



**ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ**

**ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ**

**ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ**

**Παρακολούθηση και Ανάλυση Συστήματος και Υπηρεσιών σε Περιβάλλον  
Υπολογιστικού Νέφους με Υπηρεσίες που βασίζονται σε Περιέκτες**

**ΕΥΑΓΓΕΛΙΝΟΣ ΕΛΕΥΘΕΡΙΟΣ**

141082

**Επιβλέπων Καθηγητής: Βασίλειος Μάμαλης  
Συν-επιβλέπων: Απόστολος Αναγνωστόπουλος**

Διπλωματική εργασία υποβληθείσα στο Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

ΑΙΓΑΛΕΩ, [Οκτώβριος/2021]

Πανεπιστήμιο Δυτικής Αττικής, Τμήμα Μηχανικών Πληροφορικής  
Ευαγγελινός Ελευθέριος  
© 2021 – Με την επιφύλαξη παντός δικαιώματος



**ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ**  
**ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ**

Η παρούσα διπλωματική εργασία παρουσιάστηκε

από τον

Ευαγγελινό Ελευθέριο

141082

την [8, Οκτωβρίου, 2021]

Η τριμελής επιτροπή:

Καντζάβελου Ιωάννα - Μάμαλης Βασίλης - Μπόγρης Αντώνης

Η έγκριση της διπλωματικής εργασίας δεν υποδηλοί την αποδοχή των γνώμών του συγγραφέα.  
Κατά τη συγγραφή τηρήθηκαν οι αρχές της ακαδημαϊκής δεοντολογίας.

## **Παρακολούθηση και Ανάλυση Συστήματος και Υπηρεσιών σε Περιβάλλον Υπολογιστικού Νέφους με Υπηρεσίες που βασίζονται σε Περιέκτες**

**Ευαγγελινός Ελευθέριος**

### **ΠΕΡΙΛΗΨΗ**

Αρχικά, πραγματοποιήθηκε μελέτη και ανάλυση του υπολογιστικού νέφους, των προμηθευτών υπηρεσιών νέφους, των μοντέλων υπηρεσιών τους, των αρχιτεκτονικών του υβριδικού νέφους και του πολυνέφους, όπως και της τεχνολογίας της εικονοποίησης (virtualization) σε παραλληλισμό με τους περιέκτες (containers). Στη συνέχεια, μελετήθηκαν τα καταναεμημένα συστήματα και η σχέση τους με το υπολογιστικό νέφος, ενώ δόθηκε ιδιαίτερη έμφαση στην ανεξαρτησία στο νέφος με αφορμή τον εγκλωβισμό που μπορεί να βιώσει ένας χρήστης στην προσπάθειά του να μεταβεί σε αυτό. Επιπροσθέτως, αναλύθηκε και επεξηγήθηκε η τοπολογία της αποκέντρωσης (decentralization), η σημαντικότητα και η πρακτικότητα του edge computing μέσα από τις υπηρεσίες του Cloudflare, όπως και τα πλεονεκτήματα μαζί με τις προκλήσεις των σμηνών (clusters) στο νέφος. Επίσης, πραγματοποιήθηκε μελέτη της αναγκαιότητας που υπάρχει στην παρακολούθηση, την συλλογή και την ανάλυση των δεδομένων καταγραφής σε περιβάλλον σμήνους, ενώ παρουσιάστηκαν κάποιες βέλτιστες πρακτικές προσέγγισης μίας εφαρμογής που συλλέγει και αναλύει αυτά τα δεδομένα. Επιπλέον, παρουσιάστηκαν τα εργαλεία και η σχεδίαση της τελικής εφαρμογής που αναπτύχθηκε. Τέλος, αναπτύχθηκε και επεξηγήθηκε εφαρμογή, που συλλέγει και αναλύει δεδομένα καταγραφής (logs) σε περιβάλλον, που το υποκείμενο πληροφοριακό σύστημα είναι βασισμένο σε dockerized υποδομή, με απώτερο σκοπό την παρακολούθηση του συστήματος αυτού και του ελέγχου της εύρυθμης λειτουργίας του.

Λέξεις κλειδιά: Υπολογιστική Νέφος, Περιέκτες, Παρακολούθηση, Docker, Δεδομένα Καταγραφής.

# System and Service Monitoring and Analysis in a Cloud Environment with Services based on Containers

Evangelinos Eleftherios

## ABSTRACT

Initially, the study and analysis of cloud computing, cloud service providers, their service models, hybrid cloud and multi-cloud architectures, as well as virtualization technology in parallel with containers, was carried out. Distributed systems and their relationship with the cloud were then studied, with particular emphasis on independence in the cloud in view of the lock-in that a user may experience when trying to switch to the cloud. In addition, the topology of decentralization, the importance and practicality of edge computing through Cloudflare services was analyzed and explained, as well as the advantages along with the challenges of clusters in the cloud. Also, a study of the necessity of monitoring, collecting and analyzing log data in a cluster environment was carried out, and some best practices of approaching an application that collects and analyzes this data were presented. In addition, the tools and design of the final application developed were presented. Finally, an application that collects and analyzes log data (logs) in an environment, where the underlying information system is based on a dockerized infrastructure was developed and explained, with the ultimate goal of monitoring the system and controlling its proper operation.

Keywords: Cloud computing, Containerization, Monitoring, Docker, Data logs.

**ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ ΚΑΙ ΕΙΚΟΝΩΝ**

Εικόνα 1: Τι αναλαμβάνει ο πάροχος σε σχέση με κάθε μοντέλο.....σελ.	13
Εικόνα 2: Απεικόνιση από την Microsoft όσον αφορά τις ευθύνες.....σελ.	14
Εικόνα 3: Απεικόνιση των επιπέδων που απαρτίζουν κάθε τεχνολογία ξεχωριστά.....σελ.	23
Εικόνα 4: Αριθμός υπηρεσιών στο AWS ανά τα έτη.[i.18].....σελ.	25
Εικόνα 5: Κατηγορίες στο AWS.....σελ.	26
Εικόνα 6: Σχετική πορεία του μεριδίου αγοράς σύμφωνα με την JPM .....σελ.	27
Εικόνα 7: Okta Investor Day FY21, April 1, 2020.....σελ.	34
Εικόνα 8: Παράδειγμα καταναμημένου συστήματος.....σελ.	38
Εικόνα 9: Παράδειγμα 2 Καταναμημένου συστήματος.....σελ.	38
Εικόνα 10: Παράδειγμα 1 από την σελίδα του Cloudflare.....σελ.	50
Εικόνα 11: Παράδειγμα worker από την σελίδα του Cloudflare.....σελ.	51
Εικόνα 12: Edge Computing.....σελ.	57
Εικόνα 13: Αρχιτεκτονική Docker.....σελ.	72
Εικόνα 14: Redis.....σελ.	80
Εικόνα 15: Παράδειγμα αρχείου docker-compose.yml.....σελ.	86
Εικόνα 16: Δήλωση υπηρεσίας-κοντέινερ mongo.....σελ.	87
Εικόνα 17: Παραμετροποίηση κοντέινερ fluentd.....σελ.	88
Εικόνα 18: Παραμετροποίηση κοντέινερ readmongo_service.....σελ.	90
Εικόνα 19: Παραμετροποίηση του κοντέινερ dummy_service.....σελ.	91
Εικόνα 20: Αποτυχία αυθεντικοποίησης.....σελ.	97
Εικόνα 21: Διαδικασία σύνδεσης στο swarmlab.io.....σελ.	97
Εικόνα 22: Απόκτηση token.....σελ.	97
Εικόνα 23: Διεπαφή web-client.....σελ.	102
Εικόνα 24: Λειτουργία ευρετηρίου.....σελ.	102
Εικόνα 25: Αποτέλεσμα ευρετηρίασης.....σελ.	103
Εικόνα 26: Highchart.....σελ.	108

Εικόνα 27: Πίνακας on-event απλοποιημένης μορφής.....σελ.	113
Εικόνα 28: Πίνακας on-event raw logs της MongoDB.....σελ.	114
Εικόνα 29: Πίνακας on-event raw logs.....σελ.	114
Εικόνα 30: Εκτέλεση Εντολής.....σελ.	116
Εικόνα 31: Αποτέλεσμα εισαγωγής container.....σελ.	117



**ΠΕΡΙΕΧΟΜΕΝΑ**

<b>ΠΕΡΙΛΗΨΗ.....</b>	<b>V</b>
<b>ABSTRACT.....</b>	<b>VI</b>
<b>ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ ΚΑΙ ΕΙΚΟΝΩΝ.....</b>	<b>VII</b>
<b>ΠΕΡΙΕΧΟΜΕΝΑ.....</b>	<b>IX</b>
<b>ΠΡΟΛΟΓΟΣ.....</b>	<b>1</b>
<b>1 ΕΙΣΑΓΩΓΗ.....</b>	<b>3</b>
<b>1.1 ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ.....</b>	<b>4</b>
<b>1.1.1 ΣΥΝΕΙΣΦΟΡΑ ΤΩΝ ΕΤΑΙΡΙΩΝ.....</b>	<b>4</b>
<b>1.2 ΛΟΓΙΚΗ ΚΑΙ ΧΡΗΣΗ.....</b>	<b>5</b>
<b>2 ΥΠΟΛΟΓΙΣΤΙΚΟ ΝΕΦΟΣ ΚΑΙ ΠΑΡΟΧΟΙ.....</b>	<b>7</b>
<b>2.1 ΒΑΣΙΚΑ ΜΟΝΤΕΛΑ ΥΠΗΡΕΣΙΩΝ.....</b>	<b>7</b>
<b>2.1.1 ΛΟΓΙΣΜΙΚΟ ΣΑΝ ΥΠΗΡΕΣΙΑ.....</b>	<b>8</b>
<b>2.1.2 ΠΛΑΤΦΟΡΜΑ ΣΑΝ ΥΠΗΡΕΣΙΑ.....</b>	<b>9</b>
<b>2.1.3 ΥΠΟΔΟΜΗ ΣΑΝ ΥΠΗΡΕΣΙΑ.....</b>	<b>12</b>
<b>2.1.4 SERVERLESS.....</b>	<b>14</b>
<b>2.2 ΥΒΡΙΔΙΚΟ ΝΕΦΟΣ ΚΑΙ ΠΟΛΥΝΕΦΟΣ.....</b>	<b>16</b>
<b>2.3 ΑΛΛΕΣ ΚΟΙΝΕΣ ΧΡΗΣΕΙΣ ΤΟΥ ΥΠΟΛΟΓΙΣΤΙΚΟΥ ΝΕΦΟΥΣ.....</b>	<b>17</b>
<b>2.4 ΕΙΚΟΝΟΠΟΙΗΣΗ ΚΑΙ ΠΕΡΙΕΚΤΕΣ.....</b>	<b>18</b>
<b>2.4.1 ΘΕΤΙΚΑ ΚΑΙ ΑΡΝΗΤΙΚΑ ΕΙΚΟΝΟΠΟΙΗΣΗΣ.....</b>	<b>18</b>

2.4.2 ΘΕΤΙΚΑ ΚΑΙ ΑΡΝΗΤΙΚΑ ΠΕΡΙΕΚΤΩΝ.....	21
2.5 ΠΑΡΟΧΟΙ ΤΟΥ ΥΠΟΛΟΓΙΣΤΙΚΟΥ ΝΕΦΟΥΣ.....	24
2.5.1 AMAZON WEB SERVICES.....	24
2.5.2 ΕΜΦΑΝΙΣΗ ΑΛΛΩΝ ΠΑΡΟΧΩΝ.....	26
2.5.3 ΕΜΦΑΝΙΣΗ ΤΟΥ ΠΟΛΥΝΕΦΟΥΣ.....	28
2.5.4 ΕΠΙΤΥΧΙΑ ΑΝΕΞΑΡΤΗΤΩΝ ΠΑΡΟΧΩΝ.....	31
3 ΑΝΕΞΑΡΤΗΣΙΑ - ΧΕΙΡΙΣΜΟΣ ΔΕΔΟΜΕΝΩΝ ΣΤΟ ΝΕΦΟΣ.....	37
3.1 ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ.....	37
3.2 ΑΝΕΞΑΡΤΗΣΙΑ ΣΤΟ ΝΕΦΟΣ.....	39
3.2.1 ΤΕΡΜΑΤΙΣΜΟΣ ΥΠΗΡΕΣΙΩΝ.....	39
3.2.2 ΑΝΕΞΑΡΤΗΤΗ ΠΡΟΣΒΑΣΗ ΣΤΑ ΔΕΔΟΜΕΝΑ.....	40
3.2.3 ΑΝΕΞΑΡΤΗΤΑ ΕΡΓΑΛΕΙΑ ΤΡΙΤΩΝ & SLA.....	41
3.2.4 ΟΥΔΕΤΕΡΕΣ ΤΕΧΝΟΛΟΓΙΕΣ CLOUD.....	41
3.3 ΑΠΟΚΕΝΤΡΩΣΗ.....	42
3.3.1 ΠΛΕΟΝΕΚΤΗΜΑΤΑ - ΜΕΙΟΝΕΚΤΗΜΑΤΑ.....	44
3.3.2 ΧΡΗΜΑΤΟΔΟΤΗΣΗ ΚΑΙ ΚΕΡΔΟΦΟΡΙΑ.....	47
3.3.3 ΕΛΕΥΘΕΡΟ ΛΟΓΙΣΜΙΚΟ.....	47
3.4 CLOUDFLARE ΚΑΙ EDGE COMPUTING.....	48
3.4.1 ΕΛΕΥΘΕΡΙΑ ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗΣ.....	49
3.4.2 ΠΛΕΟΝΕΚΤΗΜΑΤΑ.....	51
3.4.3 JAVASCRIPT.....	52
3.4.4 ΤΟ EDGE COMPUTING.....	53

<b>3.5 ΣΜΗΝΟΣ ΣΤΟ ΝΕΦΟΣ.....</b>	<b>58</b>
<b>3.5.1 ΠΛΕΟΝΕΚΤΗΜΑΤΑ.....</b>	<b>59</b>
<b>3.5.2 ΠΡΟΚΛΗΣΕΙΣ ΣΜΗΝΟΥΣ.....</b>	<b>62</b>
<b>3.5.3 ΠΕΡΙΕΚΤΕΣ ΚΑΙ ΣΜΗΝΟΣ.....</b>	<b>62</b>
<b>3.6 ΣΥΛΛΟΓΗ ΚΑΙ ΠΑΡΑΚΟΛΟΥΘΗΣΗ ΔΕΔΟΜΕΝΩΝ.....</b>	<b>63</b>
<b>3.6.1 ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΠΑΡΑΚΟΛΟΥΘΗΣΗΣ.....</b>	<b>64</b>
<b>3.6.2 ΒΕΛΤΙΣΤΕΣ ΠΡΑΚΤΙΚΕΣ.....</b>	<b>65</b>
<b>4 ΣΧΕΔΙΑΣΜΟΣ – ΥΠΟΣΤΗΡΙΚΤΙΚΑ ΕΡΓΑΛΕΙΑ.....</b>	<b>69</b>
<b>4.1 DOCKER.....</b>	<b>69</b>
<b>4.1.1 ΛΟΓΟΙ ΧΡΗΣΗΣ ΤΟΥ DOCKER.....</b>	<b>70</b>
<b>4.1.2 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ DOCKER.....</b>	<b>71</b>
<b>4.1.3 Ο DAEMON ΤΟΥ DOCKER.....</b>	<b>72</b>
<b>4.1.4 Ο CLIENT ΤΟΥ DOCKER.....</b>	<b>72</b>
<b>4.1.5 ΜΗΤΡΩΑ DOCKER.....</b>	<b>72</b>
<b>4.1.6 ΑΝΤΙΚΕΙΜΕΝΑ DOCKER.....</b>	<b>73</b>
<b>4.1.7 ΓΙΑΤΙ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ ΤΟ DOCKER.....</b>	<b>74</b>
<b>4.2 NODEJS.....</b>	<b>74</b>
<b>4.2.1 ΓΙΑΤΙ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ ΤΟ NODEJS.....</b>	<b>75</b>
<b>4.3 FLUENTD.....</b>	<b>75</b>
<b>4.3.1 ΕΝΟΠΟΙΗΜΕΝΗ ΚΑΤΑΓΡΑΦΗ ΜΕ JSON.....</b>	<b>76</b>
<b>4.3.2 PLUGGABLE ΑΡΧΙΤΕΚΤΟΝΙΚΗ.....</b>	<b>76</b>
<b>4.3.3 ΕΛΑΧΙΣΤΟΙ ΑΠΑΙΤΟΥΜΕΝΟΙ ΠΟΡΟΙ.....</b>	<b>76</b>

---

4.3.4 ΕΝΣΩΜΑΤΟΜΕΝΗ ΑΞΙΟΠΙΣΤΙΑ.....	76
4.3.5 ΠΛΕΟΝΕΚΤΗΜΑΤΑ.....	77
4.3.6 ΓΙΑΤΙ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ ΤΟ FLUENTD.....	78
4.4 VUE.JS.....	78
4.4.1 ΓΙΑΤΙ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ ΤΟ VUE.JS.....	78
4.5 SOCKET.IO.....	79
4.5.1 ΓΙΑΤΙ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ ΤΟ SOCKET.IO.....	79
4.6 REDIS.....	80
4.6.1 ΓΙΑΤΙ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ ΤΟ REDIS.....	81
4.7 MONGODB.....	81
4.7.1 ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΧΡΗΣΗΣ.....	81
5 ΥΛΟΠΟΙΗΣΗ-ΠΕΡΙΓΡΑΦΗ ΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑΣ.....	83
5.1 DOCKERFILE ΚΑΙ DOCKER-COMPOSE.....	83
5.2 ΔΗΜΙΟΥΡΓΙΑ ΕΙΚΟΝΩΝ.....	84
5.3 ΠΕΡΙΕΚΤΕΣ ΤΗΣ ΕΦΑΡΜΟΓΗΣ.....	86
5.3.1 MONGO.....	86
5.3.2 FLUENTD.....	88
5.3.3 REDISSERVER.....	89
5.3.4 READMONGO_SERVICE.....	89
5.3.5 DUMMY_SERVICE.....	91
5.4 ΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑ-ΑΝΑΛΥΣΗ ΥΠΗΡΕΣΙΩΝ.....	92
5.4.1 FLUENTD.....	92

---

5.4.2 READMONGO_SERVICE ΚΑΙ WEB-CLIENT.....	94
5.4.3 ΣΕΝΑΡΙΟ ΧΡΗΣΗΣ.....	115
6 ΣΥΜΠΕΡΑΣΜΑΤΑ-ΕΠΕΚΤΑΣΕΙΣ.....	119
6.1 ΕΠΕΚΤΑΣΕΙΣ ΕΦΑΡΜΟΓΗΣ.....	120
ΒΙΒΛΙΟΓΡΑΦΙΑ.....	122

## ΠΡΟΛΟΓΟΣ

Αρχικά, θα ήθελα να ευχαριστήσω τον κύριο Απόστολο Αναγνωστόπουλο για την διαρκή τεχνική υποστήριξη, που μου προσέφερε καθ' όλη τη διάρκεια της εκπόνησης της εργασίας, αλλά και για την διδακτική του διάθεση όταν ακόμα εκπαιδευόμενοι πάνω στα εργαλεία που εν τέλει χρησιμοποιήθηκαν στα πλαίσια της διπλωματικής εργασίας. Επιπλέον, θα ήθελα να τον ευχαριστήσω, που με βοήθησε να καταλάβω, πως ο συγκεκριμένος τομέας που πραγματεύεται η εργασία, είναι ένας τομέας στην “άκρη του ξυραφιού”, δηλαδή ένας τομέας που μου έμαθε, όχι μόνο πώς λειτουργούν σήμερα τα πιο προηγμένα πληροφοριακά συστήματα, αλλά και πώς θα λειτουργούν στα επόμενα χρόνια. Επιπλέον, θα ήθελα να ευχαριστήσω τον κύριο Βασίλειο Μάμαλη, που μου έδωσε την ελευθερία να μελετήσω τη θεωρία του αντικείμενου έξω από πολύ στενά πλαίσια, αλλά και για την γενικότερη ευκαιρία που μου έδωσε να συνεργαστώ μαζί του, πράγμα που με βοήθησε να εξελιχθώ και να αποκτήσω εφόδια, που δεν θα είχα αποκτήσει διαφορετικά. Τέλος, ένα τεράστιο ευχαριστώ στην οικογένειά μου, που με στήριξαν καθ' όλη τη διάρκεια αυτού του ακαδημαϊκού ταξιδιού, ακολουθώντας πιστά το όνειρό μου να σπουδάσω το αντικείμενο που ονειρευόμουν από τα εφηβικά μου χρόνια.



## 1 ΕΙΣΑΓΩΓΗ

Με την πάροδο των χρόνων η τεχνολογία κάνει άλματα. Ειδικά στον τομέα της πληροφορικής και των υπολογιστών πάντα υπάρχει επιτακτική ανάγκη για εξέλιξη. Μια χαρακτηριστική τέτοια ανάγκη παρουσιάστηκε πολύ νωρίς στον κλάδο της πληροφορικής και ήταν η ανάγκη για υπηρεσίες οι οποίες θα παρέχονταν μέσω του διαδικτύου. Στον τομέα του cloud computing ή αλλιώς του υπολογιστικού νέφους, βλέπουμε τα πρώτα βήματα, που αποτέλεσαν σταθμό στην ιστορία του, να πρωτοεμφανίζονται στις αρχές του 1960. Η έννοια του υπολογιστικού νέφους, υπήρξε άγνωστη ως τότε στην επιστήμη της πληροφορικής.

Το Υπολογιστικό Νέφος όπως το γνωρίζουμε σήμερα χρησιμοποιεί δικτυακές συνδέσεις ώστε να παρέχει υπηρεσίες, άκρως σημαντικές για τους μεγαλύτερους οργανισμούς της επιστήμης της Πληροφορικής. Οι υπηρεσίες αυτές χωρίζονται σε τρία βασικά μοντέλα υπηρεσιών, που είναι το Λογισμικό σαν Υπηρεσία (Software as a Service(SaaS)), οι Πλατφόρμες σαν Υπηρεσία (Platforms as a Service(PaaS)) και οι Υποδομές σαν Υπηρεσία (Infrastructures as a Service)).[1.1]

Στα επόμενα κεφάλαια θα αναλυθούν εκτενέστερα αυτά τα μοντέλα, αλλά, πριν από αυτό θα γίνει μία σύντομη ιστορική αναδρομή για το Υπολογιστικό Νέφος. Έπειτα, θα ακολουθήσει η περιγραφή της λειτουργικότητας ενός τέτοιου συστήματος, ποιοι είναι οι λόγοι που μας κατευθύνουν στη χρήση του, τα θετικά και τα αρνητικά του, οι τεχνολογίες που εκμεταλλεύεται με ιδιαίτερη έμφαση στους περιέκτες(containers) και κατ' επέκταση στο Docker, τι είναι ένα κατανεμημένο σύστημα και πως αυτό σχετίζεται με ένα υπολογιστικό νέφος, τι είναι το edge computing και παραδείγματα τέτοιων συστημάτων, όπως το Cloudflare Workers, η μελέτη στα σμήνη στο νέφος και οι προκλήσεις της συλλογής και παρακολούθησης των δεδομένων, ενώ στο τέλος, θα παρουσιαστεί μία εφαρμογή που υλοποιήθηκε στα πλαίσια αυτής της εργασίας και των τεχνολογιών που θα έχουν περιγραφεί.



## 1.1 ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ

Ο όρος του Υπολογιστικού Νέφους έγινε ευρύτερα γνωστός τη δεκαετία του 2000, όταν η Amazon δημιούργησε τη θυγατρική Amazon Web Services, με στόχο να "δώσει τη δυνατότητα στους προγραμματιστές να δημιουργήσουν καινοτόμες και επιχειρηματικές εφαρμογές μόνοι τους". Τον Μάρτιο του 2006 η Amazon παρουσίασε το Simple Storage Service (S3), ενώ τον Αύγουστο του ίδιου έτους ακολούθησε το Elastic Compute Cloud (EC2). Τα προϊόντα αυτά ήταν πρωτοπόρα στη χρήση της εικονικοποίησης διακομιστών για την παροχή IaaS σε φθηνότερη και κατά παραγγελία τιμολογιακή βάση. [i.2]

### 1.1.1 ΣΥΝΕΙΣΦΟΡΑ ΤΩΝ ΕΤΑΙΡΙΩΝ

Οι κολοσσοί της πληροφορικής δεν άργησαν να εκμεταλλευτούν το νέο αυτό ρεύμα και να κάνουν τα μεγαλύτερα βήματα στην εξέλιξη του Υπολογιστικού Νέφους. Τον Απρίλιο του 2008, η Google κυκλοφόρησε την beta έκδοση του Google App Engine. Το App Engine ήταν ένα PaaS (ένα από τα πρώτα του είδους του) που παρείχε πλήρως συντηρούμενη υποδομή και μια πλατφόρμα ανάπτυξης για τους χρήστες ώστε να δημιουργούν διαδικτυακές εφαρμογές χρησιμοποιώντας κοινές γλώσσες/τεχνολογίες όπως η Python, η Node.js και η PHP. Ο στόχος ήταν να εξαλειφθεί η ανάγκη για ορισμένες διοικητικές εργασίες που είναι τυπικές για ένα μοντέλο IaaS, ενώ παράλληλα να δημιουργηθεί μια πλατφόρμα όπου οι χρήστες θα μπορούσαν εύκολα να αναπτύσσουν τέτοιες εφαρμογές και να τις κλιμακώνουν ανάλογα με τη ζήτηση. [i.2]

Επιπλέον, στις αρχές του 2008, το Nebula της NASA, το οποίο βελτιώθηκε στο πλαίσιο του έργου RESERVOIR που χρηματοδοτείται από την Ευρωπαϊκή Επιτροπή, έγινε το πρώτο λογισμικό ανοικτού κώδικα για την ανάπτυξη ιδιωτικών και υβριδικών νεφών, καθώς και για την ομοσπονδία νεφών. Στα μέσα του 2008, η Gartner είδε μια ευκαιρία για το cloud computing "να διαμορφώσει τη σχέση μεταξύ των καταναλωτών υπηρεσιών πληροφορικής, εκείνων που χρησιμοποιούν υπηρεσίες πληροφορικής και εκείνων που τις πωλούν" και παρατήρησε ότι "οι οργανισμοί μεταβαίνουν από τα περιουσιακά στοιχεία υλικού και λογισμικού που ανήκουν στην εταιρεία σε μοντέλα που βασίζονται σε υπηρεσίες ανά χρήση", έτσι ώστε η "προβλεπόμενη στροφή στην πληροφορική ... θα οδηγήσει σε δραματική αύξηση των προϊόντων πληροφορικής σε ορι-

σμένους τομείς και σε σημαντικές μειώσεις σε άλλους τομείς". Το 2008, το Εθνικό Ίδρυμα Επιστημών των ΗΠΑ ξεκίνησε το πρόγραμμα Cluster Exploratory για τη χρηματοδότηση ακαδημαϊκής έρευνας που χρησιμοποιεί την τεχνολογία cluster της Google-IBM για την ανάλυση τεράστιου όγκου δεδομένων. [i.2]

Μπαίνοντας στη δεκαετία του 2010, βλέπουμε τον Φεβρουάριο του 2010, η Microsoft να κυκλοφορεί το Microsoft Azure, το οποίο είχε ανακοινωθεί τον Οκτώβριο του 2008. Τον Ιούλιο του 2010, η Rackspace Hosting και η NASA εγκαινίασαν από κοινού μια πρωτοβουλία λογισμικού νέφους ανοικτού κώδικα, γνωστή ως OpenStack. Το έργο OpenStack είχε ως στόχο να βοηθήσει τους οργανισμούς που προσφέρουν υπηρεσίες υπολογιστικού νέφους που εκτελούνται σε τυποποιημένο υλικό. Ο πρώιμος κώδικας προερχόταν από την πλατφόρμα Nebula της NASA καθώς και από την πλατφόρμα Cloud Files της Rackspace. Ως προσφορά ανοικτού κώδικα και μαζί με άλλες λύσεις ανοικτού κώδικα, όπως το CloudStack, το Ganeti και το OpenNebula, έχει προσελκύσει την προσοχή αρκετών βασικών κοινοτήτων. Αρκετές μελέτες στοχεύουν στη σύγκριση αυτών των προσφορών ανοικτού κώδικα με βάση ένα σύνολο κριτηρίων. Επιπροσθέτως την 1η Μαρτίου 2011, η IBM ανακοίνωσε το πλαίσιο IBM SmartCloud για την υποστήριξη του Smarter Planet, ενώ Στις 7 Ιουνίου 2012, η Oracle ανακοίνωσε το Oracle Cloud. Τον Μάιο του 2012, το Google Compute Engine κυκλοφόρησε σε προεπισκόπηση, πριν τεθεί σε γενική διαθεσιμότητα τον Δεκέμβριο του 2013. Τέλος, το 2019, το Linux ήταν το πιο συνηθισμένο λειτουργικό σύστημα που χρησιμοποιείται στο Microsoft Azure και τον Δεκέμβριο του 2019, η Amazon ανακοίνωσε το AWS Outposts, το οποίο είναι μια πλήρως διαχειρίσιμη υπηρεσία που επεκτείνει την υποδομή AWS, τις υπηρεσίες AWS, τα API και τα εργαλεία σε σχεδόν οποιοδήποτε κέντρο δεδομένων, χώρο συνεγκατάστασης ή εγκατάσταση στις εγκαταστάσεις του πελάτη για μια πραγματικά συνεπή υβριδική εμπειρία.[i.2]

## 1.2 ΛΟΓΙΚΗ ΚΑΙ ΧΡΗΣΗ

Όπως ήδη αναφέρθηκε μία λογική του Υπολογιστικού Νέφους είναι να ελαχιστοποιήσει την επιβάρυνση που ασκείται στις υπολογιστικές συσκευές, είτε αυτή είναι ένας προσωπικός υπολογιστής ή ένα έξυπνο τηλέφωνο, από τον υπολογιστικό φόρτο και τους πόρους που

απαιτούνται σήμερα για την παροχή πολλών υπηρεσιών. Η χρήση του Υπολογιστικού Νέφους παρατηρεί συνεχώς όλο και μεγαλύτερη ζήτηση λόγω του χαμηλού κόστους, της ανώτερης προσαρμοστικότητας που διαθέτει, της ελαστικότητας και της βέλτιστης διαχείρισης πόρων.[i.3]

Στα πλαίσια της χρήσης του Υπολογιστικού Νέφους, όπως αναφέρει και η ομάδα της IBM[i.3], υπάρχουν κάποιες περιπτώσεις που βελτιώνουν σημαντικά της ικανότητα μίας επιχείρησης να πετύχει τους επιχειρηματικούς στόχους της. Στο επόμενο κεφάλαιο θα δούμε αναλυτικότερα κάποιες από αυτές τις περιπτώσεις και πώς ωφελούνται από αυτές οι μικρές αλλά και οι μεγάλες επιχειρήσεις.

## 2 ΥΠΟΛΟΓΙΣΤΙΚΟ ΝΕΦΟΣ ΚΑΙ ΠΑΡΟΧΟΙ

Νωρίτερα, κατά την εισαγωγή αναφέρθηκαν τρία κοινά μοντέλα παροχών αναφορικά με το υπολογιστικό νέφος. Τα τρία αυτά μοντέλα, σύμφωνα με την IBM[i.4], είναι το **IaaS (Infrastructure as a Service)**, το **PaaS (Platform as a Service)**, και το **SaaS (Software as a Service)**. Επιπλέον, υπάρχει και το **Serverless** μοντέλο το οποίο αποτελεί μία εξίσου ισχυρή εναλλακτική. Πλέον, η χρήση αυτών των μοντέλων από κάθε εταιρία είναι τόσο διαδεδομένη που η συντριπτική πλειοψηφία των εφαρμογών που “τρέχουν” σε έναν προσωπικό υπολογιστή ή μία “έξυπνη” κινητή συσκευή, χρησιμοποιεί στο υπόβαθρό της κάποια μορφή υπολογιστικού νέφους. Αναφορικά κάποια παραδείγματα είναι το Gmail, το Dropbox και το Netflix. [i.4]

Μια πρόσφατη έρευνα έδειξε, πως η μετάβαση προς το υπολογιστικό νέφος έχει ήδη κριθεί σχεδόν απαραίτητη με τα στατιστικά να μιλάν από μόνα τους. Η έρευνα έδειξε πως το 92% των οργανισμών έχει μεταφέρει ήδη το πληροφοριακό της περιβάλλον (υποδομή, εφαρμογές, αναλύσεις δεδομένων, κλπ.) στο νέφος ως ένα βαθμό, ενώ αναμενόταν το ποσοστό να αυξηθεί στο 95% σε ενάμιση χρόνο από την ημερομηνία που είχε γίνει η έρευνα στις 18 Ιουνίου του 2020. [i.12]

### 2.1 ΒΑΣΙΚΑ ΜΟΝΤΕΛΑ ΥΠΗΡΕΣΙΩΝ

Σε αυτό το υποκεφάλαιο θα αναλυθούν τα σημαντικότερα μοντέλα υπηρεσιών cloud, που χρησιμοποιούνται εν σήμερον ημέρα σε κάθε πάροχο cloud στην αγορά για την διανομή διάφορων υπηρεσιών. Τα πλεονεκτήματα και τα μειονεκτήματα είναι γεγονός για κάθε ένα από αυτά, ενώ οι δυνατότητες που παρέχονται τα καθιστούν κατάλληλα σε συγκεκριμένες περιπτώσεις το καθένα όπως θα δούμε και παρακάτω.

### 2.1.1 ΛΟΓΙΣΜΙΚΟ ΣΑΝ ΥΠΗΡΕΣΙΑ

Το μοντέλο του λογισμικού σαν υπηρεσία (**SaaS (Software as a Service)**) μπορεί να περιγραφεί ως ένα λογισμικό το οποίο λαμβάνει χώρα στο νέφος, που μπορείς να έχεις πρόσβαση σε αυτό μέσω ενός web browser, ενός αποκλειστικού desktop client, ή μπορεί να είναι ένα API που αλληλεπιδρά με κάποιο λειτουργικό σύστημα. Στις περισσότερες των περιπτώσεων αυτό το μοντέλο υπηρεσιών προσφέρεται στους χρήστες μέσω ετήσιας ή μηνιαίας συνδρομής, με κάποιες εξαιρέσεις που η χρέωση εξαρτάται από το ποσοστό χρήσης των υπηρεσιών. [i.4]

Όπως αναφέρεται και στην ιστοσελίδα της TechTarget από τον Wesley Chai, οι εφαρμογές και οι υπηρεσίες SaaS χρησιμοποιούν συνήθως μια προσέγγιση πολλαπλών μισθωτών, πράγμα που σημαίνει ότι ένα μόνο στιγμιότυπο της εφαρμογής SaaS θα εκτελείται στους κεντρικούς διακομιστές και αυτό το μοναδικό στιγμιότυπο θα εξυπηρετεί κάθε συνδρομητή πελάτη ή μισθωτή cloud. Η εφαρμογή θα εκτελείται σε μία ενιαία έκδοση και διαμόρφωση για όλους τους πελάτες ή ενοικιαστές. Παρόλο που διαφορετικοί συνδρομητές πελάτες θα τρέχουν στην ίδια περίπτωση cloud με κοινή υποδομή και πλατφόρμα, τα δεδομένα από διαφορετικούς πελάτες θα εξακολουθούν να διαχωρίζονται.[i.5]

Εκτός των πλεονεκτημάτων που προσφέρει η μετάβαση στο Νέφος, όπως η μείωση των δαπανών, της προσαρμοστικότητας, και την αναλογία της αξίας της ως προς τη δαπάνη του χρόνου, το μοντέλο του Λογισμικού σαν Υπηρεσία προσφέρει επιπροσθέτως τα εξής:[i.4]

- **Αυτόματες αναβαθμίσεις:** Με το συγκεκριμένο μοντέλο, ο χρήστης δεν χρειάζεται να μπει στην διαδικασία να κάνει πλάνο για την αναβάθμιση του λογισμικού του, αφού μπορεί να εκμεταλλευτεί το γεγονός ότι η αναβάθμιση των υπηρεσιών είναι στο χέρι του παρόχου.[i.4]
- **Απώλεια δεδομένων:** Με το λογισμικό να είναι εξ ολοκλήρου στο Νέφος, ο χρήστης δεν χρειάζεται πλέον να ανησυχεί για την απώλεια των δεδομένων του, σε περίπτωση που η συσκευή του χαλάσει ή δεχθεί κάποιο crash.[i.4]
- **Ευέλικτες πληρωμές.** Αντί να αγοράζουν λογισμικό για εγκατάσταση ή πρόσθετο υλικό για την υποστήριξή του, οι πελάτες εγγράφονται σε μια προσφορά SaaS. Η μετάβαση του κόστους σε μια επαναλαμβανόμενη λειτουργική δαπάνη επιτρέπει σε πολλές επιχειρήσεις να ασκούν καλύτερο και πιο προβλέψιμο προϋπολογισμό. Οι χρήστες μπορούν επίσης να

τερματίσουν τις προσφορές SaaS ανά πάσα στιγμή για να σταματήσουν αυτές τις επαναλαμβανόμενες δαπάνες. [i.5]

- **Κλιμακούμενη χρήση.** Οι υπηρεσίες νέφους όπως το SaaS προσφέρουν υψηλή κάθετη επεκτασιμότητα, η οποία δίνει στους πελάτες τη δυνατότητα να έχουν πρόσβαση σε περισσότερες ή λιγότερες υπηρεσίες ή χαρακτηριστικά κατά παραγγελία.[i.5]
- **Προσβασιμότητα και ανθεκτικότητα.** Δεδομένου ότι οι προμηθευτές SaaS παρέχουν τις εφαρμογές μέσω του διαδικτύου, οι χρήστες μπορούν να έχουν πρόσβαση σε αυτές από οποιαδήποτε συσκευή και τοποθεσία με δυνατότητα πρόσβασης στο διαδίκτυο.[i.5]
- **Προσαρμογή.** Οι εφαρμογές SaaS είναι συχνά παραμετροποιήσιμες και μπορούν να ενσωματωθούν με άλλες επιχειρηματικές εφαρμογές, ιδίως σε εφαρμογές από έναν κοινό πάροχο λογισμικού.[i.5]

Το μοντέλο του Λογισμικού ως Υπηρεσία διαθέτει εκατοντάδες χιλιάδες εναλλακτικές λύσεις, από στοχευμένες εφαρμογές μικρής κλίμακας, μέχρι ισχυρά λογισμικά βάσεων δεδομένων σε επίπεδο επιχειρήσεων, ενώ μπορεί να προσφέρει ακόμα και λογισμικά τεχνητής νοημοσύνης. [i.4]

## 2.1.2 ΠΛΑΤΦΟΡΜΑ ΣΑΝ ΥΠΗΡΕΣΙΑ

Το μοντέλο της πλατφόρμας σαν υπηρεσία (**PaaS (Platform as a Service)**) προορίζεται κυρίως για προγραμματιστές λογισμικών με έτοιμες πλατφόρμες οι οποίες παρέχουν κατάλληλο υλικό, λογισμικό, υποδομή, ακόμα και ειδικά εργαλεία για εκτέλεση, ανάπτυξη, και διαχείριση εφαρμογών με χαμηλό κόστος, άνευ πολυπλοκότητας, και δίχως την ανάγκη να συντηρείς και να αναβαθμίζεις τοπικά το εργαλείο στην συσκευή του χρήστη.[i.4] Παρόλο που το PaaS έχει κάποιες ομοιότητες με το serverless computing, υπάρχουν πολλές κρίσιμες διαφορές μεταξύ τους. Το PaaS και το serverless computing μοιάζουν στο ότι και για τα δύο, το μόνο για το οποίο πρέπει να ανησυχεί ο προγραμματιστής είναι να γράφει και να ανεβάζει κώδικα, ενώ ο προμηθευτής χειρίζεται όλες τις διαδικασίες backend. Ωστόσο, η κλιμάκωση είναι πολύ διαφορετική όταν χρησιμοποιούνται τα δύο μοντέλα. Οι εφαρμογές που έχουν κατασκευαστεί με τη χρήση serverless computing ή FaaS, θα κλιμακώνονται αυτόματα, ενώ οι εφαρμογές PaaS όχι, εκτός αν έχουν προγραμματιστεί έτσι. Οι χρόνοι εκκίνησης διαφέρουν επίσης σε μεγάλο

βαθμό- οι εφαρμογές χωρίς διακομιστή μπορούν να είναι έτοιμες και να λειτουργούν σχεδόν αμέσως, αλλά οι εφαρμογές PaaS μοιάζουν περισσότερο με τις παραδοσιακές εφαρμογές και πρέπει να λειτουργούν τις περισσότερες φορές ή όλο το χρόνο προκειμένου να είναι άμεσα διαθέσιμες για τους χρήστες. Μια άλλη διαφορά είναι ότι οι πωλητές serverless δεν παρέχουν εργαλεία ανάπτυξης ή πλαίσια, όπως οι πωλητές PaaS. Και τέλος, η τιμολόγηση διαχωρίζει τα δύο μοντέλα. Η τιμολόγηση του PaaS δεν είναι όσο ακριβής είναι στο serverless computing, όπου οι χρεώσεις κατανέμονται στον αριθμό των δευτερολέπτων ή των κλασμάτων του δευτερολέπτου που εκτελείται κάθε περίπτωση μιας λειτουργίας.[i.6]

Με αυτό το μοντέλο ο πάροχος φιλοξενεί τα πάντα. Εξυπηρετητές, δίκτυα, αποθηκευτικός χώρος, λογισμικό λειτουργικών συστημάτων, ενδιάμεσο λογισμικό και βάσεις δεδομένων βρίσκονται όλα στα κέντρα δεδομένων του.[i.4] Τα διάφορα πλεονεκτήματα του συγκεκριμένου μοντέλου διευρύνονται σε πολλούς τομείς.

- **Γρηγορότερος χρόνος διάθεσης στην αγορά:** Το PaaS χρησιμοποιείται για τη δημιουργία εφαρμογών ταχύτερα από ό,τι θα ήταν δυνατό αν οι προγραμματιστές έπρεπε να ανησυχούν για την κατασκευή, τη παραμετροποίηση και την παροχή των δικών τους πλατφορμών και υποδομών backend. Με το PaaS, το μόνο που χρειάζεται να κάνουν είναι να γράψουν τον κώδικα και να δοκιμάσουν την εφαρμογή και ο προμηθευτής αναλαμβάνει τα υπόλοιπα.[i.6]
- **Ένα περιβάλλον από την αρχή έως το τέλος:** Το PaaS επιτρέπει στους προγραμματιστές να δημιουργούν, να δοκιμάζουν, να επιδιορθώνουν, να αναπτύσσουν, να φιλοξενούν και να ενημερώνουν τις εφαρμογές τους στο ίδιο περιβάλλον. Αυτό επιτρέπει στους προγραμματιστές να είναι σίγουροι ότι μια εφαρμογή ιστού θα λειτουργεί σωστά όπως φιλοξενείται πριν την κυκλοφορία της και απλοποιεί τον κύκλο ζωής της ανάπτυξης εφαρμογών.[i.6]
- **Κόστος:** Το PaaS είναι πιο αποδοτικό από την αξιοποίηση του IaaS σε πολλές περιπτώσεις. Τα γενικά έξοδα μειώνονται επειδή οι πελάτες του PaaS δεν χρειάζεται να διαχειρίζονται και να παρέχουν εικονικές μηχανές. Επιπλέον, ορισμένοι πάροχοι διαθέτουν δομή τιμολόγησης pay-as-you-go, στην οποία ο προμηθευτής χρεώνει μόνο τους υπολογιστικούς πόρους που χρησιμοποιούνται από την εφαρμογή, εξοικονομώντας συνήθως χρήματα στους πελάτες. Ωστόσο, κάθε πάροχος έχει ελαφρώς διαφορετική δομή τιμο-

λόγησης, ενώ ορισμένοι πάροχοι πλατφόρμας χρεώνουν μια σταθερή χρέωση ανά μήνα.  
[i.6]

- **Ευκολία αδειοδότησης:** Οι πάροχοι PaaS διαχειρίζονται όλες τις άδειες για τα λειτουργικά συστήματα, τα εργαλεία ανάπτυξης και ό,τι άλλο περιλαμβάνεται στην πλατφόρμα τους.[i.6]

Μιλώντας για τα θετικά του μοντέλου δεν θα μπορούσαμε να μην αναφέρουμε και κάποια αρνητικά ενδεχόμενα της χρήσης του.

