ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Διαδικτυακές εφαρμογές σε τεχνολογίες container: Μελέτη Περίπτωσης στο CERN

ΙΩΑΝΝΗΣ ΠΑΝΑΓΙΩΤΙΔΗΣ
ΑΜ 56310

ΞΑΝΘΗ, ΙΟΥΛΙΟΣ 2020

ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Διαδικτυακές εφαρμογές σε τεχνολογίες container: Μελέτη Περίπτωσης στο CERN

Επιβλέπων Καθηγητής: Κος Τσαουσίδης Βασίλειος

ΞΑΝΘΗ, ΙΟΥΛΙΟΣ 2020

ΕΥΧΑΡΙΣΤΙΕΣ

Θα επιθυμούσα να ευχαριστήσω τον καθηγητή κ. Βασίλειο Τσαουσίδη, ο οποίος ως επιβλέπων της παρούσας διπλωματικής, προσέφερε πολύτιμη βοήθεια στην περάτωσή της, με την ενδελεχή επιστημονική του καθοδήγηση και τις υποδείξεις του σε κάθε φάση της εκπόνησής της. Τέλος, ευχαριστώ την οικογένειά μου για την υποστήριξή τους καθόλη τη διάρκεια των σπουδών μου.

Πίνακας Εικόνων

Πίνακας Περιεχομένων

Περίληψη

Τα τελευταία χρόνια έχει παρατηρηθεί μια ραγδαία αύξηση στις απαιτήσεις χρηστών στο διαδίκτυο με αποτέλεσμα να τίθενται προβλήματα στην διαθεσιμότητα, τη συντήρηση και την επεκτασιμότητα των εφαρμογών. Η παραδοσιακή μονολιθική αρχιτεκτονική εφαρμογών δεν είναι αρκετά ευέλικτη για να αντιμετωπίζει αυτά τα προβλήματα, με συνέπεια η βιομηχανία να στρέφεται προς την τάση χρησιμοποίησης της αρχιτεκτονικής των microservices και των τεχνολογιών container. Στην παρούσα εργασία θα γίνει μια εισαγωγή στις containerized εφαρμογές, θα παρουσιαστούν διάφορες τεχνολογίες container και των τρόπων χρήσης τους καθώς και τα πλεονεκτήματα τους και θα εξηγηθούν νέοι τεχνολογικοί όροι. Επίσης, χρησιμοποιήθηκε ως μελέτη περίπτωσης (case study) η τρέχουσα υποδομή του CERN και η μετανάστευση μιας εφαρμογής, με αρχιτεκτονική των microservices, από τις παραδοσιακές τεχνολογίες εικονικών μηχανών σε νέες τεχνολογίες container.

Abstract

In the latest years, a massive increase of the user demand on the internet has been observed, resulting in issues in availability, maintenance and extensibility of applications. The traditional monolithic application architecture is not convenient enough to solve these issues, thus pushing the microservices application architecture and container technologies into the industry. In this present thesis, an introduction to containerized applications is made, different container technologies and their use, as well as their advantages, are mentioned, and new technology terms are explained. Furthermore, a case study of the current infrastructure of CERN and the migration of an application from the traditional virtual machine technologies to new container technologies is presented.

Ενότητα 1: Εισαγωγή

1.1 Το διαδίκτυο στην εποχή μας

Στη σημερινή εποχή, το διαδίκτυο έχει εξελιχθεί σε ένα από τα σημαντικότερα εργαλεία του 21ου αιώνα. Το κόστος και η δυσκολία στην προσβασιμότητα του έχουν μειωθεί δραστικά, γεγονός που το κάνει διαθέσιμο στην μεγαλύτερη μερίδα του παγκόσμιου πληθυσμού. Ταυτόχρονα, η χρήση του έχει ξεφύγει από την αρχική μορφή που είχε, η οποία σύνδεδε κεντρικούς υπολογιστές μεταξύ τους. Πλέον, οι περισσότερες συσκευές που συνδέονται στο διαδίκτυο είναι κινητές και μαζί με το Διαδίκτυο των Πραγμάτων (Internet of Things) έχουν δημιουργηθεί νέες δυνατότητες στο χώρο, πολλές εφαρμογές μεταφέρθηκαν στο διαδίκτυο αλλάζοντας ολόκληρες βιομηχανίες και πλέον βρίσκουν χρήση σε αυτό εταιρείες, οργανισμοί αλλά και κράτη. Πολλές εργασίες συνεχίζουν να αυτοματοποιηούνται και ολόκληροι κλάδοι εργασιών έχουν εξαφανιστεί ή έχουν αντικατασταθεί από κάτι νεότερο και πιο εκσυγχρονιστικό μέσω του διαδικτύου. Αυτό σημαίνει ότι όλο και περισσότεροι χρήστες χρησιμοποιούν το διαδίκτυο, όχι απλώς για να συλλέγουν πληροφορίες, αλλά για να δημιουργούν επιχειρήσεις, νέες εφαρμογές ή να προσφέρουν κάποιες υπηρεσίες. Αυτή η αυξημένη χρήση του διαδικτύου έχει δημιουργήσει την ανάγκη μεταφοράς πολλών εφαρμογών, οι οποίες παλιότερα ήταν εφαρμογές συγκεκριμένης χρήσης και εφαρμογές γραφείου, σε αυτό.

1.2 Το πρόβλημα

Αυτή η μεταφορά των εφαρμογών ωστόσο, δεν περιείχε κάποια αλλαγή αρχιτεκτονικής. Οι περισσότεροι προγραμματιστές προσπάθησαν να μιμηθούν την μονολιθική αρχιτεκτονική η οποία κυριάρχησε τις προηγούμενες δεκαετίες και έτσι να μεταναστεύσουν με αυτόν τον τρόπο τις εφαρμογές τους στο διαδίκτυο. Ωστόσο η έκθεση αυτών των εφαρμογών στο διαδίκτυο έχει προκαλέσει πολλά προβλήματα στη βιωσιμότητα και εξέλιξη τους. Η μονολιθική αρχιτεκτονική δεν είχε σχεδιαστεί με γνώμονα την κινητικότητα και το αυξημένο φορτίο που θα υπάρχει στις μέρες μας. Επίσης τα δεδομένα που μεταφέρονται πλέον πιάνουν τεράστιο όγκο και οι ίδιες εφαρμογές γίνονται όλο και πιο περίπλοκες, μεγάλες και δύσκολες ως προς την συντήριση τους και την ανάπτυξη τους. Πέραν της έρευνας που γίνεται όσον αφορά το πρωτόκολλό του διαδικτύου ώστε να ανταποκρίνεται στις απαιτήσεις της εποχής [1][2], σημαντικό είναι να γίνουν αλλαγές αρχιτεκτονικής και σε επίπεδο εφαρμογών.

1.3 Στόχος

Στόχος της συγκεκριμένης διπλωματικής είναι να παρουσιάσει τρόπους με τους οποίους μπορούν να επιλυθούν τα παραπάνω προβλήματα. Θα γίνει μια εισαγωγή στα containers, μιας τεχνολογίας η οποία έχει συμβάλλει σημαντικά στην μετεξέλιξη των εφαρμογών στο διαδίκτυο και τις ευκαιρίες που έφερε αυτή στην αλλαγή αρχιτεκτονικής που παρατηρείται στις

μέρες μας. Τα containers αποτελούν ένα απομωνομένο περιβάλλον χρήστη το οποίο είναι εύκολα φορητό, είναι ελαφρύ όσον αφορά τις απαιτήσεις πόρων του συστήματος που χρειάζεται και προσφέρει νέες δυνατότητες αλλά και πλεονεκτήματα συγκριτικά με τη χρήση εικονικών μηχανών.

Επίσης, όλο και περισσότεροι προγραμματιστές μεταναστεύουν τις εφαρμογές τους στις τεχνολογίες αυτές και υιοθετούν την αρχιτεκτονική των microservices. Τα microservices είναι μια αρχιτεκτονική η οποία αντικαθιστά την μονολιθική αρχιτεκτονική την οποία χρησιμοποιούσαν αλλά και χρησιμοποιούν ακόμα και σήμερα πολλές εφαρμογές. Με αυτήν την αρχιτεκτονική, ένα πολύπλοκο πρόγραμμα μπορεί να σπάσει σε μικρότερες οντότητες όπου η καθεμιά θα έχει ένα μόνο σκοπό και θα βρίσκεται στην απλούστερη μορφή της. Με αυτόν τον τρόπο, η δυσκολία επέκτασης, διαχείρισης αλλά και ενημέρωσης μιας εφαρμογής μειώνεται δραστικά καθώς οι απαιτούμενες αλλαγές μπορούν να γίνουν μεμονωμένα στην αντίστοιχη οντότητα. Τα containers, μαζί με την αρχιτεκτονική των microservices, προσφέρουν πολλά πλεονεκτήματα στην ανάπτυξη λογισμικού και συντήρηση αυτού και θα αναλυθούν διεξοδικά στην συγκεκριμένη εργασία.

Επιπλέον, θα αναφερθεί η υποδομή του CERN [3] και η χρήση που κάνει στις τεχνολογίες container. Το CERN, όπως και πολλοί άλλοι οργανισμοί και εταιρείες, βρίσκεται σε στάδιο μετανάστευσης (migration) των υπηρεσιών και εφαρμογών του στις τεχνολογίες αυτές. Ο λόγος για τον οποίο γίνεται αυτό, είναι ακριβώς για να αντιμετωπίζονται προβλήματα που προκύπτουν στις υπάρχουσες υποδομές με ευκολότερο τρόπο. Οι τεχνολογίες container βοηθούν στην αυτοματοποίηση πολλών εργασιών, με αποτέλεσμα να δημιουργούνται εργαλεία από τις ομάδες του CERN, τα οποία βοηθούν τους χρήστες τους να κάνουν ευκολότερα τη δουλειά τους. Για παράδειγμα, με τις τεχνολογίες αυτές, δίνεται η δυνατότητα σε χρήστες των υπηρεσιών του CERN κάνουν deploy τις εφαρμογές τους μέσω αρχείων παραμετροποίησης και χωρίς να χρειάζεται να ασχοληθούν καθόλου με την υποδομή (infrastructure) που χρειάζονται αυτές.

Πιο συγκεκριμένα, στην παρούσα εργασία θα αναφερθεί ως μελέτη περίπτωσης η μετανάστευση του Matomo σε τεχνολογίες containers, όπως αυτή πραγματοποιήθηκε κατά τη διάρκεια της πρακτικής μου στον οργανισμό. Το Matomo αποτελεί μια εφαρμογή ανοιχτού κώδικα που κρατάει στατιστικά στοιχεία για τις εφαρμογές και ιστοσελίδες που παρακολουθεί. Η συγκεκριμένη εφαρμογή είναι deployed με παραδοσιακές τεχνολογίες εικονικών μηχανών, ωστόσο έχει καταλήξει δύσκολη η συντήρηση και ενημέρωση αυτής. Ο σκοπός της πρακτικής μου ήταν να μεταφέρω την συγκεκριμένη υπηρεσία του CERN σε τεχνολογίες container, ώστε να επωφεληθεί η ομάδα από τα πλεονεκτήματα των container τεχνολογιών, να αυτοματοποιήσω πολλές διεργασίες που γίνονται σε αυτή και να την κάνω εύκολα διαχειρίσιμη από τους μελλοντικούς υπευθύνους της ομάδας.

1.4 Διάρθρωση Μελέτης

Στην δεύτερη ενότητα θα γίνει η εισαγωγή στις τεχνολογίες container, του Docker, του Container Orchestration αλλά και της αρχιτεκτονικής των Microservices. Στην τρίτη ενότητα θα αναφερθούν κομμάτια της υποδομής του CERN και του τρόπου με τον οποίο γίνεται η

μετανάστευση των εφαρμογών σε τεχνολογίες container. Στην τέταρτη ενότητα θα παρουσιαστεί η μελέτη περίπτωσης μετανάστευσης μιας εφαρμογής σε τεχνολογίες container όπως αυτή πραγματοποιήθηκε κατά τη διάρκεια της πρακτικής μου στον οργανισμό. Στο πέμπτο κεφάλαιο θα αναφερθούν τα συμπεράσματα αυτής της μετανάστευσης και των πλεονεκτημάτων των τεχνολογιών αυτών.

Ενότητα 2: Εισαγωγή στο Containerization

2.1 Σύντομη ιστορική αναδρομή

Η πρώτη ιδέα αυτού που αποκαλούμε σήμερα ως τεχνολογία container εμφανίστηκε αρχικά το 2000 ως FreeBSD jails [4], μια τεχνολογία που επέτρεπε τη διχοτόμηση (partitioning) ενός FreeBSD συστήματος σε πολλαπλά υποσυστήματα, τα jails. Τα jails αναπτύχθηκαν ως ασφαλή περιβάλλοντα τα οποία ένας διαχειριστής συστήματος μπορούσε να μοιραστεί με πολλούς χρήστες εντός και εκτός του οργανισμού ή της επιχείρησης.

Το 2001, μια υλοποίηση ενός απομονωμένου περιβάλλοντος εισήχθη στο Linux με το VServer project του Jacques Gélinas. Πάνω σε αυτό βασίστηκε και εξελίχθηκε στη συνέχεια η δημιουργία πολλαπλών ελεγχόμενων χώρων χρηστών (user spaces) στο Linux και έτσι τα κομμάτια άρχισαν να συναρμολογούνται ώστε να καταλήξουν σε αυτό που σήμερα αποκαλούμε Linux container.

Πολλές τεχνολογίες συνδυάστηκαν ώστε να φέρουν αυτήν την τεχνολογία στη μορφή που βρίσκεται σήμερα. Οι ομάδες ελέγχου [5] (control groups ή αλλιώς cgroups) είναι μια λειτουργία του πυρήνα (kernel) η οποία ελέγχει και βάζει όρια στη χρήση των πόρων μιας διεργασίας ή μιας ομάδας διεργασιών. Το systemd [6], ενα σύστημα αρχικοποίησης το οποίο ρυθμίζει το χώρο χρήστη (user space) και διαχειρίζεται τις διεργασίες, χρησιμοποιείται από τις ομάδες ελέγχου για να παρέχει μεγαλύτερο έλεγχο πάνω στις απομονωμένες διεργασίες. Αυτές οι δύο τεχνολογίες ήταν σημαντικές ώστε να δημιουργηθεί η έννοια του container και να επιτευχθεί ο διαχωρισμός των περιβαλλόντων τους με αποτελεσματικό τρόπο.

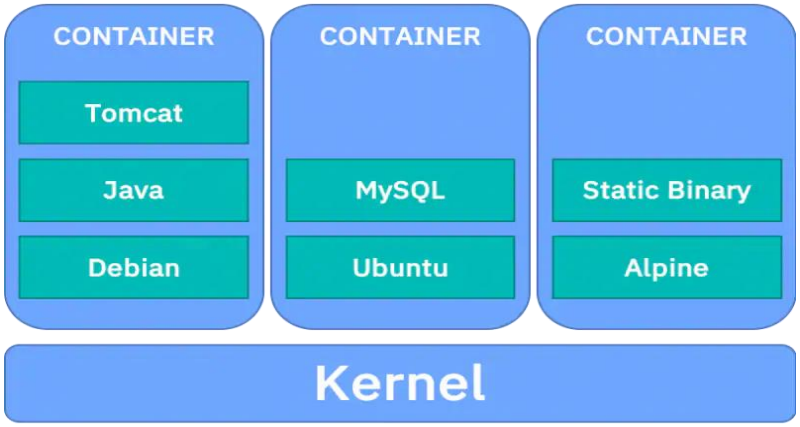
Η εισαγωγή του Docker:

Το 2008 έκανε την εμφάνιση του το Docker (υπό την ονομασία dotCloud [7]) με την τεχνολογία containers. Η τεχνολογία Docker πρόσθεσε νέες έννοιες και έφερε πολλά χρήσιμα εργαλεία: Μια απλή διεπαφή γραμμής εντολών (command line interface) για να εκτελούνται και να χτίζονται νέες εικόνες (images), ένα server daemon, μια βιβλιοθήκη με προϋπάρχουσες εικόνες container, και την έννοια του registry server, ενός server στον οποίο θα αποθηκεύονται οι εικόνες container. Οι παραπάνω τεχνολογίες συνδυαστικά βοήθησαν τους χρήστες να δημιουργούν νέα containers και να τα μοιράζονται.

2.2 Τι είναι το container;

Ένα Linux Container είναι ένα σετ από μια ή περισσότερες διεργασίες οι οποίες είναι απομονωμένες από το υπόλοιπο σύστημα. Όλα τα αρχεία που είναι απαραίτητα για να εκτελεστεί μια διεργασία ή λογισμικό παρέχονται από μια ξεχωριστή εικόνα (image) η οποία μεταφέρεται εύκολα μεταξύ διαφορετικών συστημάτων και υποδομών και μένει αναλλοίωτη κατά την ανάπτυξη του λογισμικού. Η αξιοποίηση των containers κάνει την ανάπτυξη λογισμικού και το testing αυτού πολύ γρηγορότερη από ότι χρησιμοποιώντας παραδοσιακές

μεθόδους ανάπτυξης λογισμικού. Επίσης, λόγω της δημοτικότητας τους και της ευκολίας χρήσης τους, τα containers έχουν γίνει πολύ σημαντικό κομμάτι και της ασφάλειας των υπολογιστών.



Εικόνα 1: Αρχιτεκτονική Container [8]

2.3 Docker

Το Docker [9] είναι μια πλατφόρμα λογισμικού ανοιχτού κώδικα [10] (open source) που υλοποιεί εικονικοποίηση (virtualization) σε επίπεδο λειτουργικού συστήματος. Ουσιαστικά το Docker προσφέρει αυτοματοποιημένες διαδικασίες για την ανάπτυξη εφαρμογών σε απομονωμένες περιοχές χρήστη (user spaces), δηλαδή σε containers. Το λογισμικό χρησιμοποιεί τεχνολογιες του πυρήνα (kernel) του Linux όπως τα cgroups και namespaces πυρήνα (kernel namespaces), για να επιτρέπει σε ανεξάρτητα software containers να εκτελούνται στο ίδιο λειτουργικό σύστημα. Έτσι αποφεύγεται η χρήση επιπλέον υπολογιστικών πόρων που θα απαιτούσε μια εικονική μηχανή (virtual machine). Το Docker δημιουργήθηκε από την εταιρία Docker Inc (αρχικά ως dotCloud). [11].

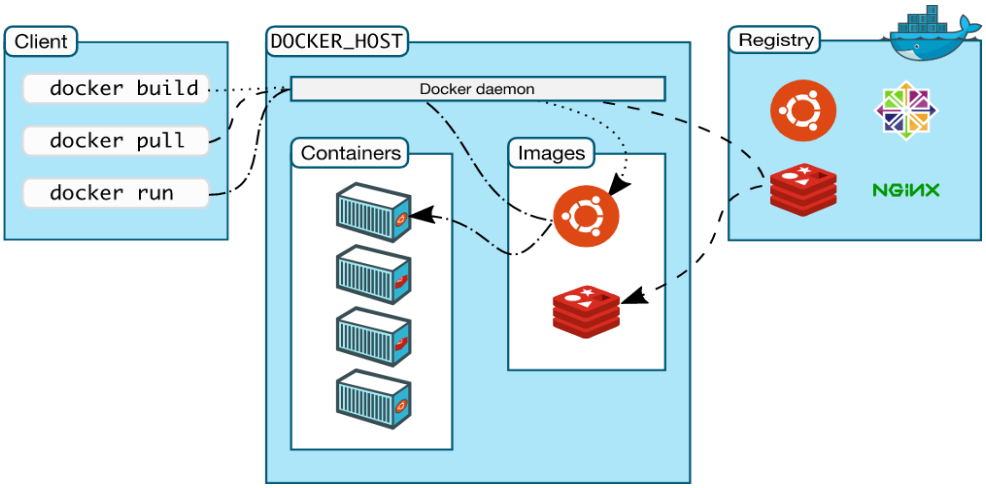
2.3.1 Docker Engine

Το Docker Engine [12] είναι μια client-server εφαρμογή με τα παρακάτω βασικά δομικά στοιχεία:

- Ένας server ο οποίος είναι ένα πρόγραμμα που τρέχει συνεχώς και ονομάζεται daemon διεργασία.
- Ένα REST API το οποίο ορίζει τις διεπαφές με τις οποίες τα προγράμματα μπορούν να επικοινωνούν με το daemon.
- Μια διεπαφή γραμμής εντολών (command line interface ή CLI). Το CLI χρησιμοποιεί το REST API με το οποίο γίνεται η επικοινωνία με τον daemon.

