ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Διπλωματική Εργασία

Μεθοδολογία και Ανάπτυξη Συστημάτων Υπολογιστικής Νέφους και Ελεύθερα Διαθέσιμες Τεχνολογίες Υποστήριξης τους

Συγγραφέας: Ιωάννης Δεληγιάννης
Αριθμός Μητρώου: 151102

Επιβλέπων Καθηγητής: Βασίλειος Μάμαλης
Συν-επιβλέπων: Απόστολος Αναγνωστόπουλος

ΑΘΗΝΑ, ΣΕΠΤΕΜΒΡΙΟΣ 2021

UNIVERSITY OF WEST ATTICA
FACULTY OF ENGINEERING
DEPARTMENT OF INFORMATICS AND COMPUTER ENGINEERING

Diploma Thesis

Methodology and Development of Cloud Native Systems and Free Technologies to Support Them

Author: Ioannis Deligiannis
Registration Number: 151102

Supervisor: Vasileios Mamalis
Co-supervisor: Apostolos Anagnostopoulos

ATHENS, SEPTEMBER 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Μεθοδολογία και Ανάπτυξη Συστημάτων Υπολογιστικής Νέφους και Ελεύθερα Διαθέσιμες Τεχνολογίες Υποστήριξης τους

Συγγραφέας: Ιωάννης Δεληγιάννης
Αριθμός Μητρώου: 151102
Επιβλέπων Καθηγητής: Βασίλειος Μάμαλης
Συν-επιβλέπων: Απόστολος Αναγνωστόπουλος

Η διπλωματική εργασία εξετάστηκε επιτυχώς από την κάτωθι Εξεταστική Επιτροπή:

Α/α	Ονοματεπώνυμο	Βαθμίδα/Ιδιότητα	Ψηφιακή Υπογραφή
1	ΚΑΡΚΑΖΗΣ ΠΑΝΑΓΙΩΤΗΣ	Επίκουρος Καθηγητής Πανεπιστημίου Δυτικής Αττικής	Panagiotis Karkazis Digitally signed by Panagiotis Karkazis Date: 2021.10.23 15:55:20 +03'00'
2	ΜΑΜΑΛΗΣ ΒΑΣΙΛΕΙΟΣ	Καθηγητής Πανεπιστημίου Δυτικής Αττικής	Vasileios Mamalis Digitally signed by Vasileios Mamalis Date: 2021.10.17 20:53:51 +03'00'
3	ΜΠΟΓΡΗΣ ΑΝΤΩΝΗΣ	Καθηγητής Πανεπιστημίου Δυτικής Αττικής	Antonios Bogris Digitally signed by Antonios Bogris Date: 2021.10.18 10:51:19 +03'00'

(κενή σελίδα)

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

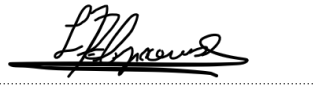
Ο κάτωθι υπογεγραμμένος **Ιωάννης Δεληγιάννης** του **Δημητρίου**, με αριθμό μητρώου **151102** φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής **Μηχανικών** του Τμήματος **Μηχανικών Πληροφορικής και Υπολογιστών**, δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της πτυχιακής/διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο/Η Δηλών/ούσα

(υπογραφή)



Ιωάννης Δεληγιάννης

Περίληψη

Με την ανάπτυξη σύγχρονων μορφών αυτοματοποίησης υπηρεσιών μέσω υπολογιστικών συστημάτων προέκυψαν νέες προκλήσεις στον σχεδιασμό και την ανάπτυξη λογισμικού. Σε επίπεδο υπολογιστικών πόρων υπάρχει μεγάλη ανάπτυξη και επενδύσεις από παρόχους υπηρεσιών υπολογιστικής νέφους για την διάθεση υπηρεσιών υποστήριξης μέσω διαδικτύου που καλύπτουν τις νέες απαιτήσεις. Οι υπηρεσίες υπολογιστικής νέφους αξιοποιούνται από πελάτες οι οποίοι δεν διαθέτουν την απαραίτητη τεχνολογική υποδομή για την υποστήριξη των συστημάτων τους ή δεν θέλουν να επενδύσουν στην υποδομή, διαχείριση και συντήρηση που εκείνη απαιτεί. Τέλος, οι νέες μορφές ανάπτυξης εφαρμογών προκάλεσαν την ανάγκη δημιουργίας νέων μεθοδολογιών που πρέπει να τηρούνται στην δημιουργία συστημάτων τα οποία θα αξιοποιούν στο έπακρο τις νέες τεχνολογίες νέφους.

Η παρούσα διπλωματική αποτελείται από τρία βασικά μέρη. Στο πρώτο μέρος αποτυπώνεται το κίνητρο που ωθεί τους οργανισμούς στην εγκατάλειψη της μονολιθικής αρχιτεκτονικής συστημάτων και δίνει έμφαση στην υιοθέτηση τεχνολογιών υπολογιστικής νέφους και στις νέες δυνατότητες που προσφέρουν. Στο δεύτερο μέρος πραγματοποιείται μελέτη και σύγκριση μεταξύ δωρεάν προσφερόμενων τεχνολογιών νέφους από τους τρεις δημοφιλέστερους πάροχους υπηρεσιών (Google, Microsoft, Amazon), και ακολουθεί αξιολόγηση αυτών με βασικά κριτήρια την υψηλότερη δωρεάν διαθεσιμότητα και την υποστήριξη των περισσότερων cloud-agnostic τεχνολογιών. Στο τρίτο και τελευταίο μέρος ακολουθεί η σχεδίαση και ανάπτυξη μίας cloud-native εφαρμογής αυτοματοποίησης της διαδικασίας εγγραφής φοιτητών σε εργαστήρια, η οποία βασίζεται αρχιτεκτονικά στην μεθοδολογία 12 παραγόντων. Η εφαρμογή αποτελείται από τέσσερεις μικρουπηρεσίες ανεπτυγμένες στο framework Spring Boot που αναπαριστούν το API του συστήματος, την εφαρμογή pwa που θα χρησιμοποιούν οι τελικοί χρήστες υλοποιημένη στο framework React.js, και δύο υποστηρικτικές υπηρεσίες, την μη-σχεσιακή βάση δεδομένων MongoDB και μία υπηρεσία μεσίτη μηνυμάτων RabbitMQ. Η εφαρμογή αυτή αναπτύσσεται αρχικά σε τοπικό περιβάλλον ανάπτυξης με χρήση του Docker, και ύστερα με την πλατφόρμα ενορχήστρωσης κιβωτίων Kubernetes. Στην συνέχεια, πραγματοποιείται επιλογή κάποιων από τις διαθέσιμες δωρεάν τεχνολογίες που μελετήθηκαν στο δεύτερο μέρος της διπλωματικής και χρησιμοποιούνται για την υποστήριξη της εφαρμογής σε περιβάλλον νέφους. Οι υπηρεσίες παρόχων που επιλέγονται είναι οι Azure Cosmos DB για την MongoDB, Azure App Service για την φιλοξενία των μικρουπηρεσιών και ένα Google Cloud VM instance για την φιλοξενία του RabbitMQ.

Λέξεις Κλειδιά: Μονολιθική Αρχιτεκτονική, Υπολογιστική Νέφους, Αρχιτεκτονικές Υπολογιστικής Νέφους, Κιβώτια, Ενορχήστρωση Κιβωτίων, Μεθοδολογία 12 παραγόντων, Μικρουπηρεσίες, Πάροχοι Υπηρεσιών Νέφους, Docker, Kubernetes, Vendor lock-in, Cloud native, Google Cloud, Microsoft Azure, AWS, Free Tier

Abstract

With the development of modern forms of service automation through computer systems, new challenges in software design and development have emerged. On a computing resources level, there is great development and investment by cloud computing service providers to provide new online services that cover the latest needs. Cloud computing services are utilized by customers who do not have the necessary technological infrastructure to support their systems or do not want to invest in the infrastructure, management and maintenance that it requires. Finally, these new forms of application development in the cloud have created the need for new methodologies that must be followed in order to create applications that fully utilize and exploit cloud technologies.

This diploma thesis consists of three main parts. The first part reflects the motivation that pushes organizations to abandon monolithic architectures and emphasizes on the adoption of cloud technologies and the new opportunities they offer. In the second part, a study and comparison was made between free cloud services offered by the three most popular cloud providers (Google, Microsoft, Amazon), then, an evaluation was followed with the key criteria being the highest free availability and the support of the most cloud-agnostic technologies. The third and the final part follows the design and development of a cloud-native application that automates the student enrollment process in laboratories, which is architecturally based on the 12-factor methodology. The application consists of four microservices developed in the Spring Boot framework that consist the system's API, the web application that end-users will use, implement in the React.js framework, and two backing services, the No-SQL MongoDB database and RabbitMQ as a message broker. The cloud native application is initially deployed in a local developement environment using Docker, then, it's deployed using the Kubernetes platform. Last, some of the available free cloud services that were studied in the second chapted are selected and used to deploy the application in a hosted cloud environment. The selected services are Azure Cosmos DB for MongoDB, Azure App Service for the microservices and a Google Cloud VM for hosting RabbitMQ.

Keywords: Monolithic Architecture, Cloud Computing, Cloud Native Architecture, 12-factor Methodology, Microservices, Containers, Container Orchestration, Docker, Kubernetes, Cloud Providers, Vendor lock-in, Google Cloud, Microsoft Azure, AWS, Free Tier

Ευχαριστίες

Η παρούσα διπλωματική πραγματεύεται αντικείμενα ενασχόλησης ιδιαίτερου ενδιαφέροντος για εμένα, και θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου, κ. Βασίλειο Μάμαλη που με στήριξε και με εμπιστεύτηκε στην ανάπτυξη της.

Επίσης, θα ήθελα να ευχαριστήσω τον συν-επιβλέπων καθηγητή μου, κ. Απόστολο Αναγνωστόπουλο, του οποίου οι συμβουλές και οι οδηγίες με βοήθησαν στην ορθή και πλήρη ανάπτυξη του θέματος της διπλωματικής.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου και τους φίλους μου που με υποστήριξαν και με ενθάρρυναν καθ όλη την διάρκεια των σπουδών μου, και κυρίως την μητέρα μου, με την βοήθεια της οποίας πραγματοποιήθηκαν οι σπουδές μου.

Ενότητες

Κατάλογος Διαγραμμάτων

Κατάλογος Πινάκων

Συντομογραφίες και Ακρωνύμια

SaaS	Software as a Service
PaaS	Platform as a Service
CaaS	Container as a Service
IaaS	Infrustructure as a Service
FaaS	Functions as a Service
CN	Cloud Native
K8s	Kubernetes
VM	Virtual Machine
pwa	Progressive web app
NIST	National Institute of Standards and Technology

Κεφάλαιο 1: Εισαγωγή

Οι σύγχρονες μορφές αυτοματοποίησης διαδικασιών μέσω του διαδικτύου ωθούν τους περισσότερους οργανισμούς στην δημιουργία διαδικτυακού λογισμικού το οποίο θα καλύπτει τις ανάγκες τους. Οι νέες απαιτήσεις λογισμικού ως προς την υψηλή διαθεσιμότητα καθώς και στο κόστος διαχείρισης και συντήρησης αυτού, δημιουργεί νέα προβλήματα στην βιομηχανία λογισμικού. Οι τεχνολογικοί κολοσσοί αντιλαμβανόμενοι τις νέες ανάγκες υποστήριξης εφαρμογών και εκμεταλλευόμενοι τις δυνατότητες του διαδικτύου ξεκίνησαν να επενδύουν στην παροχή υπολογιστικών πόρων και υπηρεσιών με τεχνολογίες υπολογιστικής νέφους. Η χρήση τεχνολογιών υπολογιστικής νέφους για την ανάπτυξη και υποστήριξη λογισμικού ενώ έφερε σημαντικά πλεονεκτήματα, αποκάλυψε ότι οι παραδοσιακές αρχιτεκτονικές και μεθοδολογίες που τηρούνται στον σχεδιασμό λογισμικού δεν είναι αρκετές για την υποστήριξη και πλήρη εκμετάλλευση των νέων δυνατοτήτων που παρέχονται. Για τον σκοπό αυτό αναπτύχθηκαν νέες αρχιτεκτονικές και μεθοδολογίες ανάπτυξης λογισμικού οι οποίες εκμεταλλεύονται πλήρως τις δυνατότητες της υπολογιστικής νέφους.

Στόχος

Στόχος της παρούσας διπλωματικής είναι η ανάδειξη των αδυναμιών που παρουσιάζουν οι μονολιθικές αρχιτεκτονικές έναντι αρχιτεκτονικών υπολογιστικής νέφους, οι δυνατότητες των δωρεάν τεχνολογιών που προσφέρουν οι πάροχοι υπηρεσιών, τα πλεονεκτήματα των τεχνολογιών που υποστηρίζουν τις νέες αρχιτεκτονικές τόσο σε τοπικό περιβάλλον όσο και προσφερόμενες από τους παρόχους υπηρεσιών, και η παρουσίαση της διαδικασίας υλοποίησης ενός συστήματος υπολογιστικής νέφους το οποίο ακολουθεί τις νέες μεθοδολογίες.

Η Δομή της Διπλωματικής

Κεφάλαιο 1: Εισαγωγή

Στο 1ο κεφάλαιο ακολουθεί η παρουσίαση του αντικειμένου ενασχόλησης, κατά το οποίο αποτυπώνονται οι βασικές σκέψεις που κατευθύνουν τις εταιρείες στην ανάπτυξη λογισμικού με τεχνολογίες υπολογιστικής νέφους και παρουσιάζεται ο στόχος και η δομή της παρούσας διπλωματικής.

Κεφάλαιο 2: Υπόβαθρο και Κίνητρο

Στο 2ο κεφάλαιο παρουσιάζεται το βασικό θεωρητικό υπόβαθρο που πρέπει να κατέχει ο αναγνώστης για την κατανόηση του κινήτρου που ωθεί τον στόχο της διπλωματικής, και των τεχνολογιών και μεθοδολογιών που ακολουθούνται στα επόμενα κεφάλαια. Ειδικότερα, στο κεφάλαιο παρουσιάζεται η μονολιθική αρχιτεκτονική και γίνεται ανάλυση των αδυναμιών της, οι τεχνολογίες υπολογιστικής

νέφους, αναλύονται οι διάφοροι τύποι υποδομών υποστήριξης συστημάτων, μεθοδολογίες που υποστηρίζουν τις τεχνολογίες υπολογιστικής νέφους και βασικές γνώσεις σε υποστηρικτικές υπηρεσίες που θα χρησιμοποιηθούν.

Κεφάλαιο 3: Μελέτη Δωρεάν Υπηρεσιών Υπολογιστικής Νέφους

Στο 3ο κεφάλαιο πραγματοποιείται μελέτη σύγκρισης και αξιολόγησης μεταξύ των τεχνολογιών που προσφέρουν οι δημοφιλέστεροι πάροχοι υπηρεσιών υπολογιστικής νέφους (Microsoft, Google, Amazon), θα αποτυπωθούν τα κριτήρια επιλογής και αξιολόγησης αυτών και θα παρουσιαστούν τα τελικά συμπεράσματα που προκύπτουν με την διεκπεραίωση της μελέτης.

Κεφάλαιο 4: Ανάπτυξη Συστήματος Υπολογιστικής Νέφους

Στο 4ο κεφάλαιο παρουσιάζεται μελέτη ανάπτυξης ενός συστήματος υποβοήθησης της διαδικασίας εγγραφής φοιτητών σε εργαστήρια. Βασικός σκοπός του κεφαλαίου είναι η υλοποίηση και σχεδιασμός της εφαρμογής με τήρηση των μεθοδολογιών συστημάτων υπολογιστικής νέφους. Η ανάπτυξη του συστήματος θα γίνει σε τοπικό επίπεδο με τεχνολογίες που θα επιλεγούν εφόσον πληρούν τα κριτήρια των ακολουθούμενων μεθοδολογιών. Στην συνέχεια θα επιλεγούν τεχνολογίες που βρέθηκαν στο 3ο κεφάλαιο και θα παρουσιαστούν οι αλλαγές που πρέπει να γίνουν στο σύστημα για την ανάπτυξη του σε περιβάλλον παραγωγής. Τέλος, θα ακολουθήσουν παραδείγματα δοκιμών της εφαρμογής.

Κεφάλαιο 5: Συμπεράσματα

Στο 5ο κεφάλαιο αποτυπώνονται τα τελικά συμπεράσματα που διεξάγονται από το 3ο και 4ο κεφάλαιο Μελέτης και Ανάπτυξης, αφού έχει υλοποιηθεί το σύστημα με τεχνολογίες υπολογιστικής νέφους. Παρουσιάζονται προτάσεις για τροποποιήσεις, σημεία τα οποία απαιτείται περισσότερη προσοχή και διεξάγονται γενικά συμπεράσματα για το μέλλον παρόμοιων συστημάτων, τεχνολογιών και σχεδιαστικών επιλογών.

Κεφάλαιο 6: Βιβλιογραφία

Στο 6ο κεφάλαιο παρατίθενται η βιβλιογραφία και οι πηγές με τις οποίες έγινε η μελέτη και συγγραφή της παρούσας διπλωματικής.

Κεφάλαιο 7: Παράρτημα

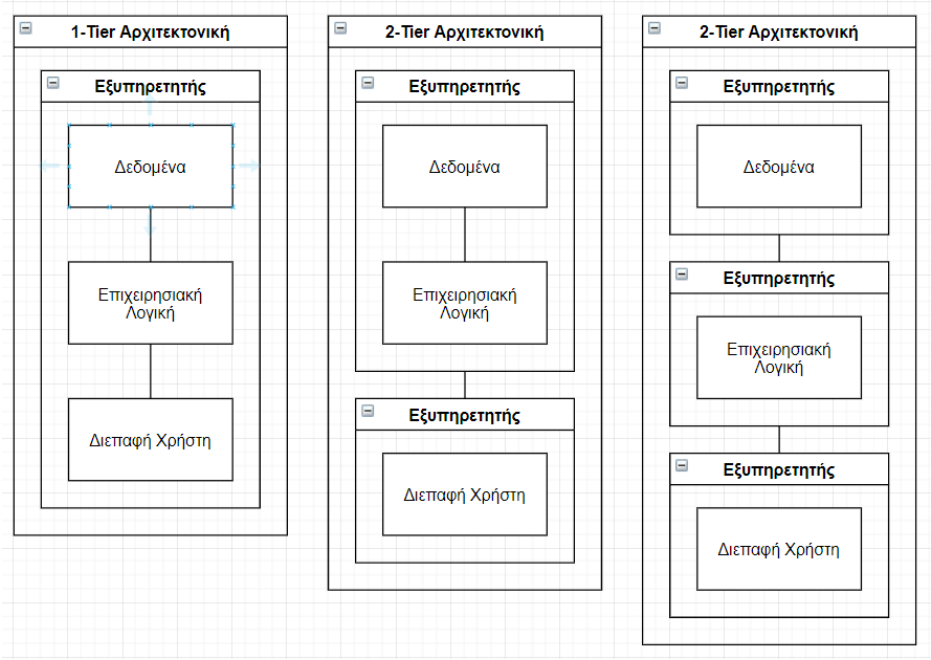
Στο 7ο κεφάλαιο παρατίθενται τα λινκ των repositories που περιέχουν τον πηγαίο κώδικα της εφαρμογής που δημιουργήθηκε στο 4ο κεφάλαιο.

Κεφάλαιο 2: Υπόβαθρο και Κίνητρο

Σε αυτό το κεφάλαιο θα τεθεί το βασικό υπόβαθρο που απαιτείται για την κατανόηση των τεχνολογιών που θα μελετηθούν στο κεφάλαιο 3, καθώς και αυτών που θα εφαρμοστούν στο κεφάλαιο 4 για των σχεδιασμό και την υλοποίηση ενός clout native συστήματος. Επίσης, μέσα από αυτό το κεφάλαιο θα αποτυπωθεί το κίνητρο που ωθεί τους μηχανικούς στον σχεδιασμό συστημάτων με τεχνολογίες υπολογιστικής νέφους.

Η Μονολιθική Προσέγγιση Ανάπτυξης Λογισμικού

Ένα από τα βασικά μέρη του σχεδιασμού ενός συστήματος είναι η αρχιτεκτονική του. Η αρχιτεκτονική καθορίζει την δομή και τα τμήματα μίας εφαρμογής, την λειτουργικότητα, την συμπεριφορά και την αλληλεπίδραση μεταξύ τους [1]. Στον βιβλίο "Τεχνολογία Λογισμικού" των Μανόλης Γιακουμάκης, Νίκος Διαμαντίδης, 2009 [2], αναφέρεται πως με τον παραδοσιακό τρόπο ανάπτυξης λογισμικού, υπάρχουν τρεις βασικές κλάσεις αρχιτεκτονικής διαδικτυακών εφαρμογών που καθορίζονται από τον αριθμό των επιπέδων που χωρίζουν τον χρήστη με τα δεδομένα. Τα επίπεδα αυτά χωρίζονται στα δεδομένα, την επιχειρησιακή λογική και την διεπαφή χρήστη. Οι βασικές αυτές κλάσεις είναι οι 1-Tier, 2-Tier και 3-Tier, με κάθε tier να σημαίνει ότι το κάθε επίπεδο τρέχει σε διαφορετικό σύστημα ή διαφορετικές διεργασίες από τα υπόλοιπα.



Σχέδιο 1: Διαγράμματα μονολιθικής αρχιτεκτονικής 1-Tier, 2-Tier, 3-Tier.

Σε ένα σύστημα με 1-Tier αρχιτεκτονική η διεπαφή χρήστη, η λογική και τα δεδομένα συνυπάρχουν σε ένα επίπεδο ενοποιημένα. Σε συστήματα 2-Tier η διαχείριση των δεδομένων γίνεται σε ξεχωριστό σύστημα από αυτό που περιέχει την διεπαφή χρήστη και την λογική. Και τέλος, σε ένα 3-Tier σύστημα η διαχείριση των δεδομένων, η διεπαφή και η λογική τρέχουν σε ξεχωριστά συστήματα.

Οι παραπάνω αρχιτεκτονικές, με το πέρας του χρόνου φάνηκαν αδύναμες μπροστά στις νέες απαιτήσεις των τελευταίων χρόνων. Τα συστήματα όσο μεγάλωναν σε κάθε ένα από τα τρία επίπεδα η διαχείριση τους γινόταν εξαιρετικά δύσκολη καθώς η δομή τους γινόταν περίπλοκη. Εξαιτίας τις βασικής τους δομής, δεν μπορούν να επιμεριστούν και έτσι η αρχιτεκτονικές αυτές χαρακτηρίζονται και μονολιθικές.

Στην μονολιθική αρχιτεκτονική, τα επιμέρους στοιχεία του συστήματος συνυπάρχουν σε ένα περιβάλλον που το διαχειρίζεται μία ομάδα. Ο Chris Richardson, στο άρθρο του "Pattern: Monolithic Architecture" [3] αναλύει διάφορα προβλήματα με τα οποία έρχονται αντιμέτωποι οι μηχανικοί λογισμικού που εργάζονται σε μονολιθικά συστήματα, με κάποια από αυτά να αναγράφονται παρακάτω:

Μεγάλα συστήματα είναι δύσκολα κατανοητά από τους προγραμματιστές. Το σύστημα γίνεται αρκετά περίπλοκο για να κατανοηθεί και να τροποποιηθεί. Ως αποτέλεσμα, η παραγωγή νέου κώδικα παρουσιάζει πτώση.

Η αξιοπιστία του συστήματος μειώνεται καθώς εάν προκύψει κάποιο σφάλμα υπάρχει περίπτωση να αποτρέψει την ομαλή λειτουργία ολόκληρου του συστήματος. Επίσης, η αποσφαλμάτωση του γίνεται δύσκολη διότι πρέπει να ακολουθηθεί ολόκληρη η ουρά εντολών που εκτελέστηκε σε αυτό από όλα τα επίπεδα του.

Υπερφόρτωση του περιβάλλοντος ανάπτυξης (IDE). Συνήθως το IDE γίνεται αργότερο, απαιτεί περισσότερους πόρους να φορτωθεί και καθυστερεί τους προγραμματιστές.

Η συνεχής ανάπτυξη (Continuous Deployment – CD) γίνεται δύσκολη. Ένα σύνθετο μονολιθικό σύστημα δεν μπορεί να αναπτύσσεται (deployed) πολύ συχνά διότι θα πρέπει να χτιστεί και να αναπτυχθεί από την αρχή. Αυτή η καθυστέρηση θα καθυστερεί διεργασίες και θα παραμένει αρκετή ώρα ανενεργό.

Η κλιμάκωση του συστήματος σε αντίγραφα παρουσιάζει δυσκολίες. Εξαιτίας της δομής του, ένα μονολιθικό σύστημα μπορεί να κλιμακωθεί μόνο ολόκληρο σε όλα του τα συστατικά. Αυτό σημαίνει ότι θα υπάρχουν τμήματα με αρκετούς πόρους που όμως θα χρησιμοποιούνται ελάχιστα, επομένως θα έχουμε σπατάλες. Επίσης, θα υπάρχουν προβλήματα συγχρονισμού των δεδομένων καθώς κάθε αντίγραφο του συστήματος θα "βλέπει" διαφορετικά δεδομένα.

Παρουσίαση δυσκολιών σε νέες προσθήκες. Σε μεγάλα συστήματα με διαφορετικές ομάδες προγραμματιστών να εργάζονται πάνω σε διαφορετικά τμήματα του κώδικα, οι ομάδες δεν μπορούν να λειτουργούν ανεξάρτητα μεταξύ τους καθώς πρέπει να συγχρονίζουν τις προσθήκες που ανεβάζουν στον κώδικα παραγωγής.

Οι τεχνολογίες που έχουν επιλεγεί και σε αυτές έχει χτιστεί το σύστημα είναι αρκετά δύσκολο να αλλάξουν. Σε μεγάλα μονολιθικά συστήματα είναι σχεδόν αδύνατον να αλλάξει μία τεχνολογία που χρησιμοποιείται καθώς σε αρκετές περιπτώσεις τα διαφορετικά τμήματα (layers) του κώδικα είναι συνδεδεμένα μεταξύ τους.

Υπολογιστική Νέφους (Cloud Computing)

Η έννοια του υπολογιστικού νέφους εμφανίστηκε την δεκαετία του 1950 σε εκπαιδευτικά ινστιτούτα και εταιρείες στα οποία υπήρχαν ισχυροί κεντρικοί υπολογιστές, οι οποίοι διαμοίραζαν τις δυνατότητες τους μέσα από τερματικά [5]. Τα τερματικά δεν είχαν μνήμη και επεξεργαστική ισχύ, παρά μόνο μέσω του δικτύου επικοινωνούσαν με τον κεντρικό εξυπηρετητή που αναλάμβανε να διεκπεραιώσει τα αιτήματα των τελικών χρηστών. Με την ανάπτυξη των εικονικών μηχανών και του ίντερνετ, κολοσσοί της πληροφορικής όπως η Microsoft και η Google εκμεταλλεύτηκαν αυτές τις τεχνολογίες και ανέπτυξαν υποδομές και υπηρεσίες τις οποίες μπορούν να χρησιμοποιούν τελικοί χρήστες μέσω διαδικτύου. Έτσι, το υπολογιστικό νέφος ορίζεται ως ένα μοντέλο διάθεσης υπολογιστικών πόρων μέσω διαδικτύου και αποτελείται από ένα σύνολο κατανεμημένων υπολογιστικών συστημάτων οργανωμένα σε κέντρα δεδομένων και εξυπηρετητών κατάλληλα για προσαρμογή στις ανάγκες και προδιαγραφές του χρήστη. Οι πόροι και οι υπηρεσίες προσφέρονται κατ' αίτηση του χρήστη για συγκεκριμένο χρόνο και κόστος.

Με την συνεχώς αυξανόμενη ανάγκη για υπολογιστικούς πόρους που μπορούν να διαχειριστούν μεγάλες ποσότητες δεδομένων, οι εταιρείες αναγκάζονταν να αγοράζουν και να οργανώνουν μόνες τους τις απαιτούμενες υποδομές για την υποστήριξη των συστημάτων τους, αναλαμβάνοντας και όλη την διαχείριση και συντήρησή τους. Με την ανάπτυξη των υπηρεσιών νέφους όμως οι εταιρείες ξεκίνησαν να αξιοποιούν αυτές τις υπηρεσίες καθώς το κόστος τους σε σχέση με την αναλαβή ενός τέτοιου έργου από τις ίδιες ήταν μικρότερο.

Σύμφωνα με το NIST [4], το μοντέλο υπολογιστικής νέφους αποτελείται από 3 βασικά μοντέλα υπηρεσιών, 5 βασικά χαρακτηριστικά και 4 μοντέλα ανάπτυξης, τα οποία θα αναλυθούν παρακάτω.

Μοντέλα Υπηρεσιών (Service Models)

Καθώς η υπολογιστική νέφους προσφέρει υπηρεσίες και πόρους μέσω διαδικτύου, προκύπτουν ποικίλες μορφές υπηρεσιών που ξεκινούν από τις έτοιμες λύσεις λογισμικού μέχρι την πρωταρχική υποδομή υποστήριξης συστημάτων. Οι Nane Kratzke και Rene Peinl στην εργασία τους με τίτλο "CloudNS - A Cloud-native Application Reference Model for Enterprise Architects" [6]

κατηγοριοποιούν τις τεχνολογίες που προσφέρονται βάση του μοντέλου αναφοράς ανοικτής διασύνδεσης συστημάτων (OSI), προσφέροντας έναν εύκολο τρόπο κατηγοριοποίησης των υπηρεσιών νέφους. Το μοντέλο OSI είναι ένα ευρέος αποδεκτό αφηρημένο μοντέλο για τη σχεδίαση τηλεπικοινωνιακών και δικτυακών πρωτοκόλλων. Το μοντέλο OSI αποτελείται από 7 αφηρημένα επίπεδα και σε κάποια μπορούν να αντιστοιχηθούν οι ανάλογες τεχνολογίες υπολογιστικής νέφους που παρέχονται.

Επίπεδο OSI	**Υπηρεσία νέφους**	**Βασική Λειτουργία**	**Παράδειγμα Τεχνολογίας**
7 - Εφαρμογής 6 - Παρουσίασης	SaaS	• Λειτουργίες τελικού χρήστη	
5 - Συνόδου	PaaS	• Υπηρεσίες βάσεων δεδομένων και αποθηκευτικού χώρου • Υπηρεσίες κατανεμημένων προτύπων και λειτουργιών	• RabbitMQ • Hadoop • SpringCloud
4 - Μεταφοράς	CaaS	• Υποδομή διασύνδεσης, διαχείρισης και ενορχήστρωσης υπηρεσιών	• Kubernetes • Swarm
3 - Δικτύου 2 – Ζεύξης δεδομένων 1- Φυσικό	IaaS	• Παροχή φυσικών και εικονικών μηχανών • Παροχή λειτουργικού συστήματος υποστήριξης εφαρμογών	• Docker • Linux • Physical Machines

Πίνακας 1: Μοντέλα υπηρεσιών υπολογιστικής νέφους

Εκτός από τα βασικά μοντέλα υπηρεσιών, αξίζει να σημειωθεί ότι υπάρχουν περισσότερα τα οποία προστίθενται και προκύπτουν από τις νέες ανάγκες των καταναλωτών. Οτιδήποτε μπορεί να προσφερθεί σαν υπηρεσία από πάροχους μπορεί να ανήκει σε αυτή την κατηγορία. Παρακάτω θα ακολουθήσουν τα βασικά μοντέλα σύμφωνα με το NIST [4].

Λογισμικό σαν Υπηρεσία (Software as a Service – SaaS)

Το Λογισμικό σαν Υπηρεσία (SaaS) είναι μία υπηρεσία υπολογιστικής νέφους κατά την οποία οι πάροχοι υπολογιστικής νέφους προσφέρουν τελικές λύσεις εφαρμογών και συστημάτων. Υπάρχουν συστήματα που δεν χρειάζεται να αναπτυχθούν από εταιρείες καθώς έχουν την ίδια λειτουργικότητα και απαιτήσεις με πολλά υπαρκτά. Επομένως, τέτοιου είδους συστήματα συνήθως οι εταιρείες επιλέγουν να τα χρησιμοποιούν έτοιμα και να τα ενσωματώνουν στο δικό τους σύστημα. Σε αυτές τις υπηρεσίες ο καταναλωτής δεν απαιτείται να έχει κανενός είδους υποδομή όπως δίκτυο, εξυπηρετητές, λειτουργικό σύστημα, περιβάλλον ανάπτυξης κ.α, καθώς οι εφαρμογή προσφέρεται ανεπτυγμένη.

Πλατφόρμες σαν Υπηρεσία (Platforms as a Service – PaaS)

Οι Πλατφόρμες σαν Υπηρεσία (PaaS) είναι μία υπηρεσία υπολογιστικής νέφους κατά την οποία οι πάροχοι υπολογιστικής νέφους προσφέρουν λογισμικό και λειτουργικό κατάλληλο για την δημιουργία και λειτουργία συστημάτων εφαρμογών. Σε αυτή την υπηρεσία παρέχονται εργαλεία αυτοματοποίησης της διαδικασίας κύκλου ανάπτυξης και παραγωγής λογισμικού. Οι εταιρείες που επιλέγουν αυτή την υπηρεσία θα πρέπει να εγκαταστήσουν την εφαρμογή και τα δεδομένα της σε αυτή, αλλά δεν χρειάζεται να ασχοληθούν με την διαχείριση της υποδομής όπως το λειτουργικό περιβάλλον, το περιβάλλον εκτέλεσης, τους εξυπηρετητές, τον αποθηκευτικό χώρο και το δίκτυο, αλλά συγκεντρώνονται στην παραγωγή και ανάπτυξη του συστήματος.

Υποδομές σαν Υπηρεσία (Infrastructure as a Servce – IaaS)

Οι Υποδομές σαν Υπηρεσία (IaaS) είναι μία υπηρεσία υπολογιστικής νέφους κατά την οποία οι πάροχοι υπολογιστικής νέφους προσφέρουν υπολογιστικούς πόρους όπως επεξεργαστική ισχύ, αποθηκευτικό χώρο, δίκτυα διασύνδεσης και εξυπηρετητές. Οι εταιρείες μπορούν να επιλέξουν να χρησιμοποιήσουν αυτή την υπηρεσία για να διαχειριστούν και να τρέξουν το λογισμικό τους όταν δεν διαθέτουν αρκετές υποδομές για να υποστηρίξουν τις απαιτήσεις τους και δεν χρειάζονται αυτούς τους πόρους πάντα αλλά μόνο όταν απαιτείται υψηλότερο φόρτο εργασίας. Οι εταιρείες που θα επιλέξουν αυτού του είδους υπηρεσίες θα έχουν στην διάθεσή τους εξυπηρετητές, αποθηκευτικό χώρο και δικτύωση, αλλά θα πρέπει να φροντίσουν για το λειτουργικό σύστημα των μηχανών και το λογισμικό περιβάλλον που θα τρέξει η εφαρμογή και τα δεδομένα της. Βασικά χαρακτηριστικά αυτών των υπηρεσιών είναι η χρησιμοποίηση και πληρωμή κατ' απαίτηση και ανάγκη πόρων, και η μη αναγκαιότητα συντήρησης.

Μοντέλα Ανάπτυξης (Deployment Models)

Ιδιωτικά Δίκτυα Νέφους (Private Cloud)

Αυτή η υποδομή νέφους παρέχεται για χρήση αποκλειστικά από έναν οργανισμό. Η ιδιοκτησία και η διαχείριση αυτού μπορεί να ανήκει στον ίδιο τον οργανισμό, σε τρίτα πρόσωπα ή σε συνδυασμό αυτών.

Δίκτυα Νέφους Κοινοτήτων (Community Cloud)

Αυτή η υποδομή νέφους απευθύνεται σε συγκεκριμένη κοινότητα καταναλωτών από οργανισμούς με κοινούς στόχους. Μπορεί να ανήκει και να την διαχειρίζονται οργανισμοί που ανήκουν στην κοινότητα αυτή ή σε τρίτα πρόσωπα.

Δημόσια Δίκτυα Νέφους (Public Cloud)

Η υποδομή δημόσιου νέφους παρέχεται για χρήση ανοιχτή χρήστη προς το κοινό. Η ιδιοκτησία και η διαχείριση της μπορεί να ανήκουν σε κάποιον οργανισμό, κρατική υπηρεσία, τρίτα πρόσωπα ή κάποιο συνδυασμό αυτών.

Υβριδικά Δίκτυα Νέφους (Hybrid Cloud)

Η υβριδική υποδομή νέφους αποτελείται από έναν συνδυασμό από 2 ή περισσότερα μοντέλα ανάπτυξης (Ιδιωτικό, Δημόσιο) που αποτελούν μοναδικές οντότητες, αλλά είναι δεμένα μεταξύ τους με τεχνολογίες και αρχιτεκτονικές ικανές να τα θέσουν ικανά για επικοινωνία μεταξύ τους.

Βασικά Χαρακτηριστικά

Χρήση κατ' απαίτηση (On-demand self-service)

Ο χρήστης χρησιμοποιεί κατ' απαίτηση υπολογιστικούς πόρους όπως επεξεργαστική ισχύ, αποθηκευτικό χώρο και χρόνο εξυπηρετητών και χρεώνεται μόνο για το διάστημα που του παρέχονται. Μόλις διακοπή η χρήση η χρέωση σταματά αυτόματα.

Ευρεία πρόσβαση δικτύου (Broad network access)

Οι παροχές προσφέρονται στο δίκτυο και είναι προσβάσιμες ανεξαρτήτως συσκευής. Επομένως μπορούν να χρησιμοποιηθούν από κινητές συσκευές, tablet, Η/Υ, κ.α.

Συγκέντρωση πόρων (Resource pooling)

Οι υπολογιστικοί πόροι του παρόχου συγκεντρώνονται και εξυπηρετούν πολλαπλούς καταναλωτές με διαφορετικούς φυσικούς και εικονικούς πόρους οι οποίοι είναι δυναμικά καταχωρημένοι ανάλογα τις ανάγκες τους. Ο καταναλωτής δεν έχει τον έλεγχο και την γνώση των πόρων και της τοποθεσίας τους.

Βέλτιστη ελαστικότητα (Rapid elasticity)

Οι δυνατότητες των πόρων που προσφέρονται στους καταναλωτές καταχωρούνται και απελευθερώνονται άμεσα και αυτόματα. Με βασικό δεδομένο τα όρια ελάχιστης και μέγιστης εκχώρησης πόρων καθώς και το κόστος αυτών, οι δυνατότητες φαίνονται απεριόριστες και κατάλληλες για κάθε συνθήκη.

Μετρούμενες υπηρεσίες (Measured service)

Τα συστήματα υπολογιστικής νέφους ελέγχουν και βελτιστοποιούν αυτόματα πόρους που χρησιμοποιούνται εκμεταλλευόμενα διάφορες μετρικές χρήστης σε κάποια αφηρημένη μορφή ανάλογα το είδος της υπηρεσίας όπως ο χώρος αποθήκευσης, η επεξεργαστική ισχύ, εύρος ζώνης καναλιών επικοινωνίας, μέγιστων αριθμός ενεργών χρηστών κ.α.

Προκλήσεις στην Χρησιμοποίηση Τεχνολογιών Υπολογιστικής Νέφους

Σε αυτή την ενότητα θα παρουσιαστούν μερικά βασικά προβλήματα και προκλήσεις με τα οποία είναι πιθανών να έρθουν αντιμέτωποι οι πελάτες υπηρεσιών νέφους.

Vendor Lock-In

Το vendor lock-in αναφέρεται σε μία κατάσταση κατά την οποία οι πελάτες έχουν επενδύσει μεγάλα χρηματικά ποσά σε έναν συγκεκριμένο πάροχο υπηρεσιών, και έτσι είναι πλέον πολύ υψηλό το κόστος αλλαγής των υπηρεσιών του με κάποιου ανταγωνιστή. Στην έρευνα με τίτλο "Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective" των Justice Opara-Martins, Reza Sahandi και Feng Tian [109], και στο άρθρο "10 Biggest Cloud Computing Challenges in 2021 for IT Service Providers" του Paresh Solanki [110] γίνεται ανάλυση του Vendor lock-in προβλήματος στην υπολογιστική νέφους, αναφέροντας τα βασικά προβλήματα και πιθανές λύσεις. Αυτά, συνοπτικά παρουσιάζονται παρακάτω.

Στον χώρο της υπολογιστικής νέφους, το vendor lock-in συμβαίνει όταν οι πελάτες χρησιμοποιούν αρκετές IaaS, SaaS τεχνολογίες ενός μόνο παρόχου για την ανάπτυξη και υποστήριξη των συστημάτων τους. Ειδικότερα, το πρόβλημα ενισχύεται όταν χρησιμοποιούνται τεχνολογίες που δεν υποστηρίζουν cloud-agnostic τεχνολογίες αλλά ιδιόκτητες τεχνολογίες των παρόχων.

Εάν οι πελάτες βρεθούν αντιμέτωποι με vendor lock-in σε υπηρεσίες υπολογιστικής νέφους, δεν θα είναι έτοιμοι να αντιμετωπίσουν περιπτώσεις όπως:

- Η ποιότητα των υπηρεσιών του παρόχου δεν είναι πλέον ικανοποιητική ή δεν πληρεί τα επιθυμητά ή και συμφωνηθέντα κριτήρια.
- Ο πάροχος μπορεί να αλλάξει τις υπηρεσίες του με τρόπους που δεν ταιριάζουν πλέον στις απαιτήσεις του πελάτη, είτε από οικονομικής πλευράς είτε προδιαγραφών λειτουργίας.
- Ο πάροχος θα μπορούσε να εκμεταλλευτεί το γεγονός ότι ο πελάτης του βρίσκεται σε lock-in κατάσταση.
- Ο πάροχος θα μπορούσε να σταματήσει να παρέχει τις υπηρεσίες του ή και να κλείσει η επιχείριση του συνολικά.

Σε αυτές τις περιπτώσεις, ο πελάτης δεν μπορεί να διαχειριστεί αποδοτικά αυτές τις καταστάσεις καθώς είναι δεσμευμένος από το μεγάλο κόστος που θα χρειαστεί η διαδικασία αλλαγής παρόχου.

Για την προφύλαξη των πελατών από τις επιπτώσεις του vendor lock-in, θα ήταν καλό να τηρούνται τα εξής:

- Προτίμηση υπηρεσιών που υποστηρίζουν open source, standard και cloud-agnostic τεχνολογίες.

- Τήρηση τακτικών αντιγράφων ασφαλείας εύκολα μεταφέρσιμων και εκτός τεχνολογιών του παρόχου.
- Προτίμηση multi-cloud ή hybrid cloud λύσεων.

Περιορισμένος Έλεγχος

Οι υποδομές και οι υπηρεσίες των πάροχων υπολογιστικής νέφους είναι διαχειρίσιμες μόνο από τους ίδιους τους παρόχους. Το γεγονός αυτό προκαλεί ανησυχία στους πελάτες καθώς δεν έχουν αρκετό έλεγχο πάνω στις υπηρεσίες στις οποίες στηρίζουν τα συστήματα τους. Για τον σκοπό αυτό μπορεί να χρησιμοποιηθεί η άδεια χρήσης τελικού χρήστη (end-user license agreement - EULA) και η συμφωνία σε επίπεδο υπηρεσιών (service-level agreement – SLA) στις οποίες αναφέρονται τα όρια τα οποία μπορεί να θέση ο πάροχος στην χρήση υπηρεσιών από τον πελάτη [112].

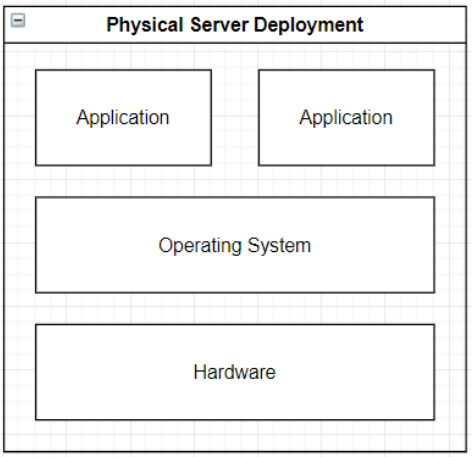
Διαχείριση Κόστους Υπηρεσιών Νέφους

Οι τεχνολογίες νέφους τιμολογούνται από τους παρόχους με διαφορετικές μεθόδους, όχι μόνο μεταξύ παρόχων, αλλά και μεταξύ των μοντέλων υπηρεσιών τους ή και την γεωγραφική τοποθεσία αυτών. Αυτές οι διαφορές προκαλούν σύγχυση στους πελάτες καθώς είναι δύσκολο να εντοπίσουν τις οικονομικότερες λύσεις και αυτές που ταιριάζουν περισσότερο στις ανάγκες τους. Επίσης, αυξάνεται η πιθανότητα λάθους επιλογών οδηγώντας σε υψηλότερα έξοδα σε υπηρεσίες που δεν χρειάζονται ή δεν είχαν αντιληφθεί ότι λειτουργούν [111].

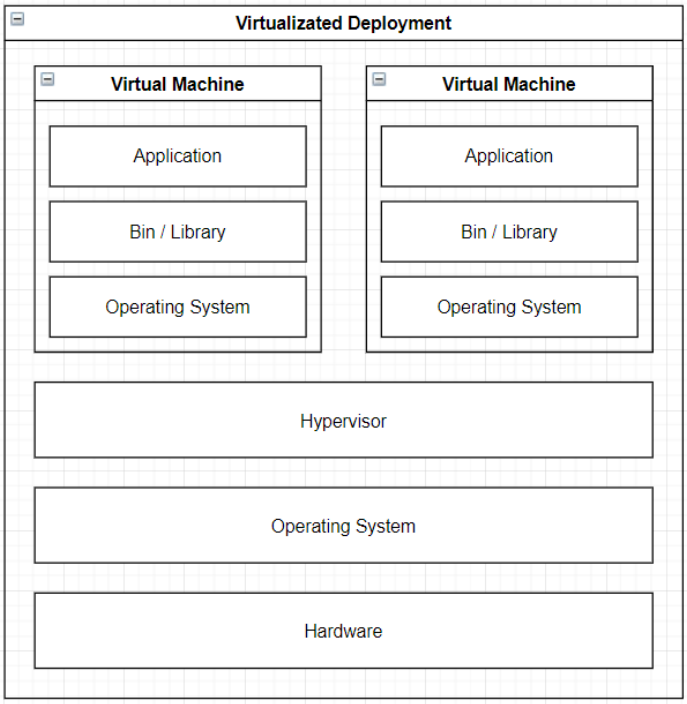
Το πρόβλημα της τιμολόγησης δημιούργησε την ανάγκη για εργαλεία αυτοματισμού πληρωμών υπηρεσιών νέφους. Τα εργαλεία αυτά παρέχουν έναν κεντρικό τρόπο καταγραφής και διαχείρισης πληρωμών υπηρεσιών.

Εικονοποίηση και Εξυπηρετητές (Vitrualization and Servers)

Οι πρώτοι εξυπηρετητές του διαδικτύου ήταν μεγάλες φυσικές μηχανές που απαιτούσαν αρκετούς πόρους για να τους διαχειριστούν και να συντηρηθούν [7]. Οι φυσικοί εξυπηρετητές (Σχήμα 2) μπορούν να παραμετροποιηθούν στις ανάγκες του καταναλωτή και να αποδώσουν ικανοποιητικά. Το πρόβλημα τους σε σύγκριση με την υποδομή νέφους είναι ότι οι φυσικοί εξυπηρετητές οδηγούν σε σπατάλες. Οι σπατάλες προκύπτουν διότι οι δυνατότητες του εξυπηρετητή χρησιμοποιούνται στο έπακρο σπάνια καθώς στα συστήματα οι ανάγκες πόρων αυξομειώνονται συνεχώς. Επιπροσθέτως, σημαντικό μειονέκτημα των φυσικών εξυπηρετητών είναι η απουσία ελέγχου της κατανομής υπολογιστικών πόρων σε επιθυμητές εφαρμογές. Ως αποτέλεσμα, εφαρμογές οι οποίες χρειάζονται περισσότερους πόρους παρουσιάζουν χαμηλότερη επίδοση. Η λύση σε αυτό το πρόβλημα θα ήταν η χρησιμοποίηση διαφορετικών φυσικών εξυπηρετητών για κάθε εφαρμογή, διαδικασία αρκετά υψηλού κόστους για τις εταιρείες και με αδυναμία στην κλιμάκωση των εφαρμογών [8]. Την λύση στα παραπάνω προβλήματα φέρνει η εικονοποίηση των φυσικών μηχανών.



Σχέδιο 2: Αρχιτεκτονική υποδομής ανάπτυξης εφαρμογών σε φυσικό εξυπηρετητή



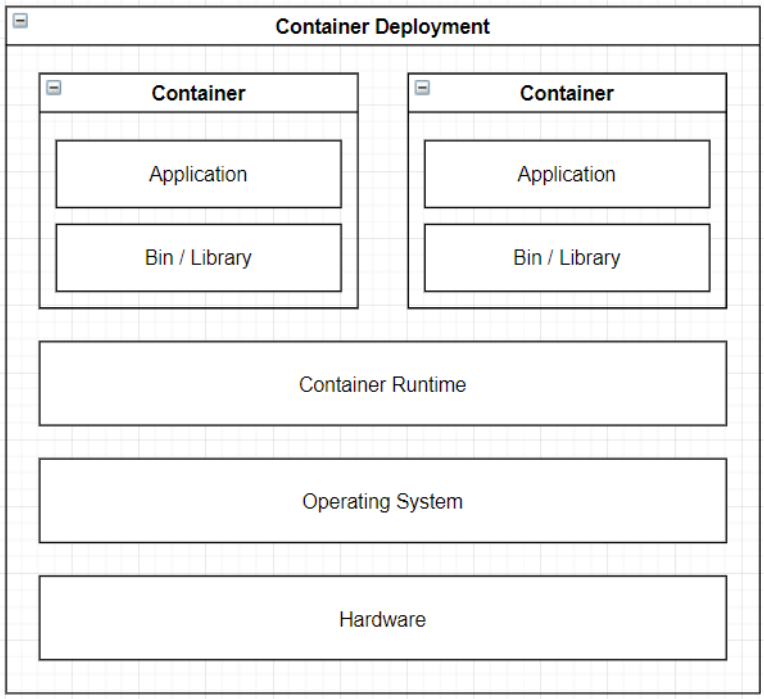
Σχέδιο 3: Αρχιτεκτονική υποδομής ανάπτυξης εφαρμογών σε εικονικό περιβάλλον

Η εικονοποίηση αποτελεί λογισμικό που μιμείται μία φυσική μηχανή [7], ενώ οι οντότητες που δημιουργούνται από την μίμηση καλούνται εικονικές μηχανές VMs (Σχήμα 3). Με αυτόν τον τρόπο μπορούμε να έχουμε πολλούς εικονικούς εξυπηρετητές σε μία φυσική μηχανή κατ' απαίτηση των αναγκών μας. Οι εφαρμογές αν και χρησιμοποιούν την ίδια φυσική μηχανή είναι φορτωμένες σε διαφορετικό VM επομένως, είναι απομονωμένες από τις υπόλοιπες. Η εικονοποίηση επιτρέπει την αποδοτικότερη χρησιμοποίηση των φυσικών πόρων και την ευκολότερη κλιμάκωση καθώς νέα Vms μπορούν να δημιουργηθούν κατ ' απαίτηση [8]. Το πρόβλημα με την χρησιμοποίηση των Vms, είναι ότι και πάλι πρέπει να υπάρχει μία φυσική υποδομή εξυπηρετητή για να αναλάβει την φόρτωση των εικονικών μηχανών.

Κιβώτια (Containers)

Τα Κιβώτια (Containers) αποτελούν καθιερωμένα τμήματα λογισμικού τα οποία οργανώνουν και συσκευάζουν τον κώδικα και όλες τις βιβλιοθήκες και εξαρτήσεις που χρειάζεται ώστε η εφαρμογή να μπορεί να εκτελεστεί σωστά και αξιόπιστα σε διαφορετικά υπολογιστικά περιβάλλοντα [9]. Η διαχείριση των εφαρμογών που είναι φορτωμένες σε containers γίνεται μέσα από τις μηχανές κιβωτίων (container engines). Οι μηχανές αυτές μαζί με τα containers βρίσκονται σε έναν πυρήνα

(kernel) λειτουργικού συστήματος και γιαυτό η έναρξη ενός container είναι γρηγορότερη από την έναρξη μίας εικονικής μηχανής.



Σχέδιο 4: Αρχιτεκτονική υποδομής ανάπτυξης εφαρμογών σε containers

Σε σύγκριση με τις εικονικές μηχανές (VM), τα containers είναι μία χαμηλού φόρτου τεχνολογία εικονοποίησης η οποία δημιουργεί εικονικά συστήματα με απομόνωση στο επίπεδο του λειτουργικού συστήματος, φορτώνει την εφαρμογή και όλες τις εξαρτήσεις της σε ένα έτοιμο προς ανάπτυξη συστατικό, ενώ παράλληλα χρησιμοποιεί λιγότερη μνήμη σε σύγκριση με μία εικονική μηχανή [9] [10].

Βασικά χαρακτηριστικά πλεονεκτήματα που παρέχουν τα containers είναι τα εξής:

- **Διαχωρισμός ευθυνών**. Η χρήση containers παρέχει σαφή διαχωρισμό των στρωμάτων διεργασιών μίας εφαρμογής. Οι προγραμματιστές συγκεντρώνονται στην διαδικασία ανάπτυξης του λογισμικού και τις εξαρτήσεις του ενώ οι μηχανικοί διαχείρισης συστημάτων στην διαδικασία χτισίματος και ανάπτυξης της εφαρμογής σε περιβάλλον παραγωγής.
- **Μεταφερσιμότητα**. Τα Containers μπορούν να εκτελεστούν εικονικά σε μεγάλο εύρος υπολογιστικών και λειτουργικών συστημάτων μέσα από εικονικές ή φυσικές μηχανές, είτε με τεχνολογίες υπολογιστικής νέφους, είτε τοπικά στον υπολογιστή κάθε μηχανικού.

- **Απομόνωση εφαρμογών**. Στα Containers υπάρχει εικονικός χώρος για την μνήμη, υπολογιστική ισχύ, δίκτυο και αποθηκευτικό χώρο που επιτρέπεται να χρησιμοποιήσει η εφαρμογή σε επίπεδο λειτουργικού συστήματος. Επομένως παρέχεται απομόνωση λειτουργικού συστήματος για κάθε εφαρμογή σε κάθε container.

Τα χαρακτηριστικά των containers ως προς την μεταφερσιμότητα, απομόνωση και ανεξαρτησία των εφαρμογών τα καθιστούν ιδανικούς υποψήφιους για χρησιμοποίηση σε τεχνολογίες υποστήριξης εφαρμογών υπολογιστικής νέφους. Οι πάροχοι υπηρεσιών νέφους μπορούν μέσω αυτών να παρέχουν CaaS υπηρεσίες για την υποδομή και υποστήριξη διαφορετικών ειδών εφαρμογών μέσα από containers.

