ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ & ΠΛΗΡΟΦΟΡΙΚΗΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«ΜΕΛΕΤΗ ΣΥΣΤΗΜΑΤΩΝ CONTAINERIZATION ΚΑΙ ΕΦΑΡΜΟΓΗ ΤΟΥ DOCKER ΣΕ ΣΥΣΤΗΜΑ ΜΕΤΡΗΣΗΣ ΤΗΣ ΠΟΙΟΤΗΤΑΣ ΕΝΟΣ WIFI ΔΙΚΤΥΟΥ»

ΘΕΟΔΩΡΟΣ ΔΗΜΟΣ

Α.Μ 4711

ΥΠΕΥΘΥΝΟΣ ΚΑΘΗΓΗΤΗΣ:

Χρήστος Μπούρας, Καθηγητής

ΕΠΙΒΛΕΠΩΝ:

Βασίλειος Κόκκινος

ΠΑΤΡΑ 2018

ΠΕΡΙΛΗΨΗ

Το Docker (https://docs.docker.com/) παρέχει τη δυνατότητα τρεξίματος εφαρμογών σε ένα container, το οποίο περιέχει όλες τις εξαρτήσεις και τις βιβλιοθήκες που είναι απαραίτητες για το τρέξιμο της εφαρμογής. Με τον τρόπο αυτό, με μια μόνο εντολή μπορούν να ξεκινήσουν όλες οι υπηρεσίες που απαιτούνται για να ξεκινήσει η εφαρμογή.

Στόχος της συγκεκριμένης διπλωματικής είναι αρχικά η μελέτη του Docker, εστιάζοντας στα βασικά του στοιχεία και δυνατότητες. Σε δεύτερη φάση, στόχος της διπλωματικής είναι η υλοποίηση σε Docker του συστήματος μέτρησης ενός WiFi δικτύου που έχει αναπτυχθεί στο πλαίσιο του ευρωπαϊκού έργου GEANT. Όλα τα απαραίτητα εργαλεία για την υλοποίηση του συστήματος (kibana, elasticsearch, postgresql, κτλ.)θα πρέπει να εγκαθίστανται αυτόματα με τη χρήση του Docker.

EXECUTIVE SUMMARY

Docker (https://docs.docker.com/) provides the ability to run applications inside a container, which contains all the dependencies and libraries that are necessary in order to run the application. This way, we can start all the services that are required to run the application with only a single command.

The target of this thesis is initially the study of Docker, focusing on its main components and benefits. In a second stage, the target of this thesis is to deploy using Docker the WiFi performance measurement system that has been developed by Geant. All the necessary tools for the implementation of the system (kibana, elasticsearch, postgresql, etc.) will be installed using Docker.

ΠΡΟΛΟΓΟΣ

Η παρούσα διπλωματική εργασία αποτελεί τον επίλογο των προπτυχιακών μου σπουδών στο Τμήμα Μηχανικών Η/Υ και Πληροφορικής. Το περιεχόμενο της εργασίας σχετίζεται με την μελέτη συστημάτων containerization και την εφαρμογή του Docker σε σύστημα μέτρησης της ποιότητας ενός WiFi δικτύου. Η εκπόνηση της διπλωματικής αυτής εργασίας αποτελεί το επιστέγασμα των πολύτιμων γνώσεων που είχα την τύχη και τη χαρά να αποκτήσω όλο αυτό το χρονικό διάστημα.

Πριν την παρουσίαση των αποτελεσμάτων της διπλωματικής εργασίας, αισθάνομαι την ανάγκη να ευχαριστήσω θερμά όλους όσους με στήριξαν και μου συμπαραστάθηκαν σε όλη τη διάρκεια των προπτυχιακών μου σπουδών.

Αρχικά, θα ήθελα να ευχαριστήσω τον Καθηγητή Χρήστο Μπούρα, Καθηγητή του Τμήματος Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής και Επιστημονικό Υπεύθυνο της Ερευνητικής Μονάδας 6 του ΕΑΙΤΥ, ο οποίος ήταν και ο υπεύθυνος καθηγητής της διπλωματικής μου εργασίας. Θέλω να τον ευχαριστήσω θερμά για τις συμβουλές, την καθοδήγηση του καθώς και για την ευκαιρία που μου έδωσε να ασχοληθώ εκτενώς με αυτό το σύγχρονο και ενδιαφέρον ερευνητικό τομέα.

Επιπλέον, αισθάνομαι την ανάγκη να ευχαριστήσω το Βασίλειο Κόκκινο, μέλος της Ερευνητικής Μονάδας 6 στο ΙΤΥ-Ε (Διόφαντος), για την άψογη συνεργασία που είχαμε και την ουσιαστική βοήθεια του στην υλοποίηση της διπλωματικής μου εργασίας.

Κλείνοντας, θα ήθελα να απευθύνω ένα μεγάλο ευχαριστώ στους γονείς μου, Επαμεινώνδα και Λαμπρινή καθώς και στα αδέρφια μου, Σοφία και Κυριάκο, για τη ψυχική και υλική στήριξη τους στην προσπάθεια των προπτυχιακών μου σπουδών.

Πάτρα, Σεπτέμβριος 2018

Δήμος Ε. Θεόδωρος

ΠΕΡΙΕΧΟΜΕΝΑ

ΛΙΣΤΑ ΕΙΚΟΝΩΝ

ΛΙΣΤΑ ΠΙΝΑΚΩΝ

ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ

ΕΙΣΑΓΩΓΗ

Στόχος της παρούσας εργασίας είναι αρχικά η μελέτη του συστήματος containerization Docker εστιάζοντας στα βασικά στοιχεία και χαρακτηριστικά ώστε να κατανοήσουμε καλύτερα τις δυνατότητες που αυτό μας προσφέρει..

Στην συνέχεια θα εξετάσουμε το σύστημα μέτρησης της ποιότητας ενός WiFi δικτύου «WiFiMon» ώστε να κατανοήσουμε τις απαιτήσεις και ιδιαιτερότητες που μπορεί να έχει η υλοποίηση σε Docker του συστήματος αυτού.

Τέλος, θα υλοποιηθεί σε Docker το σύστημα μέτρησης της ποιότητας ενός WiFi δικτύου «WiFiMon». Πρωταρχικός στόχος αυτής της υλοποίησης είναι να μπορεί να γίνεται εύκολο η εγκατάσταση του συστήματος WiFiMon. Θα φροντίσουμε έτσι ώστε όλα τα απαραίτητα εργαλεία για την υλοποίηση του συστήματος (elasticsearch, kibana, postgresql, κτλ.) να εγκαθίστανται αυτόματα με τη χρήση του Docker καθώς και ότι ολόκληρο το σύστημα να μπορεί να εκκινείτε με μόνο μια εντολή.

Συγκεκριμένα στο Κεφάλαιο 2 γίνεται μια μελέτη του Docker εστιάζοντας στα βασικά του στοιχεία και δυνατότητες.

Στο Κεφάλαιο 3 γίνεται μια αναθεώρηση της σχετικής βιβλιογραφίας.

Στο Κεφάλαιο 4 αναλύουμε το σύστημα μέτρησης της ποιότητας ενός WiFi δικτύου «WiFiMon», το οποίο έχει αναπτυχθεί στα πλαίσια του ευρωπαϊκού έργου GEANT.

Στο Κεφάλαιο 5 προχωράμε στην υλοποίηση σε Docker του συστήματος αυτού.

Στο Κεφάλαιο 6 παρουσιάζονται τα τελικά συμπεράσματα της διπλωματικής αυτής εργασίας.

Στο Κεφάλαιο 7 παρατίθενται προτάσεις για πιθανή μελλοντική εργασία.

Τέλος, στο Κεφάλαιο 8 παρουσιάζονται η βιβλιογραφία και οι δικτυακοί τόποι που αναφέρονται στη διπλωματική αυτή εργασία. Οι αναφορές στη βιβλιογραφία και τους δικτυακούς τόπους ενσωματώνονται στο κείμενο μέσα σε αγκύλες ([]).

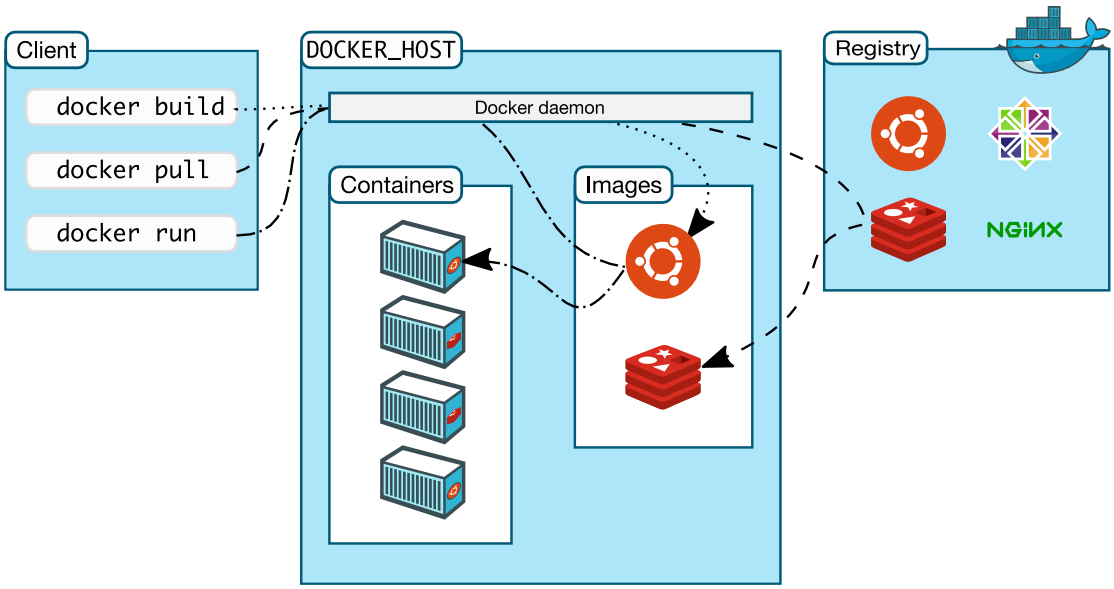
ΚΕΦΑΛΑΙΟ 2: DOCKER

DOCKER

Το Docker [1] είναι λογισμικό ανοιχτού κώδικα το οποίο εκτελεί εικονοποίηση σε επίπεδο λειτουργικού συστήματος, γνωστή και ως “containerization”. Αναπτύσσεται από την εταιρία Docker Inc. και κυκλοφόρησε για πρώτη φορά το 2013. Είναι γραμμένο στην γλώσσα προγραμματισμού Go. Μπορεί να εκτελεστεί σε Windows, Linux και MacOS.

2.1 ΒΑΣΙΚΑ ΣΤΟΙΧΕΙΑ ΤΟΥ DOCKER

Η αρχιτεκτονική του Docker φαίνεται στην Εικόνα 1. Στην συνέχεια παρουσιάζονται μερικά από τα βασικά στοιχεία του Docker.



Εικόνα 1. Αρχιτεκτονική του Docker

- **Docker client**

Το Docker client είναι ο κύριος τρόπος για να αλληλοεπιδράσουμε με το docker. Όταν χρησιμοποιούμε εντολές όπως η docker run, το client στέλνει αυτές τις εντολές στο daemon, το οποίο τις εκτελεί. Οι εντολές που ξεκινούν με “docker” χρησιμοποιούν το docker api.

- **Docker daemon**

Το Docker daemon ακούει για docker api αιτήσεις και διαχειρίζεται αντικείμενα του docker όπως είναι οι εικόνες, τα container και τα δίκτυα. Ένας daemon μπορεί επίσης να επικοινωνήσει με άλλους daemon για να διαχειριστεί υπηρεσίες του docker.

- **Docker image**

Μία docker εικόνα (Docker image) είναι ένα read-only πρότυπο που περιέχει οδηγίες για την δημιουργία ενός Docker container. Συνήθως, κάθε εικόνα βασίζεται σε κάποια άλλη εικόνα με μερικές επιπρόσθετες αλλαγές. Για παράδειγμα, θα μπορούσαμε να βασιστούμε σε μια εικόνα των Ubuntu για να δημιουργήσουμε μια νέα εικόνα η οποία θα εγκαθιστά το postgres καθώς και θα δημιουργεί τους πίνακες που θέλουμε σε αυτό.

Μπορούμε είτε να δημιουργήσουμε δικές μας εικόνες (δημιουργώντας τες με την χρήση ενός Dockerfile) είτε να χρησιμοποιήσουμε εικόνες που έχουν δημιουργήσει άλλοι χρήστες και έχουν ανεβάσει σε κάποιο registry.

- **Docker container**

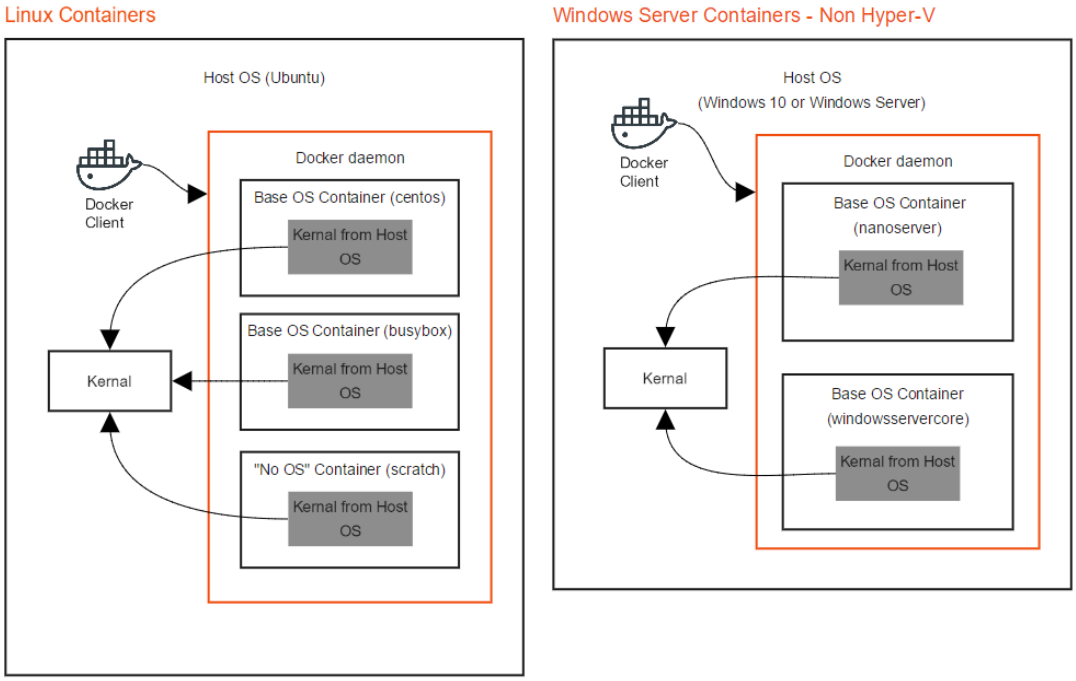
Ένα Docker container είναι ένα απομονωμένο περιβάλλον που εκτελεί εφαρμογές. Μπορούμε να διαχειριστούμε ένα container χρησιμοποιώντας το docker api.

Από προεπιλογή, ένα container είναι απομονωμένο τόσο από τα άλλα container όσο και από την host μηχανή. Μπορούμε όμως να ελέγξουμε πόσο απομονωμένο θα είναι το container.

Ένα container ορίζεται από την docker εικόνα του καθώς και ότι ρυθμίσεις του παρέχουμε κατά την δημιουργία του.

Μπορούμε να δημιουργήσουμε ένα container από μια εικόνα με την εντολή “docker run [OPTIONS] IMAGE-NAME [COMMAND]” π.χ. “docker run -i -t ubuntu /bin/bash”.

Τα container μοιράζονται το kernel με το host λειτουργικό σύστημα (όπως φαίνεται στην Εικόνα 2).



Εικόνα 2. Τα container μοιράζονται το kernel του Host OS

- **Docker Registry**

Ένα Docker Registry περιέχει docker εικόνες. Ένα registry μπορεί να είτε ιδιωτικό είτε δημόσιο. Όταν χρησιμοποιούμε την εντολή docker run ή docker pull, το docker θα τραβήξει τις απαιτούμενες εικόνες από το προεπιλεγμένο registry. Με την εντολή docker push μπορούμε να ανεβάσουμε μια εικόνα στο registry.

Το προεπιλεγμένο απομακρυσμένο registry είναι το docker hub (https://hub.docker.com/) το οποίο είναι ένα δημόσιο registry που διαχειρίζεται η

Docker Inc. και το οποίο μπορεί να χρησιμοποιήσει ο οποιοσδήποτε. Στο docker hub μπορούμε να βρούμε εικόνες για πολλές δημοφιλείς εφαρμογές.

- **Dockerfile**

Ένα Dockerfile είναι ένα αρχείο κειμένου που περιέχει οδηγίες για την δημιουργία μιας docker εικόνας. Το Dockerfile περιέχει όλες τις εντολές που θα τρέχαμε στην γραμμή εντολών προκειμένου να σχηματίσουμε την εικόνα. Με την εντολή docker build δημιουργούμε μια εικόνα από ένα dockerfile.

- **Docker Compose**

Το Compose είναι ένα εργαλείο για να ορίσουμε και να τρέξουμε εφαρμογές στο Docker που χρησιμοποιούν πολλαπλά container. Με το compose, μπορούμε να χρησιμοποιήσουμε ένα YAML αρχείο για να ρυθμίσουμε τις υπηρεσίες της εφαρμογής μας και έπειτα με μόνο μία εντολή να τις εκκινήσουμε όλες μαζί.

