ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Η Αρχιτεκτονική των Microservices στην Ανάπτυξη Λογισμικού

Μεταπτυχιακή Διπλωματική Εργασία

Ηλίας Ελευθερίου

Επιβλέπων: Μιχαήλ Βασιλακόπουλος

Φεβρουάριος 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

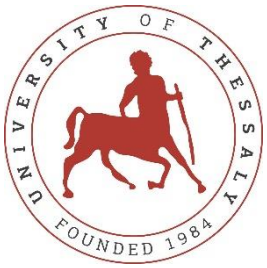
Η Αρχιτεκτονική των Microservices στην Ανάπτυξη Λογισμικού

Μεταπτυχιακή Διπλωματική Εργασία

Ηλίας Ελευθερίου

Επιβλέπων: Μιχαήλ Βασιλακόπουλος

Φεβρουάριος 2024



UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Microservices Architecture in Software Development

MSc Thesis

Ilias Eleftheriou

Supervisor: Michail Vasilakopoulos

February 2024

Εγκρίνεται από την Επιτροπή Εξέτασης:

Επιβλέπων/πουσα	**Μιχαήλ Βασιλακόπουλος** Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας
Μέλος	**Χρήστος Αντωνόπουλος** Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας
Μέλος	**Σπύρος Λάλης** Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα μεταπτυχιακή διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελούν αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλουν οποιασδήποτε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχουν έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Δηλώνω επίσης ότι τα αποτελέσματα της εργασίας δεν έχουν χρησιμοποιηθεί για την απόκτηση άλλου πτυχίου. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο Δηλών

Ηλίας Ελευθερίου

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this MSc thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

The Declarant

Ilias Eleftheriou

Ευχαριστίες ή Σχόλια

Αφιερώνω τις ευχαριστίες μου σε όλους εκείνους που στήριξαν και εμπνεύστηκαν αυτή την προσπάθεια, καθιστώντας δυνατή την ολοκλήρωση της Μεταπτυχιακής Διπλωματικής Εργασίας με θέμα: "Η Αρχιτεκτονική των Microservices στην Ανάπτυξη Λογισμικού".

Πρώτα απ' όλα, θα ήθελα να εκφράσω τη βαθιά μου εκτίμηση προς τον επιβλέποντα καθηγητή μου, για την απαραίτητη καθοδήγηση, υποστήριξη και επιστημονική κριτική καθ' όλη τη διάρκεια της έρευνας.

Επίσης, είμαι ευγνώμων στην οικογένειά μου και τους φίλους μου για την ανεκτίμητη υποστήριξη, την αγάπη και την πίστη τους στις δυνατότητές μου. Η συνεχής τους ενθάρρυνση ήταν το φως που με οδήγησε σε στιγμές αμφιβολίας και δυσκολίας.

Ειδική μνεία αξίζει στην τεχνολογική κοινότητα για τη διάθεση πολύτιμων πόρων και έρευνας που συνέβαλαν στη βάθυνση της κατανόησης μου πάνω στο θέμα των Microservices. Οι διαλέξεις, τα άρθρα και τα φόρουμ συζητήσεων ήταν ανεκτίμητες πηγές γνώσης.

Τέλος, θα ήθελα να αποδώσω τιμή σε όλους εκείνους που άμεσα ή έμμεσα συνέβαλαν στην ολοκλήρωση αυτής της διπλωματικής εργασίας. Κάθε συμβουλή, κριτική και προτροπή ήταν ένα σημαντικό διδάγμα για εμένα.

Με ειλικρινή ευγνωμοσύνη,

Ηλίας Ελευθερίου

Μεταπτυχιακή Διπλωματική Εργασία

Η Αρχιτεκτονική των Microservices στην Ανάπτυξη Λογισμικού

Ηλίας Ελευθερίου

Περίληψη

Εισαγωγή: Η αρχιτεκτονική των microservices αναπτύσσεται ως μια σύγχρονη και εξελιγμένη προσέγγιση στον σχεδιασμό και την ανάπτυξη λογισμικού, ανταποκρινόμενη στις απαιτήσεις ενός εποχικού και δυναμικού περιβάλλοντος πληροφορικής.

Σκοπός: Σκοπός της παρούσας εργασίας είναι να αποσαφηνίσει τα πλεονεκτήματα της αρχιτεκτονικής των microservices έναντι τις μονολιθικής αρχιτεκτονικής και να προτείνει εφαρμογές χρήσης της.

Μεθοδολογία: Στο πρώτο μέρος η εργασία ακολουθεί τη μέθοδο της βιβλιογραφικής ανασκόπησης, η οποία επιτρέπει στον ερευνητή να έχει πρόσβαση σε πληθώρα άρθρων και μελετών, με στόχο την εξαγωγή αντικειμενικών συμπερασμάτων.

Αποτελέσματα: Τα αποτελέσματα της εργασίας αποδεικνύουν πως η αρχιτεκτονική των microservices είναι πιο ευέλικτη σε σχέση με τη μονολιθική αρχιτεκτονική, για αυτό και μπορεί να διαχειριστεί πιο εύκολα.

Συμπεράσματα: Η αρχιτεκτονική των microservices μπορεί να εφαρμοστεί σε πλήθος συστημάτων, καθώς η διαχείρισή της είναι πιο εύκολη για τον διαχειριστή, ωστόσο λόγω της συνεχούς εξέλιξης της τεχνολογίας, θα πρέπει να λαμβάνονται υπόψη οι νέες τεχνολογίες που μπορούν να εμπλουτίσουν τις ήδη υπάρχουσες.

Λέξεις-κλειδιά:

Μονολιθική Αρχιτεκτονική, Microservices, υπηρεσιοστροφής αρχιτεκτονική, Camunda framework

MSc Thesis

Microservices Architecture in Software Development

Ilias Eleftheriou

Abstract

Introduction: Microservices architecture is developing as a modern and sophisticated approach to software design and development, responding to the demands of a seasonal and dynamic IT environment.

Purpose: The purpose of this paper is to clarify the advantages of the microservices architecture over the monolithic architecture and to propose applications for its use.

Methodology: In the first part, the work follows the bibliographic review method, which allows the researcher to have access to a large number of articles and studies, with the aim of drawing objective conclusions.

Results: The results of the work prove that the microservices architecture is more flexible than the monolithic architecture, which is why it can be managed more easily.

Conclusions: The architecture of microservices can be applied to a multitude of systems, as its management is easier for the administrator, however, due to the continuous development of technology, new technologies that can enrich the existing ones should be taken into account.

Keywords:

Monolithic Architecture, Microservices, service-oriented architecture, Camunda framework

Πίνακας περιεχομένων

Κατάλογος εικόνων

Κατάλογος σχημάτων

Συντομογραφίες

BPM	Business Process Management
BPMN	Business Process Model and Notation
CORS	Cross-Origin Resource Sharing
CRUD	Create Read Update Delete
BPF	Band Pass Filter
DI	Dependency Inj
FaaS	Function as a Service
MSA	Microservices Architecture
POJOs	Plain Old Java Objects
SOA	Service Oriented Architecture

Κεφάλαιο 1 Εισαγωγή

Τα τελευταία χρόνια ο κλάδος της πληροφορικής έχει χαρακτηριστεί από την ταχεία ανάπτυξη της κατανεμημένης επεξεργασίας δεδομένων και των αιτημάτων των χρηστών και, ως εκ τούτου, από την ευρεία χρήση της αρχιτεκτονικής μικροϋπηρεσιών [1]. Παρόλο που η αρχιτεκτονική των microservices έχει μεγάλο ερευνητικό ενδιαφέρον, δεν υπάρχει ακόμα σαφής διαμορφωμένη άποψη για τις υπάρχουσες ερευνητικές λύσεις για την αρχιτεκτονική με μικροϋπηρεσίες [2].

1.1 Αντικείμενο της διπλωματικής

Η αρχιτεκτονική των microservices αναπτύσσεται ως μια σύγχρονη και εξελιγμένη προσέγγιση στον σχεδιασμό και την ανάπτυξη λογισμικού, ανταποκρινόμενη στις απαιτήσεις ενός εποχικού και δυναμικού περιβάλλοντος πληροφορικής. Σε αντίθεση με τις παραδοσιακές μονολιθικές αρχιτεκτονικές, όπου το λογισμικό είναι συγκεντρωμένο σε ένα μεγάλο και συνήθως δύσκαμπτο μονόλιθο, οι microservices υποστηρίζουν τον διακριτικό διαχωρισμό των λειτουργιών σε μικρότερες, ανεξάρτητες υπηρεσίες [3] . Κάθε microservice αναλαμβάνει ένα συγκεκριμένο κομμάτι λειτουργικότητας και επικοινωνεί με άλλες υπηρεσίες μέσω προκαθορισμένων διεπαφών. Αυτή η δομή επιτρέπει την ευελιξία, την εύκολη επεκτασιμότητα και την ανεξαρτησία ανάπτυξης, επιτρέποντας στις ομάδες ανάπτυξης να εργάζονται ανεξάρτητα μεταξύ τους.

Η Microservices Architecture (MSA) είναι ένα αρχιτεκτονικό στυλ εγγενές στο cloud, το οποίο είναι εμπνευσμένο από την Αρχιτεκτονική με προσανατολισμό στις υπηρεσίες. Συνήθως, οι μικροϋπηρεσίες οργανώνονται ως μια σουίτα μικρών λεπτομερών υπηρεσιών που μπορούν να υλοποιηθούν (αναπτύξουν, δοκιμαστούν και αναπτυχθούν) σε διαφορετικές πλατφόρμες μέσω πολλαπλών τεχνολογικών στοίβων [4].

Η αρχιτεκτονική των Microservices είναι ιδιαίτερα δημοφιλής σε διαφορετικές πτυχές του βιομηχανικού κλάδου, διότι διαθέτει διάφορα θετικά χαρακτηριστικά, όπως η διαθεσιμότητα, η ευελιξία, η επεκτασιμότητα, η χαλαρή σύζευξη και η υψηλή ταχύτητα

[5].

Αντικείμενο, λοιπόν, της παρούσας εργασίας είναι η ανάλυση των χαρακτηριστικών και των προτύπων της αρχιτεκτονικής των microservices μέσα από τον εντοπισμό των διαφορών τους με τη μονολιθική αρχιτεκτονική. Εξετάζονται, δηλαδή, τα μειονεκτήματα και τα πλεονεκτήματα των δύο αρχιτεκτονικών, ενώ παράλληλα παρουσιάζονται τα εργαλεία και οι τεχνολογίες που βασίζονται στην αρχιτεκτονική των microservices. Η πρωτοτυπία αυτής της εργασίας έγκειται στο γεγονός πως μελετά τον ρόλο και τη λειτουργία της αρχιτεκτονικής των microservices μέσα από μια συγκριτολογική προσέγγιση με τη μονολιθική αρχιτεκτονική. Με αυτόν τον τρόπο γίνεται κατανοητή η διαδικασία της μετάβασης από τη μία αρχιτεκτονική στην άλλη και επιπλέον αναλύονται διεξοδικά οι δυνατότητες των τεχνολογικών εργαλείων που σχετίζονται με αυτές.

1.1.1 Συνεισφορά

Η συνεισφορά της παρούσας εργασίας στην υπάρχουσα βιβλιογραφία έγκειται στα παρακάτω:

1. Μελετήθηκαν συγκριτικά τα πλεονεκτήματα και τα μειονεκτήματα της μονολιθικής αρχιτεκτονικής και της αρχιτεκτονικής των Microservices.
2. Αξιολογήθηκαν οι δυνατότητες των δύο αρχιτεκτονικών.
3. Εξετάστηκαν συγκεκριμένες τεχνολογίες Microservices.
4. Υλοποιήθηκαν παραδείγματα Microservices.
5. Πραγματοποιήθηκε σύγκριση των τεχνολογιών.

1.2Οργάνωση του τόμου

Μετά το πρώτο κεφάλαιο της εισαγωγής, στο δεύτερο κεφάλαιο αναλύεται η μονολιθική αρχιτεκτονική, τα χαρακτηριστικά και τα πλεονεκτήματά της. Αναλύεται επίσης και η υπηρεσιοστροφής αρχιτεκτονική ως πιο σύγχρονη και ευέλικτη μορφή της μονολιθικής αρχιτεκτονικής.

Στο τρίτο κεφάλαιο παρουσιάζεται η αρχιτεκτονική των microservices μέσα από την οποία διερευνώνται μεταξύ άλλων οι πρακτικές και οι προκλήσεις που συνεπάγεται. Παράλληλα, εξετάζονται τα σχεδιαστικά πρότυπα.

Στο τέταρτο κεφάλαιο συντελείται μια αναδρομή στη διαδικασία μετάβασης από τη μονολιθική αρχιτεκτονική στην αρχιτεκτονική των microservices με παράλληλη σύγκριση των δύο αρχιτεκτονικών και διερεύνηση των διαφορών τους.

Στο πέμπτο κεφάλαιο αναλύονται και συγκρίνονται τα τεχνολογικά εργαλεία, προκειμένου να διαφανούν οι δυνατότητες και οι αδυναμίες τους, καθώς και η τεχνολογική εξέλιξή τους.

Στο έκτο κεφάλαιο παρουσιάζονται οι τεχνολογίες των μικροϋπηρεσιών, ενώ παράλληλα παρατίθενται τα παραδείγματα μικροϋπηρεσιών που πραγματοποιήθηκαν στο πλαίσιο της παρούσας εργασίας.

Στο έβδομο κεφάλαιο παρουσιάζονται συγκεντρωτικά τα πιο κύρια συμπεράσματα της παρούσας εργασίας, ενώ επίσης διατυπώνονται ορισμένες μελλοντικές προτάσεις, που θα συμβάλλουν στην περαιτέρω διερεύνηση του θέματος.

Τέλος, η μέθοδος που αξιοποιεί η παρούσα εργασία για τη διάρθρωση του θεωρητικού της μέρους είναι αυτή της βιβλιογραφικής ανασκόπησης. Η συγκεκριμένη μέθοδος δίνει τη δυνατότητα στον ερευνητή να έχει πρόσβαση σε μια πληθώρα τόσο εγχώριων όσο και διεθνών άρθρων, ώστε να συλλέξει με αντικειμενικό τρόπο τα δεδομένα εκείνα που ανταποκρίνονται στον σκοπό της μελέτης του [6]. Στη συγκριμένη περίπτωση αξιοποιήθηκαν επιστημονικά άρθρα, τα οποία αναλύουν την αρχιτεκτονική των microservices και τη μονολιθική αρχιτεκτονική καθώς και τις τεχνολογίες που συνδέονται με αυτές.

Κεφάλαιο 2 Μονολιθική Αρχιτεκτονική

2.1 Πλεονεκτήματα και μειονεκτήματα της μονολιθικής αρχιτεκτονικής

Η μονολιθική αρχιτεκτονική συνιστά την «παραδοσιακή» μέθοδο ανάπτυξης λογισμικού. Η ονομασία της (μονόλιθος) οφείλεται στο γεγονός πως η ο σχεδιασμός, η ανάπτυξη και η εφαρμογή ενός λογισμικού βασίζεται σε ένα ενιαίο μοντέλο. Πιο συγκεκριμένα, στη μονολιθική αρχιτεκτονική, υπάρχει στενή σύνδεση και αλληλεξάρτηση μεταξύ των στοιχείων της. Αυτό βέβαια έχει ως συνέπεια η διάκριση αυτών των στοιχείων σε επιμέρους κομμάτια να είναι ιδιαίτερα δύσκολη. Επίσης, για την εκτέλεση ενός κώδικα χρειάζονται όλα τα στοιχεία της εφαρμογής, καθώς η απουσία κάποιου ή η αλλαγή του ενδέχεται να προκαλέσουν μια σειρά από συνεχόμενες μεταβολές λόγω και της αλληλεξάρτησής τους. Παράλληλα, η ενημέρωση κάποιας λειτουργίας στη μονολιθική αρχιτεκτονική ή η προσθήκη νέων στοιχείων μπορούν να λειτουργήσουν ως εμπόδιο τόσο στην επέκτασή της όσο και στη διαδικασία συντήρησής της [7].

Ειδικότερα, στις εφαρμογές της μονολιθικής αρχιτεκτονικής υπάρχει μια ενιαία βάση κώδικα, πάνω στην οποία αναπτύσσεται η λειτουργικότητα και μπορεί να χρησιμοποιηθεί από τους προγραμματιστές. Αυτή η ενιαία βάση όμως μπορεί να οδηγήσει στην κατάρρευση του συστήματος συνολικά, διότι σε περίπτωση που συμβούν σφάλματα σε μια υπηρεσία, οι συνέπειες αυτών θα μεταφερθούν αλυσιδωτά σε ολόκληρο το σύστημα ακριβώς λόγω της σύνδεσης των υπηρεσιών. Δημιουργείται έτσι ένα συγκεκριμένο σημείο, το οποίο μπορεί να είναι υπεύθυνο για την αποτυχία μιας εφαρμογής και, όπως είναι φυσικό, αυτό έχει ως αποτέλεσμα να μην εξασφαλίζεται η διαθεσιμότητα του ίδιου του συστήματος, στοιχείο που είναι ιδιαίτερα σημαντικό και απαραίτητο στις περισσότερες εφαρμογές [8].

Επίσης, το σύστημα στη μονολιθική αρχιτεκτονική δεν έχει πολλές επιλογές για αυτόματη κλιμάκωση. Η κλιμάκωση αναφέρεται στην ποιοτική ή ποσοτική αναβάθμιση μιας εφαρμογής, η οποία ανταποκρίνεται και στις υπολογιστικές ανάγκες μιας υπηρεσίας [9]. Στην περίπτωση της μονολιθικής αρχιτεκτονικής εφόσον η λειτουργία συνδέεται με μεγάλη λειτουργικότητα, λόγω της ενιαίας δομής της αρχιτεκτονικής, συνεπάγεται ότι θα

έχει και μεγάλο κόστος τόσο εκτέλεσης όσο και συντήρησης. Σε αντίθετη περίπτωση, δηλαδή αν η υπηρεσία δεν ανήκε στο πλαίσιο της ενιαίας δομής της αρχιτεκτονικής, αλλά ήταν μεμονωμένη, τότε είναι προφανές πως και το κόστος θα ήταν μικρότερο, γιατί η κλιμάκωση θα μπορούσε να γίνει περιορισμένα.

Ωστόσο, παρά τις δυσκολίες που ενδέχεται να προκύψουν στο πλαίσιο της μονολιθικής αρχιτεκτονικής, παρατηρούνται διάφορα πλεονεκτήματα σε σύγκριση με άλλων ειδών αρχιτεκτονικές, όπως η ευκολία ως προς την εύρεση σφαλμάτων, την πραγματοποίηση δοκιμών και την ολοκλήρωση του λογισμικού χάρη στην ενιαία δομή τους. Μέσω της μονολιθικής αρχιτεκτονικής, η διαχείριση της εφαρμογής καθίσταται περισσότερο εύκολη, δεδομένου ότι τα διαγνωστικά δεδομένα συναρμόζονται πιο απλά, όπως επίσης δρομολογούνται με πιο απλό τρόπο τα αιτήματα και διαχειρίζονται ευκολότερα τα προσωρινά δεδομένα [10].

Γενικά, η μονολιθική αρχιτεκτονική χρησιμοποιεί μια ενιαία τεχνολογία ανάπτυξης, περιορίζοντας τη διαθεσιμότητα ενός κατάλληλου εργαλείου για κάθε εργασία που χρειάζεται να εκτελέσει το σύστημα. Σύμφωνα με τον [11] ,σε ένα μόνο λογικό εκτελέσιμο, οποιαδήποτε αλλαγή ολοκληρώνεται σε ένα τμήμα αυτού του είδους συστήματος περιλαμβάνει την κατασκευή και την ανάπτυξη μιας νέας έκδοσης ολόκληρου του συστήματος. Οι περισσότερες εμπλεκόμενες πτυχές ή επίπεδα παρουσίασης, επεξεργασίας και αποθήκευσης είναι ένα ενιαίο στοιχείο λογισμικού που απαιτείται για την εκτέλεση σε έναν διακομιστή. Ανάμεσα στα πιο σημαντικά πλεονεκτήματα, είναι τα σταθερά συστήματα, το σύστημα πλήρους διαχείρισης, που χρησιμοποιείται από μεγάλες εταιρείες στον κόσμο. Από την άλλη πλευρά, είναι άκαμπτα συστήματα και είναι δύσκολο να προσαρμοστούν στις νέες ανάγκες. Η αύξηση της ικανότητας επεξεργασίας τους γίνεται μέσω της αλλαγής του τρέχοντος διακομιστή για έναν μεγαλύτερο. Όπως γίνεται αντιληπτό, το κόστος απόκτησης, ανανέωσης και υποστήριξης σε τέτοιες περιπτώσεις είναι υψηλό [10].

2.2 Μοντέλο πελάτη-διακομιστή

Στον κόσμο των υπολογιστών σήμερα, το σύστημα πελάτη-διακομιστή έχει γίνει τόσο δημοφιλές επειδή χρησιμοποιείται σχεδόν καθημερινά για διαφορετικές εφαρμογές. Μερικά από τα τυποποιημένα πρωτόκολλα που χρησιμοποιούν οι πελάτες και οι

διακομιστές για να επικοινωνούν μεταξύ τους περιλαμβάνουν: Πρωτόκολλο μεταφοράς αρχείων (FTP), Πρωτόκολλο μεταφοράς απλού ταχυδρομείου (SMTP) και πρωτόκολλο μεταφοράς υπερκειμένου (HTTP). Έτσι, το σύστημα πελάτη-διακομιστή μπορεί να οριστεί ως μια αρχιτεκτονική λογισμικού που αποτελείται τόσο από τον πελάτη όσο και από τον διακομιστή, όπου οι πελάτες στέλνουν πάντα αιτήματα ενώ ο διακομιστής ανταποκρίνεται στα αιτήματα που αποστέλλονται. Ο πελάτης-διακομιστής παρέχει μια επικοινωνία μεταξύ διεργασιών, επειδή περιλαμβάνει την ανταλλαγή δεδομένων τόσο από τον πελάτη όσο και από τον διακομιστή, όπου καθένας από αυτούς εκτελεί διαφορετικές λειτουργίες [12].

Το μοντέλο πελάτη-διακομιστή αποτελεί κεντρική αρχιτεκτονική στο πλαίσιο των υπολογιστικών δικτύων και του σχεδιασμού τους. Συμβάλλει στην επικοινωνία μεταξύ των υπηρεσιών και των εφαρμογών μέσω του δικτύου και έτσι διευκολύνει την επεξεργασία των δεδομένων, την αποδοτική χρήση των πόρων καθώς και τον διαμοιρασμό των καθηκόντων [13]. Η βασική διάκριση σε αυτό το μοντέλο είναι ότι διαχωρίζει τους πελάτες και τους διακομιστές. Ειδικότερα, ο πελάτης μπορεί να είναι ένας υπολογιστής, ο οποίος αναζητά από τον διακομιστεί, δεδομένα, υπηρεσίες και πόρους για να τα χρησιμοποιήσει. Η επικοινωνία μεταξύ των πελατών και των διακομιστών εξυπηρετεί την παροχή πρόσβασης στις πληροφορίες. Ωστόσο, τα χαρακτηριστικά των πελατών δεν είναι πάντοτε τα ίδια. Δηλαδή, μπορεί να πρόκειται για επιτραπέζιους υπολογιστές, φορητούς, τάμπλετ ή έξυπνα τηλέφωνα ακόμα και συσκευές του IoT. Από την άλλη, οι εφαρμογές που χρησιμοποιούν οι πελάτες προκειμένου να επιτύχουν την επικοινωνία με τους διακομιστές ενδέχεται να είναι εργαλεία που μεταφέρουν αρχεία, προγράμματα ηλεκτρονικού ταχυδρομείου ή περιηγητές ιστού και άλλα [14].

Ο διακομιστής μπορεί να οριστεί ως ένα λογισμικό που προσφέρει δεδομένα, πόρους και υπηρεσίες προς τους πελάτες, δηλαδή προς άλλους υπολογιστές μέσα από τη χρησιμοποίηση του δικτύου. Ο σχεδιασμός των διακομιστών στοχεύει στην υψηλή διαθεσιμότητά τους, την ικανότητά τους να μπορούν να διαχειριστούν πολλά αιτήματα ταυτόχρονα και στην αξιοπιστία τους. Για αυτό και χαρακτηρίζονται από εξειδικευμένο λογισμικό που έχει τη δυνατότητα να διανείμει με ασφάλεια και με αποδοτικότητα τους πόρους μέσα από τις κατάλληλες ρυθμίσεις [13]. Οι υπηρεσίες που μπορούν να παρέχουν οι διακομιστές είναι ποικίλες, από τις υπηρεσίες ηλεκτρονικού ταχυδρομείου και την αποθήκευση δεδομένων ως τις βάσεις δεδομένων και τη φιλοξενία ιστοσελίδων.

Η επικοινωνία ανάμεσα στους πελάτες και τους διακομιστές επιτυγχάνεται με βάση ένα συγκεκριμένο πρότυπο κατά το οποίο ο πελάτης στέλνει ένα μήνυμα/αίτημα στον διακομιστή. Ο διακομιστής με τη σειρά του μετά τη λήψη του αιτήματος, το επεξεργάζεται και το επιστρέφει στον πελάτη, ο οποίος επεξεργάζεται και ο ίδιος τις πληροφορίες [7].

Τα κύρια πλεονεκτήματα του μοντέλου πελάτη-διακομιστή είναι πως χρησιμοποιεί και διαχειρίζεται αποτελεσματικά τους πόρους. Επίσης, οι διακομιστές έχουν τη δυνατότητα να κλιμακώνονται με ανοδική τάση, ώστε να μπορούν να φιλοξενήσουν μεγάλη ζήτηση μέσα από τη διανομή υπηρεσιών σε πολλούς διακομιστές. Από την άλλη πλευρά, οι πόροι, συμπεριλαμβανομένων των βάσεων δεδομένων και των εφαρμογών, είναι εφικτό να κοινοποιηθούν σε πολλούς πελάτες, ενισχύοντας τη συνεργασία μεταξύ τους, αλλά και ελαχιστοποιώντας την επανάληψη. Είναι εξίσου σημαντικό να αναφερθεί πως τα ευαίσθητα δεδομένα μπορούν να προστατευτούν με τη λήψη ορισμένων μέτρων ασφαλείας, ώστε να μειωθούν οι ενδεχόμενες απειλές για τους πελάτες [12].

Οι διαδικασίες συντήρησης, οι διορθώσεις των σφαλμάτων και οι ενημερώσεις που γίνονται στο λογισμικό μπορούν να λάβουν χώρα στους διακομιστές και κατ’ επέκταση να ελαχιστοποιήσουν τις περιπτώσεις για την ενημέρωση των πελατών. Έτσι, το μοντέλο πελάτη-διακομιστή διαχωρίζει την επεξεργασία της εφαρμογής σε πολλαπλές μηχανές, επιτρέπει την ευκολότερη κοινή χρήση πόρων από πελάτη σε διακομιστές και μειώνει την αναπαραγωγή δεδομένων αποθηκεύοντας δεδομένα σε κάθε διακομιστή αντί για πελάτη [15].

Τα παραπάνω χαρακτηριστικά του μοντέλου πελάτη-διακομιστή ακολουθούνται και από κάποιες αδυναμίες δεδομένου ότι κεντρικό ρόλο παίζει ο διακομιστής. Δηλαδή, αν ο διακομιστής υπερφορτωθεί ή διακόψει τη λειτουργία του για οποιονδήποτε λόγο, τότε και οι υπηρεσίες των πελατών θα διακοπούν [14]. Εκτός αυτού, η αποτυχία που μπορεί να εμφανιστεί σε κάποια σημεία του μοντέλου μπορεί να έχει ως συνέπεια τη μετρίαση της χρήσης των αντιγράφων ασφαλείας.

Υπάρχουν έτσι πολλά ζητήματα σε ένα σύστημα πελάτη-διακομιστή. Στα περισσότερα δίκτυα πελάτη-διακομιστή, εμπλέκονται πάντα λίγοι διακομιστές που κάνουν τη ρύθμιση να μοιάζει με χάσιμο χρόνου. Ένα δίκτυο διακομιστή πελάτη είναι αρκετά δύσκολο να εγκατασταθεί, επομένως απαιτεί πολλούς διακομιστές για να μην αχρηστευτεί η εφαρμογή. Επίσης, πολλά δίκτυα πελατών-διακομιστών δεν είναι σωστά κατασκευασμένα και διαχειριζόμενα [12]. Η εγκατάσταση ενός δικτύου διακομιστών πελάτη είναι τόσο

περίπλοκη, επομένως απαιτεί εξειδικευμένους τεχνικούς και μηχανικούς συντήρησης για να το χειριστούν. Οι διακομιστές σχεδιάζονται για να πληρούν υψηλά πρότυπα για να είναι αξιόπιστα και έχουν καλύτερη απόδοση.