2.3.2 Αρχιτεκτονική Docker

Η αρχιτεκτονική Docker [12] συνοψίζεται στην παρακάτω εικόνα:



Εικόνα 2: Αρχιτεκτονική Docker [12]

Docker Client: Αποτελεί τον κύριο τρόπο αλληλεπίδρασης των χρηστών με το Docker. Όταν χρησιμοποιούνται εντολές (π.χ. docker run), ο client στέλνει τις εντολές στον daemon, ο οποίος και τις εκτελεί. Η εντολή docker χρησιμοποιεί το API που παρέχεται από το Docker. Ο client μπορεί να επικοινωνεί με παραπάνω από ένα daemon.

Docker Daemon: Ο daemon χειρίζεται αιτήματα που αφορούν το Docker API και τα αντικείμενα Docker όπως εικόνες (images), containers και δίκτυα. Μπορεί επίσης να επικοινωνεί με άλλους daemon με σκοπό να διαχειριστεί άλλες Docker υπηρεσίες.

Docker Registry: Ένα Docker registry περιέχει εικόνες Docker. Το Docker hub είναι ένα ανοιχτό registry προσβάσιμο από όλους. Χρησιμοποιώντας τις εντολές «docker pull» ή «docker run», οι απαιτούμενες εικόνες αντλούνται από το registry. Με την εντολή «docker push», η εικόνα που έχει δημιουργηθεί, ανεβαίνει στο συγκεκριμένο registry.

Docker Image: Αποτελεί ένα υπόδειγμα με οδηγίες για τη δημιουργία Docker containers. Συχνά μία εικόνα βασίζεται σε άλλη εικόνα με κάποιες επιπλέον αλλαγές και ρυθμίσεις. Ο κάθε χρήστης μπορεί να δημιουργήσει τις δικές του εικόνες με τη χρήση ενός Dockerfile ή μπορεί να χρησιμοποιήσει εικόνες τρίτων από κάποιο Docker registry. Το Dockerfile είναι ένα αρχείο κειμένου, που περιέχει όλες τις εντολές που θα μπορούσε να τρέξει ένας χρήστης σε μια γραμμή εντολών, για να δημιουργήσει ένα image.

Containers: Μόλις εκτελεστεί μια εικόνα docker (docker image), δημιουργείται ένα docker container.

Volumes: Τα δεδομένα που δημιουργούνται από το docker και χρησιμοποιούνται από τα containers αποθηκεύονται σε volumes.

Services: Τα services επιτρέπουν την επέκταση των containers σε πολλά διαφορετικά docker daemons. Το orchestration αυτό γίνεται με τεχνολογίες όπως Docker Swarm και Kubernetes, από τα οποία το δεύτερο θα αναλυθεί παρακάτω.

Namespaces: Το Docker χρησιμοποιεί μια τεχνολογία γνωστή ως namespaces για τη παροχή ενός απομονωμένου περιβάλλοντος που ονομάζεται container. Όταν το container τρέχει, το Docker δημιουργεί ένα σύνολο από namespaces για αυτό τα οποία παρέχουν ένα στρώμα απομόνωσης. Κάθε πτυχή ενός container τρέχει σε ξεχωριστό namespace και η πρόσβασή του περιορίζεται σε αυτό το namespace.

Το Docker engine χρησιμοποιεί Linux namespaces, όπως τα ακόλουθα:

- **pid:** Απομόνωση διεργασιών (PID: Process ID)
- **net:** Διαχείριση διεπαφών δικτύου (NET: Networking)
- **ipc:** Διαχείριση πρόσβασης σε IPC πόρους (IPC: InterProcess Communication)
- **mnt:** Διαχείριση σημείων συστήματος αρχείου (MNT: Mount)
- **uts:** Απομόνωση πυρήνα και αναγνωριστικά έκδοσης (UTS: Unix Timesharing System)

2.4 Χρησιμότητα των Containers

Έστω ότι μια ομάδα εργάζεται πάνω στην ανάπτυξη ενός λογισμικού. Το κάθε μέλος της ομάδας μπορεί να διαθέτει ένα τελείως διαφορετικό περιβάλλον εργασίας. Κάποιος μπορεί να εργάζεται στον προσωπικό του υπολογιστή ενώ άλλοι στους σταθερούς υπολογιστές του γραφείου εργασίας τους. Το κάθε προσωπικό περιβάλλον εργασίας μπορεί να διαθέτει πολύ συγκεκριμένες ρυθμίσεις παραμέτρων και να απαρτίζεται από διαφορετικές βιβλιοθήκες, εξαρτήσεις ή αρχεία. Αντίθετα, το περιβάλλον εργασίας στον εργασιακό χώρο μπορεί να περιέχει διαφορετικά αρχεία ή βιβλιοθήκες, να περιέχει ένα τυποποιημένο σύστημα εργασίας, μπορεί ακόμα και να χρησιμοποιεί τελείως διαφορετικό λειτουργικό σύστημα στους υπολογιστές του. Αυτό σημαίνει ότι η ανάπτυξη λογισμικού σε κάθε ένα από αυτα τα διαφορετικά περιβάλλοντα εξαρτάται από το αντίστοιχο τοπικό περιβάλλον του καθενός. Ιδανικά θα μπορούσε να γίνει προσπάθεια κάθε περιβάλλον εργασίας να είναι ακριβώς το ίδιο, για παράδειγμα το προσωπικό περιβάλλον εργασίας να μιμηθεί αυτό των υπολογιστών πάνω στο οποίο εργάζονται οι υπόλοιποι. Αλλά αυτό δεν είναι πάντα εύκολο, ειδικά όταν θέλουμε να γλιτώσουμε όλο το overhead της δημιουργίας του περιβάλλοντος εργασίας των servers. Η λύση στο παραπάνω πρόβλημα είναι η χρησιμοποίηση των containers.

Το container το οποίο περιέχει την εφαρμογή, ταυτόχρονα περιέχει όλες τις απαραίτητες βιβλιοθήκες, εξαρτήσεις και αρχεία, οπότε μπορεί να μεταφερθεί σε διαφορετικά περιβάλλοντα χωρίς πρόβλημα και χωρίς να προκύπτουν όλες οι παρενέργειες που προκύπτουν σε διαφορετικές περιπτώσεις όταν αλλάζει το περιβάλλον εργασίας. Μια εικόνα container μπορεί

να παρομοιαστεί με την εγκατάσταση μιας διανομής Linux γιατί εμπεριέχει με τον ίδιο τρόπο τα πακέτα RPM, αρχεία κλπ. Ωστόσο η διανομή εικόνων container γίνεται πολύ πιο εύκολα από ότι η εγκατάσταση μιας διανομής Linux.

Φυσικά αυτό αποτελεί απλά το πιο συνηθισμένο παράδειγμα χρήσης containers. Τα Linux containers μπορούν να χρησιμοποιηθούν με διαφορετικούς τρόπους για να λυθούν προβλήματα όπου χρειάζεται portability, configurability και απομόνωση των εφαρμογών. Ο σκοπός των containers είναι να διευκολύνουν και να κάνουν πιο γρήγορη την ανάπτυξη λογισμικού (development) και να αντιμετωπίζουν όλα τα προβλήματα τα οποία πολλές φορές προκύπτουν λόγω διαφοράς περιβάλλοντος εργασίας. Επιπλέον τα containers έχουν συμβάλει σημαντικά και στην ανάπτυξη συστημάτων που χρησιμοποιούν microservices και γενικότερα την DevOps φιλοσοφία.

2.5 Διαφορές των containers με τις εικονικές μηχανές

2.5.1 Εικονικοποίηση (Virtualization)

Η εικονική μηχανή (virtual machine) είναι μια εξομοίωση ενός υπολογιστή μέσα στον υπολογιστή μας, σε επίπεδο υλισμικού (hardware). Έχει ακριβώς τις λειτουργίες ενός φυσικού υπολογιστή: κάνει boot, επανεκκίνηση, και τερματισμό λειτουργίας. Μπορούν να συνδεθούν κανονικά εξωτερικές συσκευές όπως θα γινόταν σε έναν φυσικό υπολογιστή, από usb μέχρι και εκτυπωτές. Διαθέτει δικό του BIOS ή UEFI και έχει εγκατεστημένο το δικό του λειτουργικό σύστημα.

Ο όρος «εικονικοποίηση» αναφέρεται στην δυνατότητα ταυτόχρονης εκτέλεσης πολλών τέτοιων εικονικών μηχανών σε μία φυσική μηχανή. Ουσιαστικά, η εικονική μηχανή αποτελεί λογισμικό προσομοίωσης ενός φυσικού συστήματος.

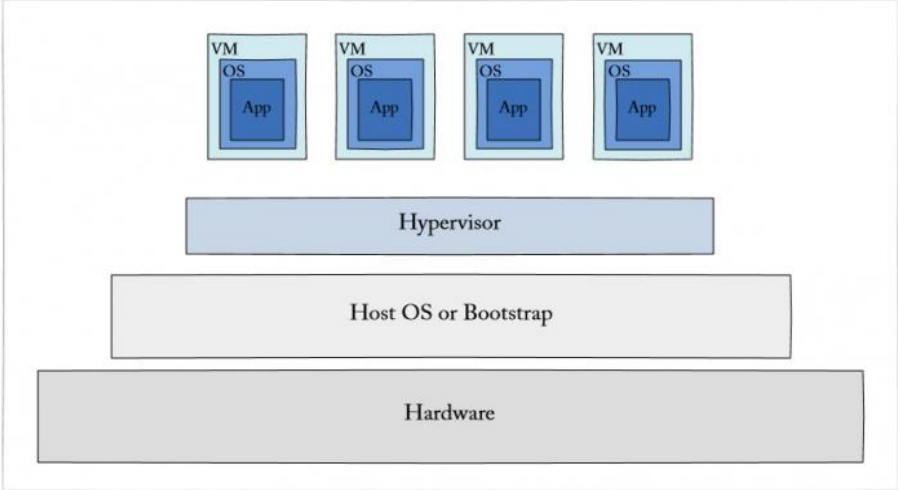
Για μια λύση εικονικοποίησης χρειάζονται τα εξής στοιχεία [13]:

- Κατάλληλος εξοπλισμός hardware
- Λογισμικό εικονικοποίησης (hypervisor)
- Λογισμικό διαχείρισης

Με την εικονικοποίηση το hardware διαχωρίζεται από το software (λειτουργικό σύστημα και εφαρμογές). Ο hypervisor είναι ένα νέο επίπεδο εικονικοποίησης μεταξύ του hardware και του software, που ενοποιεί τους φυσικούς πόρους και τους διαμοιράζει στα εικονικά συστήματα με τρόπο διάφανο. Τα εικονικά συστήματα συνεχίζουν να νομίζουν ότι επικοινωνούν απευθείας με το hardware, αλλά στην πραγματικότητα επικοινωνούν με το επίπεδο εικονικοποίησης. Αυτό επιτρέπει τη μετακίνηση επεξεργαστικής ισχύος, μνήμης ή και αποθηκευτικού χώρου από μια εικονική μηχανή σε άλλη κατα τη διάρκεια λειτουργίας και

σύμφωνα με τις εκάστοτε ανάγκες. Το αποτέλεσμα είναι η καλύτερη εκμετάλλευση των πόρων συνολικά, μεγάλη εξοικονόμηση ενέργειας και φυσικού χώρου και ευκολότερη διαχείριση της υποδομής.

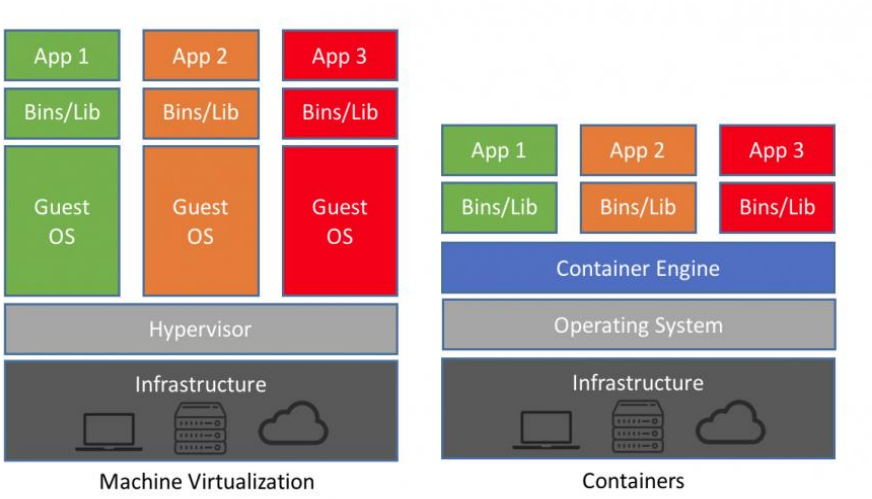
Άλλα πλεονεκτήματα που παρέχει η εικονικοποίηση είναι η υψηλή διαθεσιμότητα, τα στιγμιότυπα (snapshots) με τα οποία γίνεται εύκολα η μεταφορά της κατάστασης μιας υπάρχουσας εικονικής μηχανής σε ένα άλλο μηχάνημα, η εύκολη διαχείριση του περιβάλλοντος χρησιμοποιώντας τον hypervisor, η δημιουργία λύσεων disaster recovery και η συνεχή μείωση του κόστους και της ενέργειας που χρησιμοποιούν τα data centers.



Εικόνα 3: Cloud Virtualization [14]

2.5.2 Σύγκριση

Πολύ συχνά τα containers συγκρίνονται με τα virtual machines καθώς και τα δυο επιτρέπουν λογισμικό να τρέχει μέσα σε ένα απομονωμένο περιβάλλον. Η τεχνολογία containers αποτελεί μια εικονικοποίηση σε επίπεδο λειτουργικού συστήματος, σε αντίθεση με τα virtual machines όπου αποτελούν εικονικοποίηση σε επίπεδο hardware. Στην ουσία δεν πρόκειται για δύο τεχνολογίες που ανταγωνίζονται μεταξύ τους, αλλά για δυο τεχνολογίες που μπορούν τόσο να χρησιμοποιηθούν για να λύσουν τα ίδια προβλήματα, όσο και να χρησιμοποιηθούν συμπληρωματικά η μια με την άλλη. Ωστόσο κάποιες συγκρίσεις μπορούν να γίνουν, ειδικά όταν χρησιμοποιούνται για σαν λύση αρχιτεκτονικής του εκάστοτε συστήματος ή της εφαρμογής.



Εικόνα 4: Containers vs Virtual Machines [15]

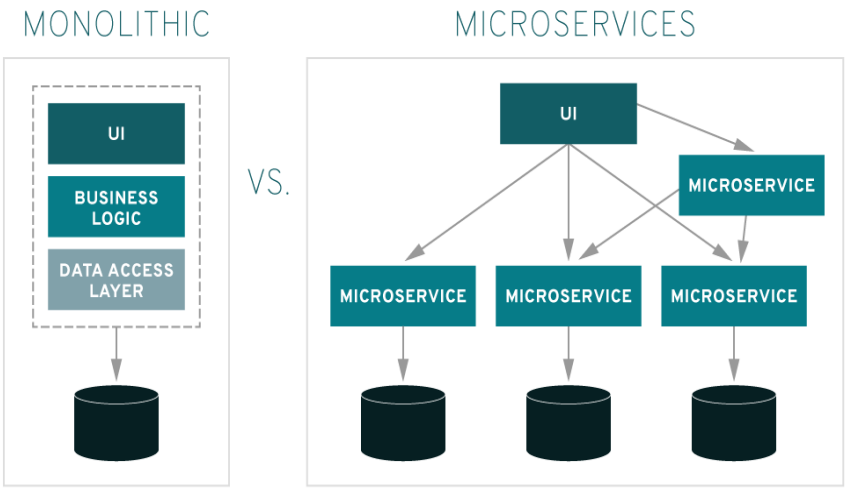
- Η εικονικοποίηση δίνει τη δυνατότητα να τρέχουν πολλά λειτουργικά συστήματα παράλληλα σε ένα συγκεκριμένο σύστημα hardware.
- Τα containers μοιράζονται τον ίδιο πυρήνα (kernel) λειτουργικού συστήματος και απομονώνουν τις διεργασίες εφαρμογών από το υπόλοιπο σύστημα.

2.5.3 Πλεονεκτήματα της χρήσης Containers

- **Φορητότητα**: Ένα container δημιουργεί ένα εκτελέσιμο πακέτο λογισμικού το οποίο είναι ανεξάρτητο από το λειτουργικό σύστημα του κεντρικού υπολογιστή και συνεπώς μπορεί να τρέξει ως έχει και με συνέπεια σε όλα τα συστήματα και πλατφόρμες υπολογιστικού νέφους.
- **Ευελιξία**: Η ανοιχτού κώδικα μηχανή Docker προσφέρει πολλά εργαλεία ανάπτυξης και το μέγεθος της κοινότητας των containers είναι μεγάλο με αποτέλεσμα να υπάρχουν διαφορετικές εικόνες Docker έτοιμες προς άμεση χρήση. Οι μηχανικοί ανάπτυξης λογισμικού μπορούν να χρησιμοποιούν Agile ή DevOps εργαλεία και διαδικασίες για γρήγορη ανάπτυξη και ενημέρωση λογισμικού.
- **Ταχύτητα**: Τα containers είναι «ελαφριά» με την έννοια ότι μοιράζονται τον πυρήνα (kernel) του λειτουργικού συστήματος (OS) και δεν περιέχουν αυτό το επιπλέον overhead. Ξεκινάνε πολύ γρηγορότερα από ότι οι εικονικές μηχανές.
- **Πλήρης απομόνωση**: Κάθε containerized εφαρμογή είναι απομονωμένη και λειτουργεί ανεξάρτητα από άλλες. Το σφάλμα που μπορεί να επηρεάσει ένα container, δεν επηρεάζει άλλα containers. Ως συνέπεια, οι ομάδες ανάπτυξης λογισμικού μπορούν να εντοπίσουν και να διορθώσουν τα σφάλματα τεχνικής φύσεως χωρίς να προκαλέσουν προβλήματα ή downtime σε άλλα containers.

