Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τηλεπικοινωνιών

Δημιουργία, διαχείριση και αποθήκευση μηνυμάτων IoT περιβάλλοντος με τη χρήση RabbitMQ Server & ανάπτυξη του full-stack συστήματος σε Kubernetes

Διπλωματική Εργασία
του
Εμμανουήλ Ζήση-Μήλη

Επιβλέπων: Εμμανουήλ Τσαρδούλιας
Μεταδιδακτορικός Συνεργάτης Α.Π.Θ.

13 Σεπτεμβρίου 2021

Περίληψη

Η μετάβαση των τεχνολογιών του διαδικτύου προς αρχιτεκτονικές μικροϋπηρεσιών *microservices* και η ανάπτυξη του Διαδικτύου των Πραγμάτων *InternetofThings – IoT* συνέβαλλαν σημαντικά στην αύξηση των αναγκών για νέες μεθόδους αποδοτικής επικοινωνίας μεταξύ ανομοιογενών και διανεμημένων συστημάτων. Οι μεθοδολογίες μεσολάβησης μηνυμάτων *(brokered messaging)* λειτουργούν καλύτερα από τις τεχνολογίες/προσεγγίσεις REST και RPC σε συστήματα επικοινωνίας παραγωγών-καταναλωτών (μηνυμάτων) όπου είναι επιθυμητή τόσο η μετάδοση μεγάλου όγκου δεδομένων σε υψηλούς ρυθμούς όπως και η απεμπλοκή των υποσυστημάτων των παραγωγών και των καταναλωτών.

Μια ελαφριά και αξιόπιστη τεχνολογία που προσφέρει τα πλεονεκτήματα της μεσολάβησης μηνυμάτων είναι το RabbitMQ. Αποτελεί έναν κοινό κόμβο αποστολής, λήψης και διαχείρισης μηνυμάτων μεταξύ των διαφορετικών μερών ενός συστήματος. Υποστηρίζει ανάπτυξη σε πολλαπλές γλώσσες προγραμματισμού, μεταξύ των οποίων βρίσκεται και η Javascript, που είναι και η κύρια πλατφόρμα ανάπτυξης της παρούσας εργασίας. Το RabbitMQ βασίζεται στο Advanced Message Queuing Protocol (AMQP). Το AMQP είναι ένα πρωτόκολλο μηνυμάτων που υλοποιείται μέσω της χρήσης παραγωγών και καταναλωτών. Οι παραγωγοί δημιουργούν και αποστέλουν τα μηνύματα σε ένα ή περισσότερα θέματα topics, ενώ οι καταναλωτές λαμβάνουν μηνύματα από ένα ή περισσότερα θέματα αντίστοιχα και τα επεξεργάζονται. Με αυτές τις λειτουργίες μπορούν να χτιστούν πολύπλοκα και αποδοτικά συστήματα ειδικά σε συνθήκες ασύγχρονης επικοινωνίας, αναξιόπιστων δικτύων και εφαρμογών μεγάλων δεδομένων big data.

Η παρούσα διπλωματική εστιάζει στην ανάπτυξη ενός πλήρους συστήματος full-stack δημιουργίας, διαχείρισης και αποθήκευσης μηνυμάτων προερχόμενων από ένα IoT περιβάλλον. Η επικοινωνία των μηνυμάτων γίνεται μέσω της τεχνολογίας διακομιστή Rabbitmq Server. Τα μηνύματα που καλύπτουν συνθήκες ορισμένες από τους χρήστες αποθηκεύονται σε μια Βάση Δεδομένων (ΒΔ), για την οποία χρησιμοποιήθηκε η τεχνολογία MongoDB. Η ανάπτυξη του frontend έγινε πάνω σε Node.js - javascript και συγκεκριμένα με τη βιβλιοθήκη react.js, ενώ για το backend χρησιμοποιήθηκε η βιβλιοθήκη express.js σε Node.js - javascript και πάλι.

Τέλος για τη διευκόλυνση της διαχείρισης ολόκληρου του συστήματος, αυτό στήθηκε στο πλαίσιο της τεχνολογίας Kubernetes, η οποία προσφέρει την αυτοματοποίηση της ενορχήστρωσης των κομματιών του συστήματος. Τα υποσυστήματα του συνόλου συνθέτουν το καθένα μια αυτόνομη μονάδα η οποία ορίζεται και εσωκλείεται σε ένα δοχείο (container). Το Kubernetes είναι υπεύθυνο μετέπειτα για την επικοινωνία μεταξύ των δοχειοποιημένων υποσυστημάτων και τον συνεχή έλεγχο της καλής λειτουργίας τους. Για τη σύσταση του περιβάλλοντος Kubernetes επιλέχθηκε η τεχνολογία minikube καθώς προσφέρει εύκολη και γρήγορη δημιουργία ενός περιβάλλοντος Kubernetes συνδυάζοντας σαν τεχνολογία πολύχρονη παρουσία στον χώρο και μια ιδιαίτερα ενεργή κοινότητα.

Abstract

The transition of internet technologies to microservice architectures and the development of the Internet of Things (IoT) have significantly increased the need for new methods of efficient communication between heterogeneous and distributed systems. Brokered messaging methodologies work better than REST and RPC technologies / approaches in producer-consumer (messaging) communication systems where both high-throughput transmission of large volumes of data is desirable as well as the abstraction of producer and consumer subsystems.

A lightweight and reliable technology that offers the benefits of brokered messaging is RabbitMQ. It is a common platform for sending, receiving and managing messages between different parts of a system. It supports development in multiple programming languages, including Javascript, which is the main development platform of this dissertation. RabbitMQ is based on the Advanced Message Queuing Protocol (AMQP). AMQP is a messaging protocol implemented through the use of producers and consumers. Producers create and send messages to one or more topics topics, while consumers receive messages from one or more topics respectively and edit them. With these functions, complex and efficient systems can be built under conditions of asynchronous communication, unreliable networks and in big data applications.

This dissertation focuses on the development of a complete system (full-stack) for creating, managing and storing messages from an IoT environment. Messaging is carried out via a Rabbitmq Server. Messages that comply to the conditions defined by users are stored in a Database (DB) for which MongoDB technology was used. The frontend was developed on Node.js - javascript and specifically with the help of the library react.js, while for the backend the library express.js was used in conjunction to Node.js - javascript just like before.

Finally, to facilitate the management of the entire system, this was set up in the context of Kubernetes, which offers the automated orchestration of the parts of the system. The subsystems of the whole, each compose an autonomous unit which is defined and enclosed in a container. Kubernetes is then responsible for the communication between the containerized subsystems and the continuous monitoring of their proper operation. For the establishment of the Kubernetes environment, the minikube technology was chosen as it offers easy and fast creation of a Kubernetes environment, combining a long-term presence in the area and a very active community.

Ευχαριστίες

Άδειο

Δημιουργία, διαχείριση και αποθήκευση μηνυμάτων IoT περιβάλλοντος με τη χρήση RabbitMQ Server & ανάπτυξη του full-stack συστήματος σε Kubernetes

Εμμανουήλ Ζήσης-Μήλης
zemmanox@ece.auth.gr

13 Σεπτεμβρίου 2021

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εισαγωγή

Περιγραφή της περιοχής

Μια αναφορά [1]. Some text in english.

1.1 Κίνητρο

Το κίνητρο για την εκπόνηση αυτής της διπλωματικής είναι η αέναη και ισχυρή επιθυμία μου να πάρω πτυχίο.

Το κίνητρο αυτής της διπλωματικής ήταν η αυτοματοποίηση της διαδικασίας διαχείρισης μηνυμάτων προερχόμενων απο συσκευές IoT.

Αναλυτικότερα, τα μηνύματα αυτά μπορεί να περιέχουν μια πληθώρα από μετρήσεις, συμβάντα και πληροφορίες. Επίσης μπορεί να αναφέρονται σε διαφορετικά συστήματα και να μπορούν να κατηγοριοποιηθούν με περισσότερους από έναν τρόπους βάσει πολλαπλών κριτηρίων. Για αυτό το λόγο, ένα τέτοιο σύστημα εμπλουτίζεται σημαντικά με τη συνεισφορά των λειτουργιών επισήμανσης και ομαδοποίησης μηνυμάτων σε λογικά θέματα (topics).

Μία ακόμα διευκόλυνση που προσφέρει το υπό εξέταση σύστημα είναι η δυνατότητά του να διαχωρίζονται των μηνυμάτων σε μηνύματα ενδιαφέροντος και μη και τα επιθυμητά μηνύματα να αποθηκεύονται σε μια ΒΔ αμέσως μετά τη λήψη τους. Αυτός ο διαχωρισμός συγκεκριμένα μπορεί να γίνει με τον απευθείας από τον χρήστη ορισμό των πληροφοριών ενδιαφέροντος που πρέπει να περιέχει κάποιο μήνυμα για να είναι επιθυμητή η αποθήκευσή του.

Προστιθέμενη αξία στη χρήση του συστήματος προσφέρει η παροχή πρόσβασης στους χρήστες σε εργαλεία διαχείρισης και παρακολούθησης των υποσυστημάτων λήψης και διαχείρισης των μηνυμάτων τους. Μέσα από αυτά οι χρήστες μπορούν να βλέπουν την κατάσταση και τον ρυθμό των μηνυμάτων που αποστέλλονται και λαμβάνονται, καθώς και το ποσοστό από αυτά που καλύπτουν τις συνθήκες ενδιαφέροντος και καταλήγουν στη ΒΔ.

Τέλος η δημιουργία ολόκληρου του συστήματος με τη χρήση δοχειοποιημένων εφαρμογών μέσα σε περιβάλλον Kubernetes προσδίδει τρία βασικά προτερήματα. Το πρώτο είναι ότι με τη χρήση των containers το σύστημα μπορεί να αναπτυχθεί σε δια-

φορετικές τοποθεσίες με σχετική ευκολία, καθώς όλες οι σχετικές βιβλιοθήκες και τα εργαλεία που απαιτούνται για τη λειτουργία του συστήματος, εμπεριέχονται στα containers. Ακόμη η χρήση του Kubernetes προσδίδει μια επιπρόσθετη ανθεκτικότητα σε τυχαίες αλλά και μη προβλεπόμενες αποτυχίες των επιμέρους υποσυστημάτων της εφαρμογής. Σαν τελευταίο πλεονέκτημα αναφέρεται η χαρακτηριστική ευκολία με την οποία να γίνει οριζόντια κλιμακοποίηση του συστήματος μέσα σε ένα περιβάλλον Kubernetes. Σε απλοποιημένους όρους, σε συνθήκες υψηλού φόρτου μπορούν να αναπτυχθούν επιπλέον αντίγραφα της εφαρμογής για εξυπηρέτηση των αιτημάτων. Αυτό μπορεί να γίνει είτε με ρητές εντολές του διαχειριστή του συστήματος, είτε εφαρμόζοντας κάποια πολιτική που να υποδεικνύει τα όρια πάνω από τα οποία θα συμβαίνει αυτόματα.

1.2 Περιγραφή προβλήματος

Ο ρυθμός εμφάνισης μηνυμάτων σε μια εφαρμογή στον τομέα του IoT μπορεί να έχει πολλές διακυμάνσεις ή και να φτάνει πολύ υψηλές τιμές. Η δυσκολία της διαχείρισης ενός τέτοιου όγκου επικοινωνίας μέσω των γνωστών πρωτοκόλλων REST / RPC οδηγεί στην ανάγκη για χρήση ενός πρωτοκόλλου που να μπορεί να ανταπεξέλθει σε αυτές τις προκλήσεις. Τα πρωτόκολλα που εφαρμόζουν την μεθοδολογία της διαμεσολάβησης μηνυμάτων έχουν πλεονέκτημα σε αυτές τις συνθήκες, καθώς ένα χαρακτηριστικό τους είναι ότι εδραιώνουν συνδέσεις διαρκείας μεταξύ του διαμοιραστή και των παραγωγών / καταναλωτών των μηνυμάτων. Το πλεονέκτημα προέρχεται από το γεγονός ότι το κόστος διατήρησης μιας σύνδεσης διαρκείας υπό αυτές τις συνθήκες είναι μικρότερο από το κόστος δημιουργίας νέας σύνδεσης σε κάθε εμφάνισης νέου μηνύματος όπως ορίζει η σύγχρονη επικοινωνία.

Τέτοιου είδους συστήματα πολλές φορές πρέπει να λειτουργήσουν κάτω από συνθήκες αναξιόπιστων δικτύων. Αυτό σημαίνει οι δικτυακές συνδέσεις των εξαρτημάτων και των συσκευών μπορεί να διακόπτονται και να επιδιορθώνονται με τυχαία συχνότητα. Η χρήση πρωτοκόλλων σύγχρονης επικοινωνίας με αυτά τα δεδομένα μπορεί να γίνει ιδιαίτερα προβληματική. Η ασύγχρονη επικοινωνία που εφαρμόζουν τα πρωτόκολλα διαμεσολάβησης μηνυμάτων καθιστούν την επικοινωνία αξιόπιστη καθώς τα μηνύματα αποθηκεύονται στον διαμεσολαβητή μηνυμάτων από τους παραγωγούς όταν αυτοί επιτύχουν να συνδεθούν. Ο διαμεσολαβητής στη συνέχεια παραδίδει τα μηνύματα στους κατάλληλους αποδέκτες όταν η δικτυακές συνθήκες το επιτρέψουν. Οι αποστολές μηνυμάτων από τους παραγωγούς είναι δηλαδή πλήρως ανεξάρτητες από τις λήψεις των μηνυμάτων από τους καταναλωτές. Εκτός αυτού μπορούν να εφαρμοστούν και λειτουργίες ανταλλαγής αναγνώρισης λήψης μηνυμάτων, κάτι που συνεισφέρει περαιτέρω στη διασφάλιση καλής λειτουργίας του συστήματος.

Ένα επιπλέον πρόβλημα είναι η ετερογένεια που χαρακτηρίζει συνήθως αυτού του είδους τα συστήματα. Τα υποσυστήματα των παραγωγών των μηνυμάτων συνήθως έχουν αρκετά διαφορετική μορφή, τοπολογία και απαιτήσεις από τα υποσυστήματα των καταναλωτών των μηνυμάτων. Η απεμπλοκή λοιπόν αυτών των δύό συστατικών του συστήματος είναι κρίσιμη ώστε να διευκολύνεται και να απλοποιείται η λειτουργία του.