- **Κλείδωμα στον προμηθευτή:** Ενδέχεται να είναι δύσκολο να αλλάξετε πάροχο PaaS, καθώς η εφαρμογή κατασκευάζεται με τα εργαλεία του συγκεκριμένου προμηθευτή και ειδικά για την πλατφόρμα του. Κάθε προμηθευτής μπορεί να έχει διαφορετικές απαιτήσεις αρχιτεκτονικής. Διαφορετικοί προμηθευτές ενδέχεται να μην υποστηρίζουν τις ίδιες γλώσσες, βιβλιοθήκες, API, αρχιτεκτονική ή λειτουργικό σύστημα που χρησιμοποιούνται για την κατασκευή και εκτέλεση της εφαρμογής. Για να αλλάξουν προμηθευτή, οι προγραμματιστές μπορεί να χρειαστεί είτε να ανακατασκευάσουν είτε να τροποποιήσουν σε μεγάλο βαθμό την εφαρμογή τους.[i.6]
- **Εξάρτηση από τον προμηθευτή:** Η προσπάθεια και οι πόροι που απαιτούνται για την αλλαγή προμηθευτών PaaS μπορεί να κάνουν τις εταιρείες να εξαρτηθούν ακόμη περισσότερο από τον τρέχοντα προμηθευτή τους. Μια μικρή αλλαγή στις εσωτερικές διαδικασίες ή την υποδομή του προμηθευτή μπορεί να έχει τεράστιο αντίκτυπο στην απόδοση μιας εφαρμογής που έχει σχεδιαστεί για να λειτουργεί αποτελεσματικά στην παλιά διαμόρφωση. Επιπλέον, εάν ο πωλητής αλλάξει το μοντέλο τιμολόγησής του, η λειτουργία μιας εφαρμογής μπορεί ξαφνικά να γίνει πιο ακριβή.[i.6]
- **Προκλήσεις ασφάλειας και συμμόρφωσης:** Σε μια αρχιτεκτονική PaaS, ο εξωτερικός προμηθευτής αποθηκεύει τα περισσότερα ή όλα τα δεδομένα μιας εφαρμογής, μαζί με τη φιλοξενία του κώδικά της. Σε ορισμένες περιπτώσεις, ο πάροχος μπορεί στην πραγματικότητα να αποθηκεύει τις βάσεις δεδομένων μέσω ενός άλλου τρίτου παρόχου, ενός παρόχου IaaS. Αν και οι περισσότεροι προμηθευτές PaaS είναι μεγάλες εταιρείες με ισχυρή ασφάλεια, αυτό καθιστά δύσκολη την πλήρη αξιολόγηση και δοκιμή των μέτρων ασφαλείας που προστατεύουν την εφαρμογή και τα δεδομένα της. Επιπλέον, για τις εταιρείες που πρέπει να συμμορφώνονται με αυστηρούς κανονισμούς για την ασφάλεια των δεδο-



μένων, η επαλήθευση της συμμόρφωσης πρόσθετων εξωτερικών προμηθευτών θα προσθέσει περισσότερα εμπόδια στην έξοδο στην αγορά.[i.6]

Το PaaS σήμερα είναι συνήθως χτισμένο με περιέκτες. Οι περιέκτες ή αλλιώς containers θα είναι ένα πολύ βασικό κομμάτι της εργασίας που θα αναλυθεί στη συνέχεια, ενώ χρησιμοποιούνται εκτενώς και στην εφαρμογή που θα περιγραφεί στο τέλος. Σε επόμενο κεφάλαιο θα υπάρξει ανάλυση των περιεκτών, αλλά προς το παρόν για λόγους σαφήνειας πρέπει να αναφερθεί, πως οι περιέκτες εικονοποιούν ένα λειτουργικό σύστημα, δίνοντας την δυνατότητα στους προγραμματιστές να “πακετάρουν” μία εφαρμογή μόνο με τα πλήρως απαραίτητα του λειτουργικού συστήματος, που απαιτεί η εφαρμογή και να την εκτελέσουν σε οποιαδήποτε πλατφόρμα, δίχως αλλαγές ή ενδιάμεσο λογισμικό. Στα πλαίσια της εφαρμογής που υλοποιήθηκε θα περιγραφεί και θα γίνει χρήση του συστήματος Docker που δουλεύει με περιέκτες.

### 2.1.3 ΥΠΟΔΟΜΗ ΣΑΝ ΥΠΗΡΕΣΙΑ

Το μοντέλο **Υποδομής σαν Υπηρεσία (IaaS)** παρέχει κατ’ απαίτηση πρόσβαση σε υπολογιστικούς πόρους, όπως φυσικούς ή εικονικούς εξυπηρετητές, δικτύωση και αποθηκευτικό χώρο, ενώ η κοστολόγηση των παροχών αυτών γίνεται με βάση την χρήση που πραγματοποιείται. Κατ’ επέκταση, επιτρέπει στους χρήστες να κλιμακώσουν και να συρρικνώσουν τους υπολογιστικούς πόρους τους με βάση τις ανάγκες τους, αφαιρώντας έτσι την υποχρέωση του χρήστη να αποκτήσει εξ αρχής αχρειάστους υπολογιστικούς πόρους, που θα απαιτούσαν μεγάλη χρηματική επένδυση, και οι οποίοι θα ήταν αναγκαίοι μόνο σε ειδικές περιπτώσεις.[i.4] Όπως αναφέρει και η Microsoft στην επίσημη ιστοσελίδα του Azure, το IaaS είναι ένας από τους τέσσερις τύπους υπηρεσιών νέφους, μαζί με το λογισμικό ως υπηρεσία (SaaS), την πλατφόρμα ως υπηρεσία (PaaS) και το serverless. Το IaaS επιτρέπει να παρακάμψετε το κόστος και την πολυπλοκότητα της αγοράς και διαχείρισης φυσικών διακομιστών και υποδομών κέντρου δεδομένων. Κάθε πόρος προσφέρεται ως ξεχωριστό στοιχείο υπηρεσίας και πληρώνετε για έναν συγκεκριμένο πόρο μόνο για όσο διάστημα τον χρειάζεστε. Ένας πάροχος υπηρεσιών υπολογιστικού νέφους, όπως το Azure, διαχειρίζεται την υποδομή, ενώ εσείς αγοράζετε, εγκαθιστάτε, ρυθμίζετε και διαχειρίζετε το δικό σας λογισμικό -

συμπεριλαμβανομένων των λειτουργικών συστημάτων, του ενδιάμεσου λογισμικού και των εφαρμογών.[i.7]

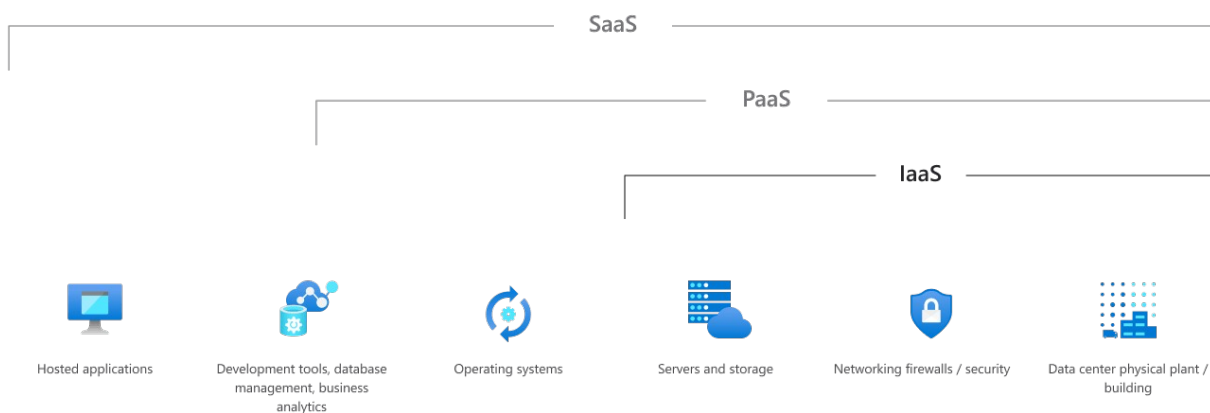
Σε αντίθεση με τα άλλα δύο μοντέλα που αναφέρθηκαν νωρίτερα, το IaaS προσφέρει το λιγότερο δυνατό επίπεδο ελέγχου στους υπολογιστικούς πόρους στο διαδίκτυο. Το IaaS αποτέλεσε το δημοφιλέστερο μοντέλο στις αρχές του 2010, αλλά ενώ παραμένει μία από τις πιο κοινές επιλογές, τα μοντέλα του Λογισμικού ως Υπηρεσία και της Πλατφόρμας ως Υπηρεσία έχουν δει πολύ μεγαλύτερο ρυθμό ανάπτυξης τα τελευταία χρόνια. [i.4]

Όπως φαίνεται και στην Εικόνα 1, κάθε μοντέλο υπηρεσιών διαφοροποιείται κυρίως στον τομέα των υποχρεώσεων που αναλαμβάνει ο χρήστης και ο πάροχος. Δεν υπάρχει απάντηση στο ερώτημα για το ποιο είναι το καλύτερο μοντέλο, εφόσον εξαρτάται από την σκοπιά του χρήστη. Αυτό που θα αναλυθεί σε επόμενο κεφάλαιο είναι κάποια αρνητικά αυτών των μοντέλων και πώς οι πάροχοι παίζουν τεράστιο ρόλο τελικά, στο κατά πόσο μία εταιρία επιλέγει να μεταβεί στο cloud και σε τι βαθμό.

### Cloud Computing Services: Who Manages What?



Εικόνα 1: Τι αναλαμβάνει ο πάροχος σε σχέση με κάθε μοντέλο.



Εικόνα 2: Απεικόνιση από την Microsoft όσον αφορά τις ευθύνες.

## 2.1.4 SERVERLESS

Όπως αναφέρει και η Amazon, ένας από τους μεγαλύτερους προμηθευτές cloud υπηρεσιών, το Serverless είναι ένας τρόπος για να περιγράψουμε τις υπηρεσίες, τις πρακτικές και τις στρατηγικές που σας επιτρέπουν να δημιουργείτε πιο ευέλικτες εφαρμογές, ώστε να μπορείτε να καινοτομείτε και να ανταποκρίνεστε στις αλλαγές ταχύτερα. Με το serverless computing, οι εργασίες διαχείρισης υποδομών, όπως η παροχή χωρητικότητας και η επιδιόρθωση, διεκπεραιώνονται από την AWS (ή από τον ανάλογο πάροχο), ώστε να μπορείτε να επικεντρωθείτε μόνο στη συγγραφή κώδικα που εξυπηρετεί τους πελάτες σας. Πιο συγκεκριμένα τώρα, οι υπηρεσίες χωρίς διακομιστή, όπως η AWS Lambda, διαθέτουν αυτόματη κλιμάκωση, ενσωματωμένη υψηλή διαθεσιμότητα και μοντέλο χρέωσης pay-for-value. Το Lambda είναι μια υπηρεσία υπολογισμού με βάση τα συμβάντα (event-based) που σας επιτρέπει να εκτελείτε κώδικα σε απόκριση σε συμβάντα (events) από περισσότερες από 200 εγγενώς ενσωματωμένες πηγές AWS και SaaS - και όλα αυτά χωρίς να διαχειρίζεστε διακομιστές. [i.8]

Κάποια πλεονεκτήματά αναφορικά με το serverless μοντέλο είναι:

- **Γρηγορότερη μετάβαση από την ιδέα στην αγορά:** Εξαλείφοντας τα λειτουργικά γενικά έξοδα, οι ομάδες σας μπορούν να κυκλοφορήσουν άμεσα, να πάρουν ανατροφοδότηση και να επαναλάβουν για να φτάσουν ταχύτερα στην αγορά. [i.8]

- **Προσαρμογή υπό κλίμακα:** Με τεχνολογίες που κλιμακώνονται αυτόματα από το μηδέν έως την αιχμή των απαιτήσεων, μπορείτε να προσαρμόζεστε στις ανάγκες των πελατών πιο γρήγορα από ποτέ. [i.8]
- **Μειώστε το κόστος σας:** Με ένα μοντέλο χρέωσης pay-for-value, δεν πληρώνετε ποτέ για υπερπρομήθεια και η χρήση των πόρων σας βελτιστοποιείται για λογαριασμό σας. [i.8]
- **Φτιάξτε καλύτερες εφαρμογές, πιο εύκολα:** Οι εφαρμογές χωρίς διακομιστή διαθέτουν ενσωματωμένες δυνατότητες ενσωμάτωσης υπηρεσιών, ώστε να μπορείτε να επικεντρωθείτε στη δημιουργία της εφαρμογής σας αντί να τη διαμορφώνετε. [i.8]

Όπως και σε κάθε μοντέλου που αναφέραμε νωρίτερα υπάρχουν και κάποια αρνητικά τα οποία πρέπει να ληφθούν υπόψιν:

- **Η δοκιμή (testing) και η αποσφαλμάτωση (debugging) γίνονται πιο δύσκολες:** είναι δύσκολο να αναπαραχθεί το περιβάλλον χωρίς διακομιστή, ώστε να διαπιστωθεί πώς θα αποδώσει ο κώδικας όταν αναπτυχθεί. Το debugging είναι πιο περίπλοκο επειδή οι προγραμματιστές δεν έχουν ορατότητα στις διαδικασίες του backend και επειδή η εφαρμογή διασπάται σε ξεχωριστές, μικρότερες λειτουργίες. [i.9]
- **Ο υπολογισμός χωρίς διακομιστή εισάγει νέες ανησυχίες σχετικά με την ασφάλεια:** Αυτό μπορεί να αποτελέσει πρόβλημα ιδιαίτερα για εφαρμογές που χειρίζονται προσωπικά ή ευαίσθητα δεδομένα. Επειδή στις εταιρείες δεν ανατίθενται οι δικοί τους διακριτοί φυσικοί διακομιστές, οι πάροχοι serverless συχνά εκτελούν κώδικα από πολλούς πελάτες τους σε έναν μόνο διακομιστή ανά πάσα στιγμή. Αυτό το ζήτημα της κοινής χρήσης μηχανημάτων με άλλα μέρη είναι γνωστό ως "multitenancy" - σκεφτείτε ότι πολλές εταιρείες προσπαθούν να μισθώσουν και να εργαστούν ταυτόχρονα σε ένα μόνο γραφείο. Το multitenancy μπορεί να επηρεάσει την απόδοση των εφαρμογών και, εάν οι διακομιστές πολλαπλής μίσθωσης δεν έχουν ρυθμιστεί σωστά, μπορεί να οδηγήσει σε έκθεση δεδομένων. Το multitenancy έχει ελάχιστες έως μηδαμινές επιπτώσεις για δίκτυα που λειτουργούν σωστά με sandbox και διαθέτουν αρκετά ισχυρή υποδομή. [i.9]
- **Οι αρχιτεκτονικές χωρίς διακομιστή δεν είναι κατασκευασμένες για μακροχρόνιες διεργασίες:** Αυτό περιορίζει τα είδη των εφαρμογών που μπορούν να τρέξουν οικονομικά αποδοτικά σε μια αρχιτεκτονική χωρίς διακομιστή. Επειδή οι πάροχοι serverless χρε-

ώνουν για τον χρόνο εκτέλεσης του κώδικα, ενδέχεται να κοστίζει περισσότερο η εκτέλεση μιας εφαρμογής με μακροχρόνιες διεργασίες σε μια υποδομή serverless σε σύγκριση με μια παραδοσιακή υποδομή. [i.9]

- **Μπορεί να επηρεαστεί η απόδοση:** Ο κώδικας χωρίς διακομιστή μπορεί να χρειαστεί να "εκκινήσει" όταν χρησιμοποιείται. Αυτός ο χρόνος εκκίνησης μπορεί να υποβαθμίσει την απόδοση. Ωστόσο, εάν ένα κομμάτι κώδικα χρησιμοποιείται τακτικά, ο πάροχος serverless θα το διατηρεί έτοιμο για ενεργοποίηση - ένα αίτημα για αυτόν τον έτοιμο κώδικα ονομάζεται " warm start ". Ένα αίτημα για κώδικα που δεν έχει χρησιμοποιηθεί για κάποιο χρονικό διάστημα ονομάζεται " cold start " (ψυχρή εκκίνηση). [i.9]
- **Ο εγκλωβισμός στον προμηθευτή αποτελεί κίνδυνο:** Επιτρέποντας σε έναν προμηθευτή να παρέχει όλες τις υπηρεσίες backend για μια εφαρμογή, αυξάνεται αναπόφευκτα η εξάρτηση από αυτόν τον προμηθευτή. Η δημιουργία μιας αρχιτεκτονικής χωρίς διακομιστή με έναν προμηθευτή μπορεί να καταστήσει δύσκολη την αλλαγή προμηθευτή εάν είναι απαραίτητο, ειδικά από τη στιγμή που κάθε προμηθευτής προσφέρει ελαφρώς διαφορετικά χαρακτηριστικά και ροές εργασίας. [i.9]

## 2.2 ΥΒΡΙΔΙΚΟ ΝΕΦΟΣ ΚΑΙ ΠΟΛΥΝΕΦΟΣ

Το **υβριδικό νέφος** (Hybrid Cloud) συνδέει τις ιδιωτικές υπηρεσίες (private cloud) μίας εταιρίας με τις διαδικτυακές παροχές ενός δημόσιου cloud σε μία ενιαία ελαστική υποδομή για την εκτέλεση των εταιρικών εφαρμογών. Αυτή η εναλλακτική προσέγγιση προσφέρει την ελευθερία σε έναν οργανισμό να εναλλάσσει μεταξύ των δύο διαφορετικών cloud αναλόγως των περιστάσεων. Σύμφωνα με την IBM έχει αποδειχθεί πως οι τεχνικοί και επιχειρησιακοί στόχοι επιτυγχάνονται πολύ αποτελεσματικότερα και φθηνότερα με ένα υβριδικό νέφος, σε σχέση με την επιλογή ενός private cloud ή ενός public cloud αποκλειστικά.[i.3]

Το **πολυνέφος** (Multicloud), που ουσιαστικά είναι πολλαπλά υπολογιστικά νέφη, πηγαίνει την συνδυαστική φιλοσοφία των clouds ένα βήμα παρακάτω. Σε μία τέτοια περίπτωση μπορεί να υπάρξει μία μίξη IaaS, PaaS, ή SaaS. Με ένα πολυνέφος, ο χρήστης μπορεί να αποφασίσει για

κάθε υπολογιστικό φόρτο ξεχωριστά, ποιο cloud είναι καταλληλότερο βασισμένος στις μοναδικές απαιτήσεις του συστήματός του. [i.3]

## 2.3 ΑΛΛΕΣ ΚΟΙΝΕΣ ΧΡΗΣΕΙΣ ΤΟΥ ΥΠΟΛΟΓΙΣΤΙΚΟΥ ΝΕΦΟΥΣ

Εκτός των παραπάνω εφαρμογών και μοντέλων που αναφέρθηκαν υπάρχουν και άλλες υπηρεσίες που προσφέρονται, οι οποίες είναι άξιες αναφοράς, αφού αποτελούν δυναμικά εργαλεία για πολλές επιχειρήσεις. Αναφορικά κάποιες περιπτώσεις είναι οι εξής:

- **Testing και ανάπτυξη:** Από τα ισχυρότερα σενάρια για τα οποία μπορεί κάποιος να ακολουθήσει το μονοπάτι του cloud, αφού περιορίζει σε μεγάλο βαθμό το κόστος του χρήστη, αλλά και τον χρόνο που θα απαιτούσε η εγκατάσταση και η προετοιμασία του κατάλληλου περιβάλλοντος τοπικά σε μία εταιρία για τον συγκεκριμένο σκοπό. [i.3]
- **Big data analytics:** Πρόκειται για τεράστιο πλεονέκτημα του cloud computing, αφού μπορεί αποτελεσματικά και γρήγορα να εξάγει συμπεράσματα από τεράστιο όγκο δομημένων και μη δομημένων δεδομένων. Αυτή η τεχνική χρησιμοποιείται εκτενώς για να εντοπιστούν μοτίβα αγοροπωλησιών στους καταναλωτές του διαδικτύου, ώστε να υπάρξει προσαρμοστικό και στοχευμένο μάρκετινγκ σε συγκεκριμένες μερίδες του πληθυσμού. [i.3]
- **Cloud Storage:** Από τις πιο κοινές υπηρεσίες, ακόμα και για έναν μέσο χρήστη είναι η δυνατότητα να αποθηκεύει στο διαδίκτυο τα δεδομένα του. Για μία εταιρία αυτό αποτελεί μεγάλο ελαφρυντικό στο φόρτο εργασίας και υποχρεώσεων, αφού γλυτώνουν από την υποχρέωση της συντήρησης του υλικού ενώ η χρέωση είναι ανάλογη του χώρου που καταναλώνεται.[i.3] Βέβαια, αυτή η υπηρεσία δεν αποτελεί μονόδρομο για τους οργανισμούς, αφού υπάρχουν περιπτώσεις που η αποθήκευση κάποιων δεδομένων πρέπει να γίνεται σε ένα ιδιωτικό cloud.
- **Ανάκτηση δεδομένων**
- **Data backup**

## 2.4 ΕΙΚΟΝΟΠΟΙΗΣΗ ΚΑΙ ΠΕΡΙΕΚΤΕΣ

Όπως προαναφέρθηκε η **εικονοποίηση** (virtualization) και οι **περιέκτες** (containers) είναι δύο τεχνικές που χρησιμοποιούνται σε πληθώρα υπηρεσιών στο cloud. Οι δύο αυτές τεχνολογίες, αν και μοιάζουν στην φιλοσοφία τους, προσφέρουν διευκολύνσεις που δεν είναι σε όλες τις περιπτώσεις ίδιες. Η κάθε μία τεχνολογία εκ των δύο έχει τα θετικά και τα αρνητικά της, τα οποία ορίζουν κατ' επέκταση και την τελική επιλογή του χρήστη για το ποια από τις δύο είναι καταλληλότερη για την περίπτωση του. Σε αυτό το κεφάλαιο θα αναλυθεί η κάθε τεχνολογία ξεχωριστά, ώστε να γίνουν κατανοητές οι διαφορές τους.

### 2.4.1 ΘΕΤΙΚΑ ΚΑΙ ΑΡΝΗΤΙΚΑ ΕΙΚΟΝΟΠΟΙΗΣΗΣ

Αρχικά, θα πρέπει να αναφερθεί, πως και οι δύο τεχνολογίες δημιουργούν εικονικά λογισμικά πακέτα. Πέρα του γενικού σκοπού που ακούγεται ίδιος, η κάθε μία διαφέρει στον τρόπο που λειτουργεί, στα χαρακτηριστικά της και στις περιπτώσεις που χρησιμοποιείται.

Πριν την εμφάνιση των περιεκτών, προϋπήρχαν οι εικονικές μηχανές οι οποίες ήταν η επικρατέστερη τεχνική για για την βελτιστοποίηση της χωρητικότητας ενός διακομιστή. Προγραμματισμένες να εξομοιώνουν το υλικό ενός φυσικού υπολογιστή με ένα πλήρες λειτουργικό σύστημα, οι εικονικές μηχανές (και ο Hypervisor) έδιναν την δυνατότητα να “τρέχουν” πολλαπλοί υπολογιστές με πολλαπλά λειτουργικά συστήματα πάνω σε ένα υλικό ενός και μόνο φυσικού υπολογιστή. [i.13]

Ο Hypervisor που αναφέρθηκε πριν, ή ο “επιβλέπων” των εικονικών μηχανών, είναι ένα λογισμικό επίπεδο το οποίο επιτρέπει σε πολλαπλά λειτουργικά συστήματα να τρέχουν ταυτόχρονα πάνω στους ίδιους φυσικούς πόρους. Στην ουσία, αυτό που αναλαμβάνει ο Hypervisor είναι να μοιράσει κατάλληλα τους υπολογιστικούς πόρους, όπως υπολογιστική ισχύ, μνήμη και αποθηκευτικό χώρο, κατάλληλα σε κάθε εικονική μηχανή. [i.13]

Με μία πρώτη ματιά κάθε εικονική μηχανή μοιάζει απλά με ένα φάκελο δεδομένων. Αυτός ο φάκελος μπορεί να αντιγραφεί, να μετακινηθεί, ή και να διαγραφεί όπως κάθε άλλο αρχείο ενός υπολογιστική. Με αυτόν τον τρόπο ένας οργανισμός μπορεί να συγκεντρώσει συγκεκριμένες διεργασίες σε συγκεκριμένες εικονικές μηχανές, αλλά πάνω από όλα να το κάνει αυτό, δίχως να

είναι αναγκαία η αγορά νέου υλικού, που να σου δίνει την δυνατότητα να “τρέξεις” πολλά λειτουργικά συστήματα ταυτόχρονα. Με τις εικονικές μηχανές δίνεται και η δυνατότητα να αναβαθμίσεις το ίδιο το λειτουργικό σύστημα και τις εφαρμογές της κάθε εικονικής μηχανής δίχως αυτό να επηρεάζει τον τελικό χρήστη.[i.13]

Αναφορικά λοιπόν κάποια από τα θετικά της χρήσης εικονικών μηχανών είναι:

1. **Λιγότερο φυσικό υλικό.** Σε ένα τυπικό καταναμεμημένο σύστημα ελέγχου (DCS), μπορεί να έχετε δύο διακομιστές Tag/OS, δύο διακομιστές δέσμης, έναν ή δύο ιστορικούς και έναν ή δύο σταθμούς μηχανικών. Εύκολα, έχετε μπροστά σας έξι διακομιστές που θα πρέπει να συντηρούνται φυσικά. Με την χρήση εικονικών μηχανών θα διαπιστώσετε τεράστια εξοικονόμηση χρόνου και συνολικού κόστους από την αντικατάσταση του υλικού και τη συντήρηση.[i.11]
2. **Κεντρική τοποθεσία για τη διαχείριση όλων των μέσων.** Όλες οι εικονικές μηχανές σας μπορούν να διαχειρίζονται από μία τοποθεσία.[i.11]
3. **Πιο φιλικό προς το περιβάλλον.** Αν κοιτάξετε την τρέχουσα διαμόρφωσή ενός τυπικού πληροφοριακού συστήματος, τα περισσότερα μηχανήματά βρίσκονται σε αδράνεια. Όμως, με την εικονικοποίησή τους και τη λειτουργία τους σε ένα cluster, μεγιστοποιούνται οι δυνατότητες των μηχανημάτων, ενώ παράλληλα εξοικονομούνται χρήματα από το ενεργειακό κόστος.[i.11]
4. **Η ανάκαμψη από καταστροφές είναι γρήγορη.** Αναπτύξτε εκ νέου εικονικές μηχανές στο σύστημά σας (μόλις επαναφέρετε σε λειτουργία τη μηχανή host) και μπορείτε να έχετε το σύστημά σας ξανά σε λειτουργία σε χρόνο μηδέν.[i.11]
5. **Δυνατότητες επέκτασης.** Με την υποδομή στη θέση της, είναι απλά θέμα ανάπτυξης ενός νέου μηχανήματος και παραμετροποίησης. Δεν χρειάζεται να αγοράσετε νέους διακομιστές.[i.11]
6. **Αναβαθμίσεις συστήματος.** Ο χρόνος και η ταλαιπωρία της δημιουργίας εικόνων συστήματος πριν από την εφαρμογή μιας ενημερωμένης έκδοσης και η αποτυχία της επαναφοράς του συστήματος αποτελούν πραγματικότητα. Με το εικονικό περιβάλλον, αν κάτι πάει στραβά κατά την εφαρμογή μιας επιδιόρθωσης ή ενημέρωσης, μπορείτε απλώς να επαναφέρετε την εικονική μηχανή εκεί που ήταν πριν εφαρμόσετε την επιδιόρθωση χρησιμοποιώντας ένα snapshot.[i.11]



7. **Αδειοδότηση λογισμικού.** Πολλά πακέτα λογισμικού (όπως τα προϊόντα της Rockwell) συνδέουν ένα κλειδί άδειας χρήσης με ένα αναγνωριστικό σκληρού δίσκου. Σε ένα εικονικό περιβάλλον, το αναγνωριστικό σκληρού δίσκου παραμένει το ίδιο, ανεξάρτητα από το σε ποιο κομμάτι υλικού εκτελείται.[i.11]
8. **Υποστηρίζει παλαιά λειτουργικά συστήματα.** Καθώς το υλικό εξελίσσεται και τα λειτουργικά συστήματα παλιώνουν, είναι πιο δύσκολο να βρείτε υλικό και λογισμικό που να είναι συμβατά. Η εικονικοποίηση αυτών των μηχανών εξαλείφει τα προβλήματα συμβατότητας των λειτουργικών συστημάτων. Αυτό δεν επιλύει το πρόβλημα των παρωχημένων λειτουργικών συστημάτων που δεν υποστηρίζονται πλέον - το οποίο αποτελεί κίνδυνο για την ασφάλεια.[i.11]
9. **Συμβατότητα προς τα εμπρός.** Καθώς γίνεται διαθέσιμο νέο υλικό, οι εικονικές μηχανές σας μπορούν να εξακολουθούν να λειτουργούν σε αυτό το νέο υλικό (εφόσον υποστηρίζεται από το λογισμικό του εικονικού κεντρικού υπολογιστή).[i.11]

Παρόλα αυτά, όπως κάθε τεχνολογία, έτσι και οι εικονικές μηχανές έχουν κάποια αρνητικά.

1. Από τη στιγμή που κάθε εικονική μηχανή εμπεριέχει ένα λειτουργικό σύστημα και ένα εικονικό αντίγραφο του υλικού που χρειάζεται το λειτουργικό σύστημα, αυτό σημαίνει πως μία εικονική μηχανή απαιτεί σημαντικό ποσοστό μνήμης (RAM) αλλά και υπολογιστικούς πόρους (CPU). [i.13]
2. Κόστος. Το αρχικό κόστος μπορεί να είναι πολύ υψηλότερο και, ανάλογα με το πόσο υψηλή διαθεσιμότητα θέλετε, θα πρέπει να είστε πρόθυμοι να σχεδιάσετε το σύστημα για τις ανάγκες σας τώρα και στο μέλλον.[i.11]
3. Λόγο της αυξανόμενης ανάγκης μίας εικονικής μηχανής για υπολογιστικούς πόρους, καθιστά τον κύκλο ζωής της ανάπτυξης του λογισμικού αρκετά πολύπλοκο. [i.13] Εάν δεν είστε εξοικειωμένοι με τις πτυχές του υλικού και του δικτύου της όλης εγκατάστασης, μπορεί να είναι αποθαρρυντικό να το καταλάβετε. Οι κανόνες δρομολόγησης και τα εικονικά τοπικά δίκτυα (VLAN) συνεχίζουν να προσθέτουν πολυπλοκότητα, ειδικά αν η ασφάλεια αποτελεί ανησυχία.[i.11]
4. Η μεταφορά των εικονικών μηχανών μεταξύ ιδιωτικών clouds, δημόσιων clouds, και παραδοσιακών κέντρων δεδομένων αποτελεί μία δύσκολη διαδικασία.[i.13] Επιπλέον, συ-

χνά το υλικό είναι συγκεντρωμένο σε μια τοποθεσία, καθιστώντας μια μεμονωμένη καταστροφή πιο πιθανό να προκαλέσει σημαντική διακοπή λειτουργίας. [i.11]

5. Κλειδιά υλικού. Ναι, μπορείτε να χρησιμοποιήσετε κλειδιά υλικού. Μπορείτε να δεσμεύσετε μια θύρα USB σε μια συγκεκριμένη εικονική μηχανή. Ωστόσο, δεν μπορείτε να μετακινήσετε την εικονική μηχανή χωρίς να μετακινήσετε φυσικά και το κλειδί.[i.11]

## 2.4.2 ΘΕΤΙΚΑ ΚΑΙ ΑΡΝΗΤΙΚΑ ΠΕΡΙΕΚΤΩΝ

Οι περιέκτες (containers) για να γίνουν κατανοητοί θα πρέπει να ρίξουμε μια ματιά στο Docker. Το Docker είναι ένα σύστημα που πραγματοποιεί μία “ενορχήστρωση” και μία διαχείριση εφαρμογών σε περιέκτες. Ένας περιέκτης εφαρμογής (application container) εικονοποιεί μία εφαρμογή μαζί με τις βιβλιοθήκες λογισμικού, τις υπηρεσίες, και το μέρη του λειτουργικού συστήματος που απαιτούνται για την εκτέλεση της εφαρμογής. Όλοι οι περιέκτες κατά την ανάπτυξη τους τρέχουν πάνω σε ένα μοναδικό λειτουργικό σύστημα, διότι μοιράζονται κοινούς χρησιμοποιούμενους πόρους αυτού του λειτουργικού συστήματος. Αυτό συνεπάγεται με το γεγονός ότι ένας περιέκτης απαιτεί πολύ λιγότερο χώρο και πόρους από μία εικονική μηχανή. Τέλος, αυτή η λογισμική “εικόνα” εντός ενός περιέκτη μπορεί να δημιουργηθεί πολύ πιο γρήγορα σε σχέση με την “εικόνα” μίας εικονικής μηχανής – Σε μία κλίμακα δευτερολέπτων, έναντι λεπτών στην άλλη περίπτωση. [i.14]

Αναφορικά κάποια θετικά των containers είναι:

1. Τα containers παρέχουν μια ελαφριά, γρήγορη και απομονωμένη υποδομή για την εκτέλεση των εφαρμογών σας. Δεδομένου ότι είναι πιο ελαφριά, πιο ευέλικτα και μπορούν να δημιουργηθούν αντίγραφα ασφαλείας και να αποκατασταθούν ταχύτερα.[i.10]
2. Η εφαρμογή, οι εξαρτήσεις, οι βιβλιοθήκες, τα δυαδικά αρχεία και τα αρχεία ρυθμίσεων είναι συνήθως συγκεντρωμένα στον περιέκτη, παρέχοντας μια εύκολη λύση για τη μεταφορά της εφαρμογής σας οπουδήποτε.[i.10]

3. Το μέσο μέγεθος του κοντέινερ είναι συνήθως μικρότερο από 100MB, σε αντίθεση με μερικά gigabytes που χρησιμοποιούνται από μια εικονική μηχανή.[i.10]
4. Τα containers μπορούν να σας βοηθήσουν να μειώσετε το κόστος λειτουργίας και ανάπτυξης.[i.10]

Αντίστοιχα κάποια από τα αρνητικά των containers είναι:

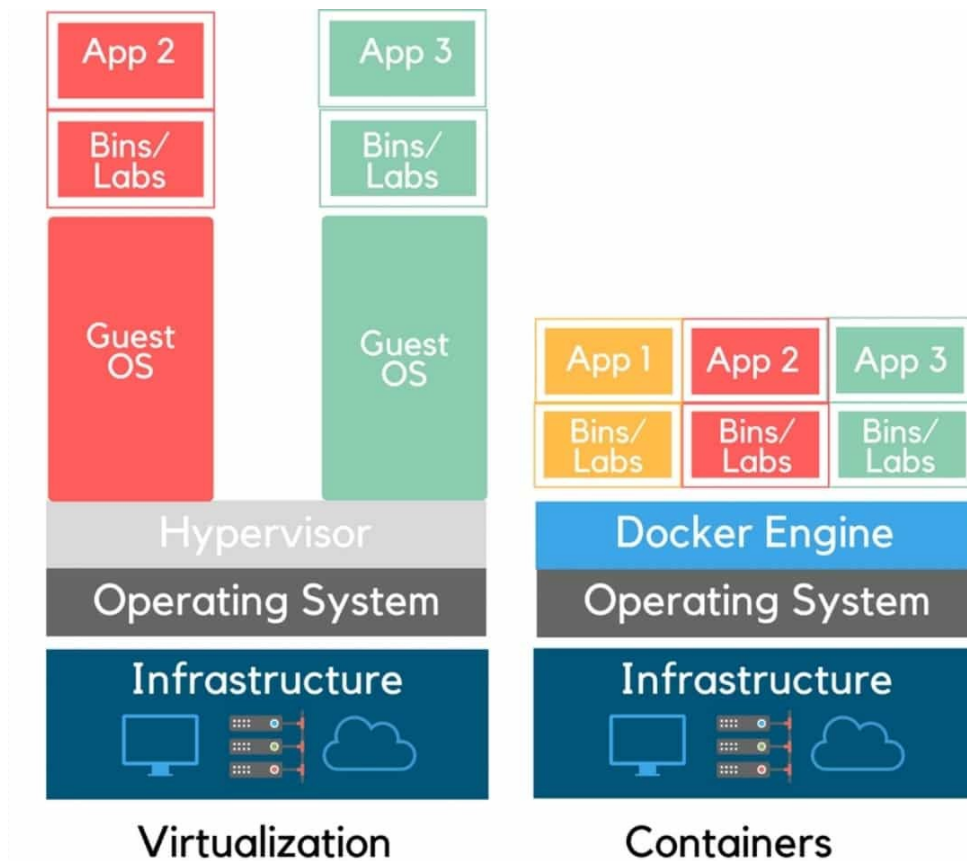
1. Πρώτον, όλοι οι περιέκτες που φιλοξενούνται σε ένα λειτουργικό σύστημα πρέπει να σχεδιαστούν να “τρέχουν” στο συγκεκριμένο είδος λειτουργικού συστήματος. Οι περιέκτες που είναι βασισμένοι σε άλλο λειτουργικό σύστημα, θα χρειαστούν και διαφορετικό host. Βέβαια, εδώ θα πρέπει να αναφερθεί, πως πλέον τα windows παρέχουν λύσεις πάνω σε αυτήν την ιδιαιτερότητα των container, όπως το WSL που εικονοποιεί το linux πάνω από τα windows.[i.13]
2. Η δικτύωση μπορεί να είναι δύσκολη όταν εργάζεστε με containers. Πρέπει να διατηρείτε μια καλή σύνδεση δικτύου, ενώ ταυτόχρονα να επιχειρείτε ενεργά να κρατούνται τα κοντέινερ απομονωμένα.[i.10] Ένα δίκτυο με κοντέινερ απαιτεί, εκτός από ειδικό λογισμικό, όπως το docker, για να δημιουργηθεί το εικονικό δίκτυο που θα τα φιλοξενεί, αλλά και επιπλέον παραμετροποίηση για τα ports του κάθε κοντέινερ στο δίκτυό του. Αυτό προσθέτει μία επιπλέον δυσκολία στο χειρισμό και την παραμετροποίηση ενός τέτοιου περιβάλλοντος.
3. Το containerization χρησιμοποιείται συνήθως για τη δημιουργία πολυεπίπεδων υποδομών. Η σταδιακή αύξηση των container σε ένα δίκτυο φέρνει και την ανάγκη μίας ενοποιημένης διεπαφής παρακολούθησης του συνόλου των container, λόγω της πρακτικής δυσκολίας να παρακολουθείς το καθένα ξεχωριστά.
4. Η παρακολούθηση των containers είναι πιο δύσκολη από ό,τι σε μια εικονική μηχανή.[i.10]
5. Η τεχνολογία των περιεκτών είναι μία γενικά νέα λύση στον τομέα της πληροφορικής, με μία μεγάλη γκάμα τεχνικών εφαρμογής και ανάπτυξης, καθιστώντας την μετάβαση προς την τεχνολογία μία δύσκολη διαδικασία για κάποιους.[i.13]

Στην Εικόνα 3 γίνεται πιο ξεκάθαρη και η διαφορά στην αρχιτεκτονική της κάθε τεχνολογίας.

Η σκέψη να χρησιμοποιηθούν κοντέινερ για κάποιο έργο δεν είναι μονόδρομος. Πριν μεταβεί κάποιος προς αυτή την κατεύθυνση θα πρέπει να ληφθεί υπόψιν πως:

- Δεν είναι απαραίτητο κάθε εφαρμογή να είναι σε κοντέινερ, και σε ορισμένες περιπτώσεις, μπορεί να μην παρέχει κανένα όφελος.[i.10]
- Υπάρχουν πολλά εργαλεία που μπορείτε να χρησιμοποιήσετε για τη δημιουργία κοντέινερ. Αυτή τη στιγμή, το **Docker** είναι μια δημοφιλής επιλογή, η οποία χρησιμοποιήθηκε και στην εφαρμογή της εργασίας.[i.10]

Οι προγραμματιστές με εμπειρία στο containerization θα είναι πιο πολύτιμοι στο μέλλον, καθώς πολλές εταιρείες εφαρμόζουν μια νοοτροπία "DevOps" στη ροή εργασίας τους.[i.10]



Εικόνα 3: Απεικόνιση των επιπέδων που απαρτίζουν κάθε τεχνολογία ξεχωριστά.

## 2.5 ΠΑΡΟΧΟΙ ΤΟΥ ΥΠΟΛΟΓΙΣΤΙΚΟΥ ΝΕΦΟΥΣ

Σε αυτό το σημείο, αφού μιλήσαμε γενικά για το υπολογιστικό νέφος, τη λογική του και τα πλεονεκτήματά του, τα μοντέλα των υπηρεσιών του, την εικονοποίηση και τους περιέκτες, ήρθε η στιγμή να σταθούμε στους πάροχους και τις εταιρίες, που αυτή τη στιγμή διαπρέπουν στο χώρο του Υπολογιστικού Νέφους και προσφέρουν τις υπηρεσίες τους σε διεθνή κλίμακα.

