Ανάλυση και Σχεδιασμός Πληροφοριακών Συστημάτων: Εργασία 1.6

Δελήμπασης Λεωνίδας
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο
Αθήνα, Ελλάδα
el18174@mail.ntua.gr

Τασιόπουλος Νικόλαος
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο
Αθήνα, Ελλάδα
el18858@mail.ntua.gr

Βόγκας Δημήτριος
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο
Αθήνα, Ελλάδα
el18007@mail.ntua.gr

***Περίληψη*—Στην παρούσα αναφορά παρουσιάζεται η εργασία της ομάδας 47 για το θέμα 1.6 των θεμάτων της Κας Καντερέ με τίτλο IoT Live Streaming.**

Λέξεις Κλειδιά*—*IoT, Live Streaming, Information Systems, NoSQL Database, Apache Kafka, Apache Flink, Apache HBase, Grafana

I. Σκοπος της Εργασιας

Σκοπό της συγκεκριμένης εργασίας αποτελεί η εξοικείωση των φοιτητών με συνδυαστικά open source εργαλεία τα οποία χρησιμοποιούνται για τη δημιουργία live streaming IoT συστημάτων. Αναλυτικότερα, η ομάδα μας δημιούργησε ένα prototype ενός IoT συστήματος με χρήση των εργαλείων Kafka, Flink, HBase, Grafana και της γλώσσας προγραμματισμού python για την παραγωγή δεδομένων.

II. Αναλυση των Βηματων

A. *Περιγραφή του Συστήματος*

Στην εργασία αυτή υλοποιήθηκε ένα πρωτότυπο ενός live streaming IoT (Internet of Things) συστήματος. Αρχικά, μέσω ενός script γραμμένο σε python παράγονται εικονικά δεδομένα τα οποία αντιστοιχούν σε μετρήσεις ενός IoT συστήματος ενός κτηρίου. Στη συνέχεια, αυτά τα δεδομένα αποστέλλονται μέσω του Kafka και υπόκεινται σε επεξεργασία μέσω του Flink. Τόσο τα αρχικά δεδομένα (raw data), όσο και τα επεξεργασμένα (aggregations) αποθηκεύονταν στη NoSQL βάση HBase και από εκεί, μέσω ενός API, επιλεγμένα δεδομένα παρουσιάζονταν στο Dashboard του Grafana.

B. *Οδηγίες για το «Στήσιμο» του Συστήματος*

Για να τρέξετε το σύστημα θα πρέπει να κλωνοποιήσετε (clone) το αποθετήριο (repository) το οποίο βρίσκεται στον σύνδεσμο: https://github.com/Analysi-Pliroforiakon/IoT-Live-Streaming. Έχει χρησιμοποιηθεί το λογισμικό Docker στην εργασία, το οποίο καθιστά δυνατή την εκτέλεση πολλών components με το πάτημα ενός κουμπιού. Ακολουθούν αναλυτικά τα βήματα για τα τρέξουν τα διάφορα τμήματα (components) της εργασίας. Στην περίπτωση εκτέλεσης σε Unix περιβάλλον ακολουθήστε τα εξής βήματα:

1. Αρχικά, μέσα από ένα terminal μεταβείτε στο directory στο οποίο βρίσκεται το αποθετήριο IoT-Live-Streaming και εκτελέστε την εντολή *docker-compose -f docker-compose.yml up*. Με αυτόν τον τρόπο θα ξεκινήσουν όλα τα components που συμπεριλαμβάνονται στο Docker: Kafka [1], Zookeeper [2], HBase [3], Grafana [4] & 1 bridge network.
2. Device Layer: για την παραγωγή των τεχνητών δεδομένων χρειάζεται να έχετε εγκατεστημένη την python και τα requirements που αναγράφονται στο αντίστοιχο αρχείο .txt, να τρέξετε το αρχείο ./deviceLayer/run_producer.py. Αυτό γίνεται μέσω της εντολής *python3 ./deviceLayer/run_producer.py*.
3. Live Streaming Layer: για την μετατροπή των απεσταλμένων δεδομένων μέσω του framework του flink, είναι απαραίτητο να είναι εγκατεστημένη κάποια έκδοση 11 του Java Development Kit (JDK). Η εντολή για την εκτέλεση του live streaming layer είναι *java -jar ./livestreamingLayer/flinkExecutable.jar*. Εναλλακτικά, για την εκτέλεση του flink χρειάζεται να έχετε κατεβασμένο το flink-1.16.0 καθώς και ένα IDE στο οποίο να μπορείτε να εκτελείτε Maven Projects. Συγκεκριμένες οδηγίες για το Eclipse IDE αναγράφονται στο αντίστοιχο README.
4. Data/Storage Layer: για την αποθήκευση των παραγόμενων δεδομένων στη βάση δεδομένων HBase τρέξτε το script *./dataStorageLayer/sync.sh*. Αυτό εισάγει τις απαραίτητες βιβλιοθήκες, δημιουργεί πίνακες στη βάση, τους οποίους επικαιροποιεί (update) κάθε φορά που έρχονται δεδομένα από το flink. Μπορείτε να ελέγξετε την κατάσταση της βάσης από την γραφική διεπαφή χρήστη (Graphical User Interface - GUI) η οποία βρίσκεται στη διεύθυνση localhost:16010.

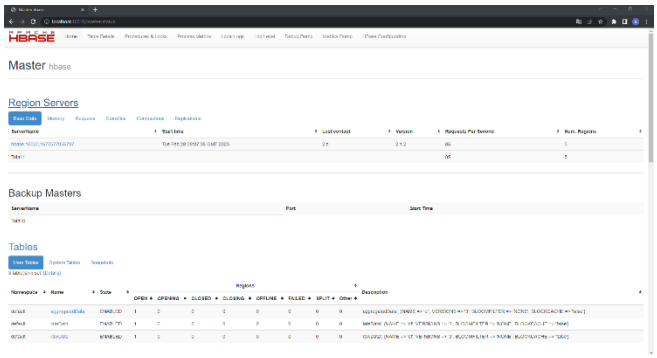

Fig. 1. Screenshot από το GUI της HBase που βρίσκεται στη διεύθυνση localhost:16010.