1.3 Στόχοι της διπλωματικής

ΕΔΩ ΠΡΕΠΕΙ ΝΑ ΠΡΟΣΤΕΘΕΙ ΜΙΑ ΜΙΚΡΗ ΠΑΡΑΓΡΑΦΟΣ ΠΟΥ ΝΑ ΠΕΡΙΓΡΑΦΟ ΤΟΝ ΓΕΝΙΚΟ ΣΤΟΧΟ ΤΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΠΡΙΝ ΠΑΡΟΥΣΙΑΣΩ ΤΑ BULLETS

Τα επιθυμητά αποτελέσματα της διπλωματικής ήταν τα εξής:

- ✓ Να δημιουργηθεί ένα σύστημα που να προσφέρει εργαλεία διαχείρισης μηνυμάτων προερχόμενων απο συσκευές IoT.
- ✓ Οι χρήστες να μπορούν να εγγραφούν στο σύστημα και να διατηρούν στοιχεία για τις εφαρμογές τους.
- ✓ Οι χρήστες να μπορούν να δημιουργήσουν έναν RabbitMQ Server για τους ίδιους και να έχουν πρόσβαση στις υπηρεσίες παρακολούθησής του.
- ✓ Οι χρήστες να μπορούν να δρομολογήσουν στον RabbitMQ Server τους μηνύματα από τις συσκευές IoT τους.
- ✓ Οι χρήστες να μπορούν να δημιουργήσουν καταναλωτές μηνυμάτων, οι οποίοι παρακολουθούν συγκεκριμένα θέματα (topics) του RabbitMQ Server και οι οποίοι παραμένουν ενεργοί μέχρι ο χρήστης να παύσει την λειτουργία τους.
- ✓ Οι χρήστες να μπορούν να εφαρμόσουν συνθήκες διαφόρων μορφών στους καταναλωτές τους, κάτω από τις οποίες τα μηνύματα που αυτοί δέχονται να θεωρούνται μηνύματα ενδιαφέροντος και να αποθηκεύονται σε μία κατάλληλη ΒΔ.
- ✓ Το σύστημα να παρέχει βασικά εργαλεία επισκόπησης στους χρήστες σχετικά με τους ενεργούς καταναλωτές τους και τα μηνύματα τα οποία αυτοί επεξεργάζονται.
- ✓ Να υλοποιηθεί δοχειοποίηση των επιμέρους κομματιών του συστήματος και το σύστημα να αναπτυχθεί μέσα σε περιβάλλον Kubernetes.
- Το σύστημα να μπορεί να υποστηρίξει διαφορετικούς ρόλους χρηστών.

1.4 Διάρθρωση

Η διάρθρωση της παρούσας διπλωματικής εργασίας είναι η εξής:

- Κεφάλαιο 2: Υπόβαθρο [Το θεωρητικό και πρακτικό υπόβαθρο που πρέπει να γνωρίζει ο αναγνώστης για να κατανοήσει το κείμενο]
- Κεφάλαιο 3: Σχετική βιβλιογραφία [Άλλοι τρόποι αντιμετώπισης του προβλήματος έως τώρα]
- Κεφάλαιο 4: Υλοποίηση [Ειδικότεροι τρόποι αντιμετώπισης. Περιγραφή της λειτουργικότητας του συστήματος μου]
- Κεφάλαιο 5: Παρουσίαση του συστήματος [Εδώ θα παρουσιαστούν use cases σε συνδυασμό με UI.]
- Κεφάλαιο 6: Πειράματα & Αποτελέσματα [Μπορεί να σπάσει ανάλογα με το μήκος του κεφαλαίου]
- Κεφάλαιο 7: Συμπεράσματα & μελλοντική εργασία [Μπορεί να σπάσει ανάλογα με το μήκος του κεφαλαίου]

- Κεφάλαιο 8:
- Κεφάλαιο 9:

Κεφάλαιο 2

Υπόβαθρο

Το θεωρητικό και πρακτικό υπόβαθρο που πρέπει να γνωρίζει ο αναγνώστης για να κατανοήσει το κείμενο

2.1 RabbitMQ

Η τεχνολογία γύρω από την οποία αναπτύχθηκε αυτή η διπλωματική είναι το RabbitMQ[1]. Αποτελεί έναν ευρέως διαδεδομένο διαμεσολαβητή μηνυμάτων (message broker) ανοιχτού κώδικα ο οποίος υποστηρίζει διάφορα πρωτόκολλα. Το RabbitMQ μπορεί να αναπτυχθεί σε διανεμημένα συστήματα για να ικανοποιήσει απαιτήσεις μεγάλης κλίμακας και υψηλής διαθεσιμότητας.

Όπως υποδεικνύει αυτή η περιγραφή, ένας διαμεσολαβητής μηνυμάτων τοποθετείται μεταξύ μιας υπηρεσίας παραγωγής μηνυμάτων και μιας υπηρεσίας κατανάλωσής τους. Τα μηνύματα αποστέλλονται στον διαμεσολαβητή όπου αποθηκεύονται σε κατάλληλες δομές που ονομάζονται ουρές (queues) και στη συνέχεια δρομολογούνται στους συνδεδεμένους καταναλωτές ανάλογα καθορισμένους κανόνες.

Παρακάτω θα παρουσιαστούν κάποιες βασικές έννοιες αυτού του διαμεσολαβητή.

2.1.1 Advanced Message Queuing Protocol

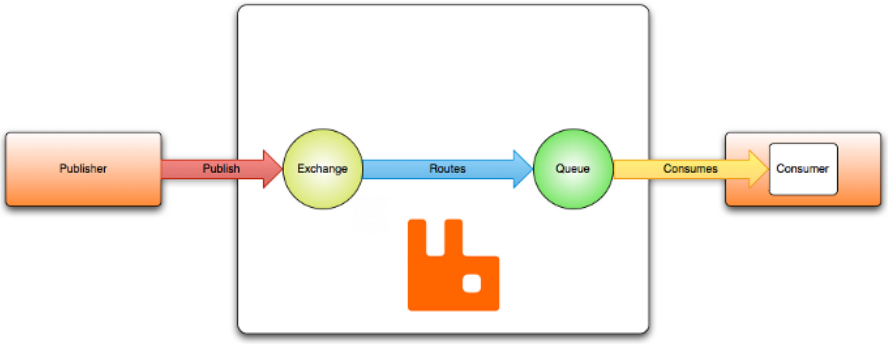
Το AMQP είναι ένα πρωτόκολλο ανταλλαγής μηνυμάτων το οποίο επιτρέπει σε εφαρμογές την αποστολή και λήψη μηνυμάτων μέσω ενός διαμεσολαβητή. Αποτελεί ένα πρωτόκολλο επιπέδου εφαρμογής που χρησιμοποιεί το Transmission Control Protocol (TCP) για αξιόπιστη παράδοση.

Η έκδοση του πρωτοκόλλου που χρησιμοποιήθηκε είναι η 0-9-1 σύμφωνα με την οποία η ανταλλαγή μηνυμάτων λειτουργεί ως εξής:

- Τα μηνύματα δημοσιεύονται σε αρχικούς κόμβους ανταλλαγής (exchanges) του διαμεσολαβητή, η λειτουργία των οποίων συχνά παρομοιάζεται με ένα ταχυδρομείο.

[1] https://www.rabbitmq.com/

- Στη συνέχεια οι αρχικοί κόμβοι διανέμουν αντίγραφα των μηνυμάτων σε ουρές, σύμφωνα με κάποιους κανόνες (bindings) στους οποίους ορίζεται η σύνδεση των κόμβων με τις ουρές.
- Εντέλει ο διαμεσολαβητής είτε παραδίδει τα μηνύματα στους καταναλωτές που έχουν εγγραφεί σε ουρές, είτε οι καταναλωτές λαμβάνουν μηνύματα από ουρές κατόπιν αιτήματος.



Σχήμα 2.1: Παράδειγμα δρομολόγησης μηνυμάτων στο πρωτόκολλο AMQP

2.1.2 Επιβεβαιώσεις λήψης (acknowledgements)

Καθώς τα δίκτυα μπορεί να είναι αναξιόπιστα, ή να συντρέξουν και άλλοι λόγοι για τους οποίους οι εφαρμογές ενδέχεται να αποτύχουν στην επεξεργασία των μηνυμάτων, στο πρωτόκολλο AMQP συμπεριλαμβάνεται μια μέθοδος αντιμετώπισης αυτού του προβλήματος: οι επιβεβαιώσεις λήψης μηνυμάτων.

Όταν αυτές είναι ενεργοποιημένες, ο καταναλωτής στέλνει μια ειδοποίηση στον διαμεσολαβητή μηνυμάτων ότι το μήνυμα έχει ληφθεί και η επεξεργασία του έχει ολοκληρωθεί επιτυχώς. Αυτό μπορεί να συμβεί είτε αυτόματα κατά τη λήψη του μηνύματος από τον καταναλωτή, είτε ρητά όταν τελειώσει η επεξεργασία του.

Στην περίπτωση που οι επιβεβαιώσεις λήψης μηνυμάτων είναι ενεργοποιημένες, τα μηνύματα αφαιρούνται από τις ουρές του διαμεσολαβητή μόνο όταν αυτός δεχθεί την επιβεβαίωση λήψης τους από τον καταναλωτή.

2.1.3 Απόρριψη μηνυμάτων (negative acknowledgements)

Υπάρχει η περίπτωση ένας καταναλωτής να μην είναι σε θέση να επεξεργαστεί κάποιο μήνυμα που έχει λάβει από μια ουρά. Τότε μπορεί να υποδείξη αυτή την κατάσταση στον διαμεσολαβητή απορρίπτοντας το μήνυμα. Τα μηνύματα που απορρίπτονται, είτε θα ξαναεισαχθούν στην ουρά, είτε θα διαγραφούν πλήρως, σύμφωνα με την επιλογή του καταναλωτή. Μεγάλη προσοχή απαιτείται όταν ξαναγίνεται εισαγωγή του μηνύματος στην ουρά ώστε να αποφεύγονται καταστάσεις ατέρμονων βρόγχων παράδοσης απόρριψης και εισαγωγής του ίδιου μηνύματος στην ουρά.

Ο διαμεσολαβητής RabbitMQ προσφέρει μια υλοποίηση αυτής της μεθόδου: τις επιβεβαιώσεις απόρριψης (negative acknowledgements). Αυτές μπορούν να χρησιμοποιηθούν αντί των επιβεβαιώσεων λήψης, έτσι ώστε όλα τα μηνύματα να θεωρούνται

καλώς ληφθέντα εκτός από αυτά για τα οποία έχει σταλεί επιβεβαίωση απόρριψης από τον καταναλωτή στον διαμεσολαβητή.

2.1.4 Κόμβοι ανταλλαγής (exchanges)

Οι κόμβοι ανταλλαγής αποτελούν τον πρώτο σταθμό στον οποίο στέλνονται τα μηνύματα. Αυτοί αναλαμβάνουν να τα προωθήσουν σε ουρές όπως ορίζεται στους κανόνες που τους έχουν δοθεί κατά τη δημιουργία τους. Αυτοί οι κανόνες περιγράφουν με ποιες ουρές συνδέεται ο κάθε κόμβος ανταλλαγής, τι τύπος κόμβου ανταλλαγής είναι και ποιο είναι το κλειδί δρομολόγησης (routing key) της κάθε σύνδεσης ενός κόμβου ανταλλαγής με μια ουρά. Κάθε σύνδεση ενός κόμβου ανταλλαγής με μια ουρά χαρακτηρίζεται από ένα κλειδί δρομολόγησης.

Στο πρωτόκολλο AMQP 0-9-1 ορίζονται 4 διαφορετικοί τύποι κόμβων ανταλλαγής:

- Άμεσος κόμβος (Direct exchange): Ένας άμεσος κόμβος ανταλλαγής προωθεί ένα μήνυμα σε μια συνδεδεμένη ουρά, μόνο αν το κλειδί δρομολόγησης του μηνύματος ταυτίζεται με το κλειδί δρομολόγησης της σύνδεσης του κόμβου ανταλλαγής με την ουρά.
- Κόμβος ευρείας μετάδοσης (Fanout exchange): Όπως δηλώνει και το όνομά του, αυτός ο τύπος κόμβου ανταλλαγής προωθεί αντίγραφα των μηνυμάτων του σε κάθε ουρά με την οποία είναι συνδεδεμένος αγνοώντας το κλειδί δρομολόγησης.
- Κόμβος θεμάτων (Topic exchange): Στους κόμβους ανταλλαγής θεμάτων ορίζεται ένα πρότυπο δρομολόγησης αντί για κλειδί δρομολόγησης σε κάθε σύνδεση του κόμβου με μια ουρά. Τα μηνύματα του κόμβου συνεπώς προωθούνται σε μια ουρά εάν το κλειδί δρομολόγησης των μηνυμάτων ταιριάζει με το πρότυπο δρομολόγησης που έχει οριστεί για τη συγκεκριμένη σύνδεση. Αυτό επιτρέπει τον έλεγχο πολλαπλών κριτηρίων κατά τη δρομολόγηση και για αυτό χρησιμοποιήθηκε αυτός ο τύπος κόμβων ανταλλαγής στο σύστημα που αναπτύχθηκε.
- Κόμβος κεφαλίδων (Headers exchange): Σε αυτή την περίπτωση το κλειδί δρομολόγησης του μηνύματος αγνοείται και στη θέση του ελέγχονται οι παράμετροι κεφαλίδας του κάθε μηνύματος. Σε περίπτωση που ταιριάζουν με το κλειδί δρομολόγησης μιας σύνδεσης κόμβου - ουράς, τότε προωθούνται στην αντίστοιχη ουρά.

Οι κόμβοι ανταλλαγής μπορεί να είναι μόνιμοι (durable) ή παροδικοί (transient). Οι ανθεκτικοί επιβιώνουν από μια επανεκκίνηση του διαμεσολαβητή, ενώ οι παροδικοί όχι (πρέπει να δηλωθούν ξανά όταν ο διαμεσολαβητής επιστρέψει σε λειτουργία).

2.1.5 Ουρές (queues)

Οι ουρές αποτελούν τη βασική δομή βραχυπρόθεσμης αποθήκευσης μηνυμάτων στον διαμεσολαβητή, πριν αυτά παραδοθούν στους καταναλωτές για επεξεργασία. Είναι δομές σειριακών δεδομένων που λειτουργούν με τη μέθοδο FIFO - First In First Out.

Για να χρησιμοποιηθεί μια ουρά πρέπει να έχει προηγηθεί ο ορισμός της. Αν μια ουρά δεν υπάρχει, ο ορισμός της θα οδηγήσει ταυτόχρονα και στη δημιουργία της, ενώ δε θα συμβεί τίποτα σε αντίθετη περίπτωση.

