Αριστοτέλειο Πανεπιστήμιο
Θεσσαλονίκης
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών &
Μηχανικών Υπολογιστών
Τομέας Ηλεκτρονικής και Υπολογιστών

Διπλωματική Εργασία

Αρχιτεκτονική ενίσχυση αγροτικών IoT συστημάτων πραγματικού χρόνου

Επιμέλεια:
Δημήτριος Ντέντας
ΑΕΜ: 10446

Επίβλεψη:
Κωνσταντίνος Παναγιώτου
Γιώργος Σιαχάμης

Θεσσαλονίκη, Ιανουάριος 2026

Περιεχόμενα

1
Εισαγωγή

2

Επισκόπηση της Ερευνητικής Περιοχής

Στη σύγχρονη γεωργία, ένας δυσλειτουργικός αισθητήρας που περνά απαρατήρητος για ώρες μπορεί να οδηγήσει σε αποτυχημένο πότισμα ή καταστροφή καλλιεργειών. Αυτό το ενιαίο σημείο αποτυχίας αναδεικνύει την ανάγκη για αυτόνομα, ανθεκτικά και παρατηρήσιμα συστήματα σε κλίμακα. Η ανάγκη αυτή εναρμονίζεται με τον οδικό χάρτη (Roadmap) προς ανθεκτικά Κυβερνοφυσικά Συστήματα (CPS) που περιγράφεται από τους Ratasich et al. [1], οι οποίοι δίνουν έμφαση στην ανίχνευση ανωμαλιών κατά τη διάρκεια λειτουργίας, στην απομόνωση σφαλμάτων και στην αυτοΐαση σε δυναμικά Internet of Things (IoT) περιβάλλοντα.

Οι συσκευές IoT παράγουν, σε πραγματικό χρόνο, μεγάλο όγκο δεδομένων, είτε σε μορφή χρονοσειράς, είτε ως ιστορικά δεδομένα [2]. Επομένως, η κλίμακα αυτή αποτελεί βασική αιτία εμφάνισης προβλημάτων και αστοχιών. Ωστόσο, για να εξασφαλιστεί η ομαλή λειτουργία των κρίσιμων υποδομών ενός νευραλγικού τομέα, όπως η γεωργία, οφείλουμε ως μηχανικοί να παρέχουμε λύσεις για τον παραγωγό και κατά συνέπεια για τον πολίτη, που εξασφαλίζουν τόσο την ποιότητα, όσο και την αναμενόμενη ποσότητα των αγροτικών προιόντων.

Σε χώρες όπως η Ελλάδα, όπου η οικονομία και η αυτονομία έγκεινται σε μεγάλο βαθμό στη γεωργική παραγωγή, η υστέρηση του γεωργικού τομέα ως προς την ενσωμάτωση τεχνολογικών καινοτομιών έχει

συμβάλει στην αποδυνάμωσή του και στη χαμηλή του συμβολή στην οικονομική ανάπτυξη, όπως επισημαίνουν οι Kyrkilis et al. [3]. Η αξιοποίηση τεχνολογιών πραγματικού χρόνου θα μπορούσε να αποτελέσει κρίσιμο παράγοντα για την αντιστροφή αυτής της τάσης και τη διασφάλιση της βιωσιμότητας και της αποδοτικότητας του πρωτογενούς τομέα.

Σε κρίσιμες εφαρμογές, όπως η αγροδιατροφή, η διατάραξη στη ροή δεδομένων ή στη λήψη των αποφάσεων μπορεί να οδηγήσει σε σπατάλη τροφίμων, οικονομική απώλεια ή επισιτιστική ανασφάλεια. Σύμφωνα με τους Callo και Mansouri [4], η ανθεκτικότητα των παγκόσμιων δικτύων διανομής τροφίμων εξαρτάται από την δυνατότητα των πληροφοριακών συστημάτων να ανταπεξέρχονται σε γεωπολιτικές ή υγειονομικές κρίσεις μέσω μηχανισμών ευελιξίας και προσαρμογής. Αυτό σημαίνει πως σε ότι αφορά σε αντίστοιχα συστήματα, στα οποία βασίζεται ο πληθυσμός (και η οικονομία) μιας χώρας, η αρχιτεκτονική που χρησιμοποιείται θα πρέπει να είναι άρτια, μελετημένη, αλλά και να εξασφαλίζει την ομαλή και συνεχή λειτουργία τους.

Η σύγχρονη ερευνητική κοινότητα έχει μετατοπίσει το ενδιαφέρον της από στατικές προσεγγίσεις παρακολούθησης προς ολοκληρωμένες αρχιτεκτονικές λειτουργικής ανθεκτικότητας, οι οποίες συνδυάζουν συνεχές monitoring, αυτόματη διάγνωση βλαβών και δυναμική ανάκαμψη. Βασισμένα στις αρχές των CPS, τα σύγχρονα αυτά πρότυπα επικεντρώνονται όχι μόνο στην αποτροπή σφαλμάτων, αλλά και στην ικανότητα του συστήματος να απορροφά, να ανακάμπτει και να προσαρμόζεται σε απρόβλεπτες συνθήκες λειτουργίας, με διατήρηση της κρίσιμης απόδοσης. Όπως επισημαίνουν οι Lee et al. [5], η ποσοτικοποίηση της ανθεκτικότητας απαιτεί νέες μετρικές, όπως η καμπύλη ανθεκτικότητας (resilience curve) και το πλαίσιο R4 (Redundancy, Robustness, Resourcefulness, Rapidity), τα οποία μπορούν να ενσωματωθούν ακόμα και σε πραγματικά περιβάλλοντα cloud-native, διασφαλίζοντας ιδιότητες όπως υψηλή διαθεσιμότητα, αυτοθεραπεία και επιχειρησιακή συνέχεια (business continuity).

Βάσει της μελέτης των Sharma et al. [6], αναδεικνύεται η ολοένα αυξανόμενη σημασία της χρήσης IoT τεχνολογιών στον τομέα της γεωργίας ακριβείας, όπου η αξιοποίηση αισθητήρων για την παρακολούθηση παραμέτρων όπως η υγρασία, η θερμοκρασία, η αγωγιμότητα και τα επίπεδα θρεπτικών συστατικών του εδάφους (Αζώτου, Φωσφόρου, Καλίου), επιτρέπει την ακριβή λήψη αποφάσεων σε πραγματικό χρόνο.

Η προσέγγιση αυτή ευθυγραμμίζεται με την ανάγκη για ευέλικτες, αποκεντρωμένες υποδομές λήψης αποφάσεων, οι οποίες υποστηρίζονται από μηχανισμούς edge analytics και cloud ενορχήστρωσης. Στο πλαίσιο

αυτό, η μελέτη των Akhtar et al. [7] τονίζει τη σημασία της ενσωμάτωσης του edge computing για την επεξεργασία δεδομένων από αισθητήρες σε πραγματικό χρόνο, επιτρέποντας την αξιολόγηση του εδάφους και την παρακολούθηση ρύπων με μεγαλύτερη αποτελεσματικότητα. Υποστηρίζει, επιπλέον, πως η ενσωμάτωση τεχνικών μηχανικής μάθησης στην αλυσίδα επεξεργασίας των δεδομένων, όπως η αυτόματη πρόβλεψη της καταλληλότητας του εδάφους για συγκεκριμένες καλλιέργειες, ενισχύει την αξία των IoT συστημάτων και απαιτεί αρχιτεκτονική σχεδίαση που υποστηρίζει δυναμική ανάλυση και διαλειτουργικότητα.

Παρά το γεγονός ότι η τεχνολογία δεν έχει ακόμη ενσωματωθεί πλήρως στον αγροτικό τομέα, παρατηρείται αυξανόμενη τάση υιοθέτησης καινοτόμων λύσεων, ιδιαίτερα στο επίπεδο του αγρού. Μια από τις πιο προσιτές προσεγγίσεις βασίζεται στην αξιοποίηση αισθητήρων (θερμοκρασίας, υγρασίας, φωτός, βροχόπτωσης, ταχύτητας ανέμου) οι οποίοι σχηματίζουν ένα τοπικό δίκτυο συλλογής δεδομένων στον χώρο της καλλιέργιας. Τα δεδομένα που συλλέγονται μεταφέρονται σταδιακά προς υπολογιστικές μονάδες αυξημένης ισχύος, σχηματίζοντας μια πολυεπίπεδη, αποκεντρωμένη υποδομή που επιτρέπει τόσο την τοπική επεξεργασία όσο και την κεντρική αποθήκευση και ανάλυση [8].

Τα ευρήματα αυτά ενισχύουν την άποψη ότι η αρχιτεκτονική ενός ανθεκτικού IoT συστήματος στον αγροδιατροφικό τομέα πρέπει να υποστηρίζει συνεχές monitoring, on-device προεπεξεργασία και ασφαλή, επεκτάσιμη μετάδοση δεδομένων, προκειμένου να επιτευχθεί πλήρης αυτοματοποίηση και ευστάθεια λειτουργίας σε ετερογενή, διασυνδεδεμένα περιβάλλοντα πεδίου.

Με στόχο, λοιπόν, την αρχιτεκτονική ενίσχυση του συστήματος *Nostradamus* προτείνεται, ως απόρροια των παραπάνω μελετών, η ενσωμάτωση μηχανισμών caching, observability και self-healing. Παρότι υπάρχουν επιμέρους τεχνολογίες που προσφέρουν τέτοιες δυνατότητες, η υιοθέτησή τους σε πραγματικές πολυεπίπεδες IoT πλατφόρμες συνοδεύεται από σημαντικές προκλήσεις. Στα κατανεμημένα συστήματα, η προσθήκη μιας νέας υπηρεσίας ή λειτουργικότητας συνοδεύεται συχνά από μη προβλέψιμες επιπτώσεις στην απόδοση του συστήματος, όπως αναφέρει επανειλημμένα ο Kleppmann [9]. Αν και οι μηχανισμοί όπως το caching και το observability αποσκοπούν στην βελτίωση της εμπειρίας και της διαχειρισιμότητας, μπορούν, ως παράπλευρη συνέπεια, να οδηγήσουν σε αύξηση του latency ή της κατανάλωσης πόρων. Το φαινόμενο αυτό αποδίδεται στην πολυπλοκότητα των αλληλεπιδράσεων μεταξύ μικροϋπηρεσιών και την έλλειψη πλήρους ορατότητας σε χαμηλό επίπεδο.

Κατ' επέκταση, η ερευνητική συνεισφορά εστιάζει όχι μόνο στην επι-

λογή σχετικών τεχνολογιών, αλλά κυρίως στη συνεκτική και δυναμική ενορχήστρωσή τους, με στόχο την επίτευξη αυτονομίας και διαχειρισιμότητας σε cloud-native κατανεμημένα περιβάλλοντα υψηλής πολυπλοκότητας, για την υποστήριξη αυτού του συστήματος πραγματικού χρόνου στο χώρο του *food security*.

3
Θεωρητικό Υπόβαθρο

Η εξέλιξη των Κυβερνο-Φυσικών Συστημάτων (CPS) και των εφαρμογών IoT βασίζεται σε θεμελιώδεις αρχιτεκτονικές αρχές που συνδυάζουν την αποδοτική διαχείριση της φυσικής πληροφορίας με τις δυνατότητες της σύγχρονης υπολογιστικής τεχνολογίας. Σε αυτό το κεφάλαιο παρουσιάζονται τα βασικά αρχιτεκτονικά μοντέλα, οι μέθοδοι διασύνδεσης υποσυστημάτων και τα πρότυπα διαχείρισης δεδομένων που επικρατούν στα σύγχρονα CPS.

3.1 Αρχιτεκτονικες CPS και Κατανεμημενα Συστηματα

Τα CPS δομούνται τυπικά σε τρία επίπεδα: το **φυσικό επίπεδο** (sensing/actuation), το **επίπεδο δικτύου** (networking/communication) και το **κυβερνητικό ή υπολογιστικό επίπεδο** (cyber/computation). Το φυσικό επίπεδο περιλαμβάνει αισθητήρες και ενεργοποιητές, το δίκτυο μεταφέρει τα δεδομένα, ενώ το κυβερνητικό επίπεδο διαχειρίζεται την επεξεργασία, την ανάλυση και τη λήψη αποφάσεων.

