ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Σύστημα Καταγραφής Τιμών Ξενοδοχείων με τεχνικές web data extraction»

Του φοιτητή
Αβραμίδη Ερωτόκριτου
Αρ. Μητρώου: 164625

Επιβλέπων
Σαλαμπάσης Μιχάλης
Βαθμίδα Καθηγητής

Ημερομηνία 05/09/2023

Τίτλος Π.Ε. Σύστημα Καταγραφής Τιμών Ξενοδοχείων με τεχνικές web data extraction
Κωδικός Π.Ε. 22334
Ονοματεπώνυμο φοιτητή Αβραμίδης Ερωτόκριτος
Ονοματεπώνυμο εισηγητή Σαλαμπάσης Μιχάλης
Ημερομηνία ανάληψης Π.Ε 02-11-2022
Ημερομηνία περάτωσης Π.Ε 05-09-2023

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

«Αφιέρωση»

Η παρούσα πτυχιακή εργασία είναι αφιερωμένη στους γονείς μου.

Πρόλογος

Σε μια εποχή όπου η τεχνολογία παίζει καθοριστικό ρόλο στη διαμόρφωση των επιχειρήσεων και της συμπεριφοράς των καταναλωτών, η σωστή συνεργασία και αλληλεπίδραση των συστημάτων αλλά και προγραμματιστών που αναπτύσσουν τα κομμάτια μιας πλατφόρμας όπως front-end και back-end είναι ζωτικής σημασίας για την επιτυχία οποιουδήποτε έργου. Αυτή η πτυχιακή εργασία, ήταν μια ιδανική ευκαιρία να εξερευνήσω το πάθος μου για τις πρακτικές DevOps και τις τεχνολογίες web scraping, στις οποίες έχω επαγγελματική εμπειρία και στις δύο. Το να είσαι μέλος μιας ομάδας όπου κάθε μέλος είναι υπεύθυνο για μια ξεχωριστή αλλά αλληλεξαρτώμενη πτυχή (front-end, back-end και DevOps)—πρόσφερε μια ανεκτίμητη μαθησιακή εμπειρία. Απαιτούσε όχι μόνο τεχνική επάρκεια αλλά και αποτελεσματική ομαδική εργασία και δεξιότητες επικοινωνίας για να ευθυγραμμίσουμε τα διακριτά μας στοιχεία σε ένα συνεκτικό σύνολο. Το έργο μου έδωσε την ευκαιρία να εφαρμόσω καινοτόμες πρακτικές DevOps, να αντιμετωπίσω προκλήσεις όπως το scaling, security και anti-bot, solutions, monitoring, deployment και τελικά να επεκτείνω το υπάρχον σύνολο των δεξιοτήτων μου. Ήταν ένα ταξίδι που ενίσχυσε την κατανόησή μου για τις πολυπλοκότητες που συνεπάγονται στη δημιουργία ενός επεκτάσιμου, αποτελεσματικού και ασφαλούς συστήματος παρακολούθησης των τιμών, θέτοντας μια ισχυρή βάση για μελλοντικές προσπάθειες στον τομέα της Πληροφορικής.

Περίληψη

Αυτή η πτυχιακή διερευνά την ανάπτυξη ενός συστήματος παρακολούθησης τιμών ειδικά σχεδιασμένου για την παρακολούθηση των τιμών των δωματίων για επιλεγμένα ξενοδοχεία από το booking.com. Ο πρωταρχικός στόχος είναι η έγκαιρη αποστολή ειδοποιήσεων στους χρήστες όταν υπάρχουν σημαντικές αλλαγές τιμών που αντιστοιχούν στα κριτήρια που έχουν θέσει οι ίδιοι. Το έργο χωρίζεται σε πέντε κύρια στοιχεία: ανάπτυξη front-end, λογική back-end, scrapping, scheduling, infrastructure, με το καθένα να το χειρίζεται διαφορετικό μέλος της ομάδας.

Το αποτέλεσμα αυτού του συλλογικού έργου είναι ένα scalable, ισχυρό σύστημα ικανό να διαχειρίζεται μεγάλο όγκο αιτημάτων web scrapping χωρίς να θέτει σε κίνδυνο την εμπειρία του χρήστη. Ένα από τα βασικά χαρακτηριστικά είναι η εφαρμογή αποτελεσματικών μηχανισμών load balancing και διαχείρισης ουρών (queues) που διασφαλίζουν ότι το σύστημα μπορεί να χειριστεί έναν αυξανόμενο αριθμό αιτημάτων. Επιπλέον, αναπτύχθηκαν dashboards που παρέχουν μετρήσεις και στατιστικά στοιχεία σε πραγματικό χρόνο. Αυτά τα dashboards προορίζονται για τους διαχειριστές για να αποκτήσουν πληροφορίες σχετικά με την απόδοση του συστήματος, τις τάσεις χρήσης και τα πιθανά σημεία bottlenecks.

Ένα άλλο σημαντικό επίτευγμα ήταν η επιτυχής ενσωμάτωση των αυτοματοποιημένων deployments (CI/CD), που επιτρέπει στους προγραμματιστές να ενημερώνουν και να συντηρούν το codebase με ελάχιστο χρόνο downtime. Ενσωματώθηκαν επίσης μέτρα ασφαλείας όπως anti-bot detection και IP rotation για να διασφαλιστεί η αξιοπιστία και η στιβαρότητα του web scrapping.

Συνοπτικά, το project δείχνει πόσο καλά μια ομάδα μπορεί να συνεργαστεί όταν κάθε άτομο φέρνει διαφορετικές δεξιότητες στο τραπέζι. Το σύστημά μας βοηθά τους χρήστες να εξοικονομήσουν χρήματα δείχνοντας τους πότε αλλάζουν οι τιμές των ξενοδοχείων η δωματίων που τους αφορούν.

«Hotel Price Monitoring System with web data extraction techniques»

«Avramidis Erotokritos»

Abstract

This project examines the development and operationalization of a specialized price monitoring system. The system is engineered to alert users about changes in room prices for specific hotels, sourcing data from predetermined online platforms. The project is uniquely segmented into three interconnected components: front-end, back-end logic, scrapping, scheduling and infrastructure each assigned to a specialized team member.

My role as the DevOps specialist involved a multitude of responsibilities, including but not limited to CI/CD setup, database architecture, containerization, and ensuring the platform's scalability and security. As a result, the project culminated in a robust, scalable system capable of handling an extensive number of web scraping requests efficiently, without compromising on user experience.

One of the standout features of the system is its capability to balance workloads effectively and manage large data queues, thereby accommodating a growing user base. To complement this, a comprehensive dashboard was designed to furnish administrators with real-time metrics and key performance indicators. This aids in the timely identification of system bottlenecks and usage trends, facilitating data-driven decision-making. Automatic deployments were also integrated into the system to ensure a seamless user experience during updates and system maintenance.

The system not only serves users by providing timely alerts to help them make informed booking decisions but also equips administrators with the necessary tools for optimal platform management. Overall, this thesis serves as a case study in effective interdisciplinary collaboration, leading to a practical, user-centric solution.

Ευχαριστίες

Ολοκληρώνοντας αυτήν την εργασία, θα ήθελα να εκφράσω την ευγνωμοσύνη μου για την ευκαιρία να εμβαθύνω σε τόσο περίπλοκα και σχετικά θέματα. Το ταξίδι μέσω του DevOps δεν ήταν μόνο εκπαιδευτικό αλλά και επαγγελματικά εμπλουτιστικό. Είμαι ιδιαίτερα ευγνώμων για την ομαδική εργασία και την τεχνογνωσία που έφερε κάθε μέλος της ομάδας στο έργο. Αυτή η προσπάθεια ήταν μια συλλογική μαθησιακή εμπειρία και ανυπομονώ να εφαρμόσω αυτές τις ιδέες σε μελλοντικές πρωτοβουλίες. Ευχαριστώ όλους όσους συνέβαλαν στην πραγματοποίηση αυτού του έργου, από τον ιδεασμό μέχρι την εκτέλεση.

Περιεχόμενα

Κατάλογος Σχημάτων

Δεν βρέθηκαν καταχωρήσεις πίνακα εικόνων.

Κατάλογος Πινάκων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία

Εισαγωγική Επισκόπηση

1.1 Εισαγωγή

Η άνοδος της ψηφιακής εποχής είχε βαθύ αντίκτυπο σε πολλούς τομείς της οικονομίας, με τον κλάδο του τουρισμού, της διαμονής και της φιλοξενίας να επηρεάζεται ιδιαίτερα. Ο ψηφιακός μετασχηματισμός σε αυτόν τον τομέα έχει προχωρήσει αρκετά τα τελευταία χρόνια. έχει εξελιχθεί σε ένα περίπλοκο οικοσύστημα διασυνδεδεμένων platforms που προσφέρουν πληθώρα υπηρεσιών. Αυτά κυμαίνονται από κρατήσεις και κριτικές πελατών έως εξελιγμένες μηχανές σύγκρισης τιμών. Σε αυτόν τον ταχέως εξελισσόμενο κλάδο, τα συστήματα παρακολούθησης των τιμών έχουν καταστεί ζωτικής σημασίας τόσο για τους καταναλωτές όσο και για τους ξενοδόχους.

Θα μπορούσε κανείς να υποστηρίξει ότι με την αφθονία των διαδικτυακών πλατφορμών, οι καταναλωτές δεν είχαν ποτέ τόσες πολλές πληροφορίες στα χέρια τους. Ωστόσο, παραδόξως, αυτή η αφθονία πληροφοριών έχει επίσης οδηγήσει σε ένα είδος «υπερφόρτωσης πληροφοριών», όπου η πρόκληση δεν είναι η απόκτηση δεδομένων αλλά η ανάλυση και η αποτελεσματική ερμηνεία τους. Επομένως, η χρησιμότητα ενός ισχυρού συστήματος παρακολούθησης των τιμών δεν μπορεί να υπερεκτιμηθεί. Αυτά τα συστήματα χρησιμεύουν ως εργαλεία λήψης αποφάσεων, απλοποιώντας τεράστιες ποσότητες δεδομένων σε πρακτικές ιδέες. Για τους καταναλωτές, αυτό μπορεί να σημαίνει τη διαφορά μεταξύ μιας οικονομικά αποδοτικής διαμονής και μιας υπερτιμημένης εμπειρίας. για τους ξενοδόχους, αυτό θα μπορούσε να επηρεάσει τις στρατηγικές τιμολόγησης και τα ποσοστά πληρότητας.

Επιπλέον, καθώς η ψηφιακή αγορά έχει επεκταθεί, τόσο διευρύνεται το εύρος και η πολυπλοκότητα των εργαλείων που απαιτούνται για τη διαχείρισή της. Τα σύγχρονα συστήματα παρακολούθησης τιμών δεν είναι πλέον αυτόνομες εφαρμογές, αλλά μέρος ενός ευρύτερου, συχνά εξατομικευμένου, οικοσυστήματος πληροφορικής. Αυτό έχει δημιουργήσει νέες προκλήσεις όσον αφορά τη διασφάλιση ότι αυτά τα συστήματα δεν είναι μόνο λειτουργικά αλλά και scable, ασφαλή και εύκολα στη διαχείριση. Εδώ παίζει ρόλο το DevOps - συντομογραφία για Ανάπτυξη (Dev) και Λειτουργίες IT (Ops). Οι πρακτικές DevOps έχουν γίνει αναπόσπαστο κομμάτι της διαχείρισης της πολυπλοκότητας των σύγχρονων διαδικτυακών εφαρμογών και αυτή η μελέτη στοχεύει να συμβάλει σε αυτό το σύνολο γνώσεων.

Η έννοια του web scraping, η οποία αναφέρεται στην αυτόματη εξαγωγή πληροφοριών από ιστότοπους, είναι επίσης κρίσιμη για τα συστήματα παρακολούθησης τιμών. Καθώς οι επιχειρήσεις έχουν συνειδητοποιήσει τη σημασία των διαδικτυακών τους δεδομένων, έχουν εφαρμόσει διάφορα προστατευτικά μέτρα για την ασφάλειά τους, συμπεριλαμβανομένων τεχνολογιών anti-bot και περιορισμών IP. Έτσι, η ανάγκη για εξελιγμένες τεχνολογίες και μεθοδολογίες web scrapping γίνεται όλο και πιο σημαντική. Αυτή η μελέτη στοχεύει να διερευνήσει και να εφαρμόσει αποτελεσματικές μεθόδους scraping, διατηρώντας παράλληλα τις ηθικές οδηγίες, μια άλλη σημαντική συμβολή στο πεδίο.

Τέλος, η εξέλιξη των τεχνολογιών cloud computing και containerization όπως το Docker έχει αλλάξει δραστικά τον τρόπο με τον οποίο αναπτύσσονται και κλιμακώνονται οι υπηρεσίες web. Οι υπηρεσίες Cloud προσφέρουν απαράμιλλη ευελιξία και scalability, ενώ το containerization επιτρέπει την αποτελεσματική χρήση των πόρων του συστήματος, γρήγορο development και εύκολο replication. Η αποτελεσματική εφαρμογή αυτών των τεχνολογιών αποτελεί μέρος των στόχων του έργου.

1.2 Σημασία της εργασίας

Η σημασία αυτής της εργασίας έγκειται στην πολύπλευρη προσέγγισή της για την επίλυση ενός σύγχρονου ζητήματος - πώς να παρακολουθείτε αποτελεσματικά και αποδοτικά οι τιμές των δωματίων του ξενοδοχείου προς όφελος τόσο των καταναλωτών όσο και των διαχειριστών. Με την υπερπληθώρα διαδικτυακών πλατφορμών που προσφέρουν διάφορες τιμές και προσφορές, οι καταναλωτές συχνά κατακλύζονται από επιλογές. Από τη πλευρά των διαχειριστών, μια ολοκληρωμένη εικόνα της απόδοσης της πλατφόρμας μπορεί να είναι εξίσου συντριπτική. Αυτή η μελέτη στοχεύει να φέρει μια ισορροπημένη λύση και στα δύο άκρα, αυξάνοντας έτσι τη συνολική εμπειρία.

Η αξία για τους καταναλωτές είναι ξεκάθαρη. Το σύστημα παρακολούθησης τιμών που αναπτύχθηκε σε αυτήν τη μελέτη προσφέρει ειδοποιήσεις στους χρήστες, διασφαλίζοντας ότι ενημερώνονται όταν συμβαίνει μια αλλαγή στην τιμολόγηση ενός επιλεγμένου δωματίου ή ξενοδοχείου. Σε μια κοινωνία που εκτιμά την αμεσότητα και την ευκολία, ένα τέτοιο χαρακτηριστικό δεν είναι απλώς πολυτέλεια αλλά προσδοκία.

Οι διαχειριστές είναι η άλλη σημαντική μεριά αυτής της μελέτης. Τα δεδομένα σε πραγματικό χρόνο είναι ένα χρυσωρυχείο για κάθε επιχείρηση. Μέσω της χρήσης πρακτικών DevOps και σύγχρονων συστημάτων διαχείρισης βάσεων δεδομένων, αυτό το έργο προσφέρει στους διαχειριστές μια ολοκληρωμένη σειρά εργαλείων για την αποτελεσματική παρακολούθηση του συστήματος. Η χρήση ενημερωτικών πινάκων εργαλείων με μετρήσεις σε πραγματικό χρόνο και βασικούς δείκτες απόδοσης παρέχει ένα ανεκτίμητο πλεονέκτημα για τη λήψη αποφάσεων.

Επιπλέον, αυτό το έργο παρέχει μια εις βάθος ματιά στην ενσωμάτωση χαρακτηριστικών scaling, anti-bot μέτρων και πρωτοκόλλων ασφαλείας σε ένα σύστημα παρακολούθησης τιμών. Δεδομένης της ευαίσθητης φύσης των διαδικτυακών δεδομένων και της ανάγκης προσαρμογής των συστημάτων σε διαφορετικά επίπεδα ζήτησης, αυτές οι πτυχές της μελέτης προσφέρουν κρίσιμες γνώσεις σχετικά με τις βέλτιστες πρακτικές για τη δημιουργία ισχυρών, scalable και ασφαλών διαδικτυακών συστημάτων.

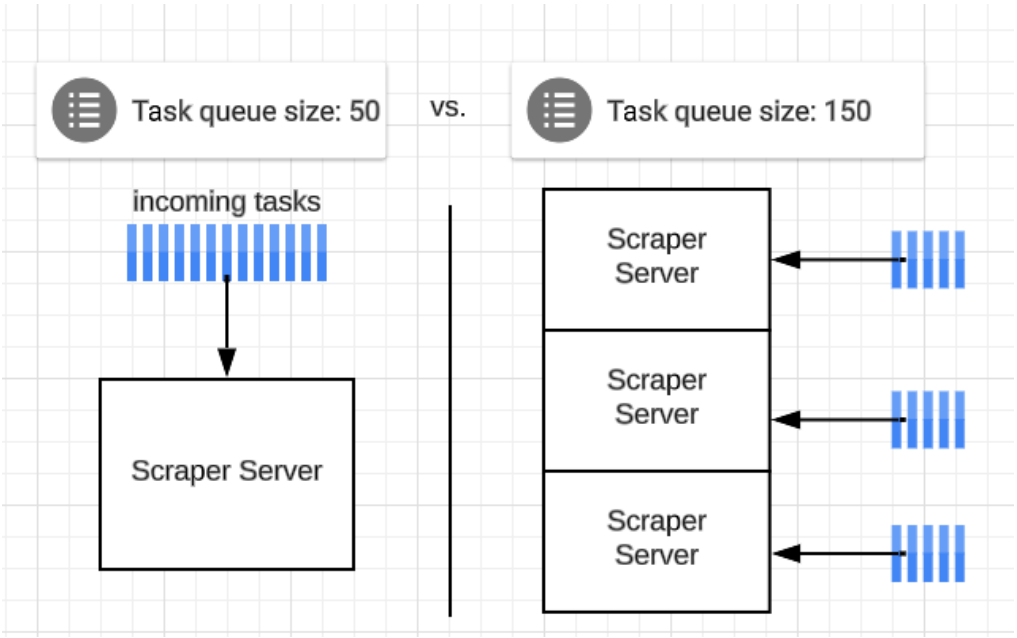
Εξίσου σημαντικό είναι ότι αυτό το project χρησιμεύει ως πρακτική εφαρμογή της διεπιστημονικής συνεργασίας. Με διαφορετικές πτυχές του συστήματος - front-end, back-end και DevOps - που αναπτύσσονται από διάφορα μέλη της ομάδας, το έργο υπογραμμίζει τη σημασία της αποτελεσματικής ομαδικής εργασίας στην ανάπτυξη λογισμικού. Αυτό παρέχει χρήσιμα μαθήματα τόσο για την ακαδημαϊκή έρευνα όσο και για πρακτικές εφαρμογές στο πεδίο.

1.3 Στόχοι

Οι στόχοι αυτής της μελέτης εκτείνονται σε πολλαπλές πτυχές, στοχεύοντας τόσο στις τεχνικές πολυπλοκότητες όσο και στις πρακτικές ανάγκες που εμπλέκονται στη δημιουργία ενός συστήματος παρακολούθησης τιμών.

Ανάπτυξη ενός Scalable συστήματος παρακολούθησης τιμών:

Μία από τις κρίσιμες προκλήσεις στην παρακολούθηση των τιμών είναι το scaling. Το σύστημα πρέπει να

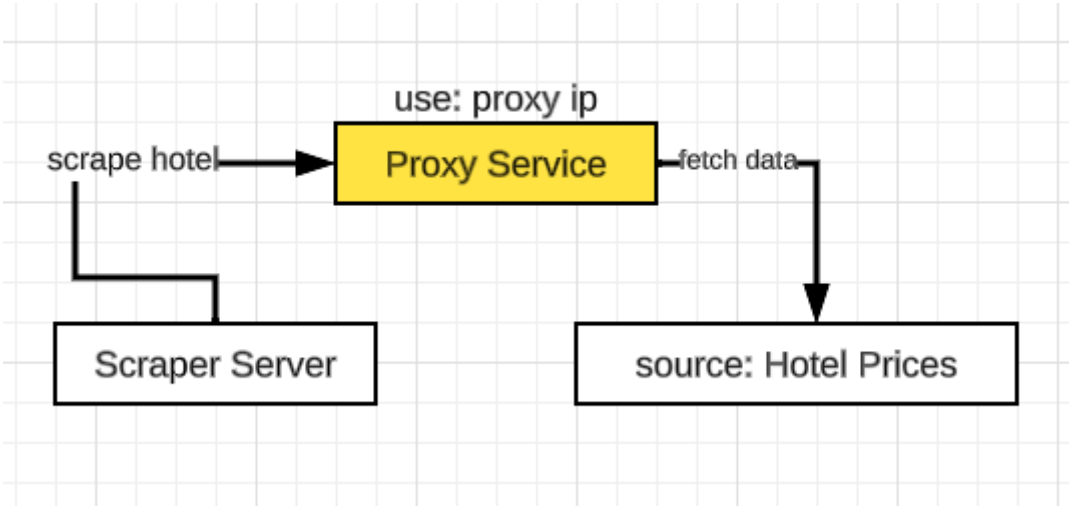


Σχήμα 1.1: scalable σύστημα

σχεδιαστεί για να δέχεται έναν αυξανόμενο αριθμό εργασιών και δεδομένων καθώς μεγαλώνει.

Εφαρμογή τεχνικών Anti-Bot για την αποφυγή μπλοκ των scrappers:

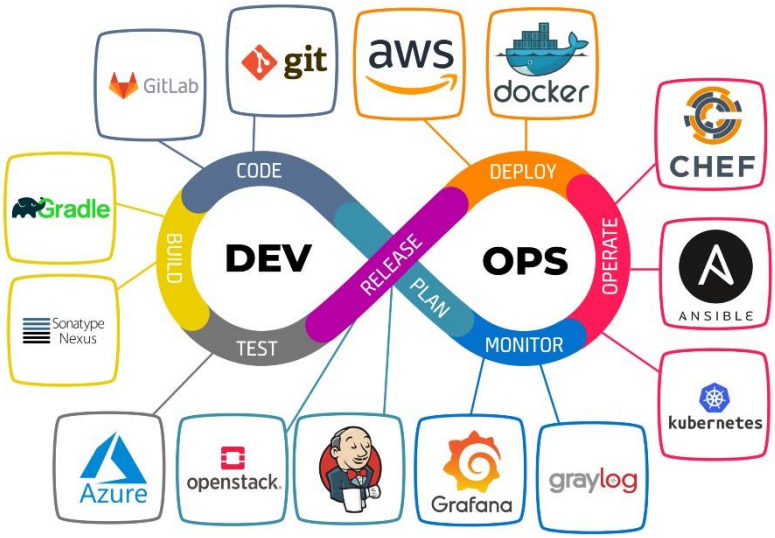
Κατά το web scrapping, το σύστημα είναι επίσης επιρρεπές σε flagging ή αποκλεισμό από τις πηγές που παρακολουθούνται. Ως εκ τούτου, η εφαρμογή τεχνικών anti-bot για τη συγκάλυψη των δραστηριοτήτων scrapping και την αποφυγή μπλοκ πηγών είναι ζωτικής σημασίας. Φυσικά, πάντα μένοντας μέσα στα νομικά πλαίσια.



Σχήμα 1.3.2: Request using proxy service

Δημιουργία ισχυρής υποδομής:

Ένας βασικός στόχος είναι ο σχεδιασμός και η εφαρμογή μιας στρατηγικής DevOps που περιλαμβάνει CI/CD pipelines, αυτοματοποιημένες πρακτικές deployments και ολοκληρωμένες υπηρεσίες logging και monitoring.



Μετρήσεις σε πραγματικό χρόνο για τους διαχειριστές:

Οι διαχειριστές έχουν πρόσβαση σε δεδομένα σε πραγματικό χρόνο, όπως αρχεία καταγραφής εφαρμογών, μετρήσεις ουρών και άλλες σημαντικές πληροφορίες μέσω ενός πίνακα dashboard.

Σχήμα 1.3.4: Monitoring Dashboard

1.4 Περιορισμοί και Γενικά Προβλήματα

Ακρίβεια δεδομένων

Το σύστημα έχει σχεδιαστεί για να κάνει scrape και να παρακολουθεί τις τιμές των δωματίων όσο το δυνατόν ακριβέστερα. Ωστόσο, η ακρίβεια των συλλεγμένων δεδομένων εξαρτάται αναπόφευκτα από την αξιοπιστία των πηγών που παρακολουθούνται. Τυχόν σφάλματα, ασυνέπειες ή αλλαγές στη μορφοποίηση των δεδομένων προέλευσης θα μπορούσαν να επηρεάσουν άμεσα την ακρίβεια του συστήματος.

Πόροι

Το scaling είναι κρίσιμο κομμάτι, αλλά είναι απαραίτητο να αναγνωρίσουμε τους υπολογιστικούς περιορισμούς, ειδικά όσον αφορά τις δυνατότητες του server και CPU/RAM. Αυτοί οι περιορισμοί θα μπορούσαν να οδηγήσουν σε μειωμένη απόδοση του συστήματος, ιδιαίτερα σε περιόδους υψηλής ζήτησης.

Νομικοί και ηθικοί περιορισμοί

Το web scrapping νομικά κατατάσσεται στην γκρίζα ζώνη και ισχύουν διαφορετικοί κανόνες από πλατφόρμα σε πλατφόρμα. Η μελέτη πρέπει να διερευνήσει επιμελώς αυτούς τους περιορισμούς, περιορίζοντας δυνητικά το εύρος και τη μέθοδο συλλογής δεδομένων.

Πολυπλοκότητα συστήματος

Δεδομένων των διαφόρων στοιχείων που εμπλέκονται, από το frontend έως το backend, οι scrappers, οι schedulers και την υποδομή DevOps, η πολυπλοκότητα του ίδιου του συστήματος αποτελεί περιορισμό. Σφάλματα ή καθυστερήσεις σε ένα στοιχείο θα μπορούσαν ενδεχομένως να επηρεάσουν ολόκληρο το σύστημα.

Κόστος συστήματος

Καθώς το σύστημα κλιμακώνεται, το κόστος που σχετίζεται με τη χρήση servers, την αποθήκευση δεδομένων και άλλα λειτουργικά έξοδα θα αυξηθεί αναπόφευκτα. Η εξισορρόπηση της απόδοσης και των δυνατοτήτων του συστήματος έναντι των περιορισμών του προϋπολογισμού είναι μια διαρκής πρόκληση.

1.5 Επίλογος

Συνοπτικά, το πρώτο κεφάλαιο έχει ως στόχο να θέσει το υπόβαθρο για την έρευνα που πραγματοποιήθηκε σε αυτή τη πτυχιακή. Εισήγαγε το πρόβλημα της παρακολούθησης των τιμών των ξενοδοχείων και περιέγραψε τους στόχους που έχουν τεθεί για την αντιμετώπιση αυτού του ζητήματος.

Αυτό το κεφάλαιο χρησιμεύει ως χάρτης για το τι βρίσκεται μπροστά. Στόχος του είναι να βοηθήσει τους αναγνώστες να κατανοήσουν το τοπίο του προβλήματος και τις μεθοδολογικές διαδρομές που ακολουθούνται για την πλοήγησή του. Καθώς εμβαθύνουμε στα επόμενα κεφάλαια, κάθε ενότητα θα επεκταθεί σε αυτά τα εισαγωγικά στοιχεία, προσφέροντας μια βαθύτερη και λεπτομερέστερη κατανόηση του θέματος.

Στο επόμενα κεφάλαια, θα πραγματοποιήσουμε μια εξαντλητική βιβλιογραφική ανασκόπηση για τις τεχνικές του web scrapping, τις πρακτικές DevOps, τις προκλήσεις scaling και άλλους βασικούς τομείς.

Ανασκόπηση της Βιβλιογραφίας

1.6 Εισαγωγή

Σε αυτό το κεφάλαιο, θα διερευνήσουμε την υπάρχον γνώση που αποτελεί τη βάση για αυτήν τη μελέτη. Η βιβλιογραφική ανασκόπηση παρέχει μια λεπτομερή ανάλυση της έρευνας που έχει πραγματοποιηθεί στο παρελθόν και των μεθόδων που σχετίζονται με το θέμα της εργασίας. Είναι μια σημαντική εξέταση του πώς η έρευνά μας εντάσσεται στα ευρύτερα ακαδημαϊκά και τεχνολογικά τοπία.

1.7 Συστήματα Παρακολούθησης Τιμών

Η έννοια της παρακολούθησης των τιμών έχει κερδίσει σημαντική προσοχή τόσο στην ακαδημαϊκή βιβλιογραφία όσο και στις βιομηχανικές εφαρμογές. Τα συστήματα παρακολούθησης τιμών έχουν σχεδιαστεί για να παρακολουθούν, αναλύουν και αναφέρουν συνεχώς αλλαγές τιμών για ένα ευρύ φάσμα προϊόντων και υπηρεσιών. Αυτά τα συστήματα μπορεί να είναι εξαιρετικά ωφέλιμα για τους καταναλωτές που επιδιώκουν να κάνουν οικονομικά αποδοτικές επιλογές και για τις επιχειρήσεις που στοχεύουν να παραμείνουν ανταγωνιστικές σε δυναμικές αγορές.

