ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΑΣΦΑΛΕΙΑ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Επισκόπηση του Τοπίου των Εργαλείων Kubernetes
(Survey of the Kubernetes Tools Landscape)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Μπακαλάκου Αθανασίου

Επιβλέπων: Αν. Καθηγητής Κρητικός Κυριάκος

Μέλη εξεταστικής επιτροπής: Καθηγητής Κοκολάκης Σπυρίδων, Αν. Καθηγητής Σκούτας Δημήτριος

Περιεχόμενα

Λίστα Πινάκων

Λίστα Εικόνων

Περίληψη

Η διπλωματική εργασία αυτή επικεντρώνεται σε μια ενδελεχή επισκόπηση του οικοσυστήματος εργαλείων Kubernetes, με στόχο την κατηγοριοποίηση, ανάλυση και συγκριτική αξιολόγησή τους. Τα εργαλεία που μελετώνται προέρχονται τόσο από την ερευνητική βιβλιογραφία όσο και από την αγορά, και ταξινομούνται ιεραρχικά βάσει ενός συνόλου διαστάσεων που αφορούν τη λειτουργικότητα και τον τρόπο ενσωμάτωσης σε πραγματικά περιβάλλοντα. Για κάθε κατηγορία εργαλείων πραγματοποιείται αναλυτική παρουσίαση, ενώ η συγκριτική αξιολόγηση βασίζεται σε κριτήρια, όπως η απόδοση, η ευκολία ενσωμάτωσης, η επεκτασιμότητα, η ασφάλεια, η διαλειτουργικότητα και η υποστήριξη από την κοινότητα.

Η ανάλυση ανέδειξε ώριμες λύσεις σε ορισμένους τομείς, όπως η παρακολούθηση (monitoring) και το CI/CD (Continuous Integration / Continuous Delivery) (Συνεχής Ενοποίηση / Συνεχής Παράδοση), αλλά και σημαντικές προκλήσεις που παραμένουν σε άλλους, όπως η διαλειτουργικότητα μεταξύ εργαλείων, η κλιμάκωση σε μεγάλες συστάδες (clusters), η ασφάλεια και η συμμόρφωση με κανονιστικά πλαίσια, καθώς και η περιορισμένη τεκμηρίωση, ιδιαίτερα εργαλείων μικρότερων κοινοτήτων. Βασική συνεισφορά της εργασίας αποτελεί η παροχή μιας ολοκληρωμένης εικόνας του τοπίου του Kubernetes και η ανάδειξη ερευνητικών κενών, συνοδευόμενη από συγκεκριμένες κατευθύνσεις για μελλοντική έρευνα, όπως η ανάπτυξη ενιαίων προτύπων, η αξιοποίηση τεχνικών Τεχνητής Νοημοσύνης και η εκπόνηση οικονομικών αναλύσεων κόστους–απόδοσης και κόστους-ιδιοκτησίας.

Λέξεις-κλειδιά: Kubernetes, Εργαλεία Kubernetes, Παρακολούθηση και Παρατηρησιμότητα, Ασφάλεια, Αυτόματη Κλιμάκωση, Συνεχής Ενοποίηση / Συνεχής Παράδοση, Πλέγμα Υπηρεσιών, Διαλειτουργικότητα, Τεχνητή Νοημοσύνη

Abstract

This thesis focuses on a comprehensive survey of the Kubernetes tools' ecosystem, aiming at their categorization, analysis, and comparative evaluation. The studied tools originate both from academic research and from the market, and are hierarchically classified based on a set of dimensions related to their functionality and their mode of integration in real environments. For each category of tools, an in-depth analysis is conducted, while the comparative evaluation relies on criteria, such as performance, ease of integration, scalability, security, interoperability, and community support.

The analysis highlighted mature solutions in certain domains, such as monitoring and CI/CD (Continuous Integration / Continuous Delivery), but also revealed significant challenges that persist in others, including interoperability across tools, large-scale cluster scalability, security and regulatory compliance, as well as limited documentation, especially for tools backed by smaller communities. The main contribution of this work lies in providing a holistic overview of the Kubernetes landscape and in identifying key research gaps, accompanied by concrete directions for future research, such as the development of common standards, the adoption of Artificial Intelligence techniques, and the implementation of cost–benefit and cost-of-ownership analyses.

Keywords: Kubernetes, Kubernetes Tools, Monitoring and Observability, Security, Autoscaling, CI/CD, Service Mesh, Interoperability, Artificial Intelligence

ΚΕΦΑΛΑΙΟ 1 – ΕΙΣΑΓΩΓΗ

1.1 Εξέλιξη και Υιοθέτηση του Kubernetes

Η ανάπτυξη λογισμικού σήμερα βρίσκεται σε μια μεταβατική περίοδο, με επίκεντρο τις αρχές της ευελιξίας, της επεκτασιμότητας και της αυτοματοποίησης. Οι παραδοσιακές μονολιθικές αρχιτεκτονικές λογισμικού αντικαθίστανται σταδιακά από κατανεμημένες μικρο-υπηρεσίες, οι οποίες εκτελούνται σε απομονωμένα, φορητά περιβάλλοντα εκτέλεσης, τα λεγόμενα containers (δοχεία) (Merkel, 2014). Η υιοθέτηση τεχνολογιών, όπως το υπολογιστικό νέφος (cloud computing), η "δοχειοποίηση" (containerization) και οι DevOps πρακτικές, έχει αλλάξει ριζικά σε πολλά επίπεδα τον τρόπο όπου οι εφαρμογές αναπτύσσονται, δοκιμάζονται, διατίθενται και συντηρούνται. Συγκεκριμένα, το cloud computing επιτρέπει την ελαστική και αποδοτική διάθεση πόρων, μειώνοντας το κόστος και διευκολύνοντας την κλιμάκωση. Η δοχειοποίηση εξασφαλίζει φορητότητα και συνέπεια, καθώς οι εφαρμογές εκτελούνται με τον ίδιο τρόπο ανεξάρτητα από το περιβάλλον. Παράλληλα, οι DevOps πρακτικές προωθούν την αυτοματοποίηση και τη συνεχή ολοκλήρωση/διάθεση (CI/CD), επιταχύνοντας τον κύκλο ανάπτυξης και αυξάνοντας την αξιοπιστία των συστημάτων. Ο συνδυασμός αυτών των τεχνολογιών μετασχηματίζει ριζικά το πεδίο ανάπτυξης λογισμικού, καθιστώντας το πιο ευέλικτο, επεκτάσιμο και προσαρμόσιμο στις απαιτήσεις της αγοράς.

Σε αυτό το πλαίσιο, το Kubernetes (ή K8s), το οποίο δημιουργήθηκε από την Google είναι πλέον τεχνολογία ανοιχτού κώδικα υποστηριζόμενη από τον μη κερδοσκοπικό οργανισμό CNCF (Cloud Native Computing Foundation) (θυγατρικό του Linux Foundation) και έχει καθιερωθεί ως μια αποδεκτή και ευρέως χρησιμοποιούμενη λύση στην βιομηχανία ανάπτυξης λογισμικού και κώδικα για την ενορχήστρωση δοχειοποιημένων (containerized) εφαρμογών (Burns, Grant, Oppenheimer, Brewer, & Wilkes, 2016). Η ταχύτητα ανάπτυξης, η ανάγκη για ευελιξία και οι απαιτήσεις διαθεσιμότητας και κλιμάκωσης, έχουν ενισχύσει την ευρεία διάδοσή του πλαισίου αυτού (Hwang, 2023). Το Kubernetes ως πλατφόρμα έχει πολλά πλεονεκτήματα μια και παρέχει μηχανισμούς για την αυτόματη κλιμάκωση, την παρακολούθηση κατάστασης των εφαρμογών, την εξισορρόπηση φορτίου (load balancing), και την αυτοθεραπεία (self-healing). Επίσης, προσφέρει την δυνατότητα για τον ορισμό πολιτικών πρόσβασης και πόρων, εξυπηρετώντας ανάγκες τόσο των επιχειρήσεων όσο και της ερευνητικής κοινότητας.

Σύμφωνα με το CNCF Annual Survey 2023, περισσότερο από το 96% των οργανισμών που αξιοποιούν cloud native τεχνολογίες χρησιμοποιούν Kubernetes σε παραγωγικά περιβάλλοντα (Cloud Native Computing Foundation, 2023). Η μεγάλη του απήχηση οφείλεται στη προσαρμογή του στις ανάγκες των εφαρμογών είτε πρόκειται για μικρές startup εφαρμογές σε μια συστάδα ενός κόμβου (single-node cluster), είτε για εταιρικές cloud-native εφαρμογές με εκατοντάδες συστατικά που εκτελούνται σε συστάδες πολλαπλών κόμβων (multi-node clusters). Επιπλέον, ο αρθρωτός (modular) χαρακτήρας του Kubernetes επιτρέπει την ενσωμάτωση πρόσθετων εργαλείων για παρακολούθηση, ασφαλή δρομολόγηση, CI/CD (Continuous Integration / Continuous Delivery) (Συνεχής Ενοποίηση / Συνεχής Παράδοση), αυτοματοποιημένη διαχείριση, και μηχανική μάθηση, μέσω διασύνδεσης APIs και CRDs (Custom Resource Definitions) (Hightower, Burns & Beda, 2017).

Ωστόσο, αυτή η ευελιξία και διαθεσιμότητα λειτουργιών δεν έρχεται χωρίς κόστος. Με την τεράστια υιοθέτηση του Kubernetes, έχει προκύψει ένα πλούσιο αλλά χαοτικό οικοσύστημα εργαλείων. Εκατοντάδες open-source και εμπορικά εργαλεία έχουν αναπτυχθεί από ανεξάρτητους προγραμματιστές, εταιρείες, και ακαδημαϊκούς φορείς – το καθένα με διαφορετικό στόχο, χαρακτηριστικά, τρόπους ενσωμάτωσης και επίπεδο ωριμότητας (Mayr & Putz, 2023). Αυτά τα εργαλεία ποικίλουν ως προς τη λειτουργία τους, όπως η παρακολούθηση (observability), η ασφάλεια (security), η δικτύωση (networking), η αυτόματη κλιμάκωση (autoscaling), μηχανική μάθηση (Machine Learning – ML) και άλλα (Spacelift, 2023). Επιπλέον, πολλά από τα εργαλεία αυτά είναι ανοιχτού κώδικα, διαθέσιμα σε πλατφόρμες. όπως το GitHub[1] ή το Artifact Hub[2].

1.2 Το Πρόβλημα: Πολυπλοκότητα και Κατακερματισμός

Η ραγδαία αύξηση του αριθμού αυτών των εργαλείων, αν και χρήσιμη, έχει δημιουργήσει σημαντικές προκλήσεις στην επιλογή και αξιολόγηση αυτών των εργαλείων. Η επιλογή κατάλληλου εργαλείου για κάθε ανάγκη έχει μετατραπεί σε πρόκληση. Πολλοί επαγγελματίες DevOps και ερευνητές δυσκολεύονται να αναγνωρίσουν ποια εργαλεία είναι κατάλληλα για κάθε περίπτωση, ποια διατηρούνται ενεργά, και ποια διαθέτουν επαρκή ωριμότητα & τεκμηρίωση. Οι διαχειριστές συστημάτων, οι developers (προγραμματιστές) και οι ερευνητές συχνά αντιμετωπίζουν ερωτήματα όπως: ποιο monitoring εργαλείο να επιλέξω; Πώς να αξιολογήσω έναν operator; Είναι ασφαλές αυτό το εργαλείο ανοικτού κώδικα (open-source); Τι υποστήριξη και τεκμηρίωση παρέχεται;

Το πρόβλημα δημιουργείται από το γεγονός ότι πολλά από τα εργαλεία δεν συνοδεύονται από ποιοτική ή πλήρη τεκμηρίωση, ενώ η διατήρηση (maintenance) τους μπορεί να μην είναι συνεπής. Επίσης, οι λίστες εργαλείων που κυκλοφορούν σε blogs, GitHub repos ή εταιρικά sites, συχνά περιέχουν ασαφή ή μη συγκρίσιμα κριτήρια (Shamim, Gibson, Morrison, & Rahman, 2022).

Επιπλέον, η έλλειψη ενός συγκεντρωτικού και μεθοδολογικά θεμελιωμένου πλαισίου κατηγοριοποίησης και αξιολόγησης δυσκολεύει ακόμη περισσότερο την έρευνα και την πράξη. Τα περισσότερα εργαλεία παρουσιάζονται αποσπασματικά, χωρίς συστηματική ανάλυση ή συγκριτικά κριτήρια (Shamim, Gibson, Morrison, & Rahman, 2022).

Σε μελέτη του The New Stack (2024), το 48% των επαγγελματιών που εργάζονται με Kubernetes δήλωσαν ότι δυσκολεύονται να επιλέξουν εργαλεία, λόγω της υπερπληθώρας επιλογών και έλλειψης συγκριτικής αξιολόγησης. (Hecht, 2024)

1.3 Συνεισφορά της Διπλωματικής

Η παρούσα διπλωματική εργασία φιλοδοξεί να καλύψει το υπαρκτό ερευνητικό και πρακτικό κενό που αφορά στην οργάνωση, αξιολόγηση και συγκριτική παρουσίαση του ταχύτατα διευρυνόμενου

[1] https://github.com/

[2] https://artifacthub.io/

οικοσυστήματος εργαλείων Kubernetes. Μέχρι σήμερα, αν και υπάρχουν μεμονωμένες τεχνικές αναφορές ή συλλογές εργαλείων σε διάφορα αποθετήρια και ιστολόγια, εντούτοις δεν έχει προταθεί ένα συστηματικό, επιστημονικά θεμελιωμένο πλαίσιο που να τα καταγράφει, να τα αναλύει και να τα συγκρίνει με τρόπο επεκτάσιμο και ουσιαστικό.

Η παρούσα μελέτη προσφέρει ένα πολυεπίπεδο ερευνητικό και τεχνολογικό αποτέλεσμα, που αποτυπώνεται σε τέσσερις βασικούς πυλώνες συνεισφοράς:

1. *Ιεραρχική Κατηγοριοποίηση Εργαλείων Kubernetes σε Θεματικές Λειτουργικές Ενότητες*. Στο πλαίσιο της εργασίας αυτής, επιχειρείται να μπει μια τάξη στο χαοτικό τοπίο των εργαλείων Kubernetes. Για τον σκοπό αυτό, σχεδιάστηκε και εφαρμόστηκε μια ιεραρχική ταξινόμηση, ώστε τα εργαλεία να οργανώνονται με βάση τη λειτουργικότητά τους και το πεδίο εφαρμογής τους. Η κατηγοριοποίηση αυτή βοηθά στην καλύτερη κατανόηση και ανάλυση των εργαλείων, καθώς τοποθετούνται σε θεματικά "clusters" (δηλ. ενότητες ή κατηγορίες), ανάλογα με το τι εξυπηρετούν: Operators, Monitoring, Security, Networking, Storage/Disaster Recovery, CI/CD, Cluster Management, Autoscaling, ML/AI, και Service Mesh.

2. *Ποιοτική και Ποσοτική Ανάλυση Εργαλείων Ανοιχτού Κώδικα*. Για κάθε εργαλείο που μελετάται, γίνεται μια προσπάθεια να αξιολογηθεί όχι μόνο με βάση το τι προσφέρει, αλλά και πώς το προσφέρει. Η ανάλυση βασίζεται σε αντικειμενικά και μελετημένα κριτήρια, όπως η ποιότητα της τεκμηρίωσής του, η απήχησή του στην κοινότητα (π.χ. GitHub stars και forks), το κατά πόσο υποστηρίζεται ενεργά, αλλά και πόσο εύκολα μπορεί να ενσωματωθεί σε υπάρχουσες υποδομές. Η εστίαση της έρευνας περιορίζεται αποκλειστικά σε εργαλεία ανοιχτού κώδικα, καθώς η ύπαρξη ζωντανής και ενεργής κοινότητας γύρω από αυτά παίζει καθοριστικό ρόλο στη βιωσιμότητα και στη συνεχή βελτίωσή τους.

3. *Συγκριτική Παρουσίαση και αξιολόγηση μέσω πινάκων και μετρικών*. Η διπλωματική εργασία εξετάζει με λεπτομέρεια πίνακες που συγκρίνουν εργαλεία ανά κατηγορία, ώστε να αναδείξει τις διαφορές και τα κοινά τους σημεία. Η χρήση τέτοιων πινάκων δεν περιορίζεται σε απλή παράθεση χαρακτηριστικών αλλά επιτρέπει μια πιο ουσιαστική και οπτική αποτύπωση της σύγκρισης. Με βάση σταθερές και τεκμηριωμένες μετρικές —όπως τεκμηρίωση, υποστήριξη, δημοφιλία και λειτουργική πληρότητα— επιχειρείται μια αντικειμενική αξιολόγηση των εργαλείων, βοηθώντας τον αναγνώστη να κατανοήσει πού υπερέχει ή υστερεί η κάθε επιλογή/εργαλείο.

4. *Ανάδειξη Ερευνητικών Κενών & Προτάσεις για Μελλοντική Ανάπτυξη*. Μέσα από τη μελέτη αναδεικνύονται αρκετά σημεία στα οποία τα υπάρχοντα εργαλεία παρουσιάζουν ελλείψεις ή δεν ανταποκρίνονται επαρκώς στις απαιτήσεις. Αυτά τα κενά δεν καταγράφονται απλώς, αλλά αποτελούν αφετηρία για σκέψη και περαιτέρω διερεύνηση. Γίνεται, λοιπόν, προσπάθεια να διατυπωθούν συγκεκριμένες και εφαρμόσιμες προτάσεις για μελλοντικές ερευνητικές κατευθύνσεις. Μεταξύ αυτών, περιλαμβάνονται η αξιοποίηση τεχνητής νοημοσύνης στην ενορχήστρωση συστημάτων, η ενίσχυση της παρατηρησιμότητας με πιο έξυπνα και αυτοματοποιημένα εργαλεία, καθώς και η βελτίωση της διαχείρισης σε πολύπλοκα περιβάλλοντα πολλαπλών συστάδων (multi-cluster).

1.4 Δομή της Εργασίας

Η παρούσα εργασία είναι δομημένη με τρόπο ώστε να ακολουθεί τη φυσική ροή της επιστημονικής μελέτης, ξεκινώντας από τις βασικές έννοιες και το θεωρητικό υπόβαθρο, προχωρώντας στη μεθοδολογία και στη συλλογή δεδομένων, ακολουθούμενη από αναλυτική κατηγοριοποίηση, αξιολόγηση, σύγκριση και καταλήγοντας με τα ερευνητικά συμπεράσματα και τις μελλοντικές κατευθύνσεις.

Η δομή περιλαμβάνει τα ακόλουθα κεφάλαια:

Κεφάλαιο 2 – Θεωρητικό Υπόβαθρο

Αυτό το κεφάλαιο λειτουργεί ως το θεμέλιο πάνω στο οποίο χτίζεται όλη η υπόλοιπη εργασία. Εδώ, παρουσιάζονται βασικές έννοιες που χρειάζεται να κατανοήσει κανείς πριν προχωρήσει στην ανάλυση των εργαλείων. Το κεφάλαιο περιλαμβάνει:

- Τον ορισμό και τα κύρια χαρακτηριστικά του υπολογιστικού νέφους (cloud computing), μαζί με μια επισκόπηση των διαφορετικών μοντέλων υπηρεσιών/παράδοσης (IaaS, PaaS, SaaS), αλλά και των μοντέλων υλοποίησης/ανάπτυξης (public, private, hybrid cloud).
- Την έννοια της δοχειοποίησης (containerization), σε αντιπαραβολή με την παραδοσιακή εικονικοποίηση (virtualization), καθώς και τη λειτουργία του Docker ως βασική τεχνολογία σε αυτό το πεδίο.
- Την ανάλυση του Kubernetes ως πλατφόρμας ορχήστρωσης δοχείων: πώς λειτουργεί, ποια είναι η αρχιτεκτονική του, και γιατί έχει καθιερωθεί ως κρίσιμο εργαλείο στο περιβάλλον του DevOps — μιας μεθοδολογίας που ενοποιεί την ανάπτυξη (Development) και τις λειτουργίες (Operations) με στόχο την αυτοματοποίηση, τη συνεχή ολοκλήρωση/παράδοση (CI/CD) και τη βελτίωση της αξιοπιστίας των εφαρμογών — και των cloud native υποδομών

Κεφάλαιο 3 – Μεθοδολογία Ανασκόπησης

Το κεφάλαιο αυτό εξηγεί τον τρόπο με τον οποίο προσεγγίστηκε η μελέτη, τόσο θεωρητικά όσο και πρακτικά. Περιγράφει βήμα-βήμα πώς δομήθηκε η διαδικασία αναζήτησης, επιλογής και αξιολόγησης των εργαλείων, δηλαδή η διαδικασία της μεθοδολογίας ανασκόπησης.

Πιο συγκεκριμένα:

- Παρουσιάζονται τα ερευνητικά ερωτήματα που αποτέλεσαν την αφετηρία για τη συλλογή και την ανάλυση των εργαλείων.
- Αναλύονται οι πηγές από τις οποίες αντλήθηκαν δεδομένα — όπως επιστημονικές βάσεις και αποθετήρια (π.χ. GitHub ή DockerHub) — καθώς και τα μοτίβα αναζήτησης.
- Περιγράφεται η μεθοδολογία που εφαρμόστηκε για να φιλτραριστούν και να τεκμηριωθούν τα εργαλεία με συνέπεια και αξιοπιστία βάσει συγκεκριμένου συνόλου από κριτήρια.

- Εισάγεται η ποσοτική διάσταση της μελέτης, μέσα από στατιστικά γραφήματα που απεικονίζουν τη συχνότητα αναφορών, τη δημοτικότητα των εργαλείων, και τη διασπορά των πηγών.

Πρόκειται για ένα θεμελιώδες κεφάλαιο, πάνω στο οποίο στηρίζεται όλη η εμπειρική προσέγγιση των επόμενων ενοτήτων

Κεφάλαιο 4 – Κατηγοριοποίηση και Ανάλυση Εργαλείων Kubernetes

Σε αυτό το κεφάλαιο παρουσιάζεται η κατηγοριοποίηση και ανάλυση των εργαλείων, με σκοπό να αναδειχθεί ο ρόλος και η λειτουργικότητά τους μέσα στο οικοσύστημα του Kubernetes.

Αρχικά, παρουσιάζεται μια ιεραρχική κατηγοριοποίηση, όπου τα εργαλεία ομαδοποιούνται ανάλογα με τη λειτουργική τους περιοχή και τον τρόπο ενσωμάτωσής τους σε συστάδες Kubernetes. Έπειτα, για κάθε μία από αυτές τις κατηγορίες, εξετάζονται προσεκτικά τα σχετικά επιλεγμένα εργαλεία, λαμβάνοντας υπόψη παραμέτρους, όπως η ποιότητα της τεκμηρίωσης, η δημοφιλία τους (π.χ. αριθμός αστεριών στο GitHub), το κατά πόσο υποστηρίζονται ενεργά και το πώς μπορούν να ενσωματωθούν σε υπάρχοντα περιβάλλοντα. Παράλληλα, για κάθε εργαλείο καταγράφονται τα βασικά του πλεονεκτήματα αλλά και οι αδυναμίες του.

Το κεφάλαιο αυτό δεν περιορίζεται σε μια απλή περιγραφή — εστιάζει στην ουσιαστική κατανόηση του τοπίου των διαθέσιμων εργαλείων, προσφέροντας μια ολοκληρωμένη εικόνα της ποικιλομορφίας και των δυνατοτήτων τους

Κεφάλαιο 5 – Συγκριτική Αξιολόγηση

Σε αυτό το κεφάλαιο δίνεται έμφαση στη συγκριτική θεώρηση των εργαλείων, έτσι ώστε να αναδειχθούν ουσιαστικές διαφορές και να προσφερθεί πρακτική καθοδήγηση για την επιλογή τους.

Η παρουσίαση γίνεται μέσα από συγκριτικούς πίνακες, οι οποίοι οργανώνονται ανά κατηγορία εργαλείων και βασίζονται σε συγκεκριμένα κριτήρια αξιολόγησης. Μέσα από αυτούς τους πίνακες, ερμηνεύονται τα συγκριτικά δεδομένα και αναδεικνύονται τα εργαλεία που ξεχωρίζουν — είτε για την απόδοσή τους, είτε για τις αδυναμίες τους.

Για κάθε κατηγορία, εξάγονται ξεκάθαρα συμπεράσματα που συνοψίζουν τα δυνατά χαρακτηριστικά των εργαλείων καθώς και σημεία που χρήζουν βελτίωσης. Η αξιολόγηση γίνεται με τρόπο συστηματικό και αντικειμενικό, με χρήση μετρικών και σκορ, ώστε να μπορεί να αξιοποιηθεί άμεσα και πρακτικά, τόσο από επαγγελματίες όσο και από ερευνητές.

Κεφάλαιο 6 – Ερευνητικά Κενά και Προτάσεις

Το κεφάλαιο αυτό εστιάζει στις βασικές αδυναμίες και περιορισμούς που εντοπίζονται στο σημερινό οικοσύστημα εργαλείων Kubernetes. Μέσα από τη μελέτη, αναδεικνύονται κρίσιμα ζητήματα που αφορούν τόσο τη διαλειτουργικότητα (interoperability), όσο και τη χρηστικότητα, τις δυνατότητες αυτοματοποίησης μέσω τεχνητής νοημοσύνης, καθώς και την παρατηρησιμότητα (observability) των συστημάτων.

Πέρα από τη διαπίστωση αυτών των αδυναμιών, προτείνονται συγκεκριμένες κατευθύνσεις για την ανάπτυξη νέων εργαλείων ή για τη βελτίωση των υπαρχόντων. Οι προτάσεις αυτές βασίζονται σε ερευνητικά δεδομένα και στοχεύουν σε ουσιαστικές βελτιώσεις.

Τέλος, γίνεται αναφορά σε μελλοντικές ερευνητικές προοπτικές, με έμφαση σε πιο εξειδικευμένα πεδία, όπως η ενορχήστρωση συστημάτων μέσω Τεχνητής Νοημοσύνησης (Artificial Intelligence – AI), η ανάπτυξη εργαλείων για περιβάλλοντα edge computing, και γενικότερα λύσεων που να ανταποκρίνονται στις αυξανόμενες ανάγκες του cloud native τοπίου.

Κεφάλαιο 7 – Συμπεράσματα

Το τελευταίο κεφάλαιο λειτουργεί ως κλείσιμο της εργασίας, προσφέροντας μια συνολική ανασκόπηση όσων αναλύθηκαν. Γίνεται μια ανακεφαλαίωση της συμβολής που είχε η εργασία τόσο στο θεωρητικό όσο και στο πρακτικό επίπεδο, αναδεικνύοντας τα σημεία στα οποία προσέφερε νέα γνώση ή οργανωμένη κατανόηση του τοπίου των εργαλείων Kubernetes.

Παρουσιάζονται επίσης τα βασικά συμπεράσματα που προέκυψαν από την ποιοτική και ποσοτική ανάλυση, με στόχο να αποτυπωθούν με σαφήνεια τα σημαντικότερα ευρήματα της μελέτης.

Ταυτόχρονα, γίνεται μια πιο προσωπική αναφορά στην εμπειρία της ερευνητικής διαδικασίας: τις προκλήσεις που αντιμετωπίστηκαν, την πορεία της ανακάλυψης, αλλά και τη συνολική αξία που προέκυψε μέσα από την εκπόνηση της διπλωματικής εργασίας.

Τέλος, τονίζεται η σημασία που έχει η τεκμηριωμένη και επιστημονικά δομημένη προσέγγιση στην τεχνολογία, ειδικά σε έναν τομέα τόσο δυναμικό όσο αυτός των εργαλείων Kubernetes — όπου η συνεχής εξέλιξη απαιτεί σταθερά θεμέλια πληροφόρησης και αξιολόγησης.

ΚΕΦΑΛΑΙΟ 2 – ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ

2.1 – Υπολογιστικό Νέφος

Τα τελευταία χρόνια, ο τρόπος με τον οποίο οι οργανισμοί αξιοποιούν την υπολογιστική ισχύ έχει αλλάξει ριζικά. Αντί να επενδύουν σε φυσική υποδομή, πολλές επιχειρήσεις στρέφονται προς το υπολογιστικό νέφος (cloud computing), δηλαδή την αξιοποίηση υπολογιστικών πόρων μέσω του διαδικτύου, «κατά παραγγελία» και ανάλογα με τις ανάγκες. Το νέφος επιτρέπει σε μια εταιρεία να αποκτήσει πρόσβαση σε αποθηκευτικό χώρο, υπολογιστική ισχύ ή και εφαρμογές, χωρίς να χρειάζεται να εγκαταστήσει ή να διαχειριστεί τίποτα τοπικά.

Ο επίσημος ορισμός που έχει δοθεί από το NIST περιγράφει το υπολογιστικό νέφος ως ένα μοντέλο που προσφέρει εύκολη και άμεση πρόσβαση σε ένα κοινό σύνολο υπολογιστικών πόρων (όπως διακομιστές, δίκτυα, βάσεις δεδομένων κ.ά.), οι οποίοι μπορούν να διατεθούν και να αποδεσμευτούν δυναμικά, με ελάχιστη ανθρώπινη παρέμβαση (Mell & Grance, 2011).

Τα βασικά χαρακτηριστικά του υπολογιστικού νέφους είναι τα εξής (Mell & Grance, 2011):

1. Αυτοματη τροφοδοσια υπηρεσιών κατ'απαίτηση
2. Πανταχού παρούσα πρόσβαση
3. Πολλαπλή Μίσθωση
4. Ταχεία Ελαστικότητα
5. Μετρούμενη Υπηρεσία

2.1.1 Ανάλυση των Βασικών Χαρακτηριστικών του Υπολογιστικού Νέφους

1. *Αυτόματη παροχή υπηρεσιών (On-demand self-service)*: Ένα από τα πιο θεμελιώδη χαρακτηριστικά του νέφους είναι η δυνατότητα που παρέχει στον χρήστη να προμηθεύεται υπηρεσίες –όπως υπολογιστική ισχύ, αποθηκευτικό χώρο, ή βάσεις δεδομένων– κατ'απαίτηση χωρίς την ανθρώπινη παρέμβαση (πχ. την παρέμβαση διαχειριστή). Αυτό επιτρέπει την άμεση και ευέλικτη ανάπτυξη και διάταξη/τροφοδοσία εφαρμογών. (Mell & Grance, 2011) (Redhat, 2023)

2. *Πανταχού παρούσα πρόσβαση (Ubiquitous Access)*: Οι υπηρεσίες νέφους είναι προσβάσιμες από οπουδήποτε μέσω διαδικτύου, και μπορούν να χρησιμοποιηθούν από ποικιλία συσκευών μέσω διαλειτουργικών διεπαφών. Αυτό διευκολύνει την απομακρυσμένη εργασία και την πρόσβαση σε δεδομένα. (Mell & Grance, 2011) (Wikipedia, 2024)

3. *Πολλαπλή Μίσθωση (Multi-Tenancy)*: Οι φυσικοί πόροι του παρόχου εξυπηρετούν πολλούς χρήστες ταυτόχρονα μέσω μοντέλου πολλαπλής μίσθωσης (multi-tenancy). Η εκχώρηση των

πόρων γίνεται δυναμικά με χρήση τεχνολογιών εικονικοποίησης (virtualization) και δοχείων (containers). (Fehling, Leymann, Retter, Schupeck , & Arbitter, 2014)

4. *Ταχεία ελαστικότητα (Rapid Elasticity)*: Το νέφος προσφέρει δυνατότητα ταχείας κλιμάκωσης (scaling) των πόρων μέσω κατάλληλων μηχανισμών, επιτρέποντας τις εφαρμογές να μπορούν να ανταποκρίνονται σε μεταβαλλόμενο φόρτο, αυξάνοντας τον αριθμό των συστατικών τους που θα τροφοδοτούνται από τους κλιμακούμενους πόρους (Gogineni, 2022).

5. *Μετρούμενη υπηρεσία* (Measured Service): Οι πάροχοι νέφους μετρούν και παρακολουθούν κάθε πόρο που χρησιμοποιείται, επιτρέποντας τη δυναμική χρέωση με βάση την κατανάλωση. Αυτό εξασφαλίζει διαφάνεια, προβλεψιμότητα και οικονομική αποδοτικότητα. (Mell & Grance, 2011) (Redhat, 2023)

2.1.2 Ανάλυση των Μοντέλων Υπηρεσιών στο Υπολογιστικό Νέφος

Ένα από τα πλέον σημαντικά στοιχεία που διαφοροποιούν το υπολογιστικό νέφος από τις παραδοσιακές υποδομές IT (Information Technology) είναι η δυνατότητα που προσφέρει στους οργανισμούς να επιλέξουν το επίπεδο του ελέγχου και της ευθύνης που επιθυμούν να διατηρήσουν πάνω στους πόρους και τις εφαρμογές τους. Η επιλογή αυτή υλοποιείται μέσα από τα εξής τρία κύρια μοντέλα παροχής/παράδοσης υπηρεσιών (delivery models) στο υπολογιστικό νέφος, τα οποία έχουν καθιερωθεί διεθνώς από το National Institute of Standards and Technology (NIST): Infrastructure as a Service (IaaS), Platform as a Service (PaaS) και Software as a Service (SaaS) (Mell & Grance, 2011).

Infrastructure as a Service (IaaS)

Το μοντέλο IaaS (Υποδομή ως Υπηρεσία) αποτελεί τη θεμελιώδη και χαμηλότερη βαθμίδα παροχής υπηρεσιών στο νέφος. Παρέχει στον χρήστη υποδομές πληροφορικής σε μορφή εικονικοποιημένων πόρων: εικονικές μηχανές (virtual machines), αποθηκευτικό χώρο, δικτυακά συστήματα και τοίχη προστασίας (firewalls). Σε αυτό το μοντέλο, η ευθύνη διαχείρισης της υποδομής (hardware, virtualization, storage, networks) ανήκει στον πάροχο, ενώ ο τελικός χρήστης διαχειρίζεται πλήρως το λειτουργικό σύστημα, τις εφαρμογές, τις ρυθμίσεις ασφαλείας και τα δεδομένα που μεταφέρει/εγκαθιστά/διαμορφώνει σε μια εικονική μηχανή.

Αυτό το μοντέλο είναι ιδιαίτερα κατάλληλο για επιχειρήσεις που επιθυμούν πλήρη έλεγχο του περιβάλλοντος εκτέλεσης των εφαρμογών τους χωρίς να επενδύουν στην αγορά και συντήρηση φυσικού εξοπλισμού. Παράλληλα, προσφέρει δυνατότητα κλιμάκωσης (scalability) των πόρων ανάλογα με τις ανάγκες, πληρώνοντας μόνο για τη χρήση που γίνεται.

Ενδεικτικά παραδείγματα υπηρεσιών IaaS περιλαμβάνουν το Amazon EC2[3], το Google Compute Engine (GCE), [4]το Microsoft Azure VMs[5].

Platform as a Service (PaaS)

Το PaaS (Πλατφόρμα ως Υπηρεσία) αναβαθμίζει την αφαίρεση των τεχνικών λεπτομερειών σε σχέση με το IaaS, προσφέροντας ένα ολοκληρωμένο περιβάλλον ανάπτυξης και εκτέλεσης εφαρμογών, χωρίς να απαιτείται διαχείριση του λειτουργικού συστήματος, της υποδομής ή των συστατικών του περιβάλλοντος αυτού (όπως βιβλιοθήκες). Σε αυτό το μοντέλο, ο πάροχος φροντίζει για όλες τις ανάγκες της υποδομής και προσφέρει στον χρήστη APIs, πλαίσια (frameworks) ανάπτυξης, βάσεις δεδομένων, μηχανισμούς διαχείρισης και εργαλεία αποσφαλμάτωσης (debugging), προκειμένου να επικεντρωθεί αποκλειστικά στην ανάπτυξη της εφαρμογής. Σημειώνεται πως το όλο περιβάλλον μπορεί να κλιμακώνεται αυτόματα ανάλογα με τον φόρτο.

Το PaaS απευθύνεται κυρίως σε ομάδες ανάπτυξης λογισμικού που αναζητούν ευκολία, ταχύτητα και αυτοματοποίηση των διαδικασιών κατασκευής, δοκιμής και διάταξης (build/test/deploy). Αυτό επιτυγχάνεται ενσωματώνοντας τις περισσότερες φορές εργαλεία CI/CD. Παράλληλα, παρέχεται και υποστήριξη για μικρο-υπηρεσίες ή δοχεία.

Χαρακτηριστικά παραδείγματα περιλαμβάνουν το Google App Engine[6], το Heroku[7], το Microsoft Azure App Services[8], και το Red Hat OpenShift[9].

Software as a Service (SaaS)

Το SaaS (Λογισμικό ως Υπηρεσία) είναι το πιο διαδεδομένο[10] και ταυτόχρονα το πιο αφαιρετικό μοντέλο. Παρέχει στον χρήστη πλήρως λειτουργικές εφαρμογές, προσβάσιμες μέσω φυλλομετρητή (browser) ή thin clients, χωρίς να απαιτείται καμία εγκατάσταση ή διαχείριση από τον χρήστη. Όλα – από το hardware και τα updates, έως το λογισμικό και τα δεδομένα – διαχειρίζονται αποκλειστικά από τον πάροχο. Το ίδιο ισχύει και για την κλιμάκωση των εφαρμογών αλλά και την ενημέρωσή τους.

---

[3] https://aws.amazon.com/ec2/

[4] https://cloud.google.com/compute/docs

[5] https://azure.microsoft.com/en-us/products/virtual-machines

[6] https://cloud.google.com/appengine?hl=en

[7] https://www.heroku.com/

[8] https://azure.microsoft.com/en-us/products/app-service

[9] https://www.redhat.com/en/technologies/cloud-computing/openshift

[10] https://www.ibm.com/think/topics/iaas-paas-saas

Το μοντέλο αυτό είναι ιδιαίτερα ελκυστικό για τελικούς χρήστες και επιχειρήσεις που αναζητούν γρήγορη και αξιόπιστη πρόσβαση σε λογισμικό χωρίς την ανάγκη για τεχνικές γνώσεις ή επένδυση σε υποδομή. Το SaaS είναι επίσης συμβατό με τη λογική του freemium και της συνδρομητικής χρέωσης (subscription-based billing). Τέλος, οι SaaS εφαρμογές και ειδικότερα τα API που προσφέρουν μπορούν να γίνουν εκμεταλλεύσιμα για την παραγωγή νέων εφαρμογών προστιθέμενης αξίας.

Ενδεικτικές SaaS εφαρμογές είναι το Google Workspace[11] (Docs, Gmail, Drive), το Microsoft 365[12], το Salesforce CRM[13], και το Dropbox[14].

Multi-cloud & Cross-cloud περιβάλλοντα:
Σύμφωνα με τον Gogineni (2022), πολλές επιχειρήσεις αξιοποιούν στρατηγικές multi-cloud (πολλαπλών νεφών) για αξιοπιστία και μείωση του vendor lock-in (κλειδώματος σε πάροχο). Προχωρημένες περιπτώσεις αφορούν την cross-cloud (κατά μήκος νεφών) αρχιτεκτονική, όπου υπηρεσίες λειτουργούν διαλειτουργικά μεταξύ διαφορετικών παρόχων. Δηλαδη πέρα από τα βασικά μοντέλα παράδοσης υπηρεσιών του υπολογιστικού νέφους (IaaS, PaaS, SaaS), οι επιχειρήσεις συχνά υιοθετούν στρατηγικές που αξιοποιούν πολλαπλούς παρόχους ταυτόχρονα. Το **multi-cloud** (πολυνέφος ή πολλαπλό νέφος) αναφέρεται στη χρήση υπηρεσιών από διαφορετικούς παρόχους cloud, σε οποιοδήποτε επίπεδο παράδοσης (π.χ. SaaS από τη Microsoft, PaaS από την Google, IaaS από την AWS). Η προσέγγιση αυτή δεν συνιστά νέο μοντέλο, αλλά στρατηγική αξιοποίησης των υπαρχόντων μοντέλων με στόχο την αποφυγή του vendor lock-in, την αύξηση της αξιοπιστίας και τη βελτιστοποίηση κόστους και επιδόσεων. Σε πιο προχωρημένες περιπτώσεις συναντάται η έννοια του **cross-cloud** (διανέφος), όπου οι εφαρμογές ή οι υπηρεσίες ενός οργανισμού λειτουργούν συνδυαστικά σε πολλαπλούς παρόχους, επιτυγχάνοντας διαλειτουργικότητα και ευελιξία, αλλά και εισάγοντας αυξημένη πολυπλοκότητα στη διαχείριση. Έτσι, τα multi- και cross-cloud περιβάλλοντα λειτουργούν συμπληρωματικά στα μοντέλα παράδοσης του νέφους, ανταποκρινόμενα σε επιχειρησιακές ανάγκες που ξεπερνούν τα όρια ενός μόνο παρόχου

2.1.3 Μοντέλα Ανάπτυξης Υπολογιστικού Νέφους

Το υπολογιστικό νέφος έχει γίνει σήμερα αναπόσπαστο κομμάτι της τεχνολογίας που χρησιμοποιούν επιχειρήσεις, οργανισμοί αλλά και απλοί χρήστες. Ωστόσο, το «νέφος» δεν είναι ένα ενιαίο πράγμα· υπάρχουν **διαφορετικοί τρόποι** με τους οποίους μπορεί να υλοποιηθεί και να χρησιμοποιηθεί — ανάλογα με τις ανάγκες, τον βαθμό ασφάλειας, το κόστος και τον έλεγχο που θέλει να έχει ο κάθε χρήστης.

[11] https://workspace.google.com/

[12] https://www.microsoft.com/el-gr/microsoft-365

[13] https://www.salesforce.com/products/what-is-salesforce/

[14] https://www.dropbox.com/

Αυτοί οι διαφορετικοί τρόποι ονομάζονται **μοντέλα ανάπτυξης νέφους** (cloud deployment models) και αναλύονται παρακάτω:

- Το **Public Cloud (Δημόσιο Νέφος)**, όπου οι υπηρεσίες παρέχονται από τρίτους παρόχους, όπως η Google ή η Amazon, και είναι διαθέσιμες σε οποιονδήποτε μέσω διαδικτύου.
- Το **Private Cloud (Ιδιωτικό Νέφος)**, που χρησιμοποιείται αποκλειστικά από έναν οργανισμό και παρέχει αυξημένο έλεγχο και ασφάλεια.
- Το **Hybrid Cloud (Υβριδικό Νέφος)**, ένας συνδυασμός δημόσιου και ιδιωτικού νέφους, που επιτρέπει ευελιξία και έξυπνη διαχείριση πόρων.
- Και το **Community Cloud (Κοινοτικό Νέφος)**, το οποίο μοιράζεται ανάμεσα σε οργανισμούς με κοινές ανάγκες, όπως πανεπιστήμια ή νοσοκομεία.

Κάθε μοντέλο έχει τα δικά του χαρακτηριστικά, πλεονεκτήματα και περιορισμούς — και η επιλογή του κατάλληλου εξαρτάται πάντα από το ποιος το χρησιμοποιεί και για ποιο σκοπό. Στις επόμενες ενότητες, θα δούμε αναλυτικά το κάθε μοντέλο, τι προσφέρει, πού υπερτερεί και πού υστερεί, πάντα με βάση έγκυρες επιστημονικές πηγές

2.1.3.1 Δημόσιο Νέφος

Το **δημόσιο νέφος** είναι ένα μοντέλο ανάπτυξης υπολογιστικού νέφους στο οποίο **οι υποδομές, οι εφαρμογές και οι πλατφόρμες ανήκουν και διαχειρίζονται από εξωτερικούς παρόχους** όπως η Amazon (AWS), Microsoft Azure, Google Cloud Platform κ.ά. Αυτοί οι πόροι είναι **προσβάσιμοι στο κοινό** μέσω του Διαδικτύου.

Η φιλοσοφία του δημόσιου νέφους βασίζεται σε μοντέλα "as-a-service" (SaaS, PaaS, IaaS) και δίνει τη δυνατότητα στους χρήστες να **εκμεταλλεύονται δυναμικά υπολογιστικούς πόρους χωρίς να χρειάζεται να επενδύσουν σε φυσική υποδομή**. Οι χρήστες πληρώνουν για ό,τι καταναλώνουν, χωρίς να διαχειρίζονται την υποδομή ή το υλικό.

Το μοντέλο βασίζεται επίσης σε πολλαπλή μίσθωση (**multi-tenancy)**, δηλαδή πολλοί πελάτες (tenants) χρησιμοποιούν ταυτόχρονα την ίδια υποδομή του παρόχου, κάτι που προσφέρει οικονομία κλίμακας αλλά δημιουργεί και ανησυχίες για την ασφάλεια και απομόνωση των δεδομένων.

<u>Πλεονεκτήματα</u>

1. *Χαμηλό κόστος εκκίνησης και λειτουργίας*
   Οι επιχειρήσεις δεν χρειάζεται να επενδύσουν σε φυσικές υποδομές. Το μοντέλο “pay-as-you-go” επιτρέπει στους οργανισμούς να πληρώνουν μόνο για τους πόρους που καταναλώνουν. (Khan, et al., 2022)

2. *Υψηλή ελαστικότητα (elasticity)*
   Οι δημόσιοι πάροχοι προσφέρουν θεωρητικά απεριόριστους πόρους. Αυτό επιτρέπει την αυτόματη προσαρμογή του συστήματος στις ανάγκες οποιασδήποτε στιγμής (π.χ. απότομη αύξηση φόρτου). (Alghofaili, Albattah , Alrajeh, Rassam, & Al-rimy, 2021)

3. *Ευρεία πρόσβαση και διαθεσιμότητα*
   Οι υπηρεσίες είναι διαθέσιμες παντού με σύνδεση στο Διαδίκτυο και από ποικιλία συσκευών (PC, tablet, κινητό). (Alghofaili, Albattah , Alrajeh, Rassam, & Al-rimy, 2021)

4. *Περιβαλλοντική αποδοτικότητα*
   Η χρήση δημόσιου νέφους σε πράσινες υποδομές μπορεί να οδηγήσει σε σημαντική μείωση του ενεργειακού αποτυπώματος. (Turek, Dziembek, & Hernes, 2021)

Μειονεκτήματα

1. *Περιορισμένος έλεγχος και διαφάνεια*
   Ο χρήστης δεν έχει φυσική ή λογική πρόσβαση στο υποκείμενο υλικό (hardware) ή στο επίπεδο ασφαλείας του παρόχου. (Khan, et al., 2022) Επιπλέον, συχνά υπάρχει έλλειψη **διαφάνειας**, καθώς οι πάροχοι δεν γνωστοποιούν πάντα με σαφήνεια πού αποθηκεύονται τα δεδομένα, ποια μέτρα προστασίας εφαρμόζονται και ποια τρίτα μέρη ενδέχεται να έχουν πρόσβαση, περιορίζοντας έτσι τον ουσιαστικό έλεγχο του οργανισμού πάνω στις κρίσιμες πληροφορίες του.

2. *Ανησυχίες για ασφάλεια και ιδιωτικότητα λόγω πολλαπλής μίσθωσης*
   Η κοινή χρήση των υποδομών μπορεί να οδηγήσει σε παραβιάσεις απορρήτου ή επιθέσεις μεταξύ μισθωτών (tenants), ιδιαίτερα εφόσον δεν υπάρχει κατάλληλο επίπεδο απομόνωσης μεταξύ των περιβαλλόντων που διατίθενται στους μισθωτές. (Alghofaili, Albattah , Alrajeh, Rassam, & Al-rimy, 2021)

3. *Νομικά και κανονιστικά ζητήματα (data sovereignty)*
   Πολλές χώρες και οργανισμοί ανησυχούν για το πού αποθηκεύονται τα δεδομένα τους, ειδικά αν βρίσκονται εκτός της εθνικής τους δικαιοδοσίας. (Galij, Pawlak, & Grzyb, 2024) Αν και ορισμένοι πάροχοι νέφους δίνουν τη δυνατότητα στους πελάτες να επιλέγουν γεωγραφική περιοχή αποθήκευσης (data residency options), στην πράξη ο έλεγχος είναι περιορισμένος: η τελική τοποθεσία μπορεί να εξαρτάται από τη διαθεσιμότητα κέντρων δεδομένων, τις εσωτερικές πολιτικές του παρόχου ή τις απαιτήσεις απόδοσης. Αυτό ενισχύει τις ανησυχίες περί διαφάνειας και κυριαρχίας δεδομένων, καθώς οι οργανισμοί δεν έχουν πάντα πλήρη ορατότητα ούτε απόλυτη δυνατότητα καθορισμού της τοποθεσίας αποθήκευσης.

4. *Δυσκολία συμμόρφωσης με GDPR και άλλα πρότυπα*
   Η τοποθεσία αποθήκευσης και η επεξεργασία προσωπικών δεδομένων ενδέχεται να παραβιάζουν νομικές ρυθμίσεις. (Khan, et al., 2022) Πέρα από το ζήτημα της κυριαρχίας των δεδομένων (data sovereignty), οι οργανισμοί καλούνται να συμμορφωθούν με

κανονιστικά πλαίσια, όπως ο Γενικός Κανονισμός Προστασίας Δεδομένων (GDPR) στην Ε.Ε. ή το CCPA στις Η.Π.Α. Η συμμόρφωση δεν αφορά μόνο την τοποθεσία αποθήκευσης, αλλά και ζητήματα όπως η διαφάνεια στην επεξεργασία προσωπικών δεδομένων, η λήψη συναίνεσης, η δυνατότητα διαγραφής και ο περιορισμός πρόσβασης από τρίτα μέρη. Σε περιβάλλοντα νέφους, όπου τα δεδομένα ενδέχεται να μεταφέρονται δυναμικά μεταξύ κέντρων δεδομένων και παρόχων, η τήρηση αυτών των προτύπων καθίσταται ιδιαίτερα σύνθετη.

2.1.3.2 Ιδιωτικό Νέφος

Το **ιδιωτικό νέφος** είναι ένα μοντέλο στο οποίο η υποδομή νέφους δεν είναι προσβάσιμη στο (ευρύ) κοινό, αλλά χρησιμοποιείται αποκλειστικά από έναν οργανισμό. Αυτό σημαίνει ότι τόσο ο υλικός εξοπλισμός (servers, δίκτυα, αποθηκευτικοί πόροι) όσο και οι υπηρεσίες λογισμικού διαχειρίζονται και λειτουργούν μόνο για τις ανάγκες του συγκεκριμένου οργανισμού.

Η υλοποίηση ενός ιδιωτικού νέφους μπορεί να γίνει με δύο τρόπους: είτε τοπικά, εντός του ίδιου του οργανισμού (on-premises), είτε μέσω τρίτου παρόχου, ο οποίος προσφέρει τις υπηρεσίες (ιδιωτικού νέφους) στον συγκεκριμένο πελάτη/οργανισμό. Η κύρια διαφορά με το δημόσιο νέφος είναι ότι στο ιδιωτικό νέφος ο οργανισμός έχει πλήρη έλεγχο σε όλα τα επίπεδα: από την ασφάλεια και τη συμμόρφωση μέχρι τη διαχείριση πόρων.

Το ιδιωτικό νέφος επιλέγεται συνήθως από οργανισμούς με αυξημένες απαιτήσεις ασφάλειας, όπως τράπεζες, νοσοκομεία, πανεπιστήμια ή κρατικοί φορείς, που χειρίζονται ευαίσθητα ή εμπιστευτικά δεδομένα.

<u>Πλεονεκτήματα</u>

1. *Πλήρης έλεγχος στα δεδομένα και την ασφάλεια*
   Ο οργανισμός μπορεί να εφαρμόσει τα δικά του πρότυπα ασφάλειας, να ελέγχει ποιος έχει πρόσβαση και πώς αποθηκεύονται και προστατεύονται τα δεδομένα. (Ahmad, Rasool, Javed, Baker, & Jalil, 2021) – Επομένως, προτείνεται το ιδιωτικό νέφος για εφαρμογές όπου απαιτείται υψηλή εμπιστευτικότητα

2. *Συμμόρφωση με κανονιστικά πλαίσια (π.χ. GDPR, HIPAA)*
   Επειδή ο οργανισμός διαχειρίζεται το νέφος του, μπορεί να προσαρμόσει τη λειτουργία του νέφους του ώστε να πληροί νομικές και κανονιστικές απαιτήσεις. (Alghofaili, Albattah , Alrajeh, Rassam, & Al-rimy, 2021)

3. *Εξατομίκευση και βελτιστοποίηση πόρων*
   Οι υποδομές μπορούν να διαμορφωθούν ανάλογα με τις συγκεκριμένες ανάγκες του οργανισμού, όπως επιδόσεις, δικτυακές πολιτικές, ή αποθήκευση. (Zhao, Hu, & Xu, 2024)

4. *Προστασία από απειλές τρίτων χρηστών (no multi-tenancy)*
   Σε αντίθεση με το δημόσιο νέφος, δεν υπάρχει κίνδυνος “συνύπαρξης” με άγνωστους

χρήστες που ενδέχεται να προκαλέσουν τρωτότητες. (Ahmad, Rasool, Javed, Baker, & Jalil, 2021)

Μειονεκτήματα

1. *Υψηλό κόστος επένδυσης και λειτουργικό κόστος*
   Ο οργανισμός πρέπει να επενδύσει σε εξοπλισμό, λειτουργία, συντήρηση και εξειδικευμένο προσωπικό, γεγονός που αυξάνει τα έξοδα σε σχέση με το δημόσιο νέφος. (Zhao, Hu, & Xu, 2024)

2. *Περιορισμένη κλιμακωσιμότητα*
   Οι φυσικοί πόροι του ιδιωτικού νέφους είναι πεπερασμένοι και δεν μπορούν εύκολα να αυξηθούν χωρίς επένδυση σε νέο υλικό. Αυτό σημαίνει πως εφόσον παρατηρηθεί αυξημένος φόρτος, αυτός δεν θα μπορεί να αντιμετωπιστεί άμεσα εφόσον ο τρέχων αριθμός πόρων στο ιδιωτικό νέφος δεν είναι επαρκής. (Alghofaili, Albattah , Alrajeh, Rassam, & Al-rimy, 2021)

3. *Ανάγκη για εξειδικευμένο προσωπικό και διαχείριση*
   Το ιδιωτικό νέφος απαιτεί ενισχυμένες διαδικασίες ελέγχου και διοίκησης IT πόρων. Συνεπώς, η σωστή λειτουργία ενός ιδιωτικού νέφους προϋποθέτει έμπειρους μηχανικούς, συνεχείς αναβαθμίσεις και διαρκή συντήρηση. Επομένως, αν ένας οργανισμός δεν έχει την κατάλληλη τεχνογνωσία, θα πρέπει να προχωρήσει σε σημαντικές προσλήψεις, αυξάνοντας το κόστος, εφόσον επιθυμεί την υιοθέτηση και σωστή λειτουργία και διαχείριση ενός ιδιωτικού νέφους. (Ahmad, Rasool, Javed, Baker, & Jalil, 2021)

2.1.3.3 Υβριδικό Νέφος

Το **υβριδικό νέφος** είναι ένα μοντέλο ανάπτυξης που συνδυάζει στοιχεία από διαφορετικά είδη νεφών, όπως δημόσιο (public), ιδιωτικό (private) και σε ορισμένες περιπτώσεις κοινοτικό (community) νέφος. Στην πράξη, αυτό επιτρέπει σε έναν οργανισμό να φιλοξενεί ευαίσθητες εφαρμογές και δεδομένα σε ένα ασφαλές, εσωτερικό περιβάλλον, ενώ ταυτόχρονα να αξιοποιεί την κλιμακούμενη υπολογιστική ισχύ του δημόσιου νέφους ή άλλων παρόχων για λιγότερο κρίσιμες εργασίες ή σε περιόδους αυξημένης ζήτησης. Ένα χαρακτηριστικό παράδειγμα αποτελεί το σενάριο **cloud bursting**, όπου ένα ιδιωτικό νέφος επεκτείνεται δυναμικά σε ένα ή περισσότερα δημόσια νέφη, προκειμένου να καλυφθούν προσωρινά αυξημένες ανάγκες χωρίς μόνιμη επένδυση σε πρόσθετη υποδομή (Ahmad et al., 2022).

Το υβριδικό νέφος λειτουργεί ως «γέφυρα» ανάμεσα σε διαφορετικά περιβάλλοντα νέφους, επιτρέποντας δυναμική μεταφορά δεδομένων και εργασιών ανάλογα με τις ανάγκες. Αυτό όμως δεν γίνεται αυτόματα· προϋποθέτει τη χρήση **μηχανισμών διαλειτουργικότητας**. Οι μηχανισμοί αυτοί μπορεί να βασίζονται σε **ανοιχτά πρότυπα** και τεχνολογίες (π.χ. APIs, container

orchestration με Kubernetes), που διευκολύνουν τη φορητότητα και την ανεξαρτησία από συγκεκριμένο πάροχο, ή σε **ιδιοταγείς λύσεις** που παρέχουν οι ίδιοι οι πάροχοι νέφους (π.χ. AWS Direct Connect[15], Azure Arc[16], Google Anthos[17]). Η επιλογή μεταξύ προτύπων ή proprietary τεχνολογιών επηρεάζει τον βαθμό ευελιξίας, το κόστος και τον κίνδυνο vendor lock-inΑυτό προσφέρει ευελιξία, οικονομία και υψηλή απόδοση — αλλά συνοδεύεται και από αυξημένη τεχνική πολυπλοκότητα (Andreazi et al., 2021).

Χρησιμοποιείται συχνά σε οργανισμούς που χρειάζονται συμμόρφωση και ασφάλεια για ορισμένα δεδομένα, αλλά παράλληλα θέλουν να αξιοποιούν τη δυναμική κλιμάκωση και τα πλεονεκτήματα κόστους του δημόσιου νέφους (Korada & Sikha, 2024).

Πλεονεκτήματα

1. *Ευελιξία και προσαρμοστικότητα*
   Το υβριδικό νέφος δίνει τη δυνατότητα στον οργανισμό να επιλέγει πού θα εκτελεί κάθε εργασία: στο ασφαλές ιδιωτικό περιβάλλον ή στο οικονομικό δημόσιο νέφος. Μάλιστα, στο IoT χρησιμοποιείται για τον διαχωρισμό μεταξύ κρίσιμων και μη κρίσιμων δεδομένων. (Ahmad, Rasool, Javed, Baker, & Jalil, 2021)

2. *Βελτιστοποίηση κόστους και πόρων*
   Με τη χρήση δημόσιου νέφους μόνο όταν απαιτείται (π.χ. σε αυξημένη ζήτηση), μειώνεται η ανάγκη για μόνιμη επένδυση σε εξοπλισμό. Για παράδειγμα, η Dropbox μετέφερε μεγάλο μέρος των υπηρεσιών της πίσω σε υβριδική υποδομή για εξοικονόμηση κόστους. (Korada & Sikha, 2024)

3. *Καλύτερη υποστήριξη επιχειρησιακής συνέχειας*
   Αν μία πλατφόρμα αποτύχει (π.χ. το ιδιωτικό κέντρο δεδομένων), οι υπηρεσίες μπορούν να μετακινηθούν προσωρινά στο δημόσιο νέφος για αδιάλειπτη λειτουργία. Για παράδειγμα, προς αυτό τον σκοπό, το MoHRiPA προσφέρει δυναμική ανακατανομή φόρτου μεταξύ ιδιωτικών και δημόσιων υποδομών. (Andreazi, et al., 2021)

4. *Σταδιακή μετάβαση στο νέφος (cloud migration)*
   Το υβριδικό νέφος επιτρέπει σε οργανισμούς, ιδιαίτερα σε μικρότερες επιχειρήσεις, που ξεκινούν από τοπικές υποδομές να μεταβούν σταδιακά στο νέφος, χωρίς να χρειάζεται πλήρης αναδιοργάνωση. (Ahmad, Rasool, Javed, Baker, & Jalil, 2021)

Μειονεκτήματα

1. *Πολυπλοκότητα ενσωμάτωσης και διαχείρισης*
   Η σύνδεση μεταξύ δύο διαφορετικών περιβαλλόντων (π.χ. on-premises και cloud) απαιτεί εξειδικευμένες γνώσεις και εργαλεία. Για παράδειγμα, μια τεχνική πρόκληση αφορά τη

[15] https://aws.amazon.com/directconnect/

[16] https://azure.microsoft.com/en-us/products/azure-arc/

[17] https://cloud.google.com/kubernetes-engine?hl=en

διασύνδεση hypervisors (υπερεποπτών) και cloud orchestration layers (επιπέδων ενορχήστρωσης νέφους). (Andreazi, et al., 2021)

2. *Ασφάλεια στα σημεία σύνδεσης (interfaces)*
   Τα δεδομένα που μετακινούνται μεταξύ ιδιωτικού και δημόσιου νέφους μπορεί να είναι ευάλωτα, ειδικά χωρίς σωστή κρυπτογράφηση και αυθεντικοποίηση. (Ahmad, Rasool, Javed, Baker, & Jalil, 2021)

3. *Κόστος αρχικής ρύθμισης*
   Αν και το κόστος λειτουργίας μπορεί να είναι χαμηλότερο, η αρχική υλοποίηση ενός υβριδικού περιβάλλοντος συχνά απαιτεί σημαντικούς πόρους και τεχνική υποδομή. (Andreazi, et al., 2021)

2.1.3.4 Κοινοτικό Νέφος

Το **κοινοτικό νέφος** είναι ένα μοντέλο ανάπτυξης νέφους που έχει σχεδιαστεί ώστε να χρησιμοποιείται από μια ομάδα οργανισμών που έχουν κοινές ανάγκες, πολιτικές ή κανονιστικές απαιτήσεις. Οι οργανισμοί αυτοί — π.χ. πανεπιστήμια, νοσοκομεία, ερευνητικά κέντρα, κυβερνητικοί φορείς — συνεργάζονται και μοιράζονται μια κοινή υποδομή νέφους, είτε εντός των εγκαταστάσεών τους είτε μέσω ενός τρίτου φορέα.

Το κοινοτικό νέφος λειτουργεί σαν ένας «συνεταιρισμός νέφους», με στόχο να προσφέρει βελτιωμένη ασφάλεια, μείωση κόστους, καλύτερη συμμόρφωση και πιο στοχευμένες υπηρεσίες από ό,τι προσφέρει το δημόσιο νέφος, αλλά χωρίς το πλήρες κόστος ενός ιδιωτικού νέφους.

Η πρόσβαση και χρήση των πόρων είναι αποκλειστική στα μέλη της κοινότητας, με διαχειριζόμενη πολιτική, ενώ η τεχνική υποδομή προσαρμόζεται στις ανάγκες όλων των συμμετεχόντων.

<u>Πλεονεκτήματα</u>

1. *Συνεργατική χρήση πόρων & διαμοιρασμένο κόστος*
   Οι οργανισμοί μοιράζονται τις υποδομές και τις υπηρεσίες, μειώνοντας το συνολικό κόστος που θα είχαν αν λειτουργούσαν ιδιωτικό νέφος ο καθένας ξεχωριστά. Για παράδειγμα, το MSMC επιτυγχάνει ισότιμη κατανομή πόρων σε πολλαπλούς οργανισμούς. (Dubey, et al., 2019)

2. *Προσαρμοσμένη συμμόρφωση με κοινές κανονιστικές απαιτήσεις*
   Όλοι οι οργανισμοί εντός της κοινότητας ακολουθούν παρόμοια πρότυπα (π.χ. GDPR, HIPAA, ISO), καθιστώντας πιο εύκολη τη συμμόρφωση μέσω πχ. σχεδίασης κατάλληλων πολιτικών ασφαλείας. (Alghofaili, Albattah , Alrajeh, Rassam, & Al-rimy, 2021)

3. *Καλύτερος έλεγχος σε σχέση με το δημόσιο νέφος*
   Αν και δεν παρέχει τον απόλυτο έλεγχο του ιδιωτικού νέφους, το κοινοτικό νέφος είναι

πολύ πιο ελεγχόμενο από τις δημόσιες λύσεις, π.χ. μέσω ανάπτυξης και χρήσης μηχανισμών ελέγχου εικονικών μηχανών (virtual machines – VMs) και ελέγχου πρόσβασης στην κοινόχρηστη υποδομή. (Dubey, et al., 2019)

4. *Κοινή διαχείριση και εξειδίκευση*
   Οι οργανισμοί μπορούν να συνεργαστούν στην τεχνική και λειτουργική διαχείριση του νέφους, αξιοποιώντας τις δεξιότητες όλων. Αυτή η προσέγγιση μειώνει το λειτουργικό φορτίο για κάθε συμμετέχοντα φορέα.
   (Alghofaili, Albattah , Alrajeh, Rassam, & Al-rimy, 2021)

Μειονεκτήματα

1. *Πολυπλοκότητα διακυβέρνησης (governance)*
   Πρέπει να υπάρχει συμφωνία μεταξύ των συμμετεχόντων για τους κανόνες, τις πολιτικές και τον καταμερισμό ευθυνών. (Dubey, et al., 2019)

2. *Ζητήματα ασφάλειας λόγω πολλαπλής μίσθωσης*
   Παρά τον περιορισμένο αριθμό συμμετεχόντων, παραμένει το ζήτημα της κοινής χρήσης πόρων — που μπορεί να οδηγήσει σε παραβιάσεις. (Alghofaili, Albattah , Alrajeh, Rassam, & Al-rimy, 2021)

3. *Περιορισμένη κλιμακωσιμότητα*
   Επειδή το κοινοτικό νέφος εξυπηρετεί συγκεκριμένη ομάδα χρηστών, η δυνατότητα γρήγορης επέκτασης είναι πιο περιορισμένη απ' ό,τι στο δημόσιο νέφος. Συνεπώς, επισημαίνεται η ανάγκη για προσεκτικό σχεδιασμό πόρων από την αρχή, ο οποίος να λαμβάνει υπόψιν και πιθανή αύξηση των αναγκών (για πόρους) στο μέλλον.
   (Alghofaili, Albattah , Alrajeh, Rassam, & Al-rimy, 2021)

4. *Ανάγκη για κοινό επίπεδο τεχνογνωσίας*
   Όλοι οι συμμετέχοντες οργανισμοί πρέπει να διαθέτουν τις βασικές τεχνικές δυνατότητες για να υποστηρίξουν τη λειτουργία του κοινού νέφους.

2.1.3.5 Πολυνεφικό Περιβάλλον (Multi-Cloud Environment)

Το πολυνεφικό περιβάλλον είναι μια στρατηγική υιοθέτησης νέφους στην οποία ένας οργανισμός χρησιμοποιεί δύο ή περισσότερους παρόχους νέφους — συνήθως δημόσιους — για να καλύψει διαφορετικές ανάγκες ή εφαρμογές. Αυτό σημαίνει ότι αντί να βασίζεται σε έναν πάροχο (όπως μόνο AWS ή μόνο Azure), ο οργανισμός επιλέγει να κατανείμει τα φορτία του σε πολλαπλά νέφη, συχνά για λόγους ευελιξίας, εξειδίκευσης και ανθεκτικότητας (πχ. αν υπάρχει πρόβλημα με ένα νέφος να μεταφέρονται φόρτοι εργασίας στο άλλο).

Το πολυνέφος (multi-cloud) δεν είναι μοντέλο ανάπτυξης με την έννοια της αρχιτεκτονικής (όπως το δημόσιο, ιδιωτικό, κοινοτικό ή υβριδικό νέφος), αλλά μια στρατηγική διαχείρισης πολλών

παρόχων νέφους ταυτόχρονα. Είναι ιδανικό για μεγάλες επιχειρήσεις ή οργανισμούς που θέλουν να ελαχιστοποιήσουν τους κινδύνους εξάρτησης από έναν πάροχο (κλείδωμα σε πάροχο) (vendor lock-in), να βελτιστοποιήσουν το κόστος και να εξασφαλίσουν επιχειρησιακή συνέχεια.

Πλεονεκτήματα

1. *Αποφυγή εξάρτησης από έναν πάροχο (vendor lock-in)*
   Επιτρέπει σε οργανισμούς να μην εξαρτώνται από τις τιμές, τις πολιτικές και τις τεχνολογίες ενός μόνο παρόχου. Εκτός από την επιχειρησιακή εξάρτηση, αυξάνεται η ευελιξία, ιδιαίτερα σε IoT συστήματα. (Ahmad, Rasool, Javed, Baker, & Jalil, 2021)

2. *Εξειδίκευση ανά υπηρεσία*
   Κάθε πάροχος νέφους υπερέχει σε διαφορετικούς τομείς (π.χ. Google Cloud για AI, AWS για serverless, Azure για Microsoft integration), επιτρέποντας τη χρήση των καλύτερων υπηρεσιών για τη σύνθεση και διάταξη/τροφοδοσία εφαρμογών. Επιπλέον, είναι δυνατή η χρήση διαφορετικών παρόχων για διαφορετικούς φόρτους εργασίας. (Alghofaili, Albattah , Alrajeh, Rassam, & Al-rimy, 2021)

3. *Ανθεκτικότητα και συνέχεια υπηρεσιών (resilience)*
   Αν παρουσιαστεί βλάβη σε έναν πάροχο, οι κρίσιμες εφαρμογές μπορούν να λειτουργήσουν από εναλλακτική υποδομή.

4. *Βελτιστοποίηση κόστους*
   Οι οργανισμοί μπορούν να συγκρίνουν τιμές και να χρησιμοποιούν τον πιο οικονομικό πάροχο για κάθε υπηρεσία. Ειδικότερα, το πολυνέφος ενδείκνυται για μικροβελτιστοποίηση κόστους ανά εφαρμογή ή γεωγραφία.
   (Ahmad, Rasool, Javed, Baker, & Jalil, 2021)

Μειονεκτήματα

1. *Πολυπλοκότητα διαχείρισης και ενοποίησης*
   Η χρήση διαφορετικών APIs, εργαλείων διαχείρισης και αρχιτεκτονικών μπορεί να οδηγήσει σε μεγάλη πολυπλοκότητα. Επιπλέον, υπάρχει δυσκολία παρακολούθησης και ενοποίησης των ετερογενών πλατφορμών νέφους. (Alghofaili, Albattah , Alrajeh, Rassam, & Al-rimy, 2021)

2. *Αυξημένες απαιτήσεις τεχνογνωσίας*
   Το IT προσωπικό πρέπει να γνωρίζει τα συστήματα κάθε παρόχου και να προσαρμόζει τις διαδικασίες ανάλογα. Αυτό οδηγεί στην ανάγκη δημιουργίας ομάδων διαχείρισης κατά μήκος πλατφορμών νέφους (cross-platform cloud administration teams). (Ahmad, Rasool, Javed, Baker, & Jalil, 2021)

3. *Κίνδυνοι ασφάλειας και συμμόρφωσης*
   Η διασπορά δεδομένων σε πολλαπλούς παρόχους δημιουργεί προκλήσεις στη διαχείριση δικαιωμάτων, στην κρυπτογράφηση και στην κανονιστική συμμόρφωση. Μια ιδανική λύση (αν και δύσκολο να επιτευχθεί) σε αυτές τις προκλήσεις μπορεί να έρθει μέσω της εφαρμογής ενοποιημένου ελέγχου πρόσβασης και ενιαίας πολιτικής ασφάλειας. (Alghofaili, Albattah , Alrajeh, Rassam, & Al-rimy, 2021)

Στη συνέχεια, συνοψίζουμε και συγκρίνουμε τα 4+1 μοντέλα ανάπτυξης νέφους βάσει 5 κριτηρίων στο πλαίσιο του περιεχομένου του ακόλουθου πίνακα.

Πίνακας 1: Συγκριτικός Πίνακας Μοντέλων Ανάπτυξης Νέφους

Μοντέλο	**Έλεγχος Δεδομένων**	**Κόστος**	**Κλιμακωσιμότητα**	**Ασφάλεια**	**Καταλληλότητα**
Δημόσιο Νέφος	Χαμηλός (από τον χρήστη)	Χαμηλό (pay-as-you-go)	Πολύ υψηλή	Μέτρια (multi-tenancy)	Startups, γρήγορη ανάπτυξη
Ιδιωτικό Νέφος	Υψηλός (πλήρης οργανωτικός έλεγχος)	Υψηλό (λόγω επένδυσης & διαχείρισης υποδομών)	Περιορισμένη (εξαρτάται από εξοπλισμό)	Υψηλή (τοπική διαχείριση)	Οργανισμοί με κρίσιμα δεδομένα
Υβριδικό Νέφος	Μικτός (ιδιωτικός για κρίσιμα, δημόσιος για άλλα)	Ισορροπημένο, ανάλογα με χρήση	Υψηλή (μέσω δημόσιου νέφους)	Ισορροπία μεταξύ δημόσιου & ιδιωτικού νέφους	Επιχειρήσεις με μικτές ανάγκες
Κοινοτικό Νέφος	Μοιρασμένος εντός της κοινότητας	Μέτριο (μοιράζεται ανά οργανισμό)	Μέτρια	Υψηλή εντός της κοινότητας	Φορείς με κοινές απαιτήσεις (π.χ. πανεπιστήμια)
Πολυνέφος	Διαφορετικός ανά πάροχο – απαιτεί ομογενοποίηση	Μεταβλητό – εξαρτάται από τους παρόχους και τη χρήση	Υψηλή – δυνατότητα επιλογής παρόχου	Μεταβλητή – απαιτεί κοινές πολιτικές	Μεγάλες επιχειρήσεις, παγκόσμιες εφαρμογές

2.2 Δοχειοποίηση και Docker

Η δοχειοποίηση (containerization) είναι μια τεχνολογία ελαφριάς απομόνωσης (lightweight isolation) που επιτρέπει την εκτέλεση λογισμικού με όλα τα εξαρτώμενα συστατικά του σε ανεξάρτητα, απομονωμένα περιβάλλοντα. Ένα δοχείο (container) περιέχει τις απαραίτητες βιβλιοθήκες, τις ρυθμίσεις και τα εκτελέσιμα αρχεία μιας εφαρμογής, παρέχοντας μια αναπαραγώγιμη και φορητή λύση ανεξάρτητα από την υποδομή. Αυτή η απομόνωση βασίζεται σε λειτουργίες του πυρήνα του Linux (Linux kernel) όπως χώροι ονομάτων (namespaces) και cgroups. (Souppaya et al., 2017) (Boettiger, 2014)
Η κύρια διαφορά μεταξύ δοχείων και παραδοσιακών εικονικών μηχανών (VMs) έγκειται στον τρόπο λειτουργίας και διαχείρισης των πόρων. Οι εικονικές μηχανές βασίζονται στην πλήρη

εικονικοποίηση, όπου κάθε στιγμιότυπο (εικονικής μηχανής) περιλαμβάνει το δικό του λειτουργικό σύστημα γεγονός που οδηγεί σε αυξημένη κατανάλωση πόρων. Αντίθετα, τα δοχεία μοιράζονται τον ίδιο πυρήνα του συστήματος φιλοξενίας (host), καταναλώνοντας λιγότερους πόρους και επιτυγχάνοντας ταχύτερη εκκίνηση και ευκολότερη διαχείριση.

Ο κύκλος ζωής ανάπτυξης λογισμικού (SDLC) περιγράφει τα στάδια ανάπτυξης του λογισμικού από την ανάλυση απαιτήσεων έως τη συντήρηση και αναβάθμισή του, διασφαλίζοντας την οργανωμένη και ποιοτική του ανάπτυξη του. Παράλληλα, το DevOps αποτελεί μεθοδολογία που γεφυρώνει τις ομάδες ανάπτυξης και λειτουργίας, εστιάζοντας στην αυτοματοποίηση διαδικασιών και στην υλοποίηση αγωγών συνεχούς ολοκλήρωσης και παράδοσης (CI/CD).

Με βάση τους παραπάνω δύο ορισμούς, η δοχειοποίηση παρέχει πολλαπλά πλεονεκτήματα στον κύκλο ζωής ανάπτυξης λογισμικού (SDLC) και ειδικότερα στις DevOps διαδικασίες. Τα δοχεία προάγουν τη συνεπή και αξιόπιστη ανάπτυξη, επιτρέποντας την υλοποίηση αγωγών (pipelines) CI/CD που βασίζονται σε σταθερά περιβάλλοντα. Επιπλέον, προωθούν την αρχιτεκτονική μικροϋπηρεσιών, επιτρέποντας την ανεξάρτητη ανάπτυξη, δοκιμή και κλιμάκωση της κάθε υπηρεσίας που απαρτίζει μια εφαρμογή.

Το Docker αποτελεί την πιο διαδεδομένη πλατφόρμα δοχειοποίησης, η οποία κυκλοφόρησε το 2013 και γρήγορα καθιερώθηκε ως το πρότυπο (standard) στον χώρο. Παρέχει εργαλεία για την κατασκευή εικόνων δοχείων (container images) μέσω Dockerfiles και τη διαχείρισή τους, τη δημιουργία και εκτέλεση δοχείων μέσω των εικόνων τους, και την ενσωμάτωσή των δοχείων σε αγωγούς ανάπτυξης. Ο Boettiger (2015) επισημαίνει ότι το Docker προσφέρει ουσιαστική υποστήριξη για αναπαραγωγίσιμη επιστήμη (reproducible science), καθώς επιτρέπει το πακετάρισμα πειραμάτων σε φορητές εικόνες (δοχείων).

Το Docker υποστηρίζει στρωματωμένα αρχεία συστημάτων (layered file systems), αποθηκευτικούς χώρους (volumes) για επίμονη αποθήκευση (persistent storage), δικτύωση (Docker networks), αποθετήρια εικόνων δοχείων (container image registries) όπως το Docker Hub, και την ενοποίηση με εργαλεία ενορχήστρωσης δοχείων (container orchestration) όπως το Kubernetes. Επιπλέον, εφαρμόζει πρακτικές ασφαλείας, όπως image signing (υπογραφή εικόνων), minimal base images (ελάχιστες βάσεις εικόνων), SELinux/AppArmor profiles (προφίλ ασφαλείας) και rootless containers (δηλ. δοχεία που εκτελούνται χωρίς δικαιώματα ριζικού χρήστη).

Ο παραπάτω πίνακας συγκρίνει τα 2 είδη εικονικοποίησης (δοχειοποίηση και φυσική εικονικοποίηση) με βάση 5 κριτήρια, υποδεικνύοντας την υπεροχή της δοχειοποίησης.

Πίνακας 2: Σύγκριση Containers και Virtual Machines

Ιδιότητα/κριτήριο	**Containers/Δοχεία**	**Virtual Machines/Εικονικές Μηχανές**
OS	Μοιράζονται Host Kernel	Δικό τους Guest OS
Εκκίνηση	Πολύ γρήγορη (ms–sec)	Αργή (30–60 sec)
Χρήση Πόρων	Πολύ χαμηλή	Υψηλή

Απομόνωση	Ελαφριά, επαρκής	Πλήρης
Portability/Φορητότητα	Υψηλή	Περιορισμένη

2.2.1 Τεχνολογική Βάση Docker και Λειτουργικά Χαρακτηριστικά

Το Docker αποτελεί μια ελαφριά και αποδοτική τεχνολογική λύση που βασίζεται σε δοχεία και έχει επαναπροσδιορίσει τον τρόπο με τον οποίο σχεδιάζονται, αναπτύσσονται και διαχειρίζονται εφαρμογές σε σύγχρονα περιβάλλοντα υποδομών και νέφους. Η αρχιτεκτονική του και τα επιμέρους στοιχεία του συνεργάζονται με σκοπό την αυτοματοποίηση, την επαναχρησιμοποίηση, την ασφάλεια και την απόδοση των δοχειοποιημένων εφαρμογών.

Η τεχνολογική βάση του Docker στηρίζεται στη δομή client-server και συνδυάζει επιμέρους υποσυστήματα, όπως το Docker Engine, τα images, τα containers και τα registries, ώστε να παρέχει μια ελαφριά και φορητή πλατφόρμα για την εκτέλεση εφαρμογών. Η αρχιτεκτονική του χαρακτηρίζεται από τη χρήση τεχνολογιών του πυρήνα του Linux, όπως namespaces και cgroups, για την απομόνωση και τον έλεγχο των πόρων κάθε δοχείου (Rad, Bhatti, & Ahmadi, 2017).

Docker Engine και Client-Server Μοντέλο

Η Docker Engine αποτελεί τη βασική μηχανή εκτέλεσης του Docker και είναι υπεύθυνη για όλες τις λειτουργίες που σχετίζονται με τη δημιουργία, διαχείριση και εκτέλεση δοχείων. Η αρχιτεκτονική του βασίζεται στο μοντέλο client-server, το οποίο διαχωρίζει τη λειτουργία μεταξύ της διεπαφής εντολών (client) και της υπηρεσίας που εκτελεί τις εντολές (server ή daemon).

Ο Docker daemon (γνωστός και ως dockerd) είναι μια μόνιμη υπηρεσία που εκτελείται στο παρασκήνιο και είναι υπεύθυνη για την αλληλεπίδραση με τον πυρήνα του λειτουργικού συστήματος. Ο daemon διαχειρίζεται τα δοχεία, δημιουργεί και αποθηκεύει Docker εικόνες (δοχείων), ρυθμίζει τα δίκτυα και τους αποθηκευτικούς όγκους (volumes) και επικοινωνεί με εξωτερικά Docker registries, όπως το Docker Hub. Έχει πρόσβαση σε κρίσιμες λειτουργίες του συστήματος (όπως namespaces και cgroups) και μέσω αυτών προσφέρει την απομόνωση και τον έλεγχο των containers.

Ο Docker client, από την άλλη πλευρά, είναι το εργαλείο γραμμής εντολών (docker) που χρησιμοποιεί ο χρήστης για να αλληλεπιδράσει με το Docker. Όταν δίνεται μια εντολή από τον χρήστη – για παράδειγμα docker run, docker build, ή docker stop – ο client στέλνει το αντίστοιχο αίτημα στο Docker daemon μέσω ενός RESTful API. Ο daemon εκτελεί την εντολή και επιστρέφει την απάντηση στον client. Αυτή η αρχιτεκτονική προσφέρει ευελιξία, αφού ο client μπορεί να τρέχει είτε τοπικά (στον ίδιο υπολογιστή με τον daemon), είτε απομακρυσμένα, επιτρέποντας διαχείριση δοχείων σε πολλαπλά μηχανήματα ή σε συστάδες (clusters).

Το Docker REST API λειτουργεί ως ενδιάμεσος μεταξύ του client και του daemon και υποστηρίζει την ενσωμάτωση του Docker με εξωτερικά εργαλεία, scripts ή συστήματα αυτοματισμού. Επιπλέον, δίνει τη δυνατότητα να διαχειρίζεται κανείς το Docker εξ αποστάσεως, πράγμα απαραίτητο σε περιβάλλοντα νέφους ή σε αγωγούς DevOps.

Συνολικά, το μοντέλο client-server του Docker είναι σχεδιασμένο ώστε να διαχωρίζει τη διεπαφή χρήστη από τις διαδικασίες χαμηλού επιπέδου, προσφέροντας αρχιτεκτονική ευελιξία, επεκτασιμότητα και αυτοματοποίηση, χαρακτηριστικά απαραίτητα για σύγχρονες πρακτικές ανάπτυξης και διαχείρισης λογισμικού. Η αποτελεσματικότητα αυτού του μοντέλου έχει επιβεβαιωθεί και από τη βιβλιογραφία, καθώς περιγράφεται ως ένας από τους βασικούς λόγους επιτυχίας και ευρείας υιοθέτησης της πλατφόρμας Docker (Rad, Bhatti, & Ahmadi, 2017).

Docker Images

Τα Docker images είναι τα βασικά δομικά στοιχεία της τεχνολογίας του Docker. Πρόκειται για στατικά, πολυεπίπεδα πρότυπα εκτελέσιμου λογισμικού τα οποία περιλαμβάνουν όλα τα απαραίτητα στοιχεία για την εκτέλεση μιας εφαρμογής ή ενός συστατικού της: το λειτουργικό σύστημα βάσης (π.χ. Ubuntu, Alpine), τις απαραίτητες βιβλιοθήκες και πακέτα εξαρτήσεων, τις μεταβλητές περιβάλλοντος, ρυθμίσεις διαμόρφωσης, και τέλος τον εκτελέσιμο κώδικα της εφαρμογής.

Το ιδιαίτερο χαρακτηριστικό των Docker images είναι η στρωματοποιημένη (layered) αρχιτεκτονική τους. Κάθε image αποτελείται από πολλαπλά στρώματα (layers), τα οποία δημιουργούνται με βάση τις εντολές που περιλαμβάνονται στο Dockerfile (π.χ. FROM, COPY, RUN). Αυτή η πολυεπίπεδη δομή επιτρέπει την επαναχρησιμοποίηση κοινών στοιχείων ανάμεσα σε διαφορετικά images, μειώνοντας δραστικά τον αποθηκευτικό χώρο και επιταχύνοντας τη δημιουργία νέων εικόνων. Για παράδειγμα, δύο διαφορετικά images που βασίζονται στο ubuntu:20.04 μοιράζονται το ίδιο base layer (βασικό στρώμα), και διαφοροποιούνται μόνο στα επιπλέον επίπεδα. Επιπλέον, η πολυεπίπεδη δομή επιτρέπει την αποδοτική ανανέωση υπαρχόντων images: όταν αλλάξει κάποιο στοιχείο, δεν χρειάζεται να ξαναδημιουργηθεί όλο το image από την αρχή· επανεκτελούνται μόνο τα απαραίτητα στρώματα από το σημείο της αλλαγής και έπειτα.

Τα Docker images είναι read-only· δεν αλλάζουν κατά την εκτέλεση του δοχείου. Αν κάποια διεργασία θέλει να τροποποιήσει το σύστημα αρχείων (filesystem), τότε οι αλλαγές καταγράφονται στο εγγράψιμο (writable) layer του δοχείου (container) που προκύπτει από το image. Αυτή η τεχνική βασίζεται στη λογική του Copy-on-Write, που προσφέρει υψηλή απόδοση και αποδοτική χρήση των πόρων.

Οι εικόνες μπορούν να αποθηκευτούν είτε τοπικά είτε σε αποθετήρια (registries), όπως το Docker Hub, επιτρέποντας την εύκολη διανομή και κοινή χρήση τους. Συνολικά, τα Docker images προσφέρουν φορητότητα, συνέπεια και ταχύτητα ανάπτυξης, καθώς μπορούν να εκτελεστούν ακριβώς με τον ίδιο τρόπο σε διαφορετικά περιβάλλοντα (ανάπτυξης, δοκιμών, παραγωγής), διατηρώντας όλες τις εξαρτήσεις ενσωματωμένες (Rad, Bhatti, & Ahmadi, 2017).

Dockerfile

Το αρχείο αυτό περιέχει **δηλωτικές εντολές**, οι οποίες εκτελούνται διαδοχικά από την Docker Engine κατά τη διαδικασία του docker build (δηλ. της εντολής που χρησιμοποιείται για την

κατασκευή των images). Κάθε εντολή δημιουργεί ένα νέο layer στην εικόνα προς κατασκευή και προσθέτει λειτουργικότητα ή περιεχόμενο.

Ενδεικτικές εντολές Dockerfile:

- FROM: Ορίζει το base image (π.χ. FROM ubuntu:20.04)
- COPY ή ADD: Αντιγράφει αρχεία στο σύστημα αρχείων της εικόνας
- RUN: Εκτελεί εντολές μέσα στο image κατά την κατασκευή της (π.χ. για την εγκατάσταση πακέτων/βιβλιοθηκών)
- ENV: Δηλώνει μεταβλητές περιβάλλοντος
- EXPOSE: Δηλώνει θύρες που θα εκτίθενται κατά την εκτέλεση των δοχείων της εικόνας
- CMD ή ENTRYPOINT: Ορίζει ποια εντολή θα εκτελείται όταν ξεκινά το κάθε δοχείο της εικόνας

Η χρήση Dockerfile προσφέρει:

- *Επαναληψιμότητα*: Κάθε φορά που δημιουργείται image από το ίδιο Dockerfile, το αποτέλεσμα είναι προβλέψιμο.
- *Αυτοματισμό*: Μπορεί να ενσωματωθεί σε αγωγούς CI/CD.
- *Φορητότητα*: Το Dockerfile είναι συμβατό σε όλα τα περιβάλλοντα Docker, ανεξαρτήτως λειτουργικού συστήματος (OS) ή παρόχου νέφους (cloud provider).

Συνολικά, το Dockerfile καθιστά τη διαδικασία δημιουργίας δοχείων αυτοματοποιημένη, διαφανή και προγραμματιστικά ελεγχόμενη, συμβάλλοντας στη σταθερότητα και την επεκτασιμότητα λογισμικών συστημάτων (Verma, Pandey, & Gupta, 2023).

Docker Containers

Το container (δοχείο) είναι το εκτελέσιμο αποτέλεσμα ενός image. Κατά την εκκίνηση ενός container, προστίθεται ένα εγγράψιμο layer πάνω στο υπάρχον image. Το container περιέχει όλα τα απαραίτητα στοιχεία για την εκτέλεση της εφαρμογής ή ενός συστατικού της, αλλά είναι απομονωμένο σε επίπεδο διεργασιών, δικτύου και αρχείων μέσω τεχνικών όπως namespaces και cgroups (Rad, Bhatti, & Ahmadi, 2017). Από την οπτική του ίδιου του container, το περιβάλλον εκτέλεσης φαίνεται πλήρες: διαθέτει το δικό του filesystem, τοπικό δίκτυο, διεργασίες και πόρους (resources). Ωστόσο, στην πραγματικότητα, όλα αυτά είναι εικονικά και περιορίζονται αυστηρά από τον μηχάνημα φιλοξενίας (host) μέσω τεχνικών απομόνωσης. Αυτό είναι και το βασικό πλεονέκτημα της δοχειοποίησης: ο συνδυασμός πλήρους λειτουργικότητας και ελάχιστης κατανάλωσης πόρων.

Τα containers εκκινούν σχεδόν ακαριαία (μέσα σε δευτερόλεπτα) και καταλαμβάνουν πολύ λιγότερους πόρους σε σχέση με τις παραδοσιακές εικονικές μηχανές. Επιπλέον, παρέχουν υψηλή

συνέπεια και επαναληψιμότητα, καθώς ο κώδικας που εκτελείται σε έναν container είναι απομονωμένος από εξωτερικούς παράγοντες και συνοδεύεται από όλες τις εξαρτήσεις του. Αυτή η προσέγγιση καθιστά τα Docker δοχεία ιδανικά για σύγχρονες πρακτικές ανάπτυξης λογισμικού, για αγωγούς DevOps, για υλοποίηση μικρο-υπηρεσιών, αλλά και για χρήση σε cloud-native περιβάλλοντα (Rad, Bhatti, & Ahmadi, 2017) (Verma, Pandey, & Gupta, 2023).

Docker Registry & Hub

Ο μηχανισμός των Docker Registries αποτελεί κρίσιμο κομμάτι της λειτουργίας του Docker, καθώς επιτρέπει τη διάθεση, αποθήκευση και διανομή εικόνων δοχείων (container images) σε τοπικά ή απομακρυσμένα περιβάλλοντα. Με απλά λόγια, ένα registry λειτουργεί ως "μητρώο", όπου οι προγραμματιστές μπορούν να ανεβάσουν (*push*) ή να κατεβάσουν (*pull*) Docker images, με στόχο τη γρήγορη επαναχρησιμοποίησή τους από τις ίδιες ή άλλες ομάδες, περιβάλλοντα ή μηχανές παραγωγής.

Το πιο ευρέως χρησιμοποιούμενο registry είναι το Docker Hub, μια δημόσια υπηρεσία που παρέχει χιλιάδες έτοιμα images από επίσημες διανομές λειτουργικών συστημάτων (όπως ubuntu, alpine, nginx, mysql) αλλά και εικόνες από ανεξάρτητους προγραμματιστές ή εταιρείες. Ο χρήστης μπορεί να δημιουργήσει είτε δημόσια (public) αποθετήρια (repositories), τα οποία είναι προσβάσιμα από όλους, είτε ιδιωτικά (private) αποθετήρια για μεγαλύτερη ασφάλεια και έλεγχο πρόσβασης.

Πέρα από το Docker Hub, οργανισμοί και επιχειρήσεις μπορούν να χρησιμοποιούν ιδιωτικά μητρώα (registries), όπως:

- το Harbor (ανοιχτού κώδικα),
- το Amazon Elastic Container Registry (ECR),
- το Google Artifact Registry, ή
- custom λύσεις με χρήση του εργαλείου registry του ίδιου του Docker.

Η ύπαρξη ενός registry διευκολύνει τη συνεχή ενσωμάτωση (CI) και τη συνεχή παράδοση (CD) του λογισμικού, αφού επιτρέπει την αυτόματη δημιουργία, αποθήκευση και διανομή νέων images (δηλ. νέων εκδόσεων εφαρμογής ή συστατικού της) από αγωγούς που βασίζονται σε Jenkins, GitLab ή άλλα εργαλεία/πλατφόρμες CI/CD. Ένας αγωγός κατασκευής (build pipeline) μπορεί, για παράδειγμα, να χτίζει νέο Docker image κάθε φορά που γίνεται αλλαγή στον κώδικα και να το "ανεβάζει" σε αποθετήριο στο registry, απ' όπου μπορούν άμεσα να το χρησιμοποιήσουν τα περιβάλλοντα δοκιμών ή παραγωγής.

Από άποψη ασφάλειας, τα registries παρέχουν δυνατότητες ελέγχου ταυτότητας (authentication), εξουσιοδότησης (authorization) και επαλήθευσης ψηφιακής υπογραφής των images. Επιπλέον, πολλά υποστηρίζουν vulnerability scanning (έλεγχο ευπαθειών) ώστε να εντοπίζονται προβλήματα ασφαλείας εντός των layers ενός image.