Στα πρώτα χρόνια της εξέλιξης του υπολογιστικού νέφους, οι πάροχοι αύξαναν με ταχύτατους ρυθμούς το περιεχόμενο των πακέτων τους. Με την πάροδο των χρόνων, και όσο οι μεγάλες εταιρίες άρχισαν να υιοθετούν περισσότερο στρατηγικές πολυνέφους (multicloud), φάνηκε πως ακόμα και οι πάροχοι που είχαν πιο περιορισμένα πακέτα υπηρεσιών, όχι μόνο μπορούσαν να επιβιώσουν σε ένα τέτοιο περιβάλλον, αλλά και να διαπρέψουν. Χαρακτηριστικά παραδείγματα τέτοιων περιπτώσεων είναι το Datalog για παρατηρησιμότητα, το Twilio για επικοινωνία, το MongoDB για βάσεις δεδομένων, κλπ. [i.17]

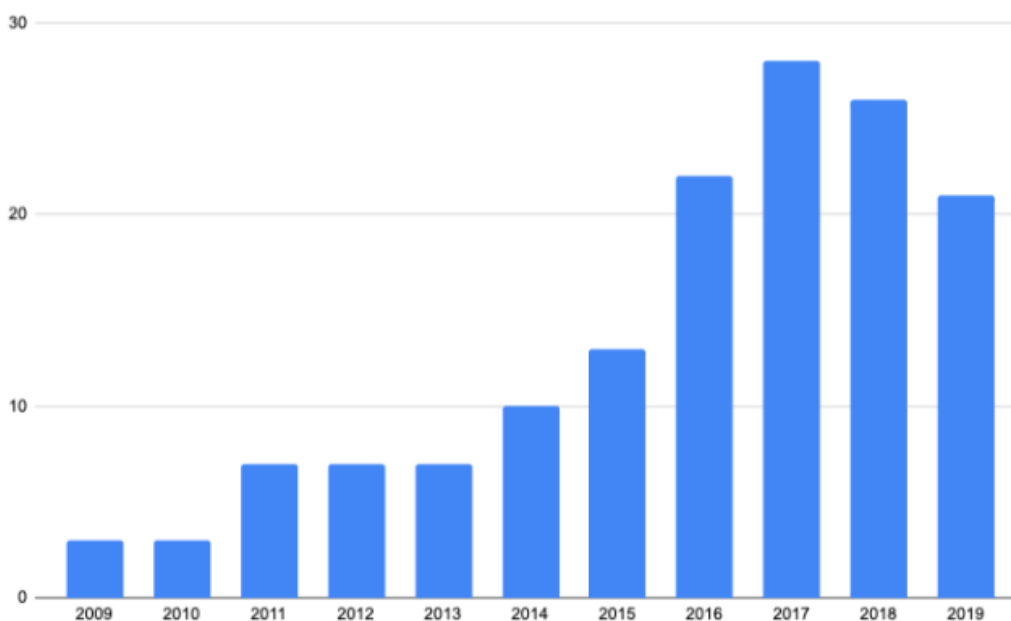
Παρακάτω, θα γίνει περιγραφή της εξέλιξης των προσφορών των προμηθευτών. Όσο αυτή η εξέλιξη οδηγούσε στην ομαδοποίηση πολλών υπηρεσιών σε ενιαία πακέτα προσφορών, δημιουργήθηκαν οι ευκαιρίες για μικρότερους προμηθευτές να επικεντρωθούν σε συγκεκριμένες υπηρεσίες, και να προσφέρουν ποιοτικότερες παροχές σε κάποιους τομείς.[i.17]

### 2.5.1 AMAZON WEB SERVICES

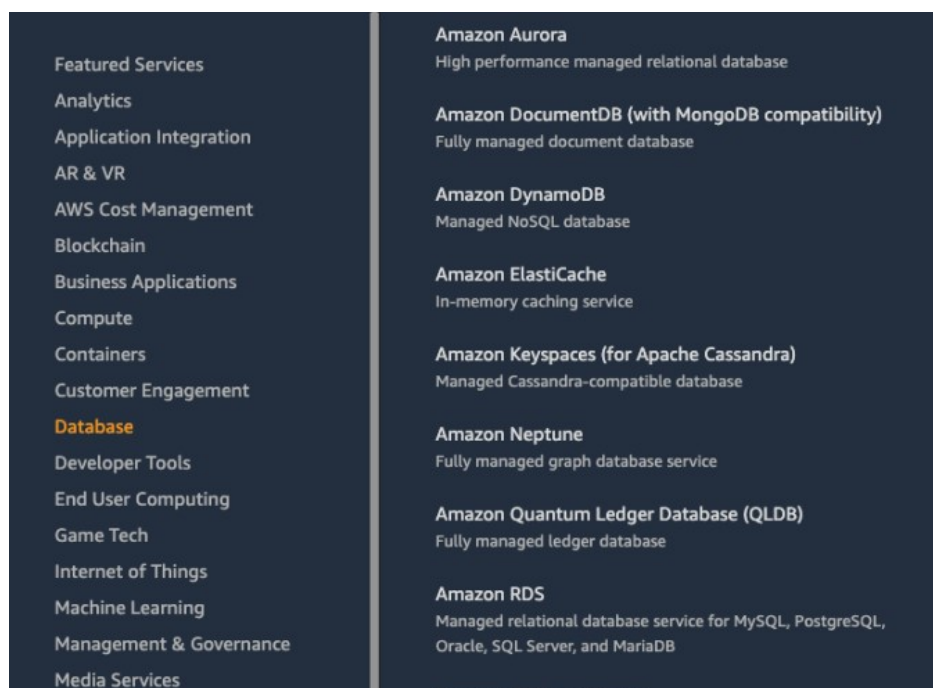
Η πλατφόρμα Amazon Web Services (AWS) πρωτοεμφανίστηκε το 2002. Τον Νοέμβριο του 2004, παρουσιάστηκε η πρώτη δημόσια διαδικτυακή υπηρεσία του AWS. Αυτή η υπηρεσία ήταν η Simple Queue Service (SQS), η οποία παραμένει διαθέσιμη ως και σήμερα. Το SQS προσφέρει τη δυνατότητα σε προγραμματισμένη αποστολή μηνυμάτων μέσω του διαδικτύου. Αυτό κατ' επέκταση δίνει την δυνατότητα στην επιτυχή ασύγχρονη επικοινωνία μεταξύ συστημάτων. Για παράδειγμα, μία e-commerce ιστοσελίδα θα μπορούσε να στέλνει μέσω αυτής της υπηρεσίας τα δεδομένα μίας αγοράς ενός χρήστη σε ένα εξωτερικό κέντρο διαχείρισης αγορών, άσχετα αν αυτό είναι ενεργό την παρούσα στιγμή.[i.17]

Το Μάρτιο του 2006 το AWS κυκλοφόρησε επισήμως ως μία σουίτα από βασικές υπηρεσίες που απάρτιζαν ένα ολοκληρωμένο περιβάλλον, το οποίο μπορούσαν άλλοι προγραμματιστές να αναπτύξουν και να φιλοξενήσουν τις διαδικτυακές εφαρμογές τους. Η αρχική έκδοση του AWS αποτελούνταν από το S3(Αποθηκευτικός χώρος), Virtualized Compute (Εικονοποιημένος Υπολογισμός), και το SQS που αναφέρθηκε παραπάνω. Σύντομα, η Amazon ξεκίνησε να διευρύνει την γκάμα υπηρεσιών που πρόσφερε στην προσπάθειά της να συμβαδίσει με τις απαιτήσεις των προγραμματιστών που χρησιμοποιούσαν το AWS. Επίσης ως ένα επιπλέον κίνητρο, μέχρι το 2010 όλες οι ιστοσελίδες λιανικής πώλησης της Amazon είχαν μεταφερθεί στο AWS. Χαρακτηριστικό παράδειγμα χρήσης του AWS, αποτελεί το Netflix το οποίο ξεκίνησε να χρησιμοποιεί την υπηρεσία της Amazon το 2008, ενώ εξέφρασε την επιθυμία της να μεταναστεύσει ολοκληρωτικά στο νέφος το 2010, κάτι που ολοκληρώθηκε με επιτυχία το 2016.

Όπως θα περίμενε ο καθένας, όσο περισσότερο αυξανόταν η χρήση του AWS, τόσο μεγαλύτερη απαίτηση παρουσιαζόταν στην ανάπτυξη περισσότερων υποδομών για την κάλυψη όλων των κοινών αναγκών σχετιζόμενων με την υλοποίηση σύγχρονων διαδικτυακών εφαρμογών. Η πορεία εξέλιξης του πλήθους των υπηρεσιών υπήρξε ταχύτατη όπως φαίνεται και στην Εικόνα 4, που πάρθηκε από blog post του Jerry Hargroove, [i.17][i.18]



Εικόνα 4: Αριθμός υπηρεσιών στο AWS ανά τα έτη.[i.18]



Εικόνα 5: Κατηγορίες στο AWS

## 2.5.2 ΕΜΦΑΝΙΣΗ ΑΛΛΩΝ ΠΑΡΟΧΩΝ

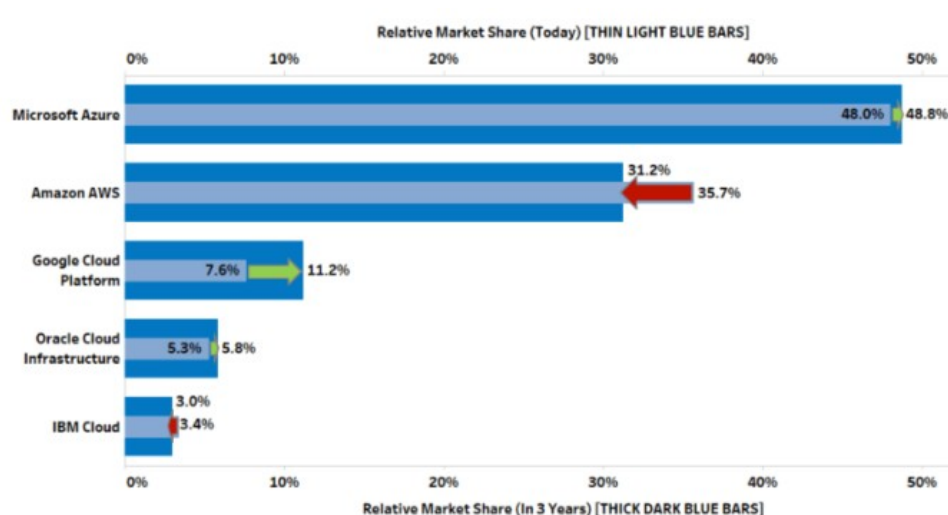
Με την εμφάνιση του AWS, άλλες μεγάλες εταιρίες τεχνολογίας είδαν τις δυνατότητες του cloud hosting και θέλησαν να βεβαιώσουν πως δεν θα μείνουν εκτός παιχνιδιού. Το 2008 η Microsoft ανακοίνωσε το Azure και κυκλοφόρησε επίσημα το 2010. Σήμερα, το Azure διαθέτει 600 ανεξάρτητες υπηρεσίες. Φυσικά η Google θα ήταν ο επόμενος “ύποπτος”, που το 2008 ανακοίνωσε το GCP (Google Cloud Platform), το οποίο αρχικά προοριζόταν ως μία πλατφόρμα ως υπηρεσία (PaaS), για την εκτέλεση κώδικα. Όπως και οι υπόλοιπες εταιρίες, έχουν διευρύνει τις υπηρεσίες τις αλλά με ένα πιο μεθοδικό τρόπο. [i.17]

Οι τρεις αυτοί κολοσσοί καλύπτουν το 58% του cloud infrastructure market με τις τελευταίες εκτιμήσεις το 2020. Υπάρχει φυσικά και ένα υπολειπόμενο 42% το οποίο αποτελείται από τις εταιρίες Alibaba, IBM, Oracle, Rackspace, κλπ. Επίσης η κατανομή του ποσοστού ανάμεσα στους τρεις κολοσσούς έχει αλλάξει τα τελευταία τρία χρόνια. Παρακάτω βλέπουμε αυτήν την αλλαγή:

- AWS: Μειώθηκε από 34% στο 32%
- Azure: Αυξήθηκε από 11% στο 18%
- Google: Αυξήθηκε από 6% στο 8%

Το μεγαλύτερο μέρος της ανάπτυξης για το Azure και το GCP προήλθε εις βάρος των μικρότερων παρόχων, αλλά το AWS είδε και αυτό μείωση στο μερίδιο του τα τελευταία χρόνια. Με μεγάλη διαφορά τη μεγαλύτερη αύξηση έχει δει το Azure. Αυτό μπορεί να αποδοθεί σε μερικούς παράγοντες - βελτιώσεις στην πλατφόρμα τους, αξιοποίηση της υφιστάμενης διείσδυσής τους μεταξύ των Global 2000 και ομαδοποίηση των προϊόντων πληροφορικής. Επιπλέον, οι φόβοι του ανταγωνισμού μεταξύ των επιχειρήσεων στο λιανικό εμπόριο, τις τράπεζες και την υγειονομική περίθαλψη, οδήγησαν ορισμένους πελάτες στην Azure από την ανησυχία ότι η Amazon θα εξετάσει τη δραστηριότητα των εφαρμογών τους για να ενημερώσει τις δικές της ελλείψεις σε αυτούς τους κλάδους. [i.17]

Ένα άλλο στοιχείο σχετικά με τη σχετική πορεία του μεριδίου αγοράς των μεγάλων παρόχων υποδομών cloud προήλθε από έρευνα της JP Morgan που διεξήχθη τον Ιούνιο του 2020. Ρώτησαν 130 CIO μεγάλων επιχειρήσεων ποιο ποσοστό των δαπανών IaaS αντιστοιχεί σήμερα σε κάθε έναν από τους παρόχους cloud και στη συνέχεια ποιο ποσοστό θα περίμεναν σε 3 χρόνια. Τα αποτελέσματα ήταν αποκαλυπτικά - το AWS προβλεπόταν να μειωθεί κατά 4%, ενώ το GCP θα αυξανόταν κατά 4%. Το Azure είχε ήδη το μεγαλύτερο μερίδιο μεταξύ αυτής της ομάδας και προβλεπόταν να αυξηθεί ελαφρώς. [i.17]



Εικόνα 6: Σχετική πορεία του μεριδίου αγοράς σύμφωνα με την JPM



Προς επιβεβαίωση της έρευνας της JP Morgan, η Goldman Sachs διεξήγαγε τον Δεκέμβριο του 2019 παρόμοια έρευνα σε 100 στελέχη πληροφορικής σε εταιρείες του Global 2000 σχετικά με τις προγραμματισμένες δαπάνες τους για την πληροφορική. Από αυτά τα στελέχη, 56 χρησιμοποιούσαν το Azure για υποδομές cloud έναντι 48 που χρησιμοποιούσαν το AWS κατά τη στιγμή της έρευνας. Στη συνέχεια, τους ζητήθηκε να κοιτάξουν 3 χρόνια μπροστά. Όπως και στην έρευνα της JP Morgan, αυτές οι τάσεις αύξησης του μεριδίου για το Azure και το GCP συνεχίζονται. Σε 3 χρόνια, 66 από τα στελέχη πληροφορικής αναμένουν να χρησιμοποιούν το Azure, 64 το AWS και 30 το GCP. Το εντυπωσιακό σε αυτούς τους αριθμούς είναι ότι αποκαλύπτουν μια κίνηση προς την ανάπτυξη πολλαπλών υπολογιστικών νεφών. Το multi-cloud γίνεται η αχίλλειος πτέρνα της ηγεμονίας των προμηθευτών cloud σε αρκετές κατηγορίες υπηρεσιών λογισμικού. [i.17]

### 2.5.3 ΕΜΦΑΝΙΣΗ ΤΟΥ ΠΟΛΥΝΕΦΟΥΣ

Όταν το cloud hosting άρχισε να κερδίζει έδαφος από το 2010 έως το 2015, οι εταιρείες ξεκινούσαν τη μετάβασή τους στο νέφος με έναν μόνο πάροχο, συνήθως τον AWS. Αυτό συνέβη με το Netflix, το οποίο προχώρησε σε μια ολοκληρωμένη επένδυση στο AWS. Ωστόσο, καθώς το Azure και το GCP έχουν επεκτείνει δραματικά την εμβέλεια και την αξιοπιστία των βασικών δυνατοτήτων υπολογισμού και αποθήκευσης, οι επιχειρήσεις αντισταθμίζουν την ανάπτυξη του cloud.[i.17] Μια έρευνα της Gartner για την υιοθέτηση του cloud το 2019 έδειξε ότι από τις εταιρείες που χρησιμοποιούν το δημόσιο cloud, το 81% χρησιμοποιεί περισσότερους από έναν παρόχους υπηρεσιών cloud. Το οικοσύστημα του cloud επεκτείνεται πέρα από το πεδίο εφαρμογής ενός μόνο παρόχου υπηρεσιών cloud για τους περισσότερους πελάτες μεγάλων επιχειρήσεων. [i.19] Αυτό έχει καταστεί δυνατό χάρη στην εμπορευματοποίηση των βασικών δυνατοτήτων μεταξύ των AWS, Azure και GCP. Οι εταιρικοί οργανισμοί πληροφορικής χρησιμοποιούν πολλαπλούς παρόχους cloud για μερικούς λόγους: [i.17]

- **Αξιοπιστία.** Με εξαιρετικά υψηλές προσδοκίες για το χρόνο διαθεσιμότητας, οι μεγάλες επιχειρήσεις θέλουν να διασφαλίσουν ότι θα έχουν παρουσία σε περισσότερους από έναν πάροχο cloud σε περίπτωση διακοπής λειτουργίας σε όλο το σύστημα. Παρόλο που οι

περισσότεροι πάροχοι cloud λειτουργούν από πολλαπλές περιοχές για λόγους ανθεκτικότητας, τα προβλήματα υποδομής δικτύου ή λογισμικού θα μπορούσαν να επηρεάσουν τη διαθεσιμότητα ενός παρόχου. Τα σχέδια επιχειρησιακής συνέχειας απαιτούν όλο και περισσότερο μια διαμόρφωση πολλαπλών cloud. Τα SLA για τους προμηθευτές cloud ποικίλλουν ανάλογα με τον πάροχο και ακόμη και την υπηρεσία, αλλά γενικά δεν υπερβαίνουν το 99,99% του χρόνου διαθεσιμότητας. Αν και αυτό ακούγεται υψηλό, εξακολουθεί να επιτρέπει σχεδόν μία ώρα διακοπής λειτουργίας το χρόνο. Το SLA για το AWS Compute στοχεύει στο 99,99%, αλλά δίνει πλήρη επιστροφή χρημάτων μόνο αν η διαθεσιμότητα πέσει κάτω από το 95% σε ένα μήνα, πράγμα που μεταφράζεται σε μιάμιση ημέρα διακοπής λειτουργίας. [i.17]

- **Στάση διαπραγμάτευσης.** Εάν ένας πωλητής cloud γνωρίζει ότι ένας πελάτης αναπτύσσεται αποκλειστικά στην υποδομή του, αισθάνεται ότι ο πελάτης είναι επιρρεπής σε εγκλωβισμό. Αυτό οδηγεί σε λιγότερη επιρροή για τον πελάτη στις διαπραγματεύσεις για μαζικές μειώσεις τιμών κατά την ανανέωση της σύμβασης. Στους μεγάλους πελάτες κάθε παρόχου cloud ανατίθεται ένας αντιπρόσωπος πωλήσεων, ο οποίος έχει περιθώριο όσον αφορά την τιμολόγηση των μακροπρόθεσμων δεσμεύσεων δαπανών. Επίσης, οι προμηθευτές cloud προσφέρουν άλλα οφέλη, όπως κλιμακούμενη υποστήριξη και συμβουλευτική αρχιτεκτονική. Ο πελάτης έχει ευκολότερη πρόσβαση σε αυτά εάν ο πωλητής αισθάνεται ότι ανταγωνίζεται για τις δαπάνες. [i.17]
- **Χαρακτηριστικά.** Οι πάροχοι νέφους αναπτύσσουν εξειδικεύσεις και οι πελάτες μπορεί να θέλουν να επωφεληθούν από συγκεκριμένες υπηρεσίες σε κάθε πάροχο νέφους. Αυτή η δυνατότητα επιλογής για τους πελάτες προέκυψε ως αποτέλεσμα της διάσπασης των μονολιθικών εφαρμογών τους σε αυτόνομες υπηρεσίες. Οι υπηρεσίες μπορούν να αναπτυχθούν σε διαφορετικούς παρόχους νέφους, εφόσον το επιθυμούν. Για παράδειγμα, το GCP έχει μια ισχυρή πρακτική γύρω από το AI/ML, ενώ το Azure έχει επενδύσει σε μεγάλο βαθμό στο IoT. [i.17]
- **Γεωγραφική εξάπλωση.** Ορισμένοι πάροχοι cloud έχουν ισχυρότερη παρουσία σε ορισμένες γεωγραφικές περιοχές. Για παγκόσμιες επιχειρήσεις, οι απαιτήσεις εντοπισμού δεδομένων μπορεί να συνηγορούν υπέρ ενός διαφορετικού παρόχου hosting ανά χώρα. [i.17]

Η προτίμηση προς τη χρήση πολλαπλών προμηθευτών cloud δημιουργεί πλεονέκτημα για τους ανεξάρτητους παρόχους λογισμικού. Αυτό οφείλεται στο γεγονός ότι οι κοινές υπηρεσίες λογισμικού, όπως η πρόσβαση σε βάσεις δεδομένων, η δημιουργία ηλεκτρονικού ταχυδρομείου, η διαχείριση ταυτότητας ή το CDN, τείνουν να προσεγγίζονται μέσω μιας διεπαφής API. Η δομή των τελικών σημείων API, η οργάνωση των δεδομένων, η ροή εργασιών επανάκλησης και η επιχειρησιακή λογική είναι γενικά μοναδικές για κάθε υπηρεσία λογισμικού. Αυτό σημαίνει ότι μια προσαρμοσμένη εφαρμογή λογισμικού που δημιουργείται από μια επιχείρηση, όπως ένας ιστότοπος ηλεκτρονικού εμπορίου, ένα εργαλείο εξυπηρέτησης πελατών ή μια υπηρεσία διανομής μέσω ενδημέρωσης, θα πρέπει να προγραμματίσει τη διεπαφή API της εν λόγω υπηρεσίας προκειμένου να έχει πρόσβαση σε αυτήν. [i.17]

Για παράδειγμα, ένας οργανισμός μπορεί να αποφασίσει να χρησιμοποιήσει μια βάση δεδομένων προσανατολισμένη σε έγγραφα για μια από τις μικροπηρεσίες του, όπως το καλάθι αγορών ή τα δεδομένα προφίλ χρήστη. Αν φιλοξενούσε αυτή την εφαρμογή σε ξεχωριστές γεωγραφικές περιοχές τόσο στο AWS όσο και στο Azure, θα έπρεπε να χρησιμοποιήσει την υπηρεσία βάσης δεδομένων προσανατολισμένη στα έγγραφα που παρέχει η κάθε μία. Για το Azure, θα μπορούσαν να χρησιμοποιήσουν την CosmosDB. Στην AWS, θα μπορούσαν να χρησιμοποιήσουν την DocumentDB. Ωστόσο, πρόκειται για ελαφρώς διαφορετικές υλοποιήσεις. Τα API, οι λειτουργίες και οι τύποι δεδομένων που υποστηρίζονται από την DocumentDB είναι διαφορετικά από αυτά που υποστηρίζονται από την CosmosDB. [i.17]

Και οι δύο προσφέρουν συμβατότητα με την έκδοση 3.6 της MongoDB. Ωστόσο, η τελευταία έκδοση της MongoDB είναι η 4.4 (από τον Ιούνιο του 2020 σε beta). Εδώ πρέπει να αναφερθεί πως η MongoDB μετά την έκδοση της 3.6 απαγόρευσε από τους μεγαλύτερους προμηθευτές να χρησιμοποιούν την έκδοσή της και να πατεντάρουν πάνω της τις δικές τους υπηρεσίες βάσεων δεδομένων. Υπήρξαν πολυάριθμες βελτιώσεις στη μηχανή MongoDB μεταξύ των εκδόσεων 3.6 και 4.4. Αντί να διατηρεί μια εφαρμογή λογισμικού με ξεχωριστές διεπαφές API για την CosmosDB και την DocumentDB ή να καρφώνει το API σε μια παλιά έκδοση της MongoDB, η επιχείρηση θα μπορούσε απλώς να χρησιμοποιήσει το MongoDB Atlas. Το Atlas είναι η υπηρεσία της MongoDB που φιλοξενείται στο cloud και προσφέρει την τελευταία έκδοση της MongoDB. Είναι διαθέσιμη στις AWS, Azure και GCP. Σε αυτή την περίπτωση, η ομάδα μηχανικών της επιχείρησης θα πρέπει να διατηρεί μόνο ένα σύνολο κώδικα για τη διασύνδεση με τη

βάση δεδομένων προσανατολισμένη στα έγγραφα. Ακόμα και αν αναπτύσσονταν μόνο σε έναν πάροχο cloud επί του παρόντος, μια μελλοντική μετάβαση ή επέκταση σε άλλον πάροχο cloud δεν θα συνεπαγόταν καμία αλλαγή κώδικα. [i.17]

Αυτή η εξήγηση είναι λίγο τεχνική, αλλά είναι σημαντικό σημείο και ισχύει για όλους τους ανεξάρτητους παρόχους υπηρεσιών λογισμικού. Οι πελάτες μεγάλων επιχειρήσεων θέλουν να αποφεύγουν τον τεχνολογικό εγκλωβισμό, όπου είναι δυνατόν. Εάν υπάρχει μια εναλλακτική λύση σε μια υπηρεσία που παρέχεται από έναν προμηθευτή cloud και λειτουργεί σε όλους τους παρόχους cloud, είναι λογικό να τη χρησιμοποιούν. Αυτό το επιχείρημα ισχύει για όλους τους τύπους υπηρεσιών. Οι πρώτοι τύποι υπηρεσιών που επωφελήθηκαν από αυτή τη συνειδητοποίηση ήταν αυτοί που μπορούσαν να φιλοξενηθούν εντελώς ανεξάρτητα, όπως οι πληρωμές (Stripe, Adyen), οι επικοινωνίες (Twilio, Bandwidth) και το CDN (Akamai, Fastly, Cloudflare). Τώρα, οι βασικές υπηρεσίες υποδομής, όπως οι βάσεις δεδομένων (MongoDB Atlas), η αναζήτηση (Elastic Cloud) και η διαχείριση ταυτότητας (Okta), μπορούν να εκτελούνται εντός των περιόχων των παρόχων cloud στο υλικό τους. Η χρήση αυτών των ανεξάρτητων υπηρεσιών εξακολουθεί να παράγει έσοδα για τους παρόχους υπηρεσιών cloud για υπολογισμό και αποθήκευση, οπότε η υποστήριξή τους είναι επωφελής.[i.17]

#### 2.5.4 ΕΠΙΤΥΧΙΑ ΑΝΕΞΑΡΤΗΤΩΝ ΠΑΡΟΧΩΝ

Όσον αφορά τον εντοπισμό των ανεξάρτητων παρόχων λογισμικού που επωφελούνται από την εξειδίκευση σε μια κατηγορία, το τοπίο είναι ευρύ. Ο οποιοσδήποτε θα μπορούσε να εξετάσει οποιαδήποτε κατηγορία και να βρει τόσο δημόσιες όσο και ιδιωτικές εταιρείες που ανταγωνίζονται για την προσοχή. Παρακάτω έχουν δοθεί ορισμένα παραδείγματα παρόχων λογισμικού, αλλά ο κατάλογος αυτός δεν έχει σκοπό να είναι εξαντλητικός:[i.17]

- NoSQL Database: MongoDB (MDB), DataStax
- Search: Elastic (ESTC)
- Observability: Datadog (DDOG), Splunk (SPLK), Dynatrace (DT), New Relic (NEWR)
- CDN / Edge Compute: Fastly (FSLY), Cloudflare (NET)

- Identity Management: Okta (OKTA), Ping Identity (PING)
- Communications: Twilio (TWLO), Bandwidth (BAND)
- Data Streaming: Confluent
- Data Warehouse: Snowflake

Προηγουμένως, αναφέρθηκε πώς το multi-cloud ευνοεί εξ ορισμού τους ανεξάρτητους παρόχους. Οι διευθύνοντες σύμβουλοι πολλών ανεξάρτητων εταιρειών λογισμικού αναφέρονται σε αυτό ως ουδετερότητα. Ισχυρίζονται ότι οι οργανισμοί πληροφορικής των επιχειρήσεων δείχνουν ολοένα και περισσότερο προτίμηση σε έναν ουδέτερο πάροχο, όταν αυτό είναι δυνατόν, λόγω των ανησυχιών σχετικά με το κλείδωμα του προμηθευτή του cloud. Πέρα από την ουδετερότητα, πιστεύεται ότι υπάρχουν αρκετοί άλλοι παράγοντες που ωφελούν τους ανεξάρτητους παρόχους λογισμικού και θα συνεχίσουν να οδηγούν την ανάπτυξή τους στο μέλλον.[i.17]

#### 2.5.4.1 ΕΣΤΙΑΣΗ ΠΑΡΟΧΩΝ

Εστιάζοντας σε μια συγκεκριμένη ειδικότητα, ο ανεξάρτητος πάροχος λογισμικού είναι σε θέση να εμβαθύνει πολύ στην τεχνολογική του λύση. Αντί να δημιουργήσει μια υπηρεσία που αντιγράφει σε μεγάλο βαθμό ό,τι είναι διαθέσιμο στην αγορά σήμερα, αφιερώνει πόρους στην καινοτομία και στην αντιμετώπιση δυσκολότερων προβλημάτων. Το γεγονός ότι εκατοντάδες ή χιλιάδες άτομα που ασχολούνται με την ανάπτυξη προϊόντων (σχεδιασμός, μηχανική, δοκιμές, ανάλυση) αφιερώνουν καθημερινά χρόνο στην επανάληψη ενός συνόλου λειτουργιών σε μια ενιαία κατηγορία, θα πρέπει τελικά να οδηγήσει σε ένα καλύτερο προϊόν. Η ανατροφοδότηση των πελατών θα είναι στοχευμένη και σε μεγάλο βαθμό αφιltrάριστη, καθώς δεν θα θολώνεται από έναν διαχειριστή λογαριασμών που θα καλύπτει πολλές κατηγορίες. Πολλές φορές επωφελείται από την άμεση σχέση μεταξύ των ομάδων μηχανικών στους οργανισμούς πελατών και παρόχων.[i.17]

Η Elastic έχει σχεδόν 5 φορές μεγαλύτερο αριθμό προγραμματιστών κώδικα στο έργο Elastic GitHub απ' ό,τι η AWS στη χρηματοδοτούμενη έκδοση του Open Distro Elasticsearch. Ναι, η AWS θα μπορούσε να αποφασίσει ότι η υποδομή αναζήτησης είναι πιο στρατηγική γι' αυτήν και να αφιερώσει περισσότερους πόρους, αλλά δεν θα ήταν τόσο παραγωγικοί ή δημιουργικοί όσο οι μηχανικοί της Elastic που επέλεξαν να εργαστούν στο έργο ανοικτού κώδικα Elastic από την αρχή. Επίσης, κάθε πάροχος cloud θα πρέπει να διπλασιάσει αυτή την προσπάθεια λόγω του

multi-cloud ενώ δεν υπάρχουν τόσοι ειδικοί σε θέματα έρευνας που μπορούν να διατεθούν για ένα τέτοιο έργο. [i.17]

Η Cloudflare (NET) έχει ανακοινώσει ορισμένες σημαντικές βελτιώσεις στις λύσεις της χωρίς διακομιστές. Στο blog post του CEO της Cloudflare, επισημαίνονται διάφορα πλεονεκτήματα της serverless λύσης της Cloudflare σε σχέση με αυτή που προσφέρουν οι πάροχοι cloud. Είναι δυνατόν (και πιθανό) οι πάροχοι cloud να βελτιώσουν σταδιακά τις serverless λύσεις τους, αλλά και πάλι, με βάση όσα έχουν ήδη αναφερθεί θα πρέπει να υποστηρίξω ότι οι ανεξάρτητοι πάροχοι όπως η Cloudflare και η Fastly θα παραμένουν πάντα ένα βήμα μπροστά.[i.17]

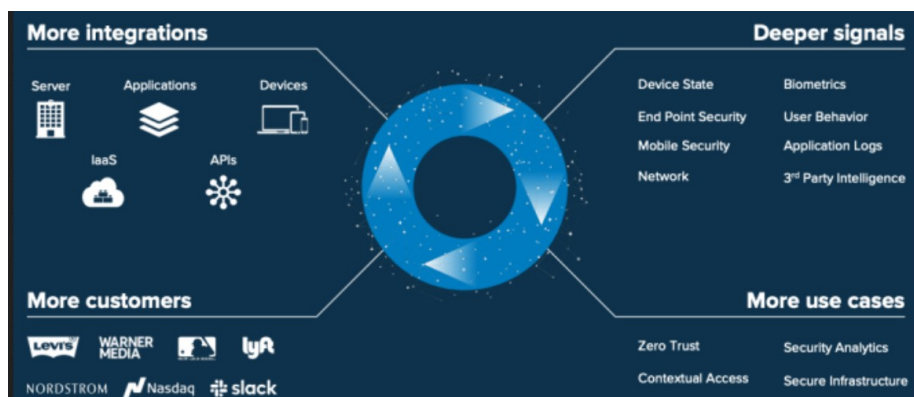
Εν κατακλείδι, δεν υπάρχει αμφιβολία πως οι μεγάλοι πάροχοι θα συνεχίσουν το έργο τους πάνω στο κομμάτι των cloud υπηρεσιών, προσφέροντας συνεχώς καινοτομίες και νέες υπηρεσίες σε διάφορους τομείς, αλλά είναι δύσκολο να πιστέψει κανείς πως θα καταφέρουν να το κάνουν το ίδιο αποτελεσματικά με έναν ανεξάρτητο πάροχο ο οποίος ασχολείται αποκλειστικά με τον ίδιο τομέα. Έτσι, η ποιοτικότερη παροχή υπηρεσιών από τους ανεξάρτητους παρόχους θα συνεχιστεί κανονικά για το επόμενο διάστημα. [i.17]

#### 2.5.4.2 ΕΚΜΕΤΑΛΛΕΥΣΗ ΤΩΝ ΔΙΚΥΩΝ

Καθώς μια υπηρεσία φτάνει σε μια κρίσιμη μάζα, αρχίζει να επωφελείται από τα αποτελέσματα του δικτύου. Εξετάζοντας τη συμπεριφορά των χρηστών στο πλαίσιο των υπηρεσιών τους, οι πάροχοι μπορούν να αντλήσουν πληροφορίες που μπορούν να βελτιώσουν το προϊόν. Αυτό μπορεί να πάρει τη μορφή λιγότερων σφαλμάτων, πιο λεπτομερούς λογικής ή περισσότερων συνδέσεων με εξωτερικά συστήματα. Η παρατήρηση της συμπεριφοράς των πελατών μπορεί επίσης να επηρεάσει μελλοντικές βελτιώσεις του προϊόντος ή να γεννήσει ολοκαίνουργια προϊόντα. [i.17]

Το Okta αποτελεί ένα καλό παράδειγμα της δύναμης των επιδράσεων του δικτύου. Ως ο μεγαλύτερος ανεξάρτητος πάροχος λύσεων διαχείρισης ταυτότητας, η πλατφόρμα της επωφελείται από έναν συνεχή κύκλο ανατροφοδότησης (Feedback) περισσότερων δεδομένων από πελάτες, ενσωματώσεις, συσκευές και περιπτώσεις χρήσης. Αξιοποιώντας όλες αυτές τις εισροές, το Okta είναι σε θέση να βελτιώνει διαρκώς την αποτελεσματικότητα των λύσεων ταυτότητάς του. Συγκεκριμένα, μπορούν να εξετάζουν την ανώμαλη συμπεριφορά μιας ομάδας χρηστών και να προσαρμόζουν γρήγορα τους κανόνες ασφαλείας ώστε να απαιτούν ένα επιπλέον επίπεδο πιστο-

ποίησης ταυτότητας. Η CrowdStrike επιτυγχάνει τα ίδια οφέλη για τις λύσεις EDR/EPP της. Αυτές οι επιδράσεις δικτύου συμβάλλουν σε μια συνεχώς αυξανόμενη τάφρο, καθιστώντας δυσκολότερο τον αγώνα των παρόμοιων λύσεων από τους μεγάλους παρόχους cloud. [i.17]



Εικόνα 7: Okta Investor Day FY21, April 1, 2020

### 2.5.4.3 ΕΠΙΛΟΓΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΤΩΝ

Οι ταλαντούχοι εργαζόμενοι γενικά στρέφονται προς τους ανεξάρτητους παρόχους λογισμικού. Αυτό οφείλεται κυρίως στο γεγονός ότι υπάρχουν μεγαλύτερες οικονομικές ευκαιρίες και μεγαλύτερη ανάπτυξη καριέρας. Το μεγαλύτερο μέρος της ανόδου έχει σημειωθεί στην Amazon, τη Microsoft ή τη Google σε αυτό το σημείο. Ένας νέος εργαζόμενος μέσα σε μία τέτοια εταιρία κινδυνεύει να χαθεί μέσα στο πλήθος. Είναι ευκολότερο να φανταστεί κανείς ότι οι μετοχές των DDOG, FSLY, OKTA, TWLO, NET κ.λπ. συνεχίζουν να αυξάνονται σε πολλαπλάσια επίπεδα αποτίμησης από αυτές των μεγάλων παρόχων. Το πιο σημαντικό για τους φιλόδοξους εργαζόμενους είναι ότι οι μικρότερες, ταχέως αναπτυσσόμενες εταιρείες προσφέρουν μεγαλύτερη επαγγελματική εξέλιξη και την αίσθηση ότι η συμβολή τους επηρεάζει άμεσα την επιτυχία της εταιρείας. [i.17]

Οι πάροχοι υπηρεσιών νέφους προσελκύουν ταλέντα στο εισαγωγικό επίπεδο και στα ανώτερα διοικητικά στελέχη. Για τους αποφοίτους κολεγίου που δεν είναι εξοικειωμένοι με το τεχνολογικό τοπίο, μια θέση εργασίας για αρχάριους σε έναν γίγαντα του νέφους αποτελεί ένα εξαιρετικό περιβάλλον για να αποκτήσουν τα πρώτα τους επαγγελματικά προσόντα. Ωστόσο, μόλις αποκτήσουν μερικά χρόνια εμπειρίας, τείνουν να φεύγουν. Στις ανώτερες βαθμίδες, όπου τα πακέτα αμοιβών είναι προσοδοφόρα, οι προμηθευτές cloud μπορούν επίσης να προσελκύσουν τα-

λέντα. Είναι η ευρύτερη μεσαία περιοχή όπου συμβαίνει η καινοτομία και αυτό τείνει να ευνοεί τις μικρότερες, ανεξάρτητες εταιρείες.[i.17]





### 3 ΑΝΕΞΑΡΤΗΣΙΑ - ΧΕΙΡΙΣΜΟΣ ΔΕΔΟΜΕΝΩΝ ΣΤΟ ΝΕΦΟΣ

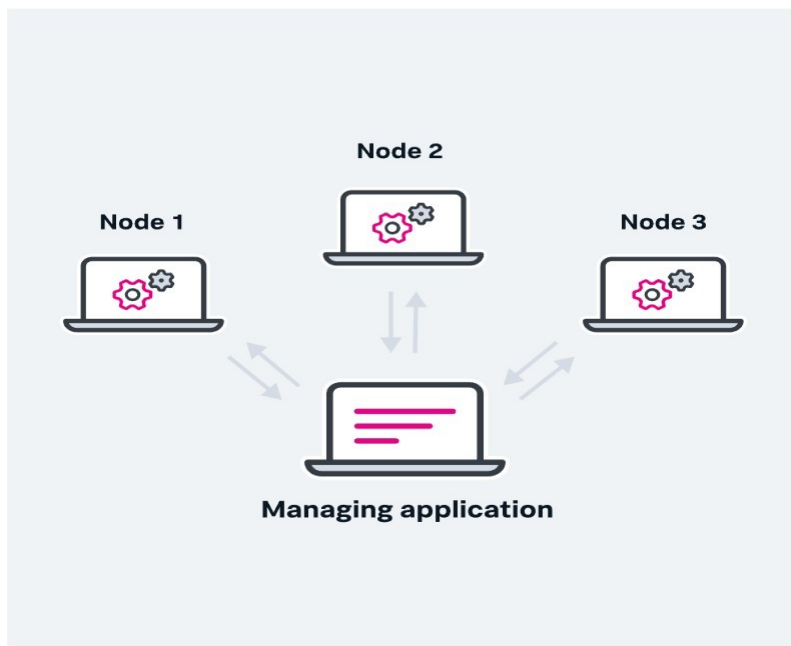
Πριν εισέλθουμε στα τετριμμένα της συλλογής δεδομένων, της ανεξαρτησίας στο νέφος και κατ' επέκταση στο διαδίκτυο, είναι άξιο αναφοράς να μιλήσουμε περιληπτικά για τα κατανεμημένα συστήματα, που αποτέλεσαν σταθμό στην εξέλιξη του υπολογιστικού νέφους, ενώ ακόμα χρησιμοποιούνται ως βασική υποδομή σε συστήματα που επεξεργάζονται μεγάλο όγκο υπολογιστικών δεδομένων.

#### 3.1 ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ

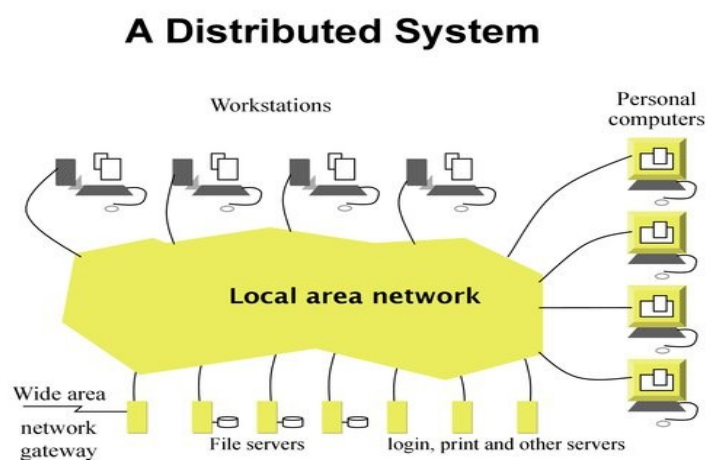
Αρχικά, θα πρέπει να επεξηγηθεί τι είναι ένα κατανεμημένο σύστημα. Ένα κατανεμημένο σύστημα είναι ένα πεδίο της επιστήμης των υπολογιστών και στην εφαρμογή του αποτελείται από πολλαπλά αυτόνομα υπολογιστικά συστήματα, που επικοινωνούν μέσω ενός δικτύου και λειτουργούν σαν μία ενιαία οντότητα. Η επικοινωνία αυτή αποσκοπεί στην επίτευξη ενός κοινού σκοπού. Ένας τέτοιος σκοπός συνήθως είναι η ανάγκη να επεξεργαστούν δεδομένα που απαιτούν μεγάλο όγκο υπολογιστικών πόρων, με απώτερο στόχο τη λύση κάποιου υπολογιστικού προβλήματος. Σε ένα κατανεμημένο σύστημα, το πρόβλημα διαχωρίζεται σε πολλαπλές εργασίες, όπου η κάθε μία αναλαμβάνεται από έναν ή περισσότερους υπολογιστές. Επίσης, ένα τέτοιο σύστημα μπορεί να παραμετροποιηθεί με ποικίλους τρόπους, δηλαδή μπορεί να αποτελείται από κεντρικούς υπολογιστές, προσωπικούς υπολογιστές, workstations, minicomputers κ.λ.π. Τελικός σκοπός είναι αυτό το δίκτυο των διαφορετικών υπολογιστικών μηχανών να λειτουργεί σαν ένας ενιαίος υπολογιστής.[i.15][i.16]

Το υπολογιστικό νέφος συχνά συγχέεται με την νόηση ότι είναι μία ακόμα εφαρμογή των κλασικών κατανεμημένων συστημάτων. Με μία πρώτη ματιά, πράγματι φαίνεται να είναι έτσι, αλλά το υπολογιστικό νέφος επικεντρώνεται κυρίως σε μία άλλη λογική η οποία περιστρέφεται γύρω από την δυναμικότητα των υπηρεσιών, την προσαρμοστικότητα, αλλά και την αύξηση της

εμβέλειας ενός τέτοιου συστήματος σε παγκόσμια κλίμακα. Εν ολίγοις, γκρεμίζει τα σύνορα μεταξύ οργανισμών, κάτι που δεν επιτυγχάνεται εύκολα σε ένα κλασικό κατακεντρωμένο σύστημα.



Εικόνα 8: Παράδειγμα κατακεντρωμένου συστήματος



Εικόνα 9: Παράδειγμα 2 Κατακεντρωμένου συστήματος

### 3.2 ΑΝΕΞΑΡΤΗΣΙΑ ΣΤΟ ΝΕΦΟΣ

Σε αυτό το κεφάλαιο θα εξεταστεί η στρατηγική που πρέπει να ακολουθηθεί από κάποια εταιρία, η οποία επιθυμεί να μεταβεί στο νέφος. Το πραγματικό πλάνο που θα πρέπει να υλοποιηθεί στην πραγματικότητα, δεν είναι το πώς θα μεταβεί κάποιος στο νέφος, αλλά αν θα μπορεί έπειτα να μεταβεί σε έναν άλλο πάροχο ή ακόμα και να μεταφέρει όλο το σύστημά του πίσω στην τοπική υποδομή του. Μία επιτυχής μετάβαση στο cloud, σημαίνει άμεσα πως θα μπορείς ανά πάσα στιγμή να εξαχθείς από τις υπηρεσίες της χωρίς να φτάσεις στο σημείο του εγκλωβισμού. Παρακάτω θα δούμε κάποια σενάρια κατά της Sara Mazer, η οποία είναι μία Αρχιτέκτονας επιχειρήσεων και σύμβουλος λύσεων, που βοηθούν ή προειδοποιούν μία επιχείρηση να αποφύγει τον εγκλωβισμό του συστήματός της εντός ενός παρόχου νέφους.

