



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

Διπλωματική Εργασία

Ανάπτυξη πλατφόρμας διαχείρισης
δεδομένων IoT με έμφαση στην απόδοση,
παρατηρησιμότητα, επεκτασιμότητα και
αυτοϊαση

Επιμέλεια:
Δημήτριος Ντέντας
ΑΕΜ: 10446

Επίβλεψη:
Καθ. Συμεωνίδης Ανδρέας
Δρ. Κωνσταντίνος Παναγιώτου
Υ. Δ. Γιώργος Σιαχάμης

Θεσσαλονίκη, Φεβρουάριος 2026

Περίληψη

Η παρούσα διπλωματική εργασία εστιάζει στον σχεδιασμό, την υλοποίηση και την πειραματική αξιολόγηση μιας ανθεκτικής, cloud-native υποδομής δεδομένων IoT για πλατφόρμες γεωργίας ακριβείας. Με δεδομένη την κρισιμότητα των Κυβερνοφυσικών Συστημάτων (CPS) στον αγροδιατροφικό τομέα, η έρευνα επικεντρώνεται στην ενσωμάτωση τριών κρίσιμων πυλώνων: της προσωρινής αποθήκευσης (caching), της παρατηρησιμότητας (observability) και των μηχανισμών αυτοϊασης (self-healing).

Η αρχιτεκτονική της πλατφόρμας βασίζεται στο Kubernetes και αξιοποιεί ένα σύγχρονο data stack αποτελούμενο από Apache Kafka για τη διαχείριση ροών, ScyllaDB για την αποθήκευση χρονοσειρών και Arroyo για την επεξεργασία streaming δεδομένων. Η παρατηρησιμότητα του συστήματος ενισχύθηκε μέσω ενός πλήρους stack (Prometheus, Grafana, Loki, Tempo), το οποίο επέτρεψε την ιχνηλάτηση (tracing) των αιτημάτων από το API layer έως το επίπεδο αποθήκευσης, διευκολύνοντας την ανάλυση βαθύτερης αιτίας (Root Cause Analysis).

Η πειραματική διαδικασία επικεντρώθηκε στην αξιολόγηση εναλλακτικών cache drivers (Valkey, Memcached) υπό ρεαλιστικό φόρτο 350 RPS. Τα αποτελέσματα ανέδειξαν ένα κρίσιμο trade-off: ενώ το Memcached πέτυχε χαμηλότερη μέση καθυστέρηση (P95 read στα 486 μ s), το Valkey επέδειξε ανώτερη σταθερότητα στα υψηλά percentiles (P99), διατηρώντας το tail latency κάτω από τα 5.6 ms ακόμη και σε συνθήκες υποβέλτιστης πυκνότητας δεδομένων (data granularity). Επιπλέον, διαπιστώθηκε ότι η αυξημένη συχνότητα μετρήσεων χωρίς αντίστοιχη πληροφοριακή αξία μπορεί να επιβαρύνει το cache layer έως και 56%.

Τα συμπεράσματα της εργασίας τεκμηριώνουν ότι η συνδυαστική χρήση stateful επεξεργασίας ροών και προηγμένης παρατηρησιμότητας επιτρέπει την ανάπτυξη συστημάτων που όχι μόνο αποδίδουν σε υψηλή κλίμακα, αλλά διαθέτουν την απαιτούμενη διαγνωστική ικανότητα για την υποστήριξη στρατηγικών self-healing και multi-cloud ανάπτυξης.

Abstract

This master thesis investigates the design, implementation, and experimental evaluation of a resilient, cloud-native IoT data infrastructure for the precision agriculture platforms. Given the critical nature of Cyber-Physical Systems (CPS) in the agrifood sector, this research focuses on the integration of three key pillars: distributed caching, full-stack observability, and self-healing mechanisms.

The platform architecture is built on Kubernetes, utilizing a modern data stack comprising Apache Kafka for stream management, ScyllaDB for time-series storage, and Arroyo for real-time stream processing. System-level observability was enhanced through a comprehensive stack (Prometheus, Grafana, Loki, Tempo), enabling end-to-end request tracing and facilitating rapid Root Cause Analysis (RCA) across the infrastructure.

Experimental evaluation focused on the comparative performance of interchangeable cache drivers (Valkey, Memcached) under a realistic load of 350 RPS. The results revealed a significant architectural trade-off: while Memcached provided lower average latency (P95 read at 486 μ s), Valkey demonstrated superior stability in higher percentiles (P99), maintaining tail latency below 5.6 ms even under suboptimal data granularity conditions. Furthermore, it was observed that excessive measurement frequency without proportional information value could increase cache write latency by up to 56%.

The findings of this thesis demonstrate that the synergy of stateful stream processing and advanced observability enables the development of IoT infrastructures that are not only performant at scale but also possess the diagnostic depth required to support autonomous recovery and multi-cloud resilience strategies.

Ευχαριστίες

Η παρακάτω διπλωματική εργασία σηματοδοτεί το τέλος της ακαδημαϊκής μου καριέρας, καθώς και το άνοιγμα των πυλών προς το επόμενο στάδιο της ζωής μου. Σε αυτό το ταξίδι αποκόμισα γνώσεις που θα με συνοδεύσουν στην επαγγελματική μου σταδιοδρομία, αλλά και θα συνεισφέρουν στον απώτερο στόχο μου: την ολοκλήρωσή μου ως άνθρωπος. Σε αυτό το σταυροδρόμι έφτασα με την αξία μου και τον προσωπικό μου κόπο· παρ' όλ' αυτά, οι άνθρωποι που υπήρξαν πλάι μου φέρουν και εκείνοι το δικό τους μερίδιο στον αγώνα μου.

Αρχικά, θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου, κ. **Ανδρέα Συμεωνίδη**, για την εμπιστοσύνη του, τον χρόνο του και την καθοδήγησή του σε αυτό το υπερσύγχρονο θέμα που προτίστως κίνησε το ενδιαφέρον μου. Θα ήθελα επίσης να ευχαριστήσω τον διδάκτορα **Κωνσταντίνο Παναγιώτου** και τον υποψήφιο διδάκτορα **Γιώργο Σιαχάμη** για την συνεισφορά και τη συνεχή υποστήριξη τους σε κάθε στάδιο της εργασίας.

Η στήριξη της οικογένειάς μου σε αυτό το ταξίδι υπήρξε ο ακρογωνιαίος λίθος της προσπάθειάς μου. Στους γονείς μου, **Κλεοπάτρα** και **Κωνσταντίνο**, οφείλω ένα βαθύ ευχαριστώ για τις θυσίες τους, την αμέριστη συμπαράσταση και την πίστη που έδειξαν στις δυνατότητές μου από τα πρώτα μου βήματα μέχρι και σήμερα. Η ηθική και έμπρακτη ενθάρρυνσή τους ήταν το κίνητρο για να ξεπεράσω κάθε δυσκολία. Στην αδελφή μου, **Ελίζα**, θέλω να πω ένα ξεχωριστό ευχαριστώ. Παρά το γεγονός ότι πολλές φορές καταφέρνει να γίνεται ο πιο επίμονος περυσπασμός μου, η σταθερή της παρουσία και η προθυμία της να με βοηθήσει υπήρξαν πολύτιμες.

Τέλος, ευχαριστώ τους φίλους μου που μοιράστηκαν μαζί μου τις προκλήσεις της φοιτητικής ζωής, κάνοντας αυτή τη διαδρομή αξιομνημόνευτη. Ιδιαίτερες ευχαριστίες αξίζουν σε εκείνους που υπήρξαν κάτι παραπάνω από απλοί συμφοιτητές: στους ανθρώπους με τους οποίους μοιράστηκα το κοινό όραμα της δημιουργίας και της καινοτομίας.

Περιεχόμενα

Περίληψη	1
Abstract	2
Ευχαριστίες	3
1 Εισαγωγή	8
1.1 Πλαισίωση του Προβλήματος	8
1.2 Κίνητρο και Ερευνητικά Ερωτήματα	9
1.3 Σκοπός και Αντικείμενο της Εργασίας	9
1.4 Συνεισφορά της Εργασίας	10
1.5 Δομή της Εργασίας	11
2 Επισκόπηση της Ερευνητικής Περιοχής	12
3 Θεωρητικό Υπόβαθρο	17
3.1 Κυβερνο-Φυσικά Συστήματα: Ορισμοί και Χαρακτηριστικά	17
3.1.1 Θεμελιώδεις Ορισμοί	17
3.1.2 Διαστάσεις Πολυπλοκότητας στα CPS	18
3.1.3 Μοντέλα Αναφοράς για CPS	18
3.2 Αρχιτεκτονικές Μοτίβα για Κατανεμημένα CPS	20
3.2.1 Layered Architecture (Στρωματοποιημένη Αρχιτε- κτονική)	20
3.3 Streams, Event-Driven και Batch vs Real-Time Processing .	21
3.4 Αρχιτεκτονικές Streams: Lambda και Kappa	22
3.5 Διασύνδεση Υποσυστημάτων	24
3.6 Αποθήκευση, Consistency Models και Caching	25
3.7 Διαχείριση Υποδομής και Cloud-Native Patterns	27
4 Εργαλεία	29
4.1 Kafka	30
4.2 Stream Processing	31

4.2.1	Flink	32
4.2.2	Arroyo	32
4.3	Distributed Storage	33
4.3.1	Cassandra	33
4.3.2	ScyllaDB	34
4.4	Caching	36
4.4.1	Redis	36
4.4.2	Valkey	37
4.4.3	DragonflyDB	38
4.4.4	Memcached	38
4.5	Kubernetes	38
5	Υλοποιήσεις	41
5.1	Αρχιτεκτονική υποδομής	42
5.1.1	Τοπολογία και ρόλοι κόμβων	44
5.1.2	Αυτοματισμοί υποδομής και διακυβέρνηση	45
5.2	Ροή δεδομένων σε πραγματικό χρόνο	46
5.2.1	Πηγές ροής δεδομένων	46
5.2.2	Τεχνολογίες Streaming	47
5.3	Προσωρινή αποθήκευση και cache drivers	49
5.3.1	Αρχιτεκτονική και ενσωμάτωση	49
5.3.2	Επιλογή και συγκριτική αξιολόγηση υποστρωμάτων	49
5.3.3	Πειραματικές υποθέσεις	50
5.3.4	Στρατηγικές caching	51
5.4	Επίπεδο υπηρεσιών και εφαρμογής	53
5.4.1	Valkey cluster	54
5.4.2	Διαχείριση Arroyo pipelines	54
5.4.3	Ανθεκτικότητα και ανοχή σε σφάλματα	55
5.4.4	Υπηρεσία /aggregate και caching layer	55
5.4.5	Frontend και διεπαφές παρακολούθησης	56
5.5	Παρατηρησιμότητα	58
5.5.1	Μετρικές και SLI παρακολούθησης	59
5.5.2	Μετρικές και SLI παρακολούθησης	60
5.5.3	Καταγραφές και διασταύρωση συμβάντων	61
5.5.4	Ιχνηλασιμότητα και ανάλυση αιτημάτων	61
5.6	Διεπαφή Χρήστη	63
6	Πειράματα	65
6.1	Πειραματικό Πλαίσιο και Κοινή Μεθοδολογία	65
6.1.1	Στόχος και Κριτήρια Αξιολόγησης	66
6.1.2	Μεθοδολογία και Παραμετροποίηση	66

6.2	Πείραμα 1: Βασική Σύγκριση Απόδοσης και Σταθερότητας	
	Cache Drivers	67
6.2.1	Υπόθεση Πειράματος	67
6.2.2	Σχέδιο Πειράματος	67
6.2.3	Δεδομένα Εισόδου	67
6.2.4	Εκτέλεση Δοκιμής	67
6.2.5	Αποτελέσματα Πειράματος	67
6.2.6	Συμπεράσματα Πειράματος	68
6.3	Πείραμα 2: Επίδραση Υποβέλτιστης Χρονικής Πυκνότητας	
	Δεδομένων	68
6.3.1	Υπόθεση πειράματος	68
6.3.2	Σχέδιο Πειράματος	68
6.3.3	Δεδομένα Εισόδου	68
6.3.4	Εκτέλεση Δοκιμής	69
6.3.5	Αποτελέσματα Πειράματος	69
6.3.6	Συμπεράσματα Πειράματος	69
6.4	Συνολικά Αποτελέσματα	70
6.5	Προεκτάσεις	71
7	Συμπεράσματα	72
7.1	Αρχιτεκτονική Ανθεκτικότητα και Επεξεργασία Ροών	72
7.2	Βελτιστοποίηση Απόδοσης και Διαχείριση Cache	73
7.3	Ολοκληρωμένο Stack Παρατηρησιμότητας	73
7.4	Μελλοντική Εργασία και Ερευνητικές Προεκτάσεις	74
7.4.1	Αρχιτεκτονική Αυτονομία και Δυναμικό Self-Healing	74
7.4.2	Ευφυής Διαχείριση Cache και Μηχανική Μάθηση .	74
7.4.3	Edge-to-Cloud και Γεωγραφική Κλιμάκωση	75
	Βιβλιογραφία	76

ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ

Ο παρακάτω πίνακας περιλαμβάνει μία συνοπτική λίστα με τα τεχνικά ακρωνύμια και συντομογραφίες που χρησιμοποιούνται σε αυτό το έγγραφο.

API	→ Application Programming Interface
CoAP	→ Constrained Application Protocol
CPS	→ Cyber-Physical System
CPU	→ Central Processing Unit
DSL	→ Domain Specific Language
EFK	→ Elasticsearch, FluentBit, Kibana
ELK	→ Elasticsearch, Logstash, Kibana
GC	→ Garbage Collection
gRPC	→ gRPC Remote Procedure Call
I/O	→ Input/Output
IoT	→ Internet of Things
JSON	→ JavaScript Object Notation
JVM	→ Java Virtual Machine
LRU	→ Least Recently Used
MQTT	→ Message Queuing Telemetry Transport
MVP	→ Minimum Viable Product
OTLP	→ Open Telemetry Protocol
P95	→ 95th Percentile
QoS	→ Quality of Service
RBAC	→ Role-Based Access Control
RPS	→ Requests Per Second
SLI	→ Service Level Indicator
TTL	→ Time To Live
UI	→ User Interface

1

Εισαγωγή

1.1 ΠΛΑΙΣΙΩΣΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ

Στην αυγή της τέταρτης βιομηχανικής επανάστασης, ο πρωτογενής τομέας μετασχηματίζεται σε ένα πεδίο έντασης δεδομένων. Η ενσωμάτωση του Διαδικτύου των Πραγμάτων (IoT) και των Κυβερνοφυσικών Συστημάτων (CPS) υπόσχεται τη βελτιστοποίηση των πόρων, ωστόσο δημιουργεί μια κρίσιμη τεχνική αντίφαση: **την εξάρτηση ζωτικών λειτουργιών από εγγενώς ασταθείς ψηφιακές υποδομές.**

Το πρόβλημα στο οποίο εστιάζει η παρούσα εργασία εντοπίζεται στην **αρχιτεκτονική ευθραυστότητα** των υπαρχουσών πλατφορμών IoT. Ενώ η συλλογή δεδομένων έχει επιλυθεί σε μεγάλο βαθμό, οι σύγχρονες υποδομές υστερούν στους εξής τρεις άξονες:

1. **Συμφόρηση και Καθυστέρηση (Bottlenecks):** Η εκθετική αύξηση του όγκου των δεδομένων από αισθητήρες πεδίου προκαλεί καθυστερήσεις στην επεξεργασία σε πραγματικό χρόνο, καθιστώντας τα συστήματα ανίκανα να ανταποκριθούν σε κρίσιμα γεγονότα (π.χ. ανάγκη άμεσης άρδευσης).
2. **Έλλειψη Εσωτερικής Ορατότητας (Observability Gap):** Τα περισσότερα συστήματα λειτουργούν ως "μαύρα κουτιά". Όταν παρουσιάζεται μια δυσλειτουργία, η διάγνωση της αιτίας είναι εξαι-

1.2. ΚΙΝΗΤΡΟ ΚΑΙ ΕΡΕΥΝΗΤΙΚΑ ΕΡΩΤΗΜΑΤΑ

ρετικά δύσκολη, οδηγώντας σε παρατεταμένους χρόνους εκτός λειτουργίας (downtime).

3. **Αδυναμία Αυτόνομης Ανάκαμψης:** Σε απομακρυσμένες αγροτικές περιοχές, η φυσική παρέμβαση για την επιδιόρθωση σφαλμάτων λογισμικού είναι κοστοβόρα ή αδύνατη. Η έλλειψη μηχανισμών αυτοϊάσης (self-healing) σημαίνει ότι ένα μεμονωμένο σφάλμα μπορεί να θέσει εκτός λειτουργίας ολόκληρο το δίκτυο παρακολούθησης.

Συνεπώς, το κεντρικό πρόβλημα δεν είναι η έλλειψη δεδομένων, αλλά η **έλλειψη ανθεκτικότητας και παρατηρησιμότητας** των συστημάτων που τα διαχειρίζονται, γεγονός που θέτει σε κίνδυνο την επιχειρησιακή συνέχεια και την επισιτιστική ασφάλεια.

1.2 ΚΙΝΗΤΡΟ ΚΑΙ ΕΡΕΥΝΗΤΙΚΑ ΕΡΩΤΗΜΑΤΑ

Η παρούσα εργασία ελλοχεύει από την παρατήρηση ότι οι υπάρχουσες *IoT* πλατφόρμες συχνά θυσιάζουν την παρατηρησιμότητα (observability) και την ανθεκτικότητα στον βωμό της ταχείας ανάπτυξης. Ιδιαίτερα για την ελληνική πραγματικότητα, όπου ο αγροτικός τομέας απαιτεί λύσεις χαμηλού κόστους αλλά υψηλής αξιοπιστίας, η έρευνα γύρω από βελτιστοποιημένες *cloud-native* αρχιτεκτονικές καθίσταται επιτακτική.

Τα κεντρικά ερωτήματα που εξετάζονται είναι:

- Μπορεί μια κατανεμημένη αρχιτεκτονική να διασφαλίσει τη συνέχεια της ροής δεδομένων υπό συνθήκες υψηλού φόρτου ή αστοχίας;
- Με ποιον τρόπο η ενσωμάτωση μηχανισμών *caching* και *self-healing* επηρεάζει το *latency* και την ολική απόδοση ενός συστήματος;
- Ποιοι είναι οι βέλτιστοι τρόποι διαχείρισης της ετερογένειας των δεδομένων σε κρίσιμες υποδομές, όπως για παράδειγμα δίκτυα ύδρευσης και ενέργειας;

1.3 ΣΚΟΠΟΣ ΚΑΙ ΑΝΤΙΚΕΙΜΕΝΟ ΤΗΣ ΕΡΓΑΣΙΑΣ

Η παρούσα διπλωματική εργασία εστιάζει στην ανάπτυξη και βελτιστοποίηση μιας ολοκληρωμένης αρχιτεκτονικής για την παροχή προηγμένων υπηρεσιών παρατήρησης και λήψης αποφάσεων στον τομέα της

έξυπνης γεωργίας και της επισιτιστικής ασφάλειας. Η πλατφόρμα φιλοξενεί την κρίσιμη υποδομή "IoT Observability", η οποία λειτουργεί ως ο κεντρικός πυλώνας συλλογής και επεξεργασίας δεδομένων από ένα ευρύ δίκτυο IoT και Edge συσκευών.