Η σύγχρονη σχεδίαση CPS ακολουθεί τα παρακάτω αρχιτεκτονικά μοτίβα:

- **Layered (στρωματοποιημένη) αρχιτεκτονική**: Διαχωρισμός φυσικών, δικτυακών και υπολογιστικών λειτουργιών, διευκολύνοντας

τη διαλειτουργικότητα και τη συντήρηση.

- **Κατανεμημένη επεξεργασία**: Τα δεδομένα δεν συγκεντρώνονται υποχρεωτικά σε έναν κεντρικό κόμβο, αλλά επεξεργάζονται τοπικά (*edge/fog computing*) ή υβριδικά, βελτιώνοντας την καθυστέρηση και την ανθεκτικότητα.
- **Event-driven και streaming αρχιτεκτονική**: Αντί της παραδοσιακής batch επεξεργασίας, τα δεδομένα ρέουν ως ακολουθίες γεγονότων (streams), επιτρέποντας την άμεση αντίδραση σε αλλαγές του περιβάλλοντος.
- **Μικροϋπηρεσίες (Microservices)**: Ανάπτυξη του λογισμικού σε μικρές, αυτόνομες υπηρεσίες με σαφή όρια ευθύνης και ανεξάρτητο κύκλο ζωής.
- **Loose coupling & αναγνωσιμότητα**: Η επικοινωνία μεταξύ των υποσυστημάτων βασίζεται σε ασύγχρονα μηνύματα (π.χ. μέσω pub/sub patterns), διατηρώντας την αυτονομία και διευκολύνοντας την κλιμάκωση.

3.2 Streams, Event-Driven και Batch vs Real-Time Processing

Η *ροή δεδομένων* (stream) ορίζεται ως η συνεχής αλληλουχία δεδομένων που παράγονται από αισθητήρες ή άλλες πηγές και διακινούνται ασύγχρονα εντός του συστήματος. Η **event-driven αρχιτεκτονική** επιτρέπει τη λήψη αποφάσεων ή την ενεργοποίηση ενεργειών αμέσως με την εμφάνιση ενός γεγονότος, κάτι που είναι κρίσιμο για εφαρμογές με απαιτήσεις απόκρισης σε πραγματικό χρόνο, όπως η γεωργία ακριβείας, το predictive maintenance, ή η αυτονομία ρομποτικών συστημάτων.

Σε αντίθεση, τα παραδοσιακά *batch systems* επεξεργάζονται δεδομένα συγκεντρωτικά και περιοδικά, οδηγώντας σε υστέρηση που δεν είναι αποδεκτή για CPS με κρίσιμες λειτουργίες. Η υιοθέτηση stream processing (με αρχιτεκτονικές Lambda/Kappa) προσφέρει καλύτερη προσαρμοστικότητα και αυτονομία, ενώ μειώνει την καθυστέρηση λήψης αποφάσεων.

3.3 Αρχιτεκτονικεσ Streams: Lambda και Kappa

Οι δύο επικρατέστερες αρχιτεκτονικές για επεξεργασία ροών είναι:

- **Lambda**: Συνδυάζει speed layer για άμεση επεξεργασία (με πιθανό μικρότερο accuracy) και batch layer για πλήρη ανάλυση με υψηλότερη ακρίβεια, υποστηρίζοντας τόσο real-time όσο και offline analytics.
- **Kappa**: Ενιαία προσέγγιση, όπου όλα τα δεδομένα αντιμετωπίζονται ως streams, απλοποιώντας τη διαχείριση και επανεπεξεργασία γεγονότων χωρίς ανάγκη ξεχωριστού batch υποσυστήματος.

Η επιλογή μοντέλου εξαρτάται από το αν προτεραιότητα δίνεται στην αμεσότητα ή στην αναλυτική επεξεργασία και από τη συνολική πολυπλοκότητα συντήρησης. Σε πρακτικό επίπεδο, υλοποιήσεις όπως το *Apache Kafka* (messaging/stream log), *Apache Flink* και *Apache Spark* (stream processing) έχουν επικρατήσει ως τεχνολογικά standards, λόγω της ευελιξίας, αξιοπιστίας και δυνατότητας διασύνδεσης με άλλα υποσυστήματα.

3.4 Διασυνδεση Υποσυστηματων

Για την αλληλεπίδραση πολλαπλών, ετερογενών υποσυστημάτων, υιοθετείται το **publish/subscribe** μοτίβο. Αυτό επιτρέπει την ασύγχρονη και χαλαρά συνδεδεμένη (*loosely coupled*) επικοινωνία μεταξύ παραγωγών (*publishers*) και καταναλωτών (*subscribers*), διασφαλίζοντας την αυτονομία, την επεκτασιμότητα και τη δυνατότητα αλλαγών στην τοπολογία του συστήματος χωρίς κεντρικό συντονισμό.

Η ποιότητα υπηρεσίας (*Quality of Service*, *QoS*) αποκτά ιδιαίτερη σημασία, καθώς καθορίζει το επίπεδο αξιοπιστίας στη διανομή μηνυμάτων, ειδικά σε περιβάλλοντα με ασταθή συνδεσιμότητα (π.χ. αγροτικά δίκτυα, βιομηχανικά IoT). Για ποιοτικά περιορισμένες συσκευές και low-power δίκτυα, πρωτόκολλα όπως το *MQTT* προσφέρουν ελαφριά, αξιόπιστη και ασφαλή ανταλλαγή μηνυμάτων, ενισχύοντας την ανθεκτικότητα του συστήματος.

3.5 Αποθηκευση, Consistency Models και Caching

Η αποθήκευση δεδομένων σε CPS συνιστά αρχιτεκτονική πρόκληση λόγω του όγκου, της ταχύτητας παραγωγής και της ανάγκης για real-

time προσπέλαση. Κατανεμημένα συστήματα βάσεων (NoSQL, wide-column stores) με replication και partitioning διασφαλίζουν διαθεσιμότητα και fault tolerance.

Η επιλογή consistency model (strong, eventual, causal) εξαρτάται από τις απαιτήσεις της εφαρμογής για ακρίβεια vs ταχύτητα και διαθεσιμότητα. Σε συστήματα που μπορούν να αντέξουν μικρές απώλειες ή προσωρινή ασυνέπεια, το eventual consistency αποτελεί πρακτικό συμβιβασμό.

Επισημαίνεται επίσης, ότι η χρήση *in-memory caching* (π.χ. με Redis/Memcached) μειώνει δραστικά το latency για δεδομένα που προσπελαύνονται συχνά ή απαιτούν άμεση διαθεσιμότητα, με trade-offs σε αντοχή σε αποτυχίες (durability).

3.6 Διαχειριση Υποδομης και Cloud-Native Patterns

Η ανάπτυξη CPS σε cloud ή υβριδικά περιβάλλοντα απαιτεί αυτοματοποιημένη διαχείριση υποδομής και υπηρεσιών. Εδώ κυριαρχεί η χρήση πλατφόρμων όπως το Kubernetes, που υλοποιούν:

- **Δηλωτική διαχείριση (declarative infrastructure)**: Η επιθυμητή κατάσταση του συστήματος ορίζεται μέσω configuration, και ο orchestrator διασφαλίζει ότι αυτή τηρείται.
- **Αυτόματη κλιμάκωση (auto-scaling), αυτοΐαση (self-healing)**: Προσθήκη/αφαίρεση pods, επανεκκινήσεις σε περίπτωση αποτυχιών, health checks.
- **Παρατηρησιμότητα (observability)**: Συλλογή μετρικών, logs, tracing και alerting, με ενδεικτικά εργαλεία Prometheus, Grafana, ELK/EFK stack.
- **Διαμοιρασμός πόρων (virtualization)**: Ευνοϊκή μεθοδολογία για την απομόνωση εφαρμογών και την καλύτερη αξιοποίηση του διαθέσιμου hardware μέσω εικονικών μηχανών ή containers.
- **Αυτοματοποιημένος έλεγχος με Kubernetes Operators**: Επέκταση του control plane του Kubernetes, με αρχές αυτοματοποιημένων συστημάτων ελέγχου για τη διαχείριση σύνθετων, κατανεμημένων εφαρμογών.

- **Service discovery, load balancing και secrets management**: Απαραίτητα για τη διασύνδεση μικροϋπηρεσιών και τη διατήρηση ασφάλειας και διαθεσιμότητας.

Έτσι, η υποδομή μεταμορφώνεται σε έναν ζωντανό οργανισμό που προσαρμόζεται διαρκώς στις απαιτήσεις του συστήματος και του περιβάλλοντος, προσφέροντας ευελιξία και ανθεκτικότητα χωρίς ανθρώπινη παρέμβαση. Με αυτόν τον τρόπο, περιορίζονται τα λάθη του ανθρώπινου παράγοντα, καθώς κρίσιμες λειτουργίες αυτοματοποιούνται και αναλαμβάνονται από το ίδιο το σύστημα.

4
Εργαλεία

Τα Κυβερνο-Φυσικά Συστήματα (CPS) είναι συστήματα που αποτελούνται από ένα φυσικό στοιχείο το οποίο ελέγχεται ή παρακολουθείται από ένα κυβερνητικό (cyber) στοιχείο, έναν αλγόριθμο βασισμένο σε υπολογιστή. Με στόχο να μετασχηματίσουν τον τρόπο με τον οποίο οι άνθρωποι αλληλεπιδρούν με τα μηχανικά συστήματα, τα νέα έξυπνα CPS οδηγούν την καινοτομία σε διάφορους τομείς, βασικός εκ των οποίων αποτελεί η γεωργία [10]. Η αρχιτεκτονική των CPS βασίζεται σε τρία βασικά επίπεδα: το φυσικό επίπεδο (physical layer), όπου καταγράφονται και παράγονται τα δεδομένα μέσω αισθητήρων· το επίπεδο δικτύου (network layer), που εξασφαλίζει τη μεταφορά των δεδομένων· και το κυβερνητικό ή υπολογιστικό επίπεδο (cyber layer), όπου λαμβάνονται αποφάσεις βάσει των εισερχόμενων δεδομένων. Η συνύπαρξη αυτών των στρωμάτων σε ένα κοινό σύστημα καθιστά τα CPS ιδιαίτερα κατάλληλα για εφαρμογές που απαιτούν χαμηλή καθυστέρηση, αξιοπιστία και αυτονομία. Όπως προαναφέρθηκε, στο πεδίο της γεωργίας ακριβείας, τα CPS διαδραματίζουν καθοριστικό ρόλο, καθώς συνδυάζουν αισθητήρες πεδίου, μηχανισμούς ελέγχου άρδευσης, και αλγορίθμους πρόβλεψης βασισμένους σε δεδομένα για να εξασφαλίσουν βέλτιστες συνθήκες καλλιέργειας. Οι τεχνολογίες αυτές επιτρέπουν τη δυναμική λήψη αποφάσεων, μειώνουν τις απώλειες και αυξάνουν την αποδοτικότητα σε όλα τα στάδια της παραγωγής.

4.1 Kafka

Για να επιτευχθεί όμως η πλήρης δυναμική των CPS, απαιτούνται ισχυρές υποδομές διασύνδεσης και διαχείρισης δεδομένων. Εδώ εντάσσεται η ανάγκη για αξιόπιστες messaging πλατφόρμες, όπως το Apache Kafka, που διασφαλίζουν την συνεχή και αξιόπιστη ροή πληροφοριών ανάμεσα στα υποσυστήματα ενός CPS. Το Apache Kafka αποτελεί ένα - πλέον - de facto πρότυπο για την υλοποίηση τέτοιων messaging υποδομών, χάρη στη δυνατότητα του να αποθηκεύει τα μηνύματα σε μορφή καταγραφής (log-based) [11]. Η αρχιτεκτονική του Kafka ενδείκνυται ιδιαίτερα για περιβάλλοντα που απαιτούν real ή near-real time επεξεργασία γεγονότων, καθώς παρέχει υψηλή διαθεσιμότητα, εγγυημένη διανομή μηνυμάτων και δυνατότητα οριζόντιας κλιμάκωσης. Συγκριτικές μελέτες [12] επιβεβαιώνουν ότι το Kafka παρουσιάζει σημαντικό πλεονέκτημα σε όρους throughput και fault tolerance σε σχέση με άλλες προσεγγίσεις, γεγονός που το καθιστά κατάλληλο για απαιτητικές IoT εφαρμογές με αυξημένο όγκο και ταχύτητα δεδομένων.