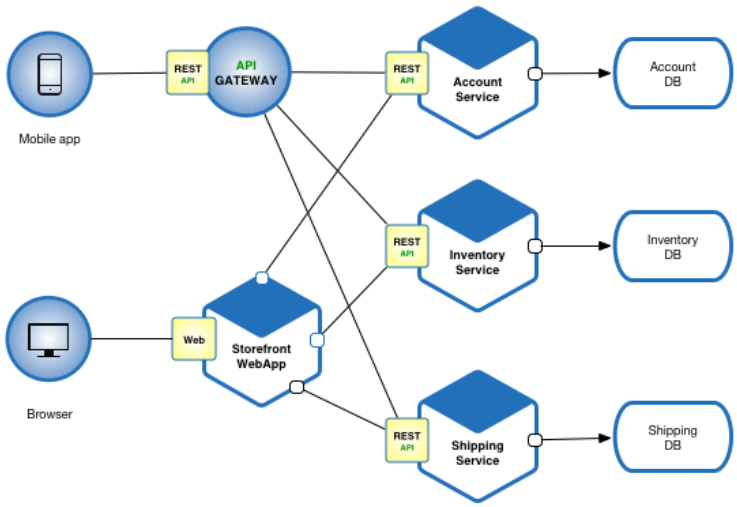
- **Αποδοτικότητα**: Το λογισμικό που τρέχει σε containers μπορεί να μοιράζεται τον πυρήνα (kernel) του λειτουργικού συστήματος του κεντρικού υπολογιστή αλλά και να μοιράζεται επίπεδα εφαρμογής με άλλα containers. Αυτό κάνει τα containers πολύ πιο μικρά από τις εικονικές μηχανές, ξεκινούν πολυ γρηγορότερα, και δίνεται η δυνατότητα να τρέχουν περισσότερα containers από ότι εικονικές μηχανές με τους ίδιους πόρους συστήματος, με αποτέλεσμα να αυξάνεται η αποδοτικότητα και να μειώνεται το κόστος του server αλλά και των αδειών χρήσης. Επίσης, τα containers χρησιμοποιούν λιγότερους πόρους (resources) και μοιράζονται πόρους μεταξύ τους. Αυτό σημαίνει ότι αν μια εφαρμογή δεν χρησιμοποιείται, οι αντίστοιχοι πόροι μπορούν να χρησιμοποιηθούν από άλλα containers και άλλες εφαρμογές.
- **Ευκολία διαχείρισης**: Μια πλατφόρμα ενορχήστρωσης containers αυτοματοποιεί την εγκατάσταση, το scaling και τη διαχείριση του φόρτου εργασίας και των services. Τέτοιες πλατφόρμες ενορχήστρωσης διευκολύνουν τις εργασίες διαχείρισης, όπως για παράδειγμα το scaling των containerized εφαρμογών, το rolling out νέων εκδόσεων των εφαρμογών και παρέχουν διάφορες λειτουργίες όπως το monitoring, το debugging, το logging εφαρμογών κλπ. Το πιο γνωστό σύστημα ενορχήστρωσης container είναι το Kubernetes, μια τεχνολογία ανοιχτού κώδικα προερχόμενη από το Borg, ένα εσωτερικό πρότζεκτ της Google, και αυτοματοποιεί όλες τις λειτουργίες των containers.
- **Ασφάλεια**: Η απομόνωση των εφαρμογών σε containers εμποδίζει την εισαγωγή κακόβουλου λογισμικού από το να επηρεάσει άλλα containers ή το κεντρικό σύστημα. Επιπλέον, δίνεται η δυνατότητα να οριστούν εξουσιοδοτήσεις (permissions) ασφάλειας ώστε να περιορίζεται η επιτρεπτή επικοινωνία διαφόρων τμημάτων των containers και να εμποδίζεται η εισαγωγή ανεπιθύμητων λειτουργιών σε εφαρμογές.

2.6 Microservices



Εικόνα 5: Monolithic vs Microservices [16]

Οι εταιρείες λογισμικού υιοθετούν τα microservices [16] σαν μια καλύτερη προσέγγιση στην ανάπτυξη και διαχείριση λογισμικού, σε σχέση με το παλαιότερο επικράτουμενο μοντέλο μονολιθικής αρχιτεκτονικής που συνδυάζει μια εφαρμογή λογισμικού με μια συνδεδεμένη διεπαφή χρήστη και μια βάση δεδομένων η οποία είναι συνδεδεμένη με μια μονάδα σε μια συγκεκριμένη πλατφόρμα server. Αντίθετα, με την αξιοποίηση των microservices, μια πολύπλοκη εφαρμογή διασπάται σε μια σειρά από μικρότερα, πιο εξειδικευμένα services, τα οποία έχουν τη δική τους βάση δεδομένων και το δικό τους business logic. Τα microservices στη συνέχεια επικοινωνούν μεταξύ τους με συνηθισμένες διεπαφές (όπως τα APIs) και REST διεπαφές (όπως το HTTP). Χρησιμοποιώντας microservices, οι ομάδες ανάπτυξης λογισμικού μπορούν να συγκεντρωθούν στην ενημέρωση συγκεκριμένων στοιχείων σε μια εφαρμογή χωρίς να την επηρεάζουν ολόκληρη, το οποίο επιτυγχάνει γρηγορότερη ανάπτυξη, testing και deployment.



Εικόνα 6: Αρχιτεκτονική Microservice [17]

Οι έννοιες πίσω από το containerization και τα microservices είναι παρόμοιες καθώς και τα δύο είναι μέθοδοι ανάπτυξης λογισμικού οι οποίες ουσιαστικά μετατρέπουν εφαρμογές σε συλλογές από μικρότερα services ή τμήματα τα οποία είναι φορητά, scalable, αποδοτικά και ευκολότερα στη διαχείρισή τους.

Επιπλέον, τα microservices και το containerization δουλεύουν πολύ καλά όταν συνεργάζονται μαζί. Τα containers παρέχουν μια ελαφριά ενθυλάκωση οποιασδήποτε εφαρμογής, είτε είναι μια παραδοσιακή εφαρμογή μονολιθικής αρχιτεκτονικής, είτε ένα δομοστοιχειωτό microservice. Ένα microservice, το οποίο έχει αναπτυχθεί εντός ενός container, στη συνέχεια αποκτά όλα τα πλεονεκτήματα του containerization, φορητότητα και αποφυγή του κλειδώματος σε προμηθευτή (vendor lock-in), όπως και ευελιξία ανάπτυξης, απομόνωση σφαλμάτων, αποδοτικότητα των servers, αυτοματοποίηση της εγκατάστασης, του scaling και της διαχείρισης, αλλά και τα επίπεδα της ασφάλειας που προσφέρει το containerization.

Οι σημερινές επικοινωνίες μετακινούνται συνεχώς προς το υπολογιστικό νέφος (cloud) όπου οι χρήστες μπορούν να αναπτύξουν εφαρμογές γρήγορα και αποδοτικά. Οι εφαρμογές και τα δεδομένα που βρίσκονται στο υπολογιστικό νέφος είναι προσβάσιμα από οποιαδήποτε συσκευή είναι συνδεδεμένη στο διαδίκτυο, επιτρέποντας τα μέλη μιας ομάδας να εργάζονται εξ αποστάσεως αλλά και εν κινήσει. Οι προμηθευτές υπηρεσιών υπολογιστικού νέφους (Cloud

Service Providers) χειρίζονται την υποδομή που βρίσκεται από πίσω, το οποίο γλιτώνει από τους διάφορους οργανισμούς και εταιρείες το κόστος των servers και άλλου εξοπλισμού και επίσης παρέχουν αυτοματοποιημένα αντίγραφα ασφαλείας για επιπρόσθετη αξιοπιστία. Οι υποδομές υπολογιστικού νέφους προσαρμόζονται κατ' απαίτηση (on demand) και μπορούν να αλλάζουν δυναμικά υπολογιστικούς πόρους και υποδομή καθώς αλλάζουν οι τρέχουσες απαιτήσεις. Επιπλέον, οι προμηθευτές υπολογιστικού νέφους ενημερώνουν πολύ συχνά τις προσφορές τους, με αποτέλεσμα οι χρήστες να έχουν πρόσβαση συνεχώς στην τελευταία λέξη της τεχνολογίας.

Τα containers, τα microservices και το υπολογιστικό νέφος συνεργάζονται μαζί για να φέρουν την ανάπτυξη και παράδοση λογισμικού σε νέα επίπεδα που δεν ήταν δυνατά με τις παραδοσιακές μεθοδολογίες και τα παλιότερα περιβάλλοντα εργασίας. Αυτές οι προσεγγίσεις νέας γενιάς προσθέτουν ευελιξία, αποδοτικότητα, αξιοπιστία και ασφάλεια στον κύκλο εργασίας της ανάπτυξης λογισμικού, τα οποία οδηγούν σε γρηγορότερη ταχύτητα στην παράδοση λογισμικού και βελτιώσεων στους χρήστες αλλά και την αγορά.

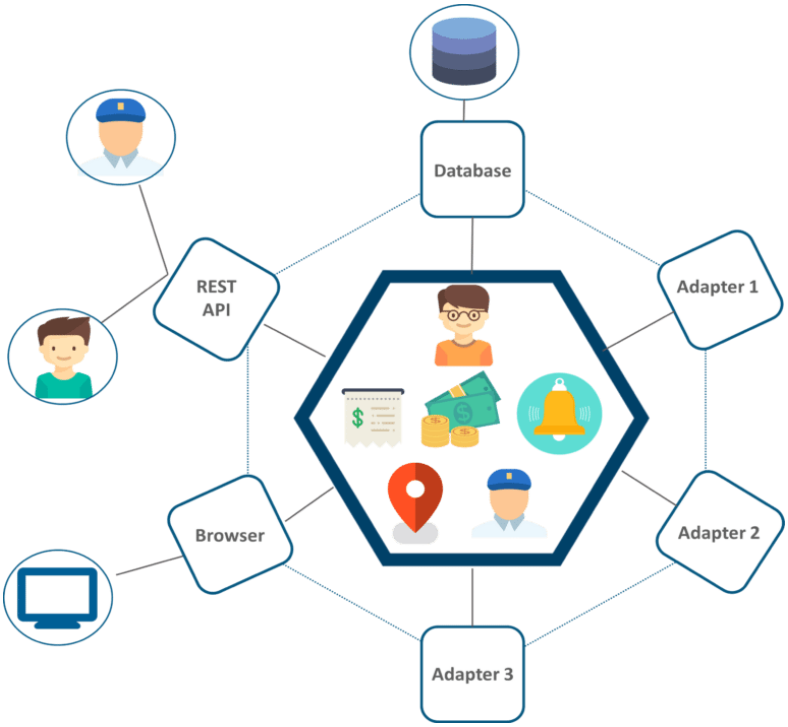
Πολλές είναι οι εταιρείες που έχουν ήδη μεταφερθεί πλήρως στην microservices αρχιτεκτονική.

2.6.1 Το παράδειγμα της Uber

Η Uber είναι μία αμερικανική επιχείρηση παροχής υπηρεσιών με έδρα στο Σαν Φρανσίσκο. Προσφέρει online παροχές υπηρεσιών μεταφοράς προσώπων σε πολλές πόλεις παγκοσμίως.

Η εφαρμογή είχε ξεκινήσει να αναπτύσσεται σαν μια κανονική εφαρμογή μονολιθικής αρχιτεκτονικής. Αρχικά, όταν η Uber είχε λίγους πελάτες, η συντήρηση της εφαρμογής ήταν μια απλή διαδικασία. Ωστόσο όσο μεγάλωνε η εφαρμογή και αυξάνονταν οι απαιτήσεις, άρχισαν να εμφανίζονται πολλά προβλήματα στη συντήρηση της εφαρμογής.

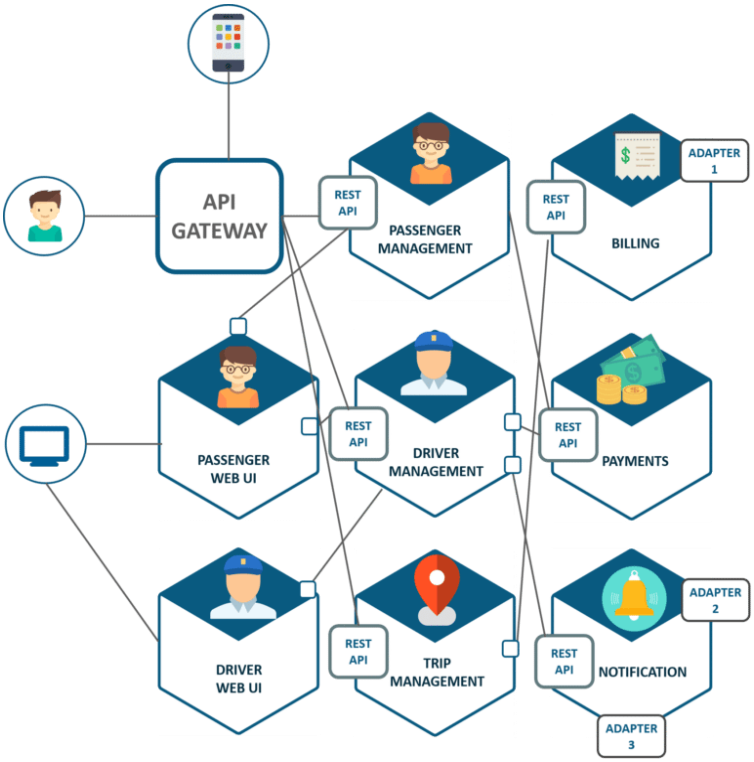
Για αυτόν τον λόγο η εταιρεία αποφάσισε να αλλάξει αρχιτεκτονική και να μεταφερθεί στην αρχιτεκτονική των microservices [18].



Εικόνα 7: Monolithic Architecture of Uber [18]

Όπως φαίνεται στο σχήμα, όλα τα στοιχεία της εφαρμογής ήταν ενοποιημένα. Υπήρχε μόνο μια βάση δεδομένων και ένα REST API. Ωστόσο μόλις η εφαρμογή άρχισε να χρησιμοποιείται από πολλούς χρήστες, εμφανίστηκαν κάποια προβλήματα. Σε περίπτωση που χρειαζόταν μια ενημέρωση, η εφαρμογή έπρεπε να χτιστεί και να γίνει deployed ολόκληρη και μάλιστα πολλές φορές, μια διδικασία δηλαδή που είναι πολύ χρονοβόρα. Η διόρθωση σφαλμάτων άρχισε να γίνεται περίπλοκη καθώς αφορούσε όλο το πρόγραμμα πράγμα που πρόσθετε ακόμα περισσότερες καθυστερήσεις στην ενημέρωση του λογισμικού. Το scaling της εφαρμογής άρχισε να δυσκολεύει και η διαθεσιμότητα της εφαρμογής εμφάνιζε μειώσεις. Για αυτόν τον λόγο η Uber άλλαξε την αρχιτεκτονική της.

Όπως φαίνεται παρακάτω, διαχωρίστηκαν πολλές οντότητες και έγιναν ανεξάρτητες μεταξύ τους. Τα services απέκτησαν δικά τους REST API και διαθέτουν διαφορετικές βάσεις δεδομένων. Με αυτόν τον τρόπο, οι αλλαγές που χρειάζεται να γίνουν σε κάποιο service δεν επηρεάζουν καθολου τα υπόλοιπα.

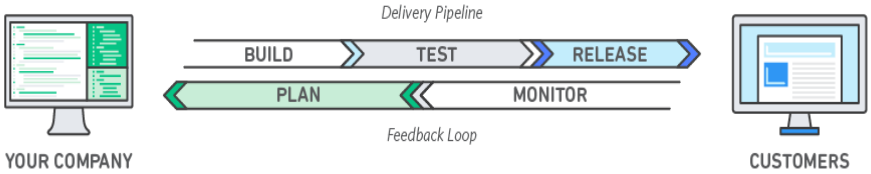


Εικόνα 8: Microservices Architecture of Uber [18]

2.7 DevOps

Παραπάνω αναφέρθηκε ο όρος DevOps και ότι με την αρχιτεκτονική microservices και την τεχνολογία containers ενεργοποιούνται οι DevOps μεθοδολογίες. Τι ακριβώς σημαίνει όμως αυτό;

Σύμφωνα με την Amazon, τον μεγαλύτερο προμηθευτή υπηρεσιών υπολογιστικού νέφους μέσω των υπηρεσιών AWS (Amazon Web Services), το DevOps [19] είναι ένας συνδυασμός φιλοσοφίας, πρακτικών και εργαλείων, που αυξάνει την ικανότητα ενός οργανισμού να μεταφέρει εφαρμογές και services με μεγαλύτερη ταχύτητα, να εξελίσσει και να βελτιώνει τα προϊόντα του με γρηγορότερο ρυθμό από οργανισμούς που χρησιμοποιούν παραδοσιακές μεθόδους ανάπτυξης λογισμικού και διεργασίες διαχείρισης υποδομών. Αυτή η ταχύτητα επιτρέπει τους οργανισμούς και τις εταιρείες να εξυπηρετούν καλύτερα τους πελάτες τους και να βελτιώνουν την ανταγωνιστικότητα τους στην αγορά.



Εικόνα 9: DevOps development circle [19]

2.7.1 Τα πλεονεκτήματα του DevOps

- **Ταχύτητα**: Η ανάπτυξη λογισμικού γίνεται με μεγαλύτερη ταχύτητα χρησιμοποιώντας microservices και continuous delivery (συνεχή παράδοση).
- **Γρηγορότερη παράδοση**: Αυξάνεται η ταχύτητα με την οποία βγαίνουν ενημερώσεις και διορθώνονται σφάλματα (bugs). Το CI/CD (continuous integration & continuous delivery) είναι πρακτική που αυτοματοποιεί την διαδικασία ενημέρωσης λογισμικού.
- **Αξιοπιστία**: Με το CI/CD πριν βγει οτιδήποτε στην παραγωγή (production) περνάει από όλα τα απαραίτητα επίπεδα testing. Επίσης το monitoring και το logging βοηθούν στο να παρακολουθείται η απόδοση σε πραγματικό χρόνο.
- **Scalability**: Οι δυνατότητες αυτοματοποίησης και η αξιοπιστία συμβάλλουν στην διευκόλυνση διαχείρισης μεγάλων και πολύπλοκων συστημάτων. Το Infrastructure as a Code (IaaS) βοηθάει στη διαχείριση των περιβαλλόντων εργασίας με ένα καλύτερο τρόπο.
- **Αυξημένη συνεργασία μεταξύ ομάδων**: Οι ομάδες που εργάζονται πάνω σε ένα πρότζεκτ πρέπει να συνεργάζονται με πιο άμεσο τρόπο, γεγονός που μειώνει τις καθυστερήσεις που προκύπτουν κάποιες φορές μεταξύ των ομάδων του development (ανάπτυξης) και των operations (διαχείρισης).
- **Ασφάλεια**: Γίνεται πιο εύκολη η παρακολούθηση λογισμικού.

Οι πιο γνωστές πρακτικές που χρησιμοποιούνται στο DevOps είναι οι εξείς:

- **Continuous Integration:** Η συνεχής ενσωμάτωση είναι μια πρακτική κατά την οποία ο κώδικας μιας εφαρμογής γίνεται συνεχώς merge σε ένα κεντρικό repository και ο οποίος στη συνέχεια περνάει από διαδικασία αυτοματοποιημένων builds και tests.
- **Continuous Delivery:** Η συνεχής παράδοση είναι μια πρακτική κατά την οποία οι αλλαγές κώδικα ετοιμάζονται γρήγορα προς παράδοση στην παραγωγή. Όταν η συνεχής παράδοση γίνεται σωστά, οι προγραμματιστές με κάθε αλλαγή του κώδικα έχουν ταυτόχρονα και ένα καινούριο artifact διαθέσιμο το οποίο έχει περάσει από την διαδικασία του testing και είναι διαθέσιμο να βγει άμεσα στην παραγωγή.
- **Microservices:** Μια αρχιτεκτονική που σπάει ένα πρόγραμμα μονολιθικής αρχιτεκτονικής σε μικρότερα services.
- **Infrastructure as Code:** Εμπεριέχει τα εργαλεία με τα οποία η υποδομή μπορεί να αλλάζει με τη χρήση κώδικα.
- **Monitoring** και **Logging:** Με αυτές τις πρακτικές δίνεται η δυνατότητα συνεχής παρακολούθησης της απόδοσης των εφαρμογών. Με την συσσώρευση πληροφοριών, σφαλμάτων, δεδομένων χρήσης, αλλά και logs και την ανάλυση τους, οι οργανισμοί μπορούν να παρέχουν καλύτερες και γρηγορότερες ενημερώσεις για τις εφαρμογές τους.

2.8 Container Orchestration

Το container orchestration [20] αυτοματοποιεί το deployment, τη διαχείριση, το scaling και το networking των containers. Όσο αυξάνεται ο αριθμός των containers που χρειάζεται να διαχειριστούν, τόσο φαίνονται τα πλεονεκτήματα του.

Το container orchestration μπορεί να χρησιμοποιηθεί σε οποιοδήποτε περιβάλλον χρησιμοποιεί containers. Μπορεί να βοηθήσει στο deployment των εφαρμογών σε διαφορετικά περιβάλλοντα χωρίς την ανάγκη επανασχεδιασμού τους. Τα microservices στα containers κάνουν πιο εύκολη την ενορχήστρωση των services, του αποθηκευτικού χώρου, του networking και της ασφάλειας.