Αντίστοιχα με τους κόμβους ανταλλαγής, και οι ουρές μπορούν να οριστούν ως μόνιμες (durable) ή παροδικές (transient). Στην πρώτη περίπτωση η περιγραφή και τα μεταδεδομένα της ουράς αποθηκεύονται στον δίσκο, ενώ στη δεύτερη περίπτωση αποθηκεύονται στη μνήμη όταν είναι αυτό δυνατό. Όταν η ανθεκτικότητα σε επανεκκινήσεις του διαμεσολαβητή είναι επιθυμητή, τότε πρέπει να ορίζονται εκτός από τις ουρές και τα ίδια τα μηνύματα ως μόνιμα κατά τη δημοσίευσή τους στον κόμβο ανταλλαγής.

2.1.6 Παραγωγοί (producers)

Παραγωγοί είναι οι εφαρμογές οι οποίες συνδέονται σε έναν διαμεσολαβητή μηνυμάτων, αυθεντικοποιούνται, εγκαθιδρύουν μια σύνδεση μέσα στην οποία ανοίγουν ένα κανάλι και τέλος δημοσιεύουν τα παραγώμενα μηνύματά τους σε έναν κόμβο ανταλλαγής.

Οι σύνδεση του παραγωγού με τον διαμεσολαβητή είναι μακράς διαρκείας, δηλαδή εγκαθιδρύεται μια φορά και όλα τα μηνύματα που πρέπει να αποσταλούν στη διάρκεια ζωής του παραγωγού στέλνονται μέσω αυτής της σύνδεσης.

Ένας παραγωγός επιτρέπεται επίσης να λαμβάνει και μηνύματα από τον διαμεσολαβητή, επομένως να λειτουργεί και ως καταναλωτής.

2.1.7 Καταναλωτές (consumers)

Μια εφαρμογή μπορεί να λειτουργήσει ως καταναλωτής στο πλαίσιο του διαμεσολαβητή μηνυμάτων με έναν από τους δύο ακόλουθους τρόπους:

1. Χρησιμοποιώντας τη συνιστώμενη μέθοδο της εγγραφής σε μια ουρά. Με αυτό τον τρόπο, όποτε εισέρχονται νέα μηνύματα στην ουρά, αυτά ωθούνται στην εφαρμογή - καταναλωτή (push API).
2. Ελέγχοντας επανειλημμένα (polling) την ουρά για την ύπαρξη νέων μηνυμάτων σε αυτή (pull API). Αυτή η μέθοδος όμως θεωρείται ιδιαίτερα αναποτελεσματική και στις περισσότερες περιπτώσεις θα πρέπει να αποφεύγεται.

Οι σύνδεση του καταναλωτή με τον διαμεσολαβητή είναι μακράς διαρκείας, δηλαδή εγκαθιδρύεται μια φορά και όλα τα μηνύματα που πρέπει να ληφθούν στη διάρκεια ζωής του καταναλωτή λαμβάνονται μέσω αυτής της σύνδεσης.

Ένας καταναλωτής επιτρέπεται επίσης να στέλνει και μηνύματα στον διαμεσολαβητή, επομένως να λειτουργεί και ως παραγωγώς.

2.1.8 Συνδέσεις και κανάλια

Οι συνδέσεις μεταξύ παραγωγών/καταναλωτών και διαμεσολαβητή στο πρωτόκολλο AMQP 0-9-1 είναι ορισμένες έτσι ώστε να είναι μακράς διαρκείας. Έτσι συνιστάται η εδραίωση μιας σύνδεσης και η χρήση της για όλες της αποστολές και λήψεις μηνυμάτων από μια εφαρμογή, αντί της δημιουργίας μιας σύνδεσης για κάθε ανταλλαγή ενός μηνύματος. Οι συνδέσεις κάνουν χρήση αυθεντικοποίησης και μπορούν να προστατευτούν με τη χρήση του πρωτοκόλλου Transport Layer Security (TLS).

Ορισμένες εφαρμογές χρειάζονται πολλαπλές συνδέσεις με τον διαμεσολαβητή. Ωστόσο, είναι ανεπιθύμητο να διατηρούνται πολλές συνδέσεις TCP ανοικτές ταυτόχρονα, διότι αυτό καταναλώνει πόρους συστήματος και καθιστά δυσκολότερη τη ρύθμιση του

τείχους προστασίας (firewall). Οι συνδέσεις AMQP 0-9-1 είναι πολυπλεγμένες με κανάλια που μπορούν να θεωρηθούν ως ¨ελαφριές συνδέσεις που μοιράζονται μία μόνο σύνδεση TCP¨.

Η επικοινωνία σε ένα συγκεκριμένο κανάλι είναι εντελώς ξεχωριστή από την επικοινωνία σε άλλο κανάλι. Επίσης ένα κανάλι υπάρχει μόνο στο πλαίσιο μιας σύνδεσης και ποτέ από μόνο του. Όταν κλείνει μια σύνδεση, κλείνουν όλα τα κανάλια σε αυτήν.

Για εφαρμογές που χρησιμοποιούν πολλαπλά νήματα/διαδικασίες για επεξεργασία, είναι πολύ συνηθισμένο να ανοίγουν ένα νέο κανάλι ανά νήμα/διαδικασία και να μην μοιράζονται κανάλια μεταξύ τους.

2.1.9 Εικονικοί διαμεσολαβητές (virtual hosts)

Για να καταστεί δυνατό για έναν διαμεσολαβητή να φιλοξενεί πολλαπλά απομονωμένα ¨περιβάλλοντα¨ (ομάδες χρηστών, κόμβοι ανταλλαγής, ουρές και ούτω καθεξής), το AMQP 0-9-1 περιλαμβάνει την έννοια των εικονικών διαμεσολαβητών τλ(ηοστς). Είναι παρόμοια με εικονικούς κεντρικούς υπολογιστές που χρησιμοποιούνται από πολλούς δημοφιλείς διακομιστές ιστού (web servers) και παρέχουν εντελώς απομονωμένα περιβάλλοντα στα οποία ζουν οντότητες AMQP. Οι χρήστες του πρωτοκόλλου καθορίζουν ποιον vhost θέλουν να χρησιμοποιήσουν κατά την εγκαθίδρυση της σύνδεσης.

2.1.10 Αυθεντικοποίηση και εξουσιοδότηση στο RabbitMQ (authentication & authorization)

Αρχικά ένας σαφής διαχωρισμός των δύο αυτών εννοιών κρίνεται απαραίτητος:

- Η αυθεντικοποίηση αναφέρεται στην αναγνώριση της ταυτότητας κάποιου χρήστη.
- Η εξουδιοδότηση περιγράφει τα δικαιώματα και τις δυνατότητες που έχει κάποιος χρήστης πάνω σε συγκεκριμένους πόρους του συστήματος.

Κάθε ενέργεια στον διαμεσολαβητή RabbitMQ γίνεται μετά από μια σύνδεση με αυτόν. Για να συμβεία αυτό, κάποιος χρήστης παρέχει τα διαπιστευτήριά του και αιτείται τη σύνδεσή του με κάποιον εικονικό διαμεσολαβητή, για τον οποίο να έχει τα αντίστοιχα δικαιώματα.

Όταν ο διαμεσολαβητής RabbitMQ εκκινεί για πρώτη φορά, αρχικοποιεί τη ΒΔ του έτσι ώστε να υπάρχει ένας προκαθορισμένος εικονικός διαμεσολαβητής με όνομα ¨/¨ και ένας προκαθορισμένος χρήστης με τα εξής διαπιστευτήρια: όνομα χρήστη ¨guest¨ και κωδικό ¨guest. Αυτός ο χρήστης έχει πλήρη πρόσβαση στον εικονικό διαμεσολαβητή ¨/¨. Για λόγους ασφαλείας συνιστάται να παρέχεται ένας διαφορετικός χρήστης με νέα διαπιστευτήρια στη θέση του προκαθορισμένου κατά τη δημιουργία του διαμεσολαβητή.

Η αυθεντικοποίηση μπορεί να διεκπεραιωθεί είτε με τη χρήση ονόματος χρήστη/κωδικού, είτε με πιστοποιητικά τύπου X.509.

Με την ολοκλήρωση της αυθεντικοποίησης, ο διαμεσολαβητής ελέγχει αν ο επιβεβαιωμένος χρήστης έχει τα απαραίτητα δικαιώματα για να αποκτήσει πρόσβαση στον εικονικό διαμεσολαβητή τον οποίο προσδιόρισε στο αίτημα σύνδεσης. Στη συνέχεια για την εκτέλεση κάθε ενέργειας στον διαμεσολαβητή ελέγχονται τα δικαιώματα του χρήστη

που σχετίζονται με την ενέργεια και τους πόρους του συστήματος πάνω στους οποίους αυτή εκτελείται. Ένας πόρος μπορεί να είναι ένας κόμβος ανταλλαγής ή μια ουρά σε έναν εικονικό διαμεσολαβητή, ενώ τα δικαιώματα που ορίζονται στο RabbitMQ είναι τα εξής:

- configure: δημιουργία και διαγραφή πόρου ή μετατροπή της συμπεριφοράς του
- write: εγγραφή μηνυμάτων στον πόρο
- read: ανάγνωση μηνυμάτων από τον πόρο

2.2 Kubernetes

Συνεχίζοντας με τις τεχνολογίες που χρησιμοποιήθηκαν, ιδιαίτερη σημασία έχει το Kubernetes (K8s). Η λειτουργία του συνιστά την αυτοματοποίηση της ανάπτυξης, της αυξομοίωσης κλίμακας και της διαχείρισης δοχειοποιημένων εφαρμογών. Στην ουσία ομαδοποιεί δοχεία που αποτελούν μια εφαρμογή σε λογικές μονάδες για εύκολη διαχείριση και ανακάλυψη. Το τελικό αποτέλεσμα είναι ένα σύμπλεγμα (cluster) κόμβων και πόρων, στους οποίους εκτελείται η εφαρμογή.

Αυτό το πετυχαίνει συνδυάζοντας ένα πλήθος από υποσυστήματα τα οποία συνεργάζονται, προσφέροντας υπηρεσίες απαραίτητες για τη λειτουργία του K8s. Το επίπεδο ελέγχου (control plane) του K8s συντονίζει όλες τις δραστηριότητες του συμπλέγματος όπως προγραμματισμός εφαρμογών, διατήρηση της επιθυμητής κατάστασης εφαρμογών, κλιμάκωση εφαρμογών και εφαρμογή νέων ενημερώσεων.

Παρακάτω θα παρουσιαστούν βασικά στοιχεία και έννοιες του K8s, τα οποία θα συναντήσει ο αναγνώστης στη συνέχεια της διπλωματικής.

2.2.1 Δοχεία και δοχειοποίηση εφαρμογών (containers & containerized apps)

Με τη δοχειοποίηση εφαρμογών εννοείται η συσκευασία του κώδικα λογισμικού με τις βιβλιοθήκες του λειτουργικού συστήματος και τις εξαρτήσεις που απαιτούνται για την εκτέλεση του κώδικα για τη δημιουργία ενός μόνο ελαφρού εκτελέσιμου - που ονομάζεται container - που λειτουργεί με συνέπεια σε οποιαδήποτε υποδομή.

Η περιγραφή των βημάτων για τη συλλογή και εγκατάσταση των απαραίτητων κομματιών ενός container ονομάζεται εικόνα του container (container image).

Καθώς είναι πιο φορητά και αποδοτικά ως προς τους πόρους από τις εικονικές μηχανές (VM), τα container έχουν γίνει οι κύριες υπολογιστικές μονάδες σύγχρονων εφαρμογών νέφους.

2.2.2 Κόμβοι (nodes)

Ένας κόμβος είναι ένας εικονικός ή ένας φυσικός υπολογιστής που χρησιμεύει ως μηχανή εργασίας σε ένα K8s cluster. Κάθε κόμβος έχει ένα Kubelet, το οποίο είναι μια διεργασία υπεύθυνη για τη διαχείριση του κόμβου και την επικοινωνία με το επίπεδο ελέγχου του K8s. Ο κόμβος θα πρέπει επίσης να διαθέτει εργαλεία για το χειρισμό λειτουργιών των containers, όπως το containerd ή το Docker. Ένα K8s cluster που

δέχεται όλη την κίνηση μιας εφαρμογής ενός πραγματικού περιβάλλοντος σε λειτουργία πρέπει να έχει τουλάχιστον τρεις κόμβους.

Όταν αναπτύσσονται εφαρμογές στο K8s, δίνεται η οδηγία στο επίπεδο ελέγχου να ξεκινήσει τα containers των εφαρμογών. Το επίπεδο ελέγχου προγραμματίζει τα ontainers να λειτουργούν στους κόμβους του cluster. Οι κόμβοι επικοινωνούν με το επίπεδο ελέγχου χρησιμοποιώντας το K8s API, το οποίο προσφέρει πρόσβαση στο επίπεδο ελέγχου.

2.2.3 Minikube

Ένα K8s cluster μπορεί να αναπτυχθεί είτε σε φυσικές είτε σε εικονικές μηχανές. Για μια εισαγωγική ενασχόληση με την ανάπτυξη σε K8s, μπορεί να χρησιμοποιηθεί το Minikube[2]. Το Minikube είναι μια ελαφριά εφαρμογή K8s που αναπτύσσει ένα K8s cluster που περιέχει μόνο έναν κόμβο σε έναν τοπικό υπολογιστή. Αυτό γίνεται είτε με την ανάπτυξη του K8s cluster μέσα σε ένα VM είτε σε ένα container. Το Minikube είναι διαθέσιμο για συστήματα Linux, macOS και Windows.

Η διασύνδεση της γραμμής εντολών του Minikube (Minikube CLI - Command Line Interface) παρέχει βασικές λειτουργίες εκκίνησης για την εκτέλεση εργασιών με ένα K8s cluster, συμπεριλαμβανομένης της έναρξης, διακοπής, εμφάνισης κατάστασης και διαγραφής.

2.2.4 Περιγραφές ανάπτυξης εφαρμογών (deployments)

Μόλις ολοκληρωθεί η εκκίνηση ενός K8s cluster, μπορούν να αναπτυχθούν εφαρμογές σε container πάνω σε αυτό. Για να συμβεί αυτό, πρέπει να δημιουργηθεί μια περιγραφή των επιθυμητών ρυθμίσεων της εφαρμογής για το K8s περιβάλλον. Αυτή η περιγραφή ονομάζεται deployment. Το deployment δίνει οδηγίες στο K8s πώς να δημιουργήσει και να ενημερώσει οντότητες της εφαρμογής. Μόλις δημιουργηθεί ένα deployment, το επίπεδο ελέγχου προγραμματίζει τις οντότητες εφαρμογών που περιλαμβάνονται σε αυτό το deployment να εκτελούνται σε μεμονωμένους κόμβους στο cluster.