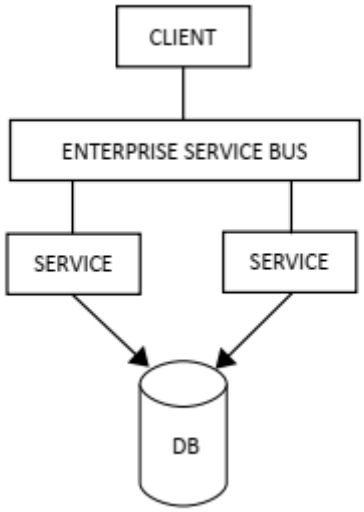
Το πιο σημαντικό όμως ζήτημα είναι αυτό της ασφάλειας. Το λειτουργικό σύστημα πελάτη είναι εύκολα προσβάσιμο από διακομιστές, και αυτό εκθέτει το σύστημα πελάτη σε μια σειρά προβλημάτων. Η ανταλλαγή μηνυμάτων μεταξύ του πελάτη και του διακομιστή οδηγεί σε πολλές προκλήσεις ασφαλείας. Πρόκειται για προκλήσεις ασφαλείας, μερικές από τις οποίες σχετίζονται με σωματικές βλάβες, απειλές και επίθεση από ιούς [12].

2.3 Υπηρεσιοστροφής αρχιτεκτονική

Η μονολιθική αρχιτεκτονική, παρά τα πλεονεκτήματά της, δημιουργεί ορισμένα εμπόδια ειδικά σε λογισμικά που έχουν μεγαλύτερη έκταση. Για αυτόν τον λόγο, παρατηρήθηκε μια στροφή προς την υπηρεσιόστροφη αρχιτεκτονική (SOA), η οποία ως προσέγγιση του λογισμικού, στοχεύει στη χαλαρή σύζευξη των υπηρεσιών του. Το κύριο στοιχείο που διακρίνει τη μονολιθική από την υπηρεσιόστροφη αρχιτεκτονική είναι ότι στη δεύτερη οι λειτουργίες της διαχωρίζονται σε ξεχωριστές υπηρεσίες. Η επικοινωνία ανάμεσα σε αυτές τις υπηρεσίες επιτυγχάνεται μέσω του δικτύου, δηλαδή, μέσω διεπαφών για την πραγματοποίηση των οποίων δεν απαιτείται ανεξάρτητο λειτουργικό σύστημα, προγραμματιστής γλώσσας ή υλικού [16].

Στην υπηρεσιόστροφη αρχιτεκτονική, επομένως, η δομή χωρίζεται σε πολλές υπηρεσίες, οι οποίες με τη σειρά τους ταυτοποιούνται σε διαφορετικά επίπεδα. Η Service Oriented Architecture (SOA) εξελίχθηκε στη συνέχεια σε μια από τις πιο επιτυχημένες αναπαραστάσεις της αρχιτεκτονικής πελάτη-διακομιστή με προστιθέμενη επιχειρηματική αξία που παρέχει επαναχρησιμοποιήσιμες και χαλαρά συνδεδεμένες υπηρεσίες. Η SOA δεν ανταποκρίθηκε στις προσδοκίες των πελατών και των επιχειρήσεων καθώς εξακολουθούσε να βασίζεται σε μονολιθικά συστήματα. Ωστόσο, η ανθεκτικότητα, η επεκτασιμότητα, η γρήγορη παράδοση λογισμικού και η χρήση λιγότερων πόρων παραμένουν ιδιαίτερα επιθυμητά χαρακτηριστικά, για αυτό και αναπτύχθηκαν νέες αρχιτεκτονικές [7].

Ορισμένοι μελετητές θεωρούν ότι οι μικροϋπηρεσίες συνιστούν κατηγορία της υπηρεσιόστροφης αρχιτεκτονικής, διότι πρόκειται για μια προσέγγιση που αξιοποιεί επαναχρησιμοποιούμενα μέρη του λογισμικού ή υπηρεσίες, οι οποίες αφορούν ένα μεγάλο εύρος της δραστηριότητας της εφαρμογής. Σε ορισμένες περιπτώσεις, η υπηρεσιόστροφη αρχιτεκτονική συνδέεται με πρωτόκολλα διαδικτυακών υπηρεσιών, ενώ επίσης χρησιμοποιεί το Enterprise Service Bus, δηλαδή έναν κεντρικό δίαυλο πληροφοριών, ο οποίος συμβάλλει στον συντονισμό και την επικοινωνία μεταξύ των υπηρεσιών [17].



Εικόνα 2. 1 Υπηρεσιοστροφής Αρχιτεκτονική

Πηγή: https://apothesis.eap.gr/archive/item/195557

Η αξιοποίηση της υπηρεσιόστροφης αρχιτεκτονικής έχει διάφορα πλεονεκτήματα. Συγκεκριμένα, συμβάλλει στην ανταπόκριση της τεχνολογίας με τις απαιτήσεις του επιχειρηματικού τομέα, παρέχει πολλές επιλογές διαφορετικών παρόχων, ενισχύει την εσωστρεφή διαλειτουργικότητα και αυξάνει την ενοποίηση στο πλαίσιο ενός οργανισμού. Όσον αφορά την ανταπόκριση της τεχνολογίας στις απαιτήσεις του επιχειρηματικού τομέα, υ υπηρεσιοστροφής αρχιτεκτονική δίνει τη δυνατότητα της ευελιξίας ώστε να υιοθετούνται και να εφαρμόζονται νέοι κανόνες, οι οποίοι επιτρέπουν την αξιοποίηση υπηρεσιών που είναι αφαιρετικές. Η συγκεκριμένη δυνατότητα της υπηρεσιόστροφης αρχιτεκτονικής παρέχει την ευκαιρία προσαρμογής στις μεταβολές των συνθηκών, που μπορεί να γίνονται κατά καιρούς και αφορούν τις συγχωνεύσεις εταιριών, τις αλλαγές στην τεχνολογία ή την προσφορά καινοτόμων επιχειρηματικών ιδεών [18].

Η παροχή πολλών και διαφορετικών παρόχων, αν και είναι σημαντικό πλεονέκτημα της υπηρεσιόστροφης αρχιτεκτονικής, δεν συνιστά κεντρικό της σημείο. Το κέρδος βέβαια που συνεπάγονται οι πολλοί πάροχοι στο εσωτερικό ενός συστήματος δεν είναι μικρό, αντιθέτως μέσω αυτού προσεγγίζεται η καλύτερα δυνατή λύση [19].

Όσον αφορά την εσωστρεφή διαλειτουργικότητα, η συνεργασία και κατ' επέκταση η σύνθεση των υπηρεσιών μπορεί να οδηγήσει στην αυτοματοποίηση της επιχειρηματικής διαδικασίας [20].Αυτό έχει ως άμεσο αποτέλεσμα τα δεδομένα να μπορούν να χρησιμοποιηθούν ξανά και επίσης να προσεγγίζονται με περισσότερη αποτελεσματικότητα οι ανάγκες και οι επιχειρησιακοί στόχοι. Μάλιστα, η υιοθέτηση κανόνων συμβάλλει στη διαλειτουργικότητα και κατ' επέκταση στην πραγματοποίηση του σκοπού των υπηρεσιών.

Επίσης, η αύξηση της ενοποίησης στον οργανισμό επιτυγχάνεται μέσα από τη διαλειτουργικότητα των πόρων και των εφαρμογών λογισμικού, γεγονός που συμβάλλει στη διατήρηση της αυτονομίας τους [20]. Η ενοποίηση είναι ιδιαίτερα χρήσιμη σε περίπτωση αντικατάστασης κάποιων μερών του συστήματος, δεδομένου ότι χάρη σε αυτή δεν χρειάζεται να αντικατασταθούν όλα τα εμπλεκόμενα μέρη, γεγονός που θα ήταν πιο δαπανηρό και χρονοβόρο.

Σχετικά με την υπηρεσιόστροφη ανάπτυξη λογισμικού, αυτή βασίζεται αρχικά σε ένα καθορισμένο συμβόλαιο στο οποίο οι διασυνδέσεις μεταξύ των υπηρεσιών είναι χαλαρές και παράλληλα αναπτύσσονται με αφαιρετικό τρόπο. Επίσης, χαρακτηριστικό των υπηρεσιών πρέπει να είναι η αυτονομία καθώς και η δυνατότητα να μην διατηρούν τα στοιχεία που λαμβάνουν μέσω των μηνυμάτων ή των αποκρίσεων που στέλνουν. Οι υπηρεσίες επίσης θα πρέπει να μπορούν να συνθέτονται και να φυλλομετρούνται [21].

Οι υπηρεσίες με χαλαρές διασυνδέσεις μπορούν να αλλάξουν πιο εύκολα και γρήγορα [20]. Αυτό το στοιχείο μειώνει την εξάρτηση μεταξύ των υπηρεσιών άρα και το ρίσκο για τη λειτουργία ολόκληρης της εφαρμογής στο σύνολό της, όταν αλλάζει ή αναπτύσσεται μια υπηρεσία. Επίσης, ο στόχος της αφαίρεσης υπηρεσιών είναι να αποκαλύψει μόνο τις λεπτομέρειες της υπηρεσίας που απαιτούνται για την αλληλεπίδραση με τον χρήστη. Η υλοποίηση θα πρέπει να γίνει έτσι ώστε το εσωτερικό, κρυφό τμήμα της υπηρεσίας να περιέχει τις λεπτομέρειες. Οποιεσδήποτε τροποποιήσεις στην υπηρεσία στο μέλλον θα πρέπει να γίνονται μόνο εσωτερικά, ώστε οι χρήστες να μπορούν να συνεχίσουν να τη χρησιμοποιούν χωρίς να χρειάζεται να τροποποιήσουν τη μορφή της κλήσης της [19].

Όταν οι υπηρεσίες επαναχρησιμοποιούνται, αυξάνεται το κέρδος και η παραγωγικότητα, καθώς ο χρόνος παράδοσης της υπηρεσίας στην επιχειρηματική αγορά θα είναι μικρότερος [22]. Άλλωστε, έχει αποδειχθεί ότι οι εφαρμογές που δομούνται σε πολλές και μικρές υπηρεσίες έχουν μεγαλύτερο κέρδος σε σύγκριση με τη μονολιθική αρχιτεκτονική καθώς είναι πιο ποιοτικές και μπορούν να ελεγχθούν πιο εύκολα [19].

Τα μέρη της υπηρεσιόστροφης αρχιτεκτονικής είναι ο καταναλωτής υπηρεσίας, ο πάροχος υπηρεσίας και ο διοργανωτής της υπηρεσίας. Ο καταναλωτής υπηρεσίας έχει την ευθύνη της κλήσης των υπηρεσιών, ενώ εναποθέτει το πρωτόκολλο μεταφοράς και την ασφάλεια στη διαδικασία της υλοποίησης. Στην υπηρεσιόστροφη αρχιτεκτονική, ο όρος καταναλωτής υπηρεσίας αναφέρεται σε ένα σύστημα ή μια εφαρμογή που χρησιμοποιεί τις υπηρεσίες που παρέχονται από άλλα συστήματα ή εφαρμογές, γνωστά ως πάροχοι υπηρεσιών. Αυτό το μοντέλο επιτρέπει την αποδοτική ανταλλαγή δεδομένων και λειτουργιών μεταξύ διαφορετικών συστημάτων που μπορεί να υπάρχουν σε διάφορες πλατφόρμες ή τοποθεσίες. Έτσι, ο καταναλωτής υπηρεσίας αναζητά και αλληλεπιδρά με υπηρεσίες που παρέχονται από άλλα συστήματα [23].

Ένας καταναλωτής υπηρεσίας μπορεί να είναι μια εφαρμογή, ένα υποσύστημα ή ακόμη και μια άλλη υπηρεσία που χρησιμοποιεί τις υπηρεσίες που προσφέρονται από άλλα μέρη του συστήματος. Ο καταναλωτής υπηρεσίας αλληλεπιδρά με τις υπηρεσίες μέσω των προκαθορισμένων διεπαφών που παρέχονται από τους παρόχους υπηρεσιών. Το μοντέλο αυτό επιτρέπει την αποδοτική ανάπτυξη και συντήρηση των εφαρμογών, καθώς ο καταναλωτής υπηρεσίας μπορεί να χρησιμοποιεί υπηρεσίες χωρίς να χρειάζεται να γνωρίζει τη λεπτομερή υλοποίηση του παρόχου υπηρεσιών. Αυτή η διαδικασία οδηγεί σε εύκολη αντικατάσταση ή αναβάθμιση των υπηρεσιών χωρίς να επηρεαστεί ο καταναλωτής, εφόσον οι διεπαφές παραμένουν σταθερές [24].

Από την άλλη, ο πάροχος υπηρεσίας αναφέρεται σε μια συνιστώσα του συστήματος ή μια εφαρμογή που παρέχει συγκεκριμένες υπηρεσίες που μπορούν να χρησιμοποιηθούν από άλλα συστήματα ή εφαρμογές μέσω προκαθορισμένων διεπαφών. Ο πάροχος υπηρεσίας αναλαμβάνει τον ρόλο της υλοποίησης της λειτουργικότητας που παρέχεται μέσω των υπηρεσιών του. Αυτές οι υπηρεσίες είναι συνήθως σχεδιασμένες να είναι αυτόνομες, ανεξάρτητες, και προσβάσιμες μέσω καθορισμένων διεπαφών [25]. Ο πάροχος υπηρεσίας είναι υπεύθυνος για την εκτέλεση συγκεκριμένων λειτουργιών και παροχής συγκεκριμένων υπηρεσιών, και παρέχει μια διεπαφή (interface) που επιτρέπει στους

καταναλωτές υπηρεσιών να αλληλεπιδρούν μαζί του. Οι λεπτομέρειες της υλοποίησης της υπηρεσίας παραμένουν κρυφές από τους καταναλωτές, που τη χρησιμοποιούν μόνο μέσω της διεπαφής που παρέχεται.

Ακόμη, ο διοργανωτής της υπηρεσίας σχετίζεται με ένα συστατικό του συστήματος ή μια υπηρεσία που αναλαμβάνει τον συντονισμό και τον έλεγχο της εκτέλεσης των υπηρεσιών και διαδικασιών σε ένα περιβάλλον υπηρεσιών. Ο διοργανωτής της υπηρεσίας διαδραματίζει σημαντικό ρόλο στον σχεδιασμό και την εκτέλεση σύνθετων επιχειρηματικών διαδικασιών που απαιτούν τη συνεργασία πολλών διαφορετικών υπηρεσιών. Ο διοργανωτής μπορεί να εκφραστεί μέσω διάφορων τεχνολογιών και μοντέλων, όπως οι γλώσσες σύνθεσης υπηρεσιών (Service Composition Languages) ή οι γλώσσες ορχηστρώσεων (Orchestration Languages) [26].

Αυτά τα μέρη της υπηρεσιόστροφης αρχιτεκτονικής πρέπει να είναι ανεξάρτητα και αυτόνομα και να έχουν αυξημένη αξιοπιστία. Μάλιστα, οι υπηρεσίες που βασίζονται σε SOA χρησιμοποιούνται για την ανάπτυξη συστημάτων μεγάλων επιχειρήσεων και, επομένως, διαδραματίζουν σημαντικό ρόλο στον τομέα του Διαδικτύου των Πραγμάτων (IoT) [27].

Κεφάλαιο 3 Η Αρχιτεκτονική Microservices

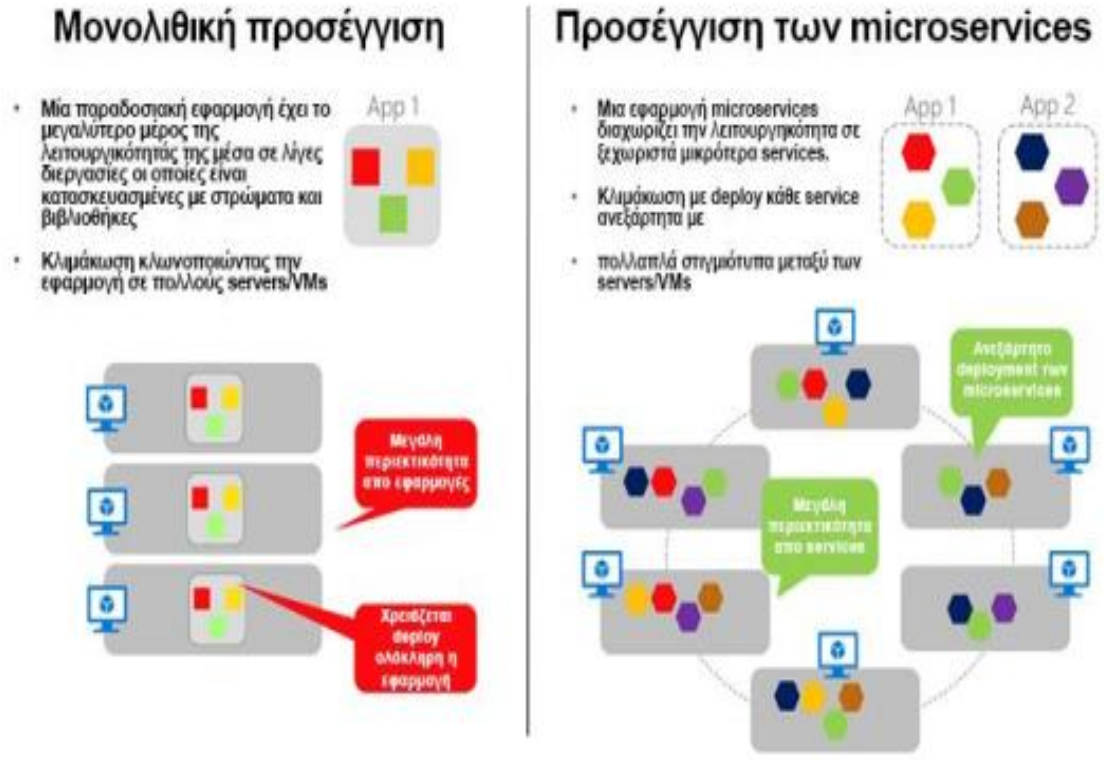
3.1 Γενικά χαρακτηριστικά

Η αρχιτεκτονική των Microservices (MSA) προέρχεται από ευέλικτες κοινότητες προγραμματιστών και έχει λάβει μετατόπιση τόσο στη βιομηχανία όσο και στον ακαδημαϊκό χώρο τα τελευταία χρόνια μετά την ευημερία της SOA (Service-oriented Architecture). Δεν έχει επιτευχθεί συμφωνία σχετικά με τη σχέση μεταξύ MSA και SOA. Ορισμένοι υποστηρικτές του MSA ισχυρίζονται ότι το MSA είναι ένα νέο αρχιτεκτονικό στυλ, ενώ ορισμένοι υποστηρικτές του SOA πιστεύουν ότι το MSA είναι απλώς μια προσέγγιση υλοποίησης του SOA [28].

Η αρχιτεκτονική microservices είναι ένα πρότυπο αρχιτεκτονικής λογισμικού που χαρακτηρίζεται από τον διαχωρισμό μιας εφαρμογής σε μικρά, ανεξάρτητα, αυτόνομα και αλληλεπιδρώντα μεταξύ τους κομμάτια, γνωστά ως υπηρεσίες. Η αρχιτεκτονική microservices περιλαμβάνει διάφορα χαρακτηριστικά στοιχεία. Έχουν δημοσιευτεί πολλές μελέτες που καλύπτουν τόσο την ίδια την αρχιτεκτονική των μικροϋπηρεσιών ως βάση για την ανάπτυξη νέων εφαρμογών όσο και τη μετάβαση σε αυτήν μέσα από τα χαρακτηριστικά και τις δυνατότητες εφαρμογής της ανάλογα με την κλίμακα της εταιρείας και το προϊόν [29].

Αρχικά, κάθε microservice λειτουργεί αυτόνομα, διαχωρισμένο από τα υπόλοιπα. Έχει, δηλαδή, τη δυνατότητα να αναπτύσσεται, να αναβαθμίζεται και να επιδιορθώνεται χωρίς να επηρεάζει τις υπόλοιπες υπηρεσίες. Επίσης, η αρχιτεκτονική microservices επιτρέπει την επιλογή διαφορετικών τεχνολογιών, γλωσσών προγραμματισμού και πλατφορμών για κάθε υπηρεσία, ανάλογα με τις απαιτήσεις της συγκεκριμένης υπηρεσίας και φυσικά τις προτιμήσεις του προγραμματιστή [30]. Τα microservices μπορούν ακόμη να επικοινωνούν μεταξύ τους μέσω προκαθορισμένων διεπαφών, που συνεργάζονται για την επίτευξη ενός συνολικού επιχειρησιακού στόχου. Έτσι, η διαχείριση των δεδομένων γίνεται κατανεμημένα μεταξύ των microservices, με κάθε υπηρεσία να διατηρεί τα δικά της δεδομένα, ενώ παράλληλα επιτρέπεται η εύκολη κλιμάκωση και η αποκλιμάκωση των μικρουπηρεσιών ανάλογα με τις ανάγκες του συστήματος.

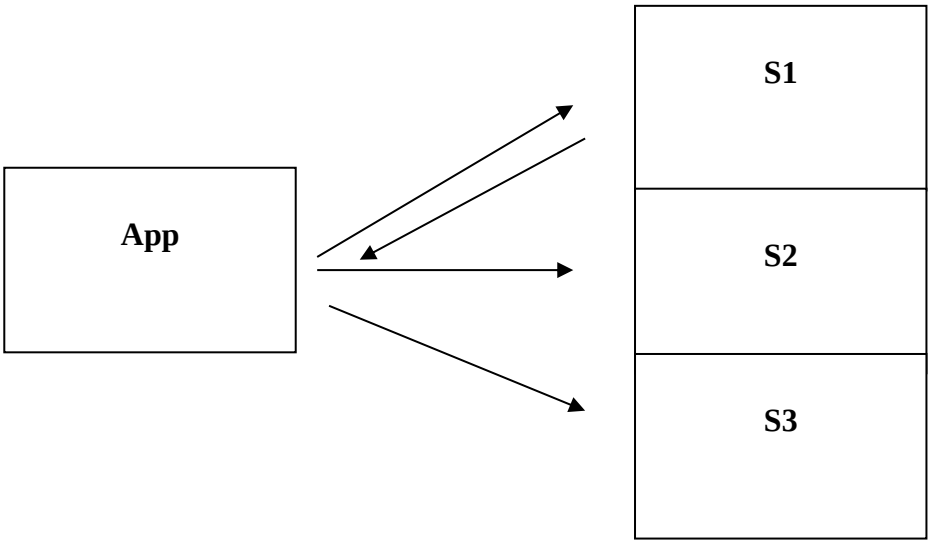
Η αρχιτεκτονική Microservices μπορεί να δημιουργήσει μια εφαρμογή με έναν τρόπο κατά τον οποίο κάθε διεργασία μιας υπηρεσίας εκτελείται μεμονωμένα μέσα από την αξιοποίηση διάφορων πρωτοκόλλων, όπως το AMQP, το Http / Https και το WebSocket. Κάθε domain που εφαρμόζει η αρχιτεκτονική των microservices μπορεί να αναπτυχθεί μεμονωμένα, ενώ μπορεί επίσης να στηριχθεί σε SQL ή NoSQL, με σκοπό την αποθήκευση των δεδομένων. Όλα αυτά τα χαρακτηριστικά της αρχιτεκτονική των Microservices προσδίδουν ευελιξία και συμβάλλουν στην αποτελεσματικότερη συντήρηση του συστήματος. Η ευελιξία έγκειται ακόμη στο γεγονός πως σε αντίθεση με τη μονολιθική αρχιτεκτονική, κάθε μονάδα μπορεί να κλιμακωθεί αυτόνομα και ανεξάρτητα. Ειδικότερα, αυτό σημαίνει πως κλιμακώνονται μόνο οι λειτουργίες, οι οποίες απαιτούν μεγαλύτερο εύρος ζώνης ή μεγαλύτερη ισχύ επεξεργασίας και όχι και εκείνες οι λειτουργίες, οι οποίες δεν έχουν αυτή την προϋπόθεση. Έτσι, αυτό συμβάλλει ώστε να εξοικονομείται παραπάνω κόστος σε σχέση με την περίπτωση της μονολιθικής αρχιτεκτονικής στο πλαίσιο ενός αξιόπιστου συστήματος [31].



Εικόνα 3. 1 Σύγκριση μονολιθικής αρχιτεκτονικής και αρχιτεκτονικής Microservices

Πηγή: https://nemertes.library.upatras.gr/server/api/core/bitstreams/14bc8842-3091-4c84-821b-6faf8c40e8a9/content

Στη βάση των microservices δεν δημιουργείται μόνο η ανταλλαγή μηνυμάτων χρήστη-διακομιστή, αλλά και η αλληλεπίδραση μεταξύ διακομιστών, παραδείγματα της οποίας είναι οι σύγχρονοι ιστότοποι πολυμέσων και οι υπηρεσίες Ιστού υψηλού φορτίου, καθώς και το Internet of Things [32]. Αυτή η ανταλλαγή μηνυμάτων ή η επικοινωνία στην αρχιτεκτονική των microservices σχετίζεται με την επικοινωνία ανάμεσα στις διάφορες υπηρεσίες. Η επικοινωνία μπορεί να επιτευχθεί με διάφορα μοτίβα, όπως με αυτό κατά το οποίο μια εφαρμογή έχει τη δυνατότητα να χρησιμοποιήσει κάθε υπηρεσία ξεχωριστά και με ταχύ τρόπο. Αυτό το μοτίβο διαφαίνεται στο σχήμα 3.1.



Σχήμα 3. 1 Μοτίβο άμεσης επικοινωνίας

Πηγή:

https://dspace.lib.ntua.gr/xmlui/bitstream/handle/123456789/48538/%CE%94%CE%B9%CF%80%CE%BB%CF%89%CE%BC%CE%B1%CF%84%CE%B9%CE%BA%CE%AE.pdf?sequence=1

Πρόκειται για ένα ευέλικτο μοτίβο επικοινωνίας μεταξύ των υπηρεσιών, το μειονέκτημα του οποίου όμως έγκειται στο γεγονός πως ενδέχεται να υπάρχει καθυστέρηση της επικοινωνίας λόγω των απομακρυσμένων θέσεων.

Γενικότερα, τα χαρακτηριστικά των micorservices θεωρούνται ιδιαίτερα ευέλικτα και για αυτόν τον λόγο η συγκεκριμένη τεχνολογία έχει υιοθετηθεί και ενσωματωθεί στο πλαίσιο πολλών επιχειρήσεων. Μάλιστα, όπως αναφέρει η έρευνα των [33], οι εταιρείες βασίζονται γενικά σε καθιερωμένες τεχνολογίες για την υλοποίηση υπηρεσιών, την επικοινωνία και την ανάπτυξη. Ο αντίκτυπος των Microservices στην ποιότητα του λογισμικού αξιολογήθηκε κυρίως ως θετικός στον επιχειρηματικό κλάδο. Βέβαια, ενώ η

συντηρησιμότητα της τεχνολογίας αυτής έλαβε τις πιο θετικές αναφορές, ορισμένα σημαντικά ζητήματα συνδέθηκαν με την ασφάλεια.

3.2 Σχεδιαστικά πρότυπα και πρακτικές

Η αυξανόμενη υιοθέτηση του cloud computing έχει επηρεάσει τη διαδικασία ανάπτυξης λογισμικού. Η ανάπτυξη εφαρμογών cloud έχει οδηγήσει στην υιοθέτηση αρχιτεκτονικών μικροϋπηρεσιών, σε αντίθεση με τη μονολιθική αρχιτεκτονική, προκειμένου να επωφεληθούν από το cloud computing χαρακτηριστικά, όπως η επεκτασιμότητα, η διαθεσιμότητα ή η αξιοπιστία [34].

Η αρχιτεκτονική των μικροϋπηρεσιών είναι ένα αρχιτεκτονικό στυλ προσανατολισμένο προς τη σπονδυλοποίηση, όπου η ιδέα είναι να χωριστεί η εφαρμογή σε μικρότερες, διασυνδεδεμένες υπηρεσίες, που εκτελούνται ως ξεχωριστή διαδικασία που μπορεί να αναπτυχθεί ανεξάρτητα, να κλιμακωθεί και να δοκιμαστεί [35].