2.8.1 Που χρησιμοποιείται το container orchestration;

Το container orchestration χρησιμοποιείται για να αυτοματοποιήσει και να διαχειριστεί τις παρακάτω διαδικασίες:

- Provisioning και deployment
- Παραμετροποίηση συστήματος και δημιουργία χρονοδιαγράμματος
- Κατανομή πόρων
- Διαθεσιμότητα των containers
- Scaling ή αφαίρεση containers με βάση την εξισορρόπηση φορτίων στην υποδομή
- Load balancing και δρομολόγηση των αιτημάτων (requests)
- Monitoring της υγείας των containers
- Ρύθμιση παραμέτρων εφαρμογών που θα εκτελούνται στα containers
- Εξασφάλιση ασφαλούς επικοινωνίας μεταξύ των containers

2.9 Kubernetes

Το Kubernetes [21] είναι μια φορητή, επεκτάσιμη πλατφόρμα ανοιχτού κώδικα για το orchestration των containers και των service τους, που χρησιμοποιεί δηλωτική (declarative) παραμετροποίηση και αυτοματοποίηση. Έχει ένα μεγάλο και γρήγορα αναπτυσσόμενο οικοσύστημα. Το όνομα Kubernetes προέρχεται από την ελληνική λέξη «κυβερνήτης». Η Google μετέτρεψε το πρότζεκτ του kubernetes σε πρότζεκτ ανοιχτού κώδικα το 2014 και συνδυάζει 15 χρόνια εμπειρίας της Google σε περιβάλλοντα παραγωγής με υψηλό φόρτο μαζί με τις καλύτερες πρακτικές και εφαρμογές που προτείνονται από την ανοιχτή κοινότητα.

2.9.1 Ορολογία

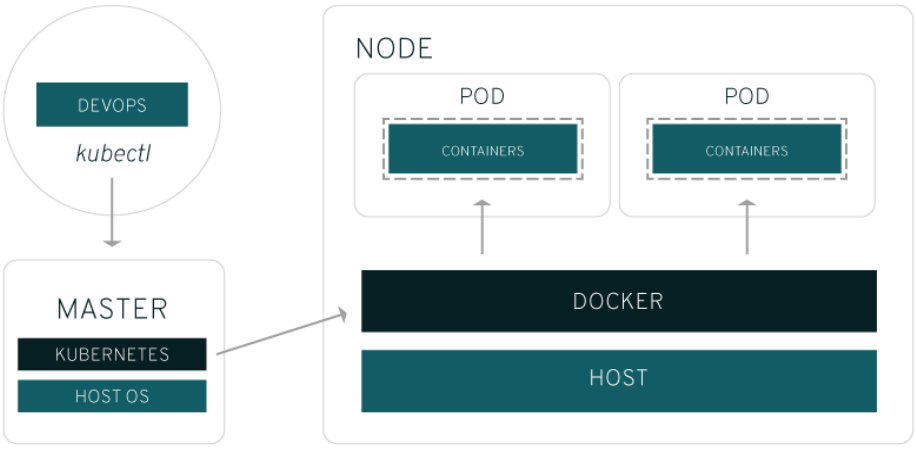
Παρακάτω θα προσδιοριστούν κάποιοι όροι που χρησιμοποιούνται συχνά σε περιβάλλοντα που χρησιμοποιούν containers και kubernetes:

- **Node:** Ονομαζεται το κάθε φυσικό μηχάνημα που εκτελεί εντολές. Τα nodes ελέγχονται από το master node.
- **Master:** Το μηχάνημα που ελέγχει τα kubernetes nodes. Όλες οι εντολές προέρχονται από αυτό.
- **Cluster:** Μια ομάδα από nodes που περιέχει τουλάχιστον ένα master node και μερικά worker nodes.
- **Pod:** Ένα ή περισσότερα containers τα οποία είναι deployed σε ένα συγκεκριμένο node. Όλα τα containers σε ένα pod μοιράζονται διευθύνσεις IP, IPC, το hostname και άλλους πόρους. Τα pods λειτουργούν αφαιρετικά ως προς τις έννοιες του networking και του αποθηκευτικού χώρου σε ένα container, έτσι ώστε να μπορούν να μετακινούνται εύκολα στο cluster.
- **Replication controller:** Ο ελεγκτής αυτός ελέγχει πόσα αντίγραφα ενός συγκεκριμένου pod τρέχουν παράλληλα σε ένα cluster.
- **Service:** Αποτελεί τον συνδετικό κρίκο μεταξύ των pods και του έξω κόσμου. Τα services χρησιμοποιούνται έτσι ώστε η επικοινωνία να γίνεται πάντα με το σωστό pod, χωρίς να παίζει ρόλο αν αλλάζει η τοποθεσία του μέσα στο cluster.
- **Kubelet:** Αυτό το service τρέχει σε όλα τα nodes, διαβάζει τα container manifests και διασφαλίζει ότι όλα τα containers έχουν ξεκινήσει και εκτελούνται.
- **kubectl:** Ένα εργαλείο διαχείρισης σε μορφή γραμμής εντολών για το kubernetes.
- **Controller-Manager:** Ένας ελεγκτής υπεύθυνος να ελέγχει και να τηρεί ότι η τρέχουσα κατάσταση είναι ίδια με την επιθυμητή κατάσταση.

2.9.2 Χρησιμότητα

Με το Kubernetes γίνεται ευκολότερη η συντήρηση τόσο των legacy συστημάτων όσο και των συστημάτων υπολογιστικού νέφους, όπως επίσης και συστημάτων που χρησιμοποιούν microservices.

Προσφέρει πολλές λειτουργίες έτοιμες για χρήση (out of the box) που αλλιώς θα χρειάζονταν τόσο υπολογιστικούς πόρους όσο και χρόνο και εργατοώρες για να εκτελεστούν σωστά. Αυτό δίνει την δυνατότητα στους προγραμματιστές να επικεντρωθούν περισσότερο στην λειτουργικότητα παρά σε εργασίες που αφορούν την υποδομή και τη διαχείριση του συστήματος και των nodes. Πλέον ο έλεγχος της υποδομής γίνεται σε υψηλότερο επίπεδο.

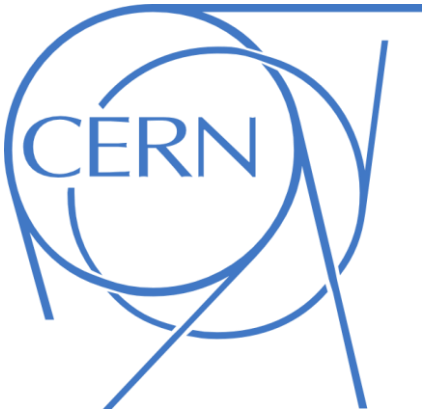


Εικόνα 10: Αρχιτεκτονική Kubernetes [22]

Ο kubernetes master node παίρνει εντολές από τον προγραμματιστή (ή την DevOps ομάδα) και τις μεταφέρει στους αντίστοιχους worker nodes, οι οποίοι μπορεί να είναι είτε φυσικοί υπολογιστές, είτε εικονικές μηχανές. Το ποιος node θα χρησιμοποιηθεί εξαρτάται από τα services που υπάρχουν στο σύστημα. Το kubernetes απλώς ελέγχει πάντα αν όλες οι παραμετροποιήσεις βρίσκονται στην επιθυμητή κατάσταση (desired state). Με λίγα λόγια πλέον γίνονται αυτόματα τα παρακάτω:

- **Service Discovery:** Ένα container εκτίθεται είτε μέσω ενός DNS ονόματος είτε μέσω της δικής του IP διεύθυνσης.
- **Load Balancing:** Όταν υπάρχει υψηλό φορτίο το kubernetes το διαμοιράζει αυτόματα μεταξύ των nodes ώστε το deployment να είναι σταθερό.
- **Αυτοματοποιημένα rollout και rollback:** Το kubernetes προσπαθεί πάντα να εφαρμόσει την επιθυμητή κατάσταση και αν αυτή δεν είναι δυνατή, θα επανέρχεται στην προηγούμενη επιθυμητή μέχρι να μπορέσει να πάει στην επόμενη. Έτσι διασφαλίζεται η υψηλή διαθεσιμότητα (high availability) και ο μηδενικός χρόνος διακοπής (zero downtime).
- **Δυναμική προσαρμογή πόρων:** Ορίζεται ένα εύρος πόρων που μπορεί να χρησιμοποιεί κάθε container και όταν δεν χρειάζεται την μέγιστη τιμή, οι πόροι μπορούν να χρησιμοποιούνται από άλλα containers.
- **Self-healing:** Τα containers στα οποία δημιουργείται σφάλμα επανεκκινούνται αυτόματα ενώ όσα δεν αποκρίνονται σωστά στους χρήστες καταστρέφονται και γίνονται διαθέσιμα μόνο όταν λειτουργούν σωστά.
- **Διαχείριση μυστικών:** Οι ευαίσθητες πληροφορίες που να περιέχονται σε ένα σύστημα μπορούν να χρησιμοποιηθούν χωρίς να εκτεθούν και η μετάδοση αυτών να γίνεται με τη χρήση κρυπτογράφησης.

Ενότητα 3: Χρήση Containers στο CERN



Εικόνα 11: CERN [3]

Το CERN [23] διατηρώντας τη σύντμηση (ακρωνύμιο) της αρχικής ονομασίας του Conseil Européen pour la Recherche Nucléaire, είναι το μεγαλύτερο σε έκταση (πειραματικό) κέντρο πυρηνικών ερευνών, και ειδικότερα επί της σωματιδιακής φυσικής, στον κόσμο. Βρίσκεται δυτικά της Γενεύης, στα σύνορα Ελβετίας και Γαλλίας. Ιδρύθηκε το 1954 από δώδεκα ευρωπαϊκές χώρες και σήμερα αριθμεί 22 κράτη-μέλη, μεταξύ των οποίων και η Ελλάδα, η οποία είναι και ιδρυτικό μέλος.

Το υπολογιστικό νέφος του CERN (CERN private cloud) αποτελείται από χιλιάδες υπολογιστές (nodes) και χρήστες και καλύπτει μια πληθώρα διαφορετικών περιπτώσεων χρήσης (use cases). Αυτό σημαίνει ότι είναι αναγκαία η αναζήτηση διαφορετικών λύσεων για τις διαφορετικές ανάγκες που προκύπτουν στον οργανισμό, δεν μπορεί να χρησιμοποιηθεί μόνο μια τεχνολογία καθολικά. Επίσης είναι σημαντικό να αποφευχθεί το κλείδωμα σε προμηθευτή (vendor lock-in) ώστε να μπορεί να έχει την ευελιξία να μετακινείται μεταξύ προμηθευτών και τεχνολογιών κατά βούληση και χωρίς επιπρόσθετο κόστος. Αυτή η φιλοσοφία φάνηκε κυρίως και στο MAlt project όπου το CERN έκανε ένα μεγάλο άνοιγμα σε τεχνολογίες ανοιχτού λογισμικού.

Παρακάτω θα δοθούν δύο διαφορετικές υποδομές που μπορούν να αξιοποιήσουν containers ώστε να φιλοξενήσουν εφαρμογές, οι οποίες χρησιμοποιούνται από διαφορετικές ομάδες και για διαφορετικούς σκοπούς. Επίσης, θα αναφερθεί η εφαρμογή η οποία θα παρουσιαστεί ως μελέτη περίπτωσης για το containerization της.

3.1 OpenStack Magnum

Μέχρι και το 2016, όταν αποφασίστηκε να γίνει η στροφή προς τις τεχνολογίες container, το υπολογιστικό νέφος OpenStack στο CERN εξυπηρετούσε 2700 χρήστες και 3300 πρότζεκτ,

με 23000 εικονικές μηχανές να τρέχουν πάνω σε 7000 hypervisors. Χρησιμοποιώντας τις τεχνολογίες container θα μπορούσαν να επωφεληθούν από όλα τα πλεονεκτήματα τα οποία έχουν αναφερθεί και παράλληλα θα εξασφαλιζόταν ότι το CERN ακολουθεί τις τελευταίες πρακτικές της βιομηχανίας.

Το OpenStack Magnum [24] είναι ένα API service που αναπτύχθηκε από την ομάδα containers Openstack ώστε να γίνεται η διαχείριση ενός cluster. Οι τρεις βασικές έννοιες στο OpenStack Magnum είναι οι εξείς:

- Cluster Template, ένα επαναχρησιμοποιήσιμο πρότυπο για τα deployments του cluster.
- Cluster, ένα συγκεκριμένο deployment ενός container cluster.
- Node, το οποίο μπορεί να είναι μια εικονική μηχανική ή ένας baremetal server.

Το Magnum προσέφερε τα παρακάτω πλεονεκτήματα:

- Καλή ενσωμάτωση με το OpenStack στο CERN καθώς μπορούσαν να επαναχρησιμοποιηθούν πολλά services και λειτουργικότητες.
- Υποστήριξη πολλαπλών τεχνολογιών ενορχήστρωσης containers, συμπεριλαμβανομένων των Kubernetes, Docker Swarm και Mesos.
- Ταχύτητα και ευκολία στη χρήση του.

Πέρα από την σωστή ενσωμάτωση στην υπάρχουσα υποδομή, σημαντικό ήταν να ελαχιστοποιηθεί η αλληλεπίδραση μεταξύ των χρηστών και του Magnum σε ότι αφορά το deployment των cluster. Παρακάτω φαίνονται οι εντολές που χρειάζεται να κάνουν οι χρήστες ώστε να έχουν ένα έτοιμο cluster μέσα σε λίγα λεπτά, και να το μεγαλώνουν ή μικρύνουν δυναμικά:

```
$ magnum cluster-create --name mycluster --cluster-template swarm --node-count 100
```

```
$ magnum cluster-list
```

```
+------+----------------+----------------+------------------+-----------------------------+
| uuid |      name      | node_count | master_count |            status             |
+------+----------------+----------------+------------------+-----------------------------+
|  .... |  mycluster  |        100      |           1          | CREATE_COMPLETE |
+------+----------------+----------------+------------------+-----------------------------+
```

```
$ $(magnum cluster-config mycluster --dir magnum/mycluster) $ docker info / …
```

```
$ docker run --volume-driver cvmfs -v atlas.cern.ch:/cvmfs/atlas -it centos /bin/bash
[root@32f4cf39128d /]#
```

Παράδειγμα εντολών στο Magnum [24]

$ magnum cluster-update mycluster replace node_count=200
$ magnum cluster-update mycluster replace node_count=5

Εντολές scaling στο Magnum [24]

3.1.1 Ενσωμάτωση αποθηκευτικού χώρου

Η ενσωμάτωση αποθηκευτικού χώρο ήταν το κορυφαίο αίτημα των αρχικών χρηστών, το οποίο για το CERN μεταφράζεται σε αρχεία συστήματος CVMFS [25] και EOS [26].

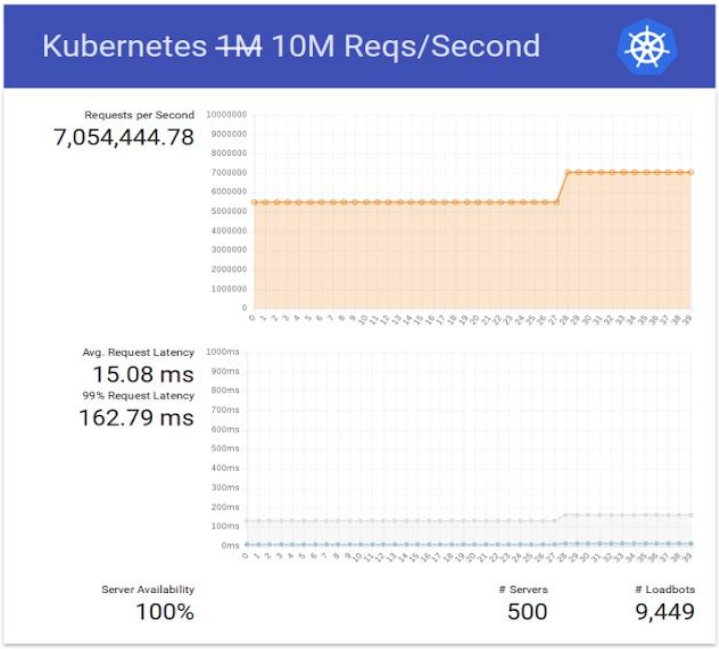
Το CVMFS είναι ένα κατανεμημένο αρχείο συστήματος μόνο για ανάγνωση, το οποίο χρησιμοποιείται για την διανομή λογισμικού ανάμεσα στις χιλιάδες ιστοσελίδες που συνεργάζονται για τα πειράματα του CERN. Η ενσωμάτωση του CVMFS με το Magnum στο CERN γίνεται μέσω ενός docker volume plugin [27].

Το EOS χειρίζεται όλα τα δεδομένα φυσικής που προέρχονται από τα πειράματα του CERN, το οποίο ήδη αποθηκεύει εκατοντάδες petabytes και το οποίο μέγεθος πρόκειται συνεχώς να αυξάνεται όσο αυξάνεται και το luminosity των πειραμάτων. Η ενσωμάτωση γίνεται πάλι μέσω ενός docker plugin [28].

Και στις δύο περιπτώσεις η ενσωμάτωση με το kubernetes βασίζεται σε ένα wrapper μέσω του FlexVolume plugin [29] το οποίο καλεί το Docker API ώστε ο κώδικας να είναι επαναχρησιμοποιούμενος.

3.1.2 Αξιολόγηση απόδοσης

Τα αποτελέσματα [24] φαίνονται στους παρακάτω πίνακες. Ο αριθμός των clusters μέχρι και 128 nodes έχει ένα πολύ καλό αποτέλεσμα με το deployment να παίρνει περίπου σταθερό χρόνο για να ολοκληρωθεί. Σε clusters με μεγαλύτερο αριθμό nodes η αύξηση του χρόνου είναι σχεδόν γραμμική, τα 23 λεπτά για το deployment ενός cluster με 1000 nodes θεωρείται απολύτως αποδεκτό.



Εικόνα 12: Kubernetes scale test [24]

Cluster Size (Nodes)	Concurrency	Deployment Time (min)
2	50	2.5
16	10	4
32	10	4
128	5	5.5
512	1	14
1000	1	23

Εικόνα 13: Χρόνος deployment [24]

Στο kubernetes scale τεστ, επιτεύχθηκαν 7 εκατομμύρια αιτήματα το δευτερόλεπτο χρησιμοποιώντας ένα cluster με 1000 nodes.

3.2 Openshift

Το Openshift [30] είναι μια πλατφόρμα στην οποία μπορούν να εκτελούνται containerized εφαρμογές και η οποία «τρέχει» kubernetes κάτω από την επιφάνεια. Στην ουσία συνδυάζει τις τεχνολογίες container και kubernetes ώστε οι χρήστες να έχουν ένα έτοιμο και λειτουργικό περιβάλλον το οποίο περιέχει όλα τα απαραίτητα εργαλεία που μπορεί να χρειαστεί κανείς κατά την ανάπτυξη και το deployment εφαρμογών. Υπάρχουν διαφορετικές εκδόσεις του OpenShift, η βασική εφαρμογή είναι ανοιχτού κώδικα και ονομάζεται OKD ή αλλιώς Origin Kubernetes Distribution, ωστόσο υπάρχουν και πιο εξειδικευμένες επιλογές οι οποίες παρέχονται από την RedHat. Μπορεί να θεωρηθεί οτι το kubernetes αποτελεί τον «πυρήνα» του OpenShift. Το OpenShift αποτελεί την «διανομή» του. Με άλλα λόγια, το kubernetes προσφέρει το σύστημα ενορχήστρωσης και όλα τα πλεονεκτήματα του, ενώ ο OpenShift χτίζει

πάνω σε αυτό και προσφέρει επιπλέον λειτουργικότητα όπως η παροχή δυνατότητας σε χρήστες να δημιουργούν πολλά πρότζεκτ με διαφορετικές ομάδες χρηστών, βελτιωμένη λειτουργικότητα όσον αφορά το load balancing και την διαδικασία του deployment, μια εύκολη στην χρήση της διεπαφή χρηστών (user interface) και πολλά έτοιμα παραδείγματα και πρότυπα εφαρμογών (templates).