5. Presentation Layer: για την έναρξη του WebSocket Server μεταβείτε στο directory ./dataStorageLayer/websockets_server/ και εκτελέστε την εντολή npm install. Αφού,

ολοκληρωθεί η εγκατάσταση εκτελέστε την εντολή npm run start.

Για την προβολή των διαγραμμάτων στα οποία απεικονίζονται τα δεδομένα συνδεθείτε στην GUI του Grafana η οποία βρίσκεται στη διεύθυνση localhost:3000. Αρχικά, θα σας ζητηθεί να εισάγετε όνομα χρήστη και κωδικό πρόσβασης. Και στα δύο πεδία εισάγετε τη λέξη «admin» όπως φαίνεται και στο παρακάτω screenshot:

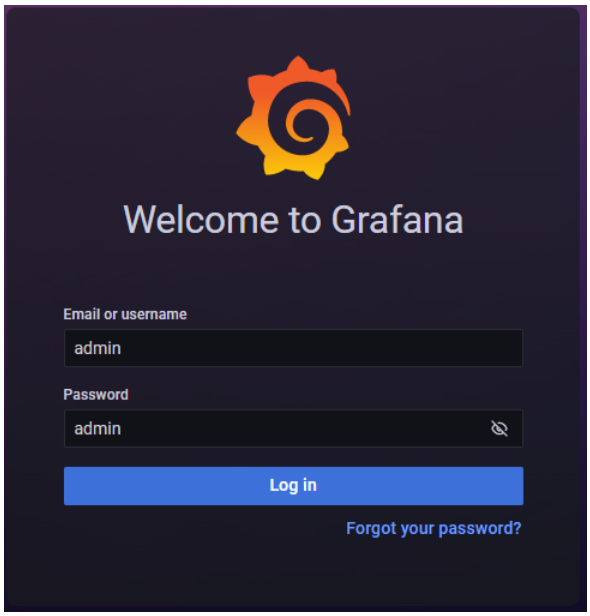

Fig. 2. Screenshot από το GUI του Grafana, στο οποίο απεικονίζονται το username και το password για την είσοδο.

Για τη διασύνδεση του WebSocket Server με το Grafana τρέξτε το script *./presentationLayer/import_data_sources.sh*.

Στο side bar αριστερά στην οθόνη του Grafana επιλέξτε Dashboards > Import και μετά επιλέξτε το αρχείο ./presentationLayer/dashboards/main_dashboard.json.

C. *Γρήγορη Εκτέλεση*

Έχει υλοποιηθεί και τρόπος εκτέλεσης όλων των λειτουργιών του συστήματος μέσω ενός bash script, το οποίο εκτελεί παράλληλα τις εντολές των παραπάνω βημάτων, αλλά δεν συνίσταται η χρήση του, καθώς ο τερματισμός των διεργασιών πρέπει να γίνει χειροκίνητα.

Να σημειωθεί ότι το σύστημα έχει αναπτυχθεί σε περιβάλλον Unix και ενδέχεται η λειτουργία του σε άλλο λειτουργικό σύστημα να μην είναι η αναμενόμενη.

III. Υποδομη και Λογισμικο

Σε αυτήν την ενότητα θα περιγραφούν αναλυτικά η υποδομή και το λογισμικό (software) όλων των components του συστήματος. Στο παρακάτω διάγραμμα ακολουθίας (Sequence Diagram) φαίνεται η διαδρομή που ακολουθούν τα δεδομένα μεταξύ των διάφορων components από την παραγωγή μέχρι την παρουσίαση:

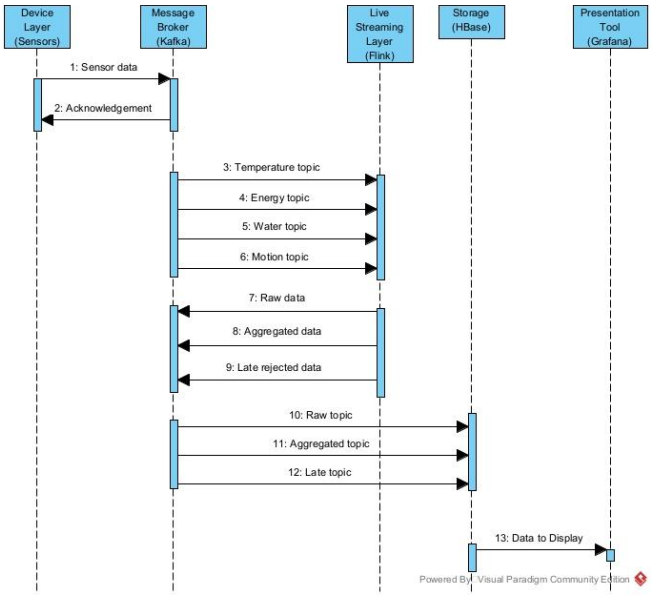

Fig. 3. Sequence Diagram του συστήματος.