Στο πλαίσιο της τιμολόγησης δωματίων ξενοδοχείου, αυτά τα συστήματα καθίστανται ιδιαίτερα κρίσιμα. Επιτρέπουν στους χρήστες να αποκτούν πληροφορίες σε πραγματικό ή σχεδόν πραγματικό χρόνο σχετικά με τις κυμαινόμενες τιμές, δίνοντάς τους έτσι τη δυνατότητα να λαμβάνουν καλά ενημερωμένες αποφάσεις. Επιπλέον, τέτοια συστήματα είναι αναπόσπαστα στην κατανόηση των

τάσεων της αγοράς, των εποχιακών διακυμάνσεων και των στρατηγικών τιμολόγησης των ανταγωνιστών.

Αρκετές ερευνητικές εργασίες και μελέτες περιπτώσεων έχουν εξερευνήσει τους αλγόριθμους και τις τεχνικές πίσω από τα συστήματα παρακολούθησης τιμών. Αυτά συχνά περιλαμβάνουν μοντέλα μηχανικής εκμάθησης για την πρόβλεψη αλλαγών τιμών, καθώς και τεχνικές εξόρυξης δεδομένων για τη συλλογή και ανάλυση τεράστιων ποσοτήτων δεδομένων.

Η αποτελεσματικότητα των συστημάτων παρακολούθησης τιμών έχει επαληθευτεί σε διάφορους τομείς, υποδεικνύοντας την ευρύτερη εφαρμογή και συνάφειά τους. Αυτό καθιστά τη μελέτη τέτοιων συστημάτων και τη βελτιστοποίησή τους σημαντικό μέρος του σύγχρονου εμπορίου και της λήψης αποφάσεων από τους καταναλωτές.

1.8 Web Scrapping και Τεχνολογίες

Η εξέλιξη των τεχνολογιών web scraping ήταν ταχεία, λόγω των αυξανόμενων ποσοτήτων δεδομένων που παράγονται στο διαδίκτυο. Το web scraping, περιλαμβάνει την ανάκτηση δεδομένων από ιστότοπους. Αυτή η ενότητα στοχεύει να παρέχει μια ολοκληρωμένη βιβλιογραφική ανασκόπηση των τεχνολογιών scraping που σχετίζονται με το σύστημα παρακολούθησης τιμών μας.

Η εξέλιξη του web scraping

Η έννοια του web scraping δεν είναι νέα. υπάρχει από τις πρώτες μέρες του Διαδικτύου. Ωστόσο, τα σύγχρονα εργαλεία και βιβλιοθήκες web scraping έχουν προχωρήσει σημαντικά στις δυνατότητές τους. Τα πρώτα scrapers βασίζονταν σε απλά αιτήματα HTTP και κανονικές εκφράσεις (RegEx), αλλά οι σημερινές τεχνολογίες είναι πολύ πιο εξελιγμένες, συχνά ενσωματώνουν προγράμματα περιήγησης, browser extensions και πολλά άλλα.

Τεχνολογίες που χρησιμοποιούνται για web scraping

Τα εργαλεία web scraping μπορούν να κατηγοριοποιηθούν ευρέως σε δύο τύπους: βασισμένα σε κώδικα και βασισμένα σε GUI. Τα εργαλεία που βασίζονται σε κώδικα, όπως το BeautifulSoup και το Scrapy στην Python, προσφέρουν μεγαλύτερη ευελιξία αλλά απαιτούν δεξιότητες προγραμματισμού. Εργαλεία που βασίζονται σε GUI, όπως το Octoparse ή το WebHarvy, επιτρέπουν στους χρήστες να ανακτούν δεδομένα μέσω μιας γραφικής διεπαφής, κάτι που μπορεί να είναι επωφελές για όσους δεν έχουν γνώσεις προγραμματισμού.

Προκλήσεις στο web scraping

Το web scraping παίζει σημαντικό ρόλο στα συστήματα παρακολούθησης τιμών. Αναφέρεται στην αυτοματοποιημένη διαδικασία εξαγωγής δεδομένων από ιστότοπους για περαιτέρω ανάλυση ή αποθήκευση. Το θέμα έχει μελετηθεί εκτενώς, αποκαλύπτοντας τόσο τα οφέλη όσο και τις προκλήσεις του.

Οι τεχνικές web scraping έχουν εξελιχθεί με την πάροδο του χρόνου, που κυμαίνονται από απλή ανάλυση HTML έως πιο σύνθετες μεθόδους που μπορούν να χειριστούν ιστοσελίδες με JavaScript. Η εφαρμογή αυτών των τεχνικών είναι απαραίτητη για τη συνιστώσα συλλογής δεδομένων των συστημάτων παρακολούθησης τιμών.

Ωστόσο, προκλήσεις όπως μηχανισμοί anti-bot, αποκλεισμός IP και CAPTCHA συχνά εμποδίζουν τη διαδικασία scraping. Αυτός είναι ο λόγος για τον οποίο το σύγχρονο web scraping χρησιμοποιεί συχνά πιο προηγμένες τεχνικές, όπως IP-rotation, proxy networks, ακόμη και αλγόριθμους μηχανικής μάθησης για να παρακάμψει τέτοια εμπόδια.

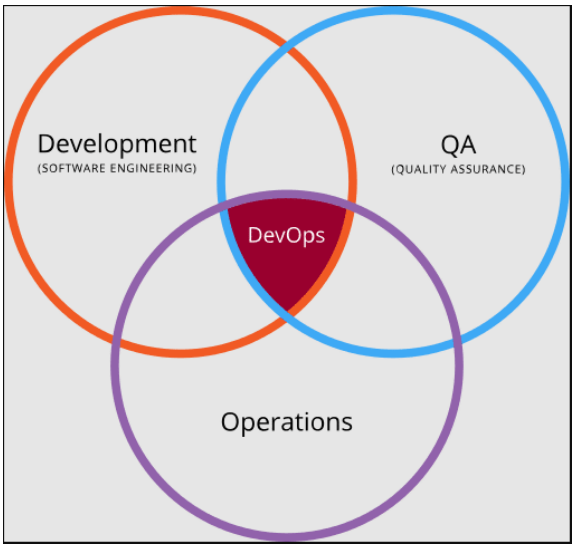
Δεδομένου ότι το web scraping είναι αναπόσπαστο στοιχείο για τη συλλογή των δεδομένων τιμών που τροφοδοτούν τα συστήματα παρακολούθησης, η κατανόηση των μηχανισμών, των δυνατοτήτων και των περιορισμών του είναι ζωτικής σημασίας για αυτήν τη μελέτη.

1.9 DevOps σε scalable συστήματα

Το DevOps, ως πρακτική, έχει αναδειχθεί ως μια επαναστατική προσέγγιση στον κόσμο της ανάπτυξης λογισμικού και των λειτουργιών. Η σημασία των DevOps δεν μπορεί να υπερεκτιμηθεί, ειδικά σε έργα που απαιτούν υψηλό scaling, availability και στιβαρότητα, όπως το σύστημα παρακολούθησης τιμών μας.

Οι πυλώνες του DevOps

Το DevOps στοχεύει στη διάσπαση του «τοίχους» μεταξύ των ομάδων ανάπτυξης και λειτουργίας της πλατφόρμας. Δίνει έμφαση σε μια κουλτούρα συνεργασίας και σε εργαλεία που βελτιστοποιούν ολόκληρη τη γραμμή παράδοσης λογισμικού. Οι βασικές αρχές περιλαμβάνουν, μεταξύ άλλων, την αυτοματοποίηση, το continuous integration, and continuous deployment. Ο όρος επινοήθηκε για να περιγράψει ένα σύνολο βέλτιστων πρακτικών που ενθαρρύνουν ταχύτερους κύκλους ανάπτυξης και πιο αξιόπιστα releases.



2.4.1: DevOps Venn diagram 1

Continuous Integration/Continuous Deployment (CI/CD)

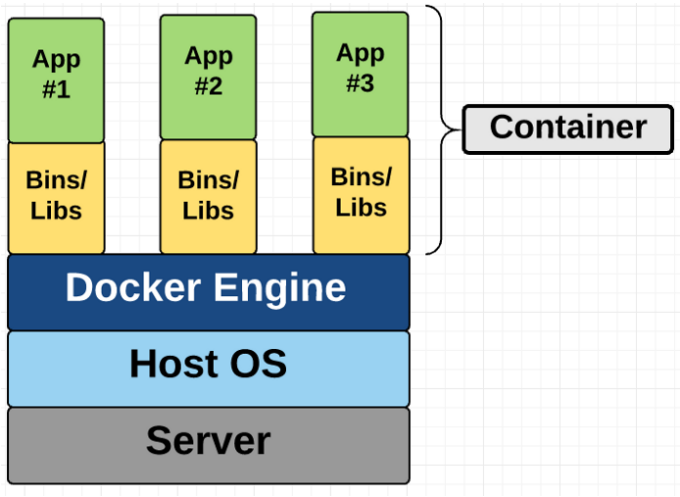
Ένας από τους ακρογωνιαίους λίθους του DevOps είναι το CI/CD. Αυτό περιλαμβάνει την αυτόματη δημιουργία και δοκιμή αλλαγών του κώδικα, επιτρέποντας ταχύτερο εντοπισμό και διόρθωση σφαλμάτων. Για scalable συστήματα όπως το έργο μας, το CI/CD δεν είναι πολυτέλεια αλλά αναγκαιότητα. Επιτρέπει την ταχεία ανάπτυξη αλλαγών του κώδικα χωρίς χειροκίνητη παρέμβαση, μειώνοντας έτσι τον χρόνο που χρειάζεται για να βγουν οι νέες εκδόσεις του συστήματος στους χρήστες στο production σύστημα, και διασφαλίζοντας ότι το σύστημα μπορεί να προσαρμοστεί γρήγορα σε νέες απαιτήσεις ή προκλήσεις

Σχήμα 2.4.2: CI/CD pipeline

Containerization

Το Containerization είναι ένας άλλος πυλώνας στο οικοσύστημα του DevOps. Τα προγράμματα που υλοποιούν τα containers, όπως το Docker, επιτρέπουν στους προγραμματιστές να συσκευάσουν μια εφαρμογή και τα dependencies της σε ένα "κοντέινερ", διασφαλίζοντας ότι θα εκτελείται το ίδιο ανεξάρτητα από το πού έχει αναπτυχθεί. Αυτό είναι ζωτικής σημασίας για scalable συστήματα, όπου ο φόρτος εργασίας μπορεί να κατανεμηθεί σε πολλούς server ή ακόμη και σε data centers.Πρακτικά το containerization λύνει ένα πρόβλημα που έχει συμβεί σχεδόν σε όλους τους προγραμματιστές όπου όταν

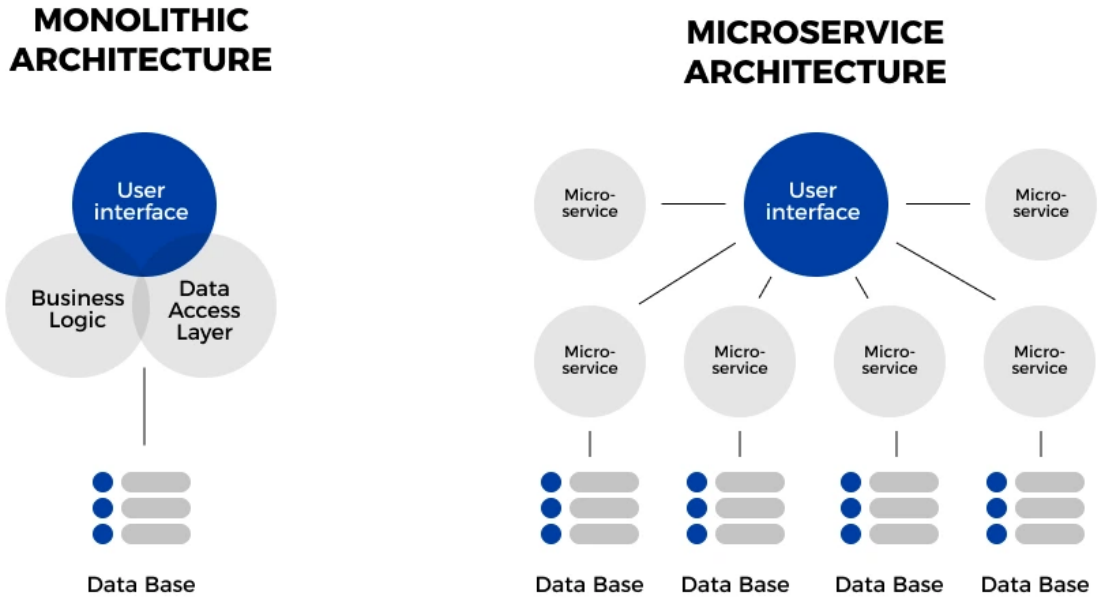
πακετάρουν ένα software σε ένα άλλο μηχάνημα αντιμετωπίζουν είτε σφάλματα είτε διαφορετική συμπεριφορά. Δίνει τέλος στο “it works on my machine”.



Σχήμα 2.4.3:Containerization arhitecture

Microservices και Αρχιτεκτονική

Σε ένα scaleable σύστημα, η υιοθέτηση μιας αρχιτεκτονικής microservice μπορεί να είναι επωφελής. Σε αντίθεση με την αρχιτεκτονική monolith όπου όλα τα στοιχεία είναι αλληλένδετα, τα microservices χωρίζουν την εφαρμογή σε μικρές, loosely coupled υπηρεσίες που μπορούν να αναπτυχθούν και να κλιμακωθούν ανεξάρτητα. Αυτό είναι ιδιαίτερα σημαντικό για το σύστημα παρακολούθησης τιμών που διαθέτουμε, όπου διαφορετικά στοιχεία όπως το scraping, το backend και το scheduling ενδέχεται να έχουν διαφορετικές απαιτήσεις κλιμάκωσης.



2.4.4: Monolith vs Microservice system 1

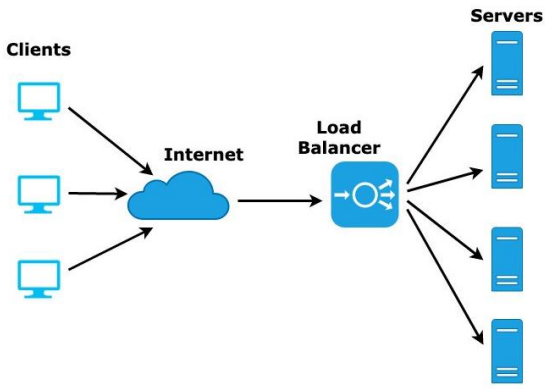
Monitoring, Logging και Tracing

Το DevOps δεν αφορά μόνο την ανάπτυξη, αλλά περιλαμβάνει επίσης τη συνεχή παρακολούθηση της υγείας του συστήματος. Τα εργαλεία για logging, την monitoring και tracing είναι ανεκτίμητα για τη διατήρηση της υψηλής διαθεσιμότητας και απόδοσης. Παρέχουν πληροφορίες σχετικά με τη συμπεριφορά του συστήματος, βοηθώντας τους διαχειριστές να εντοπίσουν και να επιλύσουν ζητήματα προτού επηρεάσουν τους τελικούς χρήστες.

Αυτή η εις βάθος προσέγγιση στις πρακτικές DevOps χρησιμεύει για να υπογραμμίσει τον κρίσιμο ρόλο τους στην ανάπτυξη και τη συντήρηση κλιμακούμενων συστημάτων. Επιτρέπει την απρόσκοπτη ενσωμάτωση νέων λειτουργιών, διασφαλίζει την αξιοπιστία του συστήματος και επιτρέπει την κλιμάκωση που απαιτείται για τον χειρισμό μεγάλου όγκου εργασιών web scrapping και user requests στο έργο μας.

1.10 Scalability και Load Balancing

Η κατανόηση του scalability και load balancing είναι πολύ σημαντική για κάθε σύγχρονη εφαρμογή που βασίζεται στο web, ειδικά για ένα έργο όπως το σύστημα παρακολούθησης τιμών μας που ασχολείται με σημαντικό όγκο web scraping tasks. Η επόμενη ενότητα εμβαθύνει στις τεχνολογίες και τις μεθοδολογίες που περιβάλλουν αυτές τις κρίσιμες πτυχές



Σχήμα 2.5: Load Balancing example

Η σημαντικότητα του scaling

Το scaling διασφαλίζει ότι ένα σύστημα μπορεί να χειριστεί τον αυξημένο φόρτο εργασίας, χωρίς να επηρεάζει την απόδοση του συστήματος η των χρηστών. Όποτε ο όγκος των αναγκών της πλατφόρμας αυξάνεται τότε χρειάζονται όλο και περισσότεροι servers για να μπορέσουν να ανταπεξέλθουν. Το scaling μπορεί να πραγματοποιείτε είτε χειροκίνητα (adhoc) είτε αυτοματοποιημένα (auto scaling). Τέλος υπάρχουν δύο ειδών scaling, το horizontal και το vertical scaling.

- **Horizontal scaling**: Αυξάνουμε τα replicas της εφαρμογής π.χ. από 2 σε 5
- **Vertical scaling**: Αυξάνουμε την CPU/RAM του server.

Τεχνικές Load Balancing

Το load balancing είναι συχνά ο ακρογωνιαίος λίθος του scaling. Διανέμει το incoming traffic των εφαρμογών ή δικτύου σε πολλούς διακομιστές, διασφαλίζοντας ότι κανένας διακομιστής δεν κατακλύζεται με υπερβολικό φορτίο. Αλγόριθμοι όπως Round Robin, Least Connections και IP Hashing χρησιμοποιούνται συνήθως σε λύσεις εξισορρόπησης φορτίου.

Στρατηγικές Auto-Scaling

Τα περισσότερα σύγχρονα περιβάλλοντα cloud προσφέρουν υπηρεσίες auto scaling. Το auto-scaling προσαρμόζει αυτόματα τον αριθμό των server με βάση την τρέχουσα φόρτωση του συστήματος. Για το σύστημα παρακολούθησης τιμών που διαθέτουμε, το auto-scaling διασφαλίζει ότι μπορούμε να διαχειριστούμε μεγάλο όγκο web scraping tasks και σε user requests χωρίς χειροκίνητη παρέμβαση.

1.11 Επίλογος

Συνοπτικά, το συγκεκριμένο κεφάλαιο χρησιμεύει για την κατανόηση των διαφόρων τεχνολογιών και μεθοδολογιών που είναι σημαντικές για την ανάπτυξη και τη λειτουργία του συστήματος παρακολούθησης τιμών. Από την εμβάθυνση στις περίπλοκες λεπτομέρειες των τεχνολογιών web scraping, τις ηθικές και νομικές θεωρήσεις τους, μέχρι την εξερεύνηση των πεδίων του scaling και load balancing, αυτό το κεφάλαιο στοχεύει να παρέχει ένα ολοκληρωμένο υπόβαθρο πάνω στο οποίο θα ξεδιπλωθεί η υπόλοιπη πτυχιακή.

Μέσω αυτού του κεφαλαίου, θέσαμε το υπόβαθρο για τα επόμενα κεφάλαια όπου θα εμβαθύνουμε στις ιδιαιτερότητες του έργου μας, από την αρχιτεκτονική του συστήματος έως τις πρακτικές DevOps, έχοντας πάντα κατά νου το θεωρητικό υπόβαθρο που έχουμε δημιουργήσει εδώ.

Μεθοδολογία

1.12 Εισαγωγή

Το κεφάλαιο μεθοδολογία χρησιμεύει ως η ραχοκοκαλιά της εργασίας. Παρέχει ένα σχέδιο για το σχεδιασμό, την εφαρμογή και την αξιολόγηση του συστήματος παρακολούθησης των τιμών. Σε ένα έργο αυτής της πολυπλοκότητας, που περιλαμβάνει διάφορα στοιχεία όπως scaling, web scraping, προγραμματισμός εργασιών, load balancing και monitoring, είναι επιτακτική ανάγκη να υιοθετηθεί μια συνεκτική, πειθαρχημένη προσέγγιση για την επίλυση προβλημάτων. Η μεθοδολογία μας περιγράφει τα διαδοχικά και παράλληλα βήματα που λαμβάνονται για να διασφαλιστεί ότι το σύστημα είναι δυνατό, επεκτάσιμο και φιλικό προς τον χρήστη.

Αυτό το κεφάλαιο στοχεύει να απομυθοποιήσει τα πολλαπλά επίπεδα του έργου μας, αναλύοντας το σκεπτικό πίσω από τεχνολογία που επιλέχθηκε. Με αυτόν τον τρόπο, προσδίδουμε αξιοπιστία στην έρευνά μας και δημιουργούμε ένα αναπαραγόμενο μοντέλο για μελλοντική εργασία σε αυτόν τον τομέα. Θα εμβαθύνουμε στα συγκεκριμένα εργαλεία και τις τεχνολογίες που χρησιμοποιούνται—όπως το Node.js για το web scraping, το RabbitMQ για την ουρά μηνυμάτων και το Docker για τη δημιουργία containers—και θα συζητήσουμε γιατί επιλέχθηκαν έναντι άλλων διαθέσιμων επιλογών. Θα εξηγήσουμε επίσης πώς αυτά τα εργαλεία διαμορφώθηκαν και βελτιστοποιήθηκαν ώστε να λειτουργούν συνεργατικά, καλύπτοντας ή και υπερβαίνοντας τα κριτήρια απόδοσης του έργου.

Θα βρείτε μια λεπτομερή συζήτηση σχετικά με τον τρόπο με τον οποίο τα δεδομένα ανακτήθηκαν από το web, υποβλήθηκαν σε επεξεργασία και στη συνέχεια διατέθηκαν στο backend σύστημα για αποθήκευση. Θα βρείτε επίσης πληροφορίες σχετικά με τον τρόπο με τον οποίο οι διαχειριστές μπορούν να παρακολουθούν διάφορες μετρήσεις και στατιστικά στοιχεία μέσω των dashboards.

Τώρα, θα αναλύσουμε τις τεχνολογίες που αποτελούν το έργο παρακολούθησης τιμών, αναλύοντας την πολυπλοκότητά του σε κατανοητά μέρη και αναδεικνύοντας την κομψότητα ενός καλά σχεδιασμένου συστήματος.

1.13 Η Προσέγγιση της Μεθοδολογίας

Η προσέγγισή μας επηρεάστηκε βαθιά από την πρακτική, επαγγελματική εμπειρία και των μελών της ομάδας μας. Καθένας από εμάς έφερε στο τραπέζι μια πληθώρα γνώσεων του κλάδου, πολλές από αυτές αποκτήθηκαν από την ανάπτυξη παρόμοιων συστημάτων σε επαγγελματικό πλαίσιο. Η μεθοδολογία μας, επομένως, δεν ήταν απλώς θεωρητική, αλλά βασίστηκε σε πραγματικές εφαρμογές.

Η επιλογή εργαλείων και τεχνολογιών, όπως το Node.js για το web scraping, το RabbitMQ για την ουρά, το Docker για containerization και το Grafana για τους πίνακες dashboard, καθοδηγείται από την αποδεδειγμένη αξιοπιστία και αποτελεσματικότητά τους σε παρόμοια έργα. Αυτές οι επιλογές δεν έγιναν αυθαίρετα αλλά ήταν αποτέλεσμα εκτεταμένης εμπειρίας και βαθιάς κατανόησης του τεχνολογικού τοπίου.

Σκεπτικό και αιτιολόγηση

Το σκεπτικό για την υιοθέτηση αυτής της ρεαλιστικής προσέγγισης ήταν απλό: κατά την επίλυση σύνθετων προβλημάτων σε ένα ταχέως εξελισσόμενο πεδίο όπως το DevOps, η εμπειρία είναι συχνά ο καλύτερος δάσκαλος. Αξιοποιώντας την πρακτική εμπειρία σε πολλά projects, μπόρεσα να παρακάμψω πολλές από τις παγίδες που συναντάμε συχνά σε έργα αυτού του μεγέθους και πολυπλοκότητας.

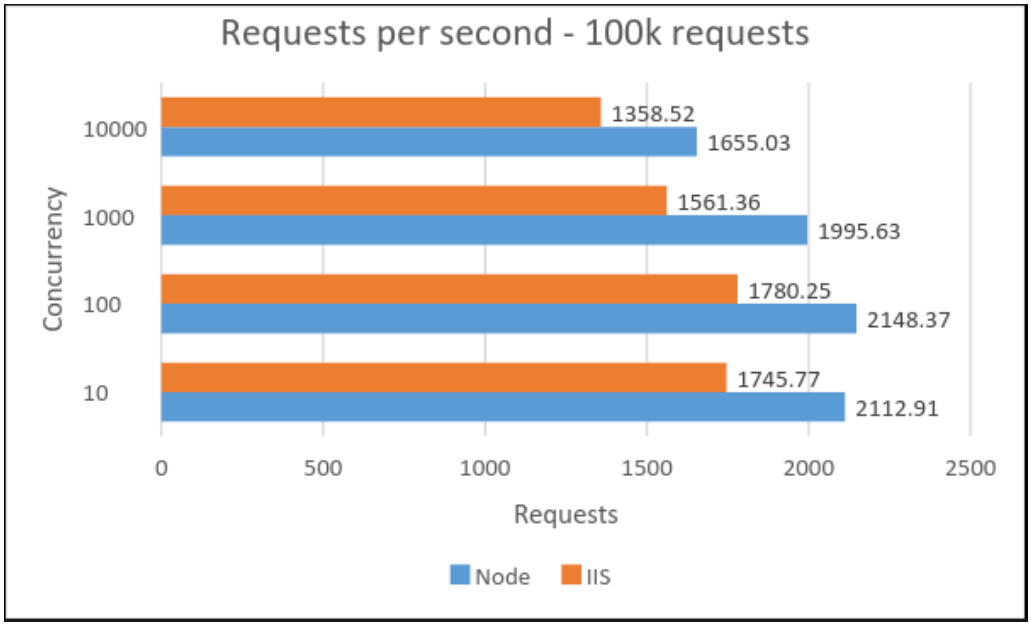
Εστιάζοντας σε αυτό που έχει αποδειχθεί ότι λειτουργεί καλά σε πρακτικές εφαρμογές, στοχεύσαμε να δημιουργήσουμε ένα ισχυρό, επεκτάσιμο σύστημα ικανό να ανταποκριθεί στις απαιτήσεις τόσο των χρηστών όσο και των διαχειριστών.

1.14 Τα εργαλεία και οι τεχνολογίες που εφαρμόστηκαν

Σε ένα έργο τόσο περίπλοκο όσο ένα scalable σύστημα παρακολούθησης τιμών, η επιλογή εργαλείων και τεχνολογιών δεν είναι απλώς μια τεχνική απόφαση, αλλά μια στρατηγική. Τα εργαλεία που επιλέξαμε δεν είναι απλά μεμονωμένα κομμάτια λογισμικού, αλλά στοιχεία ενός ευρύτερου οικοσυστήματος που έχουμε προσαρμόσει στις συγκεκριμένες ανάγκες μας. Αυτές οι επιλογές καθοδηγήθηκαν από τη συνδυασμένη επαγγελματική εμπειρία των μελών της ομάδας μας σε IT και DevOps, καθιστώντας την επιλογή ρεαλιστική και εστιασμένη στην αποδεδειγμένη αποτελεσματικότητα.

Node.js για Scraping και Scheduling

Το Node.js επιλέχθηκε για την αρχιτεκτονική non-blocking, event-driven, η οποία είναι ιδανική για εργασίες όπως web scraping και ο προγραμματισμός εργασιών που απαιτούν λειτουργίες υψηλού I/O. Η ασύγχρονη φύση του διασφαλίζει ότι το σύστημα μπορεί να χειριστεί μεγάλο αριθμό αιτημάτων χωρίς να υποβαθμίζει την απόδοση. Επιπλέον, έχει εύκολο συντακτικό και καλές βιβλιοθήκες που συνδέονται με RabbitMQ και πολύ μικρό bundle size πράγμα που του επιτρέπει να τρέχει σε μηχανήματα με πολύ μικρές προδιαγραφές διευκολύνοντας έτσι και το containerization.



Σχήμα 3.3.1: NodeJs vs IIS requests/sec

RabbitMQ για ουρές

Η δημοτικότητα και το robustness του RabbitMQ για εφαρμογές το έκαναν εξαιρετική επιλογή για τις ανάγκες μας στην ουρά. Η ικανότητά του να χειρίζεται σενάρια υψηλής απόδοσης χωρίς εμπόδια διασφαλίζει ότι το σύστημά μας παραμένει ανταποκρινόμενο ακόμη και κάτω από μεγάλο φορτίο.

Το RabbitMQ είναι ένα λογισμικό message broker που διευκολύνει την ανταλλαγή μηνυμάτων μεταξύ διαφορετικών εφαρμογών, υπηρεσιών ή συστημάτων. Εφαρμόζει το Advanced Message Queuing Protocol (AMQP) και υποστηρίζει πολλά άλλα πρωτόκολλα όπως MQTT, STOMP και HTTP.

Το RabbitMQ προσφέρει μια ποικιλία χαρακτηριστικών που το καθιστούν ιδανική λύση για το σύστημα μας:

- **Ανθεκτικότητα:** Τα μηνύματα μπορούν να διατηρηθούν στο δίσκο, επιτρέποντας την ανάκτηση.
- **Load Balancing**: Διανέμει μηνύματα σε πολλούς καταναλωτές για να εξισορροπήσει το φορτίο.
- **Scalability**: Εύκολο horizontal και vertical scaling.
- **Management UI**: Προσφέρει ένα εύχρηστο περιβάλλον χρήστη που βασίζεται στο web για παρακολούθηση και διαχείριση.

