληρούνται οι απαιτήσεις του κάτω ανεκτού χρονικού ορίου και χρησιμοποιούνται παραπάνω μηχανήματα από ότι χρειάζεται, ο Πράκτορας μειώνει τα αντίγραφα της βάσης προς εξοικονόμησης πόρων.



Σχήμα 17: Σύγκριση απόκρισης συστήματος χωρίς (μπλε χρώμα) και με τον Πράκτορα (πορτοκαλί)

Αξίζει επίσης να παρατηρηθεί ότι το αυτοματοποιημένο σύστημα εξυπηρετεί μεγαλύτερο πλήθος αιτημάτων στο χρονικό διάστημα των 15 λεπτών του σεναρίου προσομοίωσης. Αυτό απότυπώνεται στο παραπάνω διάγραμμα, όπου το σύστημα απουσία του Πράκτορα αντιπροσωπεύεται από τη χρονοσειρά μπλε χρώματος, ενώ το αυτοματοποιημένο από εκείνη με πορτοκαλί. Κάτι τέτοιο φαίνεται λογικό, δεδομένου ότι τα αιτήματα παίρνουν λιγότερο χρόνο να ικανοποιηθούν. Προς αποτύπωση της συγκεκριμένης συμπεριφοράς, τα διαγράμματα φέρουν αύξοντα αριθμό αιτήματος στον άξονα των x και όχι χρονική σφραγίδα.

6.1 Μελλοντικές Προεκτάσεις

Μελλοντικά το σύστημα θα μπορούσε να εξελιχθεί με την ενσωμάτωση και άλλων μετρικών, εκτός από εκείνη της κάλυψης των επεξεργαστών των μηχανημάτων της συστάδας, στη λογική

λειτουργίας του συστήματος. Πιο συγκεκριμένα, ο Κυβερνήτης παρέχει πληροφορίες σχετικά με την κάλυψη της μνήμης RAM για το κάθε μηχάνημα, στοιχείο το οποίο μπορεί να αποδειχθεί χρήσιμο στην απόφαση του Πράκτορα για αύξηση ή μείωση της κλίμακας. Επίσης, εξωτερικά εργαλεία θα μπορούσαν να αναπτυχθούν προς απόκτηση πλέον εξειδικευμένων μετρικών σχετικά με την απόδοση των μηχανημάτων, δεδομένου ότι εσωτερικά ο Kubernetes δεν προσφέρει κάτι παραπάνω.

Επίσης, προοπτικές φαίνεται να έχει η αυτόματη κλιμάκωση των shards της βάσης. Η συγκεκριμένη προσέγγιση χαρακτηρίζεται από αυξημένη πολυπλοκότητα σε σύγκριση με το σύστημα που προτάθηκε παραπάνω, αλλά μπορεί να αποδειχθεί κερδοφόρα ειδικά σε μεγάλα σύνολα δεδομένων. Η λογική προβλέπει τον κατακερματισμό της βάσης κάθε φορά που καλύπτεται ένα πάνω όριο της χωρητικότητας των εκάστοτε μηχανημάτων. Προφανώς, η διαδικασία είναι υπολογιστικά ακριβότερη λόγω του αλγορίθμου του κατακερματισμού, γεγονός που σημαίνει ότι οι αλλαγές στην κλίμακα του συστήματος ίσως να μην είναι άμεσες. Αξίζει, επίσης, να αναφερθεί ότι ελλοχεύει ο κίνδυνος η βάση να τίθεται εκτός λειτουργίας μέχρι να ολοκληρωθεί η διαδικασία του κατακερματισμού, ενώ η διαδικασία μείωσης των shards είναι χρονικά απαγορευτική για εφαρμογές που απαιτούν υψηλή διαθεσιμότητα.