Το παρακάτω σχήμα αποτελεί μία συνολική σχηματική απεικόνιση των τεχνολογιών που χρησιμοποιήθηκαν για το σύστημα της παρούσας εργασίας:

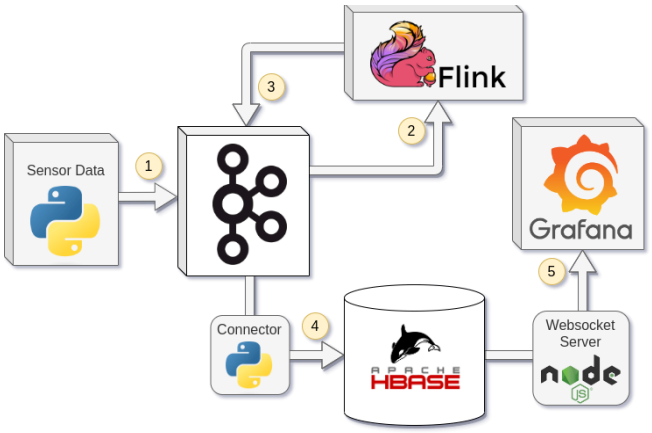

Fig. 4. Σχηματική αναπαράσταση του συστήματος

Εν συντομία, τα δεδομένα των αισθητήρων παράγονται μέσω της python. Στη συνέχεια, αποστέλλονται στον Message Broker, Kafka, και από εκεί στο Live Streaming Processing Framework, Flink, για επεξεργασία. Τα επεξεργασμένα δεδομένα επιστρέφουν στο Kafka από εκεί μέσω της python καταχωρούνται στη βάση δεδομένων, HBase, και από εκεί μέσω ενός Websocket Server/API υλοποιημένο σε nodejs/javascript αποστέλλονται στο λογισμικό οπτικοποίησης Grafana.

A. *Device Layer*

Το στρώμα επιπέδου συσκευής (device layer) περιλαμβάνει τη διαδικασία συλλογής των δεδομένων από τους αισθητήρες. Επειδή το δικό μας σύστημα δεν είναι πραγματικό σύστημα IoT, αλλά αποτελεί πρωτότυπο, χρησιμοποιήθηκε η γλώσσα προγραμματισμού python για την παραγωγή εικονικών δεδομένων. Οι αισθητήρες μπορούν να χωριστούν σε τρεις κατηγορίες με βάση τη συχνότητα λήψης μετρήσεων:

- Οι αισθητήρες δεκαπενταλέπτου, οι οποίοι προσθέτουν μία μέτρηση κάθε 15 λεπτά.

- Οι ημερήσιοι αισθητήρες, οι οποίοι αφορούν το σύνολο των μετρήσεων αθροιστικά για μία μέρα.
- Ο αισθητήρας ανίχνευσης κίνησης, ο οποίος ανιχνεύει κίνηση σε τυχαίες χρονικές στιγμές μέσα στη μέρα.

Προκειμένου τα παραγόμενα δεδομένα να καταστούν ρεαλιστικά, για τους αισθητήρες δεκαπενταλέπτου και ημέρας χρησιμοποιήθηκαν οι βιβλιοθήκες numpy και random της python. Στον παρακάτω πίνακα, φαίνονται οι μονάδες μέτρησης των μετρήσεων κάθε αισθητήρα.

Αισθητήρας	Μονάδα Μέτρησης
TH1	$°C$
TH2	$°C$
HVAC1	Wh
HVAC2	Wh
MiAC1	Wh
MiAC2	Wh
Etot	Wh
W1	lt
Wtot	lt

Fig. 5. Πίνακας μονάδων μέτρησης κάθε αισθητήρα

Για τους αισθητήρες θερμοκρασίας (TH1, TH2) χρησιμοποιούνται οι μέσες τιμές θερμοκρασίας της Αθήνας ανά μήνα. Για μία δεδομένη μέρα, χρησιμοποιείται η μέση τιμή του αντίστοιχου μήνα και με offset, τα οποία ακολουθούν ομοιόμορφη κατανομή, υπολογίζεται η μέγιστη και η ελάχιστη θερμοκρασία της ημέρας. Η μέγιστη τιμή αντιστοιχίζεται στις τέσσερις ακριβώς το απόγευμα (16:00) και η ελάχιστη στις έξι το πρωί (06:00) και παράγονται οι αντίστοιχες μετρήσεις για αυτές τις ώρες. Όλες τις ενδιάμεσες χρονικές στιγμές παράγονται μετρήσεις οι οποίες ακολουθούν μία ημιτονοειδή συνάρτηση μεταξύ των δύο ακραίων τιμών στην οποία προστίθεται επιπλέον θόρυβος.

Οι αισθητήρες ενέργειας (HVAC1, HVAC2, MiAC1, MiAC2, Etot) υπολογίζονται συναρτήσει της θερμοκρασίας και της εκάστοτε χρονικής στιγμής. Οι αισθητήρες HVAC1 και HVAC2 μετρούν μεγαλύτερη κατανάλωση ενέργειας όταν η θερμοκρασία είναι εκτός του διαστήματος $16 - 25°C$. Οι αισθητήρες MiAC1, MiAC2 παρουσιάζουν διαφορετική κατανομή μετρήσεων για διαφορετικές χρονικές στιγμές τη μέρας. Ο Etot παρουσιάζει τη συνολική κατανάλωση ενέργειας από την πρώτη μέρα λειτουργίας του συστήματος και αυξάνεται κάθε μέρα κατά $2600 \times 24 \pm 1000\ Wh$.

Για τον αισθητήρα ανίχνευσης κίνησης χρησιμοποιήθηκε οι βιβλιοθήκη random, ως γεννήτρια των τυχαίων χρονικών στιγμών. Για τον καθορισμό της χρονικής συχνότητας παραγωγής των δεδομένων χρησιμοποιήθηκε η βιβλιοθήκη time.

Οι αισθητήρες νερού (W1, Wtot) χρησιμοποιεί μίγμα κατανομών Gauss για την παραγωγή τυχαίων τιμών κατανάλωσης νερού W1 και συνολικής αύξησης του Wtot.

Οι ετεροχρονισμένες (late) μετρήσεις του αισθητήρα W1 επιλέχθηκε να έχουν την ίδια τιμή με αυτήν που είχε ο αισθητήρας την στιγμή που αφορά την μέτρηση και αντιστοιχεί στο timestamp τους. Για να επιτευχθεί αυτό, διατηρείται το ιστορικό του W1 έως και δέκα μέρες πίσω. Για τις ετεροχρονισμένες μετρήσεις που αφορούν σε χρόνο εκτός του διαστήματος λειτουργίας του συστήματος χρησιμοποιήθηκαν τυχαίες τιμές.

