ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός και ανάπτυξη ενός κατανεμημένου συστήματος για τη συλλογή, ανάλυση και αποθήκευση των δεδομένων χρήσης που παράγονται από τη λειτουργία ενός υπολογιστικού νέφους

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Σταύρου Δήμαρχου

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2014

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός και ανάπτυξη ενός κατανεμημένου συστήματος για τη συλλογή, ανάλυση και αποθήκευση των δεδομένων χρήσης που παράγονται από τη λειτουργία ενός υπολογιστικού νέφους

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Σταύρου Δήμαρχου

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 28η Ιουλίου 2014.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Νεκτάριος Κοζύρης Καθηγητής Ε.Μ.Π.	Δημήτριος Τσουμάκος Επίκουρος Καθηγητής Ιόνιο Πανεπιστήμιο	Γεώργιος Γκούμας Λέκτορας Ε.Μ.Π

Αθήνα, Ιούλιος 2014

(Υπογραφή)

.....

Σταύρος Δήμαρχος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Σκοπός της συγκεκριμένης διπλωματικής εργασίας είναι η δημιουργία ενός κατανεμημένου συστήματος για τη συλλογή των δεδομένων χρήσης (logs) που παράγονται από τη λειτουργία ενός υπολογιστικού νέφους και την αποθήκευσή τους σε κεντρική βάση. Για το σκοπό αυτό μελετούνται εργαλεία ανοιχτού κώδικα για centralized logging, όπως το Apache Flume, Apache Kafka, Logstash που αναλαμβάνουν τη συλλογή των δεδομένων από τους κόμβους, καθώς και συστήματα με κατανεμημένες τεχνικές αποθήκευσης σε συστοιχίες υπολογιστών όπως το Apache Hadoop και Elasticsearch.

Αναλύονται τα εργαλεία Elasticsearch, Logstash και Kibana (ELK stack) που επιλέχθηκαν για την δημιουργία του συστήματος ενώ στη συνέχεια παρουσιάζεται πρωτότυπο με την εφαρμογή τους στο Synnefo, μια open sourch cloud πλατφόρμα, η οποία χρησιμοποιείται στον ~Okeanos, ένα έργο που προσφέρει υπηρεσίες "Infrastructure as a service" (IaaS) στη Ελληνική ερευνητική και ακαδημαϊκή κοινότητα.

Λέξεις Κλειδιά

Logstash, Elasticsearch, Kibana, ELK stack, σύστημα παρακολούθησης (monitoring), δεδομένα χρήσης, centralized logging, ανάλυση σε πραγματικό χρόνο, Synnefo, Apache Kafka, Apache Flume, Apache Storm, Apache Hadoop

Abstract

The purpose of this thesis is the design and development of a distributed monitoring system for the log data derived from a cloud service provider. To achieve this, open source, distributed systems for centralized logging are examined, like Apache Flume, Apache Kafka and Logstash. These systems are responsible for the collection of logs from the physical nodes. Apache Hadoop and Elasticsearch which can be used for the storage of "big data", are also examined. Finally, the tools Elasticsearch, Logstash and Kibana are being selected for the development of the system and a prototype is created based on "Synnefo". Synnefo is an open source cloud management platform that is used in ~Okeanos, the GRNET cloud service, for the Greek research and academic community.

Keywords

Logstash, Elasticsearch, Kibana, ELK stack, monitoring system, logs, centralized logging, real-time processing, Synnefo, Apache Kafka, Apache Flume, Apache Storm, Apache Hadoop

Πίνακας Περιεχομένων

Πίνακας Εικόνων

Πίνακας Πινάκων

1

Εισαγωγή

1.1 Ανάγκη για centralized logging σε υπολογιστικά νέφη

Τα τελευταία χρόνια σημειώνεται σημαντική έρευνα και ανάπτυξη γύρω από τα υπολογιστικά νέφη, με αποτέλεσμα να έχουν γίνει αξιόπιστα και καλή λύση για πολλές υπηρεσίες που προσφέρονται μέσω διαδικτύου. Εταιρίες που εξυπηρετούν εκατομμύρια χρήστες όπως κοινωνικά δίκτυα (πχ Foursquare[1]) ή παροχείς video on demand όπως η Netflix[2] χρησιμοποιούν υπολογιστικά νέφη, πάνω στα οποία εγκαθιστούν τις υποδομές τους και παρέχουν τις υπηρεσίες τους.

Οι πάροχοι υπηρεσιών υπολογιστικών νεφών (cloud providers) διαθέτουν υπολογιστικά κέντρα με συστοιχίες από φυσικές μηχανές (physical machines), διασυνδεδεμένες σε τοπικά δίκτυα μεγάλης ταχύτητας (gigabit Ethernet). Μέσω του διαδικτύου, παρέχουν στους πελάτες τους υπολογιστική ισχύ χρησιμοποιώντας την τεχνολογία του virtualization, καθώς επίσης και υπηρεσίες αποθηκευτικού χώρου.

Λόγω των υψηλών απαιτήσεων των χρηστών, οι πάροχοι πρέπει να εξασφαλίζουν την απρόσκοπτη παροχή πόρων, καθώς και την ποιότητα των προσφερόμενων υπηρεσιών (QoS - Quality of Service). Σημαντικό ρόλο για να το πετύχουν αυτό, έχει η παρακολούθηση (monitoring) της κατάστασης των υπολογιστικών κέντρων μέσα από τα οποία προσφέρουν τις υπηρεσίες τους. Η χρήση της υποδομής δημιουργεί ένα μεγάλο όγκο δεδομένων χρήσης (logs) τα οποία μπορούν να αξιοποιηθούν. Χρησιμοποιώντας κατάλληλο monitoring σύστημα, είναι δυνατόν να εντοπίζονται γρήγορα οι βλάβες στις φυσικές μηχανές ή ακόμα και να αποφεύγονται εντελώς. Επίσης, γνωρίζοντας την κατάσταση λειτουργίας των υπολογιστικών κέντρων, είναι δυνατόν να επιτευχθεί η βέλτιστη αξιοποίηση της υποδομής των κέντρων.

Γενικά, τα δεδομένα χρήσης, είναι ένα κρίσιμο μέρος οποιουδήποτε συστήματος καθώς παρέχουν ζωτικής σημασίας πληροφορίες σχετικά με τις εφαρμογές που τρέχουν σε αυτό. Οι περισσότερες εφαρμογές δημιουργούν δεδομένα χρήσης τα οποία τις πλείστες φορές

[1] https://foursquare.com/
[2] https://www.netflix.com/global

καταγράφονται σε αρχεία. Στη περίπτωση που το σύστημα φιλοξενείται σε ένα μηχάνημα, τα αρχεία καταγραφής μπορούν να αξιοποιηθούν άμεσα, καθώς είναι προσβάσιμα πολύ εύκολα. Αντίθετα, όταν το σύστημα τρέχει σε υπολογιστικά κέντρα, όπως το σύστημα διαχείρισης ενός υπολογιστικού νέφους, η αξιοποίηση των αρχείων καταγραφής δεδομένων χρήσης είναι δύσκολη. Για παράδειγμα, η αναζήτηση ενός συγκεκριμένου σφάλματος σε χιλιάδες αρχεία που βρίσκονται αποθηκευμένα σε εκατοντάδες κόμβους με τη χρήση κλασικών εργαλείων, είναι μια χρονοβόρα διαδικασία που πολύ πιθανόν να μην έχει αποτέλεσμα. Μια κοινή προσέγγιση σε αυτό το ζήτημα είναι η ανάπτυξη ενός κεντρικού συστήματος καταγραφής, έτσι ώστε τα δεδομένα χρήσης που παράγονται σε κάθε κόμβο, να καταλήγουν σε κεντρική βάση. Η προσέγγιση αυτή είναι γνωστή ως "centralized logging".

Υπάρχουν αρκετές μέθοδοι που ακολουθούν τη βασική ιδέα του centralized logging. Ο μεγάλος όγκος των δεδομένων χρήσης και ο ρυθμός με τον οποίο παράγονται στα υπολογιστικά κέντρα απαιτεί την αναζήτηση λύσης μέσω κατανεμημένων συστημάτων, ώστε να είναι εφικτή η αποδοτική συγκέντρωση των δεδομένων.

1.2 Αντικείμενο διπλωματικής

Αντικείμενο της παρούσας διπλωματικής εργασίας, είναι η δημιουργία ενός κατανεμημένου συστήματος συλλογής και αποθήκευσης σε κεντρική βάση των δεδομένων χρήσης που παράγονται από τη λειτουργία ενός υπολογιστικού νέφους. Θα γίνει προσπάθεια ώστε χρησιμοποιώντας το σύστημα αυτό, να συλλέγονται ταυτόχρονα και στατιστικά στοιχεία απόδοσης του κάθε κόμβου, ώστε να παρέχεται ολική εικόνα της κατάστασης του κάθε κόμβου. Ζητούμενο είναι, το σύστημα να πραγματοποιεί μερική επεξεργασία των δεδομένων σε πραγματικό χρόνο (real-time), ώστε η αναζήτηση τους από τη βάση όπου θα καταλήγουν, να μπορεί να γίνεται άμεσα και πιο εύκολα.

Η ανάπτυξη του συστήματος θα γίνει με βάση την αρχιτεκτονική του centralized logging και τα εργαλεία που θα επιλεχθούν θα πρέπει να εξασφαλίζουν την κλιμάκωση του συστήματος και να είναι ανοιχτού κώδικα (open-source). Για την αποδοτική συλλογή των δεδομένων θα μελετηθούν κατανεμημένες τεχνικές συλλογής ροών δεδομένων, ενώ για την αποδοτική αποθήκευση τους, θα χρησιμοποιηθούν κατανεμημένες τεχνικές αποθήκευσης σε συστοιχίες υπολογιστών.

Στόχος της διπλωματικής εργασίας, μέσα από το σύστημα αυτό, είναι η δημιουργία ενός εργαλείου για τους διαχειριστές ενός υπολογιστικού νέφους, με το οποίο θα μπορούν σχεδόν σε πραγματικό χρόνο να αναζητάνε συγκεκριμένα δεδομένα χρήσης και να βλέπουν στατιστικά που αφορούν το υπολογιστικό νέφος.

Θα δημιουργηθεί πρωτότυπο του συστήματος με την εφαρμογή του στο Synnefo [1], μια πλατφόρμα διαχείρισης υπολογιστικού νέφους.

1.3 Οργάνωση κειμένου

Στην κεφάλαιο 2 παρουσιάζεται η γενική αρχιτεκτονική για centralized logging, ενώ στη συνέχεια δίνεται συνοπτική περιγραφή των πιο σημαντικών εργαλείων που αξιολογήθηκαν

καθώς και αρχιτεκτονικές συστημάτων με βάση αυτών. Ακολούθως, παρουσιάζεται η σκέψη που οδήγησε στην επιλογή των Elasticsearch, Logstash και Kibana για τη δημιουργία του συστήματος. Στο κεφάλαιο 3, παρουσιάζονται αναλυτικά τα τρία εργαλεία που επιλέχθηκαν και στο κεφάλαιο 4 γίνεται μελέτη κατανεμημένων τεχνικών συλλογής των δεδομένων χρησιμοποιώντας τα εργαλεία αυτά. Στο κεφάλαιο 5, παρουσιάζεται η cloud πλατφόρμα Synnefo και το πρωτότυπο που δημιουργήθηκε με σκοπό την εφαρμογή του centralized logging μέσω των Elasticsearch, Logstash και Kibana στο Synnefo. Στη συνέχεια, στο κεφάλαιο 6, παρουσιάζονται οι μετρήσει που έγιναν με σκοπό τη μελέτη των επιδόσεων του συστήματος του πρωτότυπου. Κατόπιν, στην ενότητα 7 παρουσιάζεται συνοπτική περιγραφή και τυχόν επεκτάσεις της εργασίας. Τέλος, στην ενότητα 8 αναγράφεται η βιβλιογραφία.

2

Κεφάλαιο 2

Σκοπός του κεφαλαίου είναι η παρουσίαση της έρευνας που έγινε σχετικά με το θέμα της διπλωματικής εργασίας. Αρχικά περιγράφεται η αρχιτεκτονική για centralized logging με αρκετές αναφορές σε διάφορα εργαλεία. Ακολουθεί συνοπτική περιγραφή των πιο σημαντικών εργαλείων που αξιολογήθηκαν ενώ στη συνέχεια δίνονται αρχιτεκτονικές συστημάτων με βάση αυτών των εργαλείων καθώς και οι σκέψεις που οδήγησαν στην τελική επιλογή.

2.1 Ανάλυση αρχιτεκτονικής Centralized Logging

Η γενική αρχιτεκτονική για centralized logging χωρίζεται σε διάφορα μέρη. Τα περισσότερα εργαλεία προσφέρουν μόνο κάποια από αυτά με αποτέλεσμα να χρειάζεται η χρήση πολλαπλών εργαλείων για τη λύση του προβλήματος.

Τα κυριότερα μέρη της αρχιτεκτονικής είναι η συλλογή, η μεταφορά, η αποθήκευση και τέλος η ανάλυση των δεδομένων χρήσης. Σε μερικές περιπτώσεις υπάρχει ανάγκη για παροχή προειδοποιήσεων (alerts), κάτι που προσφέρεται από μεμονωμένα εργαλεία ενώ υπάρχουν άλλα που σχεδιάζονται με σκοπό την παροχή της συγκεκριμένης υπηρεσίας αλλά δεν εξετάζονται στη συγκεκριμένη διπλωματική εργασία.

2.1.1 Συλλογή δεδομένων χρήσης - Collection

Οι εφαρμογές δημιουργούν δεδομένα χρήσης με διάφορους τρόπους, για παράδειγμα μέσω πρωτοκόλλου syslog ή αρχείων. Στην περίπτωση όπου το centralized logging πραγματοποιείται για εφαρμογές για τις οποίες υπάρχει ανάγκη από τους προγραμματιστές να έχουν πρόσβαση στα δεδομένα χρήσης άμεσα, ώστε να διαπιστωθεί ο λόγος κάποιου προβλήματος ή να λάβουν κάποια απόφαση, η συλλογή δεδομένων πρέπει να πραγματοποιείται σε πραγματικό χρόνο χωρίς καμία καθυστέρηση. Η διατήρηση αντιγράφων των δεδομένων (file replication) σε κεντρική βάση ανά διαστήματα δε μπορεί να αποτελέσει λύση σε τέτοιες περιπτώσεις. Αντίθετα, θα ήταν καλή προσέγγιση αν υπήρχε ανάγκη ανάλυσης των δεδομένων για στατιστικές έρευνες.

Τις περισσότερες φορές η συλλογή των δεδομένων πραγματοποιείται μέσω των εργαλείων που αναλαμβάνουν και την αποστολή τους από τους κόμβους.

2.1.2 Μεταφορά δεδομένων χρήσης - Transport

Η μεταφορά των δεδομένων χρήσης από συστοιχίες υπολογιστών αξιόπιστα και γρήγορα σε κεντρική βάση δημιούργησε την ανάγκη δημιουργίας μιας νέας κλάσης κατανεμημένων συστημάτων για αποτελεσματική μετάδοση των δεδομένων. Εργαλεία όπως το Scribe[3], Apache Flume [2], Heka[4], Logstash [3], Chukwa[5], fluentd[6], nsq[7] και Kafka [4] έχουν σχεδιαστεί για τη μεταφορά μεγάλου όγκου δεδομένων με αρχιτεκτονική κατανεμημένων συστημάτων. Παρά το γεγονός ότι όλα αυτά δίνουν λύση στο πρόβλημα μεταφοράς, εντούτοις υπάρχουν αρκετές διαφορές μεταξύ τους.

Συγκεκριμένα, το Scribe και το nsq παρέχουν APIs στους clients για την καταγραφή των δεδομένων χρήσης. Χρησιμοποιώντας αυτά τα APIs μειώνεται η καθυστέρηση και αυξάνεται η αξιοπιστία. Αυτή η μέθοδος είναι πιο αποδοτική στις περιπτώσεις που υπάρχει πρόσβαση στην εφαρμογή που δημιουργεί τα δεδομένα χρήσης ώστε να γίνουν οι κατάλληλες τροποποιήσεις.

Τα εργαλεία Logstash, Heka, fluentd και Flume παρέχουν μια σειρά από πηγές εισόδου για συλλογή των δεδομένων όπως για παράδειγμα το πρωτόκολλο Syslog από συγκεκριμένη θύρα, αλλά μπορούν να διαβάσουν δεδομένα χρήσης και από την πιο συνηθισμένη πηγή, τα αρχεία.

2.1.3 Αποθήκευση δεδομένων χρήσης - Storage

Η επιλογή του μέρους αποθήκευσης των δεδομένων καθορίζεται από τη διάρκεια για την οποία τα δεδομένα θα παραμένουν στη βάση και στο είδος της ανάλυσης που χρειάζεται να γίνει στα δεδομένα αυτά.

Στην περίπτωση που δεν απαιτείται άμεση ανάλυση των δεδομένων και αυτά θα παραμείνουν αποθηκευμένα για μεγάλο χρονικό διάστημα προτείνεται να χρησιμοποιηθούν cloud υπηρεσίες όπως το Amazon S3[8] και AWS Glacier[9] ή ακόμα και tape backup καθώς αποτελούν φτηνές λύσεις για μεγάλο όγκο δεδομένων. Στην περίπτωση όμως που το χρονικό διάστημα περιορίζεται σε βδομάδες ή μήνες η αποθήκευση σε κατανεμημένη βάση όπως το HDFS [5], Cassandra[10], MongoDB[11] και Elasticsearch [6] είναι μια σωστή επιλογή. Η αποθήκευση των δεδομένων στο HDFS ή στο Elasticsearch είναι πιο αποτελεσματική στην περίπτωση που απαιτείται διαδραστική ανάλυση των δεδομένων.

---

[3] https://github.com/facebookarchive/scribe
[4] https://github.com/mozilla-services/heka
[5] https://chukwa.apache.org/
[6] http://www.fluentd.org/
[7] http://nsq.io/
[8] http://aws.amazon.com/s3/
[9] http://aws.amazon.com/glacier/
[10] http://cassandra.apache.org/
[11] http://www.mongodb.org/

Στις περιπτώσεις που τα αρχεία χρήσης είναι πολύ μεγάλα, πράγμα που συμβαίνει πολύ σπάνια, μπορούν να επεξεργαστούν μόνο με batch processing. Η πιο συνήθης επιλογή είναι το Apache Hadoop [7].

2.1.4 Ανάλυση δεδομένων χρήση - Analyze

Μια συνηθισμένη μέθοδος μέχρι τώρα για την επεξεργασία και ανάλυση των δεδομένων χρήσης είναι το batch processing. Στην περίπτωση που τα δεδομένα αποθηκεύονται στο HDFS του Apache Hadoop, το Apache Hive [8] και το Apache Pig [9], δύο εργαλεία που προσφέρονται από τη πλατφόρμα του Hadoop, απλοποιούν την επεξεργασία και ανάλυση των δεδομένων χρήσης σε σχέση με τη χρήση MapReduce [10] εργασιών.

Στην περίπτωση όμως που χρειάζεται ανάλυση των δεδομένων μέσω διεπαφής χρήστη (user interface - UI), ιδανική προσέγγιση είναι η αποθήκευση των δεδομένων στο Elasticsearch και χρήση του Kibana 3 [11] ή του Graylog2 [12] για ανάλυση και το Logstash, το Heka ή κάποια άλλη εφαρμογή για parsing. Αυτή η προσέγγιση επιτρέπει διαδραστική πρόσβαση στα δεδομένα σε πραγματικό χρόνο.

2.2 Περιγραφή εργαλείων

2.2.1 Hadoop και HDFS

Το Hadoop είναι ένα framework ανοιχτού κώδικα που αναπτύσσεται από το Apache Software Foundation και υποστηρίζει κατανεμημένη επεξεργασία μεγάλου όγκου δεδομένων. Η ανάπτυξη του ξεκίνησε το 2005 ως μια εναλλακτική επιλογή ανοιχτού κώδικα του Google Map Reduce framework και το Google File System (GFS) στα οποία και βασίστηκε και έκτοτε αποτελεί ένα από τα καλύτερα κατανεμημένα συστήματα για batch processing.

Το Apache Hadoop framework αποτελείται από τα εξής τμήματα:

- Hadoop Common: Περιέχει βιβλιοθήκες και εργαλεία απαραίτητα για τη λειτουργία των υπόλοιπων τμημάτων.
- Hadoop Distributed File System (HDFS): Κατανεμημένο, κλιμακώσιμο (scalable) σύστημα αρχείων.
- Hadoop YARN [13]: Πλατφόρμα υπεύθυνη για τη διαχείριση των υπολογιστικών πόρων και υπεύθυνη για τη χρονοδρομολόγηση των εργασιών του χρήστη. Ενσωματώθηκε στο Hadoop framework στη έκδοση του Hadoop 2.0.
- Hadoop MapReduce: Προγραμματιστικό μοντέλο για την επεξεργασία μεγάλου όγκου δεδομένων.

Ένας από τους κυριότερους λόγους της επιτυχίας αυτού του framework είναι το γεγονός ότι σχεδιάστηκε να τρέχει σε συστάδες από απλούς υπολογιστές (clusters) χαμηλού κόστους όπου όμως η συχνότητα βλαβών είναι υψηλή. Οι βλάβες αντιμετωπίζονται αυτόματα από το framework εξασφαλίζοντας τη σταθερότητα του συστήματος, ενώ παράλληλα η διαθεσιμότητα (availability) των δεδομένων από το HDFS δεν επηρεάζεται.

Το HDFS επιτυγχάνει να είναι αξιόπιστο, αποθηκεύοντας τα δεδομένα και αντίγραφα αυτών σε περισσότερους από ένα κόμβους. Συγκεκριμένα, ο χώρος των αρχείων είναι ενιαίος για όλο το cluster με τα αρχεία να διασπώνται σε blocks με κάθε block να αντιγράφεται σε πολλαπλούς κόμβους δεδομένων. Μεταξύ των κόμβων υπάρχει επικοινωνία για εξισορρόπηση των δεδομένων, αντιγραφή ή μετακίνηση τους, έτσι ώστε να παραμείνει ψηλός ο λόγος αντιγραφής (replication).

Μοναδικό σημείο αποτυχίας του HDFS (single point of failure) αποτελεί ο κεντρικός κόμβος, ο name node, ο οποίος διατηρεί πληροφορίες σχετικά με το που βρίσκονται τα δεδομένα στο HDFS και στη περίπτωση που δεν είναι διαθέσιμος δεν είναι δυνατή η πρόσβαση σε αυτά. Ένας δεύτερος κόμβος, ο secondary namenode, διατηρεί αντίγραφα των φακέλων του name node (snapshots) και μαζί με τα αρχεία χρήσης του name node (logs) είναι δυνατή η επαναφορά του συστήματος σε περίπτωση κάποιας βλάβης. Οι κόμβοι όπου αποθηκεύονται δεδομένα ονομάζονται datanodes.

Πολλά άλλα εργαλεία χρησιμοποιούν πλεον το Hadoop framework, όπως το Apache Pig, Apache Hive, Apache HBase[12], Apache Spark[13] δημιουργώντας μια ισχυρή πλατφόρμα με πολλαπλές επιλογές για την ανάλυση και επεξεργασία δεδομένων σε κατανεμημένα περιβάλλοντα.

2.2.2 Apache Kafka

Το Apache Kafka αποτελεί ένα "message broker". Είναι ένα project ανοιχτού κώδικα το οποίο σχεδιάστηκε από την LinkIn ως ένα κατανεμημένο σύστημα μηνυμάτων (messaging) για συλλογή και μεταφορά Log αρχείων με ελάχιστη καθυστέρηση(low latency) και high-throughput σε πραγματικό χρόνο. Πλέον έχει αναλάβει την ανάπτυξή του η Apache. Το Apache Kafka βρίσκει χρήση στο messaging καθώς χαρακτηριστικά όπως high throughput, τεμάχιση αρχείων (built-in partitioning), διατήρηση αντιγράφων (replication) και η ανοχή σε σφάλματα (fault-tolerance) το κάνουν ιδανική λύση σε εφαρμογές επεξεργασίας μηνυμάτων (message processing applications) μεγάλης κλίμακας. Συχνά χρησιμοποιείται για παρακολούθηση (monitoring) δεδομένων συγκεντρώνοντας στατιστικά και δεδομένα χρήσης από κατανεμημένες εφαρμογές. Μέσω του Kafka είναι δυνατό οι λεπτομέρειες που αφορούν τα αρχεία των log να αφαιρεθούν και να αποσταλεί μια πιο "καθαρή" εκδοχή των log ή των event data ως ρεύμα (stream) μηνυμάτων. Άλλες εφαρμογές του Kafka είναι η επεξεργασία ρευμάτων (stream processing), event sourcing, παρακολούθηση δραστηριότητας web εφαρμογών κτλ.

