ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη πλαισίου ασφάλειας-αξιοπιστίας νεφοϋπολογιστικών εφαρμογών βασισμένων σε τεχνολογίες περιεκτών και μικρουπηρεσιών

Φοιτητής: **Αλέξανδρος Τσιαπάρας**

Επιβλέπων: **Βασίλειος Μάμαλης**

Συν-επιβλέπων: **Απόστολος Αναγνωστόπουλος**

Μάρτιος, 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη πλαισίου ασφάλειας-αξιοπιστίας νεφοϋπολογιστικών εφαρμογών βασισμένων σε τεχνολογίες περιεκτών και μικρουπηρεσιών

Εξεταστική Επιτροπή

Καντζάβελου Ιωάννα, Επ. Καθηγήτρια

Μάμαλης Βασίλειος, Καθηγητής

Μπόγρης Αντώνιος, Καθηγητής

Ημερομηνία Εξέτασης: 22/03/2024

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Τσιαπάρας Αλέξανδρος του Βασιλείου, με αριθμό 71141218 φοιτητής του Προγράμματος Προπτυχιακών Σπουδών του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής, δηλώνω ότι:

«Βεβαιώνω ότι είμαι συγγραφέας αυτής της Διπλωματικής εργασίας και κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο Δηλών

Τσιαπάρας Αλέξανδρος

____

Ευχαριστίες

Στο πλαίσιο της εκπόνησης της παρούσας διπλωματικής εργασίας, θα ήθελα να ευχαριστήσω τον κ. Βασίλειο Μάμαλη και τον κ. Απόστολο Αναγνωστόπουλο για την πολύτιμη βοήθειά τους.

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική εργασία εμβαθύνει στο πεδίο του υπολογιστικού νέφους, δίνοντας έμφαση στους ρόλους των περιεκτών, των μικρουπηρεσιών και της ενορχήστρωσης για την δημιουργία κλιμακούμενων και ασφαλών εφαρμογών. Γίνεται έρευνα των αρχιτεκτονικών αυτών των τεχνολογιών και των προκλήσεων τους στην ασφάλεια και αξιοπιστία, με σκοπό να υλοποιηθεί ένα πλαίσιο με προτεινόμενες τεχνολογίες για την διαχείριση αυτών των προκλήσεων. Τέλος το πλαίσιο δοκιμάζεται μέσα από υλοποίηση που προσεγγίζει την αρχιτεκτονική ενός δικτύου Μηδενικής εμπιστοσύνης και ελάχιστου προνομίου.

Λέξεις κλειδία: υπολογιστική νέφους, περιέκτες, μικρουπηρεσίες, DevSecOps, Κυβερνήτης, ασφάλεια, αξιοπιστία, CI/CD

ABSTRACT

This thesis explores the topic of cloud computing, focusing on the functions of orchestration, microservices, and content to build safe and scalable applications. In order to apply a framework of suggested technologies to meet these difficulties, an examination into the architectures of these technologies as well as their security and reliability challenges is carried out. Finally, the framework is tested through an implementation that approximates the architecture of a Zero Trust network and Minimum Privilege approach.

***Keywords**:* cloud computing, containerization, microservices, DevSecOps, Kubernetes, security, reliability, CI/CD

Κατάλογος Εικόνων

2.1 Μονολιθική αρχιτεκτονική και αρχιτεκτονική μικρουπηρεσιών 7
2.2 DevOps pipeline 8

3.1 Περιέκτες και εικονικοποίηση 13
3.2 Ομάδες ελέγχου 14
3.3 Χώρος ονομάτων 15
3.4 Αρχιτεκτονική Docker 16

4.1 Αρχιτεκτονική Kubernetes 19
4.2 Έναρξη minikube 26

5.1 Microsoft Entra ID 30

6.1 Παράδειγμα SAML 37
6.2 Διάγραμμα PCKE 40
6.3 Αρχιτεκτονική Istio 43
6.4 Hashicorp Vault Transit Engine 45

7.1 Αναπαράσταση εσωτερικού δικτύου συστοιχίας 50
7.2 Σύνδεση μέσω εξωτερικού παρόχου 51
7.3 Βάση δεδομένων υλοποίησης 53
7.4 Διάγραμμα του πλέγματος της εφαρμογής 59
7.5 Διεπαφή εφαρμογής 64

Κατάλογος Πινάκων

2.1 Κατανομή ευθυνών μοντέλων υπηρεσιών 5

5.1 Συγκριτικός πίνακας VPC 34

7.1 Επιτρεπόμενες προσβάσεις των μικρουπηρεσιών 48
7.2 Πίνακας αποτελεσμάτων δοκιμών εξισορρόπησης φόρτου 61

Περιεχόμενα

Κατάλογος Εικόνων **vi**

Κατάλογος Πινάκων **vii**

1 Εισαγωγή **1**
1.1 Δήλωση του προβλήματος . . . 1
1.2 Ερευνητικοί στόχοι και σκοποί διπλωματικής εργασίας . . . 1

2 Θεωρητικό υπόβαθρο **3**
2.1 Εισαγωγή στην Υπολογιστική νέφους . . . 3
2.1.1 Μοντέλα υπηρεσιών cloud . . . 4
2.1.2 Η ανάγκη για ασφάλεια και αξιοπιστία . . . 4
2.2 Cloud-Native εφαρμογές . . . 5
2.2.1 Περιέκτες . . . 6
2.2.2 Αρχιτεκτονική μικρουπηρεσιών . . . 6
2.2.3 Ενορχήστρωση περιεκτών . . . 6
2.3 Απο το DevOps στο DevSecOps . . . 7
2.4 Μηδενική εμπιστοσύνη . . . 8
2.5 Κορυφαία υφιστάμενα πλαίσια και πρότυπα ασφαλείας . . . 9
2.5.1 NIST Cybersecurity Framework . . . 9
2.5.2 NIST Zero Trust Architecture . . . 10

3 Εικονικοποίηση και περιέκτες **11**
3.1 Εικονικοποίηση . . . 11
3.1.1 Εικονικοποίηση στην υπολογιστική νέφους . . . 12
3.1.2 Ο ρόλος του επόπτη . . . 12
3.2 Η μετάβαση στην περιεκτοποίηση . . . 12
3.2.1 Ομάδες ελέγχου (cgroups) . . . 14

3.2.2 Χώροι ονομάτων (Namespaces) 15
3.3 Προγράμματα εκτέλεσης περιεκτών 16
3.3.1 Containerd 17
3.3.2 CRI-O 17
3.3.3 runc 17

4 Ενορχήστρωση περιεκτών με Kubernetes 18
4.1 Αρχιτεκτονική Kubernetes 18
4.1.1 Το εσωτερικό του Kubernetes 19
4.2 Οι πόροι του Κυβερνήτη 20
4.2.1 Pods 20
4.2.2 Deployment 21
4.2.3 Service 22
4.2.4 Ingress 23
4.3 Helm Charts 24
4.4 Δημιουργία μιας συστοιχίας 25
4.4.1 Χρήση ενός οδηγού βασισμένο σε περιέκτες 25
4.4.2 Εφαρμογή μιας CNI επέκτασης 25

5 Ανάλυση των πάροχων νέφους 27
5.1 Identity και Access Management 27
5.1.1 AWS IAM 28
5.1.2 Microsoft Entra ID 29
5.1.3 Google Cloud IAM 31
5.2 Network Security 32
5.2.1 Εικονικό ιδιωτικό νέφος 33
5.2.2 Υπηρεσίες WAF 34

6 Προκλήσεις στις νεοφυείς εφαρμογές και τρόποι αντιμετώπισης 35
6.1 Αυθεντικοποίηση και εξουσιοδότηση 35
6.1.1 SSO 36
6.1.2 SAML 36
6.1.3 Ο ρόλος του διαμεσολαβητή 38
6.1.4 OAuth 2.0 38
6.2 Πολιτικές δικτύου 40
6.3 Προκλήσεις σε επίπεδο εφαρμογών 41
6.3.1 Το πλέγμα του Istio 42

6.4 Προκλήσεις στη διαχείριση μυστικών κλειδιών 43
6.4.1 Hashicorp Vault 44
6.5 Ευπάθειες στις εικόνες και λανθασμένες ρυθμίσεις 45

7 Υλοποίηση 47
7.1 Προστασία σε επίπεδο δικτύου 47
7.2 Σύστημα αυθεντικοποίησης και εξουσιοδότησης 50
7.3 Έλεγχος πρόσβασης βάση ρόλων 52
7.4 Χρήση του Hashicorp Vault για πρόσβαση σε μυστικά κλειδιά 55
7.4.1 Δημιουργία κρυπτογραφικού κλειδιού 56
7.5 Διαχείριση των υπηρεσιών 57
7.5.1 Προσθέτοντας ασφάλεια στις υπηρεσίες 58
7.5.2 Δοκιμή load balancer 59
7.5.3 Πρόσθετες ρυθμίσεις αξιοπιστίας 62
7.6 Διεπαφή εφαρμογής 63

8 Συμπεράσματα και μελλοντική εργασία 65

Βιβλιογραφία 66

Παράρτημα Α Συμπληρωματικές τεχνολογίες που χρησιμοποιήθηκαν 68

Κεφάλαιο 1

Εισαγωγή

1.1 Δήλωση του προβλήματος

Στο ραγδαία εξελισσόμενο τοπίο της υπολογιστικής νέφους, οι οργανισμοί αξιοποιούν όλο και περισσότερο τους περιέκτες και τις μικρουπηρεσίες για τη δημιουργία και την ανάπτυξη εφαρμογών, αυτές οι τεχνολογίες προσφέρουν σημαντικά οφέλη όσον αφορά την επεκτασιμότητα, την ευελιξία και την αποδοτικότητα, εισάγουν επίσης και προκλήσεις που δεν μπορούν να αντιμετωπιστούν επαρκώς από τα παραδοσιακά μέτρα ασφαλείας. Καθώς οι εφαρμογές γίνονται πιο κατανεμημένες και δυναμικές, οι πιθανές ευπάθειες αυξάνονται, κάνοντας δύσκολη τη διασφάλιση της ομαλής λειτουργίας μιας εφαρμογής ή υπηρεσίας. Τα τελευταία χρόνια η ανάπτυξη νέων στρατηγικών που μπορούν να συμβαδίσουν με τις ταχύτητες στην ανάπτυξη εφαρμογών cloud-native είναι προτεραιότητα. Σύμφωνα με την ετήσια αναφορά του Red Hat σχετικά με την ασφάλεια στην τεχνολογία του Kubernetes [16] το 45% των συμμετεχόντων δήλωσαν ότι έχουν ενσωματώσει αυτοματοποιημένες δοκιμές ασφαλείας, συνεχή παρακολούθηση και άλλες τεχνικές με επίκεντρο την ασφάλεια, σε όλο τον κύκλο ανάπτυξης λογισμικού, με το 67% να δηλώνει ότι έχουνε υπάρχει στιγμές που έχουν καθυστερήσει ή επιβραδύνει την ανάπτυξη λόγω ενός προβλήματος ασφαλείας.

Αυτή η αναγνώριση των προκλήσεων και εμποδίων υπογραμμίζει την αναγκαιότητα ανάπτυξης προσαρμοσμένων πλαισίων ασφάλειας και αξιοπιστίας που είναι ειδικά σχεδιασμένα για την αντιμετώπιση των μοναδικών απαιτήσεων των εφαρμογών cloud-native.

1.2 Ερευνητικοί στόχοι και σκοποί διπλωματικής εργασίας

Οι στόχοι της παρούσας διπλωματικής είναι η διερεύνηση και αντιμετώπιση του τοπίου της ασφάλειας και της αξιοπιστίας στις τεχνολογίες cloud-native καθώς και η μελέτη των

αρχιτεκτονικών τους. Αρχικά, αφού μελετήσουμε εις βάθος τις τεχνολογίες της περιεκτοποίησης και ενορχήστρωσης όπως την πλατφόρμα του Kubernetes, αναλύοντας τόσο τα πλεονεκτήματα όσο και τους περιορισμούς της σχεδίασης του και των λειτουργιών του, θα εμβαθύνουμε στις λύσεις που προσφέρουν οι κορυφαίοι πάροχοι νέφους, αξιοποιώντας τις γνώσεις τους ως βάση για να διακρίνουμε τα κρίσιμα χαρακτηριστικά ασφαλείας και τις διαμορφώσεις που στηρίζουν τις νεφοϋπολογιστικες εφαρμογές. Έπειτα γίνεται μια μελέτη των βασικών προκλήσεων που αναδύονται σε υλοποιήσεις που βασίζονται σε μικρουπηρεσίες και καθορίζεται ένα πλαίσιο ενεργειών που μπορεί να μας προφυλάξει από αυτές τις προκλήσεις. Στο τελευταίο μέρος αυτής της διπλωματικής υλοποιούμε τις οδηγίες του πλαισίου προσεγγίζοντας μια αρχιτεκτονική μηδενικής εμπιστοσύνης. Θα δημιουργήσουμε διάφορες πολιτικές για την απομόνωση του εσωτερικού δικτύου καθώς θα χρησιμοποιήσουμε και εργαλεία για διαχείριση μυστικών κλειδιών, καλύτερη ασφάλεια μεταξύ των υπηρεσιών, και διαχείριση της πρόσβασης και εξουσιοδότησης.

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

2.1 Εισαγωγή στην Υπολογιστική νέφους

Στον τομέα της πληροφορικής ένας από τους σημαντικότερους λόγους του ψηφιακού μετασχηματισμού είναι η εμφάνιση και εξέλιξη της υπολογιστικής νέφους (cloud computing), η οποία προσφέρει ένα ευρύ φάσμα υπολογιστικών πόρων όπως εικονικές μηχανές (virtual machines), βάσεις δεδομένων, αποθήκευση, αναλυτικά εργαλεία και πολλές ακόμα υπηρεσίες, με τη διανομή τους να γίνεται σε ψηφιακή μορφή μέσω διαδικτύου, χωρίς έτσι να χρειάζονται φυσικές υποδομές και εγκαταστάσεις. [1]
Το βασικότερο χαρακτηριστικό της υπολογιστικής νέφους είναι η ευελιξία, επιτρέποντας σε όσους το χρησιμοποιούν να προσαρμόζουν και τους υπολογιστικού πόρους που χρειάζονται με βάση τις ανάγκες τους. Οι πόροι μπορούν να παρέχονται και να αποδεσμεύονται γρήγορα, κάνοντας την διαχείριση τους πολύ πιο εύκολη.
Πέρα από την ευελιξία που προσφέρει, σημαντικά πλεονεκτήματα της υπολογιστικής νέφους είναι:

- **Ταχύτητα**: Η ευελιξία που προσφέρει είναι σημαντικός παράγοντας που βοηθά σημαντικά τις εταιρείες να προσαρμόζονται γρήγορα στις αλλαγές του επιχειρηματικού τους περιβάλλοντος, κάτι που για μία επιχείρηση σημαίνει ότι παραμένουν ανταγωνιστικοί και μπορεί να είναι η διαφορά μεταξύ επιτυχίας και αποτυχίας.

- **Ασφάλεια και αξιοπιστία**: Κύριο μέλημα των πάροχων υπηρεσιών νέφους είναι να προσφέρουν ισχυρά χαρακτηριστικά ασφαλείας και την διασφάλιση της αξιοπιστίας και της διαθεσιμότητας των υποδομών τους.

- **Διαχείριση κόστους**: Οι χρήστες πληρώνουν μόνο για τους υπολογιστικούς πόρους που ακριβώς χρησιμοποιούν, μειώνοντας το ρίσκο αγοράς υλικό που μπορεί να μείνει αχρησιμοποίητο αλλά και γενικά το κόστος αγοράς μιας φυσικής υποδομής.

2.1.1 Μοντέλα υπηρεσιών cloud

Οι ανάγκες των χρηστών και επιχειρήσεων του νέφους ποικίλουν για αυτό και υπάρχουν διάφορα μοντέλα υπηρεσιών που το καθένα παρέχει ένα διαφορετικό επίπεδο ελέγχου και διαχείρισης. Τα τρία πιο καθιερωμένα είναι τα Υποδομή ως Υπηρεσία (Infrastructure as a Service, **IaaS)**, Πλατφόρμα ως Υπηρεσία (Platform as a Service, **PaaS)**, Λογισμικό ως Υπηρεσία (Software as a Service, **SaaS)**.

Infrastructure as a Service

Το IaaS είναι ένα ευέλικτο και κλιμακούμενο μοντέλο που παρέχει μια ολοκληρωμένη υποδομή που περιλαμβάνει υλικό, δίκτυα, αποθήκευση και εικονικές μηχανές (virtual machine). Οι πόροι IaaS είναι πλήρως εικονικοποιημένοι που όπως προαναφερθήκαμε προσφέρουν επεκτασιμότητα και προσαρμογή. Οι χρήστες είναι υπεύθυνοι το λειτουργικό σύστημα και τις εφαρμογές που τρέχουν στο περιβάλλον.

Platform as a Service

Το PaaS παρέχει ένα έτοιμο προς χρήση περιβάλλον που με βάση τις επιλογές του χρήστη, διανέμεται με προεγκατεστημένο λειτουργικό σύστημα, εργαλεία, βιβλιοθήκες, και γλώσσες προγραμματισμού για την εγκατάσταση και εκτέλεση εφαρμογών, έτσι ο χρήστης δεν χρειάζεται να ασχοληθεί καθόλου με την ρύθμιση των υποδομών καθώς το περιβάλλον είναι έτοιμο για χρήση.

Software as a Service

Το SaaS παρέχει ένα άμεσα διαθέσιμο προϊόν για τους χρήστες, συνήθως σε συνδρομητική βάση, χωρίς ο χρήστης να πρέπει να εγκαταστήσει τοπικά τίποτα. Τα προϊόντα SaaS είναι προσβάσιμα από φυλλομετρητές παρέχοντας ευκολία στη χρήση και επιτρέποντας στους χρήστες να επικεντρωθούν στην αξιοποίηση τους.

2.1.2 Η ανάγκη για ασφάλεια και αξιοπιστία

Οι πάροχοι νέφους υπηρεσιών επενδύουν πάρα πολύ στην ασφάλεια και στην διαθεσιμότητα των προϊόντων και υπηρεσιών που προσφέρουν στους πελάτες τους. Παρόλα αυτά οποιοδήποτε μοντέλο υπηρεσιών και να επιλέξει μια επιχείρηση, τις αναλογούν και ορισμένες

ευθύνες για την διατήρηση αυτών των αρχών στις υπηρεσίες που προσφέρει και κάνει χρήση. Στο παρακάτω πίνακα μπορούμε να δούμε τις ευθύνες που έχει ο **πελάτης (Customer)** και ποιες έχει ο **πάροχος (Provider)**.

Πίνακας 2.1 Κατανομή ευθυνών μοντέλων υπηρεσιών

	IaaS	**PaaS**	**SaaS**
Application Configuration	C	C	C
Identity & access control	C	CP	CP
Application data storage	C	CP	P
Application	C	C	P
Operating system	C	P	P
Network flow controls	CP	P	P
Host infrastructure	P	P	P
Physical security	P	P	P

Σημείωση: ”C” για τον πελάτη and ”P” για τον πάροχο. Πηγή: [https://www.ncsc.gov.uk/collection/cloud/understanding-cloud-services/cloud-security-shared-responsibility-model]

Η ανάγκη για ασφάλεια στις cloud υπηρεσίες, συγκεκριμένα για το IaaS, είναι πολύ μεγάλη διότι οι επιχειρήσεις αξιοποιούν την υποδομή cloud για να φιλοξενήσουν τις εφαρμογές και τα δεδομένα τους, εφαρμογές που οι ίδιες επιχειρήσεις αναπτύσσουν επιλέγοντας λειτουργικό σύστημα, βιβλιοθήκες και γλώσσες προγραμματισμού, συνεπώς αυτό επιφέρει και τη διαχείριση της ασφάλειας αυτών. Η φύση των υπηρεσιών cloud εισάγει κινδύνους και πιθανές ευπάθειες για τις οποίες οι επιχειρήσεις πρέπει να είναι ιδιαίτερα προσεχτικοί, οπότε αυτό απαιτεί και την υιοθέτηση ολοκληρωμένων μέτρων ασφαλείας.