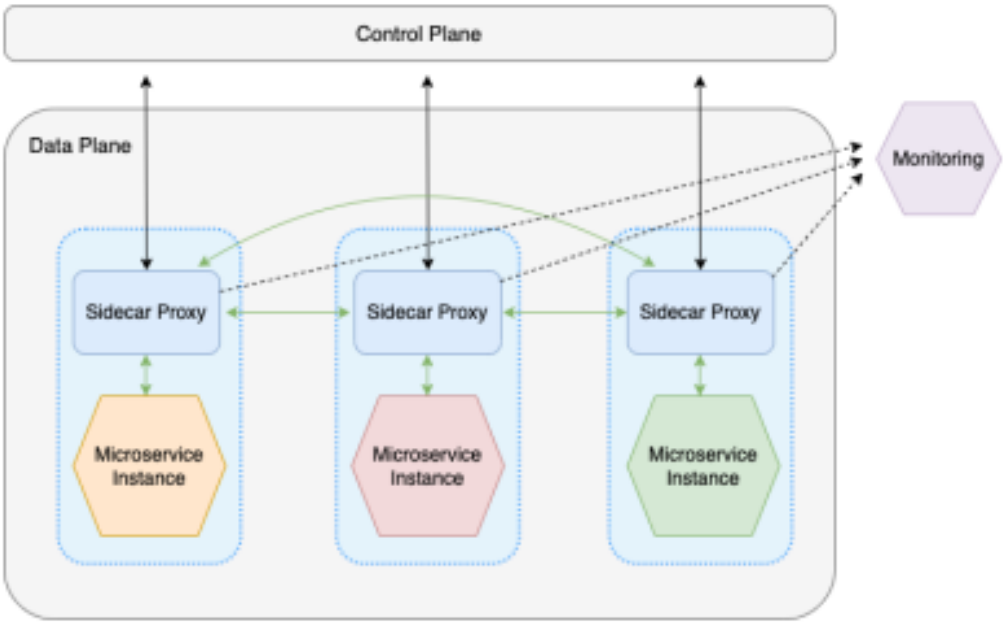
Ο σχεδιασμός μικρουπηρεσιών για μια δεδομένη επιχειρηματική ικανότητα ή τομέα, συνήθως χρησιμοποιεί μοτίβα, όπως ο Σχεδιασμός βάσει τομέα (DDD), η αρχή της ενιαίας ευθύνης (SRP) ή ο νόμος του Conway. Αυτά διαβεβαιώνουν ότι οι μικροϋπηρεσίες είναι περιορισμένες έτσι ώστε να μπορούν να κλιμακωθούν ανεξάρτητα. Ωστόσο, τα έργα συχνά δυσκολεύονται να τα συνδέσουν σωστά, με αποτέλεσμα την ανεπαρκή γνώση για αποφάσεις που σχετίζονται με τη διαμέριση της βάσης δεδομένων, το κατάλληλο μέγεθος της μικρουπηρεσίας, την επικοινωνία μεταξύ των υπηρεσιών και τα μηνύματα, τα οποία δεν αντιμετωπίζονται συστηματικά από αυτά τα πρότυπα. Με την εφαρμογή μιας μεθόδου μοντελοποίησης στη διαδικασία σχεδιασμού των microservises μπορεί κανείς να προβλέψει ζητήματα σχετικά με περιορισμένα περιβάλλοντα για μικροϋπηρεσίες, συγκεκριμένη συμπεριφορά εντός υπηρεσίας, διαχωρισμό μοντέλων διεπαφών και δεδομένων και απαιτήσεις επικοινωνίας και ανταλλαγής μηνυμάτων μεταξύ υπηρεσιών [35].

Πιο ειδικά, η μέθοδος 4SRS διασφαλίζει την ευθυγράμμιση της σχεδιασμένης αρχιτεκτονικής με τις απαιτήσεις του χρήστη, προκειμένου να υποστηρίξει έναν σωστό αρχιτεκτονικό σχεδιασμό συμβατό με τις βασικές αρχές της αρχιτεκτονικής μικροϋπηρεσιών, το 4SRS-MSLA. Ζητήματα, όπως η συνέπεια δεδομένων, η ασφάλεια, η

επικοινωνία, η ανάπτυξη και άλλα μοτίβα μπορούν επίσης να αντιμετωπιστούν με αυτή τη μέθοδο [36].

Για λόγους αναπαράστασης, αυτή η προσέγγιση υιοθετεί τη γλώσσα μοντελοποίησης αρχιτεκτονικής με γνώμονα τις υπηρεσίες (SoaML). Επιπλέον, με βάση την προϋπόθεση ότι μια σωστή προδιαγραφή απαιτήσεων επωφελείται από πολλαπλές προοπτικές, η προσαρμοσμένη μέθοδος περιλαμβάνει εξόδους για διαφορετικά διαγράμματα SoaML, όπως Συμμετέχοντες υπηρεσιών, Δυνατότητες, Αρχιτεκτονικές Υπηρεσιών και Διεπαφές υπηρεσιών. Αν και δεν υπάρχει ένα κοινό πρότυπο για τη μοντελοποίηση των MSA φαίνεται να υπάρχει μια τάση να χρησιμοποιούνται γλώσσες που προσανατολίζονται στην περιγραφή αρχιτεκτονικών που βασίζονται σε υπηρεσίες (SoaML, SOMA, SOADL), αλλά και UML για υπηρεσίες και λειτουργίες μοντελοποίησης [37].

Μεταξύ των σχεδιαστικών προτύπων βρίσκεται το Πλέγμα Service με Sidecar. Πρόκειται για να πλέγμα υπηρεσίας που είναι διαμορφώσιμο επίπεδο υποδομής χαμηλής καθυστέρησης και έχει σχεδιαστεί για να αντιμετωπίζει μεγάλο όγκο επικοινωνίας μεταξύ διεργασιών. Το μοτίβο πλέγματος υπηρεσίας εφαρμόζεται γενικά ως πίνακας ελαφρών διακομιστών δικτύου που ονομάζονται sidecar, χωρίς να χρειάζεται η εφαρμογή να γνωρίζει τους διακομιστές μεσολάβησης [38].



Εικόνα 3. 2 Σχεδιαστικό πρότυπο Πλέγμα Service με Sidecar

Πηγή: https://www.politesi.polimi.it/retrieve/04fb6ce8-cc33-4207-a932-6e0701e8f932/2022_04_ESAS.pdf

Οι διακομιστές μεσολάβησης sidecar σε κάθε παρουσία υπηρεσίας χειρίζονται την επικοινωνία μεταξύ των διεργασιών, την παρακολούθηση και πολλές άλλες λειτουργίες. Οι υποδομές περιλαμβάνουν ελαστικότητα (ανοχή σφαλμάτων, εξισορρόπηση φορτίου), ανακάλυψη υπηρεσιών, δρομολόγηση, παρατηρησιμότητα, ασφάλεια, έλεγχο πρόσβασης, υποστήριξη πρωτοκόλλου επικοινωνίας και παρόμοια. Το μοτίβο πλέγματος εξυπηρέτησης χωρίζεται σε δύο μέρη, δηλαδή, το τμήμα ελέγχου και το τμήμα δεδομένων, που συνήθως αναφέρονται ως το επίπεδο ελέγχου και το επίπεδο δεδομένων. Το επίπεδο ελέγχου δημιουργεί πίνακες δρομολόγησης και αναπτύσσει τη διαμόρφωση δρομολόγησης στους διακομιστή μεσολάβησης στο επίπεδο δεδομένων. Η πραγματική προώθηση της κίνησης του δικτύου γίνεται από τους διακομιστή μεσολάβησης στο επίπεδο δεδομένων, και για το λόγο αυτό, το επίπεδο δεδομένων λέγεται επίσης ότι είναι το επίπεδο προώθησης [38].

Επίσης, με σκοπό τη διαχείριση της δραστηριότητας επαναφοράς συναλλαγών και της συνέπειας μεταξύ των βάσεων δεδομένων ανά μικροϋπηρεσίες, το SAGA μπορεί να θεωρηθεί ως μηχανισμός πρόληψης αποτυχίας. Βασικά διατηρεί το αρχείο της ακολουθίας όλων των τοπικών συναλλαγών/δραστηριοτήτων που πρέπει να εκτελεστούν από διάφορες μικροϋπηρεσίες μαζί με την κατάστασή τους για την ολοκλήρωση μιας συναλλαγής που εκτείνεται σε πολλαπλές μικροϋπηρεσίες. Σε επιτυχημένη δραστηριότητα, η μικρουπηρεσία στέλνει ένα μήνυμα ή δημοσιεύει συμβάν που ενεργοποιεί την επόμενη δραστηριότητα στη σειρά. Σε περίπτωση αποτυχίας οποιασδήποτε μικροϋπηρεσίας, το SAGA διασφαλίζει την επαναφορά των δραστηριοτήτων που εκτελούνται στη συναλλαγή μερικής διεκπεραίωσης [39].

Ακόμη, η Βάση δεδομένων ανά Υπηρεσία επιτρέπει επίσης την ανάπτυξη και την κλιμάκωση των υπηρεσιών και δίνει τη δυνατότητα στις ομάδες να είναι πιο ανεξάρτητες. Με αυτόν τον τρόπο, κάθε υπηρεσία μπορεί να χρησιμοποιήσει τον τύπο βάσης δεδομένων που ταιριάζει καλύτερα στις ανάγκες της. Για παράδειγμα, μια υπηρεσία αρχειοθέτησης που παρέχει γρήγορες αναζητήσεις κειμένου θα μπορούσε να χρησιμοποιήσει το ElasticSearch, ενώ μια άλλη υπηρεσία που μοντελοποιεί τις κοινωνικές αλληλεπιδράσεις χρησιμοποιώντας γραφήματα θα μπορούσε να χρησιμοποιήσει τη βάση δεδομένων γραφημάτων Neo4j. Αν και η ύπαρξη ξεχωριστού διακομιστή βάσης δεδομένων ανά υπηρεσία ευθυγραμμίζεται με την ιδέα χαλαρής σύζευξης, αυξάνει την πολυπλοκότητα όσον αφορά την υλοποίηση συναλλαγών που εκτείνονται σε πολλαπλές

υπηρεσίες, καθώς δεν υποστηρίζουν όλες οι βάσεις δεδομένων No-SQL ατομικές δεσμεύσεις μεταξύ παρουσιών κατανεμημένων βάσεων δεδομένων, οι οποίες γενικά υλοποιούνται με τη χρήση αλγορίθμων ατομικών συναλλαγών [38].

3.3 Πλεονεκτήματα Microservices

Ένα από τα κύρια πλεονεκτήματα της αρχιτεκτονικής των microservices είναι η δυνατότητα προσθήκης και αφαίρεσης στοιχείων κατά περίπτωση χωρίς αλλαγή του συστήματος στο σύνολό του. Αυτή η προσέγγιση μπορεί να αξιοποιηθεί για παράδειγμα, για να καταγράψει γεγονότα που σχετίζονται με ένα συγκεκριμένο αθλητικό γεγονός υψηλού προφίλ [40]. Έτσι, δημιουργείται ένα εξειδικευμένο microservice, το οποίο ενσωματώνεται εύκολα στην πλατφόρμα πολυμέσων και αφού χάσει τη συνάφειά του, απλώς αφαιρείται από αυτήν.

Ταυτόχρονα, ένα σημαντικό φορτίο που δημιουργείται από χρήστες που ενδιαφέρονται για το υποδεικνυόμενο συμβάν πηγαίνει στους διακομιστές που είναι αφιερωμένοι για αυτήν τη μικρουπηρεσία και δεν φορτώνει το κύριο σύστημα. Για τον χρήστη, ωστόσο, αυτή η διαίρεση είναι ανεπαίσθητη [1]. Από την άποψη των οικονομικών οφελών, η προσθήκη μιας νέας μικροϋπηρεσίας κατάστασης δεν συνεπάγεται αλλαγές στην κύρια υπολογιστική υποδομή, καθώς προστίθεται «στο πλάι», πράγμα που σημαίνει ότι το κόστος της ανάπτυξής της είναι το χαμηλότερο από όλα τα δυνατά.

Επιπλέον, ένα σημαντικό πλεονέκτημα της αρχιτεκτονικής microservices είναι το γεγονός ότι, λόγω της αδύναμης σύνδεσης μεταξύ των στοιχείων δίνει τη δυνατότητα να χρησιμοποιηθούν διαφορετικές γλώσσες προγραμματισμού ακόμα και πιο πρόσφατες. Για τις μικροϋπηρεσίες κατάστασης, αυτό σημαίνει τη δυνατότητα χρήσης των πιο πρόσφατων γλωσσών για την ταχεία δημιουργία προϊόντων λογισμικού με απλοποιημένες δοκιμές, καθώς ο κύκλος ζωής ενός τέτοιου προϊόντος λογισμικού είναι σχεδόν πάντα σύντομος και το αποτέλεσμα δεν συνεπάγεται ανάπτυξη. Αυτόματα, αυτό σημαίνει εξοικονόμηση των πόρων και εύκολη πρόσβαση στην αγορά.

Οι τεχνολογίες ανάπτυξης βελτιώνονται συνεχώς και αυτό συνεπάγεται τη συνεχή εμφάνιση ολοένα και περισσότερων νέων γλωσσών προγραμματισμού, μια αλλαγή στο μερίδιό τους στην ανάπτυξη, καθώς και την εξαφάνιση όσων έχουν δημιουργηθεί στο παρελθόν λόγω της απώλειας του ενδιαφέροντος της κοινότητας σε αυτά [41]. Μαζί με

αυτό, η τεχνολογική στοίβα για την ανάπτυξη μικροϋπηρεσιών κατάστασης μπορεί επίσης να αλλάξει.

Ειδικότερα, οι μικροϋπηρεσίες μπορούν να βασίζονται στην ετερογένεια της τεχνολογίας, πράγμα που σημαίνει ότι κάθε υπηρεσία σε ένα σύστημα μπορεί να χρησιμοποιεί διαφορετική τεχνολογία από τις άλλες υπηρεσίες για την επίτευξη των επιθυμητών στόχων και της απόδοσης. Ένα άλλο πλεονέκτημα των microservices είναι ότι εάν ένα στοιχείο του συστήματος αποτύχει, τότε δεν επηρεάζει ολόκληρο το σύστημα. Επίσης, η διαδικασία της κλιμάκωσης μπορεί να είναι πιο προσιτή σε σύγκριση με τη μονολιθική κλιμάκωση εφαρμογών επειδή μόνο οι υπηρεσίες που χρειάζονται πραγματική κλίμακα κλιμακώνονται στην αρχιτεκτονική των μικροϋπηρεσιών, αντίθετα με μια μονολιθική εφαρμογή που απαιτείται να κλιμακωθεί ως ολόκληρη μονάδα που μπορεί να οδηγήσει σε υψηλότερη χρήση υλικού [42].

Ανάμεσα στα πλεονεκτήματα της παρούσας αρχιτεκτονικής είναι και η ευκολίας δημιουργίας ή ανάπτυξης, καθώς κάθε υπηρεσία μπορεί να αναπτυχθεί ανεξάρτητα χωρίς να επηρεάζεται η απόδοση άλλων υπηρεσιών. Ακόμη, η αρχιτεκτονική των microservices βοηθά τις εταιρείες να ευθυγραμμίσουν την αρχιτεκτονική της με την οργανωτική της δομή, η οποία θα τις βοηθήσει να ελαχιστοποιήσουν τον αριθμό των ατόμων που εργάζονται σε μια συγκεκριμένη βάση κωδικών. Κατά συνέπεια, οι μικροϋπηρεσίες επιτρέπουν την οργανωτική ευθυγράμμιση. Περαιτέρω πλεονεκτήματα είναι η δυνατότητα σύνθεσης και η βελτιστοποίηση για δυνατότητα αντικατάστασης [43].

Σύμφωνα με τους [44], η αρχιτεκτονική microservices θα διευκολύνει τις διαδικασίες συντήρησης, επαναχρησιμοποίησης, επεκτασιμότητας, διαθεσιμότητας και αυτοματοποιημένης ανάπτυξης όταν θα χρησιμοποιηθεί, και αυτά θεωρούνται τα πλεονεκτήματα της αρχιτεκτονικής microservices. Όλα αυτά τα χαρακτηριστικά οδηγούν πολλές εταιρείες προς την υιοθέτηση της αρχιτεκτονικής microservices για να ενεργοποιήσουν τις ομάδες ανάπτυξής τους και να μπορούν να συντονίζονται εύκολα μεταξύ τους [45].

Επομένως, παρά τα πλεονεκτήματα των γλωσσών ταχείας ανάπτυξης, που χαρακτηρίζονται από ευκολία λόγω της πιο μελετημένης σύνταξης και από ταχύτητα εξαιτίας των πολλών ενσωματωμένων τυπικών λειτουργιών, εμφανίζουν επίσης σημαντικά μειονεκτήματα λόγω της «νεότητάς» τους με τη μορφή πολλών εσωτερικών σφαλμάτων που δεν έχουν ακόμη εντοπιστεί. Συνεπώς, η επιλογή της γλώσσας

προγραμματισμού αποτελεί και ευθύνη του προγραμματιστή, προκειμένου να ελέγξει όλα τα απαραίτητα γλωσσικά εργαλεία για τυχόν σφάλματα [1]. Αυτό βέβαια έχει ως συνέπεια να επιβραδύνεται η διαδικασία της ανάπτυξης.

3.4 Προκλήσεις Microservices

Είναι σαφές πως η αρχιτεκτονική των Microservices συνεπάγεται σημαντικά πλεονεκτήματα ειδικά σε σύγκριση με τη μονολιθική αρχιτεκτονική λόγω της ευελιξίας της και της αυτονομίας των υπηρεσιών της. Παρόλα αυτά τα χαρακτηριστικά, η συγκεκριμένη αρχιτεκτονική ακολουθείται από ορισμένες προκλήσεις. Βασικό μειονέκτημα θεωρείται το γεγονός πως οι προγραμματιστές χρειάζονται να αφιερώσουν χρόνο και προσοχή για τη δημιουργία ενός συστήματος, δεδομένου πως αφενός η κατανομή των υπηρεσιών είναι μία σύνθετη διαδικασία και αφετέρου η λειτουργία κάθε αυτόνομης υπηρεσίας θα πρέπει να ελεγχθεί επαρκώς, καθώς η επικοινωνία μεταξύ των υπηρεσιών είναι ιδιαίτερα σημαντική [46].

Οι προγραμματιστές, δηλαδή, καλούνται να διαθέτουν διαφορετικές δεξιότητες και γνώσεις, ώστε να μπορούν να προσαρμόσουν τα microservices στην κάθε γλώσσα προγραμματισμού και επίσης να είναι ικανοί να διαμορφώσουν με επάρκεια και αποτελεσματικότητα το πλαίσιο πραγματοποίησης του συστήματος [47]. Γίνεται αντιληπτό, λοιπόν, πως παρατηρείται μια πολυπλοκότητα όσον αφορά τη διαμόρφωση ενός συστήματος, γεγονός που προϋποθέτει και την αγαστή συνεργασία μεταξύ των προγραμματιστών. Εκτός αυτού, εφόσον στην αρχιτεκτονική των Microservices κάθε υπηρεσία είναι αυτόνομη, αυτό συνεπάγεται πως θα καταλαμβάνει και περισσότερο χώρο στη μνήμη σε σύγκριση με τη μονολιθική αρχιτεκτονική.

Βέβαια, εκτός από τη διαδικασία της κατάτμησης για την οποία οι προγραμματιστές θα πρέπει να έχουν τις απαραίτητες γνώσεις, οι ίδιοι καλούνται να γνωρίζουν και κάποιες επιχειρησιακές διαδικασίες, ώστε να διευκολύνεται το εγχείρημα που σχετίζεται με τη λειτουργικότητα του συστήματος [48].

Ακόμη, κάθε υπηρεσία χαρακτηρίζεται από συγκεκριμένες υποχρεώσεις. Έτσι, πρόκληση αποτελεί το γεγονός πως οι κατάλληλες υποχρεώσεις πρέπει να δοθούν σε εκείνη την υπηρεσία, η οποία θα πρέπει να τις επιτελέσει. Διαφορετικά, θα τεθεί υπό αμφισβήτηση η λειτουργικότητα του συστήματος. Παράλληλα, δεδομένης της κατάτμησης των

microservices, είναι εύλογο ότι καταλαμβάνουν περισσότερους πόρους μνήμης [49], ενώ η ανοχή στα σφάλματα θα πρέπει εξίσου να ληφθεί υπόψη.

Κεφάλαιο 4 Από τη Μονολιθική Αρχιτεκτονική στην Αρχιτεκτονική Microservices

4.1 Η σημασία της μετάβασης

Η αρχιτεκτονική μικροϋπηρεσιών έχει εισαχθεί ως μια νέα εναλλακτική λύση στη μονολιθική αρχιτεκτονική. Συνεπάγεται πολλά πλεονεκτήματα όπως επεκτασιμότητα, αξιοπιστία, αύξηση ευελιξίας και παραγωγικότητας, ανθεκτικότητα σε αστοχίες, ευκολία ανάπτυξης και συντήρησης και μείωση του χρόνου διάθεσης στην αγορά. Ως εκ τούτου, οι εταιρείες λογισμικού έχουν δείξει μια τάση να μετατρέπουν την αρχιτεκτονική των παλαιών εφαρμογών τους από μονόλιθους σε αρχιτεκτονική μικροϋπηρεσιών. Σε αυτή τη διαδικασία μετασχηματισμού, οι ομάδες ανάπτυξης λογισμικού αντιμετωπίζουν την πρόκληση της μετάβασης μεγάλων εφαρμογών στη νέα αρχιτεκτονική, όπου είναι σημαντική η κατανόηση της τρέχουσας εφαρμογής και η επαναχρησιμοποίηση της υπάρχουσας βάσης κώδικα [50].

Η μονολιθική αρχιτεκτονική χαρακτηρίζεται από μια ενιαία δομή, συνεπώς η αποσύνθεσή της αποτέλεσε μια πρόκληση και συνάμα ένα σημαντικό βήμα για τον κλάδο των προγραμματισμού, ενισχύοντας παράλληλα και τις δεξιότητες, αλλά και τις γνώσεις των ειδικών [51].

Η μετάβαση από μια μονολιθική σε μια αρχιτεκτονική μικροϋπηρεσιών είναι ένα επαναλαμβανόμενο βήμα σε πολλά έργα λογισμικού. Με την αυξανόμενη κατανεμημένη φύση του μετασχηματισμένου συστήματος, προκύπτουν νέες προκλήσεις για τη συνέπεια και την ανάπτυξη δεδομένων, οι οποίες μπορούν να αντιμετωπιστούν με την ενσωμάτωση προτύπων μικροϋπηρεσιών. Ωστόσο, η χρήση τέτοιων μοτίβων είναι πολύπλοκη και χρονοβόρα [52].

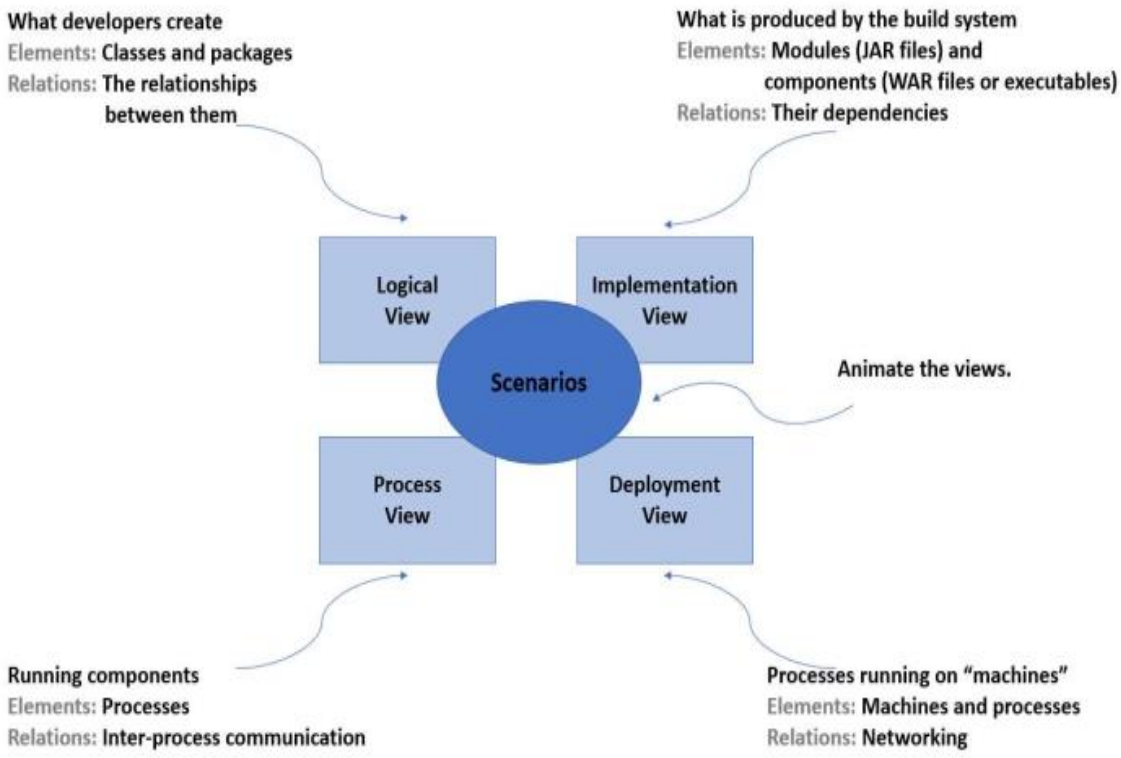
Για να μετατραπεί μια εφαρμογή που βασίζεται στη μονολιθική αρχιτεκτονική σε αρχιτεκτονική των microservices, θα πρέπει πρώτα να επιτευχθεί ενός είδους αξιολόγηση. Σε περίπτωση που ο κώδικας βάσης αποκτήσει μεγαλύτερες διαστάσεις, τότε και η αρχιτεκτονική των μικροϋπηρεσιών θα αποτελέσει καλή λύση.

Ο εκσυγχρονισμός μιας εφαρμογής από μονολιθική σε αρχιτεκτονική των microservices θεωρείται απαιτητικός, καθώς πρόκειται για μια μεγάλη αλλαγή. Σύμφωνα με τον [53], η μετάβαση αυτή δεν είναι μια εύκολη, αλλά μια σύνθετη διαδικασία, που σχετίζεται με διάφορα επίπεδα στη διαδικασία ανάπτυξης ενός λογισμικού.

Με βάση τον [54], η μετάβαση αυτή προϋποθέτει ένας προμηθευτής να ενσωματώσει ένα τυπικό λογισμικό ή να το αναπλάσει σύμφωνα με το καινούριο σχεδιαστικό πρότυπο είτε να το ανακατασκευάσει. Οι παραπάνω μέθοδοι μπορούν βέβαια να εφαρμοστούν και συνδυαστικά. Επίσης, η εφαρμογή μπορεί να γραφτεί εκ νέου συνολικά ή να εξελιχθεί σταδιακά προς την αρχιτεκτονική των microservices. Επίσης, η στρατηγική Bing Bang ωθεί τον κώδικα προς την αλλαγή, ενώ το «Διαίρει και βασίλευε» «σπάει» τον κώδικα σε δύο μέρη έως ότου να ολοκληρωθεί η διαδικασία της μετάβασης.

4.2 Μεθοδολογίες μετάβασης

Η μετάβαση από τη μονολιθική αρχιτεκτονική στην αρχιτεκτονική των microservices συνιστά μια διαδικασία αποσύνθεσης, για αυτό και είναι απαραίτητο να βασίζεται σε κάποιες αρχές, δηλαδή σε μεθοδολογίες. Όπως προέκυψε από την ανασκόπηση της βιβλιογραφίας, διάφοροι ερευνητές κατά καιρούς έχουν αναπτύξει διαφορετικού τύπου μεθοδολογίες, ώστε αυτές να είναι ικανές να καταστήσουν ένα σύστημα λειτουργικό.



Εικόνα 4. 1 Απόψεις αρχιτεκτονικής

Πηγή: https://estia.hua.gr/file/lib/default/data/26493/theFile

Αρχικά, ο [36] με βάση την ιδέα του Kruntchen, κάνει λόγο για ένα μοντέλο, το οποίο στηρίζεται σε τέσσερις βασικές αρχές. Η πρώτη από αυτές τις αρχές είναι η λογική προβολή. Πρόκειται για στοιχεία λογισμικού, δηλαδή πακέτα και κλάσεις, τα οποία συσχετίζονται μεταξύ τους και συνδέονται με εξαρτήσεις ή κληρονομικότητα. Έπειτα, η αρχή της προβολής υλοποίησης συνίσταται από λειτουργικές μονάδες, οι οποίες είναι αντιπροσωπευτικές ενός κώδικα, ενώ μεταξύ τους είναι επίσης εξαρτημένες. Επίσης, η προβολή διεργασίας αντιλαμβάνεται κάθε στοιχείο ως μια διεργασία, ενώ οι σχέσεις μεταξύ αυτών συντελούν στην επικοινωνία. Τέλος, η ανάπτυξη περιλαμβάνει στοιχεία και διεργασίες. Τα στοιχεία συνίστανται από μηχανές, η σχέση των οποίων με τις διεργασίες δημιουργεί τη δικτύωση.

