ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών

Χρονοδρομολόγηση σε Edge Cloud Συσκευές

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βασίλειος Δ. Θεοδωρόπουλος

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2023

Εθνικο Μετσοβιο Πολυτεχνειο
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών

Χρονοδρομολόγηση σε Edge Cloud Συσκευές

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βασίλειος Δ. Θεοδωρόπουλος

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 19 Ιουλίου 2023.

.....	.....	.....
Νεκτάριος Κοζύρης Καθηγητής Ε.Μ.Π.	Γεώργιος Γκούμας Αν. Καθηγητής Ε.Μ.Π.	Διονύσιος Πνευματικάτος Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2023.

.....

Βασίλειος Δ. Θεοδωρόπουλος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Το πλήθος των δεδομένων και η ταχύτητα με την οποία παράγονται στις μέρες μας έχει αρχίσει να θέτει αρκετά προβλήματα στην δομή του Cloud, επάνω στην οποία είναι σχεδιασμένες οι περισσότερες αν όχι όλες σύγχρονες εφαρμογές. Ο χρόνος που μεσολαβεί ανάμεσα στην μεταφορά δεδομένων από και προς τους απομακρυσμένους servers είναι επαρκώς μεγάλος ωστε να αυξάνει την καθυστέρηση απόκρισης των εφαρμογών αλλά και να δίνει την δυνατότητα σε κακόβουλους χρήστες να μπορούν να τα αποσπάσουν ευκολότερα. Για τους λόγους αυτούς έχει αναπτυχθεί μία νέα μορφή Cloud, το Edge to Cloud Continuum το οποίο εκμεταλλεύεται την υπολογιστική ισχύ των συσκευών που βρίσκονται στα άκρα και παράγουν τα δεδομένα ώστε να μεταφέρει την επεξεργασία σε επίπεδο τοπικού δικτύου. Η ετερογένια των συσκευών αυτών καθώς και η ανάγκη για βέλτιστη κατανομή των διάφορων διεργασιών, συχνά με διάφορα κριτήρια, όπως η ταχύτητα, η κατανάλωση ενέργειας, η ασφάλεια ή συνδυασμός αυτών, γέννησαν την ανάγκη για ένα σύστημα που θα αυτοματοποιήσει αυτήν την διαδικασία. Αυτό το σύστημα είναι ο Kubernetes, το οποίο έχει σχεδιαστεί για την κατανομή διεργασιών σε διάφορα μηχανήματα. Το γεγονός όμως οτι ο Kubernetes εκτελεί το Scheduling βασιζόμενος σε μία greedy λογική, επιλέγει δηλαδή πάντα το πιο ισχυρό μηχάνημα, καθώς και το γεγονός οτι δεν υπάρχει κάποιος τρόπος για να επιβληθεί μια τακτική Scheduling ορισμένη από τον χρήστη, μας έδωσε το έναυσμα να δημιουργήσουμε ένα σύστημα το οποίο να δέχεται ως είσοδο ένα οποιοδήποτε πλάνο τοποθέτησης και να παράγει το κατάλληλο configuration αρχείο ώστε να επιβάλει το πλάνο στον Kubernetes.

Λέξεις Κλειδιά

Kubernetes, Scheduling, Cloud, Edge to Cloud Continuum, Argo Workflows, Διεργασίες.

Abstract

Most of the applications nowadays are deployed on the Cloud, which is challenged by the amount of data produced by countless electronic devices. Transferring those data to and from the servers requires some time, which is enough to create important latency to our applications and also gives the opportunity to malicious users to easily gain access to our data. For those reasons Edge computing is moving the data processing closer to the devices that produce the data, taking advantage of the increased computing capabilities of these devices. However electronic devices on the Edge do not have enough storing resources making the use of cloud necessary for larger projects. Edge to Cloud Continuum which combines these two structures faces the heterogeneity problem that is generated by the different characteristics of the devices that constitute the cluster. In order to automate the process of assigning each task to the best available device, given a certain criterion, an open source program, Kubernetes, is used. The fact that Kubernetes Scheduler follows a greedy algorithm and also the fact that there is no way of forcing a scheduling plan created by a different system to Kubernetes, made us create a component that takes input from such systems and produces a configuration file for the Kubernetes to execute.

Keywords

Kubernetes, Scheduling, Cloud, Edge to Cloud Continuum, Argo Workflows, Tasks.

Περιεχόμενα

Εισαγωγή

Στις μέρες μας ολοένα και περισσότερες εφαρμογές αναπτύσσονται στο cloud, προκειμένου να εκμεταλλευτούν τόσο την αμεσότητα με την οποία μπορούν οι χρήστες να έχουν πρόσβαση σε προσωπικά τους δεδομένα -όπως αρχεία, φωτογραφίες, e-mails- όσο και το γεγονός οτι δεν είναι αναγκαία η εγκατάσταση και η συντήρηση φυσικών μηχανημάτων τα οποία απαιτούν επιπλέον χώρο αλλά και χρήμα για την συντήρησή τους. Το υπολογιστικό νέφος είναι μια δομή που βασίζεται στην χρήση υπολογιστικά ισχυρών μονάδων, τους servers, οι οποίοι βρίσκονται συνήθως σε μεγάλη απόσταση από τον τελικό χρήστη. Ωστόσο με την εξέλιξη της τεχνολογίας ο όγκος των δεδομένων που παράγουν οι πολυάριθμες ηλεκτρονικές συσκευές έχουν φέρει το Cloud αντιμέτωπο με μία σειρά από προβλήματα.

Αρχικά σε πολλές περιπτώσεις χρειάζεται ένα σύστημα το οποίο να μπορεί σε πολύ μικρό χρονικό διάστημα να αναλύσει δεδομένα και να πάρει μία απόφαση. Με την μέχρι τώρα δομή θα πρέπει η πληροφορία αφού συλλεχθεί να διανύσει μία μεγάλη απόσταση μέχρι τον server, να επεξεργαστεί και να επιστρέψει στις συσκευές που θα υλοποιήσουν την απόφαση. Για να κατανοήσουμε την ανάγκη επίλυσης αυτού του προβλήματος αρκεί να σκεφτούμε την περίπτωση ενός αυτοοδηγούμενου οχήματος του οποίου οι αισθητήρες ανιχνεύουν ένα μεγάλο αντικείμενο που διακόπτει την πορεία του. Θα θέλαμε λοιπόν μία δομή που θα μηδενίζει την καθυστέρηση που μεσολαβεί από την συλλογή δεδομένων μέχρι και την λήψη απόφασης και το Cloud δεν μοιάζει σαν μία επαρκής λύση. Όπως αναφέραμε παραπάνω ο όγκος των δεδομένων έχει αυξηθεί τα τελευταία χρόνια και αναμένεται να αυξηθεί ακόμα περισσότερο. Στα δεδομένα αυτά συχνά περιέχονται ευαίσθητες πληροφορίες οι οποίες δεν θα θέλαμε να διαρεύσουν. Ωστόσο η απόσταση που διανύει η πληροφορία μέσα στο Cloud είναι επαρκώς μεγάλη ωστε να μπορεί πολύ εύκολα κάποιος κακόβουλος χρήστης να παρεμβληθεί και να αποσπάσει δεδομένα παρά την ασφάλεια που μας εξασφαλίζουν οι πάροχοι Cloud. Για τους παραπάνω λόγους κρίθηκε αναγκαίο να μεταφερθεί η επεξεργασία και η διάδοση της πληροφορίας πιο κοντά στα άκρα, στις συσκευές δηλαδή που την συλλέγουν. Το λεγόμενο Edge to Cloud Continuum, εκμεταλλεύεται τις υπολογιστικές δυνατότητες των edge συσκευών μετατοπίζοντας τον φόρτο εργασίας από τους απομακρυσμένους servers σε επίπεδο τοπικού δικτύου. Έτσι οι συσκευές μπορούν να επεξεργαστούν και να ανταλλάσσουν μεταξύ τους δεδομένα, χωρίς να είναι απαραίτητη η σύνδεση στο διαδίκτυο και να χρησιμοποιούν το cloud μόνο σε περιπτώσεις που είτε χρειάζονται

επιπλέον δεδομένα, είτε που οι υπολογιστική τους ισχύς δεν επαρκεί. Την μετάδοση της πληροφορίας από και πρός το cloud σε αυτήν την περίπτωση την αναλαμβάνουν συσκευές που ονομάζονται edge devices.

Η ύπαρξη δεκάδων διαφορετικών συσκευών σε ένα τέτοιο σύστημα συχνά συνεπάγεται με διαφορετικές δυνατότητες ανα συσκευή. Σε τέτοια ετερογενή συστήματα έχουμε πολλές επιλογές όσον αφορά την ανάθεση της εκτέλεσης μιας διεργασίας σε ένα μηχάνημα και επιθυμούμε να επιλεγεί η βέλτιστη από αυτές βάσει κάποιου συγκεκριμένου κριτηρίου. Την λύση σε αυτό το πρόβλημα την δίνει ο Kubernetes, ένα σύστημα ανοιχτού κώδικα, ενορχήστρωσης containers, το οποίο μπορεί να αναθέτει την εκτέλεση διεργασιών σε μηχανήματα. Ο Kubernetes αποτελείται από διάφορα δομικά στοιχεία με τον Scheduler να είναι το υπεύθυνο στοιχείο για την κατανομή των εργασιών. Ο Scheduler αρχικά φιλτράρει τα διαθέσιμα μηχανήματα ωστε να απορίψει αυτά που δεν έχουν τους πόρους να εκτελέσουν την διεργασία και στη συνέχεια την αναθέτει σε κάποιο από τα υπόλοιπα βασιζόμενος σε μία συνάρτηση score. Στην default λειτουργία του ο Kubernetes επιλέγει το ισχυρότερο δυνατό μηχάνημα, παίρνει δηλαδή αποφάσεις με βάση κάποιον greedy αλγόριθμο. Ενώ κάτι τέτοιο είναι επιθυμητό όταν μας ενδιαφέρει η ταχύτητα εκτέλεσης συχνά θέλουμε η κατανομή των εργασιών να γίνεται και με άλλα κριτήρια, όπως η κατανάλωση ενέργειας ή η ασφάλεια.

Ο Scheduler του Kubernetes είναι σχεδιασμένος προκειμένου να μπορούμε να τροποποιήσουμε οποιοδήποτε στάδιο του Scheduling επιθυμούμε. Στην περίπτωση μας, στην οποία διαθέτουμε ένα σύστημα το οποίο με βάση κάποιο κριτήριο (ταχύτητα, κατανάλωση, ασφάλεια κ.α) προσπαθεί να βρει την βέλτιστη τοποθέτηση διεργασιών ενός γράφου διεργασιών σε μηχανήματα, κάτι τέτοιο είναι αδύνατο καθώς γνωρίζουμε μόνο την είσοδο και την έξοδο του συστήματος και τίποτα σχετικά με τον αλγόριθμο που χρησιμοποιεί. Γι' αυτόν τον λόγο σχεδιάσαμε ένα σύστημα το οποίο λαμβάνει είσοδο από ένα σύστημα τοποθέτησης διεργασιών και παράγει ένα configuration αρχείο το οποίο στη συνέχεια καταχωρεί στον Kubernetes για να εκτελέσει τις διεργασίες στα επιθυμητά μηχανήματα, χωρίς να λαμβάνει υπόψιν καθόλου την λειτουργία και την δομή του εκάστοτε planner.