2.2 Cloud-Native εφαρμογές

Στο επίκεντρο αυτής της διπλωματικής εργασίας βρίσκεται η έννοια του cloud native, η οποία προσεγγίζει τη σχεδίαση των εφαρμογών ώστε να αξιοποιούν στο μέγιστο την ευελιξία και επεκτασιμότητα της υπολογιστικής νέφους [5], παρακάτω θα αναλύσουμε κάποια από τα βασικά στοιχεία που μας βοηθάνε να πετύχουμε κάτι τέτοιο.

2.2.1 Περιέκτες

Αναφερθήκαμε στο IaaS το οποίο είναι και το κύριο μοντέλο υπηρεσιών με το οποίο θα ασχοληθούμε, ότι είναι μια υποδομή έτοιμη να εγκαταστήσουμε σε κάποια εικονική μηχανή (VM) το λογισμικό που χρειαζόμαστε για εκτέλεση των εφαρμογών μας, το λογισμικό θα χρειαστεί και τα απαραίτητα εργαλεία και βιβλιοθήκες για να "τρέξει", το να τα εγκαθιστούμε ξεχωριστά σε κάθε εικονική μηχανή που κάνει χρήση η υποδομή μας θα ήταν πολύ δύσκολο και ως ένα βαθμό αδύνατο με τις σημερινές απαιτήσεις που έχουμε από τις εφαρμογές που αναπτύσσουμε. Οι **περιέκτες (containers)** δίνουν τη λύση προσφέροντας ένα απομονωμένο περιβάλλον με το να μοιράζονται μόνο τον πυρήνα του λογισμικού συστήματος που φιλοξενούνται (host) έχοντας το όλα τα στοιχεία ενός λειτουργικού συστήματος όπως χώρους ονομάτων (namespaces), ομάδες ελέγχου (cgroups) , αυτό ονομάζεται **περιεκτοποίηση (containerization)** και θα το αναλύσουμε σε βάθος στο επόμενο κεφάλαιο. Καθοριστικό ρόλο στην ευρεία διάδοση των περιεκτών έχει το Docker [3] το οποίο απλοποιεί την ενσωμάτωση των εφαρμογών μαζί με τις γλώσσες προγραμματισμού και βιβλιοθήκες που χρειάζεται για να "τρέξει" μια εφαρμογή. Το Docker δίνει την δυνατότητα με την χρήση ενός εγγράφου **Dockerfile**, ο χρήστης να "συναρμολογεί" μια εικόνα (image) με ό,τι χρειάζεται και να μπορεί να τη χρησιμοποιήσει για να φτιάξει όσα containers θέλει.

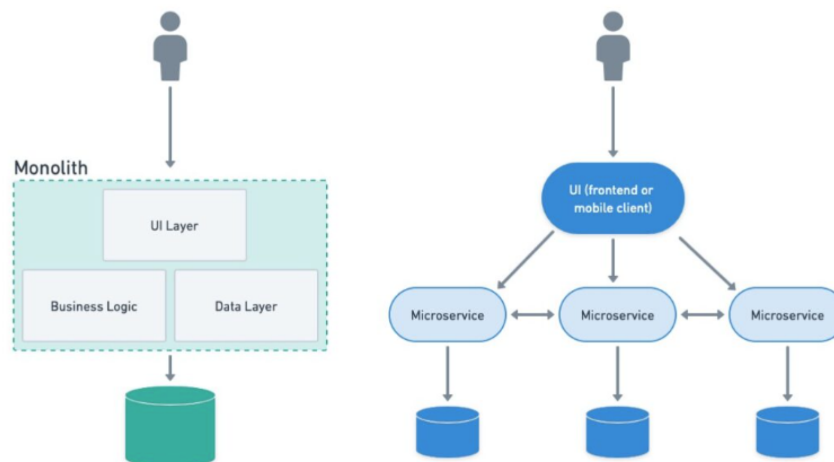
2.2.2 Αρχιτεκτονική μικρουπηρεσιών

Είναι σημαντικό να αναφερθούμε στην αρχιτεκτονική **μικρουπηρεσιών (microservices)**, η οποία αντιπροσωπεύει στο μέγιστο την cloud-native σχεδίαση διαμορφώνοντας ένα σύστημα απο μία συλλογή πολλών υποσυστημάτων που το καθένα έχει συγκεκριμένη λειτουργία. Κάθε υποσύστημα γίνεται εγκατάσταση σε δικό του container προσφέροντας απομόνωση από τα υπόλοιπα, έτσι πετυχαίνουμε το καθένα να είναι ανεξάρτητο από το άλλο κάνοντας την ανάπτυξη, συντήρηση και επέκταση των υποσυστημάτων πολύ πιο εύκολη. Αυτή η προσέγγιση ονομάζεται **deploy independently principle (DIP)**[5]
Στη παρακάτω εικόνα βλέπουμε και τις διαφορές που έχει με την παραδοσιακή μονολιθική αρχιτεκτονική όπου όλες οι λειτουργίες της εφαρμογής είναι στο ίδιο σύστημα.

2.2.3 Ενορχήστρωση περιεκτών

Έχοντας επισημάνει την επεκτασιμότητα και την ευελιξία που προσφέρει το cloud computing μαζί με τη χρήση των περιεκτών, δεν μπορούμε να μην αναφερθούμε στον κρίσιμο ρόλο της

[1] Πηγή: https://semaphoreci.com/blog/microservice-architecture



Σχήμα 2.1 Μονολιθική αρχιτεκτονική και αρχιτεκτονική μικρουπηρεσιών[1]

ενορχήστρωσης (orchestration). Όσο μια εφαρμογή επεκτείνεται χρειάζεται όλο και περισσότερα containers, έτσι δημιουργείται η ανάγκη για διαχείριση πολυάριθμων containers. Τα εργαλεία ενορχήστρωσης το διευκολύνουν αυτό, αυτοματοποιώντας την ανάπτυξη, την κλιμάκωση και τη λειτουργία των εφαρμογών που περιέχουν containers.

2.3 Απο το DevOps στο DevSecOps

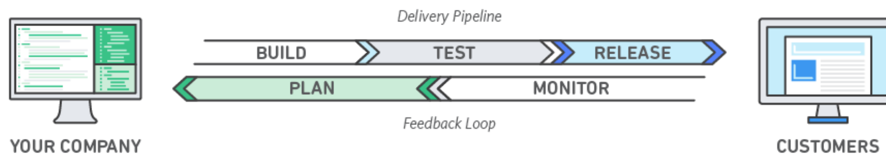
Ιστορική αναδρομή

Η ιστορία του DevOps ξεκίνησε το 2007 με τον Patrick Debois, ο οποίος αναγνώρισε την ανάγκη για καλύτερη συνεργασία μεταξύ των ομάδων ανάπτυξης (Dev) και λειτουργίας (Ops). Στα βασικά ορόσημα περιλαμβάνονται οι πρώτες DevOpsDays το 2009, η δημοσίευση του ”Continuous Delivery” το 2010, του ”The Phoenix Project” το 2013 και του ”The DevOps Handbook” το 2016. [7]

Το DevOps έχει να κάνει με τη συνεργασία προγραμματιστών (Dev) και με αυτού που το διατηρούν σε λειτουργία (Ops) ώστε να δημιουργούν λογισμικό γρήγορα και με ασφάλεια. Πρόκειται για νοοτροπία που δίνει έμφαση στη γρήγορη και συχνή αποστολή μικρών τμημάτων ενός συνόλου λογισμικού για εγκατάσταση και δοκιμή, αυτή η διαδικασία είναι πλήρως αυτοματοποιημένη και με τη συνεργασία, την συζήτηση και την συνεχή βελτίωση προς τις ανάγκες των χρηστών το τμήμα είναι έτοιμο για να περάσει στην παραγωγή. [4]

Αυτό συμβαίνει με την διαδικασία **συνεχούς ολοκλήρωσης/συνεχούς παράδοσης (CI/CD)** που είναι και το σύνολο πρακτικών του DevOps.

[2]Πηγή: https://https://aws.amazon.com/devops/what-is-devops/



Σχήμα 2.2 DevOps pipeline[2]

Έτσι, το DevOps εξελίχθηκε σε αυτό που είναι σήμερα μέσα από την ευελιξία που προσφέρει η υπολογιστική νέφους. Ωστόσο, αυτή η ευελιξία και η ταχύτητα πάνε μαζί με τις δικές τους προκλήσεις, ιδίως όσον αφορά την ασφάλεια. Αρκετές φορές οι πρακτικές και οι έλεγχοι που εκτελούμε για την ασφάλεια δεν μπορούν να "προλάβουν" την γρήγορη ανάπτυξη που προσφέρει το DevOps, άλλες φορές παραλείπονται, αυτό έχει ως συνέπεια, να δημιουργεί έδαφος για μη εξουσιοδοτημένη πρόσβαση, διακοπή της διαθεσιμότητας από επιθέσεις και άλλες απειλές.

Είναι εμφανής η ανάγκη προσαρμογής και ενσωμάτωσης των πρακτικών και ελέγχων ασφάλειας στα pipeline του DevOps, διασφαλίζοντας έτσι την ακεραιότητα και την ασφάλεια των εφαρμογών, μειώνοντας τον κίνδυνο παραβίασης της ασφάλειας δίνοντας μας την δυνατότητα να προλάβουμε έγκαιρα διάφορες επιθέσεις και να μην μειώνουμε την ταχύτητα ανάπτυξης και παράδοσης του λογισμικού, αυτό επιτυγχάνεται με το **DevSecOps**.

2.4 Μηδενική εμπιστοσύνη

Νέοι τρόποι αντιμετώπισης των κινδύνων αναδυθήκαν έτσι ώστε να αντιμετωπιστούν οι κίνδυνοι, ένας από αυτούς είναι η στρατηγική της **μηδενικής εμπιστοσύνης (Zero Trust)** που προσεγγίζει την ιδέα ότι στο δίκτυο υπάρχουν συνέχεια απειλές. Η μηδενική εμπιστοσύνη υποθέτει ότι καμία οντότητα δεν πρέπει να είναι αυτόματα έμπιστη και πρέπει να επαληθεύεται συνεχώς για να έχει πρόσβαση σε πόρους. [6]

Με βάση τις κατευθυντήριες γραμμές της αρχιτεκτονικής μηδενικής εμπιστοσύνης που περιγράφονται λεπτομερώς από την δημοσίευση του NIST, "Zero Trust Architecture", [6] οι βασικές αρχές που καθορίζουν προσέγγιση της μηδενικής εμπιστοσύνης είναι :

- **Όλα τα δεδομένα και υπηρεσίες θεωρούνται πόροι**: Δεν πρέπει να γίνεται καμία εξαίρεση, όλα τα δεδομένα είναι σημαντικά
- **Κάθε σύνδεση πρέπει να έχει ταυτοποιηθεί**: Η πρόσβαση πρέπει να δίνεται πάντα με την ίδια ασφάλεια ανεξαρτήτου αν μια συσκευή είναι σε ένα ασφαλής δίκτυο

- **Κάθε προσπάθεια αίτησης πρόσβαση είναι διαφορετική**: Η αίτηση πρόσβαση θα πρέπει να εξετάζεται κάθε φορά και με βάση τους πόρους που αιτείται

- **Η δυναμική πολιτική καθορίζει την πρόσβαση**: Η αίτηση πρόσβαση θα πρέπει να εξετάζεται κάθε φορά και με βάση τους πόρους που αιτείται

- **Συνεχής παρακολούθηση της ασφάλειας**: Η ασφάλεια των προστατευμένων πόρων θα πρέπει συνεχώς να αξιολογείται και όπου γίνεται να βελτιώνεται

- **Επιβολή δυναμικής αυθεντικοποίησης και εξουσιοδότησης**: Η πρόσβαση πρέπει να επαναξιολογείται

- **Αξιοποίηση δεδομένων**: Συλλογή των δεδομένων της πρόσβασης στου πόρους και του δικτύου και βελτίωση ασφάλειας βάση αυτών

2.5 Κορυφαία υφιστάμενα πλαίσια και πρότυπα ασφαλείας

Σε αυτό το υποκεφάλαιο, ρίχνουμε μια ματιά στα γνωστά πλαίσια κυβερνοασφάλειας, θέτοντας τις βάσεις για τη μετέπειτα έρευνά μας. Η κατανόηση αυτών των πλαισίων, χρησιμεύει ως σημείο εκκίνησης για να εμβαθύνουμε στο τρόπο αποτελεσματικής εφαρμογής αυτών των πρακτικών και για των απειλών που αναφέρονται.

2.5.1 NIST Cybersecurity Framework

Το NIST Cybersecurity Framework [2] προσφέρει ένα σχέδιο για την κατανόηση, τη διαχείριση και την διαχείριση των κινδύνων κυβερνοασφάλειας με τρόπο δομημένο. Αυτό το πλαίσιο έχει σχεδιαστεί ώστε να είναι εφαρμόσιμο σε οργανισμούς όλων των μεγεθών καθιστώντας το ένα ευέλικτο εργαλείο για την ενίσχυση της κατάστασης της κυβερνοασφάλειας. Το NIST Framework βασίζεται σε πέντε βασικές λειτουργίες, οι οποίες παρέχουν μια ταξινόμηση των δραστηριοτήτων και των αποτελεσμάτων της ασφάλειας στον κυβερνοχώρο που είναι ιδανική για μια ολοκληρωμένη προσέγγιση της διαχείρισης κινδύνων:

- **Εντοπισμός**: Ανάπτυξη κατανόησης της διαχείρισης των κινδύνων κυβερνοασφάλειας για τα συστήματα, τα περιουσιακά στοιχεία, τα δεδομένα και τις δυνατότητες.

- **Προστασία**: Ανάπτυξη κατάλληλων δικλείδων ασφαλείας για τη προστασία της παροχής υπηρεσιών.

- **Ανίχνευση**: Εφαρμογή κατάλληλων δραστηριοτήτων για τον εντοπισμό της εκδήλωσης ενός συμβάντος κυβερνοασφάλειας.

- **Ανταπόκριση**: Ανάληψη δράσης σχετικά με εντοπισμένο περιστατικό κυβερνοασφάλειας.

- **Ανάκτηση**: Ανάπτυξη λειτουργιών για την αποκατάσταση των δυνατοτήτων ή υπηρεσιών που έχουν μειωθεί λόγω κάποιου περιστατικού κυβερνοασφάλειας.

2.5.2 NIST Zero Trust Architecture

Η αρχιτεκτονική μηδενικής εμπιστοσύνης (Zero Trust Architecture - ZTA) του NIST διαφοροποιείται στη στρατηγική κυβερνοασφάλειας, από τις παραδοσιακές, άμυνες που είναι περιμετρικές σε μία προσέγγιση που επικεντρώνεται στην παραδοχή ότι η εμπιστοσύνη δεν παρέχεται ποτέ εύκολα. Σε ένα ZTA, κάθε αίτημα πρόσβασης, ανεξάρτητα από την προέλευσή του, πιστοποιείται, εξουσιοδοτείται και κρυπτογραφείται διεξοδικά πριν από τη δημιουργία οποιασδήποτε επικοινωνίας με τους πόρους ενός οργανισμού.

Περιγράφει ένα πλαίσιο κυβερνοασφάλειας που μετατοπίζει την εστίαση από τις παραδοσιακές άμυνες που βασίζονται στην περίμετρο του δικτύου σε ένα μοντέλο όπου η εμπιστοσύνη δεν είναι ποτέ αυτονόητη και πρέπει να επαληθεύεται συνεχώς. Οι αρχές της μηδενικής εμπιστοσύνης απαιτούν αυστηρό έλεγχο ταυτότητας και εξουσιοδότηση για κάθε αίτημα πρόσβασης, ανεξάρτητα από την προέλευση ή τη θέση του δικτύου. Αυτή η προσέγγιση απαιτεί μηχανισμούς ασφαλείας με επίγνωση του πλαισίου σε πραγματικό χρόνο για τη δυναμική αξιολόγηση και πιστοποίηση των επιπέδων εμπιστοσύνης των χρηστών και των συσκευών πριν από τη χορήγηση πρόσβασης σε πόρους. Με την εφαρμογή της αρχιτεκτονικής μηδενικής εμπιστοσύνης, οι οργανισμοί μπορούν να βελτιώσουν σημαντικά τη στάση ασφαλείας τους, να μειώσουν την επιφάνεια επιθέσεων και να προστατευτούν καλύτερα από εσωτερικές και εξωτερικές απειλές. [6]

Κεφάλαιο 3

Εικονικοποίηση και περιέκτες

Σε αυτό το κεφάλαιο, θα εξερευνήσουμε τις τεχνολογίες της εικονικοποίησης και των περιεκτών, οι οποίες έχουν και καθοριστική σημασία για την εξέλιξη των εφαρμογών cloud-native και η κατανόηση τους είναι σημαντική καθώς επηρεάζουν άμεσα τον τρόπο με τον οποίο σχεδιάζονται και λειτουργούν τα εργαλεία ενορχήστρωσης όπως το Kubernetes [11] που θα αναλύσουμε στο επόμενο κεφάλαιο.

3.1 Εικονικοποίηση

Η εικονικοποίηση στον απόλυτο ορισμό του είναι μια τεχνολογία που δημιουργεί εικονικές αναπαραστάσεις φυσικών πόρων, όπως διακομιστές, αποθηκευτικούς χώρους, δίκτυα και άλλες υπολογιστικές υποδομές. Η διαδικασία αυτή επιτρέπει τη δημιουργία **εικονικών μηχανών (VM)** που προσομοιώνουν τη λειτουργία ενός φυσικού πόρου, επιτρέποντας την ταυτόχρονη εκτέλεση πολλαπλών λειτουργικών συστημάτων και εφαρμογών σε μία μόνο φυσική μονάδα.

Μία από τις βασικές δυνατότητες της εικονικοποίησης είναι το να παρέχει απομόνωση μεταξύ των εικονικών περιβαλλόντων που έχει δημιουργήσει. Παρά την απομόνωση αυτή, είναι πολύ σημαντικό να διασφαλίζει ότι η απόδοση μιας εικονικής μηχανής, ανταποκρίνεται σε μεγάλο βαθμό στην απόδοση που θα είχαμε με μία αντίστοιχη φυσική μονάδα [15]. Αυτός είναι και ο στόχος, να ελαχιστοποιηθεί η επιβάρυνση της απόδοσης που εισάγεται από την εικονικοποίηση. Αυτή η ισορροπία μεταξύ της απομόνωσης και της αποδοτικότητας των επιδόσεων έχει μεγάλη σημασία για τη μεγιστοποίηση της χρήσης των πόρων και είναι και ένας από τους λόγους που πλέον η πλειοψηφία της υποδομής των οργανισμών είναι βασισμένη σε εικονικές μηχανές.

