ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ

Ευφυής Διαχείριση Συμβάντων με τη Βοήθεια Ουρών Μηνυμάτων

ΚΥΛΑΦΑΣ ΧΡΗΣΤΟΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
Επιβλέπων
ΚΟΛΟΜΒΑΤΣΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

Λαμία, Ιούλιος 2023

UNIVERSITY OF THESSALY

SCHOOL OF SCIENCE

INFORMATICS AND COMPUTATIONAL BIOMEDICINE

Intelligent Events Management through the Assistance of Message Queues

KYLAFAS CHRISTOS

Master thesis

Advisor
KOLOMVATSOS KONSTANTINOS

Lamia. July 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ
ΚΑΤΕΥΘΥΝΣΗ

«ΠΛΗΡΟΦΟΡΙΚΗ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗΝ ΑΣΦΑΛΕΙΑ, ΔΙΑΧΕΙΡΙΣΗ ΜΕΓΑΛΟΥ ΟΓΚΟΥ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ»

Ευφυής Διαχείριση Συμβάντων με τη Βοήθεια Ουρών Μηνυμάτων

ΚΥΛΑΦΑΣ ΧΡΗΣΤΟΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων
ΚΟΛΟΜΒΑΤΣΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

Λαμία, Ιούλιος 2023

«Υπεύθυνη Δήλωση μη λογοκλοπής και ανάληψης προσωπικής ευθύνης»

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, και γνωρίζοντας τις συνέπειες της λογοκλοπής, δηλώνω υπεύθυνα και ενυπογράφως ότι η παρούσα εργασία με τίτλο «Ευφυής Διαχείριση Συμβάντων με τη Βοήθεια Ουρών Μηνυμάτων» αποτελεί προϊόν αυστηρά προσωπικής εργασίας και όλες οι πηγές από τις οποίες χρησιμοποίησα δεδομένα, ιδέες, φράσεις, προτάσεις ή λέξεις, είτε επακριβώς (όπως υπάρχουν στο πρωτότυπο ή μεταφρασμένες) είτε με παράφραση, έχουν δηλωθεί κατάλληλα και ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο ΔΗΛΩΝ

Ημερομηνία

Υπογραφή

Ευφυής Διαχείριση Συμβάντων με τη Βοήθεια Ουρών Μηνυμάτων

ΚΥΛΑΦΑΣ ΧΡΗΣΤΟΣ

Τριμελής Επιτροπή:

Ονοματεπώνυμο, ΚΟΛΟΜΒΑΤΣΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

Ονοματεπώνυμο, ΤΖΙΡΙΤΑΣ ΝΙΚΟΛΑΟΣ

Ονοματεπώνυμο, ΔΗΜΗΤΡΙΟΥ ΓΕΩΡΓΙΟΣ

ΠΕΡΙΛΗΨΗ

Η ανάγκη της επεξεργασίας των δεδομένων γίνεται όλο και πιο σημαντική στις μέρες μας. Το Kafka είναι ένας από τους πιο διαδεδομένους τρόπους ανάλυσης και επεξεργασίας ροών δεδομένων, μιας πλατφόρμας κατανεμημένης ροής. Το Kafka μπορεί να χειριστεί μεγάλο όγκο δεδομένων και μπορεί να ενσωματωθεί σε διάφορα συστήματα επεξεργασίας δεδομένων. Αυτή η διπλωματική εργασία υιοθετεί το Kafka για την αποστολή και διαχείριση δεδομένων που προέρχονται από αισθητήρες και εστιάζει στην αναγνώριση συμβάντων πάνω από τα δεδομένα που έχουν συλλεχθεί.

Η εργασία ξεκινάει με την περιγραφή της λειτουργίας των Kafka και MySQL μαζί και τα πλεονεκτήματα της χρήσης τους συνδέοντας τα. Στη συνέχεια περιγράφει την αρχιτεκτονική του προτεινόμενου συστήματος, συμπεριλαμβανομένου του τρόπου χρήσης του Kafka για τη λήψη των δεδομένων και του τρόπου με τον οποίο η MySQL ενημερώνεται με αυτά τα δεδομένα. Η εργασία διερευνά επίσης τον τρόπο υλοποίησης του συστήματος, συμπεριλαμβανομένης της εγκατάστασης του Kafka και της MySQL, τη δημιουργία των παραγωγών (producers) και των καταναλωτών (consumers) και τη διαμόρφωση των θεμάτων (topics).

Η υλοποίηση του συστήματος ελέγχεται με τη δημιουργία και την κατανάλωση των δειγματοληπτικών δεδομένων χρησιμοποιώντας το Kafka. Το σύστημα δοκιμάζεται με 5 περιπτώσεις χρήσης σε διάφορα μέρη του κόσμου, όπου μια εταιρεία logistics θέλει να παρακολουθεί την τοποθεσία των οχημάτων της για πιθανές βλάβες και να ενημερώνει μια βάση δεδομένων MySQL με αυτές τις πληροφορίες. Τα αποτελέσματα των δοκιμών δείχνουν ότι το σύστημα είναι σε θέση να επεξεργάζεται και να ενημερώνει την βάση δεδομένων με δεδομένα αποτελεσματικά και με ακρίβεια.

Η εργασία ολοκληρώνεται συζητώντας τους περιορισμούς του συστήματος και πιθανούς τομείς για μελλοντική έρευνα. Τονίζει επίσης τη σημασία της επεξεργασίας δεδομένων στις μέρες μας και την αξία της χρήσης Kafka και MySQL μαζί για να επιτευχθεί αυτό.

ABSTRACT

The need for data processing is becoming more important nowadays. Kafka is one of the most widespread platforms of analyzing and processing data streams, a distributed streaming platform. Kafka can handle large volumes of data and be integrated in various data processing systems. This paper investigates the collection and management of data streams through Kafka and the detection of events upon them.

The Thesis starts with the description of the Kafka functionalities and MySQL together and the advantages of using them by linking them. It then describes the architecture of the system, including how Kafka is used to obtain data and how MySQL is updated with that data. The paper also explores how the system is implemented, including the installation of Kafka and MySQL, the creation of producers and consumers, and the configuration of the topics.

The implementation of the system is evaluated by creating and consuming sample data using Kafka. The system is tested in 5 use cases in different parts of the world, where a logistics company wants to monitor the location of its vehicles for possible breakdowns and update a MySQL database with this information. The test results show that the system can process and update MySQL with data efficiently and accurately.

The Thesis concludes by discussing the limitations of the system and possible areas for future research. It also highlights the importance of data processing nowadays and the value of using Kafka and MySQL together to achieve this.

Πίνακας περιεχομένων

Πίνακας εικόνων

ΚΕΦΑΛΑΙΟ 1 Εισαγωγή

1.1 Εισαγωγή

Σε αυτή την εργασία, καλύπτονται αρκετά αξιόλογα θέματα που έχουν σχέση με την επεξεργασία και τη διαχείριση των ροών δεδομένων. Αρχικά, συζητούνται οι βασικές που σχετίζονται με ουρές μηνυμάτων και τις εφαρμογές τους στην επεξεργασία των δεδομένων. Εστιάζουμε στους διαφορετικούς τύπους των ουρών, όπως οι ουρές μηνυμάτων και οι ουρές εργασιών, και πώς μπορούν να χρησιμοποιηθούν για την επεξεργασία και την διαχείριση των δεδομένων σε κατανεμημένα συστήματα.

Επίσης, συζητείται το θέμα της αναγνώριση των συμβάντων, όπου εξηγούνται οι διάφορες τεχνικές και αλγόριθμοι που χρησιμοποιούνται για την αναγνώριση των συμβάντων σε ροές δεδομένων. Περιγράφεται επίσης η σημασία της αναγνώριση των συμβάντων σε διάφορους κλάδους, όπως ο οικονομικός τομέας, ο τομέας της υγείας και οι μεταφορές όπου και αποτιμήθηκε ένα εικονικό σενάριο αναγνώρισης συμβάντων.

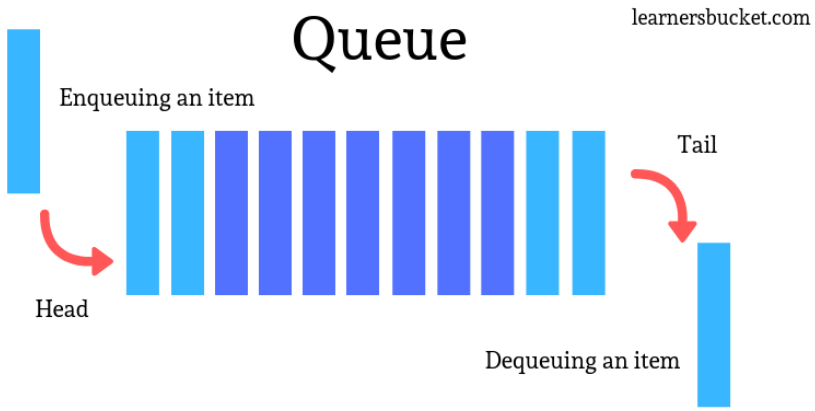
Στη συνέχεια, ερευνάται η λειτουργία του Kafka, μιας εφαρμογής κατανεμημένης ροής που χρησιμοποιείται για την κατασκευή των αγωγών δεδομένων και των εφαρμογών ροής. Εξετάζεται το πώς λειτουργεί το Kafka, τα βασικά του στοιχεία και πώς μπορεί να χρησιμοποιηθεί για τον εντοπισμό των συμβάντων. Εξερευνώνται επίσης τα πλεονεκτήματα της χρήσης του Kafka, όπως η επεκτασιμότητα, η ανοχή σε σφάλματα και η χαμηλή καθυστέρηση που έχει.

Τέλος, συζητείται η λειτουργία της MySQL, ενός ευρέως γνωστού συστήματος διαχείρισης των σχεσιακών βάσεων των δεδομένων. Εξερευνάται διάφορες δυνατότητες και λειτουργίες της MySQL, συμπεριλαμβανομένων των τύπων δεδομένων, των πινάκων και των ερωτημάτων, καθώς και το πώς γίνεται η χρήση για την αποθήκευση και την ανάκτηση των δεδομένων.

ΚΕΦΑΛΑΙΟ 2 ΟΥΡΕΣ ΜΗΝΥΜΑΤΩΝ

2.1 Ουρές

Μια βασική δομή δεδομένων στον προγραμματισμό είναι η ουρά. Οι ουρές είναι ανοικτές και στα δύο άκρα και λειτουργούν σύμφωνα με την έννοια της FIFO (First In First Out). Τα δεδομένα εισάγονται στο ένα άκρο της ουράς, που ονομάζεται πίσω άκρο ή ουρά, και τα δεδομένα αφαιρούνται στο άλλο άκρο, που ονομάζεται μπροστινό άκρο ή κεφαλή της ουράς. Όταν ένα αντικείμενο προστίθεται στην ουρά, μετακινείται από το πίσω στο μπροστινό άκρο μέχρι να είναι το επόμενο αντικείμενο που θα αφαιρεθεί από την ουρά (Εικόνα 1).



Εικόνα 1. Σχήμα ουράς

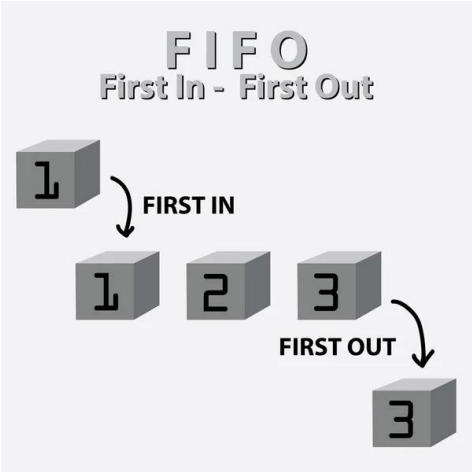
2.1.α Τύποι ουράς

Οι ουρές είναι μια ωφέλιμη δομή δεδομένων στον προγραμματισμό. Είναι το ισοδύναμο μιας ουράς για την αγορά εισιτηρίων στα εκδοτήρια από τα ΚΤΕΛ, όπου το πρώτο άτομο στην ουρά θα πάρει το πρώτο εισιτήριο.

Υπάρχουν τέσσερις τύποι ουρών:

1) Απλή ουρά.

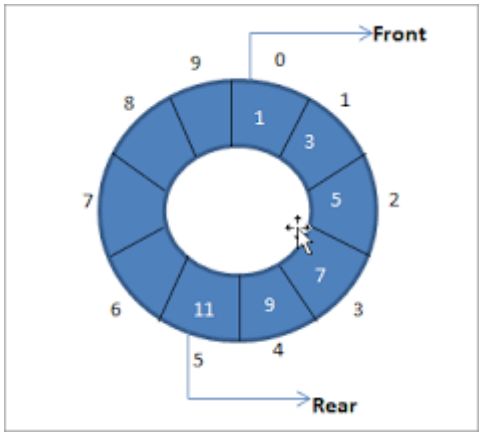
Σε μια απλή ουρά, η εισαγωγή γίνεται προς τα πίσω και η αφαίρεση προς τα εμπρός. Αυτό είναι απόλυτα σύμφωνο με την έννοια FIFO (First in First out) (Εικόνα 2).



Εικόνα 2. Σχήμα απλής ουράς

2) Κυκλική ουρά.

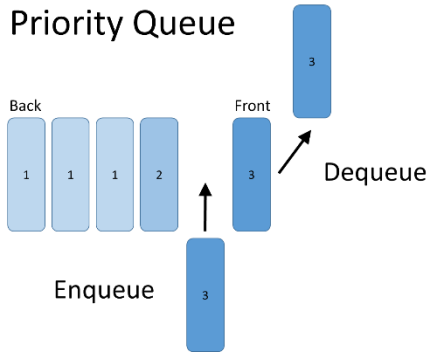
Σε μια κυκλική ουρά, το τελευταίο στοιχείο δημιουργεί μια κυκλική σχέση με το αρχικό στοιχείο. Το κύριο πλεονέκτημα των κυκλικών ουρών σε σχέση με τις απλές ουρές είναι η υψηλότερη χρήση της μνήμης. Εάν η τελευταία θέση είναι πλήρης και η πρώτη θέση είναι κενή ,ένα στοιχείο μπορεί να εισαχθεί στην πρώτη θέση. Μια απλή ουρά δεν μπορεί να ολοκληρώσει αυτή τη λειτουργία (Εικόνα 3).



Εικόνα 3. Σχήμα κυκλικής ουράς

3) Ουρές προτεραιότητας

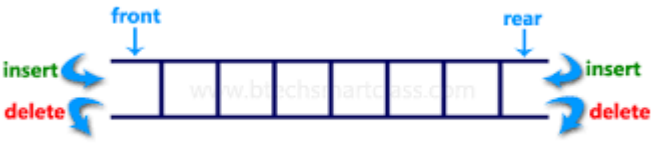
Μια ειδική ουρά, που ονομάζεται ουρά προτεραιότητας, είναι αυτή στην οποία κάθε στοιχείο έχει μια προτεραιότητα και αντιμετωπίζεται σύμφωνα με αυτή την προτεραιότητα. Εάν εμφανιστούν δύο στοιχεία με την ίδια προτεραιότητα, αντιμετωπίζονται με τη σειρά με την οποία βρίσκονται στην ουρά. Οι εισαγωγές γίνονται σύμφωνα με την άφιξη των δεδομένων και οι διαγραφές σύμφωνα με την προτεραιότητα (Εικόνα 4).



Εικόνα 4. Σχήμα ουράς προτεραιότητας

4) Αναμονή σε ουρά (ουρά διπλού τέλους).

Μια διπλή ουρά επιτρέπει την εισαγωγή και την αφαίρεση δεδομένων από το μπροστινό ή το πίσω μέρος. Ως εκ τούτου, αποκλίνει από την έννοια του First In First Out (FIFO) (Εικόνα 5).



Εικόνα 5. Σχήμα αναμονής σε ουρά

2.1.β Χρησιμότητα ουρών

Οι ουρές, οι οποίες είναι συνηθισμένες στις εφαρμογές υπολογιστών, μπορούν να χρησιμοποιηθούν σε οποιαδήποτε κατάσταση όπου χρειάζεται να γίνονται τα πράγματα με τη σειρά που έχουν κατονομαστεί, αλλά ο καταναλωτής δεν μπορεί να συμβαδίσει. Η δομή δεδομένων μιας ουράς απαιτεί ότι μετά την προσθήκη ενός νέου στοιχείου, όλα τα στοιχεία που έχουν προστεθεί προηγουμένως πρέπει να αφαιρεθούν για να δημιουργηθεί χώρος για το νέο στοιχείο. Αυτή η κατάσταση εμφανίζεται σχεδόν σε κάθε πτυχή της ανάπτυξης λογισμικού. Παραδείγματα αποτελούν οι ισότοποι που εξυπηρετούν αρχεία σε χιλιάδες χρήστες και τα πολυαπασχολούμενα λειτουργικά συστήματα. Στα λειτουργικά συστήματα, οι ουρές χρησιμοποιούνται για τον έλεγχο της πρόσβασης σε κοινόχρηστους πόρους, όπως εκτυπωτές, αρχεία και γραμμές επικοινωνίας. Σε τέτοιες περιπτώσεις, οι ουρές είναι η καλύτερη δομή δεδομένων για τους ακόλουθους λόγους:

1. Οι τοποθεσίες δεν μπορούν να ικανοποιήσουν όλες τις αιτήσεις, οπότε αυτές επεξεργάζονται με τη σειρά άφιξης σύμφωνα με την αρχή της προτεραιότητας.

2. Το λειτουργικό σύστημα δεν μπορεί να επεξεργαστεί όλες τις εργασίες ταυτόχρονα, οπότε προγραμματίζει τις εργασίες σύμφωνα με κάποια πολιτική.

