ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΠΡΟΣΤΑΣΙΑ ΚΑΙ ΑΣΦΑΛΕΙΑ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Ανάπτυξη ενός DevSecOps αγωγού για εφαρμογές/υπηρεσίες βασιζόμενες σε δοχεία

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

Μαρία Α. Σκευοφύλακα

Επιβλέπων: Κυριάκος Κρητικός, Αναπληρωτής Καθηγητής

Μέλη εξεταστικής επιτροπής:
Κυριάκος Κρητικός, Αναπληρωτής Καθηγητής
Σπύρος Κοκολάκης, Καθηγητής
Γεώργιος Στεργιόπουλος, Επίκουρος Καθηγητής

Σάμος, Μάιος 2025

UNIVERSITY OF THE AEGEAN
POLYTECHNIC SCHOOL

DEPARTMENT OF INFORMATION AND COMMUNICATION SYSTEMS ENGINEERING

POSTGRADUATE STUDIES PROGRAM
SECURITY OF INFORMATION AND COMMUNICATION SYSTEMS

Development of a DevSecOps pipeline for container-based applications/services

MASTER THESIS

of

Maria A. Skevofylaka

Supervisor: Kyriakos Kritikos, Associate Professor

Members of the Examination Committee:

Kyriakos Kritikos, Associate Professor
Spyros Kokolakis, Professor
Georgios Stergiopoulos, Assistant Professor

Samos, May 2025

Πρόλογος και ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στο πλαίσιο του Προγράμματος Μεταπτυχιακών Σπουδών με τίτλο “Προστασία και Ασφάλεια Υπολογιστικών και Πληροφοριακών Συστημάτων” του Τμήματος Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων του Πανεπιστημίου Αιγαίου.

Θα ήθελα να ευχαριστήσω εκ βαθέων καρδιάς τους φίλους και την οικογένεια μου για την στήριξη τους καθ όλη την διάρκεια εκπόνησης της παρούσας εργασίας αλλά και κατά το διάστημα φοίτησης μου στο συγκεκριμένο μεταπτυχιακό πρόγραμμα.

Ακόμη, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Δρ. Κυριάκο Κρητικό, για την καθοδήγηση του και την εμπιστοσύνη που μου έδειξε, επιτρέποντας μου να αναλάβω το θέμα της παρούσας εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω τους συμφοιτητές μου, τους καθηγητές μου και την κοινότητα του Πανεπιστημίου Αιγαίου συνολικά για την στήριξη που έλαβα αλλά και για την άκρως εποικοδομητική, γεμάτη ερεθίσματα αλληλεπίδραση κατά τη διάρκεια των μεταπτυχιακών μου σπουδών.

Μαρία Σκευοφύλακα

Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων
Πανεπιστήμιο Αιγαίου

Πίνακας Περιεχομένων

Λίστα Εικόνων

Λίστα Πινάκων

Ακρώνυμα

Πίνακας 1. Ακρώνυμα

CD	Continuous Delivery
CDep	Continuous Deployment
CI	Continuous Integration
DAST	Dynamic Application Security Testing
DevOps	Development and Operations
DevSecOps	Development, Security and Operations
QA	Quality Assurance
SAST	Static Application Security Testing
SDLC	Secure Development Life-Cycle
VM	Virtual Machine

Περίληψη

Η παρούσα διπλωματική εργασία αφορά την ανάπτυξη ενός αγωγού DevSecOps που διευκολύνει την ενοποίηση και παράδοση δοχειοποιημένων εφαρμογών ιστού (containerized web applications) με έμφαση στην ασφάλεια, αξιοποιώντας το περιβάλλον GitHub Actions. Ο αγωγός ενσωματώνει λειτουργίες ασφάλειας, όπως στατική και δυναμική ανάλυση ασφάλειας, έλεγχος εξαρτήσεων, επιθεώρηση ασφαλείας αρχείων και περιεχομένου δοχείων, καθώς και ανίχνευση μη κρυπτογραφημένων μυστικών.

Ο αγωγός σχεδιάστηκε ώστε να είναι διαμορφώσιμος, επιτρέποντας στον χρήστη να ορίζει πότε η εκτέλεσή του θα αποτυγχάνει λόγω ευρημάτων ασφάλειας. Παράλληλα, η δυνατότητα προτυποποίησης του αγωγού διευκολύνει την επαναχρησιμοποίησή του σε διαφορετικά έργα.

Η εκτέλεσή του προτεινόμενου αγωγού σταματά, εφόσον εντοπιστούν σημαντικά ευρήματα ασφάλειας, πριν από τη διάταξη των δοχείων της εφαρμογής ιστού σε κάποιο περιβάλλον (πχ. παραγωγικό). Με αυτό τον τρόπο, παρέχεται μια ευέλικτη και αποτελεσματική λύση για την ενίσχυση της ασφάλειας και της ποιότητας στη διαδικασία ανάπτυξης εφαρμογών.

Λέξεις Κλειδιά: *Αγωγός, εφαρμογές ιστού, έλεγχος εξαρτήσεων, CI/CD, DevSecOps, δοχεία, ασφάλεια, στατική ανάλυση ασφάλειας, δυναμική ανάλυση ασφάλειας*

Abstract

This thesis focuses on the development of a DevSecOps pipeline that facilitates the integration and delivery of containerized web applications with a focus on security, utilizing the GitHub Actions environment. The pipeline incorporates security-oriented functionality, such as static and dynamic security analysis, dependency checks, security inspection of container files and contents, as well as the detection of unencrypted secrets.

This pipeline has been designed to be configurable, allowing users to define failure criteria based on the produced security findings. Additionally, the possibility of templating the pipeline enables its reuse across different projects.

The execution of the proposed pipeline stops, in case of the discovery of critical security findings, before the deployment of the containers of the web application to a respective environment (e.g., a production one). As such, it represents a flexible and efficient solution for enhancing the security and quality in the application development process.

Keywords: *Pipeline, web applications, dependency check, CI/CD, DevSecOps, containers, security, static security analysis, dynamic security analysis*

1 Εισαγωγή

1.1 Τι Είναι το DevOps

Το DevOps (Development and Operations) είναι μια μεθοδολογία που ενσωματώνει και συνδυάζει/ενοποιεί την ανάπτυξη λογισμικού με τη διαχείριση της λειτουργίας του, με στόχο τη βελτίωση της συνεργασίας και της αποδοτικότητας. Βασίζεται στη στενή επικοινωνία μεταξύ των διαφορετικών ομάδων (όπως ανάπτυξης και λειτουργίας), στην αυτοματοποίηση επαναλαμβανόμενων διαδικασιών και στη συνεχή παρακολούθηση για τη διασφάλιση της ποιότητας του παραγόμενου λογισμικού [20].

Η προσέγγιση αυτή μειώνει τους κινδύνους, βελτιώνει τη σταθερότητα των συστημάτων και επιταχύνει την ανάπτυξη και παράδοση του λογισμικού, καθιστώντας το DevOps κρίσιμο για τη σύγχρονη τεχνολογία και τη διαρκή βελτίωση των διαδικασιών ανάπτυξης λογισμικού.

1.2 Το DevSecOps Σήμερα

Το DevSecOps [21] αποτελεί τη σύγχρονη εξέλιξη του παραδοσιακού DevOps, ενσωματώνοντας την ασφάλεια ως αναπόσπαστο μέρος της διαδικασίας ανάπτυξης και λειτουργίας λογισμικού. Στον σημερινό ψηφιακό κόσμο, όπου οι επιθέσεις στον κυβερνοχώρο γίνονται ολοένα και πιο εξελιγμένες, η ανάγκη για ασφαλείς εφαρμογές από το στάδιο της ανάπτυξης έως την παραγωγή τους είναι πιο κρίσιμη από ποτέ.

Το DevSecOps προωθεί την κουλτούρα της συνεργασίας μεταξύ προγραμματιστών, επαγγελματιών ασφάλειας και διαχειριστών συστημάτων, με στόχο τη δημιουργία και τη διατήρηση ασφαλών συστημάτων και εφαρμογών. Περιλαμβάνει πρακτικές, όπως αυτοματοποιημένες δοκιμές ασφαλείας, συνεχή παρακολούθηση για ευπάθειες και ενσωμάτωση εργαλείων ασφάλειας στις διαδικασίες δοκιμής.

Με τη χρήση τεχνολογιών, όπως δοχειοποίησης (containerization), μικρουπηρεσιών (micro-services) και υπολογιστικού νέφους (cloud computing), το DevSecOps προσαρμόζεται στις αυξανόμενες απαιτήσεις για ταχύτητα και ευελιξία, χωρίς να θυσιάζει την ασφάλεια. Επιπλέον, η υιοθέτηση αρχών DevSecOps μειώνει τα κόστη αντιμετώπισης απειλών, ενισχύει την εμπιστοσύνη των χρηστών και βοηθά στη συμμόρφωση με κανονισμούς και πρότυπα (legislations and regulations), όπως ο Γενικός Κανονισμός Προστασίας Προσωπικών Δεδομένων (GDPR) [19] και το πρότυπο ασφάλειας ISO 27001 [18].

Σήμερα, το DevSecOps δεν θεωρείται μόνο καλή πρακτική αλλά και απαραίτητη στρατηγική για την επιτυχή υλοποίηση ασφαλών και αξιόπιστων εφαρμογών σε έναν συνεχώς μεταβαλλόμενο τεχνολογικό τοπίο. Ωστόσο, η εφαρμογή του μοντέλου αυτού συνοδεύεται από σημαντικές προκλήσεις, όπως η αντίσταση σε πολιτισμικές αλλαγές, η έλλειψη εξειδίκευσης στις

ομάδες ανάπτυξης και ασφαλείας, καθώς και η πολυπλοκότητα ενσωμάτωσης εργαλείων ασφαλείας στις υπάρχουσες DevOps διαδικασίες

Η παρούσα εργασία έρχεται να αντιμετωπίσει αυτά τα ζητήματα, εστιάζοντας στη δημιουργία ενός πλαισίου που διευκολύνει την ενσωμάτωση εργαλείων αυτοματοποιημένου ελέγχου ασφαλείας στους DevOps αγωγούς. Με αυτόν τον τρόπο, προτείνεται μια λύση που μειώνει τα εμπόδια υιοθέτησης του DevSecOps, ενισχύοντας τη συνεργασία μεταξύ ομάδων και τη διαρκή παρακολούθηση της ασφάλειας καθ' όλη τη διάρκεια του κύκλου ζωής ανάπτυξης λογισμικού

1.3 Αντικείμενο Διπλωματικής

Η παρούσα διπλωματική εργασία επικεντρώνεται στην ανάπτυξη ενός αγωγού (pipeline) CI/CD (Continuous Integration / Continuous Delivery) (Συνεχούς Ενοποίησης & Παράδοσης) για δοχειοποιημένες εφαρμογές (containerized applications), οι οποίες είναι γραμμένες σε Java χρησιμοποιώντας το εργαλείο διαχείρισης εξαρτήσεων και κατασκευής Maven και το πλαίσιο (framework) ανάπτυξης εφαρμογών ιστού (web applications) Spring Boot. Στόχος είναι η αντιμετώπιση προβλημάτων που σχετίζονται με την ασφάλεια, την αυτοματοποίηση και την αποδοτικότητα στη διαδικασία ανάπτυξης, δοκιμής, κατασκευής και διάθεσης τέτοιων εφαρμογών. Συχνά, ζητήματα, όπως η ανίχνευση ευπαθειών στις εικόνες δοχείων (container images), η διαχείριση μυστικών (secrets) και η συμμόρφωση με βέλτιστες πρακτικές ασφαλείας, παραμένουν ανεπαρκώς αντιμετωπισμένα, οδηγώντας σε κενά ασφαλείας και αυξημένη πολυπλοκότητα σε εφαρμογές που παραδίδονται στους πελάτες ή διατάσσονται σε παραγωγικά περιβάλλοντα.

Η προτεινόμενη λύση περιλαμβάνει τη δημιουργία ενός πλήρως αυτοματοποιημένου CI/CD αγωγού που ενσωματώνει εργαλεία στατικών και δυναμικών δοκιμών ασφαλείας (SAST/DAST), ανάλυσης ευπαθειών στις εικόνες δοχείων (container images), ανίχνευσης κακόβουλου λογισμικού και ανίχνευσης μυστικών. Ο αγωγός έχει σχεδιαστεί ώστε να καλύπτει τα στάδια υλοποίησης και επικύρωσης λογισμικού, του κύκλου ζωής της κατασκευής λογισμικού, διασφαλίζοντας ότι οι εφαρμογές είναι τόσο ασφαλείς όσο και έτοιμες για παραγωγή. Επιπλέον, ο αγωγός είναι επαναχρησιμοποιήσιμος, επιτρέποντας την εύκολη εφαρμογή του σε διαφορετικά έργα ή περιβάλλοντα, ενώ διαθέτει τη δυνατότητα προσδιορισμού των κριτηρίων αποτυχίας της εκτέλεσής του ανάλογα με τα ευρήματα ασφάλειας που εντοπίζονται, όπως η βαρύτητα των ευπαθειών. Μέσω της ενσωμάτωσης αυτών των τεχνολογιών και πρακτικών, η εργασία στοχεύει στην παροχή μιας ολοκληρωμένης λύσης που να συνδυάζει ασφάλεια, αποδοτικότητα και ευελιξία.

1.4 Δομή της Διπλωματικής

Η υπόλοιπη αναφορά της παρούσας διπλωματικής εργασίας είναι δομημένη ως εξής:

Κεφάλαιο 2 - Υπόβαθρο: Στο Κεφάλαιο 2 αναλύονται βασικές έννοιες, όπως το CI/CD και το DevSecOps, και η σχέση τους με την ασφαλή ανάπτυξη λογισμικού. Οι έννοιες αυτές είναι απαραίτητες για την κατανόηση του αγωγού που αναπτύχθηκε και προτείνεται στα πλαίσια της τρέχουσας εργασίας.

Κεφάλαιο 3- Ανάπτυξη: Στο σημείο αυτό, αναλύεται σχολαστικά το πως πραγματοποιήθηκε η

ανάπτυξη του συστήματος που υλοποιήθηκε για τις ανάγκες της παρούσας εργασίας.

Κεφάλαιο 4 - Εγκατάσταση και Επίδειξη: Στο Κεφάλαιο 4 αναλύεται ο τρόπος εγκατάστασης και χρήσης του συστήματος που υλοποιήθηκε. Στο ίδιο κεφάλαιο γίνεται και επίδειξη της λειτουργίας του σε μια ποικιλία σεναρίων χρήσης.

Κεφάλαιο 5 - Συμπεράσματα και Μελλοντική Εργασία: Σε αυτό το κεφάλαιο αναλύονται τα συμπεράσματα που προέκυψαν από την παρούσα εργασία και περιγράφονται πιθανές μελλοντικές επεκτάσεις της.

2 Υπόβαθρο

2.1 Επισκόπηση Κεφαλαίου

Το κεφάλαιο αυτό αποτελεί τη βιβλιογραφική ανασκόπηση της εργασίας και αποσκοπεί στην παρουσίαση των βασικών εννοιών, μεθοδολογιών και εργαλείων που σχετίζονται με τους αγωγούς συνεχούς ολοκλήρωσης και παράδοσης (CI/CD) στο πλαίσιο του DevOps. Αρχικά, γίνεται αναφορά στη θεωρητική βάση και στις κύριες τεχνολογίες που υποστηρίζουν τη συνεχή ολοκλήρωση, τη συνεχή παράδοση και τη δοχειοποίηση εφαρμογών. Στη συνέχεια, περιγράφονται τα στάδια ενός τυπικού αγωγού CI/CD, οι πρακτικές αυτοματοποίησης και τα οφέλη που προσφέρουν στην ανάπτυξη λογισμικού. Τέλος, προσδιορίζονται τα βασικά ζητήματα ασφάλειας των δοχειοποιημένων εφαρμογών (containerised applications) ώστε έπειτα να εισαχθεί και να αναλυθεί η έννοια του DevSecOps. Στόχος του κεφαλαίου είναι να προσφέρει μια σφαιρική και τεκμηριωμένη εικόνα του αντικειμένου, ώστε να υποστηριχθεί η κατανόηση και η τεκμηρίωση των επιλογών που ακολουθούν στα επόμενα μέρη της εργασίας.

2.2 Κύκλος Ανάπτυξης Εφαρμογών

Ο κύκλος ανάπτυξης εφαρμογών (SDLC) περιλαμβάνει τα εξής στάδια [16]:

1. Ανάλυση Απαιτήσεων (Requirements Analysis): Γίνεται εξαγωγή, ανάλυση και λεπτομερής καταγραφή των λειτουργικών και μη λειτουργικών απαιτήσεων της εφαρμογής.

2. Σχεδιασμός (Design): Στο στάδιο αυτό διαμορφώνεται η αρχιτεκτονική του συστήματος, η οποία προσδιορίζει τα κύρια συστατικά του και τον τρόπο που αυτά αλληλεπιδρούν (συνήθως μέσω διεπαφών). Παράλληλα, η σχεδίαση της εφαρμογής προχωρά μέσω της μοντελοποίησης σχεδιαστικών μοντέλων βάσει άξονες, όπως η δομή και η συμπεριφορά.

3. Υλοποίηση (Development): Στο στάδιο αυτό οι προγραμματιστές προχωρούν στην υλοποίηση της εφαρμογής και ειδικότερα στη συγγραφή του (πηγαίου) κώδικα βάσει των προδιαγραφών που ορίστηκαν στα προηγούμενα στάδια.

4. Validation & Verification (Επικύρωση & Επαλήθευση): Κατά τη διάρκεια του σταδίου αυτού πραγματοποιούνται δοκιμές όπως, δοκιμές μονάδων (unit tests), δοκιμές ενσωμάτωσης (integration tests), δοκιμές συστήματος, δοκιμές απόδοσης κ.α. προκειμένου να διασφαλιστεί πως η εφαρμογή λειτουργεί με τον αναμενόμενο τρόπο και ικανοποιεί τις απαιτήσεις των πελατών της. Ενδεχομένως να πραγματοποιηθούν και επιθεωρήσεις των τεχνημάτων που έχουν παραχθεί σε προηγούμενα στάδια (όπως σχεδιαστικά μοντέλα, το έγγραφο απαιτήσεων, κα.)

5. Διάταξη (Deployment): Πραγματοποιείται η εγκατάσταση και ενεργοποίηση της εφαρμογής σε περιβάλλον παραγωγής ή σε άλλο τελικό περιβάλλον, ώστε να είναι διαθέσιμη στους χρήστες.

6. Συντήρηση και Βελτίωση (Maintenance and Updates): Μετά την ολοκλήρωση της διάταξής της, η εφαρμογή συνεχίζει να ελέγχεται διαρκώς για τυχόν λάθη (bugs), ενημερώσεις ασφάλειας και βελτιώσεις που αφορούν την απόδοσή της. Στο σημείο αυτό ενδέχεται να προστεθούν νέες λειτουργίες καθώς και να ενημερωθούν υπάρχουσες με βάση προσθήκες ή τροποποιήσεις απαιτήσεων, αντίστοιχα, ενώ παράλληλα διασφαλίζεται η σταθερότητα του λογισμικού.

Η διαχείριση διαμορφώσεων (configuration management) αποτελεί βασική δραστηριότητα στο πλαίσιο του DevOps και του CI/CD, καθώς εξασφαλίζει τη συστηματική παρακολούθηση, τον έλεγχο και τη διατήρηση της συνέπειας των διαμορφώσεων σε όλα τα τεχνήματα (artifacts) και τα περιβάλλοντα που εμπλέκονται στον κύκλο ζωής ανάπτυξης λογισμικού (SDLC). Η διαχείριση διαμορφώσεων θεωρείται μια κάθετη δραστηριότητα ως προς τον SDLC. Κι αυτό διότι δεν περιορίζεται μόνο στη φάση της υλοποίησης ή της διάταξης, αλλά εντάσσεται σε πολλαπλές φάσεις του SDLC: από τη διαχείριση εκδόσεων απαιτήσεων και σχεδιαστικών μοντέλων (δύο πρώτες φάσεις), μέχρι τη διαμόρφωση και παρακολούθηση του περιβάλλοντος δοκιμών (φάση Επικύρωσης & Επαλήθευσης), τη διαχείριση εκδόσεων λογισμικού (φάση Υλοποίησης) και την τεκμηρίωση των αλλαγών σε παραγωγικά περιβάλλοντα (φάση Συντήρησης & Βελτίωσης).

Στόχος της διαχείρισης διαμορφώσεων είναι να διασφαλίζει ότι κάθε κατασκευή ή έκδοση του λογισμικού είναι αναπαραγωγίσιμη και συνεπής, να διευκολύνει τον έλεγχο αλλαγών, τη διαχείριση γραμμών βάσης (baselines), την παρακολούθηση του ιστορικού και την επαναφορά σε προηγούμενες εκδόσεις όταν απαιτείται. Η δραστηριότητα αυτή υλοποιείται με τη χρήση εργαλείων διαχείρισης διαμορφώσεων, συστημάτων ελέγχου εκδόσεων (όπως Git) και τεχνικών όπως το infrastructure-as-code (IaC) και το configuration-as-code (CaC), ώστε όλες οι διαμορφώσεις να καταγράφονται, να ελέγχονται και να αυτοματοποιούνται.

Συνολικά, η διαχείριση διαμορφώσεων αποτελεί θεμέλιο για τη σταθερότητα, την ασφάλεια και την επαναληψιμότητα των διαδικασιών ανάπτυξης και διάθεσης λογισμικού, υποστηρίζοντας την ομαλή μετάβαση μεταξύ των φάσεων του SDLC και την αποτελεσματική υλοποίηση πρακτικών DevOps και CI/CD.

2.3 DevOps

Καθώς οι απαιτήσεις για ταχύτερες κυκλοφορίες λογισμικού και συνεχή ενημέρωση των εφαρμογών αυξάνεται, η ανάγκη για αυτοματοποίηση και συνεργασία ανάμεσα σε ομάδες ανάπτυξης και λειτουργίας οδήγησε στη δημιουργία της μεθοδολογίας DevOps. Αυτή

επικεντρώνεται στη βελτίωση της αποδοτικότητας μέσω της ενσωμάτωσης εργαλείων και διαδικασιών που εξαλείφουν τα απομονωμένα σιλό μεταξύ προγραμματιστών (developers) και λειτουργών (operations) λογισμικού.

Στο πλαίσιο του DevOps, η αυτοματοποίηση της διαδικασίας ανάπτυξης και διάθεσης λογισμικού υλοποιείται μέσω αγωγών (pipelines), οι οποίοι αποτελούν αλληλουχίες αυτοματοποιημένων βημάτων που καλύπτουν ορισμένα από τα στάδια του κύκλου ζωής του λογισμικού. Ένας αγωγός DevOps διασφαλίζει ότι κάθε αλλαγή στον πηγαίο κώδικα μπορεί να ενσωματωθεί, να δοκιμαστεί, να παραδοθεί και να διατεθεί αξιόπιστα και επαναληπτικά, υλοποιώντας τις βασικές αρχές της μεθοδολογίας DevOps

Στόχος του DevOps είναι να αυτοματοποιούνται και να εκτελούνται βασικές διαδικασίες που διατρέχουν όλο τον κύκλο ζωής του λογισμικού. Ενδεικτικά, αυτές οι διαδικασίες περιλαμβάνουν:

- Διαχείριση Πηγαίου Κώδικα (Source Code Management): Χρήση συστημάτων διαχείρισης εκδόσεων (π.χ. Git) για την παρακολούθηση αλλαγών και την αποφυγή συγκρούσεων. Εργασίες όπως το check-in πηγαίου κώδικα ή η συγχώνευση κλαδιών πραγματοποιούνται μέσω πελατών συστημάτων διαχείρισης εκδόσεων και ενδέχεται να περιλαμβάνονται στον αγωγό CI/CD, ανάλογα με τις ανάγκες.
- Συνεχής Ενσωμάτωση (Continuous Integration - CI): Μετά από κάθε καταχώρηση (commit), ο κώδικας περνά αυτόματα από μια σειρά υπο-εργασιών, όπως το check-in, η μεταγλώττιση και η δοκιμή του, ώστε να ανιχνεύονται έγκαιρα τυχόν σφάλματα.
    - Αυτοματοποιημένες Δοκιμές (Automated Testing): Εκτέλεση δοκιμών μονάδων (unit tests), δοκιμών ενσωμάτωσης (integration tests), δοκιμών συστήματος (system tests) κ.α., για διασφάλιση της λειτουργικότητας και της σωστής συμπεριφοράς του λογισμικού.
- Συνεχής Παράδοση / Διάταξη (Continuous Delivery/Deployment - CD): Αυτόματη δημιουργία εκτελέσιμων πακέτων (artifacts) και η παράδοσή τους σε αποθετήρια τεχνημάτων (artifact registries) (συνεχής παράδοση) ή η διάταξή τους σε δοκιμαστικά (staging) ή παραγωγικά (production) περιβάλλοντα (συνεχής διάταξη).

Βασικά εργαλεία και πρακτικές που εφαρμόζονται στο πλαίσιο του DevOps είναι:

- Συστήματα Διαχείρισης Πηγαίου Κώδικα / Ελέγχου Εκδόσεων (Source Code Management / Version Control Systems): Εργαλεία όπως το Git επιτρέπουν την παρακολούθηση αλλαγών, τη συνεργασία μεταξύ προγραμματιστών και τη διαχείριση διαφορετικών εκδόσεων του κώδικα, αποτελώντας θεμέλιο για τις υπόλοιπες πρακτικές DevOps.
- Εργαλεία Ολοκλήρωσης (CI Tools): Jenkins, GitLab CI, CircleCI για την εκτέλεση CI/CD αγωγών με σκοπό την αυτοματοποίηση της μεταγλώττισης (builds), των δοκιμών (tests) και της παράδοσης (του κώδικα).
- Διαχείριση Υποδομής ως Κώδικα (Infrastructure as Code - IaC): Χρήση εργαλείων, όπως Terraform ή Ansible, για την αυτοματοποιημένη και αναπαραγωγίσιμη κατασκευή περιβαλλόντων (πχ. παραγωγικών, δοκιμής) μέσω κατάλληλης δημιουργίας και διαμόρφωσης διακομιστών (servers) και δικτύων (networks).
- Δοχεία και Ενορχήστρωση (Containers & Orchestration): Χρήση Docker για δημιουργία εικόνων (images) και Kubernetes για διαχείριση εφαρμογών βασιζόμενες σε δοχεία (containerized applications). Τα δοχεία (containers) είναι ελαφριές, απομονωμένες μονάδες λογισμικού που περιλαμβάνουν εφαρμογές και τις εξαρτήσεις τους, εξασφαλίζοντας συνεπή εκτέλεση σε διαφορετικά περιβάλλοντα. Οι δοχειοποιημένες εφαρμογές διευκολύνουν τη φορητότητα, την ταχεία ανάπτυξη και την αποδοτική αξιοποίηση πόρων. Στο πλαίσιο του DevOps, η χρήση τεχνολογιών διαχείρισης δοχείων

(π.χ. Docker) και εργαλείων ενορχήστρωσης (π.χ. Kubernetes) ενισχύει τη συνέπεια, την επεκτασιμότητα και την αυτοματοποίηση των διαδικασιών ανάπτυξης και διάθεσης λογισμικού

- Αυτοματοποιημένη Διάταξη (Automated Deployment): Χρήση στρατηγικών (όπως η τεχνική των blue/green deployments[1]) διάταξης που εξασφαλίζουν την ελάχιστη ή μηδενική διακοπή λειτουργίας των εφαρμογών καθώς αυτές διατάσσονται σε αντίστοιχα περιβάλλοντα.