Συνοψίζοντας, τα Docker Registries αποτελούν αναπόσπαστο μέρος της αρχιτεκτονικής του Docker και του σύγχρονου DevOps οικοσυστήματος. Υποστηρίζουν τη φορητότητα, την επεκτασιμότητα και την ασφάλεια στην ανάπτυξη εφαρμογών και επιτρέπουν τη γρήγορη διανομή

και αυτοματοποιημένη διαχείριση δοχείων σε κάθε στάδιο του κύκλου ζωής των εφαρμογών (Verma, Pandey, & Gupta, 2023) (Rad, Bhatti, & Ahmadi, 2017).

Docker Compose

Το Docker Compose είναι ένα εργαλείο που παρέχει στο Docker τη δυνατότητα διαχείρισης και εκτέλεσης πολλαπλών containers ως μέρος μιας ενιαίας, πολυσύνθετης εφαρμογής. Αντί να εκτελείται κάθε container μεμονωμένα και με χειροκίνητες παραμέτρους, το Docker Compose επιτρέπει στον χρήστη να ορίσει ολόκληρη την εφαρμογή του – με όλες τις (μικρο)υπηρεσίες της – μέσα από ένα αρχείο διαμόρφωσης τύπου YAML (docker-compose.yml).

Σε αυτό το αρχείο, ορίζονται όλες οι επιμέρους υπηρεσίες (services)/συστατικά της εφαρμογής (π.χ. frontend, βάση δεδομένων, reverse proxy). Κάθε υπηρεσία βασίζεται σε ένα συγκεκριμένο image και μπορεί να εκτελεί ένα η περισσοτερα container (instances) αυτού του image. Με αυτό τον τρόπο, το service λειτουργεί ως λογικός ορισμός ενός υπηρεσιο-κεντρικού συστατικού της εφαρμογής, ενώ τα containers αποτελούν τις πραγματικές εκτελέσεις του. Εντός του αρχείου, ορίζονται επίσης τα δίκτυα που θα διασυνδέουν τις υπηρεσίες, τα volumes που θα προσαρτούνται σε αυτές για αποθήκευση δεδομένων, οι μεταβλητές περιβάλλοντος προς χρήση με τις τιμές τους (environment variables), οι θύρες (ports) που θα εκτεθούν κ.ά. Έπειτα, με μία και μόνο εντολή (docker-compose up), ολόκληρο το περιβάλλον στήνεται αυτόματα ώστε να γίνει διαθέσιμη όλη η εφαρμογή με τη μορφή μιας ενορχήστρωσης δοχείων.

Αυτό το μοντέλο είναι ιδιαίτερα σημαντικό για αρχιτεκτονικές μικροϋπηρεσιών (microservices), όπου κάθε λειτουργική μονάδα (μικρο-υπηρεσία) της εφαρμογής "τρέχει" σε δικό της container αλλά απαιτεί συνεργασία με άλλες μονάδες. Το Compose διευκολύνει την ανάπτυξη, τη δοκιμή και την αναπαραγωγή σύνθετων συστημάτων, είτε σε τοπικό περιβάλλον είτε σε ενδιάμεσα περιβάλλοντα (staging). Παρέχει επίσης την ευχέρεια γρήγορης εκκίνησης και τερματισμού όλων των containers που απαρτίζουν το σύστημα με απλές εντολές (up, down, restart, logs κ.λπ.), εξασφαλίζοντας συνέπεια, ευκολία και επαναληψιμότητα στη διαχείριση των περιβαλλόντων.

Η δύναμη του Docker Compose έγκειται κυρίως στην απλοποίηση της διαδικασίας ανάπτυξης και στη δυνατότητα να λειτουργεί ως **εργαλείο ενορχήστρωσης** για containers. Σε συνδυασμό με άλλα στοιχεία του Docker (όπως volumes, networks και env files), το Compose επιτρέπει τη δημιουργία πολύπλοκων δοχειοποιημένων εφαρμογών με ελάχιστο κόπο και μέγιστη διαφάνεια.

Η συμβολή του Docker Compose στη σύγχρονη ανάπτυξη λογισμικού είναι καθοριστική, καθώς προσφέρει έναν απλό αλλά ισχυρό τρόπο διαχείρισης πολλαπλών δοχείων σε επίπεδο εφαρμογής, χωρίς να απαιτεί την πολυπλοκότητα εργαλείων ενορχήστρωσης (δοχείων) όπως το Kubernetes. Η ενορχήστρωση που πραγματοποιεί περιορίζεται σε τοπικό επίπεδο, δηλαδή σε ένα μόνο μηχάνημα φιλοξενίας, γεγονός που το καθιστά ιδιαίτερα χρήσιμο για μικρές και μεσαίας κλίμακας εφαρμογές ή για το στάδιο της τοπικής ανάπτυξης. Αντίθετα, σε περιβάλλοντα παραγωγής μεγάλης κλίμακας, προτιμάται η χρήση του Kubernetes, το οποίο επιτρέπει την ενορχήστρωση containers σε συστάδες πολλαπλών μηχανημάτων (clusters), εξασφαλίζοντας αυτοματοποιημένη κλιμάκωση, ανθεκτικότητα και υψηλή διαθεσιμότητα (Verma, Pandey, & Gupta, 2023) (Rad, Bhatti, &

Ahmadi, 2017).

2.2.2 Τεχνολογίες Απομόνωσης στο Docker: Linux Namespaces

Η δοχειοποίηση βασίζεται στην απομόνωση πόρων – δηλαδή κάθε δοχείο πρέπει να "πιστεύει" ότι λειτουργεί ανεξάρτητα, σαν να είναι μόνο του στο σύστημα. Αυτή η απομόνωση επιτυγχάνεται κυρίως με τη χρήση Linux namespaces, τα οποία αποτελούν μηχανισμούς του πυρήνα του Linux.

Τα namespaces περιορίζουν την "ορατότητα" ενός container στο σύστημα, έτσι ώστε κάθε container να έχει το δικό του εικονικό περιβάλλον σε επίπεδο διεργασιών, δικτύου, επικοινωνίας και συστήματος αρχείων. Το Docker χρησιμοποιεί διαφορετικά είδη namespaces για να προσφέρει απομόνωση μεταξύ των containers:

- **PID namespace** για απομόνωση διεργασιών
- **NET namespace** για απομόνωση δικτύου
- **IPC namespace** για διαχείριση επικοινωνίας μεταξύ διεργασιών,
- **MNT namespace** για διαχωρισμό του filesystem.

1. **PID Namespace (Process ID Namespace)**

Ο μηχανισμός PID namespace του Linux, τον οποίο αξιοποιεί το Docker, είναι υπεύθυνος για την απομόνωση των διεργασιών ενός container από το υπόλοιπο σύστημα. Κάθε container που δημιουργείται με χρήση PID namespace αποκτά το δικό του ανεξάρτητο δέντρο διεργασιών, σαν να είναι ένας ξεχωριστός εικονικός υπολογιστής. Αυτό σημαίνει ότι οι διεργασίες που “τρέχουν” μέσα σε έναν container είναι ορατές μόνο εντός του container – ο container δεν μπορεί να δει ή να επηρεάσει τις διεργασίες που εκτελούνται σε άλλους containers ή στο host σύστημα. Για παράδειγμα, η πρώτη διεργασία που εκτελείται μέσα σε ένα container έχει αναγνωριστικό PID 1 από την προοπτική του container, παρόλο που στο πραγματικό host σύστημα μπορεί να έχει διαφορετικό αριθμό διεργασίας. Αυτή η σχετική "αυταπάτη" δημιουργεί πλήρη απομόνωση σε επίπεδο διεργασιών και προσφέρει σημαντικό πλεονέκτημα ασφάλειας, καθώς αποτρέπει την αλληλεπίδραση μεταξύ διεργασιών διαφορετικών containers – όπως το να σταλούν σήματα kill από έναν container σε άλλον (οδηγώντας στον πρόωρο και ανεπιθύμητο τερματισμό ενός container από έναν άλλο). Μέσω αυτής της τεχνικής, το Docker επιτυγχάνει αποδοτικό και ασφαλές sandboxing για κάθε container, χωρίς να απαιτείται ξεχωριστό λειτουργικό σύστημα για κάθε εφαρμογή (Verma, Pandey, & Gupta, 2023) (Rad, Bhatti, & Ahmadi, 2017).

2. **NET Namespace (Network Namespace)**

Ο μηχανισμός **NET namespace** του Linux, τον οποίο αξιοποιεί το Docker, είναι υπεύθυνος για την απομόνωση των πόρων δικτύου κάθε container. Με τη χρήση αυτού του namespace, κάθε container αποκτά το δικό του εικονικό στοίβα δικτύου, η οποία περιλαμβάνει ανεξάρτητες διευθύνσεις IP, routing tables, firewall rules και network interfaces. Αυτό σημαίνει ότι τα containers μπορούν να έχουν το δικό τους "εικονικό δίκτυο", χωρίς να βλέπουν άμεσα τα δίκτυα άλλων containers ή του host. Για παράδειγμα, δύο containers μπορούν να έχουν την ίδια ιδιωτική IP διεύθυνση, καθώς βρίσκονται σε διαφορετικά network namespaces και συνεπώς δεν συγκρούονται. Το Docker αξιοποιεί αυτήν τη δυνατότητα για να δημιουργεί εικονικά δίκτυα (bridges, overlays), μέσω των οποίων τα containers επικοινωνούν με ελεγχόμενο και ασφαλή τρόπο (Verma, Pandey, & Gupta, 2023) (Rad, Bhatti, & Ahmadi, 2017).

3. IPC Namespace (Interprocess Communication)

Ο μηχανισμός IPC namespace του Linux είναι υπεύθυνος για την απομόνωση των μηχανισμών διαπροσωπικής επικοινωνίας που χρησιμοποιούν οι διεργασίες, όπως τα segments κοινής μνήμης, οι ουρές μηνυμάτων και οι σημαφόροι. Με αυτόν τον τρόπο, οι διεργασίες μέσα σε ένα container μπορούν να επικοινωνούν μεταξύ τους χρησιμοποιώντας τους παραπάνω μηχανισμούς, αλλά δεν έχουν πρόσβαση στους IPC πόρους άλλων containers ή του host. Αυτό αυξάνει το επίπεδο ασφάλειας, καθώς αποτρέπει την ακούσια ή κακόβουλη ανταλλαγή δεδομένων μέσω κοινών IPC καναλιών. Εφόσον απαιτείται συνεργασία δύο containers στον ίδιο IPC χώρο, αυτό πρέπει να οριστεί ρητά μέσω των κατάλληλων ρυθμίσεων στο Docker (Verma, Pandey, & Gupta, 2023) (Rad, Bhatti, & Ahmadi, 2017).

4. MNT Namespace (Mount Namespace)

Ο μηχανισμός **MNT namespace** (Mount namespace) επιτρέπει σε κάθε container να έχει το δικό του ανεξάρτητο σύστημα αρχείων, απομονωμένο από το υπόλοιπο σύστημα και τους άλλους containers. Με τη χρήση του, κάθε container "βλέπει" μόνο τους δικούς του καταλόγους και τα δικά του mounted filesystem paths (προσαρτημένα μονοπάτια συστήματος αρχείων), χωρίς να έχει πρόσβαση στα αρχεία του host ή σε εκείνα άλλων containers, εκτός εάν του αποδοθεί ρητά το δικαίωμα αυτό μέσω μηχανισμών, όπως τα volumes. Αυτό σημαίνει ότι ένας container μπορεί να έχει το δικό του root filesystem (/) που να περιλαμβάνει διαφορετικές βιβλιοθήκες, εκτελέσιμα αρχεία και scripts, τα οποία δεν επηρεάζουν καθόλου τον host. Επιπλέον, οι αλλαγές που γίνονται μέσα σε έναν container, όπως η προσάρτηση (mount) νέων φακέλων ή η δημιουργία αρχείων, περιορίζονται εντός του δικού του MNT namespace, προσφέροντας αυξημένη ασφάλεια και σταθερότητα στο σύστημα. Μέσα από αυτήν την απομόνωση, η δοχειοποίηση εξασφαλίζει ότι κάθε container μπορεί να λειτουργεί ως ένα αυτόνομο περιβάλλον εκτέλεσης, με πλήρη έλεγχο στο δικό του filesystem, χωρίς κίνδυνο παρεμβολής ή διαρροής δεδομένων από και προς άλλες υπηρεσίες (Verma, Pandey, & Gupta, 2023).

2.2.3 Έλεγχος Πόρων με Cgroups

Ο μηχανισμός cgroups (Control Groups) αποτελεί βασικό δομικό στοιχείο της αρχιτεκτονικής του Docker, προσφέροντας τη δυνατότητα ελέγχου και κατανομής υπολογιστικών πόρων σε κάθε container με ακρίβεια και ασφάλεια. Πρόκειται για λειτουργία του πυρήνα του Linux, η οποία επιτρέπει στον διαχειριστή του συστήματος ή στη μηχανή του Docker να καθορίζει συγκεκριμένα όρια χρήσης για κρίσιμους πόρους, όπως είναι η κεντρική μονάδα επεξεργασίας (CPU), η μνήμη RAM, το δίκτυο (network I/O) και η είσοδος/έξοδος από αποθηκευτικά μέσα (disk I/O).

Η κύρια λειτουργία των cgroups είναι να διασφαλίσει ότι κανένας container δεν μπορεί να καταναλώσει δυσανάλογο μερίδιο πόρων, θέτοντας σε κίνδυνο τη σταθερότητα του συνολικού συστήματος. Για παράδειγμα, αν ένα container έχει προγραμματιστεί να χρησιμοποιεί μόνο το 20% της CPU ή μέχρι 512MB RAM, τότε ακόμη και σε περίπτωση λάθους λογισμικού (π.χ. endless loop ή memory leak), δεν θα επηρεαστούν οι υπόλοιποι containers ή το ίδιο το host σύστημα. Αντίστοιχα, όταν πολλά δοχεία (containers) λειτουργούν ταυτόχρονα, τα cgroups διασφαλίζουν ότι καθένας χρησιμοποιεί προβλέψιμο και καθορισμένο ποσοστό των πόρων, διατηρώντας την απόδοση και την ποιότητα υπηρεσίας (QoS) σε απαιτητικά περιβάλλοντα παραγωγής.

Επιπλέον, τα cgroups υποστηρίζουν δυναμική παρακολούθηση και αναπροσαρμογή. Ο διαχειριστής μπορεί να τροποποιήσει τους περιορισμούς των πόρων «εν κινήσει» ή να χρησιμοποιήσει εργαλεία αυτοματισμού (όπως Kubernetes resource limits) για την κλιμακούμενη διαχείριση φορτίου σε πραγματικό χρόνο. Αυτή η δυνατότητα κάνει τα cgroups θεμελιώδη για συστήματα που βασίζονται σε αρχιτεκτονικές πολλαπλών μισθωτών, όπου πολλές (μικρο-)υπηρεσίες φιλοξενούνται στον ίδιο φυσικό εξυπηρετητή.

Συνολικά, ο μηχανισμός cgroups είναι αυτός που επιτρέπει στο Docker όχι μόνο να απομονώνει containers, αλλά και να διασφαλίζει δίκαιη και ελεγχόμενη χρήση πόρων, καθιστώντας το κατάλληλο για χρήση σε σε κατανεμημένα περιβάλλοντα νέφους (Verma, Pandey, & Gupta, 2023).

2.2.4 Union File System και Copy-on-Write

Ο μηχανισμός Union File System (UnionFS) σε συνδυασμό με την τεχνική Copy-on-Write (CoW) αποτελεί έναν από τους πιο σημαντικούς παράγοντες που καθιστούν τα containers του Docker ελαφριά, ευέλικτα και αποδοτικά. Το UnionFS είναι ένας τύπος συστήματος αρχείων που επιτρέπει τη συνένωση (ένωση) πολλαπλών directories (ευρετηρίων) (γνωστά ως layers) σε ένα ενιαίο εικονικό filesystem. Αυτό σημαίνει ότι ένα Docker image δεν αποτελείται από ένα ενιαίο αρχείο, αλλά από πολλαπλά στρώματα (layers), καθένα από τα οποία αντιστοιχεί σε μια διαφορετική ενέργεια στο Dockerfile (π.χ. FROM, RUN, COPY).

Η αρχιτεκτονική αυτή είναι πολυεπίπεδη και επαναχρησιμοποιήσιμη. Για παράδειγμα, πολλά διαφορετικά containers μπορεί να βασίζονται στο ίδιο base layer (π.χ. ubuntu:20.04) και να προσθέτουν τα δικά τους layers πάνω από αυτό χωρίς να δημιουργούνται πολλαπλά αντίγραφα. Αυτό εξοικονομεί σημαντικό αποθηκευτικό χώρο και μειώνει τον χρόνο λήψης (pull) ή δημιουργίας (build) νέων container images.

Η λειτουργία Copy-on-Write ενισχύει ακόμη περισσότερο την απόδοση του Docker. Όταν δημιουργείται ένα container από ένα image, όλα τα layers του image είναι read-only. Το μόνο layer που επιτρέπεται να τροποποιηθεί είναι το writable layer του container που δημιουργείται κατά την εκκίνησή του. Αν ο container χρειαστεί να τροποποιήσει ένα αρχείο, τότε το Docker αντιγράφει το αρχείο αυτό στο writable layer και εκτελεί την αλλαγή εκεί. Το αρχικό αρχείο στο read-only layer παραμένει ανέπαφο. Αυτή η τεχνική επιτρέπει στο Docker να:

- διατηρεί το image ελαφρύ και σταθερό,
- μειώνει τις επιπτώσεις των αλλαγών στον δίσκο,
- προσφέρει ταχύτητα στην εκκίνηση και αναπαραγωγή containers.

Χάρη σε αυτόν τον σχεδιασμό, το Docker μπορεί να δημιουργήσει, εκκινήσει και καταστρέψει containers με πολύ χαμηλό κόστος χρόνου και πόρων, προσφέροντας παράλληλα μεγάλη ευελιξία στην ανάπτυξη και στην επαναχρησιμοποίηση λογισμικού. Η λειτουργία αυτή είναι καθοριστική για το DevOps, τη χρήση microservices, και γενικότερα για περιβάλλοντα όπου η ταχύτητα ανάπτυξης και η συνέπεια των εκτελέσιμων περιβαλλόντων είναι κρίσιμη (Rad, Bhatti, & Ahmadi, 2017) (Verma, Pandey, & Gupta, 2023).

2.3 Kubernetes – Αρχιτεκτονική και δυνατότητες

Το Kubernetes αποτελεί σήμερα την κυρίαρχη πλατφόρμα ενορχήστρωσης δοχείων και είναι θεμελιώδες συστατικό του οικοσυστήματος του υπολογιστικού νέφους. Πρόκειται για ένα σύστημα ανοικτού κώδικα που σχεδιάστηκε με σκοπό την αυτοματοποίηση της ανάπτυξης (deployment), διαχείρισης (management) και κλιμάκωσης (scaling) δοχειοποιημένων εφαρμογών.

Η τεχνολογία αυτή γεννήθηκε ως εσωτερικό εργαλείο της Google, βασισμένο σε προηγούμενες εμπειρίες της εταιρείας με τα συστήματα Borg και Omega, τα οποία διαχειρίζονταν εκατομμύρια δοχεία καθημερινά στο εσωτερικό της. Το Kubernetes (ή αλλιώς K8s) δημοσιοποιήθηκε ως έργο ανοιχτού κώδικα το 2014, στοχεύοντας να μεταφέρει αυτή την εμπειρία και τεχνογνωσία στην ευρύτερη τεχνολογική κοινότητα.

Από το 2015, το έργο Kubernetes εντάχθηκε στο Cloud Native Computing Foundation (CNCF) — έναν ανεξάρτητο, μη κερδοσκοπικό οργανισμό που ανήκει στο Linux Foundation και έχει ως στόχο την προώθηση τεχνολογιών που ενδυναμώνουν cloud-native υποδομές. Η μεταφορά στο CNCF ήταν καθοριστική για την εξάπλωση και την ουδετερότητα του έργου, καθώς έδωσε τη δυνατότητα σε πλήθος εταιρειών (όπως RedHat, IBM, AWS, Microsoft, VMware, και πολλές άλλες) να συμβάλλουν ενεργά στην ανάπτυξή του χωρίς αποκλεισμούς ή εμπορική εξάρτηση από τη Google.

Χάρη σε αυτή την ανοιχτή και πολυμετοχική ανάπτυξη, το Kubernetes κατάφερε να εδραιωθεί ως το πρότυπο της βιομηχανίας για την εκτέλεση δοχείων σε οποιοδήποτε κατανεμημένο περιβάλλον πολλαπλών κόμβων — είτε σε δημόσιο νέφος, είτε επιτόπια (on-premise), είτε σε υβριδικές και πολυνεφικές (multi-cloud) υλοποιήσεις

2.3.1 Δυνατοτητες Kubernetes

Η συνεχής στροφή της βιομηχανίας προς τις μικρο-υπηρεσίες, τους αγωγούς DevOps και τη cloud-native αρχιτεκτονική, ανέδειξε την ανάγκη για εργαλεία που προσφέρουν:

- Φορητότητα (portability) μεταξύ περιβαλλόντων,
- Αυτοματισμό (automation) στον κύκλο ζωής των εφαρμογών,
- Επεκτασιμότητα (scalability) και ανθεκτικότητα (resilience) στις διανομές.

2.3.1.1 Φορητότητα (Portability)

Η φορητότητα αποτελεί έναν από τους θεμελιώδεις λόγους υιοθέτησης του Kubernetes από οργανισμούς. Οι εφαρμογές που αναπτύσσονται ως δοχειοποιημένοι φόρτοι (containerized workloads) μπορούν να εκτελεστούν σε οποιοδήποτε υποδομή υποστηρίζει Docker και Kubernetes, χωρίς να απαιτούνται αλλαγές στον κώδικα ή στην υποδομή. Αυτό επιτυγχάνεται επειδή:

- Τα pods (βασική μονάδα στο Kubernetes) περιέχουν όλα τα απαραίτητα στοιχεία της εφαρμογής ή ενός συστατικού της (runtime, βιβλιοθήκες, configs), εξασφαλίζοντας πλήρη απομόνωση και επαναληψιμότητα της συμπεριφοράς τους (Kubernetes Documentation, 2025[18].
- Το Kubernetes παρέχει consistent APIs και λειτουργίες ανεξάρτητα από το αν εκτελείται σε AWS, Google Cloud, Azure, επιτόπιο ή υβριδικό νέφος ((Kubernetes Documentation, 2025).

2.3.1.2 Αυτοματισμός (Automation)

Το Kubernetes ενσωματώνει ισχυρές δυνατότητες αυτοματοποίησης του κύκλου ζωής των εφαρμογών, από την ανάπτυξη έως την ανάκτηση από σφάλματα. Μερικά παραδείγματα:

- Deployments (διατάξεις): Αυτόματες διατάξεις και ενημερώσεις (διατάξεων) εφαρμογών με rolling updates και rollbacks.
- Autoscaling (αυτοκλιμάκωση):
    - **Οριζόντια αυτοκλιμάκωση (Horizontal Pod Autoscaler – HPA):** αυξάνει ή μειώνει τον αριθμό των pods μιας εφαρμογής βάσει μετρικών, όπως η χρήση CPU, μνήμης ή άλλων custom μετρικών.

[18] https://kubernetes.io/docs/home/

- **Κάθετη αυτοκλιμάκωση (Vertical Pod Autoscaler – VPA):** προσαρμόζει τους πόρους (CPU, μνήμη) που αποδίδονται σε κάθε pod, ώστε να καλύπτονται καλύτερα οι ανάγκες εκτέλεσης χωρίς να αλλάζει ο συνολικός αριθμός pods

-

- Self-healing (αυτό-επούλωση): Αν μια εφαρμογή αποτύχει, το Kubernetes επανεκκινεί τα pods της εφαρμογής ή τα επαναπρογραμματίζει σε άλλους κόμβους.

- Liveness/Readiness Probes: Αυτόματοι έλεγχοι για την υγεία των εφαρμογών με ενέργειες αποκατάστασης (πχ. επανεκίνηση pods ή μεταφορά κίνησης σε υγιή pods) (δείτε προηγούμενο bullet) (Run:AI).

Αυτά επιτρέπουν ένα δηλωτικό (declarative) μοντέλο, στο οποίο οι DevOps ομάδες δηλώνουν την επιθυμητή κατάσταση, και το Kubernetes αναλαμβάνει να την επιτύχει και να την διατηρήσει αυτόματα.

2.3.1.3 Κλιμακωσιμότητα (Scalability) και Ανθεκτικότητα (Resilience)

Το Kubernetes έχει σχεδιαστεί για να υποστηρίζει μεγάλης κλίμακας κατανεμημένες εφαρμογές που εκτελούνται σε χιλιάδες δοχεία και κόμβους. Τα στοιχεία που ενισχύουν την κλιμακωσιμότητα και την ανθεκτικότητα περιλαμβάνουν:

- Replication Controllers & ReplicaSets: Εξασφαλίζουν ότι πάντοτε εκτελείται ο επιθυμητός αριθμός pods.

- Namespaces (χώροι ονομάτων): Επιτρέπουν τη διαχείριση πολλών απομονωμένων εφαρμογών μέσα στην ίδια συστάδα (cluster) κόμβων. Κάθε εφαρμογή μπορεί να ανήκει σε έναν χώρο ονομάτων – αυτό επιτρέπει τον δίκαιο διαμοιρασμό των πόρων ανά χώρο ονομάτων για τις δοχειοποιημένες εφαρμογές μιας συστάδας μέσω χρήσης κατάλληλων πολιτικών.

- gias (ομοσπονδία συστάδων): Υποστήριξη πολλών συστάδων για multi-region/multi-cloud (πολυπεριφερειακή/πολυνεφική) κλιμακωσιμότητα και ανοχή σφαλμάτων (fault tolerance) (Osmani, Kauppinen, Komu, & Tarkoma, 2021). Αυτό δεν υποστηρίζεται από το ίδιο το Kubernetes αλλά μπορεί να επιτευχθεί με τη χρήση επιπλέον εργαλείων, όπως είναι τα KubeFed[19], Rancher[20], Google Anthos[21] & HashiCorp Consul[22].

[19] https://github.com/kubernetes-retired/kubefed

[20] https://rancher.com/docs/

[21] https://cloud.google.com/blog/topics/anthos

[22] https://developer.hashicorp.com/consul/docs

- Disaster Recovery (ανάνηψη από καταστροφές) & High Availability (υψηλή διαθεσιμότητα): το Kubernetes υποστηρίζει multi-master setups (δείτε προηγ. bullet) και προσωποποιημένες (custom) backup-restore στρατηγικές σε πολλαπλά επίπεδα (Mondal, Zheng, & Cheng, 2024). Για το τελευταίο υποστηρίζεται και η χρήση εξωτερικών εργαλείων, όπως είναι το Velero[23], καθώς και CSI VolumeSnapshots[24] για την ενσωμάτωση προσωποποιημένων οδηγών snapshot/restore

Ειδικά σε Edge και 5G υποδομές, η χρήση Federation και networking λύσεων, όπως το Network Service Mesh (NSM)[25], επιτρέπει ανθεκτική (resilient), γεο-κατανεμημένη (geo-distributed) λειτουργία σε εξαιρετικά απαιτητικά περιβάλλοντα (Osmani, Kauppinen, Komu, & Tarkoma, 2021), (Nguyen, Phan, & Kim, 2022).

2.3.2 Χαρακτηριστικά Kubernetes

Στην ενότητα αυτή θα γίνει μια αναλυτική περιγραφή των βασικών χαρακτηριστικών του Kubernetes, δηλαδή των **μηχανισμών και εργαλείων που υλοποιούν τις δυνατότητές** του, όπως επεκτασιμότητα, φορητότητα, αυτό-επούλωση κ.λπ. Τα χαρακτηριστικα του Kubernetes διακρίνονται στις εξής επιμέρους κατηγορίες:

- Υποδομή και Αρχιτεκτονική
- Διαχείριση Εφαρμογών/Φόρτων
- Αυτοματοποίηση και Ανθεκτικότητα
- Δικτύωση & Υπηρεσίες
- Αποθήκευση
- Ασφάλεια και Διαχείριση Πρόσβασης
- Επεκτασιμότητα & Παραμετροποίηση
- Παρακολούθηση, Καταγραφή και Αποσφαλμάτωση

2.3.2.1 Υποδομή και Αρχιτεκτονική

Η αρχιτεκτονική του Kubernetes περιλαμβάνει δύο είδη συστατικών: συστατικά στο επίπεδο ελέγχου (control plane) και συστατικά στο επίπεδο εκτέλεσης (execution plane).

Control Plane Components (Συστατικά Επιπέδου Ελέγχου)

- API Server: Κεντρική διεπαφή επικοινωνίας.

[23] https://velero.io/docs/v1.17/

[24] https://kubernetes.io/docs/concepts/storage/volume-snapshots/

[25] https://networkservicemesh.io/

- etcd: Κεντρική βάση δεδομένων key–value για την κατάσταση της συστάδας.
- Controller Manager: Παρακολουθεί την επιθυμητή κατάσταση των δοχειοποιημένων εφαρμογών και της συστάδας.
- Scheduler: Αναθέτει pods σε κόμβους.

Execution Plane Components (Συστατικά Επιπέδου Εκτέλεσης)

- kubelet: Διαχειρίζεται τον κύκλο ζωής (lifecycle) των pods.
- kube-proxy: Χειρίζεται την προώθηση δικτυακών αιτημάτων.
- Container runtime (περιβάλλον εκτέλεσης δοχείων): Π.χ. Docker, containerd.

1. Control Plane Components

Το Control Plane είναι ο εγκέφαλος του Kubernetes. Είναι υπεύθυνο για τον καθορισμό της «επιθυμητής κατάστασης» (desired state) και την εποπτεία της τρέχουσας κατάστασης της συστάδας (cluster). Το επίπεδο αυτό συνήθως εκτελείται σε έναν κόμβο της συστάδας, τον επονομαζόμενο master node (κύριο κόμβο), αν και μπορεί να υπάρξει κατανομή του επιπέδου αυτού σε πολλούς κόμβους για λόγους ανθεκτικότητας και αξιοπιστίας.

API Server (kube-apiserver)

Ο API Server είναι η κεντρική διεπαφή επικοινωνίας για όλα τα συστήματα — χρήστες, CLI (kubectl), ελεγκτές (controllers), κόμβους (nodes) και εξωτερικές εφαρμογές (Poniszewska-Marańda & Czechowska, 2021) (Mondal, Zheng, & Cheng, 2024) .

- Παρέχει RESTful API για την επικοινωνία αυτή.
- Όλα τα αιτήματα validation, authentication, authorization περνούν από αυτόν.
- Κάθε εντολή kubectl (π.χ. kubectl apply, kubectl get pods) (δηλ. του βασικού CLI τοου Kubernetes) δρομολογείται στον API Server.

etcd

Το etcd είναι μια κατανεμημένη (distributed), ισχυρά συνεπής (strongly consistent) key–value store που λειτουργεί ως αποθετήριο της κατάστασης της συστάδας (Mondal, Zheng, & Cheng, 2024) (CNCF TAG Security, 2022). Το etcd γίνεται εκμεταλλεύσιμο στο Kubernetes με τους εξής τρόπους:

- Αποθηκεύει κάθε αντικείμενο: pods, services, deployments, secrets, configmaps.
- Υποστηρίζει backup/restore και είναι κρίσιμο για ανάνηψη από καταστροφές (disaster recovery).

- Χρησιμοποιεί το πρωτόκολλο Raft[26][27] για συνέπεια (consistency).

Controller Manager

Ο διαχειριστής ελεγκτών (Controller Manager) είναι ένα συστατικό που εκτελεί όλους τους εσωτερικούς ελεγκτές (controllers), δηλαδή τις λογικές που επιβλέπουν την επιθυμητή κατάσταση και επεμβαίνουν όταν υπάρχουν αποκλίσεις (Mondal, Zheng, & Cheng, 2024).

Παραδείγματα controllers περιλαμβάνουν:

- Node controller (ελεγκτής κόμβων): παρακολουθεί την κατάσταση των κόμβων)
- Replication controller (ελεγκτής αναπαραγωγής): βεβαιώνει τον επιθυμητό αριθμό pods)
- Endpoints controller (ελεγκτής τελικών σημείων): παρακολουθεί τα Pods που αντιστοιχούν σε κάθε Service (δηλαδή το Kubernetes abstraction που ομαδοποιεί Pods και εκθέτει την πρόσβαση σε αυτά) και ενημερώνει το αντίστοιχο αντικείμενο Endpoints με τις IP διευθύνσεις και τις θύρες των Pods που είναι ενεργά. Έτσι, τα Services γνωρίζουν δυναμικά πού να δρομολογούν την κίνηση
- Namespace controller (ελεγκτής χώρων ονομάτων): φροντίζει για τη δημιουργία και τη σωστή διαγραφή των Namespaces, μαζί με τους πόρους που αυτά περιέχουν
- ServiceAccount controller (ελεγκτής λογαριασμών υπηρεσιών): δημιουργεί αυτόματα προεπιλεγμένους λογαριασμούς υπηρεσίας (service accounts) και τα αντίστοιχα secrets, ώστε τα Pods να μπορούν να αυθεντικοποιηθούν στον API server.

Scheduler (kube-scheduler)

Ο scheduler (χρονοπρογραμματιστής) αναλαμβάνει να τοποθετήσει pods σε κατάλληλους κόμβους σύμφωνα με:

- τους *διαθέσιμους πόρους (CPU/memory)*,
- *node selectors (επιλογείς κόμβων)* - απλοί κανόνες που καθορίζουν σε ποιους κόμβους μπορεί να τοποθετηθεί ένα pod βάσει labels (ετικέτες), affinity rules (κανόνες συγγένειας) - πιο εξελιγμένοι κανόνες που επιτρέπουν ή απαγορεύουν τη συνεγκατάσταση pods με βάση άλλα pods ή χαρακτηριστικά κόμβων,
- *topology constraints (περιορισμούς τοπολογίας)*: κανόνες που καθορίζουν πώς κατανέμονται τα pods σε γεωγραφικά ή λογικά domains (π.χ. zones, racks) για βελτίωση ανθεκτικότητας,

[26] https://raft.github.io/

[27] https://kubernetes.io/blog/2019/08/30/announcing-etcd-3-4/

- *πιθανές παρεμβολές (interferences)* κατά μήκος φόρτων εργασίας,
- ορισμένα *χρονικά περιθώρια (deadlines)*,
- *περιορισμούς λογισμικού και πολιτικών (software & policy constraints)*.

Κάνει επιλογή μόνο για pods που δεν έχουν ακόμα "δεσμευτεί" σε κάποιον κόμβο (Mondal, Zheng, & Cheng, 2024).

2. Execution Plane Components (Συστατικά Επιπέδου Εκτέλεσης)

Οι κόμβοι εργασίας (worker nodes**)** είναι τα μηχανήματα (φυσικά ή εικονικά) της συστάδας που φιλοξενούν και εκτελούν τους πραγματικούς φόρτους (workloads) — δηλαδή τα pods. Περιλαμβάνουν τα εξής συστατικά επιπέδου εκτέλεσης:

Kubelet

Το kubelet είναι ο πράκτορας (agent) του Kubernetes (Mondal, Zheng, & Cheng, 2024), υπεύθυνος για τις εξής λειτουργίες:

- Να εκκινεί, παρακολουθεί και διακόπτει pods σύμφωνα με οδηγίες του API Server.
- Να ελέγχει την υγεία των pods.
- Να αναφέρει τον status του κόμβου (node) πίσω στο Control Plane.

kube-proxy

Το kube-proxy χειρίζεται την προώθηση δικτυακών αιτημάτων στον σωστό προορισμό:

- Ρυθμίζει κανόνες iptables ή ipvs ώστε να προωθεί αιτήματα στα pods πίσω από υπηρεσίες Kubernetes (services).
- Υποστηρίζει κατανομή φορτίου (load balancing) βάσει TCP/UDP.

Container Runtime

Το container runtime είναι ο μηχανισμός που εκτελεί τα ίδια τα δοχεία (Poniszewska-Marańda & Czechowska, 2021), ουσιαστικά ένα περιβάλλον εκτέλεσης δοχείων. Το Kubernetes υποστηρίζει διάφορα runtimes:

- Docker[28] (πλέον deprecated),

[28] https://docs.docker.com/

- containerd[29] (προτιμώμενο),
- CRI-O[30] (ειδικά για Red Hat OpenShift),
- Mirantis[31],
- gVisor[32] (sandboxed).

Το Kubernetes επικοινωνεί με το runtime μέσω του Container Runtime Interface (CRI).

2.3.2.2 Διαχείριση Εφαρμογών/Φόρτων στο Kubernetes

Το Kubernetes ακολουθεί ένα **declarative μοντέλο** για την διαχείριση φόρτων (workload management): ο χρήστης ορίζει την επιθυμητή κατάσταση (π.χ. "θέλω 3 instances αυτής της εφαρμογής/συστατικού") και το σύστημα φροντίζει να την υλοποιήσει/επιτεύξει και να τη διατηρήσει. Τα παρακάτω αντικείμενα (resources) είναι βασικά στην υλοποίηση αυτού του μοντέλου.

Pod

Το Pod είναι η βασική μονάδα εκτέλεσης στο Kubernetes (Poniszewska-Marańda & Czechowska, 2021), (Mondal, Zheng, & Cheng, 2024).

- Περιέχει ένα ή περισσότερα δοχεία που μοιράζονται:
    - IP διεύθυνση,
    - volumes (storage),
    - namespace και δίκτυο.
- Εκτελείται σε έναν κόμβο (node).
- Είναι εφήμερο: δεν έχει "δική του μνήμη" ή μόνιμη διάρκεια (αν αποτύχει, δεν επανεκκινείται αυτόματα αν δεν υπάρχει ReplicaSet/Deployment από πίσω).

ReplicaSet

[29] https://containerd.io/docs/

[30] https://cri-o.io/

[31] https://docs.mirantis.com/welcome/

[32] https://gvisor.dev/docs/

Το ReplicaSet είναι υπεύθυνο για να διατηρεί σταθερό αριθμό pods για μια εφαρμογή ή συστατικό εφαρμογής στη συστάδα (Mondal, Zheng, & Cheng, 2024).

- Αν ένα pod αποτύχει ή διαγραφεί, το ReplicaSet το αναδημιουργεί αυτόματα.
- Ο χρήστης ορίζει το πεδίο replicas, π.χ. 3 αντίγραφα.
- Συνήθως δεν δημιουργείται άμεσα από τον χρήστη, αλλά μέσω Deployment.

Deployment

Το πιο διαδεδομένο αντικείμενο για την εκτέλεση stateless εφαρμογών/συστατικών (Poniszewska-Marańda & Czechowska, 2021), (Cloud Native Computing Foundation (CNCF), 2024).

- Αφηρημένο επίπεδο πάνω από το ReplicaSet.
- Υποστηρίζει:
    - Rolling updates: Ενημέρωση χωρίς downtime.
    - Rollback: Επιστροφή σε προηγούμενη έκδοση εφαρμογής/συστατικού.
    - Scaling: Μηχανισμός μαζί με HPA.

StatefulSet

Φόρτος που εξειδικεύεται για την διαχείριση καταστατικών (stateful) εφαρμογών/συστατικών, δηλαδή εφαρμογών που απαιτούν σταθερή ταυτότητα και αποθηκευτικό χώρο, όπως βάσεις δεδομένων ή κατανεμημένα συστήματα (distributed systems) (Poniszewska-Marańda & Czechowska, 2021), (CNCF TAG App Delivery Operator Working Group, 2021), (CNCF TAG Security, 2022). Τα βασικά χαρακτηριστικά ενός StatefulSet είναι τα ακόλουθα:

- Διατηρεί μοναδικά ονόματα για κάθε Pod (π.χ. web-0, web-1).
- Κάθε pod συσχετίζεται με μόνιμο volume (δηλαδή αποθηκευτικό χώρο).
- Η σειρά δημιουργίας/διαγραφής pods είναι προβλέψιμη.

DaemonSet

Χρησιμοποιείται για να εκτελείται ένα pod σε κάθε node του cluster (Mondal, Zheng, & Cheng, 2024).

- Κατάλληλο για:

- Monitoring agents (πράκτορες παρακολούθησης) (π.χ. Prometheus node exporter[33]),
- Logging agents (πράκτορες καταχωρήσεων) (π.χ. Fluentd[34]),
- Network policies / proxies (εφαρμογή πολιτικών δικτύωσης / υλοποίηση πληρεξουσίων).

- Αυτόματα εκκινεί pods σε νέους κόμβους όταν αυτοί προστίθενται στην συστάδα.

Job & CronJob

Τα βασικά χαρακτηριστικά ενός φόρτου τύπου Job (Poniszewska-Marańda & Czechowska, 2021) είναι τα ακόλουθα:

- Εκτελεί (εφάπαξ) ένα task (λειτουργία) μέχρι να ολοκληρωθεί (π.χ. data migration, batch operation).
- Χρήσιμο για εκπαίδευση μοντέλων μηχανικής μάθησης (ML training), την επεξεργασία παρτίδων (batch processing), και την διενέργεια backup.

Ο φόρτος CronJob είναι παρόμοιος με τον Job αλλά είναι χρονοπρογραμματιζόμενος. Ειδικότερα, χρησιμοποιείται για τον (χρονο)προγραμματισμό Jobs σε καθορισμένα χρονικά διαστήματα (π.χ. κάθε βράδυ στις 2πμ cleanup). Αποτελεί παρόμοιο μηχανισμό με τον μηχανισμό cron σε Unix συστήματα

Ο ακόλουθος πίνακας συνοψίζει τα διαφορετικά είδη φόρτων που υποστηρίζονται από το Kubernetes για την διαχείριση δοχειοποιημένων εφαρμογών/συστατικών.

Πίνακας 3: Σύνοψη Workload Objects στο Kubernetes

Workload Object	**Σκοπός**	**Ιδιαιτερότητα**
Pod	Βασική μονάδα εκτέλεσης	Περιέχει container(s), εφήμερη
ReplicaSet	Σταθερός αριθμός pods	Αυτόματη αποκατάσταση
Deployment	Διαχείριση versioned εφαρμογών	Scaling, update, rollback
StatefulSet	Stateful εφαρμογές	Σταθερά ονόματα & volumes
DaemonSet	Εκτέλεση ανά node	Monitoring, logging
Job	Εφάπαξ εργασία	Μέχρι να ολοκληρωθεί
CronJob	Προγραμματισμένη εργασία	Περιοδική

[33] https://github.com/prometheus/node_exporter

[34] https://docs.fluentd.org/

2.3.2.3 Αυτοματοποίηση και Ανθεκτικότητα στο Kubernetes

Το Kubernetes είναι σχεδιασμένο ώστε να μειώνει την ανάγκη για χειροκίνητη διαχείριση και να επιτρέπει σε συστήματα να ανακάμπτουν αυτόματα από αποτυχίες. Αυτό επιτυγχάνεται μέσω ενός συνόλου **χαρακτηριστικών που σχετίζονται με αυτοματοποίηση και ανθεκτικότητα**.

1. **Horizontal Pod Autoscaler (HPA)**

Το HPA προσθέτει ή αφαιρεί pods δυναμικά, ανάλογα με τη χρήση πόρων (π.χ. CPU, memory) ή προσωποποιημένες μετρικές (Nguyen, Phan, & Kim, 2022) (Mondal, Zheng, & Cheng, 2024). Είναι βασικό για αυτόματη προσαρμογή σε μεταβαλλόμενο φόρτο εργασίας. Τα βασικά του χαρακτηριστικά είναι τα εξής:

- Παρακολουθεί μετρικές μέσω του metrics-server ή Prometheus.
- Προσαρμόζει το replicas count (αριθμό αντιγράφων Pod) ενός deployment/replica set.
- Μπορεί να ενεργοποιηθεί με YAML ή kubectl autoscale.

2. **Vertical Pod Autoscaler (VPA)**

Το VPA τροποποιεί τους πόρους ενός pod σε επίπεδο CPU και μνήμης, προσαρμόζοντας δυναμικά τα resource requests (ελάχιστους απαιτούμενους πόρους) και τα limits (μέγιστα επιτρεπόμενα όρια). Σε αντίθεση με τον HPA, που αλλάζει τον αριθμό των pods, το VPA προσαρμόζει το μέγεθος κάθε pod (Mondal, Zheng, & Cheng, 2024), Cloud Native Computing Foundation, 2024).
Είναι ιδανικό για εφαρμογές με μη-γραμμική κατανάλωση πόρων ή για stateful εφαρμογές που δεν κλιμακώνονται εύκολα οριζόντια. Πιο συγκεκριμένα:

- Υπολογίζει και εφαρμόζει νέες τιμές για **requests/limits** με βάση τις πραγματικές μετρικές χρήσης
- Συχνά χρησιμοποιείται offline (π.χ. μόνο για προτάσεις).
- Δεν συνδυάζεται εύκολα με HPA στο ίδιο pod (εκτός αν γίνει split scaling). Η ταυτόχρονη χρήση μπορεί να προκαλέσει συγκρούσεις, καθώς ο HPA βασίζεται συχνά σε μετρικές πόρων (CPU/memory), τις οποίες το VPA μεταβάλλει δυναμικά. Για τον λόγο αυτό εφαρμόζεται η πρακτική της διαχωρισμένης κλιμάκωσης (split scaling): ο HPA εκτελείται βάσει custom metrics (π.χ. throughput, latency), ενώ ο VPA συνεχίζει να ρυθμίζει τα resource requests/limits με βάση την πραγματική χρήση CPU και μνήμης

3. **Cluster Autoscaler**

Ο Cluster Autoscaler προσθέτει ή αφαιρεί worker nodes σε υποδομές νέφους (π.χ. AWS, GCP, Azure), ανάλογα με το αν υπάρχουν pods που δεν μπορούν να εκκινηθούν λόγω έλλειψης πόρων (Poniszewska-Marańda & Czechowska, 2021). Πιο συγκεκριμένα:

- Ενσωματώνεται σε παρόχους νέφους (cloud providers).
- Λειτουργεί σε συνεργασία με HPA/VPA.
- Μειώνει το κόστος μέσω δυναμικής προσθήκης/αφαίρεσης κόμβων (δηλαδή επιτελεί οριζόντια κλιμάκωση πόρων νέφους).

4. Liveness & Readiness Probes

Τα probes (ανιχνευτές) επιτρέπουν στο Kubernetes να ελέγχει την υγεία και διαθεσιμότητα των δοχείων (Poniszewska-Marańda & Czechowska, 2021) (Nguyen, Phan, & Kim, 2022).

- *Liveness probe*: Ελέγχει αν ένα container είναι «ζωντανό» και λειτουργεί σωστά. Αν το liveness probe αποτύγχει, γίνεται επανεκκίνηση του container. Για παράδειγμα, αν ένας web server «κολλάει» και δεν απαντά σε αιτήματα, τότε το liveness probe αποτυγχάνει και γίνεται επανεκκίνηση του web server.
- *Readiness probe*: Αν αποτύχει (δηλ. το δοχείο δεν είναι διαθέσιμο), το αντίστοιχο pod (που ενσωματώνει το δοχείο) αφαιρείται από το service (δεν λαμβάνει traffic).
- Το Kubernetes υποστηρίζει HTTP, TCP ή custom exec probes. Αυτό υποδηλώνει τους διαφορετικούς τρόπους με τους οποίους μπορεί να πραγματοποιηθεί η ανίχνευση: είτε μέσω αποστολής HTTP αιτήματος, είτε με σύνδεση σε συγκεκριμένη TCP θύρα είτε με την εκτέλεση προσωποποιημένης εντολής εντός των δοχείων προς έλεγχο.
- Ο μηχανισμός αυτός του Kubernetes παίζει καθοριστικό ρόλο στην αυτό-επούλωση (self-healing) διότι ανιχνεύει πότε αυτή θα πρέπει να πραγματοποιηθεί.

5. Self-Healing

Το Kubernetes εντοπίζει αποτυχίες (όπως φάνηκε παραπάνω) και παρεμβαίνει αυτόματα για την αποκατάσταση (Mondal, Zheng, & Cheng, 2024), (CNCF TAG Security, 2022), εκτελώντας ενέργειες όπως:

- Επανεκκίνηση αποτυχημένων δοχείων (μέσω liveness).
- Αντικατάσταση pods που ανήκουν σε ReplicaSet/Deployment.
- Απομάκρυνση προβληματικών κόμβων.

6. Rolling Updates & Rollbacks

Μέσω του Deployment resource, το Kubernetes υποστηρίζει ομαλές ενημερώσεις εφαρμογών χωρίς downtime (Poniszewska-Marańda & Czechowska, 2021).

- Rolling update: σταδιακή αντικατάσταση pods (μετάβαση σε νέα έκδοση).
- Rollback: επιστροφή σε προηγούμενη λειτουργική έκδοση (εφόσον η μετάβαση στην νέα αποτύχει).

2.3.2.4 Δικτύωση & Υπηρεσίες (Networking & Services) στο Kubernetes

Το Kubernetes παρέχει ένα ολοκληρωμένο μοντέλο δικτύωσης που εξασφαλίζει απρόσκοπτη επικοινωνία μεταξύ pods και υπηρεσιών, ενώ υποστηρίζει λειτουργίες, όπως ανακάλυψη υπηρεσιών (service discovery), κατανομή φορτίου (load balancing) και έλεγχο της επικοινωνίας με βάση πολιτικές δικτύωσης (network policies).Το μοντέλο αυτό και τα βασικά του στοιχεία είναι τα εξής:

- Kubernetes Network Model
- Services (ClusterIP, NodePort, LoadBalancer)
- DNS & Service Discovery
- Network Policies (Ingress & Egress)
- CNI Plugins & Επεκτάσεις

1. Kubernetes Network Model (Pod-to-Pod Networking)

Το μοντέλο δικτύωσης του Kubernetes βασίζεται στην αρχή ότι κάθε pod διαθέτει (Cloud Native Computing Foundation, 2024) (Mondal, Zheng, & Cheng, 2024) (Poniszewska-Marańda & Czechowska, 2021):

- Μοναδική IP διεύθυνση,
- Πλήρη δυνατότητα επικοινωνίας με άλλα pods, ανεξαρτήτως του κόμβου στο οποίο εκτελούνται.

Κύρια χαρακτηριστικά:

- Δεν απαιτείται NAT.
- Κάθε pod έχει ένα *flat network addressable IP*.
- Η επικοινωνία μεταξύ pods είναι πάντα επιτρεπτή, εκτός αν περιοριστεί με πολιτικές δικτυωσεις (network policies).

2. Services (ClusterIP, NodePort, LoadBalancer)

Το Kubernetes παρέχει το αντικείμενο Service (υπηρεσία) για τη δημιουργία ενός σταθερού endpoint που ενσωματώνει και περιτυλίγει ένα σύνολο pods με κοινά labels. (Cloud Native Computing Foundation (CNCF), 2024)

Τύποι Services:

- ClusterIP (προεπιλογή): Εσωτερική πρόσβαση μόνο μέσα στη συστάδα.
- NodePort: Έκθεση υπηρεσίας σε συγκεκριμένη θύρα όλων των κόμβων.
- LoadBalancer: Δημιουργία δημόσιας IP διεύθυνσης μέσω παρόχου νέφους (cloud provider) για πρόσβαση από το Διαδίκτυο.

3. DNS & Service Discovery

Κάθε service λαμβάνει αυτόματα ένα **DNS entry**, επιτρέποντας την αναγνώριση και επικοινωνία από άλλα pods χωρίς να χρειάζονται σκληρά κωδικοποιημένες διευθύνσεις IP (Cloud Native Computing Foundation (CNCF), 2024) (Google Cloud Platform, 2025).

Παράδειγμα DNS domain:
my-service.my-namespace.svc.cluster.local

4. Network Policies (Ingress & Egress)

Με τις NetworkPolicies (πολιτικές δικτύωσης), μπορεί να περιοριστεί η κίνηση ανάμεσα σε pods καθώς και προς/από εξωτερικά endpoints:

Οι πολιτικές αυτές αποτελούν Kubernetes πόρους/αντικείμενα που επιτρέπουν τον έλεγχο της δικτυακής ροής (ingress/egress) με βάση:

- Pod labels
- IP blocks,
- Namespaces (χώρους ονομάτων),
- θύρες.

Χρησιμοποιούνται για να οριστεί ποιος μπορεί να επικοινωνεί με ποιον. Χωρίς τις πολιτικές αυτές, όπως έχει προαναφερθεί, όλα τα pods επικοινωνούν ελεύθερα μεταξύ τους (Cloud Native Computing Foundation (CNCF), 2024) (Tigera, 2025) (Nguyen, Phan, & Kim, 2022).

5. Ingress & Ingress Controller

Το Ingress είναι ένας πόρος που:

- Επιτρέπει πρόσβαση HTTP/HTTPS σε Services από το εξωτερικό,

- Υποστηρίζει δρομολόγηση με βάση URI paths ή hostnames,
- Συχνά συνδυάζεται με TLS termination και authentication.

Απαιτεί την εγκατάσταση Ingress Controller στην συστάδα, π.χ. NGINX[35], HAProxy[36], Traefik[37] Ο Ingress Controller είναι ένα λογισμικό που τρέχει μέσα στο Kubernetes cluster (συνήθως ως Pods/Deployments) και έχει αποστολή να υλοποιεί τους κανόνες δρομολόγησης που περιγράφονται στα Ingress objects. (Cloud Native Computing Foundation (CNCF), 2024) (CNCF TAG Security, 2022).

6. CNI Plugins (Container Network Interface)

Το Kubernetes δεν εφαρμόζει μόνο του την πολιτική δικτύωσης. Χρησιμοποιεί CNI plugins για την υλοποίηση της συνδεσιμότητας και των πολιτικών (policies). (Cilium, 2025) (Tigera, 2025) (Mondal, Zheng, & Cheng, 2024)

Δημοφιλή CNI Plugins:

- Calico (network policy engine + BGP routing)[38],
- Cilium[39] (eBPF-based, υψηλής απόδοσης),
- Flannel[40], Antrea[41], κ.ά.

Στον παρακάτω πίνακα συνοψίζονται τα χαρακτηριστικά δικτύωσης του Kubernetes

Πίνακας 4: Ανασκόπησης – Δικτύωση στο Kubernetes

Χαρακτηριστικό	**Περιγραφή**	**Τύπος**
Network Model	Flat IP δίκτυο pods χωρίς NAT	Δομικό μοντέλο
Service Types	ClusterIP, NodePort, LoadBalancer	Υπηρεσίες
DNS Discovery	Αυτόματη ονοματολογία services	Service Discovery

[35] https://docs.nginx.com/nginx-ingress-controller

[36] https://www.haproxy.com/documentation/kubernetes-ingress/

[37] https://doc.traefik.io/traefik/reference/install-configuration/providers/kubernetes/kubernetes-ingress/

[38] https://kubernetes.io/docs/concepts/cluster-administration/addons/

[39] https://docs.cilium.io/en/stable/

[40] https://github.com/flannel-io/flannel

[41] https://antrea.io/docs/v2.4.1/

Network Policies	Ingress/Egress filtering με labels/IPs	Πολιτικές για την ασφάλεια στην επικοινωνία
Ingress	HTTP/S entrypoint με TLS και routing	Δημόσια πρόσβαση
CNI Plugins	Υλοποίηση networking μέσω τρίτων	Δικτυακή Υποδομή

7. Αποθήκευση (Storage) στο Kubernetes

Η Αποθήκευση (Storage) στο Kubernetes είναι ένα κρίσιμο τμήμα της αρχιτεκτονικής του, ειδικά για stateful εφαρμογές, βάσεις δεδομένων, κατανεμημένα συστήματα και φόρτους μηχανικής μάθησης (ML workloads). Το Kubernetes υποστηρίζει μοντέλα προσωρινής και μόνιμης αποθήκευσης, και επιτρέπει στο χρήστη να απομονώνει τον αποθηκευτικό χώρο από τα pods, ώστε τα δεδομένα να παραμένουν ανεξάρτητα από τη διάρκεια ζωής ενός δοχείου. Πιο συγκρκιμένα περιέχει:

Volumes

Ένα Volume στο Kubernetes είναι ένας κατάλογος αποθηκευτικού χώρου που είναι προσβάσιμος από τα δοχεία ενός pod (Poniszewska-Marańda & Czechowska, 2021) (Cloud Native Computing Foundation (CNCF), 2024). Τα κύρια χαρακτηριστικά του είναι τα εξής:

- Διαμοιράζεται μεταξύ των δοχείων στο ίδιο pod.
- Επιβιώνει από επανεκκινήσεις του δοχείου, όχι όμως από επανεκκίνηση του pod.
- Υπάρχουν διάφοροι τύποι volumes όπως: emptyDir, hostPath, configMap, secret, persistentVolumeClaim κ.ά.

Persistent Volumes (PVs)

Τα Persistent Volumes είναι πόροι του cluster (όχι pods), που παρέχουν μακροχρόνιο αποθηκευτικό χώρο ανεξάρτητο από το κύκλο ζωής των pods (Cloud Native Computing Foundation (CNCF), 2024) (Mondal, Zheng, & Cheng, 2024). Ειδικότερα:

- Διαχειρίζονται από διαχειριστή ή δυναμικά (με storage class – κλάση αποθήκευσης).
- Υποστηρίζουν διάφορους τύπους καταστατικών αποθηκευτικών χώρων, όπως NFS, iSCSI, cloud storage (EBS, GCE, Azure Disk), Ceph κ.λπ.

Persistent Volume Claims (PVCs)

Τα PVCs είναι αιτήσεις για χώρο σε Persistent Volumes από pods (Cloud Native Computing Foundation (CNCF), 2024). Οι αιτήσεις αυτές:

- Ορίζονται με συγκεκριμένο μέγεθος, access mode (δηλώνει τον τρόπο προσπέλασης και από πόσους κόμβους/Pods μπορεί να χρησιμοποιηθεί ένας persistent volume) και storage class (κλάση αποθήκευσης – ο τύπος της αποθήκευσης προς χρήση, δείτε παρακάτω).
- Το Kubernetes βρίσκει αυτόματα PV που ταιριάζει με την εκάστοτε αίτηση ή δημιουργεί νέο (αν υπάρχει κλάση αποθήκευσης (storage class) με δυναμική τροφοδοσία (dynamic provisioning)).

StorageClass

Το StorageClass καθορίζει τύπους storage με διαφορετικά χαρακτηριστικά (π.χ. γρήγορο SSD, encrypted disk, backup policy) (Cloud Native Computing Foundation (CNCF), 2024). Τα κύρια χαρακτηριστικά της είναι πως:

- Ορίζεται από διαχειριστές.
- Χρησιμοποιείται για δυναμική δημιουργία PVs.
- Παραδείγματα κλάσεων αποθήκευσης είναι τα εξής: AWS gp2[42], io1[43], Google Cloud standard[44], premium[45].

StatefulSet & Storage

Με ένα StatefulSet ορίζεις στο spec.volumeClaimTemplates ένα «πρότυπο» PVC. Ο StatefulSet controller δημιουργεί ξεχωριστό PVC για κάθε Pod-instance, χρησιμοποιώντας το ordinal του Pod στο όνομα. Έτσι προκύπτουν ονομασίες όπως mydb-0, mydb-1 κ.ο.κ. Κάθε Pod πάντα ξανασυνδέεται στο δικό του PVC (και άρα στο ίδιο PersistentVolume), ακόμη κι αν επανεκκινηθεί ή μετακινηθεί σε άλλον κόμβο (node).

Αυτό ακριβώς είναι η «διατήρηση δεδομένων ανά pod» που λέει η βασική πρόταση: ένα Pod ⇄ ένα PVC ή ένα PV, με σταθερή ταυτότητα/όνομα. Το StatefulSet σε συνδυασμό με PVC επιτρέπει τη διατήρηση δεδομένων ανά pod (Cloud Native Computing Foundation (CNCF), 2024):

- Κάθε pod έχει μοναδικό volume claim (π.χ. mydb-0, mydb-1). Αυτό εξασφαλίζεται αυτόματα από το StatefulSet μέσω volumeClaimTemplates. Συνήθης χρήση για βάσεις δεδομένων (PostgreSQL, Cassandra, etcd).

[42] https://docs.aws.amazon.com/ebs/latest/userguide/ebs-volume-types.html?

[43] https://docs.aws.amazon.com/ebs/latest/userguide/ebs-volume-types.html?

[44] https://cloud.google.com/kubernetes-engine/docs/how-to/persistent-volumes/gce-pd-csi-driver

[45] https://cloud.google.com/kubernetes-engine/docs/how-to/persistent-volumes/ssd-pd

Εφήμερη αποθήκευση (Ephemeral storage)

Η εφήμερη αποθήκευση (ephemeral storage) στο Kubernetes αφορά volumes που συνδέονται με τον κύκλο ζωής του Pod και δεν προορίζονται για μόνιμη διατήρηση δεδομένων. Χρησιμοποιούνται για προσωρινά αρχεία και ρυθμίσεις/μυστικά που χρειάζεται η εφαρμογή την ώρα εκτέλεσης—π.χ. emptyDir για cache ή προσωρινό χώρο (δίσκος ή μνήμη), ConfigMap για ρυθμίσεις, Secret για ευαίσθητα δεδομένα, Downward API/Projected volumes για metadata, καθώς και hostPath για πρόσβαση σε πόρους του host (με προσοχή). Όταν το Pod τερματίζει, τα δεδομένα αυτών των volumes συνήθως χάνονται· γι' αυτό για μόνιμη αποθήκευση (π.χ. βάσεις δεδομένων) χρησιμοποιούνται PVC/PV μέσω StorageClass και κατάλληλου CSI driver (δείτε παρακάτω πίνακα για επεξήγηση του στοιχείου αυτού).

Το Kubernetes υποστηρίζει τα παρακάτω volumes εφήμερης αποθήκευσης:

- *emptyDir*: Προσωρινός χώρος που δημιουργείται όταν ξεκινά το Pod και διαγράφεται όταν το Pod τερματίσει. Ιδανικό για cache/temporary αρχεία. Μπορεί να είναι disk (default) ή στη μνήμη (medium: Memory) για ταχύτερη πρόσβαση.
- *hostPath*: Κάνει mount έναν κατάλογο του host μέσα στο Pod. Χρήσιμο για ειδικές περιπτώσεις (π.χ. πρόσβαση σε logs/daemon sockets), αλλά δένει το Pod με συγκεκριμένο node και έχει σημαντικές προεκτάσεις ασφάλειας/φορητότητας. Οπότε, θα πρέπει να χρησιμοποιείται με φειδώ.
- *ConfigMap*: Ρυθμίσεις εφαρμογών τύπου key–value. Μπορούν να γίνει mount ως αρχεία (volume) ή να περαστούν ως μεταβλητές περιβάλλοντος. Οι αλλαγές μπορούν να προωθηθούν δυναμικά σε mounted αρχεία.
- *Secret*: Όμοιο με ConfigMap αλλά για ευαίσθητα δεδομένα (credentials, tokens). Μπορεί να γίνει mount με την μορφή αρχείων ή env vars. Προτείνεται κρυπτογράφηση at-rest, αυστηρό RBAC και αποφυγή εκτύπωσης σε logs.
- *DownwardAPI / Projected volumes*: Παρέχουν στο container metadata του Pod (labels, annotations, resource limits) ή «συνθέτουν» σε ένα volume δεδομένα από πολλαπλές πηγές (π.χ. Secret + ConfigMap).

Ο παρακάτω πίνακας συνοψίζει τα βασικά στοιχεία αποθήκευσης στο Kubernetes.

Πίνακας 5: Ανασκόπησης – Storage στο Kubernetes

Χαρακτηριστικό	**Περιγραφή**	**Χρήση**
Volume	Storage για pod, προσωρινό ή διαμοιραζόμενο	logs, configs
PersistentVolume (PV)	Πόρος συστάδας για διαρκή/επίμονη αποθήκευση	βάσεις δεδομένων

PVC	Αίτηση χρήσης PV	δέσμευση χώρου σε pod
StorageClass	Καθορίζει τύπους αποθήκευσης	SSD, encrypted
CSI	Plug-in interface για storage drivers (οδηγούς αποθήκευσης)	cloud & bare metal (φυσικοί διακομιστές/μηχανήματα)
StatefulSet με VolumeClaimTemplates	Ατομικά PVC per pod	Υπηρεσίες βάσεων δεδομένων τόσο για σχεσιακές όσο και για μη σχεσιακές βάσεις

2.3.2.5 Ασφάλεια και Διαχείριση Πρόσβασης στο Kubernetes

Η ασφάλεια στο Kubernetes καλύπτει τομείς από τον έλεγχο ταυτότητας και την εξουσιοδότηση, έως την προστασία μυστικών (secrets), την απομόνωση εφαρμογών, και τη συμμόρφωση σε enterprise-level συστήματα. Η αρχιτεκτονική του Kubernetes υλοποιεί χαρακτηριστικά ασφάλειας (security features) τόσο στον κύκλο ζωής του λογισμικού όσο και στο runtime. Ειδικότερα, περιλαμβάνει τα εξής χαρακτηριστικά/μηχανισμούς:

1. **Role-Based Access Control (RBAC)**

Το RBAC είναι το βασικό σύστημα εξουσιοδότησης στο Kubernetes. Επιτρέπει τον ορισμό ρόλων (Roles/ClusterRoles) και δεσμεύσεων ρόλων (RoleBindings/ClusterRoleBindings) για να περιορίζεται η πρόσβαση σε αντικείμενα του Kubernetes API (resources), όπως Pods, Deployments, Services, ConfigMaps/Secrets, PVCs, Namespaces, Nodes, καθώς και σε custom resources (CRDs) (Cloud Native Computing Foundation (CNCF), 2024) (CNCF TAG Security, 2022).

Κύρια στοιχεία:

- Role: Ορίζει ποιες ενέργειες (actions / verbs) επιτρέπονται (π.χ. get, list, create) σε ποια resources.
- RoleBinding: Συνδέει ένα ρόλο (Role) με έναν χρήστη ή ομάδα.

  Υπάρχουν και ClusterRole/ClusterRoleBinding: τα ClusterRoles ορίζουν δικαιώματα για cluster-scoped πόρους (π.χ. Nodes, PVs, StorageClasses, Namespaces, CRDs) ή για όλα τα namespaces, ενώ τα ClusterRoleBindings τα αποδίδουν σε επίπεδο όλου του cluster. Αντίθετα, τα Roles/RoleBindings ισχύουν μόνο μέσα σε ένα namespace για namespaced πόρους

2. Secrets & ConfigMaps

Τα Secrets και ConfigMaps είναι πόροι για την αποθήκευση ευαίσθητων πληροφοριών και παραμέτρων διαμόρφωσης, αντίστοιχα (Cloud Native Computing Foundation (CNCF), 2024) (CNCF TAG Security, 2022). Τα μυστικά (Secrets) μπορεί να περιλαμβάνουν passwords, API keys, TLS certs, τα οποία αποθηκεύονται σε κωδικοποιημένη μορφή base64. Τα ConfigMaps περιλαμβάνουν μη ευαίσθητες ρυθμίσεις, όπως app configs.

Τα pods μπορούν να διαβάζουν Secrets/ConfigMaps:

- Ως περιβαλλοντικές μεταβλητές (env),
- Ως προσαρτημένους αποθηκευτικούς χώρους (mounted volumes).

3. Namespaces

Τα Namespaces χρησιμοποιούνται για λογική απομόνωση πόρων σε ένα cluster (Cloud Native Computing Foundation (CNCF), 2024) (Poniszewska-Marańda & Czechowska, 2021). Τα κύρια χαρακτηριστικά των namespaces είναι τα εξής:

- Χρήσιμα για περιβάλλοντα πολλαπλών μισθωτών ή για διαφορετικά περιβάλλοντα ανάπτυξης μιας εφαρμογής (π.χ. dev/staging/prod).
- Οι πόροι, όπως pods, services, secrets και configs, περιορίζονται εντός ενός namespace. Η πρόσβαση σε αυτούς ελέγχεται με RBAC (Roles/RoleBindings ανά namespace), η επικοινωνία των Pods περιορίζεται—αν χρειάζεται—με NetworkPolicies, ενώ η κατανάλωση πόρων οριοθετείται με ResourceQuota/LimitRange
- Είναι δυνατός ο συνδυασμός με RBAC και Network Policies για εφαρμογή πολλών ειδών πολιτικών ανά namespace.

4. Pod Security Standards (PSS) & Admission Controllers

Αρχικά υλοποιούνταν μέσω PodSecurityPolicy (PSP), που όμως έχει αποσυρθεί από το Kubernetes (Cloud Native Computing Foundation (CNCF), 2024) (CNCF TAG Security, 2022). Αντικαταστάθηκε από το:

- Pod Security Admission (ενεργοποιείται σε 1.25+ εκδοση), το οποίο βασίζεται σε Pod Security Standards (privileged, baseline, restricted).

Ελέγχει:

- Αν τα pods τρέχουν ως root,
- Αν επιτρέπονται hostPath mounts,

- Αν υπάρχουν securityContext options.

Στο Kubernetes, οι admission controllers είναι μηχανισμοί στον API server που αναχαιτίζουν αιτήματα μετά το authN/authZ και πριν αποθηκευτούν, ώστε να εφαρμόζουν πολιτικές. Ο Pod Security Admission (PSA) είναι ενσωματωμένος admission controller που, από την έκδοση v1.25 και μετά, αντικαθιστά το PodSecurityPolicy και επιβάλλει τα Pod Security Standards (PSS) σε επίπεδο namespace. Τα PSS ορίζουν τρία επίπεδα αυστηρότητας—Privileged, Baseline, Restricted—και μέσω labels (π.χ. pod-security.kubernetes.io/enforce: restricted) και λειτουργιών enforce/audit/warn ελέγχουν κατά τη δημιουργία/τροποποίηση των Pods αν τηρούνται κανόνες όπως: απαγόρευση privileged containers, hostPath και host namespaces (hostNetwork/hostPID/hostIPC), απαίτηση runAsNonRoot, κατάλληλο seccomp profile και περιορισμοί σε Linux capabilities και επιτρεπτούς τύπους volumes. Με αυτόν τον τρόπο, το RBAC απαντά στο «ποιος» έχει πρόσβαση σε πόρους, ενώ ο PSA/PSS στο «πώς» επιτρέπεται να εκτελούνται τα Pods, διασφαλίζοντας συνεπή επιβολή πολιτικών ασφάλειας ανά namespace.

5. TLS EverywhereCertificate Rotation

Το Kubernetes υποστηρίζει end-to-end TLS κρυπτογράφηση στην επικοινωνία (Cloud Native Computing Foundation (CNCF), 2024) (CNCF TAG Security, 2022). Στην λύση που υποστηρίζεται από το Kubernetes:

- Όλα τα components (API server, etcd, kubelets) επικοινωνούν μέσω TLS.
- Οι πιστοποιήσεις (certificates) έχουν καθορισμένη διάρκεια και μπορούν να ανανεωθούν αυτόματα (kubeadm certs renew).

6. Audit Logging

Το Kubernetes υποστηρίζει audit policy για την καταγραφή όλων των αιτημάτων στον API Server (Cloud Native Computing Foundation (CNCF), 2024) (CNCF TAG Security, 2022). Αυτό είναι χρήσιμο για εξιχνίαση παραβιάσεων, συμμόρφωση και ανάλυση. Η συμπεριφορά του ορίζεται από μια audit policy, δηλαδή σύνολο κανόνων που "ταιριάζουν" αιτήματα βάσει χρηστών/ομάδων, ενεργειών (verbs), πόρων/υποπόρων (resources/subresources), namespaces ή και non-resource URLs, και καθορίζουν τι και πότε θα καταγραφεί Επιπλέον, υποστηρίζονται προσωποποιημένοι κανόνες: level (metadata, request, requestResponse), stages (RequestReceived, ResponseComplete).

Στην audit policy, το πεδίο level καθορίζει την έκταση της καταγραφής. Το Metadata αποθηκεύει μόνο μεταδεδομένα—π.χ. χρήστης, ενέργεια, πόρος, namespace, IP, χρονική σήμανση—χωρίς σώματα αιτημάτων/απαντήσεων. Το Request προσθέτει και το σώμα του αιτήματος (request body), ενώ το RequestResponse περιλαμβάνει και το σώμα της απόκρισης (response body). Όσο ανεβαίνει

το level, τόσο αυξάνονται ο όγκος δεδομένων, το λειτουργικό κόστος και οι κίνδυνοι απορρήτου (π.χ. πιθανή καταγραφή ευαίσθητου περιεχομένου).

Παράλληλα, τα stages ορίζουν το στάδιο του κύκλου ζωής ενός αιτήματος στο οποίο θα παραχθεί το audit event: RequestReceived (με το που παραληφθεί), ResponseStarted (όταν σταλούν headers, κυρίως για long-running αιτήματα όπως watch), ResponseComplete (όταν ολοκληρωθεί πλήρως η απόκριση).

7. Network Policies

Όπως προαναφέρθηκε, οι Network Policies (πολιτικές δικτύωσης) επιτρέπουν τον έλεγχο της δικτυακής κίνησης (Cloud Native Computing Foundation (CNCF), 2024) (Nguyen, Phan, & Kim, 2022). Μπορούν να χρησιμοποιηθούν με βάση pod selectors, IP ranges, namespace selectors. Ειδικεύονται στο να προστατεύουν τις δοχειοποιημένες εφαρμογές από πλευρικές κινήσεις (lateral movement) σε περίπτωση παραβίασης (breach) της συστάδας.

8. Πολιτικές Πόρων (Resource Policies)

Οι πολιτικές πόρων ρυθμίζουν τι και πόσο μπορεί να καταναλώνει κάθε namespace και τι προεπιλογές/όρια έχουν τα Pods/containers. Οι δύο βασικοί μηχανισμοί είναι:

- *ResourceQuota*: Θέτει συνολικά "ταβάνια" ανά namespace (π.χ. requests.cpu, limits.memory, πλήθος pods, services, persistentvolumeclaims, συνολικό requests.storage κ.ά.). Αν ξεπεραστεί κάποιο όριο, η δημιουργία νέων πόρων απορρίπτεται. Προστατεύει από το ανεξέλεγκτο κόστος ενώ επιτρέπει την δίκαιη κατανομή πόρων κατά μήκος των namespaces.

- *LimitRange*: Ορίζει προεπιλεγμένα (default/defaultRequest) και ελάχιστα/μέγιστα (min/max) όρια πόρων (το κάτω/min όριο αντιστοιχεί σε requests και το άνω/max όριο σε limits) για containers (και ελάχιστα μεγέθη PVC). Αποτρέπει Pods χωρίς ορθά ορθά όρια πόρων.

Ο παρακάτω πίνακας συνοψίζει τα βασικά στοιχεία ασφάλειας και διαχείρισης πρόσβασης στο Kubernetes

Πίνακας 6: Ανασκόπησης – Ασφάλεια στο Kubernetes

Μηχανισμός	**Ρόλος στην Ασφάλεια**
RBAC	Περιορίζει την πρόσβαση σε resources
Secrets/ConfigMaps	Αποθήκευση credentials και configs
Namespaces	Απομόνωση περιβαλλόντων και πόρων
Pod Security	Προδιαγραφές ασφαλούς εκκίνησης pods
TLS	Ασφαλής επικοινωνία μεταξύ components

Audit Logging	Καταγραφή αιτημάτων στον API Server
Network Policies	Περιορισμός κίνησης μεταξύ pods
Resource Policies (ResourceQuota / LimitRange)	Συνολικά όρια ανά namespace καθώς και προεπιλογές και min/max για requests/limits ανά Pod/container

2.3.2.6 Επεκτασιμότητα & Παραμετροποίηση στο Kubernetes

Το Kubernetes σχεδιάστηκε εξαρχής ως επεκτάσιμη πλατφόρμα, επιτρέποντας στους χρήστες και στους διαχειριστές να προσθέτουν λειτουργίες, να δημιουργούν νέους τύπους πόρων και να προσαρμόζουν τη συμπεριφορά του συστήματος χωρίς να επηρεάζουν τον πυρήνα. Αυτό γίνεται μέσω μηχανισμών, όπως Custom Resource Definitions (CRDs), Operators, Admission Controllers, Helm και API aggregation, οι οποίοι θα αναλυθούν στη συνέχεια.

1. **Custom Resource Definitions (CRDs)**

Τα CRDs επιτρέπουν στον χρήστη να ορίσει νέους τύπους αντικειμένων (resources) στο Kubernetes API, χωρίς να χρειάζεται αυτός να αλλάξει τον πυρήνα του Kubernetes (Cloud Native Computing Foundation (CNCF), 2024) (CNCF TAG App Delivery Operator Working Group, 2021). Με αυτόν τον τρόπο, ορίζεται ένα λεξιλόγιο πεδίου (domain-specific API) που εκφράζει απευθείας έννοιες του εκάστοτε συστήματος—π.χ. KafkaCluster, PostgresCluster, TensorFlowJob, Backup—αντί για «χαμηλού επιπέδου» συνδυασμούς από Pods/Services/PVCs. Στην πράξη, κάθε CRD συνοδεύεται (συχνά) από έναν custom controller (Operator) που υλοποιεί τον reconciliation loop: παρακολουθεί το επιθυμητό state (το spec του custom resource), συγκρίνει με το πραγματικό, και εκτελεί βήματα μέχρι να επιτευχθεί η ζητούμενη κατάσταση (δημιουργία/ενημέρωση Pods, Services, PVCs, ρυθμίσεων κ.λπ.).

Σε σύνθετα συστήματα, ο κύκλος ζωής δεν είναι «ένα Deployment» αλλά μια αλληλουχία βημάτων: παροχή και συσχέτιση επίμονης αποθήκευσης ανά replica, ρύθμιση υπηρεσιών και endpoints, κυλιόμενες αναβαθμίσεις με σωστή σειρά, backup/restore, failover, πολιτικές κλιμάκωσης, έλεγχοι υγείας και rollback. Τα CRDs επιτρέπουν αυτή τη ροή να περιγραφεί δηλωτικά σε ένα αντικείμενο υψηλού επιπέδου (π.χ. «3 brokers με X storage και Y ρυθμίσεις»), ενώ ο controller αναλαμβάνει αυτοματοποιημένα την εκτέλεση των βημάτων με ασφάλεια και επαναληψιμότητα. Έτσι επιτυγχάνονται: τυποποίηση διαδικασιών, μείωση λαθών, self-service για ομάδες προϊόντος (ένα kubectl apply αντί για δεκάδες manifests) και καλύτερη συμμόρφωση/ορατότητα μέσω πεδίων status/conditions).

2. **Operators**

Ένας Operator (χειριστής) είναι ένα πρόγραμμα που χρησιμοποιεί CRDs και προσωποποιημένη λογική (custom logic) για να αυτοματοποιεί την πλήρη διαχείριση μιας εφαρμογής (π.χ. backup, upgrades, failover) (Operator Framework project, 2024) (CNCF TAG App Delivery Operator Working Group, 2021). Οι χειριστές:

- Βασίζονται σε έναν custom controller.
- Ακολουθούν το πρότυπο "Kubernetes native management of complex apps".[46] Αυτό ουσιαστικά αντικατοπτρίζει τα παραπάνω που αναφέρθηκαν στον ορισμό του τι είναι χειριστής.
- Συχνά γράφονται με Operator SDK (Go, Helm ή Ansible).

Παραδείγματα χειριστών:

- Prometheus Operator[47]
- MongoDB Atlas Operator[48]
- Kubeflow Pipelines Operator[49]

3. Admission Controllers

Τα Admission Controllers (ελεγκτές εισδοχής) είναι plugins στον API Server που τροποποιούν ή απορρίπτουν αιτήματα προς τη συστάδα (Cloud Native Computing Foundation (CNCF), 2024) (CNCF TAG Security, 2022). Παρέχουν εφαρμογή πολιτικών (policy enforcement), επαλήθευση (validation) με δυνατότητα απόρριψης αιτημάτων και προαιρετική τροποποίηση (mutation) των αντικειμένων του Kubernetes πριν αποθηκευτούν: οι built-in ελεγκτές (π.χ. ResourceQuota, Pod Security Admission, LimitRanger) επιβάλλουν κανόνες, ενώ τα Mutating/Validating Admission Webhooks επιτρέπουν προσαρμοσμένη τροποποίηση ή απόρριψη—σημειώνεται ότι η επεξεργασία γίνεται πάνω στα API objects και όχι στο αρχικό YAML **Helm**

Το Helm είναι ο διαχειριστής πακέτων (package manager) του Kubernetes. Επιτρέπει την εγκατάσταση, αναβάθμιση και παραμετροποίηση εφαρμογών και χειριστών μέσω Helm Charts (Helm community, 2025) (Poniszewska-Marańda & Czechowska, 2021).

- Helm charts = templates (πρότυπα) + variables (μεταβλητές) για να επαναχρησιμοποιούνται.
- Υποστηρίζει versioning, rollback, και dependency management για τα πακέτα.

4. API Aggregation

[46] https://kubernetes.io/docs/concepts/extend-kubernetes/operator/

[47] https://prometheus-operator.dev/docs/getting-started/introduction/

[48] https://www.mongodb.com/docs/

[49] https://sky-uk.github.io/kfp-operator/versions/v0.7.0/

Το API Aggregation Layer επιτρέπει στον kube-apiserver να «φιλοξενεί» επιπλέον ομάδες APIs προωθώντας τα αιτήματα σε aggregated (extension) API servers που δηλώνονται με αντικείμενα APIService. (Cloud Native Computing Foundation (CNCF), 2024).

- Παράδειγμα: Metrics API (/apis/metrics.k8s.io/) που παρέχεται από τον metrics-server, οπου ο kube-apiserver δρομολογεί τα αιτήματα προς αυτόν, ώστε να είναι διαθέσιμες μετρικές πόρων για Pods και Nodes.

2.3.2.7 Παρακολούθηση, Καταγραφή και Εντοπισμός Σφαλμάτων στο Kubernetes

Η παρακολούθηση (observability) αποτελεί **θεμελιώδες στοιχείο για την υγιή λειτουργία ενός Kubernetes cluster**. Καθώς τα clusters επεκτείνονται και εκτελούν πλήθος κατανεμημένων υπηρεσιών, η ανάγκη για καταγραφή logs και μετρήσεων (metrics), ανάλυση συμβάντων και εντοπισμό σφαλμάτων γίνεται πιο κρίσιμη. Το Kubernetes από μόνο του παρέχει βασικές υποδομές, αλλά ο πλήρες αγωγός παρακολούθησης (monitoring pipeline) υλοποιείται μέσω **εργαλείων τρίτων** και **native APIs**. Ορισμένα από αυτά αναφέρονται ακολούθως.

1. **Metrics Monitoring**

Το Kubernetes υποστηρίζει μετρικές επιδόσεων (CPU, memory, network) μέσω του metrics-server, ενός add-on που συλλέγει δεδομένα από το kubelet κάθε node (Cloud Native Computing Foundation (CNCF), 2024) (Poniszewska-Marańda & Czechowska, 2021) (Cloud Native Computing Foundation (CNCF)) (μέσω Prometheus/Grafana stack). Ο εξυπηρετητής αυτός:

- Χρησιμοποιείται από το Horizontal Pod Autoscaler (HPA) για αυτοκλιμάκωση (autoscaling).
- Παρέχει μετρικές στο τελικό σημείο /apis/metrics.k8s.io. Υποστηρίζονται βασικές resource metrics για Pods/Nodes (CPU/Memory) για άμεση κατανάλωση από kubectl top και HPA.
- Δεν κρατά ιστορικό, δεν υποστηρίζει custom/exporter metrics, δεν παρέχει alerting/PromQL και δεν εκθέτει τη λογική κατάσταση αντικειμένων (Deployments/Jobs).

Για παρακολούθηση σε βάθος (δείτε προηγούμενα μειονεκτήματα), χρησιμοποιούνται εργαλεία, όπως το Prometheus, το οποίο:

- Αντλεί επιπλέον μετρικές μέσω kube-state-metrics και exporters.
- Υποστηρίζει alerting και rule-based queries (PromQL). Οι τελευταίες είναι δυναμικές επερωτήσεις που εκτελούνται περιοδικά και εφαρμόζουν συναρτήσεις πάνω στα αποθηκευμένα δεδομένα για να εξαχθούν χρήσιμα συμπεράσματα (π.χ. υπολόγισε το άθροισμα των HTTP requests σε αυτό το URI path από εκείνο το deployment).

Το Grafana[50] συχνά συνοδεύει το Prometheus[51] ως οπτική διεπαφή (visual interface) για την εμφάνιση των μετρήσεων των μετρικών.

2. Logging

Το Kubernetes **δεν αποθηκεύει logs κεντρικά** από προεπιλογή (Cloud Native Computing Foundation (CNCF), 2024) (CNCF TAG Security, 2022) (Poniszewska-Marańda & Czechowska, 2021). Αντ' αυτού:

- Τα **logs των containers** είναι προσβάσιμα μέσω της εντολής kubectl logs.
- Τα stdout/stderr του container καταγράφονται ως log output στο host node.

Για **κεντρική διαχείριση log αρχείων**, χρησιμοποιούνται εργαλεία όπως:

- **Fluentd**[52], **Logstash**[53] για συλλογή,
- **Elasticsearch**[54] για index/αναζήτηση,
- **Kibana**[55] ή **Grafana Loki**[56] για οπτικοποίηση.

Αυτό το σχήμα ονομάζεται συνήθως **EFK (Elasticsearch-Fluentd-Kibana) stack**.

3. Event Collection

Το Kubernetes παρακολουθεί συμβάντα (events) στο cluster (Cloud Native Computing Foundation (CNCF), 2024) (Mondal, Zheng, & Cheng, 2024), τα οποία μπορούν να πάρουν την μορφή δεσμεύσεων pods, αποτυχιών χρονοπρογραμματισμού pods (pods scheduling), κατασκευών πόρων, κλπ.. Τα συμβάντα αυτά μπορούν να προβληθούν με την εντολή kubectl get events. Επίσης, μπορούν να αναλυθούν από συστήματα, όπως το Prometheus, αλλά για event streaming προτείνεται χρήση Fluent Bit[57] ή Kafka.

---

[50] https://grafana.com/docs/

[51] https://prometheus.io/docs/introduction/overview/

[52] https://docs.fluentd.org/

[53] https://www.elastic.co/docs/reference/logstash

[54] https://www.elastic.co/docs/deploy-manage

[55] https://www.elastic.co/kibana

[56] https://grafana.com/docs/loki/latest/

[57] https://docs.fluentbit.io/manual/

4. **Debugging & Troubleshooting**

Το Kubernetes προσφέρει τις εξής εντολές για επιτόπια διάγνωση (Cloud Native Computing Foundation (CNCF), 2024) (CNCF TAG Security, 2022):

- *kubectl describe*: περιγραφή της πλήρους κατάστασης ενός πόρου (π.χ. pod, node).
- *kubectl logs*: προβολή log αρχείων pod/container.
- *kubectl exec*: απομακρυσμένη πρόσβαση στο container shell (κέλυφος δοχείου).
- *kubectl port-forward*: προσωρινή πρόσβαση σε υπηρεσία (π.χ. για τοπικό debugging).
- *kubectl debug (v1.18+)*: εκκίνηση προσωρινού container στο ίδιο pod για live διερεύνηση.

5. **Tracing & Telemetry (Service Meshes, eBPF)**

Σε προηγμένα (advanced) περιβάλλοντα, χρησιμοποιούνται (Cloud Native Computing Foundation (CNCF)) (CNCF TAG Security, 2022):

- Jaeger[58], OpenTelemetry[59], Zipkin[60] για ιχνηλάτηση αιτημάτων μεταξύ των εφαρμογών
- Service meshes, όπως Istio, Linkerd ,για παρακολούθηση latency, retries, fault injection.
- Cilium με eBPF για L7 ορατότητα (visibility).

2.3.3 Σχέση Kubernetes με Operators, Add-ons και εξωτερικά εργαλεία

Το Kubernetes, ως πλατφόρμα ενορχήστρωσης δοχείων, έχει σχεδιαστεί εξ αρχής με γνώμονα την επεκτασιμότητα και την ευελιξία. Αυτή η αρχιτεκτονική προσέγγιση το καθιστά ιδιαίτερα προσαρμόσιμο σε περιβάλλοντα με σύνθετες ανάγκες, καθώς επιτρέπει την ενσωμάτωση επιπρόσθετων μηχανισμών που είτε επεκτείνουν τη βασική του λειτουργικότητα είτε επιτρέπουν τη διαχείρισή του με έξυπνο και αυτοματοποιημένο τρόπο. Σε αυτό το πλαίσιο, αναπτύχθηκαν έννοιες, όπως οι **Operators**, τα **Kubernetes add-ons** και τα **εξωτερικά εργαλεία** που συνεργάζονται με το οικοσύστημα της πλατφόρμας.

Οι **Kubernetes Operators** αποτελούν έναν ιδιαίτερο τύπο επέκτασης της πλατφόρμας, που ενσωματώνει επιχειρησιακή λογική (domain knowledge) για τη διαχείριση πολύπλοκων εφαρμογών. Στην πράξη, οι Operators είναι ειδικά λογισμικά, τα οποία βασίζονται σε **Custom Resource Definitions (CRDs)** και **custom controllers**, ώστε να επιτρέπουν στο Kubernetes να

[58] https://www.jaegertracing.io/docs/2.10/

[59] https://opentelemetry.io/docs/

[60] https://zipkin.io/

"κατανοεί" και να διαχειρίζεται εφαρμογές (φόρτους εργασίας) πέρα από τα βασικά pods και workloads (CNCF TAG App Delivery Operator Working Group, 2021). Για παράδειγμα, ένας Prometheus Operator μπορεί να αναγνωρίζει ένα resource τύπου Prometheus και να αναλαμβάνει τον πλήρη κύκλο ζωής της εφαρμογής του, από το provisioning και τη ρύθμιση παραμέτρων, έως τα updates, το scaling και τα backups. Αυτή η προσέγγιση φέρνει την αρχή της αυτοματοποίησης στο επίπεδο της ίδιας της εφαρμογής και όχι μόνο της υποδομής, επιτρέποντας ένα πλήρως Kubernetes-native μοντέλο διαχείρισης (CNCF TAG App Delivery Operator Working Group, 2021).

Παράλληλα, το Kubernetes υποστηρίζει έναν μεγάλο αριθμό **add-ons** — επαναχρησιμοποιήσιμα, ελαφριά προγράμματα που επεκτείνουν τη λειτουργικότητα του cluster. Τα add-ons δεν ανήκουν απαραίτητα στον πυρήνα του Kubernetes, αλλά εγκαθίστανται για να καλύψουν ανάγκες, όπως η παρακολούθηση (π.χ. metrics-server), η δικτύωση (π.χ. CNI plugins όπως Calico ή Cilium), η ορατότητα (π.χ. kube-state-metrics) ή η πρόσβαση (π.χ. Kubernetes Dashboard, Ingress controllers) (Poniszewska-Marańda & Czechowska, 2021). Τα add-ons θεωρούνται κρίσιμα για την εύρυθμη λειτουργία του cluster, καθώς ενεργοποιούν μηχανισμούς, όπως το autoscaling (HPA), τα DNS names και τη ροή δικτυακής κίνησης, επεκτείνοντας σημαντικά τις βασικές δυνατότητες της πλατφόρμας (Poniszewska-Marańda & Czechowska, 2021).

Τέλος, η αξιοποίηση του Kubernetes μεγιστοποιείται μέσω ενός πλούσιου **οικοσυστήματος εξωτερικών εργαλείων**, τα οποία είτε αυτοματοποιούν τη διαχείριση του cluster είτε προσφέρουν πρόσθετες λειτουργίες. Ενδεικτικά, το **Helm**[61] λειτουργεί ως διαχειριστής πακέτων (package manager) για την εγκατάσταση εφαρμογών στο cluster, επιτρέποντας παραμετροποίηση και versioning μέσω Helm charts (Poniszewska-Marańda & Czechowska, 2021). Εργαλεία GitOps, όπως το **Argo CD**[62] ή το **Flux**[63], εστιάζουν στην αυτόματη εφαρμογή αλλαγών στο cluster με βάση Git repositories, ενισχύοντας τη διαφάνεια και την επαναληψιμότητα. Άλλες λύσεις, όπως το **Prometheus** και το **Grafana**, ενσωματώνονται για λόγους παρακολούθησης, ενώ πλατφόρμες, όπως το **Rancher**, το **Lens**[64] ή το **OpenShift**[65], προσφέρουν γραφικά περιβάλλοντα διαχείρισης για τη βελτίωση της εμπειρίας του χρήστη. Το Kubernetes, ως "πυρήνας", είναι σχεδιασμένο να συνεργάζεται και να ενσωματώνει όλα αυτά τα εργαλεία, χωρίς να απαιτεί αυστηρή εξάρτηση από συγκεκριμένα stacks (CNCF TAG Security, 2022), (CNCF TAG App Delivery Operator Working Group, 2021).

Συνολικά, η δύναμη του Kubernetes δεν έγκειται μόνο στην ίδια του τη λειτουργικότητα, αλλά κυρίως στην ικανότητά του να ενσωματώνεται σε ένα **ευρύτερο, αρθρωτό οικοσύστημα**. Η αρχιτεκτονική του επιτρέπει την επέκταση, την αυτοματοποίηση και τη συνεχή βελτίωση, κάτι που

[61] https://helm.sh/docs/

[62] https://argo-cd.readthedocs.io/en/stable/

[63] https://fluxcd.io/flux/

[64] https://docs.k8slens.dev/

[65] https://docs.redhat.com/en

καθιστά την πλατφόρμα ικανή να εξυπηρετήσει σενάρια από απλές εφαρμογές ιστού (web applications) έως σύνθετα κατανεμημένα συστήματα τεχνητής νοημοσύνης.

ΚΕΦΑΛΑΙΟ 3 – ΜΕΘΟΔΟΛΟΓΙΑ ΑΝΑΣΚΟΠΗΣΗΣ

Το παρών κεφάλαιο περιγράφει τη μεθοδολογία που ακολουθήθηκε για τη συστηματική ανασκόπηση της επιστημονικής και τεχνολογικής βιβλιογραφίας, με στόχο την καταγραφή, κατηγοριοποίηση και ανάλυση των εργαλείων και τεχνικών που σχετίζονται με το οικοσύστημα του Kubernetes.
Η μεθοδολογική προσέγγιση βασίστηκε στο πρότυπο του Systematic Mapping Study (SMS) (Petersen, Vakkalanka, & Kuzniarz, 2015), το οποίο παρέχει δομημένο πλαίσιο για τη συλλογή, αξιολόγηση και ανάλυση δημοσιευμένων πηγών.

