Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας

Αυτοματοποίηση κύκλου ζωής Συστημάτων Κατανεμημένης Επεξεργασίας Δεδομένων σε Ροές Εργασίας Υπολογιστικών Νεφών

Διπλωματικη Εργασια

ΦΙΛΙΠΠΟΣ Π. ΜΑΛΑΝΔΡΑΚΗΣ

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής ΕΜΠ

Αθήνα, Φεβρουάριος 2022
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας

Αυτοματοποίηση κύκλου ζωής Συστημάτων Κατανεμημένης Επεξεργασίας Δεδομένων σε Ροές Εργασίας Υπολογιστικών Νεφών

Διπλωματικη Εργασια

ΦΙΛΙΠΠΟΣ Π. ΜΑΛΑΝΔΡΑΚΗΣ

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 9η Μαρτίου 2022.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Νεκτάριος Κοζύρης	Γεώργιος Γκούμας	Ιωάννης Κωνσταντίνου
Καθηγητής ΕΜΠ	Αν. Καθηγητής ΕΜΠ	Επ. Καθηγητής ΠΘ

Αθήνα, Φεβρουάριος 2022
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας

.....
Φίλιππος Μαλανδράκης,

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

Περίληψη

Ο πολλαπλασιασμός των εστιών παραγωγής δεδομένων τα τελευταία χρόνια, έχει ως αποτέλεσμα την εντατικοποίηση των προσπαθειών για εξαγωγή αξίας και πρόσδωση νοήματος σε αυτά. Οι προσπάθειες αυτές συνδυάζουν οντότητες τόσο από τον χώρο της αναλυτικής επεξεργασίας, όσο και της μηχανικής μάθησης. Παράλληλα, έντονη είναι και η τάση για μετακίνηση των παραγωγικών υποδομών σε υπολογιστικά νέφη που βασίζονται σε ελαφριές μορφές εικονοποίησης.

Ο συγκερασμός των δύο αυτών ρευμάτων οδήγησε στην δημιουργία του Kubeflow, το οποίο ζει πάνω στον ενορχηστρωτή containers Kubernetes και μεταξύ άλλων επιτρέπει την εκτέλεση ροών εργασίας. Οι ροές εργασίας αποτελούνται από έναν πεπερασμένο αριθμό βημάτων, που εκτελούνται σε κάποιο υπολογιστικό σύστημα (όπως την εκτέλεση ενός ερωτήματος SQL σε μια συστοιχια Apache Spark). Η παρατήρηση ότι το Kubeflow Pipelines εστιάζει στη χρήση απομακρυσμένων υπολογιστικών συστημάτων, οδήγησε στην ιδέα της ανάπτυξης ροών στα πλαίσια των οποίων θα εκκινούνται, διαχειρίζονται και παύονται επιτόπια υπολογιστικά συστήματα.

Στην πράξη, επιλέχθηκε η αυτοματοποίηση του κύκλου ζωής των κατανεμημένων συστοιχιών αναλυτικής επεξεργασίας Apache Spark. Προς την κατεύθυνση αυτή, υλοποιήθηκε μια σειρά από Kubeflow Pipelines components γενικού σκοπού, τα οποία δένονται μεταξύ τους σε μια ακολουθία βημάτων. Έπειτα, η ακολουθία αυτή μπορεί να εμπλουτιστεί εύκολα για την εκτέλεση οποιασδήποτε ροής μηχανικής μάθησης / αναλυτικής επεξεργασίας που απαιτεί τη χρήση του Apache Spark. Για την πρακτική επίδειξη του συγκεκριμένου, αναπτύχθηκε μια ενδεικτική διοχέτευση μηχανικής μάθησης που αξιοποιεί ένα Data Lake. Επιπλέον, μέρος των components που δημιουργήθηκαν τέθηκαν στη διάθεση όλων των χρηστών του Kubeflow Pipelines.

Λέξεις Κλειδιά

Υπολογιστικά νέφη, Μηχανική μάθηση, Πλατφόρμες ελαφριάς εικονοποίησης, Ροές εργασίας, Συστοιχίες αναλυτικής επεξεργασίας, Κατανεμημένα αποθηκευτικά συστήματα

Abstract

The multiplication of the available data sources during the past few years has intensified the efforts to derive value and meaning from them. These efforts are combining notions from the worlds of analytics and data science. At the same time, there is a strong trend towards migrating production infrastructure to containerized cloud environments.

The combination of the aforementioned movements led to the creation of Kubeflow, which lives on Kubernetes, a container orchestrator, and facilitates among other things the execution of Machine Learning workflows. These workflows consist of a finite number of steps, which are executed on some computing system (for example, the execution of an SQL query on an Apache Spark cluster). The observation that Kubeflow Pipelines focused on using remote computing systems led to the idea of developing workflows that would deploy, operate and uninstall computing systems on-premise.

As a result, it was chosen to automate the whole lifecycle of the distributed data analytics platform Apache Spark. In this direction, a series of general-purpose Kubeflow Pipelines components were implemented, which are eventually combined in a sequence of steps. Afterwards, this sequence can be easily enriched to execute any Machine Learning / analytics workflow that makes use of an Apache Spark cluster. To demonstrate this, a demo Machine Learning pipeline was designed, which also interacts with an external Data Lake. On top of that, some of the created components were contributed to Kubeflow Pipelines.

Keywords

Cloud, Machine Learning, Kubernetes, Kubeflow, Apache Spark, HDFS

Ευχαριστίες

Με αυτήν την διπλωματική εργασία ολοκληρώνεται ο κύκλος των προπτυχιακών σπουδών μου, το τελευταίο δηλαδή χαρτογραφημένο βήμα που είχα ορίσει από μικρή ηλικία για τη ζωή μου. Συνοδεύεται με ανυπομονησία, όνειρα και μία δόση δέους για το ¨αχανές¨ μέλλον που βρίσκεται εμπρός. Αποτελεί λοιπόν μια σημαντική στιγμή και ευκαιρία ενδοσκόπησης για εμένα.

Για την εκπόνηση της συγκεκριμένης εργασίας, θα ήθελα να ευχαριστήσω τον κ. Ιωάννη Κωνσταντίνου, που μου προσέφερε τόσο μια ενδιαφέρουσα ιδέα όσο και τις απαραίτητες κατευθύνσεις ώστε να προκύψει μία λειτουργική λύση. Για την γέννηση του γενικότερου ενδιαφέροντός μου για τα υπολογιστικά συστήματα, οφείλω ένα ευχαριστώ στους ανθρώπους που τρέχουν τη Ροή στη σχολή μας, ξεκινώντας κιόλας από τη πρώτη διάλεξη της Αρχιτεκτονικής Υπολογιστών του κ. Νεκτάριου Κοζύρη.

Αυτή η ενότητα δεν θα είχε ουσία αν δεν αναφερόμουν στα άτομα που με στήριξαν, προβλημάτισαν και ενέπνευσαν όλα αυτά τα χρόνια. Πρωταρχικώς η οικογένειά μου, οι γονείς μου και ο αδερφός μου, που δημιούργησαν ένα σταθερό και υγιές περιβάλλον για μένα και βρίσκονται πάντα στο πλάι μου. Οι άνθρωποι που έπαιξαν καθοριστικό ρόλο στη διαδρομή μου στη σχολή: Ο Παναγιώτης, που αποτέλεσε τον κύριο καταλύτη για την επιλογή της. Ο Διονύσης, που ήταν εκεί όταν νόμιζα ότι δεν θα τα κατάφερνα. Και φυσικά ο Βιτάλης, που μου έδωσε τα εργαλεία για να πετυχαίνω τους στόχους μου και να πιστεύω στις δυνατότητές μου, και έθεσε τις βάσεις για το επαγγελματικό μου ξεκίνημα. Ο Νίκος Κορμπάκης, που με καθοδήγησε από την πρώτη μου μέρα στο GRNET, μου έμαθε να δομώ τη σκέψη μου και άλλαξε την οπτική μου για την αποτυχία. Τέλος, οι παιδικοί μου φίλοι, που βιώσαμε παράλληλα όλο αυτό το ταξίδι, με τις όμορφες και τις λιγότερο όμορφες στιγμές του, και δημιουργήσαμε όνειρα μαζί.

Αθήνα, Φεβρουάριος 2022

Φίλιππος Μαλαυδράκης

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εισαγωγή

Στο κεφάλαιο αυτό περιγράφεται εν συντομία το αντικείμενο και ο στόχος της διπλωματικής εργασίας.

1.1 Ορισμός του προβλήματος

Τα τελευταία έτη, η αύξηση του διαθέσιμου όγκου δεδομένων είναι ραγδαία. Οι επιχειρήσεις έχουν στη διάθεση τους ένα πακτωλό μετρικών που προκύπτουν από τη χρήση των προϊόντων τους και αφορούν τη συμπεριφορά και τις προτιμήσεις των χρηστών τους. Το γεγονός αυτό έχει οδηγήσει στην εντατικοποίηση της προσπάθειας εξαγωγής κέρδους και νοήματος από τα δεδομένα αυτά, είτε μέσω μηχανικής μάθησης ή στατιστικής ανάλυσης. Ένας επιπλέον παράγοντας που περιπλέκει την κατάσταση αποτελεί η μετακίνηση των υποδομών σε υπολογιστικά νέφη, έχοντας ως αποτέλεσμα τον διαχωρισμό των αποθηκευτικών και των υπολογιστικών συστημάτων σε ξεχωριστές πλατφόρμες. Συνεπώς, η πολυπλοκότητα των συστημάτων που επεξεργάζονται επιτυχημένα σύνολα δεδομένων και προσδίδουν αξία σε αυτά τείνει να αυξάνει.

Ένας τυπικός κύκλος επεξεργασίας αποτελείται από διάφορα διακριτά στάδια, τα οποία πρέπει να συντονιστούν από μια ενιαία πλατφόρμα. Συνοπτικά, δεδομένα αντλούνται από ένα κεντρικό αποθηκευτικό σύστημα (Data Lake), υπόκεινται σε μια σειρά από εργασίες και συντελούν στην παραγωγή κάποιου τελικού αποτελέσματος. Το αποτέλεσμα αυτό δύναται να είναι μια αναφορά που περιγράφει τις τάσεις των χρηστών ή ένα μοντέλο που θα προβλέψει μελλοντικές συμπεριφορές.

Μία ακόμη τάση στα υπολογιστικά νέφη αποτελεί η στροφή σε πιο ελαφριές μορφές εικονοποίησης στα παραγωγικά περιβάλλοντα, στοχεύοντας σε πιο ευέλικτα συστήματα με μειωμένο χρόνο εγκατάστασης και ανοχή στα λάθη. Οι εικονικές μηχανές δίνουν τη θέση τους στα containers, χωρίς βέβαια αυτό να σημαίνει ότι δεν παραμένουν προτιμότερες για πλειάδα εφαρμογών. Η άνοδος των containers συμπίπτει με την καθιέρωση πλατφόρμων διαχείρισής τους, με μία εκ των επικρατέστερων λύσεων να είναι το προϊόν ανοιχτού λογισμικού Kubernetes [1] (εμπνευσμένο από το Borg [2] της Google).

Ο συγκερασμός των δύο αυτών ρευμάτων οδηγεί στη ανάπτυξη συστημάτων τα οποία εφαρμόζουν τόσο μοτίβα μηχανικής μάθησης όσο και διαδικασίες αναλυτικής επεξεργασίας πάνω σε ενορχηστρωτές containers. Ένα τετοιο σύστημα είναι το Kubeflow [3], το οποίο προσφέρει τη δυνατότητα ορισμού και εκτέλεσης διοχετεύσεων επεξεργασίας δεδομένων πάνω

στο Kubernetes, μέσω της λειτουργικότητας Kubeflow Pipelines [4]. Οι ροές αυτές διαθέτουν τη μορφή κατευθυνόμενων ακυκλικών γράφων (Directed Acyclic Graphs), ενώ η επεξεργασία των δεδομένων γίνεται σε κατανεμημένα συστήματα όπως το Apache Spark [5].

1.2 Κίνητρο

Παρατηρώντας τις έτοιμες λύσεις που προσφέρονται από το Kubeflow και μπορούν να ενταχθούν ως τμήματα ευρύτερων διοχετεύσεων, διακρίνεται μια προτίμηση σε υποδομές που τρέχουν σε απομακρυσμένα περιβάλλοντα (παραδείγματος χάρην Google Cloud [6] και Azure [7]). Γεννάται εύλογα η απορία αν θα μπορούσε να αξιοποιηθεί αντί αυτών η ίδια η υποδομή στην οποία εκτελείται το Kubeflow (δηλαδή το Kubernetes). Πιο συγκεκριμένα, αντί να θεωρείται δεδομένο πως εξωτερικά της υποδομής υπάρχει ένα σύστημα που είναι έτοιμο να δεχτεί εντολές από το Kubeflow, να εκτελείται το ίδιο σύστημα τοπικά στο Kubernetes.

Με αφορμή το παραπάνω, σκοπός της συγκεκριμένης διπλωματικής εργασίας αποτελεί η επιτόπια διαχείριση συστοιχιών αναλυτικής επεξεργασίας Apache Spark μέσω των Kubeflow Pipelines. Η διαχείριση του κύκλου ζωής του Apache Spark θα πρέπει να είναι πλήρως αυτοματοποιημένη, αναπτύσσοντας τα απαραίτητα Kubeflow components προς αυτή την κατεύθυνση. Συμπληρωματικά, για τον έλεγχο του προκύπτοντος συστήματος θα εκτελεστεί μια διοχέτευση μηχανικής μάθησης στο Kubeflow.

1.3 Δομή της εργασίας

Το υπόλοιπο κείμενο ακολουθεί την εξής δομή:

- Στο κεφάλαιο 2 παρατίθεται το θεωρητικό υπόβαθρο που είναι απαραίτητο για την κατανόηση της εργασίας
- Στο κεφάλαιο 3 ορίζεται η αρχιτεκτονική του συστήματος και η ροή μηχανικής μάθησης που θα εκτελεστεί
- Στο κεφάλαιο 4 υλοποιείται το σύστημα και η ροή μηχανικής μάθησης που το αξιοποιεί
- Στο κεφάλαιο 5 καταγράφονται τελικά συμπεράσματα και τρόποι που θα μπορούσε να επεκταθεί η συγκεκριμένη δουλειά

Κεφάλαιο 2

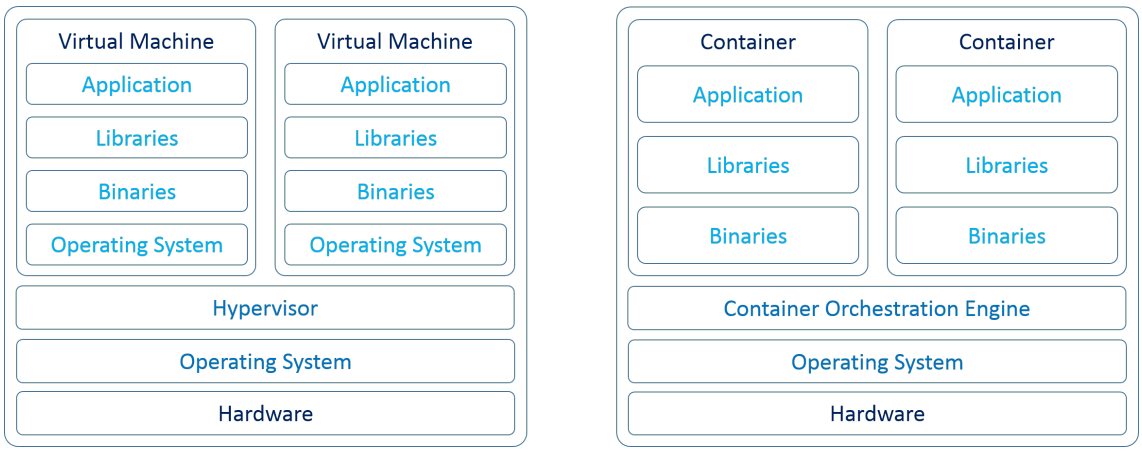
Θεωρητικό υπόβαθρο

Στο παρόν κεφάλαιο καταγράφεται το σύνολο των εννοιών και τεχνολογιών που θεωρούνται απαραίτητες για την κατανόηση της διπλωματικής εργασίας.

2.1 Containers

2.1.1 Επισκόπηση

Τα containers αποτελούν τον κυρίαρχο τύπο αντικειμένων εικονοποίησης σε επίπεδο λειτουργικού συστήματος. Η συγκεκριμένη μορφή εικονοποίησης αποτελεί ένα χαρακτηριστικό του λειτουργικού συστήματος μέσω του οποίου υπηρεσίες του πυρήνα επιτρέπουν τη συνύπαρξη πολλαπλών απομονωμένων αντικειμένων χώρων χρήστη. Η δημοφιλία τους οφείλεται μεταξύ άλλων στην ευκολία χρήσης τους, στην ταχύτητά τους και στο μειωμένο αποτύπωμά τους σε πόρους.