Αφού δημιουργηθούν οι οντότητες της εφαρμογής, ένας ελεγκτής των K8s deployments παρακολουθεί συνεχώς αυτές τις οντότητες. Εάν ο κόμβος που φιλοξενεί μια οντότητα βγει εκτός λειτουργίας ή διαγραφεί, ο ελεγκτής αντικαθιστά τη χαμένη οντότητα με μια αντίστοιχη σε έναν άλλο κόμβο του cluster. Αυτό παρέχει έναν μηχανισμό αυτοθεραπείας για την αντιμετώπιση βλαβών ή συντήρησης των μηχανημάτων.

Όταν δημιουργείται ένα deployment, πρέπει να καθορίζονται οι εικόνες των εφαρμογών για τα container και ο αριθμός των αντιγράφων των εφαρμογών που είναι επιθυμητό να δημιουργηθούν. Αυτές οι πληροφορίες μπορούν να αλλαχθούν αργότερα ενημερώνοντας το deployment.

2.2.5 Μονάδες λειτουργικότητας (pods)

Τα pods είναι η μικρότερη αυτόνομη μονάδα στην πλατφόρμα K8s. Κάθε pod αντιπροσωπεύει ένα μέρος του φόρτου εργασίας που εκτελείται στο cluster. Περιέχει μια

[2]https://minikube.sigs.k8s.io/docs/

ομάδα από ένα ή περισσότερα container εφαρμογών και μερικούς κοινούς πόρους για αυτά τα container. Οι πόροι αυτοί μπορεί να περιλαμβάνουν:

- Κοινόχρηστους αποθηκευτικούς χώρους που ονομάζονται Volumes
- Υπηρεσίες δικτύωσης, όπως μια μοναδική διεύθυνση IP στο cluster
- Πληροφορίες σχετικά με τον τρόπο εκτέλεσης κάθε container, όπως η εικόνα του container ή συγκεκριμένες θύρες προς χρήση

Το pod συστεγάζει διαφορετικά container της εφαρμογής που είναι σχετικά στενά συνδεδεμένα. Τα container σε ένα pod μοιράζονται μια διεύθυνση IP και το σύνολο των θυρών ports. Επίσης προγραμματίζονται από το επίπεδο ελέγχου με τον ίδιο ακριβώς τρόπο ώστε να συγχρονίζονται και να εκτελούνται πάντα σε κοινό πλαίσιο στον ίδιο κόμβο.

Όταν δημιουργείται ένα K8s deployment, το επίπεδο ελέγχου δημιουργεί pods με containers μέσα τους (σε αντίθεση με τη δημιουργία απευθείας containers). Αυτό σημαίνει ότι δεν υπάρχει η δυνατότητα να δημιουργηθεί απευθείας κάποιο container, αντιθέτως πρέπει να έχει περιγραφεί ως μέλος ενός pod το οποίο ανήκει σε κάποιο deployment. Κάθε pod συνδέεται με τον κόμβο όπου έχει προγραμματιστεί και παραμένει εκεί μέχρι τον τερματισμό (σύμφωνα με την πολιτική επανεκκίνησης) ή τη διαγραφή. Σε περίπτωση αποτυχίας του κόμβου, τα ίδια pods προγραμματίζονται σε άλλους διαθέσιμους κόμβους στο cluster.

2.2.6 Υπηρεσίες (services)

Μια Υπηρεσία στο K8s ομαδοποιεί ένα λογικό σύνολο από Pods και προσδιορίζει κάποιο καθορισμένο τρόπο με τον οποίο παρέχεται πρόσβαση σε αυτά.

Παρόλο που κάθε pod έχει μια μοναδική διεύθυνση IP, αυτές οι διευθύνσεις δεν εκτίθενται εκτός του cluster χωρίς την εφαρμογή μιας Υπηρεσίας. Οι Υπηρεσίες επιτρέπουν στις εφαρμογές να λαμβάνουν επισκεψιμότητα από τοποθεσίες εντός και εκτός του cluster. Οι τρόποι με τους οποίους μια Υπηρεσία παρέχει πρόσβαση στα σχετιζόμενα με αυτήν pods είναι οι εξής:

- ClusterIP (προεπιλογή) - Εκθέτει την Υπηρεσία σε μια εσωτερική cluster IP. Αυτός ο τύπος καθιστά την Υπηρεσία προσβάσιμη μόνο από το cluster.
- NodePort - Εκθέτει την Υπηρεσία στην ίδια θύρα κάθε επιλεγμένου κόμβου στο cluster χρησιμοποιώντας NAT. Κάνει μια υπηρεσία προσβάσιμη εκτός του συμπλέγματος χρησιμοποιώντας το <NodeIP>:<NodePort>. Υπερσύνολο του ClusterIP.
- LoadBalancer - Δημιουργεί έναν εξωτερικό εξισορροπητή φορτίου στο τρέχον cloud (εάν υποστηρίζεται) και εκχωρεί μια σταθερή, εξωτερική IP στην Υπηρεσία. Υπερσύνολο του NodePort.
- ExternalName - Χαρτογραφεί την Υπηρεσία στα περιεχόμενα του πεδίου externalName (π.χ. foo.bar.example.com), επιστρέφοντας μια εγγραφή CNAME με την τιμή της. Δεν εγκαθιδρύεται κανενός είδους proxying.

Η χρησιμότητα των Υπηρεσιών αναδεικνύεται με τη δυνατότητα που προσφέρει στα pods να καταστρέφονται και να επαναδημιουργούνται στο K8s χωρίς να επηρεάζεται η

λειτουργία της εφαρμογής.

2.2.7 Διεχείριση αποθήκευσης (volumes & persistent volumes)

Η διαχείριση αποθηκευτικού χώρου είναι ένα διαφορετικό πρόβλημα από τη διαχείριση υπολογιστικών πόρων.

Τα αρχεία στο δίσκο σε ένα container είναι εφήμερα, γεγονός που παρουσιάζει δύο βασικά προβλήματα. Ένα πρόβλημα είναι η απώλεια αρχείων σε περίπτωση αποτυχίας ενός container. Το container επανεκκινείται αλλά με μηδενικά δεδομένα. Ένα δεύτερο πρόβλημα παρουσιάζεται κατά την κοινή χρήση αρχείων μεταξύ container που λειτουργούν μαζί σε ένα pod. Το τλK8ς δίνει λύση σε αυτά τα προβλήματα με τη χρήση τόμων αποθήκευσης (volumes).

Το K8s υποστηρίζει πολλούς τύπους τόμων αποθήκευσης. Ένα pod μπορεί να χρησιμοποιήσει ταυτόχρονα διαφορετικούς τύπους τόμων αποθήκευσης. Οι τύποι εφήμερων τόμων μοιράζονται τη διάρκεια ζωής ενός pod, αλλά υπάρχουν και τόμοι διατήρησης (persistent volumes - PV) οι οποίοι διατηρούνται ανεξάρτητα από τη διάρκεια ζωής ενός pod. Όταν ένα pod παύει να υπάρχει, το K8s καταστρέφει τους εφήμερους τόμους, όμως όχι και τα persistent volumes. Για οποιοδήποτε είδος τόμου σε ένα δεδομένο pod, τα δεδομένα διατηρούνται σε κάθε επανεκκίνηση κάποιου container.

Για την χρήση των persistent volumes απαιτείται η καταχώρηση ενός αιτήματος κατανάλωσης αυτών των πόρων αποθήκευσης από κάποιον χρήστη. Αυτά τα αιτήματα κατανάλωσης αποθηκευτικού χώρου ονομάζονται persistent volume claims (PVC). Τα PVCs μπορούν να ζητήσουν συγκεκριμένο μέγεθος αποθηκευτικού χώρου και τρόπους πρόσβασης σε αυτόν.

2.2.8 Διαχείριση δεδομένων ρυθμίσεων (configMaps & secrets)

Για τη διευκόλυνση της διαχείρισης δεδομένων ρυθμίσεων ενός συστήματος, το K8s προσφέρει δύο χρήσιμα εργαλεία: τα configMaps και τα secrets.

Τα configMaps παρέχουν έναν τρόπο εισαγωγής δεδομένων ρυθμίσεων σε pods. Είναι αντικείμενα που χρησιμοποιούνται για την αποθήκευση μη εμπιστευτικών δεδομένων σε ζεύγη κλειδιών-τιμών. Τα pods μπορούν να καταναλώνουν configMaps ως μεταβλητές περιβάλλοντος, ορίσματα γραμμής εντολών ή ως αρχεία διαμόρφωσης σε έναν τόμο αποθήκευσης.

Τα configMaps επιτρέπουν την αποσύνδεση των παραμέτρων που σχετίζονται με το περιβάλλον από τα container images, έτσι ώστε οι εφαρμογές να είναι εύκολα φορητές.

Δεν έχουν σχεδιαστεί για να περιέχουν μεγάλα κομμάτια δεδομένων. Τα δεδομένα που αποθηκεύονται σε ένα configMap δεν μπορούν να υπερβαίνουν το 1 MiB. Εάν πρέπει να αποθηκευτούν ρυθμίσεις που είναι μεγαλύτερες από αυτό το όριο, ίσως χρειαστεί να χρησιμοποιηθούν τόμοι αποθήκευσης ή μια ξεχωριστή βάση δεδομένων ή υπηρεσία αρχείων.

Επίσης δεν παρέχουν μυστικότητα ή κρυπτογράφηση. Εάν τα δεδομένα που πρέπει να αποθηκευτούν είναι εμπιστευτικά, συνιστάται η χρήση ενός secret και όχι ενός

configMap ή η χρήση επιπλέον εργαλείων (τρίτων) για να διατηρηθεί η ιδιωτικώτητα των δεδομένων.

Τα secrets είναι παρόμοια με τα configMaps αλλά προορίζονται ειδικά για τη αποθήκευση εμπιστευτικών δεδομένων.

Επειδή τα secrets μπορούν να δημιουργηθούν ανεξάρτητα από τα pods που τα χρησιμοποιούν, υπάρχει μικρότερος κίνδυνος να εκτίθεται το secret (και τα δεδομένα του) κατά τη ροή εργασιών δημιουργίας, προβολής και επεξεργασίας pods. Το K8s και οι εφαρμογές που εκτελούνται στο cluster, μπορούν επίσης να λάβουν πρόσθετες προφυλάξεις με τα secrets, όπως η αποφυγή εγγραφής εμπιστευτικών δεδομένων σε μη πτητικό αποθηκευτικό χώρο.

Τα secrets αποθηκεύονται, από προεπιλογή, χωρίς κρυπτογράφηση στη ΒΔ του K8s API server (etcd). Οποιοσδήποτε έχει πρόσβαση στο API μπορεί να ανακτήσει ή να τροποποιήσει ένα secret, και το ίδιο μπορεί να κάνει και οποιοσδήποτε έχει πρόσβαση στη ΒΔ etcd. Επιπλέον, οποιοσδήποτε εχει εξουσιοδότηση για τη δημιουργία pods μπορεί να χρησιμοποιήσει αυτήν την πρόσβαση για να διαβάσει οποιοδήποτε secret. Αυτό περιλαμβάνει την έμμεση πρόσβαση, όπως η εξουσιοδότηση δημιουργίας ενός deployment (κάτι το οποίο με τη σειρά του θα δημιουργήσει pods.

Για την ασφαλή χρήση των secrets, τα ακόλουθα βήματα είναι κρίσιμα:

- Ενεργοποίηση της κρυπτογράφησης των secrets κατά την αποθήκευσή τους στη ΒΔ etcd.
- Ενεργοποίηση ή διαμόρφωση κανόνων Role-based Access Control (RBAC) που να περιορίζουν την ανάγνωση δεδομένων στα secrets (συμπεριλαμβανομένης της έμμεσης πρόσβασης).
- Όπου ενδείκνυται, να γίνει περιορισμός μέσω RBAC ως προς το ποιοι χρήστες επιτρέπεται να δημιουργούν νέα secrets ή να αντικαθιστούν υπάρχοντα.

2.3 Frontend

2.3.1 react.js

2.3.2 redux

2.4 Backend

2.4.1 express.js

2.5 Database

2.5.1 MongoDB

2.5.2 noSQL

2.5.3 Schema

2.5.4 Model

2.5.5 Authn Authz

Μια αναφορά [1]. Some text in english.

Κεφάλαιο 3

Σχετική βιβλιογραφία

Άλλοι τρόποι αντιμετώπισης του προβλήματος έως τώρα

Μια αναφορά [1]. Some text in english.

3.1 Κίνητρο

Άδειο

3.1.1 Ακόμα ένας τίτλος

Κεφάλαιο 4

Υλοποίηση

- Ειδικότεροι τρόποι αντιμετώπισης.

- Περιγραφή της λειτουργικότητας του συστήματος

4.1 Σφαιρική εικόνα του συστήματος

Για την ικανοποίηση των στόχων της διπλωματικής και με γνώμονα τα κίνητρα της, σχεδιάστηκε η αρχιτεκτονική του συτήματος.

Το αποτέλεσμα είναι μια πλατφόρμα, η οποία δίνει τη δυνατότητα στους χρήστες της να εγγράφονται, να αποκτούν πρόσβαση σε ένα προσωπικό server διαμεσολάβησης μηνυμάτων, να δρομολογούν μηνύματα σε αυτόν, να εφαρμόζουν φίλτρα παρακολούθησης των εισερχόμενων μηνυμάτων, να αποθηκεύουν τα μηνύματα ενδιαφέροντος σε μία ΒΔ και να βλέπουν την τρέχουσα κατάσταση των λειτουργιών που εκτελούνται στο σύστημά τους.

Η τεχνολογία που επιλέχθηκε για τον διαμεσολαβητή μηνυμάτων είναι το RabbitMQ. Αυτή η επιλογή έγινε επειδή το RabbitMQ είναι ένα έργο ανοιχτού κώδικα, ευρέως διαδεδομένο - άρα έχει μια μεγάλη ενεργή κοινότητα προς αντιμετώπιση προβλημάτων, υποστηρίζει διάφορα πρωτόκολλα, παρέχει πολλά εργαλεία σε μια μεγάλη ποικιλία γλωσσών προγραμματισμού και μπορεί να ικανοποιήσει απαιτήσεις μεγάλης κλίμακας και υψηλής διαθεσιμότητας.