- **Docker Service**

Μια υπηρεσία (service) μας επιτρέπει να ορίσουμε το πως θέλουμε να τρέξουμε τα container των εφαρμογών μας σε ένα σμήνος (swarm). Μια υπηρεσία μας επιτρέπει να ορίσουμε την επιθυμητή κατάσταση, όπως το πόσα αντίγραφα (replicas) της εφαρμογής θέλουμε να τρέξουμε συγχρόνως. Από προεπιλογή γίνεται εξισορρόπηση φορτίου (load-balancing) ανάμεσα σε όλους τους κόμβους-εργάτες (worker nodes) του σμήνους. Στο χρήστη η υπηρεσία φαίνεται σαν να είναι μία μόνο εφαρμογή.

Συχνά μια υπηρεσία είναι μια μικρουπηρεσία στο πλαίσιο μιας μεγαλύτερης εφαρμογής. Παραδείγματα υπηρεσιών ίσως περιλαμβάνουν έναν HTTP server, μια βάση δεδομένων ή οποιοδήποτε άλλο είδος εκτελέσιμου προγράμματος θα επιθυμούσαμε να τρέξουμε σε ένα κατανεμημένο περιβάλλον.

- **Swarm mode**

Ένα σμήνος (swarm) αποτελείται από πολλαπλούς Docker hosts που τρέχουν σε swarm mode και λειτουργούν ως managers ή/και εργάτες. Ένας Docker host μπορεί να είναι manager, εργάτης ή και τα δύο.

2.2 ΥΠΟΚΕΙΜΕΝΗ ΤΕΧΝΟΛΟΓΙΑ

- **Χώροι ονομάτων**

Το Docker χρησιμοποιεί μια τεχνολογία που ονομάζεται *χώροι ονομάτων* (*namespaces*) για να παρέχει τον απομονωμένο χώρο εργασίας που ονομάζεται container. Όταν εκκινείτε ένα container, το Docker δημιουργεί ένα σύνολο από χώρους ονομάτων για αυτό το container.

Αυτοί οι χώροι ονομάτων παρέχουν ένα επίπεδο απομόνωσης. Κάθε πλευρά του container τρέχει σε ένα ξεχωριστό χώρο ονομάτων και η πρόσβασή του περιορίζεται σε αυτό το χώρο ονομάτων.
Το Docker Engine χρησιμοποιεί χώρους ονομάτων όπως τα ακόλουθα στο Linux:

- **Ο χώρος ονομάτων `pid`:** Απομόνωση διαδικασιών (PID: Process ID).
- **Ο χώρος ονομάτων `net`:** Διαχείριση διεπαφών δικτύου (NET: Networking).
- **Ο χώρος ονομάτων `ipc`:** Διαχείριση πρόσβασης σε πόρους IPC (IPC: InterProcess Communication).

- **Ο χώρος ονομάτων `mnt`:** Διαχείριση σημείων προσάρτησης συστήματος αρχείων (MNT: Mount).
- **Ο χώρος ονομάτων `uts`:** Απομόνωση του πυρήνα και των αναγνωριστικών έκδοσης. (UTS: Unix Timesharing System).

- **Ομάδες ελέγχου**

Το Docker Engine στο Linux επίσης βασίζεται σε μια άλλη τεχνολογία που ονομάζεται *ομάδες ελέγχου* (`cgroups`). Ένα cgroup περιορίζει μια εφαρμογή σε ένα συγκεκριμένο σύνολο από πόρους. Οι ομάδες ελέγχου επιτρέπουν στο Docker Engine να μοιράζει τους διαθέσιμους πόρους στα container και προαιρετικά να επιβάλλει όρια και περιορισμούς. Για παράδειγμα, μπορεί να περιοριστεί η μνήμη που θα είναι διαθέσιμη σε ένα συγκεκριμένο container.

- **Union συστήματα αρχείων**

Τα Union συστήματα αρχείων, or UnionFS, είναι συστήματα αρχείων που λειτουργούν με τη δημιουργία στρώσεων, καθιστώντας τα πολύ ελαφριά και γρήγορα. Η μηχανή του Docker χρησιμοποιεί το UnionFS για να παρέχει τα δομικά στοιχεία για container. Το Docker Engine μπορεί να χρησιμοποιήσει διαφορετικές εκδοχές του UnionFS, όπως τα AUFS, btrfs, vfs και DeviceMapper.

- **Container format**

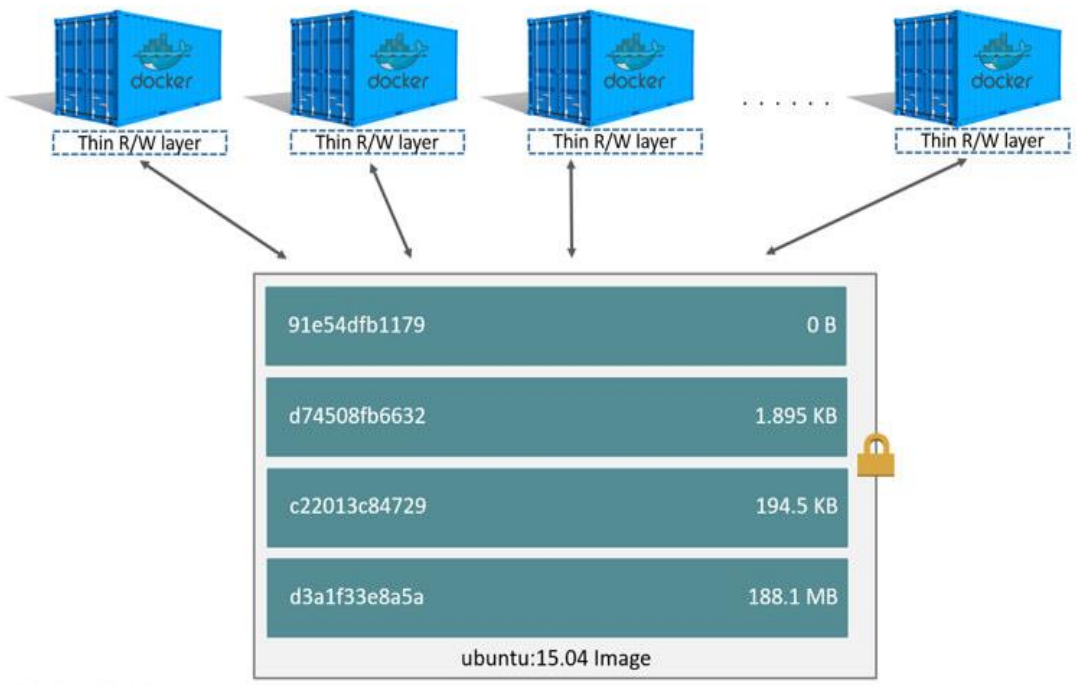
Το Docker Engine συνδυάζει τους χώρους ονομάτων, τις ομάδες ελέγχου και το UnionFS σε ένα περιτύλιγμα που ονομάζεται μορφή του container. Η προεπιλεγμένη μορφή του container είναι το libcontainer.

2.3 IMAGE LAYERS

Οι εικόνες αποτελούνται από μια σειρά από στρώσεις (layers). Κάθε στρώση αντιπροσωπεύει μία οδηγία από το Dockerfile της εικόνας. Σε κάθε στρώση εκτός από την τελευταία επιτρέπετε μόνο ανάγνωση. Για παράδειγμα, στην Εικόνα 3 μπορούμε να δούμε τις στρώσεις από τις οποίες αποτελείται η εικόνα των ubuntu:18.04, τις αντίστοιχες εντολές από το Dockerfile της εικόνας στις οποίες αντιστοιχεί η κάθε στρώση καθώς και το μέγεθος της κάθε στρώσης.

Εικόνα 3. Image layers for Ubuntu:18.04 image

Κάθε στρώση περιέχει μόνο τις διαφορές σε σχέση με την προηγούμενη στρώση. Οι στρώσεις είναι στοιβαγμένες η μία πάνω στην άλλη. Όταν δημιουργούμε ένα νέο container, προστίθεται στην κορυφή μια νέα στρώση στην οποία επιτρέπεται εκτός από την ανάγνωση και η εγγραφή. Αυτή η στρώση συχνά αποκαλείται η «στρώση του container (container layer)”. Όλες οι αλλαγές που γίνονται στο container, όπως η δημιουργία νέων αρχείων, η τροποποίηση ή η διαγραφή υπαρχόντων αρχείων, αποθηκεύονται σε αυτή την στρώση. Όταν το container διαγραφεί, θα διαγραφεί και αυτή η στρώση ενώ η υποκείμενη εικόνα θα παραμένει αναλλοίωτη. Επειδή κάθε container έχει την δική του στρώση και όλες οι αλλαγές αποθηκεύονται μόνο σε αυτή την στρώση, διαφορετικά container μπορούν να χρησιμοποιήσουν συγχρόνως την ίδια εικόνα χωρίς να επηρεάζουν το ένα το άλλο. Ένα τέτοιο παράδειγμα φαίνεται στην Εικόνα 4 όπου 4 διαφορετικά container μοιράζονται την ίδια εικόνα των Ubuntu.



Εικόνα 4. Multiple containers sharing the same image

2.4 ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΤΟΥ DOCKER

Τα containers υπερτερούν της παραδοσιακής εικονοποίησης σε πολλούς τομείς και ειδικά στον τομέα της απόδοσης και της επεκτασιμότητας. Το Docker αποτελεί καλή επιλογή για τα περισσότερα συστήματα υπολογιστικού νέφους, τα οποία χρειάζονται μια εύκολη επιλογή επεκτασιμότητας. Αντί να χρησιμοποιούνται πολλαπλές εικονικές μηχανές, οι οποίες σπαταλούν πολύτιμους πόρους, το Docker μπορεί να εκκινήσει πολύ γρήγορα πολλά περισσότερα container και με πολύ μικρότερη επιβάρυνση πόρων.

Ένα πρόβλημα που μπορεί να επιλύσει το Docker είναι η λεγόμενη «κόλαση εξαρτήσεων» (dependency hell) δηλαδή όταν δύο διαφορετικές εφαρμογές έχουν ως εξάρτηση διαφορετικές εκδόσεις του ίδιου πακέτου λογισμικού ή βιβλιοθήκης, οι οποίες είναι ασύμβατες μεταξύ τους και πιθανόν να μην μπορούν να εγκατασταθούν παράλληλα. Στο Docker κάθε container περιέχει όλες τις εξαρτήσεις της εφαρμογής και επειδή τα container είναι απομονωμένα το ένα από το άλλο, δεν υπάρχει πρόβλημα αν σε διαφορετικά container έχουν εγκατασταθεί διαφορετικές εκδόσεις της ίδιας βιβλιοθήκης [2].

Μερικά από τα ποιο βασικά πλεονεκτήματα του Docker είναι τα ακόλουθα [3], [4]:

- Φορητές Υλοποιήσεις

Καθώς τα container είναι φορητά, οι εφαρμογές μπορούν να ενσωματωθούν σε μια ενιαία μονάδα και μπορούν να αναπτυχθούν σε διάφορα περιβάλλοντα χωρίς να χρειαστεί να γίνουν αλλαγές στο container.

Τα container μπορούν να τρέχουν σε οποιοδήποτε σύστημα Linux, Windows ή Mac, καθώς και σε συστήματα υπολογιστικού νέφους, επιτραπέζιους υπολογιστές, φυσικούς διακομιστές και ούτω καθεξής. Μπορούμε να μεταφέρετε container από το περιβάλλον της επιφάνειας εργασίας μας στο νέφος και πάλι πίσω εύκολα και γρήγορα.

- Ταχεία παράδοση

Η ροή εργασίας των container καθιστά εύκολη τη συνεργασία ανάμεσα σε προγραμματιστές, διαχειριστές συστημάτων και ομάδων QA. Λόγω της τυπικής μορφής του container, μόνο οι προγραμματιστές χρειάζονται να ανησυχούν για τις εφαρμογές που τρέχουν μέσα σε αυτό ενώ οι διαχειριστές του συστήματος χρειάζονται μόνο να ανησυχούν για την τοποθέτηση του container στους διακομιστές. Αυτός ο καλά διαχωρισμένος τρόπος διαχείρισης των container οδηγεί σε γρηγορότερη ανάπτυξη και παράδοση της εφαρμογής. Επιπλέον η δημιουργία νέων container είναι πολύ γρήγορη επειδή τα container είναι πολύ ελαφριά, Αυτό με τη σειρά του μειώνει τον χρόνο για τη δοκιμή και την ανάπτυξη της εφαρμογής.

- Κλιμάκωση

Η κλιμάκωση των container προς τα επάνω και κάτω είναι απίστευτα γρήγορη. Μπορούμε εύκολα και γρήγορα να κλιμακώσουμε τα container από ένα σε μερικές εκατοντάδες και να τα ξανά μειώσουμε αργότερα όταν πια δεν τα χρειαζόμαστε. Έτσι τα container είναι ιδανικά για εφαρμογές έχουν σχεδιαστεί και κατασκευαστεί για πλατφόρμες υπολογιστικού νέφους.

- Καλύτερη απόδοση

Μπορούμε να έχουμε πολλά περισσότερα container σε σύγκριση με τις εικονικές μηχανές. Αφού τα container δεν χρειάζονται να χρησιμοποιήσουν κάποιο hypervisor, μπορούν να αξιοποιήσουν καλύτερα τους διαθέσιμους πόρους. Επειδή τα container δεν χρησιμοποιούν ένα πλήρες λειτουργικό σύστημα, οι απαιτήσεις σε πόρους είναι μικρότερες σε σύγκριση με τις εικονικές μηχανές.

- Απλό στη χρήση

Το Docker είναι εξαιρετικά απλό στη χρήση και εύκολο στην εκμάθηση.

2.5 ΣΥΓΚΡΙΣΗ CONTAINER – ΕΙΚΟΝΙΚΩΝ ΜΗΧΑΝΩΝ

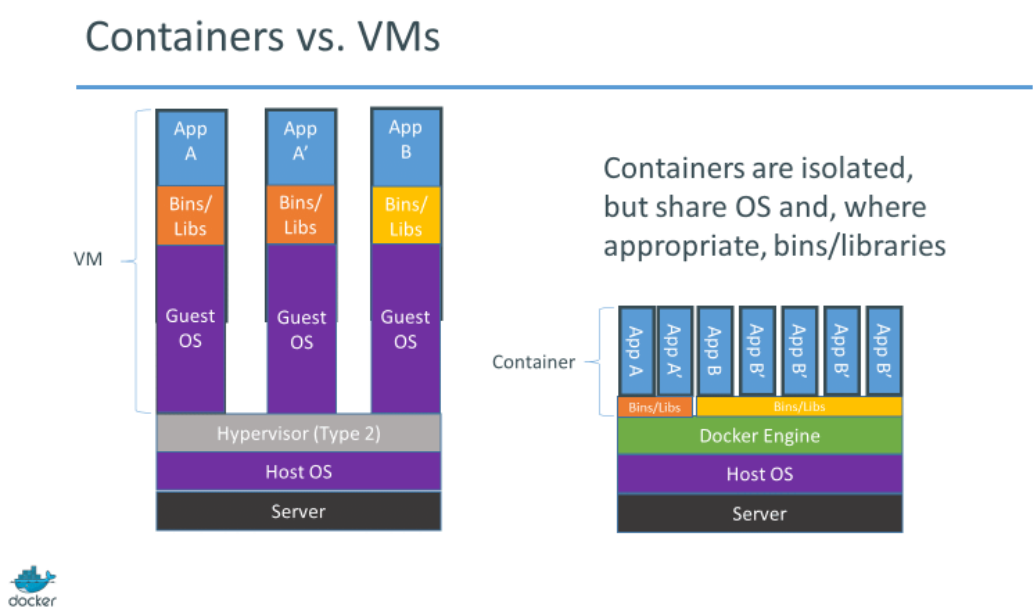
Στον Πίνακας 1 γίνεται μια σύγκριση ανάμεσα σε εικονικές μηχανές και container σε πολλαπλούς παράγοντες όπως η απόδοση, η ασφάλεια, ο χρόνος εκκίνησης και ο χώρος αποθήκευσης [5].

Παράμετρος	Εικονικές μηχανές	Container
Guest OS	Κάθε εικονική μηχανή τρέχει σε εικονικό υλικό και το kernel φορτώνεται στην δικιά του περιοχή μνήμης.	Όλα τα container μοιράζονται το ίδιο kernel. Το kernel φορτώνεται στην φυσική μνήμη.
Επικοινωνία	Μέσω συσκευών Ethernet.	Στάνταρ IPC μηχανισμοί όπως σήματα, σωληνώσεις (pipes), υποδοχές (sockets) κ.τ.λ.
Ασφάλεια	Ανάλογα με την υλοποίηση του hypervisor.	Ο υποχρεωτικός έλεγχος πρόσβασης μπορεί να χρησιμοποιηθεί.

