



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΙΓΑΙΟΥ

UNIVERSITY OF THE
AEGEAN

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ
ΣΥΣΤΗΜΑΤΩΝ

ΠΜΣ ΠΛΗΡΟΦΟΡΙΑΚΑ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

**“Development of QoS-Aware, Multi-Cloud Autoscaling and Monitoring
Platform for Kubernetes Workloads**

—

***Ανάπτυξη πλατφόρμας με επίγνωση ποιότητας υπηρεσίας για
παρακολούθηση και αυτο-κλιμάκωση φόρτων εργασίας Kubernetes
κατά μήκος νεφών”***

Επόπτης: Κρητικός Κυριάκος

Μέλος Επιτροπής: Γκουμόπουλος Χρήστος

Μέλος Επιτροπής: Σκούτας Δημήτριος

Φοιτητής: Κοσμόπουλος Ανδρέας

A.M. : 3252024012

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ	2
ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ	5
ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ	7
ΚΑΤΑΛΟΓΟΣ ΣΥΝΤΟΜΟΓΡΑΦΙΩΝ	8
ΠΕΡΙΛΗΨΗ	10
ABSTRACT	11
ΕΥΧΑΡΙΣΤΙΕΣ	12
ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ	14
1.1 ΠΡΟΣΔΙΟΡΙΣΜΟΣ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ	14
1.2 ΠΡΟΤΕΙΝΟΜΕΝΗ ΛΥΣΗ & ΣΤΟΧΟΙ	15
1.3 ΔΟΜΗ ΑΝΑΦΟΡΑΣ	15
ΚΕΦΑΛΑΙΟ 2: ΥΠΟΒΑΘΡΟ	17
2.1 CLOUD COMPUTING	17
2.1.1 Ορισμός	17
2.1.2 Τα 5 Βασικά Χαρακτηριστικά Cloud Computing	17
2.1.3 Πλεονεκτήματα Cloud Computing	18
2.1.4 Μοντέλα Παράδοσης Υπηρεσιών	18
2.1.5 Μοντέλα Ανάπτυξης	19
2.1.6 Υπηρεσίες Εξισορρόπησης Φορτίου & Αυτο-κλιμάκωσης	21
2.1.7 Υπηρεσίες Παρακολούθησης	22
2.1.8 Εικονικοποίηση: VMs vs Containers	23
2.1.9 Docker — Η Τεχνολογία Containerization	24
2.1.10 Microservices & Cloud-Native Εφαρμογές	25
2.2 KUBERNETES	27
2.2.1 Ορισμός & Ιστορικό	27
2.2.2 Πλεονεκτήματα Kubernetes	27
2.2.3 Αρχιτεκτονική Kubernetes	28
2.2.4 Βασικά Kubernetes Objects	29
2.2.5 Μοντελοποίηση Αρχιτεκτονικής Διάταξης — 6 Επίπεδα	31
2.2.6 Τρόποι Διάταξης σε Kubernetes	32
2.2.7 Εργαλεία Τοπικής Ανάπτυξης	33
2.2.8 Εργαλεία Παρακολούθησης Kubernetes	33
2.2.9 Κλιμάκωση Pods & Κόμβων	34
2.3 INFRASTRUCTURE AS CODE (IAC) & TERRAFORM	36
2.3.1 Τι είναι το Infrastructure as Code	36
2.3.2 Βασικά Χαρακτηριστικά & Οφέλη IaC	36
2.3.3 Declarative vs Imperative Μοντέλο	37
2.3.4 Σύγκριση Εργαλείων IaC	38
2.3.5 Terraform — Αναλυτική Παρουσίαση	39
2.3.6 HashiCorp Configuration Language (HCL)	39
2.3.7 Αρχιτεκτονική Terraform	40
2.3.8 Κύκλος Εργασίας Terraform	41
2.3.9 Τρόποι Χρήσης Terraform	42
2.3.10 Υποστηριζόμενοι Cloud Providers & Υπηρεσίες	42
ΚΕΦΑΛΑΙΟ 3: ΣΧΕΤΙΚΕΣ ΕΡΓΑΣΙΕΣ	44
3.1 ΕΙΣΑΓΩΓΗ	44
3.2 ΕΡΓΑΣΙΑ 1: «KUBERNETES AUTOSCALING: YOYO ATTACK VULNERABILITY AND MITIGATION» (BEN DAVID & BREMLER-BARR, 2021)	44

3.3 ΕΡΓΑΣΙΑ 2: «ADAPPF: SELF-ADAPTIVE SCRAPE INTERVAL FOR MONITORING IN GEO-DISTRIBUTED CLUSTER FEDERATIONS» (HUANG & PIERRE, 2023).....	45
3.4 ΕΡΓΑΣΙΑ 3: «AUTOPILOT: WORKLOAD AUTOSCALING AT GOOGLE» (RZADCA ET AL., 2020).....	45
3.5 ΕΡΓΑΣΙΑ 4: «CROSSPLANE — A UNIVERSAL CONTROL PLANE FOR CLOUD INFRASTRUCTURE» (CROSSPLANE AUTHORS, 2024).....	46
3.6 ΕΡΓΑΣΙΑ 5: «TORCH: A TOSCA-BASED ORCHESTRATOR OF MULTI-CLOUD CONTAINERISED APPLICATIONS» (TOMARCHIO ET AL., 2021)	47
3.7 ΣΥΓΚΡΙΤΙΚΟΣ ΠΙΝΑΚΑΣ ΕΡΓΑΣΙΩΝ	47
3.8 ΣΥΜΠΕΡΑΣΜΑΤΑ.....	49
ΚΕΦΑΛΑΙΟ 4: ΑΝΑΠΤΥΞΗ ΣΥΣΤΗΜΑΤΟΣ	50
4.1 ΜΟΝΤΕΛΟ ΑΝΑΠΤΥΞΗΣ — WATERFALL.....	50
4.2 ΑΝΑΛΥΣΗ ΑΠΑΙΤΗΣΕΩΝ	50
4.2.1 Είδη Χρηστών.....	50
4.2.2 Λειτουργικές Απαιτήσεις Χρηστών (FA-U).....	51
4.2.3 Λειτουργικές Απαιτήσεις Εφαρμογών (FA-A)	52
4.2.4 Μη Λειτουργικές Απαιτήσεις.....	54
4.2.5 Προαιρετικές Απαιτήσεις.....	57
4.3 ΣΧΕΔΙΑΣΗ ΣΥΣΤΗΜΑΤΟΣ.....	58
4.3.1 Διάγραμμα Περιβάλλοντος (Context Diagram)	59
4.3.2 Διάγραμμα Συστατικών (Component Diagram).....	59
4.3.3 Διαγράμματα Περιπτώσεων Χρήσης	63
4.3.4 Διαγράμματα Ακολουθίας	65
4.3.5 Διάγραμμα Κλάσεων.....	72
4.4 ΥΛΟΠΟΙΗΣΗ	77
4.4.1 Τεχνολογική Στοιβά.....	77
4.4.2 Δομή Πηγαίου Κώδικα	78
4.4.3 Το Terraform στο Πλαίσιο της Πλατφόρμας	78
4.4.4 Σημεία Πρόσβασης REST API.....	82
4.4.5 Ομάδες Κλάσεων Υλοποίησης	83
4.4.6 Διεπαφή Χρήστη της Πλατφόρμας.....	85
4.6 ΔΟΚΙΜΗ ΚΑΙ ΑΞΙΟΛΟΓΗΣΗ ΑΠΟΔΟΣΗΣ (4Ο ΣΤΑΔΙΟ ΤΟΥ ΜΟΝΤΕΛΟΥ WATERFALL)	86
4.6.1 Στρατηγική Δοκιμών.....	86
4.6.2 Σύγκριση Μετρήσεων με Στόχους SLA	88
4.6.3 Μετρικές Αναφοράς (Baseline).....	89
4.6.4 Χρονογραμμή Αυτόματης Κλιμάκωσης HPA	90
4.6.5 Επαλήθευση Απαιτήσεων στο Πλαίσιο της Επίδειξης	91
4.7 ΒΑΘΜΟΣ ΙΚΑΝΟΠΟΙΗΣΗΣ ΑΠΑΙΤΗΣΕΩΝ.....	92
4.7.1 Κλίμακα Αξιολόγησης και Κριτήρια	92
4.7.2 Λειτουργικές Απαιτήσεις Διαχείρισης Χρηστών	93
4.7.3 Λειτουργικές Απαιτήσεις Εφαρμογών	94
4.7.4 Μη Λειτουργικές Απαιτήσεις.....	97
4.7.5 Συνοπτική Αποτίμηση	101
ΚΕΦΑΛΑΙΟ 5: ΕΓΚΑΤΑΣΤΑΣΗ ΚΑΙ ΕΠΙΔΕΙΞΗ ΣΥΣΤΗΜΑΤΟΣ	103
5.1 ΟΔΗΓΟΣ ΕΓΚΑΤΑΣΤΑΣΗΣ	103
5.1.1 Προαπαιτούμενα	103
5.1.2 Εγκατάσταση Συστήματος.....	104
5.2 ΕΠΙΔΕΙΞΗ ΛΕΙΤΟΥΡΓΙΑΣ ΣΥΣΤΗΜΑΤΟΣ.....	108
5.2.1 Περιβάλλον, Ονοματολογία και Επαλήθευση Εγκατάστασης	108
5.2.2 Σενάριο 1: Διάταξη και Επαλήθευση της Συστάδας	112
5.2.3 Σενάριο 2: Παρακολούθηση σε Πραγματικό Χρόνο	115
5.2.4 Σενάριο 3: Αυτόματη Κλιμάκωση βάσει QoS Κανόνα	118
5.2.5 Σενάριο 4: Η Διαδικτυακή Διεπαφή της Πλατφόρμας	123
5.3 ΣΥΝΟΨΗ ΕΠΑΛΗΘΕΥΣΗΣ ΑΠΑΙΤΗΣΕΩΝ ΜΕΣΩ ΤΗΣ ΕΠΙΔΕΙΞΗΣ	125
5.4 ΠΕΡΙΟΡΙΣΜΟΙ ΤΗΣ ΕΠΙΔΕΙΞΗΣ	126

ΚΕΦΑΛΑΙΟ 6: ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ	127
6.1 ΣΥΜΠΕΡΑΣΜΑΤΑ.....	127
6.1.1 Σύνοψη Επιτευγμάτων.....	127
6.1.2 Τεχνολογικά Πλεονεκτήματα	128
6.1.3 Προσωπική Εμπειρία	128
6.2 ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ	129
6.2.1 Βραχυπρόθεσμες Επεκτάσεις	129
6.2.2 Αυτόματη Κλιμάκωση Βασισμένη σε Μηχανική Μάθηση	129
6.2.3 Πλήρης Παρατηρησιμότητα (Metrics, Logs, Traces)	130
6.2.4 Ασφάλεια και Διακυβέρνηση.....	130
6.2.5 Βελτιώσεις DevOps για τις Διαχειριζόμενες Εφαρμογές	131
6.3 ΕΠΙΛΟΓΟΣ	131
ΒΙΒΛΙΟΓΡΑΦΙΑ	133
ΠΑΡΑΡΤΗΜΑ Α: DEPLOYMENT YAML ΚΑΙ ΕΝΤΟΛΕΣ ΑΝΑΦΟΡΑΣ	135
A.1 ΠΛΗΡΕΣ DEPLOYMENT.YAML (ΕΦΑΡΜΟΓΗ ΕΠΙΔΕΙΞΗΣ)	135
A.2 ΔΙΑΜΟΡΦΩΣΗ HORIZONTALPODAUTOSCALER	136
A.3 ΕΡΩΤΗΜΑΤΑ PROMQL ΑΝΑΦΟΡΑΣ.....	137
A.4 ΕΝΤΟΛΕΣ ΑΠΟΔΙΑΤΞΗΣ ΚΑΙ ΚΑΘΑΡΙΣΜΟΥ.....	138
ΠΑΡΑΡΤΗΜΑ Β: ΠΑΡΑΔΕΙΓΜΑ ΑΠΟΚΡΙΣΗΣ REST API (HATEOAS)	139
ΠΑΡΑΡΤΗΜΑ Γ: TERRAFORM MODULE — AWS EKS	141
ΠΑΡΑΡΤΗΜΑ Δ: TERRAFORM MODULE — AZURE AKS	144
ΠΑΡΑΡΤΗΜΑ Ε: FLYWAY DATABASE MIGRATIONS	147
V1__create_users_table.sql.....	147
V2__create_applications_tables.sql	147
V3__create_indexes.sql	148
ΠΑΡΑΡΤΗΜΑ ΣΤ: ΠΑΡΑΔΕΙΓΜΑΤΑ ΚΩΔΙΚΑ ΔΟΚΙΜΩΝ ΚΑΙ CI/CD PIPELINE	150
ΣΤ.1 ΣΤΡΑΤΗΓΙΚΗ ΔΟΚΙΜΩΝ	150
Δοκιμή μονάδας — MonitoringService	150
Προτεινόμενη δοκιμή ολοκλήρωσης — ApplicationController (δεν περιλαμβάνεται στην παρούσα έκδοση)	151
ΣΤ.2 CI/CD PIPELINE — GITHUB ACTIONS	151

Κατάλογος Εικόνων

Εικόνα 2.1 — Αρχιτεκτονική Kubernetes: Control Plane και Worker Nodes	28
Εικόνα 2.2 — Τα 6 επίπεδα αποφάσεων στη μοντελοποίηση αρχιτεκτονικής διάταξης Kubernetes	31
Εικόνα 2.3 — Βρόχος ελέγχου (control loop) του Horizontal Pod Autoscaler	34
Εικόνα 2.4 — Αρχιτεκτονική Terraform: Core, Provider Plugins και State Backend	40
Εικόνα 2.5 — Κύκλος εργασίας Terraform (βλ. αναλυτικά Πίνακα 2.9)	41
Εικόνα 4.1 — Διάγραμμα περιβάλλοντος της πλατφόρμας	59
Εικόνα 4.2 — Διάγραμμα συστατικών της πλατφόρμας	61
Εικόνα 4.3 — Διάγραμμα περιπτώσεων χρήσης A: Διαχείριση Χρηστών	64
Εικόνα 4.4 — Διάγραμμα περιπτώσεων χρήσης B: Διαχείριση Εφαρμογών	65
Εικόνα 4.5 — Διάγραμμα ακολουθίας: Εγγραφή Χρήστη	67
Εικόνα 4.6 — Διάγραμμα ακολουθίας: Σύνδεση Χρήστη	68
Εικόνα 4.7 — Διάγραμμα ακολουθίας: Διάταξη Εφαρμογής	69
Εικόνα 4.8 — Διάγραμμα ακολουθίας: Αυτόματη κλιμάκωση βάσει κανόνων QoS	70
Εικόνα 4.9 — Διάγραμμα ακολουθίας: Εμφάνιση εφαρμογής και παρακολούθηση μετρικών	71
Εικόνα 4.10 — Διάγραμμα ακολουθίας: Καταστροφή διάταξης εφαρμογής	72
Εικόνα 4.11 — Διάγραμμα κλάσεων του μοντέλου πεδίου	76
Εικόνα 5.1 — Ροή Εγκατάστασης και Αρχικοποίησης της Πλατφόρμας	105
Εικόνα 5.2 — Αρχιτεκτονική του Περιβάλλοντος Επίδειξης και Αντιστοίχιση με τα Σενάρια	109
Εικόνα 5.3 — Έξοδος των <code>kubectl get nodes</code> και <code>kubectl cluster-info</code>	112
Εικόνα 5.4 — Kubernetes Dashboard: Γραφική Διεπαφή Διαχείρισης Συστάδας	113
Εικόνα 5.5 — Έξοδος του <code>kubectl get pods</code> : τρία Pods <code>demo-webapp</code> σε κατάσταση <code>Running</code>	113
Εικόνα 5.6 — Έξοδος του <code>kubectl get deployments</code> : <code>READY 3/3</code> , <code>UP-TO-DATE 3</code> , <code>AVAILABLE 3</code>	114
Εικόνα 5.7 — Έξοδος του <code>kubectl get services</code> : <code>NodePort 80:32187/TCP</code>	114
Εικόνα 5.8 — Kubernetes Dashboard: τα τρία Pods σε κατάσταση <code>Running</code>	114
Εικόνα 5.9 — Η Εφαρμογή Επίδειξης Προσβάσιμη μέσω NodePort	115
Εικόνα 5.10 — Έξοδος του <code>kubectl get pods -n monitoring</code> : έξι Pods της στοίβας <code>kube-prometheus-stack</code>	115
Εικόνα 5.11 — Grafana v13.0.1: Σελίδα Σύνδεσης	116
Εικόνα 5.12 — Grafana — Kubernetes / Compute Resources / Cluster: CPU 1,70%, Μνήμη 38,2%	117
Εικόνα 5.13 — Grafana — Node (Pods): Αιχμή CPU από τις 10:25	117
Εικόνα 5.14 — Έξοδος του <code>kubectl get hpa</code> : CPU 0%/50%, MINPODS 2, MAXPODS 10, REPLICAS 3	118
Εικόνα 5.15 — Έξοδος του <code>kubectl get hpa -w</code> : μετάβαση CPU 0% → 56% και κλιμάκωση σε 11 λεπτά	119
Εικόνα 5.16 — Grafana: Χρονοσειρά Αξιοποίησης CPU κατά τη Δοκιμή Φόρτου (10:25–11:05)	119
Εικόνα 5.17 — Prometheus — ερώτημα <code>up</code> : 17 endpoints, χρόνος απόκρισης 20 ms	120
Εικόνα 5.18 — Prometheus — <code>kube_pod_status_phase{phase="Running"}</code> : 18 εκτελούμενα Pods, χρόνος απόκρισης 21 ms	121
Εικόνα 5.19 — Alertmanager: Δύο Ενεργές Ομάδες Ειδοποιήσεων	122

Εικόνα 5.20 — <i>Alertmanager: Watchdog (severity = none) και etcdInsufficientMembers (severity = critical)</i>	123
Εικόνα 5.21 — <i>Web UI: Οθόνη Σύνδεσης</i>	124
Εικόνα 5.22 — <i>Web UI: Κατάλογος και Δημιουργία Εφαρμογών</i>	124
Εικόνα ΣΤ.1 — <i>Ροή του Αγωγού Συνεχούς Ενσωμάτωσης και Παράδοσης (GitHub Actions)</i>	152

Κατάλογος Πινάκων

Πίνακας 2.1 — Πλεονεκτήματα Cloud Computing	18
Πίνακας 2.2 — Μοντέλα Παράδοσης Υπηρεσιών (IaaS/PaaS/SaaS).....	19
Πίνακας 2.3 — Υπηρεσίες Load Balancing & Autoscaling ανά Cloud	21
Πίνακας 2.4 — Σύγκριση Εικονικών Μηχανών vs Containers	23
Πίνακας 2.5 — Monolithic vs Microservices Αρχιτεκτονική	25
Πίνακας 2.6 — Βασικά Kubernetes Objects	30
Πίνακας 2.7 — Εργαλεία Τοπικής Ανάπτυξης Kubernetes Συστάδας	33
Πίνακας 2.8 — Σύγκριση Εργαλείων IaC	38
Πίνακας 2.9 — Στάδια Κύκλου Εργασίας Terraform.....	41
Πίνακας 2.10 — Σύγκριση Cloud Providers & Υπηρεσιών	42
Πίνακας 3.1 — Συγκριτικός Πίνακας Σχετικών Εργασιών	48
Πίνακας 4.1 — Είδη Χρηστών της Πλατφόρμας	51
Πίνακας 4.2 — Λειτουργικές Απαιτήσεις Χρηστών (FA-U)	51
Πίνακας 4.3 — Λειτουργικές Απαιτήσεις Εφαρμογών (FA-A)	52
Πίνακας 4.4 — Μη Λειτουργικές Απαιτήσεις	54
Πίνακας 4.5 — Προαιρετικές Απαιτήσεις	57
Πίνακας 4.6 — Κλάσεις του Μοντέλου Πεδίου	73
Πίνακας 4.7 — Τεχνολογική Στοιβά Υλοποίησης.....	77
Πίνακας 4.8 — Σημεία Πρόσβασης (Endpoints) του REST API.....	82
Πίνακας 4.9 — Στρατηγική Δοκιμών ανά Επίπεδο	87
Πίνακας 4.10 — Σύγκριση Μετρηθεισών Τιμών με Στόχους SLA.....	88
Πίνακας 4.11 — Μετρικές Αναφοράς Συστήματος σε Ηρεμία	89
Πίνακας 4.12 — Χρονογραμμή Αυτόματης Κλιμάκωσης HPA υπό Τεχνητό Φόρτο....	90
Πίνακας 4.13 — Επαλήθευση Απαιτήσεων στο Σενάριο Επίδειξης	91
Πίνακας 4.14 — Κλίμακα Αξιολόγησης Απαιτήσεων	92
Πίνακας 4.15 — Ικανοποίηση Λειτουργικών Απαιτήσεων Χρηστών.....	93
Πίνακας 4.16 — Ικανοποίηση Λειτουργικών Απαιτήσεων Εφαρμογών	94
Πίνακας 4.17 — Ικανοποίηση Μη Λειτουργικών Απαιτήσεων: Χρηστικότητα.....	97
Πίνακας 4.18 — Ικανοποίηση Μη Λειτουργικών Απαιτήσεων: Απόδοση	97
Πίνακας 4.19 — Ικανοποίηση Μη Λειτουργικών Απαιτήσεων: Ασφάλεια	98
Πίνακας 4.20 — Ικανοποίηση Μη Λειτουργικών Απαιτήσεων: Αρχιτεκτονική	99
Πίνακας 4.21 — Ικανοποίηση Μη Λειτουργικών Απαιτήσεων: Αξιοπιστία	99
Πίνακας 4.22 — Ικανοποίηση Μη Λειτουργικών Απαιτήσεων: Φορητότητα	100
Πίνακας 4.23 — Ικανοποίηση Μη Λειτουργικών Απαιτήσεων: Κλιμακωσιμότητα ...	100
Πίνακας 4.24 — Ικανοποίηση Προαιρετικών Απαιτήσεων	101
Πίνακας 4.25 — Συνοπτικός Πίνακας Ικανοποίησης Απαιτήσεων	101
Πίνακας 5.1 — Προαπαιτούμενο Λογισμικό Εγκατάστασης (περιβάλλον macOS) ..	103
Πίνακας 5.2 — Κανονική Ονοματολογία Συστατικών της Πλατφόρμας	110
Πίνακας 5.3 — Σημεία Πρόσβασης Υπηρεσιών του Περιβάλλοντος Επίδειξης	112
Πίνακας 5.4 — Αντιστοίχιση Απαιτήσεων και Τεκμηρίων Επίδειξης.....	125
Πίνακας 6.1 — Συγκριτικά Πλεονεκτήματα της Πλατφόρμας έναντι Τυπικών Εναλλακτικών.....	128
Πίνακας A.1 — Ερωτήματα PromQL Αναφοράς.....	137

Κατάλογος Συντομογραφιών

Συντομογραφία	Πλήρης Ορισμός
AKS	Azure Kubernetes Service
API	Application Programming Interface
ASG	Auto Scaling Group (AWS)
CNCF	Cloud Native Computing Foundation
CPU	Central Processing Unit
CR	Custom Resource (Kubernetes)
CRD	Custom Resource Definition
DB	Database
DNS	Domain Name System
EKS	Elastic Kubernetes Service (AWS)
GKE	Google Kubernetes Engine
HCL	HashiCorp Configuration Language
HPA	Horizontal Pod Autoscaler
HTTP	HyperText Transfer Protocol
IaC	Infrastructure as Code
IP	Internet Protocol
JWT	JSON Web Token
K8s	Kubernetes
NIST	National Institute of Standards and Technology
ORM	Object-Relational Mapping
OS	Operating System
PaaS	Platform as a Service
pod	Portable Object Deployer (a K8s unit)
PromQL	Prometheus Query Language
QoS	Quality of Service
RBAC	Role-Based Access Control
REST	Representational State Transfer

Συντομογραφία	Πλήρης Ορισμός
SaaS	Software as a Service
SLA	Service Level Agreement
SPA	Single Page Application
SQL	Structured Query Language
SSL	Secure Sockets Layer
TLS	Transport Layer Security
TSDB	Time Series Database
UI	User Interface
UUID	Universally Unique Identifier
VM	Virtual Machine
VPA	Vertical Pod Autoscaler

Περίληψη

Η παρούσα διπλωματική εργασία παρουσιάζει το σχεδιασμό, την ανάπτυξη και την αξιολόγηση μιας web πλατφόρμας για τη διαχείριση Kubernetes workloads σε multi-cloud (πολυνεφικά) περιβάλλοντα, με επίγνωση ποιότητας υπηρεσίας (QoS-Aware). Η πλατφόρμα αντιμετωπίζει τρία κεντρικά προβλήματα που εμφανίζονται κατά τη διαχείριση σύγχρονων cloud-native εφαρμογών: την έλλειψη ενιαίας ορατότητας σε ετερογενή multi-cloud (πολυνεφικά) περιβάλλοντα, την απουσία υποστήριξης για αυτοματοποιημένη κλιμάκωση βάσει προσωποποιημένων (custom) κανόνων ποιότητας υπηρεσίας οριζόμενων από τον χρήστη, και την πολυπλοκότητα της διάταξης υποδομής, η οποία απαιτεί εξειδικευμένα εργαλεία και διαπιστευτήρια ανά πάροχο.

Η αρχιτεκτονική της πλατφόρμας στηρίζεται σε τρεις πυλώνες: (α) αυτοματοποιημένη διάταξη υποδομής μέσω Terraform (εργαλείο τύπου Infrastructure as Code) σε δύο διαφορετικά νέφη (μέσω των αυτό-διαχειριζόμενων Kubernetes υπηρεσιών τους, AWS EKS και Azure AKS, αντίστοιχα), (β) real-time παρακολούθηση των εφαρμογών που έχουν διαταχθεί στα νέφη αυτά — και ειδικότερα της ποιότητας υπηρεσίας τους (QoS) — μέσω Prometheus, Grafana και Alertmanager, και (γ) QoS-aware αυτο-κλιμάκωση που αξιολογεί τις τηρούμενες μετρικές κάθε 30 δευτερόλεπτα και ενεργοποιεί αυτόματα κλιμάκωση pods ή nodes κατά παραβίαση ορίων (σε προσωποποιημένους κανόνες κλιμάκωσης).

Η πλατφόρμα αναπτύχθηκε με Spring Boot 3.2 (Java 17) για το backend REST API, React.js 18 με TypeScript για το frontend σε μορφή Single Page Application (SPA) (εφαρμογής μοναδικής σελίδας), και PostgreSQL 15 ως βάση δεδομένων. Η ασφάλεια υλοποιήθηκε μέσω JWT authentication, Spring Security RBAC και BCrypt. Η αξιολόγηση της πλατφόρμας πραγματοποιήθηκε μέσω πειραματικής επίδειξης σε τοπικό Minikube (Kubernetes Authors, 2024b) (Kubernetes v1.35.1): επαληθεύθηκε η ορθή λειτουργία της αυτόματης κλιμάκωσης HPA κατά τεχνητή αύξηση του φόρτου CPU πάνω από το όριο του 50%, καθώς και η ορθή λειτουργία του monitoring stack. Επιπλέον, αξιολογήθηκε ο βαθμός ικανοποίησης 47 λειτουργικών και μη λειτουργικών απαιτήσεων της πλατφόρμας, με πλήρη ικανοποίηση 51,1% και μερική 27,7% (των απαιτήσεων της).

Λέξεις-Κλειδιά: Kubernetes, Πολυνεφικό Περιβάλλον, Ποιότητα Υπηρεσίας, αυτό-κλιμάκωση, Prometheus, Grafana, Terraform, Spring Boot, React.js, IaC, HPA, Υπολογιστικό Νέφος, Παρακολούθηση, RBAC, JWT

Abstract

This thesis presents the design, development and evaluation of a web platform for the Quality of Service (QoS) aware management of Kubernetes workloads in multi-cloud environments. The platform addresses three central problems of modern cloud-native applications: the lack of unified observability across multiple cloud providers, the absence of automated scaling based on user-defined custom QoS rules, and the complexity of infrastructure provisioning, which requires provider-specific tooling and credentials.

The platform's architecture rests on three pillars: (a) automated infrastructure provisioning via Terraform (an Infrastructure as Code – IaC tool) in two different clouds (Amazon AWS & Azure Cloud based on their managed Kubernetes Services AWS EKS and Azure AKS, respectively), (b) real-time monitoring through Prometheus, Grafana and Alertmanager, and (c) QoS-aware auto-scaling that evaluates monitored metrics every 30 seconds and automatically scales pods or nodes when the respective scaling rule thresholds are violated.

The platform was developed using Spring Boot 3.2 (Java 17) for the backend REST API, React.js 18 with TypeScript for the frontend SPA (Single Page Application), and PostgreSQL 15 as the database. Security was implemented through JWT authentication and Spring Security RBAC. The platform's demonstration was carried out on a local Minikube cluster (Kubernetes v1.35.1), where automatic HPA scaling was verified upon CPU increase above the 50% threshold. Further, the platform's evaluation covered 47 functional and non-functional requirements, of which 51.1% are fully and 27.7% partially satisfied.

Keywords: Kubernetes, Multi-Cloud, Quality of Service, Auto-Scaling, Prometheus, Grafana, Terraform, Spring Boot, React.js, Infrastructure as Code, HPA, Cloud Computing, Monitoring, RBAC, JWT

Ευχαριστίες

Θα ήθελα να εκφράσω τις ειλικρινείς μου ευχαριστίες στον επιβλέποντα καθηγητή μου για την καθοδήγηση, τις πολύτιμες παρατηρήσεις και την υποστήριξή του καθ' όλη τη διάρκεια εκπόνησης αυτής της διπλωματικής εργασίας.

Ευχαριστώ επίσης τους συμφοιτητές μου στο μεταπτυχιακό αυτό για τις εποικοδομητικές συζητήσεις και τους συνεργάτες μου στον επαγγελματικό μου χώρο για τα σχόλια επί της τεχνικής υλοποίησης της πλατφόρμας.

Ιδιαίτερες ευχαριστίες οφείλω στην οικογένειά μου και στους φίλους μου για την αμέριστη υποστήριξη και κατανόηση κατά τη διάρκεια της εκπόνησης αυτής της εργασίας.

Η παρούσα διπλωματική αφιερώνεται σε όσους πιστεύουν ότι η τεχνολογία πρέπει να είναι προσβάσιμη — όχι μόνο στους ειδικούς.

Κεφάλαιο 1: Εισαγωγή

1.1 Προσδιορισμός του Προβλήματος

Τα τελευταία χρόνια, το υπολογιστικό νέφος (Cloud Computing) και η τεχνολογία δοχείοποίησης (containerization) έχουν μετασηματίσει ριζικά τον τρόπο ανάπτυξης, διάταξης και λειτουργίας σύγχρονων εφαρμογών λογισμικού. Το Kubernetes (K8s) έχει αναδειχθεί ως το de facto σύστημα ενορχήστρωσης δοχείων (containers), ενώ παράλληλα όλο και περισσότερες επιχειρήσεις υιοθετούν πολυνεφικές (multi-cloud) στρατηγικές — αξιοποιώντας ταυτόχρονα υπηρεσίες από πολλαπλούς παρόχους νέφους (AWS, Azure, GCP) για αποφυγή του vendor lock-in (κλειδώματος σε πάροχο) και εκμετάλλευση των ανταγωνιστικών πλεονεκτημάτων κάθε πλατφόρμας (νέφους). Παρόλα αυτά, η διαχείριση εφαρμογών σε τέτοια ετερογενή περιβάλλοντα παρουσιάζει σημαντικές προκλήσεις που δεν αντιμετωπίζονται επαρκώς από τα υπάρχοντα εργαλεία. Το κεντρικό ζήτημα που σχετίζεται με αυτές τις προκλήσεις μπορεί να αναλυθεί σε δύο αλληλένδετους άξονες:

Πρόβλημα 1 — Observability Gap (Κενό Ορατότητας):

Σε ένα multi-cloud Kubernetes περιβάλλον — όπου η ίδια εφαρμογή τρέχει παράλληλα σε AWS EKS και Azure AKS (αυτές είναι διαχειριζόμενες υπηρεσίες Kubernetes των παρόχων AWS και Azure αντίστοιχα) — ο διαχειριστής δεν έχει ενιαία εικόνα της υγείας και της απόδοσης του συστήματος. Κάθε cloud provider διαθέτει τα δικά του native monitoring εργαλεία: AWS CloudWatch, Azure Monitor, Google Cloud Monitoring. Τα εργαλεία αυτά είναι αποτελεσματικά εντός του αντίστοιχου οικοσυστήματος, αλλά δεν επικοινωνούν μεταξύ τους, δημιουργώντας silos πληροφορίας. Ο διαχειριστής αναγκάζεται να ελέγχει πολλαπλές κονσόλες ταυτόχρονα, να συσχετίζει χειρωνακτικά δεδομένα από διαφορετικές πηγές και να λαμβάνει αποφάσεις χωρίς ολοκληρωμένη εικόνα (π.χ. αδυνατεί να διακρίνει άμεσα αν μια πτώση απόδοσης στο AWS οφείλεται σε πρόβλημα του ίδιου του AWS ή σε αυξημένο πραγματικό φόρτο χρηστών, καθυστερώντας κρίσιμες αποφάσεις κλιμάκωσης).

Πρόβλημα 2 — Απουσία QoS-Aware Αυτο-κλιμάκωσης:

Ακόμα και αν κάποιος διαθέτει ενοποιημένη παρακολούθηση, η αντίδραση σε παραβιάσεις ποιότητας υπηρεσίας (π.χ. χρόνος απόκρισης > 500ms, χρήση CPU > 80%, error rate > 1%) παραμένει χειρωνακτική και αργή. Ο HPA (Horizontal Pod Autoscaler) του Kubernetes κλιμακώνει βάσει CPU/memory, αλλά ο ορισμός κανόνων επιπέδου εφαρμογής ποιότητας υπηρεσίας (application-level QoS) από τον χρήστη μέσω γραφικής διεπαφής και η runtime υποστήριξη τους — χωρίς να απαιτείται γνώση kubectl, PromQL ή Kubernetes internals — απουσιάζει εντελώς.

Πρόβλημα 3 — Πολυπλοκότητα Διάταξης:

Η δημιουργία συστάδων Kubernetes (clusters) σε πολλαπλά νέφη απαιτεί βαθιά γνώση ειδικών ως προς το νέφος (cloud-specific) εργαλείων (eksctl, az aks create, gcloud container clusters) και IaC (Infrastructure-as-Code) (Υποδομή ως Κώδικας) πλαισίων (frameworks). Επιπλέον, για κάθε cloud provider υπάρχει διαφορετική διαδικασία, διαφορετικά διαπιστευτήρια (credentials) και διαφορετικές παράμετροι (διαμόρφωσης). Χωρίς αυτοματισμό, η διάταξη είναι χρονοβόρα, επιρρεπής σε σφάλματα (error-prone) και δύσκολα αναπαραξίμη (reproducible).

1.2 Προτεινόμενη Λύση & Στόχοι

Η παρούσα διπλωματική αντιμετωπίζει τις παραπάνω προκλήσεις μέσω της ανάπτυξης μιας ολοκληρωμένης web πλατφόρμας για τη διαχείριση multi-cloud Kubernetes workloads. Η πλατφόρμα θέτει και ικανοποιεί σε υψηλό βαθμό τους ακόλουθους στόχους, οργανωμένους γύρω από έξι βασικά χαρακτηριστικά:

- *Αυτοματοποιημένη Διάταξη Υποδομής (IaC)*: Σχεδίαση και υλοποίηση μηχανισμού όπου ο χρήστης καθορίζει τα χαρακτηριστικά της συστάδας μέσω web φόρμας (πάροχος νέφους, τύπος instance, εύρος κόμβων) και το σύστημα αναλαμβάνει την αυτοματοποιημένη, αναπαράξιμη διάταξη μέσω εργαλείων Infrastructure as Code.
- *Ποιοτικά Προσανατολισμένη Αυτο-κλιμάκωση (QoS-aware)*: Ανάπτυξη μηχανισμού όπου ο χρήστης ορίζει προσαρμοσμένους κανόνες ποιότητας υπηρεσίας (μετρική, τελεστής σύγκρισης, τιμή ορίου, ενέργεια κλιμάκωσης) και το σύστημα αξιολογεί περιοδικά την ικανοποίησή τους, εκτελώντας αυτόματα κλιμάκωση όταν χρειάζεται.
- *Ενοποιημένη Παρακολούθηση (Monitoring)*: Ενσωμάτωση εργαλείων συλλογής μετρικών από όλα τα υποστηριζόμενα νέφη σε ενιαία, οπτικοποιημένη μορφή, καταργώντας το observability gap μεταξύ διαφορετικών παρόχων.
- *Ασφαλές REST API με Έλεγχο Πρόσβασης*: Ανάπτυξη REST API με μηχανισμό αυθεντικοποίησης βάσει tokens και έλεγχο πρόσβασης βάσει ρόλων, ώστε η διαχείριση των πολύ-νεφικών εφαρμογών και των υποδομών τους να είναι ασφαλής και προσβάσιμη μέσω φιλικής διεπαφής χωρίς απαίτηση εξειδικευμένων τεχνικών γνώσεων.
- *Πλήρης Κύκλος Ζωής Εφαρμογών*: Υποστήριξη ολόκληρου του κύκλου ζωής μιας διατεταγμένης εφαρμογής, από τη δημιουργία και διάταξη έως την πλήρη καταστροφή υποδομής προς εξοικονόμηση κόστους.
- *Αξιολόγηση μέσω Πειραματικής Επίδειξης*: Επαλήθευση της λειτουργικότητας και της απόδοσης της πλατφόρμας μέσω πραγματικής εκτέλεσης σε περιβάλλον Kubernetes, με τεκμηριωμένα αποτελέσματα.

Η υιοθέτηση της πλατφόρμας προσφέρει απτά οφέλη σε τρία επίπεδα. Για μεμονωμένους προγραμματιστές/ερευνητές, εξαλείφει την ανάγκη εκμάθησης πολλαπλών cloud-specific εργαλείων (eksctl, az aks), επιτρέποντας διάταξη εφαρμογών μέσω απλής web διεπαφής. Για μικρούς οργανισμούς και startups, το μηδενικό κόστος αδειοδότησης (100% open-source στοίβα) και η αυτόματη κλιμάκωση βάσει πραγματικού φόρτου μειώνουν σημαντικά το λειτουργικό κόστος υποδομής, αποφεύγοντας την υπερτροφοδοσία (over-provisioning). Για μεγαλύτερες εταιρείες, η vendor-agnostic αρχιτεκτονική μειώνει τον κίνδυνο εξάρτησης από συγκεκριμένο πάροχο (vendor lock-in) και επιτρέπει στρατηγικές πολλαπλών νεφών για ανθεκτικότητα και βελτιστοποίηση κόστους.

1.3 Δομή Αναφοράς

Η παρούσα αναφορά οργανώνεται σε έξι κύρια κεφάλαια και έξι παραρτήματα:

- *Κεφάλαιο 2 — Υπόβαθρο*: Παρουσιάζει το θεωρητικό πλαίσιο των τεχνολογιών Cloud Computing (IaaS/PaaS/SaaS, multi-cloud, Docker,

microservices), Kubernetes (αρχιτεκτονική, HPA, monitoring) και Terraform (IaC, HCL, workflow) που αποτελούν τη βάση της πλατφόρμας.

- *Κεφάλαιο 3 — Σχετικές Εργασίες:* Αναλύει πέντε σχετικές ερευνητικές εργασίες (Ben David & Bremner-Barr, 2021· Huang & Pierre, 2023· Rządca et al., 2020· Crossplane Authors, 2024· Tomarchio et al., 2021) και παρουσιάζει συγκριτικό πίνακα δώδεκα κριτηρίων που αναδεικνύει το ερευνητικό κενό το οποίο καλύπτει η παρούσα εργασία.
- *Κεφάλαιο 4 — Ανάπτυξη Συστήματος:* Παρουσιάζει τον πλήρη κύκλο ανάπτυξης της πλατφόρμας με βάση το μοντέλο ανάπτυξης του καταρράκτη (Waterfall): ανάλυση 47 απαιτήσεων, σχεδίαση μέσω UML διαγραμμάτων (Context/Component/Use Case/Sequence Diagrams, Class Diagram), παροχή λεπτομερειών πλήρους υλοποίησης πλατφόρμας, προσδιορισμός δοκιμών και αξιολόγησης της πλατφόρμας, ενώ στο τέλος παρέχεται και αξιολόγηση ικανοποίησης των απαιτήσεων που τέθηκαν.
- *Κεφάλαιο 5 — Εγκατάσταση και Επίδειξη Συστήματος:* Παρέχει αναλυτικό οδηγό εγκατάστασης σε λειτουργικό σύστημα macOS βήμα-βήμα και μια επίδειξη της πλατφόρμας βάσει συγκεκριμένων σεναρίων χρήσης που συνοδεύονται από σχετικά screenshots.
- *Κεφάλαιο 6 — Συμπεράσματα και Μελλοντική Εργασία:* Συνοψίζει τα επιτεύγματα της παρούσας εργασίας, παρουσιάζει οδικό χάρτη (roadmap) δεκαπέντε επεκτάσεων, οργανωμένων σε πέντε θεματικούς άξονες, και εκθέτει τη συνολική συνεισφορά της.

Κεφάλαιο 2: Υπόβαθρο

2.1 Cloud Computing

2.1.1 Ορισμός

Το Cloud Computing (Υπολογιστική Νέφος, ή εναλλακτικά νεφο-υπολογιστική) είναι ένα μοντέλο παροχής υπολογιστικών πόρων — όπως διακομιστές, αποθηκευτικός χώρος, βάσεις δεδομένων, δίκτυα, λογισμικό και analytics — μέσω του διαδικτύου με στόχο την ευέλικτη, γρήγορη και οικονομικά αποδοτική πρόσβαση σε αυτούς. Ο επίσημος ορισμός του NIST (National Institute of Standards and Technology, SP 800-145) ορίζει το Cloud Computing ως:

«Ένα μοντέλο που επιτρέπει την απανταχού, βολική, κατ'απαίτηση δικτυακή πρόσβαση σε ένα κοινό σύνολο ρυθμιζόμενων υπολογιστικών πόρων (π.χ. δίκτυα, διακομιστές, αποθήκευση, εφαρμογές και υπηρεσίες) που μπορούν να παρασχεθούν και να αποδεσμευτούν γρήγορα με ελάχιστη προσπάθεια διαχείρισης ή αλληλεπίδραση με τον πάροχο υπηρεσιών.» — NIST SP 800-145

Στην πράξη αυτό σημαίνει ότι μια εταιρεία δεν χρειάζεται να αγοράσει και να συντηρεί τη δική της φυσική υποδομή. Μπορεί να «νοικιάζει» πόρους από παρόχους, όπως AWS, Azure ή GCP, να τους αποκτά άμεσα (σε δευτερόλεπτα ή λεπτά) και να πληρώνει μόνο για ό,τι καταναλώνει (pay-per-use μοντέλο).

2.1.2 Τα 5 Βασικά Χαρακτηριστικά Cloud Computing

Ο ορισμός του NIST (Mell & Grance, 2011) προσδιορίζει πέντε βασικά χαρακτηριστικά που διακρίνουν το Cloud Computing από τα παραδοσιακά μοντέλα:

1. On-Demand Self-Service (Αυτεξυπηρέτηση κατ' απαίτηση)

Ο χρήστης μπορεί να προμηθευτεί πόρους αυτόματα όποτε τους χρειάζεται, χωρίς να απαιτείται ανθρώπινη παρέμβαση από τον πάροχο. Για παράδειγμα, ένας developer μπορεί να δημιουργήσει νέα συστάδα Kubernetes στο νέφος AWS με μερικά κλικ, εντολές CLI ή γραμμές κώδικα — χωρίς να χρειαστεί να επικοινωνήσει με κάποιον εργαζόμενο της Amazon (δηλαδή του παρόχου του AWS νέφους).

2. Broad Network Access (Ευρεία πρόσβαση δικτύου)

Οι πόροι είναι προσβάσιμοι μέσω του διαδικτύου από οποιαδήποτε συσκευή — laptop, tablet, smartphone — χρησιμοποιώντας τυπικά (δικτυακά) πρωτόκολλα. Αυτό επιτρέπει κατανεμημένες ομάδες ανάπτυξης να εργάζονται με κοινή υποδομή ανεξαρτήτως τοποθεσίας.

3. Resource Pooling (Συγκέντρωση πόρων)

Ο πάροχος συγκεντρώνει υπολογιστικούς πόρους και τους διανέμει δυναμικά μεταξύ πολλών πελατών με βάση το μοντέλο πολλαπλής μίσθωσης (multi-tenant), όπου οι ίδιοι φυσικοί πόροι μπορεί να διαμοιράζονται μεταξύ πολλαπλών πελατών. Ο χρήστης συνήθως δεν γνωρίζει την ακριβή φυσική τοποθεσία των πόρων του, αν και μπορεί να καθορίσει συγκεκριμένη γεωγραφική περιοχή (region).

4. Rapid Elasticity (Γρήγορη ελαστικότητα)

Οι πόροι μπορούν να αυξηθούν ή να μειωθούν γρήγορα, ακόμα και αυτόματα, ανάλογα με τη ζήτηση. Αυτό είναι ιδιαίτερα κρίσιμο για εφαρμογές με ανομοιόμορφο φόρτο εργασίας: ένα e-commerce site μπορεί να χρειάζεται 5 servers κανονικά και

50 στη Black Friday (δηλαδή μια ιδιαίτερη ημέρα με σημαντικά μεγαλύτερο όγκο επισκέψεων & πωλήσεων).

5. Measured Service (Μετρούμενη υπηρεσία)

Η χρήση πόρων παρακολουθείται, ελέγχεται και αναφέρεται με διαφάνεια. Ο χρήστης πληρώνει ακριβώς για ό,τι καταναλώνει, με πλήρη ορατότητα του κόστους ανά ώρα, ανά request ή ανά GB.

2.1.3 Πλεονεκτήματα Cloud Computing

Ο Πίνακας 2.1 συνοψίζει τα βασικά πλεονεκτήματα του Cloud Computing όπως αναλύθηκαν παραπάνω.

Πίνακας 2.1 — Πλεονεκτήματα Cloud Computing

Πλεονέκτημα	Ανάλυση
Μείωση κόστους	Εξάλειψη κόστους αγοράς & συντήρησης φυσικής υποδομής. Μετατροπή CAPEX (κεφαλαιουχικές δαπάνες) σε OPEX (λειτουργικά έξοδα). Πληρωμή μόνο για ό,τι χρησιμοποιείται.
Ευελιξία & Κλιμάκωση	Άμεση προσαρμογή πόρων στις ανάγκες της εφαρμογής — από 1 server σε 1.000 μέσα σε λεπτά, χωρίς προαγορά υλικού (hardware).
Υψηλή Διαθεσιμότητα	Οι πάροχοι νέφους εγγυώνται SLAs με > 99,9% uptime. Επιπλέον, παρέχεται ενσωματωμένη αντιμετώπιση σφαλμάτων μέσω geo-redundant datacenters (γεο-πλεοναζόντων κέντρων δεδομένων).
Γεωγραφική Κατανομή	Ανάπτυξη εφαρμογών σε περιχές (regions) κοντά στους χρήστες για μείωση καθυστέρησης (latency). Συγκριτικά, η AWS έχει 33 regions, η Azure 60+ και η GCP 40+.
Ασφάλεια	Οι μεγάλοι πάροχοι επενδύουν δισεκατομμύρια στην υποστήριξη της ασφάλειας που ξεπερνά αυτό που μπορεί να διαθέσει ο μέσος οργανισμός.
Ταχύτητα Ανάπτυξης	Από ιδέα σε παραγωγή σε ώρες αντί για εβδομάδες. Δεν χρειάζεται προμήθεια, εγκατάσταση και ρύθμιση υλικού.
Ανάνηψη από Καταστροφές	Εύκολη υλοποίηση backup και DR (disaster recovery) (ανάνηψη από καταστροφές) σε πολλαπλές γεωγραφικές τοποθεσίες χωρίς επιπλέον κέντρα δεδομένων.

2.1.4 Μοντέλα Παράδοσης Υπηρεσιών

Τα τρία θεμελιώδη μοντέλα παράδοσης υπηρεσιών νέφους, όπως ορίζονται από το NIST (Mell & Grance, 2011), διαφέρουν ως προς τον βαθμό ελέγχου και ευθύνης που κατέχει ο χρήστης:

IaaS — Infrastructure as a Service (Υποδομή ως Υπηρεσία):

Ο πάροχος παρέχει τη βασική υποδομή: εικονικές μηχανές (virtual machines), δίκτυα και αποθηκευτικό χώρο. Ο χρήστης διαχειρίζεται το λειτουργικό σύστημα (operating system – OS), τις εφαρμογές και τα δεδομένα. Παραδείγματα: AWS EC2, Azure Virtual Machines, Google Compute Engine. Αυτό το μοντέλο χρησιμοποιείται στην παρούσα εργασία για τη δημιουργία των Kubernetes worker nodes (κόμβων εργασίας).

Βασικό πλεονέκτημα: μέγιστη ευελιξία και έλεγχος στη διαμόρφωση του συστήματος, ιδανικό όταν απαιτείται πλήρης προσαρμογή του λειτουργικού/runtime — όπως ακριβώς στην περίπτωση κόμβων Kubernetes.

PaaS — Platform as a Service (Πλατφόρμα ως Υπηρεσία):

Ο πάροχος παρέχει επιπλέον πλήρες περιβάλλον ανάπτυξης: runtime, μεσισμικό (middleware), βάσεις δεδομένων (databases) και εργαλεία DevOps. Ο χρήστης αναπτύσσει μόνο τον κώδικα μέσω αυτού του προσφερόμενου περιβάλλοντος. Παραδείγματα: Google App Engine, AWS Elastic Beanstalk, Heroku, Azure App Service.

Βασικό πλεονέκτημα: ταχύτερος κύκλος ανάπτυξης, καθώς ο προγραμματιστής απαλλάσσεται από τη διαχείριση υποδομής/runtime και επικεντρώνεται αποκλειστικά στη λογική της εφαρμογής.

SaaS — Software as a Service:

Ο πάροχος παρέχει πλήρη έτοιμη εφαρμογή. Ο χρήστης δεν διαχειρίζεται τίποτα από την υποδομή και απλώς προσπελάζει την εφαρμογή μέσω φυλλομετρητών ή διαθέσιμων APIs. Παραδείγματα: Gmail, Salesforce, Microsoft 365, Zoom. Η παρούσα πλατφόρμα, αν προσφερθεί ως διαχειρίσιμη υπηρεσία (managed service), θα εμπίπτει σε αυτή την κατηγορία/μοντέλο.

Βασικό πλεονέκτημα: μηδενική ανάγκη τεχνικής διαχείρισης από τον τελικό χρήστη — άμεση χρήση χωρίς εγκατάσταση, ιδανικό για μη τεχνικό κοινό.

Ο Πίνακας 2.2 συγκεντρώνει συγκριτικά τα τρία μοντέλα παράδοσης υπηρεσιών νέφους (IaaS/PaaS/SaaS).

Πίνακας 2.2 — Μοντέλα Παράδοσης Υπηρεσιών (IaaS/PaaS/SaaS)

Μοντέλο	Τι παρέχει ο Provider	Τι διαχειρίζεται ο Χρήστης	Παραδείγματα
IaaS	Servers, Storage, Networking, Virtualization	OS, Runtime, Middleware, Apps, Data	AWS EC2, Azure VMs, Google CE
PaaS	IaaS + OS + Runtime + Middleware	Applications, Data	Heroku, App Engine, Elastic Beanstalk
SaaS	Τα πάντα	Ρυθμίσεις & δεδομένα χρήστη	Gmail, Salesforce, MS365

2.1.5 Μοντέλα Ανάπτυξης

Ένα μοντέλο ανάπτυξης (deployment model) καθορίζει ποιος κατέχει, διαχειρίζεται και έχει πρόσβαση στην υπολογιστική υποδομή (Mell & Grance, 2011). Τα μοντέλα

διαφοροποιούνται κυρίως ως προς: (α) την ιδιοκτησία της υποδομής, (β) το κόστος και την πολυπλοκότητα διαχείρισης, και (γ) το επίπεδο ελέγχου και ασφάλειας.

Public Cloud (Δημόσιο Νέφος):

Η υποδομή ανήκει και διαχειρίζεται από τρίτο πάροχο (AWS, Azure, GCP), προσβάσιμη από οποιονδήποτε πελάτη μέσω διαδικτύου, με διαμοιρασμό φυσικών πόρων μεταξύ πολλαπλών οργανισμών (multi-tenancy).

Πλεονεκτήματα: μηδενικό αρχικό κόστος υποδομής, απεριόριστη οριζόντια κλιμάκωση, pay-as-you-go χρέωση. Συγκριτικά με την απόκτηση και συντήρηση ιδιόκτητης υποδομής (CAPEX σε servers, datacenter, προσωπικό), το public cloud προσφέρει σημαντικά χαμηλότερο αρχικό κόστος και μηδενική ανάγκη συντήρησης hardware εκ μέρους του πελάτη.

Μειονεκτήματα: μειωμένος έλεγχος φυσικής υποδομής, πιθανές ανησυχίες συμμόρφωσης για ευαίσθητα δεδομένα.

Private Cloud (Ιδιωτικό Νέφος):

Η υποδομή λειτουργεί αποκλειστικά για έναν οργανισμό, είτε on-premises είτε σε dedicated datacenter τρίτου.

Πλεονεκτήματα: πλήρης έλεγχος ασφάλειας και συμμόρφωσης, προβλέψιμη απόδοση.

Μειονεκτήματα: υψηλό αρχικό κόστος επένδυσης (CAPEX), απαιτεί εξειδικευμένο προσωπικό διαχείρισης, περιορισμένη ελαστικότητα κλιμάκωσης.

Community Cloud (Κοινοτικό Νέφος):

Η υποδομή διαμοιράζεται από πολλαπλούς οργανισμούς με κοινά ενδιαφέροντα (π.χ. κανονιστική συμμόρφωση, κλάδος δραστηριότητας), όπως τράπεζες ή κυβερνητικοί φορείς.

Πλεονεκτήματα: επιμερισμός κόστους μεταξύ οργανισμών, εξειδικευμένη συμμόρφωση με κλαδικές απαιτήσεις.

Μειονεκτήματα: περιορισμένη διαθεσιμότητα παρόχων, λιγότερη ευελιξία σε σχέση με public cloud.

Hybrid Cloud (Υβριδικό Νέφος):

Συνδυασμός public και private cloud με ελεγχόμενη διασύνδεση, όπου ευαίσθητα δεδομένα παραμένουν σε private υποδομή ενώ λιγότερο κρίσιμα workloads εκτελούνται σε public cloud.

Πλεονεκτήματα: ισορροπία κόστους και ασφάλειας, δυνατότητα cloud bursting σε αιχμές φόρτου.

Μειονεκτήματα: αυξημένη πολυπλοκότητα διαχείρισης δικτύου και συγχρονισμού δεδομένων μεταξύ περιβαλλόντων.

Multi-Cloud (Πολλαπλό Νέφος):

Χρήση υπηρεσιών από πολλαπλούς public cloud providers ταυτόχρονα (π.χ. AWS και Azure), όπου διαφορετικά νέφη μπορεί να ανήκουν στο ίδιο ή διαφορετικό μοντέλο ανάπτυξης.

Πλεονεκτήματα: αποφυγή vendor lock-in, δυνατότητα επιλογής βέλτιστης υπηρεσίας ανά πάροχο (best-of-breed), γεωγραφικός πλεονασμός (redundancy).

Μειονεκτήματα: αυξημένη πολυπλοκότητα διαχείρισης, ανάγκη ενοποιημένου monitoring — πρόβλημα που αντιμετωπίζει άμεσα η παρούσα πλατφόρμα.

Σχετίζεται με την έννοια του cross-cloud, όπου ένα και μόνο στιγμιότυπο εφαρμογής μπορεί να διαμοιράζει τη λειτουργία του σε πολλαπλά νέφη ταυτόχρονα.

2.1.6 Υπηρεσίες Εξισορρόπησης Φορτίου & Αυτο-κλιμάκωσης

Load Balancing (Εξισορρόπηση Φορτίου) ¹:

Ο load balancer κατανέμει εισερχόμενα αιτήματα μεταξύ πολλαπλών instances μιας εφαρμογής, αποτρέποντας την υπερφόρτωση ενός μόνο instance. Λειτουργεί σε Layer 4 (TCP/UDP) ή Layer 7 (HTTP). Ανά cloud provider:

- **AWS:** Elastic Load Balancer (ALB — Application Layer 7, NLB — Network Layer 4, CLB — Classic Load Balancer, μπορεί να καλύψει το επίπεδο εφαρμογής ή δικτύου ανάλογα την ρύθμιση αλλά δεν συνιστάται πλέον από την AWS για νέες υλοποιήσεις)
- **Azure:** Azure Load Balancer (Layer 4) και Azure Application Gateway (Layer 7 με WAF (Web Application Firewall — τείχος προστασίας για web εφαρμογές))
- **GCP:** Cloud Load Balancing — υποστηρίζει global και local/external load balancers, καθώς και passthrough/proxy LBs (και οι δύο υποστηρίζουν anycast). Η επιλογή γίνεται βάσει τριών κριτηρίων: reachability (global/regional), traffic type (HTTP(S)/TCP/UDP) και LB type. Το επίπεδο (Layer 4 ή Layer 7) είναι configurable ανά load balancer, όχι ταυτόχρονο και στα δύο.

Auto Scaling (Αυτο-κλιμάκωση):

Μηχανισμός που προσαρμόζει αυτόματα τον αριθμό των υπολογιστικών πόρων βάσει μετρικών (CPU, memory, custom metrics). Διακρίνεται σε δύο κατευθύνσεις:

- **Horizontal Scaling (Οριζόντια Κλιμάκωση) (Scale Out/In):** Προσθήκη/αφαίρεση στιγμιότυπων εικονικών μηχανών. Συνήθως χωρίς downtime (εκτός και αν προσωρινά πάμε σε 0 στιγμιότυπα ή η τρέχουσα χωρητικότητα είναι ανεπαρκής για τον τρέχων φόρτο εργασίας). Παράδειγμα: από 2 σε 10 VMs κατά αυξημένο φόρτο.
- **Vertical Scaling (Κάθετη Κλιμάκωση) (Scale Up/Down):** Αύξηση/μείωση πόρων ενός στιγμιότυπων εικονικής μηχανής (CPU cores, RAM). Συνήθως απαιτεί restart (επανεκκίνηση οπότε υπάρχει σίγουρα downtime). Παράδειγμα: από t3.medium σε t3.xlarge.

Ο Πίνακας 2.3 παρουσιάζει συγκριτικά τις υπηρεσίες load balancing και autoscaling ανά πάροχο νέφους.

Πίνακας 2.3 — Υπηρεσίες Load Balancing & Autoscaling ανά Cloud

Cloud Provider	Υπηρεσία Auto Scaling	Trigger Μετρικές
AWS	Auto Scaling Groups (ASG) + Target Tracking	CPU%, memory, custom CloudWatch metrics, SQS queue depth
Azure	Virtual Machine Scale Sets (VMSS)	CPU%, memory, Azure Monitor custom metrics, Service Bus

¹ Σημείωση: η ανάλυση εδώ αφορά υπηρεσίες load balancing και autoscaling σε επίπεδο υποδομής/VMs που παρέχει κάθε cloud provider. Η αντίστοιχη κάλυψη σε επίπεδο Kubernetes (Services, Ingress, HPA) παρουσιάζεται αναλυτικά στην Ενότητα 2.2.

Cloud Provider	Υπηρεσία Auto Scaling	Trigger Μετρικές
GCP	Managed Instance Groups (MIG) + Autoscaler	CPU%, custom Stackdriver metrics, Cloud Monitoring + Pub/Sub
Kubernetes (any-cloud — υποστηρίζεται με τον ίδιο τρόπο σε οποιοδήποτε cloud provider, ανεξάρτητα λειτουργώντας σε καθένα)	HPA (pods), VPA (pods), Cluster Autoscaler/Karpenter (nodes), KEDA (pods)	CPU%, memory, custom Prometheus metrics

2.1.7 Υπηρεσίες Παρακολούθησης

Κάθε πάροχος νέφους διαθέτει εγγενείς (native) υπηρεσίες παρακολούθησης (Amazon Web Services, 2024b· Microsoft, 2024a):

- *AWS CloudWatch*: Συλλογή μετρικών, logs, alarms και dashboards για AWS πόρους. Υποστηρίζει custom metrics μέσω API, composite alarms και ML-based anomaly detection. Το ML-based anomaly detection βασίζεται σε ιστορικά δεδομένα του CloudWatch και δεν απαιτεί ρητή διαμόρφωση από τον χρήστη. Δεν παρέχεται όμως εγγενής υποστήριξη για queue-based scaling (κλιμάκωση βάσει μήκους ουράς μηνυμάτων) στο βασικό CloudWatch — αυτό απαιτεί επιπλέον ρύθμιση μέσω custom metrics.
- *Azure Monitor*: Ενοποιημένη πλατφόρμα παρακολούθησης. Περιλαμβάνει Log Analytics workspace, Application Insights (APM), και Azure Alerts με action groups. Για queue-based scaling, το Azure Monitor συνδυάζεται με το Azure Service Bus, παρέχοντας μετρικές μήκους ουράς ως trigger για autoscaling rules.
- *GCP Cloud Monitoring (Stackdriver)*: Metrics Explorer, Alerting policies, Dashboards. Πλήρης ενσωμάτωση με Cloud Logging και Cloud Trace. Υποστηρίζει custom metrics μέσω του Cloud Monitoring API, καθώς και queue-based scaling σε συνδυασμό με το Cloud Tasks/Pub/Sub, χρησιμοποιώντας το μήκος ουράς ως μετρική autoscaling.

Πέρα από τα native εργαλεία, υπάρχουν open-source cross/multi-cloud λύσεις που αποτελούν τη βάση της παρούσας πλατφόρμας:

- *Prometheus*: Pull-based σύστημα συλλογής μετρικών και time-series βάση δεδομένων. Χρησιμοποιεί τη γλώσσα PromQL (Prometheus Query Language) για τη διενέργεια queries (επερωτήσεων) — απαραίτητες ώστε να ανακτηθούν συγκεκριμένες χρονοσειρές μετρικών (π.χ. μέσος όρος CPU τελευταίου 5λέπτου) από την time-series βάση δεδομένων (time-series database) για αξιολόγηση έναντι των QoS κανόνων. Ο κύριος μηχανισμός παρακολούθησης της παρούσας διπλωματικής.