#### 3.2.1 ΤΕΡΜΑΤΙΣΜΟΣ ΥΠΗΡΕΣΙΩΝ

Υπάρχουν γενικά περιπτώσεις πελατών, που έπειτα από τερματισμό της συνεργασίας τους με κάποιο προμηθευτή cloud υπηρεσιών, ζητούν τα δεδομένα τους από τον προμηθευτή και αντιμετωπίζουν σοβαρό πρόβλημα. Το πρόβλημα είναι ότι πολλές φορές, αυτά τα δεδομένα για να τα λάβει ο πελάτης μπορεί να καθυστερήσουν υπερβολικά πολύ, ενώ μπορεί να τα λάβει σε τέτοια μορφή, που να απαιτείται τεράστιος φόρτος εργασίας για την επεξεργασία τους και την αξιοποίησή τους. Αυτό είναι ένα πρόβλημα που αντιμετωπίζουν συνήθως χρήστες, που δεν γνωρίζουν αρκετά το πώς πρέπει να κινηθούν σε μία έναρξη συνεργασίας με κάποιον προμηθευτή, οπότε καταλήγουν να θυματοποιούνται από φτωχά τεκμηριωμένα συμβόλαια. [i.20]

Στην ουσία αυτό που προτείνεται για να αποφευχθούν περιπτώσεις σαν την παραπάνω είναι τα εξής:

- Βεβαιωθείτε ότι υπάρχει σαφής ρήτρα στη σύμβασή σας που αναφέρει τη συγκεκριμένη μορφή, το μηχανισμό μεταφοράς και το χρονοδιάγραμμα που θα χρησιμοποιηθεί για την επιστροφή των δεδομένων σας.
- Σχεδιάστε τις λειτουργίες σας που φιλοξενούνται στο cloud με την προσδοκία ότι τα δεδομένα και τα σχετικά μεταδεδομένα, οντότητες και προνόμια είναι ανεξάρτητα από την εφαρμογή.

### 3.2.2 ΑΝΕΞΑΡΤΗΤΗ ΠΡΟΣΒΑΣΗ ΣΤΑ ΔΕΔΟΜΕΝΑ

Η Sara Mazer συνεχίζει λέγοντας, πως εάν ο έλεγχος πρόσβασης στα δεδομένα μίας εταιρίας είναι ανεξάρτητος από τον προμηθευτή του νέφους, αυτό βοηθά με πολλούς τρόπους. Όχι μόνο μπορεί να αποφευχθούν σενάρια όπως τα παραπάνω, αλλά απλοποιεί τη διαδικασία αλλαγής συστημάτων ολοκλήρωσης, προμηθευτών cloud ή υποπρομηθευτών. Ο εξωτερικός έλεγχος πρόσβασης σε συνδυασμό με την κρυπτογράφηση σημαίνει επίσης ότι όταν μετακινούνται δεδομένα, αυτά μπορούν να παραμείνουν κρυπτογραφημένα και ασφαλή. [i.20]

Πάνω σε αυτό το σενάριο δίνει τις εξής συμβουλές:

- Η διατήρηση της άμεσης πρόσβασης στην κεντρική βάση δεδομένων και στις διαδικτυακές υπηρεσίες που υποστηρίζουν τα δεδομένα της εταιρίας θα βελτιώσει την ευελιξία της μελλοντικής της ανάπτυξης.
- Η δημιουργία των δικών της προσαρμοσμένων εφαρμογών και η χρήση ενός παρόχου cloud κυρίως για υπηρεσίες υποδομής μπορεί επίσης να βοηθήσει να διατηρηθεί η κυριότητα και ο έλεγχος των δεδομένων.
- Η υλοποίηση του ελέγχους πρόσβασης σε πολιτικές και όχι σε οτιδήποτε είναι ειδικό για το εκάστοτε περιβάλλον. Πάρτε ονόματα ρόλων από έναν κατάλογο, χρησιμοποιήστε μια πολιτική που μπορεί να αναλυθεί και μην χρησιμοποιείτε κώδικα στο cloud για να καθοδηγήσετε την πρόσβαση.

### 3.2.3 ΑΝΕΞΑΡΤΗΤΑ ΕΡΓΑΛΕΙΑ ΤΡΙΤΩΝ & SLA

Μία ακόμα συμβουλή της Sara Mazer είναι να “βεβαιωθείτε ότι χρησιμοποιείτε το καλύτερο εργαλείο για μια εργασία, αντί να βασίζεστε μόνο στα εργαλεία ενός προμηθευτή cloud, ειδικά όταν πρόκειται για τη διακυβέρνηση δεδομένων ή την ποιότητα δεδομένων. Εάν κάνετε μια αλλαγή στον έλεγχο πρόσβασης και χρειάζονται τρεις ημέρες για να τεθεί σε ισχύ, θα ήταν εύκολο να το διαπιστώσετε αυτό χρησιμοποιώντας μόνο το εργαλείο ελέγχου ενός προμηθευτή cloud; Οι πωλητές cloud δεν έχουν κίνητρο να ελέγχουν πιθανές αναποτελεσματικότητες, ούτε έχουν κίνητρο να κάνουν αποδιπλασιασμό ή κλάδεμα των δεδομένων τους. Θα πρέπει να εξεταστούν εξωτερικά αρχεία καταγραφής ελέγχου υπό ανεξάρτητο έλεγχο, ασφαλή και κρυπτογραφημένα και κεντρικά προσβάσιμα. Προχωρώντας ακόμη περισσότερο, οι συμφωνίες SLA θα πρέπει να είναι επίσης ουδέτερες ως προς τον προμηθευτή.”[i.20]

Εν συνέχεια η λύση που προτείνεται στο συγκεκριμένο πρόβλημα είναι η εξής:

- Βεβαιωθείτε ότι διαθέτετε ανεξάρτητα εργαλεία για την παρακολούθηση των δεδομένων σας και των μετρήσεων SLA. Εξετάστε το ενδεχόμενο να αποθηκεύετε τα αρχεία καταγραφής πρόσβασης και τις μετρήσεις SLA σε μια ασφαλή, εξωτερική βάση δεδομένων.

### 3.2.4 ΟΥΔΕΤΕΡΕΣ ΤΕΧΝΟΛΟΓΙΕΣ CLOUD

Επιστρέφοντας στη συζήτηση για το κλείδωμα του προμηθευτή η Sara Mazer αναφέρει πως μερικά πράγματα που πρέπει να προσέξει ο πελάτης κατά την επιλογή ουδέτερων τεχνολογιών για το νέφος, είναι το πραγματικό λογισμικό που θα εκτελείται στο νέφος. Πολλές συμφωνίες αδειών χρήσης λογισμικού είναι συγκεκριμένες για το πού εκτελείται: cloud, iron ή virtual. Πρέπει να εξασφαλίσει κάποιος πως έχει μια ρευστή συμφωνία, ώστε το λογισμικό να εκτελείται εκεί που θέλει ο ίδιος. [i.20] Επιπλέον, το λογισμικό θα πρέπει να είναι σε θέση να:

- Εκτέλεση σε οποιοδήποτε νέφος ή σε διάφορους προμηθευτές νέφους ταυτόχρονα, καθώς και on-premise
- Υποστήριξη πολλαπλών μορφών αποθήκευσης δεδομένων και βιομηχανικών προτύπων
- Εύκολη αντιγραφή δεδομένων σε διάφορα περιβάλλοντα

- Προσφέρει τη δική του ανεξάρτητη ασφάλεια (έλεγχος πρόσβασης ανεξάρτητα από έναν προμηθευτή cloud)
- Έχουν αποδεδειγμένο ιστορικό επιτυχίας για μεταναστεύσεις κέντρων δεδομένων, αλλαγές διαχείρισης, με μηδενικό χρόνο διακοπής λειτουργίας

Επιπλέον, ιδιαίτερη έμφαση όσον αφορά το λογισμικό με βάση τη Sara Mazer θα πρέπει να δοθεί στα εξής:

- Η εξέταση προβλημάτων λογικής δεδομένων και φορητότητας κανόνων. Για παράδειγμα, το JSON μπορεί να μετακινείται εύκολα μεταξύ των προμηθευτών αποθήκευσης βάσεων δεδομένων, αλλά η διαχείριση του JSON είναι εντελώς διαφορετική στην τεχνολογία κάθε προμηθευτή. Ωστόσο, το XPath είναι ένα πρότυπο και μπορεί να μεταφερθεί μεταξύ των προμηθευτών.
- Είτε είναι ιδιόκτητο είτε ανοικτού κώδικα, το λογισμικό θα πρέπει να επιτρέπει σε ένα Διευθυντικό στέλεχος πληροφοριών (Chief Information Officer) την ευελιξία που χρειάζεται για να κάνει ό,τι είναι καλύτερο για την επιχείρησή του.

Συμπερασματικά, η Sara Mazer καταλήγει, πως “η υπόσχεση του νέφους είναι να επιτρέψει στους χρήστες να αποθηκεύουν και να εκτελούν εφαρμογές από οπουδήποτε. Πολλά έχουν ειπωθεί για την ευκολία της μετάβασης στο νέφος. Ωστόσο, από τη στιγμή που βρίσκεται εκεί, είναι συχνά δύσκολο να το εγκαταλείψει κανείς. Εφόσον οι εφαρμογές σας στο σύννεφο αναπτύσσονται έτσι ώστε να είναι ουδέτερες στο σύννεφο εκ των προτέρων, χρησιμοποιείτε λογισμικό που επιτρέπει την εύκολη μετακίνηση των μορφών αποθήκευσης, των API και των γλωσσών σας, θα πετύχετε.”[i.20]

### 3.3 ΑΠΟΚΕΝΤΡΩΣΗ

Σε αυτό το κεφάλαιο θα απομακρυνθούμε λίγο από τα τετριμμένα του cloud και θα αναλυθεί η φιλοσοφία της αποκέντρωσης (decentralization), που το τελευταίο διάστημα έχει γίνει αντικείμενο σχολιασμού και έρευνας στον τομέα της πληροφορικής και του διαδικτύου. Εδώ θα αναφερθούμε σε μία άλλη μορφή ανεξαρτησίας, πιο προσωπική, αλλά άμεσα συνδεδεμένη με

τον κόσμο του cloud και των υπηρεσιών που προσφέρονται στους χρήστες σήμερα. Αρχικά, ως αποκέντρωση αναφερόμαστε στη διαδικασία κατά την οποία όλες οι δραστηριότητες ενός οργανισμού, ειδικά εκείνες που σχετίζονται με τον σχεδιασμό και την λήψη αποφάσεων, κατανέμονται ή ανατίθενται μακριά από μία κεντρική, εξουσιαστική θέση ή ομάδα. Από τον ορισμό και μόνο μπορεί να γίνει αντιληπτό, το κατά πόσο ένα τέτοιο σύστημα είναι επιθυμητό ειδικά σε ορισμένες περιπτώσεις. Ειδικά το τελευταίο διάστημα, έχουν αναδυθεί πολλές κατηγορίες εναντίων πολλών μέσων κοινωνικής δικτύωσης, για τα δικαιώματα των χρηστών, αλλά και για τον έλεγχο που ασκείται από τις αντίστοιχες εταιρίες ή και κυβερνήσεις.

Μιλώντας για decentralized κοινωνικά δίκτυα, πρέπει αρχικά να αναλυθεί σε τι ακριβώς αναφερόμαστε. Τα αποκεντρωμένα κοινωνικά δίκτυα λειτουργούν σε ανεξάρτητα διαχειριζόμενος διακομιστές και όχι σε έναν κεντρικό διακομιστή που ανήκει σε μια επιχείρηση. Το Mastodon είναι ένα παράδειγμα αποκεντρωμένου κοινωνικού δικτύου. Βασίζεται σε λογισμικό ανοιχτού κώδικα και λειτουργεί σε μεγάλο βαθμό όπως το Twitter. Ένα άλλο παράδειγμα είναι το Steem, το οποίο λειτουργεί σε μια κοινωνική αλυσίδα μπλοκ (blockchain). Η τεχνολογία blockchain επιτρέπει την αποθήκευση καταχωρίσεων δεδομένων σε διακομιστές οπουδήποτε στον κόσμο. Ενισχύει τη διαφάνεια, καθώς τα δεδομένα μπορούν να προβληθούν σχεδόν σε πραγματικό χρόνο από οποιονδήποτε σε ένα δίκτυο. Τα αποκεντρωμένα κοινωνικά δίκτυα παρέχουν στους χρήστες περισσότερο έλεγχο και αυτονομία. Ένα άτομο μπορεί να δημιουργήσει το κοινωνικό του δίκτυο και να καθορίσει τον τρόπο λειτουργίας του και το τι μπορούν να λένε οι χρήστες. Αντί να παρακολουθείται το περιεχόμενο από μια εταιρεία, ο ιδρυτής ενός αποκεντρωμένου κοινωνικού δικτύου μπορεί να καθορίσει τους όρους αποδεκτής συμπεριφοράς για τον ιστότοπό του.[i.21]

Επιπλέον, τα αποκεντρωμένα κοινωνικά δίκτυα συνθέτουν το fediverse, έναν όρο για μια συλλογή διασυνδεδεμένων διακομιστών που χρησιμοποιούνται για την κοινωνική δικτύωση και άλλες δραστηριότητες όπως το blogging και η δημοσίευση στο διαδίκτυο. Ένα ανεξάρτητα φιλοξενούμενο federated δίκτυο μπορεί να αλληλεπιδράσει με άλλα δίκτυα στο fediverse. Αυτή είναι μία από τις κύριες διαφορές μεταξύ των αποκεντρωμένων κοινωνικών δικτύων και των δημοφιλών μέσων κοινωνικής δικτύωσης, όπως το Facebook και το Twitter. Για παράδειγμα, το Twitter επιτρέπει στους χρήστες να στέλνουν και να λαμβάνουν μηνύματα μόνο σε άλλα άτομα με λογαριασμούς Twitter (π.χ. οι χρήστες του Twitter δεν μπορούν να στέλνουν μηνύματα σε λογαρια-



σμούς Facebook, επειδή δεν υπάρχει ευθυγράμμιση μεταξύ των δύο). Τα federated δίκτυα, από την άλλη πλευρά, επιτρέπουν στους χρήστες να εμπλέκονται σε όλες τις πλατφόρμες.[i.21]

Το ηλεκτρονικό ταχυδρομείο προσφέρει ένα παράδειγμα για το πώς λειτουργούν τα ομοσπονδιακά (federated) κοινωνικά δίκτυα. Πάρτε, για παράδειγμα, τη Google και τη Yahoo. Κάθε εταιρεία θέτει κανόνες ηλεκτρονικού ταχυδρομείου για τους χρήστες της. Η Google δεν επιβάλλει κανονισμούς στους χρήστες της Yahoo. Ωστόσο, οι χρήστες της Google μπορούν να στέλνουν μηνύματα ηλεκτρονικού ταχυδρομείου σε χρήστες της Yahoo και να λαμβάνουν μηνύματα ηλεκτρονικού ταχυδρομείου από αυτούς και αντίστροφα. Τα ομοσπονδιακά (federated) δίκτυα λειτουργούν με παρόμοιο τρόπο.[i.21]

### 3.3.1 ΠΛΕΟΝΕΚΤΗΜΑΤΑ - ΜΕΙΟΝΕΚΤΗΜΑΤΑ

Τα μέσα κοινωνικής δικτύωσης προωθούν τη συνδεσιμότητα, τη δημιουργία κοινοτήτων και την ανταλλαγή γνώσεων. Οι άνθρωποι μπορούν να χρησιμοποιούν τα μέσα κοινωνικής δικτύωσης για να προωθήσουν κοινωνικές και πολιτικές αλλαγές, να ευαισθητοποιήσουν για σημαντικά ζητήματα, να συγκεντρώσουν χρήματα για όσους έχουν ανάγκη και να προωθήσουν τις επιχειρήσεις τους. Ωστόσο, η άσχημη πλευρά των μέσων κοινωνικής δικτύωσης μπορεί να περιλαμβάνει διαδικτυακό εκφοβισμό, πολιτική παραπληροφόρηση, ακόμη και εγκληματική δραστηριότητα. Επειδή τα αποκεντρωμένα κοινωνικά δίκτυα είναι σε μεγάλο βαθμό μη ελεγχόμενα, τόσο τα θετικά όσο και τα αρνητικά αποτελέσματα γίνονται πιο ακραία.[i.21]

#### 3.3.1.1 Έλεγχος χρηστών, ελευθερία λόγου και αντίσταση στη λογοκρισία

Οι εταιρικές μονάδες ελέγχουν τους μεγάλους ιστότοπους κοινωνικής δικτύωσης και μια μικρή ομάδα ανθρώπων σε αυτές τις εταιρείες καθορίζει τους κανόνες συμμετοχής. Αυτό έχει εγείρει ανησυχίες σχετικά με την ελευθερία του λόγου και τη λογοκρισία μεταξύ των χρηστών. Πρόσφατα, το Facebook θέσπισε υψηλού προφίλ απαγορεύσεις σε άτομα από όλες τις πλευρές του πολιτικού φάσματος, από τον Λούις Φαρραχάν έως τον Άλεξ Τζόουνς. Η απαγόρευση βιαιών, μισητών και επικίνδυνων μηνυμάτων βοηθά στην προστασία των χρηστών των μέσων κοινωνικής δικτύωσης από κακόβουλες διαδικτυακές δραστηριότητες, αλλά ορισμένοι πιστεύουν ότι οι απαγορεύσεις έρχονται σε αντίθεση με τα ιδανικά της ελευθερίας του λόγου.[i.21]

Ένα αποκεντρωμένο κοινωνικό δίκτυο επιτρέπει στους χρήστες μεγαλύτερο έλεγχο. Σε αντίθεση με τις κεντρικές πλατφόρμες κοινωνικής δικτύωσης, τα ομοσπονδιακά δίκτυα προωθούν την ανεξαρτησία χωρίς κεντρική αρχή. Στα οφέλη περιλαμβάνονται η αντίσταση στη λογοκρισία, η κυριότητα των προσωπικών δεδομένων και ο βελτιωμένος έλεγχος του περιεχομένου που δημιουργείται από τους χρήστες. Με άλλα λόγια, οι χρήστες δεν αποδέχονται τη λογοκρισία και επιμένουν να έχουν τον τελικό λόγο για το περιεχόμενό τους. Αυτό σημαίνει ότι κανένας άλλος, είτε πρόκειται για εταιρεία είτε για διαχειριστή ιστότοπου, δεν μπορεί να κάνει τροποποιήσεις στο περιεχόμενο που δημιουργούν οι χρήστες. Επίσης, κανείς δεν μπορεί να αφαιρέσει το περιεχόμενο που δημιουργούν οι χρήστες.[i.21]

Σε ένα ομοσπονδιακό δίκτυο, καμία ομάδα δεν μπορεί να υπαγορεύει τους κανόνες άλλων ομάδων. Για παράδειγμα, οποιοσδήποτε στο Mastodon μπορεί να διαχειρίζεται το δικό του ιστότοπο κοινωνικής δικτύωσης χωρίς κεντρική αρχή, πράγμα που σημαίνει ότι (και άλλοι χρήστες) μπορούν να δημοσιεύουν ό,τι θέλουν χωρίς να ανησυχούν ότι η δημοσίευσή τους θα κατέβει. Ένα μειονέκτημα αυτής της δομής είναι ότι οι ομάδες μίσους έχουν επίσης την ελευθερία να λανσάρουν τους δικούς τους ιστότοπους κοινωνικής δικτύωσης. Ενώ τα άτομα μπορούν να μπλοκάρουν αυτές τις ομάδες, δεν μπορούν να τις εμποδίσουν να συμμετέχουν στο δίκτυο.[i.21]

### **3.3.1.2 Προσωπικά δεδομένα, ιδιωτικό απόρρητο και ασφάλεια**

Οι ανησυχίες των χρηστών σχετικά με τον έλεγχο των προσωπικών τους δεδομένων οδήγησαν στη θέσπιση του Γενικού Κανονισμού για την Προστασία Δεδομένων (GDPR) στην Ευρώπη. Η νομοθεσία θεωρεί τους χρήστες "υπεύθυνους ελέγχου των δεδομένων". Οι εταιρείες μέσων κοινωνικής δικτύωσης είναι γνωστές ως "υπεύθυνοι επεξεργασίας δεδομένων". Ο ορισμός του GDPR για τον υπεύθυνο ελέγχου δεδομένων σημαίνει ότι οι χρήστες είναι κύριοι των δικών τους δεδομένων. Βάσει του νόμου, οι εταιρείες πρέπει να παραδώσουν περισσότερο έλεγχο των προσωπικών δεδομένων στους χρήστες, τουλάχιστον όσες εδρεύουν στην Ευρώπη. Οι εταιρείες τιμωρούνται για τη μη τήρηση των κανονισμών του GDPR.[i.21]

Τα αποκεντρωμένα κοινωνικά δίκτυα έχουν δώσει μια άλλη απάντηση στο θέμα της ιδιωτικότητας και της ασφάλειας των δεδομένων. Στα ομοσπονδιοποιημένα κοινωνικά δίκτυα, οι χρήστες μπορούν να δημιουργούν λογαριασμούς χωρίς να χρειάζεται να συνδέονται με ταυτότητες του πραγματικού κόσμου, όπως διευθύνσεις ηλεκτρονικού ταχυδρομείου ή αριθμούς τηλε-

φώνου. Επιπλέον, αυτά τα δίκτυα βασίζονται συχνά στην κρυπτογραφία δημόσιου κλειδιού για την ασφάλεια των λογαριασμών, αντί να βασίζονται σε έναν μόνο οργανισμό για την προστασία των δεδομένων των χρηστών.[i.21]

Ενώ αυτό μπορεί να δημιουργήσει πλεονεκτήματα από την άποψη της ασφάλειας των δεδομένων, παρουσιάζει επίσης προκλήσεις. Για παράδειγμα, τα bootstrapped ομοσπονδιακά κοινωνικά δίκτυα μπορεί να κλείσουν λόγω έλλειψης κεφαλαίων, με αποτέλεσμα οι χρήστες να χάσουν τα δεδομένα και τις συνδέσεις τους. Σε αυτή την περίπτωση, οι χρήστες δεν έχουν απλό τρόπο να επανασυνδεθούν με άλλους χρήστες του δικτύου, επειδή τα ομοσπονδιακά δίκτυα δεν διατηρούν αρχεία προσωπικών δεδομένων σε διακομιστές. Όσον αφορά την προστασία της ιδιωτικής ζωής, αυτές οι πλατφόρμες δεν κρυπτογραφούν απαραίτητα τα δεδομένα, πράγμα που σημαίνει ότι τα ιδιωτικά μηνύματα μπορεί να είναι ορατά στους διαχειριστές.[i.21]

### 3.3.1.3 Οικονομική ουδετερότητα

Η οικονομική ουδετερότητα αποτελεί βασικό ιδανικό για πολλούς που στρέφονται στα αποκεντρωμένα κοινωνικά δίκτυα - επιθυμούν να απελευθερωθούν από την παρεμβατική διαφήμιση και τον κίνδυνο για την ιδιωτικότητα που αυτή ενέχει. Τα ομοσπονδιακά δίκτυα αναζητούν νέες μορφές νομιμοποίησης για να παραμείνουν φερέγγυα. Συχνά χρησιμοποιούν μια μορφή ψηφιακού νομίσματος, όπως το Bitcoin, για να διατηρούν τις λειτουργίες τους σε λειτουργία. Για παράδειγμα, το Steem πληρώνει τους χρήστες του για τη δημιουργία ή την επιμέλεια ενδιαφέροντος περιεχομένου, γεγονός που δίνει κίνητρο στους δημιουργούς περιεχομένου να επικεντρωθούν στην ποιότητα. Η Steem λαμβάνει τα χρήματά της από επενδυτές που πιστεύουν ότι η πλατφόρμα θα αυξηθεί σε αξία με την πάροδο του χρόνου και θα είναι μια μέρα κερδοφόρα. [i.21]

### Βαθμός Αποκέντρωσης Παραδείγματα

Centralized	Twitter, Facebook, Instagram
Federated	Email, XMPP, phone networks, physical mail
Distributed	BitTorrent, IPFS, Scuttlebutt

### 3.3.2 ΧΡΗΜΑΤΟΔΟΤΗΣΗ ΚΑΙ ΚΕΡΔΟΦΟΡΙΑ

Οι ιστότοποι της Mastodon λειτουργούν από διαφορετικά άτομα ή οργανισμούς εντελώς ανεξάρτητα. Το Mastodon δεν εφαρμόζει στρατηγικές κερδοσκοπίας στο λογισμικό. Ορισμένοι φορείς διαχείρισης διακομιστών επιλέγουν να προσφέρουν λογαριασμούς επί πληρωμή, ορισμένοι φορείς διαχείρισης διακομιστών είναι εταιρείες που μπορούν να χρησιμοποιήσουν την υπάρχουσα υποδομή τους, ορισμένοι φορείς διαχείρισης διακομιστών βασίζονται σε crowdfunding από τους χρήστες τους μέσω του Patreon και παρόμοιων υπηρεσιών και ορισμένοι φορείς διαχείρισης διακομιστών πληρώνουν απλώς από την τσέπη τους για έναν προσωπικό διακομιστή για τους ίδιους και ίσως μερικούς φίλους. Έτσι, αν θέλετε να υποστηρίξετε τον διακομιστή που φιλοξενεί τον λογαριασμό σας, ελέγξτε αν προσφέρει έναν τρόπο δωρεάς. Η ανάπτυξη του Mastodon χρηματοδοτείται επίσης από crowdfunding μέσω του Patreon και μέσω του OpenCollective. Δεν εμπλέκεται κανένα επιχειρηματικό κεφάλαιο.[i.22]

### 3.3.3 ΕΛΕΥΘΕΡΟ ΛΟΓΙΣΜΙΚΟ

Στην επίσημη ιστοσελίδα του Mastodon αναφέρεται επίσης πως, “Σε αντίθεση με τις ιδιόκτητες υπηρεσίες, ο καθένας έχει την πλήρη ελευθερία να εκτελεί, να εξετάζει, να επιθεωρεί, να αντιγράφει, να τροποποιεί, να διανέμει και να επαναχρησιμοποιεί τον πηγαίο κώδικα του Mastodon, με την προϋπόθεση ότι εγγυάται τις ίδιες ελευθερίες για κάθε παράγωγο έργο. Όπως ακριβώς οι χρήστες του Mastodon μπορούν να επιλέξουν τον πάροχο υπηρεσιών τους, έτσι και εσείς ως άτομο είστε ελεύθεροι να συνεισφέρετε χαρακτηριστικά στο Mastodon ή να δημοσιεύσετε μια τροποποιημένη έκδοση του Mastodon που περιλαμβάνει διαφορετικά χαρακτηριστικά. Αυτές οι τροποποιημένες εκδόσεις, γνωστές και ως διακλαδώσεις λογισμικού, απαιτείται να διατηρούν τις ίδιες ελευθερίες με το αρχικό έργο Mastodon. Για παράδειγμα, το glitch-soc είναι μια διανομή λογισμικού που προσθέτει διάφορα πειραματικά χαρακτηριστικά. Υπάρχουν επίσης πολλές μεμονωμένες διακλαδώσεις, που ίσως έχουν ελαφρώς διαφορετικό θέμα ή περιλαμβάνουν μικρές τροποποιήσεις στην βάση του κώδικα. Επειδή το Mastodon είναι ελεύθερο λογισμικό που σέβεται την ελευθερία σας, εξατομικεύσεις όπως αυτή όχι μόνο επιτρέπονται αλλά και ενθαρρύνονται.”[i.22]

Η σχετικά νέα αυτή προσέγγιση, αποτελεί τροφή για σκέψη, ειδικά για όσους επιθυμούν η παρουσία τους και η κίνησή τους στο διαδίκτυο να πλαισιώνεται από μία ανεξάρτητη υποδομή, πλήρως διαχειρίσιμη από τους ίδιους, χωρίς να επάγεται μέσα σε ένα κεντρικό σύστημα, το οποίο με ευκολία θα μπορεί να λογοκρίνει ή και να απορρίπτει κάποιον χρήστη. Όπως, αναφέρει χαρακτηριστικά και ο Eugen Rochko, στην ιστοσελίδα του Mastodon, “Η απόλυτη δύναμη έγκειται στο να δίνεται στους ανθρώπους η δυνατότητα να δημιουργούν τους δικούς τους χώρους, τις δικές τους κοινότητες, να τροποποιούν το λογισμικό κατά το δοκούν, χωρίς όμως να θυσιάζεται η δυνατότητα των ανθρώπων από διαφορετικές κοινότητες να αλληλεπιδρούν μεταξύ τους.”[i.22]

### 3.4 CLOUDFLARE ΚΑΙ EDGE COMPUTING

Σε αυτό το κεφάλαιο θα μπορούμε στα λημέρια της Clouflare και πώς αυτή έκανε ένα μεγάλο άλμα στον τομέα του cloud και της δικτύωσης, παρέχοντας αυτό που ονομάζεται Cloudflare Workers. Εισαγωγικά, θα πρέπει να αναφερθεί πως το Cloudflare Workers είναι μία serverless πλατφόρμα που “τρέχει” στο διεθνές δίκτυο της cloudflare και παρέχει υπηρεσίες hosting και ασφάλειας με μία νέα λογική. Το Edge Computing είναι μια κατανεμημένη, ανοικτή αρχιτεκτονική της πληροφορικής που διαθέτει αποκεντρωμένη επεξεργαστική ισχύ, επιτρέποντας το mobile computing και τις τεχνολογίες Internet of Things (IoT). Στο edge computing, τα δεδομένα επεξεργάζονται από την ίδια τη συσκευή ή από έναν τοπικό υπολογιστή ή διακομιστή, αντί να μεταδίδονται σε ένα κέντρο δεδομένων. Ο λόγος που εξετάζεται αυτή η τεχνολογία, είναι γιατί η εφαρμογή που υλοποιήθηκε στα πλαίσια αυτής της εργασίας βασίζεται εν μέρη σε λογική του edge computing, το οποίο αποτελεί μία αποτελεσματική προσέγγιση ώστε να μπορέσει κάποιος να κάνει ανάλυση, συλλογή δεδομένων και αναγνώριση προβλημάτων των υπηρεσιών του σε περιβάλλον σμήνους στο cloud, δίχως η απόσταση και το latency να είναι υπαρκτό πρόβλημα.

Στο πιο βασικό του επίπεδο, το Cloudflare είναι μια κρυφή μνήμη HTTP που λειτουργεί σε 117 τοποθεσίες παγκοσμίως (και αυξάνεται). Το πρότυπο HTTP ορίζει ένα σταθερό σύνολο χαρακτηριστικών για τις κρυφές μνήμες HTTP. Το Cloudflare, φυσικά, κάνει πολύ περισσότερα,

όπως παροχή DNS και SSL, θωράκιση του ιστότοπού σας έναντι επιθέσεων, εξισορρόπηση φορτίου στους διακομιστές προέλευσης και πολλά άλλα. [i.23]

Όμως, όλες αυτές είναι προκαθορισμένες λειτουργίες. Τι γίνεται όταν ένας χρήστης επιθυμεί να υλοποιήσει την δική του προσωπική custom λογική; Σε αυτό το σημείο έρχεται η Cloudflare να κάνει ένα βήμα παρακάτω και να δώσει πλήρη ελευθερία σε κάθε πελάτη να παραμετροποιεί τους διακομιστές της.

### 3.4.1 ΕΛΕΥΘΕΡΙΑ ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗΣ

Όπως αναφέρεται και στην επίσημη ιστοσελίδα του cloudflare, το να προσθέτει η cloudflare επ' αορίστου δυνατότητες, δεν πρόκειται ποτέ να καλύψει κάθε πιθανή ανάγκη. Αντιθέτως αυτό που προτιμά να κάνει είναι να κάνει την άκρη του δικτύου της παραμετροποιήσιμη-προγραμματίσιμη. 117 διακομιστές πλήρως ευέλικτοι από άκρη σε άκρη σε όλο τον κόσμο, διαθέσιμοι να χρησιμοποιηθούν, όπως αρμόζει ο κάθε πελάτης ξεχωριστά.[i.23]

Φυσικά, όταν μία εταιρία αποτελείται από πελάτες σε εκατοντάδες τοποθεσίες, τα παραδοσιακά μέσα φιλοξενίας λογισμικού δεν λειτουργούν σωστά. Έτσι, η Cloudflare μετά από αρκετή μελέτη κατέληξε στη λύση που ήταν η πιο διαδεδομένη γλώσσα στον ιστό σήμερα: η JavaScript. Η JavaScript εκτελείται χρησιμοποιώντας τη V8, τη μηχανή JavaScript που αναπτύχθηκε για το Google Chrome. Αυτό σημαίνει ότι μπορεί να εκτελούνται με ασφάλεια κώδικες από πολλούς πελάτες στους διακομιστές με τον ίδιο τρόπο που το Chrome εκτελεί κώδικες από πολλούς ιστότοπους - χρησιμοποιώντας τεχνολογία που έχει ελεγχθεί σχεδόν μια δεκαετία. [i.23]

Πιο συγκεκριμένα η Javascript που χρησιμοποιεί η Cloudflare, είναι γραμμένη στο Service Worker API. Οι Service Workers είναι ένα χαρακτηριστικό που υλοποιείται από τα σύγχρονα προγράμματα περιήγησης, το οποίο σας επιτρέπει να φορτώσετε ένα πρόγραμμα που αναχαιτίζει τις αιτήσεις ιστού που προορίζονται για τον διακομιστή σας πριν φτάσουν στο δίκτυο, δίνοντάς σας την ευκαιρία να τις ξαναγράψετε, να τις ανακατευθύνετε ή ακόμη και να απαντήσετε απευθείας. [i.23]

Παρακάτω μπορούμε να δούμε μερικά τέτοια παραδείγματα κώδικα που ήταν αναρτημένα και στην επίσημη ιστοσελίδα της Cloudflare. Αυτά γράφονται με βάση το τυπικό API Service

Workers. Η μόνη διαφορά είναι ότι εκτελούνται στην άκρη του Cloudflare και όχι στο πρόγραμμα περιήγησης.[i.23]

Στην Εικόνα 10 είναι ένας εργάτης (“worker”) που παραλείπει την προσωρινή μνήμη για αιτήσεις που έχουν επικεφαλίδα Cookie (π.χ. επειδή ο χρήστης είναι συνδεδεμένος). Φυσικά, ένας πραγματικός ιστότοπος θα είχε πιθανώς πιο περίπλοκες συνθήκες για την προσωρινή αποθήκευση, αλλά εδώ πρόκειται για κώδικα, οπότε μπορείτε να κάνετε οτιδήποτε.[i.23]

```
// A Service Worker which skips cache if the request contains
// a cookie.
addEventListener('fetch', event => {
  let request = event.request
  if (request.headers.has('Cookie')) {
    // Cookie present. Add Cache-Control: no-cache.
    let newHeaders = new Headers(request.headers)
    newHeaders.set('Cache-Control', 'no-cache')
    event.respondWith(fetch(request, {headers: newHeaders}))
  }

  // Use default behavior.
  return
})
```

Εικόνα 10: Παράδειγμα 1 από την σελίδα του Cloudflare

Σε ένα άλλο παράδειγμα, βλέπουμε στην Εικόνα 11 έναν εργάτη (“worker”) που αναζητά στο περιεχόμενο της σελίδας για διευθύνσεις URL που είναι τυλιγμένες σε διπλές αγκύλες, φέρνει αυτές τις διευθύνσεις URL και στη συνέχεια τις αντικαθιστά στη σελίδα. Αυτό υλοποιεί ένα είδος πρωτόγονης μηχανής προτύπων που υποστηρίζει κάτι σαν “Edge Side Includes”. [i.23]

```
addEventListener("fetch", event => {
  event.respondWith(fetchAndReplace(event.request))
})

async function fetchAndReplace(request) {
  // Fetch from origin server.
  let response = await fetch(request)

  // Make sure we only modify text, not images.
  let type = response.headers.get("Content-Type") || ""
  if (!type.startsWith("text/")) {
    // Not text. Don't modify.
    return response
  }

  // Read response body.
  let text = await response.text()

  // Modify it.
  let modified = text.replace(
    /Worker/g, "Minion")

  // Return modified response.
  return new Response(modified, {
    status: response.status,
    statusText: response.statusText,
    headers: response.headers
  })
}
```

Εικόνα 11: Παράδειγμα worker από την σελίδα του Cloudflare

### 3.4.2 ΠΛΕΟΝΕΚΤΗΜΑΤΑ

Στην ουσία του πράγματος κάποιος μπορεί να κάνει σχεδόν τα πάντα με έναν service worker, εφόσον ο κώδικας είναι δικός του για να τον γράψει. Ακολουθούν μερικές μόνο ιδέες για το πώς μπορεί κάποιος να χρησιμοποιήσει τους Service Workers στο Cloudflare σύμφωνα με την Cloudflare:[i.23]

#### Ενίσχυση της επίδοσης

- Χρησιμοποίηση προσαρμοσμένης λογική για να αποφασίσετε ποιες αιτήσεις μπορούν να αποθηκευτούν στην κρυφή μνήμη στην άκρη και να τις κανονικοποιήσετε για να βελτιώσετε το ποσοστό επιτυχίας στην κρυφή μνήμη.



- Επεκτείνετε τα πρότυπα HTML απευθείας στην άκρη, αντλώντας μόνο δυναμικό περιεχόμενο από το διακομιστή σας.
- Απαντήστε σε stateless αιτήματα απευθείας από την άκρη χωρίς να επικοινωνήσετε καθόλου με τον διακομιστή προέλευσης.
- Διαχωρίστε ένα αίτημα σε πολλαπλές παράλληλες αιτήσεις προς διαφορετικούς διακομιστές και, στη συνέχεια, συνδυάστε τις απαντήσεις.

#### **Ενίσχυση της ασφάλειας**

- Εφαρμογή προσαρμοσμένων κανόνων και φίλτρων ασφαλείας.
- Εφαρμογή προσαρμοσμένων μηχανισμών ελέγχου ταυτότητας και εξουσιοδότησης.

#### **Ενίσχυση αξιοπιστίας**

- Αναπτύξτε γρήγορες διορθώσεις στον ιστότοπό σας σε δευτερόλεπτα, χωρίς να χρειάζεται να ενημερώνετε τους δικούς σας διακομιστές.
- Εφαρμογή προσαρμοσμένης λογικής εξισορρόπησης φορτίου και failover.
- Ανταποκριθείτε δυναμικά όταν ο διακομιστής προέλευσης δεν είναι προσβάσιμος.

Όλα αυτά δεν είναι παρά παραδείγματα, εφόσον η όλη λογική του cloudflare workers είναι να κάνει κάποιος πράγματα τα οποία ούτε η ίδια η εταιρία δεν έχει σκεφτεί να παρέχει.

### **3.4.3 JAVASCRIPT**

Το αμέσως επόμενο ερώτημα που ίσως αναδύεται από αυτή την υπηρεσία είναι γιατί την JavaScript, και τι είναι αυτό που προσφέρει, ώστε να την καθιστά την καλύτερη δυνατή επιλογή. Σύμφωνα με την Clouflare τα Cloudflare Workers είναι γραμμένα σε JavaScript, τα οποία εκτελούνται χρησιμοποιώντας τη μηχανή V8 JavaScript (από το Google Chrome). Επιλέχθηκε η JavaScript και η V8 για δύο βασικούς λόγους:[i.23]

- **Ασφάλεια:** Η μηχανή V8 JavaScript είναι αναμφισβήτητα το πιο εξεταζόμενο sandbox κώδικα στην ιστορία της πληροφορικής και η ομάδα ασφαλείας του Chrome είναι μία από τις καλύτερες στον κόσμο. Επιπλέον, η Google πληρώνει μαζικές αμοιβές για σφάλματα σε όποιον μπορεί να βρει μια ευπάθεια. (Τούτου λεχθέντος, η ομάδα του Cloudflare έχει προσθέσει πρόσθετα στρώματα του δικού τους sandboxing πάνω στο V8).

- **Πανταχού παρούσα:** Η JavaScript είναι παντού. Όποιος κατασκευάζει μια εφαρμογή ιστού πρέπει ήδη να τη γνωρίζει: ενώ ο διακομιστής του μπορεί να είναι γραμμένος σε διάφορες γλώσσες, ο πελάτης πρέπει να είναι σε JavaScript, επειδή αυτό εκτελούν οι φυλλομετρητές.

