



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

Διπλωματική Εργασία

Αρχιτεκτονική ενίσχυση αγροτικών IoT συστημάτων πραγματικού χρόνου

Επιμέλεια:
Δημήτριος Ντέντας
ΑΕΜ: 10446

Επίβλεψη:
Κωνσταντίνος Παναγιώτου
Γιώργος Σιαχάμης

Θεσσαλονίκη, Ιανουάριος 2026

Περιεχόμενα

1	Εισαγωγή	1
2	Επισκόπηση της Ερευνητικής Περιοχής	2
3	Θεωρητικό Υπόβαθρο	7
3.1	Αρχιτεκτονικές CPS και Κατανεμημένα Συστήματα	7
3.2	Streams, Event-Driven και Batch vs Real-Time Processing .	9
3.3	Αρχιτεκτονικές Streams: Lambda και Kappa	10
3.4	Διασύνδεση Υποσυστημάτων	11
3.5	Αποθήκευση, Consistency Models και Caching	12
3.6	Διαχείριση Υποδομής και Cloud-Native Patterns	13
4	Εργαλεία	15
4.1	Kafka	16
4.2	Stream Processing	17
4.2.1	Flink	17
4.2.2	Arroyo	17
4.3	Distributed Storage	18
4.3.1	Cassandra	18
4.3.2	ScyllaDB	19
4.4	Caching	20
4.4.1	Redis	21
4.4.2	Valkey	21
4.4.3	DragonflyDB	22
4.4.4	Memcached	22
4.5	Kubernetes	22
5	Υλοποιήσεις	24
5.1	Αρχιτεκτονική υποδομής	25
5.1.1	Τοπολογία και ρόλοι κόμβων	25
5.1.2	Αυτοματισμοί υποδομής και διακυβέρνηση	27

5.2	Ροή δεδομένων σε πραγματικό χρόνο	27
5.2.1	Πηγές ροής δεδομένων	27
5.2.2	Τεχνολογίες Streaming	28
5.3	Προσωρινή αποθήκευση και cache drivers	29
5.3.1	Αρχιτεκτονική και ενσωμάτωση	29
5.3.2	Επιλογή και συγκριτική αξιολόγηση υποστρωμάτων	30
5.3.3	Πειραματικές υποθέσεις	30
5.3.4	Στρατηγικές caching	31
5.4	Επίπεδο υπηρεσιών και εφαρμογής	32
5.4.1	Valkey cluster	33
5.4.2	API gateway και διαχείριση πόρων	33
5.4.3	Διαχείριση Arroyo pipelines	34
5.4.4	Ανθεκτικότητα και ανοχή σε σφάλματα	34
5.4.5	Υπηρεσία /aggregate και caching layer	35
5.4.6	Frontend και διεπαφές παρακολούθησης	36
5.5	Παρατηρησιμότητα	37
5.5.1	Μετρικές και SLI παρακολούθησης	37
5.5.2	Καταγραφές και διασταύρωση συμβάντων	38
5.5.3	Ιχνηλασιμότητα και ανάλυση αιτημάτων	38
5.6	Ένταξη πελάτη	39
6	Πειράματα	42
6.1	Διατύπωση στόχου και κριτήρια αξιολόγησης	42
6.2	Μεθοδολογία	44
6.2.1	Προετοιμασία και σταθεροποίηση περιβάλλοντος	44
6.2.2	Δομή του Πειραματικού Φορτίου	44
6.2.3	Παρακολούθηση και Μέτρηση	45
6.3	Αποτελέσματα πειράματος	46
6.3.1	Σύγκριση καθυστέρησης	46
6.3.2	Ανάλυση Σταθερότητας σε Υψηλά Percentiles	47
6.3.3	Συμπεριφορά σε Περιβάλλοντα Συχνής Ανανέωσης	48
6.4	Συπεράσματα και ανάλυση	48
6.5	Περαιτέρω επεκτάσεις	49
7	Συμπεράσματα	50
	Βιβλιογραφία	54

ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ

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

Ακρωνύμιο	Περιγραφή
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

Πίνακας 1: Κατάλογος Ακρωνυμίων και Συντομογραφιών

1

Εισαγωγή

2

Επισκόπηση της Ερευνητικής Περιοχής

Στη σύγχρονη γεωργία, ένας δυσλειτουργικός αισθητήρας που περνά απαρατήρητος για ώρες μπορεί να οδηγήσει σε αποτυχημένο πότισμα ή καταστροφή καλλιεργειών. Αυτό το ενιαίο σημείο αποτυχίας αναδεικνύει την ανάγκη για αυτόνομα, ανθεκτικά και παρατηρήσιμα συστήματα σε κλίμακα. Η ανάγκη αυτή εναρμονίζεται με τον οδικό χάρτη (roadmap) προς ανθεκτικά Κυβερνοφυσικά Συστήματα (CPS) που περιγράφεται από τους Ratasich et al. [1], οι οποίοι δίνουν έμφαση στην ανίχνευση ανωμαλιών κατά τη διάρκεια λειτουργίας, στην απομόνωση σφαλμάτων και στην 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], 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. [8] 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 [9].

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. [10] 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 ασφαλή, 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 κατακευματισμένης προσωρινής αποθήκευσης (distributed caching) μπορεί να αυξήσει σημαντικά το throughput και να μειώσει το latency, λειτουργώντας ως «insulation layer» έναντι της συμφόρησης της μόνιμης βάσης δεδομένων. [11, 12]

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. [13, 14, 15]

Με στόχο, λοιπόν, την αρχιτεκτονική ενίσχυση του συστήματος Nostradamus προτείνεται, ως απόρροια των παραπάνω μελετών, η ενσωμάτωση μηχανισμών caching, observability και self-healing. Παρότι υπάρχουν επιμέρους τεχνολογίες που προσφέρουν τέτοιες δυνατότητες, η υιοθέτησή τους σε πραγματικές πολυεπίπεδες IoT πλατφόρμες συνοδεύεται από σημαντικές προκλήσεις. Στα κατακευματισμένα συστήματα, η προσθήκη μιας νέας υπηρεσίας ή λειτουργικότητας συνοδεύεται συχνά από μη προβλέψιμες επιπτώσεις στην απόδοση του συστήματος, όπως αναφέρει επανειλημμένα ο Kleppmann [16]. 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

ενορχήστρωσή τους, με στόχο την επίτευξη αυτονομίας και διαχειρισιμότητας σε cloud-native κατανεμημένα περιβάλλοντα υψηλής πολυπλοκότητας, για την υποστήριξη αυτού του συστήματος πραγματικού χρόνου στο χώρο του *food security*.

3

Θεωρητικό Υπόβαθρο

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

3.1 ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ CPS ΚΑΙ ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ

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

Η σύγχρονη σχεδίαση CPS ακολουθεί τα παρακάτω αρχιτεκτονικά μοτίβα:

- **Layered (στρωματοποιημένη) αρχιτεκτονική:** Καθαρός διαχωρισμός φυσικών, δικτυακών και υπολογιστικών λειτουργιών, ώστε

3.1. ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ CPS ΚΑΙ ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ

κάθε επίπεδο να εξελίσσεται, να ελέγχεται και να συντηρείται χωρίς παρεμβολές στα υπόλοιπα. Η στρωμάτωση επιβάλλει σαφή όρια ευθυνών, ενισχύει τη διαλειτουργικότητα ανάμεσα σε ετερογενείς τεχνολογίες και μειώνει το λειτουργικό κόστος μέσω προβλέψιμων διεπαφών.