Τα δεδομένα όλων των αισθητήρων μπορούν να παράγονται όλα μαζί από το πρόγραμμα run_producer.py το οποίο εκτελεί παράλληλα τις διεργασίες όλων των αισθητήρων.

Συγκεκριμένα, το run_producer.py ξεκινάει από την ημερομηνία 01-01-2020 και ώρα 00:00 και προχωράει ένα τέταρτο της ώρας ανά δευτερόλεπτο πραγματικού χρόνου. Ο χρήστης θα μπορούσε να δίνει ως παράμετρο και διαφορετικό χρονικό διάστημα να αντιστοιχεί στο ένα τέταρτο. Με την default λειτουργία, οι αισθητήρες δεκαπενταλέπτου παράγουν μία τιμή ανά δευτερόλεπτο, οι ημερήσιοι παράγουν μία τιμή ανά 96 δευτερόλεπτα – όταν δηλαδή η ώρα του script είναι 00:00 – ενώ ο αισθητήρας ανίχνευσης κίνησης παράγει κατά μέσο όρο 4 με 5 μοναδιαίες τιμές ανά ημέρα σε τυχαίες στιγμές, όχι απαραίτητα στα καθορισμένα τέταρτα στα οποία παράγουν οι αισθητήρες δεκαπενταλέπτου.

Τα δεδομένα κάθε μέτρησης αποστέλλονται στο Messaging Broker Layer (Kafka) ως απλό string της μορφής "sensor | datetime | value", όπου sensor είναι το όνομα του αισθητήρα (π.χ. MiAC1), datetime είναι η ημερομηνία και ώρα στην μορφή "YY-MM-DD hh:mm" και value είναι η τιμή της μέτρησης.

B. Messaging Broker Layer

Το Messaging Broker Layer αποτελεί το ενδιάμεσο σύστημα με το οποίο γίνεται η διαχείριση events-μηνυμάτων που στέλνονται από και προς διαφορετικά συστήματα. Στην εργασία μας, το σύστημα αναλαμβάνει την διαχείριση δύο κατηγοριών μηνυμάτων. Η πρώτη κατηγορία αφορά τα μηνύματα που έρχονται από τις μετρήσεις των αισθητήρων και καταλήγουν στο Live Streaming Layer. Η δεύτερη κατηγορία αφορά τα επεξεργασμένα αποτελέσματα του Live Streaming Layer και την παροχή τους στο Data/Storage Layer.

Ως message broker χρησιμοποιήθηκε το Apache Kafka. Το Kafka αποτελεί ένα κατανεμημένο σύστημα αποθήκευσης και παροχής μηνυμάτων (event store), ενώ επίσης παρέχει δυνατότητες επεξεργασίας ροών δεδομένων (stream-processing). Στην εργασία μας το Kafka χρησιμοποιείται μόνο για την λειτουργία της προσωρινής αποθήκευσης μηνυμάτων, καθώς η λειτουργία της επεξεργασίας υλοποιείται από το Apache Flink του Live Streaming Layer. Το Kafka αποτελείται από ένα σύνολο από Brokers στους οποίους αποθηκεύονται τα μηνύματα. Τα μηνύματα είναι οργανωμένα σε topics και το κάθε topic διαμερίζεται σε partitions. Επίσης το Kafka περιέχει και ένα Zookeeper quorum το οποίο συντονίζει την λειτουργία των Brokers.

Προκείμενου να καταστεί δυνατή η λειτουργία του Kafka, χρησιμοποιήθηκε το Docker το οποίο αποτελεί ένα σύνολο προϊόντων Platform as a Service που χρησιμοποιεί virtualization επιπέδου λειτουργικού συστήματος για την κατανομή λογισμικού σε containers. [5]

Στην εργασία μας δημιουργήσαμε ένα cluster που αποτελείται από τρείς Brokers και έναν Zookeeper. Ορίσαμε

τον παράγοντα αντιγραφής των Kafka logs ίσο με 2 ώστε να υπάρχουν δύο αντίγραφα των δεδομένων στους Brokers, και έτσι η λειτουργία του συστήματος να μπορεί να συνεχίσει ακόμα και αν αποτύχει ένας από τους Brokers. Ορίσαμε επίσης και ένα όριο των 25 MB για τον μέγεθος των logs, και έτσι τα μηνύματα δεν θα παραμένουν αποθηκευμένα για πάντα, αλλά τα αρχαιότερα θα διαγράφονται.

Στο σύστημα του Kafka, οι διεργασίες που στέλνουν μηνύματα στο σύστημα ονομάζονται producers και οι διεργασίες που διαβάζουν μηνύματα ονομάζονται consumers. Τα δεδομένα επιστρέφονται με την σειρά που στέλνονται, και το σύστημα διατηρεί πληροφορία για τα offsets, δηλαδή μέχρι ποιο σημείο του topic έχει ήδη διαβάσει κάθε διεργασία.

Κρίθηκε σκόπιμο τα δεδομένα των αισθητήρων να χωριστούν σε topics ανάλογα με το είδος του αντικειμένου μέτρησης του κάθε αισθητήρα. Ο λόγος είναι ότι ο επιλεγμένος τρόπος διαχωρισμού καθιστά εύκολη την υλοποίηση των ζητούμενων aggregations στο Flink. Αναλυτικότερα, τα δεδομένα χωρίστηκαν σε τέσσερα topics, όπως φαίνεται στον παρακάτω πίνακα:

Topic	Αισθητήρες
temperature	TH1, TH2
energy	HVAC1, HVAC2, MiAC1, MiAC2, Etot
motion	Mov1
water	W1, Wtot

Fig. 6. Πίνακας αντιστοίχισης αισθητήρων στα topics του Kafka

Για την υλοποίηση του κώδικα δημιουργίας του producer των δεδομένων των αισθητήρων και την επικοινωνία μεταξύ του Device Layer και του Messaging Broker Layer χρησιμοποιήθηκε το πακέτο της python confluent-kafka. Για την αποστολή ενός μηνύματος ορίζουμε το topic για το οποίο προορίζεται, και ένα ζευγάρι key-value το οποίο αποτελεί τα δεδομένα του μηνύματος. Στα partitions του Kafka συγκεντρώνονται τα μηνύματα με την ίδια τιμή του key, και για αυτό επιλέξαμε ως key το όνομα των αισθητήρων. [6, 7]

C. Live Streaming Layer

Το στρώμα ζωντανής μετάδοσης (live streaming layer) περιλαμβάνει τα components τα οποία είναι υπεύθυνα για τη ζωντανή επεξεργασία των δεδομένων που αντλούνται από το messaging broker layer.