Απόδοση	Οι εικονικές μηχανές υποφέρουν από μια μικρή επιβάρυνση καθώς οι εντολές μηχανής μεταφράζονται από το Guest στο Host λειτουργικό σύστημα.	Τα container έχουν σχεδόν την ίδια απόδοση που έχει το υποκείμενο Host λειτουργικό σύστημα.
Απομόνωση	Ο διαμοιρασμός βιβλιοθηκών, αρχείων, κ.τ.λ. ανάμεσα σε εικονικές μηχανές ή ανάμεσα στην εικονική μηχανή και το host λειτουργικό σύστημα δεν είναι δυνατός.	Τα αρχεία και οι βιβλιοθήκες μπορούν εύκολα να διαμοιραστούν ανάμεσα στα container ή μεταξύ του container και του host λειτουργικού συστήματος.
Χρόνος εκκίνησης	Η εκκίνηση στις εικονικές μηχανές χρειάζεται λίγα λεπτά.	Τα Containers μπορούν να εκκινηθούν σε λίγα δευτερόλεπτα δηλαδή σε σημαντικά λιγότερο σε σχέση με τις εικονικές μηχανές
Χώρος αποθήκευσης	Οι εικονικές μηχανές χρειάζονται αρκετά περισσότερο αποθηκευτικό χώρο αφού ολόκληρο το kernel του λειτουργικού συστήματος καθώς και τα σχετικά προγράμματα χρειάζονται να εγκατασταθούν.	Τα container χρειάζονται λιγότερο αποθηκευτικό χώρο αφού το βασικό λειτουργικό σύστημα είναι κοινόχρηστο.

Πίνακας 1. Σύγκριση Εικονικών μηχανών - Container

Στην Εικόνα 5 μπορούμε να δούμε μερικές από τις σημαντικές διαφορές ανάμεσα σε εικονικές μηχανές και container. Όπως φαίνεται στην εικόνα, το Docker, σε αντίθεση με τις εικονικές μηχανές, δεν χρησιμοποιεί κάποιο hypervisor, κάτι που μειώνει σημαντικά το overhead. Κάθε εικονική μηχανή έχει ένα πλήρες αντίγραφο του λειτουργικού συστήματος καθώς και των βιβλιοθηκών που χρησιμοποιεί. Σε αντίθεση στο Docker το kernel μοιράζεται με το host λειτουργικό σύστημα και τα αλλά container. Επιπροσθέτως τα container μοιράζονται όσες βιβλιοθήκες είναι κοινές. Αυτό έχει ως αποτέλεσμα τα container να έχουν σημαντικά μικρότερο μέγεθος σε σχέση με τις εικονικές μηχανές.



Εικόνα 5. Containers vs VMs

ΚΕΦΑΛΑΙΟ 3: ΣΧΕΤΙΚΕΣ ΕΡΓΑΣΙΕΣ

ΣΧΕΤΙΚΕΣ ΕΡΓΑΣΙΕΣ

Σε αυτό το κεφάλαιο παρατίθενται εργασίες, που έχουν γίνει στο παρελθόν , σχετικές με container και συστήματα containerization, με ιδιαίτερη έμφαση στο σύστημα containerization Docker.

Στην εργασία [2] εξετάζεται η χρήση του Docker για αναπαραγώγιμη επιστημονική έρευνα. Αρχικά διερευνώνται συνήθεις λόγοι που ο κώδικας που αναπτύχθηκε για ένα ερευνητικό έργο δεν μπορεί να εκτελεστεί ή να επεκταθεί με επιτυχία από τους επόμενους ερευνητές. Στην συνέχεια γίνεται μια αναθεώρηση των τρεχουσών προσεγγίσεων σε αυτά τα θέματα, συμπεριλαμβανομένων των εικονικών μηχανών και των συστημάτων ροής εργασίας, καθώς και τους περιορισμούς τους. Έπειτα, εξετάζεται ο τρόπος με τον οποίο το Docker συνδυάζει αρκετές περιοχές από την έρευνα συστημάτων - όπως η εικονικοποίηση του λειτουργικού συστήματος, η φορητότητα σε πολλαπλές πλατφόρμες, τα αρθρωτά επαναχρησιμοποιήσιμα στοιχεία, η ΄χρήση εκδόσεων και η φιλοσοφία “DevOps”, για την αντιμετώπιση αυτών των προκλήσεων.

Στην εργασία [3] γίνεται μια σύγκριση της απόδοσης ανάμεσα σε container και εικονικές μηχανές και καταλήγει ότι τα container σημείωσαν καλύτερες επιδόσεις από τις εικονικές μηχανές, τόσο στον τομέα της απόδοσης όσο και στον τομέα την επεκτασιμότητας.

Στην εργασία [4] εξετάζονται τα πλεονεκτήματα του Docker. Η μελέτη αυτή γίνεται με την μορφή βιβλιογραφικής αναθεώρησης. Καταλήγει στο ότι το Docker είναι μια πολλά υποσχόμενη τεχνολογία, η οποία έχει πολλά πλεονεκτήματα στα πεδία της ευχρηστίας, των επιδόσεων και της ασφάλειας. Τα container έχουν καλύτερη χρηστικότητα, απόδοση και επεκτασιμότητα από την παραδοσιακή εικονοποίηση, καθώς τα container χρησιμοποιούν τους πόρους του συστήματος πολύ καλύτερα αφού έχουν σχεδόν μηδενικό overhead καθώς και μεγαλύτερη πυκνότητα (density). Επίσης έχουν μικρότερο χρόνο εκκίνησης σε σχέση με εικονικές μηχανές.

Στην εργασία [5] διερευνάτε ο τρόπος με τον οποίο οι πάροχοι PaaS (Platform as a Service) χρησιμοποιούν container ως μέσο φιλοξενίας εφαρμογών. Αρχικά γίνεται μια εισαγωγή στις περιπτώσεις χρήσης PaaS και της τρέχουσας υιοθέτησης αρχιτεκτονικών PaaS βασισμένων σε container από τους υπάρχοντες προμηθευτές. Στην συνέχεια εξετάζονται διάφορες υλοποιήσεις των container (Linux Containers, Docker, Warden Container, lmctfy, OpenVZ) ως προς τις ομοιότητες και τις διαφορές τους. Καταλήγει ότι όλες οι υλοποιήσεις έχουν τα θετικά και τα αρνητικά τους. Προτείνει το Docker ως την πιο σχετική επιλογή για PaaS ενώ θεωρεί το OpenVZ ως την καλύτερη επιλογή από άποψη ασφάλειας τονίζοντας βέβαια ότι το OpenVZ χρειάζεται ένα τροποποιημένο kernel.

Στην εργασία [6] αξιολογείται τη χρήση του Docker ως μια πλατφόρμα για edge computing. Τα 4 βασικά κριτήρια που εξετάζονται είναι:

1) ανάπτυξη και τερματισμός
2) διαχείριση πόρων και υπηρεσιών
3) ανοχή σφάλματων
4) caching

Τα κριτήρια αυτά εξετάζονται τόσο ως τεχνική αξιολόγηση του Docker όσο και με την διεξαγωγή πειράματος. Οι συγγραφείς συμπεραίνουν ότι το Docker πληροί αυτά τα κριτήρια και κατά συνέπεια είναι ένας βιώσιμος υποψήφιος ως πλατφόρμα για edge computing συστήματα. Το Docker παρέχει γρηγορότερη ανάπτυξη, ελαστικότητα και καλύτερη απόδοση σε σχέση με ένα edge computing σύστημα βασισμένο σε εικονικές μηχανές, κάτι που το καθιστά ελκυστική επιλογή.

Στην εργασία [7] υλοποιείται στο Docker μια πλατφόρμα ανάλυσης υπολογιστικού νέφους. Η πλατφόρμα αυτή χρησιμοποιεί το Redis ως βάση δεδομένων και δυναμικές βιβλιοθήκες που έχουν παραχθεί από το flex και το bison για την ανάλυση.

Στην εργασία [8] αξιολογείται η απόδοση των container του Docker με βάση την απόδοση του host συστήματος. Οι επιδόσεις που βασίζονται στο σύστημα αρχείων αξιολογούνται χρησιμοποιώντας το εργαλείο Bonnie ++. Οι επιδόσεις που βασίζονται σε άλλους πόρους του συστήματος όπως η χρήση της CPU, η αξιοποίηση της μνήμης κ.λπ. αξιολογούνται με τη χρήση κώδικα αξιολόγησης που αναπτύχθηκε σε Python και χρησιμοποιεί την βιβλιοθήκη psutil. Τα αποτελέσματα των μετρήσεων δείχνουν ότι η απόδοση του Docker είναι πολύ καλή και είναι συγκρίσιμη με το να τρέχαμε τα προγράμματα απευθείας στο host λειτουργικό σύστημα.

Στην εργασία [9] παρουσιάζεται το BioShaDock, το οποίο είναι ένα εξειδικευμένο Docker Registry το οποίο εστιάζει στον τομέα της βιοπληροφορικής. Το BioShaDock παρέχει ένα τοπικό και πλήρως ελεγχόμενο περιβάλλον για την κατασκευή και δημοσίευση βιοπληροφορικού λογισμικού ως φορητές εικόνες Docker. Προσφέρει ορισμένες βελτιώσεις σε σχέση με το βασικό Docker Registry σχετικά με την πιστοποίηση ταυτότητας και τη διαχείριση αδειών, οι οποίες επιτρέπουν την ενσωμάτωσή του σε υπάρχουσες υποδομές βιοπληροφορικής, όπως οι υπολογιστικές πλατφόρμες. Τα μεταδεδομένα που σχετίζονται με τις καταχωρημένες εικόνες έχουν να κάνουν με τον τομέα την βιοπληροφορικής, περιλαμβάνοντας για παράδειγμα έννοιες που ορίζονται στην οντολογία EDAM, ένα κοινό και δομημένο λεξιλόγιο των κοινώς χρησιμοποιούμενων όρων στη βιοπληροφορική. Το μητρώο περιλαμβάνει επίσης ετικέτες που ορίζονται από το χρήστη για να διευκολύνουν την ανίχνευσή, καθώς και μια σύνδεση με την περιγραφή του εργαλείου στο ELIXIR registry, αν υπάρχει ήδη. Εάν δεν υπάρχει, το μητρώο BioShaDock θα συγχρονιστεί με το μητρώο Elixir για να δημιουργήσει μια νέα περιγραφή σε αυτό, με βάση τα μεταδεδομένα της εισόδου του στο BioShaDock. Αυτή η σύνδεση θα βοηθήσει τους χρήστες να αποκτήσουν περισσότερες πληροφορίες σχετικά με το εργαλείο, όπως οι EDAM λειτουργίες του και οι τύποι εισόδου και εξόδου του. Αυτό επιτρέπει την ενσωμάτωση με το ELIXIR Tools και το Data Services Registry, παρέχοντας έτσι την κατάλληλη ορατότητα τέτοιων εικόνων στην κοινότητα βιοπληροφορικής.

Στην εργασία [10] γίνεται μια εμπειρική μελέτη μεγάλης κλίμακας όπου αναλύεται το οικοσύστημα, οι πτυχές της ποιότητας και η εξελικτική συμπεριφορά των container του Docker στο Github. Η μελέτη βασίζεται σε 70197 Dockerfiles από 38079 έργα, που είναι ολόκληρος ο πληθυσμός των έργων την στιγμή που εκπονήθηκε η μελέτη (Οκτώβριος 2016). Βρίσκει ότι τα περισσότερα container κληρονομούν την υποδομή από βαρύτερα λειτουργικά συστήματα, πιθανότατα για λόγους ευκολίας, το οποίο όμως αποτυγχάνει τον σκοπό των container για τη μείωση του αποτυπώματος τους. Διαπιστώνει επίσης ότι το 28,6% των ζητημάτων ποιότητας (όπως υποδεικνύεται από το Docker Linter [11]) προκύπτουν από την έλλειψη καρφώματος των εξαρτήσεων σε

κάποια συγκεκριμένη έκδοση. Περαιτέρω, το 34% των Dockerfiles από αντιπροσωπευτικό δείγμα 560 έργων δεν μπόρεσε να χτιστεί με επιτυχία. Προτείνει ότι η ενσωμάτωση ελέγχων ποιότητας στο Docker κατά το χτίσιμο των εικόνων, όπως εμφάνιση προειδοποιήσεων για το κάρφωμα των εξαρτήσεων σε συγκεκριμένες εκδόσεις, θα οδηγούσε σε ποιο αναπαραγωγίσιμα Dockerfile στο μέλλον. Τέλος, παρατηρεί ότι τα Dockerfile δεν αλλάζουν συχνά, με μέσο όρο 3,11 έως 5,81 αναθεωρήσεις ετησίως, οι περισσότερες από τις οποίες είχαν να κάνουν με τις εξαρτήσεις.

Στην εργασία [12] παρουσιάζεται το GUIdock. Το GUIdock επιτρέπει την εύκολη διανομή μιας εφαρμογής βιολογίας συστημάτων μαζί με το περιβάλλον γραφικών της. Το GUIdock χρησιμοποιεί το Docker και διαμορφώνει μια κοινή X Windows (X11) γραφική διεπαφή σε πλατφόρμες Linux, Macintosh και Windows. Ως παράδειγμα, παρουσιάζεται ένα πακέτο Docker που περιέχει μια Bioconductor εφαρμογή για τη εξαγωγή συμπερασμάτων από γονιδιακά δίκτυα, η οποία ονομάζεται networkBMA και είναι γραμμένη σε R και C. Το πακέτο περιλαμβάνει επίσης το Cytoscape, μια πλατφόρμα βασισμένη σε java με γραφικό περιβάλλον χρήστη για την απεικόνιση και ανάλυση γονιδιακών δικτύων και την εφαρμογή CyNetworkBMA, μια Cytoscape εφαρμογή που επιτρέπει τη χρήση του networkBMA μέσω φιλικής προς το χρήστη διεπαφής Cytoscape.

Στην εργασία [13], οι συγγραφείς προτείνουν μια τεχνική καταγραφής και επανάληψης για το live migration των container του docker. Αυτή η τεχνική χωρίζεται σε 3 βασικά στάδια:

1. Αρχικά αντιγράφεται η εικόνα του container και εκκινείτε στο νέο container. Η παραπάνω διαδικασία δεν είναι στιγμιαία και κατά συνέπεια μπορεί να υπάρχουν αλλαγές στο πηγαίο container οι οποίες δεν έχουν καταγραφεί στην εικόνα. Για αυτό το λόγο κρατείται ένα αρχείο καταγραφής που περιέχει όλες τις αλλαγές που συνέβησαν στο container και οι οποίες δεν έχουν καταγραφεί στην αντιγραμμένη εικόνα.
2. Σε αυτό το στάδιο, το αρχείο καταγραφής μεταφέρεται στο νέο container όπου αναπαράγονται οι αλλαγές. Κατά την διάρκεια αυτής της διαδικασίας νέες αλλαγές μπορεί να προκύψουν στο πηγαίο container, οι οποίες καταγράφονται πάλι σε αρχεία καταγραφής ώστε να αναπαραχθούν από το νέο container αργότερα. Η διαδικασία αυτή επαναλαμβάνεται μέχρι το μέγεθος του αρχείου καταγραφής να είναι λιγότερο από ένα προκαθορισμένο όριο.
3. Το πηγαίο container τερματίζεται και το τελευταίο αρχείο καταγραφής μεταφέρεται στο νέο container όπου και αναπαράγονται οι αλλαγές.

Με αυτή την μέθοδο ο χρόνος εκτός λειτουργίας της υπηρεσίας μειώθηκε σημαντικά. Σε σύγκριση με την τεχνική για το live migration εικονικών μηχανών, ο χρόνος διακοπής της υπηρεσίας μειώθηκε κατά 65%, 55% και 44% σε τρία διαφορετικά σενάρια ενώ ο συνολικός χρόνος μειώθηκε κατά 27%, 47% και 38% αντίστοιχα.

Στην εργασία [14] παρουσιάζεται το ELASTICDOCKER. Το ELASTICDOCKER είναι ένας ελεγκτής κάθετης ελαστικότητας το οποίο αυξάνει ή μειώνει δυναμικά τους πόρους του Docker ανάλογα με τον φόρτο εργασίας. Αν δεν υπάρχουν αρκετοί διαθέσιμοι πόροι στο host μηχάνημα, τότε πραγματοποιείται live migration του container σε κάποιο άλλο μηχάνημα. Σε σύγκριση με το Kubernetes το οποίο πραγματοποιεί οριζόντια ελαστικότητα, το ELASTICDOCKER είχε καλύτερα αποτελέσματα στην ελαστικότητα κατά 37,63%.

Στην εργασία [15] οι συγγραφείς σχεδιάζουν ένα Application Oriented Docker Container (AODC) framework για την διαχείριση πόρων σε κέντρα δεδομένων (data centers). Το πρόβλημα της κατανομής των πόρων αντιμετωπίζεται ως ένα πρόβλημα βελτιστοποίησης όπου λαμβάνονται υπόψη τα χαρακτηριστικά του Docker και των container. Τα νέα χαρακτηριστικά του AODC framework μπορούν να συνοψιστούν ως εξής:

a) Ο αριθμός και η χωρητικότητα των container προσδιορίζονται με βάση όχι μόνο τις απαιτήσεις της εφαρμογής αλλά και τους διαθέσιμους πόρους του κάθε μηχανήματος.

b) Το κόστος εγκατάστασης του κάθε container λαμβάνει υπόψη την σχέση ανάμεσα στις ήδη διαθέσιμες βιβλιοθήκες στο μηχάνημα και τις βιβλιοθήκες που χρησιμοποιεί η εφαρμογή.

c) Οι λειτουργίες της διαχείρισης πόρων και της εκτέλεσης της εφαρμογής αποσυνδέονται μεταξύ τους και η διαχείριση πόρων εκτελείται με κατανεμημένο τρόπο.

Στην συνέχεια συγκρίνουν το AODC με άλλους αλγόριθμους για την τοποθέτηση εικονικών μηχανών (Optimal-VM, Best Fit-VM, Worst Fit-VM) και παρατηρούν ότι το AODC παρουσιάζει καλυτέρα αποτελέσματα.

