Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Κατανεμημένα Συστήματα, Ασφάλεια και Αναδυόμενες Τεχνολογίες Πληροφορίας»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Ασφάλεια και αξιοπιστία σε υποδομή μέσω κώδικα: Μελέτη περίπτωσης δηλωτικής προσέγγισης** **Security and Reliability in IaC:** **A declarative approach case study**
Ονοματεπώνυμο Φοιτητή	**Κόκκαλης Νικόλαος**
Πατρώνυμο	**Κωνσταντίνος**
Αριθμός Μητρώου	**ΜΠΚΣΑ20001**
Επιβλέπων	**Παναγιώτης Κοτζανικολάου,** Καθηγητής

Ημερομηνία Παράδοσης **Νοέμβριος 2024**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Χρήστος Δουληγέρης Καθηγητής	Παναγιώτης Κοτζανικολάου Καθηγητής	Μιχαήλ Ψαράκης Αναπληρωτής Καθηγητής

ΠΕΡΙΛΗΨΗ

Η ανάπτυξη λογισμικού έχει εξελιχθεί από μια τοπική και μοναχική διαδικασία σε μια συνεργατική και πολυδιάστατη προσέγγιση, η οποία αξιοποιεί πλήθος βιβλιοθηκών και τεχνολογιών. Η δημιουργία νέων εκδόσεων λογισμικού, από μια σπάνια και χρονοβόρα διαδικασία, έχει μετατραπεί σε μια ρουτίνα που μπορεί να επαναλαμβάνεται πολλές φορές μέσα σε μία μόνο ημέρα. Επιπλέον, οι υπηρεσίες βασίζονται σε περίπλοκες υποδομές και προσαρμόζονται αυτόματα στις μεταβαλλόμενες ανάγκες των χρηστών. Αυτή η πολυπλοκότητα δημιουργεί προκλήσεις όχι μόνο για την ασφάλεια, αλλά και για άλλα μη λειτουργικά χαρακτηριστικά, όπως η αξιοπιστία και η απόδοση. Ο κύκλος ανάπτυξης λογισμικού, με την υιοθέτηση των μεθοδολογιών DevOps, περιλαμβάνει τόσο τον σχεδιασμό όσο και τη λειτουργία των υποδομών και των υπηρεσιών, με κύριο στόχο την ενίσχυση των μη λειτουργικών χαρακτηριστικών. Η παρούσα μελέτη παρέχει μία εις βάθος ανάλυση των μεθοδολογιών DevOps και DevSecOps, καθώς και των σχετικών τεχνολογιών και πρακτικών, όπως της Συνεχής Ολοκλήρωσης και Παράδοσης (CI/CD), το Infrastructure as Code (IaC) και το GitOps. Αυτές οι τεχνικές συμβάλλουν στη βελτίωση των μη λειτουργικών απαιτήσεων, όπως η ασφάλεια, η αξιοπιστία και η αναπαραγωγιμότητα των συστημάτων. Μέσω αυτών των μεθοδολογιών, προωθείται η αυτοματοποίηση, η βελτιστοποίηση των διαδικασιών και η δυνατότητα ταχύτερης και ασφαλέστερης ανάπτυξης λογισμικού.

ABSTRACT

Software development has evolved from a locally executed process to a collaborative and multidimensional approach that leverages a multitude of libraries and technologies. Creating new versions of software has gone from an infrequent and time-consuming process to a routine that can be repeated multiple times in a single day. In addition, the services are based on complex infrastructures and automatically adapt to the changing needs of users. This complexity creates challenges not only for security, but also for other non-functional characteristics such as reliability and performance. The software development cycle, with the adoption of DevOps methodologies, includes both the design and operation of infrastructures and services, with the main objective of enhancing non-functional features. This study provides an in-depth analysis of DevOps and DevSecOps methodologies, as well as related technologies and practices, such as Continuous Integration and Delivery (CI/CD), Infrastructure as Code (IaC), and GitOps. These techniques help improve non-functional requirements such as safety, reliability and reproducibility of systems. Through these methodologies, automation, process optimization and the possibility of faster and safer software development are promoted.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή μου, κ. Παναγιώτη Κοτζανικολάου, για την απεριόριστη κατανόησή του.

Επίσης, ευχαριστώ τους φίλους μου, Γιώργο, Γιώργο και Γιώργο, που με στήριξαν το προηγούμενο έτος στις μεγάλες αλλαγές που έκανα στη ζωή μου.

Τέλος, θα ήθελα να ευχαριστήσω τον προπονητή μου, Ζήση, που μου έμαθε στα τριανταπέντε μου να δουλεύω σκληρά και να ανταμείβομαι από τα αποτελέσματα.

Περιεχόμενα

Πίνακας εικόνων

1. Εισαγωγή – Περιγραφή Αντικειμένου

Στη σύγχρονη εποχή, η ραγδαία εξέλιξη της τεχνολογίας επηρεάζει πολλούς τομείς, ενώ ένα από τα πιο κρίσιμα ζητήματα στον τεχνολογικό χώρο παραμένει η ανάπτυξη, η διαχείριση και η ασφάλεια του λογισμικού. Οι προσεγγίσεις **DevOps** [17] [56], **DevSecOps** [57] και **GitOps** έχουν γίνει αναπόσπαστο κομμάτι της ανάπτυξης και της λειτουργίας των σύγχρονων συστημάτων. Εργαλεία όπως το **Jenkins** [30] για DevOps, το **OWASP ZAP** [55] για DevSecOps και το **ArgoCD** [1] για GitOps διευκολύνουν την υλοποίηση αυτών των μεθοδολογιών, ενισχύοντας την αυτοματοποίηση και την αξιοπιστία.

Πλεονεκτήματα αυτών των προσεγγίσεων περιλαμβάνουν τη βελτιστοποίηση της διαδικασίας ανάπτυξης, τη μείωση των σφαλμάτων, την ταχύτερη κυκλοφορία ενημερώσεων τη βελτίωση της ασφάλειας και τη μείωση του χρόνου διάγνωσης προβλημάτων

Ωστόσο, υπάρχουν και προκλήσεις, όπως η ανάγκη για εκπαίδευση προσωπικού, το κόστος εφαρμογής και οι δυσκολίες που σχετίζονται με την αλλαγή της εταιρικής κουλτούρας. Αυτά τα μειονεκτήματα μπορούν να αντιμετωπιστούν με επένδυση στην εκπαίδευση, σταδιακή ενσωμάτωση νέων πρακτικών και ευαισθητοποίηση του προσωπικού σχετικά με τα οφέλη των αλλαγών αυτών.

Η επένδυση σε αυτές τις προσεγγίσεις ενισχύει την αποτελεσματικότητα και τη διασφάλιση της ποιότητας στη σύγχρονη ανάπτυξη λογισμικού.

Σύγχρονες Μέθοδοι Κύκλου Ζωής Λογισμικού

Ο σύγχρονος κύκλος ζωής λογισμικού έχει εξελιχθεί ώστε να δίνει έμφαση στη **συνεχή ανάπτυξη και παράδοση**, με στόχο την αυτοματοποίηση, την ενισχυμένη συνεργασία και την άμεση προσαρμογή σε αλλαγές. Αυτές οι μέθοδοι συμβάλλουν στην **αύξηση της αποδοτικότητας** και τη **μείωση του χρόνου κυκλοφορίας** νέων εκδόσεων. Επιπλέον, η ενσωμάτωση της ασφάλειας και της παρακολούθησης σε κάθε στάδιο της ανάπτυξης διασφαλίζει την **αξιοπιστία** και την **ασφάλεια των εφαρμογών** κατά τη λειτουργία τους, ενισχύοντας τη συνολική σταθερότητα και ανθεκτικότητα των συστημάτων.

DevOps για Ανάπτυξη και Λειτουργία Συστημάτων

Η μεθοδολογία **DevOps** ενισχύει τη συνεργασία ανάμεσα στις ομάδες ανάπτυξης και λειτουργίας, προωθώντας την ταχύτερη παράδοση, την ανθεκτικότητα και την ποιοτική βελτίωση των προϊόντων. Εργαλεία όπως το **Jenkins** και το **GitLab CI** χρησιμοποιούνται ευρέως για την αυτοματοποίηση διαδικασιών ανάπτυξης και δοκιμής, μειώνοντας τον κίνδυνο σφαλμάτων. Μέσω της αυτοματοποίησης, αυτά τα εργαλεία συμβάλλουν σημαντικά στη βελτίωση της συνολικής ποιότητας του λογισμικού και στην αποτελεσματική διαχείριση των εργασιών, διευκολύνοντας παράλληλα την παρακολούθηση και τη συντήρηση των συστημάτων.

Η Μεθοδολογία DevSecOps

Το **DevSecOps** ενσωματώνει την ασφάλεια σε κάθε στάδιο του κύκλου ανάπτυξης λογισμικού (SDLC), εξασφαλίζοντας την προστασία και τη συνεχή παρακολούθηση των εφαρμογών. Εργαλεία όπως το **OWASP ZAP** για δυναμικό έλεγχο ασφαλείας και το **SonarQube** για στατικό έλεγχο κώδικα χρησιμοποιούνται ευρέως σε αυτή την προσέγγιση. Με το DevSecOps, η ασφάλεια γίνεται αναπόσπαστο μέρος της διαδικασίας ανάπτυξης, μειώνοντας τις ευπάθειες και ενισχύοντας την εμπιστοσύνη των χρηστών στο λογισμικό, καθώς οι εφαρμογές παρακολουθούνται και προστατεύονται συνεχώς από πιθανούς κινδύνους.

Δημιουργία Υποδομών με Infrastructure as Code (IaC)

Το **Infrastructure as Code (IaC)** επιτρέπει την αυτοματοποίηση, τον προγραμματισμό και την επαναληψιμότητα στη διαχείριση των υποδομών. Εργαλεία όπως το **Terraform** της HashiCorp και το **Ansible** διευκολύνουν τη διαχείριση των υποδομών μέσω κώδικα, επιτρέποντας γρήγορη δημιουργία και προτυποποίηση των υποδομών. Η χρήση του IaC συμβάλλει επίσης στη μείωση του κινδύνου ανθρώπινων σφαλμάτων και στην αύξηση της αποδοτικότητας, προσφέροντας συνεπή και ασφαλή διαχείριση των πόρων σε όλο τον κύκλο ζωής τους.

Χρήση Containers και Container Orchestration

Τα **containers** παρέχουν απομονωμένα περιβάλλοντα για τις εφαρμογές, ενώ εργαλεία διαχείρισης όπως το **Kubernetes** ενισχύουν την αυτόματη κλιμακωσιμότητα και ανθεκτικότητα των συστημάτων. Η χρήση containers, όπως το **Docker**, σε συνδυασμό με το Kubernetes για την οργάνωσή τους, εξασφαλίζει υψηλή αξιοπιστία και απλοποιεί τη διαχείριση των υπηρεσιών. Η δυνατότητα για γρήγορη κλιμάκωση επιτρέπει την αποτελεσματική ανταπόκριση σε αυξημένες απαιτήσεις, διασφαλίζοντας την απρόσκοπτη λειτουργία των εφαρμογών ακόμα και υπό υψηλό φόρτο.

Αμετάβλητη υποδομή (Immutable Infrastructure)

Ο όρος **immutable infrastructure** αναφέρεται στην πρακτική όπου οι υποδομές δημιουργούνται και αντικαθίστανται, αντί να τροποποιούνται μετά την αρχική τους εγκατάσταση. Αυτή η προσέγγιση προσφέρει συνέπεια και αξιοπιστία, καθώς μειώνει την πιθανότητα σφαλμάτων που προκύπτουν από αλλαγές σε ήδη λειτουργικές υποδομές. Επιπλέον, διευκολύνει την ταχύτερη επαναφορά σε περίπτωση αποτυχίας, βοηθώντας στη διατήρηση μιας σταθερής και ασφαλούς κατάστασης της υποδομής.

Χρήση CI/CD για Κώδικα, Υποδομή και Πολιτικές

Τα **CI/CD pipelines** (Continuous Integration/Continuous Deployment) εξασφαλίζουν συνεχή έλεγχο, αυτοματοποίηση και γρήγορη παράδοση για όλα τα στοιχεία του λογισμικού. Εργαλεία όπως το **Jenkins** και το **GitLab CI/CD** αυτοματοποιούν τις διαδικασίες, μειώνοντας τον χρόνο για την κυκλοφορία νέων εκδόσεων και βελτιώνοντας την ποιότητα του λογισμικού. Η συνεχής παράδοση διασφαλίζει ότι οι ενημερώσεις και οι διορθώσεις γίνονται με συνέπεια και χωρίς διακοπές.

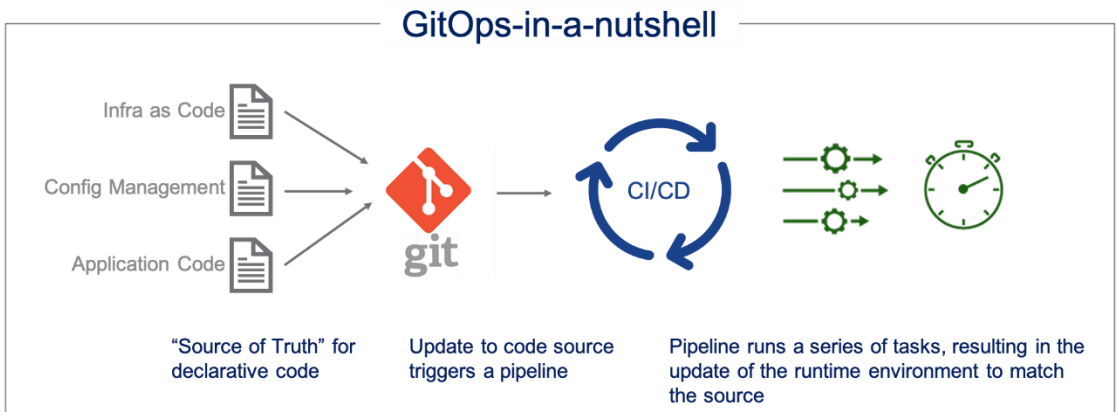
Η υιοθέτηση της μεθοδολογίας CI/CD όχι μόνο για τον κώδικα του λογισμικού αλλά και για τον κώδικα που διαχειρίζεται υποδομές, υπηρεσίες τρίτων, δίκτυα και πολιτικές, ενισχύει τις **μη λειτουργικές απαιτήσεις** και μειώνει τα ανθρώπινα σφάλματα. Αυτός ο τρόπος προσέγγισης συμβάλλει στη συνοχή, στη σταθερότητα και στην ασφάλεια του συνολικού οικοσυστήματος.

Χρήση Observability στη Λειτουργία Υπηρεσιών

Η ύπαρξη και χρήση μιας υποδομής **observability** επιτρέπει τη σε βάθος κατανόηση και παρακολούθηση των συστημάτων κατά την εκτέλεση, διευκολύνοντας την αναγνώριση προβλημάτων και την αύξηση της απόδοσης. Εργαλεία όπως το **Prometheus** και το **Grafana** συμβάλλουν σε αυτό, παρέχοντας μετρήσεις και γραφήματα που διευκολύνουν την παρακολούθηση και ανάλυση των συστημάτων. Η παρακολούθηση των **logs**, των **μετρικών** και των **traces** βοηθά στον εντοπισμό σφαλμάτων και τη βελτίωση της σταθερότητας. Αυτή η προσέγγιση επιτρέπει στις ομάδες να αντιδρούν γρήγορα σε προβλήματα και να εξασφαλίζουν τη συνεχή και αξιόπιστη λειτουργία των υπηρεσιών.

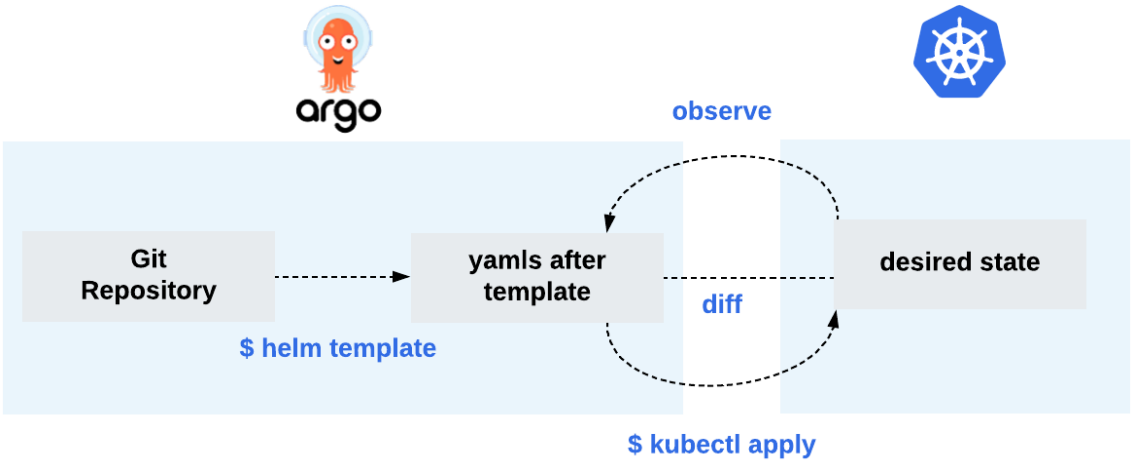
Μεθοδολογία GitOps

Η μεθοδολογία **GitOps** χρησιμοποιεί το Git ως πηγή αλήθειας για την υποδομή και τις εφαρμογές, ενισχύοντας την αυτοματοποίηση, τη διαφάνεια και την αναδρομικότητα. Εργαλεία όπως το **ArgoCD** και το **Flux** υποστηρίζουν αυτή την προσέγγιση, όπου οι αλλαγές πραγματοποιούνται μέσω commits στο Git, επιτρέποντας την εύκολη παρακολούθηση και αναίρεση αλλαγών, ενώ διασφαλίζουν τη συγχρονισμένη λειτουργία υποδομών και εφαρμογών. Η προσέγγιση αυτή μειώνει τον χρόνο ανάπτυξης, εξασφαλίζοντας **επαναληψιμότητα** και **αξιοπιστία** των αλλαγών. Στην Εικόνα 1 βλέπουμε το μοντέλο GitOps στην πράξη.



Εικόνα 1 Μοντέλο GitOps - Πηγή: https://blogs.vmware.com/cloud/2021/02/24/gitops-cloud-operating-model/

Η υιοθέτηση αυτών των σύγχρονων προσεγγίσεων στην ανάπτυξη, διαχείριση και ασφάλεια λογισμικού προσφέρει πλεονεκτήματα όπως αυξημένη **αξιοπιστία**, δυνατότητα γρήγορης **κλιμάκωσης** και ενισχυμένη **προστασία** των συστημάτων. Παρά τις προκλήσεις που μπορεί να προκύψουν κατά την υιοθέτησή τους, η επένδυση σε αυτές τις πρακτικές αποτελεί ένα σημαντικό βήμα για τη βελτίωση της αποτελεσματικότητας και της ασφάλειας των πληροφοριακών υποδομών.



Εικόνα 2 reconciliation loop - Πηγή: https://subscription.packtpub.com/book/cloud-and-networking/9781803233321/3/ch03lvl1sec15/explaining-architecture

Η προσέγγιση του **Infrastructure as Code (IaC)**, του **containerization**, του **CI/CD**, και του **GitOps** συνδέεται στενά με το **δηλωσιακό μοντέλο προγραμματισμού**, αξιοποιώντας αυτοματισμούς και επαναληψιμότητα στη διαχείριση υποδομών και εφαρμογών. Στο δηλωσιακό μοντέλο, οι χρήστες ορίζουν την επιθυμητή κατάσταση του συστήματος, και ο μηχανισμός αναλαμβάνει τις απαραίτητες ενέργειες. Το IaC επιτρέπει στους μηχανικούς να περιγράφουν τις υποδομές τους με αρχεία κώδικα (π.χ., YAML, JSON), τα οποία μπορούν να ερμηνευτούν για δημιουργία και διαχείριση υποδομών, ενώ το **containerization** (π.χ., **Docker**) διασφαλίζει ότι οι εφαρμογές λειτουργούν με συνέπεια σε κάθε περιβάλλον. Οι μέθοδοι **CI/CD** και **GitOps**

ενισχύουν τη συνεχή ενσωμάτωση και παράδοση, επιτρέποντας την αναπαραγωγή, τους ελέγχους και τις αυτοματοποιημένες αναβαθμίσεις.

Αυτές οι τεχνικές αποτελούν τον πυρήνα του **DevOps** (ή **DevSecOps** όταν εμπλέκεται η ασφάλεια), ενοποιώντας ανάπτυξη, ασφάλεια και λειτουργία υπηρεσιών με αυτοματοποίηση, βελτιώνοντας τη συνεργασία και μειώνοντας τα λάθη που προκύπτουν από χειροκίνητες διαδικασίες. Ενισχύουν τα **non-functional requirements** (NFRs), όπως **αξιοπιστία**, **κλιμάκωση**, και **συντηρησιμότητα**. Η αυτοματοποίηση με IaC μειώνει τον χρόνο αποκατάστασης σε περιπτώσεις προβλημάτων, ενώ η χρήση Docker containers και GitOps διασφαλίζει ότι το σύστημα είναι επαναλήψιμο και σταθερό ανεξάρτητα από το περιβάλλον. Παράλληλα, η συνεχής ενσωμάτωση και παράδοση (CI/CD) προσφέρει ταχύτητα και συνέπεια, διασφαλίζοντας ελέγχους ποιότητας.

Το IaC, Docker, CI/CD και GitOps σχετίζονται επίσης με την **immutable infrastructure**, το **configuration drift**, και το **reconciliation loop** – στοιχεία που στηρίζουν τη σύγχρονη υποδομή DevOps. Στην **immutable infrastructure**, αντί να τροποποιείται η υπάρχουσα κατάσταση, δημιουργούνται νέα αμετάβλητα στοιχεία που αντικαθιστούν τα παλιά, μειώνοντας τα απρόβλεπτα σφάλματα. Το Docker συνεισφέρει, παρέχοντας αμετάβλητα containers με συνεπή κατάσταση ανεξαρτήτως περιβάλλοντος. Με το IaC, είναι εύκολο να καταργηθούν παλιές εκδόσεις και να εφαρμοστούν νέες, αποφεύγοντας το **configuration drift** (την απόκλιση της πραγματικής από την επιθυμητή κατάσταση λόγω χειροκίνητων διαδικασιών).

Το **GitOps** και το **CI/CD** ενισχύουν αυτές τις έννοιες μέσω του **reconciliation loop**, εξασφαλίζοντας ότι η τρέχουσα κατάσταση του συστήματος συμφωνεί με την επιθυμητή, όπως αυτή έχει δηλωθεί σε κώδικα (π.χ., Git). Το GitOps χρησιμοποιεί το Git ως μοναδική πηγή αλήθειας και μέσω αυτοματοποιημένων ελέγχων, διασφαλίζεται ότι κάθε απόκλιση διορθώνεται, επαναφέροντας το σύστημα στην επιθυμητή κατάσταση. Αυτή η σύνθεση τεχνικών δημιουργεί μια πλήρως αυτοματοποιημένη, ανθεκτική και εύκολα κλιμακούμενη υποδομή, ενισχύοντας τα **NFRs**, όπως σταθερότητα, ασφάλεια και απόδοση. Στην Εικόνα 2 βλέπουμε πως λειτουργεί το reconciliation loop στην μεθοδολογία GitOps.

1.1. Σκοπός, Στόχοι Και Συνεισφορά Της Διατριβής

Η παρούσα διατριβή επικεντρώνεται στη συντονισμένη χρήση των μεθόδων **DevSecOps** κατά τον κύκλο ανάπτυξης και λειτουργίας μιας υπηρεσίας, αξιοποιώντας δηλωτικές μεθόδους όπως το **Infrastructure as Code (IaC)** για τον καθορισμό και την υλοποίηση της υποδομής, της παρατηρησιμότητας (observability), καθώς και της πολιτικής της υπηρεσίας. Επιπλέον, η διατριβή εξετάζει τη χρήση του **Git** ως αποθηκευτικού χώρου και μοναδικής πηγής αλήθειας, μαζί με τη μεθοδολογία **GitOps** για την ενίσχυση της λειτουργίας και την εφαρμογή των **CI/CD** διαδικασιών στον κύκλο ανάπτυξης, ώστε να εξασφαλίζονται η συνολική ασφάλεια, η αξιοπιστία, και άλλα μη λειτουργικά χαρακτηριστικά της υπηρεσίας.

Αυτές οι προσεγγίσεις στοχεύουν στη βελτίωση των μη λειτουργικών απαιτήσεων, όπως:

- **Auditability** (ελέγξιμότητα)
- **Availability** (διαθεσιμότητα)
- **Compliance** (συμμόρφωση)
- **Documentation** (τεκμηρίωση)
- **Fault tolerance** (ανοχή σφαλμάτων)
- **Maintainability** (συντηρησιμότητα)
- **Reliability** (αξιοπιστία)
- **Resilience** (ανθεκτικότητα)
- **Scalability** (κλιμακωσιμότητα)
- **Security** (ασφάλεια)
- **Testability** (δοκιμασιμότητα)

Η διατριβή αναλύει τις παραπάνω μεθοδολογίες και τεχνικές, παρουσιάζει τα εργαλεία που χρησιμοποιήθηκαν, όπως το **Gitlab CI** για CI/CD και το **Terraform** για IaC, και περιλαμβάνει συζητήσεις με τους προγραμματιστές για τις προκλήσεις και τις λύσεις που υλοποιήθηκαν. Οι υποδομές υλοποιούνται ως κώδικας και αποθηκεύονται σε **git repositories**, ενώ τα **CI/CD pipelines** επιτελούν ελέγχους και δημιουργούν τα δομικά στοιχεία της υπηρεσίας, τις υποδομές για μετρικές, και εφαρμόζουν τις πολιτικές ασφάλειας.