Σε ένα σύστημα όπως το *Nostradamus*, όπου η ροή των δεδομένων είναι αδιάκοπη και εξελίσσεται σε πραγματικό χρόνο, η ταχεία εισαγωγή και εξαγωγή των δεδομένων εκτιμάται, τόσο για λόγους απόδοσης όσο και για την ελαχιστοποίηση της κατανάλωσης υπολογιστικών πόρων. Δοθέντος ενός συνόλου αισθητήρων τοποθετημένων σε έναν αγρό, οι οποίοι παράγουν συνεχώς δεδομένα, ο κεντρικός broker πρέπει να τα λαμβάνει ορθά και εντός λογικών χρονικών πλαισίων, ενώ ταυτόχρονα να τα επεξεργάζεται χωρίς να καταπονεί το συνολικό σύστημα. Συνεπώς, πρέπει να ληφθούν υπόψη τόσο η ποιότητα υπηρεσίας (Quality of Service - QoS) της messaging υποδομής όσο και οι εγγυήσεις παράδοσης που αυτή προσφέρει. Στη συγκεκριμένη εφαρμογή, η απώλεια ενός μεμονωμένου δείγματος αισθητήρα δεν επηρεάζει σημαντικά τη συνολική ανάλυση, επομένως η πολιτική "**at-least-once**" αποτελεί μια ασφαλή και επαρκή επιλογή. Οι Kreps et al. [13] προσθέτουν, πως το Kafka σχεδιάστηκε εξαρχής με γνώμονα το υψηλό throughput, αποφεύγοντας περίπλοκους μηχανισμούς όπως το two-phase commit και υιοθετώντας πιο αποδοτικές λύσεις για περιπτώσεις όπου η απώλεια μηνυμάτων είναι αποδεκτή.

Τα δεδομένα που εισάγονται στα topics του *Kafka* επεξεργάζονται και καταλήγουν είτε σε επόμενα topics για μετέπειτα ανάλυση, είτε απευθείας σε αποθηκευτικά συστήματα. Ο τρόπος με τον οποίο πραγματοποιείται η εν λόγω επεξεργασία οφείλει να είναι, εν γένει, σε πραγματικό ή σχεδόν πραγματικό χρόνο, καθώς τα δεδομένα παράγονται και

εισάγονται στο σύστημα σε συνθήκες online ροής. Επομένως, εργαλεία όπως το *Apache Flink*, τα οποία παρέχουν native υποστήριξη για stream processing με χαμηλή καθυστέρηση, καθίστανται ιδανικά για την υλοποίηση του ενδιάμεσου επιπέδου επεξεργασίας.

4.2 Flink

Το *Flink* επιτρέπει τον ορισμό παραθύρων (windows) με βάση τον χρόνο γεγονότος (event time), την υλοποίηση πολύπλοκων λειτουργιών (όπως filtering, aggregation, enrichment), καθώς και την ενσωμάτωση με messaging και αποθηκευτικά συστήματα, μεταξύ των οποίων και το *Apache Kafka* και το *Apache Cassandra*. Επιπλέον, μέσω του μηχανισμού state management που διαθέτει, διασφαλίζεται η αξιοπιστία της επεξεργασίας, ακόμη και σε περιπτώσεις προσωρινών αποτυχιών. Η ενσωμάτωση του σε αρχιτεκτονικές τύπου CPS επιτρέπει τη δημιουργία πραγματικά αντιδραστικών συστημάτων, τα οποία μπορούν να λαμβάνουν αποφάσεις βάσει εξελισσόμενων δεδομένων, χωρίς να απαιτείται off-line επεξεργασία ή χρονική υστέρηση.

4.3 Cassandra

Η αποθήκευση μεγάλου όγκου δεδομένων σε κατανεμημένα συστήματα βασίζεται - κυρίως - σε βάσεις δεδομένων τύπου *wide-column*, με το *Apache Cassandra* να αποτελεί ένα από τα πιο διαδεδομένα και ώριμα συστήματα σε αυτόν τον χώρο. Το Cassandra ακολουθεί το μοντέλο *eventual consistency*, υποστηρίζει κατανεμημένη αποθήκευση με replication και partitioning, και έχει σχεδιαστεί για *write-heavy* εφαρμογές [14]. Σε περιβάλλοντα IoT, όπου οι συσκευές παράγουν συνεχώς δεδομένα τηλεμετρίας (π.χ. θερμοκρασία, τάση, ρεύμα) με υψηλή συχνότητα, η Cassandra μπορεί να λειτουργήσει ως backend αποθήκευσης για ροές δεδομένων σχεδόν σε πραγματικό χρόνο, διασφαλίζοντας υψηλή διαθεσιμότητα και συνεχή εγγραφή με χαμηλό latency. Η προσέγγιση αυτή έχει εφαρμοστεί επιτυχώς σε σενάρια όπως η παρακολούθηση φωτοβολταϊκών μονάδων μέσω Raspberry Pi [15], όπου το σύστημα συλλέγει και αποθηκεύει μετρήσεις αισθητήρων κάθε 15 λεπτά για περαιτέρω ανάλυση και βελτιστοποίηση απόδοσης. Αντίστοιχες αρχιτεκτονικές υποδεικνύουν τη σημασία της επιλογής αποθηκευτικού συστήματος που να ανταποκρίνεται τόσο σε επιχειρησιακές ανάγκες χαμηλής καθυστέρησης όσο και σε απαιτήσεις αξιοπιστίας και επεκτασιμότητας.

Στη συγκεκριμένη περίπτωση, η υψηλή απόδοση (high performance) της *Cassandra* σε συνδυασμό με την εύκολη και ελαστική επεκτασιμότητα (elastic scalability), καθιστά το σύστημα ιδανικό για εφαρμογές πραγματικού χρόνου με έντονη ροή δεδομένων. Η δυνατότητα προσθήκης ή αφαίρεσης κόμβων (nodes) χωρίς διακοπή λειτουργίας επιτρέπει την ομαλή προσαρμογή στις αυξομειώσεις φορτίου, επιτρέποντας το self-healing, διατηρώντας παράλληλα σταθερή τη χρονική απόκριση. Καθώς το σύστημα απαιτεί ταχεία επεξεργασία, άμεση καταχώρηση και αξιόπιστη αποθήκευση δεδομένων αισθητήρων πεδίου, η επιλογή μιας αρχιτεκτονικής βασισμένης στην *Cassandra* εξασφαλίζει υψηλή διαθεσιμότητα και ανθεκτικότητα σε αποτυχία. Το λειτουργικό όφελος που προκύπτει από αυτήν τη σχεδίαση δεν είναι απλώς επιθυμητό αλλά κρίσιμης σημασίας, ιδιαίτερα σε περιβάλλοντα με απαιτήσεις χαμηλού latency, συνεχούς διαθεσιμότητας και γραμμικής επεκτασιμότητας.

4.4 Caching

Τελικός στόχος του συστήματος Nostradamus είναι η αξιοποίηση των αποθηκευμένων και επεξεργασμένων δεδομένων σε εφαρμογές διεπαφής, ώστε οι χρήστες να μπορούν να λαμβάνουν οπτικοποιημένη και άμεσα αξιοποιήσιμη πληροφόρηση σχετικά με την καλλιέργειά τους. Οι κόμβοι της βάσης δεδομένων διαδραματίζουν κρίσιμο ρόλο στην τροφοδότηση των εφαρμογών με δεδομένα σε πραγματικό χρόνο. Η ανάγκη για άμεση προσπέλαση σε δεδομένα, ιδιαίτερα σε περιβάλλοντα όπου η εισροή πληροφορίας είναι συνεχής και εν δυνάμει μαζική, καθιστά απαραίτητη τη χρήση μηχανισμών προσωρινής αποθήκευσης (in-memory caching).

Στο πλαίσιο αυτό, τεχνολογίες όπως το *Memcached* [16] και το *Redis* [17] προσφέρουν σημαντικά πλεονεκτήματα, μέσω της υλοποίησης γρήγορων και αποδοτικών μηχανισμών αποθήκευσης ζευγών κλειδιού-τιμής (key-value pairs). Οι εν λόγω λύσεις συμβάλλουν ουσιαστικά στη μείωση του χρόνου και του κόστους προσπέλασης σε δεδομένα που διαφορετικά θα απαιτούσαν αναζήτηση στη βάση. Η βασική τους διαφοροποίηση εντοπίζεται κυρίως στο αρχιτεκτονικό τους μοντέλο και στον τρόπο διαχείρισης της ταυτόχρονης εκτέλεσης, με το *Memcached* να βασίζεται σε multithreaded επεξεργασία, ενώ το *Redis* αξιοποιεί μοντέλο event loop.

4.5 Kubernetes

Η ανάγκη για παρατηρησιμότητα, αποδοτική διαχείριση προσωρινών δεδομένων και σταθερή ροή μηνυμάτων σε περιβάλλοντα με κατανεμημένη υπολογιστική λογική, καθιστά την πλατφόρμα *Kubernetes*, συχνά, απαραίτητο δομικό στοιχείο. Η δυνατότητα αυτόματης επανεκκίνησης αποτυχημένω πόρων (self-healing), η υποστήριξη οριζόντιας κλιμάκωσης (horizontal pod autoscaling) και η ενσωματωμένη παρακολούθηση της κατάστασης των υπηρεσιών (liveness/readiness probes), προσδίδουν λειτουργική ανθεκτικότητα και διατηρούν τη διαθεσιμότητα του συστήματος σε υψηλά επίπεδα, ακόμη και σε συνθήκες έντονου φορτίου ή υλικών αποτυχιών.

Πέραν της αυτοματοποιημένης διαχείρισης πόρων, το *Kubernetes* παρέχει μια ενιαία πλατφόρμα παρατηρησιμότητας. Μέσω της ενσωμάτωσης εργαλείων όπως το *Prometheus* για συλλογή μετρικών, το *Grafana* για οπτικοποίηση και το *AlertManager* για την αποστολή ειδοποιήσεων, [18] - κοινώς τη **lingua franca** του *observability* - ενισχύεται η κατανόηση της δυναμικής συμπεριφοράς του συστήματος σε πραγματικό χρόνο. Η εγγενής υποστήριξη μηχανισμών για service discovery, load balancing και declarative configuration, επιτρέπει την ευέλικτη ανάπτυξη και τον συντονισμό μικροϋπηρεσιών, διευκολύνοντας την επίτευξη στόχων υψηλής διαθεσιμότητας και επεκτασιμότητας σε cloud-native περιβάλλοντα. Συνεπώς, υιοθετώντας αντίστοιχα πρότυπα, μπορούν να προληφθούν ανεπιθύμητες καταστάσεις και να διασφαλιστεί η διατήρηση της ομαλής λειτουργίας ακόμα και υπο συνθήκες υψηλής πολυπλοκότητας.

5
Υλοποιήσεις

Η υλοποίηση του *Nostradamus* ακολουθεί πιστά τα συμπεράσματα της θεωρητικής ανάλυσης: το οικοσύστημα IoT που εξυπηρετεί γεωργικά deployments απαιτεί πλατφόρμα που να είναι ταυτόχρονα ανθεκτική, παρατηρήσιμη και επεκτάσιμη. Για τον λόγο αυτόν επιλέχθηκε ως θεμέλιο το *Kubernetes*, καθώς παρέχει ενιαίο μοντέλο ορισμού υποδομής, ώριμα primitives αυτοΐασης και μηχανισμούς αυτοματοποιημένης κλιμάκωσης. Η επιλογή δεν περιορίζεται σε «cloud native» χαρακτηρισμούς· στην πράξη επιτρέπει στο σύστημα να ανακάμπτει χωρίς ανθρώπινη παρέμβαση όταν κάποιος κόμβος καθίσταται μη διαθέσιμος ή όταν ένα *container* παρουσιάσει προσωρινή αστάθεια. Χάρη στα *liveness* και *readiness probes*, κάθε μικροϋπηρεσία εκθέτει ρητά τη λειτουργική της κατάσταση, με αποτέλεσμα οι εξαρτώμενες υπηρεσίες να μην παραλαμβάνουν αιτήματα πριν επιβεβαιωθεί ότι είναι έτοιμες.