2.3.1 CI/CD

To Continuous Integration / Continuous Delivery (CI/CD), που σημαίνει Συνεχής Ενσωμάτωση και Συνεχής Παράδοση, αντιπροσωπεύει μία μέθοδο στην ανάπτυξη λογισμικού που αποσκοπεί στην αυτοματοποίηση των διαδικασιών ενσωμάτωσης αλλαγών στον κώδικα, εκτέλεσης αυτοματοποιημένων δοκιμών και παραγωγής/παράδοσης (εκτελέσιμου) κώδικα. Τελικός στόχος είναι να ελεγχθεί αν οι αλλαγές αυτές είναι πλέον κατάλληλες για να ενσωματωθούν στο λογισμικό (σε νέα έκδοσή του, η οποία ελέγχεται από τον αγωγό) ώστε αυτό έπειτα να μπορεί να παραδοθεί στον πελάτη.

Για να κατανοήσουμε τη σημασία της Συνεχούς Ενσωμάτωσης (CI) και της Συνεχούς Παράδοσης (CD), παρακάτω παρατίθενται οι κύριες αρχές και λειτουργίες της μεθόδου αυτής:

1. Continuous Integration (CI): Η Συνεχής Ενσωμάτωση (CI) περιλαμβάνει την αυτόματη ενσωμάτωση αλλαγών στον κώδικα από πολλούς προγραμματιστές σε ένα κοινό αποθετήριο. Κάθε ενσωμάτωση ενεργοποιεί αυτοματοποιημένες διαδικασίες ενσωμάτωσης και δοκιμών (μέσω εκτέλεσης αγωγών CI/CD) για την έγκαιρη ανίχνευση σφαλμάτων, εξασφαλίζοντας την ποιότητα του κώδικα. Επιπλέον, η CI επιτρέπει στις εταιρείες να έχουν μικρότερους και πιο συχνούς κύκλους κυκλοφορίας (release cycles). Αυτό συμβαίνει επειδή η CI αυτοματοποιεί τη διαδικασία ενσωμάτωσης και δοκιμών, μειώνοντας τον χρόνο εντοπισμού και διόρθωσης σφαλμάτων, γεγονός που επιτρέπει ταχύτερες ενημερώσεις στην ανάπτυξη του λογισμικού και την παραγωγή συχνών νέων εκδόσεων κυκλοφορίας. [1]

2. Continuous Delivery (CD): Η Συνεχής Παράδοση (CD) είναι υπεύθυνη για να διασφαλίζει ότι μια εφαρμογή βρίσκεται πάντα σε κατάσταση έτοιμη να προωθηθεί στο παραγωγικό περιβάλλον. Μετά από κάθε αλλαγή στο κώδικα, εκτελείται μία σειρά από δοκιμές που επιβεβαιώνουν ότι η εφαρμογή παραμένει λειτουργική με σωστή και αναμενόμενη συμπεριφορά. Εφόσον οι δοκιμές επιτύχουν (όπου αυτό σηματοδοτεί την επιτυχή ολοκλήρωση του σταδίου CI), ο κώδικας μεταφέρεται σε ένα περιβάλλον δοκιμαστικής παραγωγής (staging environment) που μοιάζει με παραγωγικό ενώ αποθηκεύεται και σε ένα αποθετήριο τεχνημάτων (στάδιο CD). Στο περιβάλλον δοκιμαστικής παραγωγής

[1] Blue/green deployments είναι μια στρατηγική ανάπτυξης λογισμικού όπου δύο σχεδόν ταυτόσημα περιβάλλοντα, το «blue» (τρέχον παραγωγικό) και το «green» (νέο, δοκιμαστικό), λειτουργούν παράλληλα. Μετά την επιτυχή δοκιμή του «green» περιβάλλοντος, η κυκλοφορία των χρηστών μεταφέρεται σε αυτό, εξαλείφοντας το χρόνο εκτός λειτουργίας και επιτρέποντας γρήγορη επαναφορά στην προηγούμενη έκδοση εφόσον χρειαστεί. Για να εξασφαλιστεί η επαναφορά, το «blue» περιβάλλον διατηρείται προσωρινά. Επιπλέον, ενδέχεται έπειτα να ενημερωθεί ώστε να δεχθεί μια νέα έκδοση της εφαρμογής - με αυτό τον τρόπο, ουσιαστικά γίνεται μια εναλλαγή των δύο αυτών περιβαλλόντων

πραγματοποιούνται χειροκίνητα επιπλέον δοκιμές, συνήθως με χειρωνακτικό τρόπο, όπως δοκιμές αποδοχής (acceptance tests). Εφόσον και αυτές οι δοκιμές επιτύχουν, ο νέος κώδικας μπορεί πλέον να μεταφερθεί/εγκατασταθεί σε ένα παραγωγικό περιβάλλον μέσω της χρήσης του αποθετηρίου τεχνημάτων (artifact repository). Συνολικά, η CD επιτρέπει την ταχύτερη διάθεση νέων εκδόσεων λογισμικού, καθώς μειώνει τον χρόνο μεταξύ ανάπτυξης και διάθεσης κώδικα. [1]

3. Continuous Deployment (CDep): Η Συνεχής Διάταξη (Continuous Deployment) αποτελεί το επόμενο βήμα μετά τη Συνεχή Παράδοση (Continuous Delivery), καθώς αυτοματοποιεί πλήρως τη διαδικασία διάθεσης του κώδικα σε παραγωγικό περιβάλλον. Στη Συνεχή Διάταξη κάθε αλλαγή που περνά επιτυχώς τις αυτοματοποιημένες δοκιμές και τους ελέγχους ποιότητας ωθείται αυτόματα στο παραγωγικό περιβάλλον. Αυτό μειώνει δραστικά τον χρόνο και την ανθρώπινη παρέμβαση στη διαδικασία, εξασφαλίζοντας άμεση διάθεση νέων χαρακτηριστικών ή διορθώσεων. Σημειώνεται σε αυτό το σημείο πως στην Συνεχή Διάταξη προϋποτίθεται πως οι δοκιμές αποδοχής και άλλα είδη δοκιμών που εκτελούνται σε περιβάλλοντα δοκιμαστικής παραγωγής αυτοματοποιούνται και μεταφέρονται σε αγωγούς CI/CD μαζί με τις ενέργειες διάταξης των εφαρμογών σε περιβάλλοντα παραγωγής. Για λόγους περισσότερου ελέγχου, μπορεί επίσης να πραγματοποιείται σταδιακή διάταξη (progressive rollout) ώστε να εξασφαλιστεί η σταθερότητα και αξιοπιστία των εφαρμογών πριν αυτές γίνουν διαθέσιμες σε όλους τους τελικούς χρήστες. Αυτό σχετίζεται με τις στρατηγικές διάταξης που αναφέρθηκαν προηγουμένως. [7]

Η μέθοδος CI/CD χρησιμοποιείται προκειμένου να επιτύχει στόχους, οι οποίοι διευκολύνουν την διαδικασία ανάπτυξης λογισμικού. Αυτοί οι στόχοι είναι οι εξής:

1. Αυτοματοποίηση: Οι διαδικασίες CI/CD αυτοματοποιούν τις διαδικασίες ενσωμάτωσης, παράδοσης και διάταξης. Κατά αυτό τον τρόπο, τα ανθρώπινα λάθη μειώνονται ενώ οι χρόνοι παράδοσης επιταχύνονται. Τα βήματα στη διαδικασία κατασκευής λογισμικού, τα οποία δύναται να αυτοματοποιηθούν, εκτελούνται πάντα με ακριβώς τον ίδιο τρόπο. Παράλληλα, τυχόν προβλήματα γίνονται αντιληπτά εγκαίρως, καθιστώντας την διαδικασία διόρθωσης τους πιο εύκολη. Εκτός αυτού, ο έγκαιρος εντοπισμός και διόρθωση των σφαλμάτων οδηγεί στην μείωση του κόστους που θα δημιουργούνταν αν τα λάθη αυτά έπρεπε να διορθωθούν αργότερα στη διαδικασία ανάπτυξης λογισμικού (πχ. μετά την παράδοση/διάταξη του λογισμικού). [1]

2. Διασφάλιση Ποιότητας: Η μέθοδος CI/CD διαδραματίζει σημαντικό ρόλο στη διαδικασία διασφάλισης ποιότητας (Quality Assurance - QA), μέσω της εφαρμογής αυτόματων δοκιμών στα στάδια υλοποίησης και επικύρωσης της ανάπτυξης λογισμικού. Οι αυτόματες δοκιμές, όπως unit tests (δοκιμές μονάδων), integration tests (δοκιμές ενσωμάτωσης), functional tests (λειτουργικές δοκιμές), performance tests (δοκιμές απόδοσης) καθώς και άλλες, εκτελούνται συνεχώς, χωρίς η ανθρώπινη παρέμβαση να είναι απαραίτητη, εξασφαλίζοντας κατά αυτό τον τρόπο ότι οποιαδήποτε αλλαγή δεν θα επιφέρει προβλήματα στην υπάρχουσα λειτουργικότητα αλλά και ποιότητα του λογισμικού. Κατά αυτό τον τρόπο, κρίσιμα προβλήματα γίνονται αντιληπτά εγκαίρως, πρωτού ο κώδικας φτάσει στη παραγωγή, όπου το κόστος διόρθωσής τους θα είναι αρκετά μεγάλο. [2]

3. Κλιμακωσιμότητα (Scalability): Η μέθοδος CI/CD μπορεί να προσαρμοστεί κατάλληλα, ούτως ώστε να αποτελέσει ένα ισχυρό εργαλείο για κάθε είδους ομάδα ανεξαρτήτως μεγέθους. Συγκεκριμένα, λόγω της ισχυρής της προσαρμοστικότητας, προσφέρει δυνατότητες επέκτασης και τροποποίησης, δίνοντας στις ομάδες την δυνατότητα να διαμορφώνουν τη ροή εργασιών με βάση τις δικές τους ανάγκες. [1]

Η μέθοδος CI/CD, η οποία είναι πλέον ευρέως διαδεδομένη, εισάγει σημαντικά οφέλη στη διαδικασία ανάπτυξης κώδικα, ορισμένα από τα οποία είναι τα εξής:

1. Ταχύτητα ανάπτυξης: Με την χρήση της μεθόδου CI/CD, οι ομάδες έχουν την δυνατότητα να αναπτύσσουν νέο κώδικα και να τον ελέγχουν πιο γρήγορα, αυξάνοντας έτσι την παραγωγικότητά τους.
2. Βελτιωμένη ποιότητα παραγόμενου κώδικα: Μέσω των αυτοματοποιημένων δοκιμών εξασφαλίζεται ότι τα σφάλματα ανιχνεύονται και διορθώνονται εγκαίρως, ενισχύοντας την ποιότητα του παραγόμενου λογισμικού.
3. Ευελιξία: Με τη χρήση του CI/CD, οι ομάδες μπορούν να ανταποκρίνονται πιο γρήγορα στις αλλαγές και τις απαιτήσεις της αγοράς.

Οι σημαντικότερες πλατφόρμες και εργαλεία που αυτή τη στιγμή υποστηρίζουν την μέθοδο CI/CD φαίνονται στον παρακάτω πίνακα, ο οποίος παρέχει μια συγκριτική αποτίμηση των πλατφορμών αυτών με βάση 10 κριτήρια:

Πίνακας 2. Εργαλεία που υποστηρίζουν CI/CD

CI/CD πλατφόρμες / Κριτήριο	**Jenkins**	**GitLab CI/CD**	**CircleCI**	**Travis CI**	**GitHub Actions**
Ανοικτού Κώδικα	Ναι	Όχι (αλλά υπάρχει και community edition με περιορισμένα λειτουργικά χαρακτηριστικά	Όχι	Ναι (για open source αλλά outdated)	Όχι

CI/CD πλατφόρμες / Κριτήριο	**Jenkins**	**GitLab CI/CD**	**CircleCI**	**Travis CI**	**GitHub Actions**
Τύπος Υλοποίησης	Αυτο-φιλο ξενούμενο (self-hosted)	SaaS ή self-hosted	Cloud & αυτο-φιλοξενούμενο	Cloud	SaaS
Υποστήριξη Δοχείων (Docker)	Ναι (μέσω plugins & scripts)	Ναι (native)	Ναι (native)	Ναι (μέσω scripts)	Ναι (native)
Συνεχής Διάταξη (CD)	Ναι μέσω scripts & προσθέτων (plugins)	Ναι	Ναι	Ναι	Ναι
Είδη Δοκιμών	Όλα (με plugins)	Όλα (ενσωματωμένα & scripts)	Όλα (ενσωματωμένα & scripts)	Όλα (ενσωματωμένα)	Όλα (ενσωματωμένα & scripts)
Κειμενογράφος Αγώγων	Groovy (text) / Blue Ocean Plugin (graphical)	YAML (text) / Εγγενής Γραφικός	YAML (text)	YAML (text)	YAML (text)
Ευκολία Ενσωμάτωσης	Υψηλή (με plugins	Άριστη (native με	Υψηλή (εγγενής και	Υψηλή (εγγενής,	Άριστη (native με

CI/CD πλατφόρμες / Κριτήριο	**Jenkins**	**GitLab CI/CD**	**CircleCI**	**Travis CI**	**GitHub Actions**
με VCS	αλλά με υποστήριξη για πολλά VCS)	GitLab)	κυρίως για GitHub & Bitbucket)	κυρίως για GitHub)	GitHub)
Επεκτασιμότητα / Plugins	Πολύ μεγάλη (1800+)	Περιορισμένη (100+) κυρίως μέσω APIs	1000+ Orbs (επαναχρησιμοποιήσιμα YAML snippets)	Περιορισμένη - 100+ (APIs/webhooks)	Marketplace με 15.000+ actions
Υποστήριξη Υποδομής ως Κώδικα	Ναι (με plugins)	Ναι (εγγενής & με scripts)	Ναι (με scripts)	Ναι (με scripts)	Ναι (εγγενής για Terraform & μη εγγενής για άλλα IaC πλαίσια μέσω actions/scripts)
Ανάγκη Πρόσθετης Υποδομής	Ναι	Οχι (SaaS) / Ναι (self-hosted)	Οχι (SaaS) / Ναι (self-hosted)	Όχι	Όχι

Με βάση το περιεχόμενο του παραπάνω πίνακα γίνεται εμφανές πως κάθε πλατφόρμα έχει τα δικά της σχετικά πλεονεκτήματα οπότε είναι δύσκολο να θεωρηθεί κάποια από αυτές ως επικρατέστερη ή καλύτερη. Παρατηρώντας όμως πιο προσεκτικά το περιεχόμενο αυτό, μπορεί να εξαχθεί πως οι πλατφόρμες Gitlab CI/CD & Github Actions ξεχωρίζουν με βάση ένα μεγάλο υποσύνολο των επιλεγμένων κριτηρίων. Στην Ενότητα 3.4.1 θα προσδιοριστεί ποια από αυτές τις

(δύο) πλατφόρμες επιλέχθηκε στο πλαίσιο της υλοποίησης της προτεινόμενης λύσης της διπλωματικής μας.

2.3.2 Αγωγοί CI/CD

Με τον όρο αγωγός CI/CD (CI/CD pipeline), γίνεται αναφορά σε μία αυτοματοποιημένη ροή εργασιών (workflow), η οποία έχει ως στόχο να διευκολύνει την διαδικασία ανάπτυξης λογισμικού για συγκεκριμένες δραστηριότητες, από την ενσωμάτωση νέου κώδικα μέχρι την παράδοση ή διάταξή του στο παραγωγικό περιβάλλον. Κάθε βήμα του αγωγού εκτελείται με τρόπο αυτοματοποιημένο ούτως ώστε τα ανθρώπινα λάθη να ελαχιστοποιούνται. Οι αγωγοί CI/CD ακολουθούν τις προαναφερόμενες αρχές Συνεχούς Ενοποίησης και Συνεχούς Παράδοσης (δείτε Ενότητα 2.3.1).

Ένας τυπικός αγωγός CI/CD αποτελείται από διαδοχικά στάδια, όπου κάθε στάδιο περιλαμβάνει μια σειρά από βήματα. Τα κύρια στάδια και η λειτουργία τους είναι τα εξής:

- Έλεγχος πηγαίου κώδικα (Source/Commit Stage): Ο πηγαίος κώδικας εκφορτώνεται από ένα κεντρικό αποθετήριο. Σε αυτό το στάδιο, πραγματοποιούνται ενέργειες, όπως ο συγχρονισμός των αλλαγών και ο έλεγχος για συγκρούσεις. Η εκτέλεση του αρχικού αυτού σταδίου και άρα και του όλου αγωγού ενεργοποιείται από συγκεκριμένα γεγονότα, όπως είναι το commit κώδικα στο κεντρικό αποθετήριο.

- Κατασκευή (Build Stage): Ο πηγαίος κώδικας μεταγλωττίζεται (compiled) και ενοποιείται σε ένα εκτελέσιμο ή πακέτο (δηλαδή ένα τέχνημα - artifact). Εδώ συνήθως γίνεται η συλλογή των εξαρτήσεων, η μετατροπή του κώδικα σε μορφή κατάλληλη για εκτέλεση και η ενοποίηση διαφορετικών τμημάτων του έργου. Για δοχειοποιημένες εφαρμογές, σε αυτό το στάδιο δημιουργείται συνήθως και το δοχείο (container image) της εφαρμογής, π.χ. μέσω ενός αρχείου Dockerfile, το οποίο στη συνέχεια (δείτε στάδιο Παράδοσης) αποθηκεύεται σε αποθετήριο δοχείων (container registry).

- Αυτοματοποιημένοι έλεγχοι (Test Stage): Εκτελούνται αυτοματοποιημένες δοκιμές (unit, integration, κ.ά.) για την επικύρωση της ορθής λειτουργίας του κώδικα. Εδώ ελέγχεται αν ο κώδικας πληροί τις απαιτήσεις που έχουν τεθεί και αν δεν έχει εισαχθεί κάποιο σφάλμα από τις αλλαγές / επεκτάσεις προς ενσωμάτωση.

- Παράδοση (Delivery Stage): Σε αυτό το στάδιο, το παραγόμενο artifact (εκτελέσιμο, πακέτο ή δοχείο) αποθηκεύεται σε ένα αποθετήριο (artifact repository ή container registry), ώστε να είναι διαθέσιμο για μελλοντική διάταξη σε αντίστοιχα περιβάλλοντα (πχ. παραγωγής, staging).

- Διάταξη (Deploy Stage) (Προαιρετικό): Η εφαρμογή διατάσσεται (deploys) στο προκαθορισμένο περιβάλλον (π.χ. staging ή παραγωγής). Σε αυτό το στάδιο, το πακέτο που δημιουργήθηκε μεταφέρεται και ενεργοποιείται στο κατάλληλο περιβάλλον για χρήση από τους τελικούς χρήστες. Για δοχειοποιημένες εφαρμογές, το στάδιο αυτό περιλαμβάνει την ανάκτηση της εικόνας του δοχείου από το αποθετήριο (εικόνων) και την εκτέλεσή της στο επιλεγμένο περιβάλλον (π.χ. μέσω Kubernetes ή άλλου συστήματος ενορχήστρωσης).

Δομή σταδίων και βημάτων σε έναν αγωγό CI/CD:

- Κάθε στάδιο αποτελείται από βήματα (jobs), π.χ. ένα βήμα μεταγλώττισης, ένα βήμα εκτέλεσης unit tests κ.λπ. Για παράδειγμα, το στάδιο δοκιμών μπορεί να περιλαμβάνει δύο βήματα: (α) την εκτέλεση κάποιας αυτοματοποιημένης δοκιμής (πχ. δοκιμής μονάδων) και (β) την μετέπειτα αποθήκευση της παραγόμενης αναφοράς σε ένα αποθετήριο.

- Τα στάδια εκτελούνται συνήθως σειριακά, όπου το επόμενο στάδιο εξαρτάται από την επιτυχία του προηγούμενου. Τα βήματα μέσα σε ένα στάδιο μπορούν να εκτελούνται παράλληλα, αν δεν υπάρχουν εξαρτήσεις μεταξύ τους.

- Αν αποτύχει ένα κρίσιμο βήμα (π.χ. αποτυχία δοκιμών), η εκτέλεση του αγωγού διακόπτεται.

Για την δημιουργία και τον ορισμό ενός αγωγού και των βημάτων που αυτός περιλαμβάνει μπορούν να χρησιμοποιηθούν διάφορες γλώσσες μοντελοποίησης που βασίζονται σε μορφοποιήσεις (formats), όπως YAML, JSON και Groovy. Τονίζεται όμως σε αυτό το σημείο πως κάθε πλατφόρμα CI/CD έρχεται με τη δική της γλώσσα μοντελοποίησης αγωγών οπότε δεν υπάρχει κάποιο είδος προτυποποίησης. Αυτό δυστυχώς οδηγεί σε ένα κλείδωμα (lock-in) στην πλατφόρμα. Επίσης, η υποστήριξη όσον αφορά την ύπαρξη κάποιου editor ποικίλει, όπως φαίνεται στον προηγούμενο Πίνακα 2.

Ο αγωγός CI/CD έχει ως στόχο την εξάλειψη καθυστερήσεων στις παραδόσεις/διατάξεις καθώς επίσης και στην αύξηση της παραγωγικότητας των ομάδων ανάπτυξης. Παράλληλα, οι ομάδες ανάπτυξης υποστηρίζονται σε ότι αφορά τον εντοπισμό σφαλμάτων στα αρχικά στάδια της ανάπτυξης.

Συνεπώς, χρησιμοποιώντας τις πλατφόρμες και εργαλεία που αναφέρονται στην Ενότητα 2.2 και ακολουθώντας τις αρχές του DevOps / CI/CD είναι δυνατόν να δημιουργηθούν αγωγοί, οι οποίοι ελέγχουν το λογισμικό με αυτοματοποιημένο τρόπο και κατευθύνουν τη διαδικασία της ανάπτυξης του.

2.4 Ασφάλεια Δοχειοποιημένων Εφαρμογών

Με τον όρο δοχειοποιημένες εφαρμογές αναφερόμαστε σε εφαρμογές που υλοποιούνται με τεχνολογίες δοχειοποίησης (containerisation), όπως το Docker ή το Kubernetes. Οι εφαρμογές αυτές εκτελούνται μέσα σε απομονωμένα περιβάλλοντα, στο ίδιο λειτουργικό σύστημα (οπότε έχουμε να κάνουμε με ένα είδος εικονικοποίησης του λειτουργικού συστήματος), γνωστά και ως δοχεία (containers), τα οποία περιλαμβάνουν τόσο την ίδια την εφαρμογή όσο και τις απαραίτητες για εκείνη εξαρτήσεις. Για την δημιουργία ενός δοχείου (container) απαιτείται ένα ειδικό πρότυπο που ονομάζεται εικόνα (container image), το οποίο καθορίζει πλήρως την εφαρμογή αλλά και το περιβάλλον εκτέλεσης της. Για την δημιουργία των εικόνων χρησιμοποιούνται ειδικά αρχεία οδηγιών (πχ. Dockerfiles), τα οποία περιγράφουν πλήρως την διαδικασία παραγωγής του περιεχομένου μιας εικόνας (δοχείων).

Συνεπώς, η δοχειοποίηση (containerisation) είναι η διαδικασία μέσω της οποίας μία εφαρμογή μαζί με όλες της τις εξαρτήσεις συσκευάζονται σε ένα ενιαίο πακέτο, με στόχο την φορητότητα και την εκτέλεση της σε διαφορετικά υπολογιστικά περιβάλλοντα. Σημειώνουμε σε αυτό το σημείο πως υπάρχει ευελιξία στο πως κατανέμεται μια εφαρμογή σε δοχεία. Μπορεί όλη η εφαρμογή να "χωρέσει" σε ένα δοχείο ή κάθε συστατικό της εφαρμογής να αντιστοιχεί σε διαφορετικό δοχείο. Η δεύτερη εναλλακτική θεωρείται περισσότερο διαδεδομένη και ευέλικτη διότι προωθεί την αρχιτεκτονική των μικρο-υπηρεσιών και επιτρέπει μια εφαρμογή να κλιμακώνει

ανεξάρτητα τα συστατικά της ανάλογα με τον φόρτο εργασίας που υπεισέρχεται σε αυτά. Επιπλέον, αυτό επιτρέπει την ανεξάρτητη ανάπτυξη των συστατικών της εφαρμογής από διαφορετικά μέλη της ομάδας εργασίας με μάλιστα τη χρήση διαφορετικών γλωσσών προγραμματισμού και τεχνολογιών.

Σε σχέση με τις παραδοσιακές τεχνολογίες εικονικοποίησης (virtualization), οι οποίες βασίζονται στις εικονικές μηχανές (Virtual Machines), η δοχειοποίηση παρουσιάζει σημαντικές διαφορές. Αρχικά, η χρήση δοχείων έχει ως αποτέλεσμα την σημαντική μείωση της κατανάλωσης πόρων καθώς αυτά μοιράζονται τον ίδιο πυρήνα (kernel) του λειτουργικού συστήματος του κεντρικού υπολογιστή (ουσιαστικά του μηχανήματος φιλοξενίας). Αντιθέτως, οι εικονικές μηχανές διαθέτουν το δικό τους λειτουργικό σύστημα, απαιτώντας συνεπώς περισσότερους πόρους. Επίσης, τα δοχεία είναι ελαφρύτερα (έχουν μικρότερο footprint) και αυτό σημαίνει πως εκκινούνται και κλιμακώνονται πολύ πιο γρήγορα σε σχέση με τις εικονικές μηχανές. Επιπλέον, είναι δυνατόν να "χωρέσουν" περισσότερα δοχεία σε ένα φυσικό μηχάνημα σε σχέση με το αντίστοιχο πλήθος εικονικών μηχανών που μπορεί να υποστηριχθεί. Από την άλλη όμως, λόγω της ύπαρξης διακριτού λειτουργικού συστήματος, οι εικονικές μηχανές παρέχουν ισχυρότερο επίπεδο ασφάλειας.

Πρωτού αναλυθούν οι συχνότερες ευπάθειες δοχειοποιημένων εφαρμογών είναι σημαντικό να γίνουν κατανοητές οι βασικές αρχές που διέπουν την ασφάλεια των δοχείων (containers).

- **Απομόνωση (Isolation):** Κάθε δοχείο πρέπει να λειτουργεί ως ένα ανεξάρτητο, απομονωμένο περιβάλλον σε σχέση με τα άλλα δοχεία καθώς και τις διεργασίες του υποκείμενου λειτουργικού συστήματος. Διαφορετικά αυτό μπορεί να οδηγήσει σε επικίνδυνες παρεμβολές, όπως η ανεπιθύμητη διαρροή δεδομένων.
- **Ελαχιστοποίηση Επιφάνειας Επιθέσεων (Minimize attack surface)**: Τα δοχεία πρέπει να περιέχουν μόνο τόσα στοιχεία (components) όσα είναι απαραίτητα για να πραγματώσουν τον σκοπό για τον οποίο δημιουργήθηκαν. Με αυτό τον τρόπο μειώνεται η επιφάνεια επιθέσεων και το αντίστοιχο ρίσκο ασφάλειας.
- **Έλεγχος Πρόσβασης (Access Control)**: Τα δικαιώματα χρηστών και οι συνδέσεις μεταξύ δοχείων πρέπει να είναι περιορισμένες καθώς και να ελέγχονται αυστηρά.