Συστήματα Ενορχήστρωσης Κιβωτίων (Container Orchestration Systems)

Με την τεχνολογία των container να γίνεται δημοφιλέστερη και να αντικαθιστά τα παραδοσιακά VMs, οι οργανισμοί βρέθηκαν αντιμέτωποι με την ανάγκη εργαλείων διαχείρισης τους. Η ανάγκη αυτή προκάλεσε την ανάγκη για σχεδιασμό εργαλείων που διαχειρίζονται την ανάπτυξη των containers σε ομαδοποιημένες συστάδες [11]. Τα εργαλεία αυτά καλούνται Συστήματα Ενορχήστρωσης Κιβωτίων (Container Orchestration Systems) και είναι υπεύθυνα για την ανάπτυξη, διαχείριση, έλεγχο, κλιμάκωση και επικοινωνία των Container [8]. Στηρίζονται στην Slave-Master αρχιτεκτονική και διαχειρίζονται τα containers αφού τα φορτώσουν για εκτέλεση σε απομονωμένα περιβάλλοντα. Από τα απομονωμένα αυτά σημεία τα παρακολουθούν και αποφασίζουν για την κίνηση των αιτημάτων από και προς αυτά, αν χρειάζονται κάποια φροντίδα, διαχείριση ή τροποποίηση στους πόρους που διαθέτονται για αυτά αλλά και την κλιμάκωση τους αν παρατηρείται αυξημένη κίνηση προς αυτά.

Ανάλογα τις τεχνολογίες που διαθέτει το Σύστημα Ενορχήστρωσης Κιβωτίων, κάποια από τα βασικά χαρακτηριστικά που προσφέρονται είναι τα εξής:

- **Εντοπισμός υπηρεσιών και ισορρόπηση φόρτου**. Τα συστήματα αυτά εκθέτουν στο εξωτερικό τους δίκτυο τα containers μέσα από DNS ονόματα ή δικές τους IP διευθύνσεις. Επίσης, αν η κίνηση στο δίκτυο είναι υψηλή προσφέρουν ισορρόπηση της κίνησης κατανέμοντας την κυκλοφορία με τρόπο που προσφέρει περισσότερη σταθερότητα.
- **Κλιμάκωση στιγμιοτύπων υπηρεσιών**. Σε περίπτωση που το σύστημα παρατηρήσει αυξημένη κίνηση προς μία υπηρεσία σε container, υπάρχει η δυνατότητα δημιουργίας αντιγράφων του ώστε να διασπαστεί η κίνηση και να γίνει ταχύτερα η διεκπεραίωση των αιτημάτων.
- **Ενορχήστρωση αποθηκευτικού χώρου**. Είναι δυνατή η αυτόματη δημιουργία συστημάτων αποθηκευτικού χώρου διαφόρων κατηγοριών, όπως τοπικών και δημόσιων χώρων υπηρεσιών νέφους.

- **Έλεγχος και Αυτοδιόρθωση υπηρεσιών**. Στην περίπτωση που κάποια υπηρεσία καταρρεύσει, τα συστήματα αυτά προσφέρουν την δυνατότητα επανεκκίνησης της ανάπτυξης του container, αντικατάστασης του ή διαγραφής του σε περίπτωση που αυτό δεν ανταποκρίνεται. Επίσης, σταματά η έκθεση του στον εξωτερικό κόσμο. Με αυτόν τον τρόπο δίνεται έμφαση στην υψηλή διαθεσιμότητα των υπηρεσιών χωρίς downtime.
- **Διαχείριση κρυφών πληροφοριών και διαμόρφωσης υπηρεσιών**. Οι κρυφές πληροφορίες αφορούν μυστικά κλειδιά, ρυθμίσεις, λογαριασμούς και μεταβλητές περιβάλλοντος. Αυτές οι πληροφορίες είναι δυνατόν να αποθηκεύονται σε έναν ασφαλή χώρο από τον οποίο το Σύστημα Ενορχήστρωσης και να παρέχονται σε containers που τα χρειάζονται χωρίς να χρειάζεται να χτιστούν από την αρχή και χωρίς να φαίνονται εκτεθειμένες στον πηγαίο κώδικα κάθε container.

Με τα συστήματα ενορχήστρωσης είναι δυνατή η αποδοτική διαχείριση της κλιμάκωσης, ανοχής σφαλμάτων, διαθεσιμότητας και κατανομής πόρων εφαρμογών οι οποίες μπορούν να αποτελούνται από εκατοντάδες ξεχωριστές διεργασίες που τρέχουν σε διαφορετικές μηχανές. Οι πάροχοι υπηρεσιών μέσα από CaaS υπηρεσίες υποστηρίζουν τέτοιου είδους υποδομές.

Μη-Σχεσιακές Βάσεις Δεδομένων (Non-Relational Databases – NoSQL)

Τα μοντέλα βάσεων δεδομένων (databases) αποτελούν τρόπο οργάνωσης, λογικού σχεδιασμού, αποθήκευσης και ανάκλησης δεδομένων. Τα δύο βασικά μοντέλα βάσεων δεδομένων είναι τα σχεσιακά (relational databases - SQL) και τα μη-σχεσιακά (non-relationsal databases - NoSQL).

Η έννοια του σχεσιακού μοντέλου (relational model) αναφέρεται ως ένα σετ από οντότητες με την μορφή πίνακα, με κοινές ιδιότητες από τις οποίες συνήθως μία αποτελεί το βασικό αντικείμενο ενώ οι άλλες τις υπόλοιπες ιδιότητες του. Για την σύνδεση μεταξύ των οντοτήτων χρησιμοποιούνται μία ή κάποιες από τις ιδιότητές τους ως κλειδιά [12]. Η παρούσα ανάλυση ασχολείται με τις μη-σχεσιακές βάσεις δεδομένων (non-relational models) καθώς προσφέρουν χαρακτηριστικά κατάλληλα για ανάπτυξη τους σε περιβάλλοντα υπολογιστικής νέφους.

Ο όρος No-SQL αναφέρεται σε μοντέλα βάσεων δεδομένων τα οποία δεν ακολουθούν το σχεσιακό μοντέλο αποθήκευσης δεδομένων, και χρησιμοποιείται ως ομπρέλα για να καλύψει όλων των ειδών τα μη-σχεσιακά μοντέλα, ακόμη και αυτά που είναι σχεδιασμένα για ειδικού τύπου εφαρμογές [12]. Οι No-SQL βάσεις δεδομένων δημιουργήθηκαν από την δυσκολία που παρουσίαζαν οι SQL βάσεις στην διαχείριση αρκετά μεγάλου όγκου δεδομένων, και είναι ικανές να παρέχουν μεγαλύτερη επίδοση σε κόστος αποθήκευσης και ανάκληση δεδομένων.

Βασικά χαρακτηριστικά των No-SQL είναι τα ακόλουθα:

- **Δεν υποστηρίζουν στατικά σχήματα οντοτήτων** τα οποία δεν μπορούν να μεταβληθούν δυναμικά. Τα NoSQL μοντέλα προσφέρουν ευέλικτα ή και εντελώς ελεύθερα σχήματα τα

οποία είναι σχεδιασμένα να υποστηρίζουν ευρεία ποικιλία από δομές δεδομένων. Τα βασικότερα μοντέλα που υποστηρίζονται είναι τα key-value stores, document stores, column-family stores και graph stores.

- **Δεν χρησιμοποιούν συνδέσμους (joins)** για την συσχέτιση οντοτήτων καθώς οι διαφορετικές δομές δεδομένων που υποστηρίζουν τους επιτρέπουν διαφορετικούς τρόπος συσχέτισης.
- **Υποστηρίζουν οριζόντια κλιμάκωση**. Η κλιμάκωση μπορεί να γίνει σε πολλαπλούς εξυπηρετητές.
- **Παρέχουν υψηλή διαθεσιμότητα**. Οι NoSQL εξυπηρετητές είναι σχεδιασμένοι για κατανεμημένη λειτουργία έχοντας σαν προτεραιότητα την διαθεσιμότητα και όχι την συνέπεια των κατανεμημένων δεδομένων.

Τύποι No-SQL βάσεων δεδομένων [13]:

Document-oriented. Οι document-oriented βάσεις αποθηκεύουν τις πληροφορίες σε μορφή αρχείου. Στα αρχεία αυτά οι πληροφορίες είναι αποθηκευμένες σε μη-κανονική μορφή, ημιδομημένα και σε ιεραρχική μορφή. Η μορφοποίηση του αρχείου αποθήκευσης των δεδομένων είναι κάποιας στάνταρ κωδικοποίησης όπως XML ή JSON.

Key-value pairs. Στις key-value pair βάσεις δεδομένων οι πληροφορίες αποθηκεύονται με την μορφή κλειδιού – τιμής. Οι τιμές μπορούν να είναι απλό κείμενο μέχρι περίπλοκες δομές δεδομένων.

Wide column. Οι wide column βάσεις δεδομένων παρουσιάζουν ομοιότητες σε σχέση με της σχεσιακές βάσεις, αλλά ως κεντρική διαφορά έχουν την ευελιξία στις στήλες. Οι στήλες τις κάθε γραμμής που αντιπροσωπεύει τα δεδομένα μίας οντότητας δεν χρειάζεται να είναι ίδιου αριθμού. Αυτό σημαίνει ότι σε μία γραμμή μπορούν να υπάρχουν δύο στήλες αλλά σε άλλες μία.

Graph. Οι graph βάσεις δεδομένων υποστηρίζουν δεδομένα τα οποία έχουν απροσδιόριστο αριθμό από δικτυακές συνδέσεις. Αυτός ο τύπος δεδομένων χρησιμοποιείται για την υποστήριξη δεδομένων που παρουσιάζονται σε γράφους.

Η ικανότητα λειτουργίας των NoSQL βάσεων σε κατανεμημένα συστήματα τις καθιστά ιδανικές για συνθήκες υπολογιστικής νέφους [14]. Πολλοί πάροχοι αξιοποιούν τα συγκεκριμένα μοντέλα στο νέφος και τις προσφέρουν ως λύσεις για την κάλυψη αναγκών αποθήκευσης δεδομένων.

Μεσίτες Μηνυμάτων (Message Brokers)

Σύμφωνα με την IBM Cloud Education στο άρθρο με τίτλο “What are Message Brokers?” [15], οι μεσίτες μηνυμάτων (message brokers) αποτελούν λογισμικό που επιτρέπει σε συστήματα να επικοινωνούν μεταξύ τους. Η επικοινωνία πραγματοποιείται με την μετάφραση των μηνυμάτων από τον message broker των δύο πλευρών στο πρωτόκολλο επικοινωνίας μηνυμάτων της επιλογής τους, παρέχοντας έτσι την δυνατότητα επικοινωνίας μεταξύ ανεξάρτητων υπηρεσιών που μπορούν να

χρησιμοποιούν διαφορετικές τεχνολογίες υλοποίησης. Οι message brokers αποτελούν τμήματα **message-oriented middleware (MOM)** λογισμικών. Τα MOM αποτελούν λογισμικό ή υποδομή υλικού που υποστηρίζει την αποστολή και παραλαβή μηνυμάτων μεταξύ κατανεμημένων συστημάτων.

Οι message brokers επικυρώνουν, αποθηκεύουν, δρομολογούν και παραδίδουν μηνύματα στους κατάλληλους προορισμούς και επιτρέπουν σε χρήστες να στείλουν μηνύματα χωρίς να απαιτείται να γνωρίζουν την κατάσταση, αριθμό ή τοποθεσία των παραληπτών [15]. Για την εξασφάλιση της αποθήκευσης και αποστολής μηνυμάτων, οι message brokers χρησιμοποιούν μία εσωτερική τους δομή που ονομάζεται **ουρά μηνυμάτων (message queue)**. Οι ουρές αυτές χρησιμοποιούνται για την αποθήκευση και δρομολόγηση μηνυμάτων. Δύο χρήστες μπορούν να ανήκουν σε μία ουρά και να στέλνουν μηνύματα σε αυτή. Ο message broker θα πρέπει να στείλει στους χρήστες τα περιεχόμενα της ουράς στην οποία είναι εγγεγραμμένοι με την σωστή σειρά και να δηλώσει ως παρεληφθέντα τα μηνύματα.

Υπάρχουν δύο βασικά μοντέλα επικοινωνίας που παρέχουν οι message brokers:

- **Μηνύματα από σημείο σε σημείο (point-to-point messaging)**. Σε αυτό το μοντέλο οι ουρές μηνυμάτων χρησιμοποιούνται σε ένα προς ένα σχέση μεταξύ δύο χρηστών αποστολέα και παραλήπτη. Κάθε μήνυμα αποστέλνεται από τον αποστολέα και το παραλαμβάνει ο παραλήπτης μόνο μία φορά.
- **Μηνύματα με συνδρομή και δημοσίευση (publish/subscribe messaging)**. Σε αυτό το μοντέλο υπάρχουν θέματα που καλούνται topics και πολλαπλοί χρήστες μπορούν να λαμβάνουν μηνύματα σε αυτές κάνοντας συνδρομή. Τα μηνύματα μπορούν να δημοσιεύονται σε αυτά τα topics από έναν χρήστη και θα λαμβάνονται από όλους όσους είναι συνδρομητές με ένα προς πολλά σχέση.

Όπως θα παρουσιαστεί και σε επόμενα κεφάλαια, οι εφαρμογές που υποστηρίζονται από τεχνολογίες υπολογιστικής νέφους αποτελούνται από μικρά ανεξάρτητα τμήματα υπηρεσιών που επικοινωνούν μεταξύ τους, τις μικρουπηρεσίες. Ένας από τους τρόπους επικοινωνίας μεταξύ των μικρουπηρεσιών είναι η χρήση των message brokers. Οι message brokers επιτρέπουν την ασφαλή και αξιόπιστη μεταφορά μηνυμάτων μεταξύ υπηρεσιών του υπολογιστικού νέφους.

Αρχιτεκτονική Συστημάτων Υπολογιστικής Νέφους (Cloud Native Architecture)

Ο όρος Cloud Native αναφέρεται σε μία αρχιτεκτονική η οποία βασίζεται σε συνδυασμό τεχνολογιών και συστατικών υπολογιστικής νέφους με τρόπο που αξιοποιούνται βέλτιστα οι δυνατότητες της υπολογιστικής νέφους [16]. Ενώ οι παραδοσιακές μονολιθικές αρχιτεκτονικές στόχευαν στην δημιουργία ενός στατικού, μεγάλο σε κόστος συστήματος που απαιτεί εξίσου υψηλό κόστος για να

αλλάξει, η cloud native αρχιτεκτονική στοχεύει στην εκμετάλλευση των παροχών που προσφέρει η υπολογιστική νέφους, για κατασκευή ευέλικτων, επεκτάσιμων και εύκολα διαχειρίσιμων συστημάτων.

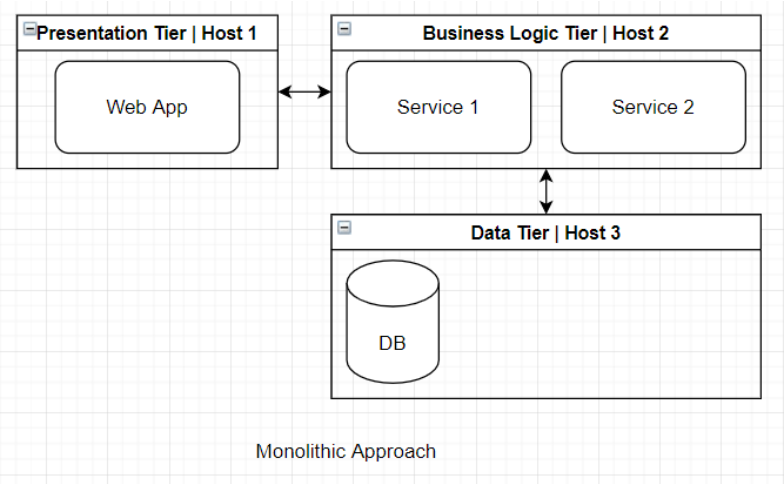
Service-Oriented Αρχιτεκτονική (SOA)

Σύμφωνα με την IBM Cloud Education στο άρθρο με τίτλο "SOA (Service-Oriented Architecture)" [17], η αρχιτεκτονική SOA αποτελεί μία ευρέος γνωστή προσέγγιση ανάπτυξης λογισμικού που στοχεύει στην εκμετάλλευση των επαναχρησιμοποιήσιμων τμημάτων και υπηρεσιών του συστήματος μέσα από διεπαφές υπηρεσιών. Κάθε υπηρεσία αποτελείται από τον πηγαίο κώδικα και τα δεδομένα που χρειάζονται για να εκτελέσει μία συγκεκριμένη επιχειρησιακή λειτουργία. Στην συνέχεια αυτές οι υπηρεσίες παρέχονται σε όλες τις εφαρμογές της επιχείρισης για χρήση. Η διεπαφή των υπηρεσιών προσφέρει loose-coupling, που σημαίνει ότι δεν χρειάζεται να υπάρχει γνώση της υποδομής της υπηρεσίας ή των εξαρτήσεων της για να χρησιμοποιηθεί. Οι υπηρεσίες παρέχονται και επικοινωνούν μέσα από στάνταρ πρωτοκολλά δικτύου όπως το SOAP, Restful και STOMP.

Ο τρόπος με τον οποίο η SOA αρχιτεκτονική διαχειρίζεται τις εφαρμογές και υπηρεσίες της είναι το Enterprise Service Bus (ESB) [17]. Το ESB αποτελεί ένα συγκεντρωτικό λογισμικό που αναλαμβάνει την διαχείριση των ενοποιήσεων μεταξύ των εφαρμογών της επιχείρισης και της προβολής των υπηρεσιών για χρήση. Επίσης, είναι υπεύθυνο για την σύνδεση, επικοινωνία, μοντελοποίηση των αιτημάτων που μεταφέρονται ανάλογα την διεπαφή κάθε υπηρεσίας και εφαρμογής καθώς και την διαχείριση της κίνησης και της δρομολόγησης. Χωρίς το ESB οι εφαρμογές και οι υπηρεσίες της επιχείρισης θα έπρεπε να καλούν κατευθείαν η μία την άλλη, άρα θα έπρεπε να αναλαμβάνουν την διαχείριση της μετάφρασης των αιτημάτων και των απαντήσεων στα ανάλογα μοντέλα επιχειρησιακής λογικής.

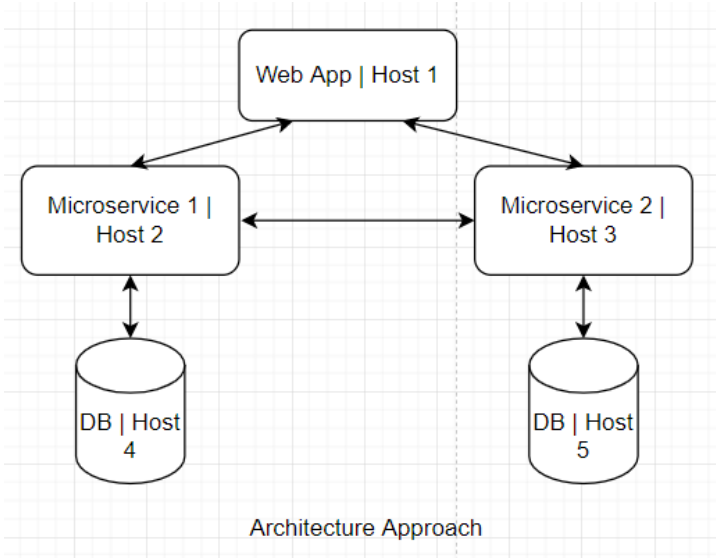
Μικρουπηρεσίες (Microservices)

Η βάση της Cloud Native αρχιτεκτονικής είναι οι μικρουπηρεσίες, οι οποίες όπως και στην αρχιτεκτονική SOA, αποτελούν μικρά, ανεξάρτητα τμήματα ενός συστήματος τα οποία έχουν απομονωμένα μία συγκεκριμένη επιχειρησιακή λειτουργία στο σύνολο του [18] [19]. Οι μικρουπηρεσίες πρέπει να είναι ανεξάρτητες οντότητες, απομονωμένες μεταξύ τους ή σε δικά τους υπολογιστικά συστήματα. Η επικοινωνία μεταξύ τους γίνεται μέσω του διαδικτύου μέσα από διεπαφές API, χρησιμοποιώντας στάνταρ πρωτοκολλά δικτύου όπως το SOAP, Restful και STOMP καθώς δεν συνυπάρχουν στο ίδιο περιβάλλον [20]. Εξαιτίας της ανεξαρτησίας τους, χτίζονται και αναπτύσσονται σε εξυπηρετητές αυτόνομα. Αξίζει να σημειωθεί ότι η βασική διαφορά μεταξύ της SOA αρχιτεκτονικής και των μικρουπηρεσιών είναι η έκταση [18]. Το SOA αναφέρεται σε υπηρεσίες που προσφέρονται σε όλο το εύρος της επιχείρισης και όλες οι εφαρμογές της μπορούν να τις αξιοποιήσουν. Οι μικρουπηρεσίες αποτελούν ανεξάρτητα τμήματα υπηρεσιών στην έκταση μίας εφαρμογής.



Σχέδιο 5: Μονολιθική Προσέγγιση Αρχιτεκτονικής

Βασικό πρόβλημα της μονολιθικής αρχιτεκτονικής (Σχήμα 5) είναι ότι σε μεγάλα συστήματα πρέπει να υπάρχει μεγάλο επίπεδο συνοχής και δομής, φροντίζοντας ο πηγαίος κώδικας να είναι αρκετά οργανωμένος και ομαδοποιημένος. Με αυτόν τον τρόπο αποφεύγεται η χαοτική λειτουργία του συστήματος που αν χρειαζόταν αλλαγή ή αποσφαλμάτωση θα αποτελούσε μία αρκετά δύσκολη διαδικασία. Στις μικρουπηρεσίες (Σχήμα 6), ο τρόπος με τον οποίο εξασφαλίζουμε την σταθερότητα της κλίμακας του συστήματος είναι κάθε υπηρεσία να είναι ανεξάρτητη και να να χωρίζεται σε επίπεδο λειτουργικότητας και επιχειρησιακής λογικής από τις υπόλοιπες.



Σχέδιο 6: Αρχιτεκτονική Προσέγγιση Μικρουπηρεσιών

Πλεονεκτήματα Μικρουπηρεσιών

Τα πλεονεκτήματα που προσφέρουν οι μικρουπηρεσίες προκύπτουν από την δυνατότητα τους να αξιοποιούν χαρακτηριστικά κατανεμημένων συστημάτων. Ο Sam Newman στο βιβλίο "Building Microservices" [19] παρουσιάζει τα χαρακτηριστικά των μικρουπηρεσιών και κάποια από αυτά αναφέρονται παρακάτω.

Ετερογένεια τεχνολογιών (Technology Heterogeneity)

Σε αρχιτεκτονικές μικρουπηρεσιών, εξαιτίας της ανεξαρτησίας και απομόνωσης της κάθε υπηρεσίας μπορούμε να επιλέξουμε τις τεχνολογίες ανάπτυξης που θα χρησιμοποιηθούν για αυτή χωρίς να σκεφτόμαστε πως είναι δομημένες οι άλλες υπηρεσίες. Υπάρχουν διαδικασίες συστημάτων οι οποίες λειτουργούν αποδοτικότερα με συγκεκριμένες τεχνολογίες, επομένως μπορούμε να επιλέξουμε τα κατάλληλα εργαλεία για την κάθε εργασία. Φυσικά πρέπει να διατηρηθεί συγκράτηση ως προς τον αριθμό των τεχνολογιών που θα επιλεγούν καθώς αν κάθε υπηρεσία χρησιμοποιούσε διαφορετικά εργαλεία θα προκαλούταν μεγάλη σύγχυση και το κόστος θα αυξανόταν εξαιτίας των αναγκών για περισσότερο προσωπικό με ειδικότητα σε κάθε τεχνολογία.

Ευελιξία και Αντοχή (Resilience)

Σε συστήματα μικρουπηρεσιών όταν μία υπηρεσία βγει εκτός λειτουργίας τότε δεν καταρρέει ολόκληρο το σύστημα. Οι υπόλοιπες υπηρεσίες που χρειάζονται την συγκεκριμένη, μπορούν να διαχειριστούν την αλλαγή και είτε να στέλνουν τα αιτήματα τους σε εναλλακτική διαδρομή, είτε αναφέρουν ότι δεν μπορούν να εκτελέσουν συγκεκριμένες λειτουργίες που την αφορούν. Μπορεί να γίνει απομόνωση του προβλήματος και να βρεθεί ευκολότερα η λύση του. Στα μονολιθικά συστήματα μπορούμε να έχουμε πολλαπλά αντίγραφα του συστήματα σε διαφορετικές εικονικές μηχανές αλλά, αν είναι ανεπτυγμένο σε μία τότε με την κατάρρευση ενός τμήματος της λειτουργικότητας τότε ολόκληρο το σύστημα μπορεί να τεθεί εκτός λειτουργίας.

Κλιμάκωση (Scaling)

Με την κλιμάκωση εννοούμε την δημιουργία αντιγράφων του συστήματος που τρέχουν σε διαφορετικές διεργασίες με στόχο την αύξηση της απόδοσης. Στην μονολιθική αρχιτεκτονική η κλιμάκωση επιτυγχάνεται όταν δημιουργούμε αντίγραφο από ολόκληρο το σύστημα, διαδικασία αρκετά υψηλού κόστους καθώς στις περισσότερες περιπτώσεις μόνο συγκεκριμένα τμήματα του συστήματος παρουσιάζουν την μεγαλύτερη ανάγκη πόρων. Στις μικρουπηρεσίες μπορούμε να κλιμακώσουμε μόνο εκείνες τις υπηρεσίες που παρουσιάζουν τον μεγαλύτερο υπολογιστικό φόρτο, βελτιστοποιώντας τους διαθέσιμους πόρους της υποδομής μας.

Ευκολία στην Ανάπτυξη (Ease of Deployment)

Σε μεγάλα μονολιθικά συστήματα όταν υπάρχει αλλαγή στον κώδικα θα πρέπει να γίνει πλήρης χτίσιμο της εφαρμογής και ανάπτυξη του σε περιβάλλον λειτουργίας. Αυτή η διαδικασία αποφεύγεται καθώς είναι χρονοβόρα και έτσι περιμένουμε για αρκετές αλλαγές μέχρι την επόμενη έκδοση του συστήματος στην οποία θα έχουμε και αυξημένο ρίσκο σφαλμάτων στο σύστημα. Στις μικρουπηρεσίες, κάθε υπηρεσία έχει τον δικό της χώρο που υπάρχει ο πηγαίος κώδικας και η ανεπτυγμένη εφαρμογή και έτσι, η κάθε αλλαγή μπορεί να γίνει γρήγορα και το χτίσιμο και η ανάπτυξη της υπηρεσίας γίνεται αρκετά εύκολη διαδικασία. Σε περίπτωση σφάλματος η επιστροφή στην προηγούμενη έκδοση της υπηρεσίας μπορεί να γίνει επίσης γρήγορα και έτσι βελτιστοποιείται η διαδικασία ανάπτυξης, ενώ οι υπόλοιπες υπηρεσίες δεν θα επηρεαστούν από την ανάπτυξη της υπηρεσίας.

Βέλτιστοποίηση Αντικατάστασης (Optimizing for Replaceability)

Σε μεγάλα μονολιθικά συστήματα υπάρχουν τμήματα κώδικα στα οποία έχουν χρησιμοποιηθεί ξεπερασμένες τεχνολογίες που θα έπρεπε να αντικατασταθούν αλλά, εξαιτίας της πολυπλοκότητας τους και του υψηλού ρίσκου της αλλαγής οι εταιρείες προτιμούν να μην τα αντικαταστήσουν. Στις μικρουπηρεσίες η διαδικασία ανάπτυξης και αντικατάστασης του κώδικα με διαφορετική τεχνολογία είναι μία σαφώς ευκολότερη διαδικασία καθώς δεν υπάρχουν εξαρτήσεις από το υπόλοιπο σύστημα και όντας μικρουπηρεσία είμαστε σίγουροι πως έχει το ελάχιστο μέγεθος κώδικα για την εκτέλεση της επιχειρησιακής λογικής οπότε, οι αλλαγές θα είναι βέλτιστα ελάχιστες.

Serverless

Στην εργασία με τίτλο “Towards Distributed Containerized Serverless Architecture in Multi Cloud Environment” των Boubaker Soltani, Afifa Ghenai, Nadia Zeghib [21] παρουσιάζονται οι serverless υπηρεσίες και τα χαρακτηριστικά τους. Η Serverless αποτελεί μία αρχιτεκτονική σχεδιασμού υπηρεσιών που μοιάζει με τη SOA και τις μικρουπηρεσίες, με την διαφορά ότι δεν χρειάζεται υποδομή εξυπηρετητή για την διαχείριση της υπηρεσίας. Οι υπηρεσίες αποτελούν ελάχιστα τμήματα επιχειρησιακής λογικής και καλούνται συναρτήσεις (Functions). Βασικά χαρακτηριστικά των **serverless** υπηρεσιών είναι:

- **Δεν απαιτούνται εξυπηρετητές**. Οι συναρτήσεις είναι αποθηκευμένες σε αποθηκευτικούς χώρους μέχρι να κληθούν σε λειτουργία. Όταν συμβεί ένα γεγονός και καλεστούν τότε φορτώνονται στην μνήμη RAM και εκτελούνται. Αρχικά, υπάρχει μόνο μία υπηρεσία ενεργή που λειτουργεί ως διεπαφή και διαχειριστής κλήσεων των υπόλοιπων υπηρεσιών.
- **Stateless συναρτήσεις ενός σκοπού**. Οι συναρτήσεις αποτελούν λειτουργίες ενός σκοπού και δεν διατηρούν την κατάσταση τους στην μνήμη.

- **Πυροδότηση συναρτήσεων μέσα από γεγονότα**. Οι συναρτήσεις λειτουργούν μόνο όταν τις πυροδοτήσει κάποιο γεγονός.

Βασικά πλεονεκτήματα της αρχιτεκτονικής **serverless** προκύπτουν εξαιτίας της **stateless** λειτουργικότητας τους και αφορούν το κόστος, την ευκολία στην κλιμάκωση και την ευκολία στον έλεγχο, χτίσιμο και ανάπτυξη. Ενώ τα μειονεκτήματα σχετίζονται με τον τρόπο κλήσης των συναρτήσεων μέσα από μία υπηρεσία διαχειριστή και αφορούν την απόδοση, ασφάλεια και ιδιωτικότητα, τα όρια στην χρήση υπολογιστικών πόρων και τον έλεγχο τους.

Αξίζει να σημειωθεί ότι υπάρχουν συζητήσεις σχετικά με την επιλογή αρχιτεκτονικής υπηρεσιών με **μικρουπηρεσίες** ή **serverless** , στην εργασία με τίτλο “Microservices vs Serverless: A Performance Comparison on a Cloud-native Web Application” των Chen-Fu Fan, Anshul Jindal, Michael Gerndt [22] γίνεται σχετική μελέτη και σύγκριση αυτών. Αμφότερες μπορούν να αξιοποιηθούν σε cloud native συστήματα αλλά υπάρχουν σημεία στα οποία διαφέρουν. Από πλευράς ανάπτυξης, οι **serverless** υπηρεσίες έχουν περιορισμούς στην υποδομή τους καθώς χρειάζονται υποστήριξη από τεχνολογίες νέφους από πάροχους υπηρεσιών νέφους. Οι **μικρουπηρεσίες** από την άλλη, μπορούν να αναπτυχθούν σε ιδιωτικά και δημόσια δίκτυα νέφους. Μία άλλη διαφορά είναι ότι οι **serverless** υπηρεσίες μπορούν να κλιμακώνονται δυναμικά χωρίς να χρειάζεται κάποια διαμόρφωση στις ρυθμίσεις του εξυπηρετητή.

Μεθοδολογία Ανάπτυξης Cloud Native Εφαρμογών (12-Factor Methodology)

Ο σύγχρονος τρόπος ανάπτυξης εφαρμογών χρησιμοποιώντας τεχνολογίες υπολογιστικού νέφους είναι οι υπηρεσίες. Μία ευρέος γνωστή και αποδεκτή μεθοδολογία για την κατασκευή εφαρμογών στο νέφος (SaaS) είναι η μεθοδολογία 12 παραγόντων (12-Factor methodology) που παρουσιάστηκε από το Heroku [23] [20]. Η 12-Factor μεθοδολογία αναφέρεται σε αρχές και πρακτικές που πρέπει να τηρηθούν από τους μηχανικούς λογισμικού για να κατασκευάσουν εφαρμογές σαν υπηρεσία (SaaS) αξιοποιώντας βέλτιστα τις τεχνολογίες υπολογιστικής νέφους. Ιδιαίτερη προσοχή δίνεται στην μεταφερσιμότητα του συστήματος και τον δηλωτικό αυτοματισμό. Παρακάτω θα ακολουθήσουν οι βασικές αρχές που πρέπει να τηρηθούν.

1. Βάση Κώδικα (Codebase)

Η βάση του πηγαίου κώδικα της εφαρμογής πρέπει να είναι διαχειριζόμενη μέσα από συστήματα ελέγχου εκδόσεων (Version Control Systems). Κάθε διασκευή (revision) του κώδικα και των εκδόσεων του ονομάζεται αποθετήριο κώδικα (code repository ή repo).

Μία βάση κώδικα (Codebase) αποτελεί ένα μοναδικό repo σε συστήματα ελέγχου συγκεντρωτικών διασκευών (SVN) ή ένα σύνολο από repo που μοιράζονται την ίδια αρχική έκδοση του κώδικα σε κατανεμημένα συστήματα ελέγχου διασκευών (Git).

Κάθε codebase πρέπει να ανήκει μόνο σε μία εφαρμογή. Αν υπάρχουν πολλαπλά codebases τότε δεν έχουμε εφαρμογή αλλά κατανεμημένο σύστημα και πρέπει κάθε ξεχωριστό συστατικό του να αποτελεί μία εφαρμογή και να τηρεί αυτόνομα τους παράγοντες. Σε περίπτωση που πολλαπλές εφαρμογές μοιράζονται το ίδιο codebase θα πρέπει να γίνει διαμόρφωση του διαμοιραζόμενου κώδικα σε βιβλιοθήκες και να γίνει η εισαγωγή του από το σύστημα διαχείρισης εξαρτήσεων (dependency manager).

Κάθε codebase αποτελεί μία εφαρμογή, αλλά η εφαρμογή μπορεί να έχει πολλές αναπτύξεις (deploys). Ένα deploy αποτελεί ένα εκτελέσιμο στιγμιότυπο της εφαρμογής.

2. Εξαρτήσεις (Dependencies)

Μία εφαρμογή 12 παραγόντων πάντα δηλώνει όλες τις εξαρτήσεις της μέσω ενός αρχείου δήλωσης εξαρτήσεων (dependency declaration manifest) το οποίο εκτελείται από εργαλεία διαχείρισης εξαρτήσεων (dependency manager tools). Με αυτόν τον τρόπο η εφαρμογή αποδεσμεύεται από εξωτερικούς παράγοντες που πρέπει να προσδιοριστούν για να εκτελεστεί από διαφορετικά συστήματα. Οι μόνες εξαρτήσεις που χρειάζεται ένας προγραμματιστής για να τρέξει την εφαρμογή θα πρέπει να είναι το λογισμικό εκτέλεσης της γλώσσας προγραμματισμού (language runtime) και ο dependency manager. Ο dependency manager θα πρέπει μέσα από μία δηλωτική εντολή εκτέλεσης της εφαρμογής (πχ mvn install) να ετοιμάζει το εκτελέσιμο της εφαρμογής με κάθε της εξάρτηση και να την εκτελεί.

3. Διαμόρφωση (Config)

Οι παράμετροι μιας εφαρμογής είναι δεδομένα τα οποία καθορίζουν τα διαφορετικά deploys και αποτελούνται από ρυθμίσεις υποστηρικτικών υπηρεσιών, διαπιστευτήρια και κρυφά δεδομένα εξωτερικών υπηρεσιών και λοιπά δεδομένα που χρησιμοποιούνται για την ανάπτυξη της εφαρμογής όπως ονόματα εξυπηρετητών και πόρτες σύνδεσης.

Σε μία εφαρμογή 12 παραγόντων, οι παράμετροι της είναι αποθηκευμένοι ως μεταβλητές περιβάλλοντος (environmental variables). Οι Μεταβλητές περιβάλλοντος είναι δεδομένα που που είναι αποθηκευμένα στο λειτουργικό σύστημα που έχει αναπτυχθεί η εφαρμογή. Με αυτόν τον τρόπο είναι εύκολο να αλλάξουμε στιγμιότυπο ανάπτυξης της εφαρμογής χωρίς να υπάρξει αλλαγή στο codebase. Επίσης, είναι σημαντικό το codebase να έχει την δυνατότητα να είναι δημόσιο (open source) χωρίς να εκθέτει μυστικά κλειδιά του. Σε περίπτωση που κάποια παράμετρος είναι αποθηκευμένη ως σταθερά στον κώδικα, τότε παραβιάζεται ο παράγοντας.

4. Υπηρεσίες Υποστήριξης (Backing Services)

Οι υπηρεσίες υποστήριξης αποτελούνται από υπηρεσίες που χρησιμοποιούνται από την εφαρμογή για διεκπεραίωση αιτημάτων της ενώ δεν είναι υλοποιημένες μέσα στο codebase της. Τέτοιες υπηρεσίες

μπορούν να είναι βάσεις δεδομένων (datastores), συστήματα μηνυμάτων (message /queues systems – AMPQ services), συστήματα διαχείρισης προσωρινής μνήμης (caching systems) κ.α.

Σε εφαρμογές 12 παραγόντων, δεν πρέπει να υπάρχει διαχωρισμός στον τρόπο κλήσης και λειτουργίας των υπηρεσιών υποστήριξης από την εφαρμογή. Είτε οι υπηρεσίες είναι ανεπτυγμένες σε τοπικούς εξυπηρετητές είτε καλούνται από κάποιον πάροχο υπηρεσιών, ο τρόπος λειτουργίας τους είναι ίδιος και δεν επηρεάζει την ομαλή λειτουργία της εφαρμογής. Η σύνδεση σε αυτές τις υπηρεσίες πρέπει να γίνεται μέσα από το αρχείο παραμέτρων (config), οπότε αλλάζοντας αυτό μπορεί να αλλάξει και ο εξυπηρετητής της υπηρεσίας όμως, η εύρυθμη λειτουργία της εφαρμογής παραμένει σταθερή.

Με αυτό τον τρόπο η εφαρμογή έχει ακόμα λιγότερη εξάρτηση από υπηρεσίες που δεν μπορεί να ελέγξει και δεν ανήκουν στο codebase της. Αν συμβεί κάποιο σφάλμα δεν χρειάζεται κάποια αλλαγή στον κώδικα, απλά θα πρέπει να αλλάξουν οι παράμετροι σύνδεσης στην υπηρεσία από το config αρχείο με κάποια εναλλακτική.

5. Κατασκευή, Έκδοση, Εκτέλεση (Build, Release, Run)

Κατά την φάση ανάπτυξης της εφαρμογής, η βάση κώδικα (codebase) της περνάει κάποια στάδια που την οδηγούν στο τελικό εκτελέσιμο προϊόν. Αρχικά, στο πρώτο στάδιο χτισίματος (build) της εφαρμογής, παράγεται η εκτελέσιμη έκδοση της που περιέχει όλες τις εξαρτήσεις και τους πόρους που χρειάζεται. Στην συνέχεια στο στάδιο της έκδοσης (release) το built αρχείο της εφαρμογής συνδυάζεται με το config αρχείο και φορτώνει όλες τις παραμέτρους που θα χρησιμοποιήσει. Τέλος, στο στάδιο εκτέλεσης (run) η εφαρμογή εκτελείται από το περιβάλλον ανάπτυξης.

Στις εφαρμογές 12 παραγόντων, υπάρχει αυστηρός διαχωρισμός μεταξύ των τριών σταδίων ανάπτυξης.

6. Διεργασίες (Processes)

Μία εφαρμογή 12 παραγόντων εκτελείται ως ένα σύνολο διεργασιών. Οι διεργασίες αυτές πρέπει να μην μοιράζονται δεδομένα μεταξύ τους, και να μην κρατούν την κατάσταση τους (stateless). Μία stateless διεργασία δεν αποθηκεύει στην τοπική μνήμη της δεδομένα των αιτημάτων που δέχεται. Σε περίπτωση που θέλουμε να διατηρήσουμε δεδομένα στην υπηρεσία πρέπει να χρησιμοποιήσουμε υποστηρικτικές υπηρεσίες (backing services). Στις εφαρμογές 12 παραγόντων υποθέτουμε ότι οι διεργασίες δεν διατηρούν τίποτα στην μνήμη τους που μπορεί να χρησιμοποιηθεί για την διεκπεραίωση αιτημάτων.

7. Πρόσδεση Πόρων (Port Binding)

Οι εφαρμογές 12 παραγόντων είναι αυτοτελής και δεν βασίζονται σε εξωτερικούς εξυπηρετητές ιστού (webservers) για την λειτουργία τους και την εξαγωγή των υπηρεσιών τους στο δίκτυο. Πρέπει να εξάγουν αυτόνομα ως υπηρεσίες τις λειτουργίες τους μέσα από πρωτόκολλα διαδικτύου (πχ HTTP) και

τα συνδυάζουν με θύρες (ports). Με αυτό τον τρόπο κάθε αυτόνομη διεργασία της εφαρμογής μπορεί να λειτουργήσει ως υποστηρικτική υπηρεσία άλλων υπηρεσιών με την κατάλληλη καταγραφή των παραμέτρων της στο config αρχείο της εφαρμογής.

8. Παραλληλία (Concurrency)

Κάθε υπηρεσία μίας εφαρμογής 12 παραγόντων εξαιτίας της stateless λειτουργικότητας των υπηρεσιών της μπορεί εύκολα να επεκταθεί σε μεγάλο αριθμό μικρών πανομοιότυπων διαδικασιών (copies) σε αντίθεση με την επέκταση μίας μεγάλης εφαρμογής σε μία ισχυρή διαθέσιμη μηχανή.

9. Απορριψιμότητα (Disposability)

Οι υπηρεσίες των εφαρμογών 12 παραγόντων πρέπει να είναι αναλώσιμες (disposable). Αναλώσιμες υπηρεσίες είναι αυτές που μπορούν να ξεκινήσουν ξεκινήσουν και να τερματιστούν αξιόπιστα και σε μικρό χρόνο, αφήνοντας την εφαρμογή στην σωστή κατάσταση. Αυτό το χαρακτηριστικό επιτρέπει στις εφαρμογές να επεκτείνονται, επανακτίζονται και να αναπτύσσονται ευκολότερα.

10. Ισοτιμία Dev/Prod (Dev/prod Parity)

Οι εφαρμογές 12 παραγόντων είναι σχεδιασμένες να υποστηρίζουν συνεχή ανάπτυξη (continuous deployment) κρατώντας μικρές διαφορές μεταξύ του περιβάλλοντος τοπικής ανάπτυξης (local deploy) με εκείνου της παραγωγής (production deploy). Οι διαφορές σχετίζονται με την τήρηση μικρής χρονικής διάρκειας ανάπτυξης καινούριου κώδικα από προγραμματιστές προς εξαγωγή στην παραγωγή, τις διαφορές σε τεχνολογίες που χρησιμοποιούνται τοπικά με εκείνες στην παραγωγή, και τέλος με το προσωπικό που εργάζεται στο τοπικό και παραγωγικό περιβάλλον. Επίσης, πρέπει να αποφεύγεται η χρήση διαφορετικών υποστηρικτικών υπηρεσιών ανάμεσα σε local και production περιβάλλοντα.

11. Αρχεία Συμβάντων (Logs)

Τα αρχεία συμβάντων (logs) αποτελούνται από χρονικά διατεταγμένα συμβάντα των διεργασιών και υπηρεσιών υποστήριξης, προσφέροντας μία εικόνα της συμπεριφοράς των υπηρεσιών της εφαρμογής, και έτσι παρουσιάζουν την κατάσταση της. Σε εφαρμογές 12 παραγόντων η εφαρμογή δεν πρέπει να ασχολείται με την διαχείριση των logs. Κάθε υπηρεσία της εφαρμογής εξάγει τα logs της στο κανάλι εξόδου (stdout) και το περιβάλλον ανάπτυξης της εφαρμογής είναι υπεύθυνο να συλλέξει την ροή κάθε υπηρεσία και να την δρομολογήσει στον χρήστη.

12. Διεργασίες Διαχείρισης (Admin Processes)

Οι διεργασίες διαχειριστή περιλαμβάνουν εκκαθάριση, μεταφορά και διορθώσεις στην βάση δεδομένων, εξαγωγή αναφορών κ.α. Οι εφαρμογές 12 παραγόντων πρέπει να εκτελούν τις διεργασίες αυτές μόνο μία φορά, στο περιβάλλον της εφαρμογής αλλά ξεχωριστά από αυτή.

Αυτοματισμός Ανάπτυξης

Με την παραπάνω ανάλυση αποτυπώθηκε η σημαντικότητα της αρχιτεκτονικής συστημάτων υπολογιστικής νέφους και φαίνεται η σημαντική βελτίωση στην ανάπτυξη των εφαρμογών. Όμως, δεν αναφέρθηκε ότι οι διαδικασίες χτισίματος και ανάπτυξης του λογισμικού για κάθε μία από τις υπηρεσίες του συστήματος γίνονται χειροποίητα. Σε μεγάλα συστήματα θα ήταν αρκετά χρονοβόρα διαδικασία η ανάπτυξη και διαμόρφωση όλων των υπηρεσιών. Σύμφωνα με τους Tom Laszewski, Kamal Arora, Erik Farr, Piyum Zonooz στο βιβλίο με τίτλο “Cloud Native Architectures” [24], αναφέρεται πως χρησιμοποιώντας τα πλεονεκτήματα των τεχνολογιών υπολογιστικής νέφους μπορούμε να αυτοματοποιήσουμε τις διαδικασίας αυτές μέσα από τα εργαλεία Συνεχούς Ενοποίησης και Συνεχούς Παράδοσης (Continuous Integration, Continuous Delivery CI/CD). Αυτά τα εργαλεία με βάση το άρθρο “What is CI/CD - all you need to know” του Maciej Manturewicz [25] παρουσιάζονται παρακάτω.

Ο όρος **Συνεχής Ενοποίηση (Continuous Integration)** αναφέρετε στο αυτόματο χτίσιμο, έλεγχο και δοκιμασία του πηγαίου κώδικα μετά από κάθε συγχώνευση (merge) του κώδικα των προγραμματιστών με τον βασικό κώδικα της πηγής. Οι προγραμματιστές πρέπει να συγχωνεύουν τον κώδικα τους με εκείνον της πηγής όσο το δυνατόν συχνότερα, κρατώντας έτσι πάντα ενήμερη την κεντρική πηγή. Σε κάθε καινούρια ενημέρωση του κώδικα της πηγής, υπάρχουν οι δοκιμές ενοποίησης (integration tests) οι οποίες τρέχουν αυτόματα. Εάν ο νέος κώδικας περάσει τις δοκιμές των integration test τότε γίνεται merge αλλιώς οι αλλαγές απορρίπτονται και ζητούνται αλλαγές. Με αυτό τον τρόπο ελαχιστοποιούμε τις πιθανότητες μελλοντικών συγκρούσεων στον κώδικα και πολλές ομάδες μπορούν να εργάζονται ταυτόχρονα κρατώντας πάντα ενήμερη την κεντρική πηγή.

Ο όρος **Συνεχής Παράδοση (Continuous Delivery)** αναφέρετε στην ετοιμότητα του κώδικα προς ανάπτυξη ανά πάσα στιγμή, κρατώντας πάντα την τελευταία και πιο πρόσφατη έκδοση του συστήματος έτοιμη προς παράδοση στην παραγωγή. Αυτός ο συνεχόμενος τρόπος παράδοσης νέων εκδόσεων του συστήματος το κάνει πιο ανεκτικό σε βλάβες και σφάλματα. Επίσης, η διαδικασία αλλαγής της έκδοσης με μία παλαιότερη σε περίπτωση μεγάλης βλάβης γίνεται ταχύτερα, και χωρίς να αφαιρεί μεγάλη νέα λειτουργικότητα του συστήματος. Εκτός από την Συνεχή Παράδοση (Continuous Delivery), υπάρχει και ο ορισμός Συνεχής Ανάπτυξη (Continuous Deployment), αυτός αναφέρετε σε μία ανεπτυγμένη μορφή συνεχούς παράδοσης κατά την οποία με την οποιαδήποτε νέα προσθήκη στον κεντρικό κώδικα, γίνεται αυτόματα η διαδικασία ανάπτυξης τους στην παραγωγή χωρίς ανθρώπινη παρέμβαση.

Οι cloud native αρχιτεκτονικές ενώ προσφέρουν νέες δυνατότητες στους μηχανικούς, αυξάνουν επίσης την πολυπλοκότητα διαχείρισης της ανάπτυξης των εφαρμογών, σύμφωνα με το άρθρο με τίτλο “What is cloud native continuous integration” του GitLab [26]. Για αυτό τον σκοπό τα συστήματα ενορχήστρωσης containers και τα εργαλεία CI/DE παρέχουν ενσωματωμένα εργαλεία για την

επικοινωνία μεταξύ τους, προκειμένου να υπάρχει συνεργασία στην αποδοτική αυτόματη ανάπτυξη των containers.

Κεφάλαιο 3: Μελέτη Δωρεάν Υπηρεσιών Υπολογιστικής Νέφους

Σε αυτό το κεφάλαιο θα πραγματοποιηθεί μελέτη σύγκρισης και αξιολόγησης μεταξύ των τεχνολογιών που προσφέρουν οι δημοφιλέστεροι πάροχοι υπηρεσιών υπολογιστικής νέφους, θα αποτυπωθούν τα κριτήρια επιλογής και αξιολόγησης και θα παρουσιαστούν τα τελικά συμπεράσματα που προκύπτουν με την διεκπεραίωση της μελέτης.

Εισαγωγή

Καθώς οι απαιτήσεις για υπηρεσίες υπολογιστικής νέφους από οργανισμούς συνεχίζουν να παραμένουν σε υψηλά επίπεδα, οι τεχνολογικοί κολοσσοί αυξάνουν τις επενδύσεις τους στην προσπάθεια τους να καλύψουν τις νέες ανάγκες. Σύμφωνα με την ανάλυση του Canalys για το 2020 [50] και το άρθρο με τίτλο “Top Cloud Service Providers & Companies of 2021 ” του james Malgure [51] για το 2021, οι μεγαλύτεροι πάροχοι υπηρεσιών νέφους αυτή τη στιγμή είναι οι Microsoft Azure, Amazon Web Services και Google Cloud. Η Amazon Web Services διατηρεί το μεγαλύτερο μερίδιο κάλυψης, η Microsoft Azure έρχεται δεύτερη ενώ η Google Cloud βρίσκεται στην τρίτη θέση. Στην συνέχεια ακολουθούν με την σειρά οι IBM Cloud, Oracle και Vmware.

Οι τεχνολογικοί κολοσσοί στην προσπάθεια τους να κάνουν περισσότερο προσιτές τις τεχνολογίες που προσφέρουν στους καταναλωτές, προσφέρουν κάποιες από τις βασικές υπηρεσίες τους με δωρεάν προγράμματα χρήσης. Αναλόγως την υπηρεσία και την εταιρεία, τα δωρεάν αυτά προγράμματα διαφέρουν ως προς ένα σύνολο παραγόντων όπως τις ώρες, χρησιμοποίηση, ημερομηνία λήξης κ.α. Τα δωρεάν προγράμματα των υπηρεσιών επιτρέπουν στους καταναλωτές να εγκαταστήσουν και να χρησιμοποιούν υπηρεσίες των παρόχων με μηδενικό κόστος. Βασικό πρόβλημα στις δωρεάν υπηρεσίες είναι ότι δεν είναι αρκετές σε υπολογιστικούς πόρους για να καλύψουν τις ανάγκες ενός συστήματος το οποίο βρίσκεται σε εταιρικό περιβάλλον παραγωγής.

Αξίζει να αναφερθεί ότι οι υπηρεσίες, προσφορές και δωρεάν προγράμματα που παρέχονται από τους παρόχους υπολογιστικής νέφους διαρκώς μεταβάλλονται και τα δεδομένα που θα παρουσιαστούν πιθανώς μελλοντικά να έχουν αλλάξει. Η αξιολόγηση των τεχνολογιών θα γίνει με βάση τα τρέχων δεδομένα που ισχύουν για το 2021.

Όρια Μελέτης

Εξαιτίας της μεγάλης ποικιλίας υπηρεσιών που παρέχονται από τους πάροχους υπηρεσιών, θα πρέπει να τεθούν κάποια όρια στο εύρος κατηγοριών των τεχνολογιών που θα παρουσιαστούν προς σύγκριση. Οι κατηγορίες υπηρεσίων που θα μελετηθούν αφορούν μόνο τις βασικές υπηρεσίες IaaS, CaaS και PaaS που απαιτούνται για την ανάπτυξη και υποστήριξη ενός συστήματος υπολογιστικής νέφους 12

παραγόντων στις βασικές συνιστώσες του (δεδομένα, επιχειρησιακή λογική και παρουσίαση). Επομένως θα μελετηθούν οι παρακάτω κατηγορίες υπηρεσιών:

- Υπηρεσίες υποστήριξης μικρουπηρεσιών (βάσεων δεδομένων, χώρου αποθήκευσης, μεσίτη μηνυμάτων)
- Υπηρεσίες εκτέλεσης μικρουπηρεσιών και συναρτήσεων
- Υπηρεσίες ενορχήστρωσης κιβωτίων υπηρεσιών
- Υπηρεσίες αυτοματισμού χτισίματος και ανάπτυξης εφαρμογών

Κριτήρια Αξιολόγησης Τεχνολογιών

Για την διαδικασία σύγκρισης μεταξύ των τεχνολογιών πρέπει να οριστούν τα κριτήρια με τα οποία θα αξιολογηθούν. Τα κριτήρια αυτά αποτελούν περιορισμούς και μετρικές με τις οποίες οι πάροχοι τεχνολογιών νέφους κοστολογούν την χρήση υπηρεσιών που προσφέρουν στους πελάτες τους. Τα κριτήρια αυτά είναι:

- Ο χρόνος χρησιμοποίησης μίας υπηρεσίας ή των υπολογιστικών πόρων της συνολικά ή σε ορισμένο χρονικό διάστημα (πχ 28 ώρες επεξεργασίας ανά ημέρα)
- Ο αριθμός των εκτελέσεων μίας υπηρεσίας συνολικά ή σε ορισμένο χρονικό διάστημα (πχ 2 εκατομμύρια εκτελέσεις μίας υπηρεσίας ανά μήνα)
- Αριθμός στιγμιοτύπων υπηρεσιών (πχ 2 εικονικές μηχανές)

Στα δωρεάν πακέτα παροχών δίνεται συγκεκριμένο όριο στις παραπάνω μετρικές ανά υπηρεσία. Θα αξιολογηθούν ως καλύτερες οι οικονομικότερες υπηρεσίες που ανήκουν στα δωρεάν προγράμματα και οι οποίες δίνουν των περισσότερο συνολικό χρόνο λειτουργίας ή τις περισσότερες εκτελέσεις, μεταξύ ενός συγκεκριμένου χρονικού διαστήματος.