Τέλος, παραπάνω μελέτη θα ήταν χρήσιμη και στα άνω και κάτω όρια που χρησιμοποιεί ο Πράκτορας για μεταβολή της κλίμακας της βάσης. Στο σενάριο που παρουσιάστηκε παραπάνω, λόγου χάρη, ο Πράκτορας αύξησε τα replicas της βάσης αρκετά νωρίς λόγω της υπολογιστικής ισχύς που κατανάλωσαν τα μηχανήματα κατά την εκκίνησή των αντιγράφων της βάσης. Μία τέτοια συμπεριφορά θα μπορούσε να αποφευχθεί, εάν ο Πράκτορας ανιχνευε υψηλό φόρτο εργασίας σε ένα χρονικό διάστημα, αντί να στηρίζεται στην ανίχνευση ενός μόνο γεγονότος.

Παραρτήματα

Βιβλιογραφία

[1] A. A. Tole, Tole, and A. Adrian, "Big Data Challenges" *Database Systems Journal*, vol. 4, no. 3, pp. 31–40, 2013, Accessed: Feb. 15, 2022. Available: https://EconPapers.repec.org/RePEc:aes:dbjour:v:4:y:2013:i:3:p:31-40.

[2] J. Chen *et al.*, "Big data challenge: a data management perspective" *Frontiers of Computer Science 2013 7:2*, vol. 7, no. 2, pp. 157–164, Apr. 2013, doi: 10.1007/S11704-013-3903-7.

[3] S. Sagiroglu and D. Sinanc, "Big data: A review" *Proceedings of the 2013 International Conference on Collaboration Technologies and Systems, CTS 2013*, pp. 42–47, 2013, doi: 10.1109/CTS.2013.6567202.

[4] A. P. McAfee and E. Brynjolfsson, "Big data: the management revolution" *Harvard Business Review*, 2012.

[5] C. Pahl, "Containerization and the PaaS Cloud" *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24–31, May 2015, doi: 10.1109/MCC.2015.51.

[6] M. Stonebraker, "SQL databases v. NoSQL databases" *Communications of the ACM*, vol. 53, no. 4, pp. 10–11, Apr. 2010, doi: 10.1145/1721654.1721659.

[7] A. Meier and M. Kaufmann, "NoSQL Databases" *Springer Fachmedien Wiesbaden*, 2019. doi: 10.1007/978-3-658-24549-8_7.

[8] M. Stonebraker, "SQL databases v. NoSQL databases" *Communications of the ACM*, vol. 53, no. 4, pp. 10–11, Apr. 2010, doi: 10.1145/1721654.1721659.

[9] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes" *Communications of the ACM*, vol. 59, no. 5, pp. 50–57, May 2016, doi: 10.1145/2890784.

[10] Psomakelis E., Nikolakopoulos A., Marinakis A., Psychas A., Moulos V., Varvarigou T., & Christou A. "A scalable and semantic data as a service marketplace for enhancing cloud-based applications" *Future Internet*, vol. 12, no.5, 2020, doi: 10.3390/FI12050077.

[11] Q. Liu, W. Zheng, M. Zhang, Y. Wang, and K. Yu, "Docker-based automatic deployment for nuclear fusion experimental data archive cluster" *IEEE Transactions on Plasma Science*, vol. 46, no. 5, pp. 1281–1284, May 2018, doi: 10.1109/TPS.2018.2795030.

[12] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of Microservices for IoT Using Docker and Edge Computing" *IEEE Communications Magazine*, vol. 56, no. 9, pp. 118–123, 2018, doi: 10.1109/MCOM.2018.1701233.

[13] Moulos V., Chatzikyriakos G., Kassouras V., Doulamis A., Doulamis N., Leventakis G., Florakis T., Varvarigou T., Mitsokapas E., Kioumourtzis G., Klirodetis P., Psychas A., Marinakis A., Sfetsos T., Koniaris A., Liapis D., & Gatzioura A., "A Robust Information Life Cycle Management Framework for Securing and Governing Critical

Infrastructure Systems" *Inventions 2018,* vol. 3, no. 4, pp. 71, doi: 10.3390/INVENTIONS3040071.

[14] J. E. Pagán, J. S. Cuadrado, and J. G. Molina, "A repository for scalable model management" *Software and Systems Modeling*, vol. 14, no. 1, pp. 219–239, Feb. 2015, doi: 10.1007/S10270-013-0326-8.