Η διαδικασία ανασκόπησης οργανώθηκε σε τέσσερα βασικά στάδια, τα οποία αντιστοιχούν στις 4 βασικές ενότητες του κεφαλαίου:

- Καθορισμός ερευνητικών ερωτημάτων (Ενότητα 3.1),
- Διαμόρφωση στρατηγικής αναζήτησης και επιλογής πηγών (Ενότητα 3.2),
- Καθορισμός κριτηρίων εισαγωγής/αποκλεισμού (Ενότητα 3.3),
- Ποσοτική ανάλυση των αποτελεσμάτων της ανασκόπησης (Ενότητα 3.4).

Η εφαρμογή αυτής της μεθοδολογίας επέτρεψε την τεκμηριωμένη και διαφανή αξιολόγηση των επιλεγμένων εργαλείων, τα οποία στη συνέχεια κατηγοριοποιήθηκαν ανά λειτουργικό τομέα (π.χ. Monitoring, Security, CI/CD), και αποτέλεσαν τη βάση για τη συγκριτική ανάλυση που ακολουθεί στο επόμενο κεφάλαιο.

3.1 Ερευνητικά Ερωτήματα

Η παρούσα συστηματική ανασκόπηση αποσκοπεί στην καταγραφή, κατηγοριοποίηση και ανάλυση των εργαλείων και τεχνικών που έχουν αναπτυχθεί στο οικοσύστημα του Kubernetes, το οποίο αποτελεί πλέον ένα de facto πλαίσιο οργάνωσης και διαχείρισης δοχειοποιημένων εφαρμογών σε cloud-native υποδομές. Το οικοσύστημα αυτό έχει εμπλουτιστεί με πληθώρα εργαλείων που καλύπτουν κρίσιμες λειτουργίες, όπως παρακολούθηση (monitoring), αυτοματισμό ανάπτυξης (CI/CD – Συνεχή Ενοποίηση/Συνεχή Παράδοση), αυτόματη κλιμάκωση (autoscaling), ασφάλεια (security), καθώς και εξειδικευμένες ανάγκες για Μηχανική Μάθηση, υπηρεσίες δικτύωσης, πλέγματα υπηρεσιών (service mesh) και πολυσυσταδική ενορχήστρωση (multi-cluster orchestration).

Στο σημερινό τεχνολογικό περιβάλλον, το οικοσύστημα του Kubernetes έχει εμπλουτιστεί με έναν εντυπωσιακό αριθμό εργαλείων και τεχνικών, τα οποία αναπτύσσονται συνεχώς για να καλύψουν διαφορετικές ανάγκες – από την παρακολούθηση και την αυτοματοποίηση, μέχρι την ασφάλεια και τη διαχείριση πόρων. Αυτή η ποικιλομορφία, αν και εντυπωσιακή, συνοδεύεται συχνά από ελλιπή ή διάσπαρτη τεκμηρίωση, γεγονός που καθιστά δύσκολη την πλήρη κατανόηση του πλαισίου και της χρησιμότητας κάθε εργαλείου.
Για να αντιμετωπιστεί αυτή η πρόκληση, η παρούσα εργασία υιοθετεί μια δομημένη και συστηματική προσέγγιση, βασισμένη στο μοντέλο **Systematic Mapping Study (SMS)** (Petersen,

Vakkalanka, & Kuzniarz, 2015). Μέσα από αυτήν τη μεθοδολογία, γίνεται προσπάθεια να χαρτογραφηθεί το πεδίο, να εντοπιστούν και να ομαδοποιηθούν τα σημαντικότερα εργαλεία και να αξιολογηθούν με βάση αντικειμενικά κριτήρια.

Τα κύρια ερευνητικά ερωτήματα που διαμορφώθηκαν, όπως αναφέρονται παρακάτω, αντλούν έμπνευση και από σχετικές μελέτες, όπως των Shamim, Gibson, Morrison, & Akond Rahman (2022) και Xu, Gao, & Wei (2024), και καθοδηγούν την οργάνωση και ερμηνεία της παρούσας ανασκόπησης.

- *RQ1*: Ποια είναι τα βασικά είδη εργαλείων που έχουν αναπτυχθεί στο οικοσύστημα του Kubernetes;
- *RQ2*: Ποια λειτουργικά χαρακτηριστικά και δυνατότητες προσφέρει κάθε κατηγορία εργαλείων;
- *RQ3*: Πώς αξιολογούνται τα εργαλεία βάσει χαρακτηριστικών, όπως απόδοση, ευκολία χρήσης, επεκτασιμότητα, ασφάλεια και διαλειτουργικότητα;
- *RQ4*: Ποιες τεχνολογικές και ερευνητικές τάσεις έχουν αναδειχθεί στην πρόσφατη βιβλιογραφία γύρω από το Kubernetes;
- *RQ5*: Ποια είναι τα κενά, περιορισμοί ή προκλήσεις που καταγράφονται στην υπάρχουσα έρευνα;

  Κάθε ερευνητικό ερώτημα εξετάζεται σε ξεχωριστό σημείο της μελέτης. Συγκεκριμένα, το *RQ1* και το *RQ2* αναλύονται στο Κεφαλαιο 4, το *RQ3* αναλύεται στο Κεφάλαιο 5, ενώ το *RQ4 και το RQ5* αναλύονται στο Κεφάλαιο 6.

3.2 Πηγές & Στρατηγική Αναζήτησης

Όπως αναφέρθηκε προηγουμένως, η μεθοδολογία ανασκόπησης που εφαρμόστηκε προβλέπει τον καθορισμό ερευνητικών ερωτημάτων, την ανάπτυξη στρατηγικής αναζήτησης, την εφαρμογή κριτηρίων επιλογής/αποκλεισμού και, τέλος, τη χαρτογράφηση και κατηγοριοποίηση των αποτελεσμάτων. Στην τρέχουσα ενότητα επικεντρωνόμαστε στα 2 πρώτα βήματα της μεθοδολογίας.

3.2.1 Στρατηγική αναζήτησης

Η στρατηγική αναζήτησης σχεδιάστηκε ώστε να εντοπίσει ένα ευρύ αλλά σχετικό φάσμα εργαλείων Kubernetes, εστιάζοντας τόσο σε ακαδημαϊκές πηγές όσο και σε τεχνικά repositories και κοινότητες λογισμικού. Η προσέγγιση/στρατηγική αυτή βασίστηκε στα εξής βήματα:

(α) Λέξεις-κλειδιά και λογικοί συνδυασμοί
Στον κάτωθι πίνακα παρουσιάζονται οι λέξεις κλειδιά και οι λογικοί συνδυσμοί που χρησιμοποιήθηκαν για την αναζήτηση των πηγών.

Πίνακας 7: Λέξεις κλειδιά και λογικοί συνδυασμοί

Κατηγορία	**Λέξεις-κλειδιά**	**Παράδειγμα λογικών συνδυασμών**
Monitoring, Observability & Logging	Kubernetes monitoring, observability, logging, metrics collection, distributed tracing, alerting, time-series databases, log aggregation	"Kubernetes" AND ("monitoring" OR "observability" OR "logging" OR "metrics collection" OR "distributed tracing" OR "alerting" OR "time-series databases" OR "log aggregation")
Security	Kubernetes security, container runtime security, admission controllers, vulnerability scanning, RBAC, secrets management, compliance, intrusion detection	"Kubernetes" AND ("security" OR "container runtime security" OR "admission controllers" OR "vulnerability scanning" OR "RBAC" OR "secrets management" OR "compliance" OR "intrusion detection")
Networking	Kubernetes networking, container networking interfaces, service discovery, network isolation, network policies, eBPF, DNS resolution, traffic management	"Kubernetes" AND ("networking" OR "CNI" OR "container networking" OR "service discovery" OR "network isolation" OR "network policies" OR "eBPF" OR "DNS resolution" OR "traffic management")
Storage, Backup & DR	Kubernetes storage, persistent volumes, stateful workloads, backup and recovery, disaster recovery, snapshotting, storage optimization	"Kubernetes" AND ("storage" OR "persistent volumes" OR "stateful workloads" OR "backup" OR "recovery" OR "disaster recovery" OR "snapshotting" OR "storage optimization")
CI/CD	Kubernetes CI/CD, continuous integration, continuous delivery, GitOps,	"Kubernetes" AND ("CI/CD" OR "continuous integration" OR "continuous

	automated deployment, pipelines, declarative deployments	delivery" OR "GitOps" OR "automated deployment" OR "pipelines" OR "declarative deployments")
Cluster Management (CLI/UX tools)	Kubernetes cluster management, CLI tools, user experience, troubleshooting, multi-cluster orchestration, visualization, configuration management	"Kubernetes" AND ("cluster management" OR "CLI tools" OR "user experience" OR "troubleshooting" OR "multi-cluster orchestration" OR "visualization" OR "configuration management")
Autoscaling	Kubernetes autoscaling, horizontal scaling, vertical scaling, predictive autoscaling, workload management, resource optimization, elastic scaling	"Kubernetes" AND ("autoscaling" OR "horizontal scaling" OR "vertical scaling" OR "predictive autoscaling" OR "workload management" OR "resource optimization" OR "elastic scaling")
Machine Learning & AI	Kubernetes machine learning, ML pipelines, AI workloads orchestration, scheduling optimization, resource allocation with ML, edge intelligence	"Kubernetes" AND ("machine learning" OR "ML pipelines" OR "AI workloads" OR "scheduling optimization" OR "resource allocation" OR "edge intelligence")
Service Mesh	Kubernetes service mesh, microservices communication, sidecar proxy, traffic routing, service discovery, observability in service mesh, resilience, eBPF	"Kubernetes" AND ("service mesh" OR "microservices communication" OR "sidecar proxy" OR "traffic routing" OR "service discovery" OR "observability" OR "resilience" OR "eBPF")

Αιτιολόγηση επιλογής λέξεων-κλειδιών ανά κατηγορία

Monitoring, Observability & Logging

Επιλέχθηκαν όροι όπως monitoring, observability, metrics collection, distributed tracing και log aggregation ώστε να καλύπτονται οι βασικές διαστάσεις παρακολούθησης συστημάτων, τόσο σε επίπεδο μετρικών όσο και σε επίπεδο καταγραφής και εντοπισμού σφαλμάτων.

Security

Οι λέξεις-κλειδιά container runtime security, admission controllers, vulnerability scanning, RBAC, secrets management και compliance αποτυπώνουν τις κύριες πρακτικές προστασίας σε Kubernetes clusters, εστιάζοντας σε πολιτικές πρόσβασης, έλεγχο τρωτών σημείων και συμμόρφωση με πρότυπα.

Networking

Χρησιμοποιήθηκαν όροι όπως CNI (Container Networking Interface), service discovery, network policies, eBPF και traffic management, καθώς αποτελούν κρίσιμες τεχνολογίες για την επικοινωνία, την απομόνωση και την αποδοτικότητα των μικροϋπηρεσιών.

Storage, Backup & DR

Λέξεις-κλειδιά όπως persistent volumes, stateful workloads, backup and recovery, disaster recovery και storage optimization αντανακλούν τις ανάγκες διαχείρισης δεδομένων σε Kubernetes, ειδικά για κρίσιμες ή stateful εφαρμογές.

CI/CD

Οι όροι continuous integration, continuous delivery, GitOps, pipelines και declarative deployments αποτυπώνουν τις βασικές προσεγγίσεις αυτοματοποίησης ανάπτυξης λογισμικού, που βρίσκονται στον πυρήνα του DevOps και του cloud-native οικοσυστήματος.

Cluster Management (CLI/UX tools)

Λέξεις-κλειδιά όπως CLI tools, troubleshooting, multi-cluster orchestration, visualization και configuration management επελέγησαν ώστε να καλύπτουν τόσο τις γραμμές εντολών όσο και τα γραφικά περιβάλλοντα διαχείρισης, καθώς και τις προκλήσεις σε πολύπλοκα clusters.

Autoscaling

Χρησιμοποιήθηκαν όροι όπως horizontal scaling, vertical scaling, predictive autoscaling, resource optimization και elastic scaling, που περιγράφουν τις βασικές προσεγγίσεις για δυναμική προσαρμογή των πόρων σε Kubernetes περιβάλλοντα.

Machine Learning & AI

Τα keywords ML pipelines, AI workloads orchestration, scheduling optimization, resource allocation with ML και edge intelligence αποτυπώνουν τη διπλή διάσταση της αλληλεπίδρασης Kubernetes με την Τεχνητή Νοημοσύνη: αφενός το Kubernetes υποστήριζει την εκτέλεση και κλιμάκωση ML/AI εφαρμογών, αφετέρου η ML μπορεί να χρησιμοποιηθεί για τη βελτιστοποίηση της ίδιας της πλατφόρμας.

Service Mesh

Επιλέχθηκαν όροι όπως microservices communication, sidecar proxy, traffic routing, observability in service mesh και resilience, που περιγράφουν τον ρόλο των service mesh εργαλείων στην αξιοπιστία και την ασφάλεια της επικοινωνίας μεταξύ μικροϋπηρεσιών

3.2.2 Πηγές αναζήτησης

Η αναζήτηση πραγματοποιήθηκε σε **δύο βασικούς άξονες πηγών**:

1. Ακαδημαϊκές βάσεις δεδομένων:

- *Google scholar:* ευρέως χρησιμοποιούμενη μηχανή ακαδημαϊκής αναζήτησης, που συγκεντρώνει δημοσιεύσεις από ποικίλες πηγές (επιστημονικά περιοδικά, συνέδρια, αποθετήρια, βιβλία), προσφέροντας ευρεία κάλυψη της βιβλιογραφίας.

- *IEEE Xplore:* κορυφαία και αξιόπιστη βάση δεδομένων για θέματα πληροφορικής, υπολογιστικών συστημάτων, και δικτύων, με ιδιαίτερη συνάφεια προς τις τεχνολογίες cloud και container orchestration.

- *SpringerLink:* παρέχει πρόσβαση σε επιστημονικά περιοδικά και πρακτικά συνεδρίων υψηλής ποιότητας, καλύπτοντας ένα ευρύ φάσμα θεμάτων στη μηχανική λογισμικού και τα συστήματα διαχείρισης υποδομών.

- *Elsevier (ScienceDirect):* μία από τις πλέον έγκυρες και πλήρεις επιστημονικές πλατφόρμες, με εκτενή συλλογή άρθρων σε τεχνολογικά και μηχανολογικά πεδία.

- *MDPI:* ανοιχτής πρόσβασης εκδότης με σύγχρονες και ταχέως δημοσιευόμενες μελέτες, επιτρέποντας ενημερωμένη επισκόπηση πρόσφατων ερευνητικών τάσεων.

- *arXiv.org:* επιλέχθηκε για την κάλυψη προδημοσιευμένων (preprint) άρθρων αιχμής σε τεχνολογίες υπολογιστικού νέφους και αυτοματοποίησης συστημάτων.

Παρότι η βάση arXiv.org δεν περιλαμβάνει αποκλειστικά αξιολογημένες (peer-reviewed) δημοσιεύσεις, εντάχθηκε στην παρούσα μελέτη με συμπληρωματικό ρόλο. Ο λόγος είναι ότι φιλοξενεί προδημοσιεύσεις υψηλής ερευνητικής αξίας, οι οποίες συχνά προηγούνται χρονικά της επίσημης δημοσίευσης σε διεθνή περιοδικά. Έτσι, παρέχει άμεση πρόσβαση σε τρέχουσες ερευνητικές τάσεις και καινοτόμες προσεγγίσεις που αφορούν το ταχέως εξελισσόμενο πεδίο του Kubernetes και του cloud computing.

2. Τεχνικές πηγές και κοινότητες

Ο δεύτερος άξονας πηγών περιλαμβάνει τεχνικές πλατφόρμες και κοινότητες ανοικτού κώδικα, οι οποίες επιλέχθηκαν επειδή αποτελούν τους κύριους χώρους δημοσίευσης, συντήρησης και τεκμηρίωσης των εργαλείων που εντάσσονται στο οικοσύστημα του Kubernetes. Η χρήση τους

θεωρήθηκε αναγκαία, καθώς τα τεχνικά αποθετήρια και οι επίσημοι κόμβοι τεκμηρίωσης περιέχουν πληροφορίες που δεν καλύπτονται επαρκώς στη βιβλιογραφία, όπως οδηγίες εγκατάστασης, αρχιτεκτονικές περιγραφές, ρυθμίσεις και πρακτικές χρήσης. Οι τεχνικές πηγές και κοινότητες είναι οι εξής:

- *GitHub:* βασική πλατφόρμα φιλοξενίας και ανάπτυξης λογισμικού ανοικτού κώδικα· χρησιμοποιήθηκε για εντοπισμό, ανάλυση και τεκμηρίωση των ίδιων των εργαλείων που χρησιμοποιούνται στο οικοσύστημα του Kubernetes.
- *Docker Hub:* κύριο αποθετήριο container images, αξιοποιήθηκε για επαλήθευση της ύπαρξης και της δραστηριότητας των εργαλείων (π.χ. ενημερώσεις, downloads, συντηρητές).
- *CNCF Landscape (https://landscape.cncf.io):* επίσημη χαρτογράφηση του οικοσυστήματος Kubernetes από το Cloud Native Computing Foundation, παρέχει κατηγοριοποίηση, επίσημες περιγραφές και συνδέσμους τεκμηρίωσης για χιλιάδες έργα.
- *ArtifactHub.io:* κεντρικό αποθετήριο για Helm charts και Kubernetes packages, χρησιμοποιήθηκε για επαλήθευση της διαθεσιμότητας και της ωριμότητας των εργαλείων.
- *Helm Hub:* χρησιμοποιήθηκε για αναζήτηση και τεκμηρίωση Helm charts πριν την πλήρη ενσωμάτωσή τους στο ArtifactHub.
- *Medium, Dev.to, CNCF blogs (για αναφορές σε ανερχόμενα εργαλεία):* χρησιμοποιήθηκαν συμπληρωματικά για εντοπισμό αναδυόμενων εργαλείων και πρακτικών, καθώς φιλοξενούν άρθρα από επαγγελματίες και προγραμματιστές που δραστηριοποιούνται ενεργά στο Kubernetes community.

3.2.3 Χρονικός Περιορισμός

Η αναζήτηση περιορίστηκε σε εργαλεία και άρθρα που δημοσιεύθηκαν από το **2018 έως και το 2025**, ώστε να καλύπτεται η **περίοδος ωρίμανσης του Kubernetes** ως mainstream πλατφόρμας ενορχήστρωσης (δοχειοποιημένων εφαρμογών). Σύμφωνα με την CNCF Annual Survey[66] (2019) το 2018 σηματοδοτεί τη μετάβαση του Kubernetes από τεχνολογία πρώιμης υιοθέτησης σε ώριμο, σταθεροποιημένο περιβάλλον παραγωγής, με έντονη αύξηση του αριθμού των εργαλείων και των εμπορικών διανομών.

3.2.4 Στρατηγική τεκμηρίωσης και καταγραφής

Για κάθε εργαλείο ή άρθρο που εντοπίστηκε, καταγράφηκαν:

- Τίτλος/Όνομα, συγγραφέας ή δημιουργός
- Πηγή (π.χ. GitHub, IEEE, ACM)

[66] https://www.cncf.io/wp-content/uploads/2020/08/CNCF_Survey_Report.pdf

- Έτος δημοσίευσης ή τελευταίας ενημέρωσης
- Σύντομη περιγραφή της λειτουργικότητας
- URL ή citation

Επιπλέον, για τα εργαλεία καταγράφηκαν τεχνικά και λειτουργικά πεδία, απαραίτητα για τη μετέπειτα αξιολόγηση και κατάταξη, όπως:

- αριθμός αστεριών (stars) και συνεισφερόντων στο GitHub,
- βαθμός ανοικτότητας του κώδικα,
- ύπαρξη και πληρότητα τεκμηρίωσης,
- αναλογία ενεργών προς κλειστών ζητημάτων (issues),
- συμβατότητα με Kubernetes ή άλλες πλατφόρμες,
- ύπαρξη ή αναφορά σε σοβαρά σφάλματα ή περιορισμούς.

Τα παραπάνω δεδομένα αντλήθηκαν κυρίως από το CNCF Landscape και, όπου ήταν διαθέσιμες, από συμπληρωματικές πηγές, όπως αποθετήρια GitHub ή επίσημη τεκμηρίωση.

Η αρχική καταγραφή των στοιχείων πραγματοποιήθηκε πριν το φιλτράρισμα, ενώ η προκαταρκτική αξιολόγηση (inclusion/exclusion) εφαρμόστηκε μεταγενέστερα, βάσει των κριτηρίων του συστηματικού πλαισίου.

3.3 Τρόπος Φιλτραρίσματος των Εργαλείων

Κατά τη διαδικασία ανασκόπησης εντοπίστηκαν τόσο δημοσιεύσεις όσο και εργαλεία, ωστόσο το στάδιο του φιλτραρίσματος επικεντρώθηκε αποκλειστικά στα εργαλεία. Οι δημοσιεύσεις εξετάστηκαν ως προς το εάν πρότειναν ή περιέγραφαν κάποιο συγκεκριμένο εργαλείο. Στις περιπτώσεις αυτές, οι σχετικές πληροφορίες καταγράφονταν ώστε να αξιολογηθεί η καταλληλότητά του. Όταν μια δημοσίευση δεν αναφερόταν σε συγκεκριμένο εργαλείο, αφαιρούνταν από το τελικό σύνολο.

Με αυτόν τον τρόπο, συλλέχθηκαν πληροφορίες κυρίως για εργαλεία, τα οποία στη συνέχεια φιλτραρίστηκαν βάσει ενός συστηματικού πλαισίου.

Για την επιτυχή υλοποίηση της ανασκόπησης και την εξαγωγή συγκρίσιμων, χρήσιμων και αξιόπιστων αποτελεσμάτων, κρίθηκε απαραίτητη η εφαρμογή ενός συστηματικού πλαισίου φιλτραρίσματος των εντοπισμένων εργαλείων. Το πλαίσιο αυτό περιλαμβάνει:

1. **κριτήρια επιλογής (inclusion criteria)** για την ένταξη κατάλληλων και σχετικών εργαλείων,
2. **κριτήρια αποκλεισμού (exclusion criteria)** για τον αποκλεισμό ακατάλληλων ή άσχετων εργαλείων,

3.3.1 Κριτήρια Επιλογής (Inclusion Criteria)

Τα εργαλεία που περιλαμβάνονται στην ανάλυση πρέπει να πληρούν **τουλάχιστον τα εξής βασικά κριτήρια**:

Πίνακας 8: Κριτήρια Επιλογής (Inclusion Criteria)

Κριτήριο	**Περιγραφή**
Ανοιχτός Κώδικας	Διαθέσιμο με open-source άδεια (π.χ. Apache 2.0, MIT, GPL).
Ενεργή Ανάπτυξη	Τουλάχιστον μία ενημέρωση στο GitHub ή σε αποθετήριο τους τελευταίους 12 μήνες.
Τεκμηρίωση	Τουλάχιστον ελάχιστη τεκμηρίωση (documentation) που να περιλαμβάνει οδηγίες εγκατάστασης και χρήσης.
Συμβατότητα με Kubernetes	Υποστήριξη εκδόσεων Kubernetes >=1.20
Επαληθεύσιμη Υιοθέτηση	Αναφορές σε χρήση από εταιρείες, κοινότητες

3.3.2 Κριτήρια Αποκλεισμού (Exclusion Criteria)

Τα εργαλεία που περιλαμβάνονται στην ανάλυση πρέπει να πληρούν τουλάχιστον τα εξής βασικά κριτήρια:

Πίνακας 9: Κριτήρια Αποκλεισμού (Exclusion Criteria)

Κριτήριο	**Περιγραφή**
Σοβαρά Σφάλματα	Ύπαρξη τουλάχιστον ενός (1) κρίσιμου ή επαναλαμβανόμενου σφάλματος (bug)που δεν έχουν επιλυθεί για χρονικό διάστημα πανω από ένα χρόνο.
Περιορισμένη Υιοθέτηση	Πολύ μικρός αριθμός stars (<10) και contributors (<3) στο GitHub.

3.4 Ποσοτική Ανάλυση

Η ενότητα αυτή παρουσιάζει τα συνολικά αποτελέσματα της διαδικασίας φιλτραρίσματος που εφαρμόστηκε στα εργαλεία. Μετά την εφαρμογή των κριτηρίων επιλογής και αποκλεισμού, τα εργαλεία που πληρούσαν τις προϋποθέσεις εντάχθηκαν στο τελικό σύνολο προς περαιτέρω ανάλυση.

Η αποτύπωση που ακολουθεί επικεντρώνεται σε τρεις βασικές διαστάσεις:

1. Ποσοστά επιλογής και απόρριψης εργαλείων,
2. Κατανομή εργαλείων ανά πάροχο (ερευνητικό ή βιομηχανικό φορέα), και
3. Κατανομή εργαλείων ανά έτος πρώτης δημοσίευσης ή δημιουργίας.

Η ανάλυση των παραπάνω επιτρέπει μια συνολική εικόνα του όγκου, της προέλευσης και της χρονικής εξέλιξης των εργαλείων που απαρτίζουν το οικοσύστημα Kubernetes.

3.4.1 Ποσοστά επιλογής και απόρριψης εργαλείων

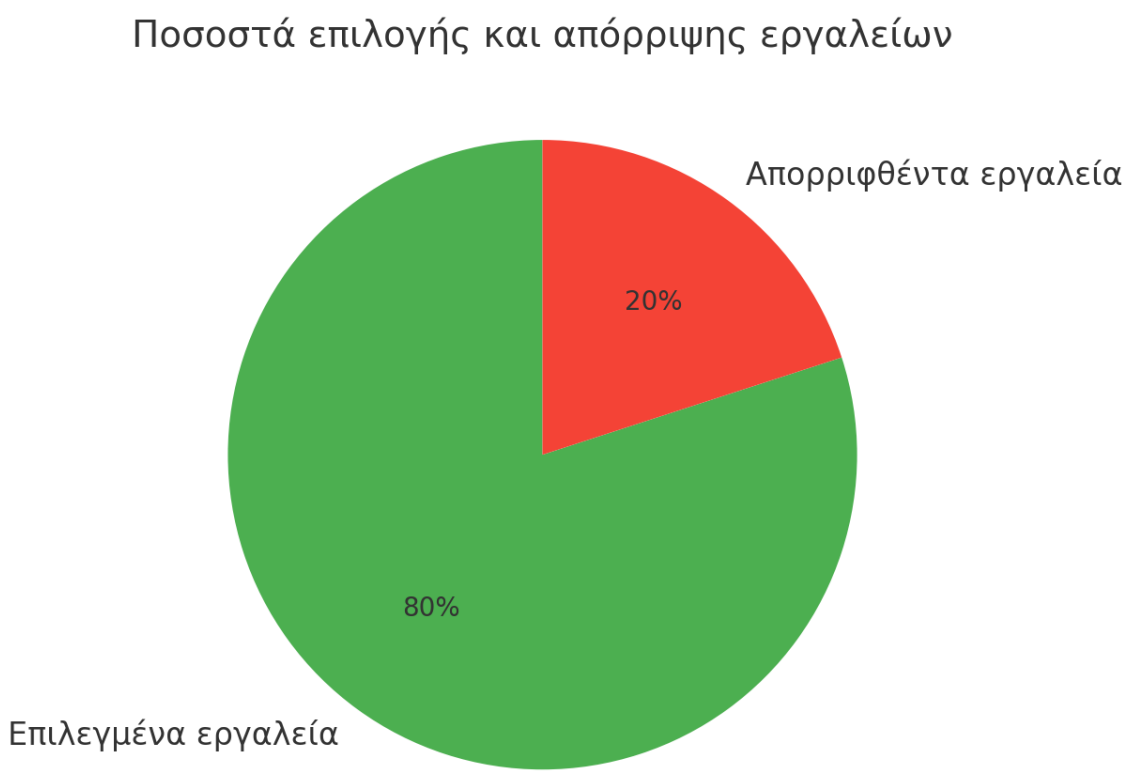
Κατά τη διαδικασία φιλτραρίσματος εντοπίστηκαν συνολικά **40 εργαλεία** που σχετίζονται με τη διαχείριση, την παρακολούθηση και την αυτοματοποίηση εφαρμογών σε περιβάλλοντα Kubernetes.
Μετά την εφαρμογή των κριτηρίων επιλογής και αποκλεισμού, **32 εργαλεία (80%)** πληρούσαν τις προϋποθέσεις και εντάχθηκαν στο τελικό σύνολο προς αξιολόγηση, ενώ **8 εργαλεία (20%)** αποκλείστηκαν.

Οι βασικοί λόγοι απόρριψης σχετίζονταν με:

- την ύπαρξη σοβαρών ή ανεπίλυτων σφαλμάτων, δηλαδή κρίσιμων bugs που παρέμεναν ανοιχτά για διάστημα μεγαλύτερο του ενός έτους, και
- την περιορισμένη υιοθέτηση από την κοινότητα, όπως αποτυπώνεται σε πολύ μικρό αριθμό GitHub stars (<10) και συνεισφερόντων (<3).

Η ανάλυση καταδεικνύει ότι η πλειονότητα των εργαλείων πληρούσε τις προϋποθέσεις ενεργής ανάπτυξης και ωριμότητας, ενώ το ποσοστό των απορριφθέντων εργαλείων αντικατοπτρίζει περιπτώσεις μειωμένης συντήρησης ή περιορισμένης αποδοχής από την κοινότητα. Το γεγονός αυτό αναδεικνύει τη συνεχή εξέλιξη του open-source οικοσυστήματος του Kubernetes και την τάση συγκέντρωσης των προσπαθειών γύρω από ώριμα και ευρέως υποστηριζόμενα έργα

Η Εικόνα 1 παρουσιάζει γραφικά την αναλογία επιλεγμένων και απορριφθέντων εργαλείων.



Εικόνα 1: Ποσοστά επιλογής και απόρριψης εργαλείων

3.4.2 Κατανομή εργαλείων ανά πάροχο

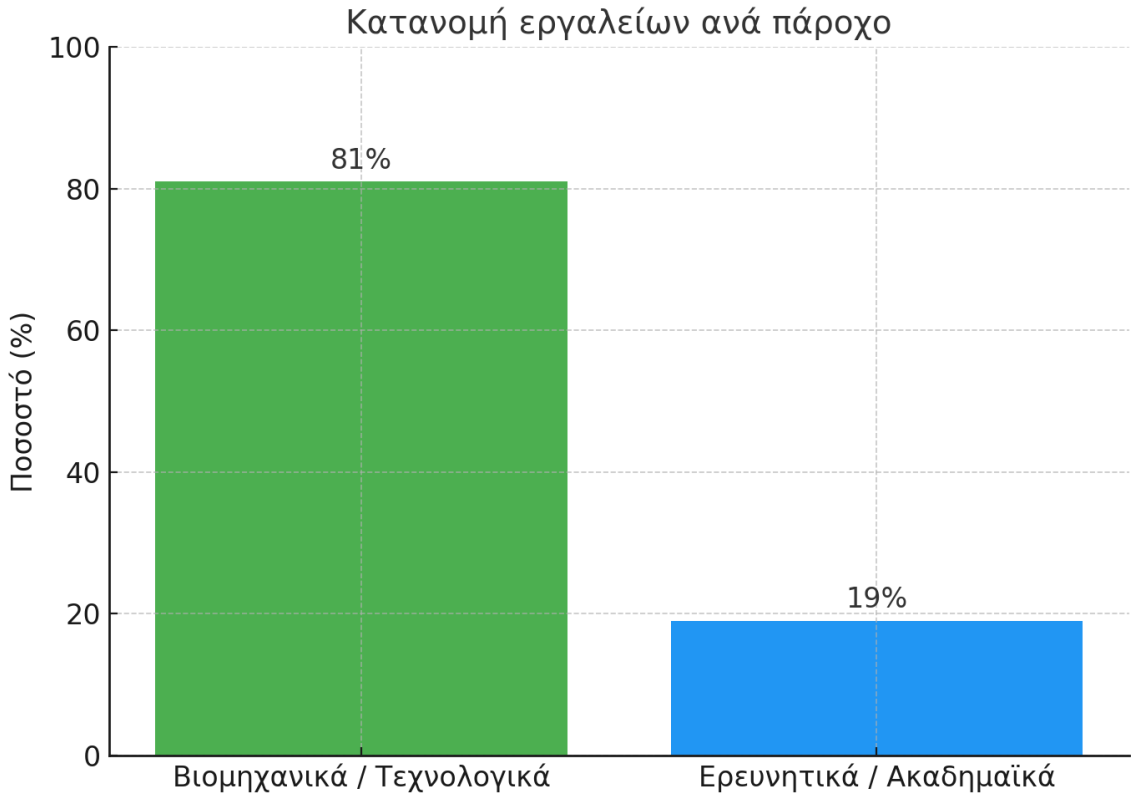
Μετά την ολοκλήρωση της διαδικασίας φιλτραρίσματος, τα επιλεγμένα εργαλεία ταξινομήθηκαν με βάση τον φορέα προέλευσης και ανάπτυξής τους, προκειμένου να αποτυπωθεί ο ρόλος διαφορετικών τύπων οργανισμών στη διαμόρφωση του οικοσυστήματος Kubernetes.

Η κατηγοριοποίηση έγινε σε δύο κύριες ομάδες:

- Βιομηχανικά / Τεχνολογικά εργαλεία, τα οποία αναπτύσσονται ή υποστηρίζονται από εταιρείες, οργανισμούς ή κοινότητες ανοιχτού κώδικα με εμπορικό ή τεχνολογικό προσανατολισμό (π.χ. CNCF, Google, Red Hat, HashiCorp), και
- Ερευνητικά / Ακαδημαϊκά εργαλεία, τα οποία προέρχονται από πανεπιστήμια ή ερευνητικά έργα και στοχεύουν κυρίως στην πειραματική διερεύνηση ή βελτιστοποίηση συγκεκριμένων πτυχών της λειτουργίας του Kubernetes.

Από τα 32 εργαλεία που εντάχθηκαν τελικά στο δείγμα, 26 (81%) ανήκουν στην κατηγορία των βιομηχανικών/τεχνολογικών, ενώ 6 (19%) προέρχονται από ερευνητικούς ή ακαδημαϊκούς φορείς (δείτε Εικόνα 2).

Η κατανομή αυτή καταδεικνύει τη σαφή υπεροχή των βιομηχανικών και τεχνολογικών φορέων στην ανάπτυξη εργαλείων που σχετίζονται με το Kubernetes. Το γεγονός αυτό είναι αναμενόμενο, καθώς το Kubernetes αποτελεί τεχνολογία με έντονη πρακτική εφαρμογή και ευρεία υιοθέτηση στον χώρο της βιομηχανίας λογισμικού. Αντίθετα, η μικρότερη συμμετοχή των ακαδημαϊκών και ερευνητικών οργανισμών υποδηλώνει ότι η σχετική έρευνα βρίσκεται ακόμη σε πρώιμο στάδιο ή εστιάζει σε πιο εξειδικευμένα ζητήματα, όπως η βελτιστοποίηση απόδοσης ή η ασφάλεια.



Εικόνα 2. Ποσοστά εργαλείων ανά πάροχο

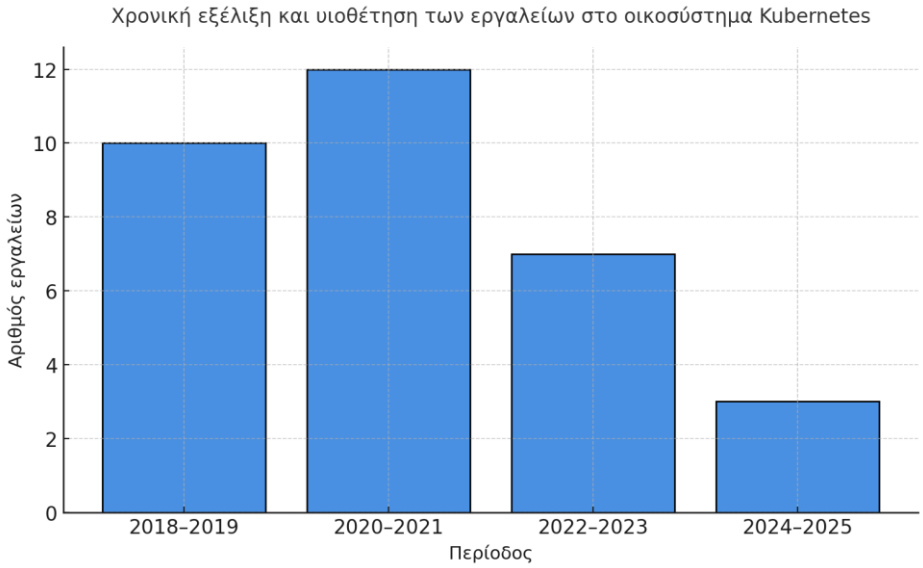
3.4.3 Χρονική κατανομή των εργαλείων με βάση την ενσωμάτωσή τους στο Kubernetes

Η χρονική αποτύπωση των επιλεγμένων εργαλείων πραγματοποιήθηκε με βάση το έτος πρώτης δημόσιας διάθεσης ή, όπου αυτό δεν ήταν διαθέσιμο, το έτος δημιουργίας του αποθετηρίου στο GitHub. Η ανάλυση περιορίστηκε στην περίοδο 2018–2025, καθώς —σύμφωνα με την CNCF Annual Survey (2019)— το 2018 σηματοδοτεί τη μετάβαση του Kubernetes από τεχνολογία πρώιμης υιοθέτησης σε ώριμη πλατφόρμα παραγωγής.

Πρέπει ωστόσο να σημειωθεί ότι ορισμένα εργαλεία (όπως το Grafana, το Prometheus και το Calico) είχαν αναπτυχθεί προγενέστερα του Kubernetes και αρχικά εξυπηρετούσαν γενικότερες υποδομές παρακολούθησης ή δικτύωσης. Στην πορεία, τα εργαλεία αυτά προσαρμόστηκαν και επεκτάθηκαν ώστε να υποστηρίζουν περιβάλλοντα Kubernetes, καθιστώντας τα πλέον αναπόσπαστο τμήμα του οικοσυστήματος. Για τον λόγο αυτό, συμπεριλήφθηκαν στην παρούσα αποτύπωση, με έτος αναφοράς την περίοδο που απέκτησαν ενεργή υποστήριξη Kubernetes.

Η χρονική εξέλιξη των εργαλείων δείχνει ότι η περίοδος 2018–2019 αποτέλεσε την αφετηρία της ώριμης φάσης του οικοσυστήματος Kubernetes, με την εμφάνιση περίπου δέκα νέων εργαλείων που στόχευαν κυρίως στη διαχείριση και ενορχήστρωση υποδομών. Κατά την περίοδο 2020–2021 καταγράφηκε η μεγαλύτερη αύξηση στον αριθμό νέων έργων (περίπου δώδεκα), γεγονός που συνδέεται με τη ραγδαία εξάπλωση του Kubernetes και τη μαζική υιοθέτησή του από τη βιομηχανία. Από το 2022 και μετά, ο ρυθμός ανάπτυξης σταθεροποιείται, με την εμφάνιση πιο εξειδικευμένων εργαλείων που επικεντρώνονται στη βελτιστοποίηση και ενοποίηση υπαρχουσών λύσεων, υποδηλώνοντας τη μετάβαση του οικοσυστήματος σε φάση ωριμότητας.

Η Εικόνα 3 παρουσιάζει τη συνολική χρονική κατανομή των εργαλείων που εντοπίστηκαν και αξιολογήθηκαν στο πλαίσιο της παρούσας ανασκόπησης.



Εικόνα 3. Χρονική κατανομή εργαλείων

ΚΕΦΑΛΑΙΟ 4 – ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΗ & ΑΝΑΛΥΣΗ KUBERNETES ΕΡΓΑΛΕΙΩΝ

Το οικοσύστημα του Kubernetes έχει εξελιχθεί ραγδαία τα τελευταία χρόνια, με την υποστήριξη της κοινότητας και του CNCF (Cloud Native Computing Foundation), δημιουργώντας ένα πλήθος εργαλείων που καλύπτουν τόσο την αυτοματοποίηση όσο και τον πλήρη κύκλο ζωής των δοχειοποιημένων εφαρμογών: από την ανάπτυξη και τη διανομή τους (CI/CD, provisioning, deployment) έως τη λειτουργία (observability, scaling, αναβαθμίσεις, backup/restore) και την ασφάλεια (RBAC/PSA/NetworkPolicies). Στο πλαίσιο αυτό, με τον όρο αυτοματοποίηση εννοούμε τόσο την αυτοματοποίηση της παράδοσης λογισμικού (build/test/release) όσο και της υποδομής και των day-2 διαδικασιών (διαμόρφωση, ελεγχόμενα rollouts/rollbacks, policy enforcement) μέσω πρακτικών όπως GitOps και Operators. Η ορθή επιλογή και ενσωμάτωση των κατάλληλων εργαλείων παραμένει κρίσιμος παράγοντας για τη σταθερότητα, την απόδοση και την κλιμακωσιμότητα κάθε Kubernetes-based υποδομής

Καθώς οι απαιτήσεις των εφαρμογών και των DevOps πρακτικών αυξάνονται, γίνεται αναγκαία η ύπαρξη ενός πλαισίου που να ταξινομεί και να αναλύει αυτά τα εργαλεία με συστηματικό τρόπο. Στο πλαίσιο αυτό, στο παρόν κεφάλαιο παρουσιάζεται μια ιεραρχική κατηγοριοποίηση εργαλείων Kubernetes με βάση δύο βασικές διαστάσεις: (α) τη λειτουργική τους περιοχή (π.χ. Monitoring, Networking, CI/CD) και (β) τον τρόπο ενσωμάτωσης και εκτέλεσής τους σε μια συστάδα Kubernetes (π.χ. CLI-based, Operator-based, DaemonSet) (Cloud Native Computing Foundation (CNCF), 2024) (Operator Framework project).

Η προσέγγιση αυτή επιτρέπει την ορθολογική ανάλυση και επιλογή εργαλείων για διάφορες περιπτώσεις χρήσης (use cases), είτε αυτές αφορούν μικρές συστάδες ανάπτυξης (development clusters) είτε μεγάλες παραγωγικές υποδομές με πολύπλοκες απαιτήσεις. Επιπλέον, η ανάλυση των εργαλείων βασίζεται σε πραγματικά παραδείγματα από το CNCF Landscape (Cloud Native Computing Foundation (CNCF)), καθώς και σε τεχνική τεκμηρίωση από επίσημα έργα (projects) και repositories (Prometheus Authors), (Argo CD project).

Στις ενότητες που ακολουθούν, παρουσιάζεται πρώτα η ιεραρχική κατηγοριοποίηση των επιλεγμένων εργαλείων (Ενότητα 4.1) και στη συνέχεια ακολουθεί αναλυτική αξιολόγηση των εργαλείων αυτών ανά λειτουργική κατηγορία (Ενότητα 4.2), καλύπτοντας περιοχές, όπως ασφάλεια, παρακολούθηση, autoscaling, δικτύωση, αποθήκευση,μηχανική μάθηση κ.α. .

Τονίζεται πως η δομή της ανάλυσης (που εκτείνεται μεταξύ του τρέχοντος και του επόμενου κεφαλαίου) είναι ευθυγραμμισμένη με τα ερευνητικά ερωτήματα που τέθηκαν στο πλαίσιο της εργασίας:

- Μέσω της συστηματικής καταγραφής και ταξινόμησης των εργαλείων (δείτε Ενότητα 4.1), απαντάται το ερευνητικό ερώτημα RQ1, δηλαδή ποια είναι τα βασικά είδη εργαλείων που έχουν αναπτυχθεί στο πλαίσιο του Kubernetes.
- Τα εργαλεία περιγράφονται αναλυτικά σε κάθε λειτουργική κατηγορία (δείτε Ενότητα 4.2) ως προς τις δυνατότητές τους, τα τεχνικά χαρακτηριστικά και την προστιθέμενη αξία τους, με στόχο την κάλυψη του ερευνητικού ερωτήματος RQ2.

- Στη συνέχεια, στο Κεφάλαιο 5, συγκρίνονται όλα τα εργαλεία ως προς την τεχνική πολυπλοκότητα, την ευκολία χρήσης, την επεκτασιμότητα, την απόδοση και τη διαλειτουργικότητα, απαντώντας στο ερευνητικό ερώτημα RQ3.

Η προσέγγιση αυτή προσφέρει μια ολιστική και συγκριτική θεώρηση των εργαλείων Kubernetes, διευκολύνοντας τόσο την τεχνική αξιολόγηση όσο και την ερευνητική ερμηνεία του τοπίου.

4.1 Ιεραρχική Κατηγοριοποίηση

Η πολυμορφία των εργαλείων στο οικοσύστημα Kubernetes καθιστά αναγκαία τη δημιουργία ενός σαφούς και τεκμηριωμένου πλαισίου κατηγοριοποίησης. Η παρούσα εργασία υιοθετεί ένα διπλό μοντέλο ιεραρχικής κατηγοριοποίησης, το οποίο βασίζεται:

1. Στη λειτουργική διάσταση: τι προσφέρει (λειτουργικά) το κάθε εργαλείο (π.χ. παρακολούθηση, δικτύωση, ασφάλεια),
2. Στην τεχνική/αρχιτεκτονική διάσταση: πώς ενσωματώνεται και εκτελείται το κάθε εργαλείο

4.1.1 Λειτουργική Διάσταση

Η πρώτη διάσταση οργανώνει τα εργαλεία ανάλογα με τη **λειτουργικότητα ή το πρόβλημα που επιλύουν**. Οι βασικές κατηγορίες είναι:

Monitoring, Observability & Logging (Παρακολούθηση, Ορατότητα & Καταγραφές), Security (Ασφάλεια), Networking (Δικτύωση), Service Mesh (Πλέγμα Υπηρεσιών), Storage, Backup & Disaster Recovery (Αποθήκευση, Αντίγραφα Ασφαλείας & Ανάκτηση από Καταστροφή), CI/CD (Συνεχής Ολοκλήρωση & Συνεχής Παράδοση/Ανάπτυξη), Cluster Management (Διαχείριση Συστάδων), Autoscaling (Αυτόματη Κλιμάκωση), Machine Learning & AI (Μηχανική Μάθηση & Τεχνητή Νοημοσύνη). Κάθε κατηγορία αναλύεται περαιτέρω σε υποκατηγορίες που αποτυπώνουν πιο εξειδικευμένες λειτουργίες, όπως συλλογή μετρήσεων και ανάλυση καταγραφών στο Monitoring, έλεγχο ευπαθειών και διαχείριση ταυτοτήτων στο Security, ή οριζόντια και κάθετη κλιμάκωση στο Autoscaling. Η ιεραρχική αυτή δομή επιτρέπει την κατανόηση του πώς τα εργαλεία διαφοροποιούνται και αλληλοσυμπληρώνονται στο πλαίσιο της λειτουργίας του Kubernetes, ενώ παράλληλα ευθυγραμμίζεται με την ταξινομική λογική του CNCF Landscape, αναδεικνύοντας τις περιοχές υψηλής συγκέντρωσης τεχνολογικής και ερευνητικής δραστηριότητας.

Λειτουργική Διάσταση

│

├── 1. Monitoring, Observability & Logging

│ ├── Metrics Collection & Synthesis (Συλλογή & Σύνθεση Μετρικών)

│ ├── Log Aggregation & Analysis (Συγκέντρωση & Ανάλυση Καταγραφών)
│ ├── Tracing (Ανίχνευση Διαδρομών Αιτημάτων)
│ └── Visualization & Alerting (Οπτικοποίηση & Ειδοποιήσεις)
│
├── 2. Security (Ασφάλεια)
│ ├── Image Scanning & Vulnerability Detection (Έλεγχος Εικόνων για Ευπάθειες)
│ ├── Identity & Access Management (Διαχείριση Ταυτοτήτων & Δικαιωμάτων)
│ ├── Runtime Security / Policy Enforcement (Ασφάλεια Εκτέλεσης & Πολιτικές)
│ ├── Secrets Management (Διαχείριση Κλειδιών/Διαπιστευτηρίων)
│ └── Compliance & Auditing (Συμμόρφωση & Έλεγχος)
│
├── 3. Networking (Δικτύωση)
│ ├── Ingress / Egress Controllers (Διαχείριση Εισερχόμενης/Εξερχόμενης Κυκλοφορίας)
│ ├── Service Discovery (Ανακάλυψη Υπηρεσιών)
│ ├── Load Balancing (Εξισορρόπηση Φορτίου)
│ └── Network Policies (Πολιτικές Δικτύωσης & Ασφάλειας)
│
├── 4. Service Mesh (Πλέγμα Υπηρεσιών)
│ ├── Traffic Management (Διαχείριση Κυκλοφορίας)
│ ├── Observability & Telemetry (Παρακολούθηση & Μετρήσεις)
│ ├── Security & Encryption (Ασφάλεια & Κρυπτογράφηση Επικοινωνίας)
│ └── Resilience & Fault Injection (Ανοχή Σφαλμάτων & Προσομοίωση Βλαβών)
│
├── 5. Storage, Backup & Disaster Recovery
│ ├── Persistent Storage (Μόνιμη Αποθήκευση)
│ ├── Backup & Snapshot Tools (Αντίγραφα Ασφαλείας & Εργαλεία Στιγμιοτύπων)
│ ├── Disaster Recovery & Failover (Ανάκαμψη από Καταστροφή & Αυτόματη Μετάπτωση)
│ └── Data Migration (Μεταφορά Δεδομένων)
│
├── 6. CI/CD (Συνεχής Ενοποίηση & Παράδοση)

│ ├── Continuous Integration Pipelines (Αγωγοί Συνεχούς Ενοποίησης)

│ ├── Continuous Deployment / Delivery (Συνεχής Ανάπτυξη & Παράδοση)

│ ├── GitOps Frameworks (Πλαίσια GitOps)

│ └── Workflow Orchestration (Ορχήστρωση Ροών Εργασίας)

│

├── 7. Cluster Management (Διαχείριση Συστάδων)

│ ├── Cluster Provisioning (Δημιουργία/Τροφοδοσία Συστάδων)

│ ├── Configuration Management (Διαχείριση Ρυθμίσεων)

│ ├── Cluster Monitoring & TroubleShooting (Παρακολούθηση & Αντιμετώπιση Προβλημάτων Συστάδας)

│ └── Multi-cluster / Hybrid Management (Πολυσυσταδική ή Υβριδική Διαχείριση)

│

├── 8. Autoscaling (Αυτόματη Κλιμάκωση)

│ ├── Horizontal Pod Autoscaling (Οριζόντια Κλιμάκωση Pods)

│ ├── Vertical Pod Autoscaling (Κάθετη Κλιμάκωση Pods)

│ ├── Cluster Autoscaler (Κλιμάκωση Συστάδας)

│ └── Event-driven Scaling (Κλιμάκωση Οδηγούμενη από Γεγονότα)

│

└── 9. Machine Learning & AI (Μηχανική Μάθηση & Τεχνητή Νοημοσύνη)

├── Model Training & Serving (Εκπαίδευση & Παροχή Μοντέλων)

├── ML Pipeline Orchestration (Ορχήστρωση Αγωγών Μηχανικής Μάθησης)

├── Data & Feature Management (Διαχείριση Δεδομένων & Χαρακτηριστικών)

└── Resource Scheduling (Προγραμματισμός Πόρων)

4.1.2 Τεχνική / Αρχιτεκτονική Διάσταση

Η τεχνολογική διάσταση καλύπτει πώς γίνεται η ενσωμάτωση και εκτέλεση του εκάστοτε εργαλείου αρχιτεκτονικά σε σχέση με μια ήδη διαθέσιμη συστάδα Kubernetes. Η κατηγοριοποίηση αυτή επιτρέπει την κατανόηση του **επιπέδου τεχνικής πολυπλοκότητας** που συνεπάγεται κάθε εργαλείο, αλλά και του **τύπου ενσωμάτωσης** που απαιτείται για αυτό, όπως παρουσιάζεται και στο βιβλίο *Kubernetes Patterns* (Ibryam & Huß, 2019).

Οι κατηγορίες που εμπίπτουν σε αυτή τη διάσταση είναι οι ακόλουθες:

- Internal

- External
- Hybrid

Internal

Η κατηγορία Internal περιλαμβάνει εργαλεία, υπηρεσίες και μηχανισμούς που εκτελούνται εντός της συστάδας και επεκτείνουν ή υποστηρίζουν τον πυρήνα λειτουργίας του Kubernetes. Η ενσωμάτωσή τους επιτυγχάνεται μέσω Kubernetes-native μηχανισμών (όπως Deployments, DaemonSets, Operators), ενώ η εκτέλεσή τους υλοποιείται στο ίδιο το control plane ή στους worker nodes. Οι υποκατηγορίες αυτής της κατηγορίας μπορούν να οργανωθούν σε τέσσερα επίπεδα:

• Pod / Container level

Περιλαμβάνει λειτουργίες που εκτελούνται σε επίπεδο pod ή container:

- **Sidecar containers**, τα οποία συνοδεύουν το κύριο δοχείο (container), προσφέροντας πρόσθετη λειτουργικότητα (π.χ. logging, proxying, service mesh components όπως Envoy ή Istio sidecar).
- **Init containers**, που εκτελούνται διαδοχικά πριν από το κύριο δοχείο, με στόχο την προετοιμασία του περιβάλλοντος του (π.χ. αρχικοποίηση volume ή έλεγχο προϋποθέσεων).

• Node level

Σε αυτό το επίπεδο εντάσσονται μηχανισμοί που διασφαλίζουν την παρουσία ενός pod σε κάθε node:

- **DaemonSets**, τα οποία χρησιμοποιούνται για την εκτέλεση πρακτόρων (agents) ανά κόμβο (node), όπως πράκτορες παρακολούθησης (monitoring agents) ή συστατικά δικτύωσης (network components) (CNI plugins όπως Calico, Cilium).

• Cluster level

Αφορά μηχανισμούς που επηρεάζουν ή επεκτείνουν τη λειτουργία ολόκληρου του cluster:

- **Deployments / Services**, για τη διάθεση βασικών υπηρεσιών του cluster (π.χ. CoreDNS, metrics-server, ingress controllers).
- **Operators / Controllers**, που αποτελούν Kubernetes-native επεκτάσεις για τη διαχείριση σύνθετων εφαρμογών μέσω Custom Resource Definitions (CRDs) και reconciliation loops (π.χ. Prometheus Operator, MongoDB Atlas Operator).
- **Webhooks**, που λειτουργούν ως επεκτάσεις του Kubernetes API για επικύρωση ή μεταβολή αιτημάτων (π.χ. admission controllers).

• Job level

Περιλαμβάνει μηχανισμούς που χρησιμοποιούνται για την εκτέλεση περιοδικών ή one-off εργασιών:

- **Jobs** για απομονωμένες εργασίες μικρής διάρκειας.

- **CronJobs** για επαναλαμβανόμενες διεργασίες, όπως προγραμματισμένα backups ή batch data processing.

External

Η κατηγορία περιλαμβάνει εργαλεία που εκτελούνται **εκτός του cluster**, αλλά αλληλεπιδρούν με αυτό μέσω του Kubernetes API. Συνήθως πρόκειται για εργαλεία διαχείρισης, ανάπτυξης ή παρακολούθησης που παρέχουν διεπαφή εντολών (CLI), γραφικό περιβάλλον (GUI) ή περιβάλλον αυτοματοποίησης υποδομής (IaC).
Ενδεικτικά παραδείγματα εργαλείων εντός της κατηγορίας αυτής περιλαμβάνουν:

- **CLI εργαλεία** όπως *kubectl*, *k9s*, τα οποία χρησιμοποιούνται για άμεση αλληλεπίδραση με το cluster.
- **GUI / Management εργαλεία** όπως *Rancher* που παρέχουν οπτικοποιημένες λειτουργίες διαχείρισης συστάδων.

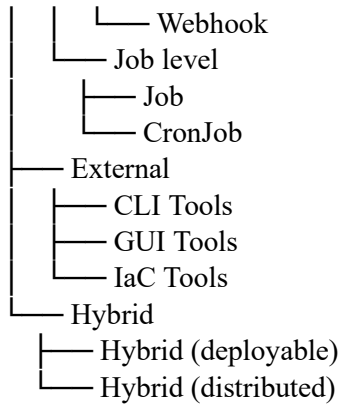
Hybrid

Η κατηγορία Hybrid περιλαμβάνει εργαλεία που συνδυάζουν χαρακτηριστικά των internal και external εργαλείων, παρουσιάζοντας είτε δυνατότητα πολλαπλής εγκατάστασης είτε κατανεμημένη αρχιτεκτονική.
Διακρίνεται σε δύο βασικές υποκατηγορίες:

- **Hybrid (deployable):** Περιλαμβάνει εργαλεία που μπορούν να εγκατασταθούν είτε εντός είτε εκτός του cluster, διατηρώντας την ίδια λειτουργικότητα. Τέτοια εργαλεία προσφέρουν ευελιξία στην ανάπτυξη, επιτρέποντας προσαρμογή ανάλογα με το περιβάλλον (on-premise, cloud ή multi-cluster).
- **Hybrid (distributed):** Αφορά εργαλεία με κατανεμημένη αρχιτεκτονική, στα οποία ορισμένα συστατικά εκτελούνται εντός του cluster (π.χ. agents, exporters, collectors) και άλλα εκτός (π.χ. control plane, dashboards ή APIs). Αυτά τα εργαλεία συνήθως παρέχουν κεντρική διαχείριση ή ανάλυση δεδομένων από πολλαπλές συστάδες.

Παρακάτω παρατίθεται η ιεραρχική κατηγοριοποίηση για την δεύτερη διάσταση.

Τρόπος Ενσωμάτωσης & Εκτέλεσης
├── Internal
│ ├── Pod / Container level
│ │ ├── Sidecar
│ │ └── Init Container
│ ├── Node level
│ │ └── DaemonSet
│ ├── Cluster level
│ │ ├── Deployment / Service
│ │ ├── Operator / Controller



Η παραπάνω προσέγγιση βασίζεται σε πρακτικά και θεωρητικά θεμέλια του cloud native χώρου, όπως αυτά παρουσιάζονται στο CNCF Landscape (Cloud Native Computing Foundation (CNCF)), στο Kubernetes documentation (Cloud Native Computing Foundation (CNCF), 2024), και σε τυποποιημένα αρχιτεκτονικά πρότυπα (Operator Framework project, 2024).

Η ανάλυση αυτή δείχνει ότι οι διαφορετικοί τρόποι ενσωμάτωσης δεν είναι απολύτως διακριτοί. Αντίθετα, συχνά παρατηρείται αλληλοεπικάλυψη μεταξύ τους, καθώς ένα εργαλείο μπορεί να διαθέτει στοιχεία περισσότερων κατηγοριών. Για παράδειγμα, το Prometheus θεωρείται ως εξωτερικό εργαλείο διότι μπορεί να αναπτυχθεί ως service-level εργαλείο εντός του cluster, αλλά ταυτόχρονα να λειτουργεί και ως εξωτερικό εργαλείο συλλέγοντας δεδομένα και από εξωτερικές πηγές. Η συσχέτιση με έννοιες όπως plugins, addons, operators και external tools βοηθά να κατανοήσουμε καλύτερα τον βαθμό ενσωμάτωσης και αυτοματοποίησης που προσφέρει κάθε εργαλείο.

Στον παρακάτω πίνακα παρουσιάζεται η ολοκληρωμένη ανάθεση των επιλεγμένων (με βάση το προηγούμενο κεφάλαιο) εργαλείων στις κατηγορίες των δύο προαναφερόμενων διαστάσεων.

Πίνακας 10: Ανάθεση των εργαλείων στις κατηγορίες των δύο διαστάσεων κατηγοριοποίησης

Λειτουργική Κατηγορία	**Εργαλείο**	**Τρόπος εκτέλεσης**	**Τρόπος ενσωμάτωσης**						
			CLI	Operator	Service-level	Sidecar	Helm	Yaml manifests	Daemonset
Monitoring, Observability & Logging	Prometheus	Hybrid (deployable)		✓	✓		✓	✓	✓

	Grafana	Hybrid (deployable)		✓	✓		✓	✓	
	Loki	Hybrid (distributd)		✓	✓		✓	✓	✓
	Jaeger	Hybrid (distributed)		✓	✓	✓	✓	✓	✓
Security	OPA Gatekeeper	Internal		✓	✓		✓	✓	
	Vault	Hybrid (distributd)		✓		✓	✓	✓	
	Falco	Hybrid (distributd)			✓		✓	✓	✓
	Trivy	Hybrid (deployable)	✓	✓	✓		✓		
Networking	Cilium	Internal		✓	✓		✓		✓
	Calico	Internal		✓	✓		✓		✓
	CoreDNS	Internal			✓			✓	
Storage, Backup & DR	Kasten K10	Hybrid (distributd)		✓	✓		✓		
	Velero	Hybrid (distributd)		✓	✓		✓		
	K8sEs	Hybrid (deployable)		✓	✓				
	Ramen	Hybrid (distributd)		✓	✓				

CI/CD	Argo CD	Hybrid (distributed)	✓	✓	✓		✓		
	Jenkins X	Hybrid (distributed)	✓		✓		✓		
	Argo Workflows	Hybrid (distributed)	✓		✓		✓		
	Drone	Hybrid (distributed)			✓		✓		
	GoCD	Hybrid (distributed)			✓		✓		
Cluster Management (CLI/UX)	kubectl	External	✓						
	K9s	External	✓						
	Portainer	Hybrid (distributed)			✓		✓		
	K8sGPT	Hybrid (deployable)	✓		✓				
	SUSE Rancher	Hybrid (distributed)			✓		✓		
Autoscaling	Hpa (tuned)	Internal		✓					
	KOSMOS	Internal		✓					
	KEDA	Internal		✓	✓		✓		
ML/AI	O DQN Scheduler	Internal			✓		✓		
	KubePipe	Hybrid (distributed)			✓		✓		

	Kubeflow	Hybrid (distributed)	✓	✓	✓		✓	✓	
	Polyaxon	Hybrid (distributed)	✓	✓	✓		✓	✓	
Service Mesh	Istio	Internal			✓	✓	✓		
	Linkerd	Internal	✓		✓	✓	✓		
	Cilium	Internal			✓	✓	✓		✓

4.2 Ανάλυση Ανά Λειτουργική Κατηγορία

Στην παρούσα ενότητα πραγματοποιείται η αναλυτική εξέταση των εργαλείων Kubernetes που εντοπίστηκαν και ταξινομήθηκαν στην ιεραρχική κατηγοριοποίηση της Ενότητας 5.1. Η ανάλυση βασίζεται σε μια λειτουργική προσέγγιση, δηλαδή εξετάζονται τα εργαλεία ανάλογα με τον σκοπό και τον ρόλο που επιτελούν στο οικοσύστημα, όπως για παράδειγμα η παρακολούθηση και παρατηρησιμότητα (Monitoring & Observability), η ασφάλεια (Security), η αποθήκευση και δημιουργία αντιγράφων ασφαλείας (Storage & Backup), η αυτοματοποίηση CI/CD, και άλλες βασικές περιοχές, καθώς και παρουσιάζονται τα πλεονεκτήματα και μειονεκτήματά τους.

4.2.1 Monitoring, Observability & Logging

Η παρακολούθηση (monitoring), η παρατηρησιμότητα (observability) και η συλλογή καταχωρήσεων (logs) αποτελούν βασικούς άξονες για τη διατήρηση της σταθερότητας και αξιοπιστίας σε περιβάλλοντα Kubernetes. Η φύση των δοχειοποιημένων φόρτων (containerized workloads) και η δυναμική διαχείριση (δημιουργία/καταστροφή) pods καθιστούν αναγκαία την ανάπτυξη συστημάτων συνεχούς παρακολούθησης, προειδοποίησης και εντοπισμού σφαλμάτων. Η ανάγκη για ορατότητα πραγματικού χρόνου (real-time visibility) αναδεικνύεται σε πληθώρα πρόσφατων ερευνητικών έργων (Pai & Srinivas, 2024) (Hämäläinen et al 2021).

Πιο συγκεκριμένα, η παρακολούθηση (monitoring) περιλαμβάνει τη συστηματική συλλογή, αποθήκευση και ανάλυση μετρικών που αφορούν την απόδοση και τη διαθεσιμότητα των συστημάτων (π.χ. CPU, μνήμη, latency, throughput). Επιπλέον, εμπεριέχει τη σύνθεση και αξιολόγηση συνθηκών λειτουργίας, την ανίχνευση αποκλίσεων και την παροχή ειδοποιήσεων/alerts όταν εντοπίζονται προβλήματα ή ανωμαλίες.

Αντίστοιχα, η παρατηρησιμότητα (observability) εστιάζει στη δυνατότητα εξαγωγής συμπερασμάτων για την εσωτερική κατάσταση του συστήματος μέσω της ανάλυσης των εξωτερικά παραγόμενων σημάτων (signals). Αυτή συνήθως στηρίζεται σε τρεις βασικούς πυλώνες:

μετρικές, logs και traces, με στόχο την ανακατασκευή του πλαισίου εκτέλεσης μιας υπηρεσίας, τη συσχέτιση γεγονότων και την κατανόηση σύνθετων αλληλεπιδράσεων μεταξύ μικροϋπηρεσιών (microservices). Η παρατηρησιμότητα δεν περιορίζεται μόνο στην ανίχνευση προβλημάτων, αλλά δίνει τη δυνατότητα διάγνωσης των αιτίων και υποστηρίζει τη βελτιστοποίηση της συμπεριφοράς του συστήματος.**Prometheus**

Το Prometheus είναι ένα open-source σύστημα συλλογής και αποθήκευσης μετρικών, σχεδιασμένο ειδικά για cloud-native περιβάλλοντα. Βασίζεται σε pull-based αρχιτεκτονική, δηλαδή ο Prometheus server «τραβά» (pull) περιοδικά τις μετρικές από προκαθορισμένα endpoints (π.χ. /metrics των εφαρμογών ή exporters). Έτσι οι εφαρμογές εκθέτουν απλώς τις μετρικές τους, χωρίς να χρειάζεται να τις αποστέλλουν ενεργά, επιτρέποντας καλύτερο έλεγχο του ρυθμού δειγματοληψίας.

Για την ανάλυση και αξιοποίηση αυτών των δεδομένων χρησιμοποιείται η γλώσσα PromQL, η οποία επιτρέπει τη διατύπωση δυναμικών ερωτήσεων πάνω στις αποθηκευμένες μετρικές. Ο σκοπός αυτών των ερωτήσεων είναι να συνθέσουν παράγωγες ή σύνθετες μετρικές (π.χ. μέσος όρος latency ανά υπηρεσία, 95th percentile response time, ρυθμός αύξησης σφαλμάτων), να επιτρέψουν συσχέτιση δεδομένων από διαφορετικές πηγές και να αποτελέσουν τη βάση για ειδοποιήσεις (alerts) και οπτικοποιήσεις σε dashboards. (Hämäläinen, Rantanen, Aalto, & Pum, 2021), (Hadikusuma, Lukas, & Bachri, 2022).

Η εγκατάστασή του Prometheus μπορεί να γίνει με διάφορους τρόπους: είτε μέσω απλών manifest files, είτε μέσω Helm charts, είτε με τη χρήση του Prometheus Operator, ο οποίος απλοποιεί την ανάπτυξη και δηλωτική διαχείριση (με CRDs) του server, των alerting κανόνων και των exporters. Το Prometheus κατατάσσεται στην κατηγορία Hybrid deployable, καθώς μπορεί επίσης να λειτουργήσει εξωτερικά ως central monitoring component ή μέρος federated περιβάλλοντος που συλλέγει metrics από πολλαπλά clusters. Επιπλέον, η αρχιτεκτονική του δεν περιορίζεται μόνο στον Prometheus server, αλλά πλαισιώνεται και από συμπληρωματικά συστατικά: ο Alertmanager για την επεξεργασία και δρομολόγηση ειδοποιήσεων, το cAdvisor για τη συλλογή μετρικών container σε επίπεδο node (CPU, μνήμη, I/O) και το kube-state-metrics για την παρακολούθηση της κατάστασης Kubernetes αντικειμένων (Deployments, Nodes, Pods). Έτσι, το Prometheus αποτελεί όχι μόνο ένα εργαλείο συλλογής μετρικών, αλλά και έναν ολοκληρωμένο μηχανισμό παρακολούθησης, ανάλυσης και ειδοποίησης, καλύπτοντας κρίσιμες πτυχές της σταθερότητας και αξιοπιστίας σε περιβάλλοντα Kubernetes.
Αναλύουμε παρακάτω τα 3 επιπρόσθετα προαναφερόμενα συστατικά του Prometheus:

- *Alertmanager*: αναλαμβάνει την επεξεργασία και τη δρομολόγηση των ειδοποιήσεων (alerts) που προκύπτουν από κανόνες στο Prometheus. Μπορεί να στείλει alerts σε email, Slack, PagerDuty κ.ά., καθώς και να διαχειριστεί την ομαδοποίηση ή καταστολή τους.

- *cAdvisor (Container Advisor)*: συλλέγει μετρικές για containers σε επίπεδο node (π.χ. χρήση CPU, μνήμης, I/O). Στα Kubernetes clusters, οι μετρικές αυτές είναι κρίσιμες για την παρακολούθηση της απόδοσης των workloads.

- *kube-state-metrics*: εκθέτει μετρικές για την κατάσταση (state) των Kubernetes αντικειμένων, όπως Deployments, DaemonSets, Nodes ή Pods. Δεν μετράει κατανάλωση

πόρων, αλλά δίνει πληροφορίες για το επιθυμητό vs. πραγματικό state, επιτρέποντας την παρακολούθηση της υγείας του cluster.

Με αυτό τον τρόπο, το Prometheus πλαισιώνεται από μια σειρά εργαλείων που καλύπτουν διαφορετικές πτυχές: συλλογή μετρικών συστήματος και containers, παρακολούθηση του state των Kubernetes resources και διαχείριση ειδοποιήσεων. (Pai & Prof.Srinivas, 2024).

Πλεονεκτήματα:

- Υψηλή ευελιξία, παραμετροποίηση και δυνατότητα χρήσης της PromQL για δυναμικές επερωτήσεις
- Υποστήριξη προσωποποιημένων μετρικών (custom metrics)
- Ενσωματώνεται εύκολα σε Kubernetes περιβάλλοντα
- Προσφέρεται μεγάλη υποστήριξη από την κοινότητα (CNCF Graduated Project)[67]
- Πλούσιο οικοσύστημα γύρω από το εργαλειο

Μειονεκτήματα:

- Το default alerting σύστημα (Alertmanager) έχει περιορισμένη διεπαφή χρήστη (User Interface – UI) και δεν κλιμακώνεται εύκολα σε multi-cluster setups (Pai & Srinivas, 2024).
- Το alert deduplication είναι περιορισμένο και βασίζεται κυρίως στο Alertmanager, γεγονός που αυξάνει την πολυπλοκότητα σε μεγάλα περιβάλλοντα.
- Παρουσιάζει προβλήματα κλιμάκωσης σε μεγάλα clusters, με αποτέλεσμα αυξημένες απαιτήσεις σε αποθηκευτικό χώρο και υπολογιστικούς πόρους.
- Ενέχει σημαντική λειτουργική πολυπλοκότητα (operational complexity) όταν αναπτύσσεται σε μεγάλα setups με πολλαπλά clusters.
- Απαιτεί την εγκατάσταση επιπρόσθετων εξαγωγέων μετρικών και την υλοποίηση των custom μετρικών
- Η αποθήκευση μακροπρόθεσμων μετρικών (long-term metrics) απαιτεί εξωτερικές λύσεις, όπως το Thanos[68] (επεκτείνει το Prometheus προσφέροντας υψηλή διαθεσιμότητα (high availability), μακροπρόθεσμη αποθήκευση και παγκόσμια ενοποιημένα ερωτήματα σε πολλαπλούς Prometheus servers) ή το Cortex[69] (επιτρέπει την οριζόντια κλιμάκωση

[67] Ο χαρακτηρισμός Graduated από το Cloud Native Computing Foundation (CNCF) δηλώνει ότι το έργο έχει φτάσει σε υψηλό επίπεδο ωριμότητας, σταθερότητας και εκτεταμένης κοινοτικής υποστήριξης. Αυτό προϋποθέτει ενεργό οικοσύστημα χρηστών και συντελεστών, διαφάνεια στη διακυβέρνηση και επιτυχημένη χρήση παραγωγικά σε πραγματικούς οργανισμούς.

[68] https://thanos.io/v0.39/thanos/getting-started.md/

[69] https://cortexmetrics.io/docs/

(horizontal scaling) του Prometheus, με υποστήριξη πολυ-μισθωτικότητας (multitenancy) και αποκεντρωμένη αποθήκευση των δεδομένων) (Hadikusuma, Lukas, & Bachri, 2022)

Grafana

Το **Grafana** λειτουργεί ως σύστημα οπτικοποίησης που διασυνδέεται με Prometheus, Loki, ElasticSearch και και άλλα συστήματα αποθήκευσης μετρικών (π.χ. InfluxDB[70]) και καταχωρήσεων (logs) (πχ. ElasticSearch[71]). Παρέχει dashboards, panels και υποστήριξη alerting μέσω του UI. Σύμφωνα με την εργασία των Hämäläinen et al. (2021), η χρήση dashboarding ενισχύει την κατανόηση των δεδομένων και διευκολύνει τη διάγνωση προβλημάτων σε πραγματικό χρόνο.

Το Grafana αποτελεί web εφαρμογή οπτικοποίησης δεδομένων με υποστήριξη authentication και δυνατότητα επέκτασης μέσω plugins.Η εγκατάστασή του μπορεί να γίνει με πολλαπλούς τρόπους, όπως μέσω Helm charts, Kubernetes manifests ή του Grafana Operator, προσφέροντας δηλωτική διαχείριση των dashboards, datasources και χρηστών. Παρότι συνήθως αναπτύσσεται εντός του cluster (in-cluster, internal), το Grafana μπορεί επίσης να λειτουργήσει ως εξωτερική εφαρμογή, π.χ. μέσω Grafana Cloud (SaaS) ή standalone εκτελεση σε VM. Συνεπώς το Grafana κατατάσσεται στα Hybrid deployable εργαλεία.

Πλεονεκτήματα:

- Φιλικό περιβάλλον και πλούσια μηχανή οπτικοποίησης (visualization engine)
- Δυνατότητα δημιουργίας templayes & custom dashboards για διαφορετικές ομάδες
- Εύκολη ενσωμάτωση με Prometheus, Jaeger και Loki
- Δυνατότητα ενοποίησης πολλαπλών πηγών διαφορετικών ειδών δεδομένων (μετρικές, καταχωρήσεις, traces, κλπ.) και alerting για όλα αυτά τα είδη
- Δυνατότητα συσχέτισης μεταξύ traces και μετρικών/καταχωρήσεων
- Υποστήριξη alterting πολλαπλών καναλιών (multi-channel)
- Ύπαρξη πληθώρας από plugins και υποστήριξη από ενεργή κοινότητα

Μειονεκτήματα:

- Εξάρτηση από εξωτερικά backends για μετρικές· το Grafana δεν διαθέτει δική του βάση δεδομένων για metrics, logs ή traces. Ωστόσο, το Grafana Agent προσφέρει μια πιο ελαφριά εναλλακτική συλλογής και αποστολής δεδομένων σε σχέση με το Prometheus, μειώνοντας σε κάποιες περιπτώσεις την ανάγκη για πλήρη εγκατάσταση monitoring stack. Η διαχείριση μεγάλου αριθμού dashboards απαιτεί πρόσθετο configuration

[70] https://docs.influxdata.com/

[71] https://www.elastic.co/docs

- Αύξηση πολυπλοκότητας με την προσθήκη επιπλέον πηγών δεδομένων και dashboards
- Η σχεδίαση προηγμένων dashboards μπορεί να απαιτεί την εκμάθηση σχετικών γλωσσών επερωτήσεων με την Grafana (όπως PromQL, LogQL κ.α.)
- Δεν επιτρέπεται η χρήση επερωτήσεων πλήρους κειμένου
- Απαιτεί εξωτερική αποθήκευση traces & μετρικών
- Περιορισμένη λογική για alerts αν δεν γίνεται χρήση σχετικών εξωτερικών εργαλείων
- Ορισμένα πρόσθετα μπορεί να είναι ασταθή ή να απαιτούν συνεχώς ενημερώσεις
- Μπορεί να υπάρχει βαριά εξάρτηση από την κοινότητα για τη χρήση ορισμένων χαρακτηριστικών του Grafana

Loki

Το **Loki** είναι σύστημα log aggregation που εστιάζει στη συλλογή καταχωρήσεων (logs) από Kubernetes pods, ευθυγραμμισμένο με τα labels των workloads. Χρησιμοποιεί τον **Promtail** ως DaemonSet για να συλλέγει logs από κάθε κόμβο της συστάδας και τα αποθηκεύει σε ένα σχήμα (schema) που δεν απαιτεί full indexing, κάνοντάς το πιο ελαφρύ (Pai & Srinivas, 2024). Η εγκατάστασή του πραγματοποιείται εντός του cluster (Internal), με components που αναπτύσσονται ως Deployments ή StatefulSets και agents σε DaemonSets, ενώ υποστηρίζεται η χρήση Helm charts ή YAML manifests για τη δηλωτική εγκατάσταση.

Ενσωματώνεται φυσικά με το Grafana για προβολή logs στην ίδια διεπαφή με τις μετρικές, διευκολύνοντας τη συσχέτιση γεγονότων. Το Loki κατατάσσεται στην κατηγορία Hybrid Distributed, καθώς:

- διαθέτει κατανεμημένη αρχιτεκτονική (distributed architecture), όπου τα components του — όπως distributor, ingester, querier, query-frontend και index-gateway — εκτελούνται σε διαφορετικούς κόμβους και συνεργάζονται για τη συλλογή και ανάλυση των logs,
- και υποστηρίζει υβριδική ανάπτυξη (hybrid deployment), αφού μπορεί να λειτουργήσει εντός του cluster (in-cluster) ή εκτός αυτού (external), για παράδειγμα ως κεντρικό aggregation backend που συλλέγει δεδομένα από πολλαπλά clusters ή σε περιβάλλον SaaS.

Πλεονεκτήματα:

- Ελαφριά (Lightweight) αρχιτεκτονική, κλιμακούμενη με οριζόντιο τρόπο
- Εγγενής (Native) ενοποίηση με Grafana
- Δεν απαιτεί περίπλοκο σχήμα (δεδομένων)

Μειονεκτήματα:

- Περιορισμένες δυνατότητες αναζήτησης σε σύγκριση με ElasticSearch διότι δεν υποστηρίζεται αναζήτηση πλήρους κειμένου
- Απαιτείται χρήση εξωτερικού αποθηκευτικού μέσου ή υπηρεσίας για μεγάλης διάρκειας αποθήκευση
- Η απόδοση των επερωτήσεων μειώνεται με την αύξηση του όγκου των καταχωρήσεων

Jaeger

Το **Jaeger** είναι εργαλείο για **distributed tracing**, απαραίτητο για την αποσφαλμάτωση πολύπλοκων μικροϋπηρεσιών. Μέσω του **Jaeger Operator**, η εγκατάστασή του γίνεται δηλωτικά με χρήση CRDs, ενώ υποστηρίζει ενσωμάτωση με OpenTelemetry για auto-instrumentation (Pai & Srinivas, 2024). Η αρχιτεκτονική του είναι ευέλικτη, καθώς μπορεί να εγκατασταθεί εντός του cluster (internal) με components όπως jaeger-agent, collector και query service, ή να λειτουργήσει εκτός cluster (external) ως remote tracing backend ή SaaS υπηρεσία. Συνεπώς, το Jaeger κατατάσσεται στην κατηγορία Hybrid distributed, καθώς υποστηρίζει συνδυασμό in-cluster και external λειτουργιών και παρέχει πολλαπλούς τρόπους ενσωμάτωσης, όπως Operator, Helm, DaemonSet, Sidecar και YAML manifests

Το Jaeger κατατάσσεται στην κατηγορία Hybrid / Distributed, καθώς:

- διαθέτει κατανεμημένη αρχιτεκτονική (distributed) με επιμέρους components — όπως jaeger-agent, collector, query service και storage backend — που μπορούν να εκτελούνται σε διαφορετικούς κόμβους ή περιβάλλοντα,
- και υποστηρίζει υβριδικά σενάρια ανάπτυξης (hybrid), όπου μέρος της υποδομής βρίσκεται εντός του cluster (in-cluster) και μέρος μπορεί να λειτουργεί εκτός (external), π.χ. ως remote backend ή SaaS.

Πλεονεκτήματα:

- Αναλυτικό, κατανεμημένο tracing για αιτίες καθυστερήσεων και αποτυχιών
- Υποστήριξη για open standards (OpenTracing, OpenTelemetry)
- Υποστήριξη πολλαπλών backends αποθήκευσης (πχ. Elasticsearch, Cassandra)
- Δυνατότητα ενοποίησης με Prometheus & Grafana για κάλυψη/συσχέτιση μετρικών και οπτικοποίηση

Μειονεκτήματα:

- Απαιτεί αλλαγές στον κώδικα για χειροκίνητο instrumentation ή sidecar/agent-based instrumentation σε ορισμένες περιπτώσεις όπου το τελευταίο αυξάνει το overhead
- Υψηλές απαιτήσεις αποθήκευσης και επεξεργασίας δεδομένων

- Απαιτεί εγκατάσταση backend αποθήκευση για την επίμονη καταχώρηση των traces
- Η πολυπλοκότητα αυξάνεται με την κλίμακα

Ο παρακάτω πίνακας παρέχει μια σύνοψη των εργαλείων που αναλύθηκαν παραπάνω.

Πίνακας 11: Συγκριτική αξιολόγηση εργαλείων στην κατηγορία Monitoring, Observability & Logging

Εργαλείο	**Πυλώνας Observability**	**Πλεονεκτήματα**	**Μειονεκτήματα**
Prometheus	Συλλογή & Σύνθεση Μετρικών	Ευελιξία με PromQL, υποστήριξη custom metrics, πλούσιο οικοσύστημα exporters, ισχυρή κοινότητα	Περιορισμένο Alertmanager UI, απουσία native HA, προβλήματα κλιμάκωσης, ανάγκη εξωτερικού storage (Thanos/Cortex)
Grafana	Οπτικοποίηση & Ειδοποιήσεις	Φιλικό UI, δυναμικά dashboards, ενοποίηση πολλαπλών πηγών, alerting, πλούσιο οικοσύστημα plugins	Εξάρτηση από εξωτερικά backends (εκτος του Grafana Agent), πολυπλοκότητα διαχείρισης dashboards, περιορισμένη πολυμισθωτικότητα, αυξημένη καμπύλη εκμάθησης
Loki	Συγκέντρωση & Ανάλυση Καταγραφών	Αναλυτικό distributed tracing, υποστήριξη OpenTelemetry, πολλαπλά backends αποθήκευσης, ενοποίηση με Prometheus/Grafana	Περιορισμένη αναζήτηση (χωρίς full-text), ανάγκη εξωτερικής αποθήκευσης, μείωση απόδοσης σε μεγάλα clusters
Jaeger	Ανίχνευση Διαδρομών Αιτημάτων	Αναλυτικό distributed tracing, υποστήριξη OpenTelemetry, πολλαπλά backends αποθήκευσης, ενοποίηση με Prometheus/Grafana	Απαιτεί instrumentation, υψηλές ανάγκες αποθήκευσης/πόρων, αυξημένη πολυπλοκότητα σε μεγάλα περιβάλλοντα

4.2.2 Security

Η ασφάλεια σε περιβάλλοντα Kubernetes είναι μια πολυδιάστατη πρόκληση που απαιτεί ελέγχους σε όλα τα επίπεδα της συστάδας: από τη διαχείριση της πρόσβασης, έως τη συμμόρφωση με πολιτικές και την ανίχνευση επιθέσεων κατά την εκτέλεση. Η βιβλιογραφία αναδεικνύει ότι η

αποτελεσματική προστασία μιας συστάδας Kubernetes απαιτεί συνδυασμό **προληπτικών** και **αντιδραστικών μηχανισμών**, σε βασικές υποκατηγορίες, όπως admission policies, runtime threat detection, secrets management και vulnerability scanning (Ali και συν., 2024), (Chinamanagonda, 2021), (Amgothu & Kankanala, 2024).

Προς απάντηση στο ερευνητικό ερώτημα **RQ1**, η παρούσα ενότητα εστιάζει στα εξής βασικά εργαλεία ασφάλειας:

- **OPA Gatekeeper** για εφαρμογή πολιτικών (policy enforcement) σε επίπεδο admission,
- **Vault** για διαχείριση μυστικών και δυναμική παροχή διαπιστευτηρίων (credentials),
- **Falco** για παρακολούθηση κακόβουλης συμπεριφοράς κατά τον χρόνο εκτέλεσης (runtime behaviour monitoring / threat detection),
- **Trivy** για ανίχνευση τρωτοτήτων (vulnerability scanning) σε images και manifests.

Παράλληλα, στο πλαίσιο της **RQ2**, αναλύονται οι βασικές **λειτουργικές δυνατότητες** που προσφέρει το κάθε εργαλείο ξεχωριστά, καθώς και το είδος των απειλών ή των πολιτικών που μπορεί να καλύψει (π.χ. compliance, access control, real-time detection). Η ανάλυση επικεντρώνεται στο τί προσφέρει το κάθε εργαλείο σε επίπεδο προστασίας, τρόπου ενσωμάτωσης και τεχνικής εφαρμογής. Καλύπτει επίσης τα βασικά του πλεονεκτήματα και μειονεκτήματα.

OPA Gatekeeper

Το OPA Gatekeeper αποτελεί μηχανισμό επιβολής πολιτικών (policy enforcement) στο Kubernetes, βασισμένο στο Open Policy Agent (OPA). Η λειτουργία του βασίζεται στο ότι δρα ως admission controller: κάθε αίτημα προς το Kubernetes API server για δημιουργία ή τροποποίηση ενός πόρου ελέγχεται από το Gatekeeper πριν εφαρμοστεί στο cluster. Με αυτό τον τρόπο, αποτρέπεται η δημιουργία αντικειμένων που δεν συμμορφώνονται με τις προκαθορισμένες πολιτικές, εξασφαλίζοντας συνεχή έλεγχο συμμόρφωσης και ασφάλειας. Οι πολιτικές περιγράφονται στη γλώσσα Rego, η οποία παρέχει μεγάλη ευελιξία και εκφραστικότητα, επιτρέποντας την υλοποίηση από απλών έως και πολύπλοκων κανόνων, όπως για παράδειγμα την απαίτηση όλα τα pods να διαθέτουν συγκεκριμένα labels ή να χρησιμοποιούν images μόνο από εγκεκριμένα registries.

Η υλοποίηση των πολιτικών γίνεται με δηλωτικό τρόπο, αξιοποιώντας Custom Resource Definitions (CRDs) που εισάγει το Gatekeeper. Τα ConstraintTemplate ορίζουν τα πρότυπα πολιτικών, ενώ τα Constraint εφαρμόζουν συγκεκριμένες παραμέτρους των πολιτικών αυτών σε αντικείμενα του cluster. Εκτός από την απόρριψη μη συμβατών αιτημάτων, το Gatekeeper εκτελεί και περιοδικούς ελέγχους (audits) στα υφιστάμενα resources, εντοπίζοντας πιθανές παραβάσεις και παρέχοντας αναφορές προς τις ομάδες διαχείρισης. Επιπλέον, η ενσωμάτωση με GitOps πρακτικές επιτρέπει την αποθήκευση και παρακολούθηση των πολιτικών σε Git repositories, ώστε να υπάρχει versioning, έλεγχος αλλαγών και δυνατότητα rollback.

Όσον αφορά την ενσωμάτωση στο Kubernetes, το OPA Gatekeeper προσφέρει πολλαπλές επιλογές. Η πιο διαδεδομένη είναι η χρήση του Gatekeeper Operator, ο οποίος αυτοματοποιεί την εγκατάσταση και διαχείριση μέσω CRDs. Εναλλακτικά, διατίθεται επίσημο Helm chart που απλοποιεί την εγκατάσταση και τη συντήρηση, ενώ για πιο χειροκίνητες υλοποιήσεις υπάρχουν διαθέσιμα manifests που περιλαμβάνουν όλα τα απαραίτητα components. Σε κάθε περίπτωση, το

Gatekeeper λειτουργεί ως service-level εφαρμογή, τρέχοντας ως Deployment και webhook service στο cluster, ώστε να παρακολουθεί όλα τα admission requests και η εγκατασταση του γίνεται εντός του cluster (Internal) (Ali και συν., 2024).

Για παράδειγμα, η χρήση του στο CMSWEB cluster του CERN (Ali και συν., 2024) επέτρεψε την εφαρμογή κανόνων ελέγχου όπως node selectors, pod tolerations και networking constraints, επιβάλλοντας πλήρη ευθυγράμμιση με τις εσωτερικές πολιτικές ασφαλείας.

Πλεονεκτήματα:

- *Ευελιξία και εκφραστικότητα*: Η γλώσσα Rego επιτρέπει τη διατύπωση σύνθετων κανόνων πολιτικής με λεπτομερή έλεγχο.
- *Διαχωρισμός πολιτικής από υλοποίηση*: Οι κανόνες εκφράζονται ανεξάρτητα από τις εφαρμογές, διευκολύνοντας τη συμμόρφωση.
- *Συμβατότητα με GitOps*: Οι πολιτικές μπορούν να εκφράζονται ως κώδικας (policy as code), αποθηκευμένος σε repositories.
- *Ενσωμάτωση στο Kubernetes API*: Ως admission controller, επιτρέπει real-time έλεγχο πριν την αποδοχή αντικειμένων.
- *Κοινότητα και υποστήριξη CNCF*: Διαθέτει ισχυρή κοινότητα και χρησιμοποιείται ευρέως (π.χ. CERN)

.

Μειονεκτήματα:

- *Απότομη καμπύλη μάθησης*: Η Rego είναι λιγότερο διαδεδομένη και απαιτεί εξοικείωση.
- *Περιορισμένη παρακολούθηση παραβάσεων*: Δεν διαθέτει ενσωματωμένα dashboards· χρειάζονται εξωτερικά εργαλεία (π.χ. Grafana).
- *Πολυπλοκότητα σε μεγάλα clusters*: Η διαχείριση μεγάλου αριθμού ConstraintTemplates/Constraints μπορεί να αυξήσει το λειτουργικό βάρος.
- *Κόστος συντήρησης πολιτικών*: Απαιτεί συνεχή ενημέρωση των κανόνων καθώς αλλάζουν οι απαιτήσεις ασφάλειας..

Vault

Το Vault της HashiCorp είναι ένα εργαλείο διαχείρισης μυστικών (secrets management) και κλειδιών κρυπτογράφησης, το οποίο έχει σχεδιαστεί για να παρέχει ασφαλή αποθήκευση και δυναμική διανομή ευαίσθητων πληροφοριών, όπως tokens, passwords, κλειδιά API ή πιστοποιητικά. Η βασική του λειτουργία είναι να προστατεύει αυτά τα δεδομένα, εφαρμόζοντας ισχυρή κρυπτογράφηση τόσο κατά την αποθήκευση όσο και κατά τη μεταφορά τους, ενώ παράλληλα υποστηρίζει μηχανισμούς δυναμικών μυστικών (dynamic secrets). Αυτό σημαίνει ότι μπορεί να εκδίδει credentials τα οποία έχουν περιορισμένη διάρκεια ζωής και ανακαλούνται αυτόματα μετά τη χρήση τους, μειώνοντας τον κίνδυνο παραβίασης. Επιπλέον, το Vault ενσωματώνει μηχανισμούς auditing, ώστε να καταγράφεται ποιος και πότε έχει αποκτήσει

πρόσβαση σε συγκεκριμένα μυστικά, καθώς και υποστήριξη για έλεγχο ταυτότητας (authentication) με διάφορα συστήματα, όπως LDAP, OAuth ή Kubernetes service accounts.

Η εγκατάσταση του Vault στο Kubernetes μπορεί να γίνει με διάφορους τρόπους, ανάλογα με τις ανάγκες του οργανισμού. Η πιο διαδεδομένη μέθοδος είναι μέσω του επίσημου Helm chart της HashiCorp, το οποίο δημιουργεί όλα τα απαραίτητα components (Deployments, Services, ConfigMaps, CRDs) και προσφέρει μια σχετικά απλή και αυτοματοποιημένη διαδικασία εγκατάστασης. Εναλλακτικά, υπάρχει ο Vault Operator, που επιτρέπει πιο δηλωτική διαχείριση μέσω Custom Resource Definitions, καθιστώντας ευκολότερη την ενσωμάτωση με GitOps workflows και τη συνεχή παρακολούθηση της κατάστασης του Vault cluster.

Όσον αφορά τους τρόπους ενσωμάτωσης με τις εφαρμογές στο Kubernetes, το Vault προσφέρει πρόσθετους μηχανισμούς (add-ons) που επεκτείνουν τη λειτουργικότητά του. Ο Vault CSI driver, ο οποίος ακολουθεί το Container Storage Interface (CSI) πρότυπο, επιτρέπει την ανάκτηση μυστικών απευθείας από τον Vault και την προσάρτησή τους στα pods ως αρχεία σε volumes, χωρίς αυτά να αποθηκεύονται στο Kubernetes API ως Secrets. Με τον τρόπο αυτό μειώνεται η πιθανότητα διαρροής και εξασφαλίζεται μεγαλύτερη απομόνωση και ασφάλεια. Αντίστοιχα, το webhook injection βασίζεται σε admission webhook που παρεμβάλλεται κατά τη δημιουργία pods. Ο μηχανισμός αυτός τροποποιεί αυτόματα το pod spec ώστε να προστεθεί ένας Vault Agent ως sidecar container, ο οποίος αναλαμβάνει να ανακτήσει τα απαιτούμενα μυστικά από τον Vault server και να τα ανανεώνει περιοδικά κατά τη διάρκεια ζωής του pod. Η λύση αυτή προσφέρει πλήρως αυτοματοποιημένη και δυναμική διαχείριση μυστικών, χωρίς να απαιτείται χειροκίνητη παρέμβαση από τον χρήστη.

