Πανεπιστημιο Δυτικης Αττικης

Σχολή Μηχανικών
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Διπλωματική εργασία

Θεμα

Ανάπτυξη πλατφόρμας FaaS (Function as a Service) σε περιβάλλον εικονικοποίησης βασισμένης σε περιέκτες

Φοιτητής:
Ιάκωβος
Μαστρογιαννόπουλος

Επιβλέπων:
Βασίλειος Μάμαλης

Συν-επιβλέπων:
Απόστολος Αναγνωστόπουλος

21 Οκτωβρίου 2022

Πανεπιστημιο Δυτικης Αττικης

Σχολή Μηχανικών
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Διπλωματική εργασία

Θεμα

Ανάπτυξη πλατφόρμας FaaS (Function as a Service) σε περιβάλλον εικονικοποίησης βασισμένης σε περιέκτες

Τριμελής Εξεταστική Επιτροπή

Παναγιώτης Καρκαζής — Panagiotis Karkazis — Digitally signed by Panagiotis Karkazis Date: 2022.10.25 11:49:23 +03'00'

Ιωάννα Καντζάβελλου — Ioanna Kantzavelou — Digitally signed by Ioanna Kantzavelou Date: 2022.10.25 11:57:16 +03'00'

Βασίλειος Μάμαλης — Vasileios Mamalis — Digitally signed by Vasileios Mamalis Date: 2022.10.23 12:07:46 +03'00'

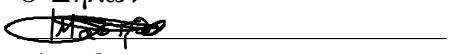
Δήλωση Συγγραφέα Πτυχιακής/Διπλωματικής Εργασίας

Ο κάτωθι υπογεγραμμένος Ιάκωβος Μαστρογιαννόπουλος του Γεωργίου, με αριθμό μητρώου 713242017102 φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής Μηχανικών του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών, δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της 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 Δηλών



Ιάκωβος
Μαστρογιαννόπουλος

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

Περίληψη

Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών
Σχολή Μηχανικών

Προπτυχιακών Σπουδών

Ανάπτυξη πλατφόρμας FaaS (Function as a Service) σε περιβάλλον εικονικοποίησης βασισμένης σε περιέκτες

από Ιάκωβος Μαστρογιαννόπουλος

Σε αυτή την διπλωματική εργασία, πραγματοποιήθηκε έρευνα πάνω στην υπολογιστική νέφους και σχεδιάστηκε μία πλατφόρμα «Συνάρτηση ως Υπηρεσία» Συγκεκριμένα, η πλατφόρμα σχεδιάστηκε με διάφορες μεθοδολογίες δίνοντας έμφαση στις ευέλικτες και στο CI/CD, δηλαδή στην συνεχές ενσωμάτωση/συνεχές παράδοση. Κύριος σκοπός της ήταν να μπορεί να εξυπηρετεί μεγάλο όγκο δεδομένων τα οποία είναι αποθηκευμένα σε μία NoSQL βάση δεδομένων και να μπορούν να χρησιμοποιηθούν από τους χρήστες της και να δημιουργηθούν νέες δικτυακές διεπαφές γράφοντας μόνο τις συναρτήσεις που χρειάζεται. Για την ανάπτυξη της πλατφόρμας, χρησιμοποιήθηκαν διάφορα εργαλεία όπως το Docker, η MongoDB, η ReactJS και η GO, ενώ ο κώδικας είναι διαθέσιμος σε μία ιστοσελίδα που παρέχει version control.

The following research concerns cloud computing and the creation of a platform; namely,'' Function as a Service''. In detail, the platform was designed using various methodologies. The emphasis is on agile and CI/CD, which means continuous integration/continuous delivery respectively. The main goal was to be able to serve a large quantity of data which are stored in a NoSQL database. Those data are to be used by the users to create new APIs using only the functions needed. A variety of tools were used to create the platform such as Docker, MongoDB, ReactJS and GO. Finally, the code is available in a version control website.

Ευχαριστίες

Ευχαριστώ πάρα πολύ τους κύριους Βασίλειο Μάμαλη και Απόστολο Αναγνωστοπούλο για την καταπληκτική τους συνεργασία κατά την διάρκεια της φοιτητικής μου σταυροδρομίας εφόσον υπήρξαν πηγή έμπνευσης και δημιούργησαν το υπάρχον ενδιαφέρον πάνω στον τομέα των κατανεμημένων συστημάτων. Ευχαριστώ επίσης όλους τους συμφοιτητές μου που προσπάθησαν να με βοηθήσουν να επιλέξω θέμα. Ευχαριστώ τον Σωτήρη Αναγνωστοπούλο που βοήθησε με όποιον τρόπο μπορούσε και ευχαριστώ και τον Παναγιώτη Φίσκιλη που βοήθησε πάρα πολύ στο τομέα της ασφάλειας. Ευχαριστώ πολύ την Μαρία Παπαδοπούλου και την Αμαλία Μαντή που βοήθησε στην επιμέλεια του κειμένου της αναφοράς.

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

Αυτή η διπλωματική αφιερώνεται πατέρα μου, Γιώργο και τον φίλο μου Παναγιώτη Μάλλιο

Κεφάλαιο 1

Θεωρητικό υπόβαθρο της Υπολογιστικής Νέφους

1.1 Κατανεμημένα Συστήματα (Distributed Systems)

Με την εκτεταμένη εμφάνιση του διαδικτύου (internet), δημιουργήθηκε η ανάγκη της διαχείρισης των δεδομένων. Στην πραγματικότητα, το διαδίκτυο δεν είναι τίποτα άλλο παρά ένα κατανεμημένο σύστημα. **Κατανεμημένο σύστημα (Distributed System)** ονομάζεται ένα ή περισσότερα υπολογιστικά συστήματα, τα οποία έχουν δημιουργηθεί με τέτοιον τρόπο ώστε να δουλεύουν παράλληλα και να αλληλοσυμπληρώνονται. Αυτό έχει σαν αποτέλεσμα ο χρήστης να θεωρεί πως η δουλεία γίνεται αποκλειστικά από έναν κόμβο στο δίκτυο.

"Κατανεμημένο σύστημα είναι μία συλλογή από ανεξάρτητους υπολογιστές οι οποίοι εμφανίζονται στους χρήστες ως ένα ενιαίο συνεκτικό σύστημα" (A.S. Tanenbaum, M. van Seen)[11]

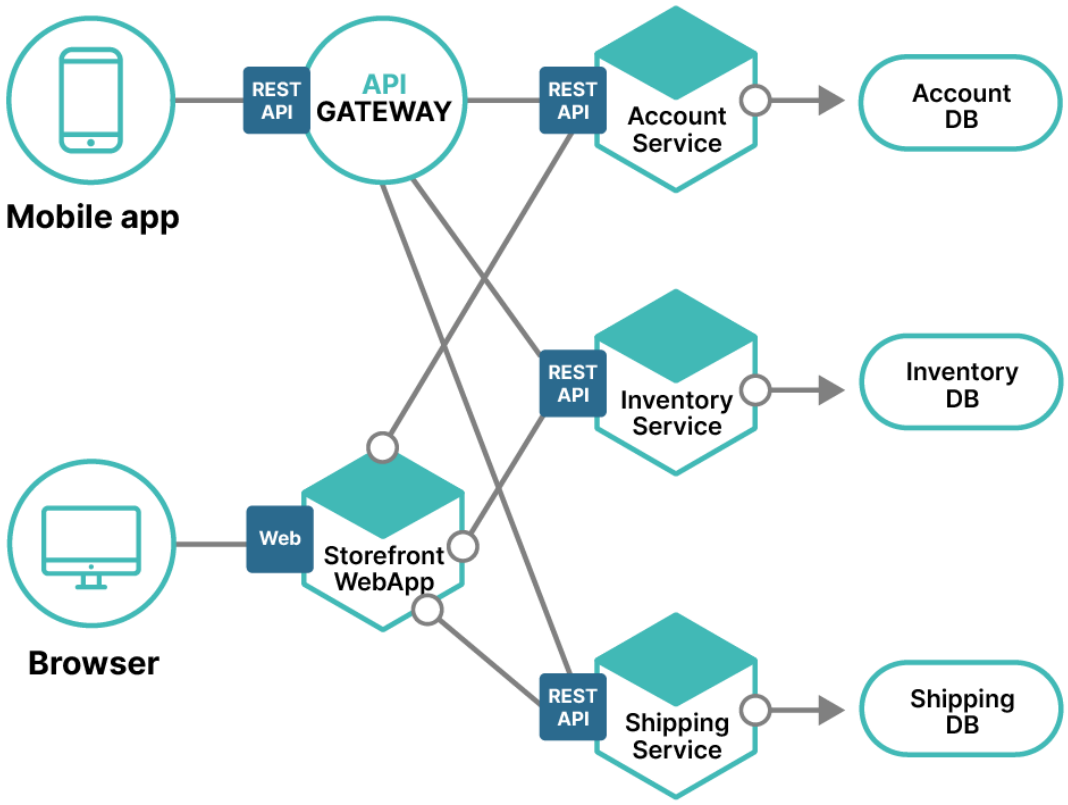
1.1.1 Πλεονεκτήματα των Κατανεμημένων Συστημάτων

Οι λόγοι που χρειάζεται να γίνετε εφαρμογή των κατανεμημένων συστημάτων είναι:[11]

- Καλύτερη κατανομή των πόρων με τους χρήστες.
- Καλύτερη διαφάνεια για το τι κάνει το κάθε υποσύστημα μέσα στο κεντρικό σύστημα.
- Ευκολότερη παροχή υπηρεσιών.
- Επεκτασιμότητα της εφαρμογής με την προσθήκη νέων συστημάτων.
- Εφαρμογή του Παράλληλου Υπολογισμού (Parallel Computing).
- Καλύτερη διαχείριση των δεδομένων.

1.1.2 Μειονέκτημα των Κατανεμημένων Συστημάτων

Ένα από τα μεγαλύτερα μειονεκτήματα των κατανεμημένων συστημάτων είναι το αυξημένο κόστος. Σε περίπτωση που χρειαστεί να γίνει μία προσθήκη στο σύστημα, θα πρέπει να αγοραστεί νέος εξοπλισμός. Αυτό σημαίνει ότι το κόστος της συντήρησης του συστήματος αυξάνεται και δυσκολεύει τη διαχείριση όχι μόνο των πόρων, αλλά και των πληροφοριών που δέχεται το σύστημα. Αυτό το πρόβλημα, έρχεται να λύσει η **Υπολογιστική Νέφους (Cloud Computing)**.



Σχημα 1.1: **Παράδειγμα Κατανεμημένου Συστήματος**: Στο συγκεκριμένο παράδειγμα, παρουσιάζεται ένα κατανεμημένο σύστημα το οποίο αποτελείται: από τρεις διακοσμιστές (servers), δύο πελάτες (clients) και τρεις βάσεις δεδομένων όπου το καθένα κάνει διαφορετική δουλεία. Αυτή η αρχιτεκτονική ονομάζεται μικρό-υπηρεσίες (micro-services) όπου η κάθε υπηρεσία κάνει μία δουλεία. (`https://orangematter.solarwinds.com/2022/01/24/what-is-a-distributed-system/`)

1.2 Ανάγκη των Cloud συστημάτων

Καθημερινά στο διαδίκτυο ανταλλάσσεται και καταγράφεται υπερβολικά μεγάλος όγκος δεδομένων. Στην επιστήμη της ανάλυσης των δεδομένων, η συλλογή μεγάλων όγκων δεδομένων ονομάζεται **Δεδομένα Μεγάλης Κλίμακας (Big Data)**. [23]

Σχημα 1.2: **Δεδομένα Μεγάλης Κλίμακας**: Λόγω των δεδομένων μεγάλης κλίμακας, θα πρέπει να λάβουμε υπόψιν πως αυτά τα δεδομένα θα αποθηκεύονται και θα παρουσιάζονται στον χρήστη. (`https://bleuwire.com/5-biggest-big-data-challenges/`)

Θα ήταν πολύ καταστροφικό για τις εταιρείες να αγοράζουν και να επεκτείνουν υπερβολικά πολύ το σύστημα τους. Το Cloud Computing έρχεται σαν λύση σε αυτό το πρόβλημα. Σκοπός του Cloud είναι να κρατήσει τα πλεονεκτήματα των κατανεμημένων συστημάτων, μειώνοντας το κόστος και κάνοντας την υλοποίηση τέτοιων συστημάτων πιο απλή.

Συγκεκριμένα, μία εταιρεία ή ένας οργανισμός παρέχει υπολογιστικούς πόρους σε μία δεύτερη εταιρεία ή οργανισμό με σκοπό να χρησιμοποιήσει αυτούς τους πόρους προς όφελος και των δύο. Η συντήρηση γίνεται αποκλειστικά από την εταιρεία ή τον οργανισμό που παραχωρεί τους πόρους. Έτσι, μία εταιρεία όπου στην εφαρμογή της γίνονται εναλλαγές πληροφοριών, θα γλυτώσει το κομμάτι της συντήρησης και θα μπορεί να επικεντρωθεί στην ίδια την εφαρμογή. Αυτό έχει σαν στόχο να υπάρχει πιο μεγάλη προσοχή στην ίδια την εφαρμογή για την εταιρεία, χωρίς να την ενδιαφέρουν οι υποδομές.

1.3 Χαρακτηριστικά του Cloud

1.3.1 Πολυχρηστικότητα και εύκολη χρήση

Αρχικά είναι σημαντικό να προσδιοριστεί ο τρόπος χρήσης της πλατφόρμας που παρέχει της υπηρεσίες. Επίσης είναι απαραίτητο να είναι εύκολη και κατανοητή η διαδικασία όπως επίσης και να υπάρχει η δυνατότητα να χρησιμοποιηθεί από διάφορες εφαρμογές και λειτουργικά συστήματα. [19][22][20]

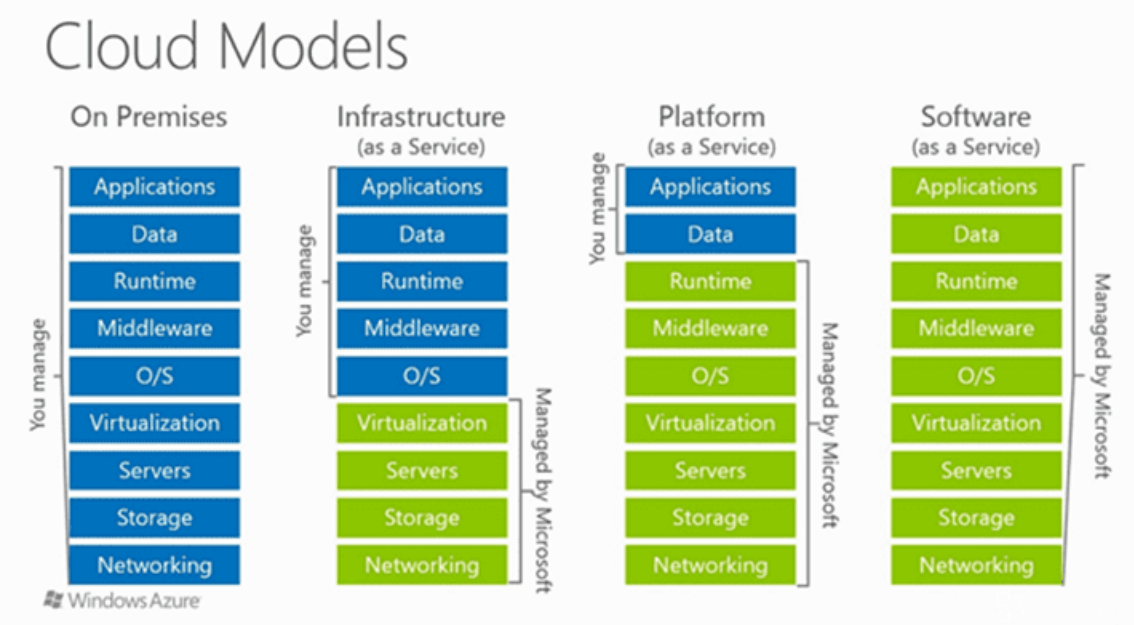
1.3.2 Ευρεία κλιμάκωση και ελαστικότητα

Το πλεονέκτημα των Cloud συστημάτων είναι η **κλιμάκωση (scale-up)**. Αυτό σημαίνει ότι μπορεί να υπάρχουν αντίγραφα της εφαρμογής σε διάφορα σημεία στον κόσμο, τα οποία είναι ενημερωμένα στην τελευταία έκδοση. Από την στιγμή που υπάρχει στο διαδίκτυο η εφαρμογή, δεν χρειάζεται η φυσική παρουσία του μηχανικού που το ελέγχει και το συντηρεί. Επιπλέον, ο μηχανικός πρέπει να παρέχει επιπλέον πόρους, σε περίπτωση που η εφαρμογή το χρειαστεί. [19][22][20]

1.3.3 Καθορισμός των πόρων και μετρούμενη πληρωμή

Κάθε εφαρμογή θα υπάρχει η δυνατότητα να είναι εξατομικευμένη. Αυτό σημαίνει ότι ο χρήστης που θα θέλει να χρησιμοποιήσει την πλατφόρμα θα πρέπει να γνωρίζει τι πόρους θα χρειαστεί, να τους ζητάει και να πληρώνει ανάλογα. [19][22][20]

1.4 Μοντέλα υπηρεσίας του Cloud



Σχημα 1.3: **Τα μοντέλα της Υπολογιστικής Νέφους**: Οι διαφορές μεταξύ των μοντέλων του Cloud. Κάθε μοντέλο παρουσιάζει έναν διαφορετικό βαθμό από αυτοματοποιημένες λειτουργίες. (`https://www.greycampus.com/cloud-computing-certifications/cloud-computing-models`)

Τα μοντέλα του Cloud ποικίλουν, ανάλογα με τις υπηρεσίες που έχουν σχεδιαστεί να αυτοματοποιούν. Οι πιο συχνές παρουσιάζονται στο Σχήμα 1.3.