2.2.3 Apache Flume

Το Flume είναι ένα αξιόπιστο κατανεμημένο σύστημα για αποτελεσματική συλλογή, συγκέντρωση και μετακίνηση μεγάλου όγκου δεδομένων καταγραφής χρήσης - log αρχείων στο HDFS ή σε κάποια άλλη βάση. Έχει μια απλή και ευέλικτη αρχιτεκτονική που βασίζεται σε ροές δεδομένων και χρησιμοποιεί ένα απλό επεκτάσιμο μοντέλο για τα

[12] http://hbase.apache.org/
[13] https://spark.apache.org/

δεδομένα. Είναι ανεκτικό σε σφάλματα με ρυθμιζόμενους μηχανισμούς σχετικά με την παροχή αξιοπιστίας και με μηχανισμούς ανάκτησης δεδομένων. Δεν προσφέρει πολλές δυνατότητες όσον αφορά την επεξεργασία των log καθώς έχει σχεδιαστεί λαμβάνοντας υπόψη ότι η ανάλυση τους γίνεται αφού αποθηκευθούν, για παράδειγμα μέσω του προγραμματιστικού μοντέλου MapReduce στο Hadoop.

2.2.4 Apache Storm

Το Apache Storm [14] είναι ένα κατανεμημένο σύστημα που επιτρέπει την ανάλυση και επεξεργασία δεδομένων σε πραγματικό χρόνο. Σχεδιάστηκε αρχικά από την Twitter Inc. ενώ στη συνέχεια ανέλαβε την ανάπτυξη του η Apache. Έχει υψηλή ανοχή σε σφάλματα (high fault-tolerance) και είναι κατακόρυφα κλιμακώσιμο (horizontal scaling) στην ποσότητα δεδομένων εισόδου.

Το Storm για να κάνει υπολογισμούς σε πραγματικό χρόνο υλοποιεί τοπολογίες (topologies). Κάθε κόμβος σε μια τοπολογία επεξεργάζεται μερικώς τα δεδομένα, ενώ συσχετίσεις μεταξύ των κόμβων καθορίζουν σε ποιους κόμβους και με ποια σειρά θα μεταβούν τα δεδομένα ώστε να ολοκληρωθεί η επεξεργασία. Πρόκειται δηλαδή για ένα μοντέλο ροής δεδομένων.

Δεδομένα από εξωτερικές πηγές φτάνουν στις τοπολογίες του Storm μέσω των spouts, που δημιουργούν ροές δεδομένων. Γενικά, τα spouts διαβάζουν πλειάδες (tuples) από τις εξωτερικές πηγές και τις κάνουν emit σε μια τοπολογία.

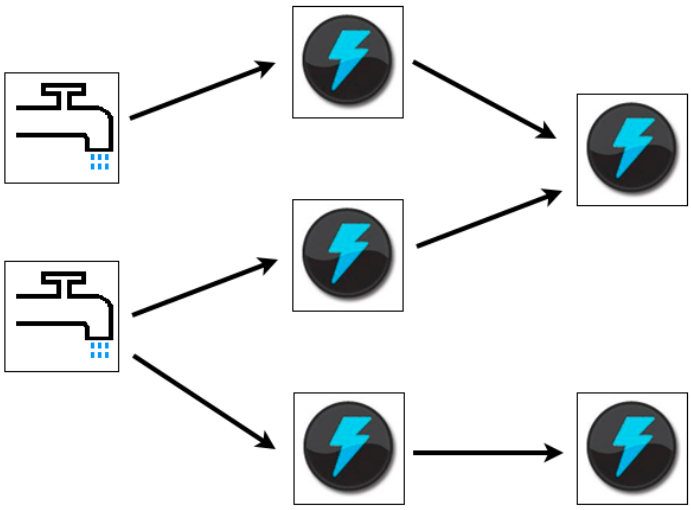
Μια ροή δεδομένων καλείται stream και αποτελείται από μια απέραντη σειρά πλειάδων (tuples). Η πλειάδα είναι σαν μια δομή που μπορεί να αντιπροσωπεύσει τυποποιημένους τύπους δεδομένων (όπως ints, floats και byte arrays) ή τύπους που καθορίζονται από το χρήστη. Κάθε stream ορίζεται από ένα μοναδικό αναγνωριστικό που μπορεί να χρησιμοποιηθεί για την κατασκευή τοπολογιών με spouts και bolts.

Τα Bolts εφαρμόζουν ένα ενιαίο μετασχηματισμό για κάθε stream σε μια τοπολογία. Μπορούν να εφαρμόσουν λειτουργίες αντίστοιχες του MapReduce ή ακόμα και πιο σύνθετες δράσεις που περιορίζονται σε ένα επίπεδο (single-step functions), όπως το φιλτράρισμα, ομαδοποιήσεις ή επικοινωνία με εξωτερικούς φορείς, όπως μια βάση δεδομένων. Μια τυπική τοπολογία Storm εκτελεί πολλαπλούς μετασχηματισμούς και ως εκ τούτου, απαιτεί πολλαπλά bolts.

Ένα bolt είναι δυνατόν να μεταφέρει δεδομένα σε πολλαπλά bolts καθώς επίσης μπορεί να δεχτεί δεδομένα από πολλαπλές πηγές.

Ένα από τα πιο ενδιαφέροντα χαρακτηριστικά στην αρχιτεκτονική του Storm είναι η εγγυημένη επεξεργασία μηνυμάτων. Το Storm μπορεί να εγγυηθεί ότι κάθε πλειάδα που γίνεται emit από ένα spout θα υποβληθεί σε επεξεργασία [15].

Χρησιμοποιείται κυρίως για εφαρμογές machine learning, μελέτη τάσεων (trends) μέσα από κοινωνικά δίκτυα και συνεχείς υπολογισμούς σε δεδομένα που λαμβάνονται σε πραγματικό χρόνο.



Εικόνα 1: Τοπολογία Storm

2.2.5 Elasticseach

Το Elasticsearch είναι μια μηχανή αναζήτησης ανοιχτού κώδικα η οποία στηρίζεται στη βιβλιοθήκη Apache Lucene [16] και έχει αρχιτεκτονική κατανεμημένου συστήματος. Παρέχει αναζήτηση πλήρους κειμένου (full-text search) μέσα από διαδικτυακή διεπαφή (web interface) ακολουθώντας το μοντέλο REST (Representational state transfer). Το Elasticsearch χρησιμοποιεί JSON [17] αρχεία για αποθήκευση των δεδομένων. Χαρακτηρίζεται για τις επιδόσεις του, καθώς προσφέρει indexing και αναζήτηση δεδομένων σχεδόν σε πραγματικό χρόνο.

(αναλυτική περιγραφή στο επόμενο κεφάλαιο)

2.2.6 Logstash

Το Logstash είναι ένα εργαλείο ανοιχτού κώδικα για συλλογή και διαχείριση δεδομένων χρήσης και μετρικών. Υποστηρίζει πολλαπλές επιλογές για είσοδο δεδομένων αλλά και για έξοδο όπου η πιο συνηθισμένη επιλογή είναι το Elasticsearch. Μέσα από τα διάφορα φίλτρα που προσφέρει δίνει αρκετές δυνατότητες για μορφοποίηση και διαχείριση των δεδομένων, ενώ ταυτόχρονα υποστηρίζει και τη μετατροπή από μια μορφή σε άλλη. Ο συνδυασμός με Elasticsearch για αποθήκευση δεδομένων και με το Kibana για δημιουργία γραφημάτων και παρουσίασης των δεδομένων μετατρέπουν το Logstash σε ένα ισχυρό εργαλείο για centralized logging. Το logstash σαν πρόγραμμα μπορεί να χρησιμοποιηθεί ως εργαλείο συλλογής και μετάδοσης των δεδομένων (shipper) από τους κόμβους και ως εργαλείο για επεξεργασία και indexing (indexer). Για επίτευξη πολύ καλών επιδόσεων συνηθίζεται να χρησιμοποιούνται πολλαπλοί indexers σε διαφορετικούς κόμβους.

(αναλυτική περιγραφή στο επόμενο κεφάλαιο)

2.2.7 Kibana 3

Το Kibana είναι ένα εργαλείο το οποίο χρησιμοποιείται για την παρουσίαση των δεδομένων που βρίσκονται στο Elasticsearch. Πρόκειται για διαδικτυακή εφαρμογή που τρέχει αποκλειστικά σε προγράμματα περιήγησης χωρίς κάποιες ιδιαίτερες απαιτήσεις. Μέσα από τις δυνατότητες που προσφέρει όπως γραφικές παραστάσεις και στατιστικές αναλύσεις επιτρέπει την ερμηνεία και ανάλυση των δεδομένων. Επιπρόσθετα παρέχει κατάλληλο πεδίο για αναζητήσεις στα δεδομένα του Elasticsearch μέσω Lucene ερωτημάτων και παρουσίαση των αποτελεσμάτων με γραφικές παραστάσεις και άλλα μέσα, κάτι που δίνει τεράστιες δυνατότητες στους χρήστες. Λόγω αυτής της δυνατότητας πολλοί χρήστες επιλέγουν να χρησιμοποιούν αποκλειστικά το Kibana για να βλέπουν τα δεδομένα τους στο Elasticsearch και αποφεύγουν τη χρήση των APIs του Elasticsearch.

(αναλυτική περιγραφή στο επόμενο κεφάλαιο)

2.2.8 Graylog2

Το Graylog2 είναι ένα πρόγραμμα ανοιχτού κώδικα για διαχείριση και ανάλυση δεδομένων χρήσης και μετρικών. Χωρίζεται σε δύο μέρη, το server που λαμβάνει και αποθηκεύει τα δεδομένα και τη διαδικτυακή εφαρμογή που χρησιμοποιείται για την ανάλυση. Η αποθήκευση των δεδομένων γίνεται στο Elasticsearch, ενώ χρησιμοποιεί τη MongoDB για αποθήκευση των γραφημάτων και των στατιστικών. Το Graylog2 δεν παρέχει τρόπους συλλογής δεδομένων. Μπορεί να λαμβάνει μόνο δεδομένα που ακολουθούν τη μορφοποίηση του γνωστού πρωτοκόλλου syslog ή με μια δική του μορφοποίηση για τα δεδομένα, το Graylog Extended Log Format(GELF) με το οποίο υποστηρίζει την ενσωμάτωση επιπρόσθετων πεδίων στα δεδομένα χρήσης. Η μορφοποίηση αυτή υποστηρίζεται και από το Logstash, οπότε μπορεί τα δεδομένα να καταλήγουν στο Graylog2 μέσω αυτού. Ένας από τους περιορισμούς του εργαλείου, είναι ότι υποστηρίζει συγκεκριμένες εκδόσεις του Elasticsearch, με αποτέλεσμα να μην ακολουθεί τη συνεχή ανάπτυξη του Elasticsearch. Επίσης, παρόλο που υποστηρίζει διαγραφή παλαιών δεδομένων, αυτή γίνεται βάση του όγκου δεδομένων και όχι βάση ημερομηνίας.

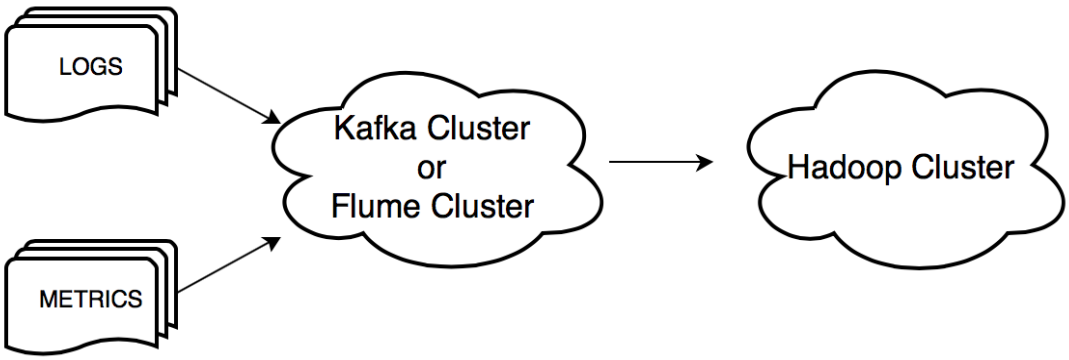
Για επίτευξη καλών επιδόσεων συνηθίζεται να χρησιμοποιούνται πολλαπλά instance του προγράμματος σε διαφορετικούς κόμβους, τα οποία όμως αποθηκεύουν στη ίδια συστοιχία με Elasticsearch.

Η διαδικτυακή εφαρμογή παρέχει τη δυνατότητα δημιουργίας απλών dashboards με γραφήματα και κατάλληλο πεδίο για αναζητήσεις όπως και το Kibana. Επιπρόσθετα παρέχει υπηρεσία προειδοποιήσεων (alerts) που ρυθμίζεται μέσα από την εφαρμογή αυτή. Επιπρόσθετα το Graylog2 διαθέτει σύστημα ταυτοποίησης χρηστών, ώστε να περιορίζεται η πρόσβαση σε ανεπιθύμητα πρόσωπα.

2.3 Ανάλυση σκέψης

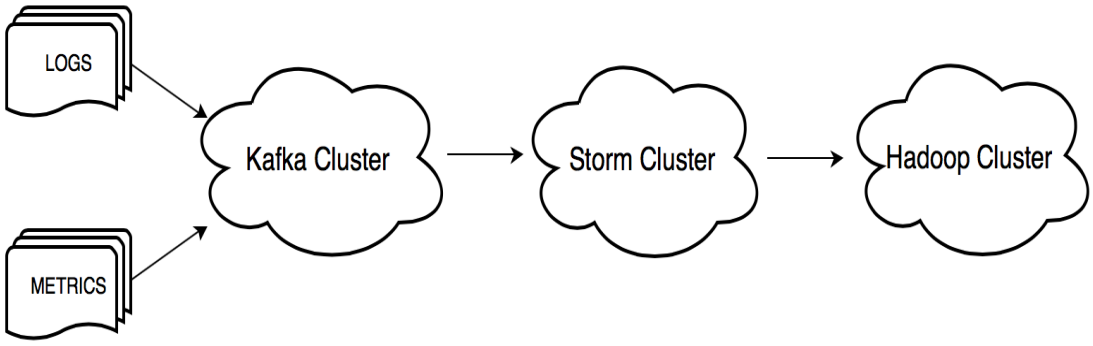
Το Apache Hadoop και το HDFS αποτελούν το σημείο αναφοράς για την επεξεργασία μεγάλου όγκου δεδομένων σε κατανεμημένα περιβάλλοντα όσο αφορά batch processing και είναι βασικά εργαλεία για την ανάλυση αρχείων χρήσης.

Οι δυνατότητες του προγραμματιστικού μοντέλου MapReduce το κάνουν ιδανικό για ανάλυση μεγάλου όγκου δεδομένων χρήσης για στατιστικά, ενώ το Apache Hive και Apache Pig απλοποιούν ακόμη περισσότερο την ανάλυση παρέχοντας στο χρήστη την ευκολία να μην ασχοληθεί ο ίδιος με κώδικα MapReduce εργασιών. Βάση αυτού, το Apache Kafka και το Apache Flume έχουν σχεδιαστεί ώστε να συλλέγουν και να μεταφέρουν αποδοτικά και έμπιστα τα δεδομένα στο HDFS χωρίς να προσφέρουν ιδιαίτερες δυνατότητες επεξεργασίας. Η αρχιτεκτονική του συστήματος με αυτά τα δύο εργαλεία φαίνεται στην εικόνα 2.



Εικόνα 2: Αρχιτεκτονική συστήματος με χρήση Kafka ή Flume

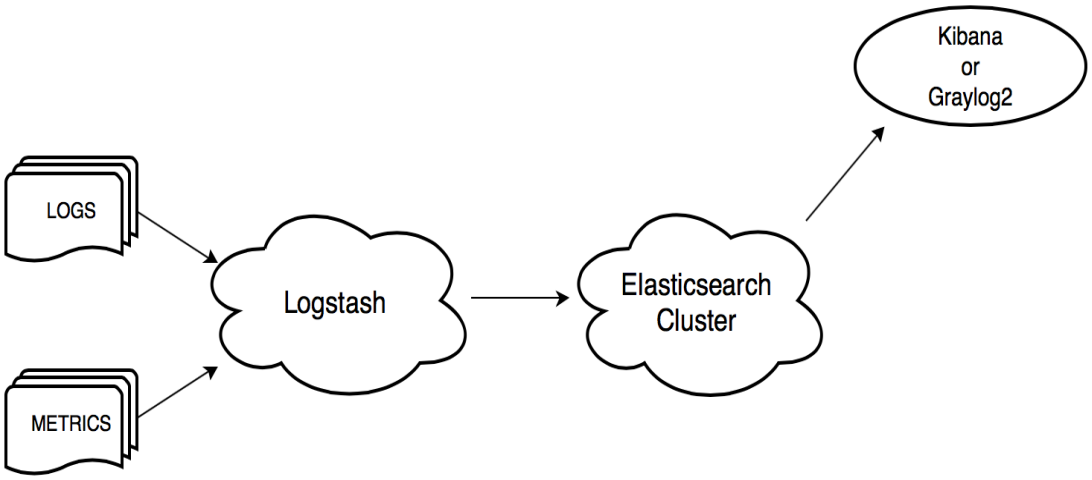
Καθώς η έρευνα έχει σκοπό τη δημιουργία ενός συστήματος με δυνατότητα παροχής των δεδομένων και ανάλυσης αυτών πιο άμεσα σε σχέση με το batch processing, εξετάστηκε η περίπτωση του Apache Storm για ανάλυση των δεδομένων χρήσης σε πραγματικό χρόνο. Η αρχιτεκτονική του συστήματος με βάση το Apache Storm φαίνεται στην εικόνα 3.



Εικόνα 3: Αρχιτεκτονική συστήματος με χρήση Storm και Kafka

Το Apache Storm δε διαθέτει τρόπο για συλλογή δεδομένων, συνεπώς απαιτείται η χρήση του Apache Kafka ή κάποιου άλλου broker (πχ RabbitMQ) ώστε να το τροφοδοτεί σε πραγματικό χρόνο με ουρές δεδομένων, ενώ μετά το Storm τα δεδομένα καταλήγουν στο

HDFS. Με το Storm είναι δυνατή η ανάλυση των δεδομένων χρήσης και των μετρικών σε πραγματικό χρόνο με σκοπό να γίνουν στατιστικές έρευνες, για παράδειγμα ανάλυση δεδομένων χρήσης από το ένα Web Server και εξαγωγή κάποιων συμπερασμάτων που θα βοηθούσαν τον οργανισμό στη λήψη αποφάσεων. Επίσης είναι δυνατή η μορφοποίηση και τροποποίηση των δεδομένων χρήσης. Το συγκεκριμένο σύστημα πέραν το στατιστικών που προσφέρει σε πραγματικό χρόνο με χρήση του Storm, περιορίζεται και πάλι στα εργαλεία που αναφέρθηκαν πιο πάνω στα πλαίσια του batch processing.
Θέλοντας το σύστημα που θα δημιουργηθεί να είναι εύχρηστο και να παρέχει διαδραστικό περιβάλλον με γρήγορα και άμεσα αποτελέσματα στις αναζητήσεις η έρευνα κατέληξε σε λύση που συνδυάζει το Elasticsearch και το Logstash και ένα από τα Kibana 3 και Graylog2 όπως φαίνεται στην εικόνα 4.



Εικόνα 4: Αρχιτεκτονική συστήματος με χρήση Logstash και Elasticsearch

Το Logstash είναι η ιδανική επιλογή για parsing των δεδομένων, ενώ μέσω των διάφορων εξόδων που παρέχει είναι δυνατή και η παροχή αρκετών στατιστικών όπως θα προέκυπταν με τη χρήση του Storm. Το Elasticsearch έχει εκπληκτικές επιδόσεις στην αναζήτηση δεδομένων, συνεπώς οι χρήστες του συστήματος θα μπορούν να έχουν πρόσβαση σχεδόν σε πραγματικό χρόνο. Ανάμεσα στο Kibana 3 και το Graylog2, επιλέχθηκε το Kibana 3 λόγω του ότι αποτελεί ένα πιο ισχυρό εργαλείο στο κομμάτι της παρουσίασης και ανάλυσης των δεδομένων με περισσότερες επιλογές. Το Kibana 3 υστερεί σε δύο βασικά σημεία σε σχέση με το Graylog2, στην παροχή υπηρεσίας προειδοποιήσεων και στο γεγονός ότι δε διαθέτει σύστημα ταυτοποίησης χρηστών. Η παροχή προειδοποιήσεων είναι δυνατόν να επιτευχθεί ως ένα βαθμό με το Logstash. Ένα θετικό του Kibana 3 είναι ότι μαζί με το Elasticsearch και το Logstash αποτελούν πλέον ένα πακέτο εργαλείων το οποίο αναπτύσσεται από τον οργανισμό Elasticsearch [18], οπότε σίγουρα θα υπάρχει πλήρης συμβατότητα μεταξύ αυτών αλλά και υποστήριξη μέσα από την κοινότητα που έχει δημιουργηθεί.

3

Κεφάλαιο 3

Στο κεφάλαιο αυτό θα γίνει αναλυτική παρουσίαση των Elasticsearch, Logastash και Kibana. Τα τρία αυτά εργαλεία ξεκίνησαν ως ανεξάρτητα έργα, ωστόσο το 2013, οι ομάδες ανάπτυξης τους ενώθηκαν και δημιούργησαν το "ELK stack". Συγκεκριμένα, το 2012 δημιουργήθηκε η εταιρεία Elasticsearch εστιάζοντας στη ανάπτυξη του Elasticsearch. Το Γενάρη του 2013 εισήλθε στην εταιρεία ο Rashid Khan, ο δημιουργός του Kibana, και τον Αύγουστο του ίδιου χρόνου τον ακολούθησε ο Jordan Sissel, ο δημιουργός του logstash. Το γεγονός ότι τα τρία αυτά εργαλεία αναπτύσσονται πλέον από την ίδια ομάδα είναι πολύ θετικό καθώς υπάρχει σωστή και άμεση υποστήριξη όσον αφορά τη συμβατότητα των εργαλείων.

3.1 Elasticsearch

Το Elasticsearch δημιουργήθηκε από τον Shay Banon με την πρώτη έκδοση να είναι διαθέσιμη το Φεβρουάριο του 2010. Πρόκειται για μια εξαιρετικά scalable μηχανή για "full-text search" (αναζήτησης πλήρους κειμένου) και παροχή στατιστικών (analytics). Στηρίζεται στη βιβλιοθήκη αναζήτησης Apache Lucene και επιτρέπει την αποθήκευση, αναζήτηση και ανάλυση μεγάλου όγκου δεδομένων πάρα πολύ γρήγορα.

Είναι μια πλατφόρμα αναζήτησης με επιδόσεις "σχεδόν σε πραγματικό χρόνο" (near realtime) και αυτό ερμηνεύεται ως μια μικρή καθυστέρηση - συνήθως ενός δευτερολέπτου από την ώρα εισαγωγής των δεδομένων για αποθήκευση μέχρι να γίνουν διαθέσιμα για αναζήτηση. Έχει σχεδιαστεί για χρήση σε κατανεμημένα περιβάλλοντα. Ερωτήματα τα οποία πραγματοποιούνται πιο συχνά μπορούν να αποθηκευθούν και να εκτελούνται πιο γρήγορα. Ως γλώσσα για τα ερωτήματα χρησιμοποιείται η σύνταξη της Lucene. Μέσα από ένα σύνολο από APIs [19] και DSLs ερωτήματα, καθώς και με clients για τις πιο γνωστές γλώσσες προγραμματισμού προσφέρει απεριόριστες δυνατότητες στο πεδίο της τεχνολογίας αναζήτησης.

Κρίνεται σκόπιμο σε αυτό το σημείο να αναφερθούν μερικές πολύ γνωστές εφαρμογές του Elasticsearch.