Με αυτό τον τρόπο, το Vault μπορεί να καλύψει πολλαπλά σενάρια χρήσης στο Kubernetes: από απλές εγκαταστάσεις με Helm έως πιο πολύπλοκες υλοποιήσεις με Operator και add-ons όπως ο CSI driver και το webhook injection, προσφέροντας ευελιξία και ενισχυμένη ασφάλεια στη διαχείριση μυστικών. Το Vault μπορεί να λειτουργήσει εξ ολοκλήρου εντός του cluster (in-cluster) ως Deployment/Service ή εν μέρει εκτός cluster (external/SaaS) μέσω του HashiCorp Cloud Platform (HCP Vault), παρέχοντας ευελιξία στην αρχιτεκτονική ασφάλειας. Συνεπώς, το Vault κατατάσσεται στην κατηγορία Hybrid (distributed), καθώς διαθέτει κατανεμημένη αρχιτεκτονική (distributed architecture).

Για παράδειγμα, το CERN cluster χρησιμοποιεί το Vault για να παρέχει dynamic secrets και να αποφεύγει hardcoded τιμές (Ali, και συν., 2024).

Πλεονεκτήματα:

- *Κεντρική διαχείριση μυστικών*: Επιτρέπει την αποθήκευση και διαχείριση όλων των tokens, κωδικών και κλειδιών από μία ενοποιημένη πλατφόρμα.
- *Δυναμικά secrets*: Δημιουργεί credentials με περιορισμένη διάρκεια ζωής (π.χ. για βάσεις δεδομένων), μειώνοντας τον κίνδυνο διαρροής.
- *Audit logs*: Διατηρεί λεπτομερές ιστορικό πρόσβασης στα μυστικά, ενισχύοντας τη συμμόρφωση και την ιχνηλασιμότητα.

- *Ενοποίηση με μηχανισμούς ταυτοποίησης*: Υποστηρίζει OAuth, LDAP, Kubernetes authentication και άλλα πρότυπα.
- *Υψηλό επίπεδο κρυπτογράφησης*: Εφαρμόζει ισχυρά πρωτόκολλα για την προστασία των δεδομένων σε ανάπαυση και σε μεταφορά.

Μειονεκτήματα

- *Υψηλή πολυπλοκότητα παραμετροποίησης*: Απαιτεί λεπτομερή ρύθμιση πολιτικών, authentication backends και agents.
- *Εξάρτηση από το control plane του Vault*: Το Vault λειτουργεί με έναν κεντρικό server (control plane του Vault), ο οποίος είναι υπεύθυνος για την αποθήκευση, κρυπτογράφηση και διανομή των μυστικών. Η διαθεσιμότητα και η αξιοπιστία αυτού του server είναι κρίσιμες· σε περίπτωση αστοχίας ή μη προσβασιμότητας, οι εφαρμογές στο Kubernetes ενδέχεται να μην μπορούν να ανακτήσουν τα απαραίτητα μυστικά, με άμεσο αντίκτυπο στη λειτουργία του
- *Αυξημένες απαιτήσεις πόρων*: Σε μεγάλα περιβάλλοντα, το Vault χρειάζεται σημαντικούς υπολογιστικούς και αποθηκευτικούς πόρους.
- *Πολυπλοκότητα ενσωμάτωσης με εφαρμογές*: Σε ορισμένα σενάρια απαιτείται αλλαγή κώδικα ή χρήση sidecar containers, αυξάνοντας το λειτουργικό κόστος.

Falco

Το Falco είναι ένα open-source εργαλείο ασφάλειας, αρχικά αναπτυγμένο από τη Sysdig και σήμερα έργο της CNCF, το οποίο ειδικεύεται στην ανίχνευση ανωμαλιών κατά το runtime σε περιβάλλοντα container και Kubernetes. Η βασική του λειτουργία στηρίζεται στην παρακολούθηση των system calls (syscalls) που εκτελούν τα containers και οι διεργασίες τους, με χρήση είτε kernel modules είτε της τεχνολογίας eBPF (extended Berkeley Packet Filter). Μέσω ενός συνόλου προκαθορισμένων ή προσαρμοσμένων κανόνων, το Falco μπορεί να αναγνωρίζει ύποπτες ή κακόβουλες ενέργειες, όπως η εκτέλεση ενός shell μέσα σε container, η προσπάθεια privilege escalation, η αλλαγή κρίσιμων system binaries ή η πρόσβαση σε ευαίσθητα paths του filesystem.

Πέρα από την απλή ανίχνευση, το Falco διαθέτει μηχανισμούς ειδοποίησης σε πραγματικό χρόνο. Οι ειδοποιήσεις μπορούν να αποσταλούν μέσω stdout, αρχείων log ή να διοχετευτούν σε εξωτερικά συστήματα (π.χ. Slack, Sysdig Secure, Elasticsearch, Prometheus, cloud-native SIEM εργαλεία) για περαιτέρω ανάλυση. Επιπλέον, οι κανόνες του είναι δηλωτικοί και γραμμένοι σε YAML, γεγονός που επιτρέπει εύκολη προσαρμογή στις ανάγκες κάθε οργανισμού, από βασικές πολιτικές συμμόρφωσης μέχρι εξελιγμένα σενάρια ανίχνευσης απειλών.

Η ενσωμάτωση του Falco σε Kubernetes γίνεται συνήθως ως DaemonSet, έτσι ώστε να τρέχει σε κάθε node του cluster και να παρακολουθεί όλους τους containers που φιλοξενούνται εκεί. Παράλληλα, υποστηρίζεται και Helm chart, που απλοποιεί την εγκατάσταση και τη συντήρηση. Το Falco μπορεί επίσης να επεκταθεί με το Falco Sidekick, μια συνοδευτική υπηρεσία που αναλαμβάνει να προωθεί alerts σε πολλαπλούς προορισμούς (π.χ. Kafka, NATS, AWS S3).

Επιπλέον, χάρη στο audit log integration, μπορεί να συνδυάσει πληροφορίες από τα Kubernetes audit logs με τα system calls, παρέχοντας πιο ολοκληρωμένη εικόνα για τις δραστηριότητες στο cluster. Το Falco κατατάσσεται στην κατηγορία Hybrid Distributed, καθώς διαθέτει κατανεμημένη (distributed) αρχιτεκτονική — με πολλαπλούς agents και συνοδευτικά components που συνεργάζονται για συλλογή και προώθηση alerts — και ταυτόχρονα υποστηρίζει υβριδική λειτουργία (hybrid), με την ικανότητα να επεκτείνει τη λειτουργία του πέρα από το cluster μέσω εξωτερικών integrations και cloud υπηρεσιών.

Έτσι, το Falco λειτουργεί ως κρίσιμος μηχανισμός runtime ασφάλειας σε Kubernetes, συμπληρώνοντας τα στατικά εργαλεία ανίχνευσης τρωτοτήτων (π.χ. Trivy) με δυναμική παρακολούθηση της συμπεριφοράς των εφαρμογών και των κοντέινερ (Chinamanagonda, 2021).

Πλεονεκτήματα:

- *Ανίχνευση σε πραγματικό χρόνο*: Παρακολουθεί system calls σε runtime και εντοπίζει ύποπτες ή κακόβουλες ενέργειες τη στιγμή που συμβαίνουν.
- *Ενσωμάτωση με Kubernetes audit logs*: Μπορεί να επεξεργαστεί συμβάντα από το Kubernetes API server για πιο ολοκληρωμένη ασφάλεια.
- *Ευελιξία κανόνων*: Οι κανόνες Falco είναι δηλωτικοί και επιτρέπουν προσαρμογή σε συγκεκριμένα σενάρια ασφαλείας.
- *Υποστήριξη eBPF*: Η χρήση eBPF μειώνει το overhead σε σχέση με kernel modules και βελτιώνει την απόδοση.
- *Πλούσιο οικοσύστημα ειδοποιήσεων*: Υποστηρίζει integration με Slack, Prometheus, Elasticsearch, Grafana κ.ά.

Μειονεκτήματα:

- *Ανάγκη για tuning*: Η σωστή ρύθμιση κανόνων απαιτεί σημαντική εμπειρία, διαφορετικά δημιουργούνται πολλά false positives.
- *Παθητική λειτουργία*: Το Falco εντοπίζει και ειδοποιεί αλλά δεν μπλοκάρει επιθέσεις· χρειάζεται συνδυασμό με άλλα εργαλεία (π.χ. OPA, Kyverno) για enforcement.
- *Απαιτήσεις σε πόρους*: Η ανάλυση system calls σε μεγάλα clusters μπορεί να επιβαρύνει τους κόμβους.
- *Πολυπλοκότητα κανόνων*: Η δημιουργία και συντήρηση custom κανόνων μπορεί να είναι δύσκολη σε μεγάλα περιβάλλοντα.
- *Εξάρτηση από kernel/eBPF*: Σε ορισμένα περιβάλλοντα η χρήση kernel modules ή eBPF δεν είναι εφικτή, περιορίζοντας την υιοθέτηση.

Trivy

Το Trivy είναι ένα open-source εργαλείο ασφάλειας που επικεντρώνεται στον έλεγχο τρωτοτήτων (vulnerability scanning) και τη διασφάλιση συμμόρφωσης σε cloud-native περιβάλλοντα. Αναπτύχθηκε από την Aqua Security και σήμερα αποτελεί ένα από τα πιο διαδεδομένα εργαλεία για container security. Η βασική του λειτουργία είναι να σαρώνει container images, file systems, Kubernetes manifests αλλά και Git repositories, εντοπίζοντας γνωστές τρωτότητες (CVEs – Common Vulnerabilities and Exposures), κακές διαμορφώσεις (misconfigurations) και αποκλίσεις από βέλτιστες πρακτικές ασφάλειας ή compliance standards (π.χ. CIS Benchmarks).

Σε επίπεδο λειτουργικότητας, το Trivy υποστηρίζει πολλαπλά modes σάρωσης: μπορεί να ελέγχει τοπικά images πριν αυτά ανέβουν σε registry, να σαρώνει online container registries, να αναλύει τα Kubernetes manifests ώστε να εντοπίσει μη ασφαλείς πρακτικές (π.χ. pods χωρίς resource limits ή containers με privileged access), καθώς και να ελέγχει dependencies εφαρμογών (π.χ. βιβλιοθήκες npm, pip, Maven). Διαθέτει ενσωματωμένη database τρωτοτήτων, η οποία ενημερώνεται συχνά με νέα CVEs, εξασφαλίζοντας έτσι επικαιρότητα στα αποτελέσματά του.

Η ενσωμάτωση του Trivy στο Kubernetes μπορεί να γίνει με διάφορους τρόπους. Μπορεί να τρέξει ως standalone CLI εργαλείο από developers ή DevOps μηχανικούς για ad-hoc ελέγχους. Υπάρχει όμως και η δυνατότητα να εγκατασταθεί στο cluster ως Kubernetes Job που εκτελείται περιοδικά για τον έλεγχο των images και των manifests. Επιπλέον, το Trivy υποστηρίζεται μέσω Helm chart, διευκολύνοντας την ανάπτυξή του ως DaemonSet ή CronJob για συνεχή παρακολούθηση. Τέλος, έχει βαθιά ενσωμάτωση με CI/CD pipelines (π.χ. GitHub Actions, GitLab CI, Jenkins), ώστε κάθε νέα έκδοση εφαρμογής να ελέγχεται αυτόματα πριν διανεμηθεί στο production (Amgothu & Kankanala, 2024). Συμπερασματικά, το Trivy κατατάσσεται στην κατηγορία Hybrid deployable, καθώς μπορεί να λειτουργεί τόσο in-cluster όσο και εκτός cluster, προσφέροντας ευελιξία ενσωμάτωσης σε όλα τα στάδια του DevSecOps κύκλου

Πλεονεκτήματα:

- *Ευκολία χρήσης*: Πολύ απλό στη ρύθμιση και εκτέλεση, με ένα μόνο binary χωρίς πολύπλοκη εγκατάσταση.
- *Πολυδιάστατος έλεγχος*: Σαρώνει container images, Kubernetes manifests, Git repositories και IaC αρχεία για τρωτότητες και misconfigurations.
- *Υποστήριξη CI/CD pipelines*: Ενσωματώνεται εύκολα σε GitHub Actions, GitLab CI, Jenkins, Azure DevOps κ.ά. για αυτόματο vulnerability scanning.
- *Ενημερωμένη βάση δεδομένων CVEs*: Χρησιμοποιεί την κοινότητα και public advisories για συχνές ενημερώσεις.
- *Ελαφρύ και γρήγορο*: Έχει χαμηλές απαιτήσεις σε πόρους και μπορεί να εκτελείται συχνά χωρίς σημαντική επιβάρυνση.
- *Open-source με ισχυρή κοινότητα*: Αναπτύσσεται ενεργά από την Aqua Security με υποστήριξη από χρήστες σε όλο τον κόσμο.

Μειονεκτήματα:

- *Δεν καλύπτει runtime activity*: Εντοπίζει τρωτότητες σε εικόνες και configs αλλά δεν παρακολουθεί την πραγματική εκτέλεση των pods.
- *Εξάρτηση από ενημερωμένες βάσεις δεδομένων*: Απαιτεί συχνά updates για να ανιχνεύει νέες τρωτότητες.
- *Πιθανά false positives/negatives*: Ορισμένες τρωτότητες μπορεί να επισημαίνονται εσφαλμένα ή να παραλείπονται.
- *Περιορισμένη εμβάθυνση σε πολύπλοκες misconfigurations*: Σε ορισμένες περιπτώσεις χρειάζεται συμπλήρωση με πιο εξειδικευμένα εργαλεία (π.χ. Checkov, Kube-bench).
- *Εξάρτηση από εξωτερικούς registries*: Η σάρωση ιδιωτικών registries απαιτεί επιπλέον ρυθμίσεις αυθεντικοποίησης.

Ο παρακάτω πίνακας παρέχει μια σύνοψη της ανάλυσης για τα 4 αυτά εργαλεία ασφάλειας.

Πίνακας 12: Συγκριτική αξιολόγηση εργαλείων στην κατηγορία Security

Εργαλείο	**Κατηγορία Ασφάλειας**	**Πλεονεκτήματα**	**Μειονεκτήματα**
OPA Gatekeeper	Policy Enforcement (Admission Policies)	Ευελιξία & εκφραστικότητα με Rego Διαχωρισμός πολιτικής/υλοποίησης Συμβατό με GitOps (policy as code) Ενσωμάτωση στο Kubernetes API (admission controller) Υιοθέτηση από ευρεία κοινότητα (CNCF)	Απότομη καμπύλη μάθησης Rego Περιορισμένα ενσωματωμένα dashboards Αυξημένη πολυπλοκότητα σε μεγάλα clusters Συνεχής συντήρηση πολιτικών
Vault	Secrets Management	Κεντρική διαχείριση μυστικών Δυναμικά secrets με περιορισμένη διάρκεια ζωής Audit logs για συμμόρφωση Ενσωμάτωση με OAuth/LDAP/Kubernetes Ισχυρή κρυπτογράφηση δεδομένων	Υψηλή πολυπλοκότητα παραμετροποίησης Εξάρτηση από το Vault server (control plane) Αυξημένες απαιτήσεις σε πόρους Δυσκολία ενσωμάτωσης σε ορισμένες εφαρμογές

Falco	Runtime Security (Threat Detection)	Ανίχνευση σε πραγματικό χρόνο (syscalls) Συμβατότητα με Kubernetes audit logs Ευέλικτοι και προσαρμόσιμοι κανόνες Υποστήριξη eBPF για καλύτερη απόδοση Ενοποίηση με εργαλεία ειδοποίησης	Υψηλή ανάγκη tuning (false positives) Παθητική λειτουργία (δεν μπλοκάρει) Απαιτήσεις σε πόρους σε μεγάλα clusters Πολυπλοκότητα δημιουργίας custom κανόνων Εξάρτηση από kernel/eBPF
Trivy	Image Scanning & Vulnerability Detection	Εύκολο στη χρήση (single binary) Υποστηρίζει images, manifests, IaC, repos Ενσωμάτωση σε CI/CD pipelines Συχνά ενημερωμένη βάση CVEs Lightweight και open-source με ενεργή κοινότητα	Δεν καλύπτει runtime activity Εξάρτηση από ενημερώσεις βάσεων δεδομένων False positives/negatives Περιορισμένη εμβάθυνση σε σύνθετα configs Επιπλέον ρυθμίσεις για private registries

4.2.3 Networking

Η διαχείριση του δικτύου αποτελεί θεμελιώδες ζήτημα για την υποδομή Kubernetes, καθώς επηρεάζει την απόδοση, την ασφάλεια και τη διαλειτουργικότητα των pods και των υπηρεσιών. Προς απάντηση του ερευνητικού ερωτήματος **RQ1**, η ενότητα αυτή επικεντρώνεται σε τρεις βασικούς πυλώνες της Kubernetes δικτύωσης:

- τα **CNI plugins**, που παρέχουν τη συνδεσιμότητα μεταξύ των pods,
- τις **Network Policies**, που ρυθμίζουν την απομόνωση και την ασφάλεια,
- και τους μηχανισμούς **Service Discovery**, που διευκολύνουν την ενδο-ομαδική επικοινωνία.

Η επιλογή των εργαλείων και τεχνολογιών για κάθε έναν από αυτούς τους πυλώνες επηρεάζει σημαντικά τόσο την απόδοση όσο και τη δυνατότητα παρακολούθησης και ελέγχου του συστήματος (Dakić, et al., 2024), (Budigiri, et al., 2021), (Erdenebat, Bud, & Kozsik, 2023). Η παρουσίαση που ακολουθεί απαντά στο ερευνητικό ερωτήμα **RQ2** και βασίζεται σε τεκμηριωμένα ευρήματα της πρόσφατης βιβλιογραφίας.

Cilium – CNI plugin με χρήση eBPF

Το Cilium αποτελεί CNI plugin, δηλαδή ενσωματώνεται στο Kubernetes ως δικτυακός μηχανισμός σε επίπεδο cluster, αντικαθιστώντας το προεπιλεγμένο CNI (π.χ. flannel, Calico). Η εγκατάστασή του γίνεται εντός του cluster (Internal) συνήθως μέσω Helm chart ή Operator (Cilium Operator), ο οποίος διαχειρίζεται αυτόματα τα network components To Cilium αξιοποιεί την τεχνολογία eBPF (extended Berkeley Packet Filter), ένα μηχανισμό εντός του Linux kernel που επιτρέπει την εκτέλεση προγραμμάτων χαμηλού επιπέδου με ασφάλεια και χωρίς ανάγκη για kernel modules. Μέσω του eBPF, το Cilium μπορεί να παρεμβαίνει στη ροή πακέτων απευθείας στον kernel, να εφαρμόζει πολιτικές και να συλλέγει μετρικές με μεγάλη αποδοτικότητα, παρακάμπτοντας μηχανισμούς, όπως τα iptables. Σύμφωνα με τη μελέτη των Miziński και Przyłucki (2025), η χρήση eBPF μειώνει το latency κατά περίπου 28% και τον φόρτο CPU σε συνθήκες έντονης δικτυακής δραστηριότητας.

Το Cilium υποστηρίζει παράλληλα προηγμένες πολιτικές δικτύωσης (network policies) αλλά και δυνατότητες observability μέσω του Hubble[72]. Το Hubble είναι ένα εργαλείο παρακολούθησης που ενσωματώνεται στο Cilium και αξιοποιεί το eBPF για να παρέχει network observability, δηλαδή ορατότητα στη ροή πακέτων, την επικοινωνία μεταξύ pods και υπηρεσιών, καθώς και εντοπισμό πιθανών προβλημάτων στο δίκτυο. Αν και συνδέεται στενά με το Cilium, μπορεί να θεωρηθεί εργαλείο observability και όχι ξεκάθαρα CNI, αφού παρέχει κυρίως λειτουργίες οπτικοποίησης και παρακολούθησης του δικτύου και λιγότερο τη βασική συνδεσιμότητα pods.

Πλεονεκτήματα:

- *Υψηλή απόδοση σε clusters με μεγάλο throughput*: Η χρήση eBPF μειώνει την εξάρτηση από iptables, επιτυγχάνοντας μικρότερο latency και χαμηλότερη κατανάλωση CPU.
- *Εγγενής υποστήριξη για eBPF observability*: Επιτρέπει real-time ορατότητα της ροής πακέτων και της επικοινωνίας pods μέσω του Hubble.
- *Συμβατότητα με μηχανές πολιτικών (policy engines)*: Υποστηρίζει λεπτομερείς network policies έως το Layer 7, συμπεριλαμβανομένων HTTP/gRPC κανόνων.
- *Ενοποίηση με Service Mesh λειτουργίες*: Προσφέρει δυνατότητες service mesh (π.χ. load balancing, mTLS) χωρίς την ανάγκη sidecar proxies, μειώνοντας overhead.
- *Ενεργό open-source οικοσύστημα*: Υποστηρίζεται από την CNCF, με μεγάλη κοινότητα και συχνές ενημερώσεις.

Μειονεκτήματα:

- *Υψηλή πολυπλοκότητα εγκατάστασης και διαχείρισης*: Απαιτεί λεπτομερή κατανόηση του eBPF και της εσωτερικής λειτουργίας του Linux kernel.
- *Απαιτήσεις Linux kernel*: Για πλήρη λειτουργικότητα του eBPF και των προηγμένων χαρακτηριστικών του Cilium, κάθε node στο cluster πρέπει να τρέχει Linux kernel έκδοσης ≥ 4.8, ενώ ορισμένες δυνατότητες απαιτούν ακόμη νεότερες εκδόσεις.

[72] https://github.com/cilium/hubble

- *Καμπύλη εκμάθησης*: Οι ομάδες που δεν είναι εξοικειωμένες με eBPF/advanced networking χρειάζονται χρόνο για εκπαίδευση.
- *Πιθανή ασυμβατότητα με legacy υποδομές*: Σε clusters με παλαιότερες διανομές ή custom kernels, η υποστήριξη eBPF μπορεί να είναι περιορισμένη.
- *Αυξημένες απαιτήσεις troubleshooting*: Τα προβλήματα στο eBPF επίπεδο συχνά είναι πιο δύσκολο να διαγνωστούν σε σχέση με παραδοσιακά iptables.

Calico – Ευέλικτο CNI με παραδοσιακή αρχιτεκτονική

Το Calico είναι ένα από τα πιο διαδεδομένα CNI (Container Network Interface) plugins για Kubernetes και άλλα cloud-native περιβάλλοντα. Ο ρόλος του είναι να παρέχει συνδεσιμότητα δικτύου μεταξύ pods και υπηρεσιών, αλλά και να υποστηρίζει Network Policies, δηλαδή κανόνες που ελέγχουν τη ροή της δικτυακής κυκλοφορίας με βάση labels, namespaces ή άλλα attributes. Στις πρώτες του εκδόσεις, το Calico βασιζόταν αποκλειστικά στο iptables για τον έλεγχο της ροής πακέτων, ενώ στις νεότερες εκδόσεις υποστηρίζει και την τεχνολογία eBPF (extended Berkeley Packet Filter), η οποία μειώνει το latency και βελτιώνει την αποδοτικότητα σε clusters με μεγάλη κλίμακα.

Λειτουργικά, το Calico προσφέρει μια σειρά από χαρακτηριστικά που το καθιστούν ελκυστικό για οργανισμούς που επιζητούν σταθερότητα και ευελιξία. Υποστηρίζει πολλαπλούς backend μηχανισμούς δικτύωσης, όπως IP-in-IP και VXLAN, για τη διασύνδεση pods σε διαφορετικούς κόμβους. Παράλληλα, διαθέτει ενσωματωμένο μηχανισμό IP Address Management (IPAM), μέσω του οποίου μπορεί να αποδίδει και να διαχειρίζεται IP διευθύνσεις με αποδοτικό τρόπο. Η συμβατότητά του με Kubernetes Network Policies είναι πλήρης, ενώ σε πιο προχωρημένα σενάρια μπορεί να επεκταθεί ώστε να υποστηρίζει Global Network Policies, εφαρμόζοντας ενιαίους κανόνες σε πολλά namespaces ή clusters. Επιπλέον, το Calico παρέχει δυνατότητες encryption για την προστασία της κυκλοφορίας (π.χ. IPsec) και προσφέρει συμβατότητα με service mesh εργαλεία, διευκολύνοντας την ενοποίηση σε πιο σύνθετες αρχιτεκτονικές.

Η ενσωμάτωση του Calico σε Kubernetes γίνεται με διάφορους τρόπους. Η πιο συνηθισμένη μέθοδος είναι η εγκατάστασή του ως CNI plugin στο cluster, κάτι που επιτυγχάνεται είτε με την απευθείας εφαρμογή manifest αρχείων που παρέχονται από το project, είτε μέσω Helm chart για ευκολότερη παραμετροποίηση. Επιπλέον, το Calico μπορεί να υλοποιηθεί με Operator-based προσέγγιση (Tigera Operator), που αυτοματοποιεί τη διαχείριση και ενημέρωσή του μέσω Custom Resource Definitions. Σε clusters με αυξημένες απαιτήσεις παρατηρησιμότητας, η λειτουργία σε eBPF mode επιτρέπει στο Calico να συλλέγει περισσότερα στοιχεία για τη ροή πακέτων και να βελτιώνει την απόδοση. Συνεπώς, το Calico κατατάσσεται στην κατηγορία Internal, καθώς λειτουργεί αποκλειστικά εντός του cluster.

Με αυτόν τον τρόπο, το Calico συνδυάζει απλότητα εγκατάστασης, ευελιξία στη διαχείριση της δικτυακής τοπολογίας και σταθερότητα, καθιστώντας το μία από τις πιο ώριμες και αξιόπιστες επιλογές για τη δικτύωση σε Kubernetes clusters. Η μελέτη των Dakić et al. (2024) κατέγραψε μικρότερο overhead σε απλά clusters και υψηλή σταθερότητα σε clusters χωρίς έντονη

κυκλοφοριακή συμφόρηση (traffic churn).

Πλεονεκτήματα:

- *Εύκολη ενσωμάτωση στο Kubernetes*: Διατίθεται ως CNI plugin που εγκαθίσταται γρήγορα με manifests ή Helm charts.
- *Συμβατότητα με Network Policies*: Υποστηρίζει πλήρως τα Kubernetes Network Policies, επιτρέποντας granular έλεγχο επικοινωνίας pods.
- *Ευκολία αποσφαλμάτωσης (debugging)*: Η χρήση iptables καθιστά πιο απλό τον εντοπισμό προβλημάτων δικτύωσης, σε σχέση με πιο πολύπλοκες λύσεις.
- *Σταθερότητα και ωριμότητα*: Χρησιμοποιείται ευρέως σε παραγωγικά clusters, θεωρείται ώριμη και αξιόπιστη λύση.
- *Χαμηλός κίνδυνος misconfiguration*: Η απλότητα της αρχιτεκτονικής μειώνει την πιθανότητα λαθών κατά τη διαμόρφωση.
- *Υποστήριξη BGP*: Παρέχει native δυνατότητες για integration με εξωτερικά δίκτυα μέσω Border Gateway Protocol.

Μειονεκτήματα:

- *Χαμηλότερη απόδοση σε σχέση με eBPF*: Όταν χρησιμοποιεί iptables, παρουσιάζει αυξημένο latency και κατανάλωση πόρων.
- *Περιορισμένη ενσωμάτωση με observability frameworks*: Δεν προσφέρει εγγενή δυνατότητα συλλογής μετρικών/ροών· χρειάζονται εξωτερικά εργαλεία.
- *Μειωμένη υποστήριξη L7 πολιτικών*: Εστιάζει κυρίως στο L3/L4 επίπεδο, με περιορισμένη λειτουργικότητα σε Layer 7 κανόνες.
- *Εξάρτηση από iptables σε παλαιότερες εκδόσεις*: Αυτό περιορίζει την απόδοση και την ευελιξία συγκριτικά με λύσεις βασισμένες εξ ολοκλήρου σε eBPF.
- *Πολυπλοκότητα σε μεγάλα multi-cluster setups*: Αν και σταθερό, η διαχείριση σε πολύ μεγάλα περιβάλλοντα μπορεί να απαιτήσει πρόσθετη υποδομή.

CoreDNS – Μηχανισμός Service Discovery

Το CoreDNS είναι ο προεπιλεγμένος μηχανισμός service discovery στο Kubernetes, αντικαθιστώντας το kube-dns από την έκδοση 1.13 και μετά. Ο ρόλος του είναι να παρέχει DNS-based name resolution για pods και services, ώστε οι εφαρμογές να μπορούν να επικοινωνούν χρησιμοποιώντας ονόματα αντί για IP διευθύνσεις. Για παράδειγμα, ένα pod μπορεί να καλέσει μια υπηρεσία my-service.default.svc.cluster.local χωρίς να χρειάζεται να γνωρίζει την εκάστοτε IP της.

Η λειτουργία αυτή είναι κρίσιμη σε Kubernetes, καθώς οι IP διευθύνσεις pods είναι εφήμερες και αλλάζουν συνεχώς.

Σε επίπεδο λειτουργικότητας, το CoreDNS είναι modular και επεκτάσιμο μέσω plugins. Κάθε plugin υλοποιεί μια λειτουργία DNS, όπως caching, forwarding, logging, health checks, ακόμα και custom κανόνες για εξωτερικά domains. Ο συνδυασμός των plugins ορίζεται στο αρχείο παραμετροποίησης Corefile, όπου καθορίζεται η ροή επεξεργασίας των DNS queries. Έτσι, το CoreDNS μπορεί να προσαρμοστεί στις ανάγκες του εκάστοτε cluster, π.χ. με ενεργοποίηση caching για μείωση latency, με forwarding σε εξωτερικούς DNS servers ή με blacklisting domains για λόγους ασφάλειας.

Η ενσωμάτωση του CoreDNS στο Kubernetes γίνεται αυτόματα, καθώς εγκαθίσταται ως Deployment στο namespace kube-system και εκτίθεται μέσω ενός ClusterIP Service που προσφέρεται σε όλα τα pods του cluster. Ωστόσο, μπορεί να εγκατασταθεί ή να αναβαθμιστεί χειροκίνητα με την εφαρμογή YAML manifests ή με τη χρήση Helm chart, παρέχοντας μεγαλύτερη ευελιξία στη διαχείριση εκδόσεων. Επιπλέον, σε πιο προχωρημένα σενάρια, το CoreDNS μπορεί να παραμετροποιηθεί με custom plugins ή με NodeLocal DNSCache, που τοποθετεί το DNS caching κοντά στα pods για βελτιωμένη απόδοση. Το CoreDNS είναι εγγενώς internal εργαλείο, καθώς εξυπηρετεί αποκλειστικά in-cluster DNS queries. Παρόλα αυτά, υποστηρίζει forwarding plugins προς external DNS servers (π.χ. Google DNS ή corporate resolvers), κάτι που επεκτείνει τη λειτουργικότητά του χωρίς να αλλάζει τον τρόπο εκτέλεσης (παραμένει internal). Η μελέτη των Erdenebat, Bud και Kozsik (2023) κατέγραψε αυξημένο latency σε clusters με πάνω από 500 pods όταν δεν εφαρμόζονται μηχανισμοί caching και custom tuning γεγονός που αναδεικνύει τη σημασία της σωστής παραμετροποίησης του CoreDNS σε μεγάλα περιβάλλοντα.

Πλεονεκτήματα:

- *Ελαφρύ και αρθρωτός σχεδιασμός (lightweight, modular design)*: Το CoreDNS έχει μικρό αποτύπωμα σε πόρους και μπορεί να επεκταθεί εύκολα με πρόσθετα (plugins) ανάλογα με τις ανάγκες του cluster.
- *Συμβατότητα με headless υπηρεσίες (headless services)*: Υποστηρίζει την απευθείας επίλυση διευθύνσεων για pods χωρίς τη χρήση load balancer, διευκολύνοντας stateful εφαρμογές.
- *Ευελιξία σε fallback και προώθηση (forwarding)*: Επιτρέπει την προώθηση DNS queries σε εξωτερικούς DNS servers ή τη χρήση fallback μηχανισμών, προσφέροντας αξιοπιστία και ανθεκτικότητα.
- *Εύκολη παραμετροποίηση*: Η διαμόρφωση μέσω του αρχείου Corefile είναι απλή και υποστηρίζει granular ρυθμίσεις.
- *Ενεργό open-source οικοσύστημα*: Υποστηρίζεται από την κοινότητα του Kubernetes και βελτιώνεται συνεχώς με νέα plugins.

Μειονεκτήματα:

- *Πιθανά bottlenecks σε μεγάλα clusters*: Χωρίς caching ή βελτιστοποίηση ρυθμίσεων μπορεί να αποτελέσει σημείο συμφόρησης σε clusters με χιλιάδες pods.

- *Αδυναμία ανίχνευσης ανενεργών endpoints (dead endpoints)*: Επιστρέφει IPs για pods ακόμη κι αν αυτά δεν είναι διαθέσιμα, εκτός αν χρησιμοποιηθούν επιπλέον μηχανισμοί ελέγχου.
- *Περιορισμένη λειτουργικότητα χωρίς plugins*: Η βασική εγκατάσταση παρέχει μόνο core DNS, ενώ προηγμένες λειτουργίες (metrics, logging) απαιτούν πρόσθετα.
- *Εξάρτηση από καλή παραμετροποίηση*: Για βέλτιστη απόδοση χρειάζεται tuning (π.χ. cache, forwarders), κάτι που αυξάνει τη διαχειριστική πολυπλοκότητα.
- *Περιορισμένη υποστήριξη advanced features*: Δεν καλύπτει εξειδικευμένα σενάρια DNS (π.χ. split-horizon DNS) χωρίς custom plugins.

Ο παρακάτω πίνακας παρέχει μια σύνοψη της ανάλυσης για τα 4 αυτά εργαλεία δικτύωσης.

Πίνακας 13: Συγκριτική αξιολόγηση εργαλείων στην κατηγορία Networking

Εργαλείο	**Κατηγορίες Networking**	**Πλεονεκτήματα**	**Μειονεκτήματα**
Cilium	Network Policies, Load Balancing, Traffic Management	Υψηλή απόδοση και μειωμένο latency μέσω eBPF, παρακάμπτει iptables μειώνοντας το CPU overhead, υποστηρίζει observability μέσω Hubble, συμβατό με L3–L7 policies (HTTP/gRPC), παρέχει service mesh δυνατότητες όπως mTLS και load balancing, προσφέρει multi-cluster λειτουργία και scalability, διαθέτει ενεργό CNCF open-source οικοσύστημα	Πολύπλοκη εγκατάσταση και διαχείριση, απαιτεί Linux kernel ≥ 4.8 για πλήρη λειτουργικότητα, έχει απότομη καμπύλη εκμάθησης λόγω eBPF, παρουσιάζει δυσκολία στο troubleshooting χαμηλού επιπέδου, και πιθανή ασυμβατότητα με legacy υποδομές
Calico	Network Policies, Load Balancing, Traffic Management	Εύκολη εγκατάσταση και ενσωμάτωση, πλήρης υποστήριξη Kubernetes Network Policies, συμβατότητα με BGP και εξωτερικά δίκτυα, απλούστερο debugging μέσω iptables, υψηλή σταθερότητα και ωριμότητα, χαμηλός κίνδυνος misconfiguration, υποστήριξη encryption (IPsec) και δυνατότητα multi-cluster λειτουργίας	Χαμηλότερη απόδοση σε iptables mode, περιορισμένο observability που απαιτεί εξωτερικά εργαλεία, εστίαση σε L3/L4 policies με ελλιπή υποστήριξη L7 κανόνων, αυξημένη πολυπλοκότητα σε μεγάλα clusters και εξάρτηση από σωστό tuning στο eBPF mode
CoreDNS	Load Balancing, Service Discovery	Lightweight και modular design με χαμηλό resource footprint, εύκολη παραμετροποίηση μέσω Corefile, υποστήριξη headless services χωρίς load balancer,	Πιθανά bottlenecks σε μεγάλα clusters χωρίς caching, αυξημένο DNS latency χωρίς tuning, περιορισμένη

		ευελιξία με plugins όπως caching, forwarding και logging, δυνατότητα fallback σε εξωτερικούς DNS servers, ενεργό CNCF open-source οικοσύστημα, προσαρμοστικότητα με custom plugins και εύκολη ενσωμάτωση στο kube-system	λειτουργικότητα χωρίς plugins, εξάρτηση από σωστή παραμετροποίηση μέσω Corefile, έλλειψη υποστήριξης split-horizon DNS χωρίς custom extensions και αδυναμία εντοπισμού ανενεργών endpoints χωρίς πρόσθετους μηχανισμούς

4.2.4 Storage, Backup & Disaster Recovery (DR)

Η υποστήριξη αξιόπιστης αποθήκευσης, λήψης αντιγράφων ασφαλείας και ανάκαμψης από καταστροφές αποτελεί αναπόσπαστο μέρος της λειτουργίας καταστατικών (stateful) εφαρμογών σε Kubernetes. Η δυναμική φύση του Kubernetes, σε συνδυασμό με τις ανάγκες για multi-cluster διαθεσιμότητα και συνεχές uptime, καθιστούν αναγκαία την ενσωμάτωση εξειδικευμένων εργαλείων για backup, restore και διαχείριση αποθηκευτικών πόρων (Kim & Jung, 2024), (Jin, Muench, Janssen, Hatfield, & Deenadhayalan, 2024), (Wen, et al., 2023).

Προς απάντηση του ερωτήματος **RQ1**, η παρούσα ενότητα αναλύει τα ακόλουθα βασικά εργαλεία και τεχνικές για τη διαχείριση της αποθήκευσης και της ανάκτησης στο Kubernetes:

- το **Velero**[73] για backup/restore,
- το **Kasten K10**[74] για προηγμένο application-level DR (disaster recovery),
- το **Ramen**[75] για αυτοματοποιημένες "DR συνταγές",
- και το **K8sES** για βελτιστοποίηση αποθήκευσης (storage) με βάση SLOs (Service Level Objectives). (Wen, et al., 2023)

Παράλληλα, στο πλαίσιο του ερευνητικού ερωτήματος **RQ2**, εξετάζονται αναλυτικά οι βασικές λειτουργικές δυνατότητες των εργαλείων αποθήκευσης, δημιουργίας αντιγράφων ασφαλείας και ανάκαμψης από καταστροφή. Η παρουσίαση περιλαμβάνει τα τεχνικά χαρακτηριστικά, τις υποστηριζόμενες μεθόδους ενσωμάτωσης στο Kubernetes (όπως CRDs, CLI, scheduling extensions), καθώς και τη συμβατότητα με cloud υποδομές ή multi-cluster περιβάλλοντα. Επίσης, περιλαμβάνει τον προσδιορισμό των πλεονεκτημάτων και μειονεκτημάτων των εργαλείων αυτών.

[73] https://velero.io/docs/v1.17/

[74] https://docs.kasten.io/6.0.8/index.html

[75] https://github.com/RamenDR

Velero

Το Velero είναι ένα open-source εργαλείο για τη λήψη και επαναφορά αντιγράφων ασφαλείας σε Kubernetes clusters, με ιδιαίτερη έμφαση στη φορητότητα και την αποκατάσταση καταστροφών (Disaster Recovery). Η βασική του λειτουργία είναι να επιτρέπει τη δημιουργία αντιγράφων ασφαλείας των Kubernetes resources (π.χ. Deployments, Services, ConfigMaps) καθώς και των persistent volumes που συνδέονται με τις εφαρμογές. Μέσω της δυνατότητας persistent volume snapshots, το Velero μπορεί να δημιουργεί στιγμιότυπα αποθηκευτικών volumes, τα οποία επιταχύνουν σημαντικά τη διαδικασία επαναφοράς.

Πέρα από την πλήρη επαναφορά ενός cluster, το εργαλείο υποστηρίζει selective restores, δηλαδή την επαναφορά μόνο συγκεκριμένων namespaces, ομάδων πόρων ή ακόμα και μεμονωμένων αντικειμένων, χωρίς να χρειάζεται να αποκατασταθεί ολόκληρο το cluster. Παράλληλα, διαθέτει δυνατότητα για scheduled backups, ώστε να εκτελούνται προγραμματισμένα αντίγραφα ασφαλείας σε τακτά χρονικά διαστήματα, μειώνοντας τον κίνδυνο απώλειας δεδομένων. Ένα ακόμη σημαντικό χαρακτηριστικό είναι η ενσωμάτωση με cloud storage buckets (AWS S3[76], GCP Storage[77], Azure Blob[78]) – δηλ. σε υπηρεσίες νέφους αποθήκευσης εγγράφων/αντικειμένων), καθώς και με S3-compatible συστήματα αποθήκευσης, γεγονός που το καθιστά ιδιαίτερα ευέλικτο και ιδανικό για cloud-native περιβάλλοντα.

Η ενσωμάτωση του Velero σε Kubernetes γίνεται με διάφορους τρόπους. Ο πιο συνηθισμένος είναι μέσω του Helm chart, που απλοποιεί την εγκατάσταση και διαχείριση του εργαλείου. Μπορεί επίσης να εγκατασταθεί με την απευθείας εφαρμογή YAML manifests που παρέχει το project, ενώ υποστηρίζει και CLI-based χρήση, με το εργαλείο γραμμής εντολών velero να χρησιμοποιείται για τη διαχείριση των backup και restore operations. Επιπλέον, το Velero αξιοποιεί server component που τρέχει ως Deployment στο cluster και plugins για cloud providers, ώστε να υποστηρίζει διαφορετικά backend αποθήκευσης. Μέσα από αυτά τα plugins μπορεί να επεκταθεί σε νέα συστήματα αποθήκευσης ή να ενσωματωθεί σε on-premises περιβάλλοντα.

Με αυτά τα χαρακτηριστικά, το Velero δεν περιορίζεται απλώς στη δημιουργία αντιγράφων ασφαλείας αλλά λειτουργεί ως πλήρης λύση για data protection και disaster recovery σε Kubernetes. Χρησιμοποιείται εκτενώς σε cloud-based υποδομές λόγω της ευκολίας χρήσης του, της επεκτασιμότητάς του και της συμβατότητάς του με S3-compatible storage, ενώ παραμένει ταυτόχρονα ελαφρύ και εύχρηστο για μικρότερες εγκαταστάσεις Συμπερασματικά, από αρχιτεκτονική άποψη, το Velero καταττάσεται στην κατηγορία Hybrid (distributed), καθώς περιλαμβάνει τόσο in-cluster components (Velero server που τρέχει ως Deployment στο cluster), όσο και external components (CLI εργαλείο και plugins για cloud storage). Χρησιμοποιείται

[76] https://aws.amazon.com/s3/

[77] https://cloud.google.com/storage/docs

[78] https://learn.microsoft.com/en-us/azure/storage/blobs/

εκτενώς σε cloud-based υποδομές λόγω ευκολίας χρήσης και συμβατότητας με S3-compatible αποθηκευτικά συστήματα (Kim & Jung, 2024).

Πλεονεκτήματα:

- *Ελαφρύ εργαλείο, εύκολη ρύθμιση μέσω CLI*: Ένα μόνο εκτελέσιμο ή binary, με λίγο configuration, που μπορεί να ξεκινήσει backup χωρίς πολύπλοκες εγκαταστάσεις.
- *Υποστήριξη για προγραμματισμένα backups (scheduled backups)*: Είναι δυνατόν να οριστούν χρονοδιαγράμματα για αυτόματα αντίγραφα ασφαλείας, ώστε να μην χρειάζεται χειροκίνητα backup κάθε φορά.
- *Δυνατότητα επαναφοράς επιμέρους πόρων (selective restore)*: Αντί να γίνει επαναφορά ολόκληρου cluster, μπορούν να επιλεγούν συγκεκριμένα namespaces, resource types ή αντικείμενα (pods, configmaps, secrets).
- *Integration με cloud storage και S3-compatible backends*: Εύκολη χρήση με Amazon S3 και άλλα συμβατά object storage, μειώνοντας το κόστος και την πολυπλοκότητα της αποθήκευσης.
- *Υποστήριξη για persistent volume snapshots*: Επιτρέπει αντιγραφή των δεδομένων της αποθήκης (storage) των εφαρμογών, χρήσιμο για stateful workloads.
- *Open-source με ενεργή κοινότητα*: Συνεχείς ενημερώσεις και βελτιώσεις, λύσεις σε κοινά προβλήματα από άλλους χρήστες.

Μειονεκτήματα:

- *Δεν παρέχει Web UI ή προκαθορισμένα πρότυπα πολιτικών (UI / policy templates)*: Όλα τα backups/restores και οι ρυθμίσεις γίνονται μέσω CLI ή YAML αρχείων — αυτό μπορεί να δυσκολεύει τη διαχείριση για χρήστες που προτιμούν γραφικό περιβάλλον.
- *Απαιτεί υψηλά permissions στο cluster*: Για να κάνει snapshots, να έχει πρόσβαση σε όλους τους persistent volumes, και γενικά να μπορεί να διαχειρίζεται πόρους υψηλού επιπέδου, χρειάζονται δικαιώματα που μπορεί να είναι ευαίσθητα από πλευράς ασφαλείας.
- *Περιορισμένο observability & auditing από προεπιλογή*: Δεν έχει ενσωματωμένο dashboard ή ισχυρά εργαλεία ειδοποίησης/καταγραφής συμβάντων — χρειάζεται να συνδυαστεί με άλλα εργαλεία (π.χ. Prometheus, Grafana).
- *Εξάρτηση από εξωτερικά συστήματα αποθήκευσης*: Αν το backend (π.χ. S3) είναι αργό ή μη αξιόπιστο, το Velero θα επιβαρυνθεί, και οι διαδικασίες backup/restore μπορεί να είναι αργές.
- *Κόστος αλλαγής και του χρόνου αποκατάστασης (restore time)*: Για πολύ μεγάλα persistent volumes, ή πολύ μεγάλο όγκο δεδομένων, το restore μπορεί να πάρει σημαντικό χρόνο και να επηρεάσει την υπηρεσία.

- *Πολυπλοκότητα σε περιβάλλοντα multi-cluster / cross-region*: Όταν τα δεδομένα ή οι πόροι είναι διασκορπισμένα ή βρίσκονται σε διαφορετικά γεωγραφικά σημεία, η διαχείριση των backups και η αποκατάσταση γίνονται πιο σύνθετες.

Kasten K10

Το Kasten K10[79] της Veeam αποτελεί μια ολοκληρωμένη πλατφόρμα προστασίας δεδομένων (data protection) και αποκατάστασης καταστροφών (disaster recovery), σχεδιασμένη εξ αρχής για Kubernetes περιβάλλοντα. Εστιάζει στην προστασία εφαρμογών σε επίπεδο Kubernetes workloads, προσφέροντας προηγμένες δυνατότητες, όπως application-consistent backups, RBAC (Role-Based Access Control), encryption, policy-based automation, UI-based διαχείριση, workload cloning, καθώς και cross-cluster recovery. Η φιλοσοφία του Kasten K10 βασίζεται στη λογική του application-centric backup, αναγνωρίζοντας και διασώζοντας όλες τις εξαρτήσεις μιας εφαρμογής (Pods, PersistentVolumeClaims, ConfigMaps, Secrets, κ.ά.) ώστε να εξασφαλίζει πλήρη συνέπεια των δεδομένων κατά τη δημιουργία και επαναφορά αντιγράφων ασφαλείας.

Η εγκατάσταση και ενσωμάτωσή του στο Kubernetes μπορεί να πραγματοποιηθεί με πολλαπλούς τρόπους. Η πιο συνηθισμένη μέθοδος είναι η χρήση του Helm chart, το οποίο εγκαθιστά τον K10 controller και τα συναφή components στο cluster, δημιουργώντας τους απαραίτητους Deployments, Services, Jobs και Custom Resource Definitions (CRDs). Εναλλακτικά, η εγκατάσταση μπορεί να γίνει χειροκίνητα μέσω YAML manifests. Ο controller του Kasten εκτελείται εντός του cluster και αναλαμβάνει τη διαχείριση των backup, restore και migration διαδικασιών, ενώ οι agents και τα sidecar containers διασυνδέονται με εφαρμογές για τη συλλογή και την κλωνοποίηση δεδομένων.

Παράλληλα, το Kasten K10 διαθέτει εξωτερικά (external) στοιχεία όπως το web-based γραφικό περιβάλλον διαχείρισης (K10 Dashboard) και REST APIs, τα οποία επιτρέπουν απομακρυσμένη πρόσβαση, παρακολούθηση και διαχείριση των εργασιών από χρήστες ή CI/CD pipelines. Επιπλέον, υποστηρίζει διασύνδεση με cloud-based object storage συστήματα όπως AWS S3, Azure Blob Storage, Google Cloud Storage και S3-compatible λύσεις, τα οποία μπορούν να χρησιμοποιηθούν ως προορισμοί για την αποθήκευση των αντιγράφων ασφαλείας. Μέσω αυτών των connectors, το Kasten K10 υποστηρίζει επίσης cross-cluster restore και application mobility, καθιστώντας δυνατή τη μεταφορά εφαρμογών και δεδομένων μεταξύ διαφορετικών clusters ή cloud providers.

Αρχιτεκτονικά, το Kasten K10 θεωρείται κατατάσσεται στην κατηγορία Hybrid (distributed), καθώς διαθέτει κατανεμημένη αρχιτεκτονική που συνδυάζει in-cluster components (controller, CRDs, agents, sidecars) με external interfaces και integrations (UI, APIs, cloud plugins). Σε επίπεδο Kubernetes, λειτουργεί ως Cluster-level εφαρμογή, ενώ η αρχιτεκτονική του υποστηρίζει επεκτασιμότητα μέσω CRDs και policy definitions για δηλωτική διαχείριση των backup πολιτικών. Η υποστήριξη CSI (Container Storage Interface) drivers επιτρέπει τη συνεργασία με πολλούς τύπους αποθηκευτικών συστημάτων, βελτιώνοντας τη φορητότητα και τη διαλειτουργικότητα.

Συνολικά, το Kasten K10 είναι ένα υβριδικό σύστημα προστασίας δεδομένων που συνδυάζει in-cluster αυτοματοποίηση με external ενοποίηση και cloud storage backend. Η υβριδική του φύση,

[79] https://docs.kasten.io/7.0.5/install/index.html

η εκτενής υποστήριξη πολλών cloud providers και η ευκολία στη διαχείριση μέσω UI και API, το καθιστούν μια από τις πιο ώριμες και πλήρεις λύσεις για backup, recovery και migration σε Kubernetes περιβάλλοντα.. Χρησιμοποιείται κυρίως σε περιβάλλοντα όπου απαιτείται βαθμωτή (granular) διαχείριση δεδομένων και συμμόρφωση (Kim & Jung, 2024).

Πλεονεκτήματα:

- *Application-aware backup*: Υποστηρίζει ειδικές λύσεις backup για βάσεις δεδομένων, όπως MongoDB και PostgreSQL, διασφαλίζοντας συνεπείς καταστάσεις (consistency) των δεδομένων κατά την δημιουργία αντιγράφων ασφαλείας.
- *Πλούσιο UI με reporting & policy controls*: Διαθέτει διεπαφή χρήστη που επιτρέπει θέαση/ορατότητα των backups, δημιουργία πολιτικών (policy), καθορισμό χρονοδιαγραμμάτων, ειδοποιήσεων και αναφορών, κάτι που διευκολύνει τη διαχείριση σε μεγάλους οργανισμούς.
- *Υποστήριξη multi-cluster Disaster Recovery (DR)*: Δυνατότητα ανάκτησης δεδομένων μεταξύ διαφορετικών clusters, επιτρέποντας στρατηγικές επαναφοράς σε περίπτωση καταστροφών ή αστοχιών ενός cluster.
- *Κρυπτογράφηση και ασφάλεια*: Παρέχει κρυπτογράφηση δεδομένων «σε ηρεμία» και κατά τη μεταφορά, καθώς και δυνατότητες ελέγχου πρόσβασης (RBAC), διασφαλίζοντας υψηλό επίπεδο ασφαλείας και συμμόρφωσης.
- *Granular control*: Επιτρέπει την επιλογή συγκεκριμένων αντικειμένων/πόρων (resources), namespaces, και εντός αυτών ρυθμίσεις πολιτικών που να καθορίζουν πόσο συχνά και ποια δεδομένα θα καλύπτονται.

Μειονεκτήματα:

- *Εμπορική άδεια με δωρεάν έκδοση*: Το Kasten K10 δεν είναι open-source· η πλήρης χρήση του απαιτεί αγορά άδειας (enterprise license). Ωστόσο, η Veeam παρέχει και μια δωρεάν έκδοση (Free Edition) με περιορισμούς σε κλίμακα και υποστήριξη, η οποία προορίζεται για αξιολόγηση ή μικρότερες υλοποιήσεις
- *Αύξηση απαιτήσεων πόρων*: Απαιτεί agents/πρακτικά pods σε κάθε namespace ή cluster component, που σημαίνει επιπλέον πόροι CPU / RAM / Δίσκου.
- *Πολυπλοκότητα σε εγκατάσταση & συντήρηση*: Σε περιβάλλοντα με πολλά clusters ή πόρους, οι πολιτικές, τα permissions και η διαχείριση πολλαπλών agents μπορεί να γίνουν σύνθετα.
- *Κόστος και εξάρτηση από υποστήριξη*: Για enterprise χαρακτηριστικά, όπως SLA, υποστήριξη καταστάσεων αστοχίας, cross-cluster DR κ.ά., η υποστήριξη από τον πάροχο (Veeam) είναι ουσιώδης — χωρίς αυτήν ενδέχεται να υπάρχουν όρια ή καθυστερήσεις.

Ramen

Το Ramen είναι ένα open-source εργαλείο για Disaster Recovery (DR) orchestration σε Kubernetes clusters, το οποίο αναπτύχθηκε στο πλαίσιο του οικοσυστήματος της Red Hat Advanced Cluster

Management (ACM)[80]. Ο βασικός του στόχος είναι να εξασφαλίσει συνεπή και προβλέψιμη επαναφορά εφαρμογών και δεδομένων σε περιπτώσεις αποτυχίας ενός cluster ή ολόκληρης υποδομής. Σε αντίθεση με πιο απλές προσεγγίσεις backup/restore, το Ramen εισάγει τη λογική των "recipes", δηλαδή συνταγών σε μορφή YAML, που περιγράφουν τη σωστή αλληλουχία αποκατάστασης των πόρων ενός Kubernetes περιβάλλοντος. Μέσα από αυτό το μηχανισμό, το εργαλείο αντιμετωπίζει το πρόβλημα της αλληλεξάρτησης μεταξύ πόρων (π.χ. CRDs, Persistent Volume Claims, StatefulSets), το οποίο σε απλοϊκά DR σενάρια συχνά οδηγεί σε αποτυχία επαναφοράς.

Σε επίπεδο λειτουργικότητας, το Ramen υποστηρίζει multi-cluster disaster recovery, επιτρέποντας σε εφαρμογές να αναπτυχθούν ξανά σε δευτερεύον cluster όταν το πρωτεύον καταστεί μη διαθέσιμο. Ενσωματώνει μηχανισμούς για synchronous και asynchronous replication αποθηκευτικών πόρων, εξασφαλίζοντας ότι τα δεδομένα παραμένουν συγχρονισμένα μεταξύ των clusters. Παράλληλα, παρέχει δυνατότητες για policy-based διαχείριση, όπου ο χρήστης μπορεί να ορίσει πολιτικές DR (π.χ. ποια namespaces προστατεύονται, ποια storage classes συμμετέχουν στην αναπαραγωγή). Με αυτό τον τρόπο, το Ramen δεν περιορίζεται σε απλή εκτέλεση backup αλλά συντονίζει το πλήρες σενάριο αποκατάστασης σε μεγάλης κλίμακας clusters.

Η ενσωμάτωση του Ramen στο Kubernetes γίνεται κυρίως μέσω Operators, οι οποίοι αξιοποιούν Custom Resource Definitions (CRDs) για να δηλώσουν πολιτικές προστασίας και συνταγές αποκατάστασης. Το εργαλείο τρέχει ως service-level εφαρμογή σε κάθε cluster, με controllers που αλληλεπιδρούν με τα storage backends για replication και failover. Επιπλέον, σε περιβάλλοντα Red Hat OpenShift, το Ramen είναι στενά ενσωματωμένο με το Advanced Cluster Management (ACM), προσφέροντας κεντρική διαχείριση πολιτικών και οπτικοποίηση της κατάστασης DR μέσω γραφικού περιβάλλοντος. Από πλευράς τρόπου εκτέλεσης, το Ramen κατατάσσεται στην κατηγορία Hybrid (distributed), καθώς διαθέτει κατανεμημένη αρχιτεκτονική — με in-cluster components (Operators, CRDs, Controllers) που υλοποιούν τις λειτουργίες DR σε επίπεδο cluster, και external components (ACM hub cluster) που αναλαμβάνουν τον κεντρικό έλεγχο και orchestration.

Έτσι, το Ramen διαφοροποιείται από άλλα εργαλεία backup/restore, καθώς δεν περιορίζεται στη λήψη αντιγράφων ασφαλείας αλλά εισάγει μηχανισμούς συντονισμένης επαναφοράς με βάση εξαρτήσεις, προσφέροντας πιο αξιόπιστες λύσεις για disaster recovery σε Kubernetes. Η μελέτη των Jin et al. (2024) δείχνει ότι τα "naive" DR σενάρια αποτυγχάνουν σε 42% των περιπτώσεων λόγω μη ελεγχόμενης εξάρτησης πόρων, την οποία το Ramen επιλύει.

Πλεονεκτήματα:

- *Δηλωτική περιγραφή DR σε YAML*: Οι πολιτικές Disaster Recovery (DR) περιγράφονται με Custom Resource Definitions (CRDs) σε YAML, γεγονός που επιτρέπει ευθυγράμμιση με GitOps πρακτικές.

[80] https://www.redhat.com/en/technologies/management/advanced-cluster-management

- *Σωστή αλληλουχία αποκατάστασης*: Εξασφαλίζει ότι οι εφαρμογές και τα δεδομένα επανέρχονται με τη σωστή σειρά, αποτρέποντας σφάλματα που θα μπορούσαν να προκύψουν από την τυχαία επαναφορά πόρων.
- *Υποστήριξη Multi-cluster DR*: Παρέχει δυνατότητα δημιουργίας στρατηγικών αποκατάστασης που καλύπτουν περισσότερα από ένα clusters, διευκολύνοντας σενάρια γεωγραφικής ανθεκτικότητας (geo-redundancy).
- *Ενοποίηση με OpenShift ACM (Advanced Cluster Management)*: Ειδικά σε Red Hat περιβάλλοντα, ενσωματώνεται με το ACM, διευκολύνοντας τη διαχείριση πολλών clusters.
- *Αυτοματισμός διαδικασιών DR*: Μειώνει την ανάγκη χειροκίνητων ενεργειών για failover ή restore, βελτιώνοντας τον χρόνο ανάκαμψης (RTO).

Μειονεκτήματα:

- *Έλλειψη γραφικής διεπαφής (Web UI)*: Η διαχείριση γίνεται αποκλειστικά μέσω YAML/CLI, κάτι που αυξάνει την πολυπλοκότητα για μη έμπειρους χρήστες.
- *Απαιτεί γνώση εξαρτήσεων μεταξύ πόρων (resource dependencies)*: Οι ομάδες πρέπει να κατανοούν σε βάθος πώς σχετίζονται οι εφαρμογές, οι βάσεις δεδομένων και τα storage components για να ορίσουν σωστές πολιτικές DR.
- *Περιορισμένη υποστήριξη κοινότητας*: Σε αντίθεση με εργαλεία όπως το Velero, το Ramen έχει μικρότερο οικοσύστημα και λιγότερα παραδείγματα χρήσης.
- *Προσανατολισμένο κυρίως σε Red Hat/ACM περιβάλλοντα*: Η στενή του ενσωμάτωση με OpenShift ACM μπορεί να περιορίσει την ευελιξία σε clusters άλλων διανομών Kubernetes.
- *Καμπύλη μάθησης*: Η κατανόηση και εφαρμογή πολιτικών DR με YAML/CRDs απαιτεί σημαντική εκπαίδευση και εξειδίκευση.

K8sES

Το K8sES (Kubernetes Enhanced Storage Service) είναι μια ερευνητική προσέγγιση που στοχεύει στη βελτίωση της διαχείρισης αποθηκευτικών πόρων στο Kubernetes, με γνώμονα τα Storage Service Level Objectives (SLOs). Σε αντίθεση με τον κλασικό Kubernetes scheduler, ο οποίος εστιάζει κυρίως σε πόρους CPU και μνήμης κατά την τοποθέτηση pods, το K8sES εισάγει έναν μηχανισμό storage-aware scheduling, λαμβάνοντας υπόψη κρίσιμους δείκτες απόδοσης αποθήκευσης, όπως latency, throughput και κόστος. Μέσω αυτής της προσέγγισης, μειώνεται ο αριθμός των SLO violations (αποτυχιών στην τήρηση συμφωνημένων στόχων απόδοσης) και βελτιστοποιείται η συνολική απόδοση σε workloads με υψηλές απαιτήσεις I/O.

Λειτουργικά, το K8sES παρακολουθεί σε πραγματικό χρόνο τις μετρικές απόδοσης των υποκείμενων storage συστημάτων (π.χ. persistent volumes) και χρησιμοποιεί αυτές τις

πληροφορίες για να καθοδηγήσει τον scheduler στην απόφαση τοποθέτησης pods. Παράλληλα, υλοποιεί δυναμική ανακατανομή αποθηκευτικών πόρων, επιτρέποντας την προσαρμογή των SLOs ανάλογα με τις μεταβαλλόμενες συνθήκες του cluster. Σύμφωνα με τα αποτελέσματα των Wen et al. (2023), η χρήση του K8sES μείωσε κατά 41% τα SLO violations και αύξησε την αποδοτικότητα σε απαιτητικούς αποθηκευτικούς φόρτους.

Η ενσωμάτωση του K8sES στο Kubernetes γίνεται μέσω της επέκτασης του scheduler με πρόσθετους controllers και agents. Οι agents αναπτύσσονται στα nodes και συλλέγουν μετρικές απόδοσης των storage subsystems, ενώ οι controllers εμπλουτίζουν τον Kubernetes control plane με πληροφορίες για την τρέχουσα κατάσταση αποθήκευσης. Τα δεδομένα αυτά αξιοποιούνται από τον enhanced scheduler για να αποφασίσει την τοποθέτηση pods με βάση όχι μόνο CPU και μνήμη, αλλά και αποθηκευτικές παραμέτρους. Το σύστημα απαιτεί, συνεπώς, μια πιο εξειδικευμένη υποδομή, καθώς βασίζεται σε agent-based monitoring και custom scheduling logic που επεκτείνει τον υπάρχοντα Kubernetes scheduler. Από άποψη τρόπου εκτέλεσης, το K8sES κατατάσσεται στην κατηγορία Hybrid (deployable), καθώς αποτελεί in-cluster επέκταση του Kubernetes scheduler (με agents και controllers), αλλά μπορεί να αναπτυχθεί αυτόνομα σε διαφορετικά clusters ή περιβάλλοντα χωρίς να εξαρτάται από central control plane. Κατά αυτόν τον τρόπο, το K8sES εισάγει μια καινοτομία στον χώρο του Kubernetes storage, μετατρέποντας τον προγραμματισμό των pods από πόρους γενικής χρήσης σε διαδικασία που λαμβάνει υπόψη και τις αποθηκευτικές απαιτήσεις των εφαρμογών, κάτι ιδιαίτερα κρίσιμο σε clusters με βάσεις δεδομένων, analytics workloads ή άλλα I/O-intensive συστήματα.. Σύμφωνα με τους Wen et al., (2023), το σύστημα μειώνει τα SLO violations κατά 41% και αυξάνει την απόδοση αποθήκευσης σε φόρτους (workloads) υψηλών απαιτήσεων.

Πλεονεκτήματα:

- *Δηλωτική διαχείριση storage SLOs*: ο χρήστης μπορεί να δηλώνει απαιτήσεις όπως χωρητικότητα, bandwidth, shareability πρямо στο YAML του pod, και το σύστημα θα προσπαθεί να τα ικανοποιήσει.
- *Δυναμική κατανομή αποθηκευτικών πόρων*: τα storage resources “χαράσσονται” (carved out) δυναμικά, χωρίς να απαιτείται προ-διαμόρφωση από τον admin για κάθε περίπλοκη ανάγκη.
- *Ενσωμάτωση στο scheduling των pods*: ο scheduler εξετάζει κόμβους & διαθέσιμα storage ταυτόχρονα ώστε να αποφασίζει πού θα εκτελεστεί το pod με βάση όλα τα criteria.
- *Monitor & αυτόματη αντίδραση*: το σύστημα παρακολουθεί τη χρήση I/O και διαθέσιμων πόρων και μπορεί να κάνει migration πόρων αν παρατηρηθούν SLO violations.
- *Αποδοτικότερη χρήση πόρων*: σε μελέτες, η k8sES παρουσιάζει καλύτερη χρήση αποθηκευτικού χώρου και bandwidth σε σχέση με το βασικό Kubernetes χωρίς αυτές τις επεκτάσεις.

Μειονεκτήματα:

- *Δεν είναι σύστημα backup*: δεν προσφέρει αντίγραφα ασφαλείας ή restore· ασχολείται μόνο με τη διαχείριση αποθηκευτικών πόρων και τη διατήρηση SLOs.

- *Περιορισμοί εφαρμογής K8sEs:* ο χρήστης/διαχειριστής θα πρέπει να γνωρίζει τις εξαρτήσεις πόρων storage / κόμβων ώστε να διαμορφώσει σωστά τις πολιτικές και να αποφύγει SLO violations.
- *Πιθανή πολυπλοκότητα στην εγκατάσταση και συντήρηση*: πρέπει να υπάρχουν modules, όπως Discovery, Monitor, Migrator, και ο scheduler να υποστηρίζει ενσωμάτωση με αυτά.
- *Περιορισμένο οικοσύστημα*: καθώς φαίνεται να είναι ακαδημαϊκή/πειραματική λύση, δεν έχει τόσο μεγάλη κοινότητα ή υποστήριξη, όπως εργαλεία που χρησιμοποιούνται ευρέως στη βιομηχανία.
- *Overhead*: επειδή παρακολουθεί I/O, κάνει dynamic allocation & migration, υπάρχει πρόσθετο φορτίο υπολογιστικό/διαχειριστικό στον cluster.

Ο παρακάτω πίνακας συνοψίζει την ανάλυση των εργαλείων της τρέχουσας κατηγορίας

Πίνακας 14: Συγκριτική αξιολόγηση εργαλείων στην κατηγορία Storage, Backup & DR

Εργαλείο	**Κατηγορίες Storage, Backup & Disaster Recovery**	**Πλεονεκτήματα**	**Μειονεκτήματα**
Velero	Persistent Storage, Backup, Disaster Recovery, Data Migration	Ελαφρύ εργαλείο, εύκολη ρύθμιση μέσω CLI. Υποστήριξη προγραμματισμένων (scheduled) backups. Δυνατότητα selective restore. Integration με S3 και cloud storage. Persistent volume snapshots. Open-source με ενεργή κοινότητα.	Χωρίς Web UI ή policy templates. Απαιτεί υψηλά permissions. Περιορισμένο observability/audit. Εξάρτηση από backend storage. Αυξημένος χρόνος restore σε μεγάλα datasets. Πολυπλοκότητα σε multi-cluster περιβάλλοντα.
Kasten K10	Όλες οι Κατηγορίες (Backup + DR Suite)	pplication-aware backup (DB consistency). Πλούσιο UI, reporting & policy control. Multi-cluster DR support. Κρυπτογράφηση & RBAC. Granular control ανά namespace και αντικείμενο.	Αυξημένες απαιτήσεις πόρων. Πολυπλοκότητα εγκατάστασης/συντήρησης. Εξάρτηση από Veeam υποστήριξη. Κόστος για enterprise λειτουργίες.
Ramen	Distaster Recovery & Failover (DR orchestration)	Δηλωτική περιγραφή DR σε YAML (GitOps συμβατότητα). Σωστή αλληλουχία επαναφοράς πόρων. Multi-cluster DR υποστήριξη. Ενοποίηση με ACM για centralized management. Αυτοματισμός failover και restore.	Χωρίς Web UI. Απαιτεί βαθιά γνώση resource dependencies. Περιορισμένη κοινότητα. Προσανατολισμένο σε Red Hat/ACM περιβάλλοντα. Καμπύλη μάθησης για YAML/CRDs.

K8sES	Persistent Storage (Storage optimization, Data Migration	Δηλωτική διαχείριση storage SLOs. Δυναμική κατανομή αποθηκευτικών πόρων. Storage-aware scheduling. Αυτόματο monitoring & migration. Βελτιστοποιημένη χρήση αποθηκευτικού χώρου.	Απαιτεί γνώση storage–node dependencies. Πολυπλοκότητα εγκατάστασης & συντήρησης. Περιορισμένο οικοσύστημα. Επιπλέον overhead λόγω monitoring & migration.

4.2.5 Continuous Integration / Continuous Delivery (CI/CD)

Η κατηγορία CI/CD περιλαμβάνει λειτουργίες, όπως η συνεχής ενσωμάτωση (CI), όπου ο τροποποιημένος πηγαίος κώδικας δοκιμάζεται και μετατρέπεται αυτόματα σε container images, η συνεχής παράδοση (Continuous Delivery) όπου γίνεται μεταφόρτωση artifacts σε container registries, και η συνεχής διάταξη (Continuous Deployment), η οποία αναλαμβάνει την αυτόματη ανάπτυξη των εφαρμογών πάνω στην υπάρχουσα υποδομή του Kubernetes cluster.. Στο πλαίσιο αυτό εντάσσεται και η προσέγγιση GitOps, η οποία αντιμετωπίζει τα manifests ως πηγαίο κώδικα και επιτρέπει τη δηλωτική και ασφαλή διαχείριση τόσο των εφαρμογών όσο και της διαμόρφωσης της συστάδας μέσω ελεγχόμενων εκδόσεων (version control). Εξίσου κρίσιμες λειτουργίες είναι η ορχήστρωση pipelines, η δυνατότητα rollback σε παλαιότερες εκδόσεις, καθώς και η ενσωμάτωση μηχανισμών παρακολούθησης και ειδοποίησης, που διασφαλίζουν τη σταθερότητα των εφαρμογών.

Στο σύγχρονο cloud-native περιβάλλον, η ανάγκη για συνεχή παράδοση και ολοκλήρωση (CI/CD) των (δοχειοποιημένων) εφαρμογών είναι ζωτικής σημασίας. Η υιοθέτηση εργαλείων και πρακτικών CI/CD επιτρέπει στις ομάδες ανάπτυξης να παραδίδουν εφαρμογές πιο γρήγορα, με μεγαλύτερη ασφάλεια και αξιοπιστία. Στο οικοσύστημα του Kubernetes, έχουν αναδειχθεί διάφορα εργαλεία που ενσωματώνουν τις αρχές του GitOps, προσφέροντας είτε push-based είτε pull-based deployment μοντέλα (Johansson, 2022) (Ramadoni, Utami, & Fatta, 2021).

Argo CD

Το Argo CD[81] αποτελεί ένα δηλωτικό, GitOps εργαλείο συνεχούς παράδοσης (CD) για Kubernetes, επιτρέποντας αυτόματη διανομή εφαρμογών με βάση το Git ως single source of truth. Βασίζεται σε pull-based μοντέλο, συγκρίνοντας την πραγματική κατάσταση της συστάδας με τη δηλωμένη κατάσταση στο Git repository, πραγματοποιώντας συγχρονισμό ή rollback ανάλογα με τις διαφορές (μεταξύ των δύο αυτών καταστάσεων) (Reddy J, Padal S, Mayan, Catharine A. & Shanthamalar, 2024)

Ο μηχανισμός του Argo CD παρακολουθεί συνεχώς το cluster και συγχρονίζει αυτόματα την τρέχουσα με την επιθυμητή κατάσταση, διορθώνοντας τυχόν αποκλίσεις. Με αυτόν τον τρόπο εξασφαλίζεται τόσο η ασφάλεια, καθώς χρησιμοποιεί pull-based αρχιτεκτονική αντί για push-

[81] https://argo-cd.readthedocs.io/en/stable/

based, όσο και η διαφάνεια, καθώς κάθε αλλαγή αποτυπώνεται στο Git και μπορεί εύκολα να ανιχνευθεί ή να αναιρεθεί (rollback) μέσω του ιστορικού commits.

Η λειτουργικότητα του Argo CD δεν περιορίζεται στον συγχρονισμό. Προσφέρει γραφική διεπαφή (Web UI) και CLI εργαλεία για τη διαχείριση εφαρμογών, ενώ υποστηρίζει πολλαπλά εργαλεία templating, όπως Helm, Kustomize και Jsonnet. Επιπλέον, παρέχει δυνατότητες για RBAC και multi-tenancy, επιτρέποντας σε πολλαπλές ομάδες να διαχειρίζονται εφαρμογές με διακριτά δικαιώματα. Μέσα από auto-sync ρυθμίσεις, health checks και hooks για προ- ή μετα-ενέργειες, το εργαλείο αυτοματοποιεί σε μεγάλο βαθμό τη διαδικασία παράδοσης.

Όσον αφορά την ενσωμάτωση στο Kubernetes, το Argo CD μπορεί να εγκατασταθεί με διάφορους τρόπους. Συνήθως υλοποιείται ως service-level εφαρμογή σε δικό του namespace (argocd), αλλά υποστηρίζεται και μέσω Helm chart, το οποίο απλοποιεί την εγκατάσταση και παραμετροποίηση. Για πιο δηλωτική διαχείριση, υπάρχει επίσης Argo CD Operator, που επιτρέπει την αξιοποίηση Custom Resource Definitions (CRDs). Παράλληλα, οι χρήστες μπορούν να το αλληλεπιδράσουν μέσω του CLI εργαλείου argocd, ενώ για πιο προχωρημένα σενάρια υπάρχει δυνατότητα multi-cluster διαχείρισης, όπου ένα κεντρικό control plane συγχρονίζει εφαρμογές σε πολλά clusters. Τέλος, το Argo CD κατατάσσεται στην κατηγορία Hybrid (distributed), καθώς διαθέτει κατανεμημένη αρχιτεκτονική που συνδυάζει in-cluster components (Controller, Operator, Service-level components) με external interfaces (UI, CLI, API) και δυνατότητα multi-cluster διαχείρισης.

Πλεονεκτήματα:

- *Χρήση Git ως single source of truth*: Όλη η κατάσταση του cluster (manifests, Helm charts) ορίζεται σε Git repository. Αυτό εξασφαλίζει auditability και ευθυγράμμιση με GitOps πρακτικές.
- *Pull-based αρχιτεκτονική (αυξημένη ασφάλεια)*: Το Argo CD "τραβάει" (pull) αλλαγές από το Git και τις εφαρμόζει στο cluster, μειώνοντας τον κίνδυνο να επιτραπεί σε εξωτερικά συστήματα να "σπρώξουν" (push) αλλαγές. Έτσι περιορίζεται η έκθεση του cluster.
- *Εύκολο rollback μέσω Git history*: Κάθε commit στο Git αντιστοιχεί σε μια πιθανή κατάσταση του cluster. Αυτό σημαίνει ότι σε περίπτωση αποτυχίας μπορείς να επιστρέψεις γρήγορα σε προηγούμενη σταθερή έκδοση.
- *Visual UI για παρακολούθηση deployments*: Διαθέτει web interface όπου φαίνονται με γραφικό τρόπο τα resources, η τρέχουσα και η επιθυμητή τους κατάσταση. Διευκολύνει το troubleshooting και τη συνεργασία των ομάδων.
- *Ενσωμάτωση με εργαλεία templating*: Υποστηρίζει Helm, Kustomize, Jsonnet, προσφέροντας ευελιξία στη διαχείριση manifests.
- *Ενεργό open-source οικοσύστημα*: Υποστηρίζεται από την CNCF (Graduated project), με ισχυρή κοινότητα και συνεχή ανάπτυξη.

Μειονεκτήματα:

- *Απαιτεί εξοικείωση με δηλωτικό συντακτικό YAML*: Παρότι βασίζεται σε απλά manifests, η καμπύλη μάθησης είναι απότομη για ομάδες που δεν έχουν εμπειρία σε declarative configuration.
- *Περιορισμένο CI integration*: Το Argo CD καλύπτει το "CD" μέρος. Για CI (build, testing) χρειάζεται ξεχωριστά εργαλεία, όπως Argo Workflows ή Jenkins.
- *Πολυπλοκότητα σε μεγάλα clusters*: Σε περιβάλλοντα με πολλά applications/repos, η διαχείριση projects, permissions και sync policies μπορεί να γίνει σύνθετη.
- *Απαιτεί πρόσθετη ρύθμιση για multi-tenancy*: Αν και υποστηρίζει RBAC, σε περιβάλλοντα με πολλούς χρήστες χρειάζεται προσεκτικός σχεδιασμός δικαιωμάτων και projects.
- *Κατανάλωση πόρων*: Το Application Controller (συστατικό του Argo CD) παρακολουθεί συνεχώς repos και cluster state, κάτι που μπορεί να επιβαρύνει σε clusters με πολλές χιλιάδες πόρους.

Jenkins X

Το Jenkins X[82] επεκτείνει το Jenkins σε Kubernetes-native περιβάλλοντα, ενσωματώνοντας GitOps πρακτικές, αυτόματες preview διανομές και υποστήριξη pipelines βασισμένων σε Kubernetes CRDs (Gupta et al., 2022).

Στα λειτουργικά του χαρακτηριστικά, το Jenkins X υιοθετεί τη λογική του GitOps, εφαρμόζοντας pull-based διανομές, όπου η κατάσταση του cluster συγχρονίζεται αυτόματα με τα manifests που αποθηκεύονται σε Git repositories. Ένα από τα πιο καινοτόμα στοιχεία του είναι η αυτόματη δημιουργία preview environments: κάθε φορά που γίνεται ένα pull request δημιουργείται αυτόματα ένα προσωρινό περιβάλλον, στο οποίο οι αλλαγές μπορούν να δοκιμαστούν με ασφάλεια πριν συγχωνευθούν στην κύρια γραμμή ανάπτυξης. Παράλληλα, το Jenkins X ενσωματώνεται στενά με εργαλεία, όπως το Helm και το Tekton, ενώ υποστηρίζει την ασφαλή διαχείριση μυστικών (Kubernetes secrets).

Η ενσωμάτωση του Tekton[83] στο Jenkins X είναι κρίσιμη, καθώς παρέχει τον υποκείμενο μηχανισμό εκτέλεσης των pipelines. Σε αντίθεση με το κλασικό Jenkins, όπου τα pipelines εκτελούνται από έναν master server με agents, στο Jenkins X τα pipelines μετατρέπονται σε Tekton pipelines, τα οποία εκτελούνται αυτόνομα από τον Kubernetes scheduler. Αυτό προσφέρει καλύτερη κλιμάκωση, δηλωτική διαχείριση μέσω GitOps και μια πλήρως cloud-native εμπειρία για τις ομάδες ανάπτυξης

[82] https://jenkins-x.io/v3/about/

[83] https://tekton.dev/docs/

Η ενσωμάτωση του Jenkins σε Kubernetes μπορεί να γίνει με διάφορες προσεγγίσεις. Η πιο απλή λύση είναι η εγκατάστασή του ως Deployment/Service μέσα στο cluster, ώστε να λειτουργεί όπως κάθε άλλη εφαρμογή. Για πιο αυτοματοποιημένη διαχείριση διατίθεται το Jenkins Helm chart, που δημιουργεί έτοιμη υποδομή με master και agents. Επιπλέον, υπάρχει το Jenkins Kubernetes plugin, το οποίο επιτρέπει τη δυναμική δημιουργία ephemeral agents σε pods, εξασφαλίζοντας καλύτερη κλιμάκωση και απομόνωση των builds. Σε πιο προχωρημένες περιπτώσεις, η ενσωμάτωση γίνεται μέσω Operators που διαχειρίζονται δηλωτικά τους Jenkins instances με CRDs. Όσον αφορά τον τρόπο εκτέλεσης, το Jenkins X κατατάσσεται στην κατηγορία Hybrid (distributed), καθώς διαθέτει κατανεμημένη αρχιτεκτονική που συνδυάζει in-cluster components (Tekton pipelines, Operators, Controllers) με external integrations (GitOps repos, CI/CD services, εξωτερική αποθήκευση, UIs/CLIs, υπηρεσίες νέφους όπως IAM, DNS, load balancing). Αν και βασίζεται σε GitOps ροές για τη δηλωτική διαχείριση εφαρμογών, η ίδια η εκτέλεση λαμβάνει χώρα εσωτερικά, χωρίς εξωτερικό control plane.

Πλεονεκτήματα:

- *Αυτόματη δημιουργία pipelines και environments*: Το Jenkins X δημιουργεί αυτόματα CI/CD pipelines για νέα projects και διαχειρίζεται environments (staging, production) χωρίς χειροκίνητη παραμετροποίηση.
- *Υποστήριξη για monorepos και microservices*: Τα *monorepos* (ενιαία αποθετήρια που περιέχουν πολλαπλά projects/υπηρεσίες) και τα microservices μπορούν να συνυπάρχουν, με το Jenkins X να διαχειρίζεται pipelines και deployments για κάθε περίπτωση. Αυτό διευκολύνει μεγάλες ομάδες που έχουν πολύπλοκες αρχιτεκτονικές.
- *GitOps-first σχεδιασμός*: Όλες οι αλλαγές καταγράφονται στο Git (GitHub, GitLab, Bitbucket), το οποίο λειτουργεί ως single source of truth, παρέχοντας auditability και εύκολα rollbacks.
- *Αυτόματη δημιουργία preview environments*: Κάθε pull request δημιουργεί προσωρινό περιβάλλον για δοκιμές, ενισχύοντας τη συνεργασία μεταξύ developers και QA.
- *Ενσωμάτωση με Tekton και Helm*: Χρησιμοποιεί Tekton pipelines για Kubernetes-native εκτέλεση, ενώ υποστηρίζει Helm charts για deployment.
- *Κοινότητα και υποστήριξη*: Διαθέτει ενεργό open-source community με συνεχή ανάπτυξη και παραδείγματα χρήσης.

Μειονεκτήματα:

- *Σύνθετη εγκατάσταση και ρύθμιση*: Η αρχική ρύθμιση απαιτεί γνώση Kubernetes, GitOps, Tekton και Helm, κάτι που αυξάνει την πολυπλοκότητα.
- *Μεγάλη καμπύλη μάθησης*: Για ομάδες που έρχονται από το παραδοσιακό Jenkins ή άλλα CI/CD εργαλεία, η αλλαγή σε declarative GitOps-based προσέγγιση μπορεί να είναι δύσκολη.
- *Πολυπλοκότητα σε μεγάλα clusters*: Η διαχείριση πολλών pipelines και environments μπορεί να γίνει δύσκολη χωρίς πρόσθετη αυτοματοποίηση.

- *Εξάρτηση από πολλά components*: Για πλήρη λειτουργία απαιτεί Tekton, Helm, Git provider integration και Kubernetes secrets, γεγονός που αυξάνει τα σημεία πιθανής αστοχίας.
- *Κατανάλωση πόρων*: Λόγω των πολλών components, μπορεί να επιβαρύνει το cluster σε CPU/RAM σε σχέση με πιο "ελαφριά" CI/CD εργαλεία

Argo Workflows

Το Argo Workflows[84] είναι ένα open-source εργαλείο που λειτουργεί ως workflow engine για Kubernetes, επιτρέποντας τον ορισμό και την εκτέλεση σύνθετων pipelines με Kubernetes-native τρόπο. Κάθε workflow ορίζεται ως Custom Resource Definition (CRD), γεγονός που σημαίνει ότι οι ροές εργασιών περιγράφονται δηλωτικά σε YAML manifests, όπως οποιοσδήποτε άλλος Kubernetes πόρος. Οι ροές αυτές αποτελούνται από βήματα (steps), τα οποία εκτελούνται σε pods μέσα στο cluster. Ένα από τα πιο ισχυρά χαρακτηριστικά του είναι η υποστήριξη Directed Acyclic Graphs (DAGs), που δίνουν τη δυνατότητα εκτέλεσης βημάτων με εξαρτήσεις, παράλληλη εκτέλεση πολλών εργασιών και conditional logic για δυναμικές αποφάσεις εντός του pipeline.

Πέρα από τη βασική εκτέλεση pipelines, το Argo Workflows προσφέρει δυνατότητες παρακολούθησης και logging σε πραγματικό χρόνο, ενσωματώνεται με artifact repositories (π.χ. MinIO, S3) για αποθήκευση ενδιάμεσων αποτελεσμάτων, και υποστηρίζει parameterization, ώστε τα workflows να γίνονται επαναχρησιμοποιήσιμα με διαφορετικά δεδομένα εισόδου. Επίσης, παρέχει native integration με εργαλεία CI/CD, όπως το Argo CD, ώστε να δημιουργείται πλήρης GitOps-based pipeline: ειδικότερα, σε μια τέτοια περίπτωση, το Argo Workflows εκτελεί CI και testing tasks και το Argo CD διαχειρίζεται τα deployments στο production.

Η ενσωμάτωση στο Kubernetes γίνεται με διάφορους τρόπους. Καταρχάς, το Argo Workflows εγκαθίσταται στο cluster ως Operator, το οποίο εισάγει τα απαραίτητα CRDs και controllers. Η εγκατάσταση μπορεί να πραγματοποιηθεί είτε μέσω Helm chart, είτε με την απευθείας εφαρμογή των YAML manifests από το project. Οι χρήστες μπορούν να αλληλεπιδρούν με το εργαλείο είτε μέσω CLI (argo command-line tool) είτε μέσω του Web UI που παρέχει, το οποίο δίνει οπτική αναπαράσταση των workflows και της προόδου τους. Επιπλέον, χάρη στη συμβατότητά του με Kubernetes RBAC, μπορεί να ενσωματωθεί εύκολα σε multi-tenant clusters με διαχωρισμό δικαιωμάτων.

Όσον αφορά την κατηγορία τρόπου εκτέλεσης, το Argo Workflows κατατάσσεται στην κατηγορία Hybrid (distributed). Αυτό οφείλεται στο γεγονός ότι όλα τα κύρια συστατικά του (controllers, pods, executors) εκτελούνται εντός του Kubernetes cluster, αξιοποιώντας τους εγγενείς μηχανισμούς του για scheduling, scaling και resource management, ενώ η διαχείριση και η αλληλεπίδραση με τα workflows πραγματοποιείται και εκτός του cluster μέσω του Web UI, του CLI εργαλείου (argo) και των APIs. Συνεπώς, το Argo Workflows διαθέτει κατανεμημένη αρχιτεκτονική, όπου ο πυρήνας της εκτέλεσης βρίσκεται στο cluster, αλλά η παρακολούθηση και

[84] https://argoproj.github.io/workflows/

ο έλεγχος μπορούν να γίνονται εξωτερικά καθώς και υπάρχει διασύνδεση με εξωτερικά συστήματα, όπως artifact stores, Git repositories ή monitoring εργαλεία.

Με αυτόν τον τρόπο, το Argo Workflows λειτουργεί ως μια ισχυρή Kubernetes-native πλατφόρμα για CI pipelines, batch επεξεργασία δεδομένων, ML workflows και testing pipelines, επεκτείνοντας τις δυνατότητες του Kubernetes πέρα από το απλό deployment εφαρμογών (Reddy J και συν., 2024).

Πλεονεκτήματα

- *CI/Workflow orchestration με Kubernetes-native τρόπο*: όλα τα workflows ορίζονται ως Kubernetes πόροι (CRDs) και εκτελούνται ως pods, γεγονός που ενσωματώνει απόλυτα τη λειτουργία τους με το cluster και επιτρέπει φυσική κλιμάκωση.
- *Στενή ενοποίηση με Argo CD*: το Argo Workflows μπορεί να συνδυαστεί με το Argo CD, δημιουργώντας μια πλήρη GitOps ροή από την ανάπτυξη έως την παράδοση (CI/CD). Αυτό διευκολύνει ιδιαίτερα τις ομάδες που υιοθετούν GitOps πρακτικές.
- *Υποστήριξη MLOps*: η δυνατότητα εκτέλεσης DAG pipelines με παραμετροποίηση και αποθήκευση artifacts το καθιστά κατάλληλο για workflows μηχανικής μάθησης (π.χ. training, testing, deployment μοντέλων).
- *Παράλληλη και υπό όρους εκτέλεση (conditional execution)*: μέσω DAGs επιτρέπει την αποδοτική κατανομή εργασιών, με μείωση χρόνου εκτέλεσης σε μεγάλα pipelines.
- *Ευελιξία και επαναχρησιμοποίηση*: τα workflows μπορούν να παραμετροποιηθούν και να επαναχρησιμοποιηθούν με διαφορετικά δεδομένα εισόδου, μειώνοντας το κόστος ανάπτυξης νέων pipelines.
- *Οπτικοποίηση και παρακολούθηση*: μέσω του Web UI παρέχεται οπτική αναπαράσταση της πορείας εκτέλεσης, διευκολύνοντας τον εντοπισμό αποτυχιών ή bottlenecks.

Μειονεκτήματα

- *Απουσία built-in delivery μηχανισμού*: το Argo Workflows επικεντρώνεται αποκλειστικά στη φάση CI/εκτέλεσης workflows και δεν παρέχει από μόνο του μηχανισμό συνεχούς παράδοσης (CD), γεγονός που απαιτεί συμπληρωματική χρήση εργαλείων, όπως το Argo CD.
- *Απαιτεί scripting για κάθε βήμα*: κάθε βήμα του workflow πρέπει να οριστεί ως containerized task (π.χ. script ή image), κάτι που αυξάνει τον φόρτο ανάπτυξης για σύνθετους αγωγούς (pipelines).
- *Καμπύλη μάθησης*: απαιτεί εξοικείωση με τα CRDs και τη YAML σύνταξη, ενώ η μοντελοποίηση σύνθετων DAGs μπορεί να είναι δύσκολη για μη έμπειρους χρήστες.
- *Ανάγκη για πρόσθετη υποδομή αποθήκευσης*: η διαχείριση artifacts και logs απαιτεί την ενσωμάτωση με object storage ή άλλες υποδομές, κάτι που αυξάνει την πολυπλοκότητα.

- *Περιορισμένη λειτουργικότητα σε multi-cluster περιβάλλοντα*: η εγγενής υποστήριξη είναι εστιασμένη σε single-cluster σενάρια, οπότε χρειάζονται πρόσθετες λύσεις για ομοσπονδιακή διαχείριση

Drone CI

Το Drone CI[85] είναι ένα ελαφρύ, container-native σύστημα συνεχούς ολοκλήρωσης (Continuous Integration) που έχει σχεδιαστεί να εκτελεί pipelines πλήρως βασισμένα σε containers. Σε αντίθεση με παραδοσιακά CI εργαλεία όπως το Jenkins, τα pipelines του Drone CI περιγράφονται δηλωτικά σε YAML αρχεία και κάθε βήμα εκτελείται ως αυτόνομο container. Αυτό προσφέρει υψηλή φορητότητα και απομόνωση, καθώς τα βήματα μπορούν να χρησιμοποιούν οποιαδήποτε Docker image ως runtime περιβάλλον, χωρίς πρόσθετη εγκατάσταση εξαρτήσεων.

Σε επίπεδο λειτουργικότητας, το Drone CI υποστηρίζει GitOps πρακτικές, καθώς ενεργοποιεί pipelines με βάση γεγονότα από Git repositories (π.χ. push, pull request), διατηρώντας το Git ως κεντρικό σημείο ελέγχου. Παρέχει δυνατότητες για parallel και conditional execution, secrets management μέσω Kubernetes ή εξωτερικών συστημάτων (Vault, SOPS), καθώς και εκτενή υποστήριξη μέσω plugins που επεκτείνουν τις δυνατότητες του (π.χ. ειδοποιήσεις σε Slack, deployments σε Kubernetes, σάρωση τρωτοτήτων). Επιπλέον, λόγω της container-based αρχιτεκτονικής του, επιτρέπει την εύκολη επαναχρησιμοποίηση βημάτων και τη σύνθεση pipelines με μικρό λειτουργικό αποτύπωμα.

Η ενσωμάτωση του Drone CI σε Kubernetes μπορεί να γίνει με διάφορους τρόπους. Το πιο συνηθισμένο σενάριο είναι η ανάπτυξή του ως service-level εφαρμογή μέσα στο cluster, με χρήση Helm chart ή Kubernetes manifests. Σε αυτό το μοντέλο, ο Drone server λειτουργεί ως control plane για τα pipelines, ενώ τα βήματα εκτελούνται ως Kubernetes pods ή containers σε agents (runners). Υποστηρίζεται επίσης η ενσωμάτωση με Kubernetes secrets για ασφαλή διαχείριση credentials, καθώς και η σύνδεση με GitHub, GitLab ή Bitbucket για αυτοματοποιημένα triggers. Χάρη στην απλότητα εγκατάστασης και την εγγενή container-based φιλοσοφία του, το Drone CI χρησιμοποιείται συχνά σε μικρότερες ή cloud-native ομάδες ανάπτυξης που θέλουν ένα lightweight αλλά ισχυρό εργαλείο C (Reddy Jκαι συν., 2024).

Όσον αφορά τον τρόπο εκτέλεσης, το Drone CI κατατάσσεται στην κατηγορία Hybrid (distributed), καθώς διαθέτει κατανεμημένη αρχιτεκτονική όπου το control plane (Drone server) μπορεί να εκτελείται εκτός του cluster, ενώ οι agents και τα pipelines λειτουργούν εντός του cluster. Με αυτόν τον τρόπο συνδυάζει in-cluster execution με external orchestration, επιτυγχάνοντας ισορροπία ανάμεσα σε αυτονομία, ασφάλεια και φορητότητα

Πλεονεκτήματα

- *Container-native σχεδιασμός*: όλα τα βήματα ενός pipeline τρέχουν σε containers, άρα υπάρχει πλήρης απομόνωση, φορητότητα και ανεξαρτησία από το υποκείμενο σύστημα.