Σχήμα 2.1: *Διαφοροποίηση εικονικών μηχανών από containers*

Οι διεργασίες που εκτελούνται εσωτερικά σε containers δεν έχουν αντίληψη πέρι αυτού, θεωρώντας ότι εκτελούνται σε πραγματικά μηχανήματα. Τα containers μοιράζονται το λειτουργικό σύστημα του μηχανήματος που τα 'φιλοξενεί', και χρησιμοποιούν τη διεπαφή κλήσεων συστήματος αυτού. Αυτό έρχεται σε πλήρη αντίθεση με τις τεχνικές πλήρους εικονοποίησης, οι οποίες εκτελούν ένα διακριτό λειτουργικό σύστημα και εξομοιώνουν όλες τις συσκευές υλικού του (παραδείγματως χάρην τις κάρτες δικτύου, αποθηκευτικό χώρο κλπ). Εδώ οφείλεται η υψηλή απόδοση και η ελαφρότητα των containers, εφόσον αλληλεπιδρούν

απευθείας με το υλικό του υπαρχόντος συστήματος και έχουν τον ελάχιστο δυνατό αριθμό εξαρτήσεων (σε βιβλιοθήκες και αρχεία). Παρόλα αυτά, η απομόνωση του εικονικού συστήματος που επιτυγχάνεται μέσω των τεχνικών πλήρους εικονοποίησης εγγυάται υψηλότερη ασφάλεια, αναλόγως και τις περιστάσεις.

Τα παραπάνω καταδεικνύουν ότι τα containers ταιριάζουν απόλυτα στη σύγχρονη πραγματικότητα των δημοσίων υπολογιστικών νεφών και των κέντρων δεδομένων, στα οποία εφαρμογές αναπτύσσονται, εγκαθιστώνται και ανακυκλώνονται με ιλιγγιώδεις ρυθμούς. Μία ακόμη σημαντική συνεισφορά τους είναι η διευκόλυνση της καθημερινής ζωής του προγραμματιστή, που δύναται πλέον να πειραματιστεί με πολύπλοκα συστήματα μέσα σε δευτερόλεπτα και χωρίς το επιπλέον έξοδο που θα επέφερε ένας απομακρυσμένος εξυπηρετητής.

2.1.2 Συστατικά στοιχεία

Πίσω από ένα container βρίσκονται κάποιοι θεμελιώδεις μηχανισμοί των Linux-based πυρήνων, οι οποίοι θα αναλυθούν στη συνέχεια.

Namespaces

Η συγκεκριμένη δομή επιτρέπει (σε υψηλό επίπεδο) την απομόνωση διαφόρων πόρων του συστήματος μεταξύ διαφορετικών διεργασιών. Κάθε διεργασία εκτελείται εντός κάποιου namespace, και το namespace έχει πρόσβαση σε ένα υποσύνολο των πόρων του συστήματος. Στην προκειμένη περίπτωση, οι πόροι αυτοί μπορεί να είναι χρήστες, διεπαφές δικτύου, αναγνωριστικά διεργασιών, συστήματα αρχείων και πολλά άλλα.

cgroups

Τα cgroups προσφέρουν και αυτά απομόνωση υπολογιστικών πόρων μεταξύ των διαφόρων διεργασιών. Πιο συγκεκριμένα, καθιστούν εφικτό να τίθεται ένα άνω όριο χρήσης της CPU, τη μνήμης, του εύρους ζώνης, των εισόδων/εξόδων δίσκου (και άλλων) για μια διεργασία.

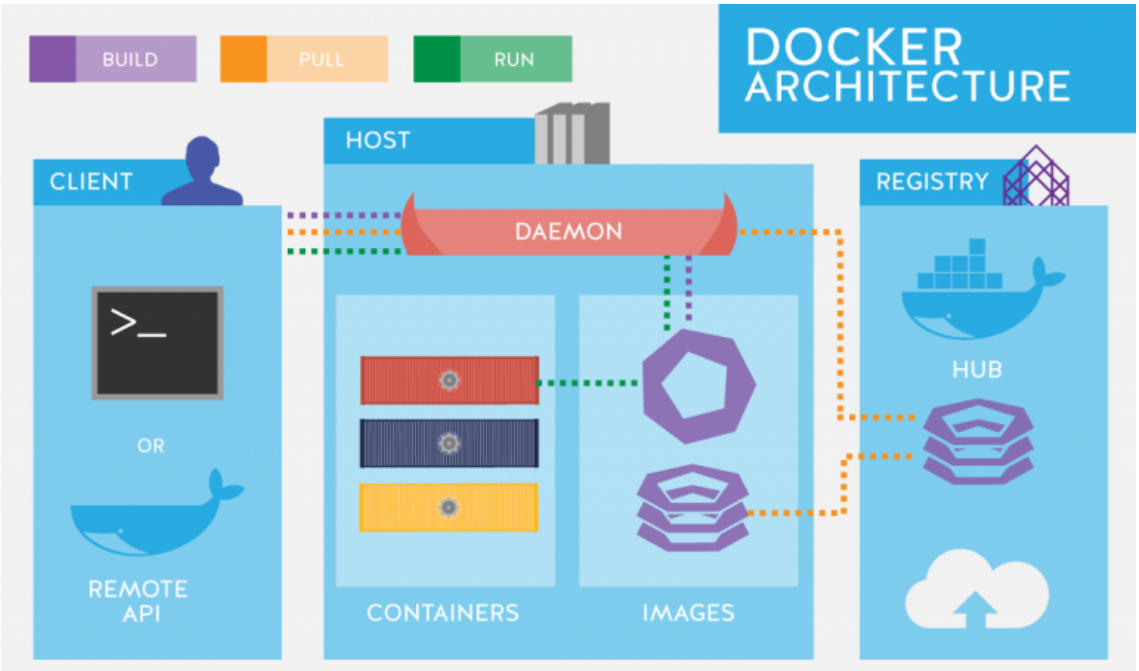
Union filesystems

Έχοντας περιγράψει τις παραπάνω λειτουργικότητες, είναι φανερό ότι το μόνο που λείπει από μια διεργασία για να γίνει ένα ανεξάρτητο container είναι η δική της ιεραρχία συστήματος αρχείων. Αυτό έρχονται να προσφέρουν τα union (unification) filesystems, τα οποία είναι αποτέλεσμα της συνένωσης πολλαπλών συστημάτων αρχείων σε ένα. Στο σύστημα αυτό αποθηκεύονται όλες οι βιβλιοθήκες και τα αρχεία που είναι απαραίτητα για τη λειτουργικότητα του container.

Όσον αφορά τώρα στα Windows, η Microsoft προχώρησε και αυτή στην υλοποίηση αντίστοιχων χαρακτηριστικών, σε δεύτερο χρόνο.

2.1.3 Υλοποιήσεις

Υπάρχουν διάφορετικοί τρόποι με τους οποίους μπορούν τόσο να υλοποιηθούν, όσο και να διαχειριστούν containers. Για την εκπόνηση της συγκεκριμένης διπλωματικής χρησι-



Σχήμα 2.2: *Τα βασικά συστατικά του οικοσυστήματος Docker*

μοποιήθηκε η δημοφιλέστερη τη στιγμή αυτή διαχειρίστρια containers, Docker [8]. Η Docker αποτελεί μια εταιρεία που προσφέρει λύσεις για δημιουργία και διαμοιρασμό containers, υπό τη μορφή στιγμιοτύπων. Όσον αφορά την ίδια την υλοποίηση των containers, η προεπιλεγμένη λύση της Docker είναι τα περιβάλλοντα εκτέλεσης runc [9] (τα οποία συνδυάζουν όλα τα συστατικά που αναλύθηκαν παραπάνω, ώστε να συνθέσουν ένα container).

Δομή των στιγμιοτύπων

Τα στιγμιότυπα περιγράφουν την ακριβή επιθυμητή κατάσταση στην οποία θα πρέπει να βρίσκεται ένα container μόλις εκκινηθεί. Η κατάσταση αυτή συμπεριλαμβάνει τα εξής:

- Την πλήρη εικόνα του συστήματος αρχείων εσωτερικά του container (rootfs). Το σύστημα αυτό, έχει μία ρίζα (/) αντίστοιχη κάθε λειτουργικού συστήματος, και διάφορα αρχεία και φακέλους κάτω από αυτήν.
- Ποια διεργασία θα πρέπει να τρέχει μέσα στο container μόλις αυτό εκκινηθεί, καθώς και το φάκελο στον οποίο θα τρέχει, διάφορες μεταβλητές περιβάλλοντος και άλλα. Η συγκεκριμένη παραμετροποίηση καταγράφεται στο εσωτερικό ενός JSON [10] αρχείου.

Τα χαρακτηριστικά που θα πρέπει να έχει ένα στιγμιότυπο περιγράφονται μέσω ενός Dockerfile [11]. Στη συνέχεια, το Dockerfile συντίθεται (build) ώστε να προκύψει το στιγμιότυπο. Το στιγμιότυπο πρακτικά είναι ένα συμπιεσμένο αρχείο (TAR - με περιεχόμενα το rootfs και ενα JSON αρχείο), το οποίο μπορεί να εκκινηθεί απεριόριστες φορές και να δημιουργήσει containers. Το σύστημα αρχείων των containers που προκύπτουν αποτελείται από ένα σύνολο στρώσεων μη μεταβλητών συστημάτων αρχείων, με μία τελική επίστρωση ενός μεταβλητού συστήματος (εδώ υπεισέρχεται και η έννοια των union filesystems). Ο χρήστης αλληλεπιδρά αποκλειστικά με την τελική επίστρωση.

Διαμοιρασμός των στιγμιοτύπων

Μετά τη δημιουργία ενος στιγμιοτύπου, ο χρήστης μπορεί πολύ εύκολα να το μοιραστεί με την κοινότητα, δημοσιεύοντάς το σε κάποιο αποθετήριο στιγμιοτύπων. Η Docker διαθέτει το δικό της αποθετήριο εικόνων που ονομάζεται DockerHub [12], και παρομοίως υπάρχουν ποικίλες εναλλακτικές (τόσο δημόσιες όσο και εταιρικές λύσεις). Τα αποθετήρια αυτά επιτρέπουν την διαχείριση διαφόρων εκδόσεων του ίδιου container (version control), ακολουθώντας την λογική των αποθετηρίων που υποστηρίζουν το πρωτόκολλο git [13] (GitHub, GitLab, ...).

2.2 Kubernetes

2.2.1 Επισκόπηση

Τα containers αποτελούν ένα θαυμάσιο τρόπο για να υλοποιηθούν εφαρμογές υψηλής κλιμακωσιμότητας, οι οποίες θα συντίθενται από επιμέρους containers. Οι εφαρμογές αυτές θα πρέπει να είναι ανθεκτικές σε σφάλματα ενός ή περισσοτέρων containers, εξασφαλίζοντας με κάποιο τρόπο ότι θα επαναδημιουργούνται αενάως. Γίνεται εύκολα κατανοητό ότι αν ήταν μία φορά δύσκολο για έναν ανθρώπινο χειριστή να κατορθώσει το παραπάνω μέσω εικονικών μηχανών, εδώ η δυσκολία πολλαπλασιάζεται. Η συγκεκριμένη κατάσταση οδήγησε στην δημιουργία διαφόρων λυσεων οι οποίες αναλαμβάνουν να διαχειρίζονται τον πλήρη κύκλο ζωής των containers.

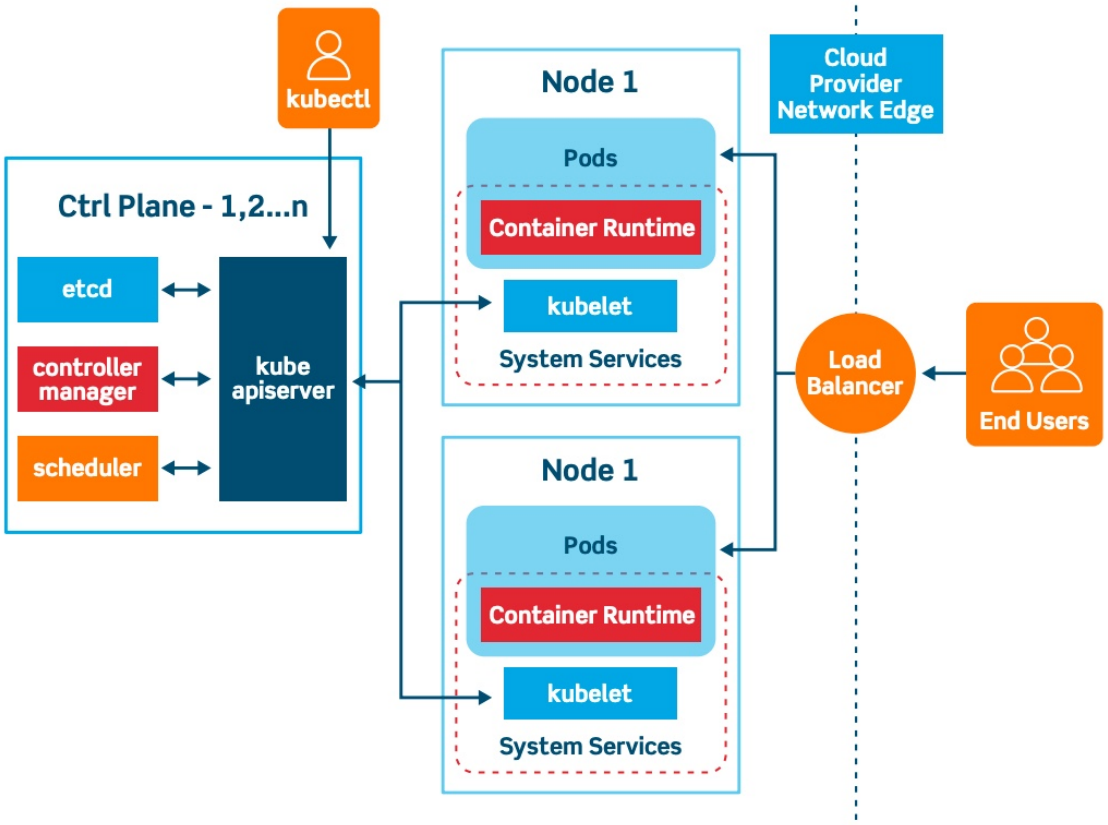
Ο κυρίαρχος διαχειριστής containerized όγκων εργασίας (workloads) είναι το Kubernetes. Πρόκειται για ένα ισχυρό προϊόν, που επιτυγχάνει τη μαζική εκτέλεση χιλιάδων containers, εγγυόμενο μεταξύ άλλων την απομόνωσή τους, την ενδοεπικοινωνία τους, την επιτήρηση της κατάστασής τους, την επαναδημιουργία τους σε περίπτωση αποτυχίας και πολλά ακόμη.

Όπως τα Dockerfiles καθορίζουν τα περιεχόμενα ενός στιγμιοτύπου, έτσι και το Kubernetes δέχεται ως εισόδους αρχεία YAML [14], τα οποία περιγράφουν επιθυμητούς όγκους εργασίας που αποτελούνται από ένα ή παραπάνω containers. Η περιγραφική αυτή αποτύπωση των πολλαπλών στοιχείων που συνθέτουν μια εφαρμογή ονομάζεται Υποδομή ως Κώδικας (Infrastructure as Code).

2.2.2 Αρχιτεκτονική

Μια εγκατάσταση Kubernetes αποτελείται από ένα σύμπλεγμα κόμβων (cluster), οι οποίοι διαχωρίζονται σε δύο διακριτά επίπεδα: το επίπεδο ελέγχου (control plane) και το επίπεδο δεδομένων (data plane). Όλη η 'κατάσταση' του Kubernetes (οι όγκοι εργασιών που πρέπει να τρέχουν ανά πάσα στιγμή) αποθηκεύεται σε μια κατανεμημένη βάση κλειδιών-τιμών (key-value storage) υψηλής διαθεσιμότητας, που ονομάζεται etcd [15]. Το σύνολο των παραγωγικών όγκων εργασίας (που δεν σχετίζονται με ζωτικές λειτουργίες του cluster) σηκώνονται αποκλειστικά στο επίπεδο δεδομένων του.

Τα κομμάτια που απαρτίζουν το επίπεδο ελέγχου (και άρα εκτελούνται στους κόμβους που το συνθέτουν) είναι τα παρακάτω:



Σχήμα 2.3: *Συνοπτική αρχιτεκτονική του Kubernetes*

- kube-apiserver: Πρακτικά, ένας εξυπηρετητής που εκθέτει μια διεπαφή προγραμματισμού εφαρμογών (API). Τόσο ο προγραμματιστής (μέσω του λογισμικού γραμμής εντολών kubectl) όσο και το ίδιο το cluster χρησιμοποιούν τη συγκεκριμένη διεπαφή για να διαβάσουν / μεταβάλλουν την κατάστασή του.

- kube-scheduler: Ελέγχει ανά τακτικά διαστήματα το etcd ('ρωτώντας' τον kube-apiserver) για όγκους εργασίας οι οποίοι δεν έχουν ανατεθεί σε κάποιο κόμβο του επιπέδου δεδομένων. Όταν βρει τέτοιους, τους αναθέτει βάσει αλγορίθμου σε κάποιο κόμβο και ενημερώνει την κατάσταση στο etcd.