Docker για containerization

Το Docker επιλέχθηκε για την αποτελεσματικότητά του στη δημιουργία, την ανάπτυξη και την εκτέλεση εφαρμογών με τη χρήση κοντέινερ. Το Containerization επιτρέπει στην εφαρμογή μας να εκτελείται με συνέπεια σε διαφορετικά υπολογιστικά περιβάλλοντα, μειώνοντας σημαντικά τα ζητήματα "λειτουργεί στον υπολογιστή μου".

Grafana για administrator dashboards

Για παρακολούθηση και οπτικοποίηση δεδομένων σε πραγματικό χρόνο, το Grafana ήταν το βασικό μας εργαλείο. Ενσωματώνεται πανεύκολα με διάφορες πηγές δεδομένων και παρέχει ένα ευρύ φάσμα επιλογών οπτικοποίησης, καθιστώντας το εξαιρετικά προσαρμόσιμο για διαφορετικές ανάγκες παρακολούθησης. Στη δική μας περίπτωση το χρησιμοποιούμε για να τρέχουμε queries πάνω στα logs του scraper και scheduler και για να ενημερώνουμε τους διαχειριστές με alerts στο email τους, όταν εμφανίζονται σφάλματα.

Loki για αποθήκευση και aggregation των logs

Η αποτελεσματική αποθήκευση logs και η γρήγορη αναζήτηση του Loki το έκαναν ιδανική επιλογή για τη συγκέντρωση logs από διαφορετικά μέρη του συστήματος. Ο ελαφρύς σχεδιασμός και οι δυνατότητες ενσωμάτωσής του με το Grafana το έκαναν μια απρόσκοπτη προσθήκη στο κιτ εργαλείων παρακολούθησης.

Express.js HTTP Server για τον Scraper

Η εφαρμογή Express μπορεί να ακούσει για εισερχόμενα αιτήματα HTTP από το backend μέσω του εσωτερικού μας δικτύου, επιτρέποντας το άμεσο scraping σε πραγματικό χρόνο. Αντίθετα, ο scraper αλλάζοντας μερικά environmental variables μπορεί να τεθεί λειτουργία όπου ακούει εργασίες από μια ουρά RabbitMQ, προσφέροντας μια πιο ασύγχρονη, προγραμματισμένη προσέγγιση για το web scraping. Αξιοποιώντας τα environmental variables για τον έλεγχο της συμπεριφοράς της εφαρμογής, μπορούμε να προσαρμόζουμε εύκολα και γρήγορα σε διαφορετικές επιχειρησιακές ανάγκες χωρίς να χρειάζεται να αλλάξουμε τον κώδικα, διευκολύνοντας έτσι ένα πιο agile περιβάλλον ανάπτυξης.

GitHub Actions για CI/CD

Χρησιμοποίησα το GitHub Actions για Continuous Integration and Continuous Deployment (CI/CD). Αυτό μας επέτρεψε να αυτοματοποιήσουμε τη διαδικασία παράδοσης λογισμικού, καθιστώντας την ταχύτερη, πιο συνεπή και λιγότερο επιρρεπή σε σφάλματα.

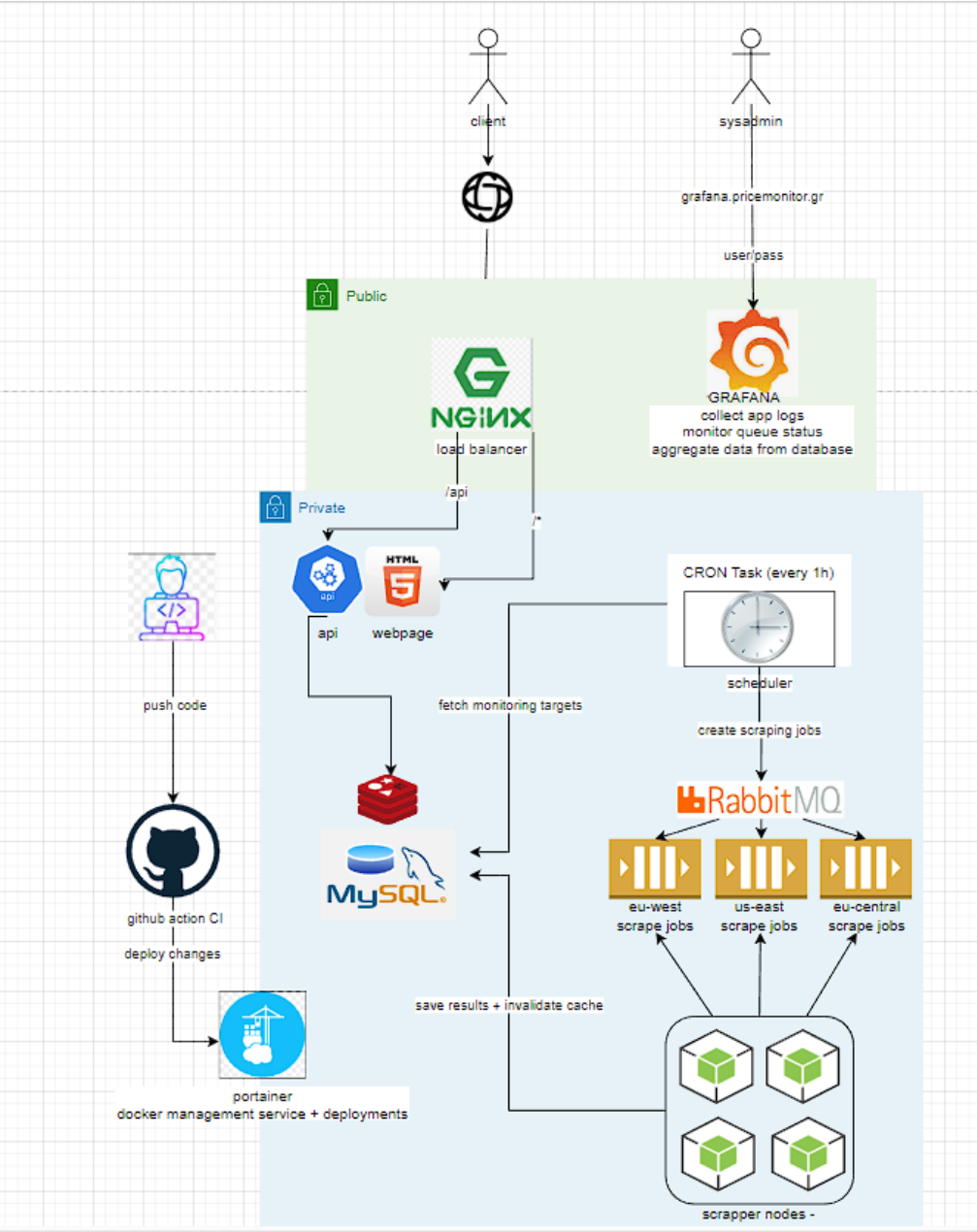
Nginx για Load Balancing και Reverse Proxy

Το Nginx χρησιμοποιήθηκε ως Load Balancer και Reverse Proxy για τη διανομή της εισερχόμενης κίνησης εφαρμογών ή δικτύου σε πολλούς κόμβους scraper. Η απόδοση, η αξιοπιστία και η ευελιξία του το καθιστούν βασικό στοιχείο στις σύγχρονες αρχιτεκτονικές web.

Αρχιτεκτονική Συστήματος

Σε αυτό το σημείο παρουσιάζεται το σχήμα με τις αρχιτεκτονικές επιλογές που έγιναν για την κατασκευή της πλατφόρμας παρακολούθησης τιμών μας, διευκρινίζοντας πώς αυτές οι αποφάσεις διευκόλυναν το scaling, την αξιοπιστία και την αποτελεσματικότητα του συστήματος.

Μια σωστή αρχιτεκτονική καθοδηγεί τη διαδικασία ανάπτυξης, διασφαλίζοντας ότι κάθε στοιχείο έχει σχεδιαστεί και υλοποιηθεί βέλτιστα. Ταυτόχρονα, επιτρέπει στους ενδιαφερόμενους να κατανοήσουν τη δομή και τις αλληλεπιδράσεις υψηλού επιπέδου εντός του συστήματος, διευκολύνοντας τη λήψη αποφάσεων και τον μελλοντικό σχεδιασμό.



Σχήμα 3.3.9: Αρχιτεκτονική Συστήματος

1.15 Συλλογή δεδομένων και Ανάλυση

Στη σφαίρα του web scraping και της παρακολούθησης των τιμών, τα δεδομένα είναι αναμφίβολα το βασικό στοιχείο. Το έργο αξιοποίησε μια ολοκληρωμένη στρατηγική συλλογής και ανάλυσης δεδομένων για τη διευκόλυνση των στόχων του. Η ποιότητα και η επικαιρότητα των δεδομένων που αφαιρούνται από το web διαδραματίζουν κρίσιμο ρόλο στη διασφάλιση ότι το σύστημα παρέχει ακριβείς και ενημερωμένες πληροφορίες τιμών.

Συλλογή δεδομένων

Για τη συλλογή δεδομένων μας, βασιστήκαμε κυρίως σε scrapers που βασίζονται στο Node.js. Αυτά οι scrapers έχουν προγραμματιστεί να συλλέγουν δεδομένα από συγκεκριμένες πηγές σε προκαθορισμένα χρονικά διαστήματα. Η χρήση του RabbitMQ για εργασίες στην ουρά εξασφάλισε ότι ήμασταν σε θέση

να χειριστούμε μεγάλο αριθμό αιτημάτων χωρίς να πέσει το σύστημα. Τα δεδομένα συλλέγονται σε δομημένη μορφή, καθιστώντας ευκολότερη την ανάλυση και την αποθήκευση για μελλοντική χρήση.

Αποθήκευση δεδομένων

Μετά την επικύρωση, τα δεδομένα αποθηκεύονται σε μια βάση δεδομένων. Η αρχιτεκτονική επιτρέπει την ανάκτηση των δεδομένων σε πραγματικό χρόνο από το frontend, καθώς επίσης επιτρέπει στους διαχειριστές να εκτελούν πιο σύνθετα ερωτήματα.

Ανάλυση δεδομένων

Από πλευρά των διαχειριστών, χρησιμοποιούμε πίνακες εργαλείων Grafana για την εμφάνιση διαφόρων μετρήσεων και στατιστικών στοιχείων, προσφέροντας σε πραγματικό χρόνο την απόδοσης του συστήματος και των δεδομένων που συλλέγει. Το Loki χρησιμοποιείται για αποθήκευση και συνάθροιση αρχείων καταγραφής, το οποίο βοηθά στον εντοπισμό σφαλμάτων και στην παρακολούθηση της απόδοσης.

Ηθικά ζητήματα

Εφόσον το web scraping περιλαμβάνει την αλληλεπίδραση με ιστότοπους τρίτων, φροντίσαμε να εφαρμόσουμε τεχνικές anti-bot για να ελαχιστοποιήσουμε τον κίνδυνο αποκλεισμού από αυτές τις πηγές.

1.16 Επίλογος

Συνοπτικά, οι μεθοδολογίες που αναπτύχθηκαν σε αυτή τη διατριβή επιλέχθηκαν προσεκτικά για να ευθυγραμμιστούν με τις συγκεκριμένες απαιτήσεις ενός επεκτάσιμου, αποτελεσματικού και ηθικού συστήματος παρακολούθησης των τιμών. Μέσω ενός συνδυασμού τεχνολογιών και δοκιμασμένων στρατηγικών, το έργο πέτυχε τους στόχους του, προσφέροντας παράλληλα ευελιξία για μελλοντικές επεκτάσεις.

Η χρήση του Node.js για scraping, του RabbitMQ για την ουρά εργασιών, του Docker για το containeraztion, του Grafana για παρακολούθηση σε πραγματικό χρόνο, του Loki για αποθήκευση αρχείων καταγραφής και του Express ως διακομιστή HTTP, συνέβαλαν στην επιτυχία του έργου με τον δική του μοναδικό τρόπο. Αυτά τα εργαλεία δεν επιλέχθηκαν τυχαία, αλλά ήταν αποτέλεσμα της επαγγελματικής εμπειρίας της ομάδας και της βαθιάς κατανόησης των αναγκών του έργου.

Επιπλέον, το έργο αποτελεί παράδειγμα μιας επιτυχημένης ομαδικής προσπάθειας, φέρνοντας σε επαφή ειδικούς στην ανάπτυξη frontend και backend, scraping και DevOps, για να δημιουργήσουν μια λύση.

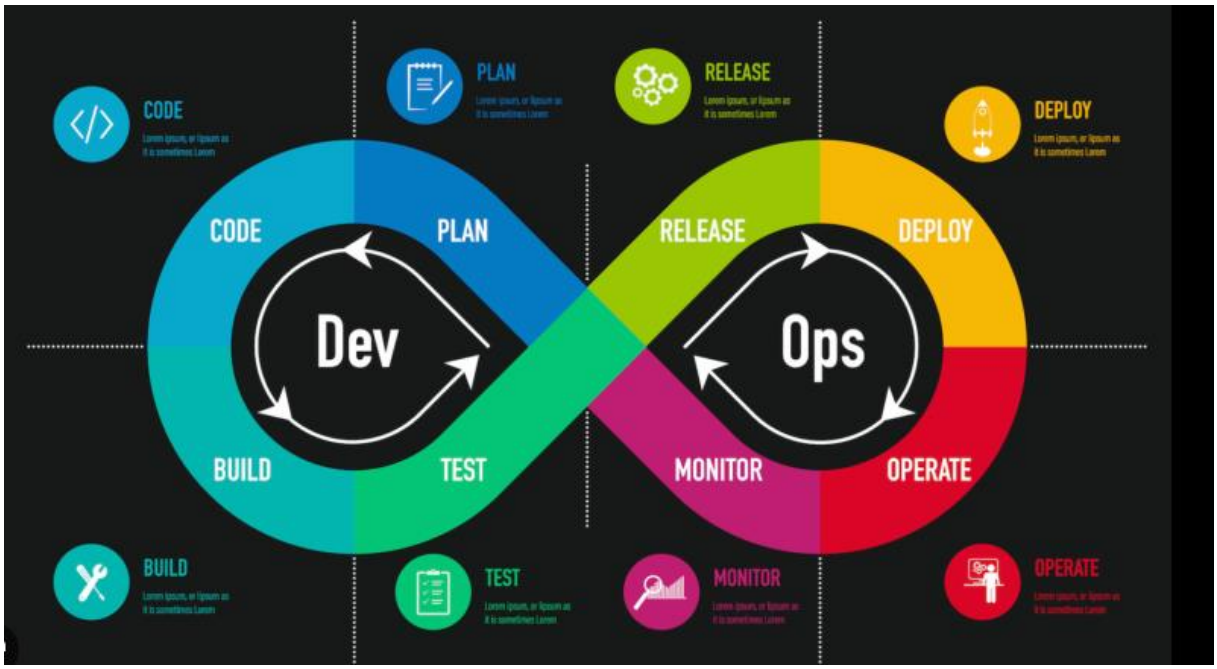
Κεφάλαιο 4ο: Τι είναι το DevOps

4.1 Εισαγωγή

Ο κύκλος ζωής του DevOps συνήθως περιλαμβάνει τα ακόλουθα βήματα ή φάσεις, τα οποία δημιουργούν έναν συνεχή κύκλο με στόχο τη συνεχή βελτίωση. Τα περισσότερα από αυτά θα αναλυθούν παρακάτω:

- **Plan**: Η αρχική φάση όπου συγκεντρώνονται οι απαιτήσεις της εφαρμογής και γίνεται ο σχεδιασμός για τον κώδικα και την υποδομή.
- **Code**: Σε αυτή τη φάση λαμβάνει χώρα η πραγματική ανάπτυξη κώδικα. Οι προγραμματιστές γράφουν κώδικα σύμφωνα με τις προγραμματισμένες προδιαγραφές και απαιτήσεις.
- **Build**: Ο πηγαίος κώδικας μεταγλωττίζεται σε εκτελέσιμο κώδικα σε αυτό το στάδιο. Αυτή η διαδικασία κατασκευής μπορεί επίσης να περιλαμβάνει άλλες εργασίες, όπως το linting κώδικα, την εκτέλεση unit test κ.λπ.
- **Test**: Η κατασκευή δοκιμάζεται σε πολλά σενάρια για να βεβαιωθείτε ότι είναι έτοιμη για παραγωγή. Αυτή η φάση είναι κρίσιμη για τη διασφάλιση ποιότητας.
- **Release**: Σε αυτή τη φάση η καινούρια έκδοση του προτζεκτ είναι έτοιμη για να μεταφερθεί στα επόμενα βήματα. Αυτό μπορεί να γίνει είτε αυτόματα είτε χειροκίνητα.
- **Deploy**: Η νέα εφαρμογή μεταφέρετε στο περιβάλλον production όπου μπορεί να χρησιμοποιηθεί από τους τελικούς χρήστες.
- **Operate**: Παρακολούθηση της απόδοσης, των σφάλματων και άλλων μετρήσεων. Αυτά τα δεδομένα είναι πολύ χρήσιμα για τη διασφάλιση της ομαλής λειτουργίας του συστήματος.
- **Monitor**: Πραγματοποιείται συνεχής παρακολούθηση για την αξιοπιστία του συστήματος, την απόδοση και άλλες σημαντικές μετρήσεις.
- **Feedback and Iterate**: Feedback των τελικών χρηστών και των διαχειριστών συστημάτων, και επιστροφή στη φάση σχεδιασμού (plan) για να εφαρμοστούν οι απαραίτητες αλλαγές.

Κάθε ένα από αυτά τα βήματα είναι διασυνδεδεμένα και οι αλλαγές σε ένα μπορούν να προκαλέσουν αλλαγές σε άλλα, καθιστώντας ολόκληρο τον κύκλο ζωής έναν συνεχή βρόχο με στόχο τη βελτίωση της εφαρμογής.



Σχήμα 5.1: Διάγραμμα DevOps

4.2 Σχεδιασμός (Plan)

Το πλάνο και αρχική ιδέα ενός νέου feature της εφαρμογής είναι η αρχική φάση του κύκλου ζωής του DevOps, που θέτει τις βάσεις για όλα τα επόμενα στάδια. Συχνά παραβλέπεται, αλλά το στάδιο του σχεδιασμού είναι ζωτικής σημασίας επειδή καθορίζει την τροχιά για το πώς θα αναπτυχθεί και θα συντηρηθεί το σύστημα. Ένας κοινός τρόπος όπου λειτουργούν πολλές ομάδες είναι προγραμματίζοντας τα λεγόμενα sprints που διαρκούν 1-2 εβδομάδες και έχουν ως στόχο την ολοκλήρωση συγκεκριμένων tasks μέσα σε αυτό το χρονικό διάστημα.

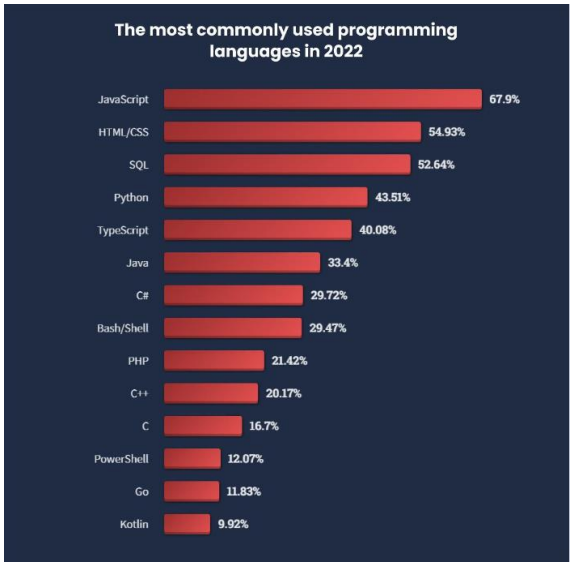
Στο πλαίσιο της πλατφόρμας παρακολούθησης τιμών μας, η φάση σχεδιασμού ήταν κρίσιμη για τον καθορισμό των απαιτήσεων, την κατανομή πόρων και τον καθορισμό χρονοδιαγραμμάτων. Δεδομένου ότι το έργο ήταν μια συνεργασία μεταξύ διαφορετικών μελών της ομάδας που ειδικεύονται στο front-end, το back-end και το scrapping, ο αποτελεσματικός σχεδιασμός ήταν πρωταρχικής σημασίας για να εξασφαλιστεί η απρόσκοπτη ενσωμάτωση αυτών των διαφορετικών στοιχείων.

Χρησιμοποιήθηκαν εργαλεία project management όπως GitHub Issues για να δημιουργηθούν τα απαραίτητα tasks. Αυτό διευκόλυνε την καλύτερη επικοινωνία μεταξύ των μελών της ομάδας και παρείχε μια ενιαία πηγή αλήθειας για την κατάσταση του έργου. Στην αρχή κάθε sprint πραγματοποιούνταν κλήσεις στο discord για να συζητηθούν οι διεργασίες που χρειάζονταν περισσότερη εστίαση και να λυθούν τυχών απορίες. Δεδομένης της συνεργατικής φύσης του έργου, η σαφής επικοινωνία ήταν κρίσιμη.

4.3 Ανάπτυξη Κώδικα (Code)

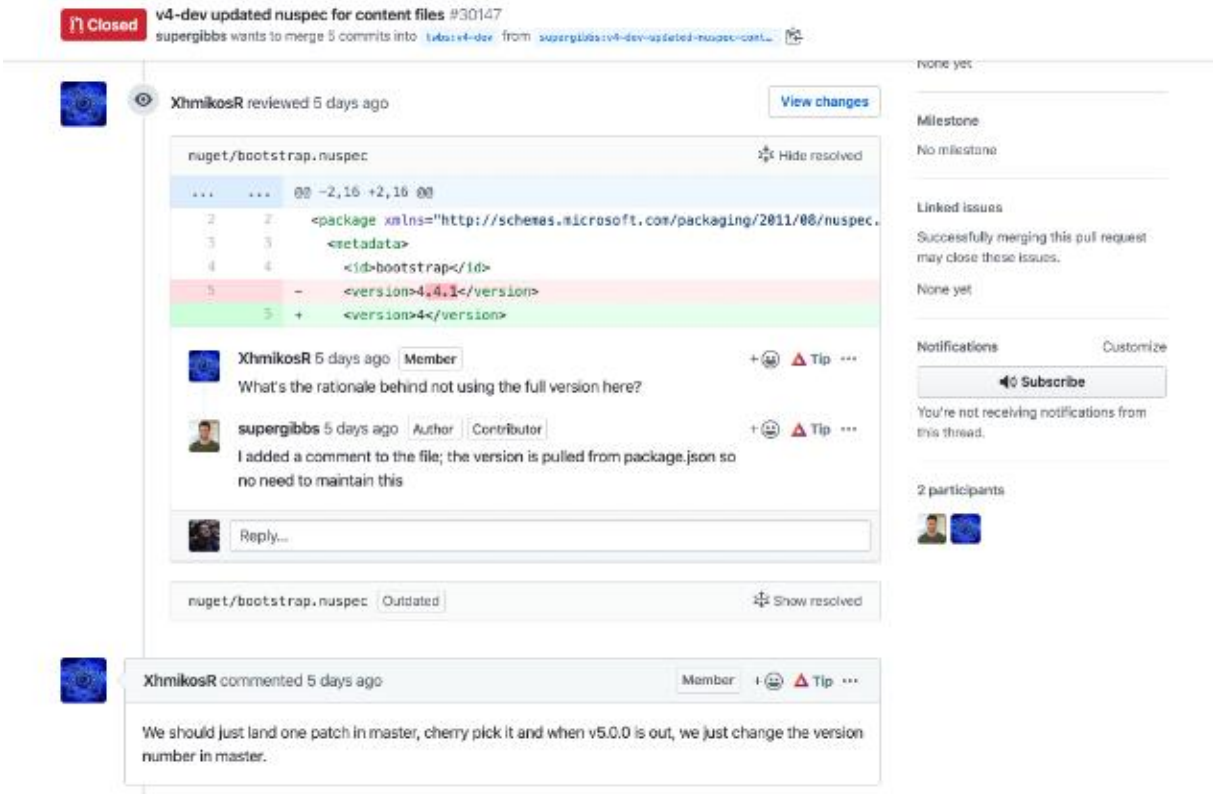
Στον κύκλο ζωής του DevOps, το στάδιο του κώδικα είναι το σημείο όπου ο σχεδιασμός υλοποιείται σε απτά στοιχεία λογισμικού. Δεν πρόκειται μόνο για τη σύνταξη γραμμών κώδικα, αλλά για τη διασφάλιση ότι ο κώδικας ευθυγραμμίζεται με τους αρχιτεκτονικούς στόχους του έργου, τα πρότυπα προγραμματισμού και τη μακροπρόθεσμη δυνατότητα συντήρησης.

- Επιλογές γλώσσας και Framework: Είναι συχνά μια σημαντική απόφαση σε αυτό το στάδιο. Διαφορετικές γλώσσες έχουν διαφορετικά δυνατά σημεία. Για παράδειγμα, η Node.js είναι γνωστή για την αρχιτεκτονική της που είναι non-blocking και event-driven, καθιστώντας την μια δημοφιλή επιλογή για scalable εφαρμογές web.



Σχήμα 5.3: Most common languages of 2022

- Collaborative ανάπτυξη και code reviews: Η συνεργασία είναι μια βασική πτυχή του DevOps. Οι προγραμματιστές συχνά εργάζονται σε μεμονωμένα git branches, κάνοντας commits σε ένα κοινό repository. Τα code reviews, που συχνά διευκολύνονται από πλατφόρμες όπως το GitHub ή το GitLab, είναι μια πρακτική για τη διασφάλιση της ποιότητας, της λειτουργικότητας του κώδικα και για την έγκαιρη αντιμετώπιση σφαλμάτων.

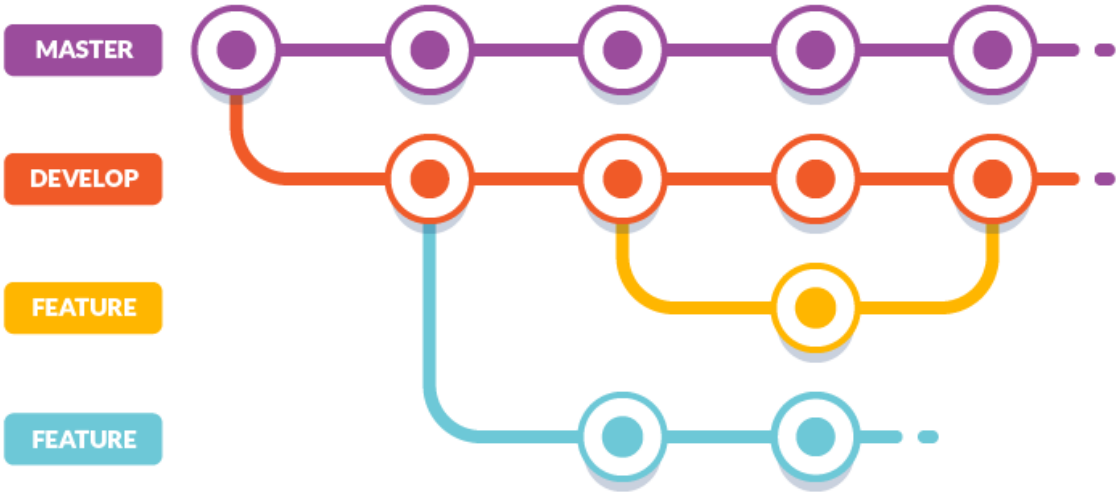


Σχήμα 5.3: Code review example

- Coding Standards: Έχουν οριστεί για τη διατήρηση της ομοιομορφίας και της αναγνωσιμότητας σε όλη τη βάση κώδικα. Αυτοματοποιημένα εργαλεία όπως το ESLint για JavaScript ή το RuboCop για Ruby μπορούν να επιβάλουν αυτούς τους κανόνες, διασφαλίζοντας ότι ο κώδικας ευθυγραμμίζεται με τα συμφωνημένα πρότυπα. Τρέχουν μέσα στα IDEs και υπογραμμίζουν τα σημεία του κώδικα που δεν τηρούν τους κανόνες. Για παράδειγμα όλα τα ονόματα των κλάσεων πρέπει να είναι σε Pascal Case.

Σχήμα 5.3: Linter highlight example

Code Repositories και Version Control: Τα συστήματα version control, όπως το Git, παρέχουν ένα ιστορικό αρχείο αλλαγών του κώδικα, βοηθούν στην επίλυση code conflict και διευκολύνουν τη συνεργασία μεταξύ των μελών της ομάδας. Μια σωστή στρατηγική branching, όπως feature branching ή Git flow μπορεί να βελτιστοποιήσει τη διαδικασία ανάπτυξης, καθιστώντας ευκολότερη τη διαχείριση λειτουργιών, επειγουσών επιδιορθώσεων και εκδόσεων.