Συγκεκριμένα, χρησιμοποιήθηκε το framework live επεξεργασίας Apache Flink, οι λειτουργίες του οποίου υλοποιήθηκαν σε γλώσσα Java. Δεδομένου ότι στην συγκεκριμένη εφαρμογή χειριζόμαστε μη-φραγμένα (unbounded) δεδομένα, το flink παρείχε ιδιαίτερα χρήσιμες λειτουργίες.

Το Flink περιλαμβάνει έτοιμους (builtin) connectors για τη σύνδεση με το Kafka, μέσω της κλάσης KafkaSource. Για κάθε topic του Kafka το Flink λαμβάνει τα δεδομένα του topic σε μορφή data stream, δηλαδή μίας συνεχόμενης «ροής» δεδομένων, την οποία μπορεί να επεξεργάζεται και να χρησιμοποιεί για συσσωματώσεις (aggregations).

Στα εισερχόμενα data streams πραγματοποιούνται windowed aggregations, δηλαδή σε κάθε aggregation χρησιμοποιούνται τα δεδομένα τα οποία εμπίπτουν σε ένα χρονικό παράθυρο.

Σε αυτό το σημείο, να σημειωθεί ότι τα windowed aggregations πραγματοποιήθηκαν με βάση τον χρόνο γεγονότος (event time) και όχι τον χρόνο επεξεργασίας (processing time). Ο χρόνος επεξεργασίας είναι ο πραγματικός χρόνος, όπως μετρείται από το ρολόι του μηχανήματος στο οποίο τρέχει ένα πρόγραμμα. Ένα processing time window περιλαμβάνει όλα τα δεδομένα τα οποία κατέφθασαν στο Flink εντός ενός καθορισμένου χρονικού διαστήματος. Από την άλλη, ο χρόνος γεγονότος αφορά τον χρόνο στον οποίο παρήχθησαν τα δεδομένα. Ένα event time window θα χρησιμοποιήσει δεδομένα τα οποία αφορούν γεγονότα τα οποία πραγματοποιήθηκαν εντός του καθορισμένου χρονικού διαστήματος του παραθύρου. [8]

Όλα τα aggregations της παρούσας εργασίας πραγματοποιήθηκαν σε event time windows, δηλαδή η επεξεργασία των δεδομένων έγινε με βάση τη χρονική στιγμή μέτρησης κάθε αισθητήρα, η οποία εξάγεται στο Flink από τα timestamps τα οποία αποστέλλονται μέσω του Kafka. Η χρήση χρόνου γεγονότος παρέχει πολλά πλεονεκτήματα, όπως καθορισμός συγκεκριμένων δεδομένων για επεξεργασία ανεξαρτήτως της χρονικής στιγμής άφιξής τους, ενώ αποτελεί μονόδρομο για την διαχείριση δεδομένων εκτός σειράς (out-of-order) και ετεροχρονισμένων δεδομένων.

Παρακάτω παρουσιάζονται όλα τα aggregations που πραγματοποιήθηκαν:

- AggDay[x]: όπου x = {TH1, TH2, HVAC1, HVAC2, MiAC1, MiAC2, W1}, Tumbling Event Time Window διάρκειας μίας μέρας, υπολογίζεται ο ημερήσιος μέσος όρος για τα δεδομένα θερμοκρασίας (TH1, TH2) και το ημερήσιο άθροισμα για τα υπόλοιπα.
- AggDayDiff[y]: όπου y = {Etot, Wtot}, Sliding Event Time Window διάρκειας δύο ημερών ανά μία μέρα, υπολογίζεται η διαφορά των δύο τελευταίων μετρήσεων των αθροιστικών αισθητήρων.
- AggDayRest[y]: όπου y = {Etot, Wtot}, Tumbling Event Time Window διάρκειας μίας μέρας, υπολογίζεται η διαφορά του AggDayRest[y] με το άθροισμα όλων των AggDay[x] του αντίστοιχου topic.
- AggDayMov[Mov1], Tumbling Event Time Window διάρκειας μίας μέρας, υπολογίζεται το πλήθος των ανιχνεύσεων κίνησης κατά τη διάρκεια μίας μέρας.

Όσον αφορά τα ετεροχρονισμένα δεδομένα (late events), το Flink διαθέτει μεθόδους για τον αποτελεσματικό χειρισμό τους. Ετεροχρονισμένα δεδομένα υπάρχουν μόνο για τον αισθητήρα νερού W1. Τα δύο μέρες πίσω δεδομένα είναι αποδεκτά (accepted late events) και υπολογίζονται κανονικά στα aggregations του topic water, ενώ τα δέκα μέρες πίσω απορρίπτονται (rejected late events) και οδηγούνται σε διαφορετικό data stream.