Για την ανάπτυξη όλου του απαραίτητου κώδικα της πλατφόρμας, χρησιμοποιήθηκε η γλώσσα προγραμματισμού Javascript στο περιβάλλον Node.js runtime environment. Η καταλληλότητα της Javascript για τον προγραμματισμό διεπαφών σε Web Browser και η ομοιογένεια που προσφέρει η χρήση μίας μοναδικής γλώσσας προγραμματισμού σε όλο το σύστημα, συντέλεσαν σε αυτήν την επιλογή.

Τέλος, ολόκληρο το σύστημα εντάχθηκε μέσα σε περιβάλλον Kubernetes. Έτσι, τα παραγόμενα επιμέρους κομμάτια του συτήματος μετατράπηκαν σε αυτόνομες containerized εφαρμογές. Η δομή της κάθε επιμέρους εφαρμογής και ο τρόπος με τον οποίο εντάσσεται στο K8s περιβάλλον περιγράφονται στα αντίστοιχα αρχεία τύπου yaml. Στο K8s ανατέθηκε η διαχείριση και ενορχήστρωση των containers αυτών, όπως επίσης και των επικοινωνιών μεταξύ τους.

Για την υλοποίηση των προαναφερθέντων λειτουργιών, αναπτύχθηκαν οι λογικές οντότητες που θα αναλυθούν περισσότερο στη συνέχεια:

- Backend
- Frontend
- Βάση Δεδομένων
- Διαμεσολαβητής μηνυμάτων - RabbitMQ Server
- Φίλτρα - Καταναλωτές

4.2 Διάταξη της υλοποίησης και διαθέσιμοι πόροι

ΕΔΩ ΘΑ ΠΕΡΙΓΡΑΨΩ ΤΑ ΜΗΧΑΝΗΜΑΤΑ ΜΟΥ ΤΙ ΣΟΦΤΓΟΥΕΡ ΚΑΙ ΤΙ ΧΑΡΝΤΓΟΥΕΡ ΕΧΩ

4.3 Παρουσίαση των οντοτήτων του συστήματος

Παρακάτω θα εξηγηθεί ο ρόλος της κάθε οντότητας, μαζί με τις λειτουργίες που υλοποιεί και τις λεπτομέρειες της υλοποίησής της.

4.3.1 Backend

Το Backend αποτελεί τον κύριο πυλώνα του συστήματος, καθώς είναι υπεύθυνο για τη διαχείριση όλων των δράσεων που πρέπει να πραγματοποιηθούν.

Η βασικότερή του λειτουργία είναι να δέχεται αιτήματα από τις άλλες εφαρμογές του συστήματος, να τα επεξεργάζεται, να εκτελεί τις απαιτούμενες ενέργειες και να στέλνει απαντήσεις στις εφαρμογές που τα δημιούργησαν σχετικά με την εξέλιξη των αιτημάτων.

Η ανάπτυξη του Backend βασίστηκε στο Express.js[1] framework, το οποίο παρέχει ένα ισχυρό σύνολο δυνατοτήτων για εφαρμογές ιστού και εφαρμογές κινητών συσκευών. Μέσω αυτού διευκολύνεται η δρομολόγηση των αιτημάτων που δέχεται το Backend και η εφαρμογή εργαλείων και ενδιάμεσων λογισμικών (middlewares) στα αιτήματα πριν αρχίσει η κυρίως επεξεργασία τους.

Το Backend προσφέρει μια πληθώρα από Διεπαφές Προγραμματισμού Εφαρμογών - Application Programming Interfaces (API) για όλες τις λειτουργίες που μπορεί να πραγματοποιήσει.

Μέσω των Backend API, μπορούν να εξυπηρετηθούν αιτήματα σχετικά με τη δημιουργία / διαγραφή RMQ Servers, τη δημιουργία / διαγραφή φίλτρων - καταναλωτών μηνυμάτων, την αποστολή περιοδικών στατιστικών της λειτουργίας του συστήματος προς τη διεπαφή του χρήστη, τη σύνδεση / αποσύνδεση των χρηστών από το σύστημα και την αυθεντικοποίηση των χρηστών.

[1] https://expressjs.com/

Εγγραφή: Κατά την εγγραφή χρηστών στο σύστημα δημιουργείται ένα αίτημα από τη διεπαφή του χρήστη το οποίο περιλαμβάνει τα απαραίτητα στοιχεία εγγραφής. Το Backend αφού επιβεβαιώσει τη σωστή μορφή των στοιχείων αυτών, τα αποθηκεύει στη ΒΔ σε κατάλληλη δομή. (Όλες οι επικοινωνίες του Backend με τη ΒΔ γίνεται μέσω της βιβλιοθήκης Mongoose[2].)

Αυθεντικοποίηση: Η αυθεντικοποίηση των χρηστών υλοποιήθηκε στο Backend με τη χρήση session cookies. Αυτή η μέθοδος λειτουργεί ως εξής:

1. Δημιουργείται ένα αίτημα αυθεντικοποίησης μέσω της διεπαφής του χρήστη το οποίο περιλαμβάνει τα διαπιστευτήριά του.
2. Το Backend δέχεται αυτό το αίτημα και ελέγχει αν στη ΒΔ υπάρχει αποθηκευμένος κάποιος χρήστης με τα συγκεκριμένα διαπιστευτήρια. (Η σύγκριση των κωδικών πρόσβασης γίνεται μέσω της βιβλιοθήκης bcrypt.js[3].)
3. Εάν τα διαπιστευτήρια που δόθηκαν ήταν σωστά, τότε εισάγεται σε κατάλληλο πεδίο του session cookie ένα αναγνωριστικό του χρήστη. Σε αντίθετη περίπτωση το αίτημα απορρίπτεται.
4. Τέλος κάθε αίτημα εισερχόμενο στο Backend που απαιτεί αυθεντικοποίηση του χρήστη, πρώτα περνάει από μηχανισμό επιβεβαίωσης ότι το session cookie περιλαμβάνεται στο αίτημα και το αναγνωριστικό του χρήστη είναι έγκυρο. (Τα session cookies δημιουργούνται με χρήση της βιβλιοθήκης node-client-sessions[4].)

Διαχείριση RMQ Servers: Για την επεξεργασία αιτημάτων δημιουργίας / διαγραφής RMQ Servers, επιστρατεύθηκε η βιβλιοθήκη @kubernetes/client-node[5].

Όταν ένα τέτοιο αίτημα καταφτάνει στο Backend, επαληθεύονται τα στοιχεία του χρήστη και στη συνέχεια καλείται το K8s API μέσω του client που υλοποιεί η βιβλιοθήκη ώστε να αναπτυχθεί ένα νέο στιγμιότυπο της εφαρμογής του RMQ Server σε μορφή container στο περιβάλλον του K8s (ή να διαγραφεί ένα προϋπάρχον στιγμιότυπο στην αντίστοιχη περίπτωση).

Στη συνέχεια δημιουργούνται μέσω του ίδιου εργαλείου άλλα απαραίτητα στοιχεία για τη λειτουργία του διαμεσολαβητή μηνυμάτων (όπως είναι τα K8s Services που θα αναφερθούν παρακάτω) και τέλος ανανεώνεται η καταγραφή του χρήστη στη ΒΔ ώστε να περιλαμβάνει (ή να μην περιλαμβάνει σε περίπτωση διαγραφής) τα στοιχεία του διαμεσολαβητή.

Διαχείριση Φίλτρων - Καταναλωτών μηνυμάτων: Αντίστοιχα με την προηγούμενη λειτουργικότητα εξυπηρετούνται και τα αιτήματα δημιουργίας / διαγραφής φίλτρων - καταναλωτών μηνυμάτων. Επαληθεύονται τα στοιχεία του χρήστη και καλείται το ίδιο εργαλείο με σκοπό την ανάπτυξη ενός στιγμιοτύπου της εφαρμογής Φίλτρου. Τέλος ενημερώνεται η καταγραφή του χρήστη στη ΒΔ για να περιλαμβάνει τα νέα δεδομένα.

[2]https://mongoosejs.com/
[3]https://www.npmjs.com/package/bcryptjs
[4]https://github.com/mozilla/node-client-sessions
[5]https://www.npmjs.com/package/@kubernetes/client-node

Περιοδικά στατιστικά: Το Backend μπορεί να προσφέρει την άμεση και περιοδική μετάδοση στατιστικών δεδομένων σχετικών με τη λειτουργία του συστήματος.

Για να γίνει αυτό, ξεκινάει μια επικοινωνία με ένα από τα στιγμιότυπα Φίλτρου - Καταναλωτή μηνυμάτων του χρήστη και του αναθέτει την ανάκτηση χρήσιμων δεδομένων από τον RMQ Server. Όταν το Φίλτρο - Καταναλωτής απαντήσει με τα δεδομένα, το Backend τα στέλνει πίσω στη διεπαφή του χρήστη.

Αυτή η ανάθεση γίνεται αφενός για την αποφυγή αποθήκευσης στο Backend των διαπιστευτηρίων του χρήστη που απαιτούνται για τη σύνδεση με τον RMQ Server και αφετέρου για την αποφόρτωση του Backend από πρόσθετη λειτουργικότητα.

Η περιοδική αποστολή των δεδομένων προς τη διεπαφή του χρήστη πραγματοποιείται με τη χρήση της μεθόδου Server Sent Events (SSE). Αυτή η μέδοθος περιγράφει μια ενσωματωμένη κλάση μέσω της οποίας εγκαθιδρύεται μια σύνδεση ανάμεσα στη διεπαφή χρήστη και τον διακομιστή και επιτρέπεται η λήψη συμβάντων από αυτόν.

Βιβλιοθήκες / εργαλεία

- express: 4.17.1 - Διαχείριση αιτημάτων API
- @kubernetes/client-node: 0.14.3 - Χρήση του K8s API
- bcryptjs: 2.4.3 - Κρυπτογράφηση διαπιστευτηρίων χρήστη
- client-sessions: 0.8.0 - Διαχείριση session cookies
- mongoose: 5.11.19 - Επικοινωνία με τη ΒΔ
- validator: 13.5.2 - Επαλήθευση σωστής μορφής εισαγώμενων δεδομένων από τον χρήστη
- cors: 2.8.5 - Επιτρέπει τη φόρτωση πόρων στο πρόγραμμα περιήγησης από προέλευση διαφορετική του διακομιστή του αιτήματος
- csurf: 1.11.0 - Προσφέρει προστασία ενάντια σε επιθέσεις τύπου Cross-Site Request Forgery (CSRF)
- helmet: 4.4.1 - Προσφέρει προστασία προσθέτοντας συγκεκριμένα HTTP Headers
- axios: 0.21.1 - Δημιουργία αιτημάτων HTTP

4.3.2 Frontend

Ο ρόλος του Frontend είναι να υλοποιεί τη διεπαφή του χρήστη με το σύστημα (User Interface - UI). Αυτό σημαίνει την παρουσίαση ενός περιβάλλοντος φιλικού προς το χρήστη, το οποίο να εμφανίζει με τρόπο κατανοητό τις πληροφορίες που του διατίθενται και να παρέχει εργαλεία αξιοποίησης όλων των λειτουργιών που προσφέρονται από το σύστημα.

Καθώς είναι το κομμάτι του συστήματος που εκτελείται στην πλευρά του χρήστη και συγκεκριμένα στον Browser του, δεν έχει απευθείας πρόσβαση σε καμία πληροφορία του συστήματος. Για οτιδήποτε χρειάζεται να παρουσιαστεί στον χρήστη, δημιουργείται ένα κατάλληλο αίτημα προς το Backend το οποίο αναλαμβάνει να του επιστρέψει τις αντίστοιχες πληροφορίες.

Η ανάπτυξη του Frontend βασίστηκε στη βιβλιοθήκη React[6], η οποία διευκολύνει τη δημιουργία διαδραστικών διεπαφών χρήστη.

Οι δυνατότητες που προσφέρει το Frontend στον χρήστη είναι οι αντίστοιχες λειτουργίες που παρουσιάστηκαν στο Backend. Επιγραμματικά περιλαμβάνουν την εγγραφή και σύνδεση / αποσύνδεση χρηστών, τη δημιουργία / διαγραφή RMQ Servers, την πρόσβαση στο γραφικό περιβάλλον διαχείρισης των RMQ Servers, την εφαρμογή / διαγραφή Φίλτρων - Καταναλωτών μηνυμάτων και την παρουσίαση περιοδικών στατιστικών σχετικά με τη λειτουργία του συστήματος.

Δημιουργία / Διαγραφή διαμεσολαβητή μηνυμάτων

Όπως αναφέρθηκε στην ανάλυση του Backend, κατά την εγγραφή του χρήστη στο σύστημα δημιουργείται ένας νέος RMQ Server για αυτόν. Ο χρήστης μπορεί από την καρτέλα Message Broker της διεπαφής να τον διαγράψει για διατήρηση πόρων σε περίπτωση που δεν επιθυμεί την διακίνηση μηνυμάτων άμεσα. Αφού προβεί στη διαγραφή του μπορεί πάλι να δημιουργήσει έναν άλλο RMQ Server οποιαδήποτε στιγμή από την ίδια καρτέλα.

Είναι σημαντικό να αναφερθεί ότι ο χρήστης δεν μπορεί να διαγράψει τον διαμεσολαβητή μηνυμάτων του αν υπάρχουν ενεργά Φίλτρα εκείνη τη στιγμή. Σε εκείνη την περίπτωση πρώτα πρέπει να διαγραφούν τα Φίλτρα και μετά ο διαμεσολαβητής. Αυτό συμβαίνει γιατί αλλιώς τα ενεργά Φίλτρα θα προσπαθούσαν να συνδεθούν με έναν ανύπαρκτο διαμεσολαβητή, κάτι το οποίο θα οδηγούσε σε άτακτη αποτυχία τους.

Εφαρμογή / Διαγραφή Φίλτρων - Καταναλωτών Μηνυμάτων

Ο χρήστης μπορεί να δημιουργήσει ένα προσαρμόσιμο Φίλτρο της προτίμησής του, το οποίο θα ακούει μηνύματα συγκεκριμένων θεμάτων, κάτω από συγκεκριμένες συνθήκες θα χαρακτηρίζει τα μηνύματα ως μηνύματα ενδιαφέροντος και θα τα αποθηκεύει στη ΒΔ.

Επομένως η διεπαφή χρήστη του επιτρέπει να ορίσει το όνομα της ουράς στην οποία θα ακούει το Φίλτρο του. Ομοίως ορίζει το όνομα του RMQ exchange στο οποίο θα συνδεθεί η ουρά αυτή. Προσδιορίζεται το κλειδί σύνδεσης μεταξή της ουράς και του exchange.