- **Υποδομή ως Υπηρεσία (Infrastructure as a Service, IaaS)**
- **Πλατφόρμα ως Υπηρεσία (Platform as a Service, PaaS)**
- **Συνάρτηση ως Υπηρεσία (Function as a Service, FaaS)**

- **Λογισμικό ως Υπηρεσία (Software as a Service, SaaS)**

Κάθε μοντέλο προσπαθεί να λύσει το πρόβλημα του κλασσικού τρόπου (ευρέως γνωστό ως **σκέτο μέταλλο (bare metal)**), το οποίο αποτελείται από ένα υπολογιστικό σύστημα. Ο όρος bare metal χρησιμοποιείται παραπάνω για να ξεχωρίσει άλλα συστήματα τα οποία δεν χρησιμοποιούν το Cloud και τις υπηρεσίες τους. Η εταιρεία θα πρέπει να έχει πόρους και να διαχειριστεί αυτά τα δεδομένα ένας υπεύθυνος μηχανικός, ο οποίος ονομάζεται DevOps και πρέπει να φτιάξει τα πρωτόκολλα που θα επικοινωνούν μεταξύ τους. Αυτό αποφεύγεται με τις χρήσεις των μοντέλων, είτε με την χρήση **Εικονικών Μηχανών (Virtual Machines, VMs)** ή **Περιέκτες (Containers)** όπου έχουν έτοιμα πρωτόκολλα για την επικοινωνία των ξεχωριστών μερών τους. [19][22][20]

1.4.1 Υποδομή ως Υπηρεσία (Infrastructure as a Service, IaaS)

Υποδομή ως Υπηρεσία (Infrastructure as a Service, IaaS) είναι μία πλατφόρμα, η οποία διαμοιράζει υπολογιστικά συστήματα (πολλές φόρες VMs), όπου ο πελάτης μπορεί να χρησιμοποιήσει τις πλατφόρμες με σκοπό να του δίνει ένα σχεδόν ολοκληρωμένο σύστημα, στο οποίο θα πρέπει να αποφασίζει ένα λειτουργικό σύστημα, να γίνετε εγκατάσταση των απαιτήσεων που χρειάζεται, όπως και να ανεβάσει το πρόγραμμα του στο σύστημα. Γνωστά παραδείγματα του IaaS είναι το Google Cloud, το Microsoft Azure και ο Okeanos. [19][22][20]

1.4.2 Πλατφόρμα ως Υπηρεσία (Platform as a Service, PaaS)

Πλατφόρμα ως Υπηρεσία (Platform as a Service, PaaS) ονομάζεται μία πλατφόρμα, η οποία διαμοιράζει ένα έτοιμο περιβάλλον. Σε αυτό το περιβάλλον έχει εγκατεστημένα πακέτα και απαιτήσεις ώστε να εκτελεστεί ένα πρόγραμμα, για το όποιο υπεύθυνος για το ανέβασμα και την εκτέλεσή του είναι ο μηχανικός. Δεν χρειάζεται να ρυθμίσει κάτι ή να κάνει εγκατάσταση ο ίδιος. Ένα γνωστό τέτοιο παράδειγμα είναι το Swarmlab Hybrid. [19][22][20]

1.4.3 Λογισμικό ως Υπηρεσία (Software as a Service, SaaS)

Λογισμικό ως Υπηρεσία (Software as a Service, SaaS) ορίζεται μία πλατφόρμα η οποία παρέχει μία ολόκληρη εφαρμογή μέσω ενός προγράμματος περιήγησης (browser). Γνωστό παράδειγμα είναι της Cisco το WebEx που βοήθησε κατά την διάρκεια της πανδημίας Covid19 ώστε να γίνει εξ'αποστάσεως εκπαίδευση στα ελληνικά σχολεία και οι ηλεκτρονικές διευθύνσεις (e-mail). [19][22][20]

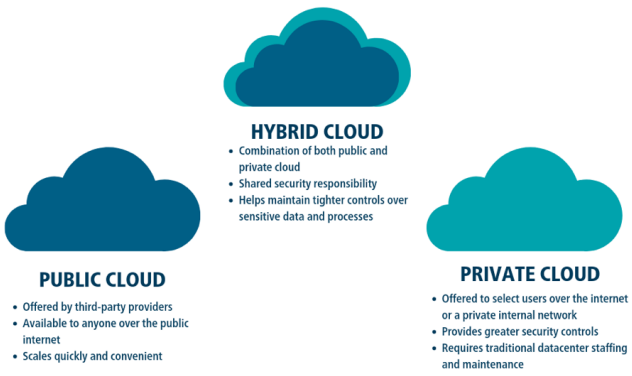
1.4.4 Επιπλέον μοντέλα της Υπολογιστικής Νέφους

Υπάρχουν ποικίλα μοντέλα του Cloud. Ένα από αυτά τα μοντέλα είναι το **Συνάρτηση ως Υπηρεσία (Function as a Service, FaaS)** το οποίο είναι νέο μοντέλο του cloud computing που πρωτοεμφανίστηκε το 2010 με σκοπό να παρέχει υπηρεσίες οι οποίες επιτρέπουν τον χρήστη να μπορεί να αναπτύξει κώδικα σε βάση μίας συνάρτησης. Ένα παράδειγμα του FaaS αποτελεί το AWS Lambda και αποτελεί το κεντρικό επίκεντρο αυτής της διπλωματικής εργασίας. Παραπάνω πληροφορίες, υπάρχουν στο Κεφάλαιο 3. [19][22][20]

Επιπλέον μοντέλα αποτελούν τα:

- **Αποθήκευση ως Υπηρεσία (Storage as a Service, STaaS)**: δουλεύουν πλατφόρμες όπως το Google Drive και το OneDrive. Σκοπός τους είναι η αποθήκευση αρχείων του χρήστη
- **Υλικό ως Υπηρεσία (Hardware as a Service, HaaS)**
- **Επικοινωνία ως Υπηρεσία (Communication as a Service, CaaS)**
- **Το οτιδήποτε ως Υπηρεσία (Anything as a Service, XaaS)**: είναι μία ειδική κατηγορία-ομπρέλα που ανήκουν διάφορα μοντέλα από κάτω του.
- **Desktop ως Υπηρεσία (Desktop as a Service, DaaS)**: παρέχει εικονικές μηχανές ως ενέργεια. Παράδειγμα: Windows 365

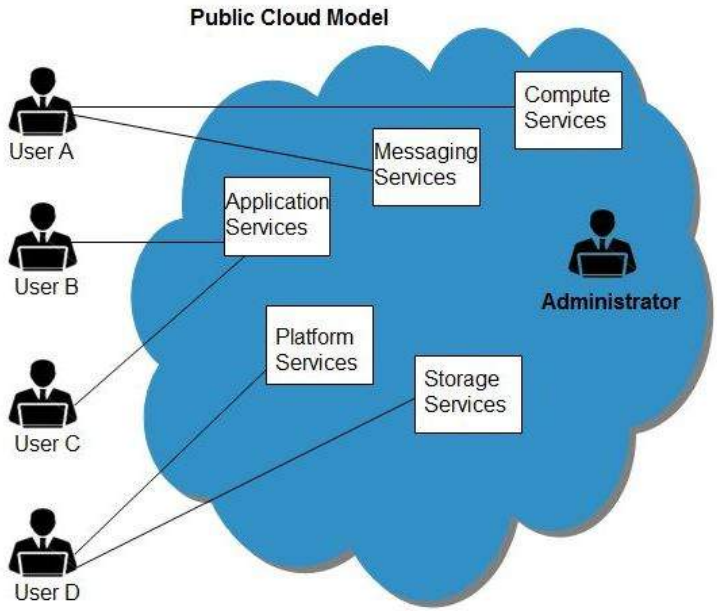
1.5 Μοντέλα Διανομής του Cloud



Σχημα 1.4: **Τα τρία κύρια μοντέλα διανομής της Υπολογιστικής Νέφους**: Αυτά τα μοντέλα έχουν παραπάνω σχέση με το πως και ποιος έχει πρόσβαση στις πλατφόρμες. (`https://karansinghreen.medium.com/what-is-the-difference-between-public-private-and-hybrid-cloud-a41bba631479`)

Εκτός από τα μοντέλα σε επίπεδο υπηρεσίας, υπάρχουν και τα μοντέλα διανομής του Cloud. Τα τρία κύρια μοντέλα διανομής είναι τα δημοσία, τα ιδιωτικά και τα υβριδικά, όπου το καθένα έχει τα πλεονεκτήματα και τα μειονεκτήματα τους. [22]

1.5.1 Δημόσιο Cloud



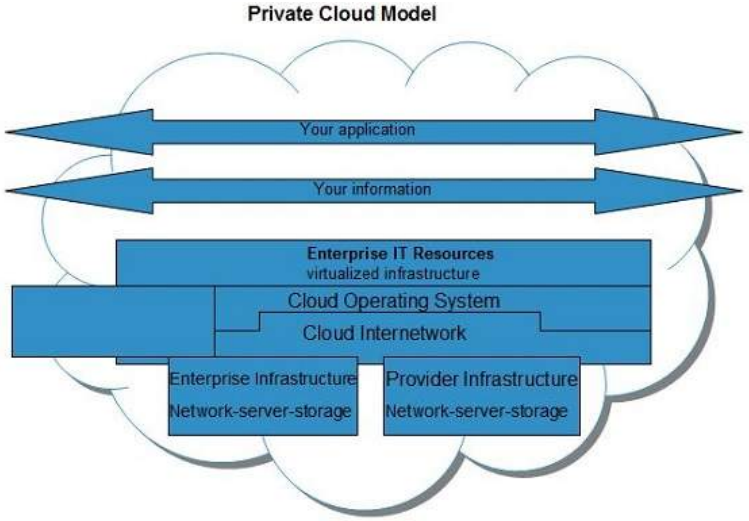
Σχημα 1.5: **Δημόσιο Μοντέλο Διανομής** (https://www.tutorialspoint.com/cloud_computing/cloud_computing_public_cloud_model.htm)

Το **Δημόσιο Cloud** είναι η πιο συχνή μορφή που συνηθίζεται να προσφέρεται στον τελικό χρήστη. Σε αυτή την περίπτωση, ο πάροχος κάνει διαθέσιμη την υπηρεσία του μέσω του διαδικτύου δημόσια και μπορεί ο οποιοσδήποτε να το χρησιμοποιήσει. Γνώστη περίπτωση χρήσης τέτοιων μοντέλων είναι οι δικτυακές εφαρμογές. [22]

Πλεονεκτήματα Μοντέλου	**Μειονεκτήματα Μοντέλου**
Υψηλή εμπορευσιμότητα	Μερικά θέματα ασφάλειας μπορεί να μην το επιτρέπουν
Μετρούμενη πληρωμή	Μπορεί να υπάρξουν νομικά προβλήματα
Δεν υπάρχει ανάγκη να υπάρξει συντήρηση στους υπολογιστές	Υπάρχει περιορισμένη πρόσβαση στους υπολογιστές
Χρειάζεται ελάχιστη τεχνική γνώση για να το χρησιμοποιήσει κάποιος χρήστης	Μερικές φόρες κάποιες επαγγελματικές απαιτήσεις δεν μπορούν να καλυφθούν
Οι υπηρεσίες είναι ανοιχτές για όλους	

Πινακας 1.1: Πλεονεκτήματα και μειονεκτήματα του Δημοσίου Μοντέλου

1.5.2 Ιδιωτικό Cloud



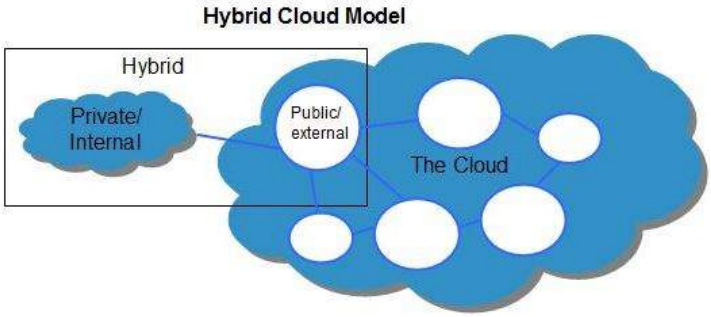
Σχημα 1.6: **Ιδιωτικό Μοντέλο Διανομής** (https://www.tutorialspoint.com/cloud_computing/cloud_computing_private_cloud_model.htm)

Το **Ιδιωτικό Cloud** αναφέρεται σε συστήματα τα οποία ανήκουν σε έναν οργανισμό και μόνο αυτός ο οργανισμός μπορεί να τα χρησιμοποιήσει πλήρως. Μία περίπτωση χρήσης τέτοιου συστήματος είναι για συστήματα με ευαίσθητα δεδομένα τα οποία δεν μπορούν να είναι δημόσια. [22]

Πλεονεκτήματα Μοντέλου	**Μειονεκτήματα Μοντέλου**
Μπορούν να εφαρμοστούν στις ανάγκες της εφαρμογής και να τις υποστηρίξουν όσο παλιές και να είναι	Χρειάζεται να γίνουν αγορές νέων υπολογιστικών συστημάτων όπως και να γίνεται συχνή συντήρηση σε αυτά
Υπάρχει πλήρης έλεγχος στην ασφάλεια	Χρειάζεται να υπάρχουν τεχνικές γνώσεις για να χρησιμοποιηθεί ένα τέτοιο σύστημα
Μπορεί να εκπληρώσει οποιασδήποτε νομική ανάγκη	Δεν είναι εμπορεύσιμο προϊόν

Πινακας 1.2: Πλεονεκτήματα και μειονεκτήματα του Ιδιωτικού Μοντέλου

1.5.3 Υβριδικό Cloud



Σχημα 1.7: **Υβριδικό Μοντέλο Διανομής** (https://www.tutorialspoint.com/cloud_computing/cloud_computing_hybrid_cloud_model.htm)

Το **Υβριδικό Cloud** είναι ένας συνδυασμός των Δημοσίων και των Ιδιωτικών Cloud προσπαθώντας να πάρει τα πλεονεκτήματα των δύο τους. Για παράδειγμα, θα μπορούσε να έχει δημόσια την εφαρμογή, ένω τα ευαίσθητα δεδομένα να είναι κρυμμένα σε μία ιδιωτική βάση. [22]

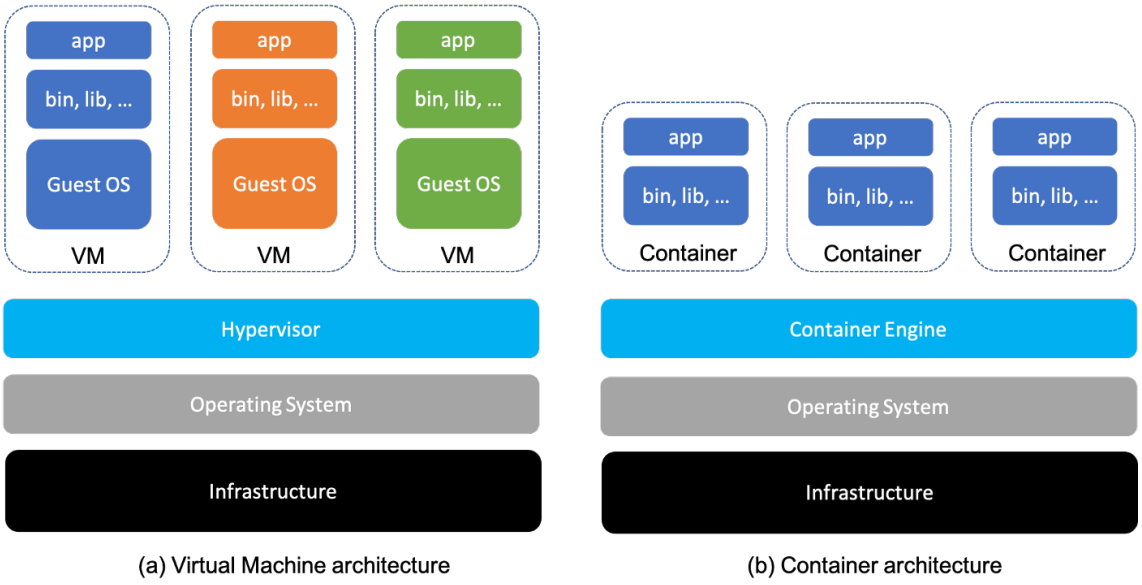
Πλεονεκτήματα Μοντέλου	**Μειονεκτήματα Μοντέλου**
Μπορεί να αφήνει εκτός συστήματα τα οποία είναι παλαιωμένα είτε από πλευράς υλικού είτε από πλευράς λειτουργικού συστήματος	Είναι πιο περίπλοκο να δημιουργηθεί και να διαχειριστεί
Είναι πιο εύκαμπτο και πιο εμπορεύσιμο συγκριτικά με τα δημόσια συστήματα	Είναι πιο ακριβό από το να χρησιμοποιείται μόνο μία τεχνική
Κληρονομεί τα πλεονεκτήματα των οικονομικών του δημοσίου Cloud	
Μπορούν να χρησιμοποιηθούν εσωτερικά συστήματα σε περίπτωση ευαίσθητων δεδομένων	

Πινακας 1.3: Πλεονεκτήματα και μειονεκτήματα του Υβριδικού Μοντέλου

Κεφάλαιο 2

Εργαλεία και Μεθοδολογίες της Υπολογιστικής Νέφους

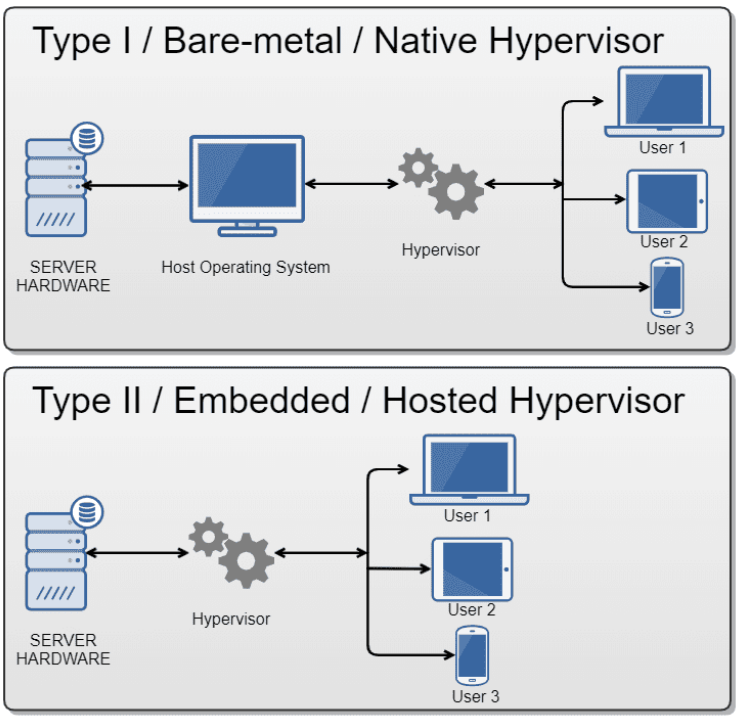
2.1 Εικονικοποίηση (Virtualization)