Σχήμα 5.3: Git flow

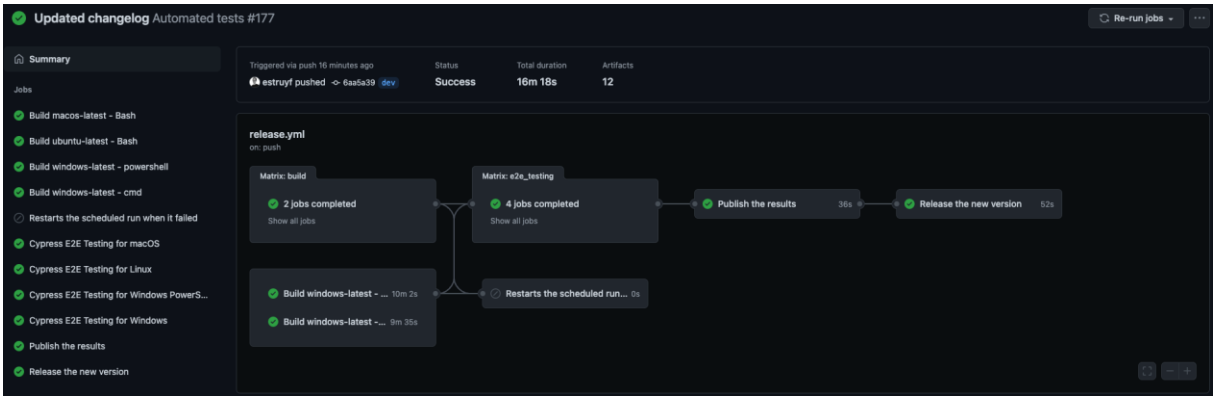
Συνεχής βελτίωση κώδικα: Το στάδιο του Code δεν είναι μια εφάπαξ διαδικασία, είναι επαναληπτικό. Καθώς το έργο προχωρά, η βάση κώδικα ενημερώνεται συνεχώς με feedback loops από automated testing, εργαλεία παρακολούθησης ή ανθρώπινες αναθεωρήσεις. Αυτή η επαναληπτική προσέγγιση επιτρέπει ευέλικτες απαντήσεις σε μεταβαλλόμενες απαιτήσεις, ευπάθειες ασφαλείας ή ζητήματα απόδοσης.

Συνολικά, το στάδιο του Code στο DevOps είναι μια πολύπλοκη και πολύπλευρη φάση που απαιτεί ένα συνδυασμό τεχνικών δεξιοτήτων, συνεργασίας και αυστηρής τήρησης βέλτιστων πρακτικών για την αποτελεσματική διαχείριση.

4.4 Build

Μετά το στάδιο Code, η φάση build είναι το σημείο όπου ο κώδικας μεταγλωττίζεται — συνήθως σε ένα εκτελέσιμο αρχείο. Αυτή είναι μια κρίσιμη συγκυρία στον κύκλο ζωής του DevOps, καθώς γεφυρώνει το χάσμα μεταξύ ανάπτυξης και deployment, διασφαλίζοντας ότι ο κώδικας είναι πλέον έτοιμος για να τρέξει σε ένα περιβάλλον (ή runtime).

- Δημιουργία εργαλείων και μεταγλώττιση: Σε αυτή τη φάση, η δημιουργία εργαλείων όπως το Maven για Java ή οι μεταγλωττιστές όπως το GCC για C/C++ και tsc για TypeScript παίζουν κεντρικό ρόλο. Παίρνουν τον πηγαίο κώδικα, μαζί με τα dependencies, και τον μετατρέπουν σε εκτελέσιμα αρχεία.
  Continuous Integration: Η φάση Build συχνά εφαρμόζει αρχές Continuous Integration (CI), ενεργοποιώντας αυτόματα builds από το σύστημα git κάθε φορά που γίνεται push νέος κώδικας. Οι υπηρεσίες CI όπως το Jenkins, το Travis CI ή το GitHub Actions χρησιμοποιούνται συνήθως για αυτόν τον σκοπό.



Σχήμα 5.4: GitHub Actions example

- Artifact Repository: Μετά από ένα επιτυχημένο build, τα artifact που δημιουργούνται - είτε πρόκειται για δυαδικά αρχεία, βιβλιοθήκες ή container - αποθηκεύονται σε ένα artifact repository όπως το GitHub, JFrog Artifactory ή ένα Docker Registry. Αυτά τα repositories λειτουργούν ως αποθηκευτικός χώρος με τις εκδόσεις του κώδικα, καθιστώντας εύκολη την επαναφορά ή το deployment συγκεκριμένων εκδόσεων του λογισμικού.
- Build Verification Tests (BVT): Πριν μετακινηθούν τα build artifacts στο στάδιο deployment, συνήθως υποβάλλονται σε ένα σύνολο γρήγορων, αυτοματοποιημένων tests γνωστών ως Build Verification Tests. Αυτά τα test έχουν σχεδιαστεί για να εντοπίζουν τυχόν κραυγαλέα ζητήματα που θα μπορούσαν να εμποδίσουν την εκτέλεση του λογισμικού όπως αναμένεται.
- Code Scanning και Security Checks: Η ασφάλεια είναι μια αυξανόμενη ανησυχία στον κύκλο ζωής ανάπτυξης λογισμικού και το στάδιο Build συχνά περιλαμβάνει static code analysis ή security scans για τον εντοπισμό ευπαθειών από νωρίς.

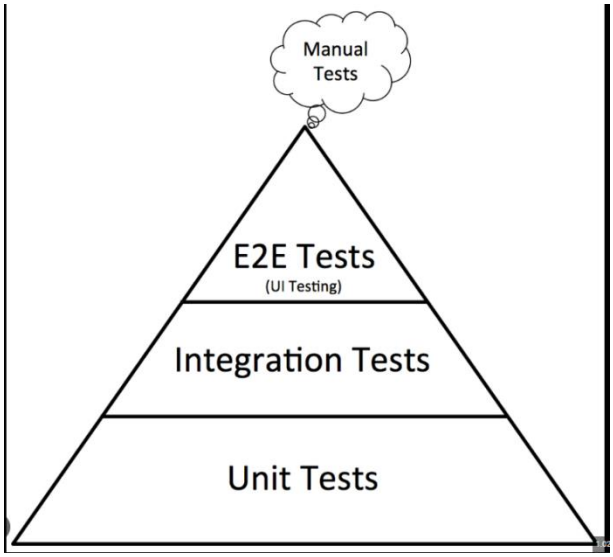
Συνοπτικά, το στάδιο Build είναι μια πολυ-επίπεδη διαδικασία που περιλαμβάνει διάφορα εργαλεία και πρακτικές που στοχεύουν στη μετατροπή του κώδικα σε deployable artifacts, διασφαλίζοντας ταυτόχρονα ποιότητα και ασφάλεια.

4.5 Test

Μετά τη φάση Build, το στάδιο Test μπαίνει στο παιχνίδι. Εδώ πραγματοποιείται αυστηρή αξιολόγηση του λογισμικού για να διασφαλιστεί ότι πληροί τα κριτήρια ποιότητας και λειτουργικότητας προτού αναπτυχθεί σε ένα ζωντανό περιβάλλον.

Σε αυτό το σημείο θα αναλύσουμε τους διάφορους τύπους test:

- **Unit Testing**: Επαληθεύει τη λειτουργικότητα μεμονωμένων στοιχείων ή μονάδων κώδικα. Για παράδειγμα, ένα function.
- **Integration Testing**: Ελέγχει την αλληλεπίδραση πολλαπλών στοιχείων ή συστημάτων. Για παράδειγμα, σε έναν web server εάν θα θέλαμε να τεστάρουμε το login service σαν σύνολο θα γράφαμε ένα integration test.
- **Performance Testing**: Μετρά την απόδοση του συστήματος σε αυξημένο φορτίο. Για παράδειγμα το εργαλείο artillery ή ChaosMonkey
- **Security Testing**: Αξιολογεί την ευπάθεια και τους παράγοντες κινδύνου.



Σχήμα 5.5: Test pyramid

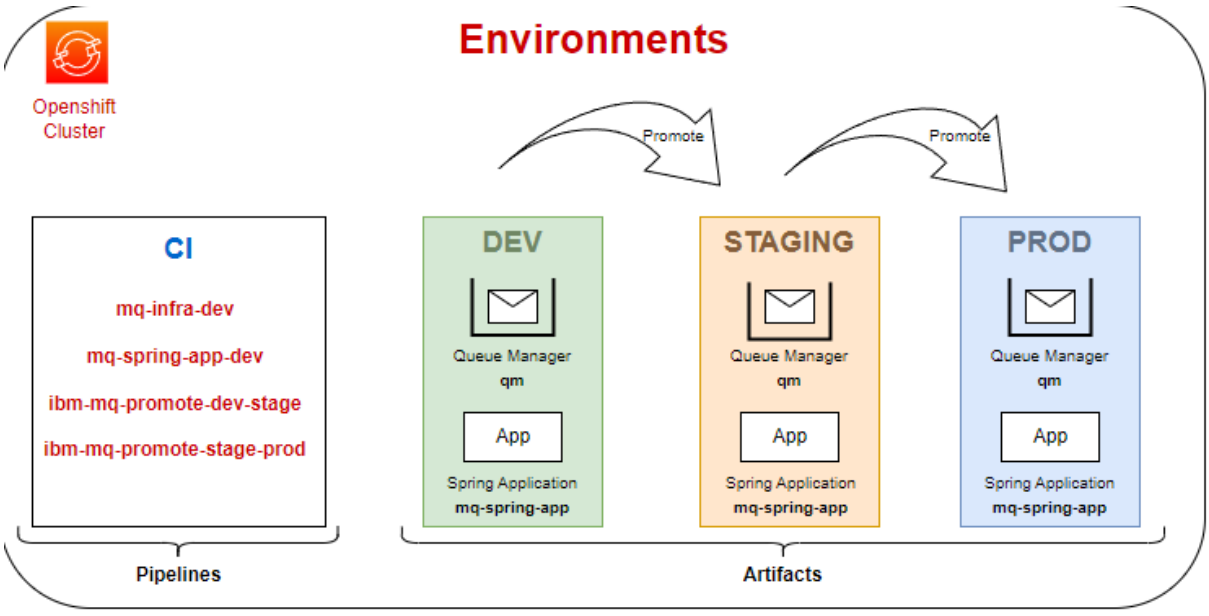
- **Test Automation**: Ο αυτοματισμός είναι ζωτικής σημασίας σε αυτό το στάδιο για γρήγορο feedback. Συχνά για το σκοπό αυτό χρησιμοποιούνται αυτοματοποιημένα testing frameworks όπως το Selenium, το JUnit ή το TestNG. Αυτά τα εργαλεία εκτελούν προκαθορισμένες test-cases στον build κώδικα και αναφέρουν την επιτυχία ή την αποτυχία τους.

```
describe('makeParagraph', () => {
  it('Should return an HTML paragraph object', () => {
    expect(makeParagraph('foo').tagName).toEqual('P');
  });

  it('Should render an empty paragraph, if not text provided', () => {
    expect(makeParagraph().textContent).toEqual('');
  });

  it('Should render the correct paragraph text', () => {
    const text = faker.lorem.sentences(3);
    expect(makeParagraph(text).textContent).toEqual(text);
  });
});
```

Σχήμα 5.5: A unit test example using Jes

- **Continuous Testing**: Όπως το Continuous Integration στο στάδιο Build, το Continuous Testing διασφαλίζει ότι οι αλλαγές κώδικα ελέγχονται αυτόματα καθώς ενσωματώνονται, ενισχύοντας το feedback loop για τους προγραμματιστές.

- **Απομόνωση Περιβάλλοντος**: Το test περιβάλλον είναι συχνά απομονωμένο από τα περιβάλλοντα development και production για την αποφυγή συγκρούσεων και την εξασφάλιση ακριβών αποτελεσμάτων δοκιμών. Εργαλεία όπως το Docker μπορεί να είναι πολύ χρήσιμα για τη διατήρηση τέτοιων απομονωμένων περιβαλλόντων δοκιμών. Συνήθως τα περιβάλλοντα είναι ακριβώς ίδια μεταξύ τους για να διευκολυνθεί η διαδικασία ανάπτυξης και δοκιμών.



Σχήμα 5.5: dev, stage, prod environments

- **Quality Assurance**: Οι ομάδες Διασφάλισης Ποιότητας (QA) ενδέχεται επίσης να εμπλακούν σε αυτό το στάδιο, οι οποίες εργάζονται παράλληλα με τα automation tests για να παρέχουν ανθρώπινη εικόνα για τη χρηστικότητα, τη λειτουργικότητα και τη συνολική ποιότητα.

- **Αναφορές και feedback**: Τα αποτελέσματα των tests, συχνά ορατά μέσω dashboards ή εργαλείων reporting, χρησιμοποιούνται για την παροχή άμεσου feedback στην ομάδα development. Τυχόν αποτυχημένα tests θα πρέπει να αντιμετωπιστούν πριν προχωρήσει ο κώδικας στο επόμενο στάδιο του κύκλου ζωής του DevOps.

Ουσιαστικά, το στάδιο Test είναι ζωτικής σημασίας για τη διασφάλιση ότι το λογισμικό είναι ταυτόχρονα λειτουργικό και αξιόπιστο, θέτοντας το στάδιο για την επόμενη φάση ανάπτυξης.

4.6: Deployment

Το στάδιο Deploy στον κύκλο ζωής του DevOps διαδραματίζει ουσιαστικό ρόλο, καθώς σηματοδοτεί τη μετάβαση της εφαρμογής από ένα περιβάλλον development σε ένα production. Αυτό το στάδιο είναι όπου όλα τα προηγούμενα στάδια plan, code, build, test — κορυφώνονται.

Ο κύριος στόχος του σταδίου Deploy είναι η κυκλοφορία των νέων features, επιδιορθώσεων ή ενημερώσεων στο περιβάλλον production με ελεγχόμενο και συστηματικό τρόπο. Αυτό διασφαλίζει ότι τυχόν αλλαγές που εισάγονται δεν διαταράσσουν το υπάρχον σύστημα και, εάν συμβεί, μπορούν να επαναφερθούν αποτελεσματικά.

Σε αυτή τη φάση, ο εκτελέσιμος κώδικας, μαζί με τα απαραίτητα αρχεία και βιβλιοθήκες, τοποθετούνται σε production servers. Αυτή η διαδικασία μπορεί να ποικίλλει ευρέως ανάλογα με την αρχιτεκτονική της εφαρμογής, τον τύπο του διακομιστή που χρησιμοποιείται και τις συγκεκριμένες οργανωτικές ανάγκες. Είναι αυτοματοποιημένο για τη μείωση του ανθρώπινου λάθους, τη βελτίωση της αποτελεσματικότητας και την επιτάχυνση της διαδικασίας ανάπτυξης.

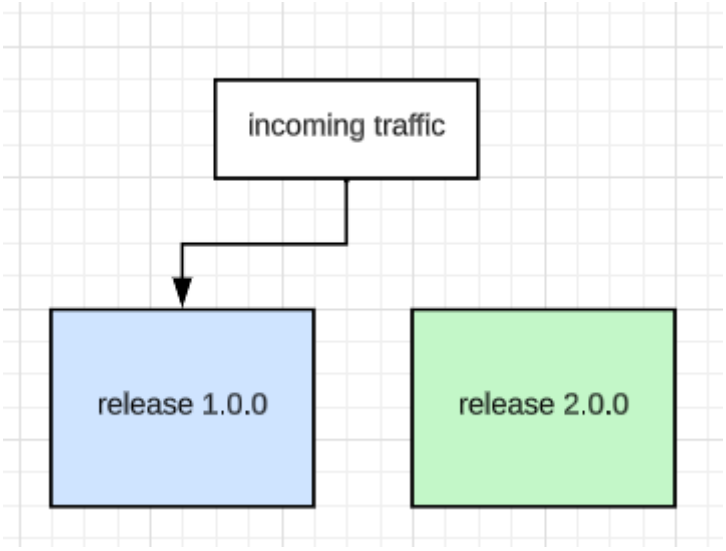
Χρησιμοποιούνται συνήθως εργαλεία ανάπτυξης όπως το Kubernetes για container orchestration, το Ansible για τη διαχείριση configuration και το Terraform για το Infrastructure as Code (IaC). Αυτά τα εργαλεία μπορούν να αυτοματοποιήσουν τη διαδικασία του deployment, επιτρέποντας ταχύτερες και πιο αξιόπιστες εκδόσεις.

4.6.1 Blue/Green Deployment

Το Blue-Green Deployment είναι μια στρατηγική release management που έχει σχεδιαστεί για να ελαχιστοποιεί το χρόνο διακοπής λειτουργίας και να μειώνει τον κίνδυνο που σχετίζεται κατά το deployment νέων εκδόσεων του κώδικα. Προσφέρει έναν τρόπο μετάβασης από μια προηγούμενη έκδοση μιας εφαρμογής σε μια νέα έκδοση με διακοπή λειτουργίας.

Σε ένα Blue-Green Deployment, υπάρχουν δύο πανομοιότυπα περιβάλλοντα production, τα οποία συνήθως αναφέρονται ως περιβάλλοντα "Blue" και "Green". Ανά πάσα στιγμή, ένα από αυτά τα περιβάλλοντα είναι ζωντανό, εξυπηρετώντας όλη την κυκλοφορία παραγωγής, ενώ το άλλο είναι σε αδράνεια.

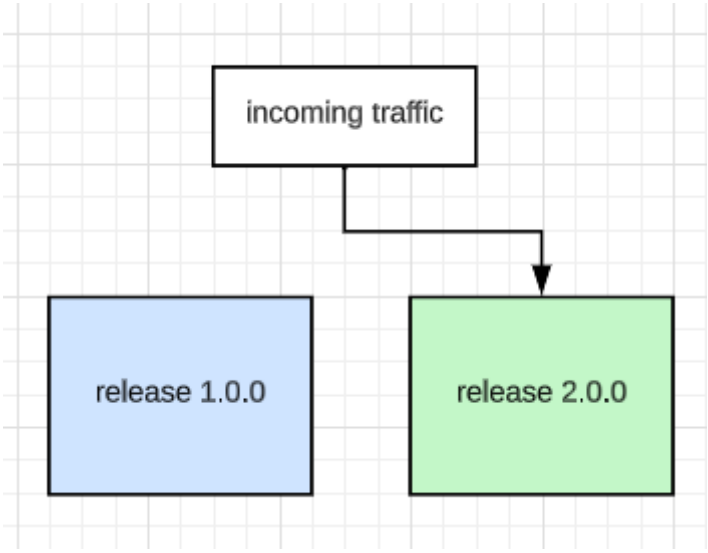
1) **Αρχική κατάσταση**: Αρχικά, το Blue περιβάλλον είναι ζωντανό και το Green περιβάλλον είναι ένας καθρέφτης του Blue.
2) **Deployment**: Η νέα έκδοση της εφαρμογής γίνεται deploy στο Green περιβάλλον. Αυτό το deployment γίνεται χωρίς να επηρεάζεται το Blue περιβάλλον, το οποίο συνεχίζει να εξυπηρετεί τους χρήστες.



Σχήμα 5.6.1: Blue/green deploy stage 1

3) **Testing**: Εκτελούνται automated tests στο περιβάλλον Green για να διασφαλιστεί ότι η νέα ανάπτυξη είναι σταθερή.

4) **Switch**: Μόλις επαληθευτεί ότι το περιβάλλον Green είναι σταθερό, η κίνηση production αλλάζει από Blue σε Green. Αυτό γίνεται συχνά με την ενημέρωση ενός load balancer για να ανακατευθύνει την κυκλοφορία.



Σχήμα 5.6.1: Blue/green deploy stage 2 1

5) **Rollback**: Εάν κάτι πάει στραβά, μπορούμε να επιστρέψουμε γρήγορα στην προηγούμενη έκδοση, επιστρέφοντας την κυκλοφορία στο περιβάλλον Blue.

Οφέλη

- **Μηδενικός χρόνος διακοπής λειτουργίας**: Επειδή ένα περιβάλλον είναι πάντα ζωντανό, το Blue-Green Deployment επιτρέπει deployments με μηδενικό χρόνο διακοπής λειτουργίας.
- **Άμεση επαναφορά**: Το περιβάλλον αδράνειας (blue) χρησιμεύει ως επιλογή άμεσης επαναφοράς σε περίπτωση αστοχιών, ελαχιστοποιώντας τους κινδύνους.
- **Isolation**: Επιτρέπει την απομόνωση της νέας έκδοσης για δοκιμή πριν γίνει ζωντανή.

Προκλήσεις

- **Πόροι**: Απαιτεί δύο περιβάλλοντα production, τα οποία θα μπορούσαν καταναλώνουν αρκετούς πόρους τόσο από άποψη υποδομής όσο και από άποψη κόστους.
- **Συγχρονισμός δεδομένων**: Οποιεσδήποτε αλλαγές δεδομένων στο ζωντανό (Blue) περιβάλλον πρέπει να συγχρονιστούν με το περιβάλλον αδράνειας (Green), το οποίο μπορεί να είναι περίπλοκο ανάλογα με την εφαρμογή. Συνήθως χρησιμοποιείτε μια Redis για τέτοια προβλήματα.

Σημασία στο DevOps

Στο DevOps, ο αυτοματισμός και τα feedback loops είναι πολύ σημαντικά. Τα Blue-Green Deployment ταιριάζουν καλά σε αυτό το παράδειγμα, επιτρέποντας γρήγορες, αξιόπιστες εκδόσεις και αποτελεσματιά rollbacks.

Συνοπτικά, το Blue-Green Deployment είναι μια ισχυρή στρατηγική για τη διαχείριση εκδόσεων, τη μείωση των κινδύνων και την επίτευξη μηδενικού χρόνου διακοπής λειτουργίας, καθιστώντας το εξαιρετικά πολύτιμο σε ένα πλαίσιο DevOps.

4.6.2 Προκλήσεις και Προβληματισμοί

.

- **Rollback Strategy**: Το να υπάρχει ένα σχέδιο για την επαναφορά των αλλαγών είναι ζωτικής σημασίας εάν κάτι πάει στραβά κατά το deployment.
- **Downtime**: Ανάλογα με τη στρατηγική deployment, η εφαρμογή ενδέχεται να αντιμετωπίσει διακοπές λειτουργίας κατά το deployment. Το blue/green deployment είναι μια στρατηγική που χρησιμοποιείται για αυτόν τον λόγο.
- **Ασφάλεια**: Τα deployments πρέπει να ταιριάζουν με τα πρωτόκολλα ασφαλείας του οργανισμού, διασφαλίζοντας ότι προστατεύονται τα ευαίσθητα δεδομένα.

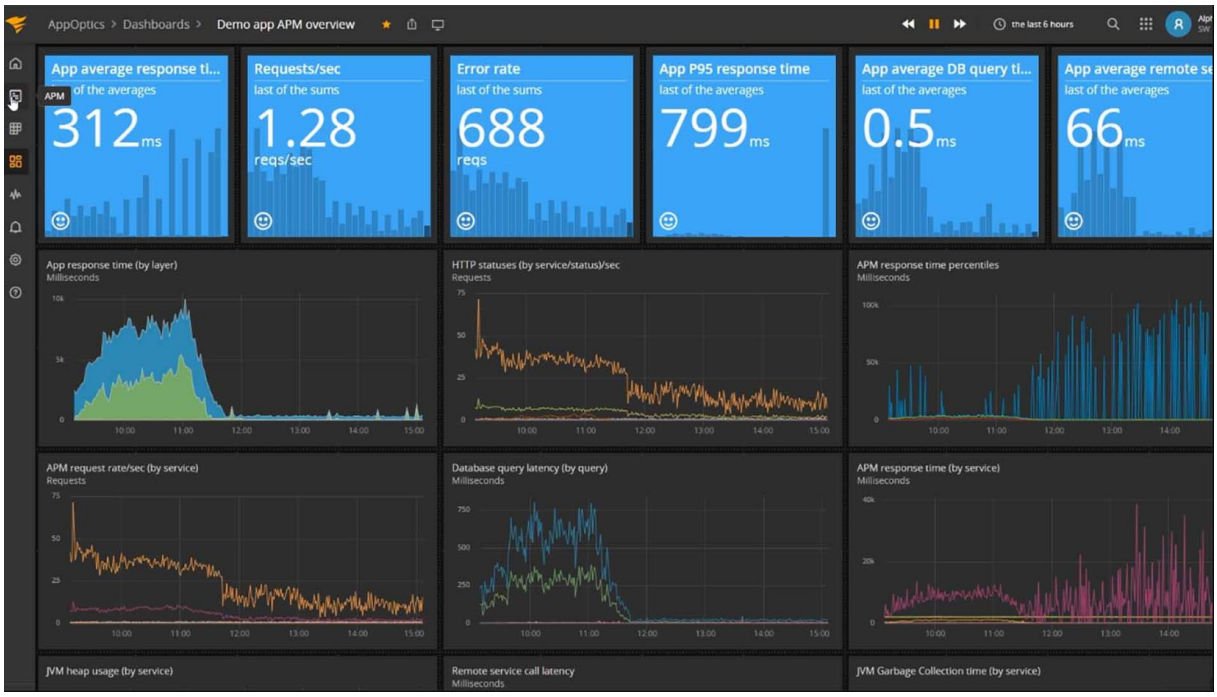
Σημασία στο DevOps

Στο παράδειγμα DevOps, το στάδιο Deploy είναι εξαιρετικά αυτοματοποιημένο, επιτρέποντας το συνεχές deployment όπου οποιαδήποτε αλλαγή κώδικα που περνάει τις αυτοματοποιημένες δοκιμές μπορεί να αναπτυχθεί αυτόματα στην παραγωγή. Αυτό οδηγεί σε ταχύτερους κύκλους ανάπτυξης και ταχύτερη απελευθέρωση νέων λειτουργιών ή διορθώσεων σφαλμάτων, κάτι που είναι επωφελές τόσο για την ομάδα ανάπτυξης όσο και για τους τελικούς χρήστες.

Συμπερασματικά, το στάδιο Deploy στον κύκλο ζωής του DevOps είναι ένα κομβικό σημείο όπου η εφαρμογή γίνεται προσβάσιμη στους τελικούς χρήστες. Η επιτυχής εκτέλεσή του εγγυάται ότι η επαναληπτική εργασία που έγινε σε προηγούμενα στάδια κορυφώνεται σε ένα σταθερό, χρησιμοποιήσιμο προϊόν.

4.7 Monitor

Το monitoring είναι το τελευταίο, στάδιο του κύκλου ζωής του DevOps που εστιάζει στην προσεκτική παρατήρηση της απόδοσης και της συμπεριφοράς των εφαρμογών και της υποδομής σε πραγματικό χρόνο. Είναι χρήσιμο για τη διασφάλιση ότι οι αναπτυγμένες εφαρμογές λειτουργούν όπως προβλέπεται και είναι διαθέσιμες, αξιόπιστες και ασφαλείς.



Σχήμα 5.7: Monitoring dashboard

Σημασία στο DevOps

Σε ένα πλαίσιο DevOps, η παρακολούθηση δεν είναι μια μεμονωμένη εργασία, αλλά μια συνεχής διαδικασία. Παρέχει feedback loops που απαιτούνται για τη συνεχή βελτίωση τόσο της εφαρμογής όσο και του pipeline. Επιτρέπει στις ομάδες να είναι προληπτικές και όχι αντιδραστικές, αντιμετωπίζοντας ζητήματα προτού επηρεάσουν την εμπειρία του χρήστη ή γίνουν μεγαλύτερα προβλήματα.

Βασικά συστατικά

- **Application Performance Monitoring (APM)**: Παρακολουθεί την εμπειρία του χρήστη και την απόδοση των διαφόρων στοιχείων μιας εφαρμογής. Μπορούν να ρυθμιστούν ειδοποιήσεις για να ειδοποιούν οι ομάδες για ανωμαλίες.
- **Infrastructure Monitoring**: Παρατηρεί την απόδοση των server, των βάσεων δεδομένων, των δικτύων και άλλων στοιχείων υποδομής. Αυτό είναι απαραίτητο για το dynamic scaling των πόρων με βάση το φορτίο.
- **Log Monitoring**: Συλλέγει και αναλύει αρχεία καταγραφής για ενδείξεις ζητημάτων ή συμβάντων ασφαλείας. Τα αρχεία καταγραφής μπορεί να προέρχονται από εφαρμογές, servers, βάσεις δεδομένων ή οποιοδήποτε άλλο μέρος του συστήματος.
- **Security Monitoring**: Επικεντρώνεται στον εντοπισμό και την ειδοποίηση για απειλές ασφαλείας σε πραγματικό χρόνο, βοηθώντας έτσι στην ταχεία απόκριση σε τυχόν περιστατικά.