- kube-controller-manager: Τρέχει ένα σύνολο από ελεγκτές (controllers) οι οποίοι επικυρώνουν διαρκώς ότι η κατάσταση που περιγράφεται στο etcd αντιστοιχεί στην πραγματική κατάσταση του cluster. Όταν παραδείγματος χάρην ο kube-scheduler αναθέσει έναν όγκο εργασίας σε κάποιο κόμβο και το καταγράψει στο etcd, οι controllers θα διαβάσουν τη νέα κατάσταση, θα δουν ότι ο συγκεκριμένος όγκος δεν τρέχει στον καθορισμένο κόμβο, και θα επικοινωνήσουν με τον κόμβο ώστε να εκκινηθούν οι απαραίτητες διεργασίες.

- etcd: Όπως προαναφέρθηκε, το etcd αποτελεί την μοναδική πηγή αλήθειας (source of truth) για το cluster. Οι κόμβοι που το απαρτίζουν μπορούν είτε να ταυτίζονται με το επίπεδο ελέγχουν του Kubernetes, είτε να είναι διακριτοί. Σε απαιτητικές παραγωγικές

συνθήκες, συνηθίζεται το δεύτερο.

Τα μέρη που απαρτίζουν το επίπεδο δεδομένων (και τρέχουν σε κάθε κόμβο που ανήκει σε αυτό) παραθέτονται στη συνέχεια.

- kubelet: Αλληλεπιδρά με τους controllers του επιπέδου ελέγχου και διαχειρίζεται όλα τα containers που σχηματίζουν έναν όγκο εργασίας, ελέγχοντας για τυχόν απώλειες και αναπληρώνοντάς τες.
- kube-proxy: Αποτελέι έναν δικτυακό διακομιστή μεσολάβησης (proxy server), θεμέλιο λίθο στην έννοια της υπηρεσίας (Service) που προσφέρει το Kubernetes για να εκθέσει τους εσωτερικούς όγκους εργασιών του εντός και εκτός του cluster. Διατηρεί τους δικτυακούς κανόνες του κόμβου στον οποίο εκτελείται (εντός των iptables / nftables).
- Container runtime: Μπορεί το kubelet να διασφαλίζει ότι ένας συγκεκριμένος αριθμός από containers εκτελούνται πάντοτε στον κόμβο που ζει, αλλα ο πραγματικός δημιουργός των containers είναι ένα περιβάλλον εκτέλεσης (πχ containerd [16], cri-o [17], docker - η υποστήριξη για το τελευταίο καταργήθηκε σε πρόσφατη έκδοση του Kubernetes). Βέβαια, και το περιβάλλον εκτέλεσης με τη σειρά του επικοινωνεί με ένα ακόμη περιβάλλον εκτέλεσης containers, το runc (αναφέρθηκε στην ενότητα του Docker) [18].

2.2.3 Όγκοι εργασίας

Μέχρι στιγμής, το σύνολο των παραγωγικών containers που εκτελούνται στο επίπεδο δεδομένων περιγράφεται με τον σχετικά ασαφή όρο των όγκων εργασίας. Στην πραγματικότητα, ένας όγκος εργασίας αποτελείται από ένα η περισσότερα αντικείμενα (resources). Ένα αντικείμενο μπορεί να περιγράφει μια συστάδα από containers, τη δικτυακή παραμετροποίησή της, τον αποθηκευτικό χώρο της, την πολιτική ασφαλείας της, διάφορες μυστικές μεταβλητές της και άλλα.

Όπως προαναφέρθηκε, τα αντικείμενα καταγράφονται σε αρχεία YAML, ακολουθώντας ορισμένες διεπαφές προγραμματισμού εφαρμογών, τα οποία στη συνέχεια υποβάλλονται στον kube-apiserver. Κάποια από τα κυριότερα πεδία ενός αντικειμένου καταγράφονται στη συνέχεια.

- kind: Το είδος του αντικειμένου που θα δημιουργηθεί. Σημαντικά είδη αποτελούν τα Pods (συστάδες containers), Deployments (συστάδες Pods), Services (ομαδοποιούν 1 ή παραπάνω Pods κάτω από μία διεύθυνση δικτύου), Ingresses (λειτουργούν ως αντεστραμμένοι διακομιστές μεσολάβησης που εκθέτουν τα διάφορα Services του cluster στον έξω κόσμο) και τα Secrets (μυστικές μεταβλητές που χρησιμοποιούνται από τα Pods).
- apiVersion: Κάθε αντικείμενο περνάει από διάφορα στάδια (alpha, beta) έως ότου να φτάσει στη στατική του διεπαφή προγραμματισμού εφαρμογών. Το συγκεκριμένο πεδίο πολύ απλά διευκρινίζει ποια μορφή του αντικειμένου θα χρησιμοποιηθεί (άρα και ποια πεδία του θα είναι διαθέσιμα).

- metadata: Δεδομένα που χαρακτηρίζουν μοναδικά ένα αντικείμενο (παραδείγματος χάρην ημερομηνία δημιουργίας, αριθμητικό αναγνωριστικό και άλλα).
- spec: Κάτω από το συγκεκριμένο πεδίο εμφωλιάζονται όλες οι εξειδικευμένες επιλογές που αφορούν ένα αντικείμενο.
- status: Η φάση ζωής στην οποία βρίσκεται το αντικείμενο σε μια ορισμένη στιγμή. Το πεδίο αυτό δεν συμπληρώνεται κατά την αρχική υποβολή του αντικειμένου στον kube-apiserver. Αντιθέτως, προστίθεται από το επίπεδο ελέγχου αφότου το αντικείμενο καταχωρηθεί στο etcd. Κάποιες πιθανές τιμές του είναι Creating (το αντικείμενο βρίσκεται υπό δημιουργία), Running (εκτελείται), Error (έχει εμφανίσει κάποιο σφάλμα) και άλλες.

Τα αντικείμενα διαχωρίζονται μεταξύ τους σε έναν ή περισσότερους χώρους ονομάτων (namespaces), οι οποίοι είναι απομονωμένοι μεταξύ τους και εξασφαλίζουν τη διάκριση μεταξύ των διαφόρων όγκων εργασίας που εκτελούνται στο Kubernetes.

Αποθηκευτικός χώρος

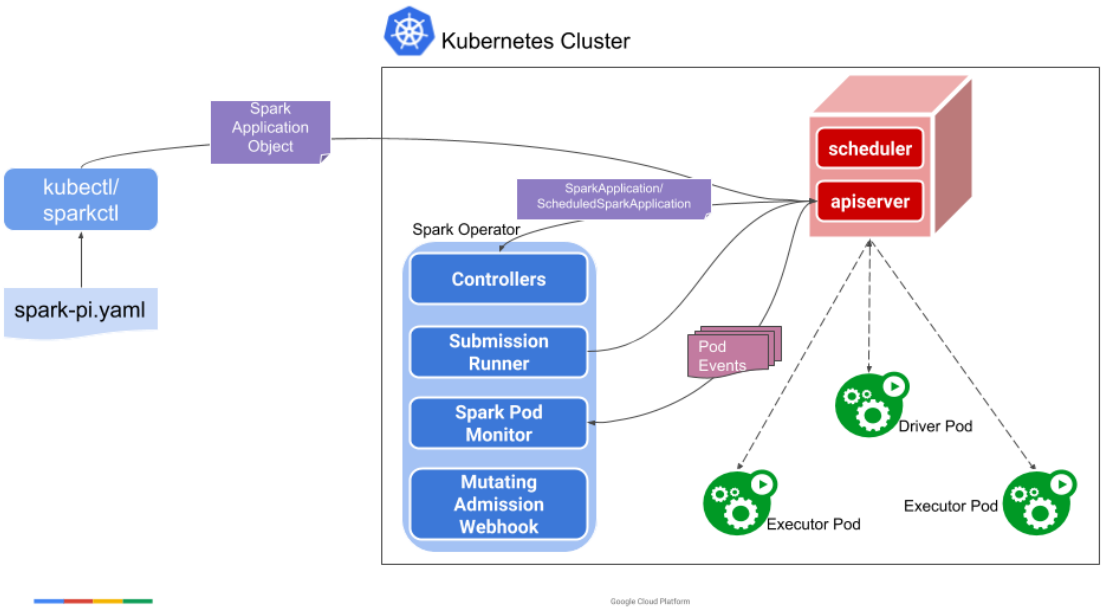
Κρίνεται χρήσιμο να γίνει μια σύντομη μνεία στους λογικούς δίσκους (Volumes), οι οποίοι είναι προσβάσιμοι από τα containers των Kubernetes Pods για ανάγνωση/εγγραφή δεδομένων. Η τοποθεσία του συστήματος αρχείων (filesystem path) υπό την οποία είναι προσβάσιμοι (mounted) οι λογικοί δίσκοι καθορίζεται από το πεδίο *volumeMounts* των αντικειμένων Kubernetes.

Ένας λογικός δίσκος δύναται να λάβει πολλές μορφές, που σχετίζονται με τον τόπο και τη μονιμότητα αποθήκευσης των δεδομένων του. Στην περίπτωση που ο χρήστης επιθυμεί τα δεδομένα ενός λογικού δίσκου να διατηρούνται ανεξάρτητα του κύκλου ζωής των Pods που το χρησιμοποιούν, τότε μπορεί να χρησιμοποιήσει τα persistent Volumes. Το Kubernetes προσφέρει ακόμα την δυνατότητα αυτοματοποιημένης 'κοπής' Persistent Volumes από κάποιο εξωτερικό αποθηκευτικό σύστημα (παραδείγματος χάρην σύστημα με προσβασιμότητα NFS / RBD - NetApp [19] και Ceph [20] αντίστοιχα), μέσω της λειτουργικότητας των κλάσεων αποθηκευτικού χώρου (Storage Class).

2.2.4 Επεκτασιμότητα του Kubernetes API

Όπως προαναφέρθηκε, το Kubernetes διαθέτει διάφορα είδη αντικειμένων που μπορούν να χρησιμοποιηθούν από έναν χρήστη. Ωστόσο, σε περιπτώσεις πολύπλοκων όγκων εργασίας, η χρήση των αντικειμένων αυτών καθίσταται υπερβολικά περίπλοκη και χρονοβόρα. Το Kubernetes προσφέρει την δυνατότητα σχεδιασμού εξειδικευμένων αντικειμένων (custom resources), η δομή των οποίων ορίζεται από μια νέα διεπαφή προγραμματισμού εφαρμογών (custom resource definition) που παρασκηνιακά συνδυάζει τα προϋπάρχοντα αντικείμενα. Για τον έλεγχο της κατάστασης των εξειδικευμένων αυτών αντικειμένων, απαιτείται και ένας αντίστοιχα ελεγκτής εξειδικευμένων αντικειμένων (custom controller) που θα κατανοεί το API τους.

Συνδυάζοντας τα εξειδικευμένα αντικείμενα με τους ελεγκτές τους προκύπτει η έννοια του χειριστή (operator). Για να υποβληθεί επιτυχημένα στον kube-apiserver ένα εξειδικευμένο



Σχήμα 2.4: *Η αρχιτεκτονική του spark-on-k8s operator*

αντικείμενο ενός χειριστή, προϋπόθεση αποτελεί η εγκατάστασή του χειριστή. Η εγκατάσταση αυτή περιλαμβάνει την προσθήκη του όρισμου του εξειδικευμένου αντικειμένου στο cluster, καθώς και του αντίστοιχου ελεγκτή. Παραδείγματα γνωστών χειριστών με υλοποιήσεις ανοιχτού λογισμικού αποτελούν το Elastic Cloud on Kubernetes [21] (σηκώνει την στοίβα τεχνολογιών Elasticsearch, (File)Beat και Kibana) και ο **spark-on-k8s-operator [22] (εκκινεί Apache Spark clusters μέσω του εξειδικευμένου αντικειμένου SparkApplication - η αρχή λειτουργίας του φαίνεται στο** Figure 2.4).

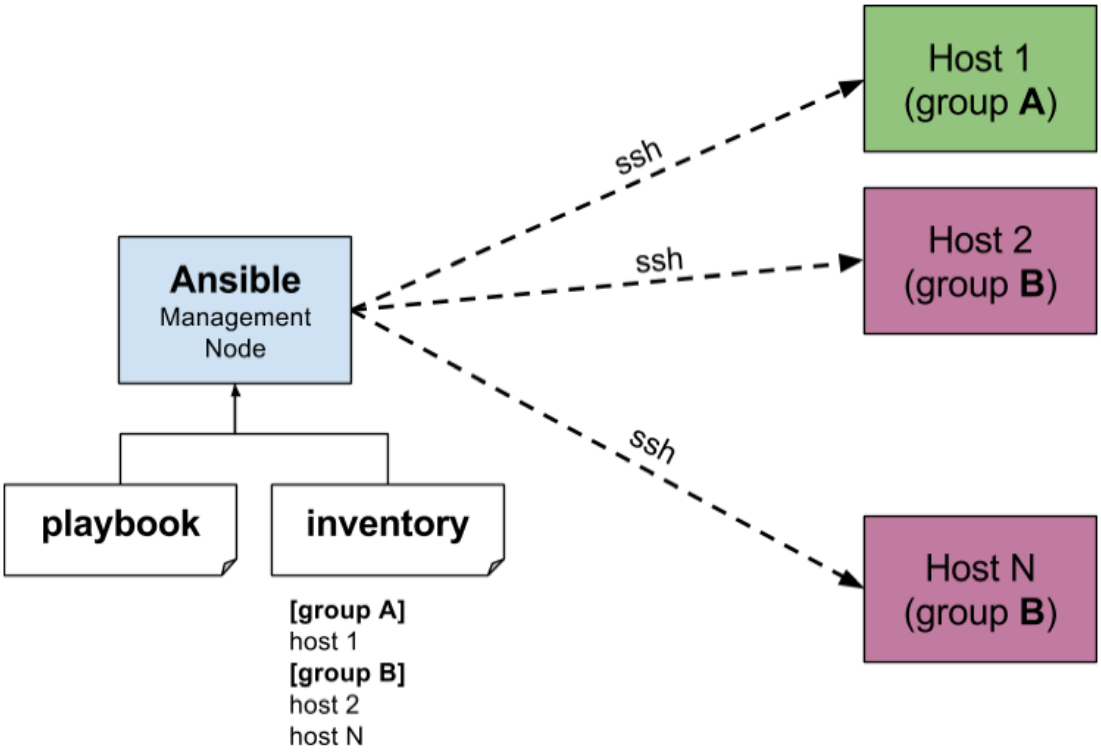
2.2.5 Προχωρημένα θέματα

Σε αυτή την ενότητα αναλύονται 2 τεχνικά εργαλεία που ενισχύουν τα επίπεδα παραγωγικότητας και αυτοματοποίησης σε εγκαταστάσεις Kubernetes με υψηλές απαιτήσεις απόδοσης.

Kubespray

Το Kubespray [23] αποτελεί μια σειρά από Ansible [24] playbooks, τα οποία χρησιμοποιούνται για την εγκατάσταση, αναβάθμιση και συντήρηση Kubernetes clusters σε μια σειρά υπολογιστικών κόμβων. Η Ansible με τη σειρά της είναι μια προγραμματιστική γλώσσα που εντάσσεται στο πεδίο της Υποδομής ως Κώδικα, και εκτελεί επαναλαμβανόμενες εργασίες μαζικά πάνω σε προκαθορισμένους κόμβους. Οι εργασίες που θα εκτελεστούν καθορίζονται μέσα σε ένα ή περισσότερα βιβλία στρατηγικής (playbooks), ενώ οι κόμβοι-στόχοι καταγράφονται σε έναν κατάλογο (inventory). Τέλος, τα playbooks υποστηρίζουν τη δημιουργία μεταβλητών οι οποίες θα λάβουν τιμές κατά την εκτέλεση, μέσω αρχείων group variables.

Με το Kubespray, γίνεται δυνατή η δημιουργία ενός Kubernetes cluster από το μηδέν μέσα στο διάστημα λίγων δεκάδων λεπτών. Το μόνο που απαιτείται από τον χρήστη είναι η



Σχήμα 2.5: *Συνοπτική αναπαράσταση της λειτουργίας της Ansible*

κατοχή μερικών εικονικών μηχανών (ή εξυπηρετητών), τις οποίες θα κατανείμει στα επίπεδα ελέγχου και δεδομένων, ενημερώνοντας κατάλληλα το inventory του Kubespray. Έτσι, το Kubespray μπορεί πολύ εύκολα να ενσωματωθεί με τη σειρά του σε αυτοματοποιημένες ροές εργασιών.

Helm Charts

Ένα Helm Chart [25] αποτελεί μια συλλογή από ένα ή περισσότερα αντικείμενα. Η υποβολή του Chart στον kube-apiserver ισοδυναμεί με την μαζική υποβολή όλων των αντικειμένων που το απαρτίζουν σε αυτό. Η εγκατάσταση / αναβάθμιση / διαγραφή των Helm Charts επιτυγχάνεται με τη βοήθεια του λογισμικου γραμμής εντολών (cli client) helm. Κρίνεται σημαντικό να αναφερθεί ότι το Helm δίνει τη δυνατότητα να οριστούν πεδία των αντικειμένων που τα απαρτίζουν ως μεταβλητές, οι οποίες μπορούν να λάβουν διάφορες τιμές κατά την εγκατάσταση των Charts. Το αρχείο στο οποίο κάθε φορά ορίζονται οι τιμές των μεταβλητών ονομάζεται Helm Values.