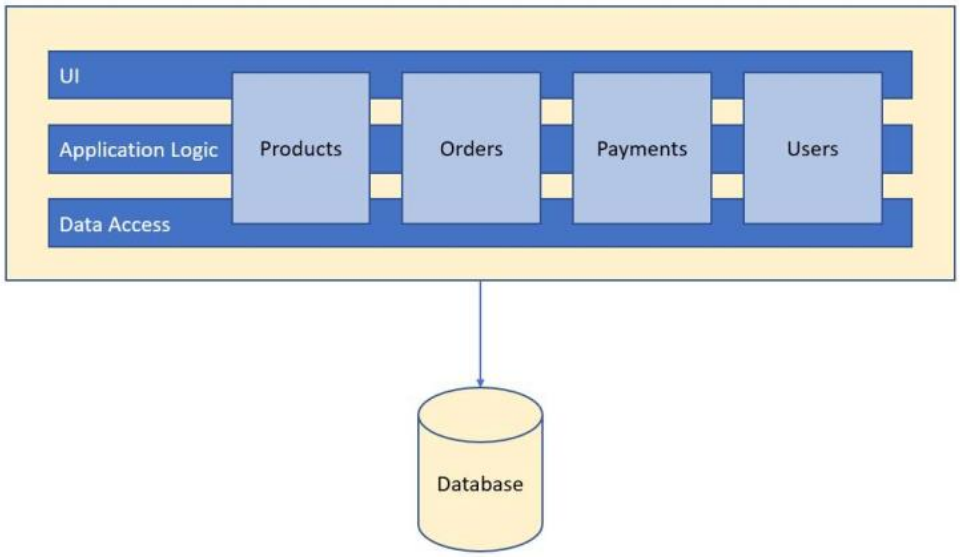
Επίσης, η μεθοδολογία κατά Amundsen έγκειται στον διαχωρισμός ενός συστήματος σε μικρότερα κομμάτια [55]. Δηλαδή, οι υπηρεσίες που έχουν γραφτεί σε C, υπάρχει δυνατότητα μέσω του Node.js να χωριστούν από τα chatty στοιχεία τους. Ακόμη, κριτήρια για τον διαχωρισμό αποτελεί και η γεωγραφία, το αρχιτεκτονικό στυλ, οι εμπειρίες και οι λειτουργίες του συστήματος.

Συμπληρωματικά, οι [56] αναφέρουν τρεις απόψεις διαχωρισμού, τη φυσική, τη λειτουργική και τη βασισμένη σε υπηρεσίες αποσύνθεση. Η πρώτη ο κώδικάς της συνδέεται άμεσα με πακέτα, αρχεία και κλάσεις. Αν το έρεισμα δηλαδή του συστήματος είναι τα αρχεία, τότε η κατάτμηση της αρχιτεκτονικής μπορεί να γίνει εύκολα, χωρίς αυτό να σημαίνει πως ένα στοιχείο μπορεί να εμπλέκεται μόνο σε μία συγκεκριμένη λειτουργία. Παρόλο που η φυσική αποσύνθεση λαμβάνει υπόψη τα φυσικά μέρη μιας εφαρμογής, η λειτουργική αποσύνθεση σχετίζεται με τη λειτουργικότητα αυτής. Βέβαια, συνήθως υπάρχουν και υπολειτουργίες που συνδέονται με στοιχεία και μεθόδους, τα οποία στοιχεία έχουν την ικανότητα να αντιγραφούν για κάθε μονάδα. Ακόμη, η service-based αποσύνθεση αναφέρεται στη χρήση ενός συστήματος, καθώς και στις απαιτήσεις που το συνοδεύουν, ώστε η εκτέλεσή του να είναι αποτελεσματική.

Από την άλλη πλευρά, η μεθοδολογία συρρίκνωσης έγκειται αρχικά στη διακοπή του ανοίγματος τρυπών. Η έννοια της διακοπής ανοίγματος τρυπών αποτελεί ένα σημαντικό στοιχείο της διαδικασίας μετάβασης από ένα μονολιθικό σύστημα σε ένα σύστημα με microservices. Η διακοπή ανοίγματος τρυπών αναλαμβάνει τον ρόλο της σύνδεσης των νέων microservices με τον υπάρχοντα μονολιθικό κώδικα. Στο πλαίσιο των microservices, η διακοπή ανοίγματος τρυπών περιλαμβάνει την αναγνώριση των λειτουργιών του μονολιθικού συστήματος που θα υποστούν αντικατάσταση ή επέκταση με microservices, τον σχεδιασμό και τη δημιουργία νέων microservices που θα αναλάβουν τις επιλεγμένες λειτουργίες και την ενσωμάτωσή τους στο υπάρχον σύστημα. Έτσι, εξασφαλίζεται ο παράλληλος τρόπος λειτουργίας των μικρουπηρεσιών και των υπολοίπων τμημάτων του συστήματος και εκτελείται η αντικατάσταση των συγκεκριμένων λειτουργιών του μονολιθικού συστήματος από τα αντίστοιχα microservices, η οποία μπορεί σταδιακά να επεκτείνεται Αυτή η προσέγγιση επιτρέπει τη σταδιακή μετάβαση προς την αρχιτεκτονική microservices χωρίς τον κίνδυνο μιας μαζικής αναδιοργάνωσης [57].

Ακόμη, η κατά Microsoft μεθοδολογία μπορεί να βασιστεί στο Domain Driven Design [58]. Η προσέγγιση της σχεδίασης με βάση τον τομέα προϋποθέτει την καλή γνώση αυτού, ενώ μπορεί να αποτελέσει και την αφετηρία για την αποσύνθεση ενός μονολιθικού συστήματος. Ωστόσο, απαιτείται το λεξιλόγιο να είναι κοινό και πλήρως κατανοητό για όλα τα εμπλεκόμενα μέρη και φυσικά να προσδιοριστούν με σαφήνεια τα μέρη της εφαρμογής που βασίζεται σε μονολιθική αρχιτεκτονική, τα οποία θα αποσυντεθούν. Η

αποσύνθεση θα υλοποιηθεί με βάση κάποια μοντέλα, τα οποία θα ενταχθούν σε ένα περιορισμένο περιβάλλον και θα επιβαρυνθούν με συγκεκριμένες ευθύνες.



Εικόνα 4. 2 Περιβάλλοντα στη μονολιθική αρχιτεκτονική

Πηγή: https://estia.hua.gr/file/lib/default/data/26493/theFile

Επίσης, μπορεί να διαμορφωθεί ένα επίπεδο παρουσίασης και αυτό να διαχωριστεί από το επίπεδο υποστήριξης. Τα APIs συνδέονται με το επίπεδο στο οποίο έχουν πρόσβαση τα δεδομένα και θέτουν ένα όριο ανάμεσα στο επίπεδο υποστήριξης και στο επίπεδο της επιχειρηματικής λογικής, στοιχείο που συμβάλλει στη διαδικασία της αποσύνθεσης, δηλαδή της μετάβασης. Από την άλλη πλευρά, ο μονόλιθος μπορεί να αποσυρθεί με σταδιακό τρόπο. Αυτή η διαδικασία μπορεί να επιτευχθεί με τη σταδιακή εξαγωγή των υπηρεσιών από τα οριοθετημένα περιβάλλοντα, ώστε να μεταφερθούν στα microservices. Γενικότερα, ανάλογα με τη μεθοδολογία διάφορα στοιχεία μιας εφαρμογής μπορεί να λειτουργήσουν ως βοηθητικά εργαλεία για τη διαδικασία της μετάβασης. Οι πρακτικές αυτές άλλωστε συμβάλλουν στην αποτελεσματική και επιτυχή αποσύνθεση, προκειμένου η νέα μορφή της εφαρμογής ή του συστήματος να είναι λειτουργική.

4.3 Σύγκριση αρχιτεκτονικών

Τα τελευταία χρόνια, υπάρχει μια τάση στην κοινότητα των μηχανικών λογισμικού προς το cloud computing. Οι πλατφόρμες cloud κερδίζουν την επικρατούσα υιοθέτηση ως το προτιμώμενο μοντέλο παράδοσης και λειτουργίας για σύγχρονες εφαρμογές από

διάφορες γνωστές και παγκόσμιας κλίμακας εταιρείες. Οι μεταβαλλόμενες συνθήκες υποδομής οδηγούν σε αρχιτεκτονικά στυλ που εκμεταλλεύονται τις ευκαιρίες που δίνουν οι υποδομές cloud. Η Αρχιτεκτονική Microservices επιτρέπει την αξιοποίηση των πλεονεκτημάτων που προκύπτουν από το cloud computing, για αυτό και υιοθετείται πιο συχνά [59].

Οι μικροϋπηρεσίες μπορούν να θεωρηθούν ως μια τεχνική για την ανάπτυξη εφαρμογών λογισμικού που, κληρονομώντας αρχές και έννοιες από το στυλ Service-Oriented Architecture (SOA), επιτρέπουν τη δομή μιας εφαρμογής που βασίζεται σε υπηρεσίες ως μια συλλογή από πολύ μικρές υπηρεσίες λογισμικού χαλαρά συζευγμένες [60].

Βασικά χαρακτηριστικά του MSA είναι τα περιορισμένα περιβάλλοντα, η ευελιξία και η αρθρωτή δομή. Από αυτή την άποψη, οι μικροϋπηρεσίες είναι μια νέα τάση στην αρχιτεκτονική λογισμικού, η οποία δίνει έμφαση στην ανάπτυξη και τον σχεδιασμό λογισμικού υψηλής δυνατότητας συντήρησης και κλιμάκωσης. Από την άλλη πλευρά, η μονολιθική αρχιτεκτονική ήταν η παραδοσιακή προσέγγιση για την ανάπτυξη λογισμικού, που χρησιμοποιήθηκε στο παρελθόν από μεγάλες εταιρείες. Στο MSA, οι συναρτήσεις ενσωματώνονται σε μια ενιαία εφαρμογή. Η εφαρμογή Monolith, αν δεν είναι περίπλοκη, έχει τη δική της δύναμη, για παράδειγμα, την ευκολία ανάπτυξης, δοκιμής και ανάπτυξης. Ωστόσο, όταν η εφαρμογή τείνει να γίνει πιο περίπλοκη, η μονολιθική δομή μεγαλώνει σε μέγεθος, καθιστώντας ένα μεγάλο, δύσκολο στη διαχείριση και κλίμακα λογισμικό [59].

Οι αδυναμίες αυτού του συστήματος εμφανίζονται όταν ο μονόλιθος αρχίζει να ενημερώνεται και να εξελίσσεται, αυξάνοντας μέγεθος και αριθμός λειτουργιών. Για παράδειγμα, εάν απαιτείται περισσότερη υπολογιστική ικανότητα, η κλιμάκωση ενός μεγάλου μονόλιθου θα οδηγήσει σε μεγαλύτερο γενικό κόστος με σεβασμό σε ένα μικρό μονόλιθο. Ένα άλλο μειονέκτημα που προκαλείται από το μεγάλο μέγεθος του μονόλιθου είναι ότι η ανάπτυξη μπορεί να επιβραδυνθεί και, κατά συνέπεια, να λειτουργήσει ως εμπόδιο στη συνεχή ανάπτυξη, λόγω του μεγαλύτερου χρόνου εκκίνησης [44].

Ένα βασικό πλεονέκτημα των μικροϋπηρεσιών είναι η δυνατότητα συντήρησης. Η διάσπαση ενός συστήματος σε ανεξάρτητες και αυτο-αναπτυσσόμενες υπηρεσίες βοηθά τις ομάδες προγραμματιστών να δοκιμάσουν τις υπηρεσίες τους και να κάνουν αλλαγές ανεξάρτητα από άλλους προγραμματιστές, απλοποιώντας την κατανεμημένη ανάπτυξη. Έτσι, το μικρό μέγεθος των μικροϋπηρεσιών συμβάλλει στην αύξηση της κατανοητότητας του κώδικα, βελτιώνοντας επίσης τη δυνατότητα συντήρησής του [61].

Ένα άλλο βασικό βασικό πλεονέκτημα των μικροϋπηρεσιών είναι η επεκτασιμότητα. Οι μικροϋπηρεσίες δεν είναι αυτόματα επεκτάσιμες, ακόμα κι αν αναπτύσσονται συνήθως σε αρχιτεκτονικές χωρίς κατάσταση. Ωστόσο, σε ένα MSA, κάθε microservice μπορεί να αναπτυχθεί σε διαφορετικά μηχανήματα, το καθένα με διαφορετικά επίπεδα απόδοσης, και μπορεί να γραφτεί στην πιο κατάλληλη γλώσσα προγραμματισμού. Εάν, για παράδειγμα, υπάρχει ένα σημείο συμφόρησης σε μια συγκεκριμένη μικρουπηρεσία, μπορεί να μεταφερθεί σε κοντέινερ και να εκτελεστεί σε πολλούς κεντρικούς υπολογιστές που βρίσκονται παράλληλα, χωρίς να χρειάζεται να αναπτυχθεί το σύστημα σε μια νέα και ισχυρή μηχανή [62].

Διάφοροι οργανισμοί ανακατασκευάζουν τα υπάρχοντα μονολιθικά τους συστήματα με αρχιτεκτονική μικροϋπηρεσιών. Ωστόσο, η εκ νέου αρχιτεκτονική ολόκληρου του συστήματος μπορεί να φέρει κάποιες προκλήσεις [63].

Η απόδοση είναι ένα από τα πιο σημαντικά μέρη της αρχιτεκτονικής, ωστόσο, υπάρχουν και άλλα τυπικά πλεονεκτήματα και μειονεκτήματα της χρησιμοποιούμενης αρχιτεκτονικής. Η μονολιθική αρχιτεκτονική είναι εύκολη στην ανάπτυξη, και απλή όμως χαρακτηρίζεται από σύνθετη συντήρηση. Όσον αφορά την αξιοπιστία, ένα σφάλμα μπορεί να καταρρίψει ολόκληρη την εφαρμογή, ενώ αναφορικά με τη διαθεσιμότητα κάθε ενημέρωση απαιτεί την ανανέωση ολόκληρης της εφαρμογής σε συνδυασμό με το χαρακτηριστικό της δύσκολης κλίμακας [64].

Από την άλλη πλευρά, η αρχιτεκτονική Microservice έχει εύκολη συντήρηση, μπορεί να γίνει ευκολότερα κατανοητή από έναν προγραμματιστή, καθώς κάθε βασική λειτουργικότητα είναι μια ξεχωριστή ενότητα, ενώ το σφάλμα επηρεάζει μόνο μια υπηρεσία και έτσι δεν θέτει υπό αμφισβήτηση την αξιοπιστία του συστήματος [64]. Επίσης, η αναδιάταξη μιας νέας έκδοσης μιας microservice απαιτεί μικρό χρόνο διακοπής λειτουργίας σε σύγκριση με τη μονολιθική αρχιτεκτονική και είναι εύκολη στην κλίμακα. Βέβαια, στα μειονεκτήματά της συγκαταλέγονται πως αναπτύσσεται με πιο πολύπλοκο τρόπο και ακόμη δεν διαθέτει αυτονομία [65].

Επίσης, η ερευνητική μελέτη των [66] έδειξε ότι για να αποφευχθούν τα προβλήματα των μονολιθικών εφαρμογών και να υπάρχει όφελος από ορισμένα από τα πλεονεκτήματα της αρχιτεκτονικής SOA, το μοτίβο αρχιτεκτονικής microservice έχει αναδειχθεί ως ένα ελαφρύ υποσύνολο του μοτίβου αρχιτεκτονικής SOA. Αυτό το μοτίβο χρησιμοποιείται από μεγάλες και πολυεθνικές εταιρείες για την υποστήριξη και την κλιμάκωση των εφαρμογών

και των προϊόντων τους. Το πρότυπο μικροϋπηρεσιών προτείνει τη διαίρεση μιας εφαρμογής A σε ένα σύνολο υπηρεσιών μικρών επιχειρήσεων μS (μS1, μS2, μSn), καθεμία από τις οποίες προσφέρει ένα υποσύνολο των υπηρεσιών S (S1, S2, Sx) που παρέχονται από την εφαρμογή A.

Η ψηφιοποίηση στο πλαίσιο του Industry 4.0 θεωρείται ο μεγαλύτερος και ταχύτερος μοχλός αλλαγής στην ιστορία της μεταποιητικής βιομηχανίας. Ενώ το μέγεθος μιας εταιρείας γίνεται λιγότερο σημαντικό, η ικανότητα γρήγορης προσαρμογής στις μεταβαλλόμενες συνθήκες της αγοράς και στις νέες τεχνολογίες είναι πιο σημαντική από ποτέ. Αυτή η τάση ισχύει ιδιαίτερα για τα τοπία λογισμικού των εταιρειών, όπου μεμονωμένες υποδιεργασίες και υπηρεσίες πρέπει να ενορχηστρωθούν, να ενσωματωθούν απρόσκοπτα και να ανανεώνονται επαναληπτικά σύμφωνα με τις συνεχώς αυξανόμενες απαιτήσεις των χρηστών. Ωστόσο, στον τομέα της μηχανικής κυριαρχούν άκαμπτες, κλειστές μονολιθικές εφαρμογές λογισμικού, καθώς και αυτοπρογραμματιζόμενα αυτόνομα εργαλεία που είναι δύσκολο να ενσωματωθούν. Η πλήρης επανεφαρμογή υφιστάμενων ιδιόκτητων εργαλείων μηχανικής και η ενσωμάτωσή τους σε μονολιθικές εφαρμογές μεγάλων παρόχων λογισμικού συχνά δεν είναι οικονομικά εφικτή, ειδικά για μικρομεσαίους κατασκευαστές μηχανημάτων και εγκαταστάσεων. Σε αυτό το πλαίσιο, η λεγόμενη Προσέγγιση που βασίζεται στη Διαδικασία (PDA) προσφέρει μια βιώσιμη και ουδέτερη από πλευράς εργαλείου ευκαιρία για ενορχήστρωση διεργασιών και εργαλείων, επιτρέποντας την εύκολη ενσωμάτωση μεμονωμένων εφαρμογών λογισμικού με συνεπή χρήση της αρχής του διαχωρισμού των ανησυχιών. Το PDA, που προέρχεται από την επιχειρηματική πληροφορική, βασίζεται κυρίως στην τυποποιημένη και εκτελέσιμη από μηχανή γλώσσα οπτικής μοντελοποίησης Business Process Model and Notation (BPMN). Χρησιμοποιώντας τις σημασιολογικές βελτιώσεις που βρέθηκαν στην έκδοση 2.0, το BPMN δεν χρησιμοποιείται μόνο για τη μοντελοποίηση των επιχειρηματικών διαδικασιών αλλά και για τη μοντελοποίηση και εκτέλεση των διαδικασιών ολοκλήρωσης μεταξύ διαφορετικών συστημάτων [67].

Κάθε microservice αναπτύσσεται χρησιμοποιώντας ανεξάρτητες βάσεις κωδικών και η ομάδα μTi είναι επίσης υπεύθυνη για την ανάπτυξη, την κλιμάκωση και τη λειτουργία της microservice σε μια λύση IaaS/PaaS υπολογιστικού νέφους. Μία από τις ανησυχίες της υλοποίησης μικρουπηρεσιών σχετίζεται με τις προσπάθειες που απαιτούνται για την ανάπτυξη και την κλιμάκωση κάθε μικρουπηρεσίας/πύλης στο cloud. Αν και οι εταιρείες

που εφαρμόζουν μικροϋπηρεσίες μπορούν να χρησιμοποιήσουν διαφορετικά εργαλεία αυτοματισμού DevOps, όπως Docker, Chef, Puppet, αυτόματη κλιμάκωση, μεταξύ άλλων, η υλοποίηση αυτών των εργαλείων καταναλώνει χρόνο και πόρους [68].

Και οι δύο δοκιμασμένες αρχιτεκτονικές έχουν ορισμένα πλεονεκτήματα και μειονεκτήματα. Όλα εξαρτώνται από το πρόβλημα που πρέπει να λυθεί. Η αρχιτεκτονική Microservice είναι πιο αποτελεσματική, εάν μια εφαρμογή πρέπει να χειριστεί μεγαλύτερο αριθμό αιτημάτων. Έχει πολλά πλεονεκτήματα που επιτρέπουν τη δημιουργία ενός λογισμικού υψηλής ποιότητας, το οποίο είναι εύκολο στην κλίμακα, πιο αξιόπιστο και μακροπρόθεσμα πιο βολικό στη συντήρηση. Αν και έχει τόσα κέρδη, η μονολιθική αρχιτεκτονική μπορεί να είναι εξίσου αποδοτική. Είναι πιο αποτελεσματική σε χαμηλότερο φορτίο και είναι εύκολο να αναπτυχθεί. Δεν υπάρχουν τόσα πολλά προβλήματα με την ενοποίηση, τη σύνδεση και τη διαμόρφωση. Η επιλογή της σωστής αρχιτεκτονικής θα πρέπει να καθορίζεται από τους σκοπούς της επιχείρησης, έτσι ώστε ο επενδυτής να πάρει το προϊόν που θα ανταποκρίνεται στις προσδοκίες του [64].

4.3.1 Διαφορές μεταξύ αρχιτεκτονικών

Με τις σημερινές μεγάλες τεχνολογικές προόδους και για να παραμείνουν ανταγωνιστικές οι εταιρείες, πολλές από αυτές τις εφαρμογές πρέπει να μετεγκατασταθούν ή/και να εκσυγχρονιστούν για να επωφεληθούν από τις τρέχουσες τεχνολογίες. Η διαδικασία μετάβασης από μια μονολιθική εφαρμογή σε μια μικρουπηρεσία είναι μια διαδικασία πρόκλησης και υπάρχουν λίγες αναφερόμενες επιτυχημένες εμπειρίες [65].

Σύμφωνα με τους [69], παρατηρούνται σημαντικές διαφορές μεταξύ της μονολιθικής αρχιτεκτονικής και της αρχιτεκτονικής των microsevices. Αρχικά, όσον αφορά τη διαθεσιμότητά τους στην αγορά, οι εφαρμογές που βασίζονται σε μονολιθική αρχιτεκτονική, εμφανίζονται σε σύντομο χρονικό διάστημα, έπειτα όμως ο χρόνος αυτός αυξάνεται αναλογικά με την αύξηση του κώδικα βάσης. Αντιθέτως, οι microservices αργούν σχετικά να εμφανιστούν στην αγορά εξαιτίας τεχνικών προβλημάτων που μπορεί να προκύψουν, αλλά στις μετέπειτα εκδόσεις τους, ο χρόνος αυτός μειώνεται καθώς υπάρχει πλέον εμπειρία ως προς τη διαχείριση των προβλημάτων που είναι πιθανό να ανακύψουν.

Σχετικά με την αναδιάρθρωση του κώδικα, στη μονολιθική αρχιτεκτονική οι αλλαγές επηρεάζουν ολόκληρο το σύστημα και κατά συνέπεια τις λειτουργίες του και έτσι η επίτευξη αυτής της διαδικασίας κρίνεται δύσκολη. Από την άλλη πλευρά, στην αρχιτεκτονική των microservices η αναδιάρθρωση είναι σχετικά γρήγορη και χαρακτηρίζεται από ασφάλεια, καθώς λαμβάνει χώρα μόνο σε μια υπηρεσία του συστήματος. Παρομοίως, σε περίπτωση αλλαγών, σε μονολιθικές εφαρμογές ο νέος κώδικας αφορά το σύστημα στο σύνολό του, ενώ στα microservices σχετίζεται μόνο με μία υπηρεσία, γεγονός που καθιστά το σύστημα πιο λειτουργικό. Το ίδιο παρατηρείται και αναφορικά με τη γλώσσα ανάπτυξης, η οποία στη περίπτωση των μονολιθικών εφαρμογών είναι πιο δύσκολο να αλλάξει, γιατί προϋποθέτει την αλλαγή ολόκληρου του συστήματος, ενώ στα microservices η γλώσσα μπορεί να ποικίλει ανά υπηρεσία. Έτσι, ο κώδικας είναι σαφώς μικρότερος, επηρεάζει ένα μόνο μέρος του συστήματος κάθε φορά και επομένως η μεταβολή του καθίσταται πιο εύκολη [10].

Επίσης, ενώ η κλιμάκωση στα microservices μπορεί να πραγματοποιηθεί ακόμα και για μία μόνο υπηρεσία, σε μονολιθικές εφαρμογές δημιουργούνται πολλά αντίτυπα και χρειάζεται έκδοση στην παραγωγή [69]. Έτσι, η τεχνολογία αυτή μπορεί πολλές φορές να μην έχει τη μεγαλύτερη απόδοση, σε αντίθεση με την αρχιτεκτονική των microservices, όπου ενδέχεται να χρησιμοποιείται ένας συνδυασμός τεχνολογιών, κατάλληλων για κάθε υπηρεσία ξεχωριστά, ώστε να συμβάλλουν στη συνολική της απόδοση.

Ακόμη, στη μονολιθική αρχιτεκτονική οι τεχνολογίες έχουν περιορισμένο εύρος δυνατοτήτων, για αυτό και οι DevOps ικανότητες δεν χρειάζεται να είναι εξειδικευμένες, σε αντίθεση με τα microservices, όπου η συμμετοχή διαφορετικών τεχνολογιών αυξάνει την απαίτηση της εξειδίκευσης. Τέλος, η λειτουργικότητα σε μονολιθικές εφαρμογές περιλαμβάνεται στον κώδικα βάσης, έτσι όποια ενδεχόμενη αλλαγή επιδρά σε όλο το σύστημα. Αντιθέτως, στα microservices ο κώδικας οι υπηρεσίες ακολουθούν την Αρχή της Μοναδικής Αρμοδιότητας και ο κώδικας είναι αρθρωτός, επομένως πιο εύκολα αντιληπτός [69].

Κεφάλαιο 5 Εργαλεία

5.1 Azure Service Fabric

Το Azure είναι μια πλατφόρμα που χτίστηκε από την αρχή χρησιμοποιώντας μια αρχιτεκτονική microservice [70]. Το Azure Service Fabric είναι ένα πλαίσιο εφαρμογής που παρέχεται από τη Microsoft ως υπηρεσία υποδομής υπολογιστικού νέφους στο Microsoft Azure. Έχει σχεδιαστεί για τη δημιουργία, την εκτέλεση και τη διαχείριση κλιμακούμενων και αξιόπιστων εφαρμογών, καθώς και τη διαχείριση των υπηρεσιών τους.

Αρχικά, το Service Fabric διαθέτει ενσωματωμένες λειτουργίες για τη διανομή και την κλιμάκωση εφαρμογών σε πολλαπλές υπολογιστικές πόρτες, είτε σε τοπικά κέντρα δεδομένων είτε στο νέφος. Υποστηρίζει την εκτέλεση υπηρεσιών εφαρμογής σε containers, όπως τα Docker containers, και παρέχει διαχείριση και οριζόντια κλιμάκωση για αυτά. Ταυτόχρονα, το Service Fabric παρέχει αυτόνομη διαχείριση και ανάκτηση από σφάλματα. Εάν ένα τμήμα της εφαρμογής αντιμετωπίσει πρόβλημα, το σύστημα μπορεί να προσπαθήσει αυτόματα να ανακτήσει την κατάσταση [71]. Το Azure επιτρέπει να οριστούν δοχεία εφαρμογών που παρέχουν ελαστική κλίμακα και ανοχή σφαλμάτων και διαχειρίζονται πλήρως μέσω αυτοματισμού.

Το Azure Service Fabric υποστηρίζει τη διαχείριση της κατάστασης των εφαρμογών, επιτρέποντας την αποθήκευση και τη διαμοίραση κατάστασης μεταξύ των διάφορων υπηρεσιών. Επιπλέον, παρέχει εργαλεία ανάπτυξης και διαχείρισης που ενσωματώνονται με τα εργαλεία ανάπτυξης της Microsoft, όπως το Visual Studio., ενώ υποστηρίζει πολλές γλώσσες προγραμματισμού, συμπεριλαμβανομένων των .NET, Java και Node.js [72].

Το Azure Service Fabric στοχεύει στη δημιουργία και την εκτέλεση υψηλής κλιμακούμενης και αξιόπιστης εφαρμογής, προσφέροντας πολλές λειτουργίες για τη διαχείριση και την ανάπτυξη εφαρμογών μειώνοντας τον χρόνο και την πολυπλοκότητα του κύκλου ζωή τους των εφαρμογών [73]. Το Azure της Microsoft είναι το τυπικό στάδιο διαμόρφωσης με κύκλους που προσελκύει τον πελάτη για αποστολή και εργασία στοιχείων με ταχύτητα προσαρμοστικότητας [74].

Επίσης, η μελέτη των [75] σημειώνει πως η Service Fabric (SF), είναι η κατανεμημένη πλατφόρμα της Microsoft για τη δημιουργία, την εκτέλεση και τη συντήρηση εφαρμογών

microservice στο cloud. Η SF λειτουργεί στην παραγωγή για 10+ χρόνια, τροφοδοτώντας πολλές κρίσιμες υπηρεσίες στη Microsoft.

Συνοπτικά, το Azure Service Fabric είναι η εφαρμογή της Microsoft μιας αρχιτεκτονικής microservice. Οι Microservices είναι ένα διαφορετικό αρχιτεκτονικό στυλ από το XX. Το Azure Service Fabric είναι μια πλατφόρμα κατανεμημένων συστημάτων που διευκολύνει τη συσκευασία, την ανάπτυξη και τη διαχείριση επεκτάσιμων και αξιόπιστων μικροϋπηρεσιών και κοντέινερ. Το Service Fabric αντιμετωπίζει επίσης τις σημαντικές προκλήσεις στην ανάπτυξη και διαχείριση εφαρμογών εγγενών στο cloud. Οι προγραμματιστές και οι διαχειριστές μπορούν να αποφύγουν πολύπλοκα προβλήματα υποδομής και να επικεντρωθούν στην εφαρμογή κρίσιμων για την αποστολή, απαιτητικών φόρτων εργασίας που είναι κλιμακωτοί, αξιόπιστοι και διαχειρίσιμοι. Το Service Fabric είναι μια πλατφόρμα επόμενης γενιάς για τη δημιουργία και τη διαχείριση αυτών των εφαρμογών εταιρικής κλάσης, επιπέδου πρώτου, σε κλίμακα cloud που εκτελούνται σε κοντέινερ [76].