Υπόβαθρο

2.1 Cloud

Το Υπολογιστικό Νέφος (Cloud Computing) σύφωνα με το Εθνικό Ινστιτούτο Προτύπων και Τεχνολογίας (N.I.S.T) είναι ένα μοντέλο το οποίο παρέχει μέσω σύνδεσης με το internet πρόσβαση σε διαμορφώσιμους υπολογιστικούς πόρους (δίκτυα, servers, χώρους αποθήκευσης και υπηρεσίες) στο οποίο οι χρήστες έχουν άμεση πρόσβαση με ελάχιστη επίβλεψη και αλληλεπίδραση με παρόχους υπηρεσιών[1]. Η λογική του Cloud είναι δομημένη επάνω στην χρήση μεγάλων data centers και απομακρυσμένων servers για την αποθήκευση και επεξεργασία δεδομένων. Με αυτόν τον τρόπο πολλές εργασίες μπορούν να εκτελούνται ταυτόχρονα χωρίς στην πραγματικότητα να ασχολούνται οι προγραμματιστές με την επάρκεια του εξοπλισμού αφού πλέον μπορούν να ελέγχουν και να μεταβάλλουν όποτε είναι αναγκαίο τους εικονικούς πόρους που χρησιμοποιούν. Ένα από τα μεγάλα πλεονεκτήματα που μας έχει δώσει το cloud είναι οτι μπορούμε πλέον να αναπτύξουμε εφαρμογές χωρίς να χρειάζεται να προβλέψουμε χώρο για την εγκατάσταση μηχανημάτων τα οποία χρησιμοποιούνται είτε ως μονάδες αποθήκευσης είτε ως υπολογιστικές μονάδες, το οποίο γλιτώνει τόσο χώρο όσο και χρήματα στις επιχειρήσεις. Για την ακρίβεια στις περισσότερες σύγχρονες εφαρμογές δεν γνωρίζουμε που βρίσκονται οι φυσικοί servers με τους οποίους αυτές αλληλεπιδρούν. Ένα άλλο μεγάλο πλεονέκτημα που μας έχει προσφέρει το cloud είναι πως μπορούμε να έχουμε πρόσβαση σε δεδομένα ανα πάσα στιγμή με μεγάλη ευκολία και ταχύτητα. Για παράδειγμα είναι πολύ εύκολο να έχουμε πρόσβαση σε αρχεία μας που έχουν αποθηκευτεί σε κάποιο drive καθώς επίσης και να έχουμε πρόσβαση στα e-mails μας μέσω οποιασδήποτε συσκευής είναι συνδεδεμένη στο internet ανεξάρτητα από το που βρισκόμαστε.

Οι διαφορετικοί τύποι cloud που μπορούμε να συναντήσουμε είναι οι εξής:

- **Private Cloud**: Είναι σχεδιασμένο για να χρησιμοποιείται μόνο από ένα σύνολο ατόμων, για παράδειγμα τα μέλη μιας επιχείρησης. Οι κυριότεροι λόγοι για τους οποίους μπορεί κάποιος να το επιλέξει είναι αφ'ενός η ύπαρξη επαρκών πόρων προκειμένου να καλυφθούν οι ανάγκες της εργασίας αφ' ετέρου και πιο σημαντικός είναι η ανάγκη για για προστασία των δεδομένων η οποία δεν μπορεί να εξασφαλιστεί σε κάποιο δημόσιο δίκτυο.

- **Community Cloud**: Χρησιμοποιείται από κοινού από συγκεκριμένους οργανισμούς οι οποίοι έχουν τις ίδιες ανάγκες σε πόρους.
- **Public Cloud**: Είναι η πλέον πιο διαδεδομένη μορφή cloud στην έχουν πρόσβαση όλοι οι χρήστες του διαδικτύου. Ο πάροχος υπηρεσιών cloud επιβάλλει τους όρους τους οποίους κάθε χρήστης πρέπει να ακολουθεί εαν θέλει να έχει πρόσβαση στις υπηρεσίες του, είτε αυτοί αφορούν κόστος είτε την παροχή των δεδομένων για βελτιστοποίηση των υπηρεσιών.
- **Hybrid Cloud**: Είναι ο συνδυασμός τουλάχιστον δύο διαφορετικών τύπων cloud.

Όσον αφορά τις υπηρεσίες που παρέχει το cloud, αυτές μπορούν να χωριστούν σε τρείς κατηγορίες, το Λογισμικό ως Υπηρεσία (Software as a Service), την Πλατφόρμα ως υπηρεσία (Platform as a Service) και την Υποδομή ως Υπηρεσία (Infrastructure as a Service).

Software as a Service (SaaS)

Αυτή η υπηρεσία δίνει στον χρήστη την δυνατότητα να χρησιμοποιεί εφαρμογές του παρόχου στο cloud από οποιαδήποτε συσκευή χωρίς ωστόσο να έχει την δυνατότητα να διαχειριστεί τα δομικά στοιχεία του cloud όπως το δίκτυο, τους servers και τους χώρους αποθήκευσης δεδομένων.

Platform as a Service (PaaS)

Οι χρήστες έχουν την δυνατότητα να αναπτύξουν εφαρμογές στο cloud χωρίς να μπορούν και σε αυτήν την περίπτωση να διαχειριστούν τα δομικά στοιχεία του cloud. Οι χρήστες μπορούν να χρησιμοποιούν γλώσσες προγραμματισμού και άλλα προγραμματιστικά εργαλεία προκειμένου να αναπτύσσουν τις εφαρμογές τους.

Infrastructure as a Service (IaaS)

Σε αυτήν την περίπτωση οι χρήστες, σε αντίθεση με τις δύο προηγούμενες υπηρεσίες, έχουν την δυνατότητα να διαχειρίζονται τους υπολογιστικούς πόρους (επεξεργαστές, μονάδες μνήμης και δίκτυα) που χρειάζονται ωστε να αναπτύσσουν τις εφαρμογές τους. Έτσι ο χρήστης μπορεί να δηλώσει το πόση μνήμη ή επεξεργαστική ισχύ θα χρησιμοποιεί η εφαρμογή του το οποίο μπορεί να αλλάζει κατ' απαίτηση.

Η δομή και οι δυνατότητες που μας παρέχει το cloud έρχονται βέβαια μαζί με αρκετά θέματα ασφαλείας. Πλέον που τα δεδομένα έχουν μεταφερθεί εκτός του τοπικού δικτύου στο Internet, όπου η ροή τους γίνεται ανάμεσα σε διάφορους servers ανα τον κόσμο, και που το πλήθος τους έχει πολλαπλασιαστεί δεδομένου του τεράστιου αριθμού των χρηστών του καθιστούν πιο έυκολο ένα λάθος είτε μια επίθεση να προκαλέσει μεγάλα προβλήματα διαρροής προσωπικών δεδομένων. Επίσης οι παραπάνω λόγοι

έλκουν περισσότερους κακόβουλους χρήστες να προσπαθήσουν να αποκτήσουν πρόσβαση στα δεδομένα αυτά. Οι πάροχοι cloud είναι υπεύθυνοι πλέον να προσφέρουν και ασφάλεια στους χρήστες τους. Ωστόσο το γεγονός οτι οι πάροχοι υπηρεσιών cloud έχουν μεγάλη εξειδίκευση στο κομμάτι της ασφάλειας και έχουν προβλέψει να κάνουν τις απαραίτητες ενέργειες ωστε τα δεδομένα μας να είναι ασφαλή δεν αλλάζει το οτι αυτά βρίσκονται εκτός του προσωπικού μας υπολογιστή κάτι που τα κάνει ευκολότερα προσβάσιμα σε κακόβουλους χρήστες.

2.2 Edge to Cloud Continuum

Ως έννοια, το Διαδίκτυο των πραγμάτων γνωστό και ως Internet of Things (IoT) έχει κάνει τα τελευταία χρόνια πολύ δυναμική είσοδο στην καθημερινότητά μας. Όπως προδίδει και ο όρος, το Internet of Things ουσιαστικά ενώνει διάφορες ηλεκτρονικές συσκευές με την χρήση του διαδικτύου. Οι συσκευές αυτές, όπως για παράδειγμα αισθητήρες, λάμπες, συστήματα ασφαλείας οι οποίες συνδέονται μεταξύ τους μέσω ενός δικτύου προκειμένου να επεξεργάζονται και ανταλλάσσουν δεδομένα.

Το IoT βρίσκεται ακόμα σε πρώιμα στάδια και καλείται να αντιμετωπίσει μια σειρά από ζητήματα. Αρχικά θα πρέπει να εξασφαλίσει πως η ταχύτητα με την οποία οι συσκευές επεξεργάζονται και ανταλλάσσουν δεδομένα θα έχει την μικρότερη δυνατή καθυστέρηση καθώς και οτι θα είναι διαθέσιμες ανα πάσα στιγμή ανεξάρτητα από την φυσική θέση του χρήστη. Για να αντιληφθούμε αυτό το πρόβλημα αρκει να σκεφτούμε ένα αυτοοδηγούμενο αυτοκίνητο του οποίου οι αισθητήρες αντιλαμβάνονται την ύπαρξη ενός μεγάλου αντικειμένου να διακόπτει την πορεία του θέτοντας σε κίνδυνο τους επιβάτες και θα πρέπει να στείλει σήμα στο αυτοκίνητο είτε να σταματήσει είτε να αλλάξει πορεία. Τόσο η αναγνώριση του αντικειμένου, όσο και η αποστολή του σήματος θα πρέπει να γίνουν σε κλάσματα του δευτερολέπτου. Επιπλέον το γεγονός οτι τα δεδομένα που συγκρατούν αυτές οι συσκευές μπορεί να είναι ευαίσθητα καθώς και το οτι έχουν πολύ μικρή υπολογιστική ισχύ προκειμένου να έχουν περίπλοκα συστήματα ασφαλείας τα καθιστούν εύκολο στόχο για κακόβουλους χρήστες. Επίσης η επικοινωνία μεταξύ των συσκευών γίνεται με ασύρματο τρόπο προσθέτοντας έναν ακόμα κίνδυνο εαν τα δεδομένα χρειάζεται να επεξεργαστούν σε ένα απομακρυσμένο δίκτυο. Τέλος το IoT καλείται να αντιμετωπίσει και το θέμα της ιδιωτικότητας. Δεδομένου οτι ο τρόπος με τον οποίον θα συλλέγονται τα προσωπικά δεδομένα θα διαφέρει από αυτόν που γνωρίζουμε μέχρι σήμερα αυξάνοντας τις περιστάσεις στις οποίες συλλέγονται, θα έχει ως αποτέλεσμα να δυσκολέψει και ο έλεγχος που θα έχουν οι χρήστες όσον αφορά την παροχή τους. Έτσι κρίνεται αναγκαία η παροχή ασφάλειας της ιδιωτικότητας καθώς οι χρήστες θα πρέπει να γνωρίζουν ποια δεδομένα τους έχουν συλλεχθεί, από ποιόν, που διανείμονται και που αποσκοπεί η επεξεργασία τους.