Στη συνέχεια ο χρήστης μπορεί να δώσει ένα πλήθος από συνθήκες, αν οποιαδήποτε από τις οποίες καλύπτεται από κάποιο μήνυμα, τότε αυτό κρίνεται ως σημαντικό και αποθηκεύεται από το Φίλτρο στη ΒΔ. Η κάθε συνθήκη αποτελείται από τρία ξεχωριστά μέρη:

α) το όνομα μιας μεταβλητής που πρέπει να βρίσκεται στα περιεχόμενα του μηνύματος

β) την τιμή της μεταβλητής αυτής και

γ) τον τελεστή που προσδιορίζει την σχέση που πρέπει να ισχύει μεταξύ της τιμής που δόθηκε και της τιμής που βρίσκεται στα περιεχόμενα του μηνύματος ώστε αυτό να καταγραφεί στη ΒΔ. (Δυνατές τιμές του τελεστή είναι οι ακόλουθες: >, <, =, >=, <=, !=)

[6]https://reactjs.org/

Κάθε ενεργό Φίλτρο - Καταναλωτής Μηνυμάτων μπορεί να διαγραφεί οποιαδήποτε στιγμή.

Παρουσίαση περιοδικών στατιστικών στοιχείων

Στην καρτέλα Overview δίνεται η δυνατότητα να εκκινηθεί ο μηχανισμός των Server Sent Events ώστε να παρουσιάζονται στατιστικά στοιχεία της πιο πρόσφατης εικόνας της διακίνησης μηνυμάτων στο σύστημα του χρήστη.

Το χρονικό παράθυρο για το οποίο θα αναζητηθούν τα στατιστικά είναι επιλέξιμο με τιμές από 1 λεπτό μέχρι και 24 ώρες. Επιλέξιμος είναι και ο χρόνος μεταξύ των δειγμάτων, με τιμές από 15 δευτερόλεπτα μέχρι και 1 ώρα[7].

Παραγωγή μηνυμάτων μέσω του Frontend

ΑΥΤΗ Η ΕΝΟΤΗΤΑ ΜΠΟΡΕΙ ΝΑ ΠΑΡΑΛΗΦΘΕΙ ΚΑΘΩΣ ΤΟ ΕΡΓΑΛΕΙΟ ΔΗΜΙΟΥΡΓΗΘΗΚΕ ΓΙΑ ΔΟΚΙΜΕΣ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ. ΑΝΤΙΣΤΟΙΧΑ ΑΝ ΥΠΑΡΧΕΙ ΑΝΑΓΚΗ ΓΙΑ ΠΕΡΙΣΣΟΤΕΡΗ ΕΚΤΑΣΗ ΜΠΟΡΕΙ ΝΑ ΣΥΜΠΛΗΡΩΘΕΙ.

———

Αξιοσημείωτα εργαλεία

Redux: Όσο αυξάνεται ο αριθμός των σελιδών και των κομματιών μιας διεπαφής χρήστη, τόσο μεγαλώνει και η πολυπλοκότητα διασύνδεσης όλων των μερών μεταξύ τους. Κάτι τέτοιο είναι ιδιαίτερα εμφανές όταν απαιτείται κάποιες πληροφορίες που παρήχθησαν σε ένα σημείο να παρουσιαστούν ή να χρησιμοποιηθούν κάπου αλλού.

Το εργαλείο Redux λύνει αυτό ακριβώς το πρόβλημα. Καθιστά δυνατή τη διατήρηση σημαντικών δεδομένων σε ένα κεντρικό σημείο και προσφέρει μεθόδους για την πρόσβαση σε αυτά και την επεξεργασία τους.

React-Router-Dom: Η δρομολόγηση (Routing) είναι η ικανότητα εμφάνισης διαφορετικών σελίδων στον χρήστη. Αυτό σημαίνει ότι ο χρήστης μπορεί να μετακινηθεί μεταξύ διαφορετικών τμημάτων μιας εφαρμογής εισάγοντας μια διεύθυνση URL (Uniform Resource Locator) ή κάνοντας κλικ σε κάποιο στοιχείο.

Το React δεν παρέχει ενσωματωμένες δυνατότητες δρομολόγησης, κάτι το οποίο αντιμετωπίστηκε με τη χρήστη της βιβλιοθήκης React-Router-Dom. Έτσι πήρε μορφή η τελική αρχιτεκτονική της διεπαφής του χρήστη, κατά την οποία η δρομολόγηση του χρήστη στις σελίδες που παρέχουν τα εργαλεία λειτουργικότητας απαιτούν την αυθεντικοποίησή του με τη σύνδεσή του στο σύστημα.

Βιβλιοθήκες / εργαλεία

- react: 17.0.1 - Δημιουργία διαδραστικών διεπαφών χρήστη

[7] Μια σημαντική λεπτομέρεια είναι ότι για τη σωστή παραγωγή των στατιστικών στοιχείων, το χρονικό παράθυρο πρέπει να είναι τουλάχιστον 4 φορές μεγαλύτερο από το διάστημα δειγματοληψίας.

- react-dom: 17.0.1 - Πακέτο που λειτουργεί συμπληρωματικά με το React
- react-router-dom: 5.2.0 - Προσθήκη δυνατοτήτων δρομολόγησης
- redux: 4.0.5 - Κεντρική διαχείριση δεδομένων κατάστασης
- react-redux: 7.2.2 - Πακέτο που λειτουργεί συμπληρωματικά με το Redux
- redux-thunk: 2.3.0 - Πακέτο που λειτουργεί συμπληρωματικά με το Redux
- styled-components: 5.3.0 - Διευκόλυνση χρήσης Cascading Style Sheet (CSS) κώδικα σε γραφικά στοιχεία
- axios: 0.21.1 - Δημιουργία αιτημάτων HTTP
- classnames: 2.2.6 - Υπό όρους ένωση κλάσεων διαμόρφωσης στοιχείων

4.3.3 Βάση Δεδομένων

Η Βάση Δεδομένων είναι το σημείο που αποθηκεύονται όλα τα δεδομένα του συστήματος τα οποία πρέπει να διατηρούνται ανεξάρτητα από τη λειτουργία του.

Για την υλοποίησή της επιλέχθηκε η MongoDB[8]. Η MongoDB είναι μια ΒΔ εγγράφων που έχει σχεδιαστεί με κύριο στόχο την ευκολία ανάπτυξης και κλιμάκωσης.

Μια εγγραφή στη MongoDB είναι ένα έγγραφο, το οποίο είναι μια δομή δεδομένων που αποτελείται από ζεύγη πεδίων και τιμών. Τα έγγραφα MongoDB είναι παρόμοια με τα αντικείμενα JSON (JavaScript Object Notation). Οι τιμές των πεδίων μπορεί να περιλαμβάνουν άλλα έγγραφα, πίνακες και πίνακες εγγράφων.

Η MongoDB αποθηκεύει έγγραφα σε συλλογές. Οι συλλογές είναι ανάλογες με τους πίνακες στις σχεσιακές βάσεις δεδομένων.

Στη MongoDB βάση δεδομένων καλείται ένα φυσικό δοχείο για συλλογές. Κάθε βάση δεδομένων λαμβάνει το δικό της σύνολο αρχείων στο σύστημα αρχείων. Ένας διακομιστής MongoDB μπορεί να περιλαμβάνει πολλές βάσεις δεδομένων.

Στην αρχιτεκτονική της πλατφόρμας που αναπτύχθηκε για αυτή τη διπλωματική, τα δεδομένα που αποθηκεύονται στη ΒΔ ειναι δύο ειδών:

1. Δεδομένα χρηστών
2. Δεδομένα μηνυμάτων των χρηστών

Δεδομένα χρηστών

Για κάθε χρήστη που εγγράφεται στο σύστημα, εισάγεται ενα νέο έγγραφο στη βάση δεδομένων "auth", στη συλλογή "users". Το κάθε έγγραφο περιλαμβάνει τα στοιχεία του χρήστη, στοιχεία σχετικά με τον προσωπικό του RabbitMQ Server αν υπάρχει, τα στοιχεία των ενεργών Φίλτρων - Καταναλωτών Μηνυμάτων αν υπάρχουν, και τέλος στατιστικά στοιχεία που περιγράφουν την πιο πρόσφατη εικόνα της διακίνησης μηνυμάτων μεταξύ του RMQ Server των Φίλτρων και της ΒΔ.

[8]https://docs.mongodb.com/manual/

Αυθεντικοποίηση: Κατά τη διαδικασία της αυθεντικοποίησης, ο κωδικός που δίνει ο χρήστης στη σελίδα σύνδεσης του Frontend συγκρίνεται με το hash του κωδικού που είχε ορίσει κατά την εγγραφή του και το οποίο είναι αποθηκευμένο στη ΒΔ.

Δημιουργία Φίλτρων - Καταναλωτών Μηνυμάτων: Οι πληροφορίες που αποθηκεύονται για το κάθε ενεργό Φίλτρο του χρήστη είναι οι εξής:

- Το όνομά του
- Οι ετικέτες του
- Το όνομα του K8s Deployment του
- Το όνομα του K8s Namespace του
- Η χρονική στιγμή δημιουργίας του
- Το όνομα της ουράς του RMQ Server στην οποία ακούει το Φίλτρο
- Το όνομα του Exchange το οποίο είναι συνδεδεμένο με την ουρά
- Το κλειδί σύνδεσης της ουράς με το Exchange
- Μία λίστα με τις συνθήκες κάτω από τις οποίες καταγράφεται κάποιο μήνυμα στη ΒΔ. Το κάθε στοιχείο αυτής της λίστας περιέχει:
  - Το όνομα μιας μεταβλητής που πρέπει να περιλαμβάνεται στα δεδομένα του μηνύματος
  - Τον τελεστή ο οποίος προσδιορίζει τη μαθηματική πράξη με βάση την οποία θα γίνει η σύγκριση της τιμής της μεταβλητής του μηνύματος και της τιμής που δόθηκε από τον χρήστη
  - Η τιμή με την οποία θα συγκριθεί το περιεχόμενο της μεταβλητής του μηνύματος

Στατιστικά στοιχεία του συστήματος του χρήστη: ΕΔΩ ΘΕΛΕΙ ΠΙΘΑΝΟΤΑΤΑ ΑΛΛΑΓΗ. ΧΡΕΙΑΖΕΤΑΙ ΝΑ ΑΠΟΘΗΚΕΥΟΝΤΑΙ ΣΤΟΙΧΕΙΑ ΣΧΕΤΙΚΑ ΜΕ ΤΗ ΣΥΝΟΛΙΚΗ ΛΕΙΤΟΥΡΓΙΑ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ ΤΟΥ ΧΡΗΣΤΗ ΚΑΙ ΟΧΙ ΜΟΝΟ ΤΟΥ ΧΡΟΝΙΚΟΥ ΠΑΡΑΘΥΡΟΥ ΠΟΥ ΖΗΤΗΣΕ Ο ΧΡΗΣΤΗΣ.

Κάθε φορά που ενεργοποιείται η παρουσίαση των στατιστικών του συστήματος από τη διεπαφή του χρήστη για κάποιο χρονικό παράθυρο, αυτές οι πληροφορίες αποθηκεύονται στη ΒΔ. Αυτές οι καταγραφές αποτελούνται από:

- Τον αριθμό των ενεργών Exchanges
- Τον αριθμό των μηνυμάτων που παραλήφθηκαν
-
-

-
-
-
-
-
-
-
-
-

Δεδομένα μηνυμάτων των χρηστών

Κατά τη λειτουργία του συστήματος, όσα από τα μηνύματα που δρομολογούνται στα Φίλτρα, παρατηρείται ότι καλύπτουν τις συνθήκες που έχουν οριστεί από τον χρήστη, αποθηκεύονται στη ΒΔ.

Η καταγραφή κάθε τέτοιου μηνύματος, συνοδεύεται από κάποιες πρόσθετες πληροφορίες:

- Το όνομα του Φίλτρου που το επεξεργάστηκε
- Το όνομα του Exchange μέσω του οποίου εισάχθηκε στο σύστημα
- Το όνομα του Queue στο οποίο δρομολογήθηκε
- Τον σειριακό αριθμό επεξεργασίας του μηνύματος από το συγκεκριμένο Φίλτρο
- Το κλείδί σύνδεσης του Queue με το Exchange
- Τη χρονική στιγμή της δημιουργίας του μηνύματος
- Τη χρονική στιγμή της καταγραφής του μηνύματος στη ΒΔ
- Τη συνθήκη η οποία καλύφθηκε με αποτέλεσμα την καταγραφή του μηνύματος στη ΒΔ

4.3.4 Διαμεσολαβητής μηνυμάτων - RMQ Server

Για την υλοποίηση του Διαμεσολαβητή μηνυμάτων επιλέχθηκε ο RMQ Server και συγκεκριμένα η εικόνα του container "rabbitmq:3-management-alpine".

Ο ρόλος του Διαμεσολαβητή είναι διττός. Η κύρια λειτουργία του είναι να υλοποιεί έναν μηχανισμό ο οποίος δέχεται μηνύματα από συσκευές εξωτερικές του συστήματος, τα αποθηκεύει προσωρινά και τα δρομολογεί κατάλληλα στις αντίστοιχες εφαρμογές προς κατανάλωση, βάσει προκαθορισμένων κανόνων.

Παράλληλα με τη διαχείριση μηνυμάτων, ο Διαμεσολαβητής που επιλέχθηκε περιλαμβάνει ένα plugin το οποίο παρέχει πρόσβαση σε υπηρεσίες παρουσίασης μιας λεπτομερούς εικόνας του συστήματος και της διακίνησης μηνυμάτων μέσα από αυτό.

Ο χρήστης έχει τη δυνατότητα να εξερευνήσει μέσω ενός φιλικού γραφικού περιβάλλοντος διάφορες πτυχές του Διαμεσολαβητή και των εσωτερικών στοιχείων του, τα ενεργά Φίλτρα που είναι συνδεδεμένα με αυτόν, ρυθμούς εισαγωγής και εξαγωγής μηνυμάτων από και προς το σύστημα και πολλά άλλα.

Δρομολόγηση μηνυμάτων: Η είσοδος του συστήματος είναι τα Exchanges του Διαμεσολαβητή. Εκείνα αναλαμβάνουν την ευθύνη να δρομολογήσουν τα εισερχόμενα μηνύματα στις Ουρές με τις οποίες είναι συνδεδεμένα. Μία Ουρά μπορεί να δημιουργηθεί όταν ο χρήστης εφαρμόσει ένα Φίλρο - Καταναλωτή μηνυμάτων. Ο χρήστης ορίζει το όνομα της Ουράς από την οποία θα λαμβάνει μηνύματα το Φίλτρο και αν δεν υπάρχει ήδη Ουρά με αυτό το όνομα τότε αυτή δημιουργείται.