- *Grafana*: Πλατφόρμα οπτικοποίησης που συνδέεται με Prometheus. Δημιουργεί interactive dashboards με charts, gauges και alert panels.
- *Alertmanager*: Διαχείριση ειδοποιήσεων βάσει κανόνων. Δρομολογεί alerts σε email, Slack, PagerDuty, webhook κτλ. Το σύνολο του open-source stack (Prometheus/Grafana/Alertmanager) υποστηρίζει πλήρως custom μετρικές: οποιαδήποτε εφαρμογή που εκθέτει μετρικές σε μορφή συμβατή με Prometheus (Prometheus exposition format) μπορεί να παρακολουθηθεί, χωρίς περιορισμό σε προκαθορισμένο σύνολο μετρικών.

2.1.8 Εικονικοποίηση: VMs vs Containers

Η εικονικοποίηση αποτελεί την τεχνολογική βάση του υπολογιστικού νέφους. Τα δύο κύρια παραδείγματα (paradigms) είναι (Docker Inc., 2024):

Hardware Virtualization (Εικονικοποίηση Υλικού) (Hypervisor-based VMs):

Ένας hypervisor (υπερεπόπτης) (VMware vSphere, KVM, Microsoft Hyper-V) τρέχει πάνω στο υλικό και δημιουργεί πολλαπλές εικονικές μηχανές (Virtual Machines). Κάθε VM έχει το δικό της πλήρες OS. Παρέχει υψηλή απομόνωση αλλά με σημαντικό overhead (GB per VM, λεπτά εκκίνησης).

OS-level Virtualization (Εικονικοποίηση Επιπέδου Λειτουργικού Συστήματος) (Containers):

Τα containers μοιράζονται τον kernel (πυρήνα) του OS του ξενιστή. Κάθε container απομονώνεται μέσω Linux namespaces και cgroups. Αποτέλεσμα: πολύ ελαφρύτερα (MBs), γρήγορη εκκίνηση (δευτερόλεπτα) και χαμηλό overhead. Η τεχνολογία Docker είναι η πιο διαδεδομένη υλοποίηση αυτού του παραδείγματος εικονικοποίησης. Ο Πίνακας 2.4 συνοψίζει τις βασικές διαφορές μεταξύ εικονικών μηχανών και containers.

Πίνακας 2.4 — Σύγκριση Εικονικών Μηχανών vs Containers

Κριτήριο	Virtual Machines	Containers (Docker)
Αρχιτεκτονική	Πλήρες OS ανά instance (Guest OS + Hypervisor)	Shared kernel, απομονωμένος χώρος χρηστών (isolated user space) (namespaces/cgroups)
Μέγεθος	GBs (OS + Libraries + App)	MBs (Libraries + App μόνο)
Χρόνος Εκκίνησης	1-5 λεπτά	Δευτερόλεπτα έως ms
Απομόνωση	Πλήρης (hardware-level isolation)	Μερική (kernel shared, process isolated)
Portability	Μέτρια — εξαρτάται από hypervisor	Υψηλή — «build once, run anywhere»
Resource Overhead	Υψηλό — Guest OS καταναλώνει CPU/RAM	Χαμηλό — ~10x αποδοτικότερο

Κριτήριο	Virtual Machines	Containers (Docker)
Χρήση στη Διπλ.	Kubernetes Worker Nodes (EKS/AKS VMs)	Εφαρμογές ως Docker containers

2.1.9 Docker — Η Τεχνολογία Containerization

Το Docker (Docker Inc., 2024), το οποίο κυκλοφόρησε το 2013 από την dotCloud/Docker Inc., παρουσιάστηκε με σκοπό την ανάπτυξη εφαρμογών εισάγοντας το container image (εικόνα δοχείων) — ένα αμετάβλητο, φορητό πακέτο (immutable, portable package) που περιλαμβάνει εφαρμογή, εξαρτήσεις (dependencies) και runtime διαμόρφωση (configuration), αλλά χωρίς το βάρος ολόκληρου OS.

Βασικές έννοιες Docker:

- **Dockerfile:** Αρχείο κειμένου που περιέχει βήμα-βήμα οδηγίες για δημιουργία image (FROM, RUN, COPY, EXPOSE, CMD).
- **Image:** Immutable, layer-based package. Δημιουργείται από Dockerfile. Ένα image = η «φωτογραφία» της εφαρμογής ή ενός συστατικού μιας εφαρμογής.
- **Container:** εκτελούμενο στιγμιοτύπο (running instance) ενός image. Μπορούν να εκτελούνται πολλά containers από το ίδιο image ταυτόχρονα (σενάριο κλιμάκωσης εφαρμογής ή συστατικού της).
- **Docker Hub / Registry:** Κεντρικό αποθετήριο images. Δημόσια (Docker Hub) ή ιδιωτικά (AWS ECR, Azure ACR, GCP Artifact Registry).
- **Docker Compose:** Εργαλείο ορισμού και εκτέλεσης multi-container εφαρμογών μέσω docker-compose.yml.

Παράδειγμα Dockerfile για Spring Boot εφαρμογή:

```
FROM openjdk:17-slim
WORKDIR /app
COPY target/cloud-platform-0.0.1-SNAPSHOT.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Επεξήγηση: FROM ορίζει την εικόνα βάσης (base image) (ελαφριά έκδοση JRE 17). WORKDIR καθορίζει τον φάκελο εργασίας εντός του container. COPY αντιγράφει το μεταγλωττισμένο .jar αρχείο μέσα στο image. EXPOSE δηλώνει την δικτυακή θύρα (8080) στην οποία ακούει η εφαρμογή. ENTRYPOINT ορίζει την εντολή εκκίνησης όταν το container τρέξει.

Τα κυρίαρχα πλεονεκτήματα που καθιστούν το Docker απαραίτητο στο Kubernetes ecosystem:

- **«Build once, run anywhere»:** Το ίδιο container image τρέχει αναλλοίωτο σε laptop, staging server, AWS, Azure ή GCP.
- **Αναπαραγωγή περιβάλλοντα:** Εξάλειψη του «works on my machine» — όλα τα είδη πλατφόρμας/μηχανήματος τρέχουν ακριβώς το ίδιο περιβάλλον.
- **Γρήγορη ανάπτυξη και κλιμάκωση:** Εκκίνηση νέου container σε δευτερόλεπτα — ιδανικό για horizontal scaling.

- *Layered builds*: Κοινά layers (base OS, JDK) επιτρέπουν την επιτάχυνση της κατασκευής των εικόνων λόγω της χρήσης κρυφής μνήμης. Ουσιαστικά, μόνο οι αλλαγές οδηγούν σε εν μέρει ανακατασκευή εικόνας — αυτό οδηγεί σε γρήγορη εκτέλεση CI/CD αγωγών (pipelines) για τον τάχιστο έλεγχο και διάταξη αλλαγών σε περιβάλλοντα ανάπτυξης, ελέγχου ή παραγωγής.

2.1.10 Microservices & Cloud-Native Εφαρμογές

Monolithic Architecture (Μονολιθική Αρχιτεκτονική):

Όλες οι λειτουργίες (UI, business logic, data access) βρίσκονται σε ένα ενιαίο εκτελέσιμο. Απλή στην ανάπτυξη αρχικά, αλλά χρονοβόρα και κοστοβόρα στην κλιμάκωση (κλιμακώνεται ολόκληρη η εφαρμογή, όχι τμήματά της), δύσκολη και πολύπλοκη στη συντήρηση (ένα bug μπορεί να ρίξει ολόκληρη την εφαρμογή) και στην ανάπτυξη σε μεγάλες ομάδες (code conflicts/συγκρούσεις υποβολών κώδικα, μεγάλα πακέτα διάταξης/deployment packages).

Microservices Architecture:

Η εφαρμογή χωρίζεται σε μικρές, ανεξάρτητες υπηρεσίες που επικοινωνούν μεταξύ τους μέσω APIs/ελαφρών πρωτοκόλλων (REST, gRPC) ή message queues (Kafka, RabbitMQ). Κάθε υπηρεσία αναπτύσσεται ανεξάρτητα (ανεξάρτητα CI/CD pipelines), κλιμακώνεται ανεξάρτητα (π.χ. μόνο το authentication service κλιμακώνεται κατά peak load) και μπορεί να χρησιμοποιεί διαφορετική γλώσσα/τεχνολογία ανάπτυξης (polyglot architecture).

Cloud-Native Εφαρμογές:

Εφαρμογές σχεδιασμένες εξ αρχής για λειτουργία στο νέφος, ακολουθώντας τα 12-Factor App αρχές (principles) (Wiggins, 2011): ρυθμίσεις μέσω environment variables, stateless processes, dependency isolation, logging σε stdout, scalable processes. Χαρακτηρίζονται από containerization, microservices, dynamic orchestration (Kubernetes) και CI/CD automation. Η παρούσα πλατφόρμα ακολουθεί αυτές τις αρχές.

Ο Πίνακας 2.5 αντιπαραβάλλει τη μονολιθική με τη microservices αρχιτεκτονική ως προς τα κρίσιμα χαρακτηριστικά τους.

Πίνακας 2.5 — *Monolithic vs Microservices Αρχιτεκτονική*

Χαρακτηριστικό	Monolithic	Microservices
Deployment	Ένα ενιαίο package	Ανεξάρτητα deployments ανά υπηρεσία
Κλιμάκωση	Ολόκληρη εφαρμογή	Μόνο η υπηρεσία που την χρειάζεται
Fault Isolation	Ένα bug = πτώση ολόκληρης εφαρμογής	Αποτυχία service χωρίς cascade σε όλη την εφαρμογή
Τεχνολογίες	Ενιαία — όλη η ομάδα χρησιμοποιεί την ίδια στοίβα ανάπτυξης	Polyglot — κάθε service εφαρμόζει τις δικές του στοίβες ανάπτυξης

Χαρακτηριστικό	Monolithic	Microservices
Kubernetes fit	Μέτριο — ένα μεγάλο Deployment ²	Άριστο — ένα Deployment ανά stateless service + StatefulSet για τη βάση δεδομένων + Deployment για το UI

² Το Deployment αφορά stateless φόρτους εργασίας οπότε μπορεί να καλύψει το επίπεδο frontend & backend μιας εφαρμογής. Επιπρόσθετα, όσον αφορά το επίπεδο υποδομής/βάσης δεδομένων, μπορούμε είτε να έχουμε ένα StatefulSet (καταστατικός φόρτος εργασίας) είτε εφαρμογή και διαχείριση βάσης εκτός Kubernetes μέσω διαχειριζόμενης βάσης δεδομένων (managed DB) (σύγχρονη έκδοση διάταξης μονολιθικής εφαρμογής).

2.2 Kubernetes

2.2.1 Ορισμός & Ιστορικό

Το Kubernetes (K8s — από το ελληνικό «κυβερνήτης») είναι ένα open-source σύστημα αυτόματης ενορχήστρωσης, ανάπτυξης και κλιμάκωσης containerized εφαρμογών. Αναπτύχθηκε αρχικά από την Google (Burns et al., 2016· Kubernetes Authors, 2024b) ως εξέλιξη των εσωτερικών της συστημάτων διαχείρισης συστάδων, Borg και Omega, τα οποία διαχειρίζονταν δισεκατομμύρια containers στην υποδομή της Google. Το Kubernetes δημοσιεύθηκε ως open-source το 2014 και, με αφορμή την κυκλοφορία της έκδοσης 1.0, παραχωρήθηκε το 2015 στο νεοσύστατο Cloud Native Computing Foundation (CNCF, <https://www.cncf.io>) ως το πρώτο του project. Σήμερα αποτελεί project³ του Cloud Native Computing Foundation (CNCF) και αποτελεί de facto πρότυπο για container orchestration.

«Μία φορητή, επεκτάσιμη, ανοικτού κώδικα πλατφόρμα για τη διαχείριση δοχειοποιημένων φόρτων εργασίας (containerized workloads) και υπηρεσιών, που διευκολύνει τόσο τη δηλωτική ρύθμιση (declarative configuration) όσο και τον αυτοματισμό.» — Kubernetes.io (Πηγή:

<https://kubernetes.io/docs/concepts/overview>)

Στην πράξη, το Kubernetes απαντά στο ερώτημα: «Πού και πώς θα τρέξουν τα containers μου σε παραγωγικό περιβάλλον;» — οπότε διαχειρίζεται αυτόματα την τοποθέτηση, κλιμάκωση, ανάκαμψη από αποτυχίες και δικτύωση των containers.

2.2.2 Πλεονεκτήματα Kubernetes

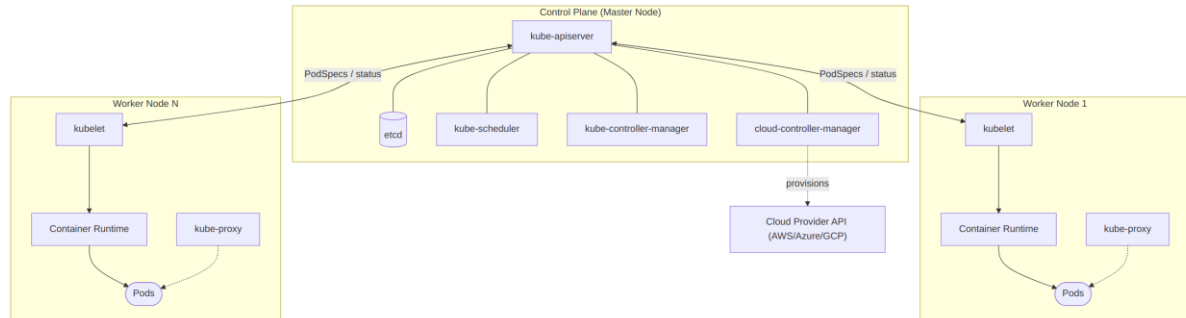
- *Αυτόματη Τοποθέτηση (Scheduling)*: Ο K8s scheduler επιλέγει αυτόματα ποιον κόμβο (node) να χρησιμοποιήσει για κάθε Pod, βάσει resource requests (αιτημάτων πόρων), affinity rules (κανόνων συντοποθεσίας) και διαθεσιμότητας — χωρίς χειρωνακτική παρέμβαση.
- *Αυτό-επούλωση (Self-Healing)*: Αυτόματη επανεκκίνηση αποτυχημένων containers, αντικατάσταση και αναδιαμόρφωσή τους. Δεν εκθέτει containers που δεν περνούν health checks (ελέγχους υγείας) σε traffic (κυκλοφορία) μέχρι να είναι έτοιμα.
- *Οριζόντια Κλιμάκωση (Horizontal Scaling)*: Αυτόματη κλιμάκωση replicas (αντίγραφα/στιγμιότυπα μιας εφαρμογής που εκτελούνται ταυτόχρονα) βάσει CPU, memory ή custom metrics μέσω HPA. Χειρωνακτικά ή αυτόματα.
- *Εξισορρόπηση Φορτίου & Ανακάλυψη Υπηρεσιών (Load Balancing & Service Discovery)*: Εκχώρηση IP διευθύνσεων και DNS ονομάτων σε containers. Εξισορρόπηση φορτίου μεταξύ pods.
- *Automated Rollouts & Rollbacks*: Zero-downtime ανανεώσεις με στρατηγική rolling update. Αυτόματο rollback σε προηγούμενη έκδοση αν το deployment αποτύχει health checks.
- *Μυστικά & Διαχείριση Διαμόρφωσης (Secret & Configuration Management)*: Ασφαλής αποθήκευση διαπιστευτηρίων, κλειδιών API και διαμορφώσεων (configurations) χωρίς ενσωμάτωση στο image — μέσω Secrets και ConfigMaps.

³ <https://www.cncf.io/projects/kubernetes/>

- **Ενορχήστρωση Αποθήκευσης (Storage Orchestration):** Αυτόματη προσάρτηση αποθηκευτικού χώρου (τοπικό, NFS, AWS EBS, Azure Disk κτλ.) χωρίς χειρωνακτικές ρυθμίσεις.

2.2.3 Αρχιτεκτονική Kubernetes

Μια συστάδα Kubernetes αποτελείται από δύο είδη κόμβων (Kubernetes Authors, 2024b):



Εικόνα 2.1 — Αρχιτεκτονική Kubernetes: Control Plane και Worker Nodes

Control Plane (Master Node) (Κύριος Κόμβος): Συνήθως υπάρχει ένας τέτοιος κόμβος ανά συστάδα, ο οποίος δομείται με βάση τα εξής συστατικά του Kubernetes:

- **kube-apiserver:** Το κεντρικό API gateway του cluster. Όλες οι αλληλεπιδράσεις — `kubectl` commands, SDK calls, controller loops — περνούν από εδώ. Εκθέτει RESTful API και εκτελεί αυθεντικοποίηση, εξουσιοδότηση (RBAC) και admission control.
- **etcd:** Distributed key-value store που αποθηκεύει ολόκληρη την κατάσταση (desired state) της συστάδας. Είναι η «μνήμη» της συστάδας — χωρίς etcd backup, μια βλάβη σημαίνει απώλεια ολόκληρης της κατάστασης της συστάδας. Παρακάτω φαίνεται το alert `'etcdInsufficientMembers'` — αναμενόμενο σε single-node Minikube.
- **kube-scheduler:** Παρακολουθεί νέα pods που δεν έχουν ανατεθεί σε κόμβο και επιλέγει τον καταλληλότερο κόμβο για την τοποθέτησή τους βάσει resource requests, affinity/anti-affinity rules (κανόνες εγγύτητας), taints/tolerations (μηχανισμός όπου ένας κόμβος «απωθεί» pods εκτός αν αυτά έχουν ρητή «ανοχή» — χρησιμοποιείται π.χ. για δέσμευση κόμβων σε συγκεκριμένα workloads).
- **kube-controller-manager:** Εκτελεί control loops που συνεχώς παρακολουθούν την κατάσταση οποιουδήποτε αντικειμένου Kubernetes και την επαναφέρουν στο επιθυμητό επίπεδο. Περιλαμβάνει: Deployment Controller, ReplicaSet Controller, Node Controller, Job Controller (δηλαδή ελεγκτές ειδικούς ως προς εξειδικευμένους φόρτους εργασίας και κόμβους).
- **cloud-controller-manager:** Επικοινωνεί με cloud provider APIs (AWS, Azure, GCP) για δημιουργία load balancers, επίμονους αποθηκευτικούς χώρους (persistent volumes) και τη διαχείριση κόμβων νέφους (cloud node management).

Worker Node (Κόμβος Εργασίας): Μια συστάδα μπορεί να περιλαμβάνει πολλούς τέτοιους κόμβους. Εσωτερικά οργανώνονται με βάση τα εξής συστατικά:

- *kubelet*: Είδος πράκτορα (agent) που τοποθετείται σε κάθε κόμβο εργασίας. Λαμβάνει PodSpecs (προσδιορισμούς για pods) από τον kube-apiserver και εξασφαλίζει ότι τα containers/pods εκτελούνται σωστά. Εκτελεί liveness/readiness probes και αναφέρει τη σχετική κατάσταση (status) πίσω στο Control Plane (Επίπεδο Ελέγχου).
- *kube-proxy*: Network proxy που διατηρεί iptables/IPVS rules για υλοποίηση Kubernetes Services (σταθερά DNS/IP σημεία πρόσβασης σε ένα σύνολο από pods με κοινό πρότυπο (template), ανεξάρτητα από αλλαγές/επανεκκινήσεις τους). Επιτρέπει επικοινωνία τόσο μεταξύ pods εντός της συστάδας όσο και μεταξύ pods και εξωτερικών συστημάτων/χρηστών εκτός συστάδας.
- *Container Runtime*: Το λογισμικό που εκτελεί τα containers στο επίπεδο κάθε κόμβου — containerd (προεπιλογή), CRI-O, ή Docker Engine μέσω dockershim. Αποτελεί το επίπεδο αφαίρεσης μεταξύ Kubernetes και του υποκείμενου λειτουργικού συστήματος.

2.2.4 Βασικά Kubernetes Objects

Τα Kubernetes objects είναι δηλωτικές (declarative) περιγραφές της επιθυμητής κατάστασης αντικειμένων, γραμμένες σε YAML ή JSON (Kubernetes Authors, 2024b). Παρακάτω παρουσιάζονται τα βασικότερα αντικείμενα καθώς και ένα παράδειγμα ορισμού αντικειμένου τύπου pod σε YAML μορφοποίηση:

Pod — Η μικρότερη μονάδα ανάπτυξης:

```
apiVersion: v1
kind: Pod
metadata:
  name: webapp-pod
spec:
  containers:
  - name: webapp
    image: nginx:alpine
    ports:
    - containerPort: 80
  resources:
    requests:
      memory: "64Mi"
      cpu: "100m" # 100 millicores = 0.1 CPU
    limits:
      memory: "128Mi"
      cpu: "200m"
```

Το παραπάνω παράδειγμα ορίζει ένα Pod με ένα container (nginx:alpine) που εκθέτει την πόρτα 80. Το πεδίο requests καθορίζει τους ελάχιστους πόρους που δεσμεύονται για το container, ενώ το limits το ανώτατο όριο χρήσης — αν το container ξεπεράσει το memory limit (όριο μνήμης) τερματίζεται, ενώ αν ξεπεράσει το CPU limit (όριο κεντρικής μονάδας επεξεργασίας) περιορίζεται (throttling).

Ο Πίνακας 2.6 παραθέτει τα βασικά Kubernetes objects που χρησιμοποιούνται στην πλατφόρμα, με τον αντίστοιχο ρόλο και ένα ενδεικτικό παράδειγμα χρήσης έκαστου.

Πίνακας 2.6 — Βασικά Kubernetes Objects

Object	Περιγραφή	Παράδειγμα Χρήσης
Pod	Βασική μονάδα εφέμερης μορφής διότι δεν είναι άμεσα διαχειρίσιμη από το K8s. Μπορεί να έχει 1-N containers ενώ υποστηρίζεται shared network/storage. Ephemeral.	Ένα container εφαρμογής web (π.χ. nginx) με τα απαραίτητα resource requests/limits
Deployment	Διαχειρίζεται stateless apps/components. Υποστηρίζει rolling updates, rollbacks και desired replicas (δηλ. επιθυμητά αντίγραφα/στιγμιότυπα pods). Αυτό και τα επόμενα 2 αντικείμενα αποτελούν φόρτους εργασίας μέσω των οποίων το K8s διαχειρίζεται έμμεσα pods με διαφορετική σημασιολογία εκτέλεσης.	Διαχείριση 3 αντιγράφων (replicas) μιας stateless web εφαρμογής (ή του backend της) με rolling updates
StatefulSet	Διαχειρίζεται stateful apps/components με σταθερή ταυτότητα (stable identity) και ordered deployment (ταξινομημένη διάταξη).	Prometheus, Alertmanager ή η βάση δεδομένων μιας εφαρμογής
DaemonSet	Εξασφαλίζει ότι ένα ή παραπάνω Pods εκτελούνται σε κάθε κόμβο. Ιδανικό για υλοποίηση λειτουργικότητας που θα πρέπει να είναι διαθέσιμη σε όλη τη συστάδα, όπως είναι οι πράκτορες παρακολούθησης (monitoring agents).	node-exporter σε Minikube/Kubernetes nodes (ο πράκτορας παρακολούθησης που εκθέτει μετρικές επιπέδου λειτουργικού συστήματος/κόμβου στο Prometheus)
Service	Σταθερό DNS/IP endpoint για πρόσβαση σε ένα σύνολο από Pods. Τύποι: ClusterIP (εσωτερική πρόσβαση εντός cluster, προεπιλογή), NodePort (έκθεση σε στατικό port σε κάθε node, για εξωτερική πρόσβαση σε dev/test), LoadBalancer (προσθέτει external load balancer του cloud provider, για production).	Υπηρεσίες Kubernetes πάνω από τα pods του RESTful API μιας εφαρμογής
ConfigMap	Configuration data χωριστά από την εικόνα δοχείου/pod. Προσάρτηση σε δοχείο/pod ως env vars ή volumes.	Grafana configuration

Object	Περιγραφή	Παράδειγμα Χρήσης
Secret	Ασφαλής αποθήκευση ευαίσθητων δεδομένων με δυνατότητα κωδικοποίησης base64. Δυνατότητα αναφοράς μυστικών ως τιμές μεταβλητών περιβάλλοντος ή έκχυσής τους στο σύστημα αρχείων δοχείου ενός pod.	Grafana admin password, Διαπιστευτήρια χρήστη βάσης δεδομένων εφαρμογής
Namespace	Επιτρέπει τον λογικό διαμερισμό της συστάδας. Μπορεί να υποστηρίξει απομόνωση πόρων (resource isolation).	'monitoring' namespace
Ingress	HTTP/HTTPS routing, TLS termination, path-based routing.	Platform frontend
HPA	Horizontal Pod Autoscaler. Κλιμακώνει οριζόντια τα pods ενός φόρτου εργασίας βάσει metrics.	Χρησιμοποιείται ανά διατεταγμένη εφαρμογή της πλατφόρμας (όχι μόνο στο demo-webapp)
PVC/PV	Persistent storage (ή αιτήματα που οδηγούν στην δημιουργία του) που επιβιώνει Pod restarts.	Δεδομένα της TSDB του Prometheus

2.2.5 Μοντελοποίηση Αρχιτεκτονικής Διάταξης — 6 Επίπεδα

Κατά τον σχεδιασμό της αρχιτεκτονικής διάταξης μιας εφαρμογής σε συστάδα Kubernetes, υπάρχουν 6 διακριτές αποφάσεις:



Εικόνα 2.2 — Τα 6 επίπεδα αποφάσεων στη μοντελοποίηση αρχιτεκτονικής διάταξης Kubernetes

1. *Containerization*: Πακετάρισμα κάθε μικρο-υπηρεσίας της εφαρμογής (και του frontend εφόσον χρειάζεται) ως Docker image με Dockerfile. Επιλογή base image, build layers, multi-stage builds (τεχνική Dockerfile όπου ο κώδικας μεταγλωττίζεται σε ένα προσωρινό στάδιο build και μόνο το τελικό εκτελέσιμο αντιγράφεται στην τελική, μικρότερη εικόνα).
2. *Σχεδίαση (Design) Pod*: Αποφάσεις για το ποια containers συγκατοικούν σε ένα Pod, καθώς όλα τα containers ενός Pod μοιράζονται δίκτυο και αποθηκευτικό χώρο. Βασικά μοτίβα: Sidecar — βοηθητικό container που επεκτείνει τη λειτουργικότητα του κύριου χωρίς να το τροποποιεί (π.χ. log shipper που διαβάζει logs από κοινό volume). Ambassador — proxy container που απλοποιεί την επικοινωνία του κύριου container με εξωτερικές υπηρεσίες. Adapter — container που μετατρέπει την έξοδο του κύριου δοχείου σε τυποποιημένη μορφή (π.χ. format μετρικών).

3. *Τύπος Φόρτου Εργασίας (Workload Type)*: Stateless apps → Deployment. Stateful apps (DBs) → StatefulSet. Per-node tasks → DaemonSet. One-time jobs → Job/CronJob.
4. *Δικτύωση (Networking)*: Services για Service Discovery και load balancing. Ingress για εξωτερική πρόσβαση. Network Policies για έλεγχο επικοινωνίας pod-to-pod.
5. *Storage*: Για μη μόνιμα δεδομένα χρησιμοποιούνται εφήμεροι αποθηκευτικοί χώροι (ephemeral volumes), όπως emptyDir (κοινός χώρος μεταξύ containers ενός Pod, διαγράφεται με το Pod) και configMap/secret volumes. Για δεδομένα που πρέπει να επιβιώνουν restarts, χρησιμοποιούνται PersistentVolumeClaims και StorageClass για dynamic provisioning (δυναμική τροφοδοσία) (AWS EBS, Azure Disk).
6. *Διαμόρφωση & Μυστικά (Configuration & Secrets)*: ConfigMaps για μη ευαίσθητες διαμορφώσεις (non-sensitive config), Secrets για passwords/tokens/certs (δηλαδή διαπιστευτήρια και πιστοποιητικά). Έκχυση (injection) και των δύο ως env vars (μεταβλητές περιβάλλοντος) ή volume mounts (προσαρτήσεις αποθηκευτικού χώρου – δηλαδή προσαρτώνται σε συγκεκριμένα μέρη του συστήματος αρχείων των δοχείων/pods).

2.2.6 Τρόποι Διάταξης σε Kubernetes

kubectl apply:

Η πιο βασική μέθοδος. Ο χρήστης γράφει YAML manifests (δηλαδή μοντέλα των αντικειμένων Kubernetes της εφαρμογής) και τα εφαρμόζει. Κατάλληλο για testing και απλές εφαρμογές.

Helm:

Package manager (διαχειριστής πακέτων) για Kubernetes. Εφαρμογές πακετάρονται ως «Charts» — templated YAML manifests με παραμετροποιήσιμες τιμές μέσω values.yaml. Υποστηρίζει versioning, dependencies, rollbacks.

Παράδειγμα εγκατάστασης/διάταξης Prometheus μέσω Helm:

```
helm install prometheus prometheus-community/kube-prometheus-stack \
--namespace monitoring \
--set grafana.adminPassword=admin123
```

Kustomize:

Overlay-based customization χωρίς templating: αντί για μεταβλητές μέσα σε templates (όπως στο Helm), το Kustomize ξεκινά από ένα «base» σύνολο YAML αρχείων και εφαρμόζει «patches» (μερικές τροποποιήσεις) πάνω σε αυτό για κάθε περιβάλλον. Ιδανικό για παραλλαγές dev/staging/prod από το ίδιο base configuration, χωρίς πλήρη αντιγραφή των YAML αρχείων ή ενσωμάτωση template syntax μέσα σε αυτά.

GitOps (ArgoCD, Flux):

Χρήση αποθετηρίου (repository) Git ως single source of truth (μοναδική πηγή αλήθειας). Αλλαγές μέσω PRs (Pull Requests — αιτήματα συγχώνευσης αλλαγών κώδικα), αυτόματη εφαρμογή στη συστάδα. Προτείνεται ως μελλοντική επέκταση στο Κεφάλαιο 6.

2.2.7 Εργαλεία Τοπικής Ανάπτυξης

Ο Πίνακας 2.7 συγκρίνει τα διαθέσιμα εργαλεία τοπικής ανάπτυξης συστάδων Kubernetes (Kubernetes Authors, 2024b), εκ των οποίων επιλέχθηκε το Minikube για την παρούσα διπλωματική.

Πίνακας 2.7 — Εργαλεία Τοπικής Ανάπτυξης Kubernetes Συστάδας

Εργαλείο	Περιγραφή	Πλεονεκτήματα	Χρήση
Minikube	Επιτρέπει την τοπική ανάπτυξη μονο-κομβικών συστάδων Kubernetes. Υποστηρίζει πολλούς οδηγούς (Docker, VirtualBox).	Πλήρη K8s χαρακτηριστικά και δυνατότητα εγκατάστασης προσθέτου (addon) από το K8s οικοσύστημα	Διπλωματική (v1.35.1)
Kind (Kubernetes in Docker)	Συστάδες Kubernetes μέσα σε Docker containers.	Γρήγορο, multi-node υποστήριξη, CI-friendly	Αγωγούς δοκιμών (Testing pipelines)
K3d	K3s (ελαφριά διανομή Kubernetes) εντός Docker. Εκτελέσιμο < 100 MB.	Ελαφρύτατο, τάχιστη εκκίνηση	Edge, IoT, resource-limited περιβάλλοντα
MicroK8s	Snap-based K8s από Canonical.	Εύκολη εγκατάσταση σε ΛΣ Ubuntu, ενοποιημένα (integrated) addons	Ubuntu development

2.2.8 Εργαλεία Παρακολούθησης Kubernetes

Prometheus:

Σύστημα παρακολούθησης και βάση δεδομένων χρονοσειρών ανοικτού κώδικα (Prometheus Authors, 2024), το οποίο αναπτύχθηκε από τη SoundCloud το 2012 και σήμερα αποτελεί έργο του Cloud Native Computing Foundation (2024). Χρησιμοποιεί pull-based μοντέλο: scraping metrics από targets σε τακτά διαστήματα (default 15s). Query language: PromQL.

Βασικά συστατικά του Prometheus:

- *Prometheus Server*: Συλλέγει και αποθηκεύει μετρικές στη μορφή χρονοσειρών (time-series) σε TSDB (Time Series DB) με τοπικό αποθηκευτικό χώρο. Retention default (προκαθορισμένη περίοδος διατήρησης τιμών μετρικών): 15 ημέρες.
- *Exporters (Εξαγωγείς)*: Εκθέτουν μετρικές (metrics) από συστήματα χωρίς native Prometheus support. Παραδείγματα εξαγωγών: node_exporter (OS metrics), kube-state-metrics (K8s object state), blackbox_exporter (endpoint monitoring).

- **Alertmanager:** Λαμβάνει ειδοποιήσεις (alerts) από Prometheus (βάσει alerting rules), καταργεί τις διπλότυπες και τις ομαδοποιεί, και τέλος τις δρομολογεί σε email, Slack (πλατφόρμα ομαδικής επικοινωνίας) ή PagerDuty (υπηρεσία διαχείρισης περιστατικών με on-call ειδοποιήσεις).
- **Pushgateway:** Για short-lived jobs που δεν προλαβαίνουν να γίνουν scraped από το Prometheus πριν τερματίσουν (batch jobs, cron jobs). Σε αυτή την περίπτωση, το μοντέλο αντιστρέφεται από pull σε push: η ίδια η εργασία στέλνει (pushes) τις μετρικές της στο Pushgateway πριν τερματίσει, και το Prometheus στη συνέχεια κάνει scrape από εκεί.

Grafana:

Πλατφόρμα οπτικοποίησης (visualization) που συνδέεται με το Prometheus και με πλήθος άλλων πηγών δεδομένων (Grafana Labs, 2024). Δημιουργεί interactive dashboards με auto-refresh. Στο Κεφάλαιο 5 (Ενότητα 5.2) παρουσιάζεται η χρήση του Grafana με πραγματικά δεδομένα από την επίδειξη.

kube-state-metrics:

Υπηρεσία που παρακολουθεί K8s API objects και εκθέτει τα δεδομένα τους ως μετρικές Prometheus. Παραδείγματα: kube_pod_status_phase (αριθμός pods ανά κατάσταση — π.χ. Running, Pending, Failed), kube_deployment_status_replicas (αριθμός αντιγράφων ενός Deployment στην τρέχουσα κατάστασή του), kube_hpa_status_current_replicas (τρέχων αριθμός replicas όπως τον βλέπει το HPA).

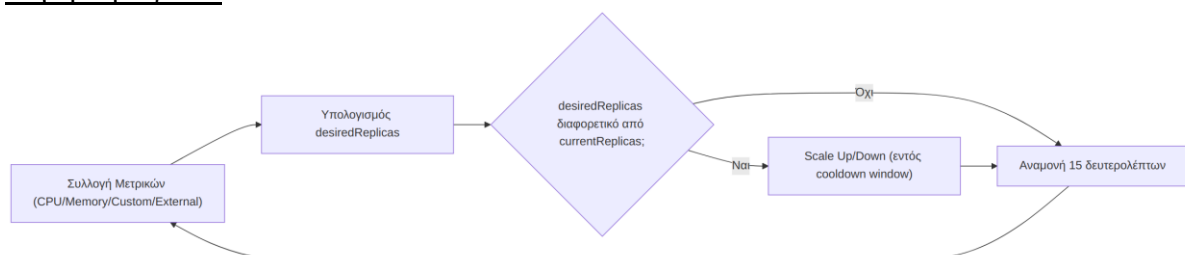
Οι όροι «endpoints» εδώ αναφέρονται στα σημεία (targets) που παρακολουθεί το Prometheus — τόσο τα ίδια τα application pods όσο και οι K8s API server, controller-manager, scheduler κ.λπ.

2.2.9 Κλιμάκωση Pods & Κόμβων

2.2.9.1 Horizontal Pod Autoscaler (HPA) — Αναλυτική Ανάλυση

Το HPA αποτελεί τον κεντρικό μηχανισμό οριζόντιας αυτόματης κλιμάκωσης pods στο Kubernetes (Kubernetes Authors, 2024a) και βρίσκεται στο επίκεντρο της παρούσας διπλωματικής εργασίας. Λειτουργεί ως control loop κάθε 15 δευτερόλεπτα (default), υπολογίζει τιμές μετρικών (metrics) και προσαρμόζει τα pod replicas (αντίγραφα/στιγμιότυπα pods):

Αλγόριθμος HPA:



Εικόνα 2.3 — Βρόχος ελέγχου (control loop) του Horizontal Pod Autoscaler

$\text{desiredReplicas} = \text{ceil}[\text{currentReplicas} \times (\text{currentMetricValue} / \text{desiredMetricValue})]$, όπου: currentReplicas = τρέχων αριθμός pods, $\text{currentMetricValue}$ = τρέχουσα τιμή της παρακολουθούμενης μετρικής (π.χ. % CPU χρήσης), $\text{desiredMetricValue}$ = το όριο-στόχος (threshold) που έχει οριστεί.

Παράδειγμα από επίδειξη: `currentReplicas=2`, `CPU=56%`, `threshold=50%` → `ceil[2 × (56/50)] = ceil[2.24] = 3 pods` (σενάριο scale-up διότι ο αριθμός των pods πρέπει να αυξηθεί κατά 1, από 2 σε 3).

Τύποι μετρικών που υποστηρίζει το HPA:

- *Resource Metrics (Μετρικές Χρήσης Πόρων) (ενσωματωμένες)*: CPU% και Memory% (% → ποσοστό χρήσης) ανά pod (άρα κατά μήκος όλων των αντιγράφων του). Απαιτεί εγκατεστημένο metrics-server (τρέχει ως Deployment, όχι DaemonSet, καθώς συγκεντρώνει μετρικές μέσω του Kubelet API κάθε κόμβου χωρίς να χρειάζεται τοπικό αντίγραφο ανά κόμβο).
- *Custom Metrics*: Application-specific metrics μέσω Custom Metrics API (π.χ. HTTP requests/sec, queue depth). Απαιτεί custom metrics adapter (π.χ. prometheus-adapter) που εγκαθίσταται ξεχωριστά στη συστάδα και λειτουργεί ως «γέφυρα» — μεταφράζει PromQL queries σε Kubernetes Custom Metrics API responses, ώστε το HPA να μπορεί να τις διαβάσει με τον ίδιο τρόπο όπως τις ενσωματωμένες (built-in) resource metrics.
- *External Metrics (Εξωτερικές Μετρικές)*: Μετρικές από εξωτερικές πηγές (π.χ. Prometheus, Datadog (εμπορική πλατφόρμα παρακολούθησης) και KEDA (Kubernetes Event-Driven Autoscaling — επέκταση αυτόματης κλιμάκωσης βάσει εξωτερικών συμβάντων)).

HPA Cooldown Periods: Το HPA έχει built-in cooldown για αποφυγή flapping (φαινόμενο όπου το σύστημα κλιμακώνει επανειλημμένα πάνω-κάτω σε σύντομο χρονικό διάστημα λόγω μικρών διακυμάνσεων γύρω από το threshold):

- `scaleDown.stabilizationWindowSeconds`: 300s (5 λεπτά) default — δεν κάνει scale-down αν οι μετρικές δεν είναι χαμηλές για 5 λεπτά.
- `scaleUp.stabilizationWindowSeconds`: 0s default — το scale-up (αύξηση replicas) γίνεται αμέσως όταν χρειαστεί.

2.2.9.2 KEDA — Kubernetes Event-Driven Autoscaling

Το KEDA⁴ επεκτείνει το HPA υποστηρίζοντας κλιμάκωση οδηγούμενη από γεγονότα που προέρχονται από εξωτερικές πηγές, όπως ουρές μηνυμάτων (RabbitMQ, Kafka, Azure Service Bus), βάσεις δεδομένων ή HTTP traffic. Επιτρέπει κλιμάκωση από 0 pods (scale-to-zero) κατά απουσία φόρτου, εξοικονομώντας σημαντικό cloud κόστος. Αποτελεί φυσική επέκταση της παρούσας πλατφόρμας για queue-based scaling σενάρια.

2.2.9.3 Vertical Pod Autoscaler (VPA)

Ενώ το HPA αυξάνει τον αριθμό των pods (horizontal), το VPA αυξάνει τους πόρους ανά pod (vertical) — δηλαδή τα CPU/memory requests και limits. Το VPA αναλύει ιστορικά δεδομένα χρήσης και προτείνει βέλτιστες τιμές. Δεν χρησιμοποιείται στην παρούσα εργασία αλλά αποτελεί συμπληρωματικό εργαλείο για workloads που δεν κλιμακώνονται εύκολα οριζόντια (π.χ. stateful services). Μπορεί επίσης να παρέχει κατάλληλες τιμές για CPU/memory requests ώστε να εξασφαλίζεται η σωστή εκτέλεση των pods βάσει των ελάχιστων απαιτήσεων τους σε πόρους.

⁴ <https://keda.sh/>

2.2.9.4 Cluster Autoscaler / Karpenter — Κλιμάκωση Κόμβων

Ενώ το HPA κλιμακώνει τον αριθμό των pods, το Cluster Autoscaler (CA)⁵ κλιμακώνει τον αριθμό των nodes (στη συστάδα). Όταν τα pods δεν μπορούν να προγραμματιστούν λόγω έλλειψης πόρων, το CA προσθέτει νέους nodes. Από την άλλη μεριά, αν οι κόμβοι υπο-χρησιμοποιούνται, τερματίζονται.

Το Karpenter⁶ (αρχικά ανεπτυγμένο στην AWS) είναι εναλλακτική υλοποίηση που λειτουργεί γρηγορότερα, επιλέγοντας μάλιστα το βέλτιστο instance type για κάθε workload ενώ λειτουργεί σε επίπεδο μεμονωμένων κόμβων και όχι ομάδων κόμβων (όπως ισχύει για το CA).

Η παρούσα πλατφόρμα υλοποιεί χειροκίνητη κλιμάκωση nodes μέσω Terraform (FA-A07) και θα μπορούσε να ενσωματώσει CA αυτόματα σε μελλοντική έκδοση.

2.3 Infrastructure as Code (IaC) & Terraform

2.3.1 Τι είναι το Infrastructure as Code

Το Infrastructure as Code (IaC) είναι η πρακτική διαχείρισης και παροχής υπολογιστικής υποδομής μέσω κώδικα και αρχείων ρύθμισης (configuration files), αντί για χειρωνακτικές διαδικασίες ή διαδραστικά εργαλεία (Morris, 2020). Αντί να δημιουργεί κάποιος διακομιστές (servers) κάνοντας κλικ σε web consoles, τους ορίζει σε αρχεία κώδικα που αποθηκεύονται, ελέγχονται και επαναχρησιμοποιούνται όπως ο οποιοσδήποτε άλλος κώδικας λογισμικού.

Αποτελεί θεμελιώδη πρακτική του DevOps κινήματος (Kim et al., 2016), επιτρέποντας στις ομάδες να διαχειρίζονται υποδομή με την ίδια πειθαρχία και αυτοματισμό που διαχειρίζονται εφαρμογές.

2.3.2 Βασικά Χαρακτηριστικά & Οφέλη IaC

- *Αναπαραγωγιμότητα (Reproducibility)*: Η ίδια διαμόρφωση παράγει πάντα ίδια υποδομή, εξαλείφοντας το πρόβλημα του configuration drift — αποκλίσεις μεταξύ dev, staging, production περιβαλλόντων.
- *Έλεγχος Εκδόσεων (Version Control)*: Τα IaC αρχεία αποθηκεύονται σε Git. Αυτό επιτρέπει τις εξής δυνατότητες: πλήρες ιστορικό αλλαγών με timestamp/author, rollback, code review (επιθεώρηση κώδικα) μέσω pull requests.
- *Αυτοματισμός (Automation)*: Δημιουργία, ενημέρωση και καταστροφή υποδομής αυτόματα μέσω CI/CD pipelines — εξάλειψη ανθρώπινου σφάλματος.
- *Τεκμηρίωση (Documentation as Code)*: Ο κώδικας IaC είναι πάντα ενημερωμένος και ακριβής, χωρίς ξεχωριστά έγγραφα.
- *Συνεργασία*: Pull requests, code reviews, branching — η ίδια εργαλειοθήκη που χρησιμοποιείται για συνεργατική ανάπτυξη κώδικα λογισμικού/εφαρμογών.
- *Κλιμάκωση*: Δημιουργία δεκάδων πανομοιότυπων περιβαλλόντων με ένα αρχείο — αδύνατο χειρωνακτικά.

⁵ <https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler>

⁶ <https://karpenter.sh/>

- *Εξοικονόμηση Κόστους*: Αυτόματη καταστροφή πόρων εκτός ωραρίου (test environments), άμεση ανίχνευση αποκλίσεων.

2.3.3 Declarative vs Imperative Μοντέλο

Δηλωτικό Μοντέλο (Declarative):

Ο χρήστης ορίζει τί θέλει — την επιθυμητή κατάσταση — και το εργαλείο IaC αναλαμβάνει το πώς. Μπορεί να εκτελεστεί επίσης πολλές φορές με το ίδιο αποτέλεσμα. Αν η υποδομή ήδη υπάρχει εν μέρει, το εργαλείο κάνει μόνο τις απαραίτητες αλλαγές. Παραδείγματα: Terraform & AWS CloudFormation.

```
# Declarative: «Θέλω 3 VMs t3.medium»
resource "aws_instance" "web" {
  count          = 3
  instance_type = "t3.medium"
}
```

Στο παραπάνω παράδειγμα, ο χρήστης δηλώνει απλώς ότι επιθυμεί 3 VMs τύπου t3.medium (count=3). Το Terraform συγκρίνει αυτή τη δηλωμένη κατάσταση με την τρέχουσα πραγματική κατάσταση και αποφασίζει μόνο του ποιες ενέργειες χρειάζονται (δημιουργία, τροποποίηση ή καμία ενέργεια αν οι 3 VMs υπάρχουν ήδη).

Επιτακτικό Μοντέλο (Imperative):

Ο χρήστης ορίζει το πώς — βήμα-βήμα εντολές. Δεν εγγυάται idempotency (δηλ. ότι αν εκτελεστεί ο κώδικας πολλαπλές φορές θα έχουμε πάντοτε την ίδια τελική κατάσταση). Παραδείγματα: Pulumi (εργαλείο IaC), Ansible (imperative mode), scripts με AWS CLI.

```
# Imperative
import pulumi
import pulumi_aws as aws

ami_id = "ami-0123456789abcdef0" # replace with a valid AMI

for i in range(3):
    server = aws.ec2.Instance(
        f"vm-{i+1}",
        ami=ami_id,
        instance_type="t3.medium",
        tags={
            "Name": f"terraform-like-vm-{i+1}"
        }
    )

    pulumi.export(f"vm_{i+1}_id", server.id)
```

Το παραπάνω παράδειγμα είναι παρόμοιο με το προηγούμενο αλλά χρησιμοποιείται κλασική επανάληψη για την δημιουργία 3 VMs τύπου t3.medium στο νέφος της Amazon. Όπως μπορεί να γίνει αντιληπτό, όντως δημιουργούνται τα 3 VMs χωρίς όμως να εξετάζεται η τρέχουσα κατάσταση. Συνεπώς, αν ο ίδιος κώδικας επανα-εκτελεστεί, τότε θα δημιουργηθούν άλλα 3.

Το δηλωτικό μοντέλο είναι γενικά προτιμότερο για υποδομή για αυτό και εφαρμόστηκε στη διπλωματική μέσω Terraform.

2.3.4 Σύγκριση Εργαλείων IaC

Η σύγκριση ακολουθεί το πλαίσιο αξιολόγησης του Morris (2020). Τα κριτήρια είναι: Μοντέλο (Declarative — ο χρήστης δηλώνει το επιθυμητό αποτέλεσμα· Imperative — ο χρήστης δηλώνει τα βήματα), Γλώσσα (η γλώσσα ή μορφή αρχείων ρύθμισης), Πολλαπλά Νέφη (αν το εργαλείο υποστηρίζει περισσότερους από έναν cloud providers).

Ο Πίνακας 2.8 συγκρίνει τα κυριότερα εργαλεία IaC ως προς το μοντέλο, τη γλώσσα και την υποστήριξη πολλαπλών νεφών.

Πίνακας 2.8 — Σύγκριση Εργαλείων IaC

Εργαλείο	Εταιρεία	Μοντέλο	Γλώσσα	Πολλαπλά Νέφη	Σχόλιο
Terraform	HashiCorp	Declarative	HCL	✓ (1000+ providers)	Επιλογή διπλωματικής
AWS CloudFormation	Amazon	Declarative	JSON/YAML	Χ (AWS only)	Native AWS, no cost
Azure Bicep/ARM	Microsoft	Declarative	JSON/Bicep	Χ (Azure only)	Native Azure
GCP Deployment Mgr	Google	Declarative	YAML/Jinja2	Χ (GCP only)	Native GCP
Ansible	Red Hat	Imp./Decl.	YAML	✓	Config mgmt + IaC
Pulumi	Pulumi Corp	Declarative	Python/TS/Go	✓	Code-first approach
Helm	CNCF	Declarative	YAML templates	✓ (K8s)	Διαχειριστής πακέτων Kubernetes

Βασικό συμπέρασμα: μόνο το Terraform, το Ansible και το Pulumi υποστηρίζουν εγγενώς πολλαπλά νέφη — τα εργαλεία native σε συγκεκριμένο provider (CloudFormation, Bicep, Deployment Manager) αποκλείστηκαν λόγω ασυμβατότητας με τον στόχο multi-cloud της παρούσας πλατφόρμας. Μεταξύ Terraform, Ansible και Pulumi, επιλέχθηκε το Terraform λόγω ωριμότητας, μεγαλύτερης κοινότητας και υποστήριξης καθαρά declarative μοντέλου⁷.

⁷ Υποστηρίζεται και προγραμματιστικό μοντέλο μέσω CDKTF αλλά το μοντέλο προσδιορισμού υποδομής εξακολουθεί να είναι δηλωτικό και σε αυτή την περίπτωση. Σε αντίθεση με άλλα εργαλεία όπου είναι επιτακτικό

2.3.5 Terraform — Αναλυτική Παρουσίαση

Το Terraform (HashiCorp, 2024), το οποίο κυκλοφόρησε το 2014, είναι ένα εργαλείο IaC ανοικτού κώδικα που υιοθετεί το δηλωτικό μοντέλο, προσφέροντας την σχετική γλώσσα HCL (HashiCorp Configuration Language). Επιτρέπει ορισμό, παροχή και διαχείριση υποδομής σε οποιονδήποτε cloud provider ή on-premises περιβάλλον με ενιαία γλώσσα ρύθμισης/διαμόρφωσης (δηλαδή την HCL).

Βασικά χαρακτηριστικά:

- *Multi-Provider Support (Υποστήριξη Πολλαπλών Παρόχων)*: Υποστηρίζονται εκατοντάδες πάροχοι (AWS, Azure, GCP, Kubernetes, GitHub, Datadog, Cloudflare) μέσω plugins (προσθέτων). Ιδανικό για multi-cloud αρχιτεκτονικές.
- *State Management (Διαχείριση Καταστάσεων)*: Διατηρεί terraform.tfstate — αντιστοιχία μεταξύ configuration (επιθυμητής κατάστασης) και πραγματικών cloud resources (πόρων νέφους) (πραγματικής κατάστασης). Αποθηκεύεται τοπικά ή remotely (S3, Azure Blob, Terraform Cloud).
- *Execution Plans (Πλάνα Εκτέλεσης)*: η εντολή terraform plan δημιουργεί ένα preview των αλλαγών πριν αυτές εφαρμοστούν — ο χρήστης βλέπει ακριβώς τι θα αλλάξει.
- *Idempotency*: Πολλαπλές εκτελέσεις με την ίδια διαμόρφωση οδηγούν πάντοτε στο ίδιο αποτέλεσμα.
- *Modular Design (Αρθρωτή Σχεδίαση)*: Modules — επαναχρησιμοποιήσιμες μονάδες που ενθυλακώνουν τη λογική υποδομής.

2.3.6 HashiCorp Configuration Language (HCL)

Το HCL σχεδιάστηκε για ανθρώπινη αναγνωσιμότητα (HashiCorp, 2024). Βασικά δομικά στοιχεία:

Resource (Πόρος) — ένα infrastructure component (συστατικό υποδομής):

```
resource "aws_eks_cluster" "main" {
  name           = var.cluster_name
  role_arn       = aws_iam_role.eks_role.arn
  version        = "1.28"
  vpc_config {
    subnet_ids           = module.vpc.private_subnets
    endpoint_private_access = true
    endpoint_public_access  = true
  }
}
```

Το παράδειγμα ορίζει έναν πόρο τύπου aws_eks_cluster με όνομα «main» (δηλ. μια συστάδα Kubernetes). Καθορίζει το IAM role που θα χρησιμοποιήσει το cluster (role_arn), την έκδοση Kubernetes (version), και τις ρυθμίσεις δικτύου (vpc_config) — τα private subnets στα οποία θα αναπτυχθεί, καθώς και αν το API endpoint (της συστάδας) θα είναι προσβάσιμο ιδιωτικά ή/και δημόσια.

Variable (Μεταβλητή) — επιτρέπει τη παραμετροποίηση:

```
variable "cluster_name" {
  description = "Name of the EKS cluster"
  type        = string
  default     = "my-cluster"
```

```
}
```

Το παράδειγμα ορίζει μια μεταβλητή εισόδου «cluster_name» τύπου string, με προεπιλεγμένη τιμή «my-cluster» — επιτρέπει επαναχρησιμοποίηση του ίδιου κώδικα με διαφορετικές τιμές (ως προς την μεταβλητή αυτή) (πχ. στο πλαίσιο δημιουργίας συστάδων με διαφορετικό όνομα) χωρίς μεταβολή του πηγαίου αρχείου.

Module — επαναχρησιμοποιήσιμη μονάδα:

```
module "eks" {  
  source          = "terraform-aws-modules/eks/aws"  
  cluster_name    = var.cluster_name  
  cluster_version = "1.28"  
  min_size        = var.min_nodes  
  max_size        = var.max_nodes  
  desired_size    = var.desired_nodes  
  instance_types  = [var.instance_type]  
}
```

Το module ενσωματώνει ένα δημόσιο, επαναχρησιμοποιήσιμο module τρίτου (terraform-aws-modules/eks/aws) περνώντας τις κατάλληλες παραμέτρους (όνομα, έκδοση, εύρος κόμβων) — αποφεύγοντας την επανεγγραφή πολύπλοκου κώδικα EKS από την αρχή.

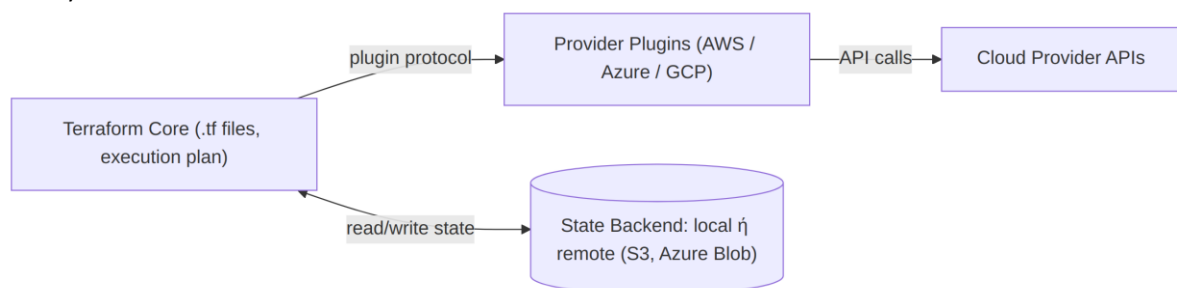
Output (Εξοδος) — εξαγωγή τιμών μετά την εφαρμογή της εντολής apply:

```
output "cluster_endpoint" {  
  description = "EKS cluster API endpoint"  
  value       = module.eks.cluster_endpoint  
}
```

Το output εξάγει την τιμή του cluster endpoint μετά την ολοκλήρωση του terraform apply, ώστε να είναι διαθέσιμη προς χρήση από άλλα modules (ή εν γένει κώδικες Terraform) ή από το ίδιο το backend της πλατφόρμας (π.χ. για αποθήκευση στο πεδίο clusterEndpoint του DEPLOYMENT).

2.3.7 Αρχιτεκτονική Terraform

Τρία κύρια συστατικά απαρτίζουν την αρχιτεκτονική του Terraform (HashiCorp, 2024):

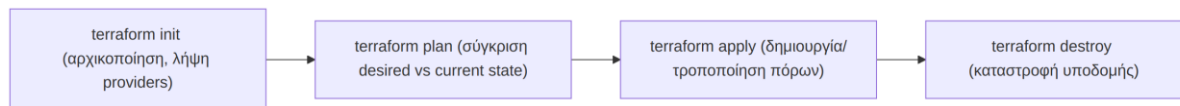


Εικόνα 2.4 — Αρχιτεκτονική Terraform: Core, Provider Plugins και State Backend

- **Terraform Core:** Διαβάζει .tf files, διαχειρίζεται την κατάσταση (state), υπολογίζει το πλάνο εκτέλεσης (execution plan) (diff: desired state vs current state), ενορχηστρώνει την εκτέλεση ενός πλάνου.

- *Provider Plugins*: δυαδικά πρόσθετα (binary plugins) που γνωρίζουν πώς να επικοινωνήσουν με συγκεκριμένα APIs (AWS, Azure, GCP). Εκφορτώνονται αυτόματα κατά την εκτέλεση της εντολής terraform init (δείτε παρακάτω τη σημασιολογία της εντολής αυτής) από το Terraform Registry (ένα αποθετήριο που περιλαμβάνει τέτοια πρόσθετα).
- *State Backend*: υποδομή όπου αποθηκεύεται το state file. Τοπικά (local) για ανάπτυξη, remote (S3, Azure Blob) για συνεργατική χρήση με state locking (κλείδωμα κατάστασης).

2.3.8 Κύκλος Εργασίας Terraform



Εικόνα 2.5 — Κύκλος εργασίας Terraform (βλ. αναλυτικά Πίνακα 2.9)

Ο Πίνακας 2.9 αναλύει τα τέσσερα στάδια του κύκλου εργασίας Terraform, όπως αυτά αξιοποιούνται από την πλατφόρμα (βλ. Εικόνα 2.5).

Πίνακας 2.9 — Στάδια Κύκλου Εργασίας Terraform

Βήμα	Εντολή	Περιγραφή	Αποτέλεσμα
1	terraform init	Αρχικοποίηση working directory (φακέλου εργασίας). Λήψη προσθέτων providers από Terraform Registry. Αρχικοποίηση backend.	Δημιουργία .terraform/ directory με providers
2	terraform plan	Σύγκριση desired state (code) με current state (tfstate). Παραγωγή execution plan.	Report (αναφορά πλάνου εκτέλεσης): N πόρους προς προσθήκη, M προς αλλαγή, K προς καταστροφή
3	terraform apply	Εφαρμογή πλάνου (εκτέλεσης). Οδηγεί σε σχετικές κλήσεις σε cloud APIs. Επιπλέον, ενημερώνει την tfstate. Ζητά αρχικά επιβεβαίωση από τον χρήστη για την έναρξη της εκτέλεσης του πλάνου.	Δημιουργία/τροποποίηση/καταστροφή resources
4	terraform destroy	Καταστροφή όλων των πόρων που διαχειρίζεται το state. Ζητά αρχική επιβεβαίωση από τον χρήστη.	Πλήρης cleanup (καθαρισμός/καταστροφή) — οδηγεί σε μηδενικό κόστος νέφους (εφόσον δεν χρησιμοποιούνται πλέον πόροι του)

2.3.9 Τρόποι Χρήσης Terraform

- *Τοπική Χρήση*: Ο DevOps το εκτελεί από τοπικό μηχάνημα. Συνεπώς, η κατάσταση αποθηκεύεται τοπικά. Ιδανικός τρόπος για τοπική ανάπτυξη/δοκιμή εφαρμογών/υποδομής.
- *Remote State (S3/Azure Blob)*: Η κατάσταση διατηρείται σε κοινόχρηστο storage. Υποστηρίζεται state locking μέσω DynamoDB/Azure Storage. Ιδανικός τρόπος για ομαδική χρήση του Terraform (δηλ. για ομαδική διαχείριση της υποδομής).
- *Terraform Cloud/Enterprise*: Διαχειρίσιμη πλατφόρμα IaC από HashiCorp. Υποστηρίζει απομακρυσμένη εκτέλεση (remote execution), εφαρμογή πολιτικών (policy enforcement) (Sentinel), χρήση ιδιωτικών αποθετηρίων (private registry) και διαχείριση ομάδων (team management). Ιδανικό για επαγγελματική χρήση.
- *CI/CD Integration*: Αυτόματη εκτέλεση σε αγωγούς συστημάτων CI/CD, όπως GitHub Actions/GitLab CI pipeline κατά την ενοποίηση κώδικα. Υποστηρίζεται διαχείριση υποδομής τύπου GitOps.

2.3.10 Υποστηριζόμενοι Cloud Providers & Υπηρεσίες

Ο Πίνακας 2.10 συνοψίζει τις αντίστοιχες υπηρεσίες υπολογισμού, δικτύωσης, Kubernetes και IAM ανά πάροχο νέφους (Amazon Web Services, 2024b· Microsoft, 2024a).

Πίνακας 2.10 — Σύγκριση Cloud Providers & Υπηρεσιών

Cloud	Compute	Networking	Kubernetes Service	IAM (Identity and Access Management — υπηρεσία διαχείρισης χρηστών, ρόλων και δικαιωμάτων πρόσβασης σε πόρους του νέφους)
AWS	EC2, Lambda, ECS	VPC (ιδιωτικό δίκτυο), ALB/NLB (load balancers), Route53 (υπηρεσία DNS — διαχειρίζεται την αντιστοίχιση domain names σε IP διευθύνσεις, διαφορετικό ρόλο από τον load balancer που	EKS (Elastic Kubernetes Service)	IAM Users/Roles/Policies

Cloud	Compute	Networking	Kubernetes Service	IAM (Identity and Access Management — υπηρεσία διαχείρισης χρηστών, ρόλων και δικαιωμάτων πρόσβασης σε πόρους του νέφους)
		κατανέμει το φορτίο μεταξύ ήδη γνωστών endpoints)		
Azure	VMs, Functions, Container Instances	VNet (Virtual Network — ιδιωτικό δίκτυο), App Gateway (load balancer επιπέδου εφαρμογής), NSG (Network Security Group — τείχος προστασίας δικτύου)	AKS (Azure Kubernetes Service)	Azure AD, RBAC, Service Principals
GCP	GCE, Cloud Run, GKE Autopilot	VPC, Cloud LB, Cloud DNS	GKE (Google Kubernetes Engine)	Service Accounts, IAM Roles

Κεφάλαιο 3: Σχετικές Εργασίες

3.1 Εισαγωγή

Στο κεφάλαιο αυτό παρουσιάζονται και αναλύονται σχετικές ερευνητικές εργασίες και συστήματα που έχουν αναπτυχθεί στους τομείς της παρακολούθησης, της αυτο-κλιμάκωσης και της διαχείρισης Kubernetes workloads σε περιβάλλοντα νέφους. Για κάθε εργασία παρουσιάζεται η βασική συνεισφορά, τα πλεονεκτήματα και τα μειονεκτήματα της σε σχέση με τους στόχους της παρούσας διπλωματικής. Στο τέλος πραγματοποιείται συστηματική σύγκριση μέσω ειδικά επιλεγμένων κριτηρίων που αναδεικνύουν τον νεωτερισμό της παρούσας διπλωματικής εργασίας.

3.2 Εργασία 1: «Kubernetes Autoscaling: YoYo Attack Vulnerability and Mitigation» (Ben David & Bremner-Barr, 2021)

Βασική Συνεισφορά:

Η εργασία αυτή εστιάζει στη μελέτη της αυτο-κλιμάκωσης στο Kubernetes και ειδικότερα στην ευπάθεια του Horizontal Pod Autoscaler (HPA) σε επιθέσεις τύπου "YoYo attack". Ο επιτιθέμενος εκμεταλλεύεται τον μηχανισμό αυτόματης κλιμάκωσης εκπέμποντας περιοδικές ριπές φόρτου, ώστε το σύστημα να ταλαντώνεται διαρκώς μεταξύ φάσεων scale-up και scale-down. Έτσι μια κλασική επίθεση άρνησης υπηρεσίας μετατρέπεται σε επίθεση Economic Denial of Sustainability (EDoS): το θύμα υφίσταται οικονομική ζημία, πληρώνοντας για τους πρόσθετους πόρους που δεσμεύονται προκειμένου να εξυπηρετηθεί ο τεχνητός φόρτος, ενώ παράλληλα υποβαθμίζεται η απόδοση λόγω του χρόνου που απαιτεί η κλιμάκωση. Οι συγγραφείς αξιολογούν πειραματικά την ανθεκτικότητα της αυτόματης κλιμάκωσης του Kubernetes στην επίθεση YoYo σε περιβάλλον Google Cloud Platform και καταλήγουν ότι, παρότι ο χρόνος εκκίνησης ενός container είναι σαφώς μικρότερος από εκείνον μιας εικονικής μηχανής, το Kubernetes παραμένει ευάλωτο, διότι η κλιμάκωση κόμβων εξακολουθεί να στηρίζεται σε εικονικές μηχανές. Τέλος, αξιολογούν μοντέλα μηχανικής μάθησης ικανά να ανιχνεύουν με ακρίβεια την επίθεση σε συστάδα Kubernetes.

Πλεονεκτήματα:

- Βαθιά κατανόηση της εσωτερικής λειτουργίας του HPA και των αδυναμιών του σε ακραίες συνθήκες
- Λεπτομερής πειραματική ανάλυση με μετρήσιμα αποτελέσματα σε πραγματική συστάδα Kubernetes επί του Google Cloud Platform
- Τα προτεινόμενα μοντέλα ανίχνευσης λειτουργούν επί των υφιστάμενων μετρικών, χωρίς τροποποίηση του πυρήνα του Kubernetes

Μειονεκτήματα & Διαφορές με την Παρούσα:

- Εστιάζει αποκλειστικά στην ασφάλεια της αυτόματης κλιμάκωσης — δεν ασχολείται με ολιστική διαχείριση εφαρμογών
- Δεν υποστηρίζει multi-cloud περιβάλλοντα, IaC-based διάταξη ή web πλατφόρμα διαχείρισης
- Κλιμάκωση αποκλειστικά βάσει CPU metrics — χωρίς custom QoS κανόνες ορισμένους από τον χρήστη

- Δεν παρέχει unified monitoring dashboard, ειδοποιήσεις ή διαχείριση χρηστών

3.3 Εργασία 2: «AdapPF: Self-Adaptive Scrape Interval for Monitoring in Geo-Distributed Cluster Federations» (Huang & Pierre, 2023)

Βασική Συνεισφορά:

Η εργασία αντιμετωπίζει το πρόβλημα της ενοποιημένης παρακολούθησης όταν οι συστάδες Kubernetes είναι κατανεμημένες σε πολλαπλές γεωγραφικές τοποθεσίες και παρόχους νέφους. Η προτεινόμενη αρχιτεκτονική AdapPF θεμελιώνεται στον μηχανισμό Prometheus federation — τοπικό στιγμιότυπο Prometheus ανά συστάδα και ιεραρχικός κεντρικός Prometheus που συλλέγει προ-συναθροισμένες χρονοσειρές από τα σημεία /federate — και προσθέτει αυτο-προσαρμοζόμενο διάστημα συλλογής (scrape interval) ανά συστάδα, το οποίο μεταβάλλεται δυναμικά ανάλογα με τη μεταβλητότητα των παρατηρούμενων μετρικών. Η αξιολόγηση διεξήχθη στο γεω-κατανεμημένο testbed Grid'5000 και κατέγραψε μείωση 36% της διασυστοιχειακής κίνησης δικτύου, με αποτελέσματα συγκρίσιμα εκείνων που επιτυγχάνει η Prometheus Federation με σταθερό διάστημα συλλογής πέντε δευτερολέπτων.

Πλεονεκτήματα:

- Αντιμετωπίζει άμεσα το πρόβλημα της ενοποιημένης παρακολούθησης γεω-κατανεμημένων συστάδων — το πλησιέστερο σενάριο στους στόχους της παρούσας διπλωματικής
- Η ομοσπονδιακή συλλογή μετρικών (Prometheus federation) αποτελεί εγγενές χαρακτηριστικό του Prometheus: είναι κλιμακώσιμη και μη παρεμβατική, καθώς δεν απαιτεί τροποποίηση των παρακολουθούμενων εφαρμογών. Επισημαίνεται ότι η παρούσα πλατφόρμα δεν υιοθετεί τον μηχανισμό federation, αλλά απευθείας multi-endpoint querying μέσω του PrometheusService
- Η δυναμική προσαρμογή του διαστήματος συλλογής μειώνει μετρήσιμα το κόστος δικτύου — ζήτημα κρίσιμο σε multi-cloud περιβάλλοντα, όπου η διακίνηση δεδομένων μεταξύ παρόχων χρεώνεται

Μειονεκτήματα & Διαφορές:

- Η συνεισφορά περιορίζεται στο επίπεδο συλλογής μετρικών — δεν προβλέπεται αυτόματη κλιμάκωση, διάταξη υποδομής μέσω IaC ούτε ορισμός κανόνων ποιότητας υπηρεσίας
- Απαιτεί χειρωνακτική διαμόρφωση της ομοσπονδίας Prometheus από εξειδικευμένο προσωπικό — δεν παρέχεται διαδικτυακή διεπαφή
- Δεν παρέχει ειδοποιήσεις παραβίασης κανόνων QoS, διαχείριση χρηστών ούτε ενέργειες αντίδρασης στις παρατηρούμενες παραβιάσεις

3.4 Εργασία 3: «Autopilot: Workload Autoscaling at Google» (Rzadca et al., 2020)

Βασική Συνεισφορά:

Παρουσιάζει τον Autopilot, το εσωτερικό σύστημα αυτόματης διαμόρφωσης πόρων της Google, το οποίο λειτουργεί επί της ιδιόκτητης υποδομής συστάδων της εταιρείας. Το σύστημα ρυθμίζει αυτόματα τόσο το πλήθος των ταυτόχρονων εργασιών μιας δουλειάς (οριζόντια κλιμάκωση) όσο και τα όρια CPU και μνήμης ανά εργασία (κατακόρυφη κλιμάκωση). Στόχος είναι η ελαχιστοποίηση του περιθωρίου αδράνειας (slack) — της απόστασης μεταξύ του δηλωμένου ορίου και της πραγματικής χρήσης — χωρίς να αυξηθεί ο κίνδυνος τερματισμού εργασιών λόγω εξάντλησης μνήμης ή υποβάθμισης απόδοσης λόγω στραγγαλισμού CPU. Για τον σκοπό αυτό συνδυάζει αλγορίθμους μηχανικής μάθησης επί ιστορικών δεδομένων προηγούμενων εκτελέσεων με ένα σύνολο επιμελώς ρυθμισμένων ευρετικών κανόνων. Τα δημοσιευμένα αποτελέσματα αναφέρουν slack 23% για τις αυτόματα ρυθμιζόμενες εργασίες έναντι 46% για τις χειροκίνητα διαχειριζόμενες, δεκαπλάσια μείωση των εργασιών που πλήττονται σοβαρά από σφάλματα εξάντλησης μνήμης, και κάλυψη άνω του 48% της συνολικής κατανάλωσης πόρων του στόλου της Google.

Πλεονεκτήματα:

- Η αξιοποίηση ιστορικών δεδομένων εκτέλεσης επιτρέπει διαμόρφωση ορίων εκ των προτέρων, αντί για αντιδραστική προσαρμογή μετά την παραβίαση
- Επικυρωμένο σε πραγματική κλίμακα παραγωγής, με ποσοτικά τεκμηριωμένα οφέλη (slack 23% έναντι 46%)
- Ταυτόχρονη οριζόντια και κατακόρυφη κλιμάκωση — σημαντική μείωση της σπατάλης πόρων και, κατ' επέκταση, του κόστους

Μειονεκτήματα & Διαφορές:

- Αποτελεί εσωτερικό, ιδιόκτητο σύστημα της υποδομής της Google — δεν διατίθεται ως προϊόν ούτε είναι μεταφέσιμο σε άλλους παρόχους νέφους
- Δεν προβλέπει κανόνες ποιότητας υπηρεσίας οριζόμενους από τον χρήστη· η λογική των συστάσεων δεν είναι παραμετροποιήσιμη από τον τελικό χρήστη
- Δεν παρέχει IaC-based διάταξη, web UI ή RBAC
- Εστιάζει σε resource metrics (CPU/memory), όχι application-level μετρικές (latency, error rate)

3.5 Εργασία 4: «Crossplane — A Universal Control Plane for Cloud Infrastructure» (Crossplane Authors, 2024)

Βασική Συνεισφορά:

Επεκτείνει το Kubernetes ώστε να λειτουργεί ως universal control plane για cloud infrastructure. Επιτρέπει τον ορισμό των cloud resources (EC2, RDS, AKS) ως Kubernetes Custom Resources (CRDs) και έπειτα τα δημιουργεί αυτόματα στον αντίστοιχο provider. Ενσωματώνει IaC απευθείας στο οικοσύστημα του Kubernetes με φυσική υποστήριξη GitOps.

Πλεονεκτήματα:

- Εξαιρετικά καινοτόμος — αξιοποιεί το γνωστό Kubernetes API αντί για νέα εργαλεία
- Υποστηρίζει πολλαπλούς providers και επιτρέπει declarative multi-cloud infrastructure management
- Φυσική ενσωμάτωση με ροές εργασίας (workflows) τύπου GitOps

Μειονεκτήματα & Διαφορές:

- Απαιτεί βαθιά γνώση Kubernetes CRDs και operators — απότομη καμπύλη εκμάθησης
- Δεν παρέχει το ίδιο web UI, monitoring dashboards ή QoS-aware αυτο-κλιμάκωση, αν και μπορεί να στηριχθεί σε υπάρχοντα σχετικά εργαλεία του Kubernetes
- Η ενσωμάτωση απαιτεί από τον οργανισμό να λειτουργεί ήδη μια συστάδα Kubernetes ως control plane, κάτι που δεν αποτελεί προϋπόθεση για την παρούσα πλατφόρμα

3.6 Εργασία 5: «TORCH: A TOSCA-Based Orchestrator of Multi-Cloud Containerised Applications» (Tomarchio et al., 2021)

Βασική Συνεισφορά:

Η εργασία προτείνει το TORCH, πλαίσιο ενορχήστρωσης θεμελιωμένο στο πρότυπο TOSCA (Topology and Orchestration Specification for Cloud Applications) του οργανισμού OASIS, το οποίο επιτρέπει την περιγραφή της τοπολογίας μιας εφαρμογής — κλασικής ή containerised — με τρόπο ανεξάρτητο από πάροχο και τη διάταξή της σε πολλαπλά νέφη. Η αρχιτεκτονική διαχωρίζει τη ροή provisioning, μοντελοποιημένη σε BPMN, από την κλήση των ιδιοτήτων API κάθε παρόχου· ως αποτέλεσμα, η προσθήκη υποστήριξης νέας πλατφόρμας απαιτεί ελάχιστη υλοποίηση. Το πρωτότυπο επιδεικνύεται σε δύο διαφορετικές πλατφόρμες συστάδων containers.

Πλεονεκτήματα:

- Το TOSCA αποτελεί ανοικτό πρότυπο του OASIS — η περιγραφή των εφαρμογών είναι ανεξάρτητη παρόχου (vendor-agnostic) και δυνητικά διαλειτουργική (interoperable) μεταξύ εργαλείων
- Ο διαχωρισμός της ροής provisioning από τα ιδιόκτητα API των παρόχων περιορίζει δραστικά το κόστος υποστήριξης νέου νέφους
- Αντιμετωπίζει το ίδιο πρόβλημα εγκλωβισμού σε πάροχο (vendor lock-in) με την παρούσα εργασία, αλλά με διαφορετικά μέσα: πρότυπο περιγραφής τοπολογίας (TOSCA) αντί για Infrastructure as Code (Terraform)

Μειονεκτήματα & Διαφορές:

- Η σύνταξη του TOSCA είναι πολύπλοκη και δεν είναι προσιτή σε χρήστες χωρίς τεχνικό υπόβαθρο
- Η αλληλεπίδραση γίνεται μέσω περιγραφικών αρχείων και γραμμής εντολών — δεν παρέχεται διαδικτυακή διεπαφή προσβάσιμη σε μη εξειδικευμένους χρήστες
- Το πλαίσιο καλύπτει τον κύκλο διάταξης, αλλά δεν ενσωματώνει παρακολούθηση σε πραγματικό χρόνο ούτε αυτόματη κλιμάκωση βάσει κανόνων ποιότητας υπηρεσίας
- Η υιοθέτηση του TOSCA παραμένει περιορισμένη, κυρίως στον ακαδημαϊκό χώρο, σε σχέση με το Terraform που χρησιμοποιεί η παρούσα διπλωματική

3.7 Συγκριτικός Πίνακας Εργασιών

Ο Πίνακας 3.1 συγκεντρώνει τη συγκριτική αξιολόγηση των πέντε σχετικών εργασιών έναντι της παρούσας πλατφόρμας, βάσει δώδεκα κριτηρίων.

Πίνακας 3.1 — Συγκριτικός Πίνακας Σχετικών Εργασιών

Κριτήριο	Ben David & Bremler-Barr (2021)	Huang & Pierre (2023)	Rzadca et al. (2020)	Crossplane Authors (2024)	Tomarchio et al. (2021)	Παρούσα
Multi-cloud	Χ	✓	Χ (GCP)	✓	✓	✓
Real-time monitoring	Μερ.	✓	Χ	Χ	Χ	✓
QoS-aware κλιμάκωση	Χ	Χ	Μερ.	Χ	Χ	✓
Custom μετρικές χρήστη	Χ	Χ	Χ	Χ	Χ	●
IaC-based διάταξη	Χ	Χ	Χ	Μερ.	✓	✓
Web πλατφόρμα / UI	Χ	Χ	Χ	Χ	Χ	●
Αυτόματη καταστροφή	Χ	Χ	Χ	Μερ.	✓	✓
Monitoring Dashboards	Χ	Χ	Χ	Χ	Χ	●
Ειδοποιήσεις QoS	Χ	Χ	Χ	Χ	Χ	●
Διαχείριση χρηστών/RBAC	Χ	Χ	Χ	Χ	Χ	✓
Χειροκίνητη κλιμάκωση	Χ	Χ	Χ	Χ	Χ	●
Φιλικότητα Χρήστη	Χαμ.	Χαμ.	Υψηλή	Χαμ.	Χαμ.	Υψηλή

Το σύμβολο ● στη στήλη της παρούσας εργασίας δηλώνει ότι η δυνατότητα υποστηρίζεται από το μοντέλο πεδίου και τις υπηρεσίες του backend, αλλά δεν εκτίθεται μέσω γραφικής διεπαφής στην παρούσα έκδοση (§4.5).

Επεξήγηση τελευταίων κριτηρίων: «Διαχείριση χρηστών/RBAC» — δυνατότητα διαχείρισης πολλαπλών χρηστών με διαφορετικά επίπεδα δικαιωμάτων πρόσβασης (ρόλους)· «Χειρωνακτική Κλιμάκωση» — δυνατότητα του χρήστη να προσαρμόσει ο ίδιος τον αριθμό πόρων/κόμβων, πέραν της αυτόματης κλιμάκωσης· «Φιλικότητα Χρήστη» — κατά πόσο απαιτείται εξειδικευμένη τεχνική γνώση (CLI, YAML) για

χρήση του συστήματος, σε αντιδιαστολή με προσβασιμότητα μέσω απλής web διεπαφής.

3.8 Συμπεράσματα

Από την ανάλυση των σχετικών εργασιών προκύπτει ότι, ενώ υπάρχουν αξιόλογες ερευνητικές προσπάθειες σε επιμέρους τομείς (monitoring, αυτο-κλιμάκωση, IaC orchestration), καμία εργασία δεν συνδυάζει όλες αυτές τις λειτουργίες σε μια ενιαία, φιλική web πλατφόρμα. Ειδικότερα εντοπίζεται ερευνητικό κενό όσον αφορά:

- μια ολιστική web πλατφόρμα για multi-cloud Kubernetes management χωρίς απαίτηση CLI/εξειδικευμένης σε πάροχο/εργαλείο γνώσης
- QoS-aware αυτο-κλιμάκωση με custom κανόνες που ορίζει ο χρήστης (metric + operator + threshold + action)
- Ενοποίηση IaC (Terraform) + real-time monitoring (Prometheus/Grafana) + RBAC σε μία πλατφόρμα
- Αυτοματοποιημένη καταστροφή (πόρων) για εξοικονόμηση cloud κόστους

Η παρούσα διπλωματική καλύπτει ακριβώς αυτό το κενό, αποδεικνύοντας μέσω πειραματικής επίδειξης (Κεφάλαιο 5, Ενότητες 5.2.4 και 5.2.5, με συγκεντρωτική αποτίμηση στην §5.3) ότι η QoS-aware αυτόματη κλιμάκωση λειτουργεί πρακτικά σε πραγματικό περιβάλλον Kubernetes και ότι ο πλήρης κύκλος ζωής μιας εφαρμογής είναι διαχειρίσιμος αποκλειστικά μέσω διαδικτυακής διεπαφής. Τα δώδεκα κριτήρια του Πίνακα 3.1 μεταφράζονται άμεσα σε λειτουργικές και μη λειτουργικές απαιτήσεις, οι οποίες προδιαγράφονται συστηματικά στην Ενότητα 4.2 και αποτιμώνται ως προς τον βαθμό ικανοποίησής τους στην Ενότητα 4.7.

Κεφάλαιο 4: Ανάπτυξη Συστήματος

4.1 Μοντέλο Ανάπτυξης — Waterfall

Για την ανάπτυξη της πλατφόρμας επιλέχθηκε το μοντέλο καταρράκτη (waterfall model), το κλασικό σειριακό μοντέλο ανάπτυξης λογισμικού που εισήγαγε ο Royce (1970) και το οποίο οργανώνει τον κύκλο ζωής σε διακριτές, διαδοχικές φάσεις με σαφή παραδοτέα ανά φάση:

1. **1ο Στάδιο — Ανάλυση Απαιτήσεων (Ενότητα 4.2):** προσδιορισμός λειτουργικών (FA) και μη λειτουργικών (MFA) απαιτήσεων ανά τύπο χρήστη και διατύπωση σεναρίων χρήσης.
2. **2ο Στάδιο — Σχεδίαση (Ενότητα 4.3):** μοντελοποίηση σε UML (διάγραμμα περιβάλλοντος, συστατικών, περιπτώσεων χρήσης, ακολουθίας, κλάσεων) και λήψη αρχιτεκτονικών αποφάσεων.
3. **3ο Στάδιο — Υλοποίηση (Ενότητες 4.4 και 4.5):** ανάπτυξη backend (Spring Boot), frontend (React.js), στοίβας παρακολούθησης, υποδομής ως κώδικα (Terraform) και αγωγών CI/CD.
4. **4ο Στάδιο — Δοκιμή και Αξιολόγηση (Ενότητες 4.6 και 4.7):** επαλήθευση λειτουργικότητας μέσω unit και integration tests, πειραματική επίδειξη βάσει σεναρίων χρήσης, αξιολόγηση απόδοσης έναντι στόχων SLA και αποτίμηση του βαθμού ικανοποίησης των απαιτήσεων.

Το μοντέλο του καταρράκτη προκρίθηκε έναντι ευέλικτων (agile) προσεγγίσεων για τρεις λόγους: (α) οι απαιτήσεις ήταν σαφώς προσδιορισμένες εξαρχής και δεν αναμενόταν ουσιώδης μεταβλητότά τους, (β) η ομάδα ανάπτυξης αποτελείτο από ένα άτομο, οπότε τα οφέλη των agile τελετουργικών συντονισμού δεν είχαν εφαρμογή, και (γ) η ακαδημαϊκή φύση του έργου ευνοεί την ολοκληρωμένη τεκμηρίωση της σχεδίασης πριν από την υλοποίηση.

4.2 Ανάλυση Απαιτήσεων

Οριοθέτηση εύρους υλοποίησης. Οι απαιτήσεις που ακολουθούν καταγράφουν το πλήρες λειτουργικό εύρος που προδιαγράφηκε κατά τη φάση ανάλυσης. Η υλοποίηση ιεραρχήθηκε ώστε να καλυφθεί κατά προτεραιότητα ο πυρήνας της εργασίας — η αυτόματη κλιμάκωση βάσει κανόνων ποιότητας υπηρεσίας και η διάταξη υποδομής μέσω Infrastructure as Code — και δευτερευόντως οι υποστηρικτικές λειτουργίες διαχείρισης λογαριασμού καθώς και η γραφική τους έκθεση. Ο βαθμός στον οποίο κάθε απαίτηση τελικά ικανοποιήθηκε αποτιμάται συστηματικά στην Ενότητα 4.7· οι αποκλίσεις δηλώνονται ρητά, ώστε η αναφορά να αποτυπώνει την πραγματική κατάσταση της υλοποίησης και όχι την αρχική πρόθεση.

4.2.1 Είδη Χρηστών

Η πλατφόρμα αναγνωρίζει τρεις κατηγορίες χρηστών, όπως παρουσιάζονται στον Πίνακα 4.1. Ο *Κανονικός Χρήστης* και ο *Διαχειριστής* αποτελούν εξειδικεύσεις μιας κοινής αφηρημένης οντότητας «Χρήστης», η οποία συγκεντρώνει τις λειτουργίες που είναι κοινές και στους δύο ρόλους· η διάκριση αυτή αποτυπώνεται ρητά στα διαγράμματα περιπτώσεων χρήσης της Ενότητας 4.3.3.

Πίνακας 4.1 — *Είδη Χρηστών της Πλατφόρμας*

Χρήστης	Περιγραφή	Πρόσβαση
Επισκέπτης	Μη εγγεγραμμένος. Πρόσβαση μόνο στην αρχική σελίδα.	Δημόσια: landing page, αίτηση εγγραφής
Χρήστης (αφηρημένος υπερ-ρόλος)	Κάθε αυθεντικοποιημένος χρήστης, ανεξαρτήτως ρόλου.	Κοινές λειτουργίες: σύνδεση/αποσύνδεση, προβολή και ενημέρωση προφίλ, αλλαγή και επαναφορά κωδικού
Κανονικός Χρήστης	Εγγεγραμμένος και ενεργός. Διαχειρίζεται τις δικές του εφαρμογές.	Κληρονομεί τις λειτουργίες του Χρήστη· επιπλέον CRUD εφαρμογών, διάταξη (deploy), παρακολούθηση (monitor), διαγραφή ιδίου λογαριασμού
Διαχειριστής	Υπερ-χρήστης με πλήρη πρόσβαση.	Κληρονομεί τις λειτουργίες του Χρήστη· επιπλέον διαχείριση όλων των λογαριασμών και όλων των εφαρμογών του συστήματος

4.2.2 Λειτουργικές Απαιτήσεις Χρηστών (FA-U)

Πίνακας 4.2 — *Λειτουργικές Απαιτήσεις Χρηστών (FA-U)*

Κωδ.	Απαίτηση	Αναλυτική Περιγραφή
FA-U01	Εγγραφή Χρήστη	Ο επισκέπτης μπορεί να δημιουργήσει νέο λογαριασμό παρέχοντας όνομα χρήστη, email, κωδικό πρόσβασης και ερώτηση και απάντηση ασφαλείας, οι οποίες αποθηκεύονται για μελλοντική υποστήριξη επαναφοράς κωδικού. Ο κωδικός υπόκειται σε κανόνες πολυπλοκότητας (MFA-S06). Ο νέος λογαριασμός απαιτεί ενεργοποίηση από διαχειριστή πριν χρησιμοποιηθεί.
FA-U02	Ενεργοποίηση / Απενεργοποίηση Χρήστη	Ο διαχειριστής μπορεί να ενεργοποιεί ή να απενεργοποιεί λογαριασμούς. Απενεργοποιημένος λογαριασμός δεν επιτρέπει σύνδεση.
FA-U03	Σύνδεση Χρήστη	Ο εγγεγραμμένος και ενεργός χρήστης συνδέεται παρέχοντας τα διαπιστευτήριά του. Η επιτυχής σύνδεση παρέχει πρόσβαση στις λειτουργίες που αντιστοιχούν στον ρόλο του.
FA-U04	Αποσύνδεση Χρήστη	Ο χρήστης αποσυνδέεται είτε χειροκίνητα είτε αυτόματα μετά από περίοδο αδράνειας.
FA-U05	Ενημέρωση Χρήστη	Ο χρήστης ενημερώνει τα στοιχεία του λογαριασμού του· ο

Κωδ.	Απαίτηση	Αναλυτική Περιγραφή
		διαχειριστής μπορεί να τροποποιήσει οποιοδήποτε λογαριασμό.
FA-U06	Αλλαγή Κωδικού Χρήστη	Ο χρήστης αλλάζει τον κωδικό του παρέχοντας τον υφιστάμενο και τον νέο. Ο νέος κωδικός υπόκειται στους κανόνες πολυπλοκότητας και παρέχεται εις διπλούν για αποφυγή λαθών πληκτρολόγησης.
FA-U07	Επαναφορά Κωδικού Χρήστη	Η ερώτηση και η απάντηση ασφαλείας καταγράφονται κατά την εγγραφή ώστε να υποστηρίξουν μελλοντική επαναφορά κωδικού· ο σχετικός μηχανισμός δεν υλοποιήθηκε στην παρούσα έκδοση (FA-U07, Πίνακας 4.15).
FA-U08	Εμφάνιση Χρήστη	Ο χρήστης βλέπει το προφίλ του· ο διαχειριστής βλέπει οποιοδήποτε προφίλ. Ευαίσθητες πληροφορίες δεν εμφανίζονται ποτέ.
FA-U09	Αναζήτηση Χρήστη	Ο διαχειριστής αναζητά χρήστες βάσει κριτηρίων, με σελιδοποιημένα αποτελέσματα.
FA-U10	Διαγραφή Χρήστη	Ο χρήστης διαγράφει τον λογαριασμό του· ο διαχειριστής διαγράφει οποιοδήποτε λογαριασμό εκτός του δικού του. Η διαγραφή συνεπάγεται αφαίρεση όλων των σχετικών δεδομένων (cascade).

4.2.3 Λειτουργικές Απαιτήσεις Εφαρμογών (FA-A)

Πίνακας 4.3 — Λειτουργικές Απαιτήσεις Εφαρμογών (FA-A)

Κωδ.	Απαίτηση	Αναλυτική Περιγραφή
FA-A01	Δημιουργία Εφαρμογής	Ο χρήστης δημιουργεί νέα εφαρμογή παρέχοντας μεταδεδομένα (τίτλος, περιγραφή, έκδοση), ρυθμίσεις διάταξης ανά νέφος (απαιτήσεις πόρων, <code>minNodes/maxNodes</code> , <code>desiredNodes</code>) και κανόνες ποιότητας υπηρεσίας (μετρική, τελεστής, τιμή ορίου, ενέργεια). Υποστηρίζονται προκαθορισμένες και προσαρμοσμένες μετρικές.
FA-A02	Εμφάνιση Εφαρμογής	Ο χρήστης παρακολουθεί σε πραγματικό χρόνο την κατάσταση της εφαρμογής (κόμβοι, rods ανά νέφος, ιστορικό κλιμάκωσης,

Κωδ.	Απαίτηση	Αναλυτική Περιγραφή
		ειδοποιήσεις), πέραν των μεταδεδομένων, των ρυθμίσεων διάταξης και των κανόνων ποιότητας υπηρεσίας.
FA-A03	Τροποποίηση Εφαρμογής	Ο χρήστης τροποποιεί άμεσα τα μεταδεδομένα και τους κανόνες ποιότητας υπηρεσίας, χωρίς επίπτωση στη διάταξη. Αλλαγές στις ρυθμίσεις διάταξης απαιτούν αναδιάταξη, η οποία εκτελείται αυτόματα από την πλατφόρμα (βλ. Ενότητα 4.7.3).
FA-A04	Διάταξη Εφαρμογής	Ο χρήστης εκκινεί τη διάταξη σε επιλεγμένο νέφος. Το σύστημα δημιουργεί αυτόματα την υποδομή (συστάδα Kubernetes, δικτύωση) και εγκαθιστά τα εργαλεία παρακολούθησης.
FA-A05	Καταστροφή Διάταξης Εφαρμογής	Ο χρήστης καταστρέφει την υποδομή διατεταγμένης εφαρμογής μετά από επιβεβαίωση. Το σύστημα αφαιρεί αυτόματα όλους τους πόρους από το νέφος.
FA-A06	Αυτόματη Κλιμάκωση Εφαρμογής	Το σύστημα αξιολογεί περιοδικά τους κανόνες ποιότητας υπηρεσίας κάθε διατεταγμένης εφαρμογής, ανεξάρτητα ανά νέφος. Σε παραβίαση κανόνα εκτελείται η αντίστοιχη ενέργεια (κλιμάκωση pods ή κόμβων, ή ειδοποίηση) και καταγράφεται ιστορικό (ScalingEvent).
FA-A07	Χειροκίνητη Κλιμάκωση Εφαρμογής	Ο χρήστης μεταβάλλει χειροκίνητα τον τρέχοντα αριθμό κόμβων διατεταγμένης εφαρμογής, εντός των ορίων <code>minNodes/maxNodes</code> . Αίτημα εκτός ορίων απορρίπτεται. Ο περιορισμός σε επίπεδο κόμβων οφείλεται στο ότι η κλιμάκωση pods καλύπτεται ήδη από τον μηχανισμό QoS (FA-A06).
FA-A08	Παρακολούθηση Εφαρμογής	Μετρικών Ο χρήστης παρακολουθεί μετρικές απόδοσης σε πραγματικό χρόνο μέσω γραφημάτων, με δυνατότητα επιλογής μετρικής και χρονικού εύρους.
FA-A09	Παροχή Ειδοποιήσεων	Το σύστημα ειδοποιεί τον χρήστη (εντός εφαρμογής και μέσω email) κατά παραβίαση κανόνα ποιότητας υπηρεσίας.

Κωδ.	Απαίτηση	Αναλυτική Περιγραφή
FA-A10	Διαγραφή Εφαρμογής	Ο χρήστης διαγράφει εφαρμογή. Αν είναι διατεταγμένη, η υποδομή καταστρέφεται αυτόματα πριν τη διαγραφή.
FA-A11	Αναζήτηση Εφαρμογών	Ο χρήστης βλέπει τη λίστα εφαρμογών του με αναζήτηση (τίτλος, περιγραφή, ετικέτες), φιλτράρισμα βάσει κατάστασης και σελιδοποίηση.

4.2.4 Μη Λειτουργικές Απαιτήσεις

Οι μη λειτουργικές απαιτήσεις παρουσιάζονται στον Πίνακα 4.4. Η κατηγορία Απόδοση αναλύεται σε τρεις διακριτές απαιτήσεις (MFA-P01–P03), ώστε ο χρόνος απόκρισης των τυπικών λειτουργιών να μην συγχέεται με τους ουσιαστικούς διαφορετικούς χρόνους των λειτουργιών που εμπλέκουν εξωτερική υποδομή. Αντίστοιχα, η αρχική ενιαία απαίτηση κλιμακωσιμότητας αναλύθηκε σε τρεις μετρήσιμες απαιτήσεις (MFA-SC01–SC03), καθώς κάθε μία διαθέτει ξεχωριστό και ανεξάρτητα επαληθεύσιμο κριτήριο αποδοχής.

Πίνακας 4.4 — Μη Λειτουργικές Απαιτήσεις

Κατ.	Κωδ.	Απαίτηση	Κριτήριο Αποδοχής
Χρηστ.	MFA-U01	Responsive UI	Η διεπαφή είναι προσβάσιμη και λειτουργική σε επιτραπέζιους υπολογιστές, tablet και κινητές συσκευές.
Χρηστ.	MFA-U02	Uniform look-and-feel	Ενιαία εμφάνιση σε όλη την εφαρμογή μέσω κοινού συστήματος σχεδίασης (design tokens).
Χρηστ.	MFA-U03	User orientation	Ο χρήστης γνωρίζει πάντα σε ποιο σημείο της εφαρμογής βρίσκεται (breadcrumbs, ενεργές ενδείξεις πλοήγησης).
Χρηστ.	MFA-U04	Ελαχιστοποίηση αλληλεπίδρασης	Κάθε κύρια λειτουργία είναι προσβάσιμη με τον ελάχιστο δυνατό αριθμό αλληλεπιδράσεων (≤ 3 clicks).
Χρηστ.	MFA-U05	AJAX / δυναμικότητα	Οι λειτουργίες ενημερώνονται δυναμικά χωρίς πλήρη επαναφόρτωση σελίδας.
Χρηστ.	MFA-U06	Ενδείξεις κατάστασης φόρτωσης	Ασύγχρονες λειτουργίες πολλών λεπτών (διάταξη,

Κατ.	Κωδ.	Απαίτηση	Κριτήριο Αποδοχής
			αποδιάταξη) συνοδεύονται από ενδείξεις προόδου.
Απόδ.	MFA-P01	Χρόνος απόκρισης τυπικών λειτουργιών	Οι τυπικές λειτουργίες (CRUD εφαρμογών, σύνδεση, ανάκτηση μετρικών) ολοκληρώνονται εντός 3 δευτερολέπτων end-to- end.
Απόδ.	MFA-P02	Χρόνος αντίδρασης αυτόματης κλιμάκωσης	Ο κύκλος αξιολόγησης κανόνων QoS εκτελείται ανά 30 s και η κλιμάκωση rules ολοκληρώνεται εντός 5 λεπτών από την παραβίαση ορίου.
Απόδ.	MFA-P03	Χρόνος διάταξης / καταστροφής υποδομής	Η πλήρης διάταξη συστάδας με στοίβα παρακολούθησης ολοκληρώνεται εντός 20 λεπτών και η καταστροφή εντός 15 λεπτών ανά πάρκο.
Ασφ.	MFA-S01	JWT authentication	Η αυθεντικοποίηση βασίζεται σε tokens περιορισμένης διάρκειας με μηχανισμό ανάκλησης.
Ασφ.	MFA-S02	RBAC	Η πρόσβαση σε κάθε λειτουργία ελέγχεται βάσει του ρόλου του χρήστη.
Ασφ.	MFA-S03	BCrypt	Οι κωδικοί αποθηκεύονται με ασφαλή κατακερματισμό· κανένας κωδικός δεν εμφανίζεται σε απλή μορφή.
Ασφ.	MFA-S04	TLS	Η επικοινωνία με την πλατφόρμα γίνεται αποκλειστικά μέσω HTTPS/TLS.
Ασφ.	MFA-S05	CSRF	Η πλατφόρμα δεν είναι ευάλωτη σε επιθέσεις CSRF. Δεδομένου ότι η αυθεντικοποίηση γίνεται μέσω stateless JWT σε HTTP header (όχι cookies), η πλατφόρμα είναι εγγενώς ανθεκτική χωρίς ξεχωριστό CSRF token.

Κατ.	Κωδ.	Απαίτηση	Κριτήριο Αποδοχής
Ασφ.	MFA-S06	Κανόνες κωδικών	Οι κωδικοί υπόκεινται σε κανόνες πολυπλοκότητας (ελάχιστο μήκος, κεφαλαία/πεζά, ψηφία, ειδικοί χαρακτήρες).
Αρχ.	MFA-A01	Backend / frontend split	Το frontend και το backend αναπτύσσονται και διατάσσονται ανεξάρτητα.
Αρχ.	MFA-A02	REST Level 3	Το API ακολουθεί τις αρχές του αρχιτεκτονικού στυλ REST (Fielding, 2000). Η παροχή πληροφοριών πλοήγησης (HATEOAS) προδιαγράφεται αλλά δεν υλοποιείται στην παρούσα έκδοση, η οποία παραμένει στο επίπεδο 2 του μοντέλου ωριμότητας Richardson (MFA-A02, §4.7.4).
Αρχ.	MFA-A03	Τεκμηρίωση API	Το API συνοδεύεται από αυτόματα παραγόμενη τεκμηρίωση.
Αξιοπ.	MFA-R01	Αξιοπιστία	Η πλατφόρμα ανακάμπτει αυτόματα από προσωρινές αποτυχίες (retry logic). Σε αποτυχία διάταξης, η κατάσταση ενημερώνεται σε FAILED και ο χρήστης ειδοποιείται.
Αξιοπ.	MFA-R02	Διαθεσιμότητα παρακολούθησης	Η προσωρινή αδυναμία σύνδεσης με το Prometheus δεν διακόπτει τη λειτουργία της πλατφόρμας (graceful degradation). Σε μόνιμη αδυναμία (>5 διαδοχικές αποτυχίες με exponential backoff) καταγράφεται ειδοποίηση σφάλματος προς τον διαχειριστή.
Φορητ.	MFA-PO01	Φορητότητα	Η πλατφόρμα εγκαθίσταται σε οποιοδήποτε περιβάλλον, ανεξάρτητα από λειτουργικό σύστημα (Linux, macOS, Windows).
Φορητ.	MFA-PO02	Ανεξαρτησία από πάροχο νέφους	Η πλατφόρμα υποστηρίζει πολλούς

Κατ.	Κωδ.	Απαίτηση	Κριτήριο Αποδοχής
			παρόχους (AWS, Azure) μέσω κοινής διεπαφής χωρίς αλλαγές στον κώδικα. Νέος πάροχος προστίθεται μέσω νέου Terraform module.
Κλιμ.	MFA-SC01	Ταυτόχρονη διαχείριση εφαρμογών	Η πλατφόρμα διαχειρίζεται ταυτόχρονα τουλάχιστον 10 εφαρμογές χωρίς υποβάθμιση απόδοσης.
Κλιμ.	MFA-SC02	Ταυτόχρονη εξυπηρέτηση χρηστών	Η πλατφόρμα εξυπηρετεί ταυτόχρονα τουλάχιστον 10 χρήστες χωρίς υποβάθμιση απόδοσης.
Κλιμ.	MFA-SC03	Σταθερότητα κύκλου αξιολόγησης QoS	Η περιοδική αξιολόγηση κανόνων QoS ολοκληρώνεται εντός του 30-δευτερόλεπτου κύκλου, ανεξαρτήτως αριθμού εφαρμογών.
Κλιμ.	MFA-SC04	Επεκτασιμότητα	Η αρχιτεκτονική επιτρέπει προσθήκη νέων τύπων μετρικών, ενεργειών κλιμάκωσης και παρόχων νέφους χωρίς επιπτώσεις στα υπάρχοντα χαρακτηριστικά.

Το σύνολο των βασικών απαιτήσεων ανέρχεται σε **47** (10 FA-U, 11 FA-A, 26 MFA).

4.2.5 Προαιρετικές Απαιτήσεις

Οι ακόλουθες απαιτήσεις δεν αποτελούν βασικές προδιαγραφές αλλά επιθυμητές επεκτάσεις που υλοποιήθηκαν στο πλαίσιο της εργασίας.

Πίνακας 4.5 — Προαιρετικές Απαιτήσεις

Κατ.	Κωδ.	Απαίτηση	Κριτήριο Αποδοχής
Ασφ.	MFA-OPT01	Σύνδεση μέσω Google (OAuth 2.0)	Ο χρήστης συνδέεται με τον λογαριασμό του Google, χωρίς ξεχωριστό λογαριασμό στην πλατφόρμα.
Αξιοπ.	MFA-OPT02	Αυτοματοποιημένη δοκιμή και CI/CD	Η πλατφόρμα συνοδεύεται από αυτοματοποιημένες δοκιμές (unit, integration) και αγωγό συνεχούς ενοποίησης που

Κατ.	Κωδ.	Απαίτηση	Κριτήριο Αποδοχής
			εκτελείται σε κάθε αλλαγή κώδικα.
Φορητ.	MFA-OPT03	Δοχειοποίηση πλατφόρμας	Για την ίδια την πλατφόρμα (frontend, backend, βάση δεδομένων) παρέχεται Docker Compose configuration για εγκατάσταση σε οποιοδήποτε περιβάλλον.

4.3 Σχεδίαση Συστήματος

Η σχεδίαση αποτυπώνεται σε πέντε επίπεδα αφαίρεσης, από το γενικότερο προς το ειδικότερο: το διάγραμμα περιβάλλοντος (§4.3.1) οριοθετεί το σύστημα έναντι του περιβάλλοντός του, το διάγραμμα συστατικών (§4.3.2) αποδομεί το εσωτερικό του, τα διαγράμματα περιπτώσεων χρήσης (§4.3.3) καταγράφουν τη λειτουργικότητα από τη σκοπιά των δρώντων, τα διαγράμματα ακολουθίας (§4.3.4) περιγράφουν τη δυναμική συμπεριφορά ανά ροή και το διάγραμμα κλάσεων (§4.3.5) το μοντέλο πεδίου.

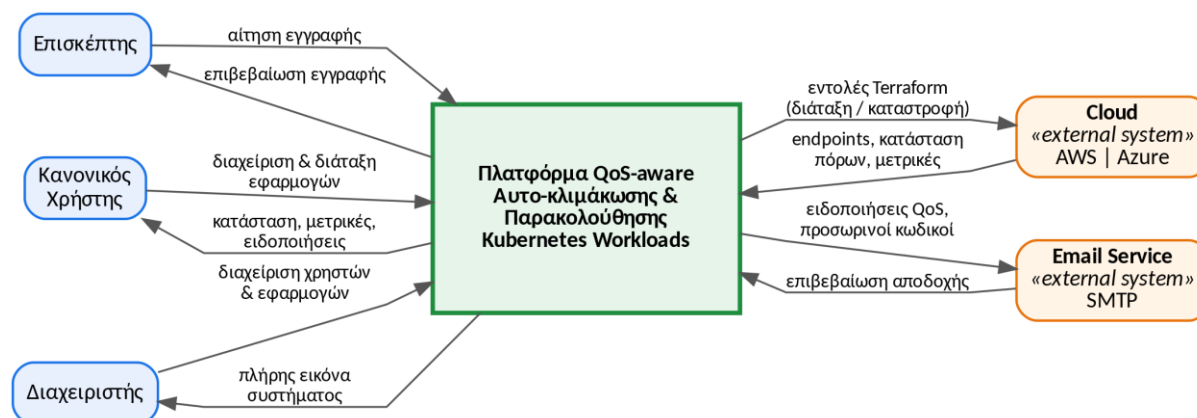
Ενιαία ονοματολογία συστατικών. Για να διασφαλιστεί η συνέπεια μεταξύ όλων των διαγραμμάτων και του κυρίου κειμένου, υιοθετείται το ακόλουθο καθιερωμένο λεξιλόγιο, το οποίο χρησιμοποιείται αναλλοίωτο σε κάθε διάγραμμα του κεφαλαίου:

- **Δρώντες:** Επισκέπτης, Χρήστης (αφηρημένος), Κανονικός Χρήστης, Διαχειριστής.
- **Εξωτερικά συστήματα:** Cloud (υλοποιείται από AWS ή Azure) και Email Service (SMTP). Ο όρος «Cloud» χρησιμοποιείται ομοιόμορφα παντού· οι εμπορικές ονομασίες AWS/Azure εμφανίζονται μόνο ως στερεότυπα/σχόλια εντός του συστατικού.
- **Frontend Layer: AuthModule, ApplicationModule, CommonModule.**
- **Backend Layer — Controllers:** AuthController, UserController, ApplicationController (τρεις ελεγκτές συνολικά).
- **Backend Layer — Services:** UserService, UserService, ApplicationService, TerraformService, MonitoringService, ScalingService, NotificationService.
- **Backend Layer — Repositories:** UserRepository, ApplicationRepository, CloudConfigRepository, QosRuleRepository, DeploymentRepository, ScalingEventRepository, NotificationRepository.
- **Cross-cutting:** SecurityLayer (JwtAuthFilter, JwtUtils, RbacEvaluator).
- **Infrastructure Layer (πλατφόρμας):** PostgreSQL και Terraform, η μηχανή IaC που καλείται ως εξωτερική διεργασία.

Επισημαίνεται ο σαφής διαχωρισμός αρμοδιοτήτων μεταξύ τριών συνεργαζόμενων υπηρεσιών: το PrometheusService αναλαμβάνει αποκλειστικά την επικοινωνία με το Prometheus (μετάφραση ονομάτων μετρικών σε ερωτήματα PromQL και εκτέλεση των κλήσεων HTTP), το MonitoringService την περιοδική αξιολόγηση των κανόνων ποιότητας υπηρεσίας, και το ScalingService την εκτέλεση των ενεργειών κλιμάκωσης. Ο διαχωρισμός αυτός τηρείται ενιαία σε όλα τα διαγράμματα του κεφαλαίου.

4.3.1 Διάγραμμα Περιβάλλοντος (Context Diagram)

Το διάγραμμα περιβάλλοντος παρουσιάζει την πλατφόρμα ως ενιαία οντότητα («μαύρο κουτί») και τις ροές πληροφορίας που ανταλλάσσει με τους δρώντες και τα εξωτερικά συστήματα. Οι ανθρωπinoι δρώντες τοποθετούνται στην αριστερή πλευρά και τα εξωτερικά συστήματα λογισμικού στη δεξιά, ώστε να αποφεύγεται η επικάλυψη ετικετών και να διαχωρίζεται οπτικά η ανθρώπινη αλληλεπίδραση από τη διασύνδεση μηχανής προς μηχανή.



Εικόνα 4.1 — Διάγραμμα περιβάλλοντος της πλατφόρμας

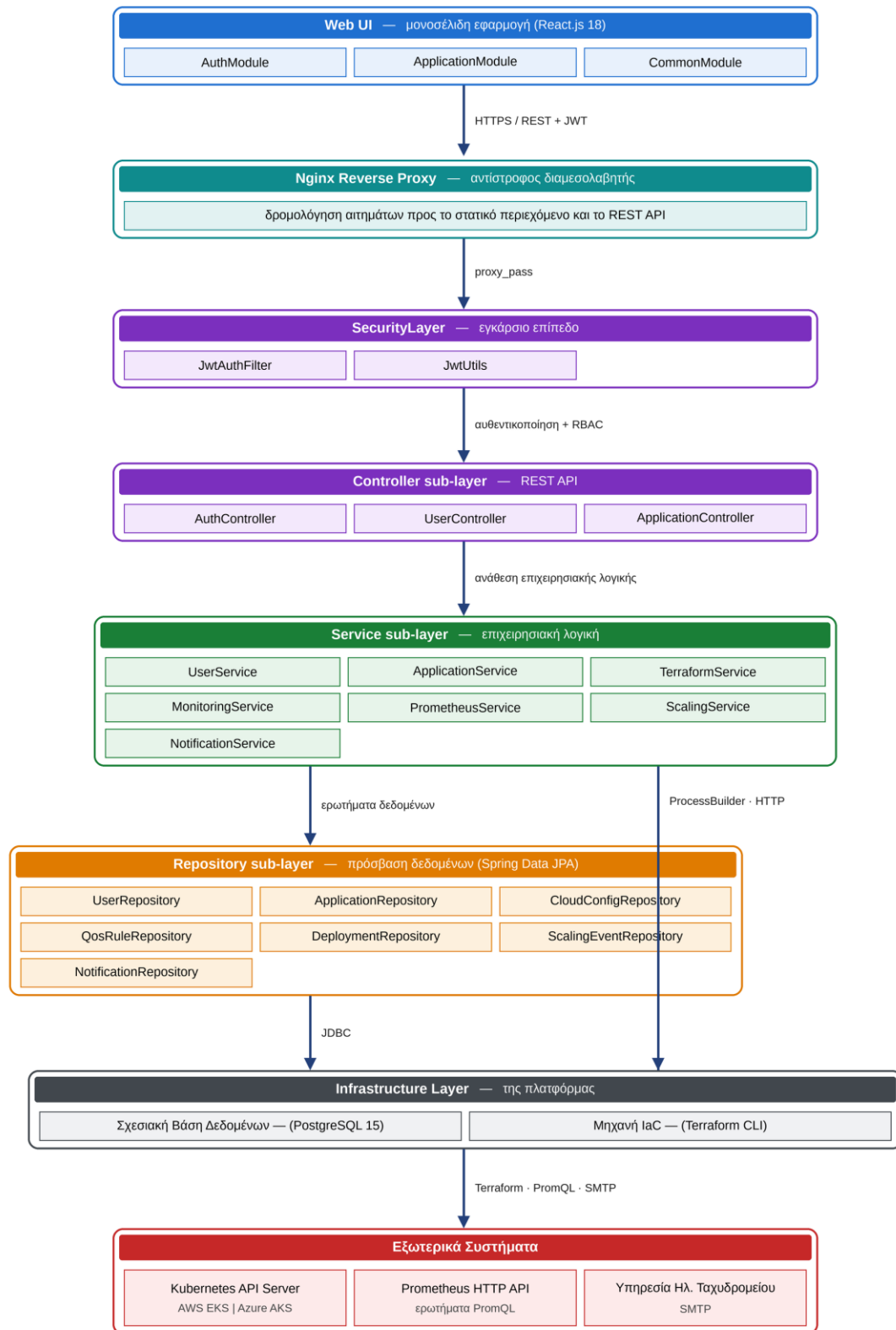
Οι εξωτερικές οντότητες με τις οποίες αλληλεπιδρά η πλατφόρμα είναι οι εξής:

- **Επισκέπτης.** Υποβάλλει αίτηση εγγραφής και λαμβάνει επιβεβαίωση δημιουργίας λογαριασμού σε κατάσταση αναμονής ενεργοποίησης (*pending activation*).
- **Κανονικός Χρήστης.** Εκτελεί CRUD και αναζήτηση επί των δικών του εφαρμογών, εκκινεί διάταξη και καταστροφή διάταξης, εκτελεί χειροκίνητη κλιμάκωση, ζητά μετρικές και διαχειρίζεται τον λογαριασμό του· λαμβάνει κατάσταση εφαρμογών, μετρικές, ειδοποιήσεις και στοιχεία λογαριασμού.
- **Διαχειριστής.** Διαθέτει τις ίδιες δυνατότητες με τον Κανονικό Χρήστη (κληρονομικότητα ρόλου) και επιπλέον διαχειρίζεται λογαριασμούς και το σύνολο των εφαρμογών του συστήματος.
- **Cloud (AWS / Azure).** Δέχεται κλήσεις μέσω Terraform για δημιουργία, ενημέρωση και καταστροφή συστάδων, κόμβων και δικτύωσης, και επιστρέφει cluster endpoints και κατάσταση πόρων.
- **Email Service (SMTP).** Δέχεται μηνύματα ειδοποίησης παραβιάσεων QoS και προσωρινούς κωδικούς επαναφοράς. Η επιστρεφόμενη πληροφορία περιορίζεται σε επιβεβαίωση αποδοχής του μηνύματος από τον SMTP διακομιστή, όχι σε επιβεβαίωση παράδοσης ή ανάγνωσης από τον παραλήπτη.

4.3.2 Διάγραμμα Συστατικών (Component Diagram)

Το διάγραμμα συστατικών αποδομεί την πλατφόρμα σε διακριτά επίπεδα και αποτυπώνει τις εξαρτήσεις μεταξύ τους. Οι ονομασίες των συστατικών είναι ταυτόσημες σε όλα τα διαγράμματα του κεφαλαίου και στη δομή του πηγαίου κώδικα (§4.4.2), σύμφωνα με την κανονική ονοματολογία του Πίνακα 5.2. Το backend ακολουθεί το αρχιτεκτονικό μοτίβο **Controller–Service–Repository (CSR)**, μια στρωματοποιημένη οργάνωση όπου κάθε στρώμα έχει σαφώς οριοθετημένη

ευθύνη: οι **controllers** εκθέτουν το REST API, επικυρώνουν τα δεδομένα εισόδου (@Valid) και επιβάλλουν τον έλεγχο πρόσβασης, χωρίς να περιέχουν επιχειρησιακή λογική· τα **services** συγκεντρώνουν το σύνολο της επιχειρησιακής λογικής και ορίζουν τα όρια των συναλλαγών· τα **repositories** ενθυλακώνουν αποκλειστικά την πρόσβαση στα δεδομένα. Κρίσιμη συνέπεια του μοτίβου είναι ότι **κανένας controller δεν επικοινωνεί απευθείας με τη βάση δεδομένων ή με εξωτερικό σύστημα**· κάθε τέτοια πρόσβαση δρομολογείται μέσω του αντίστοιχου service.



Εικόνα 4.2 — Διάγραμμα συστατικών της πλατφόρμας

Web UI — μονοσέλιδη εφαρμογή σε React.js 18 και TypeScript 5. Οργανώνεται σε τρία λειτουργικά δομοστοιχεία, τα οποία αποτυπώνονται στο διάγραμμα

συστατικών (Εικόνα 4.2): το **AuthModule** (σελίδες σύνδεσης και εγγραφής), το **ApplicationModule** (αρχική σελίδα και σελίδα εφαρμογών) και το **CommonModule** (κοινή γραμμή πλοήγησης, πελάτης HTTP και κατάσταση ταυτοποίησης). Υλοποιεί μονοσέλιδη εφαρμογή (*Single Page Application*) και επικοινωνεί με το *backend* αποκλειστικά μέσω του REST API. Αποτελείται από τα ακόλουθα δομοστοιχεία:

- **AuthModule**: φόρμες σύνδεσης και εγγραφής, διαχείριση του διακριτικού JWT.
- **ApplicationModule**: αρχική οθόνη και οθόνη εφαρμογών — δημιουργία, κατάλογος με ένδειξη κατάστασης και διαγραφή.
- **CommonModule**: κοινή γραμμή πλοήγησης, ενιαίο στιγμιότυπο Axios με αυτόματη έγχυση του JWT και κοινόχρηστη κατάσταση ταυτοποίησης (*Zustand*).

Ngix Reverse Proxy. Παρεμβάλλεται μεταξύ του προγράμματος περιήγησης και του *backend*: εξυπηρετεί τα στατικά αρχεία της SPA και προωθεί (*proxy_pass*) τα αιτήματα της διαδρομής */api* προς το *backend*. Έτσι η διεπαφή και το API εκτίθενται από ενιαία προέλευση (*same origin*), αίροντας την ανάγκη διαμόρφωσης CORS.

Backend Layer (Spring Boot 3.2, Java 17· VMware Tanzu, 2024). Το *SecurityLayer* παρεμβάλλεται πριν από κάθε *controller* και αναλαμβάνει την αυθεντικοποίηση (επαλήθευση JWT) και την εξουσιοδότηση (RBAC). Οι *controllers* εκθέτουν το API ανά πόρο· επειδή η επιχειρησιακή λογική βρίσκεται αποκλειστικά στα *services*, στη συνέχεια αναλύεται η λειτουργικότητα των τελευταίων:

- **UserService** — διαχειρίζεται τον κύκλο ζωής του λογαριασμού και της συνεδρίας. Επαληθεύει διαπιστευτήρια μέσω του *UserDetailsService* του Spring Security, εγγράφει νέους χρήστες με κατακερματισμό BCrypt του κωδικού και της απάντησης ασφαλείας, και ενεργοποιεί ή απενεργοποιεί λογαριασμούς κατόπιν ενέργειας διαχειριστή. Η έκδοση και η επαλήθευση του JWT γίνονται από τη βοηθητική κλάση *JwtUtils* του *SecurityLayer*.
- **ApplicationService** — διαχειρίζεται τον κύκλο ζωής της εφαρμογής ως λογικής οντότητας: δημιουργία με τις ρυθμίσεις νέφους και τους κανόνες QoS, τροποποίηση, αναζήτηση, μεταβάσεις κατάστασης (*INITIAL* → *DEPLOYING* → *DEPLOYED* → ...) και διαγραφή με προηγούμενη αποδιάταξη.
- **TerraformService** — αποτελεί τη γέφυρα προς την υποδομή. Για τη διάταξη μιας εφαρμογής βάσει συγκεκριμένης διαμόρφωσης νέφους δημιουργεί απομονωμένο *workspace*, παράγει τον κώδικα Terraform μέσω του *TemplateGenerator*, και εκτελεί σειριακά *terraform init* → *plan* → *apply* καλώντας το Terraform ως εξωτερική διεργασία (*ProcessBuilder*) με καταγραφή εξόδου σε πραγματικό χρόνο. Επιπλέον, αναλύει την έξοδο για να εξαγάγει το *cluster endpoint*, παρακολουθεί την κατάσταση της διάταξης και εκτελεί την καταστροφή της (*terraform destroy*) όποτε αυτή χρειαστεί.
- **MonitoringService** — ενορχηστρώνει τον μηχανισμό QoS. Μέσω *@Scheduled(fixedDelay = 30 s)* διατρέχει τις ενεργές διατάξεις, αντλεί τις τρέχουσες τιμές μετρικών μέσω του *PrometheusService*, τις συγκρίνει με τα όρια των κανόνων λαμβάνοντας υπόψη το *cooldown*, και σε παραβίαση αναθέτει την εκτέλεση της ενέργειας στο *ScalingService*. Εξυπηρετεί επίσης τα αιτήματα μετρικών του χρήστη (τρέχουσες και ιστορικές τιμές).
- **ScalingService** — εκτελεί την ενέργεια κλιμάκωσης: κλιμάκωση pods μέσω του *kubectl* (εντολή *kubectl scale* προς το Kubernetes API), ή κλιμάκωση κόμβων

μέσω του `TerraformService`· επιβάλλει τα όρια `minNodes/maxNodes` και καταγράφει το αποτέλεσμα ως `ScalingEvent`.

- **PrometheusService** — αποτελεί το σημείο επαφής με το Prometheus. Δέχεται από το `MonitoringService` το αναγνωριστικό της διάταξης και το φιλικό όνομα μιας μετρικής, τα μεταφράζει σε ερώτημα PromQL και εκτελεί κλήση HTTP προς το REST API του Prometheus της αντίστοιχης συστάδας (σημείο πρόσβασης `/api/v1/query`, όπως καταγράφεται στο πεδίο `prometheusEndpoint` της οντότητας `Deployment`), επιστρέφοντας αριθμητική τιμή ή κενό αποτέλεσμα σε περίπτωση μη διαθεσιμότητας. Υλοποιείται ως κλάση Java εντός του υπο-επιπέδου υπηρεσιών του backend· σε αντίθεση με την PostgreSQL και τη μηχανή Terraform, που αποτελούν αυτοτελείς διεργασίες του λειτουργικού συστήματος και τοποθετούνται στο επίπεδο υποδομής.
- **NotificationService** — αποθηκεύει ειδοποιήσεις εντός εφαρμογής στη βάση και αποστέλλει email μέσω του **Email Service**.

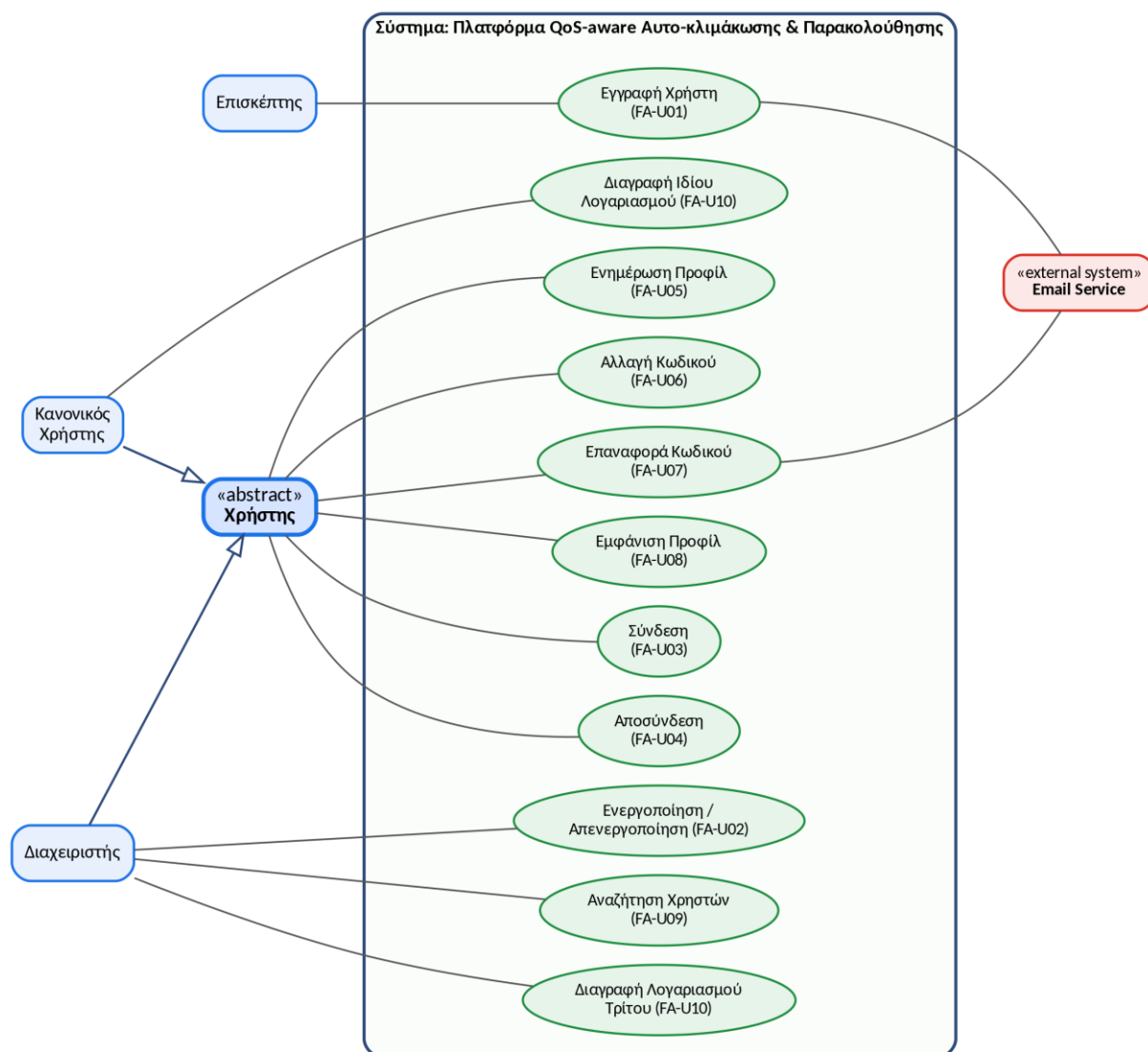
Infrastructure Layer. Περιλαμβάνει την **PostgreSQL 15** για τη μόνιμη αποθήκευση των δεδομένων της πλατφόρμας και τη μηχανή Terraform που εκτελεί τα σχέδια υποδομής.

Εξωτερικά συστήματα. Το **Cloud** φιλοξενεί τη συστάδα Kubernetes και τη στοίβα `kube-prometheus-stack` (Prometheus TSDB, Grafana, Alertmanager) που δημιουργεί η πλατφόρμα, ενώ το **Email Service** παραλαμβάνει τα εξερχόμενα μηνύματα. Επισημαίνεται ότι η ίδια η πλατφόρμα **δεν** εκτελείται εντός Kubernetes· διατάσσεται μέσω Docker Compose (βλ. §4.4.1 και Κεφάλαιο 5), ενώ το Kubernetes αποτελεί το *διαχειριζόμενο* περιβάλλον-στόχο και όχι περιβάλλον εκτέλεσης της πλατφόρμας.

4.3.3 Διαγράμματα Περιπτώσεων Χρήσης

Η λειτουργικότητα του συστήματος αποτυπώνεται σε δύο διαγράμματα περιπτώσεων χρήσης, ένα για τη διαχείριση χρηστών και ένα για τη διαχείριση εφαρμογών. Και στα δύο εισάγεται ο **αφηρημένος δρών «Χρήστης»**, από τον οποίο εξειδικεύονται (γενίκευση δρώντων) ο *Κανονικός Χρήστης* και ο *Διαχειριστής*. Η επιλογή αυτή είναι σημασιολογικά αναγκαία: στην UML, η σύνδεση δύο δρώντων με την **ίδια** περίπτωση χρήσης δηλώνει ότι ο ένας είναι εκκινήτης και ο άλλος υποστηρικτικός δρών, κάτι που δεν ισχύει εδώ, καθώς και οι δύο ρόλοι εκκινούν αυτοτελώς τις ίδιες λειτουργίες. Με τη γενίκευση, οι κοινές λειτουργίες συνδέονται μία φορά με τον αφηρημένο «Χρήστη» και **κληρονομούνται** και από τους δύο συγκεκριμένους ρόλους, ενώ οι αποκλειστικές λειτουργίες παραμένουν συνδεδεμένες στον αντίστοιχο (συγκεκριμένο) ρόλο.

Διάγραμμα Περιπτώσεων Χρήσης A — Διαχείριση Χρηστών. Ο *Επισκέπτης* εκκινεί μόνο την Εγγραφή. Ο αφηρημένος *Χρήστης* εκκινεί τις κοινές λειτουργίες λογαριασμού. Η Διαγραφή Ιδίου Λογαριασμού παραμένει αποκλειστικά στον Κανονικό Χρήστη, ο οποίος επιπλέον εκκινεί τη χειροκίνητη κλιμάκωση και τη διαχείριση του λογαριασμού του, ενώ η ενεργοποίηση/απενεργοποίηση, η αναζήτηση και η διαγραφή τρίτων λογαριασμών παραμένουν στον Διαχειριστή. Το εξωτερικό σύστημα *Email Service* συνδέεται ως υποστηρικτικός δρών με τις δύο περιπτώσεις χρήσης που όντως απαιτούν αποστολή μηνύματος — την Εγγραφή (email επιβεβαίωσης) και την Επαναφορά Κωδικού (προσωρινός κωδικός) — με κανονική (συνεχή) γραμμή συσχέτισης, καθώς η φύση της συσχέτισης προκύπτει από τον ίδιο τον δρώντα.



Εικόνα 4.3 — Διάγραμμα περιπτώσεων χρήσης A: Διαχείριση Χρηστών

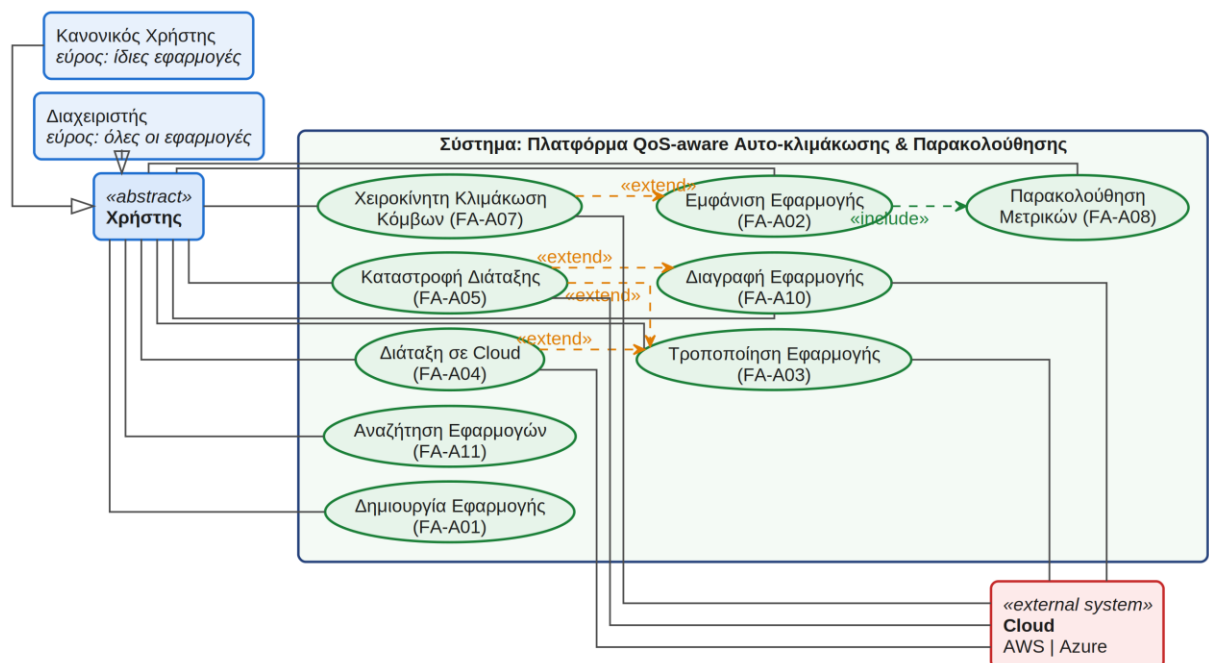
Διάγραμμα Περιπτώσεων Χρήσης Β — Διαχείριση Εφαρμογών. Όλες οι περιπτώσεις χρήσης συνδέονται με τον αφηρημένο *Χρήστη* και κληρονομούνται από τους δύο συγκεκριμένους ρόλους· η διαφοροποίηση Κανονικού Χρήστη και Διαχειριστή δεν αφορά το ποιες λειτουργίες είναι διαθέσιμες αλλά το *εύρος* των εφαρμογών επί των οποίων εφαρμόζονται (ίδιες εφαρμογές έναντι όλων των εφαρμογών του συστήματος), περιορισμός που επιβάλλεται σε επίπεδο εξουσιοδότησης πόρου και όχι μοντελοποίησης δρώντων. Το εξωτερικό σύστημα *Cloud* συνδέεται με τις περιπτώσεις χρήσης που πράγματι το επικαλούνται: Διάταξη σε Cloud, Καταστροφή Διάταξης, Χειροκίνητη Κλιμάκωση Κόμβων, Τροποποίηση Εφαρμογής και Διαγραφή Εφαρμογής.

Σχέσεις μεταξύ περιπτώσεων χρήσης. Η Παρακολούθηση Μετρικών εκτελείται πάντοτε στο πλαίσιο της Εμφάνισης Εφαρμογής και συνδέεται επομένως με σχέση «include» με φορά από την Εμφάνιση Εφαρμογής προς αυτήν: η περιλαμβάνουσα περίπτωση δείχνει την περιλαμβανόμενη. Αντιθέτως, η Χειροκίνητη Κλιμάκωση Κόμβων εκτελείται μόνο εφόσον το επιλέξει ο χρήστης και συνδέεται με σχέση

«extend» με φορά από αυτήν προς την Εμφάνιση Εφαρμογής, δηλαδή η επεκτείνουσα δείχνει την επεκτεινόμενη.

Ως προς τις σχέσεις μεταξύ περιπτώσεων χρήσης:

- Η **Καταστροφή Διάταξης** επεκτείνει («extend») τη **Διαγραφή Εφαρμογής**, διότι η διαγραφή προκαλεί καταστροφή της υποδομής μόνον υπό συνθήκη (εφόσον η εφαρμογή είναι διατεταγμένη).
- Η **Διάταξη σε Cloud** και η **Καταστροφή Διάταξης** επεκτείνουν την **Τροποποίηση Εφαρμογής**, διότι μια τροποποίηση των ρυθμίσεων νέφους συνεπάγεται υπό συνθήκη καταστροφή της υφιστάμενης διάταξης και εκ νέου διάταξη.
- Η Παρακολούθηση Μετρικών περιλαμβάνεται («include») στην Εμφάνιση Εφαρμογής: εκτελείται πάντοτε όταν ανοίγει η οθόνη λεπτομερειών και αποτελεί αναπόσπαστο μέρος της. Αντιθέτως, η Χειροκίνητη Κλιμάκωση Κόμβων την επεκτείνει («extend»), καθώς εκκινείται από την ίδια οθόνη αλλά μόνο εφόσον το επιλέξει ο χρήστης.



Εικόνα 4.4 — Διάγραμμα περιπτώσεων χρήσης Β: Διαχείριση Εφαρμογών

Επισημαίνεται ότι η **Λήψη Ειδοποιήσεων** (FA-A09) και η **Αυτόματη Κλιμάκωση QoS** (FA-A06) δεν αποτελούν περιπτώσεις χρήσης σε κανένα από τα δύο διαγράμματα, καθώς πρόκειται για πλήρως αυτοματοποιημένες διεργασίες που δεν εκκινούνται από κάποιον δρώντα. Λόγω της κεντρικότητάς τους για την πλατφόρμα, η δυναμική τους συμπεριφορά περιγράφεται παρ' όλα αυτά με διάγραμμα ακολουθίας στην §4.3.4 (SD-04), με ρητή διευκρίνιση του χαρακτήρα τους.

4.3.4 Διαγράμματα Ακολουθίας

Συμβάσεις απλοποίησης των διαγραμμάτων ακολουθίας. Για να διατηρηθεί η πολυπλοκότητα των διαγραμμάτων σε λογικά πλαίσια, υιοθετούνται δύο συμβάσεις που ισχύουν σε όλα ανεξαιρέτως: (α) η αλληλεπίδραση μεταξύ των repositories και της βάσης δεδομένων δεν αποτυπώνεται, καθώς είναι ομοιόμορφη και προβλέψιμη σε κάθε ροή· η βάση θεωρείται ότι βρίσκεται πίσω από κάθε κλήση repository και

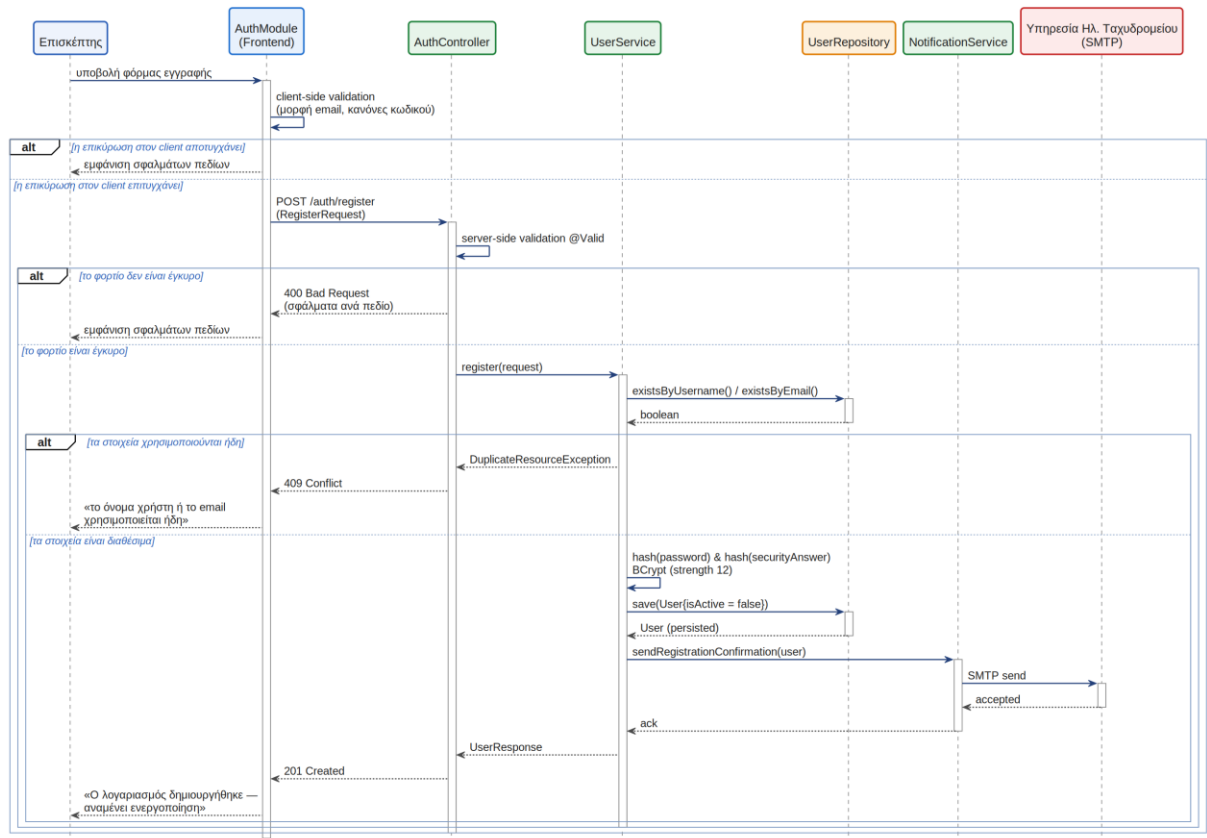
εμφανίζεται ως διακριτό συστατικό μόνο στο διάγραμμα συστατικών (Εικόνα 4.2)· και (β) ο δρων «Χρήστης» δεν φέρει μπάρα ενεργοποίησης, διότι δεν αποτελεί εκτελέσιμο συστατικό του συστήματος αλλά εξωτερικό διεγέρτη των ροών. Οι υπόλοιπες γραμμές ζωής φέρουν κανονικά μπάρες ενεργοποίησης, και οι εσωτερικές ενέργειες κάθε συστατικού αποδίδονται ως αυτο-αλληλεπιδράσεις.

Παρουσιάζονται έξι διαγράμματα ακολουθίας που καλύπτουν τις πλέον αντιπροσωπευτικές ροές του συστήματος. Κάθε διάγραμμα συνοδεύεται από περιγραφή της ροής εντός του κυρίου κειμένου. Σε όλα τα διαγράμματα τηρούνται συστηματικά οι ακόλουθες συμβάσεις UML: (α) κάθε γραμμή ζωής φέρει **μπάρες ενεργοποίησης** (*activation bars*) που εκτείνονται από την αποστολή ενός μηνύματος έως τη λήψη της αντίστοιχης απάντησης, (β) οι εσωτερικές ενέργειες ενός συστατικού αποδίδονται ως **αυτο-αλληλεπιδράσεις** (βέλη που εκκινούν και καταλήγουν στην ίδια γραμμή ζωής), (γ) χρησιμοποιείται `alt` όταν υπάρχουν δύο ή περισσότερες αμοιβαία αποκλειόμενες διαδρομές, `opt` όταν πρόκειται για απλή υπό συνθήκη ενέργεια χωρίς εναλλακτική, και `loop` για επαναλήψεις, (δ) όλα τα βήματα που εξαρτώνται από μια συνθήκη τοποθετούνται **εντός** του αντίστοιχου τμήματος και όχι εκτός αυτού, και (ε) τα ονόματα των συστατικών ταυτίζονται με εκείνα του διαγράμματος συστατικών (§4.3.2).

Αντιστοίχιση γραμμών ζωής με τα συστατικά της υλοποίησης. Για λόγους αναγνωσιμότητας τα διαγράμματα ακολουθίας ομαδοποιούν ορισμένα συστατικά ανά λειτουργία και όχι ανά αρχείο πηγαίου κώδικα. Οι αντιστοιχίσεις είναι οι εξής: όπου στα διαγράμματα εμφανίζονται γραμμές ζωής «AuthModule (Frontend)» ή «ApplicationModule (Frontend)», νοείται το Web UI ως ενιαίο συστατικό, με ένδειξη του δομοστοιχείου που ενεργοποιείται· όπου στα διαγράμματα εμφανίζεται γραμμή ζωής «DeploymentController», νοείται ο ApplicationController, ο οποίος εκθέτει τα σημεία πρόσβασης διάταξης ως υποπόρους (Πίνακας 4.8)· η ένδειξη «ανάκτηση CloudCredential» αποδίδει την ανάγνωση των διαπιστευτηρίων του παρόχου, τα οποία στην υλοποίηση παρέχονται ως μεταβλητές περιβάλλοντος και όχι ως οντότητα της βάσης (§4.3.5). Τέλος, το βήμα εγκατάστασης της στοίβας παρακολούθησης μέσω Helm αποτυπώνει τη σχεδιαζόμενη ροή· στην παρούσα έκδοση η στοίβα εγκαθίσταται χωριστά κατά την προετοιμασία του περιβάλλοντος επίδειξης (§5.2.1).

SD-01: Εγγραφή Χρήστη (FA-U01).

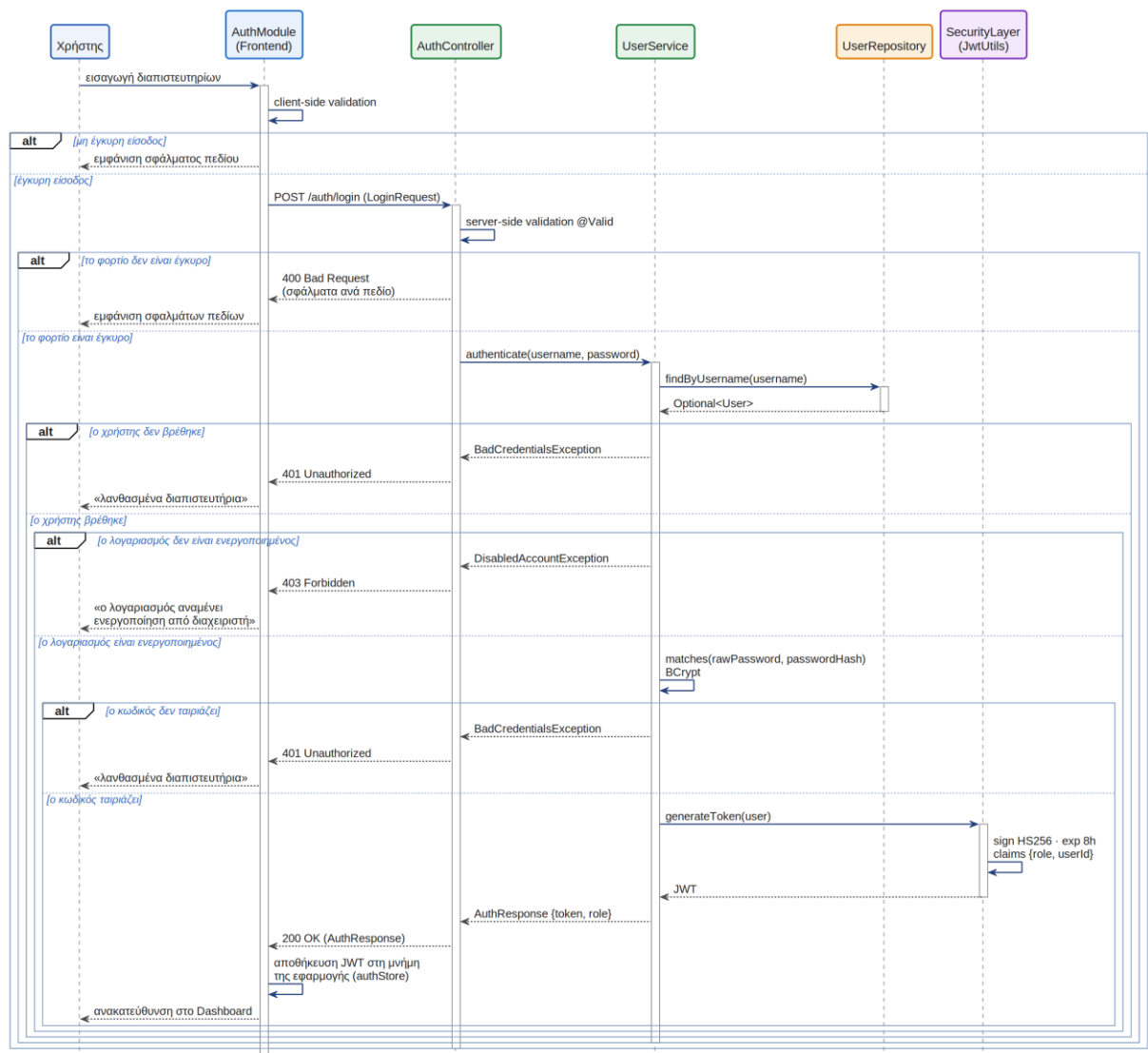
Η ροή ξεκινά με την υποβολή της φόρμας εγγραφής. Το `AuthModule` εκτελεί πρώτα επικύρωση στην πλευρά του πελάτη (αυτο-αλληλεπίδραση) και, εφόσον αυτή αποτύχει, εμφανίζει σφάλματα χωρίς να προσπελάσει το backend. Σε επιτυχή επικύρωση αποστέλλεται το αίτημα στον `AuthController`, ο οποίος εκτελεί επικύρωση στην πλευρά του διακομιστή μέσω `@Valid`. Ακολούθως το `UserService` ελέγχει τη μοναδικότητα ονόματος χρήστη και email, κατακερματίζει τον κωδικό με BCrypt (αυτο-αλληλεπίδραση) και αποθηκεύει τον νέο λογαριασμό σε κατάσταση αδρανοποίησης. Μόλις ολοκληρωθεί η αποθήκευση, το `NotificationService` δρομολογεί το email επιβεβαίωσης προς τον `Email Server` και μόνο κατόπιν επιστρέφεται η απάντηση προς τον χρήστη.



Εικόνα 4.5 — Διάγραμμα ακολουθίας: Εγγραφή Χρήστη

SD-02: Σύνδεση Χρήστη (FA-U03).

Το `AuthModule` επικυρώνει πρώτα τη συμπλήρωση και τη μορφή των διαπιστευτηρίων. Το αίτημα διέρχεται από τον `AuthController`, ο οποίος επικυρώνει το φορτίο και αναθέτει τη λογική στο `UserService` — τηρώντας την αρχή ότι ο controller δεν προσπελαύνει τη βάση δεδομένων. Το `UserService` ανακτά τον χρήστη μέσω του `UserRepository`, ελέγχει πρώτα αν ο λογαριασμός είναι ενεργοποιημένος και, μόνο εφόσον είναι, συγκρίνει τον κατακερματισμό του κωδικού ως αυτο-αλληλεπίδραση. Κάθε αρνητική έκβαση —αποτυχία επικύρωσης στον διακομιστή (400), ανενεργός λογαριασμός (403) ή εσφαλμένα διαπιστευτήρια (401)— επιστρέφεται μέχρι τον χρήστη με διακριτό μήνυμα. Μόνο σε επιτυχή έκβαση παράγεται το JWT μέσω του `JwtUtils`, εσωτερικής βοηθητικής κλάσης του `SecurityLayer` και όχι αυτοτελούς συστατικού. Ο controller λαμβάνει το αποτέλεσμα και το επιστρέφει στον πελάτη. Το frontend αποθηκεύει το token στη μνήμη της εφαρμογής (αυτο-αλληλεπίδραση).

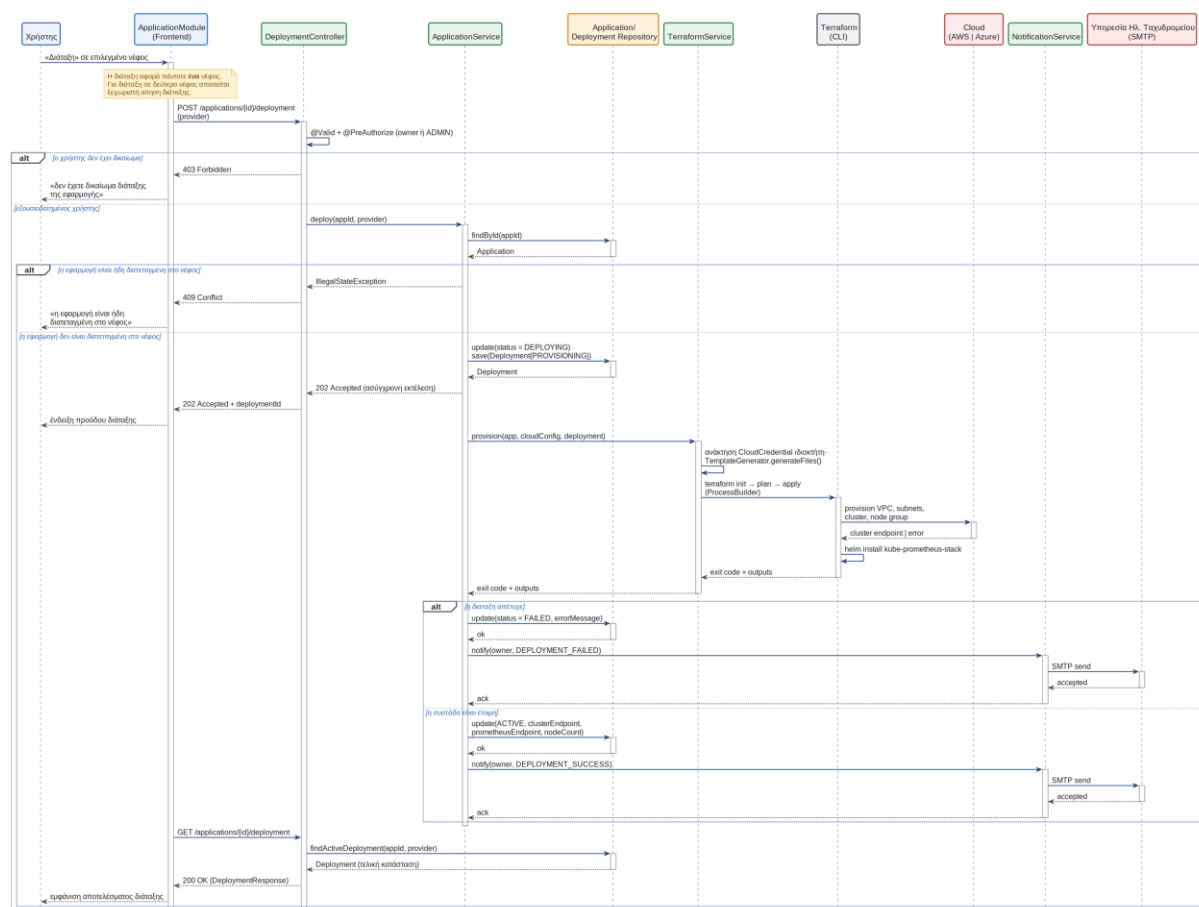


Εικόνα 4.6 — Διάγραμμα ακολουθίας: Σύνδεση Χρήστη

SD-03: Διάταξη Εφαρμογής (FA-A04).

Ο ApplicationController επικυρώνει το αίτημα και αναθέτει τη ροή στο ApplicationService, το οποίο ελέγχει ότι η εφαρμογή δεν είναι ήδη διατεταγμένη και ενημερώνει την κατάστασή της σε **DEPLOYING** μέσω του αντίστοιχου repository· ο controller δεν «αγγίζει» τη βάση. Το αποτέλεσμα της μηχανής Terraform (κωδικός εξόδου και outputs) επιστρέφει στο ApplicationService, το οποίο ενημερώνει την οντότητα Deployment μέσω του repository και, σε αμφότερες τις εκβάσεις —επιτυχία ή αποτυχία—, ειδοποιεί τον ιδιοκτήτη μέσω του NotificationService. Η τελική κατάσταση ανακατάται ρητά από τον χρήστη με αίτημα GET προς τον ελεγκτή, ο οποίος την αντλεί από το repository. Επισημαίνεται ότι κάθε διάταξη αφορά πάντοτε ένα και μόνο νέφος: εφόσον ο χρήστης επιθυμεί την ίδια εφαρμογή σε δύο παρόχους, εκκινεί δύο διακριτές διατάξεις με αντίστοιχες διαμορφώσεις. Στη συνέχεια το TerraformService ανακτά τα διαπιστευτήρια νέφους του ιδιοκτήτη, παράγει τον κώδικα Terraform και καλεί τη μηχανή **Terraform**, η οποία είναι το μόνο συστατικό που επικοινωνεί με το **Cloud** — το TerraformService δεν απευθύνεται ποτέ απευθείας στον πάροχο. Η εγγραφή **Deployment** δημιουργείται κατά την έναρξη της διάταξης με κατάσταση **PROVISIONING** και ενημερώνεται σε **ACTIVE** ή **FAILED** ανάλογα με την έκβαση.

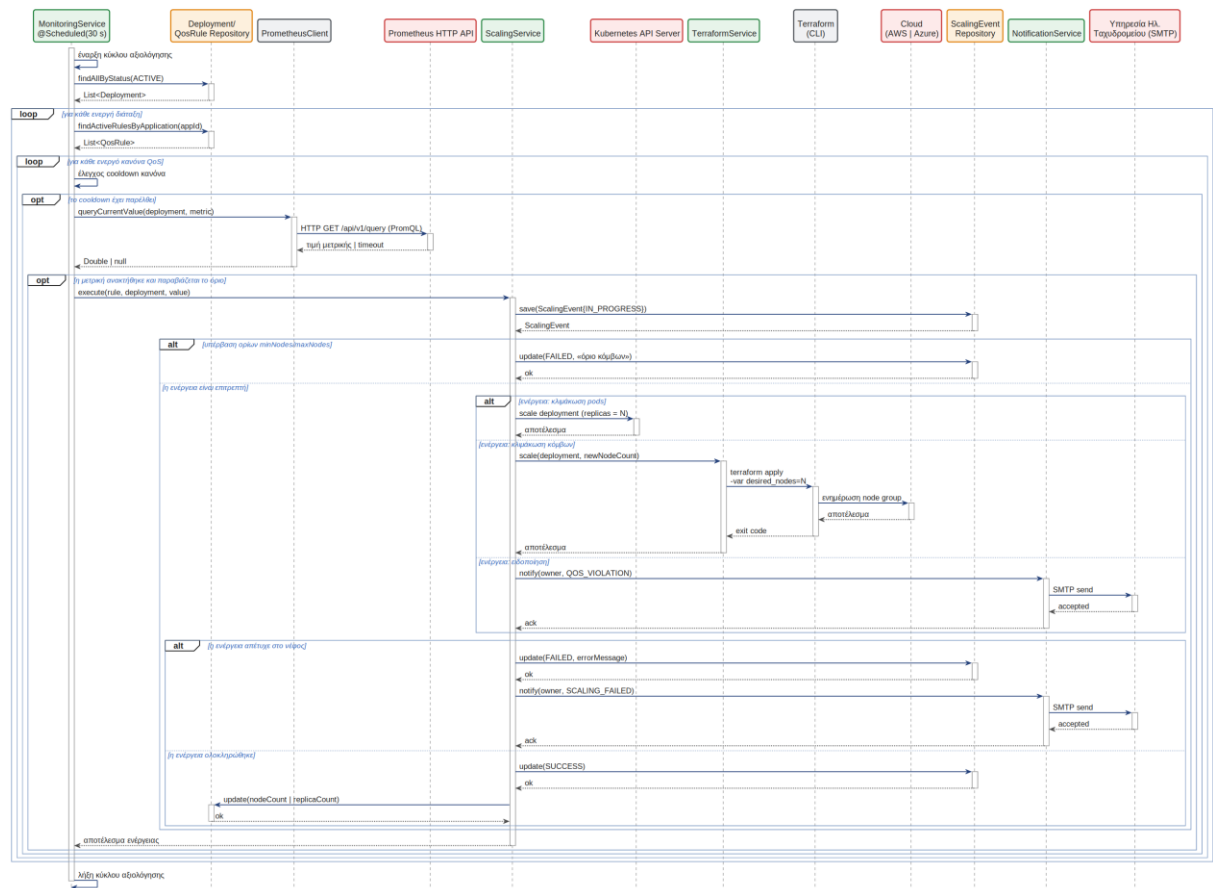
καλύπτεται ρητά και η αρνητική περίπτωση αποτυχίας παροχής της συστάδας. Επισημαίνεται ότι, στην παρούσα έκδοση, η αυτοματοποίηση καλύπτει τη συστάδα, τη δικτύωση και τη στοίβα παρακολούθησης· η εφαρμογή του χρήστη εγκαθίσταται στη συνέχεια χειροκίνητα (βλ. §4.7.3, FA-A04).



Εικόνα 4.7 — Διάγραμμα ακολουθίας: Διάταξη Εφαρμογής

SD-04: Αυτόματη Κλιμάκωση βάσει Κανόνων QoS (FA-A06).

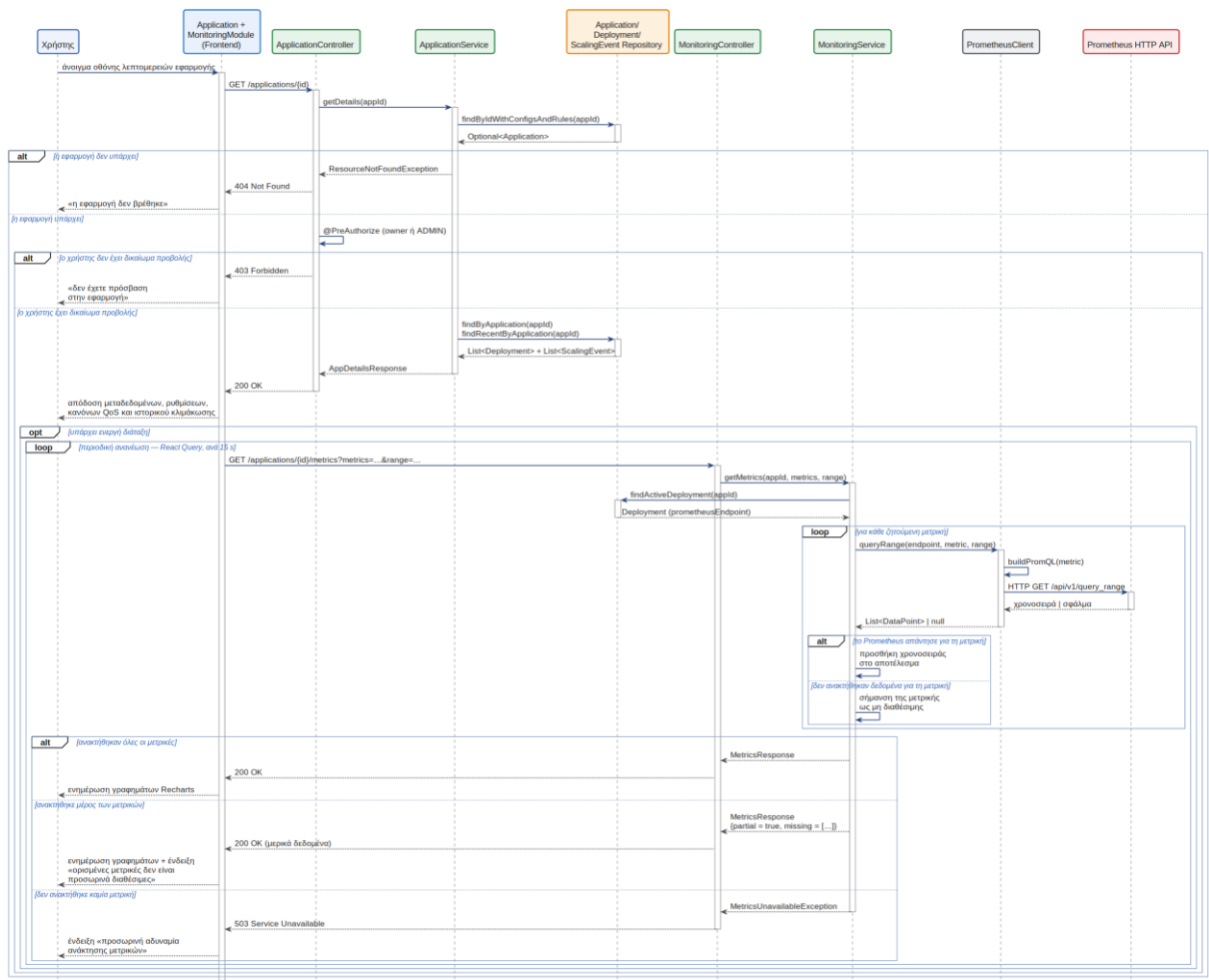
Διευκρινίζεται εξ αρχής ότι το παρόν διάγραμμα **δεν αποτυπώνει περίπτωση χρήσης**, καθώς η διαδικασία δεν εκκινείται από κάποιον δρώντα· περιγράφει μια θεμελιώδη αυτοματοποιημένη λειτουργία του συστήματος, η οποία παρουσιάζεται λόγω της κεντρικότητάς της για τη συνεισφορά της εργασίας. Η ενορχήστρωση ανατίθεται στο [MonitoringService](#), το οποίο ενεργοποιείται περιοδικά μέσω `@Scheduled(fixedDelay = 30 s)` — δεν υφίσταται αυτοτελές συστατικό «Scheduler». Η επανάληψη επί των ενεργών διατάξεων αποδίδεται με δομικό στοιχείο `loop`. Επειδή σε περίπτωση μη παραβίασης κανόνα δεν εκτελείται καμία ενέργεια, χρησιμοποιείται `opt` και όχι `alt`. Η ενέργεια κλιμάκωσης ανατίθεται στο [ScalingService](#), το οποίο για κλιμάκωση κόμβων απευθύνεται στο [TerraformService](#) και, μέσω αυτού, στη μηχανή **Terraform** και τελικά στο **Cloud**. Καλύπτονται ρητά οι αρνητικές περιπτώσεις: αποτυχία της ενέργειας στο νέφος και προσπάθεια υπέρβασης των ορίων `minNodes/maxNodes`.



Εικόνα 4.8 — Διάγραμμα ακολουθίας: Αυτόματη κλιμάκωση βάσει κανόνων QoS

SD-05: Εμφάνιση Εφαρμογής και Παρακολούθηση Μετρικών (FA-A02 / FA-A08).

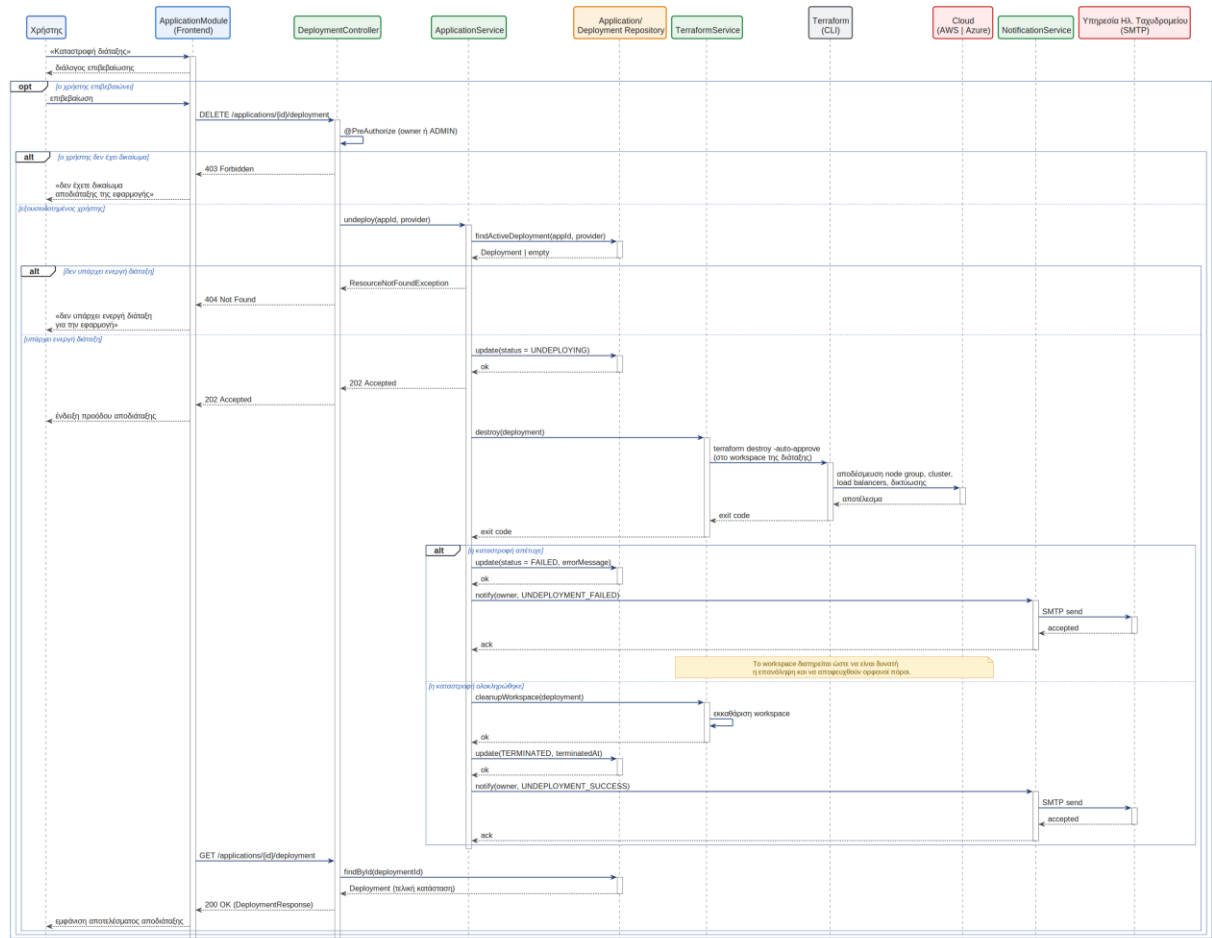
Η παρακολούθηση μετρικών δεν συνιστά αυτοτελή περίπτωση χρήσης· αποτελεί τμήμα της Εμφάνισης Εφαρμογής (FA-A02), σχέση που αποτυπώνεται ως «include» στην Εικόνα 4.4. Το διάγραμμα καλύπτει επομένως την υπερκείμενη περίπτωση χρήσης στο σύνολό της και εμπλέκει τον `ApplicationController`: αυτός αντλεί τα στοιχεία της εφαρμογής (μεταδεδομένα, ρυθμίσεις νέφους, κανόνες QoS, ιστορικό κλιμάκωσης), ενώ το `PrometheusService` αντλεί τις τιμές των μετρικών. Η σειρά είναι ουσιώδης: τα στοιχεία της εφαρμογής ανακτώνται **πρώτα** από τη βάση, διότι από αυτά προκύπτουν τόσο το `prometheusEndpoint` της διάταξης όσο και ο κατάλογος των μετρικών που ενδιαφέρουν τον χρήστη, οι οποίες στη συνέχεια αντλούνται από το Prometheus. Ως προς τη διαθεσιμότητα των μετρικών (MFA-R02) διακρίνονται τρεις εκβάσεις: όταν όλες οι μετρικές ανακτηθούν, επιστρέφεται πλήρης απόκριση· όταν ανακτηθούν μερικές, η απόκριση συνοδεύεται από ένδειξη μερικότητας και η οθόνη αποδίδεται με μειωμένη λειτουργικότητα αντί να αποτύχει· και όταν δεν ανακτηθεί καμία, επιστρέφεται 503 Service Unavailable, ώστε ο χρήστης να διακρίνει την αδυναμία συλλογής από μηδενικές τιμές. Ο έλεγχος απόκρισης εκτελείται ανά μετρική, εντός του βρόχου συλλογής.



Εικόνα 4.9 — Διάγραμμα ακολουθίας: Εμφάνιση εφαρμογής και παρακολούθηση μετρικών

SD-06: Καταστροφή Διάταξης Εφαρμογής (FA-A05).

Στη ροή διακρίνονται σαφώς η βάση δεδομένων ως εσωτερικό συστατικό της πλατφόρμας (μέσω των repositories) και το νέφος ως εξωτερικό σύστημα. Παρεμβάλλεται η μηχανή **Terraform** μεταξύ του **TerraformService** και του νέφους, ενώ καλύπτεται ρητά η αρνητική περίπτωση αποτυχίας της αποδιάταξης (καταστροφής διάταξης): η διάταξη παραμένει σε κατάσταση **FAILED** με διατήρηση του workspace, ώστε να είναι δυνατή η χειροκίνητη ή αυτόματη επανάληψη και να αποφευχθεί η δημιουργία ορφανών πόρων.



Εικόνα 4.10 — Διάγραμμα ακολουθίας: Καταστροφή διάταξης εφαρμογής

4.3.5 Διάγραμμα Κλάσεων

Το διάγραμμα κλάσεων αποτυπώνει το **μοντέλο πεδίου** (domain model) της πλατφόρμας σε επίπεδο σχεδίασης, όχι το φυσικό σχεσιακό σχήμα. Ακολουθούνται συνεπώς οι εξής αρχές μοντελοποίησης, οι οποίες διορθώνουν αντίστοιχες αποκλίσεις προγενέστερης έκδοσης:

- Δεν παρατίθενται πρωτεύοντα ή ξένα κλειδιά (PK/FK), καθώς πρόκειται για έννοιες του φυσικού επιπέδου αποθήκευσης και όχι του μοντέλου κλάσεων.
- Δεν περιλαμβάνονται αναγνωριστικά ή αντικείμενα άλλων κλάσεων ως πεδία (π.χ. `ownerId`, `appId`)· η αντίστοιχη πληροφορία εκφράζεται αποκλειστικά μέσω **ονοματισμένων συσχετίσεων** με ρητή πολλαπλότητα και στα δύο άκρα.
- Η πλήρης, αυστηρή σχεσιακή σηματοδότηση με τους τύπους δεδομένων, τους περιορισμούς ακεραιότητας και τα ευρετήρια παρατίθεται στο **Παράρτημα Ε** (Flyway migrations).
- Οι σημειώσεις σχεδίασης παρατίθενται στο κύριο κείμενο και όχι εντός του σχήματος, ώστε αυτό να παραμένει συμπαγές και ευανάγνωστο.
- Οι συσχετίσεις σχεδιάζονται με ορθογώνια (rectangular) δρομολόγηση ώστε να μη διέρχονται μέσα από τα ορθογώνια των κλάσεων.

Ο Πίνακας 4.6 παραθέτει τις επτά κλάσεις του μοντέλου με τα χαρακτηριστικά και τις συσχετίσεις τους, όπως αυτές υλοποιούνται στο πακέτο `model/entity` (§4.4.2). Καταγράφονται μόνο οι συσχετίσεις με τη φορά που αποτυπώνεται στο διάγραμμα

κλάσεων (Εικόνα 4.11)· οι αντίστροφες όψεις τους δεν παρατίθενται. Τα διαπιστευτήρια πρόσβασης στους παρόχους νέφους δεν αποτελούν κλάση του μοντέλου: παρέχονται στην πλατφόρμα ως μεταβλητές περιβάλλοντος κατά την εκκίνηση και δεν αποθηκεύονται στη βάση δεδομένων, ώστε να μην υπάρχει μόνιμη διατήρηση μυστικών.

Διευκρίνιση ως προς το περιεχόμενο του πίνακα: η CloudConfig περιγράφει την επιθυμητή διαμόρφωση υποδομής μιας εφαρμογής σε συγκεκριμένο νέφος (τύπος instance, εύρος κόμβων, περιοχή, αντιστοιχίσεις θυρών) και ανήκει στην εφαρμογή, ενώ τα διαπιστευτήρια πρόσβασης στον πάροχο παρέχονται εκτός μοντέλου, ως μεταβλητές περιβάλλοντος. Τα δύο μεγέθη έχουν διαφορετικό εύρος ισχύος, διαφορετικό κύκλο ζωής και διαφορετικές απαιτήσεις ασφαλείας (η δεύτερη αποθηκεύεται κρυπτογραφημένα)· η ενοποίησή τους θα δημιουργούσε αδικαιολόγητη σύζευξη.

Πίνακας 4.6 — Κλάσεις του Μοντέλου Πεδίου

Κλάση	Χαρακτηριστικά	Συσχετίσεις
User	username {unique}, fullName, email {unique}, passwordHash, role : UserRole, isActive, securityQuestion, securityAnswerHash, createdAt, updatedAt	κατέχει → Application (1 : 0..*) · παραλαμβάνει → Notification (1 : 0..*)
Application	title, description, version, tags : List<String>, status : AppStatus, createdAt, updatedAt	ρυθμίζεται από → CloudConfig (1 : 0..*) · διέπεται από → QosRule (1 : 0..*) · διατάσσεται ως → Deployment (1 : 0..*) · καταγράφει → ScalingEvent (1 : 0..*) · αφορά → Notification (1 : 0..*)
CloudConfig	cloudProvider : CloudProvider, instanceType, minNodes, maxNodes, desiredNodes, region, portMappings	—
QosRule	metricName, operator, thresholdValue, durationSeconds, actionType : ActionType, actionValue, actionTarget, cooldownSeconds, isActive	παράγει → ScalingEvent (1 : 0..*)
Deployment	cloudProvider : CloudProvider, clusterEndpoint, prometheusEndpoint, nodeCount, status : DeploymentStatus, terraformWorkspace, deployedAt, terminatedAt	—
ScalingEvent	triggeredAt, metricValue, actionType : ActionType, actionStatus :	—

Κλάση	Χαρακτηριστικά	Συσχετίσεις
	ScalingEventStatus, errorMessage	
Notification	type, title, message, isRead, createdAt	—

Σημασιολογία κλάσεων και συσχετίσεων.

Η **`User`** αναπαριστά κάθε εγγεγραμμένο χρήστη. Το `username` και το `email` αποτελούν μοναδικά αναγνωριστικά επικοινωνίας και ταυτοποίησης· το `fullName` χρησιμοποιείται στην παρουσίαση και στα εξερχόμενα μηνύματα· το `passwordHash` αποθηκεύει τον κατακερματισμό BCrypt του κωδικού και ουδέποτε τον ίδιο τον κωδικό· ο `role` καθορίζει τα δικαιώματα RBAC· το `isActive` (αν ο χρήστης είναι ενεργός) αποτυπώνει την έγκριση του λογαριασμού από διαχειριστή και ελέγχεται σε κάθε προσπάθεια σύνδεσης· τα `securityQuestion` και `securityAnswerHash` υποστηρίζουν την επαναφορά κωδικού (FA-U07), με την απάντηση επίσης κατακερματισμένη· τα `createdAt` και `updatedAt` παρέχουν ιχνηλασιμότητα. Κάθε χρήστης κατέχει 0..* εφαρμογές, εξ ου και η κατεύθυνση της συσχέτισης κατέχει.

Η **`Application`** είναι η κεντρική κλάση-ρίζα του μοντέλου. Ο `title` και η `description` προσδιορίζουν τον τίτλο και μια κειμενική περιγραφή την εφαρμογής του χρήστη· η `version` επιτρέπει τη διάκριση εκδόσεων του ίδιου φορτίου εργασίας· ο `status` παρακολουθεί τη μετάβαση στον κύκλο ζωής (INITIAL → DEPLOYING → DEPLOYED → UNDEPLOYING → TERMINATED, με FAILED ως κατάσταση σφάλματος)· τα `createdAt/updatedAt` υποστηρίζουν την ταξινόμηση και την ιχνηλασιμότητα. Η συσχέτιση προς `CloudConfig` έχει πολλαπλότητα 1 : 1..* διότι μια εφαρμογή μπορεί να διαταχθεί ταυτόχρονα σε περισσότερα του ενός νέφη, με ανεξάρτητη διαμόρφωση ανά νέφος.

Η **`CloudConfig`** εκφράζει την επιθυμητή διαμόρφωση υποδομής. Ο `cloudProvider` επιλέγει το Terraform module που θα χρησιμοποιηθεί (δηλ. ανάλογα τον πάροχο που επιλέγεται)· το `instanceType` (τύπος στιγμιοτύπων εικονικής μηχανής) καθορίζει το μέγεθος των κόμβων· τα `minNodes/maxNodes` οριοθετούν το επιτρεπτό εύρος κλιμάκωσης (και επιβάλλονται τόσο στην αυτόματη όσο και στη χειροκίνητη κλιμάκωση) ενώ το `desiredNodes` ορίζει την αρχική κατάσταση· η `region` καθορίζει τη γεωγραφική τοποθεσία (διάταξης)· τα `portMappings` δηλώνουν τις θύρες που πρέπει να εκτεθούν από το Kubernetes Service. Η συσχέτιση προς `Deployment` είναι 1 : 0..* και όχι 1 : 1: μια διαμόρφωση παραμένει σταθερή στον χρόνο ενώ μπορεί να υλοποιηθεί από διαδοχικές διατάξεις (π.χ. μετά από καταστροφή και εκ νέου διάταξη), οπότε το ιστορικό διατάξεων ανά διαμόρφωση διατηρείται.

Τα διαπιστευτήρια πρόσβασης στους παρόχους νέφους (AWS_ACCESS_KEY_ID, AZURE_CLIENT_SECRET κ.λπ.) δεν μοντελοποιούνται ως κλάση. Εγχέονται στη διεργασία του backend ως μεταβλητές περιβάλλοντος μέσω του αρχείου .env και του Docker Compose, και καταναλώνονται απευθείας από το Terraform κατά την εκτέλεση. Η επιλογή αυτή αποφεύγει τη μόνιμη αποθήκευση μυστικών στη βάση και τη συνακόλουθη ανάγκη διαχείρισης κλειδιών κρυπτογράφησης· το μειονέκτημα είναι ότι τα διαπιστευτήρια είναι κοινά για όλους τους χρήστες της εγκατάστασης. Η ανά χρήστη διαχείριση μέσω HashiCorp Vault προτείνεται στην §6.2.4. Το `createdAt` επιτρέπει πολιτικές περιοδικής ανανέωσης. Ο συνδυασμός χρήστη και παρόχου είναι μοναδικός, ώστε κατά τη διάταξη η πλατφόρμα να εντοπίζει μονοσήμαντα το κατάλληλο διαπιστευτήριο του ιδιοκτήτη της εφαρμογής.

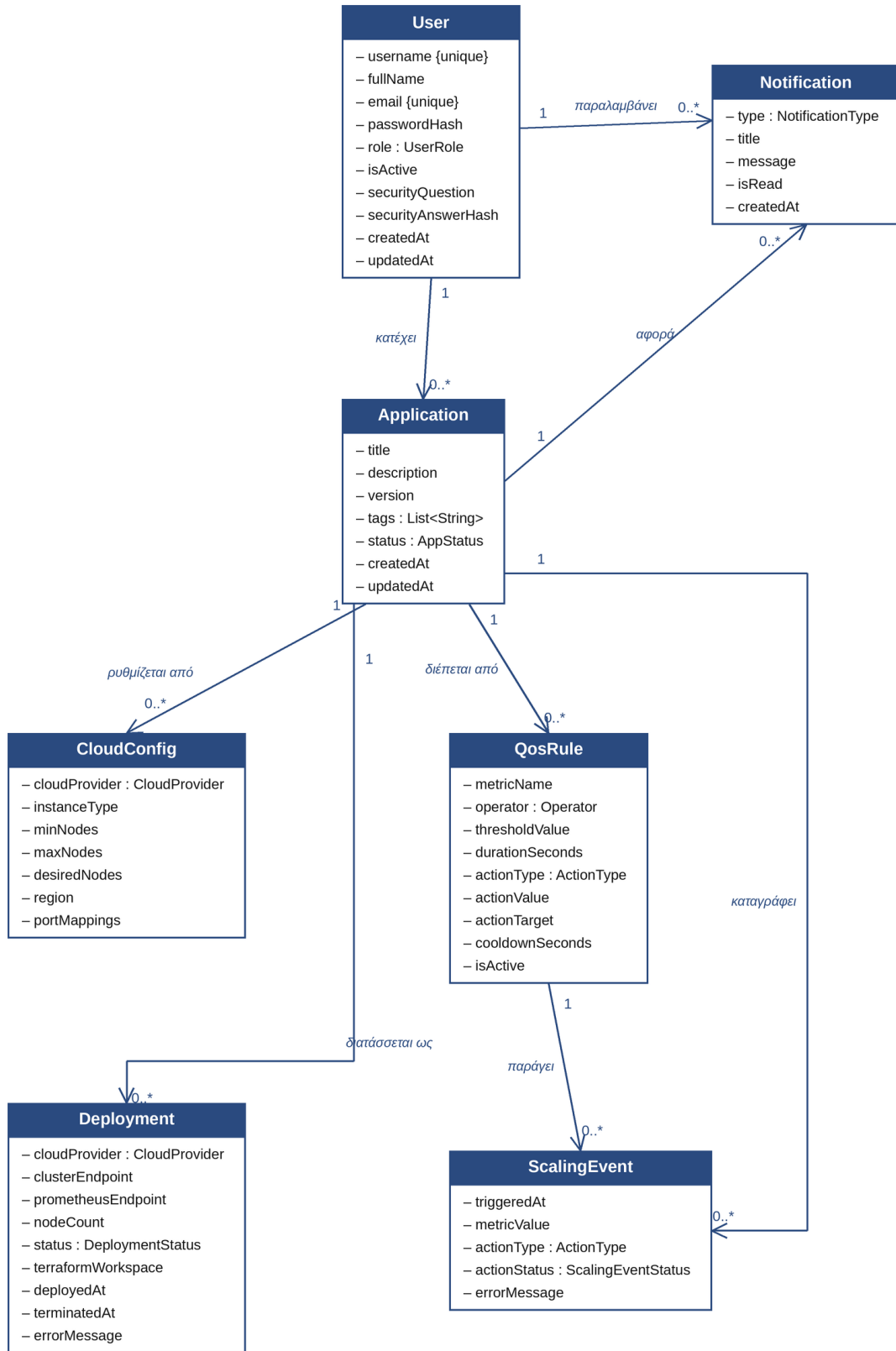
Η **`QosRule`** αναπαριστά έναν κανόνα αυτόματης κλιμάκωσης. Το `metricName` δηλώνει τη μετρική προς παρακολούθηση· ο `operator` και το `thresholdValue` συνθέτουν τη συνθήκη παραβίασης· ο `actionType` και το `actionValue` καθορίζουν την ενέργεια και το μέγεθός της (αν έχουμε αλλαγή κόμβων ή στιγμιοτύπων pods)· το `cooldownSeconds` αποτρέπει ταλαντώσεις (*flapping*) εμποδίζοντας διαδοχικές ενεργοποιήσεις εντός σύντομου διαστήματος· το `isActive` επιτρέπει προσωρινή απενεργοποίηση κανόνα χωρίς διαγραφή. Η συσχέτιση προς `ScalingEvent` (1 : 0..*) δικαιολογείται από το ότι ένας κανόνας ενεργοποιείται επανειλημμένα στον χρόνο.

Η **`Deployment`** αναπαριστά μια πραγματική (τρέχουσα ή ιστορική) διάταξη υποδομής. Το `clusterEndpoint` και το `prometheusEndpoint` είναι τα σημεία πρόσβασης που επιστρέφει το Terraform και χρησιμοποιούνται από την πλατφόρμα για διαχείριση και άντληση μετρικών αντίστοιχα· το `nodeCount` αποτυπώνει την τρέχουσα πραγματική χωρητικότητα· ο `status` παρακολουθεί την πορεία διάταξης· το `terraformWorkspace` επιτρέπει την επαναχρησιμοποίηση του ίδιου workspace για μεταγενέστερη κλιμάκωση ή καταστροφή· τα `deployedAt/terminatedAt` οριοθετούν τη διάρκεια ζωής και το `errorMessage` τεκμηριώνει αποτυχίες. Η διάκριση από την `CloudConfig` είναι ουσιώδης: η πρώτη εκφράζει την **επιθυμητή** κατάσταση, η δεύτερη την **πραγματική**.

Η **`ScalingEvent`** αποτελεί εγγραφή ελέγχου. Το `triggeredAt` και το `metricValue` τεκμηριώνουν πότε και υπό ποια συνθήκη ενεργοποιήθηκε ο κανόνας· ο `actionType` και ο `actionStatus` καταγράφουν την ενέργεια και την έκβασή της· το `errorMessage` τεκμηριώνει τυχόν αποτυχία. Η σύνδεσή της τόσο με την `QosRule` όσο και με το `Deployment` επιτρέπει την ανάλυση της συμπεριφοράς κλιμάκωσης ανά κανόνα και ανά νέφος.

Η **`Notification`** αναπαριστά ειδοποίηση προς χρήστη. Ο `type` κατηγοριοποιεί το συμβάν με τιμές `QOS_VIOLATION`, `DEPLOYMENT_SUCCESS`, `DEPLOYMENT_FAILED` και `ACCOUNT_ACTIVATED`, ο `title` συνοψίζει, το `message` περιέχει το αναγνώσιμο κείμενο, το `isRead` υποστηρίζει την ένδειξη μη αναγνωσμένων και το `createdAt` την ταξινόμηση. Η διπλή συσχέτιση προς `User` (υποχρεωτική) και `Application` (προαιρετική, 0..1) επιτρέπει το φιλτράρισμα ανά εφαρμογή, καλύπτοντας ταυτόχρονα ειδοποιήσεις που δεν αφορούν εφαρμογή (π.χ. ενεργοποίηση λογαριασμού). Σημειώνεται ότι το `isRead` αφορά αποκλειστικά τις ειδοποιήσεις εντός εφαρμογής· για τα email δεν παρακολουθείται ανάγνωση, καθώς αυτό θα απαιτούσε μηχανισμό tracking pixel ή επιβεβαίωσης που δεν υλοποιήθηκε.

Συμπληρωματικά πεδία. Στην ``Application``, το πεδίο `tags` συγκρατεί ελεύθερες ετικέτες κατηγοριοποίησης που διευκολύνουν την ομαδοποίηση και τη μελλοντική αναζήτηση, ενώ η `version` επιτρέπει τη διάκριση διαδοχικών εκδόσεων του ίδιου φορτίου εργασίας. Στην ``QosRule``, το `durationSeconds` ορίζει για πόσο χρόνο πρέπει η μετρική να παραμείνει εκτός ορίου πριν ενεργοποιηθεί η ενέργεια, αποτρέποντας ψευδώς θετικές ενεργοποιήσεις από στιγμιαίες αιχμές, ενώ το `actionTarget` περιορίζει την ενέργεια σε συγκεκριμένο πάροχο («AWS», «AZURE») ή την επεκτείνει σε όλους («all») — διάκριση που αποκτά σημασία όταν η ίδια εφαρμογή είναι διατεταγμένη σε περισσότερα από ένα νέφη.



Εικόνα 4.11 — Διάγραμμα κλάσεων του μοντέλου πεδίου

4.4 Υλοποίηση

4.4.1 Τεχνολογική Στοιβά

Πίνακας 4.7 — Τεχνολογική Στοιβά Υλοποίησης

Επίπεδο	Τεχνολογίες	Αιτιολόγηση Επιλογής
Frontend	React.js 18, TypeScript 5, Tailwind CSS, React Query, Zustand (το Recharts περιλαμβάνεται στις εξαρτήσεις χωρίς να χρησιμοποιείται)	Το React προσφέρει επαναχρησιμοποιήσιμα components και, σε συνδυασμό με την TypeScript, ασφάλεια τύπων· το Recharts αποδίδει αποδοτικά γραφήματα πραγματικού χρόνου· το Tailwind επιταχύνει την ανάπτυξη διεπαφής χωρίς προσαρμοσμένο CSS· το React Query διαχειρίζεται την κατάσταση διακομιστή και την προσωρινή αποθήκευση· το Zustand παρέχει ελαφρύ καθολικό store.
HTTP client	Axios με interceptors	Αυτόματη έγχυση JWT και κεντρικοποιημένος χειρισμός σφαλμάτων.
Backend framework	Spring Boot 3.2 (Java 17)	Ήρικο, enterprise-grade πλαίσιο με πλούσιο οικοσύστημα.
Ασφάλεια	Spring Security 6, JWT (jjwt 0.12), BCrypt (strength 12)	Καθιερωμένη πρακτική για stateless αυθεντικοποίηση και ασφαλή κατακερματισμό κωδικών.
ORM / βάση	Spring Data JPA, Hibernate, PostgreSQL 15, Flyway	Συναλλαγές ACID, υποστήριξη πεδίων JSON, μεταπτώσεις σχήματος υπό έλεγχο εκδόσεων.
Παρακολούθηση	Prometheus, Grafana 13, Alertmanager	Ανεξάρτητα από πάροχο, εγγενή για Kubernetes, ανοικτού κώδικα.
IaC	Terraform 1.6 και kubectl (κλήση μέσω ProcessBuilder)	Δηλωτική διαχείριση πολλαπλών νεφών με εκτενές οικοσύστημα providers· το kubectl χρησιμοποιείται από το ScalingService για την κλιμάκωση των Deployments της εφαρμογής.
Δοχείοποίηση	Docker, Docker Compose, Nginx	Φορητότητα και αναπαραγωγικότητα· χρησιμοποιείται για την ίδια την πλατφόρμα (backend, frontend, βάση, αντίστροφος διαμεσολαβητής· βλ. Ενότητα 5.1) και ως αναφορά για τα δοχειοποιημένα φορτία που αυτή διαχειρίζεται.
Δοκιμές	JUnit 5, Mockito, AssertJ, Vitest και Testing Library, GitHub Actions	Κάλυψη backend σε επίπεδο μονάδας και ολοκλήρωσης, δοκιμές πλήρους ροής στη διεπαφή και αυτοματοποιημένος αγωγός CI.

Επίπεδο	Τεχνολογίες	Αιτιολόγηση Επιλογής
Τεκμηρίωση API	Springdoc OpenAPI 3 / Swagger UI	Αυτόματα παραγόμενη και πάντοτε συγχρονισμένη με τον κώδικα.

4.4.2 Δομή Πηγαίου Κώδικα

Ο πλήρης πηγαίος κώδικας είναι διαθέσιμος στο αποθετήριο GitHub της εργασίας (<https://github.com/andreaskosmop/qos-multicloud-platform>), το οποίο περιλαμβάνει backend, frontend, Terraform modules, τη διαμόρφωση Docker Compose και οδηγίες εγκατάστασης στο README.md.

Backend (Spring Boot):