Σχημα 2.1: **Διάγραμμα εικονικοποιήσης**: Στην εικόνα, παρουσιάζονται οι δύο μορφές στην οποία γίνεται εικονικοποιήση των συστημάτων. Όπως φαίνεται, υπάρχει διαφορές μεταξύ των εικονικών μηχανών και των περιεκτών αλλά η ιδέα είναι ίδια. (`https://towardsdatascience.com/virtualization-for-machine-learning-da11b7a59070`)

Για να μπορούν να εφαρμοστούν αυτά τα μοντέλα του Cloud, θα πρέπει κάπως να μπορούν να μαζευτούν σε ένα πακέτο το οποίο θα περνάει εύκολα από ένα υπολογιστικό σύστημα σε ένα άλλο σύστημα. Αυτό μπορεί να επιταχυνθεί με την **εικονικοποίηση (virtualization)** του συστήματος του ίδιου, μέσω της οποίας μπορούν να χωριστούν το κάθε ξεχωριστό μέρος του σε ένα διαφορετικό και ανεξάρτητο VM. Διάσημα εργαλεία που υλοποιούν VM περιβάλλοντα είναι το VirtualBox το οποίο είναι ανοιχτού κώδικα (open source) και ανήκει στο GNU, το VMWare Workstation και το QEMU, το οποίο είναι επίσης open source. [22][20]

2.1.1 Επόπτες (Hypervisors)



Σχημα 2.2: **Είδη εποπτών (hypervisors)** (`https://www.nutanix.com/uk/info/hypervisor`)

Τα VMs για να λειτουργήσουν χρειάζονται έναν **επόπτη (hypervisor)**. Ο hypervisor είναι ένας προσομοιωτής, ο οποίος δημιουργεί και τρέχει VMs. Με αλλά λόγια, επιτρέπει σε έναν υπολογιστή να μπορεί να υποστηρίξει πολλαπλά VMs ταυτόχρονα, μοιράζοντας τους πόρους του συστήματος. Οι hypervisors κάνουν την αλλαγή από το ένα σύστημα σε ένα άλλο πολύ πιο απλή, λόγο ότι μπορούν να τρέχουν πολλαπλά VMs μέσα σε ένα σύστημα, μειώνοντας τον φυσικό χώρο, την ενέργεια και τα κόστη της συντήρησης που χρειάζεται ένα distributed σύστημα. [22]

Είδη των εποπτών (hypervisors)

- **Τύπος 1 (Type-1) Hypervisor**: πραγματοποιείται απευθείας αλληλεπίδραση με το προσομοιασμένο υλικό, φορτώνει πριν από το OS και είναι ανεξάρτητο του. Είναι επίσης γνωστό ως bare metal hypervisor λόγω του ότι κάνει full virtualization το σύστημα.
- **Τύπος 2 (Type-2) Hypervisor**: η λειτουργία του γίνετε πάνω από το OS. Αυτό έχει το αρνητικό ότι σε περίπτωση που το host OS κολλήσει, τότε θα κολλήσει και το VM μαζί του.

Είδη εικονικοποιήσης

- **Πλήρης εικονικοποίηση (full virtualization)**: γίνετε προσομοίωση του πραγματικού υλικού, χωρίς να έχει αλλάξει τίποτα. Αυτό σημαίνει ότι ολή η υπολογιστική ισχύς του συστήματος παραμένει όπως είναι. Υλοποιείται με μία μετάφραση των δυαδικών αριθμών (binary translation) και ξαναγράφει όλες τις εντολές της αρχιτεκτονικής του VM. Το Hyper-V της Microsoft είναι ένας hypervisor ο οποίος κάνει full virtualization το σύστημα.

- **Υποβοηθούμενη από το λειτουργικό σύστημα (paravirtualization)**: στα φιλοξενούμενα λειτουργικά συστήματα δεν γίνετε προσομοίωση του φυσικού υλικού. Παρόλα αυτά, έχουν ένα δικό τους απομονωμένο περιβάλλον, εκτελούνται σαν ένα ξεχωριστό σύστημα και έχουν τροποποιηθεί κατάλληλα ώστε να μπορεί να τρέξει ένα τέτοιο περιβάλλον. Οι κρίσιμες εντολές πραγματοποιούνται με ειδικές μεταφράσεις μέσω του hypervisor και ονομάζονται hyper-calls. Ένα παράδειγμα στην αγορά που χρησιμοποιεί έναν hypervisor είναι το Windows Subsystem for Linux (WSL) της Microsoft, το οποίο κάνει προσομοίωση ενός Linux συστήματος που τρέχει πάνω στο περιβάλλον των Windows.

- **Υποβοηθούμενη από το υλικό (hardware assisted)**: γίνετε χρήση του υλικού με ειδικά κατασκευασμένες κεντρικές μονάδες επεξεργασίας (cental processing power, CPU) και υλικού στο οποίο εκτελείτε το φιλοξενούμενο λειτουργικό σύστημα. Οι δύο κυρίαρχες τεχνολογίες είναι της Intel το VT-X και της AMD το AMD-V, το οποίο χρησιμοποιεί το KVM του QEMU για να μπορεί να αξιοποιήσει τρομερές επιδόσεις στα VMs που τρέχει.

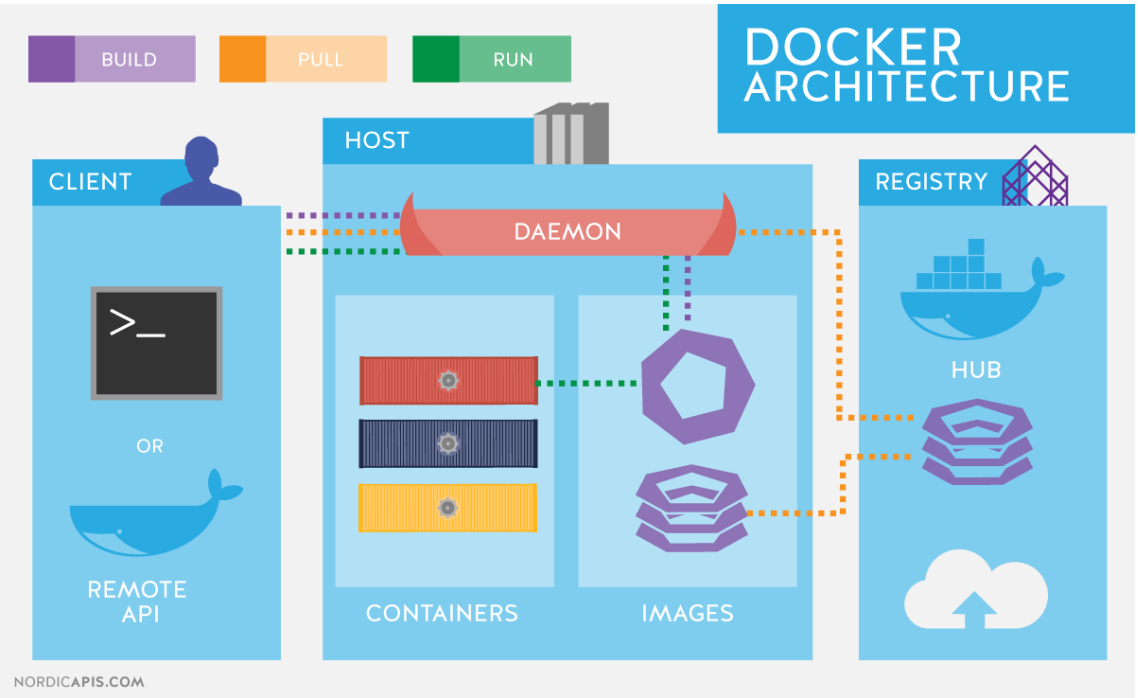
2.2 Περιέκτες (Containers)

Μία νέα μέθοδος virtualization είναι με την χρήση **περιεκτών (containers)**. Κάθε container είναι ένα πολύ μικρό VM το οποίο προσομοιώνει ένα λειτουργικό σύστημα και είναι ειδικά τροποποιημένο ώστε να είναι ελαφρύ στην μνήμη και μικρό στην χωρητικότητα. Η πιο διάσημη τεχνολογία που **δημιουργεί (build)** και **παρατάσσει (deploy)** containers είναι το Docker.

Το Docker είναι ένα σύνολο από PaaS προϊόντα, τα οποία χρησιμοποιούν paravirtualization τεχνικές για να μπορούν να εκτελέσουν containers στον χρήστη. Κάθε container είναι ανεξάρτητο από το άλλο και έρχεται μαζί με το δικό του λογισμικό, τις δικές του βιβλιοθήκες και τα δικά του αρχεία σύνθεσης (configuration).

Ένα container μπορεί να γίνει deploy μαζί με ένα άλλο και να αλληλεπιδρούν μεταξύ τους. Το υπεύθυνο εργαλείο που το κάνει αυτό είναι ο **ενορχηστρωτής (orchestrator)**. Υπάρχουν διάφοροι orchestrators στην αγορά, με τους δύο πιο διάσημους να είναι του Docker το Docker Swarm και το Kubernetes. Δουλεία τους είναι να δημιουργούν ένα έτοιμο εικονικό δίκτυο στο οποίο μπορούν να αλληλεπιδρούν μεταξύ τους τα VMs, δημιουργώντας ένα πλέγμα (cluster) μεταξύ τους. [19][20][8]

2.2.1 Τεχνικά χαρακτηριστικά του Docker



Σχημα 2.3: **Η αρχιτεκτονική του Docker** (`https://nordicapis.com/api-driven-devops-spotlight-on-docker/`)

- Το κομμάτι του λογισμικού το οποίο τρέχει ένα δαίμων (daemon), αποκαλούμενο ως dockerd, το οποίο είναι υπεύθυνο για την διαχείριση των containers, τα οποία τα ελέγχει ως αντικείμενα (objects). Το daemon ακούει για αιτήσεις και μέσω της docker εντολής που τρέχει στο terminal (command-line interface) επιτρέπει τον χρήστη να μπορεί να το διαχειριστεί.
- Το κομμάτι των αντικειμένων το οποίο είναι ένα σύνολο από εικόνες (images), containers και υπηρεσίες (services).
  - Ένα **container** είναι ένα περιβάλλων το οποίο τρέχει μία ή περισσότερες εφαρμογές.
  - Ένα **image** είναι ένα πρότυπο το οποίο χρησιμοποιείτε για να κάνει build νέα containers. Τα images υπάρχουν στην ιστοσελίδα του **Docker Hub**.
  - Ένα **service** επιτρέπει τα containers να μπορούν να κάνουν scale-up σε πολλαπλά Docker daemons και να γίνετε αυτόματα ενημέρωση σε όλα τα containers στην τελευταία έκδοση τους.
- Ένα σύστημα από εγγραφές το οποίο επιτρέπει στο τοπικό Docker daemon να κατεβάζει (pull) και να ανεβάζει (push) νέα images τα οποία έχουν γίνει built. Αυτά τα registries μπορεί να είναι είτε δημόσια είτε ιδιωτικά. [8]

2.2.2 Εργαλεία του Docker

Docker Compose

Το **Docker Compose** είναι ένα εργαλείο το οποίο επιτρέπει στο να γίνετε αυτοματοποιημένο το deploy πολλαπλών containers σε Docker. Χρησιμοποιεί YAML αρχεία για να κάνει το configuration και να ξέρει ποια services πρέπει να κάνει build και να σηκώσει με την χρήση μίας εντολής.

```yaml
version: '3.9'

services:
  rest_api:
    build: app
    container_name: rest_api
    host_name: users

  database:
    image: mongo
    container_name: mongo
    host_name: mongo
    environment:
      MONGO_INITDB_ROOT_USERNAME: dbuser
      MONGO_INITDB_ROOT_PASSWORD: dbpass
      MONGO_INITDB_DATABASE: cars
```

Στο συγκεκριμένο παράδειγμα, ανεβαίνουν αυτόματα δύο containers, το πρώτο κάνει build τον κατάλογο app ενώ το δεύτερο παίρνει το έτοιμο image της mongo από το Docker Hub και χρησιμοποιεί τα environment για να ορίσει τα ονόματα του χρήστη, των κωδικό και της βάσης. [8]

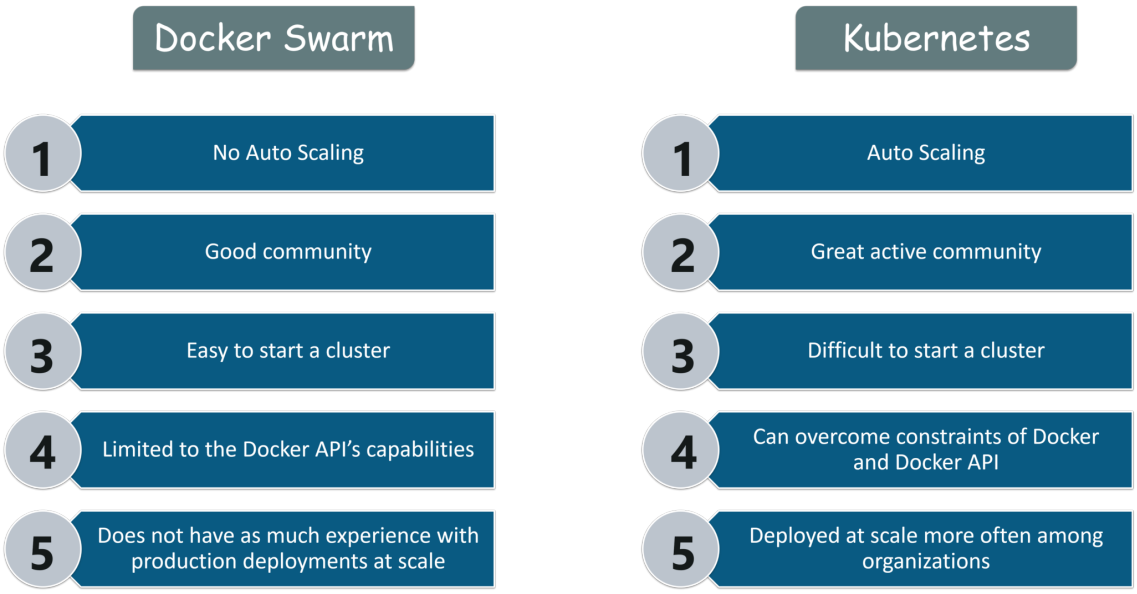
Docker Swarm

Το **Docker Swarm** είναι ένας orchestrator του Docker που το βάζει σε cluster mode. Δηλαδή, επιτρέπει να μπορεί να διαμοιράσει τα containers και τα εικονικά του δίκτυα σε δύο ή περισσότερα υπολογιστικά συστήματα. Σε κάθε Swarm, υπάρχουν δύο ειδών nodes:

- Τα **Master** που είναι υπεύθυνα για την οργάνωση των άλλων node. Υπάρχει τουλάχιστον ένας ανά Swarm.
- Οι **Workers** όπου είναι τα υπόλοιπα μέλη των node.

Με την χρήση των Cloud συστημάτων και του Docker Swarm, τα nodes θα μπορούσαν να βρίσκονται σε οποιονδήποτε σημείο στον κόσμο και από διαφορετικό πάροχο. Για παράδειγμα, η φυσική παρουσία του μηχανικού είναι η Αθήνα, το master του δικτύου βρίσκεται στο Λονδίνο και υπάρχουν δύο workers, ένας στο Σικάγο και το άλλο στο Τόκιο. Με μία απλή ενημέρωση και χρήση ενός git συστήματος, ο μηχανικός ανεβάζει τον κώδικα εκεί και οι ίδιες οι πλατφόρμες είναι έτσι φτιαγμένες ώστε να μπορούν να πάρουν τις νέες εκδόσεις χωρίς να χρειαστεί να υπάρξει ανθρώπινη παρέμβαση, κάνοντας το αυτόματο.

Υπάρχει ένας εναλλακτικός orchestrator, το Kubernetes. Το Kubernetes είναι επίσης αρκετά καλό, με δικά του χαρακτηριστικά.