Τέλος, σημαντικό κριτήριο για την επιλογή τεχνολογιών είναι υπηρεσίες που υποστηρίζουν cloud-agnostic τεχνολογίες.

Συγκέντρωση Δεδομένων και Συγκρίσεις

Δωρεάν Προγράμματα Παροχών

Κατηγορίες

Τα δωρεάν προγράμματα παροχών (Free Tiers) των AWS, Google Cloud και Microsoft Azure χωρίζονται σε δύο βασικές κατηγορίες, τα “Limited Time” και τα “Always Free”. Τα “Limited Time” είναι προγράμματα που παρέχουν δωρεάν συγκεκριμένες υπηρεσίες για κάποια χρονική περίοδο (πχ

12 μήνες), και υπάρχει όριο στην χρήση των υπηρεσιών ανά κάποιο χρονικό όριο (πχ 120 λεπτά ανά ημέρα). Τα “Always Free” προγράμματα παρέχονται για πάντα δωρεάν αλλά και αυτά μπορούν να χρησιμοποιηθούν ανά κάποιο χρονικό όριο [54].

Εκτός των παραπάνω Free Tier παροχών, η Google Cloud και η Microsoft Azure προσφέρουν και Free Trial πακέτα κατά τα οποία με την κάθε νέα εγγραφή πιστώνουν στον λογαριασμό του χρήση ένα ποσό προς χρήση σε παρεχόμενες υπηρεσίες για κάποιο δοθέν χρονικό διάστημα. Η Google Cloud προσφέρει ποσό ύψους $300 δολαρίων προς χρήση τις πρώτες 90 ημέρες από την δημιουργία του λογαριασμού [52] ενώ, η Microsoft Azure προσφέρει $200 δολάρια προς χρήση για τις πρώτες 30 ημέρες [53].

Πάροχος	**Ποσό Πίστωσης (Δολάρια)**	**Χρόνος διάρκειας (Ημέρες)**
Google Cloud	300 (255 euro)	90
Microsoft Azure	200 (169.50 euro)	30
AWS	-	-

Πίνακας 2: Σύγκριση Ποσών Πίστωσης ανά Πάροχο

Όσον αφορά τα Free Trial πακέτα, φαίνεται πως η Google Cloud παρέχει το μεγαλύτερο ποσό για το περισσότερο χρονικό διάστημα ενώ η AWS δεν διαθέτει καθόλου αυτή την δυνατότητα.

Περιορισμοί

- Οι υπηρεσίες που παρέχονται στα Free Tier προγράμματα προσφέρονται προς χρήση σε προσδιορισμένα όρια χρήσης που μπορούν να επεκταθούν μόνο με πληρωμή.
- Εμπορικό λογισμικό και λειτουργικά συστήματα δεν συμπεριλαμβάνονται στα Free Tier πακέτα [54].
- Οι χρονικοί περιορισμοί χρήσης υπηρεσιών στα Free Tier προγράμματα δεν μπορούν να προστεθούν αν δεν χρησιμοποιηθούν από τον χρήστη. Δηλαδή, αν μία υπηρεσία παρέχεται 2 επεξεργαστικές ώρες τον μήνα και ο χρήστης δεν την χρησιμοποιεί, οι ώρες αυτές δεν θα προστίθεται σαν υπόλοιπο, και κάθε μήνα θα αρχικοποιούνται [54].

Μηχανές Υπολογισμών (Compute Engines)

Οι μηχανές υπολογισμών αποτελούν IaaS στιγμιότυπα εικονικών μηχανών για την παροχή φιλοξενίας και εκτέλεσης εφαρμογών με δυνατότητα κλιμάκωσης τους. Το λειτουργικό σύστημα των μηχανών αυτών καθορίζεται από την διαμόρφωση τους.

Google Cloud's Compute Engine

Η Google Cloud στο Free Tier (Always Free) παρέχει 1 non-preemptible e2-micro VM στιγμιότυπο ανά μήνα προς χρήση το οποίο προσφέρεται από τις ακόλουθες περιοχές:

- Oregon: us-west1
- lowa: us-central1
- South Carolina: us-east1

Επιπλέον το Compute Engine στο Free Tier περιέχει:

- 30GB τον μήνα HDD χώρου αποθήκευσης
- 5GB τον μήνα αποθηκευτικού χώρου στιγμιοτύπου στις περιοχές:
  - Oregon: us-west1
  - lowa: us-central1
  - South Carolina: us-east1
  - Taiwan: asia-east1
  - Belgium: europe-west1
- 1GB εύρος ζώνης ανά μήνα από την Νότιο Αμερική σε όλες τις υπόλοιπες περιοχές εκτός από την Κίνα και Αυστραλία.

Microsoft Azure's Linux & Windows Virtual Machines

Η Microsoft Azure στο Free Tier (Limited Time) παρέχει δύο B1S Standard tier στιγμιότυπα VMs, ένα με λειτουργικό σύστημα Windows και ένα με Linux, προς χρήση για συνολικά 750 ώρες τον μήνα στους πρώτους 12 μήνες από την δημιουργία του Free Tier λογαριασμού. Οι εικονικές μηχανές B1S Standard tier περιλαμβάνουν [65] [73]:

- 1vCPU
- 1GB Ram
- 4GB τον μήνα αποθηκευτικού χώρου στιγμιοτύπου

Amazon Web Services' Amazon EC2

Η AWS στο Free Tier (Limited Time) παρέχει ένα στιγμιότυπο t2.micro ή t3.micro λειτουργικού συστήματος Linux, RHEL, SLES ή Windows προς χρήση για συνολικά 750 ώρες τον μήνα στους πρώτους 12 μήνες από την δημιουργία του Free Tier λογαριασμού [81] [82].

Οι εικονικές μηχανές t2.micro και t3.micro περιλαμβάνουν:

- 1GB RAM
- 1vCPU η t2.micro και 2vCPU η t3.micro

Σύγκριση Τεχνολογιών

Στην συνέχεια πραγματοποιείται σύγκριση μεταξύ των προσφερόμενων τεχνολογιών μηχανών υπολογισμών. Στον παρακάτω πίνακα (πίνακας 3) φαίνεται πως η εικονική μηχανή της Google είναι η μόνη που δίνεται μόνο για Linux και μόνο με ένα στιγμιότυπο όμως, προσφέρεται δωρεάν για πάντα σε σχέση με τις μηχανές της Amazon και Microsoft οι οποίες δίνονται για 12 μήνες, αλλά και για περιορισμένο αριθμό ωρών τον μήνα.

Πάροχος	**Υπηρεσία**	**Τύπος Free Tier**	**Χρόνος Λειτουργίας**	**Αριθμός Στιγμιοτύπων**	**Ονομασία VM**
Google	Compute Engine	Always Free	Επ' αόριστον	1	non-preemptible e2-micro
Microsoft	Linux And Windows VMs	Limited Time (12 μήνες)	750 ώρες τον μήνα	2 (από 1 OS)	BS1 Standard
AWS	Amazon EC2	Limited Time (12 μήνες)	750 ώρες τον μήνα	2 (από ένα OS)	t3.micro, t2.micro

Πίνακας 3: Μηχανές Υπολογισμών - Σύγκριση προτάσεων προσφορών

Στον πίνακα 4 φαίνονται οι προδιαγραφές των VMs. Η μηχανή της Google και η t3 της Amazon διαθέτουν από δύο πυρήνες vCPU ενώ οι υπόλοιπες από έναν. Όλες οι μηχανές έχουν το ίδιο μέγεθος RAM.

Ονομασία VM	**CPU**	**RAM**	**OS**
Non-preemtible e2-micro	2vCPU	1GB	Linux
BS1 Standard	1vCPU	1GB	Linux ή Windows
t3.micro	2vCPU	1GB	Linux, RHEL, SLES ή Windows
t2.micro	1vCPU	1GB	Linux, RHEL, SLES ή Windows

Πίνακας 4: Μηχανές Υπολογισμών - Σύγκριση μεταξύ προσφερόμενων VMs

Στον πίνακα 5 φαίνονται οι υποστηρικτικές υποδομές χώρου των VMs. Ο μόνιμος χώρος αποθήκευσης του VM στην Google παρέχεται δωρεάν ενώ, στην Microsft και Amazon πρέπει να γίνει χρήση της υπηρεσίας Managed Discs και Amazon EBS αντίστοιχα. Οι υπηρεσίες Managed Discs και Amazon EBS προσφέρουν περισσότερο αποθηκευτικό χώρο και υψηλότερες αποδόσεις σε σύγκριση με τον

HDD της Google αλλά, όπως και τα VMs, έτσι και αυτές οι υπηρεσίες παρέχονται για τους πρώτους 12 μήνες.

Ονομασία VM	**Storage**	**Temp Storage**	**Υποστήριξη μόνο από Συγκεκριμένες Ζώνες περιοχών**
non-preemptible e2-micro (Google Cloud)	30GB HDD τον μήνα	5GB	NAI
BS1 Standard (Microsoft Azure)	Χρήση υπηρεσίας Azure Managed Discs (2x64GB SSD)	4GB	OXI
t3.micro (AWS)	Χρήση υπηρεσίας Amazon EBS (30GB SSD)	-	NAI (Για Windows OS)
t2.micro (AWS)	Χρήση υπηρεσίας Amazon EBS (30GB SSD)	-	NAI (Για Windows OS)

Πίνακας 5: Μηχανές Υπολογισμών - Σύγκριση μεταξύ υποδομής υποστήριξης των προσφερόμενων VM

Στον πίνακα 6 ακολουθεί ο πίνακας τελικών συμπερασμάτων για τα VMs.

Ονομασία VM	**Αδυναμίες**	**Πλεονεκτήματα**
non-preemptible e2-micro (Google)	• Περιορισμένες ζώνες διαθεσιμότητας • Χαμηλή χωρητικότητα • 1 στιγμιότυπο	• Always Free • Χωρίς χρονικούς περιορισμούς
BS1 Standard (Microsoft)	• Limited Free (12 μήνες) 750 ώρες τον μήνα	• 2 στιγμιότυπα • Υψηλή επίδοση χωρητικότητας
t3.micro, t2.micro (Amazon)	• Limited Free (12 μήνες) 750 ώρες τον μήνα • Χαμηλότερη επίδοση χωρητικότητας σε σύγκριση με την Microsoft • Τα Windows παρέχονται σε συγκεκριμένες ζώνες μόνο στο t3.micro	• 2 στιγμιότυπα • Περισσότερα εκδόσεις των Linux

Πίνακας 6: Μηχανές Υπολογισμών - Συγκριτικά συμπεράσματα μεταξύ των προσφερόμενων VM

Από τις παραπάνω πληροφορίες του πίνακα 6, φαίνεται πως η καλύτερη προσφορά είναι εκείνη της Microsoft με την BS1 Standard μηχανή αλλά, για τους πρώτους 12 μήνες του λογαριασμού και με συνολικό χρόνο λειτουργίας 750 ωρών τον μήνα. Αν υπάρχει υψηλότερη ανάγκη για συνεχής διαθεσιμότητα για αόριστο χρονικό περιθώριο, τότε η καλύτερη προσφορά είναι της Google.

Πλατφόρμες Διαχείρισης και Εκτέλεσης Εφαρμογών

Google Cloud

App Engine

Η υπηρεσία App Engine της Google Cloud αποτελεί μία υπολογιστική πλατφόρμα φιλοξενίας serverless εφαρμογών και επιτρέπει την ανάπτυξη και εκτέλεση stateless containers ή και εκτελέσιμων αρχείων. Η πλατφόρμα αυτή αναλαμβάνει όλη τη διαχείριση της υποδομής για την ανάπτυξη, κλιμάκωση και φιλοξενία της εφαρμογής [56].

Στο Free Tier (Always Free) το App Engine παρέχει τα παρακάτω:

- 28 Ώρες την ημέρα από “F” στιγμιότυπα. Τα “F” στιγμιότυπα προσφέρουν αυτόματη κλιμάκωση.
- 9 Ώρες την ημέρα από “B” στιγμιότυπα. Τα “B” στιγμιότυπα προσφέρουν χειροκίνητη κλιμάκωση.
- 1GB εύρος ζώνης.

Cloud Run

Η υπηρεσία Cloud Run της Google Cloud αποτελεί μία υπολογιστική πλατφόρμα φιλοξενίας serverless εφαρμογών και επιτρέπει την ανάπτυξη και εκτέλεση stateless containers. Η πλατφόρμα αυτή αναλαμβάνει όλη τη διαχείριση της υποδομής για την ανάπτυξη, κλιμάκωση και φιλοξενία της εφαρμογής [59].

Στο Free Tier (Always Free) το Cloud Run παρέχει τα παρακάτω:

- 2 εκατομμύρια αιτήματα τον μήνα.
- 360,000GB δευτερόλεπτα μνήμης και 180,000 vCPU δευτερόλεπτα επεξεργαστικής ισχύς.
- 1GB εύρος ζώνης από την Νότιο Αμερική τον μήνα.

Cloud Functions

Η υπηρεσία Cloud Functions της Google Cloud αποτελεί μία υπολογιστική πλατφόρμα φιλοξενίας serverless συναρτήσεων που επιτρέπει την ανάπτυξη και εκτέλεση συναρτήσεων κώδικα. Η

πλατφόρμα αυτή αναλαμβάνει όλη τη διαχείριση της υποδομής για την ανάπτυξη και φιλοξενία της εφαρμογής [58].

Στο Free Tier (Always Free) η υπηρεσία αυτή παρέχει τα παρακάτω:

- 2 εκατομμύρια κλήσεις τον μήνα.
- 400,000GB και 200,000GHz ανά δευτερόλεπτο υπολογιστικής ισχύς.
- 5GB εύρος ζώνης τον μήνα.

Microsoft Azure

Azure App Service

Η υπηρεσία Azure App Service της Microsoft αποτελεί μία υπολογιστική πλατφόρμα φιλοξενίας full-stack, statefull εφαρμογών και επιτρέπει την ανάπτυξη και εκτέλεση μέσω images και εκτελέσιμων αρχείων. Η πλατφόρμα αυτή αναλαμβάνει όλη τη διαχείριση της υποδομής για την ανάπτυξη, κλιμάκωση και φιλοξενία της εφαρμογής [69].

Στο Free Tier (Always Free) η Azure App Service παρέχει τα παρακάτω [73]:

- 10 υπηρεσίες.
- 1GB χώρο αποθήκευσης για κάθε υπηρεσία.
- 1 shared “F1” VM με 60 λεπτά χρόνου την ημέρα.

Azure Functions

Η υπηρεσία Azure Functions της Microsoft αποτελεί μία υπολογιστική πλατφόρμα φιλοξενίας serverless συναρτήσεων που επιτρέπει την ανάπτυξη και εκτέλεση συναρτήσεων κώδικα. Η πλατφόρμα αυτή αναλαμβάνει όλη τη διαχείριση της υποδομής για την ανάπτυξη και φιλοξενία της εφαρμογής [70].

Στο Free Tier (Always Free) η Azure Functions παρέχει 1 εκατομμύριο κλήσεις με 400,000GBs κατανάλωσης πόρων τον μήνα [73].

Amazon Web Services

AWS Lambda

Η υπηρεσία AWS Lambda της AWS αποτελεί μία υπολογιστική πλατφόρμα φιλοξενίας serverless συναρτήσεων που επιτρέπει την ανάπτυξη και εκτέλεση συναρτήσεων κώδικα. Η πλατφόρμα αυτή αναλαμβάνει όλη τη διαχείριση της υποδομής για την ανάπτυξη και φιλοξενία της εφαρμογής [83].

Στο Free Tier (Always Free) η AWS Lamda παρέχει 1 εκατομμύριο κλήσεις με 400,000GBs κατανάλωσης πόρων τον μήνα.

AWS Amplify Console

Η υπηρεσία AWS Amplify Console της AWS αποτελεί πλατφόρμα φιλοξενίας static web apps και διαθέτει ενσωματωμένες CI/CD αυτοματοποιήσεις. Η πλατφόρμα αυτή αναλαμβάνει όλη τη διαχείριση της υποδομής για την ανάπτυξη και φιλοξενία της εφαρμογής [92].

Στο Free Tier (Limited Time) η AWS Amplify Console υπηρεσία προσφέρει για τους πρώτους 12 μήνες από την ενεργοποίηση του λογαριασμού τα παρακάτω:

- 1000 λεπτά build χρόνου τον μήνα
- 15GB χωρητικότητας εξυπηρέτησης τον μήνα
- 5GB χώρου αποθήκευσης τον μήνα

Σύγκριση Τεχνολογιών

Στην συνέχεια πραγματοποιείται σύγκριση μεταξύ των προσφερόμενων τεχνολογιών εκτέλεσης και διαχείρισης εφαρμογών. Στον παρακάτω πίνακα (πίνακας 7) φαίνονται οι προσφορές των υπηρεσιών. Όλες οι υπηρεσίες παρέχονται δωρεάν για πάντα εκτός από την Amplify της Amazon, η οποία διατίθεται για 12 μήνες.

Πάροχος	**Υπηρεσία**	**Τύπος Free Tier**
Google	App Engine	Always Free
Google	Cloud Run	Always Free
Google	Cloud Functions	Always Free
Microsoft	Azure App Service	Always Free
Microsoft	Azure Functions	Always Free
AWS	AWS Lambda	Always Free
AWS	AWS Amplify Console	Limited Time (12 μήνες)

Πίνακας 7: Πλατφόρμες Εκτέλεσης και Διαχείρισης Εφαρμογών - Σύγκριση προτάσεων προσφορών

Στον πίνακα 8 φαίνονται τα χαρακτηριστικά που υποστηρίζουν οι υπηρεσίες. Αρχικά, όλες οι υπηρεσίες διαθέτουν αυτόματη κλιμάκωση. Στην συνέχεια, η μόνη υπηρεσία που δεν απαιτεί serverless εφαρμογές είναι η Azure App Service της Microsoft.

Υπηρεσία	**Serverless**	**Αυτόματη Κλιμάκωση**	**Εκτέλεση μόνο όταν καλείται**

App Engine	NAI	NAI	NAI
Cloud Run	NAI	NAI	NAI
Cloud Functions	NAI	NAI	NAI
Azure App Service	OXI	NAI	OXI
Azure Functions	NAI	NAI	NAI
AWS Lambda	NAI	NAI	NAI
AWS Amplify Console	NAI	NAI	OXI

Πίνακας 8: Πλατφόρμες Εκτέλεσης και Διαχείρισης Εφαρμογών - Σύγκριση μεταξύ προσφερόμενων υπηρεσιών και χαρακτηριστικών που υποστηρίζουν

Στον πίνακα 9 φαίνονται οι προδιαγραφές και οι χρόνοι λειτουργίας των υπηρεσιών. Όλες οι υπηρεσίες εκτός των Azure App Service και AWS Amplify διατίθενται για συγκεκριμένο αριθμό κλήσεων και χρόνου διαθέσιμων πόρων τον μήνα. Η AWS Amplify έχει όμως όριο τα 15GB εύρους ζώνης τον μήνα επομένως, η καλύτερη προσφορά συνολικά φαίνεται να είναι η Azure App Service της Microsoft καθώς, επιτρέπει την υποστήριξη έως 10 υπηρεσιών με 1GB χώρου αποθήκευσης χωρίς κανέναν άλλο περιορισμό.

Υπηρεσία	**Κλήσεις**	**Χρόνος Υπολογισμών**	**Χρόνος Μνήμης**	**Χώρος Αποθήκευσης**	**Αριθμός Υπηρεσιών**
App Engine	28 ώρες “F” (100,800 δευτερόλεπτα) 9 ώρες “B” (32,400 δευτερόλεπτα) τον μήνα			-	-
Cloud Run	2 εκατομμύρια τον μήνα	180,000 δευτερόλεπτα τον μήνα	360,000 GB δευτερόλεπτα τον μήνα	-	-
Cloud Functions	2 εκατομμύρια τον μήνα	200,000 GB δευτερόλεπτα τον μήνα	400,000 GB δευτερόλεπτα τον μήνα	-	-
Azure App Service		60 λεπτά shared “F1” VM την ημέρα (100,800 δευτερόλπτα τον μήνα)	1GB	1GB	10
Azure Functions	1 εκατομμύρια τον μήνα	400,000 GB συνολικά δευτερόλεπτα τον μήνα		-	-

AWS Lambda	1 εκατομμύρια τον μήνα	400,000 GB συνολικά δευτερόλεπτα τον μήνα	-	-
AWS Amplify Console	15GB served χρόνος τον μήνα		5GB	-

Πίνακας 9: Πλατφόρμες Εκτέλεσης και Διαχείρισης Εφαρμογών - Σύγκριση μεταξύ προσφερόμενων υπηρεσιών και προδιαγραφών τους

Στον πίνακα 10 ακολουθεί ο πίνακας τελικών συμπερασμάτων των υπηρεσιών.

Το μεγαλύτερο μειονέκτημα των υπηρεσιών Cloud Functions, Azure Functions και AWS Lambda είναι ότι από τις προδιαγραφές τους είναι FaaS και οι πάροχοι εφαρμόζουν περισσότερους περιορισμούς στην κατασκευή τους. Οι υπηρεσίες πρέπει να ορίζονται σαν συνάρτηση και με συγκεκριμένες γλώσσες, προκειμένου να γίνει η κατασκευή και ανάπτυξη τους. Επομένως, δημιουργούνται διάφορα θέματα ως προς την διαχείριση της πολυπλοκότητας τους και την εξάρτηση τους από τον πάροχο που έχουν δημιουργηθεί.

Η υπηρεσία AWS Amplify Console μπορεί να χρησιμοποιηθεί μόνο για front-end static web apps επομένως δεν μπορεί να υποστηρίξει μικρουπηρεσίες.

Οι καλύτερες υπηρεσίες φαίνεται να είναι οι Google App Engine, Cloud Run και Azure App Service. Η Google App Engine με την Cloud Run έχουν αρκετές ομοιότητες με την διαφορά ότι η Cloud Run επιτρέπει περισσότερο έλεγχο στην χτίση των containers ενώ, η App Engine στην διαχείριση της κλιμάκωσης. Η Azure App Service υποστηρίζει statefull εφαρμογές αλλά παρέχει λιγότερο συνολικό χρόνο διαθεσιμότητας από τις Cloud Run και Google App Engine.

Υπηρεσία	**Αδυναμίες**	**Πλεονεκτήματα**
App Engine	• Χαμηλός αριθμός συνολικού χρόνου λειτουργίας • Χτίσιμο images μέσα από την Google	• Δυνατότητα διαχείρισης της κλιμάκωσης του • Δυνατότητα εκτέλεσης μόνο όταν καλείται
Cloud Run	• Χαμηλότερος χρόνος λειτουργίας από το Cloud Functions	• Επιτρέπει εκτέλεση Custom build docker images • Εκτελείται μόνο όταν καλείται • Υψηλότερος αριθμός συνολικού χρόνου λειτουργίας σε μη-FaaS
Cloud Functions	• Μειονεκτήματα FaaS	• Υψηλότερος αριθμός εκτελέσεων και

		συνολικού χρόνου πόρων από όλα • Εκτελείται μόνο όταν καλείται
Azure App Service	• Περιορισμός στις υπηρεσίες • Χαμηλός αριθμός συνολικού χρόνου λειτουργίας	• Statefull
Azure Functions	• Μειονεκτήματα FaaS	• Εκτελείται μόνο όταν καλείται
AWS Lambda	• Μειονεκτήματα FaaS	• Εκτελείται μόνο όταν καλείται
AWS Amplify Console	• Hosting μόνο σε static front-end web apps • Limited Free (12 μήνες) 750 ώρες τον μήνα	• Χώρος αποθήκευσης 5GB

Πίνακας 10: Πλατφόρμες Εκτέλεσης και Διαχείρισης Εφαρμογών - Συγκριτικά συμπεράσματα μεταξύ των προσφερόμενων υπηρεσιών

Αποθηκευτικός Χώρος (Storage)

Google Cloud's Cloud Storage

Η υπηρεσία Cloud Storage επιτρέπει την αποθήκευση και ανάκτηση δεδομένων. Αυτή η υπηρεσία μπορεί να χρησιμοποιηθεί για ποικίλους σκοπούς όπως αποθήκευση περιεχομένου ιστοσελίδων, αρχεία αντιγράφων ασφαλείας, αρχεία προς διαμοιρασμό κ.α [61].

Στο Free Tier (Always Free) η Cloud Storage υπηρεσία παρέχει τα παρακάτω:

- 5GB τον μήνα χωρητικότητας από περιοχές στις Ηνωμένες Πολιτείες (US).
- 5,000 λειτουργίες κλάσης "A" τον μήνα. Οι λειτουργίες κλάσης "A" αναφέρονται σε ενεργές λειτουργίες όπως Insert, Update.
- 50,000 λειτουργίες κλάσης "B" τον μήνα. Οι λειτουργίες κλάσης "B" αναφέρονται σε παθητικές λειτουργίες όπως Get, Select.
- 1GB εύρος ζώνης τον μήνα από την Νότιο Αμερική σε όλες τις άλλες περιοχές εκτός από την Κίνα και την Αυστραλία.

Η υπηρεσία στο Free Tier είναι διαθέσιμη μόνο από τις παρακάτω περιοχές:

- us-east1

- us-west1
- us-central1

Microsoft Azure

Managed Disks (Block Storage)

Η υπηρεσία Managed Disks της Microsoft αποτελεί IaaS χώρο αποθήκευσης πλήρως διαχειρίσιμο από υπηρεσίες της Azure και κατάλληλο να χρησιμοποιηθεί από εικονικές μηχανές [71].

Στο Free Tier (Limited Time) η Managed Disks υπηρεσία προσφέρει για τους πρώτους 12 μήνες από την ενεργοποίηση του λογαριασμού τα παρακάτω [73]:

- 2 “P6” SSDs χωρητικότητας 64GB.
- 1GB snapshot χώρο αποθήκευσης.
- 2 Εκατομμύρια I/O λειτουργίες τον μήνα.

Blob Storage

Η υπηρεσία Blob Storage της Microsoft αποτελεί PaaS διαχείρισης αποθηκευτικού χώρου δεδομένων μη προσδιορισμένης δομής [72].

Στο Free Tier (Limited Time) η Blob Storage υπηρεσία προσφέρει για τους πρώτους 12 μήνες από την ενεργοποίηση του λογαριασμού τα παρακάτω [73]:

- 5GB LRS αποθηκευτικού χώρου προς χρήση. Το “LRS” αναφέρεται σε αντίγραφα των δεδομένων που δημιουργούνται στο τοπικό κέντρο δεδομένων που υπάρχουν και τα αρχικά δεδομένα [77].
- 20,000 λειτουργίες ανάγνωσης και 10,000 λειτουργίες εγγραφής τον μήνα.

Amazon Web Services

Amazon S3

Η υπηρεσία Amazon S3 της AWS αποτελεί PaaS διαχείρισης αποθηκευτικού χώρου δεδομένων μη προσδιορισμένης δομής [84].

Στο Free Tier (Limited Time) η Amazon S3 υπηρεσία προσφέρει για τους πρώτους 12 μήνες από την ενεργοποίηση του λογαριασμού τα παρακάτω:

- 5GB Αποθηκευτικού χώρου.
- 20,000 λειτουργίες ανάγνωσης και 2,000 λειτουργίες εγγραφής τον μήνα.

Amazon Elastic Block Store

Η υπηρεσία Amazon Elastic Block Store της AWS αποτελεί IaaS χώρο αποθήκευσης πλήρως διαχειρίσιμο από υπηρεσίες της AWS και κατάλληλο να χρησιμοποιηθεί από εικονικές μηχανές [85].

Στο Free Tier (Limited Time) η Amazon Elastic Block Store υπηρεσία προσφέρει για τους πρώτους 12 μήνες από την ενεργοποίηση του λογαριασμού τα παρακάτω:

- 30GB Αποθηκευτικού χώρου.
- 1GB snapshot χώρο αποθήκευσης.
- 2 Εκατομμύρια I/O λειτουργίες τον μήνα.

Σύγκριση Τεχνολογιών

Στην συνέχεια πραγματοποιείται σύγκριση μεταξύ των προσφερόμενων τεχνολογιών αποθηκευτικών χώρων. Στον παρακάτω πίνακα (πίνακας 11) φαίνονται οι προτάσεις προσφερόμενων υπηρεσιών. Η μοναδική υπηρεσία που διατίθεται μονίμως δωρεάν είναι της Google ενώ, οι υπηρεσίες τις Microsoft είναι εκείνες που προσφέρονται για συγκεκριμένο αριθμό ωρών και μηνών.

Πάροχος	**Υπηρεσία**	**Τύπος Free Tier**	**Χρόνος Λειτουργίας**
Google	Cloud Storage	Always Free	Επ’ αόριστον
Microsoft	Blob Storage	Limited Time (12 μήνες)	750 ώρες τον μήνα
Microsoft	Managed Discs	Limited Time (12 μήνες)	750 ώρες τον μήνα
AWS	Amazon S3	Limited Time (12 μήνες)	Επ’ αόριστον
AWS	Amazon Elastic Block Store	Limited Time (12 μήνες)	Επ’ αόριστον

Πίνακας 11: Αποθηκευτικός χώρος - Σύγκριση προτάσεων προσφορών

Στον πίνακα 12 φαίνονται οι συγκρίσεις μεταξύ των αποθηκευτικών χώρων που προσφέρονται. Η Google φαίνεται πως δεν παρέχει καθόλου IaaS αποθηκευτικό χώρο για τα VM της και καλύπτει μόνο συγκεκριμένες ζώνες περιοχών αλλά, παρέχει τις περισσότερες κλήσεις για εγγραφή/ανάγνωση.

Υπηρεσία	**Τύπος Υπηρεσίας**	**Χωρητικότητα**	**Κλήσεις**	**Υποστήριξη μόνο από Συγκεκριμένες Ζώνες περιοχών**
Cloud Storage	PaaS	5GB	5,000 write 50,000 read τον μήνα	NAI
Blob Storage	PaaS	5GB (LRS)	10,000 write	OXI

			20,000 read τον μήνα	
Managed Discs	IaaS (για VMs)	64GB x 2	2,000,000 ops τον μήνα	OXI
Amazon Elastic Block Store	IaaS (για VMs)	30GB	2,000,000 ops τον μήνα	OXI
Amazon S3	PaaS	5GB	2,000 write 20,000 read τον μήνα	OXI

Πίνακας 12: Αποθηκευτικός χώρος - Σύγκριση μεταξύ προσφερόμενων αποθηκευτικών χώρων

Στον πίνακα 13 ακολουθεί ο πίνακας τελικών συμπερασμάτων των υπηρεσιών. Σύμφωνα με τις τιμές του πίνακα 12 οι καλύτερη IaaS είναι της Microsoft (Managed Discs) ενώ, η καλύτερη PaaS η Cloud Storage της Google. Οι υπηρεσίες EBS και S3 της Amazon βρίσκονται στην μέση των συγκρίσεων και δεν παρουσιάζουν αξιοσημείωτες διαφορές.

Υπηρεσία	**Αδυναμίες**	**Πλεονεκτήματα**
Cloud Storage	• Διαθεσιμότητα μόνο σε συγκεκριμένες ζώνες περιοχών	• Υψηλότερη χρονική διαθεσιμότητα
Blob Storage	• Limited Free (12 μήνες) 750 ώρες τον μήνα	• Δεύτερο σε χρονική διαθεσιμότητα • Δεν υπάρχει περιορισμός σε ζώνες περιοχών
Managed Discs	• Limited Free (12 μήνες) 750 ώρες τον μήνα	• Υψηλότεροι πόροι και απόδοση σε IaaS
Amazon Elastic Block Store	• Λιγότερη χρήση από άλλες • Limited Free (12 μήνες) 750 ώρες τον μήνα	-
Amazon S3	• Limited Free (12 μήνες) 750 ώρες τον μήνα	• Δεν υπάρχει περιορισμός σε ζώνες περιοχών

Πίνακας 13: Αποθηκευτικός χώρος - Συγκριτικά συμπεράσματα μεταξύ των προσφερόμενων υπηρεσιών

Βάσεις Δεδομένων (Databases)

Google Cloud's Firestore

Η υπηρεσία Firestore της Google Cloud αποτελεί μία No-SQL document oriented βάση δεδομένων δημιουργημένη για αυτόματη κλιμάκωση [62]. Η υπηρεσία αυτή παρέχει διεπαφή με την οποία μπορεί να χρησιμοποιηθεί ως υποστηρικτική υπηρεσία μέσα από άλλες υπηρεσίες.

Στο Free Tier (Always Free) η Firestore υπηρεσία παρέχει τα παρακάτω:

- 1GB χωρητικότητας.
- 50,000 αναγνώσεις, 20,000 εγγραφές και 20,000 διαγραφές την ημέρα.

Microsoft Azure

Azure Cosmos DB

Η υπηρεσία Azure Cosmos DB της Microsoft αποτελεί μία υπηρεσία No-SQL βάσεων δεδομένων. Η υπηρεσία αυτή παρέχεται έτοιμα διαμορφωμένη για υποστήριξη No-SQL βάσεων με διεπαφές για MongoDB, Cassandra, Gremlin και Azure table storage [66].

Στο Free Tier (Always Free) η Azure Cosmos DB υπηρεσία παρέχει τα παρακάτω [73]:

- 1000 Request Units ανά δευτερόλεπτο [78]
- 25GB αποθηκευτικού χώρου

Εκδόσεις υποστήριξης μηχανών [97] [98]:

- MongoDB: 4.0, 3.6, 3.2
- Cassandra: Cassadra Query Language (CQL) v3.11 API (backward-compatible version 2.x)

Azure Database for MySQL

Η υπηρεσία Azure Database for MySQL της Microsoft αποτελείται από έναν έτοιμα διαμορφωμένο εξυπηρετητή εικονικής μηχανής B1MS ο οποίος φιλοξενεί MySQL βάση δεδομένων [94].

Στο Free Tier (Limited Time) η Azure Database for MySQL υπηρεσία προσφέρει για τους πρώτους 12 μήνες από την ενεργοποίηση του λογαριασμού τα παρακάτω [73]:

- B1MS εικονική μηχανή
  - 1 vCPU
  - 2GB RAM
  - 4GB προσωρινός χώρος αποθήκευσης

- 32GB αποθηκευτικού χώρου
- 32GB αποθηκευτικού χώρου αντιγράφων ασφαλείας

Εκδόσεις υποστήριξης MySQL [101]: 8.0, 5.7, 5.6 (Deprecation scheduled for 1/2/2022)

Azure Database for PostgreSQL

Η υπηρεσία Azure Database for PostgreSQL της Microsoft αποτελείται από έναν έτοιμα διαμορφωμένο εξυπηρετητή εικονικής μηχανής B1MS ο οποίος φιλοξενεί PostgreSQL βάση δεδομένων [95].

Στο Free Tier (Limited Time) η Azure Database for PostgreSQL υπηρεσία προσφέρει για τους πρώτους 12 μήνες από την ενεργοποίηση του λογαριασμού τα παρακάτω [73]:

- B1MS εικονική μηχανή
  - 1 vCPU
  - 2GB RAM
  - 4GB προσωρινός χώρος αποθήκευσης
- 32GB αποθηκευτικού χώρου
- 32GB αποθηκευτικού χώρου αντιγράφων ασφαλείας

Εκδόσεις υποστήριξης PostgreSQL [103]: 11, 10, 9.6

Azure SQL Database

Η υπηρεσία Azure SQL Database της Microsoft αποτελεί μία SQL βάση δεδομένων. Η υπηρεσία αυτή παρέχει διεπαφή με την οποία μπορεί να χρησιμοποιηθεί ως υποστηρικτική υπηρεσία μέσα από άλλες υπηρεσίες [93].

Στο Free Tier (Limited Time) η Azure SQL Database υπηρεσία προσφέρει για τους πρώτους 12 μήνες από την ενεργοποίηση του λογαριασμού τα παρακάτω [73]:

- 250GB αποθηκευτικού χώρου
- S0 εικονική μηχανή
  - 40 QPUs (Query processing units)
  - 10GB RAM
- 10 transaction units

Amazon Web Services

Amazon DynamoDB

Η υπηρεσία Amazon DynamoDB της AWS αποτελεί μία No-SQL βάση δεδομένων αυτόματης κλιμάκωσης. Η υπηρεσία αυτή παρέχει διεπαφή με την οποία μπορεί να χρησιμοποιηθεί ως υποστηρικτική υπηρεσία μέσα από άλλες υπηρεσίες [86].

Στο Free Tier (Always Free) η Azure Cosmos DB υπηρεσία παρέχει τα παρακάτω [73]:

- Από 25 Read & Write Capacity Units ανά δευτερόλεπτο [87].
- 25GB αποθηκευτικού χώρου

Amazon RDS

Η υπηρεσία Amazon RDS της AWS αποτελεί μία υπηρεσία SQL βάσεων δεδομένων. Η υπηρεσία αυτή παρέχει έτοιμα διαμορφωμένα VM στιγμιότυπα υποστήριξης SQL μηχανών PostgreSQL, MySQL, MariaDB, SQL Server και Oracle BYOL [96].

Στο Free Tier (Limited Time) η Amazon RDS υπηρεσία προσφέρει για τους πρώτους 12 μήνες από την ενεργοποίηση του λογαριασμού τα παρακάτω [73]:

- Υποστήριξη MySQL, PostgreSQL, MariaDB, Oracle BYOL, SQL Server.
- db.t2.micro εικονικές μηχανές για κάθε μηχανή βάσης δεδομένων.
  - 1 vCPU
  - 1GB RAM
- 20GB χώρου αποθήκευσης.
- 20GB προσωρινού χώρου αποθήκευσης για τα VM και χώρου αποθήκευσης αντιγράφων ασφαλείας.

Εκδόσεις υποστήριξης μηχανών [99] [100] [101] [102] [103]:

- MariaDB: 10.5, 10.4, 10.3, 10.2
- MySQL: 8.0, 5.7, 5.6 (Deprecation scheduled for 1/2/2022)
- Oracle: 19c, 18c, 12c Release 2, 12c Release 1
- PostgreSQL: 12, 11, 10, 9.6
- Microsoft SQL Server: 2019 CU8 15.00.4073.23, 2017 CU23 14.00.3381.3, 2016 SP2 CU16 13.00.5882.1, 2014 CU4 12.00.6329.1, 2012 SP4 GDR 11.0.7493.4

Σύγκριση Τεχνολογιών

Στην συνέχεια πραγματοποιείται σύγκριση μεταξύ των προσφερόμενων τεχνολογιών βάσεων δεδομένων. Στον παρακάτω πίνακα (πίνακας 14) φαίνονται οι προτάσεις προσφορών των υπηρεσιών. Όλες οι σχεσιακές βάσεις δεδομένων παρέχονται για τους πρώτους 12 μήνες από την ενεργοποίηση του λογαριασμού για την Microsoft και AWS ενώ, η Google δεν διαθέτει κάποια SQL δωρεάν υπηρεσία.

Πάροχος	**Υπηρεσία**	**Τύπος Free Tier**	**Χρόνος Λειτουργίας**
Google	Firestore	Always Free	Επ' αόριστον
Microsoft	Azure Cosmos DB	Always Free	Επ' αόριστον
Microsoft	Azure SQL Database	Limited Time (12 μήνες)	750 ώρες τον μήνα
Microsoft	Azure Database for MySQL	Limited Time (12 μήνες)	750 ώρες τον μήνα
Microsoft	Azure Database for PostgreSQL	Limited Time (12 μήνες)	750 ώρες τον μήνα
AWS	Amazon DynamoDB	Always Free	Επ' αόριστον
AWS	Amazon RDS	Limited Time (12 μήνες)	750 ώρες τον μήνα

Πίνακας 14: Βάσεις Δεδομένων - Σύγκριση προτάσεων προσφορών

Στον πίνακα 15 φαίνονται οι προδιαγραφές των μη-σχεσιακών βάσεων δεδομένων. Η Firestore της Google φαίνεται να δίνει την μικρότερη χωρητικότητα από τις τρεις.

Σχετικά με την σύγκριση των κλήσεων, η κάθε εταιρεία έχει τον δικό της τρόπο να κάνει κοστολόγηση της χρήσης επομένως, λόγο των διαφορετικών μονάδων μέτρησης είναι δύσκολη μία σύγκριση μόνο με τα στοιχεία του πίνακα.

Στο άρθρο "Cosmos DB vs DynamoDB vs Cloud Datastore and Bigtable" που έγινε από τον Mike Vanbuskirk, 2021 [104], για την σύγκριση μεταξύ των δύο από τις τρεις υπηρεσίες ως προς το κόστος, παρατηρήθηκε πως η DynamoDB της AWS ήταν οικονομικότερη από την CosmosDB της Microsoft, και η Cloud Datastore ήταν στην 3η θέση (Η Cloud Datastore πλέον αποτελεί mode της Firestore υπηρεσίας για την υποστήριξη backwards compatibility με την Datastore η οποία αποτελεί παλαιό προϊόν της Google [105]).

Στην συνέχεια, οι τιμές κοστολόγησης της υπηρεσίας Cloud Datastore που συγκρίνεται στο άρθρο, είναι μικρότερες σε σύγκριση με εκείνες της Cloud Firestore αλλά έχουν τις ίδιες τιμές δωρεάν χρήσης στο Free Tier [62].

Επομένως, μπορεί να βγει το συμπέρασμα ότι ακόμα και αν η σύγκριση στο άρθρο γινόταν μεταξύ των CosmosDB, DynamoDB και Firestore, η Firestore θα βρισκόταν στην τελευταία θέση από τις τρεις όπως και η Datastore.

Υπηρεσία	Χωρητικότητα	Κλήσεις
Azure Cosmos DB	25GB	1000 Request Units ανά δευτερόλεπτο [78]
Firestore	1GB	50,000 reads, 20,000 writes ανά ημέρα
Amazon DynamoDB	25GB	Από 25 Read & Write Capacity Units ανά δευτερόλεπτο [87]

Πίνακας 15: Μη-Σχεσιακές Βάσεις Δεδομένων - Σύγκριση μεταξύ προσφερόμενων DBs

Η Azure CosmosDB εκτός από το μεσαίο συγκριτικά κόστος προσφέρει ένα ακόμη σημαντικό πλεονέκτημα, δεν είναι δεσμευμένη σε τεχνολογίες No-SQL του παρόχου της. Η υπηρεσία αυτή παρέχει API που μπορεί να συνδεθεί σε γνωστές No-SQL DBs (πχ MongoDB). Αυτό την καθιστά ισχυρότερη από τους ανταγωνιστές της καθώς, στις Firestore και DynamoDB οι χρήστες θα πρέπει να διαμορφώσουν τις βάσεις τους σε αυτές.

Στον πίνακα 16 φαίνονται οι προδιαγραφές των σχεσιακών βάσεων δεδομένων. Η Azure SQL Database φαίνεται να παρέχει τον περισσότερο χώρο αποθήκευσης και την ισχυρότερη εικονική μηχανή αλλά, βασικό της μειονέκτημα είναι ότι χρησιμοποιεί δεσμευμένες τεχνολογίες του παρόχου της.

Η υπηρεσία Amazon RDS προσφέρει υπηρεσίες σύνδεσης με MySQL και PostgreeSQL βάσεις όπως και οι υπηρεσίες Azure Database for MySQL και Azure Database for PostgreSQL της Microsoft. Οι υπηρεσίες τις Microsoft όμως προσφέρουν περισσότερο χώρο αποθήκευσης και ισχορότερα VM για αυτές. Όμως, επειδή η Amazon RDS προσφέρει και άλλες υπηρεσίες σύνδεσης (MariaDB, Microsoft SQL Server, Oracle), η τελική απόφαση για την επιλογή μεταξύ τους σχετίζεται με την ανάγκη των χρηστών ως προς κάποια συγκεκριμένη db engine.

Υπηρεσία	Χωρητικότητα	Εικονική Μηχανή
Azure SQL Database	250GB	10 transaction units S0 VM (40QPU, 10GB RAM)
Azure Database for MySQL	32GB 32GB backup	B1MS VM (1vCPU, 2GB RAM, 4GB temp)
Azure Database for PostgreSQL	32GB 32GB backup	B1MS VM (1vCPU, 2GB RAM, 4GB temp)
Amazon RDS	20GB 20GB backup + temp	db.t2.micro VM (1vCPU, 1GB RAM)

Πίνακας 16: Σχεσιακές Βάσεις Δεδομένων - Σύγκριση μεταξύ προσφερόμενων DBs

Στον πίνακα 17 ακολουθεί ο πίνακας τελικών συμπερασμάτων των υπηρεσιών.

Υπηρεσία	Αδυναμίες	Πλεονεκτήματα
Azure Cosmos DB	• Περιορισμένη υποστήριξη σε εκδόσεις τεχνολογιών	• Αποδοτικότερη λύση από την Firestore • Σύνδεση με άλλες εμπορικές No-SQL DB engines
Firestore	• Δεν συνδέεται με άλλες No-SQL engines	-
Amazon DynamoDB	• Δεν συνδέεται με άλλες No-SQL engines	• Αποδοτικότερη λύση No-SQL
Azure SQL Database	• Δεν συνδέεται με άλλες SQL engines • Limited Free (12 μήνες) 750 ώρες τον μήνα	• Αποδοτικότερη λύση SQL
Azure Database for MySQL	• Περιορισμένη υποστήριξη σε εκδόσεις τεχνολογιών • Limited Free (12 μήνες) 750 ώρες τον μήνα	• Αποδοτικότερη λύση για υποστήριξη MySQL
Azure Database for PostgreSQL	• Περιορισμένη υποστήριξη σε εκδόσεις τεχνολογιών • Δεν υποστηρίζει την έκδοση 12 • Limited Free (12 μήνες) 750 ώρες τον μήνα	• Αποδοτικότερη λύση για υποστήριξη PostgreSQL
Amazon RDS	• Χαμηλές προδιαγραφές VM • Περιορισμένη υποστήριξη σε εκδόσεις τεχνολογιών • Limited Free (12 μήνες) 750 ώρες τον μήνα	• Σύνδεση με άλλες εμπορικές SQL DB engines • Υποστηρίζει και την έκδοση 12

Πίνακας 17: Βάσεις Δεδομένων - Συγκριτικά συμπεράσματα μεταξύ των προσφερόμενων υπηρεσιών

Υπηρεσίες Μηνυμάτων (Message Services)

Google Cloud’s Pub/Sub

Η υπηρεσία Pub/Sub της Google Cloud αποτελεί μία υπηρεσία μηνυμάτων πραγματικού χρόνου που επιτρέπει την αποστολή μηνυμάτων μεταξύ ανεξάρτητων εφαρμογών [64].

Στο Free Tier (Always Free) η υπηρεσία Pub/Sub παρέχει 10GB μηνυμάτων ανά μήνα.

Microsoft Azure's Service Bus

Η υπηρεσία Service Bus της Microsoft αποτελεί μία υπηρεσία αποστολής μηνυμάτων μεταξύ ανεξάρτητων εφαρμογών [67].

Στο Free Tier (Limited Time) η υπηρεσία Service Bus παρέχει 13 εκατομμύρια αποστολές μηνυμάτων τον μήνα για τους πρώτους 12 μήνες από την ενεργοποίηση του λογαριασμού [73].

Amazon Web Services

Amazon SNS

Η υπηρεσία Amazon SNS της Amazon Web Services αποτελεί μία υπηρεσία αποστολής ειδοποιήσεων μεταξύ ανεξάρτητων εφαρμογών [88].

Στο Free Tier (Always Free) η υπηρεσία Amazon SNS παρέχει τα παρακάτω:

- 1 εκατομμύριο ειδοποιήσεις προς κινητές συσκευές
- 1,000 ειδοποιήσεις προς E-mail/E-mail-JSON
- 100,000 ειδοποιήσεις προς HTTP/s
- 1GB κίνησης εξόδου τον μήνα

Amazon SQS

Η υπηρεσία Amazon SQS της Amazon Web Services αποτελεί μία υπηρεσία αποστολής μηνυμάτων σε ουρά μεταξύ ανεξάρτητων εφαρμογών [106].

Στο Free Tier (Always Free) η υπηρεσία Amazon SQS παρέχει τα παρακάτω:

- 1 εκατομμύριο αποστολές μηνυμάτων τον μήνα
- 1GB κίνησης εξόδου τον μήνα

Amazon MQ

Η υπηρεσία Amazon MQ της Amazon Web Services αποτελεί μία υπηρεσία έτοιμα διαμορφωμένων open-source message brokers [107].

Στο Free Tier (Limited Time) η υπηρεσία Amazon MQ παρέχει για τους πρώτους 12 μήνες από την ενεργοποίηση του λογαριασμού 1 στιγμιότυπο mq.t2.micro ή mq.t3.micro για χρήση σε ένα από τα δύο message brokers.

Οι open-source message brokers που υποστηρίζονται είναι οι εξής:

- ActiveMQ (5GB EFS χώρο αποθήκευσης τον μήνα)

- RabbitMQ (20GB EBS χώρο αποθήκευσης τον μήνα)

Εκδόσεις υποστήριξης μηχανών RabbitMQ [107]:

- 3.8.11
- 3.8.6
- Υποστήριξη μόνο του πρωτοκόλλου AMQP (0-9-1)

Εκδόσεις υποστήριξης μηχανών ActiveMQ [107]:

- 5.15.2 (.15, .14, .13, .12, .12, .9, .8, .6, .0)
- Υποστήριξη πρωτοκόλλων AMQP, MQTT, OpenWire, STOMP

Σύγκριση Τεχνολογιών

Στην συνέχεια πραγματοποιείται σύγκριση μεταξύ των προσφερόμενων τεχνολογιών υπηρεσιών μηνυμάτων. Στον παρακάτω πίνακα (πίνακας 18) φαίνονται οι προτάσεις προσφορών κάθε υπηρεσίας. Οι υπηρεσίες Service Bus και Amazon MQ παρέχονται μόνο για τους πρώτους 12 μήνες ενώ, οι υπόλοιπες επ' αόριστον.

Πάροχος	**Υπηρεσία**	**Τύπος Free Tier**	**Χρόνος Λειτουργίας**
Google	Pub/Sub	Always Free	Επ' αόριστον
Microsoft	Service Bus	Limited Time (12 μήνες)	750 ώρες τον μήνα
AWS	Amazon SNS	Always Free	Επ' αόριστον
AWS	Amazon SQS	Always Free	Επ' αόριστον
AWS	Amazon MQ	Limited Time (12 μήνες)	750 ώρες τον μήνα

Πίνακας 18: Υπηρεσίες Μηνυμάτων - Σύγκριση προτάσεων προσφορών

Στον πίνακα 19 φαίνονται οι παρεχόμενες υπηρεσίες. Η υπηρεσία Pub/Sub της Google παρέχει 10GB μηνυμάτων χωρίς να προσθέτει όριο στις κλήσεις όπως στις υπηρεσίες των Microsoft και AWS. Σε αυτό το σημείο προτιμητέα θα ήταν η υπηρεσία που ταιριάζει περισσότερο στις ανάγκες του χρήστη από άποψη όγκου και συχνότητας μηνυμάτων. Ως προς τον παρεχόμενο αριθμό κλήσεων προτιμητέα επιλογή είναι η Service Bus της Microsoft με 13 εκατομμύρια κλήσεις έναντι 1 εκατομμυρίου της Amazon.

Σημαντικό πλεονέκτημα προσφέρει η υπηρεσία Amazon MQ η οποία παρέχει υπηρεσίες message broker ActiveMQ και RabbitMQ χωρίς άλλους περιορισμούς πέραν του χώρου αποθήκευσης μηνυμάτων. Οι message brokers υποστηρίζουν πρωτόκολλα AMPQ, STOMP, Web Socket, κ.α παρέχοντας αρκετές δυνατότητες εκμετάλλευσης.

Υπηρεσία	Κλήσεις	Χώρος αποθήκευσης
Pub/Sub	-	10GB τον μήνα
Service Bus	13 Εκατομμύρια αποστολές μηνυμάτων τον μήνα	-
Amazon SNS	1 εκατομμύριο ειδοποιήσεις προς κινητές συσκευές 1,000 ειδοποιήσεις προς E-mail/E-mail-JSON 100,000 ειδοποιήσεις προς HTTP/s	-
Amazon SQS	1 εκατομμύριο αποστολές μηνυμάτων τον μήνα	-
Amazon MQ	-	20GB EBS χώρου αποθήκευσης σε RabbitMQ 5GB EFS χώρου αποθήκευσης σε ActiveMQ

Πίνακας 19: Υπηρεσίες Μηνυμάτων - Σύγκριση μεταξύ προσφερόμενων υπηρεσιών

Στον πίνακα 20 ακολουθεί ο πίνακας τελικών συμπερασμάτων των υπηρεσιών.

Υπηρεσία	Αδυναμίες	Πλεονεκτήματα
Pub/Sub	• Υποστήριξη API μόνο από τον πάροχο	-
Service Bus	• Υποστήριξη API μόνο από τον πάροχο • Limited Free (12 μήνες) 750 ώρες τον μήνα	• Μεγαλύτερο όριο αποστολών τον μήνα
Amazon SNS	• Υποστήριξη API μόνο από τον πάροχο	• Ειδοποιήσεις sms, email, http
Amazon SQS	• Υποστήριξη API μόνο από τον πάροχο	-
Amazon MQ	• Limited Free (12 μήνες) 750 ώρες τον μήνα	• Υποστήριξη Message Brokers

Πίνακας 20: Υπηρεσίες Μηνυμάτων - Συγκριτικά συμπεράσματα μεταξύ των προσφερόμενων υπηρεσιών

Εργαλεία Ανάπτυξης (Development Tools)

Google Cloud

Cloud Source Repositories

Η υπηρεσία Cloud Source Repositories της Google Cloud προσφέρει φιλοξενία σε ιδιωτικά git repositories στο Google Cloud [60].

Στο Free Tier (Always Free) η Cloud Source Repositories υπηρεσία παρέχει τα παρακάτω:

- Μέχρι 5 χρήστες.
- 50GB χωρητικότητας.
- 50GB εύρος ζώνης.

Cloud Build

Η υπηρεσία Cloud Build της Google Cloud αναλαμβάνει το χτίσιμο εφαρμογών σε containers και java archives διαβάζοντας πηγαίο κώδικα από source repositories όπως το Google Cloud Storage, Cloud Source Repositories, Github, Bitbucket [57].

Στο Free Tier (Always Free) το Cloud Build προσφέρεται για 120 λεπτά εκτέλεσης (χτισίματος) ανά ημέρα.

Microsoft Azure's Azure DevOps

Η υπηρεσία Azure DevOps της Microsoft αποτελεί μία πλατφόρμα αυτοματισμού, αποθήκευσης κώδικα και χτίσης εφαρμογών [68]. Τα σημαντικότερα εργαλεία που περιλαμβάνονται στην πλατφόρμα είναι τα εξής:

- Azure Pipelines. Διαχείριση χτίσης, ελέγχου και ανάπτυξης με CI/CD και σύνδεση σε οποιοδήποτε πλατφόρμα και πάροχο.
- Azure Repos. Αποθήκευση ιδιωτικών Git Repos.