Η αρχιτεκτονική αυτή εξυπηρετεί τον επιχειρησιακό στόχο της γεωργίας ακρίβειας: απαιτείται διαρκής διαθεσιμότητα ακόμη και σε απομονωμένα αγροτικά δικτυακά περιβάλλοντα, ενώ η μετάπτωση από χαμηλό σε υψηλό φορτίο πρέπει να γίνεται με προβλέψιμο τρόπο. Παράλληλα, το *Kubernetes* λειτουργεί ως κοινό υπόστρωμα παρατηρησιμότητας. Η εγγενής υποστήριξη για *Prometheus* metrics, η στενή ολοκλήρωση με *Grafana* dashboards και η δυνατότητα ενσωμάτωσης *tracing* παρόχων (π.χ. Tempo) επιτρέπουν την πλήρη απεικόνιση της ροής δεδομένων, από τον αισθητήρα μέχρι το API. Η ενότητα που ακολουθεί παρουσιάζει πώς οι παραπάνω αρχές μεταφράστηκαν σε συγκεκριμένες επιλογές

υποδομής και υλοποίησης.

5.1 Αρχιτεκτονικη υποδομης

Η υποδομή της πλατφόρμας σχεδιάστηκε με γνώμονα τρεις αλληλένδετους στόχους: επεκτασιμότητα, διαθεσιμότητα και ασφάλεια. Η φιλοσοφία του *Kubernetes* επιτρέπει την περιγραφή αυτών των ιδιοτήτων ως κώδικα, άρα την επανάληψη και την auditability κάθε αλλαγής. Στο πλαίσιο της παρούσας εργασίας, η παραγωγική υλοποίηση φιλοξενείται σε *homelab* ώστε να διατηρείται πλήρης έλεγχος του hardware και των δικτυακών ρυθμίσεων. Το στήσιμο ωστόσο ακολουθεί πρακτικές που επιτρέπουν άμεση μετεγκατάσταση σε δημόσιους παρόχους (π.χ. χρήση *Container Storage Interface*, *LoadBalancer abstraction*), διασφαλίζοντας ότι η πλατφόρμα δεν εξαρτάται από ιδιοκτησιακά χαρακτηριστικά.

Κάθε συνιστώσα της υποδομής περιγράφεται με δηλωτικά manifests, τα οποία υλοποιούνται μέσω *Infrastructure as Code* και αναβαθμίζονται από GitOps ροές. Η συγκεκριμένη προσέγγιση παρέχει σαφή ιχνηλασιμότητα των αλλαγών, διευκολύνει την αναπαραγωγή πειραμάτων και επιτρέπει την εφαρμογή έλεγχων ασφαλείας (policy enforcement) πριν οι ρυθμίσεις περάσουν στο παραγωγικό cluster. Η ύπαρξη τέτοιων διαδικασιών είναι ιδιαίτερα σημαντική σε πλατφόρμες που διαχειρίζονται ευαίσθητες μετρήσεις και πιστοποιητικά χρήστη.

Επιπρόσθετα, η συγκεκριμένη αρχιτεκτονική ευθυγραμμίζεται με θεμελιώδεις αρχές των κατανεμημένων συστημάτων, όπως η παραμετροποίηση μέσω δηλωτικών μοντέλων, η ανοχή σε μερικές αστοχίες και η παροχή σταθερής συμπεριφοράς υπό συνθήκες μεταβλητού φορτίου. Η τυποποίηση του ελέγχου κατάστασης και η δυνατότητα επαναπροσδιορισμού της επιθυμητής κατάστασης καθιστούν το σύστημα εγγενώς συμβατό με βέλτιστες πρακτικές αξιοπιστίας που συναντώνται στη σύγχρονη βιβλιογραφία.

5.1.1 Τοπολογία και ρόλοι κόμβων

Η υποδομή αποτελείται από έναν *Kubernetes cluster* με δύο σαφείς κατηγορίες κόμβων:

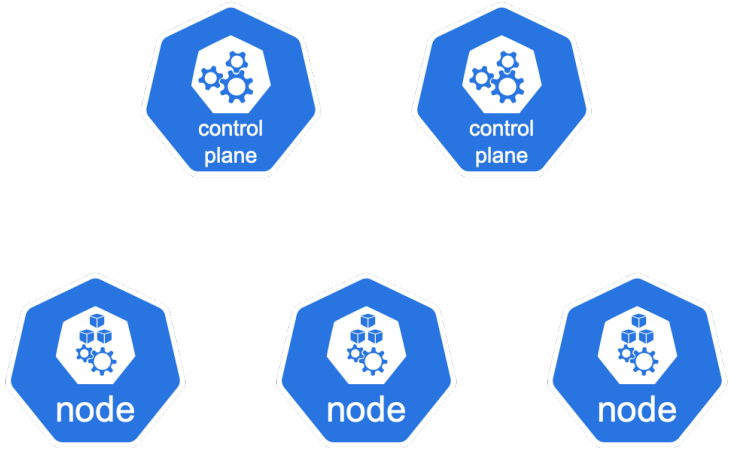
- **Control plane**: Φιλοξενεί τον *API server*, τον *scheduler*, τον *controller manager* και το *etcd*, διατηρώντας την κατάσταση του cluster και λαμβάνοντας αποφάσεις τοποθέτησης.

- **Worker nodes**: Εκτελούν τα *pods* των εφαρμογών, τους *operators* και τα ingress components που δρομολογούν τα αιτήματα των χρηστών.

Ο διαχωρισμός αυτός είναι θεμελιώδης: το control plane παραμένει απρόσβλητο από πιθανές αστάθειες των workloads, διατηρεί το quorum του *etcd* και συνεχίζει να θεραπεύει pods ακόμη και όταν κάποιο worker node αποτυγχάνει. Επιπλέον, επιτρέπει την ανεξάρτητη κλιμάκωση των δύο επιπέδων, ανάλογα με τις ανάγκες παρακολούθησης ή επεξεργασίας δεδομένων.

Ο cluster υλοποιήθηκε σε *Dell PowerEdge R630*, ο οποίος διαμερίστηκε μέσω *Proxmox VE* σε πέντε εικονικές μηχανές. Δύο από αυτές φιλοξενούν το control plane, ενώ οι τρεις υπόλοιπες αποτελούν τους worker nodes με ενισχυμένους πόρους CPU και μνήμης.

Η χρήση υπερ-ενορχηστρωτή τύπου Proxmox σε συνδυασμό με φυσικούς πόρους παρόμοιους με παραγωγικά περιβάλλοντα επιτρέπει την εκτέλεση πειραμάτων σε ρεαλιστικές συνθήκες, διατηρώντας παράλληλα πλήρη ορατότητα στους υποκείμενους πόρους και τη συμπεριφορά του υλικού. Η δυνατότητα λεπτομερούς παρακολούθησης των I/O patterns, της κατανάλωσης μνήμης και των CPU scheduling αποφάσεων αποτελεί σημαντικό πλεονέκτημα σε υλοποιήσεις που στοχεύουν τόσο στη μελέτη όσο και στη σταδιακή ωρίμανση ενός συστήματος.



Σχήμα 5.1: Kubernetes cluster με δύο *controlplane* κόμβους και τρεις *κόμβους*

5.1.2 Αυτοματισμοί υποδομής και διακυβέρνηση

Όλα τα στοιχεία της πλατφόρμας ορίζονται δηλωτικά, με *Helm charts* και manifests που αποθηκεύονται σε κεντρικό αποθετήριο. Οι αλλαγές εφαρμόζονται μέσω τυπικών ροών *GitOps*: κάθε commit που επιφέρει αλλαγές στην υποδομή «ειδοποιεί» έμμεσα έναν *controller*, ο οποίος παρακολουθεί το repository και συγχρονίζει την επιθυμητή κατάσταση με την τρέχουσα κατάσταση του *cluster*. Η πρακτική αυτή εξασφαλίζει ότι το *production cluster* αποτελεί ακριβές στιγμιότυπο του δηλωτικού μοντέλου, διευκολύνοντας την auditability και την αναπαραγωγή σφαλμάτων.

Σε επίπεδο ασφάλειας, η πλατφόρμα οργανώνεται σε διακριτά *Namespaces* με αντίστοιχους κανόνες *RBAC*, ώστε κάθε συνιστώσα να λειτουργεί εντός περιορισμένου επιχειρησιακού πλαισίου. Η λογική αυτή ακολουθεί την καθιερωμένη αρχή του «ελάχιστου απαιτούμενου δικαιώματος» και επιτρέπει στις υπηρεσίες να εκτελούν μόνο τις λειτουργίες που τους αναλογούν, χωρίς να αποκτούν πρόσβαση σε μη σχετιζόμενα υποσυστήματα. Η απομόνωση αυτή αποτελεί ικανοποιητικό επίπεδο προστασίας για το scope της παρούσας εργασίας και διατηρεί τις υπηρεσίες πλήρως ανεξάρτητες σε ό,τι αφορά την ανάπτυξη, τη συντήρηση και την παρακολούθησή τους.

5.2 Ροη δεδομενων σε πραγματικο χρονο

5.2.1 Πηγές ροής δεδομένων

Η αξιόπιστη και χαμηλής καθυστέρησης επεξεργασία ροών δεδομένων σε περιβάλλοντα γεωργίας ακρίβειας προϋποθέτει τη διαχείριση ετερογενών πηγών, από θερμόμετρα εδάφους έως αισθητήρες υγρασίας και *pH*. Κάθε συσκευή εκτελεί το δικό της ελαφρύ πρόγραμμα τηλεμετρίας, το οποίο υποστηρίζει διαφορετικές συχνότητες εκπομπής και μπορεί να ρυθμιστεί ανάλογα με τη λειτουργική ανάγκη του παραγωγού. Η πλατφόρμα παρέχει στον αισθητήρα τα απαραίτητα διαπιστευτήρια και τις παραμέτρους λειτουργίας του, συμπεριλαμβανομένης της επιθυμητής granularity του μηνύματος (π.χ. ανά 5, 30 ή 120 δευτερόλεπτα), ώστε κάθε συσκευή να αποστέλλει δεδομένα με τρόπο προσαρμοσμένο στο προφίλ καλλιέργειας και στο διαθέσιμο δίκτυο. Τα μηνύματα αποστέλλονται σε μορφή JSON μέσω *MQTT* και φτάνουν στο ingestion layer της πλατφόρμας με τη μορφή που τα εκπέμπει η συσκευή, χωρίς ενδιάμεση τροποποίηση. Από εκεί και πέρα η ροή δεδομένων δρομολο-

γείται προς τα κατάλληλα topics και pipelines για περαιτέρω επεξεργασία, όπως περιγράφεται στις επόμενες υποενότητες.

5.2.2 Τεχνολογίες Streaming

Στην προηγούμενη ενότητα παρουσιάστηκε το *Apache Kafka* ως το βασικό μέσο ενδιάμεσης επικοινωνίας μεταξύ των στοιχείων της πλατφόρμας. Σενάρια *streaming*, όπως αυτό της πλατφόρμας *Nostradamus*, βασίζονται στο *Kafka* ως τον κεντρικό κόμβο του συστήματος, ο οποίος λειτουργεί ως η κύρια «πηγή αλήθειας» για όλα τα υποσυστήματα. Κάθε γεγονός που συμβαίνει στο σύστημα καταγράφεται αρχικά στο *Kafka*, και στη συνέχεια κάθε μικροϋπηρεσία εγγράφεται στο αντίστοιχο «θέμα» (*topic*) ώστε να λαμβάνει μόνο τα μηνύματα που της είναι απαραίτητα. Με αυτόν τον τρόπο επιτυγχάνεται ο σαφής διαχωρισμός ανάμεσα στη ροή δεδομένων σε πραγματικό χρόνο και στην κατανάλωση των μηνυμάτων. Συνεπώς, η σωστή σχεδίαση της υποδομής συλλογής και μεταφοράς δεδομένων έως το *Kafka* είναι κρίσιμη, καθώς από εκεί και πέρα ο διαμοιρασμός τους στα υπόλοιπα υποσυστήματα γίνεται με απλό και αποδοτικό τρόπο.

Για το *deployment* του *Kafka* στην πλατφόρμα επιλέχθηκε το μοτίβο του *Kubernetes Operator*, με στόχο την υψηλή διαθεσιμότητα, την ανθεκτικότητα σε αστοχίες και την αυτοματοποιημένη διαχείριση του κύκλου ζωής του συστήματος. Συγκεκριμένα, χρησιμοποιήθηκε ο *Strimzi Kafka Operator*, ο οποίος παρέχει πλήρη αυτοματοποίηση στην εγκατάσταση, ρύθμιση, κλιμάκωση και αναβάθμιση των brokers, καθώς και στη διαχείριση των *topics* και των *users*. Με αυτόν τον τρόπο αξιοποιούνται τα πλεονεκτήματα του Kubernetes, όπως η ευκολία κλιμάκωσης, η αυτόματη αποκατάσταση υπηρεσιών και η συνεπής διαχείριση πόρων, εξασφαλίζοντας παράλληλα σταθερή και αποδοτική λειτουργία της υποδομής ροών. Τέλος, ο *operator* αυτός επιτρέπει την διαχείριση της υποδομής του *Kafka* μέσω *GitOps* πρακτικών, γεγονός που ευθυγραμμίζεται με τις υπόλοιπες αρχές και πρότυπα σχεδίασης της παρούσας εργασίας.