Κεντρικός σκοπός της εργασίας δεν είναι μόνο η απλή παρακολούθηση της υποδομής, αλλά η ολιστική αρχιτεκτονική αναβάθμιση μέσω της ενσωμάτωσης τριών αλληλένδετων μηχανισμών:

- **Caching:** Για τη βελτιστοποίηση της απόδοσης και την εξάλειψη των σημείων συμφόρησης κατά την ανάκτηση σύνθετων συναθροίσεων.
- **Παρατηρησιμότητα συστήματος:** Για τη δημιουργία μιας διαφανούς υποδομής που επιτρέπει τη βαθιά διάγνωση της κατάστασης του συστήματος σε πραγματικό χρόνο.
- **Αυτοϊαση:** Για την παροχή προληπτικής ικανότητας ανάκαμψης από αστοχίες, διασφαλίζοντας την επιχειρησιακή συνέχεια.

Το αντικείμενο της μελέτης επικεντρώνεται στη μεταστοιχείωση της πλατφόρμας από μια στατική υποδομή σε ένα ανθεκτικό Κυβερνοφυσικό Σύστημα (CPS), ικανό να υποστηρίξει κρίσιμες εφαρμογές γεωργίας ακριβείας υπό συνθήκες υψηλού φόρτου και μεταβλητότητας.

1.4 ΣΥΝΕΙΣΦΟΡΑ ΤΗΣ ΕΡΓΑΣΙΑΣ

Η συνεισφορά της παρούσας διπλωματικής εργασίας επικεντρώνεται στον σχεδιασμό, την υλοποίηση και την αξιολόγηση ενός διευρυμένου αρχιτεκτονικού πλαισίου για συστήματα μεγάλης κλίμακας. Συγκεκριμένα:

- **Αρχιτεκτονική Προσέγγιση:** Προτείνεται ένα πολυεπίπεδο μοντέλο που ενσωματώνει κατανεμημένη προσωρινή αποθήκευση (distributed caching) για τη μείωση της συμφόρησης.
- **Μηχανισμοί Ανθεκτικότητας:** Υλοποιούνται πρωτόκολλα αυτόματης διάγνωσης και ανάκαμψης, βασισμένα στο πλαίσιο R4 (Redundancy, Robustness, Resourcefulness, Rapidity).
- **Αξιολόγηση σε Πραγματικές Συνθήκες:** Αναλύεται η επίδραση των προτεινόμενων λύσεων στην απόδοση του συστήματος, αναδεικνύοντας τις αναγκαίες συμβιβαστικές λύσεις (trade-offs) μεταξύ πολυπλοκότητας και απόδοσης.

- **Ενίσχυση Παρατηρησιμότητας:** Προτείνεται μια *end-to-end* λύση/πλατφόρμα υποστήριξης της κύριας υποδομής, με βασικό στόχο τη συνεχή διάγνωση της κατάστασης (υγείας) του συστήματος, μέσω της συλλογής μετρικών όλων των ειδών.

1.5 ΔΟΜΗ ΤΗΣ ΕΡΓΑΣΙΑΣ

Η παρούσα διπλωματική εργασία διαρθρώνεται σε επτά κεφάλαια, ακολουθώντας μια λογική πορεία από τη θεωρητική θεμελίωση στην τεχνική υλοποίηση και την πειραματική επαλήθευση:

- **Κεφάλαιο 2 - Επισκόπηση της Ερευνητικής Περιοχής:** Παρουσιάζεται το πλαίσιο των Κυβερνοφυσικών Συστημάτων (CPS) στη σύγχρονη γεωργία και αναλύονται οι έννοιες της ανθεκτικότητας και της επισιτιστικής ασφάλειας.
- **Κεφάλαιο 3 - Θεωρητικό Υπόβαθρο:** Εξετάζονται οι αρχιτεκτονικές καταναεμημένων συστημάτων, τα μοντέλα επεξεργασίας ροών δεδομένων, καθώς και οι αρχές του *cloud-native computing* και των μοντέλων συνέπειας.
- **Κεφάλαιο 4 - Εργαλεία:** Γίνεται αναλυτική παρουσίαση των τεχνολογιών που χρησιμοποιήθηκαν, συγκεκριμένα για τη ροή δεδομένων, για την αποθήκευση, καθώς για το επίπεδο προσωρινής αποθήκευσης/ταχείας προσπέλασης.
- **Κεφάλαιο 5 - Υλοποιήσεις:** Αποτελεί τον κεντρικό πυλώνα της εργασίας, όπου περιγράφεται η αρχιτεκτονική υποδομή, η μετακίνηση των δεδομένων μεταξύ των υποσυστημάτων, και η ενσωμάτωση των επιπέδων παρατηρησιμότητας και ανθεκτικότητας.
- **Κεφάλαιο 6 - Πειράματα:** Παρουσιάζεται η μεθοδολογία και η εκτέλεση των δοκιμών. Περιλαμβάνει τη σύγκριση απόδοσης των πειραμάτων και τη μελέτη της επίδρασης της χρονικής πυκνότητας των δεδομένων στην ευστάθεια του συστήματος.
- **Κεφάλαιο 7 - Συμπεράσματα:** Ανακεφαλαιώνονται τα ευρήματα της έρευνας, αξιολογείται η επίτευξη των στόχων και προτείνονται κατευθύνσεις για μελλοντικές επεκτάσεις του συστήματος.

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 αυτοϊαση 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 προϊόντων.

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

συμβάλλει 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).

Η κατανόηση της ανθεκτικότητας είναι κρίσιμη για τη σχεδίαση συστημάτων, κυρίως αυτών μεγάλης κλίμακας. Είναι σημαντικό να διαχωριστεί η έννοια της ανοχής σφαλμάτων (fault tolerance) από την ανθεκτικότητα (resilience). Ενώ η ανοχή σφαλμάτων στοχεύει στην αποτροπή της εμφάνισης σφαλμάτων μέσω πλεονασμού (redundancy), η ανθεκτικότητα, όπως ορίζεται από τους Hollnagel et al. [6], αποτελεί μια προληπτική ικανότητα του συστήματος να λειτουργεί υπό συνθήκες πίεσης και να επανέρχεται. Οι μηχανισμοί αυτοί, όπως η αυτόματη διάγνωση και η δυναμική ανάκαμψη, δεν αφορούν μόνο την επιβίωση

σε απρόβλεπτες συνθήκες, αλλά και τη διατήρηση της ποιότητας της λειτουργίας. Αυτό καθιστά την ανθεκτικότητα έναν πιο ολιστικό και απαιτητικό στόχο σχεδίασης για κρίσιμες υποδομές.

Βάσει της μελέτης των Sharma et al. [7] και πρόσφατων ανασκοπήσεων πάνω στην ενσωμάτωση IoT και AI στη γεωργία [8], 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. [9] 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 καλλιέργειας. 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 [10].

Μια θεμελιώδης αρχιτεκτονική πρόκληση στον αγροτικό τομέα είναι η ετερογένεια των δεδομένων (Data Heterogeneity) και η διαλειτουργικότητα (Interoperability). Οι συσκευές IoT, ιδιαίτερα σε αγροτικά περιβάλλοντα, συχνά χρησιμοποιούν διαφορετικά πρωτόκολλα επικοινωνίας και παράγουν δεδομένα σε ποικίλες μορφές, γεγονός που περιπλέκει την ενοποίηση και την αξιοποίησή τους. Οι Roccatello et al. [11] επισημαίνουν ότι η απουσία ενιαίων προτύπων καθιστά τη διαδικασία ενοποίησης δύσκολη και δαπανηρή, υπογραμμίζοντας την ανάγκη για συντακτική και

σημασιολογική διαλειτουργικότητα. Συνεπώς, ένα σύγχρονο σύστημα πρέπει να περιλαμβάνει ένα στρώμα μετατροπής και κανονικοποίησης των δεδομένων (data normalization layer), ικανό να διαχειρίζεται την ετερογένεια πριν την κεντρική αποθήκευση, ενισχύοντας την ποιότητα της ανάλυσης και της λήψης αποφάσεων.

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 ασφάλη, επεκτάσιμη μετάδοση δεδομένων, προκειμένου να επιτευχθεί πλήρης αυτοματοποίηση και ευστάθεια λειτουργίας σε ετερογενή, διασυνδεδεμένα περιβάλλοντα πεδίου.

Μια θεμελιώδης αρχιτεκτονική πρόκληση στον αγροτικό τομέα είναι η ετερογένεια των δεδομένων (data heterogeneity) και η διαλειτουργικότητα (interoperability). Οι συσκευές IoT, ιδιαίτερα σε αγροτικά περιβάλλοντα, συχνά χρησιμοποιούν διαφορετικά πρωτόκολλα επικοινωνίας και παράγουν δεδομένα σε ποικίλες μορφές, γεγονός που περιπλέκει την ενοποίηση και την αξιοποίησή τους. Η χρήση κατακεντρωμένης προσωρινής αποθήκευσης (distributed caching) μπορεί να αυξήσει σημαντικά το throughput και να μειώσει το latency, λειτουργώντας ως «insulation layer» έναντι της συμφόρτωσης της μόνιμης βάσης δεδομένων. [12, 13]

Ωστόσο, η ενσωμάτωση caching εισάγει το πρόβλημα της ανανέωσης/απόσβεσης της cache (cache invalidation). Σε συστήματα χρονοσειρών με υψηλό ρυθμό εισαγωγής μετρήσεων, η διαχείριση της συνοχής μεταξύ cache και μόνιμης αποθήκευσης είναι κρίσιμη· τα μοτίβα όπως Cache-Aside προσφέρουν ευελιξία αλλά μετατοπίζουν την ευθύνη στην εφαρμογή για explicit invalidation/expiry. [14, 15, 16]

Με στόχο, λοιπόν, την αρχιτεκτονική ενίσχυση μιας σύγχρονης IoT υποδομής μεγάλης κλίμακας, 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 κατακεντρωμένα συστήματα, η προσθήκη μιας νέας υπηρεσίας ή λειτουργικότητας συνοδεύεται συχνά από μη προβλέψιμες επιπτώσεις στην απόδοση του συστήματος, όπως αναφέρει επανειλημμένα ο Kleppmann [17]. 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 επιλογή σχετικών τεχνολογιών, αλλά κυρίως στη συνεκτική και δυναμική εννορχήστρωσή 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 ΚΥΒΕΡΝΟ-ΦΥΣΙΚΑ ΣΥΣΤΗΜΑΤΑ: ΟΡΙΣΜΟΙ ΚΑΙ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ

3.1.1 Θεμελιώδεις Ορισμοί

Τα **Κυβερνο-Φυσικά Συστήματα** (Cyber-Physical Systems, CPS) αποτελούν ενσωματωμένα υπολογιστικά συστήματα που παρακολουθούν και ελέγχουν φυσικές διεργασίες μέσω στενής αλληλεπίδρασης μεταξύ κυβερνητικών (υπολογιστικών) και φυσικών στοιχείων. Σε αντίθεση με τα παραδοσιακά ενσωματωμένα συστήματα, τα CPS χαρακτηρίζονται από τα παρακάτω σημεία:

- **Κλειστοί βρόχοι ελέγχου (closed-loop control):** Η έξοδος του φυσικού συστήματος τροφοδοτείται πίσω στο υπολογιστικό σύ-

στημα, δημιουργώντας συνεχή κύκλο αισθητήρων-επεξεργασίας-ενεργοποιητών.

- **Αλληλεπίδραση πολλαπλών κλιμάκων:** Τα CPS λειτουργούν σε διαφορετικές χωρικές (από νανο-αισθητήρες έως περιφερειακά δίκτυα) και χρονικές (από μικροδευτερόλεπτα έως μήνες) κλίμακες.
- **Δικτυωμένη λειτουργία:** Η δυνατότητα επικοινωνίας και συντονισμού μεταξύ κατανεμημένων στοιχείων είναι εγγενής στον σχεδιασμό τους.
- **Χρονικοί περιορισμοί:** Πολλές λειτουργίες απαιτούν εγγυήσεις πραγματικού χρόνου (hard/soft real-time constraints).

3.1.2 Διαστάσεις Πολυπλοκότητας στα CPS

Η πολυπλοκότητα των CPS προκύπτει από την αλληλεπίδραση πολλών παραγόντων:

Ετερογένεια: Τα CPS ενσωματώνουν ετερογενή υπολογιστικά μοντέλα (event-driven, time-triggered, dataflow), πρωτόκολλα επικοινωνίας (MQTT, CoAP, HTTP) και φυσικά υποσυστήματα (μηχανικά, ηλεκτρικά, χημικά συστήματα).

Μη-ντετερμινισμός: Οι φυσικές διεργασίες εμφανίζουν στοχαστική συμπεριφορά, θόρυβο μετρήσεων και καθυστερήσεις επικοινωνίας, καθιστώντας τον προγραμματισμό απόλυτα καθοριστικών συμπεριφορών αδύνατο.

Κατανεμημένη κατάσταση: Η κατάσταση του συστήματος κατανέμεται σε πολλαπλούς κόμβους, απαιτώντας πρωτόκολλα συγχρονισμού και μηχανισμούς συνέπειας (consistency protocols).

3.1.3 Μοντέλα Αναφοράς για CPS

Διάφορα ερευνητικά και βιομηχανικά μοντέλα αναφοράς έχουν προταθεί για την οργάνωση της αρχιτεκτονικής των CPS:

Τριεπίπεδο Μοντέλο (Three-Tier Model): Η τριεπίπεδη αρχιτεκτονική CPS προτείνεται ως εννοιολογικό πλαίσιο μοντελοποίησης που διαχωρίζει το σύστημα σε τρία σαφώς ορισμένα επίπεδα, επιτρέποντας ιεραρχική ανάλυση και απομόνωση ευθυνών [18]. Ο διαχωρισμός αυτός δεν αποτελεί απλώς τεχνική ταξινόμηση, αλλά μεθοδολογικό εργαλείο σχεδίασης και επαλήθευσης.

- **Φυσικό Επίπεδο (Physical Layer):** Περιλαμβάνει τα φυσικά υποσυστήματα, αισθητήρες και ενεργοποιητές, καθώς και τις δυναμικές διεργασίες που περιγράφονται μέσω φυσικών νόμων. Το επίπεδο αυτό χαρακτηρίζεται από συνεχή χρονική εξέλιξη (continuous dynamics).
- **Επίπεδο Δικτύου (Network Layer):** Υλοποιεί την επικοινωνία, τον συγχρονισμό και τη μεταφορά δεδομένων μεταξύ καταναμεμημένων στοιχείων. Εδώ εισάγονται καθυστερήσεις, στοχαστικότητα μετάδοσης και ζητήματα αξιοπιστίας.
- **Κυβερνητικό Επίπεδο (Cyber Layer):** Περιλαμβάνει αλγορίθμους ελέγχου, λογικές αποφάσεων και υπολογιστικά μοντέλα. Σε αυτό το επίπεδο η συμπεριφορά είναι διακριτή (discrete computation) και βασίζεται σε επεξεργασία γεγονότων ή δεδομένων.

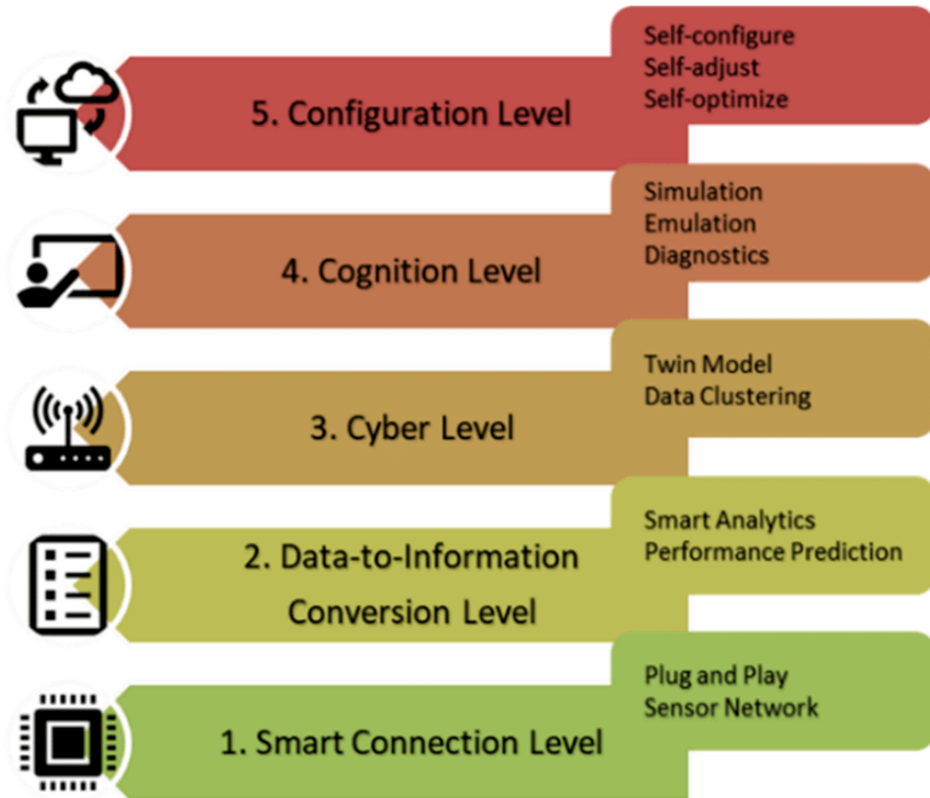
Ο σαφής αυτός διαχωρισμός επιτρέπει τη συνδυαστική ανάλυση συνεχών και διακριτών μοντέλων (hybrid systems), καθιστώντας δυνατή την τυπική επαλήθευση ιδιοτήτων ασφάλειας, χρονικής ορθότητας και σταθερότητας σε σύνθετα CPS.

Η τριεπίπεδη αυτή προσέγγιση αποτελεί τη βασική αφαιρετική δομή πάνω στην οποία αναπτύχθηκαν μεταγενέστερα πολυεπίπεδα μοντέλα, όπως το 5C, τα οποία επεκτείνουν το κυβερνητικό επίπεδο εισάγοντας έννοιες γνωσιακής επεξεργασίας και αυτο-διαμόρφωσης.

Πενταεπίπεδη Αρχιτεκτονική 5C (5C Architecture): Το μοντέλο 5C επεκτείνει το τριεπίπεδο μοντέλο με:

1. **Connection Layer:** Έξυπνη συλλογή και μετάδοση δεδομένων
2. **Conversion Layer:** Εξαγωγή πληροφορίας από δεδομένα
3. **Cyber Layer:** Δημιουργία ψηφιακών δίδυμων (digital twins)
4. **Cognition Layer:** Παραγωγή γνώσης και σύγκριση με παρόμοια συστήματα

5. Configuration Layer: Αυτο-διαμόρφωση και αυτο-βελτιστοποίηση



Σχήμα 3.1: Πενταεπίπεδη αρχιτεκτονική 5C. Πηγή: [19].