Ωστόσο, παρά τα πλεονεκτήματα τους, οι δοχειοποιημένες εφαρμογές βάλλονται συχνά από προβλήματα ασφάλειας [5].

Τα κύρια ζητήματα ασφάλειας στα δοχεία, τα οποία σχετίζονται με την παραβίαση των προαναφερόμενων τριών βασικών αρχών, μπορούν να περιγραφούν ως εξής [6]:

- Τα δοχεία περιλαμβάνουν λογισμικό από εξαρτήσεις (dependencies / libraries), οι οποίες είτε περιέχουν γνωστές ευπάθειες που δεν έχουν διορθωθεί είτε περιέχουν zero-day ευπάθειες που δεν είναι καν γνωστές.
- Το πρόβλημα επεκτείνεται και στην εικόνα βάσης ενός δοχείου, στην οποία στηρίζεται η κατασκευή της εικόνας του (πχ. από Dockerfile). Η εικόνα βάσης αποτελείται από ένα έτοιμο περιβάλλον, το οποίο ενδέχεται να περιλαμβάνει τρωτά στοιχεία.
- Πολλές φορές τα δοχεία περιλαμβάνουν έναν ριζικό (root) χρήστη που εκτελεί την κύρια διαδικασία τους, το οποίο αυξάνει κατακόρυφα τον κίνδυνο κατάχρησης ενώ ενδέχεται μέσω επιθέσεων (όπως διαφυγής δοχείων - container escape attacks) ο χρήστης αυτός να γίνει ριζικός στο υποκείμενο λειτουργικό σύστημα.
- Συχνά, κατά τη διάρκεια της ανάπτυξης, οι προγραμματιστές συμπεριλαμβάνουν εντός του κώδικα μυστικά (secrets), για λόγους ευκολίας, τα οποία στη συνέχεια θα

έπρεπε να αφαιρούνται. Ωστόσο, αυτό που συμβαίνει συνήθως είναι πως οι προγραμματιστές ξεχνούν να τα αφαιρέσουν. Το παραπάνω συνεπάγεται σοβαρούς κινδύνους ασφάλειας σε περίπτωση που τα μυστικά εκτεθούν (είτε λόγου λανθασμένης διαμόρφωσης κλειστού αποθετηρίου κώδικα/εικόνων είτε λόγω ένθεσης των μυστικών σε ανοιχτά αποθετήρια κώδικα/εικόνων).

- Το σύστημα ενορχήστρωσης δοχείων (Docker) χρησιμοποιεί έναν δαίμονα (docker daemon), ο οποίος απαιτεί ριζικά δικαιώματα (root) για τη λειτουργία του. Αυτό ελλοχεύει σημαντικούς κινδύνους, καθώς οποιαδήποτε ευπάθεια ή κακή διαμόρφωση του ίδιου του Docker daemon ή της διασύνδεσής του (όπως η έκθεση του Docker API χωρίς κατάλληλη αυθεντικοποίηση) μπορεί να οδηγήσει σε μη εξουσιοδοτημένη πρόσβαση στο υποκείμενο λειτουργικό σύστημα (host OS) ή/και στα δοχεία που ενορχηστρώνονται. Για παράδειγμα, ένας επιτιθέμενος που αποκτά πρόσβαση στον Docker daemon μπορεί να δημιουργήσει ή να τροποποιήσει δοχεία με αυξημένα δικαιώματα, να αποκτήσει πρόσβαση στο σύστημα αρχείων του host (δηλ. του μηχανήματος φιλοξενίας) ή να εκτελέσει αυθαίρετο κώδικα, οδηγώντας σε κλιμάκωση προνομίων (privilege escalation). Για αυτό και συνίσταται η εκτέλεση του δαίμονα αυτού σε rootless mode (π.χ. με βάση ειδικό χρήστη και ομάδα χρηστών) εφόσον αυτό είναι εφικτό, ώστε να περιοριστεί ο κίνδυνος μη εξουσιοδοτημένης πρόσβασης και κλιμάκωσης προνομίων.
- Ακόμη, ένα σημαντικό ζήτημα ασφάλειας αποτελεί η ανεπαρκής ρύθμιση των δικτυακών παραμέτρων. Από προεπιλογή, το σύστημα Docker επιτρέπει απεριόριστη επικοινωνία μεταξύ των διαφορετικών δοχείων στο ίδιο σύστημα υποδοχής/φιλοξενίας. Το παραπάνω ενδέχεται να οδηγήσει σε ανεπιθύμητη έκθεση δεδομένων ή σε διευκόλυνση κακόβουλων ενεργειών σε περίπτωση που κάποιο δοχείο παραβιαστεί.
- Επιπλέον, συχνά παρατηρούνται λανθασμένες ρυθμίσεις εκτέλεσης δοχείων, όπου τα δοχεία εκτελούνται με περισσότερα δικαιώματα από όσα πραγματικά απαιτούνται. Για παράδειγμα, ένα δοχείο μπορεί να έχει δικαίωμα δημιουργίας άλλων δοχείων ή να τρέχει με δικαιώματα ριζικού χρήστη (root), αυξάνοντας σημαντικά τον κίνδυνο κλιμάκωσης προνομίων σε περίπτωση παραβίασης. Επίσης, η υπερβολική έκθεση θυρών αποτελεί συχνό πρόβλημα: όταν ανοίγονται περισσότερες θύρες από όσες είναι απαραίτητες ή όταν γίνεται αντιστοίχιση θυρών των δοχείων με θύρες του υποκείμενου λειτουργικού συστήματος (host), αυξάνεται η επιφάνεια επιθέσεων και διευκολύνεται η πρόσβαση από μη εξουσιοδοτημένους χρήστες.
- Ένα ακόμη κρίσιμο σημείο είναι ο διαμοιρασμός αποθηκευτικών χώρων (volumes) τόσο μεταξύ δοχείων όσο και μεταξύ του λειτουργικού συστήματος και των δοχείων. Εάν οι κατάλληλες άδειες δεν έχουν ρυθμιστεί σωστά, ένας επιτιθέμενος που αποκτά πρόσβαση σε ένα δοχείο μπορεί να εκμεταλλευτεί τον κοινόχρηστο αποθηκευτικό χώρο για να προσπελάσει ή να αλλοιώσει δεδομένα άλλων δοχείων ή ακόμη και του ίδιου του λειτουργικού συστήματος, οδηγώντας σε διαρροή ευαίσθητων πληροφοριών ή περαιτέρω παραβιάσεις.

2.5 DevSecOps

Καθώς το CI/CD εξελισσόταν και αποκτούσε όλο και πιο ενεργό ρόλο στη διαδικασία ανάπτυξης και ελέγχου του κώδικα προέκυψε μία νέα ανάγκη: το αρμόδιο προσωπικό, το οποίο έλεγχε τον κώδικα από την οπτική της ασφάλειας, δεν ήταν πλέον σε θέση να προλάβει τους γοργούς ρυθμούς που εισήγαγε το CI/CD [3]. Επομένως, επιβλήθηκε η ανάγκη, η ασφάλεια να ενταχθεί μέσα στη ροή εργασιών CI/CD. Συνέπεια αυτού ήταν, η έως τότε ονομαζόμενη

τεχνολογία DevOps, να εμπλουτιστεί με εργαλεία που ελέγχουν την ασφάλεια της εφαρμογής και να ιδρυθεί ο κλάδος που σήμερα ονομάζεται DevSecOps.

Οι βασικές αρχές πάνω στις οποίες στηρίζεται το DevSecOps είναι οι εξής:

- Ενσωμάτωση της ασφάλειας σε όλα τα στάδια: Το DevSecOps δίνει έμφαση στην ενσωμάτωση πρακτικών ασφάλειας σε κάθε φάση του κύκλου ζωής ανάπτυξης λογισμικού, από το σχεδιασμό και την υλοποίηση μέχρι τη δοκιμή και τη διάθεση. Η ασφάλεια παύει να είναι ένα ξεχωριστό στάδιο στο τέλος (της ανάπτυξης) και γίνεται οργανικό μέρος της διαδικασίας (ανάπτυξης).
- Αυτοματοποίηση ελέγχων και διαδικασιών ασφάλειας: Η αυτοματοποίηση εργαλείων και ελέγχων ασφάλειας (όπως σάρωση ευπαθειών, στατική και δυναμική ανάλυση κώδικα, έλεγχος μυστικών, συμμόρφωση) διασφαλίζει ότι οι έλεγχοι (ασφάλειας) γίνονται συνεχώς και με συνέπεια, χωρίς να επιβραδύνουν την ανάπτυξη.
- Πολιτική "Shift Left": Η ασφάλεια μεταφέρεται όσο το δυνατόν νωρίτερα στη διαδικασία ανάπτυξης ("shift left"), ώστε τα ζητήματα να εντοπίζονται και να αντιμετωπίζονται έγκαιρα, μειώνοντας το κόστος και το ρίσκο.
- Συνεχής παρακολούθηση και ανταπόκριση: Η συνεχής παρακολούθηση εφαρμογών και υποδομών για εντοπισμό και απόκριση σε απειλές σε πραγματικό χρόνο είναι βασικό στοιχείο, όπως και η ύπαρξη σχεδίου αντιμετώπισης περιστατικών.
- Συνεργασία και κοινή ευθύνη: Η ασφάλεια γίνεται υπόθεση όλων των εμπλεκόμενων ομάδων (ανάπτυξης, λειτουργίας, ασφάλειας), ενισχύοντας τη συνεργασία και τη διαφάνεια.
- Συνεχής βελτίωση: Μέσω τακτικής ανατροφοδότησης και αναθεώρησης των διαδικασιών, οι ομάδες βελτιώνουν διαρκώς τα μέτρα ασφάλειας και προσαρμόζονται σε νέες απειλές.

Η υιοθέτηση των αρχών του DevSecOps προσφέρει σημαντικά οφέλη τόσο για τις ομάδες ανάπτυξης όσο και για τον οργανισμό συνολικά. Τα βασικά πλεονεκτήματα που προκύπτουν από αυτήν την προσέγγιση συνοψίζονται ως εξής:

- Πρόληψη ευπαθειών και μείωση κινδύνου: Η έγκαιρη ανίχνευση και διόρθωση ευπαθειών μειώνει τον κίνδυνο παραβιάσεων και επιθέσεων.
- Ταχύτερη και ασφαλέστερη παράδοση λογισμικού: Η αυτοματοποίηση και ο συνεχής έλεγχος ασφάλειας επιτρέπουν την ταχύτερη κυκλοφορία νέων και ασφαλών εκδόσεων (λογισμικού), χωρίς επομένως να θυσιάζεται η ασφάλεια.
- Συμμόρφωση με κανονισμούς: Η ενσωμάτωση αυτοματοποιημένων ελέγχων συμμόρφωσης διευκολύνει την τήρηση κανονιστικών απαιτήσεων και προτύπων.
- Βελτιωμένη συνεργασία και κουλτούρα ασφάλειας: Η στενή συνεργασία μεταξύ ομάδων ανάπτυξης, λειτουργίας και ασφάλειας ενισχύει τη συνολική ασφάλεια και καλλιεργεί κουλτούρα ευθύνης και διαφάνειας.
- Μείωση κόστους: Η επίλυση προβλημάτων ασφάλειας σε πρώιμο στάδιο είναι σημαντικά φθηνότερη από την αποκατάσταση μετά από παραβίαση ή την επιβολή προστίμων λόγω μη συμμόρφωσης.

Στο DevSecOps είναι σημαντικό να λαμβάνονται σοβαρά υπόψη οι απαιτήσεις ασφάλειας, όπως αυτές τίθενται από τα αρμόδια τμήματα. Έτσι, στον τομέα του DevSecOps, ένας αντίστοιχος αγωγός θα πρέπει να περιλαμβάνει επιπλέον εργασίες με έμφαση στην ασφάλεια, όπως οι ακόλουθες [4]:

- Έλεγχος Μυστικών (Secret Check): Αφορά τον έλεγχο του κώδικα (του λογισμικού) για την ύπαρξη μυστικών (με τον όρο μυστικά γίνεται αναφορά σε κωδικούς και συνθηματικά), τα οποία για λόγους ασφάλειας δεν θα έπρεπε να περιέχονται σε αυτόν. Η ανάγκη αυτή προέκυψε καθώς παρατηρήθηκε πως συχνά οι προγραμματιστές (developers)

αποθήκευαν μυστικά μέσα στον κώδικα χωρίς να χρησιμοποιούν τις κατάλληλες μεθόδους ασφαλής αποθήκευσής τους και στην συνέχεια αμελούσαν να τα αφαιρέσουν.

- Ανάλυση Σύνθεσης Λογισμικού (Software Composition Analysis): Σε αυτό το σημείο της διαδικασίας, ελέγχονται οι βιβλιοθήκες που χρησιμοποιούνται στον κώδικα και ενδεχομένως (αλλά όχι απαραίτητα) να προέρχονται από τρίτους, για τυχόν ευπάθειες. Σκοπός είναι να διασφαλιστεί ότι η χρήση των εν λόγω βιβλιοθηκών είναι νόμιμη και ασφαλής, βάση των πολιτικών ασφάλειας και τυχόν άλλων απαιτήσεων του εκάστοτε οργανισμού. Συνεπώς, όταν μία βιβλιοθήκη περιλαμβάνει κρίσιμες (critical) ευπάθειες, η χρήση της θα πρέπει να απαγορεύεται.
- Στατική Δοκιμή Ασφάλειας (Static Security Testing): Γίνεται έλεγχος του πηγαίου κώδικα (ή του εκτελέσιμου κώδικα) για ευπάθειες και κακές πρακτικές ανάπτυξης, μέσω τεχνικών στατικής ανάλυσης.
- Έλεγχος Αρχείου Κατασκευής Εικόνας Δοχείων (Image Build File Inspection): Το αρχείο κατασκευής (πχ. Dockerfile) της εικόνας δοχείων για την εφαρμογή ή τα συστατικά της ελέγχεται για βέλτιστες πρακτικές και κοινά προβλήματα ασφάλειας.
- Σάρωση Εικόνων Δοχείων (Image Scanning): Η εικόνα βάσης ενός δοχείου και η εικόνα του ίδιου του δοχείου ελέγχονται για ευπάθειες.
- Έλεγχος για Ιούς: Πριν από τη μεταφόρτωση της εικόνας δοχείου σε αποθετήριο, πραγματοποιείται έλεγχος για ιούς και κακόβουλο λογισμικό. Ο έλεγχος αυτός διασφαλίζει ότι η εικόνα δεν περιέχει επιβλαβές περιεχόμενο, προστατεύοντας έτσι τόσο το αποθετήριο όσο και τα συστήματα στα οποία θα αναπτυχθεί η εικόνα.
- Δυναμική Δοκιμή Ασφάλειας (Dynamic Security Testing): Σε μία δοκιμαστική έκδοση του λογισμικού (και στο αντίστοιχο περιβάλλον δοκιμών) εκτελούνται δοκιμές παρείσδυσης (penetration testing) προκειμένου να εντοπιστούν ευπάθειες. Αυτό σημαίνει πως η ασφάλεια της εφαρμογής εξετάζεται δυναμικά κατά την διάρκεια της εκτέλεσής της (runtime). Αυτός ο έλεγχος ασφάλειας μπορεί να γίνεται τοπικά ή σε ορισμένες περιπτώσεις απομακρυσμένα (πχ. να πραγματοποιείται διάταξη της εφαρμογής σε ένα άλλο περιβάλλον και έπειτα εκτέλεση των δοκιμών από το τρέχων περιβάλλον δοκιμής). Η δυναμική δοκιμή ασφάλειας είναι συμπληρωματική ως προς τη στατική δοκιμή ασφάλειας διότι μπορεί να εντοπίσει διαφορετικές ευπάθειες σε σχέση με αυτές που ανακαλύπτονται από τη στατική δοκιμή.
- Υπογραφή Εικόνας Δοχείων (Container Image Signing): Η εικόνα δοχείου που παράγεται υπογράφεται προκειμένου να εξασφαλιστεί η ακεραιότητα της. Στη συνέχεια, η υπογεγραμμένη εικόνα μεταφορτώνεται (push) σε ένα αποθετήριο εικόνων (container registry), όπως το Docker Hub ή ένα ιδιωτικό registry. Η μεταφόρτωση αυτή επιτρέπει την ασφαλή αποθήκευση και διανομή της εικόνας, ενώ παράλληλα διατηρείται η υπογραφή και τα σχετικά μεταδεδομένα, ώστε κάθε χρήστης ή σύστημα που θα την ανακτήσει να μπορεί να επαληθεύσει την προέλευση και την ακεραιότητά της πριν τη χρησιμοποιήσει.

3 Ανάπτυξη Συστήματος

3.1 Το μοντέλο του καταρράκτη

Στο παρόν κεφάλαιο παρουσιάζεται η διαδικασία ανάπτυξης του συστήματος, η οποία ακολουθεί το μοντέλο του καταρράκτη (Waterfall model). Σύμφωνα με το συγκεκριμένο μοντέλο, η ανάπτυξη πραγματοποιείται με διαδοχικά, σαφώς διαχωρισμένα στάδια/δραστηριότητες, όπου κάθε δραστηριότητα ολοκληρώνεται πλήρως πριν ξεκινήσει η επόμενη. Αυτή η σειριακή προσέγγιση επιτρέπει τη συστηματική ανάλυση και τεκμηρίωση κάθε φάσης της ανάπτυξης, διασφαλίζοντας ότι τα αποτελέσματα κάθε σταδίου αξιοποιούνται ως (ολοκληρωμένη) βάση για το επόμενο.

Στις επόμενες ενότητες του κεφαλαίου, παρουσιάζονται αναλυτικά τα αποτελέσματα από την εκτέλεση της κάθε δραστηριότητας ανάπτυξης, ξεκινώντας από την ανάλυση απαιτήσεων και συνεχίζοντας με τον σχεδιασμό και την υλοποίηση. Η τελευταία ενότητα προχωρά σε έναν απολογισμό σε σχέση με τον βαθμό ικανοποίησης των απαιτήσεων που τέθηκαν από το σύστημά μας.

3.2 Ανάλυση Απαιτήσεων

Το σύστημα, προκειμένου να λειτουργεί ικανοποιητικά, πρέπει να ικανοποιεί ορισμένες απαιτήσεις, οι οποίες έχουν τεθεί από τη συγγραφέα της παρούσας εργασίας και ορίστηκαν με βάση την ανάλυση των αναγκών και των στόχων, έχοντας ως κύριο άξονα τις βέλτιστες πρακτικές και την υπάρχουσα βιβλιογραφία. Οι απαιτήσεις του συστήματος διαχωρίζονται σε λειτουργικές και μη λειτουργικές και περιγράφονται αναλυτικά στη συνέχεια.

3.2.1 Λειτουργικές Απαιτήσεις

Για την εξαγωγή και προσδιορισμό των λειτουργικών απαιτήσεων λήφθηκε υπόψη η σχετική βιβλιογραφία ([9], [10], [11]). Εν τέλει, οι εξής λειτουργικές απαιτήσεις τέθηκαν για το σύστημά μας:

- **Αυτοματοποιημένη Διαχείριση Κώδικα:** Το σύστημα πρέπει να υποστηρίζει την αυτόματη εκτέλεση του αγωγού CI/CD (προς ανάπτυξη) κάθε φορά που κάποια αλλαγή καταχωρείται στον κώδικα. Αυτό σημαίνει πως η αλλαγή θα πρέπει να ανιχνεύεται ώστε να πραγματοποιείται έπειτα η αυτοματοποιημένη εκτέλεση του αγωγού. Ακόμη, το σύστημα πρέπει να διαχειρίζεται αποτελεσματικά τις αλλαγές και να πραγματοποιεί έλεγχο εκδόσεων (version control). Τέλος, πρέπει να αναλαμβάνει και τη δημιουργία

τεχνημάτων (artifacts) (όπως δοχεία εικόνων, εκδόσεις κώδικα, κλπ.), ώστε κάθε αλλαγή να μπορεί να αναπαραχθεί και να αναπτυχθεί αξιόπιστα.

- **Συνεχής Ενσωμάτωση και Παράδοση (CI/CD):** Ο αγωγός (προς ανάπτυξη) πρέπει να επιτρέπει την προώθηση του κώδικα σε διαφορετικά περιβάλλοντα (ανάπτυξης, δοκιμών, παραγωγής) με μηχανισμούς ελέγχου/δοκιμής και εγκρίσεων, διασφαλίζοντας την ομαλή διάθεση νέων και ασφαλών εκδόσεων του κώδικα. Ο εκτελέσιμος κώδικας θα πρέπει επίσης να αποθηκεύεται σε κατάλληλα αποθετήρια (υλοποίηση σεναρίων συνεχούς παράδοσης).

- **Έλεγχοι Ασφάλεια (Security):** Το σύστημα πρέπει να συμπεριλαμβάνει εργαλεία, τα οποία θα πρέπει να μπορούν να πραγματοποιούν διάφορα είδη ελέγχων/σαρώσεων ασφάλειας, όπως σάρωση για ευπάθειες και σάρωση για κακόβουλο ή μολυσμένο λογισμικό. Θα πρέπει επίσης να συμπεριλαμβάνονται και έλεγχοι αναφορικά με την διαχείριση μυστικών και ευαίσθητων δεδομένων. Τα εργαλεία αυτά θα πρέπει να μπορούν να γίνουν εκμεταλλεύσιμα από τον αγωγό CI/CD.

  Τα είδη ελέγχων προς υποστήριξη περιλαμβάνουν τόσο παραδοσιακούς ελέγχους (σε CI/CD αγωγούς) όσο και ελέγχους ασφαλείας (που ανήκουν πλέον σε αγωγούς DevSecOps):

  1. Δοκιμές μονάδων (unit tests)
  2. Δοκιμές ενσωμάτωσης (integration tests)
  3. Στατικός έλεγχος ασφάλειας (static security analysis)
  4. Ανάλυση εξαρτήσεων (dependency analysis)
  5. Ανίχνευση μυστικών (secrets detection)
  6. Ανάλυση (ασφάλειας) Dockerfile (dockerfile analysis) - οπότε υπάρχει εστίαση σε εικόνες δοχείων Docker λόγω του γεγονότος πως το Docker αποτελεί ένα de facto πρότυπο στην δοχειοποίηση εφαρμογών
  7. Δυναμικός έλεγχος ασφάλειας (dynamic security analysis)
  8. Σάρωση για ευπάθειες (σε εικόνες δοχείων) (vulnerability scanning in container images)
  9. Σάρωση για κακόβουλο λογισμικό (malware scan) (σε εικόνες δοχείων)
  10. Έλεγχος υπογραφών εικόνων δοχείων.

- **Παρακολούθηση και Αναφορές (Reporting):** Είναι σημαντικό το σύστημα να προσφέρει μηχανισμούς που επιτρέπουν την παρακολούθηση της κατάστασης του αγωγού σε κάθε εκτέλεσή του. Ταυτόχρονα, το σύστημα πρέπει να ανατροφοδοτεί τον χρήστη με κατάλληλα μηνύματα σφάλματος και αναλυτικές αναφορές, όπου αυτό είναι απαραίτητο. Αυτό περιλαμβάνει την δημιουργία και διανομή καταχωρήσεων (logs) σε κάθε βήμα αλλά και την άμεση ενημέρωση των προγραμματιστών (developers) σε περίπτωση αποτυχίας (ή και επιτυχίας) εκτέλεσης του αγωγού. Παράλληλα, με κατάλληλη παραμετροποίηση του αγωγού θα πρέπει ο χρήστης να μπορεί να προσαρμόσει της συνθήκες αποτυχίας σε κάθε έλεγχο που πραγματοποιείται.

- **Διασύνδεση με Τρίτα Εργαλεία (Third Party Integration):** Το σύστημα πρέπει να προσφέρει δυνατότητες διασύνδεσης με τρίτα εργαλεία (πχ. Github, Docker, Discord), όπου αυτό είναι απαραίτητο, για την βελτίωση της ποιότητας και της απόδοσης των υπηρεσιών. Αυτό αφορά τόσο λειτουργικούς σκοπούς (πχ. διαχείριση εκδόσεων εικόνας

δοχείων μέσω Docker) αλλά και μη λειτουργικούς σκοπούς, όπως την βελτίωση της απόδοσης και της παρακολούθησης του αγωγού μέσω χρήσης του Discord ως ένα εργαλείο άμεσης ενημέρωσης της ομάδας έργου (ουσιαστικά ένα τέτοιο εργαλείο θα πρέπει να χρησιμοποιείται για λόγους άμεσης ενημέρωσης των χρηστών χωρίς αυτοί να έχουν πάντοτε ή αναγκαστικά απευθείας πρόσβαση στο σύστημα).

3.2.2 Μη Λειτουργικές Απαιτήσεις

Για την ανάλυση των μη λειτουργικών απαιτήσεων, λήφθηκαν υπόψη οι εξής αναφορές: [8], [12]. Οι μη λειτουργικές απαιτήσεις που εν τέλει τέθηκαν είναι οι εξής:

- **Διαθεσιμότητα (Availability):** Το σύστημα οφείλει να παραμένει λειτουργικό και προσβάσιμο στους χρήστες του ανά πάσα χρονική στιγμή. Αυτό συνεπάγεται ότι οι διακοπές λειτουργίας θα πρέπει να ελαχιστοποιούνται, ανεξάρτητα από τον λόγο που τις προκάλεσε (προγραμματισμένη συντήρηση ή απρόβλεπτα προβλήματα). Το σύστημα προς ανάπτυξη αναμένεται να είναι διαθέσιμο κατά 99,9% ανά ημερολογιακό τρίμηνο [12].

- **Απόδοση (Performance):** Η απόδοση αναφέρεται στην ικανότητα του συστήματος να εκτελεί τις προκαθορισμένες λειτουργίες του με ταχύτητα και αποτελεσματικότητα, διασφαλίζοντας παράλληλα την ομαλή εξυπηρέτηση των αιτημάτων που λαμβάνει. Το σύστημα πρέπει να παρέχει χαμηλό χρόνο απόκρισης (ανάλογα το μέγεθος του project) υπό φυσιολογικό φόρτο εργασίας και να παραμένει σταθερό και λειτουργικό κατά την επεξεργασία πολλαπλών αιτημάτων.

- **Κλιμακωσιμότητα (Scalability):** Το σύστημα θα πρέπει να έχει την ικανότητα να κλιμακώνεται με τέτοιο τρόπο ώστε να εξασφαλίζει πως ανταποκρίνεται πάντα με την ίδια απόδοση, ανεξαρτήτως του φόρτου εργασίας.

