Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Πλατφόρμα συλλογής, αποθήκευσης και επεξεργασίας δεδομένων σε μορφή χρονοσειρών από συσκευές IoT

Διπλωματικη Εργασια

του

ΘΕΟΔΩΡΙΔΗ Σ. ΑΝΑΣΤΑΣΙΟΥ

Επιβλέπων: Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2022

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Πλατφόρμα συλλογής, αποθήκευσης και επεξεργασίας δεδομένων σε μορφή χρονοσειρών από συσκευές IoT

Διπλωματικη Εργασια

του

ΘΕΟΔΩΡΙΔΗ Σ. ΑΝΑΣΤΑΣΙΟΥ

Επιβλέπων: Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.

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

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Παναγιώτης Τσανάκας	Δημήτριος Σουντρής	Συμεών Παπαβασιλείου
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2022

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

(Υπογραφή)

.....
Αναστάσιος Θεοδωρίδης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π

Περίληψη

Οι συσκευές IoT αποτελούν σήμερα σημαντικό κεφάλαιο σε αναρίθμητες εφαρμογές, είτε οικιακές/ατομικές είτε βιομηχανικές. Προσφέρουν αξιοπιστία στην διακίνηση των δεδομένων που παράγουν καθώς και οι τεχνολογίες που έχουν στόχο για την υποστήριξη των δικτύων με τέτοιες συσκευές βρίσκονται σε συνεχή ανάπτυξη. Ο σκοπός της διπλωματικής εργασίας είναι η ανάπτυξη πλατφόρμας που θα διαχειρίζεται τα δεδομένα χρονοσειρών που παράγονται από τις συσκευές εντός δικτύων IoT καθώς και η προετοιμασία του συστήματος που θα διαχειρίζεται την πλατφόρμα.Η πλατφόρμα θα παρέχει την δυνατότητα αποθήκευσης των μηνυμάτων από τις συσκευές σε βάση δεδομένων ακολουθώντας συσχετίσεις που δίνονται από τον χρήστη καθώς και την προώθηση των δεδομένων σε προσαρμοσμένα πακέτα σε επιπρόσθετο σύστημα μετάδοσης ροών δεδομένων για περαιτέρω επεξεργασία.

Λέξεις Κλειδιά

IoT, Pub-Sub brokers, MQTT, apache kafka, Mosquitto, TimescaleDB, python

Abstract

IoT devices are today an important asset in countless applications, either residential/personal or industrial. They offer reliability in the transport of the data they produde and the technologies aimed at supporting networks with such devices are in constant development. The purpose of this thesis is the development of a platform that will manage the time-series data generated by the devices within IoT networks as well as the preparation of the system that will handle the platform. The platform will provide the ability to store the messages from the devices in a database following relations given by the user as well as forwarding the data in custom packets to additional data streaming system for further processing.

Keywords

IoT, Pub-Sub brokers, MQTT, apache kafka, Mosquitto, TimescaleDB, python

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή κ. Παναγιώτη Τσανάκα για την επίβλεψη αυτής της διπλωματικής εργασίας και την ερευνητική ομάδα I-SENSE για την ευκαιρία που μου έδωσαν να διευρύνω τις γνώσεις μου μέσω της εκπόνησής της. Επίσης θα ήθελα να ευχαριστήσω ιδιαίτερα τον κ. Άρη Δαδούκη για την συνεχή καθοδήγησή, την υποστήριξη και την εξαιρετική συνεργασία που είχαμε κατά τη συγγραφή της διπλωματικής μου εργασίας.

Με αφορμή την ολοκλήρωση των σπουδών μου θα ήθελα να ευχαριστήσω τους γονείς μου Μαρία και Στέλιο και την αδερφή μου Ναταλία για την υπομονή και υποστήριξη τους όλα αυτά τα χρόνια. Για τη συνεχή στήριξη και συμπαράσταση ευχαριστώ, τέλος, θερμά την Νεκταρία Δουρουδή.

Αθήνα, Σεπτέμβριος 2022

Αναστάσιος Θεοδωρίδης

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Εικόνων

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

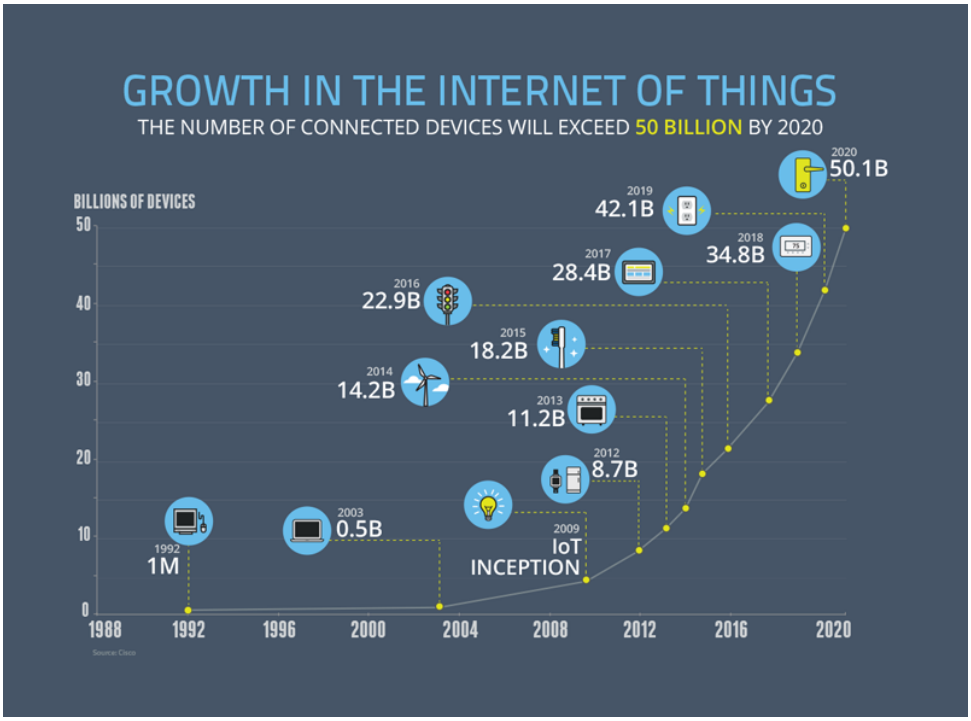
IoT	Internet of Things
MQTT	MQ Telemetry Transport
TCP	Transmission Control Protocol
QOS	Quality Of Service
SSL	Secure Socket Layer
ack	acknowledgement
SASL	Simple Authentication and Security Layer
SQL	Structured Query Language
TLS	Transport Layer Security
JSON	JavaScript Object Notation
API	Application Programming Interface

Κεφάλαιο 1

Εισαγωγή

1.1 Περιγραφή προβλήματος

Τα τελευταία χρόνια παρατηρείται ταχεία ανάπτυξη στον τομέα του Internet of Things με τα δεδομένα που διακινούνται στα IoT δίκτυα να συσσωρεύονται σε πολύ μεγάλα μεγέθη. Η παραγωγή συσκευών IoT ολοένα και πληθαίνει καθώς η κατεύθυνση εφαρμογής τους από πιο ειδικευόμενους κλάδους προσεγγίζει πλέον και πιο απλές οικιακές χρήσεις.



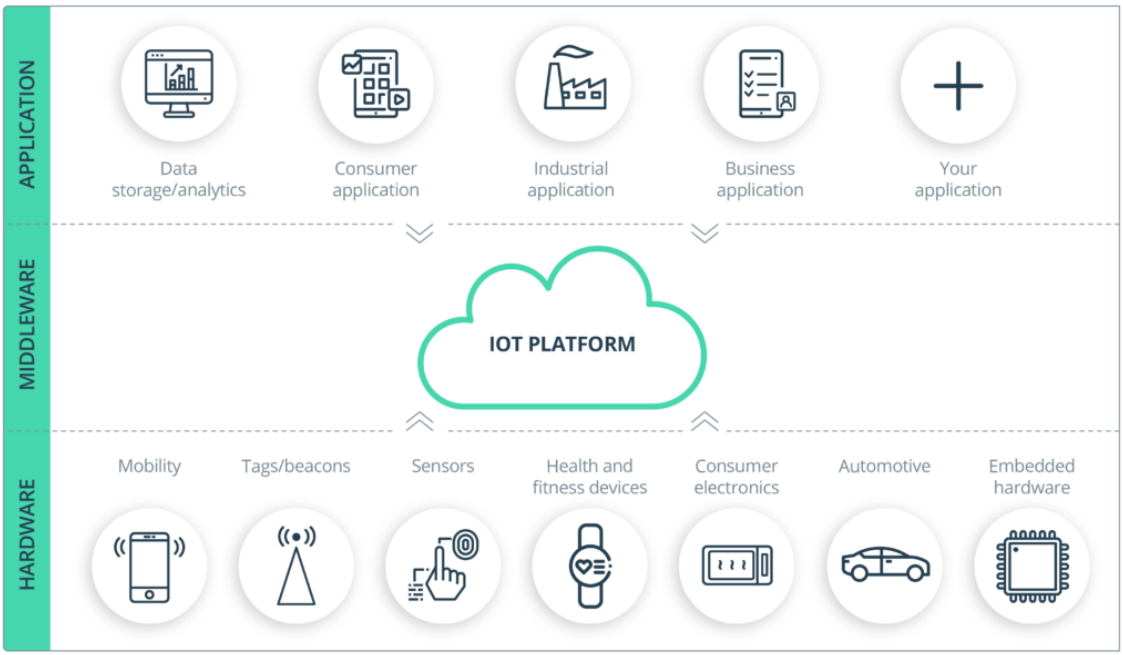
Πηγή: [1]
Εικόνα 1.1: *Ανάπτυξη πλήθους συσκευών IoT*

Για την βελτιστοποίηση των λειτουργιών σε ένα δίκτυο IoT είναι αναγκαία η σωστή διαχείριση της δρομολόγησης των δεδομένων. Παίζουν μεγάλο ρόλο οι συνθήκες και το περιβάλλον στο οποίο βρίσκεται το δίκτυο στις αποφάσεις βαθμονόμησης του

εκάστοτε δικτύου. Σε ένα οικιακό δίκτυο το πλήθος συσκευών όπως και το φόρτο δεδομένων είναι αρκετά μικρό, με ελάχιστη συχνότητα πακέτων έτσι και οι απαιτήσεις σε ταχύτητα και αξιοπιστία είναι πολύ λίγες. Αντιθέτως σε κάποιο εργαστηριακό ή βιομηχανικό περιβάλλον με μεγάλο πλήθος συσκευών και δεδομένων οι ανάγκες αυξάνονται κρίνοντας αναγκαία την ρύθμιση του δικτύου ώστε να μπορεί να διαχειριστεί το φόρτο συνδέσεων και δεδομένων με την μικρότερη δυνατή απώλεια δεδομένων.

Η αποθήκευση των δεδομένων είναι επίσης ένα πολύ σημαντικό κομμάτι στα δίκτυα IoT καθώς δίνει την δυνατότητα της πρόσβασης και ανάλυσης αυτών σε οποιαδήποτε χρονική στιγμή. Στην αποθήκευση, όπως αναφέρθηκε και για την δρομολόγηση, διαφέρουν οι σωστές πρακτικές ανάλογα το περιβάλλον και τον σκοπό του δικτύου. Ένα μεγάλο μέρος συσκευών IoT, χρησιμοποιεί υλικό όπως αισθητήρες για την καταγραφή δεδομένων από το περιβάλλον ανά τακτά χρονικά διαστήματα σε μορφή χρονοσειρών. Έχουν αναπτυχθεί πολλές λύσεις ανοιχτού λογισμικού για την σωστή δρομολόγηση επεξεργασία και αποθήκευση αυτών των δεδομένων όπως και για την σωστή διακίνηση των δεδομένων από τις συσκευές αυτές.

Ένα πρόβλημα που μπορεί να προκύψει σε ένα δίκτυο IoT με πολλές συσκευές είναι η αυτοματοποίηση της δρομολόγησης των δεδομένων σε δομές αποθήκευσης ανάλογα κάποιου χαρακτηριστικού τους και η προώθηση τους σε επιπλέον συστήματα διακίνησης μηνυμάτων. Για την επίλυση αυτού του προβλήματος αναπτύσσονται πλατφόρμες που ασχολούνται με την ενοποίηση των επί μέρους κόμβων ενός δικτύου IoT όπως τις συσκευές με κάποιον εξυπηρετητή, δομές αποθήκευσης και επιπλέον συστήματα επεξεργασίας δεδομένων.



Πηγή: [2]
Εικόνα 1.2: *Πλατφόρμα IoT*

1.2 Προσέγγιση προβλήματος

Για την λήψη δεδομένων από τις συσκευές επιλέχτηκε το πρωτόκολλο MQTT που χρησιμοποιείται ευρέως για την επικοινωνία συσκευών σε δίκτυα IoT. Για την αποθήκευση των δεδομένων επιλέχτηκε η βάση δεδομένων TimescaleDB που είναι ειδικευόμενη σε δεδομένα χρονοσειρών. Για την περαιτέρω αποστολή των δεδομένων σε επιπλέον δομή επιλέχτηκε το Apache Kafka που επιτρέπει την αποστολή και λήψη μεγάλου μεγέθους δεδομένων όπως και την προσωρινή τους αποθήκευση. Για την ανάπτυξη της πλατφόρμας χρησιμοποιήθηκε η προγραμματιστική γλώσσα python [3].

1.3 Στόχος

Ο σκοπός αυτής της διπλωματικής εργασίας είναι η ανάπτυξη πλατφόρμας που θα λαμβάνει δεδομένα χρονοσειρών από συσκευές IoT, θα τα αποθηκεύει σε βάση δεδομένων, θα τα προωθεί σε επιπλέον δομή και θα δίνει την δυνατότητα περαιτέρω επεξεργασίας τους. Πραγματοποιήθηκαν οι παρακάτω ενέργειες:

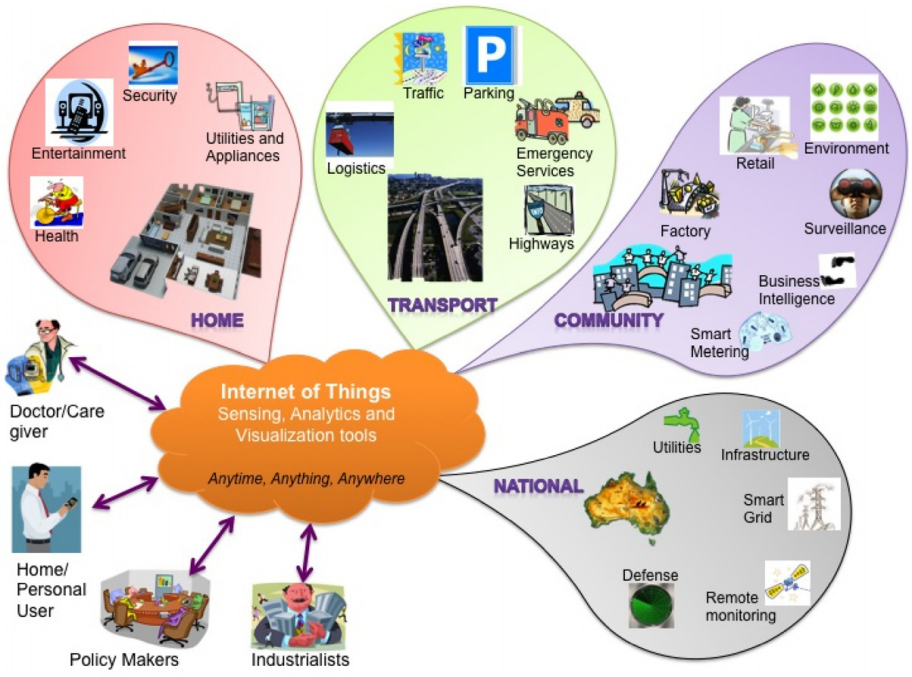
- Προετοιμασία εξυπηρετητή με την εγκατάσταση
  MQTT broker, Kafka broker και TimescaleDB.
- Ανάπτυξη της πλατφόρμας.
- Προσθήκη επιπέδου επεξεργασίας των δεδομένων
- Ανάλυση επιδόσεων της πλατφόρμας

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

2.1 Internet of Things

Internet of Things ή αλλιώς το διαδίκτυο των πραγμάτων είναι ο όρος που χρησιμοποιείται για να περιγράψει δίκτυα συσκευών που ανταλλάσσουν πληροφορίες χωρίς την ανθρώπινη παρέμβαση. Έχει γνωρίσει μεγάλη ανάπτυξη τα τελευταία χρόνια και με τη βοήθεια της συνεχόμενης εξέλιξης της βιομηχανίας του διαδικτύου έχει εξελιχθεί σε σύνολο αντικειμένων που εκτός από την συγκέντρωσή δεδομένων από το περιβάλλον και την αλληλεπίδραση με τον φυσικό κόσμο, χρησιμοποιεί υπάρχοντα διαδικτυακά πρότυπα για να παρέχει υπηρεσίες για μεταφορά δεδομένων, αναλυτικά στοιχεία, εφαρμογές και επικοινωνία. Στην εικόνα 2.1 παρουσιάζονται τα πεδία εφαρμογών δικτύων IoT και οι τελικοί χρήστες. [4]



Πηγή: Gubbi 2013[4]

Εικόνα 2.1: *Τελικοί χρήστες IoT και τα πεδία εφαρμογών του*

Στό [5] αναφέρονται 4 επίπεδα αρχιτεκτονικής για το IoT:

- Το αισθητήριο επίπεδο (sensing layer): Είναι το επίπεδο το οποίο είναι ενσωματωμένο με το υλικό όπως π.χ. διάφοροι αισθητήρες και αντιδράει με τον φυσικό κόσμο για να λάβει δεδομένα.

- Το δικτυακό επίπεδο (networking later): Είναι το επίπεδο που προσφέρει δικτύωση και τη μεταφορά δεδομένων μεταξύ δικτύου.

- Το επίπεδο εφαρμογής (service layer): Είναι το επίπεδο που δημιουργεί και διαχειρίζεται τις εφαρμογές για τις ανάγκες των χρηστών.

- Το επίπεδο διεπαφής (interface layer): Είναι το επίπεδο που παρέχει μεθόδους αλληλεπίδρασης με τους χρήστες και τις υπόλοιπες εφαρμογές

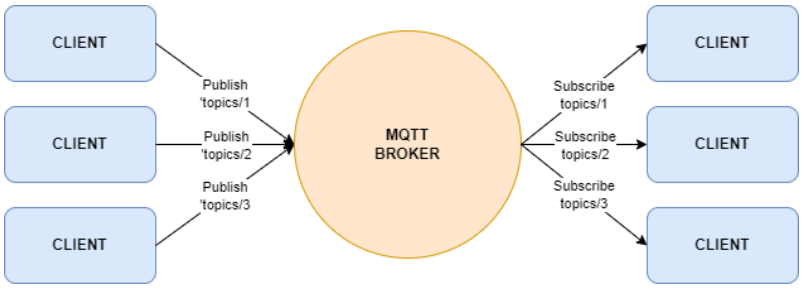
2.2 Τεχνολογίες pub/sub

Το pub/sub (publish/subscribe: δημοσίευση/εγγραφή) είναι πρωτόκολλο επικοινωνίας μεταξύ 2 οντοτήτων που ονομάζονται publisher (εκδότης) και subscriber (συνδρομητής). Πιο συγκεκριμένα, η αλληλεπίδραση publish/subscribe επιτρέπει στους subscribers να ανακοινώσουν το ενδιαφέρον τους σε κάποια γεγονότα με κάποιο κοινό χαρακτηριστικό και να ενημερωθούν κατά την παραγωγή τέτοιον γεγονότων από οποιονδήποτε publisher. Η πιο βασική κατηγορία είναι το Topic based publish/subscribe (TBPS), στο οποίο η έκδοση μηνυμάτων γίνεται σε ένα topic (θέμα) και οποιοσδήποτε subscriber μπορεί να εγγραφεί και να λαμβάνει μηνύματα σε αυτό. [6]

Κάποιες από τις πιο γνωστές εφαρμογές pub/sub messaging είναι το Apache Kafka [7], Redis, Google Pub/Sub, MQTT Protocol. Στα πλαίσια της διπλωματικής θα εξετάσουμε τα MQTT [8] και Kafka που επιλέχτηκαν για την επικοινωνία με τις συσκευές και την προώθηση των δεδομένων προς άλλες εφαρμογές.

2.2.1 MQTT Protocol

Το MQTT (Message Queuing Telemetry Transport) είναι ένα πρωτόκολλο μεταφοράς μηνυμάτων μεταξύ συσκευών σχεδιασμένο να είναι αποδοτικό ως προς τη χωρητικότητα δικτύου, την κατανάλωση ενέργειας και μνήμης συσκευής. Υλοποιείται κυρίως πάνω στο Πρωτόκολλο Ελέγχου Μεταφοράς (TCP). Το πρωτόκολλο ορίζει 2 τύπους οντοτήτων στο δίκτυο: έναν broker μηνυμάτων και τους clients. Ο broker είναι ο εξυπηρετητής (server) που δέχεται όλα τα μηνύματα από τους clients και τα δρομολογεί στους σχετικούς clients. Η βασική αρχιτεκτονική ενός δικτύου MQTT και οι σχέσεις παρουσιάζονται στο σχήμα 2.1

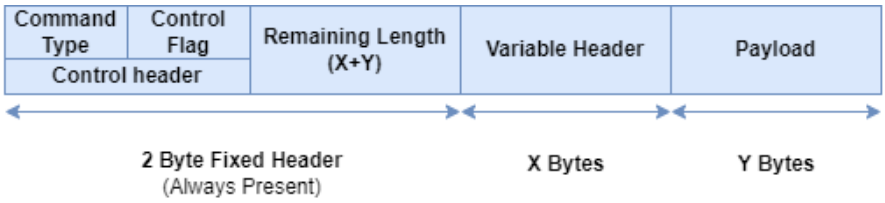


Σχήμα 2.1: *Αρχιτεκτονική δικτύου MQTT*

Γίνεται χρήση του προτύπου publish/subscribe (δημοσίευση/εγγραφή) για τον διαμοιρασμό μηνυμάτων από έναν κόμβο σε πολλές συσκευές. Τα μηνύματα οργανώνονται σε topics (θέματα) κατά τη δημοσίευσή τους και μπορεί να τα λάβει όποιος client -αν υπάρχει- έχει κάνει εγγραφή στο σχετικό topic. Αυτό προσφέρει μεγάλη επεκτασιμότητα στο δίκτυο, καθώς δεν υπάρχει αλληλεξάρτησή μεταξύ των αποστολέων και των παραληπτών. Ο τίτλος του topic μπορεί να χωριστεί σε επίπεδα με τον χαρακτήρα ¨/¨ για καλύτερη διαχείριση και κατανόηση όπως για παράδειγμα ¨/home/sensors/temperature/office¨ που μπορεί να είναι ένα topic για τη διακίνηση μηνυμάτων από τους αισθητήρες θερμοκρασίας του γραφείου σε ένα σπίτι.

Υπάρχουν ειδικοί χαρακτήρες (wildcard characters) που δεν επιτρέπονται στην ονομασία ενός topic, αλλά μπορούν να χρησιμοποιηθούν από τους clients για την εγγραφή σε ένα ή παραπάνω topic. Αυτοί είναι το ¨**#**¨ για πολυεπίπεδη εγγραφή και το ¨**+**¨ για εγγραφή ενός επιπέδου. Για την πολυεπίπεδη εγγραφή ο χαρακτήρας ¨**#**¨ εξισώνεται με πολλά επίπεδα και αντιπροσωπεύει τον γονέα και όλα τα παιδιά π.χ. η εγγραφή στο /home/sensors/# θα έχει ως αποτέλεσμα την εγγραφή σε όλα τα topics που ξεκινάνε με /home/sensors/, όπως [/home/sensors/temperature/bedroom, /home/sensors/humidity/livingroom] κτλ. Στην εγγραφή μονού επιπέδου ο χαρακτήρας ¨**+**¨ εξισώνεται μόνο με ένα επίπεδο π.χ. η εγγραφή στο [/home/sensors/+/office] θα έχει ως αποτέλεσμα την εγγραφή σε όλα τα topics 4 επιπέδων που αρχίζουν με [/home/sensors/ και τελειώνουν με /office], όπως [/home/sensors/temperature/office, /home/sensors/humidity/office] κτλ. Ο συνδυασμός αυτών των χαρακτήρων δίνει τη δυνατότητα στους clients να φιλτράρουν δεδομένα από μεγάλο σύνολο topics, καθιστώντας πιο εύκολη την επεξεργασία δεδομένων από πολλές πηγές.