- **Κατανεμημένη επεξεργασία:** Τα δεδομένα δεν απαιτείται να συγκεντρώνονται σε έναν κεντρικό κόμβο· μπορούν να υποβάλλονται σε τοπική ή υβριδική επεξεργασία (*edge/fog computing*), μειώνοντας την καθυστέρηση και αυξάνοντας την ανθεκτικότητα. Η διαθέσιμη υπολογιστική ισχύς των αισθητήρων παραμένει κρίσιμος παράγοντας, επειδή συνήθως αποτελούν ετερογενή και χαμηλής ισχύος συστήματα με περιορισμούς σε *CPU*, μνήμη και ενέργεια.

Σε τέτοιες αρχιτεκτονικές απαιτείται ακριβής αντιστοίχιση εργασιών στους κατάλληλους κόμβους, αξιοποίηση *lightweight* μοντέλων μηχανικής μάθησης, μηχανισμοί *offloading* προς ισχυρότερα επίπεδα της στοίβας και πρωτόκολλα ανοχής σφαλμάτων που λειτουργούν χωρίς κεντρικό συντονιστή.

- **Event-driven και streaming αρχιτεκτονική:** Η επεξεργασία δεν βασίζεται σε μαζικά, περιοδικά *batches*. Τα δεδομένα μεταφέρονται ως συνεχείς ροές γεγονότων, επιτρέποντας συστήματα που λειτουργούν σε διαρκή ετοιμότητα. Κάθε νέο γεγονός μπορεί να ενεργοποιήσει λογική σε πραγματικό χρόνο, εξαλείφοντας τα παράθυρα αδράνειας και επιτρέποντας ακρίβεια σε δυναμικά περιβάλλοντα. Η αρχιτεκτονική αυτή απαιτεί σταθερή ροή, χαμηλή καθυστέρηση και μηχανισμούς *backpressure* ώστε να διατηρείται ο έλεγχος στις αιχμές φορτίου.
- **Μικροϋπηρεσίες (Microservices):** Η εφαρμογή διασπάται σε μικρές, αυτόνομες υπηρεσίες με στενά καθορισμένη λειτουργικότητα και ανεξάρτητο κύκλο ζωής. Η απομόνωση επιτρέπει ασύμμετρη κλιμάκωση, στοχευμένες αναβαθμίσεις, ταχύτερες κυκλικές εκδόσεις και περιορισμό σφαλμάτων μέσα σε σαφή όρια. Η αρχιτεκτονική απαιτεί σταθερές συμβάσεις επικοινωνίας, μηχανισμούς παρατήρησης υψηλής ανάλυσης, ανοχή σε μερικές αστοχίες και απόλυτη πειθαρχία στην αποφυγή έμμεσων εξαρτήσεων που καταστρέφουν την αυτονομία.
- **Loose coupling & αναγνωσιμότητα:** Η συνεργασία των υποσυστημάτων υλοποιείται μέσω ασύγχρονων μηνυμάτων, συνήθως με *pub/sub* ή *event-bus* μηχανισμούς, ώστε καμία υπηρεσία να μην

εξαρτάται από τη διαθεσιμότητα ή την εσωτερική δομή κάποιας άλλης. Η χαλαρή σύζευξη μειώνει τα σημεία θραύσης, αποτρέπει κυκλικές εξαρτήσεις, επιτρέπει οριζόντια κλιμάκωση χωρίς ανακατασκευή ροών και καθιστά τη συνολική συμπεριφορά του συστήματος προβλέψιμη υπό μεταβαλλόμενο φορτίο.

3.2 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.

3.3 ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ STREAMS: LAMBDA ΚΑΙ 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. 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.

Οι δύο προσεγγίσεις δεν απαιτούν 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.

3.4 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.

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.

3.5 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. Η επιλογή wide-column store γίνεται συχνά για την υποστήριξη μεγάλων ροών δεδομένων με ευέλικτο σχήμα και αποτελεσματική οργάνωση ανά χρονικά διαστήματα (*date bucketing*), επιτρέποντας ταχεία προσπέλαση σε ιστορικά δεδομένα χωρίς να επηρεάζεται η απόδοση των real-time queries.

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 wide-column stores, κάθε bucket 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 αναδιάρθρωση ή ανακατασκευή των δεδομένων.

Η επιλογή **consistency model** παραμένει κρίσιμη. Το strong consistency εξασφαλίζει ότι όλοι οι κόμβοι βλέπουν την ίδια κατάσταση, ενώ το eventual consistency μειώνει latency και αυξάνει διαθεσιμότητα, αποδεχόμενο προσωρινή ασυνέπεια. Causal και session-based models προσφέρουν 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

3.6. ΔΙΑΧΕΙΡΙΣΗ ΥΠΟΔΟΜΗΣ ΚΑΙ CLOUD-NATIVE PATTERNS

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.

3.6 ΔΙΑΧΕΙΡΙΣΗ ΥΠΟΔΟΜΗΣ ΚΑΙ CLOUD-NATIVE PATTERNS

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

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

3.6. ΔΙΑΧΕΙΡΙΣΗ ΥΠΟΔΟΜΗΣ ΚΑΙ CLOUD-NATIVE PATTERNS

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 [17]. Η αρχιτεκτονική των CPS βασίζεται σε τρία βασικά επίπεδα: το φυσικό επίπεδο (physical layer), όπου καταγράφονται και παράγονται τα δεδομένα μέσω αισθητήρων· το επίπεδο δικτύου (network layer), που εξασφαλίζει τη μεταφορά των δεδομένων· και το κυβερνητικό ή υπολογιστικό επίπεδο (cyber layer), όπου λαμβάνονται αποφάσεις βάσει των εισερχόμενων δεδομένων. Η συνύπαρξη αυτών των στρωμάτων σε ένα κοινό σύστημα καθιστά τα CPS ιδιαίτερα κατάλληλα για εφαρμογές που απαιτούν χαμηλή καθυστέρηση, αξιοπιστία και αυτονομία. Όπως προαναφέρθηκε, στο πεδίο της γεωργίας ακριβείας, τα CPS διαδραματίζουν καθοριστικό ρόλο, καθώς συνδυάζουν αισθητήρες πεδίου, μηχανισμούς ελέγχου άρδευσης, και αλγορίθμους πρόβλεψης βασισμένους σε δεδομένα για να εξασφαλίσουν βέλτιστες συνθήκες καλλιέργειας. Οι τεχνολογίες αυτές επιτρέπουν τη δυναμική λήψη αποφάσεων, μειώνουν τις απώλειες και αυξάνουν την αποδοτικότητα σε όλα τα στάδια της παραγωγής.