- Η Wikipedia χρησιμοποιεί το Elasticsearch για να παρέχει "full text search", καθώς και τα APIs "search-as-you-type" και "did-you-mean" που αυξάνουν το βαθμό ευχρηστίας της συγκεκριμένης υπηρεσίας.
- Η εφημερίδα "The Guardian" χρησιμοποιεί το Elasticsearch ώστε να συνδυάζει δεδομένα χρήσης που παράγονται από τους επισκέπτες της ιστοσελίδα της με δεδομένα από τα social networks, με σκοπό να παρέχει στους συγγραφείς της την άποψη της κοινής γνώμης σχετικά με τα νέα άρθρα σε πραγματικό χρόνο.
- Η StackOverflow συνδυάζει "full text search" μαζί με geolocation queries και χρησιμοποιεί το API του Elasticseach more-like-this για να προτείνει σχετικές ερωτήσει και απαντήσεις.
- Το GitHub χρησιμοποιεί το Elasticsearch για αναζήτησεις σε 130 δισεκατομμύρια γραμμές κώδικα.
- Η Goldman Sachs, μια από τις μεγαλύτερες διεθνείς επενδυτικές τράπεζες στο κόσμο, χρησιμοποιεί το Elasticsearch για την αποθήκευση 5TB δεδομένων χρήσης κάθε μέρα.
- Ένας μεγάλος αριθμός επενδυτικών τραπεζών χρησιμοποιούν το Elasticsearch για ανάλυση των κινήσεων στο χρηματιστήριο.

Τα δεδομένα στο Elasticsearch μπορούν γενικά να χωριστούν σε δύο κατηγορίες, τις ακριβείς τιμές (exact values) και το πλήρες κείμενο (full text). Παραδείγματα ακριβών τιμών είναι η ημερομηνία, ένα αναγνωριστικό χρήστη (ID), ένα όνομα χρήστη ή ακόμα μια διεύθυνση ηλεκτρονικού ταχυδρομείου. Η ακριβής τιμή "Foo" όμως δεν είναι η ίδια με την ακριβή τιμή "foo" ούτε και η τιμή "2014" είναι η ίδια με τη τιμή "15.9.2014".

Σε αντίθεση, το πλήρης κείμενο, αναφέρεται σε δεδομένα κειμένου, συνήθως γραμμένα σε κάποια ανθρώπινη γλώσσα. Παρόλο που η φυσική γλώσσα είναι εξαιρετικά δομημένη, ένα πλήρες κείμενο χαρακτηρίζεται συνήθως ως "αδόμητα δεδομένα" στην περίπτωση που επεξεργάζεται από υπολογιστές, καθώς οι κανόνες της φυσικής γλώσσας είναι πολύπλοκοι και δύσκολα επιτυγχάνεται σωστή ανάλυση.

Στην περίπτωση των ερωτημάτων, όσον αφορά τις ακριβείς τιμές, τα πράγματα είναι πολύ απλά. Το μόνο που εξετάζεται είναι κατά πόσο μια τιμή αντιστοιχεί στο ερώτημα ή όχι. Όσον αφορά το πλήρες κείμενο όμως, το ζήτημα είναι πιο λεπτό. Δεν εξετάζεται κατά πόσο ένα κείμενο ταιριάζει με ένα ερώτημα, αλλά πόσο σχετικό είναι ένα κείμενο με το ερώτημα. Επίσης, επιπρόσθετο ζητούμενο είναι η αναζήτηση να αντιλαμβάνεται και τις προθέσεις του χρήστη. Για να γίνει κατανοητό παραθέτονται μερικά παραδείγματα.

- η αναζήτηση για "UK" θα πρέπει να δίνει και κείμενα που περιέχουν το "United Kingdom".
- η αναζήτηση για "jump" θα πρέπει να ταιριάζει και με τα "jumped", "jumps","jumping".
- η αναζήτηση για 'johnny walker" πρέπει να ταιριάζει και με "Johnnie Walker".
- η αναζήτηση για "johnnie depp" πρέπει να ταιριάζει και με το "Johnny Depp".

Προκειμένου το Elasticsearch να επιτύχει τέτοιου είδους ερωτήματα, πρώτα αναλύει το κείμενο και ακολούθως χρησιμοποιεί τα αποτελέσματα για να δημιουργήσει ένα αντεστραμμένο ευρετήριο (inverted index).

Το αντεστραμμένο ευρετήριο είναι μια δομή, ειδικά σχεδιασμένη για να επιτρέπει γρήγορες αναζητήσεις σε πλήρεις κείμενα. Για κάθε κείμενο υπάρχει μια λίστα με όλες τις μοναδικές λέξεις που εμφανίζονται σε αυτό, ενώ παράλληλα το σύστημα διατηρεί λίστα για κάθε λέξη με όλα τα κείμενα στα οποία εμφανίζεται.

3.1.1 Κύριες έννοιες

cluster: αποτελεί ένα σύνολο από κόμβους (servers) όπου αποθηκεύονται τα δεδομένα και παρέχει indexing και αναζήτηση στο σύνολο των κόμβων αυτών. Κάθε cluster καθορίζεται μοναδικά από το όνομά του, με το προκαθορισμένο όνομα να είναι "elasticsearch". Σε ένα σύστημα μπορούν να συνυπάρχουν πολλά ανεξάρτητα cluster.

node: αποτελεί ένα φυσικό μηχάνημα - server και είναι μέρος του cluster συνεισφέροντας στην αποθήκευση, indexing και αναζήτηση δεδομένων. Καθορίζεται μοναδικά από το όνομά του, το οποίο μπορεί να είναι οτιδήποτε και χρησιμοποιείται από το διαχειριστή, ώστε να καθορίσει σε ποιο cluster ανήκει. Σε ένα cluster μπορούν να υπάρχουν απεριόριστοι κόμβοι.

index: αποτελεί μια συλλογή εγγράφων που έχουν παρόμοια χαρακτηριστικά. Ένα ευρετήριο προσδιορίζεται από ένα όνομα το οποίο χρησιμοποιείται για τον προσδιορισμό συγκεκριμένου ευρετηρίου κατά την αναζήτηση, indexing, ενημέρωσης ή διαγραφής δεδομένων. Μπορούν να οριστούν απεριόριστα ευρετήρια σε ένα cluster.

type: για κάθε Index, μπορεί να οριστούν ένα ή περισσότερα types. Το type αποτελεί ένα λογικό διαχωρισμό του index, όπου η σημασιολογία εξαρτάται αποκλειστικά από το χρήστη που το ορίζει. Γενικά το type ορίζεται για ένα σύνολο δεδομένων, τα οποία έχουν κοινά πεδία. Για παράδειγμα, έαν χρησιμοποιηθεί ένα index για την αποθήκευση δεδομένων ενός blog, θα ήταν σωστό να οριστεί ένα type για τα δεδομένα του χρήστη, ένα type για τα δεδομένα του blog και ένα τρίτο για τα δεδομένα που αφορούν τα σχόλια στο blog.

document: με τον όρο "document" ορίζεται η βασική μονάδα πληροφορίας που μπορεί να τοποθετηθεί σε index και αποθηκεύεται σε μορφή JSON (JavaScript Object Notation). Το JSON είναι μια διαδεδομένη μορφή για διαμοιρασμό πληροφορίας στο διαδίκτυο μεταξύ των servers και εφαρμογών διαδικτύου. Δεν υπάρχει περιορισμός στον αριθμό documents που μπορούν να αποθηκευθούν σε ένα index, πρέπει όμως το κάθε document να συσχετίζεται με κάποιο type του index.

3.1.2 Shards & replicas

Καθώς δεν υπάρχει κάποιος περιορισμός για το μέγεθος ενός index, είναι δυνατόν να αποθηκεύονται σε αυτό τεράστιες ποσότητες δεδομένων που δε χωράνε σε ένα κόμβο λόγω περιορισμών του υλικού. Για παράδειγμα, ένα index με εκατομμύρια documents τα οποία

καταλαμβάνουν περισσότερο από 1TB αποθηκευτικό χώρο πολύ πιθανό να μην χωράνε σε ένα κόμβο ή ακόμα και αν χωράνε πολύ πιθανό να μην υπάρχει η υπολογιστική ισχύς ώστε να προσφέρει υπηρεσίες αναζήτησης αποδοτικά.

Για την επίλυση αυτού του προβλήματος, το Elasticsearch δίνει τη δυνατότητα ένα index να διαιρεθεί σε πολλά κομμάτια τα οποία ονομάζονται shards. Κατά τη δημιουργία ενός index μπορεί να καθοριστεί ο αριθμός των shards. Κάθε shard μπορεί να αποθηκευθεί σε ένα κόμβο και αποτελεί ένα πλήρως λειτουργικό και ανεξάρτητο "index". Αυτό είναι πολύ σημαντικό καθώς μέσω των shards εξασφαλίζεται η κατανεμημένη φύση του συστήματος δίνοντας τη δυνατότητα για παράλληλη επεξεργασία των shards, κάτι που αυξάνει την απόδοση του συστήματος ενώ παράλληλα επιτρέπεται η κατακόρυφη κλιμάκωση του όγκου περιεχομένου ενός index.

Ο μηχανισμός με τον οποίο τα shards κατανέμονται στους κόμβους, αλλά και ο μηχανισμός με τον οποίο τα documents συγκεντρώνονται μετά από ένα αίτημα αναζήτησης από τα διάφορα shards ενός index διαχειρίζονται πλήρως από το Elasticsearch και δεν είναι εμφανές στους χρήστες.

Καθώς τέτοιου είδους συστήματα όπως το Elasticsearch εγκαθιστώνται σε ένα δίκτυο από υπολογιστές ή σε ένα περιβάλλον cloud όπου οι βλάβες μπορούν να εμφανιστούν οποιαδήποτε στιγμή προτείνεται η χρήση κάποιου μηχανισμού διαφύλαξης των δεδομένων (failover mechanism) σε περίπτωση που κάποιος κόμβος πάψει να είναι διαθέσιμος για οποιοδήποτε λόγο. Το Elasticsearch επιτρέπει τη δημιουργία ενός η περισσοτέρων αντιγράφων των shards ενός index τα οποία αποκαλούνται "replica shards" ή απλά "replica". Αυτό εξασφαλίζει την υψηλή διαθεσιμότητα (high availability) και την κλιμάκωση στην απόδοση αναζητήσεων, καθώς είναι δυνατή η παράλληλη αναζήτηση σε όλα τα replicas. Αναγκαία προϋπόθεση είναι ένα replica shard να μην αποθηκεύεται σε καμία περίπτωση στον ίδιο κόμβο με το αντίστοιχο πρωτότυπο shard.

Περιληπτικά, κάθε index μπορεί να χωριστεί σε πολλά shards και κάθε πρωτότυπο (primary) shard μπορεί να έχει αρκετά replicas ή καθόλου. Ο αριθμός των shards και των replicas καθορίζεται κατά τη δημιουργία του index, είναι όμως δυνατή η αλλαγή του αριθμού των replicas ακόμα και μετά τη δημιουργία. Το Elasticsearch έχει προκαθορισμένο κάθε index να έχει πέντε shards και για κάθε shard να υπάρχει ένα αντίγραφο.

3.2 Logstash

Το Logstash είναι ένα από τα κορυφαία εργαλεία για διαχείριση log αρχείων καθώς εξασφαλίζει πολλαπλές επιλογές για συλλογή και μεταφορά των logs ενώ για αποθήκευση χρησιμοποιεί το elasticsearch. Η ανάπτυξη του ξεκίνησε από το 2009 από τον Jordan Sissel. Πρόκειται για ένα έργο ανοιχτού κώδικα (open source) σε γλώσσα προγραμματισμού JRuby το οποίο βρίσκεται συνεχώς υπό εξέλιξη, με την κοινότητα να έχει δραστικό ρόλο μέσω της ανάπτυξης νέων plugins.

Το κύριο πλεονέκτημα του Logstash σε σχέση με άλλα εργαλεία είναι ο τεράστιος αριθμός από plugins [20] για είσοδο (input), έξοδο (output) και για φιλτράρισμα (filters). Επίσης η διαδικασία εγκατάστασής του είναι πολύ απλή ενώ η σύνθεση του configuration αρχείου του δεν είναι πολύ δύσκολη, λόγω της πληρότητας των οδηγών χρήσης και της απλότητας της μορφής του.

3.2.1 Κύριες έννοιες

inputs: μηχανισμοί για πέρασμα μηνυμάτων από logs στο logstash. Το plugin File που διαβάζει αρχεία log, το plugin Redis που διαβάζει από Redis server, το Plugin Syslog που ακούει στη γνωστή πόρτα 514 για syslog μηνύματα και τα μορφοποιεί βάση του πρωτοκόλλου RFC3164, καθώς και τα plugins Collectd και Ganglia για metrics είναι τα πιο γνωστά.

filters: Χρησιμοποιούνται για επεξεργασία των logs. Συνήθως συνδυάζονται με συνθήκες για εφαρμογή κάποιων filters σε συγκεκριμένα logs, ανάλογα με το είδος τους. Το plugin grok που χρησιμοποιείται για parsing και δόμηση των log, το plugin clone το οποίο αντιγράφει ένα log δίνοντας τη δυνατότητα εισαγωγής ή διαγραφής πεδίων και το plugin drop που καταστρέφει - απορρίπτει οποιοδήποτε log του δοθεί ως είσοδο είναι κάποια παραδείγματα.

outputs: μηχανισμοί για έξοδο των logs. Τα πιο γνωστά plugins είναι το elasticsearch, το File για αποθήκευση σε αρχείο, το graphite το οποίο στέλνει στο Graphite.

codecs: Πρόκειται για καθορισμένες μορφοποιήσεις που μπορούν να χρησιμοποιηθούν σε συνδυασμό με ένα input ή ένα output. Γνωστά codecs είναι το json και το msgpack καθώς και αυτό για απλό κείμενο (plain text).

3.2.2 Τρόπος λειτουργίας του Logstash

Ο τρόπος που λειτουργεί το logstash είναι αρκετά απλός. Μπορούμε να το θεωρήσουμε ως έναν αγωγό (pipeline) επεξεργασίας των logs σε τρία στάδια: inputs, filters και outputs. Λαμβάνει τα δεδομένα χρήσης από τις πηγές εισόδου που καθορίζονται με τα input plugins, τα επεξεργάζεται και τα μορφοποιεί μέσω των filter plugins και τέλος τα δομημένα logs δίνονται σε διάφορες εξόδου μέσω των output plugins. Δεν υπάρχει περιορισμός στον αριθμό των plugins που θα χρησιμοποιηθούν σε κάθε στάδιο. Η δυνατότητα αποθήκευσης των δεδομένων χρήσης στο elasticsearch επιτυγχάνεται με τη χρήση του αντίστοιχου output plugin για το elasticsearch, δεν είναι δηλαδή προκαθορισμένη επιλογή.

Επίσης, υπάρχει η διάκριση ανάλογα με το σκοπό που χρησιμοποιείται το Logstash. Στην περίπτωση που εκτελείται στους κόμβους για συλλογή και μετάδοση των δεδομένων χρήσης χωρίς κάποια επεξεργασία, καλείται shipper. Γενικά υπάρχουν πάρα πολλά εργαλεία που μπορούν να χρησιμοποιηθούν για το σκοπό αυτό αντί του logstash, για τα οποία θα υπάρξει αναλυτική περιγραφή στο επόμενο κεφάλαιο. Στην περίπτωση που χρησιμοποιείται για επεξεργασία των δεδομένων χρήσης μέσω filters και αποθήκευσή τους στη συνέχεια στο Elasticsearch ή έξοδο μέσω κάποιου άλλου plugin καλείται indexer. Αυτό δε σημαίνει ότι

απαγορεύεται ή δεν υπάρχει η δυνατότητα χρήσης κάποιου filter όταν χρησιμοποιείται ως shipper.

Παράδειγμα configuration αρχείου για το Logstash:

```
input {
     stdin { }
}

filter {
     if [message] == "error"
          drop { }

}

output {

     elasticsearch { }

     stdout { codec => rubydebug }
}
```

Στο συγκεκριμένο παράδειγμα το logstash λαμβάνει μηνύματα από την κονσόλα μέσω του plugin stdin. Εάν το μήνυμα ταυτίζεται με το string "error" τότε θα απορρίπτεται μέσω του plugin drop, ενώ όλα τα υπόλοιπα θα παρουσιάζονται στη κονσόλα μέσω του stdout στη μορφή rubydebug και θα αποθηκεύονται στο elasticsearch μέσω των αντίστοιχων plugins που ορίστηκαν στο output.

Όπως φαίνεται, η δομή του configuration αρχείου ακολουθεί τα τρία στάδια επεξεργασίας που αναφέρθηκαν πιο πάνω.

Το Logstash μπορεί να χρησιμοποιηθεί και για την παροχή στατιστικών. Χρησιμοποιώντας συγκεκριμένα output plugins, για παράδειγμα το statsd σε συνδυασμό με το Graphite, μπορεί να δημιουργήσει υπολογισμούς και ποσοστά όπως μέγιστη και ελάχιστη τιμή, μέσος όρος κτλ. ανά τακτά χρονικά διαστήματα. Το statsd είναι ένας daemon για τη συγκέντρωση στατιστικών στοιχείων, με μετρητές (counters) και timers, καθώς και την αποστολή τους με UDP σε άλλα εργαλεία για αποθήκευση και προβολή τους, όπως το Graphite.

3.2.3 Grok

Το Grok είναι ίσως το σημαντικότερο filter plugin στο logstash για parsing και δόμηση μηνυμάτων. Πρόκειται για μια βιβλιοθήκη που δημιουργήθηκε από το Jordan Sissel, έναν από τους του δημιουργούς του logstash, στηρίζεται στο regex (regular expression) αλλά επιτρέπει τη δημιουργία κανόνων πιο εύκολα καθώς επιτρέπει την ονομασία των regex patterns και χρήση αυτών των ονομάτων έναντι των regex patterns.

Αποτελεί την καλύτερη επιλογή για parsing αδόμητων logs και μορφοποίηση τους σε δομημένη μορφή δίνοντας τη δυνατότητα πραγματοποίησης Lucene ερωτημάτων αργότερα σχετικά με αυτά εάν αποθηκευτούν στο Elasticsearch. Υπάρχουν περίπου 120 patterns [21]

διαθέσιμα στο logstash τα οποία μπορούν να χρησιμοποιηθούν άμεσα από τους χρήστες ενώ με τη βοήθεια του εργαλείου grokdebug [22] οι χρήστες μπορούν να δημιουργήσουν τα δικά τους patterns.

Το Grok στην ουσία ταυτίζει τα patterns κειμένου που ορίζονται στο configuration αρχείο με το περιεχόμενο των logs.

Η σύνταξη ενός grok pattern είναι η εξής: %{SYNTAX:SEMANTIC}.

Το SYNTAX είναι το όνομα του pattern το οποίο θα ταυτιστεί με το κείμενο. Για παράδειγμα, από το κείμενο "42 192.168.42.42" το "42" μπορεί να ταυτιστεί από το NUMBER pattern ενώ το κείμενο "192.168.42.42" μπορεί να ταυτιστεί από το IP pattern.

Το SEMANTIC είναι το χαρακτηριστικό όνομα που θα δοθεί στο μέρος του κείμενου που θα ταυτιστεί από το αντίστοιχο SYNTAX. Για παράδειγμα το "42" μπορεί να είναι η διάρκεια ενός γεγονότος σε δευτερόλεπτα οπότε το SEMANTIC μπορεί να είναι "duration" ενώ για το "192.168.42.42" αφού πρόκειται για διεύθυνση IP μπορεί να είναι "client".

Με βάση τα πιο πάνω, το grok filter για το παράδειγμα που δόθηκε είναι το εξης:

%{NUMBER:duration} %{IP:client}

Επιπλέον, υπάρχει η δυνατότητα καθορισμού τύπου μετατροπής των δεδομένων στο grok pattern. Η προεπιλεγμένη επιλογή για όλα τα semantics είναι να αποθηκεύονται ως κείμενο (string). Μια μετατροπή τύπου για ένα semantic καθορίζεται με την προσθήκη του κατάλληλου τύπου στο τέλος. Για παράδειγμα μέσω του %{NUMBER:num:int} μετατρέπεται το num από κείμενο (string) σε ακέραιο αριθμό (integer). Υποστηρίζονται μόνο μετατροπές σε int και float.

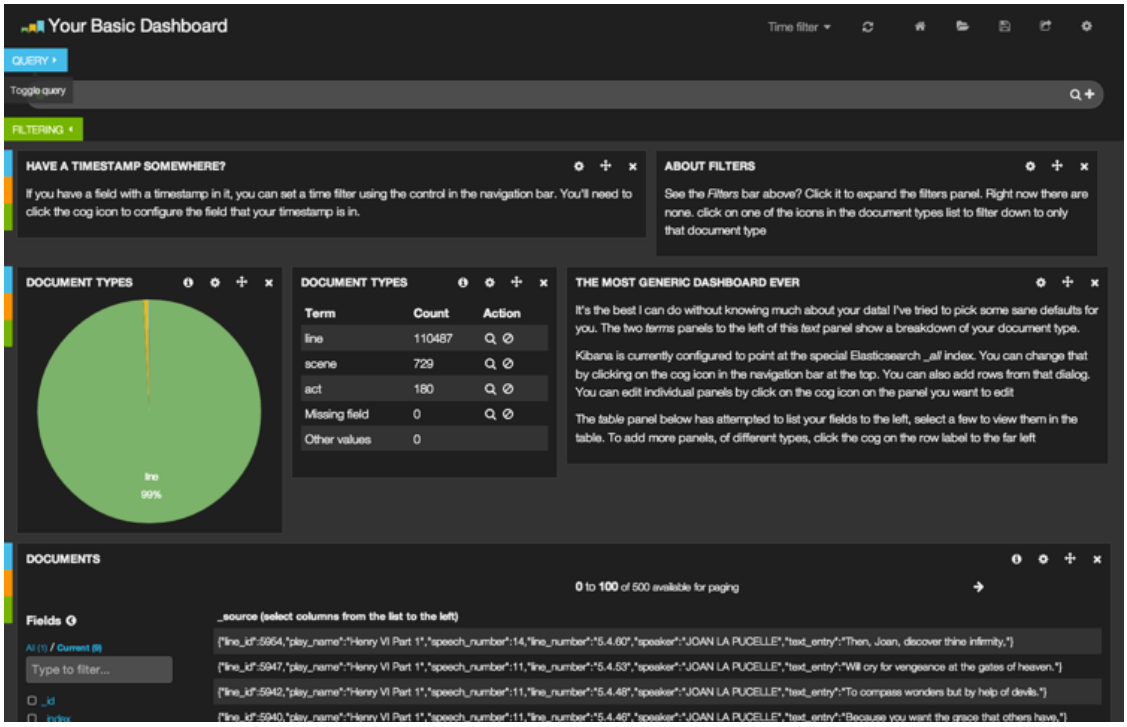
3.3 Kibana

Πρόκειται για ένα project ανοιχτού κώδικα (Apache Licensed) που ξεκίνησε το 2011 από τον Rashid Khan με σκοπό την παρουσίαση των δεδομένων που είναι αποθηκευμένα στο Elasticsearch. Το Kibana 3 είναι η νεότερη έκδοση του Kibana και είναι γραμμένο σε HTML5 και Javascript εξ ολοκλήρου ενώ το Kibana 2 στηρίζεται στη Ruby. Βασική διαφορά μεταξύ του Kibana 3 και του Kibana 2 είναι ότι το Kibana 3 δίνει τη δυνατότητα απεικόνισης δεδομένων και από άλλες πηγές που δε χρησιμοποιούν τη μορφοποίηση του elasticsearch, για παράδειγμα δεδομένα από graylog2.

Η νεότερη έκδοση τρέχει αποκλειστικά σε προγράμματα περιήγησης (Firefox, Chrome, Internet Explorer) όσον αφορά τους clients ενώ το μόνο που απαιτεί στη πλευρά του server είναι ένα web server, οπως τον Apache2 Web Server [23].

Το Kibana μέσα από τα διάφορα είδη γραφικών παραστάσεων, πινάκων και στατιστικών που προσφέρει, αποτελεί ένα κορυφαίο εργαλείο για την ερμηνεία των δεδομένων που βρίσκονται στο Elasticsearch. Ο χρήστης δημιουργεί dashboards πολύ εύκολα χωρίς να χρειάζεται να γράψει κώδικα, μιας και αυτή ήταν η κύρια ιδέα του project. Υπάρχουν έτοιμα dashboards, το sample dashboard που προβάλλει κάθε είδους documents από το

elasticsearch και το logstash dashboard που προβάλλει δεδομένα μόνο από indexes του logstash από το elasticsearch τα οποία οι νέοι χρήστες μπορούν να χρησιμοποιήσουν για αρχή και να τα τροποποιήσουν στη συνέχεια ενώ η δημιουργία κάποιου dashboard από το μηδέν είναι αρκετή απλή ακολουθώντας τους οδηγούς χρήσης.