Το πρωτόκολλο ορίζει 15 προκαθορισμένους τύπους μηνυμάτων στον MQTT που χρησιμοποιούνται για την επικοινωνία μεταξύ client και broker για τις ανάγκες της σύνδεσης, εγγραφής, απ-εγγραφής, δημοσίευσης, ελέγχου (ping), αποσύνδεσης και ταυτοποίησης. Ένα πακέτο στον MQTT έχει μέγεθος από 2 Byte μέχρι και 256 MegaBytes και η δομή ενός πακέτου αποτελείται από 3 μέρη στην εξής σειρά: το Fixed Header (σταθερή κεφαλίδα) που βρίσκεται σε όλα τα πακέτα, το Variable Header (μεταβλητή κεφαλίδα) που βρίσκεται σε μερικά πακέτα και το payload (φορ-

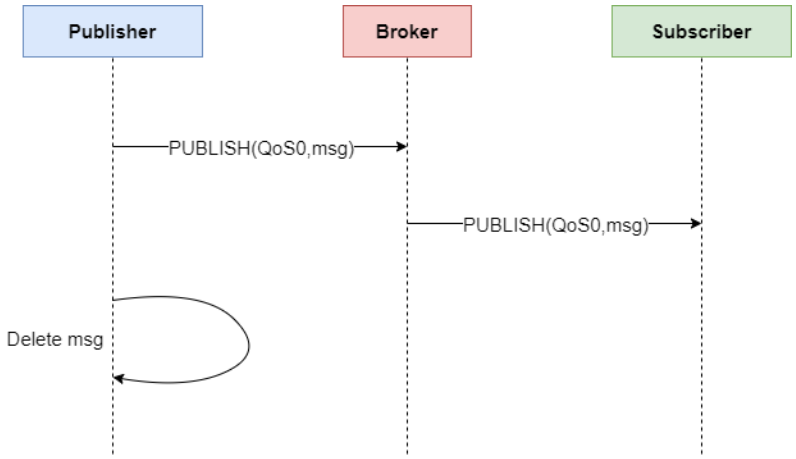


Σχήμα 2.2: *Δομή πακέτου MQTT*

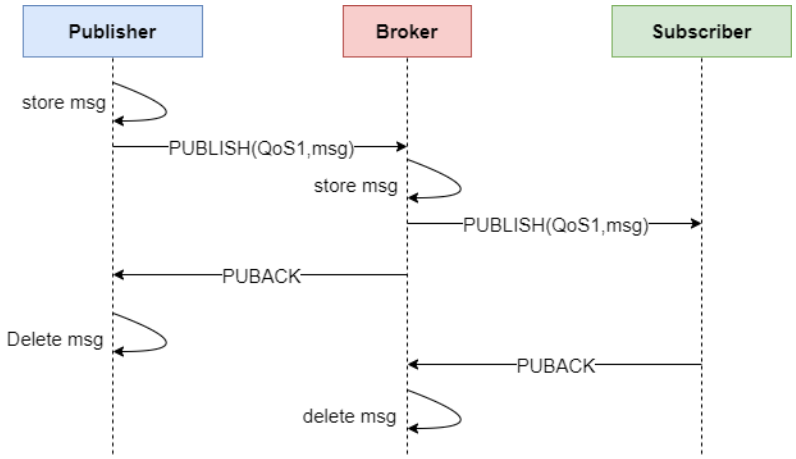
τίο), που επίσης βρίσκεται σε μερικά πακέτα. Το Fixed Header περιγράφει τον τύπο του μηνύματος και περιέχει πληροφορίες ειδικές για κάθε τύπο. Κάποιοι τύποι πακέτων προϋποθέτουν και την χρήση των Variable Header και payload. όπως π.χ. το πακέτο με τύπο PUBLISH για δημοσίευση πληροφορίας προϋποθέτει την ύπαρξη του Variable Header με πληροφορίες όπως το topic δημοσίευσης, αλλά και του payload με το μήνυμα που δημοσιεύεται. Στο σχήμα 2.2 παρουσιάζεται η δομή του πακέτου MQTT και στην εικόνα 2.2 οι διάφοροι τύποι πακέτων με την μορφή τους.

Message type	Value	Description	Fixed Header	Variable Header	Payload
Reserved	0	Reserved	Present	-	-
CONNECT	1	Client connect request to server or broker	Present	Present	Required
CONNACK	2	Connect request acknowledgment	Present	Present	None
PUBLISH	3	Publish message	Present	Present	Required
PUBACK	4	Publish acknowledgment	Present	Present	None
PUBREC	5	Publish receive	Present	Present	None
PUBREL	6	Publish release	Present	Present	None
PUBCOMP	7	Publish complete	Present	Present	None
SUBSCRIBE	8	Client subscribe request	Present	Present	Required
SUBACK	9	Subscribe request acknowledgment	Present	Present	Required
UNSUBSCRIBE	10	Unsubscribe request	Present	Present	Required
UNSUBACK	11	Unsubscribe acknowledgment	Present	Present	None
PINGREQ	12	PING request	Present	None	None
PINGRESP	13	PING response	Present	None	None
DISCONNECT	14	Client is disconnecting	Present	None	None
Reserved	15	Reserved	Present	-	-

Εικόνα 2.2: *Τύποι μηνυμάτων MQTT*



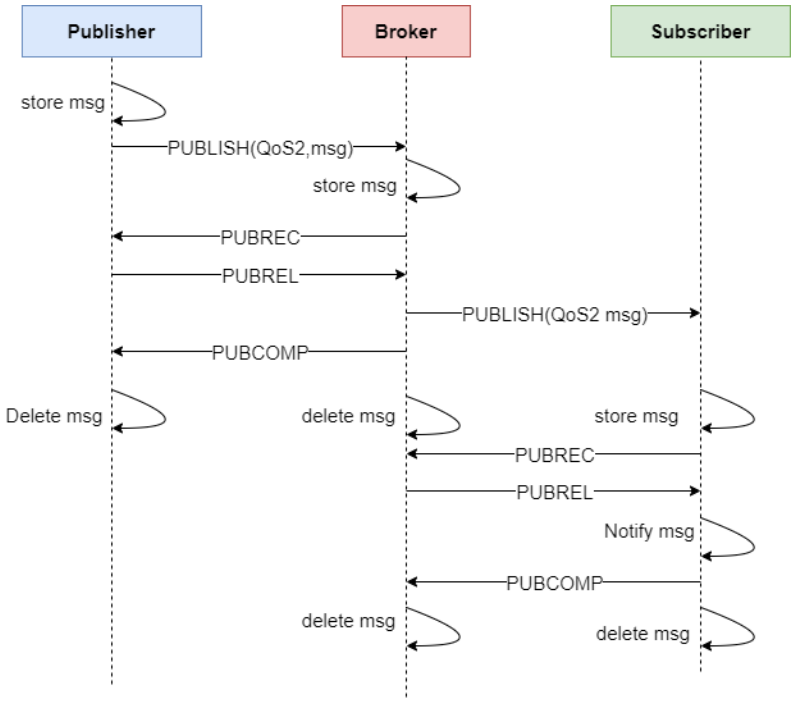
Σχήμα 2.3: *MQTT qos0*



Σχήμα 2.4: *MQTT qos1*

Το MQTT υποστηρίζει την λειτουργία QoS (Quality of Service - Ποιότητα εξυπηρέτησης) που είναι μια συμφωνία μεταξύ client και broker για τη διασφάλιση της παράδοσης των μηνυμάτων. Υπάρχουν 3 επίπεδα QoS:

- **QoS 0 - Το πολύ μία φορά**: είναι το πιο απλό επίπεδο εξυπηρέτησης στο οποίο ο client απλά παραδίδει το μήνυμα και ο broker δεν στέλνει μήνυμα επιβεβαίωσης λήψης του μηνύματος. Δεν υπάρχει εγγύηση επιτυχούς μετάδοσης του μηνύματος.
- **QoS 1 - Τουλάχιστον μία φορά**: Ο broker στέλνει μήνυμα επιβεβαίωσης λήψης του μηνύματος στον client αλλά αν χαθεί, ο client θα στείλει ξανά το μήνυμα ωσότου πάρει το μήνυμα επιβεβαίωσης από τον broker. Η παράδοση του μηνύματος είναι εγγυημένη, αλλά το μήνυμα μπορεί να σταλθεί πάνω από μία φορά.
- **QoS 2 - Ακριβώς μία φορά**: Είναι το πιο υψηλό επίπεδο εξυπηρέτησης, όπου ανταλλάσσονται 4 μηνύματα μεταξύ client και broker,δηλαδή μια χειραψία 4 μερών, στην οποία έπειτα από την λήψη μηνύματος επιβεβαίωσης από τον broker, ο client αποστέλλει μήνυμα για να απαλλάξει τον broker από την

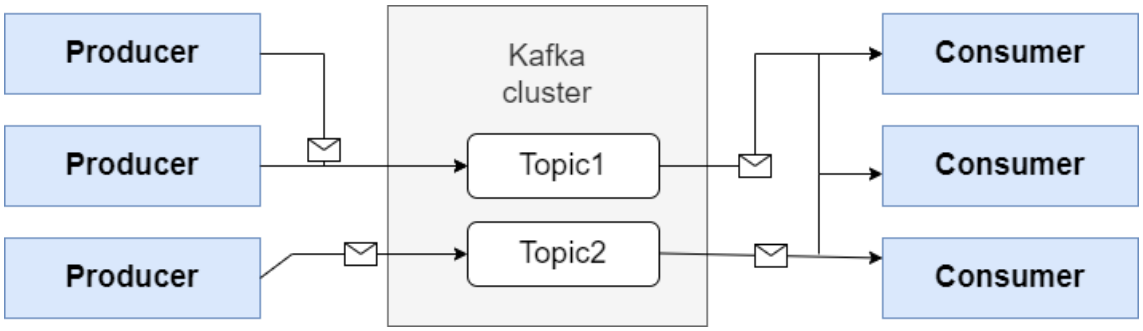


Σχήμα 2.5: *MQTT qos2*

επικείμενη αποστολή του ίδιου μηνύματος, στο οποίο ο client απαντάει με μήνυμα ολοκλήρωσης της μετάδοσης.

Όσο ανεβαίνουμε επίπεδα στο QOS τόσο μεγαλύτερο το κόστος της κάθε συναλλαγής ανά μήνυμα λόγω των πλήθος τον ξεχωριστών μηνυμάτων ελέγχου που ανταλλάσσονται για την επαλήθευσή του. Η επιλογή επιπέδου QOS καθορίζεται από την εφαρμογή και τις απαιτήσεις της σε ρυθμό διεκπεραίωσης μηνυμάτων και απώλεια δεδομένων. Στα σχήματα 2.3, 2.4, 2.5 παρουσιάζονται οι αλληλεπιδράσεις μεταξύ publisher-Broker-Subscriber για κάθε επίπεδο qos.

Το MQTT δεν υποστηρίζει αποθήκευση παρελθοντικών μηνυμάτων, ωστόσο, διαθέτει όμως λειτουργία για την διατήρηση του τελευταίου μηνύματος για κάθε topic που το ζητάει. Αυτό δίνει την δυνατότητα στους παραλήπτες να έχουν επίγνωση του τελευταίου μηνύματος μόλις εγγράφουν. Η δυνατότητα αποθήκευσης ή αλλιώς retained message είναι ιδιότητα του πακέτου μηνύματος και ορίζεται κατά την αποστολή. Επίσης στην περίπτωση που ένας client αποσυνδεθεί κατά τη διάρκεια λήψης μηνυμάτων από κάποιο topic δίνεται δυνατότητα από τον MQTT, αν έχει οριστεί persistent session (επίμονη συνεδρία) κατά την σύνδεση του client, στην επανασύνδεση του να λάβει όλα τα μηνύματα που δεν παρέλαβε όσο ήταν εκτός σύνδεσης καθώς τα αποθηκεύει ο broker. Αντιθέτως όταν ένας client κάνει εγγραφή με non persistent connection (clean session), ο broker δεν αποθηκεύει καμία πληροφορία για την εγγραφή του η τα μη παραδιδόμενα μηνύματα προς αυτόν. Για τους publisher η ιδανική σύνδεση είναι non persistent connection ενώ για τους subscribers η επιλογή της σύνδεσης είναι θέμα εφαρμογής και αναγκών.



Σχήμα 2.6: *Αρχιτεκτονική συστήματος kafka*

Για την ενημέρωση των subscribers στην περίπτωση αφύσικης αποσύνδεσης ενός publisher, υπάρχει ένας τύπος μηνύματος που στέλνει ο broker στα σχετικά topics που ονομάζεται last will and testament. Το περιεχόμενο του μηνύματος αυτού ορίζεται από τον publisher και όπως όλα τα μηνύματα περιέχει flags για το QOS και το retain.

Το MQTT υποστηρίζει αυθεντικοποίηση με κωδικό και όνομα χρήστη για την ασφάλεια των δεδομένων καθώς όμως τα μηνύματα αποστέλλονται ως απλό κείμενο (PLAINTEXT), για τη μέγιστη ασφάλεια των δεδομένων παρέχεται δυνατότητα κρυπτογράφησης της σύνδεσης με πιστοποιητικά SSL.

2.2.2 Kafka

Το apache Kafka είναι μια πλατφόρμα για τη διαχείριση ροών δεδομένων σε πραγματικό χρόνο. Παρέχει υψηλό ρυθμό διεκπεραίωσης μηνυμάτων και χαμηλή καθυστέρηση δικτύου καθιστώντας το ιδανικό για ροές δεδομένων μεγάλου μεγέθους. Χρησιμοποιεί δυαδικό πρωτόκολλο μέσω TCP.

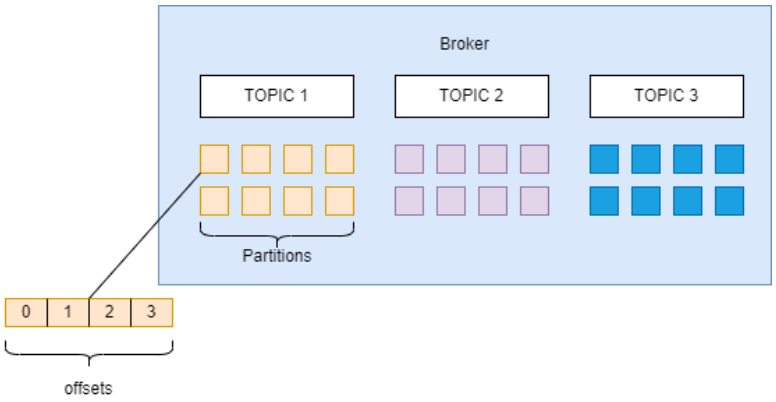
Ένα σύστημα kafka αποτελείται από έναν ή περισσότερους εξυπηρετητές (server), οι οποίοι αποκαλούνται brokers και τους πελάτες (clients). Ο broker λαμβάνει και αποθηκεύει όλες τις ροές προς αυτόν, ενώ ο client δημοσιεύει (publish) ή εγγράφεται (subscribe) σε ροές δεδομένων όπως παρουσιάζεται στο σχήμα 2.6. Όταν ένας client παράγει δεδομένα αποκαλείται producer (παραγωγός), ενώ όταν τα καταναλώνει ονομάζεται consumer (καταναλωτής). Ένα event (γεγονός) ή αλλιώς record (εγγραφή) ή μήνυμα περιέχει ένα κλειδί, μια χρονοσήμανση και προαιρετικές κεφαλίδες μεταδεδομένων (metadata headers). Οι ροές οργανώνονται και αποθηκεύονται σε topics (θέματα), στον αποθηκευτικό χώρο σε μορφή αρχείων, από τα οποία τα δεδομένα μπορούν να προβληθούν απεριόριστα, για χρονικό διάστημα που καθορίζεται στις ρυθμίσεις δημιουργίας ενός topic.

Ο kafka στηρίζεται στην ομαδοποίηση των μηνυμάτων για την καλύτερη απόδοση. Για τη λειτουργία αυτή ο παραγωγός προσπαθεί να συσσωρεύει τα δεδομένα στη μνήμη για να αποστέλλει μεγαλύτερες παρτίδες σε μια συναλλαγή. Η ομαδοποίηση αυτή μπορεί να ρυθμιστεί ώστε να μην συσσωρεύει πάνω από ένα προκαθορισμένο νούμερο μηνυμάτων και να μη διαρκεί πάνω από ένα προκαθορισμένο χρονικό διάστημα. Αυτό επιτρέπει στη συσσώρευση πακέτων μεγάλου μεγέθους για αποστολή μέσω δικτύου μειώνοντας το πλήθος τους και την μείωση ξεχωριστών εγγραφών στον αποθηκευτικό χώρο του server. Οι μεταβολές στο χρονικό διάστημα συσσώρευσής ή στο μέγεθος δημιουργίας παρτίδας μεταβάλλουν τα μεγέθη του ρυθμού διεκπεραίωσης και της καθυστέρησης των μηνυμάτων, καθώς υπάρχει αντιστρόφως ανάλογη σχέση μεταξύ τους. Λόγω αυτού κρίνονται αναγκαίες διαφορετικές ρυθμίσεις για την βελτιστοποίηση σε διαφορετικές ροές δεδομένων. Η ομαδοποίηση των μηνυμάτων δίνει τη δυνατότητα σε ροές με πολλαπλές τυχαίες ¨εκρήξεις¨ δεδομένων να μετατραπούν σε μια σταθερή γραμμική ροή προς τους καταναλωτές. Οι παρτίδες μπορούν να συμπιεστούν κατά την παραγωγή και την αποθήκευση τους και να αποσυμπιεστούν μόνο στο στάδιο της κατανάλωσης.

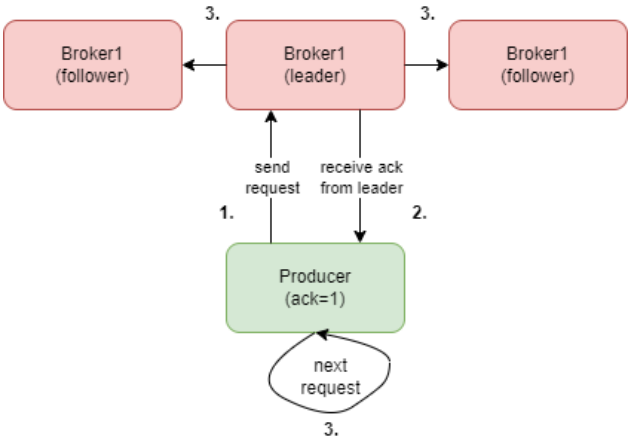
Το kafka υποστηρίζει κατανεμημένη αρχιτεκτονική, παρέχοντας τη δυνατότητα ύπαρξης πολλών server για λόγους επιδόσεων, άλλα και για την περίπτωση σφάλματος σε κάποιον. Οι kafka brokers είναι σχεδιασμένοι να λειτουργούν ως μία ομάδα από πολλούς brokers που αποκαλείται cluster, στο οποίο ένας broker εκλέγεται ως ελεγκτής (lead broker) του cluster και είναι υπεύθυνος για τη διαχείριση του.

Τα topics μπορούν να χωρίζονται σε διαμερίσεις (partitions) στον ίδιο ή σε διαφορετικούς brokers,όπως φαίνεται στο σχήμα 2.7, όπου τα δεδομένα τοποθετούνται είτε τυχαία είτε βάσει της τιμής του κλειδιού τους στην ουρά μηνυμάτων για να οδηγηθούν στο κατάλληλο partition. Αυτό προσφέρει εξισορρόπησή δεδομένων και φόρτου αιτημάτων στους brokers και μία οργάνωση δεδομένων. Επίσης, μπορεί να γίνει αντιγραφή των partitions/topics σε οποιονδήποτε αριθμό server (replicated partition), παρέχοντας ασφάλεια δεδομένων με αντίγραφα σε πάνω από ένα μέρος. Η επιλογή της διαμέρισης που θα δημοσιοποιηθεί ή θα καταναλωθεί ένα μήνυμα γίνεται από τον ίδιο τον client και ο broker δεν ασκεί κάποιον έλεγχο στην επιλογή αυτή.

Οι καταναλωτές μπορούν να ανήκουν σε ομάδες καταναλωτών (consumer group), στις οποίες οι καταναλωτές συνεργάζονται για να λάβουν τα δεδομένα από ένα topic. Οι διαμερίσεις του topic μοιράζονται στα μέλη της ομάδας και όσο εισέρχονται και εξέρχονται μέλη από την ομάδα γίνεται εξισορρόπηση της ομάδας με τη διαρκή ανάθεση διαμερίσεων στα νέα μέλη. Όταν ξεκινάει να λαμβάνει δεδομένα ένας καταναλωτής ή μια ομάδα καταναλωτών, ο kafka διατηρεί συνεχώς τη θέση του μηνύματος που βρίσκεται ο καταναλωτής (offset) για κάθε topic, ώστε να μπορεί πάντα να λαμβάνει όσα δεδομένα έχασε σε περίπτωση αποσύνδεσης. Η αρχική θέση μπορεί να ρυθμιστεί ως η αρχή των μηνυμάτων (0) για τη λήψη όλων των μηνυμάτων στη σειρά



Σχήμα 2.7: *Kafka topics με partitions*



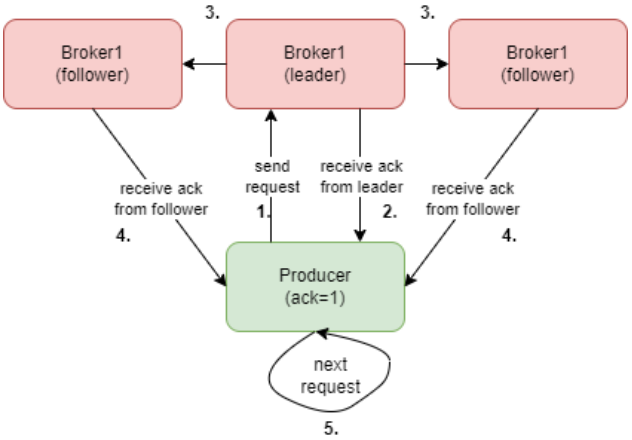
Σχήμα 2.8: *Kafka topics με acks=1*

που δημιουργήθηκαν, αλλά και ως το τέλος των μηνυμάτων για τη λήψη μηνυμάτων που δημιουργούνται από τη στιγμή της σύνδεσής του.

Για την επαλήθευση της παράδοσης των μηνυμάτων, ένας kafka producer χρησιμοποιεί την ρύθμιση acks. Το acks καθορίζει πόσες αναγνωρίσεις πρέπει να λάβει ο producer για να θεωρήσει ένα μήνυμα ως παραδομένο στον broker. Οι επιλογές είναι:

- acks=none: Ο producer θεωρεί το μήνυμα παραδομένο μόλις το δημοσιεύσει.
- acks=one: Ο producer αναμένει τον lead broker να στείλει επαλήθευση πώς το μήνυμα παραδόθηκε.
- acks=all: Ο producer αναμένει επαλήθευση από τον lead broker και από τους ακόλουθους brokers πως το μήνυμα παραδόθηκε σε όλους.

Στα σχήματα 2.8, 2.9 παρουσιάζονται οι αλληλεπιδράσεις μεταξύ broker/s-Producer για κάθε ρύθμιση acks.



Σχήμα 2.9: *Kafka topics με acks=all*

Όσες περισσότερες επαληθεύσεις αναμένει ο producer τόσο μικρότερος ο ρυθμός διεκπεραίωσης μηνυμάτων λόγω υψηλότερου κόστους, αλλά με μεγαλύτερή ανθεκτικότητα δεδομένων. Για κάθε εφαρμογή υπάρχει και η αντίστοιχη ρύθμιση, αναλόγως της ελαστικότητά της σε απώλεια δεδομένων.