Μετά και την ανάλυση της τεχνολογίας του cloud, θα περίμενε κανείς οτι το IoT θα καλυπτόταν εξ'ολοκλήρου από τις ευκολίες που αυτό προσφέρει, ωστόσο κάτι τέτοιο δεν ισχύει. Αν και οι πόροι που διαθέτει το cloud είναι πολύ ισχυροί

σε υπολογιστικά ζητήματα και μπορούν να επεξεργαστούν ταχύτατα τα δεδομένα που τους παρέχει για παράδειγμα ένας αισθητήρας, ο χρόνος που η πληροφορία κάνει για να μεταφερθεί από την συσκευή προς τον server και πίσω είναι αρκετά μεγάλος σε σχέση με αυτόν που θα επιθυμούσαμε. Επίσης ο όγκος των δεδομένων που μεταφέρει μία συσκευή είναι αρκετά μεγάλος και όπως προαναφέραμε σε πολλές περιπτώσεις περιέχει προσωπικές πληροφορίες. Αν αναλογιστούμε λοιπόν οτι αυτά τα δεδομένα μεταφέρονται για μεγάλο χρονικό διάστημα μέσα στο cloud διευκολύνουμε τους κακόβουλους χρήστες να αποκτήσουν πρόσβαση σε αυτά. Αυτό το πρόβλημα θα πολλαπλασιαστεί εαν σκεφτούμε οτι δεν είναι μόνο μια η συσκευή η οποία μεταφέρει δεδομένα αλλά εκατομμύριες και αναμένεται να πολλαπλασιαστούν τα επόμενα χρόνια.

Για να λυθούν τα παραπάνω προβλήματα αντιλαμβανόμαστε οτι θα πρέπει η επεξεργασία των δεδομένων να μεταφερθεί πιο κοντά στις συσκευές, κάτι το οποίο επιτυγχάνεται με το Edge Cloud Computing. Με αυτήν την τεχνολογία η επεξεργασία και η αποθήκευση των δεδομένων πραγματοποιουνται πιο κοντά στις συσκευές που τα συλλέγουν, σε επίπεδο τοπικού δικτύου. Το γεγονός αυτό έχει ως αποτέλεσμα να κερδίζουμε σε ταχυτητα, να αποφεύγουμε τις καθυστερήσεις στην διάδοση της πληροφορίας, αφού πλέον τα δεδομένα δεν χρειαζεται να μεταδοθούν σε κάποιον απομακρυσμένο server και να μειώσουμε την επίδραση που θα έχουν στην απόδοση του δικτύου οι χιλιάδες συσκευές που θα προσπαθούσαν να αποκτήσουν πρόσβαση στους πόρους ενός απομακρυσμένου μηχανήματος. Επιπλέον όσον αφορά τα θέματα της ασφάλειας και της ιδιωτικότητας το Edge Cloud Computing περιορίζει σε πολύ μεγάλο βαθμό τους κινδύνους που έρχονται μαζί με την χρήση του Cloud Computing όπως τους έχουμε περιγράψει παραπάνω καθώς η πληροφορία παραμένει εντός του τοπικού δικτύου, δυσκολεύοντας κάποιον κακόβουλο χρήστη να αποκτήσει πρόσβαση. Είναι σαφές οτι πλέον η χρήση του cloud από συσκευές του IoT όπως το γνωρίζουμε μέχρι τώρα θα περιοριστεί σε πολύ μεγάλο βαθμό, καθώς με το Edge Cloud Computing δεν είναι αναγκαία η χρήση του internet για μετάδοση των δεδομένων και η συχνότητα με την οποία θα το χρησιμοποιούν οι συσκευές αυτές θα είναι πολύ μικρότερη, ρίχνοντας αισθητά τα κόστη που χρειάζονται για την εξασφάλιση και την συντήρηση μηχανημάτων.

Αν και η χρήση του Cloud Computing μοιάζει να περιορίζεται το γεγονός οτι οι συσκευές που χρησιμοποιούνται από το IoT δεν διαθέτουν ισχυρή υπολογιστική ισχύ ούτε μεγάλες αποθηκευτικές μονάδες καθιστούν αναγκαία την χρήση πόρων που αυτό παρέχει. Συνεπώς θα πρέπει να επιτευχθεί επικοινωνία ανάμεσα στις συσκευές που χρησιμοποιούν το τοπικό δίκτυο οι οποίες χρησιμοποιούν πρωτόκολλα όπως το Bluetooth, το NFC και το Wi-Fi και το cloud το οποίο χρησιμοποιεί διαφορετικά πρωτόκολλα όπως HTTP και MQTT. Χρειάζονται λοιπόν ορισμένες συσκευές, τα λεγόμενα edge devices, οι οποίες θα μεταφράζουν τα πρωτόκολλα και θα κάνουν την διάδοση της πληροφορίας εφικτή τόσο εντός του τοπικού δικτύου, όσο και εντός του cloud. Παραδείγματα τέτοιων συσκευών μπορεί να είναι κινητά τηλέφωνα, ενσωματωμένοι υπολογιστές και γενικότερα συσκευές οι οποίες έχουν υπολογιστικές δυνατότητες. Οι αισθητήρες και οι λάμπες που αναφέραμε προηγουμένως για παράδειγμα ενώ αποτελούν μέρος του IoT δεν μπορούν να θεωρηθούν edge devices.

2.3 Kubernetes

Λόγω της αυξημένης ανάπτυξης των cloud εφαρμογών έχει υπερισχύσει ο σχεδιασμός και η υλοποίηση τους σε containers. Με τα containers μπορούμε να "πακετάρουμε" τον κώδικα της εφαρμογής μαζί με τις εξαρτήσεις και τις απαραίτητες βιβλιοθήκες, κάνοντας έτσι την διαδικασία ανάπτυξης εφαρμογών ανεξάρτητη από τα μηχανήματα τα οποία μπορούν να την εκτελέσουν σε αντίθεση με τα Virtual Machines τα οποία χρησιμοποιόντουσαν στο παρελθόν. Ένα άλλο πλεονέκτημα έναντι των VMs είναι οτι τα δεύτερα απαιτούν την εγκατάσταση Λειτουργικού συστήματος σε καθ' ένα από αυτά, κάνοντας τα πιο αργά και πιο βαριά από τα containers τα οποία χρησιμοποιούν τους πόρους που χρειάζονται από το λειτουργικό του μηχανήματος που τα φιλοξενεί.

Ένα από τα βασικά ζητούμενα μιας εφαρμογής σε επίπεδο παραγωγής είναι να μηδενιστεί ο χρόνος στον οποίο δεν λειτουργεί (downtime). Για να επιτευχθει αυτό θα θέλαμε με το που διακοπεί η λειτουργία ενός container αυτόματα να ξεκινάει η λειτουργία κάποιου άλλου και με αυτόν τον τρόπο ο χρήστης να μην καταλαβαίνει οτι υπήρξε κάποια διακοπή. Αυτή την λειτουργία εκτελεί ο Kubernetes, ένα σύστημα το οποίο είναι σχεδιασμένο να διαχειρίζεται containers. [9] Οι κυριότερες δυνατότητες του είναι:

- **Deployment**: Η ανάπτυξη και κατανομή containers σε συγκεκριμένους hosts
- **Rollout**: Μπορούμε να αυτοματοποιήσουμε τον Kubernetes να δημιουργεί και να διαγράφει containers
- **Automatic bin packing**: Παρέχοντας του ένα cluster από κομβους-μηχανήματα στους οποίους θέλουμε να τρέξουν ορισμένες διαδικασίες, μπορούμε να ορίσουμε το πλήθος των πόρων (CPU, RAM) που θέλουμε κάθε container να χρησιμοποιήσει
- **Service discovery**: Χρησιμοποιώντας το DNS ή την IP του container το κάνει expose στο internet
- **Load balancing**: Κατανείμει την κίνηση του δικτύου όταν αντιληφθεί οτι είναι υψιλή προκειμένου να επιτευχθεί σταθερότητα στο σύστημα
- **Self-healing**: Προκειμένου να πετύχει το χαμηλό downtime, επανεκκινεί containers τα οποία έχουν αποτύχει καθώς επίσης σταματάει containers τα οποία δεν ανταποκρίνονται σε ελέγχους τους οποίους έχει ορίσει ο χρήστης

Η λειτουργία του Kubernetes γίνεται σε clusters, τα οποία αποτελούνται από έναν συνήθως ή παραπάνω master nodes και από τουλάχιστον έναν worker node. Οι worker nodes είναι τα μηχανήματα στα οποία τρέχουν τα containers ενώ οι master κόμβοι (control plane) είναι υπεύθυνοι για όλες τις διαχειριστικές λειτουργίες του cluster. Μέσω αυτών παίρνουμε πληροφορίες για τα μηχανήματα τα οποία απαρτίζουν το cluster, πόσα από αυτά είναι διαθέσιμα ανα πάσα στιγμή καθώς επίσης είναι υπεύθυνοι για να ορίσουν που θα τρέξει κάθε container. Ένα control plane αποτελείται από τα εξής στοιχεία[11]

- **kube-apiserver**: Είναι υπεύθυνο για την έκθεση του Kubernetes API, το οποίο αποτελεί το front end του control plane

- **etcd**: Χρησιμοποιείται για την αποθήκευση δεδομένων του cluster
- **kube-scheduler**: Είναι υπεύθυνος για την κατανομή των νέων containers σε διαθέσιμα μηχανήματα
- **kube-controller-manager**: Είναι υπεύθυνο για την λειτουργεία του controller, ο οποίος επιβλέπει την κατάσταση του cluster και κάνει όλες τις απαραίτητες ενέργειες προκειμένου να πλησιάσει στην επιθυμητή κατάσταση [12]
- **cloud-controller-manager**: Ενώνει το cluster με το API του παρόχου υπηρεσιών cloud και με αυτόν τον τρόπο επιτυγχάνεται μια διαδικασία ελέγχου συμβατή με το cloud

Στην παρούσα διπλωματική εργασία θα δώσουμε παραπάνω έμφαση στον Scheduler του Kubernetes, καθώς απώτερος σκοπός μας είναι να ερευνησουμε το πως μπορούμε να επιβάλουμε στο σύστημα έναν δικό μας τρόπο κατανομής των εργασιών. Η λειτουργία του scheduler είναι να επιβλέπει ποια Pods δεν έχουν ανατεθεί σε κάποιο μηχάνημα-κόμβο και στην συνέχεια να βρίσκει τον ιδανικότερο κόμβο στον οποίο θα τρέξει η διεργασία.