3.1.1 Εικονικοποίηση στην υπολογιστική νέφους

Στο παρελθόν πριν την εικονικοποίηση, οι διαχειριστές αφιέρωναν πολύ χρόνο σε εργασίες συντήρησης και αντιμετώπισης διαφόρων προβλημάτων μιας φυσική υποδομής σε κάθε οργανισμό, αφήνοντας σε δεύτερη μοίρα δραστηριότητες όπως η καινοτομία [15]. Ερχόμενοι στο σήμερα, η εικονικοποίηση έχει λύσει τέτοια προβλήματα δίνοντας χώρο σε μια επιχείρηση να πάρει στρατηγικές αποφάσεις για την ανάπτυξή της και να βελτιώσει τις παροχές στις υπηρεσίες της. Επιπλέον, τα χαρακτηριστικά των εικονικών μηχανών , όπως η ενισχυμένη διαθεσιμότητα και η επεκτασιμότητα, έχουν θέσει τις βάσεις για τη μετάβαση στη υπολογιστική νέφους διασφαλίζοντας ότι οι επιχειρήσεις μπορούν να βασίζονται στο νέφος για τη φιλοξενία των κρίσιμων εφαρμογών και υπηρεσιών τους. Η δυνατότητα να μπορούν να δεσμεύουν και αποδεσμεύουν πόρους, ευθυγραμμίζεται με τις απαιτήσεις των σύγχρονων επιχειρηματικών λειτουργιών, κάνοντας την εικονικοποίηση απαραίτητη τεχνολογία στην εξέλιξη και την υιοθέτηση λύσεων υπολογιστικού νέφους.

3.1.2 Ο ρόλος του επόπτη

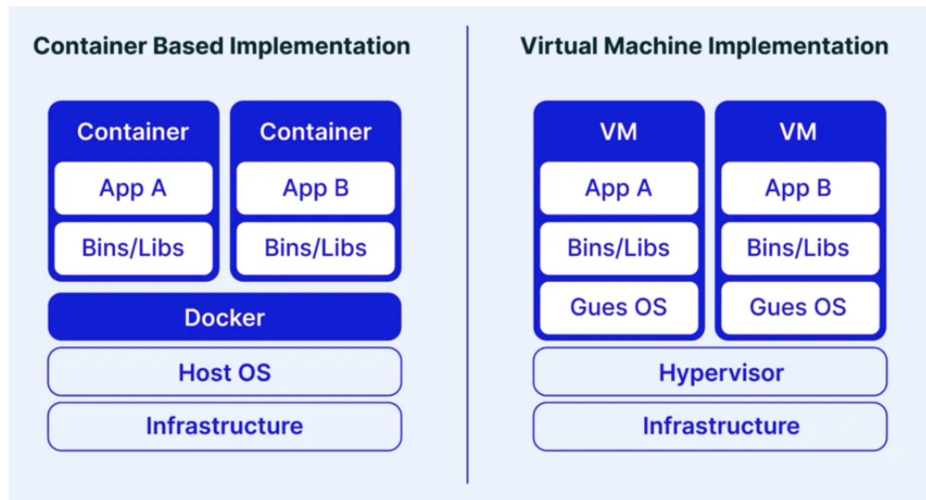
Ο **επόπτης (hypervisor)**, το βασικότερο στοιχείο της τεχνολογίας της εικονικοποίησης ο οποίος και δρα σαν διαμεσολαβητής μεταξύ του φυσικού υλικού και των εικονικών μηχανών. Ορισμένα από τα *καθήκοντά* του είναι:

- Η διαχείριση και κατανομή των πόρων στα χωρισμένα εικονικά περιβάλλοντα μιας φυσικής μονάδας στην οποία τρέχει.
- Δημιουργεί τα εικονικά περιβάλλοντα και προσομοιώνει το υλικό

3.2 Η μετάβαση στην περιεκτοποίηση

Όσο ισχυρή και επαναστατική και αν ήταν η εικονικοποίηση, έχει τους περιορισμούς της, ειδικά όταν πρόκειται για τις ανάγκες της ευελιξίας και διάθεσης εφαρμογών. Κάποιοι από τους περιορισμούς της εικονικοποίησης είναι:

- Λόγο ότι κάθε εικονική μηχανή διαθέτει ένα ολόκληρο λειτουργικό σύστημα, οδηγεί σε επιβάρυνση των πόρων και κατανάλωσης περισσότερου αποθηκευτικού χώρου και μνήμης
- Φορτώνοντας έτσι ολόκληρο το λειτουργικό σύστημα, η εκκίνηση καινούργιων εικονικών μηχανών είναι αργή, που σημαίνει ότι δεν είναι ιδανικές για διαδικασίες ανάπτυξης που χρειάζονται ταχύτητα και κλιμάκωση.



Σχήμα 3.1 Περιέκτες και εικονικοποίηση[1]

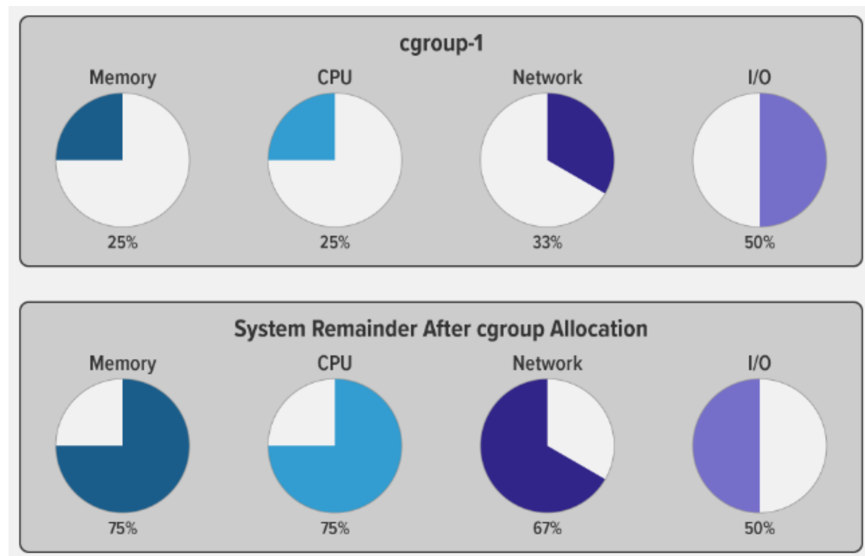
- Δεδομένο ότι η απομόνωση γίνεται σε φυσικό επίπεδο, είναι παραπάνω από όσο χρειάζεται η σύνηθες χρήση από εφαρμογές.

Συνεπώς είναι ξεκάθαρο ότι η εικονικοποίηση δεν θα μπορούσε να είναι η λύση για τα σύγχρονα προβλήματα που απαιτούν οι εφαρμογές να ανανεώνονται συνεχώς με νέες εκδόσεις και να διαμορφώνονται ανάλογα με τις ανάγκες που πρέπει να καλύψουν. Αυτή η αναγνώριση οδήγησε στην εμφάνιση της **περιεκτοποίησης (containerization)**, μιας πιο ελαφριάς και αποτελεσματικής μεθόδου εικονικοποίησης εφαρμογών.

Οι περιέκτες έχουν σχεδιαστεί για να είναι εφήμεροι και δυναμικοί, με δυνατότητες γρήγορης ανάπτυξης και κλιμάκωσης, αυτός είναι και λόγος που η περιεκτοποίηση δίνει τη λύση στου παραπάνω περιορισμούς. Συγκεκριμένα, προσφέρει τα εξής:

- **Ταχύτητα**: Γρήγοροι χρόνοι εκκίνησης και πιο αποδοτική χρήση των πόρων διότι οι περιέκτες μοιράζονται το πυρήνα του λειτουργικού συστήματος που τους φιλοξενεί (host).
- **Απομόνωση και φορητότητα**: Μοιράζονται το πυρήνα του λειτουργικού συστήματος αλλά οι εφαρμογές στους περιέκτες είναι τελείως απομονωμένες
- **Αφαίρεση και αποδοτικότητα**: Η λογική των εφαρμογών εμπεριέχεται μόνο μέσα στους περιέκτες και είναι σχεδιασμένοι να υπάρχουν μέχρι να ολοκληρώσουν την εργασία τους.

3.2.1 Ομάδες ελέγχου (cgroups)

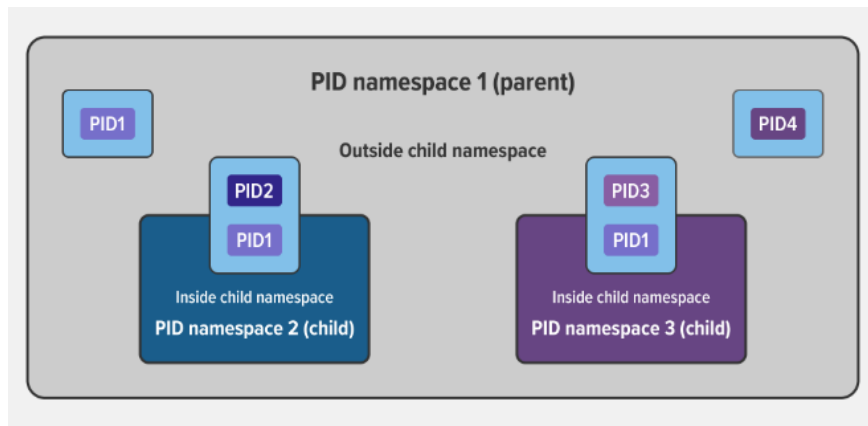


Σχήμα 3.2 Ομάδες ελέγχου[2]

Οι ομάδες ελέγχου, κοινώς γνωστές ως cgroups, είναι ένα χαρακτηριστικό του πυρήνα του λειτουργικού συστήματος Linux που επιτρέπει στο διαχειριστή του συστήματος να κατανέμει, να διαχειρίζεται και να απομονώνει τη χρήση των πόρων των ομάδων διεργασιών. Οι ομάδες cgroups παίζουν κρίσιμο ρόλο στη διαχείριση των περιεκτών, διασφαλίζοντας ότι κάθε περιέκτης θα χρησιμοποιεί του πόρους που του αναλογούν , εμποδίζοντας έτσι οποιοδήποτε να τους μονοπωλεί και να δημιουργεί πρόβλημα στου υπόλοιπους.

[2]Πηγή: https://www.nginx.com/blog/what-are-namespaces-cgroups-how-do-they-work/

3.2.2 Χώροι ονομάτων (Namespaces)



Σχήμα 3.3 Χώρος ονομάτων[3]

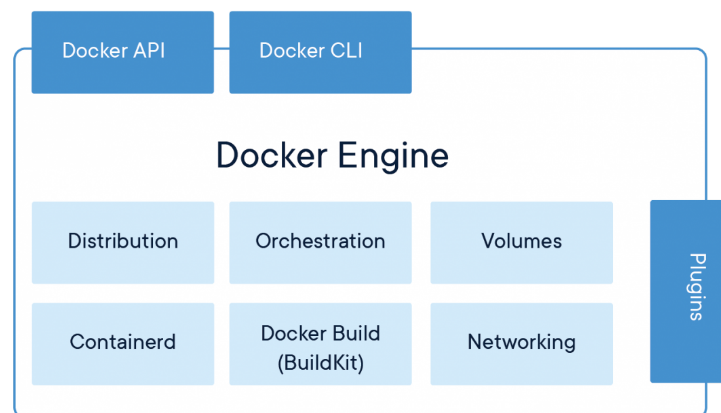
Οι χώροι ονομάτων (namespaces) του Linux είναι ένα χαρακτηριστικό που το Docker και άλλες τεχνολογίες περιεκτών αξιοποιούν για να παρέχουν απομονωμένα περιβάλλοντα για την εκτέλεση εφαρμογών. Στην ουσία αυτό που πετυχαίνουν είναι κάθε περιέκτης να φαίνεται ότι έχει το δικό του κεντρικό λειτουργικό σύστημα. Οι τύποι των χώρων ονομάτων που χρησιμοποιεί η περιεκτοποίηση είναι:

- **Αναγνωριστικό διεργασίας (PID)**: Απαραίτητο για την απομόνωση διεργασιών μεταξύ περιεκτών, επιτρέπει σε κάθε χώρο ονομάτων να αναθέσει τα ίδια αναγνωριστικά στις διεργασίες, δίνοντας την ψευδαίσθηση ότι είναι και οι μοναδικες διεργασίες στο λειτουργικό σύστημα.
- **Δικτύου**: Απομονώνει ότι έχει να κάνει με την δικτύωση του περιέκτη, όπως κανόνες τοίχους ασφαλείας (firewall) και κανόνες δρομολόγησης.
- **Χώρου αποθήκευσης**: Απομονώνει το σύστημα αρχείων, επιτρέποντας σε κάθε περιέκτη να έχει το δικό του, έτσι δεν μπορεί να δει περιέκτης τα αρχεία του άλλου.
- **Χρήστη**: Επιτρέπει στο χρήστη να έχει διαφορετικά δικαιώματα (permissions) σε κάθε διαφορετικό χώρο ονομάτων δηλαδή σε κάθε διαφορετικό περιέκτη.
- **Επικοινωνίας (IPC)**: Διαχωρίζει τους πόρους επικοινωνίας μεταξύ των διεργασιών.
- **UNIX Time-Sharing (UTS)**: Επιτρέπει σε κάθε περιέκτη να έχει τα δικά του ονόματα τομέα (domain), ξεχωριστά από το υπόλοιπο σύστημα.

[3]Πηγή: https://www.nginx.com/blog/what-are-namespaces-cgroups-how-do-they-work/

3.3 Προγράμματα εκτέλεσης περιεκτών

Παρακάτω βλέπουμε και την αρχιτεκτονική του Docker Engine. Ένα από τα πιο σημαντικά στοιχεία του είναι το containerd, το οποίο είναι ένα υψηλού επιπέδου **πρόγραμμα εκτέλεσης περιεκτών (container runtime)** που μπορεί να χρησιμοποιηθεί και ανεξάρτητα, χωρίς να χρησιμοποιήσουμε καθόλου το Docker και ό,τι αυτό μας προσφέρει. Τα προγράμματα εκτέλεσης περιεκτών είναι κυρίως υπεύθυνα για την εκτέλεση των περιεκτών και τη δημιουργία του περιβάλλοντος και είναι και αυτά που χειρίζονται τη διαμόρφωση των χώρων ονομάτων και των ομάδων ελέγχου που αναφερθήκαμε προηγουμένως. Τα χαμηλού επιπέδου προγράμματα εκτέλεσης διαχειρίζονται άμεσα τον κύκλο ζωής του περιέκτη και τις αλληλεπιδράσεις του συστήματος ενώ τα υψηλότερου επιπέδου προσφέρουν μια πιο φιλική προς το χρήστη διεπαφή για την αλληλεπίδραση των περιεκτών, που περιλαμβάνει κυρίως τη διαχείριση εικόνων. Εμβαθύνοντας περισσότερο στα container runtimes θα αναφερθούμε και στα επικρατέστερα.



Σχήμα 3.4 Αρχιτεκτονική Docker[4]

[4]Πηγή: https://www.docker.com/products/container-runtime/

3.3.1 Containerd

Όπως είδαμε το Docker βασίζεται στο containerd, το οποίο μπορεί και διαχειρίζεται τον κύκλο ζωής των περιεκτών αλλά παρέχει και λειτουργικότητα στη διαχείριση των εικόνων, στην δικτύωση και την αποθήκευση στους περιέκτες. Είναι ιδανικό για χρήσεις που μας νοιάζει να εξυπηρετήσουμε τον περιβάλλον του Kubernetes αλλά τόσο και αυτόνομα χωρίς την χρήση ενορχήστρωσης. Παρόλα αυτά δεν προσφέρει τα εργαλεία της δημιουργίας εικόνων και αρκετές φιλικά προς το χρήστη διεπαφές που προσφέρει το σύνολο του Docker ή κάποιο άλλο εργαλείο συνδυαστικά με το containerd, όπως το Buildah.

3.3.2 CRI-O

Μια εναλλακτική του containerd είναι το CRI-O, με την διαφορά ότι έχει σχεδιαστεί κυρίως για να προσφέρει μόνο τις λειτουργίες και τα χαρακτηριστικά που χρειάζεται το Kubernetes, συνεπώς είναι και πιο ελαφρύ δίχως έξτρα επιβαρύνσεις. Κάτι που σημαίνει ότι και αυτό δεν προσφέρει πολλά από τα εργαλεία που κάνουν την ζωή του προγραμματιστή πιο εύκολη όσον αφορά στο να δοκιμάζει τοπικά τους περιέκτες που σχεδιάζει.
Αυτό όμως μπορεί να βρεθεί στο εργαλείο **Podman** που είναι συμβατό με το CRI-O και προσφέρει σχεδόν τις ίδιες λειτουργίες που προσφέρει και το Docker για άμεση δοκιμή και εκτέλεση των περιεκτών.

3.3.3 runc

Και τα δύο όμως αυτά runtimes χρησιμοποιούν σε χαμηλότερο επίπεδο το **runc**, το οποίο είναι αυτό που στην ουσία διαχειρίζεται τον κύκλο ζωής τον περιεκτών και την απομόνωση αυτών με την χρήση των namespaces και cgroups του λειτουργικού συστήματος. Είναι λοιπό ένα χαμηλού επιπέδου πρόγραμμα εκτέλεσης περιεκτών το οποίο τηρεί πλήρως τις οδηγίες του οργανισμού Open Container Initiative (OCI), οι οποίες εξασφαλίζουν την συμβατότητα και έχουν να κάνουν με τη μορφή των εικόνων, την διαδικασία κατασκευής τους, την ανάκτησή τους και την εκτέλεση τους σε περιέκτες.

Κεφάλαιο 4

Ενορχήστρωση περιεκτών με Kubernetes

Έχοντας δει το πως δουλεύουν οι περιέκτες και έχοντας αναφερθεί προηγούμενος στη σημασία της ενορχήστρωσης στην υπολογιστική νέφους, σε αυτό το κεφάλαιο θα ασχοληθούμε με την ενορχήστρωση και το εργαλείο Kubernetes. Το Kubernetes ή αλλιώς *Κυβερνήτης* στα ελληνικά είναι η τεχνολογία που έχει επικρατήσει στην ενορχήστρωση των containers, η οποία κάνει τα πράγματα απλά όσον αφορά το πως οι εφαρμογές αναπτύσσονται και διαχειρίζονται στα περιβάλλοντα.

4.1 Αρχιτεκτονική Kubernetes

Ιστορική αναδρομή

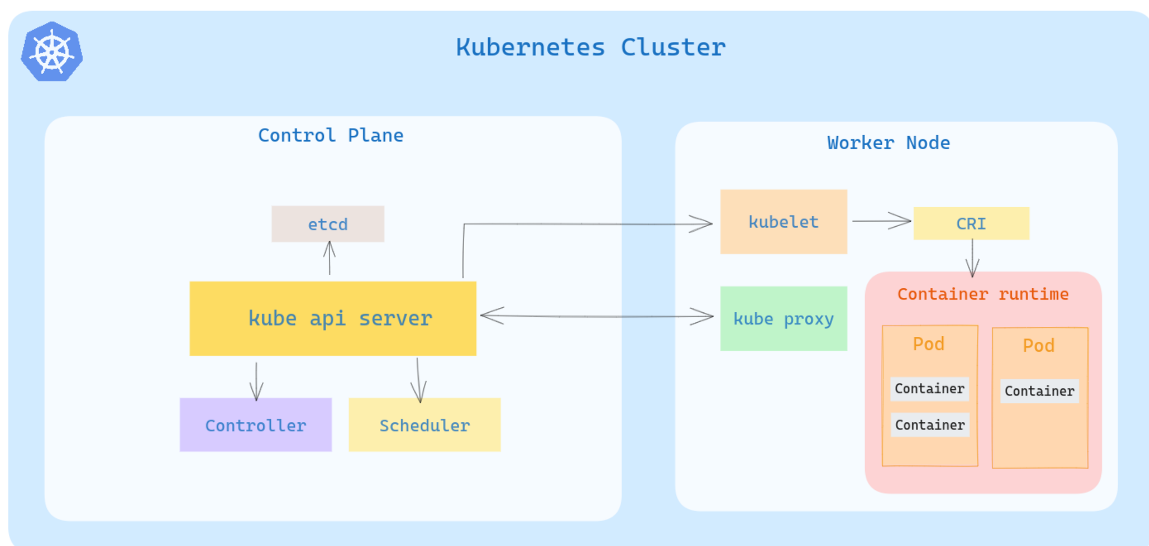
Παρότι η απομόνωση διεργασιών που αργότερα βασίστηκε και η τεχνολογία των περιεκτών ξεκινάει το 1979, το Kubernetes ξεκίνησε το 2014 να διατίθεται στο ευρύ κοινό έπειτα από την δημιουργία του από προγραμματιστές της Google που είχαν δουλέψει για τον προκάτοχο του (Borg), και θέλοντας να φτιάξουν κάτι πιο εύχρηστο και ανοιχτού κώδικα λογισμικό (Open Source), με την γνώση που είχαν αποκτήσει, δημιούργησαν το Kubernetes, το οποίο η Google και δώρησε στο Cloud Native Computing Foundation (CNCF) ενα χρόνο αργότερα. [10] Στην αρχή υποστήριζε μόνο Docker το οποίο είχε έρθει ένα χρόνο νωρίτερα, και αυτό περιόριζε αρκετά τις δυνατότητές του, έτσι στην πορεία το Kubernetes στην προσπάθεια να γίνει πιο ευέλικτο, εισήγαγε τη διεπαφή προγράμματος εκτέλεσης περιεκτών **(Container Runtime Interface, CRI)**.