Το kafka δεν υποστηρίζει απλή αυθεντικοποίηση με όνομα χρήστη και κωδικό για ασφαλής σύνδεση, αλλά κρυπτογράφηση με πιστοποιητικά SSL ή μηχανισμούς ασφαλείας μέσω του SASL που είναι μια βιβλιοθήκη για τη χρήση διάφορων μηχανισμών ασφαλείας στα υποστηριζόμενα πρωτόκολλα εφαρμογών που το υποστηρίζουν.

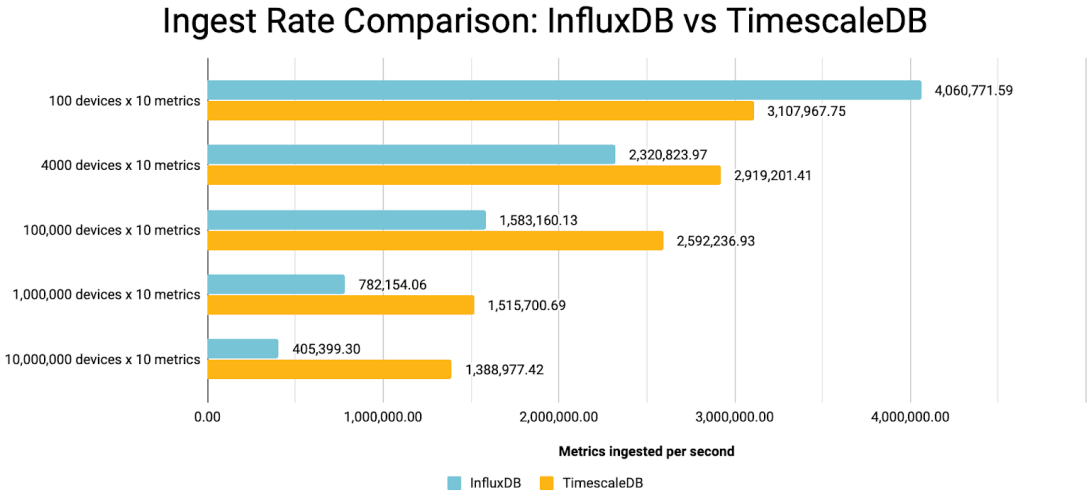
2.3 Αποθήκευση

Τα δεδομένα χρονοσειρών που συλλέγονται από τις συσκευές IoT, αναλόγως το πλήθος των συσκευών και τη δειγματοληψία τους, μπορούν να είναι κατά πολύ μεγαλύτερα από άλλους τύπους δεδομένων. Το μέγεθος τους και η ομοιομορφία τους (ζεύγη χρονοσήμανσης με δεδομένα) ώθησε στην δημιουργία εξειδικευμένων τρόπων αποθήκευσης και επεξεργασίας χρονοσειρών.

Από τις πιο γνωστές βάσεις δεδομένων για δεδομένα χρονοσειρών αποτελούν οι: InfluxDB,MongoDB,TimescaleDB,QuestDB. Η timescaleDB που επιλέχτηκε για την διπλωματική προσφέρει ταχύτητα σε μεγάλα σύνολα δεδομένων από μεγάλο αριθμό συσκευών σε σχέση με τις υπόλοιπες, όπως παρουσιάζεται και στην εικόνα 2.3 , και δεν απαιτεί μεγάλη τεχνογνωσία καθώς βασίζεται στην PostgreSQL που χρησιμοποιεί την ευρέος διαδεδομένη SQL.

2.3.1 TimescaleDB

Η TimescaleDB είναι μία επέκταση της σχεσιακής βάσης δεδομένων PostgreSQL, σχεδιασμένη για δημιουργία βάσεων αποθήκευσης δεδομένων χρονοσειρών. Περιέχει βελτιστοποιήσεις για ευκολότερη και γρηγορότερη επεξεργασία χρονοσειρών και



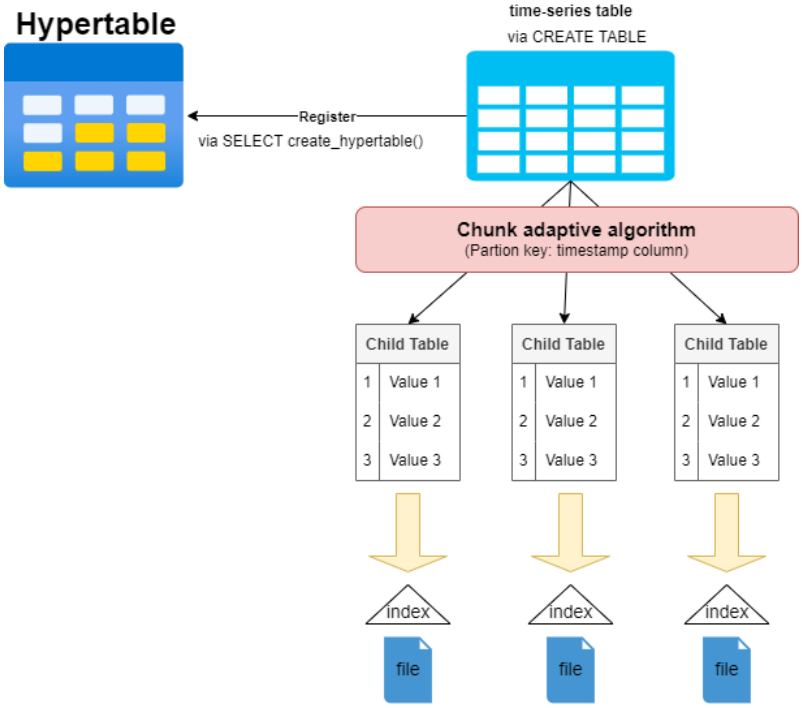
Εικόνα 2.3: *Σύγκριση του ρυθμού εισαγωγών δεδομένων ανάμεσα στην InfluxDB και την TimescaleDB*

παρέχει εργαλεία για την ανάλυσή τους.

Για την βελτιστοποίηση, χωρίζει συνεχώς τους πίνακες σε μικρότερα κομμάτια (chunks) μικρότερης διάρκειας, κυρίως βάσει της χρονοσήμανσης ή αλλιώς κάποιου άλλου κλειδιού, διατηρώντας τα δεδομένα και τους δείκτες (indexes) στη μνήμη για γρηγορότερες εισαγωγές και ερωτήματα σε πρόσφατα δεδομένα. Ένας πίνακας στην TimescaleDB ονομάζεται hypertable (υπερπίνακας) και είναι στην πραγματικότητα μία εικονική όψη πολλών ξεχωριστών πινάκων, των προαναφερθέντων chunks. Η διαμέριση, πέρα από στήλη χρονοσήμανσης, μπορεί να γίνει και βάσει άλλης στήλης, δημιουργώντας πίνακα κατακερματισμού και ομαδοποιεί τα δεδομένα βάσει αυτού του χαρακτηριστικού. Η διαδικασία μετατροπής ενός απλού πίνακα σε υπερπίνακα παρουσιάζεται στο σχήμα 2.10.

Οι πιο πρόσφατοι υποπίνακες μπορούν να βρίσκονται συνεχώς στη μνήμη για γρηγορότερες εισαγωγές και ερωτήματα, παραλείποντας την πιο αργή πρόσβαση από τον δίσκο. Κάθε υποπίνακας αποκτά τον δικό του δείκτη κατά τη δημιουργία του εξασφαλίζοντας την ύπαρξη αυτών και των πιο προσφάτων δεδομένων στη μνήμη για γρήγορη εισαγωγή δεδομένων και ενημέρωση δεικτών. Η ύπαρξη υποπινάκων διευκολύνει την διαγραφή δεδομένων συγκεκριμένου εύρους και την επιλογή διατήρησης δεδομένων μόνο συγκεκριμένου χρονικού διαστήματος, καθώς δεν πραγματοποιείται διαγραφή σειρών, αλλά των υποπινάκων ολόκληρων. Αυτό μεταφράζεται σε απλή διαγραφή αρχείων που σε αντίθεση με την διαγραφή ξεχωριστών σειρών από μια βάση είναι αρκετά πιο επιεικές σε πόρους και χρόνο. Η αρχιτεκτονική υποπινάκων επιτρέπει στην TimescaleDB να πραγματοποιεί συμπίεση αυτών με τη μετατροπή τους από μορφή ¨κατά-γραμμή¨ (row-major form) σε μία πιο στηλοειδής διάταξη.

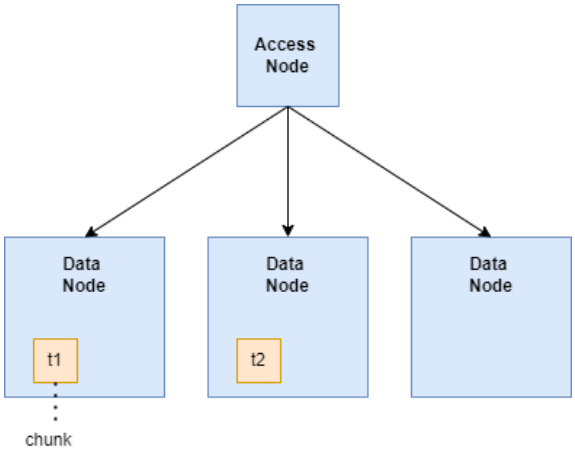
Η TimescaleDB υποστηρίζει κατανεμημένους υπερπίνακες σε πολλούς κόμβους



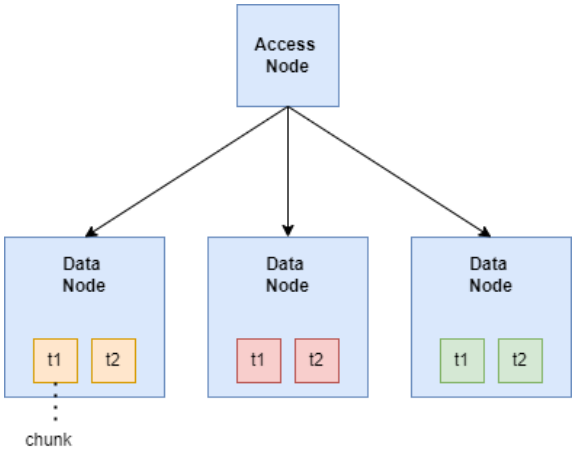
Σχήμα 2.10: *Δημιουργία υπερπίνακα*

για την προσαρμογή του συστήματος σε μεγαλύτερο όγκο δεδομένων, αποθηκεύοντας τους κατανεμημένους υποπίνακες σε μία η περισσότερες βάσεις TimescaleDB ενόσω συμπεριφέρεται ως ένας συνεχόμενος πίνακας. Μια βάση δεδομένων που συμμετέχει στο κατανεμημένο σύστημα μπορεί να είναι είτε κόμβος πρόσβασης (access node) είτε κόμβος δεδομένων (data node), αλλά όχι και τα δύο. Η εισαγωγή στο κατανεμημένο σύστημα επιτυγχάνεται ανάλογα με την διαμέριση του υπερπίνακα: στην περίπτωση που έχουμε μονοδιάστατη διαμέριση χρόνου (time partitioning), οι υποπίνακες μοιράζονται ισορροπημένα σε όλους τους κόμβους δεδομένων όπως φαίνεται στο σχήμα 2.11. Στην περίπτωση που έχουμε διαμέριση χώρου και χρόνου time and space partitioning, οι υποπίνακες μοιράζονται στους κόμβους βάσει της δεύτερης στήλης διαμέρισης που επιλέχτηκε, με αποτέλεσμα κάθε κόμβος να περιέχει δεδομένα με το ίδιο χαρακτηριστικό όπως φαίνεται στο σχήμα 2.12.

Ο κόμβος πρόσβασης διανέμει τα αιτήματα και τα ερωτήματα στους κατάλληλους κόμβους δεδομένων και συγκεντρώνει τα αποτελέσματα. Η εισαγωγή ή αφαίρεση εξυπηρετητή από το cluster γίνεται ελαστικά χωρίς την ανάγκη άμεσης εξισορρόπησης. Όταν δημιουργείται νέος εξυπηρετητής οι προϋπάρχοντες υποπίνακες παραμένουν στη θέση τους και κατά τη δημιουργία καινούριων, τα δεδομένα διαμερίζονται σε όλο το σύνολο, νέων και παλιών, εξυπηρετητών. Στο κατανεμημένο σύστημα, πέρα από τον διαμοιρασμό των υποπινάκων στους κόμβους βάσει κάποιου χαρακτηριστικού, υπάρχει και η δυνατότητα δημιουργίας πανομοιότυπων (replicas) σε πάνω από έναν εξυπηρετητή για την ασφάλεια των δεδομένων.



Σχήμα 2.11: *Διαμέριση χρόνου (time partitioning)*



Σχήμα 2.12: *Διαμέριση χώρου-χρόνου (time-space partitioning)*

Κεφάλαιο 3

Εξυπηρετητής

Στο κεφάλαιο αυτό γίνεται περιγραφή του εξυπηρετητή που βρίσκονται τα επιμέρους στοιχεία που χρησιμοποιούνται απο την πλατφόρμα.

3.1 Εξυπηρετητής

Για την παρούσα διπλωματική εργασία χρησιμοποιείται ως εξυπηρετητής σύστημα με λειτουργικό σύστημα linux (Ubuntu 20.04). Στον εξυπηρετητή θα βρίσκεται η πλατφόρμα όσο και η βάση, ο MQTT broker και ο Kafka broker για μεγαλύτερη ευκολία. Τα στοιχεία αυτά μπορούν στην πραγματικότητα να βρίσκονται σε διαφορετικό εξυπηρετητή το καθένα.

3.1.1 MQTT Broker

Για MQTT Broker χρησιμοποιείται ο Eclipse Mosquitto[9] καθώς επιφέρει μικρό αντίκτυπο στους πόρους του συστήματος και είναι πρόγραμμα ανοικτού κώδικα.

Εγκατάσταση

Η εγκατάσταση του mosquitto broker γίνεται μέσω του διαχειριστή πακέτων apt καθώς βρίσκεται στα επίσημα αποθετήρια για τις διανομές linux που το υποστηρίζουν (debian,ubuntu, κτλ).

```
$ sudo apt update && sudo apt install mosquitto -y
```

Αυτή η εντολή αναλαμβάνει την εγκατάσταση του mosquitto broker από τα επίσημα αποθετήρια και την εκκίνηση της υπηρεσίας. Μετά την εκτέλεση της εντολής μπορεί να γίνει έλεγχος της κατάστασης του broker μέσω της εντολής `sudo systemctl status mosquitto` η οποία δίνει ένα παρόμοιο αποτέλεσμα με την εικόνα 3.1.

Για την σύνδεση στον broker θα χρησιμοποιήσουμε την υλοποίηση mqtt client mosquitto

```
$ sudo apt install mosquitto-clients -y
```

```
anastasios@digit:~$ sudo systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2022-06-20 17:20:57 EEST; 10min ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
 Main PID: 652 (mosquitto)
    Tasks: 1 (limit: 4915)
   Memory: 2.9M
   CGroup: /system.slice/mosquitto.service
           └─652 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

Jun 20 17:20:55 digit systemd[1]: Starting Mosquitto MQTT Broker...
Jun 20 17:20:57 digit systemd[1]: Started Mosquitto MQTT Broker.
```

Εικόνα 3.1: *Κατάσταση υπηρεσίας mosquitto*

Που μας επιτρέπει μέσω του τερματικού να εγγραφόμαστε και να δημοσιεύουμε σε έναν mqtt broker.

Παραμετροποίηση

Τα βοηθητικά αρχεία για το mosquitto βρίσκονται στον φάκελο /etc/mosquitto/. Το πιο σημαντικό αρχείο είναι το /etc/mosquitto/mosquitto.conf που αποτελεί το κύριο αρχείο ρυθμίσεων του mosquitto. Στο αρχείο αυτό μπορούμε να ορίσουμε τις θύρες δικτύου που θα ακούει ο mosquitto broker αλλα και τις ρυθμίσεις σύνδεσης της εκάστοτε θύρας. Μπορούν να υπάρχουν πολλαπλές θύρες, ονόματι listeners, η κάθε μία με ξεχωριστές ρυθμίσεις αλλά δίνοντας πρόσβαση στα ίδια δεδομένα. Για τις ανάγκες της διπλωματικής θα ορίσουμε 2 listeners, έναν μόνο με αυθεντικοποίηση κωδικού και έναν με κρυπτογράφηση σύνδεσης.

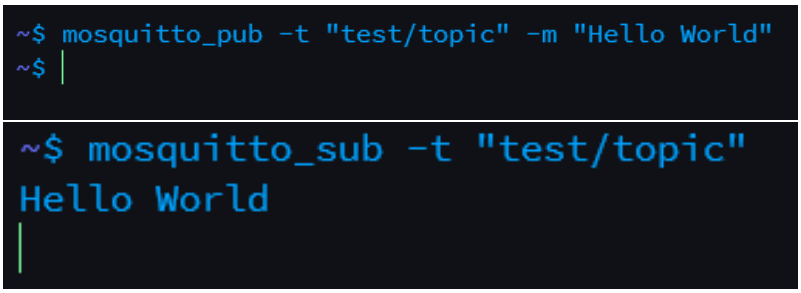
Το mosquitto υποστηρίζει κατακερματισμό κωδικών σε αρχείο για μεγαλύτερη ασφάλεια, μέσω του εργαλείου mosquitto_passwd. Με την χρήση αυτού του εργαλείου μπορούμε να μετατρέψουμε αρχείο με περιεχόμενο της μορφής username:password σε κατακερματισμένο αρχείο κωδικών. Δημιουργείται το αρχείο /etc/mosquitto/passwd με τα επιθυμητά ζεύγη αυθεντικοποίησης και εκτελείται το εργαλείο ως εξής:

```
$ sudo mosquitto_passwd -U /etc/mosquitto/passwd
```

Για την θύρα 1883 που θα απαιτεί αυθεντικοποίηση χρήστη, το αρχείο παραμετροποίησης θα πρέπει να περιλαμβάνει:

```
listener 1883
allow_anonymous false
password_file /etc/mosquitto/passwordfile
```

Με αυτές τις ενέργειες, ο broker είναι διαθέσιμος στον εξυπηρετητή στην θύρα 1883 και μπορεί να γίνει δοκιμή με την εγγραφή σε κάποιο topic με την εντολή `mosquitto_sub` και την δημοσίευση στο ίδιο topic με την εντολή `mosquitto_pub`, όπως φαίνεται στην εικόνα 3.2



Εικόνα 3.2: *pub και sub στο mosquitto*

Κρυπτογράφηση

Ο Mosquitto broker υποστηρίζει κρυπτογράφηση TLS μέσω πιστοποιητικών x509 (mqtts). Η έκδοση τέτοιον πιστοποιητικών γίνεται μέσω του εργαλείου openssl [10]. Τα αρχεία που χρειάζονται για την επιτυχή κρυπτογράφηση είναι το public Certificate Authority (CA) (δημόσια αρχή έκδοσης πιστοποιητικών) και τα ζεύγη κλειδιών-πιστοποιητικών για τον broker και τους clients τα οποία θα υπογράφονται απο το public CA. Η διαδικασία για την έκδοση των αρχείων έχει ως εξής:

1. Δημιουργία κλειδιού CA

```
$     openssl genrsa -out ca.key 4096
```

2. Δημιουργία πιστοποιητικού CA

```
$     openssl req -new -x509 -extensions v3_ca -key ca.key -out ca.crt
```

3. Δημιουργία κλειδιού broker

```
$     openssl genrsa -out server.key 4096
```

4. Δημιουργία πιστοποιητικού broker με υπογραφή του CA

```
$     openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key
$     > -CAcreateserial -out server.crt
```

5. Δημιουργία κλειδιού client

```
$     openssl genrsa -out client.key 4096
```

6. Δημιουργία πιστοποιητικού client με υπογραφή του CA

```
$     openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key
$     > -CAcreateserial -out client.crt
```

Επειδή το mosquitto απαιτεί τα πιστοποιητικά να είναι της μορφής PEM [11] μπορεί μέσω του openssl να γίνει μετατροπή των πιστοποιητικών αρχείων τύπου .crt (ca.crt,server.crt,client.crt) με την εξής εντολή:

```
$ openssl x509 -in file.crt -out file.pem -outform PEM
```

Έτσι καταλήγουμε με τα αρχεία πιστοποίησης ca.pem,server.pem,server.key, client.pem,client.key. Για την ενεργοποίηση της κρυπτογράφησης δεδομένων στην θύρα 8883 του mosquitto προσθέτουμε το εξής κομμάτι στο αρχείο παραμετροποίησης mosquitto.conf:

```
listener 8883
allow_anonymous true
require_certificate true
cafile /path/to/ca.pem
keyfile /path/to/server.key
certfile /path/to/server.pem
use_identity_as_username true
```

Οπότε για να συνδεθεί ένας client στην θύρα 8883 του broker πρέπει να χρησιμοποιήσει τα αρχεία ca.pem,client.pem,client.key.

3.1.2 Kafka Broker

Εγκατάσταση

Η εγκατάσταση του Apache Kafka broker γίνεται με κατέβασμα των αρχείων από την επίσημη ιστοσελίδα και με την αποσυμπίεση τους σε επιλεγμένο φάκελο.

```
$ curl "https://dlcdn.apache.org/kafka/3.2.0/kafka_2.13-3.2.0.tgz"
$ > -o ~/Downloads/kafka.tgz
$ mkdir ~/kafka && cd ~/kafka
$ tar -xvzf ~/Downloads/kafka.tgz --strip 1
```

Για μεγαλύτερη αυτονομία, δημιουργείται υπηρεσία για την αυτόματη εκκίνηση από το ίδιο το σύστημα. Στις διανομές linux βασισμένες σε debian η διαχείριση των υπηρεσιών γίνεται από τον systemd [12] και επιτρέπει την εκκίνηση των εφαρμογών κατά την εκκίνηση.

Ο kafka broker χρησιμοποιεί την εφαρμογή zookeeper για τον έλεγχο κατάστασής των διαφορετικών κόμβων και την διατήρηση μιας λίστας των topics και μηνυμάτων. Η εφαρμογή αυτή βρίσκεται στα αρχεία που δημιουργήθηκαν παραπάνω και για την αυτόματη εκκίνηση του δημιουργείται αρχείο υπηρεσίας στην τοποθεσία /etc/systemd/system με όνομα zookeeper.service και περιεχόμενο

```
[Unit]
Requires=network.target remote-fs.target
After=network.target remote-fs.target

[Service]
```

```
Type=simple
User=username
ExecStart=/home/username/kafka/bin/zookeeper-server-start.sh /
    home/kafka/kafka/config/zookeeper.properties
ExecStop=/home/username/kafka/bin/zookeeper-server-stop.sh
Restart=on-abnormal

[Install]
WantedBy=multi-user.target
```

Αντίστοιχα δημιουργείται αρχείο στην ίδια τοποθεσία με όνομα kafka.service για τον kafka broker

```
[Unit]
Requires=zookeeper.service
After=zookeeper.service

[Service]
Type=simple
User=username
ExecStart=/bin/sh -c '/home/username/kafka/bin/kafka-server-start
    .sh /home/kafka/kafka/config/server.properties > /home/kafka/
    kafka/kafka.log 2>&1'
ExecStop=/home/username/kafka/bin/kafka-server-stop.sh
Restart=on-abnormal

[Install]
WantedBy=multi-user.target
```

Με την ύπαρξη αυτών των αρχείων, η εκκίνηση του kafka broker μπορεί να γίνει με την εντολή

```
$ sudo systemctl start kafka
```

Για να εκκινείται ο broker σε κάθε εκκίνηση του συστήματος, πρέπει να ενεργοποιηθούν οι υπηρεσίες με τις εξής εντολές:

```
$ sudo systemctl enable zookeeper
$ sudo systemctl enable kafka
```

Παραμετροποίηση

Οι ρυθμίσεις του broker βρίσκονται στο αρχείο kafka/config/server.properties στο οποίο αξίζει να σημειωθούν οι ρυθμίσεις listeners και advertised.listeners. Η

ρύθμιση listeners θέτει την διεύθυνση και την θύρα στην οποία ο kafka broker λαμβάνει δεδομένα και η ρύθμιση advertised.listeners θέτει την διεύθυνση που στέλνει στους clients και αν δεν ρυθμιστεί παιρνει την ίδια τιμή με το listeners. Για τον εξυπηρετητή οι τιμές λαμβάνουν την εξής μορφή:

- listeners:PLAINTEXT://0.0.0.0:9092
- advertised.listeners:PLAINTEXT://server.hostname:9092

Επίσης το αρχείο παραμετροποίησης server.properties περιέχει εγγραφή με το χαρακτηριστικό auto.create.topics.enable που όταν είναι true δεν χρειάζεται να δημιουργηθούν χειροκίνητα τα νεα topics και δημιουργούνται κατά το 1ο μήνυμα που λαμβάνουν με ρυθμίσεις τις προκαθορισμένες που βρίσκονται στο ίδιο αρχείο. Για διαφορετικές ρυθμίσεις πρέπει να δημιουργθεί χειροκίνητα.