4.1 KAFKA

Για να επιτευχθεί όμως η πλήρης δυναμική των CPS, απαιτούνται ισχυρές υποδομές διασύνδεσης και διαχείρισης δεδομένων. Εδώ εντάσσεται η ανάγκη για αξιόπιστες messaging πλατφόρμες, όπως το Apache Kafka, που διασφαλίζουν την συνεχή και αξιόπιστη ροή πληροφοριών ανάμεσα στα υποσυστήματα ενός CPS. Το Apache Kafka αποτελεί ένα, πλέον, *de facto* πρότυπο για την υλοποίηση τέτοιων messaging υποδομών, χάρη στη δυνατότητα του να αποθηκεύει τα μηνύματα σε μορφή καταγραφής (log-based) [18]. 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 [19] επιβεβαιώνουν ότι το Kafka παρουσιάζει σημαντικό πλεονέκτημα σε όρους throughput και fault tolerance σε σχέση με άλλες προσεγγίσεις, γεγονός που το καθιστά κατάλληλο για απαιτητικές IoT εφαρμογές με αυξημένο όγκο και ταχύτητα δεδομένων.

Σε ένα σύστημα όπως το *Nostradamus*, όπου η ροή των δεδομένων είναι αδιάκοπη και εξελίσσεται σε πραγματικό χρόνο, η ταχεία εισαγωγή και εξαγωγή των δεδομένων εκτιμάται, τόσο για λόγους απόδοσης όσο και για την ελαχιστοποίηση της κατανάλωσης υπολογιστικών πόρων. Δοθέντος ενός συνόλου αισθητήρων τοποθετημένων σε έναν αγρό, οι οποίοι παράγουν συνεχώς δεδομένα, ο κεντρικός broker πρέπει να τα λαμβάνει ορθά και εντός λογικών χρονικών πλαισίων, ενώ ταυτόχρονα να τα επεξεργάζεται χωρίς να καταπονεί το συνολικό σύστημα. Συνεπώς, πρέπει να ληφθούν υπόψη τόσο η ποιότητα υπηρεσίας (Quality of Service - QoS) της messaging υποδομής όσο και οι εγγυήσεις παράδοσης που αυτή προσφέρει. Στη συγκεκριμένη εφαρμογή, η απώλεια ενός μεμονωμένου δείγματος αισθητήρα δεν επηρεάζει σημαντικά τη συνολική ανάλυση, επομένως η πολιτική "**at-least-once**" αποτελεί μια ασφαλή και επαρκή επιλογή. Οι Kreps et al. [20] 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 *Flink* αποτελεί ώριμη και πλήρως εξοπλισμένη λύση για επεξεργασία ροών δεδομένων, η πολυπλοκότητα και οι απαιτήσεις του JVM περιβάλλοντος δεν ήταν απαραίτητες για την πλατφόρμα *Nostradamus*. Για αυτόν τον λόγο επιλέχθηκε το *Arroyo*, ένα open-source σύστημα επεξεργασίας stream γραμμένο σε *Rust*, το οποίο προσφέρει χαμηλότερο latency, μικρότερο αποτύπωμα μνήμης και υψηλότερη απόδοση σε περιβάλλοντα cloud-native.

Το *Arroyo*, αν και δεν υποστηρίζει ακόμα όλες τις δυνατότητες του *Flink*, καλύπτει επαρκώς τις απαιτήσεις της πλατφόρμας *Nostradamus*

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 πλατφόρμας, 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 τα overheads 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 εγγράφων 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) [21]. Σε περιβάλλοντα IoT, όπου οι συσκευές παράγουν συνεχώς δεδομένα τηλεμετρίας (π.χ. θερμοκρασία, τάση, ρεύμα) με υψηλή

συχνότητα, η Cassandra μπορεί να λειτουργήσει ως backend αποθήκευσης για ροές δεδομένων σχεδόν σε πραγματικό χρόνο, διασφαλίζοντας υψηλή διαθεσιμότητα και συνεχή εγγραφή με χαμηλό latency. Η προσέγγιση αυτή έχει εφαρμοστεί επιτυχώς σε σενάρια όπως η παρακολούθηση φωτοβολταϊκών μονάδων μέσω Raspberry Pi [22], 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 πλατφόρμα Nostradamus 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.

Σε σύγκριση με την Cassandra, η ScyllaDB παρέχει:

- Χαμηλότερο latency: χάρη στο ασύγχρονο, shard-per-core μοντέλο που μειώνει lock contention αλλά και τις καθυστερήσεις λόγω GC.
- Πλήρη αξιοποίηση multi-core: κάθε CPU core διαχειρίζεται ένα shard, επιτρέποντας γραμμική κλιμάκωση με την προσθήκη κόμβων.
- Προβλέψιμη απόδοση: μειωμένη διακύμανση στο latency 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 πραγματικά real-time εφαρμογές IoT και CPS.

4.4 CACHING

Η προσωρινή αποθήκευση (caching) αποτελεί κρίσιμο στοιχείο σε CPS και πλατφόρμες όπως το Nostradamus, 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 οπτικοποιημένη 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 χρησιμοποιούνται ή εξετάστηκαν στο πλαίσιο του Nostradamus, 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 χαμηλό latency 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.

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. Για πλατφόρμες όπως το Nostradamus, όπου η υποστήριξη μακροπρόθεσμης συντήρησης και η διαλειτουργικότητα είναι κρίσιμες, 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 replacement, 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 ζευγών 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 συστήματος σε υψηλά επίπεδα, ακόμη και σε συνθήκες έντονου φορτίου ή υλικών αποτυχιών.

Πέραν της αυτοματοποιημένης διαχείρισης πόρων, το *Kubernetes* παρέχει μια ενιαία πλατφόρμα παρατηρησιμότητας. Μέσω της ενσωμάτωσης εργαλείων όπως το *Prometheus* για συλλογή μετρικών, το *Grafana* για οπτικοποίηση και το *AlertManager* για την αποστολή ειδοποιήσεων,

[23] - κοινώς τη **lingua franca** του *observability* - ενισχύεται η κατανόηση της δυναμικής συμπεριφοράς του συστήματος σε πραγματικό χρόνο. Η εγγενής υποστήριξη μηχανισμών για service discovery, load balancing και declarative configuration, επιτρέπει την ευέλικτη ανάπτυξη και τον συντονισμό μικροϋπηρεσιών, διευκολύνοντας την επίτευξη στόχων υψηλής διαθεσιμότητας και επεκτασιμότητας σε cloud-native περιβάλλοντα. Συνεπώς, υιοθετώντας αντίστοιχα πρότυπα, μπορούν να προληφθούν ανεπιθύμητες καταστάσεις και να διασφαλιστεί η διατήρηση της ομαλής λειτουργίας ακόμα και υπο συνθήκες υψηλής πολυπλοκότητας.