Το CRI επιτρέπει σε οποιοδήποτε **πρόγραμμα εκτέλεσης περιεκτών (container runtime)** να επικοινωνήσει με το Kubernetes μέσω μιας διασύνδεσης προγράμματος εφαρμογής (API), με την προϋπόθεση να έχει αναπτυχθεί με βάση το CRI, το οποίο καθορίζει τις προδιαγραφές των containers ώστε να διασφαλίσει και την συνοχή αυτών. Έτσι αυτό ενθαρρύνει την

ανάπτυξη νέων προγραμμάτων εκτέλεσης περιεκτών χωρίς να χρειάζεται το Kubernetes να προσαρμόζεται για το καθένα.

4.1.1 Το εσωτερικό του Kubernetes

Μια **συστοιχία (cluster)**, στην τεχνολογία του Kubernetes, είναι στην ουσία μια συλλογή από **κόμβους (nodes)** που συνεργάζονται ως ένα ενιαίο σύστημα για να εξασφαλίσουν την αποτελεσματική λειτουργία εφαρμογών και υπηρεσιών. Στο νέφος αυτοί οι κόμβοι συνήθως είναι εικονικές μηχανές (ωστόσο δύναται να είναι και φυσικές μονάδες) που συνδέονται μεταξύ τους για να διαχειρίζονται τον φόρτο εργασίας και τους πόρους. Αυτό το σύστημα ενισχύει τη διαθεσιμότητα και την επεκτασιμότητα, καθώς οι εργασίες μπορούν να κατανεμηθούν σε πολλούς κόμβους και, εάν ένας κόμβος παρουσιάσει κάποιο πρόβλημα, μπορούν να αναλάβουν οι υπόλοιποι και να εξασφαλίσουν τη σωστή λειτουργία. Στην αρχιτεκτονική του Kubernetes, το Control Plane είναι ο κεντρικός κόμβος εντολών, υπεύθυνος για τη διαχείριση της συστοιχίας και την ενορχήστρωση των περιεκτών.



Σχήμα 4.1 Αρχιτεκτονική Kubernetes

- **etcd**: Είναι η βάση δεδομένων της συστοιχίας που αποτελείται από δεδομένα κλειδιού-τιμών (key-value), τα δεδομένα αυτά είναι τα δεδομένα διαμόρφωσης της συστοιχίας και της κατάστασής της.

- **Kube API server**: Λειτουργεί ως το σημείο πρόσβασης του control plane και διαχειρίζεται όλες τις εσωτερικές και εξωτερικές εντολές με βάση το REST πρωτόκολλο.

Τα υπόλοιπα στοιχεία της συστοιχίας επικοινωνούν με τον API server με αιτήματα όπως δημιουργία, διαγραφή ή τροποποίηση πόρων.

- **Controller**: Ο Controller Manager είναι αυτός που συγχρονίζει μια συλλογή χειριστών που ο καθένας τους έχει τη δική του αρμοδιότητα ως προς τα στοιχεία της συστοιχίας και διαχειρίζονται εργασίες που εκτελούνται σε πιο υψηλό επίπεδο.
- **Scheduler**: Ο Scheduler βαθμολογεί τους κόμβους με βάση το ποιοι είναι πιο κατάλληλοι να δεχτούν καινούργια pods, έτσι πάντα τα αναθέτει στο κόμβο που έχει και τους περισσότερους πόρους διαθέσιμους.
- **kubelet**: Βρίσκεται σε κάθε κόμβο της συστοιχίας και δέχεται αιτήματα για να ξεκινήσει κάποιο περιέκτη μέσα σε κάποιο pod, με τη σειρά του ζητά από τον πρόγραμμα εκτέλεσης περιεκτών, να φορτώσει την εικόνα που χρειάζεται και να ξεκινήσει ένα νέο περιέκτη.
- **kube proxy**: Βρίσκεται επίσης σε κάθε κόμβο και η δουλεία του είναι να επιτρέπει την επικοινωνία με τα pods, χρησιμοποιώντας κυρίως iptables για τους κανόνες δικτύωσης.

4.2 Οι πόροι του Κυβερνήτη

Ο ορισμός των πόρων στο Kubernetes γίνεται με την χρήση των αρχείων manifests είναι μία από τις βασικές ιδιότητες του Kubernetes. Αυτά τα αρχεία, γραμμένα σε YAML μορφή, λειτουργούν ως οδηγίες για τις εφαρμογές και τις υπηρεσίες του οικοσυστήματος Kubernetes. Ουσιαστικά, τα αρχεία αυτά δηλώνουν τον τρόπο με τον οποίο οι εφαρμογές και υπηρεσίες θα πρέπει να εγκαθίστανται και να διαχειρίζονται σε όλο το cluster του Kubernetes.

4.2.1 Pods

Έχουμε αναφέρει οτι η διαδικασία που ακολουθούμε όταν θέλουμε να τρέξουμε μια εφαρμογή σε ένα container, είναι να δημιουργήσουμε ένα image και με αυτό να ξεκινήσουμε το container, εμβαθύνοντας περισσότερο στη δημιουργία ενός image, σημαίνει ότι θα χρησιμοποιήσουμε ενα base image απο κάποιο registry (πχ Docker Hub) και θα το επεκτείνουμε με τις βιβλιοθήκες και τα εργαλεία που χρειαζόμαστε για να εκτελεστεί η εφαρμογή μας. Ένα Pod το οποίο είναι και ο πιο μικρός πόρος στο οικοσύστημα του Kubernetes, είναι μια λογική συλλογή ενός ή περισσότερων container που μοιράζονται την ίδια IP και μπορούν

εύκολα να επικοινωνούν μεταξύ τους χρησιμοποιώντας το εσωτερικό δίκτυο.
Παρακάτω βλέπουμε ένα παράδειγμα ενός YAML αρχείου για το δημιουργία ενός pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: myapp:latest
    ports:
    - containerPort: 8080
```

Κώδικας 4.1 Παράδειγμα pod manifest

Τα πιο σημαντικά πεδία όπως το kind που ορίζουμε το τύπο πόρου που δημιουργούμε, έπειτα τι image θα χρησιμοποιήσουμε και με το port τη θύρα την που θα ακούει η εφαρμογή, είναι κάποιες από τις ρυθμίσεις που μας επιτρέπει το Kubernetes να καθορίσουμε με το αρχείο Pod.

4.2.2 Deployment

Όπως όλοι οι πόροι στο Kubernetes, με ένα αρχείο παρέχουμε τις οδηγίες που θέλουμε, ωστόσο, ένα Deployment είναι μια συλλογή που διαχειρίζεται τα Pods καθορίζοντας διάφορες καταστάσεις τους. Χρησιμοποιεί ένα πρότυπο για τη δημιουργία Pods και αυτό το πρότυπο περιλαμβάνει τις προδιαγραφές για τα containers που πρέπει να βρίσκονται μέσα στα Pods που διαχειρίζεται. Το Deployment βοηθάει στην δημιουργία, την διαγραφή και κυρίως την ενημέρωση των Pods βάση τις οδηγίες που έχουμε ορίσει.
Παρακάτω είναι ένα παράδειγμα Deployement:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
  labels:
    app: myapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxUnavailable: 1
    maxSurge: 1
```

```
template:
  metadata:
    labels:
      app: myapp
  spec:
    containers:
    - name: myapp
      image: myapp:1.0.5
      ports:
      - containerPort: 8080
```

Κώδικας 4.2 Παράδειγμα ενός Deployemnt

Παρατηρούμε το συγκεκριμένο manifest ότι ενορχηστρώνει τα Pods διασφαλίζοντας ότι ο επιθυμητός αριθμός με το πεδίο replicas θα είναι πάντα 3 Pods και μας δίνει τη δυνατότητα για ενημερώσεις και επαναφορές με τη διατήρηση της διαθεσιμότητας και της αξιοπιστίας της εφαρμογής. Αν θέλουμε να ενημερώσουμε την εφαρμογή μας, θα φτιάξουμε ένα νέο image και θα εκτελέσουμε ενα rolling update. Έτσι, με την παρακάτω εντολή:

```
kubectl set image deployment/myapp-deployment myapp=myapp:2.0.5 --record
```

Συμβαίνει το εξής:

1. Ξεκινά με τη δημιουργία ενός νέου Pod με την νέα έκδοση της εφαρμογής **myapp:2.0.5** και είναι 1 το νέο Pod επειδή το **maxSurge** είναι 1

2. Μόλις το νέο Pod είναι έτοιμο και εξυπηρετεί την κυκλοφορία, προχωράει στο τερματισμό ενός από τα Pods με την παλιά έκδοση

3. Αυτή η διαδικασία επαναλαμβάνεται έως ότου όλα τα Pods ενημερωθούν στην τελευταία έκδοση, διασφαλίζοντας ότι τουλάχιστον 2 Pods μπορούν να εξυπηρετήσουν αφού συνολικά είναι 3 και με το πεδίο **maxUnavailable** του ορίζουμε να είναι μόνο ένα μη διαθέσιμο.

4.2.3 Service

Οι **υπηρεσίες (services)** είναι ένας τρόπος για την ανακάλυψη και την επικοινωνία μεταξύ διαφορετικών εφαρμογών ή μεταξύ τμημάτων της ίδιας εφαρμογής αναθέτοντας μια μοναδική διεύθυνση IP και ένα σύνολο θυρών σε μια ομάδα pods.
Υπάρχουν διάφοροι τύποι υπηρεσιών στο Kubernetes:

1. **ClusterIP**: Είναι μόνο προσβάσιμη εντός του cluster, αν θέλουμε να αποκτήσουμε πρόσβαση εκτός cluster πρέπει να την κάνουμε port forward

2. **NodePort**: Με βάση την θύρα που του καθορίζουμε και μέσω της IP του cluster μπορούμε να επικοινωνήσουμε και εκτός cluster

3. **LoadBalancer**: Αυτή η υπηρεσία είναι προσβάσιμη μόνο μέσω load-balancer

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service-clusterip
spec:
  selector:
    app: myapp
ports:
    - protocol: TCP
    port: 80
    targetPort: 8080
 type: ClusterIP

---

apiVersion: v1
kind: Service
metadata:
  name: myapp-service-nodeport
spec:
  selector:
    app: myapp
ports:
    - protocol: TCP
    port: 80
    argetPort: 8080
    nodePort: 30007
  type: NodePort
```

Κώδικας 4.3 Παράδειγμα υπηρεσίας με ClusterIP και NodePort

4.2.4 Ingress

Το Ingress είναι και αυτό ένας τύπος υπηρεσίας, είναι router που διαχειρίζεται την εξωτερική πρόσβαση στις υπηρεσίες ενός cluster. Το Ingress επιτρέπει να δρομολογούμε εύκολα την κυκλοφορία από το εξωτερικό του cluster στις υπηρεσίες εντός του cluster. Για παράδειγμα, αν έχουμε πολλές μικρουπηρεσίες που πρέπει να είναι προσβάσιμες από το διαδίκτυο, αντί να ρυθμίσουμε σημεία για κάθε μία, μπορούμε να έχουμε ένα entry point που δρομολογεί το /serviceA στην υπηρεσία A, το /serviceB στην υπηρεσία B. Επίσης με τη χρήση του Ingress μπορούμε να ρυθμίζουμε το φόρτο εργασίας και να ενεργοποιούμε SSL/TLS στις υπηρεσίες που εκθέτουμε στο διαδίκτυο.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
```

```
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: myapp.example.com
    http:
      paths:
      - path: /serviceA
        pathType: Prefix
        backend:
          service:
            name: my-service
            port:
              number: 80
```

Κώδικας 4.4 Παράδειγμα υπηρεσίας Ingress

4.3 Helm Charts

Τα Helm charts [17] παρέχουν έναν εύκολο τρόπο στο να διαχειριζόμαστε τις εφαρμογές σε μια Kubernetes συστοιχία, μειώνοντας κατά πολύ την πολυπλοκότητα που δημιουργεί μια αρχιτεκτονική μικρουπηρεσιών. Το Helm είναι ένας διαχειριστής πακέτων (package manager) του Kubernetes, το οποίο χρησιμοποιεί τα charts τα οποία είναι YAML αρχεία και περιέχουν όσα χρειάζονται για την λειτουργία μιας εφαρμογής, στην ουσία το τελικό αποτέλεσμα είναι μια συλλογή από μανιφέστα, τα charts όμως επιτρέπουν τη χρήση μεταβλητών και προσφέρουν έναν πιο δυναμικό τρόπο για την δημιουργία προτύπων (templates) με βάση διαφορετικά περιβάλλοντα. Με τα Helm charts, κάθε εγκατάσταση στη συστοιχία είναι μια **έκδοση (release)**, με την εντολή *helm install* δημιουργούμε μια νέα έκδοση μαζί με όλους του πόρους που χρειάζεται, με την εντολή *helm upgrade* εφαρμόζουμε αλλαγές στη εγκατάσταση διορθώνοντας όποια ρύθμιση και αυξάνουμε την έκδοση επιτρέποντας παράλληλα να διατηρούμε το ιστορικό των εκδόσεων, όπου με την εντολή *helm rollback* μπορούμε να επαναφέρουμε μια προηγούμενη έκδοση παρέχοντας έναν τρόπο για ταχεία ανάκαμψη της εγκατάστασης από κάποιο πρόβλημα.

Η δομή ενός chart αποτελείται από ένα *Chart.yaml* αρχείο το οποίο περιέχει μεταδεδομένα (metadata) δηλαδή πληροφορίες για το chart όπως όνομα και περιγραφή, το αρχείο *values.yaml* που περιέχει τις τιμές τις οποίες θα αντικαταστήσουν στα πρότυπα για την δυναμική προσαρμογή της εγκατάστασης του chart, έπειτα ο φάκελος *templates* που περιέχει όλα τα αρχεία προτύπων. Τα βήματα είναι τα εξής, στην αρχή το Helm διαβάζει το chart και δημιουργεί μανιφέστα στέλνοντας τις τιμές απο το αρχείο *values.yaml* στα πρότυπα, έπειτα αυτά τα μανιφέστα στέλνονται στο Kubernetes όπου είναι η σειρά του να δημιουργήσει όλους του πόρους που έχουν ζητηθεί.

4.4 Δημιουργία μιας συστοιχίας

Το minikube είναι μια υλοποίηση του Kubernetes που δημιουργεί μια εικονική μηχανή στον υπολογιστή μας και δημιουργεί ένα cluster με ένα κόμβο (single node). Το Minikube έχει σχεδιαστεί για να βοηθήσει τους προγραμματιστές και τους νέους χρήστες να πειραματιστούν με το Kubernetes και να αναπτύξουν εφαρμογές τοπικά. Θα το χρησιμοποιήσουμε για την υλοποίηση την διπλωματικής εργασίας.

4.4.1 Χρήση ενός οδηγού βασισμένο σε περιέκτες

Για την υλοποίηση μας θα εγκαταστήσουμε το minikube με έναν οδηγό που βασίζεται σε περιέκτες, όπως το Docker, έτσι το cluster του Kubernetes δημιουργείται μέσα σε ένα περιέκτη αντί για μια εικονική μηχανή. Για περιβάλοντα ανάπτυξης και δοκιμών είναι ιδανικό καθώς προσομοιώνει πολύ καλά ένα ολοκληρωμένο περιβάλλον Kubernetes χωρίς του πιο αργούς χρόνους των πιο παραδοσιακών λύσεων εικονικών μηχανών όπως VirtualBox, QEMU και KVM2.

4.4.2 Εφαρμογή μιας CNI επέκτασης

Η δικτύωση στο Kubernetes αντιμετωπίζει διάφορες κρίσιμες πτυχές της λειτουργίας του cluster. Αυτό περιλαμβάνει την επικοινωνία pod-to-pod από κόμβο σε κόμβο, η οποία επιτρέπει στα pods σε διαφορετικούς κόμβους να συνομιλούν μεταξύ τους. Η ανακάλυψη υπηρεσιών (service discovery) είναι μια άλλη βασική ιδιότητα, επιτρέποντας στα pods να βρίσκουν και να επικοινωνούν μεταξύ τους χρησιμοποιώντας ονόματα υπηρεσιών αντί να βασίζονται σε άμεσες διευθύνσεις IP. Καθώς και οι υπηρεσίες για εξωτερική πρόσβαση που επιτρέπουν την πρόσβαση εκτός συστοιχίας και να μπορούν να προσεγγίζουν υπηρεσίες που εκτελούνται μέσα σε αυτή, όπως αναφερθήκαμε προηγουμένως σε αυτές.

Η ασφάλεια δικτύου στο Kubernetes αφορά την απομόνωση και την προστασία των pods και των υπηρεσιών από μη εξουσιοδοτημένη πρόσβαση, η οποία συχνά διαχειρίζεται μέσω πολιτικών δικτύου που καθορίζουν τον τρόπο με τον οποίο τα pods μπορούν να επικοινωνούν μεταξύ τους. Το Kubernetes δεν ασχολείται εν μέρει με αυτό και μέσω του πλαισίου CNI (Container Network Interface) που γεφυρώνει το Kubernetes με την υλοποίηση δικτύου, παρέχει ένα πρότυπο και αφήνει τις υλοποιήσεις να εξελίσσονται εκτός από αυτό, παραμένωντας ωστόσο συμβατά μεταξύ τους. Όταν δημιουργείται ένα pod σε έναν κόμβο, το kubelet καλεί την CNI επέκταση για να ρυθμίσει το δίκτυο για του, στην ουσία του εκχωρεί διεύθυνση IP και τις ρυθμίσεις στο με ποια άλλα pods μπορεί να επικοινωνεί.

Μεταξύ των πολλών διαθέσιμων υλοποιήσεων CNI, εμείς θα χρησιμοποιήσουμε το calico εξερευνώντας τις δυνατότητες στην υλοποίηση του μοντέλου δικτύου του Kubernetes.