Εικόνα 5: "Basic Dashboard" στο Kibana

Το Kibana επιτρέπει στους χρήστες να πραγματοποιούν αναζητήσεις στα δεδομένα του Elasticsearch μέσω της Lucene ερωτημάτων. Τα ερωτήματα εισάγονται στο κελί εισόδου ερωτημάτων στο πάνω μέρος της σελίδας. Τα αποτελέσματα των αναζητήσεων εμφανίζονται σε όλα τα μέρη του dashboard. Για παράδειγμα αν υπάρχει μια γραφική παράσταση και ένα πίνακας, και στα δύο θα υπάρχουν πληροφορίες μόνο για το ερώτημα που προηγήθηκε. Στην περίπτωση πολλαπλών ερωτημάτων, οι πληροφορίες στις γραφικές παραστάσεις διαχωρίζονται με χρήση διαφορετικών χρωμάτων δίνοντας τη δυνατότητα σύγκρισης των δεδομένων.

Εικόνα 6: Επιλογή αναζήτησης στο Kibana

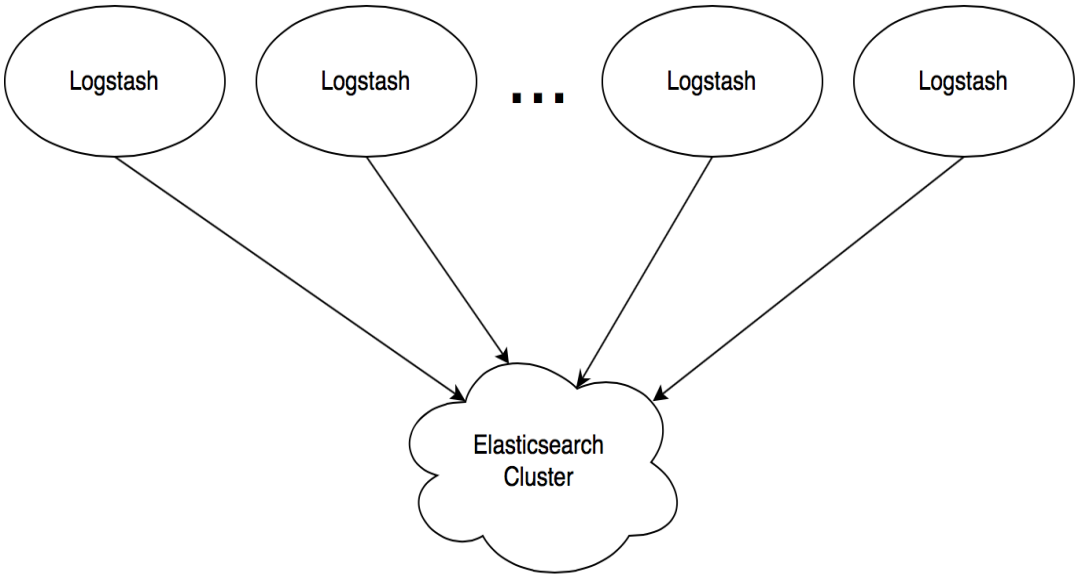
4

Κατανεμημένες τεχνικές συλλογής και μεταφοράς δεδομένων χρήσης

Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο το Logstash δίνει τη δυνατότητα να λαμβάνονται δεδομένα χρήσης από διάφορες πηγές, ενώ το ίδιο το εργαλείο μπορεί να χρησιμοποιηθεί και ως shipper και ως indexer. Στο κεφάλαιο αυτό θα περιγραφεί με ποιο τρόπο εξασφαλίζεται η κλιμακωσιμότητα του συστήματος συνδυάζοντας κάποια βοηθητικά εργαλεία, ενώ θα αναλυθούν και εναλλακτικές επιλογές για shippers πέραν του Logstash.

4.1 Ανάλυση

Μέσα από τις πολλαπλές επιλογές για είσοδο και έξοδο που δίνει το Logstash μπορούν να προκύψουν πολλοί συνδυασμοί για το πώς θα καταλήξουν τα δεδομένα χρήσης από τους κόμβους στους οποίους παράγονται, στο Elasticsearch που είναι το ζητούμενο. Στην πρώτη επαφή με το Logstash και το Elasticsearch, συνήθως ο χρήστης ακολουθεί εγκατάσταση των εργαλείων σε ένα κόμβο. Ρυθμίζοντας κατάλληλα το αρχείο ρυθμίσεων (configuration file), το Logstash παίρνει τα δεδομένα εισόδου σύμφωνα με τις πηγές εισόδου, για παράδειγμα από αρχεία ή ακούγοντας στην προκαθορισμένη θύρα για syslog μηνύματα, εφαρμόζει τα διάφορα φίλτρα και ακολούθως τα στέλνει στο Elasticsearch. Το ερώτημα που δημιουργείται άμεσα είναι κατά πόσο αυτό μπορεί να γίνει στην πράξη για μεγάλο αριθμό κόμβων. Γιατί να υπάρχει ο διαχωρισμός μεταξύ shipper και indexer από τη στιγμή που υπάρχει δυνατότητα σε κάθε κόμβο μιας συστοιχίας υπολογιστών μέσω του Logstash να πραγματοποιείται η συλλογή των δεδομένων, να εφαρμόζονται τα φίλτρα και ακολούθως η αποστολή τους στο Elasticsearch;



Εικόνα 7: Αποστολή δεδομένων χρήσης από κάθε κόμβο απευθείας στο Elasticsearch

Ο κυριότερος λόγος που αποφεύγεται η υλοποίηση της πιο πάνω εικόνας, είναι γιατί η εφαρμογή των φίλτρων αποτελεί μια διαδικασία με υψηλές απαιτήσεις σε υπολογιστικούς πόρους, κυρίως επεξεργαστική ισχύ. Γι' αυτό το λόγο αποφεύγεται η εκτέλεση της συγκεκριμένης διαδικασίας στους κόμβους που παράγονται τα δεδομένα χρήσης, καθώς θα επηρεάζεται αρνητικά η επίδοση των εφαρμογών που τρέχουν στους συγκεκριμένους κόμβους.

Αυτό που προτείνεται, είναι η δημιουργία συστοιχίας υπολογιστών όπου θα τρέχουν αποκλειστικά οι Logstash indexers, ώστε να υπάρχει αρκετή επεξεργαστική ισχύ για τη διεκπεραίωση των εργασιών τους. Συνηθίζεται στους υπολογιστές αυτούς να τρέχει και το Elasticsearch.

Στη συνέχεια θα εξεταστούν τρεις διαφορετικοί τρόποι για το πως μπορούν να καταλήξουν τα δεδομένα χρήσης στη συστοιχία με τους indexers και ακολούθως στο Elasticsearch. Στην ουσία η διαφορά περιορίζεται στο είδος του shipper, το εργαλείο δηλαδή που θα αναλάβει τη συλλογή και αποστολή των δεδομένων χρήσης.

Συχνά συνδυάζεται το Logstash με Syslog-ng [link] ή Rsyslog [link] ως shippers σε κόμβους που τρέχουν Linux διανομές. Τα δύο αυτά εργαλεία στηρίζονται στο πρωτόκολλο Syslog για τη συλλογή και μετάδοση δεδομένων χρήσης και αρχικά σχεδιάστηκαν με σκοπό να υποστηρίζουν μόνο τέτοια μηνύματα. Αυτή τη στιγμή υποστηρίζουν και άλλες πηγές για δεδομένα χρήσης όπως αρχεία, αλλά μετατρέπουν τα δεδομένα σε μηνύματα σύμφωνα με το πρωτόκολλο Syslog, κάτι που αυξάνει την πολυπλοκότητα στο κομμάτι της διαχείρισης μέσω φίλτρων. Γενικά τα δύο αυτά εργαλεία επιλέγονται όταν τα δεδομένα χρήσης χρησιμοποιούν στην πλειοψηφία το πρωτόκολλο Syslog.

Οι επιλογές είναι πάρα πολλές όσον αφορά τα εργαλεία - shippers. Προτιμήθηκε να γίνει αναφορά στα εργαλεία που αναπτύχθηκαν συγκεκριμένα για το Logstash. Γενικά δεν υπάρχει σωστή ή λάθος επιλογή. Το τι θα επιλεγεί σχετίζεται άμεσα με τις απαιτήσεις που

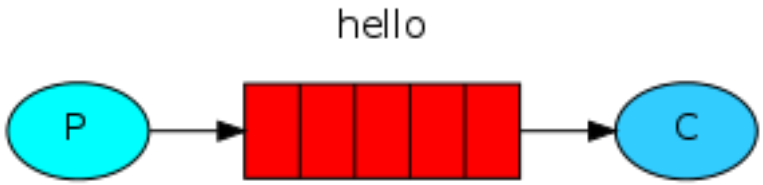
έχει όσον αφορά την ταχύτητα και την ασφάλεια κατά την μετάδοση, καθώς και το σκοπό για τον οποίο μαζεύει τα δεδομένα χρήσης.

4.2 Αρχιτεκτονική συστήματος χρησιμοποιώντας το Logstash ως Shipper

Αρχικά θα γίνει περιγραφή δύο εργαλείων απαραίτητων για την υλοποίηση του συστήματος.

4.2.1 rabbitMQ

Η RabbitMQ [24] είναι ένας "message broker" ανοιχτού κώδικα κάτω από την άδεια Mozilla Public License. Η βασική ιδέα ενός "message broker" είναι πολύ απλή, δέχεται, αποθηκεύει προσωρινά και προωθεί μηνύματα. Το πρόγραμμα που στέλνει μηνύματα καλείται "producer" ενώ το πρόγραμμα που αναμένει μηνύματα καλείται "consumer". Τα μηνύματα αποθηκεύονται σε ουρές (queues), οι οποίες αισθητικά αποτελούν ένα ατελείωτο buffer. Πολλοί producers μπορούν να στείλουν μηνύματα σε μια ουρά, και πολλοί consumers μπορούν να αναμένουν μηνύματα από αυτή την ουρά.



Εικόνα 8: Producer και Consumer για ένα stream

Αναπτύχθηκε από την RambbitMQ ενώ πλέον αποτελεί μέρος της Pivotal Software, Inc. Στηρίζεται στη γλώσσα προγραμματισμού Erlang και στο framework "Open Telecom Platform" [link] το οποίο προσφέρεται για συστοιχίες υπολογιστών και αντιμετώπιση βλαβών. Η Erlang είναι μια γλώσσα προγραμματισμού για τη σχεδίαση κλιμακώσιμων συστημάτων που τρέχουν σε πραγματικό χρόνο με την ανάγκη συνεχής διαθεσιμότητας (high availability). Είναι γνωστή για τη δυνατότητα να πραγματοποιούνται επανεκκινήσεις προγραμματιστικών κομματιών του συστήματος χωρίς να χρειάζεται η διακοπή λειτουργίας ολόκληρου του συστήματος. Από την πλευρά των clients υπάρχουν βιβλιοθήκες σε πολλές γλώσσες προγραμματισμού για την αλληλεπίδραση με τη RabbitMQ.

Η RabbitMQ διαθέτει πύλες (gateways) για επικοινωνία μέσω AMQP, STOMP και MQTT. Υποστηρίζει κρυπτογραφημένη μεταφορά δεδομένων μέσω TLS χρησιμοποιώντας τη βιβλιοθήκη openSSL[14]. Μπορεί να ρυθμιστεί να τρέχει σε συστοιχία υπολογιστών αλλά να αντιμετωπίζεται ως ένα λογικό μηχάνημα. Αυτό δίνει τη δυνατότητα να μπορεί να διακοπεί η λειτουργία κάποιου υπολογιστή χωρίς τη διακοπή ολόκληρου του συστήματος. Είναι επίσης δυνατή η αντιγραφή ουρών σε διάφορους υπολογιστές για να εξασφαλιστεί ότι

[14] http://www.openssl.org/

σε περίπτωση αστοχίας υλικού κανένα μήνυμα δε θα χαθεί. Προφανώς αυτές οι υπηρεσίες που εξασφαλίζουν υψηλή διαθεσιμότητα έχουν επίπτωση στην επίδοση.

4.2.2 Redis

Η Redis (REmote DIctionary Server) [25] αποτελεί την πιο γνωστή "key-value" βάση (σύμφωνα με db-engines.com), εμπίπτει δηλαδή στην κατηγορία "NoSQL" βάσεων. Ανακοινώθηκε το 2008 ως ένα μικρό έργο ανοιχτού κώδικα, αλλά πήρε μεγάλη διάσταση μετά την ανάμειξη της VMware. Το μοντέλο "key-value" είναι μια απλή διεπαφή όπου ένα κλειδί (key) σχετίζεται άμεσα με μια τιμή (value) που μπορεί να είναι οποιουδήποτε τύπου δεδομένων. Η Redis σχεδιάστηκε να διατηρεί όλα τα ζεύγη key-value στη κύρια μνήμη (RAM) του κάθε κόμβου (in-memory) για επίτευξη καλύτερης απόδοσης, αλλά στην περίπτωση που η μνήμη δεν είναι αρκετή, κάποια από τα ζεύγη μεταφέρονται στο σκληρό δίσκο μέσω εικονικού συστήματος μνήμης (virtual memory system). Αυτό όμως επηρεάζει αρνητικά την επίδοση. Το σύστημα παρέχει ακεραιότητα των δεδομένων (data persistance) σε περίπτωση επανεκκίνησης ή βλάβης του κόμβου, μέσω journal files και διατήρησης στιγμιότυπων της μνήμης (snapshot). Λόγω αυτής της υπηρεσίας επιτρέπει τη χρήση της και ως κύρια βάση και όχι μόνο για προσωρινή αποθήκευση (volatile cache). Επίσης είναι δυνατή η διατήρηση αντιγράφων(replication). Συγκεκριμένα είναι δυνατόν ένας κόμβος, ο "slave", να διατηρεί αντίγραφα για τα δεδομένα κάποιου άλλου κόμβου, του "master" (μοντέλο master-slave). Αυτό πραγματοποιείται ασύγχρονα, γι' αυτό και ο "slave" δεν είναι συνεχώς ενημερωμένος.

Η Redis τρέχει σε συστοιχίες υπολογιστών ως κατανεμημένο σύστημα, ωστόσο μπορεί να εκτελεστεί και σε ένα μόνο κόμβο. Προσθέτοντας κόμβους αυξάνεται η απόδοση του συστήματος και η διαθεσιμότητα.

Όσον αφορά τη μεταφορά δεδομένων χρήσης, χρησιμοποιούνται τα "channels" που επιτρέπουν τη δημιουργία μιας publish-subscribe υποδομής μηνυμάτων.

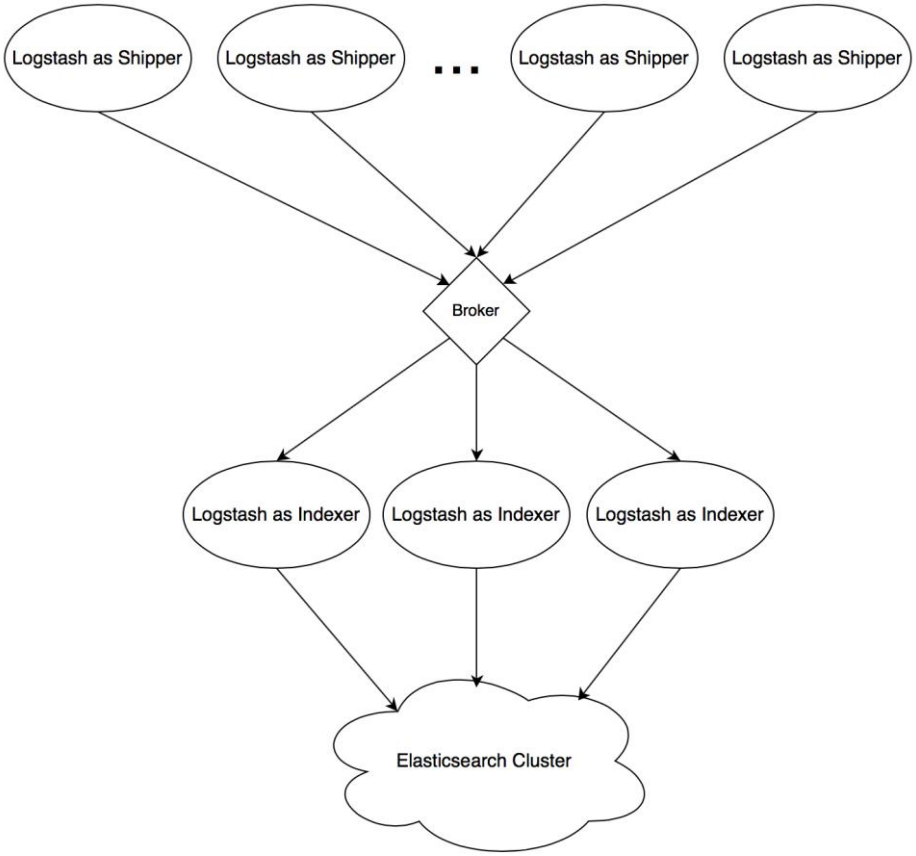
Μειονέκτημα της Redis είναι ότι δεν υποστηρίζει κρυπτογράφηση δεδομένων. Για την προστασία δεδομένων διαθέτει μόνο κωδικό πρόσβασης [redis-security].

4.2.3 Περιγραφή Αρχιτεκτονικής συστήματος

Η συγκεκριμένη αρχιτεκτονική στηρίζεται στο γεγονός ότι τα αρχεία δεδομένων μπορούν να προκύψουν από διάφορες πηγές, όχι μόνο από αρχεία ή μηνύματα syslog. Το Logstash ως Shipper σε κάθε κόμβο μέσα από τα πολλαπλά input plugins που διαθέτει μπορεί να εξασφαλίσει τη συλλογή τους, ενώ μέσω των διαφόρων output plugins εξασφαλίζει διάφορους τρόπους μετάδοσης.

Μεταξύ των κόμβων στους οποίους συλλέγονται τα δεδομένα με το Logstash ως Shipper και της συστοιχίας από τους Logstash - Indexers παρεμβάλλεται ένας broker, ένα εργαλείο που λαμβάνει, αποθηκεύει προσωρινά και προωθεί τα μηνύματα προς τους indexers. Τα πιο συνηθισμένα εργαλεία που συνδυάζονται με το Logstash και λειτουργούν ως broker είναι η Redis και η RabbitMQ που περιγράφηκαν πιο πάνω. Το καθένα από αυτά έχει ξεχωριστά

πλεονεκτήματα γι' αυτό και οι επιλογή καθορίζεται ανάλογα με τις ανάγκες του συστήματος. Η Redis ξεχωρίζει για την επίδοση που έχει, ενώ η RabbitMQ επιλέγεται στην περίπτωση που απαιτείται η κρυπτογραφημένη μετάδοση των αρχείων μιας και η Redis δεν υποστηρίζει αυτή την υπηρεσία. Χρησιμοποιώντας broker τα δεδομένα χρήσης συγκεντρώνονται εκεί και διαμοιράζονται στους indexers οι οποίοι τα επεξεργάζονται μέσω των φίλτρων και τα προωθούν στο Elasticsearch ή και σε άλλες εξόδους. Η αρχιτεκτονική αυτή φαίνεται στην εικόνα 9.



Εικόνα 9: Αρχιτεκτονική συστήματος με τη χρήση broker και του Logstash ως shipper

Καθώς και τα δύο εργαλεία που προτείνονται, η Redis και η RabbitMQ, εκτελούνται σε συστοιχίες υπολογιστών ως κατανεμημένα συστήματα δεν περιορίζουν τη κλιμάκωση του συστήματος. Συγκεκριμένα, αυτή η αρχιτεκτονική παρουσιάζει άψογη κλιμάκωση σε όλα τα σημεία, αφού, οι indexers, οι κόμβοι με το Elasticsearch, αλλά και οι κόμβοι του broker μπορούν να αυξηθούν ώστε να επιτευχθεί υψηλή επίδοση σε πολύ μεγάλη κλίμακα. Επιπρόσθετη χρησιμότητα του broker είναι ότι επιτρέπει την αναβάθμιση των indexers και των κόμβων του Elasticsearch χωρίς τη διακοπή του συστήματος αφού τα δεδομένα θα συνεχίσουν να συσσωρεύονται στο broker και θα μεταφερθούν αργότερα στους indexers όταν είναι διαθέσιμοι.

Βασικός περιορισμός όταν χρησιμοποιείται broker είναι τα δεδομένα χρήσης που είναι πολλαπλών γραμμών να ομαδοποιούνται ως ενιαίο μήνυμα στην πλευρά των shippers, καθώς αν σταλούν σε διαφορετικά μηνύματα δεν είναι βέβαιο ότι θα καταλήξουν στον ίδιο indexer, με αποτέλεσμα να μην συμπληρωθεί ποτέ ολόκληρο το μήνυμα. Το Logstash

διαθέτει το φίλτρο multiline, το οποίο αναλαμβάνει την ομαδοποίηση τέτοιων μηνυμάτων. Αν και αποφεύγεται η χρήση φίλτρων στους shippers, το συγκεκριμένο είναι απαραίτητο για τη σωστή λειτουργία του συστήματος.

4.3 Εναλλακτικοί shippers

Η ανάγκη για τη δημιουργία εναλλακτικών shippers προέκυψε λόγω του ότι το Logstash εκτελείται σε Java Runtime, με αποτέλεσμα να καταναλώνει αρκετούς πόρους του συστήματος, δυσανάλογο από ότι μπορούσε κανείς να θεωρήσει λογικό στα πλαίσια ενός εργαλείου για τη διαχείριση των δεδομένων χρήσης. Προφανώς το Logstash έχει μεγάλο πλεονέκτημα σε σχέση με άλλα εργαλεία όσον αφορά το σύνολο των πηγών που υποστηρίζει, αλλά σε κόμβους χαμηλής υπολογιστικής ισχύς, για παράδειγμα εικονικές μηχανές στο cloud, αποφεύγεται να χρησιμοποιείται καθώς περιορίζει τη μνήμη σημαντικά επηρεάζοντας την επίδοση τους. Ο logstash-forwarder [26] και το Beaver [27] είναι δύο shippers, οι οποίοι υποστηρίζουν τη συλλογή και αποστολή δεδομένων χρήσης από αρχεία. Καθώς τα αρχεία είναι ο πιο γνωστός τρόπος παροχής των δεδομένων χρήσης από τα προγράμματα, τις περισσότερες φορές η χρήση αυτών των εργαλείων δεν περιορίζει το σύστημα. Επίσης, υπάρχουν προγράμματα τα οποία αν και δεν έχουν προεπιλεγμένη επιλογή την παροχή των δεδομένων σε αρχείο, δίνεται η επιλογή κατόπιν αντίστοιχης ρύθμισης.

4.3.1 Logstash-forwarder

Ο Logstash-forwarder είναι ένας deamon, που αναλαμβάνει την αποστολή δεδομένων χρήσης που βρίσκονται αποκλειστικά σε αρχεία και μπορεί να συνδεθεί μόνο με ένα logstash indexer. Αναπτύχθηκε στη γλώσσα προγραμματισμού GO και υποστηρίζεται ως έργο ανοιχτού κώδικα από τον Jordan Sissel, το δημιουργό του logstash. Στόχος του έργου είναι να λύσει το πρόβλημα μνήμης (RAM) και επεξεργαστικής ισχύς (CPU) που υπάρχει σε μικρά υπολογιστικά συστήματα τα οποία χρησιμοποιούν το logstash ως shipper. Ο lsf (logstash-forwarder) χρησιμοποιεί το πρωτόκολλο δικτύου (network protocol) Lumberjack το οποίο αναπτύχθηκε για τους σκοπούς του συγκεκριμένου έργου και το οποίο προσφέρει ασφάλεια μέσω κρυπτογράφησης, περιορισμένη καθυστέρηση (low latency), χρησιμοποιεί ελάχιστους πόρους του συστήματος και είναι αξιόπιστο. Χρησιμοποιεί τη βιβλιοθήκη OpenSSL για κρυπτογράφηση και πιστοποιητικά τύπου X509 για έλεγχο της πλευράς του server, ενώ είναι δυνατή η χρήση πιστοποιητικών και για την πλευρά του client. Επίσης χρησιμοποιώντας τη βιβλιοθήκη zlip[15] συμπιέζει τα δεδομένα για μείωση της υπερφόρτωσης της σύνδεσης.