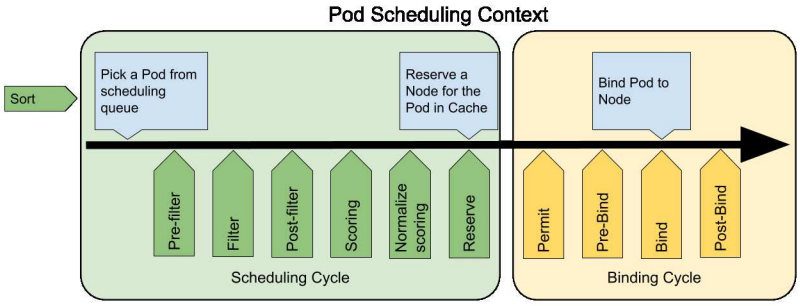
Με τον όρο Pods αναφερόμαστε στην μικρότερη δυνατή υπολογιστική μονάδα που μπορούμε να δημιουργήσουμε και να διαχειριστούμε με τον Kubernetes και στην πράξη αποτελείται από ένα ή περισσότερα containers τα οποία μοιράζονται μνήμη και ρυθμίσεις δικτύου. Με τα Pods μπορούμε να ορίσουμε τον κώδικα που θα χρησιμοποιηθεί προκειμένου να δημιουργηθεί το container, γνωστό και ως image, το πλήθος των πόρων (CPU, μνήμη) που θέλουμε να χρησιμοποιήσει το container από τους διαθέσιμους πόρους του συστήματος καθώς επίσης να ορίσουμε και σε ποιό μηχάνημα θέλουμε να τρέξει το container.

Τα βήματα τα οποία συνθέτουν την διαδικασία του Scheduling [13], όπως φαίνονται και στην Εικόνα 2.1 [1] είναι τα εξής:

- **queueSort**: Συνάρτηση που ταξινομεί τα Pods που βρίσκονται σε κατάσταση αναμονής
- **preFilter**: Χρησιμοποιείται για προεπεξεργασία των Pods
- **filter**: Είναι η συνάρτηση που αποφασίζει ποιοί κόμβοι μπορούν να τρέξουν συγκεκριμένα Pods
- **postFilter**: Καλείται όταν δεν υπάρχει κάποιος κόμβος που μπορεί να εκτελέσει κάποια διεργασία
- **preScore**: Χρησιμοποιείται για προεπεξεργασία των Pods
- **score**: Είναι η συνάρτηση που βαθμολογεί την ικανότητα ενός κόμβου να τρέξει το Pod που βρίσκεται σε αναμονή και έχει περάσει το στάδιο του filtering
- **reserve**: Ενημερώνει τα υπόλοιπα plugins σε περίπτωση που έχουν δεσμευτεί πόροι για κάποιο Pod
- **permit**: Αποτρέπουν ή καθυστερούν την εκτέλεση ενός Pod από έναν κόμβο
- **preBind**: Χρησιμοποιείται για προεπεξεργασία των Pods πριν αυτά δωθούν σε κάποιον κόμβο προς εκτέλεση

[1] https://dzone.com/articles/beyond-kube-scheduler-a-need-for-a-k8s-cluster-bal

- **bind**: Είναι υπεύθυνο για να αντιστοιχηθεί το Pod με τον κόμβο που θα το τρέξει
- **postBind**: Χρησιμοποιείται για να παρέχει πληροφορίες αφού ένα Pod έχει δεθεί με έναν κόμβο
- **multiPoint**: Χρησιμοποιείται για να ενεργοποιεί και να απενεργοποιεί plugins ανάλογα με το τι επιθυμεί ο χρήστης



Εικόνα. 2.1: Η διαδικασία του Scheduling

Αρχικά αφού ταξινομηθούν τα προς εκτέλεση Pods βάσει της διαδοχής εκτέλεσης κάθε ένα θα περάσει από την διαδικασία του filtering προκειμένου να απορριφθούν τα μηχανήματα τα οποία δεν έχουν τους απαραίτητους πόρους για να το εκτελέσουν. Εαν μετά από αυτό το βήμα κανένα μηχάνημα δεν έχει κριθεί κατάλληλο για να εκτελέσει την διεργασία, τότε το Pod παραμένει σε κατάσταση αναμονής. Στη συνέχεια όσοι κόμβοι έχουν κριθεί κατάλληλοι να εκτελέσουν το Pod βαθμολογούνται από την συνάρτηση Score, η οποία στην default λειτουργία του Kubernetes βαθμολογεί με Greedy λογική και εν συνεχεία δεσμεύεται ο κόμβος με το υψηλότερο score για να εκτελέσει το Pod. Μετά από αυτό το βήμα τελειώνει ο κύκλος του Scheduling και ακολουθεί το δέσιμο του Pod με τον κόμβο που έχει επιλεγεί για να το εκτελέσει.

Να τονιστεί σε αυτό το σημείο οτι εαν κάποιος χρήστης επιθυμεί να αλλάξει κάποιο από τα παραπάνω βήματα προκειμένου να μην ακολουθούν την default λειτουργία, ο Scheduler είναι φτιαγμένος με τέτοιον τρόπο που να μπορεί να τα αντικαταστήσει ανάλογα με τις ανάγκες του.

Σχεδίαση Συστήματος

Το πρόβλημα το οποίο καλούμαστε να λύσουμε είναι το πώς μπορούμε να επιβάλλουμε στον Kubernetes να εκτελέσει μια σειρά από διεργασίες σε ένα σύνολο μηχανημάτων παρακάμπτοντας τον default Scheduler και χρησιμοποιώντας στην θέση του ένα σύστημα που λαμβάνει αποφάσεις ως προς την τοποθέτηση βασισμένο σε διάφορα κριτήρια. Όπως είδαμε και παραπάνω (Βλ. 2.3) ένας τρόπος για να ρυθμίσουμε τον Scheduler του Kubernetes να λειτουργεί σύμφωνα με τις δικές μας ανάγκες είναι να γράψουμε κατάλληλες συναρτήσεις-plugins για όλα τα στάδια της διαδικασίας του scheduling που μας ενδιαφέρει. Ωστόσο αυτή η τεχνική έχει ορισμένα προβλήματα.

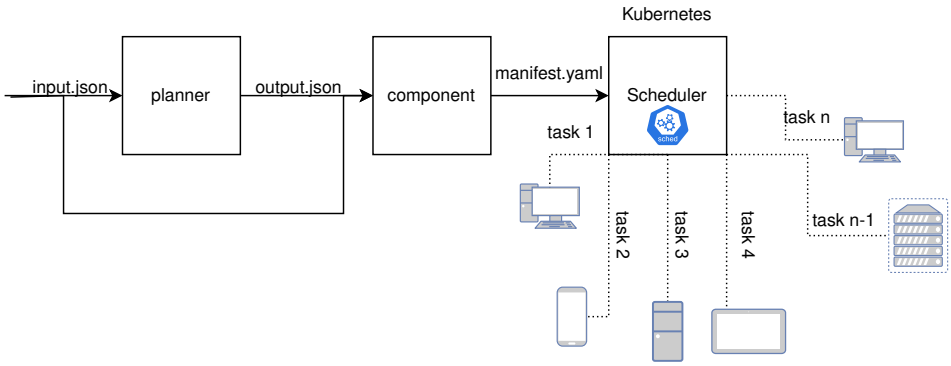
Αρχικά ένα τέτοιο εγχείρημα περιέχει μια μεγάλη πολυπλοκότητα καθώς θα πρέπει να γραφτούν συναρτήσεις σύμφωνα με τα πρότυπα στα οποία είναι δομημένος ο Kubernetes. Συνεπώς θα πρέπει να κατανοήσουμε σε επίπεδο κώδικα πως λειτουργεί ένα τόσο περίπλοκο σύστημα πριν ξεκινήσουμε να αναπτύσουμε τα δικά μας plugins. Στην συνέχεια, όταν εξετάζουμε την απόδοση ενός συστήματος όπως ο planner συνήθως θέλουμε να το συγκρίνουμε με την απόδοση άλλων συστημάτων τα οποία εκτελούν τον ίδιο σκοπό. Είναι σαφές οτι για να γίνει κάτι τέτοιο ο αριθμός των plugins που θα πρέπει να γραφτεί θα είναι ανάλογος των συστημάτων τα οποία εξετάζουμε. Τέλος όπως και στην περίπτωσή μας που διαθέτουμε ένα έτοιμο σύστημα πρόβλεψης και τοποθέτησης διεργασιών δεν γίνεται να μεταφέρουμε όλη τη λογική στην οποία έχει δομηθεί κάποιο τέτοιο σύστημα στην λογική του Kubernetes καθώς κάτι τέτοιο απαιτεί πολλή δουλειά και γνώση του αλγορίθμου τον οποίο χρησιμοποιεί σε κάθε στάδιο.

Τα παραπάνω προβλήματα μας οδηγούν στο συμπέρασμα οτι θα πρέπει να φτιαχτεί ένα σύστημα το οποίο θα μπορεί να λαμβάνει ως είσοδο τα αποτελέσματα των εκάστοτε συστημάτων πρόβλεψης τοποθέτησης διεργασιών, να είναι ανεξάρτητο από την λειτουργία τους και να δίνει στον Kubernetes ένα πλάνο σύμφωνα με το οποίο θα τρέξει τις διεργασίες.

3.1 Αρχιτεκτονική

Όπως μπορούμε να δούμε και στην Εικόνα 3.1 το σύστημά μας αποτελείται από τρία μέρη. Έναν planner, ο οποίος είναι αυτός ο οποίος υπολογίζει την βέλτιστη

τοποθέτηση των διεργασιών σε μηχανήματα, το component το οποίο σχεδιάσαμε και υλοποιήσαμε ωστε να δημιουργεί το Workflow το οποίο θα τρέξει ο Kubernetes σύμφωνα με την τοποθέτηση που έχει ορίσει ο planner και τέλος τον Kubernetes ο οποίος είναι υπεύθυνος να κατανείμει τις προς εκτέλεση διεργασίες στα αντίστοιχα μηχανήματα. Καθώς έχουμε αναπτύξει σε βάθος την λειτουργία του Kubernetes, παρακάτω θα περιγράψουμε πιο αναλυτικά τις λειτουργίες του planner και του component.



Εικόνα. 3.1: Περιγραφή της τοπολογίας

3.1.1 Planner

Ο planner είναι ένα οποιοδήποτε σύστημα το οποίο μπορεί να παράγει μία τοποθέτηση ενός γράφου διεργασιών σε ένα σύνολο από διαθέσιμα μηχανήματα. Ο planner τον οποίο διαθέτουμε και αποτέλεσε αφορμή για την δημιουργία του συστήματός μας δέχεται ώς είσοδο ένα json αρχείο, θα αναφερόμαστε σε αυτό ως input.json για ευκολία, το οποίο περιγράφει τον γράφο διεργασιών, την τοπολογία των μηχανημάτων καθώς και την μετρική την οποία έλαβε υπόψιν στους υπολογισμούς του και παράγει ως έξοδο ένα αρχείο, output.json, το οποίο κατά κύριο λόγο περιέχει την τοποθέτηση των εργασιών.