Στο Free Tier (Always Free) το Azure DevOps προσφέρει τα εξής [73]:

- Έως 5 χρήστες με απεριόριστο αριθμό από ιδιωτικά repos.
- Συνολικά έως 1,800 λεπτά χτίσης για CI/CD χρήση μίας διεργασίας τον μήνα.

Amazon Web Services

AWS CodeBuild

Η υπηρεσία AWS CodeBuild της AWS αναλαμβάνει το χτίσιμο εφαρμογών και την διαχείριση του continuous integration δημιουργώντας τμήματα λογισμικού έτοιμα προς ανάπτυξη [89].

Στο Free Tier (Always Free) η AWS CodeBuild υπηρεσία προσφέρεται για 100 λεπτά εκτέλεσης (χτισίματος) τον μήνα.

AWS CodeCommit

Η υπηρεσία AWS CodeCommit της AWS προσφέρει φιλοξενία σε ιδιωτικά git repositories στο AWS [90].

Στο Free Tier (Always Free) η AWS CodeCommit υπηρεσία παρέχει τα παρακάτω:

- Μέχρι 5 χρήστες.
- 50GB χωρητικότητας.
- 10,000 Git Requests τον μήνα

AWS CodePipeline

Η υπηρεσία AWS CodePipeline της AWS αναλαμβάνει το continuous delivery καθώς επιτρέπει αυτοματοποιήσεις στην διαδικασία έκδοσης των εφαρμογών [91].

Στο Free Tier (Always Free) η AWS CodePipeline υπηρεσία προσφέρει 1 ενεργό pipeline τον μήνα προς χρήση.

Σύγκριση Τεχνολογιών

Στην συνέχεια πραγματοποιείται σύγκριση μεταξύ των προσφερόμενων εργαλείων ανάπτυξης. Στον παρακάτω πίνακα (πίνακας 21) φαίνονται οι προτάσεις προσφορών κάθε υπηρεσίας. Όλες οι υπηρεσίες παρέχονται δωρεάν και επ' αόριστον.

Πάροχος	**Υπηρεσία**	**Τύπος Free Tier**	**Χρόνος Λειτουργίας**
Google	Cloud Build	Always Free	Επ' αόριστον
Google	Cloud Source Repositories	Always Free	Επ' αόριστον
Microsoft	Azure DevOps	Always Free	Επ' αόριστον
AWS	AWS CodeBuild	Always Free	Επ' αόριστον
AWS	AWS CodeCommit	Always Free	Επ' αόριστον
AWS	AWS CodePipeline	Always Free	Επ' αόριστον

Πίνακας 21: Εργαλεία Ανάπτυξης - Σύγκριση προτάσεων προσφορών

Στον πίνακα 22 φαίνονται οι προδιαγραφές των υπηρεσιών. Η Cloud Build της Google και η Azure DevOps της Microsoft παρέχουν αυτοματοποίηση στο CI/CD ενω, η AWS χρειάζεται και την υπηρεσία AWS CodeDeploy (Δεν ανήκει στο Free Tier) για να έχει την πλήρη ίδια λειτουργικότητα.

Η Cloud Build παρέχει τον περισσότερο build χρόνο τον μήνα, αλλά χωρισμένο σε 120 λεπτά την ημέρα ενώ, η Azure DevOps παρέχει 1800 λεπτά συνολικά για όλο τον μήνα. Η Cloud Source Repositories παρέχει ίδια χωρητικότητα με την υπηρεσία AWS CodeCommit, αλλά με 50GB egress τον μήνα ενώ η CodeBuild δίνει όριο στα requests. Η Azure DevOps δεν προσδιορίζει τον αποθηκευτικό χώρο των repo της.

Υπηρεσία	Χρόνος Εκτέλεσης Ανάπτυξης (Build Time)	Όριο χρηστών	Χωρητικότητα repo	Pipelines
Cloud Build	120 λεπτά ανά ημέρα (~3600 τον μήνα)	-	-	-
Cloud Source Repositories	-	5	50GB 50GB egress	-
Azure DevOps	1800 λεπτά τον μήνα	5	-	1
AWS CodeBuild	100 λεπτά τον μήνα	-	-	-
AWS CodeCommit	-	5	50GB 10,000 Git Requests τον μήνα	-
AWS CodePipeline	-	-	-	1

Πίνακας 22: Εργαλεία Ανάπτυξης - Σύγκριση μεταξύ προσφερόμενων υπηρεσιών

Στον πίνακα 23 ακολουθεί ο πίνακας τελικών συμπερασμάτων των υπηρεσιών.

Υπηρεσία	Αδυναμίες	Πλεονεκτήματα
Cloud Build	-	• Υψηλότερος συνολικός χρόνος build • Αυτοματοποίηση CI/CD
Cloud Source Repositories	-	-
Azure DevOps	-	• Αυτοματοποίηση CI/CD
AWS CodeBuild	• Χαμηλότερος χρόνος build	-
AWS CodeCommit	• Όριο σε git requests	-
AWS CodePipeline	-	-

Πίνακας 23: Εργαλεία Ανάπτυξης - Συγκριτικά συμπεράσματα μεταξύ των προσφερόμενων υπηρεσιών

Ενορχήστρωση Κιβωτίων (Container Orchestrators)

Google Cloud's Kubernetes Engine

Η υπηρεσία Google Kubernetes Engine αποτελεί την πλατφόρμα ανάπτυξης, διαχείρισης και ενορχήστρωσης εφαρμογών Kubernetes παρεχόμενη από την Google Cloud [63] με επιπρόσθετα εργαλεία αυτοματοποίησης και γραφικές διεπαφές για την διαχείριση της.

Το Google Kubernetes Engine παρέχει δύο ήδη λειτουργίας, το “Autopilot” και το “Standard”. Οι λειτουργίες αυτές καθορίζουν το επίπεδο αυτοματισμού διαχείρισης του k8s απο τον χρήστη. Στην “Autopilot” λειτουργία, η διαχείριση των Cluster και Nodes γίνεται αυτόματα και δεν χρειάζεται διαμόρφωση. Ο χρήστης χρεώνεται μόνο για τους πόρους που χρησιμοποιεί η εφαρμογή. Στην “Standard” λειτουργία πρέπει να προσδιοριστεί η λειτουργία και διαμόρφωση των Nodes. Σε αυτή την λειτουργία ο χρήστης χρεώνεται για τα Nodes που επιλέγει να χρησιμοποιήσει [63].

Στο Free Tier (Always Free) η υπηρεσία Google Kubernetes Engine παρέχει τα παρακάτω:

- Μηδενικό κόστος διαχείρισης Clusters για ένα “Autopilot” ή “Zonal Cluster” ανά λογαριασμό χρέωσης.
- Για Clusters σε “Autopilot” λειτουργία, το κάθε Pod κοστολογείται ανά δευτερόλεπτο χρήσης vCPU, μνήμης και δίσκου.
- Για Clusters σε “Standard” λειτουργία, το κάθε Node κοστολογείται με βάση την αντίστοιχη τιμή του Compute Engine που χρησιμοποιεί.

Παρόλο που το κόστος της υπηρεσίας είναι μηδενικό, φαίνεται πως για την χρησιμοποίησή της ανάλογα την λειτουργία (“Autopilot”, “Standard”) πρέπει να χρεωθούν είτε οι υπολογιστική πόροι είτε η εικονική μηχανή.

Microsoft Azure

Azure Kubernetes Service

Η υπηρεσία Azure Kubernetes Engine αποτελεί την πλατφόρμα ανάπτυξης, διαχείρισης και ενορχήστρωσης εφαρμογών kubernetes παρεχόμενη από την Microsoft Azure [74]. Η υπηρεσία παρέχει επίσης επιπρόσθετα εργαλεία αυτοματοποίησης και γραφικές διεπαφής για την διαχείριση της.

Στο Free Tier (Always Free) η υπηρεσία Azure Kubernetes Engine παρέχεται δωρεάν. Παρόλο που το κόστος διαχείρισης είναι μηδενικό, κοστολογούνται οι εικονικές μηχανές, αποθηκευτικοί χώροι και δικτυακοί πόροι οι οποίοι θα χρησιμοποιούνται από την υπηρεσία [73].

Azure Service Fabric

Η υπηρεσία Azure Service Fabric αποτελεί μία container orchestrator πλατφόρμα χτισίματος, ανάπτυξης και διαχείρισης μικρουπηρεσιών και containers [75]. Σε σύγκριση με το Azure Kubernetes Service, η βασική διαφορά του Service Fabric είναι ότι μπορεί να αναπτύξει εκτός από Docker, και Windows Server containers, αλλά και επειδή αναλαμβάνει την αυτόματη διαχείριση των master nodes και της συντήρησης των cluster [76].

Στο Free Tier (Always Free) η υπηρεσία Azure Service Fabric παρέχεται δωρεάν. Παρόλο που το κόστος διαχείρισης είναι μηδενικό, κοστολογούνται η υπολογιστική χρήση, χώρος αποθήκευσης, δικτυακοί πόροι και οι IP διευθύνσεις του cluster [73].

Σύγκριση Τεχνολογιών

Στην συνέχεια πραγματοποιείται σύγκριση μεταξύ των προσφερόμενων τεχνολογιών ενορχήστρωσης κιβωτίων. Στον παρακάτω πίνακα (πίνακας 24) φαίνονται οι προτάσεις προσφορών. Όλες οι υπηρεσίες παρέχονται δωρεάν επ' αόριστον. Η Amazon δεν παρέχει κάποια δωρεάν υπηρεσία υποστήριξης ενορχήστρωσης κιβωτίων.

Πάροχος	**Υπηρεσία**	**Τύπος Free Tier**	**Χρόνος Λειτουργίας**
Google	Google Kubernetes Engine	Always Free	Επ' αόριστον
Microsoft	Azure Kubernetes Service	Always Free	Επ' αόριστον
Microsoft	Azure Service Fabric	Always Free	Επ' αόριστον
AWS	-	-	-

Πίνακας 24: Ενορχήστρωση Κιβωτίων - Σύγκριση προτάσεων προσφορών

Στον πίνακα 25 φαίνονται το συνολικό κόστος των υπηρεσιών. Παρατηρείται πως οι υπηρεσίες ενώ ως προς την διαχείριση του Kubernetes cluster παρέχονται δωρεάν, δεν είναι εντελώς δωρεάν καθώς υπάρχει κοστολόγηση ως προς είτε τους πόρους που θα χρησιμοποιηθούν, είτε προς τα VM που θα αναπτυχθούν οι υπηρεσίες.

Εδώ θα μπορούσαν να χρησιμοποιηθούν τα δωρεάν στιγμιότυπα VM που δίνονται αλλά, θα χρειάζονταν τουλάχιστον τρία στιγμιότυπα για την υποστήριξη των υπηρεσιών K8s και της εφαρμογής προς ανάπτυξη. Τα στιγμιότυπα που δίνονται όμως είναι το ανώτερο ένα επομένως, αυτή η υπηρεσία δεν μπορεί να χρησιμοποιηθεί εντελώς δωρεάν σε κανέναν πάροχο.

Υπηρεσία	**Κόστος Διαχείρισης**	**Κόστος Χρήσης Υποδομής**
Google Kubernetes Engine	Δωρεάν	Κόστος VMs ή Κόστος CPU, RAM, Storage
Azure Kubernetes Service	Δωρεάν	Κόστος VMs, Storage, Network
Azure Service Fabric	Δωρεάν	Κόστος CPU, RAM, Storage

Πίνακας 25: Ενορχήστρωση Κιβωτίων - Σύγκριση μεταξύ προσφερόμενων υπηρεσιών

Σε σύγκριση με τις kubernetes τεχνολογίες (GKE, AKS), το Azure Service Fabric είναι μία υπηρεσία η οποία στηρίζεται αρκετά στο τεχνολογικό stack της Microsoft. Επομένως, Cloud-agnostic τεχνολογίες όπως το kubernetes είναι καταλληλότερες για cloud native εφαρμογές.

Στον πίνακα 26 ακολουθεί ο πίνακας τελικών συμπερασμάτων των υπηρεσιών.

Υπηρεσία	Αδυναμίες	Πλεονεκτήματα
Google Kubernetes Engine	• 1 Cluster δωρεάν μόνο προς διαχείριση	• Επιλογή για αυτοματοποίηση της διαχείρισης του cluster ή όχι
Azure Kubernetes Service	• Απαιτεί περισσότερη χειροκίνητη διαχείριση για ενημερώσεις Nodes [108]	• Δωρεάν διαχείριση πάντα
Azure Service Fabric	• Ως ιδιώκτητο εργαλείο της Microsoft είναι αρκετά συνδεδεμένο με τις υπηρεσίες της	• Δωρεάν διαχείριση πάντα • Περιέχει περισσότερες δυνατότητες από το AKS σε σύνδεση με υπηρεσίες της Microsoft

Πίνακας 26: Ενορχήστρωση Κιβωτίων - Συγκριτικά συμπεράσματα μεταξύ των προσφερόμενων υπηρεσιών

Συμπεράσματα Συγκρίσεων Παρόχων

Στην ενότητα αυτή θα αποτυπωθούν τα τελικά συμπεράσματα που εξάγονται μετά την ανάλυση των δωρεάν τεχνολογιών που παρέχονται.

Google Cloud

Οι υπηρεσίες που παρέχει η Google Cloud στο Free Tier είναι λιγότερες σε ποικιλία σε σχέση με τους ανταγωνιστές της με σημαντική έλλειψη σε υπηρεσίες αποθηκευτικού χώρου και βάσεων δεδομένων αλλά και γεωγραφικές περιοχές κάλυψης αυτών. Πλεονεκτική θέση φαίνεται να έχει σε πλατφόρμες υποστήριξης serverless εφαρμογών και στο γεγονός ότι οι περισσότερες υπηρεσίες της αν και πιο αδύναμες, προσφέρονται μόνιμα δωρεάν και όχι μόνο για διάρκεια 12 μηνών όπως εκείνες των ανταγωνιστών της.

Μηχανές Υπολογισμών: Η πιο αδύναμη σε ισχύ και περιορισμένη σε Linux σε σχέση με τους ανταγωνιστές της αλλά παρέχεται πάντα δωρεάν προς χρήση, σε αντίθεση με τους 12 μήνες που παρέχονται τα VM των ανταγωνιστών.

Πλατφόρμες Διαχείρισης και Εκτέλεσης Εφαρμογών: Παρέχει αποδοτικές λύσεις με ανταγωνιστικό χρόνο λειτουργίας σε πλατφόρμες διαχείρισης και υποστήριξης εφαρμογών. Πλεονέκτημα παρουσιάζει η υπηρεσία Cloud Run η οποία εκτελεί serverless containerized εφαρμογές και κοστολογείται μόνο όταν εκτελείται. Επίσης, η Cloud Functions υπηρεσία προσφέρει τον περισσότερο υπολογιστικό χρόνο σε σχέση με αντίστοιχες τεχνολογίες ανταγωνιστών.

Αποθηκευτικός Χώρος: Παρέχει μόνο μία υπηρεσία διαχείρισης αποθηκευτικού χώρου χαμηλών επιδόσεων αλλά, αποτελεί την μοναδική που παρέχεται μόνιμα δωρεάν σε σύγκριση με τους ανταγωνιστές της.

Βάσεις Δεδομένων: Προσφέρει μία υπηρεσία No-SQL βάσης μόνο document-oriented και με χαμηλή χωρητικότητα. Βασικό μειονέκτημα είναι επίσης ότι πρόκειται για ιδιόκτητη τεχνολογία που δεν συνδέεται με άλλες No-SQL Engines. Παράλληλα, δεν διαθέτει καθόλου υποστήριξη SQL βάσεων στο Free Tier.

Υπηρεσίες Μηνυμάτων: Παρέχει αρκετό χώρο αποθήκευσης για μεταφορά μηνυμάτων αλλά δεν αναφέρει τον μέγιστο αριθμό κλήσεων μηνιαίος όπως με παρόμοιες τεχνολογίες ανταγωνιστών.

Εργαλεία Ανάπτυξης: Ικανοποιητική λύση σε παροχή αυτοματοποίησης CI/CD με τα περισσότερα λεπτά χτίσης (build time), υπηρεσία που δεν παρέχει η AWS δωρεάν.

Ενορχήστρωση Κιβωτίων: Συγκριτικά με τους ανταγωνιστές της, παρέχει περισσότερες αυτοματοποιήσεις διαχείρισης κατ' επιλογή για Kubernetes. Η δωρεάν διαχείριση όμως προσφέρεται για έναν cluster. Η υπηρεσία δεν μπορεί να χρησιμοποιηθεί εντελώς δωρεάν καθώς υπάρχει κοστολόγηση στην υποστήριξη της υποδομής της (VMs, CPU, RAM, Storage).

Microsoft Azure

Η υπηρεσίες που προσφέρει η Microsoft Azure στο Free Tier είναι περισσότερες και σε μεγαλύτερη ποικιλία συγκριτικά με της Google Cloud. Οι σημαντικότερες υπηρεσίες δίνονται για τους πρώτους 12 μήνες από την ενεργοποίηση του λογαριασμού αλλά, κυρίως στον αποθηκευτικό χώρο και τις βάσεις δεδομένων προσφέρουν μεγαλύτερη ποικιλία λύσεων καλύπτοντας περισσότερα σενάρια χρήσης πελατών. Τέλος, οι υπηρεσίες παρέχονται με integrations και υποστήριξη σε αρκετές εφαρμογές της Microsoft και τον Windows.

Μηχανές Υπολογισμών: Παρέχει 2 στιγμιότυπα με Windows ή Linux, προσφορά καλύτερη από της Google αλλά μόνο για τους πρώτους 12 μήνες χρήσης.

Πλατφόρμες Διαχείρισης και Εκτέλεσης Εφαρμογών: Προσφέρει την υπηρεσία App Service η οποία υποστηρίζει statefull εφαρμογές αλλά, με συγκριτικά λιγότερο διαθέσιμο χρόνο λειτουργίας. Σημαντικό μειονέκτημα είναι τα 60 λεπτά διαμοιραζόμενου επεξεργαστικού χρόνου την ημέρα που δίνονται για τις μέχρι συνολικά 10 εφαρμογές που μπορούν να υποστηριχτούν από την υπηρεσία.

Αποθηκευτικός Χώρος: Παρέχονται υπηρεσίες block, file, archive storage σε PaaS αλλά και επιπλέον IaaS χώρος προς χρήση στα VMs προσφέροντας μεγαλύτερη διαχείριση αποθηκευτικού χώρου στους πελάτες. Οι υπηρεσίες ισχύουν επίσης για τους πρώτους 12 μήνες χρήσης.

Βάσεις Δεδομένων: Προσφέρεται η υπηρεσία CosmosDB η οποία υποστηρίζει MongoDB και Cassadra μη-σχεσιακές βάσεις δεδομένων. Επίσης παρέχει ιδιόκτητη SQL βάση αλλά και VMs με

υποστήριξη MySQL και PostgreSQL σε σχεσιακές βάσεις. Οι υπηρεσίες εκτός της CosmosDB, ισχύουν για τους πρώτους 12 μήνες χρήσης.

Υπηρεσίες Μηνυμάτων: Παρέχει τον μεγαλύτερο αριθμό μεταφοράς μηνυμάτων (13 εκατομμύρια) σε σύγκριση με υπηρεσίες ανταγωνιστών. Η υπηρεσία ισχύει για τους πρώτους 12 μήνες χρήσης και είναι η μοναδική που δεν είναι δωρεάν για πάντα.

Εργαλεία Ανάπτυξης: Ικανοποιητική λύση σε παροχή αυτοματοποίησης CI/CD, υπηρεσία που δεν παρέχει η AWS δωρεάν.

Ενορχήστρωση Κιβωτίων: Παρέχει δύο υπηρεσίες ενορχήστρωσης με εντελώς δωρεάν την διαχείριση τους. Η Service Fabric υπηρεσία παρέχει επίσης επιπρόσθετες δυνατότητες για Windows based εφαρμογές. Η υπηρεσία δεν μπορεί να χρησιμοποιηθεί εντελώς δωρεάν καθώς υπάρχει κοστολόγηση στην υποστήριξη της υποδομής της (VMs, CPU, RAM, Storage).

Amazon Web Services

Η υπηρεσίες που προσφέρει η AWS στο Free Tier είναι περισσότερες και σε μεγαλύτερη ποικιλία συγκριτικά με των ανταγωνιστών της. Οι σημαντικότερες υπηρεσίες δίνονται για τους πρώτους 12 μήνες από την ενεργοποίηση του λογαριασμού αλλά, κυρίως στον αποθηκευτικό χώρο, τις βάσεις δεδομένων και τις υπηρεσίες μηνυμάτων προσφέρουν μεγαλύτερη ποικιλία λύσεων καλύπτοντας περισσότερα σενάρια χρήσης πελατών. Τέλος, σημαντικό μειονέκτημα έναντι των ανταγωνιστών είναι η έλλειψη υποστήριξης δωρεάν διαχείρισης της Kubernetes υπηρεσίας.

Μηχανές Υπολογισμών: Παρέχει 2 στιγμιότυπα με Windows, Linux, RHEL, SLES, προσφορά καλύτερη από της Google αλλά μόνο για τους πρώτους 12 μήνες χρήσης. Επίσης, το ένα από τα δύο στιγμιότυπα παρέχει 2vCPU επομένως, παρέχουν περισσότερες δυνατότητες από αυτά των Google και Microsoft.

Πλατφόρμες Διαχείρισης και Εκτέλεσης Εφαρμογών: Προσφέρει την υπηρεσία App Service η οποία υποστηρίζει statefull εφαρμογές. Σημαντικό μειονέκτημα είναι τα 60 λεπτά διαμοιραζόμενου επεξεργαστικού χρόνου τον μήνα που δίνονται για τις μέχρι συνολικά 10 εφαρμογές που μπορούν να υποστηριχτούν από την υπηρεσία.

Αποθηκευτικός Χώρος: Παρέχονται υπηρεσίες block, file, κ.α storage σε PaaS αλλά και επιπλέον IaaS χώρος προς χρήση στα VMs προσφέροντας μεγαλύτερη διαχείριση αποθηκευτικού χώρου στους πελάτες. Οι υπηρεσίες ισχύουν επίσης για τους πρώτους 12 μήνες χρήσης.

Βάσεις Δεδομένων: Προσφέρεται η υπηρεσία AmazonRDS η οποία υποστηρίζει αρκετές σχεσιακές βάσεις δεδομένων. Επίσης παρέχει ιδιόκτητη No-SQL βάση (DynamoDB). Οι υπηρεσίες εκτός της DynamoDB, ισχύουν για τους πρώτους 12 μήνες χρήσης.

Υπηρεσίες Μηνυμάτων: Παρέχει μεγαλύτερη ποικιλία υπηρεσιών μηνυμάτων αλλά και υποστήριξη open source message brokers μέσα από την υπηρεσία Amazon MQ. Η υπηρεσία Amazon MQ ισχύει για τους 12 πρώτους μήνες χρήσης ενώ, οι υπόλοιπες παρέχονται χωρίς χρονικό όριο.

Εργαλεία Ανάπτυξης: Συγκριτικά λιγότερες δυνατότητες σε σχέση με τους ανταγωνιστές καθώς δεν υπάρχει δωρεάν υποστήριξη αυτόματου CI/CD.

Ενορχήστρωση Κιβωτίων: Δεν παρέχεται δωρεάν κάποια υπηρεσία ενορχήστρωσης κιβωτίων.

Κεφάλαιο 4: Ανάπτυξη Συστήματος Υπολογιστικής Νέφους

Στο 4ο κεφάλαιο θα αξιοποιηθούν οι γνώσεις που αποκοιμήθηκαν στα προηγούμενα κεφάλαια και χρησιμοποιηθούν για την ανάπτυξη ενός cloud native συστήματος. Θα δοθεί έμφαση στην μεθοδολογία που θα ακολουθηθεί για τον σχεδιασμό του και στις τεχνολογίες που θα το υποστηρίξουν τόσο σε τοπικό περιβάλλον, όσο και σε περιβάλλον νέφους. Τέλος, θα παρουσιαστούν δοκιμές του συστήματος.

Προσδιορισμός Προβλήματος

Οι σύγχρονες μορφές εκπαίδευσης στοχεύουν στην αξιοποίηση νέων τεχνολογιών για την διεκπεραίωση τους. Πλατφόρμες όπως το e-class προσφέρουν δυνατότητες διαχείρισης ηλεκτρονικών μαθημάτων υποστηρίζοντας λειτουργίες όπως ασύγχρονη τηλεκπαίδευση, διαχείριση μαθημάτων, σημειώσεων, εργαστηρίων, εξετάσεων κ.α. Η χρήση διαδικτυακών πλατφορμών έχει γίνει μέρος της ζωής των μαθητών και καθηγητών προσφέροντας ποικίλες ευκολίες στην καθημερινότητα τους. Υπάρχουν όμως διαδικασίες στην εκπαίδευση που δεν έχουν κάποια τεχνολογία να τις υποστηρίξει καθώς, η μετάβαση αυτή αποτελεί νέα πρακτική που δεν έχει αξιοποιηθεί πλήρως.

Έχοντας ως στόχο την προσπάθεια βελτίωσης της καθημερινότητας των φοιτητών και καθηγητών, δόθηκε έμφαση στα προβλήματα που αντιμετωπίζουν κατά τις διάφορες διαδικασίες που αμφότεροι τηρούν κατά την διάρκεια των εξαμήνων. Παρατηρήθηκε ότι προκαλείται σύγχυση στην αρχή κάθε εξαμήνου από φοιτητές που θέλουν να εγγραφούν σε εργαστήρια, καθώς αρκετοί δεν εγγράφονται στο εργαστήριο της προτίμησης τους, αλλά σε εκείνο που προλαβαίνουν τυχαία. Η τυχαία επιλογή εργαστηρίου τους προκαλεί δυσαρέσκεια καθώς μπορεί να έρχεται αντίθετη με το πρόγραμμα μαθημάτων που ήδη έχουν. Στην προσπάθεια τους να λύσουν την κατάσταση αυτή, θα πρέπει να έρθουν σε συνεννόηση με συμφοιτητές τους μέσω τυχαίων μέσων επικοινωνίας και να βρουν κάποιον ο οποίος και εκείνος επιθυμεί να αλλάξει το εργαστήριο του. Μόλις βρουν κάποιον θα πρέπει να στείλουν αμφότεροι μήνυμα στον καθηγητή και εκείνος να κάνει την αλλαγή τους στο σύστημα του e-class.

Η διαδικασία ανταλλαγής εργαστηρίων από φοιτητές αποτελεί μία διαδικασία υποψήφια για την μεταφορά της σε κάποιο αυτοματοποιημένο σύστημα διαχείρισης. Ένα σύστημα ανταλλαγής εργαστηρίων θα βοηθούσε τους φοιτητές και τους καθηγητές προσφέροντας έναν ενιαίο τρόπο διαχείρισης του προβλήματος βελτιστοποιώντας την διεκπεραίωση των διαδικασιών που πρέπει να εκτελεστούν. Εδώ θα πρέπει να σημειωθεί πως το σύστημα δεν θα μπορεί να έχει πρόσβαση στην πλατφόρμα του e-class για να πραγματοποιήσει την ανταλλαγή εργαστηρίων μεταξύ φοιτητών, αλλά

μπορεί να παρέχει συνολικές αναλυτικές αναφορές όλων των ανταλλαγών στους αρμόδιους καθηγητές ώστε να χρειαστεί να διεκπεραιώσουν την διαδικασία αυτή μόνο μία φορά.

Με το σύστημα αμοιβαίων ανταλλαγών εργαστηρίων οι φοιτητές θα έχουν την δυνατότητα για κάθε μάθημα τους να δημιουργούν δημοσιεύσεις ανταλλαγής του εργαστηρίου τους με κάποιο άλλο είτε της επιλογής τους είτε τυχαίο. Οι φοιτητές θα μπορούν να δουν όλες τις δημοσιεύσεις από τα μαθήματα τους και αν τους ενδιαφέρει κάποια, θα μπορούν να δηλώσουν ενδιαφέρων για ανταλλαγή αλλά και να επικοινωνήσουν με τους συμφοιτητές τους μέσω συνομιλίας μηνυμάτων. Οι χρήστες που δημιούργησαν την δημοσίευση θα μπορούν να δουν όλους τους φοιτητές που δήλωσαν ενδιαφέρων για την δημοσίευση τους και θα μπορούν να επικοινωνήσουν και επιλέξουν κάποιον από αυτούς για ολοκλήρωση της ανταλλαγής. Μόλις γίνει η ανταλλαγή, τότε στο σύστημα θα φαίνονται με τα επιθυμητά τους εργαστήρια. Ο καθηγητής θα μπορεί να χρησιμοποιήσει το σύστημα για να επιλέξει το κάθε μάθημα και να δει όλες τις ανταλλαγές που έχουν γίνει με συνολικές αναφορές.

Απαιτήσεις Συστήματος και Παραδοχές

Αφού προσδιορίστηκε το πρόβλημα και παρουσιάστηκε το σύστημα που πρέπει να αναπτυχθεί, θα πρέπει να αναφερθούν οι ελάχιστες απαιτήσεις που πρέπει να υποστηρίζει. Οι απαιτήσεις αυτές είναι απαραίτητες προϋποθέσεις που πρέπει να τηρούνται για την πλήρη και εύρυθμη λειτουργία του συστήματος. Οι παραδοχές αναφέρονται στον τρόπο διεκπεραίωσης κάποιον λειτουργιών καθώς το σύστημα θα πρέπει να συνεργάζεται με την πλατφόρμα του e-class αλλά και σε παράγοντες που θα έπρεπε να ληφθούν υπόψιν για την ανάπτυξη του σε πλήρη παραγωγικό επίπεδο αλλά δεν ικανοποιούνται διότι ξεφεύγουν από τα όρια ανάλυσης της διπλωματικής.

Παραδοχές

Το σύστημα σε παραγωγικό επίπεδο θα πρέπει να τροφοδοτείται από την πλατφόρμα του e-class για τα δεδομένα φοιτητών, καθηγητών, μαθημάτων και εργαστηρίων. Επειδή δεν μπορούμε να έχουμε πρόσβαση στην πλατφόρμα του e-class για την τροφοδότηση δεδομένων, θα πρέπει να θεωρηθεί ότι τα δεδομένα δίνονται μέσα από μέθοδο η οποία θα λειτουργεί ως γεννήτρια τυχαίων δεδομένων κατάλληλα για έλεγχο του συστήματος.

Στο πρόβλημα σύνδεσης και συγχρονισμού με την πλατφόρμα του e-class, συμπεριλαμβάνεται και η λειτουργία της ανταλλαγής εργαστηρίου η οποία θα γίνεται και αποθηκεύεται μόνο στο σύστημα ανταλλαγής εργαστηρίων και όχι στο e-class.

Απαιτήσεις

Ρόλοι συστήματος. Οι ρόλοι των οντοτήτων που απαρτίζουν το σύστημα είναι οι φοιτητές και οι καθηγητές. Το σύστημα πρέπει να τους ξεχωρίζει και να έχει συγκεκριμένες λειτουργίες που εμφανίζονται στον κάθε ένα.

Είσοδος στο σύστημα. Το σύστημα πρέπει να εφαρμόζει λειτουργίες αυθεντικοποίησης και εξουσιοδότησης. Η διεκπεραίωση αιτημάτων πρέπει να γίνεται μόνο σε πιστοποιημένους χρήστες και μόνο σε εκείνους που συγκεκριμένου ρόλου σε αυτό. Τα στοιχεία εισόδου των φοιτητών και καθηγητών πρέπει να δημιουργηθούν από το σύστημα και να σταλούν σε αυτούς για να μπορούν να αυθεντικοποιηθούν.

Απαιτήσεις Ρόλου Φοιτητών

- Εμφάνιση όλων των μαθημάτων των φοιτητών στα οποία είναι εγγεγραμμένοι.
- Με την επιλογή ενός μαθήματος, οι φοιτητές μπορούν να δουν τις δημοσιεύσεις ανταλλαγής εργαστηρίων που έχουν γίνει για αυτό.
- Οι φοιτητές έχουν την επιλογή να δηλώσουν ενδιαφέρων σε μία δημοσίευση που τους ενδιαφέρει αν δεν αναφέρεται εργαστήριο προτίμησης ή αν αναφέρεται και ανήκουν σε αυτό.
- Οι φοιτητές μπορούν να δουν και να ακυρώσουν τις δημοσιεύσεις στις οποίες έχουν δηλώσει ενδιαφέρων.
- Οι φοιτητές μπορούν να δημιουργήσουν δημοσιεύσεις δηλώνοντας το μάθημα και το εργαστήριο που τους ενδιαφέρει. Το εργαστήριο στο οποίο ανήκουν δηλώνεται αυτόματα μόλις επιλέξουν το μάθημα τους. Φοιτητές που δεν ανήκουν καθόλου σε εργαστήριο δεν μπορούν να δημιουργήσουν δημοσίευση.
- Οι φοιτητές μπορούν να δουν και να ακυρώσουν τις δημοσιεύσεις τους. Επίσης, μπορούν να δουν όλες τις δηλώσεις ενδιαφέροντος από άλλους φοιτητές για την δημοσίευση τους αλλά και να εγκρίνουν μία από αυτές.
- Σε περίπτωση που ένας φοιτητής εγκρίνει μία ανταλλαγή εργαστηρίων, αμφότεροι ενημερώνονται για την ανταλλαγή και πλέον το σύστημα λειτουργεί με τις νέες αλλαγές στα εργαστήρια.
- Σε όλες τις δημοσιεύσεις και τις δηλώσεις ενδιαφέροντος, οι φοιτητές έχουν την δυνατότητα να ανοίξουν συνομιλία με τον φοιτητή που δημιούργησε την δημοσίευση ή δήλωσε το ενδιαφέρων του σε αυτή.
- Δυνατότητα εμφάνισης καρτέλας συνομιλιών στην οποία αποθηκεύονται όλες οι συνομιλίες που έχουν κάνει οι φοιτητές.
- Αποστολή ενημερώσεων σε κάθε φοιτητή που υπάρχει εκδήλωση ενδιαφέροντος σε δημοσιεύσεις του, σε ανταλλαγή που μόλις πραγματοποιήθηκε και σε μηνύματα που στάλθηκαν σε αυτόν.

Απαιτήσεις Ρόλου Καθηγητών

- Εμφάνιση όλων των μαθημάτων στα οποία ο καθηγητής είναι εγγεγραμμένος.
- Σε κάθε μάθημα, ο καθηγητής έχει την επιλογή να ενεργοποιήσει ή απενεργοποιήσει την επιλογή ανταλλαγής εργαστηρίων για το συγκεκριμένο μάθημα. Όταν η επιλογή είναι απενεργοποιημένη όλες οι διαδικασίες ανταλλαγής εργαστηρίων μεταξύ φοιτητών παγώνουν και δεν μπορούν εκτελεστούν.
- Για κάθε μάθημα ο καθηγητής μπορεί να δει όλα τα εργαστήρια που υπάρχουν σε αυτό. Επίσης, σε κάθε μάθημα μπορεί να δει όλες τις ανταλλαγές εργαστηρίων που έχουν συμβεί για εργαστήρια αυτού του μαθήματος.
- Σε κάθε εργαστήριο ο καθηγητής μπορεί να δει όλους τους φοιτητές που είναι εγγεγραμμένοι σε αυτό.

Σχεδιασμός Αρχιτεκτονικής

Αφού ολοκληρώθηκε η διαδικασία εύρεσης των απαιτήσεων που πρέπει να πληρεί το σύστημα, επόμενο στάδιο είναι ο σχεδιασμός της αρχιτεκτονικής του. Η αρχιτεκτονική του συστήματος αποτελεί τον τρόπο με τον οποίο θα συνδυαστούν και θα κατασκευαστούν όλες οι συνιστώσες που θα το απαρτίζουν.

Οι βασικές συνιστώσες κάθε συστήματος είναι:

- Διεπαφή χρήστη
- Λειτουργίες επιχειρησιακής λογικής
- Δεδομένα

Οι cloud native εφαρμογές αξιοποιούν την αρχιτεκτονική των μικρουπηρεσιών για την κατασκευή ανεξάρτητων υπηρεσιών, οι οποίες αποτελούνται από λειτουργίες επιχειρησιακής λογικής και δεδομένα. Οι μικρουπηρεσίες αποτελούν το **πί**σω μέρος (back-end) του συστήματος. Η διεπαφή χρήστη θα είναι το μπροστά μέρος (front-end) του συστήματος και αποτελεί ανεξάρτητο τμήμα το οποίο συνεργάζεται με τις μικρουπηρεσίες για την διεκπεραίωση και παρουσίαση αιτημάτων και απαντήσεων στον τελικό χρήστη.

Διεπαφή Χρήστη

Η διεπαφή χρήστη αποτελεί τον ενδιάμεσο μεταξύ της επιχειρησιακής λογικής του συστήματος και του τελικού χρήστη και θα έχει την μορφή προοδευτικής εφαρμογής ιστού (progressive web app - pwa). Οι pwa είναι εφαρμογές οι οποίες αποτελούν δυναμικές ιστοσελίδες που αναπτύσσονται χρησιμοποιώντας τεχνολογίες διαδικτύου και είναι σχεδιασμένες να λειτουργούν σε όλες τις συσκευές

που χρησιμοποιούν το διαδίκτυο (smartphone, tablet, laptop.) Η διεπαφή δεν χρειάζεται να χωριστεί σε μικρότερες υπηρεσίες καθώς δεν έχει νόημα ο επιπλέον διαχωρισμός μίας δυναμικής ιστοσελίδας σε επιπλέον τμήματα, ειδικότερα για εφαρμογές μικρής κλίμακας.

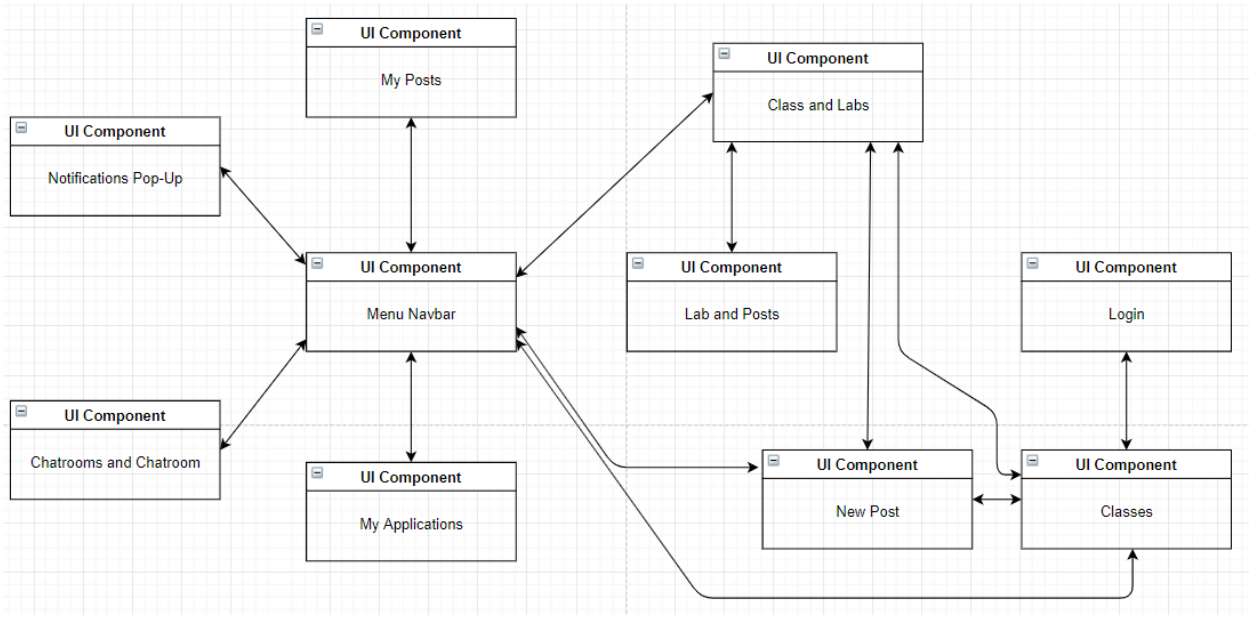
Η εφαρμογή θα αποτελείται από διαφορετικές σελίδες στις οποίες οι χρήστες θα κατευθύνονται για να διεκπεραιώσουν τις λειτουργίες τους και θα διαφέρουν ανάλογα τον ρόλο τους στο σύστημα. Οι σελίδες αυτές θα αποτελούν δυναμικά UI τμήματα της εφαρμογής και θα φορτώνονται ασύγχρονα.

Κοινές σελίδες για τους καθηγητές και τους φοιτητές είναι:

- Είσοδος στο σύστημα.

Σελίδες που μπορούν να χρησιμοποιήσουν οι μαθητές:

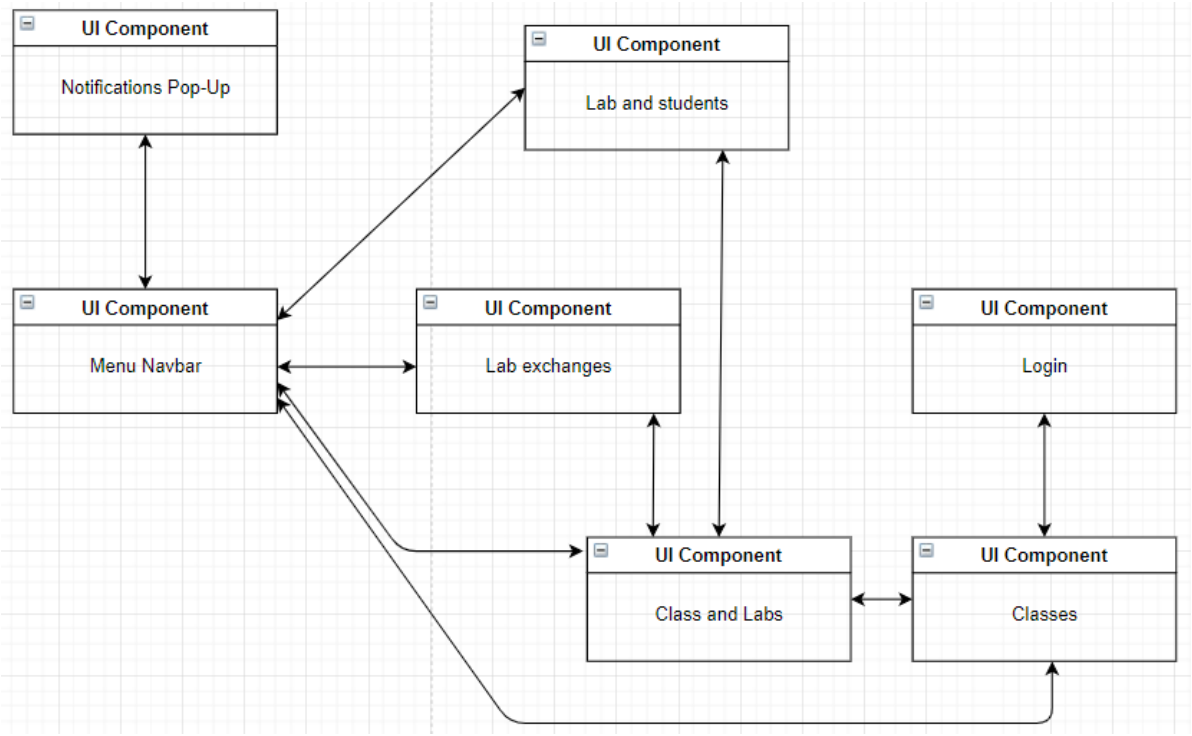
- Μαθήματα στα οποία είναι εγγεγραμμένος ο φοιτητής.
- Μάθημα και εργαστήρια μαθήματος στο οποίο είναι εγγεγραμμένος ο φοιτητής.
- Δημιουργία νέας δημοσίευσης.
- Δημοσιεύσεις σε μάθημα στο οποίο είναι εγγεγραμμένος ο φοιτητής.
- Δημοσιεύσεις που ανήκουν στον φοιτητή.
- Δημιουργία νέας δήλωσης ενδιαφέροντος.
- Δηλώσεις ενδιαφέροντος που ανήκουν στον φοιτητή.
- Παράθυρο ειδοποιήσεων.
- Συνομιλίες.
- Συνομιλία.



Σχέδιο 7: Διάγραμμα γραφικής διεπαφής ρόλου φοιτητή.

Σελίδες που μπορούν να χρησιμοποιήσουν οι καθηγητές:

- Μαθήματα στα οποία είναι εγγεγραμμένος ο καθηγητής.
- Μάθημα και εργαστήρια στα οποία είναι εγγεγραμμένος ο καθηγητής.
- Εργαστήριο και μαθητές εγγεγραμμένοι σε αυτό.
- Μάθημα και όλες οι ανταλλαγές εργαστηρίων σε αυτό.



Σχέδιο 8: Διάγραμμα γραφικής διεπαφής ρόλου καθηγητή.

Μικρουπηρεσίες

Η επιχειρησιακή λογική (business logic) αποτελεί το λογικό τμήμα της εφαρμογής που επικοινωνεί με την διεπαφή χρήστη και τα δεδομένα προκειμένου να εκτελέσει τις λειτουργίες χρηστών. Στην αρχιτεκτονική μικρουπηρεσιών οι λειτουργίες αυτές πρέπει να προσδιοριστούν και να ομαδοποιηθούν κατάλληλα σε μικρουπηρεσίες. Μετά την ομαδοποίηση των λειτουργιών – απαιτήσεων που αναφέρθηκαν προκύπτουν 4 βασικές μικρουπηρεσίες που είναι υπεύθυνες για την εκτέλεση όλων των λειτουργιών του συστήματος. Κάθε μία από τις υπηρεσίες παρουσιάζει αυτονομία στις λειτουργίες και στην διαχείριση των δεδομένων της. Η χρησιμοποίηση των μικρουπηρεσιών θα γίνεται αφού θα λειτουργεί ως υπηρεσία διαδικτύου (web service) που ακολουθεί το Restful API. Το Restful API είναι ένα είδος αρχιτεκτονικής υπηρεσιών διαδικτύου που μέσα από HTTP κλήσεις με POST, GET, PUT, DELETE μεθόδους επικοινωνεί ο τελικός χρήστης ή η γραφική διεπαφή χρήστη με τις μεθόδους της εφαρμογής μεταφέροντας δεδομένα με μία προκαθορισμένη δομή (πχ JSON). Επιπροσθέτως, το Restful API είναι stateless, γεγονός που έρχεται σε σύνδεση με τον παράγοντα 6 (Διεργασιών) της 12-factor μεθοδολογίας που απαιτεί επίσης την stateless λειτουργία διεργασιών.

Τα δεδομένα (data) αποτελούν το τμήμα της εφαρμογής που είναι υπεύθυνο για την επεξεργασία, αποθήκευση και ανάκληση των δεδομένων του συστήματος. Οι υπηρεσίες εξυπηρέτησης δεδομένων

(datastore hosts) θα πρέπει να παρέχονται σαν υπηρεσία υποστήριξης και να καλούνται από τις υπόλοιπες μικρουπηρεσίες. Το μοντέλο βάσης δεδομένων που θα υποστηρίζουν οι backing services είναι το μη-σχεσιακό (non-relational) No-SQL εξαιτίας των πλεονεκτημάτων που παρουσιάζει στην ανάπτυξη με τεχνολογίες υπολογιστικής νέφους.

Ο τύπος No-SQL βάσης δεδομένων που θα επιλεγεί είναι ο document-oriented. Οι document-oriented βάσεις μας προσφέρουν ευκολία στην κατανόηση και τον σχεδιασμό των οντοτήτων που θα χρησιμοποιεί το σύστημα τόσο στο back-end όσο και στο front-end μέρος του καθώς θα είναι εύκολη η διαχείριση των documents ως JSON αρχεία.

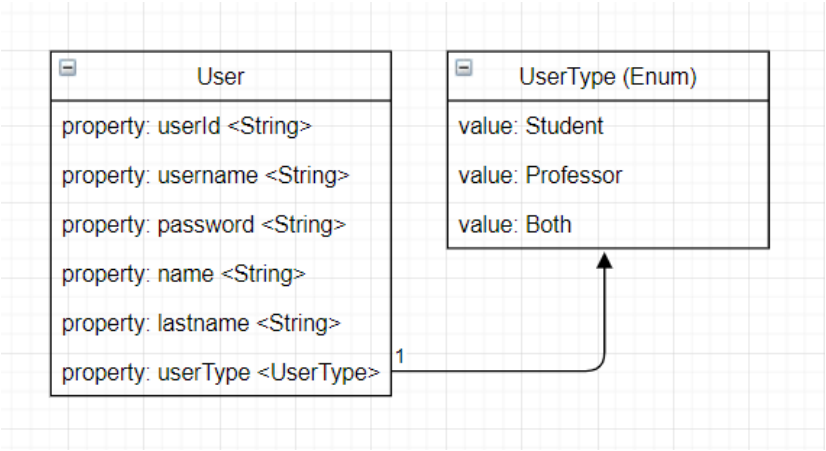
Άλλη μία επιλογή σχεδιασμού που πρέπει να γίνει εδώ είναι αν θα υπάρχει μία κεντρική υπηρεσία υποστήριξης εξυπηρέτησης που θα καλούν οι υπόλοιπες μικρουπηρεσίες για την διαχείριση των βάσεων δεδομένων τους ή θα έχουν μία ξεχωριστή υποστηρικτική υπηρεσία η κάθε μία. Εξαιτίας της μικρής κλίμακας του συστήματος, θα θεωρήσουμε ότι θα υπάρχει μία υποστηρικτική υπηρεσία παροχής βάσεων δεδομένων που θα πρέπει να καλούν οι υπόλοιπες μικρουπηρεσίες.

Στην συνέχεια ακολουθούν οι μικρουπηρεσίες που θα δημιουργηθούν με βάση τις παραπάνω επιλογές.

Υπηρεσία αυθεντικοποίησης και εξουσιοδότησης

Η υπηρεσία αυτή είναι υπεύθυνη για την είσοδο του χρήστη στο σύστημα, την αυθεντικοποίηση και παραγωγή μυστικού κλειδιού για τον κάθε χρήστη και τον έλεγχο εξουσιοδότησης του χρήστη ανάλογα τον ρόλο του. Η υπηρεσία αυτή θα καλείται τόσο από την διεπαφή χρήστη από την οποία ο χρήστης θα κάνει είσοδο στο σύστημα, όσο και από τις υπόλοιπες μικρουπηρεσίες ώστε να ελέγξουν την αυθεντικοποίηση των αιτημάτων που δέχονται όταν καλούνται.

Διάγραμμα Κλάσεων



Σχέδιο 9: Authentication service: Διάγραμμα κλάσεων.

Δομή βάσης δεδομένων

```
User {
   userId: <string, ID>,
   username: <string, UNIQUE>,
   password: <string>,
   name: <string>,
   lastname: <string>,
   userType: <string | Enumeration with values 'Student', 'Professor', 'Both'>
}
```

Σχέδιο 10: Authentication service: Σχέδιο No-SQL (document-oriented) βάσης δεδομένων χρηστών σε JSON κωδικοποίηση.

Πίνακας διεπαφής υπηρεσίας

Μονοπάτι υπηρεσίας	/account
Μονοπάτι συνάρτησης	/create-users
Περιγραφή	Η λειτουργία αυτή καλείται από την μικρουπηρεσία μαθημάτων και εργαστηρίων για την μαζική δημιουργία λογαριασμών χρηστών για είσοδο στο σύστημα.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + SECRET_INTERNAL_KEY}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: [
	{**User** Data Object}
	]
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {string}
	}

Μονοπάτι συνάρτησης	/login
Περιγραφή	Η λειτουργία αυτή χρησιμοποιείται για την αυθεντικοποίηση χρηστών με βάση το όνομα λογαριασμού και τον κωδικό τους, και την δημιουργία και αποστολή μυστικού κλειδιού στον χρήστη για την δυνατότητα χρησιμοποίησης άλλων υπηρεσιών.

HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	-
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {
	username: {string},
	password: {string}
	}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {object | contains encrypted JWT secret KEY}
	}

Μονοπάτι συνάρτησης	/validate-key
Περιγραφή	Η λειτουργία αυτή χρησιμοποιείται από άλλες μικρουπηρεσίες για τον έλεγχο της αυθεντικότητας του μυστικού κλειδιού που στέλνει ο τελικός χρήστης.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + SECRET_INTERNAL_KEY}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {
	jwt: {string},
	userType: {string - enumeration with values ('Student', 'Professor', 'Both')}
	}
	}

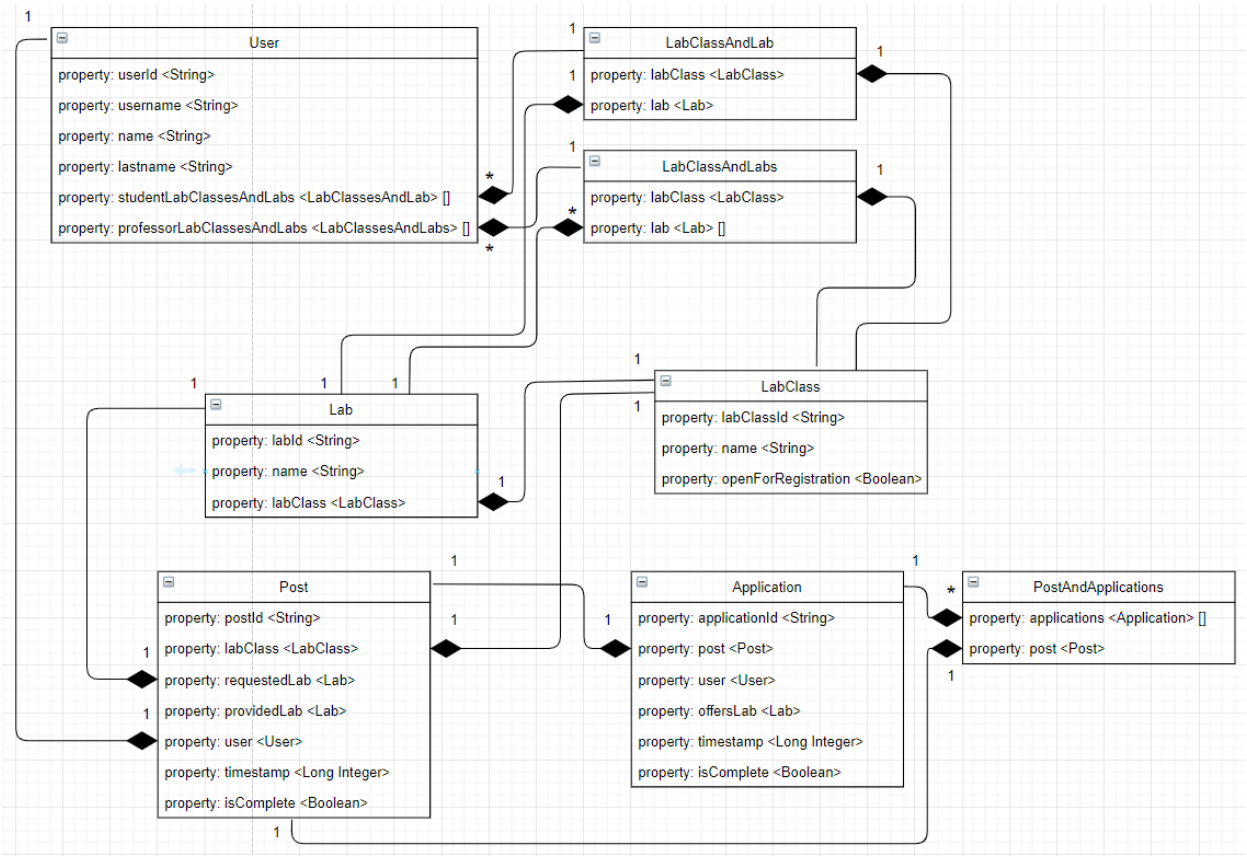
Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {
	username: {string},
	userType: {string - enumeration with values ('Student', 'Professor', 'Both')}
	}
	}

Πίνακας 27: Authentication service: Πίνακας διεπαφής.