Είναι πλέον καθιερωμένη πρακτική οι αισθητήρες και γενικότερα οι χαμηλής κατανάλωσης μικροελεγκτές που χρησιμοποιούνται σε υποδομές *IoT* να αποστέλλουν τα μηνύματά τους μέσω του πρωτοκόλλου *MQTT*, καθώς αυτό εξοικονομεί ενέργεια και επιτρέπει αποδοτική μετάδοση δεδομένων. Για την εισαγωγή των μηνυμάτων αυτής της μορφής στο *Kafka* απαιτείται η ύπαρξη μηχανισμού γεφύρωσης. Στο πλαίσιο αυτό, αξιοποιήθηκε αρχικά το *Strimzi MQTT bridge*.

Ωστόσο, η ενσωματωμένη υλοποίηση του *Strimzi* δεν υποστηρίζει

ασφαλή μεταφορά μέσω *MQTTs*, γεγονός που αποτελεί κρίσιμο ζήτημα για την παρούσα πλατφόρμα, δεδομένου ότι οι αισθητήρες βρίσκονται σε εξωτερικά δίκτυα. Για την επίλυση του προβλήματος, ενσωματώθηκε ένας *EMQX broker*, ο οποίος λειτουργεί ως ασφαλής πύλη (*secure gateway*) μεταξύ των εξωτερικών συσκευών και του *Kafka*, παρέχοντας υποστήριξη για κρυπτογράφηση και μηχανισμούς πιστοποίησης σύμφωνα με τις απαιτήσεις της αρχιτεκτονικής. Παράλληλα, ο *EMQX broker* παρέχει δυνατότητες φιλτραρίσματος βάσει των εισερχόμενων *MQTT topics* και αντιστοίχισης (*mapping*) αυτών σε *Kafka topics*, ώστε να προωθούνται στο *Kafka* μόνο τα απαραίτητα μηνύματα και να διατηρείται συνεπής η ονοματολογία και η δομή των θεμάτων στην πλατφόρμα.

Η επιλογή του Kafka ως κεντρικού μηχανισμού μεταφοράς δεδομένων δικαιολογείται και από τη φύση των γεωργικών συστημάτων, όπου παρατηρείται συχνά συνδυασμός διαλείπουσας συνδεσιμότητας, απότομων αλλαγών στον ρυθμό παραγωγής δεδομένων και ανομοιογενών payloads. Η εγγυημένη σειρά παράδοσης και η ανθεκτικότητα του commit log καθιστούν το Kafka τον πλέον κατάλληλο μηχανισμό για ροές που απαιτούν επαναληψιμότητα και ανοχή σε αστοχίες, ανεξαρτήτως του backend που καταναλώνει τα μηνύματα.

5.3 Προσωρινη αποθηκευση και cache drivers

5.3.1 Αρχιτεκτονική και ενσωμάτωση

Το επίπεδο προσωρινής αποθήκευσης υιοθετεί το πρότυπο *cache-aside*, με αυστηρά όρια χρόνου ζωής (TTL 120 δευτερολέπτων) και υλοποίηση ως διακριτή διασύνδεση. Η επιλογή αυτή επιτρέπει την εναλλαγή υποστρωμάτων χωρίς αλλαγές στον κώδικα της εφαρμογής, ενώ το ίδιο interface παρέχει ενιαία συλλογή μετρικών και ιχνηλασιμότητα. Εκτός του *Valkey*, έχουν δημιουργηθεί adapters για *Memcached* και *Dragonfly*, τους οποίους η πλατφόρμα ενεργοποιεί ακόμη και κατά τη διάρκεια πειραμάτων συγκριτικής αξιολόγησης ή ως μηχανισμό εφεδρείας. Η ύπαρξη πολλαπλών drivers προσφέρει επιχειρησιακή ευελιξία: σε περίπτωση που ένα υπόστρωμα προσωρινής αποθήκευσης παρουσιάσει αυξημένη καθυστέρηση ή μειωμένη διαθεσιμότητα, ο controller επιτρέπει την άμεση μετάβαση σε εναλλακτικό driver. Σε συνδυασμό με τις *readiness probes*, εξασφαλίζεται ότι οι διεπαφές των χρηστών όπως το *UI* και τα τρίτα συστήματα δεν βλέπουν ποτέ ασταθή κατάσταση.

5.3.2 Επιλογή και συγκριτική αξιολόγηση υποστρωμάτων

Η επιλογή των τριών επιμέρους υποστρωμάτων καθορίστηκε από κριτήρια συμβατότητας, λειτουργικότητας και αναμενόμενης επίδοσης. Το *Valkey* υιοθετήθηκε ως αναφορά, καθώς αποτελεί την πλέον συμβατή και κοινοτικά υποστηριζόμενη μετεξέλιξη του οικοσυστήματος Redis, με πλούσιο σύνολο δομών δεδομένων και δυνατότητα scripting, στοιχεία απαραίτητα για μελλοντικές λειτουργίες της πλατφόρμας. Το *Dragonfly* επιλέχθηκε λόγω της αρχιτεκτονικής του που εστιάζει στην ενιαία πολυνηματική αξιοποίηση της μνήμης και στο περιορισμένο αποτύπωμα πόρων, ενώ διατηρεί συμβατότητα σε επίπεδο πρωτοκόλλου με το Valkey/Redis, καθιστώντας την εναλλαγή πρακτικά διαφανή. Τέλος, το *Memcached* ενσωματώθηκε ως ελαφριά, εδραιωμένη λύση με περιορισμένο σύνολο λειτουργιών, η οποία όμως προσφέρει προβλέψιμη καθυστέρηση, καθιστώντας την ιδανικό σημείο αναφοράς για συγκρίσεις καθαρού ρυθμού διεκπεραίωσης.

Σημαντική παράμετρος της μελέτης είναι ότι το *Valkey* και το *Dragonfly* μοιράζονται κοινό πρωτόκολλο και μοντέλο δεδομένων, γεγονός που επιτρέπει την αξιολόγησή τους ως εναλλακτικές υλοποιήσεις της ίδιας λογικής διασύνδεσης. Και τα δύο υποστρώματα παρέχουν μονόκλωνη σημασιολογία εκτέλεσης εντολών, υποστηρίζουν scripting και σύνθετες δομές (δομές κατακερματισμού, ταξινομημένα σύνολα), καθώς και χαρακτηριστικά όπως replication και μονιμότητα δεδομένων. Κατά συνέπεια, τυχόν διαφοροποιήσεις στην επίδοση μπορούν να αποδοθούν αποκλειστικά στην εσωτερική μηχανική υλοποίηση (μηχανή κατακερματισμού, χρονοδρομολογητής, στρατηγικές χρήσης μνήμης) και όχι σε αλλαγές του προγραμματιστικού μοντέλου. Η επιλογή τους επιτρέπει, επομένως, τη συγκριτική απομόνωση των πλεονεκτημάτων που φέρει μια πιο επιθετική πολυνηματική προσέγγιση (*Dragonfly*) έναντι της κλασικής, πλήρως συμβατής ερμηνείας (*Valkey*), χωρίς να απαιτείται προσαρμογή του εφαρμοστικού κώδικα.

5.3.3 Πειραματικές υποθέσεις

Πριν από την πειραματική διαδικασία διατυπώθηκαν συγκεκριμένα υποθετικά σενάρια: αναμενόταν ότι το *Memcached*, χάρη στο απλούστερο δυαδικό πρωτόκολλο και την απουσία σύνθετων δομών, θα υπερτερεί σε P95 καθυστέρηση ανάγνωσης και εγγραφής όταν το φορτίο είναι πλήρως συμμετρικό. Αντίστοιχα, η υπόθεση για το *Valkey* ήταν ότι θα εμφανίσει υψηλότερη σταθερότητα υπό παρατεταμένα φορτία

και καλύτερη συμπεριφορά σε σενάρια με ανάγκη αδιαίρετων λειτουργιών. Για το *Dragonfly*, η αναμενόμενη συμπεριφορά τοποθετήθηκε ενδιάμεσα: στόχος ήταν να επαληθευθεί αν ο πολυνηματικός χρονοδρομολογητής και η διαχείριση μνήμης που διαθέτει οδηγούν σε ρυθμό διεκπεραίωσης συγκρίσιμο με το *Memcached*, διατηρώντας παράλληλα τη λειτουργική πληρότητα του *Valkey*. Οι υποθέσεις αυτές καταγράφονται εδώ προκειμένου να συσχετιστούν στη συνέχεια με τα αποτελέσματα του Κεφαλαίου 6.

Η διατύπωση των παραπάνω υποθέσεων δεν στοχεύει απλώς στη σύγκριση διαφορετικών υλοποιήσεων, αλλά και στην αξιολόγηση του κατά πόσο η πλατφόρμα μπορεί να υποστηρίξει ετερογενείς drivers χωρίς αλλαγές στην επιχειρησιακή λογική. Η προσέγγιση αυτή επιτρέπει τον διαχωρισμό της λειτουργικότητας της εφαρμογής από τις επιδόσεις του υποστρώματος, στοιχείο που αποτελεί κρίσιμο παράγοντα για την επεκτασιμότητα της πλατφόρμας σε μελλοντικά περιβάλλοντα και διαφορετικά workloads.

5.3.4 Στρατηγικές caching

Η υιοθέτηση μηχανισμών κρυφής μνήμης σε κατανεμημένα συστήματα συνοδεύεται από διαφορετικά πρότυπα (*patterns*) υλοποίησης, καθένα από τα οποία φέρει πλεονεκτήματα και περιορισμούς. Τα συνηθέστερα είναι τα ακόλουθα:

Read-through

Στο πρότυπο αυτό, όλες οι αναγνώσεις δεδομένων γίνονται μέσω της cache. Σε περίπτωση που το ζητούμενο κλειδί δεν υπάρχει, η ίδια η cache αναλαμβάνει να ανακτήσει τα δεδομένα από τη βάση και να τα αποθηκεύσει πριν τα επιστρέψει στον χρήστη. Το πλεονέκτημα είναι η απλοποίηση της εφαρμογής, καθώς η λογική ανάκτησης μετατίθεται στο cache layer. Ωστόσο, απαιτείται στενή ενσωμάτωση της cache με τη βάση, κάτι που δυσχεραίνει την παραμετροποίηση σε περιβάλλοντα με ετερογενείς πηγές δεδομένων.

Write-through

Σε αυτήν τη στρατηγική, κάθε εγγραφή περνάει πρώτα από την cache και στη συνέχεια στη βάση δεδομένων. Η μέθοδος αυτή εξασφαλίζει συνεπή δεδομένα, καθώς cache και βάση ενημερώνονται ταυτόχρονα,

αλλά αυξάνει τον χρόνο απόκρισης για κάθε write. Επιπλέον, το συνολικό throughput περιορίζεται από τον πιο αδύναμο (αργό) κρίκο της αλυσίδας.

Write-behind (ή Write-back)

Τα δεδομένα γράφονται αρχικά μόνο στην cache και σε δεύτερο χρόνο, με ασύγχρονο τρόπο, προωθούνται στη βάση. Το σχήμα αυτό προσφέρει υψηλές επιδόσεις για σενάρια έντονου write load, αλλά εισάγει κινδύνους απώλειας δεδομένων σε περίπτωση αστοχίας της cache, ενώ η βάση ενδέχεται να μην αντικατοπτρίζει την πιο πρόσφατη κατάσταση.

