ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

NETMODE – NETWORK MANAGEMENT & OPTIMAL DESIGN LABORATORY

Κλιμακώσιμο Σύστημα για κατ' απαίτηση Συλλογή και Επεξεργασία Δεδομένων Δικτυακής Κίνησης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Νεκτάριος Δ. Σωτηρόπουλος

Επιβλέπων : Βασίλειος Μάγκλαρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2017

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

NETMODE – NETWORK MANAGEMENT & OPTIMAL DESIGN LABORATORY

Κλιμακώσιμο σύστημα ελέγχου δειγματοληπτημένης δικτυακής κίνησης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Νεκτάριος Δ. Σωτηρόπουλος

Επιβλέπων : Βασίλειος Μάγκλαρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 28η Σεπτεμβρίου 2017.

.....	.....	.....
Βασίλειος Μάγκλαρης	Νεκτάριος Κοζύρης	Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2017

.....

Ιωάννης Νεκτάριος Σωτηρόπουλος

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

Ευχαριστίες

Η παρούσα διπλωματική εργασία αποτελεί το τελευταίο στάδιο των προπτυχιακών σπουδών μου στο Εθνικό Μετσόβιο Πολυτεχνείο.

Αρχικά θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Βασίλειο Μάγκλαρη για την εμπιστοσύνη που μου έδειξε, την ευκαιρία που μου έδωσε καθώς και για τις πολύτιμες συμβουλές του. Ακόμη, ευχαριστώ όλα τα μέλη του εργαστηρίου NETMODE για την βοήθεια τους, και ιδιαιτέρως τον υπεύθυνο της διπλωματικής μου εργασίας Κωνσταντίνο Γιώτη και Αδάμ Παυλίδη, για την άριστη συνεργασία μας καθώς και για την πολύτιμη καθοδήγησή τους σε όλα τα στάδια εκπόνησης της.

Τέλος θα ήθελα να ευχαριστήσω την οικογένεια και τους φίλους μου για την ανεκτίμητη υποστήριξη και συμπαράσταση τους, στη διάρκεια των σπουδών μου.

Περίληψη

Σκοπός της διπλωματικής εργασίας είναι η ανάπτυξη πολύ-επίπεδης πλατφόρμας που παρέχεται κατ' απαίτηση σε πολλαπλούς ενοίκους-χρήστες για την καταγραφή και αποθήκευση προσωποποιημένης δικτυακής πληροφορίας (sflow). Δίνεται ιδιαίτερη έμφαση στην ικανότητα προσαρμογής και κλιμάκωσής της, εκμεταλλευόμενοι των χαρακτηριστικών και ιδιαιτεροτήτων του κάθε σταδίου της πλατφόρμας. Επομένως, το τελικό επιθυμητό αποτέλεσμα είναι η αυτόματη παροχή, ύστερα από αίτημα, προσωποποιημένου συστήματος για την καταγραφή και παρακολούθηση δεδομένων που αντιστοιχούν στον εκάστοτε χρήστη.

Ειδικότερα, πρόκειται για ένα κατανεμημένο σύστημα συλλογής δεδομένων, με δεδομένα, στη συγκεκριμένη περίπτωση, την κίνηση που λαμβάνεται από τους μεταγωγείς μιας δικτυακής υποδομής. Για τον σκοπό αυτό χρησιμοποιήθηκε το Apache Kafka για τη συλλογή των δεδομένων από τους μεταγωγείς (centralized logging), το Logstash και το Smonnet, το όνομα του python προγράμματος που αναπτύχθηκε για την επεξεργασία και τον εμπλουτισμό τους, το Elasticsearch σαν σύστημα ανάλυσης με κατανεμημένη τεχνική αποθήκευσης σε συστοιχίες υπολογιστών για την αποθήκευση και καταγραφή τους και τέλος το Kibana για την οπτικοποίηση συσχετισμών και μετρήσεων πάνω στον χρόνο.

Με αυτόν τον τρόπο, η πλατφόρμα παρέχει κατ' απαίτηση μηχανισμό που περιλαμβάνει τα απαραίτητα εργαλεία για να αποτελέσει βάση για τη δημιουργία ενός ενοποιημένου πλαισίου για την άντληση δεδομένων και αυτόματης παρουσίασης των δεδομένων αυτών στους χρήστες των διαφόρων (πειραματικών) υποδομών μεγάλης κλίμακας σαν μέρος της λειτουργικότητας που προσφέρει το περιβάλλον.

Λέξεις Κλειδιά: <<σύστημα παρακολούθησης (monitoring) και καταγραφής δικτυακών δεδομένων, Monitoring as a Service, Elasticsearch, Kibana, ELK stack, Kubernetes, Docker, logging, NFV, ανάλυση σε πραγματικό χρόνο>>

Abstract

The scope of this thesis is the development of a multilayer platform which is provided on demand to many users in order to store and monitor network traffic (sflow). Following the features of each layer, there was a focus on the adaptability and scalability of the whole platform. Therefore, the final service would be a personalized monitoring system provided on demand with data which corresponds to each user.

Specifically, the main part of this platform is a distributed data collection system, where in this case the data is received from the switches of the network infrastructure. For this purpose, Apache Kafka was used for central logging, Logstash and Smonnet, a custom python application developed in-house for processing and data enrichment, Elasticsearch as an analytics engine which has a distributed storage system for monitoring and Kibana for the visualization of the correlations and metrics against time.

This platform provides on demand the necessary tools in order to be used as the foundation for the creation of a unified framework for collecting –network– monitoring data, and automatically presenting them to the corresponding tenants of various (experimental) infrastructures as part of the functionality of the system.

Keywords: <<network traffic monitoring system, Monitoring as a Service, Elasticsearch, Kibana, ELK stack, Kubernetes, Docker, logging, NFV, data enrichment in real time >>

Πίνακας περιεχομένων

Κατάλογος Σχημάτων

1

Εισαγωγή

1.1 Αντικείμενο

Η ανάπτυξη του Internet και των εικονικών υποδομών (virtual infrastructures) προσέφερε τη δυνατότητα διαμοιρασμού των πόρων του συστήματος και του δικτύου στους χρήστες-ενοικιαστές (tenants) σε υπολογιστικά νέφη. Εταιρείες χρησιμοποιούν τέτοιες υποδομές για να εξυπηρετούν εκατομμύρια χρήστες και να παρέχουν αποδοτικά τις υπηρεσίες τους.

Επίσης, τηλεπικοινωνιακοί πάροχοι και οργανισμοί προσφέρουν στους χρήστες του δικτύου τους υπηρεσίες που σιγά σιγά μεταφέρονται από hardware σε software προσφέροντας εικονοποιημένες δικτυακές υπηρεσίες (virtual network functions, VNFs). Για τις υπηρεσίες αυτές χρησιμοποιούνται και σε αυτήν την περίπτωση εικονικές υποδομές ώστε να παρέχονται δυναμικά και κατά απαίτηση με στόχο την καλύτερη χρησιμοποίηση των πόρων.

Οι δύο παραπάνω τύποι υπηρεσιών αφορούν διαφορετικούς τύπους χρηστών, ενός παρόχου δικτυακών υπηρεσιών μέσω νέφους και των διαχειριστών ενός δικτύου πολλαπλών ενοίκων. Αυτό που έχουν σαν κοινό είναι ότι και οι δύο έχουν υψηλές απαιτήσεις και θέλουν να εξασφαλίζουν την απρόσκοπτη παροχή πόρων, καθώς και την ποιότητα των προσφερόμενων υπηρεσιών (QoS - Quality of Service). Σημαντικό ρόλο για την επίτευξη αυτού του στόχου έχει η παρακολούθηση (monitoring) πόρων που χρησιμοποιούνται. Η χρήση της υποδομής δημιουργεί ένα μεγάλο όγκο δεδομένων χρήσης (logs) για τα συστήματα και το δίκτυο τα οποία μπορούν να αξιοποιηθούν. Χρησιμοποιώντας κατάλληλο monitoring σύστημα, είναι δυνατόν να εντοπίζονται γρήγορα βλάβες, τυχόν δικτυακά προβλήματα και επιθέσεις.

Κινούμενοι σε αυτήν την κατεύθυνση και λόγω του ότι έχουν γίνει διάφορες προσπάθειες προς την καταγραφή logs συστημάτων, η παρούσα διπλωματική έχει να κάνει με τη

δημιουργία μιας πλατφόρμας καταγραφής και παρακολούθησης δειγματοληπτημένης δικτυακής κίνησης που μπορεί να παρέχεται κατά απαίτηση στους χρήστες ενός δικτύου. Σκοπός είναι τηλεπικοινωνιακοί πάροχοι, εταιρείες, οργανισμοί και γενικά σε εικονοποιημένα δίκτυα πολλών χρηστών να μπορούν να παρέχονται τα εργαλεία σε χρήστες ή οντότητες ενός δικτύου ώστε να έχουν μια εποπτεία της δικτυακής κίνησης. Βεβαίως αυτή πρέπει να αφορά μόνο τα δικά τους συστήματα και άρα να μπορούν να έχουν τη δυνατότητα να εντοπίζουν τυχόν δικτυακές επιθέσεις.

Η πλατφόρμα στήθηκε με σκοπό να δέχεται δεδομένα με μεγάλη ροή και επιτρέπει την αυτόματη διαχείριση των πόρων ώστε να χρησιμοποιούνται κάθε στιγμή μόνο οι πόροι που χρειάζονται. Αυτό το χαρακτηριστικό την κάνει πιο δυναμική και επιτρέπει την πιο γενική χρήση της, για παρακολούθηση πολλών δεδομένων και όχι μόνο δικτυακών.

Συγκεκριμένα, για τις ανάγκες της διπλωματικής μελετώνται μηχανισμοί και εργαλεία για την ανάπτυξη μεθόδων συλλογής δικτυακών δεδομένων (network monitoring data) και παρέχοντάς τους κατά απαίτηση σε ενδιαφερόμενους χρήστες που βρίσκονται στο δίκτυο. Λαμβάνοντας υπ όψιν την ποικιλομορφία των δικτυακών υποδομών που συναντώνται στις μέρες μας, βαρύνουσα σημασία έχει η ικανότητα προσαρμογής του μηχανισμού σε αυτές. Μιλώντας για υποδομή πολλαπλών ενοίκων, ο καθένας πρέπει να έχει πρόσβαση στα δεδομένα που συλλέγονται από πόρους που του έχουν εκμισθωθεί. Επιπροσθέτως, είναι επιθυμητό τα δεδομένα που αντλούνται να παρουσιάζονται στον εκάστοτε ενδιαφερόμενο απομονωμένα από αυτά που δεν τον αφορούν και δεν πρέπει να έχει πρόσβαση. Τέτοια δεδομένα είναι εξαιρετικά χρήσιμα για τους χρήστες των πειραματικών υποδομών, οργανισμών με πολλούς διαφορετικούς χρήστες ή και τηλεπικοινωνιακών παρόχων παρέχοντάς τους δυνατότητες όπως: (1) επόπτευση δικτυακής κίνησης και εύκολη εξαγωγή αναφορών, και (2) ανάπτυξη μηχανισμών ανίχνευσης και αντιμετώπισης περιστατικών δικτυακών επιθέσεων (Anomaly Detection and Attack Mitigation).

1.2 Οργάνωση

Η παρούσα διπλωματική εργασία μπορεί να διαχωριστεί σε δύο τμήματα. Το πρώτο τμήμα συνίσταται στην ανάλυση εργαλείων χρήσιμων ως υπόβαθρο για την κατανόηση της διπλωματικής εργασίας μαζί με την ανάλυση της αρχιτεκτονικής που χρειάζεται μια τέτοια πλατφόρμα. Το δεύτερο τμήμα αποτελεί και το πρακτικό κομμάτι της εργασίας, όπου περιλαμβάνονται η υλοποίηση, οι βασικές αρχές πάνω στις οποίες στηρίχθηκε, η αξιολόγηση της καθώς και ο επίλογος.

Πιο συγκεκριμένα, το πρώτο τμήμα απαρτίζεται εξ ολοκλήρου από το **Κεφάλαιο 2 και 3**. Σε αυτό το κεφάλαιο αναλύονται οι απαιτήσεις και τα χαρακτηριστικά μιας πλατφόρμας καταγραφής και κατ' απαίτηση και 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 και Kibana για τη δημιουργία του συστήματος καθώς και όσον αφορά την παρακολούθηση – συλλογή δικτυακών δεδομένων, το πρότυπο sFlow.

Το δεύτερο τμήμα περιέχει τα υπόλοιπα τέσσερα κεφάλαια:

- ✓ **Κεφάλαιο 4:** Αρχικά, παρουσιάζεται η αρχιτεκτονική της πλατφόρμας και των σταδίων που την απαρτίζουν και ύστερα αναλύεται η υλοποίηση και οι αναφέρονται οι λεπτομέρειες του κάθε σταδίου της πλατφόρμας ξεχωριστά.
- ✓ **Κεφάλαιο 5:** Αναλύεται η πειραματική διαδικασία. Παρουσιάζεται η διάταξη και τα αποτελέσματα για διάφορα σενάρια καθώς και η απόδοση και η κλιμακωσιμότητα της πλατφόρμας. Επίσης, γίνεται εμφανής ο λόγος επιλογής για την επεξεργασία και τον εμπλουτισμό των ροών δικτυακής κίνησης το Smonnet, που είναι το όνομα που δόθηκε στο python πρόγραμμα που υλοποιήθηκε για τις ανάγκες της διπλωματικής.
- ✓ **Κεφάλαιο 6:** Συνοψίζονται τα συμπεράσματα που εξήχθησαν στα πλαίσια της παρούσας διπλωματικής εργασίας και προτείνονται επιπλέον βελτιώσεις και επεκτάσεις.

2

Θεωρητικό Υπόβαθρο

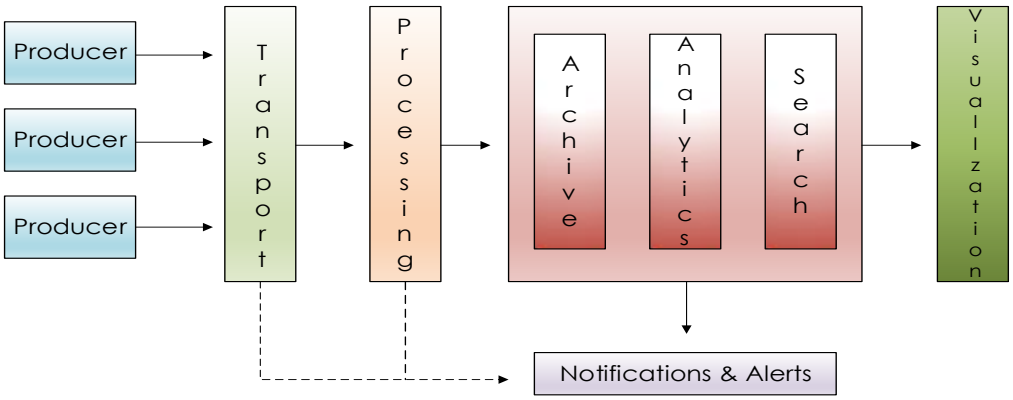
Στο συγκεκριμένο κεφάλαιο αναλύεται το θεωρητικό υπόβαθρο (εργαλεία, μηχανισμοί και εφαρμογές), πάνω στο οποίο στηρίζεται η παρούσα διπλωματική εργασία.

2.1 Στάδια συστήματος καταγραφής και παρακολούθησης δεδομένων

Ένα σύστημα καταγραφής και παρακολούθησης δεδομένων περιλαμβάνει τουλάχιστον τα εξής στάδια:

- ✓ Συλλογή
- ✓ Μεταφορά
- ✓ Επεξεργασία
- ✓ Αποθήκευση
- ✓ Αναζήτηση

Ένα ολοκληρωμένο σύστημα περιλαμβάνει επίσης και την ανάλυση των δεδομένων μέσα από την αναζήτηση, όπως π.χ. ένα όνομα που εμφανίζεται τις περισσότερες φορές. Ακόμη, μπορεί να περιλαμβάνει και την οπτικοποίηση των δεδομένων. Ένα τέτοιο σχήμα παρουσιάζεται στο σχήμα παρακάτω.



Σχήμα 2.1 Ολοκληρωμένο σύστημα για centralized logging

Παρά το γεγονός ότι το σύστημα που αφορά η διπλωματική δεν έχει μόνο κάποιο κεντρικό σύστημα αποθήκευσης αλλά έχει και τα επιμέρους συστήματα καταγραφής των διαφόρων χρηστών του δικτύου, η αρχιτεκτονική που ακολουθείται είναι αυτή για centralized logging.

2.1.1 Συλλογή δεδομένων – Collection

Το δίκτυο δημιουργεί δεδομένα χρήσης με διάφορους τρόπους, για παράδειγμα μέσω του sflow[10] ή netflow[11]. Στην περίπτωση όπου το centralized logging πραγματοποιείται για να δίνει τη δυνατότητα στους 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 ή κοντά σε αυτόν για την άμεση εύρεση και αντιμετώπιση τυχόν προβλημάτων ή επιθέσεων. Η διατήρηση αντιγράφων των δεδομένων (file replication) σε κεντρική βάση ανά διαστήματα δεν μπορεί να αποτελέσει λύση σε τέτοιες περιπτώσεις. Αντίθετα, θα ήταν καλή προσέγγιση αν υπήρχε ανάγκη ανάλυσης των δεδομένων για στατιστικές έρευνες.

2.1.2 Μεταφορά και Επεξεργασία – Transport and Processing

Η μεταφορά των δεδομένων χρήσης αξιόπιστα και γρήγορα σε κεντρική βάση δημιούργησε την ανάγκη δημιουργίας μιας νέας κλάσης κατανεμημένων συστημάτων για αποτελεσματική μετάδοση των δεδομένων και προσωρινή αποθήκευση. Εργαλεία όπως το Apache Flume [12], Scribe [13], Logstash [25], fluentd [14], Kafka και Redis [15] 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 παρέχει 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, fluentd και Flume παρέχουν μια σειρά από πηγές εισόδου για συλλογή των δεδομένων όπως για παράδειγμα το πρωτόκολλο Syslog ή Netflow ή Sflow από συγκεκριμένη πόρτα, αλλά μπορούν να διαβάσουν δεδομένα χρήσης από αρχεία. Επίσης, παρέχουν τρόπους επεξεργασίας των ροών που λαμβάνουν και έχουν μια μεγάλη ποικιλία από πηγές εξόδου καθώς επίσης δίνουν τη δυνατότητα υλοποίησης νέας πηγής που δεν υπάρχει στις βιβλιοθήκες τους.

Όπως είναι εμφανές από τα παραπάνω, πολλά από τα διαθέσιμα εργαλεία δεν έχουν να κάνουν μόνο με τη μεταφορά των δεδομένων επικοινωνώντας από τις πηγές εισόδου στις πηγές εξόδου, αλλά έχουν τη δυνατότητα επεξεργασίας ή εμπλουτισμού των ροών που λαμβάνουν.

2.1.3 Αποθήκευση και Ανάλυση – Store and Analysis

Προκειμένου να αναλυθούν περεταίρω τα δεδομένα και να μπορεί να γίνει αναζήτηση μέσα σε αυτά είναι αναγκαία η αποθήκευσή τους στην οποία βασίζεται η όποια ανάλυση ακολουθήσει.

Στο κομμάτι της αποθήκευσης υπάρχουν πολλές επιλογές ανοιχτού κώδικα όπως το Hadoop [16] και HBase [17], Cassandra [18], MongoDB [19], Elasticsearch [25] το οποίο παρέχει και κάποιους μηχανισμούς για συσχετίσεις μεταξύ των δεδομένων συναθροίζοντάς τα. Το σύστημα που θα αποθηκευθούν τα δεδομένα θα αποτελέσει ουσιαστικά την πηγή εισόδου για το σύστημα ανάλυσης.

Σχετικά με την ανάλυση, υπάρχουν διαφορετικοί τρόποι που μπορεί κανείς να αναλύσει τα δεδομένα, είτε σε πραγματικό χρόνο είτε εκτός σύνδεσης (offline) για μεγαλύτερη σε διάρκεια τύπου ανάλυση. Η τελευταία περίπτωση συνδέεται συνήθως και με αλγορίθμους εκμάθησης (machine learning) που στόχο έχουν να μάθουν συγκεκριμένα patterns και να προβλέψουν συγκεκριμένες ενέργειες ή καταστάσεις. Για την ανάλυση, λοιπόν, υπάρχουν διάφορα εργαλεία που βοηθούν σε αυτήν την κατεύθυνση όπως το Apache Spark [20], Apache Storm [21] που μπορούν να κάνουν και batch processing καθώς και κάποια από τα εργαλεία που αναφέρθηκαν ήδη στο κομμάτι της μεταφοράς των δεδομένων όπως το Logstash, Flume, fluentd.

Στην περίπτωση που χρειάζεται ανάλυση των δεδομένων μέσω διεπαφής χρήστη (user interface, UI), ιδανική προσέγγιση είναι η αποθήκευση των δεδομένων στο Elasticsearch και χρήση του Kibana[27] ή του Graylog2 [22] για ανάλυση και το Logstash [26] ή κάποια άλλη

εφαρμογή για parsing. Αυτή η προσέγγιση επιτρέπει διαδραστική πρόσβαση στα δεδομένα σε πραγματικό χρόνο.

2.2 Elasticsearch

Το Elasticsearch [24] είναι ένα μια μηχανή αναζήτησης και ανάλυσης με κύριο σκοπό την αναζήτηση κειμένου. Παρέχει μία κατανεμημένη μηχανή αναζήτησης «πλήρους κειμένου» (fulltext search) με δυνατότητα εξυπηρέτησης πολλαπλών χρηστών, μία HTTP διεπαφή και JSON [27] έγγραφα χωρίς schema. Έχει αναπτυχθεί σε Java και είναι λογισμικό ανοιχτού κώδικα κάτω από τους όρους της άδειας χρήσης Apache License. Η Elasticsearch είναι χτισμένη πάνω από τον Apache Lucene [23], και είναι η δημοφιλέστερη μηχανή αναζήτησης στην βιομηχανία δεδομένων ακολουθούμενη από το Apache Solr, το οποίο είναι επίσης βασισμένο στο Lucene. Το Elasticsearch μπορεί να χρησιμοποιηθεί για αναζήτηση σε κάθε είδους έγγραφα. Προσφέρει κλιμακώσιμη αναζήτηση, έχει αναζήτηση σχεδόν πραγματικού χρόνου (near real-time search) και υποστηρίζει ύπαρξη πολλών χρηστών. Είναι κατανεμημένη, κάτι που σημαίνει ότι τα indexes της (αντίστοιχα με βάση δεδομένων σε παραδοσιακά DBMS) μπορούν να χωριστούν σε shards και κάθε shard να έχει από καθόλου μέχρι πολλά αντίγραφα. Κάθε κόμβος φιλοξενεί ένα ή περισσότερα shards, και λειτουργεί ως συντονιστής ώστε να διαμοιράζει τις ενέργειες στα κατάλληλα shards. Υποστηρίζεται αυτόματη αναπροσαρμογή φόρτου εργασίας μεταξύ των κόμβων. Το σύστημα αυτό χρησιμοποιεί το εργαλείο Lucene, μία πολύ ισχυρή βιβλιοθήκη ανοιχτού κώδικα για αναζήτηση πλήρους κειμένου (full-text search), και προσπαθεί να διαθέσει όλα τα χαρακτηριστικά του μέσω του JSON και Java API που παρέχει.

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.

Αξιοσημείωτοι χρήστες της Elasticsearch σήμερα είναι οι Adobe, Facebook, Mozilla, Quora, Foursquare, GitHub, Stack Exchange, Netflix και άλλοι.

Τέλος, το Elasticsearch είναι το βασικό εργαλείο του ELK stack (Elasticsearch, Logstash, Kibana) τα οποία περιγράφονται στα επόμενα κεφάλαια.

Σχήμα 2.2 ELK stack

2.2.1 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 (χρησιμοποιείται σαν αντίστοιχο του insert στη συγκεκριμένη περίπτωση), 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.

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. LBL Δεν υπάρχει περιορισμός στον αριθμό documents) που μπορούν να αποθηκευθούν σε ένα index, λόγω του ότι είναι αρκετά μεγάλος

(2,147,483,519 = Integer.MAX_VALUE – 128), πρέπει όμως το κάθε document να συσχετίζεται με κάποιο type του index.

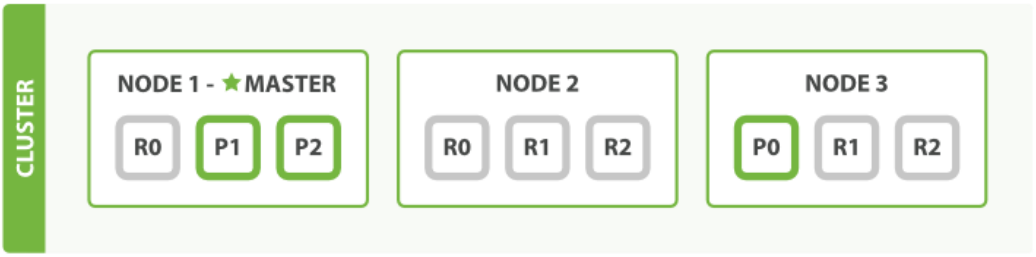
2.2.2 Shards και Replicas

Καθώς δεν υπάρχει κάποιος μεγάλος 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 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 “replicas”. 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. Παράδειγμα ενός cluster με nodes και primary και replica shards φαίνεται παρακάτω.



Σχήμα 2.3 Επισκόπηση των Elasticsearch cluster nodes και shards

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.

2.3 Logstash

Το Logstash [25] 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 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.

2.3.1 Κύριες έννοιες

inputs: μηχανισμοί για πέρασμα μηνυμάτων από logs στο logstash. Για παράδειγμα, το plugin «File» που διαβάζει αρχεία log, το plugin «Kafka» που διαβάζει από Kafka 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 «translate» το οποίο αντικαθιστά με βάση ένα yaml ή json αρχείο με key-value ζευγάρια κάποιο πεδίο του log 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» και το «File» για αποθήκευση σε αρχείο.

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

2.3.2 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 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. 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. Υποστηρίζονται μόνο μετατροπές σε int και float.

2.3.3 Τρόπος λειτουργίας

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 εξόδους 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.

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 όπως το beaver ή τα beats, που είναι οι ελαφριοί shippers του ELK stack. 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:

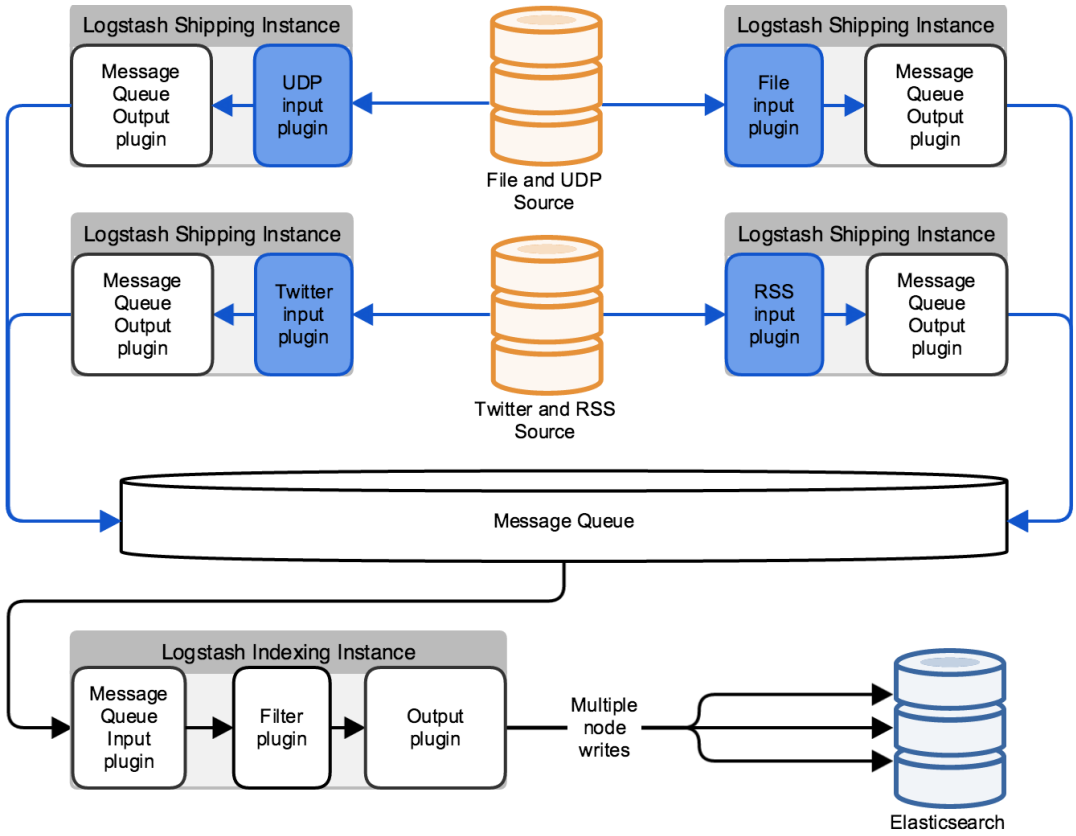
```
input {
stdin { }
}
filter {
if [message] == "error"
drop { }
}
output {
elasticsearch { }
stdout { codec => rubydebug }
}
```

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.