3.2 ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ ΜΟΤΙΒΑ ΓΙΑ ΚΑΤΑΝΕΜΗΜΕΝΑ CPS

3.2.1 Layered Architecture (Στρωματοποιημένη Αρχιτεκτονική)

Η στρωματοποιημένη αρχιτεκτονική οργανώνει το σύστημα σε ιεραρχικά επίπεδα με σαφή διαχωρισμό ευθυνών:

Φυσικό Επίπεδο (Physical Layer):

- Αισθητήρες: Θερμοκρασίας, υγρασίας, pH, πίεσης, κίνησης
- Ενεργοποιητές: Ηλεκτροβάνες, κινητήρες, ρελέ
- Φυσικές διεργασίες: Καλλιέργειες, μηχανήματα, δίκτυα ύδρευσης

3.3. STREAMS, EVENT-DRIVEN KAI BATCH VS REAL-TIME PROCESSING

Επίπεδο Διασύνδεσης (Connectivity Layer):

- Ασύρματα πρωτόκολλα: LoRaWAN, Zigbee, Bluetooth LE
- Ενσύρματα: Ethernet, RS-485
- Πύλες IoT: Protocol translation, buffering, edge processing

Επίπεδο Ροής Δεδομένων (Data Flow Layer):

- Message brokers: Apache Kafka, RabbitMQ
- Stream processors: Apache Flink, Kafka Streams
- Event buses: Pub/Sub mechanisms

Επίπεδο Υπηρεσιών (Service Layer):

- APIs: RESTful, GraphQL, gRPC
- Microservices: Ανεξάρτητες, κλιμακούμενες υπηρεσίες
- Business logic: Κανόνες, workflows, αυτοματισμοί

Επίπεδο Παρουσίασης (Presentation Layer):

- Dashboards: Grafana, custom web UIs
- Mobile applications
- Notification systems

3.3 STREAMS, EVENT-DRIVEN KAI BATCH VS REAL-TIME PROCESSING

Η ροή δεδομένων (stream) αποτελεί μια συνεχώς εξελισσόμενη αλληλουχία τιμών που παράγονται από αισθητήρες, ενσωματωμένες συσκευές ή άλλες πηγές και διακινούνται ασύγχρονα εντός του συστήματος. Η συνεχής ροή επιβάλλει μοντέλο επεξεργασίας όπου τα δεδομένα αντιμετωπίζονται ως διαρκές σήμα και όχι ως στατικά σύνολα προς περιοδική ανάλυση. Η **event-driven αρχιτεκτονική** αξιοποιεί αυτό το σήμα ώστε κάθε νέο γεγονός να ενεργοποιεί αμέσως έναν υπολογισμό,

μια σειρά κανόνων (rules), ενημερώσεις ή οποιεσδήποτε άλλες ελεγκτικές ενέργειες. Η μηδενική ανοχή σε καθυστερήσεις είναι θεμελιώδης σε πεδία όπως η γεωργία ακριβείας, το predictive maintenance, ο βιομηχανικός αυτοματισμός και οι αυτόνομες πλατφόρμες, όπου η πληροφορία χάνει αξία όταν παλιώνει ακόμη και κατά μερικά δευτερόλεπτα.

Τα παραδοσιακά *batch* συστήματα λειτουργούν με συγκεντρωτικές παρτίδες δεδομένων, δημιουργώντας αμετάβλητη καθυστέρηση. Το μοντέλο αυτό επαρκεί μόνο όταν οι απαιτήσεις απόκρισης είναι χαμηλές ή όταν το κόστος λάθους δεν είναι κρίσιμο. Σε CPS, όμως, η καθυστέρηση ενός batch κύκλου ισοδυναμεί με απώλεια συστημικής επίγνωσης και αδυναμία δυναμικής προσαρμογής. Η μετάβαση σε **stream processing** (με αρχιτεκτονικές Lambda/Kappa) ελαχιστοποιεί το latency και επιτρέπει συνεχή αξιολόγηση της κατάστασης του συστήματος.

Η λογική ροών συνεπάγεται ανάγκη για stateful operators, μηχανισμούς windowing, backpressure και αποδοτική ανοχή σφαλμάτων, ώστε το σύστημα να παραμένει λειτουργικό υπό πίεση και ασταθείς ροές εισόδου. Η αλλαγή παραδείγματος από batch σε stream δεν αφορά μόνο τη χρονική διάσταση, αλλά και τον τρόπο μοντελοποίησης της πραγματικότητας: το σύστημα αντιμετωπίζει τα δεδομένα ως ζωντανή, κινούμενη πληροφορία και όχι ως στατικό αρχείο προς εκ των υστέρων επεξεργασία. Αυτό οδηγεί σε αρχιτεκτονικές που μπορούν να επιβάλλουν αυτονομία, να προσαρμόζονται στο περιβάλλον και να διατηρούν συστημική συνοχή χωρίς εξάρτηση από περιοδικούς κύκλους επανυπολογισμού.

3.4 APXITEKTONIKEΣ STREAMS: LAMBDA KAI KAPPA

Οι δύο επικρατέστερες αρχιτεκτονικές για επεξεργασία ροών αντιμετωπίζουν το ίδιο πρόβλημα (συνεχή, ασύγχρονη ροή δεδομένων) με διαφορετική φιλοσοφία ως προς την οργάνωση, την ακρίβεια και το λειτουργικό κόστος.

- **Lambda:** Διατηρεί δύο παράλληλα επίπεδα. Το *speed layer* επεξεργάζεται γεγονότα σχεδόν στιγμιαία, προσφέροντας γρήγορα αλλά ενδεχομένως προσεγγιστικά αποτελέσματα. Το *batch layer* αναλαμβάνει ακριβή, πλήρη ανάλυση πάνω σε ολόκληρο το ιστορικό, ώστε τα τελικά δεδομένα να είναι συνεπή ακόμη κι όταν το *fast path* παράγει προσεγγίσεις. Η αρχιτεκτονική αυτή παρέχει ισορροπία μεταξύ απόδοσης και ακρίβειας, αλλά εισάγει αυξημένη πολυπλοκότητα λόγω διπλής υλοποίησης λογικής και ανάγκης συγχρονισμού των *layers*.

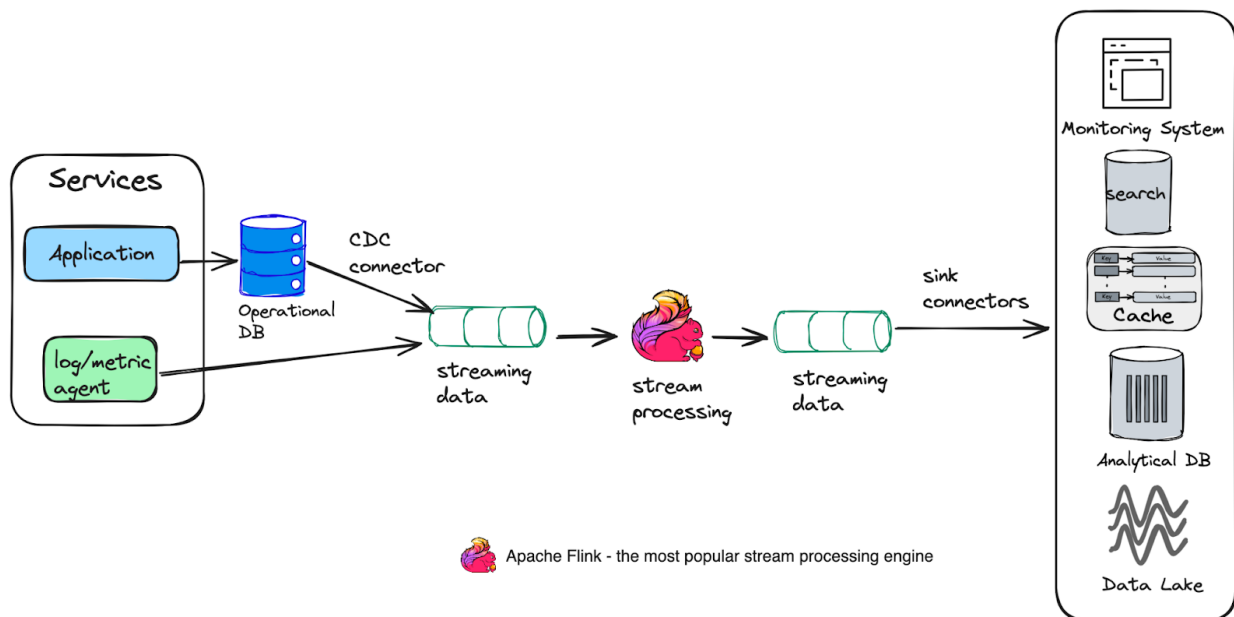
- **Kappa:** Επιλέγει ενιαίο μοντέλο ροών. Όλα τα δεδομένα αντιμετωπίζονται ως *streams*, τόσο για *real-time* όσο και για *reprocessing*. Η επανεπεξεργασία επιτυγχάνεται απλώς με επανάγνωση του *log* των γεγονότων, χωρίς ξεχωριστού *batch* υποσυστήματος. Το αποτέλεσμα είναι απλούστερη λειτουργική δομή, ομοιομορφία στον κώδικα και ελαχιστοποίηση των σημείων ασυνέπειας που προκύπτουν από διπλές υλοποιήσεις. Η *Kappa* απαιτεί ισχυρό και ανθεκτικό *log* (π.χ. *Kafka*) και *stateful stream processors* με ικανότητα διαχείρισης μεγάλης κατάστασης.

Η επιλογή μεταξύ των δύο εξαρτάται από την ανοχή του συστήματος σε καθυστέρηση και από το κόστος που είναι διατεθειμένο να πληρώσει ο σχεδιαστής, σε πολυπλοκότητα. Η **Lambda** ταιριάζει σε περιβάλλοντα όπου οι αναλυτικές εργασίες απαιτούν **πλήρη ακρίβεια**, χωρίς να θυσιάζεται η άμεση απόκριση. Η **Kappa** είναι καταλληλότερη όταν προέχει η **απλότητα**, η ενιαία λογική και η δυνατότητα συνεχούς επεξεργασίας χωρίς παράλληλες ροές υποδομής.

Σε πρακτικό επίπεδο, συστήματα όπως το *Apache Kafka* λειτουργούν ως *durable logs* που επιτρέπουν στον επεξεργαστή ροών να αναπαράγει την κατάσταση οποιαδήποτε στιγμή [20]. Πάνω σε αυτό το υπόβαθρο, πλατφόρμες όπως *Apache Flink* και *Apache Spark* διαχειρίζονται παραθυροποιήσεις (*windows*), *stateful operators*, χρονικές εγγυήσεις (*event-time processing*), ακριβή *semantics* παράδοσης (ακριβώς μια φορά) και μηχανισμούς *backpressure*. Αυτό επιτρέπει στις αρχιτεκτονικές *Lambda* και *Kappa* να λειτουργούν σε κλίμακα, να επιβάλλουν συνεκτική συμπεριφορά σε ασταθή περιβάλλοντα ρυθμού και να διασφαλίζουν ότι οι ροές δεδομένων παραμένουν χρήσιμες ακόμη και όταν οι πηγές εισόδου είναι απρόβλεπτες.

Στο Σχήμα 3.2 αποτυπώνεται μια τυπική υλοποίηση αρχιτεκτονικής ροών, όπου τα επιχειρησιακά δεδομένα παράγονται από εφαρμογές και συστήματα καταγραφής, εισάγονται σε ένα ανθεκτικό *log* μέσω μηχανισμών όπως *CDC connectors* και *agents* συλλογής, και στη συνέχεια τροφοδοτούν έναν *stream processor*. Ο επεξεργαστής (π.χ. *Apache Flink*) διαχειρίζεται κατάσταση, χρονικά παράθυρα και εγγυήσεις παράδοσης, παράγοντας νέα ρεύματα τα οποία καταλήγουν σε διαφορετικά υποσυστήματα αποθήκευσης ή αναζήτησης (βάσεις αναλυτικών δεδομένων, *caches*, *data lakes*, συστήματα *monitoring*). Η ροή αυτή αντανακλά τη λογική της αρχιτεκτονικής *Kappa*, όπου το *log* αποτελεί την πηγή αλήθειας και η επεξεργασία γίνεται ενιαία, χωρίς διαχωρισμό σε *batch* και *speed* επίπεδα.

3.5. ΔΙΑΣΥΝΔΕΣΗ ΥΠΟΣΥΣΤΗΜΑΤΩΝ



Σχήμα 3.2: Τυπική αρχιτεκτονική επεξεργασίας ροών. Τα δεδομένα παράγονται από επιχειρησιακές εφαρμογές και βάσεις, εισάγονται σε ανθεκτικό logs μέσω CDC (Change Data Capture) και agents, επεξεργάζονται από stateful stream processors (π.χ. Apache Flink) και διοχετεύονται σε πολλαπλούς αποδέκτες όπως συστήματα αναζήτησης, cache, αναλυτικές βάσεις και data lakes. Προσαρμογή από [21].

Παρά την έμφαση της παραπάνω αρχιτεκτονικής στο μοντέλο Kappa, οι δύο προσεγγίσεις (Lambda και Kappa) δεν απαιτούν κατ' ανάγκη διαφορετική φιλοσοφία ως προς τα εργαλεία, αλλά διαφορετική στάση απέναντι στη μοντελοποίηση της πληροφορίας: υβριδική διπλή ανάλυση στη μία περίπτωση, μονολιθική αντίληψη ροής στην άλλη.

3.5 ΔΙΑΣΥΝΔΕΣΗ ΥΠΟΣΥΣΤΗΜΑΤΩΝ

Για την αλληλεπίδραση πολλαπλών, ετερογενών υποσυστημάτων, το **publish/subscribe** μοτίβο προσφέρει έναν θεμελιώδη μηχανισμό ασύγχρονης επικοινωνίας, επιτρέποντας χαλαρά συνδεδεμένες (*loosely coupled*) υπηρεσίες. Κάθε παραγωγός (*publisher*) μπορεί να στέλνει μηνύματα χωρίς να γνωρίζει τους καταναλωτές (*subscribers*) και αντίστροφα, διατηρώντας την αυτονομία των υποσυστημάτων, διευκολύνοντας την επεκτασιμότητα και επιτρέποντας αλλαγές στην τοπολογία χωρίς κεντρικό συντονισμό.

3.6. ΑΠΟΘΗΚΕΥΣΗ, CONSISTENCY MODELS ΚΑΙ CACHING

Η ποιότητα υπηρεσίας (*Quality of Service, QoS*) αποκτά κρίσιμη σημασία σε περιβάλλοντα με αστάθεια σύνδεσης ή περιορισμένους πόρους. Η επιλογή μεταξύ *fire-and-forget*, τουλάχιστον-μία-φορά (*at-least-once*) ή ακριβώς-μία-φορά (*exactly-once*) παράδοσης μηνυμάτων καθορίζει την αξιοπιστία και την ανθεκτικότητα των εφαρμογών. Σε *low-power* ή *low-bandwidth* δίκτυα, πρωτόκολλα όπως το *MQTT* και το *CoAP* μειώνουν το *overhead*, υποστηρίζοντας ελαφριά, αξιόπιστη και ασφαλή ανταλλαγή μηνυμάτων.

Πέρα από την αξιοπιστία, η διαχείριση *backpressure*, η αποθήκευση προσωρινών μηνυμάτων (*message buffering*) και η δυνατότητα *replay* είναι κρίσιμα στοιχεία για τη διατήρηση συνέπειας σε συστήματα με υψηλή μεταβλητότητα φόρτου. Η χρήση ιεραρχιών των *topics* και φιλτραρίσματος επιτρέπει στους καταναλωτές (*subscribers*) να λαμβάνουν μόνο τα δεδομένα που τους ενδιαφέρουν, μειώνοντας τον όγκο μεταφερόμενων μηνυμάτων και το *latency*.

Επιπλέον, η συνδυαστική χρήση *pub/sub* με *event-driven* αρχιτεκτονικές ενισχύει τη δυνατότητα *real-time* αντίδρασης σε γεγονότα. Η ενσωμάτωση μηχανισμών παρακολούθησης και *metrics* (π.χ. *latency* ανά *topic*, ακυρωμένα μηνύματα) επιτρέπει την ενεργή βελτιστοποίηση της επικοινωνίας και την προσαρμογή σε μεταβαλλόμενες συνθήκες δικτύου ή φόρτου.

Τελικά, η διασύνδεση υποσυστημάτων σε CPS απαιτεί προσεκτική ισορροπία μεταξύ αυτονομίας, αξιοπιστίας, *latency* και κατανάλωσης πόρων, με δυνατότητα κλιμάκωσης και επαναδιαμόρφωσης χωρίς διακοπή λειτουργίας.

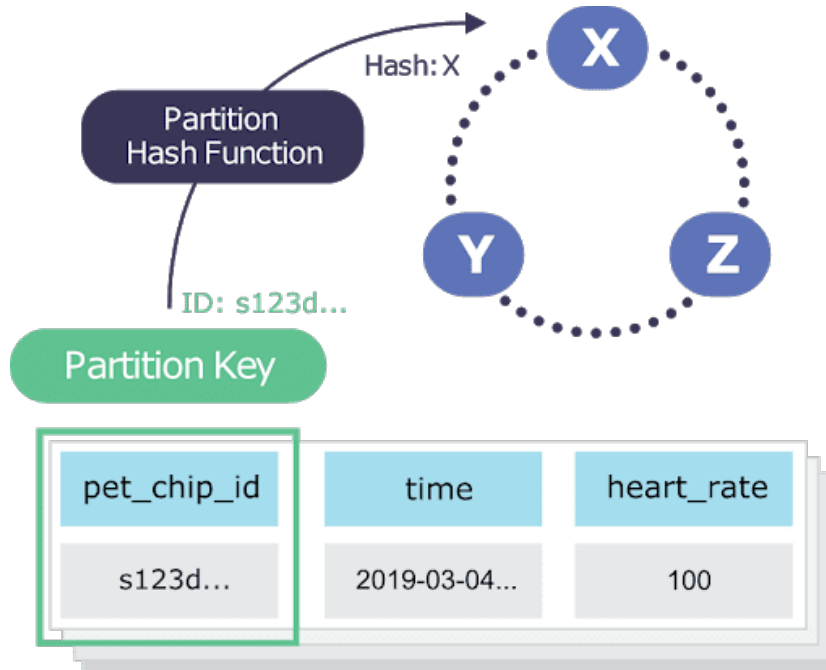
3.6 ΑΠΟΘΗΚΕΥΣΗ, CONSISTENCY MODELS ΚΑΙ CACHING

Η αποθήκευση δεδομένων σε CPS αποτελεί κρίσιμη πρόκληση λόγω του όγκου, της ταχύτητας παραγωγής και της ανάγκης για *real-time* προσπέλαση. Κατανεμημένα συστήματα βάσεων δεδομένων, όπως *NoSQL* ή *wide-column stores*, χρησιμοποιούν *replication*, *partitioning* και *sharding* για να διασφαλίσουν υψηλή διαθεσιμότητα, *fault tolerance* και οριζόντια κλιμάκωση.

Όπως φαίνεται στο Σχήμα 3.3, η διαδικασία του *partitioning* βασίζεται σε μια συνάρτηση κατακερματισμού (*hash function*) η οποία αντιστοιχίζει το *partition key* σε συγκεκριμένους κόμβους (*nodes*) ενός δακτυλίου (*ring*). Η επιλογή *wide-column store* γίνεται συχνά για την υποστήριξη μεγάλων ροών δεδομένων με ευέλικτο σχήμα και αποτελεσματική οργάνωση ανά χρονικά διαστήματα (*date bucketing*), επιτρέπο-

3.6. ΑΠΟΘΗΚΕΥΣΗ, CONSISTENCY MODELS ΚΑΙ CACHING

ντας ταχεία προσπέλαση σε ιστορικά δεδομένα χωρίς να επηρεάζεται η απόδοση των ερωτημάτων πραγματικού χρόνου.