Τελικά, έγινε σαφές ότι το V8 ήταν η καλύτερη επιλογή. Το τελευταίο καρφί στο φέρετρο ήταν η συνειδητοποίηση ότι το V8 περιλαμβάνει το WebAssembly out-of-the-box, πράγμα που σημαίνει ότι όσοι χρειάζονται πραγματικά να αναπτύξουν κώδικα γραμμένο σε άλλες γλώσσες μπορούν να το κάνουν.[i.23]

### 3.4.4 TO EDGE COMPUTING

Όπως προ-αναφέρθηκε στην αρχή του τρίτου κεφαλαίου το Edge Computing είναι μια κατανεμημένη αρχιτεκτονική τεχνολογίας πληροφοριών ( IT) στην οποία τα δεδομένα του πελάτη υποβάλλονται σε επεξεργασία στην περιφέρεια του δικτύου, όσο το δυνατόν πιο κοντά στην πηγή προέλευσης. Τα δεδομένα αποτελούν την πηγή ζωής των σύγχρονων επιχειρήσεων, παρέχοντας πολύτιμες επιχειρηματικές πληροφορίες και υποστηρίζοντας τον έλεγχο σε πραγματικό χρόνο των κρίσιμων επιχειρηματικών διαδικασιών και λειτουργιών. Οι σημερινές επιχειρήσεις κατακλύζονται από έναν ωκεανό δεδομένων και τεράστιες ποσότητες δεδομένων μπορούν να συλλέγονται τακτικά από αισθητήρες και συσκευές IoT που λειτουργούν σε πραγματικό χρόνο από απομακρυσμένες τοποθεσίες και αφιλόξενα λειτουργικά περιβάλλοντα σχεδόν οπουδήποτε στον κόσμο. [i.24]

Αλλά αυτή η ψηφιακή πλημμύρα δεδομένων αλλάζει επίσης τον τρόπο με τον οποίο οι επιχειρήσεις χειρίζονται την υπολογιστική. Το παραδοσιακό παράδειγμα της υπολογιστικής που βασίζεται σε ένα συγκεντρωτικό κέντρο δεδομένων και το καθημερινό διαδίκτυο δεν είναι κατάλληλο για τη διακίνηση ατελείωτα αυξανόμενων ροών δεδομένων του πραγματικού κόσμου. Οι περιορισμοί του εύρους ζώνης, τα προβλήματα καθυστέρησης και οι απρόβλεπτες διακοπές του δικτύου μπορούν να συμπράξουν για να εμποδίσουν τέτοιες προσπάθειες. Οι επιχειρήσεις ανταποκρίνονται σε αυτές τις προκλήσεις των δεδομένων μέσω της χρήσης της αρχιτεκτονικής του edge computing.[i.24]

Με απλούστερους όρους, το edge computing μετακινεί ένα μέρος των πόρων αποθήκευσης και υπολογισμού εκτός του κεντρικού κέντρου δεδομένων και πιο κοντά στην ίδια την πηγή των δεδομένων. Αντί να μεταδίδονται ακατέργαστα δεδομένα σε ένα συγκεντρωτικό κέντρο δεδομένων για επεξεργασία και ανάλυση, η εργασία αυτή εκτελείται εκεί όπου πραγματικά παράγονται τα δεδομένα - είτε πρόκειται για ένα κατάστημα λιανικής πώλησης, είτε για ένα εργοστάσιο, είτε για μια εκτεταμένη υπηρεσία κοινής ωφέλειας είτε για μια έξυπνη πόλη. Μόνο το αποτέλεσμα αυτής της υπολογιστικής εργασίας στην άκρη, όπως επιχειρηματικές γνώσεις σε πραγματικό χρόνο, προβλέψεις συντήρησης εξοπλισμού ή άλλες απαντήσεις που μπορούν να ληφθούν υπόψη, αποστέλλεται πίσω στο κεντρικό κέντρο δεδομένων για επανεξέταση και άλλες ανθρώπινες αλληλεπιδράσεις.[i.24]

Παρακάτω στην Εικόνα 12 βλέπουμε μία εικονική παρουσίαση ενός σεναρίου του Edge Computing. Το Edge Computing τοποθετεί τον αποθηκευτικό χώρο και τους διακομιστές εκεί όπου βρίσκονται τα δεδομένα, απαιτώντας συχνά μόνο ένα μέρος του εξοπλισμού που λειτουργεί στο απομακρυσμένο τοπικό δίκτυο για τη συλλογή και επεξεργασία των δεδομένων τοπικά. Σε πολλές περιπτώσεις, ο υπολογιστικός εξοπλισμός αναπτύσσεται σε θωρακισμένα ή ενισχυμένα περιβλήματα για την προστασία του εξοπλισμού από ακραίες θερμοκρασίες, υγρασία και άλλες περιβαλλοντικές συνθήκες. Η επεξεργασία συχνά περιλαμβάνει την εξομάλυνση και την ανάλυση της ροής δεδομένων για την αναζήτηση επιχειρηματικών πληροφοριών και μόνο τα αποτελέσματα της ανάλυσης αποστέλλονται πίσω στο κύριο κέντρο δεδομένων.[i.24]

Η υπολογιστική άκρων συνδέεται στενά με τις έννοιες της υπολογιστικής νέφους και της υπολογιστικής ομίχλης ("fog computing"). Αν και υπάρχει κάποια επικάλυψη μεταξύ αυτών των εννοιών, δεν είναι το ίδιο πράγμα και γενικά δεν θα πρέπει να χρησιμοποιούνται εναλλακτικά. Είναι χρήσιμο να συγκρίνετε τις έννοιες και να κατανοήσετε τις διαφορές τους. Ένας από τους ευκολότερους τρόπους για να κατανοήσουμε τις διαφορές μεταξύ του edge, του cloud και του fog computing είναι να αναδείξουμε το κοινό τους θέμα: Και οι τρεις έννοιες σχετίζονται με την κατανεμημένη υπολογιστική και εστιάζουν στη φυσική ανάπτυξη των πόρων υπολογισμού και αποθήκευσης σε σχέση με τα δεδομένα που παράγονται. Η διαφορά έγκειται στο πού βρίσκονται αυτοί οι πόροι.[i.24]

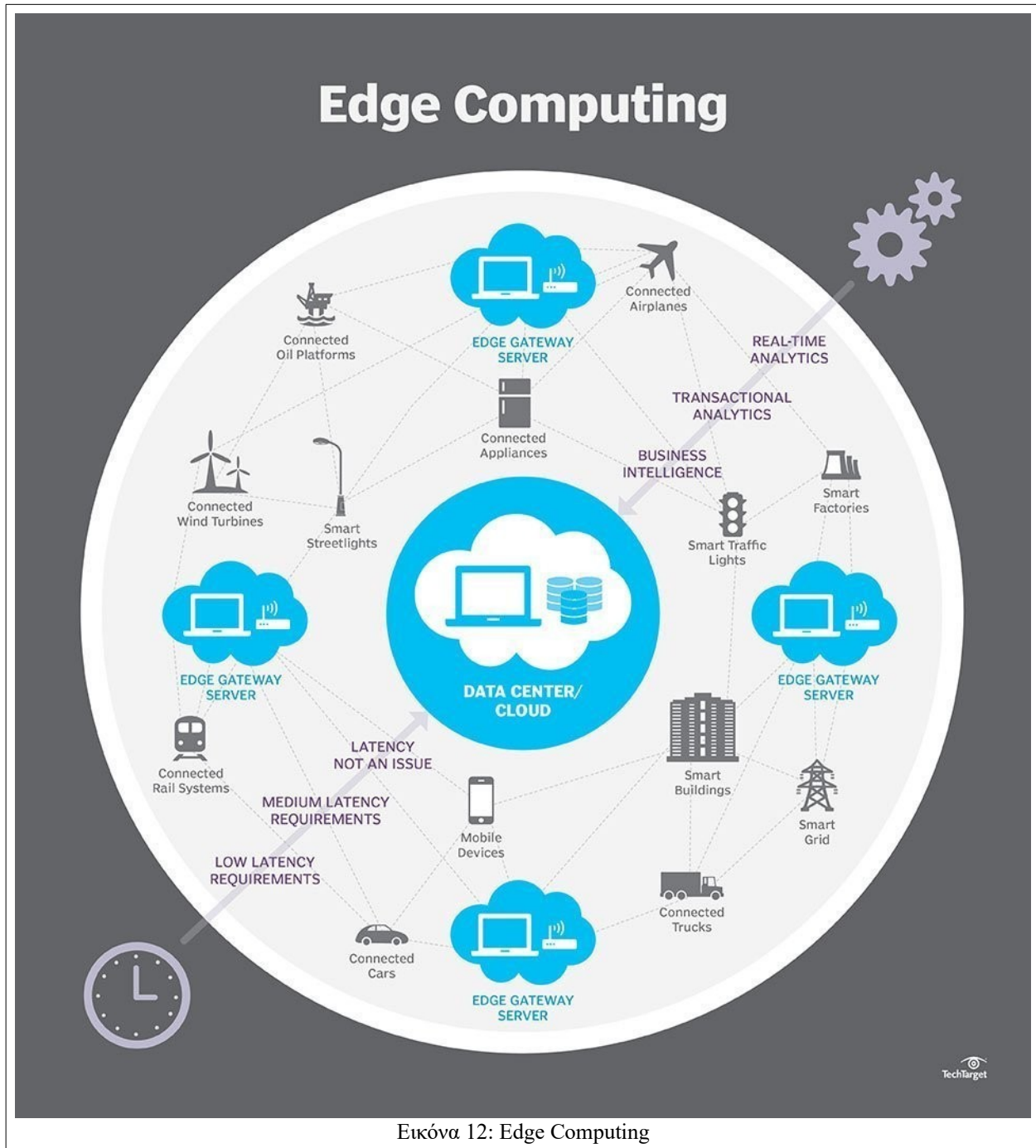
- **Άκρη.** Το Edge computing είναι η ανάπτυξη υπολογιστικών και αποθηκευτικών πόρων στη θέση όπου παράγονται τα δεδομένα. Αυτό ιδανικά τοποθετεί τον υπολογισμό και την

αποθήκευση στο ίδιο σημείο με την πηγή δεδομένων στην άκρη του δικτύου. Για παράδειγμα, ένα μικρό περίβλημα με αρκετούς διακομιστές και κάποια αποθηκευτικά μέσα μπορεί να εγκατασταθεί στην κορυφή μιας ανεμογεννήτριας για να συλλέγει και να επεξεργάζεται δεδομένα που παράγονται από αισθητήρες μέσα στην ίδια την ανεμογεννήτρια. Ως άλλο παράδειγμα, ένας σιδηροδρομικός σταθμός θα μπορούσε να τοποθετήσει μια μικρή ποσότητα υπολογιστών και αποθηκευτικού χώρου εντός του σταθμού για τη συλλογή και την επεξεργασία μυριάδων δεδομένων από αισθητήρες τροχιάς και σιδηροδρομικής κυκλοφορίας. Τα αποτελέσματα οποιασδήποτε τέτοιας επεξεργασίας μπορούν στη συνέχεια να αποστέλλονται πίσω σε ένα άλλο κέντρο δεδομένων για ανθρώπινη εξέταση, αρχειοθέτηση και συγχώνευση με άλλα αποτελέσματα δεδομένων για ευρύτερη ανάλυση.[i.24]

- **Νέφος.** Το υπολογιστικό νέφος είναι μια τεράστια, εξαιρετικά κλιμακούμενη ανάπτυξη υπολογιστικών και αποθηκευτικών πόρων σε μία από πολλές κατανομημένες παγκόσμιες τοποθεσίες (περιοχές). Οι πάροχοι υπολογιστικού νέφους ενσωματώνουν επίσης μια σειρά από προσυσκευασμένες υπηρεσίες για λειτουργίες IoT, καθιστώντας το νέφος μια προτιμώμενη κεντρική πλατφόρμα για αναπτύξεις IoT. Αλλά ακόμη και αν το υπολογιστικό νέφος προσφέρει πολύ περισσότερους από αρκετούς πόρους και υπηρεσίες για την αντιμετώπιση πολύπλοκων αναλύσεων, η πλησιέστερη περιφερειακή εγκατάσταση υπολογιστικού νέφους μπορεί ακόμη να απέχει εκατοντάδες χιλιόμετρα από το σημείο όπου συλλέγονται τα δεδομένα, και οι συνδέσεις βασίζονται στην ίδια ταραχώδη συνδεσιμότητα στο διαδίκτυο που υποστηρίζει τα παραδοσιακά κέντρα δεδομένων. Στην πράξη, το υπολογιστικό νέφος είναι μια εναλλακτική λύση - ή μερικές φορές ένα συμπλήρωμα - των παραδοσιακών κέντρων δεδομένων. Το νέφος μπορεί να φέρει τον κεντρικό υπολογισμό πολύ πιο κοντά σε μια πηγή δεδομένων, αλλά όχι στην άκρη του δικτύου.[i.24]
- **Fog (ομίχλη).** Ωστόσο, η επιλογή της ανάπτυξης υπολογιστών και αποθήκευσης δεν περιορίζεται στο νέφος ή στην άκρη. Ένα κέντρο δεδομένων νέφους μπορεί να είναι πολύ μακριά, αλλά η ανάπτυξη στην άκρη μπορεί απλώς να είναι πολύ περιορισμένη σε πόρους, ή φυσικά διάσπαρτη ή κατανομημένη, ώστε να είναι δυνατή η αυστηρή υπολογιστική στην άκρη. Σε αυτή την περίπτωση, η έννοια του fog computing μπορεί να βοηθήσει. Η υπολογιστική ομίχλης συνήθως κάνει ένα βήμα πίσω και τοποθετεί υπολογιστι-

κούς και αποθηκευτικούς πόρους "μέσα" στα δεδομένα, αλλά όχι απαραίτητα "στα" δεδομένα. Τα περιβάλλοντα υπολογισμού ομίχλης μπορούν να παράγουν ασύλληπτες ποσότητες δεδομένων αισθητήρων ή δεδομένων IoT που παράγονται σε εκτεταμένες φυσικές περιοχές, οι οποίες είναι πολύ μεγάλες για να οριστεί μια άκρη. Παραδείγματα περιλαμβάνουν έξυπνα κτίρια, έξυπνες πόλεις ή ακόμη και έξυπνα δίκτυα κοινής ωφέλειας. Σκεφτείτε μια έξυπνη πόλη, όπου τα δεδομένα μπορούν να χρησιμοποιηθούν για την παρακολούθηση, την ανάλυση και τη βελτιστοποίηση του συστήματος δημόσιων μεταφορών, των δημοτικών υπηρεσιών κοινής ωφέλειας, των υπηρεσιών της πόλης και την καθοδήγηση του μακροπρόθεσμου αστικού σχεδιασμού. Μια ενιαία ανάπτυξη άκρου απλά δεν είναι αρκετή για να χειριστεί ένα τέτοιο φορτίο, οπότε η υπολογιστική ομίχλης μπορεί να λειτουργήσει μια σειρά από αναπτύξεις κόμβων ομίχλης εντός του πεδίου εφαρμογής του περιβάλλοντος για τη συλλογή, την επεξεργασία και την ανάλυση δεδομένων. [i.24]

Σημείωση: Είναι σημαντικό να επαναληφθεί ότι το fog computing και το edge computing μοιράζονται έναν σχεδόν πανομοιότυπο ορισμό και αρχιτεκτονική και οι όροι χρησιμοποιούνται μερικές φορές εναλλακτικά ακόμη και μεταξύ των ειδικών της τεχνολογίας.[i.24]



Εικόνα 12: Edge Computing

### 3.5 ΣΜΗΝΟΣ ΣΤΟ ΝΕΦΟΣ

Τα clusters υπολογιστών έχουν γνωρίσει σημαντική αύξηση στην υιοθέτηση τους την τελευταία δεκαετία. Τόσο οι νεοσύστατες επιχειρήσεις όσο και οι τεχνολογικοί κολοσσοί αξιοποιούν αρχιτεκτονικές βασισμένες σε cluster για την ανάπτυξη και τη διαχείριση των εφαρμογών τους στο cloud. Σε γενικό επίπεδο, ένα cluster υπολογιστών είναι μια ομάδα δύο ή περισσότερων υπολογιστών, ή κόμβων, που λειτουργούν παράλληλα για την επίτευξη ενός κοινού στόχου. Αυτό επιτρέπει την κατανομή μεταξύ των κόμβων του cluster των φόρτων εργασίας που αποτελούνται από μεγάλο αριθμό μεμονωμένων, παραλληλοποιήσιμων εργασιών. Ως αποτέλεσμα, οι εργασίες αυτές μπορούν να αξιοποιήσουν τη συνδυασμένη μνήμη και την επεξεργαστική ισχύ κάθε υπολογιστή για να αυξήσουν τη συνολική απόδοση.[i.26]

Για να δημιουργηθεί ένα cluster υπολογιστών, οι επιμέρους κόμβοι πρέπει να συνδεθούν σε ένα δίκτυο ώστε να είναι δυνατή η επικοινωνία μεταξύ των κόμβων. Στη συνέχεια, μπορεί να χρησιμοποιηθεί λογισμικό cluster υπολογιστών για να ενώσει τους κόμβους μεταξύ τους και να σχηματίσει ένα cluster. Μπορεί να διαθέτει μια κοινή συσκευή αποθήκευσης ή/και τοπικό αποθηκευτικό χώρο σε κάθε κόμβο. Συνήθως, τουλάχιστον ένας κόμβος ορίζεται ως κόμβος-ηγέτης και λειτουργεί ως σημείο εισόδου στο cluster. Ο αρχηγικός κόμβος μπορεί να είναι υπεύθυνος για την ανάθεση εισερχόμενων εργασιών στους άλλους κόμβους και, εάν είναι απαραίτητο, για τη συγκέντρωση των αποτελεσμάτων και την επιστροφή μιας απάντησης στον χρήστη.[i.26]

Ιδανικά, ένα cluster λειτουργεί σαν να επρόκειτο για ένα ενιαίο σύστημα. Ένας χρήστης που έχει πρόσβαση στο cluster δεν θα πρέπει να χρειάζεται να γνωρίζει αν το σύστημα είναι ένα cluster ή ένα μεμονωμένο μηχάνημα. Επιπλέον, ένα cluster θα πρέπει να είναι σχεδιασμένο έτσι ώστε να ελαχιστοποιεί την καθυστέρηση και να αποτρέπει τα σημεία συμφόρησης στην επικοινωνία μεταξύ των κόμβων.[i.26]

Τα clusters υπολογιστών μπορούν γενικά να κατηγοριοποιηθούν σε τρεις τύπους:

1. Υψηλά διαθέσιμοι ή fail-over
2. Εξισορρόπηση φορτίου
3. Υπολογισμός υψηλών επιδόσεων

### 3.5.1 ΠΛΕΟΝΕΚΤΗΜΑΤΑ

Η υπολογιστική σε cluster παρέχει μια σειρά από πλεονεκτήματα: υψηλή διαθεσιμότητα μέσω ανοχής σφαλμάτων και ανθεκτικότητας, εξισορρόπηση φορτίου και δυνατότητες κλιμάκωσης, καθώς και βελτίωση των επιδόσεων. Ας επεκταθούμε σε καθένα από αυτά τα χαρακτηριστικά και ας εξετάσουμε πώς τα κάνουν εφικτά τα clusters. [i.26]

#### 3.5.1.1 Υψηλή διαθεσιμότητα

Υπάρχουν μερικοί σημαντικοί όροι που πρέπει να θυμάστε όταν συζητάτε για την αξιοπιστία ενός συστήματος:[i.26]

- Διαθεσιμότητα - η προσβασιμότητα ενός συστήματος ή μιας υπηρεσίας κατά τη διάρκεια μιας χρονικής περιόδου, συνήθως εκφραζόμενη ως ποσοστό διαθεσιμότητας κατά τη διάρκεια ενός συγκεκριμένου έτους (π.χ. 99,999% διαθεσιμότητα).
- Ανθεκτικότητα - πόσο καλά ένα σύστημα ανακάμπτει από αποτυχία.
- Ανοχή σφαλμάτων - η ικανότητα ενός συστήματος να συνεχίσει να παρέχει μια υπηρεσία σε περίπτωση βλάβης
- Αξιοπιστία - η πιθανότητα ένα σύστημα να λειτουργεί όπως αναμένεται
- Πλεονασμός - επανάληψη κρίσιμων πόρων για τη βελτίωση της αξιοπιστίας του συστήματος

Μια εφαρμογή που εκτελείται σε ένα μόνο μηχάνημα έχει ένα μόνο σημείο αποτυχίας, γεγονός που καθιστά το σύστημα ανεπαρκές ως προς την αξιοπιστία του. Εάν το μηχάνημα που φιλοξενεί την εφαρμογή πέσει, θα υπάρξει σχεδόν πάντα διακοπή λειτουργίας, ενώ η υποδομή θα ανακάμπτει. Η διατήρηση ενός βαθμού εφεδρείας, ο οποίος συμβάλλει στη βελτίωση της αξιοπιστίας, μπορεί να μειώσει το χρονικό διάστημα που μια εφαρμογή δεν είναι διαθέσιμη. Αυτό μπορεί να επιτευχθεί με την προληπτική εκτέλεση της εφαρμογής σε ένα δεύτερο σύστημα (το οποίο μπορεί να δέχεται ή να μην δέχεται κίνηση) ή με την προπαραμετροποίηση ενός cold system (δηλαδή ενός συστήματος που δεν λειτουργεί επί του παρόντος) με την εφαρμογή. Αυτές οι διαμορφώσεις είναι αντίστοιχα γνωστές ως διαμορφώσεις active-active και active-passive. Όταν ανιχνεύεται μια αποτυχία, ένα ενεργό-ενεργό σύστημα μπορεί να κάνει αμέσως failover στο δεύ-



τερο μηχάνημα, ενώ ένα ενεργό-παθητικό σύστημα θα κάνει failover μόλις το δεύτερο μηχάνημα είναι σε λειτουργία.[i.26]

Τα clusters υπολογιστών αποτελούνται από περισσότερους από έναν κόμβους που εκτελούν ταυτόχρονα την ίδια διεργασία και, ως εκ τούτου, είναι ενεργά-ενεργά συστήματα. Τα ενεργά-ενεργά συστήματα είναι συνήθως ανθεκτικά σε σφάλματα, επειδή το σύστημα είναι εγγενώς σχεδιασμένο να διαχειρίζεται την απώλεια ενός κόμβου. Εάν ένας κόμβος αποτύχει, οι υπόλοιποι κόμβοι είναι έτοιμοι να αναλάβουν το φόρτο εργασίας του κόμβου που απέτυχε. Με αυτό το δεδομένο, ένα cluster που απαιτεί έναν κόμβο-ηγέτη θα πρέπει να εκτελεί τουλάχιστον δύο κόμβους-ηγέτες σε διάταξη active-active. Αυτό μπορεί να αποτρέψει τη μη διαθεσιμότητα του σμήνους σε περίπτωση αποτυχίας ενός κόμβου ηγέτη.[i.26]

Εκτός του ότι είναι πιο ανθεκτικά σε σφάλματα, τα clusters μπορούν να βελτιώσουν την ανθεκτικότητα, καθιστώντας εύκολο για τους ανακτημένους κόμβους να επανενταχθούν στο σύστημα και να επαναφέρουν το cluster στο βέλτιστο μέγεθός του. Οποιαδήποτε ποσότητα διακοπής λειτουργίας του συστήματος είναι δαπανηρή για έναν οργανισμό και μπορεί να δημιουργήσει κακή εμπειρία στους πελάτες, επομένως είναι κρίσιμο ένα σύστημα να είναι ανθεκτικό και ανθεκτικό σε σφάλματα σε περίπτωση βλάβης. Η χρήση cluster μπορεί να βελτιώσει την ανθεκτικότητα και την ανοχή σε σφάλματα του συστήματος, επιτρέποντας υψηλότερη διαθεσιμότητα. Τα clusters με αυτά τα χαρακτηριστικά ονομάζονται "υψηλής διαθεσιμότητας" ή "fail-over" clusters.[i.26]

### 3.5.1.2 Εξισορρόπηση φορτίου

Η εξισορρόπηση φορτίου είναι η πράξη της κατανομής της κίνησης στους κόμβους του σμήνους για τη βελτιστοποίηση των επιδόσεων και την αποφυγή δυσανάλογου φόρτου εργασίας σε οποιονδήποτε κόμβο. Ένας εξισορροπητής φορτίου μπορεί να εγκατασταθεί στον/τους κόμβους-ηγέτες ή να παρέχεται ξεχωριστά από τη συστάδα. Εκτελώντας περιοδικούς ελέγχους υγείας σε κάθε κόμβο της συστάδας, ο εξισορροπιστής φορτίου είναι σε θέση να ανιχνεύσει εάν ένας κόμβος έχει αποτύχει και, σε αυτή την περίπτωση, θα δρομολογήσει την εισερχόμενη κυκλοφορία στους άλλους κόμβους της συστάδας.[i.26]

Παρόλο που μια συστάδα υπολογιστών δεν εξισορροπεί εγγενώς το φορτίο, επιτρέπει την εκτέλεση εξισορρόπησης φορτίου στους κόμβους της. Αυτή η διαμόρφωση αναφέρεται ως συ-

στάδα "εξισορρόπησης φορτίου" και συχνά είναι ταυτόχρονα μια συστάδα υψηλής διαθεσιμότητας.[i.26]

### 3.5.1.3 Κλιμάκωση

Υπάρχουν δύο κατηγορίες κλιμάκωσης: η κάθετη και η οριζόντια. Η κάθετη κλιμάκωση (που αναφέρεται επίσης ως κλιμάκωση προς τα πάνω/κάτω) περιλαμβάνει την αύξηση ή τη μείωση των πόρων που διατίθενται σε μια διεργασία, όπως η ποσότητα μνήμης, ο αριθμός των πυρήνων του επεξεργαστή ή ο διαθέσιμος αποθηκευτικός χώρος. Από την άλλη πλευρά, η οριζόντια κλιμάκωση (κλιμάκωση έξω/έξω) είναι όταν εκτελούνται στο σύστημα πρόσθετες, παράλληλες εργασίες.[i.26]

Κατά τη συντήρηση μιας συστάδας, είναι σημαντικό να παρακολουθείτε τη χρήση των πόρων και την κλιμάκωση, ώστε να διασφαλίζετε ότι οι πόροι της συστάδας χρησιμοποιούνται κατάλληλα. Ευτυχώς, η ίδια η φύση μιας συστάδας καθιστά την οριζόντια κλιμάκωση απλή υπόθεση - ο διαχειριστής πρέπει απλώς να προσθέσει ή να αφαιρέσει κόμβους ανάλογα με τις ανάγκες, έχοντας κατά νου το ελάχιστο επίπεδο πλεονασμού για να διασφαλίσει ότι η συστάδα παραμένει εξαιρετικά διαθέσιμη.[i.26]

### 3.5.1.4 Απόδοση

Όσον αφορά τον παραλληλισμό, οι συστάδες μπορούν να επιτύχουν υψηλότερα επίπεδα επιδόσεων από ό,τι μια μεμονωμένη μηχανή. Αυτό οφείλεται στο γεγονός ότι δεν περιορίζονται από έναν συγκεκριμένο αριθμό πυρήνων επεξεργαστών ή άλλου υλικού. Επιπλέον, η οριζόντια κλιμάκωση μπορεί να μεγιστοποιήσει τις επιδόσεις αποτρέποντας το σύστημα από το να εξαντλήσει τους πόρους. Οι συστάδες υπολογιστών "υψηλών επιδόσεων" (HPC) αξιοποιούν την παραλληλοποιησιμότητα των συστάδων υπολογιστών για να επιτύχουν το υψηλότερο δυνατό επίπεδο επιδόσεων. Ένας υπερυπολογιστής είναι ένα κοινό παράδειγμα συστάδας HPC.[i.26]

### 3.5.2 ΠΡΟΚΛΗΣΕΙΣ ΣΜΗΝΟΥΣ

Η πιο προφανής πρόκληση που παρουσιάζει η ομαδοποίηση είναι η αυξημένη πολυπλοκότητα της εγκατάστασης και της συντήρησης. Ένα λειτουργικό σύστημα, η εφαρμογή και οι εξαρτήσεις της πρέπει να εγκαθίστανται και να ενημερώνονται σε κάθε κόμβο. Αυτό γίνεται ακόμη πιο περίπλοκο αν οι κόμβοι της συστάδας δεν είναι ομοιογενείς. Η χρήση των πόρων για κάθε κόμβο πρέπει επίσης να παρακολουθείται στενά και τα αρχεία καταγραφής ("logs") πρέπει να συγκεντρώνονται για να διασφαλιστεί ότι το λογισμικό συμπεριφέρεται σωστά. Επιπλέον, η διαχείριση της αποθήκευσης γίνεται πιο δύσκολη- μια κοινή συσκευή αποθήκευσης πρέπει να εμποδίζει τους κόμβους να αντικαθιστούν ο ένας τον άλλον και οι κατανεμημένες αποθήκες δεδομένων πρέπει να διατηρούνται συγχρονισμένες.[i.26]

Αυτή η πρόκληση απαντάται με την εφαρμογή που υλοποιήθηκε στα πλαίσια αυτής της εργασίας. Μία εφαρμογή που παρακολουθεί στενά τα αρχεία καταγραφής ("logs") ενός οποιουδήποτε συστήματος φτιαγμένο με docker υποδομή, είτε είναι σε μορφή σμήνους είτε όχι, που αναλύει τα δεδομένα σε μία ενοποιημένη διεπαφή, ώστε ένας διαχειριστής αρχείων καταγραφής να μπορεί να παρακολουθήσει το σύστημά του.

### 3.5.3 ΠΕΡΙΕΚΤΕΣ ΚΑΙ ΣΜΗΝΟΣ

Τα Containers έχουν εξαλείψει πολλά από τα βάρη της ανάπτυξης εφαρμογών. Οι διαφορές μεταξύ τοπικών και απομακρυσμένων περιβαλλόντων μπορούν σε μεγάλο βαθμό να αγνοηθούν (με ορισμένες εξαιρέσεις, όπως η αρχιτεκτονική της CPU), οι εξαρτήσεις των εφαρμογών αποστέλλονται εντός του περιέκτη ("container") και η ασφάλεια βελτιώνεται με την απομόνωση της εφαρμογής από τον κεντρικό υπολογιστή. Η χρήση των containers έχει επίσης διευκολύνει τις ομάδες να αξιοποιήσουν μια αρχιτεκτονική μικρο-υπηρεσιών, όπου η εφαρμογή αναλύεται σε μικρές, χαλαρά συνδεδεμένες υπηρεσίες. Αλλά τι σχέση έχουν τα κοντέινερ με τις συστάδες υπολογιστών;[i.26]

Ακολουθεί ένα συνηθισμένο σενάριο. Ο οργανισμός σας αναπτύσσει μια απλή διαδικτυακή εφαρμογή. Το front-end και το back-end έχουν κατασκευαστεί ως μικρουπηρεσίες, που εκτελούν-

νται ανεξάρτητα η μία από την άλλη ως αυτόνομα containers και επικοινωνούν μέσω HTTPS. Τώρα ήρθε η ώρα να αναπτύξετε την εφαρμογή σας.[i.26]

Η πρώτη λύση που δοκιμάζετε μπορεί να είναι η παροχή μιας εικονικής μηχανής στο cloud για την εκτέλεση των containers σας. Αυτό λειτουργεί, αλλά υπάρχουν ορισμένα μειονεκτήματα. Η απόδοση περιορίζεται από τους πόρους που παρέχονται στο VM και η κλιμάκωση της εφαρμογής είναι πιθανό να είναι δύσκολη. Επιπλέον, εάν το VM ή το υλικό που φιλοξενεί το VM αποτύχει, η εφαρμογή δεν θα είναι διαθέσιμη μέχρι να γίνει παροχή νέου μηχανήματος ή να δρομολογηθεί η κυκλοφορία σε έναν διακομιστή αποτυχίας. Φυσικά, μια συστάδα επιλύει και τα δύο αυτά ζητήματα.[i.26]

Η ανάπτυξη εφαρμογών με container στους κόμβους ενός cluster μπορεί να βελτιώσει σημαντικά τη διαθεσιμότητα, την επεκτασιμότητα και την απόδοση της διαδικτυακής σας εφαρμογής. Η εκτέλεση πολλαπλών κοντέινερ ανά κόμβο αυξάνει τη χρήση των πόρων, ενώ η διασφάλιση της εκτέλεσης μιας περίπτωσης κάθε κοντέινερ σε περισσότερους από έναν κόμβους κάθε φορά αποτρέπει την εφαρμογή σας από το να έχει ένα μοναδικό σημείο αποτυχίας. Βέβαια, αυτό οδηγεί στο πρόβλημα της διαχείρισης των περιεκτών, που επιλύεται μέσω ειδικών λογισμικών διαχείρισης κοντέινερ όπως το Docker.[i.26]

### 3.6 ΣΥΛΛΟΓΗ ΚΑΙ ΠΑΡΑΚΟΛΟΥΘΗΣΗ ΔΕΔΟΜΕΝΩΝ

Αρχικά πρέπει να διευκρινισθεί τι εννοούμε όταν λέμε δεδομένα σε ένα τέτοιο σύστημα και ποια η ανάγκη για την συλλογή και την παρακολούθηση τους. Όπως αναφέρθηκε και στην περίπτωση του σμήνους, η ανάγκη για συνεχή παρακολούθηση της λειτουργικότητας όλης της υποδομής ενός τέτοιου συστήματος αποτελεί τον ακρογωνιαίο λίθο της υγιούς λειτουργικότητάς του. Έτσι κάθε μικρή υπηρεσία σε ένα πληροφοριακό σύστημα παράγει αυτό που λέμε “logs” ή αλλιώς δεδομένα καταγραφής της λειτουργικότητας μίας υπηρεσίας. Η διαχείριση αυτών των αρχείων καταγραφής αποτελεί μία από τις σημαντικότερες ευθύνες και παροχές εν σήμερον ημέρα στον κόσμο του διαδικτύου. Η πληθώρα των logs κάθε μικρο-υπηρεσίας σε συνδυασμό με την πρακτική δυσκολία να ελέγχετε η καθεμία ξεχωριστά για πιθανόν προβλήματα, έφερε στο προ-

σκήνιο την ανάγκη για εργαλεία-εφαρμογές διαχείρισης, συλλογής, ανάλυσης και παρακολούθησης αυτών των δεδομένων συγκεντρωτικά και σε πραγματικό χρόνο. [i.26]

Η διαχείριση αρχείων καταγραφής πιο συγκεκριμένα είναι οι συγκεντρωτικές διαδικασίες και πολιτικές που χρησιμοποιούνται για τη διαχείριση και τη διευκόλυνση της δημιουργίας, της μετάδοσης, της ανάλυσης, της αποθήκευσης, της αρχειοθέτησης και της τελικής διάθεσης των μεγάλων όγκων δεδομένων καταγραφής που δημιουργούνται σε ένα σύστημα πληροφορικής. Ένα αρχείο καταγραφής, σε ένα υπολογιστικό πλαίσιο, είναι η αυτόματα παραγόμενη και χρονοσφραγισμένη τεκμηρίωση των γεγονότων που σχετίζονται με ένα συγκεκριμένο σύστημα. Σχεδόν όλες οι εφαρμογές και τα συστήματα λογισμικού παράγουν αρχεία καταγραφής. Η αποτελεσματική διαχείριση των αρχείων καταγραφής είναι απαραίτητη τόσο για την ασφάλεια όσο και για τη συμμόρφωση. Η παρακολούθηση, η τεκμηρίωση και η ανάλυση των συμβάντων του συστήματος είναι ένα κρίσιμο στοιχείο του security intelligence (SI). Το λογισμικό διαχείρισης αρχείων καταγραφής αυτοματοποιεί πολλές από τις σχετικές διαδικασίες. Ένας διαχειριστής αρχείων καταγραφής συμβάντων (ELM), για παράδειγμα, παρακολουθεί τις αλλαγές στην πληροφοριακή υποδομή ενός οργανισμού.[i.27][i.26]

### 3.6.1 ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΠΑΡΑΚΟΛΟΥΘΗΣΗΣ

Μία τέτοια εφαρμογή παρακολούθησης μπορεί να αποτελείται από μυριάδες λειτουργιών και δυνατοτήτων. Βέβαια, υπάρχουν κάποια πλεονεκτήματα, που δίνουν μία ξεκάθαρη εικόνα της σημαντικότητας μίας τέτοιας υπηρεσίας όπως:[i.29]

- **Αναζήτηση στα logs:** Μία επαγγελματική εφαρμογή αυτής της φύσης θα μπορούσε να δίνει την δυνατότητα στον τελικό χρήστη/διαχειριστή να αναζητήσει ανάμεσα σε εκατοντάδες, χιλιάδες ή και εκατομμύρια logs, ένα ή πολλά συγκεκριμένα logs που τον ενδιαφέρουν, να τα ταξινομήσει κατάλληλα με κατάλληλα queries, να δει κάποια σχετικά ιστογράμματα ή γραφήματα και να εξάγει συμπεράσματα.[i.29]
- **Συναγερμός:** Επιπλέον, θα μπορούσε να παρέχετε μία λειτουργία συναγερμού, που για παράδειγμα, θα ειδοποιεί τον διαχειριστή των αρχείων καταγραφής (logs) για κάποιο συ-

γκεκριμένο κρίσιμο συμβάν στο σύνολο του πληροφοριακού συστήματος του χρήστη. [i.29]

- **Ανίχνευση σφαλμάτων:** Επίσης, η ανάλυση των αρχείων καταγραφών σε logs σφαλμάτων (error) και logs ενημέρωσης λειτουργικότητας (out), μπορεί να δώσει στον διαχειριστή ξεκάθαρη εικόνα για υπηρεσίες οι οποίες δυσλειτουργούν ή παρουσιάζουν συμπτώματα κατάρρευσης. Αυτή η δυνατότητα είναι σημαντικό να έχει υλοποιηθεί έτσι που τα δεδομένα να δίνονται στον διαχειριστή-χρήστη σε πραγματικό χρόνο (real time).[i.29] Βάσεις δεδομένων μεγάλων ταχυτήτων και τεχνολογίες αξιοποίησης συμβάντων, μπορούν να αποδειχθούν ικανές να εξασφαλίσουν την ενημέρωση του τελικού χρήστη σε πραγματικό χρόνο.
- **Μείωση κόστους:** Συγκεντρώνοντας την παρακολούθηση των δεδομένων σε μία ενιαία διεπαφή και μόνο, δίνοντας την δυνατότητα στον διαχειριστή την δυνατότητα πρόληψης προβλημάτων-αποτυχιών του συστήματος, επιτυγχάνεται κατ' επέκταση και μεγαλύτερη αντοχή του συστήματος, προσφέροντας έτσι αποδοτικότερα αποτελέσματα.
- **Αναβάθμιση:** Μέσο της παρακολούθησης των δεδομένων καταγραφής της λειτουργικότητας ενός πληροφοριακού συστήματος, γίνεται πολλές φορές αντιληπτή και η ανάγκη για αλλαγές/αναβαθμίσεις υπηρεσιών ή και ολόκληρου του πληροφοριακού συστήματος.

### 3.6.2 ΒΕΛΤΙΣΤΕΣ ΠΡΑΚΤΙΚΕΣ

Όπως αναφέρει και η Appdynamics της Cisco, η καταγραφή και η παρακολούθηση είναι αμφότερα πολύτιμα στοιχεία για τη διατήρηση της βέλτιστης απόδοσης της εφαρμογής. Η χρήση ενός συνδυασμού εργαλείων καταγραφής και συστημάτων παρακολούθησης σε πραγματικό χρόνο συμβάλλει στη βελτίωση της παρατηρησιμότητας και μειώνει το χρόνο που δαπανάται για την αναζήτηση αρχείων καταγραφής προκειμένου να προσδιοριστεί η βασική αιτία των προβλημάτων απόδοσης.[i.28]

Το logging χρησιμοποιείται τόσο ως ρήμα όσο και ως ουσιαστικό, αναφερόμενο είτε στην πρακτική της καταγραφής σφαλμάτων και αλλαγών είτε στα αρχεία καταγραφής εφαρμογών που συλλέγονται. Ο σκοπός του logging είναι η δημιουργία μιας συνεχούς καταγραφής των συμ-

βάντων της εφαρμογής. Τα αρχεία καταγραφής ("log files") μπορούν να χρησιμοποιηθούν για την ανασκόπηση οποιουδήποτε συμβάντος σε ένα σύστημα, συμπεριλαμβανομένων των αποτυχιών και των μετασχηματισμών κατάστασης. Κατά συνέπεια, τα μηνύματα καταγραφής μπορούν να παρέχουν πολύτιμες πληροφορίες που βοηθούν στον εντοπισμό της αιτίας των προβλημάτων απόδοσης. Τα δεδομένα καταγραφής μπορούν να βοηθήσουν τις ομάδες DevOps να επιλύσουν προβλήματα εντοπίζοντας ποιες αλλαγές οδήγησαν σε αναφορές σφαλμάτων.[i.28]

Η διαχείριση αρχείων καταγραφής ("log management") εξυπηρετεί και άλλους σκοπούς, όπως η δημιουργία γραπτών αρχείων για σκοπούς ελέγχου και συμμόρφωσης, ο εντοπισμός τάσεων με την πάροδο του χρόνου και η διασφάλιση ευαίσθητων πληροφοριών. Η καταγραφή εκτελεί πολύτιμο ρόλο σε εφαρμογές όλων των μεγεθών, αλλά θα πρέπει να εφαρμόζεται με προσοχή. Πρέπει να αποφεύγεται η αποθήκευση, η μεταφορά ή η αξιολόγηση περιττών πληροφοριών, δίνοντας προτεραιότητα στα στοιχεία που μπορούν να αξιοποιηθούν. Η καταγραφή υπερβολικά πολλών δεδομένων μπορεί να δημιουργήσει επιβάρυνση των πόρων, τόσο από άποψη κόστους όσο και χρόνου. Μια καλή στρατηγική καταγραφής παρέχει γενικά δύο τύπους δεδομένων: δομημένα δεδομένα για τις μηχανές και δεδομένα που προειδοποιούν τους διαχειριστές του συστήματος για ένα πιθανό πρόβλημα. Τα συστήματα παρακολούθησης βασίζονται σε μετρήσεις για να ειδοποιούν τις ομάδες πληροφορικής για λειτουργικές ανωμαλίες σε εφαρμογές και υπηρεσίες cloud. Ιδανικά, οι ομάδες θα εφάρμοζαν όργανα παρακολούθησης σε όλα τα συστήματα.[i.28]

Η καταγραφή και η παρακολούθηση είναι δύο διαφορετικές διαδικασίες που συνεργάζονται για να παρέχουν μια σειρά από σημεία δεδομένων που βοηθούν στην παρακολούθηση της υγείας και της απόδοσης μίας πληροφοριακής υποδομής. Η χρήση ενός συνδυασμού διαχείρισης αρχείων καταγραφής για τη συλλογή, οργάνωση και εξέταση δεδομένων και εργαλείων παρακολούθησης για την παρακολούθηση των μετρήσεων προσφέρει μια ολοκληρωμένη εικόνα της διαθεσιμότητας του συστήματός μαζί με λεπτομερή εικόνα για τυχόν προβλήματα, που θα μπορούσαν ενδεχομένως να επηρεάσουν την εμπειρία του χρήστη.[i.28]

Τελικά, ο στόχος σας είναι να διατηρήσετε υγιείς εφαρμογές και την εμπειρία του χρήστη. Με την ενσωμάτωση της καταγραφής και της παρακολούθησης για την επίτευξη αυτού του στόχου, οι ομάδες προγραμματιστών και διαχείρισης θα είναι σε θέση να σχεδιάζουν και να αντιμετωπίζουν ταχύτερα τα προβλήματα των εφαρμογών.[i.28]

Βάσει της Appdynamics υπάρχουν τέσσερις πρακτικές που συμβάλουν στην καλύτερη απόδοση ενός τέτοιου λογισμικού-εργαλείου.

1. **Δώστε τη δυνατότητα και στις δύο μεθόδους να συνεργαστούν.** Εάν ο απώτερος στόχος σας είναι να βελτιστοποιήσετε τα οφέλη από την ανάλυση των δεδομένων καταγραφής και των μετρικών της εφαρμογής, διευκολύνετε αυτή τη διαδικασία ρυθμίζοντας το σύστημά σας ώστε να στέλνει τα δεδομένα καταγραφής απευθείας στο εργαλείο παρακολούθησης. Η αποθήκευση των μηνυμάτων καταγραφής στο δίσκο ή η αποστολή τους αποκλειστικά σε ένα εργαλείο καταγραφής δημιουργεί μεγαλύτερη κατανάλωση πόρων καθώς και μια πιθανή συμφόρηση της ροής εργασιών. Βεβαιωθείτε ότι το εργαλείο παρακολούθησης υποστηρίζει τη γλώσσα προγραμματισμού της εφαρμογής σας για να διασφαλίσετε τη συμβατότητα και την ευκολία χρήσης. [i.28]
2. **Καταγράψτε τα σωστά δεδομένα.** Τα δεδομένα καταγραφής πρέπει να αφηγούνται μια συνοπτική αλλά πλήρη ιστορία. Τα δεδομένα πρέπει να είναι επιλεκτικά, περιγραφικά και να παρέχουν το σωστό πλαίσιο για να βοηθήσουν στην αντιμετώπιση προβλημάτων. Τα χρήσιμα δεδομένα καταγραφής περιλαμβάνουν γενικά στοιχεία που μπορούν να αναληφθούν και περιλαμβάνουν πληροφορίες όπως χρονοσφραγίδα, αναγνωριστικά χρήστη, αναγνωριστικά συνεδρίας και μετρήσεις χρήσης πόρων. Η συλλογή ενός πλήρους φάσματος εφαρμόσιμων δεδομένων ενισχύει τις πληροφορίες που λαμβάνονται από το εργαλείο παρακολούθησης.[i.28]
3. **Χρησιμοποιήστε δομημένα δεδομένα καταγραφής.** Βελτιστοποιήστε τα δεδομένα σας, διευκολύνοντας την αναζήτηση, την ευρετηρίαση και την αποθήκευση, εξασφαλίζοντας ότι είναι δομημένα. Τα δομημένα δεδομένα παρέχουν μια πληρέστερη εικόνα για το τι συνέβη και μπορούν να παρέχουν στο εργαλείο παρακολούθησης μοναδικά αναγνωριστικά, όπως για παράδειγμα ποιο αναγνωριστικό πελάτη παρουσίασε το σφάλμα. Η παροχή πληροφοριών για το αναγνωριστικό πελάτη που λαμβάνονται μέσω της καταγραφής επιτρέπει στο εργαλείο παρακολούθησης να δει πώς επηρεάστηκε ο συγκεκριμένος χρήστης και τι άλλα προβλήματα μπορεί να αντιμετωπίζει ως αποτέλεσμα. [i.28] Εδώ θα πρέπει να σημειωθεί, πως από τη σκοπιά ενός χρήστη αυτό αποτελεί ένα επιπλέον εμπόδιο, διότι η μορφοποίηση και η δόμηση των δεδομένων είναι μία διαδικασία που αναλαμβάνει συνήθως ο developer μίας τέτοιας εφαρμογής και όχι ο χρήστης. Άρα, ή θα πρέπει να έχει



κάποιον τυπικό έλεγχο και ο χρήστης πάνω στην διαδικασία αυτή ή θα πρέπει να δίνονται διάφοροι τρόποι απεικόνισης των δεδομένων, ώστε να εξασφαλισθεί ως ένα βαθμό ή κάθε ανάγκη του χρήστη.