Cache-aside (ή Lazy loading)

Στο πρότυπο αυτό, η εφαρμογή ελέγχει πρώτα την cache. Αν τα δεδομένα υπάρχουν, επιστρέφονται άμεσα. Αν όχι, η εφαρμογή ανατρέχει στη βάση δεδομένων, επιστρέφει τα αποτελέσματα στον χρήστη και ταυτόχρονα τα εισάγει εκ νέου στην cache για μελλοντική χρήση. Το κύριο πλεονέκτημα είναι η απλότητα: η cache λειτουργεί ως προαιρετικό επιταχυντικό στρώμα, χωρίς να απαιτείται βαθιά ενσωμάτωση με τη βάση δεδομένων. Το μειονέκτημα είναι ότι τα δεδομένα μπορεί να είναι στιγμιαία *stale*, ειδικά σε περιπτώσεις έντονων ενημερώσεων.

Για τις ανάγκες της παρούσας πλατφόρμας, επιλέχθηκε το πρότυπο *cache-aside*, καθώς τα ερωτήματα αφορούν κυρίως αναγνώσεις πρόσφατων τιμών αισθητήρων με χαμηλή πιθανότητα συνεχών τροποποιήσεων. Με τον τρόπο αυτό, η ScyllaDB παραμένει η πηγή αλήθειας (*source of truth*), ενώ το Valkey cluster χρησιμοποιείται για να μειώσει τον χρόνο απόκρισης σε συχνά αιτήματα, επιτυγχάνοντας ισορροπία ανάμεσα σε απόδοση και συνέπεια.

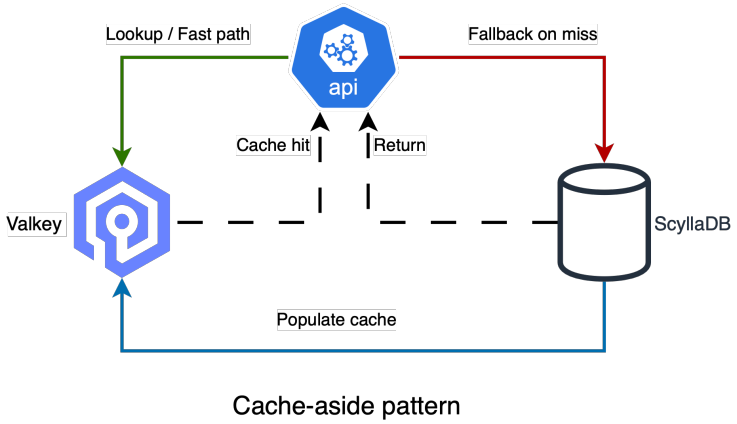
5.4 Επιπεδο υπηρεσιων και εφαρμογης

Το επίπεδο υπηρεσιών συγκροτείται γύρω από ένα κεντρικό API gateway, το οποίο αποτελεί τη βασική διεπαφή των χρηστών και των downstream συστημάτων με τα στοιχεία της πλατφόρμας. Η σχεδίασή του ακολουθεί modular αρχιτεκτονική, με σαφή διαχωρισμό ευθυνών ανάμεσα στη διαχείριση πόρων, την επεξεργασία ροών, την προσωρινή αποθήκευση και την παρατηρησιμότητα.

5.4.1 Valkey cluster

Η υποστήριξη του μηχανισμού cache υλοποιείται με *Valkey cluster*, ο οποίος αναπτύχθηκε μέσω του *Hyperspike Operator*. Η λύση αυτή εξασφαλίζει οριζόντια κλιμάκωση και υψηλή διαθεσιμότητα σε περιβάλλον Kubernetes, με αυτοματοποιημένη διαχείριση των κόμβων Redis/Valkey.

Συνδυάζοντας τον Valkey cluster για ταχύτατη εξυπηρέτηση πρόσφατων μετρήσεων με τη ScyllaDB για ανθεκτική και αναλυτική αποθήκευση, η πλατφόρμα διατηρεί ισορροπία ανάμεσα σε χαμηλή καθυστέρηση και αξιοπιστία δεδομένων.



Σχήμα 5.2: Λογικό διάγραμμα του προτύπου *cache-aside*: το API ελέγχει αρχικά την κρυφή μνήμη (Valkey) για την αναζήτηση δεδομένων και, σε περίπτωση αποτυχίας, προσφεύγει στη ScyllaDB. Τα δεδομένα επιστρέφονται στον χρήστη και ταυτόχρονα επανεισάγονται στην cache, ώστε να εξυπηρετηθούν με χαμηλή καθυστέρηση σε επόμενα αιτήματα.

5.4.2 API gateway και διαχείριση πόρων

Το API παρέχει REST endpoints για την εγγραφή αγροτεμαχίων, την ενεργοποίηση αισθητήρων, την έκδοση διαπιστευτηρίων και την προβολή συγκεντρωτικών τιμών μέσω του `/aggregate`. Το API ελέγχει τα δικαιώματα κάθε κλήσης, χαρτογραφεί τα αιτήματα σε συγκεκριμένα pipelines και επιβάλλει πολιτικές που προκύπτουν από την ανάλυση των επιχειρησιακών ροών.

Η αρχιτεκτονική του API στηρίζεται σε διακριτά πακέτα: ένα για την πρόσβαση στη βάση *Scylla*, ένα για το caching layer, ένα για τη δρομολόγηση αιτημάτων, καθώς και βοηθητικές βιβλιοθήκες για serialization, logging και ιχνηλασιμότητα. Η επικοινωνία με τη Scylla υλοποιείται

μέσω του οδηγού `gocql`, ενώ το caching layer υλοποιείται ως ανεξάρτητο module αποσυνδεδεμένο από τον αποθηκευτικό μηχανισμό.

5.4.3 Διαχείριση Arroyo pipelines

Η πλατφόρμα *Arroyo* χρησιμοποιείται ως βασικό στοιχείο του streaming layer, καθώς παρέχει connectors από *Kafka* προς *Scylla* σε λειτουργία συνεχούς επεξεργασίας. Το API οργανώνει τα *pipelines* χρησιμοποιώντας abstractions που αντιστοιχούν σε αγροτεμάχια και αισθητήρες. Μόλις καταχωρηθεί μία νέα συσκευή ή καταφτάσουν δεδομένα, ο controller δημιουργεί δυναμικά το αντίστοιχο Arroyo job, το συνδέει με το κατάλληλο *topic* και παρακολουθεί το throughput του. Αν ένα job τερματιστεί, η υπηρεσία αναλαμβάνει την αυτόματη επανεκκίνησή του και την ενημέρωση των καταναλωτών, εξασφαλίζοντας *self-healing* συμπεριφορά πέρα από τα όρια του Kubernetes.

Παρότι σήμερα το Arroyo αξιοποιείται κυρίως ως connector, η σχεδίαση του API επιτρέπει τη μελλοντική έκθεση επιπλέον λειτουργιών σε προχωρημένους χρήστες ή σε DSL που θα πατήσει στο ίδιο API. Με αυτόν τον τρόπο ο κεντρικός έλεγχος παραμένει στους developers της πλατφόρμας, αλλά παρέχεται σαφές σημείο επέκτασης για πιο πολύπλοκα υπολογιστικά pipelines.

5.4.4 Ανθεκτικότητα και ανοχή σε σφάλματα

Στο API λειτουργούν controllers (υλοποιημένοι ως goroutines), οι οποίοι:

- παρακολουθούν τις καταστάσεις των Arroyo jobs,
- αναδημιουργούν pipelines όταν απαιτείται,
- συγχρονίζουν connectors μεταξύ Kafka και Scylla,
- διασφαλίζουν ότι καμία ροή δεν μένει ανεπεξέργαστη.

Με αυτόν τον τρόπο η υπηρεσία επιτυγχάνει self-healing συμπεριφορά, η οποία υπερβαίνει την αυτόματη αποκατάσταση που προσφέρει το Kubernetes.

5.4.5 Υπηρεσία /aggregate και caching layer

Η προβολή συγκεντρωτικών μετρήσεων αποτελεί αναπόσπαστο κομμάτι κάθε συστήματος που επεξεργάζεται δεδομένα αισθητήρων σε πραγματικό χρόνο. Παρότι η πλατφόρμα διατηρεί αναλυτική ροή μέσω των pipelines, οι περισσότεροι παραγωγοί και εφαρμογές καταναλώνουν τα δεδομένα σε συνοπτική μορφή, σε παράθυρα συγκεκριμένης διάρκειας και για συγκεκριμένο τύπο αισθητήρα. Για τον λόγο αυτόν σχεδιάστηκε η υπηρεσία */aggregate*, η οποία λειτουργεί ως ενιαίο σημείο πρόσβασης για την εξαγωγή απλοποιημένων στατιστικών (π.χ. ελάχιστο, μέγιστο, μέσο όρο) πάνω σε πρόσφατα δεδομένα. Η υπηρεσία αυτή αναλαμβάνει να εξυπηρετήσει τα πιο συχνά αναγνώσιμα workloads της πλατφόρμας με προβλέψιμο latency, αξιοποιώντας το caching layer και τις μετρικές παρατηρησιμότητας που έχουν ενσωματωθεί. Κάθε κλήση δέχεται παραμέτρους παραθύρου (π.χ. 1 ώρα) και τύπο αισθητήρα (θερμοκρασία, υγρασία, *pH*), επιστρέφοντας ταυτόχρονα *min*, *max* και *average* τιμές. Η υλοποίηση ακολουθεί μοτίβο *cache-aside*: αρχικά επιχειρείται *CacheFetch*, στη συνέχεια γίνεται *DBFetch* από τη *Scylla* μόνο αν υπάρξει *miss*, και τέλος τα αποτελέσματα αποθηκεύονται με TTL 120 δευτερολέπτων. Ο κύκλος αυτός είναι πλήρως ορατός μέσω traces, ενώ οι σχετικές μετρικές τροφοδοτούν τα dashboards που χρησιμοποιήθηκαν και στα πειράματα.

Το caching layer είναι ανεξάρτητη βιβλιοθήκη με κοινό interface, στην οποία έχουν αναπτυχθεί drivers για *Valkey*, *Memcached* και *Dragonfly*. Ο εκάστοτε driver επιλέγεται με παραμετροποίηση στο *deployment*, γεγονός που επέτρεψε τη γρήγορη εκτέλεση των benchmark σεναρίων και την ανάλυση των P95 καθυστερήσεων. Παράλληλα, ο ίδιος μηχανισμός καταγράφει *cache hit/miss* με granularity ανά αισθητήρα και είδος aggregation, επιτρέποντας την αξιολόγηση της αποδοτικότητας του caching ανά workload. Ως μελλοντική βελτίωση σχεδιάζεται ένα ταχύτερο endpoint μεταφοράς δεδομένων μεταξύ υπηρεσιών, ώστε να παρακάμπτει πλήρως το UI όταν απαιτείται ενδοσυστημική ολοκλήρωση.

Η συγκεκριμένη υπηρεσία λειτουργεί, επιπλέον, ως αντιπροσωπευτικό υπόδειγμα του συνολικού τρόπου με τον οποίο η πλατφόρμα χειρίζεται δεδομένα αισθητήρων. Το */aggregate* προσφέρει ένα πλήρως λειτουργικό σημείο δοκιμής για το caching layer, καθιστώντας εφικτή την αξιολόγηση διαφορετικών drivers και τη μέτρηση της συμπεριφοράς τους κάτω από πραγματικό φόρτο. Ως εκ τούτου, η υπηρεσία διαδραματίζει διττό ρόλο: από τη μία επιτρέπει την άμεση κατανάλωση συγκεντρωτικών μετρήσεων από τους παραγωγούς, και από την άλλη παρέχει ένα σταθερό και επαναλήψιμο περιβάλλον για benchmarking, debugging

και ανάλυση των επιδόσεων της υποδομής. Παρότι το */aggregate* καλύπτει τις βασικές ανάγκες του τρέχοντος συστήματος, η υφιστάμενη αρχιτεκτονική επιτρέπει τη δημιουργία επιπλέον endpoints με παρόμοια λογική, είτε για πιο σύνθετες αναλύσεις είτε για επέκταση σε νέα είδη αισθητήρων και νέους τύπους χρονικών ερωτημάτων, μετατρέποντας σταδιακά την πλατφόρμα σε πλήρες εργαλείο προβολής και επεξεργασίας γεωργικών δεδομένων σε πραγματικό χρόνο.