Βασικό μειονέκτημα θεωρείται η μη υποστήριξη αποστολής δεδομένων σε brokers. Λόγω του ότι το πρωτόκολλο Lumberjack είναι ιδανική επιλογή καθώς συνδυάζει την ταχύτητα μετάδοσης και την κρυπτογράφηση των δεδομένων πολλοί θεωρούν ότι θα ήταν ο τέλειος shipper αν υποστήριζε την επικοινωνία με κάποιο broker. Από την πλευρά του δημιουργού του έγινε μεγάλη προσπάθεια για την παροχή αυτής της δυνατότητας, χωρίς κανένα

[15] http://www.zlib.net/

αποτέλεσμα καθώς κάνένας broker δε μπορεί να συνδυαστεί με το πρωτόκολλο Lumperjack. Για παράδειγμα η Redis αρνήθηκε να παρέχει δυνατότητες κρυπτογράφησης και συμπίεσης δεδομένων.

Επίσης δεν υπάρχει δυνατότητα αναγνώρισης δεδομένων χρήσης πολλαπλών γραμμών σε ένα αρχείο. Κάθε γραμμή αποστέλνεται ξεχωριστά και γι' αυτό και τα δεδομένα χρήσης πολλαπλών γραμμών πρέπει να αναγνωρίζονται και να ενώνονται στην πλευρά του indexer logstash μέσω του multiline φίλτρου.

Όπως αναφέρθηκε πιο πάνω, ένας lsf συνδέεται αποκλειστικά με ένα logstash indexer. Δίνει την δυνατότητα όμως να οριστεί ένας πίνακας από IPs για εναλλακτικές επιλογές από indexers αλλά κάθε φορά που εκκινεί, επιλέγει τυχαία μια από τις IPs και ανοίγει μια σύνδεση TCP με τον συγκεκριμένο Indexer.

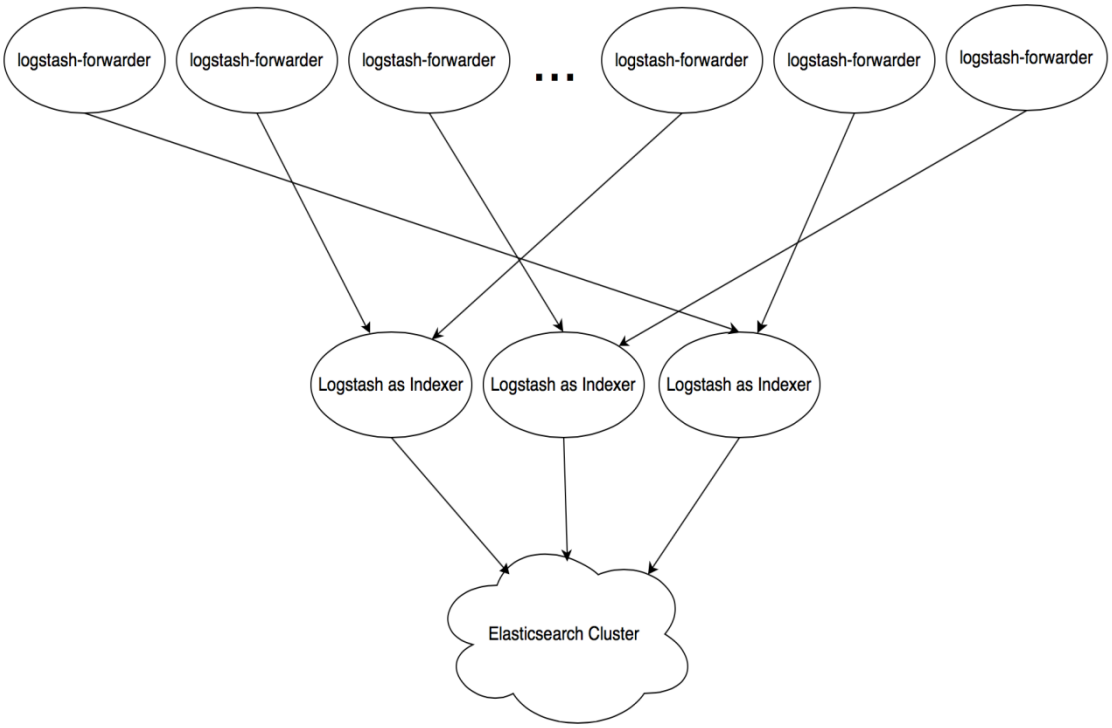
Ένας από τους στόχους του σχεδιασμού του lsf είναι η μετάδοση χωρίς απώλειες δεδομένων χρήσης. Αυτό επιτυγχάνεται μέσω του μηχανισμού "receipt acknowledgement" σύμφωνα με τον οποίο ο lsf λαμβάνει ενημέρωση ότι τα δεδομένα έχουν παραληφθεί. Στην περίπτωση που ο Logstash indexer δεν ανταποκρίνεται είτε λόγω υπερφόρτωσης είτε λόγω προβλήματος στο δίκτυο ο lsf διακόπτει την αποστολή δεδομένων και αναμένει. Αν έχει ρυθμιστεί όπως περιγράφεται πιο πάνω με πολλαπλούς indexers τότε επιλέγει κάποιο άλλο indexer για να συνδεθεί. Αυτό σε συνδυασμό με το γεγονός ότι τα δεδομένα χρήσης πολλαπλών γραμμών αποστέλνονται σε ξεχωριστά μηνύματα ανά γραμμή, κάνει το συγκεκριμένο εργαλείο αναξιόπιστο για την αποστολή τέτοιων δεδομένων. Η διακοπή της σύνδεσης κατά τη διάρκεια αποστολής ενός δεδομένου χρήσης πολλαπλών γραμμών θα έχει ως αποτέλεσμα να μην καταλήξει ποτέ στο Elasticsearch αφού τα επιμέρους μηνύματα θα καταλήξουν σε περισσότερους από ένα indexers.

Η κατανομή του φόρτου εργασίας στους indexers, στηρίζεται στο ότι μετά την τυχαία επιλογή από όλους τους logstash-forwarders κάποιοι indexers θα είναι υπερφορτωμένοι με αποτέλεσμα κάποιοι lsf να αναγκαστούν να συνδεθούν με άλλους indexers καταλήγοντας σε ισοκατανομή.

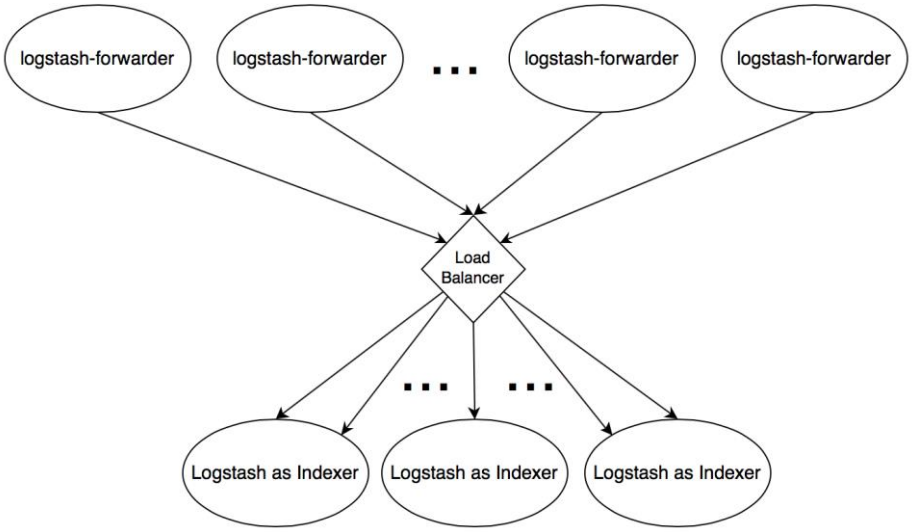
Κάποιοι επιλέγουν να εξασφαλίσουν την ισοκατανομή στους indexers χρησιμοποιώντας ένα "load balancer" όπως στο HAproxy[16]. Όλοι οι logstash-forwarder θα ρυθμιστούν να ζητάνε σύνδεση από το συγκεκριμένο server, και αυτός χρησιμοποιώντας το πρότυπο "round robin" στέλνει τις αιτήσεις στους indexers.

Σε καμία περίπτωση όμως δεν επιτυγχάνεται η ισοκατανομή του φόρτου εργασίας στους indexers που προκύπτει με τη χρήση broker. Αυτό συμβαίνει γιατί οι indexers συνδέονται με συγκεκριμένους shippers, με αποτέλεσμα στην περίπτωση που κάποιοι κόμβοι παράγουν δεδομένα χρήσης με μεγαλύτερο ρυθμό, κάποιοι indexers πιθανόν να εργάζονται υπερβολικά, ενώ άλλοι καθόλου. Αυτό δε μπορεί να συμβεί στη χρήση broker, αφού τα δεδομένα συγκεντρώνονται στο broker, και από εκεί μοιράζονται στους indexers.

[16] http://www.haproxy.org/



Εικόνα 10: Αρχιτεκτονική συστήματος με χρήση του logstash-forwarder ως shipper



Εικόνα 11: Αρχιτεκτονική συστήματος με χρήση του logstash-forwarder και load balancer

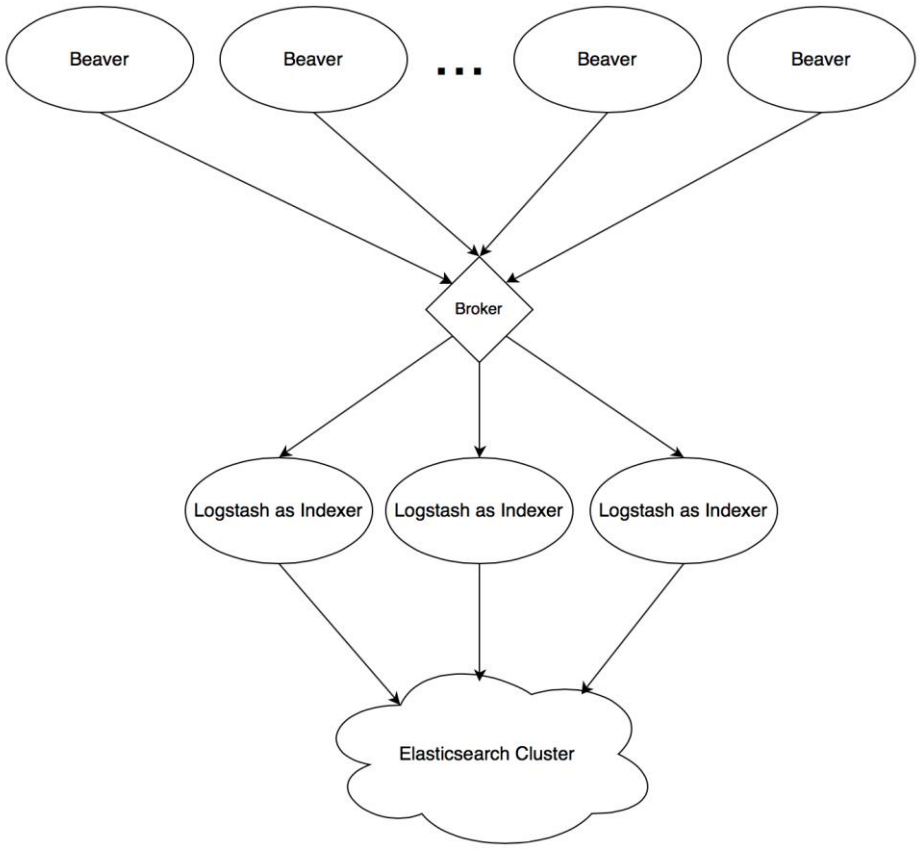
4.3.2 Beaver

Το Beaver είναι ένα έργο ανοιχτού κώδικα γραμμένο στη γλώσσα προγραμματισμού Python και πρόκειται για ένα shipper deamon, που δίνει τη δυνατότητα αποστολής δεδομένων χρήσης από τοπικά αρχεία στο Logstash χρησιμοποιώντας ως μέσο μετάδοσης ένα από τα Redis, ZeroMQ, rabbitMQ καθώς και τα πρωτόκολλα TCP και UDP. Επίσης μπορεί να χρησιμοποιηθεί και το πρωτόκολλο HTTP επιτρέποντας και αποστολή απευθείας στο Elasticsearch. Παρέχει δυνατότητα ssh tuneling κατά το ξεκίνημα δίνοντας τη δυνατότητα κρυπτογραφημένης σύνδεσης με ZeroMQ και Redis. Επίσης μπορεί να ενώσει

τα δεδομένα χρήσης πολλαπλών γραμμών σε ενιαίο μήνυμα, κάτι που είναι απαραίτητο όταν συνδυάζεται με broker.

Η κύρια εφαρμογή του Beaver γίνεται με τη χρήση brokers, όπως φαίνεται στην εικόνα 12, δίνοντας εναλλακτική επιλογή σε όσους χρειάζονται broker στη αρχιτεκτονική τους αλλά ταυτόχρονα θέλουν shippers που δεν καταναλώνουν πολλούς πόρους. Καθώς τα πλεονεκτήματα της χρήση broker αναφέρθηκαν πιο πάνω, δε θα γίνει οποιαδήποτε περαιτέρω ανάλυση.

Συγκριτικά με το logstash-forwarder, το Beaver είναι πιο αργό και δεν παρέχει μηχανισμό αντίστοιχο του "receipt acknowledgement", ούτε συμπίεση δεδομένων.



Εικόνα 12: Αρχιτεκτονική συστήματος με χρήση broker και του Beaver ως shipper

4.4 Περιληπτικά

Μέσα από την ανάλυση των τριών εναλλακτικών επιλογών γίνεται αντιληπτό αυτό που αναφέρθηκε πιο πάνω, ότι δεν υπάρχουν σωστές ή λάθος επιλογές. Αναμφίβολα δεν υπάρχει το τέλειο εργαλείο όσον αφορά τους shippers. Συνήθως το logstash ως shipper προτιμάται όταν πρόκειται για ισχυρούς κόμβους, ώστε να μην επηρεάζει την επίδοση τους και κόμβους στους οποίους παράγονται δεδομένα χρήσης με διάφορους τρόπους. Αν υπάρχει περιθώριο ίσως να γίνει και μερική επεξεργασία των δεδομένων με φίλτρα στους shippers παρόλο που αυτό είναι σπάνιο. Ο logstash-forwarder αν και δεν επιτρέπει την χρήση broker

χρησιμοποιείται ευρέως ακόμη και σε μεγάλες συστοιχίες υπολογιστών με περισσότερους από 100 κόμβους, και αυτό λόγω του εξαιρετικού πρωτοκόλλου που χρησιμοποιεί. Συνήθως όμως στους κόμβους αυτούς τρέχουν παρόμοιες εφαρμογές, με αποτέλεσμα η παραγωγή δεδομένων χρήσης να είναι περίπου η ίδια σε κάθε κόμβο. Το Beaver είναι μια προσπάθεια να προσφέρει ότι δεν έχει ο logstash-forwarder, μετάδοση μέσω broker.

5

Εφαρμογή

Στο κεφάλαιο θα παρουσιαστεί η εφαρμογή του centralized logging στη cloud πλατφόρμα "Synnefo", χρησιμοποιώντας τα εργαλεία της ELK stack. Αρχικά θα γίνει μια σύντομη περιγραφή του Synnefo και ακολούθως θα σχολιαστεί η αρχιτεκτονική συστήματος που θα επιλέγαμε για τη συγκεκριμένη περίπτωση με βάση όσα αναλύθηκαν στο προηγούμενο κεφάλαιο. Ακολούθως θα γίνει παρουσίαση ενός πρωτότυπου που δημιουργήσαμε, με βάση τα δεδομένα χρήσης που παράγονται στο Synnefo. Σκοπός είναι να δείξουμε τις δυνατότητες που μπορεί να προσφέρει ένα τέτοιο σύστημα στους διαχειριστές και προγραμματιστές του Synnefo.

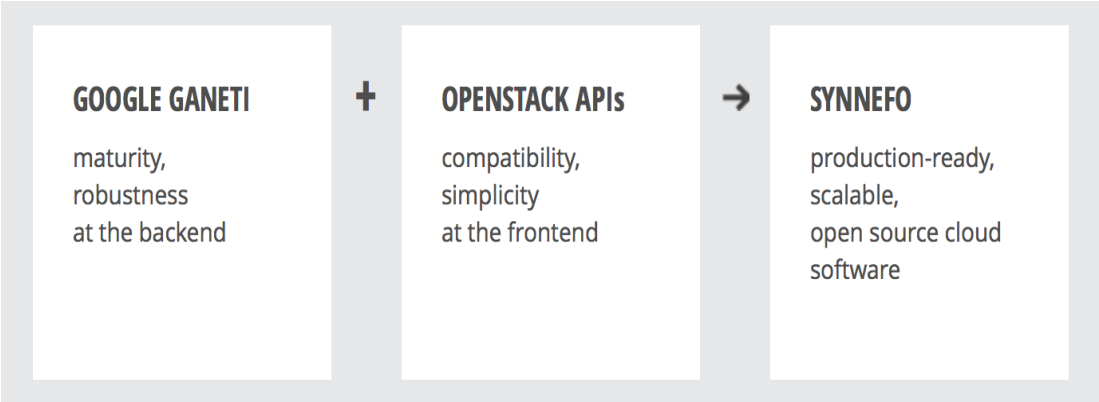
5.1 Synnefo

Το Synnefo αποτελεί μια ανοιχτού κώδικα "cloud stack" που παρέχει υπηρεσίες Compute (υπολογιστική ισχύ μέσω εικονικών μηχανών), Network (εικονικά δίκτυα), Image, Volume and Storage (χώρο αποθήκευσης), παρόμοιες με αυτές που προσφέρονται από την Amazon Web Services (AWS). Δημιουργήθηκε και συνεχίζει να αναπτύσσεται από το GRNET (Greek Research and Technology Network), με σκοπό την χρήση του για την παροχή του ~Okeanos, ένα έργο που προσφέρει υπηρεσίες "Infrastructure as a service" (IaaS) στη Ελληνική ερευνητική και ακαδημαϊκή κοινότητα.

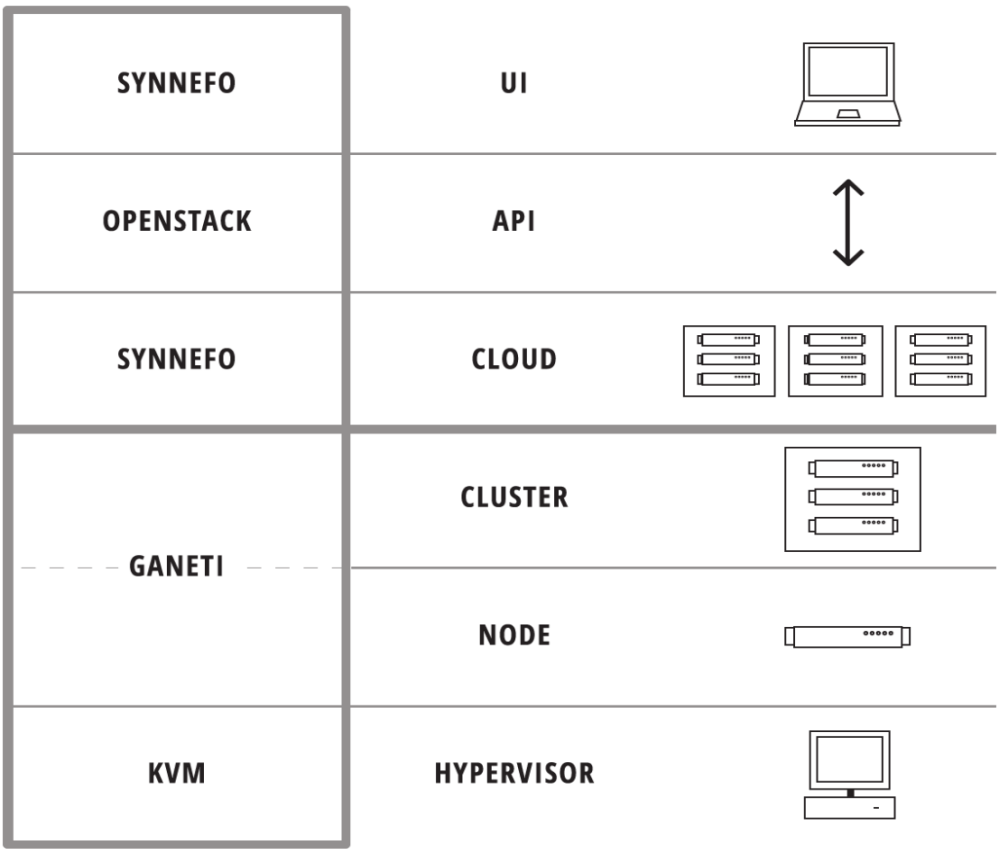
Το Synnefo βασίζεται στο Google Ganeti, ένα εργαλείο για τη διαχείριση των virtual machines (εικονικών μηχανών) σε χαμηλό επίπεδο χρησιμοποιώντας μία από τις πλατφόρμες virtualization Xen ή KVM. Συγκεκριμένα το Synnefo διαχειρίζεται πολλαπλές συστοιχίες με κόμβους (clusters) όπου εκτελείται το Ganeti, ενώ παράλληλα χρησιμοποιεί το Archipelago, για την ενοποίηση του χώρου αποθήκευση στο cloud. Το Archipelago είναι ένα κατανεμημένο στρώμα αποθήκευσης που σχεδιάστηκε με κύριο στόχο την επίλυση των προβλημάτων που προκύπτουν σε περιβάλλοντα cloud μεγάλης κλίμακας. Κύριοι στόχοι του είναι η αποδέσμευση της λογικής αποθήκευσης από την πραγματική αποθήκευση δεδομένων, η παροχή τρόπων cloning και snapshotting, η παροχή λογικής για deduplication και η παροχή των backend driver για υποστήριξη διαφόρων τεχνολογιών αποθήκευσης.

Για την εξασφάλιση συμβατότητας με τρίτες εφαρμογές (3rd-party compatibility) αναπτύχθηκαν Openstack APIs για τους χρήστες του Synnefo.

Το Synnefo ακολουθεί μια μοναδική προσέγγιση πολυεπίπεδης αρχιτεκτονικής. Όπως φαίνεται και στην εικόνα 14, υπάρχει σαφής διαχωρισμός μεταξύ του παραδοσιακού στρώματος διαχείρισης των Ganeti clusters και του στρώματος του cloud (Synnefo). Η πολυεπίπεδη αυτή προσέγγιση ενισχύει την ετοιμότητα της παραγωγής, της συντήρησης και της αναβάθμισης.

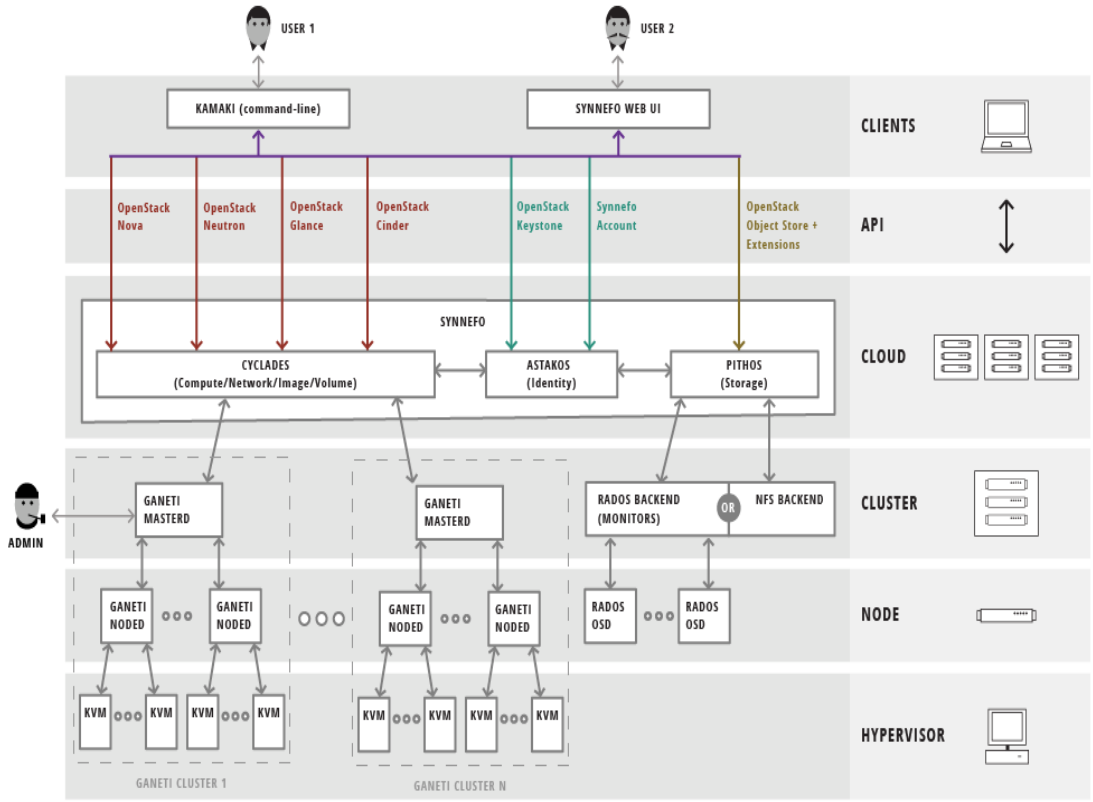


Εικόνα 13: Ganeti και Openstask APIs στο Synnefo



Εικόνα 14: Πολυεπίπεδη αρχιτεκτονική του Synnefo

Το ίδιο το Synnefo χωρίζεται σε τρία βασικά μέρη, τον Astakos, τον Pithos και τις Cyclades, τα οποία φαίνονται στη εικόνα 15.