- **Ασφάλεια (Security):** Η ασφάλεια αφορά την απαίτηση, το σύστημα και τα δεδομένα του να προστατεύονται από μη εξουσιοδοτημένη πρόσβαση και κακόβουλες επιθέσεις. Ειδικότερα, απαιτείται η ύπαρξη μηχανισμού ελέγχου πρόσβασης με βάση ρόλους (Role Based Access Control - RBAC) και ελέγχου ταυτότητας δύο παραγόντων (Two Factor Authentication - 2FA). Παράλληλα, το σύστημα θα πρέπει να χρησιμοποιεί τις νεότερες εκδόσεις λογισμικού (όπως εξαρτήσεις ή βιβλιοθήκες) και να είναι προσβάσιμο μόνο μέσω HTTPS. Για την προστασία ευαίσθητων δεδομένων, όπως κωδικοί πρόσβασης και μυστικά, θα πρέπει να χρησιμοποιούνται κατάλληλα μέτρα ασφάλειας. Ειδικότερα, στην πλατφόρμα CI/CD, τα μυστικά θα πρέπει να αποθηκεύονται με ασφάλεια στην ίδια την πλατφόρμα και όχι απευθείας στους αγωγούς που αυτή διαχειρίζεται (βλ. Ενότητα 4.1.2).

- **Αξιοπιστία (Reliability):** Η απαίτηση της αξιοπιστίας αναφέρεται στην ικανότητα του συστήματος να λειτουργεί χωρίς διακοπές και να επανέρχεται γρήγορα μετά από καταστροφές (disaster recovery). Η αξιοπιστία ενός συστήματος απαιτεί την μη παρουσίαση σφαλμάτων, την ύπαρξη μηχανισμών επανεκτέλεσης (retry policies) και τη διατήρηση κατάστασης (state persistence) για αποφυγή απώλειας δεδομένων. Επιπλέον, πρέπει να υπάρχει σύστημα παρακολούθησης (monitoring & alerts) για έγκαιρη διάγνωση προβλημάτων. Αυτές οι απαιτήσεις διασφαλίζουν τη σταθερή και ανθεκτική λειτουργία του συστήματος.

- **Ευελιξία Διαμόρφωσης & Επαναχρησιμοποίηση:** Ο αγωγός πρέπει να διαμορφωθεί κατάλληλα ώστε τα επιμέρους βήματα του να ομαδοποιούνται και να δομούνται με τέτοιο τρόπο ώστε να μπορούν να επαναχρησιμοποιηθούν σε νέους αγωγούς.

3.3 Σχεδίαση

Σε αυτή την ενότητα θα περιγραφεί ο σχεδιασμός του συστήματος που αναπτύχθηκε στο πλαίσιο της παρούσας εργασίας.

3.3.1 Αφηρημένη Αρχιτεκτονική Συστήματος

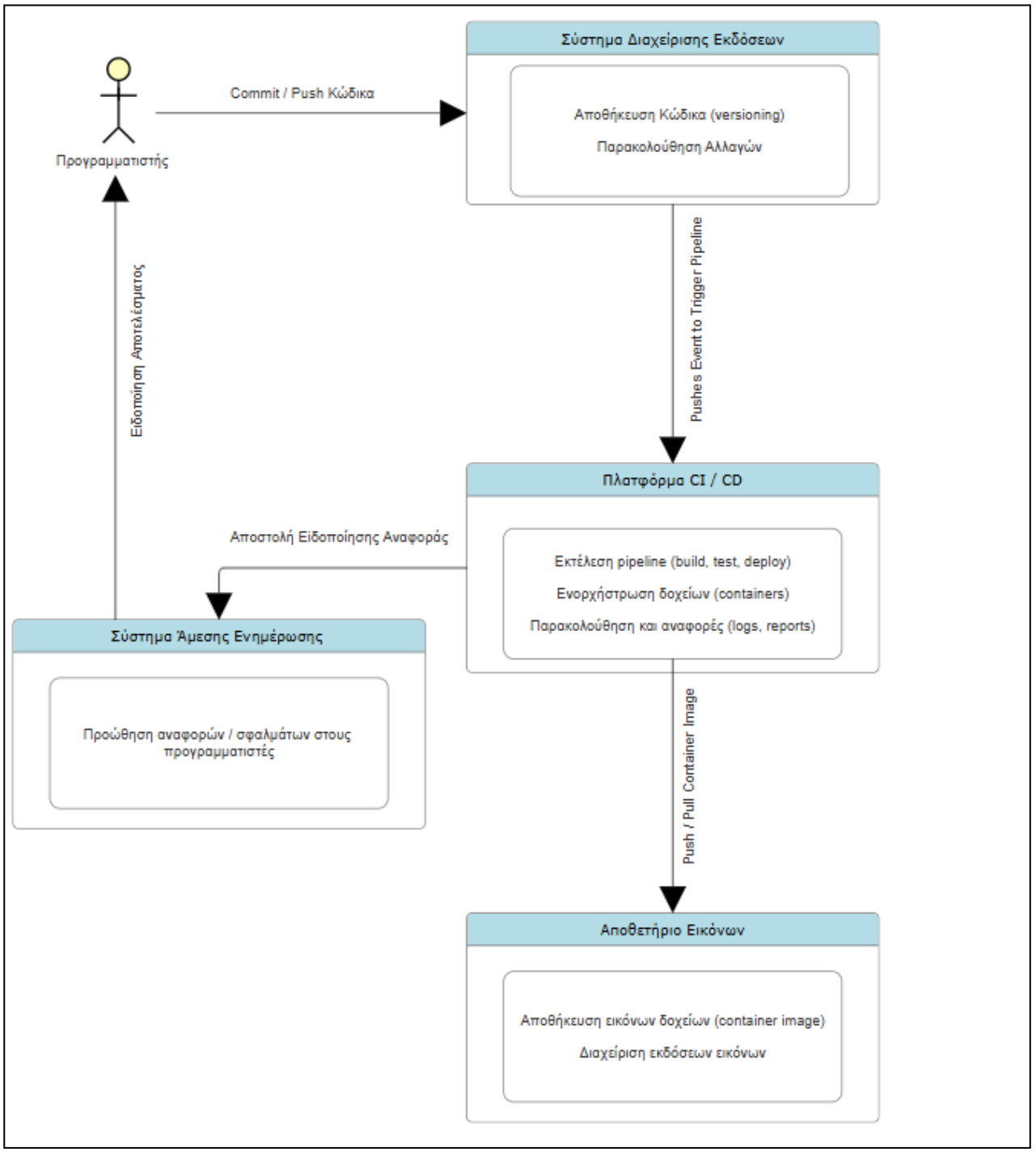
Το σύστημα, το οποίο σχεδιάστηκε και υλοποιήθηκε, περιλαμβάνει τα εξής τέσσερα τμήματα που είναι υπεύθυνα για την λειτουργία του CI/CD αγωγού (pipeline).

1. Πλατφόρμα Διαχείρισης Εκδόσεων (Version Control/Management Platform): Η πλατφόρμα διαχείρισης εκδόσεων (κώδικα) αξιοποιείται για την αποθήκευση του κώδικα της εκάστοτε εφαρμογής και του κώδικα του αγωγού ή των αγωγών (προς επίτευξη δυνατότητας CI/CD για την εφαρμογή αυτή) μέσω της χρήσης αποθετηρίων κώδικα, τα οποία φιλοξενεί και διαχειρίζεται.
2. Πλατφόρμα CI / CD: Η πλατφόρμα CI/CD είναι υπεύθυνη για την διαχείριση και εκτέλεση του αγωγού που αναπτύχθηκε στα πλαίσια της παρούσας εργασίας. Προφανώς, μπορεί να προσφέρει τη λειτουργικότητα αυτή για οποιοδήποτε αγωγό. Εσωτερικά της πλατφόρμας CI/CD υπάρχουν τα εξής υπο-συστήματα:
    a. Σύστημα Εκτέλεσης και Ενορχήστρωσης Δοχείων (Container Execution & Orchestration System): Ο αγωγός που εκτελείται στην πλατφόρμα/υπηρεσία CI/CD αιτείται τη δημιουργία και εκτέλεση όλων των απαραίτητων δοχείων (containers) (πχ. στο πλαίσιο υλοποίησης κάποιου ελέγχου ασφαλείας). Τα δοχεία αναπτύσσονται και εκτελούνται στα σημεία που έχουν οριστεί από τον αγωγό. Οι αιτήσεις από τον αγωγό λαμβάνονται και υλοποιούνται από το Σύστημα Εκτέλεσης & Ενορχήστρωσης Δοχείων, το οποίο ενσωματώνεται στη πλατφόρμα CI/CD.
    b. Σύστημα Αναφορών και Παρακολούθησης (Monitoring & Reporting System): Η πλατφόρμα CI/CD ενσωματώνει ένα ολοκληρωμένο σύστημα παρακολούθησης, το οποίο επιτρέπει την δημιουργία και αποθήκευση αναλυτικών αναφορών (reports). Αυτό επιτρέπει τη διαρκή παρακολούθηση της κατάστασης του αγωγού κατά τη διάρκεια της εκτέλεσης του. Παράλληλα, η ανατροφοδότηση που παρέχει δίνει αναλυτική αναφορά σχετικά με το αποτέλεσμα της εκτέλεσης του αγωγού και των βημάτων του (επιτυχία, αποτυχία, σφάλμα, κλπ.).
3. Σύστημα Άμεσης Ενημέρωσης (Instant Messaging System): Η πλατφόρμα CI/CD συνεργάζεται με ένα εξωτερικό σύστημα (ως προς αυτήν), το οποίο προωθεί τις αναφορές και τα μηνύματα που παράγει ο αγωγός στους προγραμματιστές (developers / programmers) σε πραγματικό χρόνο.
4. Σύστημα Διαχείρισης Εκδόσεων Εικόνων (Container Image Version Management System): Η πλατφόρμα διαχείρισης εκδόσεων (εικόνων) αξιοποιείται για την αποθήκευση της εικόνας της εφαρμογής όταν αυτή έχει περάσει με επιτυχία όλα τα στάδια του αγωγού.

Τα παραπάνω συστατικά λειτουργούν συνδυαστικά για να διασφαλίσουν την αρμονική λειτουργία του συστήματός μας. Σημειώνεται πως ενδέχεται μια πλατφόρμα CI/CD να

περιλαμβάνει όλα αυτά τα τμήματα ή να περιλαμβάνει ορισμένα από αυτά και να ενσωματώνεται με τα υπόλοιπα.

Ένα σχεδιάγραμμα της αρχιτεκτονικής σε υψηλό επίπεδο, παρουσιάζεται παρακάτω:



Εικόνα 1. Η αφηρημένη αρχιτεκτονική του συστήματος σε διάγραμμα συστατικών

3.3.1.1 Σχεδιασμός Αγωγού DevSecOps σε Περιβάλλον CI/CD

Ο αγωγός και τα απαραίτητα εργαλεία για την εκτέλεσή του θα λειτουργούν σε εικονικές μηχανές (virtual machines) και δοχεία (containers), αξιοποιώντας ένα σύστημα ενορχήστρωσης

δοχείων (δηλ. το Σύστημα Εκτέλεσης και Ενορχήστρωσης Δοχείων) που εντάσσεται στην πλατφόρμα CI/CD. Ο ίδιος ο αγωγός αποθηκεύεται στον πηγαίο κώδικα, ενώ τα εργαλεία εκτέλεσής του βρίσκονται εντός του συστήματος ενορχήστρωσης. Ο αναλυτικός σχεδιασμός και η υλοποίησή του αγωγού παρουσιάζονται στην ενότητα 3.4.2.

3.3.1.2 Αγωγός CI/CD

Ο CI / CD αγωγός, που στοχεύει στη διασφάλιση της ασφάλειας του υπό εξέταση λογισμικού, καθώς και σχεδιάστηκε και υλοποιήθηκε για τις ανάγκες της παρούσας εργασίας, περιλαμβάνει τα εξής στάδια:

1. Φάση κατασκευής & λειτουργικών δοκιμών (code build and functional testing): Σε αυτό το στάδιο πραγματοποιούνται η κατασκευή μονάδων, τεχνημάτων και του εκτελέσιμου κώδικα του λογισμικού καθώς και οι καθιερωμένοι, παραδοσιακοί έλεγχοι σε αυτό, όπως οι δοκιμές μονάδων & ενσωμάτωσης. Σκοπός των ελέγχων είναι να διασφαλίσουν ότι το λογισμικό έχει την αναμενόμενη λειτουργικότητα & συμπεριφορά.

2. Στατική ανάλυση ασφάλειας (static security analysis): Στη συνέχεια, ο κώδικας και όλες οι εξαρτήσεις σε αυτόν (dependencies) ελέγχονται με στόχο τον εντοπισμό γνωστών ή καταγεγραμμένων ευπαθειών. Συνεπώς, μεγάλη έμφαση δίνεται και στις βιβλιοθήκες (libraries) που συνοδεύουν το υπό εξέταση λογισμικό. Αυτό το βήμα αφορά την ανάλυση εξαρτήσεων, τον έλεγχο για μυστικά, και την κλασική στατική ανάλυση τρωτοτήτων.

3. Φάση παραγωγής και ελέγχου εικόνας δοχείων: Το σημείο αυτό αφορά τον έλεγχο (του αρχείου κατασκευής), την παραγωγή και τη σάρωση της εικόνας (δοχείων της εφαρμογής). Μετά τη δημιουργία της, η εικόνα της εφαρμογής ελέγχεται για ευπάθειες και κακόβουλο λογισμικό. Τα ζητήματα ασφάλειας σε αυτή την περίπτωση μπορεί να οφείλονται στην εικόνα βάσης, σε άλλα εργαλεία ή βιβλιοθήκες που ενσωματώνονται στην εικόνα ή σε σφάλματα διαμόρφωσης (configuration errors). Πέρα από την εικόνα (image) της εφαρμογής, σαρώνεται και ολόκληρο το σύστημα αρχείων (file system) της προκειμένου να εντοπιστούν μολυσμένα ή κακόβουλα αρχεία.

4. Φάση Υπογραφής και Ασφαλούς Μεταφόρτωσης Εικόνας Δοχείων: Πρωτού η εικόνα προωθηθεί στο αποθετήριο εικόνων (στο Σύστημα Διαχείρισης Εκδόσεων Εικόνων), υπογράφεται έτσι ώστε να διασφαλιστεί η ακεραιότητα της.

5. Φάση Επικύρωσης Υπογραφής: Στη συνέχεια, αφού η εικόνα μαζί με την υπογραφή της έχει προωθηθεί στο αποθετήριο ελέγχεται η υπογραφή της τοπικά προκειμένου να εξασφαλιστεί πως δεν άλλαξε και άρα να διασφαλιστεί η ακεραιότητά της.

3.4 Υλοποίηση

3.4.1 Υλοποίηση Συστήματος

Για τις ανάγκες της παρούσας εργασίας, δημιουργήθηκε ένα οργανωμένο σύστημα, το οποίο υποστηρίζει την εκτέλεση του DevSecOps CI/CD αγωγού, ο οποίος περιγράφεται αναλυτικά στην ενότητα 3.4.2. Η δημιουργία του συστήματος αυτού, όπως θα αναφερθεί στη

συνέχεια, στηρίχθηκε στην κατάλληλη επιλογή και ενοποίηση υπαρχόντων υλοποιήσεων για τα κύρια συστατικά της αρχιτεκτονικής του συστήματος, όπως αυτή παρουσιάστηκε στην Ενότητα 3.3.1.

Οι επιλογές για την υλοποίηση των συστατικών του συστήματος είναι οι εξής:

- **Πλατφόρμα Διαχείρισης Εκδόσεων:** Επιλέχθηκε το GitHub, καθώς παρέχει ένα αξιόπιστο, διαδεδομένο και ευρέως χρησιμοποιούμενο περιβάλλον για την αποθήκευση, παρακολούθηση και διαχείριση του κώδικα, με ενσωματωμένη υποστήριξη για συνεργατική ανάπτυξη. Το σύστημα διαχείρισης και εκδόσεων κώδικα αποτελεί βασικό και απαραίτητο συστατικό της υλοποίησης καθότι χρησιμοποιείται για την αποθήκευση και διαχείριση των εκδόσεων του κώδικα προς έλεγχο. Επιπλέον, δημιουργεί και προωθεί γεγονότα (όπως commits) στην πλατφόρμα CI/CD εξασφαλίζοντας την πυροδότηση της εκκίνησης αγωγών CI/CD (όπως τον υλοποιημένο) εφόσον αυτό είναι απαραίτητο.

- **Πλατφόρμα CI/CD:** Επιλέχθηκε το GitHub Actions λόγω της άμεσης ενσωμάτωσής του με το GitHub, της ευελιξίας που προσφέρει στη δημιουργία αυτοματοποιημένων ροών εργασιών (workflows) και της δυνατότητάς του να εκτελεί δοχεία (containers) χωρίς την ανάγκη πρόσθετης υποδομής από την πλευρά του χρήστη ή του οργανισμού. Όπως φαίνεται και στον Πίνακα 2, το GitHub Actions υπερτερεί έναντι άλλων εργαλείων, καθώς επιτρέπει τη διαχείριση τόσο του πηγαίου κώδικα όσο και των αγωγών συνεχούς ολοκλήρωσης και παράδοσης (CI/CD pipelines) μέσα από μία ενιαία πλατφόρμα, διευκολύνοντας έτσι τη συνολική διαδικασία ανάπτυξης. Επιπλέον, διαθέτει πλούσιο κατάστημα (marketplace) με χιλιάδες διαθέσιμες ενέργειες (actions) που καλύπτουν ποικίλες ανάγκες αυτοματοποίησης, χωρίς να απαιτείται η ανάπτυξη προσαρμοσμένων προσθηκών (custom plugins). Υποστηρίζει εγγενώς τη χρήση δοχείων (containers) και υποδομής ως κώδικα (infrastructure as code), διευκολύνοντας τη διαχείριση σύγχρονων πρακτικών DevOps. Σε αντίθεση με άλλα εργαλεία, όπως το Jenkins, δεν απαιτεί εγκατάσταση ή διαχείριση πρόσθετης υποδομής, μειώνοντας το λειτουργικό κόστος και την πολυπλοκότητα. Τέλος, το GitHub Actions υποστηρίζει αποτελεσματικά τη δημιουργία και εκτέλεση αγωγών για συνεχή ενσωμάτωση (continuous integration), συνεχή παράδοση (continuous delivery) και έλεγχο ποιότητας του κώδικα, βελτιώνοντας τη σταθερότητα και αποδοτικότητα της ανάπτυξης. Συνολικά, το GitHub Actions επιλέχθηκε καθώς προσφέρει τον βέλτιστο συνδυασμό ευκολίας ενσωμάτωσης, αυτοματοποίησης και επεκτασιμότητας, χωρίς την ανάγκη πρόσθετης υποδομής, για έργα που φιλοξενούνται στο GitHub, όπως αποτυπώνεται και στον Πίνακα 2.

- **Σύστημα Αναφορών και Παρακολούθησης:** Επειδή το GitHub Actions προσφέρει ενσωματωμένους μηχανισμούς καταγραφής (logs), ειδοποιήσεων και παρακολούθησης της εκτέλεσης των αγωγών, δεν απαιτήθηκε η επιλογή και ενσωμάτωση ξεχωριστού συστήματος προς αυτό τον σκοπό. Μέσω των υπαρχόντων σχετικών δυνατοτήτων του GitHub Actions, παρέχεται άμεση ορατότητα στην κατάσταση των αγωγών, επιτρέποντας την έγκαιρη ανίχνευση και αντιμετώπιση σφαλμάτων ή αποτυχιών.

- **Σύστημα Άμεσης Ενημέρωσης:** Για την επικοινωνία της ομάδας (εργασίας του λογισμικού) και την άμεση αποστολή ειδοποιήσεων επιλέχθηκε το Discord[2], καθώς προσφέρει εύκολη ενσωμάτωση με το GitHub Actions μέσω webhooks[3], ενώ παράλληλα είναι ένα πολύ δημοφιλές εργαλείο, το οποίο χρησιμοποιείται ήδη από πολλές ομάδες προγραμματιστών. Με αυτόν τον τρόπο, η ομάδα εργασίας λαμβάνει σε πραγματικό χρόνο ενημερώσεις για την εκτέλεση των αγωγών / ροών εργασιών (workflows), διευκολύνοντας την έγκαιρη αντίδραση σε σφάλματα ή κρίσιμα γεγονότα.

- **Σύστημα Διαχείρισης Εκδόσεων Εικόνων:** Για την αποθήκευση και διαχείριση των εικόνων επιλέχθηκε το Docker Hub[4], λόγω της διαλειτουργικότητάς του με το οικοσύστημα της τεχνολογίας Docker, της δημοφιλίας του και της ευρείας υποστήριξής του από εργαλεία ανάπτυξης. Η χρήση του επιτρέπει την εύκολη διάθεση και ανάκτηση εικόνων, διευκολύνοντας την αυτοματοποίηση της ανάπτυξης και την ενοποίηση με αγωγούς CI/CD.

3.4.2 Υλοποίηση DevSecOps CI/CD Αγωγού

Για τις ανάγκες της παρούσας εργασίας αναπτύχθηκε ένας DevSecOps αγωγός CI/CD, ο οποίος υλοποιήθηκε και δοκιμάστηκε με βάση μια υποτυπώδη εφαρμογή ιστού (web application). Η εφαρμογή αυτή υλοποιήθηκε χρησιμοποιώντας τη γλώσσα προγραμματισμού Java (έκδοση 17), το εργαλείο Maven (έκδοση 3.8.1) για τη διαχείριση εκδόσεων και κατασκευής και το πλαίσιο (framework) Spring Boot (έκδοση 3.1.5) (που εξειδικεύεται στην ανάπτυξη εφαρμογών ιστού). Με βάση την εφαρμογή αυτή, δημιουργήθηκε ένας επαναχρησιμοποιήσιμος DevSecOps αγωγός, ο οποίος πραγματοποιεί όλες τις δυνατές δοκιμές (παραδοσιακές & ασφάλειας) πάνω στην εφαρμογή αυτή. Μάλιστα, ο κατασκευασμένος αγωγός είναι κατάλληλος για παρόμοιες Spring Boot εφαρμογές, δηλ. με εφαρμογές που αναπτύσσονται με τις προαναφερόμενες τεχνολογίες.

Ο αγωγός, ο οποίος περιγράφεται παρακάτω μπορεί να εκτελείται μόνο μέσω του Github, χρησιμοποιώντας τη πλατφόρμα Github Actions. Για την υποστήριξη άλλων πλατφορμών CI/CD, το περιεχόμενο του αγωγού θα πρέπει να μεταφραστεί στη γλώσσα (περιγραφής αγωγών) που υποστηρίζεται από τις πλατφόρμες αυτές.

Ο αγωγός σε υψηλό επίπεδο διαρθρώνεται όπως φαίνεται παρακάτω. Το ολοκληρωμένο μοντέλο του αγωγού μπορεί να βρεθεί στο Παράρτημα 1 της παρούσας εργασίας.

1. **Φάση κατασκευής & λειτουργικών δοκιμών (code build and functional testing)**

- **Εκφόρτωση Κώδικα από το Αποθετήριο**

Προκειμένου να ξεκινήσει ο έλεγχος του κώδικα από τον αγωγό απαιτείται να εκφορτωθεί ο κώδικας από το αποθετήριο. Αυτό συμβαίνει κάθε φορά που νέος κώδικας προωθείται (push/commit) εντός του αποθετηρίου.

[2] https://discord.com/
[3] Ένα webhook είναι μια ελαφριά, βασισμένη σε γεγονότα επικοινωνία που αποστέλλει αυτόματα δεδομένα μεταξύ εφαρμογών μέσω HTTP.
[4] https://hub.docker.com/

- **Έλεγχος Μονάδων (Unit Tests)**

Στο στάδιο αυτό του αγωγού, εκτελούνται δοκιμές μονάδων για τον υπό εξέταση κώδικα, με στόχο την επαλήθευση της αναμενόμενης λειτουργικότητάς του, όπως αυτή έχει καθοριστεί από τις απαιτήσεις του λογισμικού. Ανεξάρτητα από την επιτυχία ή την αποτυχία των δοκιμών, ο αγωγός δημιουργεί και αποθηκεύει εντός της πλατφόρμας Github Actions την αντίστοιχη αναφορά ως ένα τέχνημα (artifact), ενώ παράλληλα την κοινοποιεί στους προγραμματιστές μέσω του συστήματος άμεσης ενημέρωσης προγραμματιστών (Discord), ενημερώνοντάς τους σε περίπτωση αποτυχίας σε αυτό το στάδιο.

- **Δοκιμές Ενσωμάτωσης (Integration Tests)**

Στη συνέχεια, ο αγωγός προχωράει στην εκτέλεση των δοκιμών ενσωμάτωσης (Integration Tests), όπως αυτές έχουν καθοριστεί από τις απαιτήσεις του λογισμικού. Ανεξάρτητα από την επιτυχία ή την αποτυχία των δοκιμών, ο αγωγός δημιουργεί και αποθηκεύει εντός της πλατφόρμας Github Actions την αντίστοιχη αναφορά ως ένα τέχνημα (artifact), ενώ παράλληλα την κοινοποιεί στους προγραμματιστές μέσω του συστήματος άμεσης ενημέρωσης προγραμματιστών (Discord), ενημερώνοντάς τους σε περίπτωση αποτυχίας σε αυτό το στάδιο.

2. **Στατική Ανάλυση Ασφάλειας (Static Security Analysis)**

- **Στατικός Έλεγχος (Static Test)**

Προκειμένου να πραγματοποιηθεί η στατική ανάλυση του κώδικα χρησιμοποιείται το εργαλείο SpotBugs[5] και συγκεκριμένα η εντολή mvn spotbugs:check. Απαραίτητη προϋπόθεση για την χρήση του εργαλείου αυτού είναι το αρχείο διαχείρισης έργου (pom.xml) της εκάστοτε εφαρμογής στο Maven να έχει ρυθμιστεί ώστε να το περιλαμβάνει στο μέρος των προσθέτων Maven της κατασκευής.

Το SpotBugs επιλέχθηκε επειδή αποτελεί ένα αποτελεσματικό εργαλείο στατικής ανάλυσης κώδικα, ικανό να εντοπίζει πιθανά σφάλματα και ευπάθειες σε εφαρμογές γραμμένες στη γλώσσα προγραμματισμού Java.

Ο στατικός έλεγχος, αποτελεί μία μορφή ελέγχου ασφάλειας όπου ο κώδικας αναλύεται χωρίς να εκτελείται.

Η στατική ανάλυση ανιχνεύει τα εξής:

- **Λογικά Σφάλματα:** Το εργαλείο SpotBugs είναι σχεδιασμένο να αναζητά και να ανιχνεύει λογικά σφάλματα. Λογικό σφάλμα μπορεί να θεωρηθεί η δήλωση μιας μεταβλητής που δεν χρησιμοποιείται στη συνέχεια ή η ύπαρξη σημείων στο κώδικα που δεν εκτελούνται ποτέ λόγω εσφαλμένης λογικής (συνθήκη αν - if που δεν οδηγείται ποτέ στο "αλλιώς" μέρος - else).
- **Προβλήματα Ασφάλειας:** Το εργαλείο ελέγχει τον κώδικα για γνωστές ευπάθειες, όπως την χρήση μη ασφαλών βιβλιοθηκών ή την διαχείριση μυστικών με μη ασφαλή τρόπο (πχ. μυστικά μέσα στον ίδιο τον κώδικα σε μη κρυπτογραφημένη μορφή - plaintext).
- **Ποιότητα Κώδικα:** Κατά τη διάρκεια της στατικής ανάλυσης, ο κώδικας αναλύεται προκειμένου να εντοπιστούν σημεία στα οποία δεν ακολουθούνται βέλτιστες πρακτικές (best practises) ανάπτυξης.