Σχημα 2.4: Οι διαφορές μεταξύ Kubernetes και Docker Swarm. (https://www.cc-radio.com/docker-compose-vs-kubernetes)

Με βάση το Σχήμα 2.4, οι διαφορές μεταξύ τους είναι αρκετά μεγάλες. Με απλά λόγια:

Docker Swarm	**Kubernetes**
Δεν υπάρχει αυτοματοποιημένο scaling σύστημα, αφήνοντας πλήρης ελέγχο στον χρήστη	Υπάρχει αυτοματοποιημένο scaling σύστημα, το οποίο μπορεί να βοηθήσει στην παραγωγικότητα της εταιρείας
Υπάρχει αρκετά μεγάλη κοινότητα που το υποστηρίζει και το δουλεύει	Υπάρχει επίσης αρκετά μεγάλη κοινότητα η οποία είναι πιο ενεργοί
Είναι αρκετά απλό στην ρυθμίσει του για την δημιουργία ενός νέου cluster	Είναι αρκετά δυσκολότερο να δημιουργηθεί ένα νέο cluster, όποτε είναι αρκετά δύσκολο για έναν αρχάριο
Είναι περιορισμένο στις δυνατότητες του Docker	Δεν είναι περιορισμένο στις δυνατότητες του docker και μπορούν να χρησιμοποιηθούν και άλλες τεχνολογίες

Πινακας 2.1: Διαφορές μεταξύ Kubernetes και Docker Swarm

Μπορούν βέβαια να συνδυαστούν οι τεχνολογίες και χρησιμοποιηθούν μεταξύ τους για το βέλτιστο αποτέλεσμα. Και οι δύο τεχνολογίες είναι πολύ καλές και χρήσιμες σε παραγωγή και θα πρέπει να γίνει η κατάλληλη ανάλυση για το πιο θεωρείται πιο χρήσιμο για την περίπτωση χρήσης που χρειάζεται. [8]

Docker Volume

Το Docker Volume είναι μία λειτουργία του Docker που επιτρέπει να δεσμεύσει (bind) έναν χώρο του τοπικού συστήματος. Έτσι, ο προγραμματιστής θα μπορεί να γράφει κώδικα και να κάνει αλλαγές χωρίς να πρέπει να κάνει επανεκκίνηση της εφαρμογής. Παρόλο που μπορεί να γίνει και κανονικό bind στο

τοπικό σύστημα του Docker, το Docker Volume δίνει μερικά πλεονεκτήματα και κάνει την διαχείριση τους αυτόματα. Τα πλεονεκτήματα αυτά είναι τα εξής:

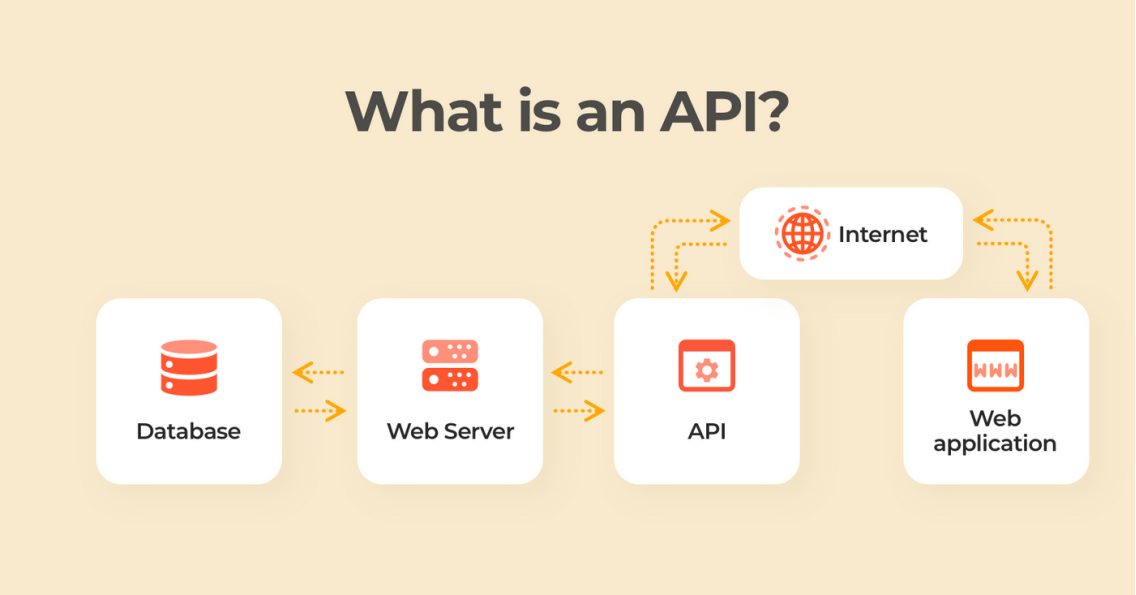
- Τα Volumes είναι αρκετά ευκολότερα ώστε να γίνονται back ups ή να γίνετε αποδήμηση (migration).
- Είναι πολύ εύκολο να γίνει η διαχείριση του με το Docker CLI.
- Τα Volumes δουλεύουν σε Linux και σε Windows.
- Τα Volumes είναι πολύ ασφαλές να διαμοιράσουν τα δεδομένα μεταξύ πολλαπλών containers.
- Οι drivers των Volumes μπορεί να βρίσκονται σε απομακρυσμένους υπολογιστές ή σε άλλες Cloud υπηρεσίες, να γίνεται κρυπτογράφηση των περιεχομένων των containers ή να προστεθεί νέες λειτουργίες.
- Τα Volumes μπορούν να έχουν αντικείμενα και από προηγούμενα containers.
- Επιπλέον, αρκετές φορές είναι πολύ προτιμότερο να μπορούν να κρατηθούν τα δεδομένα, χωρίς να μεγαλώνει το μέγεθος των containers και να μένουν πίσω πληροφορίες για να μπορούν να χρησιμοποιηθούν αργότερα. [8]

2.3 Διαχείριση των δεδομένων

Η διαχείριση των δεδομένων είναι ένα αρκετά μεγάλο και σημαντικό κομμάτι στην ανάπτυξη Cloud εφαρμογών. Στο σύστημα, θα πρέπει να εφαρμοστούν τα χαρακτηριστικά και τα πλεονεκτήματα του Cloud, όπως εμφανίζονται στο Κεφάλαιο 1.3, με την συνεργασία του Docker. Στην διαχείριση δεδομένων, γίνεται εφαρμογή πολλών διαφορετικών τομέων της πληροφορικής, όπως ο δικτυακός προγραμματισμός, οι βάσεις δεδομένων και η ανάλυση πληροφοριών. [19][23]

2.3.1 Διεπαφή Προγραμματισμού Εφαρμογών (Application Programming Interface, API)

Η **Διεπαφή Προγραμματισμού Εφαρμογών (Application Programming Interface, API)** είναι μία σύνδεση μεταξύ υπολογιστών ή προγραμμάτων και συνήθως αναφέρεται σε δικτυακά APIs. Δεν σχεδιάζεται ποτέ να μπορεί να το χρησιμοποιήσει ο τελικός χρήστης, αφού βασικός του σκοπός είναι να δημιουργεί εργαλεία ή υπηρεσίες που θα μπορεί να χρησιμοποιήσει ένα άλλο τμήμα του κώδικα. Ένα API είναι χωρισμένο σε επιμέρους κομμάτια, όπως οι μέθοδοι (methods), τα αιτήματα (requests), οι υπορουτίνες (subroutines) και τα άκρα (endpoints). Ένας άλλος σκοπός του API είναι να κρύψει στον τελικό χρήστη πως δουλεύει από πίσω το σύστημα, δείχνοντας μόνο ό,τι επιθυμεί ο προγραμματιστής του. Γνώστες υλοποιήσεις του API είναι το SOAP και το REST.

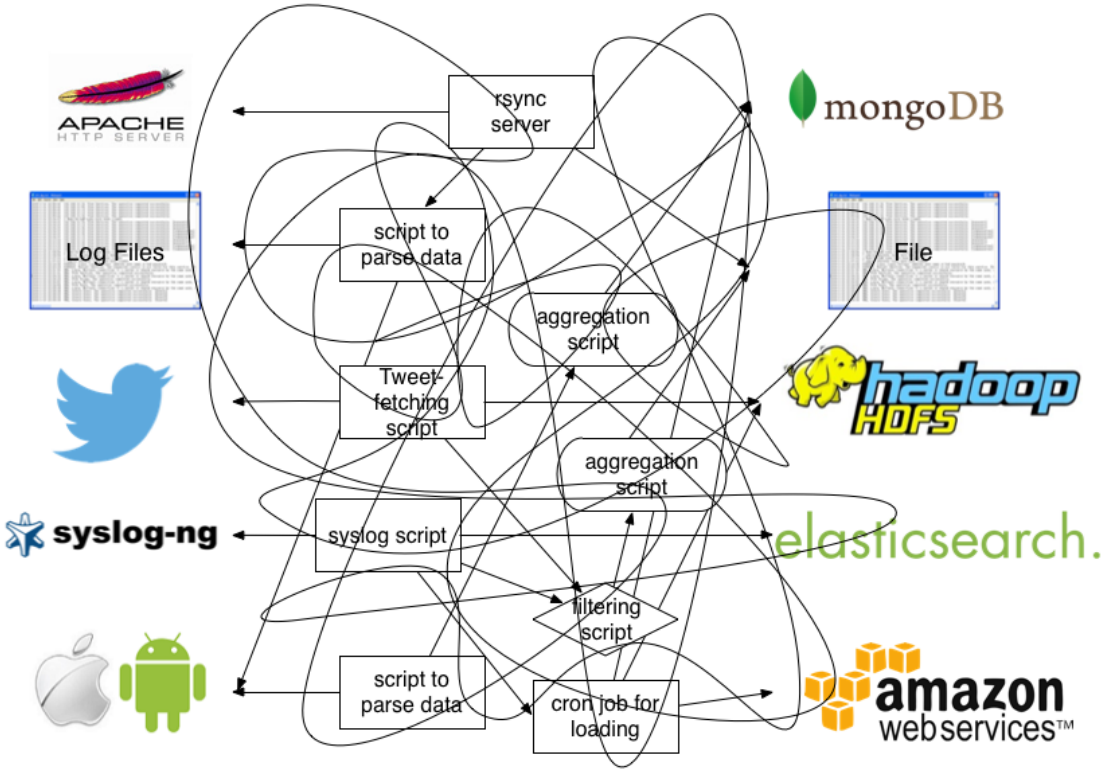


Σχημα 2.5: **Διάγραμμα API** (https://www.cleveroad.com/blog/what-is-an-api/)

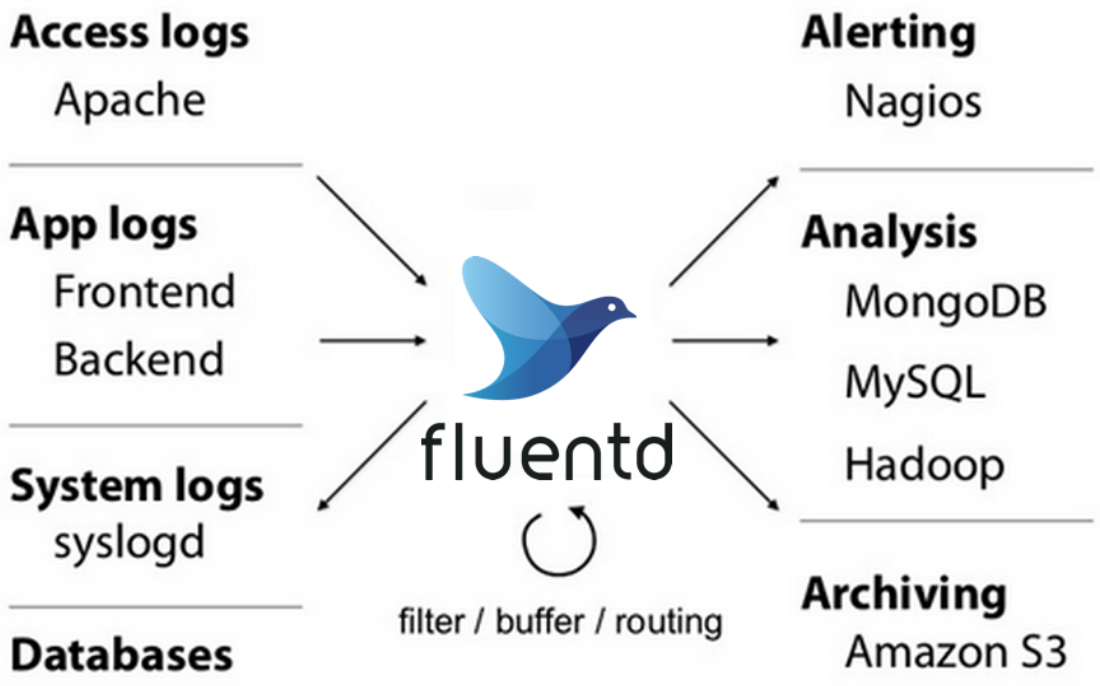
Το REST API, το οποίο σημαίνει **Αφηρημένη Φάση Μεταφοράς (Representational State Transfer)** επιτρέπει να υπάρχει κλιμάκωση στην αλληλεπίδραση μεταξύ των τμημάτων (components), στους μετασχηματισμούς διεπαφής και στην δημιουργία μίας αφηρημένης αρχιτεκτονικής για να μειώσει καθυστέρηση μεταξύ των κόμβων και ενισχύοντας την ασφάλεια. Η επικοινωνία γίνετε με HTTP requests, όπως το GET και το POST, ενώ η μεταφορά των δεδομένων πολύ συχνά γίνετε με JSON ή με XML. Η υλοποίηση μπορεί να γίνει με πολλές γλώσσες και framework. [16]

2.3.2 Συλλέκτες Πληροφοριών (Data Collectors)

Οι **Συλλέκτες Πληροφοριών (Data Collectors)** είναι μικρές εφαρμογές που μπορούν να εγκατασταθούν σε ένα Docker Swarm cluster για να συλλέγει μεταδεδομένα (metadata), δηλαδή πληροφορίες σχετικά με το σύστημα, όπως το configuration του συστήματος σε μία Βάση Δεδομένων ή σε ένα σημείο του συστήματος. Μερικά σημαντικά Data Collectors είναι το Kafka της Apache, το Logstash της Elastic και το FluentD που είναι ανοιχτού κώδικα. Το καθένα έχει τις δυνατότητες του και τις αδυναμίες του και είναι πολύ σημαντικό να βρεθεί το σωστό εργαλείο για την σωστή την δουλεία. Στα Σχήματα 2.6 και 2.6 παρουσιάζεται ο λόγος για τον οποίο είναι σκόπιμο να περιλαμβάνονται τέτοια εργαλεία στα εργαλεία που χρησιμοποιεί μία εταιρεία.



Σχημα 2.6: **Παράδειγμα συστήματος χωρίς συλλεκτή δεδομένων** (https://docs.fluentd.org/)



Σχημα 2.7: **Παράδειγμα συστήματος με συλλεκτή δεδομένων** (https://docs.fluentd.org/)

Στις εικόνες 2.6 και 2.7 ότι χωρίς το FluentD επικρατεί ένα χάος στην μεταφορά των πληροφοριών, ένω όταν υπάρχει τα οργανώνει όλα, έτσι ο προγραμματιστής είναι υπεύθυνος να γράψει μόνο τον κώδικα. [3][23][20]

2.3.3 Ανάλυση των πληροφοριών (Data Analysis)

Η **Ανάλυση των Πληροφοριών (Data Analysis)** είναι ένας κλάδος της πληροφορικής και της στατιστικής ο οποίος ασχολείται με την επεξεργασία των δεδομένων με σκοπό να βρεθεί κάποια πληροφορία που μπορεί να χρησιμοποιηθεί με σκοπό να δημιουργηθεί μία νέα ανακάλυψη που θα βοηθήσει να εντοπιστούν προβλήματα ή στο να δημιουργηθούν νέες αποφασίσεις. Υπάρχουν πολλές διαφορετικές μεθοδολογίες για να επιταχυνθεί το data analysis με τη πιο διάσημη και ευανάγνωστη μέθοδο να είναι η **Εξόρυξη Δεδομένων (Data Mining)**. Στον τομέα του Cloud Computing, αυτό που απασχολεί έναν μηχανικό είναι εφόσον γίνει η συλλογή δεδομένων από τους data collectors να υπάρχει ένα API που μπορεί με δυναμικό τρόπο να επεξεργάζεται τα δεδομένα στιγμιαία. Υπάρχουν διάφορες τεχνολογίες που μπορούν να επιτυγχάνουν να δημιουργηθεί ένα τέτοιο περιβάλλον. [23][22]

Apache Hadoop

Το Hadoop είναι ένα framework χρήσιμο για την ανάπτυξη κατανεμημένης επεξεργασίας μέσα από ένα cluster υπολογιστών χρησιμοποιώντας απλά προγραμματιστικά μοντέλα. Έχει σχεδιαστεί με τέτοιον τρόπο ώστε να μπορεί να προσαρμόζεται να δουλεύει είτε από έναν server είτε σε χιλιάδες.

Ένα κλασσικό παράδειγμα είναι το WCount, το όποιο μετράει πόσες φόρες εμφανίζεται κάθε όρος μέσα στα έγγραφα. Το MapReduce είναι χωρισμένο σε τρία τμήματα κώδικα. Στην κλάση του mapper, στην κλάση του reducer και τέλος σε έναν runner. Αυτό το μοντέλο ονομάζεται MapReducer. Υλοποιείται μόνο στην γλώσσα προγραμματισμού Java

Το πρώτο στάδιο του hadoop είναι πάντα το Mapper. Το Mapper ουσιαστικά αντιστοιχεί keywords με values, όπως και άλλες υλοποίησης της δομής Map (για παράδειγμα το HashMap της Java, το Dictionary της Python και το JSON της Javascript). Σε αυτή την περίπτωση, αντιστοιχισμένες τιμές είναι ένα Text με ένα IntWritable. Και τα δύο είναι classes του Hadoop. [23][22]

```
public class TFMapper extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    @Override
    public void map(LongWritable longWritable,
                    Text text,
                    OutputCollector<Text, IntWritable> outputCollector,
                    Reporter reporter) throws IOException {
        var line = text.toString();
        var tokenizer = new StringTokenizer(line);

        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            outputCollector.collect(word, one);
        }
```

```
    }
}
```

Το δεύτερο στάδιο του Hadoop είναι το Reducer. Το Reducer διαβάζει κάθε key αυτόματα και παίρνει τις τιμές του, κάνει κάποιους υπολογίσιμους και γράφει στο output αρχείο το τελικό αποτέλεσμα. Στο παράδειγμα του wcount, μετράει τις λέξεις που έγιναν map. Δηλαδή εάν η λέξη "up" εμφανίζεται τρεις φόρες στο κείμενο, θα γράψει στο output "up 3".

```
public class TFReducer extends MapReduceBase
        implements Reducer<Text, IntWritable, Text, IntWritable> {
    @Override
    public void reduce(Text text,
                       Iterator<IntWritable> iterator,
                       OutputCollector<Text, IntWritable> outputCollector,
                       Reporter reporter) throws IOException {
        var sum = 0;

        while (iterator.hasNext()) {
            sum += iterator.next().get();
        }

        outputCollector.collect(text, new IntWritable(sum));
    }
}
```

Ο runner δεν είναι ακριβώς στάδιο του Hadoop, άλλα πιο πολύ μία configuration κλάση, που δηλώνει ο προγραμματιστής στο MapReduce ποιες κλάσεις να χρησιμοποιηθούν, ποιος ο τρόπος εγγραφής στο τελικό έγγραφο, κτλ.