Για όλα τα δεδομένα στα οποία θα γίνει aggregation ακολουθείται η builtin στρατηγική watermark *forBoundedOutOfOrderness(Duration.ofDays(2))* του Flink κατά την οποία είναι αποδεκτή η εκτός σειράς άφιξη, όταν το timestamp του δεδομένου προς εισαγωγή είναι έως και δύο μέρες πίσω σε σχέση με το μέγιστο timestamp που έχει ληφθεί στο αντίστοιχο datastream. [9]

Το Flink αποδείχθηκε κατάλληλο για την αξιοποίηση του επιλεγμένου τρόπου διαχωρισμού των δεδομένων των αισθητήρων στα topics του Kafka. Ο λόγος είναι ότι για τα δεδομένα δεκαπενταλέπτου τα οποία προέρχονται από το ίδιο είδος αισθητήρα στον Kafka χρειάζεται να γίνει το ίδιο είδος aggregation, ενώ τα aggregations τύπου AggDayRest αξιοποιούν όλα τα aggregations που έχουν γίνει για τους αισθητήρες ενός τύπου. Αν τα δεδομένα αισθητήρων ενός τύπου έρχονταν από διαφορετικά topics, όπως για παράδειγμα αν οι αισθητήρες μέτρησης κατανάλωσης ενέργειας δεκαπενταλέπτου ήταν σε διαφορετικό topic από τους αισθητήρες συνολικής κατανάλωσης ενέργειας, θα δυσχέραινε πολύ η υλοποίηση στο Flink.

Αυτό φαίνεται χαρακτηριστικά στο παρακάτω παράδειγμα κώδικα:

```
48  SingleOutputStreamOperator<ourTuple> newStream = dataStream
49          .filter(value -> !value.sensor.contains("tot"))
50          .keyBy(value -> value.sensor)
51          .window(TumblingEventTimeWindows.of(Time.days(1), Time.minutes(-119)))
52          .sideOutputLateData(lateOutputTag)
53          .aggregate(quarterAggregateFunction);
54
55  SingleOutputStreamOperator<ourTuple> totStream = dataStream
56          .filter(value -> value.sensor.contains("tot"))
57          .keyBy(value -> value.sensor)
58          .window(SlidingEventTimeWindows.of(Time.days(2), Time.days(1), Time.minutes(-119)))
59          .aggregate(dailyAggregateFunction);
```

Fig. 7. Κομμάτι από τον κώδικα του Live Streaming Layer. Βρίσκεται στο αρχείο *.\liveStreamingLayer\src\main\java\org\flink\DataStreamClass.java*

Ο παραπάνω κώδικας υλοποιεί όλα τα aggregations τύπου AggDay και AggDayDiff, καθώς και το AggDayMov. Συγκεκριμένα, το αντικείμενο dataStream αποτελεί τη ροή δεδομένων όλων των αισθητήρων ενός topic. Η μέθοδος filter χρησιμοποιείται για τον διαχωρισμό των δεδομένων ενός topic σε δεκαπενταλέπτου και ημερήσια. Η μέθοδος keyBy χρησιμοποιείται για τον διαχωρισμό των δεδομένων με βάση τον αισθητήρα παραγωγής τους. Η μέθοδος window εφαρμόζει το επιθυμητό event time παράθυρο, ενώ η sideOutputLateData χρησιμοποιείται για την παρακράτηση των rejected late events σε ξεχωριστό stream. Τέλος, η μέθοδος aggregate δέχεται ως όρισμα την κατάλληλη συνάρτηση συσσωμάτωσης η οποία έχει οριστεί νωρίτερα για κάθε topic και για κάθε είδος αισθητήρα με βάση τη χρονική διάρκεια.

Τα τελικά data streams του Flink καταλήγουν μέσω κατάλληλων sinks σε ξεχωριστά topics στο Kafka. Χρησιμοποιήθηκε η κλάση KafkaSink του DataStream API του Flink. Συγκεκριμένα, στο topic «raw» του Kafka καταλήγουν όλα τα δεδομένα των αισθητήρων (κανονικά, late processed events, late rejected events), στο topic «aggregated» καταλήγουν όλα τα aggregations και στο topic «late» μόνο τα απορριφθέντα ετεροχρονισμένα δεδομένα (των δέκα ημερών πίσω).

D. Data/Storage Layer

Το Data/Storage layer είναι το επίπεδο που περιέχει το σύνολο των δεδομένων εξόδου του live streaming layer και ταυτόχρονα παρέχει μία διεπαφή για την προσπέλαση των δεδομένων αυτών. Για την αποθήκευση των δεδομένων χρησιμοποιήθηκε η βάση Apache HBase. Η HBase αποτελεί μία μη-σχεσιακή βάση δεδομένων που ακολουθεί το μοντέλο του συστήματος Bigtable της Google. Τα δεδομένα της HBase οργανώνονται σε πίνακες, όπου κάθε γραμμή του πίνακα έχει ένα μοναδικό κλειδί και μία σειρά από οικογένειες στηλών (column families) στις οποίες μπορούν να αποθηκεύονται δεδομένα με οποιαδήποτε αναγνωριστικά-στήλες (column qualifiers). Η HBase υλοποιεί επίσης σύστημα εκδόσεων βάσει χρονοσφραγίδων (timestamps). [10]

Το βασικότερο στοιχείο του σχήματος της βάσης είναι η δομή του κλειδιού των γραμμών. Αυτό συμβαίνει, επειδή η βάση αποθηκεύει τις γραμμές ταξινομημένες ως προς το κλειδί και, άρα, η δομή του καθορίζει και την ταχύτητα των αναγνώσεων και των εγγραφών. Επειδή στην περίπτωσή μας χειριζόμαστε δεδομένα πραγματικού χρόνου, επιλέξαμε το κλειδί να έχει την δομή “timestamp + measurement” όπου timestamp είναι η χρονοσφραγίδα των δεδομένων όπως αυτά παράγονται, και measurement είναι η μέτρηση που αφορά η γραμμή (μπορεί να είναι πχ TH1 ή AggDayDiff[Etot]). Αυτό μας δίνει την δυνατότητα να λαμβάνουμε και να αποθηκεύουμε αποδοτικά τα πιο πρόσφατα δεδομένα. Επιπλέον, μας δίνεται και η δυνατότητα να αναζητούμε γραμμές με βάση την μέτρηση (το οποίο όμως εμείς δεν χρειάστηκε να χρησιμοποιήσουμε).