Μια προτεινόμενη αρχιτεκτονική του logstash για να μπορεί να κλιμακώνεται κιόλας, απαιτεί τη χρήση κάποιας ουράς, broker ώστε να μπορεί να δέχεται το input ασύγχρονα από περισσότερες πηγές και να μη δημιουργείται συμφόρηση από την είσοδο πολλών δεδομένων στο στάδιο input. Παρακάτω φαίνεται ένα σχήμα που περιγράφει όλα τα παραπάνω.



Σχήμα 2.4 Κλιμάκωση Logstash μέσω ουράς

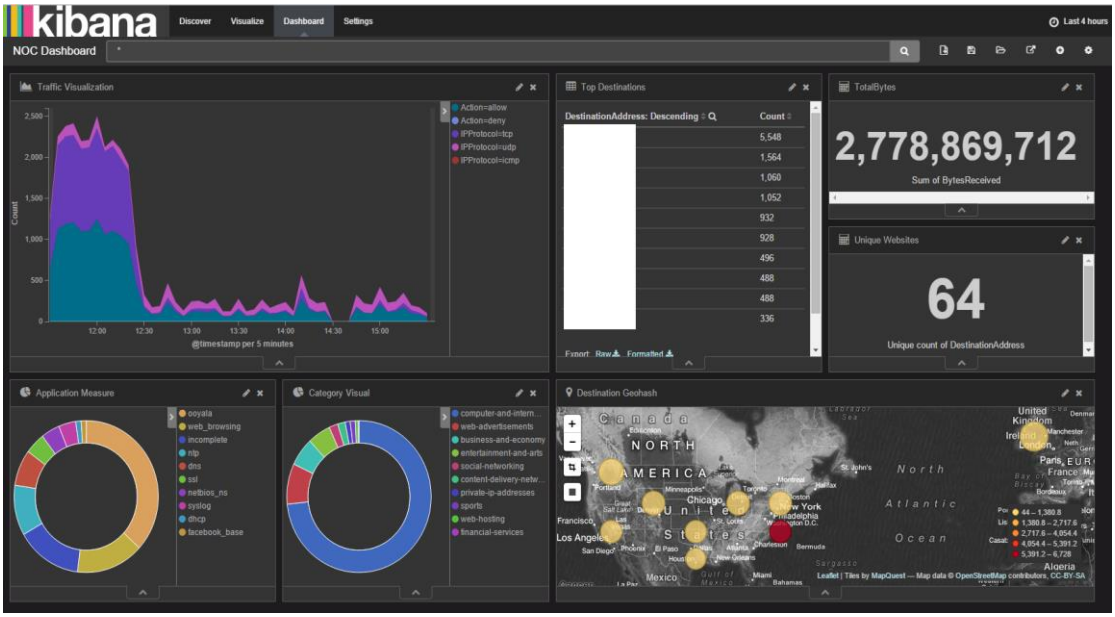
2.4 Kibana

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 5 [26] είναι η νεότερη έκδοση του Kibana και είναι γραμμένο σε HTML5 και Javascript εξ ολοκλήρου ενώ το παλιότερα στηριζόταν στη Ruby. Από την

έκδοση του Kibana 3 δίνεται η 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.

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.

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.



Σχήμα 2.5 Παράδειγμα Kibana dashboard

2.5 Containers

2.5.1 Ιστορία

Η έννοια του container εμφανίστηκε για πρώτη φόρα το 1979, με την υλοποίηση στα UNIX (στην έκδοσης 7, V7) της κλήσης συστήματος chroot. Αυτή η κλήση συστήματος είχε τη δυνατότητα να αλλάζει τον κατάλογο-ρίζα του συστήματος αρχείων μίας διεργασίας και των παιδιών της, δημιουργώντας έτσι μία διαφορετική όψη του συστήματος αρχείων στο εσωτερικό του chroot (γνωστό πλέον ως chroot jail), προσφέροντας κατ' αυτόν τον τρόπο μία πρώτη μορφή απομόνωση διεργασιών.

Λίγο αργότερα δημιουργήθηκε η έννοια του namespace από το κατανεμημένο λειτουργικό σύστημα Plan 9 της Bell Labs. Το λειτουργικό λόγω της κατανεμημένης φύσης του, χρησιμοποιούσε εκτενώς τα namespaces προκειμένου να μπορέσει να παρέχει σε κάθε διεργασία ένα ξεχωριστό χώρο εκτέλεσης, ο οποίος αναγνωρίζεται (για κάθε διεργασία) από ένα μοναδικό namespace και επηρεάζει κομμάτια το λειτουργικού, όπως το σύστημα αρχείων, τα mounts και το IPC (interprocess communication). Για παράδειγμα δύο διεργασίες μπορεί να βλέπουν χρησιμοποιώντας το ίδιο path δύο διαφορετικά αρχεία, επειδή βρίσκονται σε διαφορετικά namespaces.

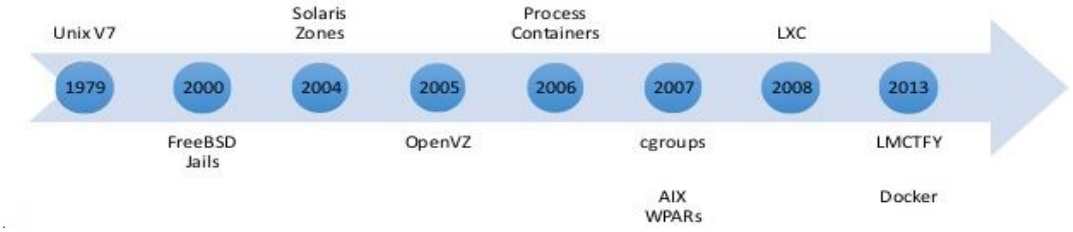
Η επόμενη σημαντική εξέλιξη ήρθε σχεδόν μία δεκαετία μετά από το Plan 9, το 2000, με τα FreeBSD jails προκειμένου να προσφέρει σαφή διαχωρισμό και ασφάλεια μεταξύ των υπηρεσιών και ευκολότερη διαχείριση αυτών. Η κλήση συστήματος jail παρείχε υπηρεσίες sandboxing, όπως απομόνωση του συστήματος αρχείων, των χρηστών, των διεργασιών και του δικτύου. Έτσι, με την κλήση jails δημιουργούνταν μικρά, απομονωμένα συστήματα, γνωστά ως jails, τα οποία μπορούσαν να έχουν τη δική τους IP, διαφορετικές εγκαταστάσεις ανά jail και διαφορετικά configurations. Αυτό έως το 2004 είχε οδηγήσει στην καθιέρωση του όρου jailbreak.

Το 2001 εμφανίστηκε ο μηχανισμός Linux VServer, ένας μηχανισμός που επέτρεπε τον διαχωρισμό διαφόρων τμημάτων του λειτουργικού, όπως το σύστημα αρχείων, οι διευθύνσεις δικτύου, η μνήμη και ο χρόνος CPU, αν και έλειπαν προχωρημένα features όπως αυτά που παρέχουν σήμερα τα namespaces. Ήταν η πρώτη φορά, όπου μία τεχνική εικονοποίησης επιπέδου λειτουργικού προσέφερε ένα σαφώς ορισμένο security context για κάθε χρήστη, δίνοντας του την αίσθηση ότι τρέχει σε ένα απομονωμένο μηχάνημα, ενώ στην πραγματικότητα έτρεχε σε έναν ιδιωτικό εικονικό εξυπηρετητή (virtual private server).

Το 2006, άρχισαν να αναπτύσσονται οι Process Containers, μία τεχνολογία που επέτρεπε σε ομάδες από διεργασίες να έχουν σαφή οριοθέτηση της χρήσης ορισμένων πόρων του συστήματος, όπως CPU, μνήμη, I/O δίσκου ή δικτύου κλπ. Τελικά οι Process Containers μετονομάστηκαν σε control groups ή cgroups για συντομία, για να αποφευχθεί οποιαδήποτε

σύγχυση με τον όρο container, καθώς ήδη από εκείνη την εποχή ήταν όρος που χρησιμοποιούνταν αρκετά και με διαφορετικές ονομασίες. Τελικά, τα control groups εντάχθηκαν το 2008 στον στάνταρ πυρήνα στην έκδοση 2.6.24 (μετά από αρκετές τροποποιήσεις που επέτρεπαν καλύτερη διαχείριση και έλεγχο περισσότερων πόρων του συστήματος).

Από το 2008 και μετά αρχίζουν αν εμφανίζονται οι πρώτες πλατφόρμες για Linux Containers, με πρώτο το LXC, το οποίο παρείχε εργαλεία σε χώρο χρήστη για τη δημιουργία και την διαχείριση containers αξιοποιώντας τα namespaces και τα control groups. Άλλα projects που ακολούθησαν και ασχολήθηκαν με τεχνολογίες σχετικές με containers είναι το Cloud Foundry Warden (αρχικά project ανοιχτού κώδικα από τη VMware, 2011), LMCTFY (Let Me Contain That For You, project ανοιχτού κώδικα της Google, 2013), Docker (2013), Kubernetes (project ανοιχτού κώδικα της Google, 2014) και Rocket (υλοποίηση του CoreOS, 2014).

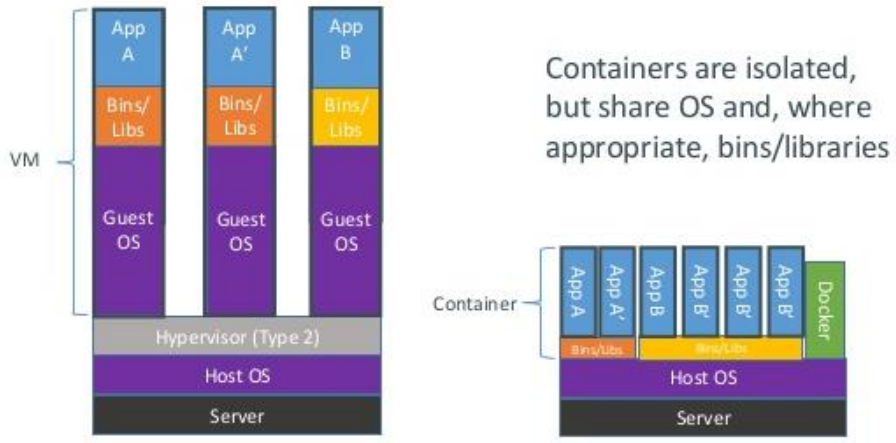


Σχήμα 2.6 Ιστορία των containers

Στην παρούσα διπλωματική γίνεται χρήση της πλατφόρμας του Docker και του Kubernetes που θα αναλυθεί σε επόμενο κεφάλαιο.

2.5.2 Οφέλη και Χρήσεις

Τα πλεονεκτήματα που προσφέρουν τα containers είναι ποικίλα και μπορούν να αξιοποιηθούν σε ένα ευρύ φάσμα εφαρμογών. Κατ' αρχάς, προσφέρουν ένα απομονωμένο περιβάλλον εκτέλεσης με ελάχιστο έως μηδενικό overhead. Αυτός είναι και ο κυριότερος λόγος για τον οποίον το ενδιαφέρον για τα containers έχει αυξηθεί σημαντικά τα τελευταία χρόνια, καθώς μπορούν να μειώσουν σημαντικά τα κόστη σε σχέση με τη χρήση εικονικών μηχανών, αφού το διαθέσιμο hardware αξιοποιείται πολύ πιο αποδοτικά (για παράδειγμα σε ένα συμβατικό laptop υπολογιστή μπορεί κανείς εύκολα να εκκινήσει 100 containers ταυτόχρονα, ενώ μπορεί να εκτελεί λιγότερο από 10 εικονικές μηχανές ταυτόχρονα).

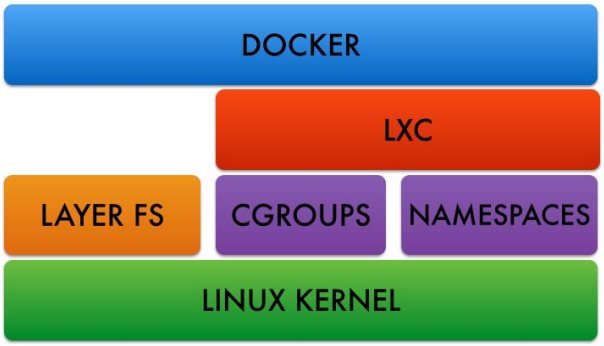


Σχήμα 2.7 Αρχιτεκτονική VM και container

Επίσης, προσφέρουν αποτελεσματική απομόνωση των χρησιμοποιούμενων πόρων, όπως μνήμη, CPU και ταχύτητα I/O, με δυνατότητες ρύθμισης των πόρων που δίνονται στον κάθε container κατά την εκτέλεση της εφαρμογής, ανάλογα με τις ανάγκες που προκύπτουν. Ακόμη, επιτρέπουν το εύκολο πακετάρισμα και διανομή των εφαρμογών μαζί με τις απαραίτητες ρυθμίσεις τους, γεγονός που μειώνει αισθητά τον κόπο και τον χρόνο που απαιτείται, ώστε να εγκατασταθεί και να είναι έτοιμη για χρήση μία εφαρμογή. Το παραπάνω πλεονέκτημα κάνει τα containers κατάλληλα για σύγχρονες αρχιτεκτονικές τύπου PaaS ή microservices και για μοντέλα ανάπτυξης εφαρμογών τύπου DevOps και Continuous Integration / Continuous Delivery (CI / CD). Συνεπώς, οι containers μπορούν να χρησιμοποιηθούν για απαιτητικές, κατανεμημένες, αξιόπιστες και highly available εφαρμογές.

2.5.3 Docker

Το Docker [28] είναι το πλέον διάσημο και ευρέως χρησιμοποιούμενο σύστημα διαχείρησης container. Αναπτύχθηκε αρχικά σαν εσωτερικό project μιας εταιρείας που δημιουργούσε λύσεις PaaS και λεγόταν DotCloud και αργότερα μετονομάστηκε σε Docker. Το Docker χρησιμοποιούσε LXC στα αρχικά στάδια και αργότερα αντικατέστησε το LXC με δική του βιβλιοθήκη που λέγεται libcontainer.



Σχήμα 2.8 Επίπεδα λειτουργικού κάτω από το Docker

Το Docker εισήγαγε ένα ολόκληρο οικοσύστημα για την διαχείρηση containers. Αρχικά, το Docker διαθέτει ένα repository από έτοιμα containers, ή, κατά την ορολογία που χρησιμοποιείται στο Docker, εικόνες (images). Εκεί μπορεί κανείς να βρει μεγάλη ποικιλία από εφαρμογές έτοιμες προς εκτέλεση, όπως βάσεις δεδομένων, web servers ή κάποιο έτοιμο λειτουργικό (χρησιμοποιώντας πάντα τον πυρήνα του host), τα οποία μπορούν να εκκινήσουν με τη χρήση μίας εντολής. Επιπλέον, διευκολύνει κατά πολύ το πακετάρισμα μίας εφαρμογής, αφού παρέχει τα κατάλληλα εργαλεία, ώστε να αποθηκευτούν οι κατάλληλες ρυθμίσεις, τα αρχεία και οι βιβλιοθήκες που χρειάζεται σε ένα και μόνο αρχείο, το οποίο ονομάζεται dockerfile.

Μέσα σε ένα dockerfile περιέχονται οδηγίες προς το Docker για την κατασκευή του επιθυμητού image (όπως πακέτα λογισμικού, ρυθμίσεις δικτύου, ορισμός χρηστών), το οποίο στη συνέχεια μπορεί να εκτελεστεί δημιουργώντας ένα instance, δηλαδή ένα container. Αξίζει να σημειωθεί ότι το dockerfile μεταφέρεται εύκολα, λόγω του μικρού μεγέθους, αφού περιέχει μόνο τις οδηγίες κατασκευής της εικόνας και όχι την ίδια την εικόνα. Ωστόσο αν κάποιος επιθυμεί μπορεί εύκολα να αποθηκεύσει σε κάποιο repository (είτε ιδιωτικό είτε δημόσιο) την εικόνα, ώστε να την μοιράζεται απευθείας με τους συνεργάτες του, γεγονός που είναι χρήσιμο σε περιπτώσεις που τα containers περιέχουν δεδομένα. Τέλος, το Docker προσφέρει εργαλεία που βοηθούν στην διαχείριση πολλών μηχανημάτων ταυτόχρονα, ώστε να διευκολύνεται η εγκατάσταση του Docker σε πολλούς servers και η διαχείριση (δημιουργία, εκκίνηση, κλπ) των containers σε αυτούς, καθώς και το πακετάρισμα και η διαχείριση εφαρμογών που απαρτίζονται από πολλά containers, ορίζοντας τις κατάλληλες παραμέτρους σε ένα απλό αρχείο αντίστοιχο του dockerfile.

2.6 Kubernetes

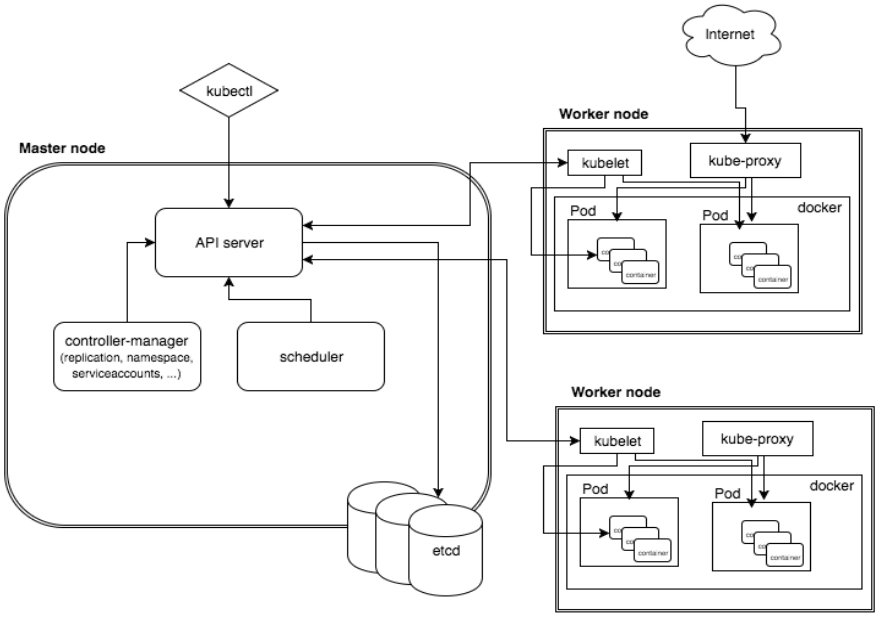
Το Kubernetes [29] (από την ελληνική λέξη Κυβερνήτης) είναι ένα σύστημα ανοιχτού κώδικα που έχει αναπτυχθεί από τη Google για την διαχείριση containerized εφαρμογών που τρέχουν σε μια συστάδα κόμβων (cluster of nodes). Στην παρούσα ενότητα θα αναλυθούν η αρχιτεκτονική του συστήματος, τα προβλήματα που επιλύει και το μοντέλο που χρησιμοποιεί για να χειριστεί τις εφαρμογές και το scaling αυτών.

Η μετάβαση από την ανάπτυξη μονολιθικών εφαρμογών στις containerized επιφέρει ευελιξία στον developer όσον αφορά στη διαχείριση της ανάπτυξης των εφαρμογών. Όσα όμως περισσότερα συστατικά κομμάτια services περιλαμβάνει μια εφαρμογή τόσο περισσότερο απαιτείται διαχείριση αυτών των κομματιών. Ο developer οφείλει να λαμβάνει υπόψη του τον προγραμματισμό σε έναν ορισμένο αριθμό containers σε συγκεκριμένους κόμβους του cluster, να διαχειρίζεται τη δικτύωση μεταξύ των containers, να παρακολουθεί την κατανομή των πόρων και να τους μετακινεί ανάλογα με την ανάγκη.

Ενώ με τα περισσότερα λογισμικά για clustering είναι δυνατόν να επιτευχθούν αυτές οι προγραμματιστικές αποφάσεις, η λειτουργία σε επίπεδο service δεν είναι ιδανική. Οι εφαρμογές που αποτελούνται από διαφορετικά services πρέπει να ενορχηστρώνονται ως μία ενιαία εφαρμογή. Το Kubernetes παρέχει ένα επίπεδο πάνω από το infrastructure για να επιτρέπει αυτού του είδους τη διαχείρηση.

2.6.1 Αρχιτεκτονική

Ένα Kubernetes cluster περιλαμβάνει διάφορα τμήματα, κάποια από αυτά προαιρετικά, και κάποια υποχρεωτικά για να λειτουργήσει. Στην παρακάτω είκόνα παρουσιάζεται μια απεικόνιση της αρχιτεκτονικής.



Εικόνα 2.9 – Αρχιτεκτονική Kubernetes

2.6.2 Pods

Το Kubernetes στοχεύει στη διαχείριση ελαστικών εφαρμογών που αποτελούνται από πολλαπλές μικρές υπηρεσίες που επικοινωνούν μεταξύ τους. Συχνά, αυτές οι υπηρεσίες είναι στενά συνδεδεμένες και φτιάχνουν μια ομάδα containers. Αυτή η ομάδα είναι η μικρότερη μονάδα εργασίας που μπορεί να προγραμματιστεί στον Kubernetes και λέγεται pod.

Το ή τα containers που αποτελούν το pod διαχειρίζονται ως μία μονάδα και μοιράζονται resources, volumes και IP, προγραμματίζονται μαζί και μπορεί να αναπτύσσονται και να γίνονται scale ως μία εφαρμογή. Το γενικό σχέδιο των pods περιλαμβάνει ένα βασικό container που ικανοποιεί τον γενικό σκοπό του pod και προαιρετικά μερικά βοηθητικά container που εξυπηρετούν σχετικές εργασίες. Το οριζόντιο scaling γενικά δεν συνίσταται σε επίπεδο pod επειδή υπάρχουν άλλες μονάδες πιο κατάλληλες για αυτην την εργασία. Τα pods δεν έχουν μεγάλη διάρκεια ζωής, δημιουργούνται και καταστρέφονται και ξαναδημιουργούνται όταν χρειάζεται σύμφωνα με την κατάσταση της εφαρμογής και του κόμβου.

2.6.3 Services

Εφόσον τα pods έχουν μικρή διάρκεια ζωής, δεν υπάρχει εγγύηση για την IP διεύθυνσή τους. Αυτό θα έκανε δύσκολη την επικοινωνία των containers. Επομένως, το Kubernetes εισάγει την έννοια του service. To service ομαδοποιεί λογικές ομάδες από pods τα οποία εκτελούν

την ίδια λειτουργία ώστε να παρουσιάζονται σαν μία οντότητα και λειτουργεί σαν βασικός load balancer και πρεσβευτής για άλλα containers.

Με αυτόν τον τρόπο το service που υλοποιείται γνωρίζει όλα τα containers στα οποία μπορεί να προωθήσει κίνηση. Εξωτερικές εφαρμογές χρειάζεται να γνωρίζουν ένα μοναδικό σημείο πρόσβασης αλλά επωφελούνται επίσης από ένα scalable backend. Η IP διεύθυνση ενός service παραμένει σταθερή χωρίς να επηρεάζεται από αλλαγές που συμβαίνουν στις διευθύνσεις των pods ή των κόμβων.

Τα services είναι μια διεπαφή σε μια ομάδα containers ώστε οι χρήστες να μην χρειάζεται να ανησυχούν για τίποτα παρά για ένα σημείο πρόσβασης. Αναπτύσσοντας ένα service η σχεδίαση των containers μπορεί να απλουστεύσει.

2.7 Network Monitoring

Με τον όρο Network Monitoring συνήθως αναφερόμαστε σε ενέργειες που πραγματοποιούνται από διαχειριστές δικτύων υπολογιστών με στόχο την παρακολούθηση και ανάλυση της δικτυακής κίνησης στο δίκτυο υπό την εποπτεία τους. Με βάση το [45] Τα πλεονεκτήματα της παρακολούθηση του δικτύου και αποθήκευση των δεδομένων αυτών ("Network Monitoring Data") είναι πολύπλευρα:

- ✓ Ασφάλεια
- ✓ Ποιότητα υπηρεσιών
- ✓ Εντοπισμός προβλημάτων
- ✓ Πρόβλεψη μελλοντικών αναγκών
- ✓ Ευκολίες στην αποσφαλμάτωση (debugging)

Οι τρόποι παρακολούθησης ποικίλουν. Παραδείγματος χάριν συχνά παρακολουθούνται δικτυακές συσκευές ή οι hosts που συνδέονται σε αυτές. Η παρούσα εργασία επιικεντρώνεται στον τομέα της δικτυακής κίνησης. Η παρακολούθηση έχει διαφορετικές μορφές, με τις πλέον διαδεδομένες να είναι (1) το καθρέφτισμα (mirror) και (2) η δειγματοληψία (sampling) της κίνησης.

Στην περίπτωση του mirroring, αυτούσια η κίνηση που διέρχεται από το υπό παρακολούθηση στοιχείο αντιγράφεται («καθρεφτίζεται») και έχει μια συγκεκριμένη μεταχείριση. Όσον αφορά την δειγματοληψία, η υλοποίηση της επιλέγει πακέτα με κάποια συγκεκριμένη συχνότητα και επίσης τα μεταχειρίζεται κατάλληλα. Συνήθως αυτή η μεταχείριση συνίσταται στην αποστολή της κίνησης (mirrored / sampled) σε ένα μηχάνημα που είναι αφιερωμένο στην συλλογή και στη συνέχεια, επεξεργασία των στοιχείων αυτών.

Υπάρχουν ποικίλοι τρόποι για να πετύχουμε την εξαγωγή και αποστολή των δεδομένων αυτών σε μηχανήματα για περαιτέρω επεξεργασία, καθένας βασισμένος σε διαφορετικά πρότυπα και πρωτόκολλα (π.χ. NetFlow, JFlow, sFlow). Ορισμένα μάλιστα είναι vendor specific, και παρέχουν ολοκληρωμένο περιβάλλον τόσο εξαγωγής όσο και ανάλυσης δεδομένων. Τα μηχανήματα που δέχονται τα δεδομένα συνήθως αποκαλούνται συλλέκτες (collectors) και υπάρχει πληθώρα διαφορετικών εφαρμογών που επιτελούν την διαδικασία της συλλογής και ανάλυσης (π.χ. Ganglia [30]).