Υπηρεσία μαθημάτων και εργαστηρίων

Στόχος της υπηρεσίας μαθημάτων και εργαστηρίων είναι η διαχείριση των πληροφοριών των βασικών οντοτήτων του συστήματος που είναι τα μαθήματα, τα εργαστήρια, οι πληροφορίες των φοιτητών και καθηγητών καθώς και οι συνδέσεις μεταξύ αυτών. Αυτή η υπηρεσία αναλαμβάνει την τροφοδότηση της υπηρεσίας αυθεντικοποίησης για την δημιουργία στοιχείων σύνδεσης των χρηστών, την ανταλλαγή εργαστηρίων και την τροφοδότηση της διεπαφής και των άλλων μικρουπηρεσιών με πληροφορίες σχετικές με τις οντότητες.

Διάγραμμα Κλάσεων



Σχέδιο 11: Classes service: Διάγραμμα κλάσεων.

Δομή βάσης δεδομένων

```
LabClass {
   labClassId: <string, ID>,
   name: <string, UNIQUE>,
   openForRegistrations: <boolean>
}
```

Σχέδιο 12: Classes service: Σχέδιο No-SQL (document-oriented) βάσης δεδομένων μαθημάτων σε JSON κωδικοποίηση.

```
Lab {
   labId: <string, ID>,
   name: <string>,
   labClass: <LabClass>
}
```

Σχέδιο 13: Classes service: Σχέδιο No-SQL (document-oriented) βάσης δεδομένων εργαστηρίων σε JSON κωδικοποίηση.

```
User {
   userId: <string, ID>,
   username: <string, UNIQUE>,
   name: <string>,
   lastname: <string>,
   studentLabClassesAndLabs: [<LabClassAndLab>],
   professorLabClassesAndLabs: [<LabClassAndLabs>]
}
```

Σχέδιο 14: Classes service: Σχέδιο No-SQL (document-oriented) βάσης δεδομένων χρηστών σε JSON κωδικοποίηση.

```
Post {
   postId: <string, ID>,
   labClass: <LabClass>,
   requestedLab: <Lab>,
   providedLab: <Lab>,
   user: <User>,
   timestamp: <long>,
   isComplete: <boolean>
}
```

Σχέδιο 15: Classes service: Σχέδιο No-SQL (document-oriented) βάσης δεδομένων δημοσιεύσεων σε JSON κωδικοποίηση.

```
Application {
   applicationId: <string, ID>,
   post: <Post>,
   user: <User>,
   offersLab: <Lab>,
   isComplete: <boolean>,
   timestamp: <long>
}
```

Σχέδιο 16: Classes service: Σχέδιο No-SQL (document-oriented) βάσης δεδομένων δήλωση ενδιαφέροντος σε JSON κωδικοποίηση.

Πίνακας διεπαφής υπηρεσίας

Μονοπάτι υπηρεσίας	/classes
Μονοπάτι συνάρτησης	/Student/get/by/me
Περιγραφή	Η λειτουργία αυτή επιστρέφει όλα τα μαθήματα και τα εργαστήρια στα οποία είναι εγγεγραμμένος ένας φοιτητής. Ο φοιτητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: ""
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: [
	{
	labClass: {**LabClass** Data Object},
	lab: {**Lab** Data Object}
	}
	]
	}

Μονοπάτι συνάρτησης	/professor/get/by/me
Περιγραφή	Η λειτουργία αυτή επιστρέφει όλα τα μαθήματα και τα εργαστήρια στα οποία είναι εγγεγραμμένος ένας καθηγητής. Ο καθηγητής προσδιορίζεται με βάση το μυστικό κλειδί του.

HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: ""
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: [
	{
	labClass: {**LabClass** Data Object},
	labs: [
	{**Lab** Data Object}
	]
	}
	]
	}

Μονοπάτι συνάρτησης	/student/get/class/by/me
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο τον κωδικό ενός μαθήματος στο οποίο ανήκει ο φοιτητής και επιστρέφει το μάθημα και το εργαστήριο στο οποίο αντιστοιχεί ο κωδικός. Ο μαθητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**LabClass** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {
	labClass: {**LabClass** Data Object},
	lab: {**Lab** Data Object}
	}
	}

Μονοπάτι συνάρτησης	/professor/get/class/by/me

Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο τον κωδικό ενός μαθήματος στο οποίο ανήκει ο καθηγητής και επιστρέφει το μάθημα και το εργαστήριο στο οποίο αντιστοιχεί ο κωδικός. Ο καθηγητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**LabClass** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {
	labClass: {**LabClass** Data Object},
	labs: [
	{**Lab** Data Object}
	]
	}
	}

Μονοπάτι συνάρτησης	/student/get/labs/by/class
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο ένα μάθημα στο οποίο ανήκει ο φοιτητής, και επιστρέφει όλα τα εργαστήρια τα οποία ανήκουν στο μάθημα αυτό. Ο μαθητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**LabClass** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: [
	{**Lab** Data Object}
	]
	}

Μονοπάτι συνάρτησης	/student/get/lab/by/me

Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο ένα μάθημα στο οποίο ανήκει ο φοιτητής, και επιστρέφει το εργαστήριο στο οποίο έχει εγγραφεί για αυτό το μάθημα. Ο μαθητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**LabClass** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {**Lab** Data Object}
	}

Μονοπάτι συνάρτησης	/professor/get/lab/by/me
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο έναν κωδικό εργαστηρίου στον οποίο ανήκει ο καθηγητής, και επιστρέφει το εργαστήριο στο οποίο αντιστοιχεί ο κωδικός. Ο καθηγητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: { **Lab** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {**Lab** Data Object}
	}

Μονοπάτι συνάρτησης	/professor/get/lab/students/by/me
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο ένα εργαστήριο στο οποίο ανήκει ο καθηγητής και επιστρέφει όλους τους εγγεγραμμένους φοιτητές σε αυτό. Ο καθηγητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}

Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: { **Lab** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: [
	{**User** Data Object}
	]
	}

Μονοπάτι συνάρτησης	/professor/toggle-registrations
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο ένα εργαστήριο στο οποίο ανήκει ο καθηγητής και το θέτει σε ενεργό ή μη ενεργό για νέες ανταλλαγές εργαστηρίων σε αυτό. Ο καθηγητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**LabClass** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: ""
	}

Μονοπάτι συνάρτησης	/get/fullname/by/username
Περιγραφή	Η λειτουργία αυτή καλείται από άλλες μικρουπηρεσίες για την απόκτηση του συνολικού φυσικού ονόματος ενός φοιτητή ή καθηγητή με βάση το όνομα λογαριασμού του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + SECRET_INTERNAL_KEY}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {string}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {string}
	}

Πίνακας 28: Classes service: Πίνακας διεπαφής μαθημάτων και εργαστηρίων.

Μονοπάτι υπηρεσίας	/posts
Μονοπάτι συνάρτησης	/get/by/class
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο ένα μάθημα στο οποίο ανήκει ο φοιτητής και επιστρέφει όλες τις δημοσιεύσεις που έχουν γίνει για εργαστήρια αυτού του μαθήματος. Ο φοιτητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**LabClass** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: [
	{
	applications: [
	{**Application** Data Object}
	],
	post: {**Post** Data Object}
	}
	]
	}

Μονοπάτι συνάρτησης	/get/by/me
Περιγραφή	Η λειτουργία αυτή επιστρέφει όλες τις δημοσιεύσεις που έχει κάνει ένας φοιτητής. Ο φοιτητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}

Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: ""
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: [
	{
	applications: [
	{**Application** Data Object}
	],
	post: {**Post** Data Object}
	}
	]
	}

Μονοπάτι συνάρτησης	/new
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο μία δημοσίευση την οποία δημιούργησε ένας φοιτητής και την αποθηκεύει στο σύστημα. Ο φοιτητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**Post** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {string}
	}

Μονοπάτι συνάρτησης	/remove
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο μία δημοσίευση την οποία δημιούργησε ένας φοιτητής και την διαγράφει. Ο φοιτητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST

Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**Post** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {string}
	}

Μονοπάτι συνάρτησης	/applications/new
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο μία δημοσίευση για την οποία ενδιαφέρεται ένας φοιτητής και δημιουργείται δήλωση ενδιαφέροντος για αυτή. Ο φοιτητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**Post** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {string}
	}

Μονοπάτι συνάρτησης	/applications/remove
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο μία δήλωση ενδιαφέροντος δημοσίευσης η οποία δημιουργήθηκε από τον φοιτητή και διαγράφεται. Ο φοιτητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**Application** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {string}
	}

Μονοπάτι συνάρτησης	/applications/get/by/me
Περιγραφή	Η λειτουργία αυτή επιστρέφει όλες τις δηλώσεις ενδιαφέροντος δημοσιεύσεων που έχει κάνει ένας φοιτητής. Ο φοιτητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {string}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: [
	{**Application** Data Object}
	]
	}

Μονοπάτι συνάρτησης	/exchange-lab
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο μία δήλωση ενδιαφέροντος δημοσίευσης την οποία θέλει να αποδεχθεί ένας φοιτητής και πραγματοποιείται η ανταλλαγή των εργαστηρίων αυτού και του ενδιαφερόμενου. Ο φοιτητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**Application** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {string}
	}

Μονοπάτι συνάρτησης	/lab-exchanges/get/by/class
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο ένα μάθημα στο οποίο ανήκει ο καθηγητής και επιστρέφει όλες τις ανταλλαγές εργαστηρίων που έχουν πραγματοποιηθεί για αυτό. Ο καθηγητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**LabClass** Data Object}
	}

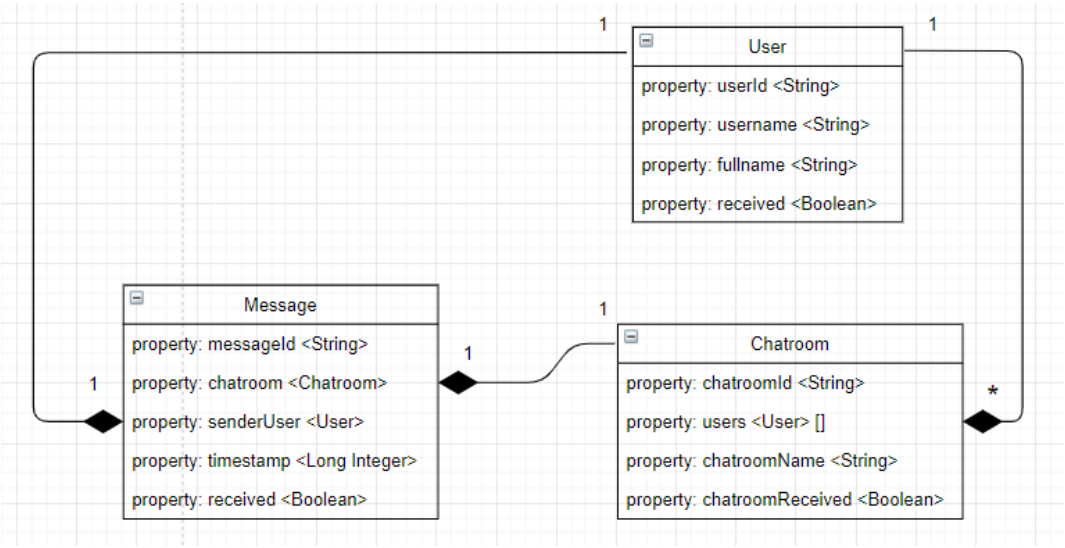
Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: [
	{**Application** Data Object}
	]
	}

Πίνακας 29: Classes service: Πίνακας διεπαφής δημοσιεύσεων και ανταλλαγών.

Υπηρεσία συνομιλιών

Στόχος της υπηρεσίας συνομιλιών είναι η διαχείριση των συνομιλιών μεταξύ των φοιτητών. Η υπηρεσία είναι υπεύθυνη για την δημιουργία δωματίου συνομιλίας μεταξύ δύο φοιτητών (chatroom) και στην συνέχεια μεταφορά μηνυμάτων μεταξύ αυτών στο δωμάτιο. Η υπηρεσία αυτή θα χρησιμοποιεί την υποστηρικτική υπηρεσία ειδοποιήσεων για την άμεση ανταλλαγή μηνυμάτων μεταξύ χρηστών χωρίς την ανάγκη ανανέωσης της εφαρμογής, και για την ενημέρωση των χρηστών σε περίπτωση που τους σταλεί μήνυμα ενώ δεν έχουν την συνομιλία ανοιχτή.

Διάγραμμα Κλάσεων



Σχέδιο 17: Messenger service: Διάγραμμα κλάσεων.

Δομή βάσης δεδομένων

```
User {
   userId: <string, ID>,
   username: <string, UNIQUE>,
   fullname: <string>,
   received: <boolean>
}
```

Σχέδιο 18: Messenger service: Σχέδιο No-SQL (document-oriented) βάσης δεδομένων χρηστών σε JSON κωδικοποίηση.

```
Chatroom {
   chatroomId: <string, ID>,
   users: [<User>],
   chatroomName: <string>,
   chatroomReceived: <boolean>
}
```

Σχέδιο 19: Messenger service: Σχέδιο No-SQL (document-oriented) βάσης δεδομένων δωματίων συνομιλιών σε JSON κωδικοποίηση.

```
Message {
   messageId: <string, ID>,
   chatroom: <Chatroom>,
   message: <string>,
   senderUser: <User>,
   timestamp: <long>,
   received: <boolean>
}
```

Σχέδιο 20: Messenger service: Σχέδιο No-SQL (document-oriented) βάσης δεδομένων μηνυμάτων σε JSON κωδικοποίηση.

Πίνακας διεπαφής υπηρεσίας

Μονοπάτι υπηρεσίας	/messenger
Μονοπάτι συνάρτησης	/student/message
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο ένα μήνυμα από έναν φοιτητή και το στέλνει στον φοιτητή παραλήπτη. Ο φοιτητής αποστολέας προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**Message** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {string}
	}

Μονοπάτι συνάρτησης	/student/chatrooms
Περιγραφή	Η λειτουργία αυτή επιστρέφει όλα τα δωμάτια συνομιλιών ενός φοιτητή. Ο φοιτητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {string}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: [
	{**Chatroom** Data Object}
	]
	}

Μονοπάτι συνάρτησης	/student/conversation
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο ένα δωμάτιο συνομιλίας στο οποίο ανήκει ο φοιτητής και επιστρέφει ολόκληρη την συνομιλία του δωματίου. Ο φοιτητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**Chatroom** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: [
	{**Message** Data Object}
	]
	}

Μονοπάτι συνάρτησης	/student/message-received

Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο ένα μήνυμα συνομιλίας της οποίας ανήκει ο φοιτητής και το θέτει ως αναγνωσμένο από αυτόν. Ο φοιτητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {Message Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {string}
	}

Μονοπάτι συνάρτησης	/student/chatroom-received
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο ένα δωμάτιο συνομιλίας στο οποίο ανήκει ο φοιτητής και το θέτει ως αναγνωσμένο από αυτόν. Ο φοιτητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**Chatroom** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {string}
	}

Μονοπάτι συνάρτησης	/student/get/chatroom
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο έναν κωδικό δωματίου συνομιλίας στο οποίο ανήκει ο φοιτητής και επιστρέφει το δωμάτιο αυτό. Ο φοιτητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST

Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {string | chatroomId}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {**Chatroom** Data Object}
	}

Μονοπάτι συνάρτησης	/student/chatroom/initialize
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο το όνομα λογαριασμού του φοιτητή στον οποίο θέλει να μιλήσει ο φοιτητής αποστολέας και δημιουργείται ένα δωμάτιο συνομιλίας μεταξύ τους. Ο φοιτητής αποστολέας προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {string | username}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {**Chatroom** Data Object}
	}

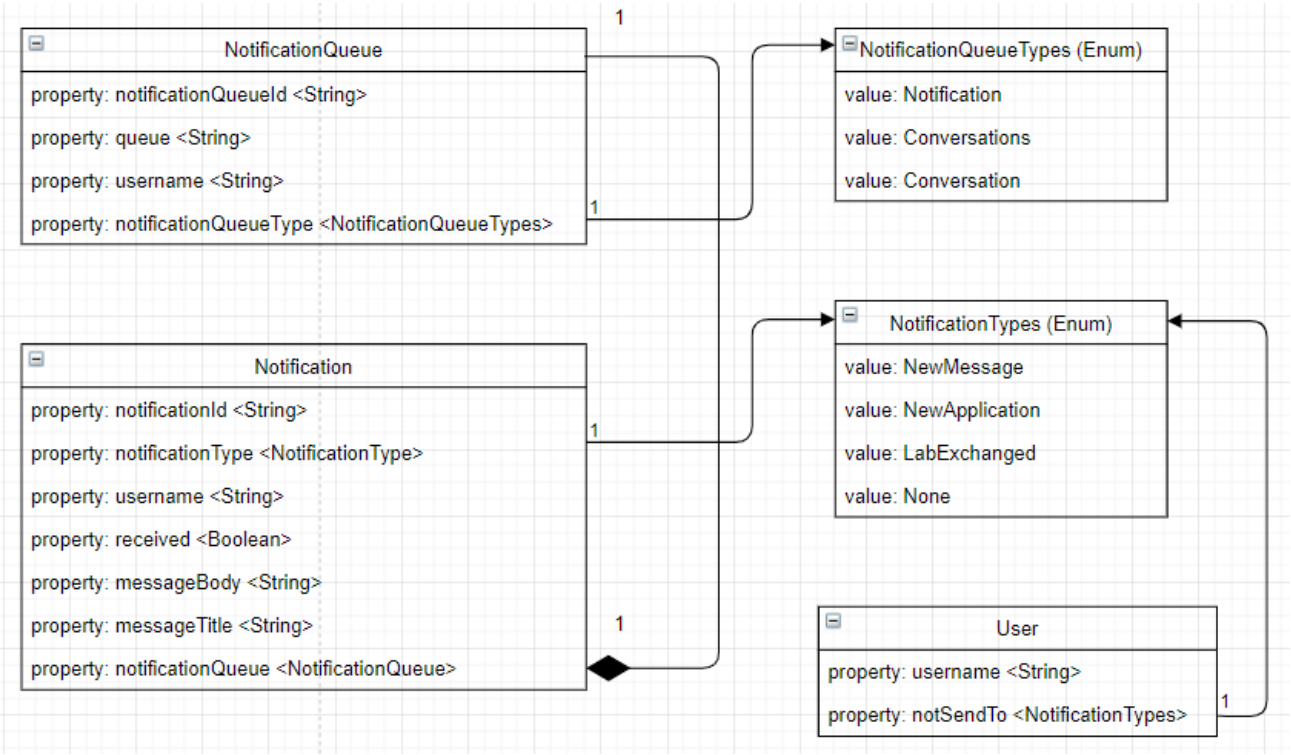
Πίνακας 30: Messenger service: Πίνακας διεπαφής.

Υπηρεσία ειδοποιήσεων

Η υπηρεσία ειδοποιήσεων θα λειτουργεί ως υποστηρικτική υπηρεσία και θα είναι υπεύθυνη για την αποστολή μηνυμάτων και ειδοποιήσεων στους τελικούς χρήστες. Άλλες μικρουπηρεσίες θα μπορούν να καλέσουν αυτή την μικρουπηρεσία για να στείλουν ειδοποιήσεις και μηνύματα στους χρήστες σχετικά με ενέργειες που έχουν συμβεί και τους ενδιαφέρουν. Οι ειδοποιήσεις είναι ειδικός τύπος μηνύματος που θέλουμε να στέλνεται κατευθείαν από την υπηρεσία χωρίς να απαιτείται αίτηση (ανανέωση της εφαρμογής) από τον χρήστη για αυτή όπως γίνεται στο πρωτόκολλο HTTP. Επομένως,

η υπηρεσία αυτή δεν θα υποστηρίζει μόνο το HTTP για τις λειτουργίες τις αλλά και WebSockets για ανταλλαγή μηνυμάτων διπλής κατεύθυνσης. Η υποστήριξη των υπηρεσιών μεταφοράς μηνυμάτων μπορεί να γίνει από αυτή την υπηρεσία χρησιμοποιώντας κάποια υποστηρικτική υπηρεσία message broker.

Διάγραμμα Κλάσεων



Σχέδιο 21: Notifications service: Διάγραμμα κλάσεων.

Δομή βάσης δεδομένων

```
User {
   username: <string, ID>,
   notSendTo: <string, Enumeration with values 'None', 'NewMessage', 'LabExchange', 'NewApplication'>
}
```

Σχέδιο 22: Notifications service: Σχέδιο No-SQL (document-oriented) βάσης δεδομένων χρηστών σε JSON κωδικοποίηση.

```
NotificationQueue {
```

```
  notificationQueueId: <string, ID>,
  queue: <string, UNIQUE>,
  username: <string>,
  notificationQueueType: <string, Enumeration with values 'notification', 'conversations', 'conversation'>
}
```

Σχέδιο 23: Notifications service: Σχέδιο No-SQL (document-oriented) βάσης δεδομένων ουρών ειδοποιήσεων σε JSON κωδικοποίηση.

```
Notification {
  notificationId: <string>,
  notificationQueue: <NotificationQueue>,
  timestamp: <long>,
  notificationType: <string, Enumeration with values 'None', 'NewMessage', 'LabExchange', 'NewApplication'>,
  username: <string>,
  received: <boolean>,
  messageTitle: <string>,
  messageBody: <string>
}
```

Σχέδιο 24: Notifications service: Σχέδιο No-SQL (document-oriented) βάσης δεδομένων ειδοποιήσεων σε JSON κωδικοποίηση.

Πίνακας διεπαφής υπηρεσίας

Μονοπάτι υπηρεσίας	/notifications
Μονοπάτι συνάρτησης	/notify
Περιγραφή	Η λειτουργία αυτή χρησιμοποιείται από άλλες μικρουπηρεσίες για την αποστολή ειδοποιήσεων στους τελικούς χρήστες. Δέχεται ως είσοδο μία ειδοποίηση και την στέλνει με web socket στον ανάλογο κωδικό ουράς χρησιμοποιώντας το message broker backing service.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + SECRET_INTERNAL_KEY}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**Notification** Data Object}
	}

Σώμα μηνύματος παραλαβής	{ status: {numeric}, message: {string}, error: {string}, timestamp: {numeric} body: {string}

	}
Μονοπάτι συνάρτησης	/get/notification-queue
Περιγραφή	Η λειτουργία αυτή καλείται από την διεπαφή του τελικού χρήστη για την αποστολή του κωδικού ουράς γενικών ειδοποιήσεων για τον συγκεκριμένο φοιτητή ώστε να μπορεί να λαμβάνει ειδοποιήσεις με web socket. Ο φοιτητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {string}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {**NoificationQueue** Data Object}
	}

Μονοπάτι συνάρτησης	/get/chatrooms-queue
Περιγραφή	Η λειτουργία αυτή καλείται από την διεπαφή του τελικού χρήστη για την αποστολή του κωδικού ουράς ειδοποιήσεων δωματίου συνομιλιών για τον συγκεκριμένο φοιτητή ώστε να μπορεί να λαμβάνει ειδοποιήσεις με web socket. Ο φοιτητής προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {string}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {**NoificationQueue** Data Object}
	}

Μονοπάτι συνάρτησης	/get/conversation-queue
Περιγραφή	Η λειτουργία αυτή καλείται από την διεπαφή του τελικού χρήστη για την αποστολή του κωδικού ουράς ειδοποιήσεων συνομιλιών για τον συγκεκριμένο χρήστη ώστε να μπορεί να λαμβάνει ειδοποιήσεις με web socket. Ο χρήστης προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {string}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {**NoificationQueue** Data Object}
	}

Μονοπάτι συνάρτησης	/notification-received
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο μία ειδοποίηση και καλείται από την διεπαφή του τελικού χρήστη για την σημείωση της ως αναγνωσμένη από εκείνον. Ο χρήστης προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**Notification** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {string}
	}

Μονοπάτι συνάρτησης	/get/notifications
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο μία ουρά ειδοποιήσεων στην οποία πρέπει να ανήκει ο χρήστης και εμφανίζει όλες τις ειδοποιήσεις που υπάρχουν για αυτήν. Ο χρήστης

	προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

Σώμα μηνύματος αποστολής	{
	body: {**NotificationQueue** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: [
	{**Notification** Data Object}
	]
	}

Μονοπάτι συνάρτησης	/block-notifications-of-type
Περιγραφή	Η λειτουργία αυτή δέχεται ως είσοδο μία ειδοποίηση στην οποία πρέπει να ανήκει ο χρήστης και σταματάει την αποστολή παρόμοιων ειδοποιήσεων μέσω web socket προς τον τελικό χρήστη. Ο χρήστης προσδιορίζεται με βάση το μυστικό κλειδί του.
HTTP μέθοδος	POST
Επικεφαλίδα μηνύματος	{Authorization | 'Bearer ' + JWT}
Τύπος μετατροπής δεδομένων	JSON

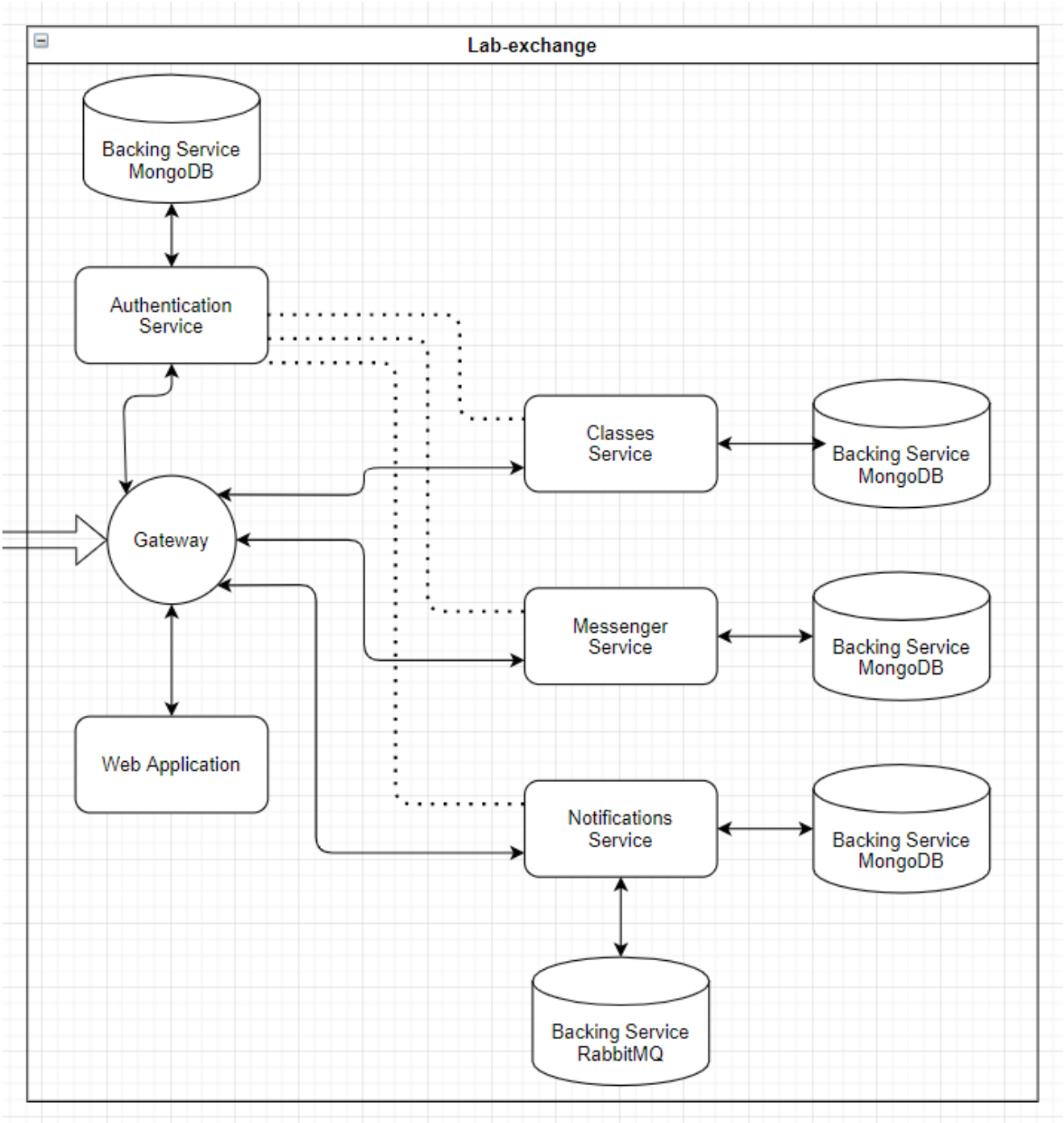
Σώμα μηνύματος αποστολής	{
	body: {**Notification** Data Object}
	}

Σώμα μηνύματος παραλαβής	{
	status: {numeric},
	message: {string},
	error: {string},
	timestamp: {numeric}
	body: {string}
	}

Πίνακας 31: Notifications service: Πίνακας διεπαφής.

Σχέδιο Αρχιτεκτονικής Υπηρεσιών

Οι συνιστώσες που προκύπτουν φαίνεται στο παρακάτω διάγραμμα αρχιτεκτονικής του συστήματος.



Σχέδιο 25: Διάγραμμα αρχιτεκτονικής υπηρεσιών εφαρμογής

Επιλογή Τεχνολογιών και Ανάπτυξη σε Τοπικό Περιβάλλον

Μετά την ανάλυση της αρχιτεκτονικής του συστήματος σε ανεξάρτητες συνιστώσες και μικρουπηρεσίες έχουν πλέον καταγραφεί οι απαιτήσεις του και είναι δυνατή η υλοποίηση του. Η διαδικασία υλοποίησης συστημάτων γίνεται αρχικά σε τοπικό περιβάλλον από τους προγραμματιστές και μόλις βρεθεί σε ένα ικανοποιητικό επίπεδο λειτουργίας τότε προχωρούν στην ανάπτυξη του σε περιβάλλον παραγωγής. Στο παρόν κεφάλαιο θα γίνει η επιλογή τεχνολογιών που θα αξιοποιηθούν για την κατασκευή και ανάπτυξη του συστήματος σε τοπικό περιβάλλον και θα ακολουθήσει η υλοποίηση αυτών.

Η επιλογή των τεχνολογιών που θα χρησιμοποιηθούν στο τοπικό περιβάλλον γίνεται προσπαθώντας να πληρείται ο 10ος παράγοντας ισοτιμίας dev/prod (dev/prod parity), κατά τον οποίο πρέπει να διατηρείται βέλτιστη ομοιότητα μεταξύ του τοπικού και παραγωγικού περιβάλλοντος. Επομένως, δίνεται έμφαση σε χρησιμοποίηση τεχνολογιών που υποστηρίζουν τόσο τοπικές λύσεις όσο και λύσεις από παρόχους υπηρεσιών.

Βάση Κώδικα

Σύμφωνα με τον 1ο παράγοντα βάσης κώδικα (codebase), το codebase της εφαρμογής θα αποτελείται από ένα σύνολο από repositories και θα τα διαχειρίζεται ένα σύστημα κατανεμημένου ελέγχου διασκευών (git). Οι συνιστώσες που έχουν αναφερθεί και αποτελούν το σύστημα είναι η διεπαφή χρήστη και οι τέσσερεις μικρουπηρεσίες μαθημάτων, αυθεντικοποίησης, μηνυμάτων και ειδοποιήσεων. Για κάθε μία από αυτές τις συνιστώσες πρέπει να δημιουργηθεί ένα ανεξάρτητο repo το οποίο θα διαχειρίζεται των κώδικα της.

Το σύστημα ελέγχου διασκευών που θα χρησιμοποιηθεί είναι το git (version 2.30.2) για Windows 10.

Εκτός από την τοπική διαχείριση του codebase στο μηχάνημα ενός προγραμματιστή, χρειάζεται ένας τρόπος να παρέχεται μέσω διαδικτύου για την ευκολία στην κατανεμημένη ανάπτυξη του από την συνολική ομάδα. Ένα σύστημα παροχής φιλοξενίας μέσω διαδικτύου στον έλεγχο έκδοσης λογισμικού χρησιμοποιώντας το git είναι το GitHub. Το Github θα χρησιμοποιηθεί για την αποθήκευση των repos για να μην εξαρτώνται από τοπικά μηχανήματα.

Για την λήψη των repo σε τοπικό μηχάνημα μέσω του github και εφόσον έχει γίνει εγκατάσταση του git, αρκεί να πληκτρολογηθεί η εντολή “git pull {url}”. Το URL για κάθε repo αναφέρεται παρακάτω:

- Web Application: https://github.com/JohnDelta/LabExchange_WebApplication.git
- Authentication service: https://github.com/JohnDelta/Authentication-Service.git
- Classes service: https://github.com/JohnDelta/Classes-Service.git
- Messenger service: https://github.com/JohnDelta/Messenger-service.git

- Notifications service: https://github.com/JohnDelta/Notification-service.git

Στο branch “master” των repos είναι διαθέσιμος ο κώδικας για την ανάπτυξη στο τοπικό περιβάλλον με K8s, Docker ενώ, στο branch “deployed_without_k8s” υπάρχει η έκδοση ανάπτυξης με χρήση υπηρεσιών νέφους.

Διεπαφή Χρήστη

Για την δημιουργία της γραφικής διεπαφής χρήστη ως PWA θα χρησιμοποιηθεί το framework React.js, και οι καθιερωμένες τεχνολογίες κατασκευής ιστοσελίδων Javascript, CSS3 και HTML5. Η React αποτελεί μία ανοιχτού κώδικα βιβλιοθήκη γραμμένη σε Javascript και χρησιμοποιείται για την κατασκευή GUI και UI τμημάτων. Με την React μπορούμε να ελέγξουμε την κατάσταση (state) της εφαρμογής μέσα από life cycle μεθόδους που διαθέτει και ανάλογα τις συνθήκες να τροποποιήσουμε το render που αποτελείται από τα συστατικά στοιχεία (components) που εμφανίζονται. Με αυτό τον τρόπο μπορούμε να αλλάζουμε δυναμικά συγκεκριμένα τμήματα - components διεπαφής χωρίς ανανεώσεις της σελίδας δίνοντας στον χρήστη την εντύπωση ότι χρησιμοποιεί desktop εφαρμογή και όχι ιστοσελίδα. Επίσης τα components μπορούν να δημιουργηθούν από επιμέρους components και αυτό οδηγεί σε μία ιεραρχική, επαναχρησιμοποιήσιμη και ευανάγνωστη δομή με την οποία μπορεί να γραφτεί η διεπαφή.

Οι δυναμικές αλλαγές τμημάτων της σελίδας προϋποθέτουν και ασύγχρονες κλήσεις στο back-end μέρος του συστήματος για την παραλαβή δεδομένων. Οι ασύγχρονες κλήσεις στο back-end γίνονται μέσω του API των μικρουπηρεσιών και γιαυτό σε cloud native συστήματα έχει νόημα η χρήση παρόμοιων framework για την κατασκευή του front-end.

Εξαρτήσεις

Εξαιτίας των περιορισμένων δυνατοτήτων της απλής έκδοσης της React, απαιτείται η προσθήκη επιπλέον βιβλιοθηκών που αυξάνουν τις λειτουργίες της. Οι απαραίτητες βιβλιοθήκες ακολουθούν παρακάτω.

- **stompjs**. Χρησιμοποιείται για την λειτουργία STOMP πελάτη στον Web browser με χρήση Web Sockets [31].
- **react-router-dom**. Χρησιμοποιείται την διαχείριση της δρομολόγησης μεταξύ GUI παραθύρων της εφαρμογής [32].

Για την ανάπτυξη της εφαρμογής σε τοπικό περιβάλλον θα χρειαστεί να έχουμε εγκατεστημένα στο μηχάνημα:

- **Node.js** (>= 10.16). Η Node.js αποτελεί πλατφόρμα ανάπτυξης λογισμικού χτισμένη σε περιβάλλον Javascript [33].

- **NMP (Node Package Manager)** (>=5.6). Το NMP αποτελεί λογισμικό διαχείρισης πακέτων (package management) για την γλώσσα προγραμματισμού Javascript [30].

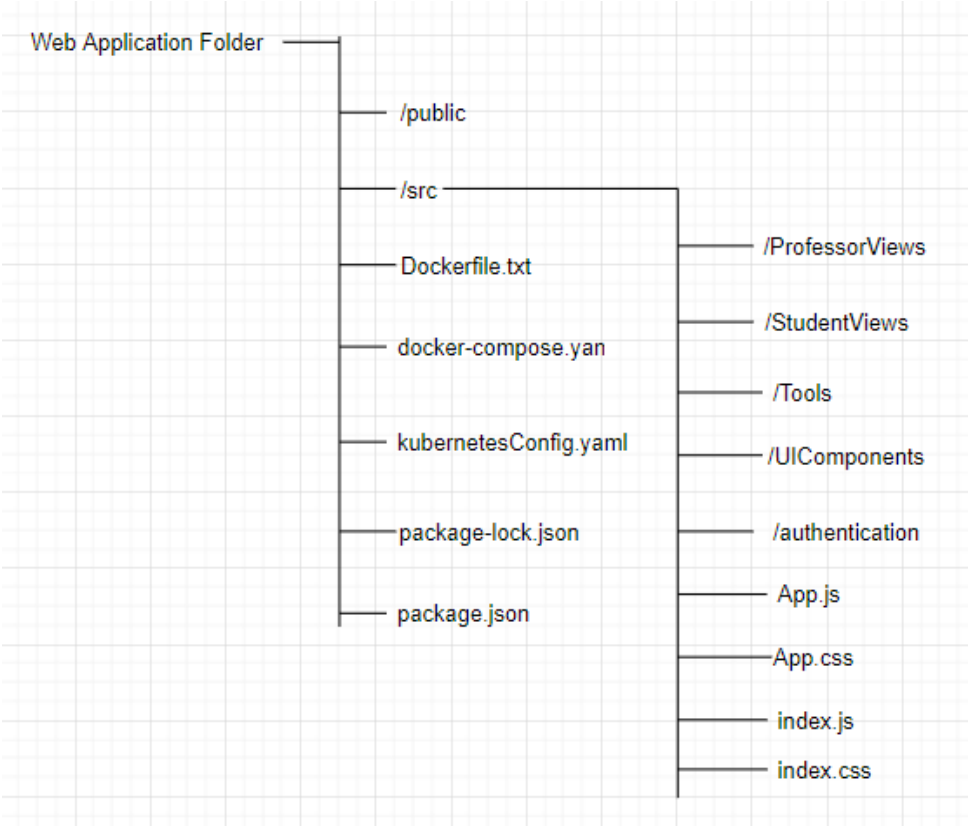
Βασική Δομή Αρχείων

Η βασική δομή της React εφαρμογής παράχθηκε με χρήση της εντολής **“npx create-react-app”** [27] η οποία δημιουργεί ένα έτοιμο προς εκτέλεση react app κατάλληλο για ανάπτυξη (Απαιτείται η χρήση των εξαρτήσεων που προαναφέρθηκαν). Το σχέδιο της τελικής βασικής δομής αρχείων της γραφικής διεπαφής φαίνεται στο παρακάτω σχήμα (Σχήμα 26).

Τα βασικά αρχεία και φάκελοι που περιέχονται είναι τα εξής:

- Στον φάκελο **/public** υπάρχουν αρχεία που δεν χρειάζονται επιπλέον επεξεργασία από το **webpack**. Το webpack αποτελεί ένα σύστημα δέσμης ανοιχτού κώδικα το οποίο αναλαμβάνει την σύνδεση τμημάτων κώδικα του front-end σε τελικά αρχεία εκτέλεσης [28]. Σε αυτό τον φάκελο υπάρχει επίσης και το index.html που αποτελεί την είσοδο στην εφαρμογή.
- Στον φάκελο **/src** εμπεριέχονται όλα τα αρχεία λειτουργιών και παρουσίασης της εφαρμογή που χρειάζονται επιπλέον επεξεργασία από το webpack για την εκτέλεση τους.
- Ο φάκελος **/ProfessorViews** περιέχει τα συστατικά στοιχεία διεπαφής για εξουσιοδοτημένους χρήστες με ρόλο καθηγητή.
- Ο φάκελος **/StudentViews** περιέχει τα συστατικά στοιχεία διεπαφής για εξουσιοδοτημένους χρήστες με ρόλο μαθητή.
- Ο φάκελος **/Tools** περιέχει δομές δεδομένων που αναπαριστούν επιχειρησιακά μοντέλα δεδομένων των μικρουπηρεσιών και επαναχρησιμοποιήσιμες μεθόδους, που χρησιμοποιούνται στον κώδικα λογικής των συστατικών τμημάτων της εφαρμογής.
- Ο φάκελος **/UIComponents** περιέχει συστατικά τμήματα της εφαρμογής τα οποία επαναχρησιμοποιούνται μέσα σε άλλα τμήματα όπως η μπάρα πλοήγησης και το παράθυρο ειδοποιήσεων.
- Ο φάκελος **/authentication** περιέχει τα συστατικά τμήματα που αναλαμβάνουν την είσοδο του χρήστη στο σύστημα και την αποθήκευση του κλειδιού ασφαλείας του χρήστη στον browser. Στην συνέχεια, ανάλογα τον ρόλο του χρήστη το σύστημα τον κατευθύνει στα κατάλληλα στοιχεία παρουσίασης.
- Τα αρχεία **App.js, App.css, index.js και index.css** χρησιμοποιούνται ως βάση για όλα τα υπόλοιπα συστατικά στοιχεία, καθώς σε αυτά υπάρχουν οι παράμετροι δρομολόγησης και ανάλογα το μονοπάτι που δίνεται από τον browser είναι υπεύθυνα για το φόρτωμα του ανάλογου συστατικού.

- Το αρχείο **Dockerfile.txt** χρησιμοποιείται για το containerization και η πλήρης λειτουργία του θα αναφερθεί σε επόμενη ενότητα.
- Το αρχείο **docker-compose.yaml** χρησιμοποιείται για το μαζικό containerization όλων των υπηρεσιών και η πλήρης λειτουργία του θα αναφερθεί σε επόμενη ενότητα.
- Το αρχείο **kubernetesConfig.yaml** χρησιμοποιείται για την ανάπτυξη των υπηρεσιών στο kubernetes και η πλήρης λειτουργία του αναφέρετε σε επόμενη ενότητα.
- Το αρχείο **package-lock.json** δημιουργείται αυτόματα για κάθε τροποποίηση συμβαίνει από το npm και επηρεάζει το package.json αρχείο. Αυτό εξασφαλίζει την χρήση σωστών εκδόσεων από ομάδες που δουλεύουν στην εφαρμογή ως προς τον έλεγχο των εξαρτήσεων που χρησιμοποιούν.
- Το αρχείο **package.json** περιέχει μεταδομένα απαραίτητα για την εφαρμογή, όπως εξαρτήσεις, scripts, εκδόσεις, κ.α.



Σχέδιο 26: Σχέδιο βασικής δομής αρχείων Γραφικής Διεπαφής

Βήματα Εκτέλεσης

Μόλις εγκατασταθούν τα προαπαιτούμενα προγράμματα, πρέπει να γίνει η εγκατάσταση των απαιτούμενων βιβλιοθηκών [31] [32] από το npm [30] με τις εντολές:

- **'npm install stompjs'**
- **'npm install react-router-dom'**

Τέλος, μπορούν να εκτελεστούν οι παρακάτω εντολές για την ανάπτυξη της εφαρμογής [29]:

- '**npm run build'**. Δημιουργία εκτελέσιμου της εφαρμογής κατάλληλο για ανάπτυξη σε περιβάλλον παραγωγής. Το εκτελέσιμο λειτουργεί ως τελική ιστοσελίδα και δεν απαιτεί Node.js και npm.
- **'npm run'**. Ανάπτυξη της εφαρμογής σε τοπικό περιβάλλον με χρήση του Node.js. Παρέχεται live server ο οποίος ανανεώνεται αυτόματα στις τροποποιήσεις του κώδικα της εφαρμογής.

Η προκαθορισμένη θύρα στην οποία παρέχεται η εφαρμογή από τον τοπικό server ανάπτυξης με την εντολή 'npm run' είναι η 3000.

Υπηρεσίες Υποστήριξης

Υπηρεσία Βάσης Δεδομένων

Για την υποστηρικτική υπηρεσία βάσης δεδομένων θα χρειαστεί μία τεχνολογία που να υποστηρίζει No-SQL document-oriented μοντέλα βάσεων. Για τον σκοπό αυτό επιλέχθηκε η MongoDB [34]. Η MongoDB είναι μία γενικού σκοπού, document-based με JSON-like encoding, κατανεμημένη βάση δεδομένων σχεδιασμένη για υποστήριξη υπηρεσιών νέφους. Η υπηρεσία θα χρησιμοποιεί την MongoDB Community Server έκδοση η οποία λειτουργεί δωρεάν και χρησιμοποιείται σε πλατφόρμες Windows, Linux και OS X. Η εγκατάσταση της MongoDB μπορεί να γίνει είτε στο λειτουργικό περιβάλλον του προγραμματιστή είτε μπορεί να χρησιμοποιηθεί το image αρχείο της και να εκτελεστεί μέσα σε περιβάλλον container και να χρησιμοποιείται από εκεί.

Με τις προκαθορισμένες ρυθμίσεις της, η MongoDB υπηρεσία δεσμεύει την θύρα 27017 και αποθηκεύει τα δεδομένα στο μονοπάτι /data/db.

Για την ενεργοποίηση της πρέπει να πληκτρολογηθεί η εντολή '**mongod'** στο system shell.

Μόλις η MongoDB εγκατασταθεί στο σύστημα μπορεί να χρησιμοποιηθεί μέσα από τις μικρουπηρεσίες για την διαχείριση και διαμόρφωση της. Πριν η υπηρεσία υποστήριξης χρησιμοποιηθεί από τις μικρουπηρεσίες θα μπορούσαν να ληφθούν επιπλέον μέτρα διαμόρφωσης της για καλύτερη εξασφάλιση της ασφάλειας του εξυπηρετητή (πχ MongoDB user access) αλλά δεν θα συμπεριληφθούν στην παρούσα ανάλυση καθώς δεν χρειάζονται.

Υπηρεσία Μηνυμάτων

Η υποστηρικτική υπηρεσία μηνυμάτων θα πρέπει να είναι ικανή να μεταφέρει ασύγχρονα μηνύματα δύο κατευθύνσεων μεταξύ των μικρουπηρεσιών και του τελικού χρήστη. Αυτή η δυνατότητα είναι δυνατή μέσα από το πρωτόκολλο STOMP (Simple Text Oriented Message Protocol) το οποίο επιτρέπει σε πελάτες να επικοινωνούν με συστήματα που το υποστηρίζουν μέσω απλών μηνυμάτων. Τα συστήματα που υποστηρίζουν πρωτόκολλα όπως το STOMP είναι οι Message Brokers, και η τεχνολογία Message Broker που θα χρησιμοποιηθεί είναι το RabbitMQ [35]. Το RabbitMQ είναι ένα message broker σύστημα ανοιχτού κώδικα, ελαφρύ και εύκολο στην ανάπτυξη, παρέχει κατανεμημένη και επεκτάσιμη λειτουργία, αναπτύσσεται από λειτουργικά συστήματα Windows, Linux και OS X και το υποστηρίζουν αρκετοί πάροχοι υπηρεσιών νέφους. Τέλος, μπορεί επίσης να χρησιμοποιηθεί το image αρχείο του και να γίνει η εκτέλεση του σε περιβάλλον container.

Για την έναρξη της υπηρεσίας θα χρειαστεί να πληκτρολογηθεί στο system shell η εντολή **‘rabbitmqctl start_app’**

Για να μπορέσει το RabbitMQ να χρησιμοποιήσει την STOMP υπηρεσία του, θα πρέπει να ενεργοποιηθεί το κατάλληλο plugin σε αυτό. Για την ενεργοποίηση του STOMP plugin θα χρειαστεί να πληκτρολογηθεί στο system shell η εντολή **‘rabbitmq-plugins enable rabbitmq_stomp’** [36].

Με τις προκαθορισμένες ρυθμίσεις της, η RabbitMQ υπηρεσία δεσμεύει τις θύρες:

- 5672, για την χρήση AMQP υπηρεσιών
- 15672, για την διεπαφη χρήσης διαχειριστής (με χρήση του management plugin)
- 61613, για το STOMP πρωτόκολλο (με χρήση του STOMP plugin)

Ενώ το όνομα λογαριασμού και ο κωδικός χρήστη για την διαχείριση της υπηρεσίας είναι **‘guest’**.

Μόλις το RabbitMQ εγκατασταθεί στο σύστημα μπορεί να χρησιμοποιηθεί μέσα από τις μικρουπηρεσίες για την διαχείριση και διαμόρφωση του. Πριν η υπηρεσία υποστήριξης χρησιμοποιηθεί από τις μικρουπηρεσίες θα μπορούσαν να ληφθούν επιπλέον μέτρα διαμόρφωσης της για καλύτερη εξασφάλιση της ασφάλειας του εξυπηρετητή (πχ αλλαγή συνθηματικών) αλλά δεν θα συμπεριληφθούν στην παρούσα ανάλυση καθώς δεν χρειάζονται.

Μικρουπηρεσίες

Όπως παρουσιάστηκε στην παραπάνω ανάλυση, οι μικρουπηρεσίες αποτελούν αυτόνομα τμήματα κώδικα τα οποία διαχειρίζονται την επιχειρησιακή λογική και τα δεδομένα. Για την υλοποίηση των μικρουπηρεσιών θα μελετηθεί η 12-factor μεθοδολογία, και σύμφωνα με αυτή θα οριστούν οι απαιτήσεις που πρέπει να πληρούν οι τεχνολογίες που θα επιλεγούν. Οι απαιτήσεις αυτές είναι οι ακόλουθες:

- Από τον **παράγοντα 2 (Εξαρτήσεων)**, φαίνεται πως θα πρέπει να υπάρχει ενσωματωμένο εργαλείο διαχείρισης εξαρτήσεων, το οποίο θα φορτώνει με μία εντολή όλες τις απαραίτητες εξαρτήσεις του κώδικα.
- Από τον **παράγοντα 4 (Υπηρεσιών υποστήριξης),** θα πρέπει να υπάρχουν εξαρτήσεις με τις οποίες η μικρουπηρεσία θα χρησιμοποιεί άλλες υποστηρικτικές υπηρεσίες για την διεκπεραίωση αιτημάτων που δεν μπορεί η ίδια να αναλάβει καθώς δεν ανήκουν στην λογική της.
- Από τον **παράγοντα 3 (Παραμετροποίησης),** θα πρέπει να υπάρχει αρχείο (config) με όλες τις παραμέτρους της εφαρμογής και θα μπορούν να φορτωθούν σε αυτή από μεταβλητές περιβάλλοντος.
- Από τον **παράγοντα 6 (Διεργασιών),** θα πρέπει οι υπηρεσίες να είναι stateless, δηλαδή να μην κρατούν τοπικά καμία κατάσταση λειτουργίας τους.
- Από τον **παράγοντα 7 (Πρόσδεσης πόρων),** θα πρέπει οι μικρουπηρεσίες να παρέχουν τις μεθόδους τους μέσα από θύρες που έχουν προσδεθεί, οι οποίες θα υπάρχουν και στο αρχείο config.
- Τέλος, οι τεχνολογίες θα πρέπει να επιτρέπουν την δημιουργία Restful Web Service υπηρεσιών, καθώς τα 12-factor apps ακολουθούν **API-First** αρχιτεκτονικές.

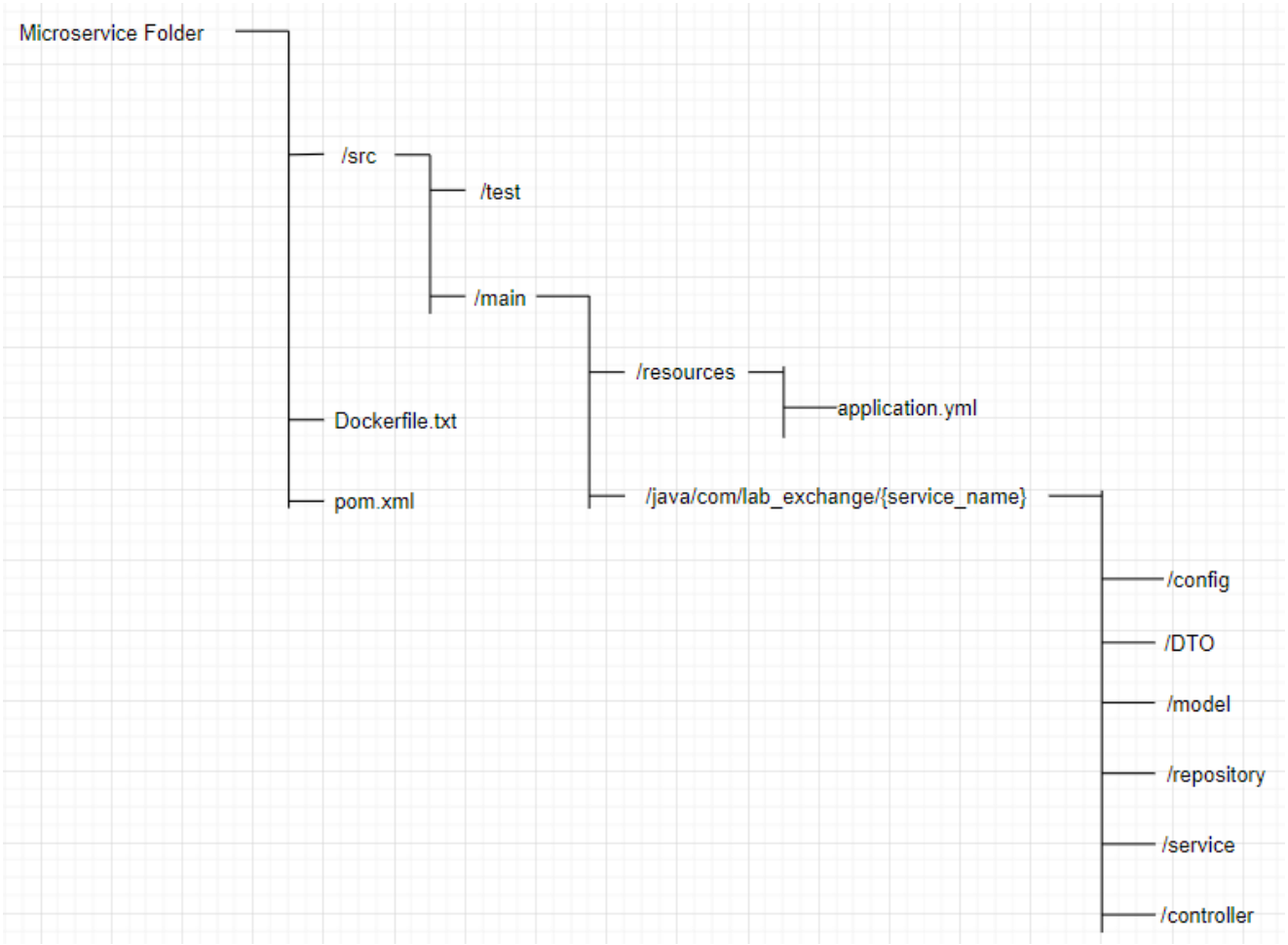
Μία τεχνολογία ικανή για την επίτευξη των παραπάνω απαιτήσεων στην κατασκευή μικρουπηρεσιών είναι το Spring Framework [37] και το Spring Boot [38]. Το Spring Framework είναι ένα framework ανοιχτού κώδικα για την γλώσσα προγραμματισμού Java το οποίο παρέχει αυτοματισμούς κτίσης βασικών δομών και συνοπτικά μοντέλα προγραμματισμού και διαμόρφωσης εφαρμογών τόσο σε κλασσικές μορφές προγραμμάτων όσο και σε διαδικτυακές εφαρμογές με χρήση του Java EE (Enterprise Edition). Το Spring Boot αποτελεί προέκταση του Spring Framework και έχει ως στόχο στην αύξηση της συγκέντρωσης των προγραμματιστών στο χτίσιμο της λογικής, αφού λαμβάνει εκείνο την ευθύνη για τις διαδικασίες ανάπτυξης της εφαρμογής σε τοπικό περιβάλλον εκτέλεσης και φορτώνοντας αυτόματα εξαρτήσεις και τροποποιήσεις που είναι απαραίτητες. Το project management tool που χρησιμοποιεί το Spring Boot Framework είναι το Maven [39]. Το Maven είναι ένα εργαλείο αυτοματισμού κατασκευής για έργα κυρίως σε Java.