Apache Spark

Παρόλο που το μοντέλο MapReduce φέρνει πολλά πλεονεκτήματα στην ανάλυση δεδομένων, έχει το μεγάλο αρνητικό ότι σε μονολιθικό σύστημα, σε πολλές περιπτώσεις, είναι πολύ πιο αργό από τους παλιούς, παραδοσιακούς τρόπους ανάλυσης δεδομένων. Για αυτό τον λόγο, έχουν δημιουργηθεί και άλλοι τρόπους ανάλυσης δεδομένων. Ενά από τα εναλλακτικά είναι το DataFrame. Το DataFrame είναι ένα από τα πιο κοινά API για δομημένα δεδομένα σε στήλες και γραμμές. Ένα framework το οποίο χρησιμοποιεί αυτό το μοντέλο είναι το Apache Spark, το οποίο είναι πολύ πιο γρήγορο του Hadoop και μπορεί να υλοποιηθεί σε πολλές διαφορετικές γλώσσες, όπως η Java, η Scala και η Python. Ακολουθείται παράδειγμα γραμμένο σε Scala που υλοποιεί το TF-IDF. [23][22]

Ο TF-IDF είναι ένας αλγόριθμος της Ανάκτησης Δεδομένων (Information Retrieval) ο οποίος προσπαθεί να βρει την σημασία της λέξης μέσα σε ένα έγγραφο ή ένα σύνολο εγγράφων.

$$tf(t,d) = 0.5 + 0.5 * \frac{f_{td}}{max\{f_{t',d} : t' \in d\}} \tag{2.1}$$

$$idf(t,D) = log\frac{N}{|d \in D : t \in d|} \tag{2.2}$$

$$tf\_idf(t,d,D) = tf(t,d) * idf(t,D) \tag{2.3}$$

```
val conf = new SparkConf().setAppName("TFIDF").setMaster("local")

val spark = SparkSession.builder.config(conf).getOrCreate()

val sc = spark.sparkContext

val input = sc
  .wholeTextFiles("file://" + System.getProperty("user.dir") + "/data/*")

// TF
val tokenizer = new Tokenizer().setInputCol("fileText").setOutputCol("words")
val wordsData = tokenizer.transform(inputDF)
val hashingTF = new HashingTF().setInputCol("words").setOutputCol("rawFeatures")
val featurizedData = hashingTF.transform(wordsData)

// IDF
val idf = new IDF().setInputCol("rawFeatures").setOutputCol("features")
val idfModel = idf.fit(featurizedData)

// TFIDF
val rescaledData = idfModel.transform(featurizedData)
rescaledData.select("fileNames", "features").show()

sc.stop()
```

NumPy και Pandas

Το module της Python, το Pandas, μπορεί να κάνει parse τα δεδομένα από CSV αρχείο ή από JSON σε DataFrames και με την βοήθεια του NumPy, η επεξεργασία γίνεται πολύ πιο εύκολη λόγω της ευκολίας στην υλοποίηση της. Η Python έχει υπερβολικές πολλές βιβλιοθήκες με υλοποιημένους αλγορίθμους που μπορούν να συνδυαστούν με το NumPy για να βγάλουν κοινό αποτέλεσμα. Για παράδειγμα, στον τομέα της επεξεργασίας εικόνας, συνδυάζονται με τις βιβλιοθήκες TensorFlow και OpenCV. [23][22]

2.3.4 Αποθήκευση των δεδομένων

Σε ένα σύστημα που μία από τις λειτουργίες του είναι η επεξεργασία των δεδομένων, είναι πολύ σημαντικό αυτά τα δεδομένα να αποθηκεύονται με τον βέλτιστο τρόπο ώστε να είναι ασφαλές και να μην χάνονται. Υπάρχουν πολλοί τρόποι να γίνεται η αποθήκευση των δεδομένων αυτών.

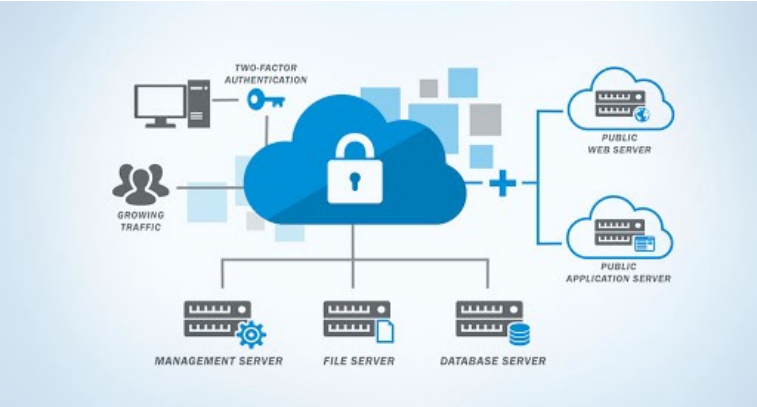
Το πιο κλασσικό παράδειγμα για να γίνεται αποθήκευση δεδομένων είναι με **Βάσεις Δεδομένων (Databases)** γραμμένες σε δομημένες γλώσσες που ονομάζονται **Structured Query Language (SQL)**. Η ιδεολογία της SQL είναι ότι έχει χτιστεί πάνω στο μοντέλο σχετικότητας του Edgar F. Codd που ουσιαστικά χωρίζει τα δεδομένα με κάποια κοινά τους στοιχεία σε πίνακες με στήλες και γραμμές, όπου κάθε στήλη αντιπροσωπεύει ένα χαρακτηριστικό της εγγραφής, ενώ η στήλη είναι η ίδια η εγγραφή. Υπάρχουν αρκετές Βάσεις Δεδομένων που χρησιμοποιούν την SQL για να διαχειριστούν τα δεδομένα τους. Μερικά παραδείγματα είναι η MariaDB, η PostgreSQL και η MySQL.

Ένα πιο μοντέρνο παράδειγμα από Βάσεις Δεδομένων είναι οι NoSQL βάσεις που στηρίζονται σε άλλους τρόπους για να αποθηκεύσουν τα δεδομένα. Για παράδειγμα, η MongoDB χρησιμοποιεί JSON αρχεία για την αποθήκευση τους. Αυτά τα αρχεία είναι αδόμητα αφήνοντας περιθώριο να αποθηκεύονται πιο ελεύθερα

τα δεδομένα στην βάση. Ένα πολύ σημαντικό λάθος που πραγματοποιείται συχνά είναι ότι χρησιμοποιείται MongoDB για κάθε λειτουργία που προσφέρει μία υπηρεσία. Μία πολύ καλή λύση που μπορεί να υπάρχει είναι να γίνει χρήση μίας SQL βάσης με συνεργασία μίας NoSQL για να υπάρχουν τα πλεονεκτήματα των δύο βάσεων, χωρίς να επηρεάζονται από τα μειονεκτήματα τους.

Επίσης, ένα πολύ σημαντικό μοντέλο για την αποθήκευση προσωρινών δεδομένων είναι η χρήση μίας Βάσης που τρέχει πάνω στην κύρια μνήμη. Μία τέτοια βάση είναι η Redis. Ένα παράδειγμα χρήσης της είναι στην περίπτωση ενός αυτοματοποιημένου αυτοκινήτου που μπορεί να οδηγήσει μόνο του. Σε περίπτωση βλάβης, αυτή η πληροφορία πρέπει να αποθηκευτεί προσωρινά, να ενημερωθεί ο χρήστης και να σβηστεί αυτόματα μέτα από κάτι ώρες. Μία άλλη χρήση είναι η αποθήκευση μοναδικών κωδικών (tokens) που κάθε έξι ώρες θα γίνεται ανανέωση και θα σβήνονται οι παλιές. [15][20][23]

2.4 Ασφάλεια του Νέφους (Cloud Security)

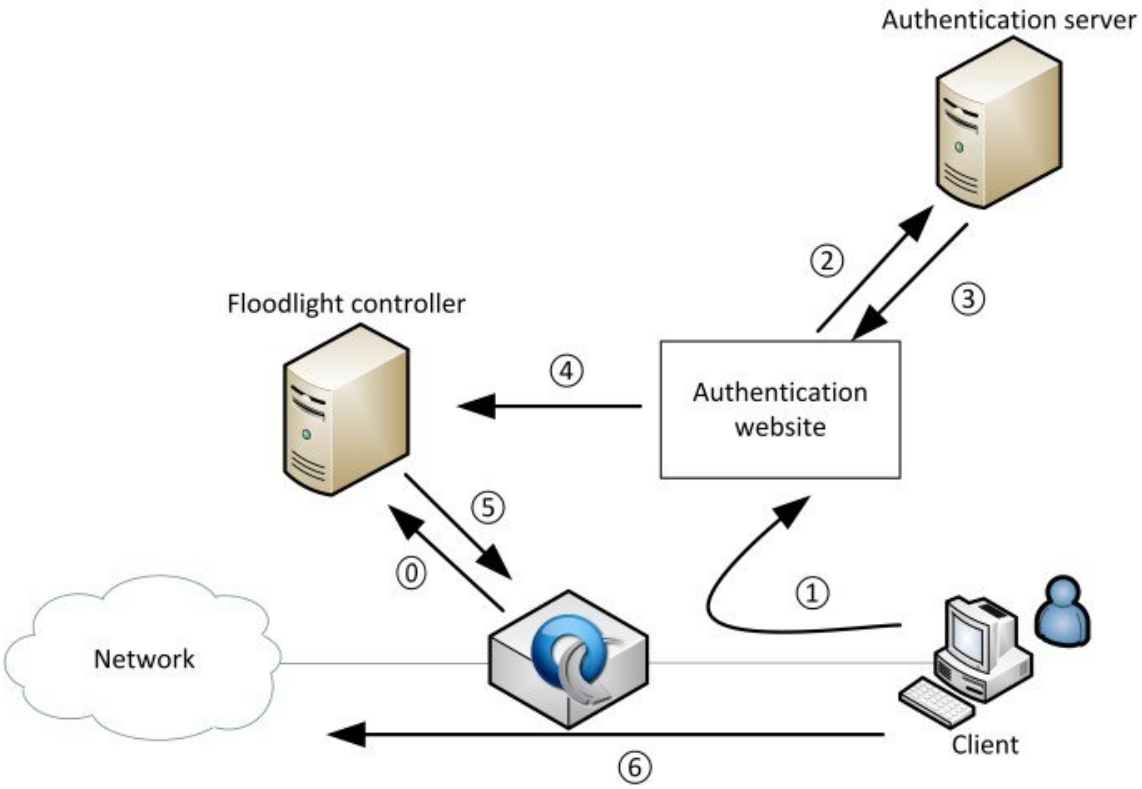


Σχημα 2.8: **Ασφάλεια του Νέφους**: Μία λεζάντα που απεικονίζει πως θα μπορούσε να δούλευε ένα σύστημα ασφάλειας σε εφαρμογή του νέφους. (`https://manrai-tarun.medium.com/what-is-different-about-cloud-security-4e50bb09c4d0`)

Η **Ασφάλεια της Υπολογιστικής Νέφους (Cloud Computing Security)** ή απλώς **Ασφάλεια Νέφους (Cloud Security)** είναι ένα πολύ σημαντικό κομμάτι της ανάπτυξης μίας Cloud υπηρεσίας. Πολλά προβλήματα τα οποία είχαν παρατηρηθεί στον τομέα της Ασφάλειας Δικτύων (Cybersecurity) και των νέων επιθέσεων που εμφανίστηκαν, μπορούν να λυθούν στο Cloud Security. Με λίγα λόγια, το Cloud Security θεωρείται μέρος της ασφάλειας των υπολογιστών και ασχολείται με μία συλλογή από πολιτικές, τεχνολογίες, εφαρμογές και ελέγχους την προστασία των πληροφοριών στο Cloud, όπως τις τοπικές IP, τις πληροφορίες και τις υπηρεσίες που τρέχουν. [18]

2.4.1 Βασικές Αρχές Ασφάλειας

Επαλήθευση (Authentication)



Σχημα 2.9: **Επαλήθευση χρήστη (authentication)**: Παράδειγμα επαλήθευσης χρήστη σε ένα σύστημα. Ο χρήστης παίρνει πρόσβαση στο σύστημα μέσω ενός authentication server. (`https://www.researchgate.net/figure/User-authentication-step-by-step-chart_fig1_306063519`)

Η **Επαλήθευση (Authentication)** αναφέρεται στο κομμάτι της άμυνας του Cloud Security που έχει σχέση με την διαχείριση των χρηστών. Είναι πολύ σημαντικό να γνωρίζει το κάθε σύστημα ποιος, πότε και από που συνδέθηκε κυρίως άμα είναι ένα κλειστό σύστημα που περιέχει ευαίσθητες πληροφορίες. Ένα σύστημα που διαχειρίζεται τους χρήστες είναι το Keycloak, το οποίο μπορεί να τροποποιηθεί πλήρως από τον διαχειριστή του συστήματος. Συνεργάζεται συνήθως με μία SQL βάση, όπως η PostgreSQL και ελέγχει ένα αναγνωριστικό του χρήστη εάν είναι έγκυρο ή όχι. Ένα τέτοιο σύστημα παρουσιάζεται στην Εικόνα 2.9. [14][2][18]

Εξουσιοδότηση (Authorization)

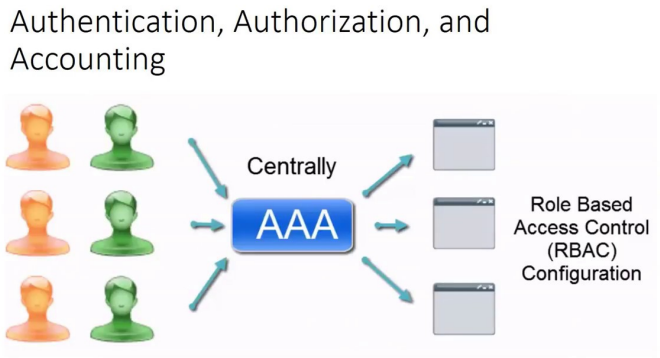
Επαλήθευση (Authentication)	**Εξουσιοδότηση (Authorization)**
Εγκρίνει την ταυτότητα του χρήστη	Δίνει άδεια πρόσβαση
Η προϋπόθεση είναι να γνωρίζει τα πιστοποιητικά του	Οι προϋποθέσεις είναι να έχει επαλειφθεί η ταυτότητα του και να έχει πρόσβαση στο σύστημα

Πινακας 2.2: Διαφορές μεταξύ επαλήθευσης και εξουσιοδότησης

Η **Εξουσιοδότηση (Authorization)** είναι η διαδικασία κατά την οποία ελέγχει έαν ο χρήστης έχει δικαιώματα να χρησιμοποιήσει την ενέργεια που επιχειρεί να εκτελέσει. Αυτό συνοδεύει το Authentication ώστε να παραχωρεί ρόλους χρηστών, όπου κάθε ρόλος είναι ξεχωριστός και διαφορετικός. Αυτό κάνει την κλοπή δεδομένων πιο δύσκολη από έναν τρίτο που μπαίνει στο σύστημα. Έστω ότι καταφέρνει να πάρει ένα αναγνωριστικό ενός χρήστη, δεν σημαίνει απαραίτητα ότι θα συμφωνεί με την εξουσιοδότηση το οποίο έχει. Συνήθως, το authorization πραγματοποιείται στα APIs και χρησιμοποιείται το JWT (JavaScript Web Token). Ουσιαστικά, ένα JWT είναι ένας μηχανισμός κατά τον οποίο ελέγχει τον ιδιοκτήτη της πληροφορίας που μεταφέρεται σε JSON και είναι κρυπτογραφημένο. Στον Πίνακα 2.2 παρουσιάζονται οι διαφορές μεταξύ της εξουσιοδότησης και της επαλήθευσης. [2][18]

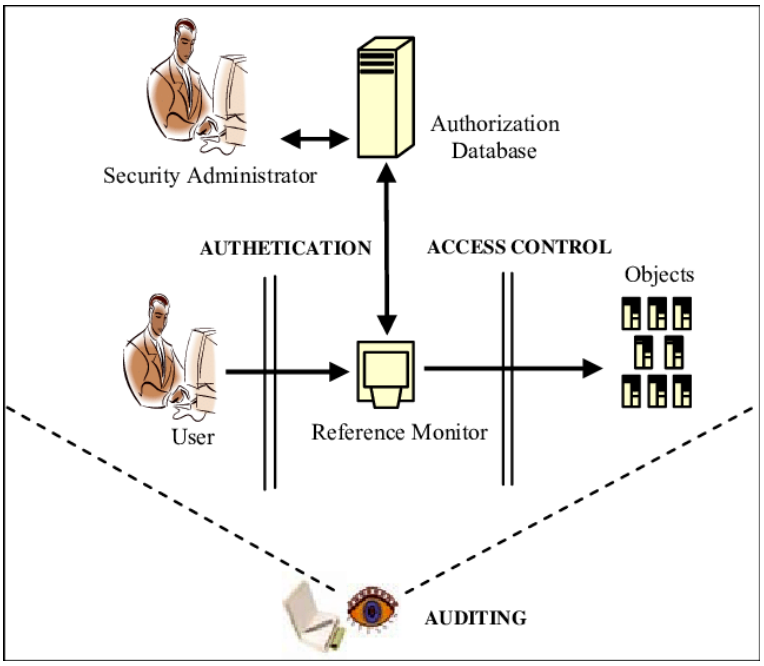
Αναφορά (Accounting)

Με τον όρο **Αναφορά (Accounting)** αναφερόμαστε στον ρόλο της ομάδα, στον οποίο τα μέλη τους είναι υπεύθυνα για τον έλεγχο του συστήματος στο δίκτυο διαβάζοντας τα logs τους. Δηλαδή, ψάχνουν να βρουν ύποπτες κινήσεις, συλλέγουν στοιχεία και τα παρουσιάζουν στην ομάδα της κυβερνοασφάλειας ώστε να βρουν το λάθος που δημιουργήθηκε στο σύστημα. [3][18]



Σχημα 2.10: **Το 3Α πλαίσιο**: Η επαλήθευση, η εξουσιοδότηση και αναφορά αποτελούν το 3Α πλαίσιο. (`https://www.youtube.com/watch?v=LkZzYp13ucg`)

2.4.2 Ελεγκτές Πρόσβασης (Access Control)



Σχημα 2.11: **Ελεγκτές Πρόσβασης**: Στην εικόνα, φαίνεται το πως θα έπρεπε να δουλεύει ένα σύστημα ελέγχου πρόσβασης με συνεργασία του 3A πλαισίου. (https://www.researchgate.net/figure/Access-Control-and-Other-Security-Services_fig1_228994574)

Οι **Ελεγκτές Πρόσβασης (Access Control)** είναι μερικά εργαλεία που αποκλείουν ή δίνουν πρόσβαση σε άλλους δέκτες. Οι ελεγκτές δουλεύουν σε δύο διαφορετικά επίπεδα του OSI.

- Στο επίπεδο Εφαρμογής, όπου η άμυνα γίνεται μέσω του πρωτοκόλλου **Cross-origin Resource Sharing (CORS)**, το οποίο ελέγχει άμα ο κόμβος X έχει πρόσβαση για την ενέργεια που προσπαθεί να κάνει. Για παράδειγμα, άμα θέλει να καλέσει το */users*, θα πρέπει να έχει πρόσβαση στο **GET /users** του API. Εάν δεν έχει, τότε αποκλείει την πρόσβαση και του στέλνει ενημερωτικό σφάλμα. Για να ρυθμιστεί αυτό, δίνεται έτοιμο από το framework που χρησιμοποιήθηκε για την υλοποίηση. [13]
- Στο επίπεδο Δικτύου, όπου χρησιμοποιείται το εργαλείο iptables του GNU υπάρχουν δύο ρυθμίσεις: α) να επιτρέπει τους πάντες (allow-all) ή β) να απορρίπτει τους πάντες (disallow-all). Είναι ένα πολύ πιο ισχυρό εργαλείο από ότι ειναι το CORS και αρκετά πιο περίπλοκο. [18]

Κεφάλαιο 3

Σχεδιασμός Πλατφόρμας Συνάρτησης ως Υπηρεσίας (Function as a Service)

3.1 Εισαγωγή

Το 2010 εμφανίστηκε ένα νέο μοντέλο του cloud computing που το λένε **Συνάρτηση ως Υπηρεσία (Function as a Service, FaaS)**. Σκοπό του μοντέλου αυτού είναι να παρέχει υπηρεσίες οι οποίες να επιτρέπουν τον χρήστη να μπορεί να ανάπτυξη κωδικά και να διαχειριστή λειτουργίες σε μία πλατφόρμα χωρίς να υπάρξει η ανάγκη να γίνετε το στάδιο του building ή η συντήρηση της υποδομής, δημιουργώντας ένα πραγματικό «serverless» μοντέλο.