5

Υλοποιήσεις

Η υλοποίηση του *Nostradamus* ακολουθεί πιστά τα συμπεράσματα της θεωρητικής ανάλυσης: το οικοσύστημα IoT που εξυπηρετεί γεωργικά deployments απαιτεί πλατφόρμα που να είναι ταυτόχρονα ανθεκτική, παρατηρήσιμη και επεκτάσιμη. Για τον λόγο αυτόν επιλέχθηκε ως θεμέλιο το *Kubernetes*, καθώς παρέχει ενιαίο μοντέλο ορισμού υποδομής, ώριμα primitives 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 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.

5.1.1 Τοπολογία και ρόλοι κόμβων

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

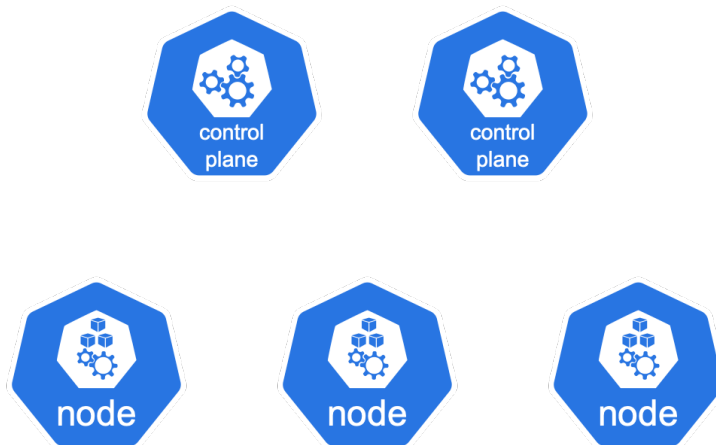
- **Control plane:** Φιλοξενεί τον *API server*, τον *scheduler*, τον *controller manager* και το *etcd*, διατηρώντας την κατάσταση του cluster και λαμβάνοντας αποφάσεις τοποθέτησης.

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

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

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

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



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

5.1.2 Αυτοματισμοί υποδομής και διακυβέρνηση

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

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

5.2 ΡΟΗ ΔΕΔΟΜΕΝΩΝ ΣΕ ΠΡΑΓΜΑΤΙΚΟ ΧΡΟΝΟ

5.2.1 Πηγές ροής δεδομένων

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

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

5.2.2 Τεχνολογίες Streaming

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

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

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

Ωστόσο, η ενσωματωμένη υλοποίηση του *Strimzi* δεν υποστηρίζει

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

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

5.3 ΠΡΟΣΩΡΙΝΗ ΑΠΟΘΗΚΕΥΣΗ ΚΑΙ CACHE DRIVERS

5.3.1 Αρχιτεκτονική και ενσωμάτωση

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

5.3.2 Επιλογή και συγκριτική αξιολόγηση υποστρωμάτων

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

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

5.3.3 Πειραματικές υποθέσεις

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

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

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

5.3.4 Στρατηγικές caching

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

Read-through

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

Write-through

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

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

Write-behind (ή Write-back)

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

Cache-aside (ή Lazy loading)

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

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

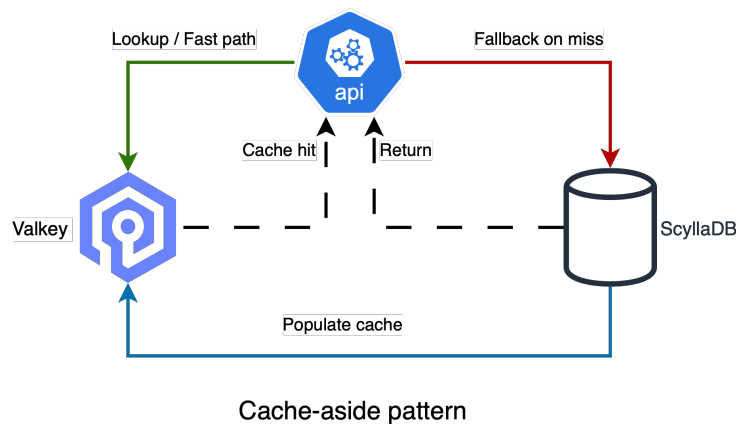
5.4 ΕΠΙΠΕΔΟ ΥΠΗΡΕΣΙΩΝ ΚΑΙ ΕΦΑΡΜΟΓΗΣ

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

5.4.1 Valkey cluster

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

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



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

5.4.2 API gateway και διαχείριση πόρων

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

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

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

5.4.3 Διαχείριση Arroyo pipelines

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

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

5.4.4 Ανθεκτικότητα και ανοχή σε σφάλματα

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

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

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

5.4.5 Υπηρεσία /aggregate και caching layer

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

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

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

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

5.4.6 Frontend και διεπαφές παρακολούθησης

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

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

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

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

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

5.5 ΠΑΡΑΤΗΡΗΣΙΜΟΤΗΤΑ

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

5.5.1 Μετρικές και SLI παρακολούθησης

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

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

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

5.5.2 Καταγραφές και διασταύρωση συμβάντων

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

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

5.5.3 Ιχνηλασιμότητα και ανάλυση αιτημάτων

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

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

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

Ιχνηλασιμότητα στο caching layer

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

Έτσι, το τελικό trace απεικονίζει ολόκληρο το μονοπάτι *CacheFetch* → *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.

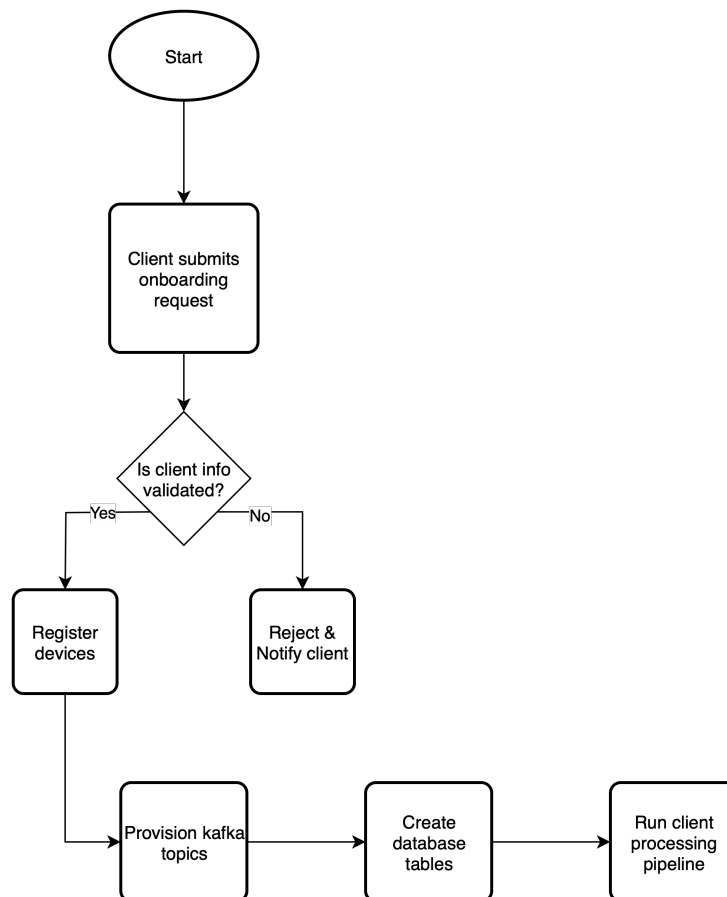
Σε ευρύτερο πλαίσιο, η ιχνηλασιμότητα αποτελεί βασικό μηχανισμό αποσφαλμάτωσης σε κατανεμημένα περιβάλλοντα, όπου οι πιθανές αιτίες καθυστέρησης δεν εντοπίζονται σε μία μόνο διεργασία αλλά προκύπτουν από την αλληλεπίδραση πολλαπλών μικροϋπηρεσιών. Η δυνατότητα οπτικοποίησης των spans επιτρέπει την κατανόηση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