Οφέλη

- **Βελτιωμένη ορατότητα**: Παρέχει μια σαφή εικόνα της απόδοσης των εφαρμογών και των πόρων.
- **Early Issue Detection**: Εντοπίζει προβλήματα προτού επηρεάσουν τους χρήστες, βελτιώνοντας έτσι την εμπειρία χρήστη και την αξιοπιστία του συστήματος.
- **Performance Tuning**: Επιτρέπει στις ομάδες να εντοπίσουν τα σημεία bottleneck και να βελτιστοποιήσουν την απόδοση της εφαρμογής και του συστήματος.
- **Compliance and Auditing**: Βοηθά στην συλλογή πληροφοριών και αρχείων που θα μπορούσαν να είναι απαραίτητα για σκοπούς ελέγχου της απόδοσης εφαρμογής

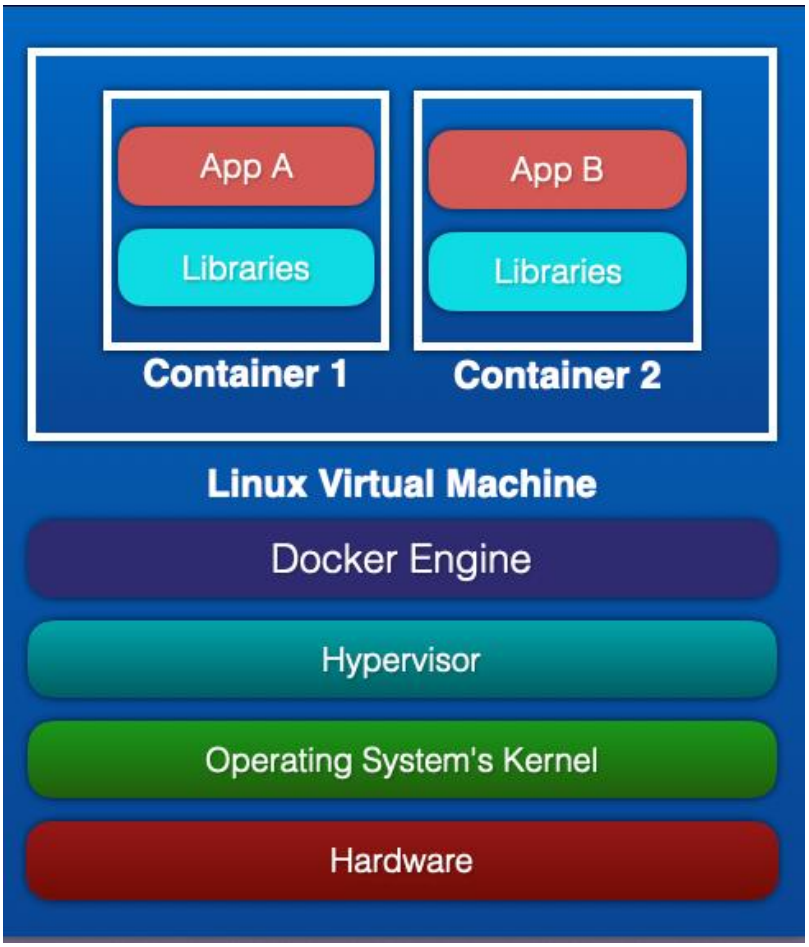
Προκλήσεις

- **Πολυπλοκότητα**: Όσο πιο περίπλοκο είναι το σύστημα, τόσο πιο δύσκολη μπορεί να είναι η αποτελεσματική παρακολούθηση του
- **Κόστος**: Τα προηγμένα εργαλεία παρακολούθησης μπορεί να είναι ακριβά. Επειδή αρχειοθετούν και συλλέγουν μεγάλο όγκο δεδομένων και προσφέρουν στατιστικά σε πραγματικό χρόνο.
- **Άσκοπες ειδοποιήσεις**: Η λανθασμένη διαμόρφωση μπορεί να οδηγήσει σε συχνές περιττές ειδοποιήσεις, οι οποίες μπορεί να αγνοηθούν, με αποτέλεσμα να αναιρείται ο σκοπός της παρακολούθησης.

Παρέχοντας ένα πλαίσιο για συνεχή παρατήρηση και ειδοποίηση, η παρακολούθηση διασφαλίζει ότι τα συστήματα είναι ασφαλή, αξιόπιστα και έχουν τη βέλτιστη απόδοση, κλείνοντας έτσι τον βρόχο DevOps.

4.8 Containerization

Η αποθήκευση container είναι μια όλο και πιο σημαντική πτυχή της σύγχρονης ανάπτυξης λογισμικού. Αν και η ιδέα δεν είναι νέα, ο τρόπος που χρησιμοποιείται και ενσωματώνεται στις διαδικασίες DevOps είναι πρωτοποριακός. Αυτό το κεφάλαιο στοχεύει να παρέχει μια σε βάθος θεωρητική κατανόηση του containerization, εξετάζοντας τα βασικά στοιχεία του, τη συνάφεια με τα DevOps και τις πρακτικές εφαρμογές του.



Σχήμα 5.8: Contrainerization layers

4.8.1 Τι είναι το Containerization

Το Containerization είναι μια ελαφριά μορφή virtualization που μας επιτρέπει να πακετάρουμε μια εφαρμογή και τα dependencies της, καθιστώντας εύκολη τη συνεπή μετακίνηση της εφαρμογής σε διάφορα υπολογιστικά περιβάλλοντα. Αυτή η ιδέα φέρνει επανάσταση στον τρόπο κατασκευής, διανομής και εκτέλεσης εφαρμογών λογισμικού.

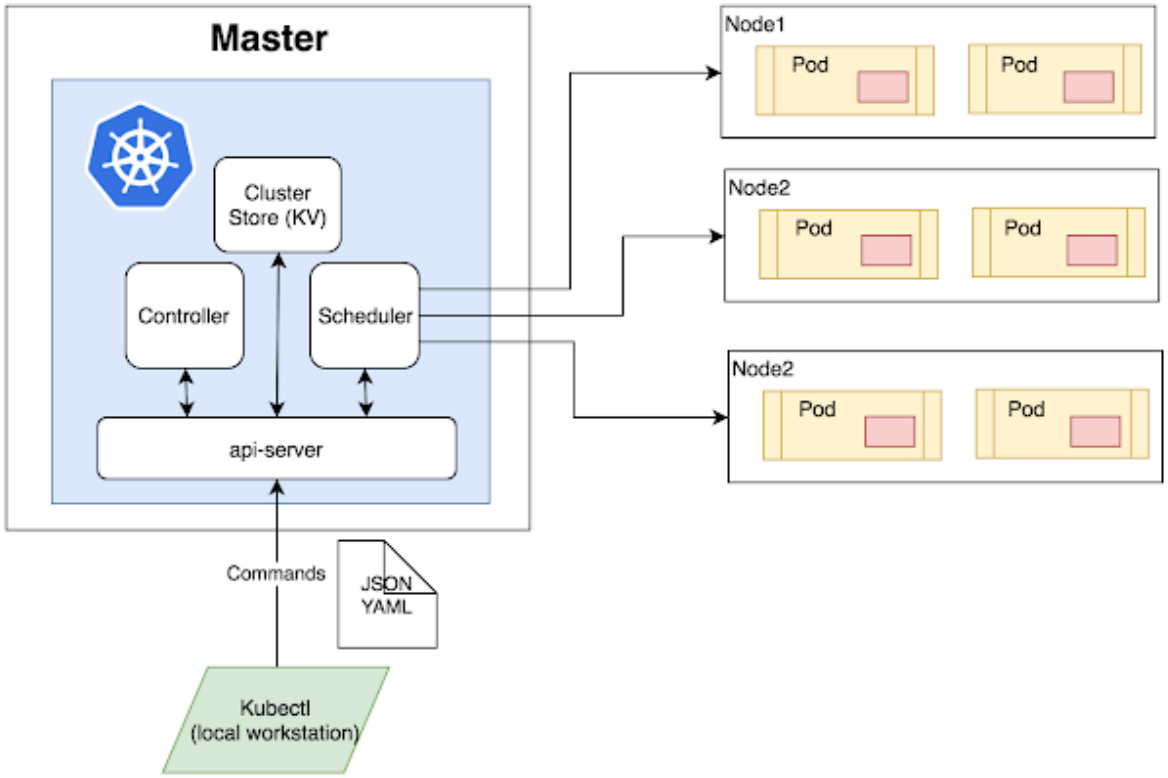
4.8.2 Γιατί το Containerization είναι σημαντικό στο DevOps;

- **Immutability**: Τα κοντέινερ διασφαλίζουν ότι μια εφαρμογή εκτελείται με τον ίδιο τρόπο, ανεξάρτητα από το πού έχει αναπτυχθεί.

- **Scalability**: Τα containers αναπροσαρμόζονται εύκολα προς τα πάνω ή προς τα κάτω, ανάλογα με τις απαιτήσεις πόρων, βελτιώνουν σημαντικά τα ποσοστά χρήσης.
- **Isolation**: Κάθε εφαρμογή και το περιβάλλον της τρέχουν απομονομένα, μειώνοντας τον κίνδυνο σφαλμάτων σε όλο το σύστημα λόγω μιας ελαττωματικής εφαρμογής.

4.8.3 Συστατικά Container

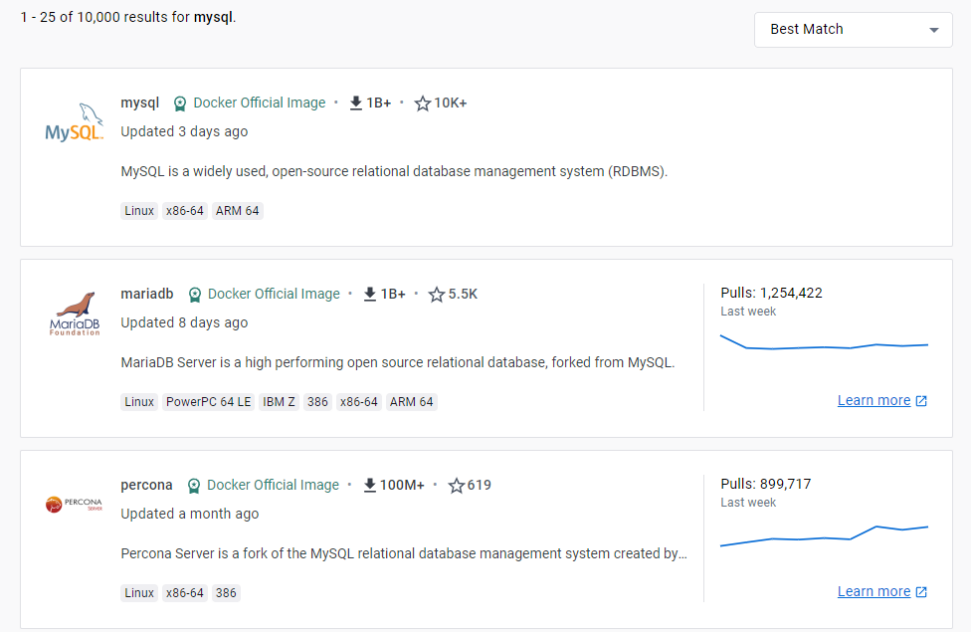
- **Container Engine**: Λογισμικό όπως το Docker παρέχει το περιβάλλον runtime για κοντέινερ.
- **Container Registry**: Μια υπηρεσία που έχει ως σκοπό να αποθηκεύει και να διανέμει container images
- **Orchestration**: Εργαλεία όπως το Kubernetes διαχειρίζονται τον τρόπο αλληλεπίδρασης, deployment, monitoring και scaling πολλαπλών κοντέινερ.



Σχήμα 5.8.3: A simple kubernetes cluster

4.8.4 Τι είναι το Docker

Το Docker είναι μια πλατφόρμα ανοιχτού κώδικα που αυτοματοποιεί το deployments, το scaling και τη διαχείριση εφαρμογών μέσα σε ελαφριά, φορητά κοντέινερ. Αυτά τα κοντέινερ ενσωματώνουν μια εφαρμογή και τα dependencies της, καθιστώντας δυνατή την ομοιόμορφη και συνεπή εκτέλεση σε διάφορα υπολογιστικά περιβάλλοντα. Ξεκινώντας ως ένα προτζεκτ για την αυτοματοποίηση των deployments των εφαρμογών μέσα σε ελαφριά κοντέινερ, το Docker έχει εξελιχθεί σε ένα οικοσύστημα που με συμπληρωματικές υπηρεσίες όπως το Docker Swarm για orchestration, το Docker Compose για multi-container εφαρμογές και το Docker Hub για κοινή χρήση image container.



Σχήμα 5.8.4:Dockerhub results for mysql

Οφέλη

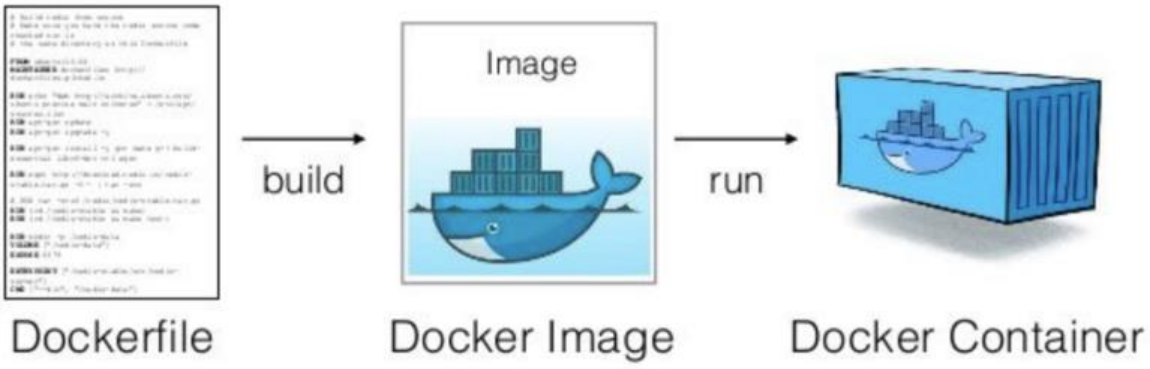
- **Φορητότητα**: Ανάπτυξη της εφαρμογής μία φορά για να τρέχει σε οποιοδήποτε περιβάλλον.
- **Αποδοτικότητα πόρων**: Τα container είναι πιο αποτελεσματικά από τις παραδοσιακές εικονικές μηχανές, επειδή μοιράζονται τον kernel του host OS, αντί να χρειάζονται το δικό τους λειτουργικό σύστημα.

Προκλήσεις

- **Ασφάλεια**: Τα κοντέινερ μοιράζονται τον πυρήνα του συστήματος κεντρικού υπολογιστή, το οποίο μπορεί να είναι μια ευπάθεια.
- **Πολυπλοκότητα**: Η διαχείριση πολλών κοντέινερ και η ενημέρωση τους μπορεί να είναι περίπλοκη.
- **Αποθήκευση**: Ο χειρισμός της αποθήκευσης στα containers παραμένει μια πρόκληση.

4.8.5 Τι είναι το Docker Image

Στο οικοσύστημα Docker, ένα image χρησιμεύει ως ένα ελαφρύ, αυτόνομο, εκτελέσιμο πακέτο που περιέχει το λογισμικό, τις βιβλιοθήκες και το runtime που απαιτούνται για την εκτέλεση μιας εφαρμογής. Με απλά λόγια, ένα Docker image είναι ένα στιγμιότυπο μιας εφαρμογής σε container, που περιλαμβάνει τα πάντα, από κώδικα και εξαρτήσεις έως environmental variables και βιβλιοθήκες συστήματος. Λειτουργεί ως blueprint για τη δημιουργία Docker Containers.



Σχήμα 5.8.5: Docker image steps

.

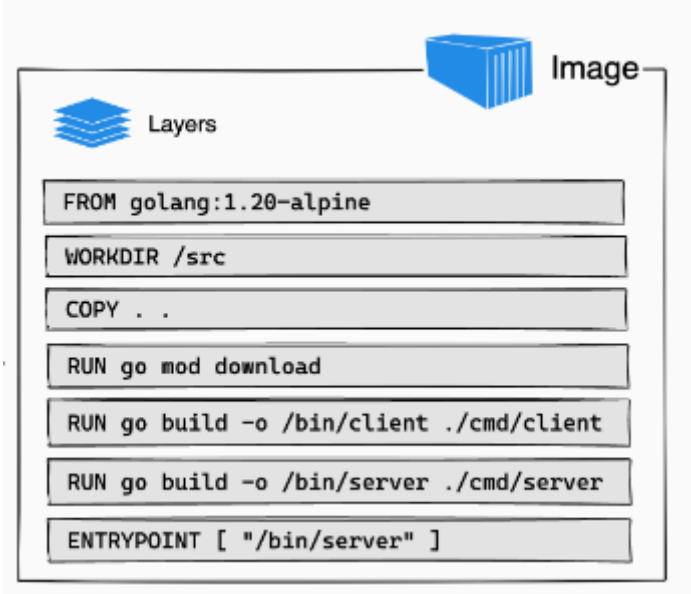
4.8.9 Ανατομία ενός Docker Image

Ένα Docker image αποτελείται από μια σειρά επιπέδων, καθένα από τα οποία αντιπροσωπεύει ένα σύνολο εντολών που καθορίζονται στο αρχείο Dockerfile. Όταν δημιουργείται ένα image, το Docker αποθηκεύει στην κρυφή μνήμη αυτά τα επίπεδα, έτσι οι επόμενες εκδόσεις μπορούν να τα επαναχρησιμοποιήσουν, επιταχύνοντας τη διαδικασία. Κάθε επίπεδο είναι μόνο readonly και νέες αλλαγές προστίθενται σε νέα επίπεδα πάνω από τα υπάρχοντα.

Base Layer: Συνήθως ένα λειτουργικό σύστημα ή κάποιο configuration στο οποίο θα εκτελείται η εφαρμογή

Dependency Layer: Στεγάζει βιβλιοθήκες, frameworks και άλλα dependencies που χρειάζεται η εφαρμογή.

Application Layer: Περιέχει τον κώδικά της εφαρμογής και τυχόν πρόσθετα στοιχεία που απαιτούνται για την εκτέλεσή της.



Σχήμα 5.8.9: Docker image layers

Η αμετάβλητη φύση των Docker images

Μία από τις βασικές αρχές πίσω από τις εικόνες Docker είναι το immutability. Μόλις ένα image, δεν αλλάζει. Τυχόν αλλαγές έχουν ως αποτέλεσμα τη δημιουργία ενός νέου image. Αυτή η αμετάβλητη φύση διασφαλίζει ότι το ίδιο το image μπορεί να χρησιμοποιηθεί σε διαφορετικά στάδια του κύκλου ανάπτυξης, διασφαλίζοντας συνέπεια και αξιοπιστία.

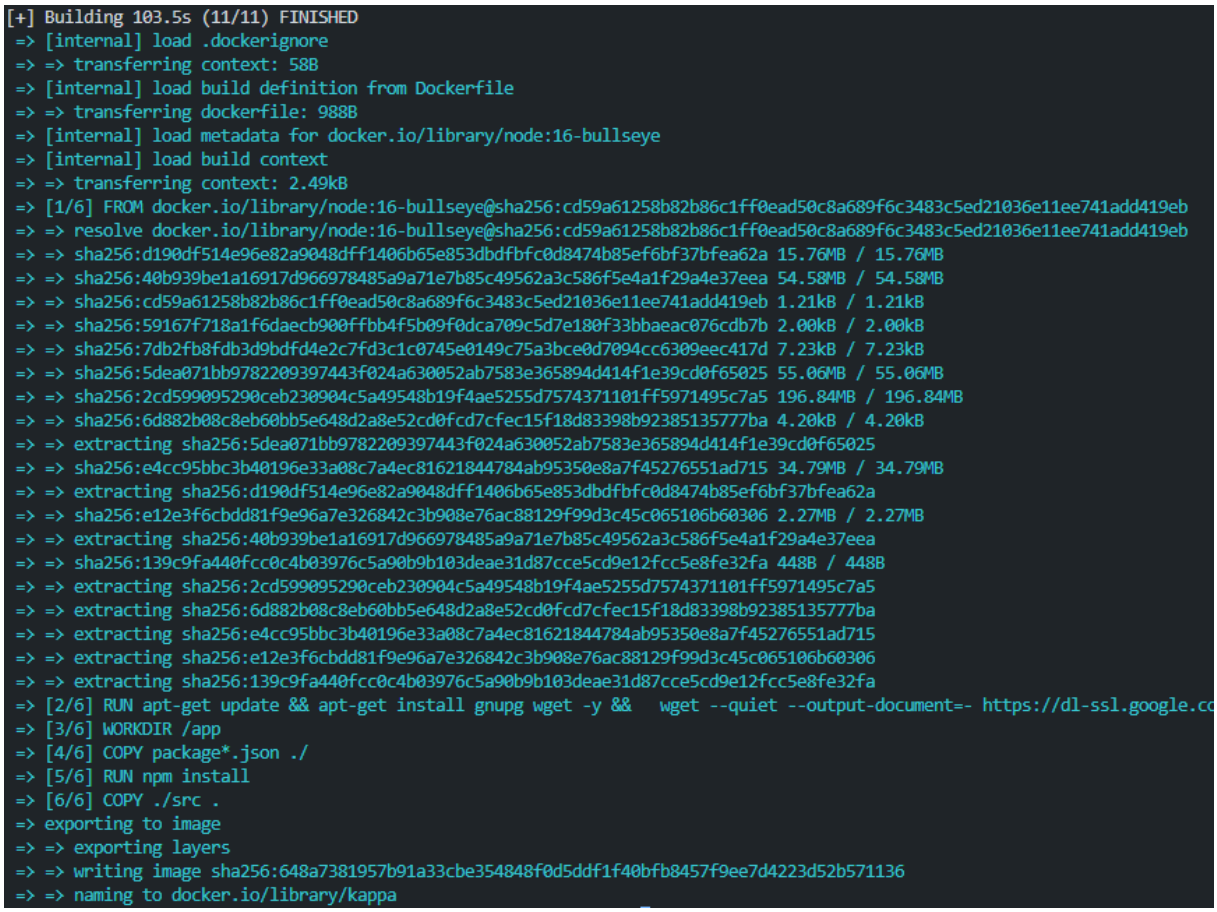
Δημιουργία Docker Image

Η δημιουργία ενός Docker image συνήθως περιλαμβάνει τον ορισμό ενός Dockerfile, ενός εγγράφου κειμένου που περιέχει εντολές και οδηγίες για τη δημιουργία του image. Το Dockerfile καθορίζει το base image, ορίζει μεταβλητές περιβάλλοντος, εγκαθιστά dependencies και αντιγράφει αρχεία εφαρμογών. Με την εκτέλεση της εντολής docker build εκτελούνται αυτές οι οδηγίες και δημιουργείται ένα νέο Docker image.

PriceMonitoringScrapper > scrapper > Dockerfile > ...

```
# Ensure an up-to-date version of Chromium
# can be installed (solves Problem 2)
FROM node:16-bullseye

ENV SERVICE_TYPE=${SERVICE_TYPE}
ENV RABBIT_MQ_HOST=${RABBIT_MQ_HOST}
ENV LOKI_HOST=${LOKI_HOST}
ENV CONCURRENCY=${CONCURRENCY}
ENV BCRYPT_SALT=${BCRYPT_SALT}
ENV PORT=${PORT}

ENV PUPPETEER_SKIP_CHROMIUM_DOWNLOAD true

RUN apt-get update && apt-get install gnupg wget -y && \
  wget --quiet --output-document=- https://dl-ssl.google.com/linux/linux_sign
  sh -c 'echo "deb [arch=amd64] http://dl.google.com/linux/chrome/deb/ stable
  apt-get update && \
  apt-get install google-chrome-stable -y --no-install-recommends && \
  rm -rf /var/lib/apt/lists/*

WORKDIR /app
COPY package*.json ./
ENV PUPPETEER_SKIP_CHROMIUM_DOWNLOAD=true
RUN npm install

COPY ./src .
CMD [ "node", "index.js" ]
```

Σχήμα 5.8.9: Docker scraper definition



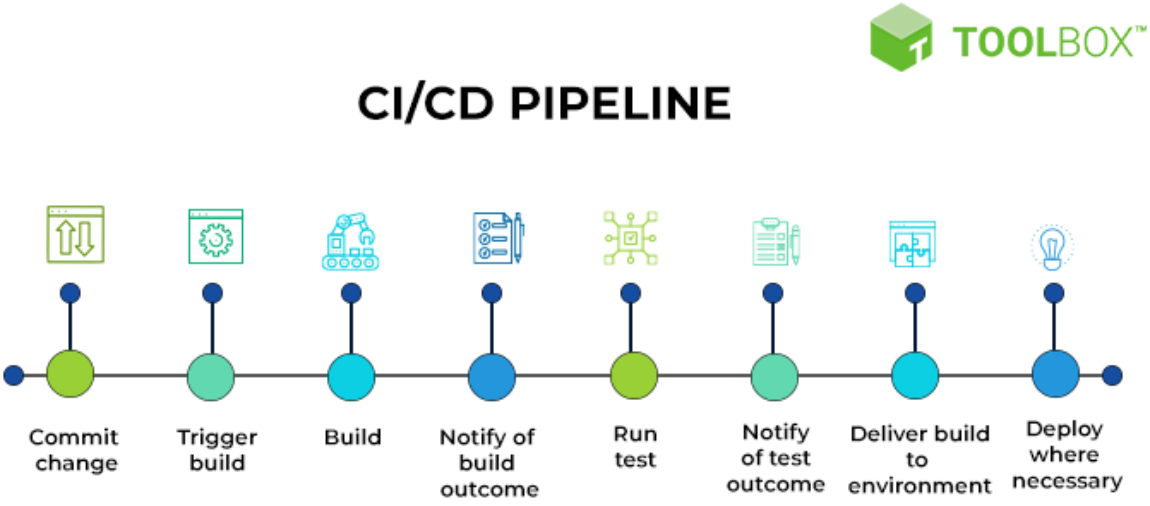
Σχήμα 5.8.9: Docker image build

Docker Images στο σύστημα μας

Στο πλαίσιο του έργου μας, χρησιμοποιήσαμε διάφορα Docker images για διαφορετικούς σκοπούς. Για παράδειγμα, χρησιμοποιήσαμε ενα official image του Node.js ως βάση για τα scraper containers. Τα images RabbitMQ και Grafana χρησιμοποιήθηκαν για την ουρά μηνυμάτων και administrator dashboards, αντίστοιχα.

4.9 Continuous Integration και Continuous Deployment (CI/CD)

Η Continuous Integration (CI) και η Continuous Deployment (CD) ενσωματώνουν μια κουλτούρα, ένα σύνολο αρχών λειτουργίας και μια συλλογή πρακτικών που επιτρέπουν στις ομάδες ανάπτυξης εφαρμογών να παρέχουν αλλαγές κώδικα πιο συχνά και αξιόπιστα. Αυτή η προσέγγιση είναι θεμελιώδης για τη σύγχρονη ανάπτυξη λογισμικού, επιτρέποντας την ταχεία κατασκευή, δοκιμή και ανάπτυξη λογισμικού αυτοματοποιώντας διάφορα στάδια της διαδικασίας ανάπτυξης.

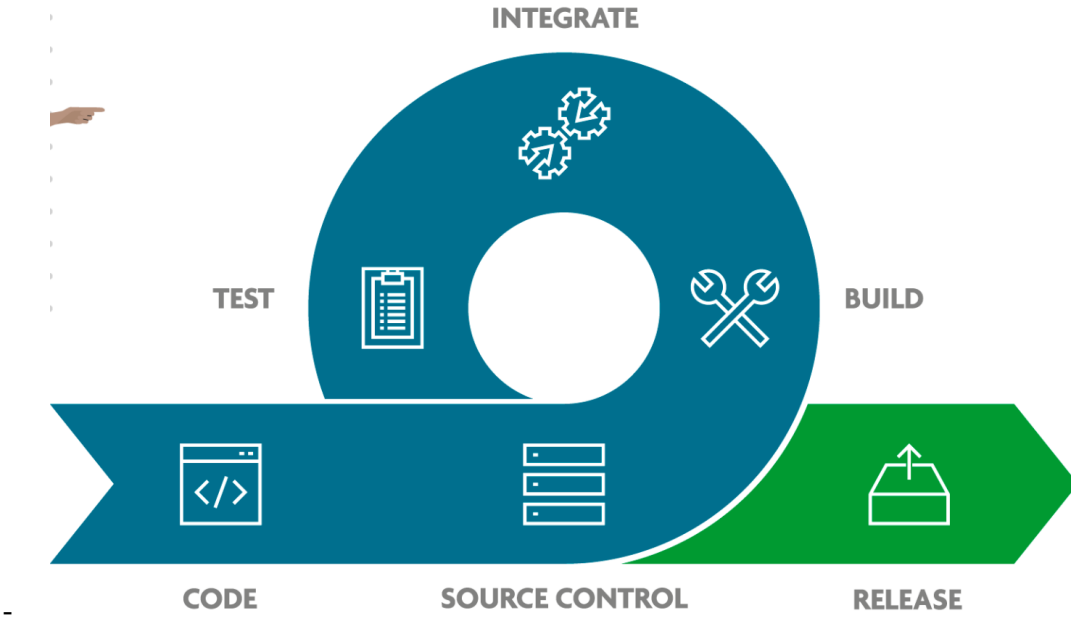


Σχήμα 5.9: CICD pipeline example

4.9.1 Βασικές αρχές Continuous Integration

Το Continuous Integration είναι η πρακτική της αυτόματης ενσωμάτωσης αλλαγών κώδικα από πολλούς προγραμματιστές σε ένα κοινόχρηστο git repository πολλές φορές την ημέρα. Οι προγραμματιστές υποβάλλουν ένα "push request" το οποίο ενεργοποιεί μια σειρά από αυτοματοποιημένες διαδικασίες build και test. Αυτό διασφαλίζει ότι ο νέος κώδικας ενσωματώνεται με στο υπάρχον code base και πληροί τα απαιτούμενα πρότυπα ποιότητας.

- **Version Control**: Ένα ουσιαστικό στοιχείο του CI, που συνήθως επιτυγχάνεται μέσω των Git repositories που παρακολουθούν τις αλλαγές και διευκολύνουν τη συνεργασία μεταξύ των μελών της ομάδας.
- **Automated Build**: Μετά από κάθε push στον κώδικα, αυτοματοποιημένα εργαλεία μεταγλωττίζουν τον πηγαίο κώδικα σε εκτελέσιμο κώδικα και βιβλιοθήκες. Αυτό ελέγχει για σφάλματα νωρίς στη διαδικασία.
- **Automated Testing**: Εκτελούνται αμέσως μετά το στάδιο build, εκτελούνται αυτοματοποιημένες δοκιμές για να διασφαλιστεί ότι οι αλλαγές δεν παραβιάζουν τις υπάρχουσες λειτουργίες.