Στα δεδομένα που διαχειριζόμαστε υπάρχουν καταχωρήσεις με το ίδιο row key (ζεύγος timestamp-measurement) που αντιστοιχούν στα κανονικά και στα late δεδομένα, όπως αναφέρεται στην ενότητα για το Device Layer. Η HBase διατηρεί διαφορετικές versions δεδομένων τα οποία έχουν το ίδιο row key. Ο ελάχιστος αριθμός versions για το ίδιο row key στη βάση ορίστηκε ίσος με 3, καθώς είναι δυνατό για το ίδιο timestamp να παραχθεί έως και τρεις φορές μέτρηση (μία στην ώρα της, μία δύο ημέρες αργότερα και μία δέκα ημέρες αργότερα).

Τα δεδομένα αποθηκεύονται σε τρεις διαφορετικούς πίνακες στη βάση. Στον πίνακα rawData αποθηκεύονται τα ακατέργαστα δεδομένα, όπως παράγονται από το device layer συμπεριλαμβανομένων όλων των late events («raw» topic), στον πίνακα aggregatedData αποθηκεύονται όλα τα aggregations που πραγματοποιούνται στο Flink («aggregated» topic) και στον πίνακα lateData, αποθηκεύονται μόνο τα απορριφθέντα ετεροχρονισμένα δεδομένα («late» topic).

Η HBase αποτελεί μια κατανεμημένη βάση δεδομένων, καθώς χρησιμοποιεί το Apache Hadoop και HDFS για να αποθηκεύει τα δεδομένα. Για τον σκοπό της άσκησης, όμως, η βάση δεδομένων λειτουργεί σε standalone mode το οποίο σημαίνει ότι όλες οι απαραίτητες διεργασίες (Master, RegionServers, ZooKeeper) τρέχουν μαζί σε μία διεργασία και σε ένα docker container και τα δεδομένα αποθηκεύονται στο σύστημα αρχείων της μηχανής αντί για το HDFS.

Τα δεδομένα της βάσης τα καθιστούμε προσβάσιμα στο Presentation Layer μέσω ενός Websocket server ο οποίος υλοποιήθηκε χρησιμοποιώντας το περιβάλλον του Node.js. Ο εξυπηρετητής αναμένει σύνδεση με την διεργασία του Presentation Layer (Grafana) και όταν έχει πλέον εγκαθιδρυθεί η σύνδεση ο εξυπηρετητής ρωτά περιοδικά την βάση για να λάβει δεδομένα τα οποία είναι νεότερα από αυτά που έχει ήδη λάβει. Όταν λάβει νέα δεδομένα, διαμορφώνει ένα μήνυμα JSON το οποίο και αποστέλλει στην υπηρεσία του Presentation Layer.[11]

Η αποθήκευση των επεξεργασμένων δεδομένων στη βάση γίνεται μέσω connectors από Kafka σε HBase. Συγκεκριμένα, χρησιμοποιήθηκε το thrift API της HBase, καθώς και το πακέτο HappyBase, το οποίο παρέχει εντολές για χρήση της HBase μέσω python. [12]

E. Presentation Layer

Για την παρουσίαση και την οπτικοποίηση των δεδομένων χρησιμοποιούμε την εφαρμογή Grafana. Το Grafana παρέχει δυνατότητες να λαμβάνει δεδομένα από μια πληθώρα πηγών μέσω του συστήματος των plug-ins που διαθέτει. Επειδή όμως δεν υπήρχε διαθέσιμο plug-in που να λαμβάνει δεδομένα κάνοντας ερωτήματα κατευθείαν στην βάση δεδομένων, χρησιμοποιήσαμε ένα plug-in που μας επιτρέπει να χρησιμοποιούμε τον Websockets server που περιγράψαμε στην ενότητα του Data/Storage Layer σαν πηγή δεδομένων. [13]

Προκειμένου να παρουσιάσουμε τα δεδομένα, για το κάθε Panel του Dashboard που δημιουργήσαμε, αρχικά αναγνωρίζουμε τα πεδία του JSON μηνύματος που λαμβάνουμε από τον server, και στην συνέχεια εφαρμόζουμε τα κατάλληλα φίλτρα με τα οποία φιλτράρουμε μόνο τα δεδομένα που αφορούν το Panel (π.χ. μόνο τα δεδομένα των μετρήσεων του αισθητήρα TH2). Παρακάτω βλέπουμε ένα παράδειγμά του Dashboard με δεδομένα:

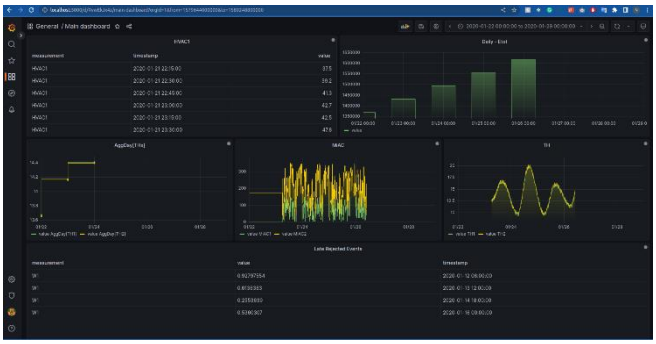

Fig. 8. Screenshot από το dashboard του Grafana

Στο παραπάνω dashboard έχουν υλοποιηθεί έξι panels. Πάνω αριστερά φαίνεται ένα table το οποίο περιέχει τις μετρήσεις του αισθητήρα HVAC1. Πάνω δεξιά παρουσιάζονται σε bar chart οι μετρήσεις του ημερήσιου αισθητήρα μέτρησης αθροιστικής ενέργειας Etot. Αριστερά στη μεσαία σειρά παρουσιάζονται οι τιμές του AggDay για τους αισθητήρες θερμοκρασίας TH1 και TH2 δηλαδή οι μέσες μετρήσεις θερμοκρασίας κάθε αισθητήρα ανά ημέρα. Στο κέντρο και αριστερά φαίνονται σε chart οι μετρήσεις δεκαπεντάλεπτου για τους αισθητήρες MiAC και TH αντίστοιχα. Τέλος, στο κάτω μέρος της οθόνης παρουσιάζονται σε table τα late rejected δεδομένα που αφορούν σε μετρήσεις του αισθητήρα W1.