Βασική Δομή Αρχείων

Το σχέδιο της βασικής δομής αρχείων των μικρουπηρεσιών φαίνεται στο παρακάτω σχήμα (Σχήμα 27). Κάθε βασικός φάκελος μικρουπηρεσιών περιέχει τους εξής φακέλους και αρχεία:

- Στον **/src** φάκελο περιέχεται ο κώδικας πηγής.
- Το **Dockerfile.txt** χρησιμοποιείται στο **containerization** της υπηρεσίας και θα αναλυθεί σε επόμενη ενότητα.

- Στο **pom.xml** περιέχονται οι εξαρτήσεις και διαμόρφωση του project από τον package manager.
- Στον **/resources** φάκελο υπάρχει το αρχείο παραμέτρων που χρησιμοποιείται στην διαμόρφωση του built εκτελέσιμου κώδικα.
- Στον φάκελο **/java/com/lab_exchange/{service_name}** υπάρχει ο βασικός κώδικας της κάθε μικρουπηρεσίας που εκτελεί την επιχειρησιακή λογική και την χρησιμοποίηση και διαμόρφωση των υποστηρικτικών υπηρεσιών.
- Στον φάκελο **/config** υπάρχουν τα αρχεία διαμόρφωσης των υποστηρικτικών υπηρεσιών.
- Στον φάκελο **/DTO** υπάρχουν τα μοντέλα δομών μετατροπής από JSON σε business model και αντίστροφα για την παρουσίαση και μεταφορά τους στο API.
- Στον φάκελο **/model** υπάρχουν τα μοντέλα δομών της βάσης δεδομένων και της επιχειρησιακής λογικής.
- Στον φάκελο **/repository** υπάρχουν **MongoRepository** των μοντέλων δεδομένων για την χρησιμοποίηση CRUD λειτουργιών στην βάση δεδομένων και την εκτέλεση queries. Το MongoRepository παρέχεται από το Spring για την διατήρηση και διαχείριση δεδομένων σε βάση δεδομένων MongoDB.
- Στον φάκελο **/service** υπάρχουν οι διάφορες λοιπές υπηρεσίες που χρησιμοποιούν οι μέθοδοι της μικρουπηρεσίας όπως οι μέθοδοι αποστολής αιτημάτων σε άλλες μικρουπηρεσίες ή η κωδικοποίηση κάποιου id.
- Στον φάκελο **/controller** υπάρχουν υλοποιημένες οι μέθοδοι της μικρουπηρεσίας που εκτίθενται στο API.



Σχέδιο 27: Σχέδιο βασικής δομής αρχείων μικρουπηρεσιών

Χειρισμός Εξαρτήσεων

Οι εξαρτήσεις ενσωματώνονται αυτόματα με το εργαλείο Maven που εμπεριέχεται στο Spring Boot Framework και έτσι πληρείται και ο παράγοντας 2 (Εξαρτήσεων). Το αρχείο εξαρτήσεων είναι στον αρχικό φάκελο με την ονομασία pom.xml [40]. Το αρχείο pom (project object model), περιέχει όλες τις πληροφορίες που χρειάζεται το Maven για την αυτοματοποίηση της κτίσης του έργου και συμπεριλαμβάνει ονόματα βιβλιοθηκών, ονομασία του έργου, έκδοση λογισμικού εκτέλεσης του προγράμματος κ.α.

Αρχείο Παραμετροποίησης

Το αρχείο παραμετροποίησης που σύμφωνα με τον παράγοντα 3 (Παραμετροποίησης) θα πρέπει να υπάρχει στην βάση κώδικα κάθε μικρουπηρεσίας βρίσκεται στον φάκελο "/resouces/" με την ονομασία application.yml. Το application.yml είναι ένα αρχείο που χρησιμοποιείται από build-in μηχανισμούς

του Spring Boot Framework για την τροποποίηση της εφαρμογής. Η δομή του application.yml παρουσιάζεται παρακάτω [41].

```
server:
  port: 8082

spring:
  application:
    name: service_name

  data:
    mongodb:
      host: localhost
      port: 27017
      database: classes
      authentication-database: admin
      username: admin
      password: admin

rabbitmq:
  stomp:
    port: 61613
  host: localhost
  username: guest
  password: guest

jwt:
  secret: totallySecretKeyTrustMe

authentication_service_host: http://localhost:8081

notification_service_host: http://localhost:8083

remote_origin: http://localhost:3000
```

Κείμενο 1: Βασική δομή αρχείου παραμετροποίησης (config.yaml) μικροϋπηρεσιών

- Αρχικά, η παράμετρος **server.port** ορίζει την θύρα στην οποία θα εκτίθενται οι μέθοδοι της υπηρεσίας.
- Η παράμετρος **spring.application.name** ορίζει το όνομα της υπηρεσίας.
- Οι παράμετροι στο **data.mongodb** μονοπάτι ορίζουν την σύνδεση της υπηρεσίας με το backing service της βάσης δεδομένων.
- Οι παράμετροι στο **rabbitmq** μονοπάτι ορίζουν την σύνδεση της υπηρεσίας με το backing service του message broker (χρησιμοποιείται μόνο στην υπηρεσία ειδοποιήσεων).

- Η παράμετρος **jwt.secret** ορίζει το κλειδί με το οποίο θα κρυπτογραφούνται τα μυστικά κλειδιά των χρηστών για την παροχή υπηρεσιών σε αυτούς από τις υπηρεσίες.
- Οι παράμετροι **authentication_service_host**, **notification_service_host** και **remote_origin** ορίζουν αντίστοιχα τις url διευθύνσεις των υπηρεσιών αυθεντικοποίησης, ειδοποιήσεων και της γραφικής διεπαφής χρήστη.

Βήματα Εκτέλεσης

Για την εκτέλεση της κάθε μικρουπηρεσίας σε τοπικό περιβάλλον, είναι απαραίτητη η εγκατάσταση μόνο του εργαλείου διαχείρισης εξαρτήσεων και του λογισμικού εκτέλεσης της γλώσσας προγραμματισμού. Με την χρήση του Spring Boot Framework οι τεχνολογίες που χρειάζονται τελικά είναι:

- **Maven (Version 4.0.0)**
- **Java (Version 15)**

Αφού έχουν εγκατασταθεί αμφότερα τα προαναφερθέντα λογισμικά, μπορεί να γίνει η κτίση του έργου σε εκτελέσιμο αρχείο από το Maven. Αυτό γίνεται με την εντολή **“mvn clean install”**.

Τέλος, για την κτίση και εκτέλεση του έργου μπορεί να πληκτρολογηθεί η εντολή **“mvn spring-boot:run”**.

Containerization

Η ανάπτυξη των μικρουπηρεσιών προκάλεσε αύξηση στην ανάπτυξη εφαρμογών με χρήση containers, διότι προσφέρουν το κατάλληλο περιβάλλον για την εύρυθμη και απομονωμένη λειτουργία υπηρεσιών. Επιπλέον, η χρήση containers έρχεται σε συμφωνία με κάποιους βασικούς παράγοντες της 12-factor μεθοδολογίας.

Docker

Η τεχνολογία που θα χρησιμοποιηθεί για την ανάπτυξη των υπηρεσιών σε containers είναι το Docker [42]. Το Docker είναι μία πλατφόρμα ανοιχτού κώδικα για την ανάπτυξη, φόρτωση και εκτέλεση εφαρμογών σε containers, η οποία επιτρέπει την διαχείριση της υποδομής με τρόπο παρόμοιο με εκείνον της διαχείρισης της εφαρμογής.

Βασικές οντότητες που χρησιμοποιεί το docker για το containerization είναι οι εξής:

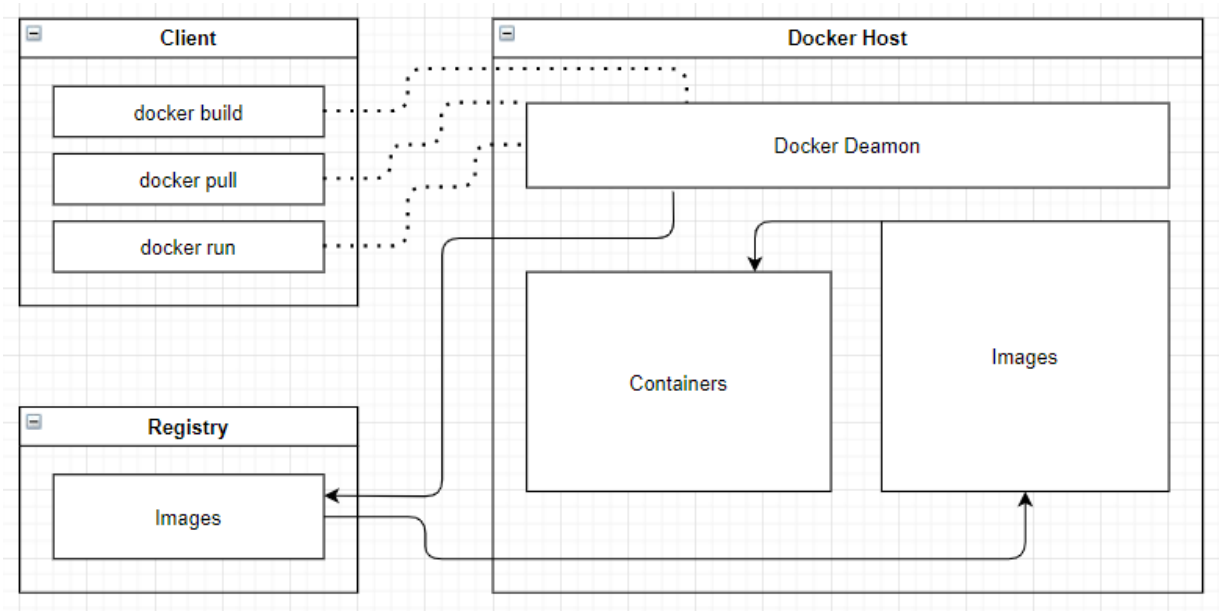
- **Εικόνες (Images)**. Τα Docker images αποτελούν πρότυπα με οδηγίες για την κατασκευή ενός container, και μπορούν να αποτελούνται από άλλα images με περισσότερες προσαρμογές. Η δημιουργία τους γίνεται με χρήση αρχείων (dockerfiles) που περιέχουν απλά βήματα εντολών.

- **Κιβώτια (Containers)**. Τα Docker containers αποτελούν εκτελέσιμα στιγμιότυπα των Docker Images. Μέσα από τον docker client είναι δυνατή η έναρξη, διαγραφή, σταμάτημα και μετακίνηση τους. Μόλις ένα container σταματηθεί, οι πληροφορίες που υπήρχαν μέσα σε αυτό θα διαγραφούν αν δεν έχει δημιουργηθεί στατικός χώρος αποθήκευσης για αυτό.
- **Δίκτυα (Networks)**. Τα Docker containers είναι απομονωμένα από άλλα containers και από το τοπικό σύστημα του εξυπηρετητή. Για την σύνδεση τους και την εύρεση της διεύθυνσης τους μεταξύ τους μπορούν να τοποθετηθούν σε κοινά δίκτυα.
- **Τομείς (Volumes)**. Για την διατήρηση δεδομένων που παράγονται από containers το docker χρησιμοποιεί τα docker volumes τα οποία τα διαχειρίζεται το docker host και όχι το αντίστοιχο docker container που τα χρησιμοποιεί.

Με την χρήση του docker υλοποιείται ο 1ος παράγοντας βάσης κώδικα (Codebase) από την χρήση των source control repositories για την αποθήκευση του codebase των υπηρεσιών. Το docker χρησιμοποιεί images για την δημιουργία των containers, τα οποία παρέχονται από το image registry και αποτελούν διαφορετικές εκδόσεις ανάπτυξης του κάθε codebase.

Ο 2ος παράγοντας εξαρτήσεων (dependencies) εφαρμόζεται από τις δυνατότητες των containers να ενθυλακώνουν όλες τις εξαρτήσεις που χρειάζεται η κάθε εφαρμογή για την λειτουργία της μέσα από την διαμόρφωση τους με χρήση των dockerfiles, στα οποία ορίζεται το εκτελέσιμο λογισμικό του προγράμματος με το οποίο γίνεται η εγκατάσταση των εξαρτήσεων μέσω του pom που χρησιμοποιεί. Επίσης με το containers παρέχεται απομόνωση των υπηρεσιών σε ικανοποιητικό βαθμό.

Το Docker χρησιμοποιεί αρχιτεκτονική πελάτη-εξυπηρετητή (Σχήμα 20). Ο Docker εξυπηρετητής (Docker Host) περιέχει τον δαίμονα διεργασιών (Docker deamon), ο οποίος ακούει σε API αιτήματα και ευθύνη του είναι το χτίσιμο, η εκτέλεση και η ανάπτυξη των containers. Επίσης, διαχειρίζεται τις υπόλοιπες Docker οντότητες όπως images, containers, networks, κ.α.



Σχέδιο 28: Διάγραμμα αρχιτεκτονικής Docker

Ο Docker πελάτης (Docker Client), αποτελεί τον βασικό τρόπο επικοινωνίας χρηστών (docker users) με τον εξυπηρετητή (docker host) χρησιμοποιώντας εντολές όπως “docker run”.

Το Docker Registry αποτελεί σύστημα αρχείου καταγραφής και αποθήκευσης Docker images. Τέτοιο σύστημα αποτελεί το Docker Hub, το οποίο είναι ένα δημόσιο σύστημα από Docker images που με εντολές όπως “docker pull” και “docker push” είναι δυνατή η λήψη και αποστολή images.

Η έκδοση του Docker που θα χρησιμοποιηθεί για την ανάπτυξη του συστήματος σε containers είναι το Docker Desktop για Windows 10.

Δημιουργία των Docker Files

Στον βασικό φάκελο κάθε υπηρεσίας βρίσκεται το docker-file [43] με τις οδηγίες που χρειάζονται για την δημιουργία του docker image της. Μερικές από τις βασικές εντολές για το χτίσιμο του docker-file ακολουθούν στην συνέχεια.

- Τα docker-files ξεκινούν με την οδηγία **FROM**, η οποία δηλώνει το parent image από το οποίο θα χτιστεί η εικόνα.
- Η οδηγία **COPY** χρησιμοποιείται για την δημιουργία/μεταφορά αρχείων μέσα στον φάκελο που υπάρχει το container.
- Με την οδηγία **RUN** εκτελούνται εντολές που αποτελούν βήματα του χτισίματος του image.

- Η οδηγία **ENTRYPOINT** ορίζει μία εντολή που θα εκτελείται πάντα κατά την έναρξη του container. Χρησιμοποιείται όταν θέλουμε το container να είναι εκτελέσιμο αρχείο χωρίς επιπλέον arguments.
- Η οδηγία **CMD** προσθέτει επιπλέον arguments που θα δοθούν στην ENTRYPOINT. Η οδηγία μπορεί να γίνει override από άλλες εντολές αν δοθούν κατά την έναρξη του container.

Για την υπηρεσία της γραφικής διεπαφής φορτώνεται το έτοιμο image του λογισμικού εκτέλεσης της Node.js, και μεταφέρονται οι βασικοί φάκελοι που χρειάζεται η react εφαρμογή για την λειτουργία της. Στην συνέχεια γίνεται μέσα από το npm η εγκατάσταση των εξαρτήσεων της εφαρμογής και τέλος με την εντολή **"npm start"** η εκτέλεση της.

```
FROM node:alpine
ENV PATH /app/node_modules/.bin:$PATH
COPY package.json ./
COPY package-lock.json ./
RUN npm install --silent
RUN npm install react-scripts@3.4.1 -g --silent
RUN npm install react-router-dom
RUN npm install stompjs
RUN npm install sockjs
COPY . ./
CMD ["npm", "start"]
```

Κείμενο 2: Dockerfile (.yaml) υπηρεσίας γραφικής διεπαφής χρήστη

Για τις μικρουπηρεσίες οι εντολές είναι ίδιες. Αρχικά φορτώνεται το βασικό image για την εκτέλεση της Java (version 15). Στην συνέχεια δημιουργείται ένας χρήστης και του δίνονται δικαιώματα για τα αρχεία της εφαρμογής. Στο σημείο αυτό θα μπορούσε ο χρήστης να έχει περιορισμένα δικαιώματα στα αρχεία αυξάνοντας την ασφάλεια του container. Στην συνέχεια μεταφέρεται το εκτελέσιμο built αρχείο της υπηρεσίας (.jar) μέσα στα αρχεία του container, και τέλος δίνεται η οδηγία εκτέλεσης του.

Για την εκτέλεση των dockerfiles μικρουπηρεσιών, είναι σημαντικό να υπάρχει το εκτελέσιμο αρχείο των υπηρεσιών (.jar) στους βασικούς φακέλους τους. Για το χτίσιμο του εκτελέσιμου αρχείου αρκεί η χρήση της εντολής **"mvn clean install"**.

```
FROM openjdk:15-jdk-alpine
RUN addgroup -S spring1 && adduser -S spring1 -G spring1
USER spring1:spring1
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} authentication.jar
ENTRYPOINT ["java","-jar","/authentication.jar"]
```

Κείμενο 3: Dockerfile (.yaml) υπηρεσίας αυθεντικοποίησης

Στο κείμενο 3 φαίνεται το dockerfile της υπηρεσίας αυθεντικοποίησης. Για τις υπόλοιπες 3 μικρουπηρεσίες τα dockerfile τους θα είναι ίδια με διαφορετική ονομασία στο .jar αρχείο τους.

Στο παρακάτω docker-file (Κείμενο 4) φαίνεται η τροποποίηση που απαιτείται για την χρησιμοποίηση του RabbitMQ καθώς, εκτός από το βασικό image θα πρέπει να γίνει και η ενεργοποίηση του plugin για την υποστήριξη του STOMP.

```
FROM rabbitmq:3.7-management
RUN apt-get update && \
apt-get install -y curl unzip
RUN curl https://dl.bintray.com/rabbitmq/community-plugins/3.7.x/rabbitmq_delayed_message_exchange/
rabbitmq_delayed_message_exchange-20171201-3.7.x.zip > rabbitmq_delayed_message_exchange-20171201-3.7.x.zip
&& \
unzip rabbitmq_delayed_message_exchange-20171201-3.7.x.zip && \
rm -f rabbitmq_delayed_message_exchange-20171201-3.7.x.zip && \
mv rabbitmq_delayed_message_exchange-20171201-3.7.x.ez plugins/

RUN rabbitmq-plugins enable rabbitmq_stomp
```

Κείμενο 4: Dockerfile (.yaml) υποστηρικτικής υπηρεσίας rabbitMQ

Για την δημιουργία του image της υποστηρικτικής υπηρεσίας MongoDB δεν χρειάζεται κάποια επιπλέον προσαρμογή για την λειτουργία της επομένως απαιτείται μόνο η εντολή **“docker pull mongo”** για την λήψη της από το Docker Hub.

Δημιουργία και Ανάπτυξη των Docker Images

Για την ανάπτυξη των docker images, αρχικά πρέπει να γίνει τροποποίηση στο αρχείο παραμέτρων (application.yaml) κάθε υπηρεσίας για την χρήση του τοπικού host που θα έχουν. Όσο οι υπηρεσίες εκτελούνταν στο τοπικό σύστημα του χρήστη, ο host ήταν το “localhost”, τώρα, με την χρήση των containers θα χρησιμοποιείται διαφορετική διεύθυνση από το docker host περιβάλλον. Αυτή η διεύθυνση του docker host είναι δυνατόν να ληφθεί στο αρχείο παραμέτρων με την ονομασία docker.host.internal.

```
server:
  port: 8082

spring:
  application:
    name: service_name

  data:
    mongodb:
      host: host.docker.internal
      port: 27017
      database: classes
      authentication-database: admin
      username: admin
      password: admin
```

```
rabbitmq:
  stomp:
    port: 61613
  host: host.docker.internal
  username: guest
  password: guest

jwt:
  secret: totallySecretKeyTrustMe

authentication_service_host: http://host.docker.internal:8081

notification_service_host: http://host.docker.internal:8083

remote_origin: http://host.docker.internal:3000
```

Κείμενο 5: Βασική δομή αρχείου παραμετροποίησης (config.yaml) μικρουπηρεσιών με χρήση του Docker

Αφού έχει πραγματοποιηθεί η παραμετροποίηση των αρχείων κάθε μικρουπηρεσίας, μπορεί να γίνει η δημιουργία των images. Σε κάθε φάκελο υπηρεσίας, θα πρέπει να εκτελεστεί η εντολή **“docker build -t {όνομα υπηρεσίας}”**.

Τέλος, για την δημιουργία και έναρξη του κάθε container από το αντίστοιχο image πρέπει να εκτελεστεί η εντολή **“docker run -it –rm -p {docker network port}:{target container port}”**.

Δημιουργία και Ανάπτυξη με το Docker Compose

Για το μαζικό χτίσιμο και εκτέλεση των images είναι δυνατή η χρήση του docker compose [44]. Το docker compose είναι ένα εργαλείο εκτέλεσης συστημάτων πολλαπλών container, και έχει δυνατότητας παραμετροποίησης των υπηρεσιών. Για την ανάπτυξη του συστήματος με χρήση του docker compose χρειάζονται τα εξής:

- Δημιουργία των dockerfile κάθε υπηρεσίας.
- Χτίσιμο του αρχείου docker-compose.yaml, το οποίο περιέχει οδηγίες για όλα τα images.
- Εκτέλεση του αρχείου με χρήση της εντολής **“docker compose up”**.

```
version: '3.9'
services:

 mongodb_service:
   container_name: mongodb_container
   image: mongo
   volumes:
     - ./mongoDB/testapp:/data/db
   ports:
```

```
    - 27017:27017
  networks:
    - internal_net

rabbitmq_service:
  container_name: rabbitmq_container
  build:
    context: ../Notifications-service
    dockerfile: Dockerfile_RabbitMQ
  ports:
    - 5672:5672
    - 15672:15672
    - 61613:61613
  volumes:
    - ./rabbitMQ/.docker-conf/rabbitmq/data/:/var/lib/rabbitmq/
    - ./rabbitMQ/.docker-conf/rabbitmq/log/:/var/log/rabbitmq
  networks:
    - internal_net

authentication_service:
  container_name: authentication_container
  build:
    context: ../Authentication-service
    dockerfile: ../Authentication-service/Dockerfile
  ports:
    - 8081:8081
  depends_on:
    - mongodb_service
  networks:
    - internal_net

classes_service:
  container_name: classes_container
  build:
    context: ../Classes-service
    dockerfile: Dockerfile
  ports:
    - 8082:8082
  depends_on:
    - mongodb_service
    - authentication_service
    - notifications_service
  networks:
    - internal_net

messenger_service:
  container_name: messenger_container
  build:
    context: ../Messenger-service
```

```
    dockerfile: Dockerfile
  ports:
    - 8084:8084
  depends_on:
    - mongodb_service
  networks:
    - internal_net

 notifications_service:
  container_name: notifications_container
  build:
    context: ../Notifications-service
    dockerfile: Dockerfile
  ports:
    - 8083:8083
  depends_on:
    - mongodb_service
    - rabbitmq_service
  networks:
    - internal_net

 webapp_service:
  container_name: webapp_container
  build:
    context: .
    dockerfile: Dockerfile
  ports:
    - 8080:3000
  depends_on:
    - mongodb_service
    - rabbitmq_service
    - authentication_service
    - classes_service
    - messenger_service
    - notifications_service
  networks:
    - internal_net

networks:
 internal_net:
  name: internal_net
  driver: bridge
```

Κείμενο 6: Docker-compose αρχείο (.yaml) των υπηρεσιών του συστήματος

Οι βασικές οδηγίες που δίνονται στο docker-compose αρχείο είναι οι εξής:

- Με την οδηγία **services** καθορίζονται οι υπηρεσίες που θα μετατραπούν σε containers.

- Στην συνέχεια για κάθε υπηρεσία δίνεται το όνομα της (Για συνοχή διατηρείται η μορφοποίηση {όνομα-υπηρεσίας_service}).
- Με την οδηγία .**container_name** δίνεται το όνομα του container που θα δημιουργηθεί για την υπηρεσία.
- Με την οδηγία .**image** δίνεται το image που θα χρησιμοποιηθεί στο container.
- Αντί της οδηγίας image, μπορεί να χρησιμοποιηθεί η οδηγία .**build**, με τις υπο-οδηγίες .**context** και .**docker-file** για τον καθορισμό του μονοπατιού και του ονόματος του docker-file και την δημιουργία του σε image από το οποίο θα δημιουργηθεί το container.
- Στην συνέχεια, με την οδηγία .**ports** καθορίζονται οι θύρες του docker network και οι αντίστοιχες που έχουν οι υπηρεσίες εσωτερικά στο container τους.
- Με την οδηγία .**depends_on** καθορίζονται οι υπηρεσίες που πρέπει να προηγούνται στην εκκίνηση και εκτέλεση για την εύρυθμη λειτουργία της υπηρεσίας.
- Με την οδηγία **services.{service_name}.networks** καθορίζονται τα δίκτυα στα οποία θα είναι συνδεμένες οι υπηρεσίες.
- Με την οδηγία **networks** καθορίζονται τα δίκτυα τα οποία θα υπάρχουν, και οι βασικοί τους παράμετροι είναι το όνομα (**name**) και ο οδηγός (**driver**). Ο driver αποτελεί τον τρόπο με τον οποίο θα συνδέονται οι υπηρεσίες. Βασικός driver αν δεν καθοριστεί κάποιος είναι ο Bridge, ο οποίος χρησιμοποιείται σε ανεξάρτητα containers οποία πρέπει να επικοινωνήσουν.
- Τέλος, με την οδηγία **volumes** καθορίζονται μονοπάτια στα οποία θα αποθηκευτεί πληροφορία η οποία χρειάζεται να διατηρηθεί και μετά τον τερματισμό του container. Στην περίπτωση του συστήματος, τα volumes χρειάζονται στις υποστηρικτικές υπηρεσίες MongoDB και RabbitMQ.

Για την εκτέλεση των containers είναι απαραίτητη όπως και πριν η μετατροπή της **“localhost”** τιμής στα αρχεία παραμετροποίησης κάθε υπηρεσίας σε **“docker.host.internal”**, και η δημιουργία των εκτελέσιμων αρχείων (.jar) κάθε μικρουπηρεσίας με την εντολή **“mvn clean install”**.

Για το μαζικό χτίσιμο όλων των images από τα docker-files τους χρειάζεται η εκτέλεση της εντολής **“docker-compose -f docker-compose.yaml –project-name lab_exchange build”**.

Η δημιουργία και εκτέλεση σε containers γίνεται με την εντολή **“docker-compose -f docker-compose.yaml –project-name lab_exchange up”**, και **“docker-compose -f docker-compose.yaml down”** αντίστοιχα για τον τερματισμό των υπηρεσιών.

Orchestration

Με την ολοκλήρωση της μεταφοράς και ανάπτυξης των υπηρεσιών σε containers είναι πλέον δυνατή η ανάπτυξη και διαχείριση τους σε κάποιο container orchestration system. Τα container orchestration

tools παρέχουν δυνατότητες που έρχονται σε συμφωνία με την 12-factor μεθοδολογία και αξιοποιούν αρκετά τεχνολογίες υπολογιστικής νέφους [20].

Kubernetes

Η τεχνολογία που θα επιλεχθεί για την ανάπτυξη και διαχείριση των containers είναι το **Kubernetes (K8s)** [8]. Το kubernetes είναι μία φορητή και επεκτάσιμη πλατφόρμα ανοιχτού κώδικα για την διαχείριση υπηρεσιών και εφαρμογών, χρησιμοποιώντας δηλωτικούς κανόνες και αυτοματισμούς για την διαμόρφωση τους. Οι λειτουργίες του παρέχονται μέσα από ένα framework με το οποίο δίνεται η δυνατότητα εκτέλεσης ανθεκτικών κατανεμημένων συστημάτων, αναλαμβάνοντας την επεκτασιμότητα, και την ανακατεύθυνση αιτημάτων σε περίπτωση αποτυχίας.

Αρχιτεκτονική και λειτουργία

Η ανάπτυξη συστημάτων στο K8s γίνεται σε συστάδες (clusters). Κάθε cluster αποτελείται από μηχανές εργάτες που καλούνται nodes και μπορούν να είναι είτε φυσικές μηχανές είτε εικονικές. Κάθε node τρέχει μέσα του pods [45], τα οποία αποτελούν την ελάχιστη μονάδα υπολογισμών η οποία μπορεί να αναπτύξει και διαχειριστεί το K8s, και ορίζονται ως ένα αφηρημένο συστατικό που περιέχει containers. Κάθε pod περιέχει ένα ή περισσότερα containers με κοινό χώρο αποθήκευσης και δικτυακών πόρων, μαζί με προδιαγραφές για τον τρόπο εκτέλεσης και διαμόρφωσης τους.

Τα Pods χρησιμοποιούνται από το k8s για να ενθυλακώνουν containers ώστε να μην χρειάζεται η επαφή του χρήστη με κάποιο containerization περιβάλλον, όπως και για να μπορούν να διαχειριστούν τις αλλαγές της κατάστασης τους ή και να τα αντικαταστήσουν με άλλα αν κριθεί απαραίτητο. Κάθε Pod έχει ένα εικονικό δίκτυο με δικιά του διεύθυνση IP, και βάσει αυτή επιτρέπει την επικοινωνία μεταξύ άλλων Pods. Η εικονική διεύθυνση του Pod μαζί με κάποια θύρα συσχετίζονται με εκείνες του container που εκτελεί για την παροχή υπηρεσιών μέσω αυτού. Συνήθως σχεδιαστικά κρίνεται ορθό να υπάρχει ένα container ανά Pod.

Τήρηση της 12-factor μεθοδολογίας

Εξαιτίας της χρήσης containers στα K8s Pods, αλλά και των σχεδιαστικών επιλογών που έγιναν κατά το στάδιο αρχιτεκτονικής και ανάπτυξης της εφαρμογής, πολλά από τα χαρακτηριστικά που ικανοποιούν τους 12 παράγοντες είναι υλοποιημένα. Παρακάτω θα αναλυθούν μόνο τα νέα επιπρόσθετα χαρακτηριστικά που προσφέρει το K8s στην 12-factor μεθοδολογία.

Ο **3ος παράγοντας διαμόρφωσης (configuration) και ο 4ος παράγοντας υποστηρικτικών υπηρεσιών (backing services)** εφαρμόζεται με χρήση μεταβλητών περιβάλλοντος που δίνονται στα deployment αρχεία διαμόρφωσης και έτσι διαμορφώνεται η εφαρμογή χωρίς καμία αλλαγή στον κώδικα, αλλά μόνο στο Pod στο οποίο εκτελείται το container της. Επίσης, υπάρχει η επιλογή δημιουργίας αρχείων διαμόρφωσης **ConfigMap** και **Secret** στα οποία διατηρούνται οι τιμές παραμετροποίησης από το K8s και δίνονται κατάλληλα.

Ο **6ος παράγοντας διεργασιών (Processes)** υλοποιείται από τον σχεδιασμό των υπηρεσιών να είναι stateless και να χρησιμοποιούν backing services για λειτουργίες που δεν ανήκουν στην λογική τους.

Ο **7ος παράγοντας πρόσδεσης θυρών (port binding)** υλοποιείται από τον σχεδιασμό των υπηρεσιών να εκθέτουν τις υπηρεσίες και εφαρμογές τους μέσω θυρών, και επιπροσθέτως από την συσχέτιση των θυρών των containers με εκείνων των Pods και των υπηρεσιών τους.

Ο **8ος παράγοντας παραλληλίας (concurrency)** υλοποιείται μέσα από το kubernetes με χρήση της δυνατότητας κλιμάκωσης των Pods σε αντίγραφα του. Η λειτουργία αυτή ορίζεται μέσα στο αρχείο διαμόρφωσης ανάπτυξης (deployment configuration).

Ο **9ος παράγοντας απορριψιμότητας (disposability)** υλοποιείται από την σχεδιαστική επιλογή των Pods του K8s αλλά και των containers να μπορούν να τερματιστούν και να εκκινηθούν γρήγορα ανάλογα την κατάσταση τους.

Ο **11ος παράγοντας αρχείων συμβάντων (logs)** υλοποιείται αρχικά σε απλή έκδοση μέσα από το τερματικό στο οποίο τρέχει το περιβάλλον του K8s cluster όλα τα Pods, και παρέχονται όλα τα μηνύματα από τα κανάλια εξόδου των containers σε αυτό αθροιστικά. Για την καλύτερη διαχείριση των συμβάντων, υπάρχουν εργαλεία που μέσα από το K8s παρέχουν αυτές τις υπηρεσίες.

Ο **12ος παράγοντας διεργασιών διαχείρισης (admin processes)** υλοποιείται κυρίως ανάλογα το μέγεθος της εφαρμογής. Για μικρής κλίμακας εφαρμογές, με την εντολή **"kubectl exec"** υπάρχει η δυνατότητα εκτέλεσης εντολών μέσα σε ένα container. Επίσης, με το K8s Jobs μπορούν να δημιουργηθούν αυτόνομες εφαρμογές. Για διεργασίες μεγαλύτερης πολυπλοκότητας μπορεί να χρησιμοποιηθεί το **kubernetes helm charts**, το οποίο αναλαμβάνει την διαχείριση περίπλοκων διεργασιών εγκατάστασης, ενημερώσεων, εκδόσεων κ.α.

Δομή αρχείων διαμόρφωσης

Ο τρόπος με τον οποίο δημιουργούνται και διαμορφώνονται τα συστατικά στοιχεία ενός k8s cluster είναι μέσα από τα αρχεία διαμόρφωσης. Το αρχείο διαμόρφωσης ανάλογα τον τύπο του περιέχει τις οδηγίες ανάπτυξης των containers και των υπηρεσιών στους στο Pod, αποθηκευτικούς χώρους, πύλη πρόσβασης στις υπηρεσίες, κ.α. Τα βασικά στοιχεία κάθε αρχείου διαμόρφωσης είναι:

- Στην αρχή κάθε αρχείου διαμόρφωσης υπάρχει το **apiVersion** που ορίζει την έκδοση του api για τον τύπο αρχείου προς δημιουργία, και το **kind**, με το οποίο ορίζεται ο τύπος της διαμόρφωσης που ακολουθεί.
- **Μεταδεδομένα (metadata)**. Πληροφορίες που αφορούν την ίδια την διαμόρφωση, όπως το όνομα της ώστε να μπορεί να βρεθεί από άλλα αρχεία διαμόρφωσης.
- **Προδιαγραφές (specifications)**. Εδώ τοποθετούνται οι οδηγίες που αφορούν την διαμόρφωση του συστατικού στοιχείου σύμφωνα με τον τύπο **(kind)** που δόθηκε.

- **Κατάσταση (status)**. Η κατάσταση δημιουργείται και ενσωματώνεται αυτόματα από το k8s. Το k8s ελέγχει την τρέχουσα και την επιθυμητή κατάσταση του component, και αν διαφέρουν τότε γνωρίζει ότι απαιτείται επιπλέον προσοχή σε αυτό (self-healing ιδιότητα).

Παρακάτω θα διατυπωθούν τα βασικά δομικά στοιχεία κάθε τύπου αρχείου διαμόρφωσης που θα χρειαστεί για την ανάπτυξη του συστήματος.

Deployment Configuration

Με το αρχείο διαμόρφωσης ανάπτυξης (deployment configuration file) ορίζονται οι αναθέσεις των containers σε Pods προς ανάπτυξη, καθώς και άλλες βασικές παραμετροποιήσεις του όπως μεταβλητές περιβάλλοντος, συνδέσεις χώρων αποθήκευσης (pvc) με το container και οι συνδέσεις των Pod στην ανάλογη επιθυμητή υπηρεσία (service).

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb-deployment
  labels:
    app: mongodb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      volumes:
        - name: storage
          persistentVolumeClaim:
            claimName: mongodb-pvc
      containers:
      - name: mongodb
        image: mongo
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 27017
        volumeMounts:
          - mountPath: /data/db
            name: storage
```

Κείμενο 7: Βασική δομή αρχείου Kubernetes deployment configuration (.yaml)

Στο παραπάνω κείμενο (Κείμενο 7) φαίνεται η βασική δομή ενός deployment configuration αρχείου [46]. Σε αυτό οι βασικές οδηγίες που χρησιμοποιούνται είναι οι ακόλουθες:

- Αρχικά ορίζεται η έκδοση του deployment αρχείου με το **.apiVersion** και το είδος της διαμόρφωσης με το **.kind** σε τιμή **deployment**.
- Για την εύρεση των Pod από τις υπηρεσίες τους και από τα deployments είναι απαραίτητο να χρησιμοποιηθούν κατάλληλα οι ετικέτες (labels) και οι selectors. Στα μεταδομένα του deployment δίνεται το όνομα του με το **.metadata.name** και η ετικέτα του με το **metadata.labels.app**. Το label του deployment θα πρέπει να είναι ίδιο με τον αντίστοιχο selector **spec.selector.app** της υπηρεσίας του.
- Στις προδιαγραφές (spec) δίνεται ο αριθμός των αντιγράφων Pods που θα δημιουργηθούν με βάση αυτό το container με το **spec.replicas**.
- Με τον selector **spec.selector.matchLabels.app** δίνεται το όνομα ετικέτας του Pod (ίδια τιμή με το deployment label **.metadata.labels.app**) ώστε να μπορεί το κάθε deployment να εντοπίσει το Pod.
- Το πρότυπο του Pod ορίζεται στο **spec.template**, το οποίο έχει όπως φαίνεται και δικά του metadata και specs. Στα metadata του, το όνομα του καθορίζεται με το **.spec.template.metadata.labels.app** και θα πρέπει να είναι ίδιο με τον αντίστοιχο selector **spec.selector.app** της υπηρεσίας του.
- Το deployment χρειάζεται πρόσβαση σε διατηρητέο χώρο αποθήκευσης, οπότε με το **template.spec.volumes.name** και **template.spec.peristentVolumeClaim.ClaimName** ορίζεται το pvc στο οποίο θα έχει πρόσβαση η υπηρεσία (το pvc component αρχικοποιείται και αναλύεται παρακάτω).
- Με το **.containers** ορίζεται λίστα με το containers προς ανάπτυξη στο Pod.
- Με τα **.containers.name** και **.containers.image** ορίζονται το όνομα και το image όνομα κάθε container. Με την επιλογή **.containers.imagePullPolicy** σε τιμή **IfNotPresent** δίνεται η επιλογή να γίνει προσπάθεια pull του Image από το image registry για την εύρεση του.
- Με την **containers.ports.containerPort** δίνεται η κάθε θύρα στην οποία εκθέτει το container τις υπηρεσίες του.
- Τέλος, με την **containers.volumeMounts**, **.mountPath** και **.name** δίνονται τα μονοπάτια φακέλων στα οποία θα έχει πρόσβαση στο pvc.

Service Configuration

Με το αρχείο διαμόρφωσης υπηρεσιών ορίζονται οι υπηρεσίες κάθε Pod [47]. Για κάθε Pod μία υπηρεσία ορίζει μία βασική διεύθυνση με την οποία μπορεί να αποκτηθεί πρόσβαση στις υπηρεσίες του. Οι υπηρεσίες δεν είναι συνδεδεμένες με τα Pods που τις αφορούν και έτσι ακόμη και ένα Pod

απενεργοποιηθεί ή και αντικατασταθεί, η υπηρεσία του θα παραμείνει. Με αυτό τον τρόπο όταν γίνει επαναλειτουργία του Pod θα έχει ακόμα την ίδια διεύθυνση πρόσβασης.

```
apiVersion: v1
kind: Service
metadata:
  name: notifications-service
spec:
  selector:
    app: notifications
  ports:
    - protocol: TCP
      port: 8083
      targetPort: 8083
  type: NodePort
```

Κείμενο 8: Βασική δομή αρχείου Kubernetes service configuration (.yaml)

Στο παραπάνω κείμενο (Κείμενο 8) φαίνεται η βασική δομή ενός service configuration αρχείου. Σε αυτό οι βασικές οδηγίες που χρησιμοποιούνται είναι οι ακόλουθες:

- Αρχικά ορίζεται η έκδοση του deployment αρχείου με το **.apiVersion** και το είδος της διαμόρφωσης με το **.kind** σε τιμή **Service**.
- Στα metadata της υπηρεσίας δίνεται το όνομα της με το **metadata.name**.
- Στις προδιαγραφές τις υπηρεσίας αρχικά ορίζεται με τον κατάλληλο selector το deployment στο οποίο αναφέρεται με το **spec.selector.app**.
- Στην συνέχεια γίνεται η ανάθεση των θυρών της εφαρμογής από το container (target.port) στο Pod (port) με χρήση της **spec.ports.protocol**, **.port**. και **.target.port**.
- Τέλος, με την **spec.type** σε τιμή **NodePort** δηλώνεται η έκθεση της υπηρεσίας στη διεύθυνση IP του Node στο συγκεκριμένο port.

Volume Claim Configuration

Τα Pods χάνουν τις πληροφορίες που έχουν αποθηκευτεί σε αυτά από τα containers που διατηρούν όταν τεθούν εκτός λειτουργίας. Για τον λόγο αυτό, ο χώρος αποθήκευσης πρέπει να μην εξαρτάτε από τον κύκλο λειτουργίας των Pod, και να είναι διαθέσιμος σε οποιοδήποτε Node διότι δεν είναι σίγουρη η τοποθεσία από την οποία θα γίνει η εκκίνηση του. Για τα προβλήματα αυτά, το K8s παρέχει τα pv - persistent volumes [48]. Τα pv αποτελούν πόρους του cluster όπως η επεξεργαστική ισχύ και η μνήμη και διαμορφώνονται μέσα από .yaml αρχεία όπως τα υπόλοιπα συστατικά. Μέσα από τα αρχεία διαμόρφωσης, μπορεί να οριστεί το pv όχι μόνο στο τοπικό μηχάνημα του χρήστη αλλά και από εξωτερικό πάροχο υπηρεσιών νέφους.

Με το αρχείο διαμόρφωσης volume claim δημιουργείται αίτηση για διατηρητέο χώρο αποθήκευσης pv από τον χρήστη (pvc - persistent volume claim). Όμοια με το Pod το οποίο απαιτεί Nodes, το pvc απαιτεί pv στα οποία αναφέρεται η ποσότητα χώρου, τα δικαιώματα πρόσβασης κ.α. Στο σύστημα lab-exchange, οι υποστηρικτικές υπηρεσίες βάσης δεδομένων και μηνυμάτων είναι που χρειάζονται μόνιμο χώρο αποθήκευσης χωρίς να χάνεται η πληροφορία μετά τον τερματισμό των Pod τους.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: rabbitmq-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Mi
```

Κείμενο 9: Βασική δομή αρχείου Kubernetes volume claim configuration (.yaml)

Στο παραπάνω κείμενο (Κείμενο 9) φαίνεται η βασική δομή ενός pvc configuration αρχείου. Σε αυτό οι βασικές οδηγίες που χρησιμοποιούνται είναι οι ακόλουθες:

- Αρχικά ορίζεται η έκδοση του pvc αρχείου με το **.apiVersion** και το είδος της διαμόρφωσης με το **.kind** σε τιμή **PersistentVolumeClaim**.
- Στα metadata του pvc δίνεται το όνομα του με το **metadata.name**.
- Στην συνέχεια δίνονται τα δικαιώματα πρόσβασης στον αποθηκευτικό χώρο με το **spec.accessModels** σε **ReadWriteOnce**, που σημαίνει ότι το volume μπορεί να το διαχειρίζεται ένα Node.
- Τέλος, γίνεται η αίτηση για pv και δίνεται η ποσότητα που απαιτείται με το **resources.requests.storage** για 10Mb. Το pv από το οποίο παρέχεται ο χώρος αποθήκευσης είναι το ίδιο το μηχάνημα της δοκιμαστικής τοπικής λειτουργίας. Σε περιβάλλον παραγωγής θα έπρεπε να οριστεί και το pv το οποίο μπορεί να παρέχεται από κάποιον πάροχο υπηρεσιών νέφους.

Πύλη Εφαρμογής (Ingress Configuration)

Με την διαμόρφωση των υπηρεσιών ως τώρα, φαίνεται πως για την πρόσβαση σε κάποια υπηρεσία της εφαρμογής θα πρέπει να χρησιμοποιηθεί η διεύθυνση IP του k8s Node και το port της υπηρεσίας, αφού έχει προηγηθεί η δημιουργία εξωτερικών υπηρεσιών (external services) που εκτίθενται στο δίκτυο και δεν είναι στο εσωτερικό δίκτυο του cluster. Αυτό σε τοπικό περιβάλλον δοκιμών ίσως έχει νόημα αλλά σε μία εφαρμογή σε επίπεδο παραγωγής είναι προβληματικό. Λύση σε αυτό το πρόβλημα προσφέρει το k8s Ingress [49].

Το Ingress αποτελεί μία υπηρεσία η οποία διαχειρίζεται την εξωτερική πρόσβαση στις υπηρεσίες ενός cluster, και μπορεί να προσφέρει load balancing, SSL, και εικονική ονομασία εξυπηρετητή. Με το Ingress γίνεται με χρήση κανόνων αντιστοίχιση ονομάτων url με επιθυμητές εσωτερικές υπηρεσίες. Με την χρήση του Ingress, εξωτερικά φαίνεται ένα κεντρικό domain όνομα για τον host και με το οποίο με διαφορετικά path γίνεται η πρόσβαση στις διαφορετικές υπηρεσίες, δίνοντας την εντύπωση πως η εφαρμογή είναι μία ενιαία οντότητα και όχι ένα σύνολο ανεξάρτητων υπηρεσιών σε Nodes.

Για την χρήση του Ingress απαιτείται η εγκατάσταση μίας υλοποίησης του που καλείται Ingress Controller. Ο Ingress Controller στην συνέχεια θα διαμορφωθεί σύμφωνα με το Ingress Configuration αρχείο για την λειτουργία του. Υλοποιήσεις του Ingress Controller παρέχονται και από τρίτες εταιρείες, αλλά, για την τοπική ανάπτυξη του συστήματος, θα χρησιμοποιηθεί η υλοποίηση που παρέχεται από το ίδιο το K8s και λέγεται K8s Ngnix Ingress Controller.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: default-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$1
    kubernetes.io/ingress.class: nginx
    #nginx.org/ssl-services: "my-svc" # fake kubernetes certificate
spec:
  rules:
  - host: lab-exchange.com
    http:
      paths:
      - path: /(.*)
        pathType: Prefix
        backend:
          service:
            name: webapp-service
            port:
              number: 8080
      - path: /api/authentication-service/(.*)
        pathType: Prefix
        backend:
          service:
            name: authentication-service
            port:
              number: 8081
```

Κείμενο 10: Βασική δομή αρχείου Kubernetes ingress configuration (.yaml)

Στο παραπάνω κείμενο (Κείμενο 10) φαίνεται η βασική δομή ενός Ingress configuration αρχείου. Σε αυτό οι βασικές οδηγίες που χρησιμοποιούνται είναι οι ακόλουθες:

- Αρχικά ορίζεται η έκδοση του pvc αρχείου με το **.apiVersion** και το είδος της διαμόρφωσης με το **.kind** σε τιμή **Ingress**.
- Στην συνέχεια χρησιμοποιούνται κάποια σχόλια (annotations) από τον Ingress controller. Με το **metadata.annotations.nginx.ingress.kubernetes.io/rewrite-target** ορίζεται ένας κανόνας κατά τον οποίο οποιαδήποτε είσοδος μονοπατιού δοθεί από τον χρήστη, ο κανόνας θα ψάξει να αναγνωρίσει το **(.*)**, και θα το εκχωρήσει στο **$1**. Στην συνέχεια, θα ξαναγραφτεί το μονοπάτι εισόδου και θα περιλαμβάνει μόνο το σημείο που καταχωρήθηκε στο **$1** και όχι όλο το path που δόθηκε από την **spec.rules.http.paths.path**. Έτσι αν ο χρήστης έδωσε **“host/api/authentication-service/fetch/all”** το Ingress θα ψάξει την υπηρεσία του Pod για **“host/fetch/all”**. Αυτό γίνεται για ευκολία στην δομή και ανάγνωση του api σε εξωτερικά δίκτυα.
- Με το **spec.rules.host** δίνεται το όνομα του εξυπηρετητή στον οποίο παρέχεται το K8s cluster.
- Μέσα στο **spec.rules.http.paths** ορίζονται τα μονοπάτια που θα αναγνωρίζονται στο εξωτερικό δίκτυο με χρήση κανόνων, και ορίζεται ο εντοπισμός του επιθυμητού Pod στο οποίο αναφέρονται.
- Με το **spec.rules.http.paths.path**, **.pathType**, ορίζεται ο κανόνας αναγνώρισης του μονοπατιού και αντίστοιχα το είδος της αναγνώρισης, το οποίο με τιμή **Prefix** σημαίνει ότι θα αναγνωρίζει το πρόθεμα του μονοπατιού που δίνεται.
- Τέλος, με το **spec.rules.http.paths.backend.service** δίνονται τα **.port** και **.name** για την σύνδεση του μονοπατιού με την επιθυμητή υπηρεσία του Pod.

Δημιουργία του Kubernetes Configuration Αρχείου

Με βάση τα παραπάνω αρχεία διαμόρφωσης, δημιουργήθηκε το συνολικό αρχείο διαμόρφωσης για την ανάπτυξη των υπηρεσιών στο Kubernetes.

```
# ================== Ingress ====================

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: default-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$1
    kubernetes.io/ingress.class: nginx
    #nginx.org/ssl-services: "my-svc" # fake kubernetes certificate
spec:
  rules:
  - host: lab-exchange.com
    http:
      paths:
```

```
    - path: /(.*)
      pathType: Prefix
      backend:
        service:
          name: webapp-service
          port:
            number: 8080
    - path: /api/authentication-service/(.*)
      pathType: Prefix
      backend:
        service:
          name: authentication-service
          port:
            number: 8081
    - path: /api/classes-service/(.*)
      pathType: Prefix
      backend:
        service:
          name: classes-service
          port:
            number: 8082
    - path: /api/notifications-service/(.*)
      pathType: Prefix
      backend:
        service:
          name: notifications-service
          port:
            number: 8083
    - path: /api/messenger-service/(.*)
      pathType: Prefix
      backend:
        service:
          name: messenger-service
          port:
            number: 8084

---

# ================== MongoDB ====================

# K8 Service Config | MongoDB Service

apiVersion: v1
kind: Service
metadata:
  name: mongodb-service
  labels:
    app: mongodb
spec:
```

```
  selector:
    app: mongodb
  ports:
    - port: 27017
      targetPort: 27017
  type: NodePort

---

# K8 Persistent Volume Claim Config | MongoDB Service

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Mi

---

# K8 Deployment Config | MongoDB Service

apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb-deployment
  labels:
    app: mongodb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      volumes:
        - name: storage
          persistentVolumeClaim:
            claimName: mongodb-pvc
      containers:
      - name: mongodb
        image: mongo
```

```
      imagePullPolicy: IfNotPresent
      ports:
        - containerPort: 27017
      volumeMounts:
        - mountPath: /data/db
          name: storage

---

# ================== RabbitMQ ====================

# K8 Service Config | RabbitMQ Service

apiVersion: v1
kind: Service
metadata:
  name: rabbitmq-service
spec:
  selector:
    app: rabbitmq
  ports:
    - protocol: TCP
      name: rabbitmq-p0
      port: 5672
      targetPort: 5672
    - protocol: TCP
      name: rabbitmq-p1
      port: 15672
      targetPort: 15672
    - protocol: TCP
      name: rabbitmq-p2
      port: 61613
      targetPort: 61613
  type: NodePort

---

# K8 Persistent Volume Claim 1 Config | RabbitMQ Service

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: rabbitmq-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Mi
```

```
---

# K8 Deployment Config | RabbitMQ Service

apiVersion: apps/v1
kind: Deployment
metadata:
  name: rabbitmq-deployment
  labels:
    app: rabbitmq
spec:
  replicas: 1
  selector:
    matchLabels:
      app: rabbitmq
  template:
    metadata:
      labels:
        app: rabbitmq
    spec:
      containers:
      - name: rabbitmq
        image: lab_exchange_rabbitmq_service
        imagePullPolicy: IfNotPresent
        volumeMounts:
          - mountPath: "/var/lib/rabbitmq/"
            name: storage
          - mountPath: "/var/log/rabbitmq"
            name: storage
        ports:
        - containerPort: 5672
        - containerPort: 15672
        - containerPort: 61613
      volumes:
        - name: storage
          persistentVolumeClaim:
            claimName: rabbitmq-pvc

---

# ================== Authentication ====================

# K8 Service Config | Authentication Service

apiVersion: v1
kind: Service
metadata:
  name: authentication-service
```

```
spec:
  selector:
    app: authentication
  ports:
    - protocol: TCP
      port: 8081
      targetPort: 8081
  type: NodePort

---

# K8 Deployment Config | Authentication Service

apiVersion: apps/v1
kind: Deployment
metadata:
  name: authentication-deployment
  labels:
    app: authentication
spec:
  replicas: 1
  selector:
    matchLabels:
      app: authentication
  template:
    metadata:
      labels:
        app: authentication
    spec:
      containers:
      - name: authentication
        image: lab_exchange_authentication_service
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 8081
        env:
          - name: MONGO_HOST
            value: mongodb-service

---

# ================== Classes ====================

# K8 Service Config | Classes Service

apiVersion: v1
kind: Service
metadata:
  name: classes-service
```

```
spec:
  selector:
    app: classes
  ports:
    - protocol: TCP
      port: 8082
      targetPort: 8082
  type: NodePort

---

# K8 Deployment Config | Classes Service

apiVersion: apps/v1
kind: Deployment
metadata:
  name: classes-deployment
  labels:
    app: classes
spec:
  replicas: 1
  selector:
    matchLabels:
      app: classes
  template:
    metadata:
      labels:
        app: classes
    spec:
      containers:
      - name: classes
        image: lab_exchange_classes_service
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 8082
        env:
          - name: MONGO_HOST
            value: mongodb-service
          - name: AUTHENTICATION_HOST
            value: authentication-service
          - name: NOTIFICATIONS_HOST
            value: notifications-service

---

# ==================== Messenger ====================

# K8 Service Config | Messenger Service
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: messenger-service
spec:
  selector:
    app: messenger
  ports:
    - protocol: TCP
      port: 8084
      targetPort: 8084
  type: NodePort

---

# K8 Deployment Config | Messenger Service

apiVersion: apps/v1
kind: Deployment
metadata:
  name: messenger-deployment
  labels:
    app: messenger
spec:
  replicas: 1
  selector:
    matchLabels:
      app: messenger
  template:
    metadata:
      labels:
        app: messenger
    spec:
      containers:
      - name: messenger
        image: lab_exchange_messenger_service
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 8084
        env:
          - name: MONGO_HOST
            value: mongodb-service
          - name: NOTIFICATIONS_HOST
            value: notifications-service
          - name: CLASSES_HOST
            value: classes-service

---
```

```
# ================== Notifications ====================

# K8 Service Config | Notifications Service

apiVersion: v1
kind: Service
metadata:
  name: notifications-service
spec:
  selector:
    app: notifications
  ports:
    - protocol: TCP
      port: 8083
      targetPort: 8083
  type: NodePort

---

# K8 Deployment Config | Notifications Service

apiVersion: apps/v1
kind: Deployment
metadata:
  name: notifications-deployment
  labels:
    app: notifications
spec:
  replicas: 1
  selector:
    matchLabels:
      app: notifications
  template:
    metadata:
      labels:
        app: notifications
    spec:
      containers:
      - name: notifications
        image: lab_exchange_notifications_service
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 8083
        env:
          - name: MONGO_HOST
            value: mongodb-service
          - name: RABBITMQ_HOST
            value: rabbitmq-service
```

```
---

# ================== Web Application ====================

# K8 Service Config | WebApp Service

apiVersion: v1
kind: Service
metadata:
  name: webapp-service
spec:
  selector:
    app: webapp
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 3000
  type: NodePort

---

# K8 Deployment Config | WebApp Service

apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp-deployment
  labels:
    app: webapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: webapp
        image: lab_exchange_webapp_service
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 3000

---
```

Κείμενο 11: Kubernetes configuration αρχείο (.yaml)

Μετατροπές Αρχείων Διαμόρφωσης Υπηρεσιών

Με την χρήση των deployment αρχείων διαμόρφωσης είναι δυνατή η προσθήκη μεταβλητών περιβάλλοντος οι οποίες θα προσδένονται στα container που εκτελούνται στα Pods. Για αυτό τον λόγο τα αρχεία διαμόρφωσης (application.yaml) στον κώδικα των μικρουπηρεσιών μπορούν να χρησιμοποιούν πλέον τιμές που διαβάζουν από μεταβλητές περιβάλλοντος και όχι στατικές.