[5] https://spotbugs.github.io/

- **Διαχείριση Πόρων:** Το SpotBugs έχει την ικανότητα να εντοπίζει περιπτώσεις όπου αντικείμενα δεν αποδεσμεύονται με σωστό τρόπο, σπαταλώντας πόρους μνήμης. Ακόμη ανιχνεύει περιπτώσεις κακής χρήσης συγχρονισμού (πχ. σε νήματα - threads), και μη κλεισίματος πόρων που έχουν ανοίξει (πχ FileReader - αντικείμενο ανάγνωσης αρχείων - που ανοίγει ένα αρχείο αλλά δεν το κλείνει ποτέ).
- **Προβλήματα Απόδοσης:** Ακόμη, στη στατική ανάλυση, το SpotBugs μπορεί να ανιχνεύσει περιπτώσεις όπου δημιουργούνται αντικείμενα χωρίς να είναι απαραίτητο.

Το εργαλείο δίνει τη δυνατότητα να οριστούν τα όρια στα οποία η εκτέλεση του αποτυγχάνει. Συγκεκριμένα, η στατική ανάλυση αποτυγχάνει όταν εντοπίζει **τουλάχιστον ένα** (1) ζήτημα ασφάλειας ή ποιότητας του οποίου η σοβαρότητα είναι **υψηλή** (high) (στην κλίμακα low - medium - high).

Ανεξάρτητα από την επιτυχία ή την αποτυχία του βήματος, ο αγωγός δημιουργεί και αποθηκεύει την αντίστοιχη αναφορά ως ένα ξεχωριστό artifact (τέχνημα), ενώ παράλληλα την κοινοποιεί στους προγραμματιστές, ενημερώνοντάς τους σε περίπτωση αποτυχίας σε αυτό το στάδιο.

- **Ανάλυση Εξαρτήσεων (Dependency Analysis)**

Στη συνέχεια, ο αγωγός προχωράει στην ανάλυση εξαρτήσεων (dependency analysis) με την χρήση του εργαλείου Trivy[6]. Το Trivy επιλέχθηκε διότι είναι ευρέως διαδεδομένο, προσφέρει έναν γρήγορο και αποτελεσματικό τρόπο εντοπισμού ευπαθειών σε εικόνες δοχείων και σε εξαρτήσεις (dependencies) ενώ παράλληλα πραγματοποιεί και ανίχνευση μυστικών (secrets detection) (δηλαδή είναι ένα πολυ-εργαλείο τρία σε ένα). Χάρη στην ευκολία χρήσης και την εκτενή κάλυψη ευπαθειών, βοηθά στη διασφάλιση της ασφάλειας του λογισμικού από τα πρώτα στάδια ανάπτυξης, μειώνοντας πιθανές ευπάθειες.

Σε αυτό το βήμα εκτελείται ένας ενδελεχής έλεγχος στις εξαρτήσεις που χρησιμοποιούνται από τον κώδικα, οι οποίες εξετάζονται για γνωστές ευπάθειες. Με τον όρο γνωστές ευπάθειες, περιγράφονται ευπάθειες για τις οποίες υπάρχει καταγεγραμμένο CVE αναγνωριστικό (κωδικός ευπάθειας).

Ο αγωγός είναι σχεδιασμένος με τέτοιο τρόπο ώστε να οδηγείται σε κατάσταση αποτυχίας (fail), όταν εντοπιστεί **τουλάχιστον ένα** (1) ζήτημα ασφάλειας, του οποίου η σοβαρότητα είναι **υψηλή** (high) ή μεγαλύτερη (δηλ. κριτική - critical), στη κλίμακα (low, medium, high, critical).

Ανεξάρτητα από την επιτυχία ή την αποτυχία του βήματος, ο αγωγός δημιουργεί και αποθηκεύει την αντίστοιχη αναφορά ως ένα ξεχωριστό artifact, ενώ παράλληλα την κοινοποιεί στους προγραμματιστές, ενημερώνοντάς τους σε περίπτωση αποτυχίας σε αυτό το στάδιο.

- **Ανίχνευση Μυστικών (Secrets Detection)**

Στο βήμα αυτό εκτελείται ένας έλεγχος εντοπισμού μυστικών (Secrets Detection) χρησιμοποιώντας το εργαλείο Trivy. Όπως αναφέρθηκε παραπάνω, το Trivy είναι ένα εργαλείο που πραγματοποιεί με αποδοτικό τρόπο ανάλυση εξαρτήσεων και ανίχνευση μυστικών. Αυτό συνεπάγεται πως για τον τρέχοντα έλεγχο ασφάλειας δεν χρειάζεται να χρησιμοποιήσουμε ένα διαφορετικό εργαλείο.

[6] https://trivy.dev/latest/

Το εργαλείο είναι σχεδιασμένο να ανιχνεύει την ύπαρξη μυστικών, τα οποία έχουν καταγραφεί σε οποιοδήποτε αρχείο της εφαρμογής. Το Trivy αναζητά και εντοπίζει μυστικά όπως: κωδικούς πρόσβασης, access tokens, κλειδιά για APIs αλλά και μυστικά που αντιστοιχούν σε κάποιο κοινό πρότυπο (πχ. Github Tokens).

Σε περίπτωση που εντοπιστεί τουλάχιστον ένα (1) καταγεγραμμένο μυστικό, ο αγωγός αποτυγχάνει και η αναφορά που παράγεται (είτε σε περίπτωση αποτυχίας είτε σε περίπτωση επιτυχίας) περιλαμβάνει λεπτομέρειες αναφορικά με το μυστικό που εντοπίστηκε, αλλά και συστάσεις για την διαχείριση των μυστικών. Η αναφορά αυτή αποθηκεύεται ως ένα ξεχωριστό τέχνημα (artifact) ενώ παράλληλα κοινοποιείται στους προγραμματιστές, ενημερώνοντάς τους σε περίπτωση αποτυχίας σε αυτό το στάδιο.

3. Φάση παραγωγής και ελέγχου εικόνας δοχείων

- **Ανάλυση dockerfile με το εργαλείο Hadolint**

Στη συνέχεια, χρησιμοποιείται το εργαλείο Hadolint[7] προκειμένου να ελέγξει το Dockerfile της εφαρμογής (δηλ. το αρχείο κατασκευής της εικόνας δοχείων της εφαρμογής) για βέλτιστες πρακτικές και προβλήματα που ενδεχομένως υπάρχουν.

Το Hadolint επιλέχθηκε καθώς είναι ευρέως διαδεδομένο, ενοποιείται εύκολα με το GitHub Actions, υποστηρίζει πολλαπλές μορφοποιήσεις αναφορών εξόδου ενώ αποτελεί ένα αξιόπιστο εργαλείο ειδικά κατασκευασμένο για την ανάλυση Dockerfiles, διασφαλίζοντας ότι ακολουθούνται βέλτιστες πρακτικές και αποφεύγονται κοινά λάθη. Με τη χρήση του, μειώνονται οι πιθανότητες ευπαθειών και βελτιστοποιείται η απόδοση των εικόνων δοχείων (Docker), συμβάλλοντας σε μια πιο ασφαλή και αποδοτική διαδικασία ανάπτυξης.

Σε περίπτωση που το εργαλείο εντοπίσει τουλάχιστον ένα (1) πρόβλημα, το οποίο έχει χαρακτηριστεί ως error (στην κλίμακα info - style - warning - error), αυτό το βήμα και ολόκληρος ο αγωγός αποτυγχάνουν.

Ανεξάρτητα από την επιτυχία ή την αποτυχία του βήματος, ο αγωγός δημιουργεί και αποθηκεύει την αντίστοιχη αναφορά ως ξεχωριστό artifact, ενώ παράλληλα την κοινοποιεί στους προγραμματιστές, ενημερώνοντάς τους σε περίπτωση αποτυχίας σε αυτό το στάδιο.

Το συγκεκριμένο βήμα υλοποιήθηκε με ρητή λήψη και εκτέλεση του εργαλείου Hadolint και όχι τη χρήση της εικόνας (image) δοχείων του. Αυτό συνέβη λόγω ελλιπούς βιβλιογραφίας και πολλών σφαλμάτων (bugs) που συνόδευαν την χρήση της εικόνας δοχείων (container image) για το εργαλείο αυτό.

- **Δημιουργία και εκτέλεση εικόνας**

Στο βήμα αυτό δημιουργείται (build) και εκτελείται (run) η εικόνα του δοχείου (container image) της Spring Boot εφαρμογής. Το βήμα αυτό είναι απαραίτητο για την συνέχεια της τρέχουσας ενότητας βημάτων της τρέχουσας φάσης.

- **Δυναμικός έλεγχος ασφάλειας με το εργαλείο Arachni (Arachni Scan)**

Στο βήμα αυτό χρησιμοποιείται το εργαλείο Arachni[8] προκειμένου να σαρώσει την Spring Boot εφαρμογή για διαδικτυακές ευπάθειες. Ειδικότερα, χρησιμοποιώντας τον σύνδεσμο (url) **http://application-container:8080/books** όπου είναι διαθέσιμη η εφαρμογή**,** το εργαλείο

[7] https://hub.docker.com/r/hadolint/hadolint
[8] https://github.com/Arachni/arachni

πραγματοποιεί σάρωση για γνωστές ευπάθειες (όπως SQL injection, XSS κλπ). Το εργαλείο αυτό επιλέχθηκε διότι είναι αρκετά διαδεδομένο & εξεζητημένο, διαμορφώσιμο και εύκολο στη χρήση.

Σε περίπτωση που εντοπιστούν ευπάθειες σοβαρότητας χαμηλής (low) ή μεγαλύτερης (στην κλίμακα informational - low - medium - high) από την σάρωση του εργαλείου, ο αγωγός επιστρέφει μήνυμα εξόδου 1 και αποτυγχάνει. Εναλλακτικά, συνεχίζει στο επόμενο βήμα.

Ο λόγος που το σημείο αποτυχίας (threshold), τέθηκε στην χαμηλή σοβαρότητα (low), είναι το ότι αυτό το αυστηρό όριο συμβάλλει στην πρόληψη της συσσώρευσης μικρών ευπαθειών, ενισχύει την κουλτούρα "ασφάλεια-πρώτα", ("security-first") και διασφαλίζει τη συμμόρφωση με κανονιστικά πλαίσια, ελαχιστοποιώντας πιθανούς κινδύνους στο μέλλον. Παράλληλα, οι ευπάθειες ιστού, θεωρούνται από πολλούς οι πιο εύκολες προς εκμετάλλευση, συνεπώς δημιουργείται ανάγκη για πιο αυστηρούς περιορισμούς.

Ανεξάρτητα από την επιτυχία ή την αποτυχία του βήματος, ο αγωγός δημιουργεί και αποθηκεύει την αντίστοιχη αναφορά ως ένα ξεχωριστό artifact, ενώ παράλληλα την κοινοποιεί στους προγραμματιστές, ενημερώνοντάς τους σε περίπτωση αποτυχίας σε αυτό το στάδιο.

- **Τερματισμός Εφαρμογής**

Στη συνέχεια, αφού ολοκληρωθεί ο δυναμικός έλεγχος της εφαρμογής με το εργαλείο Arachni, η εφαρμογή τερματίζει (δηλ. σταματάει η εκτέλεση του δοχείου της εφαρμογής).

- **Σάρωση Ευπαθειών Εικόνας (Scan Vulnerabilities in Container Image)**

Στο επόμενο βήμα χρησιμοποιείται ξανά το εργαλείο Trivy, αυτή τη φορά προκειμένου να πραγματοποιήσει έλεγχο ευπαθειών στην εικόνα (image) του δοχείου Docker της Spring Boot εφαρμογής μας. Κατά τη διάρκεια αυτής της διαδικασίας, το εργαλείο αποκτά πρόσβαση στην εικόνα και πραγματοποιεί σάρωση για ευπάθειες τουλάχιστον μεσαίας (medium) σοβαρότητας ή μεγαλύτερης, στη κλίμακα low, medium, high, critical.

Ωστόσο, ο αγωγός είναι σχεδιασμένος με τέτοιο τρόπο ώστε να οδηγείται σε κατάσταση αποτυχίας (fail), όταν εντοπιστεί **τουλάχιστον ένα** (1) ζήτημα ασφάλειας του οποίου η σοβαρότητα είναι **μεσαία** (medium) ή μεγαλύτερη. Ο λόγος για τον οποίο το όριο τέθηκε τόσο χαμηλά έγκειται στο γεγονός ότι οι ευπάθειες που αφορούν εικόνες δύναται να αποδειχθούν ιδιαιτέρως επικίνδυνες ακόμη και σε περιπτώσεις όπου η σοβαρότητα τους έχει χαρακτηριστεί μεσαία.

Ανεξάρτητα από την επιτυχία ή την αποτυχία του αγωγού σε αυτό το βήμα, ο αγωγός δημιουργεί και αποθηκεύει την αντίστοιχη αναφορά ως ένα ξεχωριστό artifact, ενώ παράλληλα την κοινοποιεί στους προγραμματιστές, ενημερώνοντάς τους σε περίπτωση αποτυχίας σε αυτό το στάδιο.

- **Σάρωση Κακόβουλου Λογισμικού στην Εικόνα (Malware Scan in Container Image)**

Στη συνέχεια, πραγματοποιείται έλεγχος για κακόβουλο λογισμικό στο σύστημα αρχείων της εικόνας (Docker image), χρησιμοποιώντας το εργαλείο ClamAV[9]. Για να επιτευχθεί αυτό, το σύστημα αρχείων της εικόνας εξάγεται και συσκευάζεται σε ένα συμπιεσμένο αρχείο .tar, το οποίο στη συνέχεια αποσυμπιέζεται από το εργαλείο ClamAV για τη σάρωση (από το ίδιο) των περιεχόμενων αρχείων.

Ο σκοπός αυτής της διαδικασίας είναι η ανίχνευση πιθανών μολυσμένων ή κακόβουλων

[9] https://www.clamav.net/

αρχείων που μπορεί να υπάρχουν στην εικόνα. Η δημιουργία του .tar αρχείου είναι ένα απαραίτητο ενδιάμεσο στάδιο, καθώς επιτρέπει στο ClamAV να έχει πρόσβαση και να ελέγχει το σύνολο των αρχείων της εικόνας.

Σε περίπτωση που εντοπιστούν κακόβουλα αρχεία (exit code 1), ή προκύψει κάποιο σφάλμα (exit code > 1), ο αγωγός αποτυγχάνει και ο λόγος αποτυχίας αποτυπώνεται στην αναφορά που παράγεται.

Η αναφορά αποθηκεύεται ως ένα ξεχωριστό artifact, ενώ παράλληλα κοινοποιείται στους προγραμματιστές, ενημερώνοντάς τους σε περίπτωση αποτυχίας σε αυτό το στάδιο.

Σημειώνουμε σε αυτό το σημείο πως το ClamAV είναι το μοναδικό εργαλείο ανοικτού κώδικα που μπορεί να καλύψει εκτενώς πολλά είδη κακόβουλου λογισμικού. Άλλα εργαλεία (πχ. ακόμη και το Trivy) είτε ανιχνεύουν ένα πολύ μικρό εύρος ειδών κακόβουλου λογισμικού είτε χρησιμοποιούν εσωτερικά το εργαλείο αυτό προς αυτό τον σκοπό (όπως το Dagda).

4. Φάση Υπογραφής και Ασφαλούς Μεταφόρτωσης Εικόνας Δοχείων

- **Υπογραφή Εικόνας (Sign Image)**

Στο βήμα αυτό, η εικόνα δοχείων της εφαρμογής (Docker image) υπογράφεται, χρησιμοποιώντας το εργαλείο Cosign[10]. Το Cosign επιλέχθηκε ως το πιο διαδεδομένο εργαλείο για την υπογραφή των εικόνων, καθώς προσφέρει έναν ασφαλή και εύχρηστο τρόπο επαλήθευσης της αυθεντικότητάς τους (σε σχέση με τον ανταγωνισμό) χωρίς να απαιτείται τροποποίηση της ίδιας της εικόνας. Με αυτόν τον τρόπο, ενισχύεται η ασφάλεια στην εφοδιαστική αλυσίδα λογισμικού, διασφαλίζοντας ότι οι εικόνες παραμένουν αξιόπιστες και αδιάβλητες. Παράλληλα, στο ίδιο βήμα πραγματοποιείται και έλεγχος/επικύρωση της υπογραφής της εικόνας προκειμένου να διασφαλιστεί η ακεραιότητα της - υπάρχει το ενδεχόμενο η υπογραφή της εικόνας να μην ταιριάζει με το περιεχόμενο της εικόνας, πράγμα το οποίο θα είναι επικίνδυνο από πλευράς ασφάλειας εφόσον η εικόνα έπειτα προωθηθεί σε σχετικό αποθετήριο.

Σκοπός της υπογραφής είναι να επιβεβαιώνει την αυθεντικότητα και ακεραιότητα της εικόνας, προκειμένου να μην είναι δυνατό να διαστρεβλωθεί από κάποιον επιτιθέμενο.

Για την υπογραφή της εικόνας χρησιμοποιείται το κλειδί COSIGN_PASSWORD, το οποίο βρίσκεται αποθηκευμένο στις μεταβλητές περιβάλλοντος του αγωγού. Αναλυτική περιγραφή των μεταβλητών περιβάλλοντος του αγωγού εντοπίζεται στο Κεφάλαιο 4. Σημειώνεται πως το Cosign υποστηρίζει και ειδικές διαμορφώσεις όπου δεν απαιτείται η χρήση κλειδιών.

Το συγκεκριμένο βήμα υλοποιήθηκε με ρητή λήψη και εκτέλεση του εργαλείου Cosign και όχι χρήση της εικόνας (image) δοχείων του. Αυτό συνέβη λόγω ελλιπούς βιβλιογραφίας και πολλών σφαλμάτων (bugs) που συνόδευαν την χρήση της εικόνας δοχείων (container image) για το εργαλείο αυτό.

- **Προώθηση Εικόνας (Push Image)**

Αν όλα τα προηγούμενα βήματα έχουν ολοκληρωθεί επιτυχώς, η εικόνα (Docker image), η οποία έχει παραχθεί, προωθείται (push) στο αποθετήριο εικόνων (Docker Hub). Για την πραγματοποίηση αυτού του βήματος χρησιμοποιούνται κατάλληλα διαπιστευτήρια, τα οποία περιγράφονται αναλυτικά στο Κεφάλαιο 4.

- **Επικύρωση Υπογραφής (Signature Verify)**

[10]https://docs.sigstore.dev/quickstart/quickstart-cosign/#:~:text=Cosign%20is%20a%20command%20line,tool%20for%20signing%20and%20verifying.

Τέλος, αφού η εικόνα έχει προωθηθεί (push) στο αποθετήριο εικόνων (Docker Hub), γίνεται εκ νέου λήψη της (pull), και στην συνέχεια πραγματοποιείται επικύρωση της υπογραφής (cosign verify) της. Σκοπός του ελέγχου αυτού είναι να ανιχνευθεί ενδεχόμενη τροποποίηση της εικόνας είτε κατά την μεταφορά της είτε κατά την αποθήκευσή της στο αποθετήριο εικόνων.

3.5 Ικανοποίηση Απαιτήσεων

3.5.1 Ικανοποίηση Μη Λειτουργικών Απαιτήσεων

Πίνακας 3. Βαθμός Ικανοποίησης Μη Λειτουργικών Απαιτήσεων

Μη Λειτουργική Απαίτηση	Βαθμός Ικανοποίησης (1-5)	Σχόλια
Διαθεσιμότητα	5/5	Το σύστημα προσφέρεται μέσω της πλατφόρμας Github Actions, η οποία προσφέρει διαθεσιμότητα (uptime) με ποσοστό 99.9%
Ευελιξία διαμόρφωσης & Επαναχρησιμοποίηση	5/5	Ο αγωγός είναι διαρθρωμένος με τέτοιο τρόπο ώστε τα βήματα του να μπορούν να αντιγραφούν και να επαναχρησιμοποιηθούν σε έναν νέο αγωγό.
Απόδοση	4/5	Σχεδόν όλα τα εργαλεία τρέχουν μέσω δοχείων (containers), γεγονός που ελαχιστοποιεί όσο το δυνατόν περισσότερο τον χρόνο εκτέλεσης του αγωγού (αντιθέτως η εγκατάσταση και εκτέλεση εργαλείων μαζί με τις εξαρτήσεις τους μπορεί να οδηγεί σε μεγάλες καθυστερήσεις). Ωστόσο, η ενσωμάτωση πολλαπλών βημάτων ελέγχου ασφαλείας επιβραδύνουν την εκτέλεση του αγωγού.
Κλιμακωσιμότητα	4/5	Το σύστημα υποστηρίζει την κλιμάκωση, διασφαλίζοντας σε μεγάλο βαθμό σταθερή απόδοση ανεξαρτήτως φόρτου. Ωστόσο, αυτό εξαρτάται από τους περιορισμούς που θέτει η πλατφόρμα Github Actions.
Ασφάλεια	5/5	Το σύστημα στο οποίο εκτελείται ο αγωγός προστατεύεται με χρήση πιστοποίησης δύο παραγόντων (2FA) ενώ χρησιμοποιούνται οι νεότερες

		εκδόσεις σε όλα τα λογισμικά. Παράλληλα τα μυστικά του συστήματος αποθηκεύονται με ασφαλή τρόπο εντός του συστήματος Github Actions και διοχετεύονται στον αγωγό με ασφαλή τρόπο ως μεταβλητές περιβάλλοντος.
Αξιοπιστία	5/5	Το σύστημα προσφέρει εξαιρετική σταθερότητα και ανάκαμψη από καταστροφές. Επιτρέπει την επανεκτέλεση εργασιών που απέτυχαν και παρέχει αναλυτικές αναφορές σε κάθε περίπτωση.

3.5.2 Ικανοποίηση Λειτουργικών Απαιτήσεων

Πίνακας 4. Βαθμός Ικανοποίησης Λειτουργικών Απαιτήσεων

Λειτουργική Απαίτηση	**Βαθμός Ικανοποίησης (1-5)**	**Σχόλια**
Αυτοματοποιημένη Διαχείριση Κώδικα	5/5	Ο αγωγός εκτελεί αυτόματα όλους τους προκαθορισμένους ελέγχους (συμπεριλαμβανομένων ελέγχων ασφαλείας) μόλις γίνει κάποιο commit στο Σύστημα Διαχείρισης Εκδόσεων.
Συνεχής Ενσωμάτωση και Παράδοση	5/5	Ο αγωγός ενσωματώνει τη χρήση όλων των εργαλείων που είναι απαραίτητα για την πραγματοποίηση δοκιμών, την αποθήκευση των παραγόμενων τεχνημάτων (αναφορές, εκτελέσιμα πακέτα, εικόνες δοχείων) και την παράδοση του λογισμικού σε προκαθορισμένα περιβάλλοντα.
Έλεγχοι Ασφάλειας	4/5	Ο αγωγός καλύπτει σε μεγάλο βαθμό την ενσωμάτωση εργαλείων που διασφαλίζουν την ασφάλεια του λογισμικού. Ωστόσο, δεν υποστηρίζεται έλεγχος συμμόρφωσης (conformance checking), κάτι που αποτελεί σημαντικό περιορισμό και θα μπορούσε να εξεταστεί ως μελλοντική εργασία. Επιπλέον, υπάρχουν περιθώρια βελτίωσης όσον αφορά ελέγχους για εξεζητημένες επιθέσεις (sophisticated attacks), όπως δοκιμές διείσδυσης για ευπάθειες τύπου zero day, οι οποίες απαιτούν ανθρώπινη

		παρέμβαση και δεν μπορούν να αυτοματοποιηθούν.
Παρακολούθηση και Αναφορές	5/5	Ο αγωγός παράγει και διαθέτει αναφορές σε κάθε βήμα καθώς και παρέχει αναλυτικά μηνύματα σε περίπτωση αποτυχίας.
Διασύνδεση με Τρίτα Εργαλεία	4/5	Ο αγωγός υποστηρίζει επιτυχώς τη διασύνδεση με Docker, Docker Hub & Discord ενώ μπορεί δυνητικά να διασυνδεθεί με οποιοδήποτε εργαλείο (ειδικά εφόσον αυτό μπορεί να χρησιμοποιηθεί μέσω της Docker τεχνολογίας). Η παρούσα αξιολόγηση αποτυπώνει την υψηλή λειτουργικότητα του συστήματος, λαμβάνοντας όμως υπόψη ότι η διαλειτουργικότητα προς το παρόν περιορίζεται σε εργαλεία που βασίζονται αποκλειστικά στο Docker, χωρίς να καλύπτει ευρύτερες τεχνολογικές πλατφόρμες.

4 Εγκατάσταση και Επίδειξη

4.1 Εγκατάσταση

4.1.1 Ρύθμιση Github Actions

4.1.1.1 Δομή αρχείων

Προκειμένου να διασφαλιστεί πως ο αγωγός θα εκτελείται με τον αναμενόμενο τρόπο προτείνεται τα αρχεία στο αποθετήριο κώδικα να είναι οργανωμένα με τον εξής τρόπο:

```
.github/workflows/                  # Περιέχει αρχεία για αυτοματοποιημένες ροές εργασίας CI/CD
  ├── ci.yml                        # Ορισμός για τον CI/CD αγωγό
my-app/                             # Κύριος φάκελος της εφαρμογής, μπορεί να ονομάζεται όπως επιθυμείτε
  ├── src/                          # Πηγαίος κώδικας της εφαρμογής
  │   ├── main/java/…               # Κύριος κώδικας Java
  │   │   ├── ..
  │   └── test/java/…               # Κώδικας για δοκιμές ενσωμάτωσης και δοκιμές μονάδων
  │       └── ..
  ├── Dockerfile                    # Dockerfile για τη δημιουργία του container image
  ├── pom.xml                       # Maven build file, περιέχει εξαρτήσεις και ρυθμίσεις για την κατασκευή της εφαρμογής
  ├── cosign.pub                    #Δημόσιο κλειδί για την επικύρωση της υπογραφής
README.md                           # Τεκμηρίωση του έργου
```

4.1.1.2 Προαπαιτούμενα

Όπως αναφέρθηκε προηγουμένως, το Github Actions είναι το περιβάλλον, το οποίο επιτρέπει την εκτέλεση του CI/CD αγωγού για την Java εφαρμογή προς εξέταση. Στην παρούσα ενότητα θα περιγραφούν οι οδηγίες για την ρύθμιση του.