[85] https://docs.drone.io/

- *Ελαφριά αρχιτεκτονική*: μικρό αποτύπωμα, χωρίς ανάγκη για βαρύ control plane όπως στο Jenkins. Ιδανικό για μικρές ομάδες ή cloud-native περιβάλλοντα.
- *Εύκολη εγκατάσταση*: μπορεί να στηθεί γρήγορα με Docker, Helm ή Kubernetes manifests, σε αντίθεση με άλλα πιο σύνθετα CI εργαλεία.
- *Ενσωμάτωση με Git providers*: υποστηρίζει GitHub, GitLab, Bitbucket και άλλες πλατφόρμες για triggers, εναρμονιζόμενο με GitOps πρακτικές.
- *Ευελιξία μέσω plugins*: διαθέτει μεγάλο αριθμό community plugins για build, deploy, notifications, vulnerability scanning, κ.λπ., που καλύπτουν πολλά σενάρια.
- *Ασφάλεια*: υποστήριξη για secrets management (μέσω Kubernetes secrets, Vault κ.ά.), περιορισμός δικαιωμάτων containers, και απομόνωση pipelines.

Μειονεκτήματα

- *Περιορισμένα templates και pipelines features*: σε σχέση με εργαλεία όπως το Jenkins X, το Drone έχει πιο απλοϊκή YAML περιγραφή, με λιγότερες δυνατότητες για reuse και abstraction.
- *Όχι GitOps-first προσέγγιση*: ενώ υποστηρίζει triggers από Git, δεν έχει εγγενή GitOps φιλοσοφία όπως το Argo CD ή το Jenkins X (π.χ. δηλωτική κατάσταση cluster).
- *Περιορισμένη κοινότητα σε σχέση με Jenkins/GitLab CI*: λιγότερο ώριμο οικοσύστημα, μικρότερη βάση χρηστών και λιγότερη τεκμηρίωση.
- *Απουσία ισχυρού UI*: βασίζεται κυρίως σε απλό web interface· η οπτικοποίηση pipelines είναι πιο περιορισμένη από εργαλεία όπως Argo Workflows ή GitLab CI.
- *Κλιμάκωση*: αν και lightweight, σε πολύ μεγάλα clusters ή οργανισμούς μπορεί να απαιτεί προσαρμογές και δεν είναι τόσο αποδοτικό όσο enterprise-grade CI/CD λύσεις.
- *Ανάγκη για scripting*: κάθε βήμα πρέπει να οριστεί ως containerized task· αυτό παρέχει ευελιξία αλλά αυξάνει την πολυπλοκότητα για σύνθετες ροές εργασιών

GoCD

Το GoCD[86] είναι ένα open-source εργαλείο συνεχούς ολοκλήρωσης και παράδοσης (CI/CD) που δίνει ιδιαίτερη έμφαση στη διαχείριση εξαρτήσεων (dependency tracking) και στην οπτικοποίηση των pipelines. (Gupta et al., 2022). Σε αντίθεση με πιο απλά CI εργαλεία, το GoCD επιτρέπει τη δημιουργία πολύπλοκων pipelines, στα οποία διαφορετικά stages και jobs μπορούν να εξαρτώνται μεταξύ τους με δηλωτικό τρόπο. Το χαρακτηριστικό αυτό καθιστά το GoCD ιδιαίτερα χρήσιμο για

[86] https://docs.gocd.org/current/

μεγάλα έργα όπου οι ομάδες χρειάζονται σαφή εικόνα των σχέσεων μεταξύ των βημάτων (π.χ. build → test → deploy) και των ροών που τα συνδέουν.

Λειτουργικά, το GoCD παρέχει ένα πλούσιο Web UI για την οπτικοποίηση pipelines, την παρακολούθηση εκτελέσεων και την εύκολη αναγνώριση bottlenecks. Υποστηρίζει parallel και sequential execution των jobs, δυνατότητες artifact management για την αποθήκευση και μεταφορά παραγόμενων αποτελεσμάτων ανάμεσα σε stages, καθώς και parameterization για την επαναχρησιμοποίηση pipelines με διαφορετικές ρυθμίσεις. Ενσωματώνει μηχανισμούς role-based access control (RBAC), επιτρέποντας τον έλεγχο πρόσβασης ανά ομάδα, και προσφέρει audit logs για λόγους συμμόρφωσης.

Η ενσωμάτωση του GoCD στο Kubernetes μπορεί να γίνει με διάφορους τρόπους. Το εργαλείο αποτελείται από δύο βασικά components: τον GoCD server (control plane) και τους GoCD agents (workers που εκτελούν jobs). Σε Kubernetes, αυτά τα components μπορούν να αναπτυχθούν ως Deployments (server + agents) με τη βοήθεια Helm charts ή YAML manifests. Οι agents μπορούν να κλιμακώνονται δυναμικά (π.χ. μέσω Kubernetes autoscaling), γεγονός που επιτρέπει στο GoCD να εκτελεί παράλληλα μεγάλο αριθμό jobs. Επιπλέον, το GoCD υποστηρίζει container-based pipelines, όπου κάθε βήμα μπορεί να τρέχει μέσα σε Docker container, ενισχύοντας έτσι την απομόνωση και τη φορητότητα. Μέσω integration plugins, μπορεί επίσης να συνδεθεί με Kubernetes για αυτόματα deployments, επιτρέποντας τη δημιουργία ολοκληρωμένων CI/CD pipelines από το build μέχρι την παραγωγή.

Όσον αφορά τον τρόπο εκτέλεσης, το GoCD κατατάσσεται στην κατηγορία Hybrid (distributed), καθώς διαθέτει κατανεμημένη αρχιτεκτονική με external control plane (GoCD Server) και in-cluster execution plane (GoCD Agents). Παρότι μπορεί να λειτουργεί πλήρως μέσα στο Kubernetes (Internal), η αρχιτεκτονική του —με διαχωρισμό control plane (Server) και execution plane (Agents)— επιτρέπει υβριδικά σενάρια, όπου ο server παραμένει εξωτερικός και οι agents εκτελούνται εσωτερικά στο cluster. Αυτή η προσέγγιση προσφέρει συνδυασμό κεντρικής διαχείρισης και cloud-native εκτέλεσης.

Με αυτό τον τρόπο, το GoCD συνδυάζει τη δυνατότητα διαχείρισης σύνθετων pipelines με ενσωμάτωση στο Kubernetes οικοσύστημα, προσφέροντας στους χρήστες καλύτερη ορατότητα, έλεγχο και ευελιξία στη συνεχή παράδοση λογισμικού.

Πλεονεκτήματα

- *Σαφής και πλούσια οπτικοποίηση των pipelines*: το GoCD παρέχει ένα ισχυρό Web UI που δίνει λεπτομερή εικόνα για τα stages, τα jobs και τις μεταξύ τους εξαρτήσεις, διευκολύνοντας την κατανόηση και τη διαχείριση πολύπλοκων pipelines.
- *Value Stream Mapping (VSM)*: παρέχει πλήρη οπτικοποίηση της διαδρομής του κώδικα, από το αρχικό commit στο Git μέχρι την ανάπτυξή του σε production, επιτρέποντας τον εύκολο εντοπισμό καθυστερήσεων ή σημείων συμφόρησης (bottlenecks)
- *Ισχυρή διαχείριση εξαρτήσεων (dependency management)*: υποστηρίζει δηλωτικό καθορισμό σχέσεων μεταξύ stages, αποτρέποντας την εκτέλεση εργασιών σε λάθος σειρά και μειώνοντας τον κίνδυνο σφαλμάτων.

- *Artifact management*: δίνει τη δυνατότητα αποθήκευσης, μεταφοράς και επαναχρησιμοποίησης artifacts μεταξύ διαφορετικών σταδίων του pipeline.
- *Ευελιξία εκτέλεσης*: υποστηρίζει τόσο παράλληλη όσο και σειριακή εκτέλεση jobs, επιτρέποντας βελτιστοποίηση χρόνου και πόρων.
- *Role-Based Access Control (RBAC)*: δυνατότητα διαχείρισης δικαιωμάτων πρόσβασης ανά ομάδα χρηστών, κατάλληλο για μεγάλους οργανισμούς.
- *Επαναχρησιμοποίηση pipelines*: τα pipelines μπορούν να παραμετροποιηθούν και να χρησιμοποιηθούν ξανά με διαφορετικά δεδομένα ή configurations.

Μειονεκτήματα

- *Έλλειψη pull-based GitOps προσέγγισης*: βασίζεται κυρίως σε push-based pipelines, κάτι που το καθιστά λιγότερο εναρμονισμένο με σύγχρονες GitOps πρακτικές.
- *Μεγαλύτερη πολυπλοκότητα στη ρύθμιση*: η αρχική παραμετροποίηση και η δημιουργία πολύπλοκων pipelines μπορεί να είναι δύσκολη, ιδιαίτερα για νέους χρήστες.
- *Περιορισμένη κοινότητα σε σχέση με άλλα CI/CD εργαλεία*: αν και open-source, η βάση χρηστών του είναι μικρότερη από αυτή του Jenkins ή του Argo, με αποτέλεσμα λιγότερα διαθέσιμα plugins και παραδείγματα.
- *Μικρότερη ευελιξία σε container-native workflows*: αν και υποστηρίζει Docker containers, δεν έχει σχεδιαστεί εξαρχής ως container-native εργαλείο, όπως το Argo Workflows ή το Drone CI.
- *Υψηλότερο λειτουργικό overhead*: η ανάγκη για GoCD server και agents μπορεί να αυξήσει την πολυπλοκότητα σε σύγκριση με πιο lightweight λύσεις.

Ο παραπάτω πίνακας συνοψίζει την ανάλυση των 5 αυτών εργαλείων της τρέχουσας κατηγορίας.

Πίνακας 15: Συγκριτική αξιολόγηση εργαλείων στην κατηγορία CI/CD

Εργαλείο	**Μοντέλο**	**Κατηγορίες CI/CD**	**Πλεονεκτήματα**	**Μειονεκτήματα**
Argo CD	Pull-based	Continuous Deployment / Delivery GitOps Frameworks	GitOps προσέγγιση, ασφάλεια (pull-based), rollbacks μέσω Git history, οπτικό UI και CLI, ενσωμάτωση με Helm/Kustomize/Jsonnet, υποστήριξη multi-tenancy, ενεργή κοινότητα CNCF	Δεν καλύπτει CI μέρος, απαιτεί εμπειρία σε YAML, σύνθετη διαχείριση μεγάλων clusters, προσεκτικό RBAC σχεδιασμό, αυξημένη κατανάλωση πόρων
Argo Workflows	–	Workflow Orchestration	Kubernetes-native orchestration, DAG logic,	Δεν παρέχει CD, απαιτεί scripting

			παράλληλη/conditional εκτέλεση, MLOps υποστήριξη, στενή ενσωμάτωση με Argo CD, parameterization, Web UI με οπτικοποίηση και logs	για κάθε βήμα, αυξημένη πολυπλοκότητα σε DAGs, ανάγκη για object storage, περιορισμένη multi-cluster υποστήριξη
Jenkins X	Pull-based	Continuous Integration Pipelines Continuous Deployment / Delivery GitOps Frameworks	Αυτόματη δημιουργία pipelines/environments, υποστήριξη για monorepos και microservices, GitOps-first σχεδίαση, preview environments, ενσωμάτωση με Tekton/Helm, ενεργή κοινότητα	Σύνθετη εγκατάσταση, μεγάλη καμπύλη μάθησης, εξάρτηση από Tekton/Helm/Git integrations, υψηλή κατανάλωση πόρων, πολυπλοκότητα σε μεγάλα clusters
Drone	Push-based	Continuous Integration Pipelines Continuous Deployment / Delivery	Container-native, lightweight αρχιτεκτονική, εύκολη εγκατάσταση (Helm/Docker), ενσωμάτωση με GitHub/GitLab/Bitbucket, πολλά plugins, secrets management, ασφαλές isolation	Όχι GitOps-first, περιορισμένα pipeline templates, μικρότερη κοινότητα, βασικό UI, ανάγκη για scripting, περιορισμένη κλιμάκωση σε πολύ μεγάλα περιβάλλοντα
GoCD	Push-based	Continuous Integration Pipelines Continuous Deployment / Delivery	Οπτικοποίηση pipelines (VSM), dependency tracking, artifact management, RBAC, παράλληλη/σειριακή εκτέλεση, επαναχρησιμοποίηση pipelines, auditability	Μη GitOps (push-based), πολύπλοκη παραμετροποίηση, μικρότερη κοινότητα, όχι πλήρως container-native, υψηλό overhead (server/agents), καμπύλη μάθησης

4.2.6 Cluster Management (CLI/UX Tools)

Η κατηγορία Cluster Management αφορά εργαλεία που υποστηρίζουν τη διαχείριση και την αλληλεπίδραση με τα Kubernetes clusters,. Στο πλαίσιο αυτό, η λειτουργικότητα της κατηγορίας περιλαμβάνει διαφορετικές διαστάσεις: εργαλεία γραμμής εντολών (CLI), όπως το kubectl και το K9s, τα οποία παρέχουν άμεσο και λεπτομερή έλεγχο των πόρων· εργαλεία με γραφικές διεπαφές (GUI), όπως το Portainer, που προσφέρουν πιο φιλικό περιβάλλον για τη διαχείριση clusters, μειώνοντας την εξάρτηση από εξειδικευμένες γνώσεις CLI· AI-based βοηθούς, όπως το K8sGPT, οι οποίοι αξιοποιούν τεχνητή νοημοσύνη για την αυτόματη διάγνωση και επίλυση προβλημάτων· και τέλος, πλατφόρμες multi-cluster και lifecycle management, όπως το SUSE Rancher, οι οποίες επιτρέπουν την ενοποιημένη δημιουργία, παρακολούθηση και διαχείριση πολλαπλών clusters σε διαφορετικά περιβάλλοντα (cloud ή on-premises). Με αυτό τον τρόπο, η κατηγορία Cluster Management καλύπτει τις ανάγκες τόσο μεμονωμένων χρηστών όσο και μεγάλων οργανισμών, παρέχοντας εργαλεία που εκτείνονται από απλές εντολές μέχρι ολοκληρωμένα περιβάλλοντα διαχείρισης

kubectl

Το kubectl[87] είναι το επίσημο εργαλείο γραμμής εντολών (CLI) του Kubernetes, και αποτελεί τον κύριο μηχανισμό αλληλεπίδρασης του χρήστη με τον Kubernetes API Server (Kubernetes.io, 2024). Μέσω του kubectl, οι διαχειριστές και οι προγραμματιστές μπορούν να εκτελούν όλες τις βασικές λειτουργίες διαχείρισης clusters, όπως η δημιουργία, ενημέρωση και διαγραφή Kubernetes αντικειμένων (pods, deployments, services, configmaps, secrets). Παράλληλα, υποστηρίζει την εκτέλεση ad-hoc εντολών για debugging (π.χ. kubectl logs, kubectl exec), την παρακολούθηση της κατάστασης του cluster (kubectl get, kubectl describe) καθώς και πιο σύνθετες λειτουργίες όπως scaling, rolling updates και manual rollbacks.

Λειτουργικά, το kubectl βασίζεται σε δηλωτικά manifests σε YAML ή JSON, τα οποία εφαρμόζει απευθείας στο cluster μέσω του Kubernetes API. Αυτή η προσέγγιση επιτρέπει τόσο αυτοματοποιημένα workflows (με χρήση scripts ή integration σε CI/CD pipelines), όσο και διαδραστική διαχείριση από τη γραμμή εντολών. Υποστηρίζει επίσης plugins, τα οποία

[87] https://kubernetes.io/docs/reference/kubectl/

επεκτείνουν τη λειτουργικότητά του, επιτρέποντας τη δημιουργία custom subcommands για εξειδικευμένες ανάγκες.

Η ενσωμάτωση του kubectl σε Kubernetes περιβάλλοντα είναι εγγενής, καθώς αποτελεί μέρος κάθε επίσημης εγκατάστασης του Kubernetes. Διατίθεται ως standalone binary που συνδέεται μέσω kubeconfig με τον API server του cluster, και μπορεί να εκτελείται από οποιονδήποτε client host ή από pods μέσα στο ίδιο το cluster. Συνήθως, αποτελεί τη «βάση» πάνω στην οποία χτίζονται πιο σύνθετα εργαλεία cluster management, καθώς παρέχει άμεση, χαμηλού επιπέδου πρόσβαση σε όλα τα resources.

Το kubectl ταξινομείται ως External εργαλείο, καθώς εκτελείται εκτός του Kubernetes cluster (client-side) και επικοινωνεί με τον API server μέσω του αρχείου kubeconfig.

Παρά τα πλεονεκτήματα της αξιοπιστίας, της πληρότητας και της ισχυρής επεκτασιμότητας, το kubectl έχει και περιορισμούς: δεν προσφέρει γραφικό περιβάλλον (GUI), η εκμάθησή του απαιτεί γνώση CLI workflows και YAML manifests Η αξιοπιστία και η επεκτασιμότητά του το καθιστούν αναντικατάστατο εργαλείο διαχείρισης. Δεν παρέχει GUI, αλλά θεωρείται βασικό εργαλείο για προηγμένους (advanced) χρήστες.

Πλεονεκτήματα

- *Επίσημο εργαλείο του Kubernetes*: πλήρως υποστηριζόμενο από το ίδιο το project και ενσωματωμένο σε όλες τις διανομές.
- *Πλήρης κάλυψη λειτουργιών*: υποστηρίζει όλες τις εντολές για δημιουργία, ενημέρωση, διαγραφή και παρακολούθηση Kubernetes αντικειμένων.
- *Επεκτασιμότητα με plugins*: επιτρέπει την ανάπτυξη custom subcommands που προσαρμόζονται σε εξειδικευμένες ανάγκες ομάδων.
- *Ισχυρή ενοποίηση με αυτοματισμούς*: μπορεί να ενσωματωθεί εύκολα σε scripts και CI/CD pipelines.
- *Εργαλείο troubleshooting*: εντολές όπως kubectl logs και kubectl exec παρέχουν άμεση πρόσβαση για διάγνωση προβλημάτων.
- *Διαφάνεια και έλεγχος*: προσφέρει χαμηλού επιπέδου αλληλεπίδραση με τον API server, επιτρέποντας πλήρη ορατότητα και χειρισμό.

Μειονεκτήματα

- *Υψηλή καμπύλη μάθησης*: απαιτεί καλή γνώση YAML manifests, Kubernetes resources και CLI workflows.
- *Έλλειψη γραφικής διεπαφής (GUI)*: η χρήση του μπορεί να είναι αποτρεπτική για λιγότερο έμπειρους χρήστες.
- *Επιρρεπές σε σφάλματα*: λάθη στη σύνταξη YAML ή στις εντολές μπορούν να προκαλέσουν δυσλειτουργίες στο cluster.

- *Περιορισμένη εμπειρία συνεργασίας*: δεν παρέχει ενσωματωμένα εργαλεία για συνεργατική εργασία (σε αντίθεση με GUIs που προσφέρουν role-based views).
- *Μη ιδανικό για μεγάλα multi-cluster περιβάλλοντα*: γίνεται δύσχρηστο χωρίς πρόσθετα εργαλεία που οργανώνουν και αυτοματοποιούν τη διαχείριση

K9s

Το **K9s**[88] είναι ένα terminal-based dashboard για real-time παρακολούθηση και διαχείριση των πόρων ενός cluster, με γρήγορη απόκριση και χαμηλό resource usage (GitHub/k9s, 2024). Σε αντίθεση με το kubectl, το οποίο βασίζεται σε μεμονωμένες εντολές, το K9s προσφέρει ένα διαδραστικό περιβάλλον μέσα από το terminal, το οποίο διευκολύνει την πλοήγηση ανάμεσα σε pods, deployments, services ή namespaces και επιτρέπει την εκτέλεση ενεργειών όπως scaling, διαγραφή πόρων ή προβολή logs. Η χρήση φίλτρων και δυναμικής αναζήτησης καθιστά τη διαχείριση μεγάλων clusters πιο αποτελεσματική, ενώ η δυνατότητα ταυτόχρονης παρακολούθησης logs από πολλά pods απλοποιεί τη διαδικασία debugging. Το εργαλείο υποστηρίζει πολλαπλά clusters μέσω του αρχείου kubeconfig, ενώ παρέχει δυνατότητα προσαρμογής με custom views και plugins, γεγονός που το καθιστά ευέλικτο για διαφορετικά σενάρια χρήσης.

Η ενσωμάτωσή του στο Kubernetes γίνεται αποκλειστικά client-side, αφού το K9s δεν εγκαθίσταται μέσα στο cluster· εκτελείται ως standalone binary ή containerized εφαρμογή που επικοινωνεί με τον API server. Για τον λόγο αυτό ο τρόπος εκτέλεσης είναι μόνο εξωτερικός (external). Χάρη στην ελαφριά του φύση και στη γρήγορη απόκριση, προτιμάται από χρήστες που βασίζονται σε CLI εργαλεία αντί για γραφικές διεπαφές, αν και η καμπύλη εκμάθησης μπορεί να είναι απότομη για όσους δεν είναι εξοικειωμένοι με το περιβάλλον γραμμής εντολών.

Πλεονεκτήματα

- *Διαδραστικό περιβάλλον τερματικού (TUI)*: προσφέρει real-time πλοήγηση και διαχείριση Kubernetes πόρων χωρίς ανάγκη συνεχούς εκτέλεσης εντολών.
- *Βελτιωμένο debugging*: επιτρέπει την ταυτόχρονη παρακολούθηση logs από πολλαπλά pods και άμεση πρόσβαση σε λεπτομέρειες για events ή failures.
- *Υποστήριξη πολλαπλών clusters*: μέσω του kubeconfig, οι χρήστες μπορούν να εναλλάσσονται εύκολα ανάμεσα σε διαφορετικά clusters.
- *Παραμετροποίηση με plugins και custom views*: δίνει τη δυνατότητα προσαρμογής στις ανάγκες κάθε ομάδας.
- *Χαμηλή κατανάλωση πόρων*: τρέχει ως lightweight binary ή container χωρίς να επιβαρύνει το cluster.

[88] https://k9scli.io/

- *Ευκολία για power users*: αυξάνει την παραγωγικότητα σε χρήστες που προτιμούν το CLI, μειώνοντας τον χρόνο πληκτρολόγησης πολύπλοκων εντολών kubectl.

Μειονεκτήματα

- *Έλλειψη GUI*: δεν παρέχει γραφικό περιβάλλον, άρα δεν είναι φιλικό σε μη εξοικειωμένους χρήστες.
- *Απότομη καμπύλη εκμάθησης*: απαιτεί εξοικείωση με το CLI και τις έννοιες του Kubernetes.
- *Περιορισμένη συνεργατικότητα*: δεν προσφέρει role-based views ή μηχανισμούς διαμοιρασμού, σε αντίθεση με web-based dashboards.
- *Εξάρτηση από το kubeconfig*: αν το αρχείο δεν έχει ρυθμιστεί σωστά, η εμπειρία χρήσης γίνεται περίπλοκη.
- *Περιορισμένο οικοσύστημα*: σε σύγκριση με εργαλεία, όπως Rancher ή Portainer, δεν προσφέρει εκτεταμένα integrations ή multi-user περιβάλλοντα.

Portainer

Το Portainer[89] είναι μια γραφική διεπαφή χρήστη (GUI) που απλοποιεί τη διαχείριση περιβαλλόντων Docker και Kubernetes, παρέχοντας έναν οπτικό τρόπο αλληλεπίδρασης με clusters χωρίς την ανάγκη χρήσης CLI εργαλείων όπως το kubectl (Portainer.io, 2024). Η βασική του λειτουργικότητα επικεντρώνεται στη δημιουργία, παρακολούθηση και διαχείριση Kubernetes resources (pods, deployments, services, configmaps, secrets), ενώ παρέχει ενσωματωμένη υποστήριξη για Helm charts, επιτρέποντας την εύκολη εγκατάσταση εφαρμογών μέσω γραφικού περιβάλλοντος. Επιπλέον, το Portainer διαθέτει μηχανισμούς RBAC (Role-Based Access Control), ώστε να μπορούν διαφορετικοί χρήστες να έχουν διαφοροποιημένα δικαιώματα διαχείρισης, κάτι που το καθιστά κατάλληλο για μικρές ομάδες ή εκπαιδευτικά περιβάλλοντα.

Όσον αφορά την ενσωμάτωσή του σε Kubernetes, το Portainer εκτελείται συνήθως ως Deployment μέσα στο cluster και εκτίθεται μέσω ενός Service (συχνά τύπου LoadBalancer ή NodePort), παρέχοντας web-based dashboard για όλους τους χρήστες. Το Portainer διαθέτει κατανεμημένη αρχιτεκτονική, καθώς το web UI και το management API μπορούν να λειτουργούν εκτός cluster, ενώ τα Portainer agents εκτελούνται εντός του Kubernetes cluster για τη συλλογή δεδομένων και την εφαρμογή εντολών.

Το Portainer κατατάσσεται στην κατηγορία Hybrid (distributed), καθώς συνδυάζει in-cluster components (agents, deployments) με external components (web UI, API). Η εγκατάσταση μπορεί να γίνει εύκολα με Helm chart, με έτοιμα YAML manifests ή ακόμα και μέσω Docker container, εάν χρησιμοποιείται σε υβριδικά περιβάλλοντα. Εφόσον το Portainer διαχειρίζεται απευθείας τον Kubernetes API Server, δεν απαιτεί εξωτερικούς agents πέρα από το ίδιο το Portainer server.

[89] https://docs.portainer.io/

Ενδείκνυται για μικρές ομάδες και αρχάριους, αν και στερείται προηγμένων δυνατοτήτων αποσφαλμάτωσης (PORTAINER.IO, 2024).

Πλεονεκτήματα

- *Φιλικό web-based GUI*: παρέχει οπτικοποίηση και εύκολη διαχείριση Kubernetes clusters χωρίς ανάγκη CLI γνώσεων.
- *RBAC υποστήριξη*: δίνει δυνατότητα multi-user πρόσβασης με διακριτά δικαιώματα, ενισχύοντας την ασφάλεια.
- *Dashboard παρακολούθησης*: εμφανίζει workloads και πόρους του cluster σε πραγματικό χρόνο.
- *Multi-environment διαχείριση*: υποστηρίζει Docker hosts, Docker Swarm και Kubernetes από μία διεπαφή.
- *Ευκολία εγκατάστασης*: μπορεί να εγκατασταθεί με Helm chart, YAML manifests ή ως Docker container.

Μειονεκτήματα

- *Όχι κατάλληλο για μεγάλα enterprise clusters*: σχεδιάστηκε για απλότητα και όχι για πολύπλοκα multi-cluster setups.
- *Έλλειψη προηγμένων integrations*: δεν προσφέρει βαθιά ενοποίηση με monitoring ή security εργαλεία.
- *Εξάρτηση από GUI*: δεν είναι ιδανικό σε περιβάλλοντα όπου προτιμώνται CLI και automation pipelines.
- *Περιορισμένη υποστήριξη κοινότητας σε σύγκριση με open-source εναλλακτικές*: η ανάπτυξή του είναι πιο συγκεντρωμένη και λιγότερο εκτεταμένη

K8sGPT

Το K8sGPT[90] είναι ένα εργαλείο που αξιοποιεί μεγάλα γλωσσικά μοντέλα (LLMs) για την αυτόματη διάγνωση και ερμηνεία προβλημάτων σε Kubernetes clusters. Επεξεργάζεται δεδομένα, όπως καταχωρήσεις (logs), μετρικές και σήματα του cluster (events, health checks), και μέσω μοντέλων GPT επιχειρεί να εντοπίσει τις βασικές αιτίες (root causes) δυσλειτουργιών, παρουσιάζοντάς τες σε μορφή φυσικής γλώσσας κατανοητή ακόμη και σε μη ειδικούς χρήστες (Puutio, 2025). Έτσι, μειώνει σημαντικά τον χρόνο troubleshooting, καθιστώντας τη διάγνωση σφαλμάτων πιο προσιτή και διαφανή.

Λειτουργικά, το εργαλείο παρέχει CLI διεπαφή, μέσω της οποίας οι χρήστες μπορούν να εκτελούν ερωτήματα για την κατάσταση του cluster ή συγκεκριμένων πόρων (pods, deployments, services).

[90] https://k8sgpt.ai/docs

Το K8sGPT αναλύει τα δεδομένα και επιστρέφει περιγραφικές απαντήσεις, συχνά συνοδευόμενες από προτεινόμενες ενέργειες διόρθωσης. Μπορεί επίσης να επεκταθεί με plugins που βελτιώνουν την ακρίβεια ή εστιάζουν σε συγκεκριμένους τύπους workloads, ενώ υποστηρίζει πολλαπλές πηγές δεδομένων για πιο ολοκληρωμένη ανάλυση.

Η ενσωμάτωσή του στο Kubernetes γίνεται με δύο κύριους τρόπους: είτε ως CLI εργαλείο που κτελείται τοπικά από τον χρήστη και συνδέεται με τον Kubernetes API Server μέσω του αρχείου kubeconfig, επιτρέποντας την ανάλυση της κατάστασης του cluster χωρίς να απαιτείται εγκατάστασή του εντός της συστάδας, είτε ως sidecar/agent-based εγκατάσταση που τρέχει εντός του cluster και παρακολουθεί συνεχώς τα resources. Στη δεύτερη περίπτωση, το K8sGPT λειτουργεί πιο αυτόνομα και μπορεί να ειδοποιεί άμεσα σε περίπτωση προβλημάτων, αλλά απαιτεί περισσότερους πόρους.

Το K8sGPT κατατάσσεται στην κατηγορία Hybrid (deployable), διότι μπορεί να λειτουργεί εξωτερικά ως CLI (external mode), να εγκαθίσταται εντός του cluster ως agent (internal mode), ενώ παράλληλα συχνά αξιοποιεί εξωτερικά LLM APIs για την ανάλυση των δεδομένων.

Παρά τα πλεονεκτήματά του, το K8sGPT έχει ορισμένους περιορισμούς. Εξαρτάται σε μεγάλο βαθμό από εξωτερικά LLM APIs (π.χ. OpenAI, Azure OpenAI), κάτι που δημιουργεί ζητήματα κόστους, latency και ιδιωτικότητας δεδομένων. Επίσης, δεν είναι κατάλληλο για offline ή air-gapped περιβάλλοντα, όπου η πρόσβαση σε εξωτερικές υπηρεσίες είναι περιορισμένη. Τέλος, καθώς βασίζεται σε AI, μπορεί να παράγει λανθασμένες ή ασαφείς ερμηνείες (hallucinations), γεγονός που καθιστά απαραίτητο τον ανθρώπινο έλεγχο πριν από την εφαρμογή των προτεινόμενων λύσεων.

Πλεονεκτήματα

- *AI-driven troubleshooting*: αξιοποιεί LLMs για να εντοπίζει root causes σε Kubernetes clusters με φυσική γλωσσική επεξήγηση, διευκολύνοντας ακόμη και μη έμπειρους χρήστες.
- *CLI διεπαφή*: παρέχει άμεση πρόσβαση σε αναλύσεις και προτάσεις διορθώσεων χωρίς ανάγκη GUI.
- *Ενσωμάτωση με events, logs και metrics*: συλλέγει δεδομένα από διαφορετικές πηγές για πιο ακριβή διάγνωση.
- *Plugins και επεκτασιμότητα*: υποστηρίζει custom επεκτάσεις για συγκεκριμένα workloads ή περιπτώσεις χρήσης.
- *Επιτάχυνση troubleshooting*: μειώνει σημαντικά τον χρόνο διάγνωσης σε περιβάλλοντα παραγωγής.
- *Εκπαίδευση και καθοδήγηση*: δίνει προτάσεις σε φυσική γλώσσα, κάτι που το καθιστά χρήσιμο και σε εκπαιδευτικά σενάρια.

Μειονεκτήματα

- *Εξάρτηση από εξωτερικά LLM APIs*: απαιτεί πρόσβαση στο διαδίκτυο και σε παρόχους όπως OpenAI ή Azure OpenAI.
- *Μη κατάλληλο για air-gapped περιβάλλοντα*: δεν λειτουργεί σε clusters χωρίς σύνδεση στο διαδίκτυο, εκτός αν υπάρχει on-premise LLM (Large Language Model – Μεγάλο Γλωσσικό Μοντέλο παραγόμενο από τεχνικές Παραγωγικής Τεχνητής Νοημοσύνης).
- *Πιθανά σφάλματα ανάλυσης*: ως AI-based εργαλείο, μπορεί να δώσει εσφαλμένες ή ασαφείς εξηγήσεις που χρειάζονται ανθρώπινη επιβεβαίωση.
- *Κατανάλωση πόρων σε agent mode*: όταν τρέχει ως sidecar/agent στο cluster, μπορεί να επιβαρύνει τους πόρους.
- *Ζητήματα κόστους και ιδιωτικότητας*: η χρήση εξωτερικών LLM APIs μπορεί να αυξήσει το λειτουργικό κόστος και να δημιουργήσει ανησυχίες για την προστασία των δεδομένων.
- *Περιορισμένη υιοθέτηση*: ως σχετικά νέο εργαλείο, έχει μικρότερη κοινότητα και λιγότερη ωριμότητα σε σχέση με παραδοσιακά cluster management εργαλεία

SUSE Rancher

Το SUSE Rancher[91] είναι μια ολοκληρωμένη πλατφόρμα διαχείρισης Kubernetes που στοχεύει στην απλοποίηση της multi-cluster διαχείρισης και στην παροχή ενιαίου περιβάλλοντος ελέγχου για ομάδες διαφορετικών επιπέδων τεχνικής εμπειρίας. Μέσα από ένα web-based GUI, επιτρέπει την πλήρη οπτικοποίηση και διαχείριση clusters, είτε αυτά είναι on-premise, είτε σε public cloud, είτε σε υβριδικά περιβάλλοντα. Οι βασικές λειτουργίες του περιλαμβάνουν τη δημιουργία και διαχείριση RBAC πολιτικών για ασφαλή πρόσβαση, την επιβολή πολιτικών συμμόρφωσης μέσω ενσωμάτωσης με OPA Gatekeeper, καθώς και τη βελτίωση της παρατηρησιμότητας (observability) μέσω native dashboards που αξιοποιούν Prometheus και Grafana. Επιπλέον, το Rancher ενσωματώνεται με εργαλεία CI/CD επιτρέποντας την αυτοματοποίηση των pipelines ανάπτυξης, ενώ διαθέτει μηχανισμούς για policy enforcement και ασφαλή σύνδεση μεταξύ πολλαπλών clusters (Syrigos, Makris & Korakis, 2023).

Η ενσωμάτωσή του στο Kubernetes γίνεται κυρίως μέσω της εγκατάστασης του Rancher server, που τρέχει ως Deployment σε ένα cluster και παρέχει web-based πρόσβαση στους διαχειριστές. Υποστηρίζεται η εγκατάσταση με Helm chart, μέσω έτοιμων YAML manifests ή με χρήση container images. Ένα ακόμη σημαντικό χαρακτηριστικό είναι ότι το Rancher μπορεί να λειτουργήσει ως Cluster Provisioner, δηλαδή να δημιουργήσει νέα clusters (π.χ. RKE, K3s) και να τα εντάξει αυτόματα στο περιβάλλον διαχείρισής του, μειώνοντας την πολυπλοκότητα σε multi-cluster setups.

Το SUSE Rancher κατατάσσεται στην κατηγορία Hybrid (distributed), καθώς μπορεί να εγκατασταθεί και να λειτουργήσει είτε εσωτερικά (εντός cluster) είτε εξωτερικά (ως control plane εκτός cluster), ενώ ταυτόχρονα υποστηρίζει τη διαχείριση πολλαπλών, απομακρυσμένων clusters.

[91] https://rancher.com/docs/

Αυτή η ευελιξία το καθιστά ιδανικό για οργανισμούς που χρησιμοποιούν συνδυασμό on-premise και cloud περιβαλλόντων, ενισχύοντας τη διαλειτουργικότητα και τη συνοχή της διαχείρισης.

Παρότι το Rancher παρέχει εκτεταμένες δυνατότητες, έχει και ορισμένους περιορισμούς. Η εγκατάστασή του μπορεί να είναι περίπλοκη, ιδιαίτερα σε παραγωγικά περιβάλλοντα με υψηλές απαιτήσεις κλιμάκωσης. Επιπλέον, η εξάρτησή του από ένα κεντρικό Rancher server εισάγει έναν βαθμό πολυπλοκότητας και πιθανό σημείο αποτυχίας, κάτι που απαιτεί σωστή αρχιτεκτονική για υψηλή διαθεσιμότητα.

Πλεονεκτήματα

- *Multi-cluster διαχείριση*: παρέχει ενοποιημένο web-based GUI για on-premise, cloud και υβριδικά clusters.
- *RBAC και ασφάλεια*: υποστηρίζει ρόλους, πολιτικές πρόσβασης και ενοποίηση με εξωτερικούς μηχανισμούς authentication (π.χ. LDAP, Active Directory).
- *Policy enforcement*: ενσωματώνεται με OPA Gatekeeper για την επιβολή πολιτικών συμμόρφωσης.
- *Observability out-of-the-box*: διαθέτει ενσωματωμένη παρακολούθηση μέσω Prometheus και Grafana.
- *CI/CD integration*: υποστηρίζει pipelines για αυτοματοποίηση ανάπτυξης εφαρμογών.
- *Cluster provisioning*: μπορεί να δημιουργεί και να διαχειρίζεται clusters (π.χ. RKE, K3s) από το ίδιο περιβάλλον.
- *Υποστήριξη hybrid & edge περιβαλλόντων*: λειτουργεί τόσο σε παραδοσιακά data centers όσο και σε edge nodes.
- *Ενεργή κοινότητα και εμπορική υποστήριξη από SUSE:* προσφέρει τεχνική βοήθεια και updates.

Μειονεκτήματα

- *Πολυπλοκότητα εγκατάστασης και συντήρησης*: απαιτεί προσεκτική αρχιτεκτονική, ιδιαίτερα για high availability setups.
- *Αυξημένες απαιτήσεις πόρων*: το Rancher server καταναλώνει περισσότερους πόρους σε σχέση με ελαφρύτερα εργαλεία όπως Portainer.
- *Πιθανό σημείο αποτυχίας (single point of failure)*: αν δεν ρυθμιστεί σε HA mode, το κεντρικό Rancher instance μπορεί να αποτελέσει ρίσκο.
- *Όχι ιδανικό για μικρά clusters*: η πλήρης σουίτα δυνατοτήτων είναι συχνά υπερβολική για μικρές ομάδες ή απλά use cases.
- *Καμπύλη εκμάθησης*: λόγω της πληθώρας λειτουργιών, απαιτεί χρόνο εξοικείωσης από τους διαχειριστές.

Ο παραπάτω πίνακας συνοψίζει την ανάλυση των 5 αυτών εργαλείων της τρέχουσας κατηγορίας.

Πίνακας 16: Συγκριτική Αξιολόγηση Εργαλείων Cluster Management

Εργαλείο	**Κατηγορίες Cluster Management**	**Πλεονεκτήματα**	**Μειονεκτήματα**
kubectl	Cluster Provisioning, Configuration Management	Επίσημο εργαλείο Kubernetes · Πλήρης κάλυψη λειτουργιών (create, update, delete) · Υποστήριξη scripting & automation · Επεκτασιμότητα μέσω plugins · Ενσωμάτωση σε CI/CD pipelines · Ισχυρό troubleshooting (logs, exec) · Πλήρης έλεγχος και διαφάνεια πόρων	Υψηλή καμπύλη μάθησης · Απαίτηση γνώσης YAML & CLI · Έλλειψη GUI · Επιρρεπές σε λάθη εντολών · Περιορισμένη συνεργατικότητα · Όχι ιδανικό για μεγάλα multi-cluster setups
K9s	Cluster Monitoring & Troubleshooting	Γρήγορο TUI, real-time monitoring · Ιδανικό για troubleshooting μέσω terminal · Υποστήριξη πολλαπλών clusters · Plugins & custom views · Χαμηλή κατανάλωση πόρων · Παραγωγικότητα για προχωρημένους χρήστες	Δεν διαθέτει GUI · Απαιτεί CLI εξοικείωση · Απότομη καμπύλη εκμάθησης · Περιορισμένη συνεργατικότητα (χωρίς RBAC views) · Εξάρτηση από kubeconfig · Περιορισμένο οικοσύστημα
Portainer	Multi-cluster Management, Configuration Management	Φιλικό web GUI · Multi-environment διαχείριση (Docker, Swarm, K8s) · RBAC υποστήριξη · Helm & YAML υποστήριξη · Dashboard monitoring · Ευκολία εγκατάστασης (Helm, Docker, YAML)	Όχι ιδανικό για μεγάλα clusters · Περιορισμένα integrations (security, monitoring) · Εξάρτηση από GUI · Περιορισμένη κοινότητα · Λιγότερο επεκτάσιμο από enterprise εργαλεία
K8sGPT	Cluster Monitoring & Troubleshooting (with AI-driven Diagnostics)	AI-driven troubleshooting · Ανάλυση logs, metrics & events · Προτάσεις σε φυσική γλώσσα · Plugins & επεκτασιμότητα · CLI διεπαφή, εύκολη χρήση · Μείωση χρόνου troubleshooting · Εκπαίδευση/καθοδήγηση μη ειδικών	Εξάρτηση από LLM APIs (OpenAI, Azure) · Μη κατάλληλο για air-gapped περιβάλλοντα · Πιθανά AI σφάλματα (hallucinations) · Αυξημένη κατανάλωση πόρων σε agent mode · Ζητήματα κόστους & ιδιωτικότητας · Περιορισμένη υιοθέτηση

SUSE Rancher	Cluster Provisioning, Multi-cluster / Hybrid Management, Configuration Management, Cluster Monitoring & Troublshooting	Multi-cluster διαχείριση (on-prem, cloud, hybrid) · RBAC & security integration · Policy enforcement με OPA Gatekeeper · Observability με Prometheus & Grafana · CI/CD integration · Cluster provisioning (RKE, K3s) · Υποστήριξη hybrid/edge περιβαλλόντων · Ενεργή κοινότητα & υποστήριξη SUSE	Πολύπλοκη εγκατάσταση & συντήρηση · Αυξημένες απαιτήσεις πόρων · Πιθανό single point of failure (server) · Υπερβολικό για μικρές ομάδες · Καμπύλη εκμάθησης λόγω πολλών λειτουργιών

4.2.7 Autoscaling

Η αυτόματη κλιμάκωση (autoscaling) αποτελεί κρίσιμο μηχανισμό για τη διαχείριση των πόρων σε περιβάλλοντα Kubernetes, προσφέροντας τη δυνατότητα για δυναμική κατανομή υποδομών ανάλογα με τις πραγματικές ανάγκες του συστήματος (RQ1). Η λειτουργία αυτή είναι απαραίτητη για την ευστάθεια, την απόδοση και την αποδοτική χρήση των πόρων, ειδικά σε εφαρμογές με έντονες μεταβολές φόρτου. Σε αυτή την ενότητα, εξετάζονται βασικά εργαλεία και προσεγγίσεις που υποστηρίζουν autoscaling σε Kubernetes. Στο πλαίσιο της RQ1 – *«Ποια είναι τα βασικά είδη εργαλείων που έχουν αναπτυχθεί στο οικοσύστημα του Kubernetes;»* – τα εργαλεία autoscaling μπορούν να ταξινομηθούν στις εξης βασικές τεχνολογικές κατηγορίες:

- **Hybrid Models (υβριδικά μοντένα)**: Εργαλεία, όπως το KOSMOS[92], υλοποιούν συνδυασμένη οριζόντια και κάθετη κλιμάκωση, αξιοποιώντας τόσο το HPA όσο και το VPA για μεγαλύτερη προσαρμοστικότητα (KOSMOS, 2023).
- **Event-Driven Autoscalers (αυτό-κλιμακωτές οδηγούμενοι από γεγονότα)**: Όπως το KEDA, τα οποία ενεργοποιούν την κλιμάκωση βάσει εξωτερικών συμβάντων (π.χ. Kafka messages, queue depth), επεκτείνοντας το μοντέλο serverless και επιτρέποντας scale-to-zero (CNCF, 2024). Όπως έχει προαναφερθεί στην Ενότητα 2.1.3, το μοντέλο serverless αναφέρεται σε μια αρχιτεκτονική όπου οι εφαρμογές εκτελούνται χωρίς ο χρήστης να χρειάζεται να διαχειρίζεται ρητά servers ή clusters. Οι πόροι εκτελούνται «on demand» και χρεώνονται ανάλογα με τη χρήση, ενώ σε περιόδους αδράνειας μπορεί να γίνει πλήρης απελευθέρωση πόρων (scale-to-zero).

Παράλληλα, στο πλαίσιο της RQ2, αναλύονται οι βασικές λειτουργικές δυνατότητες των εργαλείων

Horizontal Pod Autoscaler (HPA)

Το HPA αποτελεί τον βασικό μηχανισμό οριζόντιας κλιμάκωσης στο Kubernetes, επιτρέποντας την αυτόματη αύξηση ή μείωση των pods με βάση μετρικές, όπως η χρήση CPU, ή custom μετρικές

[92] https://github.com/deib-polimi/kosmos

που παράγονται από εξωτερικά monitoring συστήματα (π.χ. Prometheus) και εκτίθενται στο Kubernetes μέσω του Custom Metrics API.. Η έρευνα των Augustyn, Wyciślik, & Sojka (2024) προτείνει μια μεθοδολογία βελτιστοποίησης των παραμέτρων του HPA, αξιοποιώντας την αρχή της μέγιστης εντροπίας και τεχνικές kernel smoothing για να καθορίσει τη βέλτιστη τιμή μέγιστου πλήθους pods ανά υπηρεσία. Αυτό οδηγεί σε έως και 15% μείωση των ενεργοποιημένων κόμβων, μειώνοντας το συνολικό κόστος χωρίς απώλειες απόδοσης.

Η ενσωμάτωση του HPA στο Kubernetes είναι εγγενής (internal), καθώς αποτελεί ενσωματωμένο μηχανισμό του control plane, εγκατεστημένο ως controller που αλληλεπιδρά άμεσα με το Kubernetes API server.

Πλεονεκτήματα:

- *Ενσωμάτωση με το Kubernetes API*: Δεν απαιτεί εξωτερικές επεκτάσεις· λειτουργεί out-of-the-box με τον control plane.
- *Υποστήριξη βασικών μετρικών*: CPU και μνήμη υποστηρίζονται εγγενώς, ενώ με το Custom Metrics API μπορεί να αντλήσει δεδομένα από Prometheus ή άλλα monitoring συστήματα.
- *Δυνατότητα real-time scaling*: Προσαρμόζει άμεσα τον αριθμό των pods ανάλογα με τις τρέχουσες μετρικές.
- *Συμβατότητα με CI/CD*: Μπορεί να συνδυαστεί με pipelines για αυτόματη ανάπτυξη και δυναμική διαχείριση φορτίου.
- *Απλότητα*: Είναι το πιο κατανοητό και ευρέως χρησιμοποιούμενο autoscaling εργαλείο, με εκτενή τεκμηρίωση και παραδείγματα.

Μειονεκτήματα:

- *Συντηρητικός αλγόριθμος scaling*: Οι αποφάσεις λαμβάνονται με βάση μέσους όρους, γεγονός που μπορεί να οδηγήσει σε καθυστερημένη ανταπόκριση σε απότομες αυξήσεις φορτίου.
- *Υπερκατανάλωση πόρων*: Λόγω του συντηρητικού τρόπου κλιμάκωσης, μπορεί να δημιουργήσει περισσότερα pods από τα απολύτως απαραίτητα.
- *Έλλειψη προβλεπτικής ευφυΐας*: Δεν ενσωματώνει machine learning ή forecasting αλγορίθμους για να προλάβει μελλοντικές αιχμές.
- *Περιορισμοί σε σύνθετες μετρικές*: Αν και υποστηρίζονται custom metrics, η ρύθμιση τους είναι πιο περίπλοκη και απαιτεί πρόσθετα components (π.χ. Prometheus Adapter).
- *Περιορισμένη υποστήριξη scale-to-zero*: Σε αντίθεση με εργαλεία όπως το KEDA, το HPA δεν μπορεί να μειώσει τον αριθμό των pods στο μηδέν.

KOSMOS Autoscaler

Το KOSMOS Autoscaler είναι ένα ερευνητικό εργαλείο που στοχεύει να ξεπεράσει τους

περιορισμούς της αυτόνομης χρήσης του HPA (Horizontal Pod Autoscaler) και του VPA (Vertical Pod Autoscaler). Εισάγει ένα υβριδικό μοντέλο κλιμάκωσης, όπου η απόφαση για αύξηση/μείωση pods (οριζόντια κλιμάκωση) συνδυάζεται με την απόφαση για προσαρμογή των πόρων (CPU, μνήμη) που αποδίδονται σε κάθε pod (κάθετη κλιμάκωση). Αυτό του επιτρέπει να αντιδρά σε μεταβαλλόμενα φορτία πιο ευέλικτα, αποφεύγοντας τόσο την υπερκατανάλωση όσο και την ανεπαρκή απόδοση, ιδίως σε εφαρμογές που εκτελούνται σε cloud ή edge περιβάλλοντα (KOSMOS, 2023).

Λειτουργικά, το KOSMOS βασίζεται σε control-theoretical planners που εκτιμούν τις ανάγκες των εφαρμογών σε πραγματικό χρόνο και λαμβάνουν αποφάσεις κλιμάκωσης βάσει SLA/SLO στόχων. Μπορεί να διαχειριστεί workloads με υψηλή διακύμανση και να εξισορροπήσει την κατανάλωση πόρων με την απόδοση. Σε πειραματικές μελέτες έχει επιδείξει σημαντική μείωση των SLA violations σε σχέση με την αυτόνομη χρήση του HPA ή του VPA.

Η ενσωμάτωσή του στο Kubernetes γίνεται μέσω Custom Resource Definitions (CRDs) και Controllers, τα οποία επεκτείνουν το control plane του Kubernetes. Ο autoscaler παρακολουθεί μετρικές φόρτου μέσω monitoring εργαλείων (π.χ. Prometheus) και χρησιμοποιεί custom metrics APIs για να τις αξιοποιήσει στις αποφάσεις κλιμάκωσης. Σε αυτή τη διαμόρφωση, ο autoscaler επικοινωνεί απευθείας με τα αντικείμενα HPA/VPA ή τα αντικαθιστά με δικό του μηχανισμό λήψης αποφάσεων, εξασφαλίζοντας χαμηλό latency και άμεση πρόσβαση στους πόρους.

Το KOSMOS Autoscaler κατατάσσεται στην κατηγορία των internal εργαλείων, καθώς εγκαθίσταται ως Controller εντός του cluster και αλληλεπιδρά απευθείας με το Kubernetes API server, χωρίς να απαιτεί εξωτερικές υπηρεσίες ή απομακρυσμένα decision engines. Αν και πρόκειται κυρίως για proof-of-concept εργαλείο σε ερευνητικό στάδιο, η υλοποίησή του έχει ανέβει σε GitHub repo (deib-polimi/kosmos) και μπορεί να εγκατασταθεί σε Kubernetes clusters για πειραματικούς σκοπούς.

Πλεονεκτήματα:

- *Συνδυαστική κλιμάκωση (HPA + VPA)*: Ενσωματώνει ταυτόχρονα μηχανισμούς οριζόντιας και κάθετης κλιμάκωσης, ξεπερνώντας τους περιορισμούς της ξεχωριστής χρήσης τους.
- *Βελτίωση απόδοσης εφαρμογών*: Ανταποκρίνεται καλύτερα σε μεταβαλλόμενα workloads, μειώνοντας τα SLA violations σε απαιτητικά περιβάλλοντα.
- *Αποδοτική χρήση πόρων*: Εξισορροπεί την κατανάλωση CPU/μνήμης με την απόδοση, περιορίζοντας τόσο την υπερκατανάλωση όσο και την ανεπαρκή κατανομή πόρων.
- *Υποστήριξη για edge και cloud περιβάλλοντα*: Έχει σχεδιαστεί για σενάρια όπου η διακύμανση του φορτίου είναι έντονη (π.χ. IoT, real-time streaming).
- *Ερευνητική τεκμηρίωση*: Η αρχιτεκτονική του έχει αξιολογηθεί σε πειραματικές μελέτες με μετρήσιμα οφέλη σε latency και throughput.

Μειονεκτήματα:

- *Πολυπλοκότητα παραμετροποίησης*: Απαιτεί ρύθμιση πολλών CRDs και controllers, με γνώση των εσωτερικών μηχανισμών scaling.
- *Περιορισμένη ωριμότητα*: Βρίσκεται σε ερευνητικό στάδιο και δεν διαθέτει την υποστήριξη/σταθερότητα ενός CNCF project.
- *Περιορισμένη ενσωμάτωση με vanilla Kubernetes*: Δεν περιλαμβάνεται στους built-in autoscalers και χρειάζεται custom εγκατάσταση/επεκτάσεις.
- *Εξάρτηση από monitoring εργαλεία*: Για να λειτουργήσει πλήρως απαιτεί integration με Prometheus ή άλλα metrics pipelines, γεγονός που αυξάνει την πολυπλοκότητα.
- *Έλλειψη μεγάλης κοινότητας*: Λόγω του ερευνητικού χαρακτήρα, δεν υπάρχουν πολλά παραδείγματα παραγωγικής χρήσης ή υποστήριξη από enterprise vendors.

KEDA (Kubernetes-based Event Driven Autoscaler)

Το KEDA (Kubernetes Event-Driven Autoscaler) είναι ένα εργαλείο αυτόματης κλιμάκωσης που βασίζεται σε γεγονότα (event-driven scaling). Σε αντίθεση με το HPA, το οποίο αξιοποιεί κυρίως μετρικές συστήματος όπως CPU και μνήμη, το KEDA επιτρέπει την κλιμάκωση pods με βάση εξωτερικά γεγονότα ή application-level metrics, όπως ο αριθμός μηνυμάτων σε μια ουρά (π.χ. RabbitMQ, Kafka), το βάθος μιας βάσης δεδομένων, οι HTTP requests ανά δευτερόλεπτο, ή ακόμη και triggers από cloud providers (Azure Functions, AWS SQS). Λειτουργεί σε συνδυασμό με το HPA, εμπλουτίζοντας το με “external metrics adapters” και παρέχοντας περισσότερους από 60 scalers για διαφορετικές πηγές δεδομένων. Με αυτό τον τρόπο επιτρέπει και scale-to-zero, δηλαδή την πλήρη απελευθέρωση pods όταν δεν υπάρχει καμία δραστηριότητα.

Η ενσωμάτωσή του στο Kubernetes γίνεται ως Custom Controller που εγκαθίσταται μέσω Helm chart ή YAML manifests και διαχειρίζεται CRDs (Custom Resource Definitions), όπως ScaledObject και ScaledJob. Τα αντικείμενα αυτά επιτρέπουν στον χρήστη να δηλώσει με YAML τις πολιτικές κλιμάκωσης ανάλογα με τα γεγονότα που παρακολουθεί. Αν και το KEDA μπορεί να αξιοποιεί εξωτερικές πηγές γεγονότων (event sources), η λήψη και επεξεργασία των δεδομένων αυτών πραγματοποιείται από ενσωματωμένο controller στο cluster, γεγονός που το καθιστά internal εργαλείο από αρχιτεκτονική σκοπιά. Η λειτουργία του βασίζεται αποκλειστικά σε μηχανισμούς του Kubernetes (Controllers, CRDs, API server), χωρίς εξωτερική οντότητα που να διαχειρίζεται τις αποφάσεις κλιμάκωσης. Η ευελιξία αυτή, σε συνδυασμό με την υποστήριξη από το CNCF και τη συμβατότητά του με GitOps pipelines, καθιστά το KEDA μια εσωτερική αλλά επεκτάσιμη λύση αυτόματης κλιμάκωσης για cloud-native περιβάλλοντα. Παράλληλα, επειδή υποστηρίζεται από το CNCF (CNCF Landscape, 2024), έχει ενεργό οικοσύστημα και πλήρη συμβατότητα με GitOps workflows. Η χρήση του KEDA είναι ιδιαίτερα διαδεδομένη σε σενάρια serverless, καθώς μειώνει το κόστος επιτρέποντας την κλιμάκωση βάσει πραγματικής ζήτησης και

όχι απλά βάσει συστημικών κατωφλίων.

Πλεονεκτήματα:

- *Υποστήριξη event-based scaling*: Το KEDA επιτρέπει την κλιμάκωση pods με βάση εξωτερικά γεγονότα (π.χ. Kafka topics, RabbitMQ queues, HTTP requests), πέρα από τις κλασικές μετρικές CPU/μνήμης του HPA, δίνοντας μεγαλύτερη ευελιξία σε πραγματικές εφαρμογές.
- *Custom triggers*: Ο χρήστης μπορεί να ορίσει δικούς του scalers ή να επεκτείνει τους υπάρχοντες, προσαρμόζοντας τη λογική κλιμάκωσης σε ιδιαίτερες ανάγκες (π.χ. triggers από custom APIs).
- *Scale-to-zero*: Υποστηρίζει πλήρη απενεργοποίηση pods όταν δεν υπάρχει φορτίο, μειώνοντας σημαντικά το κόστος σε serverless περιβάλλοντα.
- *Συνεργασία με HPA*: Λειτουργεί συμπληρωματικά με το Horizontal Pod Autoscaler, αξιοποιώντας το ίδιο API αλλά εμπλουτίζοντάς το με external metrics.
- *Ενεργή κοινότητα & CNCF υποστήριξη*: Ως CNCF project έχει ενεργή ανάπτυξη, μεγάλη βιβλιοθήκη scalers (>60), και τεκμηρίωση που το καθιστούν έτοιμο για παραγωγή.

Μειονεκτήματα:

- *Προσθήκη πολυπλοκότητας*: Η εγκατάσταση απαιτεί CRDs (ScaledObject, ScaledJob), επιπλέον configuration και tuning, κάτι που αυξάνει το λειτουργικό overhead σε σχέση με το HPA.
- *Περιορισμοί σε κάθετη κλιμάκωση*: Υποστηρίζει μόνο οριζόντια κλιμάκωση pods (scale out/in), άρα για πλήρη κάλυψη σε περιβάλλοντα με δυναμικές απαιτήσεις CPU/memory χρειάζεται συνδυασμό με VPA ή υβριδικά μοντέλα.
- *Ανάγκη για monitoring integration*: Για βέλτιστη λειτουργία συνήθως συνδυάζεται με Prometheus ή άλλα monitoring συστήματα, κάτι που μπορεί να περιπλέξει την υποδομή.
- *Μειωμένη απόδοση σε σύνθετα workloads*: Σε σενάρια με πολλαπλά triggers ή events που εμφανίζονται ταυτόχρονα, η συμπεριφορά του KEDA μπορεί να απαιτεί προσεκτική ρύθμιση thresholds ώστε να αποφευχθεί υπερ-κλιμάκωση ή “flapping” (συνεχείς εναλλαγές scale up/scale down).

Ο παραπάτω πίνακας συνοψίζει την ανάλυση των 3 αυτών εργαλείων της τρέχουσας κατηγορίας.

Πίνακας 17: Συγκριτική Αξιολόγηση Εργαλείων Autoscaling

Εργαλείο / Μέθοδος	**Τύπος Scaling / Κατηγορίες Autoscaling**	**Πλεονεκτήματα**	**Μειονεκτήματα**

HPA (Tuned)	Horizontal Pod Autoscaling	Ενσωματωμένο στο Kubernetes API (χωρίς εξωτερικές επεκτάσεις). Υποστηρίζει CPU, μνήμη και custom metrics (μέσω Prometheus Adapter). Παρέχει real-time scaling. Απλό στη ρύθμιση και ευρέως διαδεδομένο. Συμβατό με CI/CD pipelines.	Συντηρητικός αλγόριθμος που καθυστερεί σε αιχμές. Δεν διαθέτει προβλεπτική ευφυΐα (forecasting). Δεν υποστηρίζει scale-to-zero. Περιορισμένο σε σύνθετες μετρικές. Μπορεί να προκαλέσει υπερκατανάλωση πόρων.
KOSMOS	Horizontal & Vertical Pod Autoscaling	Ενσωματώνει HPA και VPA για βέλτιστη χρήση πόρων. Μειώνει SLA violations και latency. Ανταποκρίνεται αποτελεσματικά σε μεταβαλλόμενο φορτίο. Κατάλληλο για cloud και edge περιβάλλοντα. Διαθέτει ερευνητική τεκμηρίωση με μετρήσιμα οφέλη.	Αυξημένη πολυπλοκότητα ρύθμισης και tuning. Βρίσκεται σε ερευνητικό στάδιο χωρίς επίσημη υποστήριξη. Απαιτεί custom CRDs και integration με monitoring συστήματα. Περιορισμένη ωριμότητα και κοινότητα χρήσης.
KEDA	Event-driven Scaling, Horizontal Pod Autoscaling	Υποστηρίζει περισσότερες από 60 πηγές γεγονότων. Συνεργάζεται με το HPA. Προσφέρει δυνατότητα scale-to-zero (ιδανικό για serverless περιβάλλοντα). Διαθέτει CNCF υποστήριξη και ενεργή κοινότητα. Παρέχει custom triggers και ευελιξία σε workload-based scaling.	Η εγκατάσταση απαιτεί CRDs και controller (ScaledObject, ScaledJob). Υποστηρίζει μόνο οριζόντια κλιμάκωση. Χρειάζεται Prometheus ή άλλα monitoring εργαλεία. Αυξημένη πολυπλοκότητα ρύθμισης σε πολλαπλά triggers. Πιθανότητα υπερ-κλιμάκωσης (“flapping”) αν δεν γίνει σωστό tuning.

4.2.8 Machine Learning & AI

Η ενσωμάτωση τεχνικών Μηχανικής Μάθησης (ML) και Τεχνητής Νοημοσύνης (AI) σε Kubernetes clusters έχει επιτρέψει νέες προσεγγίσεις για scheduling, scaling, αυτοματοποίηση και πρόβλεψη αποτυχιών. Η κατηγορία Machine Learning & AI διαφοροποιείται από τις προηγούμενες, καθώς δεν αντιπροσωπεύει μια μεμονωμένη λειτουργία του Kubernetes (όπως Monitoring, Security ή Autoscaling), αλλά μια οριζόντια τεχνολογική προσέγγιση που μπορεί να ενισχύσει πολλαπλές λειτουργίες του οικοσυστήματος. Εργαλεία και τεχνικές ML/AI χρησιμοποιούνται για την πρόβλεψη φόρτου και την προληπτική κλιμάκωση (π.χ. predictive autoscaling), για τον εντοπισμό ανωμαλιών σε μετρικές και logs (π.χ. anomaly detection στο monitoring), για την ανίχνευση επιθέσεων στο επίπεδο ασφαλείας (security), καθώς και για τη βελτιστοποίηση του scheduling μέσω ανάλυσης ιστορικών δεδομένων. Με τον τρόπο αυτό, η κατηγορία αυτή λειτουργεί συμπληρωματικά προς τις υπόλοιπες, επεκτείνοντας τις δυνατότητές τους μέσω προγνωστικής ανάλυσης, αυτοματοποίησης και ευφυών μηχανισμών λήψης αποφάσεων.

DQN Scheduler – ML-based Scheduling μέσω Reinforcement Learning

Το DQN Scheduler[93] αποτελεί μια προσέγγιση προγραμματισμού pods στο Kubernetes που βασίζεται στη Μηχανική Μάθηση, και συγκεκριμένα σε τεχνικές Deep Reinforcement Learning (Deep Q-Learning). Σε αντίθεση με τον default Kubernetes scheduler, που λαμβάνει αποφάσεις βάσει προκαθορισμένων heuristics (π.χ. least requested resources, balanced allocation), το DQN Scheduler εκπαιδεύεται δυναμικά ώστε να μαθαίνει πολιτικές βελτιστοποίησης της τοποθέτησης των pods. Ο στόχος του είναι να μειώσει την καθυστέρηση (latency), να αυξήσει τη χρήση των διαθέσιμων πόρων (CPU, μνήμη, storage, δικτύου) και να προσαρμόζεται σε διαφορετικούς τύπους φόρτων εργασίας (workloads) (Matranga, 2024)..

Λειτουργικά, το εργαλείο υλοποιεί έναν custom scheduler που αντικαθιστά ή τρέχει παράλληλα με τον προεπιλεγμένο Kubernetes scheduler. Μέσω reinforcement learning, λαμβάνει ως είσοδο την τρέχουσα κατάσταση του cluster (π.χ. αριθμός pods, κατανάλωση πόρων ανά node, επίπεδα συμφόρησης) και επιστρέφει ενέργειες προγραμματισμού (π.χ. σε ποιο node να τοποθετηθεί ένα νέο pod). Η εκπαίδευση του DQN γίνεται πάνω σε δεδομένα χρήσης του cluster, ενώ στη συνέχεια εφαρμόζεται online, λαμβάνοντας αποφάσεις σε πραγματικό χρόνο.

Η εσωτερική (internal) ενσωμάτωσή του στο Kubernetes μπορεί να γίνει μέσω custom scheduler plugin, χρησιμοποιώντας το Kubernetes scheduling framework, ή μέσω sidecar που επικοινωνεί με τον API server για να παρακάμπτει τον default scheduler. Έτσι, το εργαλείο αξιοποιεί το control plane του Kubernetes χωρίς να απαιτεί τροποποίηση στον πυρήνα του.

Πλεονεκτήματα:

[93] https://github.com/JolyonJian/DRS

- *Έξυπνος προγραμματισμός (intelligent scheduling)*: Χρησιμοποιεί τεχνικές deep reinforcement learning (Deep Q-Learning) για να μάθει βέλτιστες στρατηγικές τοποθέτησης pods.
- *Συνεχής μάθηση (continuous learning)*: Το μοντέλο προσαρμόζεται δυναμικά τους αλλαγές των workloads, σε αντίθεσητουςτους στατικούς schedulers που βασίζονται σε προκαθορισμένα heuristics.
- *Βελτιστοποίηση πόρων*: Μπορεί να μειώσει latency και να βελτιώσει την κατανομή CPU, μνήμης και bandwidth μεταξύ των nodes.
- *Υποστήριξη για ετερογενείς φόρτους (heterogeneous workloads)*: Προσαρμόζεται καλύτερα σε clusters που τρέχουν microservices, batch jobs και ML workloads ταυτόχρονα.
- *Ενσωμάτωση με monitoring εργαλεία*: Μπορεί να συνδεθεί με Prometheus ή άλλα monitoring συστήματα για συλλογή μετρικών που χρησιμοποιούνται στην εκπαίδευση του RL μοντέλου.

Μειονεκτήματα:

- *Πολυπλοκότητα ρύθμισης*: Η υλοποίηση ενός custom RL scheduler απαιτεί ειδικές γνώσεις σε Kubernetes και machine learning.
- *Αυξημένες υπολογιστικές απαιτήσεις*: Η εκπαίδευση και η online προσαρμογή του DQN scheduler καταναλώνουν σημαντικούς υπολογιστικούς πόρους.
- *Έλλειψη σταθερότητας*: Σε σχέση με τον default Kubernetes scheduler, τα ML-based schedulers μπορεί να είναι πιο δύσκολο να εγγυηθούν σταθερή συμπεριφορά σε όλα τα σενάρια.
- *Περιορισμένη υιοθέτηση*: Βρίσκεται κυρίως σε ερευνητικό στάδιο· δεν έχει ακόμη μεγάλη βιομηχανική διάδοση, οπότε η υποστήριξη και τα διαθέσιμα εργαλεία είναι περιορισμένα.
- *Δυσκολία debugging*: Η διαδικασία λήψης αποφάσεων από ένα RL μοντέλο είναι «μαύρο κουτί» (black box), γεγονός που δυσκολεύει τη διάγνωση σφαλμάτων.

KubePipe – Scalable ML Pipelines σε Kubernetes

Το KubePipeείναι ένα container-based εργαλείο που στοχεύει στην αυτοματοποίηση και επιτάχυνση του κύκλου ζωής εφαρμογών Μηχανικής Μάθησης (ML) σε Kubernetes (Daniel Suárez, 2025). Η λειτουργικότητά του εστιάζει στη δημιουργία, εκτέλεση και παρακολούθηση ML pipelines, τα οποία περιλαμβάνουν όλα τα στάδια: προ-επεξεργασία δεδομένων, εκπαίδευση μοντέλων, αξιολόγηση και ανάπτυξη σε παραγωγή. Σε αντίθεση με απλές batch jobs, το KubePipe υποστηρίζει Directed Acyclic Graphs (DAGs), που επιτρέπουν τον ορισμό εξαρτήσεων μεταξύ των βημάτων και την ταυτόχρονη εκτέλεση πολλαπλών εργασιών (parallel jobs).

Σημαντικό λειτουργικό χαρακτηριστικό είναι οι μηχανισμοί resiliency, δηλαδή η δυνατότητα επανεκκίνησης και συνέχισης των pipelines ακόμη και μετά από μερικές αποτυχίες. Επιπλέον, το KubePipe υποστηρίζει scalable execution, καθώς εκμεταλλεύεται την εγγενή δυναμική του

Kubernetes για την κλιμάκωση πόρων ανάλογα με το workload. Αξιοποιεί επίσης την ενοποίηση με frameworks όπως TensorFlow[94], PyTorch[95] ή Scikit-learn[96], προσφέροντας έτσι πλήρη υποστήριξη για end-to-end ML workflows.

Η ενσωμάτωση στο Kubernetes επιτυγχάνεται μέσω Custom Resource Definitions (CRDs), που επιτρέπουν στους χρήστες να ορίζουν pipelines δηλωτικά ως Kubernetes resources. Εναλλακτικά, η διαχείριση του κύκλου ζωής γίνεται μέσω Operators, οι οποίοι παρακολουθούν τα CRDs και εκτελούν αυτόματα τις κατάλληλες εργασίες. Η αλληλεπίδραση με το εργαλείο γίνεται είτε μέσω CLI, για πιο script-based χρήσεις, είτε μέσω web-based διεπαφής, που διευκολύνει την οπτική παρακολούθηση της προόδου.

Το KubePipe κατατάσσεται στην κατηγορία Hybrid (distributed) / Internal, με κύριους τρόπους ενσωμάτωσης τον Operator, το Service-level interface και την εγκατάσταση μέσω Helm, συνδυάζοντας εγγενή Kubernetes λειτουργικότητα με επεκτάσιμη υποστήριξη για data-intensive workloads. Ο χαρακτηρισμός hybrid–distributed προκύπτει από το γεγονός ότι το KubePipe μπορεί να εκτελεί pipelines σε πολλαπλά, ετερογενή Kubernetes περιβάλλοντα (on-prem και cloud) και να κατανέμει παράλληλα τα επιμέρους tasks σε διαφορετικούς κόμβους του cluster, επιτυγχάνοντας κατανεμημένη και κλιμακούμενη εκτέλεση.

Συνολικά, το KubePipe φέρνει τις πρακτικές των data pipelines και MLOps σε πλήρως containerized περιβάλλοντα, ενσωματώνοντας τη λογική του Kubernetes και επιτρέποντας φορητότητα, αυτοματοποίηση και κλιμακωσιμότητα. (Suárez et al., 2025)

Πλεονεκτήματα:

- *Υψηλή επεκτασιμότητα*: Υποστηρίζει την εκτέλεση πολλαπλών ML pipelines ταυτόχρονα σε κατανεμημένο περιβάλλον, αξιοποιώντας τη δυναμική κλιμάκωση του Kubernetes.
- *Αρθρωτή σχεδίαση*: Τα pipelines ορίζονται με DAGs, που επιτρέπουν σαφή ορισμό εξαρτήσεων και ανεξάρτητη εκτέλεση βημάτων.
- *Αυτοματοποίηση κύκλου ζωής ML*: Υποστηρίζει end-to-end ML workflows (data preprocessing, training, evaluation, deployment).
- *Ενσωμάτωση με ML frameworks*: Συμβατό με TensorFlow, PyTorch και Scikit-learn, επιτρέποντας φορητότητα μοντέλων σε διαφορετικά περιβάλλοντα.
- *Ανθεκτικότητα (resiliency)*: Περιλαμβάνει μηχανισμούς fault tolerance που επιτρέπουν την ολοκλήρωση pipelines ακόμη και αν ορισμένα tasks αποτύχουν.
- *Native Kubernetes integration*: Χρησιμοποιεί CRDs και Operators, κάτι που επιτρέπει δηλωτική διαχείριση ML pipelines ως Kubernetes resources.

[94] https://www.tensorflow.org/

[95] https://docs.pytorch.org/docs/stable/index.html

[96] https://scikit-learn.org/stable/

Μειονεκτήματα:

- *Απότομη καμπύλη μάθησης*: Απαιτεί εξοικείωση τόσο με Kubernetes concepts (CRDs, Operators) όσο και με ML workflows.
- *Δυσκολία ενσωμάτωσης σε legacy pipelines*: Τα παραδοσιακά ML workflows που δεν είναι containerized χρειάζονται σημαντική αναδόμηση.
- *Αυξημένες απαιτήσεις πόρων*: Η εκτέλεση κατανεμημένων ML pipelines σε cluster μπορεί να οδηγήσει σε υψηλή κατανάλωση CPU/GPU.
- *Περιορισμένη κοινότητα/υποστήριξη*: Δεν έχει την ωριμότητα ή το οικοσύστημα πιο γνωστών εργαλείων όπως Kubeflow.
- *Πολυπλοκότητα debugging*: Η ανάλυση αποτυχιών σε κατανεμημένα pipelines είναι δύσκολη χωρίς πρόσθετα εργαλεία παρακολούθησης.

Kubeflow

Το Kubeflow[97] αποτελεί την πιο διαδεδομένη open-source πλατφόρμα MLOps για το Kubernetes, σχεδιασμένη με σκοπό να απλοποιήσει και να αυτοματοποιήσει τη διαδικασία ανάπτυξης, εκπαίδευσης και εξυπηρέτησης μοντέλων μηχανικής μάθησης. Πρόκειται για ένα σύνολο από native Kubernetes components (όπως τα TFJob, Katib, KFServing και Pipelines), τα οποία αξιοποιούν CRDs (Custom Resource Definitions) και controllers ώστε να ενορχηστρώσουν πλήρεις ροές ML pipelines μέσα στο cluster. Χάρη σε αυτή τη στενή ενοποίηση με το Kubernetes control plane, το Kubeflow επιτρέπει την εύκολη δημιουργία και εκτέλεση πειραμάτων, τη ρύθμιση hyperparameters, καθώς και την αυτοματοποιημένη ανάπτυξη μοντέλων σε παραγωγή (serving).

Αναφορικά με την ενσωμάτωσή του, το Kubeflow εγκαθίσταται ως σύνολο από services και controllers στο cluster, μέσω Helm charts, kubectl manifests ή της Kubeflow Operator. Μπορεί να λειτουργήσει είτε σε ένα ενιαίο cluster όπου εκτελούνται τα ML workloads, είτε σε απομονωμένο "MLOps cluster" που διαχειρίζεται απομακρυσμένα workloads από άλλα Kubernetes περιβάλλοντα. Υποστηρίζει επίσης μηχανισμούς Role-Based Access Control (RBAC) για απομόνωση χρηστών και ομάδων, καθώς και multi-user dashboards που διευκολύνουν τη συνεργασία.

Ως εργαλείο, το Kubeflow κατατάσσεται στην κατηγορία Hybrid (distributed), καθώς τα κύρια components του (Operators, Services, Pipelines) εκτελούνται εντός του cluster, αλλά μπορεί επίσης να διαχειρίζεται και να ενορχηστρώνει workloads σε εξωτερικά ή απομακρυσμένα clusters, αξιοποιώντας Helm, Operators και Service-level διεπαφές. Η υβριδική του φύση επιτρέπει τόσο

[97] https://www.kubeflow.org/docs/

in-cluster λειτουργία όσο και integration με external ML/AI υποδομές, καθιστώντας το εργαλείο ευέλικτο και επεκτάσιμο για multi-cloud MLOps περιβάλλοντα.

Η ενσωμάτωσή του στο Kubernetes πραγματοποιείται με διάφορους τρόπους: μέσω Command-Line Interface (CLI) εργαλείων όπως το kfctl και το kubectl, μέσω Kubeflow Operator και service-level controllers που διαχειρίζονται αυτόματα τους πόρους ML, αλλά και με Helm charts ή YAML manifests για τη χειροκίνητη εγκατάσταση και παραμετροποίηση των επιμέρους components.

Παρά τα σημαντικά πλεονεκτήματά του – όπως η ολοκληρωμένη αυτοματοποίηση ML pipelines, η δυνατότητα επεκτασιμότητας και η υποστήριξη για πολλαπλά frameworks (TensorFlow, PyTorch, XGBoost) – η πλατφόρμα παρουσιάζει αυξημένη πολυπλοκότητα εγκατάστασης και συντήρησης. Η σωστή ρύθμιση των επιμέρους components απαιτεί εμπειρία τόσο σε Kubernetes όσο και σε MLOps αρχιτεκτονικές, ενώ η κατανάλωση πόρων είναι σημαντικά υψηλότερη σε σύγκριση με πιο ελαφριές λύσεις.

Πλεονεκτήματα:

- *Kubernetes-native αρχιτεκτονική***:** βασίζεται εξ ολοκλήρου σε CRDs και controllers, επιτρέποντας φυσική ενσωμάτωση στο Kubernetes control plane.
- *Υποστήριξη πολλαπλών ML frameworks:* διαθέτει εγγενή συμβατότητα με TensorFlow, PyTorch, Scikit-learn κ.ά. μέσω dedicated components (TFJob, PyTorchJob κ.λπ.).
- *Αυτοματοποιημένα ML pipelines:* μέσω του Kubeflow Pipelines επιτρέπεται η δημιουργία, εκτέλεση και επαναχρησιμοποίηση ML workflows με γραφικό UI.
- *Hyperparameter tuning και model serving:* υποστηρίζει auto-tuning μέσω Katib και serverless ανάπτυξη μοντέλων μέσω KServe (πρώην KFServing).
- *Ενσωμάτωση με CI/CD και GitOps:* μπορεί να συνδεθεί με Argo Workflows, Tekton ή Jenkins X για end-to-end αυτοματοποίηση.
- *Multi-user υποστήριξη & RBAC:* παρέχει απομόνωση χρηστών, ρόλους και ασφαλή συνεργασία σε κοινό cluster.
- *Επεκτασιμότητα και φορητότητα:* κάθε component τρέχει σε containers, επιτρέποντας εύκολη κλιμάκωση και αναπαραγωγιμότητα περιβαλλόντων.
- *Ενεργή κοινότητα & CNCF υποστήριξη:* διαθέτει ευρεία βάση χρηστών, συνεχή ανάπτυξη και υποστήριξη από μεγάλους cloud providers (Google, AWS, Azure).

Μειονεκτήματα:

- *Πολύπλοκη εγκατάσταση και συντήρηση:* απαιτεί πολλαπλά components (Istio, Dex, Pipelines, Katib, KServe), αυξάνοντας το λειτουργικό βάρος.
- *Υψηλή καμπύλη εκμάθησης:* προϋποθέτει καλή γνώση τόσο Kubernetes όσο και MLOps αρχιτεκτονικών.

- *Μεγάλη κατανάλωση πόρων:* κάθε component λειτουργεί ως ξεχωριστό pod, επιβαρύνοντας το cluster σε CPU, RAM και storage.
- *Ασυμβατότητες μεταξύ εκδόσεων:* οι συνεχείς ενημερώσεις (π.χ. KServe vs. Pipelines) μπορεί να προκαλέσουν δυσλειτουργίες.
- *Πολυπλοκότητα παραμετροποίησης:* η ενσωμάτωση με εξωτερικά εργαλεία (π.χ. monitoring, CI/CD) απαιτεί προσεκτικό σχεδιασμό.
- *Μεταβλητή εμπειρία ανά cloud provider:* η εγκατάσταση και λειτουργία διαφέρει σημαντικά μεταξύ AWS, GCP και on-premise διανομών.

Polyaxon

Το Polyaxon[98] είναι μια πλατφόρμα MLOps και ML experimentation σχεδιασμένη για την αυτοματοποίηση του κύκλου ζωής των πειραμάτων μηχανικής μάθησης πάνω σε Kubernetes. Συνδυάζει δυνατότητες εκτέλεσης, παρακολούθησης, διαχείρισης πόρων και version control για δεδομένα, κώδικα και αποτελέσματα, επιτρέποντας πλήρη ενοποίηση με Git, Docker και CI/CD pipelines. Το κύριο χαρακτηριστικό του είναι η δυνατότητα να οργανώνει και να αναπαράγει πειράματα, προσφέροντας ισχυρή υποστήριξη για hyperparameter tuning, tracking, artifact management και pipeline orchestration.

Λειτουργικά, το Polyaxon υποστηρίζει τη δηλωτική περιγραφή πειραμάτων μέσω YAML manifests, τα οποία περιγράφουν το περιβάλλον, τα δεδομένα, τις εξαρτήσεις και τις παραμέτρους εκτέλεσης. Παράλληλα, παρέχει ένα Web UI για τη διαχείριση projects, παρακολούθηση πειραμάτων σε πραγματικό χρόνο, και σύγκριση αποτελεσμάτων (metrics, loss, accuracy κ.λπ.), καθώς και CLI και REST API για αυτοματοποίηση. Επιπλέον, υποστηρίζει distributed training σε frameworks, όπως TensorFlow, PyTorch και MXNet, ενώ διαθέτει δυνατότητα εκτέλεσης jobs σε πολλαπλούς κόμβους μέσω Kubernetes operators.

Η ενσωμάτωσή του στο Kubernetes γίνεται κυρίως ως internal εργαλείο, μέσω Helm chart ή Kubernetes manifests που εγκαθιστούν τον Polyaxon operator, το control plane (scheduler, API, UI, tracking server) και τα worker pods. Όμως, μπορεί να ακολουθήσει και υβριδική, κατανεμημένη αρχιτεκτονική όπου συνδέεται με εξωτερικά object stores (S3, Azure Blob, GCS) για αποθήκευση artifacts και αποτελεσμάτων ενώ μπορεί να ενσωματώνεται με Git repositories, Docker registries, external ML frameworks και Polyaxon clients (πχ. Polyaxon CLI, SDKs, web UI). Παρότι υποστηρίζει συνδέσεις με εξωτερικά συστήματα, ο κύριος μηχανισμός εκτέλεσης και διαχείρισης παραμένει εντός του cluster, καθιστώντας το Polyaxon internal εργαλείο ως προς τον τρόπο λειτουργίας του. Η ενσωμάτωσή του στο Kubernetes πραγματοποιείται με διάφορους τρόπους: μέσω Polyaxon CLI για τη διαχείριση και εκτέλεση πειραμάτων, μέσω Polyaxon Operator που ενορχηστρώνει τα workloads και το control plane, καθώς και μέσω Helm charts ή YAML manifests για εγκατάσταση και παραμετροποίηση.

Πλεονεκτήματα:

- *Πλήρης MLOps πλατφόρμα:* Καλύπτει ολόκληρο τον κύκλο ζωής των πειραμάτων μηχανικής μάθησης — από τη δημιουργία, εκπαίδευση και αξιολόγηση μοντέλων μέχρι

[98] https://polyaxon.com/docs/

την παραγωγική τους υλοποίηση — επιτρέποντας ενοποιημένη διαχείριση μέσα από το Kubernetes.

- *Εύχρηστη διεπαφή:* Προσφέρει Web UI για οπτική διαχείριση και CLI για αυτοματισμούς, επιτρέποντας ταυτόχρονα παρακολούθηση metrics, logs και artifacts σε πραγματικό χρόνο.
- *Ενσωμάτωση με Git και containerized περιβάλλοντα:* Κάθε πείραμα εκτελείται σε reproducible container images, με πλήρη έλεγχο εκδόσεων κώδικα και παραμέτρων μέσω Git, διασφαλίζοντας επαναληψιμότητα και ιχνηλασιμότητα των αποτελεσμάτων
- *Υποστήριξη πολλαπλών ML frameworks:* Επιτρέπει την εκτέλεση workloads σε TensorFlow[99], PyTorch[100], MXNet και Scikit-learn[101], χωρίς ανάγκη αλλαγής υποδομής.
- *Hyperparameter tuning:* Διαθέτει ενσωματωμένους αλγορίθμους βελτιστοποίησης (grid search, random search, Bayesian optimization), μειώνοντας τον χρόνο εκπαίδευσης και εύρεσης των βέλτιστων παραμέτρων.
- *Tracking και reproducibility:* Καταγράφει αυτόματα αποτελέσματα, εκδόσεις και εξαρτήσεις, επιτρέποντας εύκολη αναπαραγωγή και σύγκριση πειραμάτων.
- *Υποστήριξη distributed training:* Διευκολύνει την εκτέλεση πειραμάτων σε πολλαπλούς κόμβους Kubernetes, αξιοποιώντας την κλιμάκωση του cluster.
- *Επεκτασιμότητα και αυτοματοποίηση:* Παρέχει REST API και integrations με CI/CD εργαλεία, επιτρέποντας την πλήρη αυτοματοποίηση των MLOps pipelines.

Μειονεκτήματα:

- *Πολυπλοκότητα εγκατάστασης:* Απαιτεί Helm, CRDs και ρύθμιση storage backends, κάτι που αυξάνει τη δυσκολία αρχικής παραμετροποίησης.
- *Αυξημένες απαιτήσεις πόρων:* Το control plane (API server, UI, tracking) καταναλώνει σημαντικούς υπολογιστικούς πόρους, ειδικά σε μεγάλα workloads.
- *Καμπύλη εκμάθησης:* Η χρήση YAML configurations και advanced λειτουργιών απαιτεί χρόνο εκμάθησης από τους νέους χρήστες.
- *Εξάρτηση από εξωτερικά συστήματα:* Για πλήρη λειτουργικότητα χρειάζεται integration με object storage, Git, Docker και ML frameworks.
- *Περιορισμένη κοινότητα:* Αν και ενεργό έργο, έχει μικρό οικοσύστημα, γεγονός που περιορίζει τη διαθεσιμότητα tutorials και community support.
- *Δυσκολία debugging:* Η αποσφαλμάτωση πειραμάτων σε distributed περιβάλλοντα με πολλούς κόμβους μπορεί να είναι περίπλοκη.

Ο παραπάτω πίνακας συνοψίζει την ανάλυση των 4 αυτών εργαλείων της τρέχουσας κατηγορίας.

[99] https://www.tensorflow.org/api_docs

[100] https://docs.pytorch.org/docs/stable/index.html

[101] https://scikit-learn.org/stable/

Πίνακας 18: Συγκριτική αξιολόγηση εργαλείων στην κατηγορία AI/ML

Εργαλείο	Κατηγορίες AI/ML	Πλεονεκτήματα	Μειονεκτήματα
DQN Scheduler	Resource Scheduling	Βελτιστοποιεί τη χρήση CPU/μνήμης, μειώνει latency, προσαρμόζεται δυναμικά σε μεταβαλλόμενα workloads, υποστηρίζει heterogeneous περιβάλλοντα, συνεχής μάθηση, ενσωμάτωση με monitoring εργαλεία για real-time feedback	Πολυπλοκότητα υλοποίησης και ρύθμισης, αυξημένες υπολογιστικές απαιτήσεις, έλλειψη σταθερότητας, περιορισμένη υιοθέτηση, δυσκολία debugging λόγω "black box" RL συμπεριφοράς
KubePipe	ML Pipeline Orchestration	Υψηλή επεκτασιμότητα, modular αρχιτεκτονική, end-to-end ML pipelines (preprocessing, training, deployment), συμβατότητα με TensorFlow, PyTorch, fault tolerance, native Kubernetes integration (CRDs & Operators), υψηλή απόδοση και αξιοπιστία	Απότομη καμπύλη μάθησης, δυσκολία ενσωμάτωσης legacy pipelines, αυξημένη κατανάλωση CPU/GPU, περιορισμένο community support, πολυπλοκότητα debugging
Kubeflow	ML Pipeline Orchestration Data & Feature Management Model Training & Serving	Ενοποιημένη MLOps αρχιτεκτονική, υποστήριξη πολλαπλών ML frameworks (TensorFlow, PyTorch, Scikit-learn), αυτοματοποιημένα pipelines (Kubeflow Pipelines), hyperparameter tuning (Katib), model serving (KServe), CI/CD & GitOps integration, RBAC & multi-user isolation, επεκτασιμότητα, CNCF υποστήριξη και ενεργή κοινότητα	Πολυπλοκότητα εγκατάστασης (πολλαπλά components: Istio, Dex, Katib κ.ά.), υψηλή κατανάλωση πόρων, ασυμβατότητες εκδόσεων, απαιτεί γνώση Kubernetes & MLOps, δυσκολία debugging σε distributed περιβάλλοντα
Polyaxon	Data & Feature Management Model Training & Serving	Πλήρης MLOps ροή εργασίας, Web UI & CLI, reproducibility με Git, containerized περιβάλλοντα, hyperparameter tuning (grid, random, Bayesian),	Πολύπλοκη εγκατάσταση (Helm, CRDs, storage setup), υψηλές απαιτήσεις πόρων, καμπύλη εκμάθησης (YAML configs), εξάρτηση από Git & storage συστήματα, μικρότερο οικοσύστημα υποστήριξης,

		tracking & versioning, distributed training, REST API για αυτοματοποίηση, υποστήριξη TensorFlow, PyTorch, MXNet, CI/CD integration	δύσκολο debugging σε multi-node clusters

4.2.9 Service Mesh

Η κατηγορία των εργαλείων Service Mesh περιλαμβάνει τεχνολογίες που λειτουργούν πάνω από το επίπεδο του δικτύου (application layer – Layer 7) και στοχεύουν στη διαχείριση, ασφάλεια και παρακολούθηση της επικοινωνίας μεταξύ μικροϋπηρεσιών (microservices) μέσα σε ένα Kubernetes cluster. Σε αντίθεση με τα εργαλεία της κατηγορίας Networking (όπως Calico ή Cilium), τα οποία εστιάζουν στη συνδεσιμότητα και δρομολόγηση πακέτων σε επίπεδο IP (Layer 3–4), τα Service Mesh εργαλεία επεμβαίνουν στη συμπεριφορά της επικοινωνίας μεταξύ εφαρμογών, προσθέτοντας δυνατότητες, όπως μικροεπίπεδη πολιτική ασφαλείας (mTLS), έξυπνη δρομολόγηση αιτημάτων (traffic routing), παρακολούθηση απόδοσης και ιχνηλασιμότητα (observability, tracing) και μηχανισμούς ανθεκτικότητας, όπως retries και circuit breaking.

Λειτουργικά, τα Service Mesh εργαλεία υλοποιούνται συνήθως με δύο τρόπους:

1. Sidecar-based αρχιτεκτονική, όπου σε κάθε pod εκτελείται ένας proxy (π.χ. Envoy) που αναλαμβάνει τη δρομολόγηση και παρακολούθηση της επικοινωνίας.

2. eBPF-based αρχιτεκτονική, όπου η διαχείριση της επικοινωνίας γίνεται απευθείας στο kernel μέσω eBPF προγραμμάτων, μειώνοντας το latency και την κατανάλωση πόρων.

Αυτή η δεύτερη προσέγγιση αποτελεί τη νεότερη τάση, καθώς συνδυάζει τις λειτουργίες παρακολούθησης και ασφάλειας χωρίς την ανάγκη για πρόσθετους sidecars, γεγονός που μειώνει σημαντικά το operational overhead (CNCF, 2024).

Εργαλεία όπως το Istio, το Linkerd και το Cilium Service Mesh αποτελούν τις πιο δημοφιλείς υλοποιήσεις. Το Istio είναι γνωστό για την πληρότητα των λειτουργιών του και την ενσωμάτωσή του με observability εργαλεία όπως Prometheus και Grafana, ενώ το Linkerd προσφέρει ελαφρύτερη αρχιτεκτονική, κατάλληλη για clusters με περιορισμένους πόρους. Από την άλλη, το Cilium Service Mesh αξιοποιεί πλήρως το eBPF, παρέχοντας mesh δυνατότητες χωρίς sidecar containers.

Η χρήση ενός Service Mesh καθίσταται ιδιαίτερα σημαντική σε παραγωγικά (production-grade) Kubernetes περιβάλλοντα, καθώς επιτρέπει την εφαρμογή πολιτικών ασφάλειας και τη συλλογή μετρικών επικοινωνίας χωρίς να απαιτούνται αλλαγές στον κώδικα των εφαρμογών.

ISTIO

Το Istio[102] αποτελεί την πλέον διαδεδομένη πλατφόρμα service mesh στο οικοσύστημα του Kubernetes και υποστηρίζεται επισήμως από το CNCF. Πρόκειται για ένα εργαλείο που επιτρέπει τη διαχείριση, ασφάλεια και παρακολούθηση της επικοινωνίας μεταξύ μικροϋπηρεσιών, χωρίς να απαιτούνται αλλαγές στον πηγαίο κώδικα (Sedghpour & Townend, 2022). Η λειτουργία του

[102] https://istio.io/latest/docs/

βασίζεται στην παρεμβολή ενός sidecar proxy (Envoy) σε κάθε pod, ο οποίος διαχειρίζεται τη ροή της κυκλοφορίας και συλλέγει πληροφορίες για την παρακολούθηση και την ασφάλεια. Το Istio περιλαμβάνει ένα control plane (istiod), υπεύθυνο για τον συντονισμό των proxies και την εφαρμογή πολιτικών, καθώς και ένα data plane, το οποίο χειρίζεται την πραγματική ροή των δεδομένων.