Όσον αφορά την σύγκριση των μεθόδων παρακολούθησης, μπορεί το mirror της κίνησης να αποδίδει με τον πιο πιστό τρόπο την κατάσταση, αλλά στα σύγχρονα περιβάλλοντα τα οποία χαρακτηρίζονται από κίνηση μεγάλου όγκου, είναι μια διαδικασία που απαιτεί πολλούς πόρους τόσο στα μηχανήματα που εκτελούν το mirroring (switches, hosts) όσο και στους collectors – analyzers. Ειδικότερα, όσον αφορά τα switches επιβαρύνονται βασικά στοιχεία όπως η CPU κάτι που μπορεί να μειώσει την μεταγωγική ικανότητα του ίδιου του switch. Επίσης, καθώς η κίνηση από διάφορα ports αθροίζεται η ζεύξη στην οποία κατευθύνεται το mirror ενδεχομένως να υπερφορτωθεί προκαλώντας packet loss, delays, buffer overflow. Ωστόσο, ακόμα μεγαλύτερο πρόβλημα εντοπίζεται στα μηχανήματα που επεξεργάζονται την ροή της πληροφορίας καθώς αυτά συνήθως υλοποιούν πολυπλοκότερους αλγόριθμους για την επεξεργασία των πακέτων. Ρεαλιστική λύση λοιπόν από πλευρά κλιμακωσιμότητας (“scalability”) σε περιβάλλοντα με υψηλό όγκο κίνησης (όπως τα δίκτυα κορμού ενός Internet Provider ή ένα Data Center) αποτελεί η δειγματοληψία πακέτων με κάποια αναλογία πακέτων.

2.7.1 Sflow

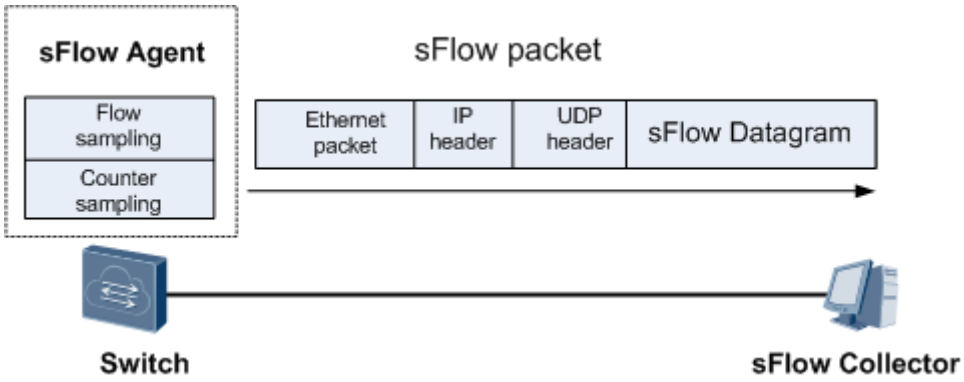
Στα πλαίσια της παρούσας εργασίας, υλοποιήθηκε ένας μηχανισμός που χρησιμοποιεί δειγματοληψία για την εξαγωγή των δεδομένων, χρησιμοποιώντας το sFlow [10]. Το sFlow, (sampled flow), αποτελεί ένα πρότυπο τρόπο για την βιομηχανία όσον αφορά την δειγματοληψία πακέτων. Δίνει τη δυνατότητα εξαγωγής περικομμένων (“truncated”) πακέτων παράλληλα με μετρητές για τα interfaces. Όσον αφορά τις μεγάλες υποδομές, το sFlow χρησιμοποιεί δειγματοληψία και έτσι παρέχει μια λύση που είναι δελεαστική (λόγω scalability) στα σύγχρονα περιβάλλοντα.

Σχολιάζοντας την συμβατότητα του sFlow με διάφορα περιβάλλοντα, σημειώνεται πως υποστηρίζεται από δικτυακές συσκευές πολλαπλών κατασκευαστών. Επίσης, τα δεδομένα στην μορφή που εξάγονται είναι αναγνωρίσιμα από μια πληθώρα εργαλείων διαχείρισης και παρακολούθησης. Ένα sFlow σύστημα μπορεί να απαρτίζεται από πολλαπλές συσκευές οι οποίες πραγματοποιούν δειγματοληψία:

- Με βάση κάποια πιθανότητα (πακέτων και ενεργειών του επιπέδου εφαρμογής)

- Ανά χρονικές περιόδους (μετρητές)

Τα δείγματα των πακέτων/ενεργειών και μετρητών αποκαλούνται flow samples και counter samples αντίστοιχα. Τα δεδομένα αποστέλλονται μέσω δεδομενογραμμάτων (“datagrams”) σε ένα μηχάνημα που τα επεξεργάζεται κατάλληλα. Επίσης, συνήθως παρουσιάζει αποτελέσματα για την δικτυακή κίνηση με βάση τα δεδομένα που συλλέχθηκαν. Αυτό αποκαλείται συνήθως στην ορολογία sFlow collector.



Σχήμα 2.10 Πακέτο Sflow

- **Flow Samples**

  Βασιζόμενο πάνω σε ένα ρυθμό δειγματοληψίας n, κατά μέσο όρο 1 από n πακέτα (ή άλλες πληροφορίες) επιλέγονται τυχαία. Παρόλο που δεν αποτυπώνονται αποτελέσματα με την μέγιστη ακρίβεια, η προσέγγιση είναι ικανοποιητική.

- **Counter Samples**

  Το διάστημα της μέτρησης (“polling interval”) ορίζει την χρονική συχνότητα με την οποία η συσκευή θα στέλνει τους μετρητές για τα interfaces της. Αυτές οι μετρήσεις παρέχουν με άμεσο τρόπο το εύρος ζώνης της γραμμής σε αυτό το χρονικό διάστημα. Τα sFlow counter samples είναι αποδοτικότερος τρόπος παρακολούθησης από το SNMP όταν παρακολουθείται μεγάλο πλήθος interfaces.

- **sFlow datagrams**

  Τα δεδομένα των δειγμάτων στέλνονται σαν UDP πακέτα στην IP και port που ορίζονται για τον collector (η πόρτα συνήθως για τον sFlow collector είναι η 6343). Παρόλο που χρησιμοποιείται το πρωτόκολλο UDP (πιθανόν να χαθεί κάποιο πακέτο καθώς το UDP δεν είναι τόσο αξιόπιστο πρωτόκολλο), η απώλεια δεν έχει σημαντικό αντίκτυπο στην ακρίβεια των μετρήσεων που πραγματοποιεί ένας sFlow agent. Αυτό δικαιολογείται καθώς, στην περίπτωση απώλειας των counter samples θα ανανεωθούν κατάλληλα οι τιμές στο επόμενο χρονικό διάστημα ενώ στην περίπτωση απώλειας των packet samples

επί της ουσίας έχουμε μείωση του ρυθμού δειγματοληψίας. Στο εσωτερικό του UDP πακέτου είναι ενθυλακωμένο το sFlow datagram, το καθένα από τα οποία παρέχει τις εξής πληροφορίες:

- Έκδοση sFlow
- Διεύθυνση IP του agent
- Έναν αύξοντα αριθμό
- Το πλήθος των samples που περιέχονται στο sFlow datagram
- Τα ίδια τα δείγματα

Οι εγγραφές για τα δείγματα που έχουν να κάνουν με τα Flow Samples περιλαμβάνουν:

- Η πόρτα εισόδου και εξόδου από το switch
- Το IP πρωτόκολλο που χρησιμοποιεί το πακέτο
- Τον τύπο της υπηρεσίας (IPTOS)
- Το IP TTL
- Το vLAN πηγής και προορισμού
- Την πόρτα και την IP πηγής και προορισμού του αρχικού πακέτου
- Τυχόν σημαίες (flags) σε περιπτώσεις πακέτου TCP
- Το μέγεθος του πακέτου
- Τον ρυθμό δειγματοληψίας

2.7.2 Netflow

Το Netflow [11] είναι ένα πρωτόκολλο που αρχικά προτάθηκε από την Cisco στους δρομολογητές της, με σκοπό τη συλλογή της κίνησης που εξέρχεται ή εισέρχεται από ένα ή σε ένα interface του δρομολογητή. Ωστόσο, πλέον έχει προτυποποιηθεί και υποστηρίζεται κι από άλλους κατασκευαστές. Τα δεδομένα που παρέχονται στον διαχειριστή του δικτύου μέσω του Netflow αφορούν λεπτομέρειες της κίνησης του δικτύου οι οποίες αναφέρονται παρακάτω και βοηθούν στην παρακολούθηση του. Το σύνηθες σενάριο ενεργοποίησης του Netflow, είναι:

- η ενεργοποίηση του στα interfaces των δρομολογητών
- η εξαγωγή των δεδομένων αυτών στο συλλέκτη της κίνησης
- η ανάλυση της προκύπτουσας πληροφορίας

Η ενεργοποίηση του Netflow γίνεται ανά δρομολογητή ανά interface και με τον ίδιο τρόπο ορίζεται και η διεύθυνση καθώς και η θύρα στην οποία θα σταλούν τα αντίστοιχα δεδομένα. Συνήθως η μεταφορά των δεδομένων γίνεται πάνω από το πρωτόκολλο UDP, ωστόσο λόγω της φύσης του (μη ενημέρωση για την απώλεια πακέτων) σε κάποιες σύγχρονες υλοποιήσεις

χρησιμοποιείται το πρωτόκολλο SCTP (Stream Control Transmission Protocol), για την αποφυγή απώλειας πακέτων.

Η εξαγωγή της πληροφορίας στον συλλέκτη γίνεται μετά από κάποιο ορισμένο χρονικό διάστημα. Τα όρια αυτά μπορούν να ρυθμιστούν στον δρομολογητή και όταν η αντίστοιχη πληροφορία σταλεί, διαγράφεται οριστικά από την μνήμη του δρομολογητή. Η μνήμη στην οποία αποθηκεύονται οι πληροφορίες που σχετίζονται με το Netflow είναι μία μνήμη cache στον δρομολογητή με χαρακτηριστικά που μπορούν να ρυθμιστούν και αυτά από τον διαχειριστή του δικτύου. Η δομή του πακέτου Netflow διακρίνεται σε δύο στοιχεία στην επικεφαλίδα και στις εγγραφές ροών. Συγκεκριμένα η επικεφαλίδα του πακέτου περιέχει πληροφορίες σχετικά με:

- την έκδοση του πρωτοκόλλου
- τον αριθμό ακολουθίας του πακέτου
- τον χρόνο εξαγωγής του πακέτου όπως αυτός προέκυψε από τον δρομολογητή
- τον αριθμό των ροών που ακολουθούν

Η εγγραφή για τις ροές περιέχει τα εξής στοιχεία:

- το interface εισόδου όπως αυτό αναγράφεται στο πρωτόκολλο SNMP
- το interface εξόδου ή τον αριθμό μηδέν αντί αυτού αν το πακέτο απορρίφθηκε
- τις χρονικές στιγμές εκκίνησης και λήξης της ροής
- τον αριθμό των πακέτων και των bytes που παρατηρήθηκαν στην αντίστοιχη ροή
- τη διεύθυνση πηγής και προορισμού
- τον τύπο και τον κωδικό του ICMP
- το πρωτόκολλο IP και την τιμή TOS (Type of Service)
- τις θύρες πηγής και προορισμού των πρωτοκόλλων TCP, UDP, SCTP
- αν πρόκειται για TCP ροές, περιέχονται και πληροφορίες για τις σημαίες των TCP πακέτων

2.8 Tornado Web Server (API)

Το Tornado [31] είναι ένα web framework της Python που αναπτύχθηκε από την FriedFeed (εξαγοράστηκε από την FaceBook) το 2009. Παράλληλα, λειτουργεί ως non-blocking web server και asynchronous network library. Έτσι, ειδικεύεται στις υψηλές επιδόσεις υποστηρίζοντας πάνω από 10.000 παράλληλες συνδέσεις (C10K problem), ιδανικό για long-polling (send, push δεδομένα χωρίς request), web socket και long-lived connections. Προσφέρεται ως fullStack με τον δικό του HTTP-Server υποστηρίζοντας και το WGSI. Τέλος

διαθέτει πλήρες documentation έχοντας αρκετά ενεργή κοινότητα στο GitHub, μ' αυτό έχουν στηθεί τα Bit.ly και Quora.

2.9 Zookeeper

Το ZooKeeper [32] είναι μία ανοιχτού κώδικα κατανεμημένη υπηρεσία, που παρέχει πληροφορίες, ονοματοδοσία και συγχρονισμό σε κατανεμημένα συστήματα. Όλα τα παραπάνω χαρακτηριστικά αποτελούν βασικότατους παράγοντες επιτυχίας όλων των σύγχρονων κατανεμημένων συστημάτων. Είναι ιδιαίτερα χρήσιμο ως εργαλείο διότι η υλοποίηση και η αποσφαλμάτωση εφαρμογών που εμπλέκουν συγχρονισμό είναι εξαιρετικά επίπονες διαδικασίες για τον προγραμματιστή, λόγω των συνθηκών ανταγωνισμού (race conditions) και των αδιεξόδων (deadlocks) που παρουσιάζονται.

Το ZooKeeper τρέχει σε cluster υπολογιστών και παρέχει πληροφορίες έκδοσης και κατάστασης των δεδομένων (versioning), ταξινόμηση με βάση την ώρα δημιουργίας και τροποποίησης τους, όπως επίσης ένα πολύ εύχρηστο σύστημα ενημερώσεων όταν συμβεί κάποιο μη επιθυμητό γεγονός (για παράδειγμα αν χαθεί η επικοινωνία με κάποιον κόμβο). Κάθε cluster μπορεί να περιλαμβάνει έναν η περισσότερους ZooKeeper εξυπηρετητές, ανάλογα με το μέγεθός του. Όλες οι κατανεμημένες διεργασίες μπορούν και συνεργάζονται μεταξύ τους κάνοντας χρήση των παραπάνω μηχανισμών, αλλά και ενός συστήματος αρχείων δεντρικής μορφής. Όλα τα δεδομένα κρατούνται στην κύρια μνήμη γεγονός που κάνει το ZooKeeper να επιτυγχάνει πολύ υψηλή απόδοση (throughput) στην επικοινωνία. Το ZooKeeper επίσης παρέχει πολύ υψηλή διαθεσιμότητα. Όλοι οι εξυπηρετητές κρατούν στην κύρια μνήμη πληροφορίες κατάστασης όλων των υπόλοιπων κόμβων του συστήματος, μαζί με αρχεία καταγραφής συναλλαγών, με αποτέλεσμα όταν οι περισσότεροι εξυπηρετητές είναι διαθέσιμοι, οι υπηρεσίες του ZooKeeper να είναι διαθέσιμες για τις εφαρμογές που τις χρησιμοποιούν.

2.10 Kafka

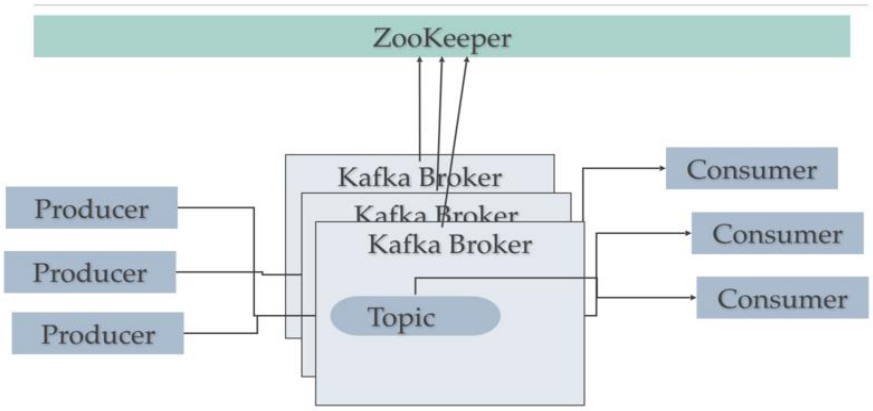
Το Kafka [33] είναι ένα ανοιχτού κώδικα κατανεμημένο σύστημα διαχείρισης μηνυμάτων (Messaging System), σχεδιασμένο να μπορεί να επεξεργάζεται πάρα πολύ μεγάλο όγκο δεδομένων, είτε αυτός προέρχεται από ροές (streams) είτε από δέσμες δεδομένων (batches). Αποτελεί ένα message broker, δηλαδή ένα ενδιάμεσο πρόγραμμα το οποίο μεταφράζει ένα μήνυμα από το επίσημο πρωτόκολλο μηνυμάτων του αποστολέα στο επίσημο πρωτόκολλο

μηνυμάτων του δέκτη. Είναι ένα project ανοιχτού κώδικα το οποίο αρχικά σχεδιάστηκε από την LinkedIn ως ένα κατανεμημένο σύστημα μηνυμάτων (messaging) για συλλογή και μεταφορά Log αρχείων με ελάχιστη καθυστέρηση (low latency) και high throughput σε πραγματικό χρόνο. Εκτός από την ίδια την εταιρία χρησιμοποιείται ευρέως από πολλές άλλες κορυφαίες του χώρου, όπως το Twitter και το Spotify. Πλέον, την ανάπτυξη του Kafka την έχει αναλάβει η Apache.

Το σύστημα λειτουργεί με βάση το μοντέλο παραγωγού-καταναλωτή (producer-consumer) και συντηρεί μηνύματα ομαδοποιημένα σε κατηγορίες οι οποίες ονομάζονται topics. Κάθε topic είναι στην ουσία μια ουρά μηνυμάτων. Ένας παραγωγός δημοσιεύει μηνύματα σε μια συγκεκριμένη κατηγορία και όσοι καταναλωτές είναι εγγεγραμμένοι στην κατηγορία-group αυτή λαμβάνουν το δημοσιευμένο μήνυμα.

Το Kafka δουλεύει σε cluster υπολογιστών, αποτελούμενο από έναν ή περισσότερους εξυπηρετητές, κάθε ένας από τους οποίους ονομάζεται broker. Ως εκ τούτου σε ένα τέτοιο κατανεμημένο σύστημα οι παραγωγοί στέλνουν μηνύματα στο cluster μέσω δικτύου, αυτά αποθηκεύονται σε ουρές των brokers και στη συνέχεια διατίθενται στους καταναλωτές που ενδιαφέρονται για αυτά.

Το Kafka βρίσκει χρήση στο messaging καθώς χαρακτηριστικά όπως high throughput, τεμαχισμός 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.



Σχήμα 2.11 Αρχιτεκτονική Kafka

Είναι πλήρως αποκεντρωμένο και χρησιμοποιεί το ZooKeeper και τους μηχανισμούς που παρέχονται από αυτό για να εγγυηθεί συγχρονισμό, υψηλή απόδοση και εξισορρόπηση φόρτου στους brokers. Οι παραγωγοί και οι καταναλωτές λαμβάνουν γνώση για τους

διαθέσιμους brokers μέσω του ZooKeeper. Στα βασικά χαρακτηριστικά του σχεδιασμού περιλαμβάνεται η προσωρινή ή αποθήκευση μηνυμάτων (caching), η δυνατότητα να ξανά καταναλωθούν μηνύματα (re-consumption), η ομαδοποίηση μηνυμάτων (για μείωση δικτυακού φόρτου), η συμπίεση μηνυμάτων κ.α.

3

Σχεδίαση προαπαιτούμενων

Αρχικά, θα γίνει παρουσίαση της λογικής που υπάρχει πίσω από ένα σύστημα που σου παρέχει μια υπηρεσία κατά απαίτηση (on demand). Επίσης, λόγω της δικτυακής φύσης του use case που χρησιμοποιείται σχετικά με την καταγραφή και τον εμπλουτισμό sFlow κίνησης παρατηρούνται ομοιότητες με την φιλοσοφία πίσω από τον σχεδιασμό και την λειτουργική αρχιτεκτονική δομή μιας εικονοποιημένης δικτυακής υπηρεσίας (Network Function Virtualization, NFV [34]).

3.1 Ανάλυση Απαιτήσεων

Τα δεδομένα και οι απαιτήσεις που έχει ένας χρήστης από ένα τέτοιο σύστημα θα βοηθήσουν στο σχεδιασμό και την εύρεση των χαρακτηριστικών που χρειάζονται.

- Καταγραφή δικτυακής κίνησης με μεγάλη ροή

Αρχίζοντας λοιπόν από την πηγή που θα τροφοδοτεί το σύστημα καταγραφής, μεγάλο ρόλο παίζει το είδος των δεδομένων και συγκεκριμένα, το μέγεθος και η ταχύτητα της κίνησης. Αυτό αποτελεί βασική απαίτηση που θα οδηγήσει το σύστημα καταγραφής να ανταπεξέρχεται σε μεγάλη ροή δεδομένων που θα λαμβάνει από το δίκτυο. Επίσης, αυτός θα είναι και ένας ενδεικτικός παράγοντας για το τον σχεδιασμό της αρχιτεκτονικής του συστήματος και τον υπολογισμό των υποδομών που χρειάζονται.

Η πηγή μπορεί να είναι τα διάφορα switches (μεταγωγείς) του δικτύου που δειγματοληπτούν την κίνηση που περνάει και τη στέλνουν στον λεγόμενο sFlow Agent. Από εκείνο το σημείο και ύστερα η κίνηση μπαίνει στην πλατφόρμα και καταγράφεται.

- Επεξεργασία-εμπλουτισμός των ροών δικτυακής κίνησης

Ύστερα από την τροφοδότηση του συστήματος καταγραφής μπορεί να χρειαστεί μια επεξεργασία των δεδομένων που λαμβάνονται για να εμπλουτιστούν με περεταίρω πληροφορία που θα προσδώσει μεγαλύτερη αξία στην καταγραφή και παρατήρησή τους.

Στη συγκεκριμένη περίπτωση θα χρειαστεί να γίνει μια αντιστοιχία των IPs από δημόσια σε ιδιωτική και το αντίστροφο μέσα από ένα json αρχείο με χιλιάδες γραμμές με αντιστοιχήσεις. Επίσης, είναι χρήσιμη η εύρεση πληροφοριών για τις IPs όπως για παράδειγμα την πόλη, χώρα, τις συντεταγμένες και το αυτόνομο σύστημα (AS) από όπου προέρχονται. Αυτές οι πληροφορίες μπορεί να δίνονται μέσα από μια βάση-αρχείο.

- Αποθήκευση των δεδομένων

Τα δεδομένα από το δίκτυο χρειάζεται να αποθηκεύονται κάπου για κάποιο διάστημα για να μπορεί κάποιος να ανατρέχει σε παλιότερα δεδομένα και να μπορεί να συγκρίνει τα αποτελέσματα. Με αυτόν τον τρόπο δίνεται μεγαλύτερη αξία στο σύστημα καταγραφής λόγω του ότι δεν έχει να κάνει μόνο με την παρακολούθηση της δικτυακής κίνησης σε πραγματικό χρόνο αλλά και την ανάλυση αυτής κρατώντας δεδομένα για κάποιο διάστημα.

- Δυνατότητα διερώτησης των δεδομένων

Εφόσον τα δεδομένα είναι κάπου αποθηκευμένα οι χρήστες θα μπορούν να κάνουν οποιαδήποτε είδους ανάλυση χρειάζονται για να εξάγουν αποτελέσματα κάνοντας τις απαραίτητες ερωτήσεις πάνω σε αυτά.

Έτσι, προσφέρεται η δυνατότητα να δημιουργηθούν επιμέρους εργαλεία τα οποία θα αξιοποιούν αυτό το χαρακτηριστικό για να καλύψουν ενδεχόμενες άλλες ανάγκες που θα προκύψουν όπως για παράδειγμα η αναγνώριση επίθεσης ή ειδοποίηση όταν γίνει κάποιο γεγονός.

Αυτό πρόκειται για ένα απαραίτητο χαρακτηριστικό του συστήματος προκειμένου να δώσει

μεγαλύτερη αξία στους χρήστες καθώς επίσης ανοίγει τις πόρτες για περεταίρω ανάπτυξη του συστήματος σε μια ενιαία πλατφόρμα καταγραφής.

- Εποπτεία των δεδομένων

Σημαντικό στο όλο σύστημα είναι να μπορεί να σταθεί σαν πλατφόρμα για monitoring (καταγραφή) δικτυακής κίνησης. Για να γίνει αυτό χρειάζονται κάποια εργαλεία που να δίνουν μια γρήγορη εικόνα της κατάστασης της δικτυακής κίνησης στις υποδομές που καταγράφονται σχεδόν σε πραγματικό χρόνο.

Μέσα από εργαλεία για visualization (οπτικοποίηση), ομαδοποίηση και συσχέτιση των δεδομένων η πλατφόρμα θα μπορεί να προσφέρει τη δυνατότητα δημιουργίας γραφημάτων και ερωτημάτων στα υπάρχοντα δεδομένα που συλλέγονται.

Τα παραπάνω βασίζονται στη δυνατότητα αποθήκευσης και διερώτησης των δεδομένων.

- Δυνατότητα προσωποποιημένης καταγραφής δεδομένων σε διαφορετικούς ενοίκους

Προκειμένου η πλατφόρμα αυτή να έχει μεγαλύτερη αξία για τον κάθε χρήστη αλλά και να επιφέρει την ιδιωτικότητα μεταξύ των χρηστών, ο κάθε χρήστης θα πρέπει να έχει πρόσβαση και να έχει εικόνα μόνο των δικών του δικτυακών πακέτων που τον αφορά.
Κάτι τέτοιο μπορεί να γίνει ξεχωρίζοντας τους χρήστες με βάση κάποιο από τα πεδία του sflow όπως για παράδειγμα η IP ή το vlan πηγής ή προορισμού.

- Προσαρμοστικότητα του συστήματος

Τέλος, το σύστημα αυτό θα πρέπει να μπορεί να προσαρμόζεται και να ανταπεξέρχεται σε διαφορετικές ροές κίνησης που δέχεται από το δίκτυο και να χρησιμοποιεί κάθε φορά τους πόρους που χρειάζεται. Αυτό σημαίνει ότι η τελική πλατφόρμα θα περιλαμβάνει πολλά κομμάτια τα οποία θα μπορούν να δουλέψουν όλα μαζί και ανεξάρτητα καθώς επίσης και να μπορούν να μοιράζονται τον φόρτο από την κίνηση του δικτύου.

- Απόκριση σχεδόν σε πραγματικό χρόνο

Αυτό το χαρακτηριστικό θα δώσει μεγαλύτερη αξία σε ένα τέτοιο σύστημα αφού θα δώσει τη δυνατότητα παρατήρησης σχεδόν σε πραγματικό χρόνο. Για κάτι τέτοιο μεγάλο ρόλο έχουν

οι πόροι που θα αποδοθούν στην πλατφόρμα και πώς αυτή θα τους διαχειριστεί αποδοτικά ώστε να μπορεί να διαχειρίζεται τον όγκο και την ταχύτητα με την οποία την τροφοδοτούν.

Για τον υπολογισμό της κίνησης που θα περνάει από το σύστημα καταγραφής θεωρούμε συνολική κίνηση από τα switches της τάξης των 10Gbps = 1.25GBps. Αν υπολογίσουμε έναν μέσο όρο των 500Bytes ανά πακέτο τότε έχουμε ~2.5M pps. Με μια δειγματοληψία του 1 προς 256 πακέτα καταλήγει ο sFlow agent να παίρνει ~9K pps. Αυτή είναι περίπου και η κίνηση που πρέπει το σύστημα να μπορεί να διαχειρίζεται.

3.2 Ανάλυση χαρακτηριστικών

Τα υψηλού επιπέδου χαρακτηριστικά [35] που χρειάζονται πίσω από τον σχεδιασμό μιας τέτοιας ενιαίας πλατφόρμας-συστήματος πρέπει να ανταποκρίνονται στην κάλυψη των παραπάνω απαιτήσεων.

- Διαθεσιμότητα

Αυτό είναι από τις πιο βασικές προκλήσεις λόγω του ότι θα προσδώσει στην πλατφόρμα την ικανότητα που απαιτείται για να εξυπηρετεί συνέχεια. Στη θεωρία ένα τέτοιο σύστημα μπορεί να εξυπηρετεί 99,9% του χρόνου. Κάνοντας LBL την πλατφόρμα συνεχώς διαθέσιμη σημαίνει ότι συνεισφέρει στην ποιότητα της υπηρεσίας και στην εμπειρία του χρήστη.

Κάθε κομμάτι της πλατφόρμας θα πρέπει να έχει ένα ίδιο αντίγραφο να τρέχει σε διαφορετικό κόμβο ούτως ώστε να αποφευχθεί ο τερματισμός λειτουργίας οποιουδήποτε μέρους της και άρα όλου του συστήματος.