5.4.6 Frontend και διεπαφές παρακολούθησης

Παρότι το κύριο βάρος της εργασίας εστιάζει στα υποσυστήματα υποδομής, επεξεργασίας ροών και προσωρινής αποθήκευσης, η ύπαρξη ενός λειτουργικού frontend κρίθηκε απαραίτητη ώστε να αναδεικνύονται στην πράξη οι δυνατότητες του API και του caching layer. Το frontend λειτουργεί ως το πρώτο σημείο επαφής των παραγωγών με την πλατφόρμα και αποτελεί ένα ελαφρύ, πλήρως λειτουργικό παράδειγμα κατανάλωσης των υπηρεσιών που περιγράφηκαν στις προηγούμενες ενότητες. Πρακτικά, χρησιμεύει ως «βιτρίνα» της αρχιτεκτονικής: επιτρέπει την ορατή επαλήθευση της ροής δεδομένων, της συμπεριφοράς του */aggregate* και της απόδοσης του caching layer, χωρίς να απαιτούνται ειδικά εργαλεία ή σύνθετα dashboard setups.

Το UI καταναλώνει το API gateway και παρέχει φόρμες για την εγγραφή αγροτεμαχίων, την ενεργοποίηση αισθητήρων και την διαχείριση των ρυθμίσεών τους. Εμφανίζει επίσης τις πιο πρόσφατες μετρήσεις ανά σημείο, αξιοποιώντας άμεσα τα αποτελέσματα της υπηρεσίας */aggregate*. Με αυτόν τον τρόπο το frontend λειτουργεί και ως δείκτης ορθής λειτουργίας του caching layer, αφού η διαφορά μεταξύ cache hits και misses αποτυπώνεται άμεσα στον χρόνο απόκρισης του UI.

Πέρα από τις βασικές ροές onboarding, το frontend παρέχει συνοπτικές μετρικές που συνοψίζουν την τρέχουσα κατάσταση του αγρού, επιτρέποντας στον χρήστη να λάβει μια γρήγορη εικόνα της δραστηριότητας των αισθητήρων. Για πιο αναλυτικά KPIs, χρονοσειρές και διαγνωστικά στοιχεία, οι χρήστες ανακατευθύνονται στα εξειδικευμένα Grafana boards, όπου συγκεντρώνονται οι μετρικές από το monitoring stack της πλατφόρμας.

Ο σχεδιασμός του frontend κρατήθηκε εσκεμμένα απλός. Στόχος του δεν είναι να αναπαράγει την πληρότητα ενός παραγωγικού dashboard, αλλά να λειτουργήσει ως reference implementation πάνω στο API, διευκολύνοντας την ανάπτυξη νέων υπηρεσιών και αποτελώντας σημείο εκκίνησης για μελλοντικές επεκτάσεις. Η υφιστάμενη δομή επιτρέπει την

προσθήκη πρόσθετων σελίδων και endpoints, από εμπλουτισμένα γραφήματα μέχρι real-time feeds, χωρίς να απαιτούνται αλλαγές στο υπόβαθρο της πλατφόρμας.

5.5 Παρατηρησιμοτητα

Η παρατηρησιμότητα αποτελεί θεμελιώδες χαρακτηριστικό της πλατφόρμας, καθώς επιτρέπει την πλήρη κατανόηση της συμπεριφοράς του συστήματος και την επαλήθευση ότι η ροή των αισθητήριων δεδομένων λειτουργεί όπως αναμένεται. Η παρούσα ενότητα αναλύει τους τρεις βασικούς άξονες παρατηρησιμότητας που υλοποιήθηκαν: *μετρικές*, *καταγραφές* και *ιχνηλασιμότητα*.

5.5.1 Μετρικές και SLI παρακολούθησης

Η προσέγγιση με **προτεραιότητα στην παρατηρησιμότητα** ακολουθείται σε όλα τα στοιχεία του συστήματος, ώστε η ροή των αισθητήριων δεδομένων να είναι πλήρως μετρήσιμη και επαληθεύσιμη. Τα βασικά *SLI* που παρακολουθούνται είναι η καθυστέρηση ανάγνωσης και εγγραφής στην cache, ο δείκτης *cache miss* και ο χρόνος ανάγνωσης της βάσης *Scylla*. Τα μεγέθη αυτά εκτίθενται από τις υπηρεσίες μέσω *Prometheus exporters* και προβάλλονται σε ενιαία *Grafana dashboards*, επιτρέποντας άμεση σύγκριση μεταξύ διαφορετικών cache drivers ή διαφορετικών pipelines.

Η συλλογή σημάτων δεν περιορίζεται σε επίπεδο υποδομής. Κάθε κριτικό σημείο της υπηρεσίας */aggregate* φέρει στοχευμένα *metrics*, όπως ο χρόνος ολοκλήρωσης των λειτουργιών *CacheFetch*, *DBFetch* και *CacheStore*, καθώς και αντιστοιχίσεις με το παράθυρο (π.χ. 15 λεπτά, 1 ώρα) και το είδος αισθητήρα (θερμοκρασία, υγρασία, pH). Με αυτόν τον τρόπο η παρατηρησιμότητα διασταυρώνεται με το *business logic* της πλατφόρμας και τα πειραματικά αποτελέσματα μπορούν να χαρτογραφηθούν άμεσα σε συγκεκριμένα workloads.

Η δυνατότητα συσχέτισης σημάτων από πολλαπλά επίπεδα, όπως ο χρόνος ανάγνωσης cache, η καθυστέρηση επεξεργασίας στους brokers και ο τελικός χρόνος απόκρισης του API, προσδίδει στην πλατφόρμα χαρακτηριστικά cross-layer παρατηρησιμότητας. Τα συνδυαστικά αυτά σήματα διευκολύνουν την αναγνώριση συστημικών προβλημάτων που δεν είναι εμφανή όταν μετρώνται απομονωμένα, όπως συμφόρηση I/O ή μη ομοιόμορφη κατανομή φορτίου σε πυρήνες.

5.5.2 Καταγραφές και διασταύρωση συμβάντων

Πέρα από τις μετρικές, εφαρμόζεται τυποποιημένο σύστημα καταγραφής (*structured logging*) με εμπλουτισμό μεταδεδομένων και *correlation IDs*. Όλα τα logs ρέουν προς *Grafana Loki*, επιτρέποντας πολυδιάστατη αναζήτηση ανά αισθητήρα, pipeline ή χρήστη. Η διάσταση της ιχνηλασιμότητας αναλύεται εκτενώς στην υποενότητα «Ιχνηλασιμότητα και ανάλυση αιτημάτων», όπου περιγράφεται ο τρόπος με τον οποίο τα traces συμπληρώνουν τα παραπάνω σήματα.

Το σύστημα καταγραφής λειτουργεί συμπληρωματικά προς τις μετρικές: σε περιπτώσεις απόκλισης χρόνου απόκρισης, τα logs παρέχουν το πλαίσιο (context) και τα συμβάντα που οδήγησαν στη συγκεκριμένη συμπεριφορά.

5.5.3 Ιχνηλασιμότητα και ανάλυση αιτημάτων

Η πλήρης ιχνηλασιμότητα (*tracing*) αποτελεί τον τρίτο πυλώνα της παρατηρησιμότητας και επιτρέπει την αποσύνθεση κάθε αιτήματος σε επιμέρους λειτουργικές φάσεις. Για τον σκοπό αυτόν χρησιμοποιήθηκε η βιβλιοθήκη *OpenTelemetry*, με δρομολόγηση των traces στην υπηρεσία *Tempo* του Grafana μέσω ενός *OTLP gRPC* εξαγωγέα.

Η αρχικοποίηση του tracer provider γίνεται κατά την εκκίνηση της υπηρεσίας: ο εξαγωγέας συνδέεται στο endpoint του Tempo, ενεργοποιείται ο batch processor για αποδοτική αποστολή παρτίδων και ορίζεται *resource* με χαρακτηριστικά όπως το όνομα υπηρεσίας (*nostradamus-api*). Η διαδικασία αρχικοποίησης μεριμνά επίσης για την ασφαλή αποδέσμευση του tracer provider κατά τον τερματισμό, ώστε να μη χαθούν spans που βρίσκονται σε εκκρεμότητα.

Σε επίπεδο εφαρμογής, κάθε κλήση στο */aggregate* ξεκινά span με το context του αιτήματος, επιτρέποντας την αλυσίδωση με downstream κλήσεις. Η υπηρεσία καταγράφει χαρακτηριστικά που περιγράφουν τα βασικά query parameters (αισθητήρας, παράθυρο, τύπος μετρήσεων) και αποτυπώνει την τελική κατάσταση με βάση τους κωδικούς επιστροφής του *OpenTelemetry* (π.χ. *OK*, *Error*). Τα σημεία ελέγχου που χειρίζονται σφάλματα παραμετρικών ελέγχων ή επικοινωνίας με τη βάση δεδομένων συνοδεύουν το span με σχετικές εγγραφές (*recorded errors*) και εμπλουτισμένα logs, έτσι ώστε τα traces να απεικονίζουν πλήρως το μονοπάτι που ακολουθήθηκε.

Ιχνηλασιμότητα στο caching layer

Ιδιαίτερη έμφαση δίνεται στην ενοποίηση της ιχνηλασιμότητας με το caching layer. Κάθε driver (π.χ. *Memcached*, *Valkey*) δημιουργεί δευτερεύοντα spans για τις λειτουργίες *FetchAggregate* και *StoreAggregate*, μεταφέροντας το context του αρχικού HTTP αιτήματος. Τα spans φέρουν κοινά attributes, όπως το είδος του driver, το cache key και τη χρονική διάρκεια ζωής της εγγραφής. Επιπλέον, η ροή αποτυπώνει με σαφήνεια τα αποτελέσματα (*hit*, *miss*, *corrupt*) και όλες τις συνθήκες σφάλματος, καθώς καταγράφονται μαζί με το αντίστοιχό τους status.

Έτσι, το τελικό trace απεικονίζει ολόκληρο το μονοπάτι CacheFetch $\rightarrow$ DBFetch $\rightarrow$ CacheStore, συμπεριλαμβανομένων των χρονικών μεγεθών που παρακολουθούνται παράλληλα και από τα metrics.

Πλεονεκτήματα της ενιαίας ιχνηλασιμότητας

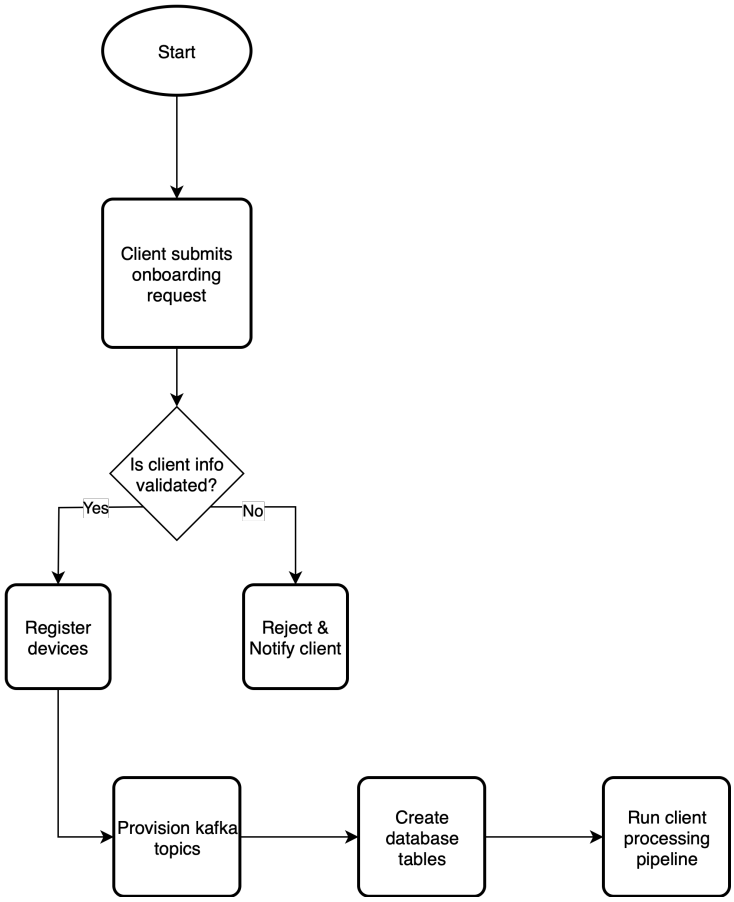
Η παραπάνω προσέγγιση έχει δύο σημαντικά πλεονεκτήματα. Πρώτον, επιτρέπει την ταχεία συσχέτιση των χρονικών αποκλίσεων που εμφανίζονται στα dashboards με συγκεκριμένες κλήσεις και παραμέτρους, γεγονός κρίσιμο για την κατανόηση των P95 καθυστερήσεων. Δεύτερον, παρέχει πλήρη ορατότητα στο σύστημα cache-aside, προσφέροντας στοιχεία για τη συμπεριφορά κάθε driver χωρίς να απαιτείται ξεχωριστή υλοποίηση debugging.