Serverless είναι η ιδεολογία στο να γίνει αντικατάσταση ενός παραδοσιακού server σε ένα απομακρυσμένο API χωρίς να υπάρχει η ανάγκη να αναπτυχθεί περίπλοκος κώδικας. Στο FaaS, αυτό που χρειάζεται να κάνει ο τελικός χρήστης είναι να ανεβάσει ένα τμήμα κώδικα και να μπορεί να εκτελεσθεί αυτόνομα.

Η πρώτη πετυχημένη υλοποίηση μίας FaaS πλατφόρμας ξεκίνησε το 2017 από την Amazon η οποία ονομάζεται AWS Lambda και παραμένει μία πολύ δυνατή επιλογή για να χρησιμοποιηθεί από μία εταιρεία. Οι περιπτώσεις χρήσεων που έχει μία τέτοια πλατφόρμα είναι αρκετές κυρίως για μικρές λειτουργίες που δεν χρειάζονται πολύ υπολογιστική ενεργεία. Για παράδειγμα, θα μπορούσε να χρησιμοποιηθεί για να γίνει data analysis μέσα από μία βάση δεδομένων, τα οποία για παράδειγμα μπορεί να έχουν σχέση με την θερμοκρασία μίας περιοχής και να εμφανίζει τα αποτελέσματα σε μία διεπαφή χωρίς να υπάρχει κάποιο άλλο API ενδιάμεσα.

Το μοντέλο του PaaS με του FaaS έχουν αρκετές ομοιότητες όπως ότι και τα δύο μοντέλα έχουν σχεδιαστεί με σκοπό να μπορούν να παρέχουν υπηρεσίες σε χρήστες χωρίς να υπάρχει η ανάγκη να έχει πρόσβαση στην υποδομή ο τελικός χρήστης. Η διαφορά είναι στην κοστολόγηση. Το PaaS χρεώνει για κάθε δευτερόλεπτο που νοικιάζει ο χρήστης το σύστημα, ενώ το FaaS μόνο όποτε εκτελείται το API.

Η πλατφόρμα θα πρέπει επίσης να υποστηρίζει **Συνεχές Ενσωμάτωση/Συνεχές Παράδοση (Continuous Integration/Continuous Delivery, CI/CD)**, δηλαδή να χρησιμοποιείται ένα πρόγραμμα **Παρακολούθησης Αλλαγών (Version Control)** και η δουλεία να γίνεται με μικρές κυκλικές κινήσεις σε γρήγορη ταχύτητα και συχνότητα. Με άλλα λόγια να δουλεύει με **Ευέλικτες Μεθοδολογίες (Agile Methodologies)**. [4][20]

3.2 Βασικά Χαρακτηριστικά

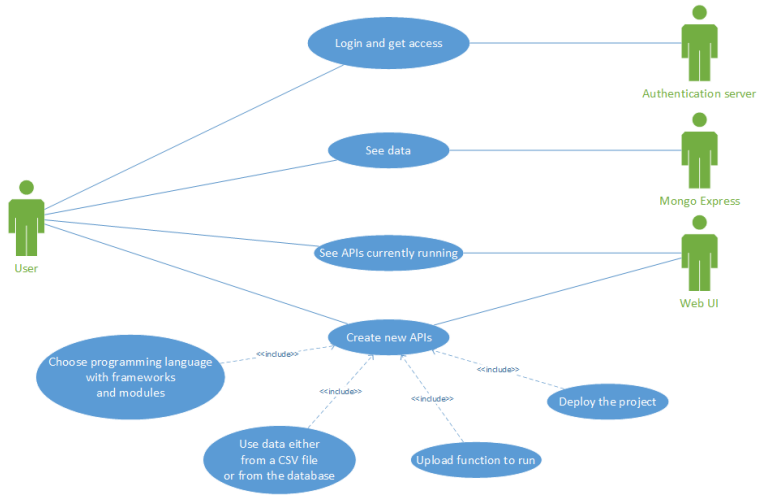
Μία πλατφόρμα πρέπει να έχει τα εξής χαρακτηριστικά:

- Ένας χρήστης συνδέεται με τον λογαριασμό του από έναν oauth API για να παρέχεται authentication προστασία
- Θα πρέπει να εμφανίζει μία λίστα από διαθέσιμα APIs τα οποία τρέχουν στο σύστημα του χρήστη και μπορεί να χρησιμοποιήσει
- Ένα GUI το οποίο θα δείχνει τα περιεχόμενα μίας δημόσιας βάσης που μπορεί να χρησιμοποιήσει ο τελικός χρήστης
- Ένα μενού περιήγησης που ο χρήστης ακολουθάει μερικά βήματα για να φτιάξει ένα API. Αυτά τα βήματα είναι:
  1. Επιλογής γλώσσας προγραμματισμού και αναφορά στις βιβλιοθήκες που θα χρειαστεί για να εκτελεσθεί η συνάρτηση
  2. Επιλογής κάποιου έτοιμου πίνακα από την βάση ή ανέβασμα νέων δεδομένων στην βάση σε μορφή JSON ή CSV
  3. Ανέβασμα της συνάρτησης μέσα σε ένα ειδικό πεδίο
  4. Deployment της εφαρμογής στο σύστημα όπου θα δίνετε ένα τυχαίο port που ο χρήστης μπορεί να χρησιμοποιήσει για να δει τα αποτελέσματα

Με αυτά τα χαρακτηριστικά, πληρούνται μερικά βασικά χαρακτηριστικά που θα πρέπει να παρέχεται από το σύστημα. [20]

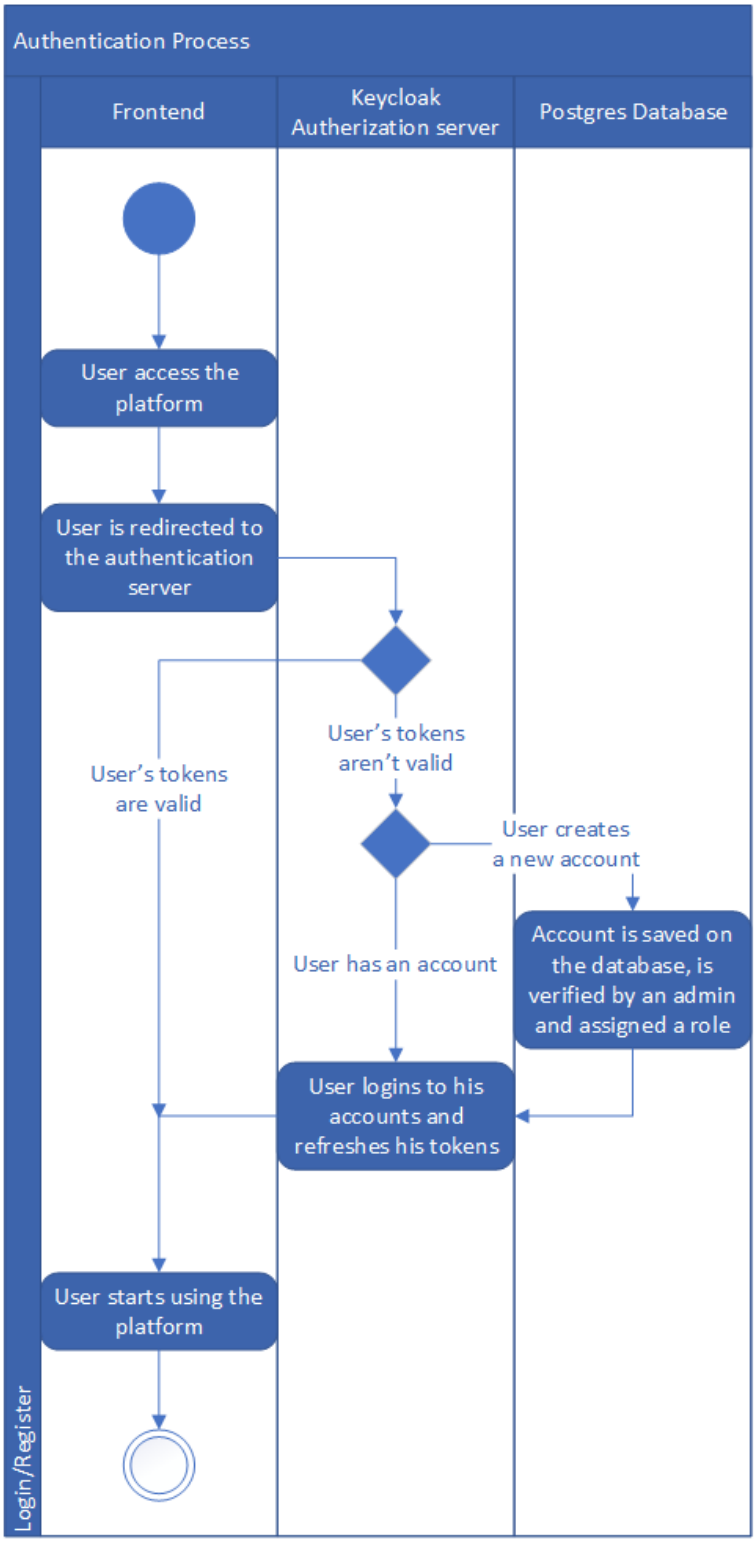
3.2.1 Διαγράμματα σχεδιασμού

Διάγραμμα Περιπτώσεων Χρήσεις



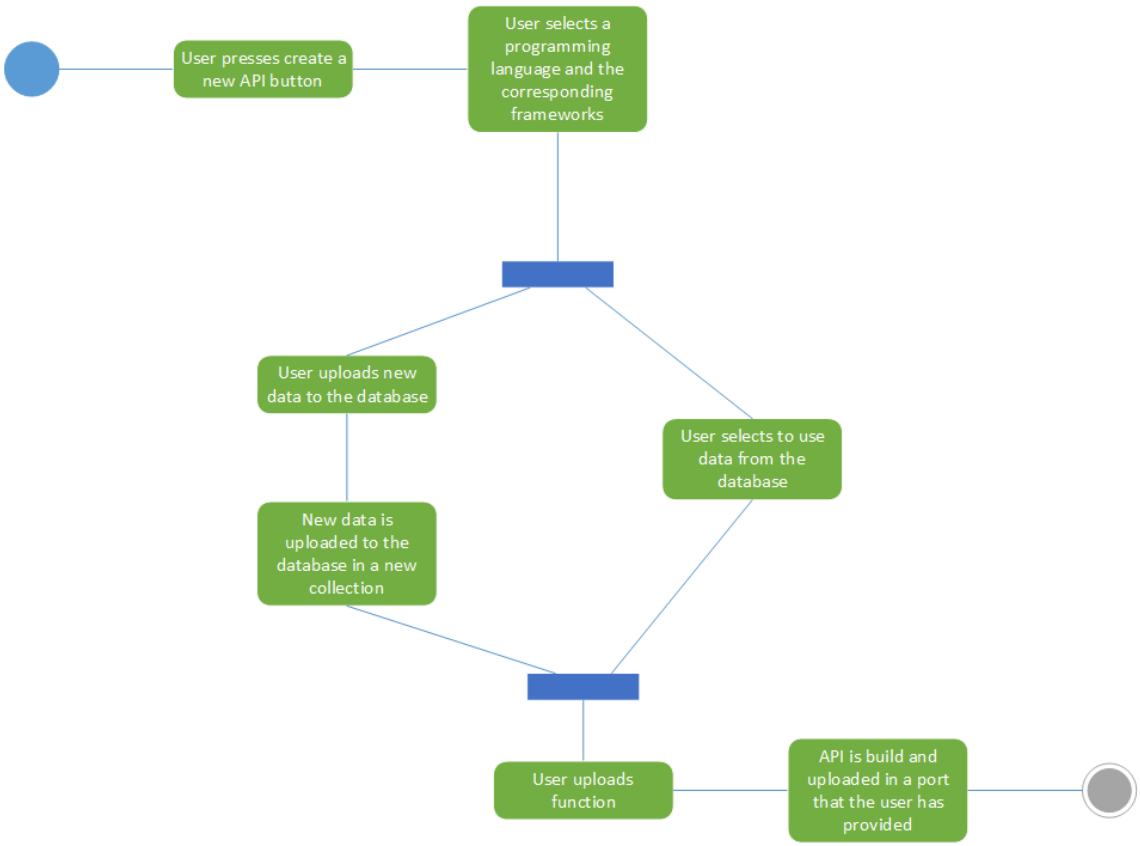
Σχημα 3.1: **Διάγραμμα Περιπτώσεων Χρήσεις (Use Case)**: στο διάγραμμα παρουσιάζονται τα μέλη του συστήματος. Ο χρήστης μπορεί να ζητήσει άδεια από τον authorization server για να πάρει πρόσβαση στο σύστημα, να συνδεθεί στην βάση για να δει τα περιεχόμενα των πληροφοριών που προσφέρονται και να δημιουργήσει νέα APIs μέσω της γραφικής διεπαφής.

Διάγραμμα Δραστηριότητας Σύνδεσης Χρήστων



Σχημα 3.2: **Διάγραμμα Δραστηριότητας (Activity Diagram) Σύνδεσης Χρηστών**: Το διάγραμμα περιγράφει την διαδικασία στην οποία ο χρήστης παίρνει το μοναδικό του χαρακτηριστικό για να μπορεί να έχει πρόσβαση στην πλατφόρμα.

Διάγραμμα Δραστηριότητας δημιουργίας νέου API



Σχημα 3.3: **Διάγραμμα Δημιουργίας Νέου API**

3.3 Επιλογή τεχνολογιών

Η επιλογή των τεχνολογιών για την ανάπτυξη της εφαρμογής είναι πολύ σημαντικό στάδιο. Πρέπει να γίνει αρκετά μεγάλη ερεύνα αναγκαιότητας και είναι ένα αρκετά σημαντικό μέρος στην ανάπτυξη μίας εφαρμογής ή πλατφόρμας. Σε αυτή την περίπτωση επιλέχθηκαν οι εξής τεχνολογίες:

- ReactJS για την δικτυακή γραφική διεπαφή
- ExpressJS για την δημιουργία ενός middleware API
- Flask για ένα μικρό API που ο μόνος του σκοπός είναι να διαβάζει τα περιεχόμενα ενός αρχείου
- Gin και Docker SDK για ένα API το οποίο διαχειρίζεται τις ενέργειες που θα μπορούν να πραγματοποιηθούν στην πλατφόρμα
- MongoDB για την αποθήκευση των δεδομένων σε NoSQL βάση
- Keycloak και PostgreSQL για την διαχείριση των χρηστών
- Docker για την δημιουργία containers και Docker Swarm για το deployment τους στο cluster

- Jest για tests στον κώδικα
- Docosaurus για την δημιουργία εγγράφων που θα βοηθάει τους χρήστες να χρησιμοποιήσουν την πλατφόρμα

Κεφάλαιο 4

Διεπαφή του Docker

4.1 Εισαγωγή

Η **διεπαφή του Docker**, αναφερόμενη και ως **Docker API**, διαχειρίζεται τις ενέργειες που πρέπει να κάνει η μηχανή του Docker για να λειτουργήσει η πλατφόρμα. Χωρίς αυτό το API, δεν θα δούλευε η πλατφόρμα και απομακρυσμένα.

Το συγκεκριμένο API είναι υλοποιημένο στην γλώσσα προγραμματισμού Go χρησιμοποιώντας την βιβλιοθήκη του Docker SDK για να μπορεί να τρέχει σε χαμηλό επίπεδο εντολές στο σύστημα. [6]

4.1.1 Η γλώσσα προγραμματισμού Go

Η Go είναι μία γλώσσα προγραμματισμού που αναπτύχθηκε από την google και έχει παρόμοια σύνταξη με την C με την διαφορά ότι παρέχει συλλεκτη σκουπιδιών (garbage collector). Κεντρικός της φτιάχνονται γρήγορα, σταθερά και αποδοτικά επεκτάσιμα λογισμικά.

Παράδειγμα στην Golang: Γεια σου κόσμε!

```
package main

import "fmt"

func main() {
  fmt.Println("Hello World!")
}
```

4.1.2 Βιβλιοθήκες για την διεπαφή του Docker

Οι βιβλιοθήκες που χρησιμοποιήθηκαν για το API του Docker ήταν δύο:

- Το Gin το οποίο είναι υπεύθυνο να δημιουργεί APIs.
- Το Docker SDK η οποία είναι η επίσημη βιβλιοθήκη του Docker για την Go.

4.2 Οργάνωση του API

4.3 Παραδείγματα υλοποίησης

Για την καλύτερη κατανόηση του API, θα μελετηθούν δύο παραδείγματα, αυτό της δημιουργίας ενός νέου container και αυτό που επιστρέφει τις πληροφορίες των τρέχων containers.