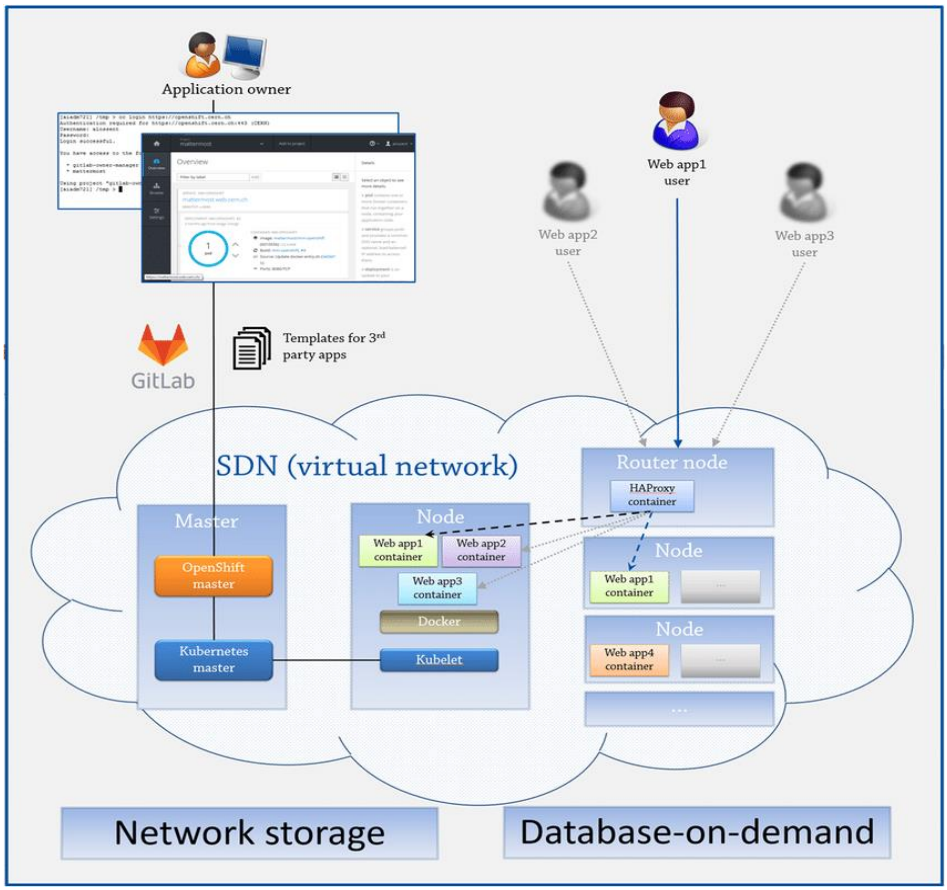
Η ομάδα του CERN Web Frameworks είναι υπεύθυνη για την παροχή υπηρεσιών και εργαλείων που απευθύνονται σε προγραμματιστές όπως επίσης και υπηρεσίες που αφορούν την κεντρική εξυπηρέτηση (central hosting) όλων των ιστοσελίδων και εφαρμογών του CERN. Σκοπός της ομάδας είναι να προσφέρει ελκυστικές υπηρεσίες ώστε περισσότερες ομάδες στο CERN να τις χρησιμοποιούν, διατηρώντας παράλληλα χαμηλό το κόστος των υπηρεσιών αυτών στον οργανισμό. Ως αποτέλεσμα, η ομάδα αποφάσισε να χρησιμοποιήσει το OpenShift Origin [31], ή αλλιώς OKD, ακολουθώντας το παράδειγμα της πλατφόρμας σε μορφή service (Platform as a Service ή Paas), διευκολύνοντας τους χρήστες, προσφέροντας μια εύκολη στην χρήση της πλατφόρμα, με μια εύκολα κατανοητή διεπαφή χρήστη (UI), και κερδίζοντας τα πλεονεκτήματα της χρήσης containers.

Τελικός στόχος είναι να μεταφερθούν και οι 14000 εφαρμογές και ιστοσελίδες που εξυπηρετεί η ομάδα Web Frameworks στο OpenShift, οι οποίες προς το παρόν παρέχονται με παραδοσιακές μεθόδους εικονικών μηχανών.

3.2.1 Αρχιτεκτονική

Η αρχιτεκτονική του OpenShift, όπως χρησιμοποιείται στο CERN, φαίνεται στο παρακάτω σχήμα. Οι διαχειριστές των εφαρμογών μπορούν να διαχειρίζονται τις εφαρμογές τους μέσω των servers διαχείρισης, είτε μέσω της διεπαφής χρήστη, είτε μέσω μιας διεπαφής γραμμής εντολών. Η επιθυμητή κατάσταση των εφαρμογών βρίσκεται στους master servers, ενώ τα containers που εκτελούν τις εφαρμογές εκτελούνται στους worker nodes.

Οι εφαρμογές μπορούν να αποτελούνται από πολλά containers, τα οποία συνδέονται μέσω ενός software-defined εικονικού δικτύου (SDN). Η επικοινωνία μεταξύ των containers είναι ανεξάρτητη από το αν αυτά βρίσκονται στο ίδιο node ή όχι. Επίσης, τα containers δεν είναι απευθείας προσβάσιμα εκτός του OpenShift SDN. Αντίθετα, αυτά είναι προσβάσιμα μέσω ενός HAProxy [32] router (δρομολογητή), ο οποίος παίρνει τα αιτήματα των χρηστών και τα αναδιανέμει στα αντίστοιχα containers. Σε ασφαλείς συνδέσεις HTTPS χρησιμοποιείται το Server Name Indication [33].



Εικόνα 14: Η αρχιτεκτονική του OpenShift στο CERN [31]

Περιπτώσεις χρήσεων για το PaaS service στην περίπτωση του CERN:

Οι κύριες χρήσεις αυτού του service στο CERN είναι τρεις:

- Η πρώτη αφορά τα έτοιμα πρότυπα εφαρμογών. Για παράδειγμα πολλές ομάδες ενδέχεται να συντηρούν οι ίδιες πολλές εφαρμογές γενικής χρήσης, σε δικούς τους servers, όπως για παράδειγμα το Jenkins [34] ή το Grafana [35]. Το OpenShift προσφέρει έτοιμα πρότυπα (templates) που μπορούν να χρησιμοποιήσουν αυτές οι ομάδες, ώστε να γλιτώσουν από τον κόπο και το κόστος να συντηρούν από μόνοι τους αυτές τις εφαρμογές.

- Η δεύτερη αφορά το deployment εφαρμογών τρίτων (third party applications). Αυτό αφορά τόσο εφαρμογές ανοιχτού κώδικα, όσο και εφαρμογές ιδιωτικών εταιρειών. Το OpenShift προσφέρει πολλούς τρόπους με τους οποίους μπορούν να γίνουν deployed αυτές οι εφαρμογές, ειδικά εφόσον έχουν γίνει containerized. Δίνει την δυνατότητα, εφόσον οι εφαρμογές υπάρχουν σε κάποιο docker repository, να εισάγονται στο OpenShift και στη συνέχεια εισάγοντας αρχεία παραμετροποίησης και μυστικά (secrets), οι εφαρμογές να μπορούν να αλλάζουν ρυθμίσεις κατά τη διάρκεια του deployment, ενώ δίνεται η δυνατότητα να επαναχρησιμοποιείται η ίδια εικόνα docker για διαφορετικές περιπτώσεις. Για παράδειγμα αν υπαρχει μια εφαρμογή που χρησιμοποιεί κάποια βάση δεδομένων, η βάση δεδομένων δεν αποτελεί κομμάτι της εφαρμογής. Αντίθετα, μπορεί να χρησιμοποιηθεί μια γενική docker εικόνα ανοιχτού

κώδικα, και η βάση δεδομένων να προσκολληθεί στην εφαρμογή δυναμικά μέσω των αρχείων παραμετροποίησης στα οποία θα αναφέρεται η βάση δεδομένων και τα μυστικά της κρυπτογραφημένα. Οι βάσεις δεδομένων στο CERN προσφέρονται από την ομάδα Database on Demand [36], το οποίο σημαίνει ότι ο τελικός διαχειριστής της εφαρμογής δεν χρειάζεται να διαχειρίζεται ούτε το deployment της εφαρμογή, ούτε και την βάση δεδομένων αυτής, αλλά απλά να χρησιμοποιήσει τα δυο αυτά service και να επικεντρωθεί στην λειτουργικότητα της ίδιας της εφαρμογής. Ωστόσο, δεν μπορούν να χρησιμοποιηθούν όλες οι εφαρμογές τρίτων πάντα, καθώς το OpenShift για λόγους ασφάλειας εμποδίζει κάποιες λειτουργικότητες και επιλογές, όπως για παράδειγμα την χρήση κάποιων συγκεκριμένων Linux UID.

- Η τρίτη αφορά το deployment εφαρμογών από τον πηγαίο κώδικα (source code). Αυτή η επιλογή απευθύνεται στις εφαρμογές οι οποίες είναι εξ ολοκλήρου ανεπτυγμένες στο CERN. Η λειτουργικότητας της πηγής-σε-εικόνα (source-to-image) του OpenShift υποστηρίζει πολλές γλώσσες προγραμματισμού συμπεριλαμβανομένων των Ruby, Python, Django, Node.js, Perl, Java και PHP. Αυτή η δυνατότητα διευκολύνει αρκετά το deployment των «in-house» εφαρμογών σε σχέση με τις παλιότερες μεθόδους.

Συνολικά, η υιοθέτηση του OpenShift σαν βασική πλατφόρμα hosting των ιστοσελίδων και εφαρμογών για τα οποία είναι υπεύθυνη η ομάδα Web Frameworks, έχει βοηθήσει πολύ την ομάδα να αντιμετωπίσει πολλές δυσκολίες, να χρησιμοποιεί λιγότερους servers και να εκσυγχρονίσει αρκετά την υποδομή του CERN ακολουθώντας τις πρακτικές των containers και προσφέροντας νέα εργαλεία στους υπόλοιπους χρήστες.

3.3 Matomo

Το Matomo [37], πρώην Piwik, είναι ένα service που παρέχει η ομάδα του Web Frameworks του CERN ώστε οι διαχειριστές άλλων εφαρμογών ή ιστοσελίδων να μπορούν να παίρνουν στατιστικά στοιχεία για την επισκεψιμότητα των ιστοσελίδων τους. Είναι ένα αρκετά σημαντικό service καθώς μέσω αυτού οι διαχειριστές μπορούν να δουν πολλά πράγματα για τους χρήστες τους, όπως για παράδειγμα από ποιες διευθύνσεις επισκέπτονται τις ιστοσελίδες τους, ποια μέρη της ιστοσελίδας επισκέπτονται πιο συχνά, από ποιες χώρες τους επισκέπτονται και πάρα πολλά ακόμη.

Το Piwik/Matomo χρησιμοποιήθηκε ως αντικαταστάτης του Google Analytics, καθώς το CERN θέλει να κρατάει τέτοια ευαίσθητα δεδομένα σε δικούς του servers και να έχει πλήρες έλεγχο της εφαρμογής και των δεδομένων της καθώς αυτά θα αποθηκεύονται σε βάσεις δεδομένων του οργανισμού. Προσφέρει πλήρη ιδιωτικότητα αλλά και εργαλεία ώστε οι διαχειριστές των εφαρμογών να μπορούν να είναι συμμορφούμενοι με τους νέους κανόνες GDPR.

Αυτό το service είναι ένα από τα πολλά που είναι deployed με παραδοσιακές μεθόδους σε servers, και η ομάδα θέλησε να είναι από τις πρώτες εφαρμογές που θα μετακινηθούν σε containers. Ο λόγος είναι ότι η εφαρμογή θα παίζει πολύ σημαντικό ρόλο για το μέλλον των

Web Frameworks, οι απαιτήσεις θα ανέβουν πολύ καθώς από τις περίπου 2000 ιστοσελίδες και εφαρμογές που παρακολουθεί τώρα, θα καταλήξει να παρακολουθεί όλες τις centrally hosted ιστοσελίδες και εφαρμογές, οι οποίες είναι 14000, πράγμα που σημαίνει με βάση τα υπάρχοντα στοιχεία ότι τα αιτήματα ανά δευτερόλεπτο (request per second) μπορούν να φτάσουν μέχρι και τα 2000. Επίσης η εφαρμογή δεν γίνεται εύκολα updated σε έναν server, είναι πολύ αργή η μετανάστευση (migration) της σε διαφορετικούς servers και βάσεις δεδομένων, και γενικότερα το CERN επί σειρά ετών λειτουργούσε με την εφαρμογή να βρίσκεται σε μια παλιά έκδοση και έτσι να χάνουν πολλές λειτουργικότητες και όλες τις βελτιώσεις που φέρνουν οι ενημερώσεις της εφαρμογής αλλά και τις διορθώσεις των κενών ασφαλείας. Επίσης, η εφαρμογή θα σταματήσει να είναι απλώς ένα service το οποίο χρησιμοποιούν άλλες ομάδες του CERN, αλλά θα μετατραπεί για εργαλείο για την συσσώρευση στατιστικών δεδομένων που θα χρησιμοποιεί η ίδια η ομάδα για τις ιστοσελίδες και εφαρμογές που θα μεταφερθούν στο OpenShift. Αυτό σημαίνει ότι θα πάψει να είναι απλά ένα εργαλείο διαθέσιμο σε τρίτους και θα γίνει ένα στοιχείο της υποδομής (infrastructure) της ομάδας Web Frameworks.

Γενικές πληροφορίες για την υποδομή του CERN:

Είναι σημαντικό να αναφερθεί, από άποψη ασφάλειας, πως επιτυγχάνει το CERN την εξουσιοδότηση (authorization) και την επαλήθευση (authentication) των χρηστών του. Όπως σε κάθε οργανισμό, είναι σημαντικό να μην μπορεί κάποιος τρίτος εκτός οργανισμού να αποκτήσει πρόσβαση σε υπολογιστές αυτού ή σε πληροφορίες τις οποίες ο οργανισμός δε θέλει να εκτεθούν προς τα έξω. Για αυτόν τον λόγο το CERN ακολουθεί συγκεκριμένες πρακτικές τις οποίες χρησιμοποιούν όλοι οι χρήστες του.

Αυτό το κομμάτι, που λέγεται authentication, επιτυγχάνεται μέσω ενός συστήματος single sign-on (ενιαίας εισόδου) [38]. Αυτό το σύστημα επιτρέπει σε ανθρώπους να ταυτοποιούνται χρησιμοποιώντας μόνο μια συγκεκριμένη ταυτότητα και να εισέρχονται σε διαφορετικά συστήματα με αυτή. Τέτοια παραδείγματα είναι το πανεπιστημιακό email, ή διάφορες ιστοσελίδες που σε αφήνουν να μπεις σε αυτές μέσω του Facebook ή του λογαριασμού Google του καθενός. Στην συγκεκριμένη περίπτωση, οι χρήστες του CERN μπορούν να συνδέονται στις υπηρεσίες και τις εφαρμογές που δεν είναι εκτεθειμένες προς το γενικότερο κοινό, μέσω του προσωπικού τους λογαριασμό στον οργανισμό, ο οποίος δημιουργείται χειροκίνητα για κάθε εργαζόμενο ξεχωριστά.

Ωστόσο, πέρα από το να εξασφαλίσει το CERN ότι δεν επιτρέπεται η πρόσβαση μη επιθυμητών χρηστών σε σημεία που δεν είναι επιθυμητή, σημαντικό είναι να μην μπορεί ένας οποιοσδήποτε χρήστης μιας ομάδας στο CERN να έχει πρόσβαση σε αρχεία, πληροφορίες και προγράμματα άλλων ομάδων τα οποία αυτές δε θέλουν να εκθέσουν. Για αυτόν τον λόγο χρησιμοποιείται το egroups [39], το οποίο είναι μια εφαρμογή στην οποία οι χρήστες του οργανισμού μπαίνουν σε διαφορετικές ομάδες, ανάλογα με τα πρότζεκτ στα οποία εργάζονται, και το κάθε πρότζεκτ ρυθμίζει τις άδειες προσβασιμότητας του με βάση τις ομάδες αυτές.

3.3.1 Deployment του Matomo στο OpenShift

Η αρχιτεκτονική του deployment της εφαρμογής στο CERN είναι ως εξής:

Τρία containers θα βρίσκονται στο ίδιο pod, το πρώτο θα εκτελεί το Matomo μέσω ενός Apache Web Server [40], το δεύτερο θα εκτελεί το Shibboleth daemon, μια Single Sign-On Authentication διεργασία με βάση την οποία οι χρήστες θα εισέρχονται στην εφαρμογή, και το τρίτο θα εκτελεί το Logstash, μια εφαρμογή η οποία θα στέλνει τα logs του Matomo στο Central Logging του CERN.

Το Shibboleth [41] είναι ένα σύστημα ενιαίας εισόδου (single sign-on) για δίκτυα υπολογιστών και το διαδίκτυο.

Το Matomo θα είναι προσβάσιμο μέσω του υπερσυνδέσμου: https://NAMESPACE.web.cern.ch όπου NAMESPACE είναι μια μεταβλητή η οποία θα ορίζεται από το namespace της εφαρμογής. Η διεύθυνση αυτή θα είναι προσβάσιμη από οπουδήποτε στο διαδίκτυο.

Όσον αφορά τα αιτήματα API αυτά θα είναι διαθέσιμα με δυο τρόπους:

Ο πρώτος είναι μέσα από το OpenShift cluster με την εσωτερική διεύθυνση που θα περιέχει το αντίστοιχο service.

Ο δεύτερος είναι μέσω ενός υπερσυνδέσμου που θα είναι προσβάσιμος από το διαδίκτυο και δεν θα προστατεύεται από το shibboleth.

Το Matomo συνδέεται με μια βάση δεδομένων MySQL η οποία παρέχεται από την ομάδα DB on Demand. Σε αυτή τη βάση δεδομένων αποθηκεύονται όλοι οι χρήστες, τα στατιστικά και γενικά όλα τα στοιχεία που συλλέγει και χρησιμοποιεί η εφαρμογή.

Εκτός από το deployment της εφαρμογής, έχουν αναπτυχθεί και επεκταθεί κάποια plugins τα οποία δίνουν επιπλέον λειτουργικότητα στο Matomo. Το plugin LoginCERN χρησιμοποιείται για να εισάγει τους χρήστες στη βάση δεδομένων, και ύστερα να τους ταυτοποιεί, ενώ το plugin GroupPermissions χρησιμοποιείται ώστε να μπορούν να ταυτοποιούνται και ομάδες χρηστών στο CERN με βάση το σύστημα των egroups, πέρα από μεμονωμένους χρήστες δηλαδή, και να δίνονται τα σωστά permissions χρηστών και ομάδων στην κάθε ιστοσελίδα που παρακολουθεί το Matomo.

Σημαντικό είναι να αναφερθεί ότι για το deployment θα χρησιμοποιηθούν τα Helm Charts. Το Helm [42] πολλές φορές συγκρίνεται με έναν διαχειριστή πακέτων (package manager) και αναφέρονται σε αυτό ως «διαχειριστής πακέτων του kubernetes». Πρόκειται για κάποια αρχεία τα οποία περιγράφουν τα αντικείμενα (resources) του kubernetes. Αυτά τα αρχεία βρίσκονται σε μορφή yaml [43] και περιέχουν όλες τις παραμετροποιήσεις που θα πρέπει να περιέχει το deployment μας. Λεπτομέρειες θα δοθούν στην ενότητα 3.

Παρακάτω θα παρουσιαστεί αναλυτικότερα ως μελέτη περίπτωσης η μετανάστευση αυτού του service σε containers.

Ενοτητα 4: Μελέτη Περίπτωσης (Case Study)

Παρακάτω θα αναλυθούν όλα τα βήματα που πρέπει να γίνουν και όλα τα αρχεία που πρέπει να δημιουργηθούν ώστε να επιτευχθεί το deployment του Matomo σε containers. Τα βήματα μπορούν να χωριστούν σε τρία κομμάτια:

- Τη δημιουργία της εικόνας (image)
- Τη δημιουργία αρχείων παραμετροποίησης
- Τη δημιουργία του Helm Chart για το deployment

4.1 Δημιουργία image

Το πρώτο κομμάτι αφορά τη δημιουργία του image και την αποθήκευση του στο GitLab Image Repository.

4.1.1 Δημιουργία Dockerfile

Αρχικά θα πρέπει να δημιουργηθεί ένα Dockerfile [44] από το οποίο θα χτιστεί η εικόνα Docker της εφαρμογής μας.