Είσοδος

Τα πεδία του αρχείου που δέχεται ως είσοδο ο planner είναι τα εξής:

- **operatorGraph**: Ο γράφος με τις διεργασίες που θέλουμε να εκτελέσουμε
- **availNodes**: Περιέχει πληροφορία για τους κόμβους-μηχανήματα
- **networkDelays**: Παρέχει πληροφορίες για την κατάσταση του δικτύου μεταξύ των κόμβων
- **optimObjectives**: Δίνει την μετρική την οποία έχει λάβει υπόψιν ο planner προκειμένου να κάνει κατανομή των εργασιών

Στον σχεδιασμό μας χρησιμοποιήσαμε κατά κύριο λόγο το πεδίο operatorGraph καθώς περιγράφει όλη την ροή των εργασιών και το optimObjectives για την εξαγωγή συμπερασμάτων στις δοκιμές μας. Αν και το πεδίο availNodes θα μπορούσε να μας φανεί χρήσιμο, ο Kubernetes έχει δικούς του μηχανισμούς ανίχνευσης των διαθέσιμων μηχανημάτων. Επίσης πολλές φορές το δίκτυο παίζει πολύ σημαντικό ρόλο κατά την απόφαση του που θα τρέξει κάποια διεργασία. Ωστόσο η παραγωγή του configuration αρχείου που καταχωρούμε στον Kubernetes δεν μπορεί να γίνει με δυναμικό τρόπο, συνεπώς το πεδίο networkDelays μπορεί να φανεί πολύ χρήσιμο κατά την εξαγωγή του πλάνου εκτέλεσης του γράφου διεργασιών αλλά στην περίπτωσή μας δεν το εκμεταλλευόμαστε με κάποιον τρόπο.

Όπως προείπαμε το πιο σημαντικό από τα πεδία του αρχείου εισόδου είναι το operatorGraph, το οποίο για κάθε κόμβο του γράφου διεργασιών περιέχει σημαντικές πληροφορίες για την κατασκευή του configuration file, όπως το πεδίο children που μας παρέχει όλα τα "παιδιά" του κόμβου και κατ' επέκταση μας δίνει μια εικόνα για τις εξαρτήσεις μεταξύ των κόμβων. Με αυτήν την πληροφορία μπορούμε να καταλάβουμε την ροή των εισόδων και εξόδων του συστήματος. Μία πληροφορία που δεν διαθέτει το σύστημα και θεωρήσαμε οτι είναι μεγάλης σημασίας είναι ο κώδικας που καλείται να εκτελέσει κάθε κόμβος-διεργασία. Επειδή το να έχουμε ένα πεδίο το οποίο να περιέχει κώδικα, ενέχει αρκετούς κινδύνους, επιλέξαμε η πληροφορία αυτή να έρχεται με την μορφή ενός container image. Ένα στιγμυότυπο ενός κόμβου του operatorGraph φαίνεται παρακάτω.

```
{
    "operatorId": "5",
    "children": [
        "6"
    ],
    "image": "billothi/thesis:merger_v2",
     "inputData": 2048
}
```

Έξοδος

Η έξοδος του planner μας παρέχει πληροφορίες σχετικά με το μηχάνημα στο οποίο θα εκτελεστεί κάθε διεργασία. Τα πεδία του αρχείο έξόδου είναι τα:

- **placement**: Η τοποθέτηση των διεργασιών σε μηχανήματα
- **objective**: Παρέχει πληροφορίες για τον χρόνο και την ισχύ που κατανάλωσε.

Η πληροφορία που παίρνουμε από αυτό το αρχείο έχει να κάνει με το σε ποίο μηχάνημα (node_id) θα τοποθετήσουμε την διεργασία operator_id όπως φαίνεται στο παρακάτω στιγμυότυπο του πεδίου placement.

```
{
  "operator_id": "5",
```

```
    "device_id": "node_1-device_1",
    "node_id": "btheo-6",
    "device_type": "CPU"
}
```

3.1.2 Component

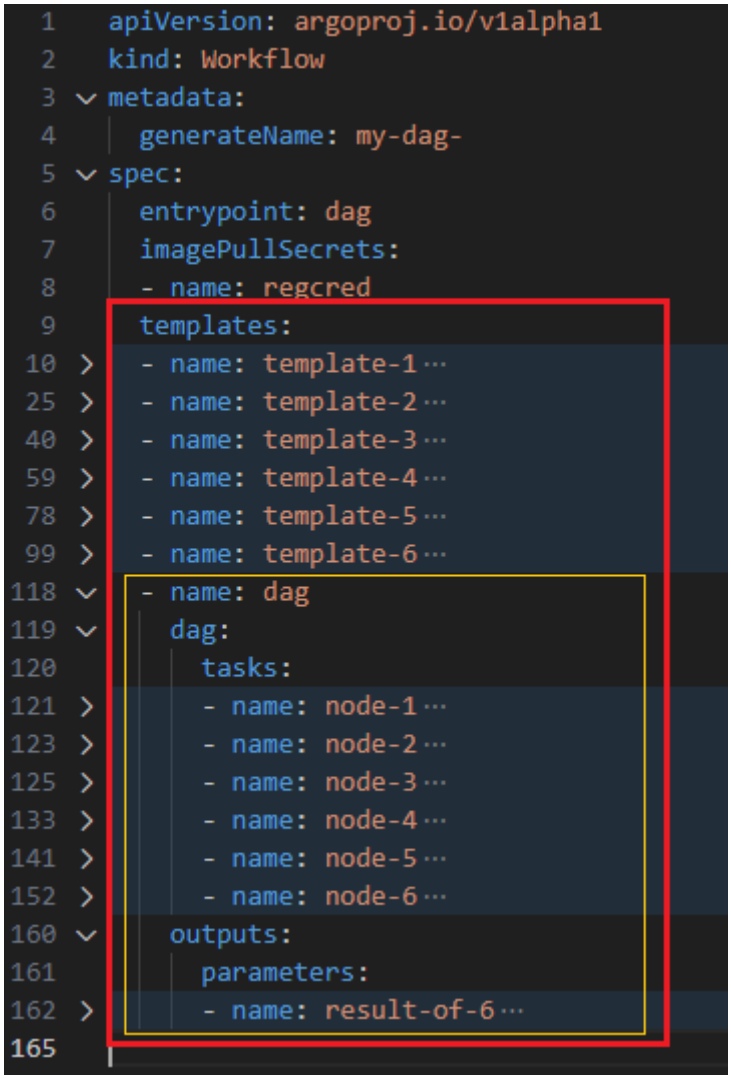
Ένας τρόπος για να τρέξουμε συγκεκριμένες διεργασίες σε ένα μηχάνημα χρησιμοποιώντας τον Kubernetes είναι να περιγράψουμε για κάθε ένα task το αντίστοιχο Pod που θα πρέπει να εκτελεστεί. Ένα τέτοιο εγχείρημα παρουσιάζει αρκετά προβλήματα όσον αφορά την μεταφορά δεδομένων από ένα Pod σε ένα άλλο. Ένα πολύ χρήσιμο εργαλείο που μας βοηθάει να αντιμετωπίσουμε αυτό το πρόβλημα είναι το Argo Workflows[15]. Το Argo Workflows είναι ένα πρόγραμμα ανοιχτού κώδικα το οποίο είναι φτιαγμένο για να εκτελεί Workflows στον Kubernetes. Με αυτό το εργαλείο ορίζουμε τα βήματα ενός Workflow είτε με την μορφή βημάτων, ώστε να εκτελείται μια αλυσίδα διεργασιών, είτε με την μορφή DAG (Directed Acyclic Graph), το οποίο και επιλέξαμε στην υλοποίηση μας.

Στην συνέχεια θα αναλύσουμε την δομή ενός Workflow και κατ' επέκταση του αρχείου που παράγουμε ως έξοδο του component. Όπως βλέπουμε στην Εικόνα 3.2 μπορούμε να χωρίσουμε το yaml αρχείο σε δύο κομμάτια. Τα templates και το dag το οποίο είναι μια ειδική περίπτωση template.

Templates

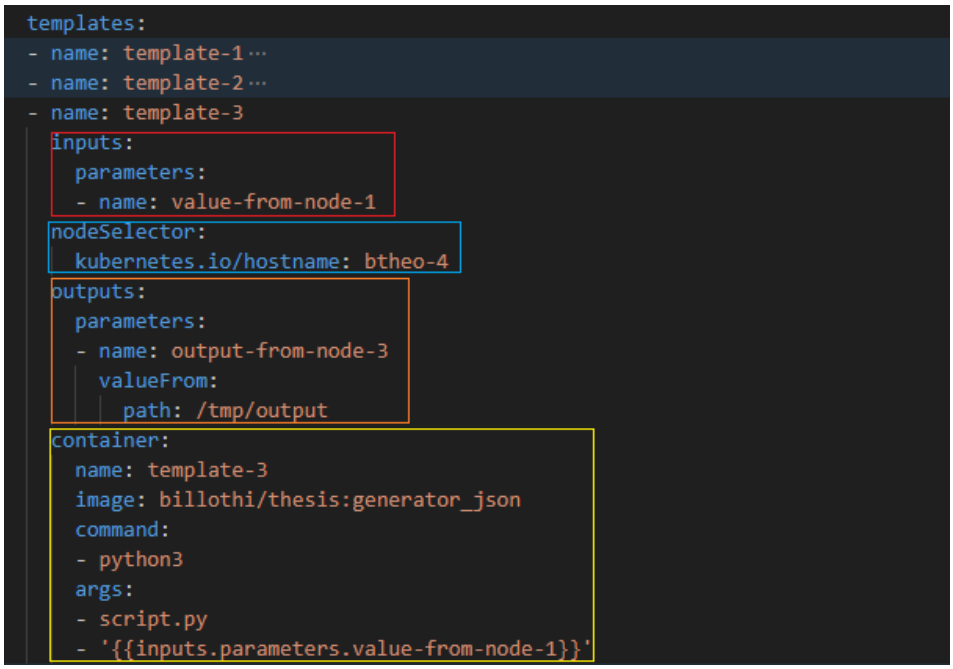
Στο πεδίο των templates ουσιαστικά ορίζουμε τον κάθε κόμβο του γράφου διεργασιών, δηλώνοντας ποιό container θα τρέξει, πόσες και ποιές εισόδους θα έχει - στο σύστημά μας έχουμε κάνει την παραδοχή πως οι κόμβοι χωρίς γονείς δεν αναμένουν κάποια είσοδο και οτι κάθε κόμβος παράγει μία έξοδο- την έξοδο που παράγει, η οποία στην συνέχεια θα χρησιμοποιηθεί ως είσοδος των κόμβων-παιδιών του και τέλος και πιο σημαντικό με το πεδίο **nodeSelector.kubernetes.io/hostname** ορίζουμε το μηχάνημα στο οποίο θα τρέξει αυτός ο κόμβος. Βλέπουμε λοιπόν πόσο εύκολα μπορούμε να ορίσουμε το μηχάνημα το οποίο θα τρέξει μια διεργασία χωρίς να χρειαστεί να έχουμε προβλέψει να γράψουμε τις συναρτήσεις scoring και πιθανώς filtering με τρόπο που να είναι συμβατός με τα plugins του Kubernetes. Το γεγονός αυτό δίνει επιπλέον την δυνατότητα στο component μας να είναι συμβατό με οποιοδήποτε σύστημα μπορεί να υπολογίσει την βέλτιστη αντιστοιχία ανάμεσα σε διεργασίες και μηχανήματα και θέλει να χρησιμοποιήσει τον Kubernetes ως μέσο εκτέλεσης των διεργασιών αυτών.