Στην εργασία [16] παρουσιάζεται το APAC . Το APAC είναι ακρωνύμιο για το Automatic Programming Assignment Checker (αυτόματος ελεγκτής προγραμματιστικών εργασιών). Κύριος σκοπός αυτού του συστήματος είναι η δημιουργία ενός αυτόνομου και ευέλικτου συστήματος για την αυτόματη βαθμολόγηση προγραμματιστικών εργασιών χρησιμοποιώντας ένα απομονωμένο περιβάλλον δοκιμών. Το APAC είναι μια Java EE εφαρμογή χτισμένη γύρω από την πλατφόρμα του Docker. Το Docker χρησιμοποιήθηκε ώστε να απομονωθεί το περιβάλλον δοκιμών κάτι που κρίθηκε απαραίτητο καθώς ο κώδικας των μαθητών/φοιτητών μπορεί να είναι κακόβουλος ή/και επικίνδυνος.

Στην εργασία [17] , οι συγγραφείς σχεδιάζουν ένα framework για το πρόβλημα της αναζήτησης ομοιότητας στα μεγάλα σύνολα δεδομένων στο κατανεμημένο περιβάλλον. Το framework αυτό χρησιμοποιεί το VP-Tree αλγόριθμο , ο οποίος ενσωματώνεται στην κορυφή του framework MapReduce για την επίτευξη καλής απόδοσης καθώς και να πληρούνται οι απαιτήσεις επεκτασιμότητας και ανοχής σφάλματος για το σύστημα καθώς αυξάνονται τα δεδομένα. Ο χρόνος εκτέλεσης της εφαρμογής συγκρίθηκε σε Docker container και σε εικονική μηχανή. Η εφαρμογή παρουσίασε καλύτερη απόδοση στο Docker.

Στην εργασία [18] αναπτύσσεται μια πλατφόρμα ελεγκτή SDN (software-defined networking) στο Docker για την υλοποίηση εικονοποίησης δικτύου. Μέσω της απομονωμένης πλατφόρμας ελεγκτή SDN μεταξύ του εικονικού και του φυσικού δικτύου, οι λειτουργίες του λογικού δικτύου αποσυνδέονται από τις φυσικές συσκευές δικτύου για να διευκολύνουν την ολοκλήρωση και την επέκταση των πόρων του δικτύου. Με την χρήση του Docker επιτυγχάνεται χαμηλότερο κόστος συστήματος, καλή κινητικότητα, ταχεία ανάπτυξη και καλή επεκτασιμότητα.

Στην εργασία [19] αναπτύσσεται στο Docker ένα framework και μια μέθοδος για την υπολογιστική εκφόρτωση (computational offloading) υπολογισμών φορητών συσκευών σε συστήματα υπολογιστικού νέφους με βάση την κατανάλωση ενέργειας.

Όπως αναμένεται, αυτό είχε σαν αποτέλεσμα την μείωση της υπολογιστικής και ενεργειακής κατανάλωσης της φορητής συσκευής.

ΚΕΦΑΛΑΙΟ 4: WIFIMON

WiFiMon

4.1 Τι είναι το WiFiMon

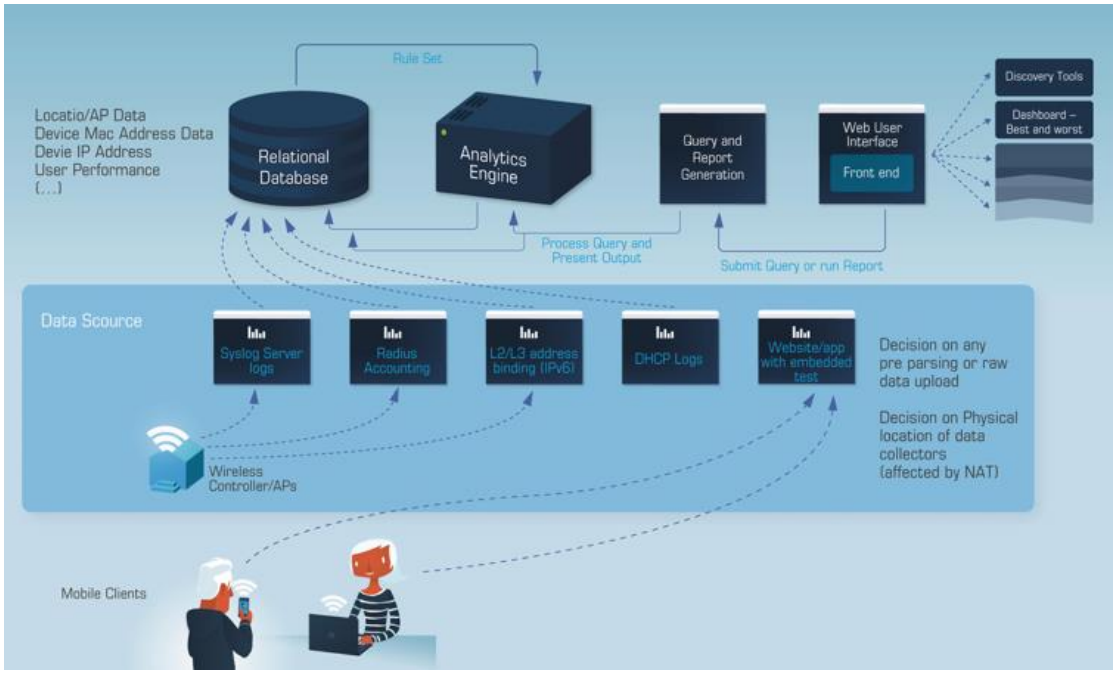
Το WiFiMon [20] αναπτύσσεται από την GEANT. Η αρχική έκδοση του προγράμματος χρηματοδοτήθηκε από το SA3-T3 του GN4-1 ενώ η τωρινή ανάπτυξη του χρηματοδοτείται από SA3-T5 του GN4-2. Έχει την δυνατότητα μέτρησης, καταγραφής και εξαγωγής στατιστικών σχετικά με την ποιότητα των ασύρματων WiFi δικτύων, όπως ακριβώς την βιώνουν οι χρήστες, καλύπτοντας έτσι το κενό άλλων εργαλείων που δεν μπορούν να χαρακτηρίσουν την ποιότητα ενός WiFi δικτύου με αυτόν τον τρόπο. Οι μετρήσεις πυροδοτούνται από τους τελικούς χρήστες όταν επισκέπτονται σελίδες στις οποίες έχει ενσωματωθεί ο κατάλληλος JavaScript κώδικας και καταγράφονται, μέσω crowdsourcing τεχνικών, χωρίς την παρέμβαση των τελικών χρηστών. Έτσι, το εργαλείο WiFiMon έχει ως κύριο στόχο την παροχή πληροφοριών στους διαχειριστές ασύρματων WiFi δικτύων σχετικά με την συνολική ποιότητα των υπηρεσιών του δικτύου όπως την αντιλαμβάνονται οι τελικοί χρήστες του δικτύου.

Αυτό μας επιτρέπει να εξάγουμε συμπεράσματα όπως:

- Αν πολλοί διαφορετικοί χρήστες παρουσιάζουν κακή απόδοση σε μια συγκεκριμένη περιοχή, τότε πιθανότατα υπάρχει πρόβλημα σε αυτή την περιοχή
- Αν μια συγκεκριμένη συσκευή παρουσιάζει κακή απόδοση σε διαφορετικές τοποθεσίες, τότε πιθανότατα υπάρχει πρόβλημα με αυτή την συσκευή.

4.2 Δομικά στοιχεία του WiFiMon

Τα δομικά στοιχεία του WiFiMon φαίνονται στην Εικόνα 6. Τα στοιχεία αυτά θα τα αναλύσουμε στην συνέχεια.



Εικόνα 6. Δομικά στοιχεία του WiFiMon

4.2.1 Πηγή δεδομένων

Στο επίπεδο πηγής δεδομένων:

1. Εξάγονται δεδομένα από συλλέκτες πηγών δεδομένων όπως τα Syslog, RADIUS [23], τα αρχεία καταγραφής DHCP και την δέσμευση διεύθυνσης L2/L3 (IPV6).
2. Παράγονται πληροφορίες μέσω ιστοσελίδων με ενσωματωμένες διαδικασίες δοκιμών, κυρίως μέσω JavaScript κώδικα που έχει ενσωματωθεί στον κώδικα της σελίδας και επιτρέπει την διεξαγωγή δοκιμών χωρίς την αλληλεπίδραση των χρηστών. Για την διεξαγωγή των δοκιμών χρησιμοποιούνται τα εργαλεία ανοιχτού κώδικα Boomerang [21] και NetTest [22]. Οι πληροφορίες που εξάγονται περιλαμβάνουν δεδομένα απόδοσης όπως η ταχύτητα λήψης και μεταφόρτωσης εικόνων διαφορετικών μεγεθών και ο χρόνος round-trip (RTT) μέσω του ping, όπως το βιώνει ο τελικός χρήστης.

Τα αποτελέσματα των μετρήσεων στέλνονται στα WiFiMon processor και/ή WiFiMon secure processor, τα οποία αναλαμβάνουν να τα αποθηκεύσουν στην βάση δεδομένων. Και το processor και το secure processor είναι Spring Boot web εφαρμογές με τον ίδιο σκοπό και λειτουργικότητα. Οι κύριες διαφορές τους είναι οι ακόλουθες:

- Το processor υποστηρίζει HTTP connectors ενώ το secure processor HTTPS connectors.
- Το processor από προεπιλογή τρέχει στην θύρα 9000, ενώ το secure processor τρέχει στην θύρα 8443.
- Όταν ο JavaScript κώδικας εισάγετε σε ασφαλής (SSL) ιστοσελίδες, θα πρέπει να χρησιμοποιείται τον WiFiMon secure processor ώστε να αποφεύγονται σφάλματα μικτού περιεχομένου.
- Το secure processor χρειάζεται τα κλειδιά και τα πιστοποιητικά του server να συμπεριληφθούν σε ένα Java KeyStore.

4.2.2 Σχεσιακή βάση δεδομένων

Τα δεδομένα που παράχθηκαν στο προηγούμενο επίπεδο (επίπεδο πηγής δεδομένων) αποθηκεύονται στην σχεσιακή βάση δεδομένων. Το WiFiMon χρησιμοποιεί δύο βάσεις δεδομένων για να αποθηκεύσει τα δεδομένα, μια βάση δεδομένων PostgreSQL [23] και μία Elasticsearch [24].

- Η PostgreSQL βάση δεδομένων χρησιμοποιείται για την αποθήκευση των καταχωρημένων υποδικτύων, των εγγεγραμμένων χρηστών του GUI, των σημείων πρόσβασης καθώς και αρκετών ρυθμίσεων του WiFiMon GUI (όπως οι ρυθμίσεις ιδιωτικότητας, δικαιώματα οπτικοποίησης, κτλ.).
- Το WiFiMon χρησιμοποιεί το Elasticsearch για να αποθηκεύσει τα αποτελέσματα των μετρήσεων που διεξάχθηκαν μέσω ιστοσελίδων καθώς και των δεδομένων που εξάχθηκαν από τους συλλέκτες πηγών δεδομένων (π.χ. RADIUS).

4.2.3 Μηχανή ανάλυσης

Η μηχανή ανάλυσης είναι το μπλοκ αρχιτεκτονικής που είναι υπεύθυνο για την εξέταση / ανάλυση των πρωτογενών δεδομένων της σχεσιακής βάσης δεδομένων και την προετοιμασία των δεδομένων αναφοράς. Έτσι, η κύρια λειτουργία της μηχανής ανάλυσης είναι να ταξινομεί τα πρωτογενή δεδομένα που συλλέγονται, να τα αναλύει και να παράγει οπτικοποιήσεις μέσω εργαλείων που επιτρέπουν την επίγνωση της απόδοσης του ασύρματου δικτύου. Στην τωρινή υλοποίηση χρησιμοποιείται το Kibana [25] για να δημιουργήσει χρονοσειρές και πίνακες παρακολούθησης.

4.2.4 Παραγωγή ερωτημάτων και αναφοράς

Ο παραγωγός ερωτημάτων και αναφοράς χρησιμοποιείται για την λήψη συγκεκριμένων πληροφοριών από την σχεσιακή βάση δεδομένων και την μηχανή ανάλυσης. Ο κύριος σκοπός του είναι (1) η αναζήτηση χρήσιμων πληροφοριών από αυτά τα δύο αρχιτεκτονικά μπλοκ και (2) η αποστολή αυτών των πληροφοριών με την μορφή αναφορών ή επιλογών οπτικοποίησης στη διαδικτυακή διεπαφή του χρήστη.

4.2.5 Διαδικτυακή διεπαφή χρήστη

Τα διαθέσιμα δεδομένα από την σχεσιακή βάση δεδομένων και την μηχανή ανάλυσης είναι προσβάσιμα μέσω της διαδικτυακής διεπαφής χρήστη του διαχειριστή δικτύου. Η διεπαφή αυτή είναι μια Spring Boot [26] διαδικτυακή εφαρμογή η οποία χρησιμοποιεί το Hibernate [27] για την υποβολή ερωτημάτων. Επιτρέπει την αναζήτηση των δεδομένων και κατά συνέπεια τον έλεγχο της κατάστασης του ασύρματου δικτύου. Η διεπαφή αυτή επιτρέπει την προβολή των συλλεγμένων δεδομένων και προσφέρει επιλογές οπτικοποίησης πραγματικού χρόνου όπως:

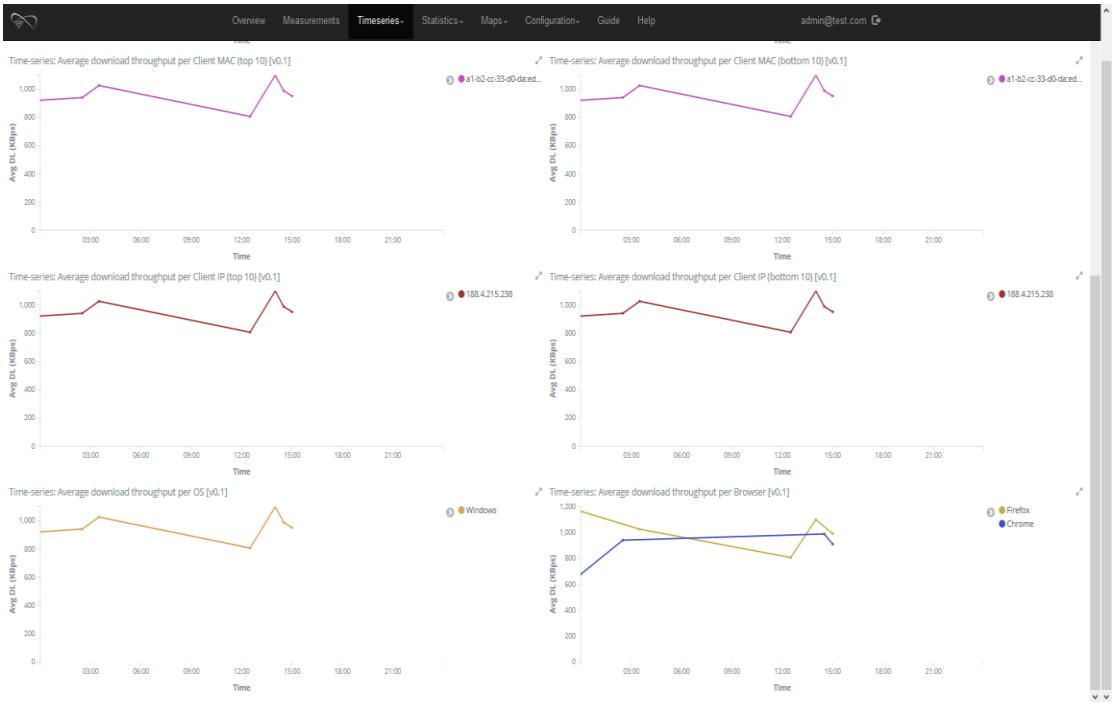
- Μετρήσεις για μια συγκεκριμένη χρονική περίοδο.
- Μετρήσεις από ένα συγκεκριμένο WAP.
- Μετρήσεις για συγκεκριμένες IP ή χρήστες.
- Την προβολή των μέγιστων, ελάχιστων και μέσων τιμών της ταχύτητας λήψης και μεταφόρτωσης.
- Αποτελέσματα μετρήσεων με βάση το ποιο πρόγραμμα περιήγησης ή ποιο λειτουργικό σύστημα χρησιμοποιήθηκε.
- Την προβολή μη φυσιολογικών μετρήσεων.
- Προβολή των πέντε τοποθεσιών όπου εμφανίζονται οι καλύτερες ρυθμίσεις.

Παρακάτω παρουσιάζονται ενδεικτικά μερικά στιγμιότυπα από την διεπαφή χρήστη του WiFiMon.