Οι βασικότερες εντολές που «χτίζουν» τα επίπεδα της εικόνας Docker είναι οι παρακάτω:

- **FROM:** Αρχικοποιεί το νέο build μας, κάνοντας αναφορά στην εικόνα την οποία θα χρησιμοποιήσουμε ως σημείο έναρξης.
- **RUN:** η εντολή που θα τρέξει, είναι η ίδια εντολή που τρέχει σαν να ήμασταν μέσα σε ένα linux shell.
- **EXPOSE:** δίνει την θύρα στην οποία θα ακούει το container.
- **ENV:** δημιουργούνται κάποιες μεταβλητές που χρησιμοποιούνται στη συνέχεια του Dockerfile.
- **COPY:** αντιγράφουμε αρχεία και φακέλους από μια πηγή μέσα σε ένα συγκεκριμένο path στο container μας.
- **ENTRYPOINT:** περιέχει την εντολή ή το αρχείο το οποίο θα εκτελεστεί στην έναρξη του container.
- **USER:** θέτουμε τον χρήστη ή το UID που θα εκτελεί τα αρχεία και τις εντολές του container.

Το πρώτο πράγμα που πρέπει να οριστεί στο Dockerfile είναι η βασική εικόνα πάνω στην οποία θα χτιστεί η δική μας εικόνα. Στην περίπτωση του CERN χρησιμοποιείται μια εικόνα

που παρέχει το αντίστοιχο τμήμα του CERN το οποίο είναι υπεύθυνο για τις παραμετροποιήσεις του λειτουργικού συστήματος Linux στον οργανισμό. Αυτή η εικόνα περιέχει το CERN CentOS 7 το οποίο είναι το λειτουργικό σύστημα CentOS της RedHat με τις αντίστοιχες παραμετροποιήσεις που χρειάζεται το CERN.

```
FROM cern/cc7-base
```

Στη συνέχεια θα οριστούν κάποιες μεταβλητές περιβάλλοντος που θα μας διευκολύνουν στην δημιουργία του Dockerfile.

```
ENV MATOMO_VERSION=3.12.0 \
  PHP=php71 \
  PATH=$PATH:/opt/rh/rh-php71/root/usr/bin
```

Η πρώτη μεταβλητή αφορά την έκδοση του λογισμικού την οποία θα τρέχουν τα container μας. Η δεύτερη αφορά την έκδοση της προγραμματιστικής γλώσσας PHP. Με αυτόν τον τρόπο, όταν θα θέλουμε να αλλάξουμε είτε την έκδοση του προγράμματος, είτε της γλώσσας PHP, μπορούμε να το κάνουμε απλώς αλλάζοντας αυτές τις μεταβλητές. Η τρίτη μεταβλητή προσθέτει ένα αρχείο στο Linux PATH ώστε να μπορούμε να τρέχουμε απευθείας αρχεία PHP στην κονσόλα του OpenShift χωρίς να χρειάζεται να το ενεργοποιούμε κάθε φορά. Οι μεταβλητές είναι προσβάσιμες με το σύμβολο του δολαρίου ($).

Στη συνέχεια πρέπει να γίνει η εγκατάσταση όλων των προγραμμάτων και των πακέτων που χρειάζεται το λειτουργικό μας σύστημα ώστε να μπορέσει να τρέξει την εφαρμογή.

```
RUN yum -y update && \
  yum install -y gettext && \
  yum --setopt=tsflags=nodocs -y install deltarpm \
                                rh-$PHP-php \
                                rh-$PHP-php-curl \
                                rh-$PHP-php-gd \
                                rh-$PHP-php-cli \
                                rh-$PHP-php-mysqlnd \
                                rh-$PHP-php-mbstring \
                                rh-$PHP-php-xml \
                                rh-$PHP-php-ldap \
                                httpd24-httpd \
                                cronolog \
                                shibboleth \
                                log4shib \
                                xmltooling-schemas \
```

```
                                opensaml-schemas \
                                libcurl-openssl && \
  rm -rf /var/cache/yum/* && \
  yum clean all && \
  curl -L -o /opt/matomo.tar.gz https://builds.matomo.org/matomo-$MATOMO_VERSION.tar.gz
&& \
  tar -C /opt -xf /opt/matomo.tar.gz && \
  rm /opt/matomo.tar.gz
```

Συνήθως χρειάζεται να εγκατασταθεί μόνο ότι είναι απαραίτητο για την εκάστοτε εφαρμογή. Στην προκειμένη περίπτωση εγκαθιστούμε την προγραμματιστική γλώσσα PHP, στην έκδοση που έχουμε ορίσει, το httpd που αποτελεί τον apache server στις εκδόσεις λειτουργικού της RedHat, το shibboleth που θα αποτελεί το εργαλείο με το οποίο θα γίνεται η πιστοποίηση των χρηστών (authentication), κάποια απαραίτητα πακέτα για να λειτουργούν όλα σωστά και στο τέλος την εφαρμογή μας στην έκδοση που έχουμε ορίσει.

Επίσης μεταφέρονται κάποια αρχεία τα οποία δουλεύουμε ξεχωριστά και τα οποία εισάγονται μέσα στην εικόνα:

```
COPY LoginCERN /opt/matomo/plugins/LoginCERN
COPY GroupPermissions /opt/matomo/plugins/GroupPermissions
COPY conf_files/config.ini.php /opt/matomo/config/config.ini.php
COPY conf_files/*.sh conf_files/shibboleth2.xml conf_files/liveness.py /
COPY conf_files/httpd.conf /opt/rh/httpd24/root/etc/httpd/conf/
COPY conf_files/*_httpd.conf /opt/rh/httpd24/root/etc/httpd/conf.d/
```

Αυτά τα αρχεία περιέχουν τους κώδικες των plugin τα οποία έχουμε επεξεργαστεί ξεχωριστά, όπως επίσης και τις παραμετροποιήσεις του Apache server μας.

Στη συνέχεια θα πρέπει να οριστούν διάφοροι φάκελοι μέσα στα directories του συστήματος και να δοθούν οι αντίστοιχες άδειες στην εφαρμογή και τους διαχειριστές. Ένα κομμάτι αυτών των εντολών εμφανίζεται παρακάτω:

```
RUN chown -R :apache /opt/matomo && \
  chmod a+w /opt/matomo/matomo.js /opt/matomo/piwik.js && \
  touch /var/run/last_matomo_update && mkdir /var/log/httpd/ && \
  chmod -R a+rw /opt/matomo/plugins /opt/matomo/tmp/ /opt/matomo/config/ && \
chmod -R a+rw /opt/rh/httpd24/root/etc/httpd/conf.d/ /opt/rh/httpd24/root/etc/httpd/conf/ && \
  chmod -R a+rwx /var/log/httpd/ && \
```

Όπως φαίνεται, η εφαρμογή Matomo απαιτεί κάποιες συγκεκριμένες ρυθμίσεις στα permissions του συστήματος. Δίνονται δυνατότητες εγγραφής σε κάποιους φακέλους της εφαρμογής αλλά και στον server. Καλό είναι να δίνονται μόνο οι απαραίτητες άδειες για λόγους ασφάλειας.

Μετέπειτα εγκαθιστούμε κάποια πακέτα με βάση τα οποία το Matomo μπορεί να βρίσκει περίπου, πάντα σύμφωνα με τους κανονισμούς του GDPR, την τοποθεσία των χρηστών.

```
RUN cd /opt/matomo/misc/ && \
  mkdir geotemp && cd geotemp && \
  curl -o GeoLite2-City.tar.gz http://geolite.maxmind.com/download/geoip/database/GeoLite2-City.tar.gz && \
  tar -xvzf GeoLite2-City.tar.gz --strip-components=1 && \
  mv *.mmdb ../ && cd ../ && \
  rm -rf geotemp
```

Το Dockerfile αποτελείται επιπλέον από τις δύο παρακάτω εντολές:

```
ENTRYPOINT ["/apache.sh"]

EXPOSE 8080
```

Στην ουσία κάνουμε expose την συγκεκριμένη θύρα (port) η οποία είναι η 8080 και την οποία θα «ακούει» το container μας και δίνουμε εντολή η εικόνα docker μας να εκτελέσει τον κώδικα που βρίσκεται στο αρχείο apache.sh.

Το script το οποίο τρέχει στην έναρξη του container είναι το παρακάτω:

```
#!/bin/bash

echo "--> Running matomo container..."

if [[ -z "$HOSTNAME_FQDN" ]]; then
 export HOSTNAME_FQDN="${NAMESPACE}.web.cern.ch"
fi

```

```
# This env variable subtitution is needed for shibboleth to work.
envsubst < /shibboleth2.xml > /etc/shibboleth/shibboleth2.xml

echo "--> Running Apache..."
exec /opt/rh/httpd24/root/usr/sbin/httpd-scl-wrapper -DFOREGROUND
```

Ορίζουμε ότι θα τρέξει ένα bash αρχείο script, καθώς δεν είναι απαραίτητο ότι θα εκτελεστεί το κέλυφος bash διαφορετικά, ορίζεται μια μεταβλητή περιβάλλοντος (environment variable) που αφορά το hostname του υπερσυνδέσμου που θα χρησιμοποιηθεί αργότερα στο container και τέλος, η σημαντικότερη εντολή του αρχείου, είναι ότι εκτελούμε το daemon του Apache. Η παράμετρος «-DFOREGROUND» είναι απαραίτητη καθώς το container κανονικά εκτελεί μια εντολή και στη συνέχεια τερματίζεται, εφόσον δεν εκτελείταο κάτι στο προσκήνιο. Με αυτόν τον τρόπο ο apache server τρέχει στο προσκήνιο και το docker container εκτελείται επ' αόριστον, όπως θα έκανε ένας παραδοσιακός server.

4.1.2 Logstash

Πέρα από την εφαρμογή του Matomo, θα χρειαστούμε μια εικόνα του Logstash [45]. Το Logstash είναι ένα εργαλείο ανοιχτού κώδικα το οποίο χρησιμοποιείται για να συλλέγονται τα logs από μια πηγή, να μετατρέπεται η δομή τους και να αποστέλλονται στον επιθυμητό προορισμό. Στην συγκεκριμένη περίπτωση τα logs θα αποστέλλονται στο Cern Central Monitoring το οποίο χρησιμοποιεί το Elasticsearch [46]. Το Elasticsearch αποτελεί μια μηχανή αναζήτησης η οποία υποστηρίζει πολλαπλά είδη αρχείων, είτε δομημένα, είτε αδόμητα.

Δημιουργούμε το αντίστοιχο Dockerfile:

```
FROM docker.elastic.co/logstash/logstash:6.6.1

RUN rm -f /usr/share/logstash/config/logstash.yml
RUN rm -f /usr/share/logstash/pipeline/logstash.conf
COPY conf/logstash.conf /etc/logstash/conf.d/
COPY conf/logstash.yml /usr/share/logstash/config/logstash.yml

CMD ["-f", "/etc/logstash/conf.d/logstash.conf"]
```

Χρησιμοποιούμε την έτοιμη εικόνα της Elastic που έχει προεγκατεστημένο το Logstash. Αυτό που αλλάζει είναι οι παράμετροι των αρχείων.

Το αρχείο παραμετροποιήσεων του Logstash έχει τρία πεδία, το input, το filter και το output.

Στο input δηλώνεται το μονοπάτι (path) του φακέλου που θα περιέχει τα αντίστοιχα logs.

```
input {
 file {
   path => "/data/*/*/*.log"
   start_position => "beginning"
   type => "matomo_apache_logs"
 }
}
```

Στο filter αλλάζει η μορφή των αρχείων αυτών ώστε να γίνει αντιστοίχιση με την μορφή την οποία περιμένει το central monitoring.

```
filter {
 if [path] =~ "access" {
   mutate { replace => { "type" => "matomo_apache_access" } }
   grok {
     match => { "message" => "%{HTTPD_COMBINEDLOG}" }
   }
 }
 else if [path] =~ "error" {
   mutate { replace => { "type" => "matomo_apache_error" } }
   grok {
     match => { "message" => "%{HTTPD24_ERRORLOG}"}
   }
 }
 mutate { add_field => { "producer" => "webinfra" } }
 date {
   match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
 }
}
```

Και στο output αναφέρουμε τη διεύθυνση στην οποία θα σταλούν τα logs.

Το gitlab ci είναι αντίστοιχο με αυτό που χρησιμοποιήθηκε και στο Matomo.

4.1.3 GitLab CI/CD

Αφού δημιουργήσουμε το Dockerfile, στη συνέχεια θα θέλουμε να κάνουμε build την εικόνα μας και να την αποθηκεύσουμε σε κάποιο docker repository. Αυτό το πρότζεκτ χρησιμοποιεί όλες τις μεθόδους που αναφέρθηκαν σε αυτήν την εργασία και για αυτόν τον λόγο ακολουθούμε τις DevOps πρακτικές.

Στην προκειμένη περίπτωση έχει δημιουργηθεί ενα .gitlab-ci.yaml αρχείο [47] έτσι ώστε κάθε φορά που θα γίνεται commit μια αλλαγή, εκτελείται το gitlab-ci.yaml αρχείο και εφόσον έχουμε ορίσει συνθήκη εκτέλεσης του build, τότε αυτό να γίνεται αυτόματα.

```
stages:
 - build
```

Το συγκεκριμένο αρχείο θα περιέχει μόνο ένα στάδιο, το build.

```
.base_build_image: &base_build_image
  stage: build
  image:
    name: gitlab-registry.cern.ch/ci-tools/docker-image-builder
    entrypoint: [""]
  script:
    - echo "{\"auths\":{\"$CI_REGISTRY\":{\"username\":\"$CI_REGISTRY_USER\",\"password\":\"$CI_REGISTRY_PASSWORD\"}}}" > /kaniko/.docker/config.json
    - /kaniko/executor --context $CI_PROJECT_DIR --dockerfile $CI_PROJECT_DIR/Dockerfile --destination $CI_REGISTRY_IMAGE:${CI_COMMIT_REF_NAME}
```

Στη συνέχεια χρησιμοποιείται ένα κομμάτι παραμετροποίησης που έχει δημιουργηθεί από την ομάδα web frameworks, το οποίο ορίζει κάποια πράγματα όπως το όνομα του build, το tag του image, το repository που θα αποθηκευτεί, και όλα αυτά μέσω μεταβλητών περιβάλλοντος. Το kaniko [48] είναι ένα εργαλείο που εκτελεί το docker build για εμάς.

Στο τέλος ορίζουμε πότε θέλουμε να εκτελείται το ανάλογο build, ανάλογα με το branch στο οποίο έγινε commit ο νέος κώδικας. Για παράδειγμα θέλουμε να αλλάξουμε την docker εικόνα της παραγωγής μας, μόνο όταν γίνει merge ο κώδικας στο master branch μας.

```
build_dev:
  <<: *base_build_image
```

```
  except:
    - master
```

```
build_master:
  <<: *base_build_image
  only:
    - master
```

Σε διαφορετική περίπτωση, αν βρισκόμαστε σε ένα branch διαφορετικό από το master, σε κάθε ανανέωση του κώδικα θα δημιουργείται και μια νέα εικόνα docker και θα αποθηκεύεται με ένα διαφορετικό tag στο repository μας.

Συνολικά με το συγκεκριμένο αρχείο για κάθε νέα αλλαγή κώδικα, φτιάχνουμε μια καινούργια εικόνα docker και την αποθηκεύουμε στο repository μας.

4.2 Αρχεία παραμετροποιήσεων

Στο Dockerfile έχουμε μερικά αρχεία τα οποία κάνουμε COPY, δηλαδή μεταφέρουμε, στο container μας. Αναφέρθηκε προς το παρόν μόνο το αρχείο που χρησιμοποιείται στο ENTRYPOINT, δηλαδή το αρχείο το οποίο εκκινεί την διεργασία του Apache server. Παρακάτω θα αναφερθούν συνοπτικά κάποια επιπλέον αρχεία:

config.ini.php: Πρόκειται για αρχείο παραμετροποίησης της εφαρμογής του Matomo. Κάποιες σημαντικές παράμετροι για την εφαρμογή μας είναι οι παρακάτω:

```
;force_ssl = 1 Access from outside the openshift cluster is protected by Shibboleth
```

Το συγκεκριμένο κομμάτι του κώδικα είναι comment, με λίγα λόγια είναι απενεργοποιημένη η προεπιλογή του redirection των http:// requests σε https://. Αυτο συμβαίνει επειδή όπως αναφέρθηκε η εφαρμογή μας είναι ήδη προστατευμένη πίσω από single sign-on και επίσης το OpenShift μπορεί να κάνει expose τον συγκεκριμένο σύνδεσμο μόνο μέσω https:// εφόσον το ρυθμίσουμε έτσι.

```
trusted_hosts[] = ${HOSTNAME_FQDN}
```

Αφήνουμε την εφαρμογή να είναι προσβάσιμη, μέσω της διεπαφής χρηστών της, στον υπερσύνδεσμο που παίρνει την τιμή του από την μεταβλητή περιβάλλοντος HOSTNAME_FQDN η οποία έχει οριστεί στην έναρξη του container.

; Minimum advised memory limit in Mb in php.ini file (see memory_limit value). Set to "-1" to always use the configured memory_limit value in php.ini file. Same for archiving.
minimum_memory_limit = -1
minimum_memory_limit_when_archiving = -1

Το Matomo από μόνο του αφήνει συγκεκριμένα όρια στα όρια της μνήμης που μπορεί να χρησιμοποιήσει η εφαρμογή. Στην περίπτωση μας αποδείχτηκε ότι σε κάποιες λειτουργίες, λόγω μεγάλου μεγέθους της βάσης δεδομένων, η εφαρμογή έπρεπε να χρησιμοποιήσει όλη την μνήμη που είναι διαθέσιμη στο container και εφόσον δεν έχει κάποια άλλη λειτουργία το container δεν υπάρχει κάποιο πρόβλημα σε αυτό.

[database]
host = ${DB_CONFIG_HOST}
port = ${DB_CONFIG_PORT}
username = ${DB_CONFIG_USERNAME}
password = ${DB_CONFIG_PASSWORD}
dbname = ${DB_CONFIG_NAME}
tables_prefix = ${DB_CONFIG_PREFIX}
adapter = PDO\MYSQL
type = InnoDB
schema = Mysql

Επίσης στο συγκεκριμένο αρχείο ορίζεται και ένας υπερχρήστης (superuser) στον οποίο θα ανήκει η εφαρμογή και θα έχει πλήρη δικαιώματα.

Τέλος βλέπουμε ότι ευαίσθητα στοιχεία της βάσης δεδομένων όπως όνομα χρήστη και κωδικός εισέρχονται στον κώδικα μέσω κάποιων μεταβλητών, οπότε δεν κινδυνεύουν αυτές οι πληροφορίες όταν ανεβάζουμε τον κώδικα μας σε κάποιο repository. Το πως δημιουργούνται αυτές οι μεταβλητές θα φανεί κατά τη διάρκεια του deployment.

httpd.conf: Το συγκεκριμένο αρχείο περιέχει τις γενικές ρυθμίσεις του Apache server μας. Δεν έχει κάτι ιδιαίτερο πέρα από το γεγονός ότι κάνει expose το σωστό hostname, ακούει στις σωστές θύρες και αφήνει μεγάλα headers καθώς κάποια ADFS_GROUP έχουν μεγάλη ονομασία.

```
Listen 8080
Listen 8081
ServerName https://${HOSTNAME_FQDN}
# allow large HTTP headers: users members of many groups can have a very large ADFS_GROUP
LimitRequestFieldsize 65536
```

shibboleth.sh: Θα αποτελεί στην ουσία το ENTRYPOINT του container που θα περιέχει το shibboleth. Οι εντολές είναι παρόμοιες με μόνη διαφορά ότι η τελευταία εντολή δεν ξεκινάει κάποιο apache server αλλά ξεκινάει τη διεργασία shibboleth.

```
exec /usr/sbin/shibd -F -c /etc/shibboleth/shibboleth2.xml
```

Πέρα από τα αρχεία παραμετροποίησης, μεταφέρονται στο container και τα αντίστοιχα plugins.