2.1.γ Υλοποίηση ουρών

Οι ουρές μπορούν να χρησιμοποιηθούν σε διάφορες δομές δεδομένων, όπως των κυκλικών buffers, τις συνδεδεμένες λίστες και τους πίνακες. Κάθε δομή δεδομένων έχει τα πλεονεκτήματα και τα μειονεκτήματά της, ανάλογα με το τι χρειάζεται η εφαρμογή.

1)Υλοποίηση κυκλικού buffer

Σε μια κυκλική εφαρμογή buffer, μια ουρά αναπαρίσταται σαν μια κυκλική προσωρινή μνήμη. Τα στοιχεία προστίθενται στο πίσω μέρος του buffer και αφαιρούνται από το μπροστινό μέρος. Αυτή η υλοποίηση είναι αποτελεσματική και επιτρέπει την δυναμική αλλαγή μεγέθους, αλλά μπορεί να είναι περισσότερο περίπλοκη στην εφαρμογή της.

2)Υλοποίηση Συνδεδεμένης Λίστας

Σε μια υλοποίηση συνδεδεμένης λίστας, μια ουρά αναπαρίσταται ως συνδεδεμένη λίστα. Τα στοιχεία προστίθενται στο πίσω μέρος της λίστας και αφαιρούνται από το μπροστινό μέρος. Αυτή η υλοποίηση είναι πιο αποτελεσματική από μια υλοποίηση πίνακα, αλλά χρειάζεται περισσότερη μνήμη.

3)Υλοποίηση πίνακα

Σε μια υλοποίηση πίνακα, μια ουρά αναπαρίσταται ως πίνακας. Τα στοιχεία προστίθενται στο πίσω μέρος του πίνακα και αφαιρούνται από το μπροστινό μέρος. Αυτή είναι μια απλή υλοποίηση, αλλά μπορεί να είναι αναποτελεσματική όταν η ουρά γίνει μεγάλη.

2.2 Apache Kafka

2.2.1 Ορισμός Kafka

Το Apache Kafka (*Kafka-Apache Kafka.pdf*.) είναι μια πλατφόρμα ροής συμβάντων ανοιχτού κώδικα που αρχικά αναπτύχθηκε στο LinkedIn για να διαχειρίζεται μεγάλους όγκους δεδομένων. Από τότε, έχει γίνει η δημοφιλέστερη επιλογή για την μεταχείριση των ροών δεδομένων σε πολλούς κλάδους, συμπεριλαμβάνοντας των χρηματοοικονομικών, της υγειονομικής περίθαλψης και του εμπορίου.

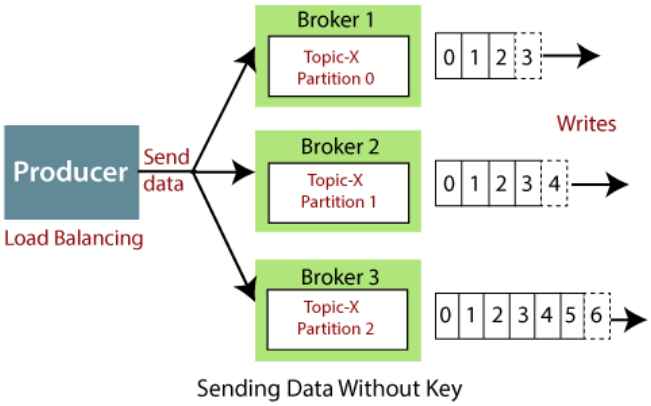
2.2.2 Βασικές έννοιες Kafka

Το Kafka αποτελείται από πολλά βασικά στοιχεία που συνεργάζονται για να παρέχουν τις δυνατότητες ανταλλαγής μηνυμάτων:

1) Παραγωγός (Producer)

Ο παραγωγός είναι ένα στοιχείο που είναι υπεύθυνο για τη δημοσίευση των μηνυμάτων στο Kafka. Μπορεί να είναι εφαρμογή ή σύστημα που δημιουργεί δεδομένα και θέλει να τα μοιράσει σε άλλα συστήματα. Ο παραγωγός (Εικόνα 6) δημιουργεί μηνύματα, τους δίνει προαιρετικά κλειδιά και τιμές και τα δημοσιεύει σε

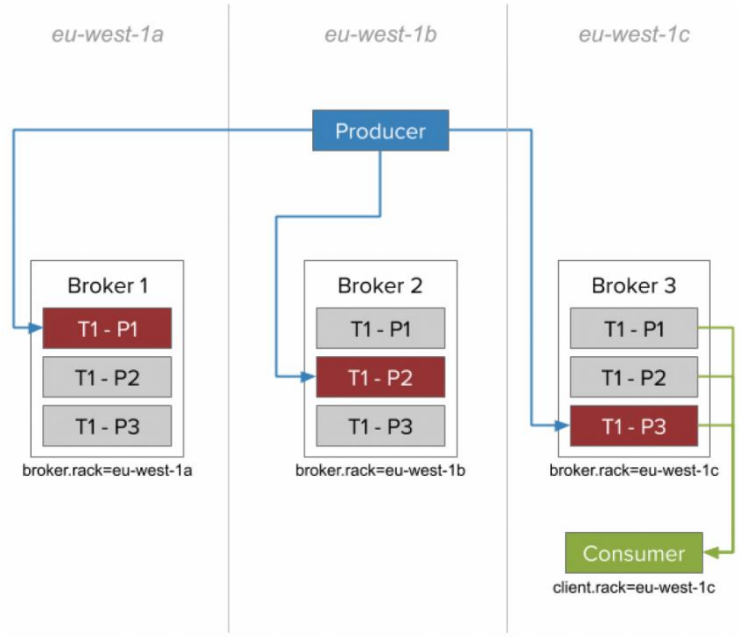
μεσολαβητής (Brokers) . Τα μηνύματα δημοσιεύονται σε συγκεκριμένα θέματα (Topics).



Εικόνα 6. Σχήμα του Παραγωγού

2) Μεσολαβητές (Brokers)

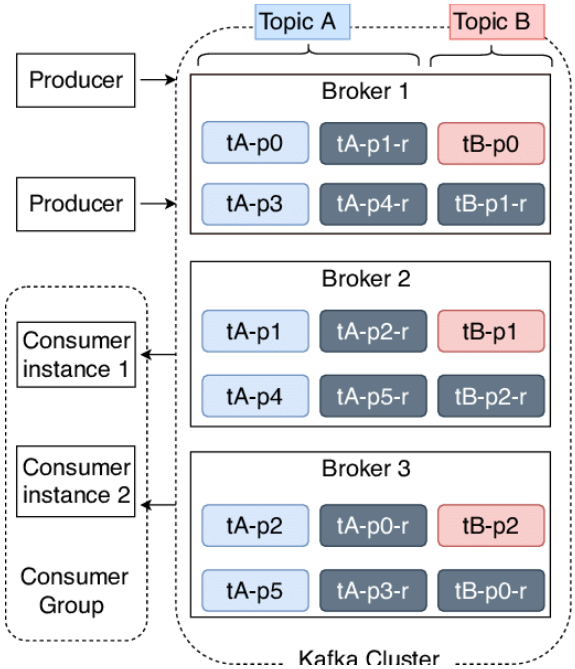
Οι μεσολαβητές είναι οι servers του Kafka και που αποτελούν μια βασική υποδομή του. Παίρνουν και αποθηκεύουν τα μηνύματα από τον παραγωγό και τα διαθέτουν στους καταναλωτές (consumers). Κάθε μεσολαβητής είναι υπεύθυνος για τη διαχείριση των διαμερισμάτων στο Kafka. Οι μεσολαβητές (Εικόνα 7) δημιουργούν αντίγραφα των δεδομένων σε όλο τη συστάδα για την ανοχή σφαλμάτων και την υψηλή διαθεσιμότητα.



Εικόνα 7. Σχήμα του Μεσολαβητή

3) Θέματα (Topics)

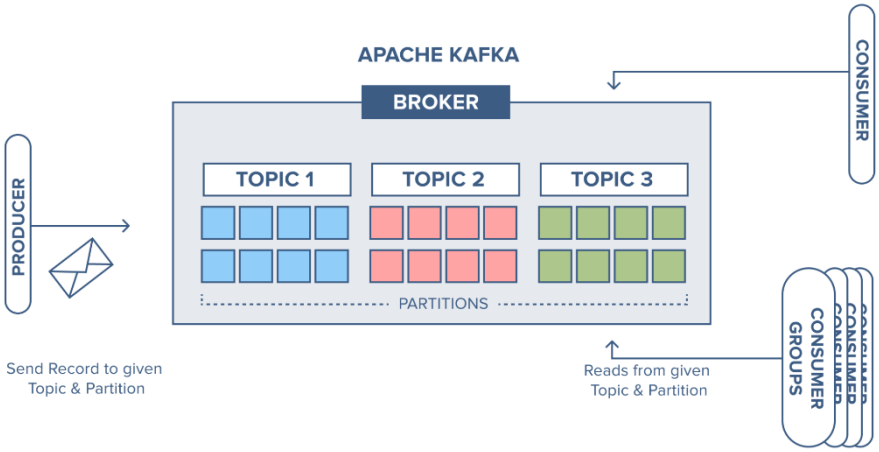
Τα θέματα είναι οι οργανωτικές μονάδες του Kafka. Εκπροσωπούν τις ροές των μηνυμάτων που ταξινομούνται ανάλογα το θέμα ή τον τύπο του συμβάντος. Ο παραγωγός δημοσιεύει τα μηνύματα σε θέματα και οι καταναλωτές εγγράφονται σε αυτά για να παίρνουν και να επεξεργάζονται τα μηνύματα. Τα θέματα (Εικόνα 8) χωρίζονται έτσι ώστε να επιτρέπεται η επεκτασιμότητα τους και η παράλληλη επεξεργασία.



Εικόνα 8. Σχήμα των Θεμάτων

4) Διαμερίσματα (Partitions)

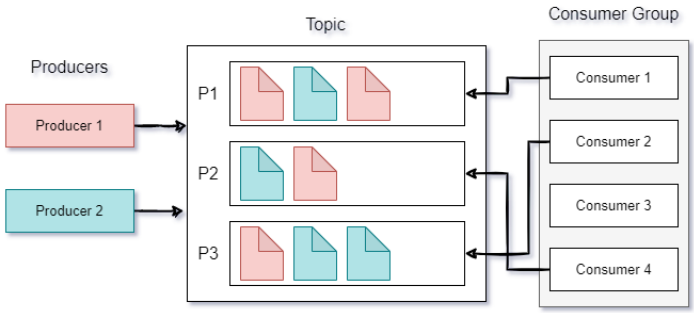
Ένα θέμα χωρίζεται σε διαμερίσματα, τα οποία είναι μεμονωμένες ταξινομημένες ακολουθίες μηνυμάτων. Κάθε διαμέρισμα (Εικόνα 9) καταχωρείται σε έναν μοναδικό μεσολαβητή. Με την διαχώριση επιτρέπεται σε πολλούς καταναλωτές να επεξεργάζονται τα μηνύματα παράλληλα. Στα μηνύματα μέσα σε ένα διαμέρισμα γίνονται διαδοχικές μετατοπίσεις που χρειάζονται ώστε να έχουν μοναδικά αναγνωριστικά.



Εικόνα 9. Σχήμα των Διαμερισμάτων

5) Καταναλωτές (Consumers)

Οι καταναλωτές είναι τα στοιχεία που εγγράφονται σε θέματα στο Kafka και καταναλώνουν τα μηνύματα. Διαβάζουν τα μηνύματα από τα διαμερίσματα ενός θέματος σε ροή. Οι καταναλωτές (Εικόνα 10) μπορούν να είναι ένα μέρος ομάδων καταναλωτών, και κάθε ομάδα μπορεί να επεξεργάζεται ένα υποσύνολο διαμερισμάτων. Αυτό επιτρέπει στην εξισορρόπηση του φορτίου και στην παράλληλη επεξεργασία των μηνυμάτων σε πολλούς καταναλωτές.



Εικόνα 10. Σχήμα του Καταναλωτή

2.2.3 Τρόπος λειτουργίας Kafka

Το Kafka είναι ένα σύστημα μηνυμάτων δημοσίευσης-εγγραφής, όπου τα μηνύματα παραμένουν σε ένα αρχείο καταγραφής που προσθέτει δεδομένα σε ουρά στους μεσολαβητές. Το αρχείο καταγραφής αποτελείται από πολλά τμήματα και κάθε ένα από αυτά περιέχει ένα σύνολο μηνυμάτων.

Κατά τη δημοσίευση ενός μηνύματος, ο παραγωγός επιλέγει το θέμα και προαιρετικά καθορίζει το κλειδί και την τιμή του μηνύματος. Το κλειδί καθορίζει σε ποιο διαμέρισμα εντός του θέματος θα καταχωρηθεί το μήνυμα. Εάν δεν έχει δημιουργηθεί κλειδί, τότε επιλέγεται τυχαία από το διαμέρισμα. Ο παραγωγός στέλνει ένα μήνυμα στο κύριο αντίγραφο του επιλεγμένου διαμερίσματος.

Όταν λαμβάνεται ένα μήνυμα, το κύριο αντίγραφο επισυνάπτει το μήνυμα στο τμήμα καταγραφής. Το μήνυμα λαμβάνει μια μοναδική μετατόπιση στο διαμέρισμα, η οποία υποδεικνύει τη θέση του στο αρχείο καταγραφής. Στη συνέχεια, το κύριο αντίγραφο επιβεβαιώνει στον παραγωγό ότι η εγγραφή ήταν επιτυχής. Επιπλέον, το κύριο αντίγραφο αναπαράγει το μήνυμα στο επόμενο αντίγραφο για την επίτευξη της ανοχής των σφαλμάτων.

Οι καταναλωτές εγγράφονται σε θέματα και καταναλώνουν τα μηνύματα από το κατανεμημένο διαμέρισμα. Κάθε καταναλωτής παρακολουθεί την αναπλήρωση στο οποίο έχει καταναλώσει μηνύματα. Καθώς φθάνουν νέα μηνύματα, ο καταναλωτής τα ανακτά από τον μεσολαβητή και τα επεξεργάζεται. Το Kafka διασφαλίζει ότι τα μηνύματα στα διαμερίσματα παραδίδονται με τη σειρά με την οποία δημιουργήθηκαν, εξασφαλίζοντας τη σημασιολογία της σειράς των μηνυμάτων εντός κάθε διαμερίσματος.

Το Kafka παρέχει επίσης παραμετροποιήσιμες πολιτικές διατήρησης για τη διαχείριση της αποθήκευσης μηνυμάτων. Τα μηνύματα μπορούν να διατηρηθούν για συγκεκριμένο χρονικό διάστημα ή με βάση το μέγεθος εκκίνησης. Αυτό επιτρέπει στο Kafka να χειρίζεται τόσο τις απαιτήσεις αποθήκευσης δεδομένων ροής όσο και τις απαιτήσεις αποθήκευσης ιστορικών δεδομένων.

Οι πολιτικές διατήρησης μπορούν να οριστούν τόσο σε επίπεδο θέματος όσο και σε επίπεδο μεσολαβητή. Σε επίπεδο θέματος, οι διαχειριστές μπορούν να ορίσουν τη διάρκεια των μηνυμάτων που θα διατηρούνται και να καθορίσουν το μέγιστο μέγεθος του αρχείου καταγραφής. Αυτό επιτρέπει τον λεπτομερή έλεγχο της συμπεριφοράς διατήρησης για συγκεκριμένα θέματα. Σε επίπεδο μεσολαβητή, μπορεί να οριστεί μια προεπιλεγμένη πολιτική διατήρησης, η οποία εφαρμόζεται σε θέματα χωρίς ρητές ρυθμίσεις διατήρησης.

Το Κάφκα υποστηρίζει δύο τύπους πολιτικής διατήρησης:

1) Διατήρηση με βάση το χρόνο

τα μηνύματα αποθηκεύονται για ένα συγκεκριμένο χρονικό διάστημα, μετά το οποίο διαγράφονται αυτόματα. Η περίοδος αυτή μπορεί να οριστεί σε ώρες, ημέρες ή και εβδομάδες, ανάλογα με τις απαιτήσεις της εφαρμογής. Στο τέλος της περιόδου διατήρησης, το Kafka διαγράφει αυτόματα τα παλαιότερα μηνύματα στο αρχείο καταγραφής, απελευθερώνοντας χώρο αποθήκευσης για τα νέα μηνύματα.

2) Διατήρηση με βάση το μέγεθος

Μια πολιτική διατήρησης με βάση το μέγεθος εξασφαλίζει ότι τα μηνύματα διατηρούνται μέχρι το αρχείο καταγραφής να φτάσει ένα συγκεκριμένο όριο μεγέθους. Αυτό το όριο μπορεί να οριστεί ως bytes, kilobytes, megabytes ή gigabytes .Όταν το μέγεθος του αρχείου καταγραφής υπερβεί το καθορισμένο όριο, το Kafka θα αρχίσει να διαγράφει τα παλιά μηνύματα για να δημιουργήσει χώρο για τα νέα εισερχόμενα μηνύματα.

Η διατήρηση με βάση το μέγεθος είναι επωφελής όταν είναι επικεντρωμένο στην αποθήκευση ενός σταθερού όγκου δεδομένων και όχι στη διατήρηση μηνυμάτων για ένα συγκεκριμένο χρονικό διάστημα. Αυτό επιτρέπει στο Kafka να διαχειρίζεται αποτελεσματικά τη χρήση του αποθηκευτικού χώρου και να διασφαλίζει ότι τα αρχεία καταγραφής δεν αυξάνονται επ'άπειρον και δεν καταναλώνουν υπερβολικό χώρο στο δίσκο.

Θέτοντας κατάλληλα τις πολιτικές διατήρησης, το Kafka μπορεί να διαχειριστεί αποτελεσματικά την αποθήκευση μηνυμάτων. Ο χρόνος διατήρησης των μηνυμάτων μπορεί να καθοριστεί με ευελιξία, επιτρέποντας στους προγραμματιστές να εξισορροπούν τις ενημερώσεις δεδομένων με τις απαιτήσεις αποθήκευσης.