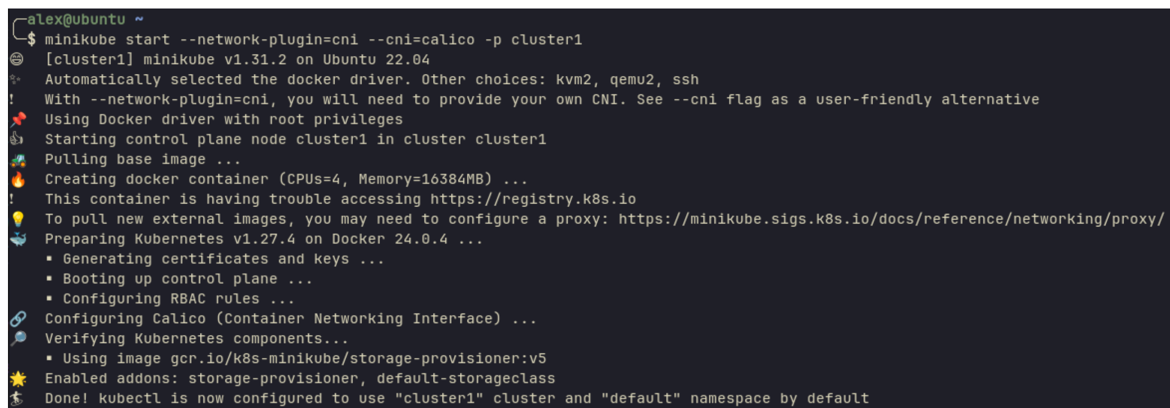
Συνοψίζοντας, η εγκατάσταση είναι πολύ εύκολη, το μόνο που έχουμε να κάνουμε είναι να κατεβάσουμε το binary και να το εγκαταστήσουμε στο περιβάλλον Linux (ή Windows, MacOS)

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Έπειτα με τις εντολές

```
minikube start --network-plugin=cni --cni=calico -p cluster1
```

δημιουργούμε τοπικά μια συστοιχία εφαρμόζοντας την επέκταση του calico για την διαχείρηση της δικτύωσης.



Σχήμα 4.2 Έναρξη minikube

Κεφάλαιο 5

Ανάλυση των πάροχων νέφους

5.1 Identity και Access Management

Στο περιβάλλον του cloud και καθώς τα cloud-native συστήματα όλο και μεγαλώνουν, περιλαμβάνοντας ένα μεγαλύτερο εύρος μικρουπηρεσιών, η ανάγκη για ασφαλής και διαχειρίσιμη πρόσβαση είναι άκρως σημαντική. Εκεί έρχεται το IAM το οποίο με τις κατάλληλες ρυθμίσεις διασφαλίζει ποίος έχει πρόσβαση, σε ποίες υπηρεσίες και κάτω από ποίες συνθήκες. Παρακάτω γίνεται μια επισκόπηση των IAM υπηρεσιών που παρέχονται από τους πιο γνωστούς παρόχους όπως Amazon Web Services (AWS), Google Cloud Platform (GCP) και Microsoft Azure.
Ορισμένα θεμελιώδη στοιχεία του είναι :

- **Αυθεντικοποίηση**: Στο IAM η αυθεντικοποίηση υπερβαίνει την επαλήθευσης της ταυτότητας ενός ατόμου, περιλαμβάνοντας τον έλεγχο ταυτότητας πολλαπλών παραγόντων (MFA), όπου οι χρήστες πρέπει να παρέχουν δύο ή περισσότερους παράγοντες επαλήθευσης για να αποκτήσουν πρόσβαση σε πόρους.

- **Εξουσιοδότηση**: Περιλαμβάνει ελέγχους πρόσβασης που ελέγχουν το κατά πόσο ένας χρήστης έχει πρόσβαση στους πόρους σύμφωνα με τους ρόλους, τις πολιτικές και τις προϋποθέσεις.

- **Ταυτότητα**: Η ”ταυτότητα” αναφέρεται στην ψηφιακή αναπαράσταση ενός ατόμου σε ένα σύστημα. Περιέχει τα χαρακτηριστικά και τους ρόλους που αποδίδονται στην εν λόγω οντότητα, με σκοπό την αυθεντικοποίηση και την εξουσιοδότηση για πρόσβαση σε πόρους και υπηρεσίες.

Πόρος

Στο οικοσύστημα μίας cloud υπηρεσίας, είτε έχει να κάνει με AWS, Azure ή GCP ο όρος του

πόρου (resource) συνοψίζει μια οντότητα που δημιουργούμε, διαχειριζόμαστε και χρησιμοποιούμε κάτω από ένα σύνολο υπηρεσιών του εκάστοτε προαναφερθέντα cloud παρόχου. Αυτή η οντότητα μπορεί να είναι μια εικονική μηχανή (VM) όπως ενα AWS EC2, Azure VMs, ή GCP Compute ή κάποια υπηρεσία βάσης δεδομένων όπως και οποιαδήποτε άλλη υπηρεσία ή προιόν του οικοσυστήματος που χρησιμοποιούμε. Η διαχείριση που σχετίζεται με την αυθεντικοποίηση και εξουσιοδότηση των χρηστών και υπηρεσιών βασίζεται πάνω σε αυτούς τους πόρους και είναι και οι ίδιες οι οντότητες που έχουν σκοπό να προστατέψουν οι ρυθμίσεις του IAM.

5.1.1 AWS IAM

Στο AWS IAM οι χρήστες εξουσιοδοτούνται ώστε να έχουν την δυνατότητα να πραγματοποιούν **ενέργειες (actions)** με βάση τις **πολιτικές (policies)** που τους καθορίζονται.

5.1.1.1 Identity-base policies και Resource-based policies

Στην διαχείριση της πρόσβασης, δύο είναι οι τύποι που έχουν σημαντικό ρόλο, αυτοί είναι οι : πολιτικές βασισμένες στη ταυτότητα και πολιτικές βασισμένες στους πόρους (identity and Resource Policies) Οι πολιτικές ταυτότητας είναι συνδεδεμένες με τις IAM οντότητες όπως οι χρήστες, τα γκρούπς και οι ρόλοι, με σκοπό να παρέχουν άδεια να χρησιμοποιούν πόρους και υπηρεσίες, ενώ αντιθέτως οι πολιτικές πόρων είναι συνδεδεμένες με τους ίδιους του πόρους. Οι πολιτικές γράφονται σε JSON μορφή όπου πεδία όπως Effect, Action, Resource και Condition είναι τα θεμελιώδη για την ρύθμιση της πρόσβασης. Το πεδίο Effect είναι αναγκαίο σε κάθε έγγραφο και καθορίζει μέσα από τιμές Allow ή Deny αν παρέχουμε άδεια ή όχι αντίστοιχα με βάση τι επόμενες ρυθμίζεις του εγγράφου.Πολιτικές ταυτότητας και πόρων Το πεδίο Action είναι μια λίστα από τις ενέργειες που επιτρέπεται ή όχι να γίνουν κάτω από συγκεκριμένες ή και όλες τις AWS υπηρεσίες. Το πεδίο Resource είναι και ο πόρος που αφορά το συγκεκριμένο έγγραφο και οι ρυθμίσεις που περιέχει. Τέλος το πεδίο Condition επιτρέπει επιπλέον ρυθμίσεις για το πότε εφαρμόζεται η πολιτική κάτω από συγκεκριμένες συνθήκες.

```
{
        "Version": "2012-10-17",
        "Statement": [
        {
                "Effect": "Allow",
                "Action": "service-prefix:action-name",
                "Resource": "*",
                "Condition": {
                        "DateGreaterThan": {"aws:CurrentTime": "2020-04-01T00:00:00Z"},
```

```
                        "DateLessThan": {"aws:CurrentTime": "2020-06-30T23:59:59Z"}
                }
        }
        ]
}
```

Κώδικας 5.1 Παράδειγμα πολιτικής βασισμένη σε ταυτότητα σε μορφή JSON

Με τις πολιτικές πόρων, όπως αναφέραμε επισυνάπτουμε τις πολιτικές στους πόρους και όχι στις IAM οντότητες, για αυτό και προσθέτωντας ενα πεδίο Principal μπορούμε να καθορίσουμε ρητά σε ποιους εφαρμόζεται η εκάστοτε πολιτική.

Ρητή απαγόρευση πρόσβασης

Χαρακτηριστικό είναι ότι πάντα η ρητή απαγόρευση πρόσβασης σε κάποιο πόρο για ορισμένο χρήστη έχει μεγαλύτερη ισχύ ακόμα και αν υπάρχει πολιτική πόρου που του επιτρέπει την πρόσβαση.

5.1.1.2 Γκρούπς και ρόλοι

Τα IAM γκρούπς είναι ένα ευκολότερος τρόπος στην διαχείριση πρόσβασης μεταξύ χρηστών, αναθέτοντας πολιτικές σε ένα γκρούπ, όσοι χρήστες ανήκουν σε αυτό το γκρούπ ”κληρονομούν” και τις πολιτικές αντί να διαχειριζόμαστε κάθε χρήστη ξεχωριστά.
Οι ρόλοι είναι ένας πιο πολύπλοκος τρόπος να εξουσιοδοτούμε ένα χρήστη συγκριτικά με τους παραπάνω τρόπους αλλά έχει τα οφέλη του. Μπορεί να χρησιμοποιηθεί είτε από έναν IAM χρήστη ή και υπηρεσία, με τον ίδιο τρόπο που αναθέτουμε μια εξουσιοδότηση σε ένα γκρούπ έτσι εξουσιοδοτούμε και ένα ρόλο με χρήση πολιτικών, θα χρειαστεί επίσης να αναθέσουμε ένα policy που να επιτρέπει στον χρήστη να κάνει assume ένα ρόλο. Έπειτα με την παρακάτω εντολή κάνουμε assume το ρόλο

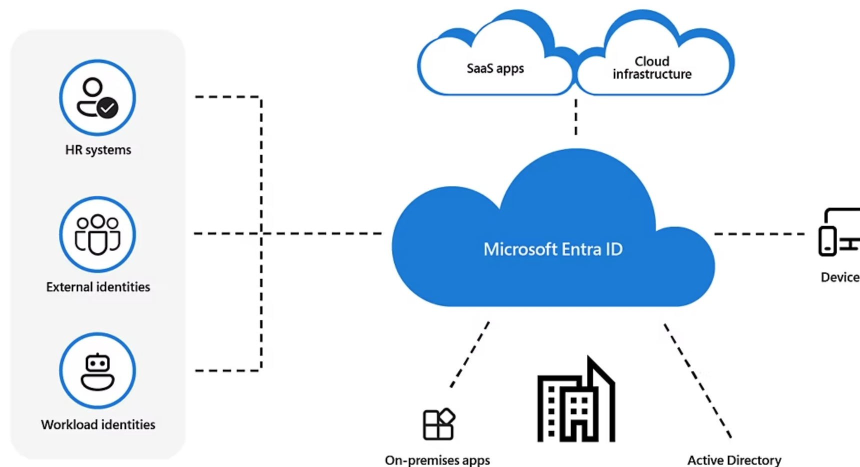
```
        aws sts assume-role \
        --role-arn "arn:aws:iam::ACCOUNT_ID:role/ROLE_NAME" \
        --role-session-name SESSION_NAME \
        --duration-seconds 800
```

Credentials με μεγάλη διάρκεια ζωής

Με αυτό το τρόπο μειώνουμε σημαντικά την χρήση των credentials που έχουν μεγάλη διάρκεια ζωής και έτσι ελαχιστοποιούμε το ρίσκο για οποιαδήποτε μη εξουσιοδοτημένη χρήση ακόμα και αν τα credentials διαρρεύσουν.

5.1.2 Microsoft Entra ID

[1]Πηγή: https://www.microsoft.com/en-us/security/business/identity-access/microsoft-entra-id



Σχήμα 5.1 Microsoft Entra ID[1]

Το Microsoft Entra ID (πρώην Azure AD) είναι η υπηρεσία διαχείρισης ταυτότητας και πρόσβασης της Microsoft, που μπορεί να χαρακτηριστεί και σαν ένας Identity Provider παρέχοντας την δυνατότητα να χρησιμοποιηθεί για εφαρμογές πέραν του Microsoft οικοσυστήματος.

5.1.2.1 Role-Based Access Control

Το Azure RBAC είναι το κύριο εργαλείο της εξουσιοδότησης του Entra ID όπου στον πυρήνα του, επιτρέπει την επιβολή της αρχής του **least privilege**, διασφαλίζοντας ότι οι χρήστες και οι υπηρεσίες έχουν μόνο την πρόσβαση που είναι απαραίτητη, ελαχιστοποιώντας τον κίνδυνο μη εξουσιοδοτημένης πρόσβασης.

Ανάθεση ρόλου

Ο Security Principal αναφέρεται σε μία ταυτότητα που μπορεί να είναι ένας χρήστης, ένα γκρούπ ή και μια υπηρεσία, συνεπώς είναι ο ”ποίος” στην αυθεντικοποίηση του Entra ID, στην οποία αναθέτουμε την συλλογή από permissions για ενέργειες κάτω από τα όρια που έχουν καθοριστεί μέσα από ένα συγκεκριμένο scope. Αυτό συμβαίνει μέσω της **ανάθεσης ρόλου**, την οποία δημιουργούμε από χρήστη που του επιτρέπεται η συγκεκριμένη ενέργεια, οι ρόλοι που μπορούμε να αναθέσουμε χωρίζονται σε:

- **Built-in roles**: Ο Owner που έχει πλήρης πρόσβαση παντού και μπορεί να παραχωρεί και πρόσβαση, Contributor μοιάζει με τον Owner χωρίς να μπορεί να παραχωρήσει πρόσβαση, Reader που του επιτρέπεται μόνο η ανάγνωση και User access administrator που κυρίως έχει να κάνει με την διαχείριση πρόσβασης.

- **Custom roles**: Το Entra ID παρέχει την ευχέρεια να μπορούμε να δημιουργούμε ρόλους κάτω από προσαρμοσμένες απαιτήσεις με ένα συγκεκριμένο συνδυασμό επιτρεπόμενων ενεργειών. Μέσω JSON μορφή, περιγράφουμε τις ενέργειες που επιτρέπεται να πραγματοποιήσει ο ρόλος.

5.1.2.2 Microsoft Entra MFA

Το Multi-Factor Authentication (MFA) είναι ένας μηχανισμός ασφαλείας που απαιτεί από τον χρήστη να ταυτοποιήσει τον εαυτό του και με έναν δεύτερο ή και περισσότερους τρόπους πέρα από τον συνδυασμό του ονόματος χρήστη και κωδικό πρόσβαση. Αυτό γίνεται συνήθως μέσω της εφαρμογής Microsoft Authenticator αλλα και με άλλες επιλογές όπως SMS ή και τηλεφωνική κλήση. Το MFA είναι εξαιρετικά σημαντικό, διότι αντιμετωπίζει τα τρωτά σημεία του να βασίζεσαι αποκλειστικά σε κωδικούς πρόσβασης, προσθέτωντας έτσι ενα έξτρα επίπεδο ασφαλείας.

Conditional access

Η πρόσβαση υπό προϋποθέσεις του Entra ID παρέχει την δυνατότητα την επιβολή του MFA σε μια προσπάθεια σύνδεσης ενός χρήστη. Η πολιτική αυτή μπορεί να διαμορφωθεί με βάση τις ανάγκες μας και να ενεργοποιείται κάτω από συγκεκριμένες προϋποθέσεις όπως τοποθεσία, συσκευή ή εφαρμογή, υπολογίζοντας σε πραγματικό χρόνο το επίπεδο ρίσκου της προσπάθειας σύνδεσης.

5.1.3 Google Cloud IAM

5.1.3.1 Το ιεραρχικό μοντέλο της Google

Το Google Cloud Platform (GCP) χρησιμοποιεί ένα ιεραρχικό σύστημα [**?**] για τις οντότητες και τους πόρους προσφέροντας μεγάλη ευκολία στη διαχείριση της πρόσβασης. Απευθυνόμενο κυρίως σε επιχειρήσεις, το μοντέλο αυτό είναι ιδανικό και πολύ φιλικό ως προς την επεκτασιμότητα του ελέγχου της πρόσβασης, ιδιαίτερα σημαντική απαίτηση από τους οργανισμούς.

Οργανισμός

Ένας οργανισμός είναι η ρίζα που περιλαμβάνει όλους τους πόρους και τις υπηρεσίες που χρησιμοποιεί μια εκάστοτε εταιρεία.

Φάκελος

Οι φάκελοι δίνουν την ευχέρεια στους οργανισμούς σε μια πιο λεπτομερή διαχείριση πρόσβασης έχοντας την δυνατότητα ένας φάκελος να μπορεί να αντιπροσωπεύει μια φυσική

οντότητα όπως ένα τμήμα ή μια ομάδα μιας εταιρείας.

Πρότζεκτ

Το πρότζεκτ είναι ο χαμηλότερος πόρος αλλά και μια αναγκαία οντότητα καθώς στην οποία θα εφαρμοστούν οι πολιτικές πρόσβασης που έχουμε καθορίσει για τις υπηρεσίες που χρησιμοποιούμε.

5.1.3.2 Service Accounts

Το service account είναι ένας ξεχωριστός λογαριασμός που προσφέρει έναν μηχανισμό για αυθεντικοποίηση και εξουσιοδότηση server-to-server ανάμεσα σε υπηρεσίες του GCP οικοσυστήματος ή και προς εξωτερικά συστήματα. Ο λογαριασμός αυτός ανήκει σε μια εφαρμογή και όχι σε κάποιο χρήστη, δίνοντας την δυνατότητα για μια αλληλεπίδραση κάτω από συγκεκριμένα δικαιώματα που έχουν δοθεί στο service account με τον ίδιο τρόπο που θα αναθέταμε έναν ρόλο σε έναν χρήστη. Γίνεται σαφές πόσο ισχυρό εργαλείο είναι για τις μικρουπηρεσίες ένας τέτοιος τρόπος αυτοματοποιημένης αυθεντικοποίησης απο μια εφαρμογή χωρίς να χρειάζεται ανθρώπινη παρέμβαση και τα credentials κάποιου χρήστη. Ωστόσο, περιέχει και μεγάλη ευθύνη, η τήρηση της αρχής του **least privelege** είναι σημαντική καθώς μια εφαρμογή δεν θα πρέπει να έχει πρόσβαση σε κάποια υπηρεσία που δεν είναι αναγκαία για την λειτουργία της.

Μίμηση ενός service account

Το GCP επιτρέπει σε έναν χρήστη να υποδυθεί έναν λογαριασμό υπηρεσίας και να χρησιμοποιήσει τα δικαιώματα πρόσβαση που έχει, αυτό εισάγει ένα ακόμη επίπεδο ευθύνης για αυτό και είναι ανάγκη να ελέγχονται αυστηρά οι χρήστες που έχουν τη δυνατότητα να υποδύονται λογαριασμούς υπηρεσιών.

Monitoring ενός service account

Δεδομένου ότι κάθε service account είναι μοναδικό, γίνεται πιο σαφή και εύκολη η καταγραφή των ενεργειών που πραγματοποιήθηκαν από τον λογαριασμό και έτσι και η ανίχνευση λαθών από την συγκεκριμένη εφαρμογή σε κάποια υπηρεσία.

5.2 Network Security

Σε αυτή την ενότητα, θα αναλύσουμε ορισμένα από τα εργαλεία και τις υπηρεσίες ασφάλειας δικτύου που έχουν να προσφέρουν οι προαναφερθείς πάροχοι, εμβαθύνοντας στις υπηρεσίες που έχουν σχεδιαστεί για να προστατεύουν την υποδομή του δικτύου μιας cloud αρχιτεκτονικής. Όπως και με το IAM έτσι και εδώ θέλουμε να ελέγχουμε την πρόσβαση στου πόρους

του οικοσυστήματος μας, έτσι από κανόνες για πολιτικές θα συνεχίσουμε με κανόνες στο επίπεδο δικτύου.

5.2.1 Εικονικό ιδιωτικό νέφος

Συνεχίζοντας με τον έλεγχο των πόρων σε επίπεδο δικτύου θα εμβαθύνουμε στο **εικονικό ιδιωτικό νέφος (Virtual Private Cloud, VPC)**, το οποίο είναι ένα εικονικό δίκτυο και έχει σαν στόχο την προσομοίωση ενός φυσικού δικτύου αλλά με πόρους cloud και την απομόνωση αυτών μέσω ορισμένων κανόνων.