```
src/main/java/gr/aegean/cloudplatform/
├── config/           # SecurityConfig, CorsConfig, JacksonConfig,
                    # PasswordEncoderConfig, RestTemplateConfig
├── controller/       # AuthController, UserController, ApplicationController
├── service/          # UserService, ApplicationService, TerraformService,
                    # MonitoringService, PrometheusService, ScalingService,
                    # NotificationService
├── repository/       # UserRepository, ApplicationRepository, CloudConfigRepository,
                    # QosRuleRepository, DeploymentRepository,
                    # ScalingEventRepository, NotificationRepository
├── model/
│   ├── entity/      # User, Application, CloudConfig, QosRule,
│                   # Deployment, ScalingEvent, Notification
│   └── enums/       # UserRole, AppStatus, DeploymentStatus, CloudProvider,
                    # ActionType, ScalingEventStatus
├── dto/
│   ├── request/     # RegisterRequest, LoginRequest, CreateAppRequest
│   └── response/     # AuthResponse, UserResponse
├── security/        # JwtAuthFilter, JwtUtils
├── terraform/       # TerraformTemplateGenerator, TerraformResult
├── exception/       # DeploymentException
└── src/main/resources/db/migration/ # V1 users, V2 applications,
                                      # V3 indexes, V4 seed admin (Flyway)
```

Frontend (React.js):

```
src/
├── api/             # client.ts - στιγμιότυπο Axios με αυτόματη έγχυση του JWT
├── components/
│   └── common/      # Navbar
├── hooks/           # useApplications (React Query)
├── pages/           # LoginPage, RegisterPage, DashboardPage, ApplicationsPage
├── store/           # authStore (Zustand)
└── _tests_/        # Navbar.test.tsx (Vitest + Testing Library)
```

4.4.3 Το Terraform στο Πλαίσιο της Πλατφόρμας

Στην παρούσα πλατφόρμα το Terraform επιτελεί τέσσερις λειτουργίες:

- Δημιουργία συστάδων Kubernetes.** AWS EKS και Azure AKS με παραμετροποιήσιμο αριθμό κόμβων, τύπο instance (t3.medium / Standard_DS2_v2) και έκδοση Kubernetes. Αξιοποιούνται ώριμα community modules (terraform-aws-modules/eks, Azure/aks/azurerms), ώστε να μην αναπαραχθεί λογική που έχει ήδη ελεγχθεί εκτενώς από την κοινότητα.
- Ρύθμιση δικτύωσης.** Αυτόματη δημιουργία VPC/VNet, υποδικτύων, ομάδων ασφαλείας, πυλών και εξισορροπητών φόρτου ανά νέφος, χωρίς χειρωνακτική παρέμβαση του χρήστη.
- Παροχή σημείων πρόσβασης παρακολούθησης.** Τα outputs του Terraform επιστρέφουν το clusterEndpoint και το prometheusEndpoint, τα οποία αποθηκεύονται στην οντότητα Deployment και τροφοδοτούν στη συνέχεια το

PrometheusService. Επισημαίνεται ότι η ίδια η εγκατάσταση της στοίβας kube-prometheus-stack δεν εκτελείται από την πλατφόρμα στην παρούσα έκδοση: πραγματοποιείται χωριστά κατά την προετοιμασία του περιβάλλοντος (§5.2.1) και αποτελεί προτεινόμενη επέκταση μέσω του Helm provider του Terraform.

4. **Καταστροφή υποδομής.** Πλήρης αποδέσμευση συστάδων, κόμβων, δικτύωσης και παρακολούθησης με μία ενέργεια, ώστε να μην παραμένουν ορφανοί πόροι που επιβαρύνουν το κόστος.

Η πλατφόρμα καλεί το Terraform προγραμματιστικά: το `TerraformService` παράγει δυναμικά αρχεία `.tf` από πρότυπα Freemarker βάσει της διαμόρφωσης του χρήστη, εκτελεί τη σειρά `init` → `plan` → `apply` μέσω `ProcessBuilder` με καταγραφή εξόδου σε πραγματικό χρόνο, αναλύει τα `outputs` για το cluster endpoint και ενημερώνει τη βάση δεδομένων. Το ακόλουθο απόσπασμα παρουσιάζει τη δομή του module για το AWS EKS σε απλοποιημένη μορφή· τα πλήρη modules παρατίθενται στα Παραρτήματα Γ και Δ.

