Live Streaming and Presentation of Sensors' Data

Μοίρας Αλέξανδρος
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο
Αθήνα, Ελλάδα
el18081@mail.ntua.gr

Παντελαίος Δημήτριος
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο
Αθήνα, Ελλάδα
el18049@mail.ntua.gr

Κώδικας Εργασίας:

https://github.com/dpantelaios/Analysis-and-Design-of-Information-Systems

***Abstract*—Τα τελευταία χρόνια αυξάνονται με ραγδαίους ρυθμούς οι ηλεκτρονικές συσκευές που επιτρέπουν την σύνδεση και την ανταλλαγή δεδομένων. Για αυτό το λόγο οδηγηθήκαμε στην δημιουργία του Διαδικτύου των πραγμάτων ή αλλιώς Internet of Things(IoT), το οποίο αποτελεί ένα δίκτυο επικοινωνίας πληθώρας συσκευών και παρέχει την δυνατότητα λήψης, συγκέντρωσης, επεξεργασίας και απεικόνισης δεδομένων που παράγονται από τις edge συσκευές, αλλά και διαχείρισης της λειτουργίας τους[1]. Εμπνευσμένοι από τα διάφορα ήδη υπάρχοντα IoT συστήματα, θα δημιουργήσουμε ένα live streaming σύστημα το οποίο θα αποτελεί prototype ενός πραγματικού IoT συστήματος εξετάζοντας παράλληλα την καταλληλότητα ορισμένων εργαλείων για αυτόν τον σκοπό.**

Keywords*—*Apache Kafka, Kafka Streams, Telegraf, Grafana, Influxdb, Websockets, IoT, sensors

I. Introduction

Σκοπός της εργασίας είναι η δημιουργία μιας εφαρμογής IoT, κατά την οποία θα παράγουμε δεδομένα από αισθητήρες, θα λαμβάνονται από κάποιους κόμβους όπου θα γίνεται η απαιτούμενη επεξεργασία τους και με την σειρά τους τα επεξεργασμένα δεδομένα θα αποστέλλονται στον αντίστοιχο κόμβο, όπου θα πραγματοποιούνται η αποθήκευση τους σε μια timeseries database και η αποστολή τους για live παρουσίαση στον κατάλληλο κόμβο.

II. STEPS TO SET UP SYSTEM

Αρχικά κάνουμε git clone το repository που έχει παρατεθεί παραπάνω και περιλαμβάνει τον κώδικα της εργασίας, προκειμένου να τον κατεβάσουμε στο σύστημα από το οποίο θα γίνει η εκτέλεση του. Εναλλακτικά μπορούμε να μεταφερθούμε στο repository που περιέχει τον κώδικα και να τον κατεβάσουμε χειροκίνητα πατώντας Code → Download Zip και στην συνέχεια να τον κάνουμε extract στο σύστημα που έγινε η αποθήκευση.

Το project εκτελείται αποκλειστικά σε docker container, οπότε η μοναδική εγκατάσταση λογισμικού που θα πρέπει να κάνουμε είναι η εγκατάσταση της πλατφόρμας docker Desktop από docker Desktop Download. Το Docker είναι μία εφαρμογή που μας επιτρέπει να χτίσουμε και να τρέξουμε portable εφαρμογές σε οποιοδήποτε περιβάλλον εντός independent containers[2].

Στην συνέχεια, προκειμένου να θέσουμε σε λειτουργία όλα τα docker containers που χρησιμοποιεί η εφαρμογή πρέπει, αφού βρισκόμαστε ήδη εντός του directory που κατεβάσαμε, να ακολουθηθούν με την σειρά τα ακόλουθα βήματα:

- cd Messaging_Broker_Layer
- docker compose up
- cd Presentation Layer
- docker compose up
- cd Data-Storage Layer
- docker compose up
- cd Live Streaming Layer Java
- docker compose build
- docker compose up
- cd Device Layer
- docker compose build
- docker compose up

Θα χρειαστούμε 5 διαφορετικά terminal, ένα για κάθε Compose Project που θέλουμε να θέσουμε σε λειτουργία. Για κάθε Docker Compose Project που κάνουμε compose up, θα υπάρχει αναμονή λίγα δευτερόλεπτα μέχρι να ολοκληρωθούν οι απαραίτητες διαδικασίες initialization των containers του, ώστε να μπορούμε να συνεχίσουμε στα επόμενα βήματα. Συγκεκριμένα οι εξαρτήσεις έχουν ως εξής: Πρώτα πρέπει να σηκωθεί το Messaging Broker Layer παράλληλα με το Presentation Layer, έπειτα μπορούν να σηκωθούν τα Data Storage και Live Streaming layers, και αφού σηκωθούν όλα και ολοκληρωθεί το initialization τους, μόνο τότε μπορεί να σηκωθεί το device layer και να τεθεί η εφαρμογή σε πλήρη λειτουργία.

Μεταξύ των εκτελέσεων θα ήταν επιθυμητό να γίνονται docker compose down -v όλα τα docker containers για να μην μένει αποθηκευμένο κάποιο state, καθώς πρόκειται να ξαναστείλουμε δεδομένα για τις ίδιες ημερομηνίες.

Μόλις ολοκληρωθούν τα παραπάνω βήματα θα μεταβούμε στην διεύθυνση http://localhost:3000/ και θα γίνει σύνδεση με username και password την λέξη admin για να εισέλθουμε στο περιβάλλον του Grafana.

Από εκεί και πέρα μπορούμε να δούμε την live απεικόνιση των δεδομένων πατώντας στο link:

http://localhost:3000/d/grS74cbVz/live-streaming?orgId=1

Τέλος μπορούμε να δούμε την απεικόνιση των δεδομένων που είναι αποθηκευμένα στην βάση πατώντας το link:

http://localhost:3000/d/1Dr8HObVk/stored-data?orgId=1

Εναλλακτικά μπορεί να γίνει περιήγηση στο Grafana πατώντας Dashboards → Browse και ανοίγοντας τα δύο αποθηκευμένα dashboards που αντιστοιχούν στην live απεικόνιση και στην απεικόνιση των δεδομένων από την βάση.

III. COMPONENT, SOFTWARE

Η εφαρμογή αποτελείται από 8 components τα οποία παρουσιάζονται παρακάτω:

1. Το component device_layer είναι υπεύθυνο για την παραγωγή δεδομένων ανά 15 λεπτά για 7 αισθητήρες, ανά ημέρα για 2 αισθητήρες, 4-6 δεδομένα την ημέρα τυχαία επιλεγμένα για τον αισθητήρα ανίχνευσης κίνησης, καθώς και ετεροχρονισμένα δεδομένα για τον αισθητήρα που παράγει μετρήσεις κατανάλωσης νερού ανά 15 λεπτά. Τα ετεροχρονισμένα δεδομένα που παράγονται ανά 6 ώρες και αντιστοιχούν σε ημερομηνία 2 μέρες πίσω θεωρούνται late processed, ενώ αυτά που παράγονται ανά 30 ώρες και αντιστοιχούν σε 10 μέρες πίσω θεωρούνται late rejected.