- Κλιμακωσιμότητα

Η κλιμακωσιμότητα έχει να κάνει με την προσθήκη πόρων σε ένα σύστημα είτε με τη μορφή παραπάνω hardware εσωτερικά (cpu, memory, etc) είτε με τη μορφή καινούριου μηχανήματος (server).

Μέσω αυτού η πλατφόρμα μπορεί να γίνει πιο ανθεκτική και να αυξήσει την αποδοτικότητά της. Ειδικότερα, σε τυχαίες βλάβες των μηχανημάτων πάντα θα χρειαστεί να υπάρχει κάποιος άλλος κόμβος που να τρέχει κάποιο από τα κομμάτια της πλατφόρμας ώστε να μη μειώσει

την ποιότητα της υπηρεσίας. Επίσης, αυξάνοντας τους πόρους με οποιαδήποτε μορφή θα έχει αντίκτυπο και στην απόδοση της τελικής υπηρεσίας προς τον χρήστη και άρα στον πραγματικό χρόνο απόκρισης που έχει αναφερθεί στις απαιτήσεις του συστήματος.

- Ιδιωτικότητα

Αυτό είναι το χαρακτηριστικό που συνδέεται με την απαίτηση που υπάρχει για προσωποποιημένη καταγραφή των δεδομένων καθώς επιτρέπει την ύπαρξη πληροφοριών πολλών οντοτήτων-ενοίκων αποτρέποντας ταυτόχρονα την πρόσβαση σε κομμάτι πληροφοριών χωρίς δικαιοδοσία και επιβεβαίωση ταυτότητας του αιτούμενου.

Η ιδιωτικότητα επίσης έχει να κάνει σε κάποιο βαθμό και με την ασφάλεια της πλατφόρμας εσωτερικά από την πλευρά της σχέσης μεταξύ των χρηστών και τη διασφάλιση ότι ο κάθε ένοικος έχει τον δικό του χώρο.

- Διαχείριση Κίνησης – Ενορχήστρωση

Η διαχείριση της κίνησης έχει να κάνει ουσιαστικά με την ενορχήστρωση της πλατφόρμας ή μέρους αυτής και με την αποδοτική διαχείριση των πόρων του συστήματος. Λόγω της έντονης και συνεχούς ροής κίνησης σε ένα δίκτυο, είναι απαραίτητο να επιβεβαιωθεί ο αριθμός πόρων που είναι κάθε φορά διαθέσιμος ώστε να ανταπεξέλθει στην κίνηση που λαμβάνει. Αυτό το χαρακτηριστικό συνδέεται και με την απαίτηση για προσαρμοστικότητα της πλατφόρμας. Ειδικότερα, η ενορχήστρωση παρέχει αυτόματη κλιμακωσιμότητα σε περίπτωση μεγαλύτερης ή μικρότερης κίνησης από το δίκτυο και διαβεβαιώνει ότι σε περίπτωση σφάλματος του υλικού (hardware failure) η υπηρεσία θα επανεκκινείται στον ίδιο ή σε άλλο κόμβο ώστε να είναι πάλι διαθέσιμη.

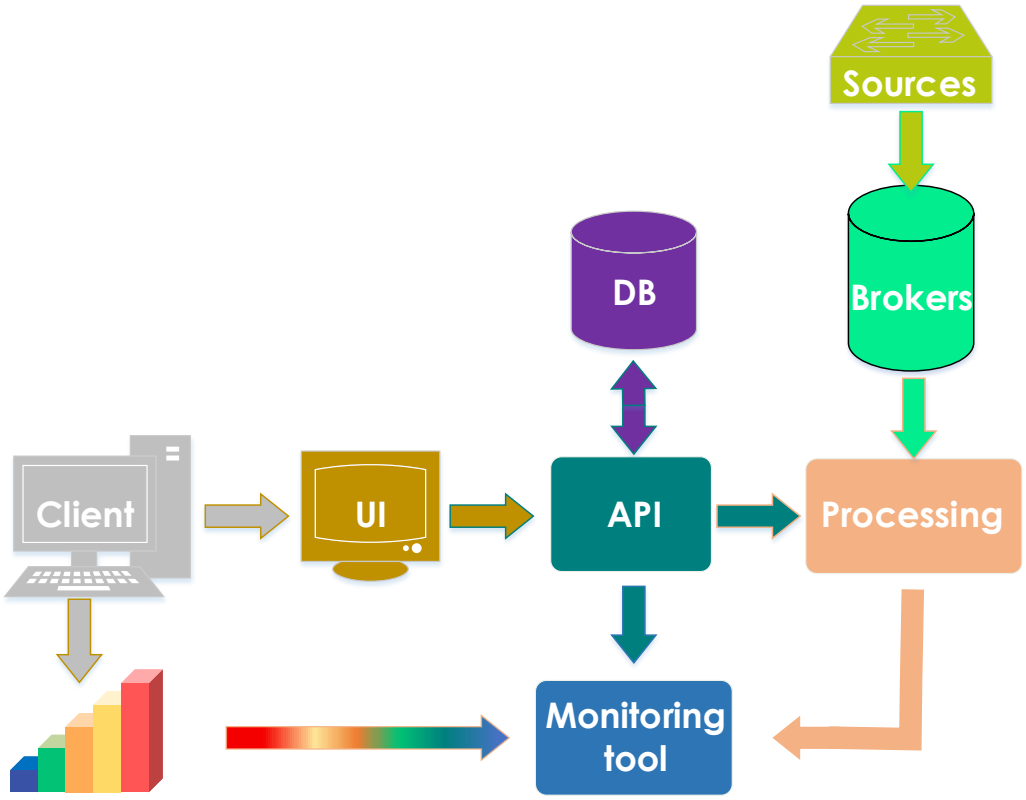
4

Ανάλυση Συστήματος

Εδώ θα ακολουθήσει η περιγραφή της αρχιτεκτονικής του συστήματος και θα γίνει η ανάλυση της αρχιτεκτονικής για τις λειτουργίες του.

4.1 Ανάλυση Αρχιτεκτονικής

Για την περεταίρω ανάλυση των χαρακτηριστικών και την εύρεση των επιμέρους στοιχείων ενός συστήματος που εξυπηρετεί κατ' απαίτηση χρειάζεται τέλος μια υψιλού επιπέδου ανάλυση της αρχιτεκτονικής του. Αυτή η ανάλυση έχει να κάνει με εκείνα τα στοιχεία που χρειάζονται από την αλληλεπίδραση με τον χρήστη μέχρι το κομμάτι της επεξεργασίας και αποθήκευσης.



Σχήμα 4.1 Αρχιτεκτονική πλατφόρμας κατά απαίτηση

- User Interface (UI)

Κάθε υπηρεσία που παρέχεται σε χρήστες χρειάζεται ένα σημείο αναφοράς όπου πέραν από την πρόσβαση στην υπηρεσία μπορούν να:

- πιστοποιούνται για το ποιοι είναι και πού έχουν δικαιοδοσία μέσα στην υπηρεσία
- ελέγχουν την κατάσταση της υπηρεσίας τους
- αλλάζουν τη διαμόρφωση και την σύνθεση της υπηρεσίας που τους παρέχεται

- Backend - API

Κάθε υπηρεσία εφαρμόζει τη λογική και την ευφυΐα όλης της υπηρεσίας κάπου κεντρικά και ονομάζεται backend της υπηρεσίας. Στη συγκεκριμένη περίπτωση το backend είναι ένα μια Διασύνδεση Προγραμματισμού Εφαρμογών ή αλλιώς API [36] (Application Programming Interface) που επιτρέπει την επικοινωνία με όλα τα επιμέρους κομμάτια της υπηρεσίας κεντρικά.

Πρόκειται για μια διεπαφή (API) των προγραμματιστικών διαδικασιών που παρέχει ένα σύστημα, βιβλιοθήκη ή εφαρμογή προκειμένου να επιτρέπει να γίνονται προς αυτά αιτήσεις

από άλλα προγράμματα ή/και ανταλλαγή δεδομένων. Σκοπός είναι να ορίζει και να διατυπώνει το σύνολο των λειτουργιών-υπηρεσιών που μπορεί να παρέχει μια βιβλιοθήκη ή ένα λειτουργικό σύστημα-πλατφόρμα σε άλλα προγράμματα, επιτρέποντας την επικοινωνία με άλλα κομμάτια της πλατφόρμας. Το τμήμα του λειτουργικού συστήματος το οποίο υλοποιεί τις υπηρεσίες που περιγράφονται στη διεπαφή, συνήθως στον πυρήνα του, λέγεται ότι είναι η υλοποίηση της διεπαφής. Ένα API συνήθως ορίζεται ως ένα σύνολο από HTTP ζητούμενα μηνύματα μαζί με έναν ορισμό της δομής των μηνυμάτων ανταπόκρισης ο οποίος είναι συνήθως σε μορφή επεκτάσιμης γλώσσας σήμανσης (XML) ή σημειογραφίας αντικειμένων JavaScript (JSON).

Ο τρόπος επικοινωνίας με το κεντρικό API για την αποστολή αιτήσεων σχετικά με την υπηρεσία γίνεται μέσω REST (Representational State Transfer). Η σύγχρονη τάση στον σχεδιασμό υπηρεσιών κατευθύνεται σε στυλ REST, ηλεκτρονικούς πόρους και resource-oriented architecture (ROA). Το REST είχε τόσο μεγάλο αντίκτυπο στο διαδίκτυο που έχει κυρίως εκτοπίσει το SOAP και την σχεδίαση διεπαφών βασισμένων σε WSDL, διότι είναι πολύ πιο απλουστευμένο στην χρήση. Μια συγκεκριμένη εφαρμογή ενός REST Web Service ακολουθεί τέσσερις βασικές αρχές σχεδιασμού:

- Αποκλειστική χρήση της μεθόδου HTTP.
- Ανεξαρτησία κατάστασης.
- Δημοσίευση URI όμοια με δομή καταλόγου.
- Μεταφέρει XML, JavaScript Object Notation (JSON), ή και τα δύο.

Οι RESTful διαδικτυακές υπηρεσίες έχουν καθιερωθεί πλέον στο διαδίκτυο ως οι πλέον κατάλληλες για τη μεταφορά, την ανάκτηση και την εισαγωγή μεγάλου όγκου δεδομένων στο διαδίκτυο. Το REST, τυπικά είναι μια αρχιτεκτονική λογισμικού για κατανεμημένα συστήματα με την οποία μπορούν να σχεδιαστούν διαδικτυακές υπηρεσίες που εστιάζουν στους πόρους ενός συστήματος, συμπεριλαμβανομένου του τρόπου που αντιμετωπίζονται οι καταστάσεις των πόρων και μεταφέρονται μέσω HTTP σε ένα ευρύ φάσμα πελατών γραμμένα σε διαφορετικές γλώσσες. Την αρχιτεκτονική REST, την αποτελούν οι πελάτες και οι εξυπηρετητές. Οι πελάτες ξεκινούν αιτήματα προς τους εξυπηρετητές, οι οποίοι με τη σειρά τους, αφού τα επεξεργαστούν, επιστρέφουν στους πελάτες που τους έστειλαν το αίτημα τις ανάλογες απαντήσεις. Τα αιτήματα και οι απαντήσεις έχουν υλοποιηθεί γύρω από αναπαραστάσεις των πόρων (resources). Πόρος μπορεί να είναι οποιοδήποτε πληροφοριακό αντικείμενο με κάποια σημασία και η αναπαράστασή του αποτυπώνει ένα στιγμιότυπό του σε μια δεδομένη χρονική στιγμή ή μετά από κάποια ενέργεια που εκτέλεσε.

- Βάση Δεδομένων (DataBase)

Η βάση δεδομένων είναι το εργαλείο για τη συλλογή, αποθήκευση, διαχείριση και άρα αξιοποίηση δεδομένων. Στην αρχιτεκτονική αυτή είναι το σημείο αποθήκευσης και διαχείρισης των πληροφοριών των χρηστών σχετικά με την υπηρεσία π.χ. αριθμός κόμβων, ποιες οι IPs κ.λ.π.
Ένα ενδεικτικό schema της βάσης είναι το ακόλουθο:

Instances	Attributes	Hosts
Id	Id	Id
Name	Instance_id	Instance_id
Creation_date	Filter_params	Host
Status	Filter_values	Port
Volume_Type		Memory
Pid		Role

Στον παραπάνω πίνακα, φαίνονται οι πίνακες που χρειάζονται για τους χρήστες της πλατφόρμας.

Στον πρώτο πίνακα φαίνεται το όνομα, η ημερομηνία δημιουργίας, η κατάσταση του συστήματος καταγραφής δικτυακών δεδομένων ενός χρήστη και το είδος του συστήματος αποθήκευσης των δεδομένων στην πλατφόρμα (π.χ. NFS, Host FileSystem, Cinder Volumes κλπ). Τέλος, καταγράφονται επίσης τα id των processes που αφορούν των συγκεκριμένο χρήστη-instance.

Ο δεύτερος πίνακας περιλαμβάνει κάποιες παραμέτρους που πρέπει να οριστούν ώστε να φιλτραριστεί η κίνηση και να ανταποκρίνεται για τον καθένα χρήστη ξεχωριστά.
Τέλος, ο τρίτος πίνακας περιλαμβάνει τους κόμβους με τις αντίστοιχες πόρτες , τη μνήμη που χρειάζεται (ανάλογα με τη ροή της κίνησης) και τον ρόλο του καθενός (π.χ. αποθήκευση, οπτικοποίηση κ.λ.π). Οι κόμβοι αυτοί τρέχουν το εργαλείου που αποθηκεύει και διαθέτει γλώσσα για να ερωτά τα δεδομένα καθώς και το εργαλείο για την οπτικοποίηση τους.

- Πηγές (Sources)

Οι πηγές όπως έχει αναφερθεί και στις απαιτήσεις του συστήματος είναι το σημείο που

τροφοδοτεί την πλατφόρμα με κίνηση από το δίκτυο. Στη συγκεκριμένη περίπτωση μπορεί να είναι ένα ή περισσότεροι μεταγωγείς (switches) που συνδέουν διαφορετικές οντότητες με το διαδίκτυο. Μπορεί να μην είναι αυτό καθ' αυτό μέρος της πλατφόρμας, ωστόσο χρειάζεται ο ενδιάμεσος μηχανισμός που συνδέει τα switches με την πλατφόρμα καταγραφής. Αυτό το ενδιάμεσο στάδιο χρειάζεται για την απαραίτητη μορφοποίηση των δεδομένων σε JSON μορφή ούτως ώστε να είναι αναγνωρίσιμη και συμβατή με την πλατφόρμα.

- Ουρά - Brokers

Αυτό το στάδιο είναι ένα κατανεμημένο σύστημα διαχείρισης μηνυμάτων (Messaging System), που μπορεί να επεξεργάζεται πάρα πολύ μεγάλο όγκο δεδομένων, είτε αυτός προέρχεται από ροές (streams) είτε από δέσμες δεδομένων (batches) . Σε ένα τέτοιο κατανεμημένο σύστημα οι παραγωγοί (switches) στέλνουν μηνύματα στο cluster μέσω δικτύου, αυτά αποθηκεύονται σε ουρές των brokers και στη συνέχεια διατίθενται στους καταναλωτές που ενδιαφέρονται για αυτά.
Αυτό το στάδιο αποτελεί ουσιαστικά το αρχικό στάδιο της πλατφόρμας και οι λόγοι πίσω από τη χρήση αυτού του σταδίου είναι για να μπορεί να συλλέγει ασύγχρονα δεδομένα από όλες τις πηγές (switches), να τα παραλληλοποιεί μέσα στο cluster και έτσι να επιτρέπει την «κατανάλωση» των δεδομένων από πολλούς καταναλωτές.

- Προεπεξεργασία – Processing

Αυτό το στάδιο αποτελεί τον καταναλωτή ουσιαστικά της ουράς που περιγράφηκε παραπάνω. Τρέχοντας πολλά αντίγραφα αυτού του σταδίου μπορεί να επιτευχθεί η ταυτόχρονη κατανάλωση διαφορετικών τμημάτων της ουράς.

Επίσης, σε αυτό το στάδιο γίνεται και η διαχώριση της κίνησης που έρχεται από τις πηγές και περνάει μέσα από την ουρά στους διαφορετικούς χρήστες-ενοίκους. Για αυτό το λόγο, πέραν των απαιτήσεων που έχουν καθοριστεί για την προεπεξεργασία που χρειάζεται αυτό το σύστημα-πλατφόρμα καταγραφής, είναι αναγκαίο το στάδιο αυτό να φιλτράρει την κίνηση για τον κάθε ένοικο ξεχωριστά και να τη στέλνει στον αντίστοιχο προορισμό. Προκειμένου να πάρει την πληροφορία σχετικά με το φίλτρο/α που χρειάζεται είναι απαραίτητη η επικοινωνία με το κεντρικό API-backend που έχει την πληροφορία αυτή μέσω της βάσης. Αυτό δε χρειάζεται να γίνεται συχνά παρά μόνο στην αρχή όταν εγγράφεται κάποιος χρήστης ή ίσως όταν θελήσει να προσθέσει κάποια φίλτρα.

Ύστερα, επεξεργάζεται το κάθε πακέτο που λαμβάνει μέσω της ουράς και στη συνέχεια εφαρμόζει τα φίλτρα. Στο τέλος, επικοινωνεί με το εργαλείο αποθήκευσης των δεδομένων και στέλνει αίτημα προς αποθήκευση. Αυτό μπορεί να το κάνει ανά ομάδες τεμαχίων (batches) ούτως ώστε να το κάνει πιο αποδοτικά χρησιμοποιώντας λιγότερες συνδέσεις και άρα πόρους.

- Εργαλείο καταγραφής και αποθήκευσης - Monitoring Tool & Οπτικοποίηση - Visualization

Σε αυτό το στάδιο έρχεται ένα μέρος της τελικής υπηρεσίας που είναι υπεύθυνο για την αποθήκευση των δεδομένων και τη διερώτηση αυτών μαζί με ττο εργαλείο για την οπτικοποίησή τους. Προκειμένου όμως να υπάρχει αυτό το εργαλείο πρέπει με κάποιον τρόπο να του ανατεθούν οι πόροι και οι διαφορετικές πόρτες για τον κάθε ένοικο. Σε αυτό το κομμάτι παίζει σημαντικό ρόλο η επικοινωνία με το κεντρικό API-backend για να δημιουργήσει πρώτα και να διαμορφώσει τον/τους κόμβους που θα φιλοξενούν αυτό το εργαλείο. Επίσης, μέσω αυτής της επικοινωνίας μπορούν να γίνονται οποιεσδήποτε ενημερώσεις για την κλιμάκωση του εργαλείου ή αλλαγής της διαμόρφωσης των κόμβων.

Σε αυτό το σημείο, αξίζει να διευκρινιστεί ότι ο κάθε χρήστης θα έχει το δικό του ξεχωριστό monitoring εργαλείο και άρα θα υπάρχουν πολλά instances του ίδιου εργαλείου στο ίδιο μηχάνημα/κόμβο. Το processing στάδιο θα τροφοδοτεί το εργαλείο του κάθε χρήστη ξεχωριστά και ταυτόχρονα, αφότου πρώτα έχουν δημιουργηθεί και ρυθμιστεί όλα.

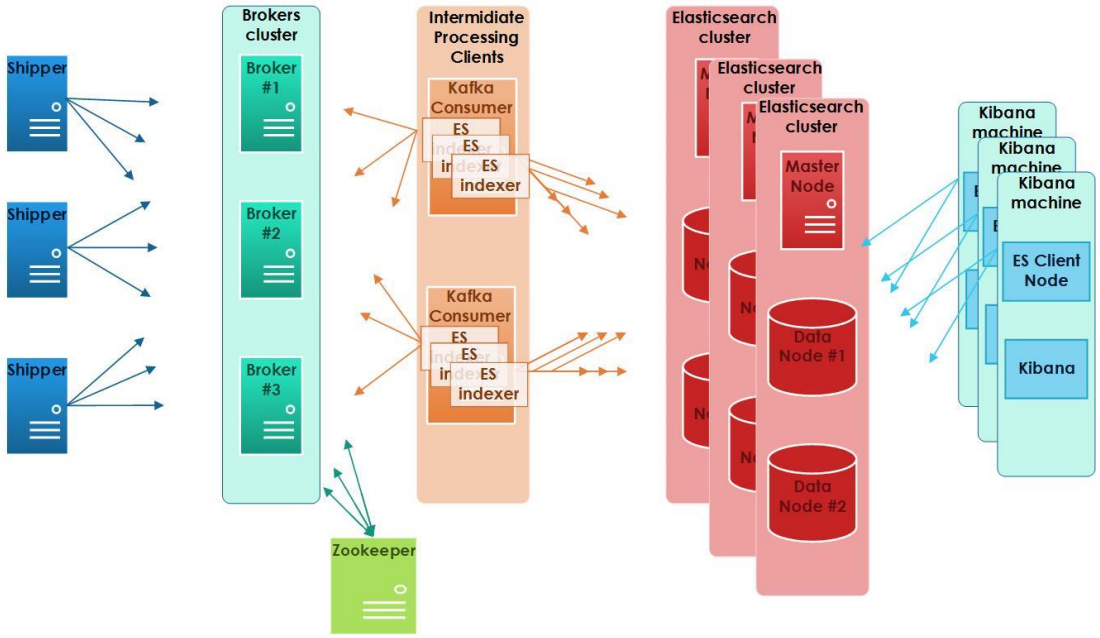
Τέλος, εδώ αρχίζει να φαίνεται και το χαρακτηριστικό της υπηρεσίας που έχει να κάνει με την κατ' απαίτηση εξυπηρέτηση των χρηστών. Συνοψίζοντας τα παραπάνω:

1. Ο κάθε χρήστης πιστοποιείται στην υπηρεσία και συμπληρώνει κάποια στοιχεία.
2. Δημιουργείται καταγραφή του χρήστη στη βάση.
3. Ο εξειδικευμένος χώρος του χρήστη δημιουργείται στην πλατφόρμα δημιουργώντας αρχικά τα instances του monitoring tool, ενημερώνοντας παράλληλα τη βάση.
4. Στο processing στάδιο ξεκινά η επεξεργασία, το φιλτράρισμα και η αποστολή των δεδομένων στα καινούρια instances του monitoring tool για τον συγκεκριμένο χρήστη.
5. Ο κάθε ένοικος έχει τον δικό του προσωποποιημένο χώρο για την καταγραφή της δικτυακής κίνησης που περνάει από κεντρικές δικτυακές υποδομές.

4.2 Ανάλυση υλοποίησης

Ύστερα από τον λεπτομερή σχεδιασμό των απαιτήσεων, των χαρακτηριστικών και της αρχιτεκτονικής του συστήματος-πλατφόρμας, σε αυτό το σημείο θα αναλυθεί ο τρόπος υλοποίησης και εφαρμογής του κάθε σταδίου που παρουσιάστηκε προηγουμένως καθώς και ο κώδικας που αναπτύχθηκε για αυτόν τον σκοπό.

Αρχικά, δίνεται παρακάτω το γενικό σχήμα της πλατφόρμας με τα στάδια όπως έχουν αναφερθεί και τα οποία θα αναλυθούν λεπτομερώς παρακάτω.



Σχήμα 4.2 Επισκόπηση αρχιτεκτονικής πλατφόρμας διπλωματικής

Στο πλαίσιο του κώδικα που χρησιμοποιήθηκε θα σχολιαστούν αρχικά οι κύριες δομές και βιβλιοθήκες που χρησιμοποιήθηκαν για την ανάπτυξη των διάφορων σταδίων της πλατφόρμας.

- Kafka-python

Αυτή η βιβλιοθήκη είναι η εφαρμογή του client του Kafka σε python και υποστηρίζει πλήρως το πρωτόκολλο που χρησιμοποιείται από το Kafka. Εφαρμόζει τους μηχανισμούς για την κατανάλωση (consume) αλλά και την παραγωγή (produce) μηνυμάτων συμπεριλαμβανομένου μερικών μηχανισμών συμπίεσης των μηνυμάτων. Λόγω της πλήρης υποστήριξης του πρωτοκόλλου του kafka δίνει τη δυνατότητα αλλαγής κάποιων παραμέτρων κατά τη διάρκεια κλήσης της εκάστοτε συνάρτησης του consumer ή producer.

- Python-elasticsearch

Αυτή είναι η επίσημη βιβλιοθήκη του elasticsearch που έχει έναν client υλοποιημένο σε χαμηλό επίπεδο. Όλα τα API αιτήματα αντιστοιχούν στα απλά REST API αιτήματα που γίνονται στο elasticsearch. Ο client περιλαμβάνει:

- Τη μετάφραση από python τύπους δεδομένων σε json και το αντίστροφο
- Ρυθμίσιμη και αυτόματη εύρεση των άλλων nodes του elasticsearch cluster
- Μόνιμες συνδέσεις με το cluster
- Εξισορρόπηση φορτίου σε όλους τους διαθέσιμους nodes του elasticsearch
- Οι συνδέσεις περιμένουν να λάβουν επιβεβαίωση κι αν δε λάβουν περιμένουν κάποιο χρονικό διάστημα μέχρι την επόμενη προσπάθεια
- Υπάρχει ασφάλεια μεταξύ κλήσεων των συναρτήσεων της βιβλιοθήκης από διαφορετικά threads (thread safety)
- Βοηθητικές βιβλιοθήκες για πιο αποδοτική χρήση των υποδομών και των λειτουργιών του elasticsearch cluster

- Maxmind-GeoIP2

Το maxmind [37] παρέχει ελεύθερα κάποια διαδικτυακές υπηρεσίες αλλά και κάποιες βάσεις σε δυαδική και csv μορφή που επιτρέπουν την ανακάλυψη πληροφοριών σχετικά με τη γεωγραφική θέση κάποιας IP (geolocation). Γενικά, η γεωγραφική θέση αυτή δεν είναι απόλυτα ακριβής καθώς η συλλογή των δεδομένων από κάποια τοποθεσία αντιστοιχεί σε ένα λεγόμενο IP block οπότε πολλές IPs φαίνονται να ανήκουν στην ίδια ή κοντινή τοποθεσία. Περιέχει πληροφορίες για τις περισσότερες δημόσιες IP και ανανεώνεται κάθε μήνα. Στο στάδιο της προεπεξεργασίας χρησιμοποιήθηκαν οι πληροφορίες που παρέχονται σχετικά μόνο με τη χώρα, την πόλη, το γεωγραφικό μήκος και πλάτος καθώς και το αυτόνομο σύστημα στο οποίο ανήκει η IP.

Επίσης, μαζί με τις βάσεις παρέχεται και η αντίστοιχη βιβλιοθήκη της python μαζί με κάποια C-extensions που βοηθούν στην καλύτερη απόδοση της βιβλιοθήκης κατά το διάβασμα και την αναζήτηση στη βάση. Η βιβλιοθήκη ουσιαστικά δίνει κάποιο API το οποίο από πίσω παρέχει τους απαραίτητους μηχανισμούς για την αναζήτηση μια συγκεκριμένης IP στη βάση. Ύστερα επιστρέφεται μια δομή το οποίο ουσιαστικά είναι από πίσω σε μορφή json.

- Cjson

Πρόκειται για βιβλιοθήκη που μπορεί να σειριοποιεί και να αποσειριοποιεί δεδομένα σε json μορφή αλλά είναι γραμμένο σε γλώσσα C και είναι 10-250 φορές πιο γρήγορο από την κλασική βιβλιοθήκη json της python και εξαρτάται κυρίως από το πόσο πολύπλοκο είναι το αρχείο προς σειριοποίηση.