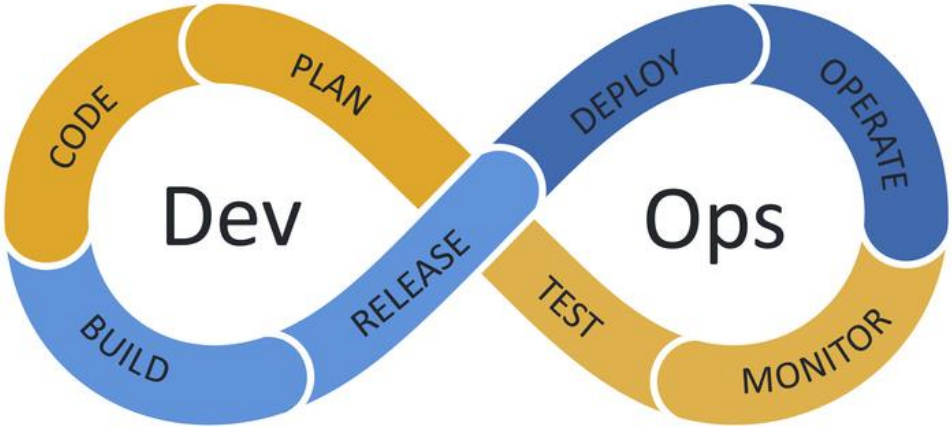
Η επιλογή πλατφόρμας cloud επικεντρώνεται πιθανότατα στο **Azure**, λόγω της σύνδεσης με τον ακαδημαϊκό τομέα και της ευελιξίας του στα εργαλεία που διευκολύνουν την υλοποίηση των DevSecOps προσεγγίσεων. Εργαλεία όπως το **DAST (Dynamic Application Security Testing)** με **OWASP ZAP** για ανίχνευση ευπαθειών και το **Caliper** για benchmarking αξιοποιούνται για την αξιολόγηση της ασφάλειας και της απόδοσης της υποδομής.

1.1.1. Καθορισμός των Στόχων της Εργασίας

Η παρουσίαση των μεθόδων ανάπτυξης υποδομών με τις μεθοδολογίες:

- **DevOps**
- **DevSecOps**
- **GitOps**
- **Observability**
- **IaC (Infrastructure as Code)**

στοχεύει στη δημιουργία μιας υπηρεσίας που μπορεί να επιβεβαιωθεί από τρίτους και να πληροί υψηλά επίπεδα μη λειτουργικών απαιτήσεων. Αυτές οι προσεγγίσεις ενισχύουν την αξιοπιστία, την ασφάλεια και την ανθεκτικότητα της υπηρεσίας, διασφαλίζοντας παράλληλα τη διαθεσιμότητα, τη συντηρησιμότητα και την επεκτασιμότητά της.



Εικόνα 3 κύκλος ζωής ανάπτυξης λογισμικού DevOps

Για την ανάπτυξη και λειτουργία υπηρεσιών, υιοθετούμε διάφορες μεθοδολογίες και πρακτικές που αποσκοπούν στη βελτίωση της **ποιότητας**, της **ασφάλειας** και της **αποδοτικότητας** των συστημάτων. Κάθε ένας από τους βασικούς μας στόχους αναλύεται λεπτομερώς ως εξής:

DevOps

Περιγραφή: Συνδυάζουμε την ανάπτυξη (Dev) και τη λειτουργία (Ops) για να βελτιώσουμε την ταχύτητα, την ποιότητα και την ανταπόκριση των αναπτυσσόμενων λύσεων.

Στόχος: Παροχή υποδομής συνεχούς παράδοσης, αυτοματοποίηση των διαδικασιών και συγχώνευση και επικοινωνία μεταξύ των ομάδων ανάπτυξης και λειτουργίας.

Στην Εικόνα 3 βλέπουμε ποια στοιχεία απαρτίζουν έναν κύκλο ζωής ανάπτυξης λογισμικού DevOps.

DevSecOps

Περιγραφή: Προσθέτουμε την ασφάλεια (Sec) στον κύκλο ζωής του DevOps, εστιάζοντας στην αυτοματοποίηση των ελέγχων ασφαλείας.

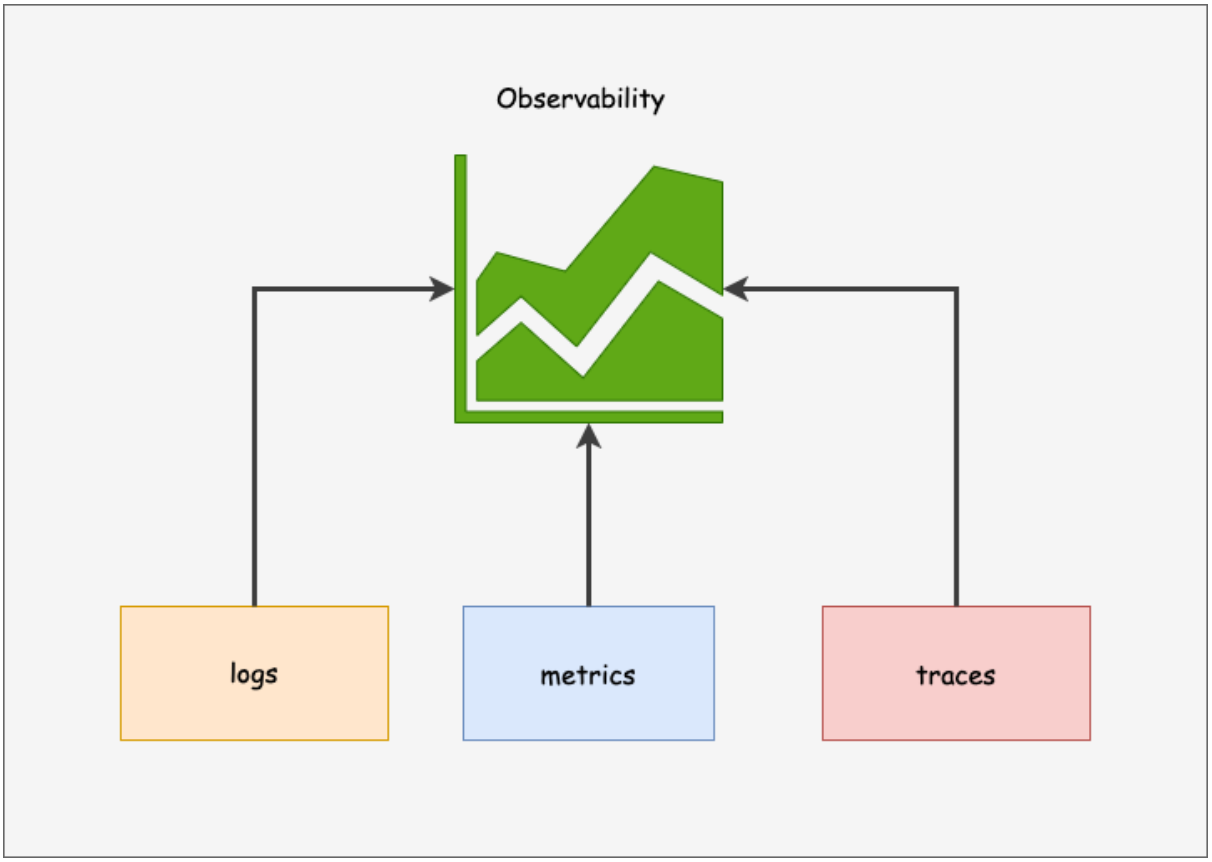
Στόχος: Διασφαλίζουμε ότι η ασφάλεια είναι ενσωματωμένη στην ανάπτυξη και λειτουργία και όχι μια μεταγενέστερη προσθήκη.

GitOps

Περιγραφή: Χρησιμοποιούμε το Git ως ενιαία πηγή αλήθειας για τη διαχείριση και αυτοματοποίηση των υποδομών.

Στόχος: Επίτευξη αυτοματοποιημένων αλλαγών στην υποδομή μέσω των commits στο Git, διασφαλίζοντας παράλληλα την αναφορά και την ευκολία επαναφοράς.

Observability



Εικόνα 4 Τρεις πυλώνες observability - Πηγή: https://devopscube.com/what-is-observability/

Περιγραφή: Παρακολουθούμε και κατανοούμε την κατάσταση του συστήματος από τα εξωτερικά δεδομένα που παράγει, όπως μετρικές, logs και traces.

Στόχος: Διασφαλίζουμε την ικανότητα των ομάδων να εντοπίζουν και να αντιμετωπίζουν τα προβλήματα γρήγορα, μειώνοντας τον χρόνο επίλυσης προβλημάτων.

Στην Εικόνα 4 βλέπουμε ποια στοιχεία απαρτίζουν το σύνολο του observability.

IaC (Infrastructure as Code)

Περιγραφή: Διαχειριζόμαστε και παραμετροποιούμε τις υποδομές μέσω κώδικα και αρχείων διαμόρφωσης.

Στόχος: Αυτοματοποίηση της δημιουργίας, τροποποίησης και διαχείρισης των υποδομών, εξασφαλίζοντας παράλληλα τη συνέπεια και την επαναληψιμότητα.

Για να επιτευχθεί μια υπηρεσία που μπορεί να επιβεβαιωθεί από τρίτους, οι παραπάνω πρακτικές πρέπει να εφαρμοστούν συντονισμένα, με έμφαση στην **τεκμηρίωση**, στις **αναθεωρήσεις** (όπως αναθεώρηση κώδικα και αρχιτεκτονικής) και στην **αυτοματοποίηση ελέγχων**. Η τεκμηρίωση είναι απαραίτητη για την κατανόηση της υπηρεσίας από εξωτερικούς ελεγκτές, ενώ οι αναθεωρήσεις διασφαλίζουν ότι οι διαδικασίες είναι ακριβείς και συμμορφωμένες με τα πρότυπα.

Οι μη λειτουργικές απαιτήσεις, όπως **απόδοση**, **αντοχή**, **ασφάλεια** και **κλιμακωσιμότητα**, πρέπει να λαμβάνονται υπόψη σε κάθε βήμα της διαδικασίας. Η συνεπής ενσωμάτωση αυτών των παραγόντων συμβάλλει στη δημιουργία ενός συστήματος ανθεκτικού, ασφαλούς και αποδοτικού, διευκολύνοντας τη συντήρησή του και τη μελλοντική του επέκταση.

Οι προσεγγίσεις που αναλύονται εδώ στοχεύουν στην ενίσχυση της **συνεργασίας**, την **ενσωμάτωση της ασφάλειας από την αρχή** και την **τυποποίηση των διαδικασιών ανάπτυξης**. Μέσω αυτοματισμών και εργαλείων που διασφαλίζουν τη συνοχή και την ποιότητα, η υπηρεσία αποκτά αξιοπιστία και ανθεκτικότητα απέναντι στις σύγχρονες προκλήσεις, προσφέροντας παράλληλα δυνατότητα επιβεβαίωσης από τρίτους.

1.2. Δομή Της Διατριβής

Η δομή της παρούσας διατριβής οργανώνεται με τέτοιο τρόπο ώστε να καθοδηγεί τον αναγνώστη από τη θεωρητική ανάλυση προς την πρακτική εφαρμογή της προτεινόμενης λύσης. Αρχικά, στην "Εισαγωγή" παρουσιάζεται μια συνοπτική περιγραφή του προβλήματος και των στόχων της εργασίας. Στη συνέχεια, η ενότητα "Σχετική Βιβλιογραφία" παρέχει μια ανασκόπηση της υπάρχουσας έρευνας και των σχετικών έργων. Ακολουθεί η ενότητα "Κύκλος Ζωής για την Ανάπτυξη Υποδομής ως Κώδικα", η οποία αποτελεί τη θεωρητική βάση για την ανάπτυξη της προτεινόμενης λύσης. Έπειτα, παρουσιάζεται μια "Μελέτη Περίπτωσης" για την ανάπτυξη μιας πολυεπίπεδης αρχιτεκτονικής blockchain, συμπεριλαμβανομένης της ανάλυσης απαιτήσεων, της περιγραφής της λύσης, των παραδοχών και της σύνθεσης της τελικής λύσης. Ακολουθεί η "Ανάλυση Αποτελεσμάτων", όπου αξιολογούνται επιμέρους χαρακτηριστικά όπως η ασφάλεια και η αξιοπιστία. Τέλος, η διατριβή κλείνει με τα "Συμπεράσματα" και τις "Μελλοντικές Κατευθύνσεις", όπου συνοψίζονται τα βασικά αποτελέσματα και προτείνονται κατευθύνσεις για μελλοντική έρευνα και βελτιώσεις.

2. Σχετική Βιβλιογραφία

Αυτή η ενότητα παρέχει μια ολοκληρωμένη ανασκόπηση της σχετικής βιβλιογραφίας που υποστηρίζει τις μεθοδολογίες και τεχνολογίες που εξετάζονται στη διατριβή. Περιλαμβάνει τόσο τη γκρίζα βιβλιογραφία, όπως τις οδηγίες και τα πρότυπα από φορείς όπως το **NIST** και το **OWASP**, όσο και την επιστημονική βιβλιογραφία που εξετάζει θέματα ασφαλείας, αξιοπιστίας και ανάπτυξης **Infrastructure as Code (IaC)**. Στόχος της ενότητας αυτής είναι να εξοπλίσει τον αναγνώστη με τη θεωρητική βάση των προτεινόμενων μεθοδολογιών, διευκολύνοντας μια βαθύτερη κατανόηση των σχετικών τεχνολογιών και πρακτικών.

Κύριες Αναφορές και Πηγές:

- **NIST SP 800-204C**: Το **National Institute of Standards and Technology (NIST)**, μέσω του SP 800-204C (2023), παρέχει κατευθυντήριες γραμμές για την εφαρμογή αρχών ασφαλείας σε αρχιτεκτονικές μικροϋπηρεσιών.
- **Implementation of DevSecOps for a Microservices-based Application with Service Mesh**: Αναλύει την εφαρμογή DevSecOps αρχών σε μικροϋπηρεσίες, εστιάζοντας στις τεχνολογίες **service mesh** για την ενίσχυση της ασφάλειας.
- **DoD Enterprise DevSecOps Reference Design: CNCF Kubernetes**: Το **Υπουργείο Άμυνας των ΗΠΑ** (DoD) δημιούργησε αυτό το πλαίσιο για την εφαρμογή του DevSecOps σε περιβάλλον **Kubernetes**, με βάση τις συστάσεις του **Cloud Native Computing Foundation (CNCF)**.
- **OWASP DevSecOps Guideline**: Ο **OWASP** παρέχει έναν οδηγό που εστιάζει στην ενσωμάτωση της ασφάλειας σε κάθε στάδιο του κύκλου ανάπτυξης λογισμικού.
- **DevSecOps Enterprise Container Hardening Guide 1.1**: Παρέχει οδηγίες για την ασφάλεια των containers σε DevSecOps περιβάλλον (Department of Defense, 2021).
- **Kubernetes Security Technical Implementation Guide (STIG)** και **CTR Kubernetes Hardening Guidance 1.1**: Προσφέρουν οδηγίες και πρακτικές για την ενίσχυση της ασφάλειας σε περιβάλλον Kubernetes.
- **Secure Software Development Framework (SSDF)**: Το SSDF (2021) παρέχει ένα πλαίσιο καλών πρακτικών για την ασφαλή ανάπτυξη λογισμικού, διασφαλίζοντας την ενσωμάτωση των αρχών ασφαλείας από την αρχή έως το τέλος του κύκλου ζωής ανάπτυξης.

Στόχος και Οφέλη του SSDF:

Ο **κύριος στόχος** του SSDF είναι να καθοδηγήσει τις οργανώσεις στην ασφαλή ανάπτυξη λογισμικού, ενσωματώνοντας την ασφάλεια σε όλη τη διαδικασία ανάπτυξης. Το SSDF περιλαμβάνει τα εξής συστατικά:

1. **Προετοιμασία**: Καθορισμός ομάδας ασφαλείας και εκπαίδευση.
2. **Σχεδίαση**: Καθορισμός των απαιτήσεων ασφάλειας και σχεδίαση χαρακτηριστικών ασφαλείας.
3. **Υλοποίηση**: Κωδικοποίηση και αναθεώρηση του κώδικα με προτεραιότητα στην ασφάλεια.
4. **Έλεγχος**: Αξιολόγηση του λογισμικού για πιθανές ευπάθειες.
5. **Διατήρηση**: Διαχείριση ευπαθειών και ενημερώσεων για τη συνεχή ασφάλεια.

Η εφαρμογή του SSDF αυξάνει την ασφάλεια του λογισμικού, μειώνοντας τις πιθανότητες ευπαθειών. Ωστόσο, απαιτείται αλλαγή στην κουλτούρα της οργάνωσης, επενδύσεις σε εκπαίδευση και συνεχή παρακολούθηση.

Πρόσθετα Κανονιστικά Πλαίσια και Στάνταρτ

Περιλαμβάνεται μια λίστα με **πρότυπα** και **κανονιστικά πλαίσια** που καλύπτουν DevOps, DevSecOps, ασφάλεια εφοδιαστικής αλυσίδας, immutable infrastructure και IaC, παρέχοντας σύντομες περιγραφές για τη σημασία και τον σκοπό τους.

2.1. Σχετικά Πρότυπα Και Κανονιστικά Πλαίσια

DevOps & DevSecOps

NIST SP 800-204C: Οδηγίες για την εφαρμογή αρχών ασφάλειας σε αρχιτεκτονικές μικροϋπηρεσιών, με έμφαση στις ασφαλείς DevSecOps διαδικασίες για περιβάλλοντα που βασίζονται σε service mesh. [39]

DoD Enterprise DevSecOps Reference Design: CNCF Kubernetes: Ένα πλαίσιο που αναπτύχθηκε από το Υπουργείο Άμυνας των ΗΠΑ για την εφαρμογή των DevSecOps πρακτικών σε Kubernetes περιβάλλοντα. [11]

OWASP DevSecOps Guideline: Κατευθυντήριες γραμμές από το OWASP για την εφαρμογή DevSecOps, με επίκεντρο την ασφάλεια σε κάθε φάση του κύκλου ζωής της ανάπτυξης λογισμικού. [44]

DevSecOps Enterprise Container Hardening Guide 1.1: Οδηγίες για την ενίσχυση της ασφάλειας των containers. [10]

Kubernetes Security Technical Implementation Guide (STIG): Οδηγίες και βέλτιστες πρακτικές για την ενίσχυση της ασφάλειας του Kubernetes.

CTR Kubernetes Hardening Guidance 1.1: Οδηγίες για την ενίσχυση του Kubernetes και τη δημιουργία ασφαλούς περιβάλλοντος. (Center for Threat Reduction, 2021) [6]

2.2. Supply Chain Security

Supply Chain Levels for Software Artifacts (SLSA): Κανονιστικό πλαίσιο για την ασφάλεια της εφοδιαστικής αλυσίδας λογισμικού.

ISO/IEC 27036: Βέλτιστες πρακτικές για την ασφάλεια στις σχέσεις προμηθευτών.

NIST Cybersecurity Supply Chain Risk Management (C-SCRM): Πλαίσιο για την ανάλυση και τον έλεγχο των κινδύνων ασφάλειας στην εφοδιαστική αλυσίδα.

2.3. Infrastructure As Code (Iac) & Immutable Infrastructure

HashiCorp Terraform Best Practices: Οδηγός για την ασφαλή και αποδοτική ανάπτυξη IaC με Terraform.

NIST SP 800-53: Μέτρα ελέγχου ασφάλειας και ιδιωτικότητας για πληροφοριακά συστήματα, με εφαρμογή σε IaC διαδικασίες

NIST SSDF Version 1.1: Ένα πλαίσιο για την ασφαλή ανάπτυξη λογισμικού, περιλαμβάνοντας τις πρακτικές προετοιμασίας, σχεδίασης, υλοποίησης, ελέγχου και διατήρησης ασφαλούς λογισμικού. [38]

2.4. Συμπληρωματικες Πηγες και Οδηγοι Ασφαλειας

Building Secure and Reliable Systems: Ένα βιβλίο που καλύπτει τις βέλτιστες πρακτικές για τον σχεδιασμό, την υλοποίηση, και τη διατήρηση ασφαλών και αξιόπιστων συστημάτων.

Τα βιβλία της Google για Site Reliability Engineering (SRE) αποτελούν μια ολοκληρωμένη πηγή γνώσης για την υλοποίηση των αρχών SRE σε μεγάλες κλίμακες. Με κύριους τίτλους όπως το "Site Reliability Engineering: How Google Runs Production Systems" [5] και το "The Site Reliability Workbook", προσφέρουν μια βαθιά κατανόηση της στρατηγικής που ακολουθεί η

Google για τη διατήρηση της σταθερότητας και της απόδοσης των συστημάτων της. Εξετάζουν τη σημασία της αυτοματοποίησης, του monitoring, της διαχείρισης κινδύνου και των SLOs (Service Level Objectives), δίνοντας παραδείγματα και πρακτικά εργαλεία για τη μείωση των συστημικών λαθών και την αύξηση της ανθεκτικότητας των υπηρεσιών. Αυτά τα βιβλία παρέχουν στους επαγγελματίες έναν αξιόπιστο οδηγό για την ανάπτυξη μιας κουλτούρας ευθύνης και συνεργασίας, που στοχεύει στη βέλτιστη εξυπηρέτηση πελατών και στη συνεχή βελτίωση των υπηρεσιών.

Αυτά τα πρότυπα και κανονιστικά πλαίσια αποτελούν βασικά εργαλεία για την ανάπτυξη, διαχείριση και διατήρηση ασφαλών υποδομών και λογισμικού. Παρέχουν καθοδήγηση για την ενσωμάτωση ασφάλειας και αξιοπιστίας από την αρχή έως το τέλος της διαδικασίας ανάπτυξης.

2.5. Πρότυπα Για Τον Κύκλο Ζωής Ανάπτυξης Λογισμικού (SdIc) Και Μη-Λειτουργικές Απαιτήσεις (Nfrs)

2.5.1. Software Development Life Cycle (SDLC)

ISO/IEC/IEEE 12207:2017: Πρότυπο που καθορίζει τις διαδικασίες, δραστηριότητες και εργασίες του κύκλου ζωής του λογισμικού, με έμφαση στην ποιότητα και την αξιοπιστία.

NIST SP 800-160: Κατευθυντήριες γραμμές για την ενσωμάτωση πρακτικών ασφάλειας στο SDLC, υποστηρίζοντας την ασφαλή σχεδίαση και διατήρηση συστημάτων.

ISO/IEC 29110: Πρότυπο για μικρούς οργανισμούς που επιθυμούν να βελτιώσουν τις διαδικασίες ανάπτυξης λογισμικού.

NIST SP 800-218: Κατευθυντήριες γραμμές για την ενσωμάτωση της ασφάλειας στο SDLC.

Μη-Λειτουργικές Απαιτήσεις (NFRs)

Οι μη-λειτουργικές απαιτήσεις καθορίζουν την ποιότητα και τη βιωσιμότητα των συστημάτων.

ISO/IEC 25010: Πρότυπο που παρέχει μοντέλα ποιότητας για συστήματα και λογισμικό, εστιάζοντας σε NFRs όπως αξιοπιστία, ασφάλεια, αποδοτικότητα, συντηρησιμότητα, και χρηστικότητα.

ISO/IEC 9126: Παλιότερο πρότυπο που καθόριζε χαρακτηριστικά ποιότητας λογισμικού, τώρα ενσωματωμένο στο ISO/IEC 25010.

ISO/IEC 27001: Πρότυπο για τη διαχείριση της ασφάλειας των πληροφοριών, εξασφαλίζοντας εμπιστευτικότητα, ακεραιότητα, και διαθεσιμότητα των δεδομένων.

NIST SP 800-44: Κατευθυντήριες γραμμές για την ασφάλεια των δημόσιων web servers, με έμφαση στην ασφάλεια, αξιοπιστία, και απόδοση.

ISO/IEC 27034: Κατευθυντήριες γραμμές για την ασφάλεια των εφαρμογών, εστιάζοντας στις NFRs ασφάλειας.

2.5.2. DevOps και Μη-Λειτουργικές Απαιτήσεις (NFRs)

Οι **μη-λειτουργικές απαιτήσεις** (NFRs) είναι καίριες για την επιτυχημένη εφαρμογή των μεθοδολογιών **DevOps** και **DevSecOps**, καθώς περιλαμβάνουν σημαντικά χαρακτηριστικά όπως:

- **Απόδοση (Performance)**: Εξασφαλίζεται μέσω συνεχούς παρακολούθησης από εργαλεία όπως το **Prometheus** και το **Grafana**, διασφαλίζοντας την αποδοτική λειτουργία των συστημάτων.
- **Αξιοπιστία (Reliability)**: Οι πρακτικές **Site Reliability Engineering (SRE)** διασφαλίζουν τη σταθερή λειτουργία των υπηρεσιών και ελαχιστοποιούν τον χρόνο διακοπής.