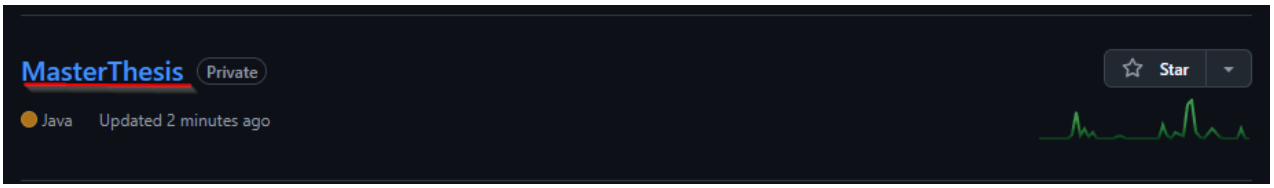
Για την ρύθμιση προς κατάλληλη χρήση του Github Actions απαιτούνται τα εξής:

1. Ένα ιδιωτικό ή δημόσιο (private or public) αποθετήριο (repository) στην πλατφόρμα Github με μία εφαρμογή που έχει γραφτεί σε γλώσσα προγραμματισμού Java, με το εργαλείο Maven και το πλαίσιο (framework) Spring Boot.
2. Δικαιώματα διαχειριστή (admin privileges) στο αποθετήριο.
3. Ένα αρχείο με κατάληξη .yml το οποίο θα χρησιμοποιηθεί για τον ορισμό του αγωγού, ο οποίος φαίνεται στο Παράρτημα 1 της παρούσας εργασίας.
4. Το κατάλληλο αρχείο Dockerfile, το οποίο θα διασφαλίζει την δημιουργία της εικόνας δοχείου για την προς εξέταση εφαρμογή. Τόσο το αρχείο Dockerfile όσο και το .yml θα πρέπει να περιλαμβάνονται στον πηγαίο κώδικα της εφαρμογής στο GitHub αποθετήριο.
5. Το αρχείο cosign.pub που θα χρησιμοποιηθεί για την επικύρωση της υπογραφής της εικόνας δοχείων της εφαρμογής που θα παραχθεί κατά τη διάρκεια εκτέλεσης του αγωγού.

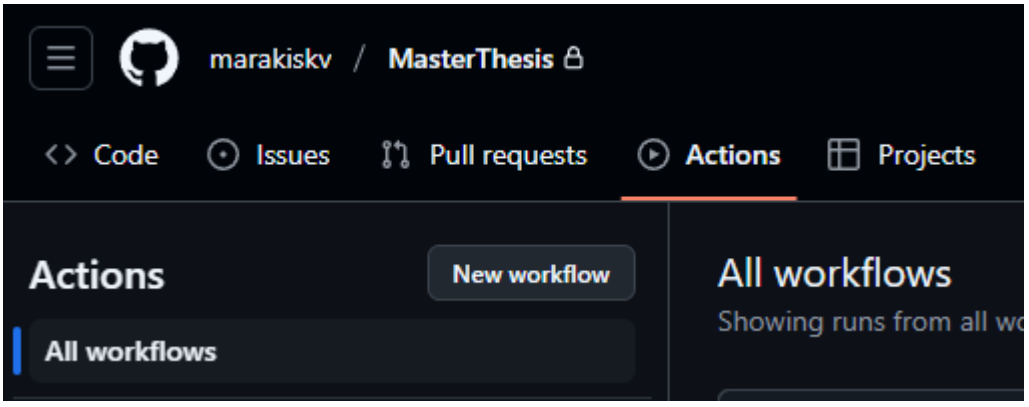
4.1.1.3 Εναλλακτική Διαδικασία Δημιουργίας Αγωγού

Αντί το αποθετήριο (του πηγαίου κώδικα) να εμπεριέχει εκ των προτέρων το αρχείο/μοντέλο περιγραφής του αγωγού, αυτός μπορεί να δημιουργεί με γραφικό τρόπο ακολουθώντας την εξής διαδικασία:

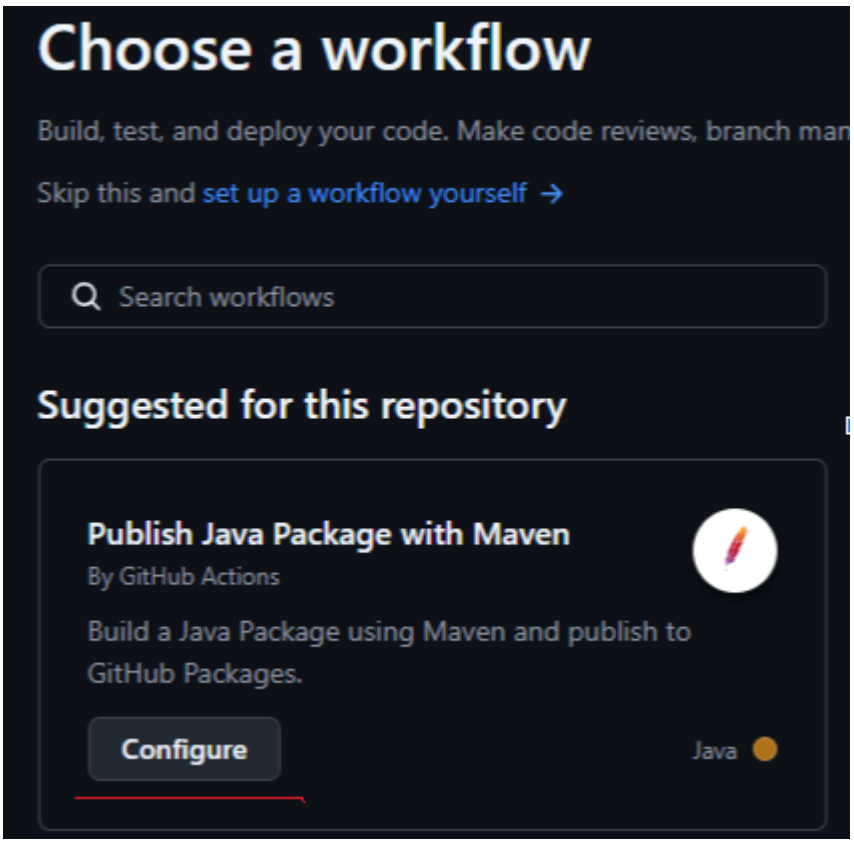
1. Μετάβαση στο σωστό αποθετήριο κώδικα στο Github (Εικόνα 2)
2. Κλικ στην καρτέλα Actions και έπειτα στο κουμπί New workflow (Εικόνα 3)
3. Επιλογή για δημιουργία νέας ροής εργασιών (workflow) (πάτημα κουμπιού Configure) (Εικόνα 4)
4. Αντιγραφή του περιεχομένου του αγωγού από το Παράρτημα 1 της παρούσας εργασίας μέσα στο .yml αρχείο του workflow.



Εικόνα 2. Μετάβαση στο σωστό αποθετήριο κώδικα στο Github



Εικόνα 3. Κλικ στην καρτέλα Actions



Εικόνα 4.Επιλογή για δημιουργία νέας ροής εργασιών (workflow)

4.1.2 Διαμόρφωση και Αποθήκευση Μυστικών (Secrets)

4.1.2.1 Είδη Μυστικών προς Χρήση

Για την ομαλή λειτουργία του αγωγού, είναι απαραίτητος ο ορισμός συγκεκριμένων μυστικών (secrets), τα οποία αποθηκεύονται στο σύστημα του GitHub Actions και χρησιμοποιούνται από τον αγωγό. Τα μυστικά αυτά περιγράφονται παρακάτω:

- **DOCKER_HUB_USERNAME**: Το όνομα χρήστη (username) για το σύστημα διαχείρισης εκδόσεων εικόνας (docker image repository). Μαζί με το **DOCKER_HUB_PASSWORD** χρησιμοποιούνται προκειμένου ο αγωγός να μπορέσει να προωθήσει την εικόνα στο αποθετήριο εκδόσεων εικόνας (docker hub).

- **DOCKER_HUB_PASSWORD:** Ο κωδικός πρόσβασης χρήστη (password) για το σύστημα διαχείρισης εκδόσεων εικόνας (docker image repository). Χρησιμοποιείται για τον σκοπό που περιγράφηκε παραπάνω.

- **DISCORD_WEBHOOK**: Το **DISCORD_WEBHOOK** αποτελεί έναν ειδικό σύνδεσμο (URL), ο οποίος επιτρέπει στον αγωγό να στέλνει αυτοματοποιημένα μηνύματα σε ένα κανάλι Discord. Στο παρόν σύστημα χρησιμοποιείται από τον αγωγό για να στέλνει ειδοποιήσεις και αποτελέσματα αναφορών στο σύστημα άμεσης ειδοποίησης (Discord) προς ενημέρωση των προγραμματιστών - αναλυτικές οδηγίες για τη διαμόρφωση και ενσωμάτωση του συστήματος αυτού παρουσιάζονται στις Ενότητες 4.1.2.2 και 4.1.2.3.

- **COSIGN_KEY:** Το **COSIGN_KEY** είναι ένα ιδιωτικό κλειδί, το οποίο χρησιμοποιείται για την υπογραφή της εικόνας δοχείων της εφαρμογής από το εργαλείο cosign και χρησιμοποιείται συνδυαστικά με το **COSIGN_PASSWORD.**

- **COSIGN_PASSWORD:** Το **COSIGN_PASSWORD** είναι ο κωδικός πρόσβασης (password), ο οποίος προστατεύει το ιδιωτικό κλειδί που χρησιμοποιείται για την υπογραφή της εικόνας.

4.1.2.2 Οδηγίες Παραγωγής Μυστικών (Secrets)

Τα μυστικά που χρησιμοποιούνται για την εκτέλεση του αγωγού προέρχονται από τρίτα συστήματα (third parties) και πρέπει να παραχθούν πρωτού αποθηκευτούν και γίνουν διαθέσιμα στον αγωγό. Συγκεκριμένα:

- **DOCKER_HUB_USERNAME** και **DOCKER_HUB_PASSWORD:** Τα δύο αυτά διαπιστευτήρια (credentials) παράγονται με την εγγραφή ενός χρήστη στο σύστημα διαχείρισης εκδόσεων εικόνας (Docker Hub repository). Ειδικότερα, μέσω της ιστοσελίδας του Docker Hub[11] μπορεί κανείς να πραγματοποιήσει εγγραφή νέου χρήστη (signup) για έναν προσωπικό λογαριασμό (personal account). Το όνομα χρήστη και ο κωδικός πρόσβασης που θα χρησιμοποιήσει κατά την εγγραφή του είναι διαπιστευτήρια, τα οποία πρέπει να αποθηκεύσει ως μυστικά με ονόματα **DOCKER_HUB_USERNAME** και **DOCKER_HUB_PASSWORD**, αντίστοιχα.

- **DISCORD_WEBHOOK:** Για την δημιουργία του **DISCORD_WEBHOOK** πρέπει κανείς να δημιουργήσει ένα νέο κανάλι στο Discord (discord channel) και στη συνέχεια μέσα σε αυτό το κανάλι να ενσωματώσει ένα webhook, το οποίο χρησιμοποιείται για την διασύνδεση με τον αγωγό. Οδηγίες και για τα δύο βήματα βρίσκονται εδώ: [13], [14].

- **COSIGN_KEY** και **COSIGN_PASSWORD:** Για την δημιουργία του ζευγαριού **COSIGN_KEY** και **COSIGN_PASSWORD**, τα οποία θα χρησιμοποιηθούν για την υπογραφή της εικόνας, μπορούν να χρησιμοποιηθούν οι οδηγίες που βρίσκονται εδώ: [15].

4.1.2.3 Οδηγίες Αποθήκευσης Μυστικών (Secrets)

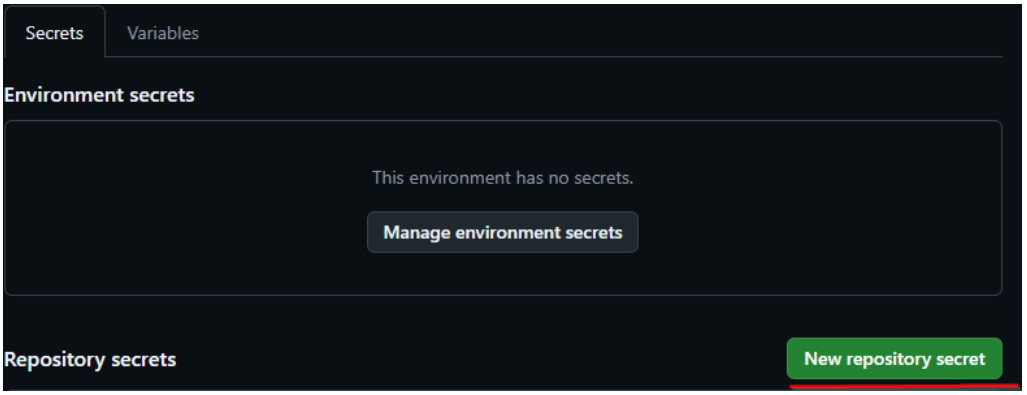
Στο περιβάλλον του Github τα μυστικά (secrets) αποθηκεύονται στο εξής μονοπάτι:

Repository Setting → Security → Secrets and variables → Actions

Αυτή είναι η τοποθεσία στην οποία αποθηκεύονται τα μυστικά που αφορούν το Github Actions και θα χρησιμοποιηθούν από τον αγωγό. Η αποθήκευση ενός νέου μυστικού γίνεται με τον τρόπο που περιγράφεται παρακάτω:

1. Κάτω από την κατηγορία “Repository Secrets” υπάρχει η επιλογή “New repository secret” (δείτε Εικόνα 5)

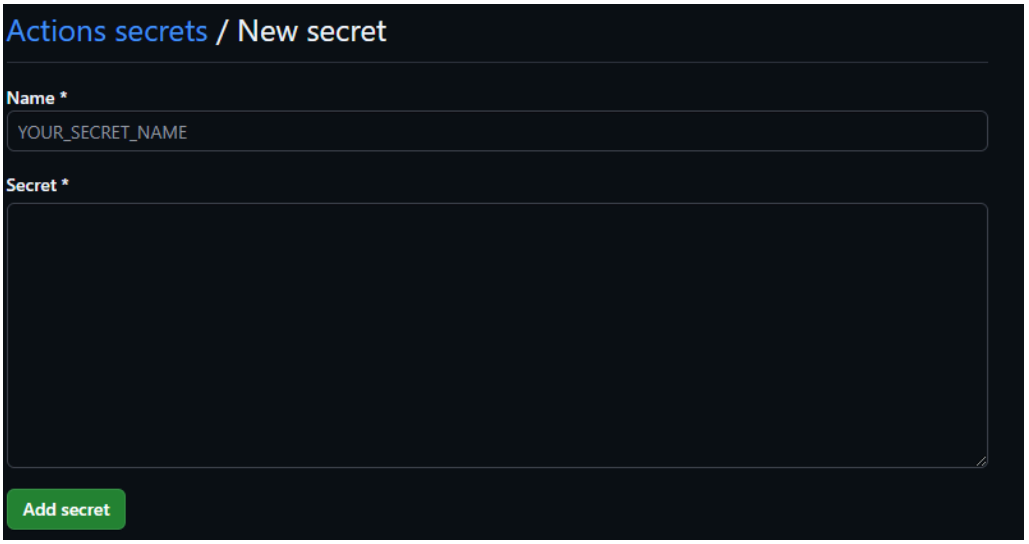
[11] https://hub.docker.com/



Εικόνα 5. Στιγμιότυπο οθόνης σχετικά με την προσθήκη νέου μυστικού

2. Μόλις αυτό πατηθεί, ανοίγει ένα νέο παράθυρο (δείτε Εικόνα 6) που επιτρέπει την αποθήκευση ενός νέου μυστικού. Αυτό προϋποθέτει την συμπλήρωση δύο πεδίων, του ονόματος (Name) και του περιεχομένου (Secret) του μυστικού.

   Σημείωση: Για την ορθή λειτουργία του συστήματος, τα μυστικά πρέπει να αποθηκευτούν με τα ονόματα, τα οποία αναφέρονται παραπάνω.



Εικόνα 6. Στιγμιότυπο οθόνης σχετικά με την προσθήκη νέου μυστικού (μέρος 2ο)

3. Μόλις τα δύο αυτά πεδία συμπληρωθούν, με το πάτημα του κουμπιού “Add secret”, το μυστικό έχει πλέον αποθηκευτεί και είναι έτοιμο για χρήση.

4.1.3 Εκτέλεση και Παρακολούθηση του Αγωγού

Εφόσον ο αγωγός έχει ρυθμιστεί σωστά, όπως περιγράφεται στα παραπάνω βήματα, και τα μυστικά έχουν αποθηκευτεί βάση των οδηγιών, τότε θα εκτελείται στις εξής περιπτώσεις:

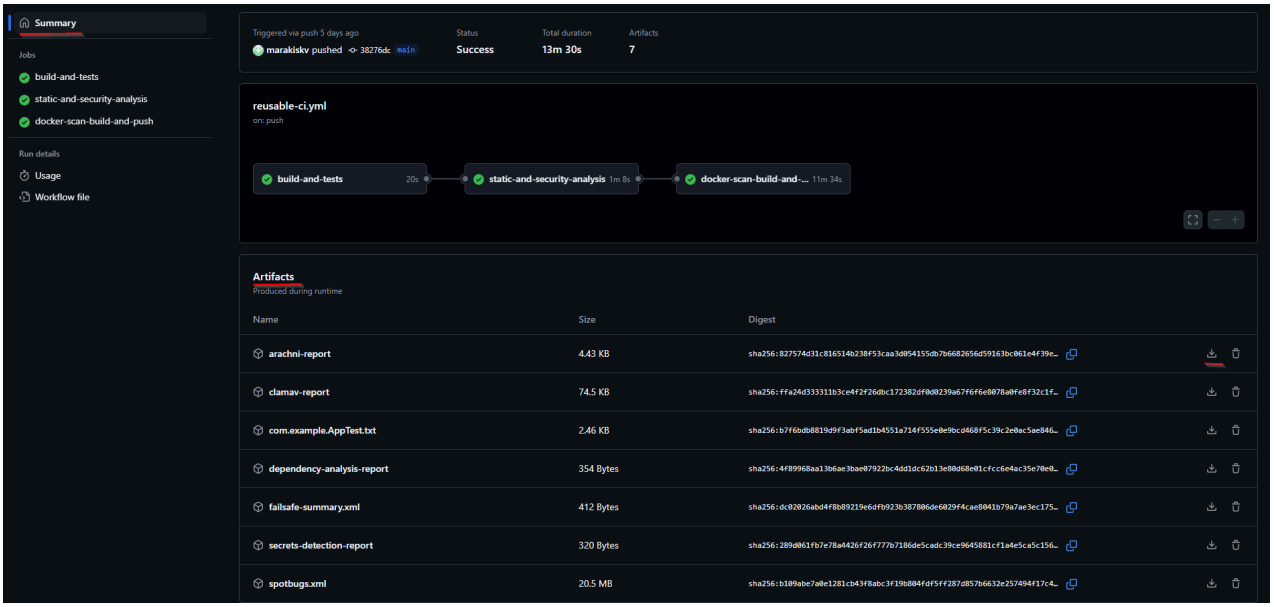
1. Κάθε φορά που γίνεται **αποστολή κώδικα** (push) προς τον κύριο (main) κλάδο (branch).
2. Κάθε φορά που γίνεται ένα **αίτημα ενσωμάτωσης** (pull request) προς τον κύριο κλάδο

3. Χειροκίνητα, μέσα από το περιβάλλον Github Actions, ύστερα από ενεργοποίηση του από τον χρήστη.

Σημειώνεται πως η ρύθμιση/προσδιορισμός των περιπτώσεων αυτών γίνεται βάσει του περιεχομένου του υλοποιημένου αγωγού. Προφανώς, ένας οργανισμός που ενδιαφέρεται στην επαναχρησιμοποίηση του αγωγού μπορεί να τροποποιήσει το περιεχόμενο αυτό ανάλογα με τις ανάγκες του (πχ. αλλάζοντας το κλαδί για το οποίο θα εκτελείται ο αγωγός).

Κατά τη διάρκεια της εκτέλεσης του αγωγού οι προγραμματιστές θα λαμβάνουν ειδοποιήσεις για την εξέλιξη του μέσω του συστήματος άμεσης ειδοποίησης προγραμματιστών (Discord) καθώς επίσης και μέσα από το περιβάλλον του Github Actions.

Στο τέλος της εκάστοτε εκτέλεσης, όλες οι αναφορές (artifacts) αποθηκεύονται στο Github Actions και είναι διαθέσιμες προς λήψη. Ο χρήστης, μπορεί να εκφορτώσει αυτές τις αναφορές (download) από το τμήμα Artifacts (όπως φαίνεται στην Εικόνα 7), το οποίο υπάρχει μέσα σε κάθε εκτέλεση του αγωγού.



Εικόνα 7. Λήψη Artifacts

4.2 Επίδειξη Συστήματος

Στην παρούσα ενότητα θα παρουσιαστεί η επίδειξη του συστήματος και θα μελετηθεί η συμπεριφορά του σε διάφορα σενάρια χρήσης προκειμένου να επιβεβαιωθεί η ορθή εκτέλεση του αγωγού.

4.2.1 Αποτυχία Εκτέλεσης Δοκιμών Ασφάλειας στον Αγωγό

Στο σημείο αυτό θα μελετηθούν οι διαφορετικές περιπτώσεις στις οποίες ο αγωγός οδηγείται σε κατάσταση αποτυχίας (fail). Αξίζει να σημειωθεί πως η αποτυχία ενός βήματος σε ένα σενάριο χρήσης οδηγεί στον τερματισμό του αγωγού. Επιπρόσθετα, προκειμένου να συνεχιστεί η εκτέλεση του αγωγού στο πλαίσιο ενός επόμενου σεναρίου χρήσης, πρέπει όλα τα ζητήματα που εμφανίστηκαν σε προηγούμενα βήματα (στο προηγούμενο σενάριο) να διορθωθούν και ο αγωγός να τρέξει ξανά από την αρχή.

4.2.1.1 Απόρριψη Λόγω Σφάλματος στον Στατικό Έλεγχο (static analysis)

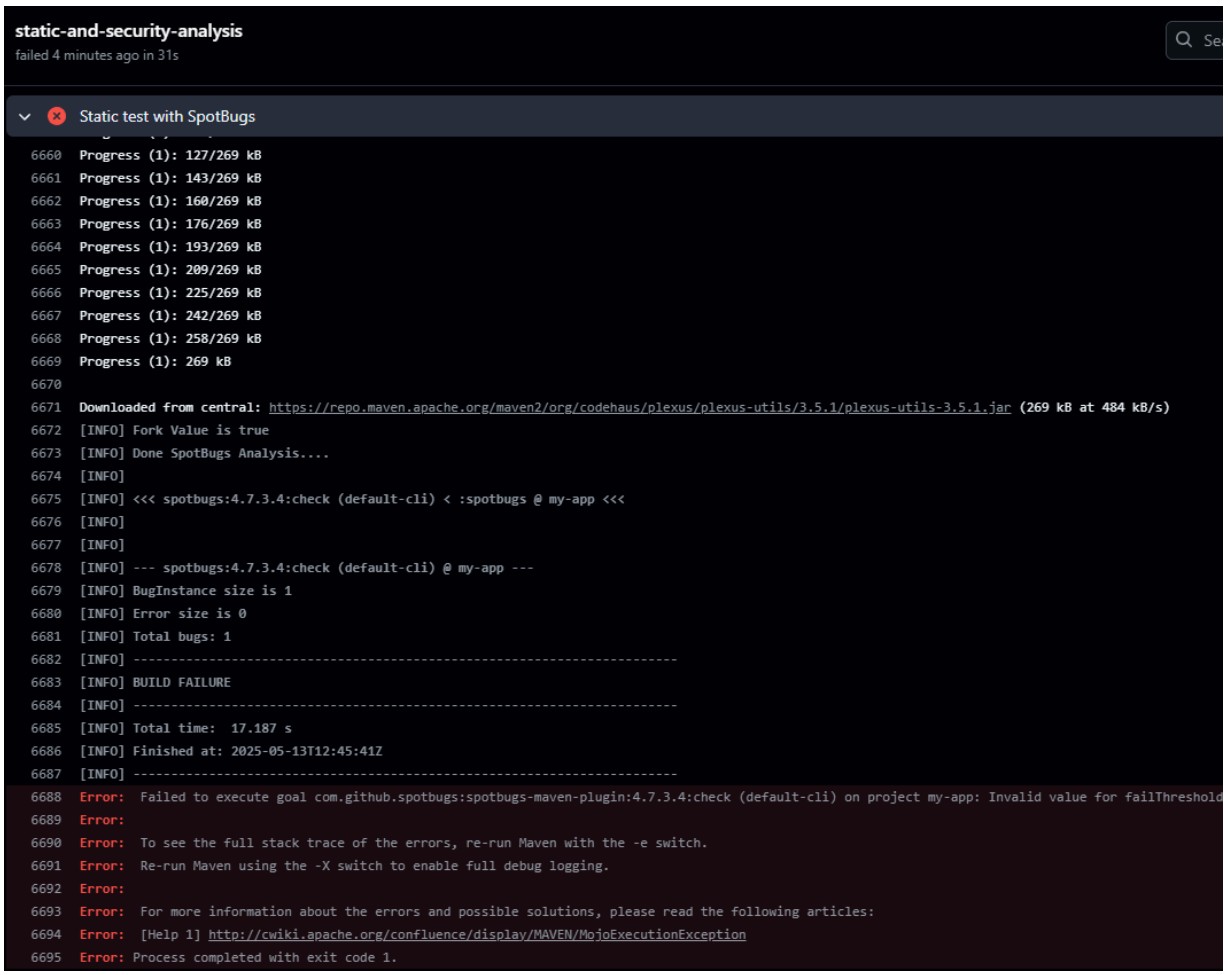
Στο σημείο αυτό πραγματοποιείται έλεγχος της εφαρμογής με την χρήση του εργαλείου Spotbugs και ειδικότερα στατική ανάλυσή της. Όπως αναφέρθηκε στην Ενότητα 3.4.2, το Spotbugs ελέγχει τον κώδικα αναφορικά με την ποιότητα και ασφάλειά του καθώς και για τον τρόπο που διαχειρίζεται τους πόρους.

Όπως φαίνεται στο στιγμιότυπο οθόνης που επισυνάπτεται παρακάτω (Εικόνα 8), έχει εισαχθεί εσκεμμένα στον κώδικα ένα σημείο όπου γίνεται προσπάθεια να τυπωθεί μία μεταβλητή το περιεχόμενο της οποίας είναι Null, πρακτική η οποία οδηγεί σε Null Pointer Exception, το οποίο αποτελεί ένα πρόβλημα με σοβαρότητα high και ως εκ τούτου ο αγωγός αποτυγχάνει.

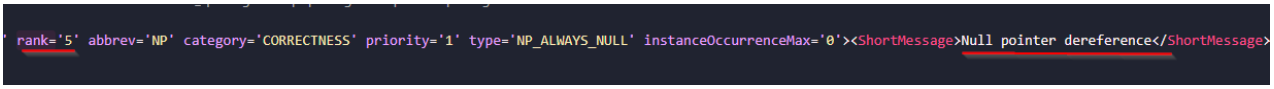
```
public BookController(BookService bookService) {

    this.bookService = new BookService(bookService);

        String s = null;
        System.out.println(s.length()); // NullPointerException
```

Εικόνα 8. Μέρος κώδικα που θα προκαλέσει αποτυχία στο στατικό έλεγχο

Στις Εικόνες 9 και 10 φαίνεται πως ο αγωγός πραγματικά οδηγείται σε κατάσταση αποτυχίας και παράγει την αντίστοιχη αναφορά, όπως ήταν το αναμενόμενο, παρουσιάζοντας επομένως συνεπή συμπεριφορά.