Ο χρήστης ορίζει επίσης με ποιο Exchange θέλει να συνδεθεί η Ουρά και παρέχει το κλειδί σύνδεσης της Ουράς με το Exchange. Αυτό σημαίνει ότι το Exchange δρομολογεί προς την Ουρά μόνο τα εισερχόμενα μηνύματα, το κλειδί δρομολόγησης των οποίων ταιριάζει με το κλειδί αυτής της σύνδεσης.

Υπάρχουν διαφορετικοί τύπου από Exchanges, τα οποία δρομολογούν μηνύματα με διαφορετικό τρόπο. Ο τύπος που χρησιμοποιήθηκε είναι τα Topic Exchanges καθώς προσφέρουν αυξημένη ευελιξία στη δρομολόγηση. Τα μηνύματα που εισέρχονται σε αυτόν τον τύπο από Exchanges πρέπει να έχουν κλειδιά δρομολόγησης συγκεκριμένης μορφής. Το κάθε κλειδί πρέπει να είναι ένα σύνολο από λέξεις χωρισμένες από τελείες. Ο αριθμός των λέξεων που μπορεί να χρησιμοποιηθεί περιορίζεται μόνο από το ανώτατο όριο των 255 bytes. Ένα παράδειγμα έγκυρου κλειδιού δρομολόγησης θα μπορούσε να είναι το "quick.orange.rabbit".

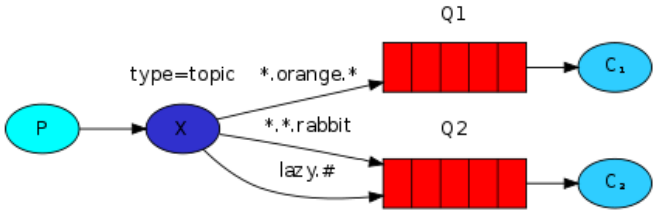
Τα κλειδιά σύνδεσης των Exchanges με τις Ουρές πρέπει να είναι της ίδιας μορφής. Για να προωθηθεί ένα μήνυμα από ένα Exchange σε μία Ουρά πρέπει το κλειδί σύνδεσής τους να ταιριάζει με το κλειδί δρομολόγησης του μηνύματος. Στα κλειδιά σύνδεσης μπορούν να χρησιμοποιηθούν δύο πρόσθετοι χαρακτήρες με ειδική σημασία:

- ¨*¨ Ο αστερίσκος χρησιμοποιείται προς αντικατάσταση ακριβώς μίας λέξης
- "#" Η δίεση χρησιμοποιείται προς αντικατάσταση καμίας ή περισσότερων λέξεων

Στο παράδειγμα του σχήματος 4.1 που ακολουθεί, στην Ουρά Q1 θα προωθηθούν όλα τα μηνύματα των οποίων το κλειδί δρομολόγησης αποτελείται από μια οποιαδήποτε λέξη, ακολουθούμενη από τη λέξη "orange", ακολουθούμενη από μια οποιαδήποτε λέξη.

Στην Ουρά Q2 θα προωθηθούν όλα τα μηνύματα των οποίων το κλειδί δρομολόγησης αποτελείται είτε από δύο οποιεσδήποτε λέξεις ακολουθούμενες από τη λέξη "rabbit", είτε από τη λέξη "lazy" ακολουθούμενη από αόριστο αριθμό από άλλες λέξεις.

Αναφορικά με το παράδειγμα που αναφέρθηκε, ένα μήνυμα με το κλειδί δρομολόγησης "quick.orange.rabbit" θα προωθούνταν και στις δύο ουρές στο παρακάτω σχήμα.



Σχήμα 4.1: Παράδειγμα δρομολόγησης μηνυμάτων μέσω Topic Exchanges

Ένα μήνυμα του οποίου το κλειδί δρομολόγησης δεν ταιριάζει με τη μορφή κανενός κλειδιού σύνδεσης, απορρίπτεται από το Exchange ως μη-δρομολογήσιμο.

Ένα μήνυμα του οποίου το κλειδί δρομολόγησης ταιριάζει με περισσότερα από ένα κλειδιά σύνδεσης, θα προωθηθεί σε όλες τις συνδέσεις αυτές. Στην περίπτωση που δύο ή περισσότερες συνδέσεις υπάρχουν μεταξύ ενός Exchange και μίας Ουράς και το κλειδί δρομολόγησης ενός μηνύματος ταιριάζει με περισσότερα από ένα κλειδιά των συνδέσεων αυτών, τότε το μήνυμα προωθείται μόνο μία φορά στη συγκεκριμένη ουρά.

Υπηρεσίες παρακολούθησης και διαχείρισης του Διαμεσολαβητή: Η πρόσβαση στο γραφικό περιβάλλον παρουσίασης του Διακομιστή επιτυγχάνεται από τον κάθε χρήστη με τα διαπιστευτήρια του λογαριασμού που δημιούργησε στην πλατφόρμα.

Σημαντικές δυνατότητες που προσφέρονται με την ενεργοποίηση του γραφικού περιβάλλοντος είναι οι ακόλουθες:

- Η δημιουργία, καταγραφή και διαγραφή στοιχείων του Διαμεσολαβητή όπως Exchanges, Queues και συνδέσεις
- Η παρακολούθηση του αριθμού των μηνυμάτων στις Ουρές, του ρυθμού εισαγωγής και εξαγωγής μηνυμάτων από κάθε στοιχείο του Διαμεσολαβητή και του ρυθμού χρήσης πόρων από τις συνδέσεις
- Παρακολούθηση χρήσης πόρων Διαμεσολαβητή: sockets και file descriptors, ανάλυση χρήσης μνήμης και διαθέσιμος χώρος στο δίσκο
- Αποστολή και λήψη μηνυμάτων (χρήσιμα σε περιβάλλοντα ανάπτυξης και για την αντιμετώπιση προβλημάτων)

Εκτός από τα παραπάνω θετικά σημεία, ο χρήστης πρέπει να λάβει υπόψιν του ότι η χρήση των υπηρεσιών αυτών ενέχει κάποια προβλήματα και περιορισμούς.

Συγκεκριμένα, το σύστημα παρακολούθησης μέρος του συστήματος που παρακολουθείται. Αυτό αρχικά σημαίνει πως εισάγονται στις μετρήσεις ορισμένες πρόσθετες καθυστερήσεις και αυξάνεται το φορτίο του συστήματος. Επομένως επιβαρύνεται η ακρίβεια των μετρήσεων.

Τα δεδομένα που αποθηκεύονται περιγράφουν την πιο πρόσφατη εικόνα του συστήματος μόνο και όχι τη συνολική λειτουργία του από την εκκίνησή του. Οι πληροφορίες οι οποίες παρουσιάζονται λοιπόν επεκτείνονται μέχρι και μερικές ώρες πριν την παρούσα στιγμή. Πληροφορίες για προηγούμενες ημέρες, εβδομάδες και μήνες δεν αποθηκεύονται.

Ο σχεδιασμός του εργαλείου δίνει έμφαση στην ευκολία χρήσης του και όχι στην βέλτιστη διαθεσιμότητα, επομένως μπορεί να παρατηρηθούν διακοπές και καθυστερήσεις στην παρουσίαση των δεδομένων.

4.3.5 Φίλτρα / Καταναλωτές - Consumers

Τα Φίλτρα αποτελούν μονάδες εφαρμογών τις οποίες δημιουργεί ο κάθε χρήστης ανάλογα με τις προτιμήσεις του. Η λειτουργία τους εξαρτάται από παραμέτρους οι οποίες καθορίζονται από τον χρήστη κατά τη στιγμή της δημιουργίας τους. Στη συνέχεια αναπτύσσονται στο K8s περιβάλλον και επικοινωνούν με τις υπόλοιπες εφαρμογές ώστε να εκπληρώσουν τον σκοπό τους.

Ο σκοπός αυτός είναι η κατανάλωση και επεξεργασία μηνυμάτων από τον Διαμεσολαβητή. Αρχικά επιτυγχάνεται η σύνδεση με τον RMQ Server και δημιουργούνται τα απαραίτητα στοιχεία του Διαμεσολαβητή (Exchanges, Queues). Στη συνέχεια το Φίλτρο περιμένει την παράδοση μηνυμάτων από τον διαμεσολαβητή όταν νέα μηνύματα οδηγηθούν στην Ουρά με την οποία είναι συνδεδεμένο. Τα περιεχόμενα των μηνυμάτων που θα παραδοθούν στο Φίλτρο ελέγχονται σε σχέση με τις συνθήκες καταγραφής που έχει ορίσει ο χρήστης. Στην περίπτωση που ικανοποιείται τουλάχιστον μία από τις συνθήκες αυτές, πραγματοποιείται μία σύνδεση με τη ΒΔ και το μήνυμα αποθηκεύεται στη συλλογή των μηνυμάτων στην προσωπική βάση δεδομένων του χρήστη.

Εκτός από τα παραπάνω, κάθε Φίλτρο έχει τη δυνατότητα συλλογής στατιστικών δεδομένων σχετικά με την πρόσφατη λειτουργία του Διαμεσολαβητή, χρησιμοποιώντας το HTTP API του RMQ Server. Αυτή η δυνατότητα χρησιμοποιείται όταν ενεργοποιηθεί η ζωντανή μετάδοση δεδομένων της λειτουργίας του συστήματος από τη διεπαφή του χρήστη.

Λεπτομέρειες της υλοποίησης: Τόσο οι Ουρές, όσο και τα Exchanges που ορίζονται μέσω των Φίλτρων, παίρνουν τις ιδιότητες durable και autoDelete.

Η σημασία της πρώτης ιδιότητας είναι ότι τα στοιχεία αυτά θα διατηρηθούν ακόμα και μετά από μια επανεκκίνηση του Διαμεσολαβητή. Προφανώς κάτι τέτοιο απαιτείται, μιας και σε αυτά τα στοιχεία βασίζεται το μοντέλο της δρομολόγησης των μηνυμάτων.

Η δεύτερη ιδιότητα σημαίνει ότι όταν αποσυνδεθεί το τελευταίο Φίλτρο από μια Ουρά τότε η Ουρά θα διαγραφεί αυτόματα. Αντίστοιχα όταν αποσυνδεθεί η τελευταία Ουρά από ένα Exchange, τότε το Exchange θα διαγραφεί. Αυτό είναι σημαντικό ώστε να μη προκαλείται συνωστισμός αχρείαστων στοιχείων στον Διαμεσολαβητή.

Μια άλλη σημαντική λεπτομέρεια είναι ότι οι Ουρές ΔΕΝ ορίζονται ως αποκλειστικές (exclusive: false). Κατά αυτόν τον τρόπο, μπορούν να δημιουργηθούν πολλά Φίλτρα τα οποία να καταναλώνουν μηνύματα από την ίδια Ουρά όταν το φορτίο σε αυτήν αυξάνεται πέρα από τις δυνατότητες κατανάλωσης ενός Φίλτρου.

Επιπλέον γίνεται χρήση της οδηγίας channel.prefetch(1). Αυτή ορίζει ότι η Ουρά δε θα παραδίδει περισσότερα από ένα μηνύματα σε κάθε καταναλωτή κάθε στιγμή. Έτσι το κάθε επόμενο μήνυμα είτε παραδίδεται στον επόμενο ελεύθερο καταναλωτή, είτε γίνεται αναμονή της επεξεργασίας του προηγούμενου μηνύματος από τον πρώτο καταναλωτή πριν του παραδοθεί το επόμενο.

Συλλογή δεδομένων πρόσφατης λειτουργίας του Διαμεσολαβητή: Ο τρόπος με τον οποίο γίνεται χρήση αυτής της δυνατότητας είναι μέσω της βιβλιοθήκης express.js. Το Φίλτρο εκτός από Καταναλωτής μηνυμάτων, λειτουργεί παράλληλα και ως server. Όταν καταφτάσει ένα αίτημα από το Backend ζητώντας δεδομένα τότε το Φίλτρο το εξυπηρετεί.

Στο αίτημα περιλαμβάνονται οι παράμετροι που προσδιορίζουν τόσο το χρονικό παράθυρο για το οποίο ζητούνται δεδομένα, όσο και ο ρυθμός δειγματοληψίας. Το Φίλτρο συνδέεται στο HTTP API του RMQ Server, αποκτάει τα ζητούμενα δεδομένα και τα επιστρέφει στο Backend.

Βιβλιοθήκες / εργαλεία:

- amqplib: 0.8.0 - Επικοινωνία με τον RMQ Server και διαχείριση μηνυμάτων
- express: 4.17.1 - Διαχείριση αιτημάτων API
- mongoose: 5.13.3 - Επικοινωνία με τη ΒΔ
- cors: 2.8.5 - Επιτρέπει τη φόρτωση πόρων στο πρόγραμμα περιήγησης από προέλευση διαφορετική του διακομιστή του αιτήματος
- node-fetch: 2.6.1 - Δημιουργία αιτημάτων HTTP
- body-parser: 1.19.0 - Προσαρμογή των περιεχομένων ενός αιτήματος HTTP στο πεδίο body του αντικειμένου req του αιτήματος

4.4 Διασύνδεση οντοτήτων μέσω K8s

Ο συνδυασμός όλων αυτών των οντοτήτων απαιτεί τον προσεκτικό καθορισμό των μέσων επικοινωνίας μεταξύ των containers τους. Ο τρόπος με τον οποίο αυτές οι υπηρεσίες επικοινωνίας γνωστοποιούνται και λαμβάνουν χώρα γίνεται κατανοητός στην ανάλυση που ακολουθεί.

4.4.1 Backend

Το Backend καθώς θα πρέπει να επικοινωνεί με τη ΒΔ για τις λειτουργίες της αυθεντικοποίησης και της διαχείρισης των χρηστών, χρειάζεται να έχει γνώση της IP διεύθυνσης του container της ΒΔ. Προς ικανοποίηση αυτής της απαίτησης, δημιουργείται ένα K8s Service του τύπου ClusterIP, το οποίο παρέχει πρόσβαση στο Pod της ΒΔ. Η σύνδεση του Backend με αυτό το Service γίνεται μέσω του yaml εγγράφου περιγραφής του Backend. Σε αυτό το έγγραφο ορίζεται μία μεταβλητή περιβάλλοντος που έχει ως τιμή το DNS (Domain Name System) όνομα του Service που εκθέτει τη ΒΔ σε επικοινωνίες εντός του K8s cluster.