- **Κλιμακωσιμότητα (Scalability)**: Τα DevOps περιβάλλοντα με εργαλεία όπως το **Kubernetes** επιτρέπουν την αυτόματη κλιμάκωση των υποδομών, προσαρμόζοντας τις δυνατότητες του συστήματος σε αυξημένες απαιτήσεις.
- **Ασφάλεια (Security)**: Το **DevSecOps** ενσωματώνει την ασφάλεια σε κάθε φάση του κύκλου ανάπτυξης λογισμικού (SDLC). Πρότυπα όπως το **SSDF** και κανονιστικά πλαίσια όπως ο **OWASP DevSecOps Guideline** προσφέρουν κατευθύνσεις για την ασφαλή ανάπτυξη και ενσωμάτωση των NFRs.

Η ενσωμάτωση των NFRs σε κάθε στάδιο του SDLC είναι σημαντική, με τα κανονιστικά πλαίσια και πρότυπα να παρέχουν καθοδήγηση για την επίτευξη ποιοτικών χαρακτηριστικών, όπως ασφάλεια και αξιοπιστία, απαραίτητων για τη δημιουργία αποδοτικών συστημάτων.

Αξιολόγηση της Απόδοσης με DORA και Space Metrics

Τα **DORA metrics** και **Space metrics** είναι καθιερωμένα πλαίσια για την αξιολόγηση της απόδοσης των ομάδων λογισμικού, προσφέροντας διαφορετικές αλλά συμπληρωματικές προσεγγίσεις για την ανάλυση των DevOps πρακτικών και την ανάπτυξη λογισμικού:

- Τα **DORA metrics**, από την έρευνα της DevOps Research and Assessment (DORA), επικεντρώνονται σε λειτουργικούς δείκτες, όπως η συχνότητα ανάπτυξης, ο χρόνος παράδοσης αλλαγών, το ποσοστό αποτυχίας και ο χρόνος αποκατάστασης υπηρεσιών, προσφέροντας ένα σαφές πλαίσιο για την αξιολόγηση της τεχνικής αριστείας και της ευελιξίας των ομάδων.
- Τα **Space metrics**, προτεινόμενα από την **Google**, προσφέρουν μια ολιστική προσέγγιση μέσω δεικτών ταχύτητας, παραγωγικότητας, ακρίβειας και συνεργασίας, δίνοντας έμφαση στην επίδραση των μεθοδολογιών στην κουλτούρα και τη δυναμική της ομάδας.

Ασφάλεια Ανοιχτού Κώδικα με OpenSSF

Το **OpenSSF** (Open Source Security Foundation), μια πρωτοβουλία του Ιδρύματος Linux, στοχεύει στη βελτίωση της ασφάλειας του λογισμικού ανοιχτού κώδικα, καθώς αυτό αποτελεί τη βάση για πολλές κρίσιμες εφαρμογές. Η OpenSSF συγκεντρώνει οργανισμούς και άτομα από διαφορετικά πεδία για την προώθηση εργαλείων, βέλτιστων πρακτικών και εκπαιδευτικών πόρων με στόχο την ενίσχυση της ασφάλειας του κώδικα.

Τα κύρια projects του OpenSSF περιλαμβάνουν:

1. **Sigstore**: Παρέχει υποδομή για υπογραφή και επαλήθευση των artifacts λογισμικού, ενισχύοντας την ακεραιότητα της αλυσίδας εφοδιασμού.
2. **Scorecards**: Αξιολογεί την ασφάλεια των έργων ανοιχτού κώδικα, παρέχοντας σκορ ασφαλείας και αναγνωρίζοντας πιθανούς κινδύνους.
3. **Allstar**: Αυτόματο εργαλείο διαμόρφωσης ασφάλειας που παρακολουθεί τα GitHub repositories και εφαρμόζει πολιτικές ασφαλείας.
4. **Security Reviews**: Παρέχει αξιολογήσεις ασφάλειας για έργα ανοιχτού κώδικα, εντοπίζοντας και αντιμετωπίζοντας ευπάθειες.
5. **Best Practices Badge Program**: Βοηθά τα έργα να ακολουθούν βέλτιστες πρακτικές ασφαλείας, απονέμοντας badges σε όσα πληρούν τα κριτήρια.
6. **OpenSSF Security Tooling**: Συλλογή εργαλείων ασφάλειας, όπως scanners ευπαθειών και αναλυτές εξαρτήσεων.
7. **Alpha-Omega**: Ενίσχυση ασφάλειας σε κρίσιμα έργα ανοιχτού κώδικα με χρηματοδότηση και άλλους πόρους.
8. **OpenSSF Training and Education**: Παρέχει εκπαιδευτικούς πόρους για προγραμματιστές με σκοπό την ενημέρωση και εκπαίδευση σχετικά με την ασφάλεια.

Αυτά τα projects αποτελούν βασικά στοιχεία της στρατηγικής του OpenSSF, διασφαλίζοντας την ακεραιότητα και την ασφάλεια του λογισμικού ανοιχτού κώδικα σε παγκόσμια κλίμακα.

3. Κύκλος Ζωής για την Ανάπτυξη Υποδομής ως Κώδικα

Αυτή η ενότητα εστιάζει στη διαδικασία υλοποίησης και διαχείρισης υποδομών με τη χρήση προγραμματιστικών εργαλείων, υιοθετώντας την προσέγγιση του **Infrastructure as Code (IaC)**. Αυτή η μέθοδος βασίζεται στην **αυτοματοποίηση** και την **αναπαραγωγιμότητα** των υποδομών, διασφαλίζοντας τη συνεπή και διαχειρίσιμη ανάπτυξή τους μέσω κώδικα. Η ανάλυση περιλαμβάνει τα βασικά στάδια του κύκλου ζωής της υποδομής, ξεκινώντας από τον σχεδιασμό και την ανάπτυξη, περνώντας στην παρακολούθηση και φτάνοντας έως τη συντήρηση. Μέσω των τεχνικών IaC, αναδεικνύονται τα πλεονεκτήματα της αυτοματοποιημένης διαχείρισης, η οποία ενισχύει την **αποτελεσματικότητα** και την **αξιοπιστία** των συστημάτων, διευκολύνοντας την ευέλικτη και συνεπή λειτουργία τους.

3.1. Βασικές Έννοιες Και Ορισμοί

DevOps

Προσέγγιση που ενσωματώνει πρακτικές μεταξύ της ανάπτυξης λογισμικού (Dev) και των λειτουργιών IT (Ops), με στόχο τη βελτίωση των σταδίων ανάπτυξης και παράδοσης λογισμικού. Αυτή η συνεργασία οδηγεί σε ταχύτερη παράδοση λογισμικού, με μεγαλύτερη ακρίβεια και λιγότερα σφάλματα.

DevSecOps

Επεκτείνουμε τη φιλοσοφία του DevOps, ενσωματώνοντας πρακτικές ασφάλειας στη διαδικασία ανάπτυξης. Αυτό σημαίνει ότι η ασφάλεια είναι μέρος της διαδικασίας από την αρχή, και όχι μια μεταγενέστερη προσθήκη.

GitOps

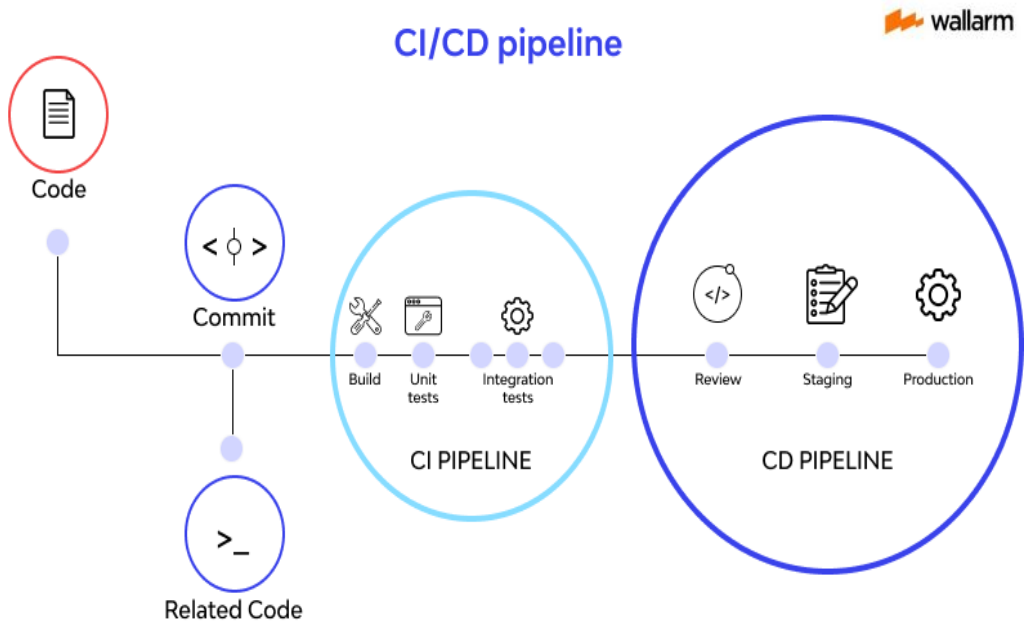
Προσέγγιση που χρησιμοποιούμε για τη διαχείριση των λειτουργιών, όπου οι κύριες αλλαγές στις υποδομές ή τις εφαρμογές διαχειρίζονται με τη χρήση Git. Σε αυτήν την προσέγγιση, το Git είναι η πηγή της αλήθειας και όλα διαχειρίζονται μέσω commits και pull requests, κάτι που προσφέρει διαφάνεια και αναπαραγωγιμότητα.

IaC (Infrastructure as Code)

Είναι η πρακτική μας για τη διαχείριση και παραμετροποίηση των υποδομών με τη χρήση κώδικα και δεδομένων. Αυτό επιτρέπει την αυτοματοποίηση της δημιουργίας και της διαχείρισης της υποδομής, με στόχο τη μείωση των σφαλμάτων και τη βελτίωση της αποτελεσματικότητας.

CI/CD (Continuous Integration / Continuous Deployment)

Πρακτική που εφαρμόζουμε για τη συνεχή ένταξη του κώδικα σε ένα κεντρικό αποθετήριο και τη συνεχή παράδοσή του ή την ανάπτυξή του σε δοκιμαστικά και παραγωγικά περιβάλλοντα. Τα εργαλεία CI/CD, όπως το Jenkins ή το GitLab CI, αυτοματοποιούν τη διαδικασία αυτή, μειώνοντας τον ανθρώπινο παράγοντα και επιταχύνοντας τις κυκλοφορίες. Στην Εικόνα 5 παρουσιάζεται ένα παράδειγμα CI/CD pipeline.



Εικόνα 5. Παράδειγμα CI/CD - Πηγή: https://www.wallarm.com/what/what-is-ci- 1

Observability

Αναφέρεται στην ικανότητά μας να κατανοούμε την εσωτερική κατάσταση ενός συστήματος από τα εξωτερικά παραγόμενα στοιχεία, όπως logs, metrics και traces. Εργαλεία όπως το Prometheus και το Grafana βοηθούν στη συλλογή και ανάλυση αυτών των δεδομένων για την καλύτερη κατανόηση και διαχείριση της απόδοσης των συστημάτων.

HLF (Hyperledger Fabric)

Συντομογραφία για το Hyperledger Fabric, ένα ανοιχτού κώδικα πλαίσιο για την ανάπτυξη blockchain εφαρμογών, το οποίο προσφέρει μια ευέλικτη και ασφαλή πλατφόρμα για την ανάπτυξη λύσεων.

Non-functional Requirements (NFRs)

Αφορούν τα χαρακτηριστικά ποιότητας του συστήματος, όπως η απόδοση, η ασφάλεια, και η αξιοπιστία, και δεν σχετίζονται άμεσα με τη λειτουργικότητα της εφαρμογής. Αυτές οι απαιτήσεις καθορίζουν πόσο καλά λειτουργεί η εφαρμογή και ποια είναι η συνολική εμπειρία του χρήστη.

Desired State

Είναι η επιθυμητή κατάσταση που θέλουμε να επιτύχουμε για ένα σύστημα, και συνήθως χρησιμοποιείται στη διαχείριση υποδομών και λειτουργιών για να καθοδηγεί τη δημιουργία και τη διαχείριση των πόρων.

SDLC (Software Development Life Cycle)

Είναι η διαδικασία μέσω της οποίας δημιουργούμε λογισμικά, περιλαμβάνοντας φάσεις όπως η ανάλυση απαιτήσεων, ο σχεδιασμός, η ανάπτυξη, οι δοκιμές και η συντήρηση. Τα μοντέλα SDLC, όπως το Agile και το Waterfall, παρέχουν διαφορετικές προσεγγίσεις για την ανάπτυξη λογισμικού.

SCA (Software Composition Analysis)

Η διαδικασία ανάλυσης του λογισμικού για να καθορίσουμε τις εξαρτήσεις του και να εντοπίσουμε πιθανά ζητήματα ασφαλείας. Η χρήση εργαλείων όπως το Black Duck ή το Snyk βοηθάει στην ανίχνευση ευπαθειών στις εξαρτήσεις.

SAST (Static Application Security Testing)

Η διαδικασία δοκιμής του κώδικα λογισμικού για ζητήματα ασφαλείας, χωρίς την εκτέλεσή του. Αυτό μπορεί να γίνει χρησιμοποιώντας εργαλεία όπως το SonarQube για να εντοπίσουμε σφάλματα και τρωτά σημεία στον κώδικα.

DAST (Dynamic Application Security Testing)

Η διαδικασία δοκιμής μιας εφαρμογής κατά την εκτέλεσή της, για ζητήματα ασφαλείας. Τα εργαλεία όπως το OWASP ZAP επιτρέπουν τη διερεύνηση των ζητημάτων που εμφανίζονται κατά την αλληλεπίδραση με την εφαρμογή.

Secrets

Στον τομέα της πληροφορικής, τα secrets αναφέρονται σε δεδομένα που πρέπει να παραμένουν απόρρητα, όπως κωδικοί πρόσβασης, κλειδιά API ή πιστοποιητικά. Η διαχείριση αυτών των secrets γίνεται συχνά με εργαλεία όπως το HashiCorp Vault ή το AWS Secrets Manager.

Code Reusability

Αναφέρεται στην ικανότητα ενός τμήματος κώδικα να χρησιμοποιηθεί εκ νέου σε διαφορετικά μέρη ή εφαρμογές, με στόχο την αύξηση της αποδοτικότητας και τη μείωση της επανάληψης. Αυτό εξασφαλίζει πιο καθαρό κώδικα και μειώνει τον χρόνο ανάπτυξης.

Non-functional Requirements (NFRs)

Οι μη λειτουργικές απαιτήσεις (NFRs) αναφέρονται σε προδιαγραφές που δεν σχετίζονται άμεσα με συγκεκριμένες λειτουργίες μιας εφαρμογής, αλλά μάλλον με την ποιότητα της εφαρμογής. Καθορίζουν το "πώς" πρέπει να λειτουργεί μια εφαρμογή, παρά το "τι" πρέπει να κάνει.

Ενδεικτικά, μερικές από τις συχνότερες NFRs σε ένα υπολογιστικό περιβάλλον περιλαμβάνουν:

- **Απόδοση**: Πόσο γρήγορα η υπηρεσία ανταποκρίνεται στα αιτήματα των χρηστών.
- **Αντοχή και Σταθερότητα**: Η ικανότητα της υπηρεσίας να λειτουργεί σωστά και χωρίς διακοπές.
- **Κλιμακωσιμότητα**: Η ικανότητα της υπηρεσίας να διαχειρίζεται αυξανόμενο φόρτο εργασίας.
- **Ασφάλεια**: Προστασία από επιθέσεις, αδικαιολόγητη πρόσβαση και απώλεια δεδομένων.
- **Διαθεσιμότητα**: Ο χρόνος κατά τον οποίο η υπηρεσία είναι διαθέσιμη για χρήση.
- **Αξιοπιστία**: Η ικανότητα της υπηρεσίας να λειτουργεί όπως αναμένεται και χωρίς απρόβλεπτα προβλήματα.

3.2. Συμβολή Των Μεθοδολογιών Devops, Gitops Στην Επίτευξη Των Μη Λειτουργικών Απαιτήσεων

- **DevOps**: Η μεθοδολογία DevOps ενισχύει τη συνεργασία μεταξύ των ομάδων ανάπτυξης και λειτουργίας, μειώνοντας τον χρόνο διανομής και αυξάνοντας την αντοχή των εφαρμογών. Εργαλεία όπως το **Jenkins** και το **GitLab CI** χρησιμοποιούνται για την αυτοματοποίηση των διαδικασιών συνεχούς ολοκλήρωσης και παράδοσης.
- **GitOps**: Αυτή η μεθοδολογία επιτρέπει τη διαχείριση υποδομών ως κώδικα, προωθώντας την αυτοματοποίηση και τη διαφάνεια. Εργαλεία όπως το **ArgoCD** και το **Flux** ενσωματώνονται άμεσα με το Git, επιτρέποντας την αυτοματοποίηση των διαδικασιών και τη συνεχή ενημέρωση της υποδομής.
- **Infrastructure as Code (IaC)**: Με το IaC, οι υποδομές προγραμματίζονται, δοκιμάζονται και αναπαράγονται με συνέπεια, εξασφαλίζοντας υψηλά επίπεδα αξιοπιστίας και διαθεσιμότητας. Εργαλεία όπως το **Terraform** και το **Ansible** επιτρέπουν την προγραμματιστική διαχείριση και ανάπτυξη πόρων.

- **CI/CD**: Η συνεχής ολοκλήρωση και παράδοση βελτιώνει την ταχύτητα και την ποιότητα της ανάπτυξης, με δυνατότητα τακτικών δοκιμών, αυτοματοποιημένων αναθεωρήσεων και γρήγορης αποκατάστασης από σφάλματα.

Συνολικά, αυτές οι μεθοδολογίες ενισχύουν τα μη λειτουργικά χαρακτηριστικά (NFRs), προσφέροντας ταχύτητα, αξιοπιστία, ασφάλεια και κλιμακωσιμότητα στις υπολογιστικές υπηρεσίες μας.

3.3. Λίστα Επιθέσεων Από Τις Οποίες Προστατεύει Η Μεθοδολογία

Η ενσωμάτωση των μεθοδολογιών **DevSecOps**, **Infrastructure as Code (IaC)** και **GitOps** παρέχει προστασία από διάφορες επιθέσεις και ευπάθειες στον χώρο της ασφάλειας λογισμικού και υποδομών, αντιμετωπίζοντας με συστηματικό τρόπο τις απειλές. Ακολουθούν ορισμένες επιθέσεις από τις οποίες μπορούν να προστατεύουν αυτές οι προσεγγίσεις:

1. **Μη εξουσιοδοτημένες αλλαγές**: Με το **GitOps**, όλες οι αλλαγές στην υποδομή γίνονται μέσω pull requests και ελέγχων στο Git, επιτρέποντας την ανίχνευση και αποτροπή μη εξουσιοδοτημένων αλλαγών.
2. **Ευπάθειες σε βιβλιοθήκες και εξαρτήσεις λογισμικού**: Η ενσωμάτωση ελέγχων ασφαλείας στις pipelines του **CI/CD** βοηθά στον εντοπισμό και την αποφυγή χρήσης ευάλωτων βιβλιοθηκών.
3. **Misconfigurations (κακή παραμετροποίηση)**: Το **IaC** επιτρέπει την αυτόματη ανίχνευση παραμετροποιήσεων, αποτρέποντας ανοικτές πόρτες σε firewalls ή μη ασφαλείς ρυθμίσεις στα containers.
4. **Drift στην υποδομή**: Με το GitOps, παρακολουθούνται οι αλλαγές στην υποδομή και συγκρίνονται με το configuration που υπάρχει στον κώδικα, εντοπίζοντας και αποκαθιστώντας αποκλίσεις (drift) από την επιθυμητή κατάσταση.
5. **Ανεπαρκής έλεγχος ταυτότητας και εξουσιοδότησης**: Η αυτοματοποιημένη διαχείριση ταυτοτήτων και δικαιωμάτων με **RBAC** και **IAM** πολιτικές μέσω IaC αποτρέπει ανεπαρκείς ελέγχους πρόσβασης.
6. **Εκτεθειμένα μυστικά (secrets)**: Εργαλεία όπως το **HashiCorp Vault**, **Sealed Secrets** ή **Secrets Manager** ενσωματώνονται στα pipelines για την προστασία ευαίσθητων πληροφοριών.
7. **Επιθέσεις μέσω κοινωνικής μηχανικής και phishing**: Ο αυτοματοποιημένος έλεγχος και η ελεγχόμενη πρόσβαση μειώνουν τον κίνδυνο εισαγωγής κακόβουλου κώδικα μέσω κοινωνικής μηχανικής.
8. **Αδυναμία αποτροπής DDoS επιθέσεων**: Παρακολούθηση και αυτοματοποιημένη ανταπόκριση (π.χ., **auto-scaling** και **WAF policies**) μπορούν να ενσωματωθούν μέσω IaC για τον μετριασμό των επιπτώσεων.
9. **Ευπάθειες containers και images**: Σαρωτικά εργαλεία στις pipelines (π.χ., **Trivy** ή **Clair**) εντοπίζουν γνωστές ευπάθειες σε container images πριν την ανάπτυξη.
10. **Απώλεια δεδομένων λόγω ανθρώπινου λάθους**: Αυτοματοποιημένες διαδικασίες επαναφοράς (rollback) και αποκατάστασης περιορίζουν τις συνέπειες ανθρώπινων λαθών.
11. **Supply Chain Attacks**: Συνεχής παρακολούθηση των εξαρτήσεων και των artifacts που χρησιμοποιούνται στο build αποτρέπει επιθέσεις στην αλυσίδα εφοδιασμού λογισμικού.

Η εφαρμογή αυτών των πρακτικών συμβάλλει στην οικοδόμηση μιας ασφαλέστερης, αυτοματοποιημένης υποδομής, μειώνοντας τον κίνδυνο επιτυχημένων επιθέσεων και εξασφαλίζοντας την ακεραιότητα των συστημάτων.

3.4. Ανάλυση Συγχρόνων Μεθόδων Κύκλου Ζωής Λογισμικού

Στον σύγχρονο κόσμο της ανάπτυξης λογισμικού, οι μέθοδοι κύκλου ζωής (Software Development Life Cycle, **SDLC**) έχουν εξελιχθεί από τις παραδοσιακές, κατακερματισμένες προσεγγίσεις προς πιο ευέλικτα και διαδραστικά μοντέλα. Αυτές οι μέθοδοι επιδιώκουν τη **βελτιστοποίηση του χρόνου ανάπτυξης**, την **ευελιξία απέναντι στις αλλαγές** και τη **συνεχή βελτίωση της ποιότητας**. Μοντέλα όπως το **Agile**, το **Scrum** και το **Lean** προσφέρουν μια πιο δυναμική και συνεργατική προσέγγιση, προωθώντας την ταχεία ανατροφοδότηση και την προσαρμογή στις απαιτήσεις των χρηστών. Μέσω αυτών των προσεγγίσεων, η ανάπτυξη λογισμικού καθίσταται πιο ευέλικτη, προσαρμοστική και εστιασμένη στη δημιουργία αξίας για τους χρήστες.

3.5. Παρουσίαση Των Μεθόδων Devops Για Ανάπτυξη Και Λειτουργία Συστημάτων

Οι σύγχρονες μέθοδοι ανάπτυξης λογισμικού και διαχείρισης υποδομών ενσωματώνουν πλήθος εργαλείων και πρακτικών, ενισχύοντας την **αποδοτικότητα**, την **ασφάλεια** και την **αξιοπιστία**. Ακολουθεί μια ανασκόπηση των κυριότερων μεθοδολογιών και πρακτικών:

1. **DevOps**: Συνδυάζει την ανάπτυξη (Dev) και τις λειτουργίες (Ops) με στόχο τη βελτίωση της συνεργασίας και την αύξηση της αποδοτικότητας μέσω της συνεχούς ολοκλήρωσης και ανάπτυξης (CI/CD). Με εργαλεία όπως το **Jenkins** και το **GitLab CI**, επιτυγχάνεται αυτοματοποίηση, μειώνοντας τα σφάλματα κατά την ανάπτυξη.
2. **DevSecOps**: Ενσωματώνει την ασφάλεια (Sec) σε κάθε στάδιο του SDLC, προωθώντας την "ασφάλεια από το σχεδιασμό". Αυτή η μεθοδολογία μειώνει τον κίνδυνο παραβιάσεων και αυξάνει την ασφάλεια του προϊόντος.
3. **Infrastructure as Code (IaC)**: Με το IaC, οι υποδομές διαχειρίζονται μέσω κώδικα, αποφεύγοντας χειροκίνητες διαδικασίες. Εργαλεία όπως το **Terraform**, το **Ansible** και το **Chef** επιτρέπουν την ταχεία δημιουργία και διαχείριση υποδομών με επαναληψιμότητα.
4. **Containers και Container Orchestration**: Τα **containers** (π.χ., Docker) προσφέρουν ελαφριά και απομονωμένα περιβάλλοντα, ενώ η ενορχήστρωσή τους με το **Kubernetes** εξασφαλίζει αυτοματοποιημένη διαχείριση, κλιμάκωση και υψηλή αξιοπιστία.
5. **Immutable Infrastructure**: Αυτή η πρακτική επιδιώκει τη δημιουργία και αντικατάσταση υποδομών χωρίς τροποποιήσεις, προσφέροντας συνέπεια και μειώνοντας τα σφάλματα.
6. **CI/CD για Κώδικα, Υποδομές και Πολιτικές**: Η συνεχής ολοκλήρωση και διανομή (CI/CD) επιτρέπει τη διαχείριση του κώδικα, των υποδομών και των πολιτικών με αυτοματοποίηση, προσφέροντας συνεπή και αποδοτική ανάπτυξη.
7. **Observability**: Εργαλεία όπως το **Prometheus** και το **Grafana** επιτρέπουν τη συλλογή και ανάλυση μετρικών, logs και traces για καλύτερη εποπτεία. Το observability είναι ιδιαίτερα σημαντικό για σύνθετα συστήματα και μικροϋπηρεσίες, επιτρέποντας την ταχεία διάγνωση και αντιμετώπιση προβλημάτων.
8. **Shift-Left**: Προωθεί τον εντοπισμό και τη διόρθωση προβλημάτων νωρίς στον κύκλο ανάπτυξης μέσω δοκιμών και αυτοματοποιημένων ελέγχων, μειώνοντας το κόστος αντιμετώπισης ζητημάτων σε μεταγενέστερα στάδια.
9. **GitOps**: Χρησιμοποιεί το Git για την αυτοματοποίηση και τη διαχείριση υποδομών και εφαρμογών, με εργαλεία όπως το **ArgoCD** και το **Flux**. Με κάθε αλλαγή στο Git αποθετήριο, ενεργοποιούνται αυτόματα διαδικασίες ενημέρωσης, διασφαλίζοντας διαφάνεια και ελέγχους πριν από την εφαρμογή αλλαγών.