   Οι αισθητήρες που προσομοιώθηκαν τα αντίστοιχα όρια των πιθανών τιμών τους παρουσιάζονται παρακάτω:

   TABLE I . DEVICES

Sensor	Sensor Name	Interval	Value Type	Value Range
Θερμοκρασία	TH1	15 min	float	12-35
Θερμοκρασία	TH2	15 min	float	12-35
Ενέργεια κλιματιστικών	HVAC 1	15 min	float	0-100
Ενέργεια κλιματιστικών	HVAC 2	15 min	float	0-200
Ενέργειας υπόλοιπων ηλεκτρικών συσκευών	MiAC 1	15 min	float	0-150
Ενέργειας υπόλοιπων ηλεκτρικών συσκευών	MiAC 2	15 min	float	0-200
Κατανάλωση νερού	W1	15 min	float	0-1
Συνολική ενέργεια	Etot	1 day	float	2600x24 – 1000 – 2600x24 + 1000
Συνολική κατανάλωση νερού	Wtot	1 day	float	100-120
Ανίχνευση κίνησης	Mov1	-	int	0-1

Fig1: Αισθητήρες που προσομοιώθηκαν

   Τα δεδομένα που παράγονται στο device layer αποστέλλονται στα αντίστοιχα topic που παρουσιάζονται στον παρακάτω πίνακα:

   TABLE II . TOPICS

Topic	Αισθητήρες	Partitions	Replication Factor
15min	Αισθητήρες με interval 15 λεπτών, μαζί με τα ετεροχρονισμένα δεδομένα του αισθητήρα κατανάλωσης νερού	7	3
day	Αισθητήρες με interval 1 μέρα	2	3
movementSensor	Αισθητήρας ανίχνευσης κίνησης	1	3

Fig2:Τα topic που στέλνονται τα δεδομένα κάθε sensor

   Στα δεδομένα του κάθε αισθητήρα που στέλνουμε στον kafka χρησιμοποιούμε ως key το όνομα του αισθητήρα, έτσι ώστε να στέλνονται τα δεδομένα του στο ίδιο partition (εκμεταλλευόμαστε το γεγονός ότι το destination partition, εφόσον υπάρχει key, επιλέγεται βάσει του hash του key επομένως messages με το ίδιο key θα καταλήγουν πάντα στο ίδιο partition. Αυτό μας είναι απαραίτητο καθώς το windowing στον χρόνο (θα αναλυθεί στο live streaming layer) θέλουμε να γίνεται σε επίπεδο ενός partition για κάθε αισθητήρα, ώστε να έχουμε total ordering στα records του και να αποφύγουμε out-of-order messages που θα δημιουργούσαν προβλήματα, σε ακραίες περιπτώσεις προχωρώντας απότομα τον χρόνο του stream και οδηγώντας στο κλείσιμο παραθύρων που έχουν ακόμα να λάβουν δεδομένα.

2. Τα component kafka και zookeeper χρησιμοποιούνται για την επικοινωνία μεταξύ των services. Ο zookeeper λειτουργεί ως συντονιστής των kafka clusters και διαχειρίζεται τα metadata τους, ενώ οι kafka brokers περιέχουν τα topics μέσω των οποίων επικοινωνούν οι producers και οι consumers. Συγκεκριμένα χρησιμοποιούμε τρεις kafka-brokers χρησιμοποιώντας έναν από αυτούς ως τον bootstrap server στον οποίο συνδέονται οι consumers και οι producers, ο οποίος γνωστοποιεί σε αυτούς την παρουσία των υπολοίπων brokers. Με το component kafka-init δημιουργούμε topics με συγκεκριμένα ονόματα στα οποία τα services μπορούν να γράφουν και να διαβάζουν δεδομένα. Ο αριθμός των partitions ανά topic επιλέχθηκε με βάση τον αριθμό των αισθητήρων-διακριτών δεδομένων που γράφουν-γράφονται σε κάθε topic ώστε να επιτύχουμε τον μέγιστο δυνατό βαθμό

παραλληλίας. Θέλουμε ιδανικά κάθε αισθητήρας να γράφει σε δικό του partition ώστε για τους διάφορους ανεξάρτητους αισθητήρες να μπορούν να καταναλωθούν δεδομένα ταυτόχρονα από τους consumers που βλέπουν τα partitions. Π.χ. για τους 15min αισθητήρες έχουμε 7 συσκευές που γράφουν εκεί (th1, th2, hvac1, hvac2, miac1, miac2, w1) επομένως επιλέγουμε να χρησιμοποιήσουμε 7 partitions. Το replication factor 3 μας εγγυάται ότι ακόμα και αν κάποιος broker αποτύχει, ένας από τους άλλους δύο θα αναλάβει τα partitions στα οποία ήταν ηγέτης ο πεσών broker. Με αυτόν τον τρόπο εξασφαλίζουμε το scalability του broker, καθώς επιπλέον αισθητήρες στο device layer απλώς θα οδηγήσουν σε μία trivial αύξηση των partitions των κατάλληλων topics, τα οποία έχουν τη δυνατότητα να μοιραστούν σε 3 διαφορετικούς brokers οι οποίοι δύνανται να χειριστούν χιλιάδες partitions χωρίς πρόβλημα. Επομένως αύξηση των devices δεν αναμένεται να έχει δυσμενείς επιπτώσεις στον broker μας. Ο κάθε broker θα είναι είτε ο ηγέτης για ένα partition ενός topic και θα εξυπηρετεί producers, consumers που συνδέονται με αυτό οι οποίοι ανταλλάσσουν μηνύματα, είτε θα ακολουθεί έναν άλλον ηγέτη και θα είναι backup του. Ακόμη το kafka χρησιμοποιείται στο παρασκήνιο για την backup αποθήκευση κάποιων aggregated δεδομένων και κάποιων states (changelog topics) κατά την επεξεργασία των δεδομένων στο live streaming layer, το οποίο θα αναλυθεί παρακάτω.

3. Το component live_streaming_layer java διαβάζει από τα δεδομένα των αισθητήρων από τα αντίστοιχα topics και πραγματοποιεί την απαιτούμενη επεξεργασία σε κάθε αισθητήρα.