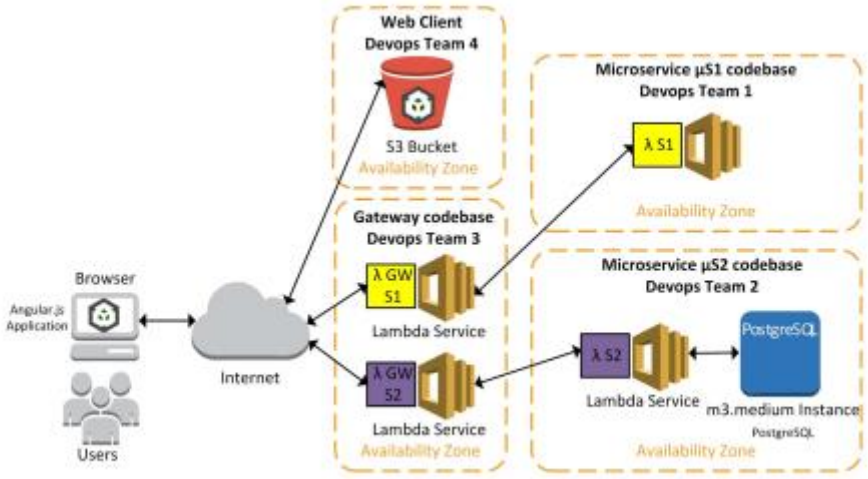
5.2 AWS Lambda

Η AWS Lambda είναι μια υπηρεσία της Amazon, η οποία δίνει τη δυνατότητα προγραμματισμού, δίχως όμως να χρειάζεται η διαχείριση server. Αυτή η υπηρεσία δεν προχωρά στην εκτέλεση του κώδικα κατά την αδράνειά του, αλλά μόνο σε περιπτώσεις που είναι απαραίτητο. Επομένως, κατά αυτόν τον τρόπο εξοικονομεί υπολογιστικούς πόρους, ενώ παράλληλα έχει υπό την ευθύνη της και όλες τις άλλες διαδικασίες, με συνέπεια ο προγραμματιστής να έχει ως κύριο μέλημά του τη σύνταξη του κώδικα μόνο όμως με τις γλώσσες προγραμματισμού, που υποστηρίζει η συγκεκριμένη υπηρεσία, δηλαδή την Python, τη Java, τη Go και τη C++ [77].

Αποτελεί μέρος της κατηγορίας των υπηρεσιών λειτουργιών ως υπηρεσία (Function as a Service - FaaS) και προσφέρει έναν απλό τρόπο για την εκτέλεση κώδικα χωρίς την ανάγκη να διαχειριστείτε την υποκείμενη υποδομή. Παράλληλα, η υπηρεσία AWS Lambda είναι κατάλληλη και για τη FaaS αρχιτεκτονική, γιατί η εκτέλεση του κώδικα μπορεί να γίνει ως απάντηση σε κάποιο συμβάν, μόνο όμως στο πλαίσιο των παροχών της Amazon. Έτσι, η AWS Lambda είναι ιδιαίτερα χρήσιμη για την ανάπτυξη εφαρμογών που απαιτούν

ευελιξία και κλιμακούμενη εκτέλεση χωρίς την προσπάθεια της διαχείρισης της υποδομής (Poccia, 2016).

Χαρακτηριστικό παράδειγμα της υπηρεσίας AWS Lambda στο πλαίσιο των microservices είναι η έρευνα των [66]. Στη μελέτη τους αυτή ανέπτυξαν μια αρχιτεκτονική microservice που λειτουργεί από το AWS Lambda, οι υπηρεσίες του οποίου αναπτύχθηκαν στο Node.js. Οι λειτουργίες microservice/gateway υλοποιήθηκαν σε τέσσερις ανεξάρτητες συναρτήσεις τύπου microservice-http-endpoint. Για την αποφυγή της κατανάλωσης χρόνου και πόρων, οι πάροχοι cloud όπως το AWS κυκλοφόρησαν πρόσφατα υπηρεσίες, όπως το AWS Lambda, το οποίο επιτρέπει την ανάπτυξη μικροϋπηρεσιών χωρίς την ανάγκη διαχείρισης διακομιστών. Αυτή η υπηρεσία έχει σχεδιαστεί για να προσφέρει μια δομή κόστους ανά αίτημα, που σημαίνει ότι οι προγραμματιστές πρέπει να ανησυχούν μόνο για τη σύνταξη μεμονωμένων συναρτήσεων για την υλοποίηση κάθε microservice και στη συνέχεια την ανάπτυξη τους στο AWS Lambda.



Εικόνα 5. 1 Ανάπτυξη του Amazon Web Services Lambda architecture

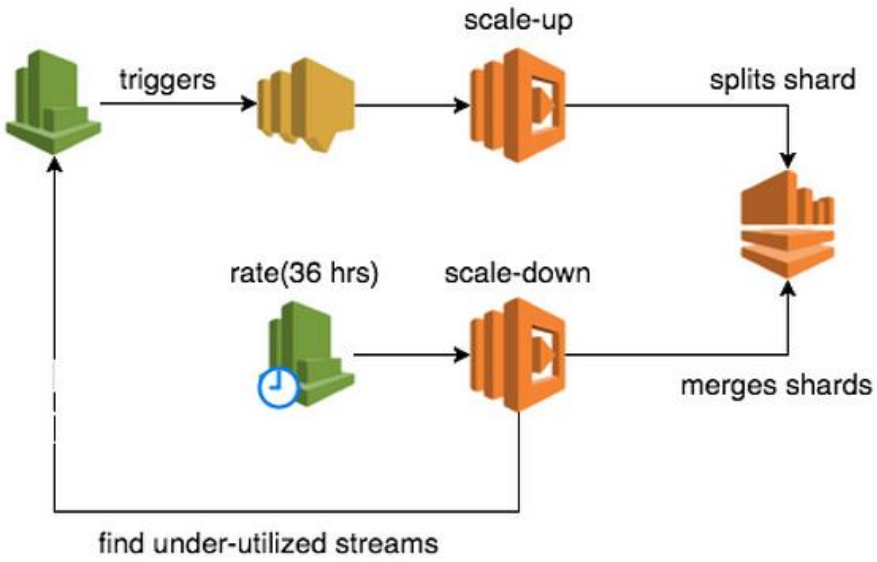
Πηγή: https://ieeexplore.ieee.org/abstract/document/7515686

5.2.1 Δυνατότητες της Lambda

Η Lambda είναι πρώτα από όλα μια υποστηρικτική υπηρεσία, η οποία επιτρέπει τις back-end υπηρεσίες, η εκτέλεση των οποίων θα γίνεται από τις παροχές της Amazon μέσω trigger. Αυτό το στοιχείο δίνει τη δυνατότητα να εξοικονομηθούν όχι μόνο πόροι αλλά και ενέργεια και μπαταρία, προκειμένου να διευκολυνθούν οι αναβαθμίσεις. Η Lambda

θεωρείται μια καινούρια υπηρεσία, η οποία όμως δεν συνεπάγεται μεγάλες αλλαγές ως προς τη γλώσσα προγραμματισμού, δεδομένου πως και πάλι μπορούν να χρησιμοποιηθούν οι βιβλιοθήκες που προτιμώνται κάθε φορά [78].

Η Lambda μπορεί να εκτελέσει τον κώδικα, να διαχειριστεί τις υποδομές, ενώ ταυτόχρονα μπορεί να συμβάλλει στην παρακολούθηση της εκτέλεσης με logs και μέσω του Amazon CloudWatch. Έτσι, ο προγραμματιστής δεν είναι πλέον υπεύθυνος για να αναβαθμίσει το λειτουργικό του χρησιμοποιούμενου μηχανήματος [79]. Επίσης, η παρούσα υπηρεσία έχει τη δυνατότητα να επεκτείνει τον κώδικα με αυτόματο τρόπο για να υπάρχει μια αποτελεσματική ανταπόκριση στις κλήσεις που έρχονται στον κώδικα, χωρίς να υπάρχει όριο. Ο κώδικας εκτελείται σε millisecond για ένα συμβάν, ενώ η επίδοση της εκτέλεσης είναι υψηλή αν και η συχνότητα των συμβάντων αυξάνεται. Επιπλέον, η χρήση υπηρεσιών που έχουν σχεδιαστεί ειδικά για την ανάπτυξη και την κλιμάκωση μικρουπηρεσιών, όπως το AWS Lambda, μειώνει το κόστος υποδομής κατά 70% ή περισσότερο, και σε αντίθεση με τις μικροϋπηρεσίες που λειτουργούν από πελάτες cloud, αυτές οι εξειδικευμένες υπηρεσίες συμβάλλουν στην εγγύηση της ίδιας απόδοσης και χρόνου απόκρισης με τους ο αριθμός των χρηστών αυξάνεται.



Εικόνα 5. 2 Scaling της Lambda

Πηγή: https://theburningmonk.medium.com/auto-scaling-kinesis-streams-with-aws-lambda-299f9a0512da

Η υπηρεσία AWS Lambda, λοιπόν, προσφέρει αρκετές παροχές για τη δημιουργία και την εκτέλεση λειτουργιών (functions) χωρίς την ανάγκη διαχείρισης της υποδομής του

υπολογιστικού περιβάλλοντος [80]. Η AWS Lambda αναλαμβάνει την εγκατάσταση και τη διαχείριση των απαραίτητων εξαρτήσεων. Ειδικότερα, η AWS Lambda ανταποκρίνεται δυναμικά στον όγκο της εισερχόμενης κίνησης. Καθώς ο αριθμός των εισερχομένων αιτημάτων αυξάνεται, η υπηρεσία αυτόματα αναπτύσσει νέα περιβάλλοντα εκτέλεσης (execution environments) για να αντιμετωπίσει το φορτίο. Επίσης, το κόστος αφορά μόνο τον χρόνο κατά τον οποίο εκτελείται η συνάρτηση και τους πόρους που χρησιμοποιεί. Δεν υπάρχουν, δηλαδή, χρεώσεις για ανενεργές περιόδους.

Ακόμη, μπορεί να συνδεθεί με διάφορες υπηρεσίες του AWS (όπως S3, DynamoDB, SNS) και να ανταποκρίνεται σε συμβάντα, εκτελώντας τις λειτουργίες όταν εκτυλίσσονται συγκεκριμένα γεγονότα. Παράλληλα, παρέχει ενσωματωμένα εργαλεία παρακολούθησης που επιτρέπουν τον έλεγχο της απόδοσης των λειτουργιών σας. Η AWS Lambda παρέχει μέσα ασφαλείας για περιορισμό των προνομίων, κρυπτογράφηση δεδομένων και ελέγχους πρόσβασης. Αυτές οι παροχές καθιστούν την AWS Lambda έναν ευέλικτο και αποτελεσματικό τρόπο για την υλοποίηση λειτουργιών χωρίς την ανάγκη ανησυχίας για την υποδομή [79].

5.3 Camunda framework

Το Camunda είναι ένα ανοιχτού κώδικα framework που παρέχει λογισμικό για τη διαχείριση διαδικασιών (BPM - Business Process Management). Συγκεκριμένα, το Camunda προσφέρει μια πλατφόρμα BPMN (Business Process Model and Notation) που επιτρέπει στους προγραμματιστές να σχεδιάζουν, να εκτελούν, και να εποπτεύουν διαδικασίες επιχειρηματικών διαδικασιών [81].

Συγκεκριμένα, παρέχει γραφικά εργαλεία για τη μοντελοποίηση επιχειρηματικών διαδικασιών χρησιμοποιώντας το πρότυπο BPMN, το οποίο είναι ένα βιομηχανικό πρότυπο για τη μοντελοποίηση διαδικασιών. Ταυτόχρονα, είναι σε θέση να εκτελεί τις διαδικασίες που έχουν οριστεί με το BPMN, επιτρέποντας την αυτοματοποίηση των επιχειρηματικών διαδικασιών, ενώ παρέχει δυνατότητες παρακολούθησης και εποπτείας για τις εκτελούμενες διαδικασίες, επιτρέποντας στους χρήστες να παρακολουθούν την πρόοδο και την απόδοση. Ωστόσο, η προδιαγραφή BPMN είναι μακρά και πολύπλοκη [82].

Το Camunda μπορεί να ενσωματωθεί σε διάφορα περιβάλλοντα, ενώ παράλληλα υποστηρίζει πολλές γλώσσες προγραμματισμού, όπως Java, JavaScript, Python κ.ά. Όσον αφορά το framework, αυτό είναι επεκτάσιμο και προσφέρει δυνατότητες προσαρμογής και επέκτασης των λειτουργιών του. Είναι ένα έργο ανοιχτού κώδικα, προσφέροντας πρόσβαση στον πηγαίο κώδικα και δυνατότητες κοινότητας για υποστήριξη και συνεισφορά [83]. Έτσι, το Camunda συνδυάζει τη δύναμη της αυτοματοποίησης διαδικασιών με την ευελιξία της προγραμματιστικής προσέγγισης, καθιστώντας το κατάλληλο για ποικίλες εφαρμογές, από απλές διαδικασίες έως πολύπλοκες επιχειρηματικές διαδικασίες.

5.3.1Δυνατότητες της Lambda

Η λειτουργία του CamundaService microservice εξαρτάται από τον τρόπο με τον οποίο έχει σχεδιαστεί και υλοποιηθεί. Συνήθως, όμως, ένα τέτοιο microservice θα χρησιμοποιεί το Camunda framework για την εκτέλεση και τη διαχείριση επιχειρησιακών διαδικασιών.
Οι λειτουργίες ενός microservice που χρησιμοποιεί το Camunda μπορεί να περιλαμβάνουν αρχικά τη μοντελοποίηση των διαδικασιών. Οι προγραμματιστές μπορούν να χρησιμοποιήσουν το Camunda για τη μοντελοποίηση των επιχειρησιακών διαδικασιών με χρήση του προτύπου BPMN. Το microservice μπορεί να εκτελεί τις διαδικασίες που έχουν καθοριστεί στο Camunda, αναλαμβάνοντας την αυτοματοποίηση και τη διαχείριση των εργασιών. Έχουν γίνει πολλές προσπάθειες για να επεκταθεί μια πολύ γνωστή γλώσσα μοντελοποίησης BP, όπως η BPMN, ώστε να ενσωματωθεί στις απαιτήσεις του IoT. Ωστόσο, όπως συμπεραίνουμε περαιτέρω στην ανάλυση της κατάστασης της τεχνολογίας, οι περισσότερες από τις υπάρχουσες λύσεις παρουσιάζουν δύο κύρια μειονεκτήματα: (1) οι επεκτάσεις που εισάγονται στο BPMN δεν υποστηρίζουν όλα τα εγγενή χαρακτηριστικά που παρουσιάζονται παραπάνω ή αυξάνουν πολύ την πολυπλοκότητα της μοντελοποίησης γλώσσα, η οποία εμποδίζει τη χρήση των μοντέλων BPMN ως εργαλείων επικοινωνίας μεταξύ των ενδιαφερομένων. (2) οι προτεινόμενες λύσεις για τη μοντελοποίηση BP ενισχυμένων με IoT δεν είναι εκτελέσιμες ή οι παρεχόμενοι μηχανισμοί εκτέλεσης εξαρτώνται σε μεγάλο βαθμό από συγκεκριμένες τεχνολογίες IoT, γεγονός που καθιστά δύσκολη την εξέλιξη του συστήματος όταν απαιτούνται τεχνολογικές αλλαγές [84].

Ωστόσο, η αρχιτεκτονική Microservices παρέχει υψηλό βαθμό αποσύνδεσης μεταξύ των δημιουργηθέντων μοντέλων και των υποκείμενων τεχνολογιών IoT. Αυτό διευκολύνει την ενσωμάτωση συσκευών IoT που υποστηρίζονται από διαφορετικές τεχνολογίες, ώστε να υπάρχει ευελιξία και εξοικονόμηση ενέργειας [85]. Επίσης, οι χρήστες μπορούν να παρακολουθούν την πρόοδο των διαδικασιών, ενώ το Camunda παρέχει εργαλεία για τη συλλογή στατιστικών δεδομένων. Μπορεί ακόμη να χρησιμοποιηθεί για τη διαχείριση κατάστασης και την αλληλεπίδραση μεταξύ των διάφορων εξαρτημένων υπηρεσιών. Ωστόσο, για να γίνει καλύτερα κατανοητή η λειτουργία του συγκεκριμένου microservice, θα πρέπει να ληφθεί υπόψη η τεκμηρίωση ή ο κώδικας που σχετίζεται με το συγκεκριμένο CamundaService.

5.4 API Gateway

Ο ρόλος της πύλης API (API gateway) είναι να αποτελεί τον διαμεσολαβητή μεταξύ των υπηρεσιών και των πελατών. Πρόκειται, δηλαδή, για έναν διακομιστή μεσολάβησης, ο οποίος λαμβάνει απαντήσεις μέσω των κλήσεων, τις συνθέτει και μετά τις επιστρέφει στις εφαρμογές πελάτη. Βέβαια, στην αρχιτεκτονική των Microservices ενδέχεται να υπάρχουν διαφορετικές εφαρμογές πελάτη, ανάμεσα στις οποίες βρίσκονται και οι κινητές συσκευές. Για αυτό με βάση τις απαιτήσεις της εκάστοτε εφαρμογής μπορεί να υπάρχει διαφορετική πύλη API κάθε φορά. Η πύλη είναι υπεύθυνη για διάφορες εργασίες, όπως την ανακάλυψη υπηρεσιών, τον συντονισμό των microservises, τον περιορισμό των αιτήσεων κ.ά. [86].

Σύμφωνα με την αρχιτεκτονική Micro-service, η πύλη API είναι ένα σημαντικό στοιχείο της συνολικής αρχιτεκτονικής. Αφαιρεί τις κοινές λειτουργίες που χρειάζονται στις μικροϋπηρεσίες. Ως η μόνη καταχώρηση για μια υπηρεσία Micro, η πύλη API ενσωματώνει τη συγκεκριμένη εσωτερική υλοποίηση και διεπαφή του συστήματος. Χαρακτηριστικά της, λοιπόν, είναι η εξισορρόπηση φορτίου και η αυτόματη εμφύσηση εξυπηρέτησης. Με τη χρήση της πύλης API, το πρόβλημα του τρόπου με τον οποίο ένας καλών μπορεί να καλέσει μια ανεξάρτητη υπηρεσία μπορεί να λυθεί, επομένως η αποτελεσματικότητα ανάπτυξης μπορεί να βελτιωθεί σημαντικά [87].

Έτσι, ορισμένα χαρακτηριστικά και λειτουργίες της πύλης API στα Microservices περιλαμβάνουν αρχικά τη δρομολόγηση. Όπως αναφέρθηκε, η πύλη API λαμβάνει τα αιτήματα από τους πελάτες και τα κατευθύνει στον κατάλληλο Microservice, το οποίο παρέχει την αντίστοιχη υπηρεσία. Συχνά, η πύλη API μπορεί να συγκεντρώνει δεδομένα από πολλαπλές υπηρεσίες για να τα ενώσει και να τα επιστρέψει ως ένα συνολικό αποτέλεσμα [88].

Επίσης, η πύλη API μπορεί να μετατρέπει τα δεδομένα και τα πρότυπα που χρησιμοποιούνται σε διάφορα Microservices. Παρέχει επιπλέον μηχανισμούς προστασίας και ασφαλείας, όπως πιστοποίηση, εξουσιοδότηση, κρυπτογράφηση κλπ., ενώ έχει τη δυνατότητα να διαχειρίζεται τη ροή της κίνησης, όπως τον περιορισμό του ρυθμού επεξεργασίας αιτημάτων για κάθε Microservice. Έπειτα, παρέχει δυνατότητες παρακολούθησης των αιτημάτων και των αποκρίσεων από τα Microservices και καταγράφει σημαντικά γεγονότα και δραστηριότητες για λόγους αποσφαλμάτωσης και ανάλυσης. Γενικά, η πύλη API βοηθάει στην απλοποίηση της διαχείρισης και της επικοινωνίας μεταξύ των Microservices, ενώ παράλληλα προσφέρει ασφάλεια, αποτελεσματική δρομολόγηση και άλλες σημαντικές λειτουργίες. Η πύλη API τρέχει μπροστά από οποιεσδήποτε μικροϋπηρεσίες και επεκτείνεται μέσω προσθηκών, οι οποίες παρέχουν επιπλέον λειτουργικότητα, όπως τον έλεγχο ταυτότητας JSON Web Token (JWT) [89].

5.5 Σύγκριση εργαλείων

Τα εργαλεία Azure Service Fabric, AWS Lambda, Camunda και API Gateway μπορούν να χρησιμοποιηθούν στην αρχιτεκτονική των Microservices και καλύπτουν διάφορες πτυχές του χώρου της ανάπτυξης και του διαχειριστικού περιβάλλοντος. Στη συνέχεια, παρουσιάζονται συνοπτικά τα βασικά χαρακτηριστικά κάθε εργαλείου, ώστε να διασαφηνιστούν μέσα από τη διαδοχική παράθεσή τους τόσο τα κοινά και τα διαφορετικά τους στοιχεία, όσο και τα πλεονεκτήματα και τα μειονεκτήματα που συνεπάγεται το καθένα.

1. Azure Service Fabric

Πρόκειται για υπηρεσία υποδομής υπολογιστικού νέφους για τη δημιουργία, την εκτέλεση και τη διαχείριση κλιμακούμενων και αξιόπιστων εφαρμογών. Είναι κατάλληλη για εφαρμογές που απαιτούν υψηλή κλιμακούμενη εκτέλεση και διαχείριση διαδικτυακών υπηρεσιών. Τα χαρακτηριστικά του είναι η αυτόματη κλιμάκωση και η υποστήριξη πολλών γλωσσών προγραμματισμού [90].

2. AWS Lambda

Είναι υπηρεσία Function as a Service (FaaS) που επιτρέπει την εκτέλεση κώδικα χωρίς την ανάγκη διαχείρισης υποδομής. Ενδείκνυται για εφαρμογές που απαιτούν ευελιξία και αυτόματη κλιμάκωση. Στα χαρακτηριστικά του συγκαταλέγονται η εκτέλεση κώδικα λειτουργία-προς-λειτουργία, η Αυτόματη κλιμάκωση και η χρέωση με βάση την πραγματική χρήση [79].

3. Camunda

Είναι ένα BPM (Business Process Management) framework για τη μοντελοποίηση, την εκτέλεση και τη διαχείριση επιχειρησιακών διαδικασιών. Κρίνεται κατάλληλο για επιχειρήσεις που χρειάζονται αυτοματοποίηση και διαχείριση διαδικασιών. Η μοντελοποίηση, λοιπόν είναι BPMN, είναι ανοιχτού κώδικα και μπορεί να εκτελέσει επιχειρησιακές διαδικασίες.

4. API Gateway

Είναι μια υπηρεσία πύλης (gateway) που διαχειρίζεται, δρομολογεί και παρακολουθεί τα αιτήματα σε ένα σύστημα. Μπορεί να αξιοποιηθεί για διαχείριση και έκθεση διασυνδέσεων (APIs) σε ένα σύστημα, να παρέχει ασφάλεια και να εξουσιοδοτεί, να παρακολουθεί και να κάνει αναφορές [91].

Σε ένα ευρύτερο πλαίσιο, το Azure Service Fabric, λοιπόν, έχει τη δυνατότητα να διαχειριστεί την κλιμάκωση και την ανάπτυξη των εφαρμογών microservices. Παρέχει υποστήριξη για τη δημιουργία, *-την εκτέλεση και τη διαχείριση κλιμακούμενων εφαρμογών, καθώς και υπηρεσίες εποπτείας και ανάλυσης [72]. Επίσης, μπορεί να επιτελεί αυτόματη αντικατάσταση εφαρμογών κατά τη διάρκεια αναβαθμίσεων. Από την άλλη πλευρά, το AWS Lambda επιτελεί την εκτέλεση κώδικα χωρίς τη διαχείριση υποδομής και δίνει τη δυνατότητα αυτόματης κλιμάκωση ανάλογα με τη ζήτηση [79].Πέρα από το γεγονός ότι υποστηρίζει πολλές γλώσσες προγραμματισμού, μπορεί να

συνεργαστεί με άλλες υπηρεσίες του AWS για πλήρη λύση εφαρμογής και να εκτελέσει λειτουργίες προς λειτουργία χωρίς τη διατήρηση συνεχούς υποδομής.

Όσον αφορά το Camunda συμβάλλει στη μοντελοποίηση, την εκτέλεση και τη διαχείριση επιχειρησιακών διαδικασιών, ενώ παρέχει ένα πλαίσιο BPMN για τη μοντελοποίηση διαδικασιών καθώς και εργαλεία εποπτείας και παρακολούθησης. Χαρακτηρίζεται από ευελιξία στη διαχείριση και εκτέλεση διαδικασιών, ενώ έχει ενσωματωμένη δυνατότητα ανάκαμψης από σφάλματα [92]. Το API Gateway συμβάλλει στην παρακολούθηση και τις αναφορές παράδοσης αλλά και στην προστασία και τη διαχείριση ασφαλείας για τις υπηρεσίες. Έχει δυνατότητα εξουσιοδότησης και διαχείρισης ταυτοποίησης, όπως και δυνατότητα μετασχηματισμού και προσαρμογής δεδομένων, ενώ μπορεί να ενσωματωθεί και με άλλες υπηρεσίες.

Κάθε εργαλείο έχει τις δικές του ισχυρές πτυχές και εφαρμογές. Είναι σημαντικό να σημειωθεί ότι τα εργαλεία αυτά έχουν διαφορετικές χρήσεις και προτιμώμενα σενάρια εφαρμογής και επομένως η επιλογή μεταξύ τους εξαρτάται από τις συγκεκριμένες ανάγκες του έργου ή της επιχείρησης, καθώς και από την προτίμηση στα χαρακτηριστικά και τις υπηρεσίες που προσφέρουν [91].

Κεφάλαιο 6 Τεχνολογίες μικροϋπηρεσιών

6.1 Γλώσσες Προγραμματισμού

Οι τεχνολογίες μικροϋπηρεσιών είναι ευέλικτες και υποστηρίζουν πολλές γλώσσες προγραμματισμού. Ο εκάστοτε στοίβος τεχνολογίας μπορεί να χρησιμοποιεί διάφορες γλώσσες ανάλογα με τις ανάγκες του έργου. Υπάρχουν διάφορες δημοφιλείς γλώσσες προγραμματισμού που χρησιμοποιούνται στον χώρο των μικρουπηρεσιών περιλαμβάνουν:

Αρχικά, η Java είναι μια από τις πιο δημοφιλείς γλώσσες προγραμματισμού για τις μικρουπηρεσίες. Η Java είναι μια ισχυρή, αντικειμενοστραφής γλώσσα προγραμματισμού που δημιουργήθηκε από την Sun Microsystems το 1995 [93]. Με δεκαετίες ιστορίας, η Java έχει γίνει ένα από τα πιο δημοφιλή εργαλεία ανάπτυξης λογισμικού σε παγκόσμιο επίπεδο. Υποστηρίζεται από μια εκτελεστική πλατφόρμα που επιτρέπει τη φορητότητα του κώδικα, καθιστώντας την κατάλληλη για εφαρμογές που λειτουργούν σε διάφορα περιβάλλοντα.

Η Java εφαρμόζει έναν αντικειμενοστραφή προγραμματισμό, επιτρέποντας στους προγραμματιστές να δομούν τον κώδικά τους με βάση αντικείμενα, τα οποία είναι περιοχές κώδικα που σχετίζονται με συγκεκριμένες λειτουργίες. Επίσης, αναλαμβάνει τη διαχείριση της μνήμης αυτόματα με το σύστημα του συλλέκτη σκουπιδιών, ελαχιστοποιώντας την ανάγκη παρέμβασης του προγραμματιστή, ενώ υποστηρίζει την πολυθρεαδικότητα με τη χρήση νημάτων, επιτρέποντας στις εφαρμογές να εκτελούν πολλαπλές εργασίες ταυτόχρονα. Ο κώδικας Java είναι φορητός, πράγμα που σημαίνει ότι μπορεί να εκτελεστεί σε διάφορες πλατφόρμες χωρίς να απαιτείται ανακατασκευή και η σύνταξη της συγκεκριμένης γλώσσας είναι σαφής και ομοιόμορφη, κάτι που διευκολύνει την εκμάθησή της, ενώ παράλληλα παρέχει ισχυρά εργαλεία για προχωρημένους προγραμματιστές [94].