- Pathos – Multiprocess

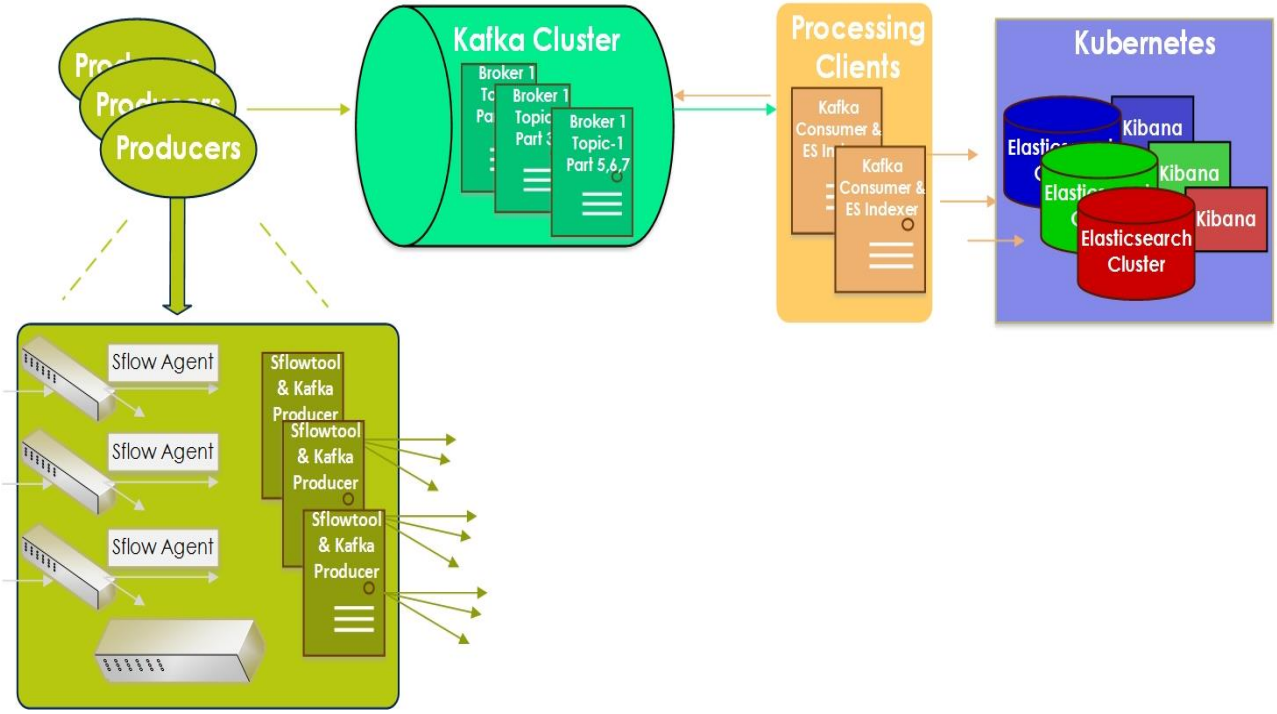
Το Pathos είναι μια δομή που παρέχει μια υψηλού επιπέδου διεπαφή για ρύθμιση και εκτέλεση παράλληλων υπολογισμών στις υποδομές. Η βιβλιοθήκη multiprocess αποτελεί μέρος του pathos και είναι μια πλήρη διακλάδωση (fork) της κλασικής βιβλιοθήκης της python multiprocessing. Για την ενεργοποίηση των πυρήνων σε σύγχρονους επεξεργαστές χρειάζεται η επικοινωνία μεταξύ διαφορετικών processes. Για την επικοινωνία των διάφορων δομών (π.χ. συναρτήσεων, κλάσεων, αντικειμένων κ.λ.π.) μεταξύ των processes χρειάζεται η σειριοποίηση σε κάποια μορφή κειμένου η οποία είναι πιο διαχειρίσιμη σε σχέση με άλλες πολύπλοκες δομές. Το pathos χρησιμοποιεί τη βιβλιοθήκη dill η οποία έχει καλύτερη απόδοση και μπορεί να σειριοποιήσει περισσότερες δομές σε σχέση με την κλασική pickle βιβλιοθήκη που χρησιμοποιείται στη multiprocessing.

- Jinja2

Η Jinja2 είναι μια βιβλιοθήκη της python που περιέχει μια μηχανή προτύπων (template engine). Έχει επίσης τη δική της γλώσσα για την συμπλήρωση των προτύπων. Χρησιμοποιήθηκε για τη συμπλήρωση αρχείων που περιέχει ρυθμίσεις και περιγραφές οντοτήτων όπου πολλές από τις τιμές εξαρτώνται από κάποια μεταβλητή.

Η παρακάτω ανάλυση θα ασχοληθεί κυρίως με τη σειρά (pipeline) των διαδικασιών της πλατφόρμας καθώς και το ενδιάμεσο στάδιο πριν την τροφοδότησή της από τις πηγές. Στη συνέχεια, θα αναλυθεί ο τρόπος υλοποίησης του κεντρικού εκείνου στοιχείου που επικοινωνεί και ρυθμίζει όλη την πλατφόρμα, το API-backend.

4.2.1 *Πηγές (Sources) – Αποστολείς (Shippers) – Παραγωγοί (Producers)*



Σχήμα 4.3 Λεπτομέρειες αρχιτεκτονικής του σταδίου των πηγών της πλατφόρμας

Από το σχήμα φαίνονται οι πήγες όπως έχουν αναφερθεί σαν switches μιας δικτυακής υποδομής. Από κει ξεκινάνε 2 βέλη. Το ένα ακολουθεί την κανονική ροή του πακέτου προς έναν κεντρικό switch ή router (δρομολογητή) απ' όπου διαμοιράζεται σε διάφορες οντότητες.

Το άλλο βέλος πηγαίνει προς τον λεγόμενο sflow agent που δέχεται τη δειγματοληπτημένη κίνηση. Αυτό είναι το ενδιάμεσο στάδιο όπως είχε αναφερθεί που αποτελεί τον sflow agent για έναν ή περισσότερα switches και μορφοποιεί το δείγμα πριν το στείλει προς την ουρά.

Στην υλοποίηση αυτού χρησιμοποιήθηκαν τα εξής εργαλεία :

- Tcpreplay [38], για την αναπαραγωγή αρχείου κίνησης pcap

Το εργαλείο αυτό μπορεί να προσομοιώνει την κίνηση του δικτύου που πηγαίνει στους μεταγωγείς. Η κίνηση μπορεί να έχει συλλεχθεί και αποθηκευθεί σε ένα αρχείο το οποίο μετά μπορεί να αναπαραχθεί με συγκεκριμένη ταχύτητα και να στέλνεται σε μια πόρτα κάποιας διεπαφής.

- Ovswitch [39], εικονικός μεταγωγέας

Το εργαλείο αυτό δημιουργεί ένα logical bridge στο οποίο προστίθεται η κατάλληλη διεπαφή. Στο bridge ρυθμίζεται ο sFlow agent για την αποστολή δειγμάτων στο SflowTool-KafkaProducer.

- SflowTool & Kafka Producer

Πρόκειται για το στάδιο που επιτρέπει την επικοινωνία και την τροφοδότηση των πηγών με το broker cluster το οποίο αποτελεί μια ουρά.

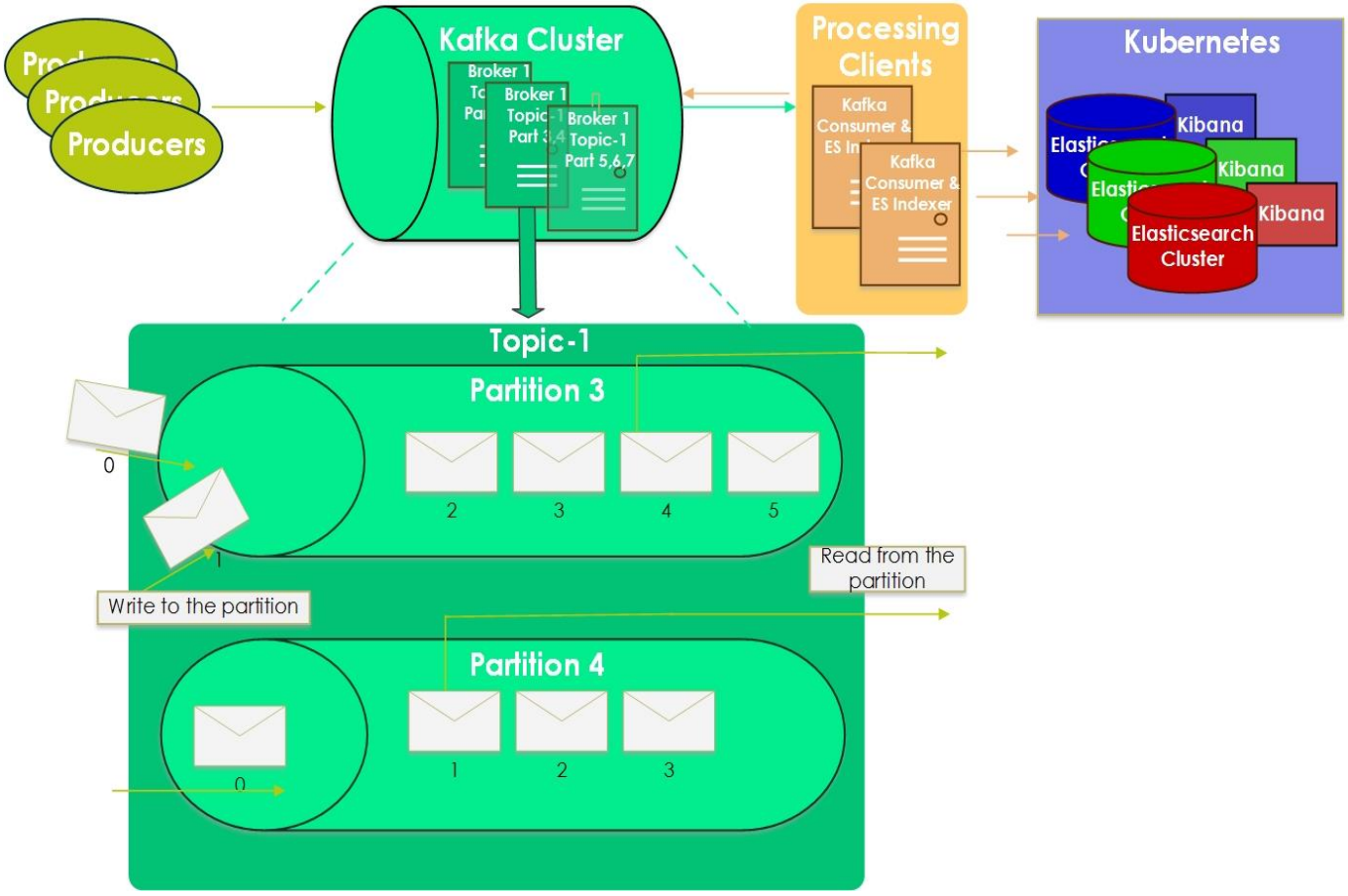
❖ SflowTool

Το SflowTool [40] είναι ένα εργαλείο αποκωδικοποίησης sFlow κίνησης για packet tracing και packet inspection. Συγκεκριμένα το sflowtool παράγει μια δομή του Sflow που περιέχει όλα τα στοιχεία χωρισμένα με κόμμα.

❖ KafkaProducer

Το κομμάτι του κώδικα που σειριοποιεί, μπορεί να συμπιέσει και να στείλει ασύγχρονα σε τεμάχια τα flows στην ουρά του Kafka. Ο producer περιλαμβάνει ένα pool από buffer space που κρατά τα μηνύματα που δεν έχουν σταλεί ακόμα στους brokers. Επίσης, περιλαμβάνει ένα thread I/O που είναι υπεύθυνο για τη μετατροπή αυτών των μηνυμάτων σε αιτήματα για να τα στείλει στο brokers cluster. Η κύρια λειτουργία του είναι να συσχετίζει κάθε μήνυμα-Sflow με το partition ενός topic και να στείλει ένα αίτημα στον master-leader του partition. Το partition βρίσκεται εσωτερικά μέσω ενός hash fuction. Κάθε μήνυμα είναι διαθέσιμο αφότου αντιγραφεί από τον leader του partition και άρα να γίνει ”commit”. Σε περίπτωση που το μήνυμα αποτύχει να σταλεί τότε έχει ρυθμιστεί ο producer να ξαναπροσπαθεί να στείλει το ίφδιο μήνυμα.

4.2.2 *Broker cluster*



Σχήμα 4.4 Λεπτομέρειες αρχιτεκτονικής του σταδίου της ουράς της πλατφόρμας

Όλα τα χαρακτηριστικά του broker που επιλέχθηκε (Kafka) είναι σημαντικά όπως έχει ήδη αναφερθεί στις απαιτήσεις του συστήματος αλλά και στο θεωρητικό κομμάτι που αναλύονται τα βασικά χαρακτηριστικά του Kafka. Είναι ένα βασικό στοιχείο της πλατφόρμας και αναλύοντας περεταίρω παρακάτω το πώς λειτουργεί και πώς έχει διαμορφωθεί για τις ανάγκες του use case γίνεται φανερό το ότι είναι ένα ζωτικό στοιχείο του συστήματος καθώς αποτελεί το πρώτο κομμάτι που τροφοδοτεί την πλατφόρμα. Αυτό θα είναι και το κομμάτι που υποστηρίζει την πλατφόρμα σε περίπτωση κάποιας αποτυχίας κάποιου μέρους της πλατφόρμας, ακόμα και του ίδιου καθώς επιτρέπει την αρχική και πιο βασική κλιμακωσιμότητα του συστήματος.

Αυτό το στάδιο δείχνει περισσότερο γραφικά το πώς συνδέεται το στάδιο με τις πήγες με αυτό της προεπεξεργασίας. Όπως φαίνεται, λοιπόν, το προηγούμενο κομμάτι του producer στέλνει τα μηνύματα ακολουθώντας κάποια πολιτική στο cluster των brokers (kafka).

Επιλέγοντας το node όπου βρίσκεται ο leader του partition, το μήνυμα στέλνεται στο τέλος της ουράς και περιμένει να καταναλωθεί από το επόμενο στάδιο που βρίσκεται ο consumer.

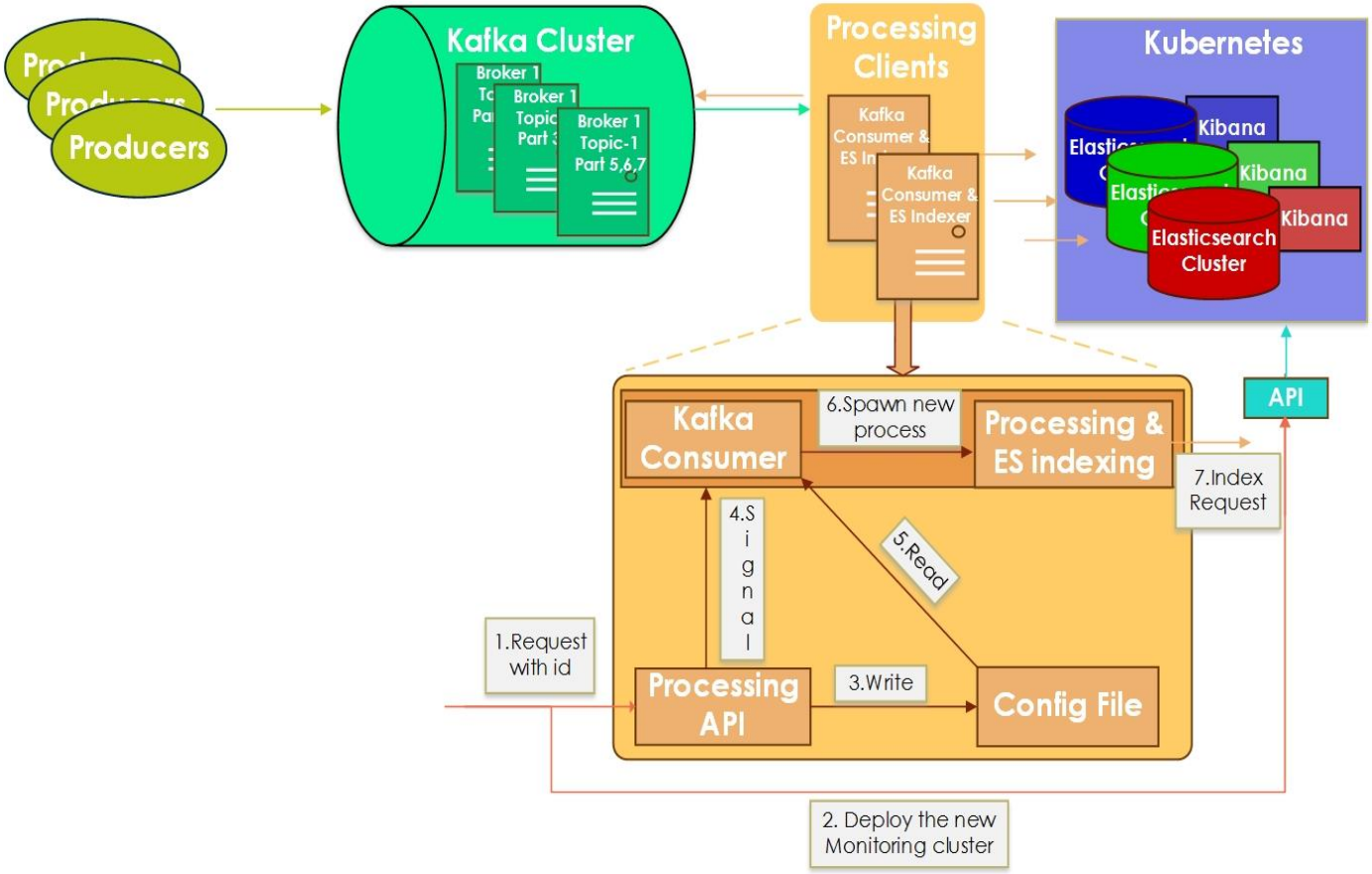
Προκειμένου το σύστημα των brokers να δέχεται περισσότερα requests δύο είναι οι κύριες ρυθμίσεις που θα βοηθήσουν προς αυτήν την κατεύθυνση. Το μέγεθος του τεμαχίου που θα σταλεί (batch size) και ο χρόνος που θα περιμένει ο producer για να καταφθάσουν παραπάνω μηνύματα (linger.ms). Μεγάλες τιμές και στις δύο ρυθμίσεις οδηγούν σε μεγαλύτερο χώρο στη μνήμη.

Σε κάθε μήνυμα όταν στέλνεται ακολουθείται μια πολιτική για το πότε θεωρείται πετυχημένη η αποστολή του. Για να θεωρηθεί ένα αίτημα επιτυχές ο leader του partition πρέπει να στείλει επιβεβαίωση ότι το μήνυμα έχει γραφτεί έστω στο τοπικό log του node. Ύστερα το μήνυμα γράφεται στο δίσκο και αντιγράφεται σε άλλο node του cluster. Σε περίπτωση που ο node κάποιου leader partition σταματήσει να λειτουργεί, τότε επιλέγεται ο επόμενος leader από τους άλλους nodes που έχουν αντιγραμμένα τα partitions από τον leader και είναι in-sync. Με αυτόν τον τρόπο επιτυγχάνεται το συγκεκριμένο κομμάτι να είναι πάντα διαθέσιμο (highly available).
Στη συγκεκριμένη περίπτωση το cluster δεν συμπιέζει τα μηνύματα ούτως ώστε να μη χάνεται χρόνος και υπολογιστική ισχύ κατά την συμπίεση και αποσυμπίεσή τους καθώς επίσης θα πρέπει να επαναλαμβάνεται η ίδια διαδικασία σε κάθε στάδιο της πλατφόρμας.

Τα μηνύματα εσωτερικά της ουράς έχουν ρυθμιστεί να τα αποθηκεύουν στην ουρά για 7 μέρες. Μετά από αυτό το διάστημα διαγράφονται όλα τα μηνύματα από μια ημερομηνία και πριν.

Φτάνοντας τώρα για το πώς λειτουργεί το συγκεκριμένο κομμάτι για την κατανάλωση των μηνυμάτων που βρίσκονται μέσα στο Kafka αξίζει να αναφερθεί ότι χρησιμοποιείται ένα πρωτόκολλο που συντονίζει τα διαφορετικά groups των consumers (τα groups έχουν αναλυθεί στο θεωρητικό κομμάτι). Ο συντονιστής αυτός που είναι κάποιος από τους nodes του Kafka και διαχειρίζεται την κατάσταση του κάθε group. Κύρια εργασία του είναι να μεσολαβεί για την ανάθεση των partitions όταν νέοι consumers μπαίνουν στο group ή όταν παλιοί φεύγουν ή όταν αλλάζουν τα metadata κάποιου topic. Αυτή η εργασία ονομάζεται εξισορρόπηση του group. Έτσι πάλι, επιτυγχάνεται η εύκολη και άκοπη κλιμακωσιμότητα του επόμενου επιπέδου καθώς απλά αρκεί να υπάρχουν πάνω από ένα διαθέσιμα instances.

4.2.3 *Προεπεξεργασία – Εμπλουτισμός*



Σχήμα 4.5 Λεπτομέρειες αρχιτεκτονικής του σταδίου επεξεργασίας της πλατφόρμας

Αυτό εδώ είναι το κεντρικό στάδιο της πλατφόρμας που πέραν της προεπεξεργασίας αποτελεί και το ενδιάμεσο στάδιο που επιτρέπει την επικοινωνία της ουράς Kafka με το σύστημα καταγραφής.

Θα αναλυθεί το κάθε κομμάτι αυτού του σταδίου ξεχωριστά και θα γίνει φανερό ποιο σκοπό εξυπηρετεί το κάθε ένα. Πριν από αυτό όμως θα αναλυθούν οι λεπτομέρειες και ο τρόπος λειτουργίας του κεντρικού Backend API που σχετίζεται με αυτό το στάδιο και απεικονίζεται κιόλας στην πάνω εικόνα.

- Backend API

Το τελικό σημείο (endpoint) που υλοποιήθηκε και χρησιμοποιήθηκε για τις ανάγκες της πλατφόρμας καταγραφής διαδικτυακής κίνησης είναι αυτό που διαχειρίζεται και συγκεκριμένα δημιουργεί, διαγράφει και παίρνει πληροφορίες από το εργαλείο καταγραφής που χρησιμοποιείται, το Elasticsearch.

Μεγάλο μέρος σε αυτό το LBL κομμάτι είναι και η διαχείριση του συστήματος αποθήκευσης. Συγκεκριμένα, υποστηρίζονται διαφορετικοί τύποι συστημάτων αποθήκευσης όπως cinder volumes, NFS volumes και το filesystem.

Τέλος, κατά τη διάρκεια ανάπτυξης του συγκεκριμένου API προκειμένου να υπάρχει μια περισσότερη ευελιξία σε αλλαγές δεν αναπτύχθηκε το endpoint που παίρνει το request από τον client και το στέλνει στο API που συνυπάρχει με το στάδιο της προεπεξεργασίας. Τον ρόλο αυτό έπαιξε απλά ένα αίτημα σε python ή σε bash με την εντολή *curl*.

- Processing API

Σε αυτό το σημείο αξίζει να σημειωθεί ξανά πως το στάδιο της προεπεξεργασίας χρειάζεται να επικοινωνεί με το BackendAPI προκειμένου να παίρνει πληροφορίες για τον χρήστη και να δρα αναλόγως, ανανεώνοντας ή σβήνοντας ή δημιουργώντας νέα συστήματα καταγραφής. Συγκεκριμένα, χρειάζεται το όνομα του συστήματος καταγραφής και το πεδίο sFlow που ουσιαστικά διαχωρίζει τον χρήστη από τους άλλους (dstvlan ή srcvlan, dstPort ή srcPort κ.λ.π.).

Έχει υλοποιηθεί ένα endpoint που έχει να κάνει με την επεξεργασία και εγκυρότητα του αιτήματος που έχει ληφθεί από τον χρήστη. Σε περίπτωση έγκυρου αιτήματος η μέθοδος στέλνει το απαραίτητο αίτημα πίσω στο BackendAPI για τη διαχείριση του συστήματος καταγραφής. Τέλος, στέλνει το κατάλληλο σήμα, SIGHUP, στο Kafka2ES, το πρόγραμμα πρόγραμμα προεπεξεργασίας που επίσης, διαχειρίζεται τα μηνύματα και τα στέλνει στα διαφορετικά συστήματα καταγραφής για τους διαφορετικούς χρήστες. Πριν από αυτήν τη διαδικασία όμως πρέπει να γράψει πρώτα στο Config File ότι χρειάζεται το Kafka2ES για να τα διαβάσει και ύστερα να εφαρμόσει τα κατάλληλα φίλτρα που διαχωρίζουν τους χρήστες μεταξύ τους.

- Config File

Προκειμένου όλα τα διαφορετικά στάδια και τμήματα της πλατφόρμας να έχουν πρόσβαση στις πληροφορίες του κάθε χρήστη ώστε να μπορούν να επικοινωνούν μεταξύ τους ή και να διαχειρίζονται τα διαφορετικά συστήματα καταγραφής, χρησιμοποιήθηκε αντί για βάση ένα αρχείο ρυθμίσεων. Τα πεδία που χρειάζονται είναι τα ίδια με αυτά όπως αναφέρθηκαν και εξηγήθηκαν στην *Ανάλυση της Αρχιτεκτονικής*.

- Kafka2ES

Αυτό είναι το κύριο πρόγραμμα που διαβάζει από το Config File όσα γράφτηκαν από το ProcAPI. Αυτό είναι ουσιαστικά το πρόγραμμα που αντιπροσωπεύει αυτό το στάδιο της πλατφόρμας που είναι η σύνδεση της ουράς Kafka με το σύστημα καταγραφής Elasticsearch.

Σε αυτό το σημείο αξίζει να αναφερθεί επίσης ότι κάθε σύνδεση στην ουρά του Kafka που αφορά διαφορετικό group consumer είναι σε διαφορετικό pathos process, που αποτελεί ένα κανονικό python process μόνο που ανήκει στη δομή pathos που αποτελεί μια διακλάδωση της κλασικής *multiprocessing* βιβλιοθήκης και χρησιμοποιεί διαφορετικό τρόπο σειριοποίησης. Κάθε διαφορετικό process είναι εν δυνάμει και ένας διαφορετικός χρήστης και άρα μια διαφορετική σύνδεση στο σύστημα καταγραφής όπου αποθηκεύονται τα sFlows που αφορούν τον χρήστη.

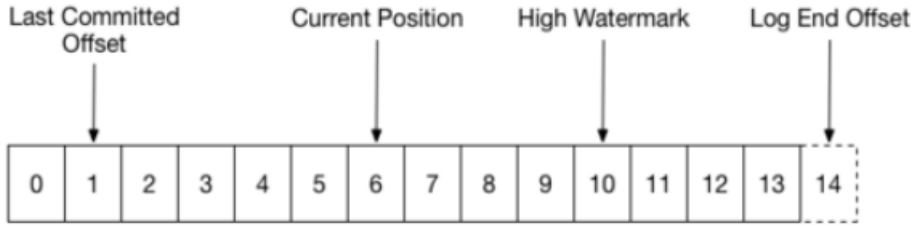
Κάθε αίτημα για νέο σύστημα καταγραφής που υπάρχει ήδη και άρα υπάρχει ήδη σύνδεση προς την ουρά, έχει το ίδιο *group id* και χρησιμοποιεί την ίδια σύνδεση αντί να δημιουργεί καινούρια ούτως ώστε να εξοικονομούνται πόροι. Αυτά τα processes που έχουν την ίδια σύνδεση προς τον Kafka αναλαμβάνουν να καταναλώνουν διαφορετικά partitions μεταξύ τους που οδηγεί σε πιο γρήγορη κατανάλωση μηνυμάτων. Για τη σύνδεση και αποστολή αιτήματος για αποθήκευση στο Elasticsearch χρησιμοποιείται νέα σύνδεση από το process, ανεξαρτήτως από το αν χρησιμοποιείται η ίδια σύνδεση για το Kafka.

Παρακάτω περιγράφονται αυτά τα 2 κύρια στοιχεία του προγράμματος που έχουν να κάνουν με τη σύνδεση με το Kafka και με το Elasticsearch μαζί με αυτό της προεπεξεργασίας.

- KafkaConsumer

Αυτό αποτελεί τον consumer της ουράς Kafka και άρα τη σύνδεση με το Kafka. Σε αυτό το σημείο, θα αναλυθεί με περισσότερες λεπτομέρειες το πώς λειτουργεί ο consumer στην Python που χρησιμοποιήθηκε, περισσότερο από την πλευρά του παρά από την πλευρά των nodes του Kafka.