TABLE III AGGREGATIONS

Αισθητήρας	Είδος επεξεργασίας
TH1	Μέσος όρος τη μέρα
TH2	Μέσος όρος τη μέρα
HVAC1	Άθροισμα τη μέρα
HVAC2	Άθροισμα τη μέρα
MiAC1	Άθροισμα τη μέρα
MiAC2	Άθροισμα τη μέρα
W1	Άθροισμα τη μέρα συμπεριλαμβανομένων των late processed events
Etot	Μέγιστο κάθε μέρα
Wtot	Μέγιστο κάθε μέρα
Mov1	Συνολικό άθροισμα
Etot	Διαφορά μεταξύ διαδοχικών ημερών
Wtot	Διαφορά μεταξύ διαδοχικών ημερών

Hvac1, Hvac2, MiAC1, MiAC1, Etot	Etot – (Hvac1_daily_sum + Hvac2_daily_sum + MiAC1_daily_sum + MiAC1_daily_sum)
W1, Wtot	Wtot – W1_daily_sum

Fig3:Τύπος επεξεργασίας για κάθε αισθητήρα

Προκειμένου να γίνουν οι επεξεργασίες που φαίνονται στον πίνακα 3 και να προωθηθούν τα δεδομένα στο επόμενο layer, χρησιμοποιείται η βιβλιοθήκη Apache Kafka Streams η οποία είναι μία βιβλιοθήκη της Java που μας επιτρέπει να δημιουργούμε ροή δεδομένων μεταξύ Kafka topics με ενδιάμεση επεξεργασία τους.

Συγκεκριμένα δημιουργούμε streams που καταναλώνουν και από τα 3 input topics τα οποία ύστερα διακλαδίζονται. Το topic των δεδομένων που στέλνονται με 15 λεπτά interval περνά από ένα φίλτρο το οποίο έχει υλοποιηθεί με έναν custom transformer ο οποίος αποθηκεύει την μέγιστη ημερομηνία που έχει συναντήσει μέχρι στιγμής και αν η ημερομηνία του νέου δεδομένου είναι τουλάχιστον 10 μέρες παλαιότερη αναγνωρίζεται ως late rejected και φιλτράρεται εκτός από την επεξεργασία των δεδομένων και αποστέλλεται σε ξεχωριστό topic στον message broker. Τα raw δεδομένα όλων των topics τα οποία γίνονται δεκτά προωθούνται στα topics εξόδου, ενώ επίσης προχωρούν για περαιτέρω επεξεργασία. Με χρήση Tumbling Windows συσσωρεύουμε τα 15min δεδομένα μίας ημέρας και εκτελούμε το κατάλληλο aggregation (Average ή Sum) όπως περιγράφεται στον πίνακα III. Συγκεκριμένα για τα δεδομένα που αθροίζονται στα οποία περιλαμβάνονται οι μετρήσεις του αισθητήρα w1 ο οποίος στην περίπτωση μας έχει late accepted events. Για να τα χειριστούμε εφαρμόζουμε grace period δύο ημερών στο window που χειρίζεται αυτό το aggregation. Αυτό σημαίνει ότι το window δε θα κλείσει έως ότου περάσουν δύο μέρες από τη λήξη του και εάν εντός αυτών έρθει δεδομένο το οποίο πρέπει να συμπεριληφθεί σε προηγούμενο window, θα συμπεριληφθεί. Αυτό προφανώς δημιουργεί latency μέχρι να δούμε τα πρώτα δεδομένα αυτού του είδους για αυτό το χρησιμοποιήσαμε μόνο στα Sum Aggregated δεδομένα, ωστόσο μπορεί να χρησιμοποιηθεί χωρίς βλάβη της γενικότητας και στα υπόλοιπα aggregated δεδομένα. Αντίστοιχα συσσωρεύουμε και τα daily δεδομένα και υπολογίζουμε το Max της μέρας (στην περίπτωσή μας έχουμε ένα τέτοιο δεδομένο ανά ημέρα οπότε το Max πάντα θα ταυτίζεται με το αντίστοιχο raw). Τα aggregated δεδομένα και αυτά αποστέλλονται στον kafka για να προχωρήσουν στο επόμενο layer και συνεχίζουν εντός των streams για να περάσουν από επόμενο στάδιο επεξεργασίας. Το επόμενο στάδιο για τα δεδομένα των 15’ είναι η άθροιση όλων των αισθητήρων ενέργειας μεταξύ τους (αντίστοιχα όλων των αισθητήρων νερού) ώστε μετά αυτό το άθροισμα να αφαιρεθεί από τη συνολική κατανάλωση ενέργειας (αντίστοιχα νερού)

εκείνη τη μέρα για να υπολογιστεί η διαρροή. Προς αυτό κάνουμε group μαζί όλα τα sum aggregated δεδομένα που αφορούν αισθητήρες ενέργειας (αντ. νερού) για την ίδια μέρα και πάλι με tumbling window μίας μέρας τα αθροίζουμε μεταξύ τους. Για να υπολογίσουμε τη συνολική κατανάλωση μίας μέρας είτε ενέργειας είτε νερού αφαιρούμε από τη συνολική κατανάλωση έως εκείνη τη μέρα τη συνολική κατανάλωση έως την προηγούμενη μέρα. Αυτό πάλι γίνεται με tumbling window δύο ημερών στα aggregated (Max) daily δεδομένα. Οι ημερήσιες καταναλώσεις πάλι προωθούνται στο επόμενο layer και συνεχίζουν εντός του συστήματος για να υπολογιστούν οι απώλειες. Για να υπολογιστούν οι διαρροές θέτουμε ως key τόσο στα αθροίσματα όλων των συσκευών, όσο και στις ημερήσιες συνολικές καταναλώσεις την ημερομηνία και εφαρμόζουμε inner join βάσει αυτής μεταξύ των 2 streams, οπότε προωθούμε και το αποτελέσμα της μεταξύ τους αφαίρεσης μέσω του κατάλληλου kafka topic στο επόμενο layer.

Τα δεδομένα του movement sensor τα χειριζόμαστε με ειδικό τρόπο. Έχουμε δημιουργήσει πάλι έναν custom transformer ο οποίος για κάθε άσσο που του έρχεται από έναν movement sensor αυξάνει έναν μετρητή και εν συνεχεία αποστέλλει πόσα movements έχει μετρήσει ως το τελευταίο που μέτρησε.