Η συνδυασμένη χρήση αυτών των μεθοδολογιών και εργαλείων προσφέρει **υψηλά επίπεδα απόδοσης, ασφάλειας και αξιοπιστίας**, βελτιστοποιώντας παράλληλα την αποδοτικότητα και την ταχύτητα των διαδικασιών ανάπτυξης και λειτουργίας λογισμικού.

3.5.1. Ενδεικτική Λίστα Ελέγχου Με Εστίαση Στις Μη Λειτουργικές Απαιτήσεις

Ακολουθεί μια γενική λίστα ελέγχου (checklist) για κάθε στάδιο ανάπτυξης, εστιασμένη στην ενίσχυση των μη λειτουργικών απαιτήσεων (απόδοση, ασφάλεια, διαθεσιμότητα, αξιοπιστία, συμβατότητα) σε ένα DevOps περιβάλλον με DevSecOps πρακτικές και χρήση Infrastructure as Code (IaC):

1. Σχεδιασμός (Plan)

- **Αξιολόγηση Απειλών και Ασφάλειας**:
  - Εκτέλεση threat modeling για αναγνώριση πιθανών επιθέσεων (π.χ., DDoS, παραβίαση δεδομένων).
  - Καθορισμός security policies (π.χ., IAM roles, RBAC) και εφαρμογή των σε όλα τα συστήματα.
- **Ποιότητα και Συμμόρφωση**:
  - Ορισμός πολιτικών ποιότητας και πρότυπων (π.χ., ISO 27001, GDPR, HIPAA) που πρέπει να πληροί το σύστημα.
  - Αξιολόγηση τεχνικών απαιτήσεων για υλοποίηση high availability και disaster recovery.
- **Υποστήριξη Κλιμάκωσης και Διαθεσιμότητας**:
  - Ανάλυση και προδιαγραφή στρατηγικών κλιμάκωσης (π.χ., horizontal/vertical scaling).
  - Σχεδιασμός υβριδικών λύσεων για διαχείριση φόρτου (on-premise, cloud, multi-cloud).

2. Υλοποίηση Κώδικα (Code)

- **Αξιολόγηση και Αναθεώρηση Κώδικα**:
  - Χρήση εργαλείων στατικής ανάλυσης (SAST) για έλεγχο ασφαλείας κώδικα (π.χ., SonarQube, Checkmarx).
  - Ομαδικές αναθεωρήσεις κώδικα (peer reviews) για ανίχνευση προβλημάτων σε ασφάλεια και ποιότητα.
- **Προδιαγραφές για Παρατηρησιμότητα (Observability)**:
  - Ενσωμάτωση μετρικών και logs στον κώδικα για κάθε μικροϋπηρεσία.
  - Προδιαγραφή tracing points για distributed tracing (π.χ., Jaeger, Zipkin).
- **Διαχείριση Εξαρτήσεων και Συμβατότητας**:
  - Χρήση εργαλείων για ανάλυση εξαρτήσεων (SCA) και ανίχνευση ευπαθειών.
  - Διασφάλιση συμβατότητας κώδικα με τις υποστηριζόμενες πλατφόρμες και τεχνολογίες.

3. Δόμηση και Σύνθεση (Build)

- **Διασφάλιση Ποιότητας Κατασκευής**:
  - Ενσωμάτωση διαδικασιών ελέγχου ασφαλείας των images/container (π.χ., Trivy, Aqua Security).
  - Αυτόματοι έλεγχοι formating και syntax linting για consistency στον κώδικα.
- **Παρατηρησιμότητα Build**:
  - Εφαρμογή tracing στα builds για εύκολη διάγνωση προβλημάτων.
  - Διαχείριση versions και tags για κάθε νέο build.

4. Δοκιμές (Test)

- **Ασφάλεια και Ποιότητα Κώδικα**:
    - Εκτέλεση δυναμικών αναλύσεων (DAST) για ανίχνευση ευπαθειών στο εκτελούμενο λογισμικό.
    - Penetration testing σε βασικές λειτουργίες και APIs για διασφάλιση ασφαλούς παραγωγής.
- **Δοκιμές Απόδοσης και Κλιμάκωσης**:
    - Διεξαγωγή stress tests, load tests, και endurance tests για αξιολόγηση της απόδοσης σε συνθήκες αυξημένου φόρτου.
    - Επαλήθευση της συμπεριφοράς των υπηρεσιών υπό κλιμακωτή χρήση (auto-scaling).

5. Κυκλοφορία και Συμμόρφωση (Release)

- **Επαλήθευση Συμμόρφωσης**:
    - Εφαρμογή compliance checks πριν την κυκλοφορία (π.χ., GDPR, SOC 2).
    - Έλεγχος αν η υποδομή πληροί όλους τους κανονισμούς και τα πρότυπα.
- **Διασφάλιση Διαθεσιμότητας**:
    - Στρατηγικές ανάπτυξης όπως blue-green ή canary για αποφυγή διακοπών κατά την παραγωγή.
    - Ορισμός checkpoints και rollback mechanisms για έγκαιρη αποκατάσταση.

6. Ανάπτυξη (Deploy)

- **Παρακολούθηση και Έλεγχος Υποδομής**:
    - Δοκιμές integrity και monitoring των υποδομών μετά την ανάπτυξη.
    - Αυτόματη καταγραφή και παρακολούθηση logs από κάθε component (π.χ., ELK, Loki).
- **Διασφάλιση Αξιοπιστίας και Διαθεσιμότητας**:
    - Αυτόματοι έλεγχοι αποδοτικότητας/συμμόρφωσης των IaC configurations με το αναμενόμενο αποτέλεσμα.
    - Επαλήθευση ύπαρξης disaster recovery points και σχεδίων αποκατάστασης.

7. Λειτουργία και Παρακολούθηση (Operate)

- **Συνεχής Παρακολούθηση και Ανίχνευση**:
    - Συνεχής συλλογή μετρήσεων και logs από Prometheus/Grafana και εφαρμογή threshold-based alerts.
    - Καθημερινή ανάλυση logs για ανίχνευση ανωμαλιών και πιθανών απειλών.
- **Διαχείριση Παρατηρησιμότητας και Αντιμετώπιση**:
    - Χρήση APM για συνεχή παρακολούθηση απόδοσης και διάγνωση bottlenecks.
    - Ορισμός διαδικασιών αντιμετώπισης συμβάντων (incident response) για έγκαιρη αντιμετώπιση προβλημάτων.

8. Βελτιστοποίηση και Συντήρηση (Optimize & Maintain)

- **Βελτιστοποίηση Πόρων και Υποδομών**:
    - Κανονικός έλεγχος των IaC για αναβάθμιση configurations και περιορισμό του drift.
    - Αξιολόγηση και ανανέωση των containers και των dependencies για βέλτιστη απόδοση.
- **Αναθεώρηση Πολιτικών και Συμμόρφωσης**:
    - Περιοδική επαναξιολόγηση των security policies και των IAM ρυθμίσεων.

- Διατήρηση του SBOM και ενημέρωση για νέες ευπάθειες σε libraries.

Αυτή η λίστα ελέγχου προσφέρει ένα συνολικό πλαίσιο για τη διασφάλιση της απόδοσης, της ασφάλειας και της συμβατότητας σε κάθε στάδιο ανάπτυξης, καλύπτοντας πλήρως τις μη λειτουργικές απαιτήσεις της υποδομής και του λογισμικού.

4. Μελέτη Περίπτωσης: Ανάπτυξη Ιεραρχικής Πολυεπιπεδης αρχιτεκτονικής Blockchain

Τα βασικά στάδια του κύκλου ζωής της υποδομής περιλαμβάνουν τον **σχεδιασμό**, την **ανάπτυξη**, την **παρακολούθηση** και τη **συντήρηση**, με έμφαση στη χρήση τεχνικών **Infrastructure as Code (IaC)** για αυξημένη αποτελεσματικότητα και αξιοπιστία. Σε αυτό το πλαίσιο, χρησιμοποιούμε **Git repositories** για την υλοποίηση μιας υποδομής που φιλοξενεί υπηρεσίες βασισμένες σε **Kubernetes**. Η υποδομή δημιουργείται με τη χρήση του **Terraform** για δηλωτική διαχείριση πόρων σε cloud providers, όπως το **AWS** ή το **Azure**, επιτρέποντας μια σαφή περιγραφή των επιθυμητών αποτελεσμάτων χωρίς λεπτομερή καθοδήγηση για κάθε βήμα υλοποίησης.

Για τη **συνεχή ενσωμάτωση και παράδοση (CI/CD)**, εφαρμόζουμε pipelines που διασφαλίζουν την αυτοματοποίηση του ελέγχου ποιότητας, της ασφάλειας, του build, και της ανάπτυξης του κώδικα στην υποδομή. Αυτό επιτρέπει όλες οι αλλαγές να δοκιμάζονται και να ελέγχονται για θέματα ασφάλειας, εξασφαλίζοντας ότι το λογισμικό είναι έτοιμο για παραγωγική χρήση.

Η **παρακολούθηση και παρατήρηση της λειτουργίας** της υπηρεσίας επιτυγχάνεται μέσω εργαλείων όπως το **Prometheus** και το **Grafana**. Η συνεχής ανάλυση των μετρικών επιτρέπει την αποδοτική λειτουργία της υποδομής και την έγκαιρη ανίχνευση προβλημάτων. Με αυτόν τον τρόπο, βελτιώνεται η αξιοπιστία των συστημάτων και εξασφαλίζεται η άμεση ανταπόκριση σε πιθανές βλάβες, ενισχύοντας την απόδοση και τη σταθερότητα της υπηρεσίας.

4.1. Υλοποίηση – Αξιολόγηση Και Σύγκριση Επιλογων

Στη συγκεκριμένη ενότητα παρουσιάζεται μια μελέτη περίπτωσης ενός πραγματικού συστήματος που εφαρμόζει τις μεθοδολογίες DevSecOps, GitOps και Infrastructure as Code (IaC). Το σύστημα περιλαμβάνει ένα πλήρες περιβάλλον ανάπτυξης με τη χρήση εξειδικευμένων εργαλείων και τεχνολογιών, προσφέροντας ένα πρακτικό παράδειγμα της εφαρμογής αυτών των προσεγγίσεων.

Η ανάλυση αυτή στοχεύει να αναδείξει:

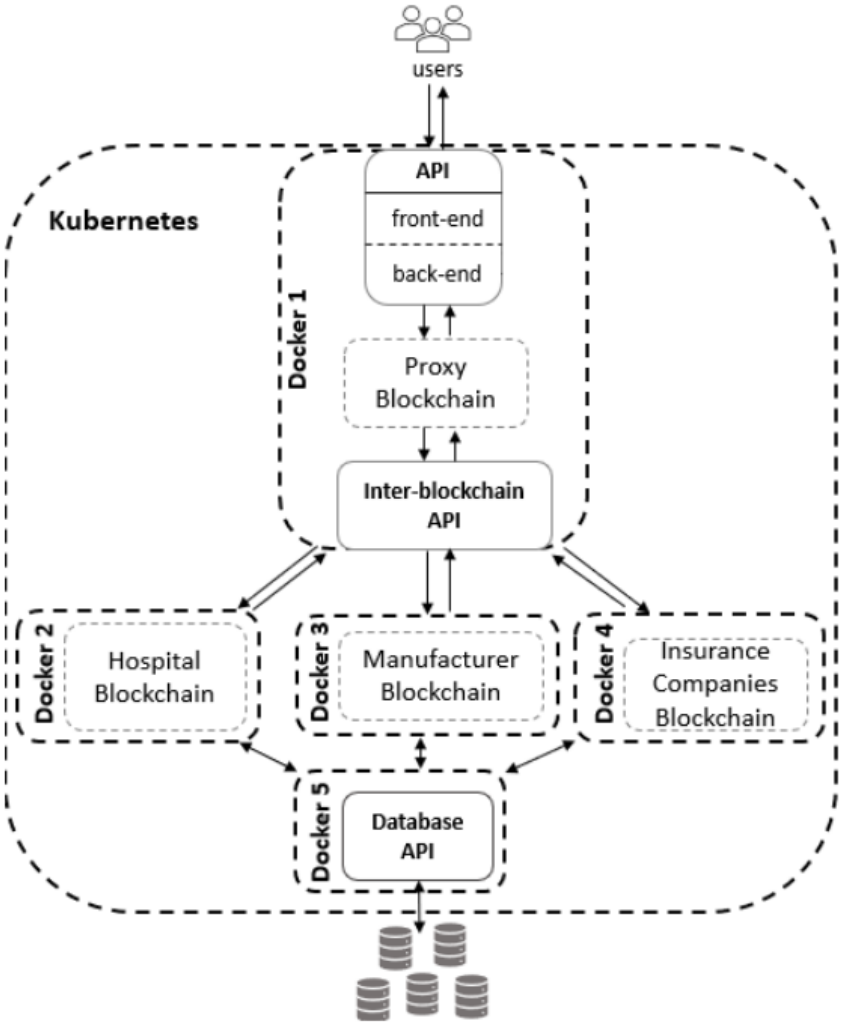
- Τα οφέλη από τη χρήση αυτών των μεθοδολογιών, όπως η βελτίωση της ασφάλειας, της ταχύτητας και της αξιοπιστίας,
- Τις προκλήσεις που μπορεί να προκύψουν κατά την ενσωμάτωσή τους σε υπάρχοντα συστήματα ή διαδικασίες, όπως η ανάγκη για εκπαίδευση ή η προσαρμογή σε νέα εργαλεία,
- Τις βέλτιστες πρακτικές που συμβάλλουν στην επιτυχή υλοποίηση, όπως η τήρηση των αρχών "ασφάλειας από το σχεδιασμό" στο DevSecOps, η χρήση του Git ως μοναδικής πηγής αλήθειας στο GitOps και η αυτοματοποίηση της υποδομής μέσω IaC.

Αυτή η μελέτη περίπτωσης προσφέρει μια ολοκληρωμένη εικόνα του τρόπου με τον οποίο οι προσεγγίσεις DevSecOps, GitOps και IaC μπορούν να εφαρμοστούν σε πραγματικό περιβάλλον, εξασφαλίζοντας την αποτελεσματική και ασφαλή διαχείριση των συστημάτων.

4.2. Περιγραφή Υπηρεσίας

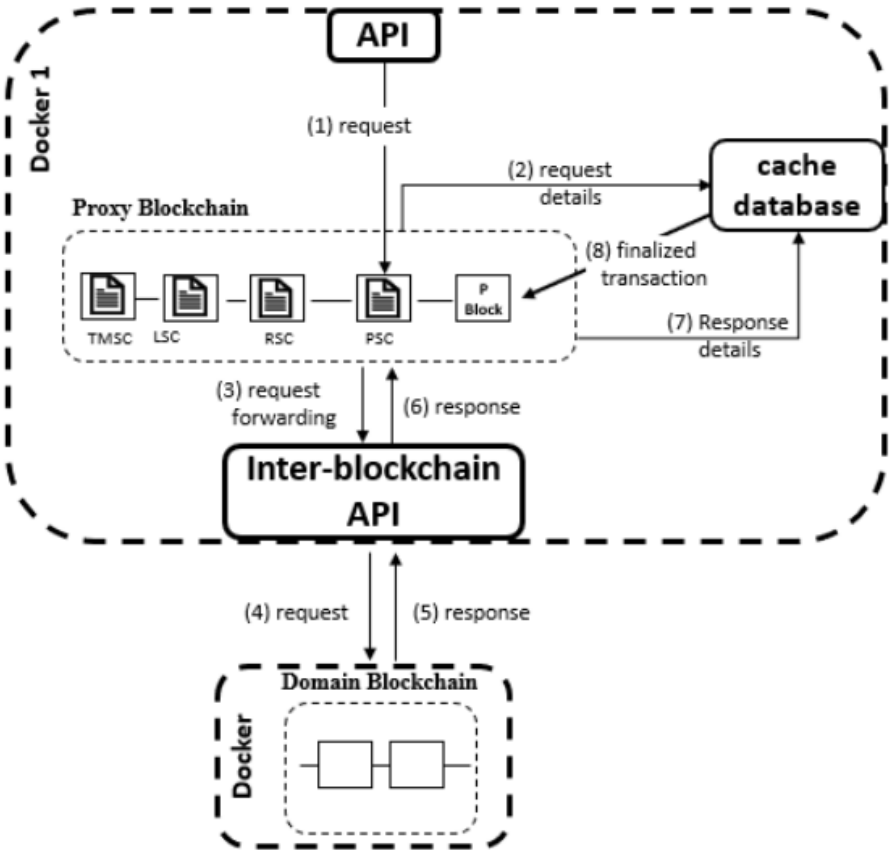
Η υπηρεσία που θα υλοποιηθεί αφορά την αρχιτεκτονική που περιγράφεταιι στη δημοσίευση (Malamas, Kotzanikolaou, Dasaklis, & Burmester, 2020) [35], η οποία είναι μια αρχιτεκτονική ιεραρχικού πολυεπίπεδου blockchain για τον λεπτομερή έλεγχο πρόσβασης σε ιατρικά δεδομένα. Η αρχιτεκτονική αυτή χρησιμοποιεί ένα proxy blockchain για να επιτρέπει την ασφαλή διαλειτουργικότητα μεταξύ διαφορετικών τομέων (όπως νοσοκομεία και κατασκευαστές ιατρικών συσκευών) και επιτρέπει την εφαρμογή πολιτικών πρόσβασης τόσο σε επίπεδο τομέα όσο και σε διατομεακό επίπεδο. Χρησιμοποιεί κρπτογράφηση Attribute-Based Encryption (ABE) για την προστασία των δεδομένων και την κατανομή της διαδικασίας αποκρυπτογράφησης, παρέχοντας

ένα αποδοτικό και επεκτάσιμο σύστημα για την ασφαλή ανταλλαγή ιατρικών πληροφοριών. Η αρχιτεκτονική παρουσιάζεται στην Εικόνα 6.



Εικόνα 6 Αρχιτεκτονική υπηρεσιας

Για τη διαχείριση της αρχιτεκτονικής, τα Docker containers και οι κόμβοι των blockchain λειτουργούν με απομόνωση για να διασφαλίζεται η ασφάλεια και η σταθερότητα του συστήματος. Χρησιμοποιώντας το σύστημα ενορχήστρωσης Kubernetes, μπορούμε να διαχειριζόμαστε αποτελεσματικά τα containers, εξασφαλίζοντας ότι κάθε λειτουργικό συστατικό εκτελείται με ασφάλεια και αποδοτικότητα. Η ενορχήστρωση μέσω Kubernetes προσφέρει τη δυνατότητα δυναμικής χρήσης των containers και εξισορρόπηση των πόρων του συστήματος με δυναμική κατανομή, βελτιώνοντας τη σταθερότητα, ακόμα και υπό συνθήκες αυξημένης ζήτησης. Λεπτομέρειες της αρχιτεκτονικής παρουσιάζονται στην Εικόνα 7.



Εικόνα 7 Αρχιτεκτονική υπηρεσίας - λεπτομέρειες συναλλαγών

Οι κόμβοι του συστήματος λειτουργούν σε απομονωμένα περιβάλλοντα, με τα Domain Blockchains (DBCs) να συνδέονται μέσω του inter-Blockchain API με το Proxy Blockchain (PBC) και με τις φυσικές βάσεις δεδομένων μέσω του κοινόχρηστου API. Αυτή η αρχιτεκτονική μας επιτρέπει να έχουμε μια ευέλικτη και επεκτάσιμη λύση, η οποία διασφαλίζει ότι οι συμμετέχοντες μπορούν να έχουν πρόσβαση στις απαραίτητες πληροφορίες χωρίς να παραβιάζεται η ασφάλεια του συστήματος. Το ενδιάμεσο API λειτουργεί επίσης ως buffer για την αποθήκευση προσωρινών δεδομένων, περιμένοντας να αποθηκευτούν μόνιμα, και ελέγχει τη ροή των δεδομένων μεταξύ των blockchain, προσφέροντας αποτελεσματικό έλεγχο της επικοινωνίας.

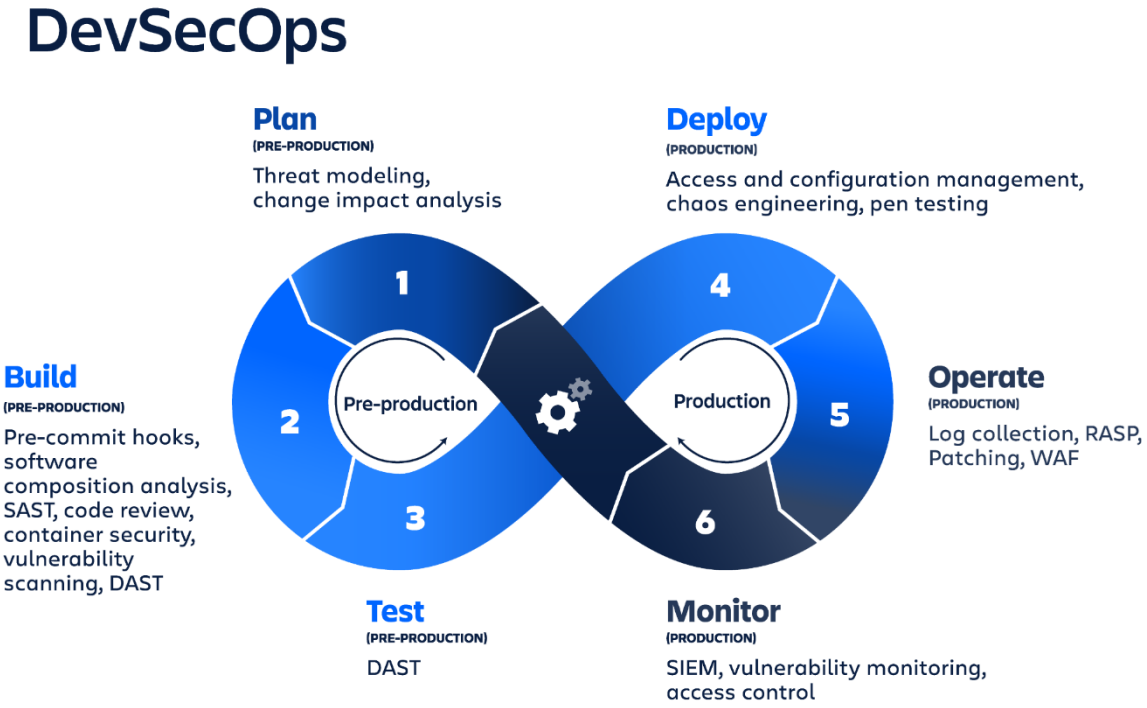
Για την υλοποίηση του ιδιωτικού blockchain, χρησιμοποιήθηκαν εργαλεία βασισμένα στο Ethereum, όπως το ganache-cli για τη δημιουργία ενός τοπικού περιβάλλοντος δοκιμών και το Truffle για την ανάπτυξη των έξυπνων συμβολαίων. Τα έξυπνα συμβόλαια αναπτύχθηκαν σε Solidity και προσομοιώνουν τις βασικές λειτουργίες του συστήματος, όπως η καταγραφή των συναλλαγών και η επικύρωση των δεδομένων, διασφαλίζοντας την ασφάλεια και την ακεραιότητα. Η χρήση της γλώσσας Solidity και των εργαλείων αυτών προσφέρει αυξημένη ασφάλεια και εγγυάται τη σωστή λειτουργία των έξυπνων συμβολαίων.

Με αυτή την αρχιτεκτονική, καταφέραμε να διασφαλίσουμε την απομόνωση των λειτουργιών και την αποδοτική διαχείριση των πόρων, διατηρώντας ταυτόχρονα υψηλά επίπεδα ασφάλειας. Η χρήση δύο ανεξάρτητων blockchain με την τεχνολογία Docker και Kubernetes μας επιτρέπει να προσαρμόζουμε το σύστημα ανάλογα με τις ανάγκες και τις απαιτήσεις των συμμετεχόντων, επιτυγχάνοντας αποδοτική κατανομή πόρων και ομαλή λειτουργία, ακόμα και υπό συνθήκες αυξημένης ζήτησης.