5.6 ΈΝΤΑΞΗ ΠΕΛΑΤΗ

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 5.3: Διάγραμμα δραστηριότητας για τη διαδικασία ένταξης νέου πελάτη

Με βάση αυτή τη λογική, η πλατφόρμα δημιουργεί και προετοιμά-

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

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 ΔΙΑΤΥΠΩΣΗ ΣΤΟΧΟΥ ΚΑΙ ΚΡΙΤΗΡΙΑ ΑΞΙΟΛΟΓΗΣΗΣ

Στόχος της πειραματικής διαδικασίας είναι η αποτίμηση της απόδοσης και της σταθερότητας δύο πλήρως εναλλάξιμων *cache drivers*

(Valkey, Memcached) όταν χρησιμοποιούνται από το /aggregate υπό συνθήκες συνεχούς φόρτου, όπως αυτές που προσομοιάζουν ένα αγροτικό deployment. Η αξιολόγηση επικεντρώνεται σε τρεις κρίσιμες παραμέτρους, οι οποίες αντικατοπτρίζουν τις πραγματικές απαιτήσεις ενός συστήματος IoT με συνεχή ροή δεδομένων:

1. **Απόδοση ανάγνωσης και εγγραφής (I/O Latency).** Εξετάζονται συμμετρικά τα μονοπάτια *cache hit* και *cache miss*, καθώς αμφότερα συνεισφέρουν στον τελικό χρόνο απόκρισης του endpoint /aggregate και την αποφόρτιση της μόνιμης βάσης δεδομένων.
2. **Σταθερότητα σε υψηλά percentiles (P95, P99).** Για συστήματα πραγματικού χρόνου, η μέση τιμή (P50) είναι ανεπαρκής δείκτης. Τα υψηλά percentiles (P95 και άνω) είναι εκεί όπου εκδηλώνονται οι χρονικές αποκλίσεις (tail latencies) που επηρεάζουν ουσιαστικά την εμπειρία χρήστη και την προβλεψιμότητα του συστήματος.
3. **Συμπεριφορά σε περιβάλλοντα συχνής ανανέωσης (High Invalidation Rate).** Η χαμηλή χρονική διάρκεια ζωής (TTL) στις χρονοσειρές αισθητήρων προκαλεί συχνές invalidations και επαναλαμβανόμενες εισαγωγές νέων τιμών. Επομένως, ο driver πρέπει να ανταποκρίνεται ορθά σε workloads που χαρακτηρίζονται από υψηλή κινητικότητα δεδομένων.

Οι παράμετροι αυτές δεν επιλέχθηκαν θεωρητικά αλλά προέκυψαν από πραγματικές ανάγκες του Nostradamus. Σε αγροτικά project, όπου παράγονται χιλιάδες μετρήσεις ανά λεπτό, η πλατφόρμα στηρίζεται στην ικανότητα της cache να προσφέρει άμεση πρόσβαση σε πρόσφατα δεδομένα ώστε να αποφεύγεται η συνεχής επικοινωνία με την κεντρική βάση δεδομένων. Το P95 και το P99 θεωρούνται ιδιαίτερα κρίσιμα, διότι οι ακραίες τιμές μπορούν να διαταράξουν τον ρυθμό με τον οποίο διαμορφώνονται τα παράθυρα, οδηγώντας σε φαινόμενα «συσσώρευσης καθυστερήσεων». Η επιλογή αυτών των ποσοστημορίων (percentiles) είναι ζωτικής σημασίας για τη διαχείριση της καθυστέρησης ουράς (tail latency). Το P95 αντιπροσωπεύει τον μέγιστο χρόνο απόκρισης που παρατηρείται στο 95% των αιτημάτων, διασφαλίζοντας μια ικανοποιητική εμπειρία για τη συντριπτική πλειοψηφία των λειτουργιών. Αντίστοιχα, το P99 εστιάζει στο πιο αργό 1% των αιτημάτων, το οποίο είναι κρίσιμο σε συστήματα μεγάλης κλίμακας ή σε λειτουργίες που αφορούν πολλαπλά βήματα, όπου ακόμη και μία σπάνια καθυστέρηση μπορεί να οδηγήσει σε σωρευτική επιβράδυνση και να επηρεάσει αρνητικά τη λήψη αποφάσεων σε πραγματικό χρόνο.

Το **P50** (50ό ποσοστημόριο), ή αλλιώς διάμεσος (median), αντιπροσωπεύει τον χρόνο απόκρισης στον οποίο ολοκληρώνεται το 50% του συνόλου των αιτημάτων. Αυτή η μετρική περιγράφει την τυπική εμπειρία του μέσου χρήστη ή την «κανονική» απόδοση του συστήματος. Ωστόσο, η διάμεσος δεν είναι επαρκής ως πρωταρχικός δείκτης απόδοσης (SLI) σε συστήματα πραγματικού χρόνου, διότι είναι ευάλωτη στο να παραπλανά, ειδικά όταν η κατανομή των καθυστερήσεων είναι ασύμμετρη (skewed). Αν ο μέσος χρόνος απόκρισης είναι χαμηλός, αλλά υπάρχουν πολλές σπάνιες, αργές αποκρίσεις (ακραίες τιμές), το **P50** παραμένει σταθερό, δίνοντας μια ψευδή αίσθηση ασφάλειας. Για το λόγο αυτό, το **P50** χρησιμοποιείται κυρίως για τον εντοπισμό γενικευμένης υποβάθμισης της απόδοσης στην απόδοση, αλλά η διαχείριση της ποιότητας και της αξιοπιστίας ενός καταναμεμένου συστήματος, όπως μια πλατφόρμα *IoT*, απαιτεί την παρακολούθηση των υψηλότερων ποσοστημορίων **P95** και **P99**.

6.2 ΜΕΘΟΔΟΛΟΓΙΑ

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

6.2.1 Προετοιμασία και σταθεροποίηση περιβάλλοντος

Πριν την εκτέλεση των μετρήσεων εφαρμόστηκε φάση προθέρμανσης (*warm-up*) διάρκειας 90 δευτερολέπτων. Κατά το διάστημα αυτό σταθεροποιούνται οι εσωτερικές δομές μνήμης και μειώνονται τα *cold-start* φαινόμενα. Χωρίς αυτή τη διαδικασία, οι μετρήσεις θα περιείχαν θόρυβο από την αρχική δέσμευση της μνήμης (allocation activity) ή από αρχικές διεργασίες συντήρησης.