Σε ευρύτερο πλαίσιο, η ιχνηλασιμότητα αποτελεί βασικό μηχανισμό αποσφαλμάτωσης σε κατανεμημένα περιβάλλοντα, όπου οι πιθανές αιτίες καθυστέρησης δεν εντοπίζονται σε μία μόνο διεργασία αλλά προκύπτουν από την αλληλεπίδραση πολλαπλών μικροϋπηρεσιών. Η δυνατότητα οπτικοποίησης των spans επιτρέπει την κατανόηση του πραγματικού data path και καθιστά εφικτή την απομόνωση αιχμών καθυστέρησης που συνήθως καλύπτονται από αθροιστικές μετρικές.

5.6 Ένταξη πελατη

Η διαδικασία onboarding λαμβάνει χώρα αποκλειστικά μέσω του API gateway. Αρχικά ο παραγωγός καταχωρεί τα στοιχεία του αγροτεμαχίου και των αισθητήρων, ενώ έπειτα μπορεί να λάβει τα διαπιστευτήρια του κάθε αισθητήρα. Στη συνέχεια, το API δημιουργεί τα αντίστοιχα topics στο *Kafka*, συνδέει το αγροτεμάχιο με το κατάλληλο Arroyo pipeline και ενεργοποιεί τη ροή δεδομένων. Τέλος, το caching

layer ενημερώνει τα indices ώστε τα πρόσφατα δεδομένα να είναι διαθέσιμα αμέσως στο UI.



Σχήμα 5.3: Διάγραμα δραστηριότητας για τη διαδικασία ένταξης νέου πελάτη

6
Πειράματα

Η παρούσα ενότητα εξετάζει τη συμπεριφορά των μηχανισμών προσωρινής αποθήκευσης που υποστηρίζουν το endpoint */aggregate*, το οποίο αποτελεί βασικό συστατικό της πλατφόρμας *Nostradamus*. Η υπηρεσία αυτή υλοποιεί παραθυρικές συναθροίσεις σε χρονοσειρές αισθητήρων και λειτουργεί ως μεσαίο επίπεδο μεταξύ της εφαρμογής και της μόνιμης βάσης δεδομένων. Η καθυστέρηση ανάγνωσης και εγγραφής στο cache layer μεταφέρεται άμεσα στον χρόνο απόκρισης του API, επομένως η απόδοση του εκάστοτε υποστρώματος αποτελεί πρωτεύοντα παράγοντα για τη συνολική σταθερότητα της πλατφόρμας.

6.1 Διατύπωση στόχου

Στόχος της πειραματικής διαδικασίας είναι η αποτίμηση της απόδοσης και της σταθερότητας δύο πλήρως εναλλάξιμων cache drivers (*Valkey*, *Memcached*) όταν χρησιμοποιούνται από το */aggregate* υπό συνθήκες συνεχούς φόρτου. Η αξιολόγηση επικεντρώνεται σε τρεις παραμέτρους:

1. **Απόδοση ανάγνωσης και εγγραφής.** Εξετάζονται συμμετρικά τα μονοπάτια *cache hit* και *cache miss*, καθώς αμφότερα συνεισφέρουν στον τελικό χρόνο απόκρισης.

2. **Σταθερότητα σε υψηλά percentiles**. Για συστήματα πραγματικού χρόνου, η μέση τιμή είναι ανεπαρκής δείκτης· τα υψηλά percentiles (P95) είναι εκεί όπου εκδηλώνονται οι χρονικές αποκλίσεις που επηρεάζουν ουσιαστικά την εμπειρία χρήστη.

3. **Συμπεριφορά σε περιβάλλοντα συχνής ανανέωσης**. Η χαμηλή χρονική διάρκεια ζωής (TTL) σε χρονοσειρές αισθητήρων προκαλεί συχνές invalidations, επομένως ο driver πρέπει να ανταποκρίνεται σωστά σε workloads που χαρακτηρίζονται από επαναλαμβανόμενη εισαγωγή νέων τιμών.

Οι παραπάνω στόχοι αποτυπώνουν τις πραγματικές απαιτήσεις ενός αγροτικού deployment, όπου οι συσκευές παράγουν συνεχή ροή δεδομένων και το API πρέπει να διατηρεί σταθερή συμπεριφορά ακόμα και υπό μεταβαλλόμενο φόρτο.

6.2 Μεθοδολογια

Για την απομόνωση των χαρακτηριστικών κάθε υποστρώματος δημιουργήθηκαν δύο ανεξάρτητες instantiations της πλατφόρμας: μία που χρησιμοποιεί *Valkey* και μία που χρησιμοποιεί *Memcached*. Και οι δύο παραμετροποιήθηκαν μέσω του κοινού cache interface, ώστε η αλλαγή driver να μην επηρεάζει την εφαρμοστική λογική του */aggregate*.

Ο φόρτος παράχθηκε μέσω synthetic load generator, ο οποίος μιμείται το προφίλ αιτημάτων πραγματικού deployment: επαναλαμβανόμενα queries σε συγκεκριμένα παράθυρα (15 λεπτών έως 1 ώρας) για διαφορετικούς τύπους αισθητήρων, με σταθερό ρυθμό αποστολής. Η συλλογή των μετρήσεων πραγματοποιήθηκε μέσω *Prometheus*, ενώ η οπτικοποίηση έγινε στο *Grafana* με έμφαση στο P95 latency, στις write καθυστερήσεις και στο ποσοστό invalidations.

Η παραμετροποίηση των drivers παρέμεινε στην προεπιλεγμένη μορφή τους, εκτός από το TTL των εγγραφών που ορίστηκε στα 120 δευτερόλεπτα, ώστε να προσεγγίζει τον λειτουργικό ορίζοντα των μετρήσεων πεδίου.

6.3 Αποτελεσματα πειραματος

Τα αποτελέσματα καταδεικνύουν σαφή διαφοροποίηση ως προς τη συμπεριφορά των δύο υποστρωμάτων. Το *Memcached* εμφανίζει χαμηλότερη καθυστέρηση σε αναγνώσεις και εγγραφές, περίπου $1.7\times$ ταχύ-

τερο από το *Valkey* στο P95. Η επίδοση αυτή συνδέεται με τον λιτό χαρακτήρα του πρωτοκόλλου του (Memcached binary protocol), το οποίο αποφεύγει πολυπλοκότητα και επιτρέπει υψηλό throughput με περιορισμένο αποτύπωμα πόρων.

Αντίθετα, το *Valkey* υπερέχει ως προς τη χρονική σταθερότητα: οι καμπύλες latency παραμένουν σχεδόν σταθερές ακόμη και όταν αυξάνεται το concurrency, χωρίς την εμφάνιση αιχμών. Η συμπεριφορά αυτή είναι σημαντική για workloads που απαιτούν προβλεψιμότητα ή κάνουν χρήση λειτουργιών που απουσιάζουν από το *Memcached* (π.χ. atomic blocks, Lua scripts, σύνθετες δομές δεδομένων).

Ως εκ τούτου, τα δύο συστήματα δεν αποτελούν εναλλακτικές που διαφέρουν απλώς ποσοτικά, αλλά ποιοτικά: το *Memcached* επιτυγχάνει υψηλότερη ακατέργαστη ταχύτητα, ενώ το *Valkey* προσφέρει συνεπή συμπεριφορά ακόμη και σε περιόδους υψηλής συμφόρησης και διαθέτει επεκτασιμότητα που το καθιστά καταλληλότερο για μελλοντική ενσωμάτωση πιο σύνθετων λειτουργιών της πλατφόρμας.

Με δεδομένη τη modular αρχιτεκτονική, η επιλογή driver μπορεί να καθοδηγείται από τις επιχειρησιακές ανάγκες κάθε deployment, χωρίς καμία αλλαγή στον κώδικα της εφαρμογής.

6.4 Περαιτερω επεκτασεις

Η τρέχουσα πειραματική διαδικασία αποτελεί θεμέλιο για μελλοντική ανάλυση. Ως επεκτάσεις σχεδιάζονται:

- διερεύνηση της επίδρασης των invalidations σε διαφορετικούς ρυθμούς εισαγωγής νέων μετρήσεων,
- αποτύπωση κατανάλωσης μνήμης και CPU ανά driver,
- μελέτη συμπεριφοράς σε συνθήκες οριζόντιας κλίμακωσης,

Οι παραπάνω επεκτάσεις αποτελούν απαραίτητη συνέχεια για την πλήρη χαρτογράφηση της συμπεριφοράς των υποστρωμάτων και για την τεκμηρίωση των επιχειρησιακών αποφάσεων που θα καθορίσουν τη μελλοντική μορφή της πλατφόρμας.

7
Συμπεράσματα

Βιβλιογραφία

[1] Denise Ratasich, Faiq Khalid, Florian Geissler, Radu Grosu, Muhammad Shafique, and Ezio Bartocci. A roadmap toward the resilient internet of things for cyber-physical systems. *IEEE Access*, 2019.

[2] Godson Michael D'silva, Azharuddin Khan, Gaurav, and Siddhesh Bari. Real-time processing of iot events with historic data using apache kafka and apache spark with dashing framework. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, 2017.

[3] Constantinos Styliaras, Dimitrios Kyrkilis, and Symeon Semasis. *The Role of Agriculture in Economic Growth in Greece*. 2013.

[4] Anthony Callo and Mo Mansouri. Food security in global food distribution networks: A systems thinking approach. In *2024 IEEE International Systems Conference (SysCon)*, 2024.

[5] Hwiwon Lee, Sosun Kim, and Huy Kang Kim. Sok: Demystifying cyber resilience quantification in cyber-physical systems. In *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2022.

[6] Dr. Rashmi Sharma, Vishal Mishra, and Suryansh Srivastava. Enhancing crop yields through iot-enabled precision agriculture. In *2023 International Conference on Disruptive Technologies (ICDT)*, 2023.

[7] Mohammad Nishat Akhtar, Abdurrahman Javid Shaikh, Ambareen Khan, Habib Awais, Elmi Abu Bakar, and Abdul Rahim Othman. Smart sensing with edge computing in precision agriculture for soil assessment and heavy metal monitoring: A review. *Agriculture*, "2021".

[8] Hanzhe Li, Xiangxiang Wang, Yuan Feng, Yaqian Qi, and Jingxiao Tian. Driving intelligent iot monitoring and control through cloud computing and machine learning, 2024. URL *https://arxiv.org/abs/2403.18100*.

[9] Martin Kleppmann. *Designing Data-Intensive Applications*. O'Reilly Media, 2017.

[10] Mario Lezoche and Hervé Panetto. Cyber-physical systems, a new formal paradigm to model redundancy and resiliency. 2018. URL *http://arxiv.org/abs/1810.06911*.

[11] Rishika Shree, Tanupriya Choudhury, Subhash Chand Gupta, and Praveen Kumar. Kafka: The modern platform for data management and analysis in big data domain. In *2017 2nd International Conference on Telecommunication and Networks (TEL-NET)*, 2017.

[12] Chung Lai Deryck Ho, Chung–Horng Lung, and Zhengding Mao. Comparative analysis of real-time data processing architectures: Kafka versus mqtt broker in iot. In *2024 IEEE 4th International Conference on Electronic Communications, Internet of Things and Big Data (ICEIB)*, 2024.

[13] Neha Narkhede Jay Kreps and Jun Rao. Kafka: a distributed messaging system for log processing. In *Proceedings of the NetDB Workshop*, 2011. URL *https://api.semanticscholar.org/CorpusID:18534081*.

[14] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 2010.

[15] Arbër Sh. Perçuku, Daniela V. Minkovska, Lyudmila Y. Stoyanova, and Arta E. Abdullahu. Iot using raspberry pi and apache cassandra on pv solar system. In *2020 XXIX International Scientific Conference Electronics (ET)*, 2020.

[16] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung, and Venkateshwaran Venkataramani. Scaling memcache at facebook. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 2013.

[17] Josiah Carlson. *Redis in Action*. Manning Publications, 2013.

[18] Sneha M. Pragathi B.C., Hrithik Maddirala. Implementing an effective infrastructure monitoring solution with prometheus and grafana. *International Journal of Computer Applications*, 2024.