Προκειμένου να ορίσουμε τις εισόδους που θα έχει ο κάθε κόμβος και κατ' επέκταση το container που θα εκτελεστεί δεν κάναμε τίποτα παραπάνω από το να αντιστρέψουμε την τοπολογία που παίρνουμε από την είσοδο του planner και αντί να έχουμε πληροφορίες για τα παιδιά κάθε κόμβου να μπορούμε να έχουμε πληροφορία για τους γονείς του. Τέλος αξίζει να σημειωθεί πως το Argo Workflows



Εικόνα. 3.2: Επισκόπηση του configuration file

μας δίνει την δυνατότητα να επιλέξουμε τον τύπο της εξόδου ανάμεσα σε parameters και artifacts. Με τις παραμέτρους μπορούμε να χρησιμοποιούμε την έξοδο κάθε βήματος ως παράμετρο κάποιου επόμενου ενώ με τα artifacts μπορούμε να μεταφέρουμε αρχεία ανάμεσα σε διαφορετικά βήματα. Χρησιμοποιώντας αυτές τις τεχνικές η έξοδος γράφεται εντός ενός αρχείου το οποίο δηλώνουμε στο πεδίο **outputs.parameters.valueFrom.path**.



Εικόνα. 3.3: Περιγραφή του πεδίου template στο configuration αρχείο

DAG

Σε αυτό το κομμάτι του configuration file περιγράφουμε τον γράφο ο οποίος αποτελεί το Workflow μας. Όπως βλέπουμε η περιγραφή του dag βρίσκεται μέσα σε ένα template το οποίο έχει το ίδιο όνομα με το πεδίο spec.entrypoint (Εικόνα 3.2). Το πεδίο αυτό χρησιμοποιείται για να αναγνωρίσει το argo από ποιό σημείο θα αρχίσει να εκτελεί τον κώδικα.

Όπως είπαμε και παραπάνω το Argo Workflows μας δίνει την δυνατότητα να επιλέξουμε τον τρόπο με τον οποίο θέλουμε να τρέχουν οι εργασίες μας ανάμεσα σε steps και dag. Η περίπτωση του κατευθυνόμενου ακυκλικού γράφου, εκτός του οτι μπορεί να περιγράψει ευκολότερα μια ροή εργασιών και τις εξαρτήσεις που υπάρχουν ανάμεσα στις διάφορες διεργασίες, κρίναμε οτι είναι η κατάλληλη στην περίπτωσή μας που ήδη έχουμε ως είσοδο έναν γράφο με διεργασίες.

Στο πεδίο dag ορίζουμε τα διαφορετικά tasks που θα τρέξουν, περιέχοντας πληροφορία για το ποιό template από αυτά που ορίσαμε παραπάνω θα τρέξει ο κόμβος, από ποιούς κόμβους εξαρτάται και συνεπώς θα περιμένει να ολοκληρώσουν τις εργασίες τους πριν ξεκινήσει και τέλος εαν ο κόμβος δέχεται κάποια είσοδο, το όνομα και την τιμή της εισόδου. Ενδεικτικά στην Εικόνα 3.4 βλέπουμε οτι οι κόμβοι με ονόματα node-1 και node-2 θα τρέξουν τα template-1 και template-2 όπως αυτά έχουν οριστεί παραπάνω στο αρχείο χωρίς να έχουν κάποια εξάρτηση ή να περιμένουν κάποια είσοδο, οπότε θα τρέξουν παράλληλα. Αντίθετα ο κόμβος node-3 ο οποίος θα εκτελέσει το template-3 παίρνοντας ως είσοδο το αποτέλεσμα του node-1, το όνομα και η

τιμή της οποίας ορίζεται στο πεδίο arguments.parameters, θα πρέπει να περιμένει να ολοκληρωθεί η εκτέλεση της διεργασίας του node-1 κάτι το οποίο βλέπουμε πως έχει οριστεί στο πεδίο dependencies.

```
dag:
  tasks:
  - name: node-1
    template: template-1
  - name: node-2
    template: template-2
  - name: node-3
    template: template-3
    dependencies:
    - node-1
    arguments:
      parameters:
      - name: value-from-node-1
        value: '{{tasks.node-1.outputs.parameters.output-from-node-1}}'
```

Εικόνα. 3.4: Περιγραφή πεδίου dag

Όπως κάθε template, έτσι και το ειδικό template dag εκτός από την περιγραφή των βημάτων του παράγει μια ή παραπάνω εξόδους η οποία είναι η έξοδος κάποιου κόμβου του dag. Αυτή η έξοδος μπορεί στην συνέχεια να διοχετευθεί σε κάποιον άλλο κόμβο και από εκεί να συνεχίσει η διαδικασία όπως έχει περιγραφεί παραπάνω.

Αφού λοιπόν έχουμε δει την δομή με την οποία κατασκευάζεται ένα Workflow μπορούμε να καταλάβουμε οτι το εγχείρημα να εκτελέσουμε τις διεργασίες στα μηχανήματα τα οποία έχει επιβάλλει ο planner δεδομένων και της πληροφορίας που μας παρέχουν τόσο η είσοδος όσο και η έξοδος του μπορεί να γίνει πολύ απλά κατασκευάζοντας ένα κατάλληλο Workflow και δηλώνοντας στο πεδίο nodeSelector του κόμβου το μηχάνημα στο οποίο θέλουμε να τρέξει. Για τον λόγο αυτόν κατασκευάσαμε ένα σύστημα[1] σε γλώσσα Python το οποίο αρχικά διαβάζοντας την είσοδο και την έξοδο του planner προχωρούν στην δημιουργία του configuration αρχείου με τον τρόπο που περιγράφηκε παραπάνω. Αρχικά αφού βρει όλες τις εξαρτήσεις των κόμβων δημιουργεί για κάθε κόμβο το αντίστοιχο block για το template και στην συνέχεια δημιουργεί το block για το dag. Τέλος αφού προσθέσει τις κατάλληλες επικεφαλίδες παράγει το αρχείο manifest.yaml το οποίο είναι και αυτό που θα τρέξει ο kubernetes. Τέλος έχουμε δώσει την επιλογή ο χρήστης να μην συμπληρώσει το πεδίο nodeSelector εκτελώντας το αρχείο main.py με το flag -ns 0 ως εξής:

```
$ python3 main.py --ns 0
```

Σε αυτήν την περίπτωση ο scheduler του kubernetes θα αποφασίσει σε ποιο μηχάνημα θα τρέξει κάθε διεργασία. Την εκτέλεση του Workflow μπορούμε να την δούμε μέσω command line χρησιμοποιώντας την εντολή:

[1] https://github.com/billothi/thesis

```
$ argo submit -n default manifest.yaml --watch
```

Με το flag –watch μπορούμε να δούμε στο command line το workflow να εκτελείται και όταν ολοκληρωθεί με την εντολή

```
$ argo logs @latest
```

Μπορούμε να δούμε τα logs που έχει παράγει κάθε κόμβος. Τέλος με το command

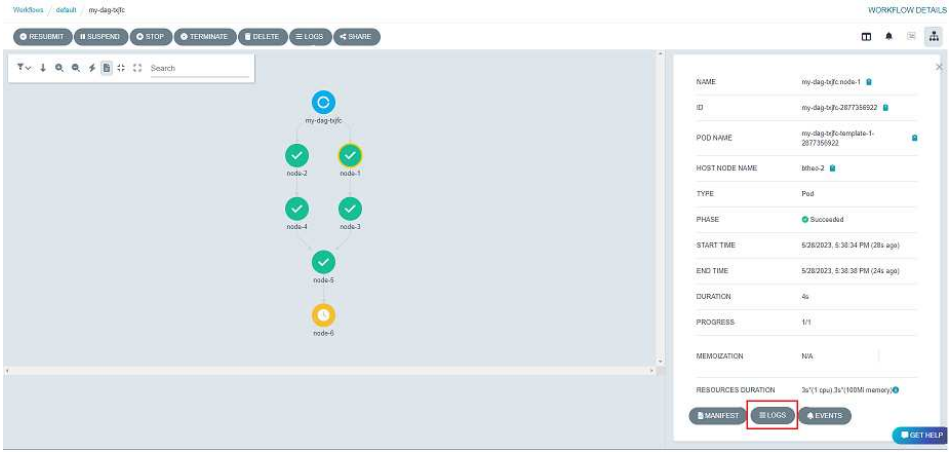
```
$ argo get @latest -o json
```

παίρνουμε σε μορφή json χρήσιμα στατιστικά για το workflow, όπως την κατάσταση στην οποία τερμάτησε κάθε κόμβος, το όνομα του μηχανήματος το οποίο το εκτέλεσε, την ώρα εκκίνησης και τερματισμου της διεργασίας, το πλήθος των πόρων που χρησιμοποίησε και τέλος το output που έβγαλε ο κάθε κόμβος. Εκμεταλλευόμενοι τις παραπάνω λειτουργίες δημιουργήσαμε ένα script (stats.py) το οποίο αφού δημιουργήσει το αρχείο manifest.yaml, εκτελεί το workflow και στην συνέχεια δημιουργεί ένα json αρχείο με τα στατιστικά του workflow, συμπεριλαμβανομένης της εξόδου και του συνολικού χρόνου εκτέλεσης του workflow. Παρακάτω παραθέτουμε ενδεικτικά τα στατιστικά για το Pod "my-dag-5wmcq-2839021703" από το αρχείο stats.json.


```
"my-dag-5wmcq-2839021703": {
        "id": "my-dag-5wmcq-2839021703",
        "name": "my-dag-5wmcq.node-4",
        "displayName": "node-4",
        "phase": "Succeeded",
        "hostNodeName": "btheo-5",
        "startedAt": "2023-05-28T14:31:33Z",
        "finishedAt": "2023-05-28T14:31:39Z",
        "resourcesDuration": {
                "cpu": 7,
                "memory": 7
        },
        "totalTime": "0:00:06",
        "outputSize": 10729
}
```


Τέλος προκειμένου να έχουμε μια καλύτερη εικόνα για το Workflow, τα βήματα που εκτελεί και τις ενδιάμεσες εξόδους, αλλά και να μας είναι ευκολότερο το debug, χρησιμοποιήσαμε το UI του Argo Workflows[2] (Εικόνα 3.5) το οποίο μας δίνει σε πραγματικό χρόνο μια οπτικοποίηση του workflow καθώς και όλες τις απαραίτητες πληροφορίες που χρειαζόμαστε για κάθε κόμβο.

[2]https://argoproj.github.io/argo-workflows/argo-server/



Εικόνα. 3.5: Argo Workflows UI

3.2 Τοπολογία

Για την υλοποίηση του παραπάνω συστήματος αλλά και για τις δοκιμές μας θα πρέπει να δημιουργήσουμε ένα cluster από διαφορετικά μηχανήματα. Στην περίπτωσή μας θέλουμε να προσομοιάσουμε όσο περισσότερο γίνεται ένα Edge Cloud περιβάλλον στο οποίο να υπάρχει το στοιχείο της ετερογένειας αλλά και ένα μηχάνημα το οποίο θα μπορεί να προσομοιώσει έναν cloud server, ώντας πολύ πιο ισχυρό υπολογιστικά από τα υπόλοιπα. Για τον λόγο αυτό χρησιοποιήσαμε τα εξής μηχανήματα:

- 8 VMs με 2 cores και 4GB μνήμη (btheo1,2,3,4,5,6,7,8)
- 2 VMs με 1 core και 1GB μνήμη (btheo-s-1,2)
- 1 VM με 2 cores και 16GB μνήμη (btheo-l)

Από αυτά τα μηχανήματα επιλέξαμε ένα για τον ρόλο του master (2 cores και 4GB μνήμη) και τα υπόλοιπα αποτελούν τους workers της τοπολογίας μας. Σε όλα τα μηχανήματα εγκαταστήσαμε τον Kubernetes και τα τοποθετήσαμε σε ένα private δίκτυο για λόγους ασφαλείας.