Το **LoginCERN**, το οποίο είχε αναπτυχθεί στο CERN από παλιότερα, εφόσον οι χρήστες επαληθεύσουν την ταυτότητα τους με το CERN SSO (Single Sign-On), δημιουργεί το αντίστοιχο όνομα χρήστη στην εφαρμογή και δίνει τα permissions μόνο αν έχει οριστεί το αντίστοιχο permission για τον συγκεκριμένο χρήστη ή βρίσκεται στην αντίστοιχη egroup ομάδα. Πιο συγκεκριμένα η μέθοδος που χρησιμοποιείται είναι η εξής: όταν κάποιος χρήστης συνδέεται στην εφαρμογή, πρέπει να επαληθεύσει την ταυτότητα του με κάποιον λογαριασμό χρήστη του CERN. Αν δεν έχει λογαριασμό χρήστη CERN, τότε η σελίδα τον κάνει redirect σε σελίδα που αναφέρει ότι δεν έχει δικαίωμα να μπει στην εφαρμογή. Αν έχει λογαριασμό CERN, τότε η εφαρμογή ελέγχει αν αυτός ο χρήστης υπάρχει ήδη στην βάση δεδομένων. Αν δεν υπάρχει, τότε δημιουργείται ένας χρήστης με όνομα χρήστη το όνομα χρήστη του λογαριασμού CERN, και με κωδικό ένα ψευδοτυχαίο αριθμό που μοιάζει με hash. Αν υπάρχει ο χρήστης, τότε δεν χρειάζεται να κάνει log in στην εφαρμογή, καθώς δεν γνωρίζει τον κωδικό του έτσι και αλλιώς, και η εφαρμογή ανοίγει απευθείας αφού πρόκειται για επαληθευμένο χρήστη. Αυτό βέβαια σημαίνει πως αν δεν είναι ενεργό το SSO, ένας οποιοσδήποτε χρήστης μπορεί να μπει στην εφαρμογή. Για αυτό είναι σημαντικό να είναι συνεχώς ενεργό το SSO.

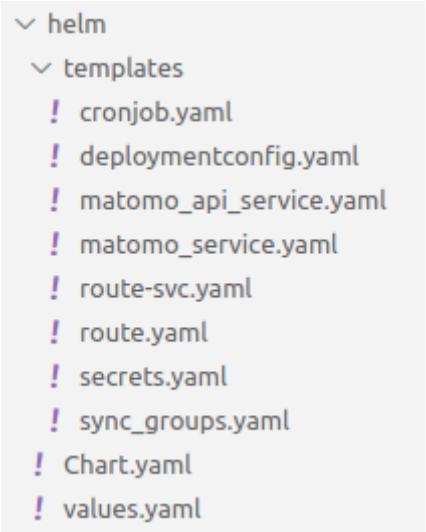
Το **GroupPermissions**, το οποίο είναι ένα υπάρχον plugin της κοινότητας του Matomo, επεκτάθηκε για τις ανάγκες του οργανισμού, καθώς έπρεπε να γίνει η αντιστοίχιση των groups στο Matomo με τα egroups του CERN.

Στο συγκεκριμένο plugin έχει αναπτυχθεί και μια συνάρτηση η οποία είναι είναι υπεύθυνη για τον συγχρονισμό των groups της εφαρμογής με τα egroups. Για κάθε group, ελέγχονται οι χρήστες που βρίσκονται στο συγκεκριμένο egroup, αν δεν υπάρχουν στο Matomo τους προσθέτει, ενώ αν έχουν αφαιρεθεί από το egroup, τότε τα δικαιώματα του εκάστοτε χρήστη υποβιβάζονται και έτσι σταματάει η πρόσβαση του στην εφαρμογή.

4.3 Deployment

4.3.1 Helm Chart

Για το deployment χρησιμοποιήθηκαν τα Helm Charts [49]. Το Helm Chart είναι directory το οποίο περιέχει αρχεία σε μορφή yaml, τα οποία περιγράφουν τα resources του kubernetes/openshift. Στη περίπτωση του Matomo, η δομή αυτή είναι η παρακάτω:



Ο φάκελος helm περιέχει όλα τα αρχεία του chart.

Chart.yaml:

Είναι το αρχείο που αναφέρει το όνομα του Chart όπως επίσης και κάποια επιπλέον στοιχεία όπως περιγραφή, έκδοση, δημιουργός κλπ.

```
name: matomo
appVersion: 0.1
description: Web Analytics using Matomo
keywords:
- webeos
- webinfra
- webservices
- matomo
home: https://cern.ch/web
maintainers:
- name: Ioannis Panagiotidis
  email: ioannis.panagiotidis@cern.ch
```

values.yaml:

Είναι το αρχείο το οποίο περιέχει όλες τις μεταβλητές που θα χρησιμοποιηθούν στα άλλα αρχεία. Συνήθως είναι το μεγαλύτερο αρχείο γιατί περιέχει τις μεταβλητές που θα χρησιμοποιούν όλα τα resources.

Το αρχείο ξεκινάει δηλώνοντας τους υπερσυνδέσμους, το όνομα και το namespace της εφαρμογής:

```
name: matomo
namespace: test-matomo
application_domain: test-matomo.web.cern.ch
api_domain: api-test-matomo.app.cern.ch
tag: BRANCH_NAME
```

Το BRANCH_NAME αντικαθίσταται κατά τη διάρκεια του CI/CD ώστε να ορίσει το αντίστοιχο tag του container.

```
secret:
 name: secr
 secrPassword: thepassword
 secrUsername: theusername
 matomoToken: justatoken
```

Αρχικοποιούνται τα μυστικά (secrets). Στην πραγματικότητα αυτές οι τιμές θα αντικατασταθούν, οπότε αποτελούν απλά ένα σύμβολο υποκατάστασης (placeholder).

```
containers:
 matomo:
  name: matomo
  image_name: gitlab-registry.cern.ch/webservices/matomo-projects/matomo:BRANCH_NAME
  imagePullPolicy: Always
  env:
  - name: NAMESPACE
    valueFrom:
      fieldRef:
        apiVersion: v1
        fieldPath: metadata.namespace
  - name: HOSTNAME_FQDN
```

```
    value: ""
  - name: DB_CONFIG_USERNAME
    valueFrom:
      secretKeyRef:
        name: secr
        key: usrname
  - name: DB_CONFIG_PASSWORD
    valueFrom:
      secretKeyRef:
        name: secr
        key: passwd
  - name: AUTH_GROUPS
    value: "it-dep-cda-wf"
  - name: DB_CONFIG_HOST
    value: "dbod-tmatomo2.cern.ch"
  - name: DB_CONFIG_PORT
    value: "5502"
  - name: DB_CONFIG_NAME
    value: "tmatomo"
  - name: DB_CONFIG_PREFIX
    value: "piwik_"
  ports:
  - containerPort: 8081
    protocol: TCP
  livenessProbe:
    exec:
      command:
      - python
      - liveness.py
    initialDelaySeconds: 5
    timeoutSeconds: 90
  readinessProbe:
    exec:
      command:
      - python
      - liveness.py
  resources:
    limits:
      cpu: "3000m"
      memory: 6Gi
    requests:
      cpu: "1000m"
      memory: 2Gi
  terminationMessagePath: /dev/termination-log
  terminationMessagePolicy: File
  volumeMounts:
      # shared mount for matomo & shibboleth containers
    - mountPath: /var/run/shibboleth
      name: shared
```

```
  - name: log-dir
    mountPath: /var/log/httpd
```

Παραπάνω ορίζεται το πρώτο container το οποίο είναι αυτό που περιέχει την εφαρμογή μας.

image_name: περιέχει τον σύνδεσμο του docker registry όπως και το tag του container. Στην προκειμένη περίπτωση χρησιμοποιήθηκε το GitLab Container Registry [50] καθώς το kaniko builder αποθηκεύει εκεί τις εικόνες μετά τη διαδικασία του build.

imagePullPolicy: Πάντα τραβάει την τελευταία έκδοση της εικόνας.

env: αναφέρονται οι μεταβλητές περιβάλλοντος του container. Στη μεταβλητή HOSTNAME η τιμή που δίνεται είναι κενή καθώς αυτή ανατίθεται στην έναρξη του container. Επίσης βλέπουμε ότι εδώ παίρνουν τιμή οι μεταβλητές που αφορούν την βάση δεδομένων, οι οποίες είχαν εμφανιστεί ξανά στο config.ini.php αρχείο παραμετροποιήσεων του Matomo. Οι τιμές οι οποίες δεν χρειάζεται να είναι μυστικές, μπορούν να δηλωθούν απευθείας. Ωστόσο οι τιμές οι οποίες πρέπει να είναι μυστικές, καθώς περιέχουν ευαίσθητες πληροφορίες, όπως για παράδειγμα ο κωδικός ή το όνομα χρήστη της βάσης δεδομένων, δε δηλώνονται απευθείας, αλλά παίρνουν την τιμή τους από ένα secretKeyRef.

ports: δηλώνεται η θύρα στην οποία θα ακούει το container.

livenessProbe και **readinessProbe:** αφορούν κάποια scripts τα οποία ελέγχουν την κατάσταση και την υγεία του container.

resources: ορίζονται οι πόροι που θα καταναλώνονται από το container. Μπορούν να οριστούν δύο διαφορετικά πλαίσια, το requests και το limits. Στο requests ορίζονται οι ελάχιστες τιμές μνήμης και πυρήνων επεξεργαστή που πρέπει να διατεθούν κατά τη δημιουργία του container. Αν το μηχάνημα δεν έχει αυτές τις ελάχιστες τιμές διαθέσιμες, τότε το container θα συνεχίζει να βρίσκεται σε κατάσταση «δημιουργίας» μέχρι να βρεθούν οι απαραίτητοι πόροι. Στο limits ορίζονται οι μέγιστες τιμές μνήμης και πυρήνων επεξεργαστή που μπορεί να χρησιμοποιήσει το συγκεκριμένο container.

volumeMounts: αφορούν φακέλους ή αλλιώς directories, οι οποίοι μπορούν να μοιράζονται και να βρίσκονται ταυτόχρονα σε πάνω από ένα container. Πρόκειται για ένα resource το οποίο δημιουργείται σε κάθε εκκίνηση ενός pod και μπορεί να προσκολληθεί σε κάποιο μονοπάτι (path) σε κάθε container. Έτσι αν για παράδειγμα αποθηκεύουμε εκεί τα logs μιας εφαρμογής που τρέχει σε ένα container, αυτά τα logs θα είναι διαθέσιμα και σε ένα άλλο container που εκτελείται στο pod.

Στη συνέχεια δηλώνονται με παρόμοιο τρόπο και τα υπόλοιπα containers:

```
shibboleth:
  name: shibboleth
  image_name: gitlab-registry.cern.ch/webservices/matomo-projects/matomo:BRANCH_NAME
```

```
imagePullPolicy: Always
command: /shibboleth.sh
```

Αυτό που αλλάζει στο shibboleth container είναι ότι αλλάζουμε την εντολή (command) με την οποία γίνεται η εκκίνηση του container. Επίσης, έχει διαφορετικά resources και μεταβλητές περιβάλλοντος, αλλά έχει κοινό το shibboleth volumeMount.

```
volumeMounts:
  # shared mount for matomo & shibboleth containers
  - mountPath: /var/run/shibboleth
    name: shared
```

Αντίστοιχα, το logstash container έχει κοινό το άλλο volumeMount:

```
volumeMounts:
  - name: log-dir
    mountPath: /data
```

Από εκεί λαμβάνει τα logs του Matomo container και τα στέλνει στο Cern Central Monitoring.

Αυτά τα δυο volumeMounts ορίζονται ως emtyDir δηλαδή ως κενοί φάκελοι:

```
volumes:
  - name: shared
    emptyDir: {}
  - name: log-dir
    emptyDir: {}
```

Οι κενοί φάκελοι έχουν την ιδιότητα ότι χάνουν τις πληροφορίες που έχουν αποθηκεύσει σε περίπτωση που εμφανιστεί σφάλμα στο container. Αυτό τα κάνει ιδανικά σε περίπτωση που τα δεδομένα που αποθηκεύονται εκεί χρειάζονται για προσωρινή χρήση, ή για να σταλούν αλλού όπως στην περίπτωση μας, αλλά δεν βολεύουν καθόλου σε περίπτωση που χρειαζόμαστε αυτά τα δεδομένα και πρέπει να τα αποθηκευσουμε. Σε αυτές τις περιπτώσεις χρειαζόμαστε persistent storage, το οποίο είναι αχρείαστο στην περίπτωση αυτής της εφαρμογής.

Οι τιμές που ορίζονται στο αρχείο values.yaml δεν είναι απαραίτητο ότι είναι αυτές που θα χρησιμοποιούνται σε διάφορα deployments. Αυτά αποτελούν απλά ένα πρότυπο. Αυτά τα αρχεία, σε διαφορετικές περιπτώσεις, μπορούν να αντικατασταθούν από τιμές που θα ορίζονται σε διαφορετικά περιβάλλοντα (environments) όπως θα εξηγηθεί παρακάτω. Ωστόσο στην προκειμένη περίπτωση οι τιμές του αρχείου values.yaml είναι αυτές που

χρησιμοποιούνται για να γίνει το deployment σε ένα dev cluster, δηλαδή ένα cluster με σκοπό το testing του deployment.

Παρακάτω θα οριστούν τα OpenShift resources που χρησιμοποιούνται στο deployment μας. Να σημειωθεί ότι τα OpenShift resources είναι στην ουσία Kubernetes resources, είτε ακριβώς ίδια, είτε με κάποιες διαφοροποιήσεις.

sync_groups.yaml:

Είναι ένα αρχείο είδους CronJob [51]. Τα CronJobs αποτελούν εργασίες οι οποίες εκτελούνται σε προγραμματισμένο πλαίσιο επαναλαμβανόμενα.

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
 name: syncegroups
```

Το συγκεκριμένο CronJob θα εκτελείται κάθε ώρα, 10 λεπτά πριν την αλλαγή της ώρας.

```
spec:
 schedule: "50 * * * *"
 successfulJobsHistoryLimit: 3
 failedJobsHistoryLimit: 3
 startingDeadlineSeconds: 200
 jobTemplate:
   spec:
     template:
       metadata:
         labels:
           parent: "syncing"
       spec:
         containers:
         - name: matomo-sync-egroup
           image: gitlab-registry.cern.ch/webservices/matomo-projects/matomo:{{ .Values.tag }}
           command: ["/bin/sh", "-c", 'curl -k https://${api_domain}/?module=API -
d"method=GroupPermissions.syncEGroup&token_auth=${MATOMO_API_TOKEN_AUTH}"']
           env:
{{ toYaml .Values.containers.matomo.env | indent 14 }}
         restartPolicy: Never
```

Η εντολή η οποία επαναλαμβάνεται είναι μια εντολή του συστήματος Linux με όνομα curl [52], η οποία στέλνει ένα αίτημα σε ένα συγκεκριμένο σύνδεσμο και λαμβάνει κάποια αναμενόμενη απάντηση. Ο σύνδεσμος ο οποίος χρησιμοποιείται είναι:

```
https://${api_domain}/?module=API -d"method=GroupPermissions.syncEGroup&token_auth=${MATOMO_API_TOKEN_AUTH}
```

Ο παραπάνω σύνδεσμος φανερώνει κάποια στοιχεία. Το request μας αποστέλλεται σε έναν σύνδεσμο ο οποίος βρίσκεται πίσω από μια μεταβλητή, την api_domain, η οποία ορίζεται στο values.yaml αρχείο. Η συνάρτηση η οποία εκτελείται βρίσκεται στο plugin του GroupPermissions και την έχουμε δημιουργήσει εμείς. Για να στείλει κάποιος ένα επιτυχημένο request θα πρέπει να περιέχει έναν κλειδάριθμο (token), το οποίο είναι μοναδικό για κάθε ταυτοποιημένο χρήστη του Matomo. Αυτός ο κλειδάριθμος παραμένει κρυφός.

cronjob.yaml:

Είναι ένα δεύτερο CronJob που έχει ως σκοπό να εκτελέσει το archiving του Matomo.

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
 name: matomoarchiver
spec:
 schedule: "50 * * * *"
 successfulJobsHistoryLimit: 3
 failedJobsHistoryLimit: 3
 startingDeadlineSeconds: 200
 concurrencyPolicy: Forbid
 jobTemplate:
   spec:
     backoffLimit: 0
     template:
       metadata:
         labels:
           parent: "archiving"
       spec:
         containers:
         - name: matomo-archiver
           image: gitlab-registry.cern.ch/webservices/matomo-projects/matomo:{{ .Values.tag }}
           command: ["/bin/sh", "-c", "source scl_source enable rh-php71; ./opt/matomo/console core:archive --url=https://${api_domain} --ansi"]
           env:
{{ toYaml .Values.containers.matomo.env | indent 14 }}
         restartPolicy: Never
```

deploymentconfig.yaml

Πρόκειται για το βασικό μας deployment. Ο τρόπος με τον οποίο ορίσαμε το values.yaml αρχείο μας επιτρέπει απλώς να αντικαταστήσουμε τις αντίστοιχες τιμές στο deploymentconfig.yaml.

```
  spec:
    containers:
      #-----------------------------
      # Matomo Container
      -
        name: {{ .Values.containers.matomo.name | quote }}
        image: {{ .Values.containers.matomo.image_name | quote }}
        imagePullPolicy: {{ .Values.containers.matomo.imagePullpolicy | quote }}
        env:
{{ toYaml .Values.containers.matomo.env | indent 12 }}
        ports:
{{ toYaml .Values.containers.matomo.ports | indent 12 }}
        livenessProbe:
{{ toYaml .Values.containers.matomo.livenessProbe | indent 12 }}
        readinessProbe:
{{ toYaml .Values.containers.matomo.readinessProbe | indent 12 }}
        resources:
{{ toYaml .Values.containers.matomo.resources | indent 12 }}
        terminationMessagePath: {{ .Values.containers.matomo.terminationMessagePath | quote
}}
        terminationMessagePolicy:  {{  .Values.containers.matomo.terminationmessagePolicy  |
quote }}
        volumeMounts:
          # shared mount for matomo & shibboleth containers
{{ toYaml .Values.containers.matomo.volumeMounts | indent 12 }}
```

Οι τιμές για τις παραμέτρους του container λαμβάνονται από το αρχείο Values στην αντίστοιχη θέση μεταβλητής.

Με ίδιο τρόπο δηλώνονται και τα Volumes:

```
    # Container volumes
    volumes:
{{ toYaml .Values.containers.volumes | indent 10 }}
```

matomo_service.yaml:

Αποτελεί το service το οποίο θα συνδέεται στο container ώστε να μπορεί να γίνει πρόσβαση σε αυτό εκτός του cluster, μέσω κάποιου υπερσυνδέσμου.

```
kind: Service
apiVersion: v1
metadata:
 name: {{ .Values.name | quote }}
 labels:
   app: {{ .Values.name | quote }}
spec:
 ports:
 - name: "8080-tcp"
   port: 8080
   protocol: TCP
   targetPort: 8080
 selector:
   app: {{ .Values.name | quote }}
   deploymentconfig: {{ .Values.name | quote }}
 sessionAffinity: None
 type: ClusterIP
```

Αντίστοιχα δηλώνεται και το matomo_api_service το οποίο θα συνδεθεί και αυτό στο container, σε άλλη θύρα, ώστε να κάνει προσβάσιμο τον υπερσύνδεσμο (route) για τα API αιτήματα .

```
kind: Service
apiVersion: v1
metadata:
 name: "matomo-api"
 labels:
   app: {{ .Values.name | quote }}
spec:
 clusterIP: None
 ports:
 - name: "api-port"
   port: 8080
   protocol: TCP
   targetPort: 8081
 selector:
   app: {{ .Values.name | quote }}
   deploymentconfig: {{ .Values.name | quote }}
 sessionAffinity: None
 type: ClusterIP
```

Πάνω σε αυτά τα δυο service θα προσκολληθούν τα αντίστοιχα routes. Τα routes είναι τα resources που περιέχουν τον σύνδεσμο ο οποίος δείχνει στα αντίστοιχα service.

```
kind: Route
```

```
apiVersion: route.openshift.io/v1
metadata:
 name: {{ .Values.name | quote }}
 labels:
   app: {{ .Values.name | quote }}
   cern.ch/sso-registration: Shibboleth
spec:
 port:
   targetPort: "8080-tcp"
 tls:
   insecureEdgeTerminationPolicy: Redirect
   termination: edge
 to:
   kind: Service
   name: {{ .Values.name | quote }}
```

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
 name: "api-matomo"
 labels:
   app: {{ .Values.name | quote }}
spec:
 host: {{ .Values.api_domain | quote }}
 port:
   targetPort: "api-port"
 tls:
   insecureEdgeTerminationPolicy: Redirect
   termination: edge
 to:
   kind: Service
   name: "matomo-api"
```

Όπως φαίνεται μόνο το πρώτο service προστατεύεται από το SSO.

secrets.yaml:

Είναι το αρχείο στο οποίο κρυπτογραφούνται τα μυστικά μας.