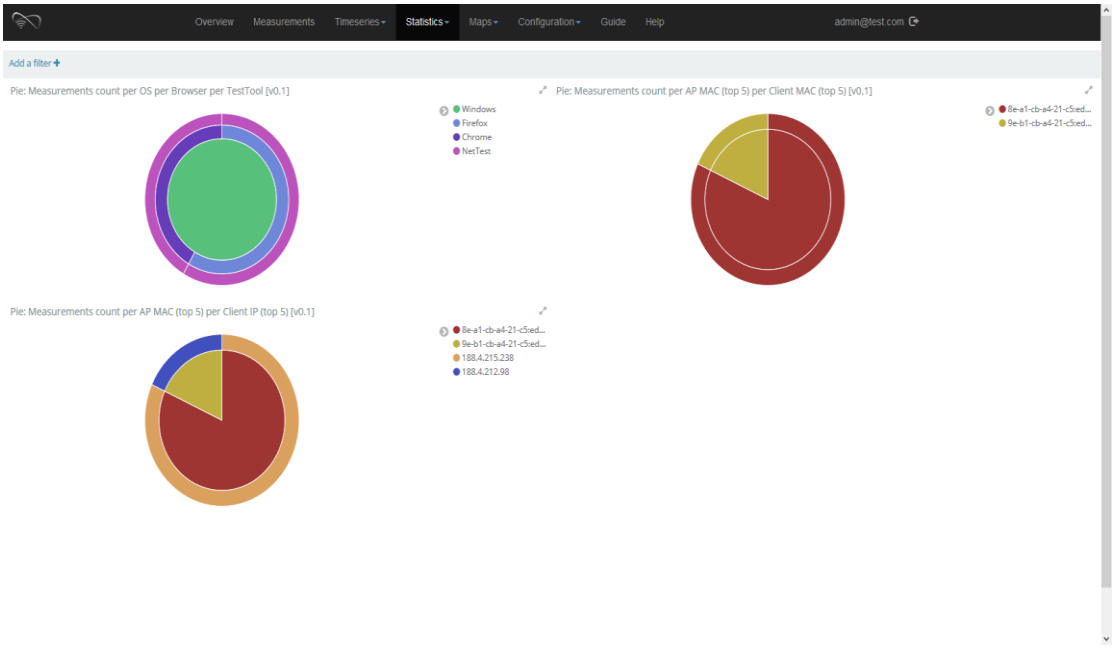
Στην Εικόνα 7 φαίνονται ορισμένα χρονοδιαγράμματα σχετικά με την ταχύτητα λήψης αρχείων. Αυτά τα στοιχεία περιλαμβάνουν την μέση ταχύτητα λήψης ανά IP, ανά MAC, ανά λειτουργικό σύστημα, ανά εργαλείο δοκιμών και ανά περιηγητή. Αντίστοιχα διαγράμματα υπάρχουν και για την ταχύτητα αποστολής και για το ping.

Στην Εικόνα 8 εμφανίζονται μερικά διαγράμματα πίτας τα οποία δείχνουν από πού προήλθαν οι μετρήσεις. Το πρώτο διάγραμμα δείχνει το ποσοστό των μετρήσεων με βάση το εργαλείο δοκιμών που χρησιμοποιήθηκε για την διεξαγωγή των μετρήσεων καθώς και τον περιηγητή και το λειτουργικό σύστημα από το οποίο διεξάχθηκαν. Το δεύτερο διάγραμμα δείχνει το ποσοστό των μετρήσεων με βάση την διεύθυνση MAC του σημείου πρόσβασης και την διεύθυνση MAC του πελάτη. Το τρίτο διάγραμμα

δείχνει το ποσοστό των μετρήσεων με βάση την διεύθυνση MAC του σημείου πρόσβασης και την διεύθυνση IP του πελάτη.

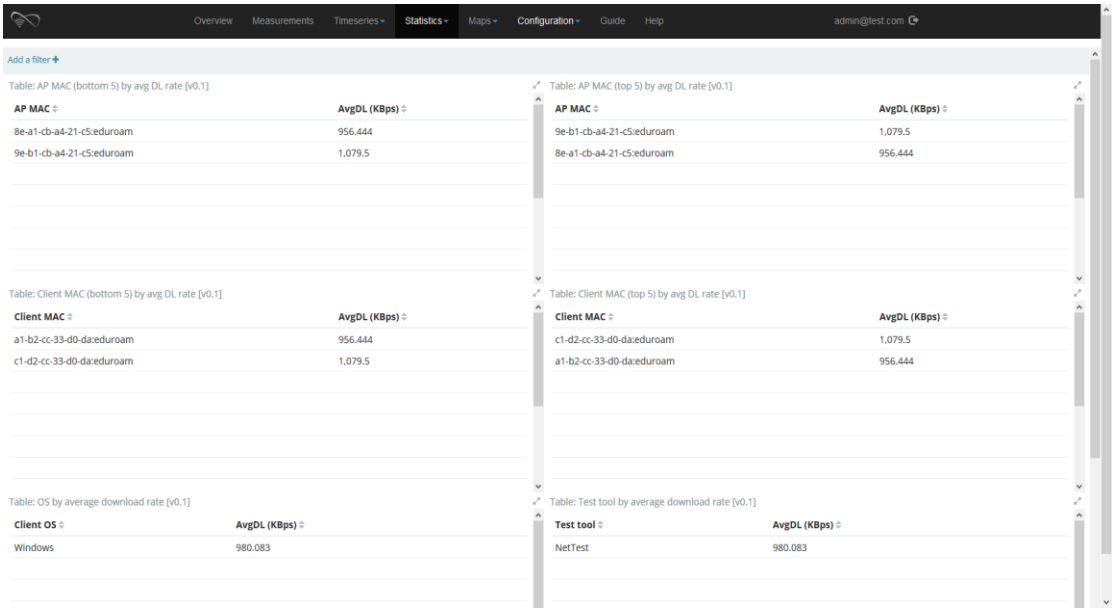


Εικόνα 7. Download Timeseries



Εικόνα 8. Pie charts

Στην Εικόνα 9 φαίνονται μερικά στατιστικά υπό την μορφή πινάκων. Παρουσιάζονται οι MAC διευθύνσεις των πέντε καλύτερων και χειρότερων σημείων πρόσβασης με βάση την ταχύτητα λήψης. Αντίστοιχοι πίνακες εμφανίζονται και για τους πελάτες. Επίσης εμφανίζονται πίνακες που δείχνουν την ταχύτητα κατεβάσματος ανά εργαλείο δοκιμών, ανά περιηγητή και ανά λειτουργικό σύστημα.



Εικόνα 9. Table Statistics

ΚΕΦΑΛΑΙΟ 5: ΥΛΟΠΟΙΗΣΗ

ΥΛΟΠΟΙΗΣΗ

Όπως αναφέραμε στο προηγούμενο κεφάλαιο, το WiFiMon αποτελείται από τα ακόλουθα 6 συστατικά μέρη:

1. Έναν Postgres server στον οποίο αποθηκεύονται οι ρυθμίσεις του WiFiMon.
2. Έναν Elasticsearch server στον οποίο αποθηκεύονται οι μετρήσεις που παίρνουμε.
3. Το kibana, το οποίο το χρησιμοποιούμε για να οπτικοποιήσουμε τις μετρήσεις που έχουμε αποθηκεύσει στο elasticsearch.
4. Το wifimon-ui το οποίο προσφέρει μια διεπαφή χρήστη για να δούμε τις μετρήσεις που έχουμε πάρει.
5. Το wifimon-processor το οποίο συλλέγει τις μετρήσεις και τις αποθηκεύει στο elasticsearch.
6. Το wifimon-secure-processor το οποίο επιτελεί την ίδια λειτουργεία με το wifimon-processor αλλά λειτουργεί τόσο με https όσο και με http σελίδες.

Άρα θα χρειαστεί να δημιουργήσουμε έξι εικόνες, μια για κάθε μια από τις από πάνω υπηρεσίες καθώς και ένα docker-compose.yml ώστε να μπορούμε να εκκινήσουμε και τις έξι υπηρεσίες με μια μόνο εντολή.

5.1 POSTGRES

Κοιτάζοντας στο Docker Hub μπορούμε να δούμε ότι υπάρχει μια εικόνα για το postgres (https://hub.docker.com/_/postgres/), την οποία μπορούμε να χρησιμοποιήσουμε.

Η εικόνα αυτή έχει 2 δυνατότητες που θα μας φανούν χρήσιμες:

1. Κατά την πρώτη εκκίνηση της εικόνας, μπορούμε να δημιουργήσουμε αυτόματα έναν χρήστη και μια βάση δεδομένων ορίζοντας τις μεταβλητές περιβάλλοντος POSTGRES_USER, POSTGRES_PASSWORD και POSTGRES_DB του container.
2. Κατά την εκκίνηση το postgres θα τρέξει ότι αρχεία τύπου *.sql, *.sql.gz ή *.sh υπάρχουν στην θέση /docker-entrypoint-initdb.d (εντός του container) προκειμένου να κάνει περαιτέρω αρχικοποίηση.

Οπότε θα δημιουργήσουμε ένα αρχείο init.sql που θα περιέχει τις εντολές για την αρχικοποίηση της βάσης δεδομένων (δημιουργία πινάκων) και στη συνέχεια μια νέα εικόνα που θα βασίζεται στην εικόνα του postgres και επιπροσθέτως θα έχει αντιγράψει το αρχείο init.sql από τον υπολογιστή μας, στη θέση /docker-entrypoint-initdb.d στο container.

Για να δημιουργήσουμε την νέα αυτή εικόνα, θα δημιουργήσουμε ένα Dockerfile σαν αυτό που δείχνει ο Πίνακας 2.

```
# Use the official postgres image as the parent image
FROM postgres

# copy init.sql to /docker-entrypoint-initdb.d (inside the container)
# init.sql contains the commands needed to initialize the database.
```

```
# Files under /docker-entrypoint-initdb.d will run during initialization.
COPY init.sql /docker-entrypoint-initdb.d
```

Πίνακας 2. Postgres Dockerfile

Η εντολή FROM δηλώνει ποια εικόνα θα χρησιμοποιήσουμε ως βάση και είναι συνήθως η πρώτη εντολή σε ένα dockerfile.

Η εντολή COPY αντιγράφει αρχεία ή φακέλους από τον υπολογιστή μας στο container.

Στην συνέχεια μπορούμε να χτίσουμε την εικόνα και να την ανεβάζουμε στο repository μας με τις παρακάτω εντολές:

```
docker build -t postgres . //build image
docker tag postgres theodim/postgres //tag image
docker push theodim/postgres //push to repository
```

Με την εντολή docker build χτίζουμε την εικόνα. Η εντολή συντάσσεται ως *docker build [OPTIONS] PATH*. Η εντολή θα χρησιμοποιήσει το αρχείο με το όνομα "Dockerfile" που βρίσκεται στο path που έχουμε δώσει. Αν το αρχείο μας έχει διαφορετικό όνομα από αυτό, μπορούμε να το δηλώσουμε με την επιλογή -f name π.χ. "docker build -f postgres-Dockerfile.txt .".

Με την εντολή docker tag δίνουμε ένα όνομα στην εικόνα. Η σημειογραφία για το όνομα μιας εικόνας που θέλουμε να συσχετίσουμε με ένα repository στο Docker Hub είναι username/repository:tag. Το tag είναι προαιρετικό και το χρησιμοποιούμε για να δώσουμε στην εικόνα μια έκδοση. Αν δεν δώσουμε κάποιο tag, το docker θα προσθέσει το "latest" ως το tag της εικόνας. Αν κάποια εικόνα δεν έχει username (όπως η εικόνα του postgres που χρησιμοποιήσαμε παραπάνω), τότε πρόκειται για επίσημο repository στο Docker Hub.

Μπορούμε να ενώσουμε τις δύο προηγούμενες εντολές σε μια χρησιμοποιώντας την επιλογή -t της εντολής docker build:

```
docker build -t theodim/postgres . //build image and tag it as theodim/postgres:latest
```

Με την εντολή docker push ανεβάζουμε την εικόνα στο registry. Η εντολή αυτή συντάσσεται ως docker push [OPTIONS] NAME[:TAG]. Η τοπική εικόνα με το όνομα NAME θα ανέβει στο repository με το όνομα NAME. Για παράδειγμα, με την εντολή "docker push theodim/postgres", η τοπική εικόνα με το όνομα theodim/postgres θα ανέβει στο repository στο Docker Hub με το όνομα theodim/postgres. Για να μπορέσουμε να ανεβάσουμε μια εικόνα στο Docker Hub, θα πρέπει πρώτα να έχουμε δημιουργήσει ένα λογαριασμό στο Docker Hub (https://hub.docker.com) καθώς και να έχουμε κάνει login με την εντολή docker login.

Αν θέλουμε να χρησιμοποιήσουμε την εικόνα μόνο σε τοπικό επίπεδο, μπορούμε να παραλείψουμε αυτό το βήμα.

Στην συνέχεια θα δημιουργήσουμε το docker-compose.yml όπως φαίνεται στον Πίνακας 3

```
version: "3.3"
services:
  postgres:
    image: theodim/postgres:latest
    ports:
      - 5432:5432
```

```
    networks:
      - webnet
    environment:
      POSTGRES_DB: wifimon_database
      POSTGRES_USER: wifimon_user
      POSTGRES_PASSWORD: wifimonpass
    volumes:
      - postgres_data:/var/lib/postgresql/data
networks:
  webnet:
volumes:
  postgres_data:
```

Πίνακας 3. Το τμήμα του docker-compose.yml για το postgres

Κάτω από το services δηλώνουμε όλες τις υπηρεσίες που επιθυμούμε να ξεκινήσουμε καθώς και τις αντίστοιχες ρυθμίσεις τους.

Το "postgres" (κάτω από το πεδίο services) είναι το όνομα που δώσαμε στην υπηρεσία. Άλλες υπηρεσίες που βρίσκονται στο ίδιο δίκτυο στο docker μπορούν να χρησιμοποιήσουν αυτό το όνομα για να συνδεθούν σε αυτήν την υπηρεσία. Για παράδειγμα το wifimon-ui μπορεί να χρησιμοποιήσει το postgres:5432 για να συνδεθεί με την υπηρεσία Postgres που δημιουργήσαμε από πάνω. Το docker θα αντιστοιχίσει το όνομα "postgres" στην ip που έχει αναθέσει για την υπηρεσία postgres στο εικονικό δίκτυο που έχει δημιουργήσει.

Με το πεδίο image δηλώνουμε ποια εικόνα θέλουμε να χρησιμοποιήσουμε για αυτήν την υπηρεσία. Το όνομα της εικόνας είναι αυτό που της είχαμε δώσει νωρίτερα με την εντολή docker tag.

Με το πεδίο ports δηλώνουμε ποιες θύρες θέλουμε να ανοίξουμε ώστε να είναι προσβάσιμες εκτός του container. Το πεδίο ports έχει την ακόλουθή σύνταξη:

```
ports:
 - HOST:CONTAINER
```

Με το πεδίο networks προσθέτουμε την υπηρεσία στο εικονικό δίκτυο webnet. Το δίκτυο αυτό θα πρέπει να το έχουμε δημιουργήσει δηλώνοντας το κάτω από το πεδίο networks στο κορυφαίο επίπεδο. Webnet είναι απλά το όνομα που δώσαμε στο δίκτυο και δεν έχει κάποια άλλη ιδιαίτερη σημασία. Θα μπορούσαμε να του είχαμε δώσει όποιο άλλο όνομα θέλαμε. Υπηρεσίες που ανήκουν στο ίδιο δίκτυο έχουν όλες τους θύρες ανοικτές μεταξύ τους. Αυτό σημαίνει ότι αν π.χ. το wifimon-ui και το postgres είναι στο ίδιο δίκτυο, τότε ακόμα και αν δεν είχαμε χρησιμοποιήσει το πεδίο ports για να ανοίξουμε την θύρα 5432, το wifimon-ui και το postgres θα μπορούσαν να επικοινωνήσουν μεταξύ τους. Αλλά δεν θα μπορούσαμε να συνδεθούμε απευθείας στο postgres. Αυτός είναι και ένας τρόπος για να περιορίσουμε την πρόσβαση σε υπηρεσίες στις οποίες δεν θέλουμε να έχουμε απευθείας πρόσβαση αλλά μόνο μέσω κάποιας άλλης εφαρμογής.

Με το πεδίο environment ορίζουμε τις μεταβλητές περιβάλλοντος του container.

Όταν ένα container τερματίσει, όλα τα δεδομένα σε αυτό χάνονται. Με το πεδίο volumes δηλώνουμε ότι θέλουμε τα δεδομένα στη θέση /var/lib/postgresql/data να

αποθηκεύονται μόνιμα στο volume με το όνομα postgres_data ώστε να μην χάνονται. Το volume πρέπει επίσης να το δηλώσουμε κάτω από το πεδίο volumes στο κορυφαίο επίπεδο.

5.2 ELASTICSEARCH

Η εικόνα του elasticsearch στο docker hub (https://hub.docker.com/_/elasticsearch/) έχει την σημείωση ότι είναι deprecated και μας παραπέμπει στο registry του elasticsearch (https://www.docker.elastic.co/).

Από το documentation της εικόνας παρατηρούμε τα ακόλουθα πράγματα:

1. Για elasticsearch ≥6.0 υπάρχουν τρεις διαφορετικές εικόνες. Η πρώτη εικόνα έχει προεγκαταστημένο το X-Pack Basic και ενεργοποιημένο με μια δωρεάν άδεια. Η δεύτερη εικόνα έχει προεγκαταστημένο το X-Pack platinum με μια δοκιμαστική άδεια 30 ημερών. Η τρίτη εικόνα δεν έχει εγκαταστημένο το X-pack και περιέχει μόνο ανοιχτού κώδικα Elasticsearch.
   Για την έκδοση 5.6 όμως υπάρχει μόνο μια εικόνα, η οποία έχει προεγκαταστημένη την δοκιμαστική έκδοση του X-pack. Αφού δεν θέλουμε να χρησιμοποιήσουμε το X-pack θα πρέπει είτε να το απενεργοποιήσουμε στις ρυθμίσει είτε να το απεγκαταστήσουμε πλήρως.
2. Σε linux χρειάζεται να τρέξουμε την εντολή "sysctl -w vm.max_map_count=262144" πριν ξεκινήσουμε το container.
3. Μπορούμε να περάσουμε ρυθμίσεις στο elasticsearch χρησιμοποιώντας μεταβλητές περιβάλλοντος, π.χ "cluster.name=mynewclustername".
4. Το elasticsearch αποθηκεύει τα δεδομένα του στη θέση /usr/share/elasticsearch/data οπότε θα χρειαστεί να χρησιμοποιήσουμε ένα volume προκειμένου να τα διατηρήσουμε.