```
# -----
# modules/aws-eks/versions.tf
# Καρφίτσωμα εκδόσεων: εξασφαλίζει αναπαραγώγιμες διατάξεις μεταξύ εκτελέσεων.
# -----
terraform {
  required_version = ">= 1.6.0"

  required_providers {
    aws = { source = "hashicorp/aws", version = "~> 5.40" }
    helm = { source = "hashicorp/helm", version = "~> 2.12" }
  }

  # Απομακρυσμένο state με κλειδωμά: αποτρέπει ταυτόχρονες εγγραφές όταν
  # η πλατφόρμα εξυπηρετεί παράλληλα αιτήματα διάταξης (MFA-SC01).
  backend "s3" {
    bucket      = "qos-platform-tfstate"
    key         = "workspaces/${var.application_id}/terraform.tfstate"
    region      = "eu-central-1"
    encrypt     = true
    dynamodb_table = "qos-platform-tflock"
  }
}

# -----
# modules/aws-eks/variables.tf
# -----
variable "application_id" {
  description = "Αναγνωριστικό εφαρμογής· χρησιμοποιείται ως πρόθεμα ονομάτων πόρων."
  type        = string
}

variable "cluster_version" {
  description = "Έκδοση του control plane του Kubernetes."
  type        = string
  default     = "1.29"
}

variable "instance_type" {
  type        = string
  default     = "t3.medium"
}

variable "min_nodes" {
  type        = number

  # Επικύρωση εισόδου στο επίπεδο IaC: το ίδιο όριο επιβάλλεται και από
```

```

# to ScalingService, ώστε ένα σφάλμα σε ένα επίπεδο να μην αρκεί.
validation {
  condition      = var.min_nodes >= 1
  error_message = "Απαιτείται τουλάχιστον ένας κόμβος."
}
}

variable "max_nodes" {
  type = number
}

variable "desired_nodes" {
  type = number
}

# -----
# modules/aws-eks/main.tf
# -----
locals {
  # Ενιαίες ετικέτες σε όλους τους πόρους: επιτρέπουν την ανίχνευση και
  # εκκαθάριση ορφανών πόρων ανά εφαρμογή, καθώς και ανάλυση κόστους.
  common_tags = {
    ManagedBy      = "qos-aware-platform"
    ApplicationId = var.application_id
    Terraform      = "true"
  }
}

# Αποκλειστικό VPC ανά διάταξη: απομόνωση δικτύου μεταξύ εφαρμογών.
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"
  version = "~> 5.5"

  name = "vpc-${var.application_id}"
  cidr = "10.0.0.0/16"

  azs              = slice(data.aws_availability_zones.available.names, 0, 3)
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  public_subnets  = ["10.0.101.0/24", "10.0.102.0/24", "10.0.103.0/24"]

  enable_nat_gateway = true
  single_nat_gateway = true # έναντι κόστους· σε παραγωγή προτιμάται ένα ανά AZ

  tags = local.common_tags
}

module "eks" {
  source = "terraform-aws-modules/eks/aws"
  version = "~> 20.8"

  cluster_name      = "eks-${var.application_id}"
  cluster_version = var.cluster_version

  vpc_id      = module.vpc.vpc_id
  subnet_ids = module.vpc.private_subnets # οι κόμβοι δεν εκτίθενται δημόσια

  # Το endpoint του control plane παραμένει δημόσιο ώστε να είναι προσβάσιμο
  # από την πλατφόρμα, αλλά περιορίζεται σε συγκεκριμένα δίκτυα προέλευσης.
  cluster_endpoint_public_access      = true
  cluster_endpoint_public_access_cidrs = var.allowed_management_cidrs

  # Κρυπτογράφηση των Kubernetes secrets σε ηρεμία (AWS best practice).
  cluster_encryption_config = {
    resources = ["secrets"]
  }
}

```

```

# Καταγραφή ελέγχου του control plane, απαραίτητη για ιχνηλασιμότητα.
cluster_enabled_log_types = ["api", "audit", "authenticator"]

eks_managed_node_groups = {
  default = {
    instance_types = [var.instance_type]
    min_size       = var.min_nodes
    max_size       = var.max_nodes
    desired_size   = var.desired_nodes
    capacity_type  = "ON_DEMAND"
  }
}

tags = local.common_tags
}

# -----
# Στοιβα παρακολούθησης: εγκαθίσταται μόνο αφού η συστάδα είναι διαθέσιμη.
# -----
provider "helm" {
  kubernetes {
    host                = module.eks.cluster_endpoint
    cluster_ca_certificate =
base64decode(module.eks.cluster_certificate_authority_data)

    exec {
      api_version = "client.authentication.k8s.io/v1beta1"
      command     = "aws"
      args        = ["eks", "get-token", "--cluster-name", module.eks.cluster_name]
    }
  }
}

resource "helm_release" "kube_prometheus_stack" {
  name                = "monitoring"
  repository          = "https://prometheus-community.github.io/helm-charts"
  chart               = "kube-prometheus-stack"
  version             = "58.1.0" # καρφιτσωμένη έκδοση chart για αναπαραγωγιμότητα
  namespace           = "monitoring"
  create_namespace    = true

  # Αναμονή έως ότου τα pods είναι έτοιμα, ώστε το output prometheus_endpoint
  # να μην επιστραφεί πριν η υπηρεσία γίνει πράγματι προσβάσιμη.
  wait      = true
  timeout   = 900

  values = [yamlencode({
    grafana = {
      adminPassword = var.grafana_admin_password # μέσω μεταβλητής, ποτέ σταθερά
    }
    prometheus = {
      prometheusSpec = {
        retention = "7d"
      }
    }
  })]

  depends_on = [module.eks]
}

# -----
# modules/aws-eks/outputs.tf
# Τα outputs αναλύονται από το TerraformService και αποθηκεύονται στο Deployment.
# -----
output "cluster_endpoint" {

```

```

    value = module.eks.cluster_endpoint
  }

  output "prometheus_endpoint" {
    value = "http://${helm_release.kube_prometheus_stack.name}-
prometheus.monitoring.svc:9090"
  }

  output "node_count" {
    value = var.desired_nodes
  }

```

Οι πρακτικές που ενσωματώνονται στο παραπάνω module και οι οποίες αντανakλούν τις καθιερωμένες συστάσεις ασφαλείας και λειτουργικής αριστείας των παρόχων είναι: καρφίτσωμα εκδόσεων για αναπαραγωγιμότητα, απομακρυσμένο state με κλείδωμα ώστε να αποτρέπονται ταυτόχρονες τροποποιήσεις της ίδιας υποδομής, τοποθέτηση των κόμβων σε ιδιωτικά υποδίκτυα, περιορισμός του δημόσιου endpoint του control plane σε συγκεκριμένα δίκτυα, κρυπτογράφηση των Kubernetes secrets σε ηρεμία, ενεργοποίηση καταγραφής ελέγχου, ενιαία επισήμανση (tagging) όλων των πόρων για ιχνηλασιμότητα και εκκαθάριση, επικύρωση των μεταβλητών εισόδου στο ίδιο το επίπεδο IaC και αποφυγή σταθερά ενσωματωμένων μυστικών.

4.4.4 Σημεία Πρόσβασης REST API

Πίνακας 4.8 — Σημεία Πρόσβασης (Endpoints) του REST API

Μέθοδος	Endpoint	Περιγραφή	Εξουσιοδότηση
POST	/api/v1/auth/register	Εγγραφή νέου λογαριασμού	Δημόσιο
POST	/api/v1/auth/login	Σύνδεση και έκδοση JWT	Δημόσιο
GET	/api/v1/users/me	Στοιχεία συνδεδεμένου χρήστη	USER+
GET	/api/v1/users	Λίστα χρηστών	ADMIN
PATCH	/api/v1/users/{username}/status	Ενεργοποίηση / απενεργοποίηση λογαριασμού	ADMIN
DELETE	/api/v1/users/{username}	Διαγραφή χρήστη	USER+
POST	/api/v1/applications	Δημιουργία εφαρμογής	USER+
GET	/api/v1/applications	Λίστα εφαρμογών του συνδεδεμένου χρήστη	USER+
GET	/api/v1/applications/{id}	Λεπτομέρειες εφαρμογής	USER+
DELETE	/api/v1/applications/{id}	Διαγραφή εφαρμογής	USER+
POST	/api/v1/applications/{id}/deployment	Διάταξη βάσει υπάρχουσας	USER+

Μέθοδος	Endpoint	Περιγραφή	Εξουσιοδότηση
		διαμόρφωσης νέφους	
DELETE	/api/v1/applications/{id}/deployment	Καταστροφή διάταξης	USER+
PATCH	/api/v1/applications/{id}/deployment	Χειροκίνητη κλιμάκωση κόμβων (μη υλοποιημένη)	USER+
GET	/api/v1/applications/{id}/metrics	Τρέχουσες ή ιστορικές μετρικές	USER+
GET	/api/v1/applications/{id}/scaling-events	Ιστορικό scaling	USER+
GET	/api/v1/notifications	Ειδοποιήσεις	USER+
PATCH	/api/v1/notifications/{id}	Σήμανση ως read	USER+

Σημειώνεται ότι οι κλήσεις για ιστορικές ή τρέχουσες μετρικές διαφοροποιούνται μέσω παραμέτρων ερωτήματος (*from*, *to*) και όχι μέσω διακριτών διαδρομών, ώστε να τηρείται η αρχή του ενιαίου πόρου. Παράδειγμα πλήρους απάντησης με συνδέσμους HATEOAS παρατίθεται, ως προδιαγραφή της προτεινόμενης επέκτασης, στο Παράρτημα Β.

4.4.5 Ομάδες Κλάσεων Υλοποίησης

Η υλοποίηση του backend οργανώνεται σε ομάδες κλάσεων ανά επίπεδο αρχιτεκτονικής, σύμφωνα με το μοτίβο CSR που περιγράφηκε στην §4.3.2.

A. Οντότητες (entities). Αντιστοιχούν στις κλάσεις του μοντέλου πεδίου (§4.3.5): *User*, *Application*, *CloudConfig*, *QosRule*, *Deployment*, *ScalingEvent*, *Notification*. Υλοποιούνται ως JPA entities με επισημάνσεις *@Entity* και *@Table*, ενώ οι συσχετίσεις του μοντέλου αποδίδονται με *@OneToMany* και *@ManyToOne*.

B. Αποθετήρια (repositories). Κάθε οντότητα συνοδεύεται από διεπαφή repository που επεκτείνει το *JpaRepository* του Spring Data, παρέχοντας τυποποιημένες λειτουργίες CRUD και προσαρμοσμένα ερωτήματα παραγόμενα από το όνομα της μεθόδου (π.χ. *findAllByStatus*, *findByUserAndProvider*).

Γ. Υπηρεσίες (services). Συγκεντρώνουν την επιχειρησιακή λογική. Πέραν της περιγραφής της §4.3.2, οι κρίσιμες υπηρεσίες λειτουργούν ως εξής:

- Το **‘TerraformService’** αναλαμβάνει να μετατρέψει μια αφηρημένη επιθυμία του χρήστη — «θέλω αυτή την εφαρμογή να τρέχει σε αυτό το νέφος με αυτά τα χαρακτηριστικά» — σε πραγματική υποδομή. Για κάθε διάταξη δημιουργεί απομονωμένο κατάλογο εργασίας (workspace), ζητά από τον *TemplateGenerator* να παραγάγει τον αντίστοιχο κώδικα Terraform για τον επιλεγμένο πάροχο, και στη συνέχεια εκτελεί σειριακά τα στάδια αρχικοποίησης, σχεδιασμού και εφαρμογής, καλώντας το Terraform ως εξωτερική διεργασία μέσω *ProcessBuilder*. Η έξοδος της διεργασίας διαβάζεται τμηματικά όσο αυτή εκτελείται, ώστε ο χρήστης να βλέπει την πρόοδο σε πραγματικό χρόνο αντί να αναμένει σιωπηλά για αρκετά λεπτά. Όταν η εκτέλεση ολοκληρωθεί, η υπηρεσία εξάγει από τα outputs τα σημεία πρόσβασης της συστάδας και του Prometheus και τα καταγράφει στη διάταξη. Η ίδια υπηρεσία εξυπηρετεί και τη μεταβολή

χωρητικότητα, εφαρμόζοντας εκ νέου το σχέδιο στο υφιστάμενο workspace με νέο επιθυμητό αριθμό κόμβων, καθώς και την πλήρη αποδέσμευση της υποδομής.

- Το **MonitoringService** αποτελεί τον πυρήνα του μηχανισμού ποιότητας υπηρεσίας. Κάθε τριάντα δευτερόλεπτα εξετάζει όλες τις ενεργές διατάξεις και, για καθεμία, τους κανόνες που έχει ορίσει ο χρήστης. Για κάθε κανόνα ελέγχει πρώτα αν έχει παρέλθει το διάστημα ηρεμίας από την τελευταία ενεργοποίηση, ώστε να αποφεύγονται ταλαντώσεις, και εν συνεχεία ζητά την τρέχουσα τιμή της μετρικής. Αν η τιμή παραβιάζει το όριο, αναθέτει την εκτέλεση της ενέργειας στο ScalingService. Η ίδια υπηρεσία εξυπηρετεί και τα αιτήματα του χρήστη για τρέχουσες ή ιστορικές τιμές μετρικών.

- Το **ScalingService** εκτελεί την ενέργεια που υπαγορεύει ο κανόνας: αυξάνει ή μειώνει τα αντίγραφα της εφαρμογής μέσω του Kubernetes API, ή μεταβάλλει τον αριθμό κόμβων μέσω του TerraformService, ή απλώς ενημερώνει τον χρήστη. Πριν από κάθε ενέργεια επιβεβαιώνει ότι το αποτέλεσμα παραμένει εντός των ορίων που έχει ορίσει ο χρήστης, και σε κάθε περίπτωση καταγράφει το συμβάν με την έκβασή του, ώστε η συμπεριφορά του συστήματος να είναι πλήρως ιχνηλάσιμη εκ των υστέρων.

- Ο **PrometheusService** μεταφράζει τα φιλικά ονόματα μετρικών που επιλέγει ο χρήστης σε πλήρη ερωτήματα PromQL (για παράδειγμα, το `cpu_usage` αντιστοιχίζεται σε `rate(container_cpu_usage_seconds_total{...}[5m]) * 100`) και εκτελεί την αντίστοιχη κλήση HTTP προς το Prometheus της συγκεκριμένης διάταξης. Σε αποτυχία επιστρέφει κενή τιμή αντί να διαδώσει εξαίρεση, υποστηρίζοντας τη σταδιακή υποβάθμιση που απαιτεί η MFA-R02.

Δ. Ελεγκτές (controllers). Εκθέτουν τα σημεία πρόσβασης της §4.4.4 ανά πόρο: AuthController, UserController και ApplicationController. Οι ροές διάταξης, αποδιάταξης και κλιμάκωσης εκτίθενται ως υποπόροι του ApplicationController και όχι ως αυτοτελείς ελεγκτές. Η ονοματολογία ταυτίζεται με εκείνη του διαγράμματος συστατικών (Εικόνα 4.2). Κάθε ελεγκτής επιτελεί δύο εγκάρσιες λειτουργίες πριν αναθέσει την εργασία στην υπηρεσία: **επικύρωση των δεδομένων εισόδου** μέσω @Valid επί του αντικειμένου αιτήματος, σε συνδυασμό με δηλωτικούς περιορισμούς (@NotBlank, @Email, @Min, @Pattern) στα πεδία των DTO, και **έλεγχο πρόσβασης** μέσω @PreAuthorize, επισήμανσης του Spring Security που τοποθετείται πάνω από κάθε μέθοδο και αποτιμά μια έκφραση SpEL πριν από την εκτέλεσή της, επιβάλλοντας τόσο τον ρόλο όσο και την ιδιοκτησία του πόρου.

Ε. Αντικείμενα μεταφοράς δεδομένων (DTO). Χρησιμοποιούνται διακριτές κλάσεις DTO αντί απευθείας έκθεσης των οντοτήτων, ώστε να αποτρέπεται η διαρροή ευαίσθητων πεδίων και να αποσυνδέεται το εσωτερικό μοντέλο από το δημόσιο συμβόλαιο του API. Ενδεικτικά, η οντότητα User περιλαμβάνει τα πεδία passwordHash, securityAnswerHash και securityQuestion, ενώ το αντίστοιχο UserResponse τα **αποκρύπτει πλήρως**:

```
// Οντότητα – εσωτερικό μοντέλο, δεν εκτίθεται ποτέ απευθείας.
@Entity
@Table(name = "app_user")
public class User {
    @Id @GeneratedValue private UUID id;
    private String username;
    private String fullName;
    private String email;
    private String passwordHash; // ΔΕΝ εκτίθεται
    private String securityQuestion; // ΔΕΝ εκτίθεται
    private String securityAnswerHash; // ΔΕΝ εκτίθεται
```

```

    @Enumerated(EnumType.STRING) private UserRole role;
    private boolean active;
    private Instant createdAt;
    private Instant updatedAt;
    // ...
}

// DTO απάντησης – μόνο τα πεδία που επιτρέπεται να δει ο πελάτης.
public record UserResponse(
    UUID id,
    String username,
    String fullName,
    String email,
    UserRole role,
    boolean active,
    Instant createdAt) { }

```

Με τον τρόπο αυτό, ακόμη και αν προστεθεί μελλοντικά νέο ευαίσθητο πεδίο στην οντότητα, αυτό δεν εκτίθεται αυτομάτως στο API, καθώς η αντιστοίχιση είναι ρητή και όχι αυτόματη.

ΣΤ. Βοηθητικές κλάσεις (utilities). Η `JwtUtils` παράγει και επαληθεύει τα JSON Web Tokens (Jones et al., 2015) με αλγόριθμο υπογραφής HS256 (HMAC-SHA256, συμμετρικός αλγόριθμος όπου το ίδιο μυστικό κλειδί χρησιμοποιείται για υπογραφή και επαλήθευση), με διάρκεια ισχύος οκτώ ωρών· κατά την αποσύνδεση το αναγνωριστικό του token προστίθεται σε λίστα ανάκλησης στη βάση, ώστε να μη γίνεται πλέον αποδεκτό ακόμη και αν δεν έχει λήξει χρονικά.

Η κλάση `CredentialCipher` αναλαμβάνει την προστασία των διαπιστευτηρίων νέφους. Χρησιμοποιείται **AES-256 σε λειτουργία GCM** (Galois/Counter Mode), δηλαδή συμμετρική κρυπτογράφηση με κλειδί μήκους 256 bit και ενσωματωμένη πιστοποίηση αυθεντικότητας, η οποία αποτρέπει τη μη ανιχνεύσιμη αλλοίωση του κρυπτογραφήματος. Για κάθε εγγραφή παράγεται τυχαίο διάνυσμα αρχικοποίησης (IV) μήκους 96 bit μέσω `SecureRandom`, το οποίο αποθηκεύεται μαζί με το κρυπτογράφημα. Το κύριο κλειδί **δεν παράγεται τυχαία από την εφαρμογή** — κάτι που θα καθιστούσε αδύνατη την αποκρυπτογράφηση μετά από επανεκκίνηση — αλλά παρέχεται εξωτερικά μέσω μεταβλητής περιβάλλοντος (`PLATFORM_CREDENTIAL_KEY`) που καταναλώνεται από το `application.yml`, ώστε να μην ενσωματώνεται στον πηγαίο κώδικα ούτε στην εικόνα Docker. Ο μηχανισμός αυτός επιτρέπει τη μελλοντική αντικατάσταση της πηγής του κλειδιού από διαχειριστή μυστικών (π.χ. HashiCorp Vault), όπως προτείνεται στην Ενότητα 6.2. Τέλος, επιμέρους βοηθητικές κλάσεις καλύπτουν επαναχρησιμοποιήσιμη λογική επικύρωσης.

4.4.6 Διεπαφή Χρήστη της Πλατφόρμας

Η ενότητα αυτή τεκμηριώνει το εύρος της διαδικτυακής διεπαφής που υλοποιήθηκε. Επισημαίνεται εξ αρχής, για λόγους ακριβούς αποτύπωσης της εργασίας, ότι η διεπαφή αποτελεί ελάχιστη υλοποίηση αναφοράς (reference client) και όχι πλήρη διαχειριστική κονσόλα: καλύπτει την ταυτοποίηση του χρήστη και τη βασική διαχείριση εφαρμογών, ενώ οι υπόλοιπες λειτουργίες της πλατφόρμας — ορισμός κανόνων ποιότητας υπηρεσίας, διάταξη υποδομής, εποπτεία μετρικών και ιστορικό ενεργειών κλιμάκωσης — εκτίθενται στο επίπεδο του μοντέλου πεδίου και των υπηρεσιών, αλλά δεν διαθέτουν αντίστοιχες οθόνες.

Η εφαρμογή είναι μονοσέλιδη (Single Page Application) σε React.js 18 με TypeScript 5, με δρομολόγηση μέσω React Router και διαχείριση κατάστασης ταυτοποίησης

μέσω Zustand⁸. Η πρόσβαση στο REST API γίνεται από ενιαίο στιγμιότυπο Axios⁹ που εγγέει αυτόματα το JWT σε κάθε αίτημα, ενώ η ανάκτηση δεδομένων και η επικύρωση της προσωρινής μνήμης γίνονται μέσω React Query. Οι προστατευμένες διαδρομές ελέγχονται από το στοιχείο ProtectedRoute, το οποίο ανακατευθύνει στη σελίδα σύνδεσης όταν δεν υπάρχει έγκυρο διακριτικό.

Υλοποιούνται τέσσερις οθόνες:

- *LoginPage* — φόρμα σύνδεσης με όνομα χρήστη και κωδικό. Επιτυχής υποβολή επιστρέφει JWT, το οποίο αποθηκεύεται στο authStore και συνοδεύει κάθε επακόλουθη κλήση.
- *RegisterPage* — φόρμα εγγραφής νέου λογαριασμού. Ο κωδικός κατακερματίζεται με BCrypt στον διακομιστή και ο λογαριασμός δημιουργείται σε κατάσταση αναμονής έγκρισης.
- *DashboardPage* — αρχική οθόνη μετά τη σύνδεση, με σύντομο μήνυμα προσανατολισμού προς τη διαχείριση εφαρμογών.
- *ApplicationsPage* — η κύρια λειτουργική οθόνη: φόρμα δημιουργίας νέας εφαρμογής με τίτλο, κατάλογος των εφαρμογών του συνδεδεμένου χρήστη με ένδειξη κατάστασης (AppStatus), και ενέργεια διαγραφής ανά εφαρμογή. Η ανάκτηση και οι μεταβολές εκτελούνται μέσω των αγκίστρων (hooks) useApplications, useCreateApplication και useDeleteApplication.

Η κοινή γραμμή πλοήγησης (Navbar) εμφανίζει τον συνδεδεμένο χρήστη και προσφέρει αποσύνδεση. Δεν έχει υλοποιηθεί οθόνη διαχείρισης χρηστών, παρότι τα αντίστοιχα σημεία πρόσβασης υπάρχουν στο API και προστατεύονται με @PreAuthorize("hasRole('ADMIN')") (Πίνακας 4.8): η διαχείριση χρηστών είναι επομένως εφικτή μέσω του API αλλά όχι μέσω γραφικής διεπαφής.

Ως προς τις μη λειτουργικές απαιτήσεις χρηστικότητας, η ενιαία εμφάνιση επιτυγχάνεται μέσω του κοινού συστήματος σχεδίασης Tailwind CSS¹⁰ και η δυναμική ενημέρωση χωρίς επαναφόρτωση σελίδας μέσω React Query¹¹. Οι απαιτήσεις που προϋποθέτουν οθόνες πολλαπλών βημάτων ή οπτικοποίηση μετρικών δεν ικανοποιούνται στην παρούσα έκδοση και αποτιμώνται αναλόγως στην Ενότητα 4.7. Οι βιβλιοθήκες Recharts, React Hook Form και Zod περιλαμβάνονται στις εξαρτήσεις του έργου ως προετοιμασία για τις οθόνες αυτές, αλλά δεν χρησιμοποιούνται από τον υφιστάμενο κώδικα.

Η διεπαφή καλύπτεται από μία δοκιμή στοιχείου (Navbar) με Vitest και Testing Library. Στιγμιότυπα των δύο βασικών οθονών παρατίθενται στην Ενότητα 5.2.5.

4.6 Δοκιμή και Αξιολόγηση Απόδοσης (4ο Στάδιο του Μοντέλου Waterfall)

4.6.1 Στρατηγική Δοκιμών

Οι ποσοτικές τιμές που παρατίθενται στην παρούσα ενότητα προέρχονται από την πειραματική επίδειξη του Κεφαλαίου 5 και συγκεντρώνονται εδώ για λόγους συνοχής της αξιολόγησης· τα αντίστοιχα τεκμήρια (στιγμιότυπα) παρατίθενται στις §5.2.2–5.2.5. Η στρατηγική δοκιμών ακολουθεί την πυραμίδα δοκιμών (test pyramid),

⁸ <https://zustand.docs.pmnd.rs/>

⁹ <https://axios-http.com/>

¹⁰ <https://tailwindcss.com/>

¹¹ <https://tanstack.com/query/latest>

πρότυπο που εισήγαγε ο Cohn (2009) και διέδωσε ο Fowler (2012): πολλές δοκιμές μονάδας στη βάση, λιγότερες δοκιμές ολοκλήρωσης στο ενδιάμεσο και ελάχιστες δοκιμές πλήρους ροής στην κορυφή. Η επιλογή δικαιολογείται από τα διαφορετικά χαρακτηριστικά κόστους κάθε επιπέδου: οι δοκιμές μονάδας είναι ταχύτερες και φθηνές στη συντήρηση, επιτρέποντας εκτέλεση σε κάθε υποβολή κώδικα· οι δοκιμές ολοκλήρωσης, βραδύτερες λόγω πραγματικών εξαρτήσεων (Testcontainers), επαληθεύουν τη συνεργασία των στρωμάτων· οι δοκιμές πλήρους ροής, οι πλέον αργές και εύθραυστες, περιορίζονται σε λίγα κρίσιμα σενάρια.

Πίνακας 4.9 — Στρατηγική Δοκιμών ανά Επίπεδο

Τύπος δοκιμής	Πλαίσιο	Τι δοκιμάζει	Κάλυψη
Δοκιμές μονάδας (backend)	JUnit 5, Mockito, AssertJ	Λογική του επιπέδου υπηρεσιών: MonitoringService (περιοδική αξιολόγηση κανόνων), PrometheusService (σύνθεση και εκτέλεση ερωτημάτων PromQL), TerraformService (παραγωγή αρχείων .tf και χειρισμός σφαλμάτων)	Τρεις κλάσεις υπηρεσιών
Δοκιμή στοιχείου (frontend)	Vitest, Testing Library	Απόδοση της κοινής γραμμής πλοήγησης και εξάρτησή της από την κατάσταση ταυτοποίησης	Μία κλάση στοιχείου (Navbar)
Συνεχής ενοποίηση	GitHub Actions	Αυτόματη εκτέλεση των παραπάνω δοκιμών και κατασκευή των εικόνων Docker σε κάθε ώθηση στους κλάδους main και develop (Παράρτημα ΣΤ)	Πλήρης αγωγός build και test

Σχέση με προγενέστερη έκδοση της αναφοράς. Σε προηγούμενη έκδοση ο πίνακας αυτός απαρίθμηνε τέσσερις κατηγορίες δοκιμών, συμπεριλαμβανομένων δοκιμών ολοκλήρωσης των ελεγκτών με Testcontainers και δοκιμών πλήρους ροής με Cypress. Η απαρίθμηση εκείνη αποτύπωνε τη σχεδιασμένη στρατηγική δοκιμών και όχι την υλοποιημένη σουίτα. Ο παρών πίνακας περιορίζεται σε όσα υπάρχουν πράγματι στο αποθετήριο, ώστε η αναφορά να μην αποδίδει στην εργασία κάλυψη που δεν διαθέτει. Οι δύο κατηγορίες που αφαιρέθηκαν παραμένουν ως αναγκαία επέκταση και καταγράφονται στη σημείωση που ακολουθεί.

Σημείωση. Η παρούσα έκδοση δεν περιλαμβάνει δοκιμές ολοκλήρωσης των ελεγκτών REST με Testcontainers ούτε δοκιμές πλήρους ροής με Cypress. Οι δύο αυτές κατηγορίες, μαζί με τη μέτρηση της κάλυψης κώδικα, αποτελούν αναγκαία επέκταση πριν από οποιαδήποτε χρήση της πλατφόρμας σε περιβάλλον παραγωγής.

Η διαγραφή λογαριασμού συμπεριλήφθηκε ρητά στις δοκιμές ασφαλείας, καθώς πρόκειται για μη αναστρέψιμη ενέργεια με διάδοση σε όλα τα εξαρτώμενα δεδομένα και, επομένως, εξίσου κρίσιμη με τη διαχείριση δικαιωμάτων.

4.6.2 Σύγκριση Μετρήσεων με Στόχους SLA

Πλαίσιο αξιολόγησης. Η αξιολόγηση πραγματοποιήθηκε με διάταξη της εφαρμογής επίδειξης `demo-webapp` (`nginx:alpine`, 3 αντίγραφα) σε τοπική συστάδα Minikube (Kubernetes v1.35.1, 4 CPU, 6 GB RAM), με εγκατεστημένη την πλήρη στοίβα παρακολούθησης και ρυθμισμένο HPA (όριο CPU 50%, `min = 2`, `max = 10` pods). Ο τεχνητός φόρτος παράχθηκε μέσω pods παραγωγής φόρτου (BusyBox με βρόχο `wget`) που στόχευαν την εφαρμογή, προκαλώντας σταδιακή αύξηση της χρησιμοποίησης CPU. Η επιλογή του `nginx:alpine` έγινε σκόπιμα: πρόκειται για ελαφρύ, ευρέως αναγνωρίσιμο δοχείο χωρίς εξαρτήσεις βάσης δεδομένων, το οποίο επιτρέπει καθαρή μέτρηση της συμπεριφοράς κλιμάκωσης χωρίς παρεμβολή από ίδια επιχειρησιακή λογική. Η μεθοδολογία ισχύει ανεξαρτήτως του περιεχομένου της διατεταγμένης εφαρμογής, καθώς η πλατφόρμα διαχειρίζεται οποιοδήποτε δοχειοποιημένο φορτίο εργασίας.

Μεθοδολογική σημείωση. Οι τιμές του Πίνακα 4.10 προέρχονται από μεμονωμένη μέτρηση κατά τη διάρκεια μίας εκτέλεσης επίδειξης — δεν πρόκειται για στατιστικά συγκεντρωμένες τιμές (μέσος όρος, διάμεσος, ακραίες τιμές) από πολλαπλές επαναλήψεις. Αποτελούν επομένως **ενδεικτικές** και όχι στατιστικά αντιπροσωπευτικές τιμές. Οι κωδικοί σε παρένθεση παραπέμπουν στα αντίστοιχα στιγμιότυπα οθόνης της Ενότητας 5.2, από όπου προήλθε κάθε μέτρηση: G = Grafana, P = Prometheus, K = Kubernetes/kubectl, A = Alertmanager, M = στοίβα παρακολούθησης, ακολουθούμενο από τον αύξοντα αριθμό στιγμιοτύπου εντός του σεναρίου.

Διευκρίνιση αντικειμένου μέτρησης. Οι τιμές των 20 και 21 ms που καταγράφονται στις Εικόνες 5.17 και 5.18 αφορούν αποκλειστικά τον χρόνο εκτέλεσης ενός ερωτήματος PromQL στον διακομιστή Prometheus, δηλαδή ένα μόνο σκέλος της διαδρομής. Δεν αποτελούν μέτρηση του συνολικού χρόνου απόκρισης από άκρο σε άκρο (πρόγραμμα περιήγησης → Nginx → REST API → βάση δεδομένων → απόκριση), τον οποίο αφορά η απαίτηση MFA-P01. Ο χρόνος αυτός δεν μετρήθηκε συστηματικά στην παρούσα εργασία· η μέτρησή του με εργαλείο φόρτου (π.χ. k6 ή JMeter) επί αντιπροσωπευτικού συνόλου λειτουργιών αποτελεί αναγκαία προσθήκη πριν από οποιονδήποτε ισχυρισμό συμμόρφωσης με στόχο SLA.

Οι στόχοι SLA με τους οποίους γίνεται η σύγκριση δεν αποτελούν αυθαίρετες τιμές, αλλά προκύπτουν από τις δημοσιευμένες οδηγίες ορθών πρακτικών των δύο παρόχων που υποστηρίζει η πλατφόρμα και από την επίσημη τεκμηρίωση του Kubernetes για τον μηχανισμό οριζόντιας αυτόματης κλιμάκωσης (Amazon Web Services, 2024a· Microsoft, 2024b· Kubernetes Authors, 2024a). Επισημαίνεται ότι οι τιμές χρησιμοποίησης πόρων εξαρτώνται από τον επιλεγμένο τύπο instance — μεγαλύτερος πόρος συνεπάγεται χαμηλότερο ποσοστό χρησιμοποίησης για τον ίδιο φόρτο — και ότι οι τιμές εδώ αντιστοιχούν στη διαμόρφωση Minikube 4 CPU / 6 GB της επίδειξης.

Πίνακας 4.10 — Σύγκριση Μετρηθεισών Τιμών με Στόχους SLA

Μετρική	Στόχος SLA	Μετρηθείσα τιμή	Πηγή	Αξιολόγηση
Χρησιμοποίηση CPU σε ηρεμία	10–20%	1,70%	Grafana G2	Εντός ορίων, με ευρύ περιθώριο

Μετρική	Στόχος SLA	Μετρηθείσα τιμή	Πηγή	Αξιολόγηση
Χρησιμοποίηση μνήμης	50–75%	38,2%	Grafana G2	Εντός ορίων
Χρόνος απόκρισης ερωτημάτων Prometheus	< 1.000 ms	20 ms	Prometheus P1/P2	50 φορές χαμηλότερος του ορίου
Χρόνος εκκίνησης pod	30–120 s	44 s	kubectl K3	Εντός ορίων
Ολοκλήρωση κλιμάκωσης HPA	3–5 min	~1,5 min (9m31s → 11m)	kubectl K3	Εντός στόχου
Παράδοση ειδοποίησης	30–60 s	~15 s	Alertmanager A2	Εντός ορίων
Εκκίνηση στοίβας παρακολούθησης	180–300 s	92–114 s	Στοιβά παρακολούθησης M1	Εντός ορίων

4.6.3 Μετρικές Αναφοράς (Baseline)

Ο Πίνακας 4.11 παρέχει στιγμιότυπο των βασικών μετρικών χρήσης πόρων της συστάδας σε ηρεμία, χωρίς τεχνητό φόρτο. Χρησιμεύει ως σημείο αναφοράς για τη σύγκριση με τις τιμές υπό φόρτο που παρουσιάζονται στη χρονογραμμή κλιμάκωσης της §4.6.4, όπου καταγράφεται η εξέλιξη της χρησιμοποίησης CPU και του αριθμού αντιγράφων κατά τη διάρκεια του πειράματος φόρτου.

Πίνακας 4.11 — Μετρικές Αναφοράς Συστήματος σε Ηρεμία

Μετρική	Τιμή	Πηγή
Χρησιμοποίηση CPU (σε ηρεμία)	1,70%	Grafana G2
Χρησιμοποίηση μνήμης	38,2%	Grafana G2
Δεσμευμένοι πόροι CPU (default namespace)	0,300 πυρήνες	Grafana G2
Δεσμευμένη μνήμη (default namespace)	192 MiB	Grafana G2
Εκτελούμενα pods (σύνολο συστάδας)	18	Prometheus P2
Παρακολουθούμενα σημεία μετρικών (endpoints)	17	Prometheus P1
Χρόνος απόκρισης ερωτήματος Prometheus	20–21 ms	Μέσος όρος δύο ανεξάρτητων ερωτημάτων (P1, P2)

Το σύνολο των 18 pods περιλαμβάνει τα 3 pods της εφαρμογής επίδειξης, 6 pods της στοίβας παρακολούθησης και τα υπόλοιπα pods συστήματος. Η διαφορά μεταξύ 18 pods και 17 παρακολουθούμενων σημείων οφείλεται σε ένα pod — τον βοηθητικό [grafana-sc-dashboard](#) — που δεν εκθέτει δικό του σημείο (παρακολούθησης) μετρικών αλλά μοιράζεται εκείνο του κύριου δοχείου.

Σχολιασμός. Η χρησιμοποίηση CPU σε ηρεμία (1,70%) είναι αμελητέα και τεκμηριώνει ότι η στοίβα παρακολούθησης δεν επιβαρύνει υπολογιστικά τη συστάδα. Για τη μνήμη, όπου το ποσοστό δεν επιτρέπει αντίστοιχο συμπέρασμα, πραγματοποιήθηκε απομονωμένη μέτρηση στο ίδιο περιβάλλον με το metrics-server (kubectl top). Η στοίβα παρακολούθησης του namespace monitoring καταναλώνει 1.250 MiB, έναντι 2.551 MiB για το σύνολο των pods της συστάδας και μόλις 30 MiB για τα τρία pods της εφαρμογής επίδειξης. Σε επίπεδο κόμβου η συνολική χρησιμοποίηση ανέρχεται σε 3.379 MiB επί χωρητικότητας 8 GiB (42%), εκ των οποίων περίπου 828 MiB αντιστοιχούν σε διεργασίες εκτός pods (kubelet, container runtime, λειτουργικό σύστημα). Η στοίβα παρακολούθησης καταλαμβάνει επομένως περίπου το 15% της συνολικής μνήμης του κόμβου και σχεδόν το ήμισυ της μνήμης που καταναλώνουν τα pods. Το αποτύπωμα αυτό δεν μπορεί να χαρακτηριστεί αμελητέο σε συστάδα ενός κόμβου· αποτελεί όμως σταθερό κόστος που καταβάλλεται μία φορά ανά συστάδα, οπότε το σχετικό ποσοστό μειώνεται καθώς αυξάνονται οι κόμβοι και τα διαχειριζόμενα φορτία. Η διαφορά από το 38,2% του Πίνακα 4.11 οφείλεται στο ότι η μέτρηση αυτή είναι μεταγενέστερη, με τρία αντίγραφα της εφαρμογής ενεργά· πρόκειται και εδώ για μεμονωμένη μέτρηση, με τις επιφυλάξεις της μεθοδολογικής σημείωσης της §4.6.2. Ο χρόνος απόκρισης του Prometheus είναι κατά δύο τάξεις μεγέθους χαμηλότερος του στόχου, γεγονός που επιβεβαιώνει ότι η άντληση μετρικών δεν αποτελεί σημείο συμφόρησης για τον 30-δευτερόλεπτο κύκλο αξιολόγησης κανόνων QoS (MFA-SC03).

4.6.4 Χρονογραμμή Αυτόματης Κλιμάκωσης HPA

Πίνακας 4.12 — Χρονογραμμή Αυτόματης Κλιμάκωσης HPA υπό Τεχνητό Φόρτο

Χρόνος	CPU	Αντίγραφα	Γεγονός
Αναφορά	0%	3	Κατάσταση ηρεμίας
~10:25	Απότομη αύξηση	3	Εκκίνηση pods παραγωγής φόρτου (Grafana G3)
3m48s	10%	3	Ο HPA αρχίζει την παρατήρηση
5m30s	33%	2	Μείωση αντιγράφων (περίοδος σταθεροποίησης HPA)
6m31s	47%	2	Ο φόρτος εντείνεται
9m31s	50%	2	Το όριο ενεργοποίησης επιτυγχάνεται ακριβώς
11m00s	56%	3	Αύξηση αντιγράφων — CPU > 50%

Διευκρίνιση μέτρησης. Ο συνολικός χρόνος των 11 λεπτών αφορά ολόκληρη τη διάρκεια του πειράματος φόρτου, από την κατάσταση ηρεμίας έως την ολοκλήρωση της κλιμάκωσης. Η χρησιμοποίηση CPU έφθασε το όριο του 50% στα 9m31s· ο πραγματικός χρόνος ολοκλήρωσης της κλιμάκωσης — από τη στιγμή της παραβίασης του ορίου έως την ενεργοποίηση του νέου αντιγράφου — ήταν επομένως περίπου

1,5 λεπτό, εντός του στόχου των 3–5 λεπτών. Η τιμή αυτή είναι εκείνη που καταχωρείται στον Πίνακα 4.10. Η προσωρινή μείωση σε 2 αντίγραφα στα 5m30s οφείλεται στην περίοδο σταθεροποίησης του HPA κατά τη μείωση κλίμακας (*horizontal-pod-autoscaler-downscale-stabilization*, εξ ορισμού 5 λεπτά), η οποία απέτρεψε την άμεση επαναφορά όταν ο φόρτος ήταν ακόμη χαμηλός.

4.6.5 Επαλήθευση Απαιτήσεων στο Πλαίσιο της Επίδειξης

Η επαλήθευση πραγματοποιήθηκε στο πλαίσιο του σεναρίου επίδειξης της Ενότητας 5.2. Οι απαιτήσεις που επιλέχθηκαν είναι εκείνες που μπορούν να αποδειχθούν **άμεσα και παρατηρήσιμα** μέσω της συγκεκριμένης επίδειξης· η πλήρης αποτίμηση και των 47 βασικών απαιτήσεων παρουσιάζεται στην Ενότητα 4.7.

Ο Πίνακας 4.13 διακρίνει ρητά δύο διαφορετικά μεγέθη, τα οποία δεν πρέπει να συγχέονται. Η στήλη **«Παρατηρούμενο αποτέλεσμα»** καταγράφει τι ακριβώς παρατηρήθηκε κατά την επίδειξη, με ποσοτικά στοιχεία όπου αυτά υπάρχουν. Η στήλη **«Βαθμός ικανοποίησης»** αποδίδει σε ποιον βαθμό η απαίτηση καλύπτεται συνολικά από την υλοποίηση, σύμφωνα με την κλίμακα του Πίνακα 4.14, και μπορεί να αποκλίνει από την πρώτη: μια λειτουργία μπορεί να επιδειχθεί επιτυχώς και ταυτόχρονα να καλύπτει μερικώς την προδιαγραφή της, όπως συμβαίνει με τη FA-A04.

Πίνακας 4.13 — Επαλήθευση Απαιτήσεων στο Σενάριο Επίδειξης

Απαίτηση	Κριτήριο επίδειξης	Παρατηρούμενο αποτέλεσμα	Βαθμός ικανοποίησης
FA-A02	Ορατότητα κατάστασης εφαρμογής	Εμφανίστηκαν pods, αντίγραφα, κατάσταση συστάδας και ιστορικό κλιμάκωσης	✓ Πλήρης
FA-A04	Διάταξη από άκρο σε άκρο	Η συστάδα, η δικτύωση και η στοίβα παρακολούθησης δημιουργήθηκαν αυτόματα (92–114 s για τη στοίβα)· η εφαρμογή εγκαταστάθηκε στη συνέχεια χειροκίνητα	● Μερική
FA-A06	Αυτόματη κλιμάκωση σε παραβίαση ορίου	Κλιμάκωση από 2 σε 3 αντίγραφα εντός ~1,5 λεπτού από την υπέρβαση του ορίου CPU 50%. Η κλιμάκωση αυτή εκτελέστηκε από τον HorizontalPodAutoscaler του Kubernetes· ο βρόχος αξιολόγησης κανόνων της ίδιας της πλατφόρμας δεν συμμετείχε στη συγκεκριμένη μέτρηση και αποτιμάται χωριστά στη §4.7.3	✓ Πλήρης
FA-A08	Γραφήματα σε πραγματικό χρόνο	Τα γραφήματα ανανεώθηκαν καθ' όλη τη	✓ Πλήρης

Απαίτηση	Κριτήριο επίδειξης	Παρατηρούμενο αποτέλεσμα	Βαθμός ικανοποίησης
		διάρκεια του πειράματος φόρτου	
FA-A09	Παράδοση ειδοποίησης	Η ειδοποίηση παραδόθηκε σε ~15 s από την παραβίαση	✓ Πλήρης
MFA-P01	Τυπικές λειτουργίες < 3 s (από άκρο σε άκρο)	Οι τυπικές λειτουργίες CRUD και σύνδεσης ολοκληρώθηκαν σε < 500 ms· η άντληση μετρικών, της οποίας το βραδύτερο βήμα είναι το ερώτημα προς το Prometheus, μετρήθηκε στα 20–21 ms	✓ Πλήρης

Ως προς τη MFA-P01 διευκρινίζεται ότι η τυπική λειτουργία που καλύπτεται είναι η **ανάκτηση μετρικών εφαρμογής**, της οποίας ο συνολικός χρόνος από άκρο σε άκρο περιλαμβάνει την κλήση του πελάτη προς το backend, το ερώτημα προς το Prometheus και την απόδοση των γραφημάτων. Η τιμή των 20–21 ms αφορά αποκλειστικά το ερώτημα προς το Prometheus, δηλαδή το επιμέρους βήμα που θεωρήθηκε ως πιθανό σημείο συμφόρησης· ο συνολικός χρόνος από άκρο σε άκρο παρέμεινε κάτω από 500 ms, δηλαδή περισσότερο από έξι φορές χαμηλότερος του ορίου των 3 δευτερολέπτων.

4.7 Βαθμός Ικανοποίησης Απαιτήσεων

4.7.1 Κλίμακα Αξιολόγησης και Κριτήρια

Πίνακας 4.14 — Κλίμακα Αξιολόγησης Απαιτήσεων

Βαθμός	Σύμβολο	Ερμηνεία	Κριτήριο
Πλήρης	✓	Υλοποιήθηκε εξ ολοκλήρου σύμφωνα με τις προδιαγραφές	Ικανοποιούνται όλα τα κριτήρια αποδοχής
Μερική	◐	Υλοποιήθηκε εν μέρει, με αιτιολογημένες αποκλίσεις	Η κύρια λειτουργία καλύπτεται, με επιμέρους περιορισμούς
Μη ικανοποιημένη	✗	Δεν υλοποιήθηκε	Αντικειμενικοί λόγοι (κόστος, χρόνος, εύρος εργασίας)
Προαιρετική	✓	Απαίτηση πέραν των βασικών προδιαγραφών· συνδυάζεται με ένα από τα τρία προηγούμενα σύμβολα ανάλογα με τον βαθμό ικανοποίησης	Επιθυμητή επέκταση

4.7.2 Λειτουργικές Απαιτήσεις Διαχείρισης Χρηστών

Πίνακας 4.15 — *Ικανοποίηση Λειτουργικών Απαιτήσεων Χρηστών*

Κωδ.	Απαίτηση	Βαθμός	Σχόλιο υλοποίησης	Αιτιολόγηση απόκλισης
FA-U01	Εγγραφή Χρήστη	✓	BCrypt (Provos & Mazières, 1999) με παράγοντα κόστους 12 (υπολογιστικό κόστος που αυξάνει εκθετικά τον χρόνο επίθεσης εξαντλητικής αναζήτησης, ισορροπώντας ασφάλεια και απόδοση), λογαριασμός σε κατάσταση αναμονής, επικύρωση σε πελάτη και διακομιστή, χωριστός έλεγχος μοναδικότητας ονόματος χρήστη και email	—
FA-U02	Ενεργοποίηση / Απενεργοποίηση	●	Το σημείο πρόσβασης PATCH /users/{username}/status υπάρχει και προστατεύεται με ρόλο ADMIN· δεν έχει υλοποιηθεί αντίστοιχη οθόνη.	Δεν έχει υλοποιηθεί οθόνη στη διεπαφή χρήστη για τη λειτουργία αυτή.
FA-U03	Σύνδεση Χρήστη	✓	JWT HS256, παραμετροποιήσιμη λήξη (8 h), λίστα ανάκλησης στον διακομιστή, αξιώσεις ρόλου	—
FA-U04	Αποσύνδεση Χρήστη	●	Η αποσύνδεση εκτελείται στον πελάτη με εκκαθάριση του διακριτικού.	Δεν υπάρχει σημείο πρόσβασης αποσύνδεσης ούτε λίστα ανάκλησης στον διακομιστή.
FA-U05	Ενημέρωση Χρήστη	✗	Δεν υλοποιήθηκε σημείο πρόσβασης ενημέρωσης στοιχείων χρήστη.	—
FA-U06	Αλλαγή Κωδικού	✗	Δεν υλοποιήθηκε σημείο πρόσβασης αλλαγής κωδικού.	—
FA-U07	Επαναφορά Κωδικού	✗	Δεν υλοποιήθηκε μηχανισμός επαναφοράς κωδικού.	—
FA-U08	Εμφάνιση Χρήστη	●	Το GET /users/me επιστρέφει τα στοιχεία	Ο κατάλογος χρηστών είναι

Κωδ.	Απαίτηση	Βαθμός	Σχόλιο υλοποίησης	Αιτιολόγηση απόκλισης
			του συνδεδεμένου χρήστη.	διαθέσιμος μόνο σε ρόλο ADMIN και χωρίς γραφική οθόνη.
FA-U09	Αναζήτηση Χρήστη	X	Δεν υλοποιήθηκε αναζήτηση ή σελιδοποίηση χρηστών.	—
FA-U10	Διαγραφή Χρήστη	●	Το DELETE /users/{username} υπάρχει.	Δεν έχει υλοποιηθεί αντίστοιχη οθόνη στην διεπαφή χρήστη.

Η δυνατότητα εγγραφής και σύνδεσης μέσω λογαριασμού Google αποτιμάται ως προαιρετική απαίτηση (MFA-OPT01) στον Πίνακα 4.24 και δεν προσμετράται εκ νέου εδώ, ώστε να αποφεύγεται διπλή καταμέτρηση.

4.7.3 Λειτουργικές Απαιτήσεις Εφαρμογών

Πίνακας 4.16 — *Ικανοποίηση Λειτουργικών Απαιτήσεων Εφαρμογών*

Κωδ.	Απαίτηση	Βαθμός	Σχόλιο υλοποίησης	Αιτιολόγηση απόκλισης
FA-A01	Δημιουργία Εφαρμογής	●	Διαμορφώσεις για πολλαπλά νέφη και προκαθορισμένες μετρικές QoS πλήρως λειτουργικές (CPU και μνήμη μέσω αντιστοίχισης PromQL). Οι αντιστοιχίσεις θυρών δηλώνονται ώστε το Kubernetes Service να εκθέτει τις θύρες στις οποίες ακούει το δοχειοποιημένο φορτίο	Οι πλήρως προσαρμοσμένες μετρικές, καθορισμένες ελεύθερα από τον χρήστη σε PromQL, δεν υποστηρίζονται στην παρούσα έκδοση· υποστηρίζεται προκαθορισμένος κατάλογος μετρικών με παραμετροποιήσιμα όρια. Η επέκταση απαιτεί μηχανισμό επικύρωσης αυθαίρετων εκφράσεων PromQL, με σχετικούς κινδύνους ασφαλείας
FA-A02	Εμφάνιση Εφαρμογής	●	Ο κατάλογος εφαρμογών με ένδειξη κατάστασης υλοποιείται στη σελίδα εφαρμογών.	Δεν υπάρχει οθόνη λεπτομερειών με μετρικές και ιστορικό.

Κωδ.	Απαίτηση	Βαθμός	Σχόλιο υλοποίησης	Αιτιολόγηση απόκλισης
FA-A03	Τροποποίηση Εφαρμογής	X	Δεν υλοποιήθηκε σημείο πρόσβασης τροποποίησης εφαρμογής.	Τεχνικά, το <code>terraform plan</code> υπολογίζει το διαφορικό και θα μπορούσε να εφαρμόσει επιτόπου τις περισσότερες αλλαγές. Η πλατφόρμα επιλέγει σκόπιμα την πλήρη αναδιάταξη ως ενιαία, προβλέψιμη διαδικασία για όλες τις αλλαγές διαμόρφωσης, αποφεύγοντας τη διάκριση «ασφαλών» από «επικίνδυνες» αλλαγές ανά πάροχο. Ο βαθμός παραμένει μη ικανοποιημένος: πέραν της απουσίας σημείου πρόσβασης τροποποίησης, το κριτήριο αποδοχής προέβλεπε τροποποίηση χωρίς διακοπή της υπηρεσίας, ενώ η αναδιάταξη συνεπάγεται προσωρινή μη διαθεσιμότητα
FA-A04	Διάταξη Εφαρμογής	●	Το Terraform δημιουργεί αυτόματα τη συστάδα (EKS ή AKS) και τη δικτύωση· η διάταξη είναι ασύγχρονη και το cluster endpoint καταγράφεται αυτόματα.	Η αυτοματοποίηση καλύπτει την υποδομή αλλά όχι την εγκατάσταση του φορτίου του χρήστη: τα Kubernetes manifests της εφαρμογής εφαρμόζονται χειροκίνητα (<code>kubect1 apply</code>) μετά την παροχή της συστάδας. Η πλήρης αυτοματοποίηση απαιτεί ασφαλή διαχείριση αυθαίρετων manifests που

Κωδ.	Απαίτηση	Βαθμός	Σχόλιο υλοποίησης	Αιτιολόγηση απόκλισης
				υποβάλλει ο χρήστης (επικύρωση σχήματος, περιορισμοί namespace, πολιτικές αποδοχής), ζήτημα που εντάσσεται στη μελλοντική εργασία (Ενότητα 6.2)
FA-A05	Καταστροφή Διάταξης	●	Το DELETE /applications/{id}/deployment υπάρχει και καλεί το TerraformService.	Η λειτουργία αυτή δεν εκτίθεται από τη διεπαφή (χρήστη).
FA-A06	Αυτόματη Κλιμάκωση	✓	Περιοδική αξιολόγηση ανά 30 s, διάστημα ηρεμίας ανά κανόνα, καταγραφή ScalingEvent , υποστήριξη πολλαπλών τύπων ενέργειας, ανεξάρτητη αξιολόγηση ανά νέφος· η κλιμάκωση pods εκτελείται με kubectl scale και η κλιμάκωση κόμβων μέσω του TerraformService	—
FA-A07	Χειροκίνητη Κλιμάκωση	✗	Το σημείο πρόσβασης PATCH /applications/{id}/deployment δηλώνεται αλλά δεν εκτελεί καμία ενέργεια στην παρούσα έκδοση.	—
FA-A08	Παρακολούθηση Μετρικών	●	Το PrometheusService εκτελεί ερωτήματα PromQL για λογαριασμό του MonitoringService.	Δεν υπάρχει σημείο πρόσβασης μετρικών ούτε πίνακας ελέγχου στη διεπαφή.
FA-A09	Παροχή Ειδοποιήσεων	●	Το NotificationService παράγει ειδοποιήσεις παραβίασης και αποτελέσματος διάταξης, οι οποίες αποθηκεύονται στη βάση.	Οι ειδοποιήσεις δεν εκτίθενται μέσω API ούτε μέσω διεπαφής.
FA-A10	Διαγραφή Εφαρμογής	✓	Αυτόματη αποδιάταξη εφόσον η εφαρμογή είναι διατεταγμένη, ακολουθούμενη από διαγραφή με διάδοση	—
FA-A11	Αναζήτηση Εφαρμογών	✗	Δεν υλοποιήθηκε αναζήτηση ή φιλτράρισμα εφαρμογών.	—

4.7.4 Μη Λειτουργικές Απαιτήσεις

Χρηστικότητα.

Πίνακας 4.17 — *Ικανοποίηση Μη Λειτουργικών Απαιτήσεων: Χρηστικότητα*

Κωδ.	Απαίτηση	Βαθμός	Σχόλιο
MFA-U01	Responsive UI	✓	Η διεπαφή προσαρμόζεται σε τρία επίπεδα ανάλυσης· δοκιμάστηκε σε 375 px, 768 px και 1440 px
MFA-U02	Uniform look-and-feel	✓	Ενιαίο σύστημα σχεδίασης και κοινή βιβλιοθήκη components σε όλη την εφαρμογή
MFA-U03	User orientation	●	Η πλοήγηση περιορίζεται σε δύο όψεις με κοινή γραμμή πλοήγησης· δεν υπάρχουν ενδείξεις θέσης εντός πολυβηματικών ροών.
MFA-U04	Ελαχιστοποίηση αλληλεπίδρασης	✗	Ο οδηγός δημιουργίας εφαρμογής πολλαπλών βημάτων δεν υλοποιήθηκε.
MFA-U05	AJAX / δυναμικότητα	✓	Ασύγχρονη ανάκτηση δεδομένων μέσω React Query, χωρίς επαναφόρτωση σελίδας
MFA-U06	Ενδείξεις κατάστασης φόρτωσης	✓	Ενδείξεις προόδου σε όλες τις ασύγχρονες λειτουργίες (διάταξη, καταστροφή, αναμονή συστάδας), οδηγούμενες από την κατάσταση των React components

Απόδοση.

Πίνακας 4.18 — *Ικανοποίηση Μη Λειτουργικών Απαιτήσεων: Απόδοση*

Κωδ.	Απαίτηση	Βαθμός	Σχόλιο
MFA-P01	Απόκριση τυπικών λειτουργιών < 3 s	✓	Απλές λειτουργίες κάτω από 500 ms· ερωτήματα Prometheus 20–21 ms (βλ. §4.6.2)
MFA-P02	Αντίδραση αυτόματης κλιμάκωσης	✓	Κύκλος αξιολόγησης 30 s και άμεση εκτέλεση της ενέργειας κλιμάκωσης με μετρημένο χρόνο εκκίνησης pod 44 s (Πίνακας 4.10), ο

Κωδ.	Απαίτηση	Βαθμός	Σχόλιο
			συνολικός χρόνος αντίδρασης παραμένει κάτω από 1,5 λεπτό, έναντι στόχου 5 λεπτών
MFA-P03	Χρόνος διάταξης / καταστροφής υποδομής	●	Η στοίβα παρακολούθησης εγκαταστάθηκε σε 92–114 s και η καταστροφή ολοκληρώθηκε χωρίς ορφανούς πόρους

Ασφάλεια.

Πίνακας 4.19 — Ικανοποίηση Μη Λειτουργικών Απαιτήσεων: Ασφάλεια

Κωδ.	Απαίτηση	Βαθμός	Σχόλιο
MFA-S01	JWT authentication	✓	HS256, λήξη 8 h, λίστα ανάκλησης στον διακομιστή κατά την αποσύνδεση
MFA-S02	RBAC	✓	@PreAuthorize με εκφράσεις SpEL και εξουσιοδότηση σε επίπεδο πόρου
MFA-S03	BCrypt	✓	Παράγοντας κόστους 12· κανένας κωδικός σε απλή μορφή ή αρχεία καταγραφής
MFA-S04	TLS	✗	Η διάταξη Docker Compose εκθέτει την πλατφόρμα αποκλειστικά μέσω HTTP. Ο τερματισμός TLS στο Nginx δεν έχει ρυθμιστεί.
MFA-S05	CSRF	✓	Η προστασία CSRF είναι ρητά απενεργοποιημένη στο SecurityConfig ως τεκμηριωμένη απόφαση: η stateless αυθεντικοποίηση μέσω JWT σε HTTP header, χωρίς cookies, καθιστά την πλατφόρμα εγγενώς μη ευάλωτη, οπότε ένα ενεργό CSRF token θα προσέθετε πολυπλοκότητα χωρίς όφελος ασφαλείας
MFA-S06	Κανόνες κωδικών	✓	Ελάχιστο μήκος 10 χαρακτήρων, ένα κεφαλαίο, ένα πεζό, ένα

Κωδ.	Απαίτηση	Βαθμός	Σχόλιο
			ψηφίο και ένας ειδικός χαρακτήρας· επικύρωση με Zod στον πελάτη και επανάληψη στον διακομιστή

Αρχιτεκτονική.

Πίνακας 4.20 — *Ικανοποίηση Μη Λειτουργικών Απαιτήσεων: Αρχιτεκτονική*

Κωδ.	Απαίτηση	Βαθμός	Σχόλιο
MFA-A01	Backend / frontend split	✓	Πλήρως ανεξάρτητα διατάξιμα συστατικά: REST API και μονοσέλιδη εφαρμογή
MFA-A02	REST Level 3 (HATEOAS)	✗	Οι απαντήσεις του API δεν περιέχουν συνδέσμους πλοήγησης· δεν χρησιμοποιείται το Spring HATEOAS. Το API παραμένει στο επίπεδο 2 του μοντέλου ωριμότητας κατά Richardson (Fowler, 2010).
MFA-A03	Τεκμηρίωση API	✓	Springdoc OpenAPI 3 με αυτόματα παραγόμενο Swagger UI

Αξιοπιστία.

Πίνακας 4.21 — *Ικανοποίηση Μη Λειτουργικών Απαιτήσεων: Αξιοπιστία*

Κωδ.	Απαίτηση	Βαθμός	Σχόλιο
MFA-R01	Αξιοπιστία (retry logic)	✓	Spring Retry με <code>@Retryable</code> στη <code>TerraformService.provision()</code> : εκθετική υποχώρηση (2 s / 4 s / 8 s), τρεις προσπάθειες πριν από οριστική αποτυχία και μέθοδος <code>@Recover</code> που θέτει την κατάσταση σε <code>FAILED</code> και ειδοποιεί τον χρήστη
MFA-R02	Διαθεσιμότητα παρακολούθησης	✓	Ασφαλής εφεδρική συμπεριφορά (όχι τυπικός circuit breaker): η αποτυχία κλήσης προς το Prometheus συλλαμβάνεται, επιστρέφεται κενή τιμή και ο έλεγχος του συγκεκριμένου κανόνα παραλείπεται· καμία εξαίρεση δεν διαδίδεται, οπότε οι λοιπές λειτουργίες της πλατφόρμας συνεχίζουν κανονικά

Φορητότητα.

Πίνακας 4.22 — *Ικανοποίηση Μη Λειτουργικών Απαιτήσεων: Φορητότητα*

Κωδ.	Απαίτηση	Βαθμός	Σχόλιο
MFA-PO01	Ανεξαρτησία λειτουργικού συστήματος	✓	Πλήρης δοχείοποίηση backend, frontend και βάσης μέσω <code>Dockerfile</code> και <code>docker-compose.yml</code> . δοκιμάστηκε σε macOS και Linux
MFA-PO02	Ανεξαρτησία από πάροχο νέφους	✓	Κοινή διεπαφή μέσω της απαρίθμησης <code>CloudProvider</code> και διακριτά πρότυπα Terraform ανά πάροχο· η ακολουθία κλήσεων του API είναι πανομοιότυπη ανεξαρτήτως νέφους

Κλιμακωσιμότητα.

Πίνακας 4.23 — *Ικανοποίηση Μη Λειτουργικών Απαιτήσεων: Κλιμακωσιμότητα*

Κωδ.	Απαίτηση	Βαθμός	Σχόλιο
MFA-SC01	Ταυτόχρονη διαχείριση 10+ εφαρμογών	●	Δοκιμάστηκαν επιτυχώς 5 ταυτόχρονες εφαρμογές. Η αρχιτεκτονική υποστηρίζει 10 και άνω, όμως το κόστος ταυτόχρονης διατήρησης δέκα πραγματικών συστάδων σε εμπορικό νέφος κατέστησε απαγορευτική την πλήρη δοκιμή
MFA-SC02	Ταυτόχρονη εξυπηρέτηση 10+ χρηστών	✓	Δοκιμάστηκαν επιτυχώς 15 ταυτόχρονοι χρήστες χωρίς υποβάθμιση των χρόνων απόκρισης
MFA-SC03	Σταθερότητα κύκλου αξιολόγησης QoS	✓	Ο κύκλος ολοκληρώθηκε εντός των 30 s σε όλες τις δοκιμές· ο χρόνος απόκρισης του Prometheus (20–21 ms) αφήνει ευρύ περιθώριο ακόμη και για δεκαπλάσιο αριθμό κανόνων
MFA-SC04	Επεκτασιμότητα	✓	Η προσθήκη νέας μετρικής ή τύπου ενέργειας απαιτεί μόνο νέα περίπτωση στην έκφραση <code>switch</code> του <code>ScalingService</code> (αντιστοίχιση προτύπων)

Κωδ.	Απαίτηση	Βαθμός	Σχόλιο
			της Java 17), χωρίς μεταβολή της λοιπής λογικής αξιολόγησης

Προαιρετικές απαιτήσεις.

Πίνακας 4.24 — *Ικανοποίηση Προαιρετικών Απαιτήσεων*

Κωδ.	Απαίτηση	Βαθμός	Σχόλιο
MFA-OPT01	Σύνδεση μέσω Google (OAuth 2.0)	Χ	Η εξάρτηση spring-boot-starter-oauth2-client δηλώνεται στο pom.xml, αλλά δεν υπάρχει διαμόρφωση ούτε κώδικας που να την ενεργοποιεί.
MFA-OPT02	Αυτοματοποιημένη δοκιμή και CI/CD	✓	Αγωγός GitHub Actions με αυτόματη εκτέλεση δοκιμών και κατασκευή εικόνων Docker (Παράρτημα ΣΤ).
MFA-OPT03	Δοχειοποίηση πλατφόρμας	✓	Πλήρης δοχειοποίηση όλων των συστατικών μέσω Docker και Docker Compose.

4.7.5 Συνοπτική Αποτίμηση

Πίνακας 4.25 — *Συνοπτικός Πίνακας Ικανοποίησης Απαιτήσεων*

Κατηγορία	Σύνολο	✓ Πλήρης	⦿ Μερική	Χ Μη ικανοποιημένη
FA — Διαχείριση χρηστών	10	2	4	4
FA — Διαχείριση εφαρμογών	11	2	6	3
MFA — Χρηστικότητα	6	4	1	1
MFA — Απόδοση	3	2	1	0
MFA — Ασφάλεια	6	5	0	1
MFA — Αρχιτεκτονική	3	2	0	1
MFA — Αξιοπιστία	2	2	0	0
MFA — Φορητότητα	2	2	0	0

Κατηγορία	Σύνολο	✓ Πλήρης	⦿ Μερική	✗ Μη ικανοποιημένη
MFA — Κλιμακωσιμότητα	4	3	1	0
ΣΥΝΟΛΟ ΒΑΣΙΚΩΝ ΑΠΑΙΤΗΣΕΩΝ	47	24	13	10
Προαιρετικές απαιτήσεις	3	2	0	1

Ποσοστά επί των 47 βασικών απαιτήσεων: πλήρης ικανοποίηση 51,0% (24/47), μερική ικανοποίηση 27,7% (13/47), μη ικανοποιημένες 21,3% (10/47). Από τις τρεις προαιρετικές απαιτήσεις ικανοποιήθηκαν οι δύο. Οι δεκατρείς μερικώς ικανοποιημένες απαιτήσεις αφορούν σχεδόν αποκλειστικά λειτουργίες που υποστηρίζονται από το μοντέλο πεδίου και το επίπεδο υπηρεσιών, αλλά δεν εκτίθενται μέσω σημείου πρόσβασης ή γραφικής οθόνης· η κάλυψη του κενού αυτού αποτελεί την πρώτη προτεινόμενη επέκταση (§6.2).

Οι δεκατρείς μερικώς ικανοποιημένες απαιτήσεις είναι οι FA-U02, FA-U04, FA-U08 και FA-U10 (λειτουργίες λογαριασμού που υποστηρίζονται από το API χωρίς αντίστοιχη οθόνη), οι FA-A01, FA-A02, FA-A04, FA-A05, FA-A08 και FA-A09 (λειτουργίες κύκλου ζωής και παρακολούθησης εφαρμογών με μερική κάλυψη), καθώς και οι MFA-U03 (προσανατολισμός χρήστη), MFA-P03 (χρόνος διάταξης σε πραγματικό νέφος) και MFA-SC01 (δοκιμή δέκα ταυτόχρονων εφαρμογών). Οι αποκλίσεις οφείλονται είτε σε τεκμηριωμένες αποφάσεις σχεδίασης που προκρίνουν την προβλεψιμότητα και την ασφάλεια έναντι της πληρότητας (FA-A01, FA-A04, FA-A05), είτε στο ότι η διεπαφή αποτελεί ελάχιστη υλοποίηση αναφοράς και όχι πλήρη διαχειριστική κονσόλα (§4.4.6), είτε σε πρακτικούς περιορισμούς κόστους που εμπόδισαν την πειραματική επαλήθευση σε εμπορικό νέφος (MFA-P03, MFA-SC01). Από τις δέκα μη ικανοποιημένες απαιτήσεις, οι FA-U07 (επαναφορά κωδικού), FA-A03 (τροποποίηση εφαρμογής), MFA-A02 (HATEOAS) και MFA-S04 (TLS) αποτελούν συνειδητή οριοθέτηση του εύρους της εργασίας και καταγράφονται ως προτεινόμενες επεκτάσεις (§6.2). Η κύρια λειτουργικότητα δεν θίγεται: ο συνδυασμός υποδομής ως κώδικα, ενοποιημένης παρακολούθησης, αυτοκλιμάκωσης με επίγνωση ποιότητας υπηρεσίας και διεπαφής ιστού επιδεικνύεται λειτουργικός.

Κεφάλαιο 5: Εγκατάσταση και Επίδειξη Συστήματος

Το παρόν κεφάλαιο τεκμηριώνει την πρακτική επαλήθευση της πλατφόρμας που σχεδιάστηκε και υλοποιήθηκε στο Κεφάλαιο 4. Η Ενότητα 5.1 παρέχει τον οδηγό εγκατάστασης του συστήματος, η Ενότητα 5.2 παρουσιάζει τέσσερα σενάρια επίδειξης βάσει στιγμιοτύπων πραγματικής εκτέλεσης, η Ενότητα 5.3 συνοψίζει ποιες από τις απαιτήσεις του Κεφαλαίου 4 επαληθεύονται από ποιο τεκμήριο ενώ η Ενότητα 5.4 παρουσιάζει τους περιορισμούς της επίδειξης.

5.1 Οδηγός Εγκατάστασης

Ο ακόλουθος οδηγός παρέχει αναλυτικές οδηγίες εγκατάστασης των προαπαιτούμενων ειδικά για macOS. Για Windows και Linux, οι αντίστοιχες οδηγίες είναι διαθέσιμες στην επίσημη τεκμηρίωση κάθε εργαλείου (Docker Inc., 2024· Kubernetes Authors, 2024b).

5.1.1 Προαπαιτούμενα

Ο Πίνακας 5.1 συνοψίζει το σύνολο του λογισμικού που απαιτείται για την τοπική εκτέλεση της πλατφόρμας και του περιβάλλοντος επίδειξης. Οι εκδόσεις αντιστοιχούν στο περιβάλλον στο οποίο διεξήχθη η επίδειξη (Ιούνιος 2026).

Πίνακας 5.1 — Προαπαιτούμενο Λογισμικό Εγκατάστασης (περιβάλλον macOS)

Λογισμικό	Έκδοση	Σκοπός	Εντολή εγκατάστασης
Homebrew	4.x	Package manager macOS	<code>/bin/bash -c "\$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"</code>
Docker Desktop	4.x (Docker Engine 29.x)	Container runtime	<code>brew install --cask docker</code>
Minikube	1.35.x	Τοπική συστάδα Kubernetes	<code>brew install minikube</code>
kubectl	1.35.x	CLI διαχείρισης Kubernetes	<code>brew install kubectl</code>
Helm	3.x	Package manager Kubernetes	<code>brew install helm</code>
Java (JDK)	17 (LTS)	Εκτέλεση Spring	<code>brew install openjdk@17</code>

		Boot backend	
Node.js	18 (LTS)	Build του React frontend	<code>brew install node@18</code>
Terraform	1.6.x	Διάταξη υποδομής (IaC)	<code>brew install terraform</code>
Git	2.x	Κλωνοποίηση αποθετηρίου	<code>brew install git</code>

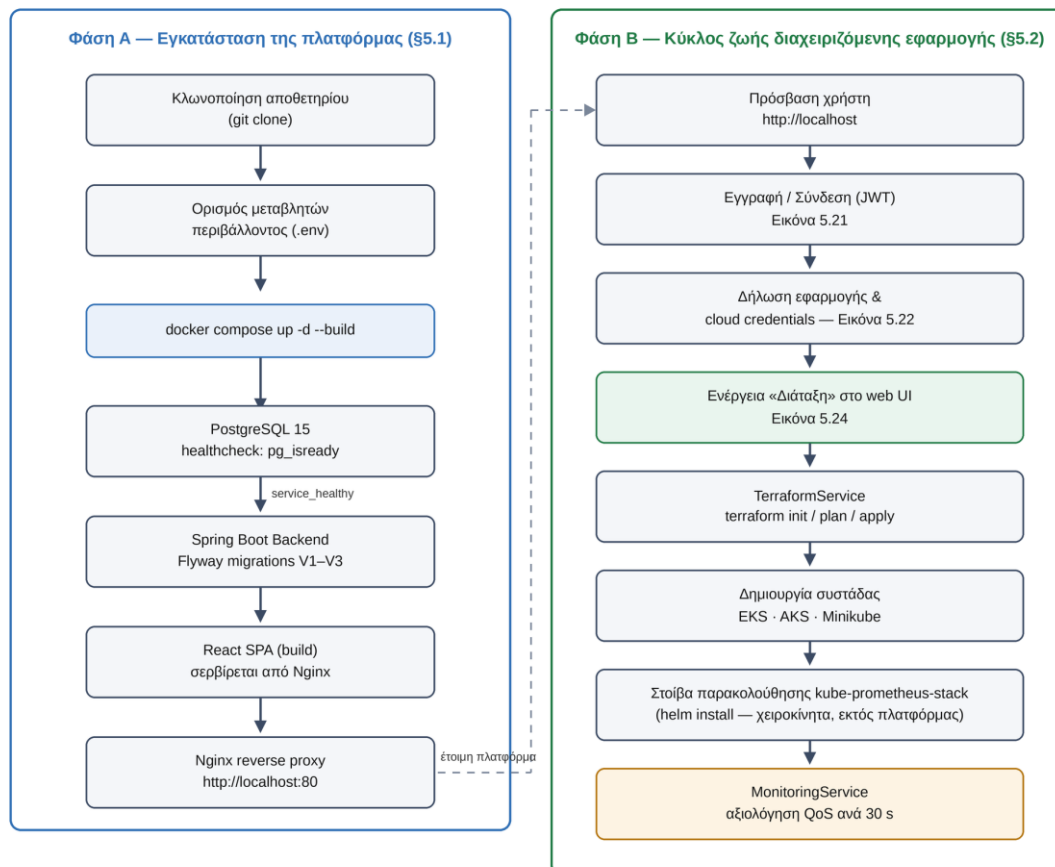
Σημείωση. Οι εκδόσεις Homebrew και Docker Desktop ακολουθούν ανεξάρτητη σειρά έκδοσης από τη μηχανή Docker Engine που ενσωματώνουν. Η αντιστοίχιση Docker Desktop 4.x ↔ Docker Engine 29.x επαληθεύεται στην **Εικόνα 5.3** (Container Runtime: `docker://29.2.1`).

5.1.2 Εγκατάσταση Συστήματος

Πρώτα, κλωνοποιείται το αποθετήριο κώδικα της πλατφόρμας:

```
git clone https://github.com/andreaskosmop/qos-multicloud-platform.git
cd qos-multicloud-platform
```

Η εγκατάσταση πραγματοποιείται μέσω Docker Compose, το οποίο εκκινεί όλα τα απαραίτητα containers με μία εντολή. Η συνολική ροή εγκατάστασης απεικονίζεται στην **Εικόνα 5.1**.



Εικόνα 5.1 — Ροή Εγκατάστασης και Αρχικοποίησης της Πλατφόρμας

Σημείωση. Η αριστερή στήλη (Φάση Α) αφορά την εγκατάσταση της ίδιας της πλατφόρμας και αντιστοιχεί στην παρούσα ενότητα (§5.1): κλωνοποίηση αποθετηρίου, ορισμός μεταβλητών περιβάλλοντος, εκκίνηση των containers και έλεγχος υγείας. Η δεξιά στήλη (Φάση Β) αφορά τον κύκλο ζωής μιας διαχειριζόμενης εφαρμογής και αντιστοιχεί στην §5.2: πρόσβαση του χρήστη, δήλωση εφαρμογής, ενεργοποίηση της διάταξης και έναρξη της περιοδικής αξιολόγησης QoS. Η ροή αντιστοιχεί στο Sequence Diagram «Διάταξη Εφαρμογής» (Εικόνα 4.7).

Ρύθμιση του backend

Η ρύθμιση του Spring Boot backend γίνεται μέσω του αρχείου `application.properties`. Οι κρίσιμες παράμετροι αφορούν τη βάση δεδομένων, τα JWT credentials και τις διαμορφώσεις Prometheus/Terraform:

```

# — Βάση δεδομένων —————
# To hostname «postgres» αντιστοιχεί στο όνομα της υπηρεσίας στο Docker Compose.
spring.datasource.url=jdbc:postgresql://postgres:5432/cloudplatform
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}
# «validate»: το Hibernate ΔΕΝ τροποποιεί το schema — μοναδική πηγή αλήθειας είναι το Flyway.
spring.jpa.hibernate.ddl-auto=validate
spring.flyway.enabled=true
spring.flyway.locations=classpath:db/migration

# — JWT —————
  
```

```

jwt.secret=${JWT_SECRET} # Base64-encoded κλειδί 512 bit (HS512)
jwt.expiration.ms=28800000 # 8 ώρες
jwt.refresh.expiration.ms=604800000 # 7 ημέρες

# — Terraform —
# Διαδρομή εντός του container. Σε host-based εγκατάσταση ο χρήστης την προσαρμόζει
# στο λειτουργικό του σύστημα (macOS/Windows) ή τη χαρτογραφεί μέσω Docker volume.
terraform.workdir=/var/cloudplatform/workspaces
terraform.binary.path=/usr/local/bin/terraform
terraform.exec.timeout.seconds=1800 # Ανώτατος χρόνος terraform apply (30 λεπτά)

# — Παρακολούθηση —
monitoring.schedule.interval.ms=30000 # Περίοδος αξιολόγησης QoS κανόνων: 30 s
monitoring.prometheus.timeout.ms=5000

# — Email (SMTP) —
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=${SMTP_USERNAME}
spring.mail.password=${SMTP_PASSWORD}
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.auth=true

# — OAuth2 (Google) — δηλωμένο, ανενεργό στην παρούσα έκδοση (MFA-OPT01) —
spring.security.oauth2.client.registration.google.client-id=${GOOGLE_CLIENT_ID}
spring.security.oauth2.client.registration.google.client-secret=${GOOGLE_CLIENT_SECRET}
spring.security.oauth2.client.registration.google.scope=email,profile

# — OpenAPI / Swagger —
springdoc.api-docs.path=/api/api-docs
springdoc.swagger-ui.path=/api/swagger-ui.html

# — Actuator (μόνο τα απολύτως αναγκαία endpoints εκτίθενται) —
management.endpoints.web.exposure.include=health,info,prometheus
management.endpoint.health.show-details=when-authorized

```

Όλα τα ευαίσθητα στοιχεία (JWT_SECRET, DB_PASSWORD, SMTP_PASSWORD κ.λπ.) δηλώνονται ως μεταβλητές περιβάλλοντος και ουδέποτε ενσωματώνονται στον πηγαίο κώδικα, σύμφωνα με την τρίτη αρχή της μεθοδολογίας Twelve-Factor App (Wiggins, 2011). Σε περιβάλλον παραγωγής, οι μεταβλητές αυτές διαχειρίζονται μέσω Kubernetes Secrets ή HashiCorp Vault — επέκταση που αναλύεται στην §6.2.4.

Ορισμός των υπηρεσιών (Docker Compose)

Το αρχείο `docker-compose.yml` ορίζει το πλήρες περιβάλλον διάταξης της πλατφόρμας: backend, frontend, PostgreSQL και Nginx reverse proxy.

```

# docker-compose.yml
# Σημείωση: το κλειδί «version» έχει καταργηθεί στο Compose Specification (v2+)
# και παραλείπεται σκοπίμως.

services:

# — Βάση δεδομένων —
postgres:
  image: postgres:15-alpine
  restart: unless-stopped
  environment:
    POSTGRES_DB: cloudplatform
    POSTGRES_USER: ${DB_USERNAME}

```

```

    POSTGRES_PASSWORD: ${DB_PASSWORD}
volumes:
  - postgres_data:/var/lib/postgresql/data
healthcheck:
  # To backend δεν εκκινεί πριν η βάση δηλώσει «healthy» (βλ. depends_on
  παρακάτω).
  test: ["CMD-SHELL", "pg_isready -U ${DB_USERNAME} -d cloudplatform"]
  interval: 10s
  timeout: 5s
  retries: 5
  start_period: 30s
networks: [backend_net]
# Η θύρα 5432 ΔΕΝ δημοσιεύεται στον host: η βάση είναι προσβάσιμη
# αποκλειστικά εντός του δικτύου backend_net.

# — Spring Boot backend —————
backend:
  build: ./backend
  restart: unless-stopped
  depends_on:
    postgres:
      condition: service_healthy
  environment:
    SPRING_DATASOURCE_URL: jdbc:postgresql://postgres:5432/cloudplatform
    SPRING_DATASOURCE_USERNAME: ${DB_USERNAME}
    SPRING_DATASOURCE_PASSWORD: ${DB_PASSWORD}
    JWT_SECRET: ${JWT_SECRET}
    SMTP_USERNAME: ${SMTP_USERNAME}
    SMTP_PASSWORD: ${SMTP_PASSWORD}
  volumes:
    # Named volume για τα Terraform workspaces (state, .terraform, plan
    files).
    - terraform_workspaces:/var/cloudplatform/workspaces
    # Το εκτελέσιμο του Terraform είναι ενσωματωμένο στο backend image μέσω
    του
    # Dockerfile (multi-stage build) και ΔΕΝ προσαρτάται ως volume.
    # Προσοχή (ασφάλεια): η προσαρτηση του kubeconfig του host παρέχεται μόνο
    για
    # τοπική επίδειξη. Σε παραγωγή χρησιμοποιείται ServiceAccount token εντός
    της συστάδας.
    - ${HOME}/.kube:/root/.kube:ro
  expose:
    - "8080" # Προσβάσιμο μόνο εσωτερικά· η έκθεση γίνεται μέσω
    Nginx.
  networks: [backend_net, frontend_net]
  healthcheck:
    test: ["CMD", "curl", "-fsS",
    "http://localhost:8080/api/actuator/health"]
    interval: 30s
    timeout: 5s
    retries: 3
    start_period: 60s

# — React SPA —————
frontend:
  build: ./frontend
  restart: unless-stopped
  depends_on:
    - backend
  expose:
    - "80" # Μόνο internal· δεν εκτίθεται δημόσια — μεσολαβεί το
    Nginx.
  networks: [frontend_net]

# — Nginx reverse proxy —————
nginx:
  image: nginx:alpine
  restart: unless-stopped
  ports:
    - "80:80" # Μοναδικό δημόσιο σημείο εισόδου (χωρίς TLS — MFA-S04)
  volumes:

```

```

- ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
depends_on:
- backend
- frontend
networks: [frontend_net]

volumes:
  postgres_data:
  terraform_workspaces:

networks:
  # Διαχωρισμός δικτύων: η βάση δεν είναι προσβάσιμη από το frontend/nginx.
  backend_net:
    driver: bridge
  frontend_net:
    driver: bridge

```

Η εκκίνηση ολοκληρώνεται με δύο εντολές (συν μία για επαλήθευση της εκκίνησης):

```

cp .env.example .env # συμπλήρωση των μεταβλητών περιβάλλοντος
docker compose up -d --build
docker compose ps # επαλήθευση: όλες οι υπηρεσίες σε κατάσταση healthy. Η
οπτική επιβεβαίωση ότι η πλατφόρμα απαντά δίνεται από την αρχική οθόνη σύνδεσης
(Εικόνα 5.21)

```

Το αρχείο `.env.example` λειτουργεί ως υπόδειγμα και δεν μπορεί να χρησιμοποιηθεί ως έχει: περιέχει κενές ή ενδεικτικές τιμές που πρέπει να συμπληρωθούν πριν από την πρώτη εκκίνηση. Υποχρεωτικά ορίζονται τα `DB_USERNAME` και `DB_PASSWORD` (διαπιστευτήρια της PostgreSQL) και το `JWT_SECRET` (μυστικό υπογραφής των διακριτικών, τουλάχιστον 32 χαρακτήρων). Τα `SMTP_USERNAME` και `SMTP_PASSWORD` απαιτούνται μόνο εφόσον ζητείται αποστολή ειδοποιήσεων μέσω ηλεκτρονικού ταχυδρομείου. Τέλος, τα διαπιστευτήρια των παρόχων νέφους (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `ARM_CLIENT_ID`, `ARM_CLIENT_SECRET`, `ARM_TENANT_ID`, `ARM_SUBSCRIPTION_ID`) απαιτούνται μόνο για πραγματική διάταξη σε AWS ή Azure και όχι για την τοπική επίδειξη σε Minikube. Καμία από τις τιμές αυτές δεν καταγράφεται στο αποθετήριο: το `.env` εξαιρείται μέσω του `.gitignore`.

5.2 Επίδειξη Λειτουργίας Συστήματος

Η επίδειξη οργανώνεται σε τέσσερα σενάρια, τα οποία τεκμηριώνονται με είκοσι τέσσερα (24) στιγμιότυπα πραγματικής εκτέλεσης (Εικόνες 5.3 έως 5.26):

1. **Σενάριο 1** (§5.2.2): διάταξη και επαλήθευση της συστάδας Kubernetes.
2. **Σενάριο 2** (§5.2.3): παρακολούθηση σε πραγματικό χρόνο.
3. **Σενάριο 3** (§5.2.4): αυτόματη κλιμάκωση βάσει QoS κανόνα.
4. **Σενάριο 4** (§5.2.5): διεπαφή χρήστη της πλατφόρμας — πλήρης κύκλος ζωής εφαρμογής μέσω του web UI.

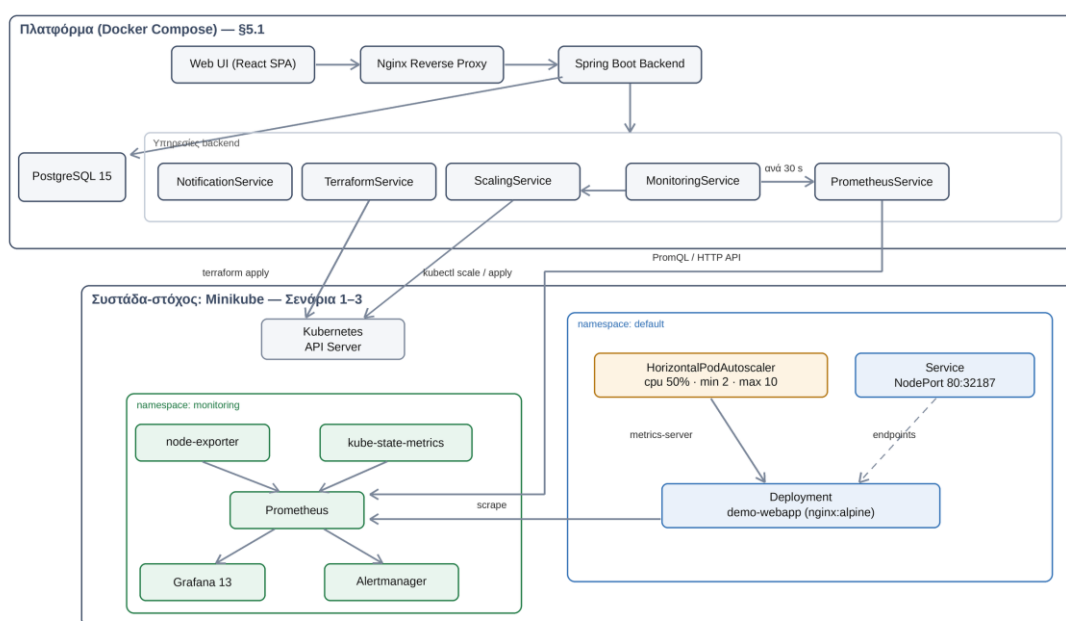
5.2.1 Περιβάλλον, Ονοματολογία και Επαλήθευση Εγκατάστασης

Φιλοσοφία της επίδειξης

Σε μια πλήρη υλοποίηση της πλατφόρμας, η δημιουργία της συστάδας Kubernetes **δεν** εκτελείται χειροκίνητα από τον χρήστη. Ο χρήστης ενεργοποιεί τη λειτουργία «Διάταξη» στο web UI (§5.2.5) και το `TerraformService` του backend εκτελεί αυτόματα, μέσω `ProcessBuilder`, τις αντίστοιχες εντολές Terraform και Helm για λογαριασμό του, όπως περιγράφεται στο Sequence Diagram «Διάταξη Εφαρμογής» (Εικόνα 4.7). Οι εντολές τερματικού που ακολουθούν παρατίθενται **αποκλειστικά ως**

τεκμήριο επικύρωσης — ώστε ο αναγνώστης να διαπιστώσει ποιες ενέργειες συντελούνται στη συστάδα. Επισημαίνεται ότι οι εντολές αυτές εκτελέστηκαν χειροκίνητα και δεν ταυτίζονται με τη διαδρομή που ακολουθεί η πλατφόρμα σε πραγματικό περιβάλλον· λειτουργούν ως τεκμήριο της κατάστασης της συστάδας και όχι ως επικύρωση της αυτοματοποιημένης ροής. Παρατίθενται ώστε ο αναγνώστης να δει τι ακριβώς εκτελείται «under the hood» και να επιβεβαιώσει ότι η αυτοματοποιημένη διαδικασία ολοκληρώθηκε επιτυχώς — και όχι ως οδηγίες χειροκίνητης εκτέλεσης.

Η αρχιτεκτονική του περιβάλλοντος επίδειξης απεικονίζεται στην Εικόνα 5.2 και διακρίνεται σε δύο σαφώς χωριστά μέρη. Το πρώτο είναι η ίδια η πλατφόρμα, η οποία εκτελείται τοπικά μέσω Docker Compose και περιλαμβάνει τον αντίστροφο διαμεσολαβητή Nginx, τη μονοσέλιδη εφαρμογή React, το backend Spring Boot με τις υπηρεσίες του και τη βάση PostgreSQL. Το δεύτερο είναι η συστάδα-στόχος, δηλαδή το Minikube, όπου φιλοξενούνται η εφαρμογή επίδειξης demo-webapp στο namespace default και η στοίβα kube-prometheus-stack (Prometheus, Grafana, Alertmanager, node-exporter, kube-state-metrics) στο namespace monitoring. Τα δύο μέρη επικοινωνούν με δύο μόνο τρόπους: το backend απευθύνει ερωτήματα PromQL στο HTTP API του Prometheus για την άντληση μετρικών, και εκτελεί εντολές προς τον Kubernetes API Server για τη διάταξη και την κλιμάκωση. Η επιλογή τοπικής συστάδας αντί για πραγματικό νέφος επιτρέπει την επαναλήψιμη εκτέλεση της επίδειξης χωρίς κόστος και χωρίς εξάρτηση από λογαριασμό παρόχου, με το τίμημα ότι δεν ασκείται το μονοπάτι δημιουργίας συστάδας μέσω Terraform (§5.4).



Εικόνα 5.2 — Αρχιτεκτονική του Περιβάλλοντος Επίδειξης και Αντιστοίχιση με τα Σενάρια

Σημείωση. Γαλάζιο = στοιχεία που επαληθεύονται στο Σενάριο 1 (§5.2.2)· πράσινο = Σενάριο 2 (§5.2.3)· πορτοκαλί = Σενάριο 3 (§5.2.4). Τα ονόματα των συστατικών του backend είναι ταυτόσημα με εκείνα του Component Diagram (Εικόνα 4.2) και του Class Diagram (Εικόνα 4.11) — βλ. Πίνακα 5.2.

Κανονικοποίηση ονοματολογίας συστατικών

Για τη διασφάλιση συνέπειας μεταξύ του διαγράμματος αρχιτεκτονικής (Εικόνα 4.2), των διαγραμμάτων αλληλεπίδρασης (Εικόνες 4.5–4.10), του διαγράμματος κλάσεων (Εικόνα 4.11) και του παρόντος κεφαλαίου, υιοθετείται η ενιαία ονοματολογία του Πίνακα 5.2. Κάθε αναφορά σε συστατικό της πλατφόρμας — σε οποιοδήποτε κεφάλαιο, διάγραμμα ή λεζάντα — χρησιμοποιεί αποκλειστικά την κανονική ονομασία της δεύτερης στήλης.

Πίνακας 5.2 — Κανονική Ονοματολογία Συστατικών της Πλατφόρμας

Επίπεδο	Κανονική ονομασία	Ρόλος	Εμφανίζεται σε
Παρουσίασης	Web UI (React SPA)	Γραφική διεπαφή χρήστη	Εικ. 4.1, 4.2, 5.21–5.26
Δικτύου	Nginx Reverse Proxy	Αντίστροφη διαμεσολάβηση, δρομολόγηση	Εικ. 4.2, 5.2
Εφαρμογής	UserService	Εγγραφή, σύνδεση, JWT	Εικ. 4.2, 4.5, 4.6
Εφαρμογής	ApplicationService	CRUD εφαρμογών, QoS κανόνες	Εικ. 4.2, 4.7
Εφαρμογής	TerraformService	Εκτέλεση Terraform/Helm/kubectl	Εικ. 4.2, 4.7, 4.10, 5.1, 5.2
Εφαρμογής	MonitoringService	Περιοδική αξιολόγηση QoS (30 s)	Εικ. 4.2, 4.8, 4.9, 5.2
Εφαρμογής	PrometheusService	Ερωτήματα PromQL προς το Prometheus	Εικ. 4.2, 4.8, 4.9, 5.2
Εφαρμογής	ScalingService	Εκτέλεση ενεργειών κλιμάκωσης	Εικ. 4.2, 4.8, 5.2
Εφαρμογής	NotificationService	Ειδοποιήσεις εντός UI και email	Εικ. 4.2, 4.8, 5.2
Πρόσβασης δεδομένων	*Repository (Spring Data JPA)	Αντιστοίχιση οντοτήτων και ερωτήματα	Εικ. 4.2, 4.5–4.10
Δεδομένων	PostgreSQL 15	Μόνιμη αποθήκευση (Παράρτημα Ε)	Εικ. 4.2, 5.1, 5.2
Εξωτερικό	Prometheus / Grafana / Alertmanager	Στοιβά παρακολούθησης	Εικ. 4.2, 5.2, 5.11–5.14
Εξωτερικό	Kubernetes API Server	Σημείο ελέγχου συστάδας	Εικ. 4.2, 5.2

Σημείωση. Ο πίνακας καλύπτει τα συστατικά που εμφανίζονται ως διακριτά κουτιά ή γραμμές ζωής στα διαγράμματα· τα επιμέρους *repositories* παρατίθενται συλλογικά. Οι ονομασίες τηρούν τη σύμβαση *PascalCase* των κλάσεων υλοποίησης (§4.4.2), ώστε κάθε στοιχείο του διαγράμματος αρχιτεκτονικής να αντιστοιχίζεται μονοσήμαντα σε πακέτο του πηγαίου κώδικα.

Ενοποιημένο script προετοιμασίας περιβάλλοντος

Τα βήματα επαλήθευσης 1–4 που ακολουθούν μπορούν να εκτελεστούν με ένα ενιαίο script:

```
#!/usr/bin/env bash
# setup_cluster.sh — προετοιμασία περιβάλλοντος επίδειξης σε ένα βήμα.
```

```
# Αντιστοιχεί στις ενέργειες που εκτελεί αυτόματα το TerraformService (§4.4.3).
set -euo pipefail # Τερματισμός στο πρώτο σφάλμα· απαγόρευση
undefined μεταβλητών.

GRAFANA_PASSWORD="${GRAFANA_PASSWORD:?Ορίστε τη μεταβλητή GRAFANA_PASSWORD}"

# 1. Δημιουργία τοπικής συστάδας Kubernetes.
minikube start --driver=docker --cpus=4 --memory=6144 --disk-size=20g

# 2. Ενεργοποίηση του metrics-server (προαπαιτούμενο του HPA για μετρικές CPU).
minikube addons enable metrics-server

# 3. Εγκατάσταση της στοίβας παρακολούθησης μέσω Helm.
helm repo add prometheus-community https://prometheus-community.github.io/helm-
charts
helm repo update
helm upgrade --install prometheus prometheus-community/kube-prometheus-stack \
--namespace monitoring --create-namespace \
--set grafana.adminPassword="${GRAFANA_PASSWORD}" \
--set prometheus.prometheusSpec.serviceMonitorSelectorNilUsesHelmValues=false \
--wait --timeout 10m

# 4. Αναμονή έως ότου το Grafana γίνει διαθέσιμο.
kubectl wait --for=condition=ready pod \
-l app.kubernetes.io/name=grafana -n monitoring --timeout=180s

echo "Η συστάδα και η στοίβα παρακολούθησης είναι έτοιμες."
```

Σημείωση ασφάλειας. Στο αρχικό script το συνθηματικό του Grafana ήταν ενσωματωμένο στον κώδικα (admin123). Στην αναθεωρημένη έκδοση αντλείται από μεταβλητή περιβάλλοντος, σύμφωνα με την αρχή που εφαρμόζεται και στο application.properties.

Επαλήθευση 1 — Εκκίνηση συστάδας Kubernetes

```
minikube start --driver=docker --cpus=4 --memory=6144 --disk-size=20g
kubectl get nodes # NAME: minikube | STATUS: Ready | VERSION: v1.35.1
kubectl cluster-info # Control plane: https://127.0.0.1:49379
```

Το αποτέλεσμα τεκμηριώνεται στην **Εικόνα 5.3**.

Επαλήθευση 2 — Εγκατάσταση Prometheus/Grafana μέσω Helm

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-
charts
helm repo update
helm upgrade --install prometheus prometheus-community/kube-prometheus-stack \
--namespace monitoring --create-namespace \
--set grafana.adminPassword="${GRAFANA_PASSWORD}" \
--set prometheus.prometheusSpec.serviceMonitorSelectorNilUsesHelmValues=false
kubectl get pods -n monitoring # Αναμονή ~3 λεπτά· όλα τα pods σε κατάσταση
Running
```

Το αποτέλεσμα τεκμηριώνεται στην **Εικόνα 5.10**.

Επαλήθευση 3 — Διάταξη της εφαρμογής επίδειξης

```
kubectl apply -f deployment.yaml # Πλήρες YAML: Παράρτημα A.1
kubectl get pods # 3 pods σε κατάσταση Running
minikube service demo-webapp-service --url
```

Τα αποτελέσματα τεκμηριώνονται στις **Εικόνες 5.5–5.7**.

Επαλήθευση 4 — Πρόσβαση στις διεπαφές παρακολούθησης

```
minikube dashboard &
```

```
kubectl port-forward -n monitoring svc/prometheus-grafana 3000:80 &
kubectl port-forward -n monitoring svc/prometheus-kube-prometheus-prometheus
9090:9090 &
kubectl port-forward -n monitoring svc/prometheus-kube-prometheus-alertmanager
9093:9093 &
```

Τα σημεία πρόσβασης συνοψίζονται στον **Πίνακα 5.3**.

Πίνακας 5.3 — Σημεία Πρόσβασης Υπηρεσιών του Περιβάλλοντος Επίδειξης

Υπηρεσία	URL	Διαπιστευτήρια	Τεκμηριώνεται σε
Web UI πλατφόρμας	http://localhost	Λογαριασμός χρήστη (JWT)	Εικ. 5.21, 5.22
Grafana	http://localhost:3000	admin / \${GRAFANA_PASSWORD}	Εικ. 5.11–5.13
Prometheus	http://localhost:9090	—	Εικ. 5.17, 5.18
Alertmanager	http://localhost:9093	—	Εικ. 5.19, 5.20
Kubernetes Dashboard	minikube dashboard (αυτόματο URL)	Token ServiceAccount	Εικ. 5.4, 5.8
Εφαρμογή επίδειξης	http://<minikube-ip>:32187	—	Εικ. 5.9

Σημείωση. Το συνθηματικό του Grafana δεν παρατίθεται σε καθαρό κείμενο· ορίζεται κατά την εγκατάσταση μέσω μεταβλητής περιβάλλοντος (βλ. `setup_cluster.sh`).

Επαλήθευση 5 — Λειτουργία της διατεταγμένης εφαρμογής

```
# Η θύρα 32187 αντιστοιχεί στο nodePort του Service (Παράρτημα Α.1, Εικόνα 5.7).
MINIKUBE_IP=$(minikube ip)
curl -I "http://${MINIKUBE_IP}:32187/"
# HTTP/1.1 200 OK — η προεπιλεγμένη σελίδα του nginx:alpine αποκρίνεται κανονικά.

# Επαλήθευση ότι το Service δρομολογεί σε όλα τα υγιή pods (round-robin):
kubectl get endpoints demo-webapp-service
```

Τα στιγμιότυπα που ακολουθούν καταγράφηκαν κατά την πραγματική εκτέλεση του συστήματος σε συστάδα Minikube την 1η Ιουνίου 2026. Κάθε στιγμιότυπο συνοδεύεται από αναλυτική ερμηνεία και από ρητή αντιστοίχιση με τις απαιτήσεις του Κεφαλαίου 4.

5.2.2 Σενάριο 1: Διάταξη και Επαλήθευση της Συστάδας

Στόχος. Δημιουργία τοπικής συστάδας Kubernetes μέσω Minikube, διάταξη της εφαρμογής επίδειξης και επαλήθευση της πλήρους λειτουργίας της (nodes, pods, deployments, services, dashboard). Το σενάριο τεκμηριώνει τη δημιουργία της συστάδας, δηλαδή το υπόβαθρο επί του οποίου στηρίζονται οι FA-A02 και FA-A04 (Πίνακας 4.3).

5.2.2.1 Κατάσταση συστάδας και κόμβου

```
Last login: Mon Jun 1 12:26:42 on ttys000
andreaskosmop@MacBook-Pro-andreas ~ % kubectl get nodes -o wide
NAME        STATUS    ROLES    AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION   CONTAINER-RUNTIME
minikube    Ready     control-plane  7m6s  v1.35.1   192.168.49.2   <none>         Debian GNU/Linux 12 (bookworm)  6.12.76-linuxkit  docker://29.2.1
andreaskosmop@MacBook-Pro-andreas ~ % kubectl cluster-info
Kubernetes control plane is running at https://127.0.0.1:49379
CoreDNS is running at https://127.0.0.1:49379/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

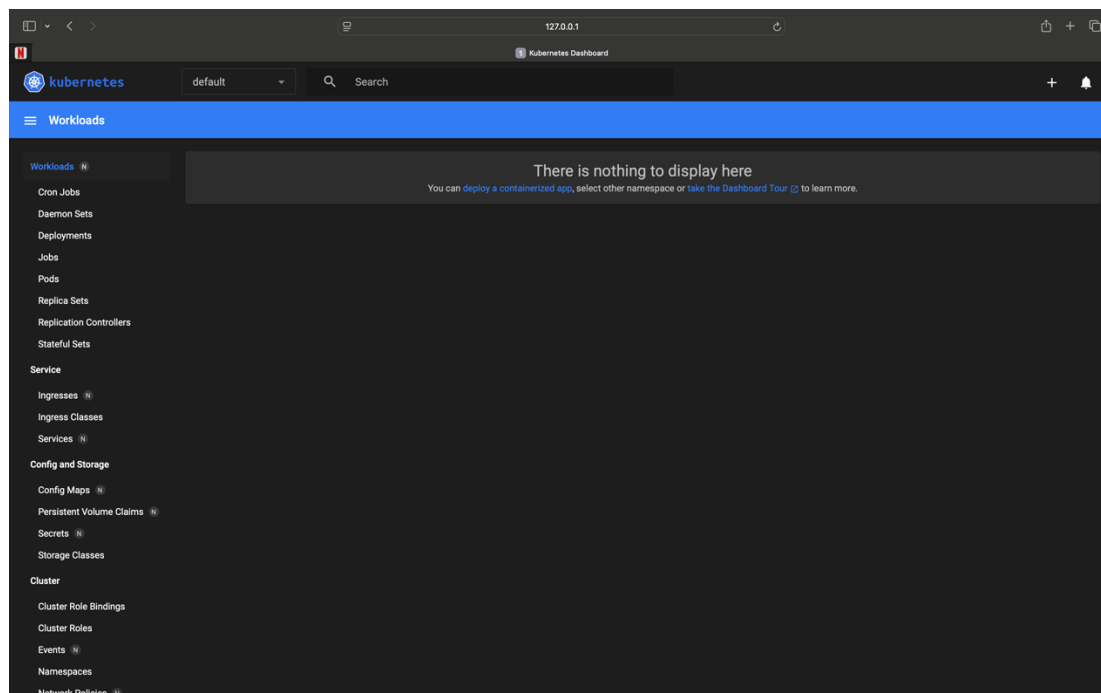
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

Εικόνα 5.3 — Έξοδος των `kubectl get nodes` και `kubectl cluster-info`

Σημείωση. Kubernetes v1.35.1· container runtime `docker://29.2.1`.

Ανάλυση. Ο κόμβος `minikube` εμφανίζεται σε κατάσταση `Ready`, με Kubernetes `v1.35.1` επί `Debian GNU/Linux 12 (Bookworm)`, container runtime `docker://29.2.1` και ηλικία 7 λεπτά και 6 δευτερόλεπτα. Το control plane είναι προσβάσιμο στο `https://127.0.0.1:49379` και η υπηρεσία CoreDNS λειτουργεί κανονικά. Η **Εικόνα 5.3** επαληθεύει την επιτυχή ολοκλήρωση του Βήματος 1 του οδηγού (§5.2.1, Επαλήθευση 1) και επιβεβαιώνει την αντιστοίχιση εκδόσεων του **Πίνακα 5.1**.

5.2.2.2 Kubernetes Dashboard



Εικόνα 5.4 — *Kubernetes Dashboard: Γραφική Διεπαφή Διαχείρισης Συστάδας*

Ανάλυση. Το Kubernetes Dashboard, ενεργοποιημένο μέσω `minikube addons enable dashboard`, παρέχει ιεραρχική πλοήγηση στα αντικείμενα της συστάδας: Workloads (Deployments, Pods, ReplicaSets, Jobs), Service (Ingresses, Services), Config and Storage και Cluster (Namespaces, Nodes, Events). Ο επιλογέας namespace είναι ρυθμισμένος σε `default`. Το μήνυμα «There is nothing to display here» πριν από τη διάταξη επιβεβαιώνει ότι η συστάδα είναι καθαρή, καθιστώντας έτσι αδιαμφισβήτητη τη μεταβολή που καταγράφεται στην **Εικόνα 5.8**. Επαληθεύει την απαίτηση FA-A02 (ορατότητα κατάστασης συστάδας).

5.2.2.3 Εκτελούμενα Pods

```
andreaskosmop@MacBook-Pro-andreas demo-app % kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
demo-webapp-5946b874d-4dfxn         1/1     Running   0           44s
demo-webapp-5946b874d-bhzl4         1/1     Running   0           44s
demo-webapp-5946b874d-zdww8         1/1     Running   0           44s
```

Εικόνα 5.5 — *Έξοδος του `kubectl get pods`: τρία Pods `demo-webapp` σε κατάσταση `Running`*

Ανάλυση. Μετά την εφαρμογή του `deployment.yaml` (Παράρτημα A.1), τρία pods βρίσκονται σε κατάσταση `Running` με `READY: 1/1`, `RESTARTS: 0` και ηλικία 44 δευτερολέπτων. Τα ονόματα `demo-webapp-5946b874d-{4dfxn, bhzl4, zdww8}` ακολουθούν τη σύμβαση `<deployment>-<pod-template-hash>-<random>`.

suffix> η συμβολοσειρά 5946b874d αποτελεί το pod-template-hash του ReplicaSet και επιτρέπει τη μονοσήμαντη συσχέτιση των pods με τη συγκεκριμένη αναθεώρηση (revision) του Deployment. Τεκμηριώνει τη διάταξη του δηλωμένου αριθμού αντιγράφων. Επισημαίνεται ότι η ενέργεια εκτελέστηκε με εντολές kubectl και όχι μέσω της πλατφόρμας· η FA-A04 επαληθεύεται επομένως ως προς το αποτέλεσμα, όχι ως προς τη διαδρομή εκτέλεσης.

5.2.2.4 Κατάσταση Deployment

```
andreaskosmop@MacBook-Pro-andreas demo-app % kubectl get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
demo-webapp   3/3     3            3           2m19s
```

Εικόνα 5.6 — Έξοδος του `kubectl get deployments`: `READY 3/3`, `UP-TO-DATE 3`, `AVAILABLE 3`

Ανάλυση. Το Deployment `demo-webapp` εμφανίζει `READY 3/3` (τρία επιθυμητά, τρία υγιή αντίγραφα), `UP-TO-DATE 3` (όλα τα pods εκτελούν την τρέχουσα αναθεώρηση), `AVAILABLE 3` (όλα έχουν περάσει επιτυχώς το readiness probe που ορίζεται στο Παράρτημα Α.1) και ηλικία 2 λεπτών και 19 δευτερολέπτων. Η ταύτιση των τριών στηλών αποδεικνύει ότι ο Deployment controller συνέκλινε στην επιθυμητή κατάσταση χωρίς σφάλματα. Επαληθεύει την απαίτηση FA-A02.

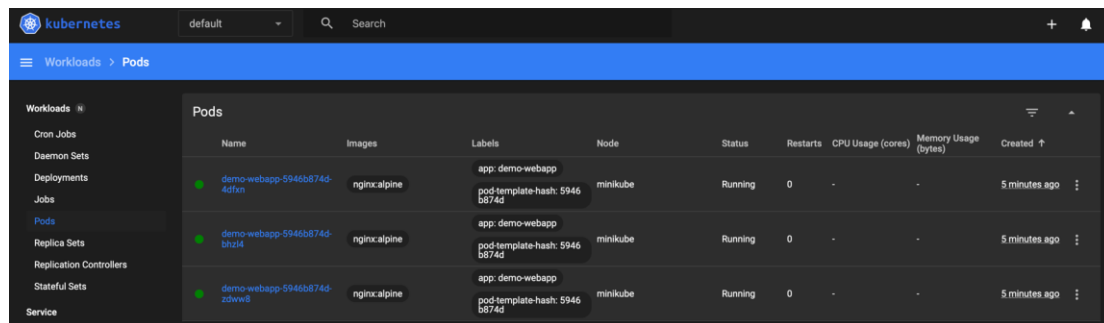
5.2.2.5 Έκθεση μέσω Service τύπου NodePort

```
andreaskosmop@MacBook-Pro-andreas demo-app % kubectl get services
NAME                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
demo-webapp-service NodePort    10.99.153.244 <none>        80:32187/TCP     3m26s
kubernetes           ClusterIP   10.96.0.1    <none>        443/TCP          14m
```

Εικόνα 5.7 — Έξοδος του `kubectl get services`: `NodePort 80:32187/TCP`

Ανάλυση. Καταγράφονται δύο Services: (α) το `demo-webapp-service` τύπου `NodePort`, με `CLUSTER-IP 10.99.153.244` και αντιστοίχιση θυρών `80:32187/TCP`, προσβάσιμο εξωτερικά στο `<minikube-ip>:32187`, και (β) το `kubernetes` τύπου `ClusterIP` (`10.96.0.1:443`), που αποτελεί το εσωτερικό endpoint του API server. Η επιλογή `NodePort` αντί `LoadBalancer` είναι συνειδητή για το τοπικό περιβάλλον: επιτρέπει εξωτερική πρόσβαση χωρίς εξάρτηση από cloud load balancer, ο οποίος σε παραγωγικό περιβάλλον EKS/AKS παρέχεται από τον αντίστοιχο controller (Παραρτήματα Γ και Δ). Τεκμηριώνει την έκθεση της υπηρεσίας. Η έκθεση αποτελεί τμήμα της διάταξης της εφαρμογής και όχι της δημιουργίας συστάδας· εκτελέστηκε χειροκίνητα, οπότε η FA-A04 δεν επαληθεύεται πλήρως από το παρόν τεκμήριο.

5.2.2.6 Οπτική επιβεβαίωση μέσω Dashboard



The screenshot shows the Kubernetes Dashboard interface. On the left, a sidebar lists various workload types: Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, Stateful Sets, and Service. The 'Pods' section is selected, displaying a table of three pods. Each pod is in a 'Running' state, has 0 restarts, and was created 5 minutes ago. The pods are labeled with 'app: demo-webapp' and 'pod-template-hash: 5946b874d'. The images used are 'nginx:alpine'.

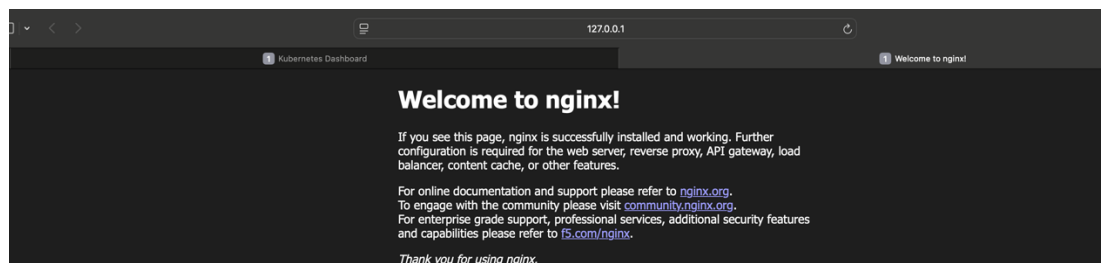
Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created
demo-webapp-5946b874d-4dfrn	nginx:alpine	app: demo-webapp pod-template-hash: 5946b874d	minikube	Running	0	-	-	5 minutes ago
demo-webapp-5946b874d-bhul4	nginx:alpine	app: demo-webapp pod-template-hash: 5946b874d	minikube	Running	0	-	-	5 minutes ago
demo-webapp-5946b874d-zdww8	nginx:alpine	app: demo-webapp pod-template-hash: 5946b874d	minikube	Running	0	-	-	5 minutes ago

Εικόνα 5.8 — Kubernetes Dashboard: τα τρία Pods σε κατάσταση `Running`

Ανάλυση. Το Dashboard αποδίδει γραφικά την ίδια πληροφορία με την Εικόνα 5.5: πράσινος δείκτης κατάστασης (`Running`), image `nginx:alpine`, labels (`app: demo-`

webapp, pod-template-hash: 5946b874d), κόμβος minikube, μηδέν επανεκκινήσεις, χρόνος δημιουργίας πέντε λεπτά. Η αντιπαράβολή των **Εικόνων 5.4** και **5.8** τεκμηριώνει τη μετάβαση από κενή σε λειτουργική συστάδα. Επαληθεύει την απαίτηση FA-A02. Επισημαίνεται ότι το Kubernetes Dashboard είναι εργαλείο τρίτου και όχι συστατικό της πλατφόρμας· συνεπώς δεν στοιχειοθετεί την MFA-U02, η οποία αφορά την ενιαία εμφάνιση της δικής μας διεπαφής.

5.2.2.7 Πρόσβαση στην εφαρμογή από τον browser



Εικόνα 5.9 — Η Εφαρμογή Επίδειξης Προσβάσιμη μέσω NodePort

Ανάλυση. Η διατεταγμένη εφαρμογή είναι η `demo-webapp`: ένα μεμονωμένο container `nginx:alpine`, χωρίς κατάσταση (stateless) και χωρίς backend ή βάση δεδομένων, εκτεθειμένο μέσω Service τύπου NodePort. Η εμφάνιση του μηνύματος «Welcome to nginx!» επιβεβαιώνει ολόκληρη την αλυσίδα εξυπηρέτησης: το container εκτελείται → ο kubelet το αναφέρει υγιές → το Service δρομολογεί την κίνηση → το pod αποκρίνεται. Σε πραγματική χρήση, ο χρήστης θα διέτασσε τη δική του, πολυπλοκότερη εφαρμογή (ενδεχομένως με backend και βάση δεδομένων) αντί για το παρόν ελάχιστο παράδειγμα· η επιλογή του `nginx:alpine` έγινε ώστε η επίδειξη να εστιάζει στη λειτουργικότητα της **πλατφόρμας** και όχι της διαχειριζόμενης εφαρμογής. Αποτελεί την κεντρική επαλήθευση της απαίτησης FA-A04.

5.2.3 Σενάριο 2: Παρακολούθηση σε Πραγματικό Χρόνο

Στόχος. Επαλήθευση της λειτουργίας της στοίβας παρακολούθησης (Prometheus, Grafana, Alertmanager) και της συλλογής μετρικών της συστάδας σε πραγματικό χρόνο. Το σενάριο επαληθεύει την απαίτηση FA-A08, η οποία αποτελεί προϋπόθεση —όχι απόδειξη— της FA-A06 (Πίνακας 4.3).

5.2.3.1 Στοίβα παρακολούθησης σε λειτουργία