```
server:
  port: 8082

spring:
  application:
    name: service_name

  data:
    mongodb:
      host: {MONGO_HOST:localhost}
      port: 27017
      database: classes
      authentication-database: admin
      username: admin
      password: admin

rabbitmq:
  stomp:
    port: 61613
  host: {RABBITMQ_HOST:localhost}
  username: guest
  password: guest

jwt:
  secret: totallySecretKeyTrustMe

authentication_service_host: {AUTHENTICATION_HOST:localhost}

notification_service_host: {NOTIFICATION_HOST:localhost}

remote_origin: http://lab-exchange.com
```

Κείμενο 12: Βασική δομή αρχείου παραμετροποίησης (config.yaml) μικρουπηρεσιών με χρήση του Kubernetes

Επομένως, πλέον η διεύθυνση host στον οποίο υπάρχει κάθε υπηρεσία δίνεται με το όνομα της μεταβλητής περιβάλλοντος όπως της παρέχεται από το deployment file.

Ανάπτυξη και Εκτέλεση σε Kubernetes

Αφού πραγματοποιήθηκε η δημιουργία του αρχείου διαμόρφωσης του Kubernetes, και τα αρχεία διαμόρφωσης των μικρουπηρεσιών έχουν την κατάλληλη μορφή χρησιμοποιώντας μεταβλητές περιβάλλοντος, μπορεί να προχωρήσει η διαδικασία ανάπτυξης του συστήματος.

Αρχικά, απαιτείται η δημιουργία των εκτελέσιμων αρχείων (.jar) κάθε μικρουπηρεσίας με την εντολή **“mvn clean install”**.

Στην συνέχεια θα πραγματοποιηθεί το μαζικό χτίσιμο των images με χρήση του docker-compose ώστε να χρησιμοποιηθούν τα images από το K8s. Αυτό γίνεται με χρήση της εντολής **“docker-compose -f docker-compose.yaml –project-name lab_exchange build”**.

Τέλος, για την δημιουργία και ανάπτυξη των components στο K8s πρέπει να γίνει χρήση της εντολής **“kubectl apply -f kubernetesConfig.yaml”**.

Τροποποιήσεις για Ανάπτυξη σε Περιβάλλον Παραγωγής

Σε αυτή την ενότητα θα πραγματοποιηθεί η επιλογή τεχνολογιών νέφους που θα χρησιμοποιηθούν για την υποστήριξη της εφαρμογής lab exchange. Οι τεχνολογίες που θα χρησιμοποιηθούν περιλαμβάνονται σε αυτές που μελετήθηκαν στο 2ο κεφάλαιο και συνολικά είναι οι εξής:

- Η υποστήριξη του RabbitMQ message broker θα γίνει με χρήση VM στιγμιοτύπου της Google Cloud καθώς δεν υπάρχει δωρεάν λύση για την λειτουργία του με υποστήριξη του πρωτοκόλλου STOMP.
- Η υποστήριξη της MongoDB θα γίνει με χρήση της υπηρεσίας CosmosDB της Microsoft Azure.
- Η υποστήριξη και φιλοξενία των microservices θα γίνει με χρήση της υπηρεσίας Azure App Service της Microsoft Azure.

Παρακάτω ακολουθεί ανάλυση των βημάτων εγκατάστασης.

Υποστήριξη Μικρουπηρεσιών

Από την μελέτη σε υπηρεσίες νέφους έγινε σαφές ότι δεν υπάρχουν δωρεάν λύσεις που να υποστηρίζουν ανάπτυξη του συστήματος σε Kubernetes. Επομένως, για την ανάπτυξη του θα πρέπει να χρησιμοποιηθούν εναλλακτικές λύσεις.

Για την υποστήριξη και φιλοξενία των μικρουπηρεσιών θα χρησιμοποιηθεί η υπηρεσία Azure App Service της Microsoft Azure διότι αποτελεί την μοναδική stateful υπηρεσία ενώ, παράλληλα διατηρεί σχετικά όμοια όρια χρήσης με τους ανταγωνιστές της. Η κάθε μικρουπηρεσία θα αναπτυχθεί ανεξάρτητα από τις υπόλοιπες στο Azure App Service χωρίς να υπάρχει δυνατότητα ενορχήστρωσης.

Για την ανάπτυξη των υπηρεσιών στο App Service απαιτείται η χρήση ενός image registry ώστε να γίνει η λήψη τους από την υπηρεσία. Για τον σκοπό αυτό χρησιμοποιήθηκε το Docker Hub. Εφόσον έχει δημιουργηθεί λογαριασμός στο DockerHub, με την εντολή "docker push {repo_name}" αποθηκεύεται το κάθε image των μικρουπηρεσιών στο registry.

Στο branch "master" των repos είναι διαθέσιμος ο κώδικας για την ανάπτυξη στο τοπικό περιβάλλον με K8s, Docker ενώ, στο branch "deployed_without_k8s" υπάρχει η έκδοση ανάπτυξης με χρήση υπηρεσιών νέφους. Το URL για κάθε repo αναφέρεται παρακάτω:

- Web Application: https://github.com/JohnDelta/LabExchange_WebApplication.git
- Authentication service: https://github.com/JohnDelta/Authentication-Service.git
- Classes service: https://github.com/JohnDelta/Classes-Service.git
- Messenger service: https://github.com/JohnDelta/Messenger-service.git
- Notifications service: https://github.com/JohnDelta/Notification-service.git

Ενώ, η ονομασία των docker hub images είναι:

- Web Application: johndelta/lab-exchange-webapp
- Authentication service: johndelta/lab-exchange-authentication
- Classes service: johndelta/lab-exchange-classes
- Messenger service: johndelta/lab-exchange-messenger
- Notifications service: johndelta/lab-exchange-notifications

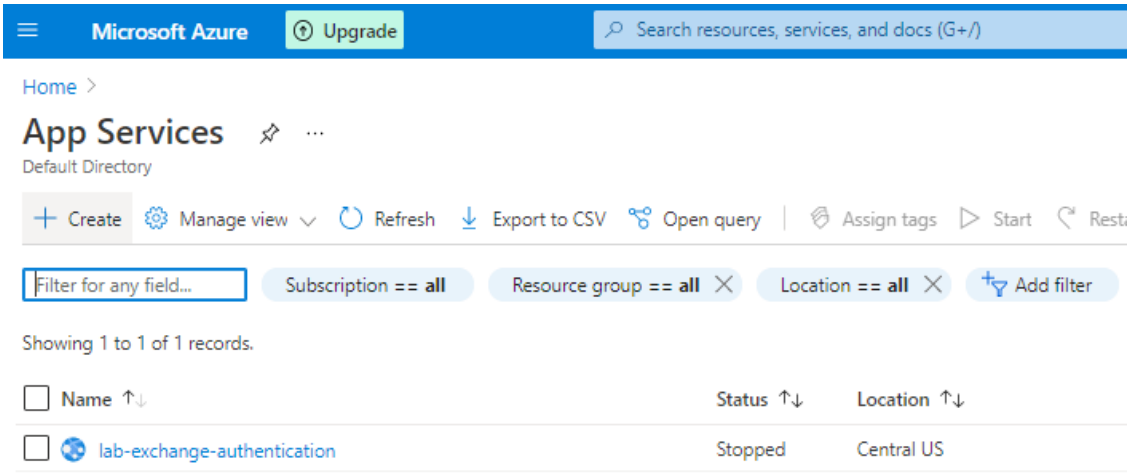
Ένα επίσης σημαντικό κομμάτι που πρέπει να αναφερθεί, είναι ότι με την χρήση των docker images η υπηρεσία App Service θα χτίσει τα containers και θα τα εκτελεί. Όμως, τα docker-files που παράγουν τα images (βλ. Κείμενο 3) έχουν χτιστεί με την λογική ότι πρέπει πρώτα να γίνει χειροκίνητα το χτίσιμο της μικρουπηρεσίας στο .jar εκτελέσιμο της. Για την επίλυση αυτού του προβλήματος απαιτούνται αλλαγές στα docker-files των μικρουπηρεσιών ώστε να ενσωματώνουν την λειτουργία του χτισίματος σε .jar πριν την εκτέλεση του ώστε να είναι πλήρως ανεξάρτητα.

```
FROM openjdk:15-jdk-alpine as build
WORKDIR /workspace/app

COPY mvnw .
COPY .mvn .mvn
COPY pom.xml .
COPY src src

RUN chmod +x mvnw
```

```
RUN ./mvnw install -DskipTests
RUN mkdir -p target/dependency && (cd target/dependency; jar -xf ../*.jar)

FROM openjdk:15-jdk-alpine

RUN addgroup -S spring1 && adduser -S spring1 -G spring1
USER spring1:spring1

VOLUME /tmp
ARG DEPENDENCY=/workspace/app/target/dependency
COPY --from=build ${DEPENDENCY}/BOOT-INF/lib /app/lib
COPY --from=build ${DEPENDENCY}/META-INF /app/META-INF
COPY --from=build ${DEPENDENCY}/BOOT-INF/classes /app
COPY --from=build /workspace/app/target/*.jar notifications.jar