Προφανώς με ένα thread και συνεπώς έναν consumer δεν μπορούμε να εκμεταλλευτούμε την παραλληλία που μας προσφέρει ο kafka με τα πολλά partitions τα οποία μπορούν να διαβάζονται/γράφονται ταυτόχρονα. Για αυτό η βιβλιοθήκη Kafka Streams δίνει τη δυνατότητα να γίνουν launch ταυτόχρονα πολλά instances της ίδιας εφαρμογής στα οποία θα φροντίσει η βιβλιοθήκη να μοιράσει τη δουλεία σε μονάδες που τις ονομάζει tasks. Ο αριθμός των tasks, δηλαδή ο βαθμός της παραλληλίας, περιορίζεται τόσο από τον αριθμό των partitions (που απεικονίζει τον βαθμό παραλληλίας που μπορεί να προσφέρει ο Kafka) όσο και από τις εξαρτήσεις μεταξύ streams. Φροντίσαμε ώστε ο Java Consumer μας να εκμεταλλεύεται στο έπακρο των αριθμό των partitions που του προσφέρονται με το να μη δημιουργούμε εξαρτήσεις μεταξύ διαφορετικών streams παρά μόνο εκεί που είναι απαραίτητο, και σε μεγάλο βάθος της ροής των δεδομένων, όπως πχ στον υπολογισμό των διαρροών. Αυτό έχει ως αποτέλεσμα η Kafka Streams βιβλιοθήκη να δημιουργεί συνολικά 54 tasks τα οποία μπορούν να ανατεθούν σε έως και 54 εργάτες και να εκτελούνται παράλληλα. Για τους σκοπούς της άσκησης και τα συστήματα που διαθέτουμε προφανώς θα χρησιμοποιήσουμε μικρότερο αριθμό tasks θυσιάζοντας μέρος της παραλληλίας.

TABLE IV TOPICS

Topic	Δεδομένα
raw	Ολα τα δεδομένα 15λεπτων αισθητήρων πριν την επεξεργασία
aggDay15min	Ολα τα δεδομένα 15λεπτων αισθητήρων μετά την επεξεργασία (μέσος όρος ή άθροισμα)
aggDayDiff	Ολα τα δεδομένα ημερήσιων αισθητήρων μετά την επεξεργασία (διαφορά μεταξύ διαδοχικών ημερών)
leaks	Το αποτέλεσμα της καθημερινής διαφοράς της συνολικής κατανάλωσης ενέργειας από το άθροισμα των τιμών κατανάλωσης ενέργειας που στέλνουν οι αντίστοιχοι αισθητήρες ανά 15 λεπτά και η αντίστοιχη διαφορά για την κατανάλωση νερού
totalMovements	Συνολικές ανιχνεύσεις κίνησης μέχρι την δεδομένη ημερομηνία
lateRejected	Τα δεδομένα του αισθητήρα νερού w1 που ήρθαν καθυστερημένα κατά 10 μέρες

Fig4: Τα δεδομένα που στέλνονται σε κάθε topic

4. Το component influxdb αποτελεί την timeseries βάση δεδομένων στην οποία αποθηκεύονται τόσο το αρχικά όσο και τα επεξεργασμένα δεδομένα.

   Το component telegraf λαμβάνει τα δεδομένα από τα κατάλληλα topic σε μορφή json και κάνει τις απαραίτητες αλλαγές τα στέλνει σε μορφή influx τόσο στην βάση για την εισαγωγή τους, όσο και μέσω web sockets στο Grafana για την απεικόνιση τους.

   Μία από τις βασικές αλλαγές που κάνει το telegraf είναι να παίρνει την ημερομηνία της κάθε εγγραφής που λαμβάνει από τα field σε millisecond και να την μεταφέρει στο πεδίο timestamp, προκειμένου να αποθηκευτεί η εγγραφή με την σωστή ημερομηνία στην βάση δεδομένων και όχι με την ημερομηνία εισαγωγής της σε αυτή.

   Ακόμη στα δεδομένα που είναι διαθέσιμο το όνομα του αισθητήρα στον οποίο αναφέρονται, μεταφέρεται στο όνομα του measurement μαζί με το αντίστοιχο prefix που αναφέρεται στον τύπο των δεδομένων, ώστε να είναι ευδιάκριτα τα ονόματα των tables στην βάση δεδομένων και να αντιπροσωπεύουν τα δεδομένα που περιέχουν.

   Τέλος στις διαρροές προστέθηκε το είδος της κατανάλωσης στην οποία αναφέρεται, το οποίο λαμβάνεται και πάλι από το αντίστοιχο πεδίο στα fields της κάθε εγγραφής.

   Στον παρακάτω πίνακα παρουσιάζονται τα ονόματα των measurements, όσο και τα δεδομένα που αντιπροσωπεύουν.

   TABLE V MEASUREMENT NAMES

Όνομα measurement	Δεδομένα που περιλαμβάνει
SensorName	Μη επεξεργασμένα δεδομένα αντίστοιχου αισθητήρα
“aggDay15min_” + sensorName	Ημερήσιος μέσος όρος ή άθροισμα αντίστοιχου 15λεπτου αισθητήρα
“aggDayDiff_” + sensorName	Διαφορά τιμών διαδοχικών ημερών αντίστοιχου ημερήσιου αισθητήρα
leakType + “_leak”	Διαρροή κατανάλωσης ενέργειας ή νερού
“Total_moves”	Συνολικός αριθμός ανιχνεύσεων κίνησης μέχρι την δεδομένη ημερομηνία
“Late_rejected” + sensorName	Τα Late rejected δεδομένα του αντίστοιχου αισθητήρα

Fig5: Όνομα measurement και τα δεδομένα που περιέχει

5. Το component grafana-server2 είναι υπεύθυνο για την απεικόνιση των δεδομένων σε διαγράμματα και πίνακες. Αρχικά δημιουργούμε ένα API key, με το οποίο θα μπορεί το component telegraf να στέλνει δεδομένα μέσω web sockets για live απεικόνιση. Ακόμη γίνεται σύνδεση της βάσης δεδομένων influxdb, προκειμένου να λαμβάνουμε δεδομένα και από εκεί για απεικόνιση.

   Δημιουργούμε ένα dashboard που περιλαμβάνει panels για live απεικονίσεις. Πιο συγκεκριμένα περιλαμβάνει time series απεικονίσεις για τους αισθητήρες TH1, W1, Wtot, τα αποτελέσματα των aggregation για TH1, W1, την διαρροή του νερού, αλλά και πίνακες για τον TH1 και πίνακα με τα Late rejected events.

   Ακόμη δημιουργούμε ένα dashboard που περιλαμβάνει αντίστοιχα panels, όπως στην live παρουσίαση, για απεικονίσεις δεδομένων από την βάση.

   Στο τέλος κάνουμε mount το image του component, ώστε να μην χρειάζεται να τα δημιουργούμε όλα ξανά από την αρχή κάθε φορά, αλλά να μπορούμε να έχουμε πρόσβαση με εύκολο τρόπο απλώς πατώντας στα link που περιλαμβάνονται στα βήματα για να σετάρουμε την εφαρμογή μας.

Παρακάτω στις εικόνες 8 και 9 παρουσιάζονται τόσο το component diagram, το οποίο περιλαμβάνει όλα τα components αλλά και το πως συνδέονται μεταξύ τους, όσο και το deployment diagram το οποίο παρουσιάζει την δομή του συστήματος, αλλά και τα components που υπάρχουν σε κάθε κόμβο[3].