Όταν το group από consumers αρχικοποιείται αρχίζει η κατανάλωση είτε των πιο πρόσφατων είτε του πρώτου λεγόμενου offset σε κάθε partition. Τα μηνύματα ύστερα σε κάθε partition log διαβάζονται κατά σειρά. Για παράδειγμα, στο από κάτω σχήμα η θέση του consumer είναι στο offset 6 και έγινε τελευταία commit όταν ήταν στο offset 1.



Όταν ένα partition ξανά ανατίθεται σε άλλον consumer στο ίδιο group, η αρχική θέση όπου θα αρχίσει να διαβάζει θα τεθεί στο offset που έγινε τελευταίο commit. Αν ο consumer στο παραπάνω παράδειγμα ξαφνικά σταματήσει να λειτουργεί, τότε ο νέος consumer που θα αναλάβει να καταναλώνει από το συγκεκριμένο partition θα ξεκινήσει από το offset 1. Σε αυτήν την περίπτωση, θα χρειαστεί να ξανά περάσει από τα μηνύματα μέχρι το offset 6.

Το διάγραμμα, επίσης, δείχνει 2 θέσεις του log. Το offset του τέλους του log είναι η θέση του τελευταίου μηνύματος που γράφτηκε στο log. Η υψηλή στάθμη (high watermark) είναι η θέση του τελευταίου μηνύματος που αντιγράφηκε σε άλλα logs στους άλλους nodes του Kafka. Από την πλευρά του consumer αυτή η θέση έχει σημασία γιατί μπορεί να διαβάσει μέχρι το high watermark. Αυτό εμποδίζει τον consumer να διαβάζει μηνύματα που δεν έχουν αντιγραφεί και που έχουν κίνδυνο ίσως να χαθούν.

- Processing

Αυτό είναι το κομμάτι της προεπεξεργασίας που απλά καθορίζει τα πεδία του sFlow και ύστερα χρησιμοποιεί τα φίλτρα που παρέχονται από τον χρήστη και γράφονται στο Config File για να τα εφαρμόσει πάνω στα εκάστοτε sFlows που λαμβάνει μέσω του

KafkaConsumer. Εφόσον, ταιριάξει το φίλτρο με το σχετικό πεδίο του sFlow τότε χρησιμοποιούνται κάποιες συναρτήσεις και έξτρα βιβλιοθήκες ώστε να εμπλουτιστεί το flow με παραπάνω πληροφορίες που δίνουν μεγαλύτερη αξία στους τελικούς χρήστες της πλατφόρμας.

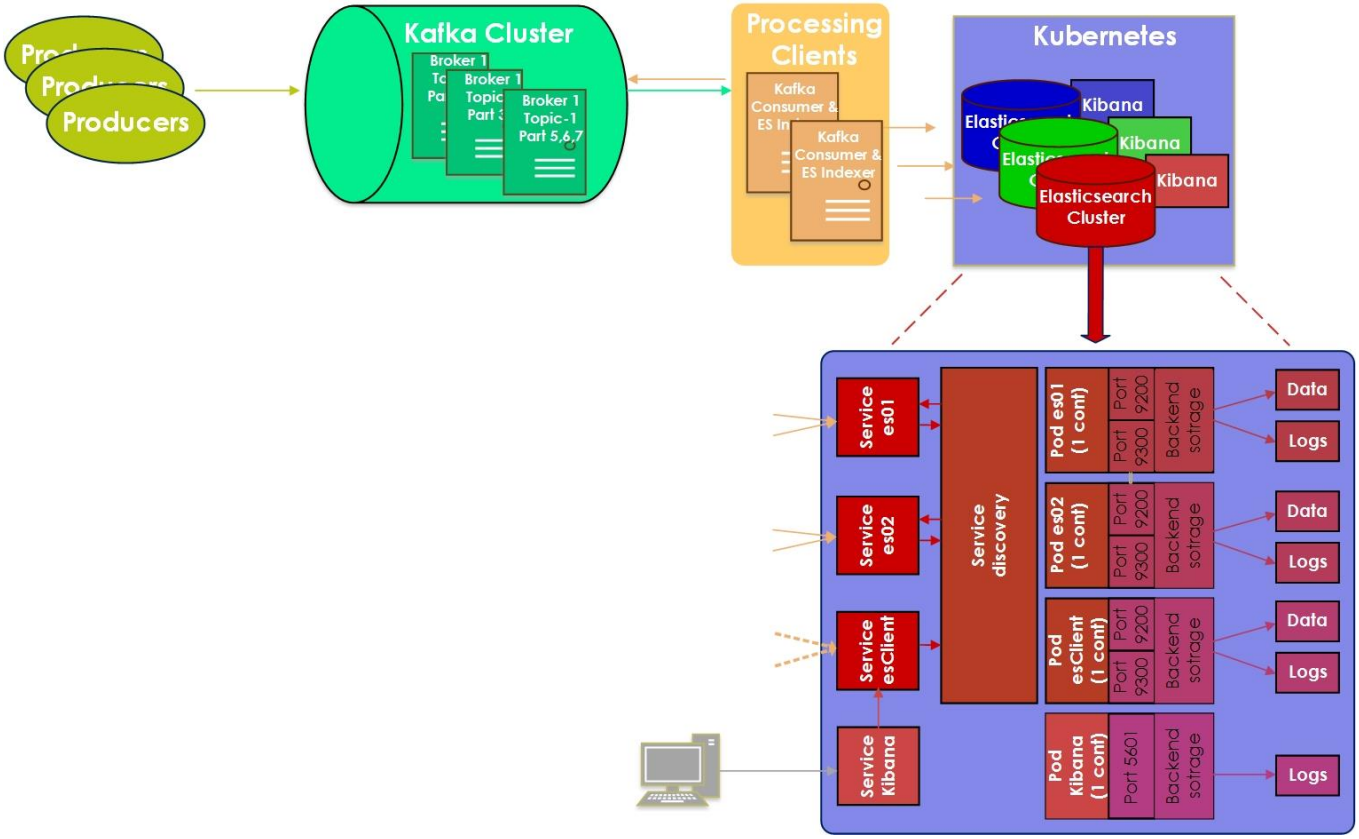
Εδώ επαναλαμβάνεται το είδος της προεπεξεργασίας που γίνεται. Αρχικά, γίνεται μια αναζήτηση με τη βοήθεια της βιβλιοθήκης *cjson* μέσα σε ένα json αρχείο με ~400.000 γραμμές που υπάρχουν ζευγάρια από IPs και γίνεται αντιμετάθεση της IP πηγής ή/και προορισμού του sFlow με αυτήν του ζευγαριού από το αρχείο. Στόχος αυτής της αντιμετάθεσης είναι η μετατροπή μιας ιδιωτικής IP σε δημόσια σε περιπτώσεις εικονοποίησης του χώρου διευθύνσεων όπου κάθε χρήστης-ένοικος χρησιμοποιεί όλο το address space χωρίς περιορισμούς. Ακόμη, ύστερα από την μετατροπή σε δημόσια IP γίνεται μια αναζήτηση σε μια *maxmind* βάση-αρχείο με δυαδική μορφή που μαζί με τη βοήθεια των *maxmind* βιβλιοθηκών αντιμετατίθενται γρήγορα οι IPv4 και IPv6 διευθύνσεις σε αρχεία δεδομένων χρησιμοποιώντας δυαδική δομή. Συγκεκριμένα, στο πρόγραμμα γίνονται αντιμεταθέσεις των IPs με τη χώρα και την πόλη καθώς και με το αυτόνομο σύστημα (AS) στο οποίο ανήκει η IP.

❖ Elasticsearch Indexing

Το κομμάτι αυτό περιγράφει τη διαδικασία κατά τη σύνδεση και αποθήκευση στο elasticsearch. Για την πιο αποδοτική σύνδεση στο cluster του elasticsearch τα αιτήματα στέλνονται στον client node του elasticsearch, που παίζει τον ρόλο του load balancer, καθώς και σε έναν node ακόμα για high availability. Επίσης, στέλνονται μεγάλα τεμάχια (batches) από flows παρά ένα-ένα ξεχωριστά. Προκειμένου να βελτιστοποιηθεί και η ταχύτητα αλλά και η καθυστέρηση κατά την απάντηση από το αίτημα για αποθήκευση στο elasticsearch κατά την σύνδεση χρησιμοποιούνται διαφορετικά threads τα οποία παίρνουν μέρος του batch και το στέλνουν στο cluster.

Τέλος, για τη σύνδεση και την αποστολή αιτημάτων στο elasticsearch σε batches χρησιμοποιήθηκε η python βιβλιοθήκη του elasticsearch και συγκεκριμένα η βοηθητική στην οποία έχουν υλοποιηθεί οι μηχανισμοί για αποστολή bulks παράλληλα. Κάθε node του elasticsearch έχει στη διάθεσή του κάποια thread pools ώστε να βελτιώσει τη χρήση και τη διαχείριση της μνήμης. Αυτός ο μηχανισμός εκμεταλλεύεται το ThreadPool που έχει ρυθμιστεί στο cluster και σε συνδυασμό με τα threads που δημιουργεί το πρόγραμμα προσπαθεί να αυξήσει τη ροή των flows προς αποθήκευση στο elasticsearch. Τα thread pools έχουν επίσης και ουρές που τους αντιστοιχούν και επιτρέπουν σε διάφορα συμβάντα, που είναι αιτήματα προς αποθήκευση στη συγκεκριμένη περίπτωση, να μπορούν να κρατιούνται μέχρι να έρθει η σειρά τους αντί να απορρίπτονται. Για τα αιτήματα προς αποθήκευση χρησιμοποιούνται 10000 events που σπάνε σε 5000 για καθένα από τα 4 threads και σταθερό αριθμό thread pools που είναι όσοι και οι πυρήνες του μηχανήματος με μέγεθος ουράς 200.

4.2.4 *Εργαλείο καταγραφής & Ενορχήστρωση*



Σχήμα 4.6 Λεπτομέρειες αρχιτεκτονικής του σταδίου ενορχήστρωσης της πλατφόρμας

Αυτό είναι το τελικό στάδιο που έχει να κάνει με τη δημιουργία και την πρόσβαση στα monitoring clusters τα οποία αποτελούνται από δύο elasticsearch data nodes που είναι και εν δυνάμει master, έναν elasticsearch client node και έναν kibana node. Η δημιουργία και η διαχείρισή τους γίνεται από το Backend API. Το kubernetes επιλέχθηκε προκειμένου να γίνεται εύκολη ενορχήστρωση του cluster, διαχείριση του συστήματος αποθήκευσης, να υπάρχει μια απομόνωση του ενός cluster με το άλλο καθώς επίσης να γίνεται εύκολα και γρήγορα η εκκίνηση και η παύση λειτουργίας του cluster.

Αρχικά, χρειάστηκε να φτιαχτούν τα απαραίτητα docker images και να ανεβούν στο Docker Hub ώστε να μπορεί το kubernetes να τα βρίσκει και να κατεβάζει τις διάφορες εκδόσεις του image. Τα images είναι βασισμένα στα επίσημα images μόνο που έχουν προστεθεί και αλλάξει κάποια πράγματα για τις ανάγκες της διπλωματικής. Μία από τις βασικότερες

αλλαγές είναι η αλλαγή του κύριου image στο οποίο βασίζεται ώστε να μειωθεί κατά 80% το μέγεθος του image και να μπορεί να εκκινήσει πιο γρήγορα καθώς αυτό φορτώνεται στη μνήμη όταν ξεκινάει τη λειτουργία του το container και άρα η μείωση αυτή βελτίωσε δραματικά τη διαχειρισιμότητα του container κυρίως κατά την εκκίνηση. Επίσης, προστέθηκαν και ενσωματώθηκαν κάποιες μεταβλητές περιβάλλοντος (environment variables) ώστε να προδίδουν τον ρόλο του elasticsearch node (master, data, ingest).

Τώρα θα αναλυθεί ο τρόπος που τα διαφορετικά container μέσα στα pods του kubernetes είναι δομημένα ώστε να δίνουν πρόσβαση στον χρήστη προς το kibana αλλά και το REST API του elasticsearch. Αρχικά, αξίζει να σημειωθεί ότι το elasticsearch επικοινωνεί με τους άλλους κόμβους σε συγκεκριμένη πόρτα με ένα εσωτερικό πρωτόκολλο μεταφοράς (transport) που είναι εντελώς ασύγχρονο, που σημαίνει ότι δεν υπάρχει κάποιο thread να μπλοκάρει περιμένοντας για κάποια απάντηση. Επίσης, ο μηχανισμός που υπάρχει ώστε δημιουργεί τα cluster στο elasticsearch χρησιμοποιεί αυτό το πρωτόκολλο και λέγεται *unicast discovery*. Ο μηχανισμός αυτός απαιτεί μια λίστα με κάποιους nodes τους οποίους χρησιμοποιεί σαν ενδιάμεσους για να μαθαίνει πληροφορίες για τους υπόλοιπους κόμβους στην πόρτα που ακούει το πρωτόκολλο μεταφοράς. Τέλος, για να γίνει τελικά ένας κόμβος του elasticsearch μέλος του cluster κατά το unicast discovery ανταλλάσσονται πληροφορίες όπως το όνομα του cluster. Αν επικοινωνία είναι επιτυχής και έχουν το ίδιο όνομα cluster τότε ο κόμβος γίνεται μέλος του cluster.

Με αυτά σαν δεδομένα και το ότι υπάρχει μια ακόμα πόρτα στην οποία ακούει το REST API του elasticsearch, που χρησιμοποιείται για την αποστολή αιτημάτων, διαπιστώνεται ότι θα χρειαστούν δύο kubernetes services, το ένα που θα είναι υπεύθυνο για την εσωτερική επικοινωνία του elasticsearch και ένα για πρόσβαση των χρηστών στο REST API. Για την εσωτερική επικοινωνία των nodes χρησιμοποιήθηκε και ενσωματώθηκε μέσα στο docker image ένα plugin, το *elasticsearch-cloud-kubernetes,* που βοηθάει στον μηχανισμό για την εύρεση των άλλων nodes (unicast discovery). Επίσης, χρησιμοποιήθηκαν ως διαχειριστές (controllers) τα λεγόμενα *deployments,* προκειμένου να μπορούν να αλλάζουν οι ρυθμίσεις των pods πιο εύκολα, να μπορούν να προστίθενται κάποιοι αυτόματοι μηχανισμοί για την κλιμάκωση των κόμβων και για να επιβλέπει ότι θα είναι διαθέσιμος πάντα ο αριθμός των κόμβων που έχει δηλωθεί στο αρχείο με την περιγραφή του controller, pod και container. Το αρχείο με τις περιγραφές αυτές αποτελεί ουσιαστικά και τις ρυθμίσεις του κάθε elasticsearch ή kibana node. Για τον λόγο αυτό χρησιμοποιούνται Jinja2 templates από το Backend API που συμπληρώνει διάφορα στοιχεία από το είδος του συστήματος αποθήκευσης μέχρι τις μεταβλητές περιβάλλοντος που πιθανόν να χρειάζονται. Με αυτόν τον τρόπο η δημιουργία

pods στο kubernetes είναι πιο δυναμική και δεν εξαρτάται άμεσα από το είδος της εφαρμογής που θα τρέξει.

Συνοψίζοντας, το kibana node μιλάει με το service του client node που είναι υπεύθυνο για το REST API του elasticsearch και ταυτόχρονα το service του δημοσιοποιεί την πόρτα στην οποία είναι διαθέσιμα τα εργαλεία του. Ο client node του elasticsearch επικοινωνεί με τους υπόλοιπους nodes του cluster μέσω του service που έχουν όλοι για την επικοινωνία μεταξύ τους. Ταυτόχρονα, το REST API του κάθε node είναι διαθέσιμο για τους χρήστες. Από πίσω αυτό που γίνεται είναι ότι υπάρχουν τα pods τα οποία διαχειρίζονται το αντίστοιχο container, τις πόρτες που δημοσιοποιούνται προς χρήστες και στους άλλους nodes και το σύστημα αποθήκευσης.

5

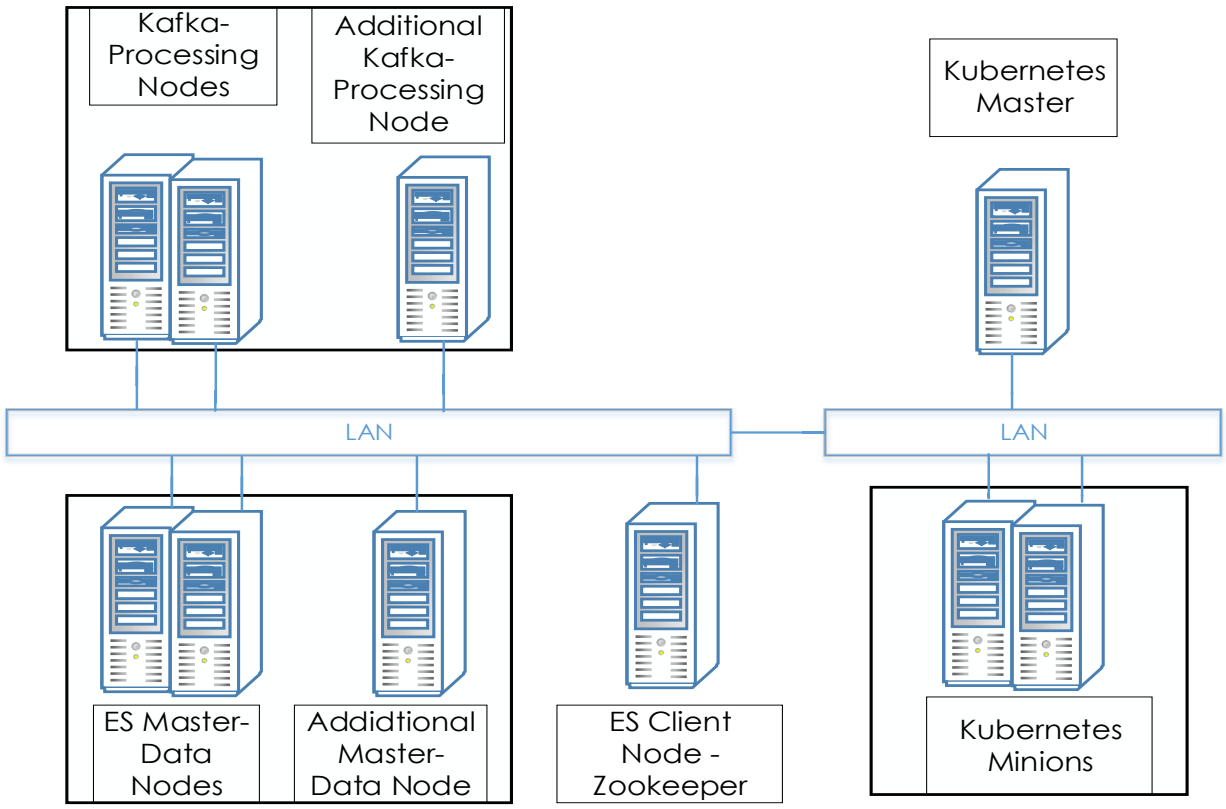
Αξιολόγηση Υλοποίησης

Στο κεφάλαιο αυτό, αρχικά αναλύεται η πειραματική διαδικασία που ακολουθήθηκε και τα αρχεία δεδομένων που δημιουργήθηκαν και χρησιμοποιήθηκαν για τις ανάγκες της προσομοίωσης αληθινής κίνησης. Επίσης, παρουσιάζονται τα δεδομένα τα οποία εξάγονται από την άνω διαδικασία και στην συνέχεια αξιολογούνται σε σχέση με το Logstash αλλά και το πώς συμπεριφέρεται με πολλαπλούς χρήστες κάτω από κίνηση ~1-2Gbps.

Για την εξαγωγή συμπερασμάτων κατά τη διάρκεια της αξιολόγησης έτρεχαν πειράματα για ~30 λεπτά, το οποίο σημαίνει ότι έπρεπε να υπάρχει μέσα στο Kafka ο απαραίτητος αριθμός μηνυμάτων ώστε η «κατανάλωση», επεξεργασία και η αποθήκευσή του στο elasticsearch cluster να μη διαρκεί λιγότερο.

5.1 Πειραματική διάταξη

Στην πειραματική διάταξη που επιλέχθηκε στόχος ήταν μεταξύ των άλλων να χρησιμοποιηθούν όσο το δυνατόν γίνεται λιγότεροι πόροι. Για αυτό το λόγο σε έναν κόμβο συνδυάστηκαν περισσότερα του ενός σταδίου από αυτά περιγράφηκαν στο κεφάλαιο 4. Βέβαια, για την τοποθέτηση του καθενός στοιχείου σε έναν κόμβο λήφθηκαν υπόψη οι υποδομές που χρειάζεται το καθένα ούτως ώστε να μη χρειαστεί, τουλάχιστον, κατά τη διάρκεια των πειραμάτων να διεκδικούν τις ίδιες.



Σχήμα 5.1 Πειραματικής διάταξη διπλωματικής

Επίσης, για την αξιολόγηση της κλιμακωσιμότητας της πλατφόρμας έχει σχεδιαστεί και ο αριθμός των κόμβων που προστέθηκαν στο κάθε cluster κατά τη διάρκεια των πειραμάτων. Οι πόροι που χρησιμοποιήθηκαν ήταν από τον ~Okeanos. Τα χαρακτηριστικά των κόμβων αναφέρονται στον παρακάτω πίνακα:

Στοιχείο	**Περιγραφή**
CPU	4cores @ 2.1 GHz
RAM	8 GB
Disk	50 GB

Για την υλοποίηση χρησιμοποιήθηκαν τα εργαλεία που σχολιάστηκαν στο κεφάλαιο 4.1 για την προσομοίωση κίνησης και εν τέλει την αποθήκευση της κίνησης στο Kafka. Σχετικά, με τις εκδόσεις των εργαλείων που χρησιμοποιήθηκαν δίνεται αναφορικά ο παρακάτω πίνακας:

Εργαλείο	**Έκδοση**
SflowTool	4.0.1
Kafka	0.10.2

Zookeeper	3.4.9
Elasticsearch	5.3.3
Kibana	5.3.3
Logstash	5.3.3
Kubernetes	1.5.2

5.2 Datasets

Για τις ανάγκες της διπλωματικής χρησιμοποιήθηκαν οι ελεύθερες εκδόσεις των datasets της Maxmind. Επίσης, για την αντιστοίχιση ιδιωτικών με δημόσιες IPs φτιάχτηκε με τη βοήθεια ενός script που αναπτύχθηκε ένα json αρχείο με ~400.000 γραμμές-αντιστοιχήσεις.

Συγκεκριμένα, χρησιμοποιήθηκαν τα GeoLite2 [37] datasets που δίνουν πληροφορίες για την τοποθεσία μιας IP και το αυτόνομο σύστημα στο οποίο ανήκει. Η δομή για την τοποθεσία μιας IP περιλαμβάνει τα εξής:

- `city`: το όνομα της πόλης από όπου προέρχεται μια IP, μέσα σε αυτήν την πληροφορία περιλαμβάνεται το όνομα της πόλης σε διάφορες γλώσσες καθώς και το γεωγραφικό μήκος και πλάτος
- `country`:το όνομα της χώρας προέλευσης μιας IP
- `continent`: ο κωδικός και το όνομα της ηπείρου
- `subdivisions`: πρόκειται για μια γραμματοσειρά με μέχρι τρεις χαρακτήρες που είναι συνδεδεμένη με την περιοχή από όπου προέρχεται η IP
- τα υπόλοιπα πεδία της δομής δε μας ενδιαφέρουν άμεσα

Η δομή για το αυτόνομο σύστημα περιλαμβάνει τα εξής:

- `network`:πρόκειται για την IPv4 ή IPv6 σε CIDR μορφή
- `autonomous_system_number`:ο αριθμός του αυτόνομου συστήματος που είναι συνδεδεμένη μια IP
- `autonomous_system_organization`: ο οργανισμός που είναι συνδεδεμένος με το παραπάνω πεδίο

Σχετικά με την κίνηση, χρησιμοποιήθηκε κίνηση από το Πολυτεχνείο που είχε γίνει anonymized. Στο πλαίσιο αξιολόγησης της πλατφόρμας και του τρόπου χρήσης της χρησιμοποιήθηκε επίσης και κομμάτι κίνησης από δικτυακή επίθεση.

5.3 Παρουσίαση αποτελεσμάτων

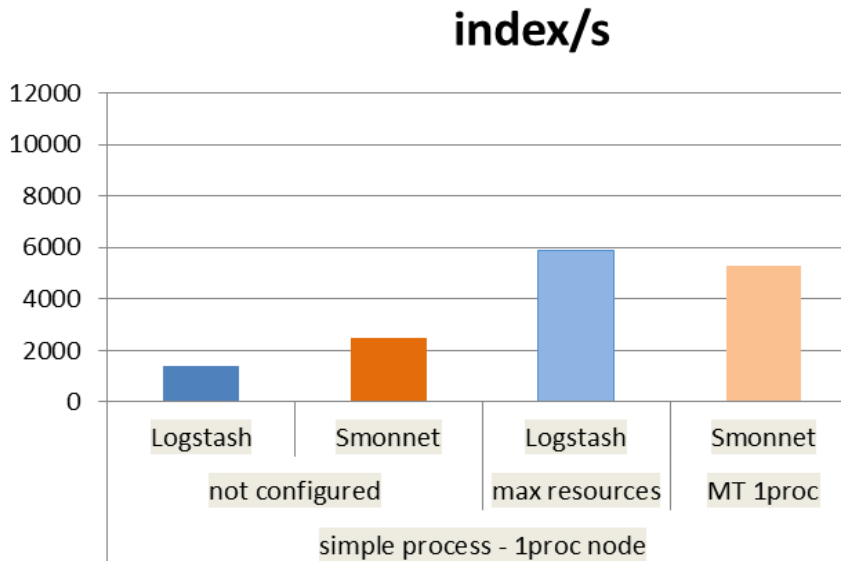
Στα επόμενα υποκεφάλαια θα εξεταστούν τα ακόλουθα:

- Το index/s στο Elasticsearch μέσω του Logstash και ενός python προγράμματος που υλοποιήθηκε με όνομα *Smonnet* (Sflow MONitoring NETwork), με και χωρίς περεταίρω εμπλουτισμό της κίνησης με βάση κάποια datasets
- Τη χρησιμοποίηση του επεξεργαστή και της μνήμης για την ίδια με πάνω περίπτωση
- Τα ίδια με τα παραπάνω σε περίπτωση δύο και τριών χρηστών χρησιμοποιώντας το πρόγραμμα σε python
- Τη χρησιμοποίηση του επεξεργαστή και της μνήμης του Elasticsearch cluster
- Τον χρόνο απόκρισης για τη δημιουργία νέου Elasticsearch cluster

Τέλος, παρουσιάζεται ένα use case χρήσης και εκμετάλλευσης της πλατφόρμας για την καταγραφή και τον εντοπισμό πιθανής DDOS επίθεσης μέσω ενός dashboard που υλοποιήθηκε στο Kibana.