4.3. Κύκλος Ζωής Για Την Ανάπτυξη Υποδομής Ως Κώδικα

Σε αυτή την ενότητα θα αναλύσουμε κάθε στάδιο του κύκλου ζωής της ανάπτυξης υποδομής ως κώδικα (IaC), συνδυάζοντάς το με σύγχρονες πρακτικές για την ανάπτυξη και λειτουργία των υπηρεσιών πληροφορικής. Η ανάλυση αυτή παρέχει μία ολοκληρωμένη εικόνα των διαδικασιών που απαιτούνται για την επιτυχημένη υλοποίηση υποδομής ως κώδικα.

Στην Εικόνα 8 παρουσιάζεται η ενσωμάτωση της ασφάλειας σε έναν κύκλο ανάπτυξης λογισμικού με την μεθοδολογία DevSecOps.



Εικόνα 8 Ενσωμάτωση ασφάλειας σε έναν κύκλο ανάπτυξης Πηγή: https://www.browserstack.com/guide/devop 1

4.3.1. Στάδια του Κύκλου Ζωής

Plan

• Threat Modeling: Σε αυτό το στάδιο, αναγνωρίζουμε τις απειλές που μπορεί να αντιμετωπίσει η υπηρεσία, όπως επιθέσεις DDoS, παραβιάσεις δεδομένων ή μη εξουσιοδοτημένη πρόσβαση. Τα μοντέλα απειλής βοηθούν τις ομάδες να κατανοήσουν τους κινδύνους και να προβαίνουν σε σχεδιαστικές επιλογές που μειώνουν αυτούς τους κινδύνους.

• Policies: Ορίζουμε τις πολιτικές που πρέπει να τηρούνται κατά τη διάρκεια της ανάπτυξης και λειτουργίας της υπηρεσίας, συμπεριλαμβανομένων των προδιαγραφών ασφαλείας και ποιότητας, ώστε να διασφαλίσουμε ότι η υπηρεσία πληροί τις απαιτούμενες προδιαγραφές.

Code

• Static Analysis: Χρησιμοποιούμε εργαλεία για την αυτόματη ανάλυση του κώδικα χωρίς την εκτέλεσή του, με στόχο την εύρεση προβλημάτων ή ελλείψεων.

• Code Review: Η ομάδα ανάπτυξης ελέγχει τον κώδικα για σφάλματα, βέλτιστες πρακτικές και πιθανά ζητήματα ασφαλείας, βελτιώνοντας την ποιότητα και την ασφάλεια του παραγόμενου λογισμικού.

Build

• Σε αυτό το στάδιο, συγκεντρώνουμε τον κώδικα και δημιουργούμε την εφαρμογή ή την υπηρεσία, διασφαλίζοντας ότι όλα τα συστατικά είναι σε θέση να λειτουργήσουν ομαλά.

Test

• Dynamic Analysis: Εξετάζουμε τον κώδικα κατά τη διάρκεια της εκτέλεσής του για να ανακαλύψουμε προβλήματα που δεν μπορούν να ανιχνευθούν με στατική ανάλυση.

• Penetration Testing: Ειδικοί στην ασφάλεια προσπαθούν να "εισβάλουν" στην εφαρμογή ή την υπηρεσία για να εντοπίσουν τυχόν αδυναμίες, εξασφαλίζοντας ότι το σύστημα είναι ανθεκτικό σε εξωτερικές απειλές.

Release

• Compliance Validation: Ελέγχουμε αν η υπηρεσία συμμορφώνεται με τις σχετικές προδιαγραφές, πρότυπα και νόμους, διασφαλίζοντας ότι όλες οι απαιτήσεις πληρούνται πριν από την κυκλοφορία.

Deploy

• Logs: Καταγράφουμε τις δραστηριότητες και τα συμβάντα της υπηρεσίας για να διασφαλίσουμε την ορατότητα και τη δυνατότητα ανάλυσης των γεγονότων.

• Audit: Εκτελούμε μια εκτενή εξέταση της υπηρεσίας για να διαπιστώσουμε αν τηρούνται όλες οι πολιτικές και οι καλές πρακτικές.

Operate

• Monitor, Detect, Respond: Παρακολουθούμε συνεχώς τις δραστηριότητες της υπηρεσίας, ανιχνεύουμε πιθανά προβλήματα και λαμβάνουμε μέτρα αντίδρασης σε περίπτωση συμβάντων, διασφαλίζοντας την απρόσκοπτη λειτουργία.

- Monitor: Σε αυτό το στάδιο, παρακολουθούμε την απόδοση, τη σταθερότητα και την ασφάλεια της υπηρεσίας, καθώς και τις πιθανές αποκλίσεις από το επιθυμητό επίπεδο λειτουργίας, βελτιώνοντας τη συνολική αξιοπιστία.

Αυτή η ανάλυση παρέχει μια γενική εικόνα των διαδικασιών που συνοδεύουν την ανάπτυξη και τη λειτουργία μιας υπηρεσίας πληροφορικής. Είναι σημαντικό να σημειώσουμε ότι, ανάλογα με την οργάνωση και τη φύση της υπηρεσίας, ορισμένα από τα παραπάνω στάδια μπορεί να προσαρμόζονται ή να προστίθενται επιπλέον βήματα.

4.3.2. Επιμέρους Τεχνολογίες

Ας προσεγγίσουμε την ανάλυση και τη σύγκριση των τεχνολογιών που αναφέραμε ανά κατηγορία:

Υποδομή - Cloud Providers

• Azure: Υπηρεσία νέφους της Microsoft με μεγάλη γκάμα υπηρεσιών, όπως Azure Kubernetes Service (AKS), Azure Functions, Azure Virtual Machines, και Azure Cosmos DB. Περιλαμβάνει δυνατότητα ενσωμάτωσης με άλλα προϊόντα της Microsoft, προσφέροντας μια πλήρη λύση για ανάπτυξη και λειτουργία. [4]

• GCP (Google Cloud Platform): Πλατφόρμα νέφους της Google με εξειδίκευση σε υπηρεσίες ανάλυσης δεδομένων και machine learning. Περιλαμβάνει Google Kubernetes Engine (GKE), BigQuery, και πολλές άλλες υπηρεσίες που ενισχύουν τη δυνατότητα επεξεργασίας μεγάλων δεδομένων.

• AWS (Amazon Web Services): Ο μεγαλύτερος και πιο δημοφιλές πάροχος cloud με την πλουσιότερη γκάμα υπηρεσιών, όπως Elastic Kubernetes Service (EKS), Lambda, Redshift, και RDS. Παρέχει ευέλικτες λύσεις για ανάπτυξη, αποθήκευση και διαχείριση δεδομένων.

• IBM Cloud: Προσφέρει υβριδικό και multi-cloud περιβάλλον με ισχυρές λύσεις για AI και blockchain, καθιστώντας το κατάλληλο για επιχειρήσεις με εξειδικευμένες ανάγκες.

• Oracle Cloud: Προσανατολισμένο σε επιχειρήσεις με ισχυρή υποστήριξη για βάσεις δεδομένων και ERP εφαρμογές, προσφέροντας λύσεις με μεγάλη αξιοπιστία και δυνατότητα επέκτασης.

Infrastructure as Code (IaC)

• Terraform: Δημοφιλές ανεξάρτητο εργαλείο IaC με υποστήριξη για πολλά cloud platforms, προσφέροντας μεγάλη ευελιξία στη δημιουργία και διαχείριση υποδομών [21].

• Pulumi: Παρέχει μια πιο σύγχρονη προσέγγιση στο IaC με τη χρήση γλωσσών προγραμματισμού όπως Python, Go, και TypeScript, επιτρέποντας μεγαλύτερη ευελιξία στους προγραμματιστές.

• CloudFormation: Εργαλείο IaC από την AWS για την αυτοματοποίηση και διαχείριση πόρων AWS, ιδανικό για όσους είναι εξοικειωμένοι με το AWS οικοσύστημα.

• Ansible: Χρησιμοποιείται για αυτοματοποίηση, διαχείριση υποδομών και διαμόρφωση συστημάτων, επιτρέποντας τη δημιουργία συνεπών και επαναλήψιμων περιβαλλόντων.

• Chef & Puppet: Παρόμοια εργαλεία διαχείρισης υποδομών που προσφέρουν διαμόρφωση μέσω κώδικα, ιδανικά για μεγάλες επιχειρήσεις με πολύπλοκες ανάγκες.

Git Hosting

• GitLab: Πλατφόρμα διαχείρισης κώδικα με ενσωματωμένες λειτουργίες CI/CD, προσφέροντας ένα ολοκληρωμένο εργαλείο για την ανάπτυξη και την παράδοση λογισμικού.

• GitHub: Πλατφόρμα διαχείρισης κώδικα από τη Microsoft, πολύ δημοφιλής στην κοινότητα του ανοιχτού κώδικα. Προσφέρει επίσης GitHub Actions για CI/CD, διευκολύνοντας την αυτοματοποίηση της διαδικασίας ανάπτυξης.

• Bitbucket: Πλατφόρμα από την Atlassian, με ενσωματωμένες λειτουργίες CI/CD και στενή ενσωμάτωση με άλλα εργαλεία της Atlassian, όπως το Jira, για καλύτερη διαχείριση των projects.

Pre-commit hooks & Signing

• Χρησιμοποιούμε εργαλεία όπως το pre-commit και signing με GPG για την εξασφάλιση της ποιότητας και της αυθεντικότητας του κώδικα πριν το commit, διασφαλίζοντας ότι ο κώδικας πληροί τις απαιτούμενες προδιαγραφές και ότι η ταυτότητα του συντάκτη είναι αξιόπιστη.

CI/CD

• Jenkins (Declarative): Δημοφιλές εργαλείο CI με μεγάλη κοινότητα, υποστηρίζει παραμετροποίηση σε groovy scripts και παρέχει ευελιξία για σύνθετα pipelines.

• GitHub Actions: Υπηρεσία CI/CD εντός του GitHub, ιδανική για γρήγορη και εύκολη αυτοματοποίηση.

• GitLab CI: Ενσωματωμένες λειτουργίες CI/CD στο GitLab, προσφέροντας μια πλήρη λύση για ανάπτυξη και παράδοση λογισμικού.

• Azure DevOps: Υπηρεσίες CI/CD μέσω του Azure DevOps, ιδανικές για οργανισμούς που χρησιμοποιούν το Azure οικοσύστημα.

• CircleCI: Υπηρεσία CI/CD για αυτοματοποίηση των αναπτύξεων, παρέχοντας ευκολία και ταχύτητα στην παράδοση κώδικα.

• Travis CI: Υπηρεσία CI για open source και εμπορικά projects, με εύκολη ενσωμάτωση σε GitHub projects.

GitOps

• ArgoCD & Flux: Δημοφιλή εργαλεία για GitOps στο Kubernetes, προσφέροντας αυτοματοποίηση και συνεχή παρακολούθηση της κατάστασης της υποδομής. [1]

• Helm: Διαχειριστής πακέτων για Kubernetes, προσφέροντας εγκαταστάσεις και αναβαθμίσεις εφαρμογών με ευκολία. (Helm) [25]

• Jenkins X: Πλατφόρμα CI/CD βελτιστοποιημένη για Kubernetes με υποστήριξη GitOps, διευκολύνοντας την ανάπτυξη cloud-native εφαρμογών.

• Kustomize: Εργαλείο διαχείρισης παραμετροποιήσεων για Kubernetes χωρίς την ανάγκη scripting, διευκολύνοντας την ευελιξία στη διαμόρφωση.

Secrets Management

• HashiCorp Vault: Κεντρική διαχείριση κλειδιών και άλλων εμπιστευτικών δεδομένων, διασφαλίζοντας την ασφαλή αποθήκευση και πρόσβαση σε ευαίσθητες πληροφορίες.

• Vendor KMS: Υπηρεσίες διαχείρισης κλειδιών από cloud providers, όπως AWS KMS, Azure Key Vault, και Google Cloud KMS, προσφέροντας απλή και ασφαλή διαχείριση κλειδιών.

• SOPS: Εργαλείο για την κρυπτογράφηση και διαχείριση secrets με υποστήριξη για GitOps, επιτρέποντας την ασφαλή αποθήκευση secrets στο Git.

Artifacts

• Harbor: Ανοιχτού κώδικα registry για Docker και Helm, ιδανικό για ασφαλή αποθήκευση και διανομή εικόνων.

• Vendor registries: Cloud-specific registries, όπως AWS ECR, Azure Container Registry, και Google Artifact Registry, προσφέροντας ενσωμάτωση με τις υπηρεσίες του cloud provider.

• JFrog Artifactory: Δημοφιλής λύση για τη διαχείριση binary artifacts, υποστηρίζοντας πολλαπλές πλατφόρμες και μορφές αρχείων.

Monitoring

• Metrics:

Prometheus: Σύστημα παρακολούθησης και συλλογής μετρήσεων, ιδανικό για την παρακολούθηση της απόδοσης των εφαρμογών. [46]

Grafana: Δημιουργία διαγραμμάτων από δεδομένα monitoring, προσφέροντας οπτικοποίηση των μετρήσεων. [20]

InfluxDB: Βάση δεδομένων για μετρήσεις και χρονοσειρές, ιδανική για αποθήκευση δεδομένων υψηλής συχνότητας.

• Logs:

ELK (Elasticsearch, Logstash, Kibana): Πλατφόρμα αναζήτησης και ανάλυσης logs, προσφέροντας πλούσια δυνατότητα επεξεργασίας και οπτικοποίησης.

Loki: Σύστημα logging από τη Grafana Labs, σχεδιασμένο για εύκολη ενσωμάτωση με το Prometheus.

Fluentd: Εργαλείο συλλογής, αποστολής και κατανάλωσης logs, επιτρέποντας την εύκολη διαχείριση των δεδομένων καταγραφής.

• APM/Instrumentation:

Jaeger: Σύστημα ανίχνευσης για microservices, διευκολύνοντας την παρακολούθηση και την ανάλυση των αιτημάτων [29].

Sentry: Παρακολούθηση σφαλμάτων και εξαιρέσεων, προσφέροντας αναλυτική εικόνα για τα σφάλματα των εφαρμογών [50].

Datadog: Πλατφόρμα παρακολούθησης και ανάλυσης για cloud-scale εφαρμογές, παρέχοντας ενσωματωμένες δυνατότητες APM.

Vendor APM: Εξειδικευμένες λύσεις από cloud providers, όπως AWS X-Ray, Azure Monitor, και Google Cloud Operations, προσφέροντας ολοκληρωμένες δυνατότητες παρακολούθησης.

Policies

• OPA (Open Policy Agent): Ανοιχτού κώδικα, γενικής χρήσης policy engine για τη διαχείριση και εφαρμογή πολιτικών, προσφέροντας ευελιξία στη διαμόρφωση κανόνων ασφαλείας [41].

• Kyverno: Εργαλείο για τη διαχείριση πολιτικών στο Kubernetes, προσφέροντας δυνατότητα παραμετροποίησης κανόνων ασφαλείας με ευκολία.

• Falco: Σύστημα ανίχνευσης απειλών για Kubernetes, που βοηθά στην εφαρμογή πολιτικών ασφαλείας και παρακολούθησης σε πραγματικό χρόνο.

Αυτή η ανάλυση παρέχει μια γενική εποπτεία των τεχνολογιών που αναφέραμε, συνοδευόμενη από παραδείγματα για την καλύτερη κατανόηση των δυνατοτήτων και της χρήσης τους. Οι συγκρίσεις μεταξύ των τεχνολογιών εξαρτώνται από τις ανάγκες του εκάστοτε έργου, την υπάρχουσα υποδομή, το κόστος, και άλλους παράγοντες, όπως η εμπειρία της ομάδας και η συμβατότητα με υφιστάμενα συστήματα. Μπορούμε να επιλέξουμε τις κατάλληλες τεχνολογίες ανάλογα με τις συγκεκριμένες απαιτήσεις του κάθε έργου, διασφαλίζοντας τη βέλτιστη απόδοση, τη συνοχή, και την αποτελεσματικότητα στην υλοποίηση.

4.4. Ανάλυση Απαιτήσεων

1. Λειτουργικές Απαιτήσεις

- **Αυτοματοποιημένη Διαχείριση Υποδομών (IaC)**
  - Χρήση εργαλείων όπως Terraform, Ansible, ή Kubernetes για τον καθορισμό και τη διαχείριση της υποδομής μέσω κώδικα.
  - Διατήρηση των διαμορφώσεων σε αποθετήριο κώδικα (Git), επιτρέποντας την επαναληψιμότητα και τη διαφάνεια των αλλαγών.
  - Υποστήριξη για πολλαπλά περιβάλλοντα (π.χ., ανάπτυξη, παραγωγή) με χρήση παραμετροποιημένων templates.
- **Παρακολούθηση και Ενημέρωση της Ασφάλειας (DevSecOps)**
  - Έλεγχοι ευπαθειών στην αλυσίδα εφοδιασμού λογισμικού κατά το build και την ανάπτυξη.
  - Αυτοματοποιημένοι έλεγχοι ασφαλείας για τον κώδικα, τα dependencies και τα container images μέσω εργαλείων όπως SonarQube, Trivy, ή Snyk.
  - Διαχείριση μυστικών (secrets management) μέσω εργαλείων όπως το HashiCorp Vault ή το AWS Secrets Manager.
- **Παρακολούθηση και Αντιμετώπιση Drift (GitOps)**
  - Σύγκριση της τρέχουσας κατάστασης των συστημάτων με το configuration στο Git και αυτοματοποιημένη αποκατάσταση αποκλίσεων.
  - Χρήση operators και εργαλείων όπως το ArgoCD ή Flux για την αυτοματοποιημένη συγχρονισμό της υποδομής με τον ορισμό της στο Git.
- **Παρακολούθηση και Καταγραφή Συμβάντων (Observability)**
  - Ενσωμάτωση παρακολούθησης με εργαλεία όπως το Prometheus, Grafana, και ELK/EFK stack.
  - Ανάλυση και ειδοποίηση για ασυνήθιστες δραστηριότητες ή ανωμαλίες μέσω λογισμικού όπως το Zabbix ή το Datadog.

2. Μη Λειτουργικές Απαιτήσεις

- **Απόδοση και Κλιμάκωση**
  - Η υποδομή πρέπει να είναι σε θέση να κλιμακώνεται δυναμικά σύμφωνα με τη ζήτηση.
  - Χρήση containerization για την παροχή ευελιξίας και ταχύτητας στις αναπτύξεις.
- **Διαθεσιμότητα και Αξιοπιστία**
  - Διασφάλιση υψηλής διαθεσιμότητας με στρατηγικές όπως η αποκατάσταση μετά από αποτυχία (failover) και τα disaster recovery σχέδια.
  - Δυνατότητα συνεχούς παράδοσης (Continuous Delivery) και συνεχούς ενσωμάτωσης (Continuous Integration).
- **Ασφάλεια**

- Χρήση Role-Based Access Control (RBAC) και πολιτικών IAM για τον έλεγχο πρόσβασης.
- Αυτοματοποιημένες ενημερώσεις ασφαλείας και διαχείριση patches.
- Υποστήριξη για πολύ-παράγοντικη αυθεντικοποίηση (MFA) μέσω τεχνολογιών όπως YubiKey.

3. Απαιτήσεις Συμβατότητας

- **Υποστήριξη Πλατφορμών και Τεχνολογιών**
  - Συμβατότητα με public cloud providers (Azure, AWS, GCP) και on-premise εγκαταστάσεις.
  - Δυνατότητα υποστήριξης hybrid cloud περιβαλλόντων.
- **Συμβατότητα με Υπάρχουσες Υπηρεσίες και Εργαλεία**
  - Ενσωμάτωση με υπηρεσίες παρακολούθησης όπως Zabbix και Grafana.
  - Χρήση του υπάρχοντος συστήματος διαχείρισης ταυτότητας (π.χ., Active Directory για ενσωμάτωση AD).

4. Απαιτήσεις Απόδοσης

- **Χρόνος Ανάπτυξης και Ανάκτησης**
  - Οι αναπτύξεις να γίνονται εντός 5 λεπτών για μη κρίσιμες ενημερώσεις.
  - Αυτόματη επαναφορά (rollback) σε περίπτωση αποτυχίας.
- **Χρόνος Ανίχνευσης Ευπαθειών**
  - Τα pipelines να ολοκληρώνουν τους ελέγχους ασφαλείας εντός 10 λεπτών.

5. Απαιτήσεις Διακυβέρνησης και Κανονιστικών Ρυθμίσεων

- **Συμμόρφωση με Κανονισμούς**
  - Διασφάλιση ότι η υποδομή και οι διαδικασίες συμμορφώνονται με κανονισμούς όπως GDPR, HIPAA, ή ISO 27001.
- **Καταγραφή και Αναφορές**
  - Διατήρηση αρχείων καταγραφής για όλες τις αλλαγές στην υποδομή και στους κώδικες ανάπτυξης.

4.5. Προτεινόμενη Λύση

Η προτεινόμενη λύση επικεντρώνεται στην **αυτοματοποίηση** και την **διασφάλιση ποιότητας** κατά την ανάπτυξη λογισμικού και υποδομών, αξιοποιώντας τεχνολογίες όπως **GitOps**, **Kubernetes**, και **Terraform**, καθώς και σύγχρονες μεθοδολογίες **DevOps** και **DevSecOps**.

Η υλοποίηση ξεκινά με τη δημιουργία αποθετηρίων (repos) στο **GitHub** ή **GitLab**, όπου αποθηκεύονται οι ορισμοί για **CI/CD** (Continuous Integration/Continuous Delivery) που καλύπτουν τη δοκιμή, κατασκευή και προετοιμασία για ανάπτυξη εφαρμογών. Η προσέγγιση **Infrastructure as Code (IaC)** με το **Terraform** επιτρέπει τη δημιουργία και διαχείριση υποδομών ως κώδικα, καθιστώντας την αναπαραγώγιμη και εύκολα διαχειρίσιμη. Ειδικά, το Terraform χρησιμοποιείται για τη δημιουργία ενός **Kubernetes cluster**, το οποίο περιλαμβάνει **Persistent Volumes (PV)** για τη διαχείριση δεδομένων.

Για την ανάπτυξη των εφαρμογών, περιλαμβάνονται **Dockerfiles** για τη δημιουργία container images, τα οποία προωθούνται σε ένα **container registry** που προσφέρει η πλατφόρμα. Με kubernetes manifests, δημιουργείται ένα δίκτυο **Hyperledger Fabric (HLF)** στο cluster, εγκαθίσταται chaincode στο HLF και μια εφαρμογή proxy εκτίθεται στο δίκτυο. Για τη **παρατηρησιμότητα** (observability), γίνεται εισαγωγη κώδικα και containers στα pods για συλλογή **tracing**, **metrics** και **logs**.

Η διαχείριση των υποδομών και των εφαρμογών γίνεται με χρήση μεθοδολογιας **GitOps**. Όλες οι αλλαγές υποδομής και υπηρεσιών γίνονται μέσω **commits** και **merge requests** στο Git, διασφαλίζοντας διαφάνεια και συνέπεια. Για την ανάλυση εξαρτήσεων και την ενίσχυση της ασφάλειας, διατηρούμε ένα **Software Bill of Materials (SBOM)** με εργαλεία όπως το **CycloneDX** ή το **SPDX**.

Στην **αυτοματοποίηση** της ανάπτυξης και διαχείρισης, το **CI/CD** επιτυγχάνεται με εργαλεία όπως το **ArgoCD**, το **Flux** ή το **Jenkins**, τα οποία προσανατολίζονται στο Kubernetes και συνεργάζονται αρμονικά μεσω της μεθοδολογίας GitOps, προσφέροντας βελτιωμένη αποδοτικότητα και ασφάλεια σε κάθε στάδιο της ανάπτυξης και λειτουργίας.

Επιπλέον, χρησιμοποιούμε τεχνικές ανάλυσης και δοκιμών όπως:

SCA (Software Composition Analysis) για την ανάλυση εξαρτήσεων κωδικα στις βιβλιοθήκες ανοικτού κώδικα.

SAST (Static Application Security Testing) και **DAST (Dynamic Application Security Testing)** για την ασφάλεια κώδικα.

Για την ανάλυση κώδικα Terraform, υπάρχουν εργαλεία lint και SAST που βοηθούν στη διασφάλιση της ποιότητας και της ασφάλειας του κώδικα. Μερικές από τις δημοφιλείς επιλογές είναι:

Linting Εργαλεία:

1. **TFLint**: Το TFLint είναι ένα εργαλείο linting για Terraform που βοηθά στην ανίχνευση κοινών σφαλμάτων στον κώδικα HCL (HashiCorp Configuration Language) και στην αποτροπή παραβιάσεων των βέλτιστων πρακτικών. Μπορεί επίσης να ενσωματωθεί με plugins για συγκεκριμένους cloud providers.
2. **Terraform Validate**: Ενσωματωμένο εργαλείο της Terraform που ελέγχει τη συντακτική ορθότητα του κώδικα και αν οι διαμορφώσεις είναι σύμφωνες με το σχέδιο εκτέλεσης. Δεν είναι ακριβώς lint, αλλά βοηθά στον εντοπισμό βασικών σφαλμάτων.
3. **Checkov**: Αν και κύρια χρησιμοποιείται ως SAST εργαλείο, το Checkov προσφέρει δυνατότητες linting για την Terraform, εξετάζοντας κανόνες ασφαλείας και συμμόρφωσης στον κώδικα.