[15] "IBM" *https://ibm.com.*

[16] "Docker" *https://docker.com.*

[17] "Kubernetes" *https://kubernetes.io.*

[18] "Microsoft" *https://docs.microsoft.com.*

[19] "MongoDB" *https://docs.mongodb.com.*

Κώδικες

Στο συγκεκριμένο κεφάλαιο αναλύονται και περιγράφονται τα αρχεία κώδικα που ήταν απαραίτητα για την υλοποίηση του συστήματος που προτείνεται από την εργασία.

MongoDB Enterprise Operator

Για την πλοήγηση του συστήματος της βάσης χρησιμοποιήθηκαν πόροι της MongoDB. Προαπαιτούμενα είναι ο mongodb-enterprise-operator και mongodb-enterprise-ops-manager. Ο πρώτος χρησιμοποιήθηκε αυτούσιος όπως προκύπτει από την επίσημη πηγή. Η διαμόρφωση του δευτέρου είναι η εξής:

```
apiVersion: v1
kind: Secret
metadata:
    name: opsmanager-admin
type: Opaque
data:
    Username: cm9vdA== # root
    Password: QWRtaW4tcm9vdDQy # Admin-root42
    FirstName: c29ucW8= # sonqo
    LastName: bnR1YQ== # ntua
---
apiVersion: mongodb.com/v1
kind: MongoDBOpsManager
metadata:
    name: mongodb-enterprise-opsmanager
spec:
    replicas: 1
    version: 5.0.0
    adminCredentials: opsmanager-admin
    externalConnectivity:
        type: LoadBalancer
    applicationDatabase:
        members: 3
        version: 4.4.4-ent
        podSpec:
            podTemplate:
                spec:
                    nodeSelector:
                        status: worker
                    topologySpreadConstraints:
                        - maxSkew: 1
                          topologyKey: kubernetes.io/hostname
                          whenUnsatisfiable: DoNotSchedule
                          labelSelector:
                              matchLabels:
                                  pod-anti-affinity: mongodb-
enterprise-opsmanager-db
    backup:
        enabled: false
```

Σημειώνεται ότι ο ορισμός του συγκεκριμένου πόρου ξεκινά με τη δήλωση ενός *Secret*, το οποίο περιέχει τα στοιχεία ταυτοποίησης του Ops Manager. Επίσης, φαίνεται και ο περιορισμός τοποθέτησης των Ακάτων σε διαφορετικά μηχανήματα μέσα από την ετικέτα *pod-anti-affinity*.

MongoDB Sharded Cluster

Πριν την δημιουργία του sharded cluster, εφαρμόζεται αρχείο δήλωσης ρόλων, το οποίο επιτρέπει στο mongodb-operator να βλέπει και διαχειρίζεται τους ανάλογους πόρους ανάλογα με τον ονοματοχώρο που θα πλοηγηθούν:

```
---
kind: ServiceAccount
apiVersion: v1
metadata:
    name: mongodb-enterprise-appdb
    namespace: mongodb-mvp
---
kind: ServiceAccount
apiVersion: v1
metadata:
    name: mongodb-enterprise-database-pods
    namespace: mongodb-mvp
---
kind: ServiceAccount
apiVersion: v1
metadata:
    name: mongodb-enterprise-ops-manager
    namespace: mongodb-mvp
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
    name: mongodb-enterprise-appdb
    namespace: mongodb-mvp
rules:
    - apiGroups:
          - ''
      resources:
          - secrets
      verbs:
          - get
    - apiGroups:
          - ''
      resources:
          - pods
      verbs:
          - patch
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
    name: mongodb-enterprise-appdb
    namespace: mongodb-mvp
roleRef:
    apiGroup: rbac.authorization.k8s.io
```

```
    kind: Role
    name: mongodb-enterprise-appdb
subjects:
    - kind: ServiceAccount
      name: mongodb-enterprise-appdb
      namespace: mongodb-mvp
```