Εικόνα 9. Αποτυχία του αγωγού στο στατικό έλεγχο



Εικόνα 10. Αναφορά που αφορά τη αποτυχία του αγωγού στο στατικό έλεγχο

4.2.1.2 Απόρριψη Λόγω Σφάλματος στην Ανάλυση Εξαρτήσεων (dependencies check)

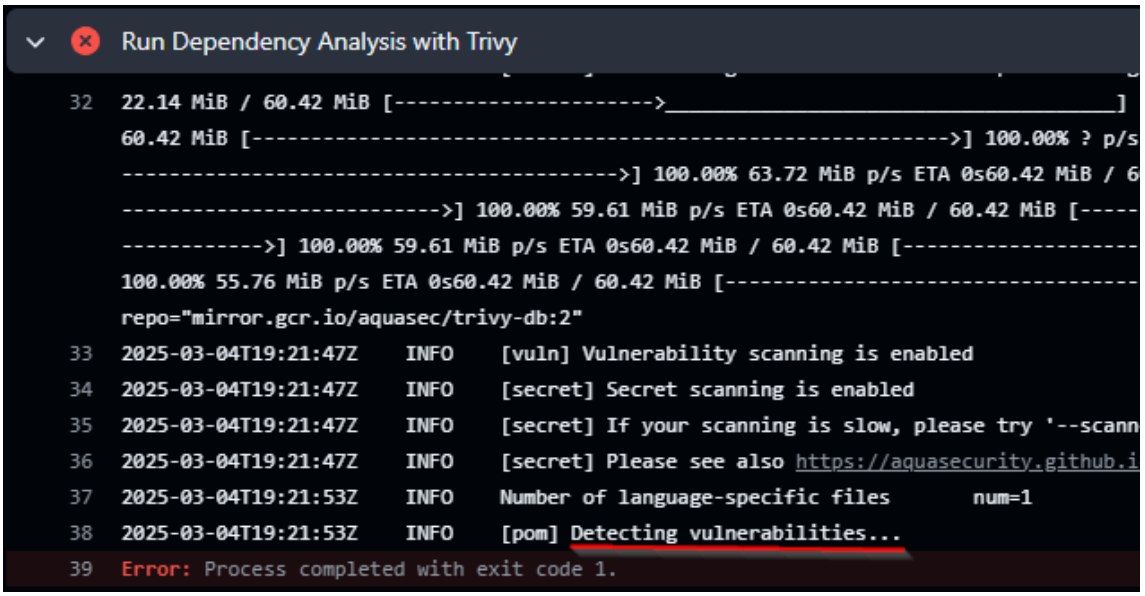
Το επόμενο στάδιο του αγωγού περιλαμβάνει, έναν ενδελεχή έλεγχο της εφαρμογής, επικεντρώνοντας στις εξαρτήσεις που χρησιμοποιεί, προκειμένου να εντοπίσει πιθανές ευπάθειες ή προβλήματα ασφαλείας. Για την πραγματοποίηση αυτού του ελέγχου αξιοποιείται το εργαλείο Trivy.

Όπως αποδεικνύεται στην Εικόνα 11, έχει προστεθεί σκόπιμα μια εξάρτηση που είναι γνωστό ότι περιλαμβάνει ευπάθειες, με σκοπό τον έλεγχο του βήματος ανίχνευσης ευπαθειών στον αγωγό.

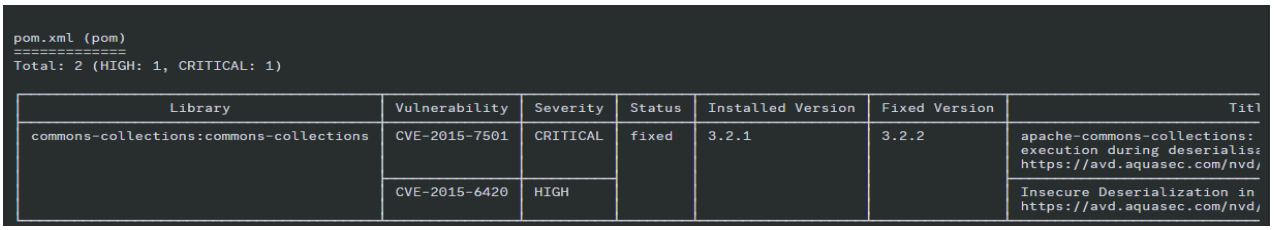
```
23      <!-- Vuln package -->
24      <dependency>
25          <groupId>commons-collections</groupId>
26          <artifactId>commons-collections</artifactId>
27          <version>3.2.1</version>
28      </dependency>
```

Εικόνα 11. Μέρος κώδικα που θα προκαλέσει αποτυχία στην ανάλυση εξαρτήσεων

Στην Εικόνα 12, απεικονίζεται η αποτυχία του αγωγού λόγω εντοπισμού κριτικής (critical severity) ευπάθειας, ενώ στην Εικόνα 13 φαίνεται η αναφορά που επιβεβαιώνει το παραπάνω και την ορθή λειτουργία του αγωγού.



Εικόνα 12. Αναφορά που αποτυπώνει την αποτυχία του αγωγού στην ανάλυση εξαρτήσεων



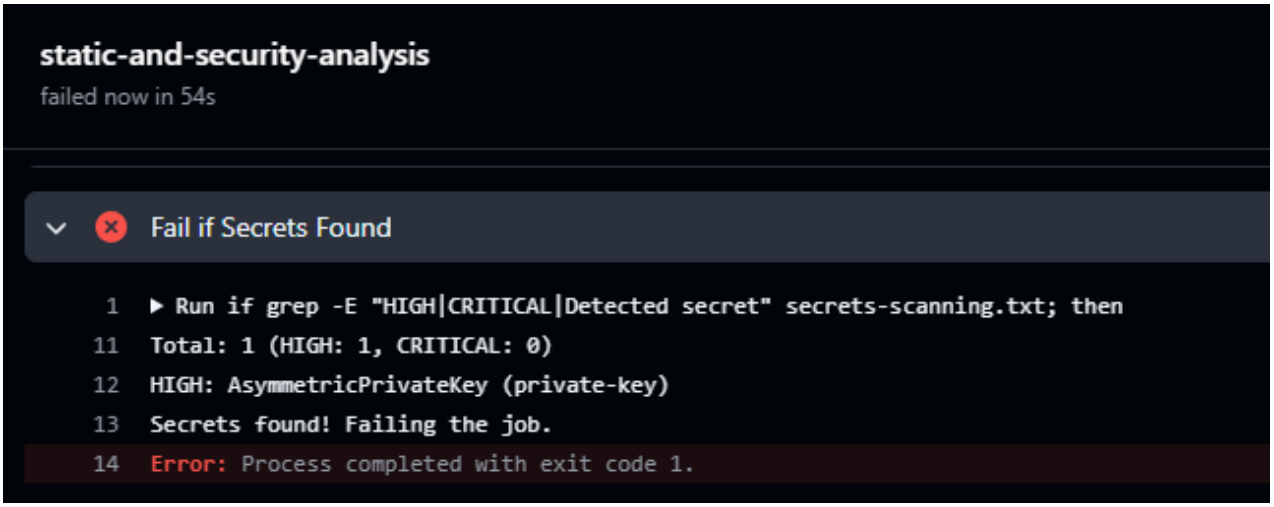
pom.xml (pom)
==========
Total: 2 (HIGH: 1, CRITICAL: 1)

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Titl
commons-collections:commons-collections	CVE-2015-7501	CRITICAL	fixed	3.2.1	3.2.2	apache-commons-collections: execution during deserialis: https://avd.aquasec.com/nvd/
	CVE-2015-6420	HIGH				Insecure Deserialization in https://avd.aquasec.com/nvd/

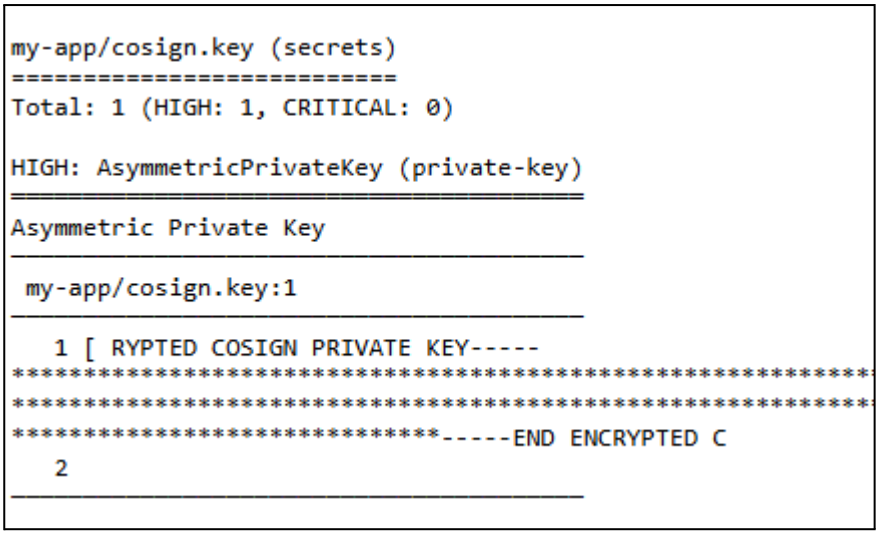
Εικόνα 13. Αναφορά που υποδηλώνει την αποτυχία του αγωγού στην ανάλυση εξαρτήσεων

4.2.1.3 Απόρριψη Λόγω Σφάλματος στην Ανίχνευση Μυστικών (secret detection)

Ο αγωγός συμπεριλαμβάνει επίσης ελέγχους που αφορούν την ανίχνευση μυστικών. Στο σημείο αυτό, προκειμένου να επιβεβαιωθεί η σωστή λειτουργία του, μέσα στο σύστημα αρχείων της εφαρμογής έχει δημιουργηθεί ένα αρχείο με το όνομα cosign.key, το οποίο εμπεριέχει το ιδιωτικό κλειδί που χρησιμοποιείται για την υπογραφή της εικόνας δοχείου (container image), σε μη κρυπτογραφημένη μορφή. Αυτό αποτελεί παραβίαση των κανόνων για την ασφαλή διαχείριση μυστικών. Στην Εικόνα 14 φαίνεται πως η ανίχνευση του μυστικού οδηγεί τον αγωγό σε κατάσταση αποτυχίας ενώ στην Εικόνα 15 φαίνεται κομμάτι της αναφοράς που επιβεβαιώνει τον εντοπισμό του μυστικού.



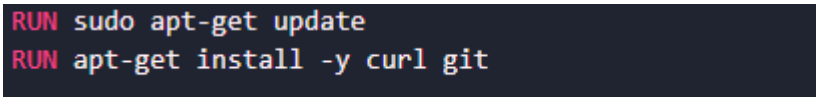
Εικόνα 14. Αποτυχία του αγωγού λόγω ανίχνευσης μυστικών



Εικόνα 15. Κομμάτι αναφοράς που υποδηλώνει την αποτυχία του αγωγού λόγω ανίχνευσης μυστικού

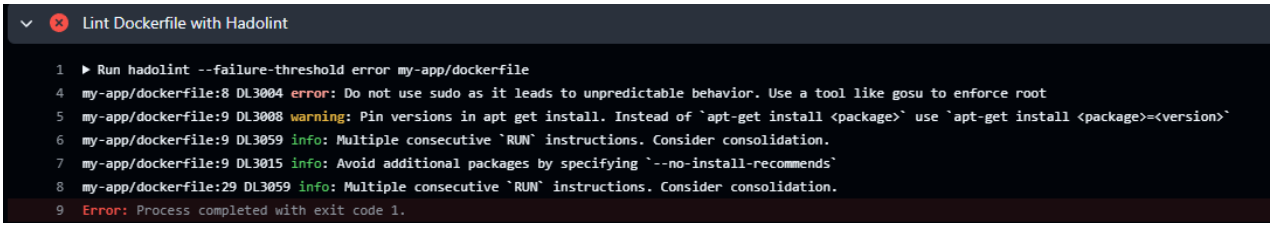
4.2.1.4 Απόρριψη Λόγω Σφάλματος στην Ανάλυση του Δοχείου με το Εργαλείο Hadolint

Όπως αναφέρθηκε στην Ενότητα 3.4.2, το εργαλείο Hadolint χρησιμοποιείται προκειμένου να ελέγξει το Dockerfile της εφαρμογής αναφορικά με τις βέλτιστες πρακτικές που πρέπει να ακολουθούνται. Στο σημείο αυτό, και όπως φαίνεται στην Εικόνα 16, το Dockerfile περιέχει μία εντολή, η οποία επρόκειτο να εκτελεστεί με δικαιώματα διαχειριστή με τη χρήση του προθέματος sudo πριν την εντολή. Η εκτέλεση εντολών με τη χρήση sudo αποτελεί μη βέλτιστη πρακτική.



Εικόνα 16. Μέρος του Dockerfile που θα οδηγήσει σε αποτυχία του ελέγχου με το εργαλείο Hadolint

Όπως συνεπώς φαίνεται στην Εικόνα 17, η χρήση μη βέλτιστης πρακτικής στο Dockerfile εντοπίζεται από το εργαλείο Hadolint και οδηγεί τον αγωγό σε κατάσταση αποτυχίας, παρουσιάζοντας συνεπώς την αναμενόμενη συμπεριφορά.

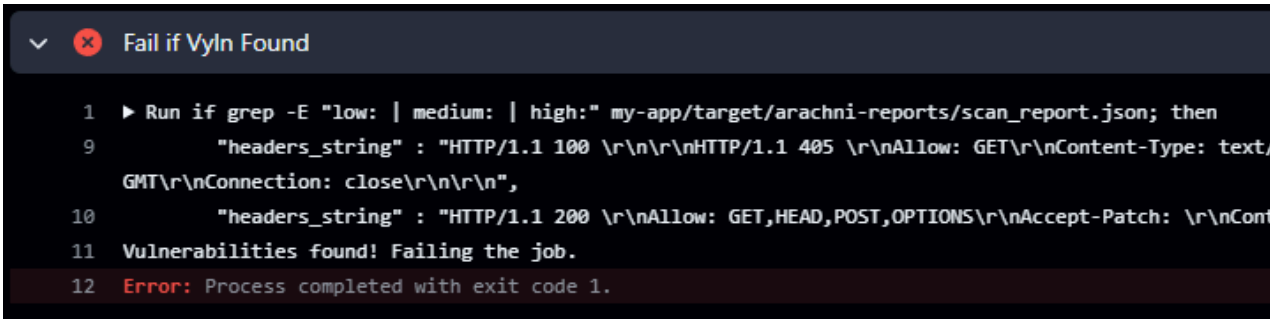


Εικόνα 17. Μήνυμα του αγωγού έπειτα από αποτυχία του ελέγχου με το εργαλείο Hadolint

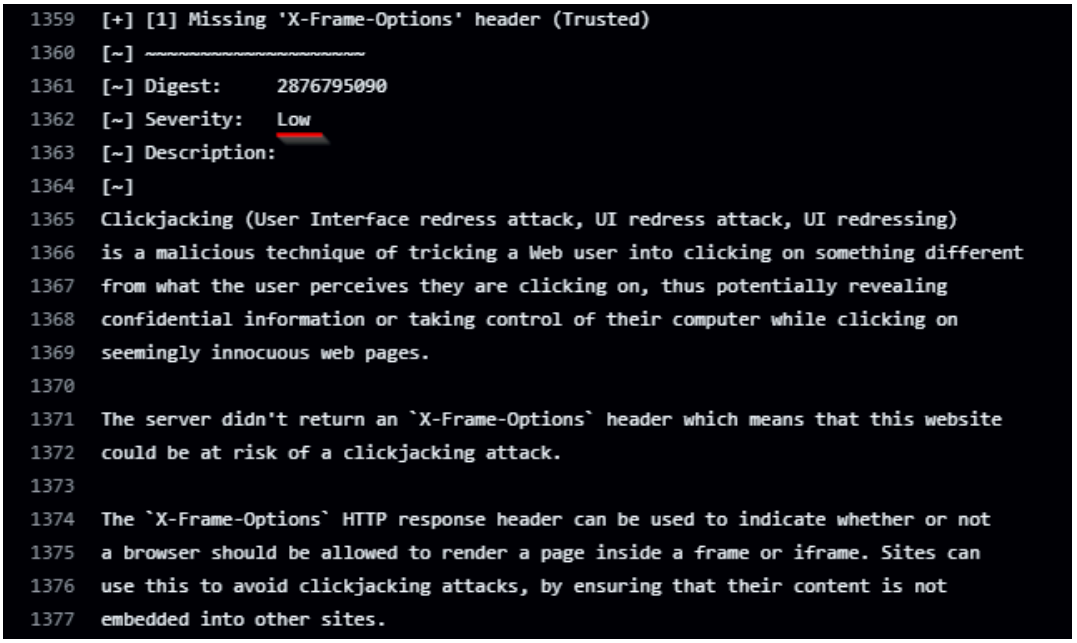
4.2.1.5 Απόρριψη Λόγω Σφάλματος στον Έλεγχο Δυναμική Ανάλυσης με το Εργαλείο Arachni (Arachni scan)

Όπως αναφέρθηκε στην Ενότητα 3.4.2, ο αγωγός αποτυγχάνει όταν ο έλεγχος (δυναμική ανάλυση) που πραγματοποιείται με το εργαλείο Arachni εντοπίσει μία ευπάθεια σοβαρότητας χαμηλής (low) ή μεγαλύτερης.

Στα στιγμιότυπα οθόνης που επισυνάπτονται παρακάτω (Εικόνες 18 και 19), φαίνεται πως πράγματι το εργαλείο Arachni στη σάρωση που πραγματοποιεί εντοπίζει μία ευπάθεια χαμηλής σοβαρότητας (low severity) και στην συνέχεια οδηγεί τον αγωγό σε κατάσταση αποτυχίας, επιβεβαιώνοντας την ορθή λειτουργία του.



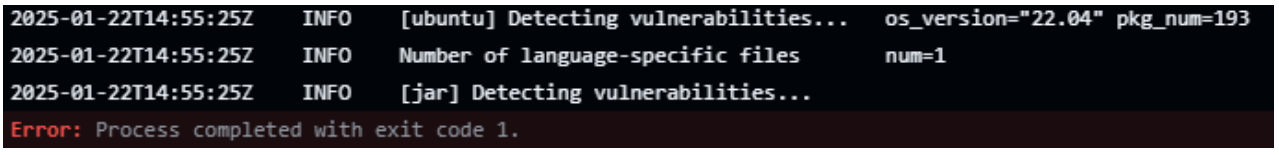
Εικόνα 18. Αποτέλεσμα εκτέλεσης του αγωγού έπειτα από αποτυχία του ελέγχου με το εργαλείο Arachni



Εικόνα 19. Αποτέλεσμα της αναφοράς που παράγει το εργαλείο Arachni

4.2.1.6 Απόρριψη Λόγω Σφάλματος στην Σάρωση Εικόνας

Όπως αναφέρθηκε στην Ενότητα 3.4.2, ο αγωγός οδηγείται σε κατάσταση αποτυχίας όταν εντοπίζονται μία (1) ή περισσότερες ευπάθειες σοβαρότητας μέτριας (medium) ή μεγαλύτερης στο στάδιο σάρωσης της εικόνας για ευπάθειες, Στην Εικόνα 20 φαίνεται πως ο αγωγός οδηγείται σε κατάσταση αποτυχίας, ενώ στην Εικόνα 21 φαίνεται πως ο αγωγός εντοπίζει 24 ευπάθειες μεσαίας σοβαρότητας.



Εικόνα 20. Μήνυμα σφάλματος του αγωγού έπειτα από αποτυχία της σάρωσης εικόνας

marakiskv/book-management:latest (ubuntu 22.04)
==========
Total: 24 (MEDIUM: 24, HIGH: 0, CRITICAL: 0)

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
gcc-12-base	CVE-2023-4039	MEDIUM	affected	12.3.0-1ubuntu1~22.04		gcc: -fstack-protector fails to guard dynamic stack allocations on ARM64 https://avd.aquasec.com/nvd/cve-2023-4039
git	CVE-2024-50349		fixed	1:2.34.1-1ubuntu1.11	1:2.34.1-1ubuntu1.12	git: Git does not sanitize URLs when asking for credentials interactively https://avd.aquasec.com/nvd/cve-2024-50349
	CVE-2024-52006					git: Newline confusion in credential helpers can lead to credential exfiltration in... https://avd.aquasec.com/nvd/cve-2024-52006
git-man	CVE-2024-50349					git: Git does not sanitize URLs when asking for credentials interactively https://avd.aquasec.com/nvd/cve-2024-50349
	CVE-2024-52006					git: Newline confusion in credential helpers can lead to credential exfiltration in... https://avd.aquasec.com/nvd/cve-2024-52006

Εικόνα 21. Μέρος της αναφοράς του Trivy από το στάδιο σάρωσης εικόνας

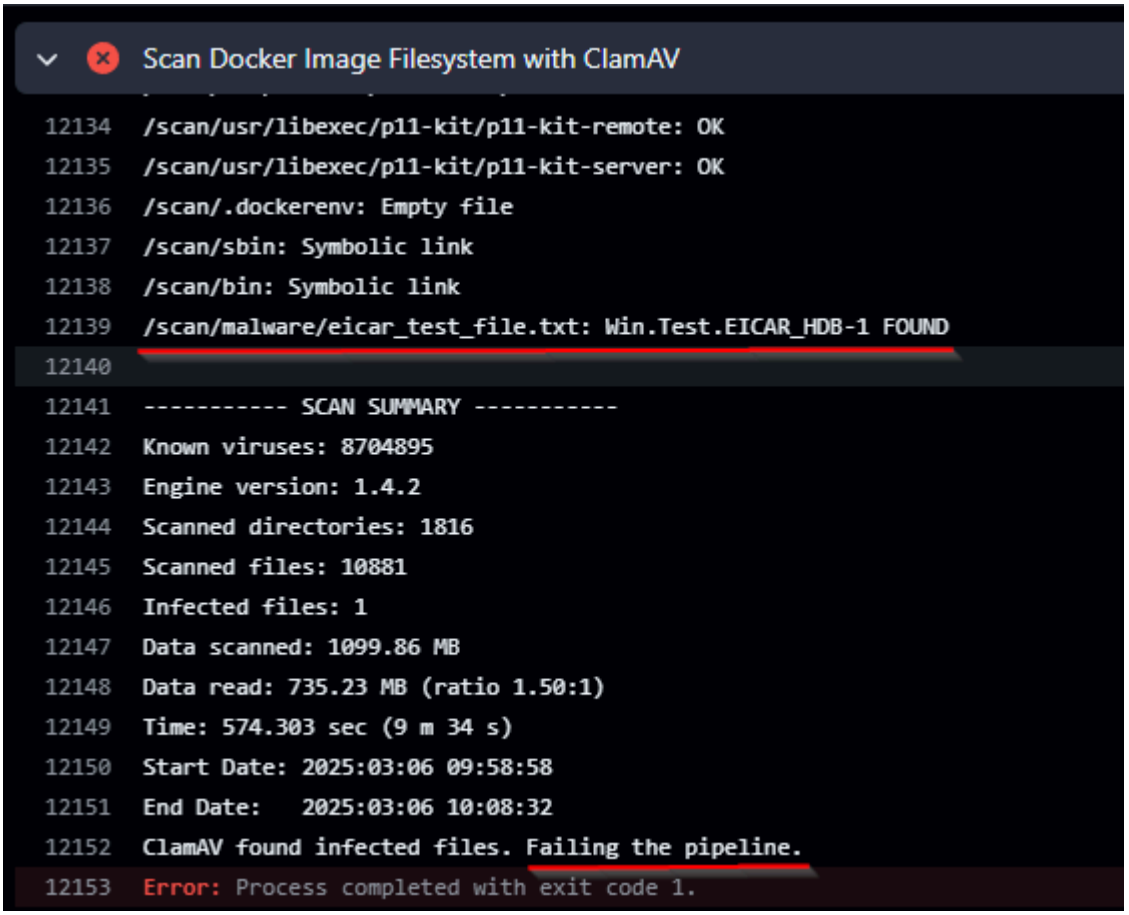
4.2.1.7 Απόρριψη Λόγω Σφάλματος στην Σάρωση για Κακόβουλο Λογισμικό στο Σύστημα Αρχείων Εικόνας

Το παρόν βήμα αφορά τον έλεγχο του συστήματος αρχείων εικόνας. Για τον σκοπό αυτό χρησιμοποιήθηκε το εργαλείο ClamAV και το δοκιμαστικό αρχείο Eicar [17], το οποίο έχει δημιουργηθεί από το Ευρωπαϊκό Ινστιτούτο Έρευνας για τους Αντι-Ιούς Υπολογιστών (EICAR) προκειμένου να δοκιμάσει την αντίδραση των προγραμμάτων προστασίας από ιούς υπολογιστών. Όπως φαίνεται στην Εικόνα 22, μέσα στον πηγαίο κώδικα της εφαρμογής υπάρχει ένα αρχείο με το όνομα eicar_test_file.txt, το οποίο περιέχει την προαναφερόμενη δοκιμαστική συμβολοσειρά Eicar, για να δοκιμαστεί η ορθή λειτουργία του εργαλείου.

my-app > eicar_test_file.txt
1 X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*

Εικόνα 22. Αρχείο που περιλαμβάνει το δοκιμαστικό αρχείο Eicar

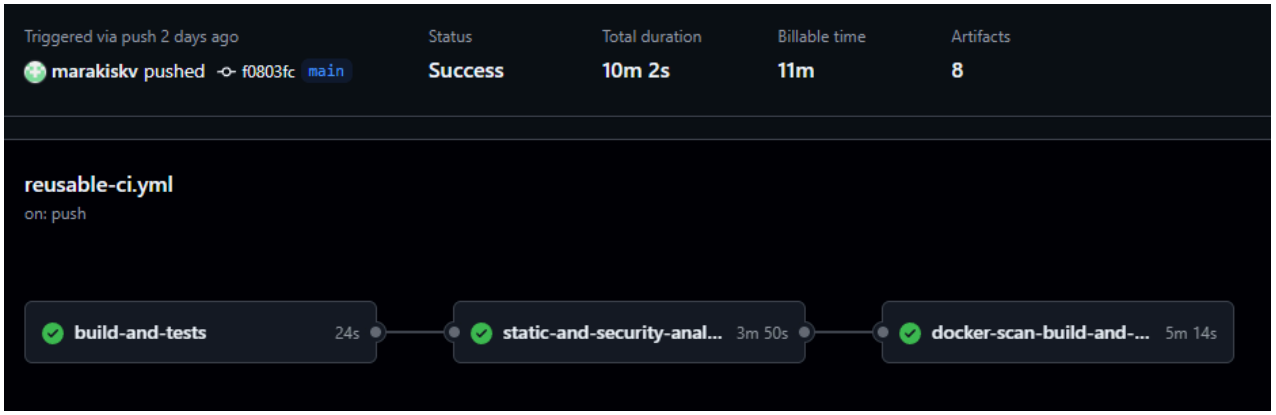
Στη συνέχεια, όπως φαίνεται στην Εικόνα 23, ο αγωγός πράγματι ανιχνεύει την ύπαρξη του δοκιμαστικού κακόβουλου λογισμικού και οδηγείται σε κατάσταση αποτυχίας. Κατά αυτό τον τρόπο επιβεβαιώνεται η ορθή λειτουργία του.