Το software που χρησιμοποιήθηκε

- Python3.10.10 για την παραγωγή των δεδομένων των αισθητήρων. Η Python είναι διερμηνευόμενη, γενικού σκοπού και υψηλού επιπέδου, γλώσσα προγραμματισμού και 3.10.10 είναι η τελευταία της έκδοση την δεδομένη στιγμή[4].

- Kafka για την μεταφορά δεδομένων ανάμεσα στα services και την επεξεργασία τους μέσω streams. Kafka είναι μια distributed event streaming πλατφόρμα που χρησιμοποιείται για επεξεργασία ροών δεδομένων[5],[6].

  Πιο συγκεκριμένα για την επεξεργασία των δεδομένων γίνεται χρήση των Kafka streams τα οποία αποτελούν μια βιβλιοθήκη ανάπτυξης εφαρμογών, όπου η είσοδος και η έξοδος αποθηκεύονται σε clusters του kafka[7].

- telegraf για την λήψη των δεδομένων από τον kafka και την αποστολή τους στην κατάλληλη μορφή, τόσο για την είσοδο τους στην time series βάση δεδομένων, όσο και στην αποστολή τους μέσω websockets στο grafana για την live απεικόνιση τους.

  Το telegraf είναι ένας ανοιχτού-κώδικα server agent που βασίζεται σε plugins και χρησιμοποιείται για την συλλογή και μεταφορά metrics[8].

- influxdb για την αποθήκευση των metrics που στέλνει το telegraf.

  InfluxDB είναι μια Time Series πλατφόρμα δεδομένων που χρησιμοποιείται κυρίως σε εφαρμογές Internet of Things, analytics και cloud[9].

- Grafana για την απεικόνιση των δεδομένων τόσο live όταν προέρχονται μέσω websockets από το telegraf, όσο και χωρίς αυτόματη ενημέρωση όταν προέρχονται από την influxdb

  Grafana είναι ένα εργαλείο ανοιχτού-κώδικα συμβατό με διαφορετικούς τύπους υπολογιστών και λειτουργικών συστημάτων και χρησιμοποιείται για analytics και για οπτικοποίηση των δεδομένων. Παρέχει πολλών ειδών διαγράμματα, γραφικές παραστάσεις και ειδοποιήσεις για το web, όταν συνδέεται με υποστηριζόμενες πηγές δεδομένων[10].

IV. RESULTS-CONCLUSION

Τα αποτελέσματα της εφαρμογής φαίνονται στο τελικό στάδιο, όπου τα δεδομένα μπαίνουν στην βάση δεδομένων, αλλά και στην απεικόνιση τους από το Grafana τόσο live μέσω websocket, όσο και χωρίς αυτόματη ενημέρωση από την time series βάση δεδομένων influxdb.
Στις παρακάτω εικόνες παρουσιάζονται ενδεικτικά τα περιεχόμενα κάποιον measurements στην βάση δεδομένων.

```
name: LateRejected_w1
time                host          sensorName      value
----                ----          ----------      -----
1588917600000000000 73d0ba79050c LateRejected_w1 0.63
1589025600000000000 73d0ba79050c LateRejected_w1 0.82
1589133600000000000 73d0ba79050c LateRejected_w1 0.51
1589241600000000000 73d0ba79050c LateRejected_w1 0.13
1589349600000000000 73d0ba79050c LateRejected_w1 0.7
```

Fig 6: Late rejected δεδομένα του 15λεπτου αισθητήρα w1 στην βάση δεδομένων.

```
name: th1
time                host          sensorName value
----                ----          ---------- -----
1589673600000000000 73d0ba79050c th1        34.2
1589674500000000000 73d0ba79050c th1        18.06
1589675400000000000 73d0ba79050c th1        23.76
1589676300000000000 73d0ba79050c th1        25.1
1589677200000000000 73d0ba79050c th1        33.74
1589678100000000000 73d0ba79050c th1        18.59
```

Fig 7: Μη επεξεργασμένα δεδομένα αισθητήρα th1 στην βάση δεδομένων.