```
apiVersion: v1
kind: Secret
metadata:
 name: {{ .Values.secret.name | quote }}
 namespace: {{ .Values.namespace | quote }}
type: Opaque
```

```
data:
 usrname: {{ .Values.secret.secrUsername | b64enc | quote }}
 passwd: {{ .Values.secret.secrPassword | b64enc | quote }}
 thetoken: {{ .Values.secret.matomoToken | b64enc | quote }}
```

Οι μεταβλητές αυτές χρησιμοποιούν τις τιμές που ορίστηκαν στο αρχείο Values. Ωστόσο, εκείνο το αρχείο δεν περιέχει τις πραγματικές τιμές αυτών των μεταβλητών. Αυτές οι τιμές αντικαθίστανται στο Values αρχείο μέσω του CI/CD.

Όλα τα παραπάνω αποτελούν το Helm Chart της εφαρμογής.

4.3.2 Περιβάλλοντα

Τα περιβάλλοντα (environments) είναι ένας εύκολος τρόπος να αντικαταστήσουμε τιμές στο αρχείο values για διαφορετικές περιπτώσεις. Έστω ότι είναι ανάγκη να υπάρχουν deployed πολλαπλά instances του Matomo, με λίγα λόγια να υπάρχουν διαφορετικά Matomo για διαφορετικές χρήσεις. Αυτό σημαίνει για παράδειγμα ότι πρέπει να υπάρχουν διαφορετικές βάσεις δεδομένων και να ορίζονται τα στοιχεία της καθεμιάς ξεχωριστά.

Είναι ανάγκη λοιπόν να μην γίνονται διαφορετικά deployment για κάθε instance. Ωστόσο, δεν χρειάζονται να αλλάξουν πολλά κομμάτια του deployment, αλλά μόνο κάποια συγκεκριμένα. Αυτό επιτυγχάνεται με τα environments. Το κάθε environment περιέχει ένα δικό του values.yaml αρχείο το οποίο θα αντικαταστήσει κάποιες ορισμένες τιμές στο αρχικό values.yaml αρχείο του chart μας.

Για παράδειγμα έστω το matomo-statistics, ένα instance μιας συγκεκριμένης χρήσης.

```
name: matomo-statistics
namespace: test-matomo-statistics
application_domain: test-matomo-statistics.web.cern.ch
api_domain: api-test-matomo-statistics.apptest.cern.ch

containers:
 matomo:
  env:
  - name: NAMESPACE
    valueFrom:
      fieldRef:
        apiVersion: v1
        fieldPath: metadata.namespace
  - name: HOSTNAME_FQDN
    value: ""
  - name: DB_CONFIG_USERNAME
    valueFrom:
```

```
      secretKeyRef:
        name: secr
        key: usrname
  - name: DB_CONFIG_PASSWORD
    valueFrom:
      secretKeyRef:
        name: secr
        key: passwd
  - name: AUTH_GROUPS
    value: "it-dep-cda-wf"
  - name: DB_CONFIG_HOST
    value: "dbod-tmatomo.cern.ch"
  - name: DB_CONFIG_PORT
    value: "5507"
  - name: DB_CONFIG_NAME
    value: "tmatomo"
  - name: DB_CONFIG_PREFIX
    value: "matomo_"
  - name: MATOMO_API_TOKEN_AUTH
    valueFrom:
      secretKeyRef:
        name: secr
        key: thetoken
  - name: api_domain
    value: "api-test-matomo-statistics.apptest.cern.ch"
```

Αυτό είναι το αρχείο στην ολοκληρωμένη του μορφή. Αλλάζει τα ονόματα των συνδέσμων της εφαρμογής όπως και τις μεταβλητές περιβάλλοντος και συγκεκριμένα αυτές τις μεταβλητές που αφορούν τη βάση δεδομένων.

4.3.3 GitLab CI/CD

Το deployment της εφαρμογής στα αντίστοιχα clusters έχει ένα δικό του gitlab-ci.yaml αρχείο το οποίο είναι υπεύθυνο για αυτή τη διαδικασία.

Στο gitlab-ci.yaml ορίζονται όλες οι μεταβλητές που χρησιμοποιεί στη συνέχεια το deployment, όπως και τα αντίστοιχα μυστικά (secrets).

```
variables:
 HELM_VERSION: 2.14.3
 HELM_RELEASE_NAME: matomo


stages:
```

```
- deploy
```

Βλέπουμε ότι το μόνο στάδιο που κάνει το συγκεκριμένο ci είναι το deploy.

Στη συνέχεια ορίζονται τα environments του κάθε cluster.

Για παράδειγμα το dev cluster:

```
.dev_environment: &dev_environment
  name: dev
  url: ${OPENSHIFT_SERVER}/console/project/${NAMESPACE}
```

```
.dev_variables: &dev_variables
  OPENSHIFT_SERVER: https://openshift-dev.cern.ch
  NAMESPACE: test-matomo
  HELM_VALUES: ${HELM_VALUES_DEV}
  TILLER_TOKEN: ${SERVICE_ACCOUNT_TOKEN_DEV}
  TILLER_NAMESPACE: ${NAMESPACE}
  ENV_SPECIFIC_VALUES: environments/dev/values.yaml
```

Δηλώνεται ο server στον οποίο θα γίνει το deployment και το όνομα του namespace που θα χρησιμοποιηθεί.

Τα μυστικά περνιούνται μέσα στο deployment μέσω της εντολής:

```
HELM_VALUES: ${HELM_VALUES_DEV}
```

Το HELM_VALUES_DEV είναι ένα «GitLab CI/CD environment variable» [53], δηλαδή μια μεταβλητή περιβάλλοντος Gitlab CI/CD. Εκεί μπορούμε με ασφάλεια να αποθηκεύσουμε τα μυστικά μας και να τα αντικαταστήσουμε κατά τη διάρκεια του deployment χωρίς να εκθέσουμε πουθενά αυτά τα μυστικά. Επίσης σε κάθε environment δηλώνεται και ο αντίστοιχος φάκελος με τις παραμέτρους του εκάστοτε environment οι οποίες δεν χρειάζεται να είναι κρυφές.

```
ENV_SPECIFIC_VALUES: environments/dev/values.yaml
```

Στην συνέχεια ορίζεται το τελικό βήμα του deployment.

```
.base_deploy: &base_deploy
 stage: deploy
 image: gitlab-registry.cern.ch/paas-tools/openshift-client
```

Πριν τρέξουμε το script του deployment πρέπει να γίνει η απαραίτητη προεργασία για να είναι σωστά εγκατεστημένος και ο tiller και να αναγνωριστεί σωστά το Helm Chart μας. Ο tiller αποτελεί το server κομμάτι του Helm, «ακούει» τις εντολές του Helm και τις εκτελεί στο cluster.

```
before_script:
 # add Openshift CA to trusted CAs if needed
 - if [ -n "${OPENSHIFT_CA}" ]; then echo "$OPENSHIFT_CA" > /etc/pki/ca-trust/source/anchors/openshift.pem && update-ca-trust; fi
 # login first: we'll need the kubeconfig before starting local tiller
 - oc project ${NAMESPACE} --token ${TILLER_TOKEN} --server ${OPENSHIFT_SERVER}
 # download helm
 - curl "https://kubernetes-helm.storage.googleapis.com/helm-v${HELM_VERSION}-linux-amd64.tar.gz" | tar zx
 - mv --force linux-amd64/{helm,tiller} /usr/bin/
 - cd helm
 - sed -i "s/BRANCH_NAME/${CI_COMMIT_REF_NAME}/g" values.yaml
 # initialize local tiller
 - cd ..
 - helm init --client-only
 - helm plugin install https://github.com/adamreese/helm-local
 - command -v which || yum install -y which # helm local plugin needs `which`, install if missing
 - helm local start
 - helm local status
 - export HELM_HOST=":44134"
```

Εφόσον εγκατασταθεί και ενεργοποιηθεί ο tiller, το deployment είναι έτοιμο να εκτελεστεί. Αντικαθιστούμε τις τιμές των μεταβλητών του gitlab πάνω στο values.yaml αρχείο και στη συνέχεια εκτελείται το helm deployment με την εντολή helm upgrade --install.

```
script:
 # Create a values file with Openstack username/password/projectID
 - echo "${HELM_VALUES}" > values.yaml
 - helm upgrade ${HELM_RELEASE_NAME} helm/ --values values.yaml --values ${ENV_SPECIFIC_VALUES} --install --namespace ${NAMESPACE} --cleanup-on-fail
```

Οι μόνες τιμές που αλλάζουν ανά περιβάλλον ορίζονται στο τέλος του αρχείου:

```
deploy_dev:
 <<: *base_deploy
 environment:
   <<: *dev_environment
 except:
```

```
  - master
when: manual
variables:
  <<: *dev_variables
```

Αντίστοιχα με το gitlab-ci.yaml αρχείο που κάνει build την εικόνα docker, το gitlab-ci.yaml αρχείο που κάνει deploy το λογισμικό, εκτελεί το αντίστοιχο deployment ανάλογα με το branch στο οποίο έγιναν οι αλλαγές. Συνήθως οι αλλαγές της παραγωγής γίνονται εφόσον γίνεται merge ο κώδικας στο master branch.

Αντίστοιχα μπορούμε να δημιουργήσουμε πολλά περιβάλλοντα (environments) σε περίπτωση που είναι απαραίτητο να γίνουν αλλαγές στο λογισμικό.

Περισσότερες λεπτομέρειες για το συγκεκριμένο πρότζεκτ μπορούν να βρεθούν στο αντίστοιχο repository [54] του CERN.

Ενότητα 5: Συζήτηση

Συνοψίζοντας, παρουσιάστηκε ο τρόπος με τον οποίο μεταφέρθηκε η εφαρμογή του Matomo σε τεχνολογίες container στο CERN. Μια αρχική παρατήρηση που μπορεί να γίνει είναι ότι η διαδικασία του deployment της εφαρμογής έχει μια αρκετά μεγάλη πολυπλοκότητα. Αυτό σημαίνει ότι μερικές εφαρμογές και ιστοσελίδες ίσως να μην δικαιολογούν τη μεταφορά τους σε τεχνολογίες container. Θα πρέπει να γίνει ξεκάθαρο από τις προδιαγραφές της κάθε περίπτωσης ξεχωριστά αν αξίζει να μεταφερθεί μια εφαρμογή σε containers ή όχι. Ο στόχος της ομάδας Web Frameworks είναι να μεταφέρει όλες τις ιστοσελίδες και εφαρμογές για τις οποίες είναι υπεύθυνη, δηλαδή 14000 ιστοσελίδες και εφαρμογές. Ωστόσο για τις εφαρμογές που δεν δικαιολογούν την παραπάνω πολυπλοκότητα μετανάστευσης θα πρέπει να βρεθεί κάποιος πιο απλός ή αυτοματοποιημένος τρόπος μεταφοράς τους σε containers.

Πέρα από το παραπάνω όμως βλέπουμε και αρκετά πλεονεκτήματα. Για παράδειγμα, η ενημέρωση της εφαρμογής κανονικά περιείχε όλη την διαδικασία της εγκατάστασης της εφαρμογής από την αρχή στον αντίστοιχο server και θα έπρεπε να προσέχουμε κατά τη μετάβαση από την μια έκδοση σε μια άλλη ώστε να μην υπάρχει καθόλου διακοπή στη λειτουργία (downtime). Τώρα η παραπάνω διαδικασία γίνεται απλά με αλλαγή μιας γραμμής στο αντίστοιχο Dockerfile, και στη συνέχεια το commit του κώδικα. Αυτή η διαδικασία πιάνει κυριολεκτικά δευτερόλεπτα για να γίνει και το OpenShift περνάει στην επόμενη έκδοση μόνο όταν είναι πλήρως λειτουργικό το νέο container, εξασφαλίζοντας μηδενική διακοπή.

Επιπλέον μπορούν να γίνουν αλλαγές και πειραματισμοί σε διαφορετικά cluster και με διαφορετικές παραμετροποιήσεις πολύ εύκολα. Δεν χρειάζεται να αλλάξουμε τίποτα στον υπάρχοντα κώδικα, απλά να προσθέσουμε τα αντίστοιχα μυστικά (secrets) και τις αντίστοιχες ρυθμίσεις στο νέο περιβάλλον (environment) και να κάνουμε commit τις αλλαγές. Με λίγα λόγια ενώ η αρχική πολυπλοκότητα ώστε να δημιουργηθεί ένα πρότυπο είναι μεγάλη, στη συνέχεια η συντήρηση και ανάπτυξη του λογισμικού γίνεται πολύ ευκολότερη. Μπορούμε να δημιουργούμε πολλά instances της εφαρμογής και να μεταπηδάμε σε διαφορετικές βάσεις δεδομένων αλλάζοντας απλά το αντίστοιχο αρχείο.

Τέλος, η μεταφορά στις νέες τεχνολογίες προσφέρει όλα τα πλεονεκτήματα τα οποία αναφέρθηκαν στα προηγούμενα κεφάλαια. Με τέτοιες ευέλικτες τεχνολογίες μπορεί να ανέβει κατακόρυφα η αποδοτικότητα των ομάδων που τις χρησιμοποιούν και να μπορέσουν αυτές να εξελίξουν τις υπηρεσίες που προσφέρουν. Μειώνεται η πολυπλοκότητα διαχείρισης της υποδομής αλλά και η επίλυση σφαλμάτων.

Βιβλιογραφία και ηλεκτρονικές αναφορές

1. L. Mamatas, A. Papadopoulou and V. Tsaoussidis, Towards an Opportunistic Software-Defined Networking Solution
2. C. Sarros, S. Diamantopoulos, S. Rene, I. Psaras, A. Lertsinsrubtavee, C. Molina-Jimenez, P. Mendes, R. Sofia, A. Sathiaseelan, G. Pavlou, and V. Tsaoussidis, Connecting the Edges: A Universal, Mobile-Centric, and Opportunistic Communications Architecture
3. "CERN", CERN. Retrieved from https://home.cern/
4. FreeBSD Handbook, Chapter 14. Jails. Retrieved from https://www.freebsd.org/doc/handbook/jails.html
5. INTRODUCTION TO CONTROL GROUPS (CGROUPS). Retrieved from RedHat: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/resource_management_guide/ch01
6. systemd. Retrieved from https://www.freedesktop.org/wiki/Software/systemd/
7. dotCloud is becoming Docker. Retrieved from https://www.docker.com/blog/dotcloud-is-becoming-docker-inc/
8. "The Benefits of Containerization and What It Means for You", IBM [Online]. Retrieved from https://www.ibm.com/cloud/blog/the-benefits-of-containerization-and-what-it-means-for-you
9. "Docker", Wikipedia. Retrieved from https://el.wikipedia.org/wiki/Docker
10. Feller, J., & Fitzgerald, B. (2002). Understanding open source software development
11. Docker. Retrieved from https://www.docker.com/company
12. Docker overview. Retrieved from https://docs.docker.com/get-started/overview/
13. Τι είναι το Virtualization. Retrieved from http://www.ivisinfo.gr/articles/virtualization-technology.php
14. Cloud virtualization. Retrieved from https://www.w3schools.in/cloud-computing/cloud-virtualization/
15. Containers vs Virtual Machines. Retrieved from https://blog.netapp.com/blogs/containers-vs-vms/
16. What are microservices. Retrieved from https://www.redhat.com/en/topics/microservices/what-are-microservices
17. C. Richardson, 2017, Pattern: Microservice Architecture. Retrieved from https://microservices.io/patterns/microservices.html
18. Microservices Architecture. Retrieved from https://dzone.com/articles/microservice-architecture-learn-build-and-deploy-a
19. What is DevOps? Retrieved from https://aws.amazon.com/devops/what-is-devops/

20. What is container orchestration? Retrieved from https://www.redhat.com/en/topics/containers/what-is-container-orchestration
21. “Kubernetes,” Kubernetes. Retrieved from: https://kubernetes.io/
22. What is Kubernetes? Retrieved from https://www.redhat.com/en/topics/containers/what-is-kubernetes
23. “CERN”, Wikipedia. Retrieved from https://el.wikipedia.org/wiki/CERN
24. B. Noel, D. Michelino, M. Velten, R. Rocha and S. Trigazis, CERN, 2017, Integrating Containers in the CERN Private Cloud
25. Cern-vm filesystem. Retrieved from https://cernvm.cern.ch/portal/filesystem
26. G. Adde and B. Chan, D. Duellmann, X. Espinal, A. Fiorot, J. Iven, L. Janyst, M. Lamanna, L. Mascetti, J. M. Pereira Rocha, A. J. Peters, E. A. Sindrilaru, CERN, 2015, Latest evolution of EOS filesystem
27. Docker cvmfs plugin. Retrieved from https://gitlab.cern.ch/cloud-infrastructure/docker-volume-cvmfs
28. Docker eos plugin. Retrieved from https://gitlab.cern.ch/cloud-infrastructure/docker-volume-eos
29. Kubernetes flexvolume plugin. Retrieved from https://github.com/kubernetes/community/blob/master/contributors/devel/sig-storage/flexvolume.md
30. “OKD”, Red Hat. Retrieved from https://www.okd.io/
31. A. Lossent, A. R. Peon and A. Wagner, CERN, 2017, PaaS for web applications with OpenShift Origin
32. HAProxy, "HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer”. Retrieved from http://www.haproxy.org/
33. Internet Engineering Task Force (IETF), "RFC6066 - Transport Layer Security (TLS) Extensions: Extension Definitions". Retrieved from https://tools.ietf.org/html/rfc6066
34. The Jenkins Project, "Jenkins - Build great things at any scale". Retrieved from https://www.jenkins.io/
35. "Grafana Documentation", Grafana Labs. Retrieved from https://grafana.com/docs/grafana/latest/
36. R. G. Aparicio and I. C. Coz, CERN, 2015, Database on Demand: insight how to build your own DBaaS
37. Matomo Analytics - The Google Analytics alternative that protects your data. Retrieved from https://matomo.org/home/
38. Computer Security: The New Single Sign-On. Retrieved from https://home.cern/news/news/computing/computer-security-new-single-sign
39. egroups, Retrieved from https://e-groups.cern.ch/
40. Apache HTTP Server Project. Retrieved from https://httpd.apache.org/

41. Shibboleth 2. Retrieved from https://wiki.shibboleth.net/confluence/display/SHIB2/Home
42. Helm. Retrieved from https://helm.sh/docs/
43. YAML. Retrieved from https://yaml.org/
44. Dockerfile reference. Retrieved from https://docs.docker.com/engine/reference/builder/
45. Logstash: Collect, Parse, Transform Logs. Retrieved from https://www.elastic.co/logstash
46. Elasticsearch. Retrieved from https://www.elastic.co/what-is/elasticsearch
47. GitLab CI/CD. Retrieved from https://docs.gitlab.com/ee/ci/
48. Building images with kaniko and GitLab CI/CD. Retrieved from https://docs.gitlab.com/ee/ci/docker/using_kaniko.html
49. Helm Chart. Retrieved from https://helm.sh/docs/topics/charts/
50. GitLab Container Registry. Retrieved from https://docs.gitlab.com/ee/user/packages/container_registry/
51. Wikipedia, cron. Retrieved from https://en.wikipedia.org/wiki/Cron
52. curl. Retrieved from https://curl.haxx.se/
53. GitLab CI/CD environment variables. Retrieved from https://docs.gitlab.com/ee/ci/variables/
54. “Matomo-projects”, CERN. Retrieved from https://gitlab.cern.ch/webservices/matomo-projects