Κρυπτογράφηση

Για την ενεργοποίηση της κρυπτογράφησης σύνδεσης μέσω TLS, απαίτουνται όπως και στον mosquitto πιστοποιητικα. Μπορουν να χρησιμοποιηθούν τα ίδια, αλλά επειδή απαιτείται μορφή .jks πρέπει να γίνουν οι απαραίτητες τροποποιήσεις για την μετατροπή τους σε αυτή την μορφή.

Για την ενεργοποίηση της κρυπτογράφησης στην θύρα 9093, οι listeners και advertised.listeners στο αρχείο ρυθμίσεων παίρνουν την εξής μορφή:

```
listeners=SSL://0.0.0.0:9093
advertised.listeners=SSL://server.hostname:9093
```

Έπειτα μπορούν να προστεθούν οι εξής γραμμές στο αρχείο ρυθμίσεων για την αναγνώριση των πιστοποιητικών:

```
ssl.keystore.location=/path/to/server.keystore.jks
ssl.keystore.password=password
ssl.key.password=password
ssl.truststore.location=/path/to/server.truststore.jks
ssl.truststore.password=password
ssl.client.auth = required
ssl.enabled.protocols = TLSv1.2,TLSv1.1,TLSv1
```

3.1.3 TimescaleDB

Εγκατάσταση

Η εγκατάσταση της timescaleDB γίνεται μέσω του apt package manager αφου γίνει η εγκατάσταση των απαραίτητων πακέτων και η προσθήκη του αποθετηρίου που βρίσκεται στα αποθετήρια του συστήματος:

```
$ sudo apt install gnupg postgresql-common apt-transport-https lsb-release wget
```

```
$ sudo /usr/share/postgresql-common/pgdg/apt.postgresql.org.sh
$ sudo echo "deb␣https://packagecloud.io/timescale/timescaledb/debian/
$ >␣$(lsb_release -c -s) main" > /etc/apt/sources.list.d/timescaledb.list
$ wget --quiet -O - https://packagecloud.io/timescale/timescaledb/gpgkey |
$ > apt-key add -
```

```
$ sudo apt install timescaledb-2-postgresql-14
```

Η timescale αποτελεί επέκταση της postgres που σημαίνει οτι για την χρήση της χρειάζεται ενεργοποίηση στην βάση που θα χρησιμοποιείται. Για την ενεργοποίηση της, πρέπει να γίνει σύνδεση και (αν δεν υπάρχει) δημιουργία βάσης.

```
$ su postgres -c psql
$ tsdb=> CREATE database example;
$ tsdb=> \c example
$ tsdb=> CREATE EXTENSION IF NOT EXISTS timescaledb;
```

Η μετατροπή ενός καινούριου πίνακα σε υπερπίνακα έχει προϋπόθεσή την ύπαρξη στήλης χρονικών σημάνσεων (timestamp) καθώς βάση αυτών των δεδομένων θα γίνει η δημιουργία του. Η δημιουργία του γίνεται μέσα από την βάση με την εντολή

```
$ tsdb=>SELECT create_hypertable('table_name', 'time');
```

όπου time το όνομα της στήλης των χρονοσημάνσεων. Αυτό καθιστά τον πίνακα έτοιμο να δεχθεί πολλά δεδομένα χρονοσειρών αξιοποιώντας την αποδοτικότητα της timescale και τα εργαλεία για χρονοσειρές που προσφέρει.

Παραμετροποίηση

Η postgres χρησιμοποιεί 2 αρχεία για τις πιο σημαντικές ρυθμίσεις:

- postgresql.conf: Το αρχείο που περιέχει τις ρυθμίσεις λειτουργίας.
- pg_hba.conf: Το αρχείο που περιέχει τις επιτρεπόμενες συνδέσεις στην βάση.

Για την πρόσβαση απο απομακρυσμένες τοποθεσίες εκτός τοπικού δικτύου στην βάση πρέπει να προστεθούν γραμμές στο αρχείο pg_hba.conf της μορφής:

```
host        DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
```

Όπου database το όνομα της βάσης που δίνει πρόσβαση, user το όνομα χρήστη που μπορεί να χρησιμοποιηθεί , address η διεύθυνση που έχει δικαίωμα πρόσβασης στη βάση και method ο τρόπος αυθεντικοποίησης χρήστη. Για την απομακρυσμένη πρόσβαση απο παντού με οποιονδήποτε χρήστη δηλωμένο στην εφαρμογή προστέθηκαν οι εξής γραμμές:

```
host  all             all             0.0.0.0/0               md5
host  all             all             ::/0                    md5
```

Αυτές επιτρέπουν σε όλους τους clients με οποιαδήποτε διεύθυνση (είτε μορφής IPv4 ή IPv6) να συνδεθούν με αυθεντικοποίηση μέσω κωδικού (md5).

Κρυπτογράφηση

Για την ενεργοποίηση της κρυπτογράφησης σύνδεσης μέσω πιστοποιητικών μπορούν να χρησιμοποιηθούν τα ίδια πιστοποιητικά που δημιουργήθηκαν για τον mqtt broker. Η ανάθεση τους γίνεται μέσω του αρχείου postgresql.conf με την προσθήκη των εξής γραμμών:

```
ssl = on
ssl_ca_file = '/path/to/ca.crt'
ssl_cert_file = '/path/to/server.crt'
ssl_key_file = '/path/to/server.key'
ssl_prefer_server_ciphers = on
```

Για την ενεργοποίηση της κρυπτογραφημένης σύνδεσης, αλλάζουν οι γραμμές που προστέθηκαν στο pg_hba.conf με τον εξής τρόπο:

```
hostssl all             all             0.0.0.0/0
   md5
hostssl all             all             ::/0
   md5
```

Κεφάλαιο 4

Πλατφόρμα

Στο κεφάλαιο αυτό παρουσιάζεται η λειτουργία, η αρχιτεκτονική, η υλοποίηση και τα επιμέρους κομμάτια της πλατφόρμας.

4.1 Λειτουργία

Η πλατφόρμα που αναπτύσσεται στα πλαίσια της διπλωματικής προωθεί τα δεδομένα που προέρχονται από συσκευές IoT σε βάση δεδομένων και σε kafka broker βάση των ρυθμίσεων που του έχουν δοθεί, όπως παρουσιάζεται στο σχήμα 4.1.

Τα μηνύματα από τις συσκευές θεωρούνται ότι είναι της μορφής json[13]. Δηλαδή πακέτα δεδομένων που περιέχουν ζεύγη κλειδιού-τιμής π.χ. {key:value, key:value, ...}. Αυτό καθιστά τα δεδομένα εύκολα στην αναγνώριση και την αποσειριοποίηση τους.

Κάθε mqtt topic θα αναλαμβάνει έναν μόνο τύπο δεδομένων, δηλαδή μία μορφή json. Αυτό μπορεί να σημαίνει πώς κάθε συσκευή θα πρέπει να αποστέλλει σε ένα topic ή αλλιώς αν παράγει πάνω από 1 είδος μηνύματος να τα προωθεί σε ξεχωριστά topics.

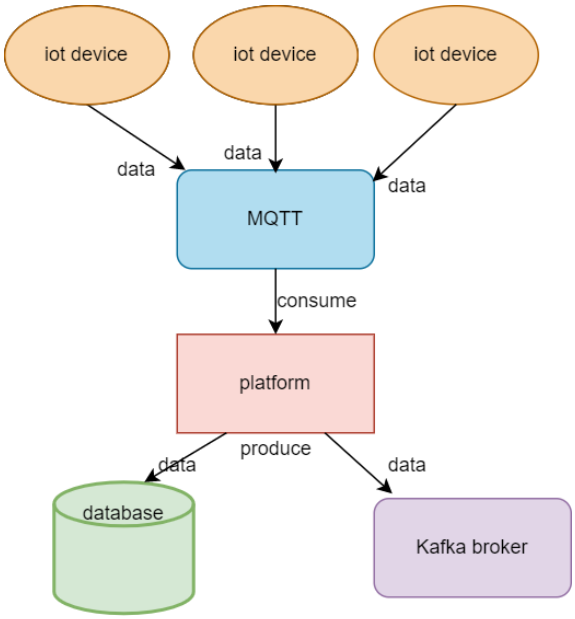
4.1.1 Ρυθμίσεις

Η πλατφόρμα δέχεται ρυθμίσεις από αρχεία σε συγκεκριμένο φάκελο. Οι βασικές ρυθμίσεις περιλαμβάνουν διευθύνσεις και λοιπές ιδιότητες για την σύνδεση της σε mqtt broker,kafka broker και βάση δεδομένων.

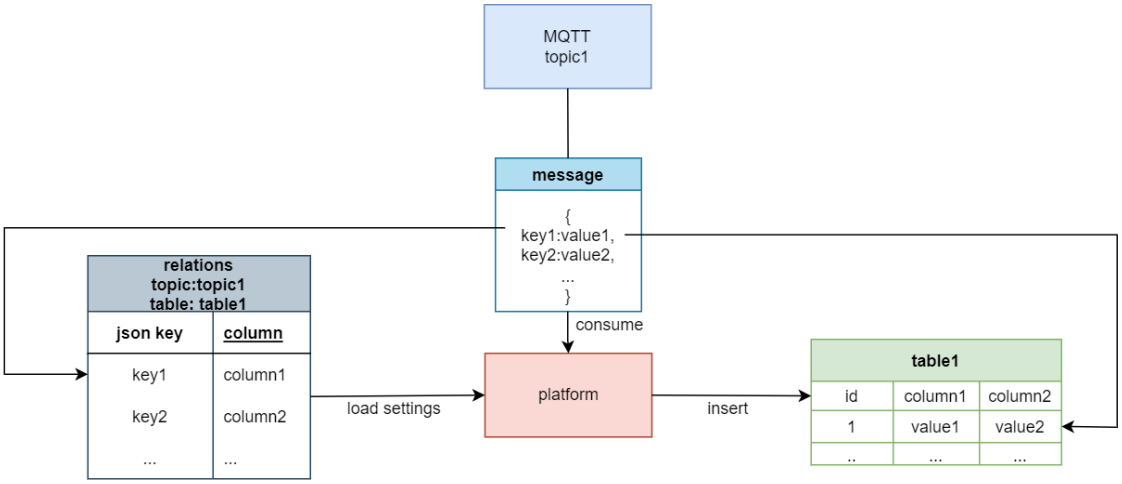
Οι υπόλοιπες ρυθμίσεις αφορούν τον τρόπο αποσειριοποίησης των δεδομένων για την εισαγωγή τους στις σωστές θέσεις της βάσης και στα σωστά kafka topics.

4.1.2 Εισαγωγή σε βάση

Για την εισαγωγή στη βάση δίνονται σχέσεις mqtt topic-πίνακα που θα δίνει την δυνατότητα εισαγωγής δεδομένων που προέρχονται από αυτό το topic μέσα σε αυτόν



Σχήμα 4.1: *Λειτουργία πλατφόρμας*



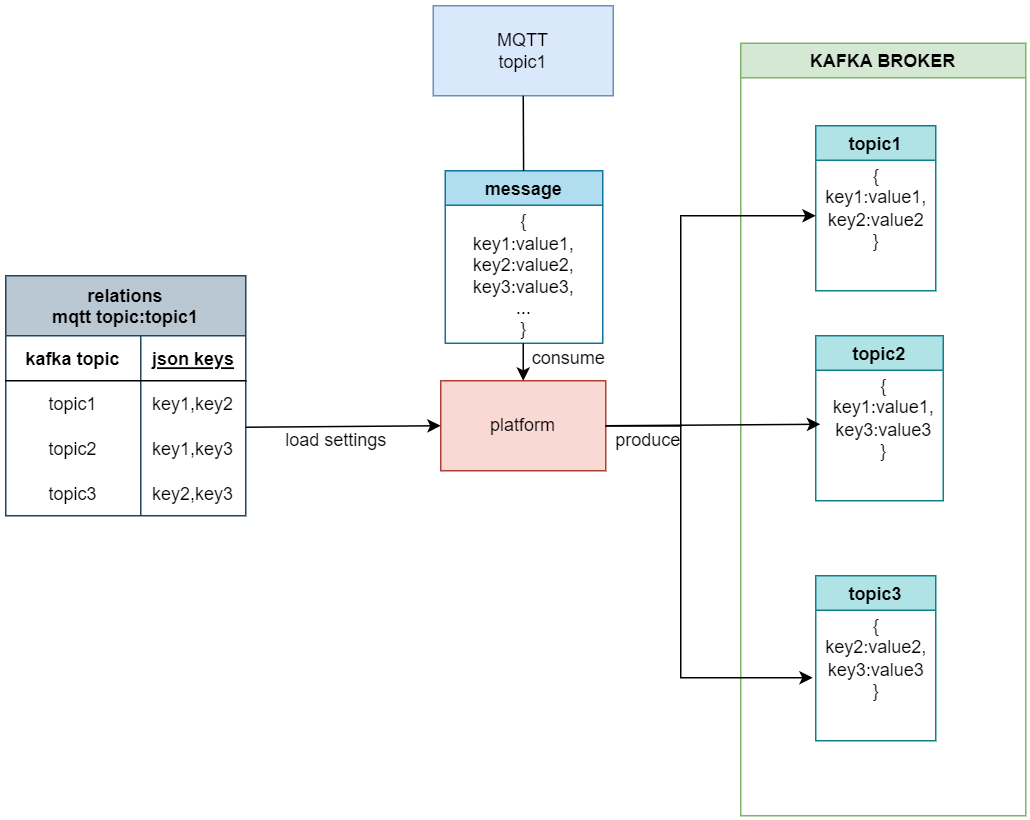
Σχήμα 4.2: *Εισαγωγή σε βάση δεδομένων*

τον πίνακα όπως παρουσιάζεται στο σχήμα 4.2. Για την επιλογή των τιμών που θα εισαχθούν στον πίνακα από το topic δίνονται σχέσης json-key-στήλη πίνακα.

Κατά την λειτουργία της πλατφόρμας θα εισάγονται όσα ξεχωριστά δεδομένα είναι αναγκαία από οποιοδήποτε mqtt topic σε έναν πίνακα. Δεν υπάρχει περιορισμός στο πλήθος των topic που εισάγουν σε ένα πίνακα αλλά κάθε στήλη έχει προέλευση από ένα topic.

4.1.3 Προώθηση σε kafka

Η προώθηση μηνυμάτων στον kafka broker ακολουθεί τις σχέσεις json key - kafka topic για κάθε mqtt topic που βρίσκονται στο αρχείο ρυθμίσεων. Υπάρχει δυνατότητα προώθησης οποιασδήποτε τιμής από το μήνυμα σε οποιοδήποτε kafka



Σχήμα 4.3: *Προώθηση σε kafka broker*

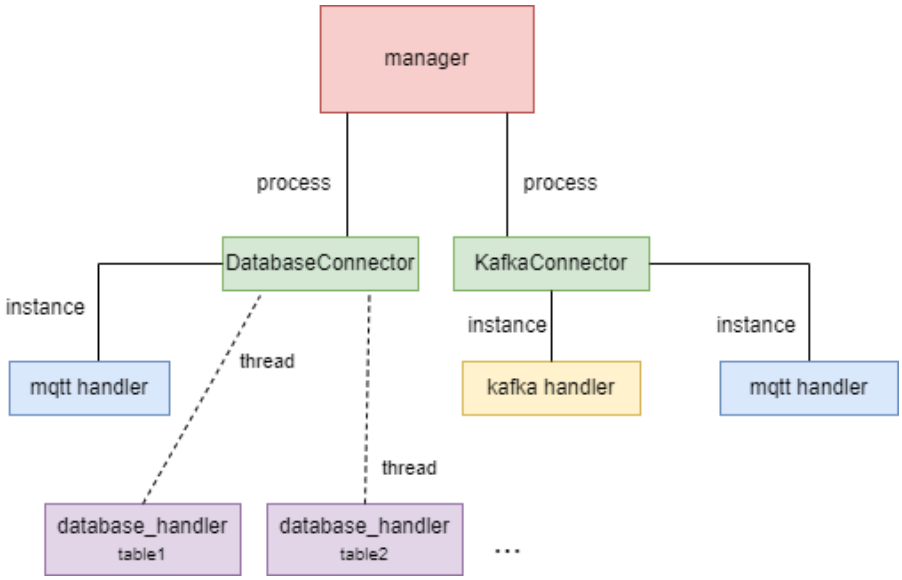
topic. Η πλατφόρμα δημιουργεί μήνυμα μορφής json με περιεχόμενα τα επιλεγμένα κλειδιά από το μήνυμα και τα προωθεί στο επιλεγμένο kafka topic. Η διαδικασία αυτή μπορεί να επαναληφθεί πολλές φορές για κάθε mqtt topic, προωθώντας διάφορους συνδυασμούς δεδομένων σε ξεχωριστά kafka topics. Η λειτουργία προώθησης δεδομένων σε kafka παρουσιάζεται στο σχήμα 4.3

4.2 Αρχιτεκτονική και Υλοποίηση

Η ανάπτυξη έγινε σε γλώσσα python3 με χρήση αντικειμενοστραφούς προγραμματισμού για τη δημιουργία κλάσεων για το κάθε επιμέρους κομμάτι. Τα κύρια κομμάτια της πλατφόρμας είναι:

- manager: Ο διαχειριστής όλων των επιμέρους κομματιών της πλατφόρμας.
- mqtt handler: Ο client που θα δέχεται τα δεδομένα από τον MQTT broker
- kafka handler: Ο client που θα προωθεί δεδομένα στον kafka broker
- database handler: Ο client που θα εισάγει δεδομένα στην βάση δεδομένων.
- connector: Ο σύνδεσμος μεταξύ mqtt client και kafka/database client.

Η βασική αρχιτεκτονική του προγράμματος περιγράφεται στην εικόνα 4.4. Ο manager δημιουργεί δύο ξεχωριστές διεργασίες, μια για την προώθηση των δεδομένων στον kafka broker και μία για την εισαγωγή των δεδομένων στην βάση δεδομένων. Κάθε διεργασία δημιουργεί από ένα mqtt handler για την λήψη των μηνυ-



Σχήμα 4.4: *Αρχιτεκτονική Πλατφόρμας*

μάτων από τον broker. Η διεργασία για την προώθηση στον kafka δημιουργεί τον kafka handler που θα προωθεί τα δεδομένα που λαμβάνει ο mqtt handler της στον kafka broker. Η διεργασία για την εισαγωγή των δεδομένων στη βάση δεδομένων δημιουργεί τόσα database handler όσα οι πίνακες της βάσης που θα χρησιμοποιούμε, το κάθε ένα στο δικό του νήμα, που θα εισάγει δεδομένα που λαμβάνει από τον mqtt handler της,

4.2.1 MQTT handler

Στην ενότητα αυτή παρουσιάζεται η υλοποίηση του εξυπηρετητή MQTT της πλατφόρμας που θα διαχειρίζεται την επικοινωνία και την λήψη μηνυμάτων από MQTT brokers. Για την επικοινωνία με MQTT brokers χρησιμοποιήθηκε η βιβλιοθήκη paho-mqtt[14].

Ο mqtt handler περιέχει την κλάση mqtt client με την μορφή που παρουσιάζεται στο σχήμα 4.5.

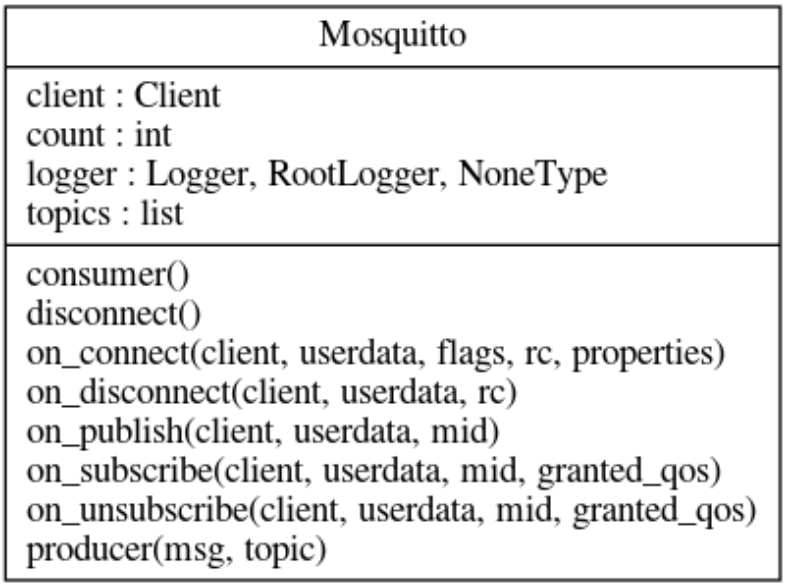
Οι εισαγωγές τρίτων βιβλιοθηκών είναι οι εξής:

```
import ssl

import paho.mqtt.client as mqtt
import yaml
```

Ο κατασκευαστής της κλάσης δημιουργεί το mqtt paho client που είναι υπεύθυνο για όλες τις ενέργειες από και προς τον mqtt broker. Σε αυτό το στάδιο δίνονται και τα απαραίτητα στοιχεία ταυτοποίησης που ζητάει ο broker για την σύνδεση σε αυτόν που είναι όνομα και κωδικός χρήστη, ή πιστοποιητικά ή και ο συνδυασμός αυτών.

```
def __init__(self)
    self.client = Client(client_id=id,clean_session=False)
```



Σχήμα 4.5: *MQTT client class*

```
#username/password authentication
self.client.username_pw_set("username","password")
#tls
self.client.tls_set("CA cert file")
```

οπου client_id θα είναι το όνομα του client για την αναγνώρισή του από τον broker και η τιμή του clean_session ορίζει το είδος της συνεδρίας, που στην περίπτωσή μας είναι αναγκαίο να είναι επίμονη (persistent session).

Αυτό το instance είναι ο εξυπηρετητής MQTT που θα χρησιμοποιηθεί από το πρόγραμμα για όλες τις λειτουργίες. Κάθε λειτουργία που θα ενσωματωθεί στον εξυπηρετητή θα περαστεί ως callback function στο αντικείμενο Client για το να κληθεί στο γεγονός που θα στοχεύει. Callback function ονομάζεται συνάρτηση που δίνεται σαν παράμετρος σε άλλη συνάρτηση ή αντικείμενο για να κληθεί από αυτό. Τα γεγονότα υποστηρίζονται είναι, μεταξύ και άλλων :

- connect: Επιτυχής σύνδεση σε broker.
- disconnect: Αποσύνδεση από broker.
- subscribe: Επιτυχής εγγραφή σε λίστα topics.
- unsubscribe: Απ-εγγραφή από topic.
- message: Λήψη μηνύματος από topic.
- publish: Αποστολή μηνύματος σε topic.

Για την σωστή διαχείρισή μηνυμάτων πρέπει να δημιουργηθεί συνάρτηση για το γεγονός message που θα επεξεργάζεται καταλλήλως το μήνυμα. Αυτή θα εκτελείται

σε συνέχεια της λήψης μηνύματος από κάποιο topic στο οποίο είναι εγγεγραμμένος ο client. Μια τέτοια απλή συνάρτηση που θα εμφανίζει τα μηνύματα μπορεί να έχει την εξής μορφή:

```
def on_message_callback(client, userdata, message):
    print("received message =",str(message.payload.decode("utf-8")))
```

Απο τις παραμέτρους μας ενδιαφέρει το message που είναι και το μήνυμα που λήφθηκε. Περίεχει μέλη : topic,payload,qos,retain,mid, οπου topic το topic που δημοσιεύτηκε το μήνυμα, payload τα δεδομενα του μηνύματος, qos το qos του μηνύματος, retain αν είναι διατηρημένο μήνυμα από προηγούμενη αποστολή η καινούριο και mid το id του μηνύματος. Το payload είναι σε μορφή byte/byte array έτσι για την σωστή επεξεργασία του πρέπει να δεχθεί αποκωδικοποίησή σε αναγνώσιμη μορφή, όπως γίνεται στο παραπάνω παράδειγμα με τη χρήση της μεθόδου decode.