Εκτός από τις πολιτικές διατήρησης, το Kafka υποστηρίζει επίσης συμπίεση, τη διαδικασία αφαίρεσης ανεπιθύμητων ή παλιών μηνυμάτων από τα αρχεία καταγραφής. Η συμπίεση βασίζεται σε κλειδιά μηνυμάτων και το Kafka διατηρεί μόνο τα πιο πρόσφατα μηνύματα για κάθε μοναδικό κλειδί στο διαμέρισμα καταγραφής. Αυτή η λειτουργία είναι χρήσιμη σε σενάρια όπου είναι σημαντικό να διατηρείται κάθε κλειδί ενημερωμένο, όπως η διατήρηση αρχείων καταγραφής αλλαγών ή η ενημέρωση συγκεντρωτικών δεδομένων.

Συνολικά, οι μηχανισμοί αποστολής και διαχείρισης μηνυμάτων του Kafka, σε συνδυασμό με διαμορφώσιμες πολιτικές διατήρησης και συμπίεσης, παρέχουν μια ισχυρή και επεκτάσιμη βάση για την επεξεργασία μηνυμάτων υψηλής απόδοσης, της ανοχής σε σφάλματα και την αποθήκευση των ροών δεδομένων.

2.3 MySQL

Η MySQL είναι ένα σύστημα διαχείρισης των σχεσιακών βάσεων δεδομένων ανοιχτού κώδικα (RDBMS) που χρησιμοποιείται ευρέως για την ανάπτυξη εφαρμογών που βασίζονται στο διαδίκτυο. Είναι εύκολη στην χρήση της, την αξιοπιστία της και την επεκτασιμότητα της, καθιστώντας τη μια καλή επιλογή για πολλές επιχειρήσεις. Η MySQL δημιουργήθηκε αρχικά το 1995 από τη MySQL AB, μια σουηδική εταιρεία λογισμικού, και αργότερα εξαγοράστηκε από την Oracle Corporation.

Η MySQL χρησιμοποιεί τη δομημένη γλώσσα των ερωτημάτων (SQL) για την αναζήτηση και τη διαχείριση των δεδομένων. Υποστηρίζει ένα ευρύ φάσμα λειτουργιών, συμπεριλαμβανομένων των συναλλαγών, των περιορισμών ξένων κλειδιών, των αποθηκευμένων διαδικασιών, των ενεργοποιήσεων και των προβολών, κάνοντάς την μια ισχυρή και ευέλικτη λύση της βάσης δεδομένων.

Ένα από τα ισχυρότερα πλεονεκτήματα της MySQL είναι η συμβατότητά της με τις διάφορες γλώσσες προγραμματισμού, όπως την Python, την PHP, την Java και άλλες πολλές. Αυτό βοηθάει τους προγραμματιστές να ενσωματώσουν εύκολα τη MySQL στις διαδικτυακές εφαρμογές τους.

Συνολικά, η MySQL είναι ένα ευέλικτο και ισχυρό σύστημα διαχείρισης βάσεων δεδομένων που έχει υιοθετηθεί από τις περισσότερες επιχειρήσεις και προγραμματιστές παγκοσμίως. Επίσης, είναι ιδανική επιλογή για τη διαχείριση των δεδομένων σε εφαρμογές που βασίζονται στο διαδίκτυο.

ΚΕΦΑΛΑΙΟ 3 ΑΝΑΓΝΩΡΙΣΗ ΣΥΜΒΑΝΤΩΝ

3.1 Αναγνώριση συμβάντων (event detection)

Η αναγνώριση των συμβάντων αναφέρεται στη διαδικασία του αυτόματου εντοπισμού και της κατηγοριοποίησης συμβάντων από μη δομημένες πηγές δεδομένων, όπως τα άρθρα ειδήσεων, τα έγγραφα κειμένου, τις αναρτήσεις στα μέσα κοινωνικής δικτύωσης και σε άλλους τύπους διαδικτυακού περιεχομένου. Σκοπός της αναγνώρισης συμβάντων είναι η εξαγωγή των χρήσιμων πληροφοριών σχετικά με ένα συμβάν, όπως ο χρόνος, η τοποθεσία, οι συμμετέχοντες και άλλες σχετικές λεπτομέρειες.

3.1.α Τεχνικές αναγνώρισης συμβάντων

Η αναγνώριση των συμβάντων είναι μια απαιτητική εργασία η οποία περιλαμβάνει την επεξεργασία του μεγάλου όγκου μη δομημένων δεδομένων. Για να επιτευχθεί αυτό, χρησιμοποιούνται διάφορες τεχνικές όπως η επεξεργασία της φυσικής γλώσσας, η μηχανική μάθηση και η ανάκτηση των πληροφοριών για τον εντοπισμό των σχετικών πληροφοριών.

Μια προσέγγιση για την αναγνώριση των συμβάντων περιλαμβάνει τη χρήση των τεχνικών της μηχανικής μάθησης χωρίς την επίβλεψη, όπως η ομαδοποίηση και η μοντελοποίηση των θεμάτων για την ομαδοποίηση παρόμοιων άρθρων ή εγγράφων. Αυτή η προσέγγιση μπορεί να είναι χρήσιμη για τον εντοπισμό των προτύπων και των τάσεων στα δεδομένα, αλλά μπορεί να μην είναι επαρκής για τον ακριβή προσδιορισμό των συγκεκριμένων γεγονότων.

Μια άλλη προσέγγιση για την αναγνώριση των συμβάντων είναι η εποπτευόμενη μηχανική εκμάθηση, όπου οι αλγόριθμοι εκπαιδεύονται σε σύνολα δεδομένων με ετικέτα για να αναγνωρίζουν τους συγκεκριμένους τύπους των συμβάντων. Αυτή η προσέγγιση μπορεί να είναι πιο ακριβής από τις μη εποπτευόμενες τεχνικές, αλλά απαιτείται μεγάλος όγκος δεδομένων με ετικέτα για εκπαίδευση.

Η αναγνώριση των συμβάντων έχει πολλές πρακτικές εφαρμογές, όπως ο εντοπισμός και η παρακολούθηση των φυσικών καταστροφών, η πρόβλεψη των τάσεων της χρηματιστηριακής αγοράς και ο εντοπισμός των πιθανών απειλών για την ασφάλεια. Είναι ένας σημαντικός τομέας της έρευνας στην τεχνητή νοημοσύνη και έχει τη δυνατότητα να παρέχει πολύτιμες γνώσεις για ένα ευρύ φάσμα των πραγματικών γεγονότων.

3.2 Σύνθετη Επεξεργασία Συμβάντων (CEP)

Η επεξεργασία σύνθετων συμβάντων (CEP) είναι μια τεχνική που χρησιμοποιείται για τον εντοπισμό και την ανάλυση σύνθετων συμβάντων σε ροές δεδομένων. Ανιχνεύει και εξάγει γεγονότα υψηλότερου επιπέδου που αποτελούνται από πολλαπλά απλά γεγονότα και κατανοεί τις χρονικές και συγκυριακές σχέσεις μεταξύ τους. Η επεξεργασία σύνθετων συμβάντων επιτρέπει την απόκτηση πολύτιμων γνώσεων, τον εντοπισμό μοτίβων και την λήψη τεκμηριωμένων αποφάσεων με βάση τα γεγονότα που συμβαίνουν στα δεδομένα.

Η επεξεργασία σύνθετων συμβάντων έχει σημαντικό ρόλο στις αρχιτεκτονικές που βασίζονται σε συμβάντα και στα συστήματα που βασίζονται σε συμβάντα, όπου η επεξεργασία σύνθετων συμβάντων υπερβαίνει την απλή αναγνώριση συμβάντων και ενσωματώνει τη συσχέτιση συμβάντων, την αναγνώριση προτύπων συμβάντων και τη σύνθεση σύνθετων συμβάντων. Με την επεξεργασία των συμβάντων κατά την εμφάνισή τους και την ανάλυση των σχέσεών τους, η επεξεργασία σύνθετων συμβάντων παρέχει μια ολοκληρωμένη κατανόηση της υποκείμενης δυναμικής των δεδομένων.

3.2.α Χαρακτηριστικά σύνθετης επεξεργασίας γεγονότων

1) Αναγνώριση μοτίβων συμβάντων

Η επεξεργασία σύνθετων συμβάντων επικεντρώνεται στον εντοπισμό μοτίβων και σχέσεων μεταξύ συμβάντων, επιτρέποντας την αναγνώριση των συμβάντων υψηλού επιπέδου που αποτελούνται από πολλαπλά απλά συμβάντα.

2) Χρονική και συμφραζόμενη συσχέτιση

Η επεξεργασία σύνθετων συμβάντων λαμβάνει υπόψη τη χρονική σειρά και τις συμφραζόμενες πληροφορίες που σχετίζονται με τα συμβάντα για τη δημιουργία ουσιαστικών συσχετίσεων.

3) Σύνθεση συμβάντων

Η επεξεργασία σύνθετων συμβάντων επιτρέπει τη σύνθεση συμβάντων, την άθροιση και το συνδυασμό σχετικών συμβάντων για τη δημιουργία συμβάντων υψηλότερου επιπέδου ή ακολουθιών συμβάντων.

4) Επεξεργασία σε πραγματικό χρόνο

Η επεξεργασία σύνθετων συμβάντων λειτουργεί με ροές δεδομένων άμεσα, επιτρέποντας την έγκαιρη ανίχνευση και ανάλυση συμβάντων.

5) Επεκτασιμότητα και ανοχή σφαλμάτων

Η επεξεργασία σύνθετων συμβάντων έχει σχεδιαστεί για να διαχειρίζεται μεγάλους όγκους συμβάντων και να διατηρεί υψηλή διαθεσιμότητα μέσω κατανεμημένης επεξεργασίας και μηχανισμών ανοχής σφαλμάτων.

3.2.β Εφαρμογές σύνθετης επεξεργασίας γεγονότων

1) Οικονομικά: Η σύνθετη επεξεργασία γεγονότων χρησιμοποιείται σε συστήματα ανίχνευσης απάτης άμεσα για τη συσχέτιση πολλαπλών οικονομικών συναλλαγών και τον εντοπισμό ύποπτων μοτίβων και ανωμαλιών

2) Μεταφορές: Η σύνθετη επεξεργασία γεγονότων αναλύει δεδομένα αισθητήρων άμεσα για τον εντοπισμό συμφόρησης και ατυχημάτων και για τη δυναμική διαχείριση της κυκλοφορίας.

3) Υγεία: Η σύνθετη επεξεργασία γεγονότων διαδραματίζει βασικό ρόλο στην αναγνώρισης συμβάντων ασθενών άμεσα, όπως ο εντοπισμός πρώιμων ενδείξεων σήψης με βάση ζωτικά σημεία και εργαστηριακά δεδομένα, και επιτρέπει την προληπτική παρέμβαση.

4) Ασφάλεια: Η σύνθετη επεξεργασία γεγονότων συμβάλλει στον εντοπισμό και την πρόληψη απειλών για την ασφάλεια στον κυβερνοχώρο, συσχετίζοντας τα αρχεία καταγραφής δικτύου, εντοπίζοντας μοτίβα εισβολής και ενεργοποιώντας ειδοποιήσεις ή αυτοματοποιημένες αντιδράσεις.

5) Διαχείριση εφοδιαστικής αλυσίδας: Η σύνθετη επεξεργασία γεγονότων επιτρέπει συμβάντα εφοδιαστικής αλυσίδας, όπως η παρακολούθηση των αποστολών προϊόντων, η διαχείριση αποθεμάτων και ο εντοπισμός σημείων συμφόρησης και καθυστερήσεων.

6) Διαδίκτυο των πραγμάτων (IoT): Η σύνθετη επεξεργασία γεγονότων χρησιμοποιείται ευρέως σε εφαρμογές IoT για την επεξεργασία συμβάντων άμεσα και τη λήψη αποφάσεων με βάση δεδομένα αισθητήρων, επιτρέποντας ευφυή συστήματα αυτοματισμού και ελέγχου.

7) Διαχείριση πελατειακών σχέσεων (CRM): Η επεξεργασία σύνθετων συμβάντων αναλύει τη συμπεριφορά των πελατών σε πραγματικό χρόνο για να επιτρέψει εξατομικευμένο μάρκετινγκ, στοχευμένες προωθητικές ενέργειες και προληπτική εξυπηρέτηση πελατών.

3.2.γ Πλατφόρμες σύνθετης επεξεργασίας γεγονότων

Υπάρχουν διάφορες πλατφόρμες επεξεργασίας της ροής των συμβάντων που διευκολύνουν την ανάπτυξη και την εγκατάσταση συστημάτων της επεξεργασίας σύνθετων συμβάντων. Αυτές οι πλατφόρμες παρέχουν την υποδομή, τα εργαλεία και το πλαίσιο που απαιτούνται για την αποτελεσματική επεξεργασία και ανάλυση των ροών συμβάντων.

Το Apache Flink (*Apache Flink® — Stateful Computations over Data Streams*.) είναι μια τέτοια πλατφόρμα, η οποία είναι γνωστή για τις δυνατότητες επεξεργασίας των ροών υψηλής απόδοσης. Παρέχει ένα ολοκληρωμένο μοντέλο για την επεξεργασία δέσμης και ροής και υποστηρίζει την επεξεργασία σύνθετων συμβάντων με τη χρήση της βιβλιοθήκης της επεξεργασίας σύνθετων συμβάντων.

Το Apache Kafka (*Kafka-Apache Kafka.pdf*.) είναι μια κατανεμημένη πλατφόρμα ροής που παρέχει μια κλιμακούμενη και ανεκτική σε σφάλματα υποδομή για τον χειρισμό των ροών συμβάντων άμεσα. Παρέχει αξιόπιστη αποθήκευση των συμβάντων, της ροής συμβάντων και δυνατότητες ολοκλήρωσης των δεδομένων.

Το Esper (‘Esper’.) είναι μια πλατφόρμα επεξεργασίας των συμβάντων ανοικτού κώδικα που παρέχει μια ισχυρή μηχανή επεξεργασίας των συμβάντων. Υποστηρίζοντας χαρτογράφηση προτύπων, την σύνθετη σύνθεση και την συσχέτιση των συμβάντων.

ΚΕΦΑΛΑΙΟ 4 ΠΡΟΤΕΙΝΟΜΕΝΟ ΣΥΣΤΗΜΑ

4.1 Προτεινόμενο σύστημα

Το προτεινόμενο σύστημα στοχεύει στην βελτίωση της γρήγορης παράδοσης πακέτων ενός δικτύου φορτηγών με την εφαρμογή ενός συστήματος παρακολούθησης και ειδοποίησης βλάβης. Αυτό το σύστημα χρησιμοποιεί ένα πρόγραμμα για να παρακολουθεί την τοποθεσία των φορτηγών κατά την διαδρομή τους και να εντοπίζει έγκαιρα τυχόν περιστατικά βλάβης. Σε περίπτωση βλάβης είτε το όχημα είναι για πολύ μεγάλο χρονικό διάστημα στο ίδιο σημείο είτε δηλώνει βλάβη χειροκίνητα ο οδηγός, το σύστημα θα εντοπίσει τα κοντινά φορτηγά που βρίσκονται τα τελευταία 30 λεπτά σε εμβέλεια των 200 χλμ. από το όχημα με βλάβη ώστε να μεταφερθούν στο σημείο για να παραλάβουν τα πακέτα.

4.2 Kafka και MySQL

Στις μέρες μας, η επεξεργασία των δεδομένων είναι κρίσιμη για να ληφθούν τεκμηριωμένες αποφάσεις. Το Kafka και η MySQL είναι δύο τεχνολογίες που μπορούν να χρησιμοποιηθούν μαζί για την υλοποίηση συστημάτων επεξεργασίας των δεδομένων. Σε αυτό το κεφάλαιο, μελετήθηκε πώς να χρησιμοποιείτε το Kafka και τη MySQL για να δημιουργηθεί μια σύνδεση επεξεργασίας των δεδομένων. Θα καλυφθούν τα ακόλουθα θέματα:

α. Ρύθμιση του Kafka και της MySQL
β. Παραγωγή δεδομένων στο Kafka
γ. Καταναλώνοντας δεδομένα από το Kafka

4.2.α Ρύθμιση του Kafka και της MySQL

Προτού ξεκινήσει η δημιουργία του αγωγού επεξεργασίας των δεδομένων, πρέπει να ρυθμιστεί το Kafka και τη MySQL. Υποθέτοντας ότι το Kafka και η MySQL είναι εγκατεστημένα και τρέχουν στον ίδιο υπολογιστή.

Για να ρυθμιστεί το Kafka, πρέπει να εγκατασταθεί η βιβλιοθήκη Kafka Python χρησιμοποιώντας το pip:

```
!pip install kafka-python
```

Για να ρυθμιστεί τη MySQL, πρέπει να εγκατασταθεί η βιβλιοθήκη Python της σύνδεσης MySQL χρησιμοποιώντας το pip:

```
!pip install mysql-connector-python
```

4.2. β Παραγωγή δεδομένων στο Kafka (Producer)

Το πρώτο βήμα για τη δημιουργία του αγωγού της επεξεργασίας των δεδομένων είναι η παραγωγή των δεδομένων στον Kafka. Σε αυτό το κομμάτι του κώδικα, θα ανακτηθούν τα δεδομένα τοποθεσίας των οχημάτων από μια βάση δεδομένων της MySQL και θα παραχθούν σε ένα θέμα Kafka.

```
8
9   import json
10  from kafka import KafkaProducer
11  import mysql.connector
12
13
14  # Connect to the database
15  db= mysql.connector.connect(
16      host="127.0.0.1",
17      user="root",
18      database="logistic"
19  )
20
21
22
23  # Retrieve vehicle data from the database
24  cursor = db.cursor()
25  cursor.execute("SELECT id, route_id, location FROM subroute_table")
26  subroute_table = cursor.fetchall()
27
28
29  # generating the Kafka Produser
30  producer=KafkaProducer(bootstrap_servers=['localhost:9092'])
31
32
33  for subroute_tables in subroute_table:
34      subroute_data={
35          "id":subroute_tables[0],
36          "route_id":subroute_tables[1],
37          "location":subroute_tables[2]
38          }
39      print(subroute_data)
40      producer.send("dok4", json.dumps(subroute_data).encode('utf-8'))
41
42
43  # Close the database connection
44  db.close()
45
```

Εικόνα 11. Κώδικας σε Python του Producer

Σε αυτόν τον κώδικα, συνδέεται πρώτα η βάση δεδομένων MySQL και ανακτά τα δεδομένα τοποθεσίας των οχημάτων από τον πίνακα subroute_table. Στη συνέχεια χρησιμοποιείται η βιβλιοθήκη Kafka Python για να δημιουργηθεί ένα αντικείμενο

KafkaProducer και να στείλει τα δεδομένα των οχημάτων σε ένα θέμα του Kafka που ονομάζεται "dok4". Τέλος, κλείνει η σύνδεση της βάσης δεδομένων (Εικόνα 6).