Πειράματα

Όπως είδαμε αναλυτικότερα παραπάνω το σύστημά μας σχεδιάστηκε προκειμένου να μπορεί να ενσωματώνεται σε διάφορες εργασίες οι οποίες αναπτύσσουν αλγόριθμους για την βέλτιστη τοποθέτηση διεργασιών σε μία συστάδα μηχανημάτων και επιθυμούν να χρησιμοποιήσουν τον Kubernetes για την εκτέλεση των διεργασιών, χωρίς να εξαρτάται καθόλου από το είδος του συστήματος ή τον αλγόριθμο που αυτό χρησιμοποιεί. Καθώς λοιπόν το πλεονέκτημα το οποίο έχει το σύστημά μας σε σχέση με τον Scheduler του Kubernetes είναι η ευκολία επιβολής διαφορετικών αλγορίθμων scoring για το οποίο δεν μπορούμε κάπως να συγκρίνουμε τις δύο μεθόδους δημιουργήσαμε ορισμένα σενάρια, από τον κόσμο του IoT και της ανάλυσης δεδομένων, στα οποία θα συγκρίνουμε την λειτουργία των δύο συστημάτων.

Πρόβλεψη Θερμοκρασίας με Knn

Σε αυτό το σενάριο θα προσομοιώσουμε την λειτουργία ενός έξυπνου σπιτιού το οποίο διαθέτει συσκευές οι οποίες δίνουν μια εκτίμηση για τον καιρό δεδομένης της ώρας και της θερμοκρασίας της μέτρησης. Η διαδικασία αυτή γίνεται σε βήματα για τα οποία δημιουργήσαμε τα αντίστοιχα images.

Δημιουργία Παρατηρήσεων

Σε αυτό το βήμα δημιουργούνται n παρατηρήσεις θερμοκρασίας με συγκεκριμένη ημερομηνία, ώρα καθώς επίσης καταγράφεται και ο καιρός που είχε όταν έγινε αυτή η παρατήρηση. Παρακάτω βλέπουμε ένα στιγμιότυπο από αυτές τις εγγραφές.


```
{
    "0":{
        "id":"14380020090219",
        "time":"14:38:00",
        "date":"2009-02-19",
        "float_time":14.38,
        "int_date":20090219,
        "temperature":-4,
        "weather":"Snow",
```

```
        "weather_class":3
    }
}
```

Φιλτράρισμα Εγγραφών

Σε αυτό το βήμα οι παρατηρήσεις φιλτράρονται και έρχονται σε μορφή κατάλληλη για είσοδο σε επόμενο βήμα που είναι ο αλγόριθμος knn. Τα πεδία αυτών των εγγραφών είναι το data που είναι μια λίστα που περιέχει τα ζεύγη θερμοκρασίας και ώρας σε μορφή δεκαδικού αριθμού και το πεδίο weather_class που περιέχει την κλάση καιρού στην οποία ανήκει το ζεύγος data.

```
{
    "data": [[-4, 14.38]],
    "weather_class": [3]
}
```

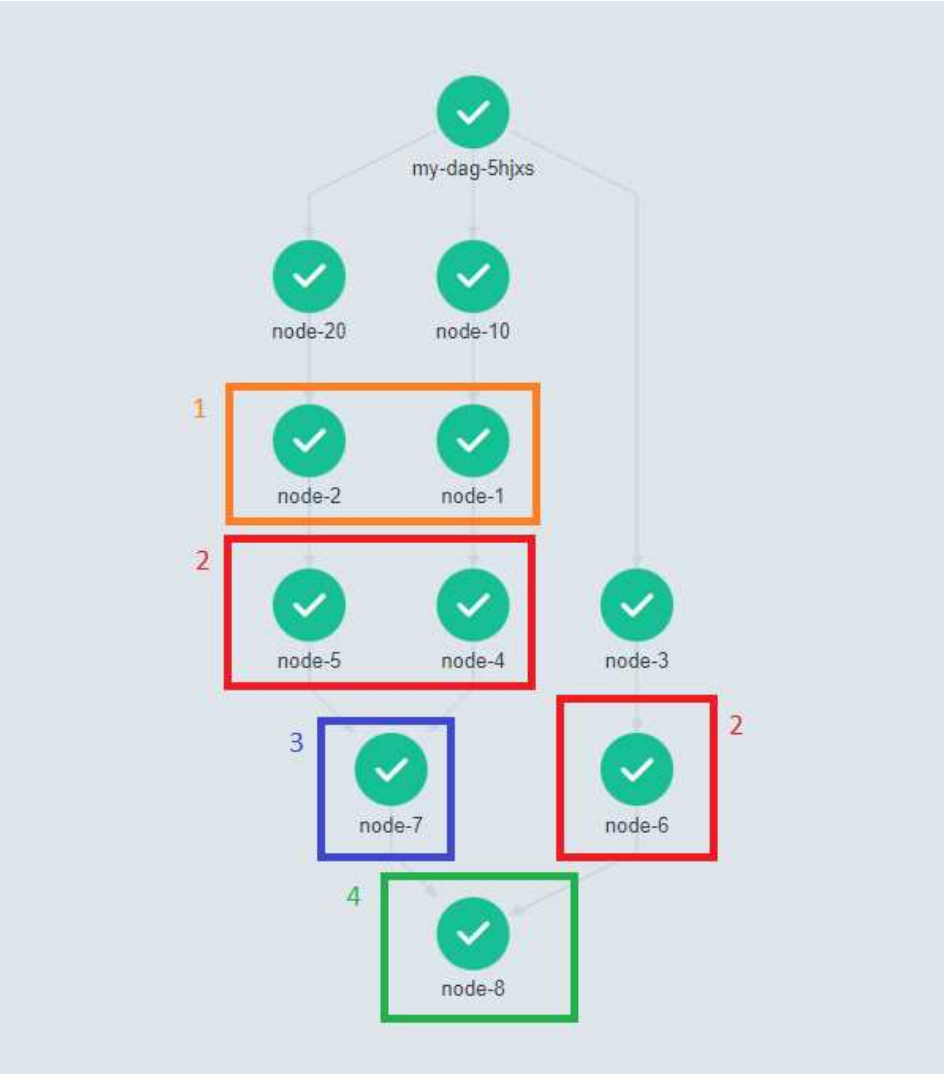
Συγχώνευση Εγγραφών

Σε αυτό το βήμα παίρνουμε διάφορες εγγραφές και τις συγχωνεύουμε σε μία. Η διαδικασία της συγχώνευσης έχει δημιουργηθεί με τέτοιον τρόπο ωστε να μπορεί είτε να προηγείται είτε να ακολουθεί το φιλτράρισμα

Πρόβλεψη

Το βήμα αυτό παίρνει ως είσοδο το ιστορικό των παρατηρήσεων καθώς και μία τυχαία παρατήρηση της θερμοκρασίας και με την χρήση του αλγορίθμου KNN προβλέπει τον καιρό που θα κάνει.

Στην παρακάτω εικόνα 4.1 βλέπουμε τα στάδια που περιγράψαμε παραπάνω. Στο βήμα 1 γίνεται η τυχαία δημιουργία ιστορικού θερμοκρασιών από τους κόμβους node-1 και node-2 οι οποίοι δέχονται ως είσοδο το μέγεθος των δεδομένων από τους node-10 και node-20 αντιστοιχα. Στο βήμα 2 γίνεται το φιλτράρισμα των δεδομένων. Εδώ να σημειωθεί οτι ο node-6 φιλτράρει τα δεδομένα της παρατήρησης για την οποία θέλουμε να προβλευθεί ο καιρός. Στο βήμα 3 γίνεται συγχώνευση των δεδομένων και τέλος στο βήμα 4 εκτελείται ο Knn.



Εικόνα. 4.1: Διαδικασία Πρόβλεψης Θερμοκρασίας με Knn

Όσον αφορά το παραπάνω σενάριο, δεδομένου του οτι οι διεργασίες γίνονται από οικιακές συσκευές μας ενδιαφέρει κατα κύριο λόγο η κατανάλωση να είναι μικρή. Αυτό σημαίνει οτι tasks τα οποία απαιτούν μικρή υπολογιστική ισχύ να εκτελούνται από μικρά μηχανήματα ενω μεγάλα tasks όπως για παράδειγμα ο Knn να εκτελούνται από μεγαλύτερα μηχανήματα. Αρχικά τρέχουμε το workflow λαμβάνοντας υπόψιν έναν planner ο οποίος έχει τοποθετήσει τις διεργασίες σε μηχανήματα λαμβάνοντας υπόψιν αυτό το γεγονός. Παρακάτω παραθέτουμε μέρος από το αρχείο json που ορίζει το που θα τρέξει κάθε διεργασία και στη συνέχεια το αποτέλεσμα του workflow.

```
{
    "placement":
    [
        {"operator_id":"10","node_id":"btheo-s-1"},
        {"operator_id":"20","node_id":"btheo-s-2"},
        {"operator_id":"1","node_id":"btheo-s-1"},
        {"operator_id":"2","node_id":"btheo-s-2"},
        {"operator_id":"3","node_id":"btheo-s-1"},
        {"operator_id":"4","node_id":"btheo-2"},
        {"operator_id":"5","node_id":"btheo-3"},
        {"operator_id":"6","node_id":"btheo-7"},
        {"operator_id":"7","node_id":"btheo-6"},
        {"operator_id":"8","node_id":"btheo-1"}
    ]
}
```


Εικόνα. 4.2: Τοποθέτηση Διεργάσιών σε Μηχανήματα από τον Custom Planner

```
{
    "my-dag-wlmpb-1429033991":
        {"id":"my-dag-wlmpb-1429033991","displayName":"node-10","hostNodeName":"btheo-s-1"},
    "my-dag-wlmpb-3274027628":
        {"id":"my-dag-wlmpb-3274027628","displayName":"node-20","hostNodeName":"btheo-s-2"},
    "my-dag-wlmpb-1141357581":
        {"id":"my-dag-wlmpb-1141357581","displayName":"node-1","hostNodeName":"btheo-s-1"},
    "my-dag-wlmpb-1091024724":
        {"id":"my-dag-wlmpb-1091024724","displayName":"node-2","hostNodeName":"btheo-s-2"},
    "my-dag-wlmpb-1107802343":
        {"id":"my-dag-wlmpb-1107802343","displayName":"node-3","hostNodeName":"btheo-s-1"},
    "my-dag-wlmpb-1057469486":
        {"id":"my-dag-wlmpb-1057469486","displayName":"node-4","hostNodeName":"btheo-2"},
    "my-dag-wlmpb-1074247105":
        {"id":"my-dag-wlmpb-1074247105","displayName":"node-5","hostNodeName":"btheo-3"},
    "my-dag-wlmpb-1023914248":
        {"id":"my-dag-wlmpb-1023914248","displayName":"node-6","hostNodeName":"btheo-7"},
    "my-dag-wlmpb-1040691867":
        {"id":"my-dag-wlmpb-1040691867","displayName":"node-7","hostNodeName":"btheo-6"},
    "my-dag-wlmpb-990359010":
        {"id":"my-dag-wlmpb-990359010","displayName":"node-8","hostNodeName":"btheo-1"}
}
```


Εικόνα. 4.3: Αποτελέσματα εκτέλεσης Workflow αφού έχει επιβληθεί η τοποθέτηση του Custom Scheduler

Ανάλυση Δεδομένων

Σε αυτό το σενάριο διαθέτουμε στοιχεία για τους φοιτητές και επιθυμούμε να εξάγουμε ορισμένα στατιστικά όπως ο μέσος όρος, το πλήθος των φοιτητών κ.α. Τα βήματα που χρησιμοποιούνται είναι παρόμοια με τα παραπάνω.