Μετά την εκκίνηση του elasticsearch, θέλουμε να τρέχουμε το script elasticsearch.sh ώστε να δημιουργούμαι τα indices του elasticsearch που χρειαζόμαστε. Οπότε θα ξεκινήσουμε αντιγράφοντας το elasticsearch.sh στο container. Στην συνέχεια όμως παρατηρούμε ένα πρόβλημα. Το elasticsearch.sh πρέπει να το τρέξουμε αφού έχουμε εκκινήσει το elasticsearch και είναι έτοιμο να δεχτεί συνδέσεις. Συγχρόνως όμως θέλουμε να εκκινήσουμε το elasticsearch με την εντολή exec ώστε να έχει PID 1. Το docker στέλνει UNIX σήματα (όπως το σήμα SIGTERM που στέλνει η εντολή docker stop <container>) μόνο στο process με PID 1. Αν το elasticsearch δεν τερματίσει από μόνο του, τότε θα το τερματίσει το docker, κάτι που θα μπορούσε να οδηγήσει σε απώλεια δεδομένων.

Η λύση στο πρόβλημα αυτό είναι απλή. Μπορούμε να εκκινήσουμε προσωρινά το elasticsearch, να τρέξουμε το script elasticsearch.sh ώστε να κάνουμε την αρχικοποίηση που θέλουμε και στην συνέχεια να το τερματίσουμε και να το ξανά εκκινήσουμε, αυτή την φορά χρησιμοποιώντας την εντολή exec.

Ο Πίνακας 4 δείχνει το dockerfile για το elasticsearch:

```
# Use the official elasticsearch image as the parent image
FROM docker.elastic.co/elasticsearch/elasticsearch:5.6.8

#disable x-pack
RUN bin/elasticsearch-plugin remove x-pack --purge
```

```
# copy elasticsearch.sh into the container and run it on startup
COPY elasticsearch.sh /usr/local/bin/

#initialize indices
RUN exec bin/elasticsearch -p /tmp/elasticsearch-pid & \
   /bin/bash elasticsearch.sh; \
         kill $(cat /tmp/elasticsearch-pid) \
         && wait $(cat /tmp/elasticsearch-pid); \
         exit 0;
```

Πίνακας 4. Elasticsearch Dockerfile

Η εντολή RUN ισοδυναμεί με το να είχαμε τρέξει την εντολή σε ένα terminal μέσα στο container.

Με την εντολή bin/elasticsearch-plugin remove x-pack –purge απεγκαταστήσαμε το x-pack ενώ με την εντολή στο δεύτερο RUN εκκινήσαμε προσωρινά το elasticsearch, τρέξαμε το script elasticsearch.sh και στην συνέχεια τερματίσαμε το elasticsearch.

Ενδεικτικά η αρχή από το elasticsearch.sh φαίνεται στον Πίνακας 5.

```
#!/bin/sh


while ! curl -s -f http://localhost:9200/
do
        echo "$(date) - wait till elasticsearch is ready to accept connections"
        sleep 5
done


# create indices wifimon and radiuslogs for Elasticsearch
echo "$(date) - create index wifimon and radiuslogs"
curl -s -XPUT 'localhost:9200/wifimon?pretty' -H 'Content-Type: application/json' -d'
{  "mappings" : {
    "measurement" : {
      "properties" : {
        "timestamp" : { "type" : "date" },
        "downloadThroughput" : { "type" : "float" },
        "uploadThroughput" : { "type" : "float" },
        "localPing" : { "type" : "float" },
        "location" : { "type" : "geo_point" },
        "locationMethod" : { "type" : "keyword" },
        "clientIp" : { "type" : "keyword" },
```

```
        "userAgent" : { "type" : "keyword" },
        "userBrowser" : { "type" : "keyword" },
        "userOS" : { "type" : "keyword" },
        "testTool" : { "type" : "keyword" },
        "username" : { "type" : "keyword" },
        "nasPort" : { "type" : "keyword" },
        "callingStationId" : { "type" : "keyword" },
        "nasIdentifier" : { "type" : "keyword" },
        "calledStationId" : { "type" : "keyword" },
        "nasIpAddress" : { "type" : "keyword" },
        "apBuilding" : { "type" : "keyword" },
        "apFloor" : { "type" : "keyword" },
        "apLocation" : { "type" : "geo_point" },
        "apNotes" : { "type" : "keyword" }
      }
    }
  }
}'
```

Πίνακας 5. Απόσπασμα από το αρχείο elasticsearch.sh

Στην συνέχεια στον Πίνακας 5 θα ακολουθούσαν οι εντολές για την δημιουργία των υπόλοιπων indices.

Το «while ! curl -s -f http://localhost:9200/» στην αρχή του script είναι για να ελέγξουμε αν το elasticsearch είναι έτοιμο να δεχθεί συνδέσεις.

Τόσο το «while ! curl -s -f http://localhost:9200/» όσο και τις εντολές για την δημιουργία των indices θα μπορούσαμε να τα είχαμε τοποθετήσει απευθείας μέσα στην εντολή RUN στο dockerfile αλλά με την χρήση ξεχωριστού αρχείου είναι ποιο ευανάγνωστα.

Όπως και με το postgres, χρησιμοποιούμε τις παρακάτω εντολές για να χτίσουμε την εικόνα και να την ανεβάζουμε στο repository μας. Το dockerfile και το elasticsearch.sh βρίσκονται μέσα στον φάκελο elasticsearch στην θέση «./elasticsearch .»

```
docker build -t theodim/elasticsearch:5.6.8  ./elasticsearch //build and tag image
docker push theodim/elasticsearch //push to repository
```

Στην συνέχεια προσθέτουμε στο αρχείο docker-compose.yml που είχαμε δημιουργήσει πριν τις αντίστοιχες ρυθμίσεις για το elasticsearch. Η τωρινή μορφή του αρχείου φαίνεται στον Πίνακας 6.

```
version: "3.3"
services:
  postgres:
    image: theodim/postgres:latest
```

```
    ports:
      - 5432:5432
    networks:
      - webnet
    environment:
      POSTGRES_DB: wifimon_database
      POSTGRES_USER: wifimon_user
      POSTGRES_PASSWORD: wifimonpass
    volumes:
      - postgres_data:/var/lib/postgresql/data

  elasticsearch:
    image: theodim/elasticsearch:5.6.8
    volumes:
      - es_data:/usr/share/elasticsearch/data
    ports:
      - 9200:9200
      - 9300:9300
    networks:
      - webnet
    environment:
      - cluster.name=elasticsearch
      #set heap size
      - "ES_JAVA_OPTS=-Xms1g -Xmx1g"
networks:
  webnet:
volumes:
  postgres_data:
  es_data:
```

Πίνακας 6. Το τμήμα του docker-compose.yml για το elasticsearch και το postgres

Ομοίως με το postgres, ονομάσαμε την υπηρεσία elasticsearch, δηλώσαμε ότι θα χρησιμοποιεί την εικόνα theodim/elasticsearch:5.6.8, αντιστοιχίσαμε το volume es_data με την θέση /usr/share/elasticsearch/data όπου το elasticsearch αποθηκεύει δεδομένα, ανοίξαμε τα ports 9200 και 9300, προσθέσαμε την υπηρεσία στο δίκτυο webnet και περάσαμε κάποιες ρυθμίσεις στο elasticsearch χρησιμοποιώντας μεταβλητές περιβάλλοντος.

5.3 KIBANA

Η εικόνα του kibana στο docker hub (https://hub.docker.com/_/kibana/) έχει την σημείωση ότι είναι deprecated και μας παραπέμπει στο registry που διατηρεί το elasticsearch (https://www.docker.elastic.co/).

Από το documentation της εικόνας παρατηρούμε τα ακόλουθα:

1. Για εκδόσεις του kibana ≥6.0 υπάρχουν δύο διαφορετικές εικόνες, μια με το x-pack προεγκαταστημένο και μια χωρίς το x-pack.
   Για την έκδοση kibana 5.6 όμως υπάρχει μόνο μια εικόνα, η οποία έχει

προεγκαταστημένο το x-pack. Οπότε θα χρειαστεί να το απενεργοποιήσουμε ή να το απεγκαταστήσουμε.

2. Μπορούμε να περάσουμε ρυθμίσεις στο kibana χρησιμοποιώντας μεταβλητές περιβάλλοντος. Οι μεταβλητές έχουν το ίδιο όνομα με τις ρυθμίσεις αλλά είναι με κεφαλαία γράμματα και χρησιμοποιούμε κάτω παύλα για διαχωριστικό των λέξεων αντί για τελεία. Π.χ SERVER_NAME για server.name ή KIBANA_DEFAULTAPPID για kibana.defaultAppId.

Θέλουμε να δημιουργήσουμε στο kibana κάποια index patterns, visualizations και dashboards με αυτόματο τρόπο. Όλα τα δεδομένα του kibana αποθηκεύονται σε ένα index με το όνομα .kibana στο elasticsearch. Οπότε ένας τρόπος για να τα δημιουργήσουμε είναι να γράψουμε απευθείας σε αυτό το index χρησιμοποιώντας το elasticsearch api. Αφού έχουμε να δημιουργήσουμε 50+ στοιχεία θα χρησιμοποιήσουμε το bulk api του elasticsearch. Το endpoint αυτού του api είναι /_bulk και δέχεται την ακόλουθη newline delimited JSON δομή.

```
action_and_meta_data\n
optional_source\n
action_and_meta_data\n
optional_source\n
....
action_and_meta_data\n
optional_source\n
```

π.χ:

```
{ "index" : { "_index" : ".kibana", "_type" : "index-pattern", "_id" : "radiuslogs_v0.1" } }
{ "title": "radiuslogs", "timeFieldName": "timestamp", "notExpandable": "true" }
{ "index" : { "_index" : ".kibana", "_type" : "index-pattern", "_id" : "wifimon_v0.1" } }
{ "title": "wifimon", "timeFieldName": "timestamp", "notExpandable": "true" }
```

Και ανεβάζουμε στο api με την εντολή

curl -s -H "Content-Type: application/x-ndjson" -XPOST localhost:9200/_bulk --data-binary "@kibana-import.json"

Παρατηρούμε όμως ότι η μορφή που δέχεται αυτό το api είναι διαφορετική από αυτή που έχει το αρχείο kibana-import.json. Οπότε δεν είναι βολικό να χρησιμοποιήσουμε αυτόν τον τρόπο.

Δεδομένου ότι η μορφή των στοιχείων αυτών δεν πρόκειται να αλλάξει συχνά, ένας άλλος τρόπος είναι να τα δημιουργήσουμε χειροκίνητα (χρησιμοποιώντας το gui του kibana για να κάνουμε import το kibana-import.json) και στην συνέχεια να αντιγράψουμε το φάκελο /usr/share/elasticsearch/data του elasticsearch και να τον χρησιμοποιήσουμε στην εικόνα μας. Θα πρέπει να έχουμε σταματήσει το elasticsearch πριν αντιγράψουμε τον φάκελο.

Οπότε θα χρειαστεί να τροποποιήσουμε το dockerfile για το elasticsearch που είχαμε δημιουργήσει ώστε να αντιγράφει αυτό τον φάκελο στο container. Ο φάκελος data θα περιέχει επίσης τα indices για το elasticsearch οπότε δεν χρειάζεται να χρησιμοποιήσουμε το script που είχαμε δημιουργήσει.

Οπότε το νέο dockerfile για το elasticsearch φαίνεται στον Πίνακας 7:

```
# Use the official elasticsearch image as the parent image
FROM docker.elastic.co/elasticsearch/elasticsearch:5.6.8
```

```
#disable x-pack
RUN bin/elasticsearch-plugin remove x-pack --purge

# copy elasticsearch data into the container
COPY ./data /usr/local/bin/data
```

Πίνακας 7. Ανανεωμένο Elasticsearch Dockerfile

Το kibana χρησιμοποιεί επίσης ένα ssl certificate και ένα ssl key τα οποία θα τα παρέχει ο χρήστης. Άρα κατά την εκκίνηση του container θα χρειαστεί να αντιγράψουμε αυτά τα αρχεία από τον υπολογιστή μας στο container. Αυτό μπορούμε να το κάνουμε χρησιμοποιώντας docker secrets (στο αρχείο docker-compose.yml).

Οπότε το dockerfile για Kibana θα είναι αυτό που φαίνεται στον Πίνακας 8:

```
# Use the official kibana image as the parent image
FROM docker.elastic.co/kibana/kibana:5.6.8

#disable x-pack
RUN bin/kibana-plugin remove x-pack && \
    kibana 2>&1 | grep -m 1 "Optimization of .* complete" # [1]
```

Πίνακας 8. Kibana Dockerfile

Χρησιμοποιούμε την εικόνα του Kibana από το elasticsearch registry και το μόνο που κάνουμε είναι να απεγκαθιστούμε το x-pack. Μετά την απεγκατάσταση του x-pack, το Kibana θα κάνει κάποιο optimization στην επόμενη εκκίνηση του. Θα πρέπει να φροντίσουμε αυτό το optimization να γίνει κατά την δημιουργία της εικόνας ώστε να αποφύγουμε το να γίνεται κάθε φόρα που εκκινούμε το container. Υπεύθυνο για αυτό είναι η εντολή «kibana 2>&1 | grep -m 1 "Optimization of .* complete" # [1]» που εκτελούμε παραπάνω.

Έπειτα χτίζουμε την εικόνα και την ανεβάζουμε την εικόνα στο repository μας.

```
docker build -t theodim/kibana:5.6.8 ./kibana //build and tag image
docker push theodim/kibana //push to repository
```

Επόμενο βήμα είναι να προσθέσουμε το αντίστοιχό τμήμα για το kibana στο αρχείο docker-compose.yml που έχουμε δημιουργήσει.

Το αντίστοιχο τμήμα για το kibana φαίνεται στον Πίνακας 9:

```
services:
  kibana:
    image: theodim/kibana:5.6.8
    ports:
      - 5601:5601
    networks:
      - webnet
    environment:
      #Do not edit the ssl certificate & key here.
      #If you want to change the path to the certificate,
      #edit the path under the secrets section on the end of the file
      SERVER_SSL_CERTIFICATE: "/run/secrets/ssl.crt"
      SERVER_SSL_KEY: "/run/secrets/ssl.key"
```

```
    secrets:
      - ssl.crt
      - ssl.key

secrets:
  ssl.crt:
    file: ./certificate/ssl.crt
  ssl.key:
    file: ./certificate/ssl.key
```

Πίνακας 9. Το τμήμα του docker-compose.yml για το kibana

Ονομάσαμε την υπηρεσία “kibana”, δηλώσαμε ότι θα χρησιμοποιήσουμε την εικόνα theodim/kibana:5.6.8, ανοίξαμε το port 5601, χρησιμοποιήσαμε μεταβλητές περιβάλλοντος για να περάσουμε κάποιες ρυθμίσεις στο kibana και προσθέσαμε το kibana στο δίκτυο webnet. Επίσης δηλώσαμε ότι χρησιμοποιούμε τα secrets ssl.crt και ssl.key. Παρόμοια με τα network και τα volumes, τα secrets πρέπει επίσης να τα δηλώσουμε σε κορυφαίο επίπεδο. Εδώ έχουμε κάνει την υπόθεση ότι τα ssl certificate και key θα βρίσκονται σε ένα φάκελο (στον υπολογιστή μας) στη θέση ./certificate και θα έχουν όνομα ssl.crt και ssl.key αντίστοιχα. Αν βρίσκονται σε διαφορετική θέση ή έχουν διαφορετικό όνομα, θα πρέπει να τροποποιήσουμε το path παραπάνω ώστε να δείχνει στην σωστή θέση.

5.4 WIFIMON PROCESSOR, SECURE PROCESSOR ΚΑΙ UI

Τα WiFiMon processor, secure processor και ui έχουν παρόμοιες απαιτήσεις οπότε θα τα εξετάσουμε παράλληλα. Χρησιμοποιούν Java οπότε μπορούμε να χρησιμοποιήσουμε μια εικόνα της Java 8. Στο Docker Hub η εικόνα της Java (https://hub.docker.com/_/java/) μας πληροφορεί ότι είναι deprecated και μας προτείνει να χρησιμοποιήσουμε την εικόνα του openjdk αλλά παρόλα αυτά μπορούμε να την χρησιμοποιήσουμε.

Θα χρησιμοποιήσουμε το αρχείο .jar του καθενός, το οποίο μπορούμε να αντιγράψουμε στο container με την εντολή COPY.

Τα secure processor και UI χρησιμοποιούν ένα Java KeyStore αρχείο. Αυτό θα το παρέχει ο χρήστης οπότε μπορούμε να το φορτώσουμε κατά την εκκίνηση χρησιμοποιώντας docker secrets.