4.2.γ Καταναλώνοντας δεδομένα από το Kafka (Consumer)

Το επόμενο βήμα για τη δημιουργία του αγωγού της επεξεργασίας των δεδομένων είναι η κατανάλωση των δεδομένων από τον Kafka. Σε αυτό το κομμάτι του κώδικα, θα καταναλώσει τα δεδομένα οχημάτων από το θέμα του Kafka και θα τα ελέγξει για μεγάλους χρόνους αναμονής στην τοποθεσία που βρίσκονται.

```
7
8   import json
9   from kafka import KafkaConsumer
10  import mysql.connector
11
12
13  # Connect to the database
14  db= mysql.connector.connect(
15      host="127.0.0.1",
16      user="root",
17      database="logistic"
18  )
19
20
21
22  # generating the Kafka Consumer
23  my_consumer = KafkaConsumer(
24      'dok4',
25       bootstrap_servers = ['localhost : 9092'],
26       auto_offset_reset = 'earliest',
27       enable_auto_commit = True,
28       group_id = 'my-group',
29       )
30
31
32  locations = {}
33
34  # iterate over messages received from the Kafka topic
35  for myevent in my_consumer:
36      ev = json.loads(myevent.value)
37      print(ev)
38
39      key = f"{ev['route_id']}_{ev['location']}"
40      if key in locations:
41          print(f"Problem: Route_id {ev['route_id']} has been in {ev['location']} for a long time")
42
43          # Update the database with the error flag value of 1
44          cur = db.cursor()
45          cur.execute("UPDATE route_table SET error = 1 WHERE id = %s", (ev['route_id'],))
46          db.commit()
47
48      else:
49          locations[key] = True
50
51
```

Εικόνα 12. Κώδικας σε Python του Consumer

Αυτός ο κώδικας λειτουργεί συνδέοντας πρώτα τη βάση δεδομένων logistic στη MySQL. Στη συνέχεια, δημιουργείται ένας καταναλωτής (consumer) Kafka για να

πληροφορηθεί το θέμα dok4 και να ανακτήσει τα μηνύματα από αυτόν. Ο καταναλωτής (consumer) ελέγχει εάν ένα μήνυμα που λαμβάνεται έχει κλειδί που είναι ήδη αποθηκευμένο στο λεξικό του locations. Εάν το κλειδί υπάρχει ήδη, τότε σημαίνει ότι το όχημα βρίσκεται στο ίδιο σημείο για μεγάλο χρονικό διάστημα. Στη συνέχεια, ο καταναλωτής (consumer) ενημερώνει το route_table στη MySQL αλλάζοντας την τιμή σημαίας σφάλματος από 0 σε 1 για το αντίστοιχο αναγνωριστικό της διαδρομής (Εικόνα 7).

4.3 Σύστημα Διαχείρισης Οχημάτων

Αυτό το κεφάλαιο περιγράφει ένα πρόγραμμα σε γλώσσα Python που χρησιμοποιεί την βιβλιοθήκη MySQL Connector για τη διαχείριση των οχημάτων logistic. Το πρόγραμμα αυτό παρέχει διάφορες λειτουργίες, όπως την προσθήκη ενός νέου οχήματος, την προσθήκη μιας νέας διαδρομής οχήματος, την διαγραφή ενός οχήματος, την διαγραφή μιας διαδρομής οχήματος και την ενημέρωση σφάλματος της διαδρομής.

Το πρόγραμμα ξεκινάει εισάγοντας τη βιβλιοθήκη MySQL Connector και ορίζοντας μια κλάση Vehicle. Η κλάση Vehicle έχει μεθόδους λήψης των δεδομένων του οχήματος, της δημιουργίας ενός νέου οχήματος, την λήψης της διαδρομής του οχήματος, της δημιουργίας μιας νέας διαδρομής οχήματος, της διαγραφής ενός οχήματος, της διαγραφής μιας διαδρομής ενός οχήματος και της ενημέρωσης σφάλματος της διαδρομής.

Για τη λήψη των δεδομένων ενός οχήματος, η συνάρτηση get_vehicle_data χρησιμοποιείται για την ανάκτηση των δεδομένων του οχήματος από τη βάση δεδομένων με βάση το αναγνωριστικό του οχήματος. Η συνάρτηση create_vehicle χρησιμοποιείται για την προσθήκη ενός νέου οχήματος στη βάση δεδομένων. Λαμβάνει τα provider_id, model, licence_plate, occupied_space, total_space, pricce_per_unit και avrg_speed_kmph του οχήματος ως είσοδο και τα εισάγει στον πίνακα του οχήματος (Εικόνα 8).

Η συνάρτηση get_vehicle_route χρησιμοποιείται για την ανάκτηση των δεδομένων διαδρομής ενός οχήματος. Λαμβάνει το vehicle_id ως είσοδο και επιστρέφει τα δεδομένα της διαδρομής (Εικόνα 8).

Η συνάρτηση create_vehicle_route χρησιμοποιείται για την προσθήκη μιας νέας διαδρομής στη βάση δεδομένων. Λαμβάνει τα στοιχεία start_location, end_location, start_date, stop_date και υποδρομολογεί ως είσοδο και τα εισάγει στους route_table και subroute_table (Εικόνα 9).

Η συνάρτηση create_subroute χρησιμοποιείται για την προσθήκη μιας νέας δευτερεύουσας διαδρομής στη βάση δεδομένων. Παίρνει το route_id και την τοποθεσία ως είσοδο και τα εισάγει στον subroute_table (Εικόνα 9).

Η συνάρτηση delete_vehicle χρησιμοποιείται για τη διαγραφή ενός οχήματος από τη βάση δεδομένων. Παίρνει το vehicle_id ως είσοδο και διαγράφει το αντίστοιχο όχημα από τον πίνακα οχημάτων (Εικόνα 9).

Η συνάρτηση delete_route χρησιμοποιείται για τη διαγραφή μιας διαδρομής από τη βάση δεδομένων. Παίρνει το route_id ως είσοδο και διαγράφει την αντίστοιχη διαδρομή από τον route_table (Εικόνα 9).

Η συνάρτηση update_route_error χρησιμοποιείται για την ενημέρωση της κατάστασης σφάλματος μιας διαδρομής στον πίνακα διαδρομής. Παίρνει το route_id ως είσοδο και ορίζει τη σημαία σφάλματος της αντίστοιχης διαδρομής σε 1 (Εικόνα 9).

Το πρόγραμμα παρέχει ένα μενού για το χρήστη για την πρόσβαση σε αυτές τις λειτουργίες. Ξεκινά εμφανίζοντας ένα μενού με τις ακόλουθες επιλογές: Προσθήκη νέου οχήματος, Προσθήκη νέας διαδρομής οχήματος, Διαγραφή οχήματος, Διαγραφή διαδρομής οχήματος, Δήλωση βλάβης και Έξοδος. Ο χρήστης μπορεί να επιλέξει μια επιλογή εισάγοντας τον αντίστοιχο αριθμό (Εικόνα 9).

Επιλέγοντας ο χρήστης την επιλογή Προσθήκη νέου οχήματος, το πρόγραμμα ζητά από τον χρήστη να εισαγάγει τα provider_id, model, licence_plate, occupied_space, total_space, pricce_per_unit και avrg_speed_kmph του οχήματος. Στη συνέχεια, το πρόγραμμα καλεί την συνάρτηση create_vehicle για να προσθέσει το νέο όχημα στη βάση δεδομένων (Εικόνα 10).

Επιλέγοντας ο χρήστης την επιλογή Προσθήκη νέας διαδρομής οχήματος, το πρόγραμμα ζητά από τον χρήστη να εισαγάγει την start_location, end_location, start_date, stop_date και subroute της διαδρομής. Στη συνέχεια, το πρόγραμμα καλεί την συνάρτηση create_vehicle_route για να προσθέσει τη νέα διαδρομή στη βάση δεδομένων (Εικόνα 10).

Επιλέγοντας ο χρήστης την επιλογή Διαγραφή οχήματος, το πρόγραμμα ζητά από τον χρήστη να εισαγάγει το licence_plate (πινακίδα) του οχήματος που πρόκειται να διαγραφεί. Στη συνέχεια, το πρόγραμμα καλεί την συνάρτηση delete_vehicle για να διαγράψει το όχημα από τη βάση δεδομένων (Εικόνα 10).

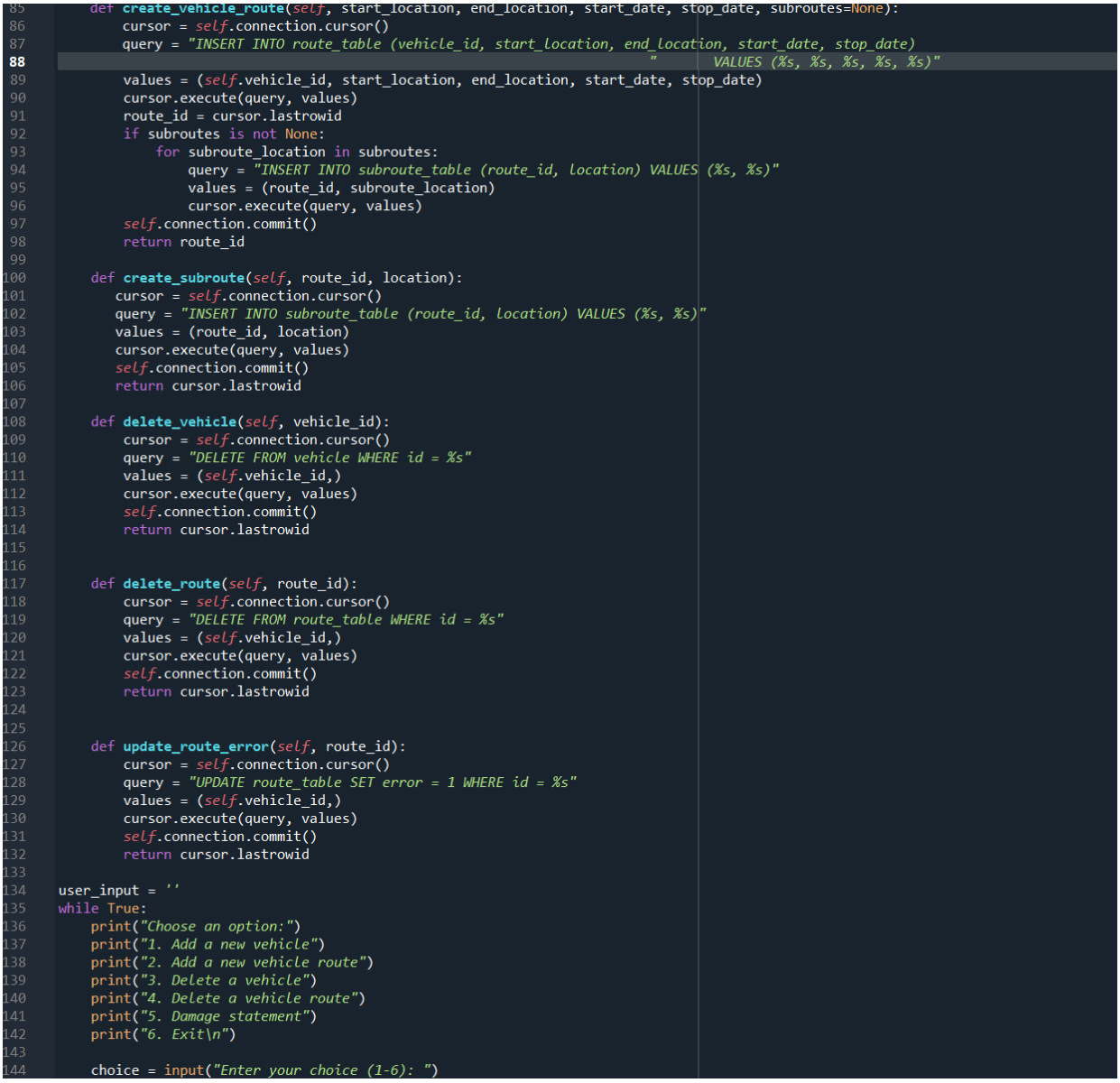
Επιλέγοντας ο χρήστης την επιλογή Διαγραφή διαδρομής οχήματος, το πρόγραμμα ζητά από τον χρήστη να εισαγάγει το route_id της διαδρομής που πρόκειται να διαγραφεί. Στη συνέχεια, το πρόγραμμα καλεί τη συνάρτηση delete_route για να διαγράψει τη διαδρομή από τη βάση δεδομένων (Εικόνα 10).

Επιλέγοντας ο χρήστης την επιλογή δήλωσης βλάβης, το πρόγραμμα παρέχει έναν τρόπο ενημέρωσης της κατάστασης σφάλματος μιας διαδρομής. Το πρόγραμμα ζητά από το χρήστη να εισάγει το route_id της διαδρομής που έχει σφάλμα. Στη συνέχεια, το πρόγραμμα καλεί την συνάρτηση update_route_error για να ορίσει τη σημαία σφάλματος της αντίστοιχης διαδρομής σε 1 (Εικόνα 10 Vehicle3).

Τέλος επιλέγοντας ο χρήστης την επιλογή εξόδου από το πρόγραμμα, το πρόγραμμα θα κλείσει (Εικόνα 10).

```python

import mysql.connector

class Vehicle:
    def __init__(self, vehicle_id):
        self.vehicle_id = vehicle_id
        self.connection = mysql.connector.connect(
            host="127.0.0.1",
            user="root",
            database="logistic"
        )

    def get_vehicle_data(self):
        cursor = self.connection.cursor()
        query = "SELECT provider_id,model, licence_plate, occupied_space,total_space,price_per_unit,avrg_speed_kmph
                                                                                "       FROM vehicle WHERE id = %s"
        values = (self.vehicle_id,)
        cursor.execute(query, values)
        result = cursor.fetchone()
        provider_id,model, licence_plate, occupied_space,total_space,price_per_unit,avrg_speed_kmph = result
        return {
            "provider_id": provider_id,
            "model": model,
            "licence_plate": licence_plate,
            "occupied_space": occupied_space,
            "total_space" : total_space,
            "price_per_unit" : price_per_unit,
            "avrg_speed_kmph": avrg_speed_kmph
        }



    def create_vehicle(self, provider_id, model, licence_plate, occupied_space, total_space, price_per_unit, avrg_speed_kmph):
        cursor = self.connection.cursor()
        query = "INSERT INTO vehicle (provider_id, model, licence_plate, occupied_space, total_space, price_per_unit,
                                                                "     avrg_speed_kmph) VALUES (%s, %s, %s, %s, %s, %s, %s)"
        values = (provider_id, model, licence_plate, occupied_space, total_space, price_per_unit, avrg_speed_kmph)
        cursor.execute(query, values)
        self.connection.commit()
        return cursor.lastrowid



    def get_vehicle_route(self):
        cursor = self.connection.cursor()
        query = "SELECT id, start_location, end_location, start_date, stop_date FROM route_table WHERE vehicle_id = %s"
        values = (self.vehicle_id,)
        cursor.execute(query, values)
        result = cursor.fetchall()
        route_data = []
        for row in result:
            route_id, start_location, end_location, start_date, stop_date = row
            cursor = self.connection.cursor()
            query = "SELECT location FROM subroute_table WHERE route_id = %s"
            values = (route_id,)
            cursor.execute(query, values)
```

Εικόνα 13. Κώδικας σε Python του Vehicle1



```
    def create_vehicle_route(self, start_location, end_location, start_date, stop_date, subroutes=None):
        cursor = self.connection.cursor()
        query = "INSERT INTO route_table (vehicle_id, start_location, end_location, start_date, stop_date)
                                                                                "      VALUES (%s, %s, %s, %s, %s)"
        values = (self.vehicle_id, start_location, end_location, start_date, stop_date)
        cursor.execute(query, values)
        route_id = cursor.lastrowid
        if subroutes is not None:
            for subroute_location in subroutes:
                query = "INSERT INTO subroute_table (route_id, location) VALUES (%s, %s)"
                values = (route_id, subroute_location)
                cursor.execute(query, values)
        self.connection.commit()
        return route_id

    def create_subroute(self, route_id, location):
       cursor = self.connection.cursor()
       query = "INSERT INTO subroute_table (route_id, location) VALUES (%s, %s)"
       values = (route_id, location)
       cursor.execute(query, values)
       self.connection.commit()
       return cursor.lastrowid

    def delete_vehicle(self, vehicle_id):
        cursor = self.connection.cursor()
        query = "DELETE FROM vehicle WHERE id = %s"
        values = (self.vehicle_id,)
        cursor.execute(query, values)
        self.connection.commit()
        return cursor.lastrowid


    def delete_route(self, route_id):
        cursor = self.connection.cursor()
        query = "DELETE FROM route_table WHERE id = %s"
        values = (self.vehicle_id,)
        cursor.execute(query, values)
        self.connection.commit()
        return cursor.lastrowid


    def update_route_error(self, route_id):
        cursor = self.connection.cursor()
        query = "UPDATE route_table SET error = 1 WHERE id = %s"
        values = (self.vehicle_id,)
        cursor.execute(query, values)
        self.connection.commit()
        return cursor.lastrowid

user_input = ''
while True:
    print("Choose an option:")
    print("1. Add a new vehicle")
    print("2. Add a new vehicle route")
    print("3. Delete a vehicle")
    print("4. Delete a vehicle route")
    print("5. Damage statement")
    print("6. Exit\n")

    choice = input("Enter your choice (1-6): ")
```