Σχήμα 3.3: Μηχανισμός κατανομής δεδομένων μέσω Partition Key και Hash Function σε κατανεμημένο δακτύλιο [22].

Η τεχνική του **date bucketing** οργανώνει τα δεδομένα σε χρονικούς “κουβάδες” ή *partitions* (π.χ. ανά ημέρα, ώρα ή λεπτό), μειώνοντας το εύρος σάρωσης (scan range) κατά την ανάκτηση ή ανάλυση μεγάλων όγκων δεδομένων. Σε συνδυασμό με τα *wide-column stores*, κάθε κουβάς μπορεί να αποθηκεύει σειρές και στήλες που αντιστοιχούν σε γεγονότα ή μετρήσεις συγκεκριμένης χρονικής περιόδου, διευκολύνοντας τόσο batch αναλύσεις όσο και *real-time streaming queries* χωρίς να απαιτείται πλήρης αναδιάταξη ή ανακατασκευή των δεδομένων.

Η επιλογή **consistency model** παραμένει κρίσιμη και διέπεται από τους περιορισμούς του Θεωρήματος CAP (Consistency, Availability, Partition Tolerance). Σύμφωνα με αυτό, σε ένα κατανεμημένο σύστημα με παρουσία δικτυακών κατατμήσεων (*partitions*), η επιλογή πρέπει να γίνει ανάμεσα στη συνέπεια (Consistency) και τη διαθεσιμότητα (Availability). Το *strong consistency* εξασφαλίζει ότι όλοι οι κόμβοι βλέπουν την ίδια κατάσταση (CP συστήματα), ενώ το *eventual consistency* μειώνει latency και αυξάνει διαθεσιμότητα (AP συστήματα), αποδεχόμενο προσωρινή ασυνέπεια. *Causal* και *session-based* μοντέλα προσφέ-

3.7. ΔΙΑΧΕΙΡΙΣΗ ΥΠΟΔΟΜΗΣ ΚΑΙ CLOUD-NATIVE PATTERNS

ρουν ενδιάμεσες λύσεις, διατηρώντας λογική αλληλουχία γεγονότων χωρίς πλήρη συγχρονισμό.

Το *in-memory caching* (Redis, Memcached) μειώνει δραστικά το latency για συχνά προσπελάσιμα δεδομένα, ενώ στρατηγικές όπως *write-through* ή *write-back* ελέγχουν την αντοχή σε αποτυχίες. Η συνδυαστική χρήση cache με *date bucketing* επιτρέπει την άμεση πρόσβαση σε “hot” χρονικά διαστήματα χωρίς να επηρεάζεται η αποδοτικότητα ανάκτησης ιστορικών δεδομένων.

Σε CPS πραγματικού χρόνου, η οργάνωση δεδομένων με *date bucketing* διευκολύνει επίσης τα *stream processing pipelines*, επιτρέποντας εύκολη επανεπεξεργασία γεγονότων σε συγκεκριμένα χρονικά διαστήματα χωρίς να επηρεάζεται η συνολική δομή της βάσης. Παρακολούθηση μετρικών όπως *bucket size*, *cache hit/miss ratio* και *latency* επιτρέπει τη βελτιστοποίηση της υποδομής και τη δυναμική προσαρμογή σε μεταβαλλόμενο φόρτο.

Η στρατηγική αυτή διασφαλίζει ισορροπία μεταξύ *latency*, *consistency*, *durability* και κλιμάκωσης, ενώ εκμεταλλεύεται πλήρως τα χαρακτηριστικά των *wide-column stores* για οργάνωση μεγάλου όγκου, χρονικά ευαίσθητων δεδομένων.

3.7 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), αυτοϊαση (self-healing):** Προσθήκη/αφαίρεση pods, επανεκκινήσεις σε περίπτωση αποτυχιών, health checks.
- **Παρατηρησιμότητα (observability):** Συλλογή μετρικών, logs, tracing και alerting, με ενδεικτικά εργαλεία Prometheus, Grafana [23], ELK/EFK stack [24].

3.7. ΔΙΑΧΕΙΡΙΣΗ ΥΠΟΔΟΜΗΣ ΚΑΙ CLOUD-NATIVE PATTERNS

- **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 [25].
- **Service discovery, load balancing και secrets management:** Απαιτήτητα 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.

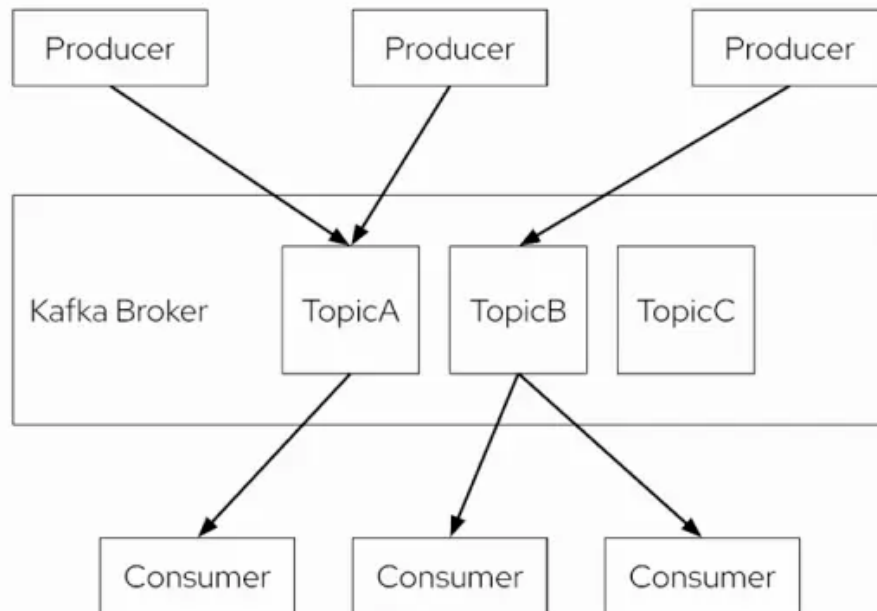
4

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 [26]. Η αρχιτεκτονική των CPS, που αναλύθηκε στο κεφάλαιο 3, 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) [27].



Σχήμα 4.1: Αρχιτεκτονική Apache Kafka: Ροή μηνυμάτων από τους producers στους consumers μέσω των topics του broker [28].

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 [29] 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 κατανεμημένο σύστημα επεξεργασίας δεδομένων μεγάλης κλίμακας, 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. [30] προσθέτουν, πως το Kafka σχεδιάστηκε εξ αρχής 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 ενδιαμέσου επιπέδου επεξεργασίας.

4.2 STREAM PROCESSING

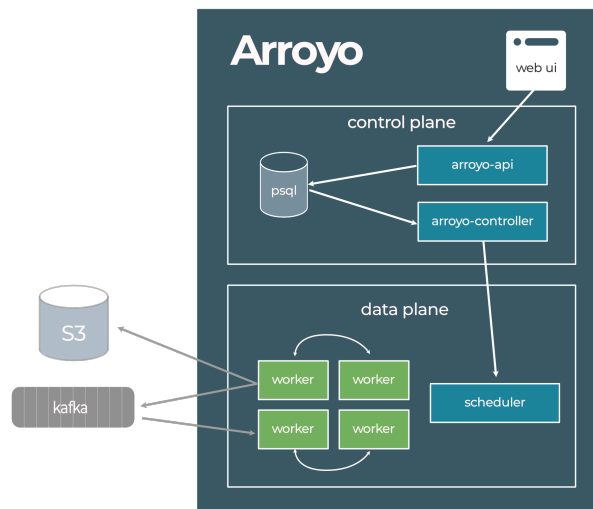
Η επεξεργασία ροών δεδομένων σε πραγματικό χρόνο απαιτεί συστήματα που μπορούν να διαχειριστούν συνεχείς ροές δεδομένων, να υποστηρίζουν stateful processing και να διασφαλίζουν αξιοπιστία ακόμη και σε περιπτώσεις προσωρινών αποτυχιών. Στην ενότητα αυτή παρουσιάζονται τα συστήματα *Flink* και *Arroyo*, με έμφαση στη λειτουργικότητα, την αρχιτεκτονική και την καταλληλότητά τους για CPS πραγματικού χρόνου και IoT pipelines.

4.2.1 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.

4.2.2 Arroyo

Παρά το γεγονός ότι το *Apache Flink* αποτελεί μια ώριμη και πλήρως εξοπλισμένη λύση για την επεξεργασία ροών δεδομένων, η υψηλή πολυπλοκότητα και οι λειτουργικές απαιτήσεις του περιβάλλοντος JVM κρίθηκαν υπερβολικές για τις ανάγκες της παρούσας αρχιτεκτονικής. Για τον λόγο αυτόν επιλέχθηκε το *Arroyo*, ένα open-source σύστημα επεξεργασίας ροών (stream processing) γραμμένο σε *Rust*.



Σχήμα 4.2: Εσωτερική αρχιτεκτονική του Arroyo: Διαχωρισμός control plane και data plane με χρήση workers [31].

Το Arroyo, αν και δεν υποστηρίζει ακόμα όλες τις δυνατότητες του Flink, καλύπτει επαρκώς τις απαιτήσεις της αρχιτεκτονικής της προτεινόμενης πλατφόρμας για *real-time* επεξεργασία και *event-driven pipelines*. Λόγω της τρέχουσας μορφής του, δεν διαθέτει native connectors για βάσεις δεδομένων όπως η Cassandra, γι' αυτό υλοποιήθηκε μια **custom διασύνδεση** που επιτρέπει την απρόσκοπτη ενσωμάτωση στο υπόλοιπο οικοσύστημα της υποδομής, χωρίς να επηρεάζεται η αξιοπιστία ή η απόδοση των pipelines.

Η χρήση της Rust εξασφαλίζει μεγαλύτερη ασφάλεια μνήμης, σταθερότερο concurrency model και χαμηλότερο latency, στοιχεία κρίσιμα για CPS πραγματικού χρόνου. Επιπλέον, η αρχιτεκτονική του επιτρέπει εύκολη κλιμάκωση, stateful processing και ενσωμάτωση με custom stream sources και sinks, γεγονός που το καθιστά κατάλληλο για πειραματικά και ερευνητικά περιβάλλοντα CPS, ενώ παραμένει μια υπό παρακολούθηση λύση για πιθανή μελλοντική χρήση σε παραγωγικά συστήματα, όταν οι δυνατότητές του επεκταθούν.

Αυτό το μοντέλο προσφέρει έναν ευέλικτο, *lightweight* και υψηλής απόδοσης μηχανισμό επεξεργασίας ροών, με χαμηλή πολυπλοκότητα ανάπτυξης και συντήρησης, χωρίς τις καθυστερήσεις και την πολυπλοκότητα που φέρνει το *JVM-based Flink*.

4.3 DISTRIBUTED STORAGE

Η αποθήκευση μεγάλου όγκου δεδομένων σε κατανεμημένα συστήματα απαιτεί συστήματα που μπορούν να διαχειριστούν *write-heavy* workloads με υψηλή διαθεσιμότητα και επεκτασιμότητα. Τα *wide-column databases* αποτελούν την κύρια επιλογή σε τέτοια σενάρια, προσφέροντας replication, partitioning και δυνατότητες *eventual consistency*, απαραίτητες για εφαρμογές πραγματικού χρόνου και streaming δεδομένων IoT. Στην ενότητα αυτή αναλύονται τα συστήματα Cassandra και ScyllaDB, παρουσιάζοντας τα χαρακτηριστικά τους, τα πλεονεκτήματα και την καταλληλότητά τους για workloads με υψηλή συχνότητα εγγραφών και απαιτήσεις χαμηλού latency.

4.3.1 Cassandra

Η Cassandra ακολουθεί το μοντέλο *eventual consistency*, υποστηρίζει κατανεμημένη αποθήκευση με replication και partitioning, και έχει σχεδιαστεί για εφαρμογές συχνής καταχώρησης και αποθήκευσης (*write-heavy*) [32]. Σε περιβάλλοντα IoT, όπου οι συσκευές παράγουν συνε-

χώς 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 [33], 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.

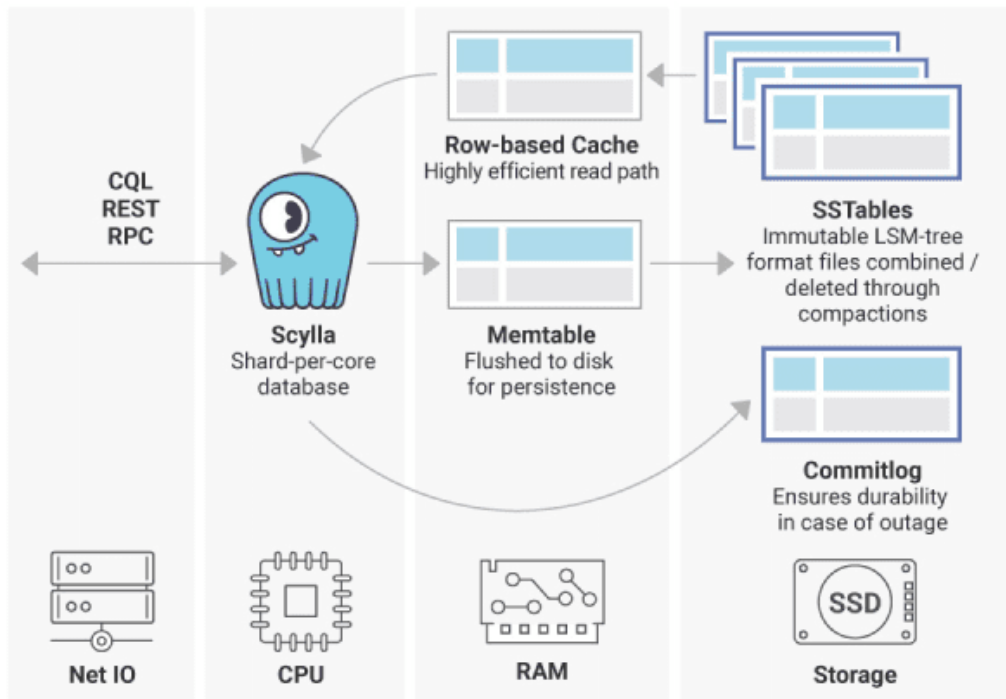
Στη συγκεκριμένη περίπτωση, η υψηλή απόδοση (high performance) της Cassandra σε συνδυασμό με την εύκολη και ελαστική επεκτασιμότητα (elastic scalability), καθιστά το σύστημα ιδανικό για εφαρμογές πραγματικού χρόνου με έντονη ροή δεδομένων. Η δυνατότητα προσθήκης ή αφαίρεσης κόμβων (nodes) χωρίς διακοπή 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.

4.3.2 ScyllaDB

Παρά την ωριμότητα και τη διαδεδομένη χρήση της Cassandra, για την κάλυψη των αναγκών του συγκεκριμένου συστήματος επιλέχθηκε να χρησιμοποιηθεί η ScyllaDB, ένα σύστημα wide-column σχεδιασμένο ως drop-in αντικαταστάτης της Cassandra, αλλά με σημαντικά βελτιωμένα χαρακτηριστικά απόδοσης και χαμηλότερο latency. Η ScyllaDB είναι γραμμένη σε C++ και χρησιμοποιεί ένα πλήρως ασύγχρονο, *shard-per-core* αρχιτεκτονικό μοντέλο, εκμεταλλευόμενη πλήρως τις δυνατότητες των πολυπύρηνων επεξεργαστών, μειώνοντας την καθυστέρηση του JVM που χαρακτηρίζει την Cassandra.

Η ScyllaDB υποστηρίζει *replication*, *partitioning* και *eventual consistency* με δυνατότητα *fine-tuned consistency levels*, επιτρέποντας τον ακριβή έλεγχο μεταξύ latency και αξιοπιστίας, ανάλογα με τις

ανάγκες κάθε *pipeline*. Η υποστήριξη *date bucketing* και *time-series optimized tables* καθιστά τη *ScyllaDB* ιδανική για ροές τηλεμετρίας IoT, όπου τα δεδομένα γράφονται συνεχώς και προσπελάζονται τόσο για dashboards πραγματικού χρόνου όσο και για ιστορικές αναλύσεις.



Σχήμα 4.3: Αρχιτεκτονική ScyllaDB: Η Shard-per-core επεξεργασία και η διαδρομή δεδομένων από τη μνήμη (Memtable) στον δίσκο (SSTables) [34].

Σε σύγκριση με την *Cassandra*, η *ScyllaDB* παρέχει:

- **Χαμηλότερο latency**: χάρη στο ασύγχρονο, *shard-per-core* μοντέλο που μειώνει το *lock contention* αλλά και τις καθυστερήσεις λόγω *garbage collection*.
- **Πλήρη αξιοποίηση multi-core**: κάθε πυρήνας του επεξεργαστή διαχειρίζεται ένα *shard*, επιτρέποντας γραμμική κλιμάκωση με την προσθήκη κόμβων.
- **Προβλέψιμη απόδοση**: μειωμένη διακύμανση στην καθυστέρηση ακόμα και σε *write-heavy* διεργασίες.

- **Time-series friendly:** εύκολη εφαρμογή *date bucketing* και *retention policies* για μακροχρόνια αποθήκευση μεγάλου όγκου δεδομένων.

Η επιλογή της *ScyllaDB* επιτρέπει άμεση καταχώρηση μεγάλου όγκου τηλεμετρίας από αισθητήρες με συνεχείς ροές δεδομένων, εξασφαλίζοντας παράλληλα την ανάγνωση σε πραγματικό χρόνο για ανάλυση ή *triggers* επεξεργασίας σε *stream processing pipelines* όπως το *Arroyo*. Η δυνατότητα εύκολης κλιμάκωσης και *self-healing* κόμβων επιτρέπει στο σύστημα να προσαρμόζεται δυναμικά σε αυξομειώσεις φορτίου, ενώ η υψηλή αξιοπιστία και η χαμηλή καθυστέρηση το καθιστούν προτιμητέα επιλογή για παραγωγικά *CPS* περιβάλλοντα σε σύγκριση με την *Cassandra*.

Με αυτήν την επιλογή, η πλατφόρμα εκμεταλλεύεται τις βελτιώσεις της *ScyllaDB* σε *throughput* και προβλεψιμότητα, χωρίς να θυσιάζει την ευελιξία και τη διαλειτουργικότητα που απαιτούνται για εφαρμογές *IoT* πραγματικού χρόνου και *CPS*.