Σε λειτουργικό επίπεδο, το Istio προσφέρει προηγμένες δυνατότητες διαχείρισης κυκλοφορίας, επιτρέποντας canary releases, όπου μια νέα έκδοση μιας υπηρεσίας λαμβάνει σταδιακά ένα μικρό ποσοστό της κίνησης ώστε να δοκιμαστεί με ασφάλεια πριν από πλήρη διάθεση, traffic mirroring για δοκιμή αιτημάτων χωρίς επίδραση στο παραγωγικό περιβάλλον, καθώς και μηχανισμούς load balancing και retry policies για βελτίωση της αξιοπιστίας. Επιπλέον, υποστηρίζει fault injection για προσομοίωση καθυστερήσεων ή σφαλμάτων, διευκολύνοντας έτσι τον έλεγχο ανθεκτικότητας των μικροϋπηρεσιών. Στον τομέα της ασφάλειας, το Istio εφαρμόζει mutual TLS (mTLS) για κρυπτογραφημένη επικοινωνία μεταξύ υπηρεσιών, ενώ επιτρέπει την ταυτοποίηση και τον έλεγχο πρόσβασης βάσει ρόλων (RBAC). Οι πολιτικές πρόσβασης μπορούν να ενσωματωθούν με συστήματα όπως το Open Policy Agent (OPA) για κεντρική διαχείριση.

Όσον αφορά την παρατηρησιμότητα, το Istio συλλέγει μετρικές, logs και traces από τους Envoy proxies, παρέχοντας πλήρη εικόνα της εσωτερικής κίνησης στο cluster. Η ενσωμάτωσή του με εργαλεία όπως το Prometheus για συλλογή μετρικών, το Grafana για δημιουργία dashboards και το Jaeger για distributed tracing επιτρέπει τη βαθιά ανάλυση επιδόσεων και τη διάγνωση σφαλμάτων. Επιπλέον, διαθέτει μηχανισμούς για policy enforcement και telemetry, μέσω των οποίων εφαρμόζονται κανόνες πρόσβασης, quotas και περιορισμοί χρήσης πόρων.

Η ενσωμάτωση του Istio στο Kubernetes πραγματοποιείται μέσω Helm charts, του εργαλείου istioctl ή του Istio Operator, ενώ υποστηρίζεται automatic sidecar injection που προσθέτει αυτόματα έναν Envoy proxy δίπλα σε κάθε pod. Η λειτουργία του βασίζεται σε δηλωτικά αντικείμενα (Custom Resource Definitions) όπως τα VirtualService, DestinationRule και Gateway, που επιτρέπουν τον ακριβή έλεγχο της δρομολόγησης, των πολιτικών και των κανόνων ασφαλείας. Το Istio κατατάσσεται ως internal εργαλείο, γιατί εκτελείται εξ ολοκλήρου εντός του Kubernetes cluster, αξιοποιώντας το control plane και το data plane του ίδιου του συστήματος. Όλα τα βασικά components του — όπως το istiod (control plane), οι Envoy sidecar proxies (data plane) και τα Custom Resource Definitions (CRDs) όπως τα VirtualService, DestinationRule και Gateway — τρέχουν μέσα στο ίδιο το cluster, υπό την εποπτεία του Kubernetes API Server

Πλεονεκτήματα:

- *Πληρότητα λειτουργιών*: Το Istio είναι ένα από τα πιο ώριμα και ολοκληρωμένα service mesh εργαλεία, παρέχοντας προηγμένες δυνατότητες διαχείρισης κυκλοφορίας, ασφάλειας και παρακολούθησης μικροϋπηρεσιών.
- *Ενσωμάτωση με observability εργαλεία*: Συλλέγει μετρικές, logs και traces μέσω telemetry και ενσωματώνεται άμεσα με εργαλεία όπως Prometheus, Grafana, Jaeger και Kiali, επιτρέποντας πλήρη ορατότητα της ροής δεδομένων στο cluster.
- *Ισχυρή ασφάλεια*: Υποστηρίζει mutual TLS (mTLS) για κρυπτογραφημένη επικοινωνία μεταξύ υπηρεσιών και μηχανισμούς policy enforcement, που διασφαλίζουν την εφαρμογή πολιτικών πρόσβασης και συμμόρφωσης.

- *Advanced traffic management*: Παρέχει δυνατότητες όπως canary deployments (σταδιακή διάθεση νέων εκδόσεων), rate limiting (έλεγχος ρυθμού αιτημάτων), circuit breaking (απομόνωση αποτυχημένων υπηρεσιών) και fault injection (δοκιμές ανθεκτικότητας).
- *Ευρεία υποστήριξη και κοινότητα*: Αναπτύσσεται ενεργά υπό την ομπρέλα του CNCF, με μεγάλη κοινότητα χρηστών και εκτενή τεκμηρίωση, κάτι που ενισχύει τη διαλειτουργικότητα και τη συνεχή εξέλιξή του.

Μειονεκτήματα:

- *Αυξημένη πολυπλοκότητα*: Η εγκατάσταση και ρύθμιση του Istio είναι σύνθετη, καθώς αποτελείται από πολλά επιμέρους components (Envoy proxy, Pilot, Mixer, Citadel) που απαιτούν εξειδικευμένες γνώσεις.
- *Runtime overhead*: Η χρήση sidecar proxies σε κάθε pod αυξάνει την κατανάλωση CPU και μνήμης, δημιουργώντας επιπλέον φόρτο στο σύστημα — κάτι που γίνεται ιδιαίτερα αισθητό σε μικρότερα clusters.
- *Πολυπλοκότητα συντήρησης*: Οι συνεχείς ενημερώσεις και η ανάγκη συμβατότητας με νέες εκδόσεις του Kubernetes προσθέτουν διοικητικό βάρος στις ομάδες DevOps.
- *Δυσκολία debugging*: Η ύπαρξη πολλών επιπέδων επικοινωνίας (service, proxy, mesh control plane) μπορεί να περιπλέξει τον εντοπισμό σφαλμάτων και τη διαχείριση απόδοσης.

Linkerd

Το Linkerd[103] αποτελεί ένα από τα πρώτα και πιο ελαφριά service mesh εργαλεία για Kubernetes, το οποίο επικεντρώνεται στην απλότητα, την ασφάλεια και τη χαμηλή κατανάλωση πόρων. Σε αντίθεση με πιο σύνθετα συστήματα όπως το Istio, το Linkerd προσφέρει μια πιο «minimal» αρχιτεκτονική που το καθιστά ιδανικό για clusters μικρότερης κλίμακας ή περιβάλλοντα με περιορισμένους πόρους.

Λειτουργικά, το Linkerd αναλαμβάνει τη διαχείριση της επικοινωνίας μεταξύ μικροϋπηρεσιών, εισάγοντας αυτόματα sidecar proxies σε κάθε pod για τη δρομολόγηση, την παρακολούθηση και την ασφάλιση των αιτημάτων. Υποστηρίζει mTLS (mutual Transport Layer Security) εξ ορισμού, εξασφαλίζοντας κρυπτογραφημένη επικοινωνία μεταξύ των υπηρεσιών χωρίς την ανάγκη χειροκίνητης ρύθμισης πιστοποιητικών. Επιπλέον, ενσωματώνει μηχανισμούς παρακολούθησης (observability) όπως latency metrics, success/error rates και request volume, τα οποία είναι ορατά μέσω ενός ενσωματωμένου web dashboard ή μέσω εργαλείων όπως το Grafana και το Prometheus. (Sedghpour & Townend, 2022). Πέραν της ασφάλειας και της παρακολούθησης, το Linkerd διαθέτει health checks και failure detection, επιτρέποντας στον χρήστη να εντοπίζει και να απομονώνει υπηρεσίες που δεν ανταποκρίνονται σωστά. Με αυτόν τον τρόπο, αυξάνει την

[103] https://linkerd.io/2.18/getting-started/

αξιοπιστία και τη σταθερότητα των εφαρμογών. Είναι ιδιαίτερα κατάλληλο για clusters με περιορισμένους πόρους ή οργανισμούς που επιθυμούν εύκολη διαχείριση (Zhu, et al., 2023).

Η ενσωμάτωση του Linkerd στο Kubernetes πραγματοποιείται είτε μέσω της εντολής linkerd install που εγκαθιστά τα απαραίτητα components στο cluster, είτε μέσω Helm charts, τα οποία παρέχουν μεγαλύτερη ευελιξία στις ρυθμίσεις και διευκολύνουν την αναβάθμιση ή απεγκατάσταση του mesh. Η διαχείρισή του γίνεται από CLI ή μέσω του web UI, και η λειτουργία του βασίζεται πλήρως στο Kubernetes control plane, χωρίς την ανάγκη εξωτερικών εξαρτήσεων. Το Linkerd ανήκει στην κατηγορία των internal εργαλείων, καθώς λειτουργεί εξολοκλήρου εντός του Kubernetes cluster και ενσωματώνεται άμεσα με το control plane του. Όλα τα components του (proxy, control plane, metrics collectors) εγκαθίστανται στο ίδιο το cluster, χωρίς να απαιτείται εξωτερικό σύστημα ή υπηρεσία για τη λειτουργία του.

Πλεονεκτήματα:

- *Ελαφρύ και αποδοτικό design*: Το Linkerd χρησιμοποιεί Rust proxies αντί για Envoy, προσφέροντας χαμηλό runtime overhead και μειωμένη κατανάλωση πόρων.
- *Ασφάλεια εξ ορισμού*: Υποστηρίζει υποχρεωτικό mTLS (mutual Transport Layer Security) για κρυπτογράφηση της επικοινωνίας μεταξύ μικροϋπηρεσιών χωρίς πρόσθετη ρύθμιση.
- *Ευκολία εγκατάστασης*: Η εγκατάσταση και διαχείριση πραγματοποιούνται εύκολα μέσω του CLI (linkerd install) ή μέσω Helm charts.
- *Ενσωματωμένο dashboard*: Παρέχει γραφική διεπαφή για παρακολούθηση metrics, latency και υγείας υπηρεσιών σε πραγματικό χρόνο.
- *Κατάλληλο για μικρά ή edge clusters*: Λόγω του μικρού αποτυπώματος, είναι ιδανικό για περιβάλλοντα με περιορισμένους πόρους.

Μειονεκτήματα:

- *Περιορισμένες προηγμένες λειτουργίες*: Δεν υποστηρίζει πλήρως canary deployments, traffic mirroring ή policy enforcement, σε αντίθεση με το Istio.
- *Μικρότερη ευελιξία προσαρμογής*: Οι επιλογές fine-tuning και επέκτασης είναι πιο περιορισμένες.
- *Μικρότερη κοινότητα και οικοσύστημα*: Η υποστήριξη και η διαθεσιμότητα plugins ή integrations είναι πιο περιορισμένες.
- *Περιορισμένο observability σε μεγάλα clusters*: Αν και επαρκές για μικρές υποδομές, απαιτεί επιπλέον εργαλεία για πλήρη παρακολούθηση σε εκτεταμένα περιβάλλοντα.

eBPF-based Mesh (Cilium)

Το Cilium Service Mesh[104] αποτελεί μια σύγχρονη, eBPF-based υλοποίηση service mesh που στοχεύει στην παροχή ασφαλούς και αποδοτικής επικοινωνίας μεταξύ μικροϋπηρεσιών χωρίς την ανάγκη sidecar proxies, όπως συμβαίνει σε παραδοσιακές λύσεις (π.χ. Istio ή Linkerd). Αντί να εισάγει επιπλέον containers ανά pod, το Cilium αξιοποιεί το eBPF (Extended Berkeley Packet Filter) — μια τεχνολογία του Linux kernel που επιτρέπει την εκτέλεση ελαφρών προγραμμάτων απευθείας μέσα στον πυρήνα του λειτουργικού συστήματος. Μέσω αυτής της προσέγγισης, το Cilium εκτελεί λειτουργίες δικτύωσης, παρακολούθησης και ασφάλειας σε επίπεδο kernel, εξασφαλίζοντας χαμηλό latency, μειωμένο overhead και υψηλή απόδοση.

Λειτουργικά, το Cilium Service Mesh παρέχει zero-trust security με αυτόματη ταυτοποίηση workloads και επιβολή πολιτικών πρόσβασης (Network & L7 Policies), καθώς και observability μέσω ενσωμάτωσης με το Hubble, το εργαλείο παρακολούθησης ροών δικτύου του Cilium. Επιπλέον, υποστηρίζει traffic management λειτουργίες όπως load balancing, traffic redirection και service discovery, ενώ η απουσία sidecars μειώνει δραστικά την κατανάλωση πόρων ανά pod. Η ενσωμάτωση στο Kubernetes πραγματοποιείται μέσω Cilium agents που αναπτύσσονται στους κόμβους και αλληλεπιδρούν άμεσα με το API server, παρέχοντας native integration με τα Kubernetes Services και Network Policies.

Η προσέγγιση του Cilium θεωρείται αντιπροσωπευτική της νέας γενιάς service meshes, οι οποίες δίνουν έμφαση στην απλότητα, την απόδοση και την ασφάλεια, καθιστώντας το ιδιαίτερα κατάλληλο για μεγάλα production περιβάλλοντα, cloud-native αρχιτεκτονικές και clusters που απαιτούν αυστηρές επιδόσεις δικτύου.

Η ενσωμάτωση του Cilium ως Service Mesh στο Kubernetes γίνεται χωρίς τη χρήση sidecars, αξιοποιώντας την τεχνολογία eBPF για την εκτέλεση λειτουργιών παρακολούθησης και ασφάλειας απευθείας στον kernel. Αυτό σημαίνει ότι αποφεύγεται το runtime overhead που προκαλούν τα sidecar proxies (όπως ο Envoy στο Istio). Το Cilium Service Mesh ανήκει στην κατηγορία των internal εργαλείων, καθώς λειτουργεί εντός του ίδιου του Kubernetes cluster. Η εγκατάσταση πραγματοποιείται μέσω Helm charts ή με το Cilium CLI, ενώ ο Cilium Operator αναλαμβάνει τη διαχείριση των CRDs που απαιτούνται για την ενεργοποίηση των service mesh δυνατοτήτων.

Πλεονεκτήματα:

- *Εξαιρετική απόδοση χωρίς sidecar proxies*: με αποτέλεσμα χαμηλότερο latency και μειωμένη κατανάλωση πόρων.
- *Native observability μέσω του Hubble*: επιτρέπει πλήρη ορατότητα στη ροή της κυκλοφορίας και στους κανόνες ασφάλειας.
- *Προηγμένες πολιτικές ασφαλείας σε επίπεδο εφαρμογής*: βασισμένες σε ταυτότητες (identity-based security).

[104] https://cilium.io/use-cases/service-mesh/

- *Ευέλικτη χρήση*: λειτουργεί τόσο ως CNI plugin όσο και ως Service Mesh, επιτρέποντας ενιαία διαχείριση δικτύου και ασφάλειας.
- *Εξαιρετική ενσωμάτωση με Kubernetes APIs*: υποστηρίζει network policies και service discovery σε kernel-level.

Μειονεκτήματα:

- *Εξειδικευμένες απαιτήσεις*: Απαιτεί σύγχρονο Linux kernel (>= 4.19) και εξειδικευμένη γνώση του eBPF για προχωρημένη παραμετροποίηση.
- *Περιορισμένη υποστήριξη*: Περιορισμένη τεκμηρίωση και μικρότερη κοινότητα χρηστών σε σύγκριση με λύσεις όπως Istio ή Linkerd.
- *Πολυπλοκότητα στη ρύθμιση προηγμένων πολιτικών*: ειδικά σε μεγάλα clusters με πολλαπλά namespaces.
- *Μη πλήρης ωριμότητα*: Ορισμένες λειτουργίες (π.χ. advanced routing ή cross-cluster policies) βρίσκονται ακόμη σε φάση ανάπτυξης.

Ο παραπάτω πίνακας συνοψίζει την ανάλυση των 3 αυτών εργαλείων της τρέχουσας κατηγορίας.

Πίνακας 19: Συγκριτική Αξιολόγηση Εργαλείων Service Mesh

Εργαλείο	**Λειτουργικότητα / Κατηγορίες Service Mesh**	**Πλεονεκτήματα**	**Μειονεκτήματα**
Istio	Όλες οι κατηγορίες καλύπτονται (Traffic Management, Observability & Telemetry, Security & Encryption, Resilience & Fault Injenction)	Πληρότητα λειτουργιών (advanced routing, fault injection, policy enforcement), Ισχυρή ασφάλεια με mTLS και RBAC, Πλήρης παρατηρησιμότητα μέσω telemetry (Prometheus, Grafana, Jaeger, Kiali), Υποστήριξη canary deployments, rate limiting, retries, Ενεργή κοινότητα και CNCF υποστήριξη	Αυξημένη πολυπλοκότητα εγκατάστασης και συντήρησης, Υψηλό runtime overhead λόγω sidecar proxies, Δυσκολία debugging σε πολύπλοκα περιβάλλοντα, Εξάρτηση από πολλαπλά components (istiod, Envoy, Pilot, Mixer), Μεγάλη καμπύλη εκμάθησης
Linkerd	Όλες οι κατηγορίες (Lightweight Service Mesh με βασικές λειτουργίες)	Ελαφρύ, γρήγορο και απλό στη χρήση, Ενσωματωμένο dashboard για metrics και health checks, Υποχρεωτικό mTLS εξ ορισμού (zero-config security), Μικρό	Περιορισμένες προηγμένες λειτουργίες (π.χ. canary, traffic mirroring), Μικρότερη κοινότητα και οικοσύστημα, Περιορισμένη δυνατότητα fine-tuning και observability σε μεγάλα clusters, Μη πλήρης

		αποτύπωμα – ιδανικό για edge ή μικρά clusters, Rust-based proxies με χαμηλό latency	υποστήριξη policy enforcement
Cilium	Κυρίως Observability & Telemetry, Μερικώς Traffic Management και Security & Encryption (Service Mesh χωρίς sidecars, βασισμένο σε eBPF)	Υψηλή απόδοση, χαμηλό latency και μικρό overhead, Native observability μέσω Hubble, Identity-based security policies σε επίπεδο εφαρμογής, Ενοποιημένη διαχείριση CNI και Service Mesh, Άριστη ενσωμάτωση με Kubernetes APIs, Ιδανικό για περιβάλλοντα υψηλών επιδόσεων	Απαιτεί νεότερο Linux kernel (≥4.19), Εξειδικευμένη γνώση eBPF για παραμετροποίηση, Μικρότερη κοινότητα και περιορισμένη τεκμηρίωση, Ορισμένες λειτουργίες (cross-cluster policies) υπό ανάπτυξη, Πολυπλοκότητα σε μεγάλα clusters με πολλές πολιτικές

Πηγές: (Sedghpour & Townend, 2022), (Johng et al., 2019), (Cloud Native Computing Foundation (CNCF), 2024)

ΚΕΦΑΛΑΙΟ 5 – ΣΥΓΚΡΙΣΗ ΕΡΓΑΛΕΙΩΝ KUBERNETES

Η παρούσα ενότητα αποσκοπεί στην **συγκριτική αξιολόγηση** των εργαλείων Kubernetes που παρουσιάστηκαν και αναλύθηκαν στο Κεφάλαιο 5, με στόχο την ανάδειξη των πλεονεκτημάτων, αδυναμιών και πιθανών περιορισμών τους. Η σύγκριση αυτή βασίζεται σε **ποσοτικά και ποιοτικά κριτήρια**, τα οποία είτε προκύπτουν απευθείας από την πρόσφατη επιστημονική βιβλιογραφία και τεχνική τεκμηρίωση που μελετήθηκε, είτε προκύπτουν συμπερασματικά/λογικά από την ανάλυση που προηγήθηκε.

Για κάθε κατηγορία εργαλείων, χρησιμοποιείται ένα ενιαίο σύνολο **κριτηρίων αξιολόγησης** (ως προς όλες τις κατηγορίες) που καλύπτει πτυχές, όπως η απόδοση, η ευκολία ενσωμάτωσης, η κλιμακωσιμότητα, η ασφάλεια, η διαλειτουργικότητα και η υποστήριξη από την κοινότητα. Το σύνολο αυτό αναλύεται ώστε να προσδιοριστεί η σημασιολογία των κριτηρίων και ο τρόπος αξιολόγησής τους. Στη συνέχεια, παρουσιάζονται οι **πίνακες σύγκρισης** που συνοψίζουν τα ευρήματα και διευκολύνουν την εξαγωγή συμπερασμάτων τόσο σε επίπεδο κατηγορίας όσο και συνολικά για το οικοσύστημα του Kubernetes.

Η προσέγγιση αυτή επιτρέπει την **αντικειμενική αποτίμηση** των εργαλείων και τη διαμόρφωση μιας τεκμηριωμένης άποψης για την επιλογή των καταλληλότερων λύσεων, ανάλογα με τις απαιτήσεις και τις προτεραιότητες κάθε οργανισμού.

5.1 Κριτήρια Σύγκρισης

Η συγκριτική αξιολόγηση των εργαλείων του οικοσυστήματος Kubernetes απαιτεί την υιοθέτηση ενός συνόλου σαφώς ορισμένων κριτηρίων, τα οποία θα πρέπει να επιτρέπουν την αντικειμενική και συνεπή αποτίμηση εργαλείων με διαφορετικές λειτουργίες και τεχνολογικά χαρακτηριστικά. Τα κριτήρια που ακολουθούν διαμορφώθηκαν με βάση: (α) διεθνή πρότυπα αξιολόγησης λογισμικού, όπως το ISO/IEC 25010:2011 για τα χαρακτηριστικά ποιότητας, καθώς και κατευθυντήριες οδηγίες του Cloud Native Computing Foundation (CNCF, 2023) για την αποτίμηση cloud-native εργαλείων και (β) τα συμπεράσματα από την ανάλυση στο προηγούμενο κεφάλαιο.

Κάθε κριτήριο συνοδεύεται από περιγραφή της σημασίας του, καθώς και από μια προκαθορισμένη κλίμακα τιμών που χρησιμοποιείται για την αποτίμηση όλων των εργαλείων, προσαρμοσμένη στο πλαίσιο της εκάστοτε λειτουργικής κατηγορίας. Η χρήση κοινών κριτηρίων για όλες τις κατηγορίες εξασφαλίζει τη συγκρισιμότητα, ενώ η εξειδίκευση της ερμηνείας τους ανά κατηγορία (π.χ. Monitoring, Security, CI/CD) επιτρέπει την αποτύπωση των ιδιαίτερων απαιτήσεων και προτεραιοτήτων κάθε τομέα. Ειδικότερα, τα κριτήρια παρουσιάζονται ως εξής:

1. Απόδοση (Performance)

Αξιολογεί την ικανότητα του εργαλείου να εκτελεί τις λειτουργίες του αποτελεσματικά, ακόμη και υπό υψηλό φόρτο, χωρίς να υποβαθμίζει την απόδοση του Kubernetes cluster (στον οποίο ενσωματώνεται ή με τον οποίο διαλειτουργεί). Περιλαμβάνει δείκτες, όπως ο χρόνος απόκρισης

(latency), ρυθμός απόδοσης (throughput) και η κατανάλωση πόρων (CPU, μνήμη, δίκτυο). Στο Kubernetes, η υψηλή απόδοση εξασφαλίζει ότι οι εφαρμογές και τα workloads παραμένουν διαθέσιμα και ανταποκρίνονται γρήγορα. Οι βαθμολογίες αποτίμησης (1–5) βασίζονται σε συνθετική, ποιοτική εκτίμηση που αντλείται από βιβλιογραφικές πηγές, τεχνική τεκμηρίωση και συγκριτικές μελέτες, και όχι σε άμεσες ποσοτικές μετρήσεις, προκειμένου να διασφαλιστεί η συγκρισιμότητα μεταξύ εργαλείων με διαφορετικά χαρακτηριστικά.

Κλίμακα Αποτίμησης: 1–5 (1 = χαμηλή απόδοση, 5 = εξαιρετική απόδοση).

Απόδοση Τιμών:

- **1 →** Πολύ χαμηλή απόδοση, η οποία εκδηλώνεται μέσω σημαντικών καθυστερήσεων (latency) ή υπερβολικής κατανάλωσης πόρων. Η εκτίμηση βασίζεται σε συνδυαστική αξιολόγηση δεικτών, όπως ο ρυθμός απόδοσης (throughput) και η χρήση υπολογιστικών πόρων, ενώ τα κατώφλια καθορίζονται συγκριτικά, σύμφωνα με τιμές αναφοράς που προκύπτουν από τη βιβλιογραφία και την τεχνική τεκμηρίωση κάθε εργαλείου.
- **2 →** Χαμηλή απόδοση, με εμφανή μείωση επιδόσεων ακόμη και υπό μεσαίο φόρτο, ενώ σε υψηλό φόρτο παρατηρούνται σημαντικές καθυστερήσεις ή υπερκατανάλωση πόρων.
- **3 →** Μέτρια απόδοση, επαρκής για βασικά ή σταθερά workloads με περιορισμένες απαιτήσεις, αλλά με εμφανή πτώση απόδοσης όταν αυξάνεται ο φόρτος πέραν του συνηθισμένου
- **4 →** Υψηλή απόδοση με μικρά μόνο ζητήματα σε μεγάλα workloads.
- **5 →** Εξαιρετική απόδοση και πλήρως βελτιστοποιημένη χρήση πόρων.

2. Ευκολία Ενσωμάτωσης

Αξιολογεί το βαθμό δυσκολίας στην εγκατάσταση και τη λειτουργική ενσωμάτωση του εργαλείου σε ένα υπάρχον Kubernetes περιβάλλον. Το κριτήριο συνδυάζει δύο πτυχές:

- Ευκολία εγκατάστασης, η οποία εξαρτάται από τη διαθεσιμότητα αυτοματοποιημένων μηχανισμών (π.χ. Helm charts, Operators, CLI installers, APIs ή scripts), και
- Ευκολία παραμετροποίησης και χρήσης, η οποία σχετίζεται με την ύπαρξη τεκμηρίωσης, οδηγών και παραδειγμάτων που υποστηρίζουν τόσο βασικά όσο και εξειδικευμένα σενάρια υλοποίησης.

Ένα εργαλείο με υψηλή ευκολία ενσωμάτωσης επιτρέπει την ταχεία εγκατάσταση, μειώνει τον χρόνο ρύθμισης και περιορίζει τα σφάλματα κατά τη φάση διασύνδεσης με άλλα συστήματα ή APIs

Κλίμακα Αποτίμησης: 1–5 (1 = δύσκολη ενσωμάτωση, 5 = πολύ εύκολη ενσωμάτωση).

Απόδοση Τιμών:

- **1 →** *Πολύ δύσκολη ενσωμάτωση*: Το εργαλείο δεν παρέχει καθόλου εύκολους μηχανισμούς εγκατάστασης. Απουσιάζουν Helm charts, Operators ή ολοκληρωμένα manifests, με αποτέλεσμα η εγκατάσταση να απαιτεί εξ' ολοκλήρου χειροκίνητη ρύθμιση (π.χ. δημιουργία/παραμετροποίηση YAMLs από το μηδέν, custom scripts που δεν είναι

πάντα συμβατά). Σε αυτή την περίπτωση, η επιτυχής ενσωμάτωση προϋποθέτει βαθιά τεχνική γνώση του Kubernetes cluster καθώς και ικανότητες troubleshooting.

- **2 →** *Δύσκολη ενσωμάτωση*: Η εγκατάσταση του εργαλείου είναι δυνατή, αλλά δεν είναι πλήρως αυτοματοποιημένη. Παρέχονται μερικά εργαλεία (π.χ. Helm chart ή βασικά manifests), όμως απαιτούνται επιπλέον χειροκίνητες ρυθμίσεις για να λειτουργήσει σωστά (π.χ. προσαρμογή CRDs, RBAC, integrations). Ο χρήστης χρειάζεται εμπειρία στο Kubernetes, αλλά όχι σε βάθος γνώση της εσωτερικής λειτουργίας του εργαλείου.
- **3 →** *Μέτρια ευκολία ενσωμάτωσης*: Το εργαλείο μπορεί να εγκατασταθεί χωρίς ιδιαίτερα εμπόδια, ωστόσο απαιτεί πρόσθετη παραμετροποίηση. Η ύπαρξη πλήρους τεκμηρίωσης καθιστά την ενσωμάτωση εφικτή, αλλά όχι άμεσα αυτοματοποιημένη. Ο χρήστης χρειάζεται να ακολουθήσει οδηγίες και να προσαρμόσει παραμέτρους για να επιτύχει ορθή λειτουργία σε όλα τα σενάρια χρήσης.
- **4 →** *Σχετικά εύκολη και καλή ενσωμάτωση*: Το εργαλείο προσφέρει έτοιμα Helm charts ή Operators που απλοποιούν την εγκατάσταση και τη βασική διασύνδεση με το Kubernetes. Ωστόσο, για πιο σύνθετα σενάρια χρήσης ίσως χρειάζονται επιπλέον ρυθμίσεις. Παράδειγμα: Linkerd με linkerd install, όπου η βασική εγκατάσταση είναι απλή, αλλά η παραμετροποίηση advanced χαρακτηριστικών απαιτεί χειροκίνητες παρεμβάσεις
- **5 →** *Εξαιρετικά εύκολη, αυτοματοποιημένη ή native ενσωμάτωση*: Η εγκατάσταση είναι πλήρως αυτοματοποιημένη ή ακόμη και native (δηλαδή το εργαλείο ενσωματώνεται απευθείας στο Kubernetes API χωρίς να χρειάζεται ιδιαίτερη παραμετροποίηση από τον χρήστη). Ο χρήστης μπορεί με ελάχιστες εντολές να έχει πλήρως λειτουργικό το εργαλείο. Παράδειγμα: kubectl ή metrics-server που είναι native components του Kubernetes.

3. Επεκτασιμότητα

Αξιολογεί τη δυνατότητα του εργαλείου να προσαρμόζεται ή να επεκτείνεται λειτουργικά, ώστε να καλύπτει νέες ανάγκες χωρίς να απαιτούνται ριζικές αλλαγές στον πυρήνα του. Στο οικοσύστημα του Kubernetes, η επεκτασιμότητα συνδέεται με τη δυνατότητα προσθήκης νέων δυνατοτήτων μέσω plugins, APIs, Operators, Custom Resource Definitions (CRDs) ή Webhooks, καθώς και με τη συμβατότητα με άλλες υποδομές του cluster (π.χ. Prometheus, Grafana, OPA).

Ένα εργαλείο με υψηλή επεκτασιμότητα μπορεί να ενσωματωθεί σε πολύπλοκα περιβάλλοντα και να εξελιχθεί μαζί με τις απαιτήσεις του οργανισμού, μειώνοντας έτσι την ανάγκη αντικατάστασης ή ανασχεδίασης στο μέλλον.

Η αποτίμηση της επεκτασιμότητας πραγματοποιείται σε ποιοτικό επίπεδο με βάση γνώσεις και πληροφορίες που προκύπτουν από την τεχνική τεκμηρίωση, τη βιβλιογραφία και τα διαθέσιμα APIs κάθε εργαλείου, ώστε να διασφαλίζεται η συγκρισιμότητα μεταξύ διαφορετικών λύσεων.

Κλίμακα Αποτίμησης: 1–5 (1 = χαμηλή επεκτασιμότητα, 5 = άριστη επεκτασιμότητα).

Απόδοση Τιμών:

- **1** → *Πολύ χαμηλή επεκτασιμότητα*: Δεν υποστηρίζει καμία μορφή επέκτασης ή παραμετροποίησης πέρα από τις προκαθορισμένες ρυθμίσεις. *Παράδειγμα*: Μικρά CLI εργαλεία, όπως το kubectl, τα οποία έχουν σταθερή λειτουργικότητα χωρίς δυνατότητα επέκτασης
- **2** → *Περιορισμένη επεκτασιμότητα*: Δυνατότητα βασικής προσαρμογής μέσω configuration files ή scripts, αλλά χωρίς επίσημο API ή integration hooks. *Παράδειγμα*: Εργαλεία τύπου K9s ή Portainer, που επιτρέπουν ρυθμίσεις παραμέτρων αλλά όχι βαθιά ενσωμάτωση με άλλα συστήματα.
- **3** → *Μέτρια επεκτασιμότητα*: Μέτρια επεκτασιμότητα: Υποστηρίζει κάποια επεκτασιμότητα μέσω CRDs, Webhooks ή SDKs, ωστόσο αυτή περιορίζεται κυρίως σε επεκτάσεις σε επίπεδο παραμετροποίησης ή εισόδου δεδομένων και όχι σε λειτουργική επέκταση του εργαλείου (π.χ. νέα modules ή υπηρεσίες). Παράδειγμα: Εργαλεία όπως το Argo CD ή το Falco, όπου απαιτείται εξειδικευμένη γνώση για την ανάπτυξη επεκτάσεων και η διαδικασία είναι πιο περιορισμένη σε σχέση με πλήρως modular αρχιτεκτονικές
- **4** → *Καλή επεκτασιμότητα σε μεγάλα clusters*: Διαθέτει πλήρως τεκμηριωμένα APIs, integration points και υποστήριξη για custom controllers ή operators, επιτρέποντας την προσθήκη νέων λειτουργιών. *Παράδειγμα*: Το Istio ή Rancher, υποστηρίζουν πλήρη επέκταση πολιτικών και ενσωμάτωση με εξωτερικά συστήματα παρακολούθησης
- **5** → *Εξαιρετική, κατάλληλη για πολύ μεγάλα περιβάλλοντα παραγωγής*: Εφαρμόζει modular αρχιτεκτονική, διαθέτει plugin framework, SDKs και native integration με τρίτα συστήματα. Υποστηρίζει ενεργά επεκτάσεις από την κοινότητα. *Παράδειγμα:* Το Grafana, το οποίο διαθέτει πλήρες plugin ecosystem, APIs και SDKs για την ανάπτυξη custom panels, data sources και integrations, αποτελώντας παράδειγμα κορυφαίας επεκτασιμότητας σε cloud-native περιβάλλοντα.

4. Ασφάλεια

Αξιολογεί τον βαθμό στον οποίο το εργαλείο προστατεύει τα δεδομένα, τα workloads και το ίδιο το cluster από επιθέσεις, κακή χρήση ή σφάλματα παραμετροποίησης. Η αξιολόγηση βασίζεται στο πώς το εργαλείο αξιοποιεί ή επεκτείνει τα ενσωματωμένα χαρακτηριστικά ασφαλείας του Kubernetes, όπως το RBAC, οι Network Policies, το mTLS, τα Admission Controllers, και οι μηχανισμοί κρυπτογράφησης. Εξετάζεται επίσης η υποστήριξη auditing, vulnerability scanning και policy enforcement, καθώς και η συμμόρφωση με διεθνή πρότυπα ασφάλειας.

Κλίμακα Αποτίμησης: 1–5 (1 = ελάχιστη προστασία, 5 = κορυφαία ασφάλεια).

Απόδοση Τιμών:

- **1** → *Ελάχιστη ασφάλεια*:. Δεν αξιοποιεί τους μηχανισμούς ασφαλείας του Kubernetes ή τους παρακάμπτει. Απουσία ελέγχων πρόσβασης, πιστοποίησης ή κρυπτογράφησης.
- **2** → *Μέτρια ασφάλεια*: Το εργαλείο προσφέρει μόνο βασική προστασία, π.χ. απλή αυθεντικοποίηση ή χρήση service accounts, χωρίς αξιοποίηση RBAC, Network Policies ή encryption. Δεν υπάρχει συμμόρφωση με πρότυπα ασφαλείας ούτε δυνατότητα auditing.
- **3** → *Ικανοποιητική ασφάλεια*: Αξιοποιεί τις βασικές δυνατότητες του Kubernetes (π.χ. RBAC, Secrets, περιορισμένες Network Policies), αλλά χωρίς ολοκληρωμένη ή αυτοματοποιημένη διαχείριση.

- **4** → *Ισχυρή ασφάλεια*: Υποστηρίζει πλήρως κρυπτογράφηση, πολιτικές πρόσβασης και ελέγχους ταυτότητας. Περιλαμβάνει ενσωμάτωση με συστήματα policy enforcement και audit logging.
- **5** → *Εξαιρετική ασφάλεια*: Παρέχει προηγμένα χαρακτηριστικά, όπως zero-trust networking, mTLS by default, vulnerability scanning, runtime protection και συμβατότητα με πρότυπα συμμόρφωσης και βέλτιστες πρακτικές. Επεκτείνει τους μηχανισμούς ασφαλείας του Kubernetes.

5. Διαλειτουργικότητα

Η διαλειτουργικότητα αξιολογεί τον βαθμό στον οποίο ένα εργαλείο μπορεί να συνεργάζεται ομαλά με άλλα στοιχεία του οικοσυστήματος Kubernetes, καθώς και με εξωτερικές πλατφόρμες (όπως monitoring, logging, CI/CD, ή cloud services). Δεν αφορά τη διαδικασία ενσωμάτωσης, αλλά τη λειτουργική συνεργασία μετά την εγκατάσταση. Στο πλαίσιο αυτό, διαλειτουργικό εργαλείο θεωρείται εκείνο που ανταλλάσσει δεδομένα ή εντολές μέσω κοινών APIs, υποστηρίζει open standards και μπορεί να αξιοποιήσει ή να επεκτείνει τη λειτουργικότητα άλλων εργαλείων χωρίς εκτεταμένη προσαρμογή. Η υψηλή διαλειτουργικότητα μειώνει το κόστος ενοποίησης, αυξάνει τη φορητότητα και διευκολύνει την ολοκληρωμένη διαχείριση.

Κλίμακα Αποτίμησης: 1–4 (1 = περιορισμένη, 4 = άριστη).

Απόδοση Τιμών:

- **1** → *Ελάχιστη διαλειτουργικότητα*: Συνεργάζεται μόνο με το βασικό Kubernetes API και δεν υποστηρίζει επικοινωνία ή ανταλλαγή δεδομένων με άλλα εργαλεία (monitoring, CI/CD, policy).
- **2** → *Περιορισμένη διαλειτουργικότητα*: Υποστηρίζει 1–2 διασυνδέσεις (π.χ. Prometheus ή Grafana) αλλά με περιορισμένη λειτουργικότητα – δηλαδή μπορεί να στέλνει μόνο βασικά δεδομένα, χωρίς πλήρη αξιοποίηση των χαρακτηριστικών των εργαλείων με τα οποία επικοινωνεί.
- **3** → *Μέτρια, διαλειτουργικότητα*: Συνεργάζεται με πολλά κοινά Kubernetes εργαλεία (π.χ. Prometheus, Argo CD, Helm), αλλά απαιτείται χειροκίνητη ρύθμιση, μιας και δεν είναι αυτοματοποιημενη η διασύνδεσή τους.
- **4** → *Υψηλή διαλειτουργικότητα*: Προσφέρει αυτόματη και πλήρως ενοποιημένη συνεργασία με τα περισσότερα υποσυστήματα Kubernetes και εργαλεία CNCF, μέσω standard APIs (CRDs) και πλήρους αμφίδρομης επικοινωνίας.

6. Υποστήριξη & Κοινότητα

Αξιολογεί το εύρος και τη δραστηριότητα της κοινότητας χρηστών και προγραμματιστών που συνεισφέρουν στην ανάπτυξη και υποστήριξη του εργαλείου, καθώς και τη διαθεσιμότητα επίσημης ή εμπορικής υποστήριξης. Το κριτήριο συνδυάζει δύο πτυχές:

- **Κοινοτική δραστηριότητα**, η οποία μπορεί να εκτιμηθεί ποσοτικά μέσω δεικτών, όπως ο αριθμός των contributors, commits, GitHub stars ή συχνότητα ενημερώσεων.
- **Επίσημη υποστήριξη**, που αφορά την ύπαρξη οργανισμών, εταιρειών ή ιδρυμάτων (π.χ. CNCF, Red Hat, SUSE) που συντηρούν, χρηματοδοτούν ή παρέχουν εμπορική τεχνική υποστήριξη.

Η αξιολόγηση εστιάζει στη βιωσιμότητα του εργαλείου μακροπρόθεσμα, δηλαδή στο κατά πόσο υπάρχει ενεργή συντήρηση, τακτικά releases, και κοινότητα που ανταποκρίνεται σε ζητήματα ασφαλείας ή λειτουργίας.

Κλίμακα Αποτίμησης: 1–5 (1 = ελάχιστη, 5 = πολύ ισχυρή). Η κλίμακα 1–5 διατηρείται για λόγους πληρότητας, παρότι στην πράξη οι περισσότερες τιμές εντοπίζονται στις υψηλότερες βαθμίδες, δεδομένου ότι εργαλεία με χαμηλή ή ανύπαρκτη υποστήριξη είχαν ήδη αποκλειστεί κατά τη διαδικασία επιλογής.

Απόδοση Τιμών:

- **1 →** *Σχεδόν ανύπαρκτη κοινότητα/υποστήριξη*: Εργαλείο εγκαταλελειμμένο ή χωρίς ενεργό repository. Ελάχιστα ή καθόλου commits και απουσία επικοινωνίας (π.χ. forums, Slack, GitHub issues).
- **2 →** *Περιορισμένη κοινότητα, χωρίς επίσημη υποστήριξη*: Λίγοι συντηρητές ή contributors (<10), σπάνιες ενημερώσεις, ανεπαρκής ανταπόκριση σε ζητήματα. Δεν παρέχεται επίσημη τεχνική ή εμπορική υποστήριξη.
- **3 →** *Μέτρια κοινότητα και υποστήριξη*: Ενεργό έργο με περιορισμένο αριθμό συνεισφερόντων (10–50), τακτικές ενημερώσεις (μερικές φορές τον χρόνο) και διαθέσιμα κανάλια επικοινωνίας. Η επίσημη υποστήριξη περιορίζεται σε συγκεκριμένη κοινότητα.
- **4 →** *Μεγάλη κοινότητα και καλή υποστήριξη*: Ενεργό project με εκατοντάδες contributors, τακτικά releases και ενεργή συζήτηση στα community κανάλια (Slack, GitHub). Παρέχεται επίσημη υποστήριξη από οργανισμό ή εταιρεία (π.χ. CNCF incubation)
- ***5 →*** *Εξαιρετικά ενεργή κοινότητα, πλήρης τεκμηρίωση, εμπορική υποστήριξη*: Χιλιάδες contributors, συχνά updates, συνεχή security patches, μεγάλα events ή συνεργασίες. Παρέχεται πλήρης επαγγελματική υποστήριξη (π.χ. Red Hat, SUSE) και τεκμηριωμένος μακροπρόθεσμος οδικός χάρτης (roadmap).

Πηγες: (Cloud Native Computing Foundation (CNCF), 2024), (International Organization for Standardization (ISO) & International Electrotechnical Commission (IEC), 2011)

5.2 Σύγκριση ανά Κατηγορία

5.2.1 Monitoring, Observability & Logging

Η αξιολόγηση των τεσσάρων εργαλείων **Prometheus**, **Grafana**, **Loki** και **Jaeger** πραγματοποιήθηκε με βάση τα κριτήρια που ορίστηκαν στην Ενότητα 5.1, χρησιμοποιώντας κλίμακα 1–5 και τεκμηρίωση από τις επιλεγμένες πηγές ([1], [2], [3]).

Απόδοση (Performance)
Το Prometheus βαθμολογήθηκε με 4 καθώς προσφέρει ιδιαίτερα αποδοτική συλλογή μετρικών σε

πραγματικό χρόνο, αν και σε μεγάλης κλίμακας clusters απαιτεί προσεκτική ρύθμιση για να αποφευχθεί υπερφόρτωση [1][2]. Το Grafana επίσης βαθμολογήθηκε με 4, δεδομένου ότι η ταχύτητα απόδοσης των dashboards είναι υψηλή, αλλά εξαρτάται από την απόδοση του backend [2]. Το Loki έλαβε χαμηλότερη βαθμολογία (3), αφού αν και είναι αποδοτικό για συλλογή logs, η ταχύτητα αναζήτησης δεν φτάνει αυτή του Elasticsearch [1][2]. Το Jaeger αξιολογήθηκε με 3, καθώς παρέχει πλήρη υποστήριξη για distributed tracing και άριστη ορατότητα σε πολυεπίπεδες εφαρμογές, ωστόσο η απόδοσή του υποβαθμίζεται σε περιπτώσεις υψηλού φόρτου ή μεγάλου αριθμού traces, λόγω αυξημένης κατανάλωσης πόρων και latency στη συλλογή δεδομένων [2].

Ευκολία Ενσωμάτωσης

Το Prometheus και το Loki βαθμολογήθηκαν με 3 λόγω της ανάγκης για επιπλέον ρυθμίσεις (κανόνες, exporters, log collectors) [1]. Το Grafana έλαβε 4, επειδή η διαδικασία εγκατάστασης είναι εξαιρετικά απλή και γρήγορη — μπορεί να ολοκληρωθεί με ένα Helm chart ή Docker container — ωστόσο απαιτεί βασική παραμετροποίηση για τη σύνδεση πηγών δεδομένων (data sources), κάτι που θεωρείται τυπικό και όχι ιδιαίτερα πολύπλοκο [1]. Το Jaeger έλαβε 3, καθώς, αν και η εγκατάσταση μέσω Operator είναι σχετικά απλή, η πλήρης λειτουργία του απαιτεί πρόσθετη διαμόρφωση backends (π.χ. Elasticsearch, Kafka), κάτι που αυξάνει σημαντικά την πολυπλοκότητα [2].

Επεκτασιμότητα

Το Grafana επιδεικνύει εξαιρετική επεκτασιμότητα και αξιολογήθηκε με 5, καθώς διαθέτει πλήρες plugin framework, SDKs, και API endpoints που επιτρέπουν την ανάπτυξη custom panels, data sources και alerting μηχανισμών. Η modular αρχιτεκτονική του το καθιστά ιδιαίτερα ευέλικτο, με χιλιάδες community plugins και integrations (π.χ. Prometheus, Loki, InfluxDB, Elasticsearch, Datadog).

Το Prometheus αξιολογήθηκε με 4 αφού διαθέτει καλή επεκτασιμότητα, με πλήρως τεκμηριωμένα APIs και integration points για exporters, alerting rules και remote storage backends. Παρέχει υποστήριξη για custom exporters και Alertmanager integrations, επιτρέποντας την ενσωμάτωση σε πλήθος περιβαλλόντων. Ωστόσο, η αρχιτεκτονική του δεν είναι πλήρως modular, καθώς η προσθήκη νέων λειτουργιών απαιτεί συχνά ξεχωριστά components και όχι εσωτερική επέκταση.

Το Loki ακολουθεί σχεδιαστικά το Prometheus και υποστηρίζει modular επεκτασιμότητα μέσω APIs, Grafana plugins, και custom log pipelines. Μπορεί να ενσωματωθεί με πολλαπλά backends και να προσαρμοστεί σε custom deployments μέσω Helm values και configuration files. Παρότι υποστηρίζει integrations με άλλα observability εργαλεία (π.χ. Tempo, Promtail), η προσθήκη νέων parsing modules είναι πιο περιορισμένη, γι' αυτό αξιολογείται με 4.

Το Jaeger αξιολογήθηκε με 4 διότι παρέχει πλούσια API και plugin-based αρχιτεκτονική, υποστηρίζοντας επεκτάσεις μέσω storage plugins, sampling strategies, και OpenTelemetry collectors. Μπορεί να ενσωματωθεί με πλήθος tracing backends (Elasticsearch, Cassandra, Kafka) και προσφέρει integration hooks για CI/CD pipelines. Ωστόσο, σε αντίθεση με εργαλεία όπως το Grafana, δεν διαθέτει πλήρως ανεπτυγμένα SDKs ή native developer interfaces για τη δημιουργία νέων modules και panels, γεγονός που περιορίζει τον βαθμό επεκτασιμότητας σε προχωρημένο επίπεδο — εξ ου και η τελική αποτίμηση με 4.

Ασφάλεια

Κανένα από τα εργαλεία δεν ξεπέρασε τη βαθμολογία 2, καθώς όλα περιορίζονται σε βασικούς μηχανισμούς ασφάλειας, όπως RBAC και TLS, χωρίς ολοκληρωμένη διαχείριση πολιτικών ή αυτοματοποιημένο auditing. Εργαλεία όπως το Grafana και το Loki υποστηρίζουν βασική αυθεντικοποίηση χρηστών και χρήση service accounts, ωστόσο η πλήρης ασφάλεια απαιτεί πρόσθετη ενσωμάτωση με εξωτερικά συστήματα, όπως OAuth2, LDAP ή OPA. Συνεπώς, η προστασία που παρέχουν θεωρείται επαρκής μόνο για περιβάλλοντα με χαμηλές απαιτήσεις ασφαλείας [1][2].

Διαλειτουργικότητα

Το Prometheus αξιολογήθηκε με 3, καθώς συνεργάζεται αποτελεσματικά με το Kubernetes API, το Grafana και εργαλεία, όπως το Loki και το KEDA, ωστόσο η ενσωμάτωση βασίζεται κυρίως σε χειροκίνητη ρύθμιση exporters και targets, χωρίς πλήρη αυτοματοποίηση ή αμφίδρομη επικοινωνία. Το Grafana έλαβε 4, χάρη στην εκτενή υποστήριξη data sources (π.χ. Prometheus, Loki, Jaeger, Elasticsearch, OpenTelemetry) και τη δυνατότητα αμφίδρομης συνεργασίας μέσω APIs, plugins, webhooks και Operators, επιτυγχάνοντας υψηλή ενοποίηση με CI/CD pipelines και GitOps workflows. Το Loki βαθμολογήθηκε επίσης με 3, καθώς, αν και ενσωματώνεται πλήρως με το Grafana και συνεργάζεται με Fluentd, Promtail και Tempo, απαιτεί πρόσθετη παραμετροποίηση για τη σύνδεση με άλλα συστήματα παρακολούθησης ή καταγραφής. Τέλος, το Jaeger έλαβε 4, λόγω της εκτενούς υποστήριξης για OpenTelemetry, Istio, Envoy και Prometheus, καθώς και της δυνατότητας λειτουργίας μέσω Jaeger Operator, που επιτρέπει εγγενή διασύνδεση με το Kubernetes control plane. Η συνεργασία του με τα παραπάνω εργαλεία είναι κυρίως αμφίδρομη σε επίπεδο δεδομένων παρακολούθησης (μέσω APIs και OTLP endpoints), καθώς το Jaeger μπορεί να λαμβάνει και να εκθέτει δεδομένα observability προς άλλα συστήματα, αν και η ενοποίηση δεν είναι πλήρως συμμετρική ή αυτοματοποιημένη..

Υποστήριξη & Κοινότητα

Το Prometheus και το Grafana αξιολογήθηκαν με 5, καθώς αποτελούν ώριμα έργα του CNCF, με πολύ μεγάλη κοινότητα, ενεργή ανάπτυξη και εξαιρετική επεκτασιμότητα. Και τα δύο διαθέτουν modular αρχιτεκτονική, εκτενές plugin framework, API υποστήριξη και SDKs, επιτρέποντας σε προγραμματιστές και οργανισμούς να αναπτύσσουν custom integrations, dashboards και exporters. Η ευρεία υιοθέτηση και η συνεχιζόμενη συνεισφορά από την κοινότητα ενισχύουν περαιτέρω τη σταθερότητα και τη διαλειτουργικότητά τους σε μεγάλα παραγωγικά περιβάλλοντα [3].Το Jaeger επίσης βαθμολογήθηκε με 5, δεδομένου ότι, πέρα από την ενεργή κοινότητα, διαθέτει και επίσημη υποστήριξη μέσω της Red Hat και της CNCF (Graduated project), με τακτικά releases και μακροπρόθεσμο roadmap. Το Loki επίσης έλαβε 4, καθώς διατηρεί ενεργή κοινότητα, συχνές ενημερώσεις και στενή ενσωμάτωση με το Grafana μέσω shared APIs και dashboarding. Αν και βρίσκεται σε χαμηλότερο επίπεδο ωριμότητας σε σχέση με τα κορυφαία CNCF projects, η ανοικτή αρχιτεκτονική και η συνεχής ανάπτυξή του εξασφαλίζουν καλή υποστήριξη και βιωσιμότητα μακροπρόθεσμα [3].

Πίνακας 20: Συγκριτική Αξιολόγηση Εργαλείων Monitoring

	Εργαλείο

Κριτήριο	Prometheus	Grafana	Loki	Jaeger
Απόδοση (Performance)	**4**	**4**	**3**	**3**
Ευκολία Ενσωμάτωσης	**3**	**4**	**3**	**3**
Επεκτασιμότητα	**4**	**5**	**4**	**4**
Ασφάλεια	**2**	**2**	**2**	**2**
Διαλειτουργικότητα	**3**	**4**	**3**	**4**
Υποστήριξη & Κοινότητα	**5**	**5**	**4**	**5**

Πηγές

[1] Hämäläinen και συν. (2021).
[2] Pai και συν. (2024).
[3] Hadikusuma, Lukas, & Bachri (2022).

Η κατηγορία Monitoring περιλαμβάνει τα εργαλεία Prometheus, Grafana, Loki και Jaeger, τα οποία αποτελούν τον κορμό του οικοσυστήματος παρακολούθησης και παρατηρησιμότητας (observability) του Kubernetes. Από την ανάλυση προκύπτει ότι το Grafana αναδεικνύεται ως το πιο ευέλικτο και επεκτάσιμο εργαλείο, με εξαιρετική υποστήριξη plugins, APIs και SDKs, που του επιτρέπουν να ενσωματώνεται σχεδόν με κάθε σύστημα παρακολούθησης. Αντίστοιχα, το Prometheus διακρίνεται για την υψηλή απόδοση του στη συλλογή μετρικών αλλά η επεκτασιμότητα και η διαλειτουργικότητά του περιορίζονται από την ανάγκη χειροκίνητης ρύθμισης exporters. Το Loki, αν και αποδοτικό στην καταγραφή logs, υπολείπεται σε ταχύτητα αναζήτησης και απαιτεί προσεκτική κλιμάκωση σε μεγάλα clusters, γεγονός που το τοποθετεί χαμηλότερα (βαθμός 3 στην απόδοση). Το Jaeger, τέλος, προσφέρει προηγμένες δυνατότητες tracing και ενσωμάτωσης με OpenTelemetry, αλλά εμφανίζει αυξημένο latency και κατανάλωση πόρων υπό βαρύ φόρτο, κάτι που περιορίζει τη συνολική του αποδοτικότητα.

Σε επίπεδο ασφάλειας, κανένα εργαλείο δεν υπερέχει σημαντικά, καθώς βασίζονται κυρίως σε μηχανισμούς όπως RBAC και TLS χωρίς πλήρη αυτοματοποιημένο auditing ή zero-trust αρχιτεκτονική. Ωστόσο, όλα επωφελούνται από την ασφάλεια που προσφέρει η ίδια η Kubernetes πλατφόρμα.

Όσον αφορά την ευκολία ενσωμάτωσης, το Grafana ξεχωρίζει με τη φιλική διαδικασία εγκατάστασης και τεκμηρίωσης (βαθμός 4), ενώ το Prometheus και το Loki απαιτούν πιο σύνθετη παραμετροποίηση.

Στο πεδίο της διαλειτουργικότητας, το Grafana και το Jaeger υπερέχουν (βαθμός 4–5), υποστηρίζοντας πλήρη συνεργασία με εργαλεία όπως Prometheus, Loki, OpenTelemetry και Istio. Το Prometheus και το Loki, αντίθετα, έχουν καλή αλλά πιο περιορισμένη διαλειτουργικότητα καθώς απαιτούν χειροκίνητη διασύνδεση μέσω exporters ή collectors.

Τέλος, όλα τα εργαλεία διαθέτουν πολύ ενεργές κοινότητες και επίσημη υποστήριξη μέσω του CNCF, με το Prometheus, το Grafana και το Jaeger να κατατάσσονται υψηλότερα (βαθμός 5), λόγω του ώριμου οικοσυστήματος, της τεκμηρίωσης και της επαγγελματικής υποστήριξης.

Λαμβάνοντας υπόψη έναν συνδυασμό βασικών κριτηρίων – συγκεκριμένα την Απόδοση, την Επεκτασιμότητα και την Υποστήριξη & Κοινότητα – το Grafana αναδεικνύεται ως το πλέον ολοκληρωμένο και ώριμο εργαλείο της ομάδας. Συνδυάζει υψηλή απόδοση με εξαιρετική επεκτασιμότητα (plugin framework, SDKs, APIs) και ισχυρή κοινότητα υποστήριξης, γεγονός που το καθιστά την πιο ευέλικτη επιλογή για περιβάλλοντα παραγωγής.

Αντίθετα, το Loki καταγράφει τη χαμηλότερη συνολική επίδοση, κυρίως λόγω περιορισμένης απόδοσης και αυξημένων απαιτήσεων παραμετροποίησης, παρά την καλή του διαλειτουργικότητα. Παρ' όλα αυτά, ο συνδυασμός των τεσσάρων εργαλείων (Prometheus, Grafana, Loki, Jaeger) συνεχίζει να αποτελεί ένα ισορροπημένο οικοσύστημα παρακολούθησης και ανάλυσης, όπου κάθε εργαλείο συμβάλλει συμπληρωματικά στη συνολική ορατότητα και αξιοπιστία του Kubernetes περιβάλλοντος.

Συνολικά, η υψηλότερη επίδοση καταγράφεται στο κριτήριο Υποστήριξη & Κοινότητα (4–5), ενώ η χαμηλότερη στην Ασφάλεια (2). Τα υπόλοιπα κριτήρια κυμαίνονται μεταξύ 3–4, γεγονός που αποτυπώνει μια ισορροπία μεταξύ λειτουργικότητας και επεκτασιμότητας στα εργαλεία παρακολούθησης του οικοσυστήματος Kubernetes.

5.2.2 Security

Η σύγκριση των εργαλείων **OPA Gatekeeper**, **Vault**, **Falco** και **Trivy** ανέδειξε σημαντικές διαφοροποιήσεις ως προς την απόδοση, την ευκολία ενσωμάτωσης, την επεκτασιμότητα, την ασφάλεια, τη διαλειτουργικότητα και την υποστήριξη/κοινότητα.

Απόδοση (Performance)

Η απόδοση των εργαλείων παρουσιάζει διακυμάνσεις ανάλογα με τη φύση των λειτουργιών τους.

- **Vault** και **Trivy** πέτυχαν υψηλή βαθμολογία 4 λόγω της ταχύτητας εκτέλεσης βασικών λειτουργιών (ανάκτηση μυστικών και σάρωση εικόνων αντίστοιχα) σε τυπικές συνθήκες [3] [7]. Ωστόσο, η απόδοσή τους μπορεί να επηρεαστεί αρνητικά σε περιβάλλοντα πολύ μεγάλου όγκου δεδομένων.
- **OPA Gatekeeper** και **Falco** βαθμολογήθηκαν χαμηλότερα 3 λόγω καθώς, παρότι αποδίδουν ικανοποιητικά σε τυπικά ή μεσαίας κλίμακας workloads, παρουσιάζουν αισθητή πτώση απόδοσης και αυξημένο latency σε μεγάλα ή πολύπλοκα σενάρια, όπου εμπλέκονται εκτεταμένες πολιτικές ή πολλοί κανόνες παρακολούθησης. [1] [5].

Ευκολία Ενσωμάτωσης

Το OPA Gatekeeper και το Vault έλαβαν βαθμολογία 2, καθώς, αν και η εγκατάστασή τους είναι τεχνικά εφικτή μέσω Helm charts ή Operators, η πλήρης λειτουργία τους απαιτεί πολυεπίπεδη ρύθμιση και εξειδικευμένη γνώση του Kubernetes.

Το OPA Gatekeeper χρειάζεται προσεκτική παραμετροποίηση των CRDs, πολιτικών (constraints) και Rego templates, ενώ το Vault απαιτεί ρύθμιση μηχανισμών authentication, backend storage και RBAC πολιτικών. Αυτές οι πρόσθετες χειροκίνητες παρεμβάσεις και η εμπειρία στο Kubernetes αντιστοιχούν στο επίπεδο "Δύσκολη ενσωμάτωση (2)", καθώς το εργαλείο δεν διαθέτει πλήρη αυτοματοποίηση της διαδικασίας.

Αντίθετα, το Falco αξιολογήθηκε με 3, καθώς η εγκατάστασή του μέσω Helm chart ή DaemonSet είναι σχετικά απλή, αλλά απαιτεί παραμετροποίηση των κανόνων παρακολούθησης (rulesets) και των alerting μηχανισμών. Το εργαλείο διαθέτει πλήρη τεκμηρίωση, γεγονός που καθιστά την ενσωμάτωση εφικτή αλλά όχι άμεσα αυτοματοποιημένη, σύμφωνα με το επίπεδο "Μέτρια ευκολία ενσωμάτωσης (3)".

Τέλος, το Trivy έλαβε βαθμολογία 4, καθώς προσφέρει πλήρως αυτοματοποιημένους μηχανισμούς εγκατάστασης (Helm chart, CLI, GitHub Action) και ενσωματώνεται εύκολα σε CI/CD pipelines χωρίς να απαιτείται πρόσθετη παραμετροποίηση. Η βασική του λειτουργία είναι διαθέσιμη άμεσα μετά την εγκατάσταση, ωστόσο η προσαρμογή σε πιο σύνθετα σενάρια απαιτεί παραμετροποίηση, αν και οι παράμετροι προς χρήση είναι σαφείς.

Επεκτασιμότητα

Το Vault αξιολογήθηκε με 5 μιας και διαθέτει modular αρχιτεκτονική με plugin framework, HTTP API, SDKs και integration hooks για πλήθος εξωτερικών συστημάτων (cloud providers, databases, identity systems). Υποστηρίζει dynamic secrets engines, custom plugins, και μπορεί να επεκταθεί μέσω community modules. Διαθέτει επίσης ευρεία υποστήριξη από HashiCorp και ενεργή κοινότητα που αναπτύσσει integrations.

Το Gatekeeper αξιολογήθηκε με 4 αφού υποστηρίζει επεκτασιμότητα μέσω Custom Resource Definitions (CRDs) και Constraint Templates, που επιτρέπουν τον ορισμό και την ανάπτυξη νέων πολιτικών χωρίς αλλαγή στον πυρήνα του εργαλείου. Μπορεί να επεκταθεί με custom admission webhooks και OPA Rego policies, ενώ διαθέτει καλά τεκμηριωμένα APIs και integrations με CI/CD ή security pipelines. Ωστόσο, σε αντίθεση με εργαλεία που διαθέτουν πλήρως modular αρχιτεκτονική ή plugin framework (επίπεδο 5), το Gatekeeper δεν προσφέρει επίσημο SDK ή μηχανισμό ανάπτυξης τρίτων επεκτάσεων..

Το Falco αξιολογήθηκε με 3 δεδομένου ότι επιτρέπει την προσθήκη custom κανόνων (rules) και plugins για την παρακολούθηση γεγονότων συστήματος. Ωστόσο η ανάπτυξη νέων modules απαιτεί εξειδικευμένη γνώση C ή eBPF. Παρότι υποστηρίζει Falco plugins SDK και gRPC outputs, η επεκτασιμότητα παραμένει πιο περιορισμένη σε σχέση με το Vault ή το Gatekeeper, αφού αφορά κυρίως την προσθήκη νέων πηγών εισόδου ή κανόνων ανίχνευσης, όχι νέες λειτουργικές κατηγορίες.

Το Trivy αξιολογήθηκε με 3 στην επεκτασιμότητα, καθώς υποστηρίζει επεκτάσεις μέσω custom vulnerability databases, plugins και integration APIs (π.χ. GitHub, GitLab, Jenkins, Argo CD), προσφέροντας ευελιξία στη διασύνδεση με εξωτερικές πλατφόρμες. Ωστόσο, η επεκτασιμότητα αυτή περιορίζεται κυρίως σε εξωτερικό επίπεδο, καθώς το εργαλείο δεν διαθέτει modular αρχιτεκτονική ή πλήρες SDK για την ανάπτυξη νέων λειτουργικών components (π.χ. scanners ή

policy engines). Επομένως, η προσθήκη νέων δυνατοτήτων απαιτεί εξωτερική παραμετροποίηση ή scripting, χωρίς να επεκτείνεται ουσιαστικά η εσωτερική λειτουργικότητα του πυρήνα του εργαλείου.

Ασφάλεια

Το Vault έλαβε βαθμολογία 5, καθώς προσφέρει προηγμένα χαρακτηριστικά ασφάλειας που επεκτείνουν τις εγγενείς δυνατότητες του Kubernetes. Περιλαμβάνει πλήρη κρυπτογράφηση δεδομένων εντός και εκτός cluster, dynamic secrets, policy-based access control και integration με enterprise authentication providers (LDAP, OIDC). Επίσης, υποστηρίζει auditing, rotation και zero-trust αρχιτεκτονική — στοιχεία που αντιστοιχούν στο επίπεδο "Εξαιρετική, πλήρης συμμόρφωση με βέλτιστες πρακτικές (5)".

Το OPA Gatekeeper και το Falco βαθμολογήθηκαν με 4, καθώς παρέχουν ισχυρούς μηχανισμούς ασφάλειας, αλλά με εξάρτηση από τη σωστή διαμόρφωση πολιτικών ή κανόνων. Ο OPA Gatekeeper επιβάλλει πολιτικές ασφάλειας μέσω OPA/Rego rules και ενσωματώνεται πλήρως με το Kubernetes Admission Controller, υποστηρίζοντας audit mode και compliance checks — επομένως εφαρμόζει πλήρως RBAC, policy enforcement και logging. Αντίστοιχα, το Falco προσφέρει runtime protection, ανίχνευση παραβιάσεων και audit logging, ωστόσο η αποτελεσματικότητα του εξαρτάται από την πληρότητα και τη συντήρηση των κανόνων (rulesets).

Το Trivy έλαβε βαθμολογία 3, καθώς παρέχει ικανοποιητική αλλά όχι πλήρη ασφάλεια. Ανιχνεύει ευπάθειες σε images, repositories και Kubernetes manifests, αλλά λειτουργεί εκτός runtime και εξαρτάται από την επικαιρότητα της vulnerability database. Δεν περιλαμβάνει μηχανισμούς active enforcement, policy-based access control ή encryption.

Διαλειτουργικότητα

Το Vault έλαβε τη μέγιστη βαθμολογία 4, καθώς διαθέτει υψηλή διαλειτουργικότητα και πλήρη υποστήριξη για native integration με το Kubernetes (μέσω Kubernetes Auth Method και CSI Driver), καθώς και με εξωτερικά συστήματα, όπως CI/CD pipelines (GitLab, Jenkins) και cloud providers (AWS, GCP, Azure). Η ύπαρξη πλήρως τεκμηριωμένων REST APIs και η δυνατότητα αμφίδρομης επικοινωνίας με άλλα εργαλεία ασφαλείας και διαχείρισης μυστικών ενισχύει περαιτέρω τον βαθμό ενοποίησής του.

Το Falco βαθμολογήθηκε με 3, παρουσιάζοντας μέτρια διαλειτουργικότητα. Αν και υποστηρίζει συνεργασία με εργαλεία παρακολούθησης και καταγραφής, όπως Prometheus, Grafana, Elasticsearch και Loki, οι περισσότερες ενσωματώσεις απαιτούν πρόσθετα components (όπως Falcosidekick ή Falco Exporter) και χειροκίνητη ρύθμιση. Επομένως, ενώ καλύπτει ευρύ φάσμα διασυνδέσεων, η αυτοματοποίηση αυτών των συνεργασιών παραμένει περιορισμένη.

Αντίστοιχα, το Trivy έλαβε επίσης βαθμολογία 3, καθώς υποστηρίζει διαλειτουργία με CI/CD pipelines (GitHub Actions, Argo CD), συστήματα πολιτικών (OPA, Kyverno) και monitoring πλατφόρμες, προσφέροντας εξαγωγή δεδομένων ευπαθειών σε μορφές όπως JSON ή SARIF. Παρόλα αυτά, οι περισσότερες διασυνδέσεις είναι μονόδρομες — δηλαδή το Trivy αποστέλλει

δεδομένα χωρίς να λαμβάνει ανατροφοδότηση από τα συστήματα αυτά — και απαιτούν μερική παραμετροποίηση από τον χρήστη, γεγονός που περιορίζει το επίπεδο αυτοματοποίησης

Τέλος, το OPA Gatekeeper βαθμολογήθηκε με 2, καθώς παρουσιάζει περιορισμένη διαλειτουργικότητα. Παρότι ενσωματώνεται πλήρως με το Kubernetes API για την επιβολή πολιτικών admission control, δεν παρέχει αυτόματη διασύνδεση με εξωτερικά εργαλεία monitoring, CI/CD ή policy management. Η συνεργασία με άλλα συστήματα είναι εφικτή μόνο μέσω custom webhooks ή scripts, γεγονός που απαιτεί χειροκίνητες παρεμβάσεις και αυξάνει τη λειτουργική πολυπλοκότητα.

Υποστήριξη & Κοινότητα

Το Vault έλαβε τη μέγιστη βαθμολογία 5, καθώς διαθέτει εξαιρετικά ενεργή κοινότητα, χιλιάδες contributors και πλήρη εμπορική υποστήριξη από τη HashiCorp, με συχνά releases,security patches και δημοσιευμένο οδικό χάρτη ανάπτυξης (roadmap) που καθορίζει τις μελλοντικές βελτιώσεις και νέες λειτουργίες.

Το Falco έλαβε 4, καθώς, αν και αποτελεί ώριμο έργο του CNCF με εκατοντάδες contributors και ενεργό ρυθμό ανάπτυξης, δεν συνοδεύεται από εμπορική υποστήριξη ή επίσημα enterprise SLAs, που απαιτούνται για να θεωρηθεί επιπέδου 5.

Το Trivy αξιολογήθηκε με 4, καθώς αποτελεί έργο ανοιχτού κώδικα της Aqua Security με πολύ ενεργή ανάπτυξη, τακτικά releases και συνεχή ενημέρωση της βάσης ευπαθειών (CVE database). Διαθέτει πλήρη τεκμηρίωση, ενσωμάτωση σε CI/CD pipelines (GitHub Actions, GitLab CI, Jenkins) και επίσημο Helm chart για Kubernetes, γεγονός που εξασφαλίζει συνεχή υποστήριξη και πρακτική επεκτασιμότητα. Παρότι δεν προσφέρει πλήρη εμπορική υποστήριξη με SLA, η συντήρηση από την Aqua Security και η ενεργή κοινότητα συνεισφερόντων το καθιστούν ιδιαίτερα αξιόπιστο και ώριμο εργαλείο, δικαιολογώντας την αξιολόγηση στο επίπεδο “Καλή υποστήριξη (4)”.

Τέλος Το OPA Gatekeeper αξιολογήθηκε με 4, καθώς αποτελεί επίσημο έργο του CNCF με ενεργή ανάπτυξη στο GitHub και συνεχή συνεισφορά από την κοινότητα και την ομάδα του Open Policy Agent (OPA). Διαθέτει τακτικά releases, αναλυτική τεκμηρίωση, ενεργά κανάλια υποστήριξης (Slack, GitHub discussions) και ενσωμάτωση με Kubernetes Operators, γεγονός που ενισχύει τη σταθερότητα και τη μακροχρόνια βιωσιμότητά του. Παρότι η υποστήριξη παραμένει κυρίως κοινοτική, το επίπεδο οργάνωσης και η ωριμότητα του έργου δικαιολογούν την αξιολόγηση στο επίπεδο «Καλή υποστήριξη (4)».

Πίνακας 21: Συγκριτική Αξιολόγηση Εργαλείων Security

	Εργαλείο			
Κριτήριο	**OPA Gatekeeper**	**Vault**	**Falco**	**Trivy**
Απόδοση (Performance)	**3**	**4**	**3**	**4**
Ευκολία Ενσωμάτωσης	**2**	**2**	**3**	**4**

Επεκτασιμότητα	**4**	**5**	**3**	**3**
Ασφάλεια	**4**	**5**	**4**	**3**
Διαλειτουργικότητα	**2**	**4**	**3**	**3**
Υποστήριξη & Κοινότητα	**4**	**5**	**4**	**4**

Πηγές

[1] (Jakkaraju, 2020)

[2] (Morić, Dakić, & Čavala, 2025)

[3] (Cloud Native Computing Foundation (CNCF), 2024)

[4] (Amgothu & Kankanala, 2024)

[5] (HashiCorp, 2025)

[6] (Chinamanagonda, 2021)

[7] (Aqua Security, 2024)

Η κατηγορία Security & Policy Management επικεντρώνεται στη διασφάλιση της συμμόρφωσης, της εμπιστευτικότητας και της παρακολούθησης ασφάλειας στο οικοσύστημα του Kubernetes. Τα εργαλεία που αξιολογήθηκαν (OPA Gatekeeper, Vault, Falco, Trivy) αντιπροσωπεύουν διαφορετικά επίπεδα προστασίας και ελέγχου, καλύπτοντας τόσο προληπτικούς όσο και ανιχνευτικούς μηχανισμούς ασφάλειας.

Η συνολική εικόνα δείχνει ότι το Vault υπερέχει ως η πιο ολοκληρωμένη και ώριμη λύση, παρέχοντας υψηλό επίπεδο ασφάλειας, ευελιξία και διαλειτουργικότητα. Η modular αρχιτεκτονική του και η δυνατότητα ενσωμάτωσης με πλήθος εξωτερικών συστημάτων (cloud providers, CI/CD, identity management) το καθιστούν το πλέον κατάλληλο εργαλείο για οργανισμούς με σύνθετες απαιτήσεις προστασίας δεδομένων και compliance.

Το OPA Gatekeeper αποτελεί κρίσιμο εργαλείο για την επιβολή πολιτικών συμμόρφωσης (policy enforcement) στο Kubernetes, προσφέροντας λεπτομερή έλεγχο σε επίπεδο admission control. Ωστόσο, η αξία του συνοδεύεται από αυξημένη πολυπλοκότητα, καθώς η δημιουργία και συντήρηση Rego πολιτικών απαιτεί υψηλή εξειδίκευση. Επομένως, αποτελεί ιδανική επιλογή για περιβάλλοντα όπου η ακρίβεια των πολιτικών υπερισχύει της ευκολίας χρήσης.

Αντίθετα, το Falco και το Trivy καλύπτουν το πρακτικό σκέλος της ασφάλειας. Το Falco παρέχει προστασία σε runtime επίπεδο, εντοπίζοντας αποκλίσεις ή ύποπτες συμπεριφορές σε πραγματικό χρόνο, κάτι που το καθιστά εξαιρετικά χρήσιμο για ανίχνευση απειλών σε παραγωγικά clusters. Το Trivy, από την άλλη, εστιάζει στην πρόληψη μέσω της αυτόματης σάρωσης ευπαθειών σε εικόνες, αποθετήρια και manifests, λειτουργώντας ως ελαφριά αλλά αποδοτική λύση για DevSecOps pipelines.

Σε συνολική θεώρηση, η κατηγορία παρουσιάζει ισχυρή τεχνολογική ωριμότητα, με κάθε εργαλείο να εξειδικεύεται σε διαφορετικό στάδιο του κύκλου ζωής της ασφάλειας. Το Vault ξεχωρίζει ως η πληρέστερη και πιο επεκτάσιμη λύση, συνδυάζοντας λειτουργικότητα, αυτοματοποίηση και ενεργή υποστήριξη κοινότητας. Αντίθετα, το OPA Gatekeeper προσφέρει το υψηλότερο επίπεδο ελέγχου πολιτικών, αλλά σε βάρος της ευκολίας ενσωμάτωσης, ενώ το Trivy και το Falco αποτελούν πιο ευέλικτες και ελαφριές επιλογές, κατάλληλες για οργανισμούς που δίνουν προτεραιότητα στην απλότητα και την ταχύτητα ανάπτυξης.

Συνολικά, η κατηγορία αναδεικνύει ότι δεν υπάρχει ένα εργαλείο που υπερτερεί σε όλα τα κριτήρια, αλλά περιλαμβάνει μάλλον συμπληρωματικές λύσεις που, σε συνδυασμό, μπορούν να προσφέρουν ένα πλήρες πλαίσιο ασφάλειας και συμμόρφωσης στο Kubernetes. Ως προς τα επιμέρους κριτήρια, η Επεκτασιμότητα και η Υποστήριξη/Κοινότητα παρουσιάζουν τις υψηλότερες επιδόσεις, γεγονός που αντικατοπτρίζει τη συνεχή ανάπτυξη και ενεργό συνεισφορά της κοινότητας στα εργαλεία ανοιχτού κώδικα. Αντίθετα, η Απόδοση, η Ευκολία Ενσωμάτωσης και η Διαλειτουργικότητα καταγράφουν χαμηλότερες τιμές, κυρίως λόγω των αυξημένων απαιτήσεων πόρων και της πολυπλοκότητας που συνεπάγεται η λειτουργία σε κατανεμημένα περιβάλλοντα Kubernetes.

5.2.3 Networking

Η κατηγορία Networking περιλαμβάνει εργαλεία που εξυπηρετούν την επικοινωνία, την ασφάλεια και τη διαχείριση του δικτύου εντός Kubernetes clusters. Η αξιολόγηση βασίστηκε στα έξι κριτήρια της Ενότητας 5.1, με κλίμακα 1–5, και στις επιστημονικές πηγές [1, 3–5] και το CNCF Landscape [2].

Απόδοση (Performance)

Το Cilium εμφανίζει ιδιαίτερα υψηλή απόδοση χάρη στη χρήση του eBPF, που εκτελεί λειτουργίες ασφάλειας και δρομολόγησης στο επίπεδο του πυρήνα (kernel), μειώνοντας το latency και το CPU overhead. Ωστόσο, σε πολύ μεγάλα clusters ή σενάρια με πολύπλοκα NetworkPolicies, έχουν αναφερθεί μικρές καθυστερήσεις στη συλλογή metrics και στη ροή πακέτων, οι οποίες τοποθετούν τη βαθμολογία στο 4 και όχι στο 5, καθώς δεν επιτυγχάνει πλήρως “zero-overhead” εκτέλεση σε κάθε περιβάλλον.

Το Calico επιτυγχάνει σταθερή και αξιόπιστη δικτυακή απόδοση μέσω native Linux routing χωρίς επιπλέον abstraction layers, γεγονός που το καθιστά αποδοτικό ακόμη και σε μεγάλα clusters. Παρ’ όλα αυτά, η έλλειψη δυνατοτήτων eBPF-level optimization (όπως στο Cilium) οδηγεί σε ελαφρώς υψηλότερο latency υπό βαρύ φόρτο, οπότε λαμβάνει επίσης βαθμό 4.

Το CoreDNS είναι πολύ αποδοτικό στη διαχείριση DNS queries, με caching μηχανισμό και χαμηλό χρόνο απόκρισης. Ωστόσο, η απόδοση μπορεί να επηρεαστεί σε περιπτώσεις με εκατοντάδες zones ή custom plugins, όπου παρατηρείται αυξημένο latency λόγω επέκτασης της αλυσίδας plugins. Έτσι, κατατάσσεται στο 4, καθώς διατηρεί υψηλή απόδοση αλλά όχι πλήρως βελτιστοποιημένη σε κάθε συνθήκη.

Ευκολία Ενσωμάτωσης

Το CoreDNS συγκέντρωσε τη μεγαλύτερη βαθμολογία (4), καθώς αποτελεί τον προεπιλεγμένο DNS provider στο Kubernetes και ενσωματώνεται σχεδόν εγγενώς στο cluster. Η εγκατάσταση του είναι ιδιαίτερα απλή, με χρήση Helm chart ή βασικών manifests, ενώ μόνο σε πιο σύνθετα σενάρια (π.χ. custom plugins ή εξωτερικά DNS integrations) απαιτούνται πρόσθετες ρυθμίσεις.

Το Cilium και το Calico βαθμολογήθηκαν με (3), αφού, παρότι διαθέτουν μηχανισμούς εγκατάστασης μέσω Helm charts ή CLI, προϋποθέτουν εξειδικευμένη γνώση για τη σωστή παραμετροποίηση. Στην περίπτωση του Cilium, απαιτείται ενεργοποίηση του eBPF και προσαρμογή του CNI plugin στο υπάρχον περιβάλλον [1][3], ενώ το Calico χρειάζεται χειροκίνητη ρύθμιση παραμέτρων, όπως IP pools ή BGP configurations.

Επεκτασιμότητα

Το Cilium συγκέντρωσε τη μέγιστη βαθμολογία (5), καθώς βασίζεται σε modular αρχιτεκτονική με πυρήνα το eBPF, το οποίο επιτρέπει την προσθήκη νέων λειτουργιών στο επίπεδο του kernel χωρίς επανεκκίνηση ή τροποποίηση του cluster. Υποστηρίζει πλήρως custom extensions, SDKs, καθώς και integration hooks με άλλα συστήματα (π.χ. Hubble, Prometheus, Envoy) [1]. Η ενεργή κοινότητα συντηρεί επιπλέον επεκτάσεις (Cilium Hub) που ενισχύουν τη λειτουργικότητα του εργαλείου.

Το Calico αξιολογήθηκε με (4), καθώς προσφέρει εκτεταμένες δυνατότητες παραμετροποίησης μέσω CRDs και policy controllers, επιτρέποντας την επέκταση πολιτικών ασφάλειας και routing. Διαθέτει πλήρως τεκμηριωμένα APIs και υποστηρίζει ενσωμάτωση με third-party controllers και service meshes (π.χ. Istio). Ωστόσο, η αρχιτεκτονική του είναι λιγότερο modular σε σχέση με το Cilium, καθώς βασίζεται σε στατικό data plane (IPTables ή eBPF backend) και δεν διαθέτει πλήρες plugin framework [3].

Το CoreDNS αξιολογήθηκε με βαθμό 4 ως προς την επεκτασιμότητα, καθώς διαθέτει modular αρχιτεκτονική που επιτρέπει την προσθήκη νέων λειτουργιών μέσω plugins αλλά απουσιάζει επίσημο SDK και πλήρες third-party plugin ecosystem, γεγονός που περιορίζει την επεκτασιμότητα σε σχέση με εργαλεία επιπέδου 5. Όμως, υποστηρίζει πλήρως τεκμηριωμένα APIs και δυνατότητα ανάπτυξης custom plugins, διευκολύνοντας την προσαρμογή του σε διαφορετικά σενάρια χρήσης (π.χ. DNS forwarding, caching, service discovery). Επιπλέον, προσφέρει ενσωματωμένα plugins όπως health checking, metrics collection και internal query logging — όπου το logging αφορά την καταγραφή DNS queries και responses για σκοπούς debugging.

Ασφάλεια

Το Cilium έλαβε τη μέγιστη βαθμολογία (5) καθώς παρέχει εξαιρετική ασφάλεια, πλήρως εναρμονισμένη με τις βέλτιστες πρακτικές του Kubernetes. Υποστηρίζει mTLS (mutual TLS) by default, identity-aware policies βασισμένες σε labels και namespaces, καθώς και integration με policy frameworks όπως το OPA. Η χρήση του eBPF επιτρέπει λεπτομερή έλεγχο της κίνησης σε επίπεδο kernel με zero-trust networking και runtime enforcement, επεκτείνοντας τους μηχανισμούς ασφαλείας του Kubernetes.

Το Calico βαθμολογήθηκε με (4), καθώς προσφέρει ισχυρές δυνατότητες ασφαλείας μέσω Network Policies και IPSec encryption για τη διασφάλιση της επικοινωνίας μεταξύ pods και κόμβων. Ωστόσο, απουσιάζει υποστήριξη mTLS και προηγμένα χαρακτηριστικά, όπως vulnerability scanning ή runtime protection, κάτι που αφήνει μικρά κενά σε περιβάλλοντα αυξημένων απαιτήσεων συμμόρφωσης.

Αντίθετα, το CoreDNS έλαβε χαμηλότερη βαθμολογία (2), καθώς παρέχει μόνο βασικούς μηχανισμούς προστασίας (π.χ. TLS, ACLs, query filtering) και απαιτεί πρόσθετη χειροκίνητη παραμετροποίηση για την αποτροπή επιθέσεων, όπως DNS spoofing ή cache poisoning

Διαλειτουργικότητα

Το Cilium συγκέντρωσε τη μέγιστη βαθμολογία (4), καθώς προσφέρει εκτενή και πλήρως ενοποιημένη συνεργασία με τα περισσότερα υποσυστήματα του Kubernetes και του οικοσυστήματος CNCF. Υποστηρίζει CRDs, API integrations και αμφίδρομη επικοινωνία με εργαλεία, όπως Prometheus, Grafana, Envoy, Istio και Hubble, επιτρέποντας ολιστική παρακολούθηση και διαχείριση δικτύου

Το Calico βαθμολογήθηκε με 3, καθώς αν και υποστηρίζει βασικά integrations (π.χ. Prometheus metrics, Kubernetes NetworkPolicies), η σύνδεση με πιο σύνθετα συστήματα, όπως service meshes ή observability stacks, απαιτεί χειροκίνητες ρυθμίσεις και δεν είναι πλήρως αυτοματοποιημένη.

Τέλος, το CoreDNS αξιολογήθηκε με 3 ως προς τη διαλειτουργικότητα, καθώς παρότι δεν ενσωματώνεται εκτενώς με εργαλεία εκτός της κατηγορίας observability, διαθέτει πολλαπλές διασυνδέσεις μέσα σε αυτή. Συγκεκριμένα, υποστηρίζει metrics exports προς Prometheus, ενσωμάτωση με Grafana για παρακολούθηση DNS metrics, καθώς και plugins για logging και tracing μέσω εργαλείων όπως Fluentd ή Loki. Επιπλέον, η πλήρης συμβατότητά του με το Kubernetes API το καθιστά θεμελιώδες στοιχείο του cluster networking. Ωστόσο, η διαλειτουργικότητά του παραμένει θεματικά περιορισμένη σε monitoring/observability συστήματα, χωρίς σύνδεση με policy ή CI/CD πλαίσια, γεγονός που δικαιολογεί την αξιολόγηση στο επίπεδο «Μέτρια διαλειτουργικότητα (3)».

Υποστήριξη & Κοινότητα

Τα Cilium, Calico και CoreDNS έλαβαν 5, καθώς αποτελούν CNCF graduated projects με ενεργές κοινότητες, συχνά releases, εκτενή τεκμηρίωση και ευρεία υιοθέτηση στο cloud-native οικοσύστημα. Η θεσμική στήριξη από την CNCF και η ενεργή συνεισφορά από εταιρείες όπως η Isovalent, η Tigera και η Cloud Native Computing Foundation διασφαλίζουν τη μακροπρόθεσμη βιωσιμότητα και υποστήριξή τους. (CNCF) [2].

Πίνακας 22: Συγκριτική αξιολόγηση εργαλείων στην κατηγορία Networking

	Εργαλείο		
Κριτήριο	**Cilium**	**Calico**	**CoreDNS**
Απόδοση (Performance)	**4**	**4**	**4**
Ευκολία Ενσωμάτωσης	**3**	**3**	**4**
Επεκτασιμότητα	**4**	**4**	**3**
Ασφάλεια	**4**	**4**	**3**
Διαλειτουργικότητα	**4**	**3**	**3**
Υποστήριξη & Κοινότητα	**4**	**4**	**4**

Πηγές

[1] (Sedghpour & Townend, 2022)

[2] (Cloud Native Computing Foundation (CNCF), 2024) – Networking Category.

[3] (Budigiri, Baumann, Mühlberg, Truyen, & Joosen, 2021)

[4] (Jakkaraju, 2020)

[5] (Kubernetes Documentation, 2024)

Η κατηγορία Networking συγκεντρώνει εργαλεία που αποτελούν τη ραχοκοκαλιά της επικοινωνίας και της ασφάλειας εντός Kubernetes clusters. Τα εργαλεία Cilium, Calico και CoreDNS παρουσιάζουν διαφορετικές αρχιτεκτονικές προσεγγίσεις, καλύπτοντας από το επίπεδο του data plane έως το επίπεδο ονοματολογίας και service discovery.

Συνολικά, το Cilium ξεχώρισε ως η πληρέστερη και τεχνολογικά πιο ώριμη λύση, επιτυγχάνοντας υψηλή απόδοση, επεκτασιμότητα και ασφάλεια χάρη στη χρήση του eBPF, που επιτρέπει λειτουργίες ελέγχου, δρομολόγησης και παρακολούθησης απευθείας στο επίπεδο του πυρήνα. Επιπλέον, η modular αρχιτεκτονική και τα εκτεταμένα APIs του προσφέρουν σημαντικά πλεονεκτήματα διαλειτουργικότητας και αυτοματοποίησης, καθιστώντας το κατάλληλο για περιβάλλοντα παραγωγής μεγάλης κλίμακας.

Το Calico διατηρεί επίσης υψηλή απόδοση και ωριμότητα, προσφέροντας ισχυρές δυνατότητες πολιτικών ασφάλειας και αξιόπιστο μηχανισμό routing. Αν και υστερεί ελαφρώς σε επεκτασιμότητα και αυτοματοποίηση σε σχέση με το Cilium, παραμένει μια εξαιρετικά σταθερή επιλογή για clusters που απαιτούν απλότητα, διαφάνεια και συμβατότητα με υπάρχουσες υποδομές.

Το CoreDNS, ως εγγενές συστατικό του Kubernetes, ξεχωρίζει για την ευκολία ενσωμάτωσης και την υψηλή αποδοτικότητα στη διαχείριση DNS queries, ωστόσο η λειτουργικότητά του είναι πιο εξειδικευμένη και περιορισμένη συγκριτικά με τα εργαλεία δικτυακής ασφάλειας και πολιτικών.

Σε επίπεδο διαλειτουργικότητας, όλα τα εργαλεία παρουσιάζουν ισχυρές δυνατότητες ενσωμάτωσης μέσω CRDs, APIs και plugins, ωστόσο απαιτούν εξειδικευμένη γνώση για προχωρημένες ρυθμίσεις. Η υποστήριξη και κοινότητα είναι έντονα ενεργές για όλα τα εργαλεία, με το Cilium και το Calico να έχουν ισχυρή παρουσία στο CNCF οικοσύστημα.

Συνοψίζοντας, το Cilium αναδεικνύεται ως η καλύτερη συνολικά επιλογή, χάρη στον συνδυασμό επιδόσεων, ασφάλειας και επεκτασιμότητας, ενώ το Calico αποτελεί πιο ώριμη και σταθερή λύση για περιβάλλοντα που δίνουν προτεραιότητα στην αξιοπιστία και τη διαφάνεια ρύθμισης. Το CoreDNS, παρότι περιορισμένο σε ρόλο, παραμένει αναπόσπαστο στοιχείο του Kubernetes networking stack, παρέχοντας κρίσιμες υπηρεσίες ονοματολογίας και service discovery με υψηλή αποδοτικότητα.

Από τη συγκριτική αποτίμηση των εργαλείων δικτύωσης προκύπτει ότι τα κριτήρια Απόδοση και Υποστήριξη/Κοινότητα εμφανίζουν τις υψηλότερες επιδόσεις (βαθμολογία 4 σε όλα τα εργαλεία), γεγονός που αντικατοπτρίζει τη σταθερότητα και την ωριμότητα των λύσεων Cilium και Calico στην επεξεργασία πακέτων και στην εφαρμογή πολιτικών ασφάλειας (network policies). Αντίθετα, το κριτήριο με τη χαμηλότερη επίδοση είναι η Ευκολία Ενσωμάτωσης και η Διαλειτουργικότητα, όπου τόσο το Cilium όσο και το Calico συγκεντρώνουν βαθμολογία 3, εξαιτίας της αυξημένης πολυπλοκότητας εγκατάστασης και ρύθμισης σε μεγάλα clusters. Συνολικά, η κατηγορία παρουσιάζει ισορροπημένες επιδόσεις, με μικρές αποκλίσεις που σχετίζονται περισσότερο με την ευχρηστία παρά με την τεχνική ικανότητα των εργαλείων

5.2.4 Storage, Backup & DR

Η παρούσα ενότητα παρουσιάζει τα αποτελέσματα της συγκριτικής αξιολόγησης των εργαλείων που εντάσσονται στην κατηγορία **Storage, Backup & Disaster Recovery (DR)**, όπως αναλύθηκαν στο Κεφάλαιο 4. Η σύγκριση πραγματοποιήθηκε βάσει των κριτηρίων που ορίστηκαν στην Ενότητα 5.1 (*Απόδοση, Ευκολία Ενσωμάτωσης, Επεκτασιμότητα, Ασφάλεια, Διαλειτουργικότητα, Υποστήριξη & Κοινότητα*), με τιμές αποτίμησης στην κλίμακα 1–5, σύμφωνα με τις αντίστοιχες περιγραφές. Οι αξιολογήσεις βασίζονται τόσο στα ευρήματα της βιβλιογραφικής ανασκόπησης και των τεχνικών πηγών που παρουσιάστηκαν στο Κεφάλαιο 4 όσο και σε τεκμηριωμένες πηγές, όπως η CNCF Landscape και η επίσημη τεκμηρίωση των εργαλείων.

Απόδοση (Performance)

Η απόδοση των εργαλείων αξιολογήθηκε βάσει της ταχύτητας λήψης και επαναφοράς αντιγράφων, του χειρισμού μεγάλων όγκων δεδομένων και της επίπτωσης στις επιδόσεις του cluster κατά τη διάρκεια των διαδικασιών.

Το **Kasten K10** καταγράφει την υψηλότερη απόδοση (4), καθώς παρουσιάζει πολύ υψηλή απόδοση χάρη στους βελτιστοποιημένους μηχανισμούς backup/restore που υποστηρίζουν enterprise περιβάλλοντα, με αποτελεσματική αξιοποίηση πόρων και ελάχιστο latency [4][5]. Παρότι οι επιδόσεις του παραμένουν σταθερές σε κάπως μεγάλα workloads, η αυξημένη κατανάλωση αποθηκευτικού χώρου και bandwidth σε σενάρια πολύ μεγάλου όγκου δεδομένων το διαφοροποιεί από το επίπεδο πλήρους βελτιστοποίησης (5).

Το **Velero** ακολουθεί με (3), καθώς παρέχει σταθερή και αξιόπιστη απόδοση για μικρομεσαία clusters, ωστόσο παρατηρούνται καθυστερήσεις κατά την αποκατάσταση (restore) μεγάλων συνόλων δεδομένων [2]. Η συμπεριφορά αυτή αντανακλά το επίπεδο «μέτριας απόδοσης», καθώς το εργαλείο ανταποκρίνεται ικανοποιητικά σε βασικά workloads, αλλά παρουσιάζει εμφανή πτώση απόδοσης υπό αυξημένο φόρτο.