Πριν την πλοήγηση της βάσης, είναι απαραίτητο να δηλωθεί ένα αρχείο διαμόρφωσης που θα συνδέει τον Ops Manager με τον πόρο προς δημιουργία. Αυτό επιτυγχάνεται μέσω του ακόλουθου αρχείο τύπου *Configural*:

```
apiVersion: v1
kind: ConfigMap
metadata:
    name: mongodb-configmap
data:
    baseUrl: http://34.140.108.204:8080
    orgId: 6201c42b2bd69048039d0e3a
    projectName: mongodb-mvp
```

Στο αρχείο αυτό παρέχεται η διεύθυνση URL του Ops Manager, το διακριτό ID του οργανισμού, καθώς και το όνομα του project, κάτω από το οποίο πρόκειται να δημιουργηθεί η συστάδα της βάσης.

Απαραίτητο, επίσης, είναι και ένα API key που παρέχει τις ανάλογες άδειας για δημιουργία και διαχείριση της συστάδας και γενικά οποιοδήποτε πόρου πρόκειται να πλοηγηθεί:

```
apiVersion: v1
kind: Secret
metadata:
    name: programmatic-api-key
type: Opaque
stringData:
    publicKey: pnmolqew
    privateKey: 92c6102b-8302-43f9-ab35-1a58d2618ca9
```

Σημειώνεται ότι το κλειδί θα πρέπει να δημιουργηθεί από την οπτική διεπαφή του Ops Manager με τα προνόμια του Project Owner.

Στη συνέχεια ακολουθεί το αρχείο διαμόρφωσης του sharded cluster:

```
apiVersion: mongodb.com/v1
kind: MongoDB
metadata:
    name: mongodb-sharded
spec:
    shardCount: 2
    mongodsPerShardCount: 1
    mongosCount: 2
    configServerCount: 1
    version: 4.4.4-ent
    opsManager:
        configMapRef:
            name: mongodb-configmap
    credentials: programmatic-api-key
    type: ShardedCluster
```

```
    persistent: true
    configSrvPodSpec:
        podTemplate:
            spec:
                nodeSelector:
                    status: master
    mongosPodSpec:
        podTemplate:
            spec:
                nodeSelector:
                    status: master
    shardPodSpec:
        podTemplate:
            spec:
                nodeSelector:
                    status: worker
                topologySpreadConstraints:
                    - maxSkew: 1
                      topologyKey: kubernetes.io/hostname
                      whenUnsatisfiable: DoNotSchedule
                      labelSelector:
                          matchLabels:
                              app: mongodb-sharded-sh
```

Πιο συγκεκριμένα, το αρχείο αυτό προμηθεύεται με το ConfigMap και το API key που περιγράφησαν παραπάνω. Επιπροσθέτως, δηλώνεται το πλήθος των shards, replicas, mongos και config servers, ενώ παρατηρείται και η απαίτηση των βάσεων να πλοηγούνται σε διαφορετικά μηχανήματα μέσα από την ετικέτα *topologySpreadConstraints*. Ο τρόπος με τον οποίο είναι δομημένη η συγκεκριμένη ετικέτα, πρακτικά πλοηγεί μονάχα ένα Pod βάσης σε κάθε διαφορετικό όνομα μηχανήματος.

Για τη λειτουργία του ολικού συστήματος, απαραίτητη είναι και η επικοινωνία της βάσης με το server της εφαρμογής, κάτι το οποίο επιτυγχάνεται με τα Services:

```
apiVersion: v1
kind: Service
metadata:
    name: mongodb-service
spec:
    selector:
        app: mongodb-sharded-svc
    ports:
        - protocol: TCP
          port: 27017
          targetPort: 27017
---
apiVersion: v1
kind: Service
metadata:
    name: mongodb-sharded-svc-ext
spec:
    selector:
        app: mongodb-sharded-svc
    type: LoadBalancer
    ports:
        - protocol: TCP
```

```
        port: 80
        targetPort: 27017
```

Στο αρχείο αυτό δηλώνονται δύο services: το πρώτο συνδέει τον εξυπηρετητή με τους δρομολογητές δικτύου της βάσης, δηλαδή του πόρους mongos. Η δεύτερη καθιστά τους δρομολογητές αυτούς προσπελάσιμους από εξωτερικές πηγές του Κυβερνήτη και του cluster. Δημιουργήθηκε για το φόρτωμα των δεδομένων στη βάση.