4. **Εκμεταλλευτείτε πλήρως τα δεδομένα καταγραφής.** Η καταγραφή προσφέρει περισσότερα από την απλή αντιμετώπιση προβλημάτων και την αποσφαλμάτωση. Προσδιορίστε τις τάσεις της εφαρμογής και του συστήματος εφαρμόζοντας στατιστική ανάλυση στα συμβάντα του συστήματος. Τα δεδομένα καταγραφής περιέχουν σημαντικές πληροφορίες σχετικά με τις εφαρμογές και την υποκείμενη υποδομή σας, συμπεριλαμβανομένων όλων των βάσεων δεδομένων σας. Χρησιμοποιήστε τις ιστορικές πληροφορίες που παρέχουν τα δεδομένα καταγραφής για να προσδιορίσετε μέσους όρους που θα διευκολύνουν τον οριστικό εντοπισμό ανωμαλιών ή για να ομαδοποιήσετε τύπους συμβάντων με τρόπο που να επιτρέπει ακριβείς συγκρίσεις. Τα δεδομένα αυτά μπορούν επίσης να είναι ωφέλιμα για τη συλλογή, τη συγκέντρωση και την προβολή αυτών των δεδομένων σύμφωνα με τις ανάγκες των επιχειρήσεών σας. Η ύπαρξη συνόλων στατιστικών δεδομένων για εξέταση επιτρέπει μια πιο ακριβή ανάλυση και μια βελτιωμένη ευκαιρία για τη λήψη τεκμηριωμένων επιχειρηματικών αποφάσεων.[i.28]

## 4 ΣΧΕΔΙΑΣΜΟΣ – ΥΠΟΣΤΗΡΙΚΤΙΚΑ ΕΡΓΑΛΕΙΑ

Αρχικά, όπως αναφέρθηκε και στην περίληψη της εργασίας, η εφαρμογή σχετίζεται με την συλλογή και την ανάλυση δεδομένων υπηρεσιών, με απώτερο σκοπό την αναγνώριση λειτουργιών και προβλημάτων σε περιβάλλον σμήνους, που το μοντέλο διανομής υπηρεσιών βασίζεται σε dockerized υποδομή. Πριν εμβαθύνουμε στα της λειτουργικότητας της εφαρμογής ή στην φιλοσοφία της, πρέπει πρώτα να αναφέρουμε και να αναλύσουμε περιληπτικά τις τεχνολογίες που χρησιμοποιήθηκαν κατά την υλοποίηση της. Τέλος, τα εργαλεία-τεχνολογίες που χρησιμοποιήθηκαν, είναι εργαλεία-τεχνολογίες επαγγελματικής κλίμακας, που χρησιμοποιούνται σήμερα από τους μεγαλύτερους οργανισμούς στον τομέα της πληροφορικής.

### 4.1 DOCKER

Το Docker είναι μια ανοιχτή πλατφόρμα για την ανάπτυξη, αποστολή και εκτέλεση εφαρμογών. Το Docker σας επιτρέπει να διαχωρίσετε τις εφαρμογές σας από την υποδομή σας, ώστε να μπορείτε να παράγετε το λογισμικό σας γρήγορα. Με το Docker, μπορείτε να διαχειρίζεστε την υποδομή σας με τους ίδιους τρόπους που διαχειρίζεστε τις εφαρμογές σας. Αξιοποιώντας τις μεθοδολογίες του Docker για τη γρήγορη αποστολή, δοκιμή και ανάπτυξη κώδικα, μπορείτε να μειώσετε σημαντικά την καθυστέρηση μεταξύ της συγγραφής κώδικα και της εκτέλεσής του στην παραγωγή.[i.30]

Το Docker παρέχει τη δυνατότητα πακεταρίσματος και εκτέλεσης μιας εφαρμογής σε ένα χαλαρά απομονωμένο περιβάλλον που ονομάζεται container. Σε προηγούμενο κεφάλαιο μιλήσαμε για τους περιέκτες (containers), αλλά για λόγους υπενθύμισης αξίζει να αναφερθεί πως η απομόνωση και η ασφάλεια, σας επιτρέπουν να εκτελείτε πολλά container ταυτόχρονα σε έναν δεδομένο κεντρικό υπολογιστή. Τα container είναι ελαφριά και περιέχουν όλα όσα απαιτούνται για

την εκτέλεση της εφαρμογής, οπότε δεν χρειάζεται να βασίζεστε σε ό,τι είναι εγκατεστημένο επί του παρόντος στον κεντρικό υπολογιστή. Μπορείτε εύκολα να μοιράζεστε κοντέινερ ενώ εργάζεστε και να είστε σίγουροι ότι όλοι όσοι μοιράζεστε μαζί τους θα έχουν το ίδιο κοντέινερ που λειτουργεί με τον ίδιο τρόπο.[i.30]

Το Docker παρέχει εργαλεία και μια πλατφόρμα για τη διαχείριση του κύκλου ζωής των containers σας:

- Αναπτύσσετε την εφαρμογή σας και τα στοιχεία που την υποστηρίζουν χρησιμοποιώντας κοντέινερ.
- Το κοντέινερ γίνεται η μονάδα για τη διανομή και τη δοκιμή της εφαρμογής σας.
- Όταν είστε έτοιμοι, αναπτύσσετε την εφαρμογή σας στο περιβάλλον παραγωγής σας, ως κοντέινερ ή ως οργανωμένη υπηρεσία. Αυτό λειτουργεί με τον ίδιο τρόπο είτε το περιβάλλον παραγωγής σας είναι ένα τοπικό κέντρο δεδομένων, ένας πάροχος cloud ή ένα υβρίδιο των δύο.[i.30]

#### 4.1.1 ΛΟΓΟΙ ΧΡΗΣΗΣ ΤΟΥ DOCKER

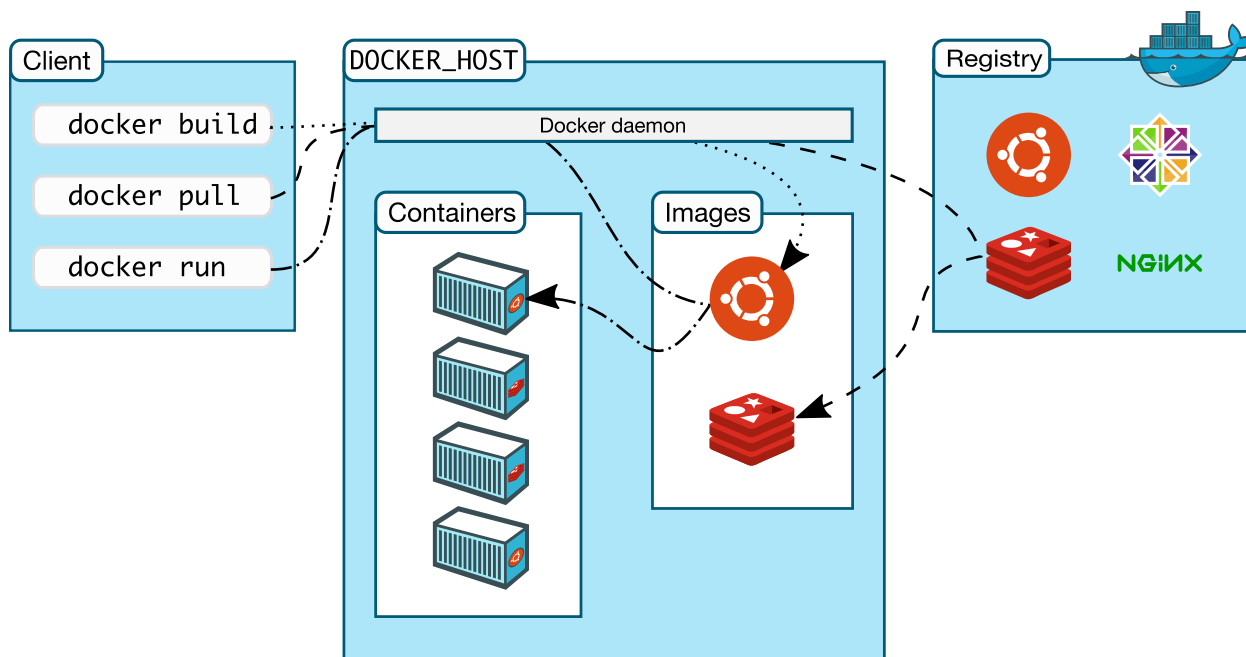
1. **Γρήγορη, συνεπής παράδοση των εφαρμογών σας.** Το Docker βελτιώνει τον κύκλο ζωής της ανάπτυξης, επιτρέποντας στους προγραμματιστές να εργάζονται σε τυποποιημένα περιβάλλοντα χρησιμοποιώντας τοπικά κοντέινερ που παρέχουν τις εφαρμογές και τις υπηρεσίες σας. Τα κοντέινερ είναι ιδανικά για ροές εργασίας συνεχούς ολοκλήρωσης και συνεχούς παράδοσης (CI/CD). Σκεφτείτε το ακόλουθο παράδειγμα σεναρίου:[i.30]
  - Οι προγραμματιστές σας γράφουν κώδικα τοπικά και μοιράζονται τη δουλειά τους με τους συναδέλφους τους χρησιμοποιώντας κοντέινερ Docker.
  - Χρησιμοποιούν το Docker για να προωθήσουν τις εφαρμογές τους σε ένα περιβάλλον δοκιμών και να εκτελέσουν αυτοματοποιημένες και χειροκίνητες δοκιμές.
  - Όταν οι προγραμματιστές βρίσκουν σφάλματα, μπορούν να τα διορθώσουν στο περιβάλλον ανάπτυξης και να τα επανατοποθετήσουν στο περιβάλλον δοκιμών για έλεγχο και επικύρωση.

- Όταν ολοκληρωθεί η δοκιμή, η παροχή της διόρθωσης στον πελάτη είναι τόσο εύκολη όσο η προώθηση της ενημερωμένης εικόνας στο περιβάλλον παραγωγής.  
[i.30]

- 2. Ανταποκρινόμενη ανάπτυξη και κλιμάκωση.** Η πλατφόρμα του Docker που βασίζεται σε κοντέινερ επιτρέπει ιδιαίτερα φορητούς φόρτους εργασίας. Τα κοντέινερ του Docker μπορούν να εκτελούνται στον τοπικό φορητό υπολογιστή ενός προγραμματιστή, σε φυσικές ή εικονικές μηχανές σε ένα κέντρο δεδομένων, σε παρόχους cloud ή σε ένα μείγμα περιβαλλόντων. Η φορητότητα και ο ελαφρύς χαρακτήρας του Docker καθιστούν επίσης εύκολη τη δυναμική διαχείριση των φόρτων εργασίας, αυξάνοντας ή κατεβάζοντας εφαρμογές και υπηρεσίες ανάλογα με τις επιχειρηματικές ανάγκες, σε σχεδόν πραγματικό χρόνο.[i.30]
- 3. Εκτέλεση περισσότερων φόρτων εργασίας στο ίδιο υλικό.** Το Docker είναι ελαφρύ και γρήγορο. Παρέχει μια βιώσιμη, οικονομικά αποδοτική εναλλακτική λύση σε σχέση με τις εικονικές μηχανές που βασίζονται σε hypervisor, ώστε να μπορείτε να χρησιμοποιήσετε περισσότερη χωρητικότητα του υπολογιστικού σας δυναμικού για την επίτευξη των επιχειρηματικών σας στόχων. Το Docker είναι ιδανικό για περιβάλλοντα υψηλής πυκνότητας και για μικρές και μεσαίες εγκαταστάσεις όπου πρέπει να κάνετε περισσότερα με λιγότερους πόρους.[i.30]

#### 4.1.2 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ DOCKER

Το Docker χρησιμοποιεί μια αρχιτεκτονική πελάτη-εξυπηρετητή. Ο πελάτης Docker μιλάει με τον daemon του Docker, ο οποίος αναλαμβάνει την κατασκευή, την εκτέλεση και τη διανομή των Docker κοντέινερ. Ο πελάτης Docker και ο daemon μπορούν να εκτελούνται στο ίδιο σύστημα ή μπορείτε να συνδέσετε έναν πελάτη Docker σε έναν απομακρυσμένο daemon Docker. Ο πελάτης Docker και ο daemon επικοινωνούν χρησιμοποιώντας ένα REST API, μέσω υποδοχών UNIX ή μιας διασύνδεσης δικτύου. Ένας άλλος client του Docker είναι το Docker Compose, το οποίο σας επιτρέπει να εργάζεστε με εφαρμογές που αποτελούνται από ένα σύνολο containers. Στα πλαίσια της εργασίας εδώ πρέπει να σημειωθεί, πως κάναμε χρήση του docker compose.[i.30]



Εικόνα 13: Αρχιτεκτονική Docker

#### 4.1.3 Ο DAEMON ΤΟΥ DOCKER

Ο daemon του Docker (dockerd) ακούει τα αιτήματα API του Docker και διαχειρίζεται αντικείμενα του Docker, όπως εικόνες, κοντέινερ, δίκτυα και τόμους. Ένας daemon μπορεί επίσης να επικοινωνεί με άλλους daemon για τη διαχείριση υπηρεσιών Docker.[i.30]

#### 4.1.4 Ο CLIENT ΤΟΥ DOCKER

Ο Docker client (docker) είναι ο κύριος τρόπος με τον οποίο πολλοί χρήστες του Docker αλληλεπιδρούν με το Docker. Όταν χρησιμοποιείτε εντολές όπως `docker run`, ο client στέλνει αυτές τις εντολές στο dockerd, το οποίο τις εκτελεί. Η εντολή `docker` χρησιμοποιεί το API του Docker. Ο client του Docker μπορεί να επικοινωνεί με περισσότερους από έναν daemon.[i.30]

#### 4.1.5 ΜΗΤΡΩΑ DOCKER

Ένα μητρώο Docker αποθηκεύει εικόνες Docker. Το Docker Hub είναι ένα δημόσιο μητρώο που μπορεί να χρησιμοποιήσει οποιοσδήποτε και το Docker είναι ρυθμισμένο να αναζητά ει-

κόνες στο Docker Hub από προεπιλογή. Μπορείτε ακόμη και να εκτελέσετε το δικό σας ιδιωτικό μητρώο. Όταν χρησιμοποιείτε τις εντολές `docker pull` ή `docker run`, οι απαιτούμενες εικόνες αντλούνται από το ρυθμισμένο μητρώο σας. Όταν χρησιμοποιείτε την εντολή `docker push`, η εικόνα σας ωθείται στο ρυθμισμένο μητρώο σας.[i.30]

#### 4.1.6 ANTIKEIMENA DOCKER

Όταν χρησιμοποιείτε το Docker, δημιουργείτε και χρησιμοποιείτε εικόνες, κοντέινερ, δίκτυα, τόμους, πρόσθετα και άλλα αντικείμενα. Αυτή η ενότητα είναι μια σύντομη επισκόπηση ορισμένων από αυτά τα αντικείμενα.

- **Εικόνες:** Μια εικόνα είναι ένα πρότυπο μόνο για ανάγνωση με οδηγίες για τη δημιουργία ενός κοντέινερ Docker. Συχνά, ένα image βασίζεται σε ένα άλλο image, με κάποια πρόσθετη προσαρμογή. Για παράδειγμα, μπορείτε να δημιουργήσετε ένα image το οποίο βασίζεται στο image του ubuntu, αλλά εγκαθιστά τον διακομιστή ιστού Apache και την εφαρμογή σας, καθώς και τις λεπτομέρειες παραμετροποίησης που απαιτούνται για την εκτέλεση της εφαρμογής σας. Μπορείτε να δημιουργήσετε τις δικές σας εικόνες ή να χρησιμοποιήσετε μόνο εκείνες που έχουν δημιουργηθεί από άλλους και έχουν δημοσιευτεί σε ένα μητρώο. Για να δημιουργήσετε το δικό σας image, δημιουργείτε ένα Dockerfile με μια απλή σύνταξη για τον ορισμό των βημάτων που απαιτούνται για τη δημιουργία του image και την εκτέλεσή του. Κάθε εντολή σε ένα αρχείο Docker δημιουργεί ένα επίπεδο στην εικόνα. Όταν αλλάζετε το Dockerfile και ανακατασκευάζετε την εικόνα, ανακατασκευάζονται μόνο τα στρώματα που έχουν αλλάξει. Αυτό είναι μέρος του γεγονότος που κάνει τα images τόσο ελαφριά, μικρά και γρήγορα, σε σύγκριση με άλλες τεχνολογίες εικονικοποίησης.[i.30]
- **Containers:** Ένα container είναι μια εκτελέσιμη περίπτωση μιας εικόνας. Μπορείτε να δημιουργήσετε, να εκκινήσετε, να σταματήσετε, να μετακινήσετε ή να διαγράψετε ένα container χρησιμοποιώντας το API του Docker ή το CLI. Μπορείτε να συνδέσετε ένα container σε ένα ή περισσότερα δίκτυα, να προσαρτήσετε αποθηκευτικό χώρο σε αυτό ή ακόμη και να δημιουργήσετε μια νέα εικόνα με βάση την τρέχουσα κατάστασή του. Από προεπιλογή, ένα κοντέινερ είναι σχετικά καλά απομονωμένο από άλλα κοντέινερ και το

μηχάνημα υποδοχής του. Μπορείτε να ελέγξετε πόσο απομονωμένο είναι το δίκτυο, η αποθήκευση ή άλλα υποκείμενα υποσυστήματα ενός κοντέινερ από άλλα κοντέινερ ή από το μηχάνημα υποδοχής. Ένα container ορίζεται από την εικόνα του, καθώς και από οποιεσδήποτε επιλογές παραμετροποίησης που του παρέχετε κατά τη δημιουργία ή την εκκίνησή του. Όταν ένα κοντέινερ απομακρύνεται, εξαφανίζονται όλες οι αλλαγές στην κατάσταση του που δεν αποθηκεύονται σε μόνιμο αποθηκευτικό χώρο.[i.30]

#### 4.1.7 ΓΙΑΤΙ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ ΤΟ DOCKER

Ανάγκη της εργασίας ήταν να εξετασθή και να υλοποιηθεί η εφαρμογή σε ένα περιβάλλον, στο οποίο η υποκείμενη υποδομή του συστήματος θα ήταν υλοποιημένη σε docker. Αυτό βέβαια δεν αποτελεί το μόνο λόγο, αφού το Docker όπως εξηγήθηκε και παραπάνω είναι ένα εξαιρετικό εργαλείο δημιουργίας, εκτέλεσης και διαχείρισης των container ενός συστήματος, έτσι ήταν κομβική επιλογή για την υλοποίηση της εφαρμογής.

### 4.2 NODEJS

Ως ένα ασύγχρονο πρόγραμμα εκτέλεσης JavaScript με βάση τα event, το Node.js έχει σχεδιαστεί για τη δημιουργία κλιμακούμενων δικτυακών εφαρμογών. Ο σχεδιασμός του Node.js είναι παρόμοιος και επηρεασμένος από συστήματα όπως το Event Machine της Ruby και το Twisted της Python. Το Node.js πηγαίνει το μοντέλο συμβάντων λίγο παραπέρα. Παρουσιάζει έναν βρόχο συμβάντων ως μια κατασκευή χρόνου εκτέλεσης αντί για μια βιβλιοθήκη. Σε άλλα συστήματα, υπάρχει πάντα μια κλήση φραγής για την εκκίνηση του βρόχου γεγονότων. Συνήθως, η συμπεριφορά ορίζεται μέσω callbacks στην αρχή ενός σεναρίου και στο τέλος ξεκινάει ένας διακομιστής μέσω μιας κλήσης φραγής όπως η EventMachine::run(). Στο Node.js, δεν υπάρχει τέτοια κλήση start-the-event-loop. Το Node.js απλώς εισέρχεται στο βρόχο συμβάντων μετά την εκτέλεση του σεναρίου εισόδου. Το Node.js βγαίνει από το βρόχο συμβάντων όταν δεν

υπάρχουν άλλα callbacks προς εκτέλεση. Αυτή η συμπεριφορά είναι όπως η JavaScript του προγράμματος περιήγησης - ο βρόχος συμβάντων είναι κρυμμένος από τον χρήστη.[i.31]

Το HTTP είναι πρώτης κατηγορίας παίκτης στο Node.js, σχεδιασμένο με γνώμονα τη ροή και τη χαμηλή καθυστέρηση. Αυτό καθιστά το Node.js κατάλληλο για τη θεμελίωση μιας βιβλιοθήκης ή ενός framework ιστού.[i.31]

Το γεγονός ότι το Node.js έχει σχεδιαστεί χωρίς threads δεν σημαίνει ότι δεν μπορείτε να εκμεταλλευτείτε τους πολλαπλούς πυρήνες στο περιβάλλον σας. Οι διεργασίες child μπορούν να δημιουργηθούν χρησιμοποιώντας το API `child_process.fork()` και έχουν σχεδιαστεί ώστε να είναι εύκολη η επικοινωνία με αυτές. Βασισμένη στην ίδια διεπαφή είναι η ενότητα cluster, η οποία σας επιτρέπει να μοιράζεστε υποδοχές μεταξύ διεργασιών για να επιτρέψετε την εξισορρόπηση φορτίου στους πυρήνες σας.[i.31]

#### 4.2.1 ΓΙΑΤΙ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ ΤΟ NODEJS

Η αξία του Node.js και ο λόγος που χρησιμοποιήθηκε στα πλαίσια της εφαρμογής, μπορεί να εξαχθεί από το γεγονός ότι πρόκειται για μία επιλογή που αποδίδει με μεγάλες ταχύτητες, ασύγχρονα, είναι βασισμένο σε events, ενώ δουλεύει με Javascript, μία γλώσσα που κάθε web developer έχει έρθει αντιμέτωπος και θα έρχεται αντιμέτωπος συνεχώς στην καριέρα του.

#### 4.3 FLUENTD

Το Fluentd είναι ένας συλλέκτης δεδομένων ανοικτού κώδικα, ο οποίος σας επιτρέπει να ενοποιήσετε τη συλλογή και την επεξεργασία δεδομένων για καλύτερη χρήση και κατανόηση των δεδομένων.[i.32]



#### 4.3.1 ΕΝΟΠΟΙΗΜΕΝΗ ΚΑΤΑΓΡΑΦΗ ΜΕ JSON

Το Fluentd προσπαθεί να δομεί τα δεδομένα ως JSON όσο το δυνατόν περισσότερο: αυτό επιτρέπει στο Fluentd να ενοποιήσει όλες τις πτυχές της επεξεργασίας δεδομένων καταγραφής: συλλογή, φιλτράρισμα, προσωρινή αποθήκευση και εξαγωγή καταγραφών σε πολλαπλές πηγές και προορισμούς (Unified Logging Layer). Η μεταγενέστερη επεξεργασία δεδομένων είναι πολύ πιο εύκολη με JSON, καθώς έχει αρκετή δόμηση ώστε να είναι προσβάσιμη, διατηρώντας παράλληλα ευέλικτα σχήματα.[i.32]

#### 4.3.2 PLUGGABLE ΑΡΧΙΤΕΚΤΟΝΙΚΗ

Το Fluentd διαθέτει ένα ευέλικτο σύστημα επέκτασης που επιτρέπει στην κοινότητα να επεκτείνει τη λειτουργικότητά του. Τα 500+ πρόσθετα που συνεισφέρει η κοινότητά μας συνδέουν δεκάδες πηγές δεδομένων και εξόδους δεδομένων. Αξιοποιώντας τα πρόσθετα, μπορείτε να αρχίσετε αμέσως να αξιοποιείτε καλύτερα τα αρχεία καταγραφής σας.[i.32]

#### 4.3.3 ΕΛΑΧΙΣΤΟΙ ΑΠΑΙΤΟΥΜΕΝΟΙ ΠΟΡΟΙ

Το Fluentd είναι γραμμένο σε συνδυασμό γλώσσας C και Ruby και απαιτεί πολύ λίγους πόρους συστήματος. Η περίπτωση vanilla τρέχει με 30-40MB μνήμης και μπορεί να επεξεργαστεί 13.000 συμβάντα/δευτερόλεπτο/πυρήνα. Ακόμα και σε πιο περιορισμένες απαιτήσεις μνήμης (-450kb), υπάρχει το Fluent Bit, μια πιο ελαφριά έκδοση του Fluentd.[i.32]

#### 4.3.4 ΕΝΣΩΜΑΤΟΜΕΝΗ ΑΞΙΟΠΙΣΤΙΑ

Το Fluentd υποστηρίζει buffering με βάση τη μνήμη και το αρχείο για να αποτρέψει την απώλεια δεδομένων μεταξύ των κόμβων. Το Fluentd υποστηρίζει επίσης αξιόπιστη επαναφορά σε περίπτωση αποτυχίας και μπορεί να ρυθμιστεί για υψηλή διαθεσιμότητα. 2.000+ εταιρείες με

γνώμονα τα δεδομένα βασίζονται στο Fluentd για να διαφοροποιήσουν τα προϊόντα και τις υπηρεσίες τους μέσω της καλύτερης χρήσης και κατανόησης των δεδομένων καταγραφής τους.[i.32]

#### 4.3.5 ΠΛΕΟΝΕΚΤΗΜΑΤΑ

- **Ενοποιημένο επίπεδο καταγραφής:** Το Fluentd αποσυνδέει τις πηγές δεδομένων από τα backend συστήματα παρέχοντας ένα ενοποιημένο επίπεδο καταγραφής στο ενδιαμέσο. Αυτό το στρώμα επιτρέπει στους προγραμματιστές και τους αναλυτές δεδομένων να χρησιμοποιούν πολλούς τύπους καταγραφών καθώς παράγονται. Εξίσου σημαντικό, μετριάζει τον κίνδυνο "κακών δεδομένων" που επιβραδύνουν και παραπληροφορούν τον οργανισμό σας. Ένα ενοποιημένο επίπεδο καταγραφής επιτρέπει σε εσάς και τον οργανισμό σας να αξιοποιείτε καλύτερα τα δεδομένα και να επαναλαμβάνετε πιο γρήγορα το λογισμικό σας.[i.33]
- **Απλό και εύκολο αλλά ευέλικτο:** Το Fluentd μπορεί να εγκατασταθεί στον φορητό σας υπολογιστή σε λιγότερο από 10 λεπτά ενώ μπορείτε να το κατεβάσετε και να το δοκιμάσετε αμέσως. Τα 500+ plugins του Fluentd το καθιστούν συμβατό με δεκάδες πηγές δεδομένων και εξόδους δεδομένων. Τα πρόσθετα είναι επίσης εύκολο να γραφτούν και να αναπτυχθούν. [i.33]
- **Ανοιχτός κώδικας:** Το Fluentd είναι λογισμικό με άδεια Apache 2.0, πλήρως ανοιχτού κώδικα. Αυτό σημαίνει ότι η φαντασία σας και όχι οι περιορισμοί της άδειας χρήσης είναι το όριο του τι μπορείτε να επιτύχετε με το Fluentd. Ο πηγαίος κώδικας είναι διαθέσιμος στο GitHub.[i.33]
- **Αποδεδειγμένη αξιοπιστία και απόδοση:** 5.000+ εταιρείες με γνώμονα τα δεδομένα βασίζονται στη Fluentd για να διαφοροποιήσουν τα προϊόντα και τις υπηρεσίες τους μέσω της καλύτερης χρήσης και κατανόησης των δεδομένων καταγραφής τους. Στην πραγματικότητα, σύμφωνα με την έρευνα της Datadog, η Fluentd είναι η 7η κορυφαία τεχνολογία που εκτελείται σε περιβάλλοντα δοχείων Docker. Ορισμένοι χρήστες του Fluentd συλλέγουν δεδομένα από χιλιάδες μηχανές σε πραγματικό χρόνο. Χάρη στο μικρό αποτύπωμα μνήμης (30~40MB), μπορείτε να εξοικονομήσετε πολλή μνήμη σε κλίμακα [i.33]

- **Κοινότητα:** Η κοινότητα πίσω από το Fluentd βελτιώνει συνεχώς το λογισμικό και βοηθάει η μία την άλλη για να κάνει το Fluentd χρήσιμο για όλους.[i.33]

#### 4.3.6 ΓΙΑΤΙ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ ΤΟ FLUENTD

Το εργαλείο Fluentd αποτέλεσε, δίχως καμία αμφιβολία, το άλφα και το ωμέγα της φιλοσοφίας της εφαρμογής, αφού όλες οι λειτουργικότητες της συγκέντρωσης, αποθήκευσης και ανάλυσης των δεδομένων καταγραφής, έγιναν χάρης αυτού.

### 4.4 VUE.JS

Το Vue είναι ένα προοδευτικό front-end framework για την κατασκευή διεπαφών χρήστη. Σε αντίθεση με άλλα μονολιθικά πλαίσια, το Vue έχει σχεδιαστεί από την αρχή για να μπορεί να υιοθετηθεί σταδιακά. Η βασική βιβλιοθήκη επικεντρώνεται μόνο στο επίπεδο προβολής και είναι εύκολο να παραληφθεί και να ενσωματωθεί με άλλες βιβλιοθήκες ή υπάρχοντα έργα. Από την άλλη πλευρά, το Vue είναι επίσης απόλυτα ικανό να τροφοδοτήσει εξελιγμένες εφαρμογές μίας σελίδας όταν χρησιμοποιείται σε συνδυασμό με σύγχρονα εργαλεία και υποστηρικτικές βιβλιοθήκες.[i.34] Επίσης, είναι άξιο να αναφερθεί και η δυνατότητα προσαρμοστικότητας μίας εφαρμογής που χτίζεται με το Vue.js, όπως η δυνατότητά της να απεικονίζεται άρτια σε μικρότερες διαστάσεων οθόνες. Αυτό βέβαια είναι κάτι το οποίο συστήνεται να έχει ξεκινήσει να χτίζεται εξ αρχής από τον developer, που έχει αναλάβει την εφαρμογή.

#### 4.4.1 ΓΙΑΤΙ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ ΤΟ VUE.JS

Εναλλακτικές περιπτώσεις θα μπορούσαν να είναι το React.js και το Angular, αλλά λόγω της ευκολίας εκμάθησης του Vue.js. του αναλυτικότατου documentation, που παρέχει σε νέους χρήστες, των πολλών επιπλέον βιβλιοθηκών και εργαλείων, και της συμβατότητας που διαθέτει σε

ορισμένες τεχνολογίες που χρησιμοποιήθηκαν όπως τα socket.io, ήταν η καλύτερη επιλογή για την υλοποίηση αρκετών λειτουργιών που λαμβάνουν χώρα στην εφαρμογή.

## 4.5 SOCKET.IO

Το Socket.IO επιτρέπει επικοινωνία σε πραγματικό χρόνο, αμφίδρομη και βασισμένη σε συμβάντα. Λειτουργεί σε κάθε πλατφόρμα, πρόγραμμα περιήγησης ή συσκευή, εστιάζοντας εξίσου στην αξιοπιστία και την ταχύτητα. Αποτελείται από: έναν διακομιστή Node.js και μια βιβλιοθήκη-πελάτη Javascript για το πρόγραμμα περιήγησης.[i.35]

Πώς λειτουργεί όμως; Ο πελάτης θα προσπαθήσει να δημιουργήσει μια σύνδεση WebSocket αν είναι δυνατόν, και αν όχι, θα επιστρέψει στη μακροχρόνια επικοινωνία HTTP. Το WebSocket είναι ένα πρωτόκολλο επικοινωνίας που παρέχει ένα κανάλι πλήρους διπλής όψης και χαμηλής καθυστέρησης μεταξύ του διακομιστή και του προγράμματος περιήγησης. Έτσι, στο καλύτερο σενάριο, υπό την προϋπόθεση ότι:

1. Το πρόγραμμα περιήγησης υποστηρίζει το WebSocket (97% όλων των προγραμμάτων περιήγησης το 2020)
2. Δεν υπάρχει κανένα στοιχείο (proxy, τείχος προστασίας, ...) που να εμποδίζει τις συνδέσεις WebSocket μεταξύ του πελάτη και του διακομιστή,

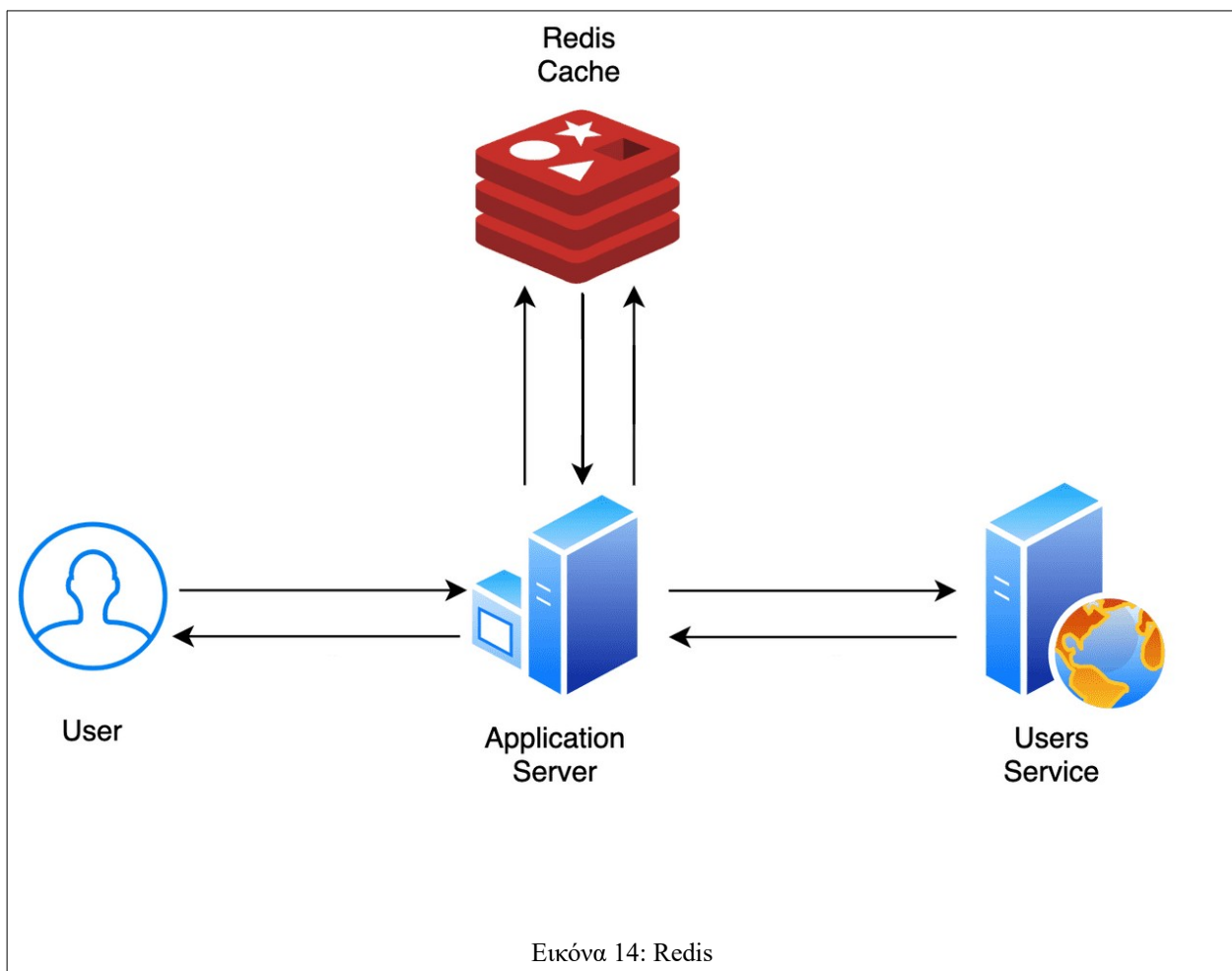
μπορείτε να θεωρήσετε τον Socket.IO client ως ένα "ελαφρύ" περιτύλιγμα γύρω από το WebSocket API.[i.35]

### 4.5.1 ΓΙΑΤΙ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ ΤΟ SOCKET.IO

Χρησιμοποιώντας sockets, και πιο συγκεκριμένα το socket.io ικανοποιήθηκε η ανάγκη να στέλνονται και να λαμβάνονται τα δεδομένα σε πραγματικό χρόνο, δυνατότητα που όπως προαναφέρθηκε σε προηγούμενο κεφάλαιο είναι ιδιαίτερα σημαντική σε μία εφαρμογή συλλογής, ανάλυσης και παρακολούθησης δεδομένων.

## 4.6 REDIS

Το Redis είναι ένας ανοιχτού κώδικα (με άδεια BSD), αποθηκευτής δομών δεδομένων στη μνήμη, που χρησιμοποιείται ως βάση δεδομένων, κρυφή μνήμη και διαμεσολαβητής μηνυμάτων. Το Redis παρέχει δομές δεδομένων, όπως συμβολοσειρές, hashes, λίστες, sets, ταξινομημένα sets με range queries, bitmaps, hyperloglogs, geospatial indexes και streams. Το Redis διαθέτει ενσωματωμένη αντιγραφή, Lua scripting, LRU eviction, συναλλαγές και διαφορετικά επίπεδα εμμόνης στο δίσκο, ενώ παρέχει υψηλή διαθεσιμότητα μέσω του Redis Sentinel και αυτόματη κατάτμηση με το Redis Cluster.



#### 4.6.1 ΓΙΑΤΙ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ ΤΟ REDIS

Ο λόγος που χρησιμοποιήθηκε το Redis στην εφαρμογή είναι, ώστε να εξυπηρετεί ως κοινή μνήμη, ακόμα και σε περίπτωση που το σύστημα τρέξει σε περιβάλλον cluster, ώστε να εξασφαλίσει την ομαλή λειτουργία της εφαρμογής.

#### 4.7 MONGODB

Η MongoDB είναι μια βάση δεδομένων εγγράφων με την επεκτασιμότητα και την ευελιξία που θέλει μία εφαρμογή και με την αναζήτηση και την ευρετηρίαση που χρειάζεται. Το μοντέλο εγγράφων της MongoDB είναι απλό στην εκμάθηση και τη χρήση από τους προγραμματιστές, ενώ παράλληλα παρέχει όλες τις δυνατότητες που απαιτούνται για την ικανοποίηση των πιο σύνθετων απαιτήσεων σε οποιαδήποτε κλίμακα. Επιπλέον παρέχονται, προγράμματα οδήγησης (“drivers”) για 10+ γλώσσες και η κοινότητα έχει δημιουργήσει δεκάδες ακόμη.[i.37]

##### 4.7.1 ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΧΡΗΣΗΣ

Επιγραμματικά τα πλεονεκτήματα χρήσης που παρέχει μία MongoDB βάση δεδομένων είναι τα εξής:

- Η MongoDB αποθηκεύει δεδομένα σε ευέλικτα έγγραφα τύπου JSON, που σημαίνει ότι τα πεδία μπορούν να διαφέρουν από έγγραφο σε έγγραφο και η δομή των δεδομένων μπορεί να αλλάξει με την πάροδο του χρόνου.[i.37]
- Το μοντέλο εγγράφου αντιστοιχίζεται με τα αντικείμενα στον κώδικα της εφαρμογής σας, καθιστώντας τα δεδομένα εύκολα επεξεργάσιμα.[i.37]
- Τα ad hoc ερωτήματα, η ευρετηρίαση και η συγκέντρωση σε πραγματικό χρόνο παρέχουν ισχυρούς τρόπους πρόσβασης και ανάλυσης των δεδομένων σας.[i.37]

- Η MongoDB είναι μια κατανεμημένη βάση δεδομένων στον πυρήνα της, οπότε η υψηλή διαθεσιμότητα, η οριζόντια κλιμάκωση και η γεωγραφική κατανομή είναι ενσωματωμένες και εύκολες στη χρήση.[i.37]
- Η MongoDB είναι ελεύθερη στη χρήση. Οι εκδόσεις που κυκλοφόρησαν πριν από τις 16 Οκτωβρίου 2018 δημοσιεύονται υπό την AGPL. Όλες οι εκδόσεις που κυκλοφόρησαν μετά τις 16 Οκτωβρίου 2018, συμπεριλαμβανομένων των διορθώσεων διορθώσεων για προηγούμενες εκδόσεις, δημοσιεύονται υπό την άδεια χρήσης Server Side Public License (SSPL) v1.[i.37]

## 5 ΥΛΟΠΟΙΗΣΗ-ΠΕΡΙΓΡΑΦΗ ΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑΣ

Για να γίνει πλήρως κατανοητή η λειτουργικότητα και η φιλοσοφία της εφαρμογής θα ξεκινήσουμε από το back-end και θα προχωρήσουμε προς την δημιουργία και υλοποίηση της διεπαφής χρήστη, όπου τελικά φαίνονται σε τελική ανάλυση όλα όσα προγραμματίστηκαν καθ' όλη τη διάρκεια. Παρακάτω αναφέρονται επιγραμματικά όλα τα container που τρέχουν στο εικονικό δίκτυο της εφαρμογής μέσω docker και στη συνέχεια, θα αναλυθούν περαιτέρω το καθένα ξεχωριστά. Τα container είναι:

- **fluentd**: Σε αυτό το container τρέχει μία μικρο-υπηρεσία που αξιοποιεί το εργαλείο fluentd που αναλύθηκε στο προηγούμενο κεφάλαιο.
- **redisserver**: Σε αυτό το container τρέχει μία έκδοση του redis στο οποίο συνδέεται η εφαρμογή μας για να επιτευχθούν διάφορες λειτουργικότητες.
- **Mongo**: Container στο οποίο τρέχει η βάση δεδομένων της MongoDB και αναλαμβάνει την αποθήκευση των δεδομένων.
- **readmongo\_service**: container στο οποίο τρέχει εφαρμογή σε node.js-express
- **dummy\_service**: container στο οποίο τρέχει εφαρμογή που σχετίζεται με testing και εξυπηρετεί ως παράδειγμα, ώστε να γίνει κατανοητή η λειτουργικότητα της εφαρμογής.

**Σημείωση:** Η διεπαφή του χρήστη που φτιάχτηκε σε vue.js δεν αποτελεί container του δικτύου αν και θα μπορούσε. Προτίμησα να μείνει εκτός του εικονικού δικτύου ώστε να είναι πιο εύκολη η διαδικασία αλλαγών στον κώδικα, αλλά να φανεί και η δυνατότητα επικοινωνίας με το εικονικό δίκτυο από μία εξωτερική οντότητα.

### 5.1 DOCKERFILE ΚΑΙ DOCKER-COMPOSE

Όλα τα κοντέινερ της εφαρμογής δημιουργούνται και διαχειρίζονται μέσω του docker και πιο συγκεκριμένα μέσω του docker-compose client που αναφέρθηκε και στα εργαλεία. Για να επι-



τευχθεί αυτό, είναι απαραίτητη η δημιουργία ενός αρχείου παραμετροποίησης όλων των κοντέινερ, συνήθως ονομαζόμενο `docker-compose.yml`. Πριν όμως από αυτό θα πρέπει να υπάρξει και η απαραίτητη εικόνα για κάθε κοντέινερ. Οι εικόνες δημιουργούνται χειροκίνητα μέσω ενός αρχείου που ονομάζεται `Dockerfile` ή παίρνονται έτοιμες από κάποιο `docker hub` αν είναι διαθέσιμες, δηλώνοντας τα απαραίτητα στοιχεία μέσω του `docker-compose`.

## 5.2 ΔΗΜΙΟΥΡΓΙΑ ΕΙΚΟΝΩΝ

Στο παρακάτω `Dockerfile` γίνεται η παραμετροποίηση της εικόνας της μικροπηρεσίας `fluentd`. Ο τρόπος που συντάχθηκε πάρθηκε από την επίσημη σελίδα του εργαλείου.

```
FROM fluent/fluentd:v1.11-1

# Use root account to use apk
USER root

# below RUN includes plugin as examples elasticsearch is not required
# you may customize including plugins as you wish
RUN apk add --no-cache --update --virtual .build-deps \
    sudo build-base ruby-dev \
    && sudo gem install fluent-plugin-mongo \
    && sudo gem sources --clear-all \
    && apk del .build-deps \
    && rm -rf /tmp/* /var/tmp/* /usr/lib/ruby/gems/*/cache/*.gem

COPY fluent.conf /fluentd/etc/
COPY entrypoint.sh /bin/

USER fluent
```

Στο παρακάτω `Dockerfile` υλοποιείται η εικόνα του container του `readmongo_service`. Βλέπουμε πως η εικόνα χτίζεται πάνω σε μία ήδη υπάρχουσα δομή `alpine`, όπως είδαμε και στο κεφάλαιο των περιεκτών. Παρακάτω, γίνεται η εγκατάσταση των απαραίτητων πακέτων λογισμικών εντός της εικόνας, δημοσιοποιείται η πόρτα 3000, που είναι η πόρτα επικοινωνίας με την διεπαφή χρήστη (web client), ενώ δηλώνεται ο τρόπος εκτέλεσης της εφαρμογής μέσω `pm2`-

runtime και του αρχείου `ecosystem.config.js`. Αξίζει να αναφερθεί πως αυτός ο τρόπος εκτέλεσης επιλέχθηκε περισσότερο για λόγους μετατροπής των `data logs` σε απευθείας μορφή `json` που είναι η μορφή αποθήκευσης στη βάση δεδομένων MongoDB.

```
FROM alpine:latest

RUN apk add --no-cache nodejs npm

COPY . /usr/src/app

WORKDIR /usr/src/app
RUN npm install pm2 -g

EXPOSE 3000

CMD ["pm2-runtime", "ecosystem.config.js"]
```

Παρακάτω βλέπουμε το τελευταίο Dockerfile της εφαρμογής, που φτιάχνει την εικόνα του πειραματικού κοντέινερ. Είναι η ίδια λογική με την προηγούμενη του `readmongo_service`.

```
FROM alpine:latest

RUN apk add --no-cache nodejs npm

COPY . /usr/src/app

WORKDIR /usr/src/app
RUN npm install pm2 -g

EXPOSE 3000

CMD ["pm2-runtime", "ecosystem.config.js"]
```

Οι υπόλοιπες εικόνες από τις οποίες δημιουργούνται τα υπόλοιπα κοντέινερ λαμβάνονται από διαδικτυακά `docker hub`. Αυτό θα φανεί στα επόμενα υποκεφάλαια, που θα δούμε αναλυτικότερα το `docker-compose.yml`, μέσα στο οποίο δηλώνεται ο τρόπος με τον οποίο τραβάει ο χρήστης τις εικόνες από το διαδίκτυο.