Εικόνα. 4.4: Αποτελέσματα εκτέλεσης Workflow με τον Default Scheduler του Kubernetes

Δημιουργία Εγγραφών

Σε αυτό το βήμα δημιουργούμε το τυχαιο dataset για n τυχαίους φοιτητές το οποίο περιέχει τα τηλέφωνα, e-mails, όνοματεπώνυμο και βαθμό σε κάποιο μάθημα

Συγχώνευση Εγγραφών

Σε αυτό το βήμα συγχωνεύουμε τις διάφορες εγγραφές σε μία

Εξαγωγή Στατιστικών

Σε αυτό το βήμα γίνεται επεξεργασία των εγγραφών και εξάγονται στατιστικά

Σε αυτό το σενάριο πολύ σημαντικό ρόλο παίζει η ασφάλεια καθώς είναι σαφές οτι δεν θέλουμε να διαρρεύσει κάποιο τηλέφωνο ή e-mail φοιτητή. Συνεπώς Θα επιθυμούσαμε να εκτελεστεί όλη η διαδικασία στο τοπικό δίκτυο. Αν θεωρήσουμε οτι το μηχάνημα btheo-l (Βλ. 3.2) είναι server στο cloud θα επιθυμούσαμε εαν είναι δυνατό να αποφευχθεί η χρήση του. Παρακάτω βλέπουμε την τοποθέτηση που έχει επιλέξει ένας planner ο οποίος έχει ως κριτήριο την ασφάλεια των δεδομένων (Εικόνα 4.5).

```
{
    "placement":
    [
        {"operator_id":"1","node_id":"btheo-s-2"},
        {"operator_id":"2","node_id":"btheo-s-1"},
        {"operator_id":"3","node_id":"btheo-4"},
        {"operator_id":"4","node_id":"btheo-5"},
        {"operator_id":"5","node_id":"btheo-6"},
        {"operator_id":"6","node_id":"btheo-7"}
    ]
}
```


Εικόνα. 4.5: Τοποθέτηση Διεργάσιών σε Μηχανήματα από τον Custom Planner


```
{
    "my-dag-5t9sx-1969669380":
        {"id":"my-dag-5t9sx-1969669380","displayName":"node-1","hostNodeName":"btheo-s-2"},
    "my-dag-5t9sx-2020002237":
        {"id":"my-dag-5t9sx-2020002237","displayName":"node-2","hostNodeName":"btheo-s-1"},
    "my-dag-5t9sx-2003224618":
        {"id":"my-dag-5t9sx-2003224618","displayName":"node-3","hostNodeName":"btheo-4"},
    "my-dag-5t9sx-1919336523":
        {"id":"my-dag-5t9sx-1919336523","displayName":"node-4","hostNodeName":"btheo-5"},
    "my-dag-5t9sx-1902558904":
        {"id":"my-dag-5t9sx-1902558904","displayName":"node-5","hostNodeName":"btheo-6"},
    "my-dag-5t9sx-1952891761":
        {"id":"my-dag-5t9sx-1952891761","displayName":"node-6","hostNodeName":"btheo-7"}

}
```


Εικόνα. 4.6: Αποτελέσματα εκτέλεσης Workflow αφού έχει επιβληθεί η τοποθέτηση του Custom Scheduler


```
{
    "my-dag-5t9sx-1969669380":
        {"id":"my-dag-5t9sx-1969669380","displayName":"node-1","hostNodeName":"btheo-1"},
    "my-dag-5t9sx-2020002237":
        {"id":"my-dag-5t9sx-2020002237","displayName":"node-2","hostNodeName":"btheo-1"},
    "my-dag-5t9sx-2003224618":
        {"id":"my-dag-5t9sx-2003224618","displayName":"node-3","hostNodeName":"btheo-1"},
    "my-dag-5t9sx-1919336523":
        {"id":"my-dag-5t9sx-1919336523","displayName":"node-4","hostNodeName":"btheo-1"},
    "my-dag-5t9sx-1902558904":
        {"id":"my-dag-5t9sx-1902558904","displayName":"node-5","hostNodeName":"btheo-1"},
    "my-dag-5t9sx-1952891761":
        {"id":"my-dag-5t9sx-1952891761","displayName":"node-6","hostNodeName":"btheo-1"}

}
```


Εικόνα. 4.7: Αποτελέσματα εκτέλεσης Workflow με τον Default Scheduler του Kubernetes

Επίλογος

5.1 Συμπεράσματα

Από τα παραπάνω σενάρια βλέπουμε οτι το Scheduling του Kubernetes γίνεται χρησιμοποιώντας κάποιον Greedy αλγόριθμο επιλέγοντας το βέλτιστο διαθέσιμο μηχάνημα για την εκτέλεση της κάθε διεργασίας. Αν και αυτό είναι επιθυμητό σε περιπτώσεις που μας ενδιαφέρει η ταχύτητα περάτωσης μίας εργασίας, δεν φαίνεται να καλύπτονται περιπτώσεις όπου μας ενδιαφέρει η ασφάλεια των δεδομένων ή η κατανάλωση ενέργειας. Ωστόσο τόσο στο Cloud όσο και στο Edge Cloud επιθυμούμε να αναπτύσσουμε εφαρμογές οι οποίες να τρέχουν σε συγκεκριμένα μηχανήματα και με άλλα κριτήρια πέρα από την ταχύτητα όπως είδαμε παραπάνω. Ενώ ο Scheduler του Kubernetes είναι σχεδιασμένος προκειμένου να μπορούμε να ορίζουμε διάφορα plugins για την εφαρμογή διαφορετικών τεχνικών Scheduling είναι δύσκολος όσον αφορά την επεξεργασία. Επίσης δεν καλύπτει περιπτώσεις στις οποίες έχουμε ήδη υλοποιημένα συστήματα τα οποία εκτελούν όλη την φάση του Scheduling και απλά επιθυμούν να χρησιμοποιήσουν τον Kubernetes για την κατανομή των εργασιών. Για αυτούς τους λόγους βλέπουμε οτι το σύστημα το οποίο αναπτύξαμε μπορεί με πολύ απλό τρόπο να αντικαταστήσει τον Scheduler με όποιο σύστημα επιθυμούμε απλά γνωρίζοντας την είσοδο και την έξοδό του.

5.2 Μελλοντική Δουλειά

Αυτή τη στιγμή το σύστημά μας έχει τρία μεγάλα μειονεκτήματα. Το πρώτο είναι οτι λειτουργεί με την χρήση των παραμέτρων και δεν λαμβάνει υπόψιν τα artifacts. Αν και σε απλές περιπτώσεις οπου θέλουμε η ροή των δεδομένων ανάμεσα σε κόμβους να είναι τέτοια οπου μπορεί να γίνει με την χρήση json αντικειμένων, το σύστημά μας λειτουργεί σωστά, δεν είναι λίγες οι φορές στις οποίες θέλουμε να μεταφέρουμε αρχεία από το ένα μηχάνημα στο άλλο κάτι που γίνεται μόνο με την χρήση των artifacts. Το δεύτερο μειονέκτημα είναι οτι απαιτείται από τους χρήστες οι διεργασίες να έρχονται υπο την μορφή images. Θα ήταν χρήσιμο λοιπόν να αναπτυχθεί ένα σύστημα το οποίο να λαμβάνει τις διεργασίες σε οποιαδήποτε μορφή και στην συνέχεια να γίνεται αυτόματη μετατροπή σε container. Τέλος κατά τον σχεδιασμό του συστήματος

δεν λάβαμε καθόλου υπόψιν την κατάσταση του δικτύου ανάμεσα στα μηχανήματα. Σε πραγματικές καταστάσεις θα μας ενδιέφερε πολύ να μπορούμε να επηρεάζουμε δυναμικά την κατανομή των εργασιών του εκάστοτε planner παρέχοντάς του πληροφορία για την κατάσταση του δικτύου. Μία ακόμα πολύ ενδιαφέρουσα προσθήκη στο σύστημα θα ήταν να μπορούμε να δεχόμαστε είσοδο από παραπάνω από έναν planner. Με αυτόν τον τρόπο αναβαθμίζεται η χρησιμότητα του συστήματός μας καθώς επιτρέπουμε στους χρήστες να μπορούν να χρησιμοποιήσουν πάνω από μία μετρική για τις διεργασίες που τους ενδιαφέρουν, να λαμβάνουν για παράδειγμα υπόψιν την ταχύτητα για κάποιες διεργασίες ενώ για κάποιες άλλες την κατανάλωση.

Βιβλιογραφία

[1] Peter Mell, Tim Grance, et al. "The NIST definition of cloud computing". In: (2011).

[2] Tharam Dillon, Chen Wu, and Elizabeth Chang. "Cloud Computing: Issues and Challenges". In: *2010 24th IEEE International Conference on Advanced Information Networking and Applications*. 2010, pp. 27–33. DOI: 10.1109/AINA.2010.187.

[3] Alexa Huth and James Cebula. "The basics of cloud computing". In: *United States Computer* (2011), pp. 1–4.

[4] Greg Boss et al. "Cloud computing". In: *IBM white paper* 321 (2007), pp. 224–231.

[5] Jianli Pan and James McElhannon. "Future edge cloud and edge computing for internet of things applications". In: *IEEE Internet of Things Journal* 5.1 (2017), pp. 439–449.

[6] Luiz Bittencourt et al. "The internet of things, fog and cloud continuum: Integration and challenges". In: *Internet of Things* 3 (2018), pp. 134–155.

[7] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The internet of things: A survey". In: *Computer networks* 54.15 (2010), pp. 2787–2805.

[8] *What are containers?* URL: https://www.ibm.com/topics/containers.

[9] *Kubernetes Overview*. URL: https://kubernetes.io/docs/concepts/overview/.

[10] *What is Kubernetes?* URL: https://www.ibm.com/topics/kubernetes.

[11] *Kubernetes Components*. URL: https://kubernetes.io/docs/concepts/overview/components/.

[12] *Controllers*. URL: https://kubernetes.io/docs/concepts/architecture/controller/.

[13] *Scheduler Configuration*. URL: https://kubernetes.io/docs/reference/scheduling/config/.

[14] *Kubernetes Scheduler*. URL: https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/.

[15] *Argo Workflows*. URL: https://argoproj.github.io/argo-workflows/.