4.3.1 Παράδειγμα Α: Δημιουργίας νέου container

Δομή BuildImage

```
type BuildImage struct {
  Name string `json:"name"`
  Path string `json:"path"`
}
```

Συνάρτηση imageBuildFunction

Η συνάρτηση imageBuildFunction διαβάζει τα τοπικά αρχεία από το τοπικό σύστημα και δημιουργεί την βάση. Για την παρουσίαση των logs, χρησιμοποιείται η βοηθητική συνάρτηση **lib.Print**.

```
// imageBuildFunction is a private method that build the image.
// If something will go wrong, it returns the error, otherwise it returns nill.
func imageBuildFunction(dockerClient *client.Client, imageBuild *BuildImage) error {
  // Create the background context
  ctx, cancel := context.WithTimeout(context.Background(), 120*time.Second)
  defer cancel()

  // Create the path and then compress it to tar
  path := fmt.Sprintf("/files/%s", imageBuild.Path)
  tar, err := archive.TarWithOptions(path, &archive.TarOptions{})
  if err != nil {
    return err
  }

  opts := types.ImageBuildOptions{
    Dockerfile: "Dockerfile",
    Tags:       []string{imageBuild.Name},
    Remove:     true,
  }
  res, err := dockerClient.ImageBuild(ctx, tar, opts)
  if err != nil {
    return err
  }

  defer res.Body.Close()

  err = lib.Print(res.Body)
  if err != nil {
    return err
  }

  return nil
}
```

Συνάρτηση Build

Η Build δέχεται τα ορίσματα που παίρνει από τον client και του επιστρέφει πίσω μήνυμα επιτυχίας.

```
func Build(c *gin.Context) {
  var imageBuild BuildImage

  if err := c.ShouldBindJSON(&imageBuild); err != nil {
    c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
    return
  }

  err := imageBuildFunction(lib.InitDockerCli(), &imageBuild)
  if err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
    return
  }

  c.JSON(http.StatusOK, gin.H{"message": "Image was built successfully"})
}
```

4.3.2 Παράδειγμα Β: Επιστροφή τρέχων containers

Δομές των containers

```
type ListContainer struct {
  ID       string    `json:"id"`
  Names    []string  `json:"names"`
  Ports    []uint16  `json:"ports"`
  State    string    `json:"state"`
  Status   string    `json:"status"`
  Networks []Network `json:"networks"`
  Volumes  []Volume  `json:"volumes"`
}

type Network struct {
  ID        string `json:"id"`
  IPAddress string `json:"ipAddress"`
}

type Volume struct {
  Name string `json:"name"`
  Path string `json:"path"`
}
```

Συνάρτηση List

```
// List is a function that simulates the docker ps unix command.
func List(c *gin.Context) {
  cli := lib.InitDockerCli()

  // Function that returns the currently running containers.
  containers, err := cli.ContainerList(context.Background(), types.ContainerListOptions{})
  if err != nil {
    panic(err)
  }

  // Parse data and keep only the useful information
  var containersList []ListContainer
  for _, container := range containers {
    var ports []uint16

    for _, port := range container.Ports {
      ports = append(ports, port.PrivatePort)
    }

    var networks []Network
    for _, network := range container.NetworkSettings.Networks {
      networks = append(networks, Network{
        ID:        network.NetworkID,
        IPAddress: network.IPAddress,
      })
    }

    var volumes []Volume
    for _, volume := range container.Mounts {
      volumes = append(volumes, Volume{
        Name: volume.Name,
        Path: volume.Destination,
      })
    }

    containersList = append(containersList, ListContainer{
      ID:       container.ID,
      Names:    container.Names,
      Ports:    ports,
      State:    container.State,
      Status:   container.Status,
      Networks: networks,
      Volumes:  volumes,
    })
  }

  c.JSON(http.StatusOK, containersList)
}
```

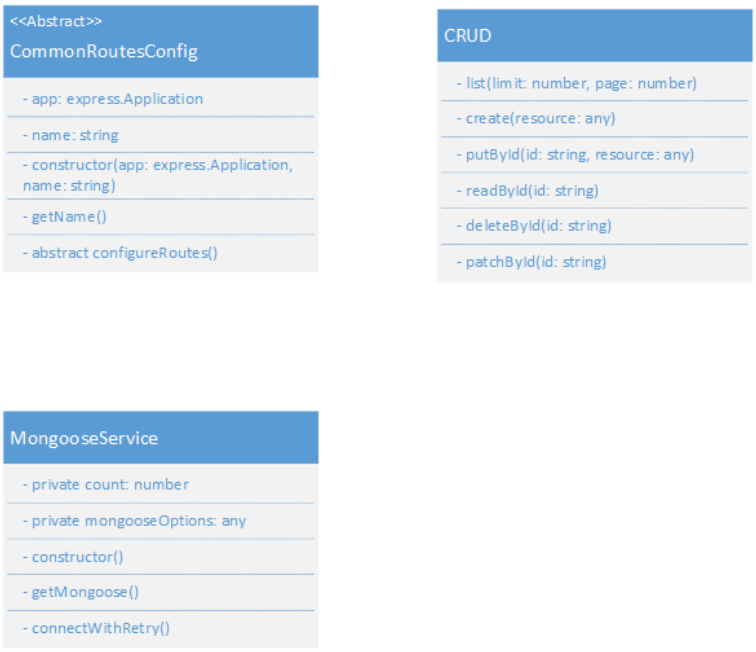
Κεφάλαιο 5

Μεσολαβητής (Middleware)

5.1 Εισαγωγή

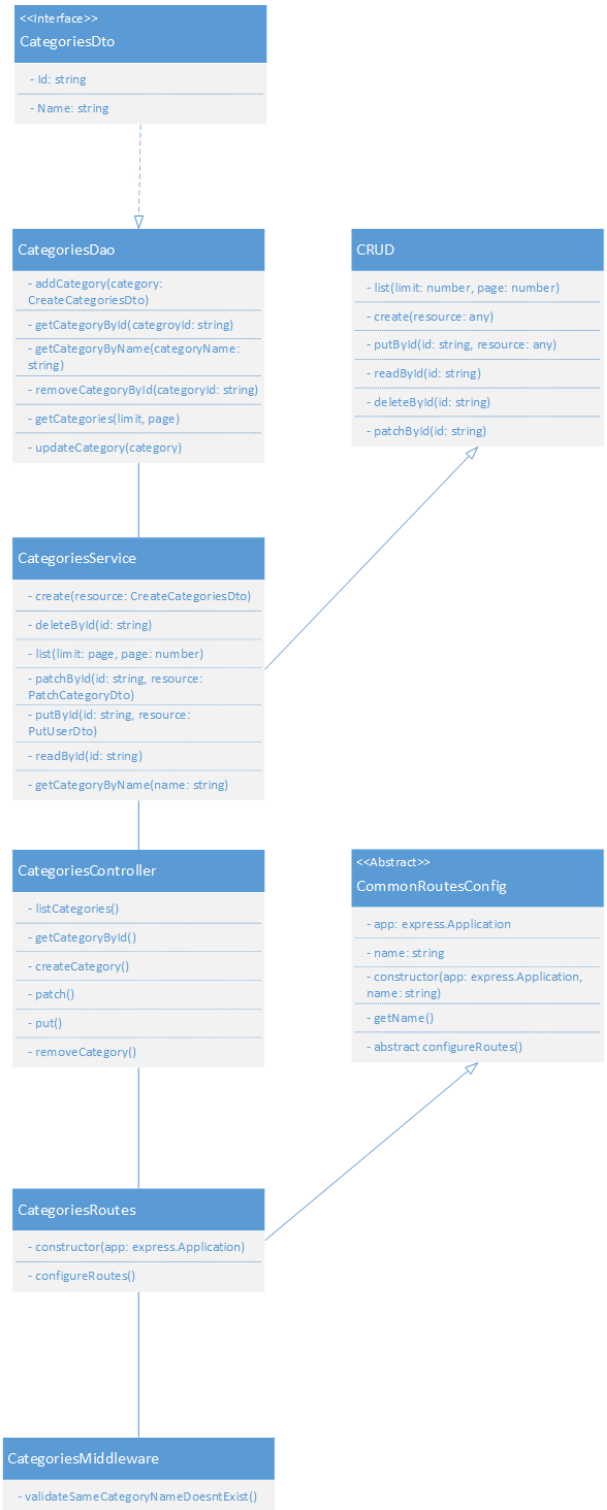
Ο **Μεσολαβητής (Middleware)** είναι ένας τύπος λογισμικού ο οποίος ενώνει τους κόμβους ενός κατανεμημένου συστήματος κάνοντας το πιο απλό για τους προγραμματιστές να υλοποιούν την επικοινωνία μεταξύ τους και να επικυρώνονται στην ίδια την εφαρμογή. Για την πλατφόρμα ο middleware είναι υλοποιημένος σε TypeScript και Express και είναι σχεδιασμένος με την αρχιτεκτονική **Μοντέλο-εμφάνιση-ελεγκτής (Model-view-controller, MVC)**. [16]

5.1.1 Διάγραμμα κοινών κλάσεων



Σχημα 5.1: Στο σχήμα, παρουσιάζονται κλάσεις οι οποίες χρησιμοποιούνται κοινώς από διάφορα μέρη (components) του middleware.

5.1.2 Διάγραμμα οργάνωσης ελεγκτή (controller)



Σχημα 5.2: Στο συγκεκριμένο σχήμα, παρουσιάζεται πως θα είχε οργανωθεί ένας controller εάν ήταν με κατηγορίες.

5.1.3 Η γλώσσα προγραμματισμού TypeScript

Η TypeScript είναι μία γλώσσα προγραμματισμού που έχει αναπτυχθεί από την Microsoft και είναι ένα υποσύνολο της JavaScript με σκοπό να φέρει στατικούς τύπους στην γλώσσα.

Παράδειγμα: συνάρτηση Hello World στην JavaScript

```
function print(message) {
  console.log(message);
}

let message = "Hello World!";
print(message);
```

Παράδειγμα: συνάρτηση Hello World στην TypeScript

```
function print(message: string): null {
  console.log(message);
}

let message: string = "Hello World!";
print(message);
```

Η βιβλιοθήκη που χρησιμοποιήθηκε ήταν η Express στην οποία είχε γίνει αναφορά στο Κεφάλαιο ??. Για κάθε συνάρτηση που παρέχει το middleware, έχει γραφτεί και ένα αντίστοιχο τεστ για να ελέγχει την ποιότητα του κώδικα. Η βιβλιοθήκη που χρησιμοποιήθηκε ήταν το Jest.

5.2 Παραδείγματα υλοποίησης

Θα μελετηθούν δύο παραδείγματα υλοποιήσεις μαζί με τα test τους για την καλύτερη κατανόηση του συστήματος. Το πρώτο είναι η δημιουργία ενός νέου dataset και το δεύτερο ενός νέου container.

5.2.1 Παράδειγμα Α: Δημιουργία ενός νέου dataset

Δομή του DatasetDto

```
export interface DatasetDto {
  name: string;
  description: string;
  data: Object;
}
```

Δομή του Dataset στην Βάση

```
datasetSchema = new this.Schema(
  {
    _id: {
      type: String,
      default: shortid.generate()
    },
    name: {
      type: String
    },
    description: {
      type: String
    },
    data: {
      type: [Object],
      default: []
    },
    created: {
      type: Date,
      default: Date.now
    }
  }
);
```

Μέθοδος createDataset στην κλάση DAO

```
async createDataset(datasetDto: DatasetDto): Promise<any> {
  try {
    const dataset = new this.DatasetModel(
      {
        _id: shortid.generate(),
        name: datasetDto.name,
        description: datasetDto.description,
        data: datasetDto.data
      }
    );

    await dataset.save();
    return dataset;
  } catch (e) {
    console.error(e);
  }
}
```

Μέθοδος create στην κλάση Service

```
async create(name: string, description: string, path: string): Promise<DatasetDto> {
  let data: object = await DatasetsDao.getData(path);
  let datasetDto: DatasetDto = {
    name: name,
    description: description,
    data: data
  };
  return DatasetsDao.createDataset(datasetDto);
};
```

Μέθοδος createDataset στην κλάση Controller

```
async createDataset(req: express.Request, res: express.Response) {
  try {
    let name: string = req.body.name;
    let description: string = req.body.description;
    let path: string = req.body.path;
    const dataset = await DatasetsService.create(name, description, path);
    res.status(201).json(dataset);
  } catch (e) {
    res.status(400).json(e);
  }
}
```

Τεστ λειτουργικότητας

```
it('should upload the dataset to the database', async () => {
  const res = await request.post("/datasets")
    .send({
      "name": "test",
      "description": "A test database to check if it works",
      "path": "test.csv"
    });
  expect(res.status).toBe(201);
  expect(res.body).toMatchObject(testDatasetDto);
  id = res.body._id;
});
```

5.2.2 Παράδειγμα Β: Δημιουργία νέου container

Δομή BuildDockerDto

```
export interface BuildDockerDto {
  name: string;
  path: string;
}
```

Μέθοδος createContainer της κλάσης DAO

```
async createContainer(container: BuildDockerDto): Promise<string> {
  let response = await axios.post(`${this.api}/build`, container);
  return await response.data.message;
}
```

Μέθοδος build της κλάσης Service

```
async build(resource: BuildDockerDto): Promise<string> {
  return await DockerDao.createContainer(resource);
}
```

Μέθοδος build της κλάσης Controller

```
async build(req: express.Request, res: express.Response) {
  let message = await dockerService.build(req.body);
  res.status(200).send(message);
}
```

Τεστ λειτουργίας

```
it('should build the container', async () => {
  const res = await request.post("/docker/build")
    .send({
      name: "test",
      path: "test"
    });
  expect(res.status).toBe(200);
  expect(res.text).toBe("Container was built successfully");
});
```

Κεφάλαιο 6

Δικτυακή Διεπαφή Χρήστη

6.1 Εισαγωγή

Η δικτυακή διεπαφή του χρήστη είναι ο τελευταίος κόμβος που υλοποιήθηκε. Κεντρικός σκοπός του ήταν να παρέχει ένα όμορφο και παρουσιαστικό περιβάλλον με καλή εμπειρία του χρήστη. Για να πραγματοποιηθεί ο σκοπός, αποφασίστηκε να γίνει σε δικτυακή μορφή, έτσι ώστε να μπορεί να λειτουργήσει σε κάθε λειτουργικό σύστημα και συσκευή το ίδιο καλά. Δόθηκε ιδιαίτερη προσοχή έτσι ώστε κάθε χρήστης να μπορεί να έχει μία άνετη εμπειρία και οι ενέργειες που να μπορεί να κάνει να είναι ξεκάθαρες. Η υλοποίηση πραγματοποιήθηκε στην TypeScript και συγκεκριμένα χρησιμοποιώντας το framework React. [24]

6.1.1 Το framework ReactJS

Η ReactJS είναι ένα framework ανοιχτού κώδικα αναπτυγμένο από την Meta (η εταιρεία που ήταν γνωστή ως Facebook) που η κύρια χρήση της είναι να φτιάχνονται εφαρμογές μικρού ή μεσαίου μεγέθους με δυναμικά δεδομένα. Η React, ομοίως με την Vue και αντιθέτως με την Angular, έχει κάθε πράγμα που τρέχει μέσα σε ένα αρχείο όπου έχει την προέκταση *.jsx* για JavaScript ή *.tsx* για TypeScript.

Βασικό στόχος είναι να χωριστεί η εφαρμογή σε μικρά τμήματα ονόματι components. Υπάρχουν δύο τρόποι που η React αρχικοποιεί τα components. Ο πρώτος τρόπος είναι με την χρήση συνάρτησης και ο δεύτερος με την χρήση κλάσης. Συνηθίζεται να χρησιμοποιείται παραπάνω η κλάση εκτός εάν υπάρχει κάποιο κάλεσμα σε API. [12][1][10]

Τρόπος Α: με την χρήση συνάρτησης

```
function HomePage(): JSX.Element {
  return (
    <h1>Hello World!</h1>
  )
}
```

Τρόπος Β: με την χρήση κλάσης

```
class HomePage extends React.Component {
  render() {
    return (
      <h1>Hello World!</h1>
    )
  }
}
```

6.1.2 Κύριες βιβλιοθήκες που χρησιμοποιήθηκαν

Redux Toolkit

Το Redux Toolkit είναι ένα εργαλείο που διαχειρίζεται την κατάσταση των μεταβλητών από τον έναν κόμβο σε έναν δεύτερο. Ένα απλό παράδειγμα χρήσης του είναι ένας μετρητής.

```
import { createSlice } from '@reduxjs/toolkit'
import type { PayloadAction } from '@reduxjs/toolkit'

export interface CounterState {
  value: number
}

const initialState: CounterState = {
  value: 0,
}

export const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    increment: (state) => {
      state.value += 1
    },
    decrement: (state) => {
      state.value -= 1
    },
    incrementByAmount: (state, action: PayloadAction<number>) => {
      state.value += action.payload
    },
  },
})

export const { increment, decrement, incrementByAmount } = counterSlice.actions

export default counterSlice.reducer
```

React Query

Το React Query είναι μία βιβλιοθήκη που διαχειρίζεται τα καλέσματα των APIs και τα κάνει όσο πιο δυναμικά γίνεται με μικρή παρέμβαση από τον προγραμματιστή.

Έστω ότι παίρνει δεδομένα για τους χρήστες από μία συνάρτηση ***GetUsers***.

```
function Users() {
  const { isLoading, error, data } = useQuery('GetUsers', GetUsers);

  if (isLoading) return 'Loading...';

  if (error) return 'An error has occurred: ' + error.message;

  return (
    data.forEach((user: User) => (
      ...
    ));
  );
}
```

TailwindCSS και HeadlessUI

Η TailwindCSS είναι μία συλλογή από κλάσης της CSS που έχουν δυναμικό χαρακτήρα. Το HeadlessUI είναι ένα module που έχει χτιστεί πάνω στην Tailwind και έχει σκοπό να έχει έτοιμα components για την React ή την Vue.

```
<h1 class="text-3xl pr-2 font-semibold underline">
  A simple Tailwind example
</h1>
```

6.2 Υλοποίηση σύνδεσης χρηστών

Σύμφωνα με το Σχήμα 3.2, θα πρέπει οι χρήστες να συνδέονται με το που έχουν πρόσβαση στο σύστημα, να έχουν ένα μοναδικό χαρακτηριστικό και αυτό να έχει ένα όριο ζωής.