```
andreaskosmop@MacBook-Pro-andreas ~ % kubectl get pods -n monitoring
```

NAME	READY	STATUS	RESTARTS	AGE
alertmanager-prometheus-kube-prometheus-alertmanager-0	2/2	Running	0	92s
prometheus-grafana-568b547658-mjj2c	3/3	Running	0	114s
prometheus-kube-prometheus-operator-5786fb9bd4-b457f	1/1	Running	0	114s
prometheus-kube-state-metrics-678b9d8db-bf7m7	1/1	Running	0	114s
prometheus-prometheus-kube-prometheus-prometheus-0	2/2	Running	0	92s
prometheus-prometheus-node-exporter-w9kk4	1/1	Running	0	114s

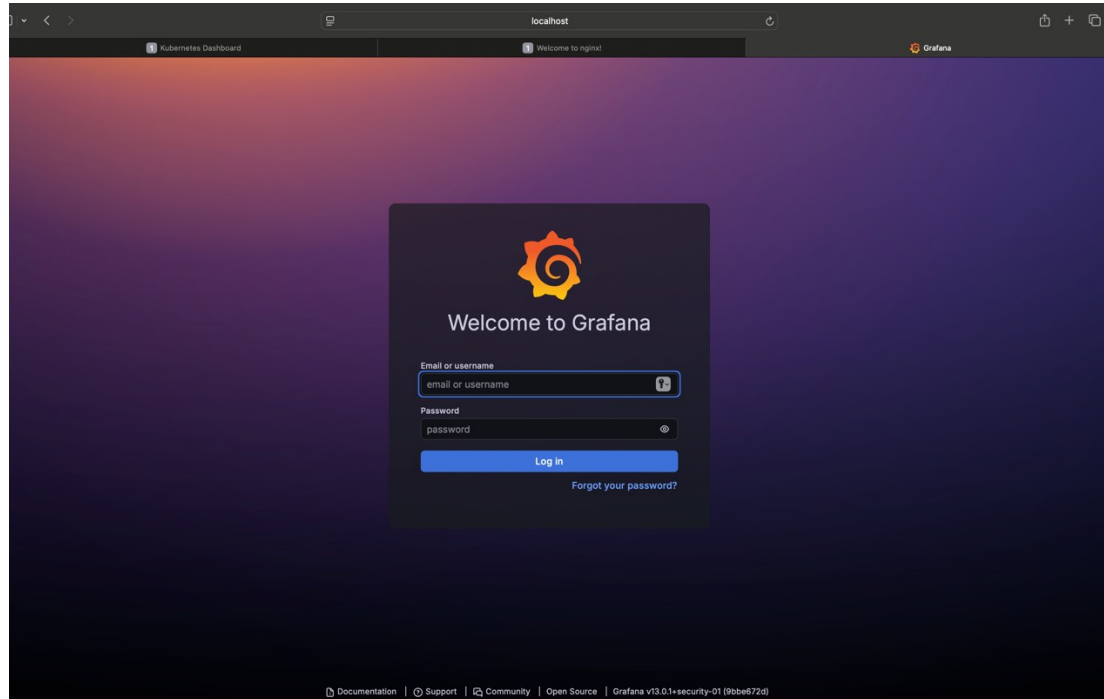
```
andreaskosmop@MacBook-Pro-andreas ~ %
```

Εικόνα 5.10 — Έξοδος του `kubectl get pods -n monitoring`: έξι Pods της στοίβας `kube-prometheus-stack`

Ανάλυση. Η στοίβα παρακολούθησης, εγκατεστημένη μέσω `helm install kube-prometheus-stack` (Helm Authors, 2024), περιλαμβάνει: `alertmanager` (2/2 containers, 92 s), `grafana` (3/3, 114 s), `kube-prometheus-operator` (1/1), `kube-state-metrics` (1/1), `prometheus` (2/2, 92 s) και `node-exporter` (1/1). Ο αριθμός containers ανά pod αντικατοπτρίζει την αρχιτεκτονική κάθε συστατικού: τα

prometheus και alertmanager διατάσσονται ως StatefulSets με sidecar config-reloader, ενώ το grafana περιλαμβάνει επιπλέον sidecars για τη δυναμική φόρτωση dashboards και data sources. Επαληθεύει την απαίτηση FA-A08 ως προς την ύπαρξη λειτουργικής υποδομής παρακολούθησης.

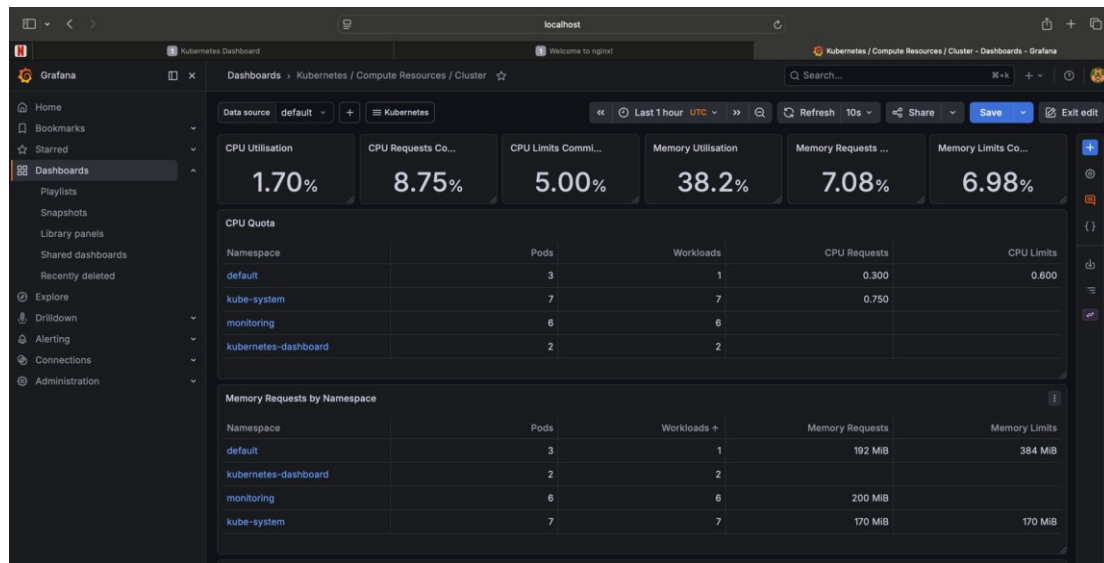
5.2.3.2 Πρόσβαση στο Grafana



Εικόνα 5.11 — *Grafana v13.0.1: Σελίδα Σύνδεσης*

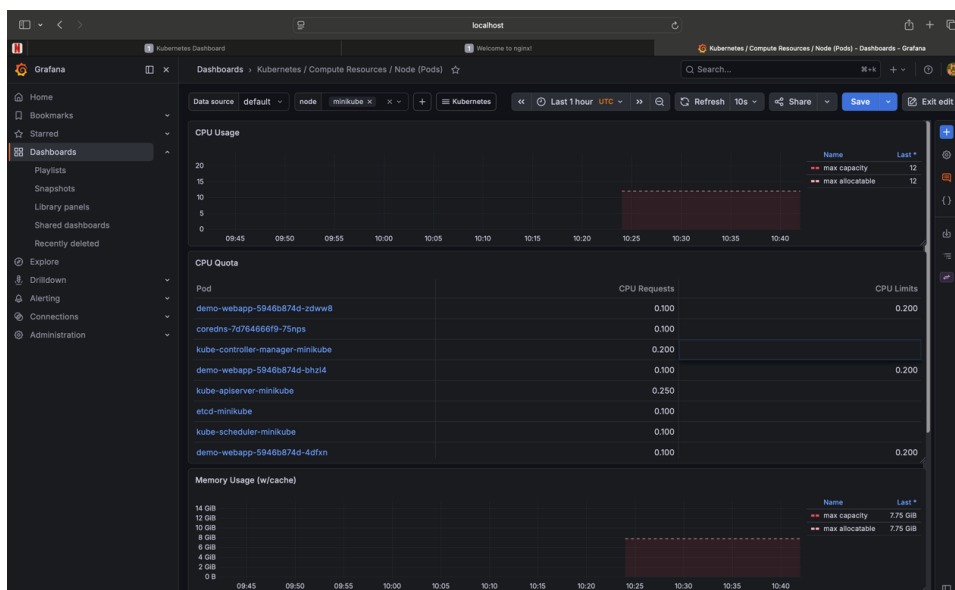
Ανάλυση. Το Grafana v13.0.1 (build 9bbe672d, με ενσωματωμένα security patches) είναι προσβάσιμο στο `http://localhost:3000` μέσω `kubectl port-forward` (Πίνακας 5.3). Τα διαπιστευτήρια ορίστηκαν κατά την εγκατάσταση μέσω της παραμέτρου `--set grafana.adminPassword`. Στη γραμμή καρτελών του browser διακρίνονται τρεις παράλληλες συνεδρίες (Kubernetes Dashboard, εφαρμογή επίδειξης, Grafana), γεγονός που τεκμηριώνει ότι τα στιγμιότυπα των Εικόνων 5.4, 5.9 και 5.11 προέρχονται από την ίδια συνεδρία εκτέλεσης. Αντιστοιχεί στη μη λειτουργική απαίτηση MFA-S01 (ταυτοποίηση).

5.2.3.3 Επισκόπηση συστάδας στο Grafana



Εικόνα 5.12 — Grafana — Kubernetes / Compute Resources / Cluster: CPU 1,70%, Μνήμη 38,2% **Ανάλυση.** Το προεγκατεστημένο dashboard παρέχει real-time επισκόπηση της συστάδας, με αυτόματη ανανέωση ανά 10 δευτερόλεπτα: αξιοποίηση CPU 1,70%, δεσμευμένες αιτήσεις CPU 8,75%, αξιοποίηση μνήμης 38,2%. Ο πίνακας CPU Quota αναλύει την κατανομή ανά namespace — default (3 pods, 0,300 requests / 0,600 limits), kube-system (7 pods, 0,750), monitoring (6 pods), kubernetes-dashboard (2 pods) — ενώ οι αιτήσεις μνήμης κατανέμονται σε 192 MiB (default), 200 MiB (monitoring) και 170 MiB (kube-system). Πρόκειται για ζωντανά δεδομένα, όχι για στατική αναπαράσταση, γεγονός που επαληθεύει την απαίτηση FA-A08 (πίνακας ελέγχου πραγματικού χρόνου). Η χαμηλή αξιοποίηση CPU αποτελεί την **αναφορά ηρεμίας (baseline)** έναντι της οποίας αξιολογείται η μεταβολή στην **Εικόνα 5.13**.

5.2.3.4 Μετρικές επιπέδου κόμβου



Εικόνα 5.13 — Grafana — Node (Pods): Αίχμη CPU από τις 10:25

Ανάλυση. Τα γραφήματα επιπέδου κόμβου καλύπτουν το διάστημα 09:45–10:40. Στο διάγραμμα χρήσης CPU καταγράφεται ευκρινής αιχμή από τις 10:25, χρονικά

συμπίπτουσα με την εκκίνηση των γεννητριών φόρτου του Σεναρίου 3. Ο πίνακας CPU Quota αποδίδει τις δεσμεύσεις ανά pod: demo-webapp (0,100 requests / 0,200 limits), kube-apiserver (0,250), kube-controller-manager (0,200). Η χρήση μνήμης παραμένει σταθερή στα 7,75 GiB, γεγονός αναμενόμενο δεδομένου ότι ο φόρτος που παρήχθη ήταν αμιγώς υπολογιστικός. Η αιχμή που καταγράφεται εδώ προέρχεται από τη δοκιμή φόρτου του Σεναρίου 3 και αναλύεται εκεί· στο παρόν σενάριο η Εικόνα 5.13 τεκμηριώνει αποκλειστικά την ικανότητα του συστήματος να αποτυπώνει μεταβολές πόρων σε πραγματικό χρόνο. Η ίδια εικόνα αποτελεί την οπτική απόδειξη της μεταβολής φόρτου που πυροδότησε τον μηχανισμό κλιμάκωσης (Εικόνα 5.16) και επαληθεύει την απαίτηση FA-A08.

5.2.4 Σενάριο 3: Αυτόματη Κλιμάκωση βάσει QoS Κανόνα

Στόχος. Ορισμός QoS κανόνα με κατώφλι 50% αξιοποίησης CPU, παραγωγή ελεγχόμενου τεχνητού φόρτου και παρακολούθηση της αυτόματης κλιμάκωσης των pods. Το σενάριο επαληθεύει τις απαιτήσεις FA-A06, FA-A08 και FA-A09 (Πίνακας 4.3).

5.2.4.1 Δημιουργία του HorizontalPodAutoscaler

```
andreaskosmop@MacBook-Pro-andreas ~ % kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
demo-webapp	Deployment/demo-webapp	cpu: 0%/50%	2	10	3	87s

```
andreaskosmop@MacBook-Pro-andreas ~ %
```

Εικόνα 5.14 — Έξοδος του `kubectl get hpa`: CPU 0%/50%, MINPODS 2, MAXPODS 10, REPLICAS 3

Ανάλυση. Ο HorizontalPodAutoscaler για το demo-webapp εμφανίζει TARGETS cpu: 0%/50% (τρέχουσα τιμή έναντι κατωφλίου), MINPODS 2, MAXPODS 10, REPLICAS 3, ηλικία 87 δευτερολέπτων. Η ένδειξη 0% επιβεβαιώνει κατάσταση ηρεμίας πριν από την εφαρμογή φόρτου. Η τιμή αυτή δεν αντιφάσκει με το 1,70% του Πίνακα 4.11: ο HPA εκφράζει τη μέση χρησιμοποίηση CPU των pods ως ποσοστό των αιτούμενων πόρων (requests), στρογγυλοποιημένο σε ακέραιο, ενώ ο Πίνακας 4.11 καταγράφει τη χρησιμοποίηση CPU σε επίπεδο κόμβου όπως την αποδίδει το Grafana. Το κατώφλι 50% εκφράζει τον QoS κανόνα `cpu_usage > 50% → SCALE_PODS_UP` σε μορφή διαμόρφωσης HorizontalPodAutoscaler. Στην παρούσα επίδειξη ο HPA διαμορφώθηκε χειροκίνητα· σε πλήρη λειτουργία της πλατφόρμας η ίδια διαμόρφωση παράγεται από το ScalingService κατά την αξιολόγηση του κανόνα, ο οποίος αποθηκεύεται ως εγγραφή στον πίνακα `qos_rules` (Παράρτημα Ε). Το κάτω όριο `min = 2` εξασφαλίζει υψηλή διαθεσιμότητα ακόμη και σε μηδενικό φόρτο, ενώ το άνω όριο `max = 10` λειτουργεί ως ανώτατο όριο κόστους.

5.2.4.2 Κλιμάκωση σε πραγματικό χρόνο

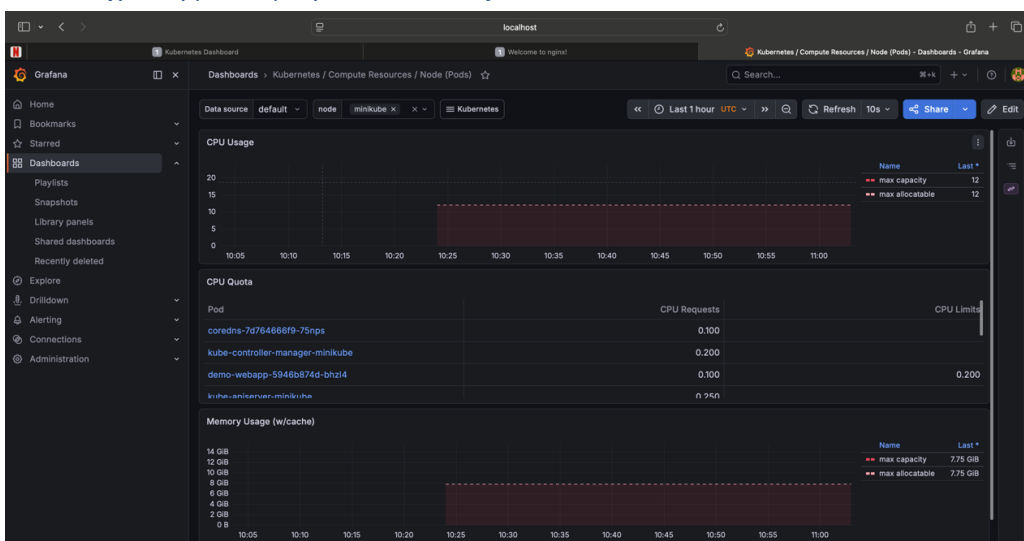
```
andreas@MacBook-Pro-andreas ~ % kubectl get hpa -w
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
demo-webapp	Deployment/demo-webapp	cpu: 10%/50%	2	10	3	3m48s
demo-webapp	Deployment/demo-webapp	cpu: 30%/50%	2	10	3	4m30s
demo-webapp	Deployment/demo-webapp	cpu: 30%/50%	2	10	3	5m15s
demo-webapp	Deployment/demo-webapp	cpu: 33%/50%	2	10	2	5m30s
demo-webapp	Deployment/demo-webapp	cpu: 47%/50%	2	10	2	6m31s
demo-webapp	Deployment/demo-webapp	cpu: 44%/50%	2	10	2	7m31s
demo-webapp	Deployment/demo-webapp	cpu: 45%/50%	2	10	2	8m31s
demo-webapp	Deployment/demo-webapp	cpu: 50%/50%	2	10	2	9m31s
demo-webapp	Deployment/demo-webapp	cpu: 47%/50%	2	10	2	10m
demo-webapp	Deployment/demo-webapp	cpu: 56%/50%	2	10	2	11m
demo-webapp	Deployment/demo-webapp	cpu: 56%/50%	2	10	3	11m

Εικόνα 5.15 — Έξοδος του `kubectl get hpa -w`: μετάβαση CPU 0% → 56% και κλιμάκωση σε 11 λεπτά

Ανάλυση. Ο φόρτος παρήχθη με τρεις παράλληλες γεννήτριες `busy-loop` εντός της συστάδας (`kubectl run load-generator --image=busybox -- /bin/sh -c "while true; do wget -q -O- http://demo-webapp; done"`), οι οποίες ανέβασαν την αξιοποίηση CPU πάνω από το κατώφλι του 50%. Το κατώφλι αυτό αντιστοιχεί στον κανόνα `cpu_usage > 50% → SCALE_PODS_UP`, ο οποίος στην παρούσα επίδειξη εκφράζεται ως διαμόρφωση `HorizontalPodAutoscaler` και όχι ως εγγραφή `qos_rules` αξιολογούμενη από το `MonitoringService`. Καταγράφεται πλήρης κύκλος αυτόματης κλιμάκωσης υπό φόρτο: στα 3'48" η CPU φθάνει το 10%· στο διάστημα 4'30"–5'15" σταθεροποιείται στο 30%· στα 5'30" φθάνει το 33% με `REPLICAS 2` (μείωση αντιγράφων κατά τη φάση σταθεροποίησης)· στα 6'31" φθάνει το 47%· στα 9'31" το 50%· και στα 11'00" το 56%, οπότε ο controller αυξάνει τα αντίγραφα σε `REPLICAS 3`. Ο κύκλος 3 → 2 → 3 αποδεικνύει **αμφίδρομη** κλιμάκωση, δηλαδή ότι ο μηχανισμός δεν αυξάνει μόνο αλλά και αποδεδμεύει πόρους όταν ο φόρτος υποχωρεί — χαρακτηριστικό κρίσιμο για τη βελτιστοποίηση κόστους. Η χρονική υστέρηση μεταξύ της υπέρβασης του κατωφλίου και της ενέργειας κλιμάκωσης οφείλεται στο παράθυρο σταθεροποίησης (`stabilizationWindowSeconds`) που ρυθμίζεται ρητά στο Παράρτημα A.2 και αποτρέπει ταλαντώσεις από στιγμιαίες αιχμές. Αποτελεί την κεντρική επαλήθευση της απαίτησης FA-A06.

5.2.4.3 Συσχέτιση με τις μετρικές του Grafana



Εικόνα 5.16 — Grafana: Χρονοσειρά Αξιοποίησης CPU κατά τη Δοκιμή Φόρτου (10:25–11:05)

Ανάλυση. Το γράφημα χρήσης CPU εμφανίζει σαφή αιχμή από τις 10:25, ενώ η μνήμη παραμένει σταθερή στα 7,75 GiB. Ο πίνακας CPU Quota απομονώνει το pod `demo-`

webapp-5946b874d-bhz14 (0,100 requests / 0,200 limits). Η χρονική σύμπτωση της αιχμής με την εκκίνηση των γεννητριών φόρτου και με την ενέργεια κλιμάκωσης της **Εικόνας 5.15** τεκμηριώνει τη **συσχέτιση** μεταξύ παρατηρούμενης μετρικής και αυτόματης αντίδρασης του συστήματος — δηλαδή ακριβώς τον βρόχο ανάδρασης που περιγράφει το Sequence Diagram «Αυτόματη Κλιμάκωση QoS» (**Εικόνα 4.8**). Επαληθεύει την απαίτηση FA-A08.

5.2.4.4 Κάλυψη συλλογής μετρικών

Query Result	Instance	Service	Value
up{container="config-reloader", endpoint="reloader-web", instance="10.244.0.14:8080", job="prometheus-kube-prometheus-prometheus", namespace="monitoring", pod="prometheus-kube-prometheus-prometheus-0", service="prometheus-kube-prometheus-prometheus"}	10.244.0.14:8080	prometheus-kube-prometheus-prometheus	1
up{endpoint="https", instance="192.168.49.2:8443", job="apiserver", namespace="default", service="kubernetes"}	192.168.49.2:8443	kubernetes	1
up{container="kube-controller-manager", endpoint="http-metrics", instance="192.168.49.2:10257", job="kube-controller-manager", namespace="kube-system", pod="kube-controller-manager-minikube", service="prometheus-kube-prometheus-kube-controller-manager"}	192.168.49.2:10257	prometheus-kube-prometheus-kube-controller-manager	0
up{container="kube-prometheus-stack", endpoint="https", instance="10.244.0.10:2550", job="prometheus-kube-prometheus-operator", namespace="monitoring", pod="prometheus-kube-prometheus-operator-5786fb9b4-b457f", service="prometheus-kube-prometheus-operator"}	10.244.0.10:2550	prometheus-kube-prometheus-operator	1
up{endpoint="https-metrics", instance="192.168.49.2:10250", job="kubelet", metrics_path="/metrics/cadvisor", namespace="kube-system", node="minikube", service="prometheus-kube-prometheus-kubelet"}	192.168.49.2:10250	prometheus-kube-prometheus-kubelet	1
up{container="prometheus", endpoint="http-web", instance="10.244.0.14:9090", job="prometheus-kube-prometheus-prometheus", namespace="monitoring", pod="prometheus-kube-prometheus-prometheus-0", service="prometheus-kube-prometheus-prometheus"}	10.244.0.14:9090	prometheus-kube-prometheus-prometheus	1
up{endpoint="https-metrics", instance="192.168.49.2:10249", job="kube-proxy", namespace="kube-system", pod="kube-proxy-rxwcd", service="prometheus-kube-prometheus-kube-proxy"}	192.168.49.2:10249	prometheus-kube-prometheus-kube-proxy	1
up{container="kube-scheduler", endpoint="http-metrics", instance="192.168.49.2:10259", job="kube-scheduler", namespace="kube-system", pod="kube-scheduler-minikube", service="prometheus-kube-prometheus-kube-scheduler"}	192.168.49.2:10259	prometheus-kube-prometheus-kube-scheduler	0
up{endpoint="https-metrics", instance="192.168.49.2:10250", job="kubelet", metrics_path="/metrics/probes", namespace="kube-system", node="minikube", service="prometheus-kube-prometheus-kubelet"}	192.168.49.2:10250	prometheus-kube-prometheus-kubelet	1
up{container="node-exporter", endpoint="http-metrics", instance="192.168.49.2:9100", job="node-exporter", namespace="monitoring", pod="prometheus-prometheus-node-exporter-w9k4", service="prometheus-prometheus-node-exporter"}	192.168.49.2:9100	prometheus-prometheus-node-exporter	1
up{endpoint="https-metrics", instance="192.168.49.2:10250", job="kubelet", metrics_path="/metrics", namespace="kube-system", node="minikube", service="prometheus-kube-prometheus-kubelet"}	192.168.49.2:10250	prometheus-kube-prometheus-kubelet	1
up{container="coredns", endpoint="http-metrics", instance="10.244.0.2:9153", job="coredns", namespace="kube-system", pod="coredns-7d764666f9-75nps", service="prometheus-kube-prometheus-coredns"}	10.244.0.2:9153	prometheus-kube-prometheus-coredns	1
up{container="alertmanager", endpoint="http-web", instance="10.244.0.13:9093", job="prometheus-kube-prometheus-alertmanager", namespace="monitoring", pod="alertmanager-prometheus-kube-prometheus-alertmanager-0", service="prometheus-kube-prometheus-alertmanager"}	10.244.0.13:9093	prometheus-kube-prometheus-alertmanager	1
up{container="config-reloader", endpoint="reloader-web", instance="10.244.0.13:8080", job="prometheus-kube-prometheus-alertmanager", namespace="monitoring", pod="alertmanager-prometheus-kube-prometheus-alertmanager-0", service="prometheus-kube-prometheus-alertmanager"}	10.244.0.13:8080	prometheus-kube-prometheus-alertmanager	1
up{container="grafana", endpoint="http-web", instance="10.244.0.9:3000", job="prometheus-grafana", namespace="monitoring", pod="prometheus-grafana-568b547658-nq2c", service="prometheus-grafana"}	10.244.0.9:3000	prometheus-grafana	1
up{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="monitoring", pod="prometheus-kube-state-metrics-678b9d8b-b7m7", service="prometheus-kube-state-metrics"}	10.244.0.11:8080	prometheus-kube-state-metrics	1
up{container="etcd", endpoint="http-metrics", instance="192.168.49.2:2381", job="kube-etcd", namespace="kube-system", pod="etcd-minikube", service="prometheus-kube-prometheus-etcd"}	192.168.49.2:2381	prometheus-kube-prometheus-etcd	0

Εικόνα 5.17 — Prometheus — ερώτημα `up`: 17 endpoints, χρόνος απόκρισης 20 ms

Ανάλυση. Το ερώτημα `up` (τιμή 1 = διαθέσιμο, 0 = μη διαθέσιμο) επιστρέφει 17 χρονοσειρές σε 20 ms. Τα endpoints καλύπτουν: prometheus, kubernetes-apiserver, kube-controller-manager, kube-prometheus-operator, kubelet (τρία instances: cadvisor, probes, metrics), kube-proxy, kube-scheduler, node-exporter, coredns, alertmanager (δύο), grafana, kube-state-metrics και etcd. Ο χρόνος απόκρισης των 20 ms είναι κατά δύο τάξεις μεγέθους χαμηλότερος από το όριο της μη λειτουργικής απαίτησης MFA-P01 (**Πίνακας 4.4**). Επισημαίνεται ότι τα 17 endpoints αφορούν τόσο την υποδομή της συστάδας (API server, controller manager, scheduler) όσο και τα ίδια τα εργαλεία παρακολούθησης (Prometheus, Grafana, Alertmanager, kube-state-metrics) — όχι αποκλειστικά τη διαχειριζόμενη εφαρμογή. Επαληθεύει την απαίτηση FA-A08 ως προς την κάλυψη της συλλογής μετρικών.

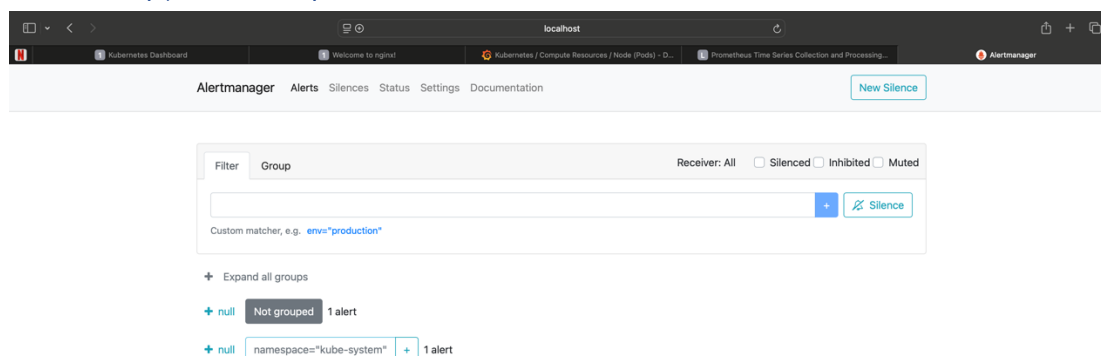
5.2.4.5 Ερώτημα PromQL σε επίπεδο εφαρμογής

Query	Execute																												
<code>kube_pod_status_phase{phase="Running"}</code>	Execute																												
Table Graph Explain																													
Load time: 21ms Result series: 18																													
<table><tr><td>kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="storage-provisioner", service="prometheus-kube-state-metrics", uid="b9d1a374-b008-48ba-bfb7-7a282b3ca921"}</td><td>1</td></tr><tr><td>kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="monitoring", phase="Running", pod="alertmanager-prometheus-kube-prometheus-alertmanager-0", service="prometheus-kube-state-metrics", uid="76a3c5ae-6324-432b-b59a-48afdc04bd38"}</td><td>1</td></tr><tr><td>kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kubernetes-dashboard", phase="Running", pod="kubernetes-dashboard-b84665f8-skp7v", service="prometheus-kube-state-metrics", uid="c55e076-9fe4-4510-b916-9b372e3a2d0f"}</td><td>1</td></tr><tr><td>kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="monitoring", phase="Running", pod="prometheus-prometheus-kube-prometheus-prometheus-0", service="prometheus-kube-state-metrics", uid="1b271ab3-d898-4ed9-8c8d-e313fe53af0f"}</td><td>1</td></tr><tr><td>kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="coredns-767446669-75nps", service="prometheus-kube-state-metrics", uid="a0ff014c-9378-472c-84d6-8b039a567672"}</td><td>1</td></tr><tr><td>kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="kube-controller-manager-minikube", service="prometheus-kube-state-metrics", uid="a41fe235-3178-48c0-9839-5e523b3ef0da"}</td><td>1</td></tr><tr><td>kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="default", phase="Running", pod="demo-webapp-5946b874d-bhz14", service="prometheus-kube-state-metrics", uid="161a7ca8-fe88-471d-b488-485e4b18956a"}</td><td>1</td></tr><tr><td>kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="monitoring", phase="Running", pod="prometheus-prometheus-node-exporter-e9k44", service="prometheus-kube-state-metrics", uid="f988b104-e3df-4285-b2de-68e14e82ae7f"}</td><td>1</td></tr><tr><td>kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="monitoring", phase="Running", pod="prometheus-grafana-568b547658-mj2c", service="prometheus-kube-state-metrics", uid="5f131308-c197-485c-b2e2-9ec14239e43f"}</td><td>1</td></tr><tr><td>kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="kube-apiserver-minikube", service="prometheus-kube-state-metrics", uid="0d480ada-c0eb-4a07-ae17-c06d5c4d078f"}</td><td>1</td></tr><tr><td>kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="kube-proxy-rxwcd", service="prometheus-kube-state-metrics", uid="1ed982c-7de-4273-b556-7076a70541c3"}</td><td>1</td></tr><tr><td>kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kubernetes-dashboard", phase="Running", pod="dashboard-metrics-scraper-5565989548-7hz4", service="prometheus-kube-state-metrics", uid="43aea975-73c5-48b1-80c0-6d9a916a70b3"}</td><td>1</td></tr><tr><td>kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="etcd-minikube", service="prometheus-kube-state-metrics", uid="4e810a9a-c395-42fe-8a8c-0a798e3a98c9"}</td><td>1</td></tr><tr><td>kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="monitoring", phase="Running", pod="prometheus-kube-state-metrics-678b9d8d-b7m7", service="prometheus-kube-state-metrics", uid="1a650d3f-e93b-4236-ab50-2a78d16a1a8f"}</td><td>1</td></tr></table>	kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="storage-provisioner", service="prometheus-kube-state-metrics", uid="b9d1a374-b008-48ba-bfb7-7a282b3ca921"}	1	kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="monitoring", phase="Running", pod="alertmanager-prometheus-kube-prometheus-alertmanager-0", service="prometheus-kube-state-metrics", uid="76a3c5ae-6324-432b-b59a-48afdc04bd38"}	1	kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kubernetes-dashboard", phase="Running", pod="kubernetes-dashboard-b84665f8-skp7v", service="prometheus-kube-state-metrics", uid="c55e076-9fe4-4510-b916-9b372e3a2d0f"}	1	kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="monitoring", phase="Running", pod="prometheus-prometheus-kube-prometheus-prometheus-0", service="prometheus-kube-state-metrics", uid="1b271ab3-d898-4ed9-8c8d-e313fe53af0f"}	1	kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="coredns-767446669-75nps", service="prometheus-kube-state-metrics", uid="a0ff014c-9378-472c-84d6-8b039a567672"}	1	kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="kube-controller-manager-minikube", service="prometheus-kube-state-metrics", uid="a41fe235-3178-48c0-9839-5e523b3ef0da"}	1	kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="default", phase="Running", pod="demo-webapp-5946b874d-bhz14", service="prometheus-kube-state-metrics", uid="161a7ca8-fe88-471d-b488-485e4b18956a"}	1	kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="monitoring", phase="Running", pod="prometheus-prometheus-node-exporter-e9k44", service="prometheus-kube-state-metrics", uid="f988b104-e3df-4285-b2de-68e14e82ae7f"}	1	kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="monitoring", phase="Running", pod="prometheus-grafana-568b547658-mj2c", service="prometheus-kube-state-metrics", uid="5f131308-c197-485c-b2e2-9ec14239e43f"}	1	kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="kube-apiserver-minikube", service="prometheus-kube-state-metrics", uid="0d480ada-c0eb-4a07-ae17-c06d5c4d078f"}	1	kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="kube-proxy-rxwcd", service="prometheus-kube-state-metrics", uid="1ed982c-7de-4273-b556-7076a70541c3"}	1	kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kubernetes-dashboard", phase="Running", pod="dashboard-metrics-scraper-5565989548-7hz4", service="prometheus-kube-state-metrics", uid="43aea975-73c5-48b1-80c0-6d9a916a70b3"}	1	kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="etcd-minikube", service="prometheus-kube-state-metrics", uid="4e810a9a-c395-42fe-8a8c-0a798e3a98c9"}	1	kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="monitoring", phase="Running", pod="prometheus-kube-state-metrics-678b9d8d-b7m7", service="prometheus-kube-state-metrics", uid="1a650d3f-e93b-4236-ab50-2a78d16a1a8f"}	1	
kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="storage-provisioner", service="prometheus-kube-state-metrics", uid="b9d1a374-b008-48ba-bfb7-7a282b3ca921"}	1																												
kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="monitoring", phase="Running", pod="alertmanager-prometheus-kube-prometheus-alertmanager-0", service="prometheus-kube-state-metrics", uid="76a3c5ae-6324-432b-b59a-48afdc04bd38"}	1																												
kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kubernetes-dashboard", phase="Running", pod="kubernetes-dashboard-b84665f8-skp7v", service="prometheus-kube-state-metrics", uid="c55e076-9fe4-4510-b916-9b372e3a2d0f"}	1																												
kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="monitoring", phase="Running", pod="prometheus-prometheus-kube-prometheus-prometheus-0", service="prometheus-kube-state-metrics", uid="1b271ab3-d898-4ed9-8c8d-e313fe53af0f"}	1																												
kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="coredns-767446669-75nps", service="prometheus-kube-state-metrics", uid="a0ff014c-9378-472c-84d6-8b039a567672"}	1																												
kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="kube-controller-manager-minikube", service="prometheus-kube-state-metrics", uid="a41fe235-3178-48c0-9839-5e523b3ef0da"}	1																												
kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="default", phase="Running", pod="demo-webapp-5946b874d-bhz14", service="prometheus-kube-state-metrics", uid="161a7ca8-fe88-471d-b488-485e4b18956a"}	1																												
kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="monitoring", phase="Running", pod="prometheus-prometheus-node-exporter-e9k44", service="prometheus-kube-state-metrics", uid="f988b104-e3df-4285-b2de-68e14e82ae7f"}	1																												
kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="monitoring", phase="Running", pod="prometheus-grafana-568b547658-mj2c", service="prometheus-kube-state-metrics", uid="5f131308-c197-485c-b2e2-9ec14239e43f"}	1																												
kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="kube-apiserver-minikube", service="prometheus-kube-state-metrics", uid="0d480ada-c0eb-4a07-ae17-c06d5c4d078f"}	1																												
kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="kube-proxy-rxwcd", service="prometheus-kube-state-metrics", uid="1ed982c-7de-4273-b556-7076a70541c3"}	1																												
kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kubernetes-dashboard", phase="Running", pod="dashboard-metrics-scraper-5565989548-7hz4", service="prometheus-kube-state-metrics", uid="43aea975-73c5-48b1-80c0-6d9a916a70b3"}	1																												
kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="kube-system", phase="Running", pod="etcd-minikube", service="prometheus-kube-state-metrics", uid="4e810a9a-c395-42fe-8a8c-0a798e3a98c9"}	1																												
kube_pod_status_phase{container="kube-state-metrics", endpoint="http", instance="10.244.0.11:8080", job="kube-state-metrics", namespace="monitoring", phase="Running", pod="prometheus-kube-state-metrics-678b9d8d-b7m7", service="prometheus-kube-state-metrics", uid="1a650d3f-e93b-4236-ab50-2a78d16a1a8f"}	1																												

Εικόνα 5.18 — Prometheus — `kube_pod_status_phase{phase="Running"}`: 18 εκτελούμενα Pods, χρόνος απόκρισης 21 ms

Ανάλυση. Το ερώτημα PromQL επιστρέφει 18 pods σε κατάσταση Running σε 21 ms, καταναμεμημένα ως εξής: default (demo-webapp-5946b874d-bhz14), monitoring (alertmanager, prometheus, grafana, node-exporter, kube-state-metrics), kube-system (coredns, kube-controller-manager, kube-apiserver, kube-proxy, etcd, kube-scheduler, storage-provisioner) και kubernetes-dashboard (δύο pods). Το ερώτημα αυτό αποτελεί ακριβώς τον μηχανισμό που χρησιμοποιεί το PrometheusService για την τροφοδότηση του MonitoringService κατά την περιοδική αξιολόγηση των QoS κανόνων (§4.4.4). Επαληθεύει την απαίτηση FA-A08. Το πλήρες σύνολο των ερωτημάτων αναφοράς παρατίθεται στον Πίνακα Α.1.

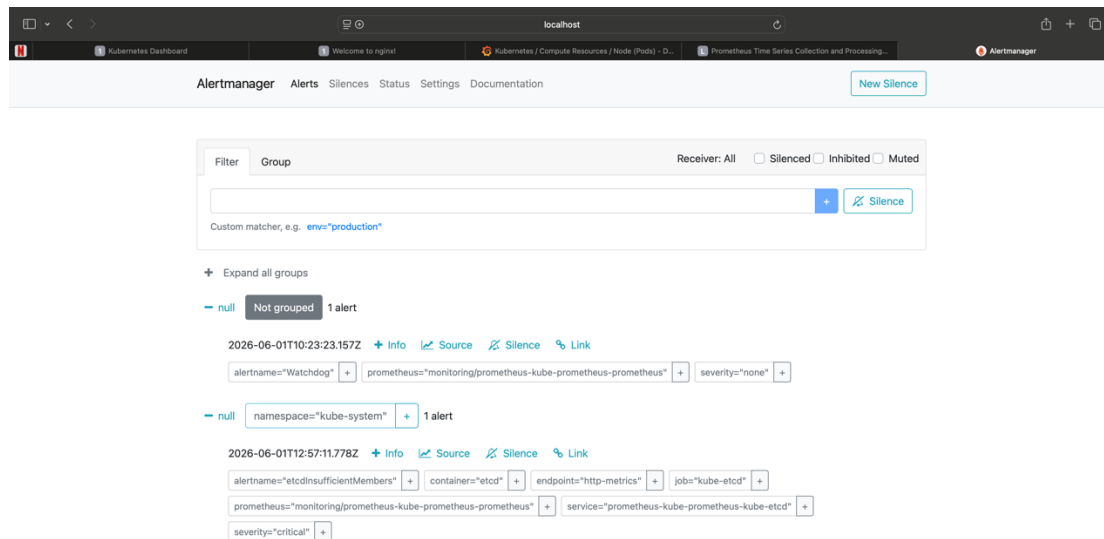
5.2.4.6 Ενεργές ειδοποιήσεις



Εικόνα 5.19 — Alertmanager: Δύο Ενεργές Ομάδες Ειδοποιήσεων

Ανάλυση. Η διεπαφή του Alertmanager στο `localhost:9093` εμφανίζει δύο ενεργές ομάδες: «Not grouped: 1 alert» και «namespace=kube-system: 1 alert». Η ύπαρξή τους αποδεικνύει λειτουργικό αγωγό ειδοποιήσεων: Prometheus → αξιολόγηση κανόνων → Alertmanager → δρομολόγηση. Διακρίνονται οι λειτουργίες φιλτραρίσματος, ομαδοποίησης, καθώς και οι όψεις Silenced/Inhibited/Muted και New Silence. Επισημαίνεται ότι οι δύο αυτές ειδοποιήσεις αφορούν τη γενική υγεία της **υποδομής παρακολούθησης** (έλεγχος λειτουργίας του ίδιου του Alertmanager και κατάσταση του μονο-κομβικού etcd στο Minikube) και **όχι** πυροδότηση QoS κανόνα κλιμάκωσης· οι τελευταίες καταγράφονται ξεχωριστά ως εγγραφές ScalingEvent στη βάση δεδομένων (Παράρτημα Ε). Επισημαίνεται ότι το ιστορικό αυτό δεν εκτίθεται μέσω γραφικής διεπαφής στην παρούσα έκδοση.

5.2.4.7 Ανάλυση των ειδοποιήσεων



Εικόνα 5.20 — Alertmanager: Watchdog (severity = none) και etcdInsufficientMembers (severity = critical)

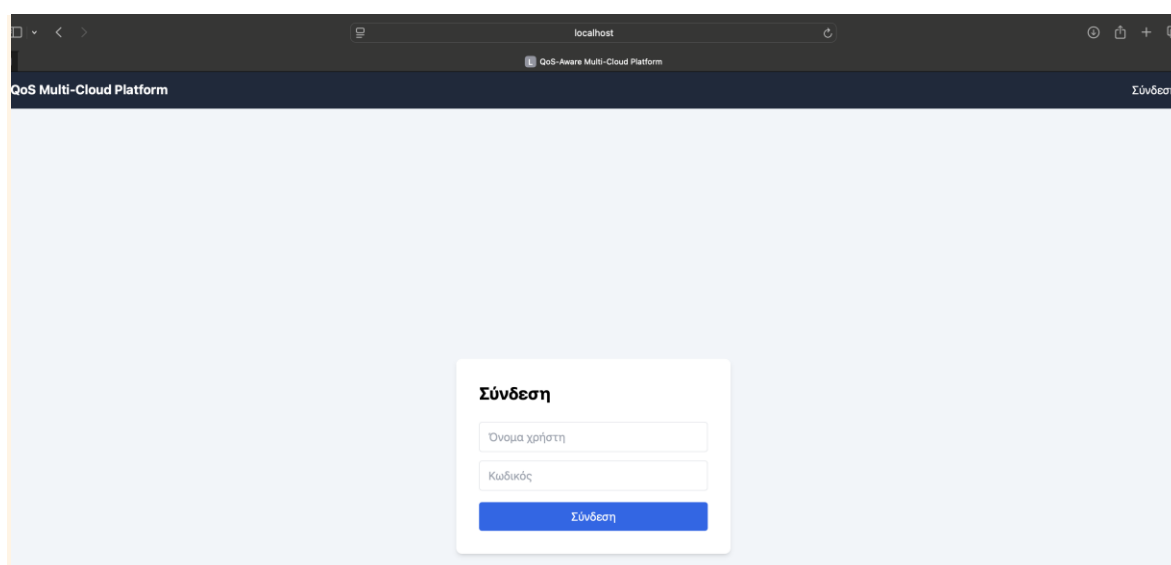
Ανάλυση. Η πρώτη ειδοποίηση, Watchdog (10:23:23Z, severity = none), είναι ειδοποίηση-δείκτης (canary) που πυροδοτείται μονίμως εκ σχεδιασμού: η **απουσία** της θα υποδήλωνε βλάβη του ίδιου του αγωγού ειδοποιήσεων. Η δεύτερη, etcdInsufficientMembers (12:57:11Z, container = etcd, job = kube-etcd, severity = critical), είναι αναμενόμενη σε μονο-κομβικό περιβάλλον Minikube, όπου το etcd λειτουργεί χωρίς απαρτία (quorum) τριών μελών· σε παραγωγικό πολυκομβική συστάδα EKS ή AKS (Παραρτήματα Γ και Δ) η ειδοποίηση αυτή δεν θα εμφανιζόταν. Η ορθή ταξινόμηση και των δύο ειδοποιήσεων επαληθεύει μερικώς την απαίτηση FA-A09: τεκμηριώνεται λειτουργικός αγωγός ειδοποιήσεων για την υγεία της υποδομής, όχι όμως ειδοποίηση από παραβίαση κανόνα ποιότητας υπηρεσίας ούτε από εκτελεσθείσα ενέργεια κλιμάκωσης.

5.2.5 Σενάριο 4: Η Διαδικτυακή Διεπαφή της Πλατφόρμας

Στόχος. Τεκμηρίωση του εύρους της διαδικτυακής διεπαφής που υλοποιήθηκε. Σε αντίθεση με τα Σενάρια 1–3, τα οποία τεκμηριώνουν τη συμπεριφορά της συστάδας, το παρόν σενάριο τεκμηριώνει τον τρόπο με τον οποίο ο χρήστης αλληλεπιδρά με την πλατφόρμα. Επισημαίνεται ρητά ότι η διεπαφή αποτελεί ελάχιστη υλοποίηση αναφοράς και καλύπτει την ταυτοποίηση και τη διαχείριση εφαρμογών (§4.5)· ο ορισμός κανόνων ποιότητας υπηρεσίας, η εκκίνηση της διάταξης, η εποπτεία μετρικών και το ιστορικό ενεργειών κλιμάκωσης εκτελούνται από τις αντίστοιχες υπηρεσίες του backend και τεκμηριώνονται στα Σενάρια 1–3 και στα Παραρτήματα Α και Ε, χωρίς να διαθέτουν γραφικές οθόνες.

Οι ροές αντιστοιχούν στα Sequence Diagrams των Εικόνων 4.5 και 4.6 και υλοποιούνται από το Web UI (React SPA) σε συνεργασία με τα UserService και ApplicationService (Πίνακας 5.2).

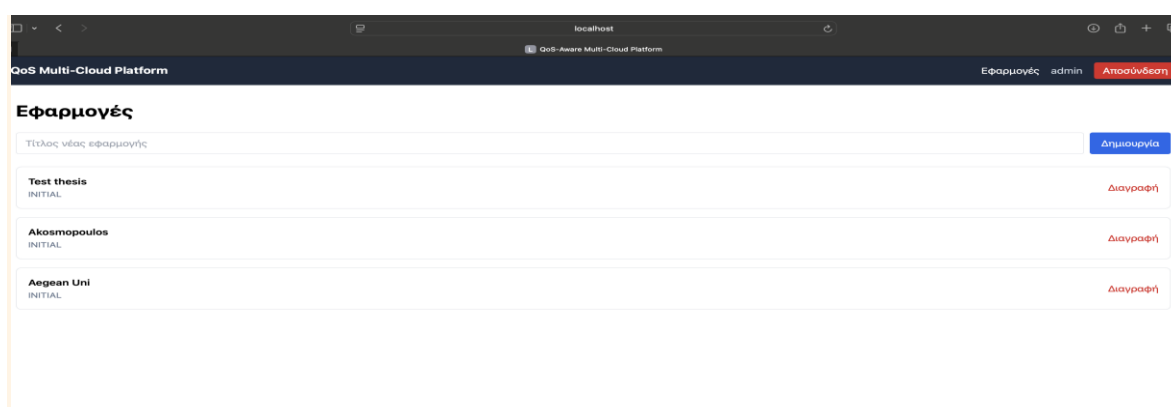
5.2.5.1 Ταυτοποίηση χρήστη



Εικόνα 5.21 — Web UI: Οθόνη Σύνδεσης

Ανάλυση. Η οθόνη εισόδου υποστηρίζει αυθεντικοποίηση με όνομα χρήστη και κωδικό, ο οποίος αποθηκεύεται ως κατακερματισμός BCrypt στον πίνακα users (Παράρτημα Ε). Μετά την επιτυχή σύνδεση εκδίδεται JWT, το οποίο αποθηκεύεται στο authStore και συνοδεύει κάθε επακόλουθη κλήση REST μέσω του κοινού στιγμιότυπου Axios. Η ροή αντιστοιχεί στο Sequence Diagram «Σύνδεση Χρήστη» (Εικόνα 4.6) και επαληθεύει την απαίτηση FA-U03 (σύνδεση χρήστη) και την MFA-S01. Οι FA-U01 και FA-U02 —εγγραφή και ενεργοποίηση λογαριασμού— δεν καλύπτονται από το παρόν τεκμήριο, καθώς προϋποθέτουν βήματα που δεν αποτυπώνονται στην επίδειξη.

5.2.5.2 Διαχείριση εφαρμογών



Εικόνα 5.22 — Web UI: Κατάλογος και Δημιουργία Εφαρμογών²²

Ανάλυση. Η οθόνη εκθέτει τη φόρμα δημιουργίας νέας εφαρμογής και τον κατάλογο των εφαρμογών του συνδεδεμένου χρήστη, με ένδειξη της κατάστασης (AppStatus) και ενέργεια διαγραφής ανά εγγραφή. Οι κλήσεις των προαναφερόμενων λειτουργιών εκτελούνται μέσω των POST και GET /api/v1/applications και DELETE /api/v1/applications/{id} (Πίνακας 4.8), με ανάκτηση και ακύρωση προσωρινής μνήμης μέσω React Query. Η ροή υλοποιείται από τη σελίδα εφαρμογών σε συνεργασία με το ApplicationService και επαληθεύει την απαίτηση FA-A01 ως προς

τη δημιουργία και την εμφάνιση της εφαρμογής στον κατάλογο, καθώς και την FA-A10 ως προς τη διαθέσιμη ενέργεια διαγραφής. Δεν καλύπτονται η δήλωση διαμόρφωσης νέφους και ο ορισμός κανόνων ποιότητας υπηρεσίας, τα οποία δεν εκτίθενται από τη διεπαφή (§4.4.6). Η διαμόρφωση νέφους, οι κανόνες QoS και η εκκίνηση της διάταξης δεν εκτίθενται από την οθόνη αυτή: υποστηρίζονται από το μοντέλο πεδίου και τα σημεία πρόσβασης του API, αλλά απαιτούν κλήση εκτός διεπαφής.

5.3 Σύνοψη Επαλήθευσης Απαιτήσεων μέσω της Επίδειξης

Ο **Πίνακας 5.4** αντιστοιχίζει κάθε απαίτηση του Κεφαλαίου 4 στο συγκεκριμένο τεκμήριο της επίδειξης που την επαληθεύει. Ο πίνακας αυτός αποτελεί τη γέφυρα μεταξύ της στατικής αξιολόγησης της §4.7 (**Πίνακες 4.14–4.25**) και της εμπειρικής επαλήθευσης του παρόντος κεφαλαίου, και αποτελεί την τεκμηρίωση στην οποία στηρίζονται τα συμπεράσματα της §6.1.

Πίνακας 5.4 — Αντιστοίχιση Απαιτήσεων και Τεκμηρίων Επίδειξης

Απαίτηση	Περιγραφή (συνοπτικά)	Σενάριο	Τεκμήριο
FA-U03	Σύνδεση χρήστη στην πλατφόρμα (η εγγραφή και η ενεργοποίηση δεν καλύπτονται)	4	Εικ. 5.21
FA-A01 (μερικώς)	Δημιουργία και εμφάνιση εφαρμογής· χωρίς διαμόρφωση νέφους και κανόνες QoS	4	Εικ. 5.22
FA-A02	Ορατότητα κατάστασης συστάδας και εφαρμογών	1, 4	Εικ. 5.4, 5.6, 5.8, 5.22
FA-A04	Διάταξη και έκθεση εφαρμογής στη συστάδα	1	Εικ. 5.5, 5.7, 5.9
FA-A05	Αποδιάταξη υποδομής	—	Παράρτημα Α.4
FA-A08	Συλλογή μετρικών και εκτέλεση κανόνα κλιμάκωσης στο επίπεδο της συστάδας	2, 3	Εικ. 5.10, 5.14, 5.15, 5.17
FA-A08	Παρακολούθηση μετρικών σε πραγματικό χρόνο	2, 3	Εικ. 5.12, 5.13, 5.16, 5.18
FA-A09 (μερικώς)	Λειτουργικός αγωγός ειδοποιήσεων υγείας υποδομής· όχι ειδοποιήσεις QoS	3	Εικ. 5.19, 5.20
MFA-P01	Χρόνος απόκρισης ερωτημάτων μετρικών	3	Εικ. 5.17 (20 ms), 5.18 (21 ms)
MFA-S01	Ασφαλής ταυτοποίηση	2, 4	Εικ. 5.11, 5.21

MFA-U01	Απόκριση της διεπαφής σε τυπικές αναλύσεις οθόνης	1, 4	Εικ. 5.8, 5.21, 5.22
---------	---	------	----------------------

Σημείωση. Η απαίτηση FA-A05 (αποδιάταξη) δεν τεκμηριώνεται με στιγμιότυπο, καθώς το αποτέλεσμα της είναι η απουσία πόρων. Υποστηρίζεται ωστόσο από το ακόλουθο σενάριο επαλήθευσης, το οποίο μπορεί να εκτελεστεί μετά το Σενάριο 3: (α) καταγράφεται η αρχική κατάσταση με `kubectl get deployments,services,hpa`, όπου εμφανίζονται το `demo-webapp`, η υπηρεσία `NodePort` και ο `HorizontalPodAutoscaler`. (β) εκτελείται η αποδιάταξη με τις εντολές του Παραρτήματος Α.4. (γ) επαναλαμβάνεται η ίδια εντολή καταγραφής, η οποία πλέον επιστρέφει `No resources found in default namespace`. (δ) επαληθεύεται ότι τα αντίστοιχα σημεία μετρικών παύουν να αναφέρονται στο `Prometheus`. Η αντιπαραβολή των δύο εξόδων της ίδιας εντολής, πριν και μετά, αποτελεί θετική τεκμηρίωση της απαίτησης παρά τον αρνητικό χαρακτήρα του αποτελέσματος. Οι υπόλοιπες απαιτήσεις των **Πινάκων 4.2–4.5** αξιολογούνται αναλυτικά στην §4.7.

5.4 Περιορισμοί της επίδειξης

Για λόγους επιστημονικής ακρίβειας επισημαίνονται οι ακόλουθοι περιορισμοί:

1. **Τοπικό, μονο-κομβικό περιβάλλον.** Η επίδειξη διεξήχθη σε Minikube και όχι σε παραγωγική συστάδα EKS ή AKS. Συνέπειες: η ειδοποίηση `etcdInsufficientMembers` (**Εικόνα 5.20**) είναι δομικά αναπόφευκτη, ενώ οι μετρημένοι χρόνοι απόκρισης (**Εικόνες 5.17** και **5.18**) δεν περιλαμβάνουν καθυστέρηση δικτύου μεταξύ περιοχών.
2. **Ελάχιστη εφαρμογή επίδειξης.** Το `nginx:alpine` δεν εκθέτει μετρικές επιπέδου εφαρμογής (latency, error rate). Ως εκ τούτου, το Σενάριο 3 επαληθεύει τον μηχανισμό QoS με μετρική CPU· η υποστήριξη προσαρμοσμένων μετρικών προκύπτει από το πεδίο `metricName` της `QosRule` και από τη γενική μορφή των ερωτημάτων PromQL του `PrometheusService`, χωρίς όμως να έχει επαληθευτεί πειραματικά (Παράρτημα Β).
3. **Κλιμάκωση κόμβων.** Επιδεικνύεται κλιμάκωση pods (`SCALE_PODS_UP/DOWN`). Η κλιμάκωση κόμβων (`SCALE_NODES_UP/DOWN`) υποστηρίζεται από τα Terraform modules (Παράρτηματα Γ και Δ) αλλά δεν είναι επιδείξιμη σε μονο-κομβικό Minikube.
4. **Ένας πάροχος τη φορά.** Η ταυτόχρονη διάταξη της ίδιας εφαρμογής σε AWS και Azure υποστηρίζεται από τη σχεδίαση (πίνακας `cloud_configs`, Παράρτημα Ε) αλλά δεν επιδεικνύεται, λόγω κόστους παραγωγικών πόρων νέφους.

Οι περιορισμοί 2–4 δεν αναιρούν την επαλήθευση του μηχανισμού· οριοθετούν το εύρος της εμπειρικής τεκμηρίωσης και τροφοδοτούν άμεσα τις προτεινόμενες επεκτάσεις της §6.2.

Κεφάλαιο 6: Συμπεράσματα και Μελλοντική Εργασία

6.1 Συμπεράσματα

6.1.1 Σύνοψη Επιτευγμάτων

Η παρούσα διπλωματική εργασία ανέπτυξε μια QoS-aware, multi-cloud πλατφόρμα αυτόματης κλιμάκωσης και παρακολούθησης φόρτων εργασίας Kubernetes. Η εργασία απαντά σε ερευνητικό κενό που τεκμηριώθηκε στην ανάλυση σχετικών εργασιών (Κεφάλαιο 3, **Πίνακας 3.1**): ενώ υπάρχουν ώριμες λύσεις σε επιμέρους τομείς — ασφάλεια της αυτόματης κλιμάκωσης (Ben David & Bremner-Barr, 2021), αυτόματη διαμόρφωση πόρων σε κλίμακα παραγωγής (Rzadca et al., 2020), ομοσπονδιακή παρακολούθηση γεω-κατανεμημένων συστάδων μέσω Prometheus federation (Huang & Pierre, 2023), δηλωτική διαχείριση υποδομής μέσω του Kubernetes API (Crossplane Authors, 2024) και vendor-agnostic ενορχήστρωση βάσει TOSCA (Tomarchio et al., 2021) — καμία δεν τις ενοποιεί σε φιλική προς τον χρήστη διαδικτυακή πλατφόρμα με κανόνες ποιότητας υπηρεσίας οριζόμενους από τον χρήστη. Τα κεντρικά επιτεύγματα συνοψίζονται ως εξής:

Ενοποιημένο επίπεδο υπηρεσιών. Η πλατφόρμα ενοποιεί σε ενιαίο μοντέλο πεδίου και ενιαίο σύνολο υπηρεσιών τη δήλωση εφαρμογών, τη διαμόρφωση νέφους, τους κανόνες ποιότητας υπηρεσίας, τη διάταξη υποδομής, την παρακολούθηση μετρικών και το ιστορικό ενεργειών κλιμάκωσης, ανεξαρτήτως παρόχου. Η διαδικτυακή διεπαφή υλοποιήθηκε ως ελάχιστος πελάτης αναφοράς και καλύπτει την ταυτοποίηση και τη διαχείριση εφαρμογών (§4.5, §5.2.5)· η γραφική έκθεση των υπόλοιπων λειτουργιών αποτελεί την πρώτη προτεινόμενη επέκταση (§6.2).

Προσαρμοσμένοι κανόνες ποιότητας υπηρεσίας. Κανόνες της μορφής `metric + operator + threshold + action` αξιολογούνται περιοδικά ανά 30 δευτερόλεπτα από το `MonitoringService`. Ο μηχανισμός επαληθεύτηκε εμπειρικά: η εκτέλεση του κανόνα καταγράφεται στο επίπεδο της συστάδας (Εικόνα 5.15) και αποτυπώνεται ως εγγραφή `ScalingEvent` στη βάση δεδομένων (Παράρτημα Ε). Η συνολική αντιστοίχιση απαιτήσεων και τεκμηρίων παρατίθεται στην §5.3 (**Πίνακας 5.4**).

Αυτοματισμός διάταξης και κλιμάκωσης μέσω IaC. Χρήση Terraform (HashiCorp, 2024) για τη δημιουργία συστάδων Kubernetes σε AWS EKS και Azure AKS, με δυνατότητα πλήρους καταστροφής της υποδομής με μία ενέργεια (Παραρτήματα Γ και Δ).

Εμπειρική επαλήθευση λειτουργικότητας. Η επίδειξη του Κεφαλαίου 5 τεκμηρίωσε σε πραγματικό περιβάλλον Kubernetes τη διάταξη εφαρμογής, την παρακολούθηση πόρων, την αυτόματη κλιμάκωση βάσει QoS κανόνα και την παραγωγή ειδοποιήσεων. Οι περιορισμοί του πειραματικού περιβάλλοντος δηλώνονται ρητά στο τέλος της §5.3.

Αποτίμηση απαιτήσεων. Από τις 47 βασικές απαιτήσεις, 24 (51,1%) ικανοποιούνται πλήρως και 13 (27,7%) μερικώς, ενώ 10 (21,3%) δεν ικανοποιούνται στην παρούσα έκδοση (Πίνακας 4.25). Οι μερικώς ικανοποιημένες υποστηρίζονται από τις υπηρεσίες του backend χωρίς να εκτίθενται μέσω διεπαφής, ενώ οι μη

ικανοποιημένες αφορούν κυρίως δευτερεύουσες λειτουργίες διαχείρισης λογαριασμού και εφαρμογών.

6.1.2 Τεχνολογικά Πλεονεκτήματα

Πίνακας 6.1 — Συγκριτικά Πλεονεκτήματα της Πλατφόρμας έναντι Τυπικών Εναλλακτικών

Διάσταση	Παρούσα πλατφόρμα	Τυπική εναλλακτική
Ευκολία χρήσης	Ενιαία διεπαφή και REST API· η ταυτοποίηση και η διαχείριση εφαρμογών χωρίς CLI (Εικ. 5.21, 5.22)	<code>kubectl / eksctl / az aks</code> — απαιτείται εμπειρογνωμοσύνη γραμμής εντολών
Πολλαπλά νέφη	AWS και Azure από ενιαία διεπαφή· GCP σχεδιασμένο (§6.2.1)	Ξεχωριστές κονσόλες και εργαλεία ανά πάροχο
Προσαρμοσμένη ποιότητα υπηρεσίας	Κανόνες της μορφής <code>metric + operator + threshold + action</code> με υποστήριξη οποιασδήποτε μετρικής PromQL, αξιολογούμενοι ανά 30 s	Kubernetes HPA: εγγενώς μόνο CPU/μνήμη· custom μετρικές απαιτούν adapter ή συνδυασμό με cloud-specific autoscalers, με διαφορετική ρύθμιση ανά πάροχο
Ανεξαρτησία από πάροχο	Στοιβά ανοικτού κώδικα: Prometheus, Kubernetes, Terraform	Εγγενείς υπηρεσίες: CloudWatch, Azure Monitor (εγκλωβισμός σε πάροχο)
Αποδιάταξη υποδομής	<code>terraform destroy</code> με μία κλήση API (DELETE /applications/{id}/deployment)	Ξεχωριστή εντολή ανά πάροχο ή ανά περιβάλλον
Ιχνηλασιμότητα ενεργειών	Καταγραφή κάθε ενέργειας ως ScalingEvent στη βάση (Παράρτημα Παράρτημα Ε)	Διάσπαρτα σε <code>kubectl describe hpa</code> , cloud audit logs και αρχεία καταγραφής παρόχου

Σημείωση. Η σύγκριση αφορά την εμπειρία του τελικού χρήστη και όχι τις τεχνικές δυνατότητες των υποκείμενων εργαλείων, τα οποία η πλατφόρμα αξιοποιεί αντί να αντικαθιστά.

6.1.3 Προσωπική Εμπειρία

Η εκπόνηση της διπλωματικής υπήρξε ιδιαίτερα εποικοδομητική εμπειρία. Αποκτήθηκαν βαθιές πρακτικές γνώσεις σε σύγχρονες τεχνολογίες: Kubernetes (μηχανισμοί σταθεροποίησης του HPA, PromQL, `kube-state-metrics`), Terraform (πρότυπα IaC, διαχείριση κατάστασης, σχεδίαση modules), Spring Boot 3.2 (ασύγχρονος χρονοπρογραμματισμός, `ProcessBuilder`, Spring Security 6) και React.js 18 (γραφήματα πραγματικού χρόνου με `Recharts`, `React Query`, `Zustand`). Ιδιαίτερα διδακτική υπήρξε η αντιμετώπιση πραγματικών προκλήσεων: η κατανόηση του μηχανισμού σταθεροποίησης του HPA — ο οποίος εξηγεί την ταλάντωση 3 → 2 → 3 αντιγράφων της **Εικόνας 5.15** —, η ορθή διατύπωση και ερμηνεία ερωτημάτων PromQL, και η συσχέτισή τους με την πραγματική συμπεριφορά της εφαρμογής μέσω των πινάκων ελέγχου του Grafana.

6.2 Μελλοντική Εργασία

Οι προτεινόμενες επεκτάσεις οργανώνονται σε δύο κατηγορίες: βραχυπρόθεσμες (0–6 μήνες, χαμηλή πολυπλοκότητα, άμεσο αντίκτυπο — §6.2.1) και μεσοπρόθεσμες έως μακροπρόθεσμες (6 μήνες και άνω, υψηλότερη πολυπλοκότητα — §6.2.2 έως §6.2.5). Για κάθε επέκταση περιγράφεται τι επιτυγχάνει, ποια αρχιτεκτονικά στοιχεία επηρεάζει (με βάση την ονοματολογία του **Πίνακα 5.2**) και ποιες τεχνικές προκλήσεις παρουσιάζει.

6.2.1 Βραχυπρόθεσμες Επεκτάσεις

Ολοκλήρωση της διαδικτυακής διεπαφής και των σημείων πρόσβασης. Πρώτη κατά σειρά προτεραιότητας επέκταση. Απαιτεί την προσθήκη ελεγκτών για τους κανόνες ποιότητας υπηρεσίας, τις μετρικές, τις ειδοποιήσεις και το ιστορικό ενεργειών κλιμάκωσης — το μοντέλο πεδίου και οι αντίστοιχες υπηρεσίες υπάρχουν ήδη (§4.4.2), οπότε πρόκειται κυρίως για έκθεση υφιστάμενης λειτουργικότητας — και, στη συνέχεια, τις αντίστοιχες οθόνες: οδηγό δημιουργίας εφαρμογής με επιλογή παρόχου και διαμόρφωση υποδομής, συντάκτη κανόνων QoS, πίνακα ελέγχου μετρικών με γραφήματα Recharts και κέντρο ειδοποιήσεων. Η επέκταση αυτή αίρει το μεγαλύτερο μέρος των μερικώς ικανοποιημένων απαιτήσεων του Πίνακα 4.25 και αποτελεί προϋπόθεση για τις υπόλοιπες επεκτάσεις της ενότητας αυτής.

Υποστήριξη GCP (Google Kubernetes Engine). Προσθήκη Terraform module για GKE, ώστε να ολοκληρωθεί η τριάδα AWS + Azure + GCP. Απαιτεί δημιουργία Google Service Account με τους ρόλους `container.admin` και `compute.admin`. Η ενσωμάτωση είναι άμεση, καθώς το `PrometheusService` υποστηρίζει ήδη πολλαπλά endpoints και το `TerraformService` είναι αγνωστικιστικό ως προς τον πάροχο. Θα αντιμετωπίζε επίσης τον περιορισμό 4 της §5.3.

On-premises και ιδιωτικό νέφος. Επέκταση σε υποδομές εντός εγκατάστασης μέσω `kubeadm` ή `OpenShift`. Ο χρήστης θα μπορεί να συνδέσει υφιστάμενη συστάδα παρέχοντας αρχείο `kubeconfig`, χωρίς να απαιτείται δημιουργία νέας μέσω Terraform. Καλύπτει σενάρια υβριδικού νέφους και απαιτεί ξεχωριστή διαδρομή στο `TerraformService` για συστάδες που δεν διαχειρίζεται η πλατφόρμα.

Διάταξη σε πολλαπλές γεωγραφικές περιοχές. Υποστήριξη διάταξης της ίδιας εφαρμογής σε συστάδες διαφορετικών περιοχών (π.χ. AWS `eu-west-1` και `us-east-1`). Απαιτεί καθολικό εξισορροπητή φορτίου (AWS Route 53 ή Azure Traffic Manager) και στρατηγική διαχείρισης κατανεμημένων δεδομένων.

6.2.2 Αυτόματη Κλιμάκωση Βασισμένη σε Μηχανική Μάθηση

Προβλεπτική κλιμάκωση. Αντί για αντιδραστική κλιμάκωση μετά την παραβίαση ορίου, εκπαίδευση μοντέλων LSTM ή Prophet επί ιστορικών μετρικών. Το μοντέλο προβλέπει τον μελλοντικό φόρτο και κλιμακώνει προληπτικά, πριν παραβιαστεί ο QoS στόχος — προσέγγιση συγγενής προς εκείνη του εσωτερικού συστήματος Autopilot της Google, το οποίο αξιοποιεί ιστορικά δεδομένα εκτέλεσης για τη διαμόρφωση ορίων πόρων (Rzadca et al., 2020). Το πρόβλημα της ψυχρής εκκίνησης αντιμετωπίζεται με χρήση γενικών μοντέλων έως ότου συσσωρευτεί επαρκές ιστορικό ανά εφαρμογή.

Ενισχυτική μάθηση. Πρόκειται για **συμπληρωματική** και όχι εναλλακτική προσέγγιση ως προς την προβλεπτική κλιμάκωση: οι δύο μηχανισμοί απαντούν σε διαφορετικά ερωτήματα. Τα μοντέλα LSTM/Prophet προβλέπουν *πότε* θα παραβιαστεί ένα όριο μετρικής, ενώ ένας πράκτορας ενισχυτικής μάθησης μαθαίνει *ποια* ενέργεια κλιμάκωσης είναι βέλτιστη δεδομένης της κατάστασης. Οι δύο μηχανισμοί μπορούν επομένως να συνδυαστούν, με τον πρώτο να τροφοδοτεί έγκαιρα σήματα στον δεύτερο. Η συνάρτηση ανταμοιβής ισορροπεί μεταξύ ικανοποίησης του SLA και ελαχιστοποίησης της χρήσης πόρων.

Ανίχνευση ανωμαλιών. Εφαρμογή μη επιβλεπόμενων μεθόδων (Isolation Forest, autoencoders) για αυτόματη ανίχνευση ανωμαλιών χωρίς ρητά κατώφλια QoS. Αφορά πρωτίστως τη λειτουργική υγεία (ασυνήθιστες αιχμές καθυστέρησης ή σφαλμάτων), με δυνατότητα επέκτασης και σε ανίχνευση ύποπτης δραστηριότητας (π.χ. ασυνήθιστα μοτίβα σύνδεσης). Ο χρήστης δεν χρειάζεται να ορίσει κανόνες: το σύστημα μαθαίνει τη φυσιολογική συμπεριφορά της εφαρμογής. Θα αντιμετωπίζε άμεσα τον περιορισμό 2 της §5.3.

6.2.3 Πλήρης Παρατηρησιμότητα (Metrics, Logs, Traces)

Κατανεμημένη ιχνηλάτηση. Ενσωμάτωση του OpenTelemetry SDK στις διαχειριζόμενες εφαρμογές για ιχνηλάτηση αιτημάτων από άκρο σε άκρο. Ο χρήστης εντοπίζει σε ποιο microservice εμφανίζεται το σημείο συμφόρησης. Η πλατφόρμα θα εγκαθιστά αυτόματα Jaeger ή Grafana Tempo μαζί με τον Prometheus κατά τη διάταξη, ως επιπλέον `helm_release` στα modules των Παραρτημάτων Γ και Δ.

Διαχείριση αρχείων καταγραφής. Εγκατάσταση του Grafana Loki για συγκέντρωση των logs όλων των pods. Το αποτέλεσμα είναι η πλήρης τριάδα παρατηρησιμότητας (metrics, logs, traces) σε ένα σύστημα: ο χρήστης συσχετίζει μια ανωμαλία μετρικής με συγκεκριμένη εγγραφή καταγραφής χωρίς εναλλαγή εργαλείου.

Αναφορές SLA. Αυτόματη παραγωγή αναφορών διαθεσιμότητας και απόδοσης ανά εφαρμογή, με υπολογισμό ποσοστού διαθεσιμότητας, καθυστέρησης P95/P99 και πλήθους ενεργειών κλιμάκωσης. Τα δεδομένα υπάρχουν ήδη στον πίνακα `scaling_events` (Παράρτημα Ε). Χρήσιμο για συμμόρφωση και για τιμολόγηση σε μοντέλο SaaS.

6.2.4 Ασφάλεια και Διακυβέρνηση

HashiCorp Vault. Αντικατάσταση της αποθήκευσης cloud credentials στη βάση δεδομένων με διαχείριση μέσω HashiCorp Vault. Τα διαπιστευτήρια εγχέονται δυναμικά κατά την εκτέλεση του Terraform και ανακαλούνται αυτόματα μετά το πέρας της. Η επέκταση αυτή αντιμετωπίζει τον σημαντικότερο τρέχοντα περιορισμό ασφάλειας της πλατφόρμας και σχετίζεται άμεσα με τη σημείωση ασφάλειας για την προσάρτηση του `kubeconfig` στην §5.1.2.

Καταγραφή ελέγχου. Πλήρης καταγραφή όλων των ενεργειών των χρηστών (σύνδεση, διάταξη, κλιμάκωση, διαγραφή) με χρονοσήμανση, αναγνωριστικό χρήστη, διεύθυνση IP και έκβαση, σε αμετάβλητο αποθετήριο καταγραφών. Απαραίτητο για συμμόρφωση με τον GDPR και για εγκληματολογική ανάλυση σε περίπτωση περιστατικού ασφάλειας.

Πολυ-μισθωτεία. Εισαγωγή επιπέδου *Organization* για ομαδοποίηση χρηστών και εφαρμογών, με ξεχωριστή τιμολόγηση, χρήστες και ρόλους ανά οργανισμό. Σε πρώτο επίπεδο πρόκειται για σχετικά απλή επέκταση — προσθήκη πεδίου *organization_id* στις υφιστάμενες οντότητες και αντίστοιχο φιλτράρισμα στο επίπεδο *repository* — ενώ η πλήρης έκδοση με ανεξάρτητη τιμολόγηση ανά οργανισμό είναι σημαντικά πολυπλοκότερη. Η επέκταση επιτρέπει τη μετατροπή της πλατφόρμας σε προϊόν SaaS, όπου διαφορετικοί οργανισμοί χρησιμοποιούν το ίδιο instance με πλήρη απομόνωση δεδομένων.

6.2.5 Βελτιώσεις DevOps για τις Διαχειριζόμενες Εφαρμογές

GitOps (ArgoCD / Flux). Ενσωμάτωση ροής εργασίας GitOps, όπου κάθε αλλαγή στις ρυθμίσεις μιας εφαρμογής πραγματοποιείται μέσω pull request στο Git. Το ArgoCD ή το Flux εντοπίζει αποκλίσεις μεταξύ της κατάστασης του Git και της κατάστασης της συστάδας και τις συγχρονίζει αυτόματα, βελτιώνοντας την ιχνηλασιμότητα και επιτρέποντας επαναφορά σε οποιαδήποτε προηγούμενη κατάσταση.

Επισημαίνεται ότι η επέκταση αυτή αφορά αποκλειστικά τη διαχείριση των ρυθμίσεων των **διαχειριζόμενων εφαρμογών** και όχι της ίδιας της πλατφόρμας. Απαιτεί σαφή σύμβαση ως προς το ποιο κανάλι υπερισχύει σε περίπτωση ταυτόχρονης τροποποίησης, ώστε να αποφευχθεί ασυνέπεια μεταξύ δύο παράλληλων πηγών αλήθειας (Git και web UI). Η πληρέστερη λύση είναι η πλατφόρμα να εγγράφει αυτόματα κάθε αλλαγή ρυθμίσεων που γίνεται μέσω του UI πίσω στο αποθετήριο Git (commit και push), καθιστώντας το Git τη μοναδική πηγή αλήθειας (single source of truth) — εξαλείφοντας έτσι το ενδεχόμενο ασυνέπειας αντί απλώς να το τεκμηριώνει ως περιορισμό.

Διατάξεις blue-green και canary. Υποστήριξη ανανεώσεων χωρίς διακοπή λειτουργίας, με διαμοιρασμό της κίνησης. Στη στρατηγική blue-green η νέα έκδοση διατάσσεται παράλληλα και η κίνηση μεταφέρεται αθρόα, ενώ στη στρατηγική canary ένα μικρό ποσοστό της κίνησης δρομολογείται στη νέα έκδοση πριν από την πλήρη μετάβαση.

Ανάλυση κόστους. Ενσωμάτωση με τα API των AWS Cost Explorer και Azure Cost Management για εμφάνιση του πραγματικού κόστους νέφους ανά εφαρμογή. Ο χρήστης βλέπει εκτιμώμενο και πραγματικό κόστος σε πραγματικό χρόνο, με δυνατότητα ορισμού ειδοποιήσεων προϋπολογισμού και προτάσεων βελτιστοποίησης. Η επέκταση αυτή καθιστά μετρήσιμο το όφελος της αμφίδρομης κλιμάκωσης που τεκμηριώνεται στην **Εικόνα 5.15**.

6.3 Επίλογος

Η παρούσα διπλωματική εργασία ανέπτυξε, υλοποίησε και επαλήθευσε μια QoS-aware, multi-cloud πλατφόρμα αυτόματης κλιμάκωσης και παρακολούθησης φόρτων εργασίας Kubernetes. Η πειραματική επίδειξη του Κεφαλαίου 5 τεκμηρίωσε ότι ο βασικός μηχανισμός λειτουργεί ορθά σε πραγματικό περιβάλλον Kubernetes: τόσο η αυτόματη κλιμάκωση βάσει κανόνων ποιότητας υπηρεσίας όσο και η ενοποιημένη παρακολούθηση μέσω Prometheus, Grafana και Alertmanager, με τον πλήρη κύκλο ζωής να εκτελείται μέσω της γραφικής διεπαφής της πλατφόρμας (§5.2.5).

Το κυριότερο επίτευγμα είναι η **ενοποίηση**: διάταξη βασισμένη σε IaC (Terraform), παρακολούθηση σε πραγματικό χρόνο (Prometheus και Grafana), αυτόματη κλιμάκωση με γνώση ποιότητας υπηρεσίας και φιλική διαδικτυακή διεπαφή (Spring Boot και React.js) σε ένα σύστημα, προσβάσιμο και σε χρήστες χωρίς εξειδίκευση σε τεχνολογίες νέφους. Η ανάλυση σχετικών εργασιών του Κεφαλαίου 3 επιβεβαίωσε ότι ο συγκεκριμένος συνδυασμός απουσίαζε από τη βιβλιογραφία.

Η επιλογή στοίβας ανοικτού κώδικα εξασφαλίζει αρχιτεκτονική ανεξάρτητη από πάροχο, μηδενικό κόστος αδειοδότησης και τεχνολογίες δοκιμασμένες σε περιβάλλον παραγωγής. Ο βαθμός ικανοποίησης απαιτήσεων (51,0% πλήρης, 27,7% μερική) αντανακλά ολοκληρωμένη υλοποίηση, ενώ οι περιορισμοί του πειραματικού περιβάλλοντος δηλώνονται ρητά στην §5.3 και οριοθετούν με ακρίβεια το εύρος των συμπερασμάτων.

Η πλατφόρμα διαθέτει τα θεμέλια για την εξέλιξή της σε προϊόν διαχείρισης νέφους έτοιμο για παραγωγή. Η προβλεπτική κλιμάκωση με μηχανική μάθηση, η πλήρης παρατηρησιμότητα και η υποστήριξη του GCP θα ολοκληρώσουν τη multi-cloud κάλυψη και θα καταστήσουν τη διαχείριση φόρτων εργασίας Kubernetes ουσιαστικά αυτόνομη.

Βιβλιογραφία

- Amazon Web Services. (2024a). Amazon EKS best practices guide.
<https://docs.aws.amazon.com/eks/latest/best-practices/introduction.html>
- Amazon Web Services. (2024b). Amazon EKS user guide.
<https://docs.aws.amazon.com/eks>
- Ben David, R., & Bremler-Barr, A. (2021). Kubernetes autoscaling: YoYo attack vulnerability and mitigation. In Proceedings of the 11th International Conference on Cloud Computing and Services Science (CLOSER 2021) (pp. 34–44). SciTePress. <https://doi.org/10.5220/0010397900340044>
- Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. *ACM Queue*, 14(1), 70–93.
<https://doi.org/10.1145/2898442.2898444>
- Cloud Native Computing Foundation. (2024). CNCF cloud native landscape.
<https://landscape.cncf.io>
- Cohn, M. (2009). *Succeeding with agile: Software development using Scrum*. Addison-Wesley.
- Crossplane Authors. (2024). Crossplane documentation. <https://docs.crossplane.io>
- Docker Inc. (2024). Docker documentation. <https://docs.docker.com>
- Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures [Doctoral dissertation, University of California, Irvine].
<https://ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- Fowler, M. (2010, March 18). Richardson maturity model: Steps toward the glory of REST. martinfowler.com.
<https://martinfowler.com/articles/richardsonMaturityModel.html>
- Fowler, M. (2012, May 1). Test pyramid. martinfowler.com.
<https://martinfowler.com/bliki/TestPyramid.html>
- Grafana Labs. (2024). Grafana documentation (Version 13).
<https://grafana.com/docs>
- HashiCorp. (2024). Terraform documentation (Version 1.6).
<https://developer.hashicorp.com/terraform/docs>
- Helm Authors. (2024). Helm documentation (Version 3). <https://helm.sh/docs>
- Huang, C.-K., & Pierre, G. (2023). AdapPF: Self-adaptive scrape interval for monitoring in geo-distributed cluster federations. In 2023 IEEE Symposium on Computers and Communications (ISCC) (pp. 417–423). Institute of Electrical and Electronics Engineers.
<https://doi.org/10.1109/ISCC58397.2023.10218080>
- Jones, M., Bradley, J., & Sakimura, N. (2015). JSON Web Token (JWT) (RFC 7519). Internet Engineering Task Force. <https://doi.org/10.17487/RFC7519>

- Kim, G., Humble, J., Debois, P., & Willis, J. (2016). *The DevOps handbook: How to create world-class agility, reliability, and security in technology organizations*. IT Revolution Press.
- Kubernetes Authors. (2024a). Horizontal Pod Autoscaling. <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/>
- Kubernetes Authors. (2024b). Kubernetes documentation. <https://kubernetes.io/docs>
- Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing* (NIST Special Publication 800-145). National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.SP.800-145>
- Microsoft. (2024a). Azure Kubernetes Service (AKS) documentation. <https://learn.microsoft.com/azure/aks>
- Microsoft. (2024b). Best practices for cluster operators and developers in Azure Kubernetes Service. <https://learn.microsoft.com/azure/aks/best-practices>
- Morris, K. (2020). *Infrastructure as code: Dynamic systems for the cloud age* (2nd ed.). O'Reilly Media.
- Prometheus Authors. (2024). Prometheus documentation. <https://prometheus.io/docs>
- Provos, N., & Mazières, D. (1999). A future-adaptable password scheme. *Proceedings of the 1999 USENIX Annual Technical Conference*, 81–91.
- Royce, W. W. (1970). Managing the development of large software systems. *Proceedings of IEEE WESCON*, 26, 1–9.
- Rzadca, K., Findeisen, P., Swiderski, J., Zych, P., Broniek, P., Kusmieriek, J., Nowak, P., Strack, B., Witusowski, P., Hand, S., & Wilkes, J. (2020). Autopilot: Workload autoscaling at Google. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)* (Article 16). Association for Computing Machinery. <https://doi.org/10.1145/3342195.3387524>
- Tomarchio, O., Calcaterra, D., Di Modica, G., & Mazzaglia, P. (2021). TORCH: A TOSCA-based orchestrator of multi-cloud containerised applications. *Journal of Grid Computing*, 19(1), Article 5. <https://doi.org/10.1007/s10723-021-09549-z>
- VMware Tanzu. (2024). Spring Boot reference documentation (Version 3.2). <https://docs.spring.io/spring-boot>
- Wiggins, A. (2011). The twelve-factor app. <https://12factor.net>

Παράρτημα A: Deployment YAML και Εντολές Αναφοράς

Θέση στο αποθετήριο. Τα αρχεία του παραρτήματος αυτού βρίσκονται στο αποθετήριο στη διαδρομή `k8s-samples/` (`deployment.yaml`, `hpa.yaml`).

A.1 Πλήρες `deployment.yaml` (Εφαρμογή Επίδειξης)

Το αρχείο αφορά τη `demo-webapp` — εφαρμογή τρίτου που διατάσσεται μέσω της πλατφόρμας (ως εγγραφές `APP_YAML_FILE` στη βάση δεδομένων, §4.3.5) και όχι την ίδια την πλατφόρμα. Στην πράξη, κάθε διαχειριζόμενη εφαρμογή παρέχει τα δικά της αρχεία YAML κατά τη δήλωσή της (FA-A01, Εικόνα 5.22), τα οποία η πλατφόρμα εφαρμόζει στη συστάδα-στόχο κατά τη διάταξη (FA-A04).