Εικόνα 14. Κώδικας σε Python του Vehicle2

```
    if choice == "1":
        # Add a new vehicle
        vehicle = Vehicle(1)
        new_vehicle_id = vehicle.create_vehicle(
            provider_id = input("Enter the provider_id of the vehicle: "),
            model = input("Enter the model of the vehicle: "),
            licence_plate = input("Enter the licence_plate of the vehicle: "),
            occupied_space = input("Enter the occupied_space of the vehicle: "),
            total_space = input("Enter the total_space of the vehicle: "),
            price_per_unit = input("Enter the price_per_unit of the vehicle: "),
            avrg_speed_kmph = input("Enter the avrg_speed_kmph of the vehicle: ")
            )
        print(f"Vehicle added with ID {new_vehicle_id}\n")

    elif choice == "2":
        # Add a new vehicle route
        vehicle = Vehicle(input("Enter the id_vehicle who take the route: "))
        new_route_id = vehicle.create_vehicle_route(
            start_location = input("Enter the starting location of the route: "),
            end_location = input("Enter the ending location of the route: "),
            start_date = input("Enter the start date of the route (YYYY-MM-DD): "),
            stop_date = input("Enter the stop date of the route (YYYY-MM-DD): ")
            )
        print(f"Vehicle route added with ID {new_route_id}\n")

    elif choice == "3":
        # Delete a vehicle
        vehicle_id = input("Enter vehicle ID: ")
        vehicle = Vehicle(vehicle_id,)
        vehicle.delete_vehicle(id)
        print(f"Vehicle with ID {vehicle} deleted\n")



    elif choice == "4":
        # Delete a vehicle route
        route_id = input("Enter route ID: ")
        route=Vehicle(route_id,)
        route.delete_route(id)
        print(f"Vehicle route with ID {route_id} deleted\n")


    elif choice == "5":
        # Damage statement
        route_id = input("Enter route ID: ")
        route=Vehicle(route_id,)
        route.update_route_error(id)
        print(f"Vehicle route with ID {route_id} has Damage statement\n")

    elif choice == "6":
        # Exit
        break

    else:
        print("Invalid choice. Please try again.\n")
```

Εικόνα 15. Κώδικας σε Python του Vehicle3

4.4 Σύστημα Αναφοράς Σφαλμάτων

Σε αυτό το κεφάλαιο περιγράφεται η υλοποίηση ενός συστήματος αναφοράς σφαλμάτων ενός οχήματος χρησιμοποιώντας την γλώσσα προγραμματισμού Python, την βάση δεδομένων MySQL, το API Mapbox και την βιβλιοθήκη Folium. Το σύστημα έχει σχεδιαστεί για να ανιχνεύει τα σφάλματα σε διαδρομές των οχημάτων, να κωδικοποιεί τη θέση του σφάλματος και να εμφανίζει τις κοντινές διαδρομές που συνέβησαν εντός των 30 λεπτών και σε απόσταση μικρότερη ή ίση των 200 km από τη θέση σφάλματος σε έναν διαδραστικό χάρτη.

Η διαδικασία της υλοποίησης περιλαμβάνει τη σύνδεση σε μια βάση δεδομένων MySQL, την ανάκτηση των αναγνωριστικών σφαλμάτων από τον πίνακα των διαδρομών, την επεξεργασία κάθε αναγνωριστικού σφάλματος και την ανάκτηση του ονόματος της πόλης, της χρονικής στιγμής και της θέσης άφιξης από τον πίνακα subroute. Στη συνέχεια, το όνομα της πόλης κωδικοποιείται γεωγραφικά χρησιμοποιώντας το Mapbox API για να ληφθούν οι τιμές του γεωγραφικού πλάτους και μήκους και τότε δημιουργείται ένας χάρτης με κέντρο την τοποθεσία χρησιμοποιώντας τη βιβλιοθήκη Folium. Ένας δείκτης με ένα εικονίδιο αυτοκινήτου προστίθεται στον χάρτη για να υποδείξει τη θέση του σφάλματος.

Στη συνέχεια, το σύστημα ανακτά τις τοποθεσίες και τις χρονικές σημάνσεις για άλλες διαδρομές που πραγματοποιήθηκαν μέσα σε 30 λεπτά και σε απόσταση μικρότερη ή ίση των 200 km από τη θέση σφάλματος, γεωκωδικοποιεί τις τοποθεσίες χρησιμοποιώντας το Mapbox API και προσθέτει δείκτες με άλλα εικονίδια οχημάτων στον χάρτη. Το σύστημα υπολογίζει επίσης την απόσταση μεταξύ της θέσης σφάλματος και κάθε κοντινής τοποθεσίας χρησιμοποιώντας τη βιβλιοθήκη Geopy και εκτυπώνει ένα μήνυμα στην κονσόλα εάν βρεθεί ένα κοντινό όχημα.

Τέλος, το σύστημα θα αποθηκεύσει τον χάρτη ως αρχείο HTML και θα το ανοίξει στο προεπιλεγμένο πρόγραμμα περιήγησης του ιστού χρησιμοποιώντας τη βιβλιοθήκη του λειτουργικού συστήματος όπως φαίνεται στις παρακάτω δύο εικόνες (Εικόνα 11) και (Εικόνα 12). Η εφαρμογή αυτού του συστήματος δίνει μια πρακτική λύση για την αναφορά των σφαλμάτων οχημάτων, η οποία μπορεί να βοηθήσει τις εταιρείες logistics να βελτιώσουν το σχεδιασμό της διαδρομής τους και να μειώσουν τις καθυστερήσεις της παράδοσης.

```python

import mysql.connector
from mapbox import Geocoder
import folium
import os
from geopy import distance
from datetime import datetime, timedelta

# replace YOUR_API_KEY with your actual Mapbox API key
api_key = 'pk.eyJ1IjoiZGltaXRyaXMtLXBwIiwiYSI6ImNsZThsMTVnODBmdWczb3J6cmNqenBwcjcifQ.mKZYQJXvBoyjJgJ7Ul1YjQ'
geocoder = Geocoder(access_token=api_key)

# connect to MySQL database
db = mysql.connector.connect(
    host="127.0.0.1",
    user="root",
    database="logistic"
)

# Retrieve error IDs from the database
cursor = db.cursor()
# fetch the error IDs from the database
error_id_query = "SELECT id FROM route_table WHERE error = 1"
cursor.execute(error_id_query)
error_ids = cursor.fetchall()

# process each error ID
for error_id in error_ids:
    error_id = error_id[0]

    # fetch the town name and timestamp from the database
    subroute_query = "SELECT location, date, end_location FROM subroute_table JOIN route_table
                      ON subroute_table.route_id = route_table.id WHERE route_table.id = %s ORDER BY date DESC LIMIT 1"
    cursor.execute(subroute_query, (error_id,))
    result = cursor.fetchone()

    if result:
        town_name = result[0]
        timestamp = result[1]
        error_end_location = result[2]

        # geocode the town name using Mapbox
        response = geocoder.forward(town_name)
        if response.status_code == 200:
            features = response.json()['features']
            if len(features) > 0:
                longitude, latitude = features[0]['center']
                print(town_name, latitude, longitude)

                # create a map centered at the latitude and longitude values
                m = folium.Map(location=[latitude, longitude], zoom_start=13, control_scale=True)

                # add a marker for the location with car icon
                icon_url = 'C:/Users/kilaf/logistics/errorv.png'
                icon_size = (50, 50)
                icon_anchor = (15, 15)
                folium.Marker([latitude, longitude], popup=town_name, icon=folium.features.CustomIcon(icon_url,
                                                                    icon_size=icon_size, icon_anchor=icon_anchor)).add_to(m)


                # get the locations and timestamps for other routes that occurred within 30 minutes
                nearby_query = "SELECT location, date, route_table.end_location FROM subroute_table JOIN route_table
                                ON subroute_table.route_id = route_table.id WHERE route_table.id != %s AND
                                ABS(TIMESTAMPDIFF(MINUTE, %s, date)) <= 30 ORDER BY date"
                cursor.execute(nearby_query, (error_id, timestamp))
                nearby_results = cursor.fetchall()
```

Εικόνα 16. Κώδικας σε Python του συστήματος αναφοράς σφαλμάτων1

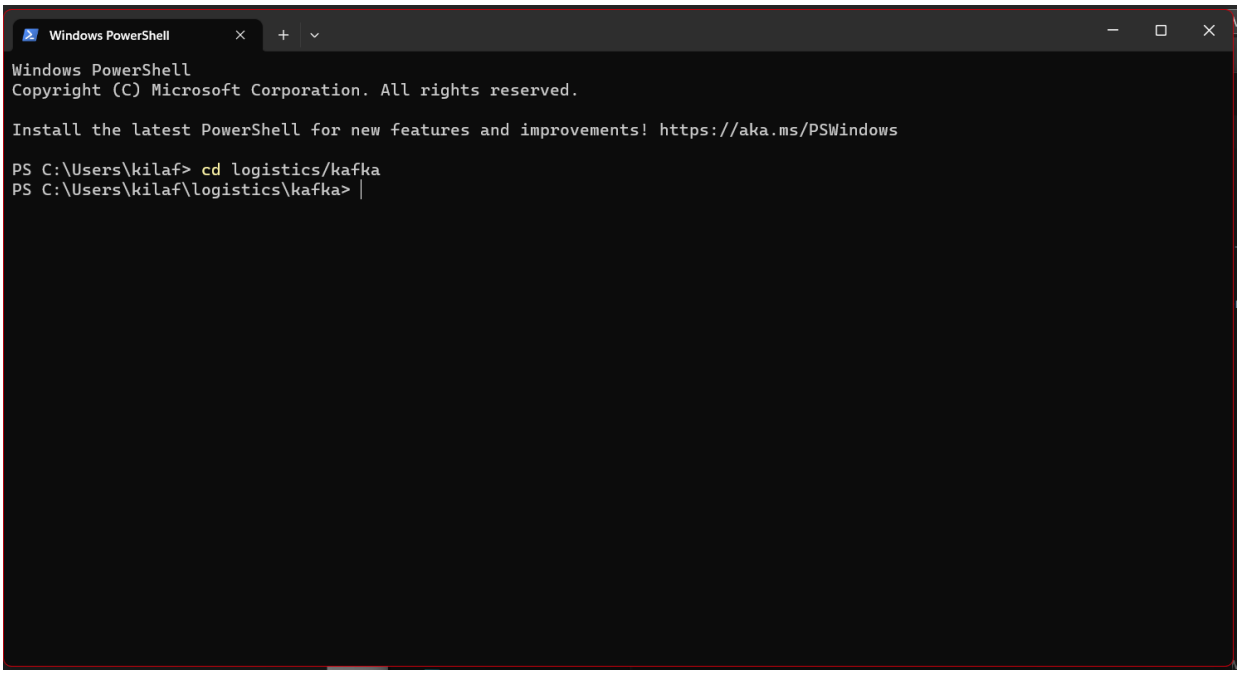
```
73
74              # geocode the locations and add markers to the map
75              for row in nearby_results:
76                  location = row[0]
77                  timestamp1 = row[1]
78                  end_location = row[2]
79                  response = geocoder.forward(location)
80                  response2 = geocoder.forward(end_location)
81
82                  if response.status_code == 200 and response2.status_code == 200:
83                      features = response.json()['features']
84                      features2 = response2.json()['features']
85
86                      # check if the distance is less than or equal to 200 km
87                      dist = distance.distance(features[0]['geometry']['coordinates'][::-1],
88                                               features2[0]['geometry']['coordinates'][::-1]).km
89                      if dist <= 200:
90                          longitude, latitude = features[0]['center']
91
92                          # add a marker for the location
93                          icon_url = 'C:/Users/kilaf/logistics/otherv.png'
94                          icon_size = (50, 50)
95                          icon_anchor = (15, 15)
96                          folium.Marker([latitude, longitude], popup=location,
97                                        icon=folium.features.CustomIcon(icon_url,
98                                                                        icon_size=icon_size, icon_anchor=icon_anchor)).add_to(m)
99                          print(f"Vehicle at location {location} is nearby")
100
101
102             # save the map and open it in the default web browser
103             m.save("mymap.html")
104             os.startfile("C:/Users/kilaf/logistics/mymap.html")
105
106             # fetch all the remaining results from the cursor object
```

Εικόνα 17. Κώδικας σε Python του συστήματος αναφοράς σφαλμάτων2

ΚΕΦΑΛΑΙΟ 5 CASE STUDY

5.1 Ρύθμιση Kafka

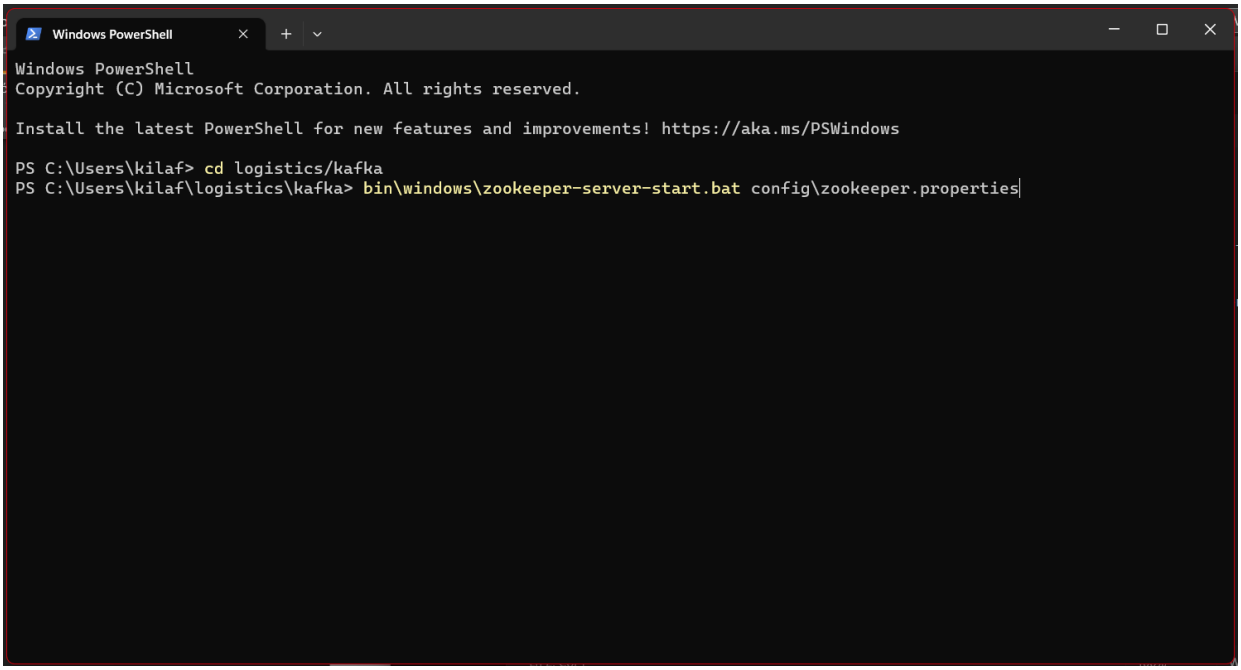
Σε αυτό το κεφάλαιο, θα ρυθμιστεί ο Kafka για το Σύστημα Διαχείρισης Οχημάτων (Εικόνα 13).



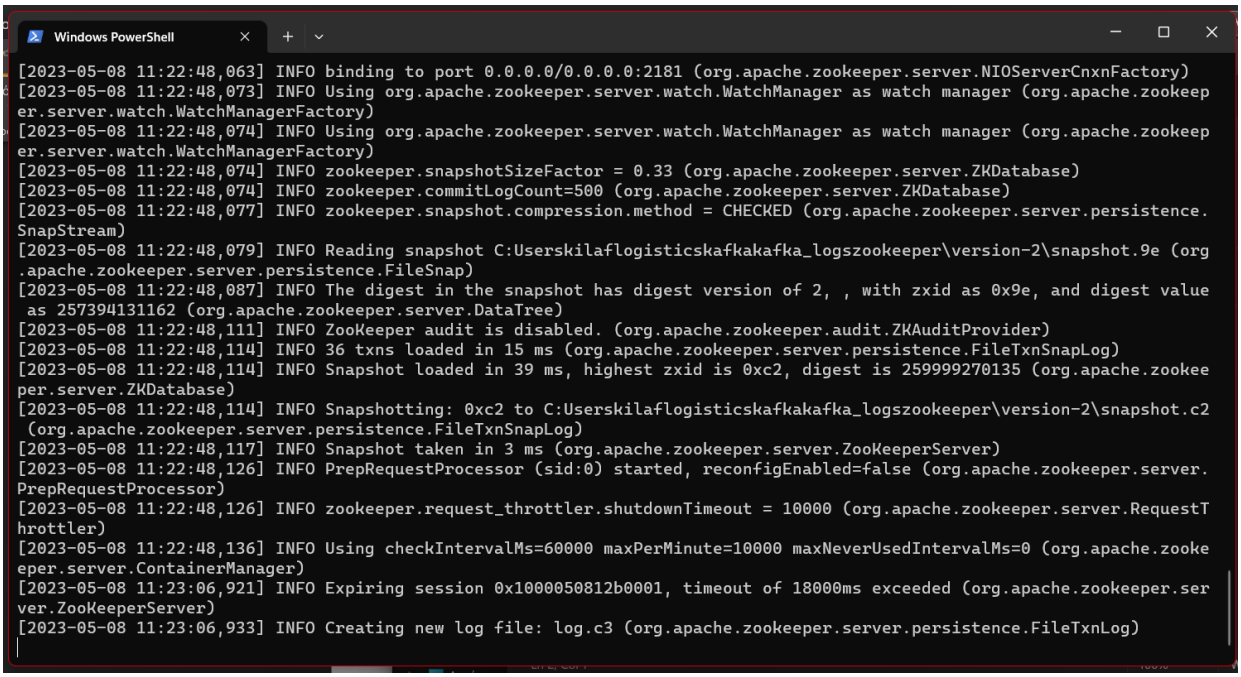
Εικόνα 18. Εισαγωγή στον φάκελο του Kafka

Producer

Αυτή η εντολή ξεκινά τον διακομιστή ZooKeeper χρησιμοποιώντας το αρχείο διαμόρφωσης zookeeper.properties (Εικόνα 14). Το ZooKeeper είναι μια κεντρική υπηρεσία που χρησιμοποιείται για τη διαχείριση και τον συντονισμό των μεσιτών Kafka σε ένα σύμπλεγμα (Εικόνα 15).