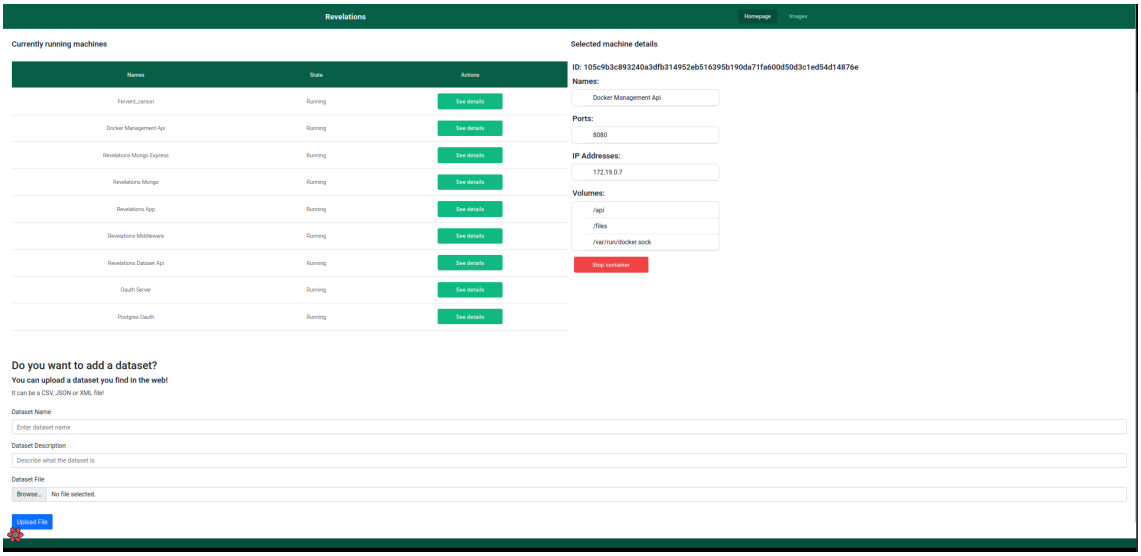
Το σύστημα Keycloak μπορεί να δουλέψει και με παραπάνω από μία εφαρμογή. Για αυτό τον λόγο, έχουν υλοποιήσει ένα σύστημα που λέγεται Realm, όπου κάθε εφαρμογή έχει ένα δικό του Realm. Για την πλατφόρμα, δημιουργήθηκε ένα Realm με το όνομα της και μέσα στον client, έχουν γίνει οι κατάλληλες ρυθμίσεις ώστε να μπορεί να συνδεθεί ο χρήστης.

Στην άλλη πλευρά, έχει χρησιμοποιηθεί το module ***keycloakjs*** για να κάνει τους ελέγχους. Συγκεκριμένα:

```
const {keycloak, initOptions} = keycloakConfig;
keycloak.init(initOptions)
  .then((auth: boolean) => {
    if (!auth)
      window.location.reload();

    root.render(
      ...
    );

    localStorage.setItem('token', keycloak.token as string);
    localStorage.setItem('token-refresh', keycloak.refreshToken as string);

    setTimeout(() => {
      keycloak.updateToken(70).then((refreshed: boolean) => {
        if (refreshed) {
          localStorage.setItem('token', keycloak.token as string);
          localStorage.setItem('token-refresh', keycloak.refreshToken as string);
        }
      }).catch(() => {
        console.log("Failed to refresh token");
      });
    }, 6000);
  })
  .catch(() => {
    console.log("Authentication failed");
  });
```

6.3 Σελίδες

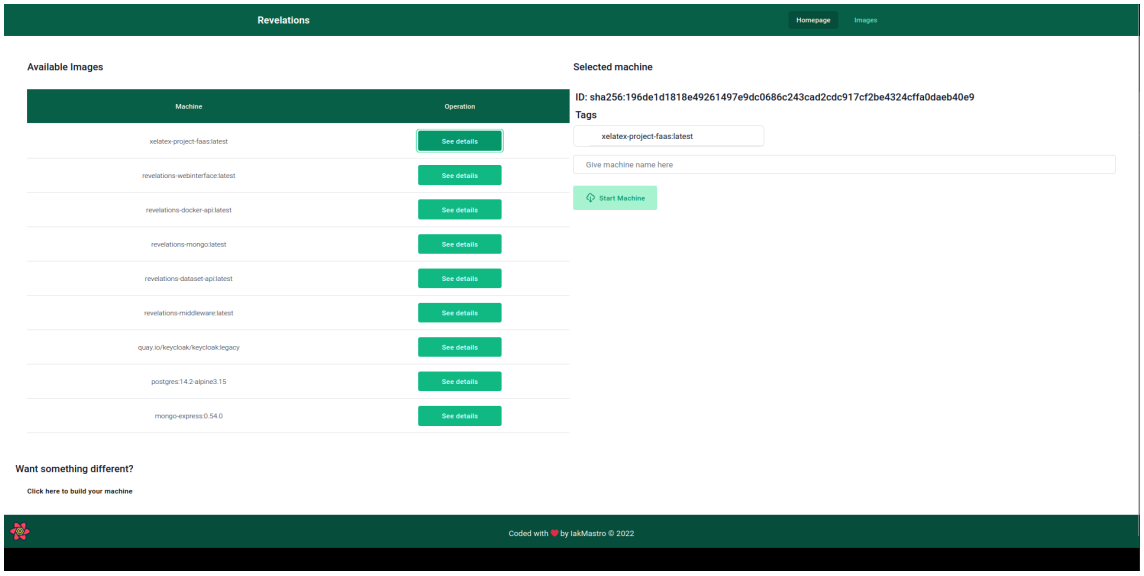
6.3.1 Αρχική σελίδα



Σχημα 6.1: Αρχική σελίδα πλατφόρμας

Η αρχική σελίδα είναι η πρώτη σελίδα που μπορεί να δει ο χρήστης. Βασικός της σκοπός είναι να δείξει στον τελικό χρήστη πληροφορίες σχετικά με τα containers, να μπορεί να επιλέξει ένα container για να δει τις πληροφορίες του ή να το σταματήσει και να ανεβάσει ένα δικό του dataset μέσω της φόρμας.

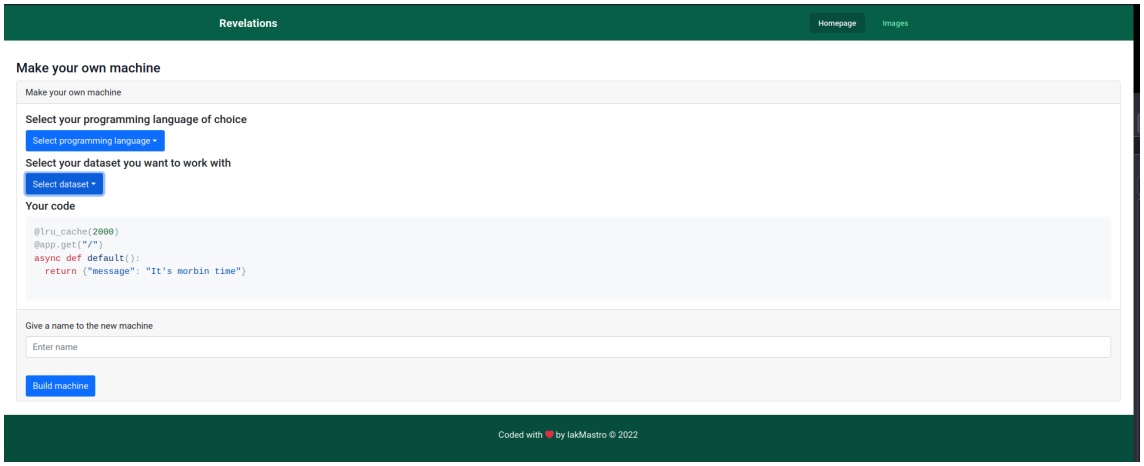
6.3.2 Σελίδα μηχανών



Σχημα 6.2: Σελίδα μηχανών πλατφόρμας

Στην σελίδα μηχανών, ο χρήστης μπορεί να δει όλα τα διαθέσιμα Docker images που είναι χτισμένα για το σύστημα, να διαλέξει ένα από αυτά και να το ανοίξει ως container δίνοντας του ένα όνομα. Τέλος, υπάρχει ένα κουμπί που τον μεταφέρει στην σελίδα νέας μηχανής.

6.3.3 Σελίδα δημιουργίας νέων μηχανών

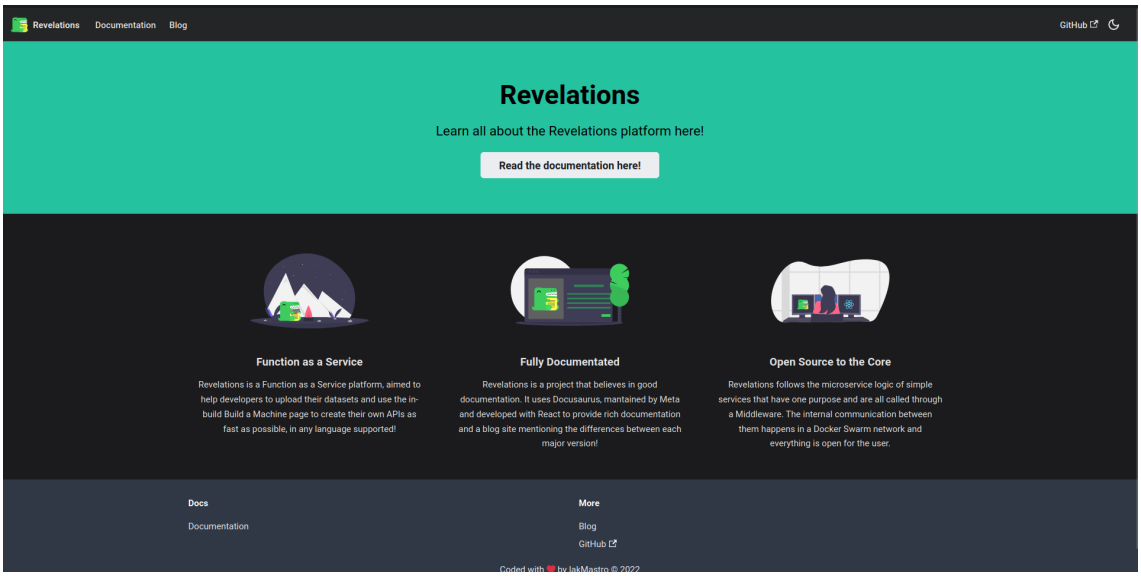


Σχημα 6.3: Σελίδα δημιουργίας νέων μηχανών πλατφόρμας

Η σελίδα δημιουργίας νέων μηχανών είναι η τελευταία σελίδα της πλατφόρμας στην οποία παρέχεται στον χρήστη μία πλατφόρμα που πρέπει να συμπληρώσει για να δημιουργήσει ένα νέο image. Έδω στο μέλλον θα μπορούσε να είχε και ένα πλαίσιο που παρουσιάζονται τα logs κατά την διάρκεια του build του image.

6.4 Τεχνική Σελίδα Εγγράφων

Μαζί με την πλατφόρμα, δημιουργήθηκε και μία ιστοσελίδα τεχνικών εγγράφων (documentation) για τρεις βασικές χρήσεις: Α) να παρέχει βοήθεια στους χρήστες ώστε να μπορούν να διαβάσουν πως να χρησιμοποιήσουν την κεντρική πλατφόρμα, Β) documentation για τους προγραμματιστές για το πως μπορούν να βοηθήσουν και να προσθέσουν δικό τους κώδικα εξηγώντας την αρχιτεκτονική και Γ) ένα βασικό μπλοκ στο οποίο θα αναφέρονται οι αλλαγές που πρόκειται να έχει η πλατφόρμα στο μέλλον.



Σχημα 6.4: Κεντρική σελίδα εγγράφων

Για την συγκεκριμένη σελίδα χρησιμοποιήθηκε το Docosaurus, το οποίο είναι ανεπτυγμένο στην NextJS. Τα άρθρα του blog και τα ίδια τα έγγραφα είναι γραμμένα σε Markdown.

Παράρτημα Α΄

Συμπεράσματα και Βελτιώσεις

Συμπερασματικά, ένα τέτοιο σύστημα έχει την δυνατότητα να γίνει πολύ δυνατό και επεκτάσιμο. Εάν γίνει μία μεγάλη πλατφόρμα στο μέλλον όπου οι πληροφορίες της θα είναι μη-κεντρικές και οι χρήστες ανώνυμοι, θα μπορούσε να χρησιμοποιηθεί κατά κύριο λόγο ως εκπαιδευτικό υλικό εκμάθησης ανάλυσης δεδομένων μεγάλης κλίμακας ή από έμπειρους χρήστες δημιουργώντας δοκιμασίες (challenges), όπως γίνεται και αντίστοιχα σε πλατφόρμες για εκμάθηση κυβερνοασφάλειας όπως το HackTheBox.

Βέβαια η πλατφόρμα, στην παρούσα μορφή, έχει μερικά θέματα που σίγουρα μπορούν να βελτιωθούν. Αρχικά, ένα σημαντικό μειονέκτημα είναι ότι δεν δόθηκε τόσο προσοχή στην ασφάλεια του συστήματος, που σημαίνει ότι εάν κάποιος έχει κακόβουλο σκοπό, υπάρχει και περίπτωση να καταφέρει να κάνει ζημιά στο σύστημα. Παρόλα αυτά είναι αξιοσημείωτο πως ο σκοπός της παρούσας εργασίας δεν ήταν η ανάπτυξη της μέγιστης ασφάλειας άλλα η διατήρηση ενός ικανοποιητικού επιπέδου. Σε αυτό το σημείο να αναφερθεί ότι επιπλέον ερεύνα πάνω στο κομμάτι της ασφάλειας θεωρείται απαραίτητη ώστε να ερευνηθούν τα κενά της παρούσας έρευνας.

Μία δεύτερη βελτίωση θα ήταν να υπάρχει περισσότερη αλληλεπίδραση μεταξύ του χρήστη και της πλατφόρμας. Δηλαδή, να του δείχνει τα logs του container είτε στην φάση Build είτε κατά την λειτουργία και να μπορεί να μπει μέσα στο container από την ίδια πλατφόρμα και όχι μόνο από εκτός. Για το πρώτο κομμάτι, θα μπορεί να χρησιμοποιηθεί ένα message broker σαν το Kafka ή το RabbitMQ για να μεταφέρονται μεγάλη κλίμακα δεδομένων από τον έναν κόμο στον άλλον γρήγορα προσπερνώντας το middleware. Έδω να επισημανθεί επίσης ότι δεν δίνεται τυχαίο port στο container που δημιουργείτε όπως ζητήθηκε, αλλά αυτό θα μπορούσε να αλλάξει στο μέλλον όταν μεγαλώσει το σύστημα.

Γενικά, το πείραμα μπορεί να χαρακτηριστεί ως πετυχημένο. Πρόσβαση στο κώδικα έχετε μέσω του **στο συγκεκριμένο repository στο GitHub**.

Βιβλιογραφία

[1] Dan Abramov. ***Redux Toolkit***. Η επίσημη ιστοσελίδα του Redux Toolkit. 2022. URL: https://redux-toolkit.js.org/.

[2] Flavio Copes. ***JWT authentication: Best practices and when to use it***. Καλές πρακτικές του JWT authentication. 2021. URL: https://blog.logrocket.com/jwt-authentication-best-practices/.

[3] Fluentd. ***Fluentd***. Πληροφορίες για τον συλλέκτη δεδομένων Fluentd. 2022. URL: https://www.fluentd.org/.

[4] D. Avison και G. Fitzgerald. ***Ανάπτυξη Πληροφοριακών Συσστήματων***, ***Μεθοδολογίες***, ***Τεχνικές και Εργαλεία***. 3 **edition**. Νέων Τεχνολογιών, 2017.

[5] Valentino Gagliardi. ***Jest Tutorial for Beginners: Getting Started With JavaScript Testing***. Άρθρο που διδάσκει πως να γίνονται tests σε NodeJS με την βιβλιοθήκη Jest. 2020. URL: https://www.valentinog.com/blog/jest/.

[6] Google. ***Golang***. Η επίσημη ιστοσελίδα της Golang. 2022. URL: https://go.dev/.

[7] A. S. Tanenbaum και H. Bos. ***Σύγχρονα Λειτουργικά Συστήματα***. 4 **edition**. Κλειδάριθμος, 2018.

[8] Docker Inc. ***Docker***. Η επίσημη ιστοσελίδα του Docker. 2022. URL: https://www.docker.com/.

[9] J. F. Kurose και K. W. Ross. ***Δικτύωση Υπολογιστών***, ***Προσέγγιση από Πάνω προς τα Κάτω***. 7 **edition**. M. Γκιούρδας, 2017.

[10] Tanner Linsley. ***React Query***. Η επίσημη ιστοσελίδα του React Query. 2020. URL: https://react-query-v3.tanstack.com/.

[11] A. S. Tanenbaum και M. van Steen. ***Κατανεμημένα Συστήματα***, ***Αρχές και Υποδείγματα***. Κλειδάριθμος, 2002.

[12] Meta. ***ReactJS***. Η επίσημη ιστοσελίδα της ReactJS. 2022. URL: https://reactjs.org/.

[13] Mozilla. ***Cross-Origin Resource Sharing (CORS)***. Πληροφορίες σχετικά με το CORS. 2022. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

[14] RedHat. ***Keycloak***. Η επίσημη ιστοσελίδα του Keycloak. 2022. URL: https://www.keycloak.org/.

[15] R. Elmasri και S. B. Navathe. ***Θεμελιώδεις Αρχές Συστημάτων Βάσεων Δεδομένων***. 7 **edition**. Διαυλός, 2016.

[16] Marcos Henrique da Silva. ***Building a Node.js/TypeScript REST API***. Άρθρο για την δημιουργία ένος API με την χρήση NodeJS και express. 2022.

[17] I. Sommerville. ***Βασικές Αρχές Τεχνολογιάς Λογισμικού***. 8 **edition**. Κλειδάριθμος, 2018.

[18] W. Stallings. ***Κρυπτογραφία Και Ασφάλεια Δικτύων, Αρχές και Εφαρμογές***. Ίων, 2011.

[19] Z. Mahmood και R. Puttini T. Erl. ***Cloud Computing - Αρχές, Τεχνολογία και Αρχιτεκτονική***. Μ. Γκιούρδας, 2013.

[20] Α. Αναγνωστόπουλος. ***Τεχνικό υλικό για το Swarmlab***. Το εργαστηριακό υλικό για τα μαθήματα του κύριου Αναγνωστόπουλου. 2022. URL: https://docs.swarmlab.io.

[21] Β. Μάμαλης και Α. Τομαράς Γ. Πάντζιου. ***Εισαγωγή στον Παράλληλο Υπολογισμό***. Νέων Τεχνολογιών, 2013.

[22] Β. Μάμαλης και Δ. Καλλέργης. ***Υπολογιστική Νέφους και Υπηρεσίες***. Το υλικό του μαθήματος για το ακαδημαϊκό έτος 2020-21. 2021. URL: https://eclass.uniwa.gr.

[23] Α. Κιούρτης. ***Διαχείριση Δεδομένων Μεγάλης Κλίμακας***. Το υλικό του μαθήματος για το ακαδημαϊκό έτος 2020-21. 2021. URL: https://eclass.uniwa.gr.

[24] Ν. Τσέλιος και Κ. Μουστάκας Ν. Αβούρης Χ. Κατσανός. ***Εισαγωγή στην Αλληλεπίδραση Ανθρώπου-Υπολογιστή***. 2 **edition**. Πανεπιστημιού Πάρτας, 2018.