Όπως οι περιέκτες Docker, έτσι και τα Helm Charts διαμοιράζονται μεταξύ των χρηστών και υπόκεινται σε version control μέσω ειδικών αποθετηρίων για Helm Charts. Εκεί αποθηκεύονται σε συμπιεσμένη μορφή, με κάποια από τα πιο γνωστά αποθετήρια είναι το ChartMuseum και το Harbor (τα GitHub, GitLab έχουν υλοποιήσει και αυτά αντίστοιχη λειτουργικότητα για Helm Charts).

Τα Helm Charts προσομοιάζουν στους Kubernetes operators από την άποψη ότι και οι 2 λύσεις συμπυκνώνουν πολλαπλά αντικείμενα κάτω από μια αφαιρετική έννοια (Chart και

Custom resource αντίστοιχα). Η προφανής διαφορά τους ωστόσο, είναι ότι μόνο οι operators επεκτείνουν το Kubernetes API, γεγονός που καθιστά την ανάπτυξή τους πιο σύνθετη από την δημιουργία ενός Helm Chart.

2.3 Kubeflow

2.3.1 Επισκόπηση

Οι πολλαπλές δυνατότητες και λειτουργικότητες που προσφέρει το Kubernetes έχουν ως αποτέλεσμα να ανεβαίνει και το επίπεδο δυσκολίας στη χρήση του, αφού απαιτείται ένα ευρύ υπόβαθρο γνώσεων στα υπολογιστικά συστήματα και τα δίκτυα. Η ανάγκη για διεύρυνση της προσβασιμότητάς του και σε άτομα εκτός του συγκεκριμένου τεχνικού κύκλου (πχ προγραμματιστές, μηχανικούς δεδομένων) αποτέλεσε έναν από τους βασικούς κινητήριους μοχλούς για την σύλληψη του Kubeflow.

Το Kubeflow προσφέρει μια πλήρη σουίτα εργαλείων που τρέχουν πάνω σε Kubernetes clusters και αφορούν ροές εργασίας μηχανικής μάθησης (machine learning pipelines). Η άνοδός του τα τελευταία έτη έδωσε τη δυνατότητα ανάπτυξης, ελέγχου και εγκατάστασης μοντέλων σε παραγωγικά περιβάλλοντα, γρήγορα και πλήρως αυτοματοποιημένα.

2.3.2 Kubeflow Pipelines

Για την προκείμενη διπλωματική, κύρια εστία ενδιαφέροντος αποτελούν οι διοχετεύσεις Kubeflow (Kubeflow Pipelines). Ο χρήστης δύναται να ορίσει προγραμματιστικά μια ολοκληρωμένη ακολουθία από βήματα (components) που πρέπει να εκτελεστούν. Μπορεί ακόμα να ορίσει εξαρτήσεις μεταξύ των διαφόρων βημάτων, και να χρησιμοποιήσει την έξοδο ενός βήματος ως είσοδο σε κάποιο επόμενο. Εφόσον όλα τα βήματα της διοχέτευσης έχουν επιτυχημένη κατάληξη, θεωρείται ότι και η ίδια η διοχέτευση έτρεξε επιτυχημένα. Σε αντίθετη περίπτωση, η διοχέτευση θα έχει αποτύχει.

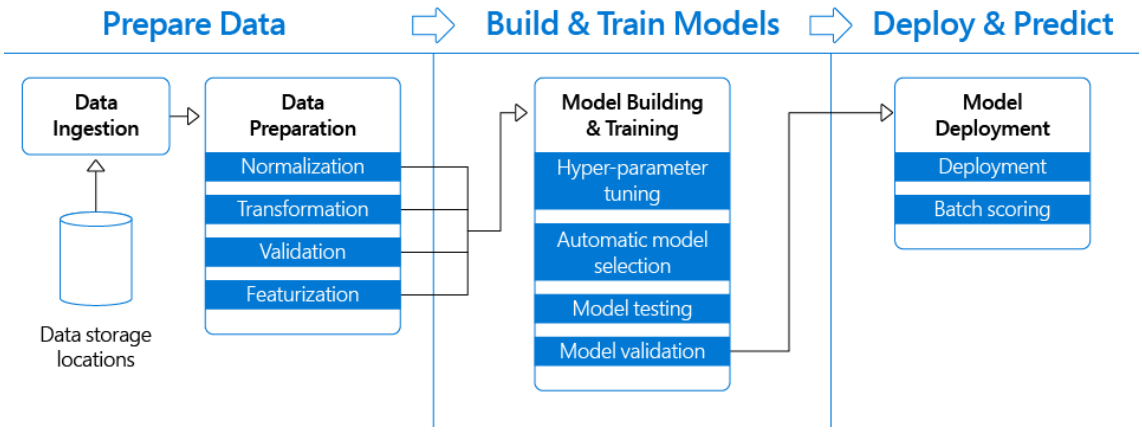
Αφού οριστεί προγραμματιστικά η επιθυμητή διοχέτευση, θα πρέπει αρχικά να μεταγλωτιστεί (compile) σε γλώσσα YAML και εν συνεχεία να υποβληθεί στο Kubeflow Pipelines API προς εκτέλεση. Η πορεία της εκτέλεσής της μπορεί να παρακολουθηθεί είτε από το πεδίο status των προκύπτοντων Kubernetes αντικειμένων, είτε από τη γραφική διεπαφή (User Interface) του Kubeflow.

Τα βήματα που συνθέτουν μια διοχέτευση Kubeflow μπορεί να εκτείνονται από την εκτέλεση ενός απλού Python [26] / shell script, μέχρι και την αλληλεπίδραση με το Kubernetes API. Δηλαδή, αν και οι συγκεκριμένες διοχετεύσεις χρησιμοποιούνται παραδοσιακά για εκτέλεση ακολουθιών μηχανικής μάθησης, οι δυνατότητές τους ξεπερνούν κατά πολύ το συσγκεκριμένο κομμάτι.

2.3.3 Διοχετεύσεις μηχανικής μάθησης

Σε αυτό το σημείο, κρίνεται χρήσιμο να γίνει μια σύντομη αναφορά στα συνηθέστερα βήματα που συνθέτουν μια διοχέτευση μηχανικής μάθησης. Σε μια τέτοια διοχέτευση, έχοντας ως είσοδο έναν πίνακα με ορισμένες στήλες - χαρακτηριστικά (features), σκοπός αποτελεί να

προβλεφθεί η τιμή ενός χαρακτηριστικού - στόχου το οποίο είναι ασυμπλήρωτο (target feature), σε έναν άλλο πίνακα ιδίας μορφής. Ένα παράδειγμα τέτοιας διοχέτευσης: θεωρώντας γνωστά τα στοιχεία για το 80% των αγοραπωλησιών αυτοκινήτων που πραγματοποιήθηκαν στην Ελλάδα το διάστημα 2000 - 2005, να εκτιμηθεί η τιμή πώλησης των αυτοκινήτων για το υπόλοιπο 20% αυτών.



Σχήμα 2.6: *Τα διάφορα στάδια μιας διοχέτευσης μηχανικής μάθησης*

- Προπαρασκευή δεδομένων (Preprocessing): Θεωρώντας ότι τα προς επεξεργασία δεδομένα είναι ήδη διαθέσιμα (παραδείγματως χάρην από κάποιο σύνολο δεδομένων ανοιχτού λογισμικού), πρώτο μέλημα αποτελεί η προετοιμασία τους. Αυτή μπορεί να περιλαμβάνει αφαίρεση και δημιουργία νέων χαρακτηριστικών (χρησιμοποιώντας τα προϋπάρχοντα), κανονικοποίηση των τιμών του πίνακα μεταξύ κάποιου συγκεκριμένου εύρους αξιών, αναπαράσταση των εξαρτήσεων ανάμεσά τους, και άλλα.

- Επιλογή μοντέλου: Ένα μοντέλο (model) αποτελεί ένα υπολογιστικό σύστημα με συμπεριφορά ρυθμιζόμενη από μία σειρά αριθμητικών παραμέτρων. Υπάρχουν πολλά διαφορετικά είδη μοντέλων, και η επιλογή μεταξύ τους εξαρτάται από τη φύση των δεδομένων εισόδου. Σε αρκετές περιστάσεις, προκαταρκτικά επιλέγονται πολλαπλά μοντέλα με πολλαπλούς διαφορετικούς συνδυασμούς αριθμητικών παραμέτρων, εκ των οποίων προκύπτει ένα / μια ομάδα βέλτιστων μοντέλων για το πρόβλημα.

- Προπόνηση μοντέλου (Training): Κατά την προπόνηση ενός μοντέλου, υποβάλλονται σε αυτό το 'προπονητικό' (απολύτως συμπληρωμένο) μέρος των δεδομένων εισόδου. Βάσει αυτού, το μοντέλο καλείται να κατανοήσει τη φύση των δεδομένων και τις συσχετίσεις μεταξύ των διαφόρων χαρακτηριστικών τους.

- Αξιολόγηση μοντέλου (Evaluation): Πριν τρέξει το μοντέλο στα πραγματικά δεδομένα, ελέγχεται η απόδοσή του πάνω σε πειραματικά δεδομένα, αποκρύπτοντας από αυτά τη συμπληρωμένη τιμή του χαρακτηριστικού - στόχου. Συγκρίνοντας τις αποκλίσεις των προβλέψεών του από τις συμπληρωμένες τιμές, προκύπτει το/α καταλληλότερο/α μοντέλο/α για την τελική φάση πρόβλεψης.

- Πρόβλεψη (Prediction): Το μοντέλο τρέχει πλέον στα πραγματικά δεδομένα και προβλέπει τις τιμές του χαρακτηριστικού - στόχου.

2.4 Apache Spark

2.4.1 Επισκόπηση

Το Apache Spark αποτελεί ένα σύστημα κατανεμημένης επεξεργασίας δεδομένων με μια σημαντική ιδιαιτερότητα: τα δεδομένα παραμένουν στη μνήμη των κόμβων που απαρτίζουν μια συστοιχία Apache Spark μεταξύ των διαφόρων σταδίων επεξεργασίας τους. Η συγκεκριμένη προσέγγιση αποδίδει καλά στην εφαρμογή επαναληπτικών αλγορίθμων, καθιστώντας το συγκεκριμένο εργαλείο ιδιαίτερα χρήσιμο για ροές μηχανικής μάθησης, καθώς και προβλήματα 'διαίρει και βασίλευαι' (πχ MapReduce [27]).

Διαθέτει την δική του εσωτερική αναπαράσταση δεδομένων, Resilient Distributed Datasets (RDD), ενώ παρέχει ακόμη και τα Dataframes (στα πρότυπα των κλασσικών πινάκων μιας σχεσιακής βάσης δεδομένων). Όσον αφορά τα Dataframes, το Apache Spark προσφέρει το Dataframe API στις προγραμματιστικές γλώσσες Python, Java [28], Scala [29] και R [30], ενώ πρόσφατα ενσωματώθηκε και εγγενής υποστήριξη για τα pandas dataframes (τα pandas [31] αποτελούν εκ των δημοφιλέστερων Python βιβλιοθηκών για επεξεργασία δεδομένων).

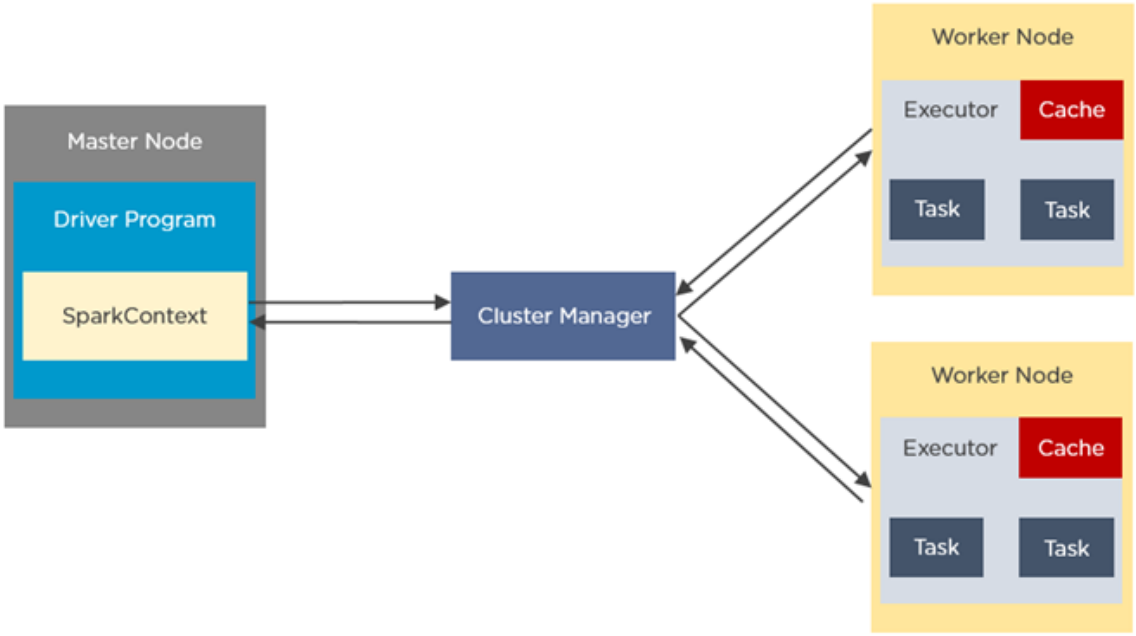
2.4.2 Αρχιτεκτονική

Οι διεργασίες που απαρτίζουν μια εγκατάσταση Apache Spark διακρίνονται σε οδηγοί (drivers) και εκτελέστριες (executors). Οι οδηγοί λαμβάνουν τον κώδικα του χρήστη και τον διασπούν σε πολλαπλά στάδια εκτέλεσης (tasks), που προκύπτει από την εσωτερική αναπαράστασή του ως γράφο σταδίων (Directed Acyclic Graph). Στη συνέχεια, μοιράζουν εργασίες στις εκτελέστριες, οι οποίες τις εκτελούν και ενημερώνουν τους οδηγούς ανά τακτά διαστήματα τόσο για την πορεία εκτέλεσης, όσο και τα τελικά αποτελέσματα. Σημαντική λεπτομέρεια απότελει ότι αν και δύνανται να υπάρχουν πολλαπλοί οδηγοί, μόνο μία μπορεί να είναι ενεργή ανά πάσα στιγμή (οι υπόλοιπες βρίσκονται σε καθεστώς αναμονής), κατά τα πρότυπα ιεραρχιών active - standby. Οι διεργασίες οδηγοί εκτελούνται αποκλειστικά σε πρωταρχικούς Apache Spark κόμβους (masters), ενώ οι εκτελέστριες σε δευτερεύοντες κόμβους (slaves).

Οι κυριότερες οντότητες μιας συστοιχίας Apache Spark θα αναπτυχθούν παρακάτω.

- Spark Core: Ο πυρήνας του Apache Spark συνδυάζει τη μηχανή υπολογισμού (Compute Engine) και ένα σύνολο από διεπαφές προγραμματισμού εφαρμογών. Η μηχανή υπολογισμού αναλαμβάνει τη διαχείριση μνήμης, τον χρονοπρογραμματισμό των εργασιών, την ανάνηψη από σφάλματα καθώς και την αλληλεπίδραση με εξωτερικά μέρη της συστοιχίας. Αναφορικά τώρα με τα APIs, αφορούν στη μεταχείριση RDDs και Dataframes.

- Cluster resource manager: Ο διαχειριστής πόρων είναι ένα σύστημα στο οποίο απευθύνεται μια διεργασία οδηγός όταν θέλει να δημιουργήσει μία ή περισσότερες εκτελέστριες εργασιών. Μπορεί να βρίσκεται εκτός του Apache Spark cluster (πχ Apache Hadoop Yarn [32], **Kubernetes**) ή εντός του (Spark standalone - δεν προτείνεται σε παραγωγικά περιβάλλοντα).



Σχήμα 2.7: *Συνοπτική αρχιτεκτονική του Apache Spark*

- Distributed storage: Το κατανεμημένο σύστημα αποθήκευσης αποτελεί το πεδίο άντλησης των προς επεξεργασία δεδομένων, αλλά και αποθήκευσής τους μετά το πέρας της. Μπορεί να είναι είτε μια πηγή δυναμικών δεδομένων ροής (παραδείγματος χάρην στατιστικά χρήσης μιας ιστοσελίδας ανά δευτερόλεπτο) ή κάποια 'λίμνη' δεδομένων (data lake), όπως το **Hadoop Distributed File System (HDFS) [33]**.