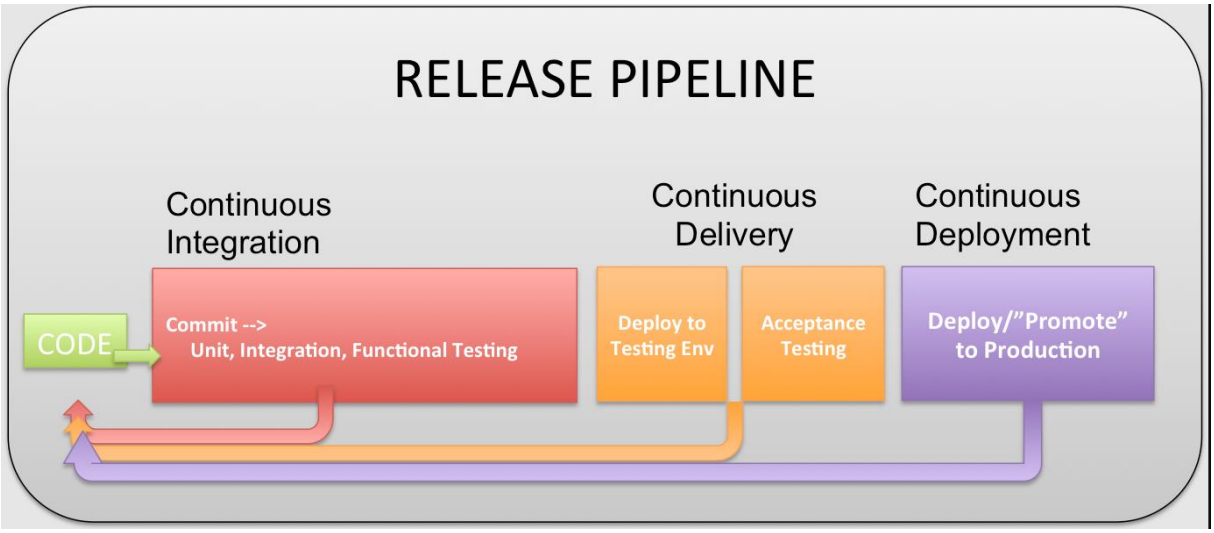


Σχήμα 5.9.1: CI diagram

4.9.2 Βασικές αρχές του Continuous Deployment

Το Continuous Deployment οδηγεί το CI στο επόμενο βήμα κάνοντας deploy αυτόματα όλες τις αλλαγές κώδικα σε ένα περιβάλλον παραγωγής αφού περάσει ένα σύνολο αυτοματοποιημένων tests. Τα βασικά συστατικά περιλαμβάνουν:

- **Automated Deployment**: Τα αυτοματοποιημένα εργαλεία πραγματοποιούν το deployment εξαλείφοντας την ανάγκη για χειροκίνητη παρέμβαση.
- **Environment Consistency**: Διασφαλίζει ότι το deployment περιβάλλον είναι συνεπές σε διάφορα στάδια, μειώνοντας έτσι τις ασυνέπειες και τα σφάλματα.
- **Monitoring and Feedback Loop**: Τα εργαλεία για την παρακολούθηση του συστήματος σε πραγματικό χρόνο και η παροχή άμεσου feedback είναι απαραίτητα για την πραγματοποίηση έγκαιρων προσαρμογών.



Σχήμα 5.9.2: CD diagram

4.9.3 Εργαλεία και Τεχνολογίες CI/CD

Υπάρχουν διάφορα εργαλεία για τη διευκόλυνση του CI/CD. Για παράδειγμα, τα Jenkins, GitLab CI/CD και GitHub Actions χρησιμοποιούνται συνήθως για CI/CD pipelines. Στο έργο μας, χρησιμοποιήθηκαν GitHub Actions για τη διαχείριση διαδικασιών CI/CD. Αυτό το εργαλείο μας δίνει τη δυνατότητα να ορίσουμε ροές εργασίας για τη δημιουργία, τη δοκιμή και το αυτόματο deployment του κώδικά μας κάθε φορά που ενημερώνεται ο κώδικας.

4.9.4 Σημασία και οφέλη του CI/CD

Η προσέγγιση CI/CD έχει πολλά πλεονεκτήματα:

- **Ταχύτεροι κύκλοι ανάπτυξης**: Οι μικρότεροι κύκλοι ανάπτυξης σημαίνουν πιο γρήγορες εκδόσεις κώδικα στο σύστημα παραγωγής.
- **Διασφάλιση ποιότητας**: Τα συνεχείς tests διασφαλίζουν ότι τα σφάλματα ανακαλύπτονται και διορθώνονται πιο γρήγορα.
- **Scalability**: Τα CI/CD pipelines μπορούν εύκολα να κάνουν scale, επιτρέποντας την ανάπτυξη και την πολυπλοκότητα καθώς εξελίσσεται το έργο.
- **Μείωση κινδύνου**: Οι μικρότερες, πιο συχνές αλλαγές είναι πιο εύκολο να διαχειριστούν από μεγάλα deployments, μειώνοντας τον συνολικό κίνδυνο.

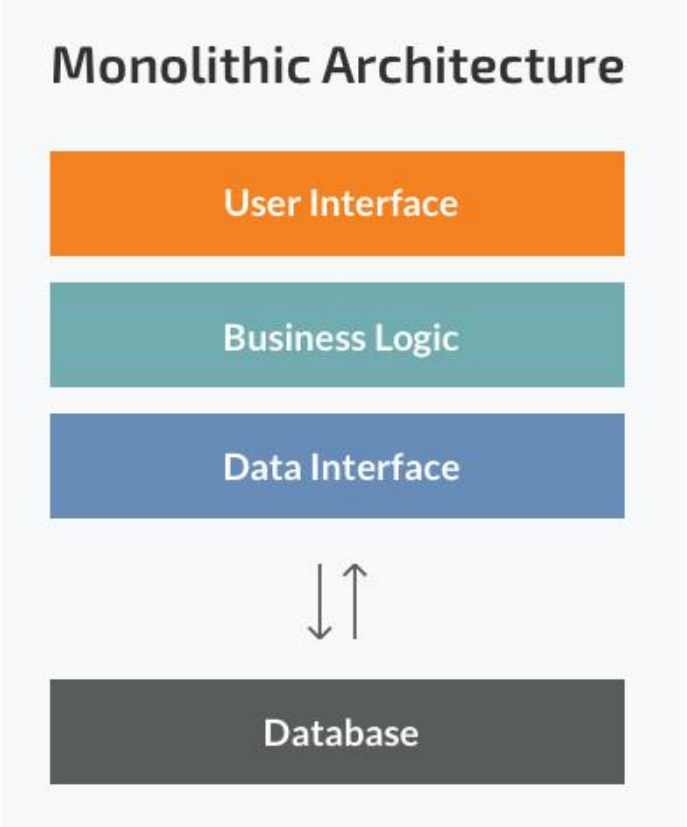
4.9.5 Συμπέρασμα

Το CI/CD δεν είναι απλώς ένα σύνολο πρακτικών, αλλά μια κουλτούρα που επιτρέπει στους προγραμματιστές να παρέχουν ποιοτικό λογισμικό γρήγορα και αποτελεσματικά. Η επαναληπτική, αυτοματοποιημένη προσέγγισή του καθιστά ευκολότερη την ενσωμάτωση, τη δοκιμή και το deployment αλλαγών του κώδικα. Μέσω διαφόρων εργαλείων και τεχνολογιών, τα CI/CD pipelines διευκολύνουν τη διαδικασία deployment, εξοικονομώντας χρόνο και πόρους.

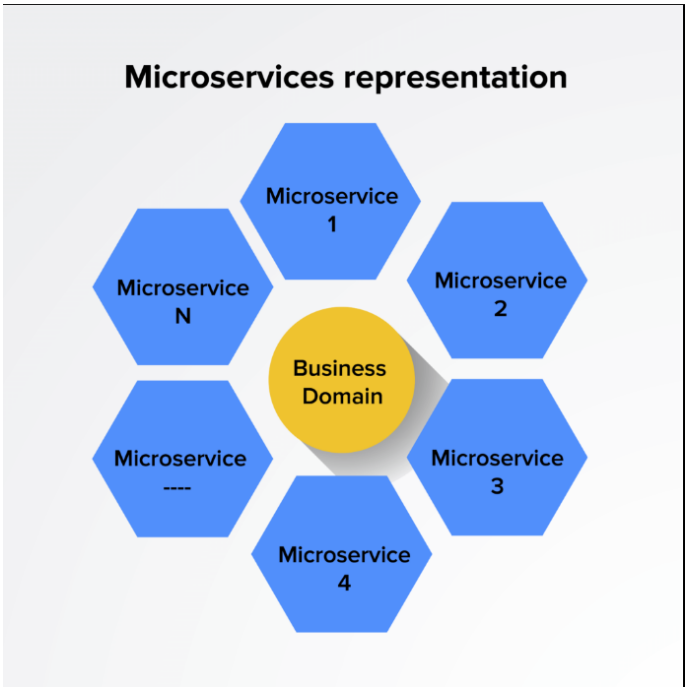
Υιοθετώντας τις αρχές CI/CD, οι ομάδες μπορούν να παραμείνουν πιο ευέλικτες, να προσαρμοστούν στις αλλαγές πιο γρήγορα και να οδηγήσουν σε υψηλότερη ικανοποίηση των πελατών, η οποία, με τη σειρά της, μεταφράζεται σε επιχειρηματική επιτυχία.

4.10 Microservices

Η αρχιτεκτονική Microservices είναι μια μέθοδος ανάπτυξης συστημάτων λογισμικού που αποτελούνται από loosely coupled, ανεξάρτητες deployable μονάδες ή υπηρεσίες. Κάθε microservice είναι μια αυτόνομη μονάδα με μοναδική ευθύνη στο συνολικό σύστημα, παρέχοντας συχνά μια συγκεκριμένη λειτουργία ή διαδικασία. Αυτό το αρχιτεκτονικό στυλ έρχεται σε αντίθεση με τις μονολιθικές αρχιτεκτονικές, όπου η εφαρμογή είναι χτισμένη ως μια ενιαία, αδιαίρετη ενότητα.



Σχήμα 5.10: Monlith architecture



Σχήμα 5.10: Microservice architecture

4.10.1 Χαρακτηριστικά των Microservices

- **Ενιαία ευθύνη**: Κάθε υπηρεσία σε μια αρχιτεκτονική microservices έχει σχεδιαστεί για να εκτελεί μια ενιαία λογική εργασία. Για παράδειγμα το payments service, user authentication service, κ.α.

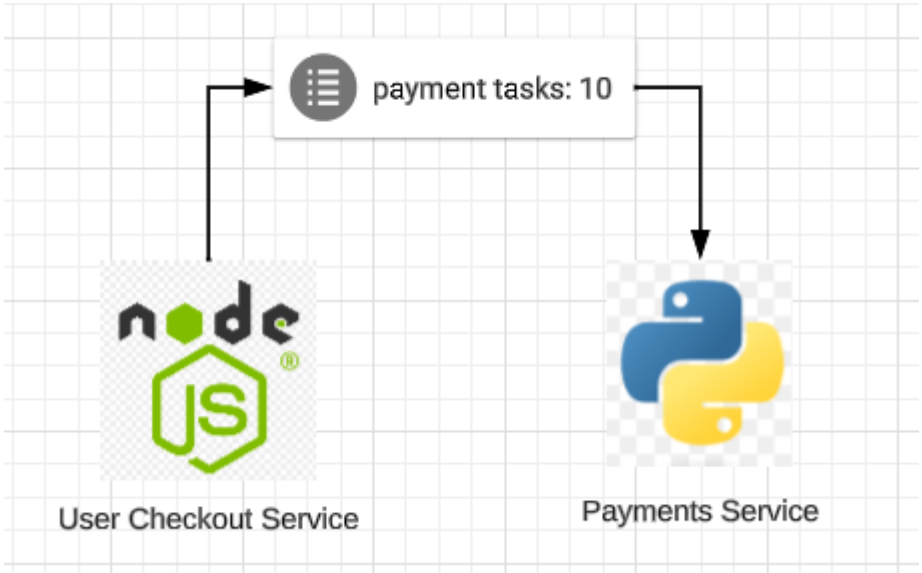
- **Ανεξαρτησία**: Τα microservices μπορούν να αναπτυχθούν, να γίνουν deployed και να κάνουν scale ανεξάρτητα, συχνά από διαφορετικές ομάδες που ειδικεύονται σε συγκεκριμένες επιχειρηματικές λειτουργίες.

- **Decentralized Data Management**: Κάθε υπηρεσία έχει τη δική της βάση δεδομένων, που της επιτρέπει να διαχειρίζεται τα δεδομένα της ανεξάρτητα από άλλες υπηρεσίες
-

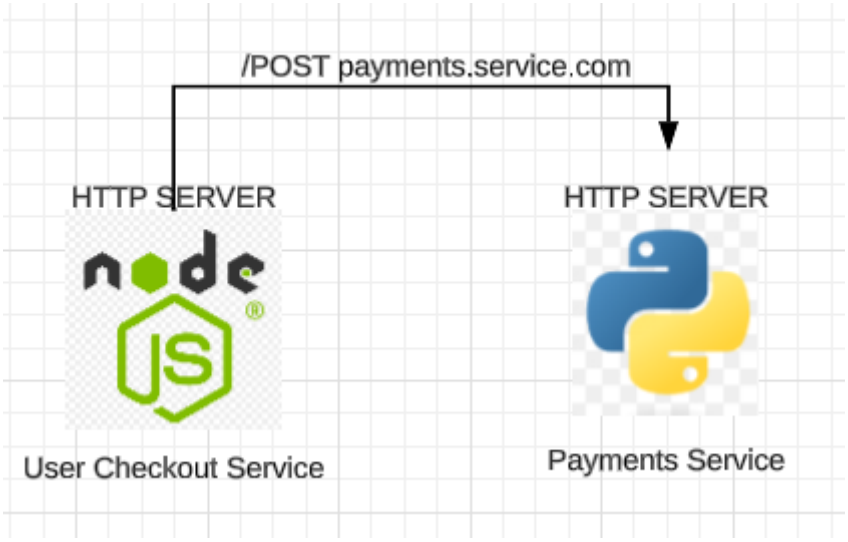
Σχήμα 5.10.1: Microservice characteritcs

4.10.2 Πλεονεκτήματα των Microservices

- **Language Agnostic**: Ένα από τα πιο σημαντικά χαρακτηριστικά μιας αρχιτεκτονικής microservice είναι ο γλωσσο-αγνωστικιστικός της χαρακτήρας. Μπορούν να αναπτυχθούν διαφορετικές υπηρεσίες σε διαφορετικές γλώσσες προγραμματισμού, αρκεί να τηρούν καλά καθορισμένα API και πρωτόκολλα επικοινωνίας. Αυτή η ευελιξία επιτρέπει στις ομάδες να επιλέξουν την καλύτερη τεχνολογία για τις συγκεκριμένες ανάγκες κάθε υπηρεσίας.

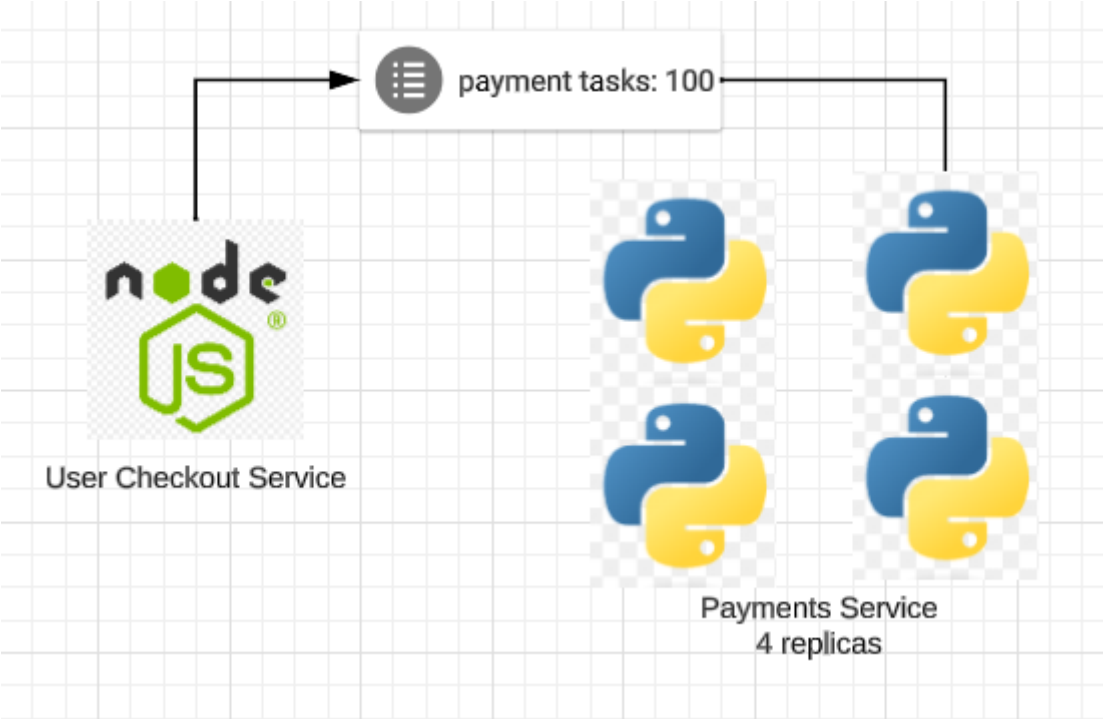


Σχήμα 5.10.2: Language Agnostic model



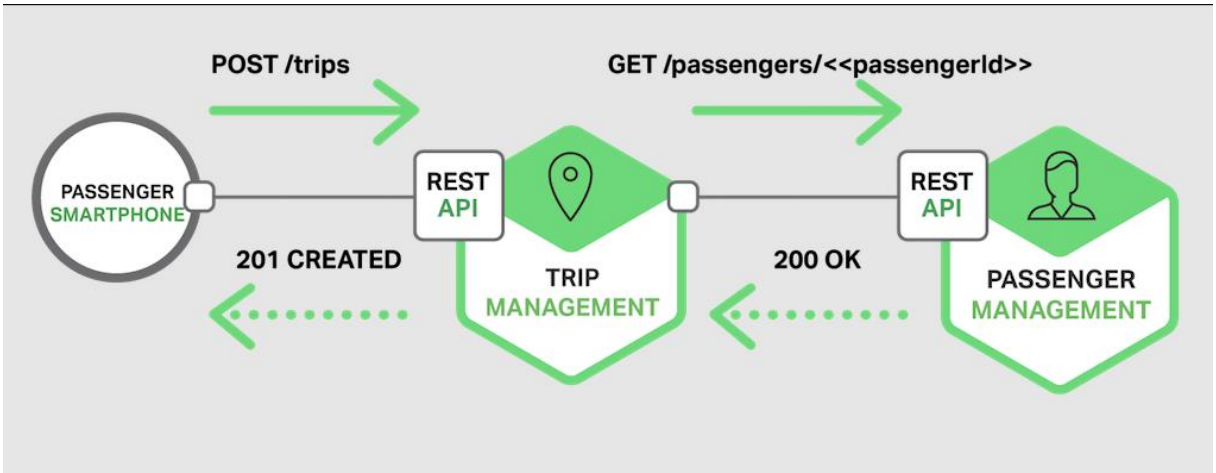
Σχήμα 5.10.2: Language Agnostic model 2

- **Scalability**: Η modular φύση των microservice επιτρέπει το οριζόντιο scaling, παρέχοντας μεγαλύτερη ευελιξία για τη διαχείριση αυξημένων φορτίων.
- **Απομόνωση σφαλμάτων**: Η αποτυχία μιας μεμονωμένης υπηρεσίας συνήθως δεν οδηγεί σε αποτυχία σε όλο το σύστημα.
- **Ευκολία Deployment και αλλαγών**: Οι ανεξάρτητες υπηρεσίες μπορούν να γίνουν deploy, να τροποποιηθούν ή να κάνουν scale χωρίς να επηρεαστεί η λειτουργία άλλων υπηρεσιών.
- **Loosely**-Coupled: Αυτός ο στο πλαίσιο της αρχιτεκτονικής microservices αναφέρεται στον βαθμό ανεξαρτησίας και αυτονομίας που έχουν μεμονωμένες υπηρεσίες μεταξύ τους. Σε ένα loosely-coupled σύστημα, κάθε υπηρεσία έχει σχεδιαστεί ώστε να είναι μια αυτόνομη μονάδα που μπορεί να λειτουργεί ανεξάρτητα από τις άλλες. Αυτή η φιλοσοφία σχεδίασης ελαχιστοποιεί τις άμεσες εξαρτήσεις μεταξύ των υπηρεσιών, επιτρέποντάς τους να εξελίσσονται, να κάνουν scale και να αποτυγχάνουν ανεξάρτητα χωρίς να προκαλούν διαταραχές σε όλο το σύστημα. Στην πράξη, το loosely-coupled επιτυγχάνεται συχνά μέσω καλά καθορισμένων interfaces, συνήθως με τη μορφή API, και τυπικών πρωτοκόλλων επικοινωνίας όπως το HTTP ή τις ουρές μηνυμάτων. Οι υπηρεσίες αλληλεπιδρούν μόνο μέσω αυτών των interfaces, παραμένοντας agnostic για τις εσωτερικές λειτουργίες του άλλου microsrrvice. Αυτό διασφαλίζει ότι οι αλλαγές στην εσωτερική υλοποίηση μιας υπηρεσίας δεν θα παραβιάσουν άλλες υπηρεσίες που αλληλεπιδρούν με αυτήν, εφόσον η διεπαφή παραμένει συνεπής.



Σχήμα 5.10.2: independent scaling

- **Inter-Service Communication**: Τα microservices επικοινωνούν μεταξύ τους μέσω καλά καθορισμένων API και πρωτοκόλλων όπως το HTTP/REST ή οι ουρές ανταλλαγής μηνυμάτων. Αυτό επιτρέπει σε κάθε υπηρεσία να αλληλεπιδρά εύκολα με άλλες και υποστηρίζει επίσης την ενσωμάτωση υπηρεσιών που είναι γραμμένες σε διαφορετικές γλώσσες προγραμματισμού.



Σχήμα 5.10.3: Interservice communication

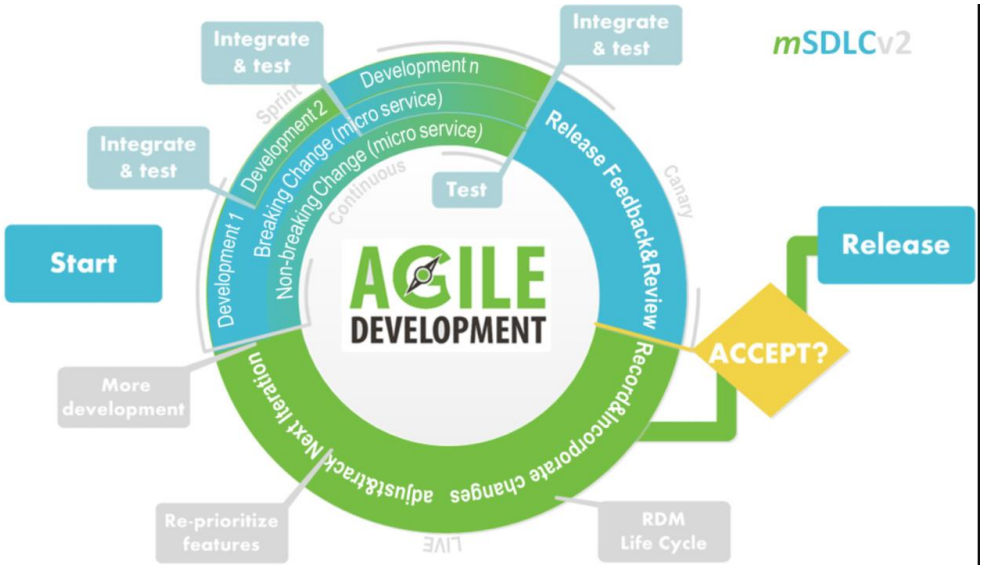
4.10.3 Προκλήσεις και Περιορισμοί

- **Πολυπλοκότητα**: Η διαχείριση πολλαπλών υπηρεσιών και οι αλληλεπιδράσεις τους μπορεί να είναι περίπλοκη.
- **Συνέπεια δεδομένων**: Η αποκεντρωμένη διαχείριση δεδομένων θα μπορούσε να οδηγήσει σε ζητήματα consistency.
- **Καθυστέρηση δικτύου**: Το Inter-service communication μέσω ενός δικτύου μπορεί να προκαλέσει καθυστέρηση.

4.10.4 Συμπέρασμα

Η αρχιτεκτονική Microservices προσφέρει έναν ισχυρό τρόπο για τη δημιουργία scalable, ευέλικτων συστημάτων λογισμικού. Αναλύοντας μια εφαρμογή σε μικρότερα, loosely-coupled στοιχεία, οι οργανισμοί μπορούν να επιτύχουν μεγαλύτερη ευελιξία, ανθεκτικότητα και επεκτασιμότητα. Ωστόσο, η αρχιτεκτονική έρχεται με το δικό της σύνολο πολυπλοκοτήτων και προκλήσεων, και ως εκ τούτου, πρέπει να εφαρμοστεί προσεκτικά.

Η αρθρωτή προσέγγιση των microservices όχι μόνο υποστηρίζει τις διαδικασίες ανάπτυξης και εγκατάστασης αλλά βελτιώνει επίσης σημαντικά την επεκτασιμότητα και την ανθεκτικότητα των συστημάτων λογισμικού. Είναι ένα συναρπαστικό αρχιτεκτονικό παράδειγμα που έχει ουσιαστικές επιπτώσεις στον τρόπο με τον οποίο συλλαμβάνουμε, κατασκευάζουμε και εξελίσσουμε το λογισμικό.



Σχήμα 5.10.4: Microservices lifecycle

4.11 Επίλογος

Ολοκληρώνοντας αυτό το κεφάλαιο για τα DevOps, πραγματοποιήσαμε μια εκτενή και λεπτομερή εξέταση της μεθοδολογίας DevOps - μια πολύπλευρη προσέγγιση που ενσωματώνει διάφορους κλάδους στην ανάπτυξη λογισμικού και τις λειτουργίες πληροφορικής. Ο στόχος ήταν να διαφωτιστεί πώς το DevOps είναι πολύ περισσότερα από ένα απλό σύνολο εργαλείων. Είναι μια πολιτιστική φιλοσοφία που επιφέρει μια αρμονική συνεργασία μεταξύ των παραδοσιακά δύο ξεχωριστών και κάπως απομονωμένων ρόλων: ανάπτυξης (development) και λειτουργίας (operations)

Σχήμα 5.11: Infinity DevOps cicle

Η ομιλία μας ξεκίνησε με τα θεμελιώδη στάδια που περιλαμβάνουν τον κύκλο ζωής του DevOps, το καθένα ξεχωριστό αλλά αλληλένδετο. Το στάδιο «Plan» χρησιμεύει ως το εννοιολογικό σημείο εκκίνησης, ενεργώντας ως το θεμέλιο όπου οι εργασίες αναλύονται και προγραμματίζονται σε ευέλικτα sprints. Από εκεί, περάσαμε στο στάδιο «Code», δίνοντας έμφαση στο ρόλο του ως ο πυρήνας του κύκλου ζωής του DevOps όπου οι αφηρημένες ιδέες μετατρέπονται σε πραγματικά στοιχεία λογισμικού.

Η μετάβαση από τον κώδικα στο πραγματικό λογισμικό απαιτεί τα «Build» και «Test», στάδια που καλύψαμε για να υπογραμμίσουμε τη σημασία τους στη διασφάλιση ποιότητας και στη λειτουργικότητα. Μετά από αυτό, αγγίξαμε το στάδιο «Deploy», το οποίο περιλαμβάνει διάφορες στρατηγικές όπως αναπτύξεις Blue/Green για να διασφαλίσουμε απρόσκοπτες ενημερώσεις εφαρμογών και rollbacks. Τέλος, συζητήσαμε για το «Monitoring», τη φάση που είναι υπεύθυνη για την παρακολούθηση της εφαρμογής, τη διασφάλιση της ομαλής λειτουργίας της και τη συλλογή δεδομένων για μελλοντικούς κύκλους βελτιώσεων.

Στη συνέχεια, μιλήσαμε για το contrainerization, με το Docker να προσδιορίζεται ως το εργαλείο επιλογής για αυτό το έργο. Εδώ, η συζήτηση επικεντρώθηκε στον τρόπο με τον οποίο τα Docker containers ενσωματώνουν την εφαρμογή και το περιβάλλον της, διασφαλίζοντας συνεπή συμπεριφορά σε πολλαπλά υπολογιστικά περιβάλλοντα

Η συζήτηση κινήθηκε προς το Continues Integration/ Continues Deployment (CI/CD), ένα σύνολο πρακτικών που περιλαμβάνει αυτοματοποιημένα pipelines που κάνουν build, test και deploy το code base με ελάχιστη χειροκίνητη παρέμβαση, επιταχύνοντας έτσι τη διαδικασία παράδοσης και μειώνοντας το περιθώριο λάθους.