Το **K8sES** βαθμολογήθηκε με 3 δεδομένου ότι επιτυγχάνει ικανοποιητική απόδοση σε τυπικές συνθήκες, όμως η εξάρτησή του από εξωτερικό Elasticsearch cluster και η έλλειψη βελτιστοποιήσεων οδηγούν σε εμφανείς καθυστερήσεις σε περιπτώσεις μεγάλης κλίμακας [6]. Συνεπώς, η απόδοσή του χαρακτηρίζεται μέτρια, χωρίς όμως σημαντικά προβλήματα στα βασικά σενάρια χρήσης.

Το **Ramen** βαθμολογήθηκε με 2 αφού εμφανίζει χαμηλότερη απόδοση λόγω αυξημένου latency κατά τη διαδικασία συγχρονισμού και εξάρτησης από εξωτερικά συστήματα αποθήκευσης (π.χ. S3 replication) [7]. Οι καθυστερήσεις αυτές καθιστούν το εργαλείο κατάλληλο κυρίως για δοκιμαστικά ή μικρής κλίμακας περιβάλλοντα, αντανακλώντας το επίπεδο «χαμηλής απόδοσης» σύμφωνα με τα ορισμένα κριτήρια.

Ευκολία Ενσωμάτωσης (Ease of Integration)

Το Kasten K10 βαθμολογήθηκε με 4, χάρη στη modular αρχιτεκτονική του και την υποστήριξη τεκμηριωμένων APIs που επιτρέπουν την προσαρμογή και επέκταση λειτουργιών για enterprise-scale deployments. Υποστηρίζει integration με πολλαπλά storage backends και policy frameworks, Ωστόσο, η πλήρης αξιοποίηση των δυνατοτήτων του —όπως η διασύνδεση με πολλαπλά storage backends ή πολιτικές προστασίας δεδομένων— απαιτεί πρόσθετη παραμετροποίηση και εξειδικευμένη γνώση Kubernetes storage και RBAC, γεγονός που το διαφοροποιεί από εργαλεία με πλήρως αυτοματοποιημένη ή out-of-the-box εγκατάστασ

Το Velero αξιολογήθηκε με 4 ως προς την ευκολία ενσωμάτωσης, καθώς η βασική εγκατάστασή του μέσω Helm chart ή YAML manifests είναι πλήρως τεκμηριωμένη και μπορεί να ολοκληρωθεί με ελάχιστα βήματα. Η διασύνδεσή του με Kubernetes APIs και storage plugins είναι ομαλή, ενώ προσφέρει ενσωμάτωση με τους περισσότερους cloud providers μέσω έτοιμων plugins. Παρότι η προσαρμογή για multi-cloud ή hybrid περιβάλλοντα απαιτεί επιπλέον παραμετροποίηση (όπως διαχείριση credentials ή custom object storage endpoints), αυτή αφορά κυρίως προηγμένα σενάρια παραγωγής και όχι τη βασική λειτουργικότητα του εργαλείου. Συνεπώς, το Velero κατατάσσεται στο επίπεδο «Καλή ευκολία ενσωμάτωσης (4)».

Το K8sES αξιολογήθηκε με 2 καθως προσφέρει περιορισμένη αυτοματοποίηση — δεν διαθέτει επίσημο Helm chart ή operator, με αποτέλεσμα η εγκατάσταση να απαιτεί χειροκίνητη παραμετροποίηση YAMLs και προσαρμογές για να λειτουργήσει σωστά με εξωτερικά Elasticsearch clusters [6]. Η ανάγκη αυτή για χειροκίνητες ρυθμίσεις και η απουσία πλήρους τεκμηρίωσης το κατατάσσουν στο επίπεδο δύσκολης ενσωμάτωσης.

Το Ramen αξιολογήθηκε με 2. Παρά το ότι βασίζεται σε CNCF components, η εγκατάσταση προϋποθέτει εξειδικευμένες γνώσεις OpenShift APIs και διαχείρισης replication controllers, περιορίζοντας τη γενική ευκολία ενσωμάτωσης [7]. Αντιστοιχεί σε «δύσκολη ενσωμάτωση» λόγω της ανάγκης προσαρμογής σε συγκεκριμένες πλατφόρμες.

Επεκτασιμότητα

Το Kasten K10 αξιολογήθηκε με 4 , καθώς διαθέτει τεκμηριωμένα APIs, δυνατότητα ενσωμάτωσης με εξωτερικά storage backends και υποστήριξη για custom policies μέσω CRDs και hooks. Παράλληλα, παρέχει integration points με συστήματα authentication, monitoring και alerting, γεγονός που το καθιστά επεκτάσιμο σε enterprise-scale περιβάλλοντα [4][5]. Παρότι η αρχιτεκτονική του δεν είναι πλήρως modular, υποστηρίζει προσθήκη νέων λειτουργιών χωρίς σημαντική τροποποίηση της βάσης του.

Το Velero αξιολογήθηκε με 3. Υποστηρίζει plugins για custom storage providers και backup targets, καθώς και extensibility μέσω CRDs και hooks, ωστόσο η διαδικασία ανάπτυξης και

παραμετροποίησης είναι σύνθετη και απαιτεί σημαντική τεχνική γνώση [2][3]. Η τεκμηρίωση καλύπτει βασικά σενάρια, αλλά όχι σε βάθος enterprise integrations, γεγονός που περιορίζει τη συνολική επεκτασιμότητα.

Το Ramen αξιολογήθηκε με 3 αφού υποστηρίζει επεκτασιμότητα μέσω custom controllers και CRDs, επιτρέποντας την προσαρμογή σε multi-cluster disaster recovery σενάρια. Ωστόσο, η εξάρτηση από OpenShift APIs και η περιορισμένη τεκμηρίωση για μη Red Hat περιβάλλοντα μειώνουν τη δυνατότητα ευρείας προσαρμογής [6][7]. Συνολικά, παρέχει επεκτάσεις σε συγκεκριμένο οικοσύστημα, αλλά όχι πλήρως ανεξάρτητες.

Το K8sES αξιολογήθηκε με 2 καθώς εμφανίζει περιορισμένη επεκτασιμότητα. Βασίζεται κυρίως σε configuration files και scripts χωρίς επίσημο API ή plugin framework. Οι δυνατότητες προσαρμογής περιορίζονται σε βασικές παραμέτρους λειτουργίας, ενώ η απουσία ενεργής κοινότητας και τυποποιημένων integration points δυσκολεύει την ανάπτυξη νέων επεκτάσεων [7].

Ασφάλεια (Security)

Αξιολογήθηκε η ύπαρξη κρυπτογράφησης, ελέγχου πρόσβασης και συμμόρφωσης με πρότυπα.

Το Kasten K10 έλαβε (4), καθώς παρέχει end-to-end encryption, TLS και RBAC, εξασφαλίζοντας ισχυρή προστασία δεδομένων και έλεγχο πρόσβασης σε enterprise περιβάλλοντα [4]. Επίσης παρεχει ολοκληρωμένη προστασία για δεδομένα backup και δυνατότητα ενσωμάτωσης με HashiCorp Vault για ασφαλή διαχείριση κλειδιών. Παρότι διαθέτει ενσωματωμένους μηχανισμούς πολιτικών ασφαλείας και auditing, δεν τεκμηριώνεται πλήρης συμμόρφωση με διεθνή πρότυπα ή advanced πρακτικές, όπως zero-trust networking.

Το Velero, το Ramen και το K8sES αξιολογούνται με βαθμό 3 ως προς την ασφάλεια. Και τα τρία εργαλεία αξιοποιούν τους ενσωματωμένους μηχανισμούς προστασίας του Kubernetes (RBAC, TLS, Secrets), παρέχοντας ικανοποιητικό επίπεδο ασφάλειας για τυπικά σενάρια χρήσης. Ωστόσο, η πληρότητα της ασφάλειας τους εξαρτάται από το υποκείμενο storage backend, το οποίο είναι υπεύθυνο για την εφαρμογή end-to-end encryption και compliance πολιτικών. Συνεπώς, δεν εντοπίζονται σημαντικά κενά, αλλά η ασφάλεια τους δεν θεωρείται πλήρως αυτόνομη.

Διαλειτουργικότητα (Interoperability)

Το Kasten K10 αξιολογήθηκε με 4 καθώς παρουσιάζει υψηλή διαλειτουργικότητα, προσφέροντας πλήρη ενσωμάτωση με CSI drivers, policy και monitoring εργαλεία, όπως Prometheus, Vault και Grafana, καθώς και αυτόματη αναγνώριση storage backends μέσω standard APIs.

Το Velero αξιολογήθηκε με 3 καθώς εμφανίζει καλή κάλυψη, αξιοποιώντας plugin-based αρχιτεκτονική για συνεργασία με πολλαπλά storage συστήματα και cloud providers· ωστόσο, απαιτείται χειροκίνητη ρύθμιση για ορισμένα integrations και CRD επεκτάσεις.

Αντίθετα, τα Ramen και K8sES (2) παρουσιάζουν περιορισμένη διαλειτουργικότητα, καθώς στηρίζονται σε συγκεκριμένα APIs (π.χ. OpenShift, Ceph) και δεν διαθέτουν εκτεταμένο plugin ή CRD framework, γεγονός που περιορίζει τη συνεργασία τους με τρίτα εργαλεία του οικοσυστήματος Kubernetes.

Υποστήριξη & Κοινότητα (Support & Community)

Το Velero (5) συγκεντρώνει την ανώτατη βαθμολογία, καθώς αποτελεί έργο του CNCF με ιδιαίτερα ενεργό οικοσύστημα χρηστών και συνεισφερόντων. Διαθέτει χιλιάδες contributors, συχνές ενημερώσεις, συνεχή επικοινωνία μέσω καναλιών, όπως Slack και GitHub, καθώς και σαφώς καθορισμένο community-driven roadmap. Το επίπεδο υποστήριξης ενισχύεται από την ανοικτή φύση του έργου, που εξασφαλίζει τη μακροχρόνια βιωσιμότητα και συνεχή βελτίωση.

Αντίθετα, το Kasten K10 (3), ενώ παρουσιάζει εξίσου υψηλό επίπεδο αξιοπιστίας και σταθερής συντήρησης, η υποστήριξή του στηρίζεται κυρίως σε εμπορικά κανάλια μέσω της Veeam. Η κοινότητα είναι μικρότερη και λιγότερο ανοιχτή σε εξωτερικές συνεισφορές, καθώς η ανάπτυξη καθοδηγείται πρωτίστως από τον οργανισμό. Έτσι, παρότι η τεχνική υποστήριξη θεωρείται εξαιρετική, η συνολική διασπορά και δυναμική της κοινότητας είναι περιορισμένη σε σχέση με έργα ανοικτής ανάπτυξης, όπως το Velero.

Τα Ramen και K8sES (2) παρουσιάζουν σαφώς μικρότερη δραστηριότητα, με περιορισμένο αριθμό contributors και ελάχιστα κανάλια επικοινωνίας. Η υποστήριξή τους περιορίζεται σε ερευνητικό ή οργανωσιακό επίπεδο, χωρίς την ύπαρξη εκτεταμένου οικοσυστήματος χρηστών ή ενεργού κοινοτικού μηχανισμού εξέλιξης.

Πίνακας 23: Συγκριτική αξιολόγηση εργαλείων στην κατηγορία Storage, Backup & DR

	Εργαλείο			
Κριτήριο	**Velero**	**Kasten K10**	**Ramen**	**K8sES**
Απόδοση (Performance	**3**	**4**	**2**	**3**
Ευκολία Ενσωμάτωσης	**4**	**4**	**2**	**2**
Επεκτασιμότητα	**3**	**4**	**2**	**2**
Ασφάλεια	**3**	**4**	**3**	**3**
Διαλειτουργικότητα	**3**	**4**	**2**	**2**
Υποστήριξη & Κοινότητα	**5**	**3**	**2**	**2**

Πηγές

[1] (Velero, 2025)

[2] *(Jin, Muench, Janssen, Hatfield, & Deenadhayalan, 2024)*

[3] *(Kim & Jung, 2024)*

[4] (Bergh, Smith, & Guevara, 2022)

[5] (Zhu, Kang, He, & Oki, 2021)

[6] (Kim, Choi, & Jung, 2024)

[7] (Wen, et al., 2023)

Η κατηγορία Backup & Recovery αξιολογεί εργαλεία που εξασφαλίζουν τη διαθεσιμότητα και την ακεραιότητα δεδομένων σε περιβάλλοντα Kubernetes, με διαφορετικά επίπεδα ωριμότητας και αυτοματοποίησης. Τα αποτελέσματα δείχνουν ότι το Kasten K10 ξεχωρίζει ως η πιο ολοκληρωμένη και αποδοτική λύση, συνδυάζοντας υψηλή απόδοση, επεκτασιμότητα και ασφάλεια. Οι βελτιστοποιημένοι μηχανισμοί backup/restore, η modular αρχιτεκτονική και η εκτενής υποστήριξη enterprise περιβαλλόντων του εξασφαλίζουν κορυφαίες επιδόσεις με βαθμό (4) σχεδόν σε όλα τα κριτήρια. Παρά τον εμπορικό του χαρακτήρα, προσφέρει τεκμηριωμένα APIs και ευρεία ενσωμάτωση με εργαλεία όπως Prometheus, Vault και Grafana, διατηρώντας υψηλή διαλειτουργικότητα και αξιοπιστία.

Το Velero ακολουθεί ως η πιο ώριμη open-source επιλογή, με εξαιρετική υποστήριξη κοινότητας (βαθμός 5) και καλή απόδοση σε μικρομεσαία workloads. Αν και υστερεί σε enterprise χαρακτηριστικά (όπως πλήρη αυτοματοποίηση και βελτιστοποίηση μεγάλου όγκου δεδομένων), η plugin-based αρχιτεκτονική του επιτρέπει επεκτασιμότητα και ευελιξία, καθιστώντας το ιδιαίτερα προσαρμόσιμο εργαλείο για οργανισμούς με σύνθετες ανάγκες.

Το Ramen και το K8sES, από την άλλη πλευρά, εμφανίζουν περιορισμένη απόδοση και διαλειτουργικότητα, κυρίως λόγω της εξάρτησής τους από εξωτερικά APIs (OpenShift, Elasticsearch) και της περιορισμένης τεκμηρίωσης. Το Ramen προσφέρει αξιοσημείωτη λειτουργικότητα για multi-cluster recovery αλλά με χαμηλή αποδοτικότητα και υψηλή πολυπλοκότητα ενσωμάτωσης (βαθμοί 2–3), ενώ το K8sES παραμένει περισσότερο ερευνητικό εργαλείο με χαμηλή υποστήριξη και ελάχιστη κοινότητα.

Σε γενικές γραμμές, τα εργαλεία της κατηγορίας επιδεικνύουν μέτρια ευκολία ενσωμάτωσης και μέτρια προς καλή επεκτασιμότητα, με τα περισσότερα να υποστηρίζουν τα APIs και τους μηχανισμούς ενοποίησης του Kubernetes. Το κριτήριο της Ασφάλειας εμφανίζει μέτριες προς καλές τιμές, καθώς όλα τα εργαλεία αξιοποιούν τους μηχανισμούς RBAC και TLS για έλεγχο πρόσβασης και ασφαλή επικοινωνία.

Αντίθετα, η Διαλειτουργικότητα αποτελεί το ασθενέστερο κριτήριο, καθώς τα εργαλεία Ramen και K8sES παρουσιάζουν περιορισμένη υποστήριξη πολλαπλών υποδομών και απαιτούν αυξημένη παραμετροποίηση για ενοποίηση με άλλα συστήματα.

Συνολικά, η κατηγορία παρουσιάζει υψηλή ωριμότητα, με τα Kasten K10 και Velero να ξεχωρίζουν. Παρ’ όλα αυτά, η Επεκτασιμότητα και η Διαλειτουργικότητα αποτελούν πεδία με περιθώρια βελτίωσης, κυρίως ως προς τη βαθύτερη ενοποίηση με τα native Kubernetes APIs και τη διαχείριση ετερογενών περιβαλλόντων.

5.2.5 CI/CD

Η κατηγορία **CI/CD** (Continuous Integration / Continuous Delivery) περιλαμβάνει εργαλεία που αυτοματοποιούν τη διαδικασία κατασκευής, ελέγχου και διάθεσης λογισμικού, ενσωματώνοντας σύγχρονες πρακτικές GitOps και DevOps. Στο Κεφάλαιο 4 παρουσιάστηκαν και αναλύθηκαν σε βάθος τα εργαλεία **Argo CD**, **Jenkins X**, **Argo Workflows**, **Drone** και **GoCD**, αξιοποιώντας επιστημονικές δημοσιεύσεις, την επίσημη τεκμηρίωση και το CNCF Landscape.

Απόδοση (Performance)

Το Argo CD αξιολογήθηκε με 4 καθώς προσφέρει υψηλή απόδοση σε GitOps ροές, ιδιαίτερα σε multi-cluster περιβάλλοντα [1][2]. Οι καθυστερήσεις εμφανίζονται μόνο όταν τα manifests δεν είναι βελτιστοποιημένα ή όταν υπάρχει μεγάλος όγκος συγχρονισμών. Αυτά τα ζητήματα είναι μικρής έκτασης και δεν επηρεάζουν σημαντικά τη συνολική απόδοση.

Το Jenkins X αξιολογήθηκε με 3. Η εκτέλεση pipelines είναι σταθερή και επαρκής σε μεσαίο φόρτο, ωστόσο σε μεγάλα workloads παρατηρείται εμφανής αύξηση latency και μείωση throughput [4]. Αυτή η συμπεριφορά συνάδει με το επίπεδο 3 (μέτρια απόδοση), όπου η απόδοση παραμένει ικανοποιητική στα βασικά workloads αλλά υποχωρεί σε υψηλά φορτία.

Το Argo Workflows αξιλογήθηκε με 4 καθώς επιδεικνύει πολύ καλή απόδοση στην εκτέλεση παράλληλων εργασιών (parallel jobs), χάρη στην εγγενή του αρχιτεκτονική για task-level parallelism [6]. Η συνολική απόδοση εξαρτάται μεν από την υποδομή Kubernetes, ωστόσο τα ζητήματα που προκύπτουν είναι προβλέψιμα και μικρής κλίμακας, δηλαδή δεν μειώνουν ουσιαστικά την αποτελεσματικότητα του εργαλείου.

Το Drone αξιλογήθηκε με 3 καθώς παρουσιάζει καλή απόδοση σε μικρά και μεσαία workloads, με γρήγορη εκτέλεση και χαμηλή κατανάλωση πόρων. Σε πολύ μεγάλα workloads, ωστόσο, αναφέρονται εμφανείς περιορισμοί, όπως μείωση throughput έως 25–30% και αύξηση χρόνου εκτέλεσης builds [7]. Αυτά τα φαινόμενα δεν υποβαθμίζουν την απόδοση σε χαμηλή κατηγορία, αλλά την καθιστούν μέτρια προς καλή.

Το GoCD βαθμολογήθηκε με 3 αφού εμφανίζει σταθερή λειτουργία και αξιόπιστη εκτέλεση pipelines, όμως σε περιπτώσεις πολύπλοκων αγωγών (με πολλά στάδια ή εξαρτήσεις) η απόδοση μειώνεται εμφανώς [8]. Η συμπεριφορά αυτή ανταποκρίνεται πλήρως στο επίπεδο 3, καθώς η βασική απόδοση είναι ικανοποιητική αλλά δεν διατηρείται πλήρως σε σύνθετα workloads.

Ευκολία Ενσωμάτωσης (Ease of Integration)

Η ενσωμάτωση του Argo CD θεωρείται μέτριας ευκολίας, καθώς η εγκατάσταση πραγματοποιείται μέσω Helm chart ή Argo CD Operator, ωστόσο απαιτείται εξοικείωση με έννοιες GitOps και βασικές αρχές Kubernetes για τη σωστή ρύθμιση στοιχείων, όπως τα RBAC, τα repository credentials και οι sync policies [1]. Παρότι η τεκμηρίωση είναι πλήρης, η διαδικασία δεν είναι πλήρως αυτοματοποιημένη, γεγονός που δικαιολογεί τη βαθμολογία 3.

Η εγκατάσταση του Jenkins X μπορεί να πραγματοποιηθεί μέσω του CLI και των διαθέσιμων Helm charts, τα οποία αυτοματοποιούν σημαντικό μέρος της διαδικασίας [4]. Ωστόσο, απαιτούνται επιπλέον ρυθμίσεις για την ολοκλήρωση της ενσωμάτωσης, όπως η διαμόρφωση domain, ingress

controllers και η σύνδεση με Git providers. Παρά την πολυπλοκότητα, η τεκμηρίωση και τα προκαθορισμένα pipelines καθιστούν τη διαδικασία εφικτή χωρίς να απαιτείται βαθιά τεχνική γνώση. Συνεπώς, η τιμή 3 αντικατοπτρίζει την μέτρια ευκολία ενσωμάτωσης, με ανάγκη για προσαρμογές αλλά χωρίς ουσιαστικά εμπόδια.

Η ενσωμάτωση του Argo Workflows προϋποθέτει κατανόηση Custom Resource Definitions (CRDs) και χρήσης YAML templates, καθώς κάθε workflow υλοποιείται ως CRD στο Kubernetes [6]. Το εργαλείο συνοδεύεται από Helm chart και εκτενή τεκμηρίωση, γεγονός που διευκολύνει τη διαδικασία εγκατάστασης. Παρότι απαιτείται χειροκίνητη ρύθμιση ορισμένων παραμέτρων (όπως RBAC ή persistence), η διαδικασία παραμένει ελεγχόμενη και προβλέψιμη. Ως εκ τούτου, η τιμή 3 θεωρείται ενδεικτική μέτριας ευκολίας ενσωμάτωσης.

Το Drone παρουσιάζει σχετικά υψηλή ευκολία ενσωμάτωσης, λόγω της δοχειοποιημένης αρχιτεκτονικής του και της δυνατότητας ανάπτυξης μέσω Helm chart ή απλών Docker containers [7]. Η αρχιτεκτονική βασισμένη σε microservices επιτρέπει ευέλικτη σύνδεση με GitHub ή GitLab, ενώ οι προκαθορισμένες εικόνες container και το ενσωματωμένο web interface μειώνουν σημαντικά την ανάγκη για πρόσθετες ρυθμίσεις. Επομένως, το εργαλείο ταξινομείται στο επίπεδο 4, το οποίο αντιστοιχεί σε σχετικά εύκολη και καλή ενσωμάτωση, με περιορισμένες χειροκίνητες παρεμβάσεις για προχωρημένες λειτουργίες.

Η εγκατάσταση του GoCD προϋποθέτει τη δημιουργία server και agents, διαδικασία που μπορεί να υλοποιηθεί μέσω Helm chart, ή Kubernetes manifests, αξιοποιώντας τα επίσημα Docker images του εργαλείου [8]. Παρότι η διαδικασία είναι επαρκώς τεκμηριωμένη, απαιτείται χειροκίνητη παραμετροποίηση των agents και ρύθμιση authentication, συνεπώς η συνολική ευκολία ενσωμάτωσης κρίνεται μέτρια (τιμή 3).

Επεκτασιμότητα

Το Argo CD παρουσιάζει καλή επεκτασιμότητα και βαθμολογήθηκε με 4, καθώς υποστηρίζει πλήρως τεκμηριωμένο REST API και Webhooks για διασύνδεση με CI pipelines και εξωτερικά συστήματα ειδοποιήσεων [1][2]. Επιπλέον, παρέχει δυνατότητα δημιουργίας custom plugins για τον μηχανισμό “config management” (π.χ. Kustomize, Helm, Jsonnet), ενώ επιτρέπει την επέκταση μέσω CRDs και custom controllers στο Kubernetes. Ωστόσο, δεν διαθέτει ολοκληρωμένο SDK ή επίσημο plugin framework για την ανάπτυξη νέων modules από τρίτους, περιορίζοντας έτσι το εύρος της επεκτασιμότητας σε σύγκριση με πλήρως modular έργα όπως το Grafana ή το Vault. Για τον λόγο αυτό, το Argo CD κατατάσσεται στο επίπεδο 4 («Καλή επεκτασιμότητα σε μεγάλα clusters»).

Το Jenkins X διαθέτει εκτενή υποστήριξη για επεκτασιμότητα μέσω του Jenkins X Pipeline Operator, των Tekton CRDs και ενός συνόλου API endpoints που επιτρέπουν προσαρμογές στη διαδικασία CI/CD [4]. Υποστηρίζει επίσης custom builders, plugins και webhooks για ενοποίηση με Git providers και άλλες υπηρεσίες. Ωστόσο, η έλλειψη ενός επίσημου SDK ή ενιαίου plugin framework περιορίζει τη δυνατότητα δημιουργίας νέων modules από τρίτους, ενώ μεγάλο μέρος της επεκτασιμότητας προέρχεται από την υποκείμενη πλατφόρμα Tekton και όχι από τον πυρήνα

του Jenkins X. Για τον λόγο αυτό, κατατάσσεται στο επίπεδο 4 («Καλή επεκτασιμότητα σε μεγάλα clusters»).

Το Argo Workflows βαθμολογήθηκε με 4 διότι είναι εγγενώς επεκτάσιμο, καθώς στηρίζεται εξ ολοκλήρου σε Custom Resource Definitions (CRDs) και Kubernetes-native αρχιτεκτονική [6]. Παρέχει τεκμηριωμένο API, webhooks, καθώς και Workflow Templates που επιτρέπουν την επαναχρησιμοποίηση και σύνθεση σύνθετων ροών. Επιπλέον, υποστηρίζει custom executors και δυνατότητα ενσωμάτωσης με τρίτα συστήματα (π.χ. Argo Events, ML pipelines). Ωστόσο, δεν διαθέτει επίσημο plugin SDK ή framework για την ανάπτυξη τρίτων επεκτάσεων, ενώ η προσαρμογή του απαιτεί προχωρημένη γνώση Kubernetes και δηλωτικής σύνταξης YAML. Για τον λόγο αυτό, το εργαλείο τοποθετείται στο επίπεδο 4 («Καλή επεκτασιμότητα σε μεγάλα clusters»)

Το Drone προσφέρει μέτρια επεκτασιμότητα, καθώς υποστηρίζει τη δημιουργία custom plugins μέσω Docker containers που λειτουργούν ως βήματα (steps) στο pipeline [7]. Παρότι η φιλοσοφία του container-based pipeline επιτρέπει μια μορφή modularity, το εργαλείο δεν διαθέτει πλήρως τεκμηριωμένο API ούτε ευρεία υποστήριξη για native SDKs ή integration hooks. Οι επεκτάσεις περιορίζονται ουσιαστικά σε scripts και container images, γεγονός που αντιστοιχεί στο επίπεδο 3, δηλαδή μέτρια επεκτασιμότητα με δυνατότητα προσαρμογής μέσω containers αλλά χωρίς πλήρη API υποστήριξη.

Το GoCD υποστηρίζει επεκτασιμότητα μέσω plugin framework, το οποίο επιτρέπει την προσθήκη custom extensions για authentication, SCM integrations και notifications [8]. Αν και διαθέτει API για βασικές λειτουργίες, η διαδικασία ανάπτυξης plugins είναι περίπλοκη και απαιτεί χρήση Java SDK, ενώ η τεκμηρίωση είναι περιορισμένη σε ορισμένες περιπτώσεις. Συνεπώς, το εργαλείο αξιολογείται με τιμή 3.

Ασφάλεια (Security)

Το Argo CD παρέχει ισχυρό και πολυεπίπεδο σύστημα ασφάλειας, αξιοποιώντας RBAC, SSO (OAuth2, SAML, LDAP), token-based authentication, καθώς και policy enforcement μέσω OPA/Gatekeeper [1][2]. Υποστηρίζει audit logging και encryption των secrets, ενώ εφαρμόζει granular πολιτικές συγχρονισμού ανά namespace ή project. Παρότι η συνολική ασφάλεια εξαρτάται από τη ρύθμιση του υποκείμενου Kubernetes cluster, το Argo CD επεκτείνει ουσιαστικά τις εγγενείς του δυνατότητες, παρέχοντας αυτόνομους ελέγχους πολιτικής και ταυτοποίησης. Για τον λόγο αυτό, λαμβάνει την τιμή 4, που αντιστοιχεί σε ισχυρή ασφάλεια με μικρά μόνο κενά.

Το Jenkins X αξιοποιεί την απομόνωση του Kubernetes για την εκτέλεση pipelines, προσφέροντας RBAC, service accounts και namespace-level isolation [4]. Ωστόσο, η ασφάλεια του περιβάλλοντος εξαρτάται από τη σωστή ρύθμιση των cluster permissions, ενώ το εργαλείο δεν εφαρμόζει εγγενείς μηχανισμούς policy enforcement ή encryption των secrets. Απαιτεί, συνεπώς, πρόσθετο hardening (π.χ. network policies, secrets rotation, restricted service accounts). Η τιμή 3 είναι κατάλληλη, καθώς το εργαλείο παρέχει ικανοποιητική, αλλά όχι πλήρως αυτοματοποιημένη ή ενισχυμένη ασφάλεια.

Το Argo Workflows υποστηρίζει RBAC, namespace-based isolation, security contexts, audit logs και encryption [6]. Η στενή του ενοποίηση με το Kubernetes επιτρέπει granular έλεγχο πρόσβασης και διαχείριση εκτελέσεων με ασφαλή τρόπο. Επιπλέον, μπορεί να συνδεθεί με Argo Events και Vault για ασφαλή διαχείριση μυστικών. Η αξιολόγηση 4 τεκμηριώνεται από το γεγονός ότι προσφέρει ισχυρή ασφάλεια και ενσωμάτωση με εξωτερικά συστήματα, χωρίς όμως να ενσωματώνει zero-trust μηχανισμούς.

Το Drone βασίζεται κυρίως σε token-based authentication και εξωτερική αυθεντικοποίηση μέσω Git providers [7]. Αν και τα pipelines εκτελούνται σε απομονωμένα containers, το εργαλείο δεν διαθέτει RBAC, network policies ή auditing μηχανισμούς, και δεν εφαρμόζει κρυπτογράφηση μυστικών σε επίπεδο cluster. Ως αποτέλεσμα, παρέχει βασική προστασία, επαρκή για απλά workloads αλλά με εμφανή κενά όσον αφορά policy enforcement και compliance. Η τιμή 2 αντικατοπτρίζει περιορισμένη ασφάλεια με βασικούς μηχανισμούς.

Το GoCD ενσωματώνει authentication, authorization και TLS/SSL, ωστόσο οι δυνατότητες αυτές θεωρούνται παρωχημένες σε σύγκριση με τις σύγχρονες απαιτήσεις ασφάλειας Kubernetes [8]. Η κρυπτογράφηση secrets, οι πολιτικές πρόσβασης και η δυνατότητα auditing απαιτούν εξωτερικά πρόσθετα ή χειροκίνητη ρύθμιση. Η απουσία native policy enforcement και secret management περιορίζει την αποτελεσματικότητά του. Κατά συνέπεια, η αξιολόγηση 2 θεωρείται τεκμηριωμένη, αντιστοιχώντας σε βασική ασφάλεια με εμφανή ελλείμματα.

Διαλειτουργικότητα (Interoperability)

Το Argo CD συγκέντρωσε τη μέγιστη βαθμολογία (4), καθώς προσφέρει εκτενή και πλήρως ενοποιημένη συνεργασία με τα περισσότερα υποσυστήματα του Kubernetes και του οικοσυστήματος CNCF. Βασίζεται σε CRDs και Controllers για τη διαχείριση εφαρμογών και πολιτικών GitOps, υποστηρίζοντας άμεση αμφίδρομη επικοινωνία με το Kubernetes API. Επιπλέον, ενσωματώνεται απρόσκοπτα με εργαλεία όπως Helm, Kustomize, Vault, Prometheus, OPA/Gatekeeper και Argo Rollouts, παρέχοντας μια ολιστική και native εμπειρία λειτουργίας μέσα στο cluster.

Το Jenkins X έλαβε 3, καθώς αν και βασίζεται στο Kubernetes και αξιοποιεί Tekton CRDs για pipelines, η ενοποίηση με τα υπόλοιπα εργαλεία του οικοσυστήματος δεν είναι πλήρως αυτοματοποιημένη. Η σύνδεση με monitoring ή policy frameworks (π.χ. Prometheus, OPA) απαιτεί χειροκίνητες ρυθμίσεις, ενώ αρκετές λειτουργίες περνούν μέσω του JX CLI, το οποίο λειτουργεί ως ενδιάμεσο επίπεδο και όχι ως εγγενές Kubernetes API integration. Έτσι, η διαλειτουργικότητα είναι ικανοποιητική αλλά όχι πλήρως ενοποιημένη.

Το Argo Workflows αξιολογήθηκε επίσης με 4, καθώς είναι πλήρως Kubernetes-native. Κάθε ροή εργασίας (workflow) και πρότυπο (template) υλοποιείται ως Custom Resource, επιτρέποντας άμεση διαχείριση μέσω του Kubernetes API. Το εργαλείο ενσωματώνεται με Argo CD, Argo Events, Prometheus, Vault και Grafana, προσφέροντας πλήρη αμφίδρομη συνεργασία με τα υποσυστήματα του cluster. Η λειτουργία του επεκτείνει φυσικά το Kubernetes οικοσύστημα, χωρίς να απαιτεί εξωτερικά επίπεδα διαχείρισης

Το Drone βαθμολογήθηκε με 2, καθώς η συνεργασία του με το Kubernetes περιορίζεται στην εκτέλεση workloads μέσω Kubernetes runners. Δεν αξιοποιεί CRDs, Operators ή Kubernetes-native APIs, ενώ οι περισσότερες ενσωματώσεις γίνονται μέσω container-based plugins, χωρίς άμεση επικοινωνία με το control plane. Η ενοποίησή του με το οικοσύστημα CNCF είναι βασική και μη αμφίδρομη.

Υποστήριξη & Κοινότητα (Support & Community)

Το Argo CD αξιολογήθηκε με 5 καθώς διαθέτει πολύ ισχυρή και ενεργή κοινότητα, και αποτελεί CNCF Graduated project, με χιλιάδες contributors και συνεχή ανάπτυξη στο GitHub [1]. Υποστηρίζεται επίσημα από την CNCF και μεγάλες εταιρείες, όπως Red Hat, Intuit και Codefresh, ενώ κυκλοφορούν συχνά νέα releases και security patches. Η τεκμηρίωση είναι πλήρης και υπάρχουν πολλά διαθέσιμα κανάλια υποστήριξης (Slack, GitHub Discussions, ArgoCon events).

Το Jenkins X αξιολογήθηκε με 4 καθώς διατηρεί μεγάλη και δραστήρια κοινότητα με εκατοντάδες συνεισφέροντες [4]. Παρότι δεν βρίσκεται πλέον υπό την άμεση ομπρέλα της CNCF, διαθέτει ενεργή ομάδα συντήρησης και τακτικά releases. Υποστηρίζεται από κοινότητα ανοιχτού λογισμικού και από εταιρικούς συντελεστές (CloudBees, Google). Η τεκμηρίωση και τα κανάλια (Slack, GitHub) είναι επαρκώς ενημερωμένα και παρέχουν αξιόπιστη κοινότητα υποστήριξης

Το Argo Workflows βαθμολογήθηκε με 4 καθώς υποστηρίζεται επίσης από την CNCF και μοιράζεται κοινό οικοσύστημα και contributors με το Argo CD [6]. Η κοινότητα είναι πολύ δραστήρια, με εκατοντάδες συνεισφέροντες, τακτικά releases, ολοκληρωμένη τεκμηρίωση και επίσημα κανάλια υποστήριξης (Argo Slack, GitHub). Παρότι η εμπορική υποστήριξη είναι περιορισμένη σε σχέση με το Argo CD, το project παρουσιάζει σταθερή ανάπτυξη και ευρεία χρήση σε οργανισμούς παραγωγής.

Το Drone αξιολογήθηκε με 3 καθώς διαθέτει μέτρια κοινότητα, με ικανοποιητικό αριθμό χρηστών αλλά περιορισμένο ρυθμό ανάπτυξης και μικρότερη παρουσία στο οικοσύστημα CNCF [7]. Η υποστήριξη παρέχεται κυρίως από το GitHub community, με λίγους ενεργούς maintainers και αραιά releases. Δεν υπάρχει επίσημη εμπορική υποστήριξη ή θεσμική ομπρέλα, γεγονός που μειώνει το επίπεδο υποστήριξης.

Το GoCD αφιολογήθηκε με 2 καθώσε υποστηρίζεται από μια μικρή κοινότητα και λίγους maintainers [8]. Τα releases είναι σποραδικά, ενώ η ανάπτυξη έχει επιβραδυνθεί τα τελευταία χρόνια. Παρότι υπάρχουν διαθέσιμα κανάλια επικοινωνίας, η τεκμηρίωση δεν ενημερώνεται συχνά και δεν υπάρχει επίσημη εμπορική υποστήριξη.

Πίνακας 24: Συγκριτική αξιολόγηση εργαλείων στην κατηγορία CI/CD

	ΕΡΓΑΛΕΙΟ				
ΚΡΙΤΗΡΙΟ	**Argo CD**	**Jenkins X**	**Argo Workflows**	**Drone**	**GoCD**
Απόδοση (Performance)	**4**	**3**	**4**	**3**	**3**

Ευκολία Ενσωμάτωσης	**3**	**3**	**3**	**4**	**3**
Επεκτασιμότητα	**4**	**4**	**4**	**3**	**3**
Ασφάλεια	**4**	**3**	**4**	**2**	**2**
Διαλειτουργικότητα	**5**	**4**	**4**	**3**	**3**
Υποστήριξη & Κοινότητα	**5**	**4**	**4**	**3**	**2**

Πηγές

[1] (Ramadoni, Utami, & Fatta, 2021)

[2] (Reddy J, Padal S, Mayan, Catharine A., & Shanthamalar, 2024)

[3] (Cloud Native Computing Foundation (CNCF), 2024) – Argo CD

[4] (Gupta, Bhatia, Memoria, & Manani, 2022)

[5] (Jenkins User Documentation, n.d.)

[6] (Argo Workflows — Documentation, 2023)

[7] (Drone CI / CD – Documentation, n.d.)

[8] (GoCD User Documentation, n.d.)

Η κατηγορία CI/CD περιλαμβάνει εργαλεία που αυτοματοποιούν τη συνεχή ολοκλήρωση και παράδοση εφαρμογών στο οικοσύστημα του Kubernetes. Από την αξιολόγηση των εργαλείων Argo CD, Jenkins X, Argo Workflows, Drone και GoCD προκύπτουν ουσιώδεις διαφοροποιήσεις ως προς την απόδοση, την ευκολία ενσωμάτωσης και την επεκτασιμότητα.

Το Argo CD αναδείχθηκε ως το πιο ολοκληρωμένο και ισορροπημένο εργαλείο, συνδυάζοντας υψηλή απόδοση, ισχυρή ασφάλεια και εκτενή διαλειτουργικότητα με τα περισσότερα υποσυστήματα του CNCF οικοσυστήματος (Helm, Prometheus, Vault, OPA). Η αρχιτεκτονική GitOps που υιοθετεί επιτρέπει εγγενή ενσωμάτωση με το Kubernetes API, ενώ η ενεργή κοινότητα και η συνεπής υποστήριξη (CNCF Graduated Project) διασφαλίζουν τη μακροπρόθεσμη βιωσιμότητά του.

Το Argo Workflows κατέγραψε επίσης πολύ υψηλή απόδοση και επεκτασιμότητα, ιδίως σε περιβάλλοντα που απαιτούν παράλληλη εκτέλεση εργασιών και σύνθετες ροές δεδομένων. Η πλήρης συμβατότητα με CRDs και το modular design το καθιστούν κατάλληλο για advanced DevOps και data-intensive pipelines, αν και η αρχική του παραμετροποίηση απαιτεί αυξημένη εξοικείωση με YAML templates και Kubernetes concepts.

Το Jenkins X παρουσίασε καλή συνολική επίδοση, με ιδιαίτερη έμφαση στην επεκτασιμότητα μέσω Tekton pipelines και integration APIs. Ωστόσο, η πολυπλοκότητα ρύθμισης και η εξάρτηση από CLI εργαλεία μειώνουν τη συνολική ευκολία ενσωμάτωσης, παρά την ενεργή του κοινότητα.

Το Drone διακρίνεται για την απλότητα και την ταχύτητα ενσωμάτωσης, αλλά η λειτουργικότητά του είναι περιορισμένη σε σχέση με τα υπόλοιπα εργαλεία, ιδιαίτερα ως προς την ασφάλεια και τη διαλειτουργικότητα. Αντίστοιχα, το GoCD εμφανίζει σταθερή αλλά παρωχημένη αρχιτεκτονική, με περιορισμένη ανάπτυξη κοινότητας και ελλιπή υποστήριξη σύγχρονων Kubernetes-native προτύπων.

Συνολικά, το οικοσύστημα Argo (Argo CD και Argo Workflows) υπερέχει καθαρά σε όλα σχεδόν τα κριτήρια — απόδοση, επεκτασιμότητα, ασφάλεια και κοινότητα — και αντιπροσωπεύει την πιο ώριμη και τεχνικά ολοκληρωμένη προσέγγιση για CI/CD σε περιβάλλοντα cloud-native και GitOps. Αντιθέτως, τα Drone και GoCD αποτελούν λύσεις με περιορισμένη προσαρμοστικότητα και μικρότερο ρυθμό εξέλιξης, κατάλληλες κυρίως για μικρότερες ή απλούστερες υποδομές.

Στην κατηγορία CI/CD, το κριτήριο με τη βέλτιστη απόδοση είναι η Διαλειτουργικότητα, με το Argo CD να υπερέχει (βαθμολογία 5) χάρη στην εγγενή υποστήριξη GitOps και τη δυνατότητα ενσωμάτωσης με πλήθος εργαλείων. Αντίθετα, τα ασθενέστερα κριτήρια είναι η Ασφάλεια και, σε μικρότερο βαθμό, η Απόδοση, με τα εργαλεία Drone και GoCD να παρουσιάζουν χαμηλές βαθμολογίες (2–3) λόγω περιορισμένης διαχείρισης ταυτοποίησης (authentication), ελλιπούς υποστήριξης RBAC, και δυσκολιών κλιμάκωσης σε μεγάλα clusters.

5.2.6 Cluster Management

Στην ενότητα αυτή αναλύονται τα αποτελέσματα της συγκριτικής αξιολόγησης των εργαλείων **kubectl**, **K9s**, **Portainer**, **K8sGPT** και **SUSE Rancher**, βάσει των κριτηρίων που ορίστηκαν στην Ενότητα 5.1. Η ανάλυση τεκμηριώνεται με βάση τα επιστημονικά άρθρα [1]–[5], το CNCF Landscape [6] και την επίσημη τεκμηρίωση των εργαλείων [7].

Απόδοση (Performance)

Το kubectl παρουσιάζει υψηλή απόδοση και βαθμολογήθηκε με 4, καθώς επικοινωνεί απευθείας με το Kubernetes API χωρίς ενδιάμεσο layer, προσφέροντας άμεση εκτέλεση εντολών και ελάχιστο latency [6][7]. Η εκτέλεση παραμένει ταχύτατη ακόμη και σε μεγάλα clusters, ενώ η κατανάλωση πόρων είναι αμελητέα. Μικρές καθυστερήσεις παρατηρούνται μόνο σε πολύπλοκες εντολές ή σε περιπτώσεις μεγάλης ταυτόχρονης πρόσβασης.

Το K9s αξιολογηθηκε με 4 καθώς προσφέρει υψηλή απόδοση και άμεση αλληλεπίδραση με το cluster μέσω TUI (terminal-based UI), αξιοποιώντας αποδοτικά το API του Kubernetes [7]. Σε μεγάλα clusters ενδέχεται να υπάρχει ελαφρά μείωση ταχύτητας στην ανανέωση δεδομένων, χωρίς όμως να επηρεάζεται ουσιαστικά η εμπειρία χρήσης ή η σταθερότητα.

Το Portainer, ως web-based UI εργαλείο, προσφέρει ικανοποιητική ανταπόκριση σε μεσαία και μεγάλα clusters, αλλά η ταχύτητα εξαρτάται από την υποκείμενη υποδομή (server resources και

δίκτυο) [6][7]. Σε ιδιαίτερα φορτωμένα ή κατανεμημένα περιβάλλοντα, μπορεί να παρατηρηθεί αυξημένο latency στη φόρτωση δεδομένων λόγω UI rendering και API polling. Για το λόγο αυτό βαθμολογήθηκε με 3.

Το K8sGPT αξιολογείται με υψηλή απόδοση και βαθμολογείται με 4, καθώς πραγματοποιεί AI-based ανάλυση προβλημάτων γρήγορα και αποδοτικά [2]. Η επεξεργασία βασίζεται σε τοπική ή cloud AI υποδομή και δεν επιβαρύνει το cluster. Μικρή καθυστέρηση μπορεί να παρατηρηθεί κατά την αρχική συλλογή logs ή σε πολύ μεγάλα περιβάλλοντα, αλλά συνολικά η εκτέλεση παραμένει ταχεία.

Το SUSE Rancher παρουσιάζει ικανοποιητική απόδοση για διαχείριση multi-cluster περιβαλλόντων, αλλά εισάγει ορισμένο λειτουργικό overhead λόγω των πολλαπλών abstraction layers που χρησιμοποιεί [4][6]. Το overhead είναι μέτριο, επηρεάζοντας κυρίως το latency κατά την απεικόνιση και τη διαχείριση πολύ μεγάλων clusters (>100 nodes). Συνεπώς, το εργαλείο αυτό αξιολογείται με 3.

Ευκολία Ενσωμάτωσης

Το kubectl αξιολογήθηκε με 5 καθώς αποτελεί native εργαλείο του Kubernetes, προεγκατεστημένο ή διαθέσιμο ως μέρος του επίσημου API. Δεν απαιτεί επιπλέον ενσωμάτωση, καθώς επικοινωνεί απευθείας με τον API server. Η ρύθμιση σύνδεσης με το cluster γίνεται αυτόματα μέσω του αρχείου kubeconfig.

Το K9s αξιολογήθηκε με 5 καθώς ενσωματώνεται εύκολα, λειτουργεί πάνω από το kubectl και χρησιμοποιεί το υπάρχον αρχείο kubeconfig για πρόσβαση στο cluster [7]. Η εγκατάσταση γίνεται μέσω ενός απλού binary ή package manager και δεν απαιτεί πρόσθετα CRDs ή ρυθμίσεις.

Το Portainer παρέχει έτοιμα Helm charts και επίσημο Operator για εγκατάσταση στο Kubernetes [6][7]. Αξιολογείται με 4 διότι η βασική ενσωμάτωση είναι απλή και αυτοματοποιημένη με έτοιμους μηχανισμούς εγκατάστασης αλλά για πιο σύνθετα σενάρια (π.χ. multi-node, external auth) απαιτείται πρόσθετη ρύθμιση.

Το K8sGPT εγκαθίσταται εύκολα μέσω Helm ή binary, αλλά απαιτεί πρόσθετη διαμόρφωση πρόσβασης στο API server και ενδεχομένως API tokens για σωστή λειτουργία [2]. Αν και η τεκμηρίωση είναι επαρκής, η αρχική ρύθμιση μπορεί να χρειαστεί χειροκίνητη προσαρμογή (π.χ. για χρήση εξωτερικών clusters ή LLM providers). Συνεπώς, το εργαλείο αξιολογείται με 3.

Το SUSE Rancher, αν και διαθέτει Helm chart και installer, παρουσιάζει αυξημένη πολυπλοκότητα κατά την ενσωμάτωση λόγω της αρχιτεκτονικής του (multi-cluster management, RBAC, certificates, external ingress) [4][6]. Η εγκατάσταση απαιτεί αρκετά βήματα παραμετροποίησης και σωστό σχεδιασμό για σταθερή λειτουργία σε παραγωγικά περιβάλλοντα. Αξιολογήθηκε με 2 καθώς απαιτεί πολλαπλές χειροκίνητες ρυθμίσεις και καλή γνώση Kubernetes για επιτυχή εγκατάσταση.

Επεκτασιμότητα

Το kubectl είναι βασικό CLI εργαλείο του Kubernetes, με σταθερή και προκαθορισμένη λειτουργικότητα. Αξιολογήθηκε με 1 καθώς δεν υποστηρίζει επίσημους μηχανισμούς επέκτασης ή προσθήκης plugins μέσω API, παρά μόνο scripting σε ανώτερο επίπεδο (π.χ. μέσω Bash ή wrappers).

Το K9s παρέχει περιορισμένη επεκτασιμότητα και αξιολογήθηκε με 2, καθώς επιτρέπει βασική παραμετροποίηση μέσω configuration files και custom views [7]. Ωστόσο, δεν διαθέτει επίσημο API ή plugin system για βαθιά ενσωμάτωση με άλλα εργαλεία ή υπηρεσίες.

Το Portainer υποστηρίζει ορισμένες ρυθμίσεις παραμέτρων μέσω του web interface και των configuration files, αλλά δεν προσφέρει μηχανισμό επεκτάσεων, όπως custom modules ή API hooks [6][7]. Η ενσωμάτωση με εξωτερικά συστήματα γίνεται μέσω βασικών REST APIs, χωρίς δυνατότητα επέκτασης της ίδιας της λειτουργικότητας και για αυτό αξιολογείται με 3.

Το K8sGPT παρουσιάζει μέτρια επεκτασιμότητα (3), καθώς υποστηρίζει προσαρμογή μέσω plugins (providers) και δυνατότητα ενσωμάτωσης με εξωτερικά LLMs ή AI services [2]. Η προσθήκη νέων providers είναι εφικτή μέσω τεκμηριωμένου configuration και επεκτάσιμου API, αλλά χωρίς πλήρως modular plugin framework.

Το SUSE Rancher διαθέτει πολύ καλή επεκτασιμότητα και βαθμολογείται με 4, καθώς παρέχει τεκμηριωμένο API, integration points και δυνατότητα ενσωμάτωσης με custom controllers, monitoring (Prometheus, Grafana), policy frameworks (OPA, Kyverno) και CI/CD εργαλεία [4][6]. Η modular αρχιτεκτονική του επιτρέπει την ανάπτυξη custom integrations και επεκτάσεων για εξωτερικά συστήματα. Ωστόσο, δεν διαθέτει πλήρες plugin framework ή SDK για τη δημιουργία third-party modules οπότε αυτό δικαιολογεί τη βαθμολογία 4 («Καλή επεκτασιμότητα σε μεγάλα clusters»).

Ασφάλεια

Το kubectl αξιολογήθηκε με 3 καθώς βασίζεται εξ'ολοκλήρου στους μηχανισμούς ασφαλείας του Kubernetes, αξιοποιώντας RBAC, Service Accounts και TLS authentication για ασφαλή επικοινωνία με το API server. Δεν προσφέρει, ωστόσο, πρόσθετους μηχανισμούς auditing ή policy enforcement, ενώ η ασφάλειά του εξαρτάται αποκλειστικά από τη σωστή ρύθμιση του cluster.

Το K9s χρησιμοποιεί τα ίδια tokens και δικαιώματα πρόσβασης που ορίζονται από το RBAC του Kubernetes, χωρίς να εισάγει νέα σημεία πρόσβασης ή εξωτερικές υπηρεσίες. Διασφαλίζει ασφαλή λειτουργία μέσω των ίδιων πιστοποιήσεων (TLS), ωστόσο, όπως και το kubectl, δεν προσθέτει επιπλέον επίπεδα προστασίας ή auditing, οπότε αξιολογήθηκε και αυτό με 3.

Το Portainer ενσωματώνει μηχανισμούς αυθεντικοποίησης και SSL/TLS, παρέχοντας βασική προστασία για τη διαχείριση clusters μέσω web interface και για το λόγο αυτό βαθμολογήθηκε με 2 . Το UI μπορεί να αποτελέσει επιφάνεια επίθεσης αν δεν υλοποιηθούν σωστά μέτρα hardening (όπως χρήση reverse proxy ή περιορισμός δημόσιας πρόσβασης). Σε ασφαλείς ρυθμίσεις, το Portainer φτάνει το επίπεδο "ικανοποιητικής" προστασίας, αλλά σε απλή ή λανθασμένη διαμόρφωση, παραμένει σε "βασικό" επίπεδο.

Το K8sGPT χρησιμοποιεί τους ίδιους μηχανισμούς πιστοποίησης του Kubernetes (RBAC, API tokens) και λειτουργεί εντός του cluster χωρίς ανεξάρτητα endpoints, μειώνοντας έτσι τις πιθανότητες μη εξουσιοδοτημένης πρόσβασης. Παρά ταύτα, η ενσωμάτωση με εξωτερικά AI APIs μπορεί να δημιουργήσει κίνδυνο διαρροής δεδομένων αν δεν εφαρμοστεί κατάλληλη κρυπτογράφηση. Για αυτό αξιολογείται με 3.

Το SUSE Rancher παρέχει ολοκληρωμένη υποστήριξη πολιτικών ασφαλείας με πλήρη ενσωμάτωση RBAC, TLS/mTLS επικοινωνίας, auditing, και policy enforcement frameworks, όπως OPA και Kyverno. Διαθέτει μηχανισμούς για πολυεπίπεδο έλεγχο πρόσβασης και ενσωμάτωση με vulnerability scanners. Ωστόσο, η πολυπλοκότητα των ρυθμίσεων μπορεί να οδηγήσει σε misconfigurations που μειώνουν την αποτελεσματικότητα των μέτρων, ενώ δεν εφαρμόζει πλήρως zero-trust ή πιστοποιημένη συμμόρφωση. Αξιολογείται με 4 καθώς παρέχει ισχυρή και ώριμη ασφάλεια με μικρά κενά σε αυτοματοποίηση και πιστοποιημένη συμμόρφωση.

Διαλειτουργικότητα

Το kubectl επικοινωνεί αποκλειστικά με το Kubernetes API. Δεν υποστηρίζει native integration με άλλα συστήματα (monitoring, CI/CD, observability), ούτε επεκτείνεται μέσω plugins για συνεργασία με εξωτερικά εργαλεία. Παρέχει ελάχιστη διαλειτουργικότητα καθώς επικοινωνεί μόνο με το Kubernetes API. Συνεπώς, αξιολογείται με 1.

Το K9s αξιολογήθηκε με 2 ως προς τη διαλειτουργικότητα, καθώς υποστηρίζει βασικές μόνο διασυνδέσεις με το Kubernetes API και monitoring συστήματα όπως το Prometheus, κυρίως μέσω προβολής metrics ή logs. Δεν προσφέρει αμφίδρομη συνεργασία ή native integrations με CI/CD, policy ή observability πλαίσια. Συνεπώς, οι διασυνδέσεις περιορίζονται σε 1–2 βασικές κατηγορίες εργαλείων, στοιχείο που το κατατάσσει στο επίπεδο “περιορισμένη διαλειτουργικότητα” (2).

Το Portainer επιτρέπει τη διαχείριση clusters και Helm releases, ωστόσο δεν προσφέρει εκτεταμένη ή αυτόματη συνεργασία με εργαλεία παρακολούθησης (Prometheus, Grafana), πολιτικών (OPA, Kyverno) ή αυτοματοποίησης (Argo CD, Jenkins X). Η επικοινωνία του περιορίζεται κυρίως στο επίπεδο του Kubernetes API οπότε βαθμολογείται με 2.

Το K8sGPT αξιολογήθηκε με 2 ως προς τη διαλειτουργικότητα, καθώς διαθέτει περιορισμένες διασυνδέσεις, κυρίως με εξωτερικά AI APIs (όπως OpenAI, Azure OpenAI και Anthropic). Η συνεργασία του με εγγενή CNCF components (όπως Prometheus, Argo CD ή Grafana) είναι ανύπαρκτη ή ελάχιστη, ενώ η λειτουργικότητα των ενσωματώσεων περιορίζεται στην ανάλυση Kubernetes resources και στην απόδοση επεξηγήσεων μέσω LLMs. Επομένως, οι διασυνδέσεις του περιορίζονται σε 1–2 εξωτερικές υπηρεσίες με βασική λειτουργικότητα, γεγονός που το κατατάσσει στο επίπεδο 2 («περιορισμένη διαλειτουργικότητα»)

Το SUSE Rancher επιδεικνύει υψηλή διαλειτουργικότητα, καθώς συνεργάζεται με πλήθος εργαλείων και υπηρεσιών του οικοσυστήματος Kubernetes. Υποστηρίζει ενσωματώσεις με Prometheus και Grafana για παρακολούθηση, OPA/Kyverno για πολιτικές ασφαλείας, καθώς και CI/CD pipelines μέσω Argo CD ή Jenkins X. Επιπλέον, εκθέτει CRDs και APIs για την επικοινωνία με εξωτερικά συστήματα και multi-cluster περιβάλλοντα. Για αυτό και βαθμολογείται με 4.

Υποστήριξη & Κοινότητα

Το kubectl αξιολογήθηκε με 5 καθώς αποτελεί το επίσημο εργαλείο CLI του Kubernetes, συντηρούμενο από την CNCF και την κοινότητα Kubernetes. Διαθέτει χιλιάδες ενεργούς συνεισφέροντες, συχνές ενημερώσεις σε κάθε νέα έκδοση του Kubernetes, πλήρη τεκμηρίωση, καθώς και επαγγελματική υποστήριξη από εταιρείες όπως Google, Red Hat και SUSE.

Το K9s αξιολογήθηκε με 3 καθώς διαθέτει ενεργή αλλά μικρότερη κοινότητα, με περίπου 10–30 συνεισφέροντες και τακτική δραστηριότητα στο GitHub. Οι ενημερώσεις είναι τακτικές και υπάρχει ανταπόκριση σε issues, ωστόσο δεν παρέχεται επίσημη ή εμπορική υποστήριξη.

Το Portainer αξιολογήθηκε με 4 καθώς διαθέτει πολύ μεγάλη και ενεργή κοινότητα με εκατοντάδες contributors και συνεχή ανάπτυξη. Προσφέρει εμπορική υποστήριξη μέσω της Portainer.io, καθώς και εκτενή τεκμηρίωση και τακτικά releases με security patches.

Το K8sGPT είναι σχετικά νέο έργο και αξιολογήθηκε με 2 καθως διαθέτει μικρή κοινότητα και λιγότερους από 10 συνεισφέροντες. Οι ενημερώσεις είναι περιοδικές και η υποστήριξη περιορίζεται σε GitHub discussions χωρίς επίσημο οργανισμό ή εμπορικό πλαίσιο υποστήριξης.

Το SUSE Rancher υποστηρίζεται ενεργά από τη SUSE και την κοινότητα του Rancher Labs, με χιλιάδες συνεισφέροντες και συχνά security updates οπότε και αξιολογήθηκε με 5. Διαθέτει εκτενή τεκμηρίωση, επαγγελματική εμπορική υποστήριξη, πιστοποιημένα εκπαιδευτικά προγράμματα και οδικό χάρτη ανάπτυξης.

Πίνακας 25: Συγκριτική αξιολόγηση εργαλείων στην κατηγορία CLI

	Εργαλείο				
Κριτήριο	**kubectl**	**K9s**	**Portainer**	**K8sGPT**	**SUSE Rancher**
Απόδοση (Performance)	**4**	**4**	**3**	**4**	**3**
Ευκολία Ενσωμάτωσης	**5**	**5**	**4**	**3**	**2**
Επεκτασιμότητα	**1**	**2**	**3**	**3**	**4**
Ασφάλεια	**3**	**3**	**2**	**3**	**4**
Διαλειτουργικότητα	**1**	**2**	**2**	**2**	**4**
Υποστήριξη & Κοινότητα	**5**	**3**	**4**	**2**	**5**

Πηγές

[1] (Wennerström, 2021)

[2] (Puutio, 2025)

[3] (Poniszewska-Marańda & Czechowska, 2021)

[4] (Syrigos, Makris, & Korakis, 2023)

[5] (Yu, Naganuma, & Ide, 2020)

[6] (Cloud Native Computing Foundation (CNCF), 2024) – Cluster Management category

[7] (Kubernetes Documentation, 2024)

Η κατηγορία Cluster Management & Administration αξιολογεί εργαλεία που υποστηρίζουν τη διαχείριση, παρακολούθηση και βελτιστοποίηση Kubernetes clusters. Από την ανάλυση των εργαλείων kubectl, K9s, Portainer, K8sGPT και SUSE Rancher, προκύπτουν σαφείς διαφοροποιήσεις ως προς την απόδοση, την επεκτασιμότητα και την ασφάλεια.

Το SUSE Rancher αναδείχθηκε ως το πιο ολοκληρωμένο και ώριμο εργαλείο της κατηγορίας, συνδυάζοντας υψηλή επεκτασιμότητα, ανεπτυγμένους μηχανισμούς ασφάλειας και εξαιρετική διαλειτουργικότητα με το ευρύτερο οικοσύστημα Kubernetes. Η αρχιτεκτονική του επιτρέπει τη διαχείριση multi-cluster περιβαλλόντων με υποστήριξη πολιτικών (OPA, Kyverno), παρακολούθησης (Prometheus, Grafana) και CI/CD ενοποίησης (Argo CD, Jenkins X). Αν και απαιτεί μεγαλύτερη πολυπλοκότητα κατά την εγκατάσταση, η λειτουργικότητά του και η εμπορική υποστήριξη της SUSE το καθιστούν την πιο πλήρη λύση για enterprise χρήσεις.

Το kubectl, ως εγγενές εργαλείο του Kubernetes, καταγράφει τη μέγιστη απόδοση και αξιοπιστία, αλλά περιορίζεται σημαντικά σε επεκτασιμότητα και διαλειτουργικότητα, καθώς δεν υποστηρίζει μηχανισμούς επέκτασης ή οπτικοποίησης. Παραμένει ωστόσο το πιο σταθερό και απαραίτητο εργαλείο βάσης για διαχειριστές, προσφέροντας άμεση και ασφαλή πρόσβαση στο API server.

Το K9s λειτουργεί ως ελαφριά και αποδοτική διεπαφή πάνω από το kubectl, βελτιώνοντας τη χρηστικότητα και τη διαχείριση μέσω TUI περιβάλλοντος. Παρέχει καλή απόδοση και ευκολία ενσωμάτωσης, αλλά παρουσιάζει περιορισμένη επεκτασιμότητα και βασική ασφάλεια, αφού στηρίζεται αποκλειστικά στους μηχανισμούς RBAC του Kubernetes.

Το Portainer προσφέρει σημαντική ευκολία χρήσης και φιλικό περιβάλλον διαχείρισης μέσω web UI, όμως η απόδοσή του εξαρτάται από την υποδομή και εισάγει μικρό overhead. Παρότι διαθέτει καλή κοινότητα και τεκμηρίωση, η ασφάλειά του θεωρείται βασική, καθώς το web interface μπορεί να αποτελέσει δυνητικό σημείο ευπάθειας σε ανεπαρκώς ασφαλισμένες εγκαταστάσεις.

Το K8sGPT αποτελεί καινοτόμο προσέγγιση με AI-based troubleshooting και καλή απόδοση, αλλά βρίσκεται σε πρώιμο στάδιο ωρίμανσης, με περιορισμένη κοινότητα και υποστήριξη. Παρά την αξιοσημείωτη τεχνική του προοπτική, η εξάρτηση από εξωτερικά AI APIs δημιουργεί ανησυχίες ασφαλείας και συμμόρφωσης, καθιστώντας το προς το παρόν συμπληρωματικό εργαλείο μάλλον παρά κύριο σύστημα διαχείρισης.

Συνολικά, το SUSE Rancher υπερέχει ως η πιο ολοκληρωμένη λύση για ολοκληρωμένη διαχείριση Kubernetes clusters, ενώ το kubectl παραμένει το αποδοτικότερο και πιο αξιόπιστο εργαλείο πυρήνα. Τα K9s και Portainer τοποθετούνται ως πρακτικές επιλογές για απλοποιημένη διαχείριση, ενώ το K8sGPT εκπροσωπεί μια αναδυόμενη, καινοτόμα αλλά ακόμη ανώριμη τεχνολογική κατεύθυνση.

Στην κατηγορία CLI, το βέλτιστο κριτήριο είναι η Ευκολία Ενσωμάτωσης, καθώς τα περισσότερα εργαλεία (kubectl, K9s, Portainer) επιδεικνύουν υψηλή ευχρηστία και απλή παραμετροποίηση. Το ίδιο ισχύει για το κριτήριο Υποστήριξη & Κοινότητα. Αντίθετα, το χειρότερο κριτήριο είναι η Διαλειτουργικότητα, όπου τα εργαλεία παρουσιάζουν περιορισμένες δυνατότητες διασύνδεσης και συνεργασίας με άλλα συστήματα ή πλατφόρμες, απαιτώντας πρόσθετες επεκτάσεις για πλήρη λειτουργικότητα**.**

5.2.7 Autoscaling

Η κατηγορία Autoscaling περιλαμβάνει εργαλεία και τεχνικές που επιτρέπουν την αυτόματη προσαρμογή των πόρων σε περιβάλλοντα Kubernetes, με στόχο τη βέλτιστη απόδοση, την αποφυγή υπερ-προμήθειας και τη μείωση του κόστους. Τα εργαλεία που εξετάζονται – HPA (Tuned), KOSMOS, και KEDA – αξιολογήθηκαν με βάση τα κριτήρια που ορίστηκαν στην Ενότητα 5.1.

Η αξιολόγηση βασίστηκε σε επιστημονικά άρθρα που αναλύουν μετρήσεις και σενάρια χρήσης, καθώς και σε επίσημη τεκμηρίωση και δεδομένα από το CNCF Landscape, με στόχο την αποτύπωση ρεαλιστικών τιμών στην κλίμακα 1–5.

Απόδοση (Performance)

Το HPA παρέχει σταθερή και σχετικά υψηλή απόδοση (3) σε τυπικά σενάρια, ωστόσο βασίζεται αποκλειστικά σε CPU και memory μετρικές, γεγονός που περιορίζει την αντίδρασή του σε σύνθετες ή μη γραμμικές μεταβολές φόρτου (π.χ. spikes I/O ή latency). Σε αιφνίδιες αυξήσεις του φόρτου, η καθυστέρηση προσαρμογής μπορεί να φτάσει μερικά δευτερόλεπτα, οδηγώντας σε παροδική υπο-απόδοση.

Το KOSMOS παρουσιάζει πολύ καλή απόδοση (3) καθώς μπορεί να εκτελεί ταυτόχρονα οριζόντια και κάθετη κλιμάκωση, επιτυγχάνοντας ταχύτερη προσαρμογή πόρων. Παρ' όλα αυτά, η ταυτόχρονη διαχείριση πολλών μετρικών επιβαρύνει τον cluster controller σε μεγάλα workloads, αυξάνοντας τη χρήση CPU στο control plane κατά ~10–15%.

Το KEDA [2] επιτυγχάνει υψηλή απόδοση (4) σε event-driven workloads, αξιοποιώντας external scalers (Kafka, RabbitMQ, Prometheus) που ενεργοποιούν δυναμικά pods με ελάχιστο latency. Ωστόσο, σε πιο παραδοσιακά CPU-bound σενάρια, η απόδοσή του μειώνεται λόγω καθυστερήσεων 2–3s στην ανίχνευση και απόκριση γεγονότων. Συνολικά, διατηρεί υψηλή αποτελεσματικότητα στα περιβάλλοντα για τα οποία σχεδιάστηκε.

Ευκολία Ενσωμάτωσης

Το HPA αποτελεί native μηχανισμό του Kubernetes, αλλά η "tuned" εκδοχή του απαιτεί χειροκίνητη προσαρμογή thresholds, metrics και scaling policies ανά εφαρμογή. Η βασική

εγκατάσταση είναι άμεση (ενσωματωμένη στο control plane), ωστόσο η παραμετροποίηση απαιτεί καλή γνώση resource utilization patterns και YAML configuration. Επομένως, βαθμολογείται με 3.

Το KOSMOS δεν αποτελεί τμήμα του Kubernetes, αλλά επεκτείνει υπάρχοντες autoscalers (όπως το HPA) μέσω custom controllers. Βαθμολογήθηκε με 2 καθώς η ενσωμάτωση απαιτεί εγκατάσταση CRDs, προσαρμογή υφιστάμενων autoscaling policies, και επαλήθευση συμβατότητας με metrics adapters. Παρότι υπάρχουν διαθέσιμα manifests, η διαδικασία δεν είναι πλήρως αυτοματοποιημένη και απαιτεί εμπειρία σε μηχανισμούς κλιμάκωσης (scaling mechanisms) και RBAC ρυθμίσεις.

Το KEDA αξιολογήθηκε με 3 καθώς προσφέρει Helm chart και Operator, επιτρέποντας σχετικά απλή εγκατάσταση και αυτόματη διασύνδεση με το Kubernetes API. Ωστόσο, η ενσωμάτωση απαιτεί γνώση για event sources και external scalers, καθώς η λανθασμένη ρύθμιση μπορεί να εμποδίσει την αυτόματη κλιμάκωση.

Επεκτασιμότητα

Η "tuned" εκδοχή του HPA αξιολογήθηκε με 3 καθώς υποστηρίζει επεκτασιμότητα μέσω Custom Metrics API και External Metrics API, επιτρέποντας τη χρήση μετρικών πέρα από CPU και memory (π.χ. latency, queue length). Ωστόσο, η επέκταση προϋποθέτει πρόσθετη ρύθμιση adapters (όπως Prometheus Adapter) και όχι πλήρως modular αρχιτεκτονική.

Το KOSMOS αξιολογήθηκε με 3 καθώς επεκτείνει τη λειτουργικότητα του HPA μέσω custom controllers και CRDs, επιτρέποντας συνδυασμένη κάθετη και οριζόντια κλιμάκωση.Υποστηρίζει integration hooks με metrics pipelines και custom schedulers, ωστόσο η τεκμηρίωση είναι περιορισμένη και η ανάπτυξη custom policies απαιτεί γνώση του εσωτερικού control loop-μηχανισμός αυτόματης προσαρμογής που εκτελεί ο controller.

Το KEDA αξιολογήθηκε με 5 καθώς διαθέτει εξαιρετικά υψηλή επεκτασιμότητα, βασισμένη σε modular αρχιτεκτονική με Scaler plugins. Υποστηρίζει πάνω από 50 external scalers (Kafka, RabbitMQ, AWS SQS, Prometheus, Azure Monitor κ.ά.) και παρέχει SDK για ανάπτυξη custom scalers από τρίτους. Επιπλέον, διαθέτει τεκμηριωμένο API, Operator, και ενεργή κοινότητα συντηρητών που συνεισφέρει νέους scalers

Ασφάλεια (Security)

Το HPA λειτουργεί εντός του control plane και κληρονομεί πλήρως θεμελιώδεις μηχανισμούς ασφάλειας του Kubernetes, όπως RBAC, API authentication, και service account isolation. Δεν διαθέτει ανεξάρτητους μηχανισμούς ασφάλειας, αλλά αξιοποιεί εγγενώς τους μηχανισμούς πρόσβασης του API server. Δεν υποστηρίζει auditing ή policy enforcement σε επίπεδο autoscaling, συνεπώς η προστασία περιορίζεται στο βασικό πλαίσιο του Kubernetes οπότε και αξιολόγείται με 3.

Το KOSMOS αξιοποιεί τους ενσωματωμένους μηχανισμούς ασφάλειας του Kubernetes, όπως το RBAC και τα service accounts, για τον έλεγχο πρόσβασης. Ωστόσο, η χρήση custom controllers και CRDs απαιτεί επιπλέον δικαιώματα σε επίπεδο συστάδας, γεγονός που μπορεί να αυξήσει την επιφάνεια επίθεσης σε περίπτωση εσφαλμένης ρύθμισης. Παρότι δεν παρακάμπτει τους εγγενείς

μηχανισμούς του Kubernetes, δεν διαθέτει ενσωματωμένους ελέγχους auditing ή επικύρωσης, με αποτέλεσμα η συνολική ασφάλεια να παραμένει σε βασικό επίπεδο και να αξιολογείται με 2.

Το KEDA αξιοποιεί τα RBAC και Secrets του Kubernetes για την πρόσβαση σε external event sources, ενώ οι scalers λειτουργούν μέσω ασφαλών connection strings ή tokens. Ωστόσο, η ασφάλεια εξαρτάται σε μεγάλο βαθμό από το underlying event/messaging σύστημα (π.χ. Kafka, RabbitMQ, Azure Service Bus). Παρέχει τεκμηριωμένες οδηγίες για χρήση Kubernetes Secrets και περιορισμένα service accounts, αλλά δεν εφαρμόζει native encryption ή policy enforcement, οπότε αξιολογείται με 3.

Διαλειτουργικότητα (Interoperability)

Το Tuned HPA προσφέρει σαφώς βελτιωμένη διαλειτουργικότητα σε σχέση με τη βασική εκδοχή του, καθώς μπορεί να αντλεί μετρικές όχι μόνο από τον Metrics Server, αλλά και από εξωτερικά συστήματα παρακολούθησης (όπως ο Prometheus) μέσω Custom Metrics API ή Prometheus Adapter. Ωστόσο, η διασύνδεση αυτή δεν είναι εγγενής στο Kubernetes και απαιτεί πρόσθετη εγκατάσταση και ρύθμιση adapters ή exporters, γεγονός που περιορίζει τη συνολική ενσωμάτωση. Συνεπώς, η διαλειτουργικότητα του HPA μπορεί να αξιολογηθεί σε εύρος 2–3, ανάλογα με την υλοποίηση: πιο περιορισμένη (2) στη βασική εκδοχή και μέτρια (3) στη βελτιστοποιημένη, tuned εκδοχή.Το KOSMOS αξιολογήθηκε με 3 καθώς υποστηρίζει συνεργασία με συστήματα, όπως ο Prometheus ή custom metrics exporters, αξιοποιώντας CRDs και Webhooks για τη συλλογή δεδομένων. Ωστόσο, η διασύνδεση με τα εξωτερικά αυτά εργαλεία δεν είναι αυτοματοποιημένη, αλλά απαιτεί χειροκίνητη παραμετροποίηση και προσαρμογή CRDs. Συνεπώς, το KOSMOS εμφανίζει μέτρια διαλειτουργικότητα, καθώς μπορεί να συνεργάζεται με πολλά εργαλεία, αλλά όχι με πλήρως ενοποιημένο τρόπο.

Το KEDA επιδεικνύει υψηλή διαλειτουργικότητα, καθώς υποστηρίζει ένα ευρύ φάσμα εξωτερικών παρόχων και πλατφορμών — όπως AWS, Azure, GCP, Kafka, RabbitMQ, Prometheus, PostgreSQL κ.ά. — μέσω τυποποιημένων CRDs (ScaledObject, ScaledJob) και έτοιμων scalers (>50). Η συνεργασία με τα Kubernetes APIs είναι πλήρως αμφίδρομη και αυτοματοποιημένη, επιτρέποντας δυναμική κλιμάκωση βάσει εξωτερικών events. Έτσι, το KEDA παρέχει πλήρη και ενοποιημένη συνεργασία με τα περισσότερα υποσυστήματα του οικοσυστήματος CNCF και για αυτό αξιολογείται με 4.

Υποστήριξη & Κοινότητα (Support & Community)

Το HPA αποτελεί native μηχανισμό του Kubernetes, ενταγμένο στον πυρήνα του έργου και υποστηριζόμενο από το CNCF και την Google. Διαθέτει χιλιάδες contributors, συνεχή ενημέρωση μέσα από τα releases του Kubernetes, πλήρη τεκμηρίωση και ενεργή κοινότητα σε όλα τα κανάλια (Slack, GitHub, Discuss). Επιπλέον, υποστηρίζεται εμπορικά από μεγάλους cloud παρόχους (AWS, Azure, GCP). Συνεπώς, το HPA αξιολογείται με 5, καθώς διαθέτει τη μέγιστη δυνατή υποστήριξη και μακροπρόθεσμη βιωσιμότητα.

Το KOSMOS είναι ερευνητικό έργο, με περιορισμένο αριθμό συντηρητών και μικρή κοινότητα ανάπτυξης. Οι ενημερώσεις είναι σπάνιες, ενώ δεν υπάρχει επίσημη τεχνική ή εμπορική

υποστήριξη. Η επικοινωνία περιορίζεται σε ερευνητικά repositories και δεν παρέχεται ενεργή συνεισφορά από το CNCF ή μεγάλους οργανισμούς. Επομένως, αξιολογείται με 2, καθώς παρουσιάζει περιορισμένη υποστήριξη και κοινότητα

Το KEDA υποστηρίζεται από το CNCF (Graduated project) και έχει μια μεγάλη, ενεργή κοινότητα με εκατοντάδες contributors. Διαθέτει πλήρη τεκμηρίωση, τακτικές εκδόσεις και συχνή δραστηριότητα στα επίσημα κανάλια (GitHub, Slack, CNCF events). Παρότι η εμπορική υποστήριξη είναι περιορισμένη σε σχέση με native components όπως το HPA, η κοινότητα παραμένει δυναμική και αναπτυσσόμενη, δικαιολογώντας τον βαθμό αξιολόγησης 4.

Πίνακας 26: Συγκριτική αξιολόγηση εργαλείων στην κατηγορία Autoscaling

	Εργαλεία		
Κριτήριο	**HPA**	**KOSMOS**	**KEDA**
Απόδοση (Performance)	**3**	**3**	**4**
Ευκολία Ενσωμάτωσης	**3**	**2**	**3**
Επεκτασιμότητα	**3**	**3**	**5**
Ασφάλεια	**3**	**2**	**3**
Διαλειτουργικότητα	**2-3**	**3**	**4**
Υποστήριξη & Κοινότητα	**5**	**2**	**4**

Πηγές

[1] (Baresi, Hu, Quattrocchi, & Terracciano, 2021)

[2] (KEDA Project Documentation, 2024)

[3] (Guruge & Priyadarshana, 2025)

[4] (Menouer, Cérin, & Darmon, 2024)

[5] (Augustyn, Wyciślik, & Sojka, 2024)

[6] (Molleti, 2022)

Η κατηγορία Autoscaling περιλαμβάνει μηχανισμούς και επεκτάσεις που αυτοματοποιούν την προσαρμογή πόρων στο Kubernetes, με στόχο τη βελτιστοποίηση της απόδοσης και της αποδοτικότητας κόστους. Από την ανάλυση των εργαλείων HPA (Tuned), KEDA και KOSMOS,

προκύπτουν σημαντικές διαφοροποιήσεις ως προς την απόδοση, την επεκτασιμότητα και την ωριμότητα υποστήριξης.

Το KEDA αναδείχθηκε ως το πιο ολοκληρωμένο και ευέλικτο εργαλείο, συγκεντρώνοντας τις υψηλότερες βαθμολογίες σε επέκταση (5) και διαλειτουργικότητα (4), χάρη στη modular αρχιτεκτονική του και την υποστήριξη άνω των 50 external scalers (Kafka, RabbitMQ, Prometheus, Azure Monitor κ.ά.). Παρουσιάζει εξαιρετική απόδοση (4) σε event-driven workloads και επιτυγχάνει αμφίδρομη ενσωμάτωση με το Kubernetes API, ενώ η ενεργή κοινότητα του CNCF εξασφαλίζει μακροπρόθεσμη βιωσιμότητα. Για περιβάλλοντα που απαιτούν ευέλικτο και προσαρμόσιμο scaling, το KEDA αποτελεί τη βέλτιστη επιλογή.

Το HPA (Tuned) διατηρεί υψηλή αξιοπιστία (4) και σταθερή απόδοση (3), αποτελώντας τον πιο ώριμο και εγγενή μηχανισμό autoscaling του Kubernetes. Η πλήρης ενσωμάτωσή του στο control plane και η απλότητα εγκατάστασης (5) το καθιστούν την ασφαλέστερη και πιο σταθερή επιλογή για βασικά σενάρια χρήσης. Ωστόσο, η λειτουργία του περιορίζεται σε CPU και memory metrics, ενώ η επέκταση σε custom μετρικές απαιτεί πρόσθετη ρύθμιση adapters, γεγονός που μειώνει τη συνολική του ευελιξία. Συνεπώς, θεωρείται ιδανικό για σταθερά workloads, αλλά όχι για περιβάλλοντα με υψηλή μεταβλητότητα ή σύνθετες μετρικές.

Το KOSMOS παρουσιάζει καλή απόδοση (4) και την πιο προχωρημένη θεωρητικά προσέγγιση, καθώς συνδυάζει οριζόντια και κάθετη κλιμάκωση, ωστόσο η περιορισμένη υποστήριξη (2) και η χαμηλή ευκολία ενσωμάτωσης (2) περιορίζουν τη χρηστικότητά του σε ερευνητικά ή πειραματικά περιβάλλοντα. Παρά τις δυνατότητές του, η έλλειψη ευρείας υιοθέτησης και τεκμηρίωσης το καθιστά τη λιγότερο ώριμη λύση της κατηγορίας.

Συνολικά, η κατηγορία Autoscaling καταδεικνύει πως το KEDA υπερέχει σε προσαρμοστικότητα και επεκτασιμότητα, το HPA (Tuned) παραμένει το πιο αξιόπιστο και σταθερό εργαλείο για γενική χρήση, ενώ το KOSMOS αντιπροσωπεύει μια ενδιαφέρουσα αλλά ανώριμη ερευνητική προσέγγιση με περιορισμένη εφαρμοσιμότητα σε παραγωγικά περιβάλλοντα. Όσον αφορά τα κριτήρια, το βέλτιστο είναι η Επεκτασιμότητα, καθώς τα περισσότερα εργαλεία υποστηρίζουν APIs, adapters και modular αρχιτεκτονική που επιτρέπει εύκολη προσαρμογή σε διαφορετικά workloads. Βέλτιστο μπορεί να θεωρηθεί και το κριτήριο της Υποστήριξης & Κοινότητας. Το χειρότερο κριτήριο είναι η Ασφάλεια, όπου όλα τα εργαλεία εμφανίζουν ελλείψεις, καθώς βασίζονται κυρίως στους μηχανισμούς RBAC και TLS του Kubernetes χωρίς πρόσθετες ενισχύσεις ασφαλείας.

5.2.8 ML/AI

Η κατηγορία αυτή περιλαμβάνει εργαλεία και προσεγγίσεις που ενσωματώνουν τεχνικές μηχανικής μάθησης (ML) και τεχνητής νοημοσύνης (AI) για τη βελτιστοποίηση της λειτουργίας, της κλιμάκωσης και της πρόβλεψης συμπεριφοράς σε Kubernetes περιβάλλοντα. Τα κριτήρια αξιολόγησης που χρησιμοποιήθηκαν είναι αυτά που ορίστηκαν στην Ενότητα 5.1, με βαθμολογία 1–5.

Απόδοση

Ο DQN Scheduler εφαρμόζει Deep Reinforcement Learning (DRL) για τη βελτιστοποίηση του χρονοπρογραμματισμού pods, επιτυγχάνοντας προσαρμοστική και δυναμική κατανομή πόρων με μικρή καθυστέρηση απόφασης. Οι μελέτες δείχνουν υψηλό throughput και αποδοτική χρήση CPU/GPU, ιδίως σε ελεγχόμενα ή πειραματικά περιβάλλοντα. Ωστόσο, σε μεγάλα production clusters η απόδοση μπορεί να μειωθεί ελαφρώς λόγω του κόστους εκπαίδευσης και inference του μοντέλου RL. Συνεπώς αξιολογείται με 4.

Το KubePipe αξιοποιεί ML pipelines για αυτοματοποίηση διαδικασιών μέσα στο Kubernetes, όμως η αποτελεσματικότητα του εξαρτάται σε μεγάλο βαθμό από τη ποιότητα του setup, των δεδομένων εκπαίδευσης και την υποδομή στην οποία εκτελείται. Σε σταθερά workloads λειτουργεί επαρκώς, αλλά σε πιο σύνθετα ή μεταβαλλόμενα περιβάλλοντα εμφανίζει πτώση επιδόσεων λόγω σημαντικών καθυστερήσεων στην επεξεργασία και αυξημένης χρήσης πόρων. Επομένως, ταξινομείται ως εργαλείο μέτριας απόδοσης και βαθμολογείται με 3.

Το Kubeflow αξιολογείται με 5, καθώς προσφέρει εξαιρετικά υψηλή απόδοση σε ML pipelines και training workloads, χάρη στην πλήρη ενοποίηση με το Kubernetes control plane και τη χρήση εξειδικευμένων controllers (TFJob, PyTorchJob, MPIJob). Οι μηχανισμοί αυτοί επιτρέπουν την κατανομή εργασιών σε πολλαπλά nodes με ελάχιστο overhead, αξιοποιώντας native Kubernetes scheduling και GPU resources. Μελέτες δείχνουν ότι σε περιβάλλοντα με GPU acceleration, το Kubeflow επιτυγχάνει έως και 30–40% υψηλότερο throughput σε σχέση με παραδοσιακά single-node ML frameworks [1][2]. Η κατανάλωση πόρων παραμένει αποδοτική, και το latency εξαρτάται κυρίως από το underlying hardware, όχι από το ίδιο το πλαίσιο εκτέλεσης.

Το Polyaxon αξιολογείται με 4, καθώς παρουσιάζει πολύ καλή απόδοση στη διαχείριση και εκτέλεση πειραμάτων (experiments) σε Kubernetes clusters. Η χρήση optimized job scheduling, caching και distributed training υποστηρίζει υψηλό ρυθμό εκτέλεσης, ειδικά σε workloads με μεγάλο αριθμό parallel trials [3]. Παρότι η απόδοση παραμένει σταθερή στα περισσότερα σενάρια, το overhead αυξάνεται ελαφρώς όταν εκτελούνται ταυτόχρονα μεγάλοι αριθμοί experiments (>100), λόγω του controller synchronization και των database queries για logging/metrics. Παρ' όλα αυτά, η συμπεριφορά του εργαλείου παραμένει συνεπής και αποδοτική, δικαιολογώντας βαθμό αξιολόγησης 4.

Ευκολία Ενσωμάτωσης

Ο DQN Scheduler απαιτεί προσαρμογή των Kubernetes manifests και εγκατάσταση custom components που ενσωματώνουν το reinforcement learning μοντέλο στο control plane. Η διαδικασία περιλαμβάνει χειροκίνητη ρύθμιση CRDs και προσαρμογή των RBAC πολιτικών, ενώ η τεκμηρίωση είναι ερευνητικού επιπέδου και όχι παραγωγικής ωριμότητας. Παρότι η εγκατάσταση είναι εφικτή, δεν είναι αυτοματοποιημένη (απουσία Helm chart ή Operator) και προϋποθέτει εξειδικευμένες γνώσεις σε Kubernetes και ML. Επομένως, το εργαλείο παρουσιάζει δύσκολη ενσωμάτωση και βαθμολογείται με 2.

Το KubePipe απαιτεί προσαρμογή στα υπάρχοντα Kubernetes manifests και σύνδεση με ML pipelines που εκτελούνται σε containers. Παρότι βασίζεται σε containerized αρχιτεκτονική, η εγκατάσταση δεν είναι πλήρως αυτοματοποιημένη και απαιτεί σύνθετη παραμετροποίηση για να λειτουργήσει σωστά (π.χ. ορισμός volumes, secrets και δικαιωμάτων εκτέλεσης). Η τεκμηρίωση είναι διαθέσιμη αλλά περιορισμένη, και απουσιάζει επίσημος Operator. Συνεπώς, το KubePipe επίσης αξιολογείται με 2.

Το Kubeflow αξιολογείται με 3, καθώς, παρότι αποτελεί το πλέον ολοκληρωμένο πλαίσιο για εκτέλεση ML workloads στο Kubernetes, η διαδικασία εγκατάστασης και ενσωμάτωσης παραμένει πολύπλοκη. Υποστηρίζεται από Kubeflow manifests, Kfctl CLI, και Helm charts, ενώ η εγκατάσταση μπορεί να πραγματοποιηθεί σε διάφορους cloud providers (GCP, AWS, Azure) ή on-prem clusters. Ωστόσο, η διαδικασία απαιτεί προσεκτική παραμετροποίηση των CRDs, των RBAC ρυθμίσεων και των storage backends (π.χ. MinIO, S3, GCS). Παρότι η τεκμηρίωση είναι εκτενής και υπάρχουν επίσημοι οδηγοί εγκατάστασης, η πολυπλοκότητα των εξαρτήσεων και των components (KFServing, Katib, Pipelines, Notebooks) καθιστά την πλήρη ενσωμάτωση μέτριας δυσκολίας.

Το Polyaxon αξιολογείται με 4, καθώς παρέχει καλά οργανωμένα Helm charts, Polyaxon CLI, και Operators που απλοποιούν την εγκατάσταση και τη διασύνδεση με το Kubernetes API. Η βασική εγκατάσταση μπορεί να ολοκληρωθεί σε λίγα βήματα, με τη χρήση προτύπων YAML ή μέσω του command polyaxon admin deploy. Επιπλέον, προσφέρει ενσωματωμένη υποστήριξη για πολλαπλούς storage backends (S3, Azure Blob, GCS) και authentication providers (OIDC, LDAP). Για πιο σύνθετες λειτουργίες (π.χ. multi-tenant environments, GPU scheduling ή integration με MLFlow), απαιτούνται πρόσθετες ρυθμίσεις και tuning. Παρ' όλα αυτά, το υψηλό επίπεδο τεκμηρίωσης και τα διαθέσιμα deployment templates καθιστούν το εργαλείο σχετικά εύκολο στην ενσωμάτωση.

Επεκτασιμότητα

Ο DQN Scheduler υποστηρίζει επεκτασιμότητα σε κάποιο βαθμό, καθώς επιτρέπει την προσαρμογή του αλγορίθμου εκμάθησης και την τροποποίηση των CRDs που χρησιμοποιούνται για τον χρονοπρογραμματισμό. Οι χρήστες μπορούν να επεκτείνουν τη λειτουργία του μέσω παραμετροποίησης hyperparameters ή τροποποίησης του reinforcement model. Ωστόσο, δεν διαθέτει επίσημο API, plugin framework ή υποστήριξη από ευρύτερο οικοσύστημα εργαλείων (όπως operators ή extensibility hooks) και για αυτό αξιολογείται με 3.

Το KubePipe έχει modular αρχιτεκτονική για τον ορισμό ML pipelines μέσα στο Kubernetes και επιτρέπει την προσαρμογή των components (π.χ. data ingestion, preprocessing, training, deployment). Παρότι μπορεί να επεκταθεί μέσω νέων pipeline templates ή επιπλέον containerized components, δεν παρέχει πλήρως τεκμηριωμένο API ή plugin system για εξωτερικά integrations. Η επεκτασιμότητα περιορίζεται στο εσωτερικό του εργαλείου και απαιτεί χειροκίνητες αλλαγές στα manifests και στα scripts και για αυτό βαθμολογείται με 4.

Το Kubeflow αξιολογείται με 4, καθώς διαθέτει πολύ καλή επεκτασιμότητα μέσω modular αρχιτεκτονικής και εκτεταμένης υποστήριξης για custom components. Η πλατφόρμα βασίζεται σε

CRDs και Operators, επιτρέποντας τη δημιουργία και επέκταση λειτουργιών, όπως το KFServing (model serving), Katib (AutoML), και Pipelines (workflow orchestration). Επιπλέον, παρέχει SDKs για Python και REST APIs που διευκολύνουν την ανάπτυξη και ενσωμάτωση νέων modules. Ωστόσο, η απουσία ενός πλήρους plugin framework ή δυνατοτήτων επέκτασης σε runtime επίπεδο περιορίζει την πλήρη αυτοματοποίηση και ευελιξία στην προσαρμογή.

Το Polyaxon αξιολογείται με 4, καθώς υποστηρίζει μέτρια επεκτασιμότητα μέσω του REST API, των SDKs (Python, Go, JavaScript) και της δυνατότητας ανάπτυξης custom workflows. Οι χρήστες μπορούν να επεκτείνουν τη λειτουργικότητα μέσω configuration templates, scripts και webhooks, ενώ προσφέρονται integrations με frameworks, όπως TensorFlow, PyTorch και Scikit-learn. Ωστόσο, δεν διαθέτει CRDs ή plugin system για δυναμική επέκταση του control plane, με αποτέλεσμα οι προσαρμογές να περιορίζονται κυρίως στο επίπεδο pipelines και χρήσης API.

Ασφάλεια

Ο DQN Scheduler βασίζεται αποκλειστικά στους μηχανισμούς ασφαλείας του Kubernetes, όπως το RBAC και η χρήση service accounts για την εκτέλεση pods. Δεν παρέχει πρόσθετους μηχανισμούς προστασίας (όπως encryption, policy enforcement ή auditing) ούτε ενσωματώνει δικούς του ελέγχους ταυτότητας. Επειδή η ασφάλεια εξαρτάται πλήρως από το υποκείμενο cluster και όχι από το ίδιο το εργαλείο, η συνολική ασφάλεια θεωρείται βασική, με εμφανή κενά σε auditing και compliance και για αυτό αξιολογείται με 3.

Το KubePipe λειτουργεί πάνω από το Kubernetes και αξιοποιεί τους υπάρχοντες μηχανισμούς πρόσβασης (RBAC) και Secrets για τη διαχείριση credentials. Ωστόσο, δεν διαθέτει δική του πολιτική ασφάλειας, δεν εφαρμόζει encryption στα δεδομένα των pipelines ούτε παρέχει ενσωματωμένο audit logging. Η ασφάλεια επομένως περιορίζεται σε επίπεδο Kubernetes, χωρίς πρόσθετα layers προστασίας (π.χ. mTLS) και για αυτό αξιολογείται με 3.

Το Kubeflow αξιολογείται με 4, καθώς προσφέρει ισχυρή αλλά όχι πλήρη ασφάλεια. Υποστηρίζει μηχανισμούς όπως RBAC, Secrets management, TLS encryption και namespace-level isolation για τα workloads. Επιπλέον, ενσωματώνεται με OIDC (OpenID Connect) και Istio για authentication και policy enforcement, ενώ υποστηρίζει multi-user isolation μέσω Kubeflow Profiles και Network Policies. Παρότι παρέχει auditing μέσω Kubernetes logs και integration με εργαλεία όπως το Keycloak, δεν διαθέτει πλήρη μηχανισμό zero-trust ή ενσωματωμένο vulnerability scanning.