Υποδίκτυα

Τα **Υποδίκτυα (Subnets)** είναι μικρότερα δίκτυα που δημιουργούμε σε ενα VPC, κάθε εικονική μηχανή θα πρέπει να ανήκει σε ένα υποδίκτυο για αυτό και μαζί με τον VPC είναι αναγκαία και η δημιουργία υποδίκτυο καθώς καθορίζουν

Security Groups

Τα **security groups** θα μπορούσαμε να τα χαρακτηρίσουμε μια πιο φιλική εκδοχή των παραδοσιακών iptables σε cloud επίπεδο, χρησιμοποιούνται για τον έλεγχο της εισερχόμενης και εξερχόμενης (inbound και outbound) κίνησης δικτύου. Τα security groups των AWS και Azure αλλά και ο παρόμοιος μηχανισμός του GCP τα firewall rules, είναι stateful που σημαίνει αν έχουμε καθορίσει ρητά να επιτρέπουμε την εξερχόμενη κίνηση επιτρέπουμε αυτόματα και την επιστροφή από αυτή την διεύθυνση. Το AWS δεν υποστηρίζει την δυνατότητα να ρυθμίσεις ένα κανόνα άρνησης (deny) της κίνησης σε ενα security group, κάτι όμως που μπορεί να γίνει με έναν παρόμοιο μηχανισμό τα δίκτυα ACL.

Δίκτυα ACL

Εφαρμόζονται στο επίπεδο υποδικτύου και διαφέρουν στο ότι αν θέλουμε να επιτρέψουμε την επιστροφή από μια εξερχόμενη κίνηση στο δίκτυο θα πρέπει να το δηλώσουμε ρητά εν αντιθέσει με τα security groups που δεν χρειάζεται.

Πίνακας 5.1 Συγκριτικός πίνακας VPC

	AWS VPC	**Azure VNet**	**Google Cloud VPC**
Security groups	Stateful	Stateful	Stateful
Υποστήριξη Υποδικτύωσης	Ναι	Ναι	Ναι
Υποστήριξη κανόνων άρνησης σε Security groups	Όχι	Ναι	Ναι
Υποστήριξη ACL Δικτύων	Ναι	Όχι	Όχι

5.2.2 Υπηρεσίες WAF

Η υπηρεσία **WAF (Web Application Firewalls)** είναι ακόμη ένας μηχανισμός ασφάλειας που όμως δεν λειτουργεί στο επίπεδο δίκτυο αλλά στο επίπεδο εφαρμογής, παρακολουθώντας την κυκλοφορία που εισέρχεται και εξέρχεται σε μια cloud εφαρμογή. Αυτό επιτρέπει τον εντοπισμό διαφόρων ευπαθειών της εφαρμογής διασφαλίζοντας την ακεραιότητα των δεδομένων και διατηρώντας την διαθεσιμότητα της εφαρμογής από τυχόν επιθέσεις μέσω κανόνων που προσαρμόζονται στις ανάγκες της.

Κεφάλαιο 6

Προκλήσεις στις νεοφυείς εφαρμογές και τρόποι αντιμετώπισης

Έχοντας αναφερθεί στο εξελισσόμενο τοπίο των cloud-native τεχνολογιών και στην στροφή προς τις μικρουπηρεσίες, το containerization και την ενορχήστρωση, πολλές φορές οι γρήγοροι αυτοί ρυθμοί ανάπτυξης και διάδοσης λογισμικού, παραβλέπουν κάπως την ασφάλεια που δεν συμβαδίζει πάντα με την ταχύτητα των προηγουμένων. Αυτό απαιτεί μια διαφορετική προσέγγιση και μια στρατηγική που να εκμεταλλεύεται και να συμβαδίζει με την ευελιξία σε αυτές τις τεχνολογίες. Αυτό το κεφάλαιο έχει ως στόχο να εμβαθύνει στις προκλήσεις ασφαλείας που προκύπτουν στο Kubernetes και στις cloud-native εφαρμογές, τις απειλές και τις λανθασμένες ρυθμίσεις που μπορούν να θέσουν σε κίνδυνο την ασφάλεια και αξιοπιστία των εφαρμογών και του περιβάλλον τους. Διερευνώντας λύσεις και βέλτιστες πρακτικές που έχουν σχεδιαστεί για την πρόληψη των cloud-native εφαρμογών έναντι προκλήσεων και παραβάσεων, θέτοντας τις βάσεις για την υλοποίηση μιας ασφαλούς και ανθεκτικής cloud-native αρχιτεκτονικής.

6.1 Αυθεντικοποίηση και εξουσιοδότηση

Ας υποθέσουμε ότι έχουμε μια πλατφόρμα που παρέχει διάφορες υπηρεσίες στους χρήστες της, αυτό χωρίς ένα ενιαίο σύστημα αυθεντικοποίησης θα μπορούσε εκτός απο το να μην είναι φιλικό προς τους χρήστες, να προκαλέσει και θέματα ασφαλείας διότι οι χρήστες μπορεί να χρησιμοποιούν αδύναμους κωδικούς για δική τους ευκολία. Οι χρήστες επίσης συνήθως θέλουν να μπορούν σε διάφορες εφαρμογές να χρησιμοποιούν κάποιο υφιστάμενο λογαριασμό απο κάποιον μεγάλο πάροχο υπηρεσιών (όπως Google, Microsoft). Συχνά στην αρχιτεκτονική μικρουπηρεσιών (και όχι μονο) βρίσκεται η ανάγκη να πρέπει να ενώθουν

δύο συστήματα για ανταλλαγή των πληροφοριών εξουσιοδότησης ενός χρήστη υποστηρίζοντας διαφορετικά πρωτόκολλα ή έχοντας υλοποιημένους διαφορετικούς ρόλους ως προς την πρόσβαση στις εκάστοτε υπηρεσίες. Αυτό μαζί με την ανάγκη αυτές οι πληροφορίες να πρέπει να είναι συγχρονισμένες και διατηρήσιμες και απο τις δύο πλευρές ταυτόχρονα κάνει πολύ δύσκολο εώς και ακατόρθωτο το έργο των διαχειριστών διότι σε περίπτωση μη διατήρησης της συνέπειας αυτών των δεδομένων μπορεί να προκαλέσει προβλήματα πρόσβασης μειώνοντας την παραγωγικότητα των χρηστών ή βάζοντας την ασφάλεια σε κίνδυνο για πρόσβαση σε πληροφορίες που δεν θα έπρεπε να έχουν. Με βάση τα παραπάνω φαίνεται η ανάγκη για ενα ενοποιημένο σύστημα αυθεντικοποίησης και εξουσιοδότησης.

6.1.1 SSO

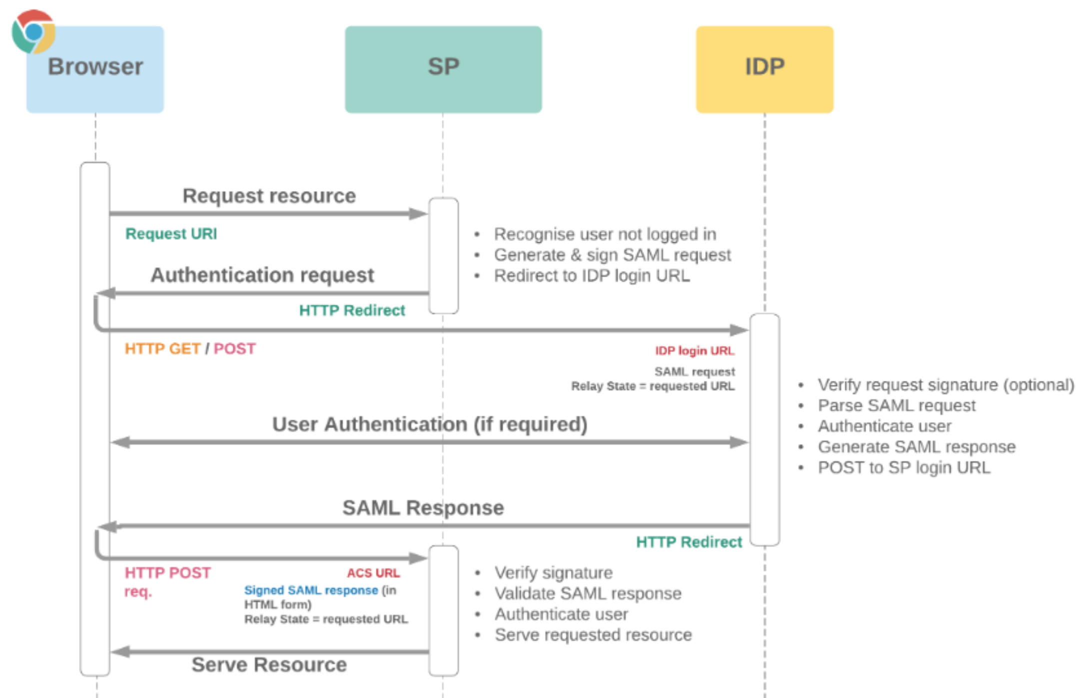
Με το Single sing-on (SSO) ο χρήστης μπορεί να αυθεντικοποιείται μια φορά και να του παρέχεται η πρόσβαση σε διάφορες εφαρμογές κάτω απο ένα ενιαίο σύστημα εφόσον η συνεδρία (session) του χρήστη είναι ακόμα ενεργή. Έτσι παρέχει μεγάλη ευκολία στους χρήστες αλλά και συνέπεια ως προς τις πρακτικές που έχει ορίσει η οντότητα που κάνει χρήση του SSO που μπορεί να είναι είτε μια δημόσια υπηρεσία, είτε ένας οργανισμός ή ενα πανεπιστήμιο.

6.1.2 SAML

Το SAML (Security Assertion Markup Language) είναι ένα πρωτόκολλο βασισμένο σε XML (Extensible Markup Language) για την ανταλλαγή δεδομένων αυθεντικοποίησης και εξουσιοδότησης μεταξύ ενός Παρόχου Ταυτότητας **(Identity Provider)** και ενός Παρόχου Υπηρεσίας **(Service Provider)**. Χρησιμοποιείται για το (SSO) ώστε να επιτρέπει στους χρήστες να έχουν πρόσβαση σε πολλαπλές εφαρμογές ή υπηρεσίες κάνοντας log-in μόνο μια φορά, βελτιώνοντας έτσι την εμπειρία του χρήστη και ενισχύοντας την ασφάλεια. Το SAML ορίζει ένα πλαίσιο για την ανταλλαγή πληροφοριών ασφαλείας, συμπεριλαμβανομένων βεβαιώσεων σχετικά με την ταυτότητα, ταχαρακτηριστικά και τα δικαιώματα ενός χρήστη. Το SAML παρέχει έναν μηχανισμό για μια εφαρμογή και έναν πάροχο ταυτότητας να χρησιμοποιούν ένα κοινό αναγνωριστικό για έναν χρήστη προκειμένου να ανταλλάσσουν πληροφορίες για αυτόν σε διάφορα συστήματα. Τα πιο σημαντικά στοιχεία του SAML είναι τα εξής:

- **Identity Provider**: Ο IdP είναι υπεύθυνος για την αυθεντικοποίηση των χρηστών και την έκδοση των SAML assertions.

- **Service Provider**: Ο πάροχος υπηρεσιών είναι η εφαρμογή ή η υπηρεσία στην οποία θέλει να έχει πρόσβαση ο χρήστης, επικοινωνεί με τον πάροχο ταυτότητας για την αυθεντικοποίηση των χρηστών.
- **SAML Assertion**: Είναι στην ουσία ένα πακέτο απο πληροφορίες που στέλνει ο IdP στον SP και περιέχει τις πληροφορίες όπως ταυτότητα και δικαιώματα του χρήστη, με την δυνατότητα να μπορούμε να προσθέσουμε σε αυτό ό,τι πεδία χρειαζόμαστε να μεταφέρουμε.
- **Συνδέσεις**: Είναι οι μεθόδοι που χρησιμοποιούν οι δύο άκρες για να μεταφέρουν τα μηνύματα, μπορεί να είναι HTTP Post καί HTTP Redirect.



Σχήμα 6.1 Παράδειγμα SAML[1]

Στο πάνω διάγραμμα βλέπουμε μια ροή του SAML πρωτοκόλλου που ξεκινάει απο τον πάροχο υπηρεσιών (SP-Initiated)

[1]Πηγή: https://cloudsundial.com/salesforce-sso-flows/

6.1.3 Ο ρόλος του διαμεσολαβητή

Το SAML είναι σχετικά πολύπλοκο διότι έχει πολλές ρυθμίσεις οι οποίες απαιτούν και βαθιά γνώση του πως δουλεύει το πρωτόκολλο, αυτό έχει ως αποτέλεσμα την χρήση διάφορων βιβλιοθηκών και λύσεων που μπορούν να ελαχιστοποιήσουν το φόρτο ανάπτυξης ενός client από το μηδέν. Μία από αυτές τις λύσεις είναι και ένας διαμεσολαβητής αυθεντικοποίησης (authentication broker) που βοηθάει να ενσωματωθούν διάφοροι μέθοδοι αυθεντικοποίησης χρηστών σε εφαρμογές. Μπορεί να υποστηρίξει πολλαπλά πρωτόκολλα και επιτρέπει μια πιο κεντρική αλλά και αγνωστική διαχείριση των ελέγχων ταυτότητας.

6.1.4 OAuth 2.0

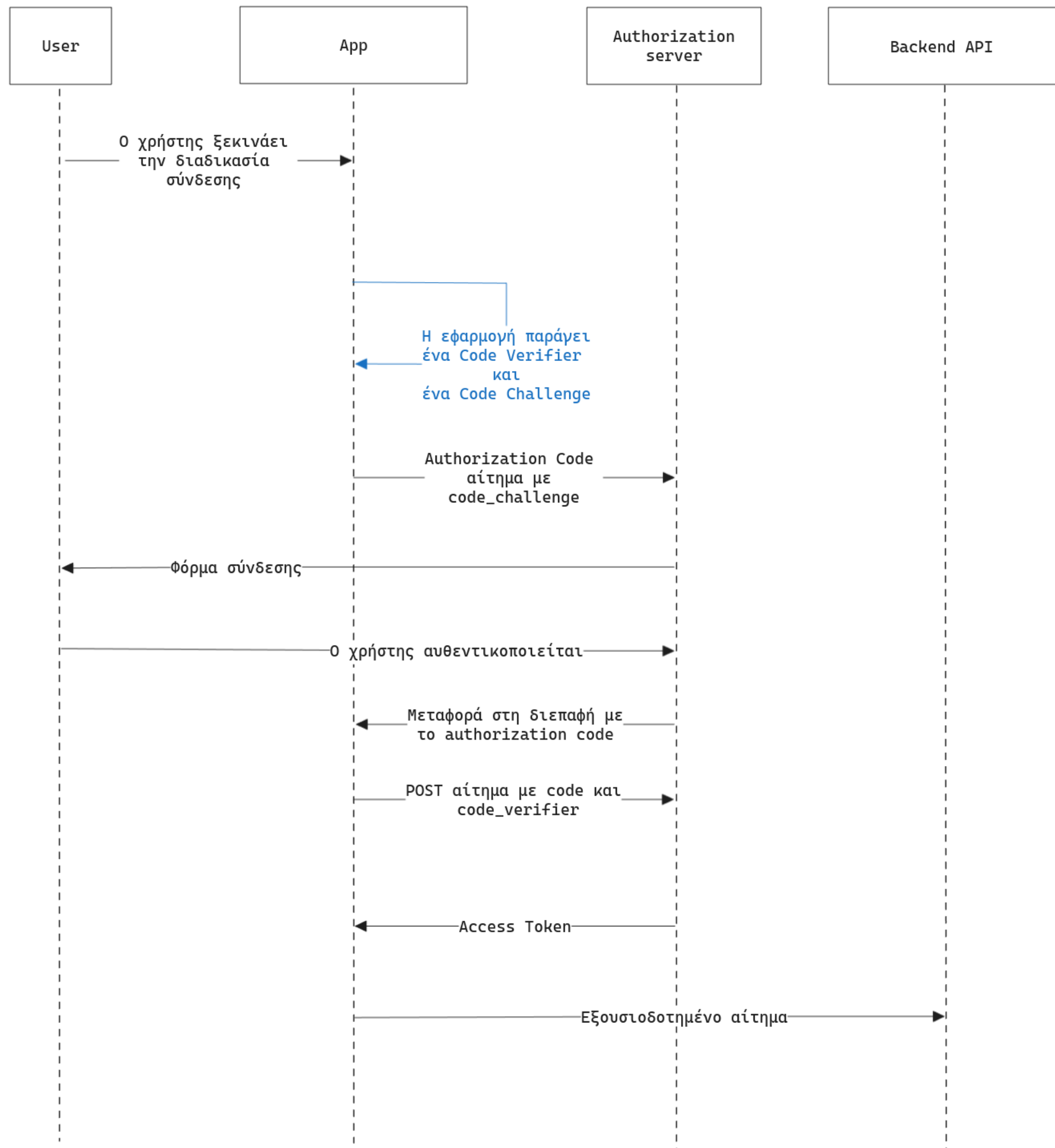
Το SAML είναι πολύ αποτελεσματικό στην αυθεντικοποίηση, όμως στην εξουσιοδότηση δεν συνιστάται έντονα. Παρόλο που μπορεί να υποστηρίξει πληροφορία μέσα από προσαρμογές του assertion και έτσι να προστεθούν πεδία με τιμές που εμείς θέλουμε, η αποστολή των ρόλων με αυτό τον τρόπο δεν είναι ιδανική επιλογή γιατί δεν μπορεί να υποστηρίξει τις απαιτήσεις μια πιο δυναμικής εξουσιοδότησης που αλλάζει συνεχώς. Το OAuth 2.0 είναι ένα πρωτόκολλο που μπορεί να βοηθήσει σε αυτό, διότι παρέχει ένα καλύτερο μηχανισμό για τη διαχείριση της πρόσβασης. Όταν μια εφαρμογή, εκ μέρους του χρήστη προσπαθεί να αποκτήσει πρόσβαση σε ορισμένους πόρους, δεν χρειάζεται τα διαπιστευτήρια του. Παρόμοια με το SAML, μεταβιβάζει τον χρήστη στον διακομιστή αυθεντικοποίησης, όπου και ο πρώτος αφού συμπληρώσει τους κωδικούς του, επιλέγει αν εξουσιοδοτεί την εφαρμογή που με τη σειρά της θα λάβει μία απάντηση που τέλος θα την ανταλλάξει με τον διακομιστή για ένα acccess token. Το code που θα ανταλλάξει η εφαρμογή για να πάρει ένα τόκεν έχει διάφορους τύπους.

- **Password**: Σύνδεση με κατευθείαν εισαγωγή των κωδικών του χρήστη.
- **Authorization Code**: Ο πιο σύνηθες πλέον τύπος για εφαρμογές.
- **Implicit**: Δεν χρησιμοποιείται πλέον διότι έχει θέματα ασφαλείας.
- **Client Credentials**: Ιδανικό για σύνδεση διακομιστή προς διακομιστή.