Καθώς στην πλατφόρμα ορίζονται δύο διαφορετικές συνθήκες για την επεξεργασία των ληφθέντων μηνυμάτων, μια για την προώθηση στον kafka και μία για την εισαγωγή στη βάση δεδομένων, θα υλοποιηθούν δύο ξεχωριστές callback function. Ορίζονται στο κομμάτι του connector και θα περιγραφούν παρακάτω.

Η εγγραφή του client στα επιθυμητά topics πρέπει να γίνει μετά την σύνδεση του στον broker, έτσι είναι αναγκαία η δημιουργία callback function για το γεγονός connect που θα εκτελεί την εγγραφή:

```
def on_connect_callback(self, client, userdata, flags, rc, properties):
        self.client.subscribe(self.topics)
```

Η μεταβλητή topics είναι λίστα από tuples (πλειάδα) με μορφή (όνομα topic, qos σύνδεσης)

Η ανάθεση των παραπάνω συναρτήσεων ως callback functions γίνεται ως εξής:

```
self.client.on_message = self.on_message_callback
self.client.on_connect = self.on_connect_callback
```

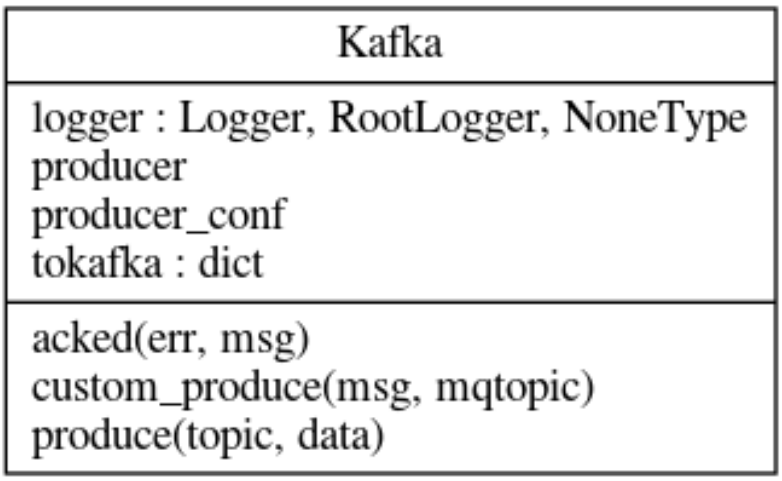
Τέλος, για την σύνδεση εκτελείται

```
self.client.connect(hostname,port)
```

όπου hostname η διεύθυνση του broker και port η θύρα του broker. Αυτό μας δίνει ένα αντικείμενο συνδεδεμένο στον broker που μας επιτρέπει την κατανάλωση και την παραγωγή μηνυμάτων από και σε αυτόν.

Υπάρχουν 3 τρόποι για την εκκίνηση κατανάλωσης μηνυμάτων από τα εγγεγραμμένα topics:

- client.loop(t): Μπλοκάρει το πρόγραμμα και καταναλώνει μηνύματα για t ms.
- client.loop_start(): Δημιουργεί νέο νήμα που καλεί το loop ανά τακτά χρονικά διαστήματα.
- client.loop_forever: Μπλοκάρει το πρόγραμμα και καταναλώνει μηνύματα επ' αορίστων.



Σχήμα 4.6: *Kafka client class*

Στην πλατφόρμα γίνεται χρήση της τρίτης μεθόδου καθώς το κομμάτι αυτό ασχολείται μόνο με την κατανάλωση μηνυμάτων και δεν υπάρχει ανάγκη εκτέλεσης άλλων εντολών από τον client κατά τη διάρκεια αυτή. Η μέθοδος αυτή διαχειρίζεται αυτόματα της επανασυνδέσεις σε περίπτωση κάποιου σφάλματος.

Εισάγοντας όλα τα παραπάνω στην κλάση mqtt, και δημιουργώντας την μέθοδο:

```
def consumer(self):
    self.client.loop_forever()
```

Μπορούμε να εκτελέσουμε το client που θα χρησιμοποιήσουμε στην πλατφόρμα με τον εξής τρόπο:

```
mqtt_client = mqtt()
mqtt_client.topics = [...] # topics to subscribe
mqtt_client.consumer()
```

Αυτό δημιουργεί ένα instance της κλάσης mqtt, θέτει τα topics που θέλουμε να εγγραφεί και ξεκινάει την κατανάλωση σε αυτά εκτελώντας τις callback functions που έχουν οριστεί στην αρχικοποίηση.

4.2.2 Kafka handler

Στην ενότητα αυτή παρουσιάζεται η υλοποίηση του εξυπηρετητή kafka που θα διαχειρίζεται την προώθηση δεδομένων στον kafka broker. Για την σύνδεση χρησιμοποιήθηκε η βιβλιοθήκη confluent-python[15].

Δημιουργήθηκε η κλάση Kafka με τη μορφή που παρουσιάζεται στο σχήμα 4.6.

Οι εισαγωγές τρίτων βιβλιοθηκών είναι οι εξής:

```
import yaml
import json
from confluent_kafka import Producer
```

Η διαδικασία για την δημιουργία ενός kafka Producer μέσα στην κλάση κατά την αρχικοποίησή της είναι:

```
self.producer = self.confluent_kafka.Producer(settings)
```

Που δημιουργεί ένα instance ασύγχρονου kafka [producer.Η παράμετρος settings είναι ένα python dictionary με τις ρυθμίσεις για την σύνδεση στον broker και περιγράφονται από το librdkafka[16] πάνω στο οποίο βασίζεται η βιβλιοθήκη. Η απαραίτητη ρύθμιση μεταξύ άλλων είναι η διεύθυνση του broker

```
self.settings['bootstrap.servers'] = 'broker.address:9092'
```

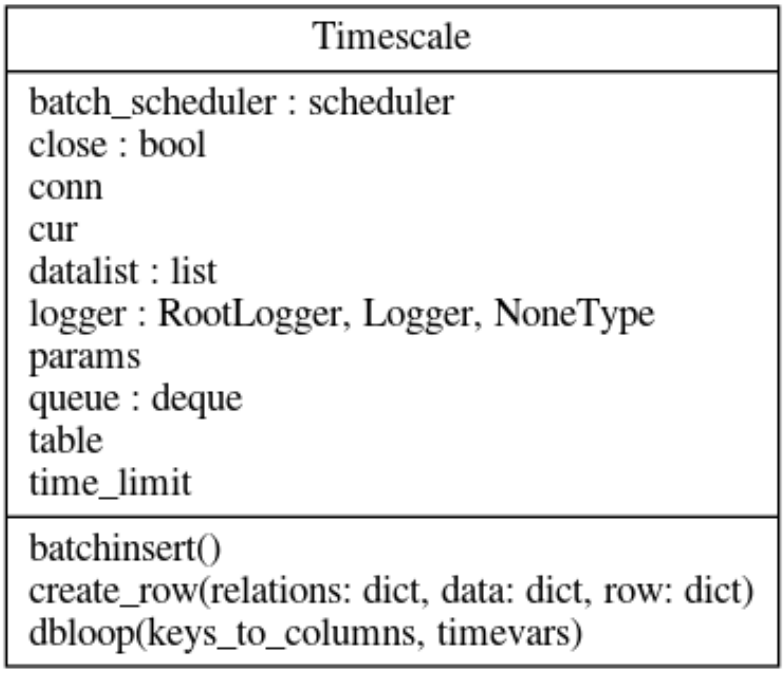
Η κλάση kafka περιέχει την μέθοδο produce που στέλνει ένα μήνυμα στον kafka. Η αποστολή γίνεται ασύγχρονα, εκμεταλεύοντας την ιδιότητα του kafka να μαζεύει πολλά μηνύματα μαζί και τα στέλνει στον broker ανά τακτά χρονικά διαστήματα. Αυτό γίνεται μέσω ουράς οπότε και τα δεδομένα διατηρούν την χρονική τους προτεραιότητα. Δίνεται ως παράμετρος επίσης callback function που εκτελείται μόλις εισαχθεί ένα μήνυμα στην ουρά, χρήσιμο για την εμφάνιση τυχόν σφαλμάτων στην αποστολή. Για την εκτέλεση της είναι αναγκαίο να γίνει ενημέρωση από τον producer για οποιοδήποτε συμβάν , συνήθως αναφορές παράδοσης προηγούμενων μηνυμάτων, με την χρήση του poll.

```
def produce(self,topic,data)
    self.producer.produce(topic,data,callback=acked)
    self.producer.poll(0) #0 seconds block
```

Καθώς η λειτουργία της πλατφόρμας προϋποθέτει την αποστολή των δεδομένων σε διαφορετικά topics, ολόκληρα αλλά και επιλεκτικά, δημιουργήθηκε συνάρτηση διαχωρισμού και προώθησης δεδομένων στο κατάλληλο topic.

Η μέθοδος custom_produce της κλάσης, θα δέχεται τα δεδομένα της πλατφόρμας και θα τα προωθεί αναλόγως στα κατάλληλα kafka topics. Θα περιγραφεί παρακάτω, στο κομμάτι του connector. Θα ορίζεται από τον connector dictionary "tokafka" με τις σχέσεις json-keykafka-topic για κάθε mqtt topic προέλευσης. Για παράδειγμα το tokafka[mtopic][ktopic][json-key] = value περιγράφει πώς η τιμή που ανήκει στο json-key του μηνύματος που προήλθε από το mqtt topic:mtopic πρέπει να προωθηθεί στο kafka topic ktopic.

```
def custom_produce(self,msg,mqtt_topic):
    for kafka_topic in tokafka[mqtt_topic]:
        //create json output
        out = json.dumps(
            {
                json_key: msg.get(json_key,None)
                for json_key in tokafka[mqtt_topic][kafka_topic]
            }
            self.produce(kafka_topic,out)
```



Σχήμα 4.7: *Database client class*

4.2.3 Database Handler

Στην ενότητα αυτή παρουσιάζεται η υλοποίηση του εξυπηρετητή για την εισαγωγή δεδομένων στην βάση δεδομένων. Για την σύνδεση στην βάση χρησιμοποιήθηκε η βιβλιοθήκη psycopg2[17]. Δημιουργήθηκε η κλάση database_client που έχει την μορφή που παρουσιάζεται στο σχήμα 4.7.

Οι εισαγωγές τρίτων βιβλιοθηκών είναι οι εξής:

```
import csv
import json
import sched
from collections import deque
from io import StringIO
import psycopg2
import yaml
```

Η εισαγωγή δεδομένων στην βάση ακολουθεί τις σχέσεις json_key - db_column_name που δίνονται από το αρχείο ρυθμίσεων. Αυτό σημαίνει ότι υπάρχει dictionary keys_to_columns όπου περιγράφει σε ποια στήλη του πίνακα θα μπει κάθε τιμή του μηνύματος. Για παράδειγμα η τιμή keys_to_columns["key1"] = "column_1" περιγράφει πώς η στην στήλη "column1" του πίνακα της βάσης δεδομένων θα εισάγονται οι τιμές που ανήκουν στο κλειδί key1 του μηνύματος ({"key1":"value").

Για την καλύτερη ομαδοποίηση των δεδομένων, δημιουργείται από τον connector ξεχωριστό νήμα για τις διεργασίες του database handler για κάθε πίνακα της βάσης που θα διαχειρίζεται η πλατφόρμα. Αυτό διευκολύνει την δημιουργία συνόλου δεδομένων με κριτήρια που αφορούν τον κάθε πίνακα και την προετοιμασία

τους.

Η αρχικοποίηση της κλάσης δημιουργεί instance της κλάσης psycopg2 που μας επιτρέπει την εκτέλεση ερωτημάτων προς την βάση καθώς και τις απαραίτητες μεταβλητές που θα χρησιμοποιεί η διεργασία όπως η λίστα απο την οποία θα εισάγονται δεδομένα στον πίνακα και το dictionary το οποίο θα περιέχει τις σχέσεις που θα καθορίζουν τον προορισμό κάθε τιμής στον πίνακα.

```
def __init__(self,table,config):
    self.conn = psycopg2.connect(**config)
    self.time_limit = config['time_limit']
    self.conn.autocommit = True
    self.cursor = self.conn.cursor()
    self.datalist = []
    self.table = table
    self.relations = None #set in connector
```

Η παράμετρος config περιέχει τις ρυθμίσεις για την σύνδεση στη βάση όπως η διεύθυνση, το όνομα της βάσης, στοιχεία αυθεντικοποίησης και άλλα που θα χρησιμοποιήσει η κλάση όπως ο χρόνος επανεκτέλεσης της εισαγωγής.

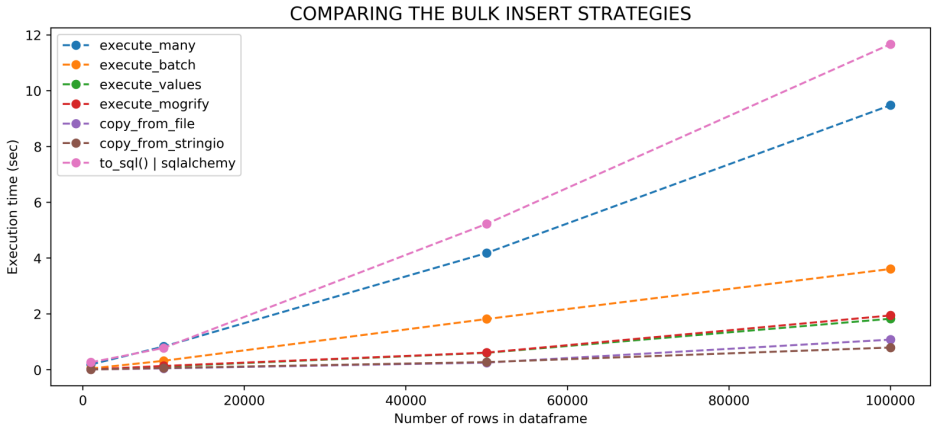
Τα ερωτήματα στη βάση γίνονται με τη χρήση της κλάσης cursor του conn π.χ.:

```
cursor = conn.cursor()
#e.g.
cursor.execute("SELECT * FROM TABLE")
```

Για να έχει η πλατφόρμα την δυνατότητα να εισάγει πολλά δεδομένα χωρίς χρονικές καθυστερήσεις θα πρέπει να υπάρχει πολιτική ομαδοποίησης και η ομαδική εισαγωγή αυτών στη βάση. Όπως φαίνεται στην εικόνα 4.1, η μέθοδος COPY_FROM είναι η καταλληλότερη για την επίτευξη αυτού του σκοπού. Για την χρήση αυτής της μεθόδου θα πρέπει να γίνεται ομαδοποίηση των δεδομένων ανάλογα τον πίνακα που θα εισάγονται. Η COPY_FROM χρησιμοποιεί δομές δεδομένων όπως dataframe ή αρχεία όπως csv. Στην υλοποίηση της πλατφόρμας γίνεται χρήση προσωρινού εικονικού αρχείου csv, από το οποίο λαμβάνει τα δεδομένα και τα εισάγει στον κατάλληλο πίνακα της βάσης. Για την υλοποίηση της θα χρησιμοποιήσουμε την μέθοδο copy_from του cursor.

Για την μεταφορά των μηνυμάτων από τον mqtt consumer χρησιμοποιείται ουρά τύπου deque (FIFO). Ο mqtt client θα εισάγει απευθείας το μήνυμα στην κατάλληλη ουρά για να το μορφοποιήσει ο database client. Αυτή η διαδικασία γίνεται επ' αορίστων όσο η ουρά περιέχει αντικείμενα, και μετά την κατάλληλη μορφοποίησή τους βάση των σχέσεων που περιγράφονται από το keys_to_columns εισάγονται στον πίνακα datalist. Η ροή των δεδομένων παρουσιάζεται στο σχήμα 4.8

```
def dbloop():
    while True:
        while queue:
            #pop first item
```



Πηγή: [18]

Εικόνα 4.1: *Σύγκριση μεθόδων μαζικής εισαγωγής*

```
            json_dict = self.queue.popleft()

            #create row dict based on given relations
            row = {}
            self.create_row(self.relations,json_dict,row)
            self.datalist.append(row)
        self.batch_scheduler.run(False) #no data means no need to insert
```

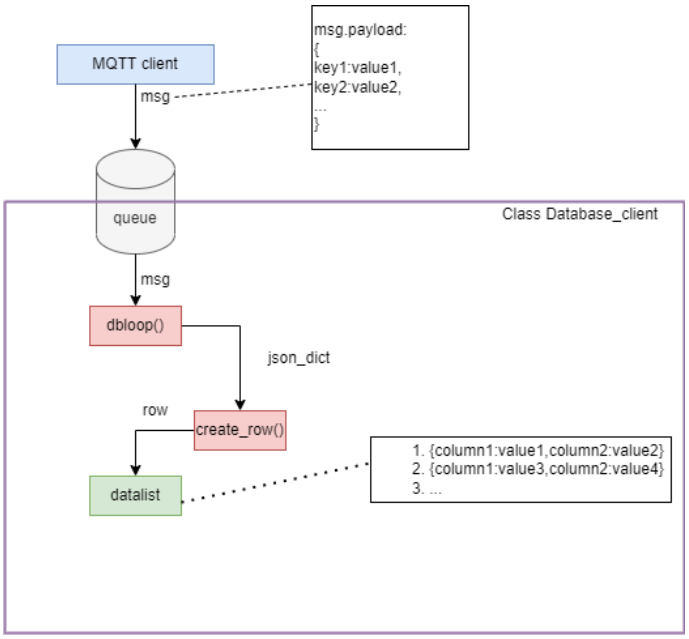
Η συνάρτηση create_row δημιουργεί το τελικό dictionary με τις τιμές του μηνύματος στο κατάλληλο κλειδί (row[column_name] = "value"). Λειτουργεί αναδρομικά καθώς η πλατφόρμα πρέπει να προσφέρει την δυνατότητα εισαγωγής τιμών που βρίσκονται σε φωλιασμένα json μέσα στο μήνυμα.

```
def create_row(self,relations,data,row)

    for key, value  in relations.items():
        #if json key in relations
        if key in data:
            cur = data[key]
            #check if nested json
            if isinstance(cur,dict):
                create_row(value,cur,row)
    else:
        row[value] = cur
```

Τα δεδομένα θα έρχονται μορφοποιημένα σε λίστα από dictionaries, όπου το κάθε dictionary θα περιέχει ζεύγη με κλειδί το όνομα της στήλης του πίνακα της βάσης που θα εισαχθεί και τιμή την τιμή που θα εισαχθεί σε αυτή την στήλη.

```
#list of data dictionaries
datalist = [dict1,dict2,...]
datalist[0]["column_name"] = "value_to_be_inserted"
```

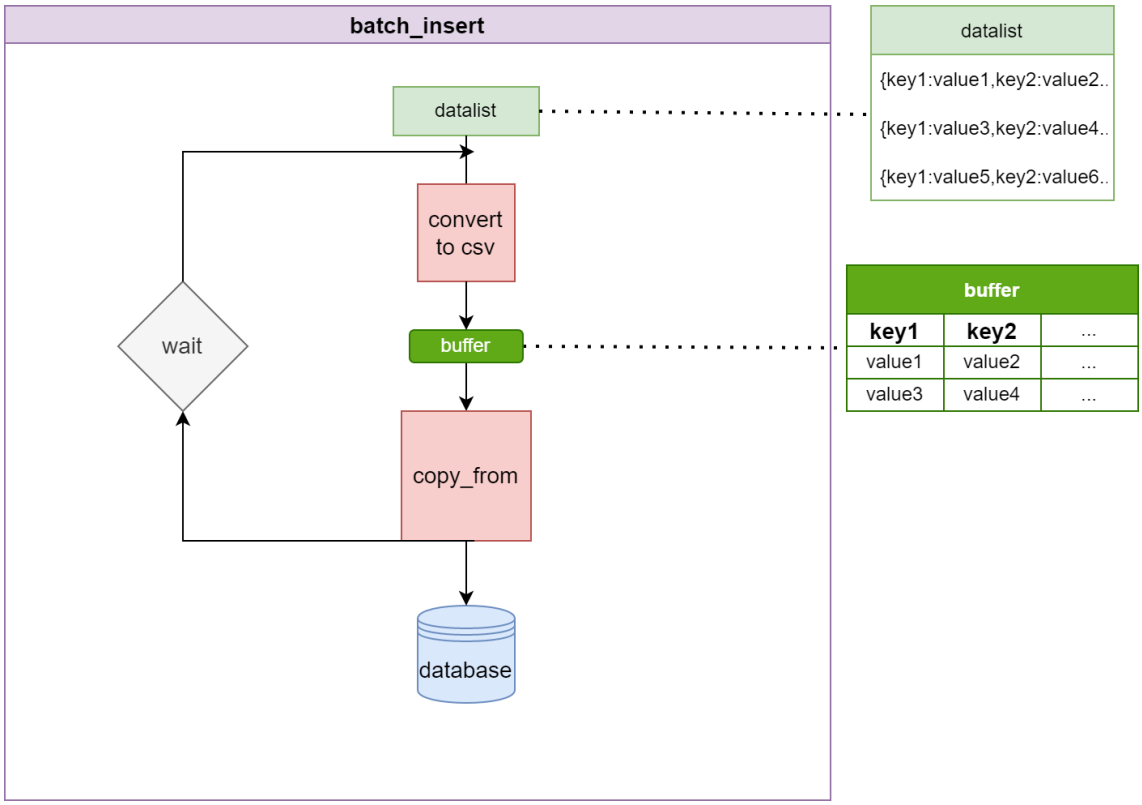


Σχήμα 4.8: *Ροή δεδομένων*

Αυτό μας δίνει την δυνατότητα με τη χρήση της βιβλιοθήκης csv της python να μετατρέψουμε τα dictionaries σε csv με τίτλους στηλών τα κλειδιά τους. Το εικονικό csv δημιουργείται με την χρήση της βιβλιοθήκης StringIO που δημιουργεί εικονικά αρχεία. Τέλος με το copy_from εισάγουμε όλα τα δεδομένα από την datalist στον πίνακα της βάσης.

```
def batchinsert(self):
    if self.datalist:
        #virtual file
        buffer = StringIO()
        #table columns = keys
        keys = self.datalist[0].keys()
        #write data to csv
        dictwrite = csv.DictWriter(buffer, keys)
        dictwrite.writerows(datalist)
        buffer.seek(0)
        #insert data
        self.cursor.copy_from(buffer,self.table,columns=list(keys))
```

Για να υπάρχει συνεχής εισαγωγή δεδομένων, η batchinsert πρέπει να τρέχει ανά τακτά χρονικά διαστήματα και να εισάγει όσα δεδομένα είναι έτοιμα εκείνη τη χρονική στιγμή. Για την επίτευξη αυτού του σκοπού χρησιμοποιήθηκε η βιβλιοθήκη sched της python που υλοποιεί έναν event scheduler ο οποίος εκτελεί καθορισμένες εργασίες σε συγκεκριμένες χρονικές στιγμές. Υλοποιείται με την προσθήκη του στην αρχή της διαδικασίας εισαγωγής για τον προγραμματισμό της επανεκτέλεσής της μετά από το χρονικό διάστημα που θα ορίσουμε εμείς. Για να ξεκινήσει η όλη διαδικασία εκτελούμε απλά μια φορά την συνάρτηση batchinsert() που θα ξεκινήσει



Σχήμα 4.9: *Ομαδοποιημένη εισαγωγή*

το μέτρημα. Η διαδικασία της ομαδικής εισαγωγής δεδομένων παρουσιάζεται στο σχήμα 4.9

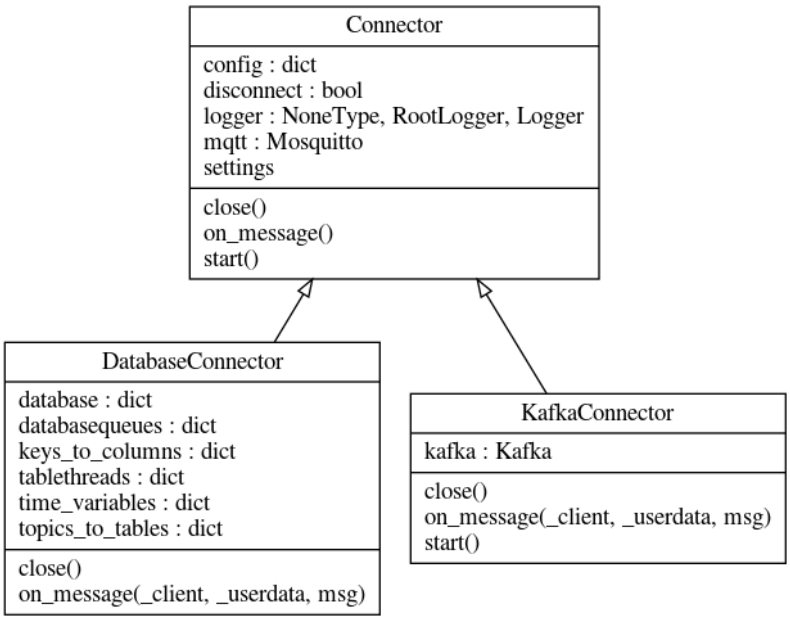
```
def __init__():
    #sched init
    self.scheduler = sched.scheduler(time.time,time.sleep)
    ...
def batchinsert():
    #run again after t seconds, priority 1
    self.scheduler.enter(t,1,batchinsert)
    ...
def dbloop():
    self.batchinsert()
    ...
```

4.2.4 Connector

Στην ενότητα αυτή παρουσιάζεται η υλοποίηση του connector που αποτελεί το κύριο μέρος της πλατφόρμας καθώς διαχειρίζεται όλα τα προηγούμενα κομμάτια. Φορτώνει τα αρχεία ρυθμίσεων και βάση αυτών δημιουργεί τον mqtt client με την χρήση του αντικειμένου mqtt handler και ορίζει τα σωστά callback functions για κάθε προορισμό δεδομένων. Αποτελείται από την βασική κλάση connector και τις KafkaConnector,DatabaseConnector, που με τη χρήση της κληρονομικότητας επε-

κτείνουν τις λειτουργίες τις βασικής κλάσης. Η κάθε μία από τις δύο αυτές κλάσεις περιέχει λειτουργίες για την προώθηση στον kafka, και την εισαγωγή στην βάση.