Εικόνα 15: Διαχωρισμός του Synnefo

Ο Astakos είναι το μέρος του έργου το οποίο είναι υπεύθυνο για τη διαχείριση ταυτοποίησης (Identity management) μέσα από μια κοινή βάση χρηστών για όλο το Synnefo. Συγκεκριμένα, χειρίζεται τη δημιουργία χρηστών, τις ομάδες χρηστών, τη διαχείριση των πόρων για κάθε λογαριασμό χρήστη, την δημιουργία έργων και την λογική με την οποία οι χρήστες συμμετέχουν σε αυτά καθώς και τα ζητήματα πιστοποίησης ταυτότητας που χρειάζονται σε όλη την υποδομή.

Ο Pithos προσφέρει την υπηρεσία αποθήκευσης αρχείων στο cloud, αντίστοιχη των γνωστών εργαλείων όπως το Dropbox, box.net και Google Drive. Οι χρήστες ανεβάζουν τα αρχεία τους χρησιμοποιώντας είτε την επιφάνεια χρήσης μέσα από την ιστοσελίδα της υπηρεσίας, είτε με το πρόγραμμα-πελάτη μέσω της γραμμής εντολών (command-line), ή με άλλα προγράμματα που συγχρονίζονται με την υπηρεσία. Πρόκειται για ένα λεπτό στρώμα χαρτογράφησης των αρχείων σε content-addressable blocks, τα οποία στη συνέχεια αποθηκεύονται σε ένα backend χώρο αποθήκευσης. Τα αρχεία χωρίζονται σε blocks σταθερού μεγέθους, τα οποία κατακερματίζονται ανεξάρτητα για να δημιουργηθεί ένα μοναδικό αναγνωριστικό για κάθε block, έτσι ώστε κάθε αρχείο αντιπροσωπεύεται από μια σειρά με τα ονόματα των blocks που το αποτελούν (ένα HashMap δηλαδή).

Οι Cyclades είναι το μέρος του Synnefo που υλοποιεί τις υπηρεσίες Compute, Network, Images και Volume. Προσφέρει μια σειρά από OpenStack REST APIs: OpenStack Compute, Network, Glance και Cinder. Επί της ουσίας, οι Cyclades είναι το μέρος που διαχειρίζεται τα Ganeti Clusters μέσα από ένα σύνολο εντολών που παρέχονται από το Ganeti's Remote API (RAPI). Ο διαχειριστής μπορεί να επεκτείνει την υποδομή δυναμικά με την προσθήκη νέων Ganeti clusters. Όσον αφορά όμως την διαχείριση των λειτουργιών των virtual machines σε χαμηλό επίπεδο, όπως τη δημιουργία τους, τη μετακίνηση τους από ένα φυσικό κόμβο σε ένα άλλο κτλ, οι Cyclades δεν προσφέρουν απολύτως τίποτα. Το API προς τον τελικό χρήστη είναι περιορισμένο στο θέμα της διαχείρισης των virtual machines στο backend.

Μέσα από τη σύντομη περιγραφή του Synnefo, γίνεται άμεσα αντιληπτή η πολυπλοκότητα της διαχείρισης μιας πλατφόρμας παροχής cloud υπηρεσιών, καθώς αποτελείται από πολλά μέρη, και τρέχει σε πάρα πολλούς κόμβους. Σίγουρα η πολυεπίπεδη αρχιτεκτονική που ακολουθεί το Synnefo ευκολύνει τη συντήρηση του, στη περίπτωση βλαβών. Ταυτόχρονα όμως, η ανάγκη για παρακολούθηση των κόμβων της πλατφόρμας, είναι μεγάλη, καθώς με την παρακολούθηση οι διαχειριστές γνωρίζουν την κατάσταση της πλατφόρμας και είναι δυνατή η αποφυγή βλαβών ή η άμεση εντόπιση τους.

5.2 Προτεινόμενη αρχιτεκτονική συστήματος για το Synnefo

Στο προηγούμενο κεφάλαιο έγινε αναλυτική παρουσίαση των εναλλακτικών επιλογών που υπάρχουν για τα εργαλεία συλλογής και αποστολής των δεδομένων χρήσης από τους κόμβους καθώς και διάφορες αρχιτεκτονικές για το σύστημα.

Στην προκειμένη περίπτωση του Synnefo, λόγο του ότι υπάρχουν διαφόρων ειδών κόμβοι, στους οποίους τα δεδομένα χρήσης παράγονται με διαφορετικό ρυθμό, αναμφίβολα θα προτείναμε μια αρχιτεκτονική συστήματος που συμπεριλαμβάνει broker, ώστε να γίνεται ισοκατανομή του φόρτου εργασίας σε όλους τους indexers. Ως shipper θα προτείναμε το ίδιο το Logstash αντί του Beaver, λόγω του ότι προσφέρει περισσότερες δυνατότητες και επειδή τα μηχανήματα στα οποία τρέχει το synnefo είναι αρκετά ισχυρά, ώστε να μην επηρεάζονται ως προς την επίδοσή τους λόγω του Logstash.

5.3 Δημιουργία πρωτότυπου

Για τη δημιουργία του πρωτότυπου έγινε εγκατάσταση του Synnefo σε virtual machine στον ~Okeanos, με τη μέθοδο εγκατάστασης σε ένα κόμβο (one node installation). Η έκδοση που χρησιμοποιήθηκε ήταν η υπό ανάπτυξη έκδοση v0.15.2. Το γεγονός ότι χρησιμοποιήθηκε εγκατάσταση ενός κόμβου δεν επηρεάζει καθόλου την παρουσίαση της γενικής ιδέας και της χρησιμότητας του εργαλείου καθώς όλα τα δεδομένα χρήσης από τα διάφορα μέρη του Synnefo είναι διαθέσιμα.

Για το πρωτότυπο χρησιμοποιήθηκε η αρχιτεκτονική συστήματος που προτάθηκε στην προηγούμενη ενότητα για την εφαρμογή στο Synnefo. Ως broker επιλέχθηκε η Redis,

καθώς μας ενδιέφερε περισσότερο να εξετάσουμε τις επιδόσεις του συστήματος. Στο επόμενο κεφάλαιο θα παρουσιαστούν αναλυτικά οι μετρήσεις που έγιναν.

Στο πίνακα που ακολουθεί φαίνονται οι εικονικές μηχανές που χρησιμοποιήθηκαν για το πρωτότυπο.

Πίνακας 1: Εικονικές μηχανές που χρησιμοποιήθηκαν για το πρωτότυπο

hostname	IP	CPU	RAM	DISK
Redis	192.168.0.1	1x	2	10GB
Elastic1	192.168.0.4	2x	4	20GB
Elastic2	192.168.0.2	2x	4	20GB
Synnefo	192.168.03	2x	4	60GB

5.3.1 Δεδομένα χρήσης - logs που παράγονται σε ένα κόμβο

Τα σημαντικότερα δεδομένα χρήσης που παράγονται στο Synnefo, βρίσκονται σε αρχεία, στο γνωστό κατάλογο (directory) των Linux διανομών για δεδομένα χρήσης, /var/log και παράγονται είτε από το ίδιο το Synnefo, είτε από τα Ganeti clusters, είτε από διάφορα βοηθητικά εργαλεία που χρησιμοποιεί το Synnefo, όπως η RabbitMQ.

Σημειώνεται, ότι σε μια πραγματική εγκατάσταση του Synnefo, οι κόμβοι χωρίζονται ανάλογα με τη λειτουργία που εκτελούν, γι' αυτό και στον κατάλογο /var/log θα είναι διαθέσιμα μόνο δεδομένα χρήσης του αντίστοιχου εργαλείου - προγράμματος. Επομένως, θα πρέπει να γίνει η σωστή ρύθμιση του configuration file του shipper ανάλογα με τον κόμβο.

Τα αρχεία που χρησιμοποιήθηκαν στο πρωτότυπο φαίνονται στον πίνακα που ακολουθεί. Πέραν από δεδομένα χρήσης που παράγονται από το Synnefo, επιλέχθηκαν και δεδομένα χρήσης συστήματος (τα Syslog στο πίνακα) που αφορούν το κάθε μηχάνημα, καθώς παρέχουν σημαντική πληροφορία, ειδικά στη περίπτωση βλάβης.

Πίνακας 2: Αρχεία που χρησιμοποιήθηκαν στο πρωτότυπο

Directory	Files	Type
/var/log/synnefo	dispatcher.log	dispatcher
/var/log/archipelago	blockerb.log, blockerm.log, mapperd.log, vlmcd.log	archipelago
/var/log/ganeti	master-daemon.log, node-daemon.log	ganeti
/var/log	snf-ganeti-eventd.log	eventd-ganeti
/var/log/rabbitmq	rabbit@snf-540047.log (rabbit@<hostname>.log)	rabbitMQ

Directory	Files	Type
/var/log/apache2	access.log, error.log, other_vhosts_access.log	apache
/var/log	Syslog, kern.log, auth.log, messages, user.log, daemon.log	syslog

5.3.2 Στατιστικά στοιχεία απόδοσης ενός κόμβου

Όπως αναφέρθηκε, πέραν των δεδομένων χρήσης που παράγονται σε ένα κόμβο, σημαντική πληροφορία είναι και τα στατιστικά στοιχεία της απόδοσης του συστήματος. Ο συνδυασμός των δεδομένων χρήσης και των στατιστικών αυτών, μπορούν να δώσουν την πραγματική κατάσταση του συστήματος.

Υπάρχουν αρκετά εργαλεία τα οποία αναλαμβάνουν τη συλλογή στατιστικών στοιχείων απόδοσης, όπως το collectd [28] και το Ganglia [29], το καθένα όμως προσφέρει διαφορετικές δυνατότητες.

Το collectd είναι ένας δαίμονας που αναλαμβάνει αποκλειστικά την περιοδική συλλογή των στατιστικών στοιχείων και παρέχει μηχανισμούς για την αποθήκευση τους με διάφορους τρόπους, όπως για παράδειγμα τα RRD files. Περιλαμβάνει βελτιστοποιήσεις και χαρακτηριστικά για να χειρίζεται εκατοντάδες χιλιάδες δεδομένα. Έρχεται με πάνω από 90 plugins που αφορούν τόσο συνηθισμένα στατιστικά στοιχεία όσο και πιο εξειδικευμένα. Παρέχει πολλές επιλογές δικτύωσης (networking) και είναι επεκτάσιμο με διάφορους τρόπους.

Το Ganglia είναι ένα κατανεμημένο σύστημα παρακολούθησης υπολογιστικών συστημάτων υψηλής απόδοσης. Βασίζεται σε ιεραρχική σχεδίαση καθώς απευθύνεται κυρίως σε συστοιχίες υπολογιστών. Πρόκειται για ένα εργαλείο χωρίς ιδιαίτερες απαιτήσεις και πολύ εύκολο στην εγκατάσταση και στη χρήση του. Σε συνδυασμό με το Gaglia Web, την εφαρμογή διαδικτύου που χρησιμοποιείται για προβολή των στατιστικών, αποτελεί μια ολοκληρωμένη πρόταση παρακολούθησης συστημάτων. Χρησιμοποιείται ευρέως από πολλούς οργανισμούς, ενώ έχει δοκιμαστεί σε συστοιχίες με περισσότερους από 2000 κόμβους.

Για το πρωτότυπο, προτιμήθηκε το collectd λόγω της συνεχής ανάπτυξης του αντίστοιχου input plugin για το Logstash από την ομάδα του Elasticsearch. Μετά από πολλά αιτήματα χρηστών του logstash, οι οποίοι ήθελαν να χρησιμοποιούν ενιαίο σύστημα για τη συλλογή μετρικών και δεδομένων χρήσης, η ομάδα του elasticsearch ασχολήθηκε ιδιαίτερα με το collectd ώστε να υποστηριχτεί από το Logstash και το Kibana. Η προβολή δεδομένων χρήσης και στατιστικών στοιχείων απόδοσης στο ίδιο dashboard του Kibana σίγουρα προσφέρει μια ιδιαίτερη ευελιξία.

5.3.3 Configuration file του Shipper

Στη ενότητα αυτή θα παρουσιαστεί η σύνθεση του configuration file του Logstash ως shipper που χρησιμοποιήθηκε στο πρωτότυπο. Στη τελευταία ενότητα παρατίθεται ολοκληρωμένο το configuration file του shipper, γι' αυτό και στα σημεία που χρησιμοποιούνται κάποια plugins κατ' επανάληψη δεν περιλαμβάνεται ολόκληρο το μέρος.

Για τη συλλογή των δεδομένων χρήσης από αρχεία, χρησιμοποιήθηκε το input plugin "file", ενώ για τη συλλογή των στατιστικών που καταγράφονται μέσω του collectd, χρησιμοποιήθηκε το plugin "udp" χρησιμοποιώντας το αντίστοιχο codec "collectd".

Με βάση τα πιο πάνω, παρατίθεται ένα μέρος του input στο configuration file του shipper καθώς χρησιμοποιείται κατά επανάληψη το input plugin file.

```
input{

    udp {
    host => "127.0.0.1"
    port => 25826
    buffer_size => 1452
    codec => collectd { }
    type => "collectd"
    }

    file{
        path =>
["/var/log/syslog","/var/log/alternatives.log","/var/log/dea
emon.log","/var/log/debug","/var/log/kern.log","/var/log/use
r.log","/var/log/auth.log","/var/log/dmesg"]
        type => "syslog"
        }
. . .

    file{
        path => "/var/log/apache2/*.log"
        type => "apache"
        }

}
```

Όσον αφορά τα filters, χρησιμοποιήθηκαν μόνο τα plugins drop και multiline.

Το plugin drop χρησιμοποιείται για την απόρριψη μηνυμάτων και συνήθως χρησιμοποιείται με το προγραμματιστικό μοντέλο if-then-else για απόρριψη μηνυμάτων υπό κάποιο όρο (conditional). Στην προκειμένη περίπτωση, χρησιμοποιήθηκε για την απόρριψη των κενών μηνυμάτων, αυτών δηλαδή που αντιστοιχούν στις κενές γραμμές στα αρχεία καταγραφής δεδομένων χρήσης.

Το plugin multiline χρησιμοποιείται για την ομαδοποίηση δεδομένων χρήσης πολλαπλών γραμμών σε ενιαίο μήνυμα. Το φίλτρο αναγνωρίζει μηνύματα που ακολουθούν συγκεκριμένο "pattern" που ορίζεται με την αντίστοιχη επιλογή, ενώ στη συνέχεια τα

ομαδοποιεί με προηγούμενα ή με επόμενα μήνυμα ανάλογα και πάλι με βάση της αντίστοιχης επιλογής. Χρησιμοποιείται με το προγραμματιστικό μοντέλο "if-then" ώστε να επιτυγχάνεται η ομαδοποίηση συγκεκριμένου τύπου δεδομένων (type).

Στο σημείο αυτό παρατίθενται τα filter στο configuration file του shipper. Όπως φαίνεται μόνο τρία είδη αρχείων, περιέχουν μηνύματα πολλαπλών γραμμών. Χρησιμοποιώντας συνθήκες ανάλογα με το κάθε είδος, επιτυγχάνεται η ομαδοποίηση των δεδομένων.

```
filter {
    if [message] ==”” {
        drop { }
    }

    if [type] == "archipelago" {
        multiline{
            pattern => "(^XSEG)"
                what => "next"
        }
    }

    else if [type] == "dispatcher" {
        multiline{
            pattern =>
"(^DEBUG:)|(^WARNING:)|(^INFO:)|(^CRITICAL:)(^ERROR:)"
             what => "previous" }
    }

    else if [type] == "rabbitMQ" {
        multiline{
            pattern => "(^=INFO REPORT====)"
             what => "next" }
    }
}
```

Σύμφωνα με την αρχιτεκτονική συστήματος που επιλέχθηκε, τα δεδομένα πρέπει να αποστέλνονται στο Redis-server. Το output χρησιμοποιώντας το αντίστοιχο output plugin για την Redis, φαίνεται πιο κάτω.

```
output{
    redis {
        host => “192.168.0.1”
        data_type => "list"
        key => "logstash"
    }
}
```

5.3.4 Configuration file για Indexer

Στην ενότητα αυτή θα παρουσιαστεί η σύνθεση του configuration file του Logstash ως indexer. Στην τελευταία ενότητα παρατίθεται ολοκληρωμένο το configuration file του indexer, γι' αυτό και στα σημεία που χρησιμοποιούνται κάποια plugins κατ' επανάληψη δεν περιλαμβάνεται ολόκληρο το μέρος.

Προφανώς, ως είσοδο στους indexers, χρησιμοποιείται το αντίστοιχο input plugin για την Redis. Ακολουθεί το input στο για το configuration file ενός indexer του συστήματος.

```
input {
  redis {
    host => "192.168.0.1"
    data_type => "list"
    key => "logstash"
  }
}
```

Στη συνέχεια παρατίθεται μέρος από τα filters.

```
filter {
    if [type] == "syslog" {

        if [path] == "/var/log/alternatives.log" {
                        grok {
                                match => {"message" =>
"update-alternatives %{TIMESTAMP_ISO8601:timestamp}:
%{GREEDYDATA:info}"}
                        }
                }

        else if [path] == "/var/log/dmesg" {
                        grok {
                                match => {"message" =>
"%{BASE16FLOAT:ring_buffer}\] %{GREEDYDATA:info}"}
                        }
                }

        else {
            grok {
                match => {"message" => "%{SYSLOGBASE}
%{GREEDYDATA:info}"}
            }
        }
    }
## ----------------------------------------
    else if [type] == "archipelago" {
        grok {
        match => {"message" => "XSEG\[%{GREEDYDATA:XSEG}\]:
.?*: %{DAY} %{GREEDYDATA:timestamp}
.?*:\n\t%{GREEDYDATA:info}" }
        }
    }
```

```
. . .

## ----------------------------------------
    else if [type] == "apache" {
                grok {
               match => { "message" => "%{COMBINEDAPACHELOG}"}
        }
                geoip {
                    source =>"clientip"
                 }
    }
## ----------------------------------------
    date {
        timezone => "Europe/Athens"
        match => [ "timestamp" , "MMM dd HH:mm:ss YYYY",
"MMM  dd HH:mm:ss YYYY","YYYY-MM-dd HH:mm:ss,SSS","YYY-MM-DD
HH:mm:ss", "dd-MMM-YYYY::HH:mm:ss","MMM dd HH:mm:ss","MMM
dd HH:mm:ss","dd/MMM/yyyy:HH:mm:ss Z" ]
        remove_field => [ "timestamp" ]
      }
}
```

Καθώς έγινε αναλυτική περιγραφή των δυνατοτήτων του grok στο κεφάλαιο 3, δεν περιγράφεται το συγκεκριμένο plugin. Όπως φαίνεται, γίνεται συνεχής χρήση συνθηκών για να εφαρμοστεί το κατάλληλο φίλτρο grok στα δεδομένα χρήσης ανάλογα με το είδος τους (type). Στην περίπτωση των syslog, χρειάστηκε να χρησιμοποιηθεί το πεδίο "path" για να γίνει διαχωρισμό των δεδομένων ανάλογα με το αρχείο που προέρχονται. Βέβαια, θα μπορούσε να χρησιμοποιηθεί διαφορετικό 'type" για δεδεομένα από κάθε αρχείο, αλλά προτιμήθηκε η κοινή ονομασία "syslog".

Στη περίπτωση των Apache μηνυμάτων χρησιμοποιήθηκε το plugin geoip, το οποίο προσθέτει πληροφορίες για τη γεωγραφική θέση των διευθύνσεων IP που εισάγονται στο πεδίο source. Στη συγκεκριμένη περίπτωση δίνεται το πεδίο "clientip" που υπάρχει στα apache μηνύματα, ώστε να παράγεται η γεωγραφική θέση των επισκεπτών του Synnefo για στατιστικούς λόγους.

Στο τέλος, υπάρχει το φίλτρο "date" το οποίο αναλαμβάνει την ενημέρωση του πεδίου "@timestamp". Το πεδίο αυτό περιέχει ημερομηνία και ώρα και χρησιμοποιείται για την ταξινόμηση των δεδομένων. Βάση αυτού του πεδίου γίνεται και η προβολή των δεδομένων χρήσης. Το logstash, ενημερώνει αυτό το πεδίο κατά την συλλογή των δεδομένων, δηλαδή στην προκειμένη περίπτωση, μέσω των shipper, τοποθετώντας την υφιστάμενη ώρα και ημερομηνία, δηλαδή τη στιγμή της συλλογής. Καθώς, τα δεδομένα χρήσης περιέχουν το πραγματικό timestamp, με την ώρα και ημερομηνία που παράχθηκε το κάθε μήνυμα, χρησιμοποιείται το φίλτρο date, ώστε να ενημερωθεί το πεδίο "@timestamp" με το πραγματικό. Στον πίνακα, υπάρχουν όλες οι πιθανές μορφές timestamp, που χρησιμοποιούνται από τα δεδομένα χρήσης που συλλέγει το σύστημα. Χρειάστηκε, βάση του grok, να ονομαστεί ως "timestamp" το αντίστοιχο μέρος του μηνύματος, έτσι ώστε να

αναγνωρίζεται στη συνέχεια από το "date". Καθώς το συγκεκριμένο πεδίο δε χρειάζεται πλέον, αφαιρείται μέσω του remove_field.

Τέλος, χρησιμοποιώντας το output plugin "elasticsearch", τα δεδομένα χρήσης αποθηκεύονταν στο Elasticsearch. Χρησιμοποιήθηκε η προεπιλεγμένη τιμή για index του plugin, όπου δημιουργεί indexs της μορφής "logstash-%{+YYYY.MM.dd}", δηλαδή ένα index ανά ημέρα. Καθώς είναι η προεπιλεγμένη τιμή δεν χρειάζεται να οριστεί.

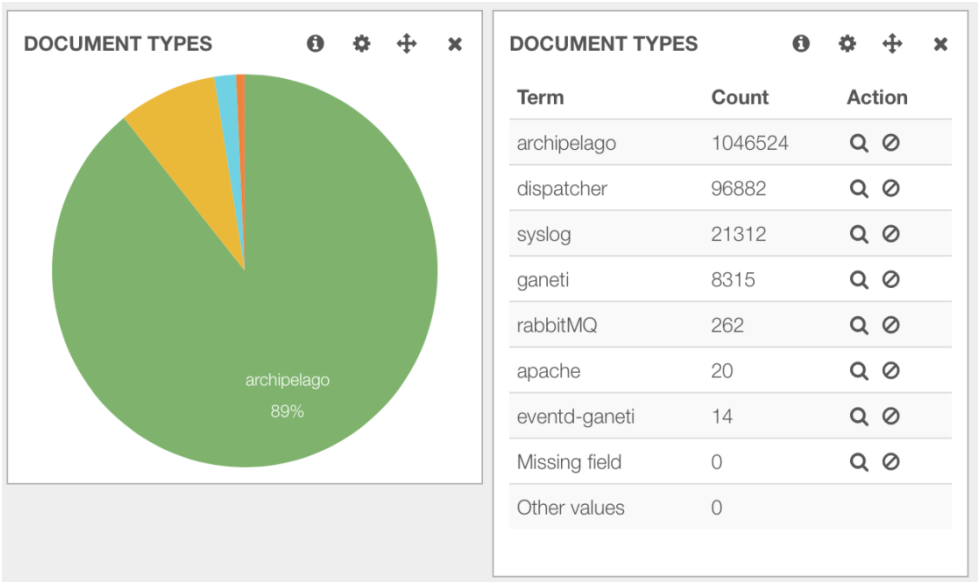
```
output {
  elasticsearch {
    host => "localhost"
}
```

5.3.5 Ανάλυση δεδομένων μέσω Kibana 3

Στην ενότητα αυτή θα γίνει μια συνοπτική παρουσίαση των δυνατοτήτων που προσφέρει η ανάλυση των δεδομένων χρήσης μέσω του Kibana. Όπως έχει αναφερθεί στο κεφάλαιο 3, το Kibana διαθέτει έτοιμα dashboards, τα οποία μπορούν να χρησιμοποιηθούν άμεσα, ταυτόχρονα όμως η δημιουργία νέων dashboards είναι αρκετά εύκολη.