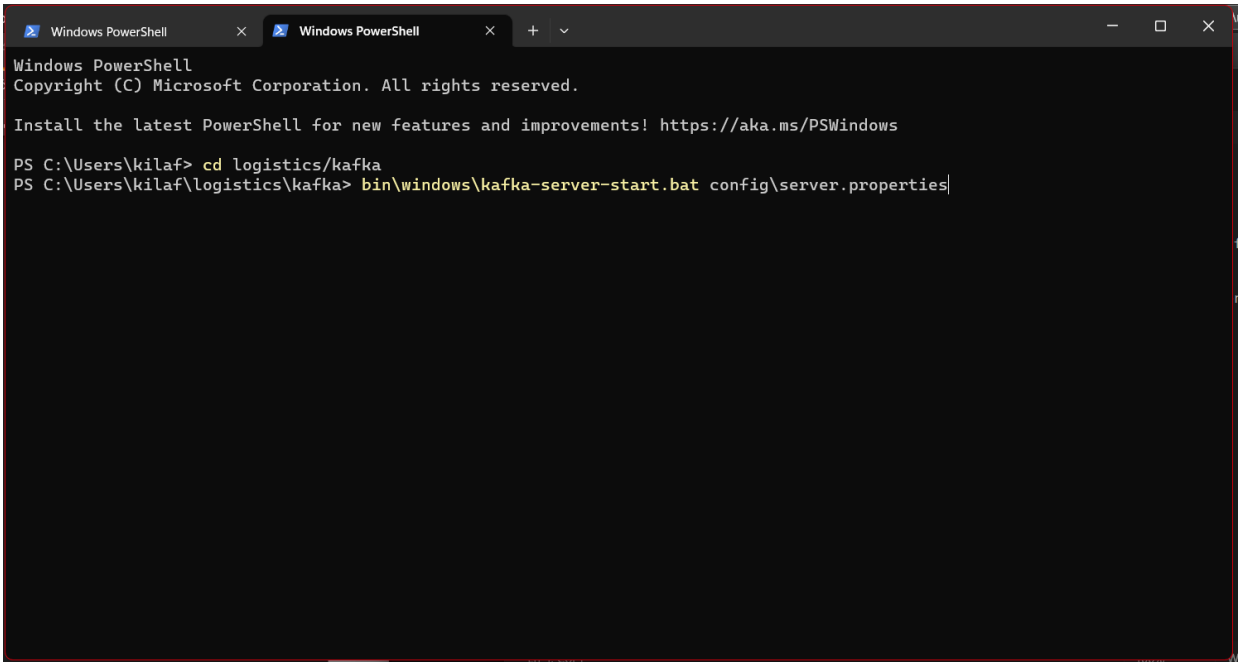


Εικόνα 19. Εντολή για σύνδεση με το Zookeeper

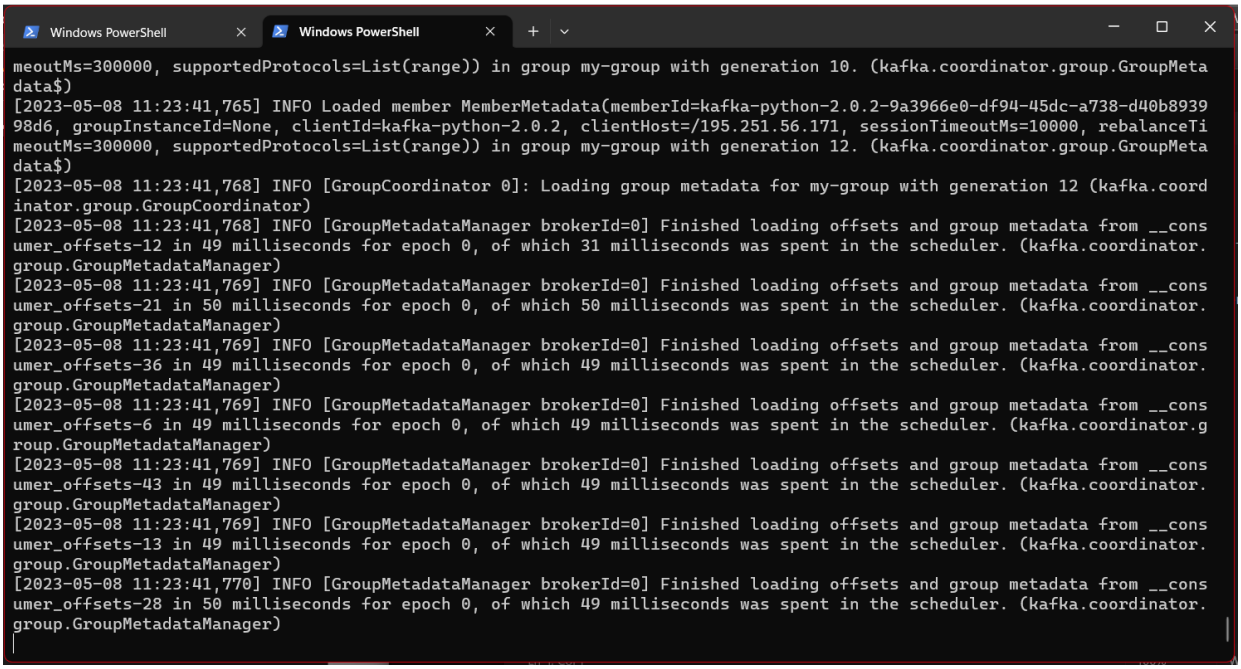


Εικόνα 20. Εντολές που τρέχει το zookeeper

Αυτή η εντολή ξεκινά τον διακομιστή Kafka χρησιμοποιώντας το αρχείο διαμόρφωσης server.properties (Εικόνα 16). Ο διακομιστής Kafka είναι υπεύθυνος για τη διαχείριση θεμάτων, την αποθήκευση μηνυμάτων και τον συντονισμό της παράδοσης μηνυμάτων μεταξύ παραγωγών και καταναλωτών (Εικόνα 17).



Εικόνα 21. Εντολή για σύνδεση με τον Server

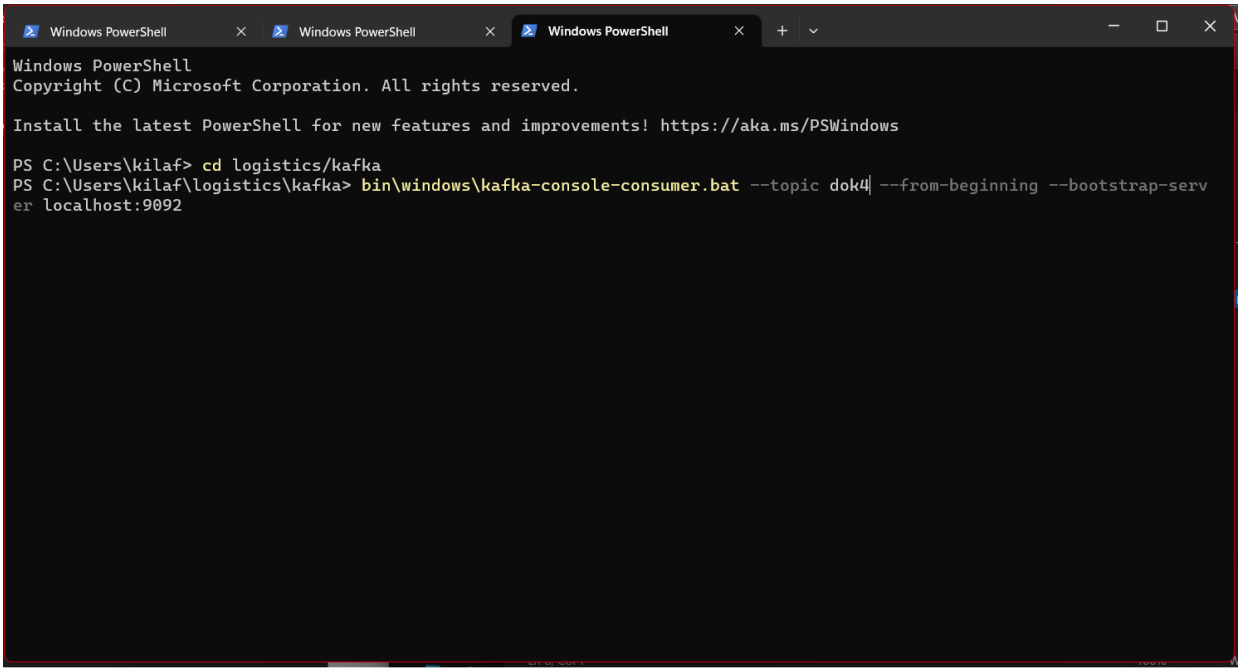


Εικόνα 22. Εντολές που τρέχει ο Server

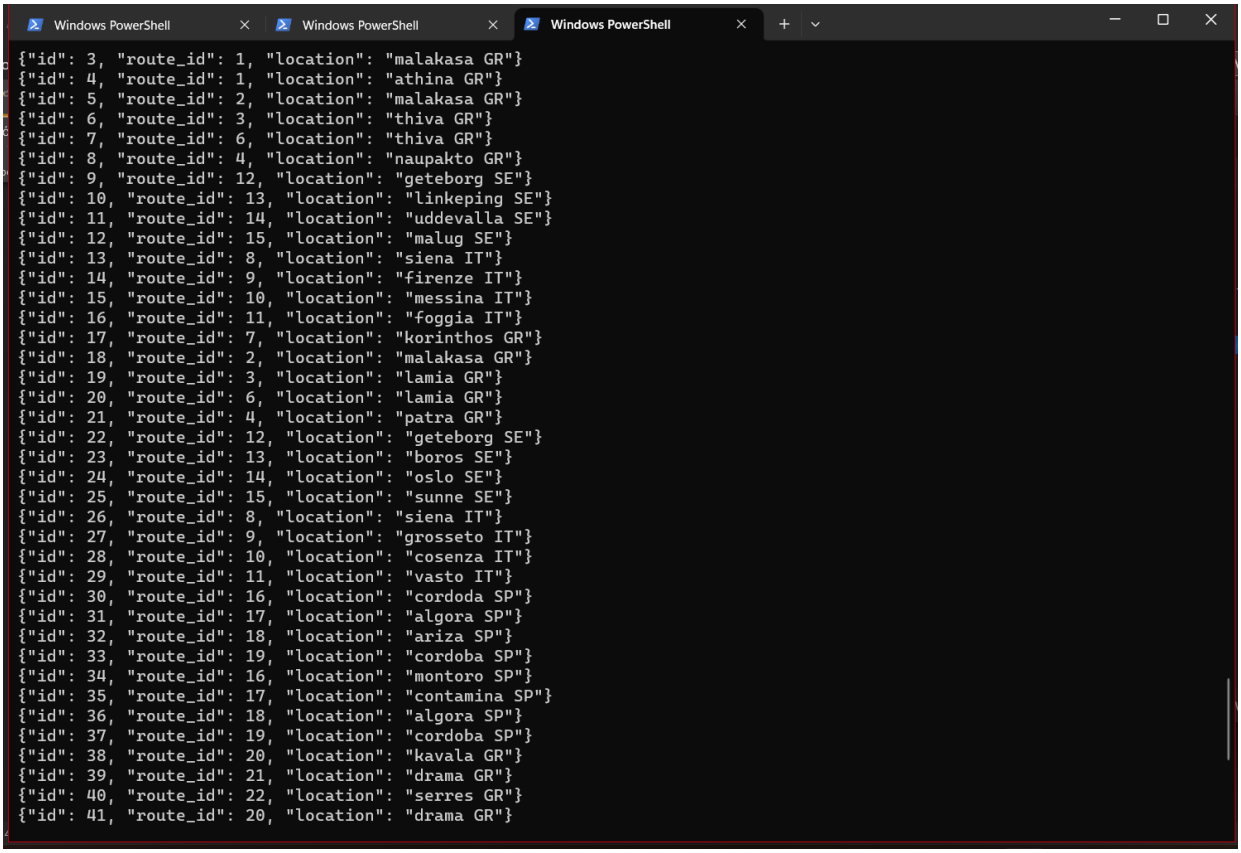
Consumer

Αυτή η εντολή ξεκινά έναν καταναλωτή Kafka χρησιμοποιώντας το kafka-console-consumer.bat σενάριο. Ο καταναλωτής διαβάζει μηνύματα από το θέμα dok4 και τα εξάγει στην κονσόλα (Εικόνα 18). Η --from-beginning επιλογή λέει στον καταναλωτή

να αρχίσει να διαβάζει μηνύματα από την αρχή του θέματος και όχι από το σημείο στο οποίο ξεκίνησε ο καταναλωτής (Εικόνα 19).



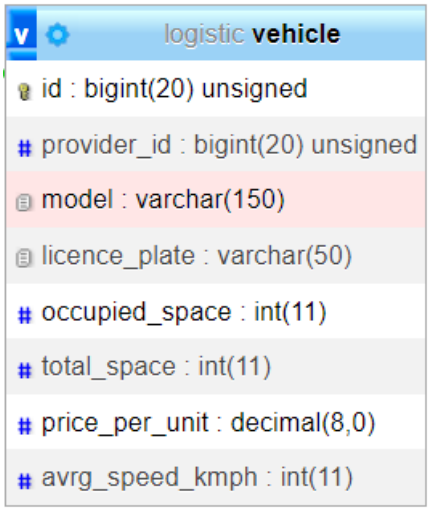
Εικόνα 23. Εντολή για σύνδεση με τον Consumer



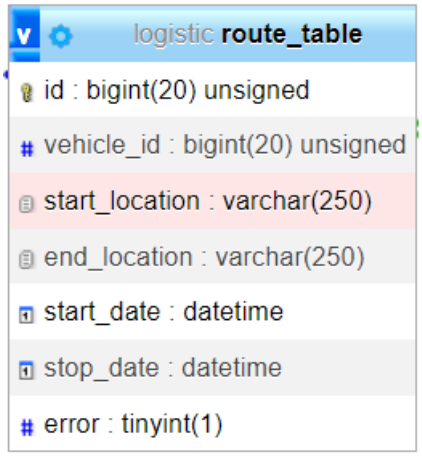
Εικόνα 24. Λίστα μηνυμάτων που διαβάζει ο Consumer

5.2 Ρύθμιση MySQL

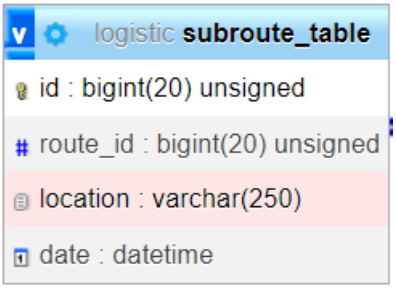
Σε αυτή την παράγραφο, θα συζητηθεί η διαμόρφωση της MySQL για το Σύστημα Διαχείρισης Οχημάτων. Θα δημιουργηθεί μια λογιστική βάση δεδομένων logistic και θα καθοριστεί η δομή των τριών πινάκων, δηλαδή του vehicle (Εικόνα 20), του route_table (Εικόνα 21) και του subroute_table (Εικόνα 22). Κάθε πίνακας θα έχει δηλωμένα τα στοιχεία και τους τύπους των δεδομένων του. Στη συνέχεια, οι πίνακες θα ενωθούν με βάση τα αντίστοιχα πεδία τους (Εικόνα 23). Αυτό θα δημιουργήσει τις σχέσεις μεταξύ του πίνακα vehicle, route_table και subroute_table, επιτρέποντας την δραστική ανάκτηση και διαχείριση των δεδομένων.