Τέλος, κρίνεται σκόπιμο να γίνει μια σύντομη αναφορά στα Spark Sessions, τα οποία δημιουργούνται από τις διεργασίες οδηγούς κατά την υποβολή μιας εφαρμογής προς εκτέλεση. Αποτελούν μια δομή που δίνει πρόσβαση στις διάφορες βιβλιοθήκες του Apache Spark και διατηρεί τις απαραίτητες πληροφορίες για τις εκτελέστριες.

2.5 Hadoop Distributed File System

2.5.1 Επισκόπηση

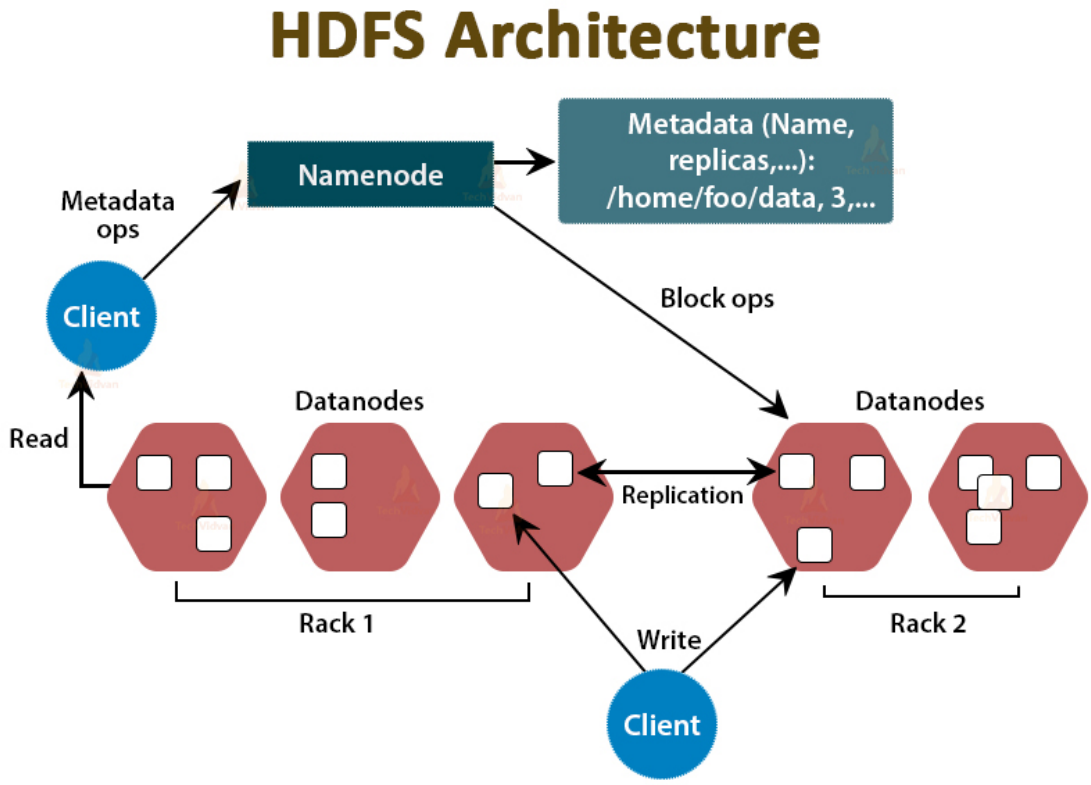
Το Hadoop Distributed File System (HDFS) αποτελεί ένα κατανεμημένο σύστημα αποθήκευσης δεδομένων που εξασφαλίζει κλιμακωσιμότητα και ανθεκτικότητα στα σφάλματα. Μπορεί να τρέξει σε εξυπηρετητές του εμπορίου (bare-metal servers) δίχως κάποια εξειδικευμένη απαίτηση σε υλικό. Το γεγονός αυτό το καθιστά ελκυστικό για χρήση ως παραγωγική 'λίμνη' δεδομένων, καθώς οι εταιρείες τείνουν να προμηθεύονται υλικό από πολλούς διαφορετικούς παραγωγούς (vendors) ανά περίσταση. Το HDFS διαθέτει όλες τις κλασσικές λειτουργίες ενός συστήματος αρχείων όπως δημιουργία / ενημέρωση / διαγραφή αρχείων / φακέλων.

Το HDFS αποτελεί μέρος του οικοσυστήματος προγραμματιστικών λύσεων Apache Hadoop. Βασίζεται κατά κύριο λόγω στην προγραμματιστική γλώσσα Java, ενώ κάποια ακόμα από τα κυριότερα στελέχη του αποτελούν το Hadoop MapReduce (μια υλοποίηση του αλγορίθμου

Map-Reduce), το Hadoop Yarn (ένας διαχειριστής υπολογιστικών πόρων), και το Hadoop common (η βάση για όλα τα υπόλοιπα Hadoop προϊόντα). Στο ευρύτερο οικοσύστημα εντάσσεται και το Apache Spark, που αναλύθηκε σε προηγούμενη ενότητα.

2.5.2 Αρχιτεκτονική

Όπως και τα Apache Spark και Kubernetes, το HDFS ακολουθεί την αρχιτεκτονική πρωταρχικών - δευτερευόντων κόμβων (master-slave architecture). Τα δεδομένα διασπώνται σε ισομεγέθη κομμάτια (blocks) που καταλαμβάνουν 128 MB (η τιμή αυτή είναι παραμετροποιήσιμη), και στη συνέχεια αποθηκεύονται στη συστοιχία HDFS σε 1 ή (ιδανικά) περισσότερα αντίγραφα.



Σχήμα 2.8: *Συνοπτική αρχιτεκτονική του HDFS*

Οι κυριότερες οντότητες του HDFS περιγράφονται στη συνέχεια.

- Name Node: Ο συγκεκριμένος κόμβος είναι μοναδικός και αναλαμβάνει τη διαχείριση του συστήματος δεδομένων και την αρχική επικοινωνία με τον έξω κόσμο. Καταγράφει όλα τα αρχεία που αυτό περιέχει το σύστημα (σημειώνοντας ποιοι κόμβοι περιέχουν τις υποδιαιρέσεις του καθώς και τα αντίγραφα αυτών) καθώς και άλλα μετα-δεδομένα (metadata). Όπως και στην περίπτωση των οδηγών διεργασιών στο Apache Spark, υπάρχει η δυνατότητα να οριστούν αναπληρωματικοί κόμβοι του ίδιου είδους, έτοιμοι να αναλάβουν δράση σε περιπτώσεις αποτυχίας. Όταν ένας χρήστης επιθυμεί να διαβάσει / γράψει σε κάποιο αρχείο, επικοινωνεί πρώτα με τον συγκεκριμένο κόμβο ώστε να μάθει ποιοι κόμβοι είναι υπεύθυνοι για αυτό.

- Data Node: Οι κόμβοι δεδομένων είναι αυτοί που αποθηκεύουν τα δεδομένα, και έρχονται σε επικοινωνία με τους χρήστες για την πρόσβαση σε αυτά. Βρίσκονται ακόμη σε διαρκή επικοινωνία με τον 'ενοχρηστρωτή' Name Node για να επικυρώσουν ότι είναι υγιείς, στέλνωντας σύντομα μηνύματα (heartbeats). Αν ο Name Node δεν λάβει επιβεβαίωση παρουσίας από έναν κόμβο δεδομένων για κάποιο διάστημα, θα θεωρήσει ότι δεν είναι πλέον λειτουργικός και θα κατανείμει τα δεδομένα του στα υπόλοιπα Data Nodes.

- Secondary Name Node: Οι κόμβοι αυτοί παίρνουν στιγμιότυπα (checkpoints) της κατάστασης του Name Node ανά τακτά χρονικά διαστήματα. Δεν μπορούν ωστόσο να δράσουν ως αναπληρωτές του, παρά μόνο να λάβουν αντίγραφα ασφαλείας του.

- Job / Task Tracker: Ως απλή αναφορά, σχετίζονται με τη λειτουργικότητα MapReduce που υποστηρίζει εγγενώς μια συστοιχία HDFS, και συγκεκριμένα στη συλλογή των τοποθεσιών των δεδομένων που θα υποβληθούν για κάποια MapReduce εργασία.

Η πρόσβαση στο σύστημα αρχείων του HDFS μπορεί να επιτευχθεί με πολλαπλούς τρόπους, οι οποίοι καταγράφονται παρακάτω.

- Γραμμή εντολών, μέσω του hdfs cli client

- Γραφική διεπαφή, ενεργοποιώντας το WebHDFS

- Διεπαφή προγραμματισμού εφαρμογών, ενεργοποιώντας είτε το WebHDFS είτε το HttpFS

- Αρχεία κώδικα, χρησιμοποιώντας την κατάλληλη βιβλιοθήκη ανάλογα την προγραμματιστική γλώσσα

Κεφάλαιο 3

Σχεδιασμός

Στο συγκεκριμένο κεφάλαιο αναπτύσσεται η αρχιτεκτονική και η μορφή του σύστηματος που θα υλοποιηθεί στα πλαίσια της διπλωματικής εργασίας, προσδιορίζοντας όλες τις επιμέρους οντότητές του.

3.1 Μέθοδος

Όπως στοιχειοθετήθηκε και στο κεφάλαιο 1, σκοπός της συγκεκριμένης διπλωματικής εργασίας αποτελεί η διαχείριση του κύκλου ζωής συστοιχιών Apache Spark πλήρως αυτοματοποιημένα, μέσω διοχετεύσεων Kubeflow. Για την προσομοίωση της παραγωγικής λειτουργίας του προκύπτοντος συστήματος, θα εκτελεστεί σε αυτό μια πλήρης ροή μηχανικής μάθησης.

Τα συστατικά στοιχεία που θα απαρτίζουν το τελικό σύστημα παρατίθενται στη συνέχεια, ενώ βάση για το σύστημα αποτελούν 3 εικονικές μηχανές που στεγάζονται σε συστοιχία Openstack [34] του CSLab.

- Kubernetes: Η συστοιχία Kubernetes βρίσκεται στην καρδιά του παρόντος εγχειρήματος, καθώς θα στεγάσει τις διοχετεύσεις Kubeflow. Το επίπεδο ελέγχου θα αποτελείται από έναν κόμβο, και το επίπεδο δεδομένων από 2. Όσον αφορά την κατάσταση του Kubernetes, θα αποθηκεύεται σε συστοιχία etcd η οποία παρομοίως αποτελείται από τους 3 ίδιους κόμβους.

- Kubeflow Pipelines: Οι διοχετεύσεις Kubeflow τρέχουν ως Deployment πάνω στη συστοιχία Kubernetes.

- Apache Spark: Οι συστοιχίες Apache Spark δημιουργούνται, εκτελούν τις εργασίες τους και παροπλίζονται αποκλειστικά μεσω των διοχετεύσεων Kubeflow. Ζουν πάνω στη συστοιχία Kubernetes, και συγκεκριμένα στο επίπεδο δεδομένων της.

- Hadoop Distributed File System: Το κατανεμημένο σύστημα αποθήκευσης HDFS βρίσκεται εξωτερικά της συστοιχίας Kubernetes, και συντίθεται από 1 Name Node και 2 Data Nodes. Χρησιμοποιείται από τις διοχετεύσεις Kubeflow ως 'λίμνη' δεδομένων, τόσο για την άνλτηση όσο και την αποθήκευσή τους.

Αξίζει να σημειωθεί ότι οι 3 κόμβοι προφανώς θα συμμετέχουν και στις 2 συστοιχίες (Kubernetes και HDFS), με τον κόμβο που ανήκει στο επίπεδο ελέγχου του Kubernetes να εκτελεί και χρέη HDFS Name Node (αντίστοιχα και για τους υπόλοιπους 2).

3.2 Προσομοίωση παραγωγικής λειτουργίας

Η ροη μηχανικής μάθησης που θα εκτελεστεί στο σύστημα θα αφορά ένα σύνολο δημογραφικών δεδομένων ανά οικοδομικό μπλοκ της Καλιφόρνιας, για το έτος 1990 [35]. Λαμβάνοντας υπόψη στοιχεία όπως το διάμεσο εισόδημα, τον πληθυσμό και τον αριθμό των νοικοκυριών ανά μπλοκ, θα επιχειρηθεί να προβλεφθεί η διάμεση αξία ενός οικήματος σε κάθε μπλοκ.

Μοναδικό προαπαιτούμενο για την εκτέλεση της συγκεκριμένης ροής είναι η φόρτωση των δεδομένων στο HDFS. Εν συνεχεία περιγράφονται τα στάδια που θα έχει η ροή. Κρίνεται σημαντικό να διευκρινιστεί ότι τα στάδια της ροής δεν ταυτίζονται με τα στάδια της διοχέτευσης Kubeflow που θα υλοποιηθεί.

- Συγκέντρωση δεδομένων: Όπως προαναφέρθηκε, τα δεδομένα θα έχουν φορτωθεί στο HDFS πριν την εκκίνηση της ροής. Στο στάδιο αυτό η διοχέτευση Kubeflow θα επικοινωνήσει με το HDFS ώστε να τα μεταφορτώσει από αυτό.

- Προπαρασκευή δεδομένων: Εδώ, θα εφαρμοστούν στα δεδομένα εισόδου μαθηματικές μετατροπείες όπως κανονικοποίηση, και θα πραγματοποιηθεί αντικατάσταση κάποιων χαρακτηριστικών με πιο ευκρινείς αναπαραστάσεις τους. Ακόμη, τα δεδομένα θα διαχωριστούν σε προπονητικά και πραγματικά μέρη.

- Επιλογή και προπόνηση μοντέλου: Στα πλαίσια της επίδειξης αυτής, θα επιλεγεί ένα μοντέλο βάσει των ποιοτικών χαρακτηριστικών των δεδομένων. Το μοντέλο αυτό θα προπονηθεί πάνω στο προπονητικό μέρος τους.

- Πρόβλεψη: Το μοντέλο θα κληθεί να προβλέψει τη μέση αξία των οικημάτων ανά μπλοκ για τα πραγματικά δεδομένα.

Κεφάλαιο 4

Υλοποίηση

Στο συγκεκριμένο κεφάλαιο σκιαγραφείται η συνολική διαδικασία που ακολουθήθηκε προς την υλοποίηση του συστήματος και της προσομοίωσης που ορίστηκαν στο κεφάλαιο 3.

4.1 Εγκατάσταση οικοσυστημάτων

Πρώτο μέλημα αποτελεί η αρχικοποίηση όλων των επιμέρους υποδομών πάνω στις οποίες θα προχωρήσει η εργασία. Επιπλέον στόχος αποτελεί η πραγματοποίησή της με τον πιο καθαρό και αυτοματοποιημένο τρόπο, στα πλαίσια του δυνατού.

Όπως προαναφέρθηκε, η υποδομή που θα στεγάσει όλα τα παρακάτω αποτελείται από 3 εικονικές μηχανές με 8 GB μνήμη RAM, 64 GB τοπικό αποθηκευτικό χώρο και 4 υπολογιστικούς πυρήνες έκαστος. Οι μηχανές είναι διασυνδεδεμένες μεταξύ τους μέσω τοπικού δικτύου, ενώ η προσβασιμότητα σε αυτές επιτυγχάνεται μέσω εικονικού ιδιωτικού δικτύου (Virtual Private Network).

4.1.1 Συστοιχία Kubernetes

Για την αρχικοποίηση της συστοιχίας Kubernetes επιλέγεται το αντίστοιχο Ansible playbook του Kubespray, *cluster.yml*. Για λόγους καταγραφής, η έκδοση του Kubespray git αποθετηρίου που χρησιμοποιήθηκε είναι **1.17.0**, η οποία έχει ως προκαθορισμένη έκδοση Kubernetes την **1.21.5**.

Υπό αφαιρετική σκοπία, το playbook αυτό κάνει κατά σειρά τα εξής:

- Επιβεβαίωση ότι όλοι οι κόμβοι διαθέτουν εγκατεστημένα τα απαραίτητα πακέτα (όπως η Python)
- Εγκατάσταση και ρύθμιση του Docker σε όλους τους κόμβους
- Εγκατάσταση της συστοιχίας etcd σε όλους τους κόμβους (εκτελείται ως περιέκτης Docker)
- Παραγωγή των απαραίτητων πιστοποιητικών για την κρυπτογραφημένη επικοινωνία μεταξύ etcd και Kubernetes
- Εκκίνηση του επιπέδου ελέγχου του Kubernetes (στον ένα κόμβο)
- Εκκίνηση του επιπέδου δεδομένων του Kubernetes (στους υπόλοιπους 2 κόμβους)

- Περαιτέρω δικτυακή παραμετροποίηση, χρησιμοποιώντας το λογισμικό Calico [36] (δεν κρίνεται σκόπιμη η περαιτέρω εμβάθυνση στο συγκεκριμένο κομμάτι)
- Προσθήκη προαιρετικών πρόσθετων Deployments

Τα επιπλέον προαιρετικά Deployments που θα ρυθμιστούν αναλύονται παρακάτω.