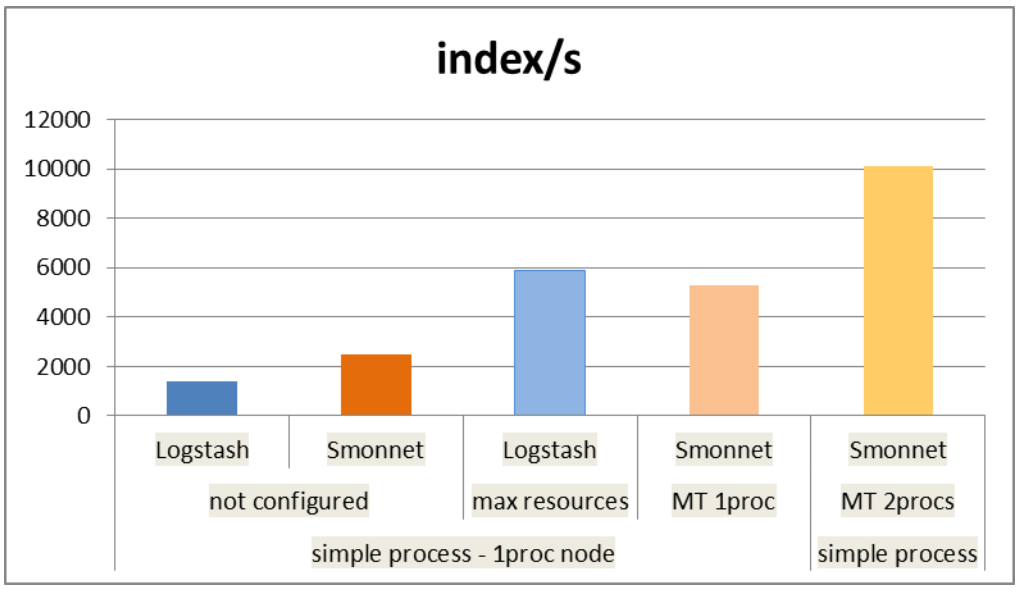
5.3.1 Logstash vs Smonnet

Αρχικά, συγκρίνονται 2 λύσεις του Logstash και του Smonnet με απλές ρυθμίσεις, μικρά batches και χωρίς τη χρησιμοποίηση όλων των πόρων που δίνονται. Ύστερα, δημιουργούνται μεγαλύτερα batches και χρησιμοποιούνται παραπάνω πόροι του συστήματος στην περίπτωση του logstash αυξάνοντας μεταξύ άλλων τους *workers* και στην περίπτωση του smonnet δημιουργώντας threads ώστε να υπάρχει ο λιγότερο δυνατός νεκρός χρόνος μεταξύ της απάντησης του elasticsearch για επιτυχή αποθήκευση και δημιουργίας του επόμενου batch. Τέλος, γίνεται απλά η αντιστοίχηση μεταξύ ιδιωτικών και δημόσιων IPs.



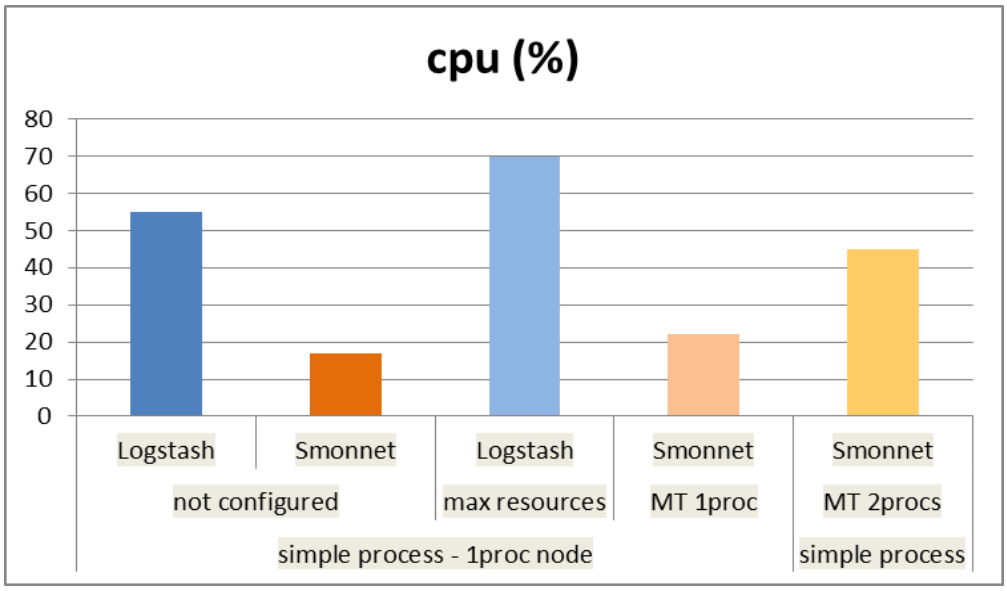
Σχήμα 5.2 Διάγραμμα απόδοσης Logstash και Smonnet ενός process

Παρατηρείται ότι στην τελική του μορφή το smonnet με 1 process είναι κατά 500 index/s χειρότερο. Για το λόγο αυτό προστίθεται ένα ακόμα process για να βελτιώσει τα αποτελέσματα του smonnet και για καλύτερη χρησιμοποίηση του συστήματος.

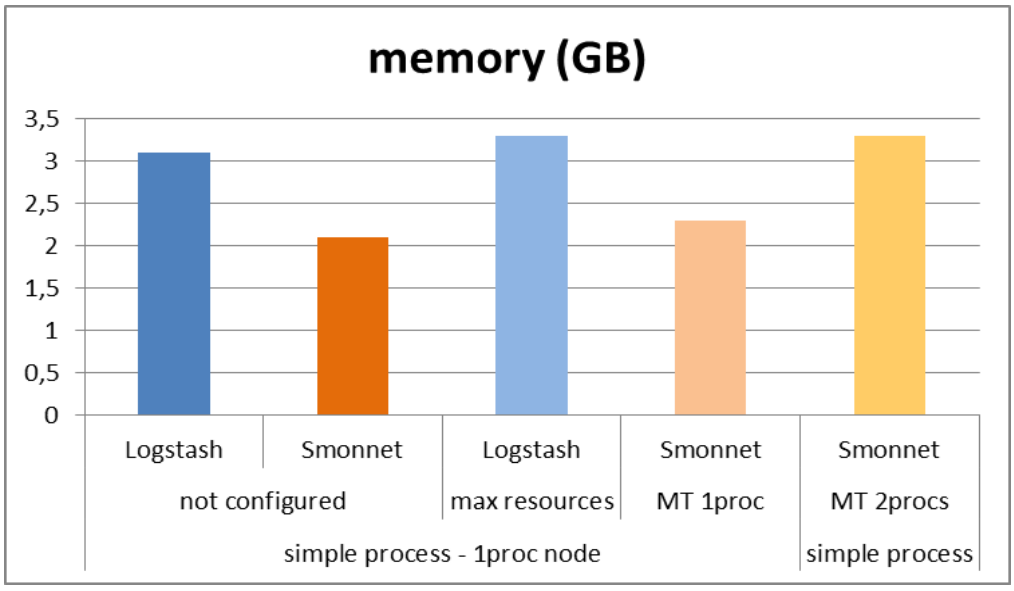


Σχήμα 5.3 Διάγραμμα απόδοσης Logstash και Smonnet 2 processes

Η τελευταία προσθήκη βελτίωσε σημαντικά τα αποτελέσματα και από τα παρακάτω γραφήματα γίνεται φανερό ότι παρά το process που προστέθηκε το smonnet χρησιμοποιεί λιγότερους πόρους από το logstash που δίνει περιθώριο για καλύτερη απόδοση.

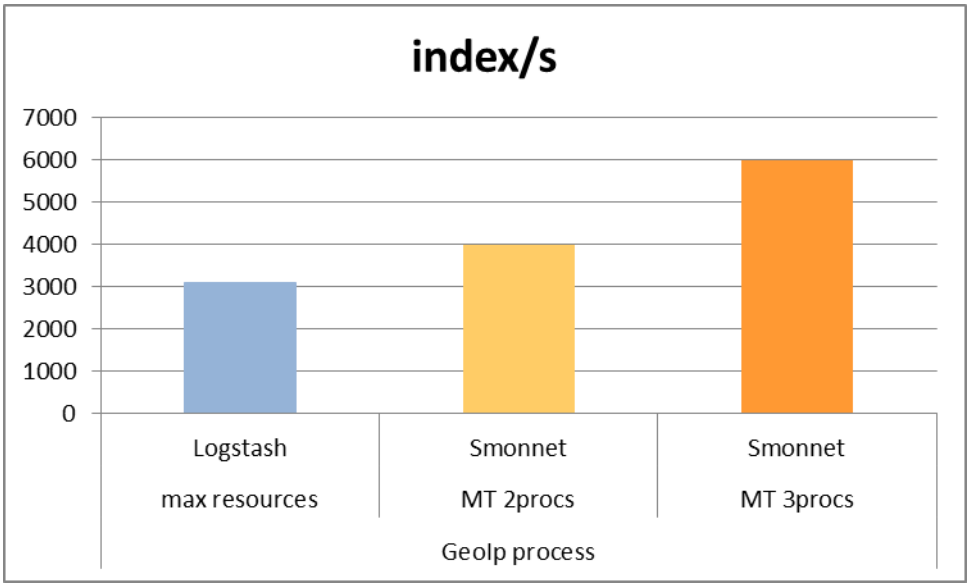


Σχήμα 5.4 Διάγραμμα χρησιμοποίησης CPU του Logstash και Smonnet 2 processes



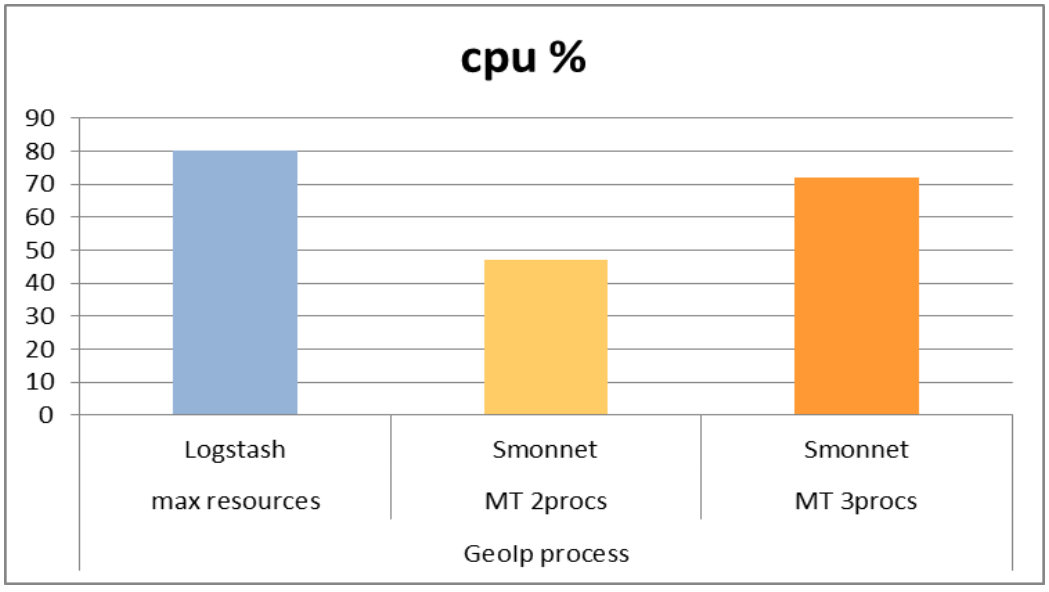
Σχήμα 5.5 Διάγραμμα χρήσης μνήμης του Logstash και Smonnet 2 processes

Τέλος, χρησιμοποιούνται και τα datasets που περιγράφηκαν για τον εμπλουτισμό της κίνησης και για να παρατηρηθεί η απόδοση του συστήματος κάτω από μεγαλύτερο φόρτο υπολογισμών κλιμακώνοντας κάθετα πάλι (scale up) το smonnet.



Σχήμα 5.6 Διάγραμμα απόδοσης Logstash και Smonnet 2 και 3 processes κάνοντας GeoIP processing

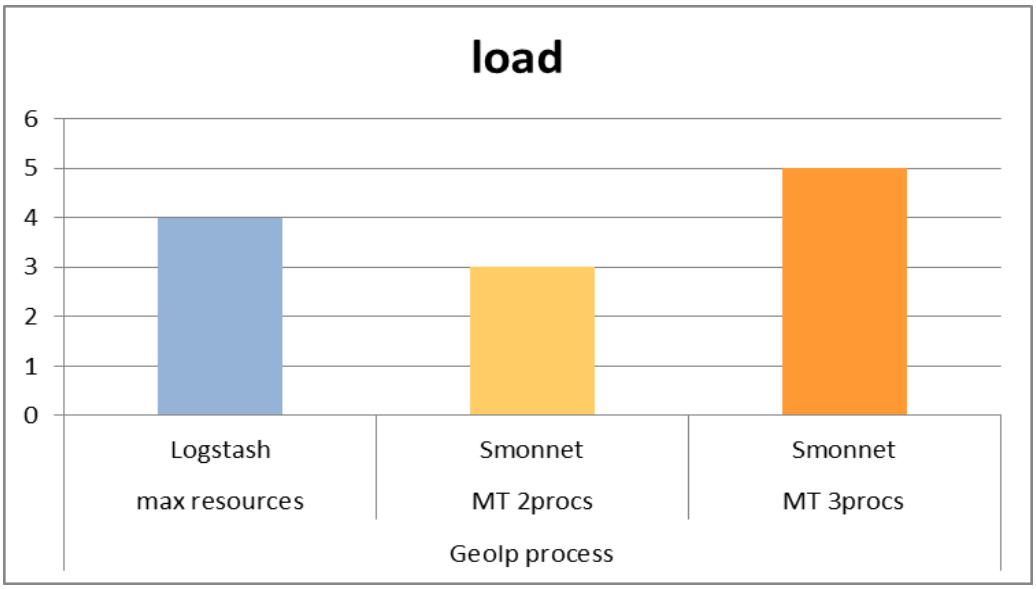
Παρατηρείται απευθείας ότι λόγω της παραπάνω επεξεργασίας και του μεγέθους των datasets καταναλώνονται παραπάνω πόροι και επίσης μειώνεται η απόδοση κατά την αποθήκευση.



Σχήμα 5.7 Διάγραμμα χρησιμοποίησης CPU Logstash και Smonnet 2 και 3 processes κάνοντας GeoIP processing

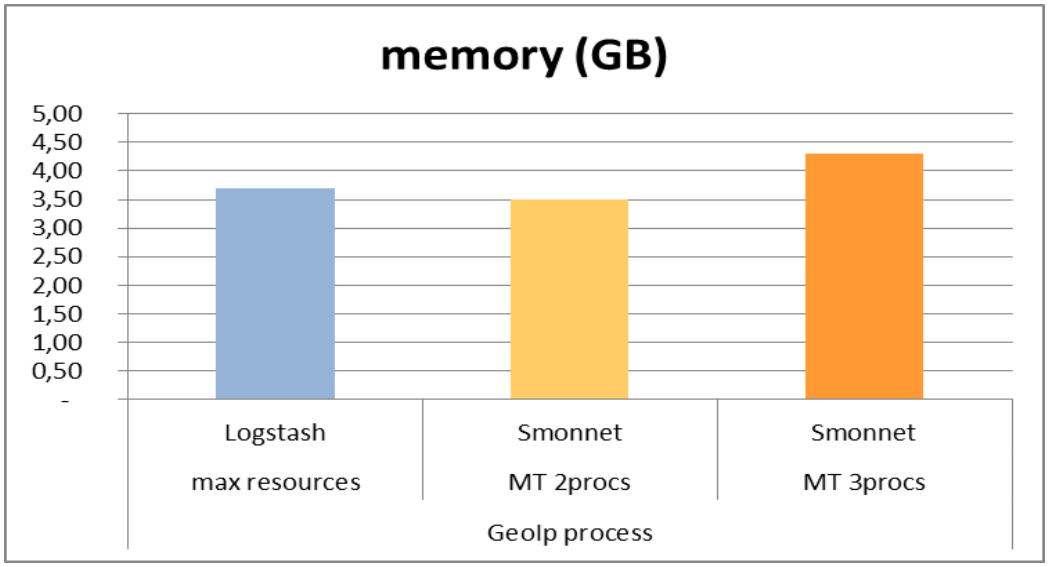
Σε αυτό το σημείο προστίθεται και γράφημα για το load του συστήματος που δείχνει το κατά πόσο χρησιμοποιούνται οι πυρήνες. Συνδυαστικά με το παραπάνω γράφημα με τη χρησιμοποίηση του επεξεργαστή, φαίνεται τυχόν συμφόρηση και η τάση αδρανοποίησης του επεξεργαστή κατά τη διακοπή για επικοινωνία με τον δίσκο ή με το δίκτυο. Λόγω του

περιβάλλοντος με πολλά threads και processes, σύμφωνα με το [43], οποιαδήποτε metrics που σχετίζονται με το IPS δε συμβάλλουν στην αξιολόγηση ενός συστήματος.



Σχήμα 5.8 Διάγραμμα του load του Logstash και Smonnet 2 και 3 processes κάνοντας GeoIP processing

Παρατηρείται, λοιπόν, ότι υπάρχει χαμηλή σχετικά χρησιμοποίηση του επεξεργαστή αλλά μεγάλος φόρτος στο σύστημα. Αυτό οφείλεται από τη μία λόγω της σειριοποίησης των γρήγορων μεν αλλά πολλών και συχνών δε δικτυακών αιτήσεων προς το Elasticsearch cluster που προκαλεί αναμονή στον επεξεργαστή (I/O wait) και από την άλλη λόγω της αναμονής της επεξεργασίας και συλλογής όλων αυτών των αιτήσεων.



Σχήμα 5.9 Διάγραμμα χρήσης μνήμης του Logstash και Smonnet 2 και 3 processes κάνοντας GeoIP processing

Τέλος, παρατηρείται, όπως είναι αναμενόμενο, πως λόγω των περισσότερων αρχείων που φορτώνονται στη μνήμη αλλά και περισσότερων processes που τρέχουν αυξάνεται η χρηση της μνήμης.

5.3.2 Απόκριση σε σύστημα με διαφορετικούς χρήστες

Στο συγκεκριμένο σημείο αξίζει να σημειωθεί ξανά ότι λόγω του τρόπου που δουλεύει το Kafka και ο consumer του, όπως έχει ήδη περιγραφεί, βοηθάει στη κλιμάκωση του συστήματος. Επίσης, λόγω του τρόπου που αναπτύχθηκε το smonnet υπάρχει η δυνατότητα, όπως μόλις φάνηκε, να προστίθενται processes και άρα να κλιμακώνει το σύστημα (scale up) βοηθώντας στην απόδοσή του και καλύτερη χρησιμοποίηση των πόρων. Στην περίπτωση παραπάνω χρηστών, πάλι δημιουργείται ένα παραπάνω process που έχει ξεχωριστή σύνδεση στο Kafka και στο καινούριο Elasticsearch cluster και έχει την ίδια ισχύ με το process που έτρεχε πριν.

Με όλα τα παραπάνω, συμπεραίνεται και αποδεικνύεται ότι δύο processes έχουν την ίδια απόδοση και καταναλώνουν τους ίδιους πόρους είτε βρίσκονται στον ίδιο κόμβο είτε σε διαφορετικό. Άρα, η μελέτη για περισσότερους χρήστες δίνει τα ίδια αποτελέσματα με αυτά των πολλαπλών processes. Ειδικότερα, πολλαπλοί χρήστες καταναλώνουν τους ίδιους πόρους που καταναλώνουν πολλαπλά processes και η απόδοση του συστήματος κατά την αποθήκευση είναι η ίδια καθώς υπάρχουν διαφορετικές συνδέσεις και κατανάλωση χρόνου διαφορετικού πυρήνα.

Αυτό περιορίζει κάπως τον αριθμό των χρηστών που μπορεί να αντέξει ένα σύστημα στον αριθμό των πυρήνων που διαθέτει. Παρόλα αυτά, σύμφωνα με τα παραπάνω γραφήματα στην περίπτωση πολλών processes και άρα ενδεχόμενων χρηστών, λόγω της χαμηλότερης χρήσης των πόρων διαπιστώνεται πως υπάρχει κάποιο περιθώριο για παραπάνω χρήστες από τον αριθμό των πυρήνων του συστήματος. Ο αριθμός των processes-χρηστών που μπορεί να αντέξει τελικά ένα σύστημα μελετάται στο παρακάτω υποκεφάλαιο σε συνδυασμό με την αξιολόγηση της κλιμακωσιμότητας του Elasticsearch.

5.3.3 Κλιμακωσιμότητα Elasticsearch και Smonnet

Σκοπός αυτής της αξιολόγησης είναι να βρεθεί ο αριθμός των πόρων που χρειάζεται το elasticsearch σε σχέση με τον αριθμό των processes του smonnet ούτως ώστε να τοποθετούνται κατά τη δημιουργία του cluster οι κατάλληλη πόροι στα container του Kubernetes. Επίσης, η σχέση processes με shards και elasticsearch κόμβων θα μπορεί να αναδείξει στο περίπου και έναν ανώτατο αριθμό processes που μπορεί να αντέξει το elasticsearch. Με αυτόν τον τρόπο θα μπορεί να φανεί και κατά πόσο μπορούν να υπάρχουν περισσότερα processes και άρα χρήστες από πυρήνες.

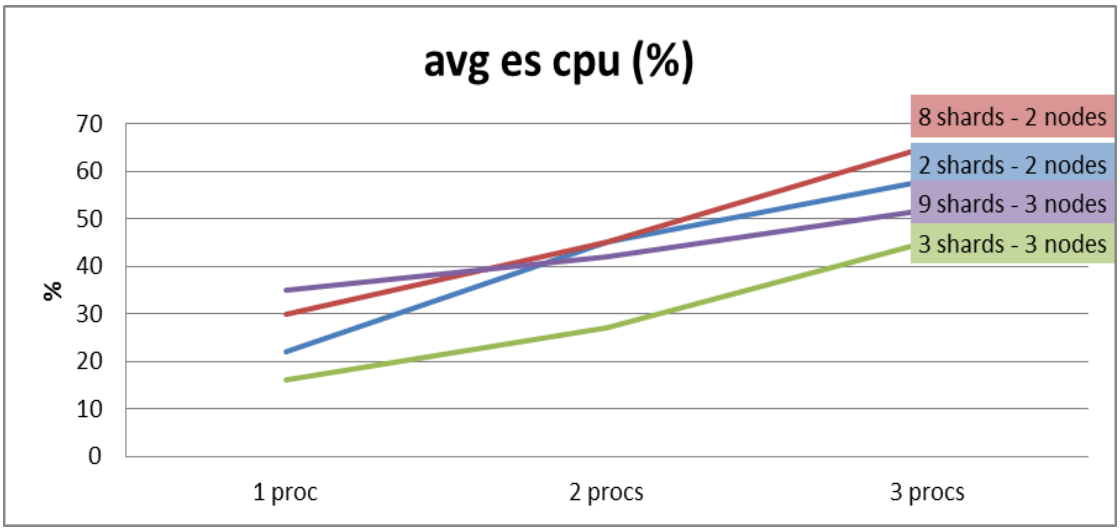
Τα κύρια χαρακτηριστικά των ρυθμίσεων που εφαρμόστηκαν στους κόμβους του elasticsearch πέραν των ρυθμίσεων στο σύστημα σχετικά με την εικονική μνήμη, τη filesystem cache και άλλα, είναι:

- `index.refresh_interval = 5s`, με αυτή τη ρύθμιση το elasticsearch μέσω του Lucene φτιάχνει ένα καινουριο τμήμα (segment) κάθε πέντε δευτερόλεπτα και τότε ανανεώνεται το index με καινούρια έγγραφα

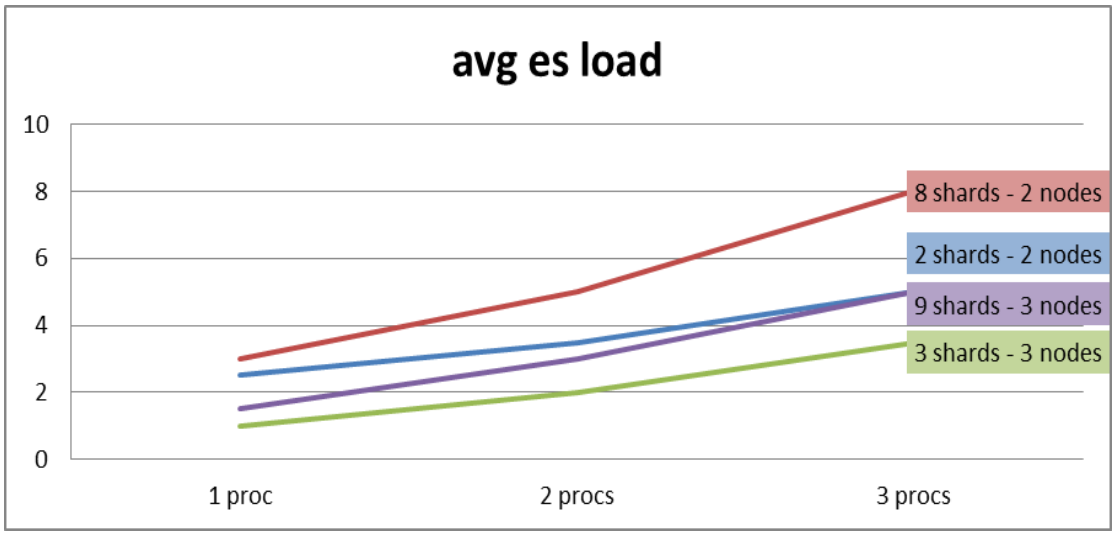
- `index.translog.durability = async`, κάθε shard έχει ένα transaction log ή write ahead log και κάθε αίτημα γράφεται πρώτα εκεί ύστερα στον δίσκο. Με αυτήν τη ρύθμιση γίνεται flush των transaction logs στον δίσκο κάθε ένα συγκεκριμένο διάστημα στο παρασκήνιο, χωρίς να μπλοκάρει τη διαδικασία

- `indices.memory.index_buffer_size = 512MB`, πρόκειται για το μέγεθος ενός μοιραζόμενου buffer μεταξύ των shards

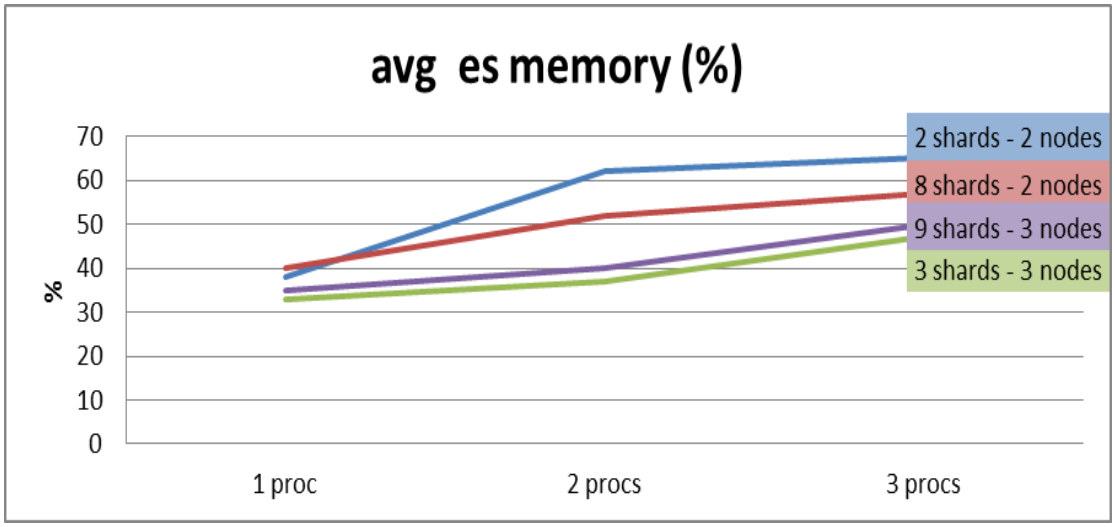
Παρακάτω λοιπόν παρουσιάζονται τα αποτελέσματα για τη χρησιμοποίηση του επεξεργαστή και της μνήμης των κόμβων του Elasticsearch, παίρνοντας τον μέσο όρο των κόμβων κάθε φορά στις περιπτώσεις που στέλνουν αιτήματα ένα, δύο ή τρία processes.