Στην φιγούρα 4.10 παρουσιάζονται τα διαγράμματα των κλάσεων.



Σχήμα 4.10: *Διάγραμμα κλάσεων connector*

Οι εισαγωγές τρίτων βιβλιοθηκών είναι οι εξής:

```
import threading

import paho.mqtt.client as mqtt
import yaml

from platform kafka_handler, mqtt_handler, database_handler
```

Connector Base Class

Η κλάση Connector αποτελεί κοινό σημείο και των δύο υποκλάσεων και περιέχει τις κοινές λειτουργίες που εκτελούνται για τον κάθε σκοπό. Η αρχικοποίησης της εκτελεί την εκκίνηση του mqtt client. Οι κοινές μέθοδοί είναι η εκκίνηση του βρόγχου κατανάλωσης μηνυμάτων.

```
def __init__(self,config):

    self.mqtt_client = mqtt_handler.Mosquitto()
    #reference for custom callback function
    self.mqtt_client.on_message = self.on_message

def start(self):
    try:
        self.mqtt.consumer()
```

```
    except KeyboardInterrupt:
        self.close()

def on_message():
    pass

def close(self):
    pass
```

KafkaConnector

Η κλάση αυτή δημιουργεί το kafka client με τη χρήση του kafka handler και φορτώνει από τις ρυθμίσεις τις σχέσεις προώθησης mqtt topics - kafka topics.

```
def __init__(self):
    self.tokafka = self.config['tokafka']
    self.kafka_client = kafka_handler.Kafka()
```

Ορίζει την callback function που θα καλεί την συνάρτηση του kafka handler για την διαχείριση των μηνυμάτων.

```
def on_message(self,client,userdata,msg):
    message = json.loads( msg.payload.decode("utf-8","ignore"))

    self.kafka_client.custom_produce(message,msg.topic)
```

Για την εκκίνηση της λειτουργίας χρησιμοποιεί την συνάρτηση εκκίνησης του mqtt handler που αρχίζει την κατανάλωση μηνυμάτων. Χρησιμοποιείται διαχείρισή σφαλμάτων try..except για την ομαλή αποσύνδεση και κλείσιμο της πλατφόρμας στην περίπτωση οποιουδήποτε σφάλματος ή τερματισμού μέσω CTRL+C.

```
def start(self):
    try:
        self.mqtt_client.consumer()
    except:
        self.close()
```

Η μέθοδος close() αποσυνδέει τον mqtt client.

```
def close(self):
    self.mqtt_client.disconnect()
```

Έτσι μπορούμε να αρχίσουμε την λειτουργία προώθησης μηνυμάτων από mqtt topics προς kafka topics με τον εξής τρόπο:

```
KafkaConnector().start()
```

DatabaseConnector

Η κλάση αυτή δημιουργεί τα απαραίτητα νήματα για την εισαγωγή στους πίνακες της βάσης με τη χρήση του databse handler και φορτώνει τις σχετικές ρυθμίσεις για

την εισαγωγή των μηνυμάτων στην κατάλληλη θέση.

Δημιουργούνται από το αρχείο ρυθμίσεων τα dictionaries που θα περιέχουν τις συσχετίσεις mqtt topics-πίνακες-στήλες.

```
def __init__(self):
    super().init() #inheritance

    config = self.config["todb"]

    self.topics_to_tables = {}
    # 1 topic to many tables : dict['topic'] = ['table1','table2',..]

    self.keys_to_columns = {}
    # dict['table']['json_key'] = 'column_name'

    self.tables=set()

    for topic in config:
        self.topics_to_tables[topic] = []
        self.mqtt.topics.add((topic,qos)) #add mqtt topic subscription

        for table in config[topic]:
            self.tables.add(table)
            self.topics_to_tables[topic].append(table)

            self.keys_to_columns[table] = config[topic][table]['relations']
```

Κάθε νήμα εκτελεί ξεχωριστή διεργασία και περιέχει ξεχωριστό instance του database handler και κάθε ένα εισάγεται σε λίστα για την μετέπειτα χρήση του. Η λίστα με τα αντικείμενα database_handler είναι της μορφής dictionary με κλειδιά το όνομα του πίνακα για την γρήγορη αναφορά σε αυτά.

```
    self.database = {} #list of database_handler instances
    for table in self.tables:
        self.database[table] = database_handler.Timescale(table)
        self.database[table].relations = self.keys_to_columns

        threading.Thread(target = self.database[table].dbloop,
        args=(self.keys_to_columns[table])).start()
        #start new thread
```

Ορίζεται η callback function που θα κάνει τις απαραίτητες εισαγωγές στις κατάλληλες ουρές:

```
    def on_message(self,_client,_userdata,msg):
       for topic,tables in self.topics_to_tables.items():
           if mqtt.topic_matches_sub(topic,msg.topic):#wildcard subscription
               for table in tables:
                   self.database[table].queue.append(msg)
```

Γίνεται πέρασμα σε όλα τα ζεύγη topic-πίνακες καθώς χρησιμοποιείται η βοηθητική συνάρτηση topic_matches_sub που αντιστοιχεί το όνομα ενός topic με κάποιον κανόνα εγγραφής που μπορεί να περιέχει ειδικούς χαρακτήρες (#,+).

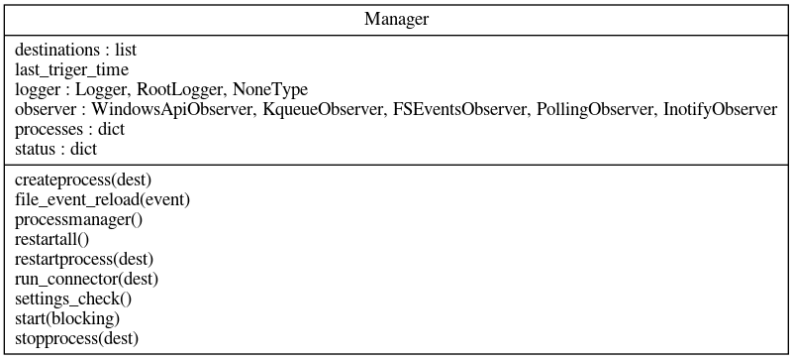
Η συνάρτηση που διαχειρίζεται το κλείσιμο πρέπει να τερματίζει όλες τις συνδέσεις που έχουν δημιουργηθεί από τα νήματα των βάσεων και να περιμένει τα νήματα να τερματίσουν πρώτου τερματίσει τον mqtt client:

```
    def close(self):
        for table in self.database:
            self.database[table].conn.close() #end database connection
            self.database[table].close = True
            self.tablethreads[table].join() #wait thread to terminate

        self.mqtt.disconnect() #terminate mqtt consumer
```

4.2.5 Manager

Στην ενότητα αυτή παρουσιάζεται η υλοποίηση του manager που δημιουργεί και διαχειρίζεται τις διεργασίες των connector και συνιστά τον κόμβο έναρξης της πλατφόρμας. Στην φιγούρα 4.11 παρουσιάζονται οι μέθοδοι της κλάσης manager.



Σχήμα 4.11: *Διάγραμμα κλάσεων manager*

Οι εισαγωγές τρίτων βιβλιοθηκών είναι οι εξής:

```
import sys
import time
from multiprocessing import Process
from watchdog.observers import Observer
from watchdog.events import PatternMatchingEventHandler
import yaml
import threading
from platform import connector
```

Μία υπευθυνότητα του manager είναι να ελέγχει τα αρχεία ρυθμίσεων ως προς την συντακτική ορθότητα και να παρακολουθεί τυχόν αλλαγές. Στην περίπτωση που πραγματοποιηθεί αλλαγή στα αρχεία ρυθμίσεων, επανεκκινεί την πλατφόρμα για να

φορτώσει εκ νέου τις νέες ρυθμίσεις. Ο έλεγχος για την τροποποίηση αρχείων γίνεται από την βιβλιοθήκη watchdog [19].

Η αρχικοποίηση της κλάσης manager κάνει τις απαραίτητες αρχικοποιήσεις για την παρακολούθηση των αρχείων που απαιτεί το watchdog και δημιουργεί τις απαραίτητες μεταβλητές. Δέχεται ως παράμετρο μια λίστα που ορίζει ποιές διεργασίες θα δημιουργηθούν (kafka Connector,database Connector) και έχει ως προκαθορισμένη την λίστα [kafka,database] για την δημιουργία και των δύο connector.

```
def __init__(self,destinations=["database","kafka"])
    #watchdog init
    handler = PatternMatchingEventHandler(
        ["*"],ignore_directories=True,case_sensitive=True
    )
    handler.on_modified = self.file_event_reload #watchdog callback function
    self.observer = Observer()
    self.observer.schedule(handler,path,recursive=False)

    #processes list
    self.processes = {}
    self.destinations = destinations
```

Η κύρια μέθοδος του manager είναι η proccessmanager, η οποία δημιουργεί τις διεργασίες και ελέγχει συνέχεια την κατάστασή τους. Στην περίπτωση που κάποια διεργασία βρεθεί τερματισμένη, αναλαμβάνει να την επανεκκινήσει.

```
    def processmanager(self):
        if len(self.processes)==0:
            self.createprocess(dest)
        while len(self.processes):
            for destination,process in self.processes.items():
                if process.exitcode is None and not proc.is_alive():
                    self.restartprocess(dest)
```

Ο manager περιέχει μεθόδους για την δημιουργία, τερματισμό και επανεκκίνησή των διεργασιών και είναι οι createprocess,stopprocess,restartprocess,restartall. Για την δημιουργία των διεργασιών, χρησιμοποιείται η μέθοδος run_connector που αρχικοποιεί και εκκινεί τις κλάσεις connector.

```
    def run_connector(self,destination):
        if dest=='kafka':
            connector.KafkaConnector().start()
        elif dest=='database':
            connector.DatabaseConnector().start()
```

```
    def create_process(self,destination):
        self.processes[destination] = Process(
            target=self.run_connector,args=(destination,)
        )
        self.processes[destination].start() #start process
```

```
    def stopprocess(self,destination):
        self.processes[destination].terminate()
```

```
    def restartprocess(self,destination):
        self.stopprocess(destination)
        self.createprocess(destination)
```

```
    def restartall(self):
        for destination in self.processes:
            self.restartprocess(destination)
```

Η callback function του watchdog που είναι υπεύθυνη για την επανεκκίνησή του προγράμματος κατά την αλλαγή των αρχείων ρυθμίσεων καλεί την παραπάνω συνάρτηση restartall για να φορτωθούν όλες οι νέες ρυθμίσεις. Για να γίνει όμως, πρέπει να υπάρξει συντακτικός έλεγχος στα αρχεία για τυχόν λάθη που θα δημιουργήσουν πρόβλημα στην σωστή λειτουργία του προγράμματος. Ο έλεγχος γίνεται κατά την φόρτωση του αρχείου yaml με try...except ελέγχοντας για εξαίρεση.

```
def settings_check(self):
    config_files=['connector.yml','kafka.yml','database.yml','mqtt.yml']
    for filename in config_files:
        try:
            with open(filename,"r") as file
                yaml.safe_load(file)
        except Exception as e:
            return False

    return True
```

```
def file_event_reload(self,event):
    if self.settings_check():
        self.restartall()
```

Η εκκίνηση όλης της πλατφόρμας γίνεται από την μέθοδο start του manager. Δέχεται την μεταβλητή blocking, που ορίζει αν η πλατφόρμα θα εκτελεστεί ως κύριο πρόγραμμα (blocking) η ως ξεχωριστό νήμα (nonblocking).

```
def start(self,blocking=True):
    self.observer.start() #starts watchdog
    if blocking:
        try:
            self.processmanager()
        except:
            sys.exit()
    else:
        threading.Thread(target=self.processmanager).start()
```

4.2.6 Αρχεία ρυθμίσεων

Τα αρχεία ρυθμίσεων που χρησιμοποιούνται από όλα τα κομμάτια της πλατφόρμας χρησιμοποιούν την μορφοποίηση yaml [20] για ευκολία ανάγνωσης και τροποποίησης. Η μορφοποίηση yaml μας επιτρέπει να έχουμε ζεύγη κλειδιών τιμών και πιο περίπλοκες δομές όπως φωλιασμένα σύνολα εντός κλειδιών.

Τα αρχεία βρίσκονται εντός φακέλου ονόματος settings και είναι τα εξής:

- mqtt.yml: Ρυθμίσεις σύνδεσης στον mqtt broker.
- database.yml: Ρυθμίσεις σύνδεσης στην βάση δεδομένων.
- kafka.yml: Ρυθμίσεις σύνδεσης στον kafka broker.
- connector.yml: Ρυθμίσεις πλατφόρμας.

mqtt.yml

Η μορφή του αρχείου ρυθμίσεων για την σύνδεση στον mqtt broker είναι η εξής:

```yaml
---
ssl:
  client_id: client-name-prefix
  host: secure.mqtt.hostname
  port: 8883
  username: username
  password: password
  ca_cert: /path/to/ssl/ca.crt
  client_key: /path/to/ssl/client.key
  client_cert: /path/to/ssl/client.crt
plain:
  client_id: client-name-prefix
  host: unsecure.mqtt.hostname
  port: 1883
  username: username
  password: password
```

Τα 2 κύρια υποσύνολα (ssl/plan) είναι οι ρυθμίσεις για τον κάθε τύπο σύνδεσης που μπορεί να επιτευχθεί από τον client, ασφαλής ή απλή. Στην ασφαλή σύνδεση πρέπει να ορίζονται και τα απαραίτητα πιστοποιητικά για την κρυπτογράφηση των δεδομένων.

Οι υπόλοιπες παράμετροι αποτελούνται από διεύθυνση και θύρα mqtt broker, όνομα και κωδικός αυθεντικοποίησης αν χρειάζονται και αν είναι επιθυμητό όνομα για τον client για καλύτερη αναγνώριση από τη μεριά του broker.

kafka.yml

Η μορφή του αρχείου ρυθμίσεων για την σύνδεση στον kafka broker είναι η εξής:

```
---
plain:
  bootstrap.servers: plain.kafka.host:9092
  client.id: client-name-prefix
  linger.ms: 1000
  broker.address.family: v4
  acks: 0

ssl:
  bootstrap.servers: ssl.kafka.host:9092
  client.id: client-name-prefix
  linger.ms: 1000
  broker.address.family: v4
  acks: 0
  security.protocol: ssl
  ssl.ca.location: /path/to/ssl/ca.crt
  ssl.certificate.location: /path/to/ssl/server.crt
  ssl.key.location: /path/to/ssl/server.key
  ssl.key.password: key password (ifexists)
```

Τα σημαντικά πεδία είναι το bootstrap.servers που είναι η διεύθυνση και η θύρα του kafka broker, το linger ms που αποτελεί τον ρυθμό δημοσιοποίησης όλων των μηνυμάτων που βρίσκονται στην ουρά και το acks που αποτελεί το επίπεδο αναγνώρισης που απαιτεί ο client από τον/τους broker/s, όπως περιγράφηκε στο 2ο κεφάλαιο.

Πολύ σημαντικό είναι επίσης στο μέρος της ασφαλής σύνδεσης ssl, τα πεδία που ορίζουν την τοποθεσία των πιστοποιητικών και κλειδιών για την κρυπτογράφηση της σύνδεσης.

Οι ρυθμίσεις που επιτρέπονται περιγράφονται από το librdkafka[16] στο οποίο βασίζεται και η βιβλιοθήκη που χρησιμοποιείται για τον kafka client της πλατφόρμας. Μπορούν να προστεθούν επιπλέον ρυθμίσεις για τον kafka producer ανάλογα τις ανάγκες που προκύπτουν.

database.yml

Η μορφή του αρχείου ρυθμίσεων για την σύνδεση στον kafka broker είναι η εξής:

```yaml
---
ssl:
  host: ssl.host.name
  database: databasename
  user: username
  password: password
  sslmode: verify-full
  sslrootcert: /path/to/ssl/ca.crt
  sslecert: /path/to/ssl/client.crt
  sslkey: /path/to/ssl/client.key
plain:
  host: plain.host.name
  database: databasename
  user: username
  password: password

custom:
  batchtime: 10
```

Όπως και στα παραπάνω παραδείγματα, υπάρχουν πεδία για την σύνδεση και την αυθεντικοποίηση του χρήστη και στην περίπτωση της ασφαλούς σύνδεσης τα αρχεία πιστοποίησης. Σημαντική διαφορά αποτελεί το όνομα της βάσης στην οποία θα συνδεθεί η πλατφόρμα, και στο custom μέρος το πεδίο batchtime, που αποτελεί το χρονικό διάστημα ανάμεσα σε κάθε μαζική εισαγωγή δεδομένων στην βάση σε δευτερόλεπτα.

connector.yml

Οι ρυθμίσεις που περιέχονται στο αρχείο αυτό αποτελούν σημείο αναφοράς για τη βασική λειτουργία της πλατφόρμας. Αποτελείται από 3 μέρη:

- settings
- tokafka
- todb

Το settings περιέχει boolean τιμές για το αν η κάθε σύνδεση θα είναι κρυπτογραφημένη ή όχι.

```yaml
---
settings:
  kafkatls: true/false
  mqtttls: true/false
  dbtls: true/false
```

Το μέρος tokafka περιγράφει για κάθε mqtt topic, σε ποια kafka topics προωθούνται τα επιλεγμένα κλειδιά των μηνυμάτων.

```yaml
---
tokafka:
  /mqtt/topic/name/1:
    kafka.topic.name.1:
      - json_key1
      - kson_key2
    kafka.topic.name.2:
      - json_key3
      - kson_key4
  /mqtt/topic/name/2:
    kafka.topic.name.3:
      - json_key1
      - kson_key2
    kafka.topic.name.4:
      - json_key3
      - kson_key4
```

Το μέρος todb περιγράφει για κάθε mqtt topic τους πίνακες που θα λαμβάνουν δεδομένα από το μήνυμα και τις σχέσεις κλειδιών - στηλών.

```yaml
---
todb:
  /mqtt/topic/name/1:
    table1:
      relations:
        subjson1:
          json_key1:column_name1
        json_key2:column_name2

    table2:
      relations:
        subjson1:
          json_key1: column_name1
        subjson2:
          json_key2: column_name2
        json_key3: column_name3

  /mqtt/topic/name/2:
    table3:
      relations:
        json_key1: column_name1
        json_key2: column_name2
```

Είναι πολύ σημαντικό σε όλα τα αρχεία να διατηρούνται οι εσοχές καθώς έτσι ορίζεται η ιεραρχία των σχέσεων στην μορφή yaml.

4.3 Σχεδιαστικές επιλογές

Για την υλοποίηση της πλατφόρμας έγινε δημιουργία και χρήση κλάσεων για τα πλεονεκτήματα που συνεισφέρουν στο έργο. Προσφέρουν οργάνωση στον κώδικα με την διατήρηση των σχετικών δεδομένων και μεθόδων σε ένα σημείο συνεισφέροντας θετικά και στην αναγνωσιμότητα του με πιο καθαρή δομή. Η επαναχρησιμοποίηση κώδικα γίνεται πιο αποδοτικά καθώς κάθε κλάση προσφέρει ένα μεγάλο εύρος λειτουργιών μέσα από τις μεθόδους της.

Η χρήση αρχείων ρυθμίσεων δίνει στην πλατφόρμα την δυνατότητα να παρέχει σταθερές λειτουργίας ανεξάρτητα της κατάστασης της πλατφόρμας σε συνεχή λειτουργία (π.χ. ανάμεσα σε επανεκκινήσεις). Μέσω αυτών γίνεται αποφυγή της ανάγκης αλλαγής του κώδικα για κάθε διαφορετική περίσταση προσφέροντας ευκολία στην λειτουργία της. Η δομή των αρχείων ρυθμίσεων αντικατοπτρίζει την λειτουργία της πλατφόρμας σε κάθε στάδιο και συνεισφέρει θετικά στην αλληλεπίδραση με τον χρήστη.

Για την πολυεπεξεργαστική επιλογή της πλατφόρμας έπαιξε μεγάλο ρόλο η ανάγκη για ταχύτητα. Έγιναν δύο επιλογές σε διαφορετικά σημεία του κώδικα, η παραλληλία με νήματα και η παραλληλία με διαφορετικές διεργασίες. Στα παράλληλα νήματα, το κάθε νήμα αναφέρεται στο ίδιο υποσύνολο μνήμης με την διεργασία που το δημιούργησε με αποτέλεσμα να μοιράζεται τους ίδιους πόρους με όλα τα υπόλοιπα νήματα εντός της ίδιας διεργασίας. Για την ασφάλεια των δεδομένων η Python ενσωματώνει μηχανισμό ασφαλείας για τα νήματα που τα αποτρέπει ξεχωριστά νήματα να κατέχουν τον διερμηνέα ταυτόχρονα με αποτέλεσμα να μην υπάρχει παράλληλη επεξεργασία στα ίδια δεδομένα.

Κατά την εισαγωγή στην βάση δεδομένων δεν παρατηρήθηκε προβληματική λειτουργία με τη χρήση νημάτων για κάθε ξεχωριστό πίνακα. Όταν γίνεται παράλληλη εισαγωγή στην βάση με την προώθηση στον kafka παρατηρείται καθυστέρηση επεξεργασίας δεδομένων καθώς το πακέτο που λαμβάνεται από τον MQTT broker βρίσκεται σε ένα σημείο στην μνήμη με αποτέλεσμα να υπάρχει ¨σύγκρουση¨ από τα διαφορετικά νήματα για την βάση με αυτά για τον kafka για την πρόσβαση σε αυτό το σημείο. Σε αυτό το πρόβλημα δίνουν λύση οι παράλληλες διεργασίες.