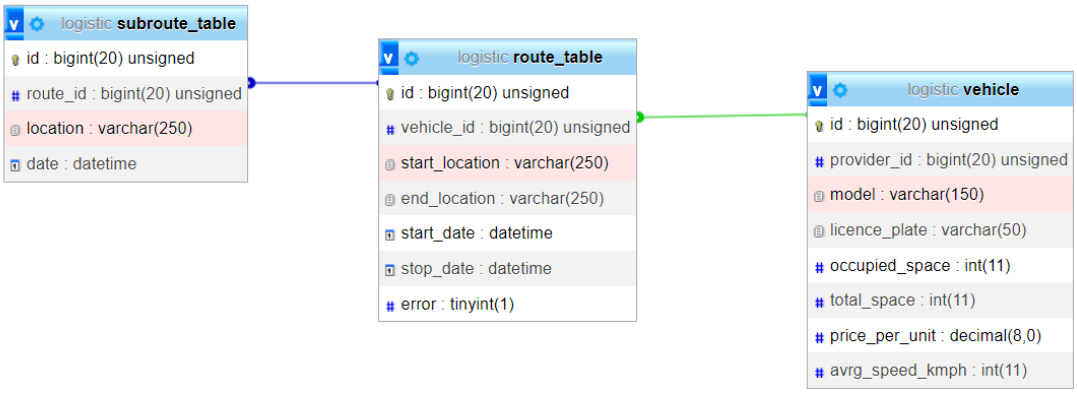
Εικόνα 25. Πίνακας βάσης οχήματος



Εικόνα 26. Πίνακας βάσης διαδρομής



Εικόνα 27. Πίνακας βάσης τοποθεσίας

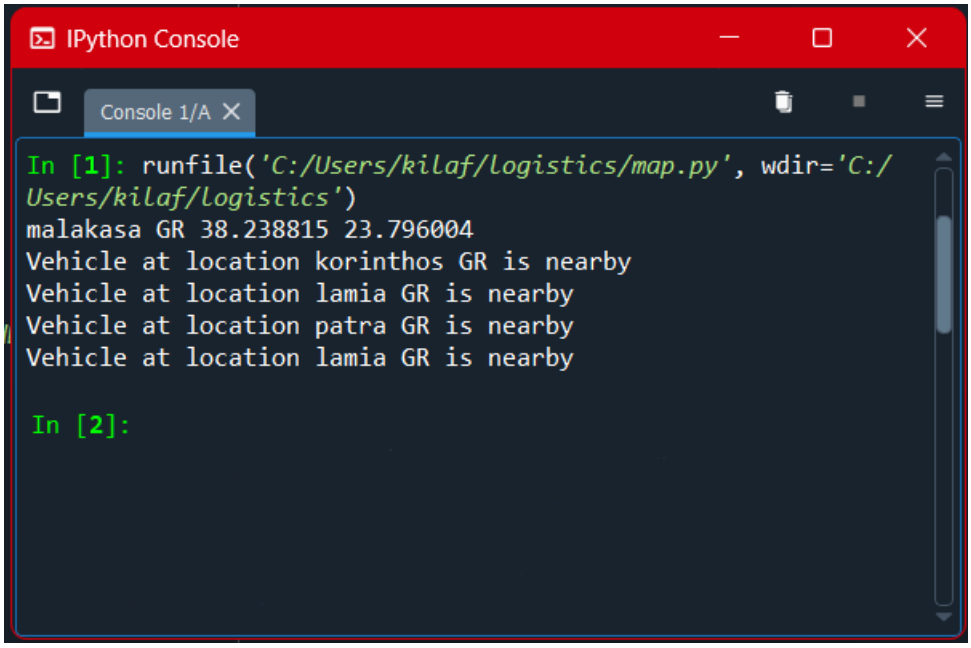


Εικόνα 28. Σύνδεση πινάκων

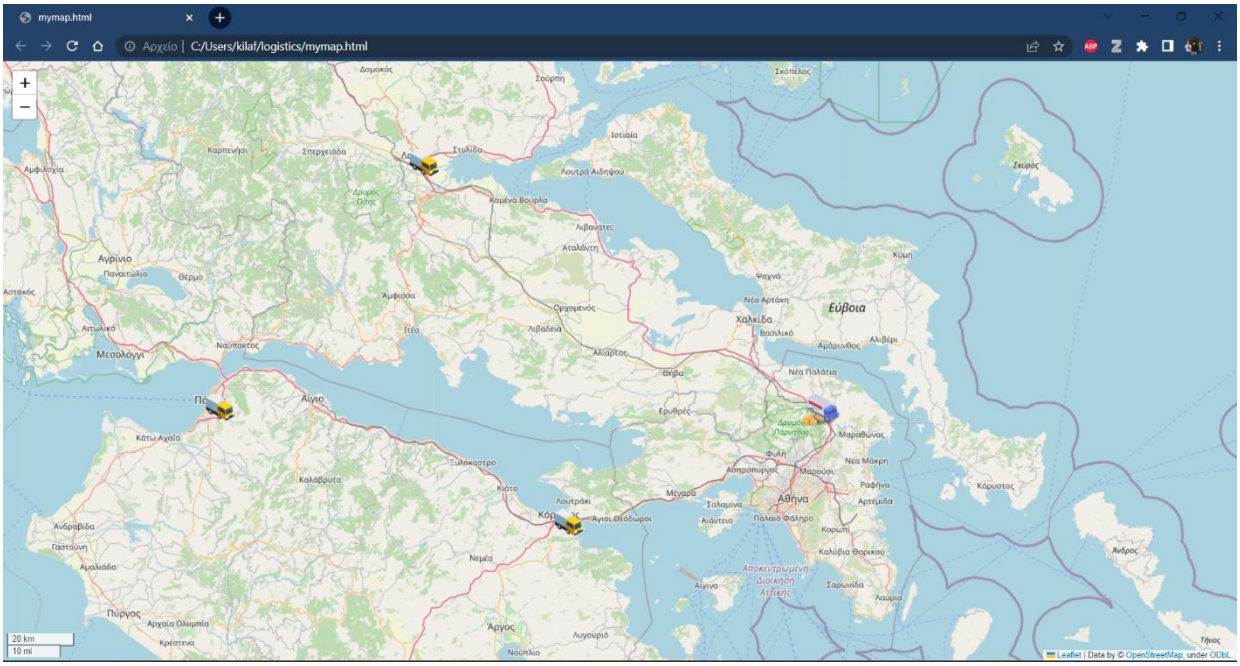
5.3 Σενάρια

Παράδειγμα 1

Στο πρώτο υποτιθέμενο σενάριο, ένα όχημα με πινακίδα κυκλοφορίας «sdf_3456» διαπιστώθηκε πως βρίσκεται για πολύ ώρα στο ίδιο σημείο. Το περιστατικό σημειώθηκε στη Μαλακάσα της Ελλάδας. Η τοποθεσία της βλάβης αναφέρθηκε στο κέντρο, δίνοντας το πρόγραμμα τις ακριβείς συντεταγμένες του οχήματος. Το κέντρο ξεκίνησε αμέσως τη διαδικασία εντοπισμού των κοντινών οχημάτων που βρίσκονται σε απόσταση των 200 χλμ. την τελευταία μισή ώρα και δημιουργήθηκε μια λίστα (Εικόνα 24) με τα κοντινά οχήματα οπού εμφανίζονται στον χάρτη (Εικόνα 25).



Εικόνα 29. Αποτελέσματα Console1

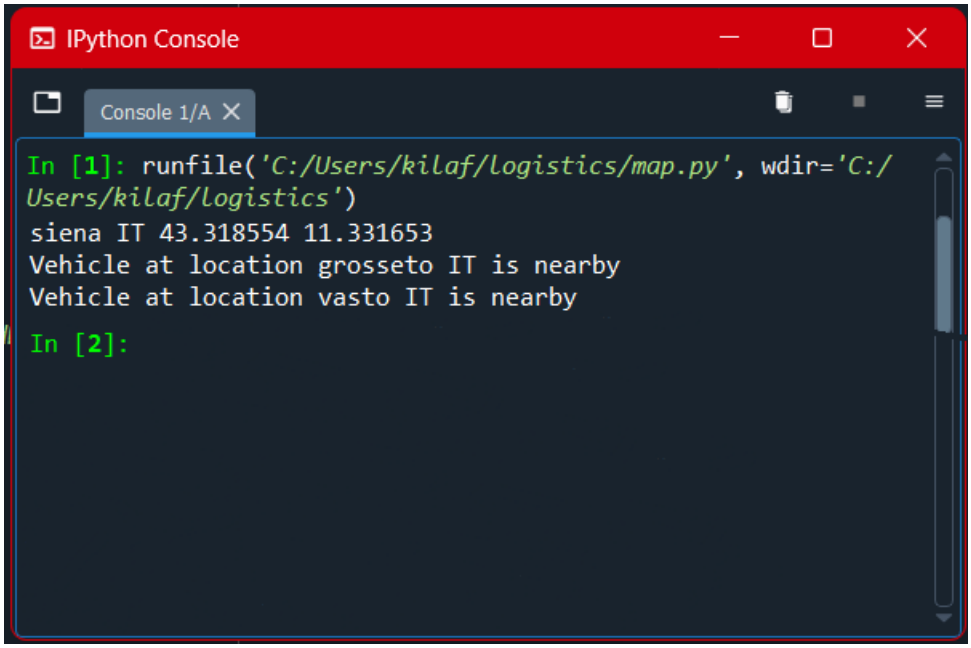


Εικόνα 30. Χάρτης Συμβάντος Map1

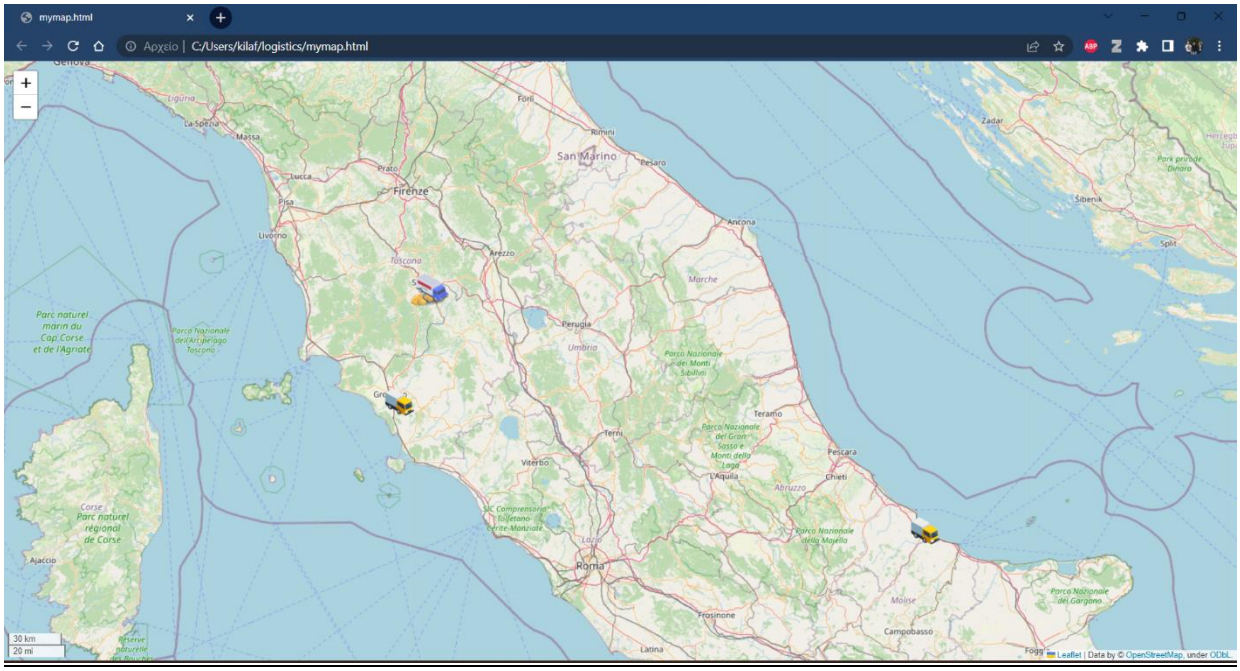
Παράδειγμα 2

Στο δεύτερο υποτιθέμενο σενάριο, ένα όχημα με πινακίδα κυκλοφορίας «rfv-5432» διαπιστώθηκε πως βρίσκεται για πολύ ώρα στο ίδιο σημείο. Το περιστατικό σημειώθηκε στη Siena της Ιταλίας. Η τοποθεσία της βλάβης αναφέρθηκε στο κέντρο, δίνοντας το πρόγραμμα τις ακριβείς συντεταγμένες του οχήματος. Το κέντρο ξεκίνησε αμέσως τη διαδικασία εντοπισμού των κοντινών οχημάτων που βρίσκονται σε

απόσταση των 200 χλμ. την τελευταία μισή ώρα και δημιουργήθηκε μια λίστα (Εικόνα 26) με τα κοντινά οχήματα οπού εμφανίζονται στον χάρτη (Εικόνα 27).



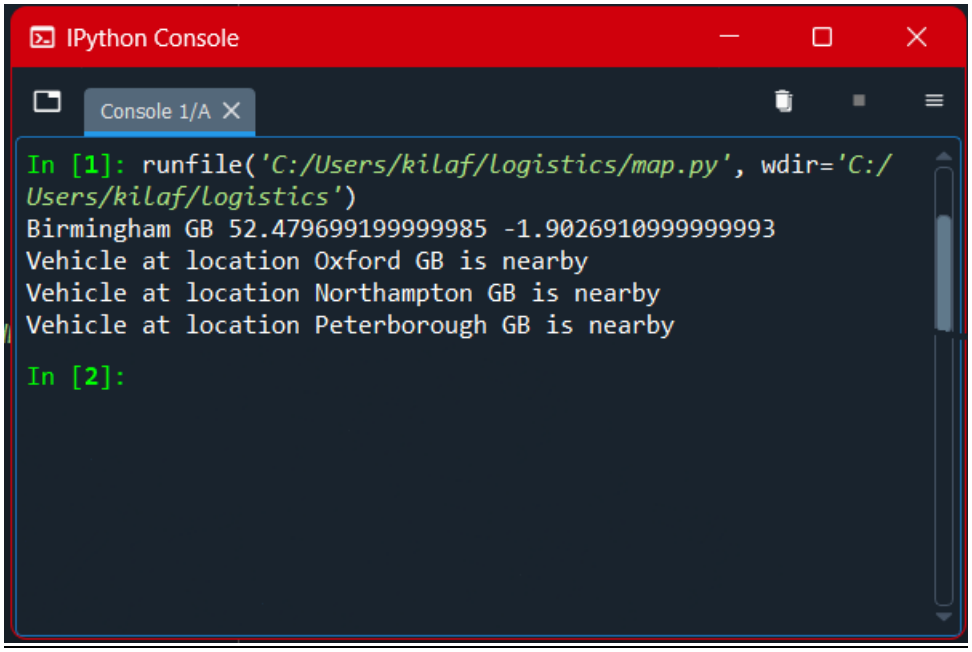
Εικόνα 31. Αποτελέσματα Console2



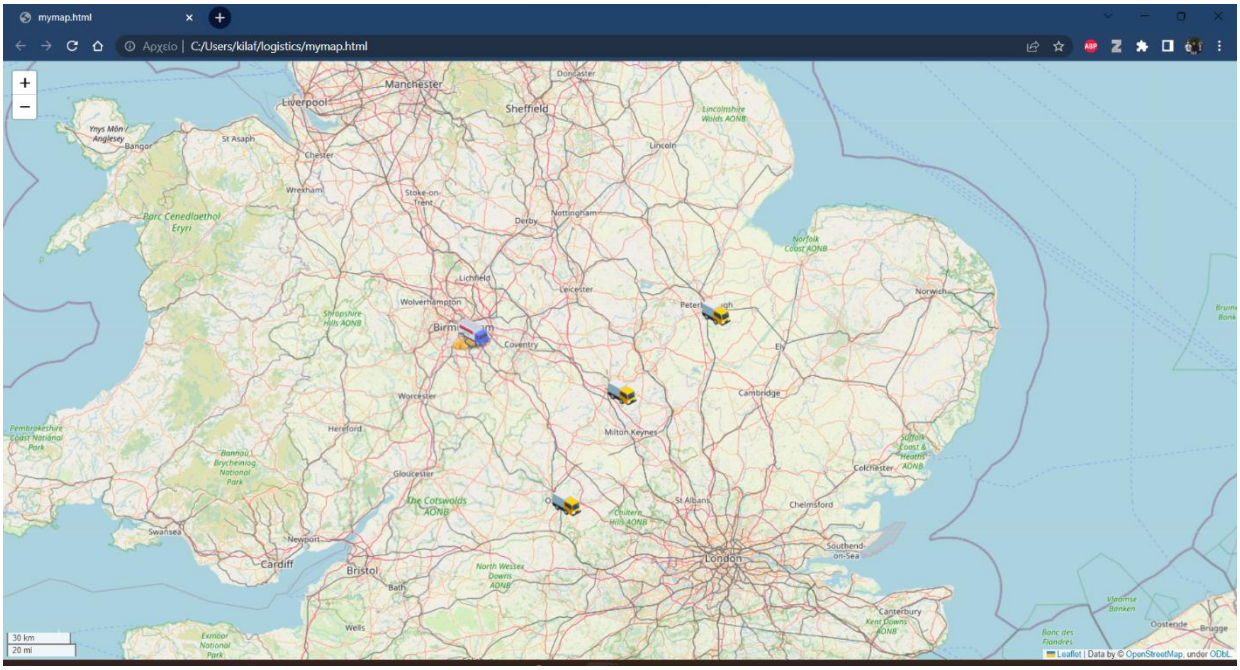
Εικόνα 32. Χάρτης Συμβάντος Map2

Παράδειγμα 3

Στο τρίτο υποτιθέμενο σενάριο, ένα όχημα με πινακίδα κυκλοφορίας «rfv-5432» διαπιστώθηκε πως βρίσκεται για πολύ ώρα στο ίδιο σημείο. Το περιστατικό σημειώθηκε στο Birmingham του Ηνωμένου Βασίλειου. Η τοποθεσία της βλάβης αναφέρθηκε στο κέντρο, δίνοντας το πρόγραμμα τις ακριβείς συντεταγμένες του οχήματος. Το κέντρο ξεκίνησε αμέσως τη διαδικασία εντοπισμού των κοντινών οχημάτων που βρίσκονται σε απόσταση των 200 χλμ. την τελευταία μισή ώρα και δημιουργήθηκε μια λίστα (Εικόνα 28) με τα κοντινά οχήματα οπού εμφανίζονται στον χάρτη (Εικόνα 29).