Τέλος, στα πλαίσια της βάσης δημιουργήθηκε και ένας χρήστης, προκειμένου να ικανοποιηθούν στοιχειώδη πρωτόκολλα ασφαλείας:

```
apiVersion: v1
kind: Secret
metadata:
    name: mongodb-user-password
type: Opaque
stringData:
    password: root
---
apiVersion: mongodb.com/v1
kind: MongoDBUser
metadata:
    name: mongodb-user
spec:
    passwordSecretKeyRef:
        name: mongodb-user-password
        key: password
    username: user
    db: admin
    mongodbResourceRef:
        name: mongodb-sharded
    roles:
        - db: admin
          name: clusterAdmin
        - db: admin
          name: dbAdminAnyDatabase
        - db: admin
          name: userAdminAnyDatabase
        - db: admin
          name: readWriteAnyDatabase
```

Πράκτορας Αυτόματης Μεταβολής

Το κώδικας για τον Πράκτορα Αυτόματης Μεταβολής παρουσιάζεται παρακάτω:

```
import time
from kubernetes import client, config


def fecth_cpu_usage(group, version, plural, label_selector):
    node_metrics = kube_api.list_cluster_custom_object(group, version,
plural, label_selector=label_selector)
    acc = []
    for metric in node_metrics['items']:
        cpu_usage = metric['usage']['cpu']
```

```
        if cpu_usage[-1] == 'n':
            cpu_usage = int(cpu_usage.strip('n')) // 1000000
        elif cpu_usage[-1] == 'm':
            cpu_usage = int(cpu_usage.strip('m'))
        else:
            raise ValueError('Unhandled CPU metric.')
        cpu_percentage = int(cpu_usage) // (2 * 10)
        acc.append(cpu_percentage)
    return acc


def fetch_mongodb_resource(group, version, plural):
    return kube_api.list_cluster_custom_object(group, version,
plural)['items'][0]


config.load_incluster_config()
kube_api = client.CustomObjectsApi()

while True:
    time.sleep(15)
    cpu_usage_list = fecth_cpu_usage('metrics.k8s.io', 'v1beta1',
'nodes', 'status=worker')
    mongo = fetch_mongodb_resource('mongodb.com', 'v1', 'mongodb')

    nodes_number = len(cpu_usage_list)
    print('Number of worker nodes identified:{}'.format(nodes_number))

    spec = mongo['spec']['mongodsPerShardCount'] *
mongo['spec']['shardCount']
    status = mongo['status']['mongodsPerShardCount'] *
mongo['status']['shardCount']

    min_mongods = 1
    max_mongods = nodes_number // 2
    curr_mongods = mongo['status']['mongodsPerShardCount']

    if spec == status: # stable state
        overworked_nodes = 0
        underworked_nodes = 0
        for metric in cpu_usage_list:
            if metric > 50:
                overworked_nodes += 1
            elif metric < 30:
                underworked_nodes += 1
        patch_body = None
        if overworked_nodes >= 2 and nodes_number > status:
            print('CHANGE NEEDED')
            if curr_mongods < max_mongods:
                patch_body = {
                    'spec': {
                        'mongodsPerShardCount' : curr_mongods + 1
                    }
                }
                print('INCREASED MONGODS')
            else:
                print('NO WORKER NODES AVAILABLE')
```

```
        elif underworked_nodes == nodes_number and curr_mongods >
min_mongods:
            patch_body = {
                'spec': {
                    'mongodsPerShardCount' : curr_mongods - 1
                }
            }
        else:
            print('OK')
        if patch_body:
            patch_resource = kube_api.patch_namespaced_custom_object(
                group='mongodb.com',
                version='v1',
                name='mongodb-sharded',
                namespace='mongodb-mvp',
                plural='mongodb',
                body=patch_body
            )
            print('Sleeping for 3 minutes')
            time.sleep(180)
```