4.4 CACHING

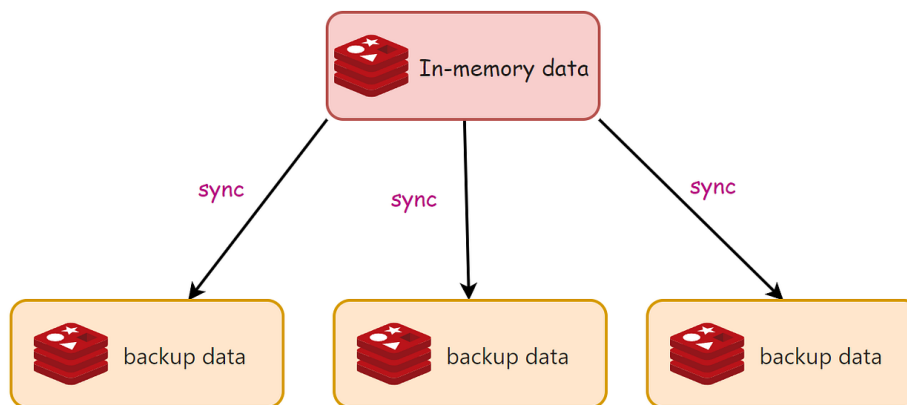
Η προσωρινή αποθήκευση (*caching*) αποτελεί κρίσιμο στοιχείο σε *CPS* και κατανεμημένες υποδομές, όπου η συνεχής εισροή δεδομένων απαιτεί γρήγορη και αξιόπιστη προσπέλαση για εφαρμογές πραγματικού χρόνου. Ο κύριος στόχος των μηχανισμών *caching* είναι η μείωση του *latency* και η αποσυμφόρηση των βάσεων δεδομένων, επιτρέποντας την άμεση παροχή δεδομένων στους χρήστες ή στις υπηρεσίες ανάλυσης. Μέσω *caching*, οι εφαρμογές διεπαφής μπορούν να παρέχουν οπτικοποιημένα και άμεσα αξιοποιήσιμη πληροφόρηση χωρίς να επηρεάζεται η απόδοση του *backend* ή η συνολική επεκτασιμότητα του συστήματος.

Στην ενότητα αυτή αναλύονται οι κύριες τεχνολογίες *in-memory caching* που εξετάστηκαν, με έμφαση στα χαρακτηριστικά τους, τα πλεονεκτήματα, τις αρχιτεκτονικές επιλογές και την καταλληλότητά τους για εργασίες πραγματικού χρόνου. Οι υποενότητες καλύπτουν τις λύσεις *Redis*, *Valkey*, *DragonflyDB* και *Memcached*, προσφέροντας συγκριτική θεώρηση και επιχειρήματα για την τελική επιλογή.

4.4.1 Redis

Το *Redis* αποτελεί ένα από τα πιο δημοφιλή συστήματα *in-memory key-value stores*, προσφέροντας γρήγορη προσπέλαση σε δεδομένα με

χαμηλή καθυστέρηση και υποστήριξη σύνθετων data structures όπως *lists*, *sets*, *sorted sets* και *hashes*. Η αρχιτεκτονική του βασίζεται σε μοντέλο *event loop*, το οποίο εξασφαλίζει αποτελεσματική διαχείριση ταυτόχρονων αιτημάτων χωρίς τις καθυστερήσεις του *multithreading*. Επιπλέον, το *Redis* προσφέρει δυνατότητες *persistence*, *replication* και *pub/sub messaging*, γεγονός που το καθιστά κατάλληλο για εφαρμογές πραγματικού χρόνου όπου απαιτείται υψηλή ταχύτητα ανάγνωσης και εγγραφής.



Σχήμα 4.4: Αρχιτεκτονική Master-Slave Replication: Απεικόνιση της ροής δεδομένων από τον Master κόμβο προς τους Slaves. Ο μηχανισμός αυτός επιτρέπει την ανάγνωση από πολλαπλούς κόμβους (read scalability) και την παροχή αντιγράφων ασφαλείας σε πραγματικό χρόνο [35].

Η αρχιτεκτονική που απεικονίζεται στο Σχήμα 4.4 αποτελεί τη βάση για τη διασφάλιση της ανθεκτικότητας (*durability*) και της διαθεσιμότητας (*availability*). Η χρήση *Master-Slave replication* επιτρέπει την αυτόματη ανάκαμψη (*failover*) σε περίπτωση αστοχίας, καθώς ένας εκ των Slaves μπορεί να προαχθεί σε Master, ενώ η παράλληλη χρήση μηχανισμών *persistence to disk* εγγυάται την ακεραιότητα των δεδομένων μετά από επανεκκίνηση.

4.4.2 Valkey

Με την προσωρινή μετατροπή του *Redis* σε κλειστό λογισμικό, εμφανίστηκε η ανάγκη για μια ανοικτή, και υποστηριζόμενη από την κοινότητα, εναλλακτική. Το *Valkey* προέκυψε ως φυσική συνέχεια του *Redis* υπό την αιγίδα του *Linux Foundation*, διατηρώντας συμβατότητα με

το API του *Redis* αλλά ως πλήρως ανοιχτό λογισμικό. Μιας και η υποστήριξη μακροπρόθεσμης συντήρησης και η διαλειτουργικότητα είναι κρίσιμες ζητούμενα, το *Valkey* παρέχει ένα ασφαλές και αξιόπιστο πυλώνα για *caching* και πρόσβαση δεδομένων σε πραγματικό χρόνο.

4.4.3 DragonflyDB

Το *DragonflyDB* αποτελεί μια σύγχρονη, *Redis-compatible* υλοποίηση, σχεδιασμένη ως *drop-in* αντικαταστάτης, προσφέροντας ταχύτερη εκκίνηση και βελτιστοποιημένη διαχείριση μνήμης. Αν και χρήσιμο για γρήγορα tests ή περιβάλλοντα ανάπτυξης, η εμπειρία σε παραγωγικές διαδικασίες έδειξε ότι, για συνεχή, υψηλό throughput και εγγυημένη διαθεσιμότητα, η επιλογή *Valkey* παραμένει προτιμητέα. Η δυνατότητα *seamless* μετακίνησης μεταξύ *Redis*, *DragonflyDB* και *Valkey* προσφέρει ευελιξία, χωρίς να δεσμεύει τη μακροπρόθεσμη στρατηγική του συστήματος.

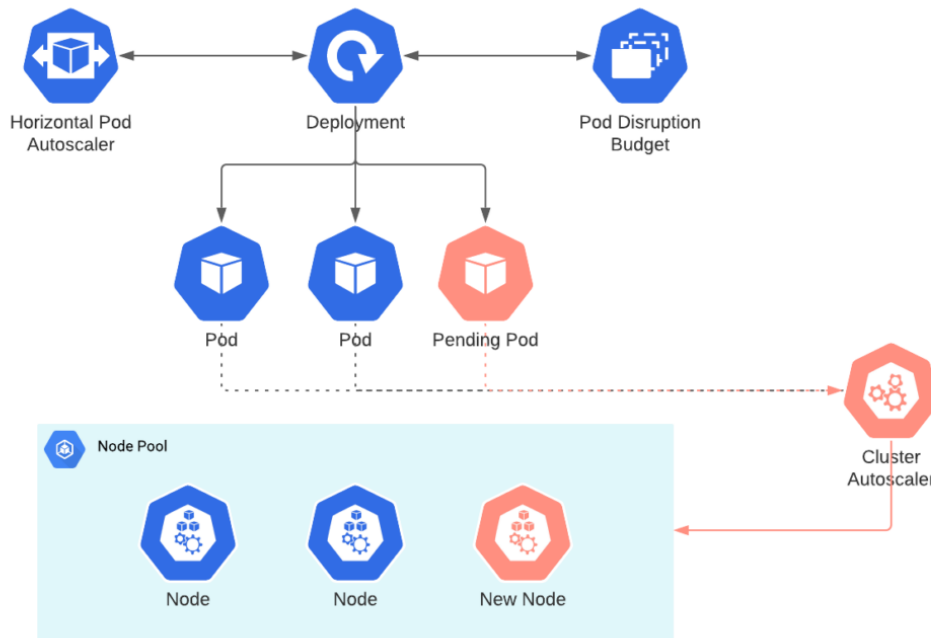
4.4.4 Memcached

Παρά την κυριαρχία των λύσεων βασισμένων στο *Redis*, το *Memcached* παραμένει μια ελαφριά και δοκιμασμένη τεχνολογία για *caching* απλών ζεύγων κλειδιού-τιμής, με πολυνηματική αρχιτεκτονική που επιτρέπει την αποτελεσματική αξιοποίηση πολλαπλών πυρήνων CPU. Η απλότητα και η σταθερότητα του *Memcached* το καθιστούν ιδανικό για σενάρια όπου απαιτείται αποκλειστικά γρήγορη προσωρινή αποθήκευση χωρίς σύνθετες δομές δεδομένων ή *persistence*, διατηρώντας χαμηλό footprint και προβλέψιμη σταθερότητα στην απόδοση σε *write-heavy* workloads.

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 αποτυχημένων 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 συστή-

ματος σε υψηλά επίπεδα, ακόμη και σε συνθήκες έντονου φορτίου ή υλικών αποτυχιών.



Σχήμα 4.5: Μηχανισμοί Autoscaling στο Kubernetes: Απεικόνιση της αλληλεπίδρασης μεταξύ του Horizontal Pod Autoscaler (HPA), του Cluster Autoscaler και των pods για τη διαχείριση μεταβαλλόμενων υπολογιστικών αναγκών. Προσαρμογή από [36].

Όπως φαίνεται στο Σχήμα 4.5, η χρήση του *Horizontal Pod Autoscaler* επιτρέπει την αυξομείωση των *pods* βάσει μετρικών (όπως CPU ή RAM), ενώ ο *Cluster Autoscaler* διασφαλίζει ότι υπάρχουν αρκετοί φυσικοί πόροι (Nodes) για να φιλοξενήσουν τα νέα *pods*. Αυτοί οι μηχανισμοί, σε συνδυασμό με τα *liveness/readiness probes*, προσδίδουν λειτουργική ανθεκτικότητα και διατηρούν τη διαθεσιμότητα του συστήματος σε υψηλά επίπεδα.

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, [37] - 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 λύση που προτείνει η παρούσα διπλωματική εργασία 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 αυτοϊασης 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

ζει 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 ελέγχων 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.

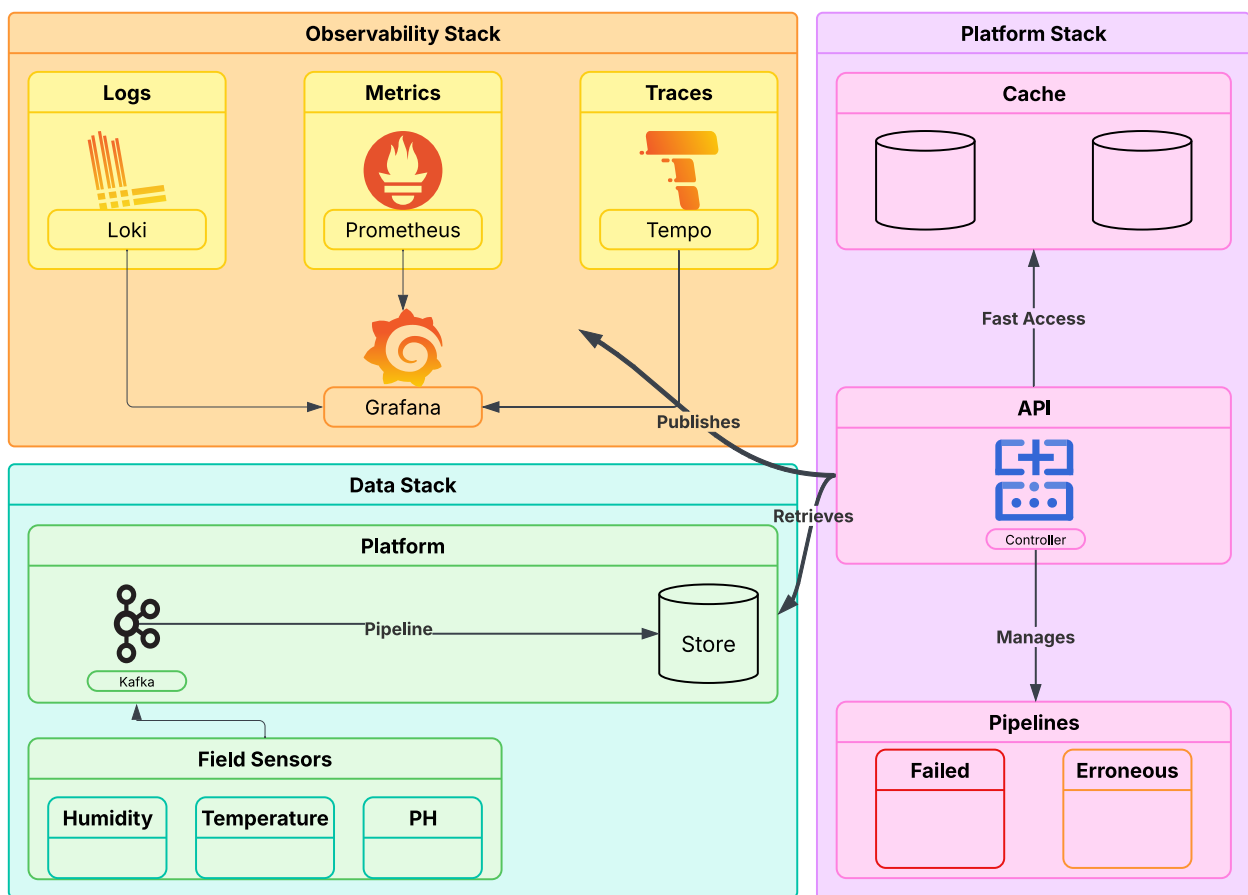
Η συνολική αρχιτεκτονική της πλατφόρμας, όπως αυτή υλοποιείται στο περιβάλλον *Kubernetes*, αποτυπώνεται διαγραμματικά στο Σχήμα 5.1. Η δομή οργανώνεται σε τρία διακριτά αλλά αλληλεξαρτώμενα λειτουργικά επίπεδα (*stacks*), τα οποία συνεργάζονται για την απρόσκοπτη ροή δεδομένων από το πεδίο προς την τελική οπτικοποίηση και διαχείριση.

- **Data Stack:** Αποτελεί τη βάση της αρχιτεκτονικής, όπου οι αισθητήρες πεδίου (*field sensors*) τροφοδοτούν με μετρήσεις μια υπο-

5.1. ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΔΟΜΗΣ

δομή βασισμένη στο *Apache Kafka*, εξασφαλίζοντας την αξιόπιστη μεταφορά και αποθήκευση της πληροφορίας.

- **Platform Stack:** Περιλαμβάνει το κεντρικό *API* και τους ελεγκτές (controllers) που διαχειρίζονται τις ροές (pipelines) και την προσωρινή αποθήκευση (cache), λειτουργώντας ως ο "εγκέφαλος" του συστήματος.
- **Observability Stack:** Διασφαλίζει τη διαφάνεια της υποδομής μέσω της συλλογής καταγραφών (Loki), μετρήσεων (Prometheus) και ιχνηλάτησης (Tempo), με τελικό προορισμό το περιβάλλον του *Grafana*.



Σχήμα 5.1: Αρχιτεκτονική υποδομής δεδομένων

5.1.1 Τοπολογία και ρόλοι κόμβων

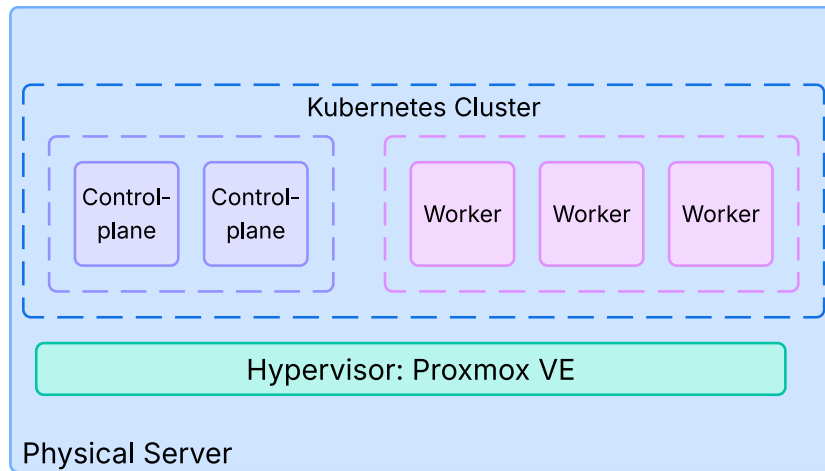
Η υποδομή αποτελείται από έναν *Kubernetes cluster* με δύο σαφείς κατηγορίες κόμβων:

- **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 5.2: Kubernetes cluster με δύο *controlplane* κόμβους και τρεις κόμβους

5.1.2 Αυτοματισμοί υποδομής και διακυβέρνηση

Η διαχείριση της πλατφόρμας βασίζεται στην αρχή του **GitOps**, χρησιμοποιώντας το **ArgoCD** ως τον κεντρικό ελεγκτή συγχρονισμού. Όλα τα στοιχεία της υποδομής - από τα δίκτυα έως τις παραμετροποιήσεις των βάσεων δεδομένων - ορίζονται δηλωτικά σε **Helm charts** και **manifests**, καθιστώντας το **Git repository** τη μοναδική «πηγή αλήθειας» (Source of Truth).

Η λειτουργία του αυτοματισμού βασίζεται στον συνεχή βρόχο **συμφιλίωσης (Reconciliation Loop)**. Ο ελεγκτής του ArgoCD παρακολουθεί ενεργά δύο καταστάσεις:

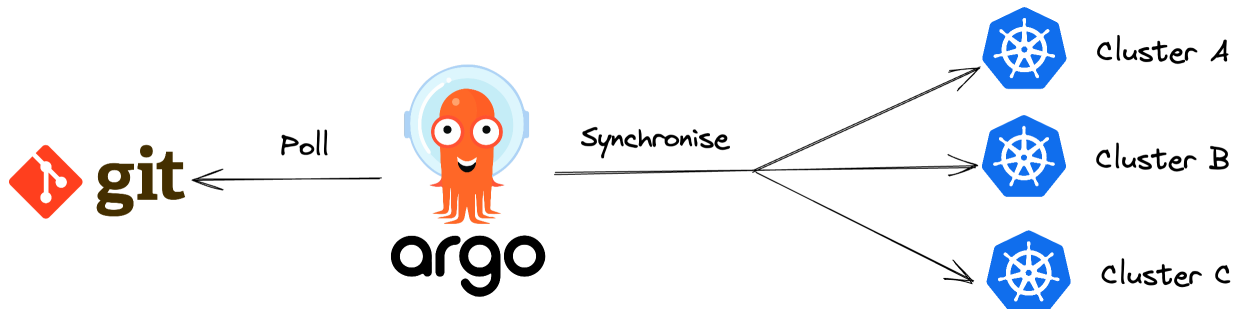
- **Desired State:** Την επιθυμητή κατάσταση όπως ορίζεται στον κώδικα στο *Git*.
- **Live State:** Την πραγματική κατάσταση των πόρων μέσα στο *Kubernetes cluster*.

Σε περίπτωση που εντοπιστεί απόκλιση (Configuration Drift) - είτε λόγω αλλαγής στον κώδικα είτε λόγω χειροκίνητης παρέμβασης στο *cluster* - ο μηχανισμός προβαίνει σε αυτόματο συγχρονισμό (Self-healing), επαναφέροντας το *cluster* στην εγκεκριμένη κατάσταση. Αυτή η προσέγγιση ελαχιστοποιεί το ανθρώπινο σφάλμα και διασφαλίζει ότι η υποδομή είναι πλήρως αναπαραγώγιμη (Reproducible).

Παράλληλα, η διακυβέρνηση ενισχύεται μέσω της χρήσης **Namespaces** και αυστηρών κανόνων **RBAC**. Η απομόνωση αυτή, σε συνδυασμό με

5.2. ΡΟΗ ΔΕΔΟΜΕΝΩΝ ΣΕ ΠΡΑΓΜΑΤΙΚΟ ΧΡΟΝΟ

την αυτοματοποιημένη προώθηση αλλαγών, επιτρέπει τον πλήρη έλεγχο (Audit Log) του ποιος άλλαξε τι και πότε, αφού κάθε τροποποίηση στην υποδομή πρέπει να περάσει από τη διαδικασία του *Pull Request* και της έγκρισης κώδικα.