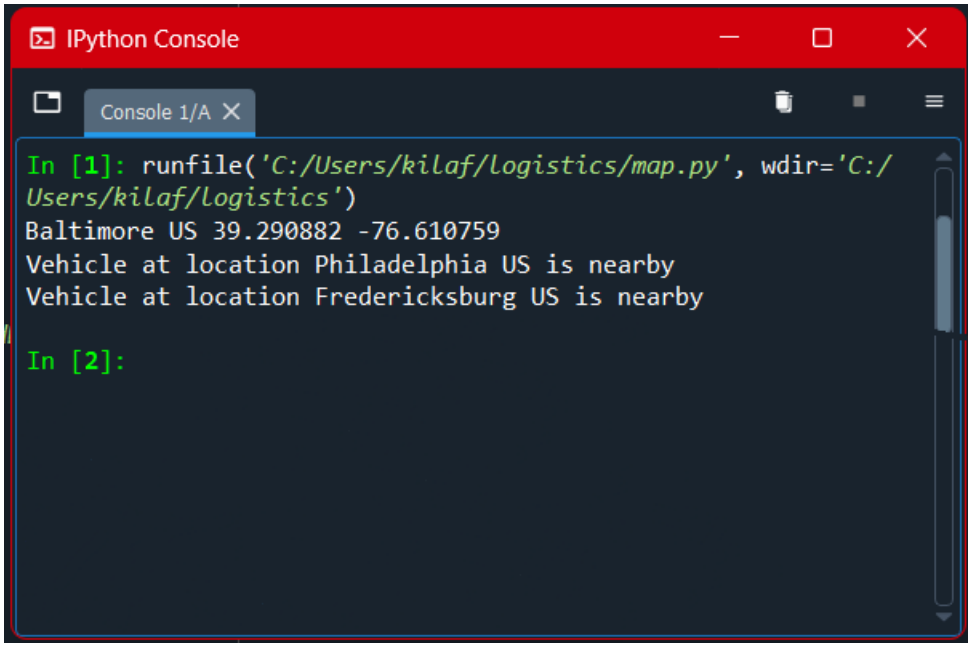
Εικόνα 33. Αποτελέσματα Console3



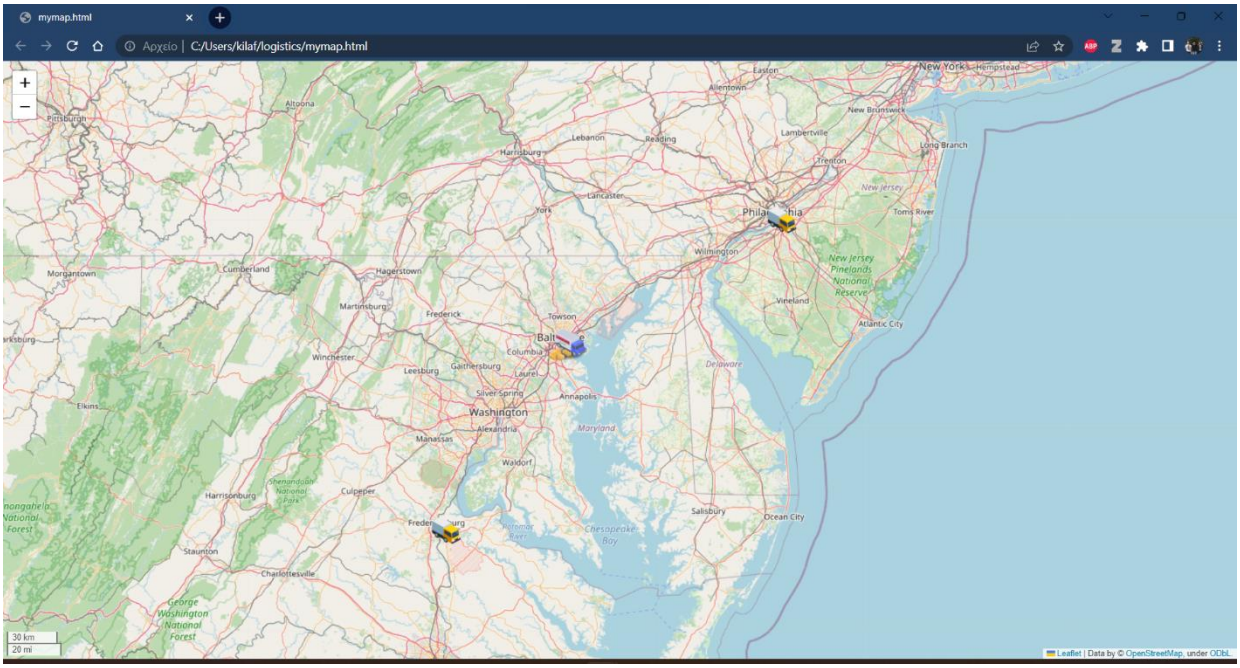
Εικόνα 34. Χάρτης Συμβάντος Map3

Παράδειγμα 4

Στο τέταρτο υποτιθέμενο σενάριο, ένα όχημα με πινακίδα κυκλοφορίας «khj-4893» διαπιστώθηκε πως βρίσκεται για πολύ ώρα στο ίδιο σημείο. Το περιστατικό σημειώθηκε στη Baltimore της Αμερικής. Η τοποθεσία της βλάβης αναφέρθηκε στο κέντρο, δίνοντας το πρόγραμμα τις ακριβείς συντεταγμένες του οχήματος. Το κέντρο ξεκίνησε αμέσως τη διαδικασία εντοπισμού των κοντινών οχημάτων που βρίσκονται σε απόσταση των 200 χλμ. την τελευταία μισή ώρα και δημιουργήθηκε μια λίστα (Εικόνα 30) με τα κοντινά οχήματα οπού εμφανίζονται στον χάρτη (Εικόνα 31).



Εικόνα 35. Αποτελέσματα Console4

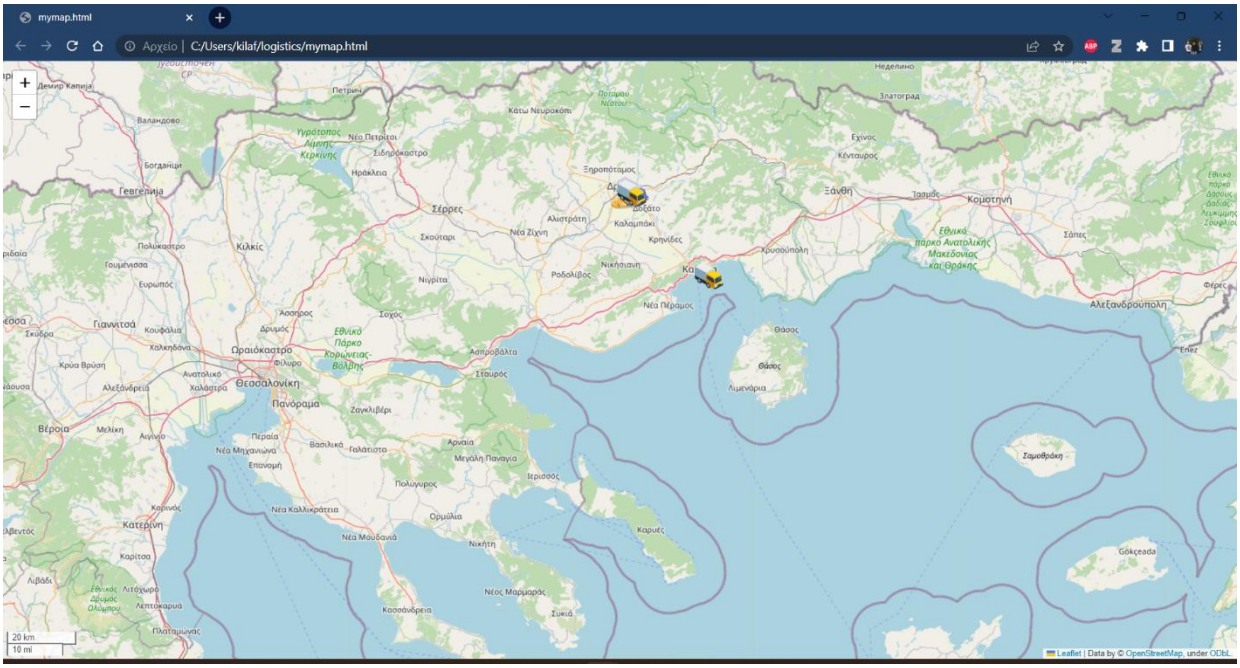


Εικόνα 36. Χάρτης Συμβάντος Map4

Παράδειγμα 5

Στο πέμπτο και τελευταίο υποτιθέμενο σενάριο, ο οδηγός στο όχημα με πινακίδα κυκλοφορίας «cvb-7654» αντιλήφθηκε βλάβη στο όχημα του και μέσω του προγράμματος που του παρέχεται δήλωσε τη βλάβη. Το περιστατικό σημειώθηκε στη Δράμα της. Η τοποθεσία της βλάβης αναφέρθηκε στο κέντρο, δίνοντας το πρόγραμμα τις ακριβείς συντεταγμένες του οχήματος. Το κέντρο ξεκίνησε αμέσως τη διαδικασία εντοπισμού των κοντινών οχημάτων που βρίσκονται σε απόσταση των 200 χλμ. την τελευταία μισή ώρα και δημιουργήθηκε μια λίστα (Εικόνα 32) με τα κοντινά οχήματα οπού εμφανίζονται στο χάρτη (Εικόνα 33).

Εικόνα 37. Αποτελέσματα Console5



Εικόνα 38. Χάρτης Συμβάντος Map5

ΚΕΦΑΛΑΙΟ 6 ΣΥΜΠΕΡΑΣΜΑΤΑ & ΜΕΛΛΟΝΤΙΚΕΣ ΠΡΟΕΚΤΑΣΕΙΣ

6.1 Συμπεράσματα

Σε αυτό το άρθρο, διερευνήθηκαν διάφορα θέματα που σχετίζονται με την επεξεργασία και τη διαχείριση των δεδομένων, συμπεριλαμβανομένων του Kafka, της MySQL, της ουράς και της αναγνώρισης των συμβάντων. Έχουν παρουσιαστεί πολλά αποσπάσματα κώδικα και μελέτες περιπτώσεων για να φανεί η πρακτική χρήση αυτών των τεχνολογιών.

Αρχικά εξετάστηκε η τεχνολογία ουράς, οπού είναι μια θεμελιώδης δομή δεδομένων που παίζει ζωτικό ρόλο σε διάφορες εφαρμογές της επιστήμης των υπολογιστών. Συζητήθηκαν διάφοροι τύποι ουράς, συμπεριλαμβανομένης της ουράς προτεραιότητας και της κυκλικής ουράς, και τις εφαρμογές τους.

Επιπλέον, αναλύθηκε η αναγνώριση των συμβάντων και η σημασία του εντοπισμού των σημαντικών συμβάντων σε ροές δεδομένων. Παρουσιάστηκαν διάφορες προσεγγίσεις για την αναγνώριση των συμβάντων, συμπεριλαμβανομένων των μεθόδων που βασίζονται σε κανόνες και των μεθόδων της μηχανικής μάθησης. Επίσης επισημάνθηκαν οι πιθανές εφαρμογές της αναγνώριση των συμβάντων σε διάφορους τομείς όπως τα οικονομικά, η υγειονομική περίθαλψη και οι μεταφορές οπού και εξερευνήθηκαν.

Μελετώντας το Kafka, γνωστοποιήθηκαν τα οφέλη του στη διαχείριση των μεγάλων ροών δεδομένων. Παρέχει μια επεκτάσιμη, ανεκτική σε σφάλματα πλατφόρμα που μπορεί να επεξεργαστεί εκατομμύρια συμβάντα ανά δευτερόλεπτο. Διευκρινίστηκε πώς μπορεί να χρησιμοποιηθεί το Kafka για την επεξεργασία των δεδομένων και την αναγνώριση των συμβάντων.

Επίσης η MySQL, είναι ένα σύστημα διαχείρισης των σχεσιακών βάσεων δεδομένων που χρησιμοποιείται πολύ συχνά. Εξερευνήθηκαν οι δυνατότητές του στη διαχείριση των δομημένων δεδομένων και τα πλεονεκτήματά της όσον αφορά στην συνέπεια και στην αξιοπιστία των δεδομένων. Έχει διευκρινιστεί πώς να δημιουργείται και γίνεται η διαχείριση των πινάκων στη MySQL, καθώς και πώς να εκτελούνται βασικά ερωτήματα SQL για την ανάκτηση και τον χειρισμό δεδομένων μέσα από τον κώδικα.

Συμπερασματικά, πιστεύουμε ότι τα θέματα που καλύπτονται σε αυτή την εργασία προσφέρουνε μια σταθερή βάση για την επεξεργασία και την διαχείριση των δεδομένων. Οι τεχνολογίες που συζητήθηκαν, όπως το Kafka, η MySQL, η ουρά και η αναγνώριση συμβάντων, προσφέρουν πολλές πρακτικές λύσεις για την διαχείριση των δεδομένων.

6.2 Μελλοντικές Προεκτάσεις

Όσον αφορά τις μελλοντικές επεκτάσεις, το σύστημα έχει τη δυνατότητα προέκτασης και σε περεταίρω δυνατότητες, όπως να αναγνωρίζει το περιεχόμενο του οχήματος με βλάβη. Αυτό θα επέτρεπε στο σύστημα να συντονίζει τα κοντινά οχήματα, στέλνοντας τα να συλλέξουν τα πακέτα που τους ανήκουν με βάση τον διαθέσιμο χώρο και την εξυπηρέτηση του δρομολογίου τους. Με τη προέκταση αυτή θα βελτιώσει σημαντικά τη συνολική απόδοση και την αποτελεσματικότητα της αποστολής τον πακέτων στον χρόνο εκτέλεσης τους.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ανάλυση της διαχείρισης σύνθετων συμβάντων (Complex Event Processing) στη λήψη αποφάσεων και εφαρμογή της σε χαρακτηριστικές μελέτες περίπτωσης—ProQuest. (χ.χ.). Ανακτήθηκε 6 Ιούνιος 2023, από https://www.proquest.com/openview/08b000a55f72b914eea04aad271405bf/1?cbl=2026366&diss=y&parentSessionId=86YbSVpauUIaD3RfHpx%2B3oyod34xAbZfTYGsBf%2BQiJg%3D&pq-origsite=gscholar&parentSessionId=plARb%2B%2FEdyAI1luwQhjyR%2BAbfsLQeX2sEd4iTDF%2FTsU%3D

Κυραμά, Σ. Ε. (2022). Complex Event Processing in a Resource-Constrained Environment.

Apache Flink®—Stateful Computations over Data Streams. (χ.χ.). Ανακτήθηκε 6 Ιούνιος 2023, από https://flink.apache.org/

Brants, T., Chen, F., & Farahat, A. (2003). A System for new event detection. Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, 330–337. https://doi.org/10.1145/860435.860495

Complex Event Processing: Introduction to CEP. (χ.χ.). Confluent. Ανακτήθηκε 6 Ιούνιος 2023, από https://www.confluent.io/learn/complex-event-processing/

Cooper, R. B. (1981). Queueing theory. Proceedings of the ACM '81 conference, 119–122. https://doi.org/10.1145/800175.809851

Esper. (χ.χ.). EsperTech. Ανακτήθηκε 6 Ιούνιος 2023, από https://www.espertech.com/esper/

Event Detector Glossary. (χ.χ.). Analysis Tech. Ανακτήθηκε 2 Μάιος 2023, από https://analysistech.com/event-detectors/event-detector-glossary/

Flouris, I., Giatrakos, N., Deligiannakis, A., Garofalakis, M., Kamp, M., & Mock, M. (2017). Issues in complex event processing: Status and prospects in the Big Data era. Journal of Systems and Software, 127, 217–236. https://doi.org/10.1016/j.jss.2016.06.011

Giambene, G. (2014). Queuing theory and telecommunications (τ. 585). Springer.

IBM Documentation. (2021, Μάρτιος 5). https://www.ibm.com/docs/en/itcam-transactions/7.4.0.1?topic=event-context-information

Jun, C., & Chi, C. (2014). Design of Complex Event-Processing IDS in Internet of Things. 2014 Sixth International Conference on Measuring Technology and Mechatronics Automation, 226–229. https://doi.org/10.1109/ICMTMA.2014.57

Kafka-Apache Kafka.pdf. (χ.χ.). Ανακτήθηκε 2 Μάιος 2023, από http://archive.keyllo.com/L-%E7%BC%96%E7%A8%8B/Kafka-Apache%20Kafka.pdf

Mazon-Olivo, B., Hernández-Rojas, D., Maza-Salinas, J., & Pan, A. (2018). Rules engine and complex event processor in the context of internet of things for precision agriculture. Computers and Electronics in Agriculture, 154, 347–360. https://doi.org/10.1016/j.compag.2018.09.013

Morton, A., Liu, J., & Song, I. (2007). Efficient Priority-Queue Data Structure for Hardware Implementation. 2007 International Conference on Field Programmable Logic and Applications, 476–479. https://doi.org/10.1109/FPL.2007.4380693

Mredula, M. S., Dey, N., Rahman, M. S., Mahmud, I., & Cho, Y.-Z. (2022). A Review on the Trends in Event Detection by Analyzing Social Media Platforms' Data. Sensors, 22(12), Article 12. https://doi.org/10.3390/s22124531

Queue (abstract data type). (2023). Στο Wikipedia. https://en.wikipedia.org/w/index.php?title=Queue_(abstract_data_type)&oldid=1140036240

Queue: Definition, Types, Implementation, Usage, and More. (2022, Σεπτέμβριος 24). DevOps and Software Engineering Glossary Terms | Atatus. https://www.atatus.com/glossary/queue/

Rahmani, A. M., Babaei, Z., & Souri, A. (2021). Event-driven IoT architecture for data analysis of reliable healthcare application using complex event processing. Cluster Computing, 24(2), 1347–1360. https://doi.org/10.1007/s10586-020-03189-w

Wang, G., Koshy, J., Subramanian, S., Paramasivam, K., Zadeh, M., Narkhede, N., Rao, J., Kreps, J., & Stein, J. (2015). Building a replicated logging system with Apache Kafka. Proceedings of the VLDB Endowment, 8(12), 1654–1655. https://doi.org/10.14778/2824032.2824063