Τα processor, secure processor και ui χρησιμοποιούν από ένα αρχείο config το καθένα που περιέχει τις ρυθμίσεις για την εφαρμογή. Αυτά μπορούμε να τα φορτώσουμε κατά την εκκίνηση χρησιμοποιώντας docker configs. Τα configs δουλεύουν με τον ίδιο τρόπο όπως τα secrets με κύρια διαφορά ότι τα secrets κρυπτογραφούνται ενώ τα configs όχι. Οπότε τα secrets συνήθως χρησιμοποιούνται για ευαίσθητα δεδομένα ενώ τα configs για μη ευαίσθητα δεδομένα.

Κατά την εκκίνηση, τα WiFiMon processor, secure processor και ui επικοινωνούν με τον postgres server οπότε πριν τα εκκινήσουμε θα πρέπει να φροντίσουμε ο postgres server να είναι έτοιμος να δεχθεί συνδέσεις. Για αυτό τον σκοπό μπορούμε να χρησιμοποιήσουμε το wait-for-it.sh (https://github.com/vishnubob/wait-for-it) που είναι ένα bash script που εκτελεί αυτήν την λειτουργία. Το wait-for-it.sh περιμένει μέχρι ένας host και μια θύρα να είναι διαθέσιμα προτού προχωρήσει παρακάτω. Έχει την σύνταξη που φαίνεται στον Πίνακας 10.

```
wait-for-it.sh host:port [-s] [-t timeout] [-- command args]
-h HOST | --host=HOST       or IP under test
-p PORT | --port=PORT       TCP port under test
                            Alternatively, you specify the host and
                            port as host:port
-s | --strict               Only execute subcommand if the test
                            succeeds
-q | --quiet                Don't output any status messages
-t TIMEOUT | --timeout=TIMEOUT
                            Timeout in seconds, zero for no timeout
-- COMMAND ARGS             Execute command with args after the
                            test finishes
```

Πίνακας 10. wait-for-it.sh usage

Η διεύθυνση του postgres ορίζεται στο αρχείο config της κάθε υπηρεσίας (π.χ. ./config/processor.properties για το processor) οπότε θα χρειαστεί να διαβάσουμε την διεύθυνση του postgres από αυτό το αρχείο. Αυτό μπορούμε να το κάνουμε με μια εντολή όπως η «grep -oP '(?<=^spring.datasource.url=jdbc:postgresql:\/\/).*?:[0-9]*' ./config/processor.properties»

Οπότε τα dockerfile για τα wifimon ui, processor και secure processor φαίνονται στους Πίνακας 11, Πίνακας 12 και Πίνακας 13 αντίστοιχα.

```
# Use the official java image as the parent image
FROM java:8

#set work directory to /usr/lib/wifimon/
WORKDIR /usr/lib/wifimon/

#download wait-for-it-sh
ADD https://raw.githubusercontent.com/vishnubob/wait-for-it/master/wait-for-it.sh /usr/local/bin/

# copy wifimon ui.jar into the container
COPY ./ui-0.1.1-SNAPSHOT.jar /usr/lib/wifimon/

COPY docker-entrypoint.sh /usr/local/bin/

#Give execute permission to scripts
RUN chmod +x /usr/local/bin/wait-for-it.sh && chmod +x /usr/local/bin/docker-entrypoint.sh

EXPOSE 8441
ENTRYPOINT ["docker-entrypoint.sh","java","-jar","./ui-0.1.1-SNAPSHOT.jar"]
CMD ["--spring.config.location=classpath:/ui.properties,file:./config/ui.properties"]
```

Πίνακας 11. Wifimon ui Dockerfile

```
# Use the official java image as the parent image
FROM java:8
```

```
#set work directory to /usr/lib/wifimon/
WORKDIR /usr/lib/wifimon/

#download wait-for-it-sh
ADD https://raw.githubusercontent.com/vishnubob/wait-for-it/master/wait-for-it.sh /usr/local/bin/

# copy wifimon ui.jar into the container
COPY ./processor-0.1.1-SNAPSHOT.jar /usr/lib/wifimon/

COPY docker-entrypoint.sh /usr/local/bin/

#Give execute permission to scripts
RUN chmod +x /usr/local/bin/wait-for-it.sh && chmod +x /usr/local/bin/docker-entrypoint.sh

EXPOSE 8441
ENTRYPOINT ["docker-entrypoint.sh","java","-jar","./processor-0.1.1-SNAPSHOT.jar"]
CMD ["--spring.config.location=classpath:/processor.properties,file:./config/processor.properties"]
```

Πίνακας 12. Wifimon processor Dockerfile

```
# Use the official java image as the parent image
FROM java:8

#set work directory to /usr/lib/wifimon/
WORKDIR /usr/lib/wifimon/

#download wait-for-it-sh
ADD https://raw.githubusercontent.com/vishnubob/wait-for-it/master/wait-for-it.sh /usr/local/bin/

# copy wifimon ui.jar into the container
COPY ./secure-processor-0.1.1-SNAPSHOT.jar /usr/lib/wifimon/

COPY docker-entrypoint.sh /usr/local/bin/

#Give execute permission to scripts
RUN chmod +x /usr/local/bin/wait-for-it.sh && chmod +x /usr/local/bin/docker-entrypoint.sh

EXPOSE 8441
ENTRYPOINT ["docker-entrypoint.sh","java","-jar","./secure-processor-0.1.1-SNAPSHOT.jar"]
CMD ["--spring.config.location=classpath:/secure-processor.properties,file:./config/secure-processor.properties"]
```

Πίνακας 13. Wifimon Secure Processor Dockerfile

Με την εντολή WORKDIR αλλάζουμε το working directory.

Η εντολή ADD εκτελεί την ίδια λειτουργία όπως η COPY (αντιγράφει αρχεία από τον υπολογιστή μας στο container) αλλά έχει και ορισμένες παραπάνω δυνατότητες. Αυτόματα αποσυμπιέζει tar αρχεία που έχουν μια αναγνωρισμένη μορφή συμπίεσης (identity, gzip, bzip2 ή xz) και μπορεί επίσης να χρησιμοποιηθεί για να αντιγράψουμε αρχεία από το internet στο container. Είναι καλή πρακτική [28] γενικά να χρησιμοποιούμε την COPY που η χρήση της είναι πιο εμφανής και να χρησιμοποιούμε την ADD μόνο όταν θέλουμε να χρησιμοποιήσουμε κάποια από τις επιπλέον δυνατότητες της.

Με την εντολή EXPOSE ενημερώνουμε το docker για το σε ποιες θύρες ακούει το container. Η εντολή αυτή δεν ανοίγει τις θύρες. Απλά αποτελεί ένα είδος documentation ανάμεσα σε αυτόν που χτίζει την εικόνα και σε αυτόν που την τρέχει, σχετικά με το ποιες θύρες θα πρέπει να ανοίξει.

Οι εντολές ENTRYPOINT και CMD έχουν παρόμοια λειτουργία. Καθορίζουν ποια εντολή θα εκτελεστεί όταν τρέξουμε την εικόνα. Μπορούμε να έχουμε μόνο ένα ENTRYPOINT και ένα CMD σε μια εικόνα. Αν έχουμε περισσότερα από ένα, τότε μόνο το τελευταίο θα χρησιμοποιηθεί και τα υπόλοιπα θα αγνοηθούν. Αν έχουμε και τα δυο τότε θα ενωθούν σε μια εντολή με τα ορίσματα της CMD να προστίθενται στο τέλος της ENTRYPOINT. Δηλαδή αν έχουμε:

```
ENTRYPOINT ["java","-jar","./secure-processor-0.1.1-SNAPSHOT.jar"]
CMD["--spring.config.location=classpath:/secure-processor.properties,file:./config/secure-processor.properties"]
```

τότε η εντολή που θα εκτελεστεί κατά την εκκίνηση της εικόνας είναι η

```
exec java -jar ./secure-processor-0.1.1-SNAPSHOT.jar --spring.config.location=classpath:/secure-processor.properties,file:./config/secure-processor.properties
```

Η διαφορά ανάμεσα στο CMD και το ENTRYPOINT είναι ότι, κατά την εκκίνηση της εικόνας, μπορούμε να παρακάμψουμε τα ορίσματα που δηλώνονται στο CMD δίνοντας άλλα δικά μας.

Αν εκκινήσουμε δηλαδή το container με την εντολή «docker run secure-processor –help» τότε η εντολή που θα εκτελεστεί θα είναι:

```
exec java -jar ./secure-processor-0.1.1-SNAPSHOT.jar --help
```

Για αυτό το λόγο το ENTRYPOINT χρησιμοποιείται για την κύρια εντολή ενώ το CMD για να δώσουμε προεπιλεγμένα ορίσματα τα οποία θα μπορεί να αλλάξει ο χρήστης.

Τα αρχεία docker-entrypoint.sh για τα WiFiMon ui, processor και secure processor θα είναι αυτά που φαίνονται στους Πίνακας 14, Πίνακας 15 και Πίνακας 16 αντίστοιχα.

```
#!/usr/bin/env bash


#wait till postgres is ready

wait-for-it.sh $(grep -oP '(?<=^spring.datasource.url=jdbc:postgresql:\/\/).*?:[0-9]*' ./config/ui.properties) -t 0
```

```
exec "$@"
```

Πίνακας 14. Wifimon Ui docker-entrypoint.sh

```
#!/usr/bin/env bash


#wait till postgres is ready
wait-for-it.sh $(grep -oP '(?<=^spring.datasource.url=jdbc:postgresql:\/\/).*?:[0-9]*' ./config/processor.properties) -t 0


exec "$@"
```

Πίνακας 15. WiFiMon Processor docker-entrypoint.sh

```
#!/usr/bin/env bash


#wait till postgres is ready
wait-for-it.sh $(grep -oP '(?<=^spring.datasource.url=jdbc:postgresql:\/\/).*?:[0-9]*' ./config/secure-processor.properties) -t 0


exec "$@"
```

Πίνακας 16. WiFiMon Secure Processor docker-entrypoint.sh

Στα αρχεία αυτά διαβάζουμε την διεύθυνση του postgres από το αρχείο ρυθμίσεων της κάθε εφαρμογής αντίστοιχα, περιμένουμε μέχρι το postgres να είναι έτοιμο να δεχθεί συνδέσεις και στην συνέχεια εκκινούμε την εφαρμογή.

Με τις ακόλουθες εντολές χτίζουμε τις εικόνες και τις ανεβάζουμε στο repository μας:

wifimon-ui:

```
docker build -t theodim/wifimon-ui:0.1.1 ./wifimon-ui //build and tag image
docker push theodim/wifimon-ui //push to repository
```

wifimon-processor:

```
docker build -t theodim/wifimon-processor:0.1.1 ./wifimon-processor //build and tag image
docker push theodim/wifimon-processor //push to repository
```

wifimon-secure-processor:

```
docker build -t theodim/wifimon-secure-processor:0.1.1 ./wifimon-secure-processor //build and tag image
docker push theodim/wifimon-secure-processor //push to repository
```

Το αντίστοιχο τμήμα του docker-compose.yml για τα WiFiMon ui, processor και secure processor φαίνεται στον Πίνακας 17.

```
services:
  ui:
```

```
  image: theodim/wifimon-ui:0.1.1
  ports:
    - 8441:8441
  networks:
    - webnet
  configs:
    - source: ui.properties
      target: /usr/lib/wifimon/config/ui.properties
  secrets:
    - source: keystore.jks
      target: /usr/lib/wifimon/keystore/keystore.jks

processor:
  image: theodim/wifimon-processor:0.1.1
  ports:
    - 9000:9000
  networks:
    - webnet
  configs:
    - source: processor.properties
      target: /usr/lib/wifimon/config/processor.properties
  deploy:
    replicas: 0

secure-processor:
  image: theodim/wifimon-secure-processor:0.1.1
  ports:
    - 8443:8443
  networks:
    - webnet
  configs:
    - source: secure-processor.properties
      target: /usr/lib/wifimon/config/secure-processor.properties
  secrets:
```

```
    - source: keystore.jks
      target: /usr/lib/wifimon/keystore/keystore.jks
  deploy:
    replicas: 1


networks:
 webnet:
configs:
 ui.properties:
  file: ./config/ui.properties
 processor.properties:
  file: ./config/processor.properties
 secure-processor.properties:
  file: ./config/secure-processor.properties
secrets:
 keystore.jks:
  file: ./keystore/keystore.jks
```

Πίνακας 17. Το τμήμα του docker-compose.yml για τα WiFiMon ui, processor και secure processor

Ονομάζουμε τις υπηρεσίες ως ui, processor και secure-processor, χρησιμοποιούμε τις εικόνες που δημιουργήσαμε πριν, ανοίγουμε τις αντίστοιχες θύρες, προσθέτουμε τις υπηρεσίες στο δίκτυο webnet, φορτώνουμε χρησιμοποιώντας configs τα αντίστοιχα αρχεία ρυθμίσεων και χρησιμοποιώντας secrets το java keystore. Όλα τα δίκτυα, configs και secrets που χρησιμοποιούμε πρέπει επίσης να τα δηλώσουμε σε top-level επίπεδο.

Με την εντολή replicas ορίζουμε πόσα αντίγραφα της υπηρεσία θα τρέξουν. Η προεπιλεγμένη τιμή είναι 1 που σημαίνει ότι για κάθε υπηρεσία θα έχει ένα container. Η τιμή 0 σημαίνει ότι δεν θα ξεκινήσει κανένα container για την συγκεκριμένη υπηρεσία. Μια τιμή >1 σημαίνει ότι θα ξεκινήσουν πολλαπλά container για την ίδια υπηρεσία. Σε αυτή την περίπτωση το docker θα χρησιμοποιήσει το load-balancer του για να διανείμει τα container στους διαθέσιμους κόμβους (nodes) καθώς και για να μοιράσει τις εισερχόμενες συνδέσεις σε αυτά τα container.

Συνήθως θα χρησιμοποιήσουμε είτε το processor είτε το secure-processor και όχι και τα δύο συγχρόνως. Για αυτό βάλαμε στο ένα από τα δύο την τιμή replicas: 0.

5.5 ΤΕΛΕΥΤΑΙΑ ΒΗΜΑΤΑ

Αρχικά δημιουργούμε ένα νέο σμήνος με την εντολή «docker swarm init». Σε αυτό το σημείο, αν θέλουμε, μπορούμε να προσθέσουμε και άλλους υπολογιστές στο σμήνος. Η εντολή «docker swarm join-token (worker|manager)» θα εμφανίσει ένα

αναγνωριστικό το οποίο μπορούμε να εισάγουμε σε άλλους υπολογιστές για να προστεθούν στο σμήνος.

Στην συνέχεια μπορούμε να ξεκινήσουμε όλες τις υπηρεσίες με την εντολή «docker stack deploy -c docker-compose.yml wifimon». Μπορούμε να τις σταματήσουμε με την εντολή «docker stack rm wifimon».

Σε αυτό το σημείο παρατηρούμε κάποια προβλήματα:

Στο docker για windows, κατά την εκκίνηση κάποια container τερματίζουν με το μήνυμα “returned a non-zero code: 126”. Αυτό οφείλεται σε έλλειψη μνήμης. Μπορούμε να αυξήσουμε τη μνήμη που διατίθεται στο docker κάνοντας δεξί click στο εικονίδιο του docker στο system tray και επιλέγοντας settings->advanced.

Μετά από μερικά λεπτά λειτουργίας, όταν τα wifimon ui, processor ή secure processor προσπαθούν να επικοινωνήσουν με το postgres, η σύνδεση αποτυγχάνει και εμφανίζεται το ακόλουθο σφάλμα: «org.postgresql.util.PSQLException: This connection has been closed.». Αυτό οφείλεται στο ότι η tcp σύνδεση ανάμεσα στο postgres και το wifimon κλείνει μετά από 15 λεπτά αδράνειας, κάτι που παραμένει ένα ανοιχτό θέμα[1]. Ένας τρόπος να αντιμετωπίσουμε αυτό το πρόβλημα είναι χρησιμοποιώντας την ρύθμιση tcp_keepalives_idle του postgres. Η ρύθμιση αυτή καθορίζει μετά από πόσα δευτερόλεπτα αδράνειας στην TCP σύνδεση, θα στείλει ένα keepalive μήνυμα στον πελάτη. Επομένως μπορούμε να θέσουμε αυτή την ρύθμιση σε κάποιο χρόνο μικρότερο τον 15 λεπτών ώστε να μην φτάσουμε σε 15 λεπτά αδράνειας, κάτι που θα έκλεινε την σύνδεση. Μπορούμε να περάσουμε αυτήν την ρύθμιση στο postgres χρησιμοποιώντας την sql εντολή ALTER SYSTEM. Οπότε θα προσθέσουμε την εντολή

```
ALTER SYSTEM SET tcp_keepalives_idle = 600;
```

στο αρχείο init.sql που είχαμε δημιουργήσει για το postgres.