Σχήμα 5.3: Αρχιτεκτονική λειτουργία του ArgoCD και ροή GitOps (Προσαρμογή από [38])

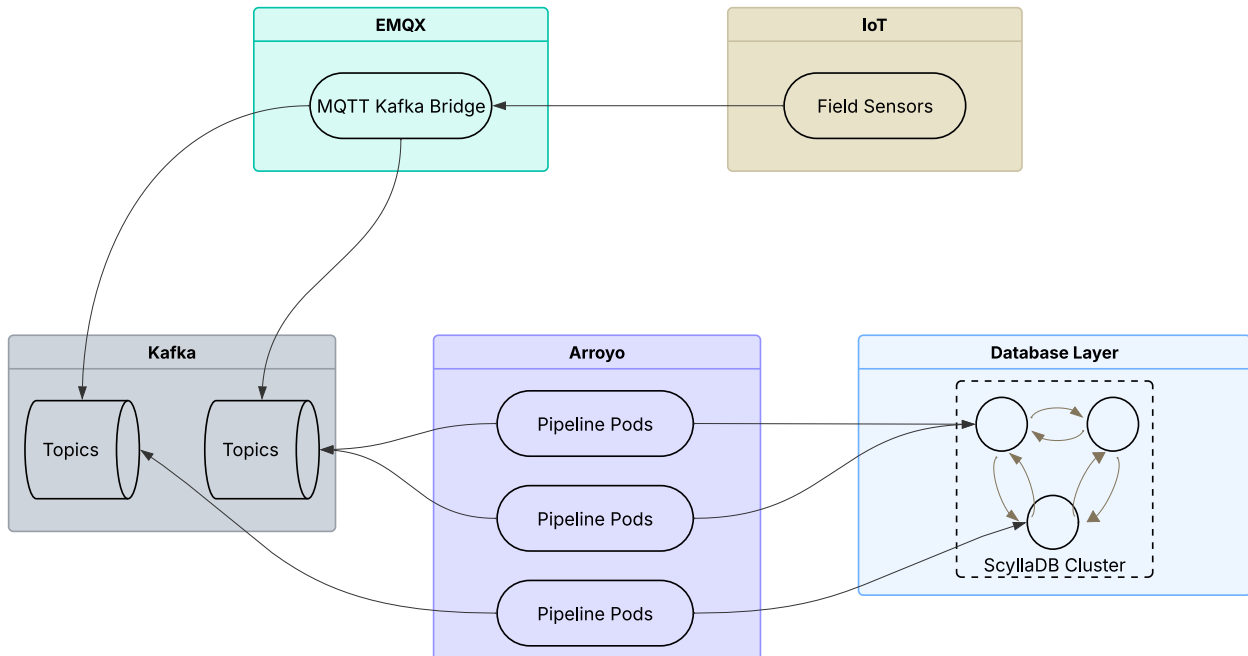
5.2 ΡΟΗ ΔΕΔΟΜΕΝΩΝ ΣΕ ΠΡΑΓΜΑΤΙΚΟ ΧΡΟΝΟ

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

Η αρχιτεκτονική εισαγωγής δεδομένων (Data Ingestion), από τη στρώση παραγωγής των δεδομένων (αισθητήρες πεδίου) μέχρι τη συλλογή τους στο αντίστοιχο Kafka topic μέσω της γέφυρας *MQTT-Kafka*

(MQTT Kafka Bridge), και εν συνεχεία την καταχώριση στη βάση δεδομένων μοντελοποιείται στο ακόλουθο διάγραμμα:



Σχήμα 5.4: Αρχιτεκτονική υποδομής δεδομένων

5.2.2 Τεχνολογίες Streaming

Το σύστημα βασίζεται 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 εγκα-

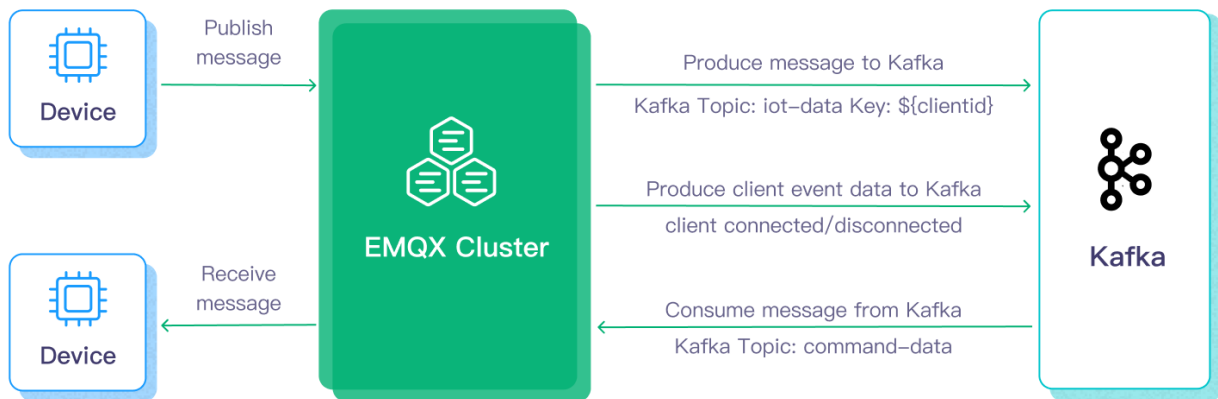
τάσταση, ρύθμιση, κλιμάκωση και αναβάθμιση των brokers, καθώς και στη διαχείριση των *topics* και των *users*. Με αυτόν τον τρόπο αξιοποιούνται τα πλεονεκτήματα του Kubernetes, όπως η ευκολία κλιμάκωσης, η αυτόματη αποκατάσταση υπηρεσιών και η συνεπής διαχείριση πόρων, εξασφαλίζοντας παράλληλα σταθερή 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



Σχήμα 5.5: Αρχιτεκτονική διασύνδεσης EMQX και Apache Kafka ((Προσαρμογή από [39]))

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 διεκ-

περαίωσης 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.

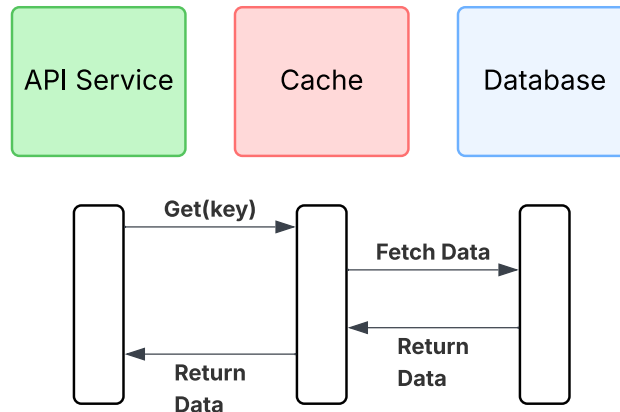
Η διατύπωση των παραπάνω υποθέσεων δεν στοχεύει απλώς στη σύγκριση διαφορετικών υλοποιήσεων, αλλά και στην αξιολόγηση του κατά πόσο η πλατφόρμα μπορεί να υποστηρίξει ετερογενείς drivers χωρίς αλλαγές στην επιχειρησιακή λογική. Η προσέγγιση αυτή επιτρέπει τον διαχωρισμό της λειτουργικότητας της εφαρμογής 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

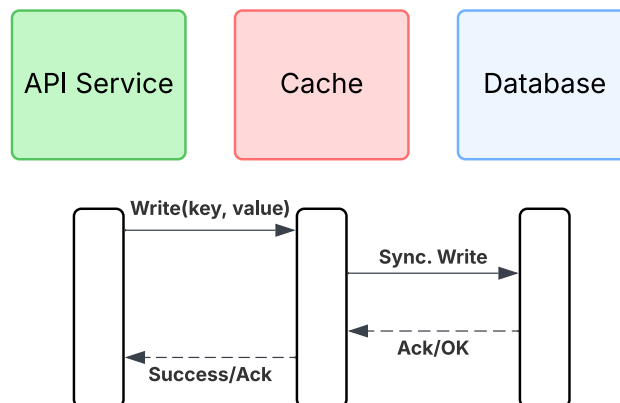
Στο πρότυπο αυτό, όλες οι αναγνώσεις δεδομένων γίνονται μέσω της cache. Σε περίπτωση που το ζητούμενο κλειδί δεν υπάρχει, η ίδια η cache αναλαμβάνει να ανακτήσει τα δεδομένα από τη βάση και να τα αποθηκεύσει πριν τα επιστρέψει στον χρήστη. Το πλεονέκτημα είναι η απλοποίηση της εφαρμογής, καθώς η λογική 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.



Σχήμα 5.6: Read-through στρατηγική

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.



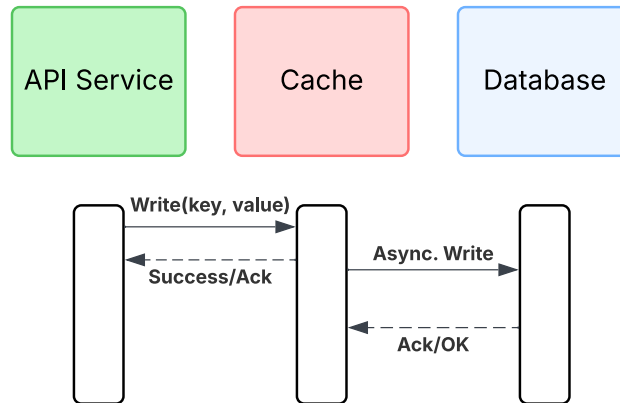
Σχήμα 5.7: Write-through στρατηγική

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,

5.4. ΕΠΙΠΕΔΟ ΥΠΗΡΕΣΙΩΝ ΚΑΙ ΕΦΑΡΜΟΓΗΣ

ενώ η βάση ενδέχεται να μην αντικατοπτρίζει την πιο πρόσφατη κατάσταση.



Σχήμα 5.8: Write-behind στρατηγική

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. Η λεπτομερής αλληλουχία μηνυμάτων (*sequence flow*) και η αρχιτεκτονική υλοποίησης του προτύπου αυτού στην πλατφόρμα αναλύεται διεξοδικά στο Σχήμα 5.9 (σελ. 54).

5.4 ΕΠΙΠΕΔΟ ΥΠΗΡΕΣΙΩΝ ΚΑΙ ΕΦΑΡΜΟΓΗΣ

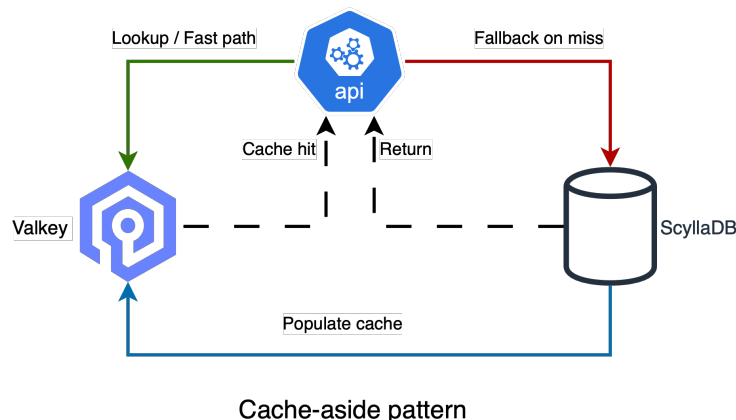
Το επίπεδο υπηρεσιών συγκροτείται γύρω από ένα κεντρικό API gateway, το οποίο αποτελεί τη βασική διεπαφή των χρηστών και των

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 5.9: 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 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.

5.4.3 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.

5.4.4 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 συγκεκριμένων 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.

5.4.5 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.

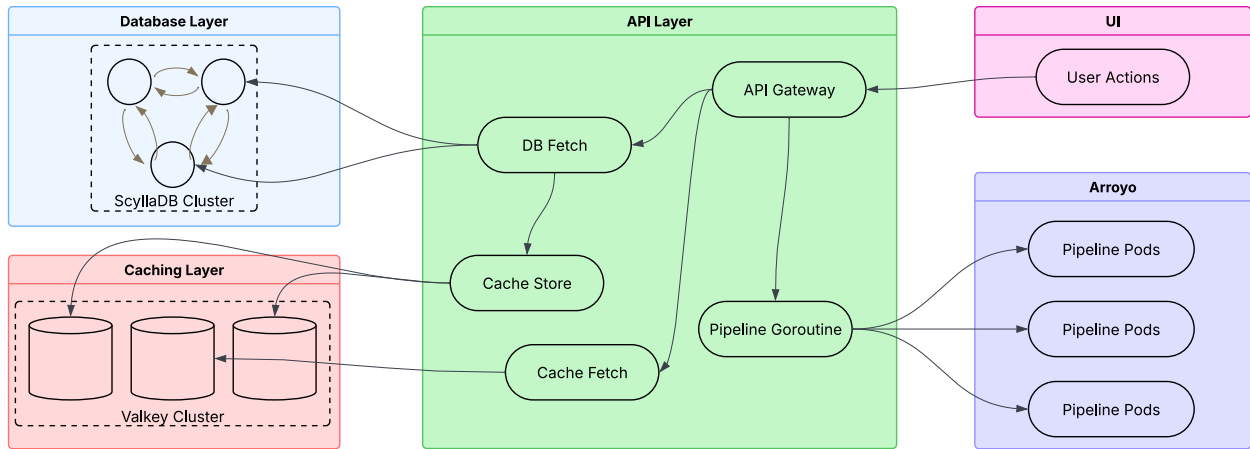
Το UI καταναλώνει το API gateway και παρέχει φόρμες για την εγγραφή αγροτεμαχίων, την ενεργοποίηση αισθητήρων και την διαχείριση των ρυθμίσεών τους. Εμφανίζει επίσης τις πιο πρόσφατες μετρήσεις ανά σημείο, αξιοποιώντας άμεσα τα αποτελέσματα της υπηρεσίας /aggregate. Με αυτόν τον τρόπο το frontend λειτουργεί 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.

Η δομή του επιπέδου υπηρεσιών, συμπεριλαμβανομένης της ροής δεδομένων μεταξύ του API Gateway, του caching layer και της βάσης δεδομένων, παρουσιάζεται στο ακόλουθο σχήμα:

5.5. ΠΑΡΑΤΗΡΗΣΙΜΟΤΗΤΑ

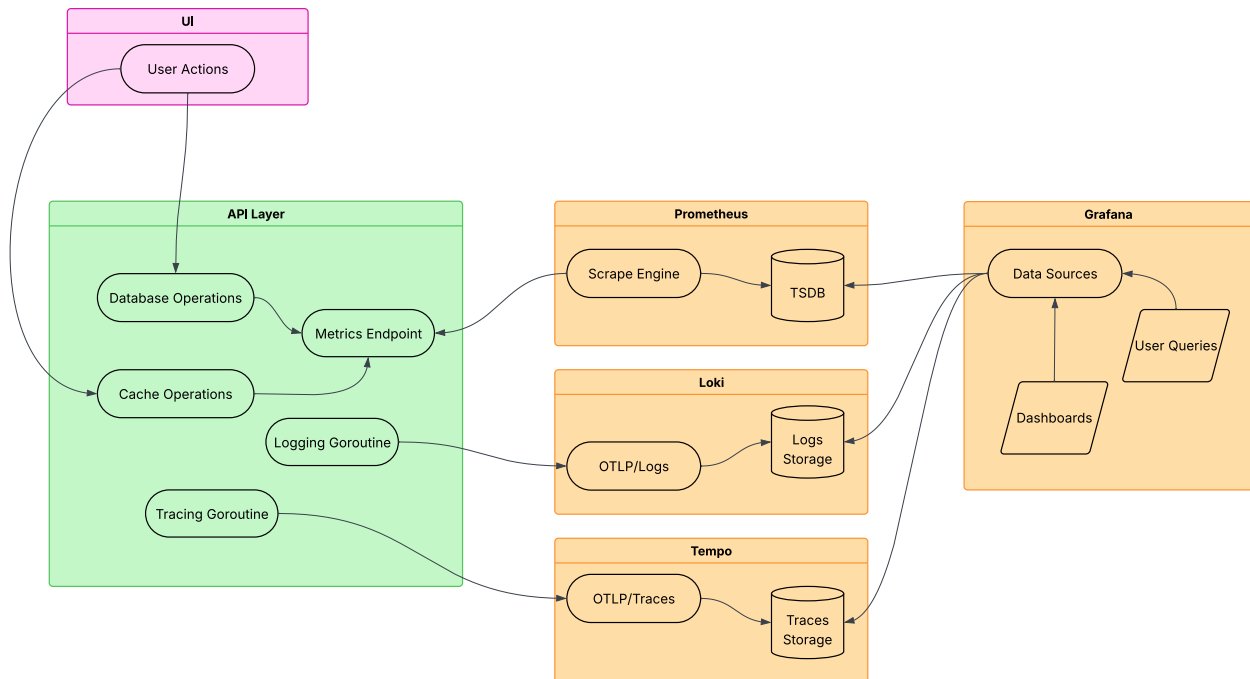


Σχήμα 5.10: Αρχιτεκτονική του επιπέδου υπηρεσιών (API Stack)

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.

Η παρατηρησιμότητα αποτελεί θεμελιώδες χαρακτηριστικό της πλατφόρμας, καθώς επιτρέπει την πλήρη κατανόηση της συμπεριφοράς του συστήματος.



Σχήμα 5.11: Αρχιτεκτονική υποδομής παρατηρησιμότητας

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 Μετρικές και SLI παρακολούθησης

Η προσέγγιση με **προτεραιότητα στην παρατηρησιμότητα** απαιτεί τον σαφή προσδιορισμό των Δεικτών Επιπέδου Υπηρεσίας (Service Level Indicators - SLIs). Προκειμένου η ροή των δεδομένων να είναι πλήρως μετρήσιμη, ορίζονται οι παρακάτω βασικές μετρικές με τη σχετική μαθηματική τους τεκμηρίωση:

1. **Ρυθμός Επιτυχίας Κρυφής Μνήμης (Cache Hit Ratio - CHR):** Εκφράζει την αποδοτικότητα του επιπέδου προσωρινής αποθήκευσης και ορίζεται ως ο λόγος των επιτυχών ανακτήσεων (H) προς το σύνολο των αιτημάτων ανάγνωσης ($H + M$):

$$CHR = \frac{\sum H}{\sum H + \sum M}$$

όπου H τα Cache Hits και M τα Cache Misses.

2. **Μέση Καθυστέρηση (Average Latency - L_{avg}):** Αντιπροσωπεύει τον μέσο χρόνο που μεσολαβεί από την υποβολή ενός αιτήματος έως τη λήψη της απόκρισης για N αιτήματα:

$$L_{avg} = \frac{1}{N} \sum_{i=1}^N (t_{response,i} - t_{request,i})$$

3. **Ρυθμός Διεκπεραίωσης (Throughput - T):** Ορίζεται ως ο αριθμός των επιτυχών λειτουργιών (O) ανά μονάδα χρόνου (Δt):

$$T = \frac{\Delta O}{\Delta t}$$

Οι παραπάνω δείκτες εκτίθενται μέσω *Prometheus exporters* και ομαδοποιούνται στον Πίνακα 5.1, ο οποίος αποτελεί τη βάση για την πειραματική αξιολόγηση στο Κεφάλαιο 6.