### 5.3 ΠΕΡΙΕΚΤΕΣ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

Σε αυτό το κεφάλαιο θα δούμε λίγο αναλυτικότερα το κάθε κοντέινερ ξεχωριστά, και πως παραμετροποιήθηκε μέσω του `docker-compose.yml`. Σε επόμενο κεφάλαιο θα δούμε αναλυτικότερα την λειτουργία και τον κώδικα του κάθε κοντέινερ-υπηρεσία.

#### 5.3.1 MONGO

Το κοντέινερ `mongo` όπως αναφέρθηκε και νωρίτερα είναι το κοντέινερ το οποίο προσφέρει τις υπηρεσίες της βάσης δεδομένων της εφαρμογής. Η παραμετροποίηση της γίνεται μέσα στο `docker-compose.yml`.

```
1  version: "3.3"
2
3  services:
4    redisserver:
5      #image: hub.swarmlab.io:5480/playground-redisserver:latest
6      container_name: redisserver
7      depends_on:
8        - writetomongo
9      logging:
10       driver: "fluentd"
11       options:
12         fluentd-async-connect: "true"
13         tag: mongo.redis
14      image: redis:6.0.9-alpine
15      ports:
16        - "6379:6379"
17      networks:
18        playground-net:
19    writetomongo:
20      image: writemongo:latest
21      container_name: fluentd
22      user: root
23      build: ./WriteToMongo/fluent
24      volumes:
25        - ./WriteToMongo/fluent:/fluentd/etc
```

Εικόνα 15: Παράδειγμα αρχείου `docker-compose.yml`

Παραπάνω στην Εικόνα 15: Παράδειγμα αρχείου docker-compose.yml, βλέπουμε πως ένα τέτοιο αρχείο πρέπει να ξεκινάει με την δήλωση της έκδοσης του docker και στη συνέχεια να ακολουθεί η δήλωση της κάθε υπηρεσίας-κοντέινερ ξεχωριστά. Πιο συγκεκριμένα, στην Εικόνα 16: Δήλωση υπηρεσίας-κοντέινερ mongo, βλέπουμε την δήλωση της υπηρεσίας της βάσης δεδομένων (mongo).

```
92  mongo:
93    container_name: mongo
94    logging:
95      driver: "fluentd"
96      options:
97        fluentd-async-connect: "true"
98        tag: mongo.mongod
99    image: mongo
100    expose:
101      - 27017
102    ports:
103      - "27017:27017"
104    networks:
105      playground-net:
```

Εικόνα 16: Δήλωση υπηρεσίας-κοντέινερ mongo

Αναλυτικότερα για κάθε γραμμή:

- 92: Δηλώνεται το όνομα της υπηρεσίας.
- 93: Δηλώνεται το όνομα του κοντέινερ.
- 94: Δηλώνεται η έναρξη παραμετροποίησης του logging.
- 95: Δηλώνεται ο κατάλληλος οδηγός για το που θα στέλνονται τα logs, που παράγει η υπηρεσία.
- 98: Δηλώνεται το tag που θα προσθέτει η υπηρεσία στα log της κάθε φορά που θα παράγει. Αυτό βοηθάει στο να μπορεί ο χρήστης μετέπειτα να διακρίνει από ποια υπηρεσία παράγεται το κάθε log. Σημείωση, πως το tag είναι ιδιαίτερα σημαντικό και πρέπει ο χρή-

στης να δηλώνει με προσοχή την υπηρεσία του στο σύστημα για να γίνει σωστά η επεξεργασία των δεδομένων.

- 99: Δηλώνεται το όνομα της εικόνας από την οποία θα δημιουργηθεί το container. Αν αυτή η εικόνα δεν υπάρχει τοπικά, θα αναζητηθεί στο επίσημο docker hub, εκτός αν δηλωθεί κάποιο άλλο. Η συγκεκριμένη εικόνα την πρώτη φορά που θα τρέξει το αρχείο θα αναζητηθεί στο επίσημο docker hub, θα κατέβει και μετά θα δημιουργηθεί το container.
- 100,101,102,103: Γίνεται η δημοσίευση της πόρτας που απαιτείται, ώστε να μπορούν οι υπηρεσίες που είναι εξαρτώμενες από την ίδια να επικοινωνούν με αυτή, αλλά και η ίδια η μηχανή που φιλοξενεί το container.
- 104,105: Δήλωση έναρξης παραμετροποίησης του δικτύου στο οποίο θα ανήκει το κοντέινερ. Στη συγκεκριμένη περίπτωση δηλώνεται το playground-net, το οποίο όπως φαίνεται και στην εικόνα έχει δηλωθεί στο τέλος του αρχείου μαζί με τα volumes που θα εξηγηθούν αναλυτικά στη συνέχεια.

### 5.3.2 FLUENTD

Όπως ήδη αναφέρθηκε, το συγκεκριμένο κοντέινερ υλοποιεί την υπηρεσία του εργαλείου fluentd, το οποίο ευθύνη του είναι να συλλέξει, να στείλει και να αποθηκεύσει όλα τα logs των μικρουπηρεσιών στη βάση δεδομένων στην κατάλληλη μορφή. Στην παρακάτω Εικόνα 17: Παραμετροποίηση κοντέινερ fluentd βλέπουμε την παραμετροποίηση που έχει γίνει στο αρχείο docker-compose.yml για το συγκεκριμένο κοντέινερ.

```
19 writetomongo:
20   image: writemongo:latest
21   container_name: fluentd
22   user: root
23   build: ./WriteToMongo/fluent
24   volumes:
25     - ./WriteToMongo/fluent:/fluentd/etc
26   ports:
27     - "24224:24224"
28     - "24224:24224/udp"
29   networks:
30     playground-net:
```

Εικόνα 17: Παραμετροποίηση κοντέινερ fluentd

Οι νέες γραμμές κώδικα που παρατηρούμε σε σύγκριση με την παραμετροποίηση που είδαμε στα πλαίσια της MongoDB είναι στην γραμμή 23, που πλέον δηλώνουμε πού μπορεί να βρει το κατάλληλο dockerfile, ώστε να δημιουργηθεί η απαραίτητη εικόνα, σε περίπτωση που δεν υπάρχει, για να φτιαχτεί το ανάλογο κοντέινερ.

### 5.3.3 REDISSERVER

Η παραμετροποίηση αυτού του κοντέινερ φαίνεται στην Εικόνα 15: Παράδειγμα αρχείου docker-compose.yml που τοποθετήθηκε ως παράδειγμα νωρίτερα. Η ουσία αυτής της μικρουπηρεσίας θα αναλυθεί και θα εκτιμηθεί αργότερα, όταν θα δούμε την λειτουργικότητα της υπηρεσίας readmongo\_service. Εκεί θα δούμε, πώς το redis, ως κοινή μνήμη μας βοηθά να πετύχουμε επιθυμητά αποτελέσματα.

### 5.3.4 READMONGO\_SERVICE

Σε αυτό το κοντέινερ γίνεται στην ουσία η ανάλυση των δεδομένων που λαμβάνονται από την βάση δεδομένων, όπως μαρτυρά και το όνομα της υπηρεσίας, αλλά και η επικοινωνία με τον web-client, όπου στέλνονται τα δεδομένα για περαιτέρω επεξεργασία και απεικόνιση στην ιστοσελίδα. Το αντίστοιχο κοντέινερ φτιάχνεται φυσικά μέσω του docker-compose και φαίνεται στην Εικόνα 18: Παραμετροποίηση κοντέινερ readmongo\_service.

Εδώ αξίζει να σημειωθεί και να επεξηγηθεί η αξία του volumes που δηλώνεται στην γραμμή 45. Είναι κυρίως μία ρύθμιση που εξυπηρετεί τις ανάγκες του προγραμματιστή και έγκειται στο ερώτημα “Τι κάνω όταν χρειάζεται να κάνω αλλαγές στον κώδικα ενός κοντέινερ;”. Θεωρητικά θα έπρεπε κάθε φορά, που γίνεται μία αλλαγή στον κώδικα της υπηρεσίας, να αναδημιουργείται το κοντέινερ με τις νέες αλλαγές. Αυτό φυσικά αποτελεί μία πολύ κακή και χρονοβόρα πρακτική. Το volumes με απλά λόγια δημιουργεί μία σύνδεση μεταξύ των αρχείων εντός του κοντέινερ και αυτών του εξωτερικού φακέλου της υπηρεσίας που προγραμματίζεις. Έτσι, κάθε φορά που κάνεις μία αλλαγή στον κώδικα της υπηρεσίας, οι αλλαγές περνάν και εσωτερικά στο κοντέινερ

αφαιρώντας έτσι την ανάγκη της αναδημιουργίας του κοντέινερ κάθε φορά που γίνεται μία αλλαγή.

Επίσης, παρατηρούμε το tag του service, πως διαφέρει σε σχέση πχ με αυτό της mongo. Το readmongo\_service έχει γραφτεί με node.js, οπότε το tag του με τιμή mongo.node1 μαρτυρά ακριβώς αυτό. Η εφαρμογή στην παρούσα μορφή της μπορεί και αναγνωρίζει τρία διαφορετικά tags.

1. Αυτό του Redis (mongo.redis), που το δεύτερο τελευταίο μισό πρέπει να εμπεριέχει πάντα τη λέξη redis μέσα.
2. Αυτό της MongoDB βάσης δεδομένων, που το δεύτερο μισό πρέπει να εμπεριέχει πάντα τη λέξη mongoddb μέσα.
3. Τέλος, αυτό του Nodejs (mongo.node), που το δεύτερο τελευταίο μισό πρέπει να εμπεριέχει πάντα τη λέξη node μέσα.

```
31 readmongo:
32   container_name: readmongo_service
33   restart: always
34   depends_on:
35     - writetomongo
36   logging:
37     driver: "fluentd"
38     options:
39       fluentd-async-connect: "true"
40       tag: mongo.node2
41   build: ./readmongo
42   image: readmongo:latest
43   ports:
44     - "3000:3000"
45   volumes:
46     - ./readmongo:/usr/src/app
47   links:
48     - mongo
49   networks:
50     playground-net:
```

Εικόνα 18: Παραμετροποίηση κοντέινερ readmongo\_service

### 5.3.5 DUMMY\_SERVICE

Τελευταίο και λιγότερο σημαντικό από όλα τα υπόλοιπα είναι το κοντέινερ του `dummy_service` το οποίο φτιάχτηκε με την ίδια ακριβώς λογική όπως το `readmongo_service`, χωρίς βέβαια να εμπεριέχει την ίδια πολύπλοκη λειτουργικότητα όπως το δεύτερο.

Διαθέτοντας ένα απλό API, παράγει logs έπειτα από κλήση του χρήστη, ενώ αποσκοπεί στην παραγωγή “χαζών” logs για λόγους ελέγχου και κατανόησης της λειτουργικότητας ολόκληρης της εφαρμογής. Η παραμετροποίησή του φαίνεται παρακάτω στην Εικόνα 19: Παραμετροποίηση του κοντέινερ `dummy_service`.

Επίσης, παρατηρούμε αυτό που προ-αναφέρθηκε για το tag. Η συγκεκριμένη υπηρεσία είναι γραμμένη σε `node.js` οπότε και το tag της είναι κατάλληλα γραμμένο, ώστε να μπορεί να γίνει κατανοητό κατά την επεξεργασία από τι είδους υπηρεσία προέρχεται το log. Αυτό είναι ιδιαίτερα σημαντικό για παράδειγμα στην διαδικασία εξακρίβωσης αν ένα log είναι τύπου `error` ή `out`.

```
71 dummy:
72   container_name: dummy_service
73   restart: always
74   depends_on:
75     - writetomongo
76   logging:
77     driver: "fluentd"
78     options:
79       fluentd-async-connect: "true"
80       tag: mongo.node1
81   build: ./dummyservice
82   image: dummyservice:latest
83   ports:
84     - "3001:3001"
85   volumes:
86     - ./dummyservice:/usr/src/app
87   links:
88     - mongo
89   networks:
90     playground-net:
```

Εικόνα 19: Παραμετροποίηση του κοντέινερ `dummy_service`



## 5.4 ΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑ-ΑΝΑΛΥΣΗ ΥΠΗΡΕΣΙΩΝ

Σε αυτό το κεφάλαιο θα εμβαθύνουμε και θα αναλύσουμε κάθε κομμάτι της εφαρμογής, πώς λειτουργεί, θα δούμε κομμάτια κώδικα που συντέλεσαν στο τελικό αποτέλεσμα, τη σκέψη πίσω από κάθε απόφαση που πάρθηκε, αλλά και τις δυνατότητες που προσφέρει σαν τελική ολότητα στην ανάγκη της συλλογής, παρακολούθησης και ανάλυσης δεδομένων.

Αρχικά, θα πρέπει να σημειωθεί, πως δεν θα υπάρξει υποκεφάλαιο σχετικό με τη βάση δεδομένων ξεχωριστά και το redis, αφού δεν χρειάστηκε να κάνω κάποια εσωτερική αλλαγή στα συγκεκριμένα κοντέινερ. Κάθε παραμετροποίηση και χρήση ορίστηκε από άλλες υπηρεσίες, ενώ για παράδειγμα, η παραμετροποίηση του fluentd που θα δούμε πρώτο στο επόμενο υποκεφάλαιο, χρειάστηκε να γίνει εσωτερικά με ειδικό αρχείο.

### 5.4.1 FLUENTD

Στο προηγούμενο κεφάλαιο είδαμε τις παραμετροποιήσεις που αφορούσαν την υλοποίηση των κοντέινερ μέσω του docker-compose. Σε αυτό το σημείο θα δούμε τι γίνεται μέσα σε κάθε κοντέινερ. Το Fluentd είναι από μόνο του ένα εργαλείο που προσφέρει πάρα πολλές δυνατότητες παραμετροποίησης μέσω του δικού του documentation. Στην συγκεκριμένη περίπτωση υλοποίησης το configuration file με όνομα fluent.conf έχει ως εξής:

```
1. <source>
2.   @type forward
3.   port 24224
4.   bind 0.0.0.0
5. </source>
6.
7. <filter mongo.redis>
8.   @type record_transformer
9.   <record>
10.     gen_host "#{Socket.gethostname}"
11.   </record>
12. </filter>
13.
14. <filter mongo.*>
15.   @type record_transformer
```

```
16.     <record>
17.     tag ${tag}
18.     </record>
19.     </filter>
20.
21.     <match mongo.*>
22.     @type copy
23.     <store>
24.     @type stdout
25.     </store>
26.     <store>
27.     @type mongo
28.
29.     host mongo
30.     port 27017
31.
32.     database fluentdb
33.     collection test
34.
35.     capped
36.     capped_size 1024m
37.
38.     <buffer>
39.     flush_interval 10s
40.     </buffer>
41.     <format>
42.     @type json
43.     </format>
44.     </store>
45.     # interval
46.     </match>
```

#### 5.4.1.1 Επεξήγηση παραμετροποιήσεων

- 1-5: Δηλώνεται από που να δέχεται δεδομένα το εργαλείο. Με αυτήν την παράμετρο που δόθηκε όλη η ροή δεδομένων καταγραφής του δικτύου λαμβάνεται από το fluentd, εφόσον η εκάστοτε υπηρεσία έχει δηλώσει το fluentd ως logging driver στην παραμετροποίησης που έχει γίνει στο docker-compose.yml.
- 7-12: Περνάει όλα τα logs που έχουν παραχθεί από το redis μέσα από ένα φίλτρο το οποίο προσθέτει το hostname στη json δομή του log.

- 14-19: Προσθέτει στη json δομή του log το tag που έχει ορίσει ο χρήστης στο docker container του.
- 21-46: Για όλα τα logs που εμπεριέχουν το tag “mongo.\*” αρχικά, τυπώνει τα δεδομένα στο stdout (για λόγους testing), στέλνει τα δεδομένα σε μία βάση δεδομένων που παραμετροποιείται ως database:fluentd, με collection test, με cap size τα 1024m (γιατί η mongoddb δεν είναι σχεδιασμένη για μεγάλους όγκους δεδομένων). Επίσης, δηλώνει ένα buffer ο οποίος κρατάει για δέκα δευτερόλεπτα όλα τα δεδομένα καταγραφής και τα στέλνει μαζικά στη βάση δεδομένων, ενώ αμέσως μετά τα σβήνει και για κάθε επόμενη δέκα δευτερόλεπτα επαναλαμβάνει την ίδια διαδικασία. Αυτή η επαναληπτική διαδικασία έγινε για να μην υπάρχει υπερφόρτωση του δικτύου, διότι η ροή των δεδομένων σε ένα περιβάλλον με ιδιαίτερα μεγάλη κίνηση, μπορεί να γίνει τεράστια ως αποτέλεσμα να διογκώνεται το πρόβλημα με τη συνεχή αποστολή logs στην βάση δεδομένων. Τέλος, μετατρέπει κάθε συμβάν σε μορφή json, ώστε να γίνεται πιο γρήγορα και εύκολα η διαδικασία ανάλυσης του κάθε log στην υπηρεσία που διαβάζει τα logs από τη βάση δεδομένων.

Αναφορικά με το Fluentd δεν υπάρχει κάτι άλλο που μπορεί να ειπωθεί για να δείξει σε μεγαλύτερη κλίμακα τη λειτουργικότητά του, αφού η φύση του εργαλείου είναι να είναι απλό στη χρήση, αλλά ταυτόχρονα να παρέχει δυνατότητες κομβικής σημασίας στη συλλογή και προώθηση των δεδομένων.

#### 5.4.2 READMONGO\_SERVICE ΚΑΙ WEB-CLIENT

Σε αυτό το επίπεδο της εφαρμογής γίνονται οι περισσότερες εργασίες. Όπως αναφέρθηκε και νωρίτερα το readmongo\_service είναι γραμμένο σε node.js και εξυπηρετεί ως μεσολαβητής ανάμεσα στην βάση δεδομένων και τον web-client που βλέπει ο χρήστης. Εν ολίγοις, εκτός το ότι αντλεί τα ανάλογα δεδομένα από τη MongoDB και τα στέλνει κατάλληλα μορφοποιημένα προς τον web-client, επίσης αναλαμβάνει και τη δουλειά της αυθεντικοποίησης του κάθε αιτήματος από τον web-client. Όπως θα εξηγηθεί και παρακάτω, με τη χρήση ενός token μπορεί η εφαρμογή να πιστοποιήσει την ταυτότητα του χρήστη. Κάθε φορά, που ένα instance του web-client προσπαθεί να συνδεθεί στις υπηρεσίες που προσφέρει το readmongo\_service, μία διαδικα-

σία αυθεντικοποίησης πυροδοτείται μέσω socket, που κρίνει αν ο χρήστης έχει το δικαίωμα να λάβει αυτές τις υπηρεσίες ή όχι. Στα της φιλοσοφίας των συγκεκριμένων υπηρεσιών, θα ακολουθήσουμε βήμα-βήμα κάθε σημαντικό κομμάτι την εφαρμογής, ώστε να γίνει κατανοητή η προσφορά τους στο σύνολο της λειτουργικότητας. Δεν θα γίνει εμβάθυνση πίσω από κάθε διαφορετικό αντικείμενο ή βιβλιοθήκη που χρησιμοποιήθηκε, παρά μόνο στη λογική πίσω από κάθε γενική λειτουργία της εφαρμογής. Παρακάτω ωστόσο, μπορούμε να δούμε λίγο από τον κώδικα του readmono\_service που δηλώνει κάποια από τα πακέτα λογισμικού που χρειάστηκαν κατά την υλοποίηση.

```
var pathmodule = require("path");
var app = require("express")();
var http = require("http").Server(app);
var https = require("https");
var CONFIG = require(pathmodule.resolve(__dirname, "runconfig.js"));
const io = require("socket.io")(http, {
  cors: {
    origin: "http://localhost:8080",
    methods: ["GET", "POST"],
    allowedHeaders: ["my-custom-header"],
    credentials: true,
  },
  cookie: {
    name: "test",
    httpOnly: false,
    path: "/custom",
  },
});

const createAdapter = require("socket.io-redis");

const Redis = require("ioredis");

const pubClient = new Redis({
  host: "redisserver",
  port: 6379,
});

const subClient = pubClient.duplicate();

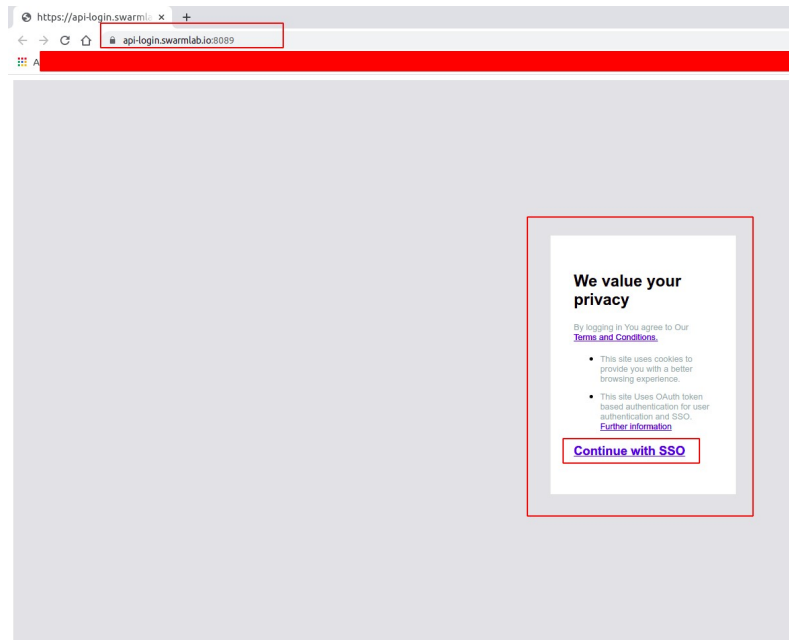
io.adapter(createAdapter({ pubClient, subClient }));
```

```
pubClient.on("connect", function () {  
  console.log("You are now connected");  
});
```

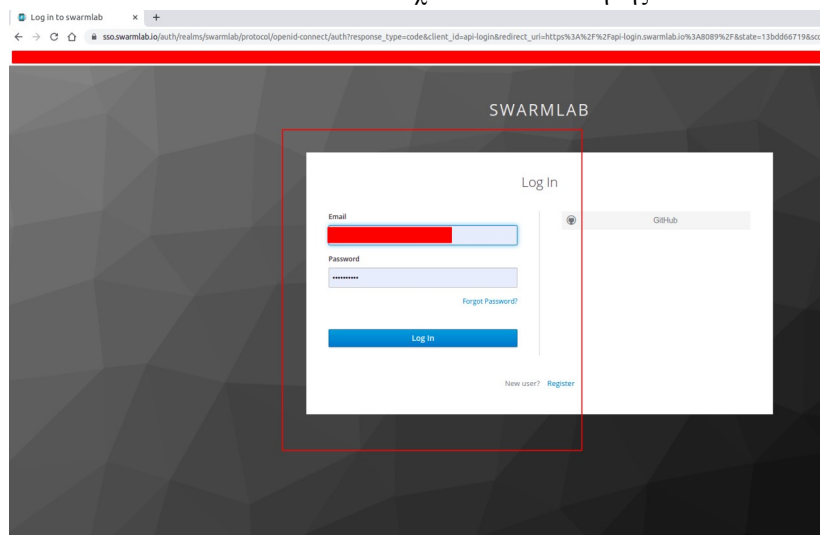
Φυσικά ο κώδικας του readmongo εκτείνεται σε μία κλίμακα των χιλίων γραμμών, οπότε για ευνόητους λόγους θα γίνονται μονάχα τέτοιου είδους αναφορές. Αναφορικά με τι βλέπουμε παραπάνω, αρχικά βλέπουμε πως γίνεται χρήση του express και των sockets που προαναφέρθηκαν, ενώ στη συνέχεια δημιουργείται η σύνδεση με την υπηρεσία του Redis.

#### 5.4.2.1 Αυθεντικοποίηση

Ξεκινώντας από τον web-client, αρχικά θα πρέπει να τονιστεί πως έχει χτιστεί βάση κάποιων ήδη υπάρχουσών προδιαγραφών. Δηλαδή, η αυθεντικοποίηση του χρήστη όταν εισέρχεται στο web interface της σελίδας γίνεται μέσω ενός token το οποίο στέλνεται σε απομακρυσμένο API βασισμένο στο cloud cluster του swarmlab.io. Εν ολίγοις, για να έχει πρόσβαση κάποιος στην εφαρμογή μέσω του web-client, υπάρχει ρητή απαίτηση το url του browser την παρούσα στιγμή να κουβαλάει το κατάλληλο token. Στις παρακάτω εικόνες βλέπουμε την διαδρομή ενός χρήστη που απέτυχε να αυθεντικοποιηθεί από το API του swarmlab.io, οπότε στέλνετε να συνδεθεί με τα στοιχεία του ώστε να λάβει το απαραίτητο token. Πιο αναλυτικά, στην Εικόνα 20 βλέπουμε την σελίδα στην οποία η εφαρμογή σε κάνει redirect, σε περίπτωση που δεν καταφέρει να κάνει authentication με το token. Στην Εικόνα 21 βλέπουμε την διαδικασία σύνδεσης με τα απαραίτητα στοιχεία στην ιστοσελίδα του swarmlab.io ενώ στην Εικόνα 22, στην γραμμή του url παρατηρούμε, πως πλέον ο χρήστης έχει αποκτήσει το απαραίτητο token, που θα του δώσει το δικαίωμα να εισέλθει στην εφαρμογή μας.



Εικόνα 20: Αποτυχία αυθεντικοποίησης



Εικόνα 21: Διαδικασία σύνδεσης στο swarmlab.io



Εικόνα 22: Απόκτηση token

```
created() {  
  
  var logintoken = new URL(location.href).searchParams.get("token");  
  this.logintoken = logintoken;  
  // === We get the user + check for the token if exists  
  this.checktoken(this.logintoken);  
  var log = store.dispatch("pipelineLLO/settoken", {  
    token: logintoken,  
  });  
  
  this.socketopen();  
},
```

Στο παραπάνω κομμάτι κώδικα του web-client γίνεται ακριβώς αυτό που περιγράψαμε παραπάνω, δηλαδή λαμβάνεται το token και ύστερα στέλνεται μέσω κατάλληλου API για αυθεντικοποίηση.

Εφόσον ο χρήστης περάσει από το στάδιο της αυθεντικοποίησης, περνάει πλέον στην διεπαφή του χρήστη η οποία φαίνεται παρακάτω στην Εικόνα 23: Διεπαφή web-client. Σε αυτό το στάδιο δίνονται στον χρήστη κάποιες δυνατότητες, τις οποίες θα εξερευνήσουμε αναλυτικά μία προς μία. Ξεκινώντας από τα αριστερά της διεπαφής όπως βλέπουμε και στην Εικόνα 24: Λειτουργία ευρετηρίου, υπάρχει μία ένδειξη που αναγράφει “online”. Αυτή η ένδειξη μαρτυρά ότι η επικοινωνία με τον server είναι ανοιχτή και μπορεί ο web client να λάβει δεδομένα δίχως πρόβλημα. Σε αντίθετη περίπτωση θα υπήρχε μία ένδειξη “socket disconnected” ενώ τα υπόλοιπα buttons που φαίνονται τώρα θα απουσίαζαν, στερώντας έτσι τον χρήστη από την διαδικασία να κάνει άσκοπες κινήσεις.

#### 5.4.2.2 Ευρετηρίαση

Στην Εικόνα 24: Λειτουργία ευρετηρίου, βλέπουμε επίσης και την δυνατότητα ευρετηρίασης που παρέχεται από την εφαρμογή. Όπως είχαμε αναφέρει και στο κεφάλαιο με τις βέλτιστες πρακτικές παρακολούθησης, η δυνατότητα ευρετηρίασης δεν θα μπορούσε να απουσιάζει. Έτσι, με τη χρήση αυτής της απλής διεπαφής, η εφαρμογή μπορεί να αντλήσει είτε όλα τα διαθέσιμα logs που είναι αποθηκευμένα στη βάση δεδομένων, είτε να ψάξει κάποιο συγκεκριμένου περιεχομένου, είτε κάποιου συγκεκριμένου container είτε έναν συνδυασμό των παραπάνω. Έτσι στην Εικόνα 25: Αποτέλεσμα ευρετηρίασης βλέπουμε το αποτέλεσμα μίας τέτοιας αναζήτησης, που

ψάχνει τα logs που περιέχουν την λέξη “socket” στο κοντέινερ readmongo\_service. Η δυνατότητα αυτή έχει στηριχθεί πάνω στο εργαλείο vuetable-2, το οποίο δίνει την δυνατότητα σελιδοποίησης των αποτελεσμάτων που τυπώνει, ενώ ταυτόχρονα δίνει μεγάλη ελευθερία στον τρόπο που μπορεί κάποιος να απεικονίσει τα αποτελέσματα. Όπως μπορούμε να δούμε τα error logs απεικονίζονται με έντονο κόκκινο χρώμα, ώστε να είναι ευδιάκριτα στον παρατηρητή. Αντίστοιχα, όλα τα υπόλοιπα logs εμφανίζονται με πράσινο χρώμα. Η κατηγοριοποίηση των logs σε error, είναι μία διαδικασία που διαφέρει από υπηρεσία σε υπηρεσία. Για παράδειγμα, στην υπηρεσία της MongoDB, έπρεπε να μελετηθεί αναλυτικά το documentation, ώστε να αναλύονται τα logs με συγκεκριμένο τρόπο, για να γίνει κατανοητό αν πρόκειται για error log ή απλά για log ομαλής λειτουργικότητας

Σε επίπεδο υλοποίησης, η διαδικασία της ευρετηρίασης επιτυγχάνεται με τη χρήση API που είναι προγραμματισμένο στο readmongo\_service. Το συγκεκριμένο API αντλεί όλα τα δεδομένα από τη βάση δεδομένων και ύστερα εκτελεί την κατάλληλη ανάλυση με βάση τις επιλογές του χρήστη, που έχουν αποσταλεί από τον web-client. Πιο συγκεκριμένα, σε επίπεδο κώδικα παρακάτω βλέπουμε το σημείο που ο web-client καλεί το API του back-end, το οποίο call συμβαίνει μέσω του εργαλείου vuetable-2, μίας έτοιμης βιβλιοθήκης για την απεικόνιση δεδομένων σε πίνακες. Επίσης, μπορούμε να δούμε και παραμετροποιήσεις σχετικές με το pagination των δεδομένων.

```
<vuetable
ref="vuetable"
:api-url="api_url"
:api-mode="true"
:fields="fields"
:item-actions="itemActions"
:sort-order="sortOrder"
:show-sort-icons="true"
:multi-sort="multiSort"
:per-page="perpage"
:append-params="extraparams"
table-height="300px"
pagination-path="links.pagination"
detail-row-id="id"
wrapper-class="vuetable-wrapper"
loading-class="loading"
```



```
@vuetable:pagination-data="onPaginationData"  
@vuetable:load-success="loadsuccess"  
@vuetable:load-error="onLoadError"  
:css="css.table"  
>  
</vuetable>
```

Η κλίση προς το api γίνεται την στιγμή που ο πίνακας γίνεται render στην διεπαφή του χρήστη. Μέσο του button που λέει search στην ουσία το μόνο που κάνουμε είναι να ανανεώνουμε τις απαραίτητες μεταβλητές, με τις νέες επιλογές του χρήστη και να κάνουμε refresh τον πίνακα, ώστε να πυροδοτείται η διαδικασία του api call. Έτσι ανανεώνεται και το περιεχόμενο του πίνακα. Παρακάτω βλέπουμε τη ρουτίνα που εκτελείται όταν ο χρήστης πατάει το search button της ευρετηρίασης. Εδώ αξίζει να αναφερθεί πως η κλήση της συνάρτησης `getServices`, εκτελεί ένα άλλο API call, το οποίο αναλύει το σύνολο των logs και ξεχωρίζει τα διάφορα services που υπάρχουν στο δίκτυο. Η τελική λίστα είναι αυτή που δίνεται και στον χρήστη τελικά για να επιλέξει από ποια υπηρεσία θέλει να απεικονίσει τα logs του. \*Αυτή η ανανέωση ιδανικά θα έπρεπε να συμβαίνει πιο αυτοματοποιημένα και αποτελεί επέκταση της εφαρμογής.

```
showHistory() {  
  this.items = [];  
  this.getServices();  
  this.extraparams.logtext = this.searchparam;  
  this.extraparams.selected = this.selected;  
  if (this.history == false) {  
    this.history = true;  
  } else {  
    this.$refs.vuetable.refresh();  
  }  
},
```

Το `api_url` που είδαμε στο `vuetable` παραπάνω είναι στην ουσία το url που καλεί στον server και στην προκειμένη περίπτωση είναι το `api_url: "http://localhost:3000/test2"`. Αυτό το service έχει αναπτυχθεί στο `readmono_service` και μπορούμε να δούμε ένα σημείο από τον κώδικά του παρακάτω. Στην ουσία λαμβάνει και αρχίζει να επεξεργάζεται όλες τις απαραίτητες παραμέτρους για την ευερετηρίαση. Στην πορεία του κώδικα γίνεται και η σύνδεση με τη

βάση δεδομένων, που αναλύονται τα καταλλήλως δομημένα log και στέλνονται πίσω στον web-client σε κατάλληλη μορφή. Η διαδικασία που συλλέγονται τα απαραίτητα logs δεν φαίνεται, μιας και πρόκειται για μία συνηθισμένη αλγοριθμική υπόθεση επεξεργασίας json δεδομένων.

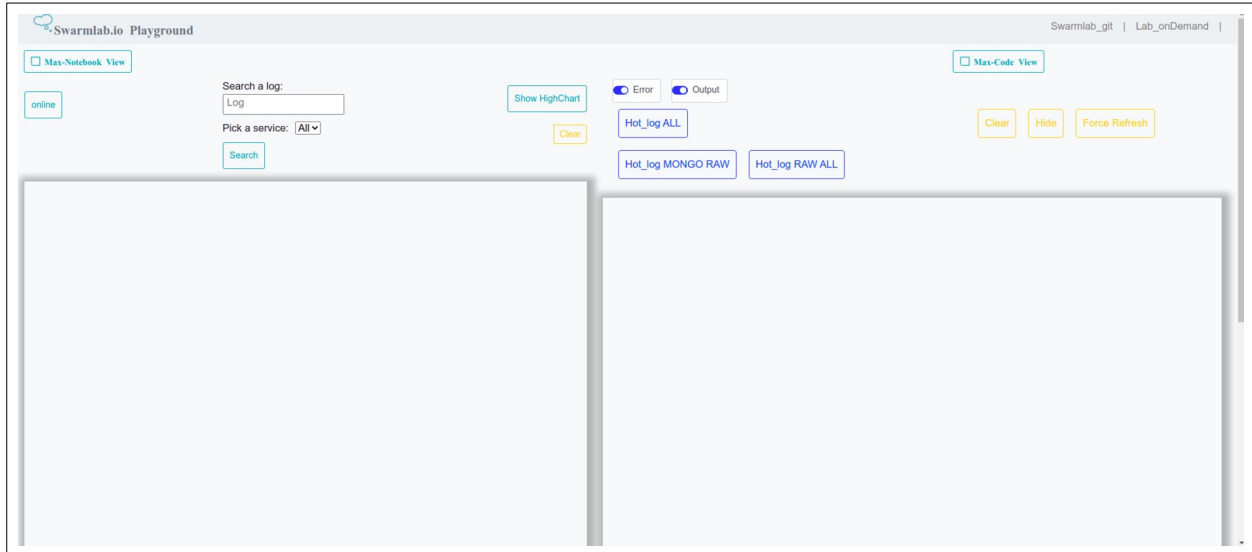
```
app.get("/test2", cors(corsOptions), (req, res) => {

  var RES = new Object();
  const page = req.query["page"];
  const per_page = req.query["per_page"];
  var sort = req.query["sort"];
  var filter = req.query["filter"];
  var type = req.query["type"];
  var sorttmp1 = sort.split("|");
  var sortname = sorttmp1[0];
  var sortorder = sorttmp1[1];

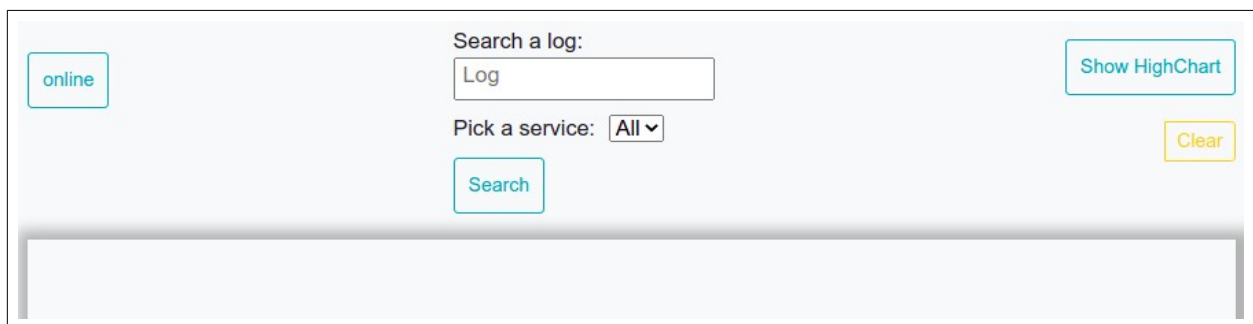
  // text to search in logs
  var logtext = req.query["logtext"];
  // service to choose from all containers
  var selected = req.query["selected"];

  console.error("EXTRA PARAMS: " + logtext);
  //console.log("TEST LOG");
  var url = "mongodb://mongo:27017/";
  var jsonfinal = [];
  MongoClient.connect(
    url,
    { useNewUrlParser: true, useUnifiedTopology: true },
    function (err, db) {
      if (err) throw err;
      var dbo = db.db("fluentdb");
      dbo
        .collection("test")
        .find({})
        .toArray(function (err, result) {
          if (err) throw err;
          // EPIDI EXW NESTED JSON PREPEI NA TO KANW PARSE DUO FORES
          var obj = JSON.parse(JSON.stringify(result));
```

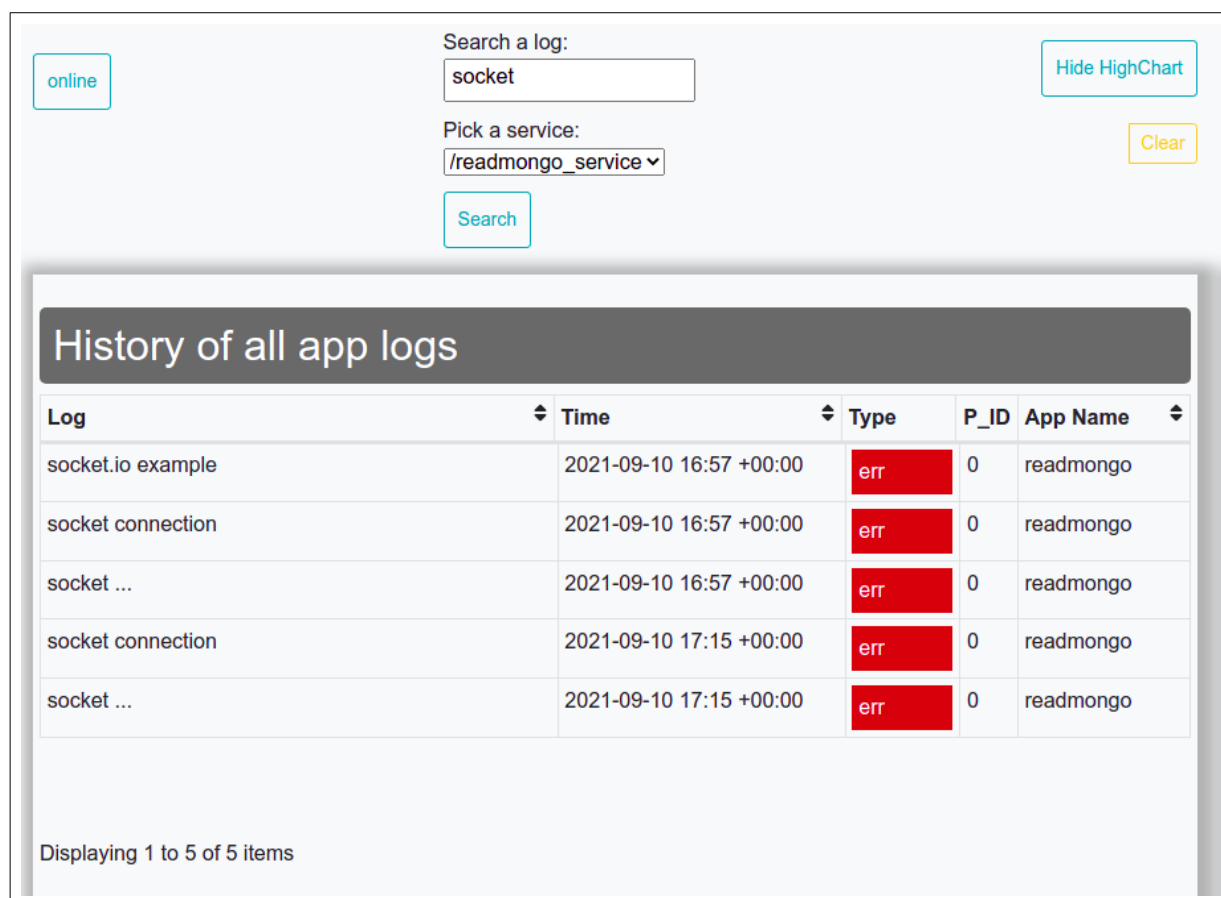
Παρακάτω βλέπουμε και τις τρεις εικόνες που αναφέρθηκαν νωρίτερα που μπορούμε να δούμε την διεπαφή σαν σύνολο, αλλά και μία περίπτωση ευρετηρίασης.



Εικόνα 23: Διεπαφή web-client



Εικόνα 24: Λειτουργία ευρετηρίου



Εικόνα 25: Αποτέλεσμα ευρετηρίασης

### 5.4.2.3 Διάγραμμα - Highchart

Στις εικόνες που παρατηρήσαμε και μελετήσαμε κατά την ευρετηρίαση, μπορούμε επίσης να παρατηρήσουμε πως υπάρχει ένα button που αναγράφει show/hide highchart. Το button αυτό δίνει την δυνατότητα στον χρήστη να εμφανίσει ένα διάγραμμα, κάτω ακριβώς από τον πίνακα αποτελεσμάτων της ευρετηρίασης, το οποίο δείχνει κάποια δεδομένα σχετικά με όλα τα container που τρέχουν στο εικονικό δίκτυο του docker. Όπως βλέπουμε και στην Εικόνα 26, εμφανίζονται δεδομένα σχετικά με το πλήθος των logs ανά κατηγορία και ανά container. Η λογική είναι πως ένας διαχειριστής μπορεί να δει μαζικά την υγεία του συστήματός του, δίχως να

χρειάζεται να βλέπει ένα-ένα τα container στην ευρετηρίαση. Ειδικά αν επρόκειτο για ένα δίκτυο με δεκάδες container ή και περισσότερα, η παρακολούθηση μέσω της ευρετηρίασης είναι τουλάχιστον κακή πρακτική. Στο highchart μπορεί να δει σε ένα ευέλικτο interface όλα τα container που τρέχουν στο δίκτυό του και τροφοδοτούν με δεδομένα το fluentd. Αριστερά βλέπει τα logs ομαλής λειτουργικότητας ενώ στα δεξιά βλέπει τα error logs που έχουν ανιχνευτεί κατά την ανάλυση. Έτσι, θα μπορούσε μέσω αυτής της λειτουργικότητας, να εντοπίσει γρήγορα κάποιο σφάλμα στο σύστημα ή κάποια προδιάθεση για κατάρρευση και μέσω της ευρετηρίασης να ψάξει και να μελετήσει το πρόβλημα. Επιπλέον, επιλέγοντας με τον κέρσορα τα ονόματα των κοντέινερ που φαίνονται στο κάτω μέρος του διαγράμματος, ο χρήστης μπορεί να ακυρώσει προσωρινά την απεικόνιση των πληροφοριών των logs κάποιου/κάποιων κοντέινερ, ενώ με τον ίδιο τρόπο να τα ενεργοποιήσει ξανά. Τέλος, το συγκεκριμένο highchart δίνει την δυνατότητα της αποθήκευσης του στιγμιότυπου την παρούσα στιγμή του διαγράμματος, σε μορφή εικόνας (png,jpeg,svg), ενώ μπορεί να αποθηκευτεί και σε pdf ή να γίνει απευθείας εκτύπωση.