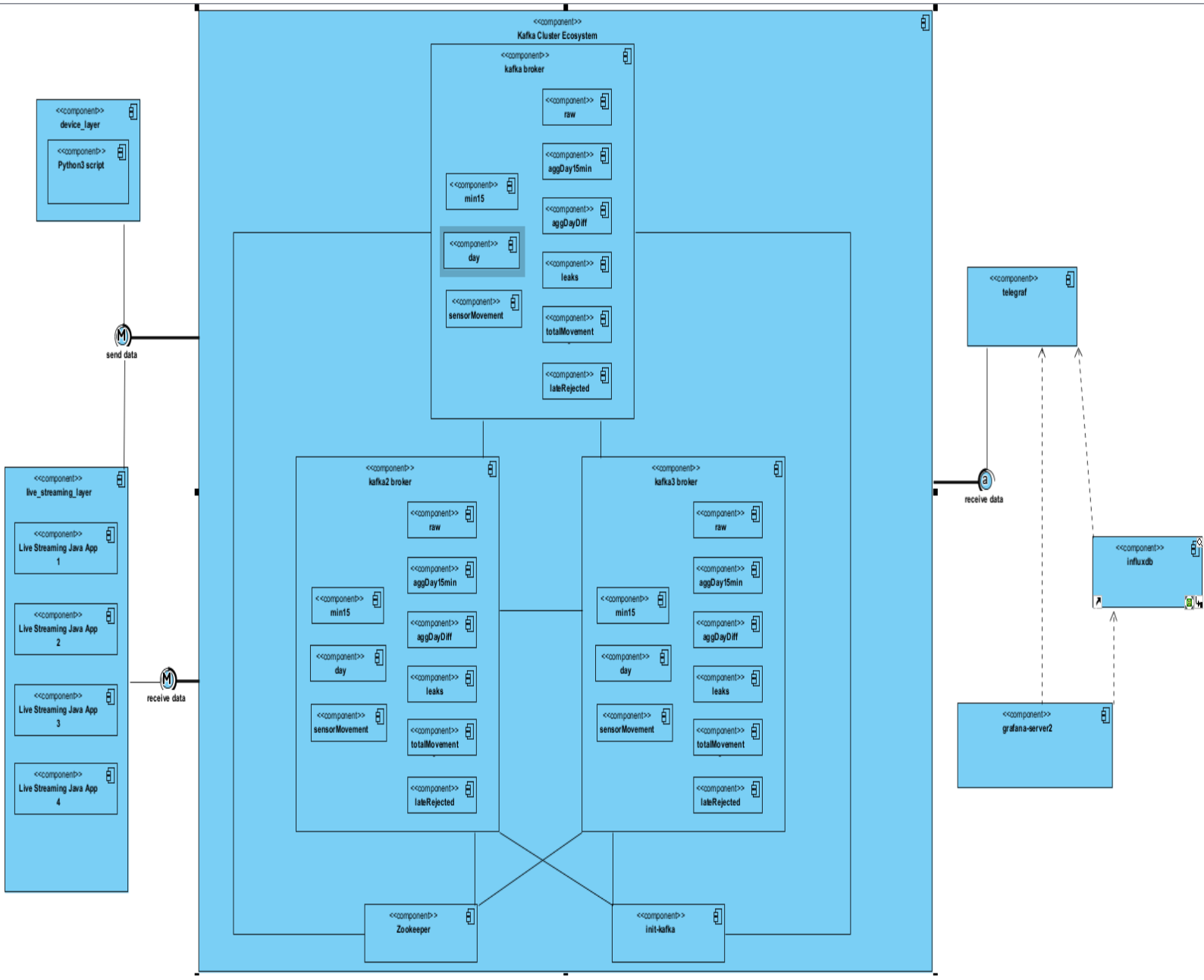


Fig8: Component diagram της εφαρμογής μας

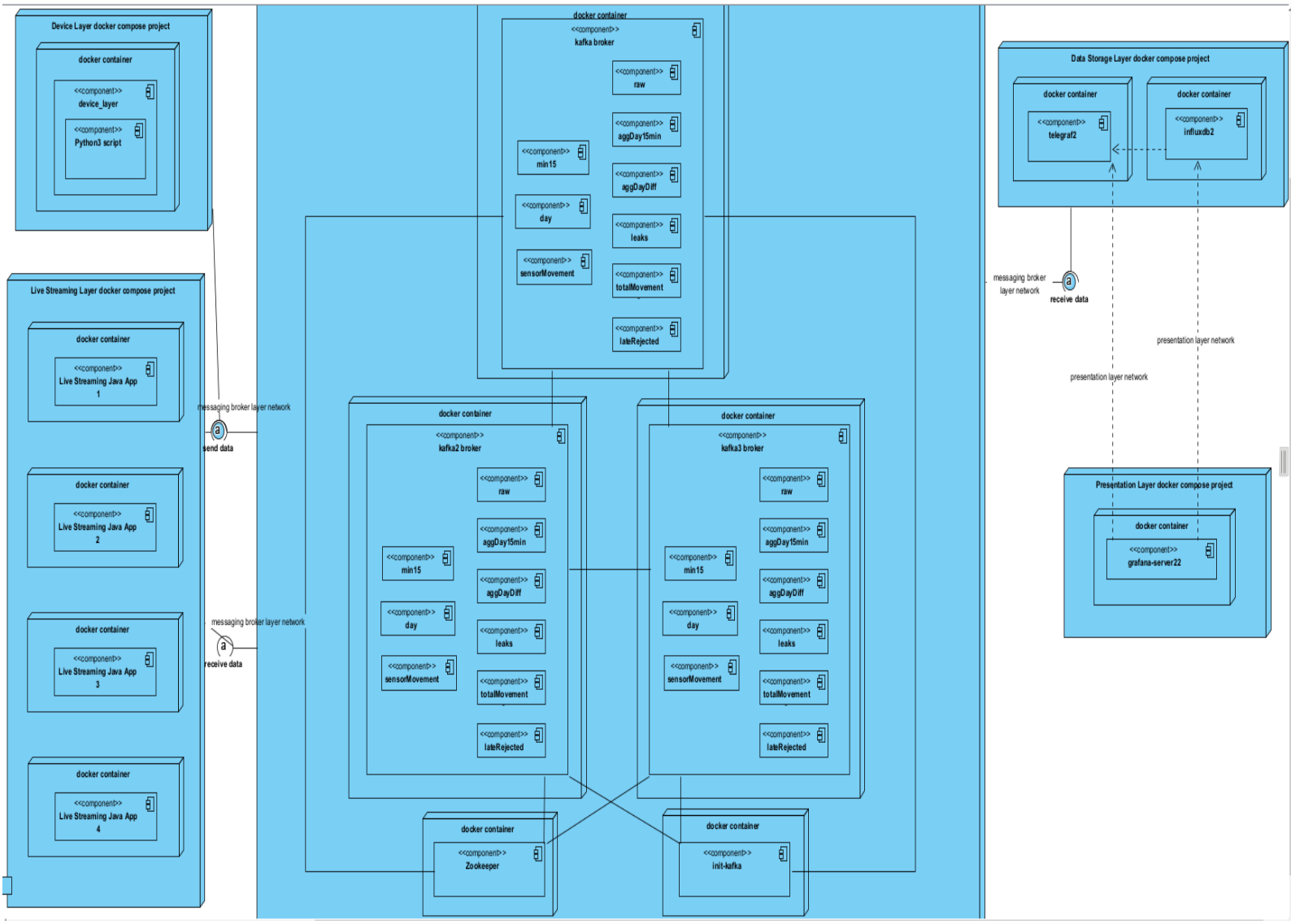


Fig9: Deployment diagram της εφαρμογής μας

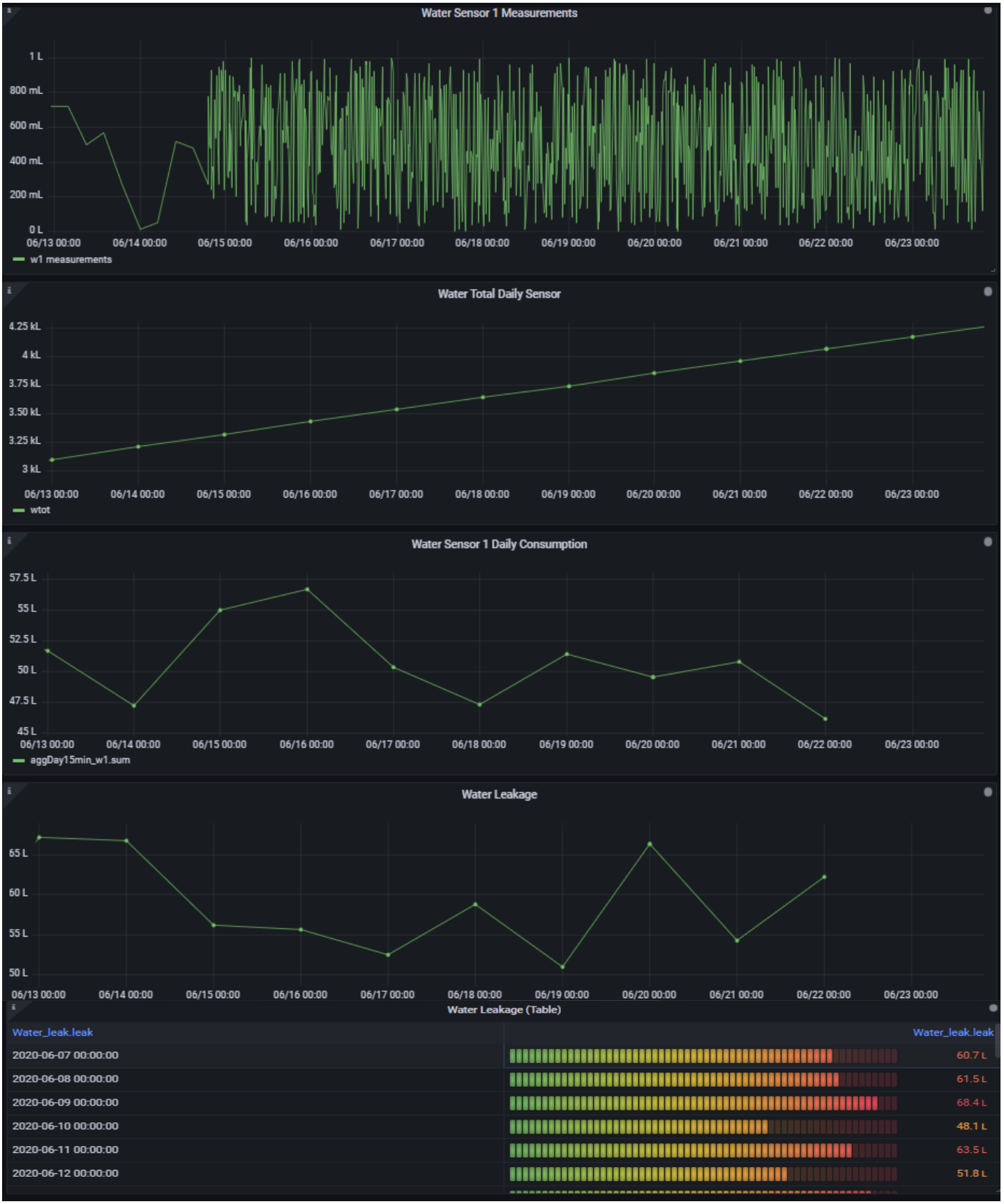


Fig10: Απεικόνιση διαγραμμάτων για τα δεδομένα του 15λεπτου αισθητήρα νερού, το ημερήσιο άθροισμα των τιμών του, τα δεδομένα