- Kubernetes dashboard: Αποτελεί ένα γραφικό περιβάλλον εποπτείας και διαχείρισης μιας συστοιχίας Kubernetes, προσβάσιμο εντός φυλλομετρητή.
- Metrics server: Προσφέρει δυνατότητα ελέγχου των διαθέσιμων πόρων (μνήμη, υπολογιστικοί πυρήνες) ενός κόμβου ανά πάσα στιγμή, ενώ είναι προαπαιτούμενο για τις λειτουργικότητες αυτοκλιμάκωσης (autoscaling) που προσφέρει το Kubernetes.
- Ingress nginx controller [37] : Υλοποιεί έναν αντεστραμμένο εξυπηρετητή nginx, ο οποίος εκθέτει όλα τα Services της συστοιχίας Kubernetes στον έξω κόσμο. Πρακτικά, συλλέγει όλα τα αντικείμενα τύπου Ingress που δημιουργούνται στο Kubernetes, και τα μεταφράζει σε παραμετροποίηση nginx, την οποία στη συνέχεια εφαρμόζει και στον nginx.

Τα μόνα αρχεία που απαιτούν παραμετροποίηση πριν την εκτέλεση του playbook είναι το *inventory.ini* (προσθήκη των διευθύνσεων IP των κόμβων που θα εγκατασταθεί το Kubernetes), καθώς και το *addons.yml* που καθορίζει ποια προαιρετικά Deployments θα στηθούν στο Kubernetes (και τα 2 αυτά αρχεία βρίσκονται κάτω από τον φάκελο *inventory* στο git αποθετήριο Kubespray.

Για την εκτέλεση του playbook, δημιουργείται και ενεργοποιείται ένα εικονικό περιβάλλον Python 3, στο οποίο εγκαθίστανται όλα τα απαιτούμενα από το Kubespray πακέτα (μεταξύ αυτών και η Ansible). Όλα πλέον είναι έτοιμα για την εκτέλεση του playbook, το οποίο μετά από μισή περίπου ώρα έχει ετοιμάσει τα πάντα στους 3 κόμβους.

Για την απόκτηση προσβασης στην προκύπτουσα συστοιχία Kubernetes, απαιτείται η εγκατάσταση του kubectl στο τοπικό μηχάνημα, καθώς και η αντιγραφή του kubeconfig από τον κόμβο που ανήκει στο επίπεδο ελέγχου αυτής. Το kubeconfig είναι ένα αρχείο YAML το οποίο περιέχει την IP του προαναφερθέντος κόμβου, καθώς και ένα διαχειριστικό πιστοποιητικό (admin certificate) που δίνει πρόσβαση στο σύνολο των πόρων του Kubernetes.

4.1.2 Kubeflow Pipelines

Προτού εγκατασταθεί το Kubeflow Pipelines πάνω στο Kubernetes, εκτελούνται τα εξής βήματα:

- Δημιουργείται ένα διακριτο namespace (ονόματι *kubeflow* στο οποίο θα ζουν όλα τα αντικείμενα που σχετίζονται με τη λειτουργία του
- Στο namespace αυτό, δημιουργούνται 2 Persistent Volumes που θα χρησιμοποιηθούν από τις 2 αποθηκευτικές οντότητες του Kubeflow Pipelines (μία σχεσιακή βάση δεδομένων mysql και μια αποθήκη αντικειμένων minio).

Πλέον, μπορεί να προχωρήσει η εγκατάσταση, για την οποία επιλέγεται η έκδοση **1.7.0** των Kubeflow Pipelines. Η διαδικασία είναι αρκετά απλή, και περιλαμβάνει αρχικά την προσθήκη νέων ορισμών εξειδικευμένων αντικειμένων στο Kubernetes και εν συνεχεία την υποβολή στον kube-apiserver (μέ την εντολή *kubectl apply*) όλων των απαραίτητων αντικειμένων.

Το Kubeflow Pipelines είναι πλέον απολύτως λειτουργικό, και για την πρόσβαση στο γραφικό του περιβάλλον δημιουργείται ένα αντικείμενο Ingress. Το αντικείμενο αυτό δίνει εντολή στον Ingress nginx controller, όταν δέχεται αίτημα πρόσβασης στην διεύθυνση *kubeflow.ui* να το ανακατευθύνει στο Kubeflow Pipelines UI Service (στην πόρτα 80), που έγινε apply προηγουμένως.

Μια τελευταία λεπτομέρεια αποτελεί η παραχώρηση πρόσβασης για δημιουργία / μεταβολή της κατάστασης αντικειμένων σε όλα τα namespaces του Kubernetes στο Kubeflow Pipelines. Αυτό διότι κατά την εκτέλεση μιας διοχέτευσης, δημιουργούνται Pods σε διαφορετικά namespaces από αυτό που ζει το ίδιο το Kubeflow Pipelines. Για το συγκεκριμένο σκοπό πραγματοποιούνται τα εξής βήματα:

- Δημιουργία αντικειμένου είδους ClusterRole, το οποίο πρακτικά κατασκευάζει έναν νέο 'ρόλο' (role) που έχει απόλυτη πρόσβαση σε αντικείμενα κάθε είδους, σε κάθε namespace του Kubernetes. Γενικότερα, στο Kubernetes εφαρμόζεται η φιλοσοφία Role-Based Access Control (RBAC) [38], όπου η δυνατότητα εκτέλεσης κάποιας συγκεκριμένης ενέργειας αποδίδεται σε μια οντότητα μέσω ενος ρόλου που προσφέρει τα αντίστοιχα δικαιώματα. Οι οντότητες με τη σειρά τους εκπροσωπούνται από αντικείμενα τύπου ServiceAccount.

- Δημιουργία ενός ClusterRoleBinding, που 'δένει' το ClusterRole που κατασκευάστηκε παραπάνω με την οντότητα που αντιστοιχεί στο Kubeflow Pipelines.

Αξίζει να σημειωθεί ότι σε παραγωγικά περιβάλλοντα αποθαρρύνεται η παραχώρηση δικαιωμάτων εκτός του namespace που ζει ένα Deployment, για να ελαχιστοποιηθούν οι απώλειες σε περίπτωση κατάληψής αυτού από κακόβουλες οντότητες.

4.1.3 Hadoop Distributed File System

Σειρά έχει η εκκίνηση της κατανεμημένης συστοιχίας αποθηκευτικού χώρου HDFS. Η συγκεκριμένη εγκατάσταση απαιτεί τα περισσότερα χειροκίνητα βήματα από όσες περιγράφηκαν μέχρι στιγμής, γεγονός που αποδίδεται στην απουσία σύγχρονων εργαλείων αυτοματοποίησής της. Στα πλαίσια της παρούσας καταγραφής, κρίνεται χρήσιμη μονάχα μια επιφανειακή απαρίθμηση των βημάτων που ακολουθούνται.

Η παρακάτω διαδικασία απαιτεί ταυτόχρονη πρόσβαση και εκτέλεση βημάτων και στους 3 κόμβους ταυτοχρόνως. Για τη διευκόλυνσή της, χρησιμοποιείται η λειτουργικότητα μετάδοσης εντολών σε πολλαπλά τερματικά (broadcast) που προσφέρει ο προσομοιωτής τερματικών terminator.

- Εγκατάσταση της Java 8 σε όλους τους κόμβους

- Δημιουργία διακριτού χρήστη *hadoop* σε όλους τους κόμβους, με χρήση του οποίου θα εκτελεστούν τα υπόλοιπα βήματα
- Δημιουργία ζεύγους δημοσίου-ιδιωτικού κλειδιού στον (μέλλοντα) Name Node, και τοποθέτηση του δημοσίου κλειδιού στα 2 Data Nodes
- Λήψη του Hadoop και ορισμός απαραίτητων περιβαλλοντικών μεταβλητών σε όλους τους κόμβους (έκδοση **3.3.1**)
- Τοποθέτηση απαραίτητων παραμετροποιήσεων στα αρχεία *hdfs-site.xml*, *core-site.xml* και *workers* - τα σημαντικά στοιχεία αυτών θα αναφερθούν ξεχωριστά
- Καθαρισμός (format) και εκκίνηση του HDFS στο Name Node

Πλέον, η συστοιχία HDFS είναι λειτουργική, και το γραφικό περιβάλλον της ακούει στην πόρτα 9870 του Name Node.

Όσον αφορά τώρα την επιπλέον παραμετροποίηση που έγινε:

- Ενεργοποιήθηκε το WebHDFS, το οποίο δίνει την δυνατότητα ανάγνωσης / εγγραφής αρχείων στο HDFS μέσω ενός RESTful API. Αυτό χρησιμεύει τόσο για την διαχείριση αρχείων από το γραφικό περιβάλλον του HDFS, όσο και για την αλληλεπίδραση με αυτό μέσα από αρχεία Python (θα είναι απαραίτητο αργότερα).
- Επιτρέπεται σε οποιονδήποτε να κάνει αλλαγές στο σύστημα αρχείων του HDFS (η προκαθορισμένη ρύθμιση είναι να επιτρέπεται μόνο σε συγκεκριμένους χρήστες). Η συγκεκριμένη ρύθμιση προφανώς και αποθαρρύνεται σε παραγωγικά περιβάλλοντα, απλώς στην προκειμένη περίπτωση ήταν μια βολική λύση ώστε να μπορεί ένα Kubernetes container να επικοινωνήσει με το HDFS.

4.2 Ανάπτυξη δομικών Kubeflow Pipelines components

Για να είναι εφικτή η μεταχείριση συστοιχιών Apache Sparks μέσω μιας διοχέτευσης Kubeflow, χρειάζεται πρώτα να δημιουργηθούν ορισμένα components για το Kubeflow Pipelines. Τα components αυτά είναι γενικού προσανατολισμού, δηλαδή μπορούν να αξιοποιηθούν για πλειάδα σκοπών πέρα από τον στόχο αυτής της διπλωματικής εργασίας.

Ο κώδικας YAML των components που αναπτύχθηκαν επισυνάπτονται στο Παράρτημα Α΄.

4.2.1 Διαχείριση αντικειμένων Kubernetes

Η πρώτη συλλογή Kubeflow Pipelines components που αναπτύσσεται αφορά στην δημιουργία και διαγραφή αντικειμένων Kubernetes. Πιο αναλυτικά:

- **Apply Kubernetes resource** (Σχήμα Α.1): Λαμβάνει ως ορίσμα ένα αρχείο YAML που περιέχει ακριβώς ένα αντικείμενο Kubernetes. Στη συνέχεια, το υποβάλλει στον kube-apiserver της συστοιχίας Kubernetes που τρέχει και το ίδιο το Kubeflow Pipelines (τρέχοντας την εντολή *kubectl apply*). Ως έξοδος επιστρέφεται το όνομα και το είδος

του αντικειμένου που δημιουργήθηκε στο Kubernetes, καθώς και το namespace στο οποίο ζει.

- **Delete Kubernetes resource** (Σχήμα A.2): Λαμβάνει ως ορίσματα το είδος και το όνομα του αντικειμένου που θα διαγραφεί, καθώς και το namespace στο οποίο ζει. Εν συνεχεία, το διαγράφει από το Kubernetes τρέχοντας την εντολή *kubectl delete*.

Και τα 2 components εκτελούνται μέσα σε containers που τρέχουν το *kubectl* image της bitnami [39] (με προεγκατεστημένα το CLI client του Kubernetes, kubectl).

4.2.2 Διαχείριση Helm Charts

Η δυνατότητα εγκατάστασης, αναβάθμισης και διαγραφής Helm Charts από το Kubernetes είναι πολύ χρήσιμη, και μπορεί επίσης να επιτευχθεί μέσω διοχετεύσεων Kubeflow. Πιο συγκεκριμένα:

- **Install or Upgrade Helm Chart** (Σχήμα A.4): Παίρνει ως ορίσματα τη διεύθυνση (URL) ενός Helm αποθετηρίου, το όνομα του αποθετηρίου, το όνομα και την έκδοση του Helm Chart που θα εγκατασταθεί / αναβαθμιστεί, ένα αρχείο που περιέχει τα Helm Values, καθώς και το όνομα και namespace που θα ζει το Helm Chart. Εν συνεχεία, κατά πρώτον προσθέτει το αποθετήριο Helm στα αποθετήρια από τα οποία το helm cli client τραβάει Charts (με την εντολη *helm repo add*), και ελέγχει ποια Charts υπάρχουν διαθέσιμα σε αυτό (με την εντολή *helm repo update*). Κατά δεύτερον, εγκαθιστά το Chart στο namespace που ζητήθηκε, εφόσον δεν είναι ήδη εγκατεστημένο. Διαφορετικά, το αναβαθμίζει σε περίπτωση που υπάρχουν διαφορές στα Helm Values που δώθηκαν ως είσοδος σε σύγκριση με τα υπάρχοντα.

- **Uninstall Helm Chart** (Σχήμα A.5): Δέχεται ως ορίσματα το όνομα του Helm Chart που θα απεγκατασταθεί, καθώς και το namespace στο οποίο ζει. Στη συνέχεια, το απεγκαθιστά από το Kubernetes τρέχοντας την εντολή *helm uninstall*.

Και τα 2 components εκτελούνται μέσα σε containers που τρέχουν το δημοφιλές *helm-kubectl* image [40] του αποθετηρίου DockerHub (με προεγκατεστημέα τα CLI clients kubectl και helm).

4.2.3 Έλεγχος κατάστασης εκτέλεσης SparkApplications

Όπως προαναφέρθηκε στην υποενότητα 2.2.4, τα SparkApplications αποτελούν τα εξειδικευμένα αντικείμενα του Kubernetes τα οποία όταν υποβληθούν στον kube-apiserver, έχουν ως αποτέλεσμα την εκκίνηση μιας συστοιχίας Apache Spark. Προϋπόθεση φυσικά για να κατανοεί ο kube-apiserver τα συγκεκριμένα αντικείμενα αποτελεί η εγκατάσταση του spark-on-k8s operator. Για την ώρα, ας γίνει η υπόθεση ότι τηρούνται αυτές οι συνθήκες, και ότι κατά κάποιο τρόπο έχει εκκινηθεί και εκτελείται μια συστοιχία Apache Spark πάνω στο Kubernetes.

Στα πλαίσια ενός Kubeflow Pipeline, η εκτέλεση μιας συστοιχίας Apache Spark δύναται να αποτελεί ένα μόνο βήμα σε μια σειρά από διαδικασίες. Ενδέχεται λοιπόν κάποιο επόμενο

βήμα να πρέπει να περιμένει την ολοκλήρωση των εργασιών του Apache Spark για να λάβει κάποια αποτελέσματα ως είσοδο. Ακόμα όμως και να μην είναι αυτή η περίπτωση, εφόσον το Apache Spark έχει αναλάβει την εκτέλεση μιας αποστολής, πρέπει με κάποιο τρόπο να γίνει αντιληπτό αν την έφερε εις πέρας. Και αυτό πλήρως αυτοματοποιημένα, μέσα από τη διοχέτευση Kubeflow και χωρίς ανθρώπινη παρέμβαση.

Για το σκοπό αυτό, αναπτύχθηκε ένα ειδικό Kubeflow Pipelines component (**Verify SparkApplication's completion** - Σχήμα A.3) το οποίο ελέγχει ανά τακτά διαστήματα (με τη μέθοδο polling) την κατάσταση της συστοιχίας Apache Spark. Το component λαμβάνει ως είσοδο το όνομα του SparkApplication (που ισοδυναμεί με το όνομα της συστοιχίας), το namespace στο οποίο ζει, την επιθυμητή κατάσταση στην οποία πρέπει εν τέλει να επέλθει η συστοιχία και την 'προθεσμία' (timeout) που έχει για να συμβεί αυτό. Εν συνεχεία, ελέγχει ανά 5 δευτερόλεπτα την κατάσταση της συστοιχίας μέσω του πεδίου status που περιέχεται στο αντικείμενο SparkApplication. Η πρόσβαση στο συγκεκριμένο πεδίο αποκτάται μέσω της εντολής *kubectl get*. Ο έλεγχος συνεχίζεται έως ότου η κατάσταση να είναι η επιθυμητή, ή να τελειώσει η προθεσμία. Ως έξοδος επιστρέφεται η κατάσταση του Apache Spark μετά το πέρας του ελέγχου.

Στην περίπτωση που ελέγχεται εφόσον το Apache Spark ολοκλήρωσε κανονικά τις εργασίες του (που είναι και ο λόγος που αναπτύχθηκε το συγκεκριμένο component), η επιθυμητή κατάσταση λαμβάνει την τιμή *Complete*. Όσον αφορά το περιβάλλον εκτέλεσης του component, ισχύει ό,τι και για τα υπόλοιπα 2 Kubernetes components.

4.3 Αυτοματοποίηση κύκλου ζωής συστοιχιών Apache Spark

Πλέον, είναι διαθέσιμα όλα τα απαραίτητα εργαλεία ώστε μέσω μιας διοχέτευσης Kubeflow να ελεγχθεί από την αρχή μέχρι το τέλος η ζωή μιας συστοιχίας Apache Spark.

Αρχικά, σκιαγραφούνται τα βήματα που πρέπει να εμπεριέχει μια διοχέτευση, ώστε στα πλαίσιά της να εκτελεστεί μια συστοιχία Apache Spark. Στόχος εδώ αποτελεί η κατάσταση του Kubernetes πριν και μετά το πέρας της διοχέτευσης να είναι πανομοιότυπη, δηλαδή να μην μείνουν 'υπολείμματα' (ορφανά αντικείμενα) στη συστοιχία.