SAST Εργαλεία:

1. **Checkov**: Ένα εργαλείο ανοιχτού κώδικα που ανιχνεύει προβλήματα ασφάλειας στην υποδομή ως κώδικα (IaC). Υποστηρίζει Terraform, CloudFormation, Kubernetes, και άλλες διαμορφώσεις.
2. **tfsec**: Εργαλείο ανοιχτού κώδικα που αναλύει τον κώδικα Terraform για να βρει προβλήματα ασφαλείας και μη ασφαλείς ρυθμίσεις. Σαρώνει τα αρχεία HCL για κοινά σφάλματα ασφαλείας.
3. **Terrascan**: Παρέχει δυνατότητες στατικής ανάλυσης για Terraform και άλλα IaC εργαλεία, ανιχνεύοντας ζητήματα ασφαλείας και συμμόρφωσης. Υποστηρίζει πολλούς κανόνες για διαφορετικούς cloud providers.
4. **Snyk IaC**: Προσφέρει ανάλυση ασφάλειας για Terraform καθώς και για άλλες γλώσσες και διαμορφώσεις IaC. Ενσωματώνεται εύκολα σε CI/CD pipelines.

Συγκεκριμένα, ελέγχουμε:

- Την ορθότητα και τη μορφοποίηση των Terraform αρχείων με terraform validate και terraform fmt.
- Τη συμμόρφωση των Kubernetes manifests με εργαλεία όπως kubeval ή kube-score.
- Τις δηλώσεις παρατηρησιμότητας με κατάλληλα εργαλεία ανάλογα με το πλαίσιο εφαρμογής.
- Τον κώδικα επιχειρηματικής λογικής με frameworks δοκιμών, π.χ. Jest για TypeScript/JavaScript ή το testing πακέτο της Go.

- Για το SAST, μπορούμε να ενσωματώσουμε εργαλεία όπως το SonarQube ή το Checkmarx στο CI pipeline, ενώ για το DAST χρησιμοποιούμε εργαλεία όπως το OWASP ZAP, που αυτοματοποιούν τις δοκιμές μετά την ανάπτυξη.

Με αυτή την ολοκληρωμένη προσέγγιση, επιδιώκουμε τη βελτιστοποίηση της διαδικασίας ανάπτυξης, διασφαλίζοντας την ασφάλεια, τη συνέπεια και την ποιότητα των εφαρμογών και υποδομών.

Η προτεινόμενη λύση περιλαμβάνει:

1. **CI/CD Pipelines:**
   - Αυτοματοποιημένα pipelines για τη δοκιμή, κατασκευή και ανάπτυξη του κώδικα.
   - Workflows βασισμένα σε GitOps για συνεχή παράδοση.
2. **Terraform Code για Υποδομή:**
   - Ορισμοί για τη δημιουργία και διαχείριση AKS resources.
   - Διαμόρφωση Kubernetes deployments & services.
   - Διαχείριση εξωτερικών υπηρεσιών όπως Sentry.io, Grafana, Docker Container Registry.
3. **Κώδικας Παρατηρησιμότητας (Observability) για το Λογισμικό:**
   - Σενάρια συλλογής ιχνών (tracing), μετρήσεων (metrics) και καταγραφών (logs).
   - Ενσωμάτωση εργαλείων παρακολούθησης για καλύτερη διάγνωση και επίλυση προβλημάτων.

Αξιοποιώντας το Kubernetes και το Terraform, μπορούμε να δούμε την εφαρμογή τους σε δράση μέσα στα CI/CD pipelines μας. Αυτό όχι μόνο καταδεικνύει την πρακτική τους χρησιμότητα, αλλά επίσης αναδεικνύει την ομαλή ενσωμάτωση της διαχείρισης υποδομών με την ανάπτυξη λογισμικού.

Ας δούμε τα κομμάτια που συνθέτουν την λύση:

Terraform Code

Πάροχος Azure (provider "azurerm"): Δηλώνει τον πάροχο Azure για το Terraform, επιτρέποντας τη διαχείριση υποδομών στο Azure.

Resource Group (azurerm_resource_group): Δημιουργεί μια ομάδα πόρων (Resource Group) στο Azure με την ονομασία "aks-acr-resources" στην τοποθεσία "East US".

Azure Container Registry (azurerm_container_registry): Δημιουργεί ένα Container Registry για την αποθήκευση Docker εικόνων, με admin-enabled πρόσβαση.

Azure Kubernetes Service (azurerm_kubernetes_cluster): Δημιουργεί ένα Kubernetes cluster στο Azure με ένα node pool από δύο κόμβους (τύπος VM: Standard_DS2_v2) και υποστηρίζει SystemAssigned ταυτότητα και το azure network plugin.

Παροχή Kubernetes και Helm Providers: Διαμορφώνει τον πάροχο Kubernetes και Helm για τη σύνδεση στο AKS cluster, επιτρέποντας τη διαχείριση των πόρων Kubernetes και των εγκαταστάσεων μέσω Helm.

Εγκατάσταση ArgoCD με Helm (helm_release): Εγκαθιστά το ArgoCD στο namespace "argocd" μέσω Helm chart, ορίζοντας το service τύπου LoadBalancer.

Εγκατάσταση kube-prometheus-stack με Helm: Εγκαθιστά το Helm chart "kube-prometheus-stack" στο namespace "monitoring" για παρακολούθηση και διαχείριση παρατηρησιμότητας.

```
1. provider "azurerm" {
2.   features {}
3. }
4.
5. # Resource Group
6. resource "azurerm_resource_group" "rg" {
```

```
 7.   name     = "aks-acr-resources"
 8.   location = "East US"
 9. }
10.
11. # Azure Container Registry
12. resource "azurerm_container_registry" "acr" {
13.   name                = "exampleacr"
14.   resource_group_name = azurerm_resource_group.rg.name
15.   location            = azurerm_resource_group.rg.location
16.   sku                 = "Basic"
17.   admin_enabled       = true
18. }
19.
20. # Azure Kubernetes Service
21. resource "azurerm_kubernetes_cluster" "aks" {
22.   name                = "exampleaks"
23.   location            = azurerm_resource_group.rg.location
24.   resource_group_name = azurerm_resource_group.rg.name
25.   dns_prefix          = "exampleaks"
26.
27.   default_node_pool {
28.     name       = "default"
29.     node_count = 2
30.     vm_size    = "Standard_DS2_v2"
31.   }
32.
33.   identity {
34.     type = "SystemAssigned"
35.   }
36.
37.   network_profile {
38.     network_plugin = "azure"
39.   }
40. }
41.
42. # Add kubeconfig output for convenience
43. output "kube_config" {
44.   value = azurerm_kubernetes_cluster.aks.kube_config_raw
45.   sensitive = true
46. }
47.
48. # Kubernetes Provider
49. provider "kubernetes" {
50.   host                   = azurerm_kubernetes_cluster.aks.kube_config.0.host
51.                          client_certificate                                    =
base64decode(azurerm_kubernetes_cluster.aks.kube_config.0.client_certificate)
52.              client_key                                                        =
base64decode(azurerm_kubernetes_cluster.aks.kube_config.0.client_key)
53.                                                 cluster_ca_certificate         =
base64decode(azurerm_kubernetes_cluster.aks.kube_config.0.cluster_ca_certificate)
54. }
55.
56. # Helm Provider
57. provider "helm" {
58.   kubernetes {
59.     host                   = azurerm_kubernetes_cluster.aks.kube_config.0.host
60.                                  client_certificate                            =
base64decode(azurerm_kubernetes_cluster.aks.kube_config.0.client_certificate)
61.                       client_key                                               =
base64decode(azurerm_kubernetes_cluster.aks.kube_config.0.client_key)
62.                                                       cluster_ca_certificate   =
base64decode(azurerm_kubernetes_cluster.aks.kube_config.0.cluster_ca_certificate)
```

```
  }
}

# Install ArgoCD with Helm
resource "helm_release" "argocd" {
  name       = "argocd"
  repository = "https://argoproj.github.io/argo-helm"
  chart      = "argo-cd"
  namespace  = "argocd"

  create_namespace = true

  values = [
    "server.service.type=LoadBalancer"
  ]
}

# Install kube-prometheus-stack with Helm
resource "helm_release" "kube_prometheus_stack" {
  name       = "kube-prometheus-stack"
  repository = "https://prometheus-community.github.io/helm-charts"
  chart      = "kube-prometheus-stack"
  namespace  = "monitoring"

  create_namespace = true

  values = [
    "grafana.adminPassword=admin"
  ]
}
```

CI/CD Pipelines

Code Pipeline: Αυτοματοποιεί τα στάδια ανάπτυξης κώδικα, από το setup και τις δοκιμές μέχρι το build, το security, και την ανάπτυξη. Περιλαμβάνει σενάρια για Python, TypeScript, και Solidity, καθώς και εργαλεία SAST, DAST, και SCA για ασφάλεια.

Infrastructure Pipeline: Διαχειρίζεται το Infrastructure as Code (IaC) μέσω Terraform, περιλαμβάνοντας στάδια validation, security (με το εργαλείο tfsec), plan, και apply.

Docker Pipeline: Δημιουργεί Docker images και τα ανεβάζει σε container registry, ακολουθώντας μια διαδικασία build και push.

Service Pipeline: Εφαρμόζει GitOps για την ανάπτυξη Kubernetes manifests και τον συγχρονισμό της εφαρμογής με το ArgoCD.

Policies Pipeline: Χρησιμοποιεί το Open Policy Agent (OPA) για την αξιολόγηση πολιτικών και ανιχνεύει τυχόν αποκλίσεις.

Observability Pipeline: Εισάγει εργαλεία παρατηρησιμότητας, όπως το Jaeger, για την παρακολούθηση ιχνών σε υπηρεσίες Kubernetes.

Code pipeline

```
stages:
  - setup
  - test
  - build
  - security
  - deploy

variables:
  PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache/pip"
  NPM_CACHE_DIR: "$CI_PROJECT_DIR/.cache/npm"
```

```
11.
12. cache:
13.   key: "$CI_COMMIT_REF_SLUG"
14.   paths:
15.     - .cache/
16.     - node_modules/
17.     - venv/
18.
19. before_script:
20.   - python -m venv venv
21.   - source venv/bin/activate
22.   - pip install --upgrade pip
23.
24. setup:
25.   stage: setup
26.   script:
27.     - pip install -r requirements.txt
28.     - npm install
29.
30. lint_python:
31.   stage: test
32.   script:
33.     - source venv/bin/activate
34.     - flake8 app/
35.
36. lint_typescript:
37.   stage: test
38.   script:
39.     - npm run lint
40.
41. test_python:
42.   stage: test
43.   script:
44.     - source venv/bin/activate
45.     - pytest tests/
46.
47. test_typescript:
48.   stage: test
49.   script:
50.     - npm run test
51.
52. test_solidity:
53.   stage: test
54.   image: ethereum/solc:stable
55.   script:
56.     - npm run test:solidity
57.
58. build_python:
59.   stage: build
60.   script:
61.     - source venv/bin/activate
62.     - pyinstaller --onefile app/main.py
63.
64. build_typescript:
65.   stage: build
66.   script:
67.     - npm run build
68.
69. sast:
70.   stage: security
71.   script:
72.     - echo "Running SAST..."
```

```
    - pip install bandit
    - bandit -r app/
    - npm install -g @nodesecurity/nsp
    - nsp check

dast:
  stage: security
  script:
    - echo "Running DAST..."
    - docker run --rm -v $PWD:/zap/wrk/:rw -t owasp/zap2docker-stable zap-baseline.py -t https://example.com

sca:
  stage: security
  script:
    - echo "Running SCA..."
    - pip install safety
    - safety check

deploy:
  stage: deploy
  script:
    - echo "Deploy step here"
  environment:
    name: production
    url: https://example.com
```

Infrastructure pipeline

Αυτό το pipeline θα διαχειρίζεται το IaC (Terraform) και θα δημιουργεί ένα σύστημα Kubernetes.

```
stages:
  - validate
  - security
  - plan
  - apply

variables:
  TF_IN_AUTOMATION: "true"

terraform_validate:
  stage: validate
  script:
    - terraform init
    - terraform validate

terraform_security:
  stage: security
  script:
    # Χρησιμοποιεί το εργαλείο tfsec για έλεγχο ασφάλειας
    - tfsec .

terraform_plan:
  stage: plan
  script:
    - terraform init
    - terraform plan -out=plan.tfplan

terraform_apply:
  stage: apply
  script:
```

```
31.     - terraform apply plan.tfplan
32.   when: manual
33.   only:
34.     - main
35.
```

Artifacts pipeline

Αυτό το pipeline θα δημιουργεί Docker εικόνες από Dockerfiles και θα τις ανεβάζει στο container registry.

```
 1. stages:
 2.   - build
 3.   - push
 4.
 5. variables:
 6.   IMAGE_NAME: my-app
 7.
 8. docker_build:
 9.   stage: build
10.   script:
11.     - docker build --pull -t $IMAGE_NAME:$CI_COMMIT_SHA .
12.     - docker tag $IMAGE_NAME:$CI_COMMIT_SHA $IMAGE_NAME:latest
13.
14. docker_push:
15.   stage: push
16.   script:
17.     - echo $DOCKER_PASSWORD | docker login -u $DOCKER_USER --password-stdin
18.     - docker push $IMAGE_NAME:$CI_COMMIT_SHA
19.     - docker push $IMAGE_NAME:latest
20.   only:
21.     - main
22.
```

Service pipeline

Αυτό το pipeline αναπτύσσει τους ορισμούς Kubernetes (συμπεριλαμβανομένου του δικτύου HLF) και την εφαρμογή χρησιμοποιώντας GitOps.

```
 1. stages:
 2.   - deploy
 3.
 4. variables:
 5.   ARGOCD_SERVER: argocd.example.com
 6.
 7. argo_deploy:
 8.   stage: deploy
 9.   script:
10.     # Σύνδεση με το ArgoCD
11.     - argocd login $ARGOCD_SERVER --username $ARGOCD_USER --password $ARGOCD_PASSWORD --insecure
12.     # Εφαρμογή των Kubernetes manifests (εφόσον απαιτείται)
13.     - kubectl apply -f k8s_manifests/
14.     # Συγχρονισμός της εφαρμογής με ArgoCD
15.     - argocd app sync my-app
16.     # Επιβεβαίωση ότι η εφαρμογή είναι σε κατάσταση HEALTHY
17.     - argocd app wait my-app --health --timeout 300
18.   only:
19.     - main
20.
```

Policies pipeline

Υποθέτοντας τη χρήση OPA για τους ελέγχους πολιτικών:

```
 1. stages:
```

```
  - policy_check

variables:
  POLICY_FILE: policy.rego
  INPUT_FILE: input.json

policy_check:
  stage: policy_check
  script:
    # Αξιολόγηση πολιτικών χρησιμοποιώντας το OPA
      - opa eval --data $POLICY_FILE --input $INPUT_FILE "data.example.violation" > opa_results.json
    # Καταγραφή αποτελεσμάτων για ανάλυση
    - cat opa_results.json
    # Έλεγχος αν υπάρχει παραβίαση
    - |
      violations=$(jq '.result[0].expressions[0].value | length' opa_results.json)
      if [ "$violations" -gt 0 ]; then
        echo "Policy violations found, failing the build."
        exit 1
      fi
```

Observability pipeline

Για την εισαγωγή εργαλείων παρακολούθησης στις υπηρεσίες σας:

```
stages:
  - observe

variables:
  JAEGER_NAMESPACE: observability

observe_trace:
  stage: observe
  script:
    # Δημιουργία ή εφαρμογή του namespace για Jaeger
    - kubectl create namespace $JAEGER_NAMESPACE || echo "Namespace already exists"
    # Εφαρμογή των Kubernetes manifests για Jaeger
    - kubectl apply -f jaeger_k8s_manifests/ -n $JAEGER_NAMESPACE
    # Έλεγχος της κατάστασης των deployments
    - kubectl rollout status deployment/jaeger -n $JAEGER_NAMESPACE
    # Έλεγχος των pods για να βεβαιωθεί ότι τρέχουν σωστά
    - kubectl get pods -n $JAEGER_NAMESPACE
  only:
    - main
```

Παρακολούθηση Traces: Διαμορφώνει το Jaeger για συλλογή και προβολή ιχνών (traces), με διασφάλιση ότι οι υπηρεσίες παρατηρησιμότητας είναι σωστά εγκατεστημένες και σε λειτουργία.

Σενάρια Εφαρμογής Kubernetes Manifests για Εργαλεία Παρακολούθησης: Χρησιμοποιούνται Kubernetes manifests για την ανάπτυξη και διαμόρφωση εργαλείων παρατηρησιμότητας όπως το Jaeger.

Rollout Checks και Pod Status Monitoring: Βεβαιώνει ότι οι υπηρεσίες παρατηρησιμότητας αναπτύσσονται σωστά και τα pods λειτουργούν κανονικά.

Ενσωμάτωση SCA, SAST, DAST και Άλλων Αναλύσεων Ασφάλειας και Παρατηρησιμότητας σε CI/CD

Η χρήση των τεχνικών SCA, SAST, και DAST είναι ουσιαστικής σημασίας για την ενίσχυση της ασφάλειας στις διαδικασίες ανάπτυξης λογισμικού. Η ενσωμάτωση αυτών των εργαλείων στο pipeline CI/CD εξασφαλίζει ότι οι εφαρμογές ελέγχονται σε διαφορετικά στάδια για ευπάθειες και κινδύνους ασφαλείας.

SCA, SAST, DAST

- **SCA (Software Composition Analysis)**: Αναφέρεται στην ανάλυση της σύνθεσης του λογισμικού για τον εντοπισμό ευπαθειών σε εξαρτήσεις ανοιχτού κώδικα. Εργαλεία όπως το Sonatype Nexus, το Snyk ή το GitLab Dependency Scanning μπορούν να ενσωματωθούν στο CI pipeline για να βοηθήσουν στην αξιολόγηση της ασφάλειας των βιβλιοθηκών και των εξαρτημάτων τρίτων που χρησιμοποιούνται.
- **SAST (Static Application Security Testing)**: Το SAST περιλαμβάνει την ανάλυση του πηγαίου κώδικα για τον εντοπισμό ευπαθειών ασφαλείας πριν από την εκτέλεση του λογισμικού. Εργαλεία όπως το SonarQube ή το Checkmarx μπορούν να ενσωματωθούν στο CI pipeline για την ανάλυση εφαρμογών βασισμένων σε γλώσσες προγραμματισμού όπως Python, JavaScript και Solidity, παρέχοντας λεπτομερείς αναφορές για πιθανές αδυναμίες στον κώδικα.
- **DAST (Dynamic Application Security Testing)**: Το DAST πραγματοποιεί έλεγχο της εφαρμογής κατά την εκτέλεσή της σε περιβάλλον staging ή παραγωγής. Εργαλεία όπως το OWASP ZAP μπορούν να χρησιμοποιηθούν για τη διενέργεια αυτοματοποιημένων δοκιμών ασφάλειας μετά την ανάπτυξη, επιτρέποντας τον εντοπισμό ζητημάτων ασφαλείας που μπορεί να εμφανίζονται λόγω της αλληλεπίδρασης της εφαρμογής με εξωτερικά συστήματα.

Ενσωμάτωση Εργαλείων Ασφαλείας στα CI/CD Pipelines

Η ενσωμάτωση εργαλείων ασφαλείας στα CI/CD pipelines εξασφαλίζει συνεχείς ελέγχους ασφαλείας καθ' όλη τη διάρκεια του κύκλου ανάπτυξης λογισμικού. Αυτή η προληπτική προσέγγιση βοηθά στον εντοπισμό και την αντιμετώπιση ευπαθειών από τα πρώτα στάδια της ανάπτυξης, διατηρώντας υψηλά πρότυπα ασφάλειας χωρίς να διαταράσσει τη ροή ανάπτυξης. Η επιλογή των εργαλείων γίνεται με γνώμονα συγκεκριμένες απαιτήσεις ασφαλείας και τη φύση της εφαρμογής.

1. **Εργαλεία Στατικού Ελέγχου Ασφαλείας Κώδικα (SAST)**, όπως το **SonarQube** και το **Checkmarx**, ενσωματώνονται νωρίς (shift-left) στο pipeline για να σαρώσουν τον κώδικα και να εντοπίσουν ευπάθειες. Αυτά τα εργαλεία επιλέχθηκαν γιατί παρέχουν εκτενή ανάλυση για διάφορες γλώσσες προγραμματισμού και ενσωματώνονται εύκολα στο υπάρχον CI/CD περιβάλλον, προσφέροντας αναλυτική ανατροφοδότηση απευθείας στους προγραμματιστές. Η δυνατότητά τους να εντοπίζουν κενά ασφαλείας και μη ασφαλή μοτίβα πριν από την κατασκευή της εφαρμογής μειώνει σημαντικά τους πιθανούς κινδύνους.
2. **Εργαλεία Ελέγχου Εξαρτήσεων**, όπως το **OWASP Dependency-Check** και το **Snyk**, χρησιμοποιούνται για τον εντοπισμό ευπαθειών σε βιβλιοθήκες τρίτων. Επειδή οι εφαρμογές μας βασίζονται σε μεγάλο βαθμό σε ανοιχτού κώδικα εξαρτήσεις, αυτά τα εργαλεία διασφαλίζουν ότι δεν θα μεταφερθούν γνωστές ευπάθειες στην παραγωγή. Το **Snyk** προτιμήθηκε λόγω της πλούσιας βάσης δεδομένων του και των δυνατοτήτων ενσωμάτωσης με το GitHub, παρέχοντας αυτόματους ελέγχους pull requests και προτάσεις διόρθωσης.
3. **Εργαλεία Δυναμικού Ελέγχου Ασφαλείας (DAST)**, όπως το **OWASP ZAP** και το **Burp Suite**, χρησιμοποιούνται για τη σάρωση αναπτυγμένων εφαρμογών σε περιβάλλοντα staging. Αυτά τα εργαλεία προσομοιώνουν επιθέσεις για να εντοπίσουν πιθανούς κινδύνους σε εφαρμογές που λειτουργούν. Το **OWASP ZAP** επιλέχθηκε για τον ανοιχτού κώδικα χαρακτήρα του και την ευκολία ενσωμάτωσης σε αυτοματοποιημένους ελέγχους, ενώ το **Burp Suite** προσφέρει προηγμένες δυνατότητες για χειροκίνητους ελέγχους από τις ομάδες ασφαλείας κατά τη διάρκεια πιο ενδελεχών αξιολογήσεων.
4. **Ασφάλεια Containers** εξασφαλίζεται με εργαλεία όπως το **Aqua Security** και το **Trivy**, που σαρώνουν Docker images για ευπάθειες και λανθασμένες ρυθμίσεις. Δεδομένης της χρήσης περιβαλλόντων container στις υπηρεσιες μας, η ασφάλεια των container images πριν την ανάπτυξη είναι κρίσιμη. Το **Trivy** επιλέχθηκε για την ταχύτητά του και την υποστήριξη σάρωσης τόσο των εικόνων όσο και των αρχείων ρυθμίσεων (π.χ. Dockerfiles και Kubernetes manifests), ενώ το **Aqua Security** προσφέρει πιο ολοκληρωμένες πολιτικές και

δυνατότητες προστασίας κατά τη λειτουργία, καθιστώντας το κατάλληλο για περιβάλλοντα υψηλής ασφάλειας.

5. **Ασφάλεια Υποδομής ως Κώδικα (IaC)** υλοποιείται με τη χρήση του **Checkov**. Καθώς η υποδομή μας ορίζεται ως κώδικας με το Terraform, είναι απαραίτητο να εντοπίζονται λανθασμένες ρυθμίσεις και να εφαρμόζονται πολιτικές συμμόρφωσης πριν την ανάπτυξη. Το **Checkov** επιλέχθηκε για την ικανότητά του να σαρώνει τον κώδικα του Terraform για ζητήματα ασφαλείας και να επιβάλλει οργανωτικές πολιτικές, εξασφαλίζοντας ότι τα cloud περιβάλλοντά μας συμμορφώνονται με τις βέλτιστες πρακτικές ασφαλείας.

Η ενσωμάτωση αυτών των εργαλείων στο CI/CD pipeline επιτρέπει την αυτοματοποίηση των ελέγχων ασφαλείας σε διάφορα στάδια, διασφαλίζοντας ότι η ασφάλεια ενσωματώνεται στην ανάπτυξη από την αρχή. Αυτή η προσέγγιση μειώνει τους κινδύνους εντοπίζοντας ευπάθειες νωρίς, ελαχιστοποιεί την προσπάθεια για διόρθωση και ευθυγραμμίζει τις πρακτικές ασφαλείας με την ταχύτητα και την ευελιξία της σύγχρονης ανάπτυξης λογισμικού.

SBOM (Software Bill of Materials)

Το **SBOM** είναι ένας λεπτομερής κατάλογος των στοιχείων που συνθέτουν ένα λογισμικό, συμπεριλαμβανομένων των βιβλιοθηκών και των εξαρτημάτων τρίτων. Η χρήση εργαλείων όπως το CycloneDX ή το SPDX βοηθά στη δημιουργία και διαχείριση του SBOM, παρέχοντας διαφάνεια και τη δυνατότητα γρήγορης ανίχνευσης ευπαθειών σε εξαρτήματα τρίτων.

Αξιολόγηση ArgoCD, Flux, Jenkins(-X)

Τα **ArgoCD** και **Flux** είναι αυτοματοποιημένα εργαλεία ανάπτυξης προσανατολισμένα για περιβάλλοντα Kubernetes, ακολουθώντας τη μεθοδολογία **GitOps**. Επιτρέπουν τον συγχρονισμό μεταξύ των κώδικα στο Git και της υποδομής, διευκολύνοντας την αυτόματη ανάπτυξη και τον συνεχή έλεγχο των αλλαγών. Το **Jenkins-X** παρέχει παρόμοιες δυνατότητες CI/CD, αλλά απαιτεί περισσότερη παραμετροποίηση σε σχέση με το ArgoCD ή το Flux.