του ημερήσιου αισθητήρα νερού, αλλά και η ημερήσια διαρροή νερού που προκύπτει αφαιρώντας από την συνολική ημερήσια κατανάλωση του νερού το άθροισμα των τιμών του 15λεπτου αισθητήρα κατανάλωσης νερού για την ίδια μέρα.

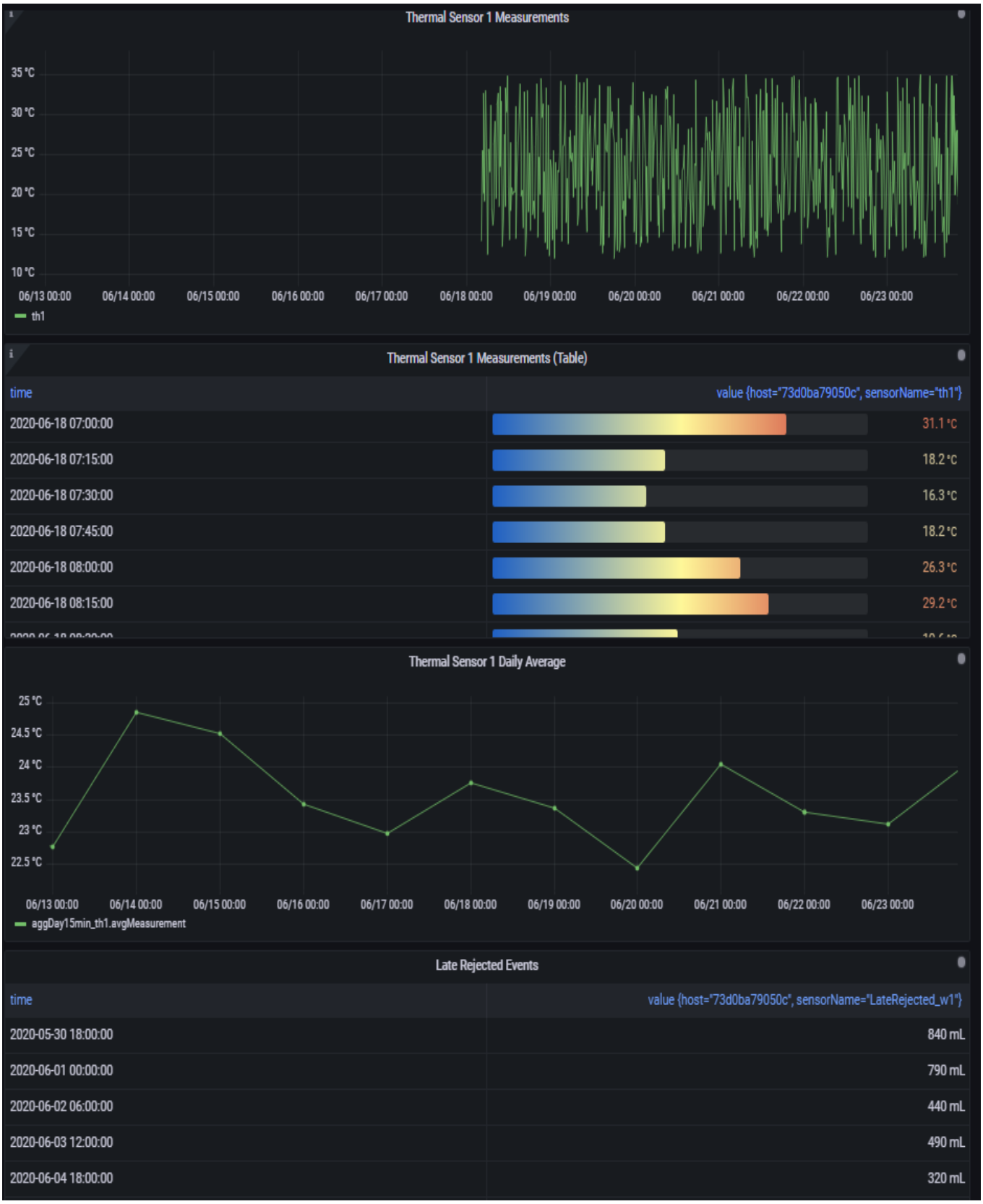
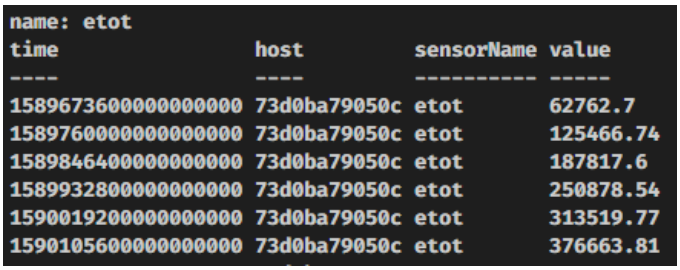
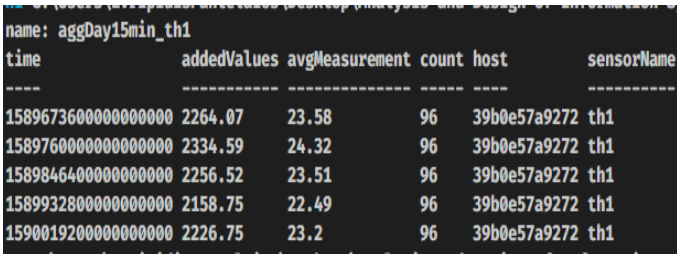