1. Εγκατάσταση του spark-on-k8s operator: Όπως προαναφέρθηκε, για να μπορεί ο kube-apiserver να μεταχειριστεί αντικείμενα SparkApplication, θα πρέπει πρώτα να προστεθούν στο Kubernetes οι ορισμοί αυτών. Αυτό επιτυγχάνεται με την εγκατάσταση του operator.

2. Εκκίνηση συστοιχίας Apache Spark: Εδώ, υποβάλλεται στον kube-apiserver ένα SparkApplication. Αυτό έχει ως αποτέλεσμα την εκκίνηση μιας συστοιχίας Apache Spark.

3. Επικύρωση ολοκλήρωσης εργασιών του Apache Spark: Ελέγχεται ότι οι εργασίες που εκτελεί η συστοιχία ολοκληρώθηκαν επιτυχώς.

4. Απεγκατάσταση του spark-on-k8s operator: Καθώς στόχος είναι να μην αφεθούν υπολείμματα στο Kubernetes, θα πρέπει να αφαιρεθεί ο operator. Πλέον, το Kubernetes δεν κατανοεί αντικείμενα SparkApplication και δεν μπορεί να εκκινήσει συστοιχίες Apache Spark (όπως και πριν την εκτέλεση της διοχέτευσης).

Εν συνεχεία, θα αναλυθεί πώς πραγματοποιείται κάθε βήμα μέσα σε μια διοχέτευση, χρησιμοποιώντας τα components που αναπτύχθηκαν.

4.3.1 Αντιστοίχηση σταδίων διοχέτευσης σε components

Εγκατάσταση του spark-on-k8s operator

Ο ευκολότερος τρόπος που μπορεί να εγκατασταθεί ο spark-on-k8s operator είναι μέσω του Helm Chart που συντηρείται από την ίδια ομάδα. Άρα, τα πράγματα εδώ είναι απλά: χρησιμοποιείται το component **Install or Upgrade Helm Chart**, φροντίζοντας να περαστεί ως όρισμα το αρχείο με τα επιθυμητά Helm Values.

Εκκίνηση συστοιχίας Apache Spark

Η δυνατότητα να περιγραφεί η επιθυμητή συστοιχία μέσω ενός αντικειμένου SparkApplication καθιστά και εδώ ξεκάθαρη την προσέγγιση που θα ακολουθηθεί. Είναι εντυπωσιακό το ότι η δημιουργία μιας ολόκληρης συστοιχίας Apache Spark ισοδυναμεί με την υποβολή ενός απλού YAML αρχείου στον kube-apiserver. Συνεπώς, χρησιμοποιείται το component **Apply Kubernetes resource**.

Στο σημείο αυτό θεωρείται χρήσιμο να γίνει μια σύντομη παρένθεση, εστιάζοντας λίγο παραπάνω στη δομή του εξειδικευμένου αντικειμένου SparkApplication. Τα σημαντικότερα πεδία του (εμφωλιασμένα κάτω από το πεδίο spec) που καθορίζουν τους πόρους που θα διαθέτει και τις εργασίες που θα εκτελέσει η προκύπτουσα συστοιχία Apache Spark, παρατίθενται παρακάτω.

- driver: Κάτω από το πεδίο αυτό καταγράφονται παραμετροποιήσεις όπως τα όρια χρήσης του driver Pod σε μνήμη και υπολογιστικούς πυρήνες καθώς και ένα ή παραπάνω volumeMounts.
- executor: Στην ίδια λογική με το πεδίο driver, εδώ καταγράφονται παραμετροποιήσεις για τα executor Pods. **Μια σημαντική επιπλέον επιλογή που δίνεται εδώ είναι να επιλεγεί ο αριθμός των executor Pods.**
- image: Το Docker image που περιέχει την έκδοση Apache Spark που θα τρέχει η συστοιχία.
- mainApplicationFile: Το αρχείο που περιέχει το σύνολο των εργασιών που θα εκτελέσει η συστοιχία. Το αρχείο αυτό είτε προστίθεται σε κάποιο Volume που στη συνέχεια γίνεται προσβάσιμο για τα Apache Spark Pods μέσω volumeMounts, ή μεταφορτώνεται από κάποιο υπολογιστικό νέφος.
- type: Η προγραμματιστική γλώσσα στην οποία έχει συνταχθεί το παραπάνω αρχείο.

Επικύρωση ολοκλήρωσης εργασιών του Apache Spark

Εδώ, χρησιμοποιείται απευθείας το component **Verify SparkApplication's completion** που αναπτύχθηκε αποκλειστικά για τον σκοπό αυτό.

Απεγκατάσταση του spark-on-k8s operator

Εφόσον ο spark-on-k8s operator έχει εγκατασταθεί ύπο τη μορφή Helm Chart, η απεγκατάστασή του ισοδυναμεί με την αφαίρεση αυτού του Helm Chart (με μία υποσημείωση). Για την αφαίρεση του Helm Chart, χρησιμοποιείται το component **Uninstall Helm Chart**.

Παρατηρήθηκε ωστόσο ότι η απεγκατάσταση του συγκεκριμένου Helm Chart αφήνει ως υπόλειμμα ένα αντικείμενο είδους ServiceAccount. Για την αφαίρεση του συγκεκριμένου αντικειμένου, επιστρατεύεται το component **Delete Kubernetes Resource**. Η διαγραφή του ServiceAccount πρέπει να γίνει αφού έχει ολοκληρωθεί η απεγκατάσταση του Helm Chart, πράγμα που έχει ληφθεί υπόψη κατά την ανάπτυξη του component **Uninstall Helm Chart** (είναι blocking).

Συνεπώς, το συγκεκριμένο βήμα απαιτεί την σειριακή εκτέλεση 2 διαφορετικών Kubeflow Pipeline components, και ισοδυναμεί με 2 διακριτά στάδια εκτέλεσης.

4.4 Ανάπτυξη δοκιμαστικής διοχέτευσης Kubeflow

Για την πρακτική επίδειξη όσων αναπτύχθηκαν παραπάνω, κρίνεται σκόπιμη η δημιουργία και εκτέλεση μιας ενδεικτικής διοχέτευσης Kubeflow. Η δομή της περιγράφηκε αφαιρετικά στην ενότητα 3.2, οπότε επόμενο βήμα αποτελεί η 'μετάφρασή' της σε μια ακολουθία βημάτων στο Kubeflow. Αυτά παρατίθενται κατά σειρά σειριακής εκτέλεσης στη συνέχεια.

1. **Φόρτωση και προπαρασκευή δεδομένων** (ενδεικτικά αναφέρεται ως **Preprocessing** component)
2. Εγκατάσταση του spark-on-k8s operator
3. **Εκκίνηση συστοιχίας Apache Spark, που αναλαμβάνει την προπόνηση του μοντέλου και την πρόβλεψη** (2 σειριακά components)
4. Επικύρωση ολοκλήρωσης εργασιών του Apache Spark
5. Απεγκατάσταση του spark-on-k8s operator (2 σειριακά components)

Η συγκεκριμένη διοχέτευση καταδεικνύει την ευκολία σχηματισμού ροών εργασίας πάνω σε συστοιχίες Apache Spark. Όλη η ειδική παραμετροποίηση για τη διοχέτευση συμπυκνώνεται στα στάδια 1 και 3 (σημειώνονται με **bold**). Αυτά αποτελούν την κύρια εστία ενδιαφέροντος στο εξής, αφού τα υπόλοιπα έχουν ήδη επεξηγηθεί στην ενότητα 4.3.

4.4.1 Φόρτωση και προπαρασκευή δεδομένων

Φύση των δεδομένων

Προτού αναλυθεί το συγκεκριμένο στάδιο, αξίζει να γίνει μια σύντομη επιθεώρηση στα δεδομένα που βρίσκονται αποθηκευμένα στο HDFS (σε μορφή csv) και θα τροφοδοτήσουν τη διοχέτευση.

Ως οικοδομικό μπλοκ ορίζεται μια γεωγραφική περιοχή στην οποία κατοικούν από 600 μέχρι και 3000 άτομα, με 1425.5 άτομα ανά μπλοκ κατά μέσο όρο στο σύνολο των δεδομένων. Ακόμη, τα χαρακτηριστικά που δίνονται για κάθε μπλοκ είναι τα παρακάτω:

- Γεωγραφικό πλάτος του μπλοκ
- Γεωγραφικό μήκος του μπλοκ
- Διάμεσος ηλικία των κατοίκων που απαρτίζουν το μπλοκ
- Άθροισμα αριθμού χώρων που διαθέτουν όλα τα οικήματα που απαρτίζουν το μπλοκ
- Άθροισμα αριθμού υπνοδωματίων που διαθέτουν όλα τα οικήματα που απαρτίζουν το μπλοκ
- Πληθυσμός του μπλοκ
- Αριθμός οικημάτων του μπλοκ
- Διάμεσο εισόδημα των κατοίκων που απαρτίζουν το μπλοκ
- **Διάμεση αξιά των οικημάτων που απαρτίζουν το μπλοκ** (χαρακτηριστικό - στόχος)

Περιβάλλον εκτέλεσης

Η φόρτωση και η προπαρασκευή των δεδομένων γίνεται χρησιμοποιώντας την προγραμματιστική γλώσσα Python, μέσω δηλαδή ενός Python Kubeflow Pipeline component. Το συγκεκριμένο component εκτελείται μέσα σε Kubernetes container, που πρέπει να τρέχει ένα image με εγκατεστημένο τόσο το πακέτο Python, όσο και τις απαραίτητες βιβλιοθήκες (στην προκειμένη περίπτωση pandas και sklearn [41]).

Για τον σκοπό αυτό, δημιουργήθηκε ένα Dockerfile που βασίζεται στο επίσημο *python* image [42] που ζει στο αποθετήριο Dockerhub και του προσθέτει επιπλέον τα απαραίτητα πακέτα. Εν συνεχεία, το image έγινε build τοπικά και μεταφορτώθηκε σε προσωπικό Dockerhub αποθετήριο.

Ανάλυση της διαδικασίας

Όσον αφορά τώρα τη μεθοδολογία που ακολουθήθηκε, συνοψίζεται παρακάτω.

1. Μεταφόρτωση των δεδομένων από το HDFS, υποβάλλοντας ένα GET αίτημα (request) στο WebHDFS REST API
2. Δημιουργία ενός pandas dataframe
3. Αναγωγή των διαμέσων αξιών των οικημάτων σε εκατομμύρια δολάρια
4. Υπολογισμός και προσθήκη νέων χαρακτηριστικών στα δεδομένα (χώροι, υπνοδωμάτια και κάτοικοι ανά σπίτι)
5. Αφαίρεση των χαρακτηριστικών γεωγραφικό μήκος, γεωγραφικό πλάτος, διάμεσος ηλικία κατοίκων και συνολικός αριθμός χώρων
6. Κανονικοποίηση όλων των χαρακτηριστικών πέραν του χαρακτηριστικού - στόχου
7. Αποθήκευση των επεξεργασμένων δεδομένων στο HDFS, με POST request στο WebHDFS.

4.4.2 Προπόνηση του μοντέλου και πρόβλεψη

Περιβάλλον εκτέλεσης

Όπως προαναφέρθηκε, η προπόνηση του μοντέλου και η πρόβλεψη θα πραγματοποιηθούν εντός μιας νεοσύστατης συστοιχίας Apache Spark. Ο τρόπος που αυτή θα εκκινηθεί έχει αναλυθεί ήδη (μέσω αντικειμένου SparkApplication), οπότε θα γίνει επισκόπηση μόνο 2 θεμάτων με μεγαλύτερη λεπτομέρεια.

Όσον αφορά το image που χρησιμοποιείται για τα containers, δημιουργείται ένα Dockerfile που βασίζεται στο *spark-operator/spark-py* image [43] που ζει στο ιδιωτικό αποθετήριο Docker του Google Cloud (με έκδοση **3.1.1**). Πάνω σε αυτό προστίθεται μία στρώση που περιέχει τις βιβλιοθήκες Python numpy [44] και koalas [45]. Παρομοίως με πριν, το image γίνεται build και μεταφορτώνεται σε προσωπικό αποθετήριο DockerHub.

Όσον αφορά τον τρόπο που φορτώνονται οι εργασίες που θα τρέξουν στη συστοιχία Apache Spark, καταγράφονται σε αρχείο Python το οποίο αποθηκεύεται μέσα σε ένα αντικείμενο Configmap (πρακτικά ένα key-value storage). Εν συνεχεία, τα περιεχόμενα του Configmap φορτώνονται σε ένα Volume, και το Volume αυτό εφάπτεται στα Apache Spark Pods. Πρακτικά λοιπόν, το αντικείμενο Configmap πρέπει να υποβληθεί στον kube-apiserver πριν από το SparkApplication, για να μπορέσει η συστοιχία Apache Spark να αντλήσει τις εργασίες που θα πρέπει να εκτελέσει κατά τη σύστασή της. Έτσι, το συγκεκριμένο στάδιο συμπεριλαμβάνει την σειριακή εκτέλεση 2 **Apply Kubernetes resource** components.

Ανάλυση των εργασιών

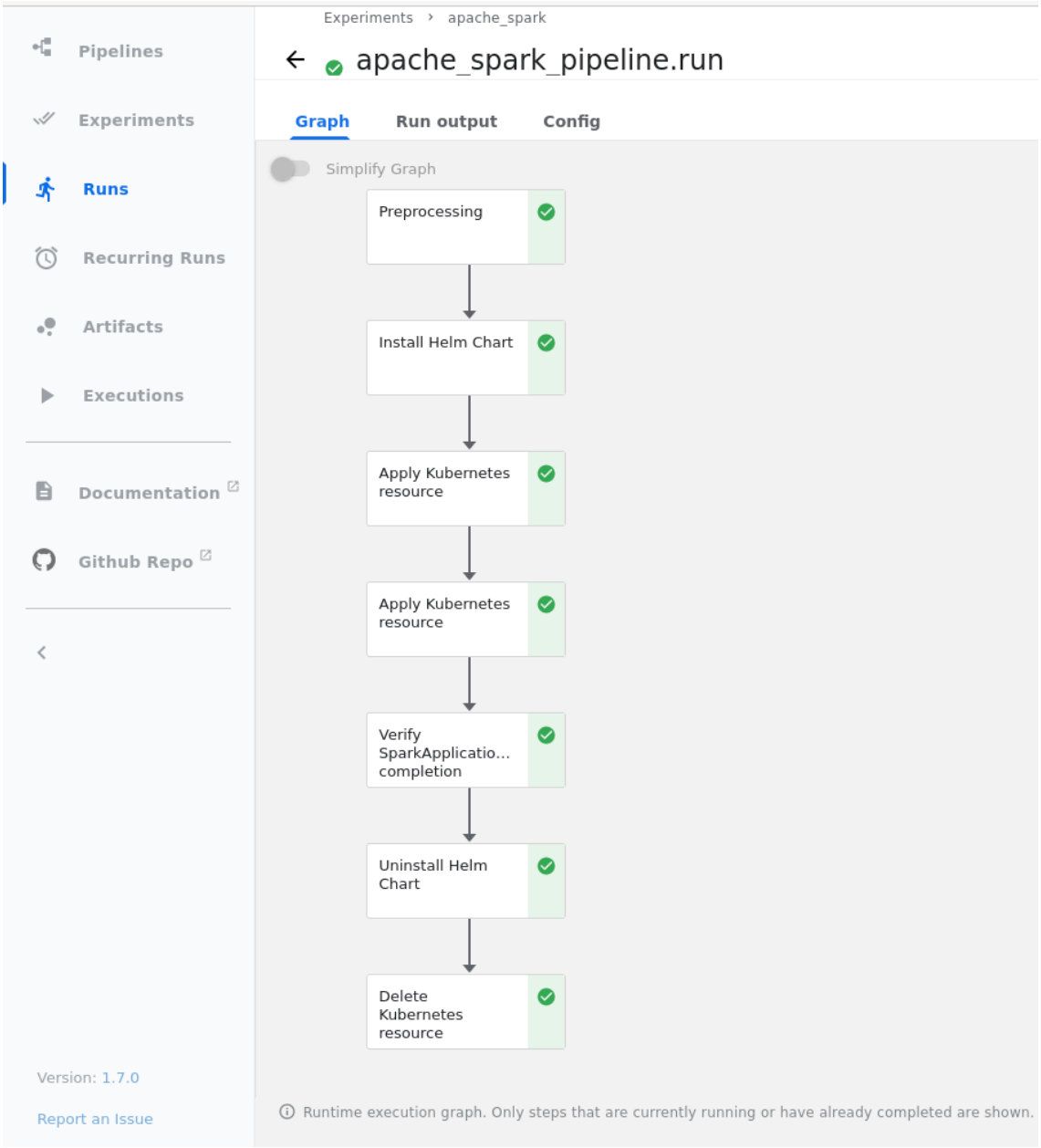
Για την προπόνηση του μοντέλου και την παραγωγή των προβλέψεων, έγιναν περιληπτικά τα εξής:

- Δημιουργία SparkContext και μεταφόρτωση των προπαρασκευασμένων δεδομένων από το HDFS
- Τυχαίος διαχωρισμός των δεδομένων σε προπονητικά και πραγματικά, με αναλογία 80/20
- Εφαρμογή μοντέλου γραμμικής παλινδρόμησης στο προπονητικό τμήμα
- Παραγωγή προβλέψεων και μετρικών αξιολόγησης της ακρίβειάς τους

4.4.3 Εκτέλεση της διοχέτευσης

Η διοχέτευση που περιγράφηκε καταχωρείται σε αρχείο Python, παραθέτοντας ένα-ένα τα components που θα πρέπει να δημιουργηθούν και να τρέξουν, καθώς και τις μεταξύ τους εξαρτήσεις. Το αρχείο αυτό μεταγλωτίζεται και παράγεται ένα αντικείμενο YAML, έτοιμο προς υποβολή στον kube-apiserver. Η υποβολή του γίνεται εκ νέου μέσω Python αρχείου, στο οποίο καθορίζονται και οι τιμές διαφόρων μεταβλητών.