Αποτελείται από 2 συναρτήσεις οι οποίες προσπελαύνουν το εσωτερικό API του Kubernetes και επιστρέφουν πληροφορίες σχετικά με την κάλυψη πόρων των κόμβων της συστάδας και σχετικά με τους εγκατεστημένους πόρους της βάσης. Η λογική λειτουργίας τους συναντάται στον επαναληπτικό βρόχο ο οποίος εκτελείται συνέχεια και τοποθετεί τον Πράκτορα σε αδράνεια κάθε 15 δευτερόλεπτα.

Προφανώς ο Πράκτορας θα πρέπει να περάσει από τη διαδικασία του containerization, προτού πλοηγηθεί. Το αντίστοιχο *Dockerfile* παρουσιάζεται ακολούθως:

```
FROM python:3

ADD monitoring/metrics.py /

RUN pip install kubernetes

CMD [ "python", "-u", "./metrics.py" ]
```

Το αρχείο πλοήγησης του Πράκτορα στον Κυβερνήτη παρουσιάζεται παρακάτω:

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: monitoring
    labels:
        app: monitoring
spec:
    replicas: 1
    selector:
        matchLabels:
            app: monitoring
    template:
        metadata:
            labels:
                app: monitoring
        spec:
```

```
            serviceAccountName: sonqo
            nodeSelector:
                status: master
            containers:
                - name: monitoring
                  image: sonqo/monitoring-app:latest
```

Τέλος, όπως έχει ήδη αναφερθεί, απαραίτητες είναι οι ανάλογες άδειες προκείμενου ο Πράκτορας να έχει πρόσβαση στο εσωτερικό API του Κυβερνήτη:

```
apiVersion: v1
kind: ServiceAccount
metadata:
    name: sonqo
    namespace: mongodb-app
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
    namespace: mongodb-app
    name: sonqo-role
rules:
    - apiGroups: ['*']
      resources: ['*']
      verbs: ['*']
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
    name: sonqo-controller
    namespace: mongodb-app
subjects:
    - kind: ServiceAccount
      name: sonqo
      apiGroup: ''
      namespace: mongodb-app
roleRef:
    kind: ClusterRole
    name: sonqo-role
    apiGroup: ''
```

Σημειώνεται ότι ο ServiceAccount που δημιουργείται παραπάνω, δηλώνεται στη χρήση του Πράκτορα στο αντίστοιχο .yaml αρχείο.

Εξυπηρετητής

Ο εξυπηρετητής που υλοποιήθηκε χρειάζεται μία μεταβλητή περιβάλλοντος, η οποία περιέχει το URL σύνδεσης στη βάση. Το αρχείο αυτό είναι:

```
module.exports = {
    url: process.env.MONGO_URL,
};
```

Το αρχικό αρχείο εκτέλεσης του server δομείται ως εξής:

```
const express = require('express');
const mongoose = require('mongoose');

const home = require('./routes/home');
const ride = require('./routes/ride');

const dbConfig = require('./config/database-config');

const server = express();
server.use(express.urlencoded({ extended: true }));
server.use(express.json());

mongoose
    .connect(dbConfig.url, {
        useNewUrlParser: true,
    })
    .then(() => {
        console.log('Successfully connected to database');
    })
    .catch((err) => {
        console.log('Could not connect to database');
    });

const baseUrl = '/api';
server.use(`${baseUrl}/`, home);
server.use(`${baseUrl}/ride`, ride);

const port = 3000;
server.listen(port, () => {
    console.log(`Server listening on port ${port}`);
});
```

Για τη δημιουργία του σχήματος δεδομένων χρησιμοποιήθηκε η βιβλιοθήκη της Javascript, *mongoose*. Το αντίστοιχο σχήμα έχει την ακόλουθη μορφή:

```
const mongoose = require('mongoose');

const rideSchema = new mongoose.Schema({
    ride_id: {
        type: String,
        required: true,
    },
    rideable_type: {
        type: String,
        required: true,
    },
    started_at: {
        type: String,
        required: true,
    },
    ended_at: {
        type: String,
        required: true,
    },
    start_station_name: {
        type: String,
        required: true,
    },
```

```
    start_station_id: {
        type: Number,
        required: true,
    },
    end_station_name: {
        type: String,
        required: true,
    },
    end_station_id: {
        type: Number,
        required: true,
    },
    start_lat: {
        type: Number,
        required: true,
    },
    start_lng: {
        type: Number,
        required: true,
    },
    end_lat: {
        type: Number,
        required: true,
    },
    end_lng: {
        type: Number,
        required: true,
    },
    member_casual: {
        type: String,
        required: true,
    },
});

const Ride = mongoose.model('Ride', rideSchema, 'Ride');

module.exports = Ride;
```

Το βασικό endpoint που χρησιμοποιήθηκε στη μελέτη της εφαρμογής χαρακτηρίζεται από τη διεύθυνση /api/ride/show και λαμβάνει παραμέτρους s_Date και e_Date, μεταβλητές οι οποίες αντιστοιχίζονται στις ημερομηνίες έναρξης και λήξης της ποδηλατικής διαδρομής:

```
const mongoose = require('mongoose');
const router = require('express').Router();

const Ride = require('../models/Ride');

const dbConfig = require('../config/database-config');

mongoose
    .connect(dbConfig.url, {
        useNewUrlParser: true,
        useUnifiedTopology: true,
        db: {
            readPreference: 'nearest',
        },
    })
```

```
    .catch((err) => console.error('Problem connecting to Mongo', err));

router.get('/show', async (req, res) => {
    docs = await Ride.find({
        started_at: { $gte: req.body.s_Date, $lt: req.body.e_Date },
        start_lat: { $gte: 40, $lt: 50 },
        start_lng: { $gte: -90, $lt: -85 },
        start_station_id: { $gte: 200, $lt: 250 },
        member_casual: 'member',
    }).read('nearest');
    if (!docs) {
        res.sendStatus(403);
    } else {
        res.send(docs);
    }
});

module.exports = router;
```

Το αρχείο συγκέντρωσης του αντίστοιχου κώδικα σε container φέρει την ακόλουθη μορφή:

```
FROM node:14-alpine

WORKDIR /app

COPY package*.json ./

RUN npm install --production

COPY . .

CMD node server.js
```

Τέλος, το αρχείο .yaml πλοήγησης του server στον Kubernetes είναι:

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: express-deployment
    labels:
        app: express
spec:
    replicas: 1
    selector:
        matchLabels:
            app: express
    template:
        metadata:
            labels:
                app: express
        spec:
            nodeSelector:
                status: master
            topologySpreadConstraints:
                - maxSkew: 1
                  topologyKey: kubernetes.io/hostname
                  whenUnsatisfiable: DoNotSchedule
                  labelSelector:
```

```
                    matchLabels:
                        app: express
            containers:
                - name: express
                  env:
                      - name: MONGO_URL
                        value: mongodb://user:root@mongodb-
service.mongodb-mvp:27017/mvp?authMechanism=SCRAM-SHA-
256&authSource=admin&readPreference=nearest
                  image: sonqo/express-app:latest
                  ports:
                      - containerPort: 3000
                  resources:
                      limits:
                          cpu: 500m
                      requests:
                          cpu: 250m
---
apiVersion: v1
kind: Service
metadata:
    name: express-ext
    labels:
        app: express
spec:
    selector:
        app: express
    type: LoadBalancer
    ports:
        - name: http
          protocol: TCP
          port: 80
          targetPort: 3000
```

Αξίζει να σημειωθεί το URL που περνάει το αρχείο στις μεταβλητές περιβάλλοντος, η οποία φέρει τη δήλωση *readPreference* με τιμή *nearest*.