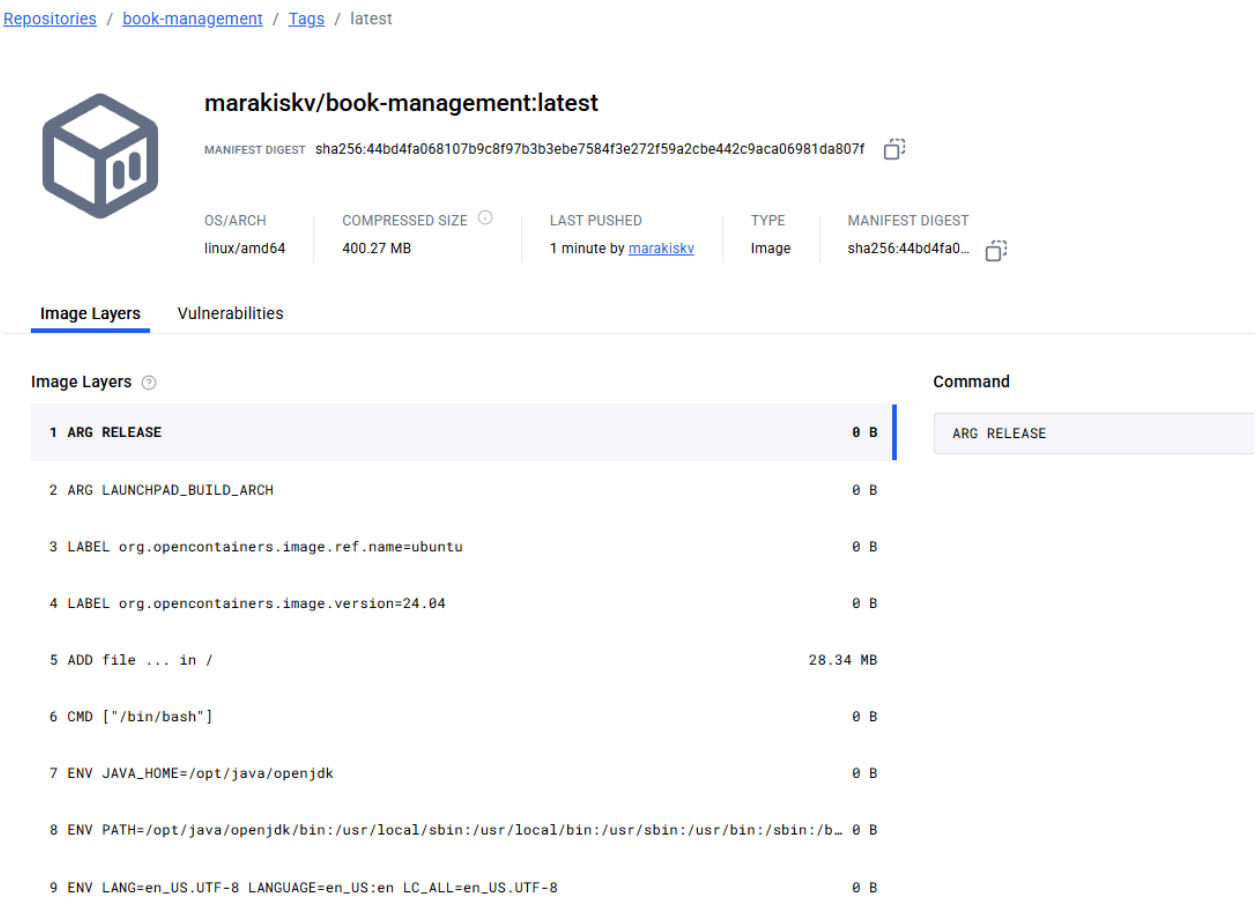
Εικόνα 23. Αποτέλεσμα εκτέλεσης του αγωγού έπειτα από την σάρωση για κακόβουλο λογισμικό

4.2.2 Έγκριση του Λογισμικού / Δοχείου

Σε περίπτωση όπου όλοι οι έλεγχοι πραγματοποιηθούν χωρίς σφάλμα, ο αγωγός ολοκληρώνεται με επιτυχία, όπως φαίνεται στην Εικόνα 24 και η κατασκευασμένη εικόνα δοχείων (docker image) προωθείται στο σύστημα διαχείρισης εκδόσεων εικόνας (Docker Hub), όπως φαίνεται στην Εικόνα 25.



Εικόνα 24. Έγκριση του λογισμικού από τον αγωγό



Εικόνα 25. Προώθηση του λογισμικού στο αποθετήριο δοχείων

Ακόμη, στην Εικόνα 26, φαίνεται πως η υπογραφή της εικόνας έχει πραγματοποιηθεί χωρίς σφάλματα και έχει επικυρωθεί. Αυτό ελέγχεται αυτοματοποιημένα στο τελευταίο βήμα του αγωγού.

```
▶ Run docker pull ***/book-management:latest
latest: Pulling from ***/book-management
Digest: sha256:28ea4582eac4662cc5d5a71f5f7d98603d45913c215351263b5686bb96f4bb1e
Status: Image is up to date for ***/book-management:latest
docker.io/***/book-management:latest

Verification for index.docker.io/***/book-management:latest --
The following checks were performed on each of these signatures:
  - The cosign claims were validated
  - The signatures were verified against the specified public key
  - Any certificates were verified against the Fulcio roots.
```

Εικόνα 26. Επικύρωση Υπογραφής

5 Συμπεράσματα και Μελλοντική Εργασία

5.1 Συμπεράσματα

Η παρούσα διπλωματική εργασία κατάφερε να αποδείξει τη σημαντικότητα της υιοθέτησης σύγχρονων DevOps πρακτικών στον τομέα της ανάπτυξης εφαρμογών που βασίζονται σε δοχεία (containers). Η βασικότερη συνεισφορά της παρούσας εργασίας έγκειται στον σχεδιασμό και στην ανάπτυξη ενός CI/CD αγωγού (pipeline), ο οποίος ενσωματώνει καλές πρακτικές ασφάλειας, ακολουθώντας και υλοποιώντας μια προσέγγιση DevSecOps.

Ο αγωγός που προτείνεται στη παρούσα αναφορά διέπεται από τα εξής χαρακτηριστικά:

1. **Αυτοματοποίηση:** Η χρήση του αγωγού κατά τη διαδικασία ανάπτυξης συμβάλλει στη μείωση του ανθρώπινου παράγοντα καθότι μεγάλο μέρος των απαιτούμενων εργασιών (ανάπτυξης) έχει πλέον αυτοματοποιηθεί. Αυτό το χαρακτηριστικό του αγωγού συμβάλλει σημαντικά στην επιτάχυνση των διαδικασιών ανάπτυξης και παράδοσης/διάταξης ενώ παράλληλα συνεισφέρει στη μείωση των σφαλμάτων.

2. **Ασφάλεια:** Ο αγωγός έχει ως στόχο να διασφαλίσει πως η εφαρμογή είναι ασφαλής και για αυτό το λόγο σε πολλά από τα στάδιά του έχουν ενσωματωθεί κατάλληλα μέτρα/έλεγχοι ασφάλειας. Με αυτό τον τρόπο, εξασφαλίζεται πως η κάθε νέα έκδοση της εφαρμογής θα είναι πάντοτε ασφαλής όταν θα παραδίδεται στον πελάτη ή διατάσσεται σε παραγωγικά περιβάλλοντα.

3. **Επεκτασιμότητα:** Ο αγωγός παρέχει δυνατότητες προσαρμογής του συστήματος βάση των εκάστοτε αναγκών. Επομένως, είναι δυνατόν να χρησιμοποιηθεί σε πληθώρα εφαρμογών ενώ είναι ευέλικτος έτσι ώστε να μπορέσει να υποστηρίξει μελλοντικές επεκτάσεις και νέες τεχνολογίες. Οι εφαρμογές στις οποίες μπορεί να χρησιμοποιηθεί, αφορούν εφαρμογές ιστού (web) γραμμένες σε γλώσσα προγραμματισμού Java (έκδοση 17), με τη χρήση του εργαλείου Maven (έκδοση 3.8.1) και το πλαίσιο (framework) Spring Boot (έκδοση 3.1.5), οι οποίες βασίζονται σε δοχεία (containers). Ο αγωγός είναι πλήρως διαθέσιμος και επαναχρησιμοποιήσιμος μέσω της πλατφόρμας CI/CD (Github Actions) που διαχειρίζεται και εκτελεί αγωγούς σε αποθετήρια κώδικα Github. Προφανώς, ο αγωγός ενδέχεται να μπορεί να χρησιμοποιηθεί και για εφαρμογές ιστού που χρησιμοποιούν τις προαναφερόμενες τεχνολογίες αλλά με διαφορετικές εκδόσεις, εφόσον πραγματοποιηθούν οι απαραίτητες ρυθμίσεις/αλλαγές σε αυτόν (ουσιαστικά θα πρέπει να αλλάξει μόνο η έκδοση της Java σε όλα τα βήματα που εγκαθιδρύουν τη χρήση του JDK).

4. **Παραμετροποίηση:** Ο αγωγός υποστηρίζει παραμετροποίηση πριν την εκτέλεση, επιτρέποντας τη διαμόρφωση των κριτηρίων αποτυχίας και των ελέγχων ασφάλειας σύμφωνα με τις εκάστοτε απαιτήσεις. Αυτό παρέχει ευελιξία στη ρύθμιση των συνθηκών λειτουργίας, βελτιώνοντας την προσαρμοστικότητα του αγωγού σε διαφορετικά περιβάλλοντα και σενάρια χρήσης.

5.2 Μελλοντική Εργασία

Παρότι η παρούσα εργασία ικανοποίησε τους βασικούς της στόχους, υπάρχει περιθώριο για βελτίωση και επέκταση της βασικής συνεισφοράς της που είναι ο αγωγός DevSecOps. Ενδεικτικά, οι κατευθύνσεις μελλοντικής εργασίας περιλαμβάνουν:

1. **Βελτίωση Ασφάλειας:** Ο αγωγός μπορεί να επεκταθεί μελλοντικά προσθέτοντας προηγμένους μηχανισμούς εντοπισμού ευπαθειών αλλά και μηχανισμούς που επιτρέπουν την αυτοματοποιημένη διόρθωση τους. Παράλληλα, η ενσωμάτωση εργαλείων ασφάλειας βασισμένων στη τεχνητή νοημοσύνη (ΤΝ), στον αγωγό θα συνεισέφερε σημαντικά στον εντοπισμό ευπαθειών που δεν είναι γνωστές ή καταγεγραμμένες. Κατά αυτό τον τρόπο, ο αγωγός θα μπορούσε να εντοπίζει και να προλαμβάνει προηγμένους τύπους επιθέσεων (sophisticated attacks), προσφέροντας έτσι ένα επιπλέον μέτρο ασφαλείας.

2. **Επέκταση Υποστήριξης:** Ο συγκεκριμένος αγωγός σχεδιάστηκε προκειμένου να εξυπηρετεί εφαρμογές ιστού (web), οι οποίες είναι γραμμένες στη γλώσσα προγραμματισμού Java και χρησιμοποιούν το πλαίσιο (framework) Maven. Σε μία μελλοντική επέκταση, ο αγωγός θα μπορούσε να αναβαθμιστεί προκειμένου να υποστηρίζει εφαρμογές γραμμένες σε οποιαδήποτε έκδοση Java και Maven καθώς και περισσότερα είδη εφαρμογών, υλοποιημένα σε διαφορετικές γλώσσες προγραμματισμού. Στο ιδανικό σενάριο, ο αγωγός θα μπορούσε να είναι λειτουργικός ανεξαρτήτως του είδους της εφαρμογής και προγραμματιστικής γλώσσας (language and application agnostic).

3. **Συνεργασία με κανονιστικά πλαίσια:** Ο αγωγός μπορεί να επεκταθεί προκειμένου να ενσωματώσει εργαλεία, τα οποία διασφαλίζουν συμμόρφωση με κανονιστικά πλαίσια, όπως τα GDPR[12], ISO 27001:2013[13] και SOC 2[14]. Μία τέτοια λειτουργία θα συνεισέφερε σημαντικά στην απλοποίηση των διαδικασιών ελέγχου από τις κανονιστικές αρχές, αφού τα αποτελέσματα θα προέκυπταν σε μορφή αναφορών (συμμόρφωσης) με αυτοματοποιημένο τρόπο.

4. **Ανεξαρτησία από την εκάστοτε πλατφόρμα CI/CD:** Μια μελλοντική πιθανή επέκταση του συστήματος αφορά την ανεξαρτησία του από συγκεκριμένες πλατφόρμες CI/CD, διασφαλίζοντας τη φορητότητα και την ευελιξία του. Αυτό σημαίνει ότι ο αγωγός θα μπορεί να εκτελείται σε οποιοδήποτε περιβάλλον, χωρίς να εξαρτάται από μια προϋπάρχουσα πλατφόρμα για την εκτέλεσή του.

[12] https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng
[13] https://www.itgovernance.co.uk/iso27001#:~:text=ISO%2FIEC%2027001%20is%20the,(information%20security%20management%20system).
[14] https://www.imperva.com/learn/data-security/soc-2-compliance/

Βιβλιογραφία

1. Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges, and Practices. *IEEE Access*, 5, 3909-3943.
2. Leppänen, M., Mäntylä, M. V., Pagels, M., Eloranta, V., Itkonen, J., Mäntylä, T., & Mikkonen, T. (2015). The Highways and Country Roads to Continuous Deployment. IEEE Software, 32(2), 64-72.
3. H. Myrbakken and R. Colomo-Palacios, "DevSecOps: A Multivocal Literature Review," in Communications in Computer and Information Science, Cham: Springer International Publishing, 2017, pp. 17–29. Available: http://dx.doi.org/10.1007/978-3-319-67383-7_2
4. F. Lombardi and A. Fanton, "From DevOps to DevSecOps is not enough. CyberDevOps: an extreme shifting-left architecture to bring cybersecurity within software security lifecycle pipeline," Software Quality Journal, vol. 31, no. 2, pp. 619–654
5. https://docs.docker.com/engine/security/
6. https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html
7. https://www.ibm.com/think/topics/continuous-deployment
8. https://ortelius.io/blog/2024/02/28/what-are-non-functional-requirements-and-why-do-they-matter/
9. https://visuresolutions.com/el/%CE%BF%CE%B4%CE%B7%CE%B3%CF%8C%CF%82-%CE%B9%CF%87%CE%BD%CE%B7%CE%BB%CE%B1%CF%83%CE%B9%CE%BC%CF%8C%CF%84%CE%B7%CF%84%CE%B1%CF%82-%CE%B4%CE%B9%CE%B1%CF%87%CE%B5%CE%AF%CF%81%CE%B9%CF%83%CE%B7%CF%82-%CE%B1%CF%80%CE%B1%CE%B9%CF%84%CE%AE%CF%83%CE%B5%CF%89%CE%BD/%CE%BB%CE%B5%CE%B9%CF%84%CE%BF%CF%85%CF%81%CE%B3%CE%B9%CE%BA%CE%AD%CF%82-%CE%AD%CE%BD%CE%B1%CE%BD%CF%84%CE%B9-%CE%BC%CE%B7-%CE%BB%CE%B5%CE%B9%CF%84%CE%BF%CF%85%CF%81%CE%B3%CE%B9%CE%BA%CE%AD%CF%82-%CE%B1%CF%80%CE%B1%CE%B9%CF%84%CE%AE%CF%83%CE%B5%CE%B9%CF%82/?
10. https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/
11. Doe, J., Smith, A. (2017). *Software Requirements Engineering*. Academia. Retrieved from https://www.academia.edu/30541351/Software_Requirements_Engineering
12. https://github.com/customer-terms/github-online-services-sla
13. https://support.discord.com/hc/en-us/articles/204849977-How-do-I-create-a-server
14. https://support.discord.com/hc/en-us/articles/228383668-Intro-to-Webhooks
15. https://edu.chainguard.dev/open-source/sigstore/cosign/cosign-manual-way/
16. https://www.geeksforgeeks.org/software-development-life-cycle-sdlc/
17. https://www.eicar.org/download-anti-malware-testfile/
18. https://www.iso.org/standard/27001
19. https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng
20. https://learn.microsoft.com/en-us/devops/what-is-devops
21. https://www.microsoft.com/en-us/security/business/security-101/what-is-devsecops

Παράρτημα Ι: DevSecOps CI/CD Αγωγός

```yaml
name: CI

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]
  workflow_dispatch:
    inputs:
      logLevel:
        description: 'Log level'
        required: true
        default: 'warning'
      environment:
        description: 'Environment to deploy'
        required: false
        default: 'staging'

jobs:
  build-and-tests:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2
        with:
          fetch-depth: 1

      - name: Set up JDK 17
        uses: actions/setup-java@v2
        with:
          java-version: '17'
          distribution: 'adopt'

      - name: Unit Tests
        run:  mvn clean test
        working-directory: ./my-app
```

```
      - name: Send to Discord
        if: always()
        env:
          DISCORD_WEBHOOK: ${{ secrets.DISCORD_WEBHOOK }}
        run: |
          curl -X POST \
          -H "Content-Type: multipart/form-data" \
          -F "file=@./my-app/target/surefire-reports/com.example.AppTest.txt" \
          $DISCORD_WEBHOOK

      - name: Upload unit tests report
        if: always()
        uses: actions/upload-artifact@v4
        with:
          name: com.example.AppTest.txt
          path: ./my-app/target/surefire-reports/


  # Job 2: Static and Security Analysis
  static-and-security-analysis:
    runs-on: ubuntu-latest
    needs: build-and-tests
    steps:
      - name: Checkout code
        uses: actions/checkout@v2
        with:
          fetch-depth: 1

      - name: Set up JDK 17
        uses: actions/setup-java@v2
        with:
          java-version: '17'
          distribution: 'adopt'

      - name: Integration Tests
        run: mvn verify -Dskip.unit.tests=true -Ddependency-check.skip=true
        working-directory: ./my-app

      - name: Send to Discord
        if: always()
```

```
        env:
          DISCORD_WEBHOOK: ${{ secrets.DISCORD_WEBHOOK }}
        run: |
          curl -X POST \
          -H "Content-Type: multipart/form-data" \
          -F "file=@./my-app/target/failsafe-reports/failsafe-summary.xml" \
          $DISCORD_WEBHOOK

      - name: Upload coverage report
        if: always()
        uses: actions/upload-artifact@v4
        with:
          name: failsafe-summary.xml
          path: ./my-app/target/failsafe-reports/

      - name: Static test with SpotBugs
        id: spotbugs
        run: mvn spotbugs:check
        working-directory: ./my-app

      - name: Upload SpotBugs Report
        if: always()
        uses: actions/upload-artifact@v4
        with:
          name: spotbugs.xml
          path: ./my-app/target/

      - name: Send to Discord
        if: always()
        env:
          DISCORD_WEBHOOK: ${{ secrets.DISCORD_WEBHOOK }}
        run: |
          curl -X POST \
          -H "Content-Type: multipart/form-data" \
          -F "file=@./my-app/target/spotbugs.xml" \
          $DISCORD_WEBHOOK

      - name: Fail if SpotBugs found issues
        if: failure()
        run: echo "SpotBugs failed. Stopping the pipeline." && exit 1
```

```
      # Dependency Analysis with Trivy
      - name: Run Dependency Analysis with Trivy
        run: |
          docker run --rm \
            -v $(pwd):/app \
            aquasec/trivy:latest fs  --offline-scan --exit-code 1 --severity
HIGH,CRITICAL /app/my-app > dependency-analysis.txt

      - name: Upload Dependency Analysis Report
        if: always()
        uses: actions/upload-artifact@v4
        with:
          name: dependency-analysis-report
          path: dependency-analysis.txt

      - name: Send to Discord
        if: always()
        env:
          DISCORD_WEBHOOK: ${{ secrets.DISCORD_WEBHOOK }}
        run: |
          curl -X POST \
            -H "Content-Type: multipart/form-data" \
            -F "file=@dependency-analysis.txt" \
            $DISCORD_WEBHOOK

      - name: Scan for Secrets with Trivy
        run: |
          docker run --rm  \
          -v $(pwd)/my-app:/app:rw  \
          aquasec/trivy:latest repo \
          --scanners secret \
          --severity HIGH,CRITICAL \
          --offline-scan \
          /app > secrets-scanning.txt

      - name: Fail if Secrets Found
        run: |
          if grep -E "HIGH|CRITICAL|Detected secret" secrets-scanning.txt; then
          echo "Secrets found! Failing the job."
          exit 1
          else
```

```
          echo "No secrets found."
          fi

      - name: Upload Secrets Detection Report
        if: always()
        uses: actions/upload-artifact@v4
        with:
          name: secrets-detection-report
          path: secrets-scanning.txt

      - name: Send to Discord
        if: always()
        env:
          DISCORD_WEBHOOK: ${{ secrets.DISCORD_WEBHOOK }}
        run: |
          curl -X POST \
          -H "Content-Type: multipart/form-data" \
          -F "file=@secrets-scanning.txt" \
          $DISCORD_WEBHOOK

  # Job 3: Docker Build, Runtime Security Test, and Push
  docker-scan-build-and-push:
    runs-on: ubuntu-latest
    # build-and-tests,
    needs: [build-and-tests, static-and-security-analysis]
    steps:
      - name: Checkout code
        uses: actions/checkout@v2
        with:
          fetch-depth: 1

      - name: Install Hadolint
        run: |
            wget -O /usr/local/bin/hadolint
https://github.com/hadolint/hadolint/releases/latest/download/hadolint-Linux-x86
_64
            chmod +x /usr/local/bin/hadolint

      - name: Lint Dockerfile with Hadolint
        run: hadolint --failure-threshold error my-app/dockerfile
```

```
      - name: Set up Docker
        working-directory: ./my-app
        run: |
          docker network create app_network
          docker build -t ${{ secrets.DOCKER_HUB_USERNAME
}}/book-management:latest --no-cache .
          docker run -d --name application-container --network app_network -p
8080:8080 ${{ secrets.DOCKER_HUB_USERNAME }}/book-management:latest

      - name: Wait for the application to start
        run: sleep 10  # Αναμονή 10 δευτερολέπτων για να ξεκινήσει η εφαρμογή

      - name: Check running containers
        run: docker ps

      - name: Check if app is running
        run: curl http://localhost:8080/books || (docker logs
application-container && exit 1)

      - name: Arachni Scan
        run: |
          mkdir -p $(pwd)/my-app/target/arachni-reports

          docker run --name arachni_scan  \
          --network app_network \
          -v $(pwd)/my-app/target/arachni-reports:/arachni/reports \
          arachni/arachni:latest \
          /bin/sh -c "
              /usr/local/arachni/bin/arachni http://application-container:8080/
--report-save-path /arachni/reports/scan_report.afr &&
              /usr/local/arachni/bin/arachni_reporter
--report=json:outfile=/arachni/reports/scan_report.json
/arachni/reports/scan_report.afr
          "

      - name: Stop Application
        run: docker stop application-container

      - name: Upload Arachni Report
        uses: actions/upload-artifact@v4
        with:
```

```
          name: arachni-report
          path: my-app/target/arachni-reports/scan_report.json

      - name: Fail if Vyln Found
        run: |
            if grep -E "medium: | high:"
my-app/target/arachni-reports/scan_report.json; then
            echo "Vulnerabilities found! Failing the job."
            exit 1
            else
            echo "No vulnerabilities found."
            fi

      - name: Send to Discord
        if: always()
        env:
          DISCORD_WEBHOOK: ${{ secrets.DISCORD_WEBHOOK }}
        run: |
          curl -X POST \
          -H "Content-Type: multipart/form-data" \
          -F "file=@my-app/target/arachni-reports/scan_report.json" \
          $DISCORD_WEBHOOK

      - name: Log in to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_HUB_USERNAME }}
          password: ${{ secrets.DOCKER_HUB_PASSWORD }}

      - name: Scan Docker Image with Trivy
        run: |
            docker run --rm \
            -v /var/run/docker.sock:/var/run/docker.sock \
            aquasec/trivy:latest image --offline-scan \
            --severity MEDIUM,HIGH,CRITICAL \
            --exit-code 1 \
            ${{ secrets.DOCKER_HUB_USERNAME }}/book-management:latest

            docker run --rm \
            -v /var/run/docker.sock:/var/run/docker.sock \
            aquasec/trivy:latest image --offline-scan \
```

```
            --severity HIGH,CRITICAL,MEDIUM \
            --exit-code 0 \
            ${{ secrets.DOCKER_HUB_USERNAME }}/book-management:latest > 
trivy-report.txt



      - name: Upload Vulnerabilities Scan Report
        if: always()
        uses: actions/upload-artifact@v4
        with:
          name: trivy-report
          path: trivy-report.txt

      - name: Send to Discord
        if: always()
        env:
          DISCORD_WEBHOOK: ${{ secrets.DISCORD_WEBHOOK }}
        run: |
          curl -X POST \
          -H "Content-Type: multipart/form-data" \
          -F "file=@trivy-report.txt" \
          $DISCORD_WEBHOOK

      - name: Scan Docker Image Filesystem with ClamAV
        run: |
          mkdir -p extracted
          docker create --name temp-container ${{ secrets.DOCKER_HUB_USERNAME 
}}/book-management:latest
          docker export temp-container > book-management.tar
          tar -xf book-management.tar -C extracted
          docker run --rm \
          -v $(pwd)/extracted:/scan \
          clamav/clamav:latest clamscan -r /scan > clamav-report.txt || 
EXIT_CODE=$?
          cat clamav-report.txt
          if [ "$EXIT_CODE" -eq 1 ]; then
            echo "ClamAV found infected files. Failing the pipeline."
            exit 1
          elif [ "$EXIT_CODE" -gt 1 ]; then
            echo "ClamAV encountered an error during the scan. Failing the 
pipeline."
```

```
            exit $EXIT_CODE
          fi

      - name: Upload ClamAV Report
        if: always()
        uses: actions/upload-artifact@v4
        with:
          name: clamav-report
          path: clamav-report.txt

      - name: Send to Discord
        if: always()
        env:
          DISCORD_WEBHOOK: ${{ secrets.DISCORD_WEBHOOK }}
        run: |
          curl -X POST \
          -H "Content-Type: multipart/form-data" \
          -F "file=@clamav-report.txt" \
          $DISCORD_WEBHOOK

      - name: Push Docker image
        if: success()
        run: docker push ${{ secrets.DOCKER_HUB_USERNAME
}}/book-management:latest

      - name: Install Cosign
        run: |
          wget
https://github.com/sigstore/cosign/releases/download/v1.4.1/cosign-linux-amd64
          chmod +x cosign-linux-amd64
          mv cosign-linux-amd64 /usr/local/bin/cosign

      - name: Sign Docker image
        env:
          COSIGN_PASSWORD: ${{ secrets.COSIGN_PASSWORD }}
        run: |
          echo "${{ secrets.COSIGN_KEY }}" > cosign.key
          cosign sign --key cosign.key ${{ secrets.DOCKER_HUB_USERNAME
}}/book-management:latest

      - name: Verify Docker image
```

```
        env:
          COSIGN_PUB: ${{ secrets.COSIGN_PUB }}
        run: |
          docker pull ${{ secrets.DOCKER_HUB_USERNAME }}/book-management:latest
          cosign verify --key my-app/cosign.pub ${{ secrets.DOCKER_HUB_USERNAME
}}/book-management:latest
```