Μέσω της γραφικής διεπαφής του Kubeflow (αλλά και αλληλεπίδρασης με το Kubernetes μέσω του kubectl) παρακολουθείται η πορεία εκτέλεσης της διοχέτευσης. Η τελική εικόνα της εκτέλεσης, όπως φαίνεται στη γραφική διεπαφή, φαίνεται στο Σχήμα 4.1.



Σχήμα 4.1: *Επιτυχημένη διοχέτευση Kubeflow*

Κεφάλαιο 5

Συμπεράσματα

Σε αυτό το κεφάλαιο πραγματοποιείται μια σύντομη ανασκόπηση της εργασίας και των συνεισφορών που προέκυψαν από αυτή, και παραθέτονται πιθανές μελλοντικές επεκτάσεις της.

5.1 Ανακεφαλαίωση και συνεισφορά

Συνοψίζοντας, στα πλαίσια της παρούσας εργασίας σχεδιάστηκαν και υλοποιήθηκαν τα εξής:

- Kubeflow Pipelines components για τον χειρισμό Kubernetes αντικειμένων και Helm Charts. Τα components σχεδιάστηκαν ώστε να μπορούν να ενταχθούν σε οποιαδήποτε διοχέτευση Kubeflow γενικής χρήσης. Για τον λόγο αυτό, δημιουργήθηκε pull request στο αποθετήριο Github των Kubeflow Pipelines, ώστε να ενταχθεί μέρος αυτών στα προτεινόμενα προς χρήση components [46].
- Μια ακολουθία βημάτων Kubeflow μέσα από την οποία μπορεί να ελεγχθεί πλήρως αυτοματοποιημένα ο κύκλος ζωής συστοιχιών Apache Spark, οι οποίες θα εκτελούνται επιτόπια στη συστοιχία Kubernetes που τρέχει το Kubeflow. Η ακολουθία αυτή μπορεί εύκολα να ενταχθεί σε ροές μηχανικής μάθησης ή στατιστικης ανάλυσης.

5.2 Μελλοντικό έργο

Ως κλείσιμο της εργασίας, καταγράφονται ορισμένες κατευθύνσεις που θα μπορούσαν να λειτουργήσουν ως αφορμήσεις για την περαιτέρω επέκτασή της.

- Προσθήκη του χαρακτηριστικού της ατομικότητας στη διαχείριση του κύκλου ζωής του Apache Spark. Πιο συγκεκριμένα, αν κατά την εγκατάσταση ή την εκτέλεση των εργασιών του Apache Spark προκύψει κάποιο σφάλμα, θα πρέπει το Kubernetes να καθαρίζεται από όλα τα αντικείμενα που είχαν δημιουργηθεί μέχρι στιγμής προτού παυθεί η διοχέτευση.
- Όπως αναφέρθηκε στην υποενότητα 4.3.1, κατά την περιγραφή μιας συστοιχίας Apache Spark σε ένα αντικείμενο SparkApplication, ο χρήστης πρέπει να καθορίσει έναν στατικό αριθμό απο executor Pods. Μια ιδέα είναι να μπορεί να λειτουργήσει το

SparkApplication στη λογική ενός DeamonSet, δημιουργώντας τόσα executor Pods όσοι και οι διαθέσιμοι κόμβοι του επιπέδου δεδομένων. Η συγκεκριμένη αλλαγή είναι αρκετά δομική και απαιτεί συνεισφορά στον κώδικα των Kubeflow Pipelines.

- Ενσωμάτωση μηχανισμού εποπτείας (observability) της κατάστασης εκτέλεσης της συστοιχίας Apache Spark μέσω γραφικού περιβάλλοντος. Η εποπτεία αφορά τόσο τα μηνύματα (logs) που εκκρίνει η συστοιχία κατά την εκτέλεσή της, όσο και διαγράμματα που καταγράφουν την χρήση πόρων και την υγεία της. Οι συγκεκριμένες προσθήκες θα μπορούσαν να επιτευχθούν μέσω της στοίβας τεχνολογιών (technology stack) Elasticsearch - (File)Beat - Kibana και Prometheus - Grafana [47, 48], αντίστοιχα.

Παραρτήματα

Παράρτημα Α΄

Kubeflow Pipelines components

Α.1 Kubernetes components

```yaml
name: Apply Kubernetes resource
inputs:
  - {name: manifest, type: JsonObject}
outputs:
  - {name: manifest_output, type: JsonObject}
  - {name: name, type: String}
  - {name: kind, type: String}
  - {name: namespace, type: String}
implementation:
  container:
    image: bitnami/kubectl:1.21.5
    command:
      - bash
      - -exc
      - |
        manifest=$0
        manifest_output_path=$1
        name_output_path=$2
        kind_output_path=$3
        namespace_output_path=$4
        mkdir -p "$(dirname "$manifest_output_path")"
        mkdir -p "$(dirname "$name_output_path")"
        mkdir -p "$(dirname "$kind_output_path")"
        mkdir -p "$(dirname "$namespace_output_path")"
        kubectl apply -f "$manifest" --output=json > "$manifest_output_path"
        < "$manifest_output_path" jq '.metadata.name' --raw-output > "$name_output_path"
        < "$manifest_output_path" jq '.kind' --raw-output > "$kind_output_path"
        < "$manifest_output_path" jq '.metadata.namespace' --raw-output > "$namespace_output_path"
        cat "$manifest_output_path"
      - {inputPath: manifest}
      - {outputPath: manifest_output}
      - {outputPath: name}
      - {outputPath: kind}
      - {outputPath: namespace}
```

Σχήμα Α.1: *Apply Kubernetes resource*

```yaml
name: Delete Kubernetes resource
inputs:
  - {name: kind, type: String}
  - {name: name, type: String}
  - {name: namespace, type: String}
implementation:
  container:
    image: bitnami/kubectl:1.21.5
    command:
      - bash
      - -exc
      - |
        kind=$0
        name=$1
        namespace=$2
        kubectl delete --wait=true --namespace "$namespace" "$kind" "$name"
      - {inputValue: kind}
      - {inputValue: name}
      - {inputValue: namespace}
```

Σχήμα Α.2: *Delete Kubernetes resource*

A.2 Helm Chart components

```yaml
name: Verify SparkApplication's completion
inputs:
  - {name: name, type: String}
  - {name: kind, type: String}
  - {name: namespace, type: String}
  - {name: expected_state, type: String}
  - {name: timeout, type: int}
outputs:
  - {name: state, type: String}
implementation:
  container:
    image: bitnami/kubectl:1.21.5
    command:
    - bash
    - -exc
    - |
      name=$0
      kind=$1
      namespace=$2
      expected_state=$3
      timeout=$4
      state_output_path=$5
      mkdir -p "$(dirname "$state_output_path")"

      until [[ `cat "${state_output_path}"` = `echo "${expected_state}"` || "$timeout" < 0 ]]; do
        sleep 5
        ((timeout-=5))
        kubectl get --namespace "$namespace" "$kind" "$name" --output jsonpath='{.status.applicatio
nState.state}' > "$state_output_path"
      done

      if [[ `cat "${state_output_path}"` != `echo "${expected_state}"` ]]; then
        exit 1
      fi
    - {inputValue: name}
    - {inputValue: kind}
    - {inputValue: namespace}
    - {inputValue: expected_state}
    - {inputValue: timeout}
    - {outputPath: state}
```

Σχήμα A.3: *Verify SparkApplication's completion*

```yaml
name: Install Helm Chart
inputs:
  - {name: helm_repo_url, type: String}
  - {name: helm_repo_name, type: String}
  - {name: helm_chart_name, type: String}
  - {name: helm_chart_version, type: String}
  - {name: helm_values, type: JsonObject}
  - {name: release_name, type: String}
  - {name: release_namespace, type: String}
implementation:
  container:
    image: dtzar/helm-kubectl:3.7.1
    command:
    - bash
    - -exc
    - |
      helm_repo_url=$0
      helm_repo_name=$1
      helm_chart_name=$2
      helm_chart_version=$3
      helm_values=$4
      release_name=$5
      release_namespace=$6

      helm repo add "$helm_repo_name" "$helm_repo_url"
      helm repo update
      helm upgrade --install --namespace "$release_namespace" --create-namespace --debug --atomic --version "$helm_chart_version" --values "$helm_values" "$release_name" "$helm_chart_name"
    - {inputValue: helm_repo_url}
    - {inputValue: helm_repo_name}
    - {inputValue: helm_chart_name}
    - {inputValue: helm_chart_version}
    - {inputPath: helm_values}
    - {inputValue: release_name}
    - {inputValue: release_namespace}
```

Σχήμα Α.4: *Install or Upgrade Helm Chart*

```yaml
name: Uninstall Helm Chart
inputs:
  - {name: release_name, type: String}
  - {name: release_namespace, type: String}
implementation:
  container:
    image: dtzar/helm-kubectl:3.7.1
    command:
      - bash
      - -exc
      - |
        release_name=$0
        release_namespace=$1

        helm uninstall --wait --namespace "$release_namespace" "$release_name"
      - {inputValue: release_name}
      - {inputValue: release_namespace}
```

Σχήμα A.5: *Uninstall Helm Chart*

Βιβλιογραφία

[1] *Kunernetes - Production-Grade Container Orchestration*. https://kubernetes.io/. Ημερομηνία πρόσβασης: 06-02-2022.

[2] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune και John Wilkes. *Large-scale cluster management at Google with Borg*. *EuroSys '15: Tenth EuroSys Conference*, Bordeaux, France, 2015.

[3] *Kubeflow - The Machine Learning Toolkit for Kubernetes*. https://www.kubeflow.org/. Ημερομηνία πρόσβασης: 05-02-2022.

[4] *Kubeflow Pipelines Github repository*. https://github.com/kubeflow/pipelines. Ημερομηνία πρόσβασης: 07-02-2022.

[5] *Apache Spark - Unified engine for large-scale data analytics*. https://spark.apache.org/. Ημερομηνία πρόσβασης: 07-02-2022.

[6] *The official Google Cloud web page*. https://cloud.google.com/. Ημερομηνία πρόσβασης: 07-02-2022.

[7] *The official Microsoft Azure web page*. https://azure.microsoft.com/en-us/. Ημερομηνία πρόσβασης: 04-02-2022.

[8] *The official Docker web page*. https://www.docker.com/. Ημερομηνία πρόσβασης: 07-02-2022.

[9] *runc Github repository*. https://github.com/opencontainers/runc. Ημερομηνία πρόσβασης: 07-02-2022.

[10] *The JSON data interchange syntax*. https://www.ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf. Ημερομηνία πρόσβασης: 06-02-2022.

[11] *Dockerfile syntax reference*. https://docs.docker.com/engine/reference/builder/. Ημερομηνία πρόσβασης: 07-02-2022.

[12] *DockerHub - Container Image Library*. https://hub.docker.com/. Ημερομηνία πρόσβασης: 05-02-2022.

[13] *The official git web page*. https://git-scm.com/. Ημερομηνία πρόσβασης: 07-02-2022.

[14] *The official YAML web page*. https://yaml.org/. Ημερομηνία πρόσβασης: 05-02-2022.

[15] *The official etcd web page*. https://etcd.io/. Ημερομηνία πρόσβασης: 04-02-2022.

[16] *The official containerd web page*. https://containerd.io/. Ημερομηνία πρόσβασης: 04-02-2022.

[17] *cri-o - Lightweight Container Runtime for Kubernetes*. https://cri-o.io/. Ημερομηνία πρόσβασης: 07-02-2022.

[18] *Container ecosystem analysis*. https://www.tutorialworks.com/difference-docker-containerd-runc-crio-oci/. Ημερομηνία πρόσβασης: 05-02-2022.

[19] *The official NetApp web page*. https://www.netapp.com/. Ημερομηνία πρόσβασης: 04-02-2022.

[20] Sage Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long και Carlos Maltzahn. *Ceph: A Scalable, High-Performance Distributed File System*. *Proceedings of the 7th Conference on Operating Systems Design and Implementation*, Seattle, WA, 2006.

[21] *Elastic Cloud on Kubernetes (ECK) Github repository*. https://github.com/elastic/cloud-on-k8s. Ημερομηνία πρόσβασης: 04-02-2022.

[22] *spark-on-k8s operator Github repository*. https://github.com/GoogleCloudPlatform/spark-on-k8s-operator. Ημερομηνία πρόσβασης: 07-02-2022.

[23] *Kubespray Github repository*. https://github.com/kubernetes-sigs/kubespray. Ημερομηνία πρόσβασης: 05-02-2022.

[24] *Ansible Github repository*. https://github.com/ansible/ansible. Ημερομηνία πρόσβασης: 07-02-2022.

[25] *Helm - The package manager for Kubernetes*. https://helm.sh/. Ημερομηνία πρόσβασης: 06-02-2022.

[26] *The official Python web page*. https://www.python.org/. Ημερομηνία πρόσβασης: 07-02-2022.

[27] *The MapReduce Wikipedia web page*. https://en.wikipedia.org/wiki/MapReduce. Ημερομηνία πρόσβασης: 03-02-2022.

[28] *The official Java web page*. https://www.java.com/en/. Ημερομηνία πρόσβασης: 05-02-2022.

[29] *The Scala programming language web page*. https://www.scala-lang.org/. Ημερομηνία πρόσβασης: 05-02-2022.

[30] *The official R project web page*. https://www.r-project.org. Ημερομηνία πρόσβασης: 05-02-2022.

[31] *The official Python's pandas library web page*. https://pandas.pydata.org/. Ημερομηνία πρόσβασης: 07-02-2022.

[32] *The official Apache Hadoop YARN web page*. https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html. Ημερομηνία πρόσβασης: 02-02-2022.

[33] *The official Hadoop Distributed File System web page*. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. Ημερομηνία πρόσβασης: 05-02-2022.

[34] *Openstack - Open Source Cloud Computing Infrastructure*. https://www.openstack.org/. Ημερομηνία πρόσβασης: 07-02-2022.

[35] *The 1990's California Housing dataset*. https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html. Ημερομηνία πρόσβασης: 05-02-2022.

[36] *The official Project Calico web page*. https://www.tigera.io/project-calico/. Ημερομηνία πρόσβασης: 07-02-2022.

[37] *Ingress NGINX Controller Github repository*. https://github.com/kubernetes/ingress-nginx. Ημερομηνία πρόσβασης: 05-02-2022.

[38] *Role-Based Access Control in Kubernetes*. https://kubernetes.io/docs/reference/access-authn-authz/rbac/. Ημερομηνία πρόσβασης: 07-02-2022.

[39] *Bitnami's kubectl Docker image on DockerHub*. https://hub.docker.com/r/bitnami/kubectl. Ημερομηνία πρόσβασης: 05-02-2022.

[40] *Popular Docker image on DockerHub containing helm and kubectl*. https://hub.docker.com/r/dtzar/helm-kubectl/. Ημερομηνία πρόσβασης: 06-02-2022.

[41] *The official Python's sklearn library web page*. https://scikit-learn.org/stable/. Ημερομηνία πρόσβασης: 07-02-2022.

[42] *The official Python Docker image on DockerHub*. https://hub.docker.com/_/python. Ημερομηνία πρόσβασης: 07-02-2022.

[43] *The official Apache Spark Docker image (bundled with pyspark) on Google Cloud Platfrorm container registry*. https://console.cloud.google.com/gcr/images/spark-operator/global/spark-py. Ημερομηνία πρόσβασης: 06-02-2022.

[44] *The official Python's numpy library web page*. https://numpy.org/. Ημερομηνία πρόσβασης: 06-02-2022.

[45] *Databricks' Koalas Github repository*. https://github.com/databricks/koalas. Ημερομηνία πρόσβασης: 07-02-2022.

[46] *Pull Request on Kubeflow Pipelines repository to incorporate Helm Chart components*. https://github.com/kubeflow/pipelines/pull/7236. Ημερομηνία πρόσβασης: 07-02-2022.

[47] *The official Prometheus web page*. https://prometheus.io/. Ημερομηνία πρόσβασης: 06-02-2022.

[48] *The official Grafana web page*. https://grafana.com/. Ημερομηνία πρόσβασης: 07-02-2022.