6.2.2 Δομή του Πειραματικού Φορτίου

Ο φόρτος παράχθηκε μέσω συνθετικού load generator, ο οποίος σχεδιάστηκε ώστε να μιμείται το προφίλ αιτημάτων ενός πραγματικού γεωργικού deployment, όπου οι συσκευές παράγουν συνεχή ροή δεδομένων. Ο φόρτος χαρακτηρίστηκε ως εξής:

- **Σταθερός Ρυθμός:** Ο φόρτος διατηρήθηκε σε ένα σταθερό επίπεδο 350 *Requests Per Second* (RPS), προσομοιώνοντας ένα συνεχή φόρτο εργασίας από πληθώρα συνδεδεμένων συσκευών.
- **Επίπεδο Ταυτόχρονης Εκτέλεσης (concurrency):** Για την αξιολόγηση της σταθερότητας σε συνθήκες πίεσης, ο *generator* χρησιμοποίησε 10 ταυτόχρονα threads/συνδέσεις (concurrent connections) καθ' όλη τη διάρκεια της δοκιμής.
- **Μείγμα Αιτημάτων:** Η κατανομή των λειτουργιών διαμορφώθηκε ώστε να αντικατοπτρίζει ένα ρεαλιστικό σενάριο χρήσης του *API*, όπου οι περισσότερες αναγνώσεις είναι επιτυχημένες (Hit) και υπάρχει μια συνεχής, αλλά μικρότερη, ανάγκη για ανανέωση (Miss/Write):
 - **Cache Hit:** 70%
 - **Cache Miss / Write:** 30%

Το πρότυπο αυτό του φορτίου ανταποκρίνεται πιστά στη φυσιολογική λειτουργία του */aggregate*, όπου οι ίδιες μετρήσεις ανακτώνται επανειλημμένα μέσα σε διαδοχικά παράθυρα (υψηλή πιθανότητα hit), ενώ οι νέες μετρήσεις εισάγονται με ρυθμό που προκαλεί συνεχή εγγραφή και invalidation. Η επιλογή ενός σταθερού ρυθμού 350 RPS βασίστηκε σε μετρήσεις πεδίου που κατέδειξαν ότι ο συγκεκριμένος φόρτος αντιπροσωπεύει τυπικό σενάριο ενός μεσαίου αγροτικού σταθμού.

6.2.3 Παρακολούθηση και Μέτρηση

Η συλλογή των μετρήσεων πραγματοποιήθηκε μέσω του συστήματος παρακολούθησης *Prometheus*, ενώ η οπτικοποίηση των χρονοσειρών έγινε στο *Grafana*. Δόθηκε έμφαση στην παρακολούθηση του *P95 latency*, στις καθυστερήσεις εγγραφής (write latency) και στον ρυθμό *invalidations*.

Στο πλαίσιο του monitoring και του observability, ο ρυθμός των *invalidations* αποτελεί κρίσιμη μετρική, ειδικά όταν το *Time-To-Live* (TTL) των δεδομένων ορίζεται σκόπιμα σε χαμηλή τιμή. Τα *invalidations* είναι τα συμβάντα κατά τα οποία τα αποθηκευμένα δεδομένα κρίνονται παρωχημένα (stale) και πρέπει να απομακρυνθούν από την *cache*. Η παρακολούθηση της συχνότητας αυτών των συμβάντων είναι απαραίτητη για δύο λόγους:

- **Πρώτον:** επιβεβαιώνεται η ορθή λειτουργία του μηχανισμού εκκαθάρισης, διασφαλίζοντας τη διατήρηση της επικαιρότητας των δεδομένων, ανάλογα με τις ανάγκες της εφαρμογής.
- **Δεύτερον:** παρέχεται βαθύτερη παρατηρησιμότητα στις επιπτώσεις που έχει αυτή η επιβαλλόμενη συχνή εκκαθάριση στους πόρους του συστήματος.

Ο υψηλός ρυθμός *invalidations* συνεπάγεται αυξημένο υπολογιστικό κόστος (CPU, I/O) για την ανανέωση των δεδομένων. Ο συσχετισμός του με το **P95 read και write latency** επιτρέπει την ακριβή αξιολόγηση της επιβάρυνσης στην απόκριση του συστήματος, προσφέροντας έτσι κρίσιμα *insights* για τη βελτιστοποίηση των επιδόσεων.

Η βασική παραμετροποίηση των *drivers* παρέμεινε στην προεπιλεγμένη μορφή τους, εκτός από το *TTL* των εγγραφών, το οποίο ορίστηκε στα 120 δευτερόλεπτα, διασφαλίζοντας ότι η πρόσβαση των *drivers* γίνεται σε δεδομένα μέγιστης επικαιρότητας, ανάλογα με τον ρυθμό συλλογής του πεδίου και επιβάλλοντας συχνή εκκαθάριση (eviction/invalidation).

6.3 ΑΠΟΤΕΛΕΣΜΑΤΑ ΠΕΙΡΑΜΑΤΟΣ

Τα αποτελέσματα καταδεικνύουν σαφή διαφοροποίηση ως προς τη συμπεριφορά των δύο υποστρομάτων, τόσο σε επίπεδο ακατέργαστης ταχύτητας όσο και σε επίπεδο χρονικής σταθερότητας.

6.3.1 Σύγκριση καθυστέρησης

Η ποσοτικοποίηση της καθυστέρησης στο 95ο εκατοστημόριο (P95) αποτυπώνει την εικόνα που επηρεάζει τους περισσότερους χρήστες και επιβεβαιώνει την ταχύτητα του Memcached.

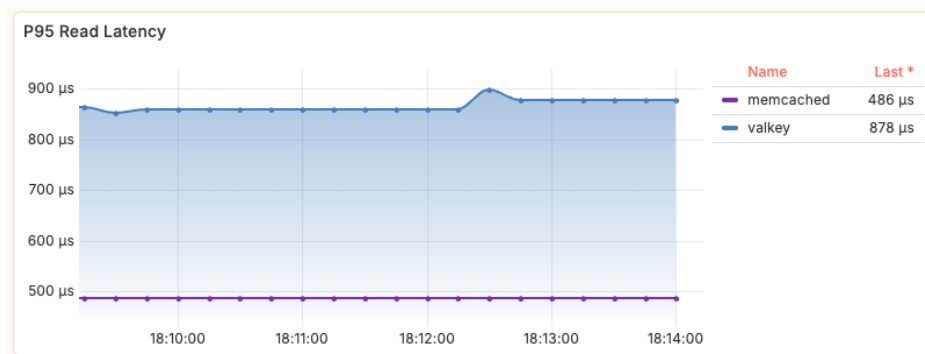
Μετρική	Memcached (μ s)	Valkey (μ s)	Σχετική Διαφορά
Ανάγνωση	486	878	$\approx 1.80 \times$ ταχύτερο
Εγγραφή	2500	4400	$\approx 1.76 \times$ ταχύτερο

Πίνακας 6.1: Σύγκριση Καθυστέρησης (Latency) Valkey έναντι Memcached στο 95ο Εκατοστημόριο (P95)

Όπως φαίνεται στον Πίνακα 6.1 και επιβεβαιώνεται από το Γράφημα 6.1, το *Memcached* εμφανίζει σημαντικά χαμηλότερη καθυστέρηση

6.3. ΑΠΟΤΕΛΕΣΜΑΤΑ ΠΕΙΡΑΜΑΤΟΣ

σε αναγνώσεις, με τιμές P95 που κινούνται στα 486 μ s. Αυτό το αποτέλεσμα καθιστά το Memcached περίπου 1.8 φορές ταχύτερο από το Valkey στην καθαρή διεκπεραίωση αιτημάτων. Η επίδοση αυτή συνδέεται άμεσα με τον λιτό χαρακτήρα του πρωτοκόλλου του (Memcached binary protocol) και τη φύση του ως ένα καθαρό *Key-Value store*, το οποίο αποφεύγει πολυπλοκότητα και επιτρέπει υψηλό throughput με περιορισμένο αποτύπωμα πόρων.