Η Java έχει μια τεράστια και ενεργή κοινότητα προγραμματιστών που προσφέρει υποστήριξη, βιβλιοθήκες και πληροφορίες. Χρησιμοποιείται ευρέως για την ανάπτυξη εφαρμογών όπως ιστοσελίδες, εφαρμογές κινητών, συστήματα επιχειρηματικής

τεχνολογίας (enterprise systems) και πολλές άλλες, καθιστώντας τη μία από τις πιο επιτυχημένες γλώσσες προγραμματισμού στον κόσμο.

Από την άλλη πλευρά, η JavaScript χρησιμοποιείται για την ανάπτυξη μικρουπηρεσιών, κυρίως με τη χρήση του Node.js [95]. Με την αυξανόμενη δημοτικότητα του Ιστού, εμφανίστηκαν ορισμένες νέες τεχνολογίες Ιστού και εισήγαγαν δυναμική στις εφαρμογές Ιστού. Η JavaScript είναι η γλώσσα που παρείχε έναν δυναμικό ιστότοπο που επικοινωνεί ενεργά με τους χρήστες. Η JavaScript χρησιμοποιείται στις σημερινές διαδικτυακές εφαρμογές ως γλώσσα σεναρίου πελάτη και από την πλευρά του διακομιστή. Η γλώσσα JavaScript υποστηρίζει την αρχιτεκτονική του Ελεγκτή Προβολής Μοντέλου (MVC) που διατηρεί έναν αναγνώσιμο κώδικα και διαχωρίζει σαφώς τμήματα του κώδικα του προγράμματος [96].

Η γλώσσα προγραμματισμού JavaScript, επομένως, παίζει σημαντικό ρόλο στον τομέα των microservices, καθώς προσφέρει δυνατότητες για την ανάπτυξη ευέλικτων και αποδοτικών υπηρεσιών. Οι λειτουργίες της JavaScript στα microservices περιλαμβάνουν αρχικά την επιδεξιότητα στον Εντοπισμό του Κώδικα. Η JavaScript χρησιμοποιείται ευρέως για την ανάπτυξη εξυπηρετητών (servers) μέσω πλατφορμών όπως το Node.js. Οι εξυπηρετητές αυτοί είναι συχνά απαραίτητοι για τον εντοπισμό του κώδικα και την παροχή λειτουργιών σε αιτήσεις microservices. Στον πελάτη η JavaScript χρησιμοποιείται για τη δημιουργία δυναμικών και αλληλεπιδραστικών διεπαφών χρήστη σε ιστοσελίδες και εφαρμογές. Αυτό βοηθάει στη βελτίωση της εμπειρίας του χρήστη, ενώ μπορεί να ενσωματωθεί εύκολα με άλλες τεχνολογίες και γλώσσες προγραμματισμού, επιτρέποντας τη δημιουργία ολοκληρωμένων λύσεων μικρουπηρεσιών που χρησιμοποιούν πολλαπλές πλατφόρμες και υπηρεσίες [97].

Η υποστήριξη για ασύγχρονο προγραμματισμό μέσω Promises και των νέων async/await δυνατοτήτων στη γλώσσα επιτρέπει την αποτελεσματική διαχείριση εργασιών χωρίς να απαιτείται αποκλειστικός πόρος για κάθε αίτημα. Η JavaScript μπορεί να χρησιμοποιηθεί επίσης για την κατασκευή middleware, που προσφέρουν επιπλέον λειτουργικότητα όπως ασφάλεια, διαχείριση λογαριασμών και άλλα στο microservices αρχιτεκτονική. Η εύκολη σύνταξη της JavaScript και το μεγάλο οικοσύστημα των βιβλιοθηκών που προσφέρει καθιστούν την ανάπτυξη και συντήρηση των microservices πιο ευέλικτη [98]. Γενικά, η JavaScript συμβάλλει στη δημιουργία ευέλικτων, αναπτυσσόμενων και αποδοτικών

microservices, παρέχοντας ευκολία στον προγραμματισμό, αποδοτική επιδόση, και ικανότητα ανταπόκρισης σε αυξημένο φορτίο.

Επίσης, η Python είναι εξίσου μια γλώσσα προγραμματισμού με ευανάγνωστο κώδικα και ευελιξία, και χρησιμοποιείται συχνά για την ανάπτυξη της τεχνολογίας των μικρουπηρεσιών. Η γλώσσα προγραμματισμού Python αποτελεί μια ισχυρή, εύκολη στην εκμάθηση και ευέλικτη γλώσσα προγραμματισμού που έχει κερδίσει μεγάλη δημοτικότητα σε παγκόσμιο επίπεδο. Δημιουργήθηκε από τον Guido van Rossum και πρωτοεμφανίστηκε το 1991 [99] με τη σταθερή ανάπτυξη και εξέλιξή της να την καταστήσει μια από τις πιο ευρέως χρησιμοποιούμενες γλώσσες προγραμματισμού.

Ένα από τα κύρια χαρακτηριστικά της Python είναι η ευανάγνωστη σύνταξη της. Ο κώδικας γράφεται σε λίγες γραμμές και χρησιμοποιεί λέξεις-κλειδιά αγγλικής γλώσσας, κάνοντας τον κώδικα προσιτό σε νέους προγραμματιστές και εξαπλώνοντας τη χρήση της γλώσσας σε διάφορους κλάδους. Επίσης, η Python υποστηρίζει πολλαπλασιαστικούς παραδοτέους προγραμματισμού, αντικειμενοστραφή προγραμματισμό και λειτουργικό προγραμματισμό, προσφέροντας έτσι ευελιξία και δυνατότητα επαναχρησιμοποίησης του κώδικα [100]. Η Python διαθέτει μια ζωντανή και ενεργή κοινότητα προγραμματιστών, που συνεισφέρει σε πολλά έργα και βιβλιοθήκες. Αυτό επομένως διασφαλίζει συνεχή εξέλιξη και υποστήριξη.

Η γλώσσα προγραμματισμού Python είναι εξαιρετικά κατάλληλη για την ανάπτυξη και την υποστήριξη των microservices λόγω των πολλών της χαρακτηριστικών και των δυνατοτήτων της. Ειδικότερα, η Python προσφέρει συντακτική απλότητα και ευανάγνωστο κώδικα, καθιστώντας την ευέλικτη και ευκολοδιάβαστη. Αυτό το στοιχείο είναι σημαντικό όταν αναπτύσσονται και διαχειρίζονται πολλές υπηρεσίες. Εκτός των άλλων, η Python υποστηρίζει και αντικειμενοστραφή προγραμματισμό και λειτουργικό προγραμματισμό, παρέχοντας ευελιξία στους προγραμματιστές να επιλέγουν το κατάλληλο παραδειγματικό μοντέλο [101].

Η Python έχει μια τεράστια κοινότητα προγραμματιστών και ένα εκτεταμένο οικοσύστημα βιβλιοθηκών, που διευκολύνουν την ανάπτυξη και τη συντήρηση των microservices. Δηλαδή, παρέχει δομές δεδομένων όπως λίστες και σύνολα, καθιστώντας εύκολο τον χειρισμό και τη μεταφορά δεδομένων μεταξύ των υπηρεσιών. Οι βιβλιοθήκες όπως η asyncio και η aiohttp επιτρέπουν στους προγραμματιστές να αναπτύσσουν ασύγχρονες εφαρμογές, βελτιώνοντας την αποδοτικότητα σε περιβάλλοντα όπου ο χρόνος απόκρισης

είναι κρίσιμος [102]. Τέλος, η Python προσφέρει ενσωματωμένη υποστήριξη για τη δημιουργία και την κλήση των RESTful APIs, που αποτελεί συχνή πρακτική στον σχεδιασμό microservices. Πρόκειται επομένως για μια ισχυρή επιλογή για την ανάπτυξη microservices λόγω της ευελιξίας, της απλότητας, και της εκτενούς υποστήριξής της από την κοινότητα και το οικοσύστημα βιβλιοθηκών [103].

Η C# χρησιμοποιείται κυρίως στο πλαίσιο του .NET για την ανάπτυξη μικρουπηρεσιών σε περιβάλλον Microsoft και η Go (Golang) είναι σχεδιασμένη για την απλότητα και την αποτελεσματικότητα, ενώ έχει καλή υποστήριξη για τις μικρουπηρεσίες. Εξάλλου, τα χαρακτηριστικά της ευνοούν την καλή λειτουργία των μικρουπηρεσιών. Το C++ αντιπροσωπεύει μια ισχυρή και ευέλικτη γλώσσα προγραμματισμού που έχει εξελιχθεί από τη γλώσσα C, προσθέτοντας σε αυτήν την αντικειμενοστραφή προσέγγιση. Δημιουργήθηκε από τον Bjarne Stroustrup στις αρχές των '80s και έκτοτε έχει γίνει μια από τις πιο ευρέως χρησιμοποιούμενες γλώσσες στον χώρο της προγραμματιστικής κοινότητας [104].

Το C++ υποστηρίζει τον αντικειμενοστραφή προγραμματισμό, επιτρέποντας τη δημιουργία κλάσεων και αντικειμένων για τον οργανωμένο και αποτελεσματικό κώδικα. Πρόκειται για μια υψηλού επιπέδου γλώσσα προγραμματισμού που επιτρέπει την άμεση επικοινωνία με το υλικό, προσφέροντας συγχρόνως υψηλή απόδοση. Το C++ χρησιμοποιείται ευρέως για την ανάπτυξη λειτουργικών συστημάτων, συστηματικού προγραμματισμού και εφαρμογών που απαιτούν απόδοση και έλεγχο, ενώ μπορεί να μεταφερθεί σε διάφορες πλατφόρμες χωρίς την ανάγκη για σημαντικές τροποποιήσεις [105].

Το C++ επιτρέπει τον άμεσο έλεγχο της μνήμης, παρέχοντας τη δυνατότητα για χειροκίνητη διαχείριση της μνήμης όταν αυτό απαιτείται. Επιπλέον, η γλώσσα υποστηρίζει την παραλληλισμένη εκτέλεση με τη χρήση νημάτων (threads) και περιλαμβάνει ένα εκτεταμένο σύνολο βιβλιοθηκών που καλύπτουν ποικίλες ανάγκες, από γραφικές διεπαφές χρήστη μέχρι δομές δεδομένων και δικτυακή επικοινωνία [106].

Η C++ στο πλαίσιο των microservices πιτρέπει στους προγραμματιστές να έχουν άμεσο έλεγχο στις λειτουργίες του υπολογιστή. Αυτό σημαίνει ότι ο κώδικας που γράφεται σε C++ μπορεί να είναι αποδοτικός και να απαιτεί λιγότερους πόρους, κάτι που είναι σημαντικό για τα microservices που πρέπει να λειτουργούν αποδοτικά και να κλιμακώνονται εύκολα [107].

Οι σύγχρονες εκδόσεις της C++ περιλαμβάνουν σχεδιαστικά πρότυπα όπως οι "smart pointers" και ο "RAII (Resource Acquisition Is Initialization)," πράγμα που βοηθά στην αποφυγή διαρροών μνήμης και την εύκολη διαχείριση των πόρων. Ακόμη, η C++ μπορεί να συνδυαστεί εύκολα με άλλες γλώσσες προγραμματισμού, παρέχοντας έτσι ευελιξία στον σχεδιασμό του συστήματος. Αυτή η συνδεσιμότητα επιτρέπει τη χρήση της C++ για τα κομμάτια του συστήματος που απαιτούν υψηλή απόδοση και άμεση πρόσβαση στη μνήμη. Παράλληλα, προσφέρει ανεξαρτησία από την πλατφόρμα, επιτρέποντας τη μεταφορά του κώδικα μεταξύ διαφορετικών λειτουργικών συστημάτων χωρίς μεγάλες τροποποιήσεις [108].

Αυτές οι γλώσσες προγραμματισμού είναι μόνο μερικές από τις πολλές που μπορούν να χρησιμοποιηθούν στον χώρο των μικρουπηρεσιών, και η επιλογή εξαρτάται συχνά από τις προτιμήσεις των προγραμματιστών, τις απαιτήσεις του έργου, και τις δυνατότητες των εργαλείων και των πλατφορμών που χρησιμοποιούνται.

6.2 Weather microservice

Στις μέρες μας, η μοντελοποίηση προσομοίωσης είναι ένα σχετικό και πρακτικά σημαντικό μέσο στον τομέα για την έρευνα της λειτουργίας των αντικειμένων υποδομής. Η εφαρμογή μετεωρολογικών δεδομένων, σε αυτό το πλαίσιο, γίνεται σημαντικό ζήτημα. Μια τέτοια εφαρμογή υποστηριζόμενη από την τεχνολογία των microservices μπορεί να οργανώσει τη μοντελοποίηση προσομοίωσης σε ετερογενή κατανεμημένα υπολογιστικά περιβάλλοντα. Στο πλαίσιο της προτεινόμενης προσέγγισης, όλες οι λειτουργίες που σχετίζονται με την προετοιμασία δεδομένων, την εκτέλεση μοντέλων και την ανάλυση των ληφθέντων αποτελεσμάτων υλοποιούνται ως μικροϋπηρεσίες. Τα κύρια πλεονεκτήματα της προτεινόμενης προσέγγισης είναι ο υπολογισμός σάρωσης παραμέτρων στο πλαίσιο της μοντελοποίησης προσομοίωσης και η δυνατότητα ενσωμάτωσης πόρων των κέντρων υπερυπολογιστών δημόσιας πρόσβασης με πλατφόρμες cloud και fog. Με αυτή την εφαρμογή μπορούν να αξιολογηθούν οι μετεωρολογικές συνθήκες σε μια συγκεκριμένη περιοχή [109].

Τα μετεωρολογικά δεδομένα είναι σημαντικά για τον σωστό προσδιορισμό των ημερομηνιών έναρξης και λήξης της περιόδου θέρμανσης, πρόβλεψη κατανάλωσης

θερμότητας και ηλεκτρικής ενέργειας, προσομοίωση λειτουργίας εξοπλισμού κ.λπ. Στο πλαίσιο της μοντελοποίησης προσομοίωσης, παρέχεται μια πρόβλεψη καιρού σχεδόν σε πραγματικό χρόνο (Rasheed, San & Kvamsdal, 2020).

6.3 Email microservice

Ο κλάδος των υπηρεσιών μάρκετινγκ που βασίζονται σε ηλεκτρονικό ταχυδρομείο, ή το συντομότερο, "eMarketing", χαρακτηρίζεται από εμπόδια εισόδου στην αγορά, καθώς το προϊόν "email" δεν έχει αρκετή τεχνική πολυπλοκότητα για να επιτρέψει επαρκή διαφοροποίηση. Η προκύπτουσα ευρεία γκάμα ανταγωνιστών οδήγησε σε σταθερή πτώση των τιμών, γι' αυτό εδώ και αρκετά χρόνια, οι κορυφαίοι πάροχοι eMarketing διαφοροποιούν και παρέχουν ολοένα πιο εξατομικευμένες και ολοένα πιο σύνθετες υπηρεσίες στους πελάτες τους. Για να αποφευχθεί ένας ανταγωνισμός για την ηγεσία των τιμών με άγνωστο αποτέλεσμα, οι πάροχοι eMarketing προς το παρόν απομακρύνονται από τεχνικά ευθυγραμμισμένους παρόχους εξειδικευμένου λογισμικού για την αποστολή καμπανιών email, προς εταιρείες για τη δημιουργία εκστρατειών μάρκετινγκ πλήρους υπηρεσίας: από την παραγωγή ιδεών έως την παραγωγή κειμένου έως τη μετάφραση και την αξιολόγηση των σχολίων [110].

Μέσω του εργαλείου Email microservice υπάρχει η δυνατότητα να στέλνονται ειδοποιήσεις μέσα από ένα cloud-based service που είναι αξιόπιστο. Το χαρακτηριστικό αυτού του εργαλείου είναι ότι παρέχει στοιχεία με αναλυτικό τρόπο σε πραγματικό χρόνο, ενώ η έρευνα των [110] τονίζει ότι οι ad-hoc αλλαγές οδηγούν σε απώλεια σταθερότητας και απόδοσης.

Η έρευνα των [111] περιγράφει τις λειτουργίες μιας τέτοιας εφαρμογής. Ο συνδρομητής PubSub θα είναι υπεύθυνος για τη λήψη των μηνυμάτων για τον συνδρομητή. Αυτή η κλάση θα καλέσει την κλάση Messsage Receiver Impl για να λάβει το μήνυμα και θα επιστρέψει τις πληροφορίες συνδρομητή στην κύρια κατηγορία εφαρμογής. Ο δέκτης μηνυμάτων αναλαμβάνει τη λήψη του μηνύματος από το Cloud PubSub με τη μορφή JSON. Το μήνυμα JSON δηλαδή που θα λάβει θα περιέχει τις πληροφορίες, ενώ το JSON που ελήφθη θα μετατραπεί σε αντικείμενο Java για τα παραπάνω πεδία χρησιμοποιώντας αυτήν την κλάση. Ο αποστολέας email μπορεί να στείλει τόσο μαζικά όσο και μεμονωμένα

μηνύματα, η υπηρεσία επικύρωσης email είναι υπεύθυνη για την επικύρωση της μορφής email του παραλήπτη και θα αναφέρει σφάλμα εάν η μορφή είναι λανθασμένη. Θα χρησιμοποιήσει την κλάση Apache Email Validator για την επικύρωση της μορφής email. Η micro υπηρεσία email θα μπορεί να στείλει 1000 mail με μοναδικό μήνυμα σε καθένα από αυτά σε 8-10 δευτερόλεπτα. Αυτό θα μειώσει τον χρόνο, θα κάνει την υπηρεσία αυτοματοποιημένη και η χρήση της αρχιτεκτονικής μικροϋπηρεσιών θα μειώσει τις πολυπλοκότητες και τα προβλήματα που αντιμετωπίζει η μονολιθική αρχιτεκτονική. Χρησιμοποιώντας Rest API, η υπηρεσία μπορεί να χρησιμοποιηθεί από οπουδήποτε και με οποιαδήποτε εφαρμογή.

6.4 Data microservice

Η Microservice Architecture γίνεται πρότυπο σχεδιασμού για σύγχρονα συστήματα λογισμικού που βασίζονται σε cloud. Ωστόσο, η διαχείριση της επικοινωνίας δεδομένων παραμένει μια πρόκληση. Αυτό είναι ιδιαίτερα εμφανές κατά τη μετάβαση από ένα υπάρχον μονολιθικό σύστημα σε μικροϋπηρεσίες. Σημαντικό ρόλο, λοιπόν, διαδραματίζει ο συγχρονισμός των δεδομένων και η βελτίωση της απόδοσης της πηγής δεδομένων. Για την αντιμετώπιση του συγχρονισμού δεδομένων η έρευνα των [112] προτείνει τη χρήση ενός αυτοματοποιημένου συστήματος ροής δεδομένων μεταξύ βάσεων δεδομένων. Για τη βελτίωση της απόδοσης μιας πηγής δεδομένων, γίνεται εισαγωγή μιας λύσης με την κατανεμημένη κρυφή μνήμη.

Χαρακτηριστικό παράδειγμα αποτελεί το περιεχόμενο της έρευνας των [113]. Σε αυτή την έρευνα, παρουσιάζεται μια πλατφόρμα ανάλυσης μεγάλων δεδομένων που βασίζεται σε μικροϋπηρεσίες. Η πλατφόρμα αυτή βοηθά στην υλοποίηση της κοινής χρήσης δεδομένων, υπολογιστικής ισχύος και πόρων υποδομής. Επιπλέον, δίνει τη δυνατότητα στους χρήστες να χρησιμοποιούν ένα πιο φιλικό περιβάλλον για την εγγραφή και την κοινή χρήση της πειραματικής διαδικασίας και του οπτικοποιημένου μοντέλου. Χρησιμοποιεί το JupyterHub και την οπτικοποιημένη μοντελοποίηση ως δύο κύριες εφαρμογές του. Τα στοιχεία περιβάλλοντος υποδομής της πλατφόρμας είναι το στοιχείο μεγάλων δεδομένων και το περιβάλλον επιστήμης δεδομένων. Προκειμένου να προσαρμοστεί εύκολα στα

χαρακτηριστικά των ταχέων περιβαλλοντικών αλλαγών, η πλατφόρμα χρησιμοποιεί μικροϋπηρεσίες ως τεχνική εξάρτηση [114].Συγκεκριμένα, χρησιμοποιεί το Docker.

Γενικότερα, πολλά συστήματα που βασίζονται σε υπηρεσίες ακολουθούν το στυλ αρχιτεκτονικής microservice. Καθώς οι μικροϋπηρεσίες χρησιμοποιούνται για τη δημιουργία κατανεμημένων συστημάτων και την προώθηση αρχιτεκτονικών ιδιοτήτων όπως η ανάπτυξη ανεξάρτητων υπηρεσιών, οι στοίβες τεχνολογίας πολυγλωσσίας συμπεριλαμβανομένης της επιμονής πολυγλωσσίας και των χαλαρά συζευγμένων εξαρτήσεων, η αρχιτεκτονική διαχείριση δεδομένων είναι ζωτικής σημασίας στις περισσότερες αρχιτεκτονικές μικροϋπηρεσιών [115].Ωστόσο, στις βάσεις δεδομένων η γνώση είναι διάσπαρτη και πολλές φορές κατακερματισμένη. Για αυτό απαιτείται κανείς να διαθέτει επαρκή γνώση της συγκεκριμένης αρχιτεκτονικής, ώστε να μπορεί να την αξιοποιήσει κατάλληλα.

6.5 Σύγκριση τεχνολογιών

Τα παραπάνω εργαλεία επιτελούν διαφορετικές λειτουργίες και έχουν κατασκευαστεί για διαφορετικούς σκοπούς. Παρόλα αυτά και τα τρία, δηλαδή το Weather Microservices, το Email Microservices και το Data Microservices βασίζονται στην τεχνολογία των μικρουπηρεσιών.

Η πρώτη εφαρμογή παρέχει καιρικές πληροφορίες. Αυτό σημαίνει πως περιλαμβάνει υπηρεσίες που σχετίζονται με τη συλλογή, την επεξεργασία και την παρουσίαση καιρικών πληροφοριών, όπως θερμοκρασία, καιρικές συνθήκες, προγνώσεις κ.ά. [109]. Μια τέτοια εφαρμογή μπορεί να χρησιμοποιεί διαφορετικές υπηρεσίες, όπως RESTful APIs, χρήση προγραμματιστικών γλωσσών όπως Python, Java, ή Node.js, ενσωματωμένη χρήση βάσεων δεδομένων για αποθήκευση ιστορικών δεδομένων και προβλέψεων.

Στη συνέχεια, οι υπηρεσίες Αποστολής Email επιτρέπουν την αποστολή, παρακολούθηση και διαχείριση email μηνυμάτων [110]. Για αυτό το σύστημα είναι πιο κατάλληλες ορισμένες διαφορετικές τεχνολογίες σε σύγκριση με την υπηρεσία καιρικών πληροφορικών. Αυτές περιλαμβάνουν τη χρήση πρωτοκόλλων SMTP, ενσωμάτωση με email APIs (π.χ., SendGrid, SMTP.com), αλλά και τη χρήση γλωσσών όπως η Python, η Java, ή η Node.js.

Τέλος, τα Data Microservices περιλαμβάνουν υπηρεσίες που αφορούν τη συλλογή, αποθήκευση, επεξεργασία και παροχή δεδομένων. Είναι πιθανό να χρησιμοποιούνται βάσεις δεδομένων (SQL ή NoSQL), RESTful APIs για την ανταλλαγή δεδομένων με άλλες υπηρεσίες, γλώσσες προγραμματισμού όπως Java, Python, ή C++ [112]. Είναι εύλογο πως κάθε κατηγορία microservices έχει τις δικές της ξεχωριστές απαιτήσεις και προκλήσεις, και η επιλογή της κατάλληλης τεχνολογίας εξαρτάται από τις συγκεκριμένες ανάγκες και απαιτήσεις του έργου.

6.6 Ανάπτυξη microservices

Στην παρούσα ενότητα αναλύονται δύο παραδείγματα mocroservices που έχουν αναπτυχθεί στα πλαίσια της παρούσας διπλωματικής. Τα βασικά εργαλεία/εφαρμογές που χρησιμοποιήθηκαν είναι τα εξή:

- **Apache Kafka** είναι μια πλατφόρμα ανοιχτού κώδικα για την επεξεργασία και αποθήκευση ροών δεδομένων. Σχεδιασμένο για υψηλή διαθεσιμότητα, επεκτασιμότητα, και δυνατότητες ανάκτησης, ο Kafka επιτρέπει την αποδοτική μεταφορά δεδομένων μεταξύ συστημάτων και εφαρμογών σε πραγματικό χρόνο. Χρησιμοποιείται συχνά για την κατασκευή πλατφορμών δεδομένων, λογισμικού ολοκλήρωσης, και συστημάτων μηνυμάτων λόγω της αξιοπιστίας, ταχύτητας και ανθεκτικότητάς του.
- **CRUD** αναφέρεται στις βασικές λειτουργίες που απαιτούνται για την διαχείριση δεδομένων σε ένα σύστημα: Δημιουργία (Create), Ανάγνωση (Read), Ενημέρωση (Update), και Διαγραφή (Delete). Αυτές οι λειτουργίες αποτελούν τη βάση για την αλληλεπίδραση με τα δεδομένα σε οποιαδήποτε βάση δεδομένων ή αποθηκευτικό σύστημα. Η χρήση τους είναι κρίσιμη σε εφαρμογές που απαιτούν την διατήρηση, την ανάκτηση, την ενημέρωση, και την εξάλειψη δεδομένων, προσφέροντας έναν σταθερό και καθολικά κατανοητό σκελετό για την διαχείριση πληροφοριών.
- **CORS** (Cross-Origin Resource Sharing) είναι μια πολιτική ασφαλείας που επιτρέπει σε web εφαρμογές που εκτελούνται σε έναν browser να ζητούν πόρους από διαφορετική προέλευση από αυτήν που φορτώθηκε η εφαρμογή. Αυτό προστατεύει τον χρήστη από μαλικιούς ιστοτόπους που μπορεί να προσπαθήσουν

να διαβάσουν ευαίσθητα δεδομένα. CORS ρυθμίζεται μέσω HTTP headers που επιτρέπουν στον server να δηλώσει ποιες προελεύσεις επιτρέπονται να αξιοποιήσουν τους πόρους του.

- Μια **Spring Boot** εφαρμογή είναι ένα Java-based framework που απλοποιεί την ανάπτυξη και το deployment web εφαρμογών. Παρέχει ένα προκαθορισμένο σύνολο τεχνολογιών και αυτοματισμών για την γρήγορη δημιουργία αξιόπιστων, επεκτάσιμων και εύκολων στη διαχείριση εφαρμογών. Επικεντρώνεται στην "συμβατική πάνω από τη ρύθμιση" φιλοσοφία, μειώνοντας την ανάγκη για εκτεταμένες ρυθμίσεις.
- Το **Spring Framework** είναι ένα ισχυρό, ευέλικτο πλαίσιο (framework) για την ανάπτυξη εφαρμογών Java. Παρέχει ένα εκτενές προγραμματιστικό και διαμορφωτικό μοντέλο για σύγχρονες εφαρμογές Java - είτε αυτές είναι βασισμένες στο web, επιχειρηματικές εφαρμογές, μικρουπηρεσίες, είτε απλές standalone εφαρμογές. Προωθεί την ανάπτυξη με βάση τα POJOs (Plain Old Java Objects) προσφέροντας διαχείριση διεργασιών, διαχείριση των εξαρτήσεων μέσω Dependency Injection (DI) και πλούσια υποστήριξη για διαμόρφωση μέσω annotations ή XML. Επίσης, περιλαμβάνει υποστήριξη για την πρόσβαση σε δεδομένα, την ανάπτυξη web εφαρμογών, ασφάλεια, messaging και πολλά άλλα, κάνοντάς το ένα ολοκληρωμένο πλαίσιο για την ανάπτυξη σύγχρονων Java εφαρμογών.

6.6.1 emailHandler

Το έργο που αναπτύχθηκε είναι μια μικροϋπηρεσία που σχεδιάστηκε για να χειρίζεται ειδοποιήσεις μέσω email, ιδιαίτερα στο πλαίσιο των λειτουργιών του καλαθιού αγορών. Είναι χτισμένο με Spring Boot και τα βασικά στοιχεία που περιλαμβάνει είναι τα εξής:

1. EmailNotifierApplication: Η κύρια κλάση που εκκινεί την εφαρμογή Spring Boot.
2. Consumer: Ένας Kafka listener κάνει consums τα μηνύματα από ένα Kafka topic, επεξεργάζεται τα δεδομένα και ενεργοποιεί ειδοποιήσεις μέσω email με βάση το "ShoppingCartModel".

3. MailTemplateBuilder: Κατασκευάζει μηνύματα ηλεκτρονικού ταχυδρομείου χρησιμοποιώντας πληροφορίες καλαθιού αγορών για τη δημιουργία εξατομικευμένου περιεχομένου για ειδοποιήσεις.

4. ServletInitializer: Επιτρέπει στην εφαρμογή Spring Boot να αναπτυχθεί ως traditional WAR αρχείο σε ένα web container, αντί να εκτελείται ως αυτόνομη εφαρμογή.