Πρέπει να σημειώσουμε ότι το Grafana δεν υποστηρίζει δυναμικό ορισμό των ορίων του άξονα του χρόνου στα χρονικά διαγράμματα. Επομένως για να βλέπουμε τα νέα δεδομένα πρέπει να ορίζουμε χειροκίνητα τα επιθυμητά όρια του άξονα στα σχετικά Panels. Αυτό συμβαίνει επειδή τα δεδομένα που παράγουμε είναι δεδομένα προσομοίωσης και όχι πραγματικού χρόνου, στην οποία περίπτωση θα μπορούσαμε να βλέπουμε διαρκώς τα δεδομένα π.χ. των τελευταίων 24 ωρών.

IV. Αποτελεσματα και Συμπερασματα

Η εργασία είχε ως αποτέλεσμα τη δημιουργία ενός ολοκληρωμένου pipeline live επεξεργασίας και απεικόνισης δεδομένων IoT, όπως και σε μία πραγματική εφαρμογή.

Τόσο το συνολικό pipeline όσο και οι επιμέρους τεχνολογίες που το αποτελούν είναι σχεδιασμένες για να χρησιμοποιούνται για την ανάπτυξη σύγχρονων κλιμακώσιμων (scalable) IoT εφαρμογών και τον χειρισμό «μεγάλων δεδομένων» (big data). Το Apache Kafka μπορεί να χρησιμοποιηθεί για την ασφαλή και αποτελεσματική μετάδοση μηνυμάτων σε μεγάλες εφαρμογές. Το framework του Flink παρέχει έναν εύκολο τρόπο για live επεξεργασία, φιλτράρισμα και συσσωμάτωση (aggregation) των δεδομένων. Ειδικότερα, παρέχει εύχρηστες λειτουργίες για επεξεργασία βασιζόμενη στον χρόνο γεγονότος (event time), το οποίο καθίσταται ιδιαίτερα χρήσιμο για πραγματικά IoT συστήματα, στα οποία τα δεδομένα μπορεί να καταφθάνουν εκτός σειράς (out-of-order) και σε χρόνο ο οποίος δεν αντιστοιχεί με τον χρόνο παραγωγής τους. Η NoSQL βάση HBase είναι καταλληλότερη για τη διαχείριση δεδομένων μεγάλης κλίμακας σε σχέση με τις παραδοσιακές σχεσιακές βάσεις δεδομένων (relational databases). Ένα πλεονέκτημά της είναι η δυνατότητα για την αποθήκευση δεδομένων με το ίδιο key διατηρώντας πολλαπλές εκδόσεις (version) για μία καταχώρηση στη βάση. Αυτό είναι σημαντικό για ένα σύστημα το οποίο διαχειρίζεται ετεροχρονισμένα δεδομένα στο οποίο ενδέχεται να υπάρχουν μετρήσεις με ίδια timestamps. Τέλος, το Grafana ενδείκνυται για την οπτικοποίηση σε πραγματικό χρόνο, το οποίο μπορεί να είναι χρήσιμο για την ερμηνεία των καταγραφών των αισθητήρων ενός IoT συστήματος. Να σημειωθεί ότι το Docker καθιστά δυνατή την εκτέλεση πολλών κομματιών των παραπάνω συστημάτων με το πάτημα ενός κουμπιού, περιορίζοντας δραστικά τις απαιτήσεις για χειροκίνητο configuration των χρησιμοποιούμενων τεχνολογιών.

Κάποιοι περιορισμοί των παραπάνω συστημάτων είναι οι εξής:

- Τα documentations του Flink και της HBase είχαν περιορισμένα παραδείγματα και ήταν δύσχρηστα για αρχάριους στις συγκεκριμένες τεχνολογίες.
- Υπήρχε ασυμβατότητα μεταξύ του Flink και της HBase με αποτέλεσμα να χρησιμοποιηθεί το Kafka για την αποστολή των δεδομένων στη βάση.
- Ακόμα και μεταξύ της HBase και του Kafka το οποίο παρέχει μέσω του Kafka Connect δυνατότητα διασύνδεσης με πληθώρα συστημάτων δεν κατέστη δυνατή η χρήση κάποιου sink connector. Εν τέλει, αναπτύχθηκε ad hoc script.
- Το Grafana δεν παρέχει την δυνατότητα αυτόματης εστίασης στα τελευταία δεδομένα που καταγράφει, με αποτέλεσμα να μην καθιστά δυνατή την απεικόνιση δεδομένων σε ζωντανό χρόνο, όταν αυτά αφορούν χρονικές στιγμές του παρελθόντος. Παρέχει, ωστόσο, αυτή τη δυνατότητα για τωρινά timestamps.

Παραπομπες

[1] https://hub.docker.com/r/confluentinc/cp-kafka

[2] https://hub.docker.com/r/confluentinc/cp-zookeeper

[3] https://github.com/dajobe/hbase-docker

[4] https://grafana.com/docs/grafana/latest/setup-grafana/installation/docker/

[5] https://en.wikipedia.org/wiki/Docker_(software)

[6] https://docs.confluent.io/kafka/introduction.html#producers

[7] https://docs.confluent.io/platform/current/clients/confluent-kafka-python/html/index.html

[8] https://nightlies.apache.org/flink/flink-docs-master/docs/concepts/time/

[9] https://nightlies.apache.org/flink/flink-docs-master/docs/dev/datastream/event-time/built_in/

[10] https://hbase.apache.org/
[11] https://www.npmjs.com/package/websocket
[12] https://happybase.readthedocs.io/en/latest/
[13] https://grafana.com/grafana/plugins/golioth-websocket-datasource/