Fig11: Απεικόνιση των μη επεξεργασμένων δεδομένων του 15λεπτου αισθητήρα θερμοκρασίας th1 σε γραφική παράσταση και σε πίνακα, του ημερήσιου μέσου όρου αυτών των δεδομένων αλλά και τα late rejected events του αισθητήρα νερού w1



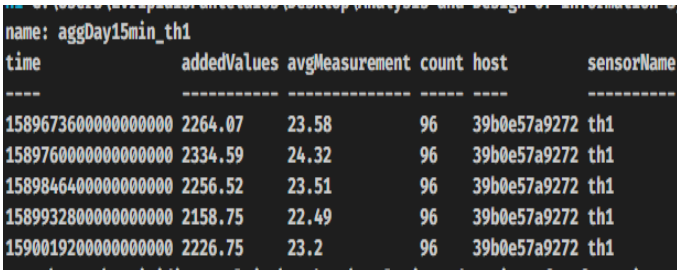
```
name: etot
time                host          sensorName value
----                ----          ---------- -----
1589673600000000000 73d0ba79050c etot       62762.7
1589760000000000000 73d0ba79050c etot       125466.74
1589846400000000000 73d0ba79050c etot       187817.6
1589932800000000000 73d0ba79050c etot       250878.54
1590019200000000000 73d0ba79050c etot       313519.77
1590105600000000000 73d0ba79050c etot       376663.81
```

Fig 12: Μη επεξεργασμένα ημερήσια δεδομένα αισθητήρα etot στην βάση δεδομένων.



```
name: aggDay15min_th1
time                addedValues avgMeasurement count host         sensorName
----                ----------- -------------- ----- ----         ----------
1589673600000000000 2264.07     23.58          96    39b0e57a9272 th1
1589760000000000000 2334.59     24.32          96    39b0e57a9272 th1
1589846400000000000 2256.52     23.51          96    39b0e57a9272 th1
1589932800000000000 2158.75     22.49          96    39b0e57a9272 th1
1590019200000000000 2226.75     23.2           96    39b0e57a9272 th1
```

Fig 13: Ημερήσιος μέσος όρος δεδομένων του 15λεπτου αισθητήρα th1 στην βάση δεδομένων.



```
name: aggDay15min_th1
time                addedValues avgMeasurement count host         sensorName
----                ----------- -------------- ----- ----         ----------
1589673600000000000 2264.07     23.58          96    39b0e57a9272 th1
1589760000000000000 2334.59     24.32          96    39b0e57a9272 th1
1589846400000000000 2256.52     23.51          96    39b0e57a9272 th1
1589932800000000000 2158.75     22.49          96    39b0e57a9272 th1
1590019200000000000 2226.75     23.2           96    39b0e57a9272 th1
```

Fig 14: Ημερήσιο άθροισμα δεδομένων του 15λεπτου αισθητήρα hvac1 στην βάση δεδομένων.

Η απεικόνιση των παραπάνω δεδομένων γίνεται στο Grafana, συνδέοντας το με την time series βάση δεδομένων και φαίνεται στα διαγράμματα 10 και 11

Από όλα τα παραπάνω αντιλαμβανόμαστε την χρησιμότητα των IoT εφαρμογών, καθώς δίνεται η δυνατότητα με εύκολο και γρήγορο τρόπο, ακόμα και για πάρα πολύ μεγάλο αριθμό αισθητήρων, να υπάρχει απεικόνιση των δεδομένων που προέρχονται από τους αισθητήρες και να υπάρχει γρήγορος εντοπισμός αν υπάρχει βλάβη σε κάποιο σημείο του συστήματος και γρήγορη ενημέρωση για τις συνθήκες που επικρατούν στον χώρο, είτε μέσω παρατήρησης των raw μετρήσεων των αισθητήρων είτε μέσω των επεξεργασμένων δεδομένων, όπως είναι η διαρροή ενέργειας, διαρροή νερού ή και το ημερήσιο aggregation που εφαρμόζεται σε κάθε αισθητήρα.

Όσον αφορά τα εργαλεία που χρησιμοποιήθηκαν, είναι εμφανές ότι ο Kafka είναι κατάλληλος για ένα τέτοιο project, καθώς παρέχει πολύ γρήγορη και άμεση επικοινωνία μεταξύ των components, μπορεί να κλιμακώσει για χιλιάδες devices όπως εξηγήθηκε παραπάνω, ενώ παρέχει και fault tolerance.

Η βιβλιοθήκη kafka streams μας επιτρέπει να επεξεργαζόμαστε ταυτόχρονα και να εκτελούμε υπολογισμούς σε πολλά διαφορετικά streams και μάλιστα παράλληλα με το launch πολλών tasks εξασφαλίζοντας μας την κλιμακωσιμότητα και στο live streaming layer.

Το telegraf αποτελεί έναν γρήγορο και αξιόπιστο connector ικανό να χειριστεί τον όγκο των δεδομένων.

Η influxdb ως timeseries βάση εξυπηρετεί πλήρως την ανάγκη μας να αποθηκεύουμε μετρήσεις στην μονάδα του χρόνου, ενώ το Grafana παρέχει άνετη απεικόνιση όλων αυτών των δεδομένων και δύναται να προσαρμοστεί εύκολα για να απεικονίζει περισσότερα ή διαφορετικά δεδομένα.

V. Acknowledgment

Θα θέλαμε να ευχαριστήσουμε την καθηγήτρια Vassiliki (Verena) Kantere και τον Παρασκευά Κερασιώτη για την πολύτιμη και άμεση βοήθεια τους, καθ'όλη την διάρκεια της εργασίας.

References

[1] https://el.wikipedia.org/wiki/Διαδίκτυο_των_πραγμάτων
[2] https://www.docker.com/
[3] https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/
[4] https://el.wikipedia.org/wiki/Python
[5] https://el.wikipedia.org/wiki/Apache_Kafka
[6] https://kafka.apache.org/
[7] https://kafka.apache.org/documentation/streams/
[8] https://www.influxdata.com/time-series-platform/telegraf/
[9] https://www.influxdata.com/
[10] https://en.wikipedia.org/wiki/Grafana