5. MsgListener Config: Ρυθμίζει τους Kafka listeners ώστε να κάνουν consum τα μηνύματα που έχουν γίνει σειριακά (serialized) ως JSON και τα αποσυνδέει (deserializes) σε αντικείμενα «ShoppingCartModel».

Αυτή η μικροϋπηρεσία αναγνωρίζει τα συμβάντα στο καλάθι αγορών, δημιουργεί ειδοποιήσεις μέσω email με βάση αυτά τα συμβάντα και τις στέλνει στους κατάλληλους παραλήπτες. Χρησιμοποιεί το Kafka για τη μετάδοση μηνυμάτων, το Spring Boot για τη ρύθμιση και τη διαχείριση εφαρμογών και ενσωματώνεται με μια υπηρεσία email για την αποστολή ειδοποιήσεων.

Αναλυτικότερα οι παραπάνω κλάσεις λειτουργούν ως εξής:

Consumer

```
package com.emailNotifierApp.services;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.messaging.MessageHeaders;
import org.springframework.messaging.handler.annotation.Headers;
import org.springframework.messaging.handler.annotation.Payload;
import org.springframework.stereotype.Service;

import com.emailNotifierApp.entities.ShoppingCartModel;
import com.emailNotifierApp.services.mail.MailTemplateBuilder;

import lombok.extern.slf4j.Slf4j;

@Slf4j
@Service
public class Consumer {

    @Autowired
    private JavaMailSender mailSender;


    @KafkaListener(topics = "${spring.kafka.topic.json}")
```

```
    public void receive(@Payload ShoppingCartModel shoppingCartModel,
                        @Headers MessageHeaders headers) {

        log.info("New notification received: {}", shoppingCartModel);
        MailTemplateBuilder mailTemplateBuilder= new MailTemplateBuilder();
        mailTemplateBuilder.setShoppingCartModel(shoppingCartModel);

        sendMail(mailTemplateBuilder, shoppingCartModel);

        log.info("Email Sent to {}",shoppingCartModel.getEmail());

    }

    private void sendMail(MailTemplateBuilder mailTemplateBuilder,
ShoppingCartModel model) {
        mailSender.send(mailTemplateBuilder.prepareMessage(model));
    }
}
```

Ο παραπάνω κώδικας αναπτύσσει έναν Kafka consum στην Java, χρησιμοποιώντας το Spring Framework για τη λήψη μηνυμάτων από ένα topic Kafka. Ο καταναλωτής αυτός “ακούει” για μηνύματα που περιέχουν δεδομένα `ShoppingCartModel`. Όταν λάβει ένα μήνυμα, εκτυπώνει ένα log με τα δεδομένα που έλαβε, δημιουργεί ένα email χρησιμοποιώντας το `MailTemplateBuilder` για να διαμορφώσει το περιεχόμενο του μηνύματος με βάση το `ShoppingCartModel`, και στη συνέχεια αποστέλλει το email μέσω του `JavaMailSender`. Καταλήγει στην εκτύπωση ενός log με την επιτυχή αποστολή του email στη διεύθυνση που παρέχεται από το `ShoppingCartModel`.

MailTemplateBuilder

```
package com.emailNotifierApp.services.mail;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;

import com.emailNotifierApp.entities.ShoppingCartModel;

import lombok.extern.slf4j.Slf4j;

@Component
@Slf4j
public class MailTemplateBuilder {
```

```
    @Value("${email.from}")
    private String from;
    @Value("${email.subject}")
    private String subject;

    private ShoppingCartModel shoppingCartModel;

    public MailTemplateBuilder setShoppingCartModel(ShoppingCartModel
shoppingCartModel) {
        this.shoppingCartModel = shoppingCartModel;
        return this;
    }


    private SimpleMailMessage getMailTemplate(SimpleMailMessage mailMessage)
{
        log.info("Created mail template");

        mailMessage.setFrom(from);
        mailMessage.setSubject(subject);

        return mailMessage;
    }

    public SimpleMailMessage prepareMessage (ShoppingCartModel model) {
        SimpleMailMessage mail = new  SimpleMailMessage();
        log.info("Preparing mail for : {}",model.getEmail());
        if (!StringUtils.isEmpty(model.getEmail())) {
            //mail = getMailTemplate(mail);
            //mail.setFrom("yourSushiOrder-ByIlias-donotreply@mail.com");
            mail.setSubject("Your order is succesfully submitted!");
            mail.setTo(model.getEmail());
            mail.setText(prepareMailBody(model));
        }
        return mail;
    }

    private String prepareMailBody(ShoppingCartModel model) {
        StringBuilder mailBody = new StringBuilder();
        String dearText = String.format("Dear %s, ", model.getFirstName() +
" " + model.getLastName());
        String mainText = "Your delicious sushi order has been
accepted.\nOur team is getting it ready and will be sent to your address
shortly.";
        String endText = "Thank you for choosing us,\n Your Sushi team";
        String credits = "This is an automated message sent through Spring
emailNotifier app microservice\n";
        String developer = "Creator: Ilias Eleftheriou";
```

```
        mailBody.append(dearText);
        mailBody.append("\n\n");
        mailBody.append(mainText);
        mailBody.append("\n\n");
        mailBody.append(endText);
        mailBody.append("\n\n\n");
        mailBody.append(credits);
        mailBody.append(developer);
        return mailBody.toString();
    }
}
```

Ο παραπάνω κώδικας αποτελεί μέρος του microservice για την αποστολή email, χρησιμοποιώντας το Spring Framework. Ο `MailTemplateBuilder` είναι ένας δόμητης προτύπων email που διαμορφώνει και αποστέλλει απλά μηνύματα email με βάση τα δεδομένα ενός `ShoppingCartModel`. Χρησιμοποιεί τις ρυθμίσεις `from` και `subject` από το αρχείο properties της εφαρμογής για να ορίσει τον αποστολέα και το θέμα του μηνύματος. Η μέθοδος `prepareMessage` δημιουργεί το περιεχόμενο του email, περιλαμβάνοντας έναν ευγενή χαιρετισμό, λεπτομέρειες για την παραγγελία, και έναν ευχαριστήριο καταληκτικό χαιρετισμό, καθώς και πληροφορίες από τον δημιουργό της εφαρμογής.

EmailNotifierApplication

```
package com.emailNotifierApp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class EmailNotifierApplication {

    public static void main(String[] args) {
        SpringApplication.run(EmailNotifierApplication.class, args);
    }
}
```

Ο παραπάνω κώδικας ορίζει την κύρια κλάση για την εφαρμογή Spring Boot με το όνομα `EmailNotifierApplication`. Χρησιμοποιεί την `@SpringBootApplication`, η οποία είναι μια συντόμευση για την προσθήκη `@Configuration`, `@EnableAutoConfiguration`, και `@ComponentScan`. Αυτό ενεργοποιεί την αυτόματη διαμόρφωση και την ανίχνευση

συστατικών στο πακέτο και τα υποπακέτα της κλάσης. Η μέθοδος `main` εκκινεί την εφαρμογή με τη χρήση της `SpringApplication.run`, περνώντας την κλάση `EmailNotifierApplication` και τα εισερχόμενα ορίσματα από τη γραμμή εντολών.

MsgListener

```
package com.emailNotifierApp;

import java.util.HashMap;
import java.util.Map;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.annotation.EnableKafka;
import
org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;
import org.springframework.kafka.config.KafkaListenerContainerFactory;
import org.springframework.kafka.core.ConsumerFactory;
import org.springframework.kafka.core.DefaultKafkaConsumerFactory;
import
org.springframework.kafka.support.converter.StringJsonMessageConverter;

import com.emailNotifierApp.entities.ShoppingCartModel;


@Configuration
@EnableKafka
public class MsgListener {

    @Value("${spring.kafka.bootstrap-servers}")
    private String bootstrapServers;

    @Bean
    public StringJsonMessageConverter jsonConverter() {
        return new StringJsonMessageConverter();
    }

    private ConsumerFactory<String, ShoppingCartModel>
accountConsumerFactory() {
        Map<String, Object> configProps = new HashMap<>();
        configProps.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
bootstrapServers);
        configProps.put(ConsumerConfig.GROUP_ID_CONFIG, "device_topics");
```

```
        configProps.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
        configProps.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
        return new DefaultKafkaConsumerFactory<>(configProps);
    }

    @Bean
    public KafkaListenerContainerFactory<?>
kafkaJsonListenerContainerFactory() {
        ConcurrentKafkaListenerContainerFactory<String, ShoppingCartModel>
factory = new ConcurrentKafkaListenerContainerFactory<>();
        factory.setConsumerFactory(accountConsumerFactory());
        factory.setMessageConverter(new StringJsonMessageConverter());
        return factory;
    }
}
```

Ο παραπάνω κώδικας δημιουργεί τη διαμόρφωση για έναν Kafka consumer σε μια εφαρμογή Spring. Χρησιμοποιεί τις ρυθμίσεις από το αρχείο properties της εφαρμογής για να συνδεθεί με τους Kafka servers. Διαμορφώνει έναν consumer για την ακρόαση μηνυμάτων που είναι serialized ως JSON και μετατρέπει αυτά τα μηνύματα πίσω σε Java objects τύπου `ShoppingCartModel`. Χρησιμοποιεί το `StringJsonMessageConverter` για τη μετατροπή των μηνυμάτων JSON σε Java objects, επιτρέποντας στην εφαρμογή να επεξεργάζεται τα δεδομένα που λαμβάνει από τα μηνύματα Kafka.

ServletInitializer

```
package com.emailNotifierApp;

import org.springframework.boot.builder.SpringApplicationBuilder;
import
org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

public class ServletInitializer extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder
application) {
        return application.sources(EmailNotifierApplication.class);
    }

}
```

Ο παραπάνω κώδικας επεκτείνει την κλάση `SpringBootServletInitializer` για να υποστηρίξει την παραδοσιακή ανάπτυξη WAR σε έναν web container. Χρησιμοποιείται

όταν θέλουμε να αναπτύξουμε την εφαρμογή Spring Boot ως traditional WAR, αντί για ένα αυτόνομο JAR. Η μέθοδος `configure` ρυθμίζει την εφαρμογή προσθέτοντας πηγές - σε αυτή την περίπτωση, την κλάση `EmailNotifierApplication`. Αυτό επιτρέπει στην εφαρμογή να ρυθμιστεί και να ξεκινήσει μέσα από έναν servlet container.

Συνοψίζοντας για το παραπάνω microservice ισχύουν τα εξής:
Το αρχείο EmailNotifierApplication.java ορίζει την κύρια κλάση για μια εφαρμογή Spring Boot που ονομάζεται EmailNotifierApplication. Έχει σχολιαστεί με το @SpringBootApplication, το οποίο είναι ένας βολικός σχολιασμός που προσθέτει όλα τα ακόλουθα: @Configuration, @EnableAutoConfiguration και @ComponentScan. Αυτή η κλάση περιέχει την κύρια μέθοδο που εκκινεί την εφαρμογή Spring Boot χρησιμοποιώντας SpringApplication.run(EmailNotifierApplication.class, args);. Αυτή η ρύθμιση είναι τυπική για την εκκίνηση ενός έργου Spring Boot, που χρησιμεύει ως σημείο εισόδου για την εφαρμογή.
Η κλάση MsgListener.java διαμορφώνει ένα πρόγραμμα ακρόασης μηνυμάτων Kafka για τη microservice. Έχει σχεδιαστεί για να καταναλώνει μηνύματα από θέματα Kafka, ειδικά για το χειρισμό μηνυμάτων που περιέχουν δεδομένα ShoppingCartModel. Αυτή η ρύθμιση επιτρέπει στη microservice να λαμβάνει και να επεξεργάζεται πληροφορίες καλαθιού αγορών σε πραγματικό χρόνο, μετατρέποντας ωφέλιμα φορτία μηνυμάτων JSON σε αντικείμενα ShoppingCartModel. Ουσιαστικά, αυτή η μικροϋπηρεσία λειτουργεί ως consumer σε μια αρχιτεκτονική που βασίζεται σε μηνύματα, όπου μπορεί να λαμβάνει ενημερώσεις ή εντολές που σχετίζονται με τις λειτουργίες του καλαθιού αγορών, ενεργοποιώντας ειδοποιήσεις μέσω email με βάση τα δεδομένα που λαμβάνονται.

6.6.2 shoppingCart

Το πρόγραμμα shoppingCart αναπτύχθηκε ως μια εφαρμογή microservice για τη διαχείριση ενός καλαθιού αγορών. Χρησιμοποιεί Spring Boot για τη δημιουργία και την εκκίνηση της εφαρμογής, και διαθέτει υποστήριξη για CORS, επιτρέποντας την αλληλεπίδραση με άλλες υπηρεσίες μέσω διαφορετικών πηγών. Παρέχει λειτουργίες για τη διαχείριση των αντικειμένων μέσα στο καλάθι αγορών, όπως η προσθήκη, η αφαίρεση

και η ενημέρωση προϊόντων. Επιπλέον, ενσωματώνει την ασύγχρονη επικοινωνία μέσω Kafka για την αποστολή δεδομένων του καλαθιού αγορών σε ένα Kafka topic, πιθανώς για περαιτέρω επεξεργασία ή ανάλυση. Αυτή η εφαρμογή μπορεί να χρησιμοποιηθεί σε οποιοδήποτε περιβάλλον χρειάζεται τη διαχείριση ενός καλαθιού αγορών, όπως ηλεκτρονικές αγορές ή ηλεκτρονικό εμπόριο.

Στη συνέχεια αναλύονται οι κύριες κλάσεις του shoppingCart.

ShoppingCartApplication

```
package com.shoppingCart;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.filter.CorsFilter;

@SpringBootApplication
public class ShoppingCartApplication {

    public static void main(String[] args) {
        SpringApplication.run(ShoppingCartApplication.class, args);
    }

    @Bean
    public CorsFilter corsFilter() {

        UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
        CorsConfiguration config = new CorsConfiguration();
        config.setAllowCredentials(true);
        config.addAllowedOrigin("*");
        config.addAllowedHeader("*");
        config.addAllowedMethod("OPTIONS");
        config.addAllowedMethod("HEAD");
        config.addAllowedMethod("GET");
        config.addAllowedMethod("PUT");
        config.addAllowedMethod("POST");
        config.addAllowedMethod("DELETE");
        config.addAllowedMethod("PATCH");
        source.registerCorsConfiguration("/**", config);
        return new CorsFilter(source);
    }
}
```

Ο παραπάνω κώδικας είναι η κύρια κλάση για την εφαρμογή Spring Boot, η οποία διαχειρίζεται ένα καλάθι αγορών. Χρησιμοποιεί το `@SpringBootApplication` για να δηλώσει ότι πρόκειται για μια Spring Boot εφαρμογή. Η μέθοδος `main` εκκινεί την εφαρμογή. Παρέχει επίσης μια ρύθμιση CORS (Cross-Origin Resource Sharing) μέσω του `CorsFilter`, επιτρέποντας την πρόσβαση από οποιαδήποτε πηγή (με `*`) και υποστηρίζοντας διάφορες HTTP μεθόδους. Αυτό εξασφαλίζει ότι η εφαρμογή μπορεί να δεχτεί αιτήματα από διαφορετικά domains.

ServletInitializer

```
package com.shoppingCart;

import org.springframework.boot.builder.SpringApplicationBuilder;
import
org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

public class ServletInitializer extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder
application) {
        return application.sources(ShoppingCartApplication.class);
    }

}
```

Ο παραπάνω κώδικας είναι μέρος της εφαρμογής Spring Boot και χρησιμοποιείται για την προσαρμογή της διαδικασίας αρχικοποίησης του Servlet container. Κληρονομεί από την κλάση `SpringBootServletInitializer` και υπερκαλύπτει τη μέθοδο `configure`, ορίζοντας την κύρια κλάση της εφαρμογής, εδώ `ShoppingCartApplication`, ως την πηγή της εφαρμογής. Αυτό επιτρέπει την ομαλή ενσωμάτωση και λειτουργία της εφαρμογής Spring Boot μέσα σε ένα περιβάλλον web server ή servlet container όπως το Tomcat.

ShoppingCartController

```
package com.shoppingCart.controllers;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import com.shoppingCart.entities.ShoppingCartModel;
import com.shoppingCart.services.ShoppingCartService;
import lombok.extern.slf4j.Slf4j;

@Slf4j
@RestController
@RequestMapping(value = "/shoppingCart", consumes = "application/json")
public class ShoppingCartController {

    @Autowired
    private ShoppingCartService shoppingCartService;

    @RequestMapping(method = RequestMethod.POST)
    public ResponseEntity<?> newOrder(@RequestBody ShoppingCartModel
shoppingCartModel) {
        log.info("New order received: {}", shoppingCartModel);
        shoppingCartService.newOrder(shoppingCartModel);
        return new ResponseEntity<String>(HttpStatus.OK);
    }
}
```

Ο παραπάνω κώδικας παρουσιάζει έναν Spring Boot controller για τη διαχείριση νέων παραγγελιών σε ένα καλάθι αγορών. Χρησιμοποιεί το `@RestController` και `@RequestMapping` για να δηλώσει ένα REST API endpoint που δέχεται JSON στο `/shoppingCart`. Με την μέθοδο POST, δέχεται ένα αίτημα με το μοντέλο `ShoppingCartModel` ως το σώμα του αιτήματος. Καταγράφει την παραγγελία και καλεί την υπηρεσία `shoppingCartService` για να εκτελέσει την λειτουργία `newOrder`. Επιστρέφει ένα `ResponseEntity` με κατάσταση `HttpStatus.OK`.

Entities

ShoppingCartModel

```
package com.shoppingCart.entities;
import java.util.Date;
import java.util.List;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Transient;
import com.fasterxml.jackson.annotation.JsonIgnore;
import lombok.Data;
@Entity
@Data
public class ShoppingCartModel {
    public ShoppingCartModel() {
        this.orderDate = new Date();
    }
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    @JsonIgnore
    private Long id;
    private String firstName;
    private String lastName;
    private String userName;
    private String email;
    private String address;
    private String address2;
    private String country;
    private Integer zip;
    private Boolean shippingAddrSameToBilling;
    private Boolean saveInfo;
    private Date orderDate;
    private String wayOfPayment;
    private String nameOnCard;
    private String cardNumber;
    private String cardExpiration;
    private Integer cvv;
    @Transient
    private List<ShoppingItem> listOfItems;
}
```

Ο παραπάνω κώδικας ορίζει μια JPA entity `ShoppingCartModel` για την αντιπροσώπευση ενός καλαθιού αγορών σε μια βάση δεδομένων. Διαθέτει πεδία για την αποθήκευση πληροφοριών του χρήστη, όπως όνομα, διεύθυνση, email, και πληροφορίες πληρωμής. Τα

πεδία `@Id` και `@GeneratedValue` χρησιμοποιούνται για τον καθορισμό ενός μοναδικού αναγνωριστικού για κάθε καταχώρηση. Το `@Transient` σημαίνει ότι το `listOfItems` δεν θα περιλαμβάνεται στη βάση δεδομένων. Το `@JsonIgnore` αποκρύπτει το πεδίο `id` από αποτελέσματα JSON.

ShoppingItem

```
package com.shoppingCart.entities;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

import com.fasterxml.jackson.annotation.JsonIgnore;

import lombok.Data;

@Entity
@Data
public class ShoppingItem {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    @JsonIgnore
    private Long id;

    private String name;

    private Integer price;

    @JsonIgnore
    private String username;
}
```

Ο παραπάνω κώδικας ορίζει μια JPA entity `ShoppingItem` για την αποθήκευση πληροφοριών σχετικά με αντικείμενα σε ένα καλάθι αγορών. Περιλαμβάνει πεδία για το `id` (μοναδικό αναγνωριστικό για κάθε αντικείμενο, το οποίο δημιουργείται αυτόματα), το `name` (όνομα του προϊόντος), την `price` (τιμή του προϊόντος), και το `username` (που αναφέρεται στον χρήστη που προσθέτει το αντικείμενο, αλλά αποκρύπτεται από τις JSON αποκρίσεις με τη χρήση του `@JsonIgnore`).

Repositories

ItemRepo

```
package com.shoppingCart.repositories;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import com.shoppingCart.entities.ShoppingItem;

@Repository
public interface ItemRepo extends CrudRepository<ShoppingItem, Long> {

}
```

Ο παραπάνω κώδικας ορίζει ένα Spring Data repository για την entity `ShoppingItem`, χρησιμοποιώντας το `CrudRepository` του Spring. Αυτό παρέχει μια σειρά από βασικές CRUD (Create, Read, Update, Delete) λειτουργίες για τη διαχείριση αντικειμένων `ShoppingItem` στη βάση δεδομένων, χωρίς την ανάγκη για ρητή υλοποίηση των μεθόδων. Η διακριτική ένδειξη `@Repository` δηλώνει ότι η διεπαφή είναι ένα Spring bean και λειτουργεί ως αποθετήριο δεδομένων.

ShoppingCartRepo

```
package com.shoppingCart.repositories;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import com.shoppingCart.entities.ShoppingCartModel;

@Repository
public interface ShoppingCartRepo extends CrudRepository<ShoppingCartModel,
Long> {

}
```

Ο παραπάνω κώδικας ορίζει ένα Spring Data repository για την entity `ShoppingCartModel`, χρησιμοποιώντας το interface `CrudRepository` του Spring Framework. Αυτό το repository παρέχει μια σειρά από βασικές λειτουργίες για την εκτέλεση CRUD (Create, Read, Update, Delete) ενεργειών στη βάση δεδομένων για τα αντικείμενα τύπου `ShoppingCartModel`. Η διεπαφή επεκτείνει το `CrudRepository`, δηλώνοντας τον τύπο της entity και τον τύπο του πρωτεύοντος κλειδιού (εδώ, `Long`). Η

ετικέτα `@Repository` σηματοδοτεί ότι η διεπαφή είναι ένα Spring bean που λειτουργεί ως αποθετήριο δεδομένων, διευκολύνοντας την ενσωμάτωση με το Spring Data infrastructure.

Services

ShoppingCartService

```
package com.shoppingCart.services;

import java.util.List;
import java.util.stream.Collectors;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;
import com.shoppingCart.entities.ShoppingCartModel;
import com.shoppingCart.entities.ShoppingItem;
import com.shoppingCart.repositories.ItemRepo;
import com.shoppingCart.repositories.ShoppingCartRepo;


@Service
public class ShoppingCartService {

    @Autowired
    private ShoppingCartRepo shoppingCartRepo;
    @Autowired
    private ItemRepo itemRepo;
    @Autowired
    private KafkaTemplate<String, ShoppingCartModel> kafkaTemplate;
    @Value("${spring.kafka.topic}")
    private String topic;
    public void newOrder(ShoppingCartModel shoppingCartModel) {
        List<ShoppingItem> listOfItems =
shoppingCartModel.getListOfItems().stream().map(item -> {
            item.setUsername(shoppingCartModel.getUserName());
            return item;
        }).collect(Collectors.toList());

        shoppingCartRepo.save(shoppingCartModel);
        itemRepo.saveAll(listOfItems);
        kafkaTemplate.send(topic, shoppingCartModel);
    }
}
```

Ο παραπάνω κώδικας αναπτύσσει ένα service στο Spring, το `ShoppingCartService`, το οποίο χειρίζεται νέες παραγγελίες για το καλάθι αγορών. Αυτό επιτυγχάνεται με την

αποθήκευση της παραγγελίας και των προϊόντων που περιέχει στη βάση δεδομένων, και στη συνέχεια αποστέλλοντας τα στοιχεία της παραγγελίας σε ένα Kafka topic. Χρησιμοποιεί το `KafkaTemplate` για την αποστολή της παραγγελίας στο Kafka, διευκολύνοντας έτσι την ασύγχρονη επεξεργασία ή ανάλυση των παραγγελιών.

Συνοψίζοντας για το παραπάνω microservice ισχύουν τα εξής:

Το αρχείο `ServletInitializer.java` είναι μέρος ενός Spring Boot application, το οποίο χρησιμεύει για την αρχικοποίηση της εφαρμογής σε ένα Servlet container. Κληρονομεί από την κλάση `SpringBootServletInitializer`, η οποία είναι βοηθητική για τη δημιουργία εφαρμογών Spring Boot που μπορούν να τρέξουν σε ένα Servlet container. Συγκεκριμένα, παρέχει μια μέθοδο `configure` που συνδέει την κύρια κλάση της εφαρμογής, `ShoppingCartApplication`, με το Spring application builder.

Συγκεκριμένα, το microservice είναι μια εφαρμογή Spring Boot που σχετίζεται με ένα σύστημα αγορών ή καλαθιού αγορών.

Το αρχείο `ShoppingCartApplication.java` αποτελεί την κύρια κλάση εκκίνησης για την εφαρμογή Spring Boot, ορίζοντας την ως μια Spring Boot εφαρμογή με τη χρήση του annotation `@SpringBootApplication`. Εκκινεί την εφαρμογή με την κλήση `SpringApplication.run()`. Επιπλέον, περιλαμβάνει μια ρύθμιση για το CORS (Cross-Origin Resource Sharing) μέσω του `CorsFilter`, επιτρέποντας requests από οποιαδήποτε πηγή (με το `config.addAllowedOrigin("*")`) και υποστηρίζοντας διάφορες HTTP μεθόδους όπως GET, POST, DELETE, κ.α.

Δηλαδή, η παρούσα εφαρμογή είναι σχεδιασμένη για να υποστηρίζει web-based πρόσβαση και διαλειτουργικότητα με άλλες εφαρμογές ή υπηρεσίες μέσω διαδικτύου, εντός του πλαισίου ενός συστήματος καλαθιού αγορών.

Το `ShoppingCartService.java` είναι υπεύθυνο για τη διαχείριση των λειτουργιών του καλαθιού αγορών. Χρησιμοποιεί τα `ShoppingCartRepo` και `ItemRepo` για την αποθήκευση και διαχείριση δεδομένων στη βάση δεδομένων. Επίσης, ενσωματώνει την

έννοια της ασύγχρονης επικοινωνίας με τη χρήση του Kafka, όπου αποστέλλει τα δεδομένα του καλαθιού αγορών σε ένα Kafka topic. Αυτό επιτρέπει την επεξεργασία ή την ανάλυση των παραγγελιών καλαθιού αγορών σε πραγματικό χρόνο ή την ενσωμάτωση με άλλες υπηρεσίες ή συστήματα.

Κεφάλαιο 7 - Συμπεράσματα

Συμπερασματικά, η σύγκριση μεταξύ μονολιθικής αρχιτεκτονικής και αρχιτεκτονικής microservices αποτελεί ένα σημαντικό κομμάτι της σύγχρονης ανάπτυξης λογισμικού [1]. Για αυτό εξετάστηκαν τόσο τα πλεονεκτήματα όσο και τα μειονεκτήματα κάθε προσέγγισης, καθώς και η διαδικασία μετάβασης, οι προκλήσεις και τα εργαλεία που χρησιμοποιούνται, όπως το Azure Service Fabric, οι δυνατότητες της Lambda, το Camunda framework και η λειτουργία του CamundaService microservice, καθώς και ο ρόλος του API Gateway και οι γλώσσες προγραμματισμού.

Η μονολιθική αρχιτεκτονική προσφέρει απλότητα στην ανάπτυξη και τη διαχείριση εφαρμογών, καθώς όλος ο κώδικας ενσωματώνεται σε ένα μόνο σύνολο. Αυτό μπορεί να καταστήσει την ενίσχυση και τη συντήρηση ευκολότερη, ειδικά για μικρά ή αρχικά στάδια έργων. Ωστόσο, η μονολιθική αρχιτεκτονική μπορεί να αντιμετωπίσει προβλήματα όσο η εφαρμογή μεγαλώνει, καθώς το μονολιθικό σύστημα γίνεται δυσχερές να κλιμακωθεί και να επεκταθεί [7].

Αντίθετα, η αρχιτεκτονική microservices διασπά την εφαρμογή σε μικρά, αυτόνομα και ανεξάρτητα microservices, τα οποία επικοινωνούν μεταξύ τους μέσω διαδικτυακών πρωτοκόλλων. Αυτό επιτρέπει την ευελιξία και την επεκτασιμότητα των εφαρμογών, καθώς κάθε microservice μπορεί να αναπτυχθεί, να ενημερωθεί και να κλιμακωθεί ανεξάρτητα. Ωστόσο, αυτή η προσέγγιση απαιτεί σύνθετη διαχείριση και συντήρηση, καθώς και πιο προηγμένες δεξιότητες ανάπτυξης [28].

Η μετάβαση από μονολιθική σε αρχιτεκτονική microservices μπορεί να είναι προκλητική και να απαιτεί σταδιακή διαδικασία. Απαιτείται, συνεπώς, εκπαίδευση και εξοικείωση του προσωπικού με νέες τεχνολογίες και πρακτικές ανάπτυξης λογισμικού. Τα εργαλεία, όπως το Azure Service Fabric, οι δυνατότητες της Lambda και το Camunda framework παρέχουν λύσεις για την ανάπτυξη και τη διαχείριση microservice, ενώ η λειτουργία του CamundaService microservice και ο ρόλος του API Gateway [91] συμβάλλουν στην αποτελεσματική διαχείριση της επικοινωνίας και του δικτύου συνολικά.