Επίσης καθώς μέσω του Backend θα πρέπει να δημιουργούνται και να διαγράφονται άλλα K8s components (όπως είναι τα τλΠοδς των Φίλτρων και τα Services που τα εκθέτουν), είναι απαραίτητο το Backend να έχει τα απαραίτητα δικαιώματα δημιουργίας και διαγραφής K8s αντικειμένων. Η διαδικασία απόδοσης δικαιωμάτων στο K8s περιγράφεται ως εξής:

Αρχικά δημιουργείται ένας λογαριασμός ServiceAccount για την εφαρμογή η οποία χρειάζεται τα δικαιώματα. Ο λογαριασμός αυτός πρέπει να συνδεθεί με κάποιον ρόλο δικαιωμάτων Role. Υπάρχουν συγκεκριμένοι ρόλοι που ομαδοποιούν λογικά συναφή δικαιώματα, όμως μπορεί να δημιουργηθεί και κάποιος προσαρμοσμένος ρόλος σε περίπτωση που τα δικαιώματα που απαιτούνται δεν καλύπτονται ή υπερκαλύπτονται από κάποιον έτοιμο ρόλο. Η σύνδεση του ServiceAccount με τον ρόλο γίνεται μέσω ενός άλλου K8s αντικειμένου που λέγεται RoleBinding.

Αφού ολοκληρωθεί η παραπάνω διαδικασία και δημιουργηθούν τα απαραίτητα K8s αντικείμενα, τότε μπορεί να εισαχθεί το ServiceAccount στο yaml έγγραφο περιγραφής του Backend. Έτσι, όταν δημιουργηθεί ένα αίτημα από το Backend προς το σύστημα διαχείρισης του K8s, θα ελεγχθούν τα δικαιώματα που απαιτούνται για την εκτέλεσή του. Αν τα δικαιώματα αυτά περιλαμβάνονται στα δικαιώματα που είναι συνδεδεμένα με τον λογαριασμό του Backend, τότε θα επιτραπεί η εκτέλεση του αιτήματος από το K8s.

Στη συγκεκριμένη περίπτωση, τα δικαιώματα που αποδίδονται στο Backend είναι αυτά της καταγραφής, της δημιουργίας και της διαγραφής των K8s αντικειμένων Deployments, Services και Secrets.

Τέλος για να παρέχεται πρόσβαση στο Backend από άλλα Pods, δημιουργείται και ένα Service τύπου ClusterIP το οποίο εκθέτει το Backend εσωτερικά του K8s cluster.

4.4.2 Frontend

Nginx: Τα παραγόμενα αρχεία της ανάπτυξης μιας εφαρμογής μέσω του React.js πρέπει να παραδίδονται στους χρήστης που ζητάνε πρόσβαση στην εφαρμογή μέσω αιτημάτων HTTP. Ταυτόχρονα, οι λειτουργίες του Frontend παράγουν HTTP αιτήματα τα οποία πρέπει να δρομολογηθούν στο Backend.

Τη διαχείριση και εξυπηρέτηση αυτών των αιτημάτων αναλαμβάνει ένας HTTP Proxy Server. Για την υλοποίηση αυτού του server επιλέχθηκε το Nginx. Λόγοι που οδήγησαν σε αυτή την επιλογή ήταν η ευρεία διάδοση του Nginx, η μακρά παρουσία του στον χώρο και η πληθώρα των λειτουργικών επιλογών που παρέχει. Το container του Frontend έτσι χτίστηκε πάνω στην Docker εικόνα "nginx:1.19-alpine".

Το Nginx συνεπώς εξυπηρετεί τις συνδέσεις των χρηστών προσφέροντάς τους τα αρχεία της εφαρμογής του Frontend. Παράλληλα διαχειρίζεται τα εσωτερικά παραγόμενα αιτήματα και προωθεί όσα από αυτά έχουν ως προορισμό το Backend στην κατάλληλη τοποθεσία.

Frontend - Backend: Για τη γνωστοποίηση της τοποθεσίας του Backend στο Frontend, ορίζεται μία μεταβλητή περιβάλλοντος στο yaml έγγραφο περιγραφής του Frontend. Η μεταβλητή αυτή έχει ως τιμή το DNS όνομα του Service που εκθέτει το Backend σε επικοινωνίες εντός του K8s cluster.

Εξωτερική πρόσβαση στο Frontend: Το Frontend πρέπει να είναι προσβάσιμο από τοποθεσίες εκτός του K8s cluster ώστε να να επικοινωνούν με αυτό οι χρήστες μέσω των Browser τους. Αυτή η δυνατότητα πρόσβασης δίνεται με τη χρήση ενός K8s Service του τύπου LoadBalancer.

Στις τυπικές cloud υποδομές, παρέχονται υπηρεσίες LoadBalancers από τους διαχειριστές των cloud για αυτόν τον σκοπό. Οι LoadBalancers προσφέρουν μια δημόσια διεύθυνση IP προς χρήση, ενώ ταυτόχρονα εκτελούν λειτουργίες εξισορρόπησης φορτίου μεταξύ των στιγμιοτύπων των εφαρμογών τις οποίες εκθέτουν.

Καθώς όμως επιλέχθηκε λόγω διαθεσιμότητας πόρων το Minikube για την ανάπτυξη του K8s cluster, αναλύεται παρακάτω η διαδικασία παροχής πρόσβασης σε μια εφαρμογή εντός του K8s cluster από τοποθεσία εκτός του cluster.

Στο περιβάλλον του Minikube ενώ υποστηρίζονται τα LoadBalancer Services, δεν έχουν την ίδια λειτουργικότητα. Αυτό που συμβαίνει αντίθετα είναι ότι το Service εκτίθεται σε μία θύρα (Port) της διεύθυνσης IP του K8s Node που έχει δημιουργήσει το Minikube. Πρακτικά δηλαδή πρόκειται για ένα Service του τύπου NodePort. Η διαφοροποίηση είναι ότι το Minikube LoadBalancer Service μπορεί να λάβει μία επιπλέον εξωτερική διεύθυνση IP η οποία όμως θα είναι προσβάσιμη μόνο από το μηχάνημα στο οποίο έχει αναπτυχθεί το Minikube. Αυτό γίνεται με την εκτέλεση της εντολής "minikube tunnel".[9]

Για το λόγο αυτό, για να μπορέσει να έχει πρόσβαση κάποιος άλλος υπολογιστής του ίδιου δικτύου σε ένα LoadBalancer Service πρέπει να επικοινωνήσει με τη διεύθυνση <nodeIP>:<ServicePort>. Όμως η IP του K8s Node δεν είναι γνωστή στο router γιατί όπως αναφέρθηκε είναι host-only. Έτσι για να επιτευχθεί η επικοινωνία πρέπει να προστεθεί η πληροφορία στον δεύτερο υπολογιστή ότι τα αιτήματα με παραλήπτη το IP του Minikube θα πρέπει να έχουν ως gateway την τοπική διεύθυνση IP του μηχανήματος στο οποίο είναι ανεπτυγμένο το Minikube και να δρομολογηθούν εκεί.[10]

Αντίστοιχα στο πρώτο μηχάνημα πρέπει να επιτραπεί η προώθηση μηνυμάτων με τελικό παραλήπτη την τοπική διεύθυνση του K8s cluster, καθώς η προκαθορισμένη πολιτική είναι να απορρίπτονται όλα τα μηνύματα τα οποία δεν έχουν ως τελικό παραλήπτη την τοπική διεύθυνση του συγκεκριμένου μηχανήματος.[11]

4.4.3 Βάση Δεδομένων

Κύρια λειτουργία της ΒΔ είναι η αποθήκευση και διατήρηση δεδομένων του συστήματος. Επομένως τα δεδομένα της πρέπει να αποθηκεύονται εκτός του εφήμερου συστήματος αρχείων του container στο οποίο στεγάζεται. Αυτή η απαίτηση καλύπτεται με τη χρήση των Persistent Volumes που προσφέρει το K8s.

Πρώτα δημιουργείται ένα Persistent Volume (PV) και στη συνέχεια εκδίδεται μία αξίωση πάνω σε αυτόν τον τόμο αποθήκευσης μέσω ενός Persistent Volume Claim (PVC). Αυτή η αξίωση συνδέεται με το container της ΒΔ μέσω του yaml εγγράφου περιγραφής της ΒΔ.

Επίσης η ΒΔ αρχικοποιείται με ενεργοποιημένη την ελεγχόμενη πρόσβαση στις υπηρεσίες της μέσω αυθεντικοποίησης. Αυτό γίνεται με την εισαγωγή διαπιστευτηρίων

[9] Η εντολή "minikube tunnel" λειτουργεί ως ένα process, δημιουργώντας ένα network route στο μηχάνημα (host) στο οποίο είναι ανεπτυγμένο το Minikube προς την υπηρεσία CIDR (Classless inter-domain routing) του K8s cluster χρησιμοποιώντας τη διεύθυνση IP του cluster ως gateway. Έτσι εκτίθεται κάθε εξωτερική IP απευθείας σε οποιοδήποτε πρόγραμμα εκτελείται στο λειτουργικό σύστημα του host.

[10] e.g.: route add 192.168.49.0 mask 255.255.255.0 192.168.1.154

[11] sudo sysctl net.ipv4.ip_forward=1, sudo iptables -A FORWARD -j ACCEPT

επιπέδου διαχειριστή στην εφαρμογή της ΒΔ κατά τη δημιουργία της. Το έγγραφο περιγραφής της ΒΔ συνδέεται για αυτόν τον λόγο με τα περιεχόμενα ενός K8s Secret στο οποίο είναι αποθηκευμένα τα διαπιστευτήρια.

Για την παροχή πρόσβασης στην εφαρμογή της ΒΔ δημιουργείται ένα Service τύπου ClusterIP. Έτσι τα άλλα Pods εντός του K8s cluster μπορούν να επικοινωνούν με τη ΒΔ.

4.4.4 Διαμεσολαβητής μηνυμάτων - RMQ Server

Κατά την εγγραφή κάθε χρήστη, το Backend αναλαμβάνει να δημιουργήσει ένα Pod που υλοποιεί έναν RMQ Server μόνο για αυτόν τον χρήστη. Ταυτόχρονα αναπτύσσει δύο Services τα οποία προσφέρουν πρόσβαση στον Διαμεσολαβητή για διαφορετικούς λόγους το καθένα.

Δημιουργείται ένα ClusterIP Service που παρέχει πρόσβαση στο Port 5672 του Διαμεσολαβητή ώστε τα Φίλτρα να μπορούν να καταναλώνουν μηνύματα από αυτόν με χρήση του πρωτοκόλλου AMQP.

Επίσης ανατπύσσεται και ένα LoadBalancer Service που εκθέτει τα Ports 15672 και 5672 εκτός του K8s cluster. Το Port 15672 παρέχει πρόσβαση στον χρήστη στο γραφικό περιβάλλον διαχείρισης και εποπτείας του Διαμεσολαβητή, ενώ το Port 5672, δίνει τη δυνατότητα σε συσκευές ή εφαρμογές εκτός του cluster να παράγουν και να στέλνουν μηνύματα στον RMQ Server.

4.4.5 Φίλτρα / Καταναλωτές - Consumers

Η εκτέλεση των λειτουργιών του Φίλτρου απαιτεί πρόσβαση τόσο στον RMQ Server όσο και στη ΒΔ. Για αυτό δέχεται σαν μεταβλητές περιβάλλοντος τα διαπιστευτήρια του χρήστη, με τα οποία συνδέεται και στις δύο εφαρμογές που προαναφέρθηκαν.

Η τοποθεσία της ΒΔ και του RMQ Server εισάγονται επίσης με τη μορφή μεταβλητών περιβάλλοντος στο container του Φίλτρου κατά τη δημιουργία του από το Backend.

Κεφάλαιο 5

Παρουσίαση του συστήματος

Εδώ θα παρουσιαστούν use cases σε συνδυασμό με UI.

5.1 Κίνητρο

Άδειο

5.1.1 Ακόμα ένας τίτλος

Κεφάλαιο 6

Πειράματα & Αποτελέσματα

στρες τεστινγκ -᾽ φτιάχνω ένα ή πολλά φιλτερς και βλέπω πως θα αποδίδουν στο μηχνάνημα. -απόδοση αν γίνει κλιμάκωση και αν δε γίνει -να δουμε ποσους χρήστες σηκώνει -ποσους κονσουμερς σηκώνει -να μετρήσω δελαψσ: πότε παράχθηκε το μήνυμα και πότε γράφηκε στη ΒΔ -αν στείλω παρα πολλά μηνύματα σε ένα φιλτερ να δώ πως επηρεάζονται τα δελαψς -αν δώσω πρόσβαση σε κάποιον να στέλνει μηνύματα από λαν να δω πως επηρεατοναι τα ντιλειζ -και ερώτηση στο ντισκορντ

Μια αναφορά [1]. Some text in english.

6.1 Κίνητρο

Άδειο

6.1.1 Ακόμα ένας τίτλος

Κεφάλαιο 7

Συμπεράσματα & μελλοντική εργασία

Μπορεί να σπάσει ανάλογα με το μήκος του κεφαλαίου

Μια αναφορά [1]. Some text in english.

7.1 Ασφάλεια - Security context

7.1.1 Κρυπτογράφηση ΒΔ

7.1.2 Εφαρμογή κανόνων τείχους προστασίας (firewall)

7.1.3 Εκτέλεση των διεργασιών ως απλοί χρήστες - όχι ως root μέσα στα containers

7.2 Ανάπτυξη μιας Domain Specific Language (DSL) για την εφαρμογή φίλτρων

7.2.1

Παράρτημα Α΄

Ακρωνύμια και συντομογραφίες

LAN Local Area Network

REST Representational State Transfer

RPC Remote Procedure Call

AMQP Advanced Message Queuing Protocol

DB Database

TCP Transmission Control Protocol

TLS Transport Layer Security

K8s Kubernetes

CLI Command Line Interface

UI User Interface

PV Persistent Volume

PVC Persistent Volume Claim

RBAC Role-based Access Control

API Application Programming Interface

RMQ Rabbit MQ

CSRF Cross-Site Request Forgery

HTTP Hyper-Text Transfer Protocol

SSE Server Sent Events

URL Uniform Resource Locator

CSS Cascading Style Sheet

JSON JavaScript Object Notation

DNS Domain Name System

IP Internet Protocol

CIDR Classless inter-domain routing

PV Persistent Volume

PVC Persistent Volume Claim

ΒΔ Βάση Δεδομένων

Bibliography

[1] “Google.” [Online]. Available: https://www.google.com/