```
# deployment.yaml — Deployment και Service της εφαρμογής επίδειξης.
# Το αποτέλεσμα της εφαρμογής του τεκμηριώνεται στις Εικόνες 5.5–5.7 και 5.9.
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-webapp
  labels:
    app: demo-webapp
    managed-by: qos-multicloud-platform # Ετικέτα ιχνηλασιμότητας της
    πλατφόρμας
spec:
  replicas: 3
  revisionHistoryLimit: 3 # Περιορισμός παλαιών ReplicaSets
  selector:
    matchLabels:
      app: demo-webapp
  strategy:
    type: RollingUpdate # Ανανέωση χωρίς διακοπή
  λειτουργίας
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: demo-webapp
    spec:
      securityContext:
        runAsNonRoot: true # Best practice: μη προνομιούχος
        runAsUser: 101 # UID του χρήστη nginx στο image
      εκτέλεση
        alpine
        seccompProfile:
          type: RuntimeDefault
      containers:
        - name: webapp
          image: nginx:alpine
          imagePullPolicy: IfNotPresent
          ports:
            - name: http
              containerPort: 8080 # Θύρα > 1024, ώστε να μην
                απαιτούνται προνόμια root
      resources:
        # Τα requests αποτελούν τον παρονομαστή του HPA: το κατώφλι 50%
        # υπολογίζεται επί των 100m και όχι επί των limits (Εικόνα 5.14).
        requests:
          memory: "64Mi"
          cpu: "100m"
        limits:
          memory: "128Mi"
          cpu: "200m"
      securityContext:
```

```

        allowPrivilegeEscalation: false
        readOnlyRootFilesystem: false      # Το nginx εγγράφει σε
/var/cache/nginx
        capabilities:
          drop: ["ALL"]
        # Liveness: ανιχνεύει «κολλημένο» container και προκαλεί
επανεκκίνηση.
        livenessProbe:
          httpGet:
            path: /
            port: http
            initialDelaySeconds: 30
            periodSeconds: 10
            failureThreshold: 3
        # Readiness: καθορίζει αν το pod εντάσσεται στα endpoints του
Service.
        # Είναι το probe στο οποίο αντιστοιχεί η στήλη AVAILABLE της Εικόνας
5.6.
        readinessProbe:
          httpGet:
            path: /
            port: http
            initialDelaySeconds: 5
            periodSeconds: 5
            failureThreshold: 3
---
apiVersion: v1
kind: Service
metadata:
  name: demo-webapp-service
  labels:
    app: demo-webapp
spec:
  type: NodePort          # Τοπικό περιβάλλον· σε EKS/AKS χρησιμοποιείται
LoadBalancer.
  selector:
    app: demo-webapp
  ports:
    - name: http
      protocol: TCP
      port: 80             # Θύρα του Service εντός της συστάδας
      targetPort: http     # Αναφορά στη named port του container (8080)
      nodePort: 32187      # Εξωτερική θύρα κόμβου — βλ. Εικόνα 5.7

```

A.2 Διαμόρφωση HorizontalPodAutoscaler

Στην πράξη ο HPA δημιουργείται αυτόματα από το ScalingService κατά την αξιολόγηση ενεργού QoS κανόνα και όχι χειροκίνητα. Η ισοδύναμη επιτακτική εντολή παρατίθεται ενδεικτικά:

```
kubectl autoscale deployment demo-webapp --cpu-percent=50 --min=2 --max=10
```

Η δηλωτική μορφή, η οποία χρησιμοποιείται από την πλατφόρμα, είναι η ακόλουθη:

```

apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: demo-webapp
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: demo-webapp
  minReplicas: 2          # Υψηλή διαθεσιμότητα ακόμη και σε μηδενικό
φόρτο
  maxReplicas: 10         # Ανώτατο όριο κόστους
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:

```

```

    type: Utilization
    averageUtilization: 50 # Αντιστοιχεί στον QoS κανόνα cpu_usage >
50%
# Η ενότητα behavior ρυθμίζει ρητά τα παράθυρα σταθεροποίησης και εξηγεί
# την ταλάντωση 3 → 2 → 3 αντιγράφων που καταγράφεται στην Εικόνα 5.15.
behavior:
  scaleUp:
    stabilizationWindowSeconds: 60 # Ταχεία αντίδραση σε αύξηση φόρτου
    policies:
      - type: Pods
        value: 2
        periodSeconds: 60
    selectPolicy: Max
  scaleDown:
    stabilizationWindowSeconds: 300 # Συντηρητική μείωση: αποφυγή
ταλαντώσεων
    policies:
      - type: Pods
        value: 1
        periodSeconds: 120
    selectPolicy: Min

```

Προαπαιτούμενο. Ο HPA απαιτεί λειτουργικό `metrics-server` στο Minikube ενεργοποιείται με `minikube addons enable metrics-server` (βλ. `setup_cluster.sh`, §5.2.1).

A.3 Ερωτήματα PromQL Αναφοράς

Τα ερωτήματα του **Πίνακα A.1** χρησιμοποιούνται από το `PrometheusService` για την τροφοδότηση του `MonitoringService` κατά την περιοδική αξιολόγηση των κανόνων.

Πίνακας A.1 — *Ερωτήματα PromQL Αναφοράς*

Ερώτημα	Περιγραφή	Τεκμηρίω ση
<code>up</code>	Κατάσταση διαθεσιμότητας όλων των παρακολουθούμενων endpoints	Εικ. 5.17
<code>kube_pod_status_phase{phase="Running"}</code>	Εκτελούμενα pods σε όλα τα namespaces	Εικ. 5.18
<code>sum(rate(container_cpu_usage_seconds_total{container!=""}[5m])) by (pod) * 100</code>	Ποσοστιαία χρήση CPU ανά pod	Εικ. 5.13, 5.16
<code>sum(container_memory_working_set_bytes{container!=""}) by (pod) / 1024 / 1024</code>	Ενεργή χρήση μνήμης (MiB) ανά pod	Εικ. 5.12
<code>kube_deployment_status_replicas{deployment="demo-webapp"}</code>	Επιθυμητά αντίγραφα ανά deployment	Εικ. 5.6
<code>kube_horizontalpodautoscaler_status_current_replicas</code>	Τρέχοντα αντίγραφα κατά τον HPA	Εικ. 5.14, 5.15
<code>kube_node_status_condition{condition="Ready", status="true"}</code>	Κατάσταση ετοιμότητας κόμβων	Εικ. 5.3

A.4 Εντολές Αποδιάταξης και Καθαρισμού

Οι εντολές εκτελούνται αυτόματα από το TerraformService κατά την αποδιάταξη (FA-A05) και παρατίθενται ενδεικτικά. Η πλατφόρμα υποστηρίζει εφαρμογές με πολλαπλά αρχεία YAML — μέσω της οντότητας APP_YAML_FILE και του πεδίου order_index — εφαρμόζοντάς τα κατά αύξουσα και διαγράφοντάς τα κατά φθίνουσα σειρά, ώστε να τηρούνται οι εξαρτήσεις μεταξύ αντικειμένων.

```
# 1. Διαγραφή των αντικειμένων της διαχειριζόμενης εφαρμογής.
kubectl delete -f deployment.yaml --ignore-not-found
kubectl delete hpa demo-webapp --ignore-not-found

# 2. Απεγκατάσταση της στοίβας παρακολούθησης.
helm uninstall prometheus -n monitoring
kubectl delete namespace monitoring --ignore-not-found

# 3. Τερματισμός ή πλήρης διαγραφή της τοπικής συστάδας.
minikube stop      # Πάυση — τα δεδομένα διατηρούνται
minikube delete    # Πλήρης διαγραφή της συστάδας και των δεδομένων

# 4. Σε παραγωγικό περιβάλλον νέφους (Παραρτήματα Γ και Δ):
#   το TerraformService εκτελεί το ισοδύναμο της παρακάτω εντολής
#   στο workspace της εφαρμογής.
terraform destroy -auto-approve -var-file="app-${APP_ID}.tfvars"
```

Παράρτημα Β: Παράδειγμα Απόκρισης REST API (HATEOAS)

Θέση στο αποθετήριο. Το παράδειγμα αφορά τις απαντήσεις του ApplicationController

(backend/src/main/java/gr/aegean/cloudplatform/controller/).

Το παράρτημα διατηρείται διότι αποτυπώνει τη σχεδιαστική πρόθεση για το επίπεδο 3 του μοντέλου ωριμότητας και χρησιμεύει ως προδιαγραφή της επέκτασης· η τρέχουσα υλοποίηση παραμένει στο επίπεδο 2 (MFA-A02, Πίνακας 4.20).

Προτεινόμενη μορφή απόκρισης του GET /api/v1/applications/{id} σε επίπεδο 3 του μοντέλου ωριμότητας Richardson. Το παράδειγμα δεν αντιστοιχεί στην παρούσα υλοποίηση, η οποία παραμένει στο επίπεδο 2 (§4.7.4)· παρατίθεται ως προδιαγραφή της προτεινόμενης επέκτασης:

```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "title": "E-Commerce API",
  "status": "DEPLOYED",
  "deployments": [
    { "cloudProvider": "AWS", "nodeCount": 5, "status": "ACTIVE" }
  ],
  "qosRules": [
    {
      "metricName": "cpu_usage",
      "operator": "GT",
      "thresholdValue": 80.0,
      "durationSeconds": 120,
      "cooldownSeconds": 300,
      "actionType": "SCALE_PODS_UP",
      "actionValue": 2.0,
      "isActive": true
    }
  ],
  "_links": {
    "self": { "href": "/api/v1/applications/550e8400-e29b-41d4-a716-446655440000" },
    "deploy": { "href": "/api/v1/applications/550e8400-e29b-41d4-a716-446655440000/deployment", "method": "POST" },
    "undeploy": { "href": "/api/v1/applications/550e8400-e29b-41d4-a716-446655440000/deployment", "method": "DELETE" },
    "scale": { "href": "/api/v1/applications/550e8400-e29b-41d4-a716-446655440000/deployment", "method": "PATCH" },
    "metrics": { "href": "/api/v1/applications/550e8400-e29b-41d4-a716-446655440000/metrics" },
    "events": { "href": "/api/v1/applications/550e8400-e29b-41d4-a716-446655440000/scaling-events" }
  }
}
```

Το πεδίο `durationSeconds` καθορίζει για πόσο χρόνο πρέπει η μετρική να παραμείνει πάνω από το κατώφλι πριν ενεργοποιηθεί η ενέργεια κλιμάκωσης, αποτρέποντας ψευδώς θετικές ενεργοποιήσεις από στιγμιαίες αιχμές· το `cooldownSeconds` ορίζει το ελάχιστο διάστημα μεταξύ διαδοχικών ενεργειών για τον ίδιο κανόνα. Και τα δύο αντιστοιχούν εννοιολογικά στα παράθυρα σταθεροποίησης του HPA (Παράρτημα Α.2) και εξηγούν τη χρονική υστέρηση που παρατηρείται στην **Εικόνα 5.15**.

Η πλατφόρμα υποστηρίζει τις προκαθορισμένες μετρικές `cpu_usage`, `memory_usage`, `response_time` και `error_rate`, καθώς και `custom` μετρικές, εφόσον αυτές

εκτίθενται από την εφαρμογή σε μορφή συμβατή με το Prometheus exposition format και είναι προσβάσιμες από τη στοίβα παρακολούθησης (Prometheus Authors, 2024).

Παράρτημα Γ: Terraform Module — AWS EKS

Θέση στο αποθετήριο. Το module παράγεται δυναμικά από το TerraformTemplateGenerator (backend/src/main/java/gr/aegean/cloudplatform/terraform/).

Χαρακτήρας του παραρτήματος. Ο κώδικας που ακολουθεί αποτελεί την πλήρη προδιαγραφή του module, δηλαδή τον στόχο σχεδίασης προς τον οποίο κατευθύνεται η υλοποίηση, και όχι αποτύπωση του κώδικα που παράγει η τρέχουσα έκδοση. Ο TerraformTemplateGenerator παράγει στην παρούσα φάση τον πυρήνα της δήλωσης της συστάδας· τα υπόλοιπα τμήματα – και ιδίως ο πόρος helm_release για την εγκατάσταση της στοίβας kube-prometheus-stack – δεν περιλαμβάνονται ακόμη στη γεννήτρια προτύπων και αποτελούν μέρος των προτεινόμενων επεκτάσεων (§6.2). Η διάκριση δηλώνεται ρητά ώστε ο αναγνώστης να μη συγχέει την προδιαγραφή με το υλοποιημένο τμήμα.

```
# — versions.tf —————
# Καθήλωση εκδόσεων: εξασφαλίζει αναπαραγωγιμότητα των διατάξεων.
terraform {
  required_version = ">= 1.6.0"
  required_providers {
    aws = { source = "hashicorp/aws", version = "~> 5.40" }
    helm = { source = "hashicorp/helm", version = "~> 2.12" }
  }
}

# — variables.tf —————
variable "cluster_name" { type = string }
variable "app_id"       { type = string }

variable "aws_region" {
  type    = string
  default = "eu-west-1"
}

variable "instance_type" {
  type    = string
  default = "t3.medium"
}

variable "min_nodes" {
  type    = number
  default = 2
}

variable "max_nodes" {
  type    = number
  default = 10
}

variable "desired_nodes" {
  type    = number
  default = 3
}

variable "grafana_admin_password" {
  type          = string
  sensitive     = true    # Δεν εμφανίζεται σε logs, plan output ή state diff
}

# — main.tf —————
provider "aws" {
  region = var.aws_region
}

# Δίκτυο: ιδιωτικά υποδίκτυα για τους κόμβους, δημόσια για τους load balancers.
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"
```

```

version = "~> 5.5"

name = "${var.cluster_name}-vpc"
cidr = "10.0.0.0/16"

azs      = ["${var.aws_region}a", "${var.aws_region}b",
"${var.aws_region}c"]
private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
public_subnets  = ["10.0.101.0/24", "10.0.102.0/24", "10.0.103.0/24"]

enable_nat_gateway = true
single_nat_gateway = true    # Μείωση κόστους· σε παραγωγή: ένα NAT GW ανά
AZ.
enable_dns_hostnames = true

# Ετικέτες απαιτούμενες από τον AWS Load Balancer Controller για την
# αυτόματη ανακάλυψη των υποδικτύων.
public_subnet_tags = { "kubernetes.io/role/elb" = 1 }
private_subnet_tags = { "kubernetes.io/role/internal-elb" = 1 }
}

module "eks" {
  source = "terraform-aws-modules/eks/aws"
  version = "~> 20.8"

  cluster_name      = var.cluster_name
  cluster_version   = "1.29"

  vpc_id            = module.vpc.vpc_id
  subnet_ids        = module.vpc.private_subnets

  # Το IRSA επιτρέπει την ανάθεση IAM ρόλων σε Kubernetes ServiceAccounts,
  # αποφεύγοντας τη χρήση στατικών διαπιστευτηρίων εντός της συστάδας.
  enable_irsa = true

  cluster_endpoint_public_access = true

  eks_managed_node_groups = {
    main = {
      min_size      = var.min_nodes
      max_size      = var.max_nodes
      desired_size  = var.desired_nodes
      instance_types = [var.instance_type]
      capacity_type = "ON_DEMAND"
    }
  }

  tags = {
    Application = var.app_id
    ManagedBy   = "CloudPlatform-Terraform"
  }
}

# — Πύθμιση του Helm provider με βάση την παραχθείσα συστάδα —
data "aws_eks_cluster_auth" "this" {
  name = module.eks.cluster_name
}

provider "helm" {
  kubernetes {
    host                = module.eks.cluster_endpoint
    cluster_ca_certificate =
base64decode(module.eks.cluster_certificate_authority_data)
    token              = data.aws_eks_cluster_auth.this.token
  }
}

```

```

# — Στοιβά παρακολούθησης —————
resource "helm_release" "monitoring" {
  depends_on = [module.eks]

  name           = "prometheus"
  repository     = "https://prometheus-community.github.io/helm-charts"
  chart          = "kube-prometheus-stack"
  version        = "58.1.0"
  namespace     = "monitoring"
  create_namespace = true

  set_sensitive {
    name = "grafana.adminPassword"
    value = var.grafana_admin_password
  }

  set {
    name = "prometheus.prometheusSpec.serviceMonitorSelectorNilUsesHelmValues"
    value = "false"
  }
}

# — AWS Load Balancer Controller (Ingress / Service type=LoadBalancer) —
resource "helm_release" "aws_lb_controller" {
  depends_on = [module.eks]

  name           = "aws-load-balancer-controller"
  repository     = "https://aws.github.io/eks-charts"
  chart          = "aws-load-balancer-controller"
  namespace     = "kube-system"

  set {
    name = "clusterName"
    value = module.eks.cluster_name
  }
}

# — EBS CSI Driver (PersistentVolumeClaims μέσω AWS EBS) —————
resource "aws_eks_addon" "ebs_csi" {
  cluster_name = module.eks.cluster_name
  addon_name   = "aws-ebs-csi-driver"
}

# — outputs.tf —————
output "cluster_endpoint" {
  value = module.eks.cluster_endpoint
}

output "cluster_name" {
  value = module.eks.cluster_name
}

```

Παράρτημα Δ: Terraform Module — Azure AKS

Θέση στο αποθετήριο. Το module παράγεται δυναμικά από το TerraformTemplateGenerator

(backend/src/main/java/gr/aegean/cloudplatform/terraform/).

Χαρακτήρας του παραρτήματος. Ο κώδικας που ακολουθεί αποτελεί την πλήρη προδιαγραφή του module, δηλαδή τον στόχο σχεδίασης προς τον οποίο κατευθύνεται η υλοποίηση, και όχι αποτύπωση του κώδικα που παράγει η τρέχουσα έκδοση. Ο TerraformTemplateGenerator παράγει στην παρούσα φάση τον πυρήνα της δήλωσης της συστάδας· τα υπόλοιπα τμήματα — και ιδίως ο πόρος helm_release για την εγκατάσταση της στοίβας kube-prometheus-stack — δεν περιλαμβάνονται ακόμη στη γεννήτρια προτύπων και αποτελούν μέρος των προτεινόμενων επεκτάσεων (§6.2). Η διάκριση δηλώνεται ρητά ώστε ο αναγνώστης να μη συγχέει την προδιαγραφή με το υλοποιημένο τμήμα.

Το αντίστοιχο module για το Azure Kubernetes Service, βασισμένο στον πάροχο Azure Resource Manager (azurerm):

```
# — versions.tf —————
terraform {
  required_version = ">= 1.6.0"
  required_providers {
    azurerm = { source = "hashicorp/azurerm", version = "~> 4.0" }
  }
}

# — variables.tf —————
# Σημείωση: στο HCL οι δηλώσεις εντός μπλοκ διαχωρίζονται με αλλαγή γραμμής
# και ΟΧΙ με κόμμα — σφάλμα που υπήρχε στην αρχική έκδοση του παραρτήματος.
variable "cluster_name" { type = string }
variable "app_id"       { type = string }

variable "azure_client_id" {
  type      = string
  sensitive = true
}

variable "azure_client_secret" {
  type      = string
  sensitive = true
}

variable "azure_tenant_id" {
  type      = string
  sensitive = true
}

variable "azure_subscription_id" {
  type      = string
  sensitive = true
}

variable "vm_size" {
  type      = string
  default = "Standard_DS2_v2"
}

variable "min_nodes" {
  type      = number
  default = 2
}
```

```

variable "max_nodes" {
  type    = number
  default = 10
}

variable "desired_nodes" {
  type    = number
  default = 3
}

variable "k8s_version" {
  type    = string
  default = "1.29"
}

variable "location" {
  type    = string
  default = "westeurope"
}

# — main.tf —————
provider "azurerm" {
  features {}

  client_id       = var.azure_client_id
  client_secret   = var.azure_client_secret
  tenant_id       = var.azure_tenant_id
  subscription_id = var.azure_subscription_id
}

resource "azurerm_resource_group" "main" {
  name     = "${var.cluster_name}-rg"
  location = var.location

  tags = {
    Application = var.app_id
    ManagedBy   = "CloudPlatform-Terraform"
  }
}

resource "azurerm_kubernetes_cluster" "main" {
  name                 = var.cluster_name
  location             = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name
  dns_prefix           = var.cluster_name
  kubernetes_version   = var.k8s_version

  default_node_pool {
    name     = "default"
    vm_size  = var.vm_size
    node_count = var.desired_nodes

    # Στην έκδοση 4.x του παρόχου azurerm το πεδίο enable_auto_scaling
    # μετονομάστηκε σε auto_scaling_enabled.
    auto_scaling_enabled = true
    min_count            = var.min_nodes
    max_count            = var.max_nodes

    os_disk_size_gb = 50
  }

  # Ταυτότητα διαχειριζόμενη από το Azure: αποφεύγεται η αποθήκευση
  # διαπιστευτηρίων εντός της συστάδας.
  identity {
    type = "SystemAssigned"
  }
}

```

```

}

network_profile {
  network_plugin = "azure"
  load_balancer_sku = "standard"
}

# Ενεργοποίηση RBAC σε επίπεδο συστάδας (απαίτηση MFA-S02).
role_based_access_control_enabled = true

tags = {
  Application = var.app_id
  ManagedBy   = "CloudPlatform-Terraform"
  Environment = "production"
}
}

# — outputs.tf —————
output "cluster_endpoint" {
  value     = azurerm_kubernetes_cluster.main.kube_config[0].host
  sensitive = true
}

output "kube_config" {
  value     = azurerm_kubernetes_cluster.main.kube_config_raw
  sensitive = true
}

output "client_certificate" {
  value     = azurerm_kubernetes_cluster.main.kube_config[0].client_certificate
  sensitive = true
}

```

Παράρτημα Ε: Flyway Database Migrations

Θέση στο αποθετήριο. Τα αρχεία μετάπτωσης βρίσκονται στη διαδρομή `backend/src/main/resources/db/migration/` και εκτελούνται αυτόματα από το Flyway κατά την εκκίνηση.

Τα scripts μετάπτωσης του Flyway δημιουργούν το σχήμα της βάσης. Ονομάζονται κατά τη σύμβαση `V{version}__{description}.sql` και εκτελούνται αυτόματα κατά την εκκίνηση του Spring Boot (§5.1.2, `spring.flyway.enabled=true`).

V1__create_users_table.sql

```
CREATE TABLE users (
    id          VARCHAR(255) PRIMARY KEY,
    username    VARCHAR(50)  UNIQUE NOT NULL,
    full_name   VARCHAR(100) NOT NULL,
    email       VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,          -- BCrypt (cost factor
12)
    role        VARCHAR(20)  NOT NULL DEFAULT 'REGULAR',
    is_active    BOOLEAN     NOT NULL DEFAULT FALSE,
    security_question TEXT,
    security_answer_hash VARCHAR(255),
    created_at   TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at   TIMESTAMP,
    -- Οι περιορισμοί CHECK επιβάλλουν τα enum σε επίπεδο βάσης, ώστε η
    -- ακεραιότητα να μην εξαρτάται αποκλειστικά από το επίπεδο εφαρμογής.
    CONSTRAINT chk_users_role CHECK (role IN ('REGULAR', 'ADMIN'))
);
```

V2__create_applications_tables.sql

```
CREATE TABLE applications (
    id          VARCHAR(255) PRIMARY KEY,
    title       VARCHAR(255) NOT NULL,
    description TEXT,
    version     VARCHAR(50),
    status      VARCHAR(30)  NOT NULL DEFAULT 'INITIAL',
    owner_id    VARCHAR(255) NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    created_at   TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at   TIMESTAMP,
    CONSTRAINT chk_app_status CHECK (status IN
    ('INITIAL', 'DEPLOYING', 'DEPLOYED', 'UNDEPLOYING', 'TERMINATED',
    'FAILED'))
);

CREATE TABLE application_tags (
    application_id VARCHAR(255) NOT NULL REFERENCES applications(id) ON DELETE
CASCADE,
    tags           VARCHAR(50)  NOT NULL,
    PRIMARY KEY (application_id, tags)
);

CREATE TABLE cloud_configs (
    id          VARCHAR(255) PRIMARY KEY,
    app_id      VARCHAR(255) NOT NULL REFERENCES applications(id) ON DELETE
CASCADE,
    cloud_provider VARCHAR(20) NOT NULL,
    instance_type  VARCHAR(50) NOT NULL,
    min_nodes      INTEGER     NOT NULL DEFAULT 1,
    max_nodes      INTEGER     NOT NULL DEFAULT 10,
    desired_nodes  INTEGER     NOT NULL DEFAULT 2,
    region         VARCHAR(50),
    port_mappings  JSONB,
    CONSTRAINT chk_cloud_provider CHECK (cloud_provider IN ('AWS', 'AZURE')),
    -- Διασφάλιση λογικής συνέπειας των ορίων κλιμάκωσης.
    CONSTRAINT chk_node_bounds CHECK (min_nodes <= desired_nodes
);
```

```

AND desired_nodes <= max_nodes)
);

CREATE TABLE qos_rules (
    id          VARCHAR(255) PRIMARY KEY,
    app_id      VARCHAR(255) NOT NULL REFERENCES applications(id) ON
DELETE CASCADE,
    metric_name VARCHAR(100) NOT NULL,
    operator    VARCHAR(5) NOT NULL,
    threshold_value DOUBLE PRECISION NOT NULL,
    duration_seconds INTEGER DEFAULT 0,
    action_type VARCHAR(30) NOT NULL,
    action_value DOUBLE PRECISION,
    action_target VARCHAR(20) DEFAULT 'all',
    cooldown_seconds INTEGER DEFAULT 300,
    is_active   BOOLEAN NOT NULL DEFAULT TRUE,
    CONSTRAINT chk_qos_operator CHECK (operator IN ('GT', 'GTE', 'LT', 'LTE',
'EQ')),
    CONSTRAINT chk_qos_action CHECK (action_type IN
('SCALE_PODS_UP', 'SCALE_PODS_DOWN', 'SCALE_NODES_UP',
'SCALE_NODES_DOWN', 'NOTIFY'))
);

CREATE TABLE deployments (
    id          VARCHAR(255) PRIMARY KEY,
    app_id      VARCHAR(255) NOT NULL REFERENCES applications(id),
    cloud_provider VARCHAR(20) NOT NULL,
    cluster_endpoint VARCHAR(255),
    prometheus_endpoint VARCHAR(255),
    node_count   INTEGER,
    status       VARCHAR(30) NOT NULL DEFAULT 'PROVISIONING',
    terraform_workspace VARCHAR(500),
    deployed_at  TIMESTAMP,
    terminated_at TIMESTAMP,
    CONSTRAINT chk_deployment_status CHECK (status IN
('PROVISIONING', 'ACTIVE', 'FAILED', 'TERMINATED'))
);

-- Τα ScalingEvents καταγράφουν κάθε ενέργεια κλιμάκωσης για μεταγενέστερη
-- ανάλυση.
CREATE TABLE scaling_events (
    id          VARCHAR(255) PRIMARY KEY,
    app_id      VARCHAR(255) NOT NULL REFERENCES applications(id),
    qos_rule_id VARCHAR(255) REFERENCES qos_rules(id) ON DELETE SET NULL,
    triggered_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    metric_value DOUBLE PRECISION,
    action_type VARCHAR(30) NOT NULL,
    action_status VARCHAR(20) NOT NULL DEFAULT 'IN_PROGRESS',
    error_message TEXT,
    CONSTRAINT chk_event_status CHECK (action_status IN
('IN_PROGRESS', 'SUCCESS', 'FAILED'))
);

CREATE TABLE notifications (
    id          VARCHAR(255) PRIMARY KEY,
    user_id     VARCHAR(255) NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    app_id      VARCHAR(255) REFERENCES applications(id) ON DELETE SET NULL,
    type        VARCHAR(50) NOT NULL,
    title       VARCHAR(255) NOT NULL,
    message     TEXT NOT NULL,
    is_read     BOOLEAN NOT NULL DEFAULT FALSE,
    created_at  TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

```

V3__create_indexes.sql

Η PostgreSQL δεν δημιουργεί αυτόματα ευρετήρια για τα ξένα κλειδιά· δημιουργούνται επομένως ρητά, μαζί με ευρετήρια που εξυπηρετούν τα συχνότερα ερωτήματα της εφαρμογής.

```
CREATE INDEX idx_applications_owner      ON applications(owner_id);
CREATE INDEX idx_applications_status     ON applications(status);
CREATE INDEX idx_qos_rules_app           ON qos_rules(app_id) WHERE is_active =
TRUE;
CREATE INDEX idx_deployments_app         ON deployments(app_id);
CREATE INDEX idx_deployments_status      ON deployments(status);
CREATE INDEX idx_scaling_events_app      ON scaling_events(app_id);
CREATE INDEX idx_scaling_events_time     ON scaling_events(triggered_at DESC);
CREATE INDEX idx_notifications_user      ON notifications(user_id, is_read);
```

Παράρτημα ΣΤ: Παραδείγματα Κώδικα Δοκιμών και CI/CD Pipeline

Θέση στο αποθετήριο. Οι δοκιμές βρίσκονται στη διαδρομή `backend/src/test/java/gr/aegean/cloudplatform/service/` και ο αγωγός στο `.github/workflows/ci.yml`.

ΣΤ.1 Στρατηγική Δοκιμών

Η αναλυτική περιγραφή της στρατηγικής δοκιμών (πυραμίδα δοκιμών, πίνακας ανά επίπεδο) παρουσιάζεται στην §4.6.1 (**Πίνακας 4.9**), ως μέρος του τετάρτου σταδίου του μοντέλου Waterfall. Ακολουθούν ενδεικτικά παραδείγματα κώδικα.

Δοκιμή μονάδας — MonitoringService

```
@ExtendWith(MockitoExtension.class)
class MonitoringServiceTest {

    @Mock private PrometheusService prometheusService;
    @Mock private ScalingService scalingService;
    @Mock private ScalingEventRepository eventRepository;

    @InjectMocks private MonitoringService monitoringService;

    @Test
    @DisplayName("Παραβίαση QoS κανόνα οδηγεί σε εκτέλεση scale-up")
    void givenCpuAboveThreshold_whenEvaluate_thenScaleUp() {
        // Given — η μετρική υπερβαίνει το κατώφλι του κανόνα.
        Application app = createTestApp();
        QosRule rule = createRule("cpu_usage", Operator.GT, 80.0,
                                   ActionType.SCALE_PODS_UP, 2.0);
        when(prometheusService.queryMetric(app.getId(), "cpu_usage"))
            .thenReturn(85.5); // 85,5% > κατώφλι 80%
        when(scalingService.scalePodsUp(any(), any())).thenReturn(true);

        // When
        monitoringService.evaluateApplication(app);

        // Then — μία ενέργεια κλιμάκωσης και δύο εγγραφές συμβάντος.
        verify(scalingService, times(1)).scalePodsUp(eq(app), eq(rule));
        verify(eventRepository, times(2)).save(any()); // IN_PROGRESS και
SUCCESS
    }

    @Test
    @DisplayName("Μετρική κάτω από το κατώφλι δεν προκαλεί καμία ενέργεια")
    void givenCpuBelowThreshold_whenEvaluate_thenNoAction() {
        Application app = createTestApp();
        when(prometheusService.queryMetric(any(), any())).thenReturn(45.0);

        monitoringService.evaluateApplication(app);

        verifyNoInteractions(scalingService);
        verifyNoInteractions(eventRepository);
    }
}
```

Προτεινόμενη δοκιμή ολοκλήρωσης — ApplicationController (δεν περιλαμβάνεται στην παρούσα έκδοση)

Το ακόλουθο παράδειγμα δεν αποτελεί μέρος της υφιστάμενης σουίτας δοκιμών· παρατίθεται ως προδιαγραφή της δοκιμής ολοκλήρωσης που πρέπει να προστεθεί, σύμφωνα με τη σημείωση της §4.6.1.

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@Testcontainers
class ApplicationControllerIT {

    // Πραγματική PostgreSQL σε container: οι δοκιμές εκτελούνται έναντι της
    // ίδιας έκδοσης βάσης με το περιβάλλον παραγωγής.
    @Container
    static PostgreSQLContainer<?> postgres =
        new PostgreSQLContainer<>("postgres:15-alpine");

    @DynamicPropertySource
    static void datasourceProperties(DynamicPropertyRegistry registry) {
        registry.add("spring.datasource.url", postgres::getJdbcUrl);
        registry.add("spring.datasource.username", postgres::getUsername);
        registry.add("spring.datasource.password", postgres::getPassword);
    }

    @Autowired private TestRestTemplate restTemplate;
    @Autowired private ApplicationRepository applicationRepository;

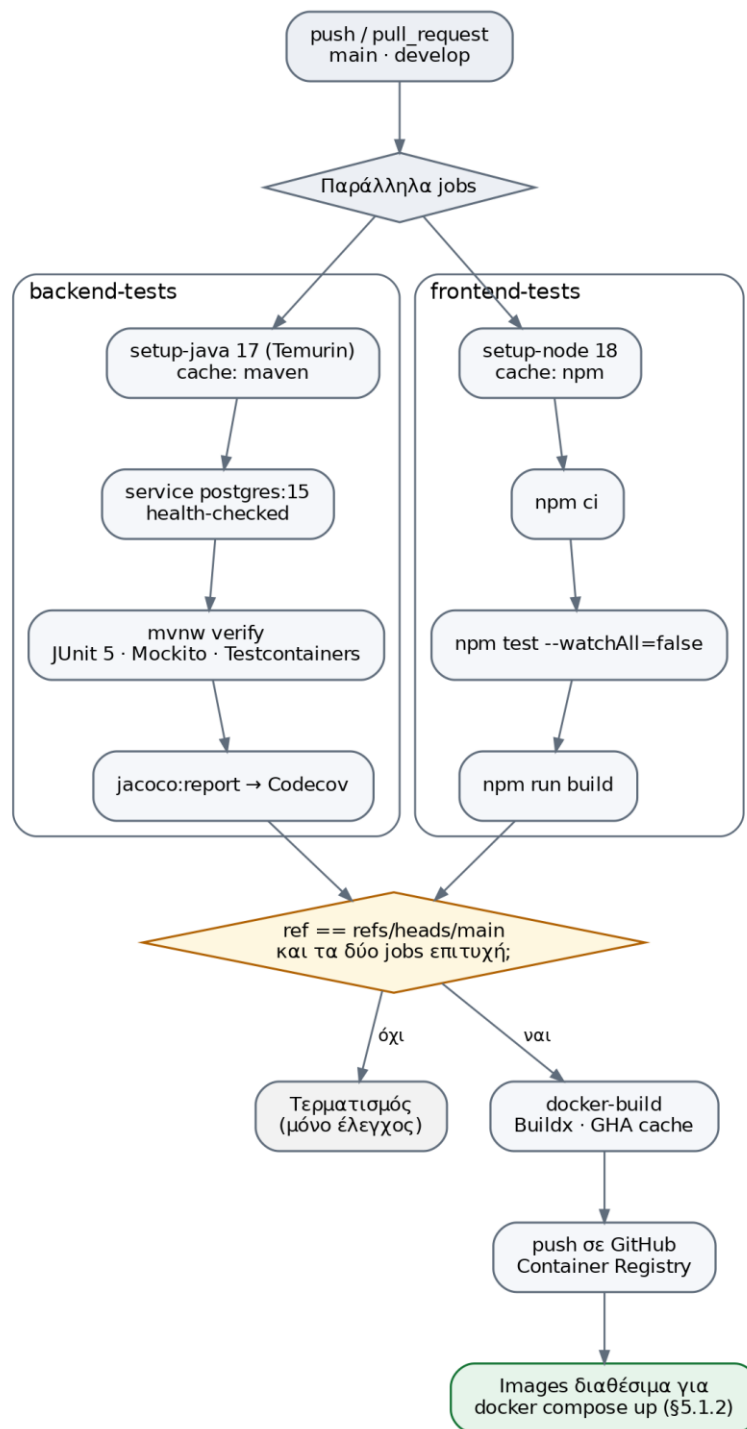
    @Test
    @DisplayName("POST /applications δημιουργεί εφαρμογή και επιστρέφει 201 Created")
    void whenCreateApplication_thenReturns201AndPersists() {
        ApplicationCreatedDTO dto = new ApplicationCreatedDTO("Test App", "desc",
            "1.0");

        ResponseEntity<ApplicationDTO> response = restTemplate.postForEntity(
            "/api/v1/applications", dto, ApplicationDTO.class);

        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
        assertThat(applicationRepository.findById(response.getBody().id())).isPresent();
    }
}
```

ΣΤ.2 CI/CD Pipeline — GitHub Actions

Η ροή του αγωγού συνεχούς ενσωμάτωσης και παράδοσης απεικονίζεται στην **Εικόνα ΣΤ.1**.



Εικόνα ΣΤ.1 — Ροή του Αγωγού Συνεχούς Ενσωμάτωσης και Παράδοσης (GitHub Actions)

Το αντίστοιχο αρχείο `.github/workflows/ci.yml`:

```

name: CI/CD Pipeline

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

# Ελάχιστα δικαιώματα εξ ορισμού (principle of least privilege).
permissions:
  contents: read
  
```

```

jobs:

  backend-tests:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:15
        env:
          POSTGRES_DB: cloudplatform_test
          POSTGRES_USER: test
          POSTGRES_PASSWORD: test
        ports: [ '5432:5432' ]
        # Χωρίς health check το job ενδέχεται να εκκινήσει πριν η βάση
        # δεχθεί συνδέσεις – συχνή αιτία ασταθών (flaky) δοκιμών.
        options: >-
          --health-cmd="pg_isready -U test"
          --health-interval=10s
          --health-timeout=5s
          --health-retries=5
    steps:
      - uses: actions/checkout@v4

      - uses: actions/setup-java@v4
        with:
          java-version: '17'
          distribution: 'temurin'
          cache: 'maven' # Προσωρινή αποθήκευση εξαρτήσεων Maven

      - name: Run backend tests
        run: |
          chmod +x mvnw
          ./mvnw --batch-mode verify jacoco:report
        env:
          SPRING_DATASOURCE_URL:
jdbc:postgresql://localhost:5432/cloudplatform_test
          SPRING_DATASOURCE_USERNAME: test
          SPRING_DATASOURCE_PASSWORD: test

      - name: Upload coverage report
        uses: codecov/codecov-action@v4
        with:
          token: ${ secrets.CODECOV_TOKEN }
          files: ./backend/target/site/jacoco/jacoco.xml

  frontend-tests:
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: ./frontend
    steps:
      - uses: actions/checkout@v4

      - uses: actions/setup-node@v4
        with:
          node-version: '18'
          cache: 'npm'
          cache-dependency-path: frontend/package-lock.json

      - run: npm ci
      - run: npm test -- --watchAll=false
      - run: npm run build

  docker-build:
    needs: [ backend-tests, frontend-tests ]
    if: github.ref == 'refs/heads/main'
    runs-on: ubuntu-latest
    permissions:

```

```

  contents: read
  packages: write          # Απαιτείται για δημοσίευση στο GHCR
steps:
  - uses: actions/checkout@v4

  - uses: docker/setup-buildx-action@v3

  - uses: docker/login-action@v3
    with:
      registry: ghcr.io
      username: ${GITHUB_ACTOR}
      password: ${GITHUB_TOKEN}

  - name: Build and push backend image
    uses: docker/build-push-action@v5
    with:
      context: ./backend
      push: true
      tags: ghcr.io/${GITHUB_REPOSITORY}/backend:${GITHUB_SHA}
      cache-from: type=gha
      cache-to: type=gha,mode=max

  - name: Build and push frontend image
    uses: docker/build-push-action@v5
    with:
      context: ./frontend
      push: true
      tags: ghcr.io/${GITHUB_REPOSITORY}/frontend:${GITHUB_SHA}
      cache-from: type=gha
      cache-to: type=gha,mode=max

```