Συνοψίζοντας, η μετάβαση από μονολιθική σε αρχιτεκτονική microservices μπορεί να προσφέρει ευελιξία και βελτιωμένη απόδοση στην ανάπτυξη εφαρμογών, αλλά απαιτεί

προσεκτικό σχεδιασμό, διαχείριση και εξοικείωση με νέες τεχνολογίες και διαδικασίες. Τα εργαλεία και οι πλατφόρμες, όπως τα προαναφερθέντα μπορούν να διευκολύνουν αυτήν τη μετάβαση και να βοηθήσουν στην αποτελεσματική διαχείριση των microservices [28].

7.1 Σύνοψη και συμπεράσματα

Συνοψίζοντας, η μετάβαση από μονολιθική σε αρχιτεκτονική microservices μπορεί να προσφέρει ευελιξία και βελτιωμένη απόδοση στην ανάπτυξη εφαρμογών, αλλά απαιτεί προσεκτικό σχεδιασμό, διαχείριση και εξοικείωση με νέες τεχνολογίες και διαδικασίες. Τα εργαλεία και οι πλατφόρμες, όπως τα προαναφερθέντα μπορούν να διευκολύνουν αυτήν τη μετάβαση και να βοηθήσουν στην αποτελεσματική διαχείριση των microservices [28].

Όσον αφορά τα microservices που αναπτύχθηκαν στα πλαίσια της παρούσας εργασίας συμπερασματικά προέκυψαν τα εξής: αναφέρονται σε ένα σύστημα καλαθιού αγορών, με λειτουργίες για διαχείριση παραγγελιών και αντικειμένων. Στα πλεονεκτήματά τους περιλαμβάνονται η ευκολία διαχείρισης δεδομένων μέσω CRUD operations, η αποδοτική ενσωμάτωση με Kafka για ασύγχρονη επεξεργασία, και η υποστήριξη CORS για ευρύτερη διαλειτουργικότητα. Μειονεκτήματα μπορεί να θεωρηθούν η πολυπλοκότητα διαχείρισης κατανεμημένων μηνυμάτων και η ανάγκη για λεπτομερή διαχείριση ασφάλειας και προστασίας δεδομένων. Γενικώς, χρησιμοποιούνται κυρίως για web-based εμπορικές εφαρμογές, προσφέροντας μια ολοκληρωμένη λύση για τη διαχείριση καλαθιών αγορών και την επεξεργασία παραγγελιών.

7.2 Μελλοντικές επεκτάσεις

Οι μελλοντικές επεκτάσεις στον τομέα της αρχιτεκτονικής λογισμικού, ειδικά όσον αφορά τη μετάβαση από μονολιθικές προσεγγίσεις σε αρχιτεκτονικές microservices, μπορούν να επηρεαστούν από διάφορους παράγοντες και τάσεις του χώρου της τεχνολογίας. Για αυτό μερικές από τις μελλοντικές έρευνες μπορούν να περιλαμβάνουν:

- Αυτοματοποίηση και ολοκλήρωση DevOps: Οι αρχιτεκτονικές microservices απαιτούν προηγμένες διαδικασίες DevOps για την αποτελεσματική ανάπτυξη,

διαχείριση και κατανομή εφαρμογών. Η αυτοματοποίηση των διαδικασιών DevOps και η ολοκλήρωσή τους με εργαλεία CI/CD μπορεί να εξελιχθεί περαιτέρω.

- Εξέλιξη των εργαλείων και πλατφορμών: Νέα εργαλεία και πλατφόρμες που είναι σχεδιασμένα ειδικά για τη διαχείριση και την επέκταση αρχιτεκτονικών microservices αναμένεται να εμφανιστούν. Οι υπηρεσίες cloud και οι πλατφόρμες PaaS (Platform as a Service) επίσης θα συνεχίσουν να προσφέρουν νέες λύσεις και υπηρεσίες.
- Ασφάλεια και προστασία δεδομένων: Καθώς οι αρχιτεκτονικές microservices αυξάνουν τον αριθμό των σημείων επαφής και επιφέρουν περισσότερα δίκτυα επικοινωνίας, η ασφάλεια των εφαρμογών και η προστασία των δεδομένων θα είναι ουσιαστικής σημασίας. Εργαλεία, όπως η κρυπτογράφηση δεδομένων και η αυτοπροστασία εφαρμογών και η αυτοματοποίηση θα είναι ιδιαίτερα σημαντικά.

Εξέλιξη των αρχιτεκτονικών και προτύπων επικοινωνίας: Νέες αρχιτεκτονικές και πρότυπα επικοινωνίας μεταξύ microservices πρόκειται να εξελιχθούν, καθώς η ανάπτυξη τεχνολογιών όπως η gRPC και η GraphQL αποκτά ολοένα και μεγαλύτερη δημοτικότητα.

Βιβλιογραφία

[1] Zolotariov, D. (2021). AUTOMATED DEPLOYMENT OF A SOFTWARE ENVIRONMENT FOR MICROSERVICES IN A RAPIDLY CHANGING TECHNOLOGY STACK. Innovative Technologies and Scientific Solutions for Industries, (4 (18), 23-30. https://doi.org/10.30837/ITSSI.2021.18.023

[2] Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. Journal of Systems and Software, 150, 77-97. https://doi.org/10.1016/j.jss.2019.01.001

[3] Newman, S. (2021). Building microservices. " O'Reilly Media, Inc.".

[4] Bucchiarone, A., Dragoni, N., Dustdar, S., Lago, P., Mazzara, M., Rivera, V., & Sadovykh, A. (2020). Microservices. Science and Engineering. Springer.

[5] Waseem, M., Liang, P., & Shahin, M. (2020). A systematic mapping study on microservices architecture in devops. Journal of Systems and Software, 170, 110798. https://doi.org/10.1016/j.jss.2020.110798.

[6] Thomas, J., Utley, J., Hong, S. Y., Korkmaz, H., & Nugent, G. (2020). A Review of the Research. Handbook of Research on STEM Education. Routledge.

[7] Salah, T., Zemerly, J. M., Yeun, C. Y., Al-Qutayri, M. & Al-Hammadi, Y. (2016). The evolution of distributed systems towards microservices architecture, 11th International Conference for Internet Technology and Secured Transactions (ICITST), pp. 318-325, https://doi.org/10.1109/ICITST.2016.7856721

[8] Mendonça, N. C., Box, C., Manolache, C., & Ryan, L. (2021). The monolith strikes back: Why istio migrated from microservices to a monolithic architecture. IEEE software, 38(5), 17-22. https://doi.org/ 10.1109/MS.2021.3080335

[9] Jaramillo, D., Nguyen, D. V. & Smart, R. (2016). Leveraging microservices architecture by using Docker technology.

[10] Tapia, F., Mora, M. Á., Fuertes, W., Aules, H., Flores, E., & Toulkeridis, T. (2020). From monolithic systems to microservices: A comparative study of performance. Applied sciences, 10(17), 5797. https://doi.org/10.3390/app10175797

[11] Nielsen, M. A. (2015). Neural networks and deep learning (Vol. 25, pp. 15-24). San Francisco, CA, USA: Determination press.

[12] Oluwatosin, H. S. (2014). Client-server model. IOSR Journal of Computer Engineering, 16(1), 67-71.

[13] Zhang, H. (2013). Architecture of network and client-server model. arXiv preprint arXiv:1307.6665.https://doi.org/10.48550/arXiv.1307.6665

[14] Hanson, M. D. (2000). The client/server architecture. In Server Management (pp. 17-28). Auerbach Publications.

[15] Kratky, S., & Reichenberger, C. (2013). Client/Server Development based on the Apple Event Object Model. Atlanta.

[16] Shadija, D., Rezai, M. & Hill, R. (2017). Towards an understanding of microservices, 23rd International Conference on Automation and Computing (ICAC), 2017, pp. 1-6, https://doi.org/10.23919/IConAC.2017.8082018

[17] Jamshidi, P., Pahl, C. Mendonça, N. C. Lewis, J. & Tilkov, S. (2018). The Journey So Far and Challenges Ahead.

[18] Sheikh, K., Gilson, L., Agyepong, I. A., Hanson, K., Ssengooba, F., & Bennett, S. (2011). Building the field of health policy and systems research: framing the questions. PLoS medicine, 8(8), e1001073.

[19] Ingeno, J. (2018). Software Architect's Handbook: Become a successful software architect by implementing effective architecture concepts. Packt Publishing Ltd.

[20] Beydoun, G., Xu, D., & Sugumaran, V. (2013). Service Oriented Architectures (SOA) Adoption Challenges. International Journal of Intelligent Information Technologies (IJIIT), 9(2), 1-6. https://doi.org/10.4018/jiit.2013040101

[21] Alanazi, S. T., Abdullah, N., Anbar, M., & Al-Wesabi, O. A. (2019). Evaluation approaches of service oriented architecture (soa)-a survey. In 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS) (pp. 1-6). IEEE. . https://doi.org/10.1109/CAIS.2019.8769543.

[22] Berry, L. L., & Parasuraman, A. (2004). Marketing services: Competing through quality. Simon and Schuster.

[23] Blum, N., Magedanz, T., Schreiner, F., & Wahle, S. (2009, April). A research infrastructure for SOA-based Service Delivery Frameworks. In 2009 5th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities and Workshops (pp. 1-6). IEEE. https://doi.org/10.1109/TRIDENTCOM.2009.4976202

[24] Wen, B., Luo, Z., & Liang, P. (2014). Toward Customer-centric SOA: Services Resource Active Provisioning Approach. J. Softw., 9(4), 1007-1018.

[25] Keen, M., Acharya, A., Bishop, S., Hopkins, A., Milinski, S., Nott, C., ... & Verschueren, P. (2004). Patterns: Implementing an SOA using an enterprise service bus. IBM Redbooks, 336, 20-28.

[26] Amend, J., Fridgen, G., Rieger, A., Roth, T., & Stohr, A. (2021). The evolution of an architectural paradigm-using blockchain to build a cross-organizational enterprise service bus. In 54th Hawaii International Conference on System Sciences (HICSS), Maui, Hawaii (Virtual). https://hdl.handle.net/10993/46299

[27] Kohar, R. (2020). IoT systems based on SOA services: Methodologies, Challenges and Future directions. In 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC) (pp. 556-560). IEEE. https://doi.org/10.1109/ICCMC48092.2020.ICCMC-000103

[28] Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., ... & Babar, M. A. (2021). Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. Information and software technology, 131, 106449. https://doi.org/10.1016/j.infsof.2020.106449

[29] Sorgalla, J., Sachweh, S., Zündorf, A. (2020). "Exploring the Microservice Development Process in Small and Medium-Sized Organizations", In: Morisio M., Torchiano M., Jedlitschka A. (eds) Product-Focused Software Process Improvement. PROFES 2020. Lecture Notes in Computer Science, Springer, Cham, Vol. 12562. https://doi.org/10.1007/978-3-030-64148-1_28

[30] Di Francesco, P. (2017). Architecting microservices. In 2017 IEEE International Conference on Software Architecture Workshops (ICSAW) (pp. 224-229). IEEE. https://doi.org/10.1109/ICSAW.2017.65

[31] Munari, S., Valle, S., & Vardanega, T. (2018). Microservice-based agile architectures: An opportunity for specialized niche technologies. In Reliable Software Technologies–Ada-Europe 2018: 23rd Ada-Europe International Conference on Reliable Software Technologies, Lisbon, Portugal, June 18-22, 2018, Proceedings 23 (pp. 158-174). Springer International Publishing. https://doi.org/10.1007/978-3-319-92432-8_10

[32] CIO Dive (2021), "Guardian Life steers tech transformation with microservices", available at: https://www.ciodive.com/news/Cloud-Microservices-Guardian-Life/578732/ (Τελευταία πρόσβαση 4/1/24).

[33] Bogner, J., Fritzsch, J., Wagner, S., & Zimmermann, A. (2019, March). Microservices in industry: insights into technologies, characteristics, and software quality.

In 2019 IEEE international conference on software architecture companion (ICSA-C) (pp. 187-195). IEEE. https://doi.rg/10.1109/ICSA-C.2019.00041

[34] Pahl, C., & Jamshidi, P. (2016). Microservices: A Systematic Mapping Study. CLOSER (1), 137-146.

[35] Santos, N., Salgado, C. E., Morais, F., Melo, M., Silva, S., Martins, R., ... & Pereira, M. (2019). A logical architecture design method for microservices architectures. In Proceedings of the 13th European Conference on Software Architecture-Volume 2 (pp. 145-151). https://doi.org/10.1145/3344948.3344991

[36] Richardson, C. (2018). Microservice Patterns, 1st ed. Manning.

[37] Rademacher, F., Sachweh, S. & Zündorf, A. (2018). Analysis of Service-oriented Modeling Approaches for Viewpoint-specific Model-driven Development of Microservice Architecture, arXiv Prepr. arXiv1804.09946

[38] Esas, O. (2022). Design patterns and anti-patterns in microservices architecture: a classification proposal and study on open source projects.

[39] Munaf, R. M., Ahmed, J., Khakwani, F., & Rana, T. (2019). Microservices architecture: Challenges and proposed conceptual design. In 2019 International Conference on Communication Technologies (ComTech) (pp. 82-87). IEEE. https://doi.org/10.1109/COMTECH.2019.8737831

[40] Martin Fowler (2014), "Microservices", διαθέσιμο στο: https://martinfowler.com/articles/microservices.html (Τελευταία πρόσβαση 4/1/24).

[41] Lee, C. Y. (2019). Temporal Correlation Analysis of Programming Language Popularity", J. Korean Phys. Soc., Vol. 75, P. 755– 763. https://doi.org/10.3938/jkps.75.755

[42] Al-Debagy, O., & Martinek, P. (2018). A comparative review of microservices and monolithic architectures. In 2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI) (pp. 000149-000154). IEEE. https://doi.org/10.1109/CINTI.2018.8928192

[43] Newman, S. (2015). Building microservices: designing fine-grained systems. First Edition. Beijing Sebastopol, CA: O'Reilly Media.

[44] Chen, R. Li, S. & Li, Z. (2017). From Monolith to Microservices: A Dataflow-Driven Approach, in 2017 24th Asia-Pacific Software Engineering Conference (APSEC., pp. 466–475.

[45] Singh, V. & Peddoju, S. K. (2017). Container-based microservice architecture for cloud applications, in 2017 International Conference on Computing, Communication and Automation (ICCCA), pp. 847–852. https://doi.org/10.1109/CCAA.2017.8229914

[46] Richardson, C. (2020). Microservice Architecture. Ανακτήθηκε στις 4/1/24 από http://microservices.io/patterns

[47] Bozan, K., Lyytinen, K., & Rose, M. G. (2021). How to Transition Incrementally to Microservice Architecture, Communications of the ACM, Vol. 64 No. 1

[48] Auera, F., Lenarduzzi, V., Feldererac, M., & Taibid, D. (2021). From monolithic systems to Microservices: An assessment framework. [Journal] Information and Software Technology, Vol. 137. Retrieved from: https://doi.org/10.1016/j.infsof.2021.106600.

[49] Pradhan, R., & Dash, A. K. (2020). An Overview of Microservices. In ICDSMLA 2019: Proceedings of the 1st International Conference on Data Science, Machine Learning and Applications (pp. 620-625). Springer Singapore. https://doi.org/10.1007/978-981-15-1420-3_66

[50] Eski, S., & Buzluca, F. (2018). An automatic extraction approach: Transition to microservices architecture from monolithic application. In Proceedings of the 19th International Conference on Agile Software Development: Companion (pp. 1-6). https://doi.org/10.1145/3234152.3234195

[51] Taibi, D., & Systä, K. (2020). A decomposition and metric-based evaluation framework for microservices. In Cloud Computing and Services Science: 9th International Conference, CLOSER 2019, Heraklion, Crete, Greece, May 2–4, 2019, Revised Selected Papers 9 (pp. 133-149). Springer International Publishing. https://doi.org/10.1007/978-3-030-49432-

[52] Volynsky, E., Mehmed, M., & Krusche, S. (2022). Architect: A Framework for the Migration to Microservices. In 2022 International Conference on Computing, Electronics & Communications Engineering (iCCECE) (pp. 71-76). IEEE. https://doi.org/ 10.1109/iCCECE55162.2022.9875096

[53] Haselböck, S., & Weinreich, R. (2017). Decision guidance models for microservice monitoring. In 2017 IEEE international conference on software architecture workshops (ICSAW) (pp. 54-61). IEEE. https:doi.org/10.1109/ICSAW.2017.31

[54] Scheuch, R. (2015). Risikominimierung bei der Applikations-modernisierung mittels Microservices. OBJEKTspektrum Innovation in und durch Architekturen, 20-22.

[55] Amundsen, M. et al. (2016). Microservice Architecture. O'Reilly

[56] Hölttä-Otto, K., Chiriac, N., Lysy, D., & Suh, E. S. (2014). Architectural decomposition: the role of granularity and decomposition viewpoint. Advances in Product Family and Product Platform Design: Methods & Applications, 221-243. https://doi.org/10.1007/978-1-4614-7937-6_9

[57] Li, C. Y., Ma, S. P., & Lu, T. W. (2020). Microservice migration using strangler fig pattern: A case study on the green button system. In 2020 International Computer Symposium (ICS) (pp. 519-524). IEEE. https://doi.org/10.1109/ICS51289.2020.00107

[58] Evans, E. (2014). Domain-Driven Design Reference: Definitions and Pattern Summaries. Dog Ear Publishing.

[59] De Lauretis, L. (2019). From monolithic architecture to microservices architecture. In 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW) (pp. 93-96). IEEE. https://doi.org/10.1109/ISSREW.2019.00050

[60] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday, today, and tomorrow. Present and ulterior software engineering, 195-216. https://doi.org/10.1007/978-3-319-67425-4_12

[61] Vasiloudis, B. I. (2018). Cloud service analysis and design using microservices for cross-platform mobile computing devices.

[62] Autili, M., Perucci, A., & De Lauretis, L. (2020). A hybrid approach to microservices load balancing. Microservices: Science and Engineering, 249-269. https://doi.org/10.1007/978-3-030-31646-4_10

[63] Hamza, M. (2023). Transforming Monolithic Systems to a Microservices Architecture. ACM SIGSOFT Software Engineering Notes, 48(1), 67-69. https://doi.org/10.1145/3573074.3573091

[64] Gos, K., & Zabierowski, W. (2020). The comparison of microservice and monolithic architecture. In 2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH) (pp. 150-153). IEEE. https://doi.org/10.1109/MEMSTECH49584.2020.9109514

[65] Velepucha, V., & Flores, P. (2021). Monoliths to microservices-migration problems and challenges: A sms. In 2021 Second International Conference on Information Systems and Software Technologies (ICI2ST) (pp. 135-142). IEEE. https://doi.org/10.1109/ICI2ST51859.2021.00027

[66] Villamizar, M., Garces, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., ... & Lang, M. (2016). Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures. In 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid) (pp. 179-182). IEEE. https://doi.org/10.1109/CCGrid.2016.37

[67] Schäffer, E., Stiehl, V., Schwab, P. K., Mayr, A., Lierhammer, J., & Franke, J. (2021). Process-driven approach within the engineering domain by combining business

process model and notation (BPMN) with process engines. Procedia CIRP, 96, 207-212. https://doi.org/10.1016/j.procir.2021.01.076

[68] Nemeth, F., Steinert, R., Kreuger, P. & Skoldstrom, P. (2015) Roles of devops tools in an automated, dynamic service creation architecture, in Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on, pp. 1153–1154. https://doi.org/ 10.1109/INM.2015.7140455

[69] Kalske, M., Mäkitalo, N., & Mikkonen, T. (2018). Challenges when moving from monolith to microservice architecture. In Current Trends in Web Engineering: ICWE 2017 International Workshops, Liquid Multi-Device Software and EnWoT, practi-O-web, NLPIT, SoWeMine, Rome, Italy, June 5-8, 2017, Revised Selected Papers 17 (pp. 32-47). Springer International Publishing. https://doi.org/10.1007/978-3-319-74433-9_3

[70] Familiar, B., & Familiar, B. (2015). Service Fabric. Microservices, IoT, and Azure: Leveraging DevOps and Microservice Architecture to Deliver SaaS Solutions, 165-182. https://doi.org/10.1007/978-1-4842-1275-2_8

[71] Tanasseri, N., & Rai, R. (2017). Microservices with Azure. Packt Publishing Ltd.

[72] Bai, H. (2018). Programming Microsoft Azure Service Fabric. Microsoft Press.

[73] Chawla, H., & Kathuria, H. (2019). Building microservices applications on Microsoft azure: designing, Developing, Deploying, and Monitoring. Apress.

[74] Verma, A., Malla, D., Choudhary, A. K., & Arora, V. (2019, February). A detailed study of azure platform & its cognitive services. In 2019 International conference on machine learning, big data, cloud and parallel computing (COMITCon) (pp. 129-134). IEEE. https://doi.org/10.1109/COMITCon.2019.8862178

[75] Kakivaya, G., Xun, L., Hasha, R., Ahsan, S. B., Pfleiger, T., Sinha, R., ... & Gupta, I. (2018). Service fabric: a distributed platform for building microservices in the cloud. In Proceedings of the thirteenth EuroSys conference (pp. 1-15). https://doi.org/10.1145/3190508.3190546

[76] Sahay, R., & Sahay, R. (2020). Service Fabric. Microsoft Azure Architect Technologies Study Companion: Hands-on Preparation and Practice for Exam AZ-300 and AZ-303, 623-659. https://doi.org/10.1007/978-1-4842-6200-9_17

[77] Sbarski, P., & Kroonenburg, S. (2017). Serverless architectures on AWS: with examples using Aws Lambda. Simon and Schuster.

[78] Chapin, J., & Roberts, M. (2020). Programming AWS Lambda: build and deploy serverless applications with Java. O'Reilly Media

[79] Wadia, Y., & Gupta, U. (2017). Mastering AWS Lambda. Packt Publishing Ltd.

[80] Spillner, J., & Dorodko, S. (2017). Java code analysis and transformation into AWS lambda functions. arXiv preprint arXiv:1702.05510. https://doi.org/10.48550/arXiv.1702.05510

[81] Schneid, K., Di Bernardo, S., Kuchen, H., & Thöne, S. (2021). Data-Flow analysis of BPMN-based process-driven applications: detecting anomalies across model and code (No. 38). ERCIS Working Paper.

[82] Geiger, M., Harrer, S., Lenhard, J., Casar, M., Vorndran, A., & Wirtz, G. (2015, March). BPMN conformance in open source engines. In 2015 IEEE Symposium on Service-Oriented System Engineering (pp. 21-30). IEEE. https://doi.org/ 10.1109/SOSE.2015.22

[83] Roman, A. (2021). Adaptable Processes: Concepts, Design and Implementation in Camunda (Doctoral dissertation).

[84] Valderas, P., Torres, V., & Serral, E. (2022). Modelling and executing IoT-enhanced business processes through BPMN and microservices. Journal of Systems and Software, 184, 111139. https://doi.org/10.1016/j.jss.2021.111139

[85] Mass, J., Chang, C., & Srirama, S. N. (2016). Workflow model distribution or code distribution? ideal approach for service composition of the internet of things. In 2016 IEEE International Conference on Services Computing (SCC) (pp. 649-656). IEEE. https://doi.org/10.1109/SCC.2016.90

[86] Montesi, F., & Weber, J. (2016). Circuit breakers, discovery, and API gateways in microservices. arXiv preprint arXiv:1609.05830. https://doi.org/10.48550/arXiv.1609.05830

[87] Zhao, J. T., Jing, S. Y., & Jiang, L. Z. (2018, September). Management of api gateway based on micro-service architecture. In Journal of Physics: Conference Series (Vol. 1087, No. 3, p. 032032). IOP Publishing. https://doi.org/10.1088/1742-6596/1087/3/032032

[88] Gadge, S., & Kotwani, V. (2018). Microservice architecture: API gateway considerations. GlobalLogic Organisations, Aug-2017, 11.

[89] Xu, R., Jin, W., & Kim, D. (2019). Microservice security agent based on API gateway in edge computing. Sensors, 19(22), 4905. https://doi.org/10.3390/s19224905

[90] Chappell, D. (2008). Introducing the Azure services platform. White paper, Oct, 1364(11).

[91] Ali, S. A., & Zafar, M. W. (2021). API GATEWAY ARCHITECTURE EXPLAINED. INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY, 5(1), 506-546. Retrieved from https://www.ijcst.com.pk/index.php/IJCST/article/view/269

[92] Fernandez, A. (2013). Camunda BPM platform loan assessment process lab. Brisbane, Australia: Queensland University of Technology.

[93] Caulfield, C., Maj, S. P., & Veal, D. (2010). Learning Java with Sun SPOTs. In Advanced Techniques in Computing Sciences and Software Engineering (pp. 251-256). Springer Netherlands.

[94] Radhakrishnan, R., Vijaykrishnan, N., John, L. K., Sivasubramaniam, A., Rubio, J., & Sabarinathan, J. (2001). Java runtime systems: Characterization and architectural implications. IEEE Transactions on computers, 50(2), 131-146. https://doi.org/10.1109/12.908989

[95] Ghiya, P. (2018). TypeScript Microservices: Build, deploy, and secure Microservices using TypeScript combined with Node. js. Packt Publishing Ltd.

[96] Delcev, S., & Draskovic, D. (2018). Modern javascript frameworks: A survey study. In 2018 Zooming Innovation in Consumer Technologies Conference (ZINC) (pp. 106-109). IEEE. https://doi.org/10.1109/ZINC.2018.8448444

[97] Hofmann, M., Schnabel, E., & Stanley, K. (2017). Microservices best practices for Java. IBM Redbooks.

[98] Crockford, D. (2008). JavaScript: The Good Parts: The Good Parts. " O'Reilly Media, Inc.".

[99] Van Rossum, G. (2007, June). Python Programming Language. In USENIX annual technical conference (Vol. 41, No. 1, pp. 1-36).

[100] Lutz, M. (2001). Programming python. " O'Reilly Media, Inc.".

[101] Fraser, S., & Ziadé, T. (2021). Python Microservices Development: Build efficient and lightweight microservices using the Python tooling ecosystem. Packt Publishing Ltd.

[102] Hattingh, C. (2020). Using Asyncio in Python: understanding Python's asynchronous programming features. " O'Reilly Media, Inc.".

[103] Ziade, T. (2017). Python Microservices Development: Build, test, deploy, and scale microservices in Python. Packt Publishing Ltd.

[104] Stroustrup, B. (2022). A Tour of C++. Addison-Wesley Professional.

[105] Meyers, S. (2014). Effective modern C++: 42 specific ways to improve your use of C++ 11 and C++ 14. " O'Reilly Media, Inc.".

[106] Schäling, B. (2011). The boost C++ libraries. Boris Schäling.

[107] Ostrowski, A., & Gaczkowski, P. (2021). Software Architecture with C++: Design modern systems using effective architecture concepts, design patterns, and techniques with C++ 20. Packt Publishing Ltd.

[108] Josuttis, N. M. (2012). The C++ standard library: a tutorial and reference. Addison-Wesley.

[109] Kostromin, R., Basharina, O., Feoktistov, A., & Sidorov, I. (2021). Microservice-Based Approach to Simulating Environmentally Friendly Equipment of Infrastructure Objects Taking into Account Meteorological Data. Atmosphere, 12(9), 1217. https://doi.org/10.3390/atmos12091217

[110] Brüggemann, M. E., Vallon, R., Parlak, A., & Grechenig, T. (2014). Modelling microservices in email-marketing concepts, implementation and experiences. In 2014 9th International Conference on Software Paradigm Trends (ICSOFT-PT) (pp. 67-71). IEEE.

[111] Sharma, D., Anandan, R., Manikandan, A., Narayanan, K., & Paul, C. S. (2018). Building micro service for user engagement. Int J Eng Technol, 7, 420-422.

[112] Smid, A., Wang, R., & Cerny, T. (2019). Case study on data communication in microservice architecture. In Proceedings of the Conference on Research in Adaptive and Convergent Systems (pp. 261-267). https://doi.org/10.1145/3338840.3355659

[113] Miao, K., Li, J., Hong, W., & Chen, M. (2020). A microservice-based big data analysis platform for online educational applications. Scientific Programming, 2020, 1-13. https://doi.org/10.1155/2020/6929750

[114] Salah, T., Zemerly, J. M., Yeun, C. Y., Al-Qutayri, M. & Al-Hammadi, Y. (2016). The evolution of distributed systems towards microservices architecture, 11th International Conference for Internet Technology and Secured Transactions (ICITST), pp. 318-325, https://doi.org/10.1109/ICITST.2016.7856721

[115] Ntentos, E., Zdun, U., Plakidas, K., Schall, D., Li, F., Meixner, S. (2019). Supporting Architectural Decision Making on Data Management in Microservice Architectures. In: Bures, T., Duchien, L., Inverardi, P. (eds) Software Architecture. ECSA 2019. Lecture Notes in Computer Science(), vol 11681. Springer, Cham. https://doi.org/10.1007/978-3-030-29983-5_2

.