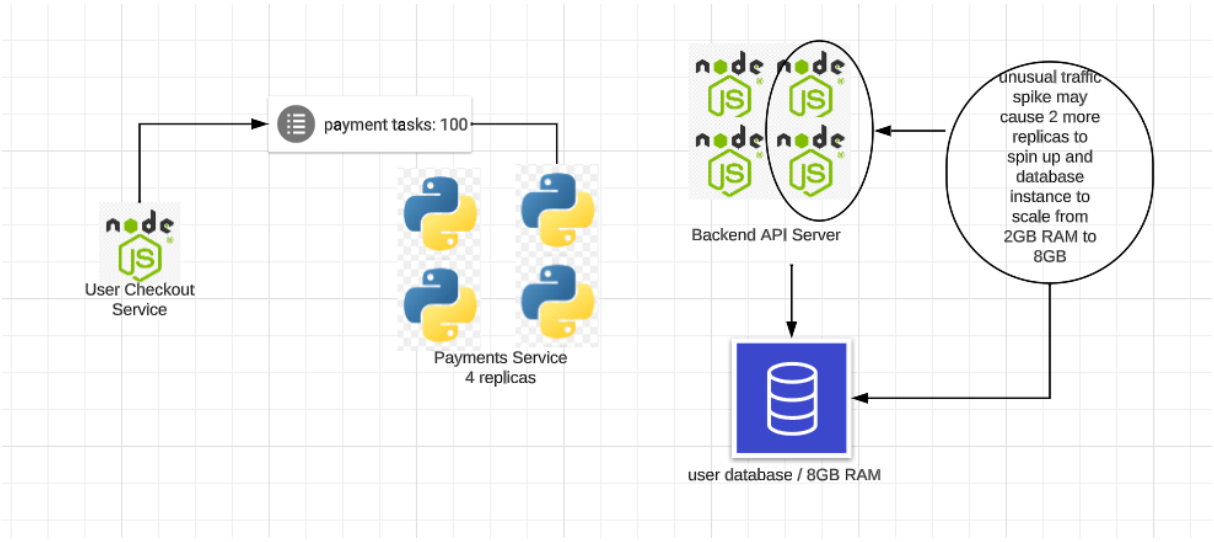
Η εμφάνισή μας στο τοπίο του DevOps δεν θα ήταν ολοκληρωμένη χωρίς να εμβαθύνουμε στον κόσμο των microservices. Ως αρθρωτά στοιχεία που μπορούν να αναπτυχθούν, να αναπτυχθούν και να κάνουν scale ανεξάρτητα, τα microservices αντιπροσωπεύουν ένα διαφορετικό παράδειγμα από τις παραδοσιακές μονολιθικές αρχιτεκτονικές.

Συνοπτικά, το κεφάλαιο έχει σχεδιαστεί για να χρησιμεύσει τόσο ως θεωρητικός οδηγός όσο και ως πρακτικό εγχειρίδιο, προσφέροντας μια εις βάθος κατανόηση των αρχών και πρακτικών του DevOps, των εφαρμογών και των συνεπειών τους. Αυτές οι μεθοδολογίες, τα εργαλεία και οι πρακτικές συμβάλλουν συλλογικά σε έναν ταχύτερο, πιο αποτελεσματικό και πιο ισχυρό κύκλο ζωής ανάπτυξης λογισμικού, επιδεικνύοντας τη μεταμορφωτική δύναμη του DevOps στο ταχέως εξελισσόμενο τεχνολογικό τοπίο του σήμερα.

Κεφάλαιο 5ο: Scalability

5.1 Εισαγωγή

Σε αυτό το κεφάλαιο, εμβαθύνουμε σε βάθος στη σφαίρα του scaling, ένα χαρακτηριστικό που συχνά καθορίζει την επιτυχία ή την αποτυχία των σύγχρονων εφαρμογών λογισμικού. Ο όρος «επεκτασιμότητα» μπορεί να ακούγεται απλός, αλλά οι επιπτώσεις και οι εφαρμογές του είναι πολύπλευρες και πολύπλοκες. Σε μια εποχή όπου οι χρήστες απαιτούν υψηλή διαθεσιμότητα και απρόσκοπτες εμπειρίες, η ικανότητα ενός συστήματος να προσαρμόζεται αποτελεσματικά σε διαφορετικά επίπεδα ζήτησης είναι πρωταρχικής σημασίας.



Σχήμα 6.1: Scalable system example

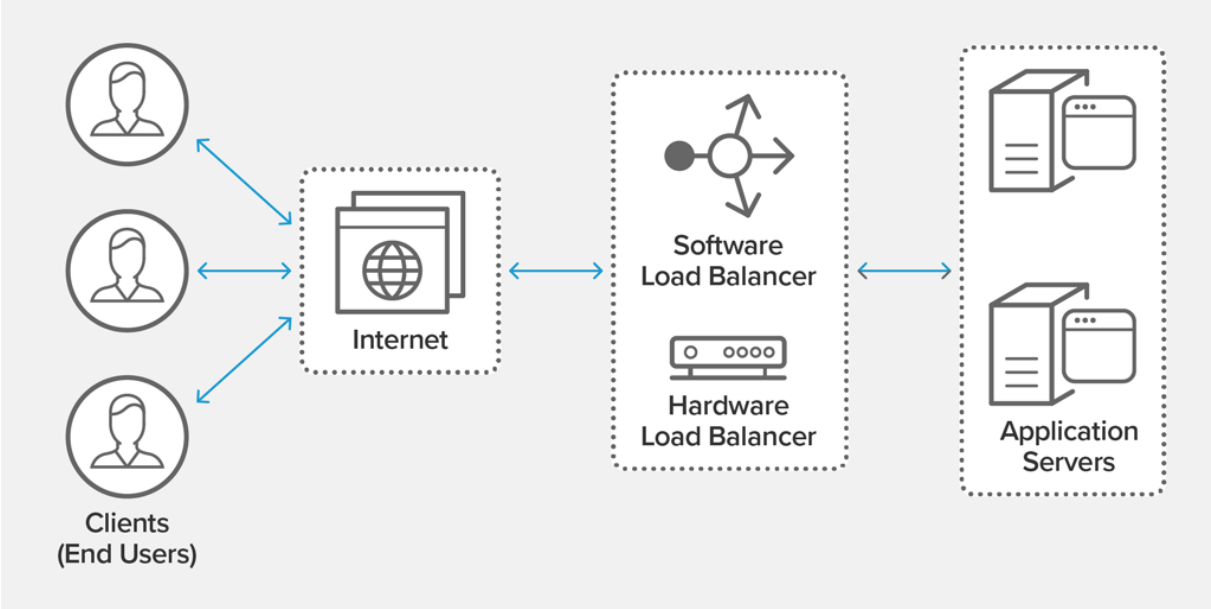
Η ανάγκη για επεκτασιμότητα δεν αφορά μόνο την υποδοχή ενός μεγάλου αριθμού χρηστών αλλά και τη βελτιστοποίηση της χρήσης πόρων και της απόδοσης του συστήματος υπό διαφορετικές συνθήκες λειτουργίας. Από τις τεχνικές load balance έως το partition βάσεων δεδομένων και την αρχιτεκτονική microservices, αυτό το κεφάλαιο θα διερευνήσει τους διάφορους μηχανισμούς που συμβάλλουν συλλογικά στο να γίνει ένα σύστημα επεκτάσιμο.

Σε αυτό το πλαίσιο, θα συζητήσουμε τις θεωρητικές αρχές που διέπουν τα scalable συστήματα, τους τύπους scaling —οριζόντιο και κάθετο— και τις προκλήσεις που συχνά συνοδεύουν τις προσπάθειες scaling ενός συστήματος. Είτε είμαστε διαχειριστής που προσπαθεί να διατηρήσει την ακεραιότητα του συστήματος σε περιόδους αιχμής χρήσης είτε προγραμματιστής που προσπαθεί να δημιουργήσει επεκτασιμότητα σε μια νέα εφαρμογή, οι ακόλουθες ενότητες θα σας εξοπλίσουν με τις βασικές γνώσεις και τις βέλτιστες πρακτικές για να περιηγηθείτε στις πολυπλοκότητες της επεκτασιμότητας.

Στο τέλος αυτού του κεφαλαίου, θα έχουμε μια ολοκληρωμένη κατανόηση του τι συνεπάγεται η επεκτασιμότητα, την εγγενή της αξία και πώς να την εφαρμόσετε καλύτερα σε διάφορα σενάρια. Στόχος μας είναι να σας παρέχουμε μια ολοκληρωμένη, σε βάθος κατανόηση του θέματος, ώστε να μπορέσουμε να εφαρμόσουμε αυτές τις αρχές αποτελεσματικά στα δικά μας έργα.

5.2 Load Balancing

Το load balance είναι ένα κρίσιμο στοιχείο της επεκτασιμότητας και της υψηλής διαθεσιμότητας σε κάθε σύγχρονο σύστημα λογισμικού. Στον πυρήνα του, το Load balance είναι η πρακτική της διανομής της εισερχόμενης κίνησης δικτύου ή εφαρμογών σε πολλούς servers ή κόμβους. Αυτό διασφαλίζει αποτελεσματικά ότι κανένας διακομιστής δεν κατακλύζεται, διατηρώντας έτσι τα βέλτιστα επίπεδα απόδοσης.



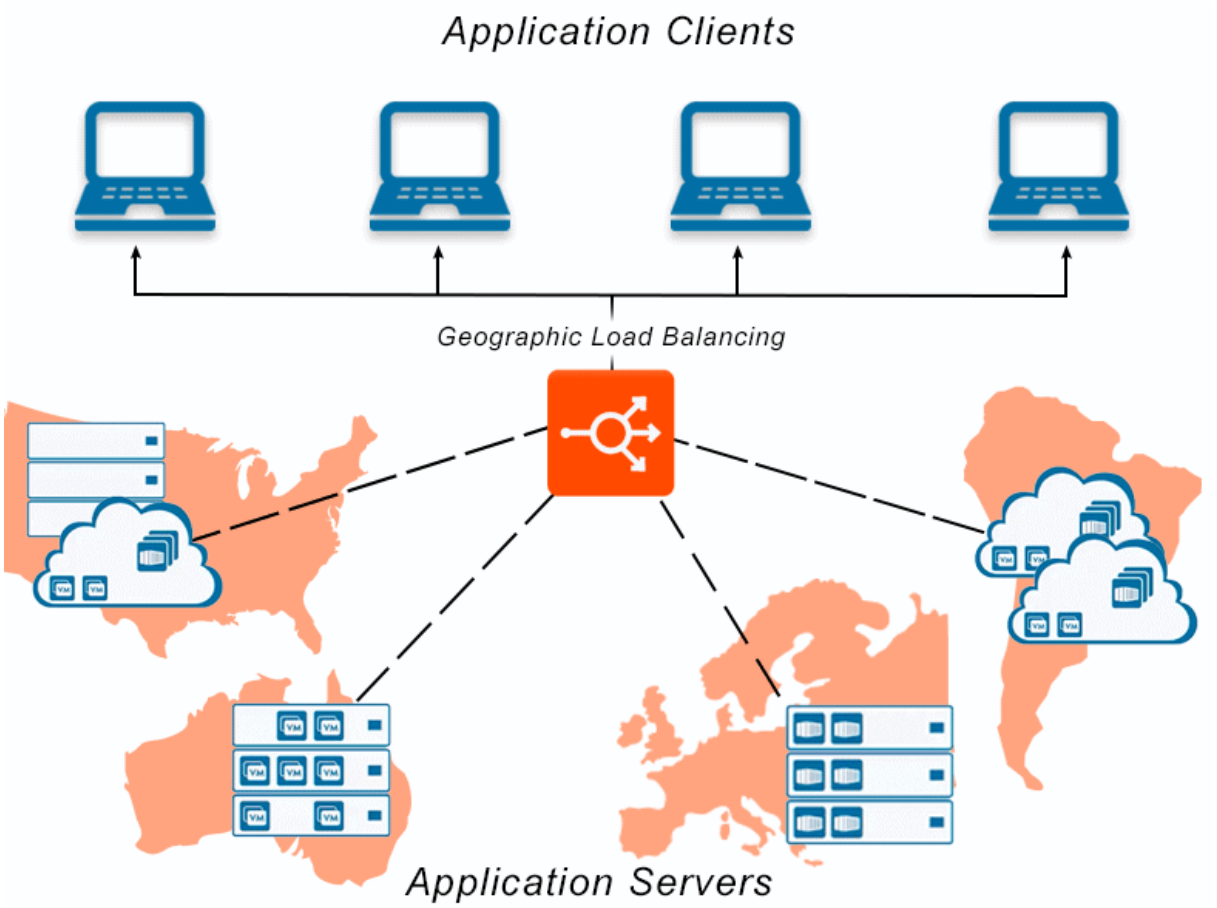
Σχήμα 6.2: Load balancer diagram

Αν και η ιδέα είναι απλή, η εφαρμογή του load balancing στον πραγματικό κόσμο είναι πολύ πιο περίπλοκη. Διαφορετικές στρατηγικές, αλγόριθμοι και λύσεις υλικού/λογισμικού μπαίνουν στο παιχνίδι, το καθένα με τα δικά του πλεονεκτήματα, μειονεκτήματα και περιπτώσεις χρήσης.

Οι κοινοί αλγόριθμοι περιλαμβάνουν το Round Robin, το Least Connections και το IP Hashing. Το Round Robin είναι το απλούστερο, που διανέμει αιτήματα με κυκλικό τρόπο. Το Least Connections κατευθύνει την κυκλοφορία στον διακομιστή με τις λιγότερες ενεργές συνδέσεις, ιδανικό για περιπτώσεις όπου οι περίοδοι αιτημάτων έχουν διαφορετική διάρκεια. Το IP Hashing κατευθύνει όλα τα αιτήματα από μια συγκεκριμένη IP στον ίδιο διακομιστή, κάτι που είναι χρήσιμο για τη διατήρηση των περιόδων σύνδεσης χρήστη (sticky connection).

Πέρα από τους αλγόριθμους, το πεδίο του load balancing επεκτείνεται σε πιο προηγμένες στρατηγικές όπως το application-layer load balancing, όπου ο load balancer λαμβάνει αποφάσεις με βάση το περιεχόμενο του αιτήματος.

Για οργανισμούς με παγκόσμιο κοινό, η γεωγραφική εξισορρόπηση φόρτου μπορεί να είναι απαραίτητη. Αυτό περιλαμβάνει τη δρομολόγηση των αιτημάτων των χρηστών στο πλησιέστερο κέντρο δεδομένων για τη μείωση του χρόνου απόκρισης. Είναι μια προηγμένη μορφή load balancing που απαιτεί βαθιά κατανόηση της δικτύωσης και της ανάλυσης DNS.



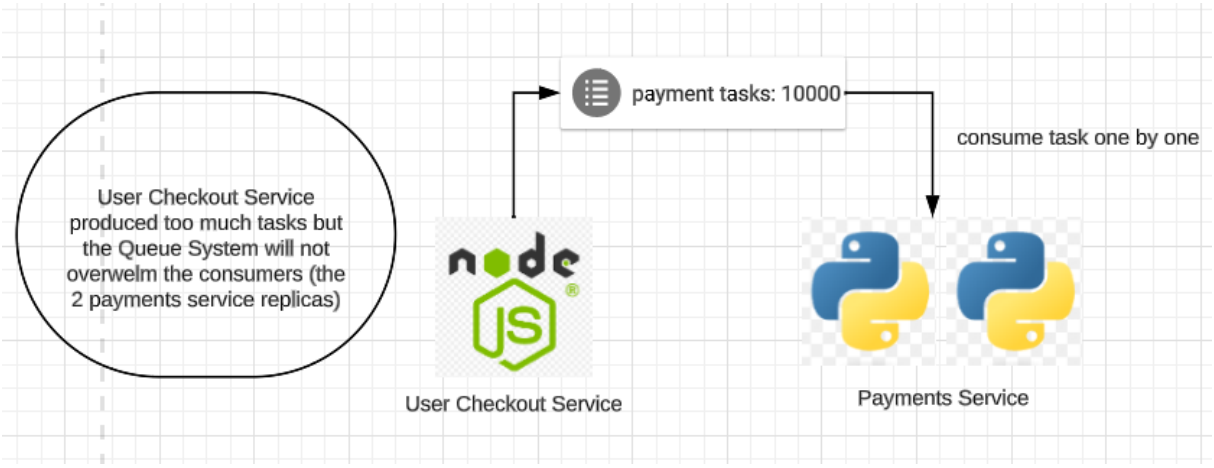
Σχήμα 6.2: Geographical load balancing

Για να το συνοψίσουμε, η εξισορρόπηση φορτίου δεν είναι απλώς μια "καλή χρήση", αλλά μια θεμελιώδης ανάγκη για κάθε σύστημα που στοχεύει σε υψηλή διαθεσιμότητα και αποτελεσματική χρήση πόρων.

5.3 Queue Management

Οι μηχανισμοί ουράς είναι τα θεμελιώδη δομικά στοιχεία για τη δημιουργία μιας πραγματικά scalable αρχιτεκτονικής κατανεμημένου συστήματος. Η έννοια της ουράς έχει τις ρίζες της σε σενάρια του πραγματικού κόσμου, παρόμοια με μια σειρά ανθρώπων που περιμένουν τη σειρά τους για να επωφεληθούν από μια υπηρεσία. Σε ένα περιβάλλον λογισμικού, οι ουρές εκτελούν παρόμοια λειτουργία λειτουργώντας ως χώρος αναμονής για μηνύματα ή εργασίες, διασφαλίζοντας μια ισορροπημένη κατανομή του φόρτου εργασίας σε όλο το σύστημα.

Η ομορφιά των ουρών έγκειται στην ικανότητά τους να ενσωματώνονται απρόσκοπτα με διάφορα στοιχεία ενός συστήματος, παρέχοντας έτσι ένα στρώμα αφαιρετικότητας. Η χρήση ουρών βοηθά στην απομόνωση των υπηρεσιών producer και consumer, διασφαλίζοντας ότι ένα υψηλό ποσοστό εισερχόμενων εργασιών δεν κατακλύζει το σύστημα.

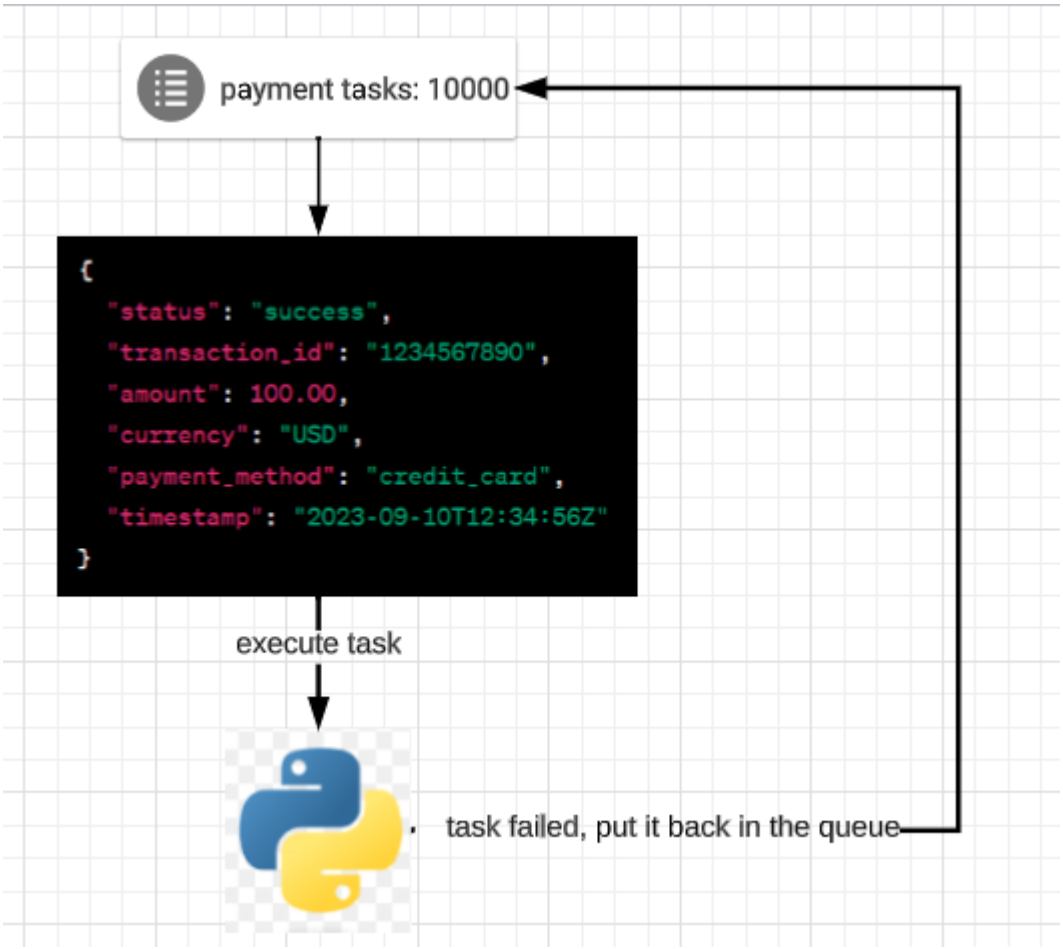


Σχήμα 6.3: Queue with many tasks diagram

Χρησιμοποιώντας ουρές, το σύστημα μπορεί να χειριστεί αποτελεσματικά μεγάλα φορτία αναθέτοντας δυναμικά εργασίες από την ουρά σε διαθέσιμους πόρους. Αυτός ο μηχανισμός δεν είναι απλώς ένα ενδεχόμενο για σενάρια αιχμής αλλά και ένας βελτιστοποιημένος τρόπος κατανομής πόρων κατά τις περιόδους αδράνειας, καθιστώντας το σύστημα προσαρμόσιμο σε μια σειρά σεναρίων.

Επιπλέον, οι ουρές προσφέρουν την ευελιξία για την εισαγωγή μηχανισμών προτεραιότητας, ενισχύοντας έτσι την ικανότητα του συστήματος να χειρίζεται εργασίες με βάση τον επείγοντα χαρακτήρα ή τη σημασία τους. Με αυτόν τον τρόπο, ακόμη και υπό μεγάλο φορτίο, οι κρίσιμες εργασίες μπορούν να τεθούν σε προτεραιότητα έναντι των λιγότερο σημαντικών, διασφαλίζοντας ότι το σύστημα παραμένει ανταποκρινόμενο και πληροί βασικά κριτήρια απόδοσης.

Επιπλέον, η χρήση ουρών επιτρέπει πιο λεπτομερή παρακολούθηση και πιο εύρωστο χειρισμό σφαλμάτων. Εάν ένα μέρος του συστήματος αποτύχει, οι εργασίες στην ουρά μπορούν να επαναδρομολογηθούν σε άλλο μέρος του συστήματος ή να τοποθετηθούν ξανά στην ουρά για μεταγενέστερη επεξεργασία, προσθέτοντας έτσι ένα επίπεδο ελαστικότητας.

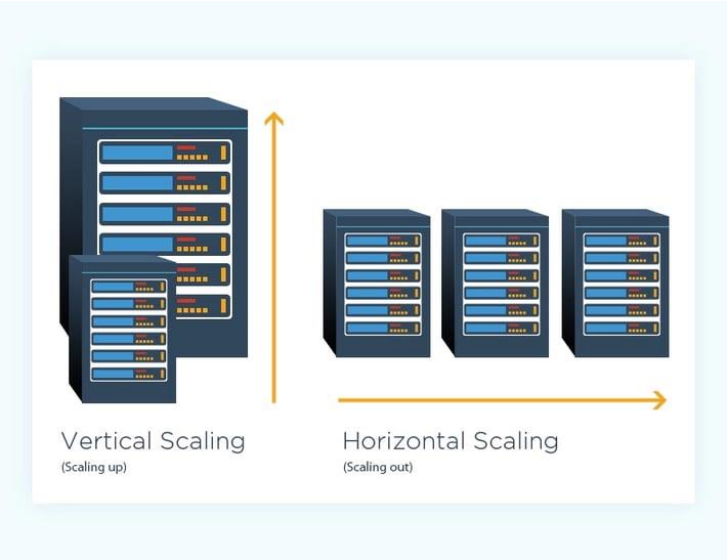


Σχήμα 6.3: Failed task re-routing

Συνοπτικά, οι ουρές είναι πολύ περισσότερα από ένα απλό buffer. αποτελούν στρατηγικό πλεονέκτημα σε μια κατανεμημένη αρχιτεκτονική που προάγει την επεκτασιμότητα, την ανθεκτικότητα και τη βέλτιστη χρήση των πόρων. Η υιοθέτηση μηχανισμών ουράς, είτε είναι γενικοί είτε εξειδικευμένοι όπως το RabbitMQ, όχι μόνο επιλύει άμεσες τεχνικές προκλήσεις αλλά και προετοιμάζει το σύστημα για dynamic scaling.

5.4 Στρατηγικές Scaling

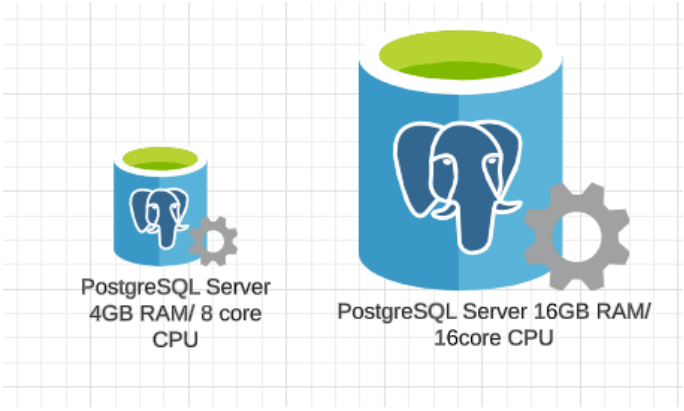
Οι στρατηγικές scaling είναι οι θεμελιώδεις πυλώνες που καθοδηγούν την προσέγγιση ενός οργανισμού για να χειριστεί ένα αυξανόμενο φορτίο στα συστήματα λογισμικού του. Με απλά λόγια, το scaling είναι η ικανότητα του συστήματος να προσαρμόζει με χάρη την ανάπτυξη χρηστών, δεδομένων ή συναλλαγών. Εδώ, εμβαθύνουμε σε διάφορες στρατηγικές που συχνά αξιοποιούνται για την επίτευξη της βέλτιστης επεκτασιμότητας.



Σχήμα 6.4.1: Scaling strategies

5.4.1 Vertical Scaling

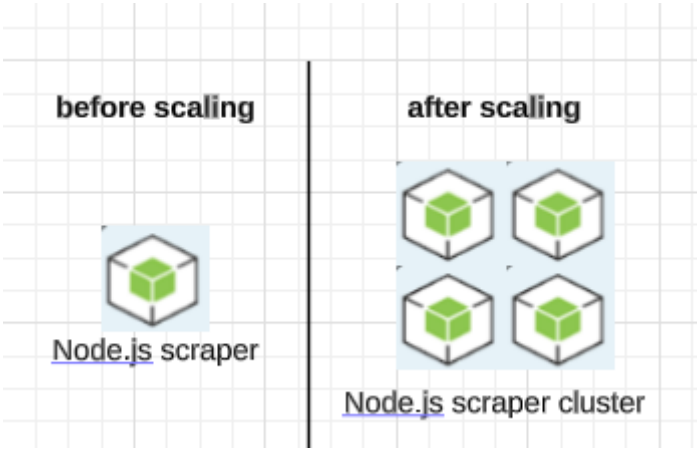
Γνωστό και ως "scaling up", αυτή η προσέγγιση περιλαμβάνει την αύξηση της ισχύος των μεμονωμένων πόρων μέσα σε ένα σύστημα. Θα μπορούσε να σημαίνει αύξηση των δυνατοτήτων της CPU, ενίσχυση της μνήμης ή ενίσχυση της αποθήκευσης. Το πλεονέκτημα του vertical scaling είναι η απλή φύση του, απλώς προσθέτουμε περισσότερη δύναμη στην υπάρχουσα αρχιτεκτονική μας. Ωστόσο, υπάρχουν περιορισμοί στο πόσο μπορεί να αναβαθμιστεί ένας μεμονωμένος server, καθιστώντας αυτήν την επιλογή λιγότερο εφικτή για μακροπρόθεσμο scaling.



Σχήμα 6.4.1:Verical scaling of db server

5.4.2 Horizontal Scaling

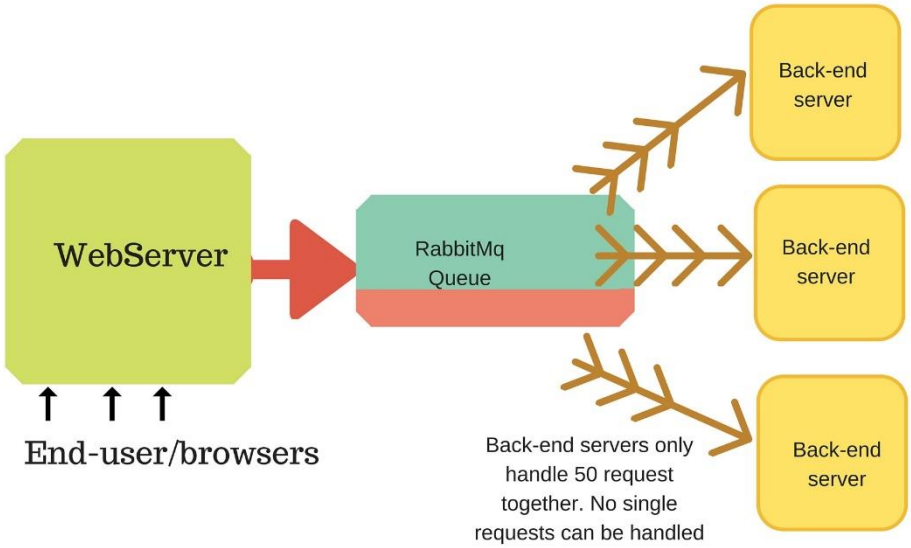
Συνήθως αναφέρεται ως "scaling out", το horizontal scaling συνίσταται στην προσθήκη περισσότερων μηχανών ή κόμβων στο υπάρχουν σύστημα. Σε αντίθεση με το vertical scaling, αυτή η στρατηγική δεν έχει προκαθορισμένο ανώτατο όριο και μπορεί να επεκταθεί επ' αόριστον. Αυτή η μέθοδος είναι ιδιαίτερα ωφέλιμη όταν το σύστημα αντιμετωπίζει διακοπτόμενες αιχμές στην κίνηση που πρέπει να διαχειρίζονται χωρίς να επηρεάζεται η απόδοση.