Σχήμα 5.10 Διάγραμμα χρησιμοποίησης CPU διαφορετικών Elasticsearch clusters σε σχέση με τον αριθμό των Smonnet processes



Σχήμα 5.11 Διάγραμμα load διαφορετικών Elasticsearch clusters σε σχέση με τον αριθμό των Smonnet processes

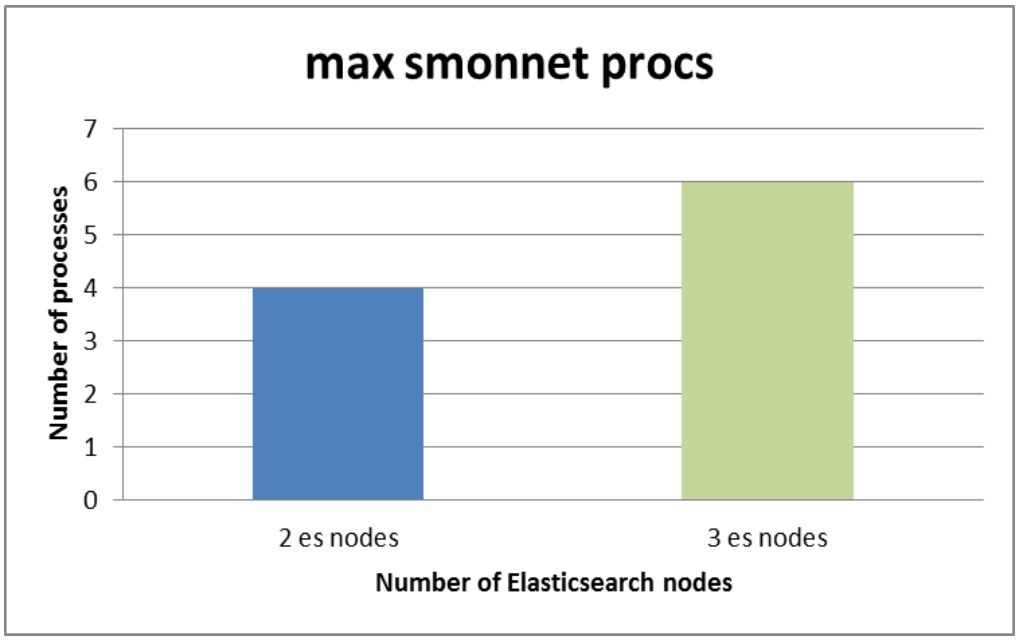


Σχήμα 5.12 Διάγραμμα χρήσης μνήμης διαφορετικών Elasticsearch clusters σε σχέση με τον αριθμό των Smonnet processes

Από τα παραπάνω γραφήματα, παρατηρείται πως στην περίπτωση που υπάρχουν παραπάνω shards από κόμβους χρειάζεται παραπάνω επεξεργαστική ισχύς για το διαμοιρασμό της πληροφορίας στα παραπάνω shards μέσα στον κάθε κόμβο. Σαφώς οποιαδήποτε διαφορά είναι πιο εμφανή στην περίπτωση που υπάρχουν λιγότεροι πόροι για αυτό στην περίπτωση με τους δύο κόμβους η μνήμη και η χρησιμοποίηση του επεξεργαστή φαίνεται μεγαλύτερη.

Επίσης, παρατηρείται μεγάλη αύξηση του φόρτου του συστήματος καθώς αυξάνονται τα processes του smonnet. Αυτό το φαινόμενο έχει να κάνει με την αναμονή του επεξεργαστή για απάντηση από τον δίσκο κάθε φορά που σκανδαλίζει κάθε shard ένα fsync στον δίσκο. Για το λόγο αυτό κιόλας περισσότερα shards έχουν μεγαλύτερο αντίκτυπο στον φόρτο του συστήματος.

Τέλος, παρακάτω δίνεται το γράφημα που υποδεικνύει τον μέγιστο αριθμό smonnet processes που μπορούν να στέλνουν αιτήματα σε ένα elasticsearch cluster με δύο και τρεις κόμβους. Αξίζει να σημειωθεί ότι ήδη από πιο νωρίς ίσως συναντηθεί κάποια απρόσμενη συμπεριφορά κατά την αποθήκευση (indexing) καθώς και ο ρυθμός των εγγράφων ανά δευτερόλεπτο (index/s) σε αυτές τις περιπτώσεις εμπεριέχει πιο μεγάλες διακυμάνσεις ή και μειώσεις.

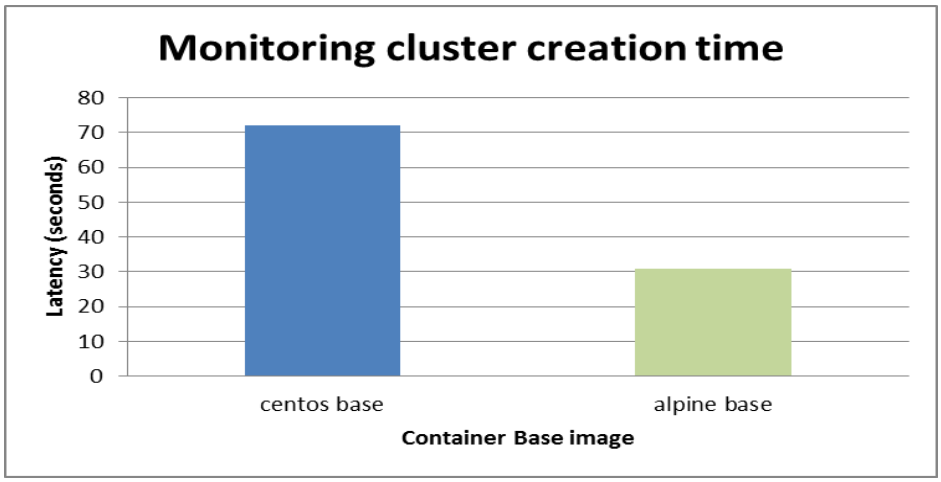


Σχήμα 5.13 Μέγιστος αριθμός Smonnet processes για διαφορετικά Elasticsearch clusters

5.3.4 Χρόνος απόκρισης νέων Elastisearch clusters στο Kubernetes

Με βάση τα παραπάνω, όπως έχει ήδη περιγραφεί, δημιουργήθηκαν κάποια docker images για τις ανάγκες της διπλωματικής. Ωστόσο, λόγω του μεγάλου μεγέθους που είχαν έκανε τη διαχείρισή τους αρκετά αργή. Για αυτόν το λόγο, έγινε αλλαγή του βασικού λειτουργικού που στηριζόταν το image ώστε να μπορεί να εκκινήσει πολύ πιο γρήγορα.

Παρακάτω παρουσιάζεται ο χρόνος απόκρισης για τη δημιουργία ενός ενός Elasticsearch cluster στο Kubernetes με δύο master-data κόμβους και έναν client μαζί με το kibana από τη στιγμή που γίνει το αίτημα για τη δημιουργία του μέχρι να δημιουργηθεί και να γίνει εμφανές το νέο index στο Kibana.



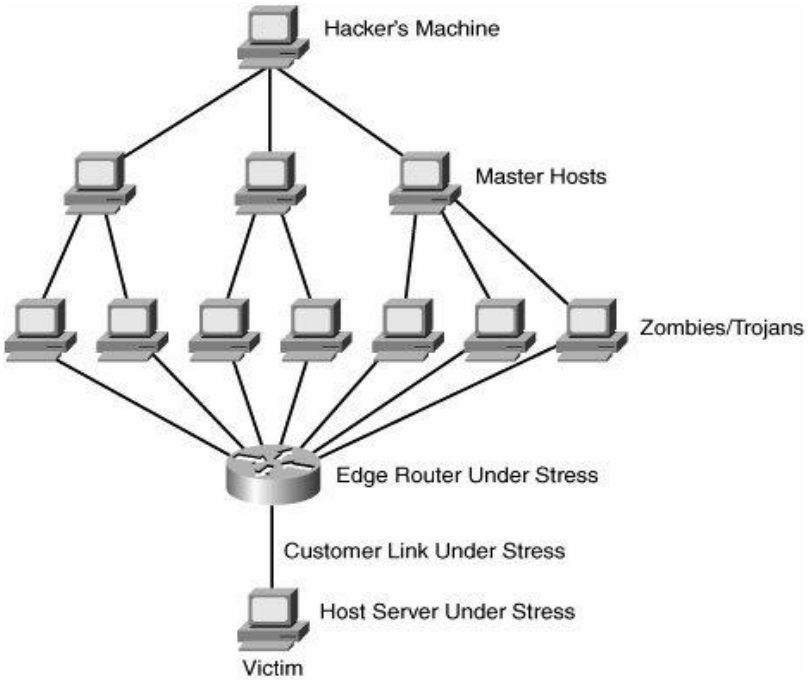
Σχήμα 5.14 Χρόνος δημιουργίας νέων monitoring clusters στο Kubernetes

5.4 Παράδειγμα χρήσης

Ξεφεύγοντας κάπως από τα προηγούμενα στάδια σε αυτό το υποκεφάλαιο δίνεται έμφαση στους τρόπους χρήσης και εκμετάλλευσης της τελικής υπηρεσίας που προσφέρει αυτή η πλατφόρμα και έχει να κάνει με το εργαλείο καταγραφής και παρακολούθησης. Συγκεκριμένα, το Kibana είναι το εργαλείο που μπορεί να χρησιμοποιηθεί για την τελική ανάλυση οπτικοποιώντας τις συσχετίσεις πάνω σε σύνολα δεδομένων.

Τα dashboards που μπορούν να δημιουργηθούν στο Kibana δίνουν μια εικόνα των δεδομένων κάθε στιγμή και μπορεί να αναδείξουν την περίπτωση κάποιας δικτυακής επίθεσης και πιο συγκεκριμένα κάποιου τύπου DDoS (Distributed Denial of Service). Μια επίθεση είναι τύπου DDoS, όταν ένας μεγάλος αριθμός μηχανημάτων έχει προσβληθεί από κακόβουλο κώδικα και προσπαθεί ταυτόχρονα και συντονισμένα, μετά των οδηγιών του επιτιθέμενου να εξουθενώσουν τους πόρους του θύματος και να το αναγκάσουν να αρνηθεί τις υπη- ρεσίες του στους νόμιμους χρήστες.

Σε μία απλή επίθεση άρνησης παροχής υπηρεσίας ο επιτιθέμενος χρησιμοποιεί ένα δίκτυο από προσβεβλημένους υπολογιστές οι οποίοι διακρίνονται σε αφέντες (master) και σκλάβους (slave). Ο επιτιθέμενος καθοδηγεί και προστάζει στους αφέντες και εκείνοι με τη σειρά τους καθοδηγούν τους σκλάβους να εκκινήσει η επίθεση προς το επιθυμητό θύμα.



Σχήμα 5.15 DDoS επίθεση

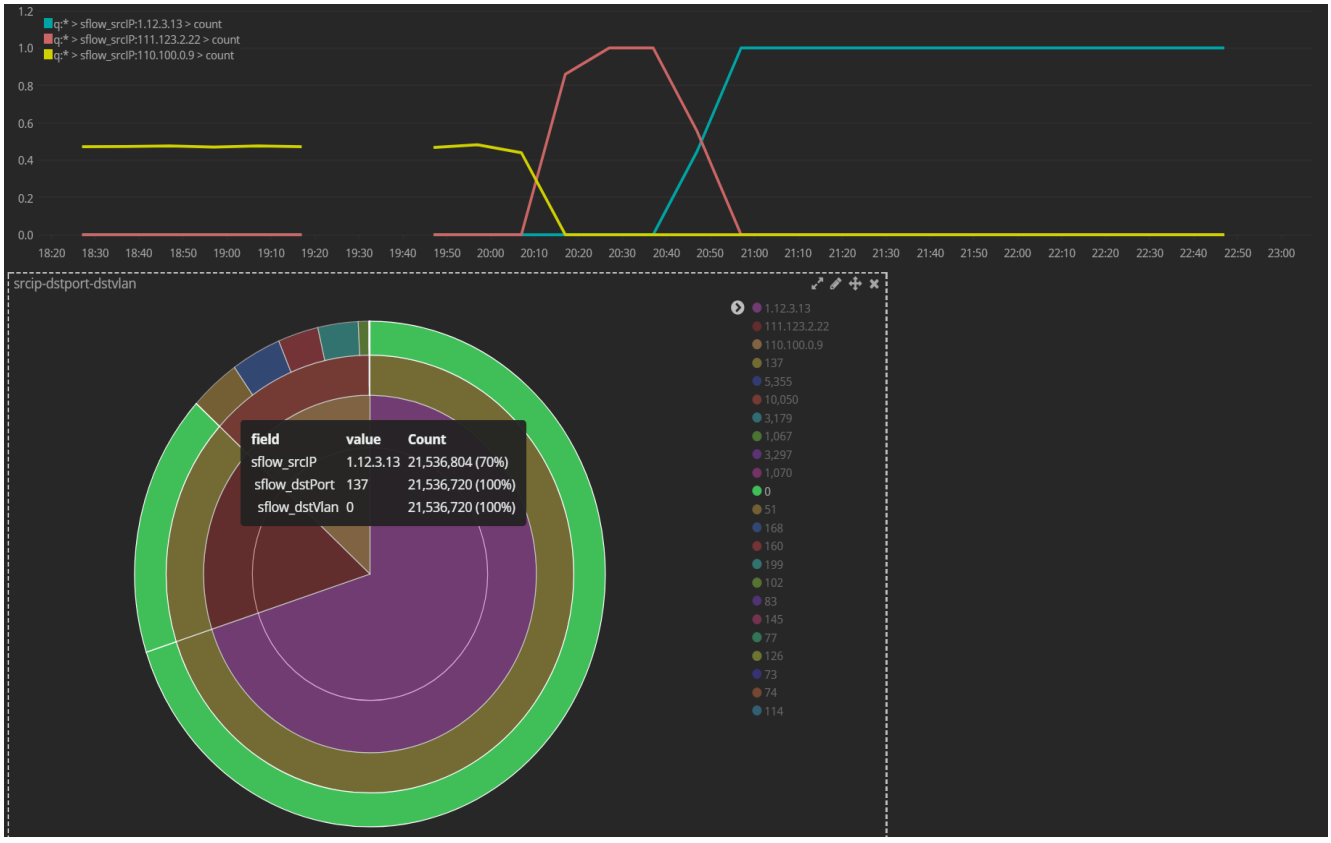
Είναι αρκετά σύνηθες σε τέτοιου είδους επιθέσεις τα πακέτα που στέλνονται στο θύμα να έχουν διαφορετικές διευθύνσεις από τις πραγματικές διευθύνσεις (spoofed). Αυτή η τεχνική προτιμάται για δύο λόγους. Από τη μία ο επιτιθέμενος θέλει να αποκρύψει τις πραγματικές διευθύνσεις των σκλάβων για να μην μπορεί να ανιχνευθεί η πηγή της επίθεσης και από την άλλη να μπορεί να εναλλάσσει δυναμικά τις διευθύνσεις των επιτιθέμενων ώστε να μην μπορούν να εφαρμοσθούν στατικά φίλτρα για την αντιμετώπιση των επιθέσεων.

Για τις ανάγκες της διπλωματικής χρησιμοποιήθηκε dataset που πληροί τα χαρακτηριστικά DDoS κίνησης. Συγκεκριμένα, έχει να κάνει με τρεις διαφορετικές IP προέλευσης οι οποίες πηγαίνουν και «χτυπάνε» κάποια/ες συγκεκριμένες IP ενός οργανισμού σε συγκεκριμένη πόρτα.

Στο παρακάτω dashboard του Kibana φαίνονται τα εξής:

- Ένας χάρτης σαν heatmap που δείχνει από ποιες περιοχές προέρχονται οι περισσότερες IP πηγής
- Μετρητές σε ένα συγκεκριμένο χρονικό διάστημα σχετικά με:
    - Τον αριθμό των μοναδικών IPs (πηγής και προορισμού)
    - Την IP προέλευσης του sflow που συναντάται περισσότερο
    - Τη χώρα από όπου προέρχονται οι περισσότερες IPs προέλευσης
    - Η IP προορισμού των περισσότερων flows που έχουν για IP προέλευσης αυτήν που συναντάται περισσότερο
- Διάγραμμα που δείχνει τον αριθμό των χωρών από όπου προέρχονται οι IPs προέλευσης σε σχέση με τον χρόνο
- Διάγραμμα που λειτουργεί σαν δείκτης για επίθεση μορφής DDoS σύμφωνα με το [42] και περιλαμβάνει τους μετρητές από τις τρεις IPs προέλευσης που συναντούνται περισσότερο διαιρεμένες με τον συνολικό αριθμό των πακέτων στο ίδιο χρονικό διάστημα
- Διάγραμμα σε μορφή πίτας που συσχετίζει από ποια IP προέλευσης προέρχεται ένα πακέατο και σε ποιο vLAN και πόρτα προορισμού κατευθύνεται

Σχήμα 5.16 Kibana dashboard για εντοπισμό κακόβουλων ενεργειών (1)



Σχήμα 5.17 Kibana dashboard για εντοπισμό κακόβουλων ενεργειών (2)

6

Επίλογος

Σε αυτό το κεφάλαιο παρουσιάζεται μια σύνοψη της παρούσας διπλωματικής εργασίας και συμπεράσματα πάνω στην πειραματική διαδικασία. Επιπροσθέτως, παρατίθενται διάφορες προτάσεις για βελτίωση και περαιτέρω επέκταση στο μέλλον.

6.1 Σύνοψη και συμπεράσματα

Στην παρούσα διπλωματική εργασία προτάθηκε μια πλατφόρμα που συλλέγει δεδομένα (logs) από υποδομές και έχει στόχο να παρέχει κατ' απαίτηση σε διαφορετικούς ενοίκους τον απαραίτητο μηχανισμό για καταγραφή και παρακολούθηση αυτών των δεδομένων. Στη συγκεκριμένη διπλωματική δόθηκε έμφαση στη συλλογή δικτυακών δεδομένων από τους μεταγωγείς και για να ανταποκριθεί στις απαιτήσεις που έχει η δικτυακή κίνηση μελετήθηκε η απόδοση της πλατφόρμας κατά την καταγραφή (indexing) στο εργαλείο αποθήκευσης και ανάλυσης, το Elasticsearch. Στο πλαίσιο της απόδοσης μελετήθηκε η κλιμακωσιμότητα της πλατφόρμας και συγκρίθηκε μια λύση με χρήση του Logstash που ανήκει κι αυτό στο ELK stack με το Smonnet. Το Smonnet είναι ένα python πρόγραμμα που αναπτύχθηκε για τις ανάγκες τις διπλωματικής το οποίο επιτρέπει και τη σύνδεση σε διαφορετικά Elasticsearch clusters ταυτόχρονα.

Σχετικά με την απόδοση πάνω στην αποθήκευση των δεδομένων παρατηρήθηκε ότι το Logstash καταναλώνει μεγάλη υπολογιστική ισχύ λόγω των φίλτρων που χρησιμοποιεί με αποτέλεσμα να επηρεάζει αρνητικά τον ρυθμό αποθήκευσης. Σε αντίθεση το Smonnet φάνηκε αρκετά πιο ευέλικτο χρησιμοποιώντας παραπάνω processes ώστε να κάνει scale up και εν τέλει να κάνει στέλνει μεγαλύτερο αριθμό αιτημάτων προς το elasticsearch. Προς αυτήν την κατεύθυνση, έγινε και η αξιολόγηση σχετικά με τον αριθμό των processes και τον

αριθμό των shards και κόμβων του elasticsearch. Σε αυτήν την περίπτωση διαπιστώθηκε ότι η αύξηση των κόμβων του elasticsearch δεν βελτιώνουν τόσο την απόδοση και τον ρυθμό κατά την αποθήκευση, αντίθετα διευκολύνει κατά το διάβασμα, σύμφωνα και με τις πηγές [43] και [44]. Συγκεκριμένα, κατά την αποθήκευση και το διάβασμα το shard overallocation καταναλώνει περισσότερους πόρους και δίνει σχετικά χειρότερα αποτελέσματα. Τέλος, βρέθηκε ο μέγιστος αριθμός των processes που μπορεί να δέχεται ένα elasticsearch cluster ώστε να μη γεμίζουν οι ουρές και οι buffers των κόμβων του cluster. Αυτόο έδωσε και κάποιες πληροφορίες για τον αριθμό των processes που μπορούν να συνυπάρχουν παράλληλα.

Σχετικά με την αρχιτεκτονική της πλατφόρμας, χρησιμοποιήθηκε το Kafka ως message broker εξαιτίας του σταδίου επεξεργασίας και εμπλουτισμού αλλά και κυρίως λόγω του bottleneck που δημιουργείται, όπως είναι φυσικό, κατά την αποθήκευση στο Elasticsearch. Τα δύο αυτά στάδια κάνουν μεγάλη χρήση της CPU για τα flows που λαμβάνονται. Το Kafka, λοιπόν, λαμβάνει ασύγχρονα flows από απροσδιόριστο αριθμό πηγών (μεταγωγέων) και αποθηκεύει προσωρινά τα δεδομένα εξομαλύνοντας έτσι την κίνηση που τροφοδοτείται στην πλατφόρμα.

Όσον αφορά την ενορχήστρωση δόθηκε βαρύτητα στη γρήγορη έναρξη ενός νέου cluster που αντιστοιχεί σε νέο πελάτη στο Kubernetes. Για το λόγο αυτό έγιναν αλλαγές και προτιμήθηκε τελικά το alpine σαν λειτουργικό στη βάση των images που δημιουργήθηκαν για τις ανάγκες της διπλωματικής.

6.2 Μελλοντικές επεκτάσεις

Η παρούσα διπλωματική έδωσε έμφαση σε δικτυακά δεδομένα που λαμβάνονται από τους μεταγωγείς. Ωστόσο, λόγω του ότι θα μπορούσε να εξυπηρετήσει πολλούς διαφορετικούς σκοπούς για διαφόρων ειδών δεδομένα αρχείων (logs) θα μπορούσε να δημιουργηθεί ένα πλαίσιο όπου αποτυπώνονται τα μέρη των logs και εμπλουτίζονται ανάλογα. Συγκεκριμένα, logs που θα φαινόντουσαν ακόμα χρήσιμα είναι αυτά από τα υπολογιστικά συστήματα καθώς και περεταίρω πληροφορίες από το δίκτυο, όπως για παράδειγμα μέσω του netflow που περιγράφηκε στο Θεωρητικό Υπόβαθρο.

Η ενορχήστρωση είναι επίσης ένα σημαντικό κομμάτι της πλατφόρμας που αξίζει περεταίρω μελέτη. Το στάδιο της επεξεργασίας και εμπλουτισμού συγκεκριμένα λόγω του περιορισμού στον αριθμό των processes που μπορούν να στέλνουν ταυτόχρονα στο Elasticsearch και επειδή κλιμακώνεται για διαφορετικούς χρήστες στη μορφή νέων processes, θα μπορούσε να τρέχει σαν πρόγραμμα ενός process στο Kubernetes μαζί με το Elasticsearch cluster. Σε αυτό τον τρόπο έχει ίσως ενδιαφέρον ο τρόπος που δίνονται πόροι από το Kubernetes και ο τρόπος

που δικτυώνει τα διαφορετικά Pods μεταξύ τους. Με αυτόν τον τρόπο στόχος είναι να γίνει η πλατφόρμα πιο αυτόνομη και να πλησιάσει πιο πολύ σε λειτουργία και διαχείριση με αυτή μιας εικονοποιημένης υπηρεσίας (Virtual Network Function).

Τέλος, λόγω του ότι η συγκεκριμένη πλατφόρμα μπορεί να αποτελέσει βάση μιας ενιαίας πλατφόρμας άντλησης δεδομένων, μια επιπλέον επέκταση θα μπορούσε να είχε να κάνει με την αλυσιδωτή σύνδεση άλλων υπηρεσιών όπως για παράδειγμα αυτής που θα εντοπίζει κάποιες επιθέσεις η οποία με τη σειρά της θα συνδέεται με την υπηρεσία που είναι υπεύθυνη να αντιμετώπιση αυτήν την επίθεση. Πέραν από υπηρεσίες σχετικά με ασφάλεια δικτύων που μπορούν να βασίζονται στα δεδομένα και λειτουργίες μιας τέτοιας πλατφόρμας, η περεταίρω ανάλυση logs και η εφαρμογή αλγορίθμων μηχανικής μάθησης μπορούν να εφαρμοστούν και να γίνονται προβλέψεις χρήσης των υπολογιστικών ή δικτυακών πόρων ώστε να υπάρχει κάθε φορά πιο αποδοτική χρησιμοποίησή τους. Επίσης, μια τέτοιου είδους ανάλυση μπορεί να βοηθήσει επίσης πάλι σε θέματα ασφαλείας ή ακόμα και σε περιπτώσεις βλαβών.

7

Βιβλιογραφία

[1] ELK stack, https://oliververeits.files.wordpress.com

[2] Deploying and Scaling Logstash, https://www.elastic.co/guide/en/logstash/5.0/

[3] Kibana Dashboards, http://anderikistan.com

[4] Container Security, https://www.slideshare.net/jlkinsel/understanding-container-security

[5] Container Architecture, https://www.nextplatform.com

[6] Docker, http://slides.com/stevenborrelli/docker

[7] Kubernetes Architecture, https://x-team.com/blog/introduction-kubernetes-architecture/

[8] Sflow Datagram, http://support.huawei.com/enterprise

[9] Kafka Architecture, http://cloudurable.com/blog/kafka-architecture/index.html

[10] sFlow, http://www.sflow.org/.

[11] "CiscoSystemsNetFlowServicesExportVersion9", https://www.ietf.org/rfc/rfc3954.txt .

[12] Apache Flume, http://flume.apache.org/.

[13] Scribe, https://github.com/facebookarchive/scribe .

[14] Fluentd, http://www.fluentd.org/.

[15] Redis, http://redis.io/.

[16] Apache Hadoop, http://hadoop.apache.org

[17] Apache HBase, http://hbase.apache.org/

[18] Cassandra, http://cassandra.apache.org/

[19] MongoDB, http://www.mongodb.org/

[20] Apache Spark, https://spark.apache.org/

[21] Apache Storm, https://storm.apache.org/

[22] Graylog2, http://graylog2.org/.

[23] Apache Lucene, http://lucene.apache.org

[24] Elasticsearch, http://www.elasticsearch.org/overview/elasticsearch/.

[25] Logstash, http://www.elasticsearch.org/overview/logstash/.

[26] Kibana, http://www.elasticsearch.org/overview/kibana/.

[27] JSON, http://json.org/.

[28] Docker, https://docs.docker.com/

[29] Kubernetes, https://kubernetes.io/docs/home/

[30] Ganglia, http://ganglia.sourceforge.net/.

[31] Tornado Web Server, http://www.tornadoweb.org/en/stable/

[32] Apache Zookeeper, https://zookeeper.apache.org/

[33] Apache Kafka, http://kafka.apache.org/.

[34] ETSI GS NFV 002 V1.1.1 (2013-10).

[35] M.Boniface, B. Nasser, J. Papay, "Platform-as-a-Service Architecture for Real-Time Quality of Service Management in Clouds", ACM SIGCOMM, 2009

[36] "RESTful Web APIs", Leonard Richardson, Mike Amundsen, Sam Ruby, Εκδόσεις O'Reilly

[37] GeoLite Database. http://dev.maxmind.com/geoip/geoip2/geolite2/

[38] tcpreplay, http://linux.die.net/man/1/tcpreplay.

[39] Open vSwitch, http://www.openvswitch.org/.

[40] sflowtool | sFlow collector, https://github.com/xchenum/sflowtool-mod.

[41] Christina Delimitrou and Christos Kozyrakis, "Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters", 2016

[42] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras and V. Maglaris, "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," *Computer Networks*, 2014.

[43] Σταύρος Δήμαρχος, "Σχεδιασμός και ανάπτυξη ενός κατανεμημένου συστήματος για τη συλλογή, ανάλυση και αποθήκευση των δεδομένων χρήσης

που παράγονται από τη λειτουργία ενός υπολογιστικού νέφους", Εθνικό Μετσόβιο Πολυτεχνείο, Αθήνα, Ιούλιος 2014.

[44] Χρήστος Μάρκου, "Μελέτη εγκατάστασης και κλιμακωσιμότητας κατανεμημένου συστήματος διαχείρισης μεγάλου όγκου δεδομένων χρήσης υπολογιστικών νεφών", Εθνικό Μετσόβιο Πολυτεχνείο, Αθήνα, Οκτώβριος 2015.

[45] Αδάμ Παυλίδης, "Σχεδιασμός και ανάπτυξη μηχανισμών συλλογής δεδομένων σε πειραματικές πλατφόρμες για το Internet του μέλλοντος", Εθνικό Μετσόβιο Πολυτεχνείο, Αθήνα, Ιούλιος 2015.