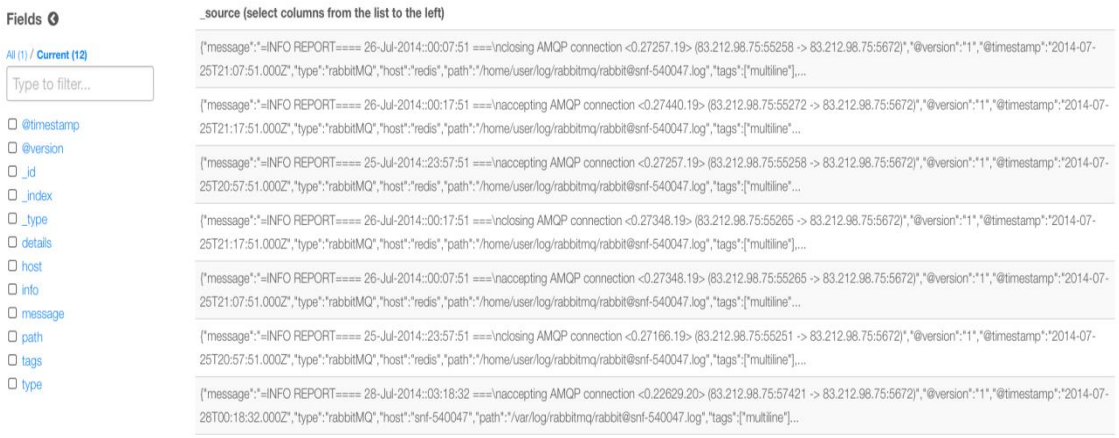
Για τους σκοπούς του συγκεκριμένου πρωτότυπου χρησιμοποιήσαμε τα έτοιμα dashboards, προσθέτοντας κάποια επιπλέον γραφήματα. Στη συνέχεια επεξηγούνται μέρη από τα dashboards όπου φαίνεται η ευχρηστία που προσφέρει η χρήση του συγκεκριμένου εργαλείου.

Στην πιο κάτω εικόνα, φαίνονται όλα τα είδη δεδομένων χρήσης που υπήρχαν αποθηκευμένα στο Elasticsearch μια δεδομένη στιγμή. Αριστερά υπάρχει μια pie chart, ώστε να δίνει αίσθηση του όγκου των δεδομένων ανάλογα με το είδος, ενώ δεξιά στο πίνακα υπάρχουν επιλογές για εφαρμογή φίλτρων για κάθε είδος, είτε για επιβολή του συγκεκριμένου είδους στις αναζητήσεις είτε για αποκλεισμό του από τις αναζητήσεις.



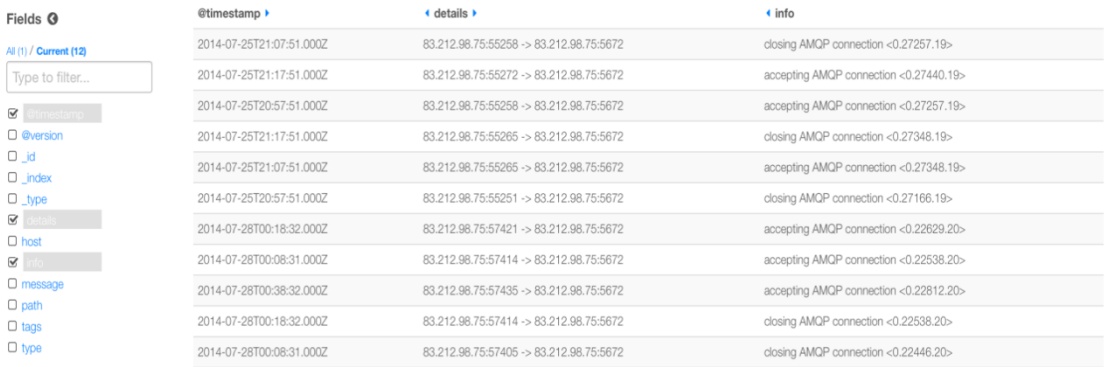
Εικόνα 16: Παραδείγματα στο Kibana

Επιλέγοντας για παράδειγμα την εφαρμογή φίλτρου για παρουσίαση μόνο των μηνυμάτων τύπου rabbitMQ, εμφανίζονται τα documents όπως είναι αποθηκευμένα στο Elasticsearch, με όλα τα πεδία και με σειρά εμφάνισης βάση του πεδίου @timestamp. Προφανώς, η αναζήτηση κάποιας πληροφορίας θα ήταν αρκετά δύσκολη.



Εικόνα 17: Προβολή αποτελεσμάτων στο Kibana

Επιλέγοντας ο χρήστης, από αριστερά τα πεδία που τον ενδιαφέρουν, παρουσιάζονται μόνο αυτά, με αποτέλεσμα να μπορεί να διαβάσει πιο εύκολα τις πληροφορίες. Σημειώνεται ότι τα πεδία που υπάρχουν, είναι αυτά που δημιουργήθηκαν μέσω του Logstash, χρησιμοποιώντας το φίλτρο grok. Εδώ φαίνεται, η τεράστια σημασία του parsing των δεδομένων χρήσης μέσω του Logstash.



Εικόνα 18: Προβολή αποτελεσμάτων στο Kibana με επιλογή πεδίων

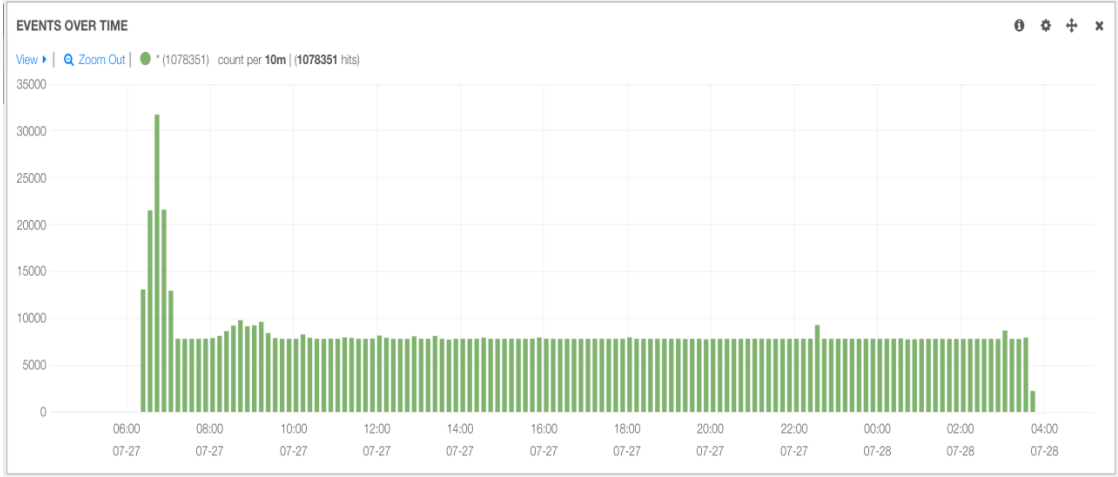
Επίσης, χρησιμοποιώντας τα Lucene queries, ο χρήστης μπορεί να κάνει αναζητήσεις πολύ γρήγορα βάση των πεδίων των δεδομένων χρήσης.

Πραγματοποιώντας το ερώτημα που φαίνεται στη εικονα 19, παρουσιάζονται όλα τα δεδομένα χρήσης που προέρχονται από τα αρχεία /var/log/ganeti/master-deamon.log. Λόγω των δυνατοτήτων που προσφέρουν τα Lucene queries, ο χρήστης δεν χρειάζεται να γράψει ολόκληρο το κατάλογο του αρχείου, αρκεί να γράψει τη λέξη που θέλει να υπάρχει στο συγκεκριμένο πεδίο. Στην προκειμένη περίπτωση, το όνομα του αρχείου, αρκεί για να ξεχωρίσει τα μηνύματα τύπου "ganeti".

Εικόνα 19: Πραγματοποίηση ερωτήματος μέσω του Kibana

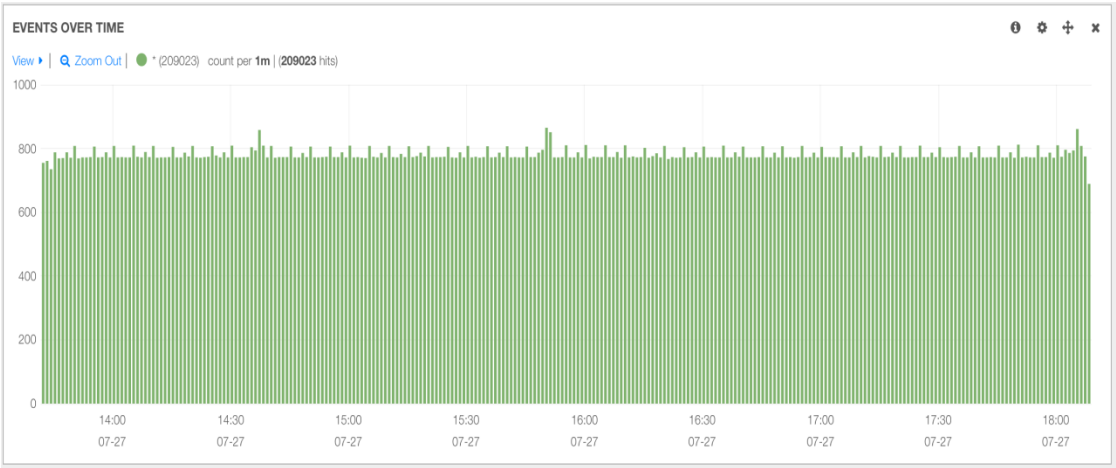
Το συγκεκριμένο query θα επιστρέψει μηνύματα από όλους τους κόμβους που διαθέτουν τέτοια αρχεία. Αν η ζητούμενη πληροφορία βρίσκεται σε συγκεκριμένο κόμβο, το query διαμορφώνεται ως εξής: type:“ganeti” AND path:“master-deamon.log” AND host:“<hostname>”, ώστε η αναζήτηση να επιστρέψει μηνύματα μόνο από αυτόν τον κόμβο.

Στην εικόνα που ακολουθεί, φαίνεται ένα γράφημα όπου παρουσιάζεται ο αριθμός των δεδομένων χρήσης σε σχέση με το χρόνο.



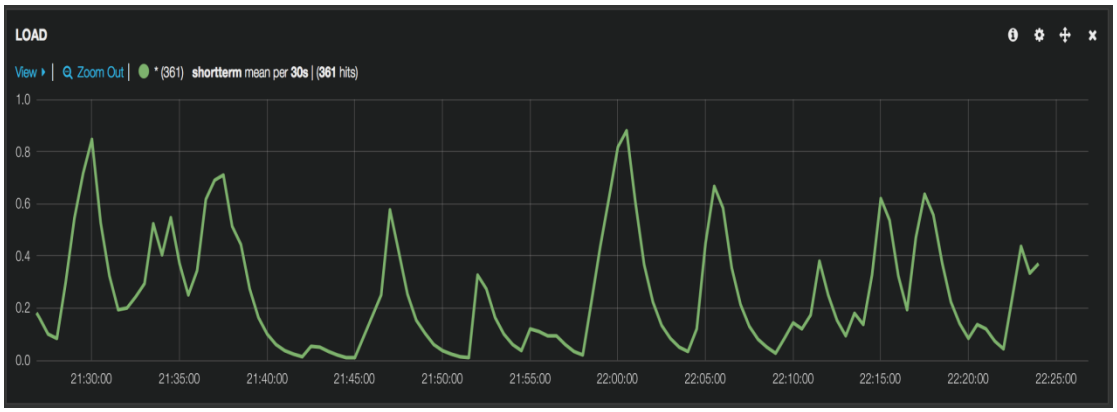
Εικόνα 20: Γράφημα στο Kibana

Χρησιμοποιώντας πολύ απλά το δείκτη (mouse), μπορεί ο χρήστης να επιλέξει την περίοδο του χρόνου που τον ενδιαφέρει, ώστε να περιορίσει τα αποτελέσματα με αυτό το φίλτρο. Με βάση την εικόνα 20, επιλέγοντας το διάστημα μεταξύ 14:00 και 18:00 προκύπτει το αποτέλεσμα που φαίνεται στην εικόνα 21.



Εικόνα 21: Εφαρμογή φίλτρου στο γράφημα

Τέλος, στη εικόνα 22, φαίνεται πως μπορεί το Kibana να παρουσιάσει τα στατιστικά στοιχεία απόδοσης που συλλέγονται μέσω του collectd. Στη προκειμένη περίπτωση παρουσιάζεται η μέτρηση "Load".



Εικόνα 22: Γράφημα του "Load" στο Kibana

5.4 Παράρτημα με ολοκληρωμένα αρχεία κώδικα

Στη ενότητα αυτή παραθέτονται τα δύο configuration files, του shipper και του indexer, που χρησιμοποιήθηκαν στο πρωτότυπο.

5.4.1 Configuration file shipper: shipper.conf

```
input{

    udp {
    host => "127.0.0.1"
    port => 25826
    buffer_size => 1452
    codec => collectd { }
    type => "collectd"
    }

    file{
        path =>
["/var/log/syslog","/var/log/alternatives.log","/var/log/dea
emon.log","/var/log/debug","/var/log/kern.log","/var/log/use
r.log","/var/log/auth.log","/var/log/dmesg"]
        type => "syslog"
        }

    file{
        path => "/var/log/apache2/*.log"
        type => "apache"
        }

    file{
    path => "/var/log/synnefo/dispatcher.log"
    type => "dispatcher"
    }

    file{
        path => "/var/log/archipelago/*.log"
        type => "archipelago"
        }

    file{
        path => ["/var/log/ganeti/master-
daemon.log","/var/log/ganeti/node-daemon.log"]
        type => "ganeti"
        }

    file{
        path => "/var/log/snf-ganeti-eventd.log"
        type => "eventd-ganeti"
        }
```

```
    file{
        path => "/var/log/rabbitmq/rabbit@snf-540047.log"
        type => "rabbitMQ"
        }
}

filter {
    if [message] =="" {
        drop { }
    }

    if [type] == "archipelago" {
        multiline{
            pattern => "(^XSEG)"
                what => "next"
        }
    }

    else if [type] == "dispatcher" {
        multiline{
            pattern =>
"(^DEBUG:)|(^WARNING:)|(^INFO:)|(^CRITICAL:)(^ERROR:)"
                what => "previous" }
    }

    else if [type] == "rabbitMQ" {
        multiline{
            pattern => "(^=INFO REPORT====)"
                what => "next" }
    }
}

output{
    redis {
        host => "192.168.0.1"
        data_type => "list"
        key => "logstash"
    }
}
```

5.4.2 Configuration file indexer: indexer.conf

```
input {
  redis {
    host => "192.168.0.1"
    data_type => "list"
    key => "logstash"
  }
}

filter {
    if [type] == "syslog" {

        if [path] == "/var/log/alternatives.log" {
                        grok {
                                match => {"message" =>
"update-alternatives %{TIMESTAMP_ISO8601:timestamp}:
%{GREEDYDATA:info}"}
                        }
                }

        else if [path] == "/var/log/dmesg" {
                        grok {
                                match => {"message" =>
"%{BASE16FLOAT:ring_buffer}\] %{GREEDYDATA:info}"}
                        }
                }

        else {
            grok {
                match => {"message" => "%{SYSLOGBASE}
%{GREEDYDATA:info}"}
            }
        }
    }
## ----------------------------------------
    else if [type] == "archipelago" {
        grok {
        match => {"message" => "XSEG\[%{GREEDYDATA:XSEG}\]:
.?*: %{DAY} %{GREEDYDATA:timestamp}
.?*:\n\t%{GREEDYDATA:info}" }
        }
    }

## ----------------------------------------
    else if [type] == "dispatcher" {
        grok {
        match => {"message" =>
"%{TIMESTAMP_ISO8601:timestamp} \w+
%{GREEDYDATA:application} \[%{GREEDYDATA:event_level}\]
%{GREEDYDATA:info}" }
        }
    }
```

```
## ----------------------------------------
    else if [type] == "eventd-ganeti" {
        grok {
            match => {"message" => "%{GREEDYDATA:timestamp}
%{GREEDYDATA:program} INFO: %{GREEDYDATA:info}" }
        }
    }

## ----------------------------------------
    else if [type] == "ganeti" {
        grok {
            match => {"message" => "%{GREEDYDATA:timestamp}:
%{GREEDYDATA:kind} pid=%{GREEDYDATA:pid} INFO
%{GREEDYDATA:info}" }
        }
    }

## ----------------------------------------
    else if [type] == "rabbitMQ" {
        grok {
            match => {"message" => "=INFO REPORT====
%{GREEDYDATA:timestamp} ===\n%{GREEDYDATA:info}
\(%{GREEDYDATA:details}\).*?" }
        }
    }
## ----------------------------------------
    else if [type] == "apache" {
        grok {
            match => { "message" => "%{COMBINEDAPACHELOG}"}
        }
        geoip {
            source =>"clientip"
        }
    }
## ----------------------------------------
    date {
        timezone => "Europe/Athens"
        match => [ "timestamp" , "MMM dd HH:mm:ss YYYY",
"MMM  dd HH:mm:ss YYYY","YYYY-MM-dd HH:mm:ss,SSS","YYY-MM-DD
HH:mm:ss", "dd-MMM-YYYY::HH:mm:ss","MMM dd HH:mm:ss","MMM
dd HH:mm:ss","dd/MMM/yyyy:HH:mm:ss Z" ]
        remove_field => [ "timestamp" ]
     }
}

output {
  elasticsearch {
    host => "localhost"
    }
}
```

6

Επιδόσεις συστήματος - Μετρήσεις

Στο κεφάλαιο αυτό θα παρουσιαστούν διάφορες μετρήσεις που έγιναν με σκοπό τη μελέτη της κλιμάκωσης του συστήματος που επιλέχθηκε για την συγκεκριμένη εφαρμογή. Ενδιαφέρον παρουσιάζει η εξέταση του χρόνου που απαιτείται ώστε τα δεδομένα χρήσης που παράγονται στους κόμβους, να μεταφερθούν και να είναι διαθέσιμα στο Elasticsearch καθώς επίσης και επιδόσεις του Elasticsearch στις αναζητήσεις.

6.1 Πειραματική διάταξη

Για της διάφορες μετρήσεις που πραγματοποιήθηκαν, χρησιμοποιήθηκαν τρεις εικονικές μηχανές, τα χαρακτηριστικά των οποίων παρουσιάζονται στον πιο κάτω πίνακα.

Πίνακας 3: Εικονικές μηχανές που χρησιμοποιήθηκαν στις μετρήσεις

hostname	CPU	RAM	DISK
Host-1	1x	2	10GB
Host-2	2x	4	20GB
Host-3	2x	4	20GB

6.2 Επιδόσεις Indexing δεδομένων χρήσης

Το κύριο πλεονέκτημα της αρχιτεκτονικής που επιλέχθηκε, είναι ότι επιτυγχάνεται η κατανομή του φόρτου εργασίας σε όλους τους indexers μέσω του broker. Θέλοντας να εξετάσουμε πώς επηρεάζει τις επιδόσεις του συστήματος η προσθήκη ενός επιπλέον indexer, πήραμε μετρήσεις χρησιμοποιώντας ένα indexer και ακολούθως με δύο indexer. Παράλληλα, μελετήσαμε κατά πόσο επηρεάζει την επίδοση του συστήματος η διατήρηση αντιγράφων (replicas) των shards στο Elasticsearch. Όπως έχει αναφερθεί, το Elasticsearch έχει προεπιλεγμένη επιλογή τη διατήρηση ενός αντιγράφου για κάθε shards του index που δημιουργεί. Τροποποιώντας την επιλογή αυτή, πήραμε μετρήσεις με ένα κόμβο Elasticsearch χωρίς διατήρηση αντιγράφων και ακολούθως με 2 κόμβους Elasticsearch διατηρώντας την προεπιλεγμένη επιλογή.

Για τις μετρήσεις αυτές χρησιμοποιήθηκε το input plugin "generator" του Logstash το οποίο δίνει τη δυνατότητα παραγωγής μεγάλου όγκου δεδομένων χρήσης, σε μικρό χρονικό διάστημα, το οποίο δημιουργήθηκε αποκλειστικά για έλεγχο των επιδόσεων του συστήματος.

Για τις συγκεκριμένες μετρήσεις τα μηνύματα ήταν απλά μηνύματα "Let's Rock!", όπως φαίνεται στο πιο κάτω παράδειγμα.

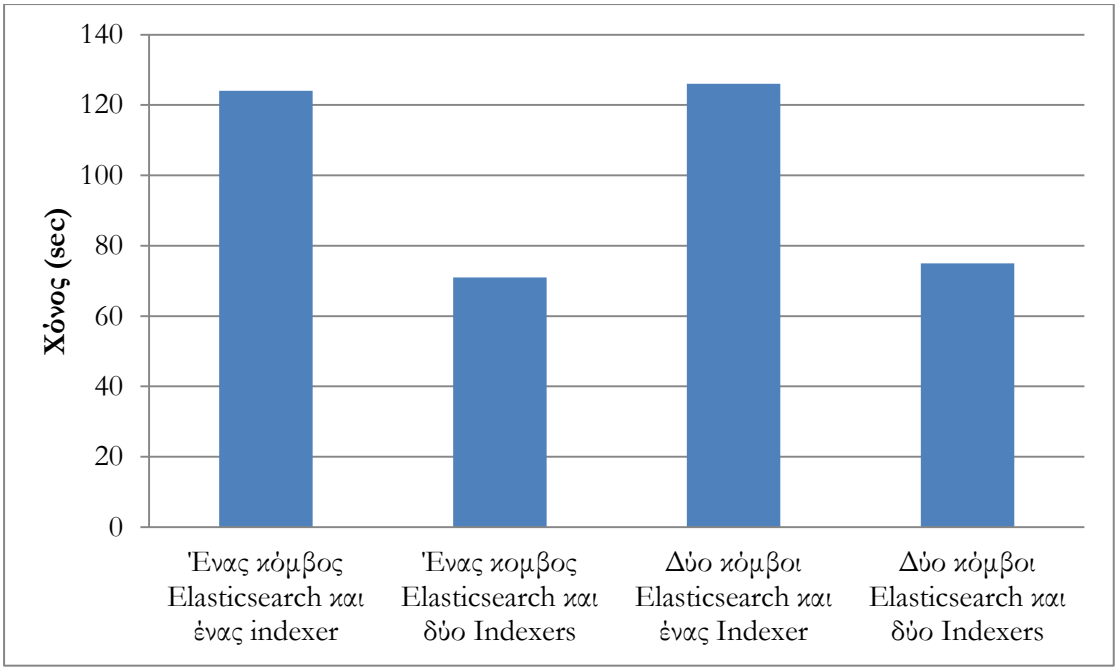
```
input{
    generator {
    count => 50000
    message => "Let's Rock!"
    type => "test"
    threads => "1"
    }
}
```

Το Logstash που χρησιμοποιούσε το πιο πάνω input, που παρήγαγε δηλαδή τα μηνύματα, έτρεχε στο κόμβο Host-1, όπου έτρεχε και ο Redis-server. Οι indexers έτρεχαν στους κόμβους Host-2 και Host-3 παράλληλα με το Elasticsearch.