Εδώ θα πρέπει να σημειωθεί πως το συγκεκριμένο component λειτουργεί συνεργατικά για την απόκτηση των δεδομένων με το readmongo\_service μέσω ειδικού rest API call, που προγραμματίστηκε για αυτό ακριβώς το σκοπό. Η απεικόνιση των δεδομένων επιδέχονται παραμετροποιήσεις ως ένα βαθμό, αλλά υπάρχουν αρκετοί περιορισμοί, μιας και πρόκειται για σχεδόν έτοιμο εργαλείο, όπως ο περιορισμός της μη αυτόματης ανανέωσης του περιεχομένου του διαγράμματος. Για την επίλυση αυτού του προβλήματος έχει τοποθετηθεί και η λειτουργία refresh πάνω αριστερά, ώστε να μπορεί ο χρήστης χειροκίνητα να ανανεώνει το περιεχόμενο του highchart. \*Πιθανή επέκταση θα ήταν η αυτόματη εκτέλεση της ρουτίνας που εκτελεί το button refresh ανά κάποια δευτερόλεπτα.

Παρακάτω βλέπουμε τον κώδικα στον web-client που εμπεριέχει το ίδιο το διάγραμμα όσο και το button που κάνει την ανανέωση του περιεχομένου.

```
<div class="highch" v-show="chartstatus">
<h2>HighChart with log sum Data (all)</h2>
<div class="input-group-append">
<button
type="button"
round
class="btn btn-outline-warning btn-sm">
```

```
@click="callback(1) "  
>  
Refresh  
</button>  
</div>  
<highcharts  
class="hc"  
:options="chartOptions"  
:callback="callback"  
ref="chart"  
></highcharts>  
</div>
```

Επίσης, παρατηρούμε μία callback συνάρτηση. Μέσα σε αυτήν γίνεται η απαραίτητη επεξεργασία ώστε να εμφανιστούν τα δεδομένα στο διάγραμμα. Η συγκεκριμένη δραστηριότητα λειτουργεί εντελώς δυναμικά. Με το που εισαχθεί μία νέα υπηρεσία στο δίκτυο θα μπορούμε να δούμε κατευθείαν τα logs της στο διάγραμμα με ένα refresh. Όπως φαίνεται και στον παρακάτω κώδικα που εκτελείται στον web-client, τα δεδομένα που παρέχονται στο διάγραμμα έρχονται από μία κλήση σε rest API service που είναι στημένο στο readmongo\_service.

```
async callback(params) {  
  var res = await axios.get("http://localhost:3000/length");  
  //new code  
  if (params == 1) {  
    this.chartOptions.series.splice(0, this.chartOptions.series.length);  
    res["data"].forEach((val) => {  
      this.chartOptions.series.push({  
        name: val.name,  
        data: [val.lengtho, val.lengthe],  
      });  
    });  
  } else {  
    console.log(res["data"]);  
    res["data"].forEach((val) => {  
      this.chartOptions.series.push({  
        name: val.name,  
        data: [val.lengtho, val.lengthe],  
      });  
    });  
  }  
}
```

```
}  
},
```

Παρακάτω, μπορούμε να δούμε απόσπασμα από το API service (/length) του readmongo\_service, που χρησιμοποιήθηκε παραπάνω στον web-client. Ουσιαστικά εδώ βλέπουμε, που ελέγχει ένα προς ένα όλα τα logs και δημιουργεί το τελικό array που στέλνει στον web-client.

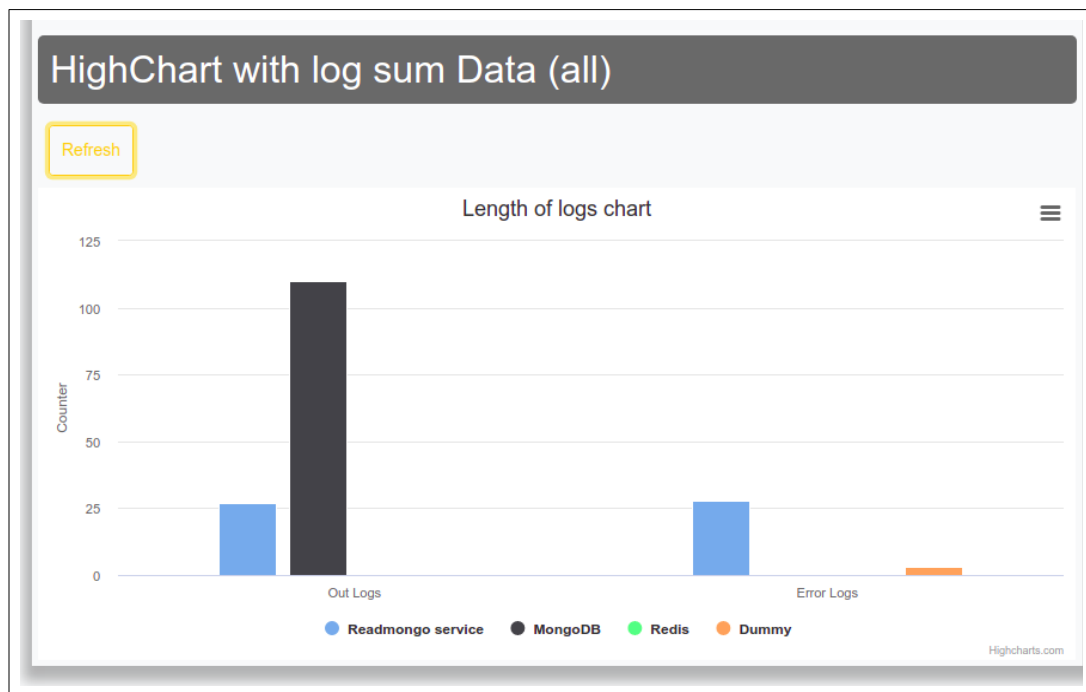
```
obj.forEach((value) => {  
  //new code  
  if (containers.length == 0) {  
    type = checkService();  
    if (type == "out") {  
      containers.push({  
        name: value.container_name,  
        lengtho: 1,  
        lengthe: 0,  
      });  
    } else {  
      containers.push({  
        name: value.container_name,  
        lengtho: 0,  
        lengthe: 1,  
      });  
    }  
  } else {  
    containers.forEach((val) => {  
      if (val.name == value.container_name) {  
        found = 1;  
        type = checkService();  
        if ((type == "out")) {  
          val.lengtho++;  
        } else val.lengthe++;  
      }  
    });  
    if (found == 0) {  
      type = checkService();  
      if (type == "out") {  
        containers.push({  
          name: value.container_name,
```

```
lengtho: 1,
lengthe: 0,
});
} else {
containers.push({
name: value.container_name,
lengtho: 0,
lengthe: 1,
});
}
}
found = 0;
}
```

Η συνάρτηση `checkService()` που βλέπουμε να εκτελείται παραπάνω, υλοποιεί έναν έλεγχο για κάθε log ξεχωριστά με βάση το tag του, που έχει δηλωθεί στο `docker-compose` για κάθε υπηρεσία και μετά από ανάλυση του log καταλήγει στο αν το log είναι out ή error. Παρακάτω μπορούμε να δούμε αναλυτικά και τον κώδικά της. \*Αυτό χωράει πάρα πολλές επεκτάσεις για τις διάφορες περιπτώσεις κάθε πιθανής υπηρεσίας και για κάθε διαφορετική γλώσσα προγραμματισμού.

```
function checkService() {
if (value.tag.includes("mongodb")) {
var tmp = JSON.parse(value.log);
if (tmp.s == "I") return "out";
// count every informative log as an output log
else if (tmp.s == "W" || tmp.s == "E") return "error"; //count every
warning and error log
} else if (value.tag.includes("nodejs")) {
var test = JSON.parse(value.log);
var type = test.type;
if (type == "err") return "error";
else return "out";
} else if (value.tag.includes("redis")) {
return "out";
}
}
```





Εικόνα 26: Highchart

#### 5.4.2.4 On-event Logs

Επιστρέφοντας στην Εικόνα 23, στα δεξιά του interface του web-client παρατηρούμε κάποιες άλλες λειτουργίες. Εδώ πρέπει να διευκρινισθεί πως όσα δεδομένα απεικονίζονται στο δεξί σημείο του interface, είναι γεγονότα που συμβαίνουν live στο δίκτυο, για αυτό τα αποκαλούμε και on-event δεδομένα. Επιγραμματικά, αυτά που βλέπουμε είναι:

- **Error – Output switch component:** Όπως γίνεται κατανοητό είναι η επιλογή που δίνεται στον χρήστη να απεικονίζονται μόνο συγκεκριμένη κατηγορία από logs (error ή out). ΣΗΜΕΙΩΣΗ: το συγκεκριμένο component λειτουργεί μονάχα στους on-event πίνακες και όχι στον πίνακα ευρετηρίασης που είδαμε νωρίτερα. \*Πιθανή επέκταση θα ήταν να μπει η ίδια λειτουργία και στην ευρετηρίαση.
- **Button – Hot\_log ALL:** Button που εμφανίζει πίνακα, ο οποίος απεικονίζει όλα τα on-event logs σε σχετικά απλοποιημένη μορφή. Η απεικόνιση είναι σχετικά απλή και μοιάζει πολύ με αυτή του πίνακα ευρετηρίασης. Τα error logs απεικονίζονται με κόκκινο,

ενώ τα υπόλοιπα με πράσινο. Απουσιάζει η λειτουργία σελιδοποίησης, αλλά έχει ρυθμιστεί το overflow μέσω scrolling στον πίνακα. Στην Εικόνα 27 βλέπουμε τον συγκεκριμένο πίνακα.

- **Button – Hot\_log MONGO RAW:** Button που εμφανίζει πίνακα, ο οποίος απεικονίζει όλα τα on-event logs της MongoDB (αν υπάρχει στο δίκτυο) σε Raw μορφή. Στην προκειμένη περίπτωση απεικονίζει τα logs της βάσης δεδομένων που χρησιμοποιείται από την ίδια την εφαρμογή, αλλά αυτό γίνεται μόνο για να φανεί η λειτουργικότητα. Σε ένα ρεαλιστικό πλαίσιο, εκεί θα έβλεπε κάποιος μία άλλη βάση δεδομένων που θα έτρεχε στο δίκτυο. Η λογική πίσω από αυτόν τον πίνακα είναι η εξής: Τα logs που παράγονται από τη MongoDB είναι πολλά, παράγονται ταχύτατα, ενώ είναι αρκετά μεγάλα και πολύπλοκα, για να μπορέσουν να γίνουν κατανοητά στον προηγούμενο, απλοποιημένης μορφής, πίνακα. Έτσι, αυτός ο πίνακας προσφέρει την απεικόνιση αυτών των logs σε raw μορφή, δηλαδή έτσι ακριβώς όπως παράγονται από την ίδια την βάση, έτσι ώστε ο διαχειριστής να μπορεί να επικεντρωθεί μόνο σε αυτά και να παρακολουθήσει κάποιο τυχόν πρόβλημα. Στην Εικόνα 28 βλέπουμε τον συγκεκριμένο πίνακα.
- **Button – Hot\_log RAWALL:** Button που εμφανίζει πίνακα, ο οποίος απεικονίζει τα πάντα on-event σε raw μορφή όπως έρχονται κατευθείαν από το stream στο fluentd. Ο πίνακας αυτός εξυπηρετεί ως εναλλακτική του πρώτου, για κάποιον που θέλει να δει την πλήρη μορφή του log. Ο πίνακας προσφέρει κάποιες επιπρόσθετες λεπτομέρειες όπως: το container\_id και το container\_name του container που παράγει το κάθε log. Στην Εικόνα 29 βλέπουμε τον συγκεκριμένο πίνακα.
- **Clear:** Καθαρίζει τα περιεχόμενα όλων των πινάκων. Βοηθάει στην περίπτωση που έχει μαζευτεί μεγάλος όγκος δεδομένων και αρχίζει να καθυστερεί ο browser.
- **Force-Refresh:** Εξαναγκάζει την ανανέωση των περιεχομένων των πινάκων σε περίπτωση που ο χρήστης αλλάξει την κατηγορία πχ μόνο σε error logs. Το refresh θα γίνει έτσι και αλλιώς αυτόματα όταν ληφθούν νέα δεδομένα στον browser, αλλά επειδή αυτό μπορεί να καθυστερήσει μερικά δευτερόλεπτα ανάλογα με την κίνηση στο δίκτυο, το button εξυπηρετεί ως μία ταχύτερη προσέγγιση.
- **Hide:** Κρύβει όλους τους πίνακες.

Σε επίπεδο κώδικα, αυτό που συμβαίνει είναι, πως με το άνοιγμα του client, εκπέμπεται ένα event προς τον server, που μετά από κάποιους απαραίτητους ελέγχους ξεκινά την σύνδεση του session με τη βάση δεδομένων, ώστε να αρχίσει ο server να στέλνει δεδομένα στον client. Παρακάτω βλέπουμε το event που πυροδοτεί αυτή ακριβώς την διαδικασία.

```
s.on("onevent", function (data) {  
  //console.log("I GOT THE DATA: ", data);  
  var binddata = {  
    user: data,  
    id: s.id,  
  };  
  checkstream(binddata);  
});
```

Κατά την συνάρτηση checkstream γίνεται έλεγχος για το κατά πόσο υπάρχει χρήστης με τρέχον ενεργό stream με τη βάση. Εφόσον υπάρχει δεν γίνεται τίποτα. Αυτό συμβαίνει σε περίπτωση που ο χρήστης δύο παράθυρα της ίδιας εφαρμογής. Αν υπήρξαν πάνω από δύο stream για κάθε χρήστη, τότε θα βλέπαμε πολλαπλές φορές τα ίδια logs σε κάθε διεπαφή. Στην περίπτωση που δεν υπάρχει, καλείται η onCollectionNew ως callback και δημιουργείται το απαραίτητο stream.

```
async function checkstream(data) {  
  var res = await getKey(data.id);  
  if (res == "1") {  
    console.log("Stream is on!");  
  } else {  
    console.log("Creating Stream...");  
  
    var url = "mongodb://mongo:27017/";  
    MongoClient.connect(  
      url,  
      { useNewUrlParser: true, useUnifiedTopology: true },  
      function (err, db) {  
        if (err) throw err;  
        var dbo = db.db("fluentdb");  
        dbo.collection("test", onCollectionNew.bind(data));  
      }  
    );  
  }  
}
```

```
}  
}
```

Επίσης, παρατηρούμε την συνάρτηση `getKey` η οποία είναι κομβικής σημασίας για τον έλεγχο του stream με τη βάση. Όπως μπορούμε να δούμε και παρακάτω, η διαδικασία ελέγχου έγκειται στην ύπαρξη του `redis` που χρησιμοποιούμε σαν κοινή `global` μνήμη. Έτσι μπορούμε να ελέγξουμε αν υπάρχει ενεργή σύνδεση κάποιου χρήστη χωρίς να περιοριζόμαστε μονάχα μέσα σε ένα `container`.

```
async function getKey(id) {  
  return new Promise((resolve) => {  
    pubClient.get(id, function (err, reply) {  
      if (err) {  
        console.log("-----error-----");  
  
        resolve(null);  
      } else {  
        if (reply) {  
          //console.log("-----fount-----");  
          resolve(1);  
        } else {  
          console.log("-----not fount-----");  
          resolve(2);  
          //return 2  
        }  
      }  
    });  
  });  
}
```

Η συνάρτηση στην οποία αρχικά εισάγεται ο χρήστης στο `redis` φαίνεται παρακάτω μέσα στην `callback` που αναφέρθηκε νωρίτερα. Πρόκειται για την συνάρτηση `setUser` της οποίας ο κώδικας βρίσκεται ακριβώς κάτω από αυτόν της `onCollectionNew`. Η `onCollectionNew`, πρόκειται για μία συνάρτηση που τρέχει μέχρι να κλείσει το stream με τον χρήστη, δηλαδή μόλις ο χρήστης τερματίσει την σύνδεση του socket του με τον server. Όταν τερματίσει το `socket connection` στην ουσία αυτό που συμβαίνει είναι να διαγραφεί από τον `redis` η καταχώρηση της ενεργής σύνδεσης, έτσι η `onCollectionNew` αποφασίζει να τερματίσει το stream με τη βάση.

```
async function onCollectionNew(err, collection) {  
  
  /*  
  Prepei na elegxw kathe fora an to socket id tou user einai energo  
  wste na mhn diathreita zwnthan h callback kai lamvanw duplicate  
  data ston client  
  */  
  let options = {  
    tailable: true,  
    awaitdata: true,  
    numberOfRetries: -1,  
    tailableRetryInterval: 500,  
  };  
  var cursor = collection.find({}, options).stream();  
  var itemsProcessed = 0;  
  var room = this.user;  
  var sid = this.id;  
  console.log("Inside callback: " + room + " Id: " + sid);  
  // LEFOS --- STORE USER IN REDIS  
  var rep = setUser(sid, room);  
  
  cursor.on("data", async function (data) {  
    cursor.pause();  
    var res = await getKey(sid);  
  
    if (res == "1") {  
      cursor.resume();  
      var obj = JSON.parse(JSON.stringify(data));  
      io.in(room).emit("logsend", obj);  
    } else if (res == "2") {  
      cursor.resume();  
      console.log("Cursor is closing...");  
      cursor.close();  
    }  
  });  
}  
  
var setUser = function setus(id, user) {  
  return new Promise((resolve) => {  
    //pubClient.set(key,value, 'EX', expire, function(err,reply){  
    pubClient.set(id, user, function (err, reply) {  
      if (err) {  
        resolve(null);  
      }  
    });  
  });  
}
```

```

} else {
  resolve(reply);
}
});
});
};

```

Παρακάτω μπορούμε να δούμε μία προς μία και τις εικόνες που αναφέρθηκαν πριν, που δείχνουν το τελικό αποτέλεσμα στον web-client. Τα δεδομένα που στέλνονται μέσω του stream με τον server, λαμβάνουν μία τυπική επεξεργασία μέσω του vue.js και απεικονίζονται στους παρακάτω πίνακες.

ON-event logs for all apps (Simplified - ALL)				
<a href="#">Scroll to Bottom</a>				
socket GET from client undefined	2021-09-11 12:58 +00:00	err	0	dummy
socket GET from client undefined	2021-09-11 12:58 +00:00	err	0	dummy
socket GET from client undefined	2021-09-11 12:58 +00:00	err	0	dummy
WiredTiger message	2021-09-11T12:58:49.628+00:00	out	Unknown	Mongo
Connection accepted	2021-09-11T12:58:50.026+00:00	out	Unknown	Mongo
client metadata	2021-09-11T12:58:50.028+00:00	out	Unknown	Mongo
Connection accepted	2021-09-	out	Unknown	Mongo

Εικόνα 27: Πίνακας on-event απλοποιημένης μορφής

ON-event logs for MongoDB only (RAW)				
<a href="#">Scroll to Bottom</a>				
{ "\$date": "2021-09-11T12:58:26.195+00:00", "type": "NETWORK", "source": "listener", "conn": "conn19", "doc": { "driver": { "name": "nodejs", "version": "3.6.5" }, "os": { "type": "Linux", "name": "linux", "architecture": "x64", "version": "5.11.0-27-generic" }, "platform": "Node.js v14.16.1, LE (unified)" } }				
{ "\$date": "2021-09-11T12:58:26.195+00:00", "type": "NETWORK", "source": "listener", "conn": "conn20", "doc": { "driver": { "name": "nodejs", "version": "3.6.5" }, "os": { "type": "Linux", "name": "linux", "architecture": "x64", "version": "5.11.0-27-generic" }, "platform": "Node.js v14.16.1, LE (unified)" } }				
{ "\$date": "2021-09-11T12:58:26.195+00:00", "type": "NETWORK", "source": "conn20", "conn": "conn20", "doc": { "driver": { "name": "nodejs", "version": "3.6.5" }, "os": { "type": "Linux", "name": "linux", "architecture": "x64", "version": "5.11.0-27-generic" }, "platform": "Node.js v14.16.1, LE (unified)" } }				

Εικόνα 28: Πίνακας on-event raw logs της MongoDB

ON-event logs (RAW - ALL)				
Container_id	Log	Container Name	Source	Time
2eb1f5addad...	{"message": "cs141082@uniwa.gr created \n", "timestamp": "2021-09-11 12:58 +00:00", "type": "out", "process_id": 0, "app_name": "readmongo"}	/readmongo_service	stdout	2021-09-11T12:58:29.000Z
595fab75a4b8fd9b70fefee4536e851883bb780439b34a1e6d94fd0a84cf9cf3		service	stderr	2021-09-11T12:58:35.000Z
595fab75a4b...	{"message": "socket GET from client undefined\n", "timestamp": "2021-09-11 12:58 +00:00", "type": "err", "process_id": 0, "app_name": "dummy"}	/dummy_service	stderr	2021-09-11T12:58:41.000Z
595fab75a4b...	{"message": "socket GET from client undefined\n", "timestamp": "2021-09-11 12:58 +00:00", "type": "err", "process_id": 0, "app_name": "dummy"}	/dummy_service	stderr	2021-09-11T12:58:41.000Z

Εικόνα 29: Πίνακας on-event raw logs

Άξιος αναφοράς και ανάλυσης είναι ο ρόλος του Redis ως μία καθολική κοινή μνήμη. Όπως αναφέρθηκε ήδη παραπάνω, κάθε stream που δημιουργείται με τη βάση είναι ένας χρήστης που έχει ζητήσει σύνδεση με τον web-client. Δηλώνοντας στην κοινή μνήμη του redis το αναγνωριστικό id του socket session του χρήστη και της εκάστοτε σύνδεσης του με τη βάση, μπορούμε να στέλνουμε δεδομένα στον καθένα, δίχως να υπάρχουν διπλότυπες αποστολές ή αποτελέσματα. Επίσης, στην περίπτωση που αναπτυχθεί η εφαρμογή σε σμήνος, κάθε κοντέινερ θα ξέρει μέσω του redis ποιοι είναι οι ενεργοί χρήστες χωρίς η πληροφορία να είναι αποθηκευμένη σε κάθε κοντέινερ ξεχωριστά.

Φυσικά, άμεσα δεν σημαίνει ακαριαία. Για λόγους υγείας του δικτύου, όπως αναφέρθηκε και κατά την παραμετροποίηση του fluentd, η ενημέρωση της βάσης γίνεται κάθε δέκα δευτερόλεπτα, άρα μία μικρή καθυστέρηση θα υπάρχει πάντα. Όχι αρκετή βέβαια για να μην μπορεί ο χρήστης να παρακολουθήσει την τρέχουσα ροή του δικτύου αποτελεσματικά.

### 5.4.3 ΣΕΝΑΡΙΟ ΧΡΗΣΗΣ

Όπως αναφέρθηκε και νωρίτερα, εντός του δικτύου υπάρχει ένα κοντέινερ που ονομάζεται dummieservice. Το συγκεκριμένο κοντέινερ δεν βοηθάει σε καμία λειτουργικότητα της εφαρμογής, αλλά θα μας βοηθήσει να καταλάβουμε, πώς μπορεί κάποιος να εισάγει τα δικά του κοντέινερ στο δίκτυο της εφαρμογής και να δει τα logs του. Το μόνο που χρειάζεται, είναι ένα πρόγραμμα μέσα σε μια docker εικόνα και να δημιουργηθεί κατάλληλα ένα container αυτής της εικόνας. Αυτό μπορεί να επιτευχθεί με δύο τρόπος. Είτε εισάγοντας τα κατάλληλα στοιχεία στο αρχείο docker-compose της εφαρμογής, πριν τρέξει όλη η εφαρμογή, είτε εκτελώντας χειροκίνητα μέσω γραμμής εντολών την εντολή docker run, για κάθε κοντέινερ ξεχωριστά, με τις κατάλληλες παραμέτρους. Σε ένα σενάριο που η εφαρμογή βρίσκεται στο ίδιο φυσικό δίκτυο με τα κοντέινερ που χρίζουν παρακολούθησης, η εντολή θα ήταν η εξής:

```
docker run --network="diploma_playground-net" --log-driver=fluentd --log-opt tag="mongo.node3" -p 3001:3001 dummieservice
```

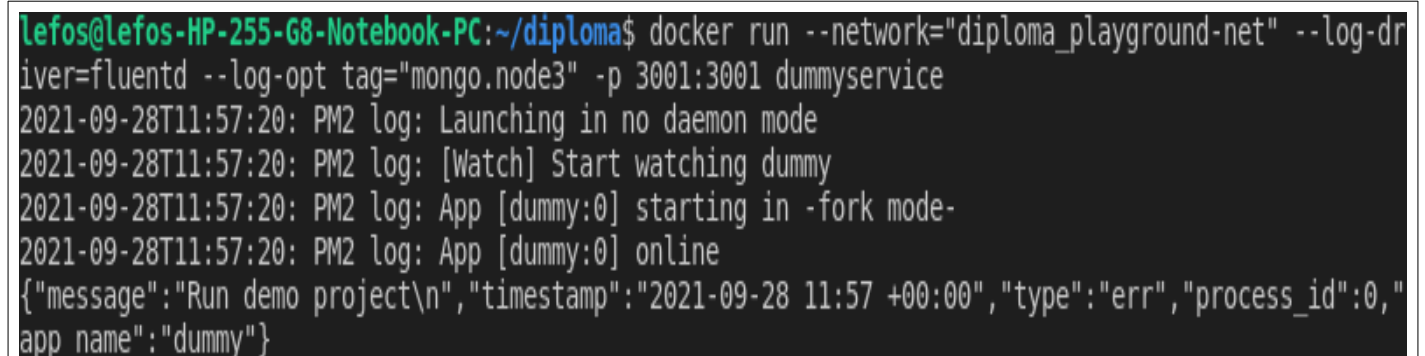
Ουσιαστικά η εντολή εκτελεί περίπου ότι κάνει και το docker-compose για ένα συγκεκριμένο κοντέινερ. Σημείωση: η εντολή αυτή τρέχει, αφού έχει δημιουργηθεί πρώτα το docker image του ανάλογου κοντέινερ. Εισάγει λοιπόν το κοντέινερ στο κατάλληλο δίκτυο (diploma\_playground-



net), ορίζει τον κατάλληλο driver (fluentd), ώστε να στέλνονται τα δεδομένα στο fluentd, ορίζει κατάλληλο tag (mongo.node3), κάνει expose την κατάλληλη πόρτα (3001) και τέλος, δηλώνει το όνομα της docker εικόνας από την οποία θα φτιαχτεί το container. Εκτελώντας αυτήν την εντολή δημιουργείται εκ των υστέρων ένα κοντέινερ μέσα στο δίκτυο της εφαρμογής από το οποίο πλέον μπορείς να παρακολουθήσεις τα ανάλογα logs. Παρακάτω στην Εικόνα 30 βλέπουμε ένα παράδειγμα της εκτέλεσης αυτής της εντολής. Το nostalgic\_galois στην Εικόνα 31 είναι το όνομα που έλαβε το κοντέινερ αυτόματα, μιας και δεν δήλωσα εγώ κάποιο συγκεκριμένα.

Σε διαφορετικό σενάριο, σε ένα ήδη υπάρχων μεγάλο docker δίκτυο μίας εταιρίας, ή εφαρμογή θα μπορούσε να εκτελεστεί μέσω του docker-compose, αλλάζοντας το όνομα του δικτύου σε αυτό της εταιρίας, εντός του αρχείου docker-compose.yml, και να παραμετροποιηθούν τα ήδη υπάρχοντα κοντέινερ ώστε να στέλνουν τα logs τους στο fluentd.

Ιδανικά, μία τέτοια υπηρεσία θα μπορούσε να γίνεται deploy στο cloud πχ μέσα σε VMs και μέσω λογικής edge computing (για την μείωση του delay) να προσφέρεται σε εταιρίες, ώστε να παρακολουθούν τις dockerized υποδομές τους.



```
lefos@lefos-HP-255-G8-Notebook-PC:~/diploma$ docker run --network="diploma_playground-net" --log-driver=fluentd --log-opt tag="mongo.node3" -p 3001:3001 dummyservice
2021-09-28T11:57:20: PM2 log: Launching in no daemon mode
2021-09-28T11:57:20: PM2 log: [Watch] Start watching dummy
2021-09-28T11:57:20: PM2 log: App [dummy:0] starting in -fork mode-
2021-09-28T11:57:20: PM2 log: App [dummy:0] online
{"message":"Run demo project\n","timestamp":"2021-09-28 11:57 +00:00","type":"err","process_id":0,"app name":"dummy"}
```

Εικόνα 30: Εκτέλεση Εντολής

online

Search a log:

Show HighChart

Pick a service:

Clear

Search

### History of all app logs

Log	Time	Type	P_ID	App Name
Run demo project	2021-09-11 14:28 +00:00	err	0	dummy
listening on http://0.0.0.0:3000/	2021-09-11 14:28 +00:00	err	0	dummy
socket GET from client undefined	2021-09-11 14:29 +00:00	err	0	dummy
Hello World!	2021-09-11 14:28 +00:00	out	0	dummy

Εικόνα 31: Αποτέλεσμα εισαγωγής container



## 6 ΣΥΜΠΕΡΑΣΜΑΤΑ-ΕΠΕΚΤΑΣΕΙΣ

Ξεκινώντας από τα συμπεράσματα που έγιναν αντιληπτά έπειτα από την όλη μελέτη της θεωρίας, αλλά και της υλοποίησης της εφαρμογής, πρέπει να αναφέρω πως ο κόσμος του cloud είναι ένα πεδίο που μόλις αρχίσαμε να εξερευνούμε και η επεκτάσεις του είναι άνευ ορίων, όσο η τεχνολογία προοδεύει δίπλα του. Μέσο όλων των παραπάνω έγινε αντιληπτό πως η μονολιθική προσέγγιση των εφαρμογών μικρής, αλλά και μεγάλης κλίμακας, έχει σβήσει σχεδόν ολοκληρωτικά από τον κόσμο της πληροφορικής, δίνοντας έτσι έδαφος σε αυτό που λέμε cloud computing να γίνει νομοτέλεια στην ανάπτυξη του λογισμικού και της δικτύωσης.

Όπως αλλάζει η νοοτροπία και η αρχιτεκτονική αντίληψη της ανάπτυξης των διαφόρων εφαρμογών, έτσι αλλάζει και ο τρόπος παρακολούθησης, συλλογής και ανάλυσης των δεδομένων αυτών των έργων. Σε μία μονολιθική εποχή δεν υπήρχε τέτοια απαίτηση, αλλά πλέον, όταν μία εφαρμογή αποτελείται από δεκάδες ή και εκατοντάδες μικρο-υπηρεσίες, που η κάθε μία αποτελεί ένα κοντέινερ, η ανάγκη για μία πιο εξειδικευμένη διαχείριση όλων αυτών των δεδομένων γίνεται επιτακτική. Αυτό επιδιώκει να πετύχει και η εφαρμογή που υλοποιήθηκε. Το πετυχαίνει; Ως ένα βαθμό, ναι. Χωράει βελτιώσεις; Πάρα πολλές. Μία τέτοια εφαρμογή δεν δύναται να καλύψει όλες τις ανάγκες, που ιδανικά θα έπρεπε να καλύπτει, όταν προγραμματίζεται από μονάχα έναν άνθρωπο, αφού πρόκειται για υπηρεσία που θα χρειαζόταν ολόκληρη ομάδα εξειδικευμένων προγραμματιστών, για να φτάσει γρήγορα σε ένα εμπορικό και ιδανικό αποτέλεσμα. Ακόμα και τότε βέβαια, θα χρειαζόταν συνεχής αναβάθμιση και ενημέρωση. Εν κατακλείδι, η συλλογή και η ανάλυση δεδομένων, είναι το άλφα και το ωμέγα σε μικρά αλλά και μεγάλα συστήματα ενώ πρόκειται για μία υπηρεσία, που μόλις μάθεις να την χρησιμοποιείς, δεν γυρνάς ποτέ ξανά πίσω σε παλιές πρακτικές, διότι αντιλαμβάνεσαι, πως πλέον η ανάπτυξη των λογισμικών έχει μεταβεί σε ένα πλαίσιο, που τέτοιες εφαρμογές είναι απαραίτητες.

## 6.1 ΕΠΕΚΤΑΣΕΙΣ ΕΦΑΡΜΟΓΗΣ

Σε αυτό το σημείο, θα ήθελα να αναφέρω κάποιες πιθανές επεκτάσεις της εφαρμογής που υλοποιήθηκε, οι οποίες έγιναν αντιληπτές κατά την διάρκεια της υλοποίησης, αλλά δεν υπήρχε ο απαραίτητος χρόνος για υλοποίηση, ή ήταν επεκτάσεις που έφευγαν από τα πλαίσια της εργασίας. Κάποιες πιθανές επεκτάσεις έχουν σημειωθεί και κατά τη διάρκεια επεξήγησης της λειτουργικότητας της εφαρμογής.

- **Επέκταση των tags:** Η σημαντικότερη επέκταση κατά την γνώμη μου είναι να μπορεί η εφαρμογή να δίνει περισσότερες πληροφορίες αναφορικά με τα logs, ανάλογα από τι υπηρεσία προέρχονται και από ποια γλώσσα προγραμματισμού παράγονται. Ως τώρα η εφαρμογή μπορεί και κάνει ανάλυση logs από υπηρεσίες που προέρχονται από μία βάση δεδομένων MongoDB και από εφαρμογές γραμμένες σε Node.js. Φυσικά εδώ πρέπει να αναφερθεί, πως η εφαρμογή δίνει στον χρήστη όλα τα logs, ασχέτως προγραμματιστικής υποδομής, εφόσον τρέχουν μέσα σε docker container, αλλά για να μπορέσει να τα αναλύσει σε μεγάλο βαθμό, πρέπει να γνωρίζει την δομή του κάθε log, και αυτό τείνει να διαφέρει από υπηρεσία σε υπηρεσία και από τεχνολογία σε τεχνολογία. Με το Fluentd, μπορείς να πειράξεις το αρχικό log και να το διαμορφώσεις ως ένα επίπεδο, αλλά η ουσιαστική ανάλυση γίνεται, αφού μελετηθεί το documentation της εκάστοτε τεχνολογίας, όπως για παράδειγμα χρειάστηκε να γίνει ειδική επεξεργασία αναφορικά με τα logs της mongo, που διέφεραν εντελώς σε σχέση με αυτά που παρήγαγε το node.js.
- **Προσαρμογή της ιστοσελίδας για μικρότερα πλαίσια οθόνης:** Η εφαρμογή υλοποιήθηκε με βάση τη λογική, ότι θα απεικονίζεται σε 16:9 οθόνες, οπότε η έντονη αλλαγή αυτών των διαστάσεων καταστρέφουν ριζικά τον τρόπο που απεικονίζονται τα αποτελέσματα. Η λύση βρίσκεται ξεκάθαρα στο front-end κομμάτι της εφαρμογής που είναι υλοποιημένο σε vue.js, αλλά λόγω το ότι η λειτουργικότητα βρισκόταν σε προτεραιότητα, δεν δόθηκε η απαραίτητη σημασία σε αυτόν τον τομέα.
- **Δοκιμή της εφαρμογής σε περιβάλλον cloud:** Η εφαρμογή υλοποιήθηκε και δοκιμάστηκε μέσα στα πλαίσια ενός και μόνο υπολογιστή, αλλά υλοποιήθηκε με τη λογική να δουλεύει αποτελεσματικά και σε ένα περιβάλλον cloud. Δεν υπήρχε η δυνατότητα να γίνει μία τέτοια δοκιμή κατά την διάρκεια της εργασίας, αλλά σίγουρα η εκτέλεσή της σε

ένα τέτοιο οικοσύστημα θα είχε ιδιαίτερο ενδιαφέρον, αναφορικά με την αποδοτικότητα της.

- **Συναγερμός σε συγκεκριμένο log:** Μία λειτουργία που σίγουρα απαιτεί τεράστια βοήθεια για έναν διαχειριστή ενός μεγάλου συστήματος.
- **Επιπλέον κριτήρια ευρετηρίασης:** Η ευρετηρίαση απαιτεί πολύ σημαντικό κομμάτι της εφαρμογής, αλλά και γενικά της αναζήτησης δεδομένων, οπότε η προσθήκη επιπλέον λειτουργιών, όπως για παράδειγμα η λειτουργία αναζήτησης μόνο error ή out logs, μοιάζει μία καλή επέκταση.
- **Επέκταση σε εφαρμογή android:** Αναμφισβήτητα σημαντική θα ήταν μία τέτοια επέκταση για έναν διαχειριστή κάποιου σταματήματος, που θα μπορούσε οποιαδήποτε στιγμή να επιβλέπει το σύστημά του μέσω του κινητού του με ειδική εφαρμογή, δίχως να χρειάζεται η επίσκεψή του στην απαραίτητη ιστοσελίδα, αλλά ταυτόχρονα και η υλοποίηση της ίδιας της ιστοσελίδας με τρόπο, ώστε να απεικονίζεται κατάλληλα σε τέτοιες συσκευές. Σχετικά με την κατάλληλη απεικόνιση που αναφέρθηκε και νωρίτερα, θα μπορούσε κάλλιστα να επιτευχθεί και μέσω του Vue.js, που παρέχει αυτήν την δυνατότητα μέσω ειδικών σκελετών όπως αυτοί του BootstrapVue.

**ΒΙΒΛΙΟΓΡΑΦΙΑ**

- i.1 Kalliopi Kanaki, (2015), Υπολογιστικό Νέφος, Ηράκλειο, Ελλάδα
- i.2 Retrieved from [https://en.wikipedia.org/wiki/Cloud\\_computing#Early\\_history](https://en.wikipedia.org/wiki/Cloud_computing#Early_history)
- i.3 IBM Cloud Team, IBM Cloud, Top 7 Most Common Uses of Cloud Computing, (2020, July 20), retrieved from <https://www.ibm.com/cloud/blog/top-7-most-common-uses-of-cloud-computing>
- i.4 IBM Cloud Team, IBM Cloud, What is Cloud Computing, (2020, July 20), retrieved from <https://www.ibm.com/cloud/learn/cloud-computing>
- i.5 Wesley Chai, (2021), Software as a service (SaaS), <https://searchcloudcomputing.techtarget.com/definition/Software-as-a-Service>
- i.6 CloudFlare, (2021), What is Platform as a service?, retrieved from <https://www.cloudflare.com/learning/serverless/glossary/platform-as-a-service-paas/>
- i.7 Microsoft, (2021), What is IaaS, <https://azure.microsoft.com/en-us/overview/what-is-iaas/#overview>
- i.8 Amazon, (2021), Serverless on AWS, retrieved from <https://aws.amazon.com/serverless/>
- i.9 CloudFlare, (2021), Why use serverless computing, retrieved from <https://www.cloudflare.com/learning/serverless/why-use-serverless/>
- i.10 Stephen Guyton, (2019, May 24), Containers and Containerization – Pros and Cons, retrieved from <https://spin.atomicobject.com/2019/05/24/containerization-pros-cons/>

- i.11 David Ward, (2015, March 10), The pros and cons of using a virtualized machine, retrieved from <https://www.controleng.com/articles/the-pros-and-cons-of-using-a-virtualized-machine/>
- i.12 IDG Communications, Inc., The shift to Cloud Computing Persists as Organizations Use Multiple Public Clouds, (2020, June 18) retrieved from <https://www.globenewswire.com/news-release/2020/06/18/2050275/0/en/The-Shift-to-Cloud-Computing-Persists-as-Organizations-Use-Multiple-Public-Clouds.html>
- i.13 Aaron Strong, Containerization vs Virtualization: What's the Difference?, (2019, November 1), retrieved from <https://www.burwood.com/blog-archive/containerization-vs-virtualization>
- i.14 Frank Brown, Containers vs Virtualization, (2019, September 19) retrieved from <https://rancher.com/learning-paths/containers-vs-virtualization/>
- i.15 Pinal V Chauhan, (2012), Cloud Computing In Distributed System, International Journal of Engineering Research & Technology (IJERT), 1, pp.1
- i.16 IBM, What is distributed computing, (2021, July 3), retrieved from <https://www.ibm.com/docs/en/txseries/8.1.0?topic=overview-what-is-distributed-computing>
- i.17 POFFRINGA, Independent Software Providers and the Cloud Vendors, (2020, July 26), retrieved from <https://softwarestackinvesting.com/independent-software-providers-and-the-cloud-vendors>
- i.18 Jerry Hargoove,(2020, 26 July), A history of Amazon Web Services, retrieved from <https://www.awsgeek.com/AWS-History/>



- i.19 STAMFORD,(2019, July 1), Gartner says the future of the database Market is the cloud, <https://www.gartner.com/en/newsroom/press-releases/2019-07-01-gartner-says-the-future-of-the-database-market-is-the>
- i.20 SARA MAZER, Cloud neutrality: 5 things every CIO should plan for when moving to cloud, <https://www.marklogic.com/blog/cloud-neutrality-strategy/>
- i.21 Tulane University, (2021, September), What you need to know about decentralized social networks, retrieved from <https://sopa.tulane.edu/blog/decentralized-social-networks>
- i.22 MASTODON, (2021, 17 July), What is Mastodon?, <https://docs.joinmastodon.org/>
- i.23 Kenton Varda, (2017, 29 September), Introducing Cloudflare Workers: Run JavaScript Service Workers at the Edge, <https://blog.cloudflare.com/introducing-cloudflare-workers/>
- i.24 Stephen J. Bigelow, (2021, September), What is Edge Computing? Everything you need to know, retrieved from <https://searchdatacenter.techtarget.com/definition/edge-computing>
- i.25 Mathew Prince, (2020, 26 July), The Edge Computing Opportunity: It's Not What You Think, retrieved from <https://blog.cloudflare.com/cloudflare-workers-serverless-week/>
- i.26 Aaron Nordhoff, (2021, August), What is a cluster? An overview of Clustering in the Cloud, retrieved from <https://www.capitalone.com/tech/cloud/what-is-a-cluster/>
- i.27 Ivy Wingmore, (2016, January), Log management, retrieved from <https://searchitoperations.techtarget.com/definition/log-management>
- i.28 Appdynamics website, (2021, September), Logging vs Monitoring: Best practices for integration, retrieved from <https://www.appdynamics.com/product/how-it-works/application-analytics/log-analytics/monitoring-vs-logging-best-practices#~cloudops-vs-devops>

- i.29 Google, (2021, September), Cloud logging, retrieved from  
<https://cloud.google.com/logging>
- i.30 Docker Website, (2021, 28 August), Docker Overview, retrieved from  
<https://docs.docker.com/get-started/overview/>
- i.31 Nodejs Website, (2021, 28 August), About Node.js, retrieved from  
<https://nodejs.org/en/about/>
- i.32 Fluentd Website, (2021, 28 August), What is Fluentd?, retrieved from  
<https://www.fluentd.org/architecture>
- i.33 Fluentd Website, (2021, 28 August), Why use Fluentd?, retrieved from  
<https://www.fluentd.org/why>
- i.34 Vuejs Website, (2021, 28 August), What is Vue.js?, retrieved from  
<https://vuejs.org/v2/guide/>
- i.35 Socket.io Website, (2021, 28 August), What Socket.IO is?, retrieved from  
<https://socket.io/docs/v4>
- i.36 Redis.io Website, (2021, September), What is Redis?, retrieved from <https://redis.io/>
- i.37 MongoDB Website, (2021, September), What is MongoDB?, retrieved from  
<https://www.mongodb.com/what-is-mongodb>