6.1.4.1 Επέκταση PCKE

Το PCKE (Proof Key for Code Exchange) flow είναι μια επέκταση του OAuth 2.0 πρωτοκόλλου, που σκοπό έχει να αντιμετωπίσει προβλήματα που εμφανίζονται σε εφαρμογές που δεν μπορούν να διαχειριστούν κλειδιά με ασφάλεια, κυρίως οι δυναμικές ιστοσελίδες

(SPA), οι οποίες εκτελούνται κυρίως στο πελάτη και όλος ο κώδικας της εφαρμογής μπορεί να γίνει προσβάσιμος από τον χρήστη. Αυτό καθιστά αδύνατο να ενσωματώσει κλειδιά στον κώδικά του διότι ακόμη και αν γίνει με κάποιο έμμεσο τρόπο, πάλι στην εκτέλεση θα γίνει εμφανής. Έτσι δεν μπορούμε να είμαστε σίγουροι ότι δεν έχει παρέμβει κάποιος τρίτος στη διαδικασία με σκοπό να υποκλέψει το authorization code και να βγάλει τόκεν εξουσιοδότησης με βάση τα στοιχεία του χρήστη. Η λύση δίνεται με την επέκταση PCKE όπου η εφαρμογή στην εκκίνηση της διαδικασίας αυθεντικοποίησης, παράγει ένα string υψηλής εντροπίας *code verifier* και έπειτα από αυτό, παράγει και ένα *code challege* το οποίο και στέλνει μαζί με την αίτηση για εξουσιοδότηση του χρήστη, στο διακομιστή που εκτελεί την αυθεντικοποίηση και εξουσιοδότηση. Έπειτα η εφαρμογή ανακατευθύνει το χρήστη στον διακομιστή για να βάλει τα στοιχεία του και με την επιτυχημένη σύνδεση στέλνει το authorization code στην εφαρμογή οπού και το στέλνει πίσω μαζί με το code verifier που είχε παράξει, λαμβάνει το τόκεν και πλέον μπορεί να κάνει εξουσιοδοτημένα αιτήματα σε resource διακομιστές, αντιμετωπίζοντας τον κίνδυνο κάποιος να έχει παρέμβει.



Σχήμα 6.2 Διάγραμμα PCKE

6.2 Πολιτικές δικτύου

Από προεπιλογή το μοντέλο δικτύωσης επιτρέπει σε όλα τα pods να επικοινωνούν μεταξύ τους χωρίς περιορισμούς, κάτι που αν σε μια αρχιτεκτονική δοθεί περισσότερο βάρος στην προστασία από εξωτερικά αιτήματα, αφήνει χώρους για προβλήματα, κατανοώντας τους

πιθανούς κινδύνους σε αυτό θα μας βοηθήσει να ακολουθήσουμε πρακτικές που θα τους ελαχιστοποιούν. Παρακάτω αναφέρουμε ορισμένους από αυτούς :

- Ο πιο σύνηθες κίνδυνος είναι ένας επιτιθέμενος να αποκτήσει πρόσβαση σε ένα pod που ενδεχομένως παρουσιάζει μια ευπάθεια είτε στο εσωτερικό του περιέκτη είτε στην εφαρμογή που εκτελεί και με βάση αυτό να επιχειρήσει να αποκτήσει πρόσβαση και σε άλλα pods εντός του cluster.
- Ακόμα όμως και χωρίς κάποια κακόβουλη χρήση, η έλλειψη τμηματοποίησης του εσωτερικού δικτύου μπορεί να οδηγήσει σε μη εξουσιοδοτημένη πρόσβαση.
- Σε μια συστοιχία ενός οργανισμού είναι σύνηθες να περιέχει πολλαπλά περιβάλλοντα είτε εξυπηρετώντας το κύκλο ανάπτυξης ενός λογισμικού δηλαδή περιβάλλοντα όπως παραγωγής, δοκιμών και ανάπτυξης αλλά άλλοτε και όταν εξυπηρετεί διαφορετικούς πελάτες, εκεί παρουσιάζεται κίνδυνος διασταύρωσης αυτών που μπορεί να οδηγήσει σε προβλήματα.

Οι πολιτικές δικτύου στο Kubernetes έχουν σχεδιαστεί για να αντιμετωπίζουν αυτές τις προκλήσεις ασφαλείας, επιτρέποντας να ελέγχουμε ποιο pod μπορεί να μιλήσει με ποιο. Στην ουσία, αυτή είναι και η εφαρμογή του λιγότερου προνομίου (least priveledge) περιορίζοντας τις επικοινωνίες μόνο εκεί που απαιτούνται. Αναφερθήκαμε ότι στην υλοποίηση μας θα χρησιμοποιήσουμε την CNI επέκταση (plugin) **Calico** η οποία θα μας βοηθήσει να αντιμετωπίσουμε τους παραπάνω κινδύνους. Πιο συγκεκριμένα προσφέρει:

- Απομόνωση σε επίπεδο namespaces και ξεχωριστά σε υπηρεσίες, χωρίζοντας την τοπολογία δικτύου με βάση και το περιβάλλον που θέλουμε
- Όρια επικοινωνίας στα pod.
- Καταγραφή της κίνησης του εσωτερικού δικτύου.

6.3 Προκλήσεις σε επίπεδο εφαρμογών

Έχουμε αναφερθεί στη σημασία της ασφάλειας του εσωτερικού δικτύο στη συστοιχία, το ίδιο σημαντικό όμως είναι και το επίπεδο εφαρμογών, καθώς η αρχιτεκτονική των μικρουπηρεσιών σχεδιασμένη με τρόπο να είναι ευέλικτη και προεκτάσιμη χωρίζοντας τις λειτουργίες μια εφαρμογής σε μικρότερες υπηρεσίες αυξάνει όμως έτσι και το περιθώριο λαθών αλλά και επιθέσεων. Η διαίρεση μίας εφαρμογής σε μικρότερα κομμάτια απαιτεί και την μεταξύ

τους επικοινωνία εκεί που χρειάζεται που ορισμένες φορές αυτή θα παρουσιάσει προβλήματα που σε μία συστοιχία που αποτελείται από πολλές μικρουπηρεσίες, η ανίχνευση αυτών των προβλημάτων μπορεί να γίνει πολύ δύσκολη. Επίσης, στους πλέον πολύ γρήγορους ρυθμούς της παράδοσης λογισμικού, παρουσιάζονται προβλήματα όταν θέλουμε να εισάγουμε νέες λειτουργίες σε μία υφιστάμενη εφαρμογή με τρόπο ομαλό που σε περίπτωση λαθών να μην επηρεάσουμε όλους τους χρήστες μας. Προβλήματα μπορούν να παρουσιαστούν επίσης αν δεν ελέγξουμε τον ρυθμό των αιτημάτων στις υπηρεσίες μας καθώς σε περίπτωση υπερφόρτωσης μπορούν να τις καταστήσουν μη διαθέσιμες. Τέλος, η επικοινωνία μεταξύ υπηρεσιών που συνήθως γίνεται μέσω TCP πρωτοκόλλων όπως HTTP, HTTPS, gRCP, θα πρέπει να είμαστε σίγουροι ότι είναι ασφαλής και δεν έχει υπάρξει κάποιος τρίτος που να έχει αλλοιώσει ή υποκλέψει δεδομένα (MITM Attack).

6.3.1 Το πλέγμα του Istio

Μια από τις πλέον καθιερωμένες τεχνολογίες στην αρχιτεκτονική μικρουπηρεσιών είναι το Istio όπου είναι ένα πλέγμα συστοιχίας που συμπληρώνει το Kubernetes και δίνει λύση στα παραπάνω με την δυνατότητα να δρα σαν μεσάζοντας μεταξύ υπηρεσιών και να μπορεί με αυτό τον τρόπο να προσθέτει επιπλέον λειτουργίες που θα ήταν αδύνατο να υλοποιηθούν μονομερώς σε κάθε μικρουπηρεσία.

Data Plane

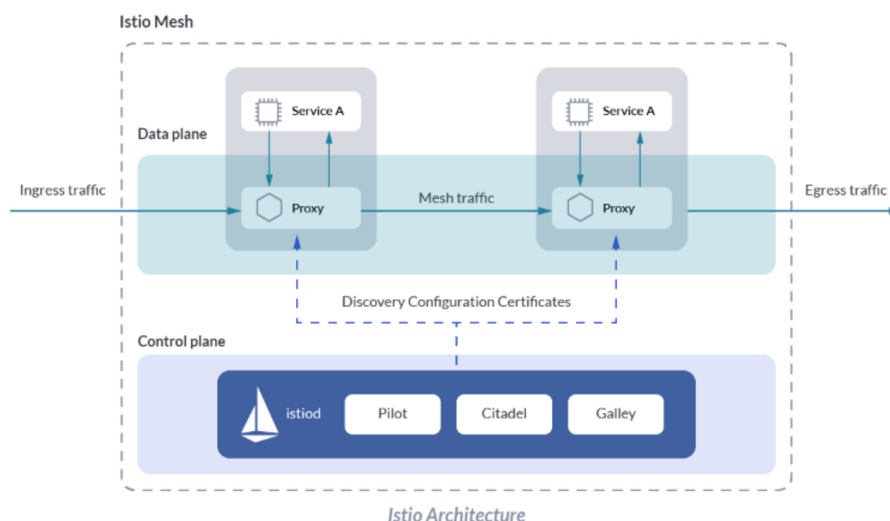
Το επίπεδο δεδομένων(data plane) στο Istio είναι κυρίως υπεύθυνο για το χειρισμό της κίνησης (αιτήματα και απαντήσεις) μεταξύ των υπηρεσιών στο πλέγμα υπηρεσιών. Εδώ πραγματοποιείται η δρομολόγηση, η εξισορρόπηση φορτίου και η επιβολή διάφορων κανόνων που έχουμε δημιουργήσει. Το βασικότερο στοιχείο του επιπέδου δεδομένων του Istio είναι τα Envoy, τα οποίοι αναπτύσσονται ως sidecar συμπληρωματικοί περιέκτες σε κάθε pod που έχουμε δημιουργήσει. Εμείς το μόνο που έχουμε να κάνουμε είναι να τα ενσωματώσουμε μέσω της εντολής

```
kubectl label namespace backend istio-injection=enabled
```

Control Plane

Το επίπεδο ελέγχου (control plane) είναι ο πυρήνας διαχείρισης που προσφέρει το Istio και σε αυτό περιέχεται η υπηρεσία istiod που την ολοκληρώνουν τρία οντότητες με διαφορετικά καθήκοντα η κάθε μια. Η πρώτη οντότητα είναι το Pilot όπου είναι το βασικότερο κομμάτι στο καθορισμό της διαχείρισης κίνησης του πλέγματος, είναι πάντα ενήμερο στο που βρίσκονται οι υπηρεσίες ώστε να μπορεί να κατευθύνει και τα αιτήματα που εισέρχονται στο πλέγμα αλλά και να τα κατανέμει με βάση το υφιστάμενο φόρτο που βρίσκεται στις υπηρεσίες, προσφέροντας την δυνατότητα διαμόρφωσης της κίνησης μέσα από κανόνες που του

έχουμε ορίσει του οποίους και ακολουθάνε τα envoy proxies. Η δεύτερη οντότητα είναι το *Cidatel* το οποίο έχει τον ρόλο μιας αρχής πιστοποίησης (Certificate Authority) και είναι υπεύθυνο για την επικύρωση και την διανομή των πιστοποιητικών που το ίδιο δημιουργεί, προσφέροντας στις υπηρεσίες που τη χρησιμοποιούν ένα ασφαλές κανάλι επικοινωνίας. Όταν μια νέα υπηρεσία εγκατασταθεί στο πλέγμα της συστοιχίας, το Cidatel θα δημιουργήσει ένα νέο πιστοποιητικό και ένα ιδιωτικό κλειδί και μέσω μιας υπηρεσίας ανακάλυψης (Secret Discovery Service) θα τα διανείμει στο envoy proxy κάθε υπηρεσίας έτσι ώστε κάθε υπηρεσία που θέλει να επικοινωνήσει με μια άλλη να ξεκινήσει την χειραψία TLS. Η τρίτη οντότητα είναι το *Galley*, υπεύθυνο για την επικύρωση των διαμορφώσεων που καθορίζουμε στην συστοιχία μέσα από αρχεία YAML, αφού τα επεξεργαστεί και τα ελέγξει ότι είναι σωστά, τα μετατρέπει με βάση συγκεκριμένων προτύπων ώστε να υπάρχει μια κοινή τυποποίηση και με τη σειρά του να τα διαχειριστεί το pilot και τα envoy proxies.



Σχήμα 6.3 Αρχιτεκτονική Istio[2]

6.4 Προκλήσεις στη διαχείριση μυστικών κλειδιών

Στη αρχιτεκτονική μικρουπηρεσιών η ανάγκη για πολλαπλά μυστικά κλειδιά (secrets) είναι τεράστια καθώς κάθε κομμάτι μιας υλοποίησης περιέχει τα δικά του, είτε έχει να κάνει με κωδικούς βάσης δεδομένων, API tokens, κωδικούς για εφαρμογές που κάνουμε χρήση ή και πιστοποιητικά SSL/TLS, ο όγκος μονίμως αυξάνεται και κάνει τη διαχείρισή τους μεγάλο

[2]Πηγή: https://sysdig.com/blog/monitor-istio/

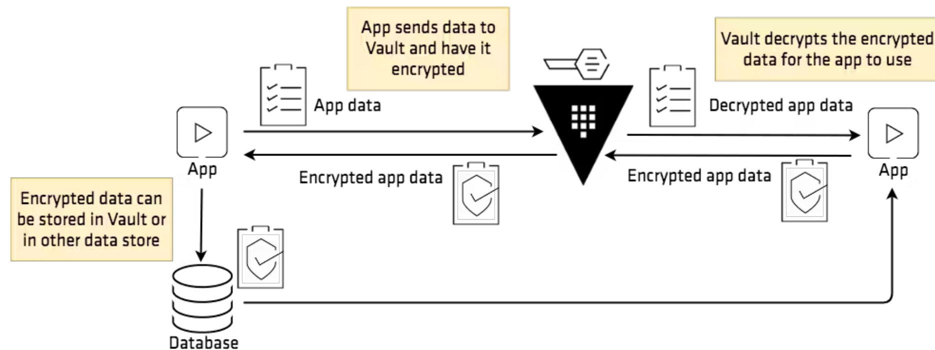
πρόβλημα στο κύκλο της ανάπτυξης λογισμικού. Το πρόβλημα των κλειδιών μπορεί να γίνει και σοβαρή ευπάθεια όταν τύχει από λάθος να ενσωματωθούν σε κάποιο κομμάτι κώδικα και έπειτα αυτό να ανεβεί σε κάποιο σύστημα ελέγχου εκδόσεων (Github, Gitlab) που ακόμα και αν είναι ιδιωτικό, εφόσον δεν βρίσκεται κάτω από δικό μας διακομιστή σημαίνει ότι έχουμε διαρρεύσει κλειδιά σε μια εξωτερική αρχή εκτός του οργανισμού μας, αλλά θα υπάρξουν περιπτώσεις που θα χρειαστεί στην διαδικασία των pipeline συνεχούς ολοκλήρωσης/ συνεχούς παράδοσης, να πρέπει να ενσωματώσουμε και εκεί κωδικούς που αναγκαία θα ανέβει σε κάποιο repository και οι συμβατικές λύσεις δεν αρκούν. Η περίπτωση να έχει διαρρεύσει ένα μυστικό κλειδί σημαίνει ότι θα πρέπει να ξεκινήσουμε διαδικασίες να αλλάξουμε κωδικούς με τον κίνδυνο να βλάψουμε την διαθεσιμότητα των υπηρεσιών μας που βρίσκονται στην παραγωγή. Μια επίσης μεγάλη πρόκληση στη διαχείριση που κάνουμε είναι στο ποιος έχει πρόσβαση στα κλειδιά διότι σε κάθε ομάδα που αποτελείται από διάφορους ρόλους και συνήθως υποομάδες, θα πρέπει να δίνεται βάση στο εάν κάποιος χρειάζεται να έχει πρόσβαση στο εκάστοτε κλειδί που ανήκει στο πρότζεκτ.

Το Kubernetes προσφέρει λειτουργίες αποθήκευσης μυστικών κλειδιών, παρόλα αυτά, δεν συνιστάται για ανάγκες όπου χρειαζόμαστε ισχυρή κρυπτογράφηση, κάτι που δεν προσφέρει όπως αδύναμο είναι και στη κατηγοριοποίηση των κλειδιών και αν ένας χρήστης μπορεί να πρόσβαση ή όχι.

6.4.1 Hashicorp Vault

Το Hashicorp Vault είναι ένα εργαλείο που λειτουργεί σαν βάση δεδομένων για κλειδιά σε μορφή κλειδιού/τιμής τα οποία αποθηκεύονται κρυπτογραφημένα. Υποστηρίζει δυναμικά κλειδιά που μπορούν να αλλάζουν είτε προγραμματισμένα είτε όταν έχει λήξει η διάρκεια τους. Προσφέρει λεπτομερή διαχείριση της πρόσβασης και εξουσιοδότησης σε αυτά μέσα από πολιτικές και ρόλους που ορίζουμε εμείς. Μπορεί να ενσωματωθεί σε ποικίλες τεχνολογίες και συνδέεται με ευκολία στις υπηρεσίες που προσφέρουν οι πάροχοι νέφους.

Επίσης προσφέρει το Transit Engine, το οποίο είναι μια κρυπτογράφηση σαν υπηρεσία **(Encryption as a Service, EaaS)** η οποία κρυπτογραφεί και αποκρυπτογραφεί δεδομένα χωρίς να χρειάζεται να ορίσουμε κάπου το κλειδί της κρυπτογράφησης στον κώδικά μας, παρά μόνο να το δημιουργήσουμε στο vault.



Σχήμα 6.4 Hashicorp Vault Transit Engine[3]

6.5 Ευπάθειες στις εικόνες και λανθασμένες ρυθμίσεις

Στο ευέλικτο περιβάλλον του νέφους η παρακολούθηση και διαχείριση του μεγάλου αριθμού των μικρουπηρεσιών είναι ένα δύσκολο έργο υπολογίζοντας τις απαιτήσεις που μπορεί να έχει η κάθε μια για τις δικές της διαμορφώσεις στα αρχεία YAML. Το σχέδιο για μια πιο στοχευμένη διαμόρφωση αυτών των αρχείων ακολουθώντας κάποια τυποποίηση που έχει αποφασιστεί από μια ομάδα που ασχολείται με την διαδικασία εγκατάστασης των υπηρεσιών στην συστοιχία είναι αναγκαίο. Επίσης, σε στην ανάπτυξη λογισμικού μιας cloud native εφαρμογής ή υπηρεσίας όπως έχουμε αναφερθεί αρκετές φορές, είναι πολύ σίγουρο ότι θα χρησιμοποιηθεί κάποια υπάρχων βιβλιοθήκη, κάποια βάση δεδομένων και γενικά κάτι που δεν έχει φτιαχτεί από την αρχή από εμάς. Αν η δικιά μας υλοποίηση είναι πιθανό να έχει ορισμένα λάθη τότε το ίδιο ισχύει και τα κομμάτια που ενσωματώνουμε στο κώδικα μας από άλλους προγραμματιστές. Η πιο σωστή πρακτική για την αντιμετώπιση των παραπάνω προκλήσεων είναι να ενσωματώσουμε ελέγχους στην διαδικασία συνεχούς ολοκλήρωσης/συνεχούς παράδοσης(CI/CD) που θα βρίσκουν έγκαιρα λάθη στις ρυθμίσεις που έχουμε υλοποιήσει καθώς και ευπάθειες που έχουν διαγνωστεί σε βιβλιοθήκες που έχουμε χρησιμοποιήσει. Ένα εργαλείο για την διατήρηση σωστών αρχείων διαμόρφωσης είναι το KubeLinter [12] το οποίο αναλύει τα αρχεία YAML και τα Helm Charts με βάση τις βέλτιστες πρακτικές και μας ειδοποιεί για τυχόν διαμορφώσεις που μπορούν να προκαλέσουν ζητήματα ασφαλείας και αξιοπιστίας με ορισμένα από αυτά να είναι η μή τήρηση της αρχής του ελάχιστου προνομίου καθώς και την εκτέλεση περιεκτών με root δικαιώματα. Ακόμα ένα πολύτιμο εργαλείο είναι το Trivy [14] που σκανάρει ευπάθειες (CVEs) που μπορεί να βρίσκονται στις εικόνες που χρησιμοποιούμε. Το Trivy χρησιμοποιεί διάφορες δημόσιες βάσεις δεδομένων όπως τη National Vulnerability Database [8] και είναι πάντα ενήμερο για

[3]Πηγή: https://developer.hashicorp.com/vault/tutorials/encryption-as-a-service/eaas-transit

τις ευπάθειες που έχουν ανακαλυφτεί πρόσφατα, προσφέροντας στους προγραμματιστές έγκαιρα τρόπο ώστε να αντιμετωπίσουν τις ευπάθειες που ανιχνεύθηκαν.