ENV PORT 8000
EXPOSE 8000

ENTRYPOINT ["java", "-Dserver.port=8000", "-jar", "/notifications.jar"]
```

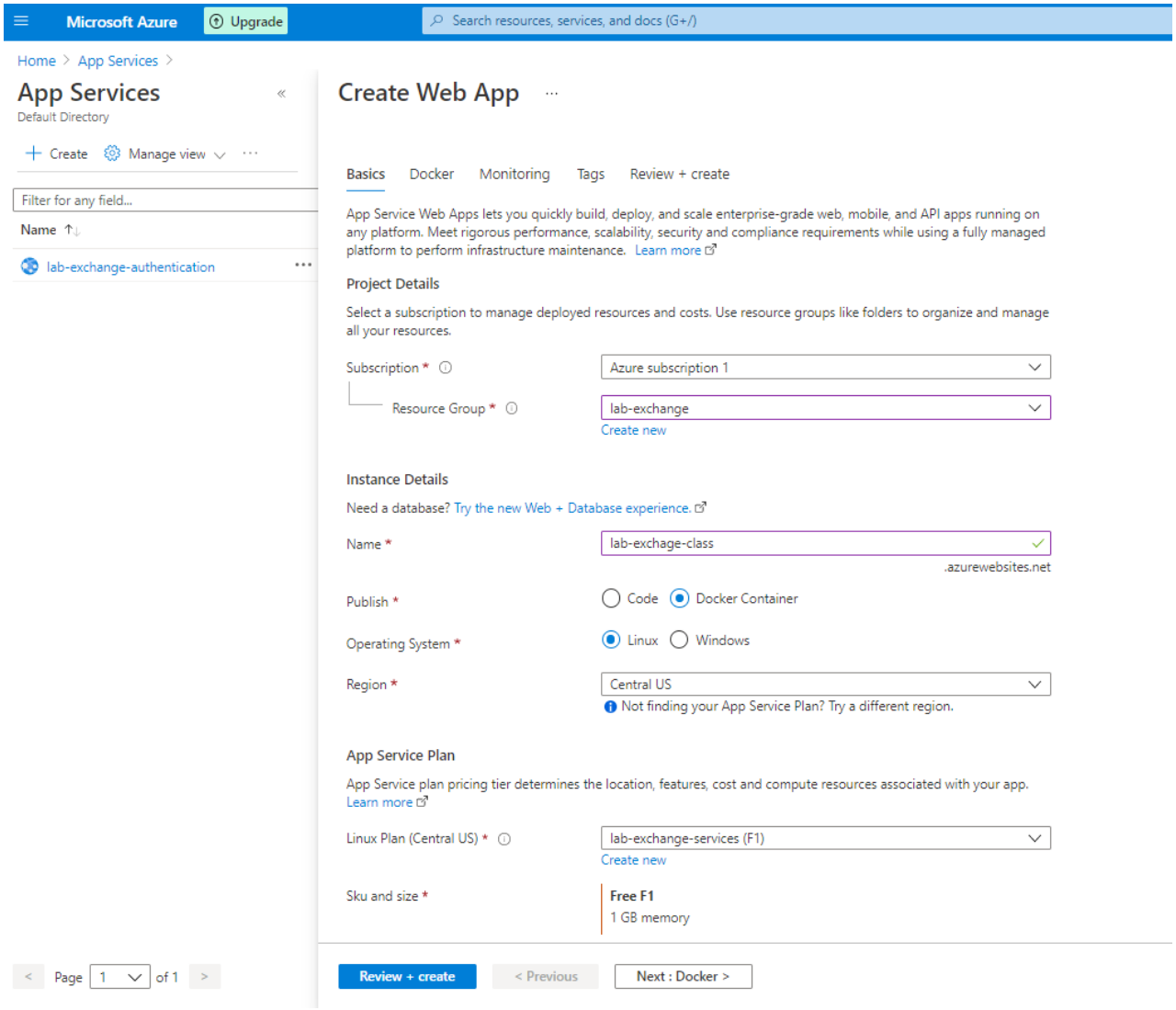
Κείμενο 13: Dockerfile (.yaml) υπηρεσίας ειδοποιήσεων με ενσωματωμένη λειτουργία ανάπτυξης εκτελέσιμου αρχείου κώδικα

Εφόσον τα images των υπηρεσιών είναι έτοιμα, και αφού έχει πραγματοποιηθεί η δημιουργία του Free Tier λογαριασμού στην Microsoft Azure, πρέπει να γίνει επιλογή της “App Services” υπηρεσίας.



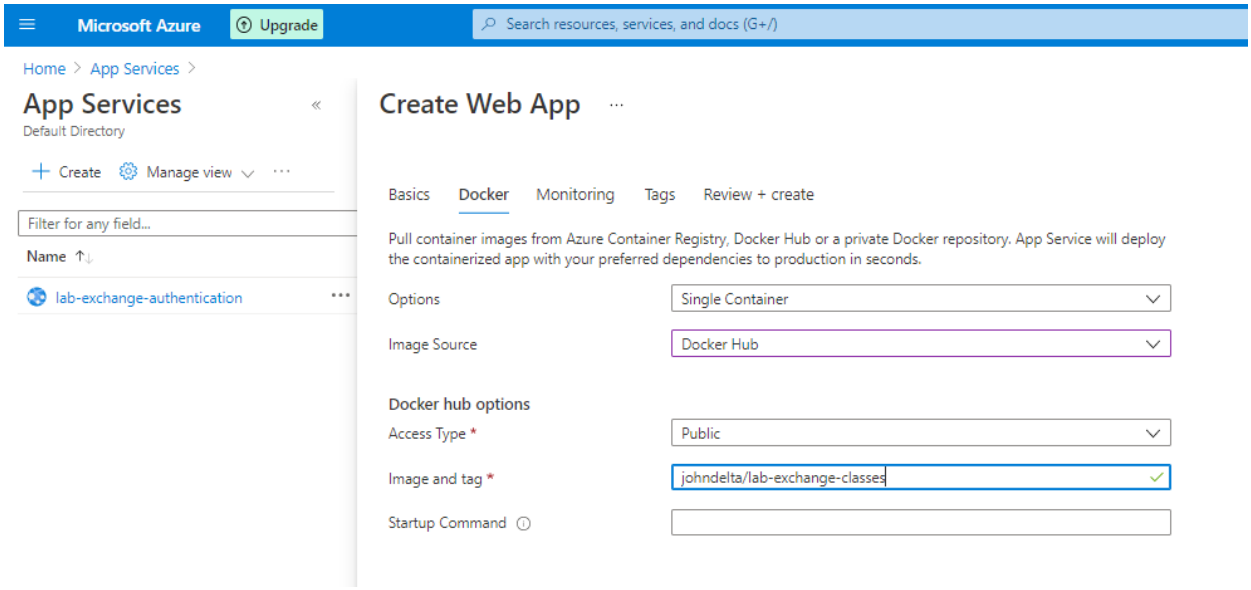
Σχέδιο 29: Επιλογή Τεχνολογιών | Microservices – Καρτέλα λίστας υπηρεσίων

Στην συνέχεια, με τη επιλογή “Create” γίνεται μεταφορά στην σελίδα δημιουργίας υπηρεσίας προς ανάπτυξη. Στην σελίδα αυτή δηλώνονται οι παράμετροι τις υπηρεσίας. Εδώ απαιτείται προσοχή στην επιλογή του “App Service Plan” που θα χρησιμοποιηθεί, θα πρέπει να επιλεγούν τα F1 instances διότι αυτά παρέχονται δωρεάν στο Free Tier. Επιπροσθέτως, στην επιλογή “Publish” δίνεται η επιλογή “Docker Container” για να δοθεί η δυνατότητα ανάπτυξης με docker image.



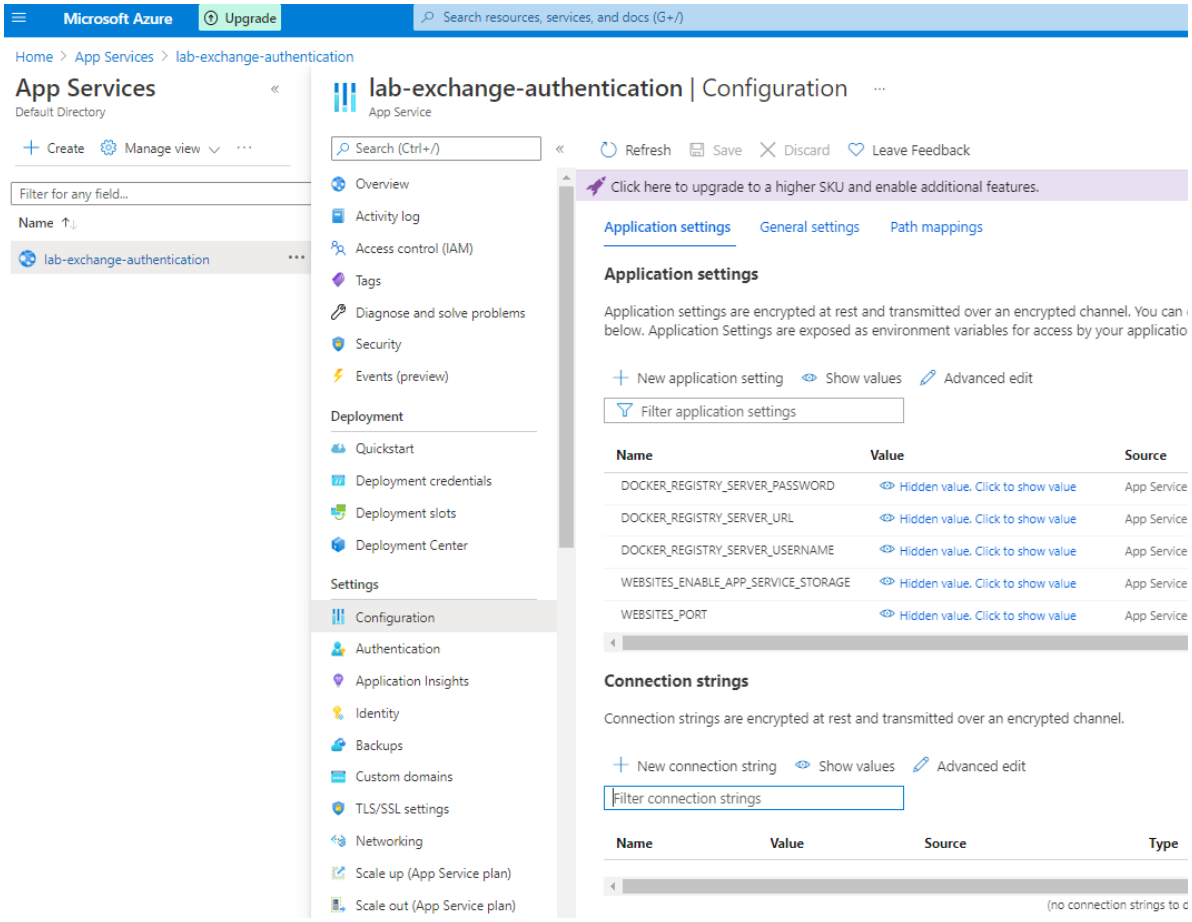
Σχέδιο 30: Επιλογή Τεχνολογιών | Microservices – Καρτέλα εισόδου παραμέτρων της προς δημιουργία υπηρεσίας

Στο δεύτερο σκέλος της δημιουργίας, πραγματοποιείται η επιλογή του docker registry και του image URI που θα χρησιμοποιηθεί για την ανάπτυξη της υπηρεσίας. Αφού η επιλογή πραγματοποιηθεί η υπηρεσία μπορεί να δημιουργηθεί.



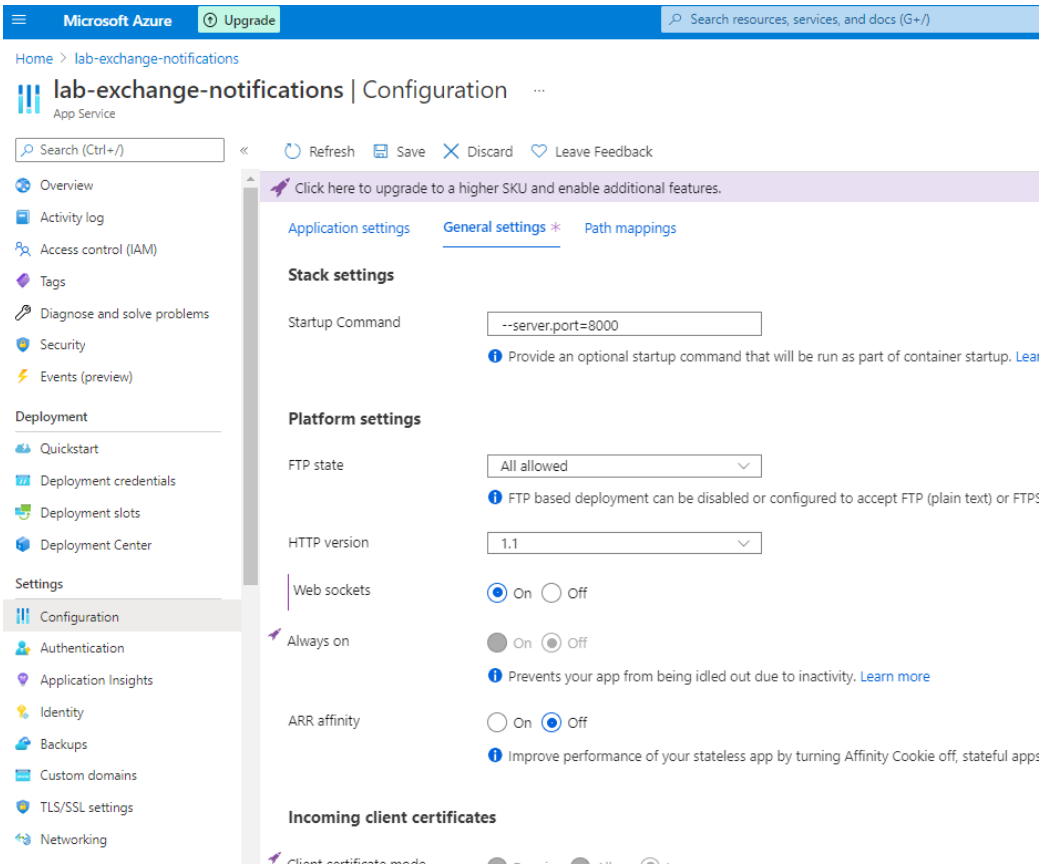
Σχέδιο 31: Επιλογή Τεχνολογιών | Microservices – Καρτέλα επιλογής Image Registry υπηρεσίας

Ένα σημείο που χρειάζεται επίσης προσοχή είναι να δοθεί η παράμετρος “WEBSITES_PORT” στην θύρα 8000 για την έκδοση της υπηρεσίας στο διαδίκτυο. Η υπηρεσίες στο App Services παρέχονται μέσα από τις θύρες 8080 και 8000 επομένως πρέπει να γίνει πρόσδεση τους με αυτές του container.



Σχέδιο 32: Επιλογή Τεχνολογιών | Microservices – Ανάθεση μεταβλητών περιβάλλοντος

Για την υπηρεσία ειδοποιήσεων, θα πρέπει επίσης να ενεργοποιηθούν τα web sockets όπως φαίνεται στην εικόνα 33, ώστε να μπορεί να συνδεθεί στον message broker.



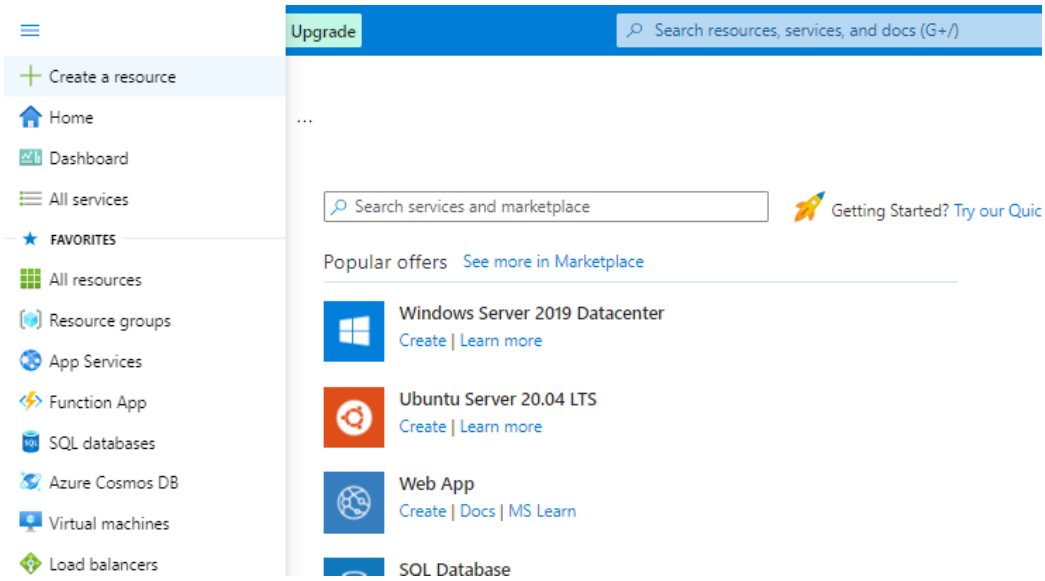
Σχέδιο 33: Επιλογή Τεχνολογιών | Microservices – Ενεργοποίηση Web socket χρήσης

Τέλος, θα πρέπει να ρυθμιστεί το CORS δίνοντας πρόσβαση στην χρήση των μικρουπηρεσιών μόνο από URLs που ανήκουν μεταξύ του συστήματος και όχι από τρίτους. Στην εικόνα 34 ακολουθεί η αλλαγή που πρέπει να γίνει για την κάθε υπηρεσία στο App Service.

Υπηρεσία Υποστήριξης MongoDB

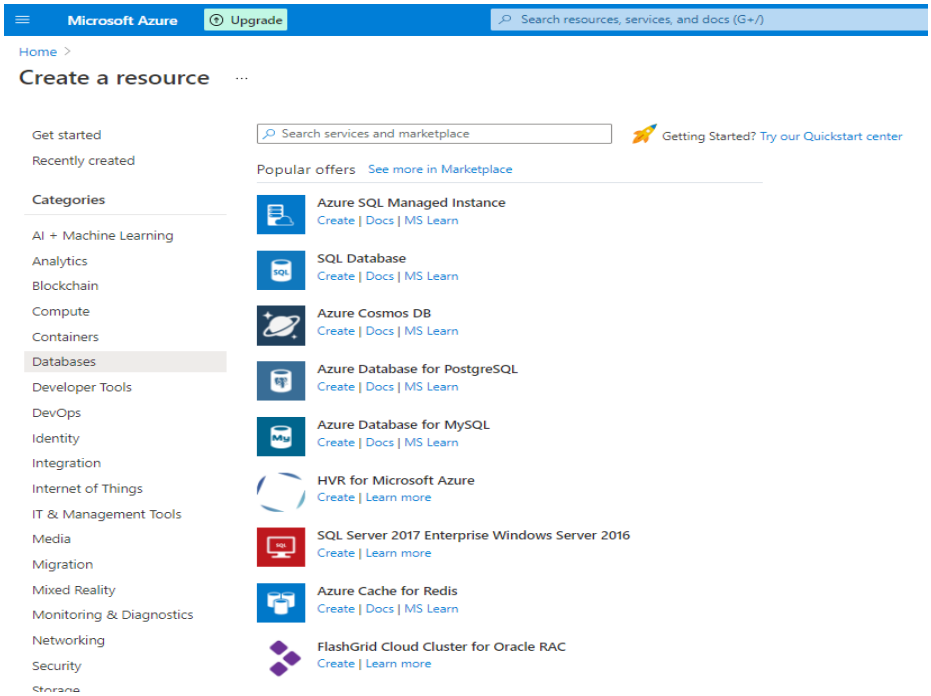
Η μοναδική υπηρεσία παρόχου που μελετήθηκε και προσφέρει δωρεάν MongoDB βάση δεδομένων ως υπηρεσία προς χρήση είναι η CosmosDB της Microsoft Azure.

Εφόσον έχει δημιουργηθεί λογαριασμός Free Tier στην Microsoft Azure, πρέπει να πατηθεί η επιλογή “Create a resource”.



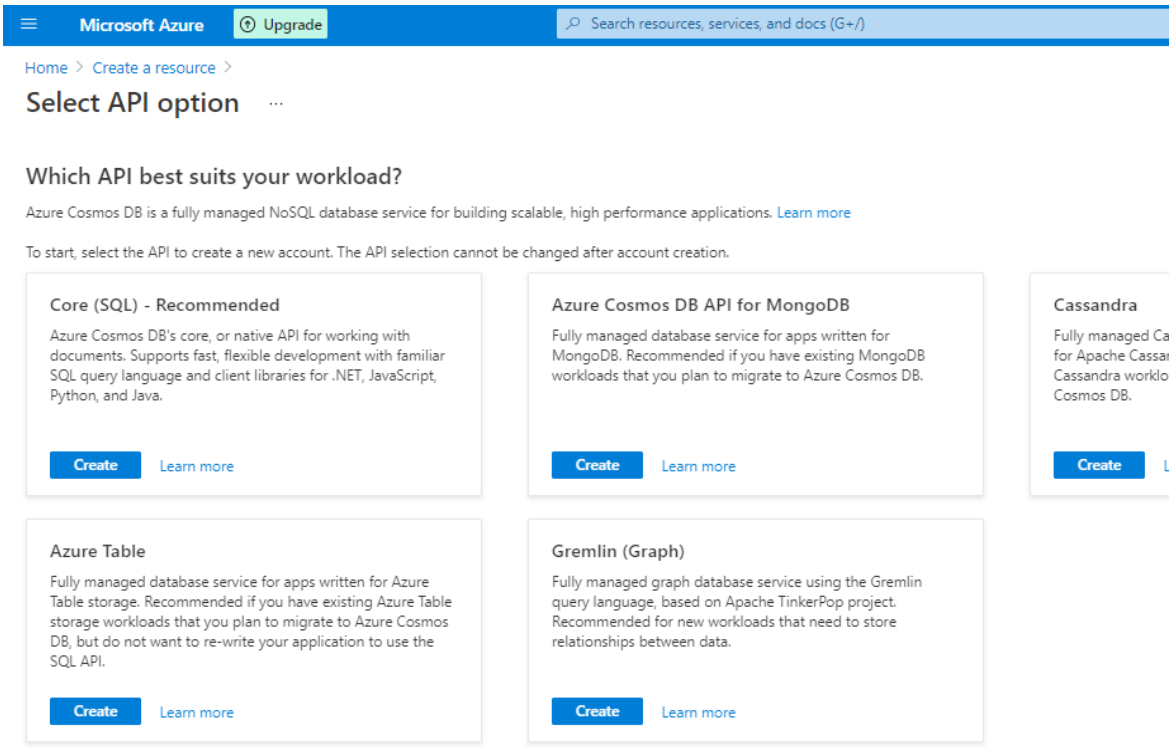
Σχέδιο 34: Επιλογή Τεχνολογιών | MongoDB – Επιλογή δημιουργίας νέας υπηρεσίας

Στην συνέχεια, επιλέγεται η κατηγορία “Databases”, και στην συνέχεια η “Azure Cosmos DB”.



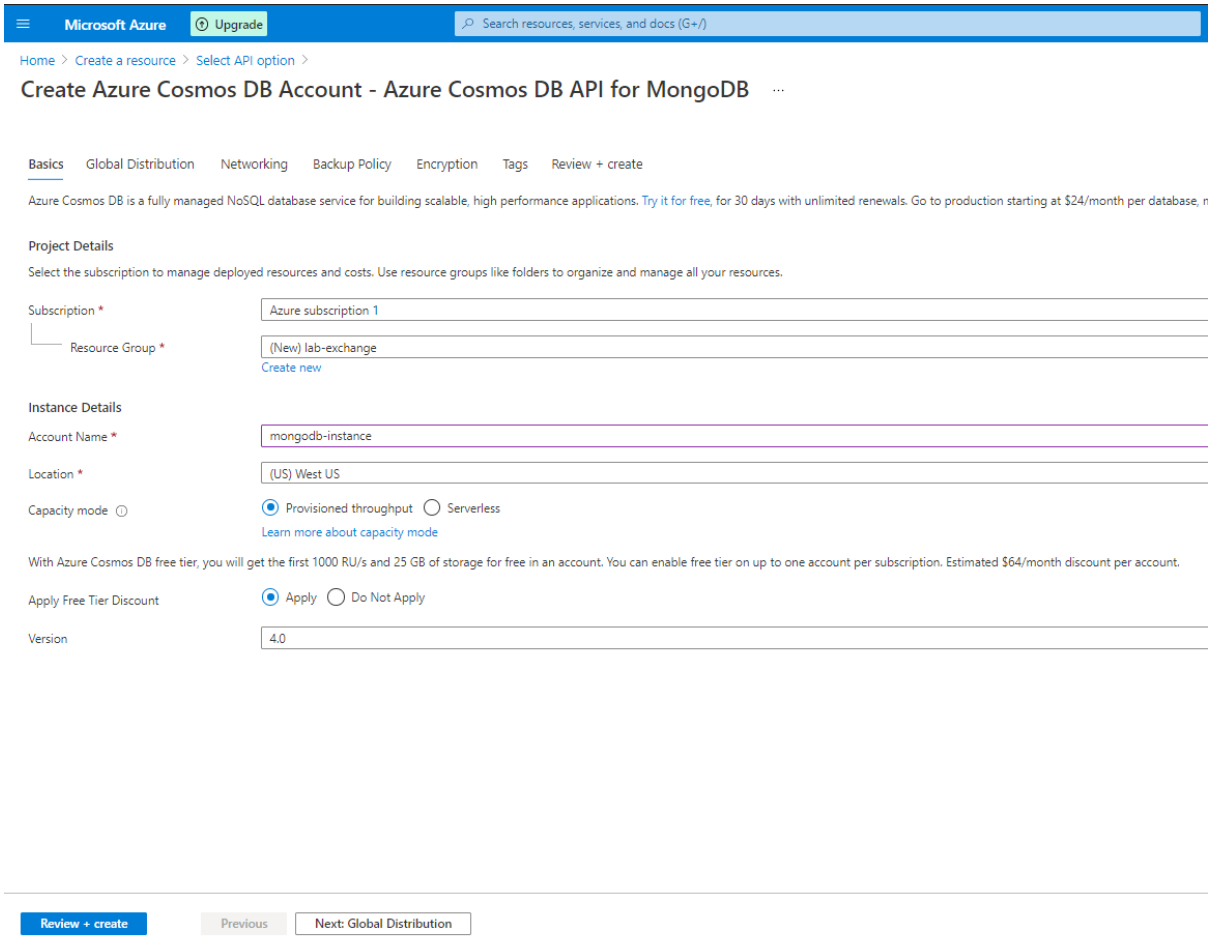
Σχέδιο 35: Επιλογή Τεχνολογιών | MongoDB – Καρτέλα επιλογής υπηρεσιών

Εδώ γίνεται επιλογή της “Azure Cosmos DB API for MongoDB”.



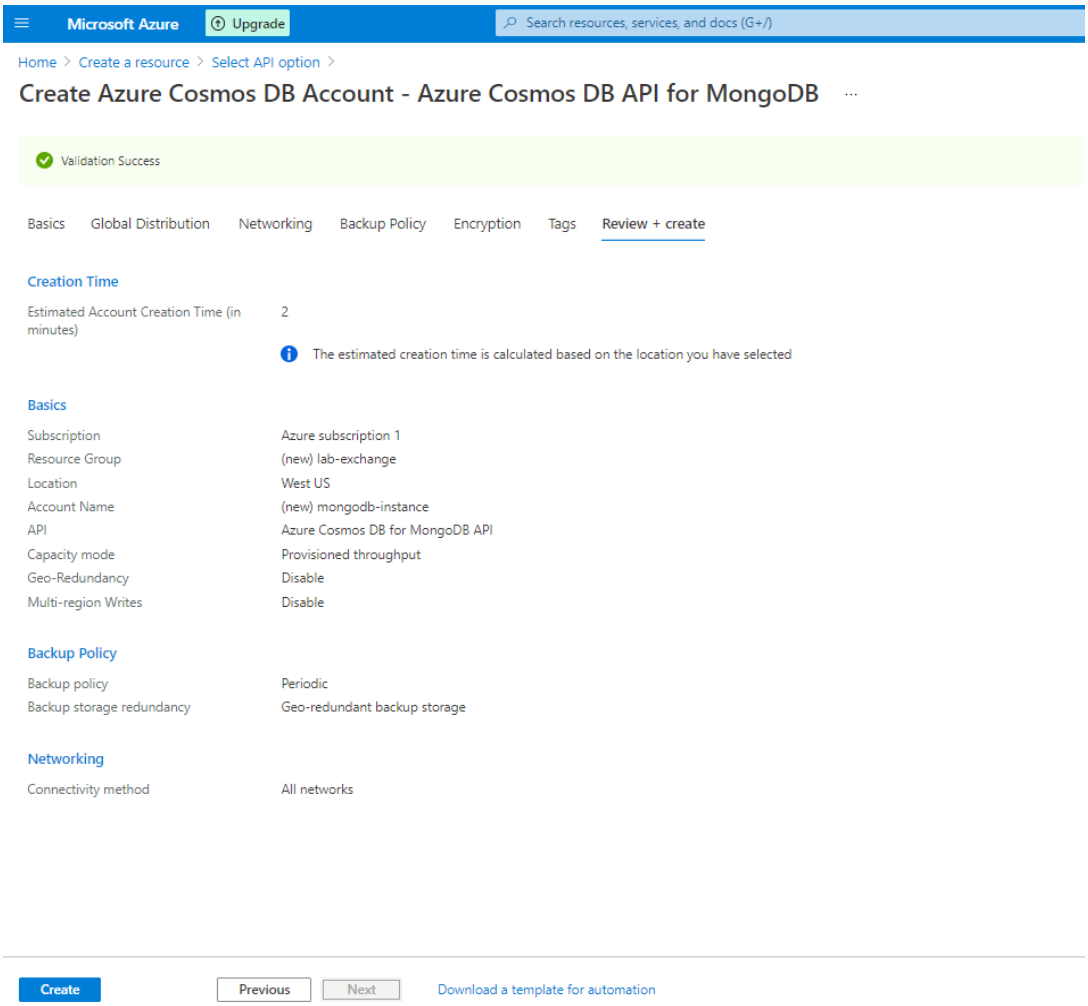
Σχέδιο 36: Επιλογή Τεχνολογιών | MongoDB – Καρτέλα επιλογής API βάσης προς δημιουργία

Σε αυτό το σημείο δηλώνονται οι προδιαγραφές της υπηρεσίας, όπως όνομα, περιοχή, πακέτο κοστολόγησης και έκδοση MongoDB API.



Σχέδιο 37: Επιλογή Τεχνολογιών | MongoDB – Καρτέλα προδιαγραφών βάσης προς δημιουργία

Τέλος, η υπηρεσία είναι έτοιμη προς χρήση μόλις πατηθεί η επιλογή “Create”. Για την χρήση της, αρκεί να χρησιμοποιηθεί το URI που δίνεται στο config file (application.yaml) των υπηρεσιών στο σημείο που συνδέεται η mongodb βάση.



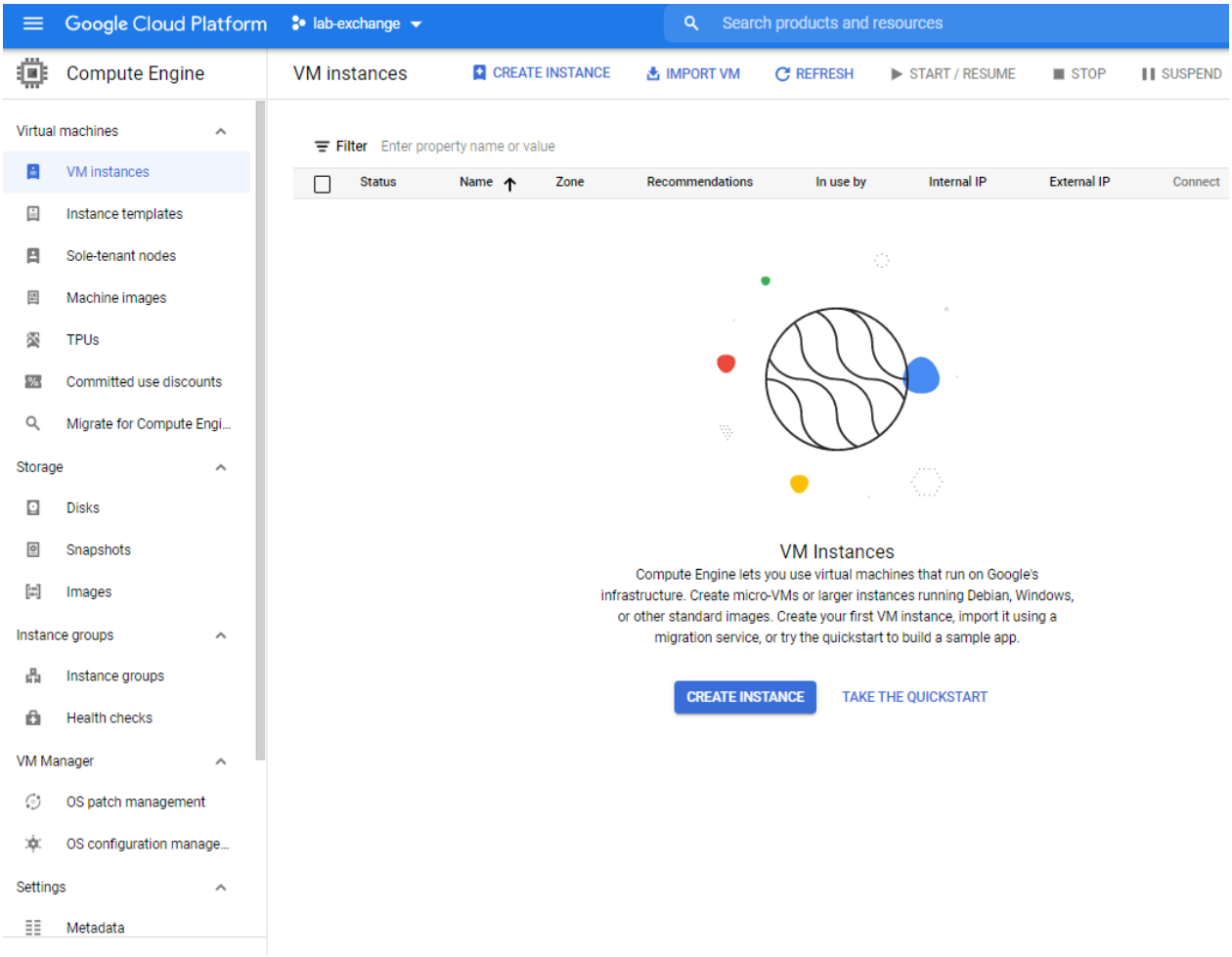
Σχέδιο 38: Επιλογή Τεχνολογιών | MongoDB – Ανασκόπηση προδιαγραφών του προς δημιουργία στιγμιοτύπου βάσης

Υπηρεσία Υποστήριξης RabbitMQ

Η μοναδική δωρεάν υπηρεσία που παρέχει RabbitMQ είναι η Amazon MQ της AWS. Όμως, η Amazon MQ RabbitMQ δεν υποστηρίζει το STOMP πρωτόκολλο το οποίο χρησιμοποιείται αρκετά από την εφαρμογή lab exchange. Επομένως, θα πρέπει να πραγματοποιηθεί χειροκίνητη εισαγωγή της υπηρεσίας με χρήση της μέσα από VM.

Το VM που θα αξιοποιηθεί θα είναι αυτό της Google Cloud το οποίο σε αντίθεση με των

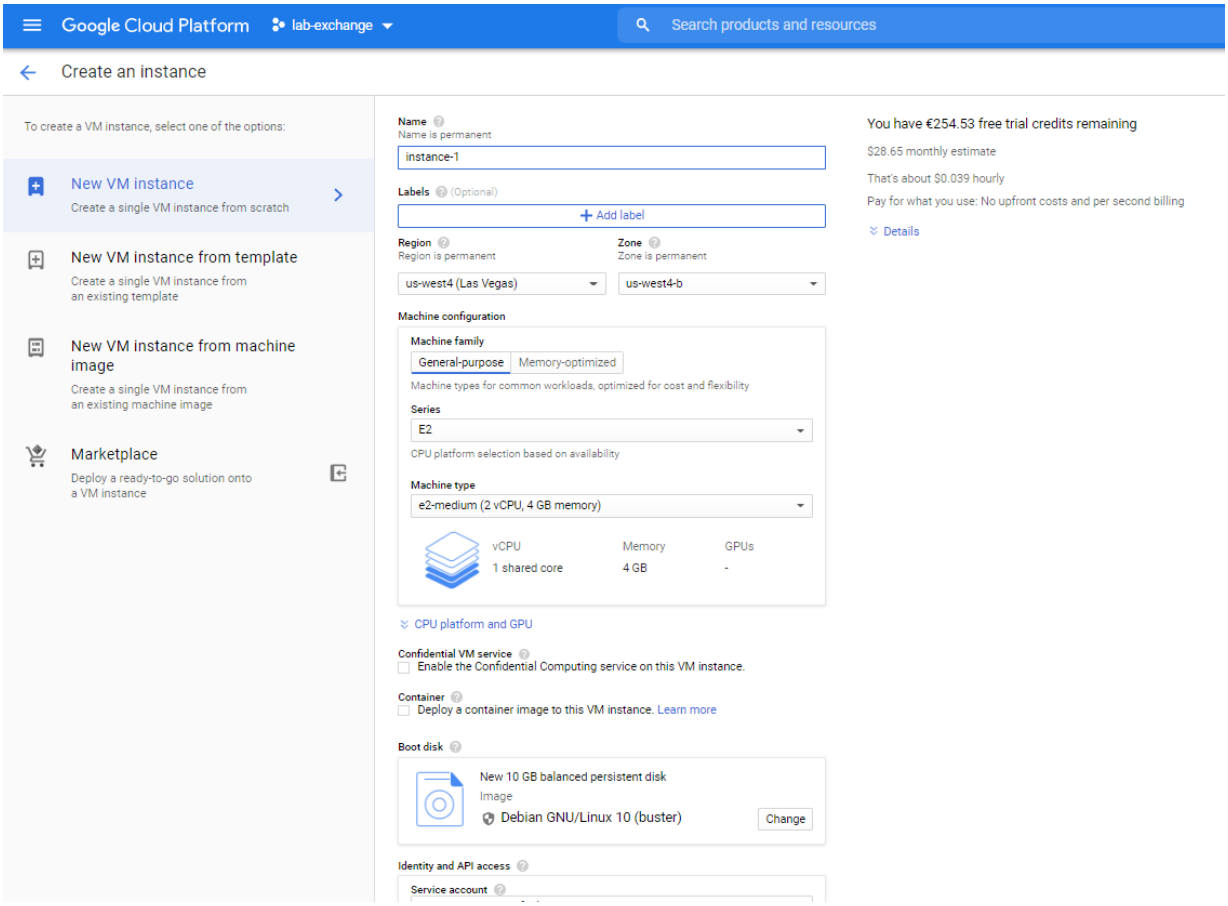
ανταγωνιστών παρέχεται δωρεάν αορίστως.



Σχέδιο 39: Επιλογή Τεχνολογιών | RabbitMQ – Καρτέλα λίστας VMs

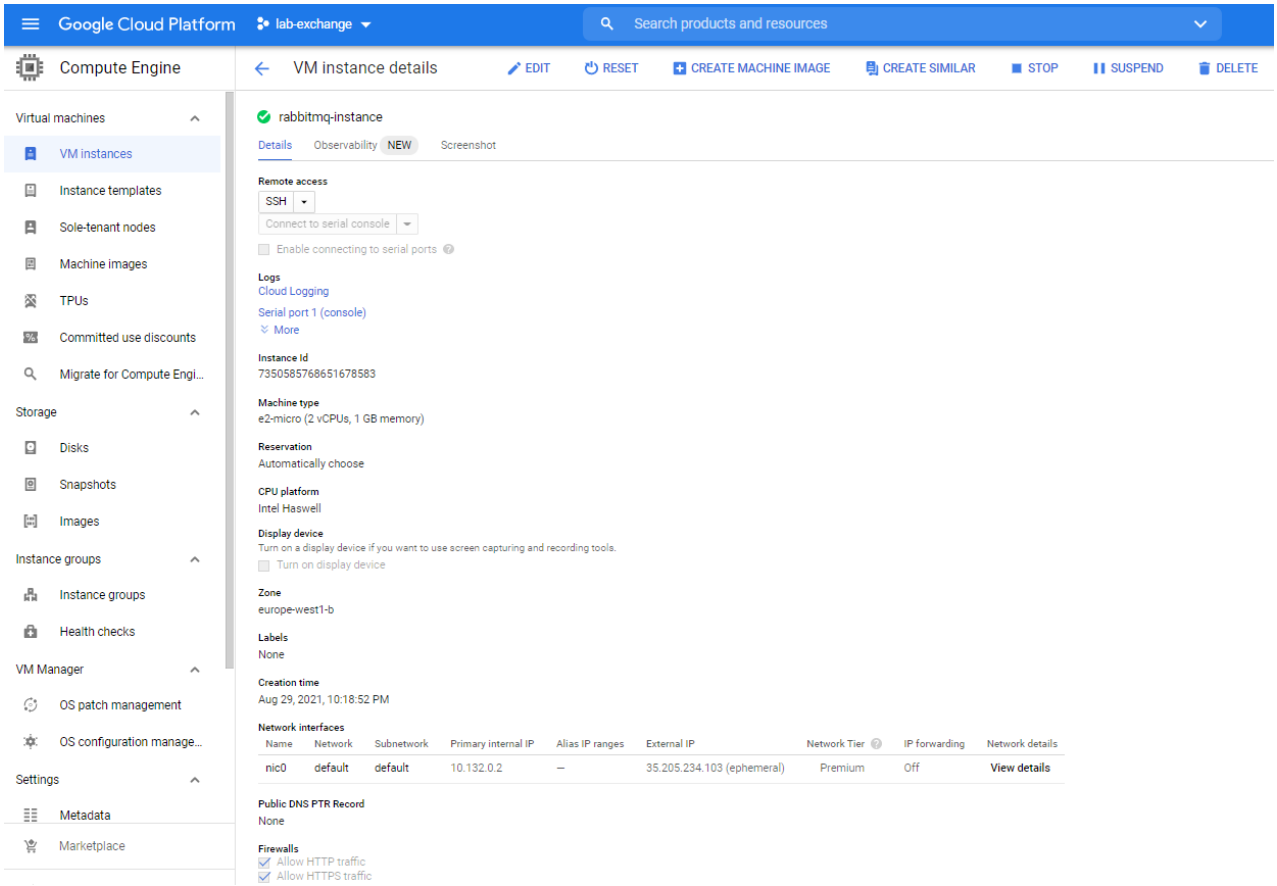
Επομένως, εφόσον έχει δημιουργηθεί Free Tier λογαριασμός στο Google Cloud, γίνεται η επιλογή της κατηγορίας “Compute Engine” και “VM Instances”.

Στην συνέχεια επιλέγονται οι προδιαγραφές του VM. Εδώ απαιτεί προσοχή το μοντέλο του VM που πρέπει να είναι non-preemptible e2-micro instance (σε region όπως το us-west1) για να βρίσκεται εντός του Free Tier πακέτου.



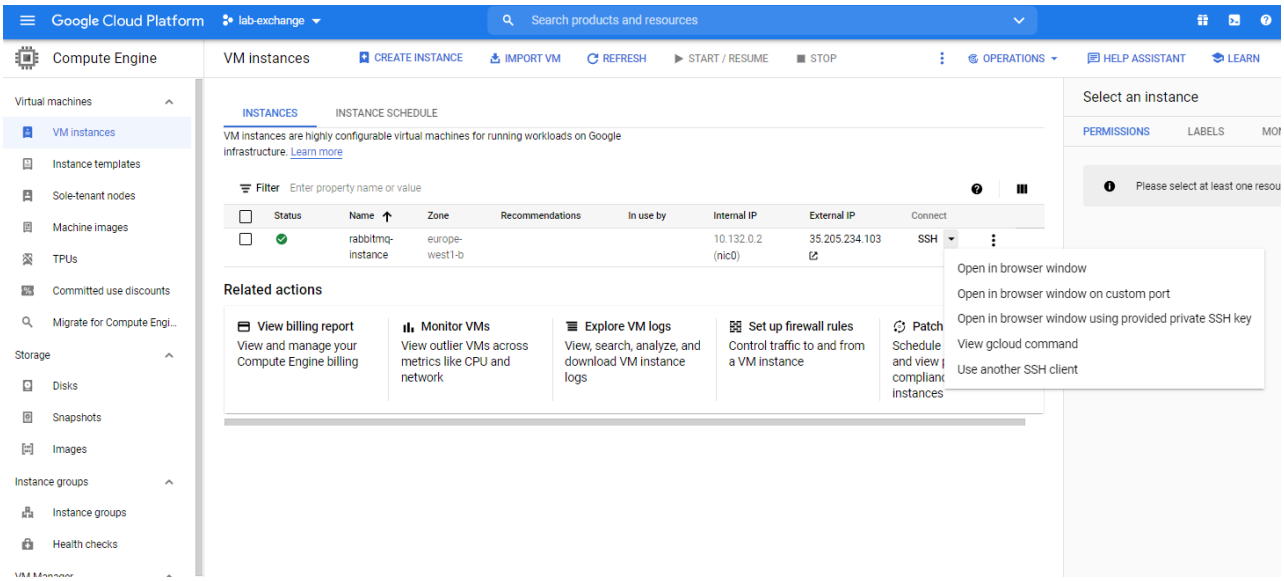
Σχέδιο 40: Επιλογή Τεχνολογιών | RabbitMQ – Πρώτο μέρος προδιαγραφών δημιουργίας VM

Στην συνέχεια, με την επιλογή “Create Virtual Machine” γίνεται η δημιουργία του VM.



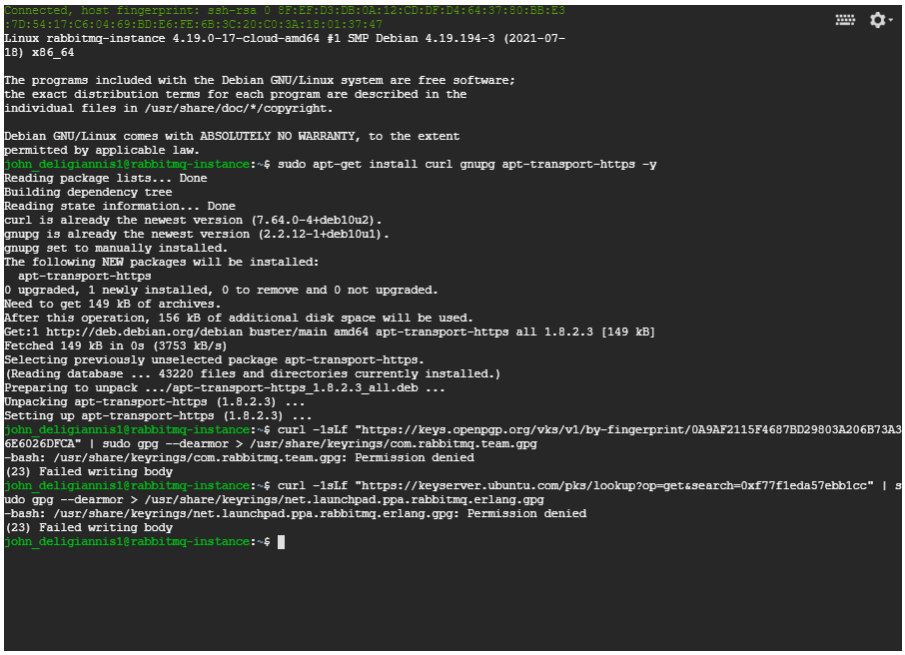
Σχέδιο 41: Επιλογή Τεχνολογιών | RabbitMQ – Ανασκόπηση προδιαγραφών του προς δημιουργία VM

Αφού έγινε η δημιουργία του VM, μπορεί να αποκτηθεί πρόσβαση σε αυτό με την επιλογή “open in browser window” όπως φαίνεται στην εικόνα 41.



Σχέδιο 42: Επιλογή Τεχνολογιών | RabbitMQ – λίστα των VM instances

Μόλις ενεργοποιηθεί το VM, πραγματοποιείται η εγκατάσταση του RabbitMQ για Linux.



Σχέδιο 43: Επιλογή Τεχνολογιών | RabbitMQ – VM's terminal

Τελευταία αλλαγή που απαιτείται, είναι η αλλαγή των RabbitMQ credentials στα config (application.yaml) αρχεία των μικρουπηρεσιών που το χρησιμοποιούν (notification service).

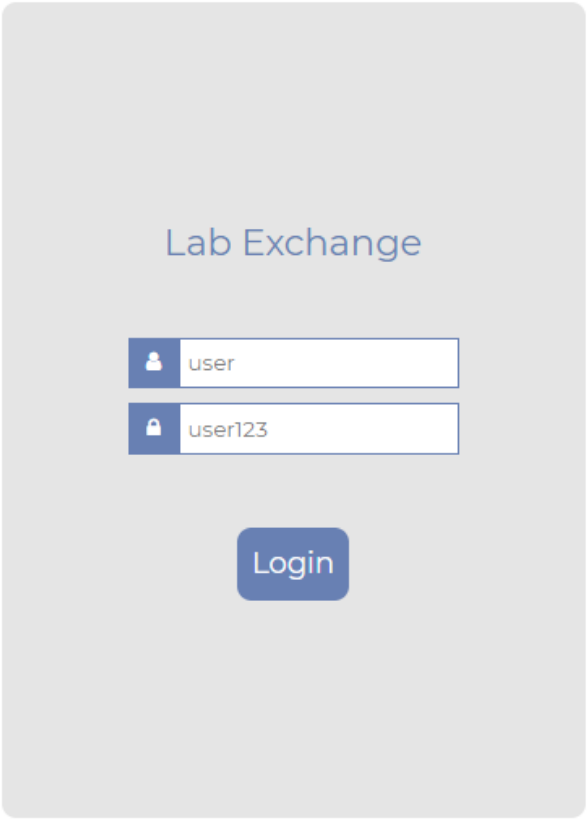
Παρουσίαση Εφαρμογής

Σε αυτή την ενότητα ακολουθεί η παρουσίαση της εφαρμογής. Για να είναι εφικτή η παρουσίαση χρειάζονται ψεύτικα δεδομένα δοκιμής. Για τον σκοπό αυτό έχουν υλοποιηθεί δύο συναρτήσεις που είναι εκτεθειμένες στο API της εφαρμογής για την δημιουργία ψεύτικων δεδομένων και τον καθαρισμό των βάσεων μετά. Αυτές είναι οι εξής:

"https://lab-exchange.com/api/classes/data/fill" για την δημιουργία των δεδομένων

"https://lab-exchange.com/api/classes/data/clean" για τον καθαρισμό των βάσεων

Αφού λοιπόν η εφαρμογή λειτουργεί και έχουν τοποθετηθεί τα ψεύτικα δεδομένα ο χρήστης οδηγείται στην αρχική σελίδα της.



Σχέδιο 44: Παρουσίαση Εφαρμογής - Είσοδος στο σύστημα

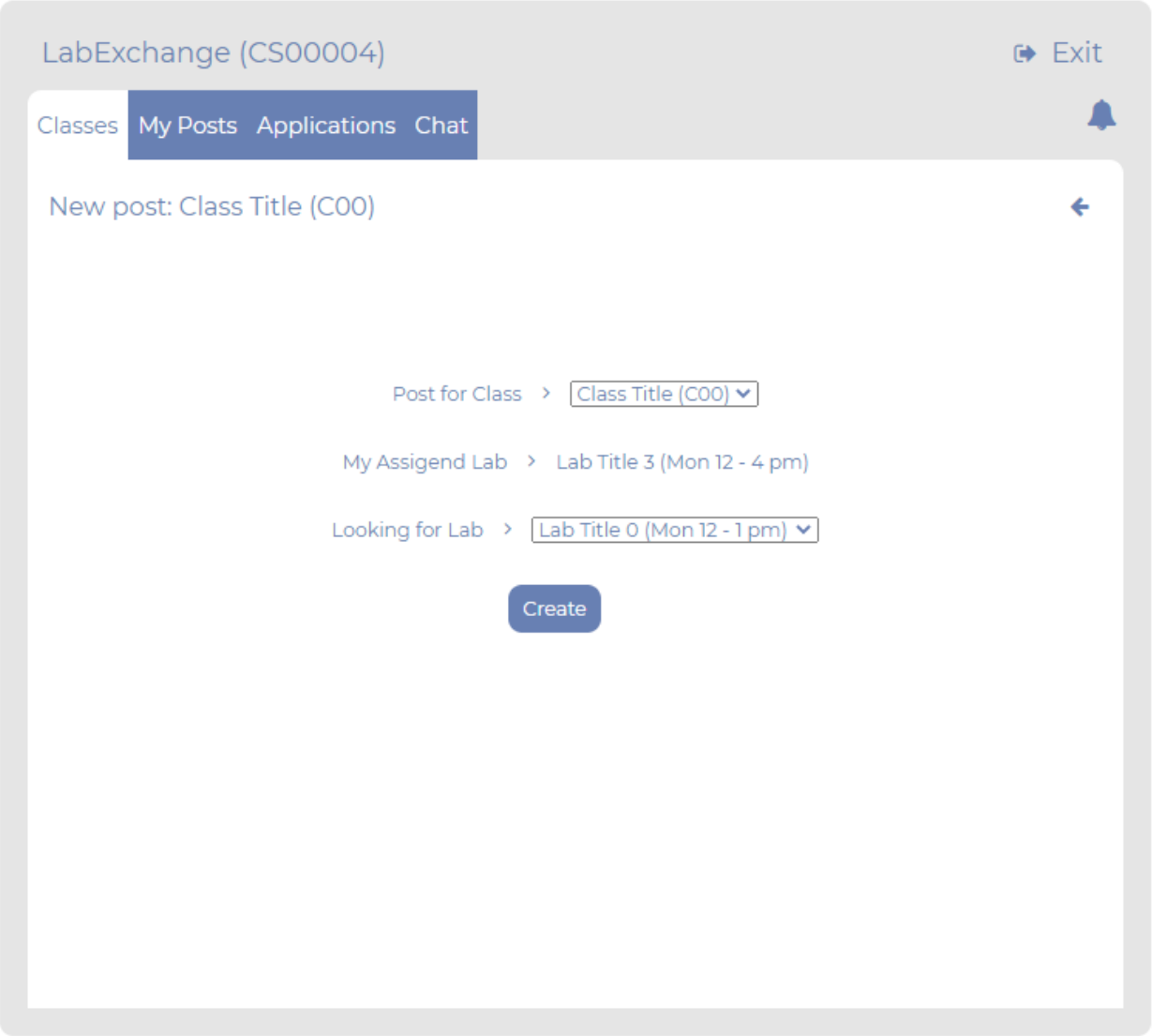
Είσοδος ως φοιτητής

Στην αρχική σελίδα του φοιτητή φαίνονται όλα τα μαθήματα στα οποία είναι εγγεγραμμένος. Στο κάθε μάθημα φαίνεται το εργαστήριο στο οποίο είναι εγγεγραμμένος (αν είναι) και αν στο μάθημα αυτό επιτρέπονται οι ανταλλαγές εργαστηρίων.

Σχέδιο 45: Παρουσίαση Εφαρμογής - Καρτέλα Μαθημάτων Φοιτητή

Ο χρήστης με την επιλογή κάποιου από τα μαθήματα τις καρτέλας μεταφέρεται στην καρτέλα μαθήματος.

Πατώντας την επιλογή “New Post”, ο χρήστης μεταφέρεται στην καρτέλα δημιουργίας δημοσίευσης ανταλλαγής εργαστηρίου. Εδώ ο χρήστης επιλέγει το μάθημα στο οποίο επιθυμεί ανταλλαγή εργαστηρίου, και το εργαστήριο στο οποίο επιθυμεί να πάει.



Σχέδιο 46: Παρουσίαση Εφαρμογής - Καρτέλα Δημιουργίας Δημοσίευσης Ανταλλαγής Εργαστηρίου Φοιτητή

Το εργαστήριο στο οποίο ανήκει φορτώνεται αυτόματα αν έχει. Αν ο χρήστης δεν ανήκει σε κάποιο εργαστήριο δεν μπορεί να χρησιμοποιήσει την συγκεκριμένη λειτουργία.

Πατώντας σε κάποιο από τα μαθήματα στην αρχική καρτέλα μαθημάτων, ο χρήστης μεταφέρεται στην καρτέλα μαθήματος. Στην καρτέλα αυτή εμφανίζονται δημοσιεύσεις άλλων φοιτητών για

εργαστήρια που ανταλλάσουν και ανήκουν στο μάθημα αυτό.

Σχέδιο 47: Παρουσίαση Εφαρμογής - Καρτέλα Μαθήματος και Δημοσιεύσεων Ανταλλαγής Εργαστηρίων Φοιτητή

Εδώ ο χρήστης μπορεί να δηλώσει ενδιαφέρων σε κάποια δημοσίευση και να κάνει αίτηση για ανταλλαγή. Για την εκτέλεση της αίτησης ο χρήστης θα πρέπει να ανήκει στο εργαστήριο το οποίο αναγράφεται στην αντίστοιχη δημοσίευση ως αναζητούμενο. Επίσης, ο χρήστης έχει την δυνατότητα να στείλει άμεσο μήνυμα στον δημιουργό της δημοσίευσης πατώντας στην επιλογή “Message”.

Στην καρτέλα “Applications” παρουσιάζονται οι αιτήσεις σε δημοσιεύσεις ανταλλαγής εργαστηρίου στις οποίες έχει δηλώσει ενδιαφέρων ο χρήστης.

Σχέδιο 48: Παρουσίαση Εφαρμογής - Καρτέλα Αιτήσεων σε Δημοσιεύσεις Φοιτητή

Ο χρήστης μπορεί από εδώ να συνομιλήσει με άμεσο μήνυμα με τον δημιουργό της δημοσίευσης ή και να ακυρώσει την αίτηση του.

Στην καρτέλα “My Posts” παρουσιάζονται οι δημοσιεύσεις τις οποίες έχει δημιουργήσει ο χρήστης. Σε κάθε δημοσίευση ο χρήστης μπορεί να δει τους φοιτητές οι οποίοι έχουν δηλώσει ενδιαφέρων και έχουν κάνει αίτηση για ανταλλαγή του εργαστηρίου τους με του χρήστη.

Σχέδιο 49: Παρουσίαση Εφαρμογής - Καρτέλα Δημοσιεύσεων Αναζήτησης Εργαστηρίου Φοιτητή

Ο χρήστης μπορεί να δει το εργαστήριο στο οποίο ανήκουν όσοι έχουν κάνει αίτηση στις δημοσιεύσεις του. Αν κάποια αίτηση ανταλλαγής τους ενδιαφέρει, τότε με την επιλογή "Exchange labs" ανταλλάσσονται τα εργαστήρια τους και τους στέλνεται σχετική ειδοποίηση. Ο χρήστης έχει επίσης την δυνατότητα να στείλει άμεσο μήνυμα στους φοιτητές που έχουν κάνει αίτηση.

Πατώντας στην επιλογή των ειδοποιήσεων, ο χρήστης ενημερώνεται για γεγονότα ενδιαφέροντος που έχουν συμβεί. Οι ειδοποιήσεις που εμφανίζονται αφορούν: Αιτήσεις που έγιναν σε δημοσίευση του χρήστη, Μηνύματα που έχουν στείλει φοιτητές στον χρήστη, Ολοκληρώσεις ανταλλαγών

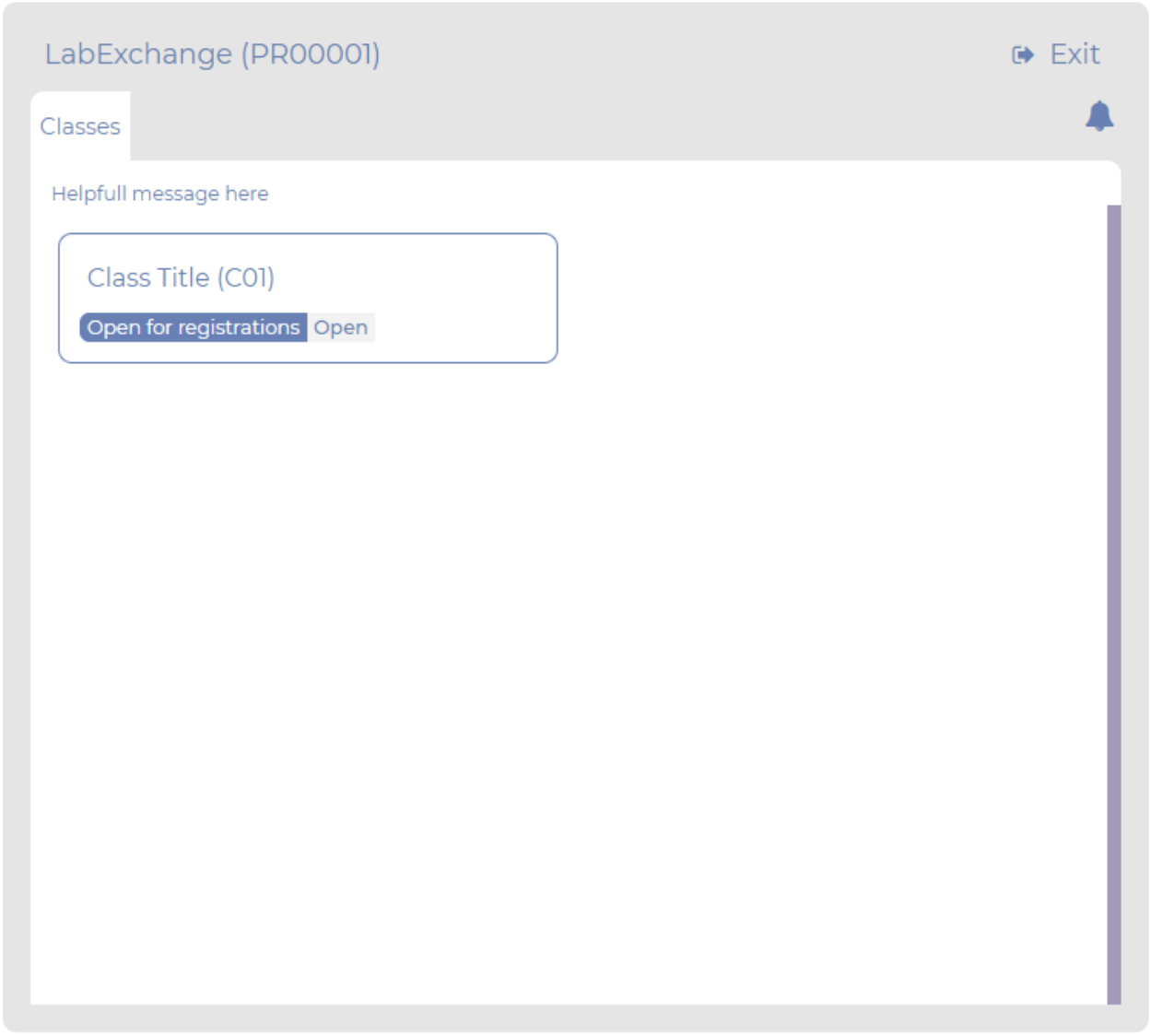
εργαστηρίων που έχουν γίνει.

Σχέδιο 50: Παρουσίαση Εφαρμογής - Καρτέλα Μαθημάτων - Ειδοποιήσεις Φοιτητή

Πατώντας στην κάθε ειδοποίηση ο χρήστης μεταφέρεται στην αντίστοιχη καρτέλα. Επίσης, όταν ο χρήστης βρίσκεται ήδη στην αντίστοιχη καρτέλα που συμβαίνει το γεγονός, δεν στέλνεται ειδοποίηση για αυτό, εκτός από την ανταλλαγή εργαστηρίου.

Είσοδος ως Καθηγητής

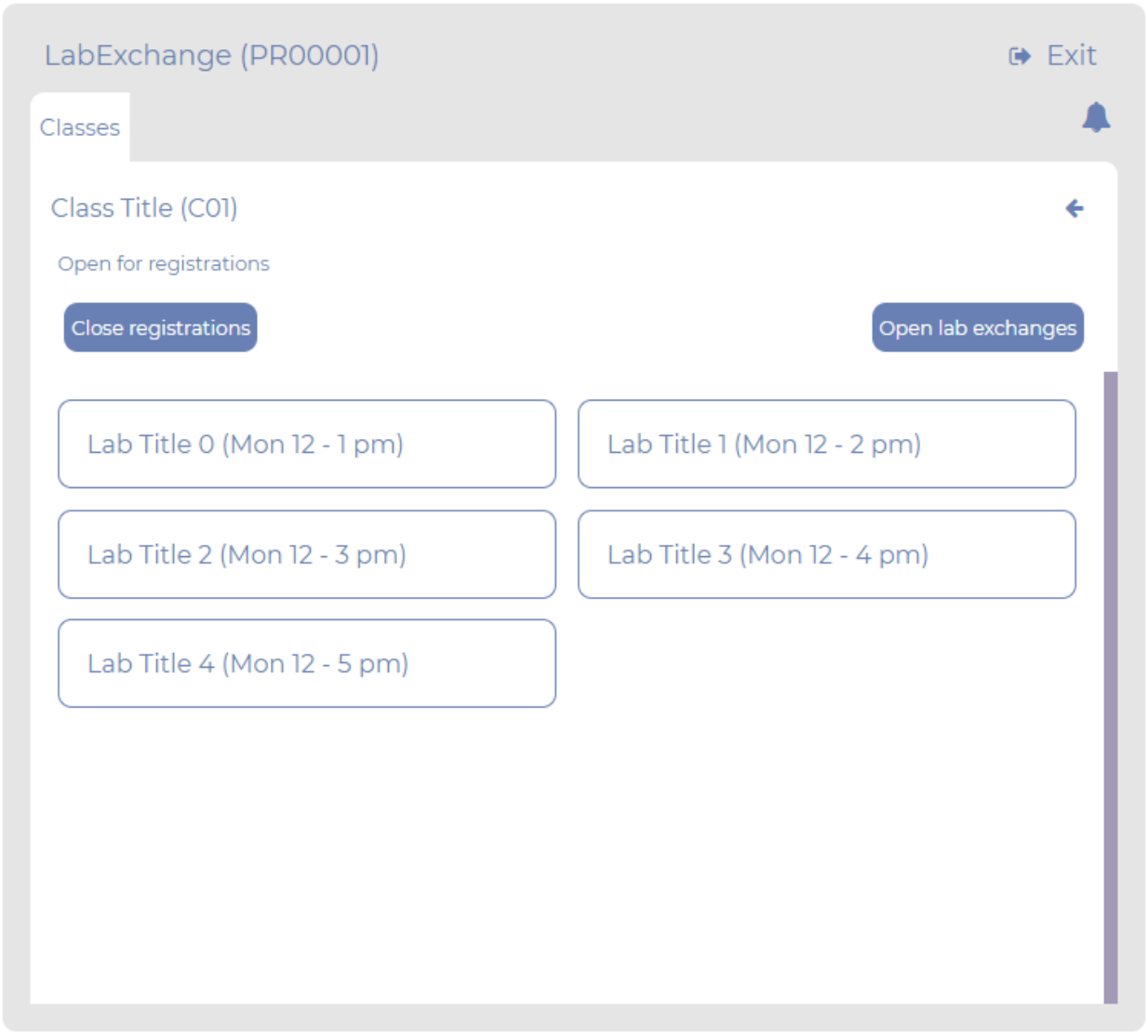
Στην αρχική σελίδα του καθηγητή φαίνονται όλα τα μαθήματα στα οποία είναι εγγεγραμμένος. Για κάθε μάθημα φαίνεται και αν είναι ενεργοποιημένη σε αυτό η επιλογή για ανταλλαγή εργαστηρίων.



Σχέδιο 51: Παρουσίαση Εφαρμογής - Καρτέλα Μαθημάτων Καθηγητή

Ο χρήστης μπορεί να μεταφερθεί στην καρτέλα συγκεκριμένου μαθήματος με την επιλογή κάποιου μαθήματος από εδώ.

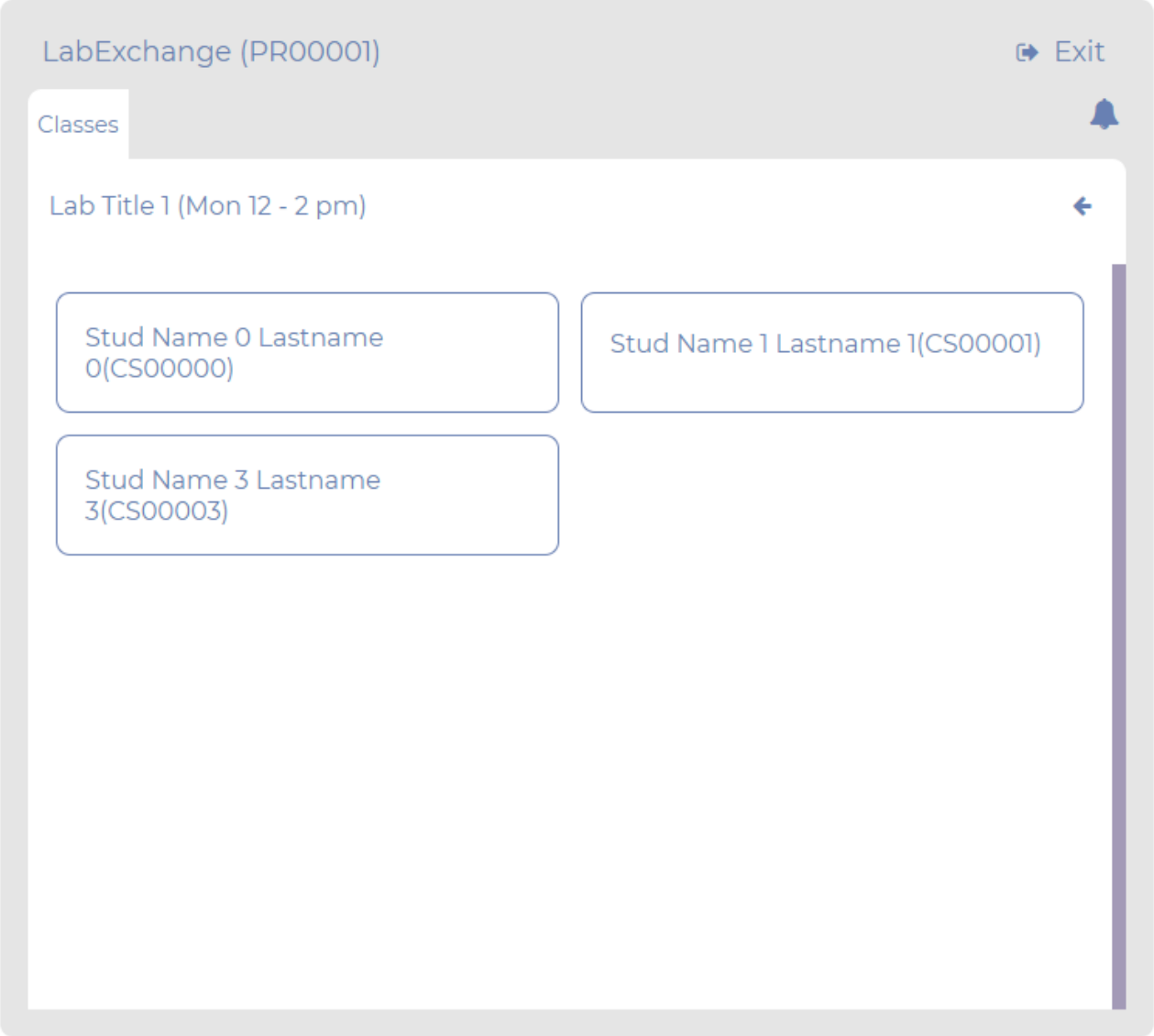
Στην καρτέλα μαθήματος ο χρήστης βλέπει τα εργαστήρια τα οποία είναι διαθέσιμα σε αυτό. Ο χρήστης έχει την επιλογή να απενεργοποιήσει τις ανταλλαγές εργαστηρίων για αυτό το μάθημα πατώντας την επιλογή “Close registrations”.



Σχέδιο 52: Παρουσίαση Εφαρμογής - Καρτέλα Μαθήματος και Εργαστηρίων Καθηγητή

Ο χρήστης έχει επίσης την επιλογή να ανοίξει την καρτέλα κάποιου εργαστηρίου και την καρτέλα των ανταλλαγών.

Στην καρτέλα εργαστηρίου μαθήματος φαίνονται όλοι οι φοιτητές οι οποίοι είναι εγγεγραμμένοι σε αυτό.



Σχέδιο 53: Παρουσίαση Εφαρμογής - Καρτέλα Φοιτητών Εργαστηρίου Καθηγητή

Στην καρτέλα εμφάνισης ανταλλαγών εργαστηρίων σε κάποιο μάθημα φαίνονται όλες οι αμοιβαίες ανταλλαγές εργαστηρίων μεταξύ φοιτητών για το συγκεκριμένο μάθημα.

Σχέδιο 54: Παρουσίαση Εφαρμογής - Καρτέλα Ανταλλαγών Εργαστηρίων Καθηγητή

Με χρήση της καρτέλας αυτής ως ιστορικό και της καρτέλας φοιτητών σε κάθε εργαστήριο, οι καθηγητές μπορούν να έχουν μία εικόνα της τελικής κατάστασης φοιτητών ανά εργαστήριο μετά το πέρας των ανταλλαγών. Με αυτό τον τρόπο μπορούν ύστερα να πραγματοποιήσουν μαζικά τις απαραίτητες αλλαγές στην πλατφόρμα του e-class χωρίς επιπλέον διαδικασίες.

Συμπεράσματα Δοκιμών και Βελτιώσεις

Σε αυτή την ενότητα θα αποτυπωθούν τα τελικά συμπεράσματα που εξάγονται μετά την ανάπτυξη της εφαρμογής και χρήσης των υπηρεσιών νέφους.

Containerization

Η συσκευή των συνιστωσών της εφαρμογής σε containers έφερε σημαντικά πλεονεκτήματα στην μεταφερσιμότητα και ανεξαρτησία τους. Η ανάλυση και χρήση των containers έγινε σε βασικό επίπεδο και έτσι, για την πλήρη αξιοποίηση του θα έπρεπε να αναλυθούν και άλλες πτυχές του όπως η κατασκευή και χρήση διαφορετικών δικτύων.

Orchestration

Το orchestration του Kubernetes πρόσφερε ένα ενιαίο περιβάλλον φιλοξενίας της εφαρμογής. Οι δυνατότητες του ως προς την παραμετροποίηση των δικτύων, χώρων αποθήκευσης, διαχείρισης των αιτημάτων και ελέγχου των containers το καθιστά ιδανικό για μικρουπηρεσίες και διαχείριση πόρων νέφους. Όμως, οι δυνατότητες αυτές, αλλά και άλλες που διαθέτει δεν αξιοποιήθηκαν στο έπακρο. Με χρήση υπηρεσιών του όπως Namespaces, Helm (package managing), ConfigMap, Secret etc, μπορεί να παραχθεί αρκετά καλύτερο αποτέλεσμα ικανό για διαχείριση μεγάλων εμπορικών συστημάτων.

Μικρουπηρεσίες

Με την χρήση του Spring Boot δεν παρουσιάστηκαν δυσκολίες στην ανάπτυξη των μικρουπηρεσιών και την σύνδεση τους με τις υποστηρικτικές υπηρεσίες. Όμως, αν και δόθηκε προσοχή στην ασφάλεια και χρησιμοποιήθηκε JWT κλειδί για την αυθεντικοποίηση των χρηστών, ακόμη και στην σύνδεση (subscribe) με την υπηρεσία STOMP από το front-end, το κομμάτι της ασφάλειας θα χρειαζόταν επιπλέον διερεύνηση.

Υποστηρικτικές Υπηρεσίες

Η χρήση των RabbitMQ και MongoDB έγινε σε βασικό επίπεδο και χρησιμοποιήθηκαν οι βασικές ρυθμίσεις τους. Για χρήση σε περιβάλλον παραγωγής θα έπρεπε να αλλάξουν αυτές οι ρυθμίσεις (πχ κλειδιά ασφαλείας) και να ακολουθηθούν οι σωστές μεθοδολογίες εγκατάστασης.

Lab Exchange

Η εφαρμογή lab-exchange φάνηκε να ανταποκρίνεται ικανοποιητικά κατά την χρήση της και η σύνδεση των μικρουπηρεσιών με το pwa είναι διάφανη και δίνει την εντύπωση desktop εφαρμογής. Όμως, το επίπεδο λειτουργιών και εμφάνισης της εφαρμογής διατηρήθηκε σε βασικά επίπεδα και δεν ανταποκρίνεται σε τελική λύση έτοιμη για ανάπτυξη σε περιβάλλον παραγωγής.

Κεφάλαιο 5: Συμπεράσματα

Η τήρηση του μεγαλύτερου μέρους της μεθοδολογίας 12 παραγόντων στον σχεδιασμό της cloud native εφαρμογής αποδείχθηκε εύκολη διαδικασία η οποία προσφέρει σημαντικά πλεονεκτήματα στην διαχείριση της. Βασικά πλεονεκτήματα της εφαρμογής είναι η πλήρης ανεξαρτησία της από το περιβάλλον ανάπτυξης που φιλοξενείται, η ευκολία διαχείρισης του κώδικα σε τμήματα και η εύκολη ανάπτυξη με αυτοματοποιημένες τεχνικές CI/CD. Δυσκολίες στην υλοποίηση παρουσίασαν οι παράγοντες Αρχείων Συμβάντων (11) και Διεργασιών Διαχείρισης (12), οι οποίοι απαιτούν έναν ενιαίο τρόπο παρακολούθησης και διαχείρισης όλων των μικρουπηρεσιών. Βασικές τεχνολογίες που βοηθούν στην τήρηση των παραγόντων είναι οι πλατφόρμες containerization και orchestration οι οποίες όπως αποδείχθηκε, με απλούς δηλωτικούς κανόνες μέσα από αρχεία διαμόρφωσης ορίζουν όλη την απαραίτητη υποδομή που απαιτείται για τον έλεγχο και διαχείριση των εφαρμογών.

Η χρήση cloud native μεθοδολογιών για την ανάπτυξη συστημάτων ωθεί τους μηχανικούς στην κατασκευή συστημάτων που διαχειρίζονται αποδοτικά τους υπολογιστικούς πόρους που χρησιμοποιούν διότι με αυτό τον τρόπο μειώνουν και τα έξοδα τους προς τους πάροχους υπηρεσιών.

Σχετικά με τους πάροχους υπηρεσιών νέφους, παρατηρήθηκαν παρόμοιες λύσεις με τις σημαντικότερες διαφορές σε αποθηκευτικό χώρο και βάσεις δεδομένων ενώ, οι υπηρεσίες που προσφέρουν τις μεγαλύτερες δωρεάν παροχές είναι συνήθως οι ιδιόκτητες λύσεις.

Οι ιδιόκτητες λύσεις εγκυμονούν κινδύνους όπως το vendor lock-in και θα πρέπει να χρησιμοποιούνται κατόπιν διεξοδικής ανάλυσης ειδικά σε πελάτες υψηλών απαιτήσεων. Οι υπηρεσίες που υποστηρίζουν cloud-agnostic τεχνολογίες απαιτούν εξίσου ιδιαίτερη προσοχή, καθώς συνήθως έχουν περιορισμένο εύρος σε υποστηριζόμενες εκδόσεις λογισμικού αλλά και δεν εγγυούνται ότι θα συνεχίσουν να υποστηρίζουν τις επόμενες εκδόσεις του.

Αξιοσημείωτο πρόβλημα που υπήρχε κατά την διάρκεια της μελέτης τεχνολογιών από διαφορετικούς παρόχους ήταν οι διαφορετικοί τρόποι με τους οποίους τιμολογούν τις υπηρεσίες τους. Οι υπηρεσίες τιμολογούνται όχι μόνο με διαφορετικό τρόπο ανά μοντέλο αλλά και ανά διαθέσιμης γεωγραφικής περιοχής, γεγονός που προσφέρει επιπρόσθετη περιπλοκότητα στην διαδικασία αξιολόγησης του οικονομικότερου πακέτου.

Τα δωρεάν πακέτα προσφορών αποτελούν μία καλή λύση για μηχανικούς οι οποίοι θέλουν να δοκιμάσουν τους διαφορετικούς τρόπους ανάπτυξης και λειτουργίας που παρέχουν οι πάροχοι υπηρεσιών σε μικρής κλίμακας έργα δοκιμών. Πελάτες υψηλών απαιτήσεων που θέλουν να δοκιμάσουν τα προϊόντα δεν θα βοηθηθούν ιδιαίτερα από τα δωρεάν πακέτα καθώς είναι αρκετά περιορισμένα σε υπολογιστικούς πόρους, και ακόμη και με την χρήση όλου του δωρεάν ποσού πίστωσης έχουν στην διάθεση τους λίγες συνολικά μέρες για την αξιολόγηση τους.

Κεφάλαιο 6: Βιβλιογραφία

Ελληνική

[1] Μανόλης Γιακουμάκης, Νίκος Διαμαντίδης, 2009. Τεχνολογία Λογισμικού. Διαστρωματωμένη αρχιτεκτονική, σελίδα 356.

[2] Μανόλης Γιακουμάκης, Νίκος Διαμαντίδης, 2009. Τεχνολογία Λογισμικού. Διαγράμματα παράτασης, σελίδα 374.

[5] Καλιόπη Κανάκη, 2015. Υπολογιστικό Νέφος. Available at https://www.researchgate.net/publication/302873104_YPOLOGISTIKO_NEPHOS

Ξένη

[3] Chris Richardson, 2020, Pattern: Monolithic Architecture. Available at https://microservices.io/patterns/monolithic.html

[4] Peter Mell, Timothy Grance, 2011. The NIST Definition of Cloud Computing. Available at https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[6] Nane Kratzke, Rene Peinl, 2017. CloudNS - A Cloud-native Application Reference Model for Enterprise Architects. Available at https://arxiv.org/pdf/1709.04883.pdf

[7] Justin Garrison, Kris Nova, 20018. Cloud Native Infrastructure. What is Cloud Native Infrastructure, Servers, Virtualization. Available at https://books.google.gr/books?hl=el&lr=&id=zlk7DwAAQBAJ&oi=fnd&pg=PP1&dq=cloud+native+technology+providers&ots=tgpX_f1zIN&sig=LO_moZ9bjdz9a2FW2e7d5rUTboo&redir_esc=y#v=onepage&q=cloud%20native%20technology%20providers&f=false

[10] Mohamed K. Hussein, Mohamed H. Mousa, Mohamed A. Alqarni, 2019. A placement architecture for a container as a service (CaaS) in a cloud environment. Available at https://link.springer.com/article/10.1186/s13677-019-0131-1

[11] Maria A. Rodriguez, Rajkumar Buyya, 2018. Container-based cluster orchestration systems: A taxonomy and future directions. Available at https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2660?

[12] Adam Lith, Jakob Mattsson, 2010. Investigating storage soludations for large data. Available at https://publications.lib.chalmers.se/records/fulltext/123839.pdf

[13] Robert T. Mason, 2015. NoSQL Databases and Data Modeling Techniques for a Document-oriented NoSQL Database. Available at http://proceedings.informingscience.org/InSITE2015/InSITE15p259-268Mason1569.pdf

[14] Katarina Grolinger, Wilson A Higashino, Abhinav Tiwari, Miriam AM Capretz, 2013. Data management in cloud environments: NoSQL and NewSQL data stores. Available at https://journalofcloudcomputing.springeropen.com/track/pdf/10.1186/2192-113X-2-22.pdf

[16] Google Cloud, Tom Grey, 2019. 5 principles for cloud-native architecture - what it is and how to master it. Available at https://cloud.google.com/blog/products/application-development/5-principles-for-cloud-native-architecture-what-it-is-and-how-to-master-it

[19] O'REILLY | Sam Newman, 2015. Building Microservices. Available at http://xfido.com/pdf/building_microservices.pdf

[20] Microsoft | Robert Vettor, Steve Smith, 2021. Architecting Cloud-Native .NET Apps for Azure. Available at https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/definition

[21] Boubaker Soltani, Afifa Ghenai, Nadia Zeghib, 2018. Towards Distributed Containerized Serverless Architecture in Multi Cloud Environment. Available at https://www.sciencedirect.com/science/article/pii/S1877050918311153

[22] Chen-Fu Fan, Anshul Jindal, Michael Gerndt, 2020. Microservices vs Serverless: A Performance Comparison on a Cloud-native Web Application. Available at https://www.researchgate.net/profile/Anshul-Jindal-2/publication/341483472_Microservices_vs_Serverless_A_Performance_Comparison_on_a_Cloud-native_Web_Application/links/5fd008d1a6fdcc697bef60da/Microservices-vs-Serverless-A-Performance-Comparison-on-a-Cloud-native-Web-Application.pdf

[24] Tom Laszewski, Kamal Arora, Erik Farr, Piyum Zonooz, 2018. Cloud Native Architectures. Available at https://books.google.gr/books?hl=el&lr=&id=QshsDwAAQBAJ&oi=fnd&pg=PP1&dq=ci+cd+cloud+native&ots=qE9NhhFEJA&sig=rd2ndJ0l-fK2yJSH4ba6kNfwJm0

Διαδικτυακές Πηγές

[8] Kubernetes, 2021. What is Kubernetes. Available at https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[9] Docker, 2021. What is a Container. Available at https://www.docker.com/resources/what-container

[15] IBM Cloud Education, 2020. What are Message Brokers? Available at https://www.ibm.com/cloud/learn/message-brokers

[17] IBM Cloud Education, 2021. SOA (Service-Oriented Architecture). Available at https://www.ibm.com/cloud/learn/soa

[18] IBM Cloud Education, 2021. SOA vs. Microservices: What's the Difference? Available at https://www.ibm.com/cloud/blog/soa-vs-microservices

[23] Heroku | Adam Wiggins, 2017. The Twelve-Factor App. Available at https://12factor.net/

[25] codilime | Maciej Manturewicz, 2019. What is CI/CD - all you need to know. Available at https://codilime.com/what-is-ci-cd-all-you-need-to-know/

[26] GitLab, 2021. What is cloud native continuous integration. Available at https://about.gitlab.com/topics/ci-cd/cloud-native-continuous-integration/

[27] React | Create React App. Folder Structure. Available https://create-react-app.dev/docs/folder-structure/

[28] Webpack. Available at https://webpack.js.org/

[29] React | Create a New React App. Available at https://reactjs.org/docs/create-a-new-react-app.html

[30] NPM | NPM Docs. Available at https://docs.npmjs.com/about-npm

[31] NPM | stompjs. Available at https://www.npmjs.com/package/stompjs

[32] React | React router. Quick start. Available at https://reactrouter.com/web/guides/quick-start

[33] Node. Available at https://nodejs.org/en/

[34] Mongo DB | Documentation. Introduction. Available at https://docs.mongodb.com/manual/introduction/

[35] RabbitMQ | Documentation. Available at https://www.rabbitmq.com/documentation.html

[36] RabbitMQ | Stomp Plugin. Available at https://www.rabbitmq.com/stomp.html

[37] Spring | Spring Framework Documentation. Available at https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html#overview

[38] Spring | Spring Boot. Available at https://spring.io/projects/spring-boot

[39] Maven. Available at https://maven.apache.org/

[40] Java Point | Maven pom.xml file. Available at https://www.javatpoint.com/maven-pom-xml

[41] Java Point | Spring Boot Application Properties. Available at https://www.javatpoint.com/spring-boot-properties

[42] Docker | Docker overview. Available at https://docs.docker.com/get-started/overview/

[43] Docker | Dockerfile. Available at https://docs.docker.com/engine/reference/builder/

[44] Docker | Docker Compose. Available at https://docs.docker.com/compose/

[45] Kubernetes | Pods. Available at https://kubernetes.io/docs/concepts/workloads/pods/

[46] Kubernetes | Deployments. Available at https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

[47] Kubernetes | Service. Available at https://kubernetes.io/docs/concepts/services-networking/service/

[48] Kubernetes | Persistent Volumes. Available at https://kubernetes.io/docs/concepts/storage/persistent-volumes/

[49] Kubernetes | Ingress. Available at https://kubernetes.io/docs/concepts/services-networking/ingress/

[50] Canalys, 2021, Global Cloud Infrastructure Market Q4 2020. Available at https://www.canalys.com/newsroom/global-cloud-market-q4-2020

[51] Datamation | James Magulre, 2021. Top Cloud Service Providers & Companies of 2021. Available at https://www.datamation.com/cloud/cloud-service-providers/

[52] Google Cloud | Google Cloud Free Program. Available at https://cloud.google.com/free/docs/gcp-free-tier/#free-trial

[53] Microsoft Azure | Free Tier. Available at https://azure.microsoft.com/en-us/free

[54] InfoWorld | Serdar Yegulalp, 2020. AWS vs. Azure vs. Google Cloud: Which free tier is the best? Available at https://www.infoworld.com/article/3179785/aws-vs-azure-vs-google-cloud-which-free-tier-is-best.html

[55] Google Cloud | Compute Engine Documenation. Available at https://cloud.google.com/compute/docs

[56] Google Cloud | App Engine. Available at https://cloud.google.com/appengine/docs

[57] Google Cloud | Cloud Build. Available at https://cloud.google.com/build/docs

[58] Google Cloud | Cloud Functions. Available at https://cloud.google.com/functions/docs

[59] Google Cloud | Cloud Run. Available at https://cloud.google.com/run/docs

[60] Google Cloud | Cloud Source Repositories. Available at https://cloud.google.com/source-repositories/docs

[61] Google Cloud | Cloud Storage. Available at https://cloud.google.com/storage/docs

[62] Google Cloud | Firestore. Available at https://cloud.google.com/firestore/docs

[63] Google Cloud | Google Kubernetes Engine. Available at https://cloud.google.com/kubernetes-engine/docs

[64] Google Cloud | Pub/Sub. Available at https://cloud.google.com/pubsub/docs

[65] Microsoft Azure | Virtual Machines. Available at https://azure.microsoft.com/en-us/services/virtual-machines/linux/#pricing

[66] Microsoft Azure | Azure Cosmos DB. Available at https://azure.microsoft.com/en-us/services/cosmos-db/#overview

[67] Microsoft Azure | Service Bus. Available at https://azure.microsoft.com/en-us/services/service-bus/#overview

[68] Microsoft Azure | Azure DevOps. Available at https://azure.microsoft.com/en-us/services/devops/#overview

[69] Microsoft Azure | Azure App Service. Available at https://azure.microsoft.com/en-us/services/app-service/

[70] Microsoft Azure | Azure Functions. Available at https://azure.microsoft.com/en-us/services/functions/

[71] Microsoft Azure | Azure Disk Storage. Available at https://azure.microsoft.com/en-us/services/storage/disks/#overview

[72] Microsoft Azure | Azure Blob Storage. Available at https://azure.microsoft.com/en-us/services/storage/blobs/

[73] Microsoft Azure | Azure Free Account. Available at https://azure.microsoft.com/en-us/free/free-account-faq/

[74] Microsoft Azure | Azure Kubernetes Engine. Available at https://azure.microsoft.com/en-us/services/kubernetes-service/

[75] Microsoft Azure | Azure Service Fabric. Available at https://azure.microsoft.com/en-us/services/service-fabric/

[76] TechTarget - Search Cloud Computing | What is the difference between AKS and Azure Service Fabric? Available at https://searchcloudcomputing.techtarget.com/answer/What-is-the-difference-between-AKS-and-Azure-Service-Fabric

[77] Google Cloud | Google Cloud for Azure Professionals: Storage. Available at https://cloud.google.com/docs/compare/azure/storage

[78] Microsoft | Find the request unit charge for operations executed in Azure Cosmos DB SQL API. Available at https://docs.microsoft.com/en-us/azure/cosmos-db/find-request-unit-charge?tabs=dotnetv2

[79] Microsoft | Rate limits. Available at https://docs.microsoft.com/en-us/azure/devops/integrate/concepts/rate-limits?view=azure-devops

[80] Amazon Web Services (AWS) | AWS Free Tier. Available at https://aws.amazon.com/free/

[81] Amazon Web Services (AWS) | Get started with the AWS Free Tier. Available at https://docs.aws.amazon.com/whitepapers/latest/how-aws-pricing-works/get-started-with-the-aws-free-tier.html

[82] Amazon Web Services (AWS) | Amazon EC2. Available at https://aws.amazon.com/ec2/

[83] Amazon Web Services (AWS) | AWS Lambda. Available at https://aws.amazon.com/lambda/

[84] Amazon Web Services (AWS) | Amazon S3. Available at https://aws.amazon.com/s3/

[85] Amazon Web Services (AWS) | Amazon Elastic Block Store. Available at https://aws.amazon.com/ebs/

[86] Amazon Web Services (AWS) | Amazon DynamoDB. Available at https://aws.amazon.com/dynamodb/

[87] Amazon Web Services (AWS) | Request Units in Azure Cosmos DB. Available at https://docs.microsoft.com/en-us/azure/cosmos-db/request-units

[88] Amazon Web Services (AWS) | Amazon SNS. Available at https://aws.amazon.com/sns/

[89] Amazon Web Services (AWS) | AWS CodeBuild. Available at https://aws.amazon.com/codebuild/

[90] Amazon Web Services (AWS) | AWS CodeCommit. Available at https://aws.amazon.com/codecommit/

[91] Amazon Web Services (AWS) | AWS CodePipeline. Available at https://aws.amazon.com/codepipeline/

[92] Amazon Web Services (AWS) | AWS Amplify. Available at https://aws.amazon.com/amplify/hosting/

[93] Microsoft | Azure SQL Database. Available at https://azure.microsoft.com/en-us/products/azure-sql/database/

[94] Microsoft | Azure Database for MySQL. Available at https://azure.microsoft.com/en-us/services/mysql/#overview

[95] Microsoft | Azure Database for PostgreSQL. Available at https://azure.microsoft.com/en-us/services/postgresql/#overview

[96] Amazon Web Services (AWS) | Amazon RDS. Available at https://aws.amazon.com/rds/?did=ft_card&trk=ft_card

[97] Microsoft | Azure Cosmos DB API for MongoDB. Available at https://docs.microsoft.com/en-us/azure/cosmos-db/mongodb/mongodb-introduction

[98] Microsoft | Apache Cassandra features supported by Azure Cosmos DB Cassandra API. Available at https://docs.microsoft.com/en-us/azure/cosmos-db/cassandra/cassandra-support

[99] Amazon Web Services (AWS) | MariaDB on Amazon RDS. Available at https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_MariaDB.html

[100] Amazon Web Services (AWS) | Microsoft SQL Server on Amazon RDS. Available at https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_SQLServer.html

[101] Amazon Web Services (AWS) | MySQL on Amazon RDS. Available at https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_MySQL.html

[102] Amazon Web Services (AWS) | Oracle on Amazon RDS. Available at https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Oracle.html

[103] Amazon Web Services (AWS) | PostgreSQL on Amazon RDS. Available at https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_PostgreSQL.html

[104] Mike Vanbuskirk, 2021. NoSQL databases comparison: Cosmos DB vs DynamoDB vs Cloud Datastore and Bigtable. Available at https://acloudguru.com/blog/engineering/comparing-cloud-nosql-databases-dynamodb-vs-cosmos-db-vs-cloud-datastore-and-bigtable

[105] Google Cloud | Choosing between Native mode and Datastore mode. Available at https://cloud.google.com/datastore/docs/firestore-or-datastore

[106] Amazon Web Services (AWS) | Amazon SQS. Available at https://aws.amazon.com/sqs/?did=ft_card&trk=ft_card

[107] Amazon Web Services (AWS) | Amazon MQ. Available at https://aws.amazon.com/amazon-mq/

[108] Alexander Potasnick, 2021. AKS vs EKS vs GKE: Managed Kubernetes services compared. Available at https://acloudguru.com/blog/engineering/aks-vs-eks-vs-gke-managed-kubernetes-services-compared

[109] CloudFlare | What is vendor lock-in? | Vendor lock-in and cloud computing. Available at https://www.cloudflare.com/learning/cloud/what-is-vendor-lock-in/

[110] Justice Opara-Martins, Reza Sahandi, Feng Tian, 2016. Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. Available at https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-016-0054-z

[111] Mindinventory | Paresh Solanki, 2021. 10 Biggest Cloud Computing Challenges in 2021 for IT Service Providers. Available at https://www.mindinventory.com/blog/cloud-computing-challenges/

[112] Morefield | Pros and cons of cloud computing. Available at https://www.morefield.com/blog/pros-and-cons-of-cloud-computing/

Κεφάλαιο 7: Παράρτημα

Ο κώδικας για το χτίσιμο της εφαρμογής Lab Exchange, μαζί με τα αρχεία διαμόρφωσης της για την ανάπτυξη του σε Docker και Kubernetes μπορεί να βρεθεί στα ακόλουθα repos:

- Web Application: https://github.com/JohnDelta/LabExchange_WebApplication.git
- Authentication service: https://github.com/JohnDelta/Authentication-Service.git
- Classes service: https://github.com/JohnDelta/Classes-Service.git
- Messenger service: https://github.com/JohnDelta/Messenger-service.git
- Notifications service: https://github.com/JohnDelta/Notification-service.git

Τα branches “deployed_without_k8s” αφορούν τα codebases που χρησιμοποιήθηκαν για την ανάπτυξη με τεχνολογίες νέφους, ενώ τα “master” περιέχουν την έκδοση για εκτέλεση τοπικά στο K8s.