Έλεγχοι και Αναλύσεις

- **Terraform**: Ο έλεγχος των υποδομών μέσω των terraform validate και terraform fmt είναι σημαντικός για να εξασφαλιστεί η σωστή σύνταξη και μορφοποίηση των αρχείων υποδομής.
- **Kubernetes Manifests**: Η χρήση εργαλείων όπως το **kubeval** ή το **kube-score** για τον έλεγχο των Kubernetes manifests είναι απαραίτητη για να διασφαλιστεί ότι ακολουθούνται οι καλύτερες πρακτικές.
- **Observability Declarations**: Η παρατηρησιμότητα εξαρτάται από τα εργαλεία που χρησιμοποιούνται, όπως **Prometheus/Grafana** ή **ELK Stack**, και περιλαμβάνει τον έλεγχο των metrics και logs.
- **Κώδικας Επιχειρηματικής Λογικής**: Για τον κώδικα της επιχειρηματικής λογικής, προτείνονται frameworks ελέγχου όπως το **Jest** για TypeScript/JavaScript και η ενσωματωμένη βιβλιοθήκη testing του Go για κώδικα γραμμένο σε Go.

GitOps Μεθοδολογία

Η αποθήκευση όλων των ορισμών υποδομής και υπηρεσιών ως κώδικας στο αποθετήριο Git αποτελεί βασικό στοιχείο της μεθοδολογίας **GitOps**. Με αυτήν τη μεθοδολογία, κάθε αλλαγή στην υποδομή ή τις υπηρεσίες ξεκινά με ένα commit ή merge request στο Git. Εργαλεία όπως το ArgoCD βοηθούν στον συγχρονισμό αυτών των αλλαγών με το πραγματικό περιβάλλον παραγωγής.

Συμπεράσματα

Η ενσωμάτωση των SCA, SAST, και DAST σε ένα pipeline CI/CD προσφέρει σημαντική προστασία σε ό,τι αφορά την ασφάλεια και την ακεραιότητα του κώδικα και των υποδομών. Παράλληλα, η δημιουργία ενός SBOM και η χρήση εργαλείων GitOps ενισχύουν τη διαφάνεια, την ασφάλεια, και τη συνεχή διαχείριση των αλλαγών, καθιστώντας τις διαδικασίες ανάπτυξης και παραγωγής πιο αξιόπιστες και ασφαλείς.

4.6. Παραδοχές

1. Υιοθέτηση Cloud-Native Τεχνολογιών

- **Παραδοχή**: Η υποδομή θα χρησιμοποιεί cloud-native τεχνολογίες, όπως Kubernetes και containerization. Αυτό σημαίνει ότι η πλατφόρμα θα υποστηρίζει την ταχεία ανάπτυξη και κλιμάκωση υπηρεσιών με χρήση containers και microservices.
- **Ανάλυση**: Η παραδοχή αυτή δικαιολογείται λόγω της αυξημένης χρήσης Kubernetes και containers για DevOps, αλλά μπορεί να μην ισχύει σε περιβάλλοντα που χρησιμοποιούν legacy συστήματα ή μονολιθικές αρχιτεκτονικές.

2. Χρήση Σύγχρονων Εργαλείων Αυτοματισμού (CI/CD)

- **Παραδοχή**: Υπάρχει η δυνατότητα χρήσης εργαλείων για συνεχή ενσωμάτωση και παράδοση (CI/CD), όπως GitLab CI, Jenkins, ή GitHub Actions, και αυτά τα εργαλεία θα ενσωματωθούν πλήρως στις διαδικασίες ανάπτυξης.
- **Ανάλυση**: Προϋποθέτει ότι η ομάδα ανάπτυξης έχει την εμπειρία ή την εκπαίδευση για να χειριστεί αυτά τα εργαλεία, και ότι υπάρχει επαρκής υποστήριξη για την εφαρμογή αυτών των πρακτικών.

3. Κεντρική Διαχείριση της Υποδομής με IaC

- **Παραδοχή**: Η διαχείριση της υποδομής θα γίνεται εξ ολοκλήρου μέσω Infrastructure as Code, με εργαλεία όπως Terraform, Ansible, ή CloudFormation.
- **Ανάλυση**: Η παραδοχή αυτή προϋποθέτει ότι η υποδομή μπορεί να οριστεί πλήρως μέσω κώδικα και ότι τα μέλη της ομάδας έχουν εμπειρία στη χρήση IaC εργαλείων. Σε ορισμένες περιπτώσεις, μπορεί να υπάρξουν τμήματα υποδομής που δεν είναι δυνατόν να αυτοματοποιηθούν πλήρως.

4. Υποστήριξη Υβριδικού ή Multi-Cloud Περιβάλλοντος

- **Παραδοχή**: Η υπηρεσία θα είναι σε θέση να διαχειρίζεται υποδομές που βρίσκονται σε πολλαπλά cloud ή σε υβριδικό περιβάλλον (συνδυασμός cloud και on-premises).
- **Ανάλυση**: Αυτό μπορεί να μην είναι απαραίτητο σε εταιρείες που χρησιμοποιούν αποκλειστικά έναν πάροχο cloud. Επιπλέον, η υποστήριξη υβριδικού περιβάλλοντος αυξάνει την πολυπλοκότητα και την ανάγκη για διαχείριση διαφορετικών πολιτικών ασφαλείας.

5. Η Ασφάλεια Ενσωματώνεται Πλήρως στον Κύκλο Ζωής Ανάπτυξης Λογισμικού

- **Παραδοχή**: Η ασφάλεια δεν είναι μια ξεχωριστή φάση αλλά ενσωματώνεται σε κάθε στάδιο της ανάπτυξης, από τον σχεδιασμό μέχρι την ανάπτυξη και την υποστήριξη.
- **Ανάλυση**: Η παραδοχή αυτή υποθέτει ότι υπάρχει κουλτούρα και κατανόηση στην ομάδα σχετικά με τη σημασία της ασφάλειας σε κάθε στάδιο, κάτι που μπορεί να μην ισχύει αν δεν έχει γίνει η κατάλληλη εκπαίδευση.

6. Πλήρης Παρακολούθηση και Εποπτεία της Υποδομής (Observability)

- **Παραδοχή**: Η υποδομή θα παρέχει πλήρη παρακολούθηση και παρατηρησιμότητα, με συλλογή metrics, logs, και tracing.
- **Ανάλυση**: Απαιτεί την ύπαρξη και τη συντήρηση εργαλείων παρακολούθησης και καταγραφής. Σε μικρότερες υποδομές ή σε περιπτώσεις με περιορισμένα resources, αυτό μπορεί να μην είναι πρακτικό.

7. Ευελιξία και Συνεχής Ενημέρωση της Υποδομής

- **Παραδοχή**: Η υποδομή θα πρέπει να είναι δυναμική και να μπορεί να αλλάζει συχνά για να υποστηρίζει νέες εκδόσεις κώδικα ή νέες τεχνολογίες.
- **Ανάλυση**: Η συνεχής ενημέρωση μπορεί να είναι δύσκολη σε περιβάλλοντα με αυστηρές κανονιστικές απαιτήσεις ή όταν υπάρχει legacy λογισμικό.

8. Ομαδοποίηση και Αυτοματοποίηση των Διαδικασιών Ανάπτυξης

- **Παραδοχή**: Όλες οι διαδικασίες ανάπτυξης και συντήρησης μπορούν να αυτοματοποιηθούν, από το build και το testing μέχρι την ανάπτυξη και τη διαχείριση της υποδομής.
- **Ανάλυση**: Παρόλο που η αυτοματοποίηση είναι ένας επιθυμητός στόχος, δεν είναι πάντα εφικτό να αυτοματοποιηθούν όλες οι διαδικασίες, ειδικά όταν υπάρχουν πολύπλοκες ή μη προβλέψιμες διαδικασίες.

Οι παραδοχές αυτές βοηθούν στη διαμόρφωση των απαιτήσεων, αλλά πρέπει να αξιολογηθούν για να διαπιστωθεί αν ευθυγραμμίζονται με την υπάρχουσα κατάσταση και τις πραγματικές ανάγκες της επιχείρησης ή της ομάδας ανάπτυξης.

4.7. Ανάλυση Επιλογών και Σύνθεση της Λύσης

Για την υλοποίηση μιας σύγχρονης, αυτοματοποιημένης και ασφαλούς υποδομής, αναλύθηκαν διάφορες επιλογές σε κρίσιμα συστατικά, και η τελική λύση συνδυάζει τις καλύτερες πρακτικές με στόχο τη μέγιστη ευελιξία και προσαρμοστικότητα στις επιχειρηματικές απαιτήσεις.

1. Code Repositories

Επιλογές:

- **GitHub, GitLab, Bitbucket**: Όλα προσφέρουν εξελιγμένες λειτουργίες ελέγχου εκδόσεων, διαχείριση pull/merge requests, και δυνατότητες για ενσωμάτωση CI/CD pipelines.
- **Self-hosted Git Servers (π.χ., Gitea)**: Κατάλληλο για εταιρείες που απαιτούν μεγαλύτερο έλεγχο και απομόνωση των αποθετηρίων κώδικα.

Σύνθεση της Λύσης:
Επιλέχθηκε το **GitHub/GitLab** για την υποστήριξη cloud-based λύσεων και την παροχή ενσωματωμένων CI/CD δυνατοτήτων. Η λύση επιτρέπει την υιοθέτηση πρακτικών **GitOps**, όπου κάθε αλλαγή στην υποδομή και τον κώδικα πραγματοποιείται μέσω **pull requests**, διασφαλίζοντας ότι κάθε αλλαγή αξιολογείται και εγκρίνεται πριν την ενσωμάτωση στην παραγωγή.

2. Infrastructure

Επιλογές:

- **Cloud Providers (AWS, Azure, GCP)**: Προσφέρουν εκτενή υποστήριξη υπηρεσιών και αυτοματισμού με εργαλεία **IaC** όπως το **Terraform** και το **CloudFormation (AWS)**.
- **On-Premises με Kubernetes (MicroK8s, OpenShift, Rancher)**: Ιδανική λύση για περιπτώσεις που απαιτείται διαχείριση φυσικών διακομιστών και ιδιωτικών cloud.
- **Hybrid Cloud**: Συνδυάζει τις δυνατότητες cloud και on-premises για την υποστήριξη διαφορετικών αναγκών (π.χ., ευαίσθητα δεδομένα on-premise και ευέλικτες υπηρεσίες στο cloud).

Σύνθεση της Λύσης:
Υιοθετήθηκε μια **hybrid cloud στρατηγική**, με κύριες υποδομές στο **Azure** (που ήδη χρησιμοποιείται), ενώ τα περιβάλλοντα ανάπτυξης διαχειρίζονται μέσω **MicroK8s on-premises** για μεγαλύτερο έλεγχο. Ο αυτοματοποιημένος ορισμός και διαχείριση της υποδομής επιτυγχάνεται με χρήση **Infrastructure as Code (Terraform/Ansible)**.

3. Third-Party Services

Επιλογές:

- **Authentication και Authorization**: Το **Keycloak** επιλέγεται για τη διαχείριση ταυτότητας και ενσωμάτωση με **Active Directory** για **Single Sign-On (SSO)**.

- **Secrets Management**: Εργαλεία όπως το **HashiCorp Vault**, το **AWS Secrets Manager** ή το **Sealed Secrets** στο **Kubernetes**.
- **Log Management**: Χρήση του **ELK/EFK Stack** ή εργαλείων όπως το **Splunk** για παρακολούθηση και ανάλυση αρχείων καταγραφής.

Σύνθεση της Λύσης:
Για τον έλεγχο ταυτότητας, επιλέγεται το **Keycloak** για **SSO** και **YubiKey MFA** για αυξημένη ασφάλεια. Η διαχείριση των μυστικών γίνεται με το **HashiCorp Vault**, ενώ το **EFK Stack** χρησιμοποιείται για τη συλλογή και διαχείριση των logs.

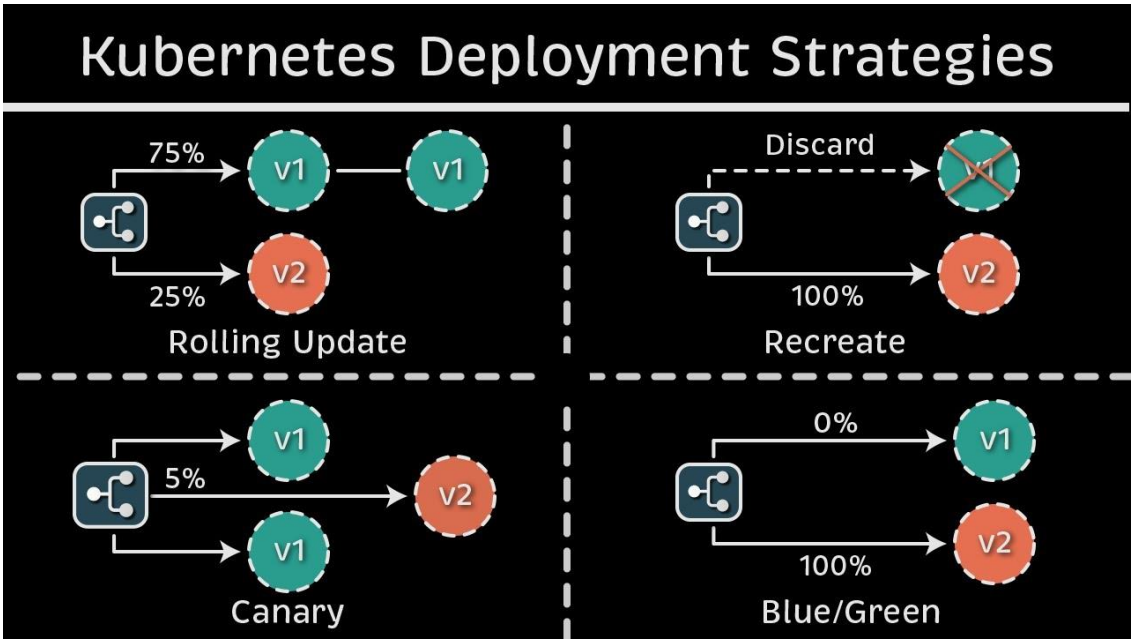
4. CI/CD Pipelines

Επιλογές:

- **Jenkins, GitLab CI, GitHub Actions, CircleCI**: Παρέχουν διάφορες δυνατότητες για αυτόματο **build**, **testing**, και **deployment**.
- **ArgoCD, Flux**: Υποστηρίζουν **GitOps deployment** και παρακολούθηση της κατάστασης των Kubernetes clusters.

Σύνθεση της Λύσης:
Επιλέχθηκε το **GitHub Actions** ή το **GitLab CI** για την αυτοματοποίηση των **CI/CD pipelines**, ενσωματώνοντας **αυτόματους ελέγχους ασφάλειας**. Το **ArgoCD** χρησιμοποιείται για **GitOps deployment** στο **Kubernetes**, παρακολουθώντας το **drift** και αυτοματοποιώντας την αποκατάσταση.

6. **Deployment Strategies**



Εικόνα 9 Στρατηγικές παράδοσης κώδικα

Επιλογές:

- **Blue-Green Deployment**: Επιτρέπει τη συντήρηση δύο περιβαλλόντων (παλιού και νέου), διευκολύνοντας την αναίρεση αλλαγών.
- **Canary Releases**: Σταδιακή εφαρμογή νέων εκδόσεων σε υποσύνολο χρηστών για δοκιμή.
- **Rolling Updates**: Αντικατάσταση των instances με νέες εκδόσεις σταδιακά.

Στην Εικόνα 9 βλέπουμε σύγκριση των διάφορων στρατηγικών παράδοσης κώδικα.

Σύνθεση της Λύσης:
Υιοθετήθηκε η χρήση των **Canary releases** για τον προοδευτικό έλεγχο νέων εκδόσεων, ενώ τα **Rolling updates** επιλέγονται για μικρότερες αλλαγές, εξασφαλίζοντας την απρόσκοπτη λειτουργία.

6. Monitoring

Επιλογές:

- **Prometheus και Grafana**: Για συλλογή μετρικών και δημιουργία dashboards.
- **Zabbix, Datadog, New Relic**: Προσφέρουν ολοκληρωμένη παρακολούθηση και ειδοποιήσεις.
- **Jaeger ή Zipkin**: Για **distributed tracing**.

Σύνθεση της Λύσης:
Επιλέχθηκε το **Prometheus** για τη συλλογή μετρικών και το **Grafana** για την οπτικοποίηση και ειδοποίηση. Για το **distributed tracing** σε περιβάλλοντα με **microservices**, χρησιμοποιείται το **Jaeger**, ενώ το **Zabbix** αξιοποιείται για την παρακολούθηση συστημάτων εκτός **Kubernetes**.

Η τελική σύνθεση της λύσης εξασφαλίζει την ομαλή υλοποίηση και διαχείριση μιας υποδομής υψηλής απόδοσης, διασφαλίζοντας παράλληλα την ευελιξία, την ασφάλεια, και την προσαρμοστικότητα στις εξελισσόμενες ανάγκες.

4.8. Ανάλυση Αποτελεσμάτων

Η ενότητα αυτή επικεντρώνεται στην ανάλυση των αποτελεσμάτων των δοκιμών και την αξιολόγηση των επιμέρους χαρακτηριστικών των συστημάτων που μελετήθηκαν, συμπεριλαμβανομένης της ασφάλειας, της αξιοπιστίας και του κόστους υποδομής. Αξιοποιούμε διάφορες τεχνικές και εργαλεία για να επιβεβαιώσουμε την απόδοση και τη συμμόρφωση των συστημάτων με τις βέλτιστες πρακτικές, με απώτερο στόχο τη βελτίωση της αποτελεσματικότητας και της ανθεκτικότητάς τους. Η παρακάτω ανάλυση παρουσιάζει τις μεθόδους και τα εργαλεία που χρησιμοποιήθηκαν για stress testing, ανάλυση ασφαλείας, benchmarking blockchain, καθώς και την παρακολούθηση και βελτιστοποίηση του κόστους υποδομής.

4.9. Επιμέρους Χαρακτηριστικά Που Θα Αξιολογηθούν (Ασφάλεια, Αξιοπιστία, Κόστος Υποδομής)

Σε αυτή την ενότητα, θα παρουσιάσουμε τα επιμέρους χαρακτηριστικά που αξιολογούνται σε μια σύγχρονη υποδομή, εστιάζοντας σε παράγοντες όπως η **ασφάλεια**, η **αξιοπιστία** και το **κόστος**. Με την ολοένα αυξανόμενη πολυπλοκότητα των συστημάτων και την ενσωμάτωση καινοτόμων τεχνολογιών, η αξιολόγηση αυτών των χαρακτηριστικών αποτελεί κρίσιμη προϋπόθεση για την αποτελεσματική διαχείριση των πόρων και την εγγύηση της απόδοσης και της ασφάλειας των υπηρεσιών. Κατά την ανάλυση αυτή, εξετάζουμε τη χρήση τεχνικών όπως το **stress testing** για την αντοχή του συστήματος, την αξιολόγηση της ασφάλειας μέσω **στατικών και δυναμικών ελέγχων κώδικα** (SAST/DAST), και τη **βελτιστοποίηση κόστους** με εργαλεία που βοηθούν στη διαχείριση πόρων με οικονομικό αποδοτικό τρόπο. Μέσα από αυτές τις τεχνικές, εξασφαλίζουμε την ποιότητα και τη βιωσιμότητα των συστημάτων, διατηρώντας παράλληλα υψηλά πρότυπα απόδοσης και ασφαλείας.

4.9.1. Stress Testing

Το Stress Testing είναι μια τεχνική δοκιμής που χρησιμοποιείται για να προσδιορίσουμε τον τρόπο με τον οποίο ένα σύστημα θα ανταποκριθεί υπό εξαιρετικά εντατικές συνθήκες φόρτου, κάτι που είναι ιδιαίτερα σημαντικό για την αξιολόγηση της αντοχής και της αξιοπιστίας των

συστημάτων σε πραγματικές καταστάσεις υψηλού φορτίου. Οι δοκιμές αυτές μας βοηθούν να εντοπίσουμε πιθανά σημεία αποτυχίας και να βελτιώσουμε τη συνολική απόδοση και ανθεκτικότητα των συστημάτων μας. Παρακάτω παρατίθενται δύο από τα δημοφιλέστερα εργαλεία για stress testing:

Locust

Γενικά: Το Locust είναι ένα ανοιχτού κώδικα εργαλείο stress testing γραμμένο σε Python. Είναι γνωστό για την ευελιξία του και τη δυνατότητά του να δημιουργεί σενάρια δοκιμών με χρήση κώδικα Python.

Χαρακτηριστικά: Επεκτάσιμο με προσθήκη βιβλιοθηκών Python, κατανεμημένες δοκιμές με πολλαπλούς πελάτες, παρακολούθηση σε πραγματικό χρόνο μέσω web UI.

Χρήση: Κατάλληλο για εφαρμογές και συστήματα που απαιτούν προσαρμοσμένα σενάρια δοκιμών με δυνατότητα κώδικα.

JMeter

Γενικά: Το Apache JMeter είναι επίσης ένα εργαλείο ανοιχτού κώδικα για δοκιμές φόρτου και μέτρησης απόδοσης. Δημιουργήθηκε αρχικά για δοκιμές web εφαρμογών, αλλά έχει διευρυνθεί για να υποστηρίξει διάφορους τύπους δοκιμών.

Χαρακτηριστικά: Γραφικό περιβάλλον για σχεδίαση σεναρίων δοκιμών, υποστηρίζει πολλούς τύπους αιτημάτων (π.χ. HTTP, JDBC, FTP), πλούσια οπτική ανατροφοδότηση κατά τη διάρκεια των δοκιμών.

Χρήση: Κατάλληλο για εταιρείες και ομάδες που απαιτούν ένα πιο ολοκληρωμένο γραφικό περιβάλλον και υποστηρίζει πολλούς τύπους πρωτοκόλλων και εφαρμογών.

Και τα δύο εργαλεία προσφέρουν υψηλό επίπεδο ευελιξίας και επεκτασιμότητας, αλλά η επιλογή ανάμεσά τους εξαρτάται συχνά από τις ανάγκες της εφαρμογής, τις προτιμήσεις των χρηστών και τις δεξιότητες της ομάδας που τα χρησιμοποιεί. Για παράδειγμα, εάν απαιτούνται πιο εξειδικευμένα σενάρια δοκιμών, το Locust μπορεί να είναι η καλύτερη επιλογή, ενώ το JMeter είναι ιδανικό για πιο ολοκληρωμένες γραφικές δοκιμές.

SAST/DAST/SCA Findings

Infrastructure: Αναλύουμε τον κώδικα υποδομών (π.χ., Terraform, Kubernetes manifests) για να διασφαλίσουμε ότι συμμορφώνεται με τις βέλτιστες πρακτικές ασφαλείας.

Code: Χρησιμοποιούμε SAST και DAST εργαλεία όπως το SonarQube ή το Checkmarx για ανάλυση κώδικα και OWASP ZAP για τη δοκιμή της εφαρμογής σε πραγματικό περιβάλλον.

Policies: Ελέγχουμε αν οι πολιτικές ασφαλείας και οι κανόνες διαμόρφωσης είναι σωστά ενσωματωμένοι στο σύστημα για να διασφαλίσουμε τη συμμόρφωση.

Blockchain Benchmarking - Caliper

Το Caliper είναι ένα εργαλείο από το Hyperledger που χρησιμοποιείται για την αξιολόγηση και την αναφορά της απόδοσης των blockchain. Με το Caliper, μπορούμε να μετρήσουμε διάφορες μετρήσεις απόδοσης για διάφορα blockchain συστήματα. Ας εξετάσουμε το Caliper πιο αναλυτικά:

Γενικό Πλαίσιο: Το Caliper επιτρέπει στους χρήστες να ορίζουν και να εκτελούν δοκιμαστικά σενάρια που αποτελούνται από ένα συγκεκριμένο σύνολο των blockchain εργασιών. Αυτές οι εργασίες μπορεί να περιλαμβάνουν εγγραφές, αναγνώσεις ή άλλες λειτουργίες, ανάλογα με την εφαρμογή.

Πλατφόρμες που Υποστηρίζονται: Ενώ το Caliper ξεκίνησε κυρίως για το Hyperledger Fabric, πλέον υποστηρίζει πολλές άλλες πλατφόρμες blockchain, όπως το Hyperledger Besu, το Ethereum, το FISCO BCOS και άλλα.

Μετρήσεις Απόδοσης: Με το Caliper μπορούμε να αποκτήσουμε μετρήσεις όπως τα TPS (transactions per second), τον χρόνο καθυστέρησης (latency), και τους πόρους CPU/μνήμης που χρησιμοποιούνται. Αυτές οι μετρήσεις μας βοηθούν να αξιολογήσουμε την απόδοση του blockchain και να εντοπίσουμε τυχόν σημεία βελτίωσης.

Ευελιξία: Μια από τις βασικές δυνατότητες του Caliper είναι η ικανότητά του να εξατομικεύει τα σενάρια δοκιμής με βάση τις ανάγκες του χρήστη. Για παράδειγμα, ένας χρήστης μπορεί να δημιουργήσει σενάρια που προσομοιώνουν υψηλή κίνηση συναλλαγών ή συγκεκριμένες επιχειρησιακές διαδικασίες, προσαρμόζοντας έτσι τις δοκιμές στις πραγματικές απαιτήσεις του συστήματος. Αυτό επιτρέπει στους χρήστες να δημιουργούν πολύ συγκεκριμένες δοκιμές που μπορούν να αντικατοπτρίζουν την πραγματική χρήση του συστήματος.