5.5. ΠΑΡΑΤΗΡΗΣΙΜΟΤΗΤΑ

SLI	Περιγραφή	Μονάδα	Prometheus Query (Logic)
Cache Latency	Χρόνος απόκρισης του επιλεγμένου driver	ms	<code>rate(...duration_sum) / rate(...count)</code>
Cache Hit Rate	Ποσοστό επιτυχίας του caching layer	%	<code>hits / (hits + misses)</code>
DB Latency	Καθυστερήση ανάγνωσης από ScyllaDB	ms	<code>latency_bucket{le="0.1"}</code>
Throughput	Ρυθμός εξυπηρέτησης του /aggregate	req/s	<code>irate(http_requests_total[1m])</code>
Error Rate	Ποσοστό αποτυχημένων αιτημάτων API	%	<code>rate(errors) / rate(total_calls)</code>

Πίνακας 5.1: Συγκεντρωτικός πίνακας SLI και αντίστοιχων μετρικών Prometheus

Η δυνατότητα συσχέτισης αυτών των σημάτων επιτρέπει την αναγνώριση συστημικών προβλημάτων (όπως I/O wait) που δεν είναι εμφανή όταν μετρώνται μεμονωμένα.

5.5.3 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.

5.5.4 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* → *DBFetch* → *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,

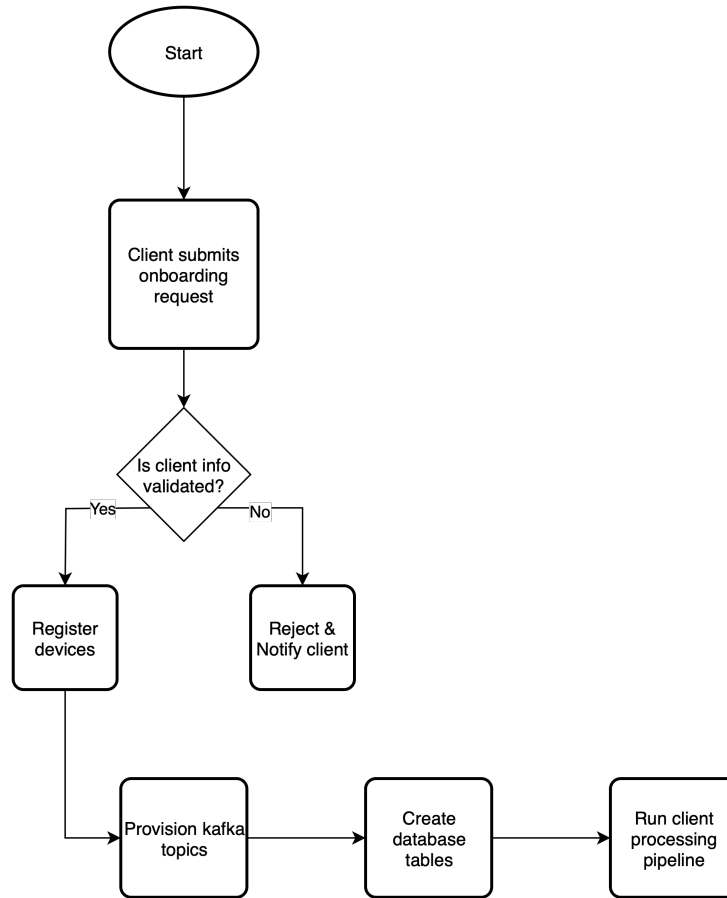
παρέχει πλήρη ορατότητα στο σύστημα cache-aside, προσφέροντας στοιχεία για τη συμπεριφορά κάθε driver χωρίς να απαιτείται ξεχωριστή υλοποίηση debugging.

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 ΔΙΕΠΑΦΗ ΧΡΗΣΤΗ

Η διαδικασία *onboarding* πραγματοποιείται αποκλειστικά μέσω του *API gateway*, το οποίο λειτουργεί ως ενιαίο σημείο εισόδου για όλες τις λειτουργίες που απαιτούνται για την ενεργοποίηση ενός νέου παραγωγού. Αρχικά ο χρήστης καταχωρεί τα στοιχεία του αγροτεμαχίου και των αισθητήρων του· το *API* αναλαμβάνει την ακεραιότητα αυτής της πληροφορίας, τον έλεγχο των μεταδεδομένων και την αποθήκευσή τους στα αντίστοιχα υποσυστήματα της πλατφόρμας. Στη συνέχεια ο παραγωγός μπορεί να αιτηθεί τα διαπιστευτήρια που αντιστοιχούν σε κάθε αισθητήρα. Η δημιουργία των *credentials* γίνεται δυναμικά, ώστε κάθε συσκευή να αποκτήσει ανεξάρτητη ταυτότητα και τα κατάλληλα δικαιώματα πρόσβασης.

Με την ολοκλήρωση αυτών των βημάτων, το *API* εκτελεί τη λογική προετοιμασίας των υποδομών που απαιτούνται για τη ροή δεδομένων. Δημιουργεί τα αντίστοιχα *Kafka topics*, συνδέει το αγροτεμάχιο με το κατάλληλο *Arroyo pipeline* ανάλογα με τον τύπο των μετρήσεων, και ενεργοποιεί το *data path* που θα εξυπηρετεί τον συγκεκριμένο παραγωγό. Παράλληλα το *caching layer* ενημερώνει τα εσωτερικά *indices*, διασφαλίζοντας ότι τα πιο πρόσφατα δείγματα θα είναι άμεσα διαθέσιμα στο *UI* χωρίς καθυστέρηση.



Σχήμα 5.12: Διάγραμμα δραστηριότητας για τη διαδικασία ένταξης νέου πελάτη

Με βάση αυτή τη λογική, η πλατφόρμα δημιουργεί και προετοιμάζει όλους τους αναγκαίους πόρους τη στιγμή που χρειάζονται, ακολουθώντας μια ξεκάθαρα *lazy* στρατηγική. Η εγγραφή ενός νέου χρήστη από μόνη της δεν επιβαρύνει την υποδομή, καθώς δεν εκκινούνται *workloads* πριν χρησιμοποιηθούν τα διαπιστευτήρια από τους αισθητήρες του. Επομένως, το σύστημα αποφεύγει το φαινόμενο της περιττής κατανάλωσης υπολογιστικών πόρων και επεμβαίνει μόνο όταν ο παραγωγός ξεκινήσει να στέλνει δεδομένα προς την πλατφόρμα. Αυτή η προσέγγιση επιτρέπει οριζόντια κλιμάκωση με ελεγχόμενο κόστος και ακριβή αντιστοίχιση πόρων ανά ενεργό πελάτη.

6

Πειράματα

Η παρακάτω ενότητα εξετάζει τη συμπεριφορά των μηχανισμών προσωρινής αποθήκευσης που υποστηρίζουν το endpoint `/aggregate`, το οποίο αποτελεί βασικό συστατικό της πλατφόρμας. 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.

Πριν από την παρουσίαση των αποτελεσμάτων, αξίζει να σημειωθεί ότι το `/aggregate` είναι ιδιαίτερα ευαίσθητο σε καθυστερήσεις, επειδή κάθε αίτημα ενεργοποιεί υπολογισμό παραθύρου με βάση πρόσφατες μετρήσεις αισθητήρων. Το *cache layer* δεν λειτουργεί ως συμπληρωματικό στοιχείο αλλά ως κρίσιμο μέρος του χρόνου διαδρομής των αιτημάτων. Επομένως, ακόμη και μικρές διαφορές στα *percentiles* των καθυστερήσεων αντικατοπτρίζονται άμεσα στη συνολική εμπειρία χρήσης και στη σταθερότητα της πλατφόρμας.

6.1 ΠΕΙΡΑΜΑΤΙΚΟ ΠΛΑΙΣΙΟ ΚΑΙ ΚΟΙΝΗ ΜΕΘΟΔΟΛΟΓΙΑ

Η πειραματική αξιολόγηση βασίζεται στη σύγκριση δύο πλήρως εναλλάξιμων *cache drivers* (*Valkey*, *Memcached*) οι οποίοι χρησιμο-

ποιούνται από το endpoint `/aggregate` υπό συνθήκες συνεχούς φόρτου, προσομοιώνοντας ρεαλιστικά αγροτικά *IoT deployments*.

6.1.1 Στόχος και Κριτήρια Αξιολόγησης

Στόχος της πειραματικής διαδικασίας είναι η αποτίμηση τόσο της ακατέργαστης απόδοσης όσο και της χρονικής σταθερότητας των δύο υποσυστημάτων. Η αξιολόγηση επικεντρώνεται σε τρεις κρίσιμες παραμέτρους:

1. **Απόδοση ανάγνωσης και εγγραφής (I/O Latency)** σε σενάρια *cache hit* και *cache miss*.
2. **Σταθερότητα σε υψηλά percentiles (P95, P99)**, όπου εκδηλώνονται φαινόμενα *tail latency*.
3. **Συμπεριφορά υπό συχνές invalidations**, λόγω χαμηλού TTL και υψηλής κινητικότητας δεδομένων.

Τα percentiles **P95** και **P99** θεωρούνται πρωτεύοντες δείκτες απόδοσης, καθώς η διάμεσος (P50) δεν αποτυπώνει σπάνιες αλλά κρίσιμες καθυστερήσεις που μπορούν να διαταράξουν τη λειτουργία παραθυρικών συναθροίσεων.

6.1.2 Μεθοδολογία και Παραμετροποίηση

Για την απομόνωση των χαρακτηριστικών κάθε υποσυστήματος, δημιουργήθηκαν δύο ανεξάρτητες υλοποιήσεις της πλατφόρμας, μία με *Valkey* και μία με *Memcached*, μέσω κοινού *cache interface*.

Πριν από κάθε δοκιμή εφαρμόστηκε φάση *warm-up* διάρκειας 90 δευτερολέπτων, ώστε να εξλειφθούν φαινόμενα *cold-start*.

Ο πειραματικός φόρτος χαρακτηρίζεται από:

- Σταθερό ρυθμό 350 RPS
- 10 ταυτόχρονες συνδέσεις
- Hit/Miss ratio 70/30
- TTL cache entries 120 s

Η συλλογή μετρήσεων πραγματοποιήθηκε μέσω *Prometheus*, με οπτικοποίηση στο *Grafana*. Ιδιαίτερη έμφαση δόθηκε στη συσχέτιση των *invalidations* με τις καθυστερήσεις εγγραφής και ανάγνωσης.

6.2 ΠΕΙΡΑΜΑ 1: ΒΑΣΙΚΗ ΣΥΓΚΡΙΣΗ ΑΠΟΔΟΣΗΣ ΚΑΙ ΣΤΑ- ΘΕΡΟΤΗΤΑΣ CACHE DRIVERS

6.2.1 Υπόθεση Πειράματος

Υπό συνθήκες σταθερού και συνεχούς φόρτου, το *Memcached* αναμένεται να παρουσιάζει χαμηλότερη μέση καθυστέρηση (*read/write latency*) λόγω της απλούστερης αρχιτεκτονικής του, ενώ το *Valkey* αναμένεται να εμφανίζει μεγαλύτερη χρονική σταθερότητα και μικρότερη διακύμανση στα υψηλά *percentiles* (*P95*, *P99*), ιδιαίτερα σε σενάρια συχνών *invalidations*.

6.2.2 Σχέδιο Πειράματος

Η μόνη μεταβλητή του πειράματος είναι το υποκείμενο *cache driver* (*Memcached* ή *Valkey*).

6.2.3 Δεδομένα Εισόδου

Τα δεδομένα που ανακτώνται από το */aggregate* αντιστοιχούν σε πρόσφατα παράθυρα χρονοσειρών αισθητήρων, οργανωμένα βάσει του σχήματος (*sensor_id*, *bucket_date*). Τα δεδομένα είναι συνθετικά, αλλά βασίζονται σε πραγματικές μετρήσεις πεδίου από αγροτικά *IoT deployments* και αντικατοπτρίζουν τυπική κατανομή προσβάσεων με υψηλή πιθανότητα επαναχρησιμοποίησης δεδομένων (*cache hits*).

6.2.4 Εκτέλεση Δοκιμής

Για κάθε *driver* εκτελέστηκε δοκιμή διάρκειας 10 λεπτών, μετά από φάση σταθεροποίησης (*warmup*) 90 δευτερολέπτων.

6.2.5 Αποτελέσματα Πειράματος

Τα αποτελέσματα καταδεικνύουν σαφή διαφοροποίηση μεταξύ των δύο υποστρωμάτων.

Στο **P95**, το *Memcached* παρουσίασε χαμηλότερη καθυστέρηση τόσο σε αναγνώσεις όσο και σε εγγραφές, με τιμές περίπου 486 μs για *read* και 2.5 *ms* για *write*. Αντίστοιχα, το *Valkey* κατέγραψε υψηλότερες μέσες τιμές, γεγονός που συνάδει με την αυξημένη λειτουργική του πολυπλοκότητα.

6.3. ΠΕΙΡΑΜΑ 2: ΕΠΙΔΡΑΣΗ ΥΠΟΒΕΛΤΙΣΤΗΣ ΧΡΟΝΙΚΗΣ ΠΥΚΝΟΤΗΤΑΣ ΔΕΔΟΜΕΝΩΝ

Στα υψηλότερα percentiles, και ειδικά στο **P99**, η εικόνα αντιστρέφεται: το *Valkey* διατήρησε χαμηλότερη διακύμανση και περιορισμένες αιχμές, με μέγιστες καθυστερήσεις περίπου $4.8\ ms$, ενώ το *Memcached* παρουσίασε σποραδικές αιχμές που έφτασαν τα $6.1\ ms$.

6.2.6 Συμπεράσματα Πειράματος

Το πείραμα επιβεβαιώνει ότι το *Memcached* υπερέχει σε καθαρή απόδοση (*throughput* και χαμηλή μέση καθυστέρηση), όταν το workload είναι ομοιόμορφο και οι καταχωρήσεις μικρές. Ωστόσο, η απλούστερη διαχείριση μνήμης το καθιστά πιο ευάλωτο σε χρονικές αποκλίσεις υπό πίεση.

Αντίθετα, το *Valkey* εμφανίζει πιο προβλέψιμη συμπεριφορά στα υψηλά percentiles, διατηρώντας χαμηλότερη διακύμανση σε περιβάλλοντα συχνής ανανέωσης και invalidation. Η ιδιότητα αυτή το καθιστά καταλληλότερο για συστήματα πραγματικού χρόνου, όπου η χρονική σταθερότητα υπερισχύει της απόλυτης μέσης ταχύτητας.

6.3 ΠΕΙΡΑΜΑ 2: ΕΠΙΔΡΑΣΗ ΥΠΟΒΕΛΤΙΣΤΗΣ ΧΡΟΝΙΚΗΣ ΠΥΚΝΟΤΗΤΑΣ ΔΕΔΟΜΕΝΩΝ

6.3.1 Υπόθεση πειράματος

Η αύξηση της χρονικής πυκνότητας των μετρήσεων ανά αισθητήρα, χωρίς αντίστοιχη αύξηση της πληροφορικής αξίας τους, οδηγεί σε υποβέλτιστα μεγέθη ενδιάμεσων αποτελεσμάτων στο *endpoint /aggregate*. Η αύξηση αυτή αναμένεται να επηρεάσει αρνητικά την απόδοση του cache layer, ιδιαίτερα σε όρους *write latency* και *tail latency*.

6.3.2 Σχέδιο Πειράματος

Το πείραμα εκτελέστηκε ως παραλλαγή του βασικού workload. Η μοναδική μεταβολή αφορά την πυκνότητα των εισαγόμενων μετρήσεων ανά αισθητήρα, προσομοιώνοντας σενάρια όπου τα δεδομένα συλλέγονται σε υπερβολικά μικρά χρονικά διαστήματα.

6.3.3 Δεδομένα Εισόδου

Ορίστηκαν δύο σενάρια:

6.3. ΠΕΙΡΑΜΑ 2: ΕΠΙΔΡΑΣΗ ΥΠΟΒΕΛΤΙΣΤΗΣ ΧΡΟΝΙΚΗΣ ΠΥΚΝΟΤΗΤΑΣ ΔΕΔΟΜΕΝΩΝ

- **Baseline density:** μία μέτρηση ανά αισθητήρα ανά 10 δευτερόλεπτα.
- **Suboptimal density:** μία μέτρηση ανά αισθητήρα ανά 1 δευτερόλεπτο.

Λόγω του σχήματος αποθήκευσης (sensor_id, bucket_date), η αύξηση αυτή οδηγεί σε σημαντικά μεγαλύτερο αριθμό εγγραφών (rows) ανά partition και σε μεγαλύτερα σύνολα δεδομένων που πρέπει να ανακτηθούν και να επεξεργαστούν από το /aggregate endpoint.

Τα δεδομένα είναι συνθετικά, αλλά ευθυγραμμισμένα με ρεαλιστικές αστοχίες ρύθμισης αισθητήρων σε πραγματικά IoT deployments.

6.3.4 Εκτέλεση Δοκιμής

Για κάθε σενάριο εκτελέστηκε δοκιμή διάρκειας 10 λεπτών, μετά από warm-up 90 δευτερολέπτων.

6.3.5 Αποτελέσματα Πειράματος

Τα αποτελέσματα της δοκιμής αποτυπώνουν την επιβάρυνση που προκύπτει από την πυκνότητα των δεδομένων στο επίπεδο αποθήκευσης και ανάκτησης.

Μετρική	Memcached (P95)	Valkey (P95)
Write Latency (baseline)	2.5 ms	4.4 ms
Write Latency (suboptimal)	3.9 ms	5.0 ms
Σχετική Αύξηση	+56%	+14%

Πίνακας 6.1: Αποτελέσματα πειράματος υπό συνθήκες υψηλού granularity δεδομένων.

Στο 99ο ποσοστημόριο, το Memcached παρουσίασε αυξημένες αιχμές έως 7.1 ms, ενώ το Valkey διατήρησε σταθερότερη συμπεριφορά κάτω από 5.6 ms.

6.3.6 Συμπεράσματα Πειράματος

Το πείραμα καταδεικνύει ότι η υποβέλτιστη χρονική πυκνότητα δεδομένων, ακόμη και όταν το σχήμα αποθήκευσης και οι τύποι δεδομένων παραμένουν αμετάβλητοι, μπορεί να οδηγήσει σε σημαντική επιβάρυνση

του *cache layer*. Η αύξηση του αριθμού των μετρήσεων ανά *aggregation window* διευρύνει το μέγεθος των καταχωρήσεων στην *cache* και αυξάνει την πίεση στη διαχείριση μνήμης. Υπό αυτές τις συνθήκες, το *Memcached*, αν και βελτιστοποιημένο για ιδανικά *workloads* με μικρές και ομοιόμορφες καταχωρήσεις, εμφανίζει εντονότερες αποκλίσεις στα υψηλά *percentiles*. Αντίθετα, το *Valkey* διατηρεί μεγαλύτερη ανθεκτικότητα σε αυξημένο *working set* και υποβέλτιστη χρονική πυκνότητα, παρουσιάζοντας πιο προβλέψιμη συμπεριφορά στα υψηλά *percentiles*.

6.4 ΣΥΝΟΛΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

Τα αποτελέσματα των πειραμάτων καταδεικνύουν ότι τα δύο υποσυστήματα δεν αποτελούν απλώς ποσοτικά εναλλακτικές λύσεις, αλλά διαφοροποιούνται ποιοτικά ως προς τα χαρακτηριστικά απόδοσης που προκρίνουν.