Στις παράλληλες διεργασίες, κάθε διεργασία κατέχει ξεχωριστό υποσύνολο στην μνήμη και αποτελεί πρακτικά ξεχωριστό πρόγραμμα με τη διαφορά πως υπάρχει δυνατότητα επικοινωνίας διεργασιών με τη χρήση ειδικής ουράς. Η χρήση δύο διαφορετικών διεργασιών για την βάση και για τον kafka προσφέρει την αναμενόμενη ταχύτητα στην επεξεργασία και δρομολόγηση των δεδομένων.

Κεφάλαιο 5

Εκτέλεση

Για την παρουσίαση της εκτέλεσης της πλατφόρμας θα χρησιμοποιηθεί python script για την παραγωγή δοκιμαστικών μηνυμάτων στον mqtt broker του εξυπηρετητή. Θα δημιουργηθούν πίνακες στην βάση και θα οριστούν οι σχέσεις μεταξύ πινάκων και πεδίων του μηνύματος και kafka topics και πεδίων στα αρχεία ρυθμίσεων της πλατφόρμας.`

5.1 Δεδομένα

Για τον τύπο των δεδομένων θα δημιουργήσουμε μια εικονική συσκευή έξυπνου ρολογιού που θα παράγει και στέλνει τυχαία δεδομένα στον mqtt broker. Η μορφή του μηνύματος που θα στέλνει η συσκευή θα έχει ως εξής:

- ts: Χρονική σήμανση μηνύματος.
- heartrate: Καρδιακοί παλμοί
- spO2: Επίπεδο κορεσμού οξυγόνου στο αίμα.
- steps: Βήματα που έχουν πραγματοποιηθεί.
- calories: Θερμίδες που έχουν καταναλωθεί.
- gps: Τοποθεσία.
- height: Υψόμετρο.
- id: Χαρακτηριστικό νούμερο συσκευής.

Για να δημιουργήσουμε τέτοιας μορφής δεδομένα θα χρησιμοποιήσουμε συνάρτηση που παράγει ένα κατάλληλο dictionary ως εξής:

```
def create_data(id):
    msg = {}
    msg['ts'] = datetime.today().strftime("%m/%d/%Y %H:%M:%S.%f")
```

```
    msg['heartbeat'] = random.randint(85,95)
    msg['spO2'] = round(random.uniform(0.97,1.00),2)
    msg['steps'] = random.randint(1500,3000)
    msg['calories'] = random.randint(200,500)
    msg['height'] = random.randint(400,500)
    msg['id'] = id
    return msg
```

Η εκτέλεση της συνάρτησης έχει έξοδο που παρουσιάζεται στην εικόνα 5.1

```
>>> msg = create_data()
>>> pprint.pprint(msg)
{'calories': 476,
 'heartbeat': 87,
 'height': 424,
 'spO2': 0.98,
 'steps': 2269,
 'ts': '06/30/2022 23:09:55.197456'}
```

Εικόνα 5.1: *Έξοδος συνάρτησης create_data*

5.2 Εικονικές συσκευές

Για την δημιουργία ρεαλιστικού σεναρίου θα σχεδιαστούν πάνω από ένα εικονικά έξυπνα ρολόγια που θα στέλνουν δεδομένα στον broker. Θα πραγματοποιηθεί με την χρήση διεργασιών και κάθε εικονική συσκευή θα κατέχει την δική της διεργασία. Η δημοσίευση στον broker θα γίνεται με την χρήση της κλάσης που υλοποιήθηκε στα πλαίσια της πλατφόρμας.

Μια εικονική συσκευή θα δημιουργεί τα δεδομένα, θα τα μετατρέπει σε κείμενο και θα τα δημοσιεύει στον broker όσες φορές απαιτηθεί με ορισμένη καθυστέρηση ανά μήνυμα.

```
def virtual_device(topic,times,delay,id):
    mqtt_client = mqtt_handler.Mosquitto()
    for _ in range(times):
        data = create_data(id)
        msg = json.dumps(data)
        mqtt_client.producer(msg,topic)
        time.sleep(delay)
```

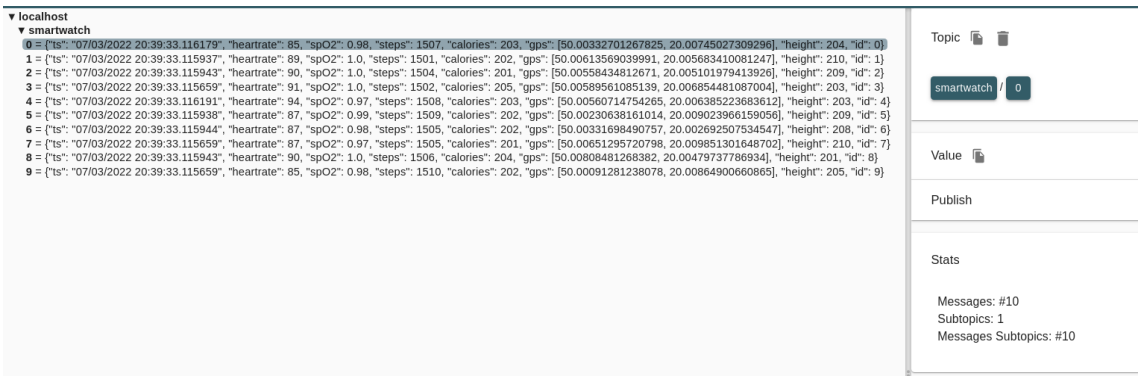
Η δημιουργία της συστάδας συσκευών δημιουργεί ξεχωριστό topic για τη κάθε μία και την εκκινεί σε ξεχωριστό context με την χρήση της βιβλιοθήκης multiprocessing.

```
def virtual_iot(topic_prefix,device_number,msg_number,frequency)
    delay = 1 / frequency
    for id in range(device_number):
        topic = topic_prefix + "/" + id
```

```
        dev = Process(target=virtual_device,args=(topic,msg_number,delay,id))
        dev.start()
```

Για την εμφάνιση των δεδομένων στον broker θα χρησιμοποιηθεί το λογισμικό mqtt-explorer[21] που αποτελεί mqtt client με γραφική διεπαφή χρήστη για την προβολή των δεδομένων στον broker.

Η εκτέλεση της εντολής `virtual_iot(smartwatch,10,10,1)` πρέπει να δημιουργήσει 10 topics (smartwatch/0 - smartwatch/9) και να αποστείλει 10 μηνύματα στο καθένα παράλληλα με συχνότητα 1 μήνυμα το δευτερόλεπτο. Το αποτέλεσμα της εκτέλεσης της παραπάνω εντολής στον broker στο mqtt-explorer εμφανίζεται στην εικόνα 5.2



Εικόνα 5.2: *Στιγμιότυπο δεδομένων* `virtual_iot`

5.3 Βάση δεδομένων

Για την βάση δεδομένων, θα δημιουργηθούν 3 πίνακες που θα περιέχουν τα 3 είδη δεδομένων που παράγει η συσκευή:

- **health**: Τα δεδομένα σχετικά με την υγεία του ατόμου.
- **fitness**: Τα δεδομένα σχετικά με τον αθλητισμό του ατόμου.
- **location**: Τα δεδομένα σχετικά με την τοποθεσία του ατόμου.

Ο πίνακας **health** θα περιέχει τις εξής στήλες:

- **time**: η χρονική στιγμή της μέτρησης
- **heartrate**: Καρδιακοί παλμοί
- **spO2**: Κορεσμός οξυγόνου στο αίμα.
- **id**: Χαρακτηριστικό νούμερο συσκευής.

Ο πίνακας **fitness** θα περιέχει τις εξής στήλες:

- **time**: η χρονική στιγμή της μέτρησης
- **steps**: Βήματα
- **calories**: Θερμίδες που έχουν καταναλωθεί.
- **id**: Χαρακτηριστικό νούμερο συσκευής.

Ο πίνακας **location** θα περιέχει τις εξής στήλες:

- **time**: η χρονική στιγμή της μέτρησης
- **coordinates**: Τοποθεσία
- **height**: Υψόμετρο
- **id**: Χαρακτηριστικό νούμερο συσκευής.

Οι εντολές για την δημιουργία των πινάκων και την μετατροπή τους σε υπερπίνακες είναι οι εξής:

```
01 |   CREATE TABLE health (
02 |    time TIMESTAMPTZ NOT NULL,
03 |    heartrate INT NULL,
04 |    spo2 INT NULL,
05 |    id INT NOT NULL);
06 |  SELECT create_hypertable('health', 'time');
07 |
08 |   CREATE TABLE fitness(
09 |    time TIMESTAMPTZ NOT NULL,
10 |    steps BIGINT NULL,
11 |    calories INT NULL,
12 |    id INT NOT NULL);
13 |   SELECT create_hypertable('fitness', 'time');
14 |
15 |   CREATE TABLE location(
16 |    time TIMESTAMPTZ NOT NULL,
17 |    coordinates DOUBLE PRECISION[2] NULL,
18 |    height INT NULL,
19 |    id INT NOT NULL);
20 |   SELECT create_hypertable('location', 'time');
```

5.4 kafka Topics

Για τον kafka θα χρησιμοποιήσουμε την ίδια μορφή με τους παραπάνω πίνακες της βάσης έτσι θα έχουμε 3 topic με τα εξής δεδομένα το καθένα:

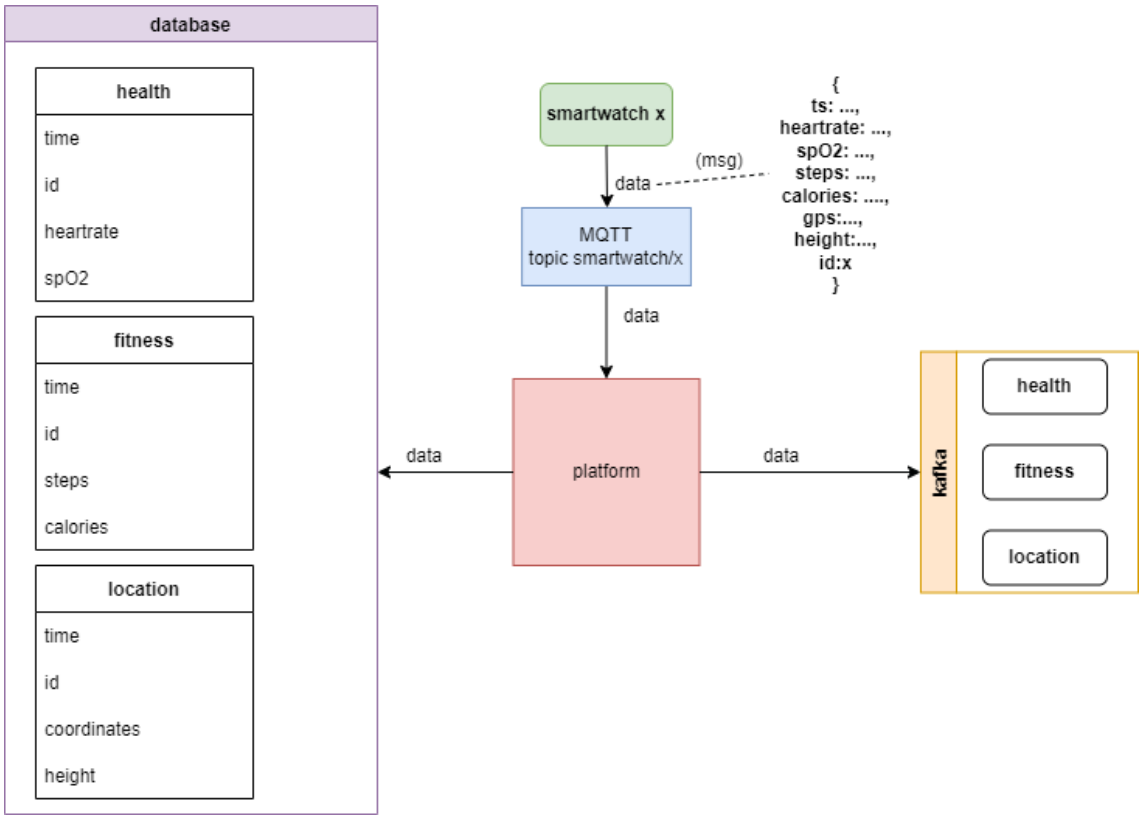
- **health**: ts, heartrate, spO2, id

- **fitness**: ts, steps, calories, id
- **location**: ts, gps, height, id

5.5 Ρυθμίσεις

Το αρχείο connector.yaml θα είναι αυτής της μορφής για την δημιουργία των παραπάνω σχέσεων:

```yaml
---
tokafka:
  smartwatch/#:
    health:
      - ts
      - heartrate
      - spO2
      - id
    fitness:
      - ts
      - steps
      - calories
      - id
    location:
      - ts
      - gps
      - height
      - id
todb:
  smartwatch/#:
    health:
      relations:
        ts: time
        heartrate: heartrate
        spO2: spo2
        id: id
    fitness:
      relations:
        ts: time
        steps: steps
        calories: calories
        id: id
    location:
      relations:
        ts: time
        gps: coordinates
        height: height
        id: id
```



Σχήμα 5.1: *Εικονικό δίκτυο εκτέλεσης*

Το δίκτυο που θα προσομοιωθεί θα έχει την μορφή που παρουσιάζεται στο σχήμα 5.1.

5.6 Εκτέλεση

Με την εκτέλεση της εντολής `python manager.py` ξεκινάει η πλατφόρμα και μπορεί να δεχτεί και να προωθήσει δεδομένα βάση του παραπάνω αρχείου. Εκτελώντας σε διαφορετικό τερματικό python το script που περιγράφηκε παραπάνω για την δημιουργία εικονικών συσκευών με την εξής μορφή : `» virtual_iot("smartwatch", 3, 10, 1)`, 3 ξεχωριστές συσκευές από 10 μηνύματα η κάθε μια παρατηρούνται τα δεδομένα στη βάση στις εικόνες 5.3, 5.4, 5.5 και στον kafka στις εικόνες 5.6, 5.7, 5.8.

Το αποτέλεσμα δείχνει την σωστή λειτουργία της πλατφόρμας καθώς τα δεδομένα βρίσκονται στους σωστούς πίνακες στις σωστές στήλες και στα σωστά kafka topics.

time	heartrate	spo2	id
2022-07-04 17:31:50.386281+00	94	0.98	2
2022-07-04 17:31:50.386844+00	87	0.97	1
2022-07-04 17:31:50.3858+00	95	0.97	0
2022-07-04 17:31:51.386073+00	94	0.98	0
2022-07-04 17:31:49.385236+00	93	1	0
2022-07-04 17:31:51.387562+00	86	0.99	1
2022-07-04 17:31:49.385581+00	91	0.99	1
2022-07-04 17:31:49.385871+00	91	1	2
2022-07-04 17:31:51.386831+00	88	0.97	2
2022-07-04 17:31:54.387091+00	94	1	0
2022-07-04 17:31:56.389553+00	86	0.98	2
2022-07-04 17:31:52.388826+00	87	1	1
2022-07-04 17:31:56.392347+00	87	0.99	1
2022-07-04 17:31:55.387438+00	90	0.99	0
2022-07-04 17:31:52.386352+00	85	1	0
2022-07-04 17:31:57.389793+00	95	0.97	2
2022-07-04 17:31:58.389803+00	94	0.98	0
2022-07-04 17:31:53.387457+00	91	0.98	2
2022-07-04 17:31:55.391788+00	93	0.99	1
2022-07-04 17:31:58.393803+00	90	0.97	1
2022-07-04 17:31:52.387195+00	93	1	2
2022-07-04 17:31:57.388675+00	85	0.98	0
2022-07-04 17:31:54.390559+00	94	0.97	1
2022-07-04 17:31:58.390156+00	87	0.99	2
2022-07-04 17:31:55.38835+00	91	1	2
2022-07-04 17:31:53.38673+00	86	0.99	0
2022-07-04 17:31:54.387899+00	87	0.97	2
2022-07-04 17:31:53.389883+00	89	0.99	1
2022-07-04 17:31:57.393559+00	85	0.98	1
2022-07-04 17:31:56.388345+00	92	0.98	0

(30 rows)

Εικόνα 5.3: *Δεδομένα στην βάση δεδομένων του πίνακα health*

time	steps	calories	id
2022-07-04 17:31:51.386831+00	1504	204	2
2022-07-04 17:31:51.387562+00	1507	204	1
2022-07-04 17:31:50.386281+00	1507	202	2
2022-07-04 17:31:49.385236+00	1503	204	0
2022-07-04 17:31:50.386844+00	1504	201	1
2022-07-04 17:31:50.3858+00	1502	203	0
2022-07-04 17:31:49.385581+00	1508	205	1
2022-07-04 17:31:49.385871+00	1503	202	2
2022-07-04 17:31:51.386073+00	1504	205	0
2022-07-04 17:31:57.389793+00	1505	204	2
2022-07-04 17:31:54.387091+00	1505	201	0
2022-07-04 17:31:52.386352+00	1510	201	0
2022-07-04 17:31:55.391788+00	1507	203	1
2022-07-04 17:31:56.389553+00	1508	204	2
2022-07-04 17:31:53.387457+00	1510	203	2
2022-07-04 17:31:57.388675+00	1505	202	0
2022-07-04 17:31:58.393803+00	1503	203	1
2022-07-04 17:31:57.393559+00	1501	203	1
2022-07-04 17:31:53.389883+00	1507	204	1
2022-07-04 17:31:55.38835+00	1510	205	2
2022-07-04 17:31:58.390156+00	1509	203	2
2022-07-04 17:31:52.388826+00	1503	204	1
2022-07-04 17:31:54.390559+00	1506	204	1
2022-07-04 17:31:53.38673+00	1501	203	0
2022-07-04 17:31:55.387438+00	1504	201	0
2022-07-04 17:31:58.389803+00	1505	203	0
2022-07-04 17:31:54.387899+00	1507	205	2
2022-07-04 17:31:56.388345+00	1506	205	0
2022-07-04 17:31:56.392347+00	1509	205	1
2022-07-04 17:31:52.387195+00	1502	201	2

(30 rows)

Εικόνα 5.4: *Δεδομένα στην βάση δεδομένων του πίνακα fitness*

time	coordinates	height	id
2022-07-04 17:31:51.386073+00	{50.007329748570 01,20.002921366283665}	201	0
2022-07-04 17:31:49.385236+00	{50.00783424300062,20.00363262037442}	205	0
2022-07-04 17:31:50.386844+00	{50.00357510871612,20.00353871153248}	210	1
2022-07-04 17:31:51.386831+00	{50.00115896139267 6,20.006161072345236}	202	2
2022-07-04 17:31:49.385871+00	{50.00206588955495 6,20.00425943238018}	203	2
2022-07-04 17:31:51.387562+00	{50.00803158233242,20.00859921617486 5}	203	1
2022-07-04 17:31:50.3858+00	{50.00645135206486,20.00756402588613}	200	0
2022-07-04 17:31:49.385581+00	{50.00170959060748,20.00030239749793}	202	1
2022-07-04 17:31:50.386281+00	{50.00688876304323 6,20.004683783654542}	200	2
2022-07-04 17:31:56.388345+00	{50.00227477348864 4,20.005432533712366}	206	0
2022-07-04 17:31:53.38673+00	{50.00388205285389,20.00181595563962}	201	0
2022-07-04 17:31:52.388826+00	{50.00560076824516 6,20.005793286457024}	208	1
2022-07-04 17:31:52.386352+00	{50.00684021592183 4,20.007307648064643}	209	0
2022-07-04 17:31:57.389793+00	{50.00765515912726,20.00216325663621 2}	203	2
2022-07-04 17:31:57.388675+00	{50.00788300805558 6,20.00350449951093 5}	207	0
2022-07-04 17:31:56.392347+00	{50.00327363024917,20.00187116463779}	201	1
2022-07-04 17:31:52.387195+00	{50.00092933816138 4,20.00380192150712 2}	209	2
2022-07-04 17:31:55.391788+00	{50.00002996174054,20.00406475448534 5}	202	1
2022-07-04 17:31:56.389553+00	{50.00339220265280 5,20.007267064692137}	208	2
2022-07-04 17:31:54.387091+00	{50.00221498395461,20.00641896914729 8}	200	0
2022-07-04 17:31:58.393803+00	{50.00068844691892,20.00029900475727 4}	202	1
2022-07-04 17:31:53.387457+00	{50.00830084802607,20.00091714096834 7}	201	2
2022-07-04 17:31:54.390559+00	{50.00477140386575,20.00813169034530 5}	210	1
2022-07-04 17:31:55.387438+00	{50.00789276866388,20.00808067777260 7}	200	0
2022-07-04 17:31:54.387899+00	{50.00188869553705,20.00952341060893 5}	210	2
2022-07-04 17:31:57.393559+00	{50.0075991426268,20.00932462045980 7}	210	1
2022-07-04 17:31:58.390156+00	{50.00111463414426 5,20.005839446753583}	207	2
2022-07-04 17:31:58.389803+00	{50.00128376436542 5,20.00013004923552}	200	0
2022-07-04 17:31:53.389883+00	{50.00256587988731,20.00919000198736}	210	1
2022-07-04 17:31:55.38835+00	{50.00050497810138,20.00640561682385 5}	208	2

(30 rows)

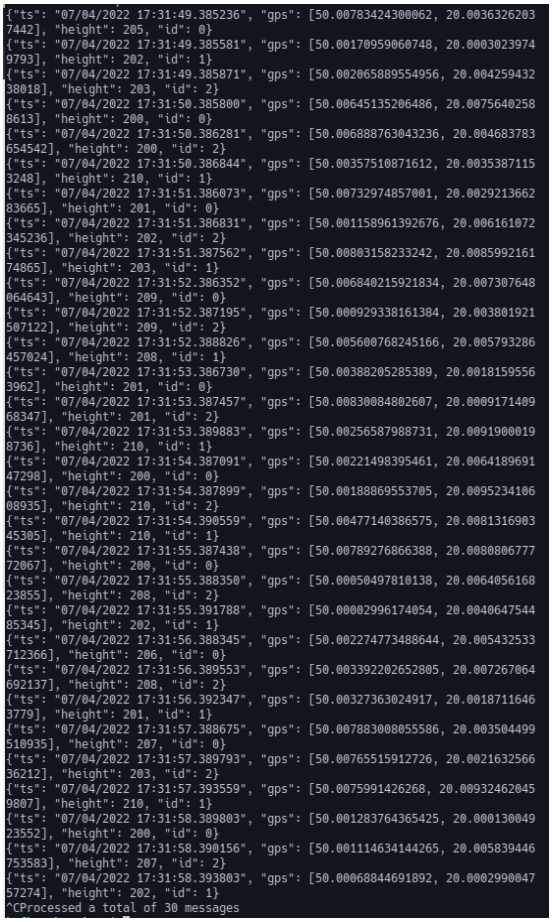
Εικόνα 5.5: *Δεδομένα στην βάση δεδομένων του πίνακα location*