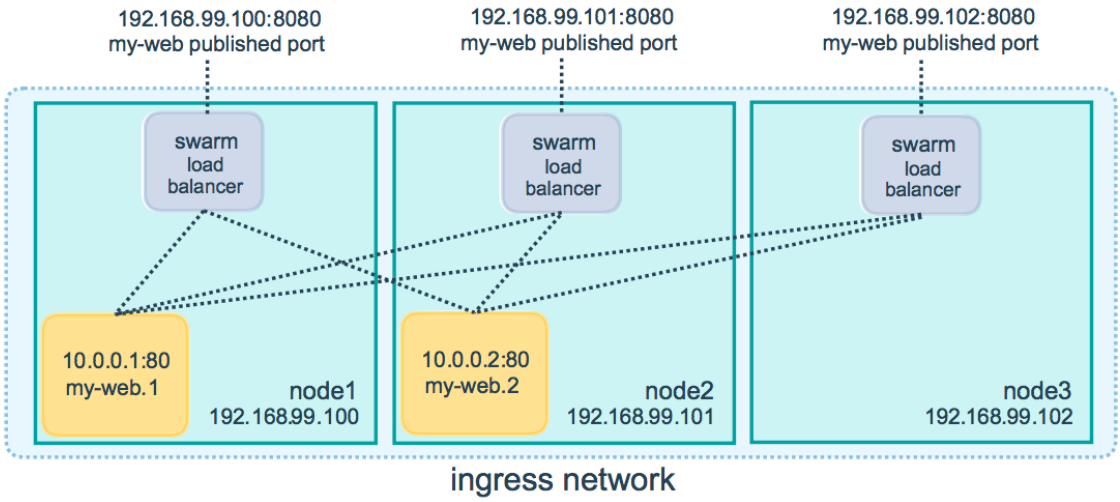
Ένα άλλο πρόβλημα είναι ότι τα wifimon ui, processor και secure processor δεν λαμβάνουν την πραγματική ip του χρήστη αλλά μια ip από το ingress δίκτυο η οποία χρησιμοποιήθηκε για να προωθηθεί η αίτηση. Σε αυτό το σημείο, προτού προχωρήσουμε στην εύρεση λύσης, είναι αναγκαίο να εξηγήσουμε πως δουλεύει το δίκτυο του docker.

Σε ένα docker σμήνος, όλοι οι κόμβοι συμμετέχουν σε ένα ingress πλέγμα δρομολόγησης (ingress routing mesh). Το πλέγμα δρομολόγησης επιτρέπει σε κάθε κόμβο του σμήνους να δεχθεί συνδέσεις σε δημοσιευμένες θύρες για οποιαδήποτε υπηρεσία τρέχει στο σμήνος, ακόμα και αν δεν τρέχει κάποιο αντίγραφο της υπηρεσίας στον συγκεκριμένο κόμβο. Το πλέγμα δρομολόγησης στην συνέχεια θα δρομολογήσει την εισερχόμενη αίτηση σε κάποιο από τα ενεργά container.

Για παράδειγμα, στην Εικόνα 10 έχουμε τρεις κόμβους (node1, node2, node3). Η υπηρεσία my-web ακούει στην θύρα 8080 και έχει δύο αντίγραφα (my-web.1, my-web.2) τα οποία τρέχουν στους κόμβος node1, node2 αντίστοιχα. Εισερχόμενες αιτήσεις στη θύρα 8080 στα node1, node2 ή node3 θα διαμοιραστούν στα container

[1] https://github.com/moby/moby/issues/31208

my-web.1 και my-web.2. Ακόμα και αν τα my-web.1 και my-web.2 βρίσκονταν και τα δύο στο node1 και δεν θα άλλαζε κάτι στην λειτουργία του δικτύου.



Εικόνα 10. Ingress πλέγμα δρομολόγησης

Το πρόβλημα που αντιμετωπίζουμε παραμένει ένα ανοιχτό θέμα[1] και μπορούμε να το αντιμετωπίσουμε μόνο παρακάμπτοντας το δίκτυο ingress. Μπορούμε να παρακάμψουμε το δίκτυο ingress θέτοντας τις θύρες σε mode: host, το οποίο έχει ως αποτέλεσμα κάθε container να συνδέεται άμεσα με την θύρα στο κόμβο στον οποίο τρέχει. Χρησιμοποιώντας ως παράδειγμα πάλι την εικόνα 2, αυτό θα σήμαινε ότι αιτήσεις στην θύρα 8080 του node1 θα προωθούνταν αποκλειστικά στο container my-web.1. Αιτήσεις στην θύρα 8080 του node2 θα προωθούνται αποκλειστικά στο container my-web.2. Αιτήσεις στην θύρα 8080 του node3 θα αγνοούνται αφού δεν υπάρχει κάποιο container που να ακούει σε αυτήν την θύρα στο node3. Επίσης δεν μπορούμε να έχουμε παραπάνω του ενός container που να ακούνε στην ίδια θύρα στον ίδιο κόμβο.

Αυτή την μειωμένη λειτουργικότητα μπορούμε να την αντιμετωπίσουμε σε κάποιο βαθμό θέτοντας την υπηρεσία σε mode: global που διασφαλίζει ότι θα τρέχει ένα container σε κάθε κόμβο.

Το τροποποιημένο κομμάτι για το wifimon-ui στο docker-compose.yml φαίνεται στον Πίνακας 17. Παρόμοια θα πρέπει να τροποποιηθούν και τα αντίστοιχα τμήματα για τα wifimon processor και secure processor.

```
services:
  ui:
    image: theodim/wifimon-ui:0.1.1
    ports:
      - target: 8441
        published: 8441
        mode: host
    networks:
      - webnet
```

[1] https://github.com/moby/moby/issues/25526

```
configs:
  - source: ui.properties
    target: /usr/lib/wifimon/config/ui.properties
secrets:
  - source: keystore.jks
    target: /usr/lib/wifimon/keystore/keystore.jks
deploy:
  mode: global
```

Πίνακας 18. Το τροποποιημένο τμήμα του docker-compose.yml για το Wifimon ui

ΚΕΦΑΛΑΙΟ 6: ΣΥΜΠΕΡΑΣΜΑΤΑ

ΣΥΜΠΕΡΑΣΜΑΤΑ

Στην παρούσα διπλωματική εργασία αρχικά μελετήσαμε το σύστημα containerization Docker. Παρουσιάσαμε τα βασικά στοιχεία του Docker όπως είναι το οι εικόνες, τα container, το daemon, το registry κ.τ.λ.

Στην συνέχεια εξετάσαμε τις τεχνολογίες που χρησιμοποιεί το docker για να υλοποιήσει τα container όπως οι χώροι ονομάτων, οι ομάδες ελέγχουν και τα union συστήματα αρχείων.

Έπειτα εξετάσαμε τα πλεονεκτήματα του Docker. Το Docker απλοποιεί αρκετά την διαδικασία αρκετά την διαδικασία εγκατάστασης αφού μπορούμε με μια μόνο εντολή και σε μερικά μόνο δευτερόλεπτα να έχουμε μια εφαρμογή σε λειτουργία, ασχέτως από το τι λειτουργικό σύστημα χρησιμοποιούμε. Το περιβάλλον της εφαρμογής παραμένει σταθερό σε όλα τα στάδια της κύκλου ζωής της, κάτι που οδηγεί σε ταχύτερη ανάπτυξη και παράδοση της εφαρμογής. Τα container έχουν αρκετά πλεονεκτήματα και στο τομέα της κλιμάκωσης και της απόδοσης. Μπορούμε να έχουμε πολλά περισσότερα container σε σύγκριση με τις εικονικές μηχανές καθώς και να τα κλιμακώσουμε εύκολα και γρήγορα από ένα σε μερικές εκατοντάδες και να τα ξανά μειώσουμε αργότερα όταν πια δεν τα χρειαζόμαστε. Το γεγονός ότι τα container δεν χρειάζονται να χρησιμοποιήσουν κάποιο hypervisor, τους επιτρέπει να αξιοποιούν καλύτερα τους διαθέσιμους πόρους. Επειδή τα container δεν χρησιμοποιούν ένα πλήρες λειτουργικό σύστημα, οι απαιτήσεις σε πόρους είναι μικρότερες σε σύγκριση με τις εικονικές μηχανές. Όλα τα παραπάνω πλεονεκτήματα καθιστούν το Docker μία άριστη επιλογή για τα περισσότερα συστήματα υπολογιστικού νέφους.

Στην συνέχεια εξετάσαμε το σύστημα μέτρησης της ποιότητας ενός WiFi δικτύου «WiFiMon» ώστε να αποκτήσουμε μια καλύτερη εικόνα του συστήματος το οποίο καλούμαστε να υλοποιήσουμε στο Docker.

Τέλος, υλοποιήσαμε σε Docker το σύστημα μέτρησης της ποιότητας ενός WiFi δικτύου «WiFiMon». Το Docker αποδείχτηκε εξαιρετικά εύκολο στην χρήση. Τα Dockerfile και docker-compose.yml είχαν αρκετά απλή μορφή και το documentation του Docker ήταν αρκετά λεπτομερές. Ως προαπαιτούμενες γνώσεις για την δημιουργία των εικόνων ήταν μια στοιχειώδη γνώση του συστήματος που θέλουμε να υλοποιήσουμε και των απαιτήσεων του καθώς και γνώση linux shell scripting ενώ η διαδικασία για την δημιουργία του Dockerfile της εικόνας θύμιζε σε μεγάλο βαθμό την διαδικασία που θα ακολουθούσαμε για να εγκαταστήσουμε την εφαρμογή χρησιμοποιώντας την γραμμή εντολών του linux.

ΚΕΦΑΛΑΙΟ 7: ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Η μελλοντική εργασία η οποία μπορεί να επιτελεστεί έχοντας σαν βάση την παρούσα διπλωματική εργασία περιλαμβάνει βελτιστοποιήσεις της τρέχουσας υλοποίησης ή εναλλακτικές υλοποιήσεις της. Μερικές τέτοιες ιδέες παρουσιάζονται παρακάτω.

Στην τρέχουσα υλοποίηση επιλέξαμε να χρησιμοποιήσουμε τα αρχεία .jar των WiFiMon ui, processor και secure-processor για την δημιουργία των εικόνων. Σε μια εναλλακτική υλοποίηση θα μπορούσαμε να δημιουργούμε τις εικόνες απευθείας από τον πηγαίο κώδικα του WiFiMon.

Οι εικόνες των WiFiMon ui, processor και secure-processor υποστηρίζουν την εισαγωγή ρυθμίσεων μόνο με την χρήση των αρχείων των αρχείων ui.properties, processor.properties και secure-processor.properties αντίστοιχα. Οι εικόνες αυτές θα μπορούσαν να δίνουν την επιλογή στον χρήστη να περάσει ρυθμίσεις χρησιμοποιώντας μεταβλητές περιβάλλοντος αντί των αρχείων ρυθμίσεων ή και σε συνδυασμό με αυτά.

Για να επιλύσουμε ένα πρόβλημα που αντιμετωπίσαμε χρειάστηκε να παρακάμψουμε το δίκτυο Ingress του Docker. Θα μπορούσαν να διερευνηθούν εναλλακτικές λύσεις που θα μας επέτρεπαν να αντιμετωπίσουμε το πρόβλημα αυτό χωρίς να θυσιάσουμε τα πλεονεκτήματα που προσφέρει το δίκτυο Ingress.

Αυτή την στιγμή η πρόσβαση στο Kibana δεν περιορίζεται, με αποτέλεσμα να μπορεί να έχει ο οποιοσδήποτε πρόσβαση σε αυτό, χωρίς να διαθέτει την κατάλληλη εξουσιοδότηση. Θα μπορούσαν να διερευνηθούν τρόποι περιορισμού της πρόσβασης στο Kibana όπως το searchguard και το x-pack.

Ορισμένες δυνατότητες του Docker παρουσιάζουν ενδιαφέρον αλλά δεν χρησιμοποιήθηκαν στην τρέχουσα υλοποίηση επειδή βρίσκονται ακόμη σε πειραματικό στάδιο. Στο μέλλον, όταν δεν θα βρίσκονται πια σε πειραματικό στάδιο, θα μπορούσε να επανεξεταστεί η χρήση τους.

Τέτοιες δυνατότητες είναι το Distributed Application Bundles[29] το οποίο θα επέτρεπε τον διαμοιρασμό του docker-compose.yml με τρόπο παρόμοιο με αυτόν που διαμοιράζονται οι εικόνες και το Docker Application Packages[30] που επιτρέπει να τοποθετηθούν σε διαφορετικό αρχείο οι ρυθμίσεις που χρησιμοποιεί το docker-compose.yml και πιθανόν να θέλει να αλλάξει ο χρήστης. Τέτοιες ρυθμίσεις περιλαμβάνουν το ποιες θύρες θα είναι ανοιχτές στα κοντέινερ και σε ποια τοποθεσία θα βρίσκονται το Java Keystore.

ΚΕΦΑΛΑΙΟ 8: ΑΝΑΦΟΡΕΣ

ΑΝΑΦΟΡΕΣ

[1] "Docker Documentation.", Docker Documentation, 2018. Available from: https://docs.docker.com. [13 March 2018].

[2] Boettiger, C. (2015). An introduction to docker for reproducible research. ACM SIGOPS Operating Systems Review, 49(1), 71-79.

[3] Joy, A. M. (2015). Performance comparison between linux containers and virtual machines. Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances In, 342 - 346. doi:10.1109/ICACEA.2015.7164727

[4] Vase, T. (2015). Advantages of Docker.

[5] Dua, R., Raja, A. R., & Kakadia, D. (2014). Virtualization vs Containerization to Support PaaS. 2014 IEEE International Conference on Cloud Engineering.

[6] Ismail, B. I., Mostajeran Goortani, E., Ab Karim, M. B., Ming Tat, W., Setapa, S., Luke, J. Y., & Hong Hoe, O. (2015). Evaluation of Docker as Edge computing platform. 2015 IEEE Conference on Open Systems (ICOS).

[7] Liu, D., & Zhao, L. (2014). The research and implementation of cloud computing platform based on docker. 2014 11th International Computer Conference on Wavelet Actiev Media Technology and Information Processing.

[8] Preeth E N, Mulerickal, F. J. P., Paul, B., & Sastri, Y. (2015). Evaluation of Docker containers based on hardware utilization. 2015 International Conference on Control Communication & Computing India (ICCC).

[9] Moreews, F., Sallou, O., & Ménager, H. (2015). BioShaDock: a community driven bioinformatics shared Docker-based tools registry. *F1000Research*, *4*.

[10] Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S., & Gall, H. C. (2017). An Empirical Analysis of the Docker Container Ecosystem on GitHub. 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR).

[11] Lukas Martinelli. Haskell Dockerfile Linter. Available from: https://github.com/lukasmartinelli/hadolint, accessed [20 Oct. 2018].

[12] Hung, L. H., Kristiyanto, D., Lee, S. B., & Yeung, K. Y. (2016). GUIdock: using Docker containers with a common graphics user interface to address the reproducibility of research. *PloS one*, *11*.4

[13] Yu, C., & Huan, F. (2015, December). Live migration of docker containers through logging and replay. In *Advances in Computer Science Research, International Conference on Mechatronics and Industrial Informatics*.

[14] Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., & Merle, P. (2017). Autonomic Vertical Elasticity of Docker Containers with ELASTICDOCKER. 2017 IEEE 10th International Conference on Cloud Computing (CLOUD).

[15] Guan, X., Wan, X., Choi, B.-Y., Song, S., & Zhu, J. (2017). Application Oriented Dynamic Resource Allocation for Data Centers Using Docker Containers. IEEE Communications Letters, 21(3), 504–507.

[16] Špaček, F., Sohlich, R., & Dulík, T. (2015). Docker as platform for assignments evaluation. *Procedia Engineering*.

[17] Nguyen, D.-T., Yong, C. H., Pham, X.-Q., Nguyen, H.-Q., Loan, T. T. K., & Huh, E.-N. (2016). An Index Scheme for Similarity Search on Cloud Computing using MapReduce over Docker Container. Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication - IMCOM '16.

[18] Xingtao, L., Yantao, G., Wei, W., Sanyou, Z., & Jiliang, L. (2016). Network virtualization by using software-defined networking controller based Docker. 2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference.

[19] Huaijun, W., Ling, T., Junhuai, L., & Zhe, G. (2017). Research and Implementation of Mobile Cloud Computing Offloading System Based on Docker Container. 2017 10th International Symposium on Computational Intelligence and Design (ISCID).

[20] V. Kokkinos, K. Stamos, N. Kanakis, K. Baumann, A. Wilson and J. Healy, "Wireless crowdsourced performance monitoring and verification: WiFi performance measurement using end-user mobile device feedback," *2016 8th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, Lisbon, 2016, pp. 432-437.

[21] Boomerang, "SOASTA/boomerang". Available from: https://github.com/SOASTA/boomerang. [09 June 2016]

[22] NetTest, "Google Code Archive". Available from: https://code.google.com/archive/p/nettest/. [09 June 2016]

[23] Postgresql.org, *PostgreSQL: The world's most advanced open source database*. Available at: https://www.postgresql.org/ [2 Sep. 2018].

[24] Elastic.co., *Open Source Search & Analytics · Elasticsearch | Elastic*. Available at: https://www.elastic.co/ [2 Sep. 2018].

[25] Elastic.co., *Kibana: Explore, Visualize, Discover Data | Elastic*. Available at: https://www.elastic.co/products/kibana [2 Sep. 2018].

[26] Spring, "Building an Application with Spring Boot". Available from: https://spring.io/guides/gs/spring-boot/. [09 June 2016]

[27] HIBERNATE, "Hibernate. Everything data.". Available from: http://hibernate.org/. [09 June 2016]

[28] Docker Documentation. *Best practices for writing Dockerfiles*. Available from: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/ [Accessed 2 Sep. 2018].

[29] Github, "Docker-ce:docker-stacks-and-bundles.md". Available from: https://github.com/docker/docker-ce/blob/master/components/cli/experimental/docker-stacks-and-bundles.md. [5 Sep. 2018]

[30] Github, "Docker/app: Make your Docker Compose applications reusable, and share them on Docker Hub ". Available from: https://github.com/docker/app. [5 Sep. 2018]