Σχήμα 6.1: Σύγκριση των καμπυλών καθυστέρησης ανάγνωσης (Read Latency) Valkey έναντι Memcached υπό συνεχή φόρτο (P95).

6.3.2 Ανάλυση Σταθερότητας σε Υψηλά Percentiles

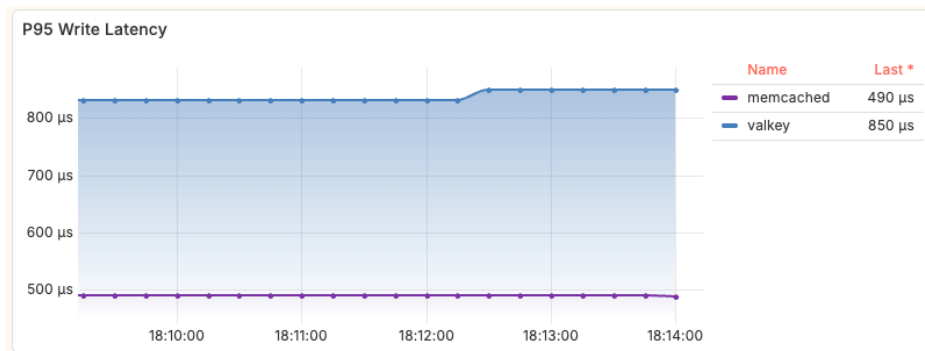
Ενώ το Memcached υπερέχει σε μέγιστη ταχύτητα I/O, όπως φαίνεται στο Γράφημα 6.1, το Valkey υπερέχει ως προς τη χρονική σταθερότητα και την προβλεψιμότητα σε ακραίες συνθήκες. Αυτή η διαφορά εκδηλώνεται ιδιαίτερα στα υψηλότερα percentiles, όπως το P99, όπου το Valkey διατηρεί χαμηλότερη διακύμανση (low variance) στην καθυστέρηση.

Σε συνθήκες αύξησης του concurrency και του ρυθμού invalidations, οι καμπύλες latency του Valkey, αν και υψηλότερες σε μέση τιμή (P95), παρουσιάζουν μικρότερες χρονικές αποκλίσεις και είναι λιγότερο πιθανό να εμφανίσουν τις απότομες αιχμές (spikes) που παρατηρούνται περιστασιακά στο Memcached κατά την εκτέλεση εργασιών συντήρησης. Για παράδειγμα, η καθυστέρηση στο 99ο εκατοστημόριο (P99) για το Valkey ήταν 4.8 ms, ενώ για το Memcached, λόγω της διαχείρισης πόρων και των περιστασιακών καθυστερήσεων, έφτασε τα 6.1 ms.

Η συμπεριφορά αυτή είναι κρίσιμη για συστήματα πραγματικού χρόνου, καθώς η προβλεψιμότητα (deterministic latency) υπερισχύει της απόλυτης μέσης ταχύτητας.

6.3.3 Συμπεριφορά σε Περιβάλλοντα Συχνής Ανανέωσης

Η ρύθμιση του TTL στα 120 δευτερόλεπτα επέβαλε συνεχή φόρτο εκκαθάρισης και επανεισαγωγής δεδομένων.



Σχήμα 6.2: Καθυστέρηση εγγραφής (P95 Write Latency) στους δύο drivers υπό συνθήκες χαμηλού TTL. Η γραφική απεικόνιση αναδεικνύει τις διαφορές στη διαχείριση της συμφόρησης και των λειτουργιών εκκαθάρισης.

Όπως φαίνεται στο Γράφημα 6.2, η συμπεριφορά των δύο υποσυστημάτων παρουσιάζει ποιοτικές διαφορές κατά την εκτέλεση write operations:

- **Memcached:** Λόγω της απλής διαχείρισης μνήμης (εστιασμένης στην ταχύτητα), το Memcached έδειξε μεγαλύτερη ευαισθησία στις λειτουργίες εκκαθάρισης, οδηγώντας σε μικρές, αλλά μετρήσιμες, αυξήσεις στο *P95/P99 latency* όταν ο ρυθμός *invalidation* κορυφωνόταν.
- **Valkey:** Η δομημένη διαχείριση της μνήμης και των διαδικασιών του (όπως οι πολιτικές εκκαθάρισης LRU) του επέτρεψαν να διατηρήσει χαμηλότερη διακύμανση (low variance) στην καθυστέρηση, ακόμη και όταν το υποσύστημα μνήμης ήταν υπό πίεση λόγω των συχνών λήξεων (expirations).

6.4 ΣΥΠΕΡΑΣΜΑΤΑ ΚΑΙ ΑΝΑΛΥΣΗ

Ως εκ τούτου, τα δύο συστήματα δεν αποτελούν απλώς εναλλακτικές που διαφέρουν ποσοτικά, αλλά ποιοτικά:

- **Memcached:** Επιτυγχάνει την υψηλότερη **ακατέργαστη ταχύτητα I/O**. Είναι ιδανικό για ελαφριές διεργασίες όπου η κύρια ανάγκη είναι η μέγιστη δυνατή throughput και η εφαρμογή χρησιμοποιεί μόνο βασικές λειτουργίες κλειδιού-τιμής.
- **Valkey:** Προσφέρει την υψηλότερη **χρονική σταθερότητα** και προβλεψιμότητα (deterministic latency), ακόμη και σε περιόδους υψηλής συμφόρησης. Επιπλέον, διαθέτει επεκτασιμότητα (π.χ., υποστήριξη για *Atomic Blocks*, *Lua Scripts*, σύνθετες δομές δεδομένων) που το καθιστά καταλληλότερο για μελλοντική ενσωμάτωση πιο σύνθετων λειτουργιών της πλατφόρμας *Nostradamus*.

Με δεδομένη τη *modular* αρχιτεκτονική της πλατφόρμας *Nostradamus*, η επιλογή *driver* μπορεί να καθοδηγείται από τις επιχειρησιακές ανάγκες κάθε deployment, χωρίς καμία αλλαγή στον κώδικα της εφαρμογής. Για την παρούσα, υπεραπλουστευμένη υλοποίηση (Minimum Viable Product - MVP), το *Memcached* είναι επαρκές και προτιμητέο λόγω της ανώτερης απόδοσης I/O.