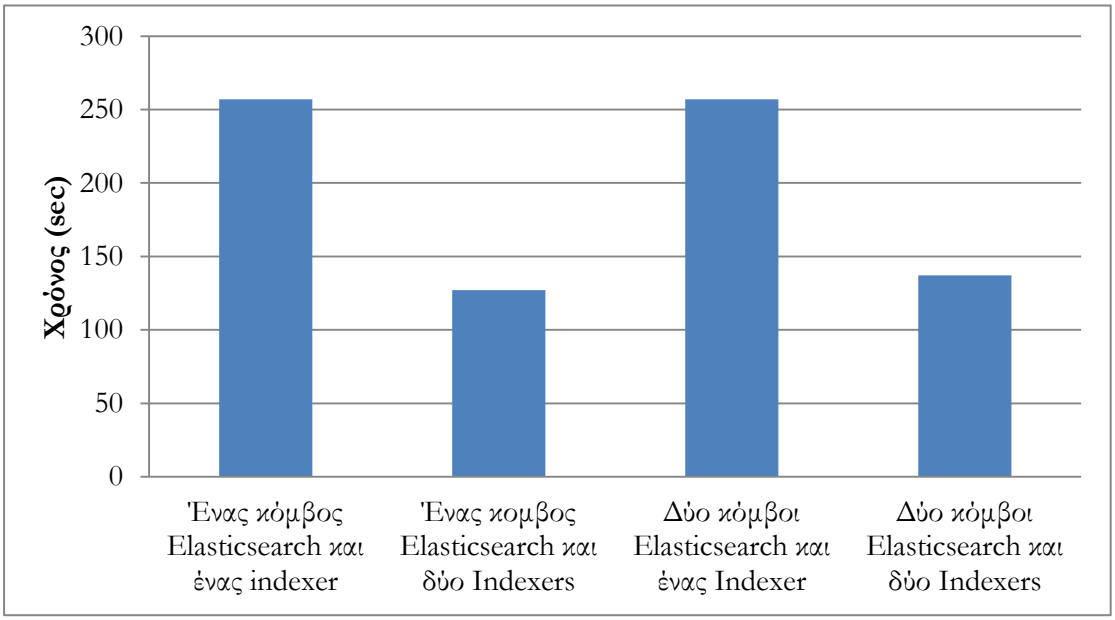
Πραγματοποιήθηκαν μετρήσεις για 50000 και 100000 μηνύματα, τα οποία παράγονταν και μεταφέρονταν στη Redis μέσα σε 38 και 55 δευτερόλεπτα αντίστοιχα. Αυτό που μετρήθηκε σε κάθε περίπτωση, ήταν ο χρόνος που χρειάστηκε ώστε όλα τα μηνύματα να είναι διαθέσιμα στο Elasticsearch, από τη στιγμή που ξεκίνησε η παραγωγή μηνυμάτων. Τα αποτελέσματα φαίνονται στον πίνακα και στις γραφικές παραστάσεις που ακολουθούν.

Πίνακας 4: Μετρήσεις χρόνου για τη διαδηκασία του Indexing

Αριθμός μηνυμάτων	**Χρόνος παραγωγής και μεταφοράς μηνυμάτων στη Redis**	**Ένας κόμβος Elasticsearch και ένας indexer**	**Ένας κομβος Elasticsearch και δύο Indexers**	**Δύο κόμβοι Elasticsearch και ένας Indexer**	**Δύο κόμβοι Elasticsearch και δύο Indexers**
50000	**38 sec**	124 sec (2 min 4 sec)	71 sec (1 min 11 sec)	126 sec (2 min 6 sec)	75 sec (1 min 15 sec)
100000	**55 sec**	257 sec (4 min 17 sec)	127 sec (2 min 7 sec)	257 sec (4 min 17 sec)	137 sec (2 min 17 sec)



Εικόνα 23: Γραφική παράσταση του χρόνου σε σχέση με τους κόμβους στην περίπτωση των 50000 μηνυμάτων



Εικόνα 24: Γραφική παράσταση του χρόνου σε σχέση με τους κόμβους στην περίπτωση των 100000 μηνυμάτων

Γίνεται άμεσα αντιληπτό, ότι η προσθήκη του δεύτερου indexer βελτιώνει σημαντικά τις επιδόσεις του συστήματος, καθώς προκαλεί μείωση στο χρόνο από 40% μέχρι και 50%. Παρατηρείται επίσης ότι η διαφορά στους χρόνους μεταξύ των περιπτώσεων ενός κόμβου και δύο κόμβων Elasticsearch, είναι μηδαμινή. Καθώς, η διατήρηση αντιγράφων των shards (replicas) παρέχει ασφάλεια των δεδομένων και ταυτόχρονα αύξηση στη ταχύτητα

των αναζητήσεων, σαφώς θεωρείται καλύτερη επιλογή η χρήση δύο ή περισσότερων κόμβων για Elasticsearch με διατήρηση αντιγράφων των shards.

6.3 Επιδόσεις Elasticsearch

Για τον έλεγχο των επιδόσεων του Elasticsearch χρησιμοποιήθηκε το Apache Jmeter, μια εφαρμογή που σχεδιάστηκε για μέτρηση της απόδοσης και της λειτουργικής συμπεριφοράς συστημάτων. Καθώς το Elasticsearch παρέχει Rest API, χρησιμοποιώντας το Jmeter, μπορέσαμε να πραγματοποιήσουμε μεγάλο αριθμό http requests προς το Elasticsearch. Θέλοντας να δούμε πώς ανταποκρίνεται το σύστημα κάτω από υψηλές απαιτήσεις, χρησιμοποιήσαμε πολλά threads, ώστε να πραγματοποιούνται ταυτόχρονα πολλά requests.

Για τις μετρήσεις αυτές δημιουργήθηκε συγκεκριμένο index, στο οποίο χρησιμοποιώντας το Logstash, έγινε εισαγωγή των τίτλων της Wikipedia, υπό μορφή "log", τα οποία περιείχαν μόνο δύο πεδία, timestamp και message. Το index, περιείχε 11077809 documents, καταλάμβανε 1.23GB, και δημιουργήθηκε με τις προεπιλεγμένες ρυθμίσεις του Elasticsearch, δηλαδή χωρισμένο σε πέντε shards και με ένα replica για κάθε shard.

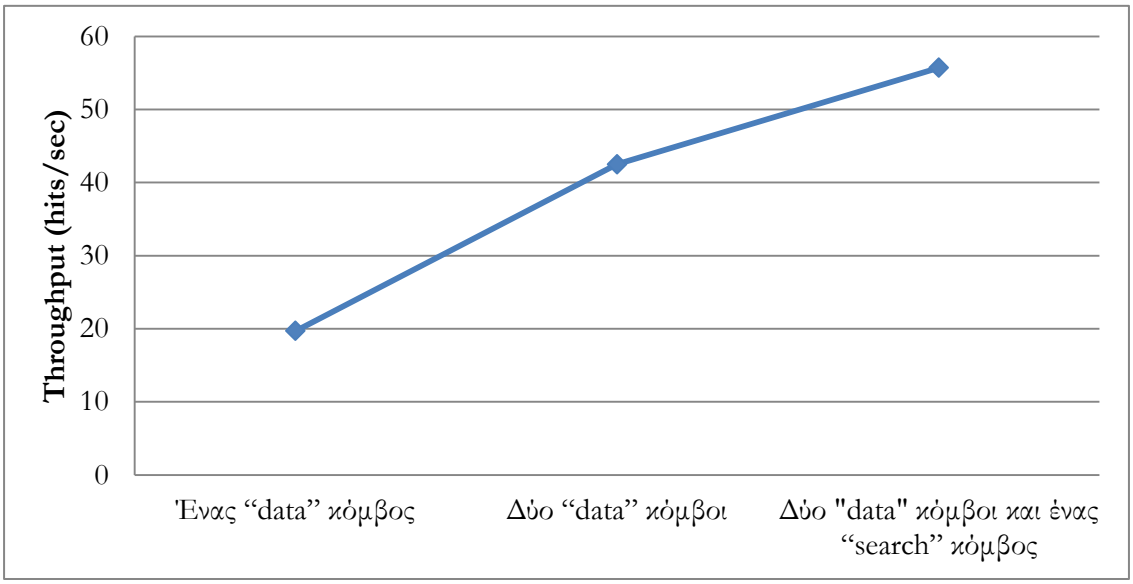
Τα requests που πραγματοποιήθηκαν για το σκοπό των μετρήσεων ήταν απλά "empty search", τα οποία δε διευκρίνιζαν συγκεκριμένο ερώτημα, αλλά απλώς επέστρεφαν όλα τα documents στο συγκεκριμένο index. Γενικά, το πιο σημαντικό τμήμα των απαντήσεων είναι τα "hits", το οποίο περιέχει τον συνολικό αριθμό των documents που ταιριάζουν στο ερώτημα, όπου στην προκειμένη περίπτωση πρέπει να είναι 11077809, οσα δηλαδή τα documents του index. Οι απαντήσεις (responses) στα requests περιέχουν μόνο τα πρώτα δέκα documents από το σύνολο που πληρεί το ερώτημα.

Οι μετρήσεις πραγματοποιήθηκαν σε ένα, δύο και ακολούθως σε τρεις κόμβους όπου έτρεχε το Elasticsearch. Στους Host-2 και Host-3 υπήρχαν τα δεδομένα, ήταν δηλαδή "data" κόμβοι. Αρχικά, απενεργοποιήθηκε ο Host-3, και πραγματοποιήθηκαν τα requests μέσω public ip, στο Host-2. Στη συνέχεια, ενεργοποιήθηκε ο κόμβος Host-3 και έγιναν και πάλι τα ίδια requests στο Host-2. Ακολούθως, απενεργοποιήθηκε η δυνατότητα αποδοχής http request από τους δύο κόμβους, και ενεργοποιήθηκε ο τρίτος κόμβος, ο Host-1, ως "search" κόμβος ο οποίος αποδεχόταν τα τα requests. Ένας "search" κόμβος αναλαμβάνει το διαμοιρασμό του φόρτου εργασίας στους data κόμβους ανάλογα με τα requests που δέχεται, είναι δηλαδή ένας "load balancer".

Πραγματοποιήθηκαν μετρήσεις χρησιμοποιώντας 20 και 40 threads, όπου το κάθε thread εκτελούσε 500 requests. Ακολουθούν τα αποτελέσματα στους πίνακες και τις γραφικές παραστάσεις όπου παρουσιάζεται ο μέσος όρος του throughput και των δεδομένων ανά δευτερόλεπτο.

Πίνακας 5: Μετρήσεις με 20 threads

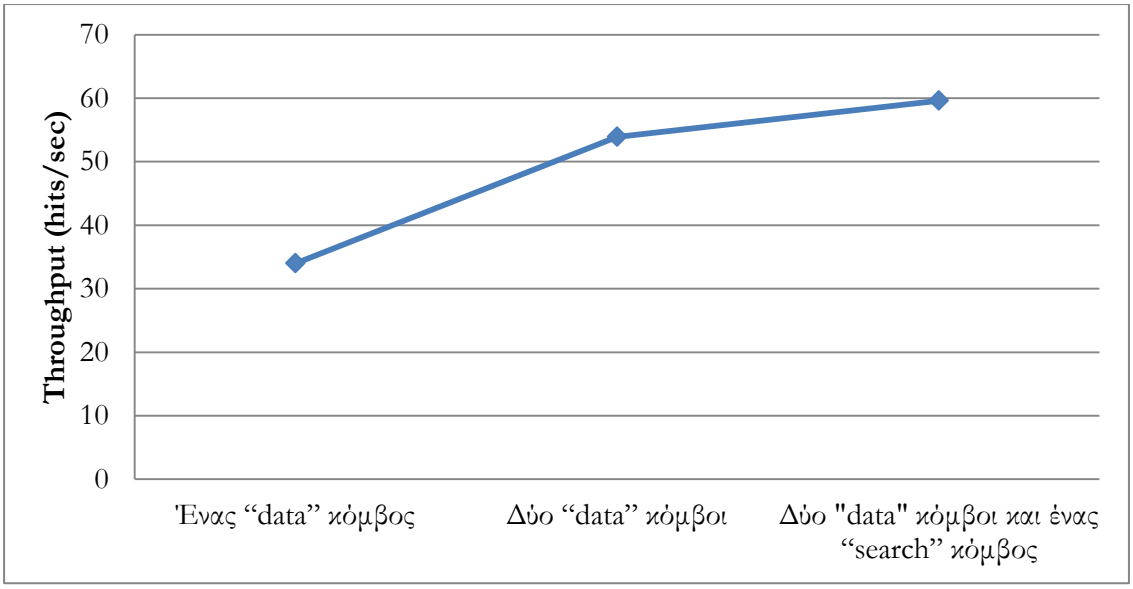
	Throughput	KB/sec
Ένας "data" κόμβος	19.7/sec	32.68
Δύο "data" κόμβοι	42.5/sec	70.68
Δύο "data" κόμβοι και ένας "search" κόμβος	55.7/sec	92.51



Εικόνα 25: : Γραφική παράσταση Throughput σε σχέση με κόμβους στη περίπτωση των 20 threads

Πίνακας 6: Μετρήσεις με 40 threads

	Throughput	KB/sec
Ένας "data" κόμβος	34.0/sec	56.51
Δύο "data" κόμβοι	53.9/sec	89.62
Δύο "data" κόμβοι και ένας "search" κόμβος	59.6/sec	99.12



Εικόνα 26: Γραφική παράσταση Throughput σε σχέση με κόμβους στη περίπτωση των 40 threads

Όπως φαίνεται από τα αποτελέσματα, η προσθήκη δεύτερου κόμβου στο Elasticsearch, λόγω της ύπαρξης των replica shards που επιταχύνουν τις αναζητήσεις, έχει ως αποτέλεσμα τη σημαντική αύξηση της επίδοσης του συστήματος, ενώ η χρήση τρίτου “search” κόμβου έδωσε ακόμη καλύτερα αποτελέσματα. Συγκεκριμένα, στη περίπτωση των 20 threads υπήρξε αύξηση του throughput κατά 115%, ενώ με το “search” κόμβο η αύξηση έφτασε τα 182% σε σχέση με τον ένα κόμβο. Ο διπλασιασμός των threads, είχε ως αποτέλεσμα την αύξηση γενικά του throughput, αλλά η βελτίωση με τη προσθήκη του δεύτερου κόμβου περιορίστηκε στο 58% και με το search κόμβο στο 76%.

7

Επίλογος

7.1 Σύντομη ανακεφαλαίωση

Σκοπός της παρούσας διπλωματικής εργασίας, ήταν η δημιουργία ενός κατανεμημένου συστήματος συλλογής και αποθήκευσης των δεδομένων χρήσης που παράγονται από την λειτουργία ενός υπολογιστικού νέφους. Απώτερος στόχος ήταν η παροχή ενός εργαλείου στους διαχειριστές του υπολογιστικού νέφους, με το οποίο θα μπορούν σχεδόν σε πραγματικό χρόνο να αναλύουν τα δεδομένα χρήσης που παράγονται από τους κόμβους του νέφους.

Για το σκοπό αυτό, στο κεφάλαιο 2, μελετήθηκε η γενική αρχιτεκτονική του centralized logging, μια τεχνική σύμφωνα με την οποία τα δεδομένα συλλέγονται από τους κόμβους, και αποθηκεύονται σε κεντική βάση. Μελετήθηκαν διάφορα κατανεμημένα συστήματα για τη συλλογή και μεταφορά των δεδομένων, όπως το Apache Kafka, Apache Flume και Apache Storm που συνδυάζεται με το Apache Kafka, καθώς και σύστημα Apache Hadoop το οποίο μέσω του HDFS, ένα κατανεμημένο σύστημα αρχείων, προσφέρει ιδανική λύση αποθήκευσης. Καθώς όμως, τα συστήματα που προέκυπταν με βάση τα πιο πάνω εργαλεία, περιορίζονταν στην επεξεργασία και ανάλυση των δεδομένων χρήσης μέσω "batch processing", αναζητήθηκε εναλλακτική λύση.

Επιλέχθηκαν τα Elasticsearch, Logstash και Kibana, ως τα βασικά εργαλεία για την δημιουργία του συστήματος, τα οποία αναλύονται στο κεφάλαιο 3. Το Elasticsearch είναι μια πλατφόρμα αναζήτησης, σχεδιασμένη να τρέχει σε κατανεμημένα περιβάλλοντα. Η δυνατότητα του Elasticsearch να αποθηκεύει τα δεδομένα, και να τα διαθέτει προς αναζήτηση σχεδόν σε πραγματικό χρόνο ήταν ο βασικός λόγος της επιλογής αυτών των εργαλείων. Το Logstash, αποτελεί ένα εξαιρετικό εργαλείο για επεξεργασία δεδομένων χρήση καθώς, μέσω των plugins που έχουν δημιουργηθεί, προσφέρει πολλές επιλλογές και συνδυάζεται με πάρα πολλά εργαλεία. Το Kibana, είναι μια εφαρμογή, που αναλαμβάνει την παρουσίαση των δεδομένων που βρίσκονται αποθηκευμένα στο Elasticsearch ενώ προσφέρει και τη δυνατότητα αναζητήσεων.

Στο κεφάλαιο 4, αναλύθηκαν κατανεμημένες τεχνικές συλλογής και μεταφοράς δεδομένων χρήσης τα οποία συνδυάζονται με το Logstash. Συγκεκριμένα μελετήθηκαν τρεις

"shippers", εργαλεία δηλαδή που αναλαμβάνουν τη συλλογή και αποστολή δεδομένων χρήσης από τους κόμβους. Αρχικά μελετήθηκε η αρχιτεκτονική με χρήση του ίδιου το Logstash ως shipper και ενός "broker" ενώ ακολούθως μελετήθηκαν αρχιτεκτονικές με χρήση του logstash-forwarder και του beaver. Σε όλες τις αρχιτεκτονικές συστημάτων που μελετήθηκαν, υπήρχαν Logstash ως "Indexers", αυτοί που αναλάμβαναν την επεξεργασία των δεδομένων χρήσης και αποστολής τους στο Elasticsearch για αποθήκευση.

Στο κεφάλαιο 5, έγινε παρουσίαση του "Synnefo", μιας πλατφόρμας υπεύθυνη για τη διαχείριση ενός υπολογιστικού νέφους. Ακολούθως έγινε αναλυτική παρουσίαση ενός πρωτοτύπου που δημιουργήθηκε για εφαρμογή του centralized logging στο Synnefo, μέσω των εργαλείων που επιλέχθηκαν. Η αρχιτεκτονική συστήματος που επιλέχθηκε για την εφαρμογή στο Synnefo, είναι αυτή με το Logstash ως shipper και με τη χρήση broker. Καθώς, η χρήση broker, εξασφαλίζει την ισοκατανομή του φόρτου εργασίας στους Logstash indexers, αποτελεί ιδανική λύση στην περίπτωση του Synnefo όπου υπάρχουν πάρα πολλοί κόμβοι στους οποίους η παραγωγή των δεδομένων χρήσης γίνεται με διαφορετικό ρυθμό.

Ακολούθως, στο κεφάλαιο 6 πραγματοποιήθηκαν μετρήσεις ώστε να παρουσιάσουμε τις επιδόσεις του συστήματος που δημιουργήθηκε. Αυτό που παρουσίαζε ιδιαίτερο ενδιαφέρον ήταν εξέταση του χρόνου που απαιτείται ώστε τα δεδομένα χρήσης που παράγονται στους κόμβους, να μεταφερθούν και να είναι διαθέσιμα στο Elasticsearch καθώς επίσης και οι επιδόσεις του Elasticsearch στις αναζητήσεις. Σύμφωνα με τις μετρήσεις που έγιναν, η επιδόσεις του συστήματος αυξάνονται σημαντικά με την χρήση επιπλέον indexer. Επίσης, όπως φάνηκε, η διατήρηση αντιγράφων (replicas) των shards δεν επηρεάζει αισθητά το σύστημα κατά τη διαδικασία του indexing , και δεδομένου ότι μέσω των αντιγράφων εξασφαλίζεται ασφάλεια των δεδομένων και ταυτόχρονα καλύτερη απόδοση στις αναζητήσεις, σίγουρα είναι σωστή επιλογή. Την επίδοση του Elasticsearch στις αναζητήσεις εξετάσαμε χρησιμοποιώντας "empty queries". Πράγματι, χρησιμοποιώντας δεύτερο κόμβο ο οποίος διατηρούσε αντίγραφα των shards του πρώτου κόμβου, το σύστημα απαντούσε σχεδόν τα διπλάσια ερωτήματα ανά δευτερόλεπτο. Ακολούθως, με την προσθήκη τρίτου κόμβου, ο οποίος λειτουργούσε ως "search" κόμβος, αναλάμβανε δηλαδή την διαμοίραση του φόρτου εργασίας χωρίς ο ίδιος να κάνει αναζητήσεις, πήραμε ακόμα καλύτερη απόδοση.

7.2 Επεκτάσεις

Η περεταίρω διερεύνηση του αντικειμένου της εργασίας θα μπορούσε να περιλαμβάνει έρευνα κατά πόσο θα μπορούσε το συγκεκριμένο σύστημα να εξασφαλίζει και το σύστημα προειδοποιήσεων, ώστε να μην χρειάζεται η χρήση δεύτερου monitoring συστήματος στη υποδομή. Για παράδειγμα στον ~Okeanos, χρησιμοποιείται το "Nagios", το οποίο δημιουργεί "warnings" και αναλαμβάνει να προειδοποιεί τους διαχειριστές του συστήματος για τυχόν βλάβες μέσω ηλεκτρονικού ταχυδρομείου σύμφωνα με συγκεκριμένες παραμέτρους. Λόγω της δυνατότητας που προστέθηκε στο Logstash, της δημιουργίας

συνθηκών με το προγραμματιστικό μοντέλο if-then-else και με τη χρήση συγκεκριμένων plugin, θα μπορούσε να υπάρξει ένα καλό αποτέλεσμα.

8

Βιβλιογραφία

[1] July 2014. [Ηλεκτρονικό]. Available: https://www.synnefo.org/.

[2] July 2014. [Ηλεκτρονικό]. Available: http://flume.apache.org/.

[3] July 2014. [Ηλεκτρονικό]. Available: http://logstash.net/.

[4] July 2014. [Ηλεκτρονικό]. Available: http://kafka.apache.org/.

[5] July 2014. [Ηλεκτρονικό]. Available: http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[6] July 2014. [Ηλεκτρονικό]. Available: http://www.elasticsearch.org/overview/elasticsearch/.

[7] July 2014. [Ηλεκτρονικό]. Available: http://hadoop.apache.org/.

[8] July 2014. [Ηλεκτρονικό]. Available: https://hive.apache.org/.

[9] July 2014. [Ηλεκτρονικό]. Available: http://pig.apache.org/.

[10] S. G. Jeffrey Dean, «MapReduce: Simplifed Data Processing on Large Clusters».

[11] July 2014. [Ηλεκτρονικό]. Available: http://www.elasticsearch.org/overview/kibana/.

[12] July 2014. [Ηλεκτρονικό]. Available: http://graylog2.org/.

[13] July 2014. [Ηλεκτρονικό]. Available: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

[14] July 2014. [Ηλεκτρονικό]. Available: https://storm.incubator.apache.org/.

[15] July 2014. [Ηλεκτρονικό]. Available: https://storm.incubator.apache.org/documentation/Guaranteeing-message-processing.html.

[16] July 2014. [Ηλεκτρονικό]. Available: http://lucene.apache.org/core/.

[17] July 2014. [Ηλεκτρονικό]. Available: http://json.org/.

[18] July 2014. [Ηλεκτρονικό]. Available: http://www.elasticsearch.org/.

[19] July 2014. [Ηλεκτρονικό]. Available: http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/docs.html.

[20] July 2014. [Ηλεκτρονικό]. Available: http://logstash.net/docs/1.4.2/.

[21] July 2014. [Ηλεκτρονικό]. Available: https://github.com/logstash/logstash/tree/v1.4.2/patterns.

[22] July 2014. [Ηλεκτρονικό]. Available: http://grokdebug.herokuapp.com.

[23] July 2014. [Ηλεκτρονικό]. Available: http://httpd.apache.org/docs/2.0/.

[24] July 2014. [Ηλεκτρονικό]. Available: http://www.rabbitmq.com/.

[25] July 2014. [Ηλεκτρονικό]. Available: http://redis.io/.

[26] July 2014. [Ηλεκτρονικό]. Available: https://github.com/elasticsearch/logstash-forwarder.

[27] July 2014. [Ηλεκτρονικό]. Available: http://beaver.readthedocs.org/en/latest/index.html.

[28] July 2014. [Ηλεκτρονικό]. Available: https://collectd.org/.

[29] July 2014. [Ηλεκτρονικό]. Available: http://ganglia.sourceforge.net/.