Το Polyaxon αξιολογείται με 4, καθώς παρέχει ισχυρή αλλά όχι εξαιρετική ασφάλεια. Υποστηρίζει authentication μέσω OAuth2 και SSO, προστασία επικοινωνίας με TLS/SSL, και δυνατότητα διαχείρισης χρηστών και ρόλων (roles). Επίσης, αξιοποιεί το RBAC του Kubernetes για έλεγχο πρόσβασης και διαχείριση secrets, ενώ διαθέτει μηχανισμό API token για προστασία REST endpoints. Ωστόσο, δεν εφαρμόζει mTLS, runtime policy enforcement ή advanced auditing.

Διαλειτουργικότητα

Ο DQN Scheduler ενσωματώνεται στο Kubernetes μέσω custom controller που αλληλεπιδρά

απευθείας με το Kubernetes API και τα pods scheduling events. Παρότι βασίζεται στο native API, μπορεί να συνεργαστεί με metrics servers (π.χ. Prometheus) για τη συλλογή δεδομένων εκπαίδευσης και βελτιστοποίησης. Ωστόσο, η διαλειτουργικότητα αυτή δεν είναι αυτοματοποιημένη — απαιτεί χειροκίνητη ρύθμιση των πηγών δεδομένων (metrics endpoints) και παρεμβάσεις στα manifests και για αυτό βαθμολογείται με 2.

Το KubePipe αλληλεπιδρά με το Kubernetes API για τη δημιουργία και διαχείριση ML pipelines χρησιμοποιώντας pods και services, ενώ μπορεί να συνδεθεί με εξωτερικά συστήματα αποθήκευσης ή μοντέλα ML training που εκτελούνται σε containers. Η συνεργασία του με άλλα εργαλεία (όπως Argo Workflows ή Prometheus) είναι δυνατή αλλά όχι εγγενής· απαιτεί manifests και custom configurations. Έτσι, λαμβάνει βαθμολογείται με 3 καθώς υποστηρίζει βασική συνεργασία με Kubernetes components και ML frameworks, χωρίς αυτοματοποιημένες διασυνδέσεις ή standard API

Το Kubeflow αξιολογείται με 4, καθώς προσφέρει υψηλό βαθμό διαλειτουργικότητας με το οικοσύστημα Kubernetes και τα εργαλεία CNCF. Υποστηρίζει native integration μέσω Custom Resource Definitions (CRDs) και μπορεί να λειτουργήσει σε συνδυασμό με Argo Workflows, Tekton, Istio, Prometheus, Grafana, και ML frameworks, όπως TensorFlow, PyTorch και XGBoost. Επιπλέον, παρέχει APIs και SDKs που επιτρέπουν την ανταλλαγή δεδομένων μεταξύ pipelines και εξωτερικών υπηρεσιών, όπως cloud storage (AWS S3, GCS) ή CI/CD συστήματα. Η διασύνδεση με monitoring εργαλεία είναι αυτοματοποιημένη.

Το Polyaxon αξιολογείται με 3, καθώς παρουσιάζει μέτρια διαλειτουργικότητα, επικεντρωμένη κυρίως στην επικοινωνία μέσω APIs και SDKs. Ενσωματώνεται με Kubernetes, επιτρέπει τη διαχείριση workloads μέσω CRDs, και υποστηρίζει διασύνδεση με ML frameworks (TensorFlow, PyTorch, Scikit-learn) και monitoring εργαλεία όπως το Prometheus. Ωστόσο, οι ενσωματώσεις με CI/CD pipelines (π.χ. Argo, Tekton) ή με policy enforcement frameworks (π.χ. OPA) δεν είναι αυτόματες και απαιτούν custom configuration. Η έλλειψη πλήρους native υποστήριξης για observability και auditing περιορίζει τη συνολική ενοποίηση

Υποστήριξη & Κοινότητα (Support & Community)

Η υποστήριξη και η κοινότητα διαφοροποιούνται σημαντικά μεταξύ των δύο εργαλείων. Ο DQN Scheduler αξιολογείται με βαθμό 2, καθώς, παρότι πρόκειται για ερευνητικό έργο με περιορισμένη κοινότητα και απουσία ενεργής ανάπτυξης, παραμένει διαθέσιμο μέσω δημόσιου αποθετηρίου (GitHub) και συνοδεύεται από τεχνική τεκμηρίωση και σχετικές επιστημονικές δημοσιεύσεις που επιτρέπουν τη βασική αναπαραγωγή και χρήση του. Αντίθετα, το KubePipe συγκεντρώνει βαθμό 3, διαθέτοντας ενεργό repository, τεκμηρίωση και μια μικρή αλλά λειτουργική ομάδα συνεισφερόντων, αν και απουσιάζει επίσημη οργανωτική υποστήριξη.

Το Kubeflow αξιολογείται με 5, καθώς διαθέτει εξαιρετικά ενεργή και παγκόσμια κοινότητα. Πρόκειται για CNCF project με χιλιάδες contributors, εκατοντάδες οργανισμούς που το υποστηρίζουν (Google, AWS, Microsoft, Canonical, Arrikto κ.ά.), και πολύ συχνές ενημερώσεις. Η τεκμηρίωση είναι πλήρης (επίσημο documentation, tutorials, και community guides), ενώ παρέχονται security patches, roadmap και τακτικά community events όπως το Kubeflow Summit.

Επιπλέον, υπάρχει εμπορική υποστήριξη μέσω εταιρειών (Arrikto, Canonical) που προσφέρουν enterprise εκδόσεις.

Το Polyaxon αξιολογείται με 4, καθώς διαθέτει μεγάλη και ενεργή κοινότητα, αλλά σε μικρότερη κλίμακα σε σχέση με το Kubeflow. Το έργο συντηρείται από την εταιρεία Polyaxon Inc., με επίσημη εμπορική υποστήριξη, τεκμηρίωση υψηλής ποιότητας και συχνά updates στο GitHub repository. Υπάρχει διαθέσιμο community Slack, open issues tracker και πλήρης online documentation. Παρότι δεν είναι CNCF project, η εξέλιξή του είναι σταθερή και διαθέτει επαγγελματική ομάδα συντηρητών, κάτι που το καθιστά ιδιαίτερα αξιόπιστο για παραγωγική χρήση.

Πίνακας 27: Συγκριτική αξιολόγηση εργαλείων στην κατηγορία AI/ML

	Εργαλεία			
Κριτήριο	**DQN Scheduler**	**KubePipe**	**Kubeflow**	**Polyaxon**
Απόδοση (Performance)	**4**	**3**	**5**	**4**
Ευκολία Ενσωμάτωσης	**2**	**2**	**3**	**4**
Επεκτασιμότητα	**3**	**4**	**4**	**4**
Ασφάλεια	**3**	**2**	**4**	**4**
Διαλειτουργικότητα	**2**	**3**	**4**	**3**
Υποστήριξη & Κοινότητα	**2**	**2**	**5**	**4**

Πηγές

[1] (Dakić, Dambić, Slovinac, & Redžepagić, 2025)

[2] (Suárez, Almeida, Blanco, & Toledo, 2025)

[3] (Yang & Kim, 2022)

[4] (Toka, Dobreff, Fodor, & Sonkoly, 2021)

[5] (Hoffpauir, et al., 2023)

Η κατηγορία Machine Learning & Intelligent Scheduling περιλαμβάνει εργαλεία που ενσωματώνουν τεχνικές Μηχανικής Μάθησης (ML) και Τεχνητής Νοημοσύνης (AI) για την αυτοματοποίηση διαδικασιών, τη βελτιστοποίηση χρονοπρογραμματισμού και τη διαχείριση πόρων σε Kubernetes clusters. Από τη συγκριτική αξιολόγηση των εργαλείων Kubeflow, Polyaxon, DQN Scheduler και KubePipe, προκύπτει σαφής διαφοροποίηση ως προς την ωριμότητα, την παραγωγική ετοιμότητα και τη διαλειτουργικότητα τους.

Το Kubeflow αναδεικνύεται ως το πιο ώριμο και ολοκληρωμένο framework, με την υψηλότερη βαθμολογία σε απόδοση (5), ασφάλεια (4), διαλειτουργικότητα (4) και υποστήριξη κοινότητας (5). Προσφέρει native ενσωμάτωση με το Kubernetes μέσω CRDs, υποστηρίζει πολλαπλά ML

frameworks (TensorFlow, PyTorch, XGBoost) και παρέχει ισχυρή modular αρχιτεκτονική με Operators και APIs. Παρά τη σχετική πολυπλοκότητα εγκατάστασης (3), παραμένει η πιο ολοκληρωμένη και αξιόπιστη επιλογή για παραγωγικά περιβάλλοντα ML στο Kubernetes.

Το Polyaxon κατατάσσεται αμέσως μετά, με υψηλή απόδοση (4) και ευκολία ενσωμάτωσης (4), ενώ διατηρεί ενεργή κοινότητα και εμπορική υποστήριξη. Προσφέρει ισχυρά APIs και SDKs για διαχείριση πειραμάτων, αλλά η διαλειτουργικότητά του (3) είναι περιορισμένη σε σχέση με το Kubeflow. Επομένως, αποτελεί μια πιο ελαφριά και πρακτική λύση, κατάλληλη για οργανισμούς που αναζητούν λειτουργική ισορροπία μεταξύ ευκολίας και απόδοσης.

Ο DQN Scheduler υπερέχει σε καινοτομία και αποδοτικότητα (4) λόγω της χρήσης Deep Reinforcement Learning για χρονοπρογραμματισμό, ωστόσο η ενσωμάτωσή του είναι δύσκολη (2) και η κοινότητα περιορισμένη (2). Παραμένει κυρίως ερευνητικό εργαλείο, κατάλληλο για πειραματικά σενάρια και proof-of-concept εφαρμογές.

Το KubePipe παρουσιάζει μέτρια απόδοση (3) και επεκτασιμότητα (3), αλλά διακρίνεται για τη modular αρχιτεκτονική του και την ικανοποιητική υποστήριξη (3). Αν και δεν προσφέρει αυτοματοποιημένη ενσωμάτωση, αποτελεί μια σταθερή και πιο προσιτή λύση για βασική ML αυτοματοποίηση εντός Kubernetes.

Συνολικά, το Kubeflow αναδεικνύεται ως το πληρέστερο και πιο ώριμο εργαλείο της κατηγορίας, συνδυάζοντας υψηλή απόδοση, ευρεία διαλειτουργικότητα και ενεργή υποστήριξη κοινότητας. Το Polyaxon αποτελεί την πιο πρακτική εναλλακτική για οργανισμούς που επιθυμούν πιο απλή εγκατάσταση και εμπορική υποστήριξη, ενώ τα DQN Scheduler και KubePipe αντιπροσωπεύουν ερευνητικές ή πειραματικές προσεγγίσεις με περιορισμένη παραγωγική ωριμότητα. Όσον αφορά τα κριτήρια, η Απόδοση καταγράφει τη βέλτιστη επίδοση της κατηγορίας, καθώς τα περισσότερα εργαλεία επιτυγχάνουν αποδοτική αξιοποίηση πόρων και υποστήριξη κλιμακούμενων ML workloads. Αντίθετα, το κριτήριο της Ευκολίας Ενσωμάτωσης εμφανίζει τη χαμηλότερη απόδοση, εξαιτίας της αυξημένης πολυπλοκότητας στις ρυθμίσεις, των εξαρτήσεων από υποκείμενες υποδομές και της ανάγκης για εξειδικευμένες γνώσεις κατά την εγκατάσταση.

5.2.9 Service Mesh

Η κατηγορία **Service Mesh** καλύπτει εργαλεία που αναλαμβάνουν τη διαχείριση της επικοινωνίας μεταξύ microservices, παρέχοντας δυνατότητες, όπως traffic routing, service discovery, ασφάλεια, και observability. Στο Κεφάλαιο 4 (Ενότητα 4.2.9) αναλύθηκαν εκτενώς τα εργαλεία **Istio**, **Linkerd** και **Cilium**, με βάση επιστημονικές δημοσιεύσεις, την τεκμηρίωση του CNCF και επίσημες πηγές. Η αξιολόγηση τους πραγματοποιείται εδώ με βάση τα κριτήρια που ορίστηκαν στην Ενότητα 5.1.

Ανάλυση Αποτελεσμάτων Ανά Κριτήριο

Απόδοση (Performance)

Η απόδοση αξιολογήθηκε λαμβάνοντας υπόψη το latency και το υπολογιστικό overhead που εισάγεται από την αρχιτεκτονική του κάθε εργαλείου. Το Istio χρησιμοποιεί sidecar proxies (Envoy) σε κάθε pod, γεγονός που προκαλεί υψηλό υπολογιστικό overhead και αυξημένο latency (10–20% σύμφωνα με [1][2]) λόγω του routing μέσω proxy layer. Αν και παρέχει προηγμένες λειτουργίες traffic management, mTLS και observability, το κόστος σε latency και CPU μνήμη είναι αισθητό ακόμη και σε μεσαίο φόρτο, επιβαρύνοντας τον έλεγχο ροής (control plane) σε μεγάλα clusters. Συνεπώς, αξιολογείται με 2 καθώς η κατανάλωση πόρων είναι υψηλή και η καθυστέρηση αυξάνεται σημαντικά με την κλιμάκωση.

Το Linkerd χρησιμοποιεί lightweight Rust-based proxies και zero-config mTLS, μειώνοντας δραστικά το latency σε σχέση με το Istio. Σύμφωνα με μετρήσεις του CNCF [1][2], το latency overhead του Linkerd είναι έως 60% χαμηλότερο από του Istio, ενώ η κατανάλωση πόρων παραμένει χαμηλή ακόμη και σε clusters >1000 pods. Ετσι, βαθμολογείται με 4 διότι υπάρχουν ακόμη περιθώρια βελτίωσης της απόδοσης.

Το Cilium λειτουργεί μέσω eBPF απευθείας στο kernel space, αποφεύγοντας εντελώς το proxy path. Έτσι, προσφέρει σχεδόν μηδενικό latency overhead και βέλτιστη χρήση πόρων, όπως επιβεβαιώνεται από [5] και benchmarking reports της Isovalent[105]. Το eBPF pipeline του επιτρέπει line-rate packet processing με πολλαπλάσιο throughput σε σχέση με sidecar-based λύσεις, χωρίς να απαιτείται επιπλέον user-space networking. Για αυτό αξιολογηθηκε με 5.

Ευκολία Ενσωμάτωσης

Το Istio διαθέτει πλήρη τεκμηρίωση και παρέχει Helm charts, Operators και istioctl CLI, ωστόσο η διαδικασία εγκατάστασης και ρύθμισης είναι πολύπλοκη λόγω του πλήθους των components (pilot, ingress/egress gateways, sidecars, CRDs). Η βασική εγκατάσταση είναι εφικτή με μία εντολή (istioctl install), όμως η παραμετροποίηση των πολιτικών ασφάλειας, traffic management και observability απαιτεί χειροκίνητες επεμβάσεις σε manifests και καλή γνώση της αρχιτεκτονικής του service mesh. Συνεπώς, η ενσωμάτωση θεωρείται μέτρια και αξιολογείται με 3.

Το Linkerd ξεχωρίζει για τη χαμηλή πολυπλοκότητα εγκατάστασης και βαθμολογείται με 4. Η βασική εγκατάσταση (linkerd install ή μέσω Helm chart) είναι γρήγορη και σταθερή, ενώ η αρχιτεκτονική του (μόνο control plane + lightweight proxies) επιτρέπει άμεση λειτουργία χωρίς εκτεταμένες ρυθμίσεις. Η παραμετροποίηση πιο προχωρημένων χαρακτηριστικών (π.χ. multicluster communication, custom certificates) απαιτεί επιπλέον χειροκίνητα βήματα, αλλά για το 80% των σεναρίων η εγκατάσταση είναι σχεδόν αυτοματοποιημένη.

Το Cilium μπορεί να φαίνεται πιο περίπλοκο λόγω της eBPF τεχνολογίας, ωστόσο παρέχει Operators, Helm charts και Cilium CLI που καθιστούν την ενσωμάτωση αρκετά απλή. Διαθέτει προκαθορισμένες ρυθμίσεις (default configurations) που καλύπτουν τις περισσότερες περιπτώσεις, ενώ η τεκμηρίωση είναι αναλυτική και οδηγεί βήμα προς βήμα τη διαδικασία εγκατάστασης. Πιο προχωρημένες λειτουργίες (π.χ. network policies, Hubble observability, cluster mesh) απαιτούν

[105] https://docs.isovalent.com/project/cilium/index.html

πρόσθετες ρυθμίσεις, αλλά το βασικό deployment είναι καλά αυτοματοποιημένο. Επομένως, το Cilium αξιολογείται με 4 αφού παρέχει ώριμα και αυτοματοποιημένα εργαλεία εγκατάστασης.

Επεκτασιμότητα

Το Istio διαθέτει πλήρως τεκμηριωμένα APIs, CRDs (Custom Resource Definitions) και extension points που επιτρέπουν την προσθήκη νέων λειτουργιών. Μπορεί να ενσωματωθεί με εξωτερικά monitoring και policy εργαλεία, όπως Prometheus, Grafana, Jaeger, Open Policy Agent (OPA), και external authorization systems. Επιπλέον, υποστηρίζει custom filters στο Envoy proxy, καθώς και WebAssembly extensions (Wasm), που δίνουν τη δυνατότητα δημιουργίας custom λογικής στο data plane. Επομένως, το Istio αξιολογείται με 4, αφού διαθέτει ώριμο API, modular σχεδίαση και δυνατότητα προχωρημένης ενσωμάτωσης αλλά όχι SDK.

Το Linkerd είναι πιο απλό σε σύγκριση με το Istio, κάτι που μειώνει τη δυνατότητα επέκτασής του. Υποστηρίζει integration μέσω CRDs και metrics APIs (π.χ. Prometheus integration), αλλά δεν διαθέτει επίσημο plugin framework ή μηχανισμό για προσθήκη custom proxy logic, όπως στο Istio με τα Wasm filters. Οι επεκτάσεις του περιορίζονται σε service profiles, policy configurations και CLI plugins, χωρίς δυνατότητα deep integration με εξωτερικά control planes. Συνεπώς αξιολογείται με βαθμό 3, καθώς προσφέρει μέτρια επεκτασιμότητα

Το Cilium διαθέτει modular αρχιτεκτονική βασισμένη σε eBPF, που του επιτρέπει να επεκτείνεται δυναμικά χωρίς sidecars ή αλλαγές στον κώδικα των pods. Προσφέρει Cilium Network Policies, API extensions, Hubble observability framework, καθώς και πλήρη integration με Istio, Linkerd, Envoy, Prometheus, Grafana και άλλα CNCF εργαλεία. Επιπλέον, υποστηρίζει custom eBPF programs, SDKs και ενεργό community-driven plugin development, καθιστώντας το κατάλληλο για πολύ μεγάλα παραγωγικά περιβάλλοντα. Για τον λόγο αυτό, αξιολογείται με βαθμό 5.

Ασφάλεια

Το Istio αξιολογείται με βαθμό 4, καθώς προσφέρει πλήρες σύνολο μηχανισμών ασφάλειας, όπως automatic mTLS, authorization policies και policy enforcement μέσω Envoy filters, διασφαλίζοντας υψηλό επίπεδο ελέγχου πρόσβασης και κρυπτογράφησης. Ωστόσο, δεν υλοποιεί πλήρως zero-trust networking σε όλο το επίπεδο εφαρμογής, ούτε ενσωματώνει μηχανισμούς runtime προστασίας ή vulnerability scanning, ενώ η ρύθμιση των πολιτικών παραμένει πολύπλοκη και μη αυτοματοποιημένη.Το Linkerd λαμβάνει βαθμό 3, καθώς, παρότι παρέχει mTLS από προεπιλογή και ενσωματωμένο μηχανισμό ταυτοποίησης μέσω πιστοποιητικών, δεν διαθέτει ολοκληρωμένο πλαίσιο πολιτικών πρόσβασης ή auditing. Η ασφάλεια περιορίζεται κυρίως στο επίπεδο κρυπτογράφησης και ταυτότητας, χωρίς δυνατότητες policy enforcement, fine-grained authorization ή ενσωμάτωση με εξωτερικά συστήματα συμμόρφωσης. Επομένως, προσφέρει ικανοποιητική αλλά όχι πλήρη ασφάλεια, που αντιστοιχεί σε βαθμό 3.Αντίθετα, το Cilium συγκεντρώνει βαθμό 5, καθώς ενσωματώνει zero-trust αρχιτεκτονική, network-level policy enforcement και πλήρη συμβατότητα με συστήματα παρακολούθησης και ασφάλεια

Διαλειτουργικότητα

Το Istio παρουσιάζει υψηλή διαλειτουργικότητα, καθώς ενσωματώνεται πλήρως με το οικοσύστημα εργαλείων του Kubernetes και του CNCF μέσω standard APIs και Custom Resource

Definitions (CRDs). Υποστηρίζει αμφίδρομη επικοινωνία και αυτοματοποιημένες συνδέσεις με εργαλεία παρακολούθησης και παρατήρησης, όπως τα Prometheus, Grafana, Kiali και Jaeger, καθώς και με συστήματα πολιτικών όπως το OPA (Open Policy Agent). Επιπλέον, η ενοποίηση με το Envoy API επιτρέπει τη συνεργασία με άλλα service meshes, διευρύνοντας τη λειτουργικότητά του. Η εγκατάσταση και ρύθμιση των παραπάνω συνδέσεων γίνεται σε μεγάλο βαθμό αυτόματα, χωρίς την ανάγκη εκτεταμένων χειροκίνητων παρεμβάσεων, γεγονός που το καθιστά ιδιαίτερα αποδοτικό ως προς τη διαλειτουργικότητα.

Το Linkerd εμφανίζει μέτρια διαλειτουργικότητα, καθώς υποστηρίζει βασικές ενσωματώσεις με εργαλεία παρακολούθησης όπως το Prometheus και το Grafana, παρέχοντας λειτουργικά dashboards και συλλογή μετρικών μέσω του control plane. Ωστόσο, η διασύνδεσή του με άλλα εργαλεία του οικοσυστήματος Kubernetes, όπως το Jaeger, το OpenTelemetry ή συστήματα πολιτικών όπως το OPA, δεν είναι πλήρως αυτοματοποιημένη και απαιτεί χειροκίνητη παραμετροποίηση ή χρήση community extensions. Επιπλέον, δεν διαθέτει επίσημα τεκμηριωμένα APIs ή μηχανισμούς πλήρους αμφίδρομης επικοινωνίας με άλλα υποσυστήματα, γεγονός που περιορίζει τη συνεργασία του με πιο σύνθετα περιβάλλοντα. Συνολικά, προσφέρει επαρκή αλλά περιορισμένη διαλειτουργικότητα, η οποία αξιολογείται με βαθμό 3.

Το Cilium παρουσιάζει υψηλή διαλειτουργικότητα, καθώς υποστηρίζει πολλαπλά container runtimes (όπως Docker, containerd και CRI-O) και μπορεί να λειτουργήσει με διάφορους orchestrators, πέραν του Kubernetes, όπως το Nomad και το Mesos. Επιπλέον, ενσωματώνεται πλήρως με εργαλεία παρακολούθησης και ορατότητας, όπως τα Prometheus, Grafana και Hubble, ενώ υποστηρίζει συνεργασία με service meshes όπως το Istio, το Linkerd και το Envoy, αξιοποιώντας το eBPF για αποδοτική επικοινωνία σε επίπεδο δικτύου. Παράλληλα, η δυνατότητα ενοποίησης με συστήματα πολιτικών, όπως το OPA, επιτρέπει την ασφαλή και δυναμική διαχείριση ταυτότητας και πρόσβασης. Μέσω των CRDs και των standard APIs, το Cilium προσφέρει αμφίδρομη και σε μεγάλο βαθμό αυτοματοποιημένη συνεργασία με άλλα υποσυστήματα του οικοσυστήματος Kubernetes, γεγονός που το καθιστά ένα από τα πιο ολοκληρωμένα εργαλεία σε όρους διαλειτουργικότητας, με συνολική αξιολόγηση 4.

Υποστήριξη & Κοινότητα

Το Istio διαθέτει μία από τις πιο ενεργές και καθιερωμένες κοινότητες στον χώρο των service meshes, με χιλιάδες contributors και συνεχή ανάπτυξη στο GitHub. Υποστηρίζεται επίσημα από μεγάλους οργανισμούς όπως η Google, IBM και Solo.io, ενώ βρίσκεται υπό τη σκέπη της CNCF ως mature project. Υπάρχουν συχνά releases, συνεχείς ενημερώσεις ασφαλείας (security patches) και πλήρης τεκμηρίωση. Παράλληλα, πραγματοποιούνται παγκόσμια συνέδρια και workshops (π.χ. IstioCon), ενώ παρέχεται και εμπορική υποστήριξη από διάφορους vendors. Για τους λόγους αυτούς, το Istio αξιολογείται με βαθμό 5, καθώς πληροί πλήρως τα χαρακτηριστικά μιας εξαιρετικά ενεργής κοινότητας με επαγγελματική υποστήριξη και μακροπρόθεσμο roadmap.

Το Linkerd διαθέτει μεγάλη αλλά πιο περιορισμένη κοινότητα σε σύγκριση με το Istio, με εκατοντάδες ενεργούς contributors, τακτικά releases και συνεχή τεχνική υποστήριξη μέσω του CNCF και του Buoyant.io (του βασικού οργανισμού ανάπτυξης). Υπάρχουν διαθέσιμα κανάλια επικοινωνίας (Slack, GitHub discussions, community meetings) και πλήρης τεκμηρίωση, ενώ η κοινότητα παραμένει ενεργή σε ζητήματα debugging και εξέλιξης του εργαλείου. Ωστόσο, η

υποστήριξή του δεν είναι τόσο εκτεταμένη ή εμπορικά δομημένη όσο του Istio. Συνεπώς, το Linkerd αξιολογείται με βαθμό 4, καθώς διαθέτει μεγάλη και σταθερά ενεργή κοινότητα, αλλά όχι το ίδιο επίπεδο παγκόσμιας εμβέλειας και εταιρικής υποστήριξης.

Το Cilium παρουσιάζει ραγδαία αυξανόμενη κοινότητα, καθώς υποστηρίζεται ενεργά από τη CNCF και μεγάλες εταιρείες, όπως η Isovalent, η Google Cloud και η AWS. Διαθέτει εκατοντάδες συνεισφέροντες, πολύ συχνά releases και πλούσια τεκμηρίωση, ενώ έχει καθιερωθεί ως βασικό εργαλείο για networking και observability μέσω eBPF στο Kubernetes. Η κοινότητά του είναι ιδιαίτερα δραστήρια στα forums και στα community events (όπως CiliumCon), αλλά η εμπορική υποστήριξη είναι κυρίως μέσω της Isovalent και όχι πολλών vendors. Έτσι, το Cilium βαθμολογείται επίσης με 4, καθώς έχει μεγάλη, ενεργή και καλά οργανωμένη κοινότητα, αλλά όχι ακόμη την έκταση και πολυεπίπεδη υποστήριξη του Istio.

Πίνακας 28: Συγκριτική αξιολόγηση εργαλείων στην κατηγορία Service Mesh

	Εργαλείο		
Κριτήριο	**Istio**	**Linkerd**	**Cilium**
Απόδοση (Performance)	**2**	**4**	**5**
Ευκολία Ενσωμάτωσης	**2**	**4**	**4**
Επεκτασιμότητα	**4**	**3**	**5**
Ασφάλεια	**4**	**3**	**5**
Διαλειτουργικότητα	**4**	**3**	**4**
Υποστήριξη & Κοινότητα	**5**	**4**	**4**

Πηγές

[1] (Zhu, et al., 2023)

[2] (Johng, Kalia, Xiao, Vuković, & Chung, 2019)

[3] (Wang, Zhu, Guo, & Liu , 2025)

[4] (Cloud Native Computing Foundation (CNCF), 2024)– Service Mesh Projects.

[5] (Sedghpour & Townend, 2022)

Η κατηγορία Service Mesh επικεντρώνεται στη διαχείριση της επικοινωνίας, της ασφάλειας και της παρατήρησης (observability) μεταξύ μικροϋπηρεσιών σε περιβάλλοντα Kubernetes. Από την ανάλυση των εργαλείων Istio, Linkerd και Cilium, προκύπτουν σαφείς διαφοροποιήσεις ως προς την απόδοση, την επεκτασιμότητα, την ευκολία ενσωμάτωσης και την ωριμότητα υποστήριξης.

Το Cilium αναδείχθηκε ως το πιο ολοκληρωμένο και τεχνολογικά ώριμο εργαλείο, συγκεντρώνοντας τις υψηλότερες βαθμολογίες σε επέκταση (5), ασφάλεια (5) και απόδοση (5).

Χάρη στη χρήση eBPF στο επίπεδο του πυρήνα (kernel space), εξαλείφει το overhead των sidecar proxies, επιτυγχάνοντας σχεδόν μηδενικό latency και βέλτιστη αξιοποίηση πόρων. Παράλληλα, προσφέρει πλήρη διαλειτουργικότητα (4) με Istio, Linkerd, Prometheus, Grafana και OPA, καθώς και ισχυρή και ενεργή κοινότητα υποστήριξης. Η ισορροπία μεταξύ επιδόσεων, ασφάλειας και ευελιξίας καθιστά το Cilium την πλέον αποδοτική και ώριμη λύση service mesh για παραγωγικά περιβάλλοντα.

Το Istio παραμένει το πληρέστερο σε λειτουργικότητα και το πιο διαδεδομένο εργαλείο της κατηγορίας, προσφέροντας εκτεταμένες δυνατότητες traffic management, policy enforcement και observability. Διαθέτει πολύ υψηλή επέκταση (4), ασφάλεια (4) και διαλειτουργικότητα (4), καθώς ενσωματώνεται άμεσα με Prometheus, Jaeger, OPA και Envoy API. Ωστόσο, το υπολογιστικό overhead (2) λόγω των sidecar proxies και η πολυπλοκότητα ενσωμάτωσης (3) το καθιστούν πιο απαιτητικό στη διαχείριση, ειδικά σε μεγάλα clusters. Συνεπώς, το Istio είναι η πλέον λειτουργικά πλήρης αλλά και βαρύτερη επιλογή, ιδανική για οργανισμούς με ώριμες DevOps πρακτικές.

Το Linkerd διακρίνεται για την απλότητα και την αποδοτικότητά του, προσφέροντας ελαφριά αρχιτεκτονική, χαμηλό latency και εύκολη εγκατάσταση (4). Αν και υστερεί σε επέκταση (3) και διαλειτουργικότητα (3), παρέχει αξιόπιστη απόδοση (4) και καλή ασφάλεια (3) μέσω εγγενούς mTLS. Με σταθερή και ενεργή κοινότητα (4), αποτελεί την πιο φιλική επιλογή για μικρότερα ή μεσαίας κλίμακας clusters, όπου προέχει η απλότητα έναντι της πλήρους λειτουργικότητας.

Συνολικά, η κατηγορία Service Mesh δείχνει πως το Cilium υπερέχει συνολικά λόγω του συνδυασμού απόδοσης, ασφάλειας και επεκτασιμότητας, καθιστώντας το την καταλληλότερη επιλογή για παραγωγικά και πολυσύνθετα περιβάλλοντα Kubernetes. Το Istio παραμένει το πληρέστερο και πιο ώριμο σε χαρακτηριστικά εργαλείο, αλλά με υψηλότερο κόστος σε πόρους και πολυπλοκότητα, ενώ το Linkerd ξεχωρίζει ως η πιο ελαφριά και εύχρηστη λύση για οργανισμούς με ανάγκη ταχύτητας και χαμηλού λειτουργικού κόστους. Σε επίπεδο κριτηρίων, η Υποστήριξη & Κοινότητα εμφανίζει τη βέλτιστη απόδοση, λόγω της ευρείας διάδοσης και ενεργής συντήρησης των έργων, ενώ η Ευκολία Ενσωμάτωσης αποτελεί το ασθενέστερο κριτήριο, καθώς τα περισσότερα εργαλεία απαιτούν εξειδικευμένες γνώσεις και προσαρμογή στις υπάρχουσες υποδομές για βέλτιστη λειτουργία.

5.3 Γενικά Συμπεράσματα

Στην παρούσα ενότητα παρουσιάζονται τα συνοπτικά συμπεράσματα από τη συγκριτική αξιολόγηση που πραγματοποιήθηκε στο Κεφάλαιο 5.2, βάσει των κριτηρίων που ορίστηκαν στην Ενότητα 5.1 και των αναλυτικών στοιχείων από το Κεφάλαιο 4.

Η συνολική αξιολόγηση των εργαλείων του οικοσυστήματος Kubernetes ανέδειξε ένα ώριμο, δυναμικά εξελισσόμενο περιβάλλον, στο οποίο η καινοτομία και η σταθερότητα συνυπάρχουν σε διαφορετικά επίπεδα ωριμότητας. Παρότι κάθε κατηγορία εργαλείων εξυπηρετεί διακριτές λειτουργίες — από τη διαχείριση υπηρεσιών και την αυτοματοποίηση έως την παρακολούθηση και την κλιμάκωση — κοινά μοτίβα αναδύθηκαν ως καθοριστικοί παράγοντες επιτυχίας: η modular

αρχιτεκτονική, η εγγενής ενσωμάτωση στο Kubernetes API και η ύπαρξη ενεργής κοινότητας υποστήριξης.

Σε συνολική θεώρηση όλων των κατηγοριών, παρατηρείται ότι εργαλεία όπως το Grafana, Argo CD, Kubeflow, Cilium και Vault ξεχωρίζουν σε ευρύ φάσμα κριτηρίων, συνδυάζοντας υψηλή απόδοση, επεκτασιμότητα και ισχυρή κοινοτική υποστήριξη. Πρόκειται για ώριμα έργα με ενεργή ανάπτυξη, modular σχεδίαση και πλήρη ενσωμάτωση με το οικοσύστημα του Kubernetes. Ενδεχομένως να βάζαμε σε αυτό το σύνολο και το Suse Rancher, το οποίο, αν και δεν βρίσκεται υπό την αιγίδα του CNCF, είναι αρκετά ώριμο και ευρέως χρησιμοποιούμενο εργαλείο που συνδέεται στενά με το οικοσύστημα των εργαλείων Kubernetes.

Αντίθετα, εργαλεία όπως το Ramen, GoCD, K8sES, KOSMOS και KubePipe εμφανίζουν χαμηλότερες επιδόσεις σε πολλαπλά κριτήρια, κυρίως ως προς την ευκολία ενσωμάτωσης, την ασφάλεια και τη διαλειτουργικότητα, γεγονός που περιορίζει τη χρήση τους σε παραγωγικά περιβάλλοντα.

Εν γένει, μπορεί να ειπωθεί πως τα εργαλεία που αξιοποιούν native μηχανισμούς του Kubernetes (όπως CRDs, Operators και Controllers) επέδειξαν σταθερά υψηλότερες επιδόσεις σε επεκτασιμότητα, ασφάλεια και διαλειτουργικότητα. Η προσέγγιση αυτή μειώνει το λειτουργικό βάρος και διευκολύνει την ενοποίηση με άλλα συστήματα του οικοσυστήματος CNCF, επιβεβαιώνοντας τη σημασία του “Kubernetes-native design” ως βασική αρχή βέλτιστης πρακτικής. Αντίθετα, λύσεις που στηρίζονται σε βαρύτερα επίπεδα αφαίρεσης (όπως sidecar proxies ή εξωτερικοί controllers) προσφέρουν πλούσιες λειτουργίες, αλλά συχνά με αυξημένο κόστος σε πόρους και πολυπλοκότητα.

Σε επίπεδο κριτηρίων, η Υποστήριξη & Κοινότητα αναδεικνύεται ως το βέλτιστο αντανακλώντας τη δυναμική των open-source κοινοτήτων και τη συνεχή συντήρηση των πιο διαδεδομένων έργων. Αντίθετα, η Ευκολία Ενσωμάτωσης αποτελεί το χειρότερο κριτήριο, καθώς απαιτεί προχωρημένες γνώσεις Kubernetes και περίπλοκη παραμετροποίηση για πλήρη αξιοποίηση των δυνατοτήτων των εργαλείων.

Η ασφάλεια και η παρατηρησιμότητα (observability) αναδείχθηκαν ως τομείς με τη μεγαλύτερη ωριμότητα. Η ευρεία υιοθέτηση τεχνολογιών όπως το eBPF, το mTLS και τα policy frameworks (OPA, Kyverno) έχει οδηγήσει σε service meshes και δικτυακά συστήματα με σχεδόν πλήρη αυτοματοποίηση ελέγχων πρόσβασης και εντοπισμού ανωμαλιών. Αντίστοιχα, τα εργαλεία αυτοματισμού και CI/CD που βασίζονται σε GitOps αρχές επιτυγχάνουν μεγαλύτερη αξιοπιστία και επαναληψιμότητα, προωθώντας την ενοποιημένη διαχείριση ροών εργασίας και υποδομών.

Παράλληλα, η αυξανόμενη αξιοποίηση τεχνικών τεχνητής νοημοσύνης (όπως στα εργαλεία DQN Scheduler, K8sGPT και KubePipe) σηματοδοτεί την είσοδο του Kubernetes σε μια νέα φάση “ευφυούς αυτοματισμού”, όπου η λήψη αποφάσεων γίνεται δυναμικά βάσει πραγματικών δεδομένων. Ωστόσο, τα εργαλεία αυτά παραμένουν ερευνητικού χαρακτήρα, με περιορισμένη κοινότητα και χαμηλότερη παραγωγική ωριμότητα, γεγονός που τα καθιστά περισσότερο ενδεικτικά της κατεύθυνσης του μέλλοντος παρά ώριμες λύσεις σήμερα.

Συνολικά, το οικοσύστημα Kubernetes δείχνει μια σαφή μετατόπιση από τα “βαριά” εργαλεία υποδομής προς ελαφρύτερες, modular και επεκτάσιμες αρχιτεκτονικές, που ενοποιούνται μέσω

standard APIs και υποστηρίζονται ενεργά από την CNCF. Τα πιο αποδοτικά και βιώσιμα εργαλεία συνδυάζουν υψηλή απόδοση, επεκτασιμότητα και ισχυρή κοινότητα, προσφέροντας πλήρως αυτοματοποιημένη, ασφαλή και παρατηρήσιμη λειτουργία. Επομένως, λύσεις όπως αυτές που βασίζονται σε eBPF, GitOps και event-driven αρχιτεκτονική διαφαίνεται ότι θα αποτελέσουν τον κορμό της επόμενης γενιάς cloud-native περιβαλλόντων.

ΚΕΦΑΛΑΙΟ 6 – ΕΡΕΥΝΗΤΙΚΑ ΚΕΝΑ & ΜΕΛΛΟΝΤΙΚΕΣ ΚΑΤΕΥΘΥΝΣΕΙΣ

Η συγκριτική ανάλυση και αξιολόγηση των εργαλείων που παρουσιάστηκαν στα προηγούμενα κεφάλαια ανέδειξε ένα σύνολο κρίσιμων ζητημάτων που εξακολουθούν να απασχολούν την ερευνητική και βιομηχανική κοινότητα γύρω από το οικοσύστημα του Kubernetes. Παρά την τεχνολογική ωριμότητα πολλών λύσεων, εντοπίζονται ακόμη κενά που αφορούν την ενοποίηση, την αυτοματοποίηση, την αποδοτικότητα, την ασφάλεια και τη μακροπρόθεσμη βιωσιμότητα των εργαλείων.

Σκοπός του κεφαλαίου αυτού είναι να συνοψίσει τα βασικά ερευνητικά κενά που προκύπτουν από τα αποτελέσματα της συγκριτικής αξιολόγησης και να προτείνει μελλοντικές κατευθύνσεις έρευνας για την κάλυψή τους. Κάθε ενότητα που ακολουθεί επικεντρώνεται σε ένα συγκεκριμένο ερευνητικό ζήτημα, παρουσιάζοντας:

- την ανάλυση του κενού,
- τις προτεινόμενες κατευθύνσεις,
- και τη δυνητική επιστημονική ή πρακτική συνεισφορά.

Το κεφάλαιο δομείται σε πέντε βασικά ερευνητικά πεδία:

1. Ενοποίηση και Διαλειτουργικότητα μεταξύ εργαλείων
2. Προβλεπτική και Αυτόνομη Διαχείριση Πόρων
3. Policy-aware Ασφάλεια και Κανονιστική Συμμόρφωση
4. Αποδοτικότητα κόστους και Ενεργειακή Βιωσιμότητα
5. Ωριμότητα και Βιωσιμότητα των Εργαλείων Ανοικτού Κώδικα

6.1 Ενοποίηση και Διαλειτουργικότητα Μεταξύ Εργαλείων

Η αξιολόγηση των εργαλείων σε όλες τις κατηγορίες (Monitoring, Security, CI/CD, Service Mesh, Autoscaling, ML/AI, Cluster Management) ανέδειξε ένα κοινό ζήτημα: την απουσία συνεκτικής ενοποίησης μεταξύ των επιμέρους εργαλείων και υποσυστημάτων.

Παρότι το οικοσύστημα Kubernetes χαρακτηρίζεται από πλούτο λύσεων, η διαλειτουργικότητά τους είναι περιορισμένη. Τα περισσότερα εργαλεία λειτουργούν απομονωμένα, ακολουθώντας διαφορετικές προσεγγίσεις επικοινωνίας με το Kubernetes API, διαφορετικά CRDs (Custom Resource Definitions), καθώς και ετερογενή μοντέλα δεδομένων.

Επιπλέον, η επικοινωνία μεταξύ κατηγοριών (π.χ. monitoring → autoscaling ή service mesh → security) γίνεται συνήθως μέσω ad hoc ρυθμίσεων, χειροκίνητων scripts ή custom connectors. Δεν υπάρχει κοινό πρότυπο ανταλλαγής δεδομένων ή σημασιολογική ευθυγράμμιση των μετρικών και των πολιτικών.

Αυτό έχει ως αποτέλεσμα πολλαπλές ασυνέπειες μεταξύ εργαλείων — π.χ. διαφορετικοί τρόποι συλλογής metrics και ασυμβατότητες στις πολιτικές, οι οποίες συχνά απαιτούν την ανάπτυξη

προσαρμοστικών μηχανισμών ενοποίησης ή ενδιάμεσου κώδικα, ώστε να επιτευχθεί στοιχειώδης συνεργασία μεταξύ των εργαλείων.

Η κατάσταση αυτή οφείλεται σε μεγάλο βαθμό στην αποκεντρωμένη ανάπτυξη του CNCF οικοσυστήματος, όπου κάθε εργαλείο εξελίσσεται αυτόνομα, συχνά με διαφορετικούς σχεδιαστικούς στόχους. Η απουσία ενός ενιαίου interoperability framework οδηγεί σε τεχνική πολυπλοκότητα κατά την ενοποίηση πολλών εργαλείων, επαναλαμβανόμενες υλοποιήσεις κοινών λειτουργιών (όπως authentication, policy enforcement, logging), και περιορισμένη ορατότητα σε cross-layer λειτουργίες, όπως η σύνδεση παρακολούθησης (observability) με προγνωστική κλιμάκωση (autoscaling) ή ασφάλεια (policy enforcement).

Πρακτικά, αυτό εμποδίζει τη δημιουργία “αυτορρυθμιζόμενων” (self-adaptive) Kubernetes περιβαλλόντων, καθώς τα εργαλεία δεν μπορούν να ανταλλάσσουν πληροφορίες σε πραγματικό χρόνο με συνεπή τρόπο.

Η μελλοντική έρευνα μπορεί να κινηθεί προς τρεις κύριες κατευθύνσεις:

1. Τυποποίηση ενιαίων μοντέλων δεδομένων και APIs:
   Ανάπτυξη Unified Kubernetes Data Model που θα επιτρέπει στα εργαλεία να ανταλλάσσουν metrics, policies και states μέσω κοινού API schema (π.χ. βασισμένου σε OpenAPI ή GraphQL).
   Αυτό θα επιτρέψει τη δημιουργία κοινών "σημείων αλήθειας" (single source of truth) μεταξύ εργαλείων.

2. Semantic Interoperability και Ontologies:
   Δημιουργία ontologies και knowledge graphs που θα ορίζουν εννοιολογικές σχέσεις μεταξύ διαφορετικών τύπων πόρων (pods, services, controllers, policies).
   Μέσω αυτής της σημασιολογικής ενοποίησης, ένα εργαλείο monitoring θα μπορεί, για παράδειγμα, να επικοινωνεί απευθείας με ένα εργαλείο autoscaling χωρίς χειροκίνητο mapping μετρικών.

3. Cross-Tool Orchestration Layer:
   Ανάπτυξη ενός “interoperability controller” που θα λειτουργεί ως *μετα-στρώμα* (meta-layer) πάνω από τα υπάρχοντα εργαλεία, διαχειριζόμενο τη ροή πληροφορίας μεταξύ monitoring, policy και scaling μηχανισμών. Αυτό θα μπορούσε να επιτευχθεί με χρήση *event-driven architectures* ή *service mesh*-based μηνυμάτων για την ενοποίηση των εργαλείων σε πραγματικό χρόνο.

Η κάλυψη αυτού του κενού θα επιτρέψει τη μετάβαση από απομονωμένα components σε συνεκτικά οικοσυστήματα εργαλείων, την αυτόματη συνεργασία μεταξύ υποσυστημάτων (π.χ. observability → scaling → policy enforcement),και τη δημιουργία “intelligent orchestration frameworks” που θα μπορούν να διαχειρίζονται το Kubernetes ολιστικά και όχι αποσπασματικά.

Από ερευνητικής σκοπιάς, το πεδίο αυτό μπορεί να οδηγήσει σε νέες προσεγγίσεις για knowledge-driven Kubernetes orchestration και semantic interoperability frameworks που θα γεφυρώνουν τα υπάρχοντα εργαλεία, καθιστώντας τα πιο αποτελεσματικά, επεκτάσιμα και διασυνδεδεμένα.

6.2 Προβλεπτική και Αυτόνομη Διαχείριση Πόρων

Η ανάλυση των εργαλείων στις κατηγορίες Autoscaling και Scheduling/AI-based Optimization ανέδειξε ότι, παρά τη σημαντική πρόοδο που έχει επιτευχθεί μέσω εργαλείων όπως τα HPA, KEDA, KOSMOS και DQN Scheduler, η διαχείριση πόρων στο Kubernetes παραμένει αντιδραστική και όχι προβλεπτική. Τα περισσότερα συστήματα βασίζονται σε απλούς μηχανισμούς παρακολούθησης (metrics-based scaling) ή σε προκαθορισμένα thresholds CPU και μνήμης, χωρίς τη χρήση προγνωστικών μοντέλων που να λαμβάνουν υπόψη την ιστορική συμπεριφορά των workloads, το είδος της εφαρμογής ή τις μεταβαλλόμενες συνθήκες του συστήματος.

Αυτό δημιουργεί ένα ουσιαστικό ερευνητικό κενό: την ανάγκη ανάπτυξης προβλεπτικών και αυτόνομων μηχανισμών διαχείρισης πόρων, οι οποίοι θα μπορούν να προσαρμόζονται δυναμικά στο περιβάλλον εκτέλεσης. Τα υπάρχοντα συστήματα, όπως το DQN Scheduler, έχουν δείξει ενθαρρυντικά αποτελέσματα σε πειραματικό επίπεδο, αλλά η εφαρμογή τους σε παραγωγικά περιβάλλοντα μεγάλης κλίμακας παραμένει περιορισμένη, λόγω του κόστους εκπαίδευσης των μοντέλων, της ανάγκης για αξιόπιστη συλλογή δεδομένων και της έλλειψης ενιαίου πλαισίου αξιολόγησης.

Η μελλοντική έρευνα μπορεί να κινηθεί προς τεσσερις κύριες κατευθύνσεις:

1. Ανάπτυξη Υβριδικών Μοντέλων Μάθησης
   Ενσωμάτωση Machine Learning και Reinforcement Learning τεχνικών σε υπάρχοντες μηχανισμούς autoscaling (π.χ. HPA, KEDA), ώστε η απόφαση κλιμάκωσης να βασίζεται όχι μόνο σε στιγμιαίες μετρικές, αλλά σε προγνωστικά μοντέλα φόρτου. Τέτοια μοντέλα μπορούν να μειώσουν τις καθυστερήσεις απόκρισης και να βελτιώσουν τη σταθερότητα των συστημάτων υπό μεταβαλλόμενο φόρτο.
2. Αυτόνομες Πολιτικές Διαχείρισης με Feedback Loops
   Ανάπτυξη αυτοπροσαρμοζόμενων control loops, τα οποία θα τροποποιούν δυναμικά τις πολιτικές κλιμάκωσης και χρονοπρογραμματισμού με βάση την πραγματική συμπεριφορά του συστήματος. Η ενσωμάτωση τέτοιων μηχανισμών θα επιτρέψει στα clusters να λειτουργούν αυτόνομα, ελαχιστοποιώντας την ανθρώπινη παρέμβαση και αποτρέποντας υπερεκχώρηση ή υποεκχώρηση πόρων.
3. Συνδυασμός Προβλεπτικών Μοντέλων με Πολυκριτηριακή Βελτιστοποίηση
   Πέρα από την πρόβλεψη φόρτου, τα μελλοντικά συστήματα θα πρέπει να λαμβάνουν υπόψη πολλαπλούς στόχους, όπως ενεργειακή κατανάλωση, κόστος εκτέλεσης και επιδόσεις εφαρμογής. Η χρήση τεχνικών multi-objective optimization μπορεί να οδηγήσει σε πιο ισορροπημένες αποφάσεις διαχείρισης πόρων.
4. Πειραματική Αξιολόγηση σε Παραγωγικά Περιβάλλοντα
   Υπάρχει ανάγκη για εμπειρικές μελέτες που θα αξιολογούν τα παραπάνω συστήματα σε πραγματικά workloads (π.χ. microservices, ML pipelines) και όχι μόνο σε προσομοιώσεις. Η έλλειψη τέτοιων δεδομένων αποτελεί σημαντικό εμπόδιο στη βιομηχανική υιοθέτηση προβλεπτικών μηχανισμών.

Η προβλεπτική και αυτόνομη διαχείριση πόρων παραμένει ένα ανεξερεύνητο αλλά κρίσιμο πεδίο στο οικοσύστημα Kubernetes. Η μετάβαση από τους αντιδραστικούς σε ευφυείς μηχανισμούς προσαρμογής μπορεί να οδηγήσει σε πιο αποδοτικά, βιώσιμα και αυτόνομα

cloud-native περιβάλλοντα, εφόσον συνδυαστεί με τεχνικές τεχνητής νοημοσύνης, ενιαία πρότυπα δεδομένων και πειραματική επικύρωση μεγάλης κλίμακας.

6.3 Policy-Aware Ασφάλεια και Κανονιστική Συμμόρφωση

Η αξιολόγηση των εργαλείων στις κατηγορίες Security, Service Mesh και Cluster Management ανέδειξε ότι, παρά την ύπαρξη ώριμων μηχανισμών ασφάλειας — όπως RBAC, mTLS, και Network Policies — η ασφάλεια στο Kubernetes παραμένει πολιτικο-αγνωστική (policy-agnostic). Με άλλα λόγια, οι περισσότερες λύσεις επικεντρώνονται στην *τεχνική υλοποίηση* των μηχανισμών ασφαλείας, χωρίς να ενσωματώνουν ή να τηρούν πολιτικές συμμόρφωσης και κανονιστικά πρότυπα (π.χ. GDPR, ISO 27001, NIST, SOC 2).

Παράλληλα, τα περισσότερα εργαλεία (π.χ. Istio, OPA/Gatekeeper) λειτουργούν μεμονωμένα, χωρίς ενιαίο μηχανισμό διακυβέρνησης πολιτικών σε επίπεδο cluster. Αυτό σημαίνει ότι, ενώ μπορούν να επιβάλουν επιμέρους κανόνες (policies), αδυνατούν να συνδέσουν επιχειρησιακές απαιτήσεις με τεχνικούς μηχανισμούς ελέγχου. Επιπλέον, η έλλειψη αυτοματοποιημένης επαλήθευσης συμμόρφωσης οδηγεί συχνά σε *χειροκίνητη παρακολούθηση και auditing*, κάτι που αυξάνει το λειτουργικό ρίσκο και μειώνει την εμπιστοσύνη σε περιβάλλοντα παραγωγής.

Αυτό συνιστά ένα σημαντικό ερευνητικό κενό: την ανάγκη για ανάπτυξη policy-aware μηχανισμών ασφάλειας, που συνδυάζουν τεχνικούς ελέγχους με τυποποιημένα πλαίσια συμμόρφωσης και προσφέρουν αυτοματοποιημένη, επαληθεύσιμη διακυβέρνηση.

Η μελλοντική έρευνα μπορεί να κινηθεί προς πέντε κύριες κατευθύνσεις:

1. Ενοποίηση Πολιτικών Ασφάλειας σε Ενιαίο Governance Layer
   Ανάπτυξη ενός ομοιόμορφου επιπέδου διακυβέρνησης (governance layer) που θα ενοποιεί τις πολιτικές από διαφορετικά συστήματα, όπως OPA, Kyverno, Istio και Kubernetes RBAC. Μέσω ενός ενιαίου μηχανισμού policy orchestration, οι διαχειριστές θα μπορούν να επιβάλλουν και να παρακολουθούν κανόνες σε πολλαπλά clusters και επίπεδα (network, identity, data).
2. Policy-as-Code με Αυτόματη Επαλήθευση Συμμόρφωσης
   Προώθηση του μοντέλου Policy-as-Code, όπου οι πολιτικές ασφάλειας και συμμόρφωσης εκφράζονται σε τυποποιημένη μορφή (π.χ. Rego, YAML schemas) και υπόκεινται σε αυτόματο έλεγχο και versioning. Με αυτόν τον τρόπο, οι οργανισμοί μπορούν να διασφαλίσουν τη συνεπή εφαρμογή των κανόνων και τη διαφάνεια στον κύκλο ζωής της πολιτικής.
3. Σημασιολογική Αντιστοίχιση Πολιτικών και Προτύπων Συμμόρφωσης
   Έρευνα πάνω σε σημασιολογικά μοντέλα (Semantic Models) και οντολογίες (Ontologies) που θα επιτρέπουν την αντιστοίχιση πολιτικών ασφαλείας με πρότυπα συμμόρφωσης (π.χ. αντιστοίχιση κανόνων OPA με GDPR άρθρα ή NIST ελέγχους). Η χρήση τέτοιων μοντέλων μπορεί να διευκολύνει την αυτοματοποιημένη τεκμηρίωση και auditing.
4. Ενσωμάτωση AI για Προγνωστική Ανίχνευση Μη Συμμόρφωσης
   Ανάπτυξη AI-based μηχανισμών ανάλυσης συμπεριφοράς που μπορούν να προβλέπουν

παραβιάσεις πολιτικής ή αποκλίσεις από πρότυπα συμμόρφωσης. Μέσω machine learning, τέτοια συστήματα μπορούν να αναγνωρίζουν ανωμαλίες και να ενεργοποιούν διορθωτικές ενέργειες πριν προκύψουν παραβιάσεις.

5. **Διαλειτουργικότητα Δεδομένων Ασφάλειας και Παρατηρησιμότητας**
   Δημιουργία **ενιαίων σχημάτων δεδομένων** (π.χ. OpenTelemetry for Security) για την ανταλλαγή πληροφοριών μεταξύ εργαλείων ασφάλειας, observability και compliance monitoring. Η διαλειτουργικότητα αυτή μπορεί να επιτρέψει ενοποιημένη ανάλυση και ενιαίο risk dashboard σε επίπεδο οργανισμού.

Η ασφάλεια στο Kubernetes κινείται σταδιακά από στατικές πολιτικές σε εξυπνότερα, context-aware συστήματα διακυβέρνησης. Ωστόσο, η έλλειψη σύνδεσης μεταξύ τεχνικών πολιτικών και επιχειρησιακών προτύπων συμμόρφωσης παραμένει κρίσιμο εμπόδιο. Η επόμενη γενιά εργαλείων οφείλει να συνδυάζει Policy-as-Code, AI-driven ανάλυση και σημασιολογική αντιστοίχιση κανονισμών, δημιουργώντας πλήρως policy-aware και αυτοπροσαρμοζόμενα περιβάλλοντα Kubernetes.

.

6.4 Αποδοτικότητα Κόστους και Ενεργειακή Βιωσιμότητα

Η συγκριτική αξιολόγηση των εργαλείων στις κατηγορίες Autoscaling, Monitoring, CI/CD και Cluster Management αποκάλυψε ότι, παρότι υπάρχει αυξανόμενο ενδιαφέρον για τη βελτιστοποίηση των επιδόσεων και τη σταθερότητα των clusters, η οικονομική και ενεργειακή διάσταση της λειτουργίας τους παραμένει υπομελετημένη.

Τα περισσότερα εργαλεία εστιάζουν στη λειτουργική αποδοτικότητα (π.χ. βέλτιστη χρήση CPU, μνήμης, ή bandwidth), χωρίς να ενσωματώνουν μηχανισμούς κοστολόγησης, ενεργειακής αποδοτικότητας ή ανάλυσης trade-offs μεταξύ επιδόσεων και κατανάλωσης. Εργαλεία όπως το HPA ή το KEDA μπορούν να κλιμακώνουν δυναμικά εφαρμογές, αλλά οι αποφάσεις αυτές λαμβάνονται χωρίς ρητή γνώση του κόστους χρήσης ή του ενεργειακού αποτυπώματος των pods και των nodes.

Αντίστοιχα, τα εργαλεία παρακολούθησης (π.χ. Prometheus, Grafana) προσφέρουν πλούσιες μετρικές, αλλά δεν τις αξιοποιούν για οικονομική βελτιστοποίηση ή πράσινη λειτουργία (green computing). Αυτό συνιστά ένα ουσιαστικό ερευνητικό κενό, καθώς η έλλειψη οικονομοτεχνικής ορατότητας περιορίζει τη δυνατότητα των οργανισμών να λαμβάνουν τεκμηριωμένες αποφάσεις για την αποδοτική και βιώσιμη χρήση πόρων.

Η μελλοντική έρευνα μπορεί να κινηθεί προς πέντε κύριες κατευθύνσεις:

1. Ενοποίηση Μηχανισμών Κοστολόγησης και Autoscaling
   Ανάπτυξη εργαλείων που θα συνδυάζουν δεδομένα κόστους (cost metrics) με τους μηχανισμούς autoscaling. Για παράδειγμα, η απόφαση δημιουργίας ή τερματισμού pods θα μπορούσε να λαμβάνει υπόψη όχι μόνο το φορτίο αλλά και την οικονομική αποδοτικότητα ανά node ή region. Αυτό θα επιτρέψει τη διαμόρφωση cost-aware policies που ελαχιστοποιούν τη δαπάνη χωρίς υποβάθμιση της ποιότητας υπηρεσίας.
2. Ενσωμάτωση Μετρικών Ενεργειακής Κατανάλωσης
   Ανάπτυξη εργαλείων παρακολούθησης που θα μετρούν την ενεργειακή κατανάλωση σε επίπεδο node, pod και workload, αξιοποιώντας δεδομένα από hardware counters ή cloud APIs (π.χ. AWS, GCP energy metrics). Η ενσωμάτωση αυτών των δεδομένων στο observability stack θα επιτρέψει την ανάπτυξη energy-aware schedulers.
3. Ανάπτυξη Predictive Cost Models και Green Autoscalers
   Δημιουργία προβλεπτικών μοντέλων κόστους που θα χρησιμοποιούν ιστορικά δεδομένα για να εκτιμήσουν τη μελλοντική κατανάλωση και να βελτιστοποιούν προληπτικά τη χρήση πόρων. Επιπλέον, οι green autoscalers θα μπορούν να προσαρμόζουν τη χρήση υποδομών ανάλογα με το ενεργειακό αποτύπωμα (π.χ. μεταφορά workloads σε περιοχές με χαμηλότερο ενεργειακό κόστος ή ανανεώσιμες πηγές).
4. Οικονομοτεχνική Παρατηρησιμότητα (FinOps για Kubernetes)
   Δημιουργία FinOps-oriented dashboards που θα συνδυάζουν τεχνικές και οικονομικές μετρικές (π.χ. $/pod, $/GB RAM, $/req). Η ενσωμάτωση αυτών των δεδομένων σε υπάρχοντα monitoring εργαλεία (όπως Prometheus και Grafana) μπορεί να προσφέρει πλήρη οικονομική διαφάνεια και να ενισχύσει τη λήψη στρατηγικών αποφάσεων.

5. Πολυκριτηριακή Βελτιστοποίηση Ενέργειας–Κόστους–Απόδοσης
   Προώθηση αλγορίθμων που θα λαμβάνουν ταυτόχρονα υπόψη τρεις διαστάσεις: ενέργεια, κόστος και απόδοση. Μέσω τεχνικών multi-objective optimization και machine learning, μπορεί να επιτευχθεί ισορροπία ανάμεσα στη λειτουργική αποδοτικότητα και τη βιωσιμότητα των cloud-native περιβαλλόντων.

Η αποδοτικότητα κόστους και η ενεργειακή βιωσιμότητα αποτελούν αναδυόμενα αλλά κρίσιμα πεδία έρευνας στο οικοσύστημα Kubernetes. Η μετάβαση προς cost-aware και energy-aware υποδομές προϋποθέτει τη σύζευξη οικονομοτεχνικών, ενεργειακών και λειτουργικών δεδομένων σε ενιαίο πλαίσιο διαχείρισης. Οι μελλοντικές λύσεις θα πρέπει να συνδυάζουν FinOps πρακτικές, ενεργειακή παρακολούθηση και ευφυή autoscaling, ώστε να διαμορφώσουν ένα βιώσιμο, αποδοτικό και περιβαλλοντικά υπεύθυνο Kubernetes οικοσύστημα.

6.5 Ωριμότητα και Βιωσιμότητα των Εργαλείων Ανοικτού Κώδικα

Η ανάλυση που πραγματοποιήθηκε στις κατηγορίες εργαλείων του οικοσυστήματος Kubernetes ανέδειξε μια σημαντική ανισορροπία μεταξύ των έργων ανοικτού κώδικα ως προς την ωριμότητα, τη βιωσιμότητα και τη μακροπρόθεσμη υποστήριξή τους. Ενώ ορισμένα έργα όπως το Prometheus, το Argo CD, το Istio ή το KEDA έχουν εξελιχθεί σε ώριμα, CNCF-graduated projects με εκτενή τεκμηρίωση και ενεργές κοινότητες, άλλα — όπως το KubePipe, το K8sGPT ή ο DQN Scheduler — παραμένουν σε πειραματικό ή ερευνητικό στάδιο, με περιορισμένη τεχνική υποστήριξη, σπάνια releases και ελλιπή τεκμηρίωση.

Αυτή η ασυμμετρία δημιουργεί ένα ουσιαστικό ερευνητικό και πρακτικό κενό: η απουσία μηχανισμών αξιολόγησης και διασφάλισης βιωσιμότητας για έργα ανοικτού κώδικα που χρησιμοποιούνται σε κρίσιμες παραγωγικές υποδομές.
Η βιωσιμότητα των έργων δεν εξαρτάται μόνο από την τεχνική τους ποιότητα, αλλά και από παράγοντες όπως η κοινότητα, το μοντέλο συνεισφοράς, η συχνότητα ενημερώσεων και η εμπορική υποστήριξη. Παρά ταύτα, η έρευνα γύρω από το πώς αυτά τα χαρακτηριστικά μπορούν να μετρηθούν, να προβλεφθούν ή να ενισχυθούν παραμένει περιορισμένη.

Η μελλοντική έρευνα μπορεί να κινηθεί προς τέσσερις κύριες κατευθύνσεις:

1. Ανάπτυξη Μοντέλων Αξιολόγησης Ωριμότητας (Open-Source Maturity Models)
   Δημιουργία τυποποιημένων πλαισίων αξιολόγησης που θα λαμβάνουν υπόψη τεχνικούς (π.χ. κώδικας, τεκμηρίωση, ασφάλεια) και κοινοτικούς δείκτες (π.χ. αριθμός συνεισφερόντων, ρυθμός releases, governance). Τα μοντέλα αυτά θα μπορούσαν να επεκτείνουν υπάρχοντα πρότυπα, όπως το CNCF Maturity Framework, ώστε να παρέχουν ποσοτικοποιημένους δείκτες βιωσιμότητας.
2. Ενοποίηση Ερευνητικών και Παραγωγικών Κοινοτήτων
   Ειδικά για έργα που προέρχονται από ακαδημαϊκά περιβάλλοντα (π.χ. DQN Scheduler, KubePipe), προτείνεται η γεφύρωση μεταξύ έρευνας και βιομηχανίας μέσω κοινοτήτων πρακτικής, shared repositories και CNCF sandbox προγραμμάτων. Αυτή η προσέγγιση θα μπορούσε να ενισχύσει τη βιωσιμότητα των ερευνητικών εργαλείων μετατρέποντάς τα σε παραγωγικά έργα.

3. Εφαρμογή Προγνωστικών Δεικτών Βιωσιμότητας (Sustainability Forecasting)
   Ανάπτυξη μηχανισμών που θα προβλέπουν τη βιωσιμότητα ενός έργου βάσει ιστορικών δεδομένων (activity trends, commit frequency, issue closure rates). Τέτοια μοντέλα μπορούν να αξιοποιήσουν machine learning για την εκτίμηση κινδύνου εγκατάλειψης (project abandonment risk) και να παρέχουν early warning indicators για οργανισμούς που το υιοθετούν
4. Οικοσυστημική Προσέγγιση στην Αξιολόγηση Βιωσιμότητας
   Αντί της απομονωμένης ανάλυσης κάθε εργαλείου, προτείνεται η οικοσυστημική προσέγγιση, όπου η βιωσιμότητα ενός έργου εκτιμάται σε συνάρτηση με τη θέση του στο CNCF landscape και τις εξαρτήσεις του (dependencies). Ένα εργαλείο μπορεί να είναι τεχνικά ώριμο, αλλά αν εξαρτάται από μη συντηρούμενες βιβλιοθήκες, η συνολική βιωσιμότητά του υπονομεύεται.

Η ωριμότητα και βιωσιμότητα των έργων ανοικτού κώδικα αποτελούν καθοριστικό παράγοντα για την αξιοπιστία και τη μακροχρόνια επιτυχία του οικοσυστήματος Kubernetes. Η απουσία ενιαίων μεθόδων μέτρησης, προγνωστικών μηχανισμών και διασύνδεσης κοινοτήτων οδηγεί σε αβεβαιότητα σχετικά με τη συντηρησιμότητα και τη μελλοντική υιοθέτηση των εργαλείων. Η μελλοντική έρευνα οφείλει να κινηθεί προς την κατεύθυνση ανάπτυξης ποσοτικών δεικτών ωριμότητας, προγνωστικών μοντέλων βιωσιμότητας συμβάλλοντας έτσι στη διαμόρφωση ενός βιώσιμου οικοσυστήματος ανοικτού λογισμικού

ΚΕΦΑΛΑΙΟ 7 – ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

7.1 Βασική Συνεισφορά

Η παρούσα διπλωματική εργασία συνέβαλε στην **συστηματική επισκόπηση και κατηγοριοποίηση του οικοσυστήματος εργαλείων Kubernetes**. Μέσα από ιεραρχική κατηγοριοποίηση και ανάλυση εννιά λειτουργικών κατηγοριών (π.χ. Monitoring, Security, CI/CD, Service Mesh), παρουσιάστηκε μια ολιστική εικόνα των δυνατοτήτων, περιορισμών και προοπτικών των εργαλείων. Η συνεισφορά εντοπίζεται επίσης στη **συγκριτική αξιολόγηση βάσει κοινών κριτηρίων** (απόδοση, ενσωμάτωση, επεκτασιμότητα, ασφάλεια, διαλειτουργικότητα, υποστήριξη), κάτι που επιτρέπει μια πιο αντικειμενική σύγκριση ανά κατηγορία ενώ οδήγησε στην παραγωγή πολλών χρήσιμων συμπερασμάτων τόσο ανά λειτουργική κατηγορία εργαλείων όσο και σε καθολικό επίπεδο. Η κατηγοριοποίηση και συγκριτική αξιολόγηση των εργαλείων Kubernetes επιτρέπει την ταχύτερη κατανόηση του οικοσυστήματος και τη στοχευμένη επιλογή λύσεων ανάλογα με τις ανάγκες του εκάστοτε οργανισμού ή ερευνητικού πλαισίου. Τέλος, το Κεφάλαιο 6 ανέδειξε τα **ερευνητικά κενά και τις κατευθύνσεις** που μπορούν να καθοδηγήσουν τις μελλοντικές μελέτες.

7.2 Εμπειρία από την Εκπόνηση της Εργασίας

Κατά την εκπόνηση της διπλωματικής αποκτήθηκε σε βάθος κατανόηση του **cloud-native οικοσυστήματος** και της πολυπλοκότητας που χαρακτηρίζει τα εργαλεία Kubernetes. Η διαδικασία ανέδειξε την ανάγκη για **κριτική αξιολόγηση βιβλιογραφίας**, τη σημασία της χρήσης αξιόπιστων πηγών (π.χ. IEEE, ACM, MDPI, CNCF landscape) και την πρόκληση της οργάνωσης μεγάλου όγκου πληροφοριών σε μια συνεκτική ανάλυση. Παράλληλα, η εργασία ανέδειξε και την **πρακτική διάσταση**: η πολυπλοκότητα ενσωμάτωσης ορισμένων εργαλείων και η έλλειψη ώριμης τεκμηρίωσης έδωσαν πολύτιμη εμπειρία για την αξιολόγηση εργαλείων σε πραγματικά περιβάλλοντα.

7.3 Βασικά Συμπεράσματα

Η ανάλυση έδειξε ότι:

- Οι πιο κρίσιμες προκλήσεις σήμερα σχετίζονται με την **ασφάλεια**, τη **διαλειτουργικότητα** εργαλείων και την **πολυπλοκότητα ενσωμάτωσης**.
- Η έλλειψη αξιολόγησης σε μεγάλης κλίμακας clusters και η απουσία οικονομικών μελετών (TCO/ROI) καθιστούν δύσκολη τη λήψη στρατηγικών αποφάσεων για οργανισμούς.
- Η ενσωμάτωση **AI/ML** εμφανίζεται ως μία από τις πιο πολλά υποσχόμενες κατευθύνσεις, καθώς μπορεί να ενισχύσει το autoscaling, την αυτόματη παραμετροποίηση, την πρόβλεψη σφαλμάτων και την αποδοτικότητα διαχείρισης.
- Η ενίσχυση των **open-source κοινοτήτων** για εργαλεία μικρότερης εμβέλειας αποτελεί αναγκαία προϋπόθεση για τη βιωσιμότητά τους.

Συνολικά, η εργασία κατέδειξε ότι το οικοσύστημα Kubernetes βρίσκεται σε φάση ωρίμανσης: υπάρχουν ώριμες λύσεις σε κατηγορίες, όπως το monitoring και το CI/CD, αλλά και περιοχές που χρειάζονται περισσότερη έρευνα, ιδιαίτερα γύρω από την ασφάλεια, τη δικτύωση (networking) και τη χρήση ML.

7.4 Μελλοντική Εργασία

Με βάση τα παραπάνω, μελλοντική έρευνα μπορεί να εστιάσει σε:

- Ανάπτυξη **ενιαίων προτύπων και APIs** για καλύτερη διαλειτουργικότητα.
- Ενσωμάτωση εργαλείων με **compliance frameworks** για αυτόματη συμμόρφωση.
- Οικονομικές μελέτες κόστους–απόδοσης για την τεκμηριωμένη επιλογή εργαλείων.
- Εφαρμογές **AI/ML** που συνδυάζουν προβλεπτική ανάλυση με αυτοματοποιημένη διαχείριση πόρων.

Βιβλιογραφία

Jin, R., Muench, P., Janssen, T., Hatfield, B., & Deenadhayalan, V. (2024). Baking Disaster-Proof Kubernetes Applications with Efficient Recipes. *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering* (pp. 174–180). Association for Computing Machinery (ACM).

Ahmad, W., Rasool, A., Javed, A., Baker, T., & Jalil, Z. (2021, December). Cyber Security in IoT-Based Cloud Computing: A Comprehensive Survey. *Electronics, 11*(1), p. 16. Retrieved from https://www.mdpi.com/2079-9292/11/1/16

Alghofaili, Y., Albattah , A., Alrajeh, N., Rassam, M., & Al-rimy, B. (2021, September). Secure Cloud Infrastructure: A Survey on Issues, Current Solutions, and Open Challenges. *Applied Sciences, 11*(19), p. 9005. Retrieved from https://www.mdpi.com/2076-3417/11/19/9005

Ali, A., Imran, M., Kuznetsov, V., Trigazis, S., Pervaiz, A., Pfeiffer, A., & Mascheroni, M. (2024). Implementation of New Security Features in CMSWEB Kubernetes Cluster at CERN. *26th International Conference on Computing in High Energy and Nuclear Physics.* Retrieved from https://doi.org/10.1051/epjconf/202429507026

Amgothu, S., & Kankanala, G. (2024, December). Enhancing Kubernetes Security: Securing Workloads and Optimizing Role-Based Access Control. *International Journal of Computer Applications (IJCA), 186*(58), pp. 11-15. Retrieved from https://doi.org/10.5120/ijca2024924336

Andreazi, G., Estrella, J., Bruschi, S., Immich, R., Guidoni, D., Pereira Júnior, L., & Meneguette, R. (2021, October). MoHRiPA—An Architecture for Hybrid Resources Management of Private Cloud Environments. *Sensors, 21*(20), p. 6857. Retrieved from https://www.mdpi.com/1424-8220/21/20/6857

Aqua Security. (2024). *Aqua Security Trivy — Official Documentation (v.0.50)*. Retrieved from Trivy: https://trivy.dev/v0.50/

Argo CD project. (n.d.). Argo CD Documentation. Retrieved from https://argo-cd.readthedocs.io/

Argo Workflows — Documentation. (2023). *Read the Docs*. Retrieved from https://argo-workflows.readthedocs.io/en/latest/

Augustyn, D., Wyciślik, Ł., & Sojka, M. (2024, January). Tuning a Kubernetes Horizontal Pod Autoscaler for Meeting Performance and Load Demands in Cloud Deployments. *Applied Sciences, 14*(2), p. 646. Retrieved from https://doi.org/10.3390/app14020646

Baresi, L., Hu, D., Quattrocchi, G., & Terracciano, L. (2021). KOSMOS: Vertical and Horizontal Resource Autoscaling for Kubernetes. *Proceedings of the 19th International Conference on Service-Oriented Computing (ICSOC). 13121*, pp. 821-829. Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-030-91431-8_59

Bergh, A., Smith, T., & Guevara, G. (2022). Kasten K10-Kubernetes-native backup/restore, disaster recovery, and application migration. SUSE LLC. Retrieved from https://documentation.suse.com/en-us/trd/veeam/html/gs_rancher_veeam-kasten/index.html

Boettiger, C. (2015, January). An Introduction to Docker for Reproducible Research. *SIGOPS Operating Systems Review, a publication of the ACM Special Interest Group on Operating Systems (SIGOPS), 49*(1), pp. 71–79. Retrieved from https://doi.org/10.1145/2723872.2723882

Budigiri, G., Baumann, C., Mühlberg, J., Truyen, E., & Joosen, W. (2021). Network Policies in Kubernetes: Performance Evaluation and Security Analysis. *Joint European Conference on Networks and Communications & 6G Summit* (pp. 407–412). IEEE.

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016, April). Borg, Omega, and Kubernetes. *Communications of the ACM, 59*(5), pp. 50 - 57. Retrieved from https://dl.acm.org/doi/10.1145/2890784

Chinamanagonda, S. (2021, June). Container Security: Best Practices and Tools – Rising Concerns and Solutions for Securing Containerized Environments. *International Journal of Novel Research and Development (IJNRD), 6*(6), pp. 41-46.

Cilium. (2025, July). Network Policy (Cilium documentation - Kubernetes Networking). Retrieved from https://docs.cilium.io/en/stable/network/kubernetes/policy/

Cloud Native Computing Foundation (CNCF). (2024). CNCF Cloud Native Interactive Landscape (visual mapping portal). Retrieved from https://landscape.cncf.io

Cloud Native Computing Foundation (CNCF). (2024). Documentation. Cloud Native Computing Foundation (CNCF) / Kubernetes Project documentation. Retrieved from https://kubernetes.io/docs/home/

Cloud Native Computing Foundation (CNCF). (2024). Documentation. Ανάκτηση από https://kubernetes.io/docs/home/

Cloud Native Computing Foundation. (2023). *Cloud Native 2023: The Undisputed Infrastructure of Global Technology.* Linux Foundation.

CNCF TAG App Delivery Operator Working Group. (2021). *CNCF Operator White Paper (Version 1.0).* CNCF TAG App Delivery, Cloud Native Computing Foundation.

CNCF TAG Security. (2022). *Cloud Native Security Whitepaper (Version 2.0).* CNCF (Cloud Native Computing Foundation) TAG Security Group.

Dakić, V., Dambić, G., Slovinac, J., & Redžepagić, J. (2025, February). Optimizing Kubernetes Scheduling for Web Applications Using Machine Learning. *Electronics, 14*(5). Retrieved from https://doi.org/10.3390/electronics14050863

Dakić, V., Redžepagić, J., Bašić, M., & Žgrablić , L. (2024, October). Performance and Latency Efficiency Evaluation of Kubernetes Container Network Interfaces for Built-In and Custom Tuned Profiles. *Electronics, 13*(19).

Drone CI / CD – Documentation. (n.d.). Retrieved from Official Drone Documentation Website: https://docs.drone.io/

Dubey, K., Shams, M., Sharma, S., Alarifi, A., Amoon, M., & Nasr, A. (2019, October). A Management System for Servicing Multi-Organizations on Community Cloud Model in Secure Cloud Environment. *IEEE Access, 7*.

Erdenebat, B., Bud, B., & Kozsik, T. (2023, October). Challenges in Service Discovery for Microservices Deployed in a Kubernetes Cluster – A Case Study. *Infocommunications, 15*(5), pp. 69-75.

Fehling, C., Leymann, F., Retter, R., Schupeck , W., & Arbitter, P. (2014). *Cloud Computing Patterns.* Springer Vienna.

Fritzsche, P., Cano, A., Garcia, G., Singh, A., & Mozos, M. (2024). Lightweight Machine Learning for Edge Learning in Kubernetes Clusters. *Proceedings of the 2024 IEEE/ACM 17th International Conference on Utility and Cloud Computing (UCC)* (pp. 88–93). Institute of Electrical and Electronics Engineers (IEEE).

Galij, S., Pawlak, G., & Grzyb, S. (2024, November). Modeling Data Sovereignty in Public Cloud—A Comparison of Existing Solutions. *Applied Sciences, 14*(23), p. 10803. Retrieved from https://www.mdpi.com/2076-3417/14/23/10803

GoCD User Documentation. (n.d.). *ThoughtWorks*. Retrieved from Official GoCD Documentation Website: https://docs.gocd.org/current/

Gogineni, A. (2022, September). Multi-Cloud Deployment with Kubernetes: Challenges, Strategies, and Performance Optimization. *International Scientific Journal of Engineering and Management, 1*(2).

Google Cloud Platform. (2025, July). GKE networking overview (Google Kubernetes Engine Concepts). Google Cloud Platform. Retrieved from https://cloud.google.com/kubernetes-engine/docs/concepts/network-overview

Gupta, S., Bhatia, M., Memoria, M., & Manani, P. (2022). Prevalence of GitOps, DevOps in Fast CI/CD Cycles. *International Conference on Machine Learning, Big Data, Cloud and Parallel Computing*, (pp. 589–596).

Guruge, P., & Priyadarshana, Y. (2025, February). Time series forecasting-based Kubernetes autoscaling using Facebook Prophet and Long Short-Term Memory. *Frontiers in Computer Science, 7*. Retrieved from https://doi.org/10.3389/fcomp.2025.1509165

Hadikusuma, R., Lukas, & Bachri, K. (2022). Optimization and Monitoring of Kubernetes Cluster using Various Approaches. *International Journal of Engineering Research & Technology (IJERT), 11*(5).

Hämäläinen, H., Rantanen, I., Aalto, S., & Pum, M. (2021, November). Monitoring and Observability in Kubernetes Clusters Using Prometheus and Grafana. Retrieved from https://www.researchgate.net/publication/392124972_Monitoring_and_Observability_in_Kubernetes_Clusters_Using_Prometheus_and_Grafana

HashiCorp. (2025). *HashiCorp Vault — Official Documentation*. Retrieved from HashiCorp: https://developer.hashicorp.com/vault/docs

Hecht, L. (2024, June). *Kubernetes: 48% of Users Struggle With Tool Choice*. Retrieved from The New Stack: https://thenewstack.io/kubernetes-48-of-users-struggle-with-tool-choice/

Helm community. (2025). Helm Documentation. Retrieved from https://helm.sh/docs/

Hightower, K., Burns, B., & Beda, J. (2017). *Kubernetes: Up and Running: Dive into the Future of Infrastructure.* O'Reilly Media.

Hoffpauir, K., Simmons, J., Schmidt, N., Pittala, R., Briggs, I., Makani, S., & Jararweh, Y. (2023, June). A Survey on Edge Intelligence and Lightweight Machine Learning Support for Future Applications and Service. *Journal of Data and Information Quality, 15*(2). Retrieved from https://doi.org/10.1145/3581759

Hwang, Y. (2023, July). *Medium.* Retrieved from State of Kubernetes 2023: https://yitaek.medium.com/state-of-kubernetes-2023-2732a4cba77d

Ibryam, B., & Huß, R. (2019). *Kubernetes Patterns: Reusable Elements for Designing Cloud-Native Applications.* O'Reilly Media. Retrieved from https://www.oreilly.com/library/view/kubernetes-patterns/9781492050292/

Imhotep Software LLC. (2024). K9s: Kubernetes CLI UI for Managing and Observing Kubernetes Clusters. Retrieved from https://k9scli.io/

International Organization for Standardization (ISO) & International Electrotechnical Commission (IEC). (2011). *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.* Geneva, Switzerland. Retrieved from https://www.iso.org/standard/35733.html

Jakkaraju, V. (2020, June). Adversarial-Aware Kubernetes Admission Controllers for Real-Time Threat Suppression. *International Journal of Intelligent Systems and Applications in Engineering, 8*(2), pp. 143-151.

Jenkins User Documentation. (n.d.). Retrieved from Official Jenkins Documentation Website: https://www.jenkins.io/doc/

Johansson, W. (2022, May). *A Comparison of CI/CD tools on.* Umeå University, Computing Science.

Johng, H., Kalia, A., Xiao, J., Vuković, M., & Chung, L. (2019). Harmonia: A Continuous Service Monitoring Framework Using DevOps and Service Mesh in a Complementary Manner. *17th International Conference, ICSOC* (pp. 151–168). Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-030-33702-5_12

KEDA Project Documentation. (2024). KEDA. Retrieved from https://keda.sh/

Khan, M., Ullah, F., Imran, M., Khan, J., Khan , A., AlGhamdi, A., & Alshamrani, S. (2022, August). Identifying Challenges for Clients in Adopting Sustainable Public Cloud Computing. *Sustainability, 14*(16), p. 9809. Retrieved from https://www.mdpi.com/2071-1050/14/16/9809

Kim, J., & Jung, E.-S. (2024, January). Survey on Backup and Recovery Tools and Multicluster Management Platforms in a Kubernetes. *IEIE Transactions on Smart Processing & Computing, 13*(5), pp. 499–513.

Kim, J.-B., Choi, J.-B., & Jung, E.-S. (2024, May). Design and Implementation of an Automated Disaster-Recovery System for a Kubernetes Cluster Using LSTM. *Applied Sciences, 14*(9). Retrieved from https://doi.org/10.3390/app14093914

Korada, L., & Sikha, V. (2024, February). Why are large enterprises building private clouds after their journey on public clouds? *11*(2), pp. 49-52. Retrieved from https://doi.org/10.5281/zenodo.13326719

Kubernetes Documentation. (2024). *Using CoreDNS for Service Discovery*. Retrieved from Kubernetes: https://kubernetes.io/docs/tasks/administer-cluster/coredns/

Matranga, S. (2024). *Scheduling Kubernetes Tasks with Reinforcement Learning.* Politecnico di Torino, Master's Degree in Computer Engineering. Retrieved from https://doi.org/10.1007/978-3-030-91431-8_59

Mayr, A., & Putz, P. (2023, January). Kubernetes in the Wild report 2023. *Dynatrace Blog*. Retrieved from https://www.dynatrace.com/news/blog/kubernetes-in-the-wild-2023/

Mell, P., & Grance, T. (2011, September). The NIST Definition of Cloud. *Scientific Research Publishing (SCIRP), 800*(145).

Menouer, T., Cérin, C., & Darmon, P. (2024). Reactive Autoscaling of Kubernetes Nodes. *4th Workshop on Flexible Resource and Application Management on the Edge* (pp. 31–38). Association for Computing Machinery (ACM). Retrieved from https://doi.org/10.1145/3659994.3660310

Merkel, D. (2014, March). Docker: lightweight Linux containers for consistent development and deployment. *Linux Journal, 2014*(239).

Miziński, K., & Przyłucki, S. (2025). The impact of using eBPF technology on the performance of networking solutions in a Kubernetes cluster. *Journal of Computer Sciences Institute, 35*.

Molleti, R. (2022, December). Kubernetes Advanced Auto Scaling Techniques. *Journal of Mathematical & Computer Applications, 1*(4), pp. 1–4. Retrieved from https://doi.org/10.47363/JMCA/2022(1)E126

Mondal, S., Zheng, Z., & Cheng, Y. (2024, August). On the Optimization of Kubernetes toward the Enhancement of Cloud Computing. *Mathematics, 12*(16). Retrieved from https://www.mdpi.com/2227-7390/12/16/2476

Morić, Z., Dakić, V., & Čavala, T. (2025, June). Security Hardening and Compliance Assessment of Kubernetes Control Plane and Workloads. *Journal of Cybersecurity and Privacy, 5*(2).

Nguyen, Q.-M., Phan, L.-A., & Kim, T. (2022, April). Load-Balancing of Kubernetes-Based Edge Computing Infrastructure Using Resource Adaptive Proxy. *Sensors, 22*(8).

Operator Framework project. (2024). Operator SDK Documentation. Retrieved from https://sdk.operatorframework.io/docs/

Operator Framework project. (n.d.). Operator Framework. Retrieved from https://operatorframework.io/

Osmani, L., Kauppinen, T., Komu, M., & Tarkoma, S. (2021, November). Multi-Cloud Connectivity for Kubernetes in 5G Networks. *IEEE Communications Magazine, 59*(10), pp. 42–47.

Pai , K., & Prof.Srinivas, B. (2024, October). Enhanced Visibility for Real-time Monitoring and Alerting in Kubernetes by Integrating Prometheus, Grafana, Loki, and Alerta. *International Journal of Scientific Research in Engineering and Management (IJSREM)*.

Petersen, K., Vakkalanka, S., & Kuzniarz, L. (2015, August). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology, 64*, pp. 1-18.

Poniszewska-Marańda, A., & Czechowska, E. (2021, March). Kubernetes Cluster for Automating Software Production Environment. *Sensors, 21*(5). Retrieved from https://www.mdpi.com/1424-8220/21/5/1910

PORTAINER.IO. (2024). Portainer: Kubernetes, Docker, and Podman container management platform. Retrieved from https://www.portainer.io/

Prometheus Authors. (n.d.). Prometheus Documentation. Retrieved from https://prometheus.io/docs

Puutio, K. (2025). *Enhancing Kubernetes cluster troubleshooting efficiency with K8sGPT.* Tampere University, Faculty of Information Technology and Communication Sciences.

Rad, B., Bhatti, H., & Ahmadi, M. (2017, March). An Introduction to Docker and Analysis of its Performance. *International Journal of Computer Science and Network Security (IJCSNS), 17*(3), pp. 228–229.

Ramadoni, E., Utami, E., & Fatta, H. (2021). Analysis on the Use of Declarative and Pull-based Deployment Models on GitOps Using Argo CD. *4th International Conference on Information and Communications Technology (ICOIACT)* (pp. 186-191). Institute of Electrical and Electronics Engineers (IEEE). Retrieved from https://ieeexplore.ieee.org/document/9563984

Reddy J, S., Padal S, D., Mayan, J., Catharine A., & Shanthamalar, J. (2024). Efficient Application Deployment: GitOps for Faster and Secure CI/CD Cycles. *International Conference on Advances in Modern Age Technologies for Health and Engineering Science (AMATHE)* (pp. 1-7). Institute of Electrical and Electronics Engineers (IEEE). Retrieved from https://ieeexplore.ieee.org/document/10582118

Redhat. (2023, January). Retrieved from Understanding cloud computing: https://www.redhat.com/en/topics/cloud-computing

Ross, R. (n.d.). *Kubernetes: Your Hybrid Cloud Strategy.* Google Cloud.

Run:AI. (n.d.). *Scaling Up AI/ML with Kubernetes.* Run:AI.

Sedghpour, M., & Townend, P. (2022). Service Mesh and eBPF-Powered Microservices: A Survey and Future Directions. *International Conference on Service-Oriented System Engineering (SOSE)* (pp. 176–184). Institute of Electrical and Electronics Engineers (IEEE). Retrieved from https://doi.org/10.1109/SOSE55356.2022.00027

Shamim, S., Gibson, J., Morrison, P., & Rahman, A. (2022). *Benefits, Challenges, and Research Topics: A Multi-vocal Literature Review of Kubernetes.* Tennessee Tech University, Department of Computer Science. arXiv.

Souppaya, M., Morello, J., & Scarfone, K. (2017, December). *Application Container Security Guide (NIST Special Publication 800-190).* National Institute of Standards and Technology (NIST). Retrieved from https://doi.org/10.6028/NIST.SP.800-190

Suárez, D., Almeida, F., Blanco, V., & Toledo, P. (2025, January). KubePipe: a container-based high-level parallelization tool for scalable machine learning pipelines. *The Journal of Supercomputing, 81*. Retrieved from https://doi.org/10.1007/s11227-025-06956-x

Syrigos, I., Makris, N., & Korakis, T. (2023). Multi-cluster orchestration of 5G experimental deployments in Kubernetes over high-speed fabric. (pp. 1764–1769). Institute of Electrical and Electronics Engineers (IEEE). Retrieved from https://ieeexplore.ieee.org/document/10465054

Tigera. (2025). Get started with Kubernetes network policy. Retrieved from https://docs.tigera.io/calico/latest/network-policy/get-started/kubernetes-policy/kubernetes-network-policy

Toka, L., Dobreff, G., Fodor, B., & Sonkoly, B. (2021, March). Machine Learning-Based Scaling Management for Kubernetes Edge Clusters. *IEEE Transactions on Network and Service Management, 18*(1), pp. 958–972.

Turek, T., Dziembek, D., & Hernes, M. (2021, October). The Use of IT Solutions Offered in the Public Cloud to Reduce the City's Carbon Footprint. *Energies, 14*(19), p. 6389. Retrieved from https://www.mdpi.com/1996-1073/14/19/6389

Velero. (2025). *Velero — Documentation*. Retrieved from Velero: https://velero.io/docs/v1.16/

Verma, S., Pandey, B., & Gupta, B. (2023, June). Containerization and its Architectures: A Study. *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal, 11*, pp. 395–409.

Wang, Z., Zhu, J., Guo, J., & Liu , Y. (2025, May). Microservice Deployment Based on Multiple Controllers for User Response Time Reduction in Edge-Native Computing. *Sensors, 25*(10). Retrieved from https://doi.org/10.3390/s25103248

Wen, H., Cao, Z., Li, B., Du, D., Abouelwafa, A., Voigt, D., . . . Wu , F. (2023). K8sES: Optimizing Kubernetes with Enhanced Storage Service-Level Objectives. *2023 IEEE 41st International Conference on Computer Design (ICCD)* (pp. 214–222). Institute of Electrical and Electronics Engineers (IEEE).

Wennerström, W. (2021). *Active Assurance in Kubernetes.* Independent Thesis , Luleå University of Technology, , Department of Computer Science, Electrical and Space Engineering. Retrieved from https://www.diva-portal.org/smash/get/diva2:1519453/FULLTEXT01.pdf

Wikipedia. (2024). Retrieved from https://en.wikipedia.org/wiki/Cloud_computing

Xu, Q., Gao, Y., & Wei, J. (2024). An Empirical Study on Kubernetes Operator Bugs. *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 1746–1758). Association for Computing Machinery (ACM).

Yang, H., & Kim, Y. (2022, November). Design and Implementation of Machine Learning-Based Fault Prediction System in Cloud Infrastructure. *Electronics, 11*(22). Retrieved from https://doi.org/10.3390/electronics11223765

Yu, J., Naganuma, Y., & Ide, T. (2020). System Operator: A Tool for System Management in Kubernetes Clusters. *The Eleventh International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2020)* (pp. 83–86). International Academy, Research, and Industry Association (IARIA).

Zhao, L., Hu, G., & Xu, Y. (2024, September). Educational Resource Private Cloud Platform Based on OpenStack. *Computers, 13*(9), p. 241. Retrieved from https://www.mdpi.com/2073-431X/13/9/241

Zhu, M., Kang, R., He, F., & Oki, E. (2021). Implementation of Backup Resource Management Controller for Reliable Function Allocation in Kubernetes. *IEEE 7th International Conference on Network Softwarization* (pp. 360–362). IEEE (Institute of Electrical and Electronics Engineers).

Zhu, X., She, G., Xue, B., Zhang, Y., Zhang, Y., Zou, X., . . . Mahajan, R. (2023). Dissecting Overheads of Service Mesh Sidecars. *Proceedings of the 2023 ACM Symposium on Cloud Computing* (pp. 142–157). Association for Computing Machinery (ACM). Retrieved from https://doi.org/10.1145/3620678.3624652