{"ts": "07/04/2022 17:31:49.385236", "heartrate": 93, "spO2": 1.0, "id": 0}
{"ts": "07/04/2022 17:31:49.385581", "heartrate": 91, "spO2": 0.99, "id": 1}
{"ts": "07/04/2022 17:31:49.385871", "heartrate": 91, "spO2": 1.0, "id": 2}
{"ts": "07/04/2022 17:31:50.385800", "heartrate": 95, "spO2": 0.97, "id": 0}
{"ts": "07/04/2022 17:31:50.386281", "heartrate": 94, "spO2": 0.98, "id": 2}
{"ts": "07/04/2022 17:31:50.386844", "heartrate": 87, "spO2": 0.97, "id": 1}
{"ts": "07/04/2022 17:31:51.386073", "heartrate": 94, "spO2": 0.98, "id": 0}
{"ts": "07/04/2022 17:31:51.386831", "heartrate": 88, "spO2": 0.97, "id": 2}
{"ts": "07/04/2022 17:31:51.387562", "heartrate": 86, "spO2": 0.99, "id": 1}
{"ts": "07/04/2022 17:31:52.386352", "heartrate": 85, "spO2": 1.0, "id": 0}
{"ts": "07/04/2022 17:31:52.387195", "heartrate": 93, "spO2": 1.0, "id": 2}
{"ts": "07/04/2022 17:31:52.388826", "heartrate": 87, "spO2": 1.0, "id": 1}
{"ts": "07/04/2022 17:31:53.386730", "heartrate": 86, "spO2": 0.99, "id": 0}
{"ts": "07/04/2022 17:31:53.387457", "heartrate": 91, "spO2": 0.98, "id": 2}
{"ts": "07/04/2022 17:31:53.389883", "heartrate": 89, "spO2": 0.99, "id": 1}
{"ts": "07/04/2022 17:31:54.387091", "heartrate": 94, "spO2": 1.0, "id": 0}
{"ts": "07/04/2022 17:31:54.387899", "heartrate": 87, "spO2": 0.97, "id": 2}
{"ts": "07/04/2022 17:31:54.390559", "heartrate": 94, "spO2": 0.97, "id": 1}
{"ts": "07/04/2022 17:31:55.387438", "heartrate": 90, "spO2": 0.99, "id": 0}
{"ts": "07/04/2022 17:31:55.388350", "heartrate": 91, "spO2": 1.0, "id": 2}
{"ts": "07/04/2022 17:31:55.391788", "heartrate": 93, "spO2": 0.99, "id": 1}
{"ts": "07/04/2022 17:31:56.388345", "heartrate": 92, "spO2": 0.98, "id": 0}
{"ts": "07/04/2022 17:31:56.389553", "heartrate": 86, "spO2": 0.98, "id": 2}
{"ts": "07/04/2022 17:31:56.392347", "heartrate": 87, "spO2": 0.99, "id": 1}
{"ts": "07/04/2022 17:31:57.388675", "heartrate": 85, "spO2": 0.98, "id": 0}
{"ts": "07/04/2022 17:31:57.389793", "heartrate": 95, "spO2": 0.97, "id": 2}
{"ts": "07/04/2022 17:31:57.393559", "heartrate": 85, "spO2": 0.98, "id": 1}
{"ts": "07/04/2022 17:31:58.389803", "heartrate": 94, "spO2": 0.98, "id": 0}
{"ts": "07/04/2022 17:31:58.390156", "heartrate": 87, "spO2": 0.99, "id": 2}
{"ts": "07/04/2022 17:31:58.393803", "heartrate": 90, "spO2": 0.97, "id": 1}
^CProcessed a total of 30 messages

Εικόνα 5.6: *Δεδομένα στο kafka topic health*

{"ts": "07/04/2022 17:31:49.385236", "steps": 1503, "calories": 204, "id": 0}
{"ts": "07/04/2022 17:31:49.385581", "steps": 1508, "calories": 205, "id": 1}
{"ts": "07/04/2022 17:31:49.385871", "steps": 1503, "calories": 202, "id": 2}
{"ts": "07/04/2022 17:31:50.385800", "steps": 1502, "calories": 203, "id": 0}
{"ts": "07/04/2022 17:31:50.386281", "steps": 1507, "calories": 202, "id": 2}
{"ts": "07/04/2022 17:31:50.386844", "steps": 1504, "calories": 201, "id": 1}
{"ts": "07/04/2022 17:31:51.386073", "steps": 1504, "calories": 205, "id": 0}
{"ts": "07/04/2022 17:31:51.386831", "steps": 1504, "calories": 204, "id": 2}
{"ts": "07/04/2022 17:31:51.387562", "steps": 1507, "calories": 204, "id": 1}
{"ts": "07/04/2022 17:31:52.386352", "steps": 1510, "calories": 201, "id": 0}
{"ts": "07/04/2022 17:31:52.387195", "steps": 1502, "calories": 201, "id": 2}
{"ts": "07/04/2022 17:31:52.388826", "steps": 1503, "calories": 204, "id": 1}
{"ts": "07/04/2022 17:31:53.386730", "steps": 1501, "calories": 203, "id": 0}
{"ts": "07/04/2022 17:31:53.387457", "steps": 1510, "calories": 203, "id": 2}
{"ts": "07/04/2022 17:31:53.389883", "steps": 1507, "calories": 204, "id": 1}
{"ts": "07/04/2022 17:31:54.387091", "steps": 1505, "calories": 201, "id": 0}
{"ts": "07/04/2022 17:31:54.387899", "steps": 1507, "calories": 205, "id": 2}
{"ts": "07/04/2022 17:31:54.390559", "steps": 1506, "calories": 204, "id": 1}
{"ts": "07/04/2022 17:31:55.387438", "steps": 1504, "calories": 201, "id": 0}
{"ts": "07/04/2022 17:31:55.388350", "steps": 1510, "calories": 205, "id": 2}
{"ts": "07/04/2022 17:31:55.391788", "steps": 1507, "calories": 203, "id": 1}
{"ts": "07/04/2022 17:31:56.388345", "steps": 1506, "calories": 205, "id": 0}
{"ts": "07/04/2022 17:31:56.389553", "steps": 1508, "calories": 204, "id": 2}
{"ts": "07/04/2022 17:31:56.392347", "steps": 1509, "calories": 205, "id": 1}
{"ts": "07/04/2022 17:31:57.388675", "steps": 1505, "calories": 202, "id": 0}
{"ts": "07/04/2022 17:31:57.389793", "steps": 1505, "calories": 204, "id": 2}
{"ts": "07/04/2022 17:31:57.393559", "steps": 1501, "calories": 203, "id": 1}
{"ts": "07/04/2022 17:31:58.389803", "steps": 1505, "calories": 203, "id": 0}
{"ts": "07/04/2022 17:31:58.390156", "steps": 1509, "calories": 203, "id": 2}
{"ts": "07/04/2022 17:31:58.393803", "steps": 1503, "calories": 203, "id": 1}
^CProcessed a total of 30 messages

Εικόνα 5.7: *Δεδομένα στο kafka topic fitness*



{"ts": "07/04/2022 17:31:49.385236", "gps": [50.00783424300062, 20.0036326203
7442], "height": 205, "id": 0}
{"ts": "07/04/2022 17:31:49.385581", "gps": [50.00170959060748, 20.0003023974
9793], "height": 202, "id": 1}
{"ts": "07/04/2022 17:31:49.385871", "gps": [50.002065889554956, 20.004259432
38018], "height": 203, "id": 2}
{"ts": "07/04/2022 17:31:50.385800", "gps": [50.00645135206486, 20.0075640258
8613], "height": 200, "id": 0}
{"ts": "07/04/2022 17:31:50.386281", "gps": [50.006888763043236, 20.004683783
654542], "height": 200, "id": 2}
{"ts": "07/04/2022 17:31:50.386844", "gps": [50.00357510871612, 20.0035387115
3248], "height": 210, "id": 1}
{"ts": "07/04/2022 17:31:51.386073", "gps": [50.00732974857001, 20.0029213662
83665], "height": 201, "id": 0}
{"ts": "07/04/2022 17:31:51.386831", "gps": [50.001158961392676, 20.006161072
345236], "height": 202, "id": 2}
{"ts": "07/04/2022 17:31:51.387562", "gps": [50.00803158233242, 20.0085992161
74865], "height": 203, "id": 1}
{"ts": "07/04/2022 17:31:52.386352", "gps": [50.006840215921834, 20.007307648
064643], "height": 209, "id": 0}
{"ts": "07/04/2022 17:31:52.387195", "gps": [50.000929338161384, 20.003801921
507122], "height": 209, "id": 2}
{"ts": "07/04/2022 17:31:52.388826", "gps": [50.005600768245166, 20.005793286
457024], "height": 208, "id": 1}
{"ts": "07/04/2022 17:31:53.386730", "gps": [50.00388205285389, 20.0018159556
3962], "height": 201, "id": 0}
{"ts": "07/04/2022 17:31:53.387457", "gps": [50.00830084802607, 20.0009171409
68347], "height": 201, "id": 2}
{"ts": "07/04/2022 17:31:53.389883", "gps": [50.00256587988731, 20.0091900019
8736], "height": 210, "id": 1}
{"ts": "07/04/2022 17:31:54.387091", "gps": [50.00221498395461, 20.0064189691
47298], "height": 200, "id": 0}
{"ts": "07/04/2022 17:31:54.387899", "gps": [50.00188869553705, 20.0095234106
08935], "height": 210, "id": 2}
{"ts": "07/04/2022 17:31:54.390559", "gps": [50.00477140386575, 20.0081316903
45305], "height": 210, "id": 1}
{"ts": "07/04/2022 17:31:55.387438", "gps": [50.00789276866388, 20.0080806777
72067], "height": 200, "id": 0}
{"ts": "07/04/2022 17:31:55.388350", "gps": [50.00050497810138, 20.0064056168
23855], "height": 208, "id": 2}
{"ts": "07/04/2022 17:31:55.391788", "gps": [50.00002996174054, 20.0040647544
85345], "height": 202, "id": 1}
{"ts": "07/04/2022 17:31:56.388345", "gps": [50.002274773488644, 20.005432533
712366], "height": 206, "id": 0}
{"ts": "07/04/2022 17:31:56.389553", "gps": [50.003392202652805, 20.007267064
692137], "height": 208, "id": 2}
{"ts": "07/04/2022 17:31:56.392347", "gps": [50.00327363024917, 20.0018711646
3779], "height": 201, "id": 1}
{"ts": "07/04/2022 17:31:57.388675", "gps": [50.007883008055586, 20.003504499
510935], "height": 207, "id": 0}
{"ts": "07/04/2022 17:31:57.389793", "gps": [50.00765515912726, 20.0021632566
36212], "height": 203, "id": 2}
{"ts": "07/04/2022 17:31:57.393559", "gps": [50.0075991426268, 20.00932462045
9807], "height": 210, "id": 1}
{"ts": "07/04/2022 17:31:58.389803", "gps": [50.001283764365425, 20.000130049
23552], "height": 200, "id": 0}
{"ts": "07/04/2022 17:31:58.390156", "gps": [50.001114634144265, 20.005839446
753583], "height": 207, "id": 2}
{"ts": "07/04/2022 17:31:58.393803", "gps": [50.00068844691892, 20.0002990047
57274], "height": 202, "id": 1}
^CProcessed a total of 30 messages

Εικόνα 5.8: *Δεδομένα στο kafka topic location*

5.7 Ρυθμός διακίνησης

Για την εκτίμηση ενός μέσου όρου ρυθμού διακίνησης μηνυμάτων θα πραγματοποιηθούν 2 μικρές προσθήκες στην πλατφόρμα για την μετάδοση στατιστικών αριθμών κατά τη διάρκεια της εισαγωγής στην βάση και στην προώθηση στον kafka.

Για μια απλή μέτρηση των δεδομένων που εισάγονται στην βάση κάθε φορά θα προστεθεί μια εκτύπωση με χρονοσήμανση και το νούμερο των γραμμών που προστέθηκαν στην βάση με την copy_from. Η προσθήκη θα γίνει στην συνάρτηση `batchinsert` του database_handler.py αμέσως μετά την copy_from.

```
print(f"{datetime.now()}:{len(self.datalist)} items
    inserted in table {self.table}")
```

Για την προώθηση στον kafka θα χρησιμοποιηθεί η ενσωματωμένη λειτουργία στατιστικών του librdkafka. Για την ενεργοποίηση της πρέπει να προστεθεί στο αρχείο ρυθμίσεων του kafka handler kafka.yml εγγραφή `¨statistics.interval.ms: 10000¨` που θέτει τον ρυθμό δημοσιοποίησης των στατιστικών και στην συγκεκριμένη περίπτωση τέθηκε στα 10 δευτερόλεπτα.

Για την προβολή και επεξεργασία των στατιστικών δημιουργείται callback function με είσοδο κείμενο μορφής JSON που θα περιέχει όλα τα στατιστικά που δημοσιοποιήθηκαν. Η συγκεκριμένη συνάρτηση έχει την εξής μορφή:

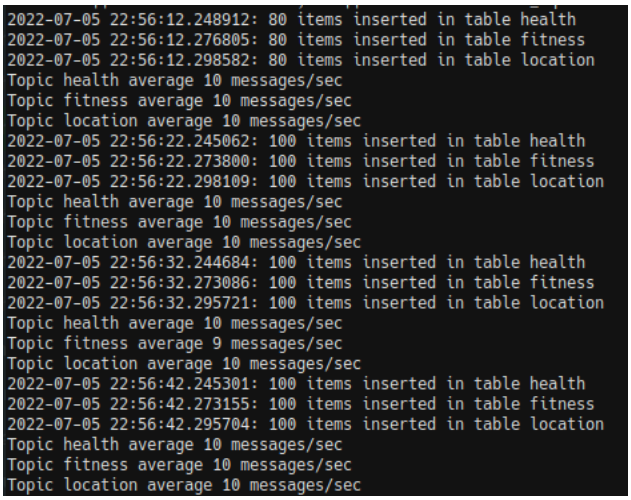
```
def stats_cb(json_str):
    json_dict = json.loads(json_str)
    for topic in json_dict['topics']:
        avg_msg = json_dict['topics'][topic]['batchcnt']['avg']
        print(f"Topic {topic} average {avg_msg} messages/sec")
```

Και για την ανάθεση της ως callback function χρειάζεται η προσθήκη της στις ρυθμίσεις με τον εξής τρόπο:

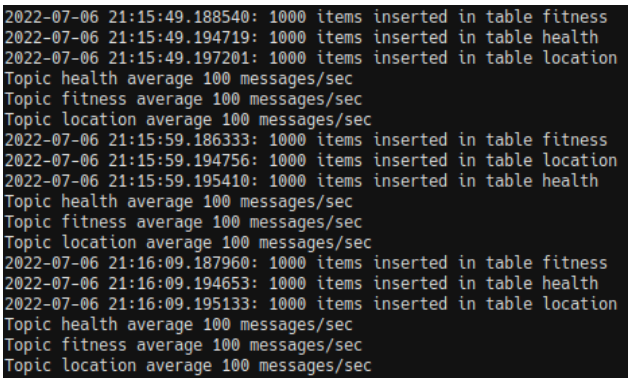
```
producer_conf['stats_cb'] = stats_cb
```

Με τις αλλαγές αυτές, κατά τη διάρκεια της παραγωγής δεδομένων με τη χρήση της εντολής `virtual_iot('smartwatch',10,100,1)` παρατηρείται η έξοδος που φαίνεται στην εικόνα 5.9. Τα αποτελέσματα της μικρής ροής δεδομένων είναι ικανοποιητικά καθώς τα 100 μηνύματα ανά 10 δευτερόλεπτα στην βάση και η προώθηση στον kafka 10 μηνυμάτων ανά δευτερόλεπτο είναι η συχνότητα της εισερχόμενης ροής δεδομένων (10 συσκευές * 1 μήνυμα ανά δευτερόλεπτο).

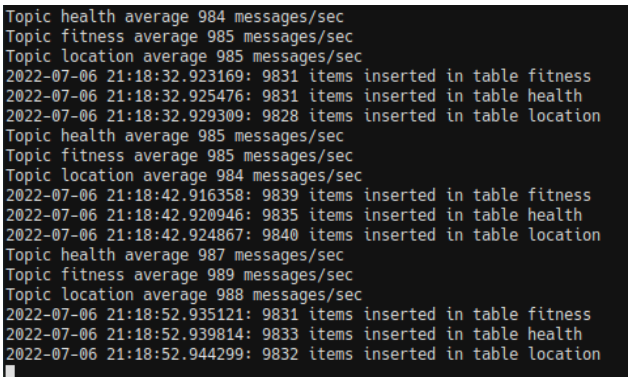
Το ίδιο αποτέλεσμα υπάρχει όπως φαίνεται και στην εικόνα 5.10 με 10 συσκευές με συχνότητα 10 μηνυμάτων το δευτερόλεπτο δηλαδή ροή 100 μηνυμάτων ανά δευτερόλεπτο. Ο περιορισμός μεγέθους ροής αρχίζει να φαίνεται σε αρκετά μεγάλα νούμερα, όπως φαίνεται στην εικόνα 5.11 για 10 συσκευές με 100 μηνύματα το δευτερόλεπτο η ροή των δεδομένων από την πλατφόρμα προς την βάση και τον kafka είναι μικρότερη από την συνολική ροή που εισέρχεται στην πλατφόρμα.



Εικόνα 5.9: *Ρυθμός δεδομένων για 10 συσκευές με 1 μήνυμα ανά δευτερόλεπτο*



Εικόνα 5.10: *Ρυθμός δεδομένων για 10 συσκευές με 10 μηνύματα ανά δευτερόλεπτο*



Εικόνα 5.11: *Ρυθμός δεδομένων για 10 συσκευές με 100 μηνύματα ανά δευτερόλεπτο*

Κεφάλαιο 6

Επίλογος

Σε αυτό το κεφάλαιο παρουσιάζονται τα συμπεράσματα της εργασίας καθώς και πιθανές μελλοντικές προεκτάσεις για την επέκταση λειτουργίας της πλατφόρμας.

6.1 Συμπεράσματα

Στα πλαίσια της διπλωματικής εργασίας εξετάστηκαν κάποιες τεχνολογίες σχετικές με συστήματα IoT και αναπτύχθηκε πλατφόρμα σε γλώσσα Python 3 για την διαχείριση των δεδομένων εντός ενός τέτοιου συστήματος. Η πλατφόρμα υποστηρίζει βάση δεδομένων PostgreSQL για την αποθήκευση των δεδομένων και Apache Kafka για την δρομολόγηση τους σε προσαρμοσμένη ροή δεδομένων. Υποστηρίζει μορφή JSON για τα μηνύματα που λαμβάνει από τις συσκευές αλλά και που αποστέλλει στον kafka. Για την βάση δεδομένων γίνεται αντιστοίχηση τιμών από κάθε μήνυμα σε συγκεκριμένες στήλες και για τον kafka γίνεται αντιστοίχηση συνόλου τιμών σε συγκεκριμένα topics. Χρησιμοποιούνται αρχεία ρυθμίσεων για την αλληλεπίδραση με τον χρήστη για το πέρασμα των στοιχείων σύνδεσης αλλά και των συσχετίσεων μεταξύ τιμών μηνυμάτων και βάσης και kafka.

Σύμφωνα με τις δοκιμές διαχειρίζεται ικανοποιητική ποσότητα δεδομένων αλλά ανάλογα τον σκοπό του δικτύου οι απαιτήσεις μπορούν να είναι πολύ μεγαλύτερες. Στην απόδοση παίζουν μεγάλο ρόλο τα τεχνικά χαρακτηριστικά των εξυπηρετητών που εμπλέκονται στην όλη διαδικασία.

6.2 Μελλοντικές Επεκτάσεις

Η συγκεκριμένη υλοποίηση της πλατφόρμας αγγίζει ένα μικρό κομμάτι εφαρμογών στα πλαίσια ενός δικτύου IoT. Μπορούν να γίνουν προσθήκες και μεταποιήσεις για καλύτερη αλληλεπίδραση με τους χρήστες, περισσότερες λειτουργίες σχετικές με τα δεδομένα και υποστήριξη περισσότερων τεχνολογιών IoT.

Για καλύτερο έλεγχο της πλατφόρμας, μπορεί να δημιουργηθεί API για έλεγχο των διεργασιών, επεξεργασία ρυθμίσεων και παραγωγή στατιστικών σχετικά με την

πλατφόρμα. Το API θα βοηθήσει επίσης στην δημιουργία γραφικού περιβάλλοντος σε ιστοσελίδα (front-end) με δυνατότητες επεξεργασία ρυθμίσεων της πλατφόρμας, παρακολούθηση κατάστασης και στατιστικών λειτουργίας παρέχοντας καλύτερη κατανόηση των ρυθμίσεων από τον χρήστη και μεγαλύτερο έλεγχο. Για την δημιουργία του API μπορεί να χρησιμοποιηθεί framework όπως django [22] ή flask [23] που είναι βιβλιοθήκες σε Python για την εύκολη ενσωμάτωση της πλατφόρμας.

Το MQTT είναι μόνο μία τεχνολογία διακίνησης δεδομένων και μια ευέλικτη πλατφόρμα έχει ανάγκη να υποστηρίζει διαφορετικές πηγές δεδομένων. Μπορεί να προστεθεί ρύθμιση για επιλογή πηγής δεδομένων αντί του MQTT όπως πχ από kafka. Το ίδιο μπορεί να γίνει αντίστοιχα και για την περαιτέρω δρομολόγηση ή αποθήκευση δεδομένων με την προσθήκη περισσοτέρων υποστηριζομένων βάσεων και τεχνολογιών ροής δεδομένων με την προσθήκη επιλογής χρήσης ενός η περισσοτέρων επιλεγμένων τελικών προορισμών.

Βιβλιογραφία

[1] *Behind The Numbers: Growth in the Internet of Things | NCTA — The Internet & Television Association*.

[2] Ashutosh Pandey. *Part 4: IoT Platforms*, 2020.

[3] Guido Van Rossum και Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[4] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic και Marimuthu Palaniswami. *Internet of Things (IoT): A vision, architectural elements, and future directions*. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[5] Li Da Xu, Wu He και Shancang Li. *Internet of Things in Industries: A Survey*. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, 2014. Conference Name: IEEE Transactions on Industrial Informatics.

[6] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui και Anne Marie Kermarrec. *The many faces of publish/subscribe*. *ACM Computing Surveys*, 35(2):114–131, 2003.

[7] *Apache Kafka*. https://kafka.apache.org/. Ημερομηνία πρόσβασης: 1-8-2022.

[8] *MQTT Version 5.0. Edited by Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. 07 March 2019. OASIS Standard*. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html. Latest version: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[9] R. A. Light. *Mosquitto: server and client implementation of the MQTT protocol*. *The Journal of Open Source Software*, 2(13), 2017.

[10] *OpenSSL*. https://www.openssl.org/. Ημερομηνία πρόσβασης: 22-6-2022.

[11] *PEM format*. https://www.rfc-editor.org/rfc/rfc7468#page-11. Ημερομηνία πρόσβασης: 22-6-2022.

[12] *systemd*. https://systemd.io/. Ημερομηνία πρόσβασης: 22-6-2022.

[13] Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte και Domagoj Vrgoč. *Foundations of JSON schema*. *Proceedings of the 25th International Conference on World Wide Web*, σελίδες 263–273. International World Wide Web Conferences Steering Committee, 2016.

[14] *mqtt-paho python library*. https://pypi.org/project/paho-mqtt/. Ημερομηνία πρόσβασης: 10-5-2022.

[15] *confluent kafka python library*. https://docs.confluent.io/kafka-clients/python/current/overview.html. Ημερομηνία πρόσβασης: 11-5-2022.

[16] *librdkafka*. https://github.com/edenhill/librdkafka. Ημερομηνία πρόσβασης: 11-5-2022.

[17] *psycopg2*. https://www.psycopg.org/. Ημερομηνία πρόσβασης: 12-5-2022.

[18] *Bulk Insert Performance Benchmark*. https://naysan.ca/2020/05/09/pandas-to-postgresql-using-psycopg2-bulk-insert-performance-benchmark/. Ημερομηνία πρόσβασης: 12-5-2022.

[19] *Watchdog*. https://pypi.org/project/watchdog/. Ημερομηνία πρόσβασης: 6-6-2022.

[20] *yaml*. https://yaml.org/. Ημερομηνία πρόσβασης: 17-6-2022.

[21] *mqtt-explorer*. http://mqtt-explorer.com/. Ημερομηνία πρόσβασης: 1-7-2022.

[22] *Django*. https://djangoproject.com/. Ημερομηνία πρόσβασης: 4-7-2022.

[23] Miguel Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

Ορισμοί

broker	Μία ενδιάμεση οντότητα που επιτρέπει την επικοινωνία μεταξύ clients.
client	Συσκευές ¨πελάτες¨ εντός ενός δικτύου.
ping	Έλεγχος κατάστασης συσκευής από μια άλλη.
PLAINTEXT	Απλό κείμενο χωρίς κρυπτογράφηση.
SSL	Μέθοδοι κρυπτογράφησης δεδομένων που ανταλλάσσονται μεταξύ συσκευών.
κατακερματισμός	Η διαδικασία μετασχηματισμού ενός κλειδιού η κειμένου σε άλλη τιμή.
python dictionary	Δομή δεδομένων στην python που αποθηκέυει τιμές σε ζεύγη key:value.
callback function	Συνάρτηση που δίνεται σε άλλη συνάρτηση ως μεταβλητή για την εκτέλεση της από αυτή.