Το **Memcached** μεγιστοποιεί την ακατέργαστη απόδοση *I/O*, επιτυγχάνοντας χαμηλότερη μέση καθυστέρηση σε σενάρια ομοιόμορφου φορτίου και μικρών καταχωρήσεων. Η σχεδιαστική του απλότητα και η εστίασή του σε βασικές λειτουργίες κλειδιού–τιμής το καθιστούν κατάλληλο για περιβάλλοντα όπου η μέγιστη δυνατή ταχύτητα υπερισχύει της χρονικής προβλεψιμότητας.

Αντίθετα, το **Valkey** εμφανίζει αυξημένη ανθεκτικότητα σε συνθήκες αυξημένου *working set*, συχνών *invalidations* και υποβέλτιστης χρονικής πυκνότητας δεδομένων, διατηρώντας χαμηλότερη διακύμανση στα υψηλά *percentiles*. Η συμπεριφορά αυτή μεταφράζεται σε μεγαλύτερη χρονική σταθερότητα και προβλεψιμότητα (*deterministic latency*), χαρακτηριστικά κρίσιμα για συστήματα πραγματικού χρόνου. Επιπλέον, η υποστήριξη σύνθετων δομών δεδομένων και μηχανισμών όπως *Lua scripting* προσφέρει δυνατότητες μελλοντικής λειτουργικής επέκτασης.

Με στόχο μια *modular* αρχιτεκτονική, η επιλογή *cache driver* μπορεί να γίνεται με βάση τις επιχειρησιακές απαιτήσεις κάθε deployment, χωρίς τροποποίηση της λογικής της εφαρμογής. Για την παρούσα υλοποίηση τύπου *Minimum Viable Product*, όπου προτεραιότητα αποτελεί η μέγιστη απόδοση *I/O*, το *Memcached* κρίνεται επαρκές και αποδοτικό. Ωστόσο, σε σενάρια αυξημένης πολυπλοκότητας και απαιτήσεων χρονικής σταθερότητας, το *Valkey* αναδεικνύεται ως καταλληλότερη επιλογή.

6.5 ΠΡΟΕΚΤΑΣΕΙΣ

Η παρούσα αξιολόγηση ανοίγει τον δρόμο για μια βαθύτερη πειραματική διερεύνηση των μηχανισμών προσωρινής αποθήκευσης. Συγκεκριμένα, η μελλοντική έρευνα στο επίπεδο των *drivers* θα επικεντρωθεί στην:

- **Ποσοτική ανάλυση κατανάλωσης πόρων:** Συστηματική συσχέτιση του *I/O latency* με τη χρήση μνήμης και *CPU* ανά *driver*, για τον προσδιορισμό του οικονομικού κόστους (*resource cost*) της απόδοσης.
- **Δυναμική συμπεριφορά *invalidations*:** Μελέτη της επίδρασης εναλλακτικών πολιτικών εκκαθάρισης (όπως *LFU* ή *ARC*) σε σενάρια όπου ο ρυθμός ανανέωσης των δεδομένων υπερβαίνει τη χωρητικότητα της μνήμης.
- **Οριζόντια κλιμάκωση (*clustering*):** Αξιολόγηση της συμπεριφοράς του *Valkey* και του *Memcached* σε διατάξεις συμπλέγματος (*cluster*), προκειμένου να μετρηθεί η καθυστέρηση του συγχρονισμού των κόμβων σε συνθήκες υψηλού ανταγωνισμού (*write contention*).

Οι επεκτάσεις αυτές κρίνονται απαραίτητες για την πλήρη κατανόηση των επιπτώσεων κάθε επιλογής σε επίπεδο συστήματος και για την ποσοτική τεκμηρίωση των μελλοντικών αρχιτεκτονικών αποφάσεων της πλατφόρμας. Η βαθύτερη κατανόηση των ορίων κάθε υποστρώματος (*bottlenecks*) θα επιτρέψει τη βέλτιστη παραμετροποίηση της υποδομής σε περιβάλλοντα παραγωγής.

7

Συμπεράσματα

Η παρούσα διπλωματική εργασία πέτυχε τον κύριο στόχο της, που ήταν η ποσοτική αξιολόγηση των επιδόσεων συστημάτων γεωργίας ακριβείας υπό ρεαλιστικές συνθήκες φόρτου, και η διερεύνηση της επίδρασης των δομικών του αποφάσεων (όπως το *TTL* και το *caching*) στις βασικές μετρικές καθυστέρησης. Τα συμπεράσματα που προέκυψαν από την ανάλυση των επιμέρους συστατικών της πλατφόρμας (ScyllaDB, Kafka, EMQX, Arroyo) συνοψίζονται ως εξής:

7.1 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΑΝΘΕΚΤΙΚΟΤΗΤΑ ΚΑΙ ΕΠΕΞΕΡΓΑΣΙΑ ΡΟΩΝ

Η υιοθέτηση μιας σύγχρονης, **Cloud-Native αρχιτεκτονικής** αποτέλεσε τον θεμέλιο λίθο για τη διαχείριση υψηλών ρυθμών εισαγωγής δεδομένων. Η χρήση του *Kafka* ως κεντρικού διαύλου ενοποίησης επέτρεψε την ανεξάρτητη οριζόντια κλιμάκωση των παραγωγών και των καταναλωτών, διασφαλίζοντας παράλληλα μηχανισμούς *back-pressure* για την αποφυγή απώλειας πληροφορίας.

Η αλυσίδα επεξεργασίας ροών **EMQX → Kafka → Arroyo** λειτούργησε ως η κύρια ραχοκοκαλιά της πλατφόρμας. Το *Kafka* διαχειρίστηκε με επιτυχία την **ανεξάρτητη επεκτασιμότητα** (*horizontal scalability*) των *producers* (EMQX) και των *consumers* (Arroyo/ScyllaDB), διασφα-

7.2. ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΑΠΟΔΟΣΗΣ ΚΑΙ ΔΙΑΧΕΙΡΙΣΗ CACHE

λίζοντας παράλληλα μηχανισμούς **back-pressure** και την **μη-απώλεια** δεδομένων. Επιπλέον, η χρήση του *Arroyo* επιβεβαίωσε τη δυνατότητα για **stateful** επεξεργασία με **exactly-once semantics**, γεγονός κρίσιμο για την αξιοπιστία των εξαγόμενων χρονοσειρών σε εφαρμογές γεωργίας ακριβείας.

7.2 ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΑΠΟΔΟΣΗΣ ΚΑΙ ΔΙΑΧΕΙΡΙΣΗ CACHE

Τα βασικά υποσυστήματα επέδειξαν εξαιρετικές επιδόσεις, με τη *ScyllaDB* να διατηρεί το **P95 latency** σε τιμές **<20ms** ακόμη και υπό φόρτο **350 RPS**. Η έρευνα ανέδειξε τη σημασία του επιπέδου προσωρινής αποθήκευσης μέσω δύο αξόνων:

- **Στρατηγική Επιλογή Cache Drivers:** Η σύγκριση μεταξύ *Valkey* και *Memcached* κατέδειξε ένα σαφές *trade-off*. Ενώ το *Memcached* υπερέχει σε απόλυτο *throughput*, η ανθεκτικότητα του *Valkey* στις ακραίες τιμές καθυστέρησης (*P99 tail latency*) τον καθιστά καταλληλότερο για συστήματα πραγματικού χρόνου που απαιτούν προβλεψιμότητα.
- **Data Granularity και TTL:** Διαπιστώθηκε ότι η ενεργός διαχείριση της επικαιρότητας μέσω *TTL* και *invalidations* είναι βιώσιμη χωρίς σημαντική επιβάρυνση. Αντίθετα, η υποβέλτιστη πυκνότητα δεδομένων από τους αισθητήρες μπορεί να αυξήσει την καθυστέρηση εγγραφής στην *cache* έως και 56%, αναδεικνύοντας την ανάγκη για ολιστική παραμετροποίηση από το πεδίο έως το σύννεφο.

7.3 ΟΛΟΚΛΗΡΩΜΕΝΟ STACK ΠΑΡΑΤΗΡΗΣΙΜΟΤΗΤΑΣ

Η υλοποίηση του πλαισίου **monitoring** με *Prometheus* και *Grafana* επέτρεψε την ποσοτική ανάλυση της απόδοσης, αναδεικνύοντας φαινόμενα όπως το **jitter** στην καθυστέρηση εγγραφής λόγω διεργασιών συντήρησης (GC).

Η στρατηγική επέκταση με το **Loki** (Logs) και το **Tempo** με *OpenTelemetry* (Distributed Tracing) ολοκληρώνει ένα πλήρες **Observability Stack**. Η τριπλή αυτή προσέγγιση επιτυγχάνει:

- **Tracing:** Τον εντοπισμό καθυστερήσεων σε επίπεδο υπηρεσίας, επιτρέποντας την παρακολούθηση ενός αιτήματος σε όλη τη διαδρομή του.

7.4. ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ ΚΑΙ ΕΡΕΥΝΗΤΙΚΕΣ ΠΡΟΕΚΤΑΣΕΙΣ

- **Root Cause Analysis (RCA):** Την άμεση συσχέτιση των *logs* με ανωμαλίες στα *metrics* και τα *traces*, επιταχύνοντας τον χρόνο εντοπισμού της ρίζας των προβλημάτων.

Με αυτόν τον τρόπο, η πλατφόρμα αποκτά την πλήρη διαγνωστική ικανότητα που απαιτείται για τη λήψη αυτοματοποιημένων αποφάσεων και την υποστήριξη *self-healing* μηχανισμών.

7.4 ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ ΚΑΙ ΕΡΕΥΝΗΤΙΚΕΣ ΠΡΟΕΚΤΑΣΕΙΣ

Η ολοκλήρωση της παρούσας εργασίας θέτει τις βάσεις για μια σειρά από στρατηγικές επεκτάσεις, οι οποίες στοχεύουν στη μετάβαση αντίστοιχων συστημάτων από ισχυρά αρχιτεκτονικά πρότυπα σε πλήρως αυτόνομα οικοσυστήματα βιομηχανικού επιπέδου. Οι προτεινόμενες κατευθύνσεις συνοψίζονται στους εξής άξονες:

7.4.1 Αρχιτεκτονική Αυτονομία και Δυναμικό Self-Healing

Η πλήρης αξιοποίηση του εγκατεστημένου *Observability Stack* (Metrics, Logs, Traces) επιτρέπει τη μετάβαση από την παθητική επίτηρηση στην ενεργή αυτοθεραπεία του συστήματος.

- **Προσαρμοστικό Circuit-Breaking:** Προτείνεται η ενσωμάτωση μηχανισμών που θα απομονώνουν αυτόματα "αργούς" κόμβους ή δυσλειτουργικούς *cache drivers* σε πραγματικό χρόνο, βασιζόμενοι σε δυναμικά κατώφλια (thresholds) που ορίζονται από τα *traces* του *Tempo*.
- **Auto-scaling βάσει Πίεσης (Back-Pressure):** Η ανάπτυξη ενορχηστρωτών που θα κλιμακώνουν οριζόντια τους *Arroyo workers* ανάλογα με την καθυστέρηση του καταναλωτή (consumer lag) στο *Kafka*, διασφαλίζοντας την τήρηση των *SLAs* καθυστέρησης ακόμη και σε απότομες αυξήσεις της ροής δεδομένων.

7.4.2 Ευφυής Διαχείριση Cache και Μηχανική Μάθηση

Η έρευνα στο επίπεδο της προσωρινής αποθήκευσης μπορεί να επεκταθεί πέρα από τους στατικούς κανόνες *TTL*.

7.4. ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ ΚΑΙ ΕΡΕΥΝΗΤΙΚΕΣ ΠΡΟΕΚΤΑΣΕΙΣ

- **Predictive Pre-caching:** Η χρήση μοντέλων Μηχανικής Μάθησης (όπως *LSTMs* ή *Graph Neural Networks*) για την πρόβλεψη των μοτίβων πρόσβασης των χρηστών μπορεί να επιτρέψει τον προ-υπολογισμό κρίσιμων δεδομένων πριν αυτά ζητηθούν, μειώνοντας δραστικά το *cold-start latency*.
- **Σημασιολογικό Caching (Semantic Caching):** Ανάπτυξη ενός επιπέδου που θα αναγνωρίζει τη σπουδαιότητα των δεδομένων (π.χ. προτεραιότητα σε "κρύα" δεδομένα έναντι μετρήσεων ρουτίνας), προσαρμόζοντας δυναμικά τις πολιτικές εκκαθάρισης (eviction policies).

7.4.3 Edge-to-Cloud και Γεωγραφική Κλιμάκωση

Για τη βελτιστοποίηση των πόρων δικτύου και τη μείωση του *latency*, η επεξεργασία πρέπει να μεταφερθεί εγγύτερα στην πηγή.

- **Αποκεντρωμένο Stream Processing:** Η υλοποίηση μέρους της λογικής του *Arroyo* σε *Edge Gateways* εντός του αγρού θα επιτρέψει το τοπικό φιλτράρισμα και τη συμπίεση των δεδομένων, στέλνοντας στο *Cloud* μόνο τις πληροφορίες που απαιτούνται για ιστορική ανάλυση.
- **Multi-region State Consistency:** Διερεύνηση μεθόδων συγχρονισμού της κατάστασης (*state consistency*) μεταξύ διαφορετικών γεωγραφικών περιοχών, διασφαλίζοντας υψηλή διαθεσιμότητα και επιχειρησιακή συνέχεια (*business continuity*) ακόμη και σε περιπτώσεις ολοκληρωτικής αστοχίας ενός *datacenter*.
- **Multi-cloud Deployment και Vendor Agnosticism:** Ως τελικό στάδιο θωράκισης, προτείνεται η διασπορά της υποδομής σε ετερογενείς παρόχους νέφους (*Multi-cloud strategy*). Η προσέγγιση αυτή εξαλείφει την εξάρτηση από έναν μεμονωμένο πάροχο (*vendor lock-in*) και απομονώνει πλήρως το σύστημα από γενικευμένες αστοχίες υποδομής σε επίπεδο παρόχου, εξασφαλίζοντας το μέγιστο δυνατό επίπεδο ανθεκτικότητας για την κρίσιμες υποδομές.

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] Erik Hollnagel, David D. Woods, and Nancy Leveson. *Resilience Engineering: Concepts and Precepts*. Ashgate Publishing, Ltd., Aldershot, 2006.
- [7] 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.
- [8] Tymoteusz Miller, Grzegorz Mikiciuk, Irmina Durlik, Małgorzata Mikiciuk, Adrianna Łobodzińska, and Marek Śnieg. The iot and

- ai in agriculture: The time is now—a systematic review of smart sensing technologies. *Sensors*, 2025. URL <https://www.mdpi.com/1424-8220/25/12/3583>.
- [9] 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”.
- [10] 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>.
- [11] Eduard Roccatello, Antonino Pagano, Nicolò Levorato, and Massimo Rumor. State of the art in internet of things standards and protocols for precision agriculture with an approach to semantic interoperability. *Network*, 5(2):14, 2025. doi: 10.3390/network5020014.
- [12] Cristiana Amza, Gokul Soundararajan, and Emmanuel Cecchet. Transparent caching with strong consistency in dynamic content web sites. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, pages 264–273, Bangalore, India, 2005. URL <https://www.eecg.toronto.edu/~amza/papers/transparent-cache.pdf>.
- [13] Dan R. K. Ports, Austin T. Clements, Irene Y. Zhang, Samuel Madden, and Barbara Liskov. Transactional consistency and automatic management in an application data cache. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010. URL <https://drkp.net/papers/txcache-osdi10.pdf>.
- [14] Microsoft Docs. Cache-aside pattern. <https://learn.microsoft.com/en-us/azure/architecture/patterns/cache-aside>, 2024-2025. practical guidance on Cache-Aside, problems and considerations (cache lifetime, eviction, consistency).
- [15] Amazon Web Services. Database caching strategies using redis — caching patterns (cache-aside, write-through, write-behind). <https://docs.aws.amazon.com/whitepapers/latest/>

- [database-caching-strategies-using-redis/caching-patterns.html](#), 2024-2025. practical comparison of Cache-Aside vs Write-Through and tradeoffs.
- [16] Jhonny Mertz and Ingrid Oliveira de Nunes. A qualitative study of application-level caching. *IEEE Transactions / preprint (full text available)*, 2016. study of developer challenges in application-level caching; full text on ResearchGate / arXiv, DOI/identifier available in the paper.
- [17] Martin Kleppmann. *Designing Data-Intensive Applications*. O'Reilly Media, 2017.
- [18] Naoufel Boulila. Guidelines for modeling cyber-physical systems – a three-layered architecture for cyber physical systems, 2017.
- [19] Anro Redelinghuys, Anton Basson, and Karel Kruger. A six-layer architecture for the digital twin: a manufacturing case study implementation. *Journal of Intelligent Manufacturing*, 2020.
- [20] Karan Alang and Ajay Kushwaha. Stream processing with apache kafka: Real-time data pipelines. *International Journal of Research in Modern Engineering & Emerging Technology*, 2025.
- [21] Sharon Xie. Where does stream processing fit into your data platform?, 2024. URL <https://www.decodable.co/blog/where-does-stream-processing-fit-into-your-data-platform>.
- [22] ScyllaDB Team. What is a cassandra partition key?, 2024. URL <https://www.scylladb.com/glossary/cassandra-partition-key/>.
- [23] Pragathi B.C., Hrithik Maddirala, and Sneha M. Implementing an effective infrastructure monitoring solution with prometheus and grafana. *International Journal of Computer Applications*, pages 7–15, 2024.
- [24] Yukun Li, Jianshe Cao, Qiang Ye, and Yaoyao Du. Research on visualization and indexing of data based on the elk stack. In *Proceedings of the International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS)*, 2024.
- [25] Brandon Philips. Introducing kubernetes operators: Putting operational knowledge into software. <https://coreos.com/blog/introducing-operators.html>, 2016. Accessed: 2026-03-11.

- [26] 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>.
- [27] 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.
- [28] Red Hat Developers. An introduction to kafka consumers, 2023. URL <https://developers.redhat.com/articles/2023/12/14/introduction-kafka-consumers>.
- [29] 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.
- [30] 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>.
- [31] Arroyo Systems. Arroyo architecture: Control plane and data plane, 2024. URL <https://doc.arroyo.dev/architecture/>.
- [32] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 2010.
- [33] 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.
- [34] ScyllaDB. MongoDB vs scylladb: Architecture comparison, 2023. URL <https://www.scylladb.com/2023/11/02/mongodb-vs-scylladb-architecture-comparison/>.
- [35] Stackademic. Redis: What is master-slave replication and how is it implemented?, 2023. URL <https://blog.stackademic.com/redis-what-is-master-slave-replication-and-how-is-it-implemented-717f750809dc>.

- [36] Zesty. Kubernetes autoscaling: A comprehensive guide, 2024. URL <https://zesty.co/finops-glossary/kubernetes-autoscaling/>.
- [37] Sneha M. Pragathi B.C., Hrithik Maddirala. Implementing an effective infrastructure monitoring solution with prometheus and grafana. *International Journal of Computer Applications*, 2024.
- [38] RedHat Developer. Argo cd. URL <https://redhat-scholars.github.io/argocd-tutorial/argocd-tutorial/index.html>.
- [39] EMQX. Stream mqtt data into apache kafka. URL <https://docs.emqx.com/en/emqx/latest/data-integration/data-bridge-kafka.html>.