6.5 ΠΕΡΑΙΤΕΡΩ ΕΠΕΚΤΑΣΕΙΣ

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

- Διερεύνηση της επίδρασης των invalidations σε διαφορετικούς ρυθμούς εισαγωγής νέων μετρήσεων,
- Αποτύπωση κατανάλωσης μνήμης και CPU ανά driver,
- Μελέτη συμπεριφοράς σε συνθήκες οριζόντιας κλίμακωσης,

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

7

Συμπεράσματα

Η παρούσα διπλωματική εργασία πέτυχε τον κύριο στόχο της, που ήταν η ποσοτική αξιολόγηση των επιδόσεων του συστήματος **Nostradamus** 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:

1. **Cloud-Native Αρχιτεκτονική:** Η υιοθέτηση μιας σύγχρονης, **Cloud-Native αρχιτεκτονικής** με βάση τεχνολογίες όπως *Kafka* για την ενοποίηση των δεδομένων, *ScyllaDB* για την αποθήκευση χρονοσειρών και *Arroyo* για την επεξεργασία ροών, απέδειξε την ικανότητα της πλατφόρμας να διαχειρίζεται υψηλούς ρυθμούς εισαγωγής δεδομένων. Η αρχιτεκτονική είναι ανθεκτική σε διακοπές, θέτοντας τα θεμέλια για *self-healing* συμπεριφορές.
2. **Αποδοτικότητα και Αξιοπιστία στην Επεξεργασία Ροών:** Η αλυσίδα επεξεργασίας ροών (*streaming pipeline*) *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*), διασφαλίζοντας *back-pressure* και **μη-απώλεια** δεδομένων. Η χρήση του *Arroyo* για την επεξεργα-

σία των ροών επιβεβαίωσε τη δυνατότητα για **stateful** (διατηρώντας έτσι την κατάσταση) και **exactly-once semantics** (ακριβώς μια φορά επεξεργασίας), γεγονός κρίσιμο για την αξιοπιστία των εξαγόμενων χρονοσειρών.

3. **Βέλτιστη Απόδοση Υποσυστημάτων:** Τα βασικά υποσυστήματα του Nostradamus επέδειξαν εξαιρετικές επιδόσεις. Ειδικότερα, η βάση δεδομένων *ScyllaDB* διατήρησε το P95 latency σε **εξαιρετικά χαμηλά επίπεδα** (τιμές < 20 ms) ακόμη και κάτω από τον προσομοιωμένο φόρτο 350 RPS, επιβεβαιώνοντας την καταλληλότητά της για εφαρμογές χρονοσειρών με απαιτήσεις *low latency*.
4. **Κρισιμότητα του TTL και των Invalidations:** Η μεθοδολογία της ρύθμισης του *TTL* και η παρακολούθηση του ρυθμού *invalidations* απέδειξαν ότι η ενεργός διαχείριση της επικαιρότητας των δεδομένων μπορεί να γίνει χωρίς σημαντική επιβάρυνση. Το κόστος της συχνής εκκαθάρισης απορροφάται αποτελεσματικά, γεγονός που καθιστά το *TTL* ένα στρατηγικό εργαλείο για τον έλεγχο της κατανάλωσης πόρων μνήμης και την εξασφάλιση του *data freshness*.
5. **Ανάλυση Επιδόσεων μέσω Metrics (Prometheus/Grafana):** Η υλοποίηση του πλαισίου **monitoring** με *Prometheus* και *Grafana* επέτρεψε την **ποσοτική ανάλυση** και την ανάδειξη λεπτομερειών της απόδοσης του συστήματος. Ειδικότερα:
 - Η παρακολούθηση μηχανισμών όπως ο ρυθμός *invalidations* (ως δείκτης κόστους εκκαθάρισης) επιβεβαίωσε την αποτελεσματική διαχείριση της επικαιρότητας των δεδομένων.
 - Ο εντοπισμός του *jitter* στο *write latency* (που πιθανότατα οφείλεται σε GC ή διεργασίες συντήρησης) παρείχε κρίσιμα *insights* για τα υποκείμενα σημεία συμφόρησης που πρέπει να αντιμετωπιστούν για την περαιτέρω βελτιστοποίηση της πλατφόρμας.

Αυτή η μεθοδολογία τεκμηρίωσε την τρέχουσα απόδοση και έθεσε τη βάση για τη διαγνωστική ικανότητα.

6. **Ολοκληρωμένο Stack Παρατηρησιμότητας:** Η στρατηγική επέκταση του πλαισίου με την ενσωμάτωση των υποσυστημάτων **Loki** για τη συλλογή **Logs** και **Tempo** με *OpenTelemetry* για **Distributed Tracing** ολοκληρώνει το *Observability Stack* (Metrics, Logs, Traces). Αυτή η τριπλή προσέγγιση είναι θεμελιώδης για την υποστήριξη **self-healing** συμπεριφορών και επιτυγχάνει:

-
- **Tracing:** Τον εντοπισμό και τη διάγνωση καθυστερήσεων σε επίπεδο υπηρεσίας, επιτρέποντας την παρακολούθηση ενός μεμονωμένου αιτήματος καθ' όλη τη διαδρομή της μέσα από τις κατανεμημένες υπηρεσίες, καθώς και τα επιμέρους στάδια των υπηρεσιών.
 - **Root Cause Analysis (RCA):** Την άμεση συσχέτιση των καταγεγραμμένων συμβάντων (*logs*) με τις ανώμαλες τιμές των *metrics* και των *traces*, επιταχύνοντας δραστικά τον χρόνο εντοπισμού της ρίζας του προβλήματος.

Με αυτόν τον τρόπο, η πλατφόρμα Nostradamus αποκτά την πλήρη διαγνωστική ικανότητα που απαιτείται για τη λήψη αυτοματοποιημένων αποφάσεων.

7. **Επιχύρωση Λύσης Caching:** Οι δοκιμές στο */aggregate endpoint* έδειξαν ότι το ενσωματωμένο *caching* λειτουργεί όπως αναμενόταν, επιτρέποντας την γρήγορη ανάκτηση των ίδιων μετρήσεων (*high hit probability*) και μειώνοντας την επιβάρυνση στην υποκείμενη βάση δεδομένων.

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] 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".
- [9] 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>.
- [10] Eduard Roccattello, 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.
- [11] 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>.
- [12] 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>.
- [13] 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).
- [14] 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.
- [15] 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.
- [16] Martin Kleppmann. *Designing Data-Intensive Applications*. O'Reilly Media, 2017.
 - [17] 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>.
 - [18] 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.
 - [19] 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.
 - [20] 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>.
 - [21] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 2010.
 - [22] 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.
 - [23] Sneha M. Pragathi B.C., Hrithik Maddirala. Implementing an effective infrastructure monitoring solution with prometheus and grafana. *International Journal of Computer Applications*, 2024.