Σχήμα: 6.4.2: Horizontal Scaling example

5.4.3 Scaling με βάση τον όγκο tasks στην ουρά

Η εφαρμογή ουρών μπορεί να χρησιμεύσει ως μια ισχυρή στρατηγική για τον χειρισμό ακανόνιστης κυκλοφορίας. Τοποθετώντας μια ουρά μεταξύ των αιτημάτων των χρηστών και του διακομιστή, η εισερχόμενη κίνηση μπορεί να κρατηθεί προσωρινά, παρέχοντας στον διακομιστή τον χρόνο που απαιτείται για την επεξεργασία κάθε αιτήματος.



Σχήμα 6.4.3: Scaling by queue size

5.4.4 Auto Scaling

Μια πιο δυναμική προσέγγιση, το auto scaling περιλαμβάνει προσαρμογές σε πραγματικό χρόνο στους πόρους με βάση το τρέχον φορτίο. Αυτή η στρατηγική χρησιμοποιείται συχνά σε περιβάλλοντα cloud όπου το scaling μπορεί να αυτοματοποιηθεί με βάση συγκεκριμένες μετρήσεις απόδοσης. Για παράδειγμα, εάν το μέσο όρο χρήσης της CPU ενός cluster που αποτελείτε από 5 servers ξεπεράσει το

50% για 5 συνεχόμενα λεπτά, τότε σήκωσε άλλους 5 servers. Αντίστοιχα, εάν ο μέσος όρος χρήσης είναι χαμηλότερος από 20% για 5 συνεχόμενα λεπτά, τότε κλείσε 5 servers. Αυτή η μέθοδος μας επιτρέπει να ανταπεξέλθουμε στις απότομες αυξήσεις φορτίου (spikes) και να ελαχιστοποιήσουμε το κόστος χρήσης.

5.4.5 Geographical Scaling

Αυτή είναι μια επέκταση του horizontal scaling αλλά κατανεμημένη σε διαφορετικές γεωγραφικές τοποθεσίες. Αυτή η στρατηγική χρησιμοποιείται συχνά από παγκόσμιες πλατφόρμες για τη μείωση του λανθάνοντος χρόνου και τη βελτίωση της εμπειρίας χρήστη δρομολογώντας αιτήματα χρηστών στο πλησιέστερο κέντρο δεδομένων.

5.4.6 Συμπέρασμα

Με την υιοθέτηση μιας ή ενός συνδυασμού αυτών των στρατηγικών, οι οργανισμοί μπορούν να προσαρμόσουν την κλιμάκωσή τους για να ανταποκρίνονται σε συγκεκριμένες ανάγκες και συνθήκες, διασφαλίζοντας ότι είναι πάντα προετοιμασμένοι για αλλαγές στο φορτίο και μπορούν να παρέχουν μια σταθερά υψηλής ποιότητας εμπειρία στους χρήστες.

5.5 Επίλογος

Ολοκληρώνοντας τη συζήτηση σχετικά με τα scaling strategies, είναι σαφές ότι το θέμα δεν είναι απλώς μια τεχνική απαίτηση, αλλά μια επιχειρηματική επιταγή. Από το load balancing έως τη διαχείριση ουρών, κάθε πτυχή της επεκτασιμότητας στοχεύει στη βελτίωση της απόδοσης του συστήματος, της αξιοπιστίας και, εν τέλει, της ικανοποίησης των χρηστών. Αυτό το κεφάλαιο ήταν ένα διερευνητικό ταξίδι μέσα από τις διάφορες μεθοδολογίες και τεχνολογίες που κάνουν εφικτή την επεκτασιμότητα. Καθώς οι επιχειρήσεις αναπτύσσονται και οι απαιτήσεις των χρηστών εξελίσσονται, αυτές οι στρατηγικές θα συνεχίσουν να βρίσκονται στην πρώτη γραμμή κάθε καλά σχεδιασμένου, αποτελεσματικού συστήματος.

Κεφάλαιο 6ο: Αποτελέσματα

6.1 Εισαγωγή

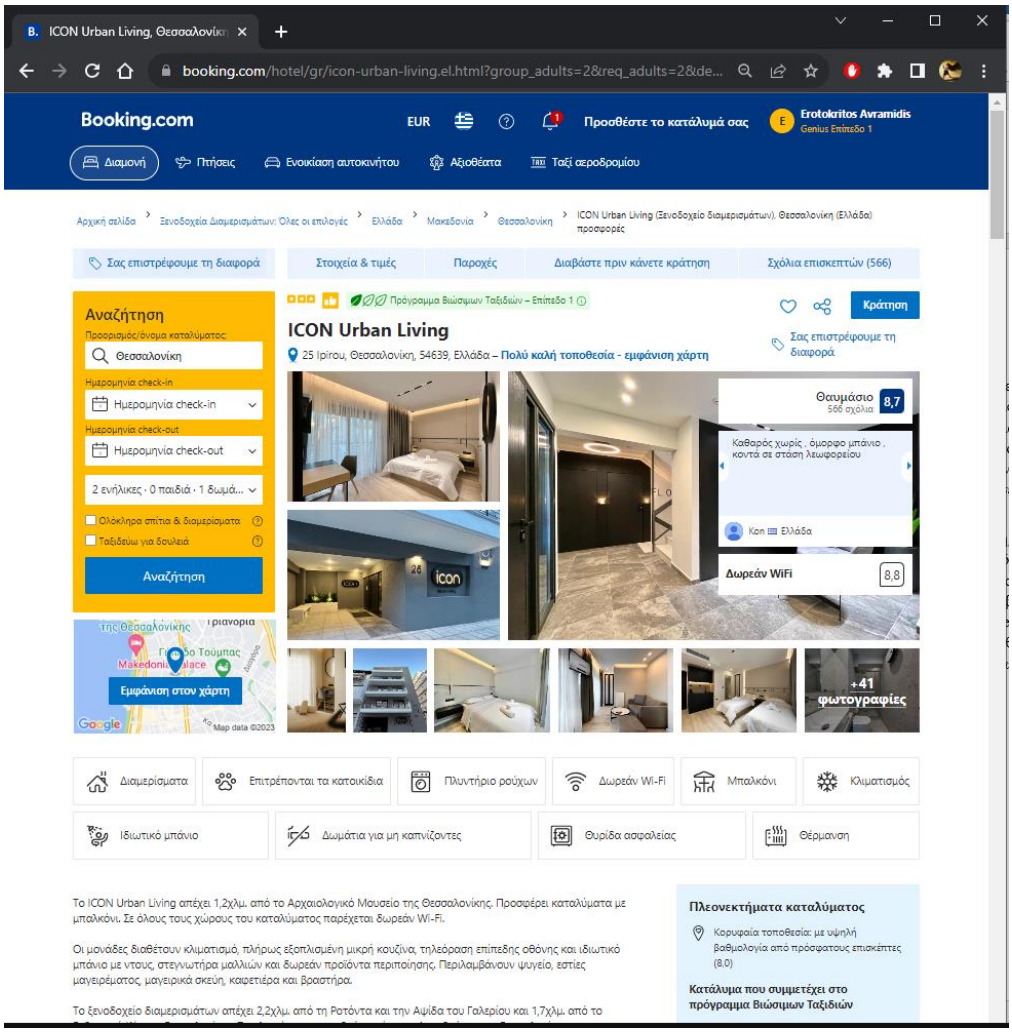
Καθώς πλησιάζουμε στο τελευταίο κεφάλαιο αυτής της εργασίας, είναι σημαντικό να σταματήσουμε και να αναλογιστούμε το ταξίδι που έχουμε αναλάβει για να φτάσουμε στα αποτελέσματα που περικλείουν την εξαμηνιαία προσπάθειά. Αυτό το κεφάλαιο στοχεύει να συγκεντρώσει όλα τα στοιχεία—αρχές DevOps, μεθοδολογίες κοντέινερ και τεχνικές κλιμάκωσης— για να τονίσει τα αποτελέσματα και τα αποτελέσματα του έργου. Θα εξετάσουμε την αποτελεσματικότητα των εργαλείων και των τεχνολογιών που χρησιμοποιήθηκαν και θα εμβαθύνουμε στις προκλήσεις που αντιμετωπίσαμε και ξεπεράσαμε.

Θα αναλύσουμε πώς ο σχεδιασμός μας, που υλοποιήθηκε μέσω πολλών φωνητικών κλήσεων και σχολαστικών κύκλων sprint, μεταφράστηκε σε ένα λειτουργικό, επεκτάσιμο σύστημα. Θα εξηγήσουμε επίσης πώς οι επιλογές που κάναμε στις φάσεις της αρχιτεκτονικής και της ανάπτυξης επηρέασαν τα τελικά αποτελέσματα, ειδικά όσον αφορά την επεκτασιμότητα και την αξιοπιστία του συστήματος.

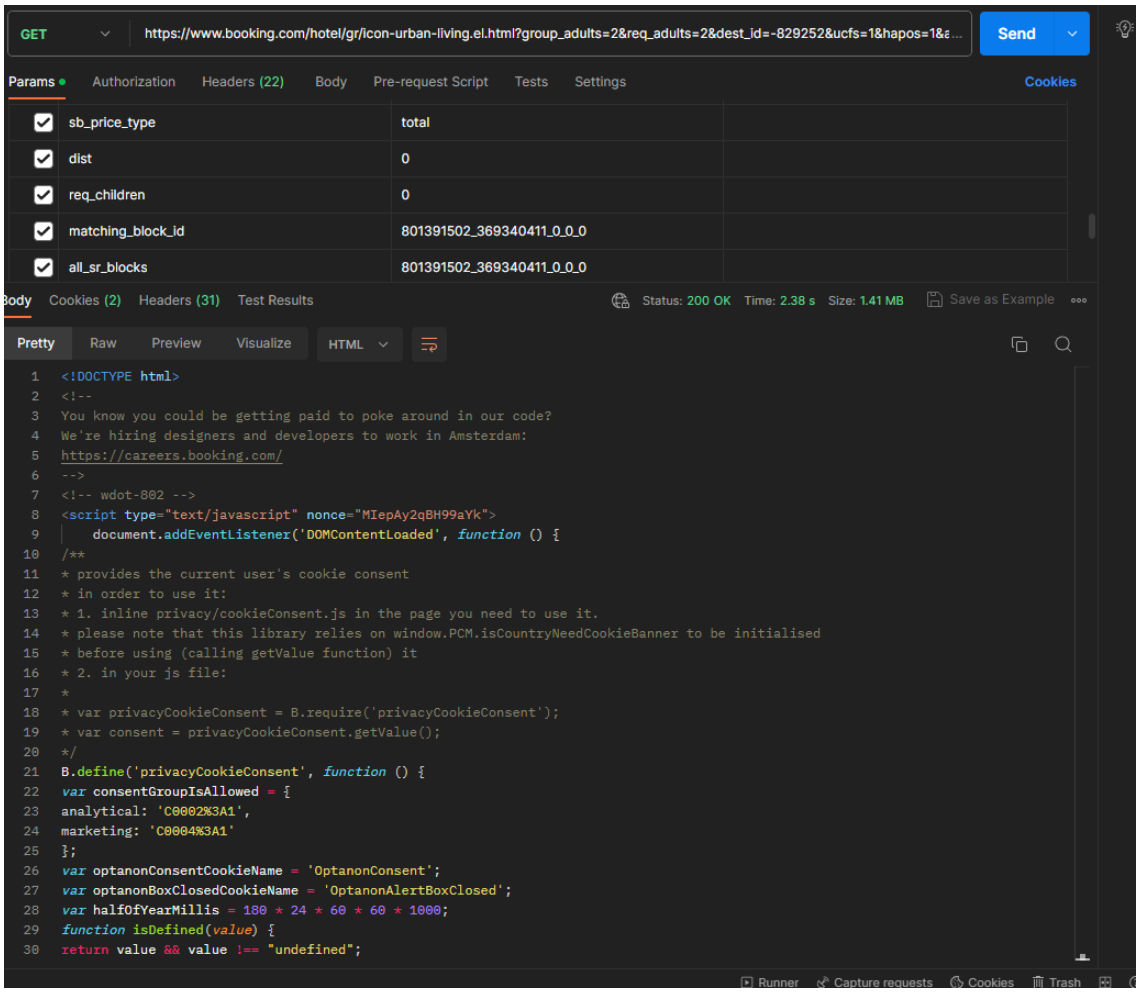
6.2 Web Scraper και Scheduler

Πρώτον, ένα από τα επιτεύγματα ήταν η επιτυχής δημιουργία και λειτουργία ενός web scraper. Αυτός ο scraper κατάφερε να ολοκληρώσει έναν εντυπωσιακό μέσο όρο πέντε εργασιών ανά δευτερόλεπτο. Αυτό το πετύχαμε ενσωματώνοντας μια μέθοδο που χρησιμοποίησε αιτήματα μέσω proxy server επιτρέποντας μας να χρησιμοποιούμε διαφορετική IP διεύθυνση για κάθε ξενοδοχείο. Η χρήση proxy server είχε επίσης το πρόσθετο πλεονέκτημα της ελαχιστοποίησης της πιθανότητας εντοπισμού των δραστηριοτήτων του scraper και στη συνέχεια του αποκλεισμού από τις ιστοσελίδες από τους οποίους αντλούσαμε δεδομένα.

Η αρχική του μορφή ήταν πολύ πιο σύνθεση και οδήγησε σε διαφορά προβλήματα. Στην αρχή, για να αντλήσει τα δεδομένα του κάθε ξενοδοχείου έπρεπε να ανοίγει ένας browser όπου θα επισκεπτόταν τη σελίδα και θα τραβούσε τα δεδομένα. Αυτό οδήγησε σε πολύ μεγάλους χρόνους scraping (10-15 sec) και πολύ μεγάλο χώρο στη μνήμη RAM (1 browser/1 task). Επιπλέον, σε περιβάλλον docker container ο browser δεν έτρεχε σωστά. Η λύση ήταν να αποφύγουμε εντελώς τον browser και να πάμε σε μια πιο low level προσέγγιση, δηλαδή να στέλνουμε /GET requests μέσω της βιβλιοθήκης axios της Node.js και να παρσάρουμε το HTML που λαμβάνουμε το οποίο περιλαμβάνει και τα δεδομένα που μας ενδιαφέρουν.



Σχήμα 6.2: Scraper browser navigation

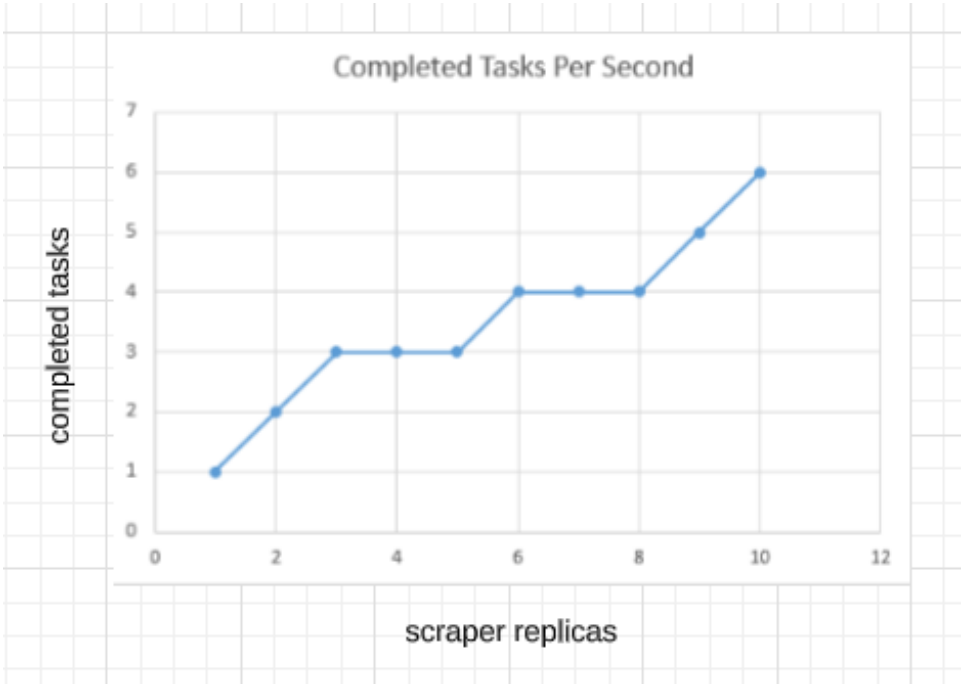


Σχήμα 6.2: Scraper Request HTML content

Η δουλεία του scheduler είναι αρκετά απλή. Τραβάει τα αιτήματα των χρηστών ανά τακτικά χρονικά διαστήματα από την βάση δεδομένων και να τα προωθεί στην ουρά που πυροδοτεί τον scraper.

6.3 Scalability and Queues

Δεύτερον, η προσοχή μας στράφηκε προς την επίτευξη scalability του συστήματος. Οι web scraper μας σχεδιάστηκαν σχολαστικά ώστε να κάνουν scale στις διακυμάνσεις στη χρήση τόσο της CPU όσο και της RAM. Αυτός ο μηχανισμός δυναμικής επεκτασιμότητας ήταν βασικός για τη βελτιστοποίηση της χρήσης των πόρων του συστήματος, διασφαλίζοντας παράλληλα ότι ο scraper μπορεί να χειριστεί ποικίλα φορτία χωρίς να απαιτείται χειροκίνητη παρέμβαση.



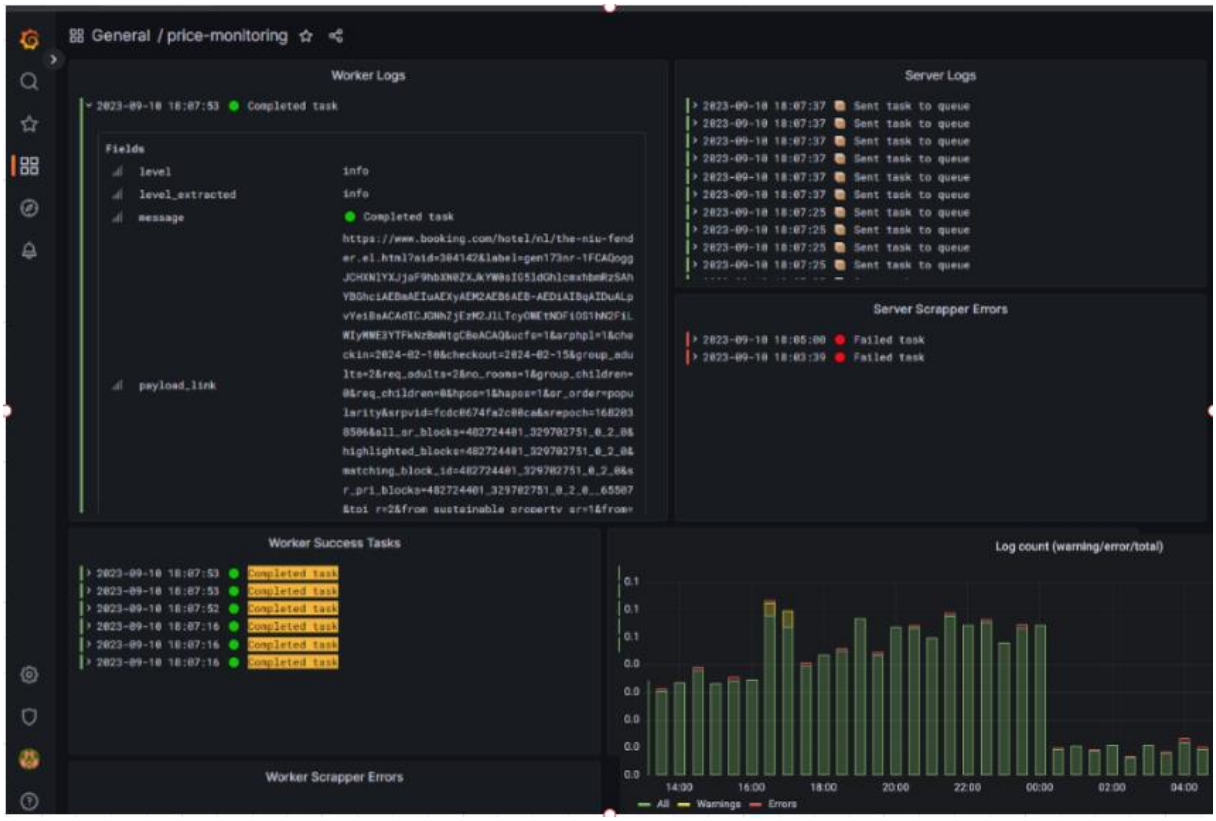
Σχήμα 6.3: Task completion graph

Η αποδοτικότητα σε αυτό το σημείο είναι αρκετά ικανοποιητική φτάνοντας στα 6 επιτυχημένα requests ανά δευτερόλεπτο στα 10 scraper replicas. Αξίζει να αναφερθεί ότι σε κάποιες περιπτώσεις προκύπτουν προβλήματα όπως το proxy service δεν καταφέρνει να στείλει το αίτημα, το page format του συγκεκριμένου δωματίου να μην ταιριάζει στην λογική του scraper ή κάποιο άλλο απρόβλεπτο γεγονός.

Σε αυτή τη περίπτωση η χρήση ουρών με RabbitMQ σώζει την κατάσταση. Τα tasks τα οποία δε καταφέρνουν να ολοκληρωθούν, επαναπροθούντε στην ουρά για να γίνουν retry. Αυτή η διαδικασία για κάθε task μπορεί να γίνει μέχρι 3 φορές και όταν ξεπεραστεί αυτό το όριο το task προωθείτε σε ένα dead letter queue όπου ο σκοπός του είναι να κρατά όλα τα tasks που δεν κατάφεραν ποτέ να ολοκληρωθούν για περαιτέρω ανάλυση από τους διαχειριστές.

6.4 Grafana Monitoring

Το Grafana είναι το service όπου οι διαχειριστές έχουν πλήρη εικόνα των logs του scraper και του scheduler. Αυτό του βοηθάει να παρατηρούν την συμπεριφορά της εφαρμογής και να κάνουν debug. Επίσης, το grafana διαθέτει διάφορα χρήσιμα plugins. Ένα από αυτά είναι το notification plugin που μπορούμε να θέσουμε συγκεκριμένα κριτήρια όπου όταν πληρούνται να λαμβάνουμε ένα ενημερωτικό email, μήνυμα στο κίνητο η slack. Για παράδειγμα, εάν βρεις στα Logs του scraper περισσότερα από 1 errors τα τελευταία 5 λεπτά κάνε trigger το notification. Αυτό σημαίνει ότι δεν χρειάζεται να υπάρχει ανθρώπινη επίβλεψη πάνω στον scraper παρά μόνο όταν υπάρχει ανάγκη.



Σχήμα 6.4: Grafana price-monitoring

6.5 Docker

Όπως έχει προαναφερθεί τα κύρια κομμάτια της εφαρμογής είναι ο scheduler, scraper, RabbitMQ, loki, NGINX load balancer και Grafana. Όλα αυτά τα services τρέχουν σε ένα κοινό compose stack μέσω docker images και containers. Επιπλέον, χρησιμοποιώντας τα GitHub Actions κάνουμε build, test και deploy τον scraper και τον scheduler ενημερώνοντας το docker image σε κάθε git push.

Σχήμα 6.5: docker-compose stack

6.6 Επίλογος

Από το αποτέλεσμα του έργου, είμαι ιδιαίτερα ικανοποιημένος που βλέπω πόσο απρόσκοπτα ενσωματώθηκαν οι επιλεγμένες τεχνολογίες για να σχηματίσουν ένα συνεκτικό, αποτελεσματικό σύστημα. Από τη χρήση του Node.js για scraping, RabbitMQ για διαχείριση ουρών, Docker για containers, μέχρι Grafana και Loki για παρακολούθηση και οπτικοποίηση δεδομένων σε πραγματικό χρόνο, κάθε εργαλείο όχι μόνο ανταποκρίθηκε αλλά ξεπέρασε τις προσδοκίες. Οι συνδυασμένες δυνατότητές τους μου επέτρεψαν να αντιμετωπίσω τις πραγματικές προκλήσεις, όπως η αποφυγή μπλοκαρίσματος του παρόχου πηγής, διατηρώντας παράλληλα την επεκτασιμότητα και την αξιοπιστία του συστήματός.

Επιπλέον, τα συστήματα ειδοποίησης και οι πίνακες εργαλείων έχουν αποδειχθεί ανεκτίμητα τόσο για την παρακολούθηση της απόδοσης της πλατφόρμας όσο και για τη λήψη αποφάσεων βάσει δεδομένων. Το αποτέλεσμα είναι μια εξελιγμένη, ισχυρή πλατφόρμα που ολοκληρώνει περίπου πέντε εργασίες ανά δευτερόλεπτο, κλιμακώνεται σύμφωνα με τους πόρους του συστήματος και παρέχει αρχεία καταγραφής και μετρήσεις σε πραγματικό χρόνο. Δεν είναι απλώς μια εκπλήρωση των στόχων του έργου. είναι μια απόδειξη της δύναμης της επιλογής των σωστών εργαλείων για τη σωστή δουλειά.

Για μένα, το αποτέλεσμα ήταν βαθιά ικανοποιητικό. Οι τεχνολογίες που χρησιμοποιήθηκαν ήταν ιδανικές για τους στόχους που θέσαμε και το τελικό προϊόν έχει απήχηση τόσο στην ποιότητα όσο και στην αποτελεσματικότητα.

Κεφάλαιο 7ο: Συμπέρασμα

Αυτή η εργασία έγινε για να δείξει πόσο σημαντική είναι η παρακολούθηση των τιμών ξενοδοχείων/δωματίων για τις επιχειρήσεις. Εξήγησε πώς η πληροφορική μπορεί να βοηθήσει τις επιχειρήσεις και τα οφέλη που φέρνει. Μίλησε επίσης για τις διαφορετικές γλώσσες προγραμματισμού που χρησιμοποιούνται για τη δημιουργία μιας εφαρμογής. Η εφαρμογή παρακολούθησης τιμών μπορεί να βοηθήσει τους ιδιοκτήτες ξενοδοχείων να κερδίσουν περισσότερα χρήματα και να εξοικονομήσουν χρόνο αλλά και τους χρήστες για να εντοπίζουν τις καλύτερες ευκαιρίες. Εάν το λογισμικό επεκτείνει τις πηγές των δεδομένων που λαμβάνει, μπορεί να αποφέρει ακόμη περισσότερα χρήματα για την επιχείρηση. Οι ιδιοκτήτες ξενοδοχείων θα πρέπει να χρησιμοποιούν την εφαρμογή τακτικά και να είναι προσεκτικοί με τα δεδομένα που αναλύουν. Οι πρακτικές DevOps, agile μεθοδολογίες και microservices βοηθούν τις ομάδες ανάπτυξης να εξελίσσουν, αναπτύξουν και παραδώσουν ένα δυνατό software που θα εξυπηρετεί τους χρηστές της του όσο αλλάζουν οι ανάγκες τους.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία

[1] https://www.besanttechnologies.com/what-is-expressjs

[2] https://www.infoworld.com/article/3210589/what-is-nodejs-javascript-runtime-explained.html

[3] https://www.netrivals.com/resources/guides/price-monitoring-software-reasons-use/

[4] https://relevant.software/blog/why-and-when-to-use-node-js/

[5]https://www.cloudamqp.com/blog/rabbitmq-use-cases-explaining-message-queues-and-when-to-use-them.html

[6] https://climbtheladder.com/10-rabbitmq-best-practices/

[7] https://www.atatus.com/blog/a-beginners-guide-for-grafana-loki/

[8] https://www.clickittech.com/devops/docker-use-cases/

[9] https://www.geeksforgeeks.org/architecture-of-docker/

[10] https://aws.amazon.com/what-is/containerization/

[11] https://www.pricerest.com/what-is-price-monitoring/

[12] https://dealavo.com/en/additional-information/what-is-a-price-monitoring-tool/

[13] https://www.premise.com/price-monitoring/

[14] https://www.competitormonitor.com/blog/price-monitoring-strategy/

[15]https://www.centricsoftware.com/blog/what-is-competitor-price-monitoring-how-can-price-benchmarking-benefit-you/

[16] https://www.toptal.com/insights/innovation/what-is-devops

[17] https://www.tandfonline.com/doi/full/10.1080/23311916.2022.2083474

[18] https://medium.com/design-microservices-architecture-with-patterns/microservices-architecture-2bec9da7d42a

[19] https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith

[20] https://reactjs.org/

[21] https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html

[22] https://www.cloudbees.com/blog/autoscaling-purpose-strategies

[23] https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-019-0146-7

[24] https://www.parsehub.com/blog/what-is-web-scraping/

[25] https://www.fortinet.com/resources/cyberglossary/proxy-server

[26] https://www.zyte.com/learn/what-is-web-scraping/

[27] https://perfectafternoon.com/2021/ethical-issues-when-scraping-the-web/