Σε γενικές γραμμές, το Caliper προσφέρει μια αντικειμενική και συγκριτική εκτίμηση της απόδοσης διαφόρων blockchain συστημάτων, επιτρέποντας στους προγραμματιστές και τους ερευνητές να προσαρμόζουν και να βελτιώνουν τα συστήματά τους με βάση τα αποτελέσματα των δοκιμών. Επιπλέον, η δυνατότητα προσαρμογής των σεναρίων δοκιμής επιτρέπει την καλύτερη κατανόηση των δυνατοτήτων και των περιορισμών των blockchain τεχνολογιών υπό διαφορετικές συνθήκες.

4.9.2. Κόστος Υποδομής

Ένας σημαντικός παράγοντας που αξιολογούμε είναι το κόστος της υποδομής. Χρησιμοποιούμε εργαλεία παρακολούθησης κόστους, όπως το AWS Cost Explorer ή το Microsoft Cost Management, για να διασφαλίσουμε ότι η χρήση των πόρων είναι βέλτιστη και οικονομικά αποδοτική. Επιπλέον, εφαρμόζουμε μεθόδους βελτιστοποίησης, όπως η αυτοματοποίηση της κλιμακωσιμότητας ή η χρήση spot instances, προκειμένου να μειώσουμε το συνολικό κόστος, ενώ ταυτόχρονα διατηρούμε υψηλά επίπεδα απόδοσης και ασφάλειας. Για παράδειγμα, η χρήση auto-scaling policies διασφαλίζει ότι οι πόροι προσαρμόζονται στις ανάγκες της εφαρμογής, αποφεύγοντας έτσι την υπερβολική κατανάλωση.

Η χρήση spot instances προσφέρει μια οικονομικά αποδοτική λύση για εφαρμογές που δεν απαιτούν συνεχή διαθεσιμότητα, εξοικονομώντας σημαντικά ποσά στο συνολικό κόστος λειτουργίας. Η παρακολούθηση και η ανάλυση των τάσεων κόστους μάς επιτρέπει να προσαρμόζουμε τις πολιτικές χρήσης των πόρων ώστε να μεγιστοποιούμε την αποδοτικότητα, διατηρώντας παράλληλα την επιχειρησιακή συνέχεια και τη διαθεσιμότητα των υπηρεσιών.

Συνολικά, η συντονισμένη χρήση εργαλείων και τεχνικών βελτιστοποίησης κόστους μας επιτρέπει να διαχειριζόμαστε αποτελεσματικά την υποδομή, εξασφαλίζοντας παράλληλα την επίτευξη των στόχων απόδοσης και ασφάλειας, χωρίς να ξεπερνάμε τους οικονομικούς περιορισμούς που έχουν τεθεί.

5. Συμπεράσματα και Προτάσεις

Οι προσεγγίσεις DevSecOps, GitOps και Infrastructure as Code (IaC) προσφέρουν ουσιαστικά οφέλη στην ανάπτυξη, τη δοκιμή και την παράδοση λογισμικού, όπως η αυτοματοποίηση της διαχείρισης υποδομών μέσω εργαλείων όπως Terraform ή Ansible. Συγκεκριμένα, αυτές οι προσεγγίσεις ενισχύουν την αυτοματοποίηση, τη διαφάνεια και την ασφάλεια σε όλα τα στάδια του κύκλου ζωής του λογισμικού, επιτρέποντας την αποτελεσματική διαχείριση τόσο της υποδομής όσο και του κώδικα.

Η μεθοδολογία DevSecOps ενσωματώνει την ασφάλεια σε κάθε στάδιο του κύκλου ζωής του λογισμικού, καθιστώντας την μια αδιάλειπτη και ενσωματωμένη πρακτική. Αυτό διασφαλίζει ότι οι ευπάθειες και οι απειλές ανιχνεύονται και αντιμετωπίζονται πριν φτάσουν στο παραγωγικό περιβάλλον.

Αντίστοιχα η μεθοδολογία GitOps αξιοποιεί το Git ως την "πηγή αλήθειας" για την υποδομή και τις εφαρμογές. Μέσα από αυτήν τη μεθοδολογία, διασφαλίζεται η συνέπεια, η επαναληψιμότητα και η δυνατότητα ανάκτησης, ενώ παράλληλα επιτρέπει τη γρήγορη και αξιόπιστη ανάπτυξη.

Το Infrastructure as Code (IaC) μας επιτρέπει να διαχειριζόμαστε και να αναπτύσσουμε την υποδομή μέσω κώδικα, εξασφαλίζοντας την τυποποίηση και τη μείωση των σφαλμάτων κατά τη δημιουργία και την παραμετροποίηση πόρων. Μέσω των εργαλείων IaC, η δημιουργία υποδομών γίνεται ταχύτερα και με μειωμένο κίνδυνο ανθρώπινων σφαλμάτων.

Ωστόσο, η επιτυχής υλοποίηση αυτών των προσεγγίσεων απαιτεί συνεχή κατάρτιση του προσωπικού για την κατανόηση των νέων τεχνολογιών και πρακτικών, καθώς και επένδυση σε κατάλληλα εργαλεία, όπως Jenkins για CI/CD ή Terraform για IaC, για τη διαχείριση της υποδομής, της ασφάλειας και της παρατηρησιμότητας. Πέρα από την τεχνική διάσταση, η πραγματική επιτυχία εξαρτάται από μια δεσμευτική προσέγγιση για την αλλαγή της εταιρικής κουλτούρας, έτσι ώστε όλες οι ομάδες να υιοθετήσουν και να στηρίξουν τις νέες μεθόδους.

Η ενσωμάτωση των DevSecOps, GitOps, και IaC επιφέρει βελτίωση στην ποιότητα και στην ασφάλεια του παραγόμενου λογισμικού, μειώνοντας παράλληλα τον χρόνο κυκλοφορίας και τα σφάλματα κατά την ανάπτυξη. Αυτή η προσέγγιση, αν και απαιτεί σημαντικές αλλαγές και επενδύσεις, προσφέρει μακροπρόθεσμα οφέλη σε επίπεδο αποδοτικότητας, ευελιξίας και ασφάλειας.

5.1. Κύρια Συμπεράσματα Της Διατριβής

Η υιοθέτηση των προσεγγίσεων DevSecOps, GitOps, IaC και CI/CD αποτελεί έναν κρίσιμο παράγοντα για την επιτυχημένη ανάπτυξη και λειτουργία των σύγχρονων συστημάτων. Αυτές οι πρακτικές, όταν εφαρμόζονται συντονισμένα, συμβάλλουν στη βελτίωση της ασφάλειας, της απόδοσης και της συνολικής ποιότητας των υπηρεσιών. Επιπλέον, διευκολύνουν τη συνεργασία μεταξύ των ομάδων και μειώνουν τον χρόνο παράδοσης, επιτρέποντας στις οργανώσεις να ανταποκρίνονται γρήγορα στις ανάγκες των χρηστών και στις αλλαγές της αγοράς.

Η τεκμηρίωση και η αυτοματοποίηση των ελέγχων είναι κρίσιμες για τη διασφάλιση της ποιότητας και της αξιοπιστίας του συστήματος, ενώ η ενσωμάτωση της ασφάλειας από την αρχή του κύκλου ανάπτυξης συμβάλλει στη μείωση των ευπαθειών και στην ενίσχυση της προστασίας των δεδομένων.

Η συνεχής εκπαίδευση των ομάδων ανάπτυξης και λειτουργίας σε αυτές τις τεχνολογίες και πρακτικές είναι ζωτικής σημασίας για την επιτυχημένη υλοποίηση και τη διατήρηση της ποιότητας των υπηρεσιών. Με τη σωστή χρήση των μεθοδολογιών DevSecOps, GitOps, IaC και CI/CD, οι οργανώσεις μπορούν να επιτύχουν υψηλά επίπεδα απόδοσης, ασφάλειας και αξιοπιστίας, καθιστώντας τις υπηρεσίες τους ανθεκτικές και έτοιμες για τις προκλήσεις του μέλλοντος.

5.2. Μελλοντικές Επεκτάσεις

Παρά τα πλεονεκτήματα της προτεινόμενης λύσης, η επαύξηση της αξιοπιστίας μίας υπηρεσίας θα μπορούσε περαιτέρω να βελτιωθεί, μέσα από την εφαρμογή στοχευμένων επεκτάσεων, όπως είναι η εφαρμογή των τεχνολογιών k3s και P2P container image distribution βασισμένης στο σύστημα IPFS. Η αξιοπιστία είναι κρίσιμη για υπηρεσίες IoT και edge computing, καθώς αυτές λειτουργούν σε περιβάλλοντα με περιορισμένους πόρους και απομακρυσμένες τοποθεσίες, όπου η συνεχής λειτουργία και η αποφυγή διακοπών είναι απαραίτητες.

Η χρήση μιας ελαφριάς έκδοσης του Kubernetes, όπως το k3s, μπορεί να συμβάλει σημαντικά στη βελτίωση της αξιοπιστίας των υπηρεσιών. Σε συνδυασμό με την τεχνολογία P2P διανομής εικόνων container μέσω του IPFS, αυτό είναι ιδιαίτερα χρήσιμο σε περιβάλλοντα με περιορισμένους πόρους ή απομακρυσμένες τοποθεσίες, όπως είναι το IoT και το edge computing.

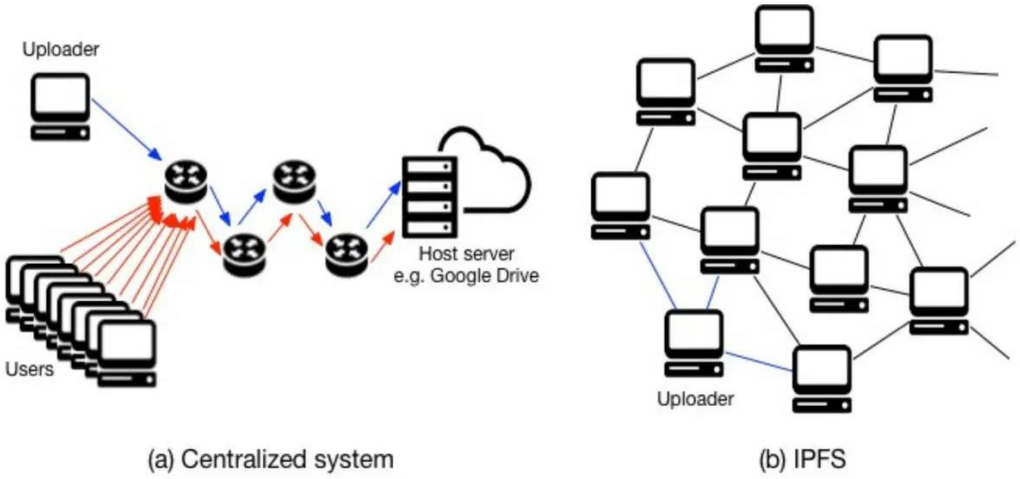
Στο μέλλον, θα ήταν χρήσιμο να εξεταστεί η περαιτέρω βελτίωση των αυτοματισμών και η χρήση προηγμένων εργαλείων παρατήρησης (observability) για την καλύτερη κατανόηση και διαχείριση των συστημάτων. Επίσης, η ενσωμάτωση τεχνολογιών όπως η τεχνητή νοημοσύνη και η μηχανική μάθηση μπορεί να προσφέρει επιπλέον δυνατότητες για την ανίχνευση προβλημάτων και τη βελτίωση της απόδοσης.

5.2.1. Εφαρμογή k3s

Ελάφρυνση αρχιτεκτονικής Το k3s είναι μια ελαφριά (lightweight) έκδοση του Kubernetes που έχει σχεδιαστεί για να λειτουργεί σε περιβάλλοντα με περιορισμένους πόρους. Αυτό το καθιστά ιδανικό για edge computing, IoT συσκευές και μικρές καταστάσεις. Η λειτουργία σε περιβάλλοντα με περιορισμένους πόρους μπορεί να βελτιώσει την αξιοπιστία, καθώς υπάρχουν λιγότερες εξαρτήσεις που μπορούν να παρουσιάσουν σφάλματα.

Απλοποίηση διαχείρισης: Επειδή το k3s έρχεται με αρκετά μέρη του Kubernetes αφαιρεμένα ή ενσωματωμένα, είναι πιο απλό στη διαχείριση, μειώνοντας τις πιθανότητες αστοχιών λόγω πολυπλοκότητας. Αυτό επιτρέπει μια πιο ευέλικτη και σταθερή υποδομή, η οποία είναι ιδιαίτερα χρήσιμη σε περιπτώσεις που χρειάζεται ταχύτατη ανάπτυξη και εύκολη διαχείριση.

5.2.2. P2P Container Image Distribution βασιζόμενη στο IPFS



Εικόνα 10 Πρωτόκολλο IPFS - Πηγή: https://symphony.is/about-us/blog/introd 1

Διαθεσιμότητα: το σύστημα κατανεμημένης αποθήκευσης IPFS (InterPlanetary File System) χρησιμοποιεί μια P2P μέθοδο για την αποθήκευση και τη λήψη δεδομένων. Αυτό σημαίνει ότι τα δεδομένα δεν εξαρτώνται από έναν κεντρικό διακομιστή, αλλά είναι διασκορπισμένα σε πολλούς κόμβους. Αν ένας κόμβος αποτύχει, τα δεδομένα εξακολουθούν να είναι διαθέσιμα από άλλους κόμβους, ενισχύοντας τη διαθεσιμότητα της υπηρεσίας.

Ανθεκτικότητα σε DDoS επιθέσεις: Εφόσον δεν υπάρχει κεντρικό σημείο αποτυχίας, η υπηρεσία είναι λιγότερο ευάλωτη σε DDoS επιθέσεις. Αυτό σημαίνει ότι η υποδομή μπορεί να παραμείνει λειτουργική ακόμα και σε περίπτωση μεγάλης κλίμακας επίθεσης.

Ταχεία Διανομή: Καθώς τα δεδομένα μπορούν να ληφθούν από τον κοντινότερο διαθέσιμο κόμβο, οι λήψεις μπορούν να είναι γρηγορότερες, ειδικά σε σενάρια μεγάλης κλίμακας. Αυτό είναι ιδιαίτερα χρήσιμο σε περιπτώσεις που υπάρχουν πολλαπλοί χρήστες ή συσκευές που χρειάζονται πρόσβαση στις εικόνες container ταυτόχρονα.
Στην Εικόνα 10 βλέπουμε πως λειτουργεί το κατανεμημένο σύστημα IPFS σε σύγκριση με ένα κεντρικοποιημένο σύστημα.

Συνολικά, τόσο το k3s όσο και μια P2P container image distribution βασισμένη στο IPFS προσφέρουν μηχανισμούς που μπορούν να επαυξήσουν την αξιοπιστία μιας υπηρεσίας μέσω της απλότητας, της διαθεσιμότητας, της ανθεκτικότητας και της γρήγορης διανομής. Αυτές οι τεχνολογίες μπορούν να βοηθήσουν στην αποτελεσματική διαχείριση των περιορισμένων πόρων και να εξασφαλίσουν ότι οι υπηρεσίες μπορούν να παραμείνουν διαθέσιμες και λειτουργικές ακόμη και σε δύσκολες συνθήκες, όπως σε περιβάλλοντα IoT ή απομακρυσμένα σημεία με περιορισμένες δυνατότητες δικτύου.

5.2.3. Άλλες Πιθανές Μελλοντικές Επεκτάσεις

Επιπλέον των παραπάνω, υπάρχουν και άλλες τεχνολογίες και πρακτικές που μπορούν να βελτιώσουν την αξιοπιστία και την ασφάλεια των υπηρεσιών. Παραδείγματα περιλαμβάνουν τη χρήση εργαλείων όπως το Prometheus για τη συλλογή μετρικών και το Grafana για την απεικόνιση αυτών, ώστε να υπάρχει συνεχής παρακολούθηση της κατάστασης των συστημάτων. Επίσης, η εφαρμογή της τεχνητής νοημοσύνης για την ανάλυση των δεδομένων παρακολούθησης μπορεί να βοηθήσει στην πρόβλεψη και πρόληψη πιθανών προβλημάτων.

Τέλος, η ενσωμάτωση πρακτικών Zero Trust Architecture μπορεί να ενισχύσει την ασφάλεια του συστήματος, διασφαλίζοντας ότι κάθε συσκευή και κάθε χρήστης πρέπει να αποδείξει την ταυτότητά του πριν αποκτήσει πρόσβαση στους πόρους, ακόμη και εντός του δικτύου.

Συνολικά, η συνεχής εξέλιξη των πρακτικών και τεχνολογιών για τη διαχείριση της υποδομής και των εφαρμογών είναι απαραίτητη για να διασφαλίζεται η ανθεκτικότητα, η αποδοτικότητα και η ασφάλεια των συστημάτων σε ένα συνεχώς μεταβαλλόμενο τεχνολογικό τοπίο.

Βιβλιογραφία

1. Argo Project. *ArgoCD documentation.* Ανάκτηση από https://argo-cd.readthedocs.io/en/stable/
2. Atlassian. Bitbucket. *Bitbucket*.
3. Azure. *Azure DevOps.* Ανάκτηση από https://azure.microsoft.com/en-us/services/devops/
4. Azure. *Azure Kubernetes Service (AKS).* Ανάκτηση από https://azure.microsoft.com/en-us/services/kubernetes-service/
5. Beyer, B., Jones, C., Petoff, N., & Murphy, N. (2016). *Site Reliability Engineering: How Google Runs Production Systems.* O'Reilly Media.
6. Center for Threat Reduction. (2021). *CTR Kubernetes Hardening Guidance 1.1.* Tech. rep., Department of Defense. Ανάκτηση από https://media.defense.gov/2021/Aug/03/2002820425/-1/-1/0/CTR_Kubernetes_Hardening_Guidance_1.1_20220315.PDF
7. Checkmarx. *Checkmarx documentation.* Ανάκτηση από https://checkmarx.com/
8. CircleCI. CircleCI Continuous Integration and Delivery. *CircleCI Continuous Integration and Delivery*.
9. CNCF. Harbor: Cloud Native Registry for Kubernetes. *Harbor: Cloud Native Registry for Kubernetes*.
10. Department of Defense. (2021). *DevSecOps Enterprise Container Hardening Guide 1.1.* Tech. rep., Department of Defense. Ανάκτηση από https://dl.dod.cyber.mil/wp-content/uploads/devsecops/pdf/Final_DevSecOps_Enterprise_Container_Hardening_Guide_1.1.pdf
11. Department of Defense. (2021). *DoD Enterprise DevSecOps Reference Design: CNCF Kubernetes.* Tech. rep., Department of Defense. Ανάκτηση από https://software.af.mil/wp-content/uploads/2021/05/DoD-Enterprise-DevSecOps-Reference-Design-v2.0-CNCF-Kubernetes.pdf
12. Elastic. ELK Stack: Elasticsearch, Logstash, Kibana. *ELK Stack: Elasticsearch, Logstash, Kibana*.
13. Elastic. *The ELK Stack: Elasticsearch, Logstash, & Kibana.* Ανάκτηση από https://www.elastic.co/what-is/elk-stack
14. Fotopoulos, F., Malamas, V., Dasaklis, T. K., Kotzanikolaou, P., & Douligeris, C. (2020, October). A blockchain-enabled architecture for IoMT device authentication. *2020 IEEE Eurasia Conference on IoT, Communication and Engineering (ECICE)* (σσ. 89-92). IEEE. doi:10.1109/ECICE50969.2020.9309995
15. GitHub, I. GitHub. *GitHub*.
16. GitLab. *GitLab CI/CD documentation.* Ανάκτηση από https://docs.gitlab.com/ee/ci/
17. GitLab. *What is DevOps?* Ανάκτηση από https://about.gitlab.com/topics/devops/
18. GmbH, T. C. Travis CI. *Travis CI*.
19. Google. Google Cloud Platform (GCP). *Google Cloud Platform (GCP)*.
20. Grafana Labs. *Grafana documentation.* Ανάκτηση από https://grafana.com/docs/
21. HashiCorp. *Terraform documentation.* Ανάκτηση από https://www.terraform.io/docs/index.html
22. HashiCorp. Terraform by HashiCorp. *Terraform by HashiCorp*.
23. HashiCorp. Vault - Secure, Store and Control Access to Tokens, Passwords, Certificates, and Encryption Keys. *Vault - Secure, Store and Control Access to Tokens, Passwords, Certificates, and Encryption Keys*.

24. Helm. Helm Kubernetes Package Manager. *Helm Kubernetes Package Manager*.
25. Helm. *Helm documentation.* Ανάκτηση από https://helm.sh/docs/
26. IEEE. *Application of DevSecOps within Infrastructure as Code.* Ανάκτηση από https://ieeexplore.ieee.org/
27. IEEE. *Systematic mapping study on the observability of cloud-native applications.* Ανάκτηση από https://ieeexplore.ieee.org/
28. InfluxData. InfluxDB - Time Series Database. *InfluxDB - Time Series Database*.
29. Jaeger. *Jaeger documentation.* Ανάκτηση από https://www.jaegertracing.io/docs/
30. Jenkins. Jenkins Automation Server. *Jenkins Automation Server*.
31. Jenkins, X. Jenkins X: CI/CD for Kubernetes. *Jenkins X: CI/CD for Kubernetes*.
32. Kubernetes. Kustomize - Kubernetes Native Configuration Management. *Kustomize - Kubernetes Native Configuration Management*.
33. Labs, G. Grafana. *Grafana*.
34. Labs, G. Loki: Like Prometheus, but for logs. *Loki: Like Prometheus, but for logs*.
35. Malamas, V., Kotzanikolaou, P., Dasaklis, T. K., & Burmester, M. (2020). A hierarchical multi blockchain for fine-grained access to medical data. *IEEE Access, 8*, 134393-134412. doi:10.1109/ACCESS.2020.3010503
36. Microsoft. (2023). *DevSecOps for Infrastructure as Code (IaC) - Azure Architecture Center.* Ανάκτηση από https://learn.microsoft.com/en-us/azure/architecture/solution-ideas/articles/devsecops-infrastructure-as-code
37. Mozilla. SOPS: Secrets OPerationS. *SOPS: Secrets OPerationS*.
38. National Institute of Standards and Technology. (2021). *Secure Software Development Framework (SSDF) Version 1.1.* Tech. rep., National Institute of Standards and Technology.
39. National Institute of Standards and Technology. (2023). *NIST Special Publication 800-204C: Implementation of DevSecOps for a Microservices-based Application with Service Mesh.* Tech. rep., National Institute of Standards and Technology. Ανάκτηση από https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204C.pdf
40. Nirmata. Kyverno - Kubernetes Native Policy Management. *Kyverno - Kubernetes Native Policy Management*.
41. Open Policy Agent. *OPA documentation.* Ανάκτηση από https://www.openpolicyagent.org/docs/latest/
42. Open Web Application Security Project. (2023). *OWASP DevSecOps Guideline.* Ανάκτηση από https://owasp.org/www-project-devsecops-guideline/
43. OWASP. (2023). *Infrastructure as Code (IaC) Security Cheat Sheet.* Ανάκτηση από https://cheatsheetseries.owasp.org/cheatsheets/Infrastructure_as_Code_Security_Cheat_Sheet.html
44. OWASP Foundation. *OWASP DevSecOps Guideline - v-0.2.* Ανάκτηση από https://owasp.org/www-project-devsecops-guideline/
45. OWASP Foundation. *OWASP DevSecOps Maturity Model (DSOMM).* Ανάκτηση από https://owasp.org/www-project-devsecops-guideline/
46. Prometheus. *Prometheus documentation.* Ανάκτηση από https://prometheus.io/docs/introduction/overview/
47. Pulumi. Pulumi - Modern Infrastructure as Code. *Pulumi - Modern Infrastructure as Code*.
48. Puppet. Puppet Enterprise. *Puppet Enterprise*.
49. Semaphore. (2023). *DevSecOps in Modern Software Development.* Ανάκτηση από https://semaphoreci.com/blog/devsecops

50. Sentry. *Sentry documentation.* Ανάκτηση από https://sentry.io/welcome/

51. Sentry. Sentry - Application Monitoring and Error Tracking. *Sentry - Application Monitoring and Error Tracking*.

52. Stubblefield, A., Blankinship, P., Lewandowski, P., Oprea, A., Adkins, H., & Beyer, B. (2020). *Building Secure and Reliable Systems: Best Practices for Designing, Implementing, and Maintaining Systems.* O'Reilly Media.

53. Sysdig. Falco - Cloud Native Runtime Security Project. *Falco - Cloud Native Runtime Security Project*.

54. Weaveworks. Flux - GitOps for Kubernetes. *Flux - GitOps for Kubernetes*.

55. The OWASP Foundation. (n.d.). *OWASP Zed Attack Proxy (ZAP) project*. Open Worldwide Application Security Project (OWASP). Retrieved November 18, 2024, from https://owasp.org/www-project-zap/

56. Hüttermann, M. (2012). *DevOps for developers*. Apress. https://doi.org/10.1007/978-1-4302-4570-4

57. Householder, A., Houser, T., Flynn, L., & Pohl, H. (2020). *DevSecOps: Defense in depth for software ecosystems*. CERT Division, Software Engineering Institute, Carnegie Mellon University.