Κεφάλαιο 7

Υλοποίηση

Η υλοποίηση θα βασίζεται στη ανάλυση που κάναμε στο προηγούμενο κεφάλαιο δοκιμάζοντας εκτενώς τις διάφορες λύσεις που προτείναμε, προσεγγίζοντας μία αρχιτεκτονική δικτύου μηδενικής εμπιστοσύνης **(Zero Trust Network Architecture)** θα υλοποιήσουμε μία εφαρμογή για ασφαλής αποθήκευση αρχείων όπου εφαρμόσαμε τις βέλτιστες πρακτικές για αυτές τις περιπτώσεις.

7.1 Προστασία σε επίπεδο δικτύου

Ξεκινώντας από το χαμηλό επίπεδο του δικτύου που θα υλοποιήσουμε στη συστοιχία, καθορίζοντας ένα σύνολο από πολιτικές δικτύου (network policies) θα προστατεύεται η κίνηση από pod σε pod. Για να αναδείξουμε καλύτερα τις δυνατότητες των πολιτικών δικτύου, διαιρέσαμε το δίκτυο σε ξεχωριστά namespaces που σχετίζονται με την λογική μιας εφαρμογής και όχι με το περιβάλλον (ανάπτυξης, δοκιμών, παραγωγής) που είναι μια σύνηθες προσέγγιση στο κύκλο της ανάπτυξης λογισμικού.

Οπότε δημιουργήσαμε τα εξής namespaces:

- **frontend**: Περιέχει την διεπαφή που θα αλληλεπιδρά ο χρήστης.
- **backend**: Περιέχει την μικρουπηρεσία που θα αποθηκεύει στη βάση .
- **data**: Περιέχει τη βάση της εφαρμογής του backend και τη βάση του Keycloak.
- **iam**: Περιέχει την εφαρμογή Keycloak.
- **istio-system**: Περιέχει τον API gateway του Istio.

- **vault**: Περιέχει τους μηχανισμούς του Hashicorp Vault.

Ακολουθώντας την προσέγγιση του ελάχιστου προνομίου, καθορίσαμε τις απαιτήσεις της επικοινωνίας των pod στα namespaces όπως φαίνεται και στον παρακάτω πίνακα:

Πίνακας 7.1 Επιτρεπόμενες προσβάσεις των μικρουπηρεσιών

	frontend	backend	data	iam	istio-system	vault
frontend	-				*	
backend		-	*		*	*
data			-		*	
iam				-	*	
istio-system	*	*	*	*	-	*
vault					*	-

Αυτό το πετυχαίνουμε μέσα από συγκεκριμένες πολιτικές δικτύου οι οποίες είναι πόροι του Calico αλλά η βάση τους έρχεται από το Kubernetes, στην παρακάτω πολιτική βλέπουμε ότι επιτρέπουμε τη κίνηση από τα deployments του backend στα pods του data σε συγκεκριμένο port, οποιαδήποτε άλλη κίνηση απορρίπτεται.

```yaml
apiVersion: crd.projectcalico.org/v1
kind: NetworkPolicy
metadata:
  name: allow-backend-to-data
  namespace: data
spec:
  order: 100
  selector: all()
  ingress:
    - action: Allow
      source:
        namespaceSelector: 'projectcalico.org/name == "backend"'
  egress:
   - action: Allow
```

Κώδικας 7.1 Network policy για το backend

Το ίδιο κάναμε και με τα υπόλοιπα namespaces. Παρόλα αυτά, ακολουθώντας την προσέγγιση του ελάχιστου προνομίου στην εφαρμογή της διεπαφής δεν δώσαμε πρόσβαση δικτύου σε ολόκληρο το namespace του iam αλλά μόνο στο keycloak, το ίδιο και με το keycloak οπού του επιτρέπουμε να έχει πρόσβαση μόνο στη βάση δεδομένων του και όχι

και σε αυτή της εφαρμογής των αρχείων. Παρακάτω βλέπουμε με την χρήση ping στην pod ip της postgres, επιχειρούμε να επικοινωνήσουμε από frontend pod στη postgres

```
Pods(data)[1]
NAME↑          PF   READY   STATUS    RESTARTS IP
postgresql-0   ●    2/2     Running          6 10.244.226.169
```

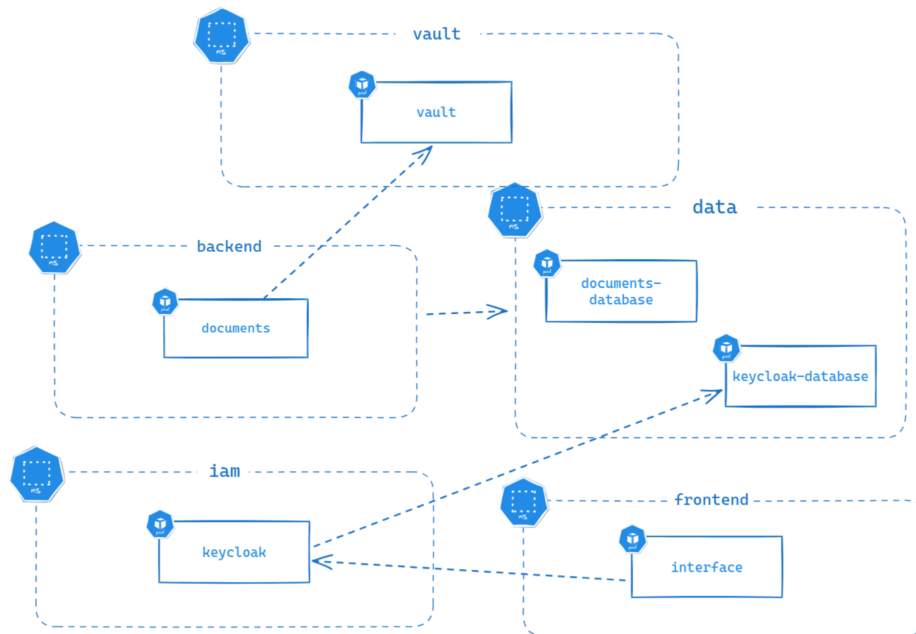
Παρατηρούμε ότι δεν μπορούμε

```
<<K9s-Shell>> Pod: frontend/frontend-cc6865f7b-rhwf4 | Container: frontend
/ # ping 10.244.226.169
PING 10.244.226.169 (10.244.226.169): 56 data bytes
^C
--- 10.244.226.169 ping statistics ---
7 packets transmitted, 0 packets received, 100% packet loss
```

Ενώ από pod του backend namespace η επικοινωνία λειτουργεί.

```
<<K9s-Shell>> Pod: backend/document-management-846f795dfd-2t46x | Container: document-management
~ # ping 10.244.226.169
PING 10.244.226.169 (10.244.226.169): 56 data bytes
64 bytes from 10.244.226.169: seq=0 ttl=63 time=0.069 ms
64 bytes from 10.244.226.169: seq=1 ttl=63 time=0.135 ms
64 bytes from 10.244.226.169: seq=2 ttl=63 time=0.129 ms
^C
--- 10.244.226.169 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.069/0.111/0.135 ms
```

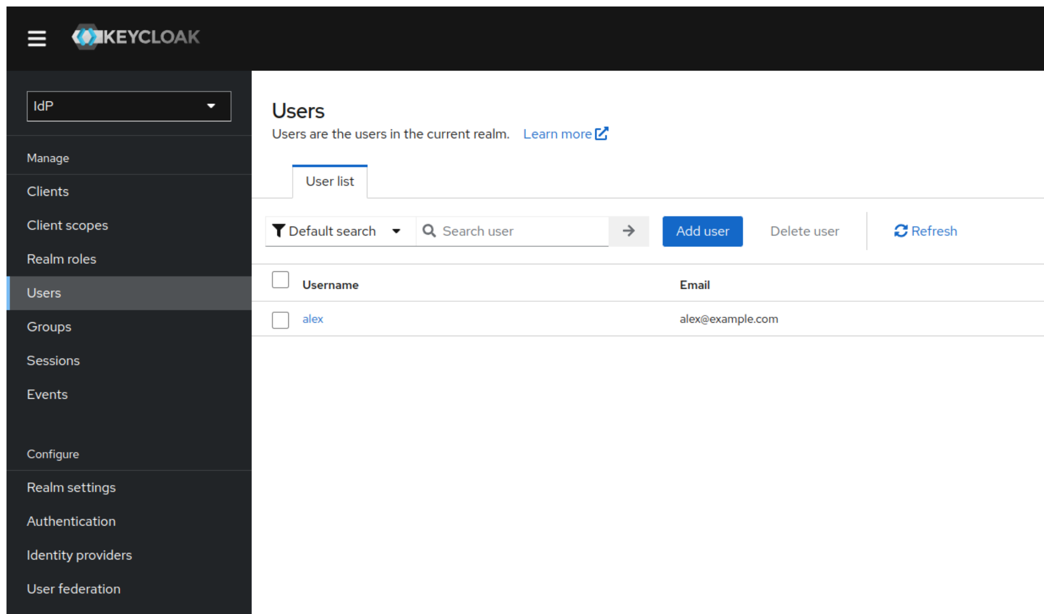
Έτσι με αυτό το τρόπο σαν πρώτο βήμα απομονώσαμε το εσωτερικό δίκτυο με την επικοινωνία να είναι περιορισμένη εκεί που χρειάζεται. Ακολουθεί μια απεικόνιση του εσωτερικού δικτύου της συστοιχίας.



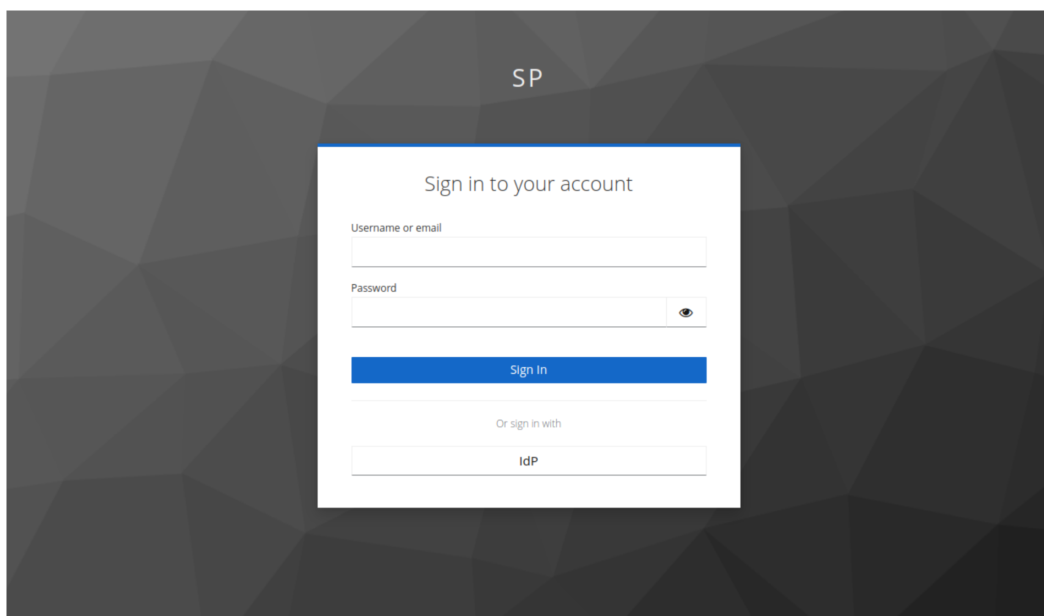
Σχήμα 7.1 Αναπαράσταση εσωτερικού δικτύου συστοιχίας

7.2 Σύστημα αυθεντικοποίησης και εξουσιοδότησης

Για την σωστή ανάδειξη των πρωτοκόλλων που μελετήσαμε θα δημιουργήσουμε στο Keycloak δύο realms τα οποίο το ένα θα δρα σαν πάροχος ταυτότητας και το άλλο σαν ένας πάροχος υπηρεσίας. Τα realms είναι χώροι του Keycloak οι οποίοι είναι απομονωμένοι μεταξύ τους και κάθε realm έχει ξεχωριστούς χρήστες και πάροχους. Ο descriptor είναι ένα αρχείο σε μορφή XML που βρίσκεται στα SAML μεταδεδομένα, περιγράφει όλες τις λεπτομέρειες που χρειάζεται για την εγκαθίδρυση της επικοινωνίας μεταξύ δύο πλευρών. Με αυτό λοιπόν το αρχείο του παρόχου ταυτότητας θα ξεκινήσει η εγκαθίδρυση περνώντας το στον πάροχο υπηρεσίας, έπειτα παρομοίως περνάμε ένα πελάτη (client) με τα medatata του πάροχου υπηρεσίας στον πάροχο ταυτότητας.



Φτιάχνοντας το πρώτο μας χρήστη, δοκιμάσαμε να συνδεθούμε στο Realm του πάροχου υπηρεσίας. Παρατηρούμε ότι πλέον έχουμε την επιλογή να συνδεθούμε με κάποιον εξωτερικό πάροχο.



Σχήμα 7.2 Σύνδεση μέσω εξωτερικού πάροχου

Μόλις επιλέξουμε τον εξωτερικό πάροχο, θα μεταβούμε στον IdP και θα σταλθεί ένα SAML αίτημα από τον SP στον IdP. Έπειτα βάζοντας τα σωστά διαπιστευτήρια θα σταλθεί πίσω στο SP ένα SAML Response που θα περιέχει και την αυθεντικοποίηση του χρήστη.

Path	Method	Status
http://localhost:8080/realms/IdP/protocol/saml	POST	302
http://localhost:8080/realms/SP/broker/IdP/endpoint	POST	302

SAML Request Response

Response Data
status
statusText
httpVersion
redirectURL
headersSize

Έτσι μπορούμε να έχουμε χρήστες από έναν εξωτερικό πάροχο στην εφαρμογή μας χωρίς να τον αναγκάζουμε να δημιουργήσει νέο λογαριασμό με την χρήση του πρωτοκόλλου SAML και ταυτόχρονα να εξουσιοδοτούμε τους χρήστες με το πρωτόκολλο OAuth2. Έπειτα φτιάξαμε έναν νέο Keycloak OIDC client στο realm του παρόχου υπηρεσίας το οποίο θα χρησιμοποιούν οι frontend μικρουπηρεσίες για να αιτούνται νέα τόκενς εφόσον ο χρήστης δώσει τα σωστά διαπιστευτήρια. Το μόνο που απομένει είναι να φτιάξουμε ορισμένους ρόλους που θα ενσωματωθούν στο τόκεν και έπειτα θα χρησιμοποιούνται από τις υπηρεσίες backend για έλεγχο πρόσβασης.

Role name	Composite
Admin	False
Editor	False
Viewer	False

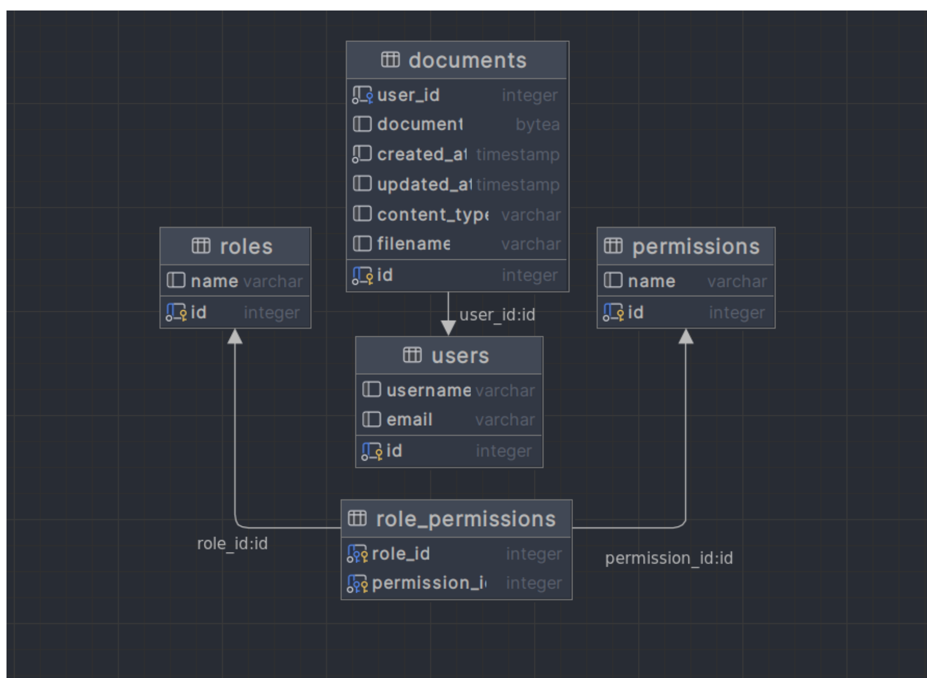
7.3 Έλεγχος πρόσβασης βάση ρόλων

Στην συνέχεια, υλοποιήσαμε με χρήση της γλώσσας προγραμματισμού Golang ένα REST API backend που περιέχει τα εξής endpoints:

- **GET** /documents: Επιστρέφει τη λίστα με τα αρχεία που έχουν ανέβει στη εφαρμογή.
- **GET** /document/id: Ο χρήστης έχει την δυνατότητα για λήψη του αποκρυπτογραφημένο αρχείου.
- **POST** /document/id: Ο χρήστης ανεβάζει το αρχείο, αυτό κρυπτογραφείτε και αποθηκεύεται στην βάση.

- **PUT** /document/id: Ο χρήστης ανεβάζει νέα έκδοση του αρχείου και η παλιά διαγράφεται.
- **DELETE** /document/id: Ο χρήστη έχει την δυνατότητα διαγραφής του αρχείου.

Το API χρησιμοποιεί τους ρόλους (RBAC) μέσω του OAuth 2.0 για να παίρνει αποφάσεις σχετικά με την εξουσιοδότηση που θα επιτρέπει, με βάση πάντα τον ρόλο που έχει ο χρήστης που κάνει και το αίτημα. Στο namespace data όπου έχουμε τις βάσεις δεδομένων δημιουργήσαμε μια βάση δεδομένων Postgres η οποία έχει το εξής schema:



Σχήμα 7.3 Βάση δεδομένων υλοποίησης

Έτσι με την συνάρτηση *validateToken* και την χρήση του δημόσιου κλειδιού που έχει χρησιμοποιήσει το Keycloak, επικυρώνουμε το τόκεν και εξάγουμε τα claims που περιέχουν και τους ρόλους.

```
func validateToken(tokenString string) (*CustomClaims, error) {
  cnf := config.Get()
  publicKey, err := parsePublicKey(cnf.PublicKey)
  if err != nil {
    return nil, err
  }
  token, err := jwt.ParseWithClaims(tokenString, &CustomClaims{}, func(token *jwt.
      Token) (interface{}, error) {
    _, ok := token.Method.(*jwt.SigningMethodRSA)
    if !ok {
      return nil, err
```

```
    }
    return publicKey, nil
  })
  if err != nil {
    return nil, err
  }
  if claims, ok := token.Claims.(*CustomClaims); ok && token.Valid {
    return claims, nil

  return nil, err
}
```

Οι ρόλοι φαίνονται παρακάτω:

- **Admin**: Ο ρόλος του διαχειριστή μπορεί να κάνει τα πάντα από δημιουργία, αλλαγή και διαγραφή όλων των αρχείων.
- **Editor**: Μπορεί να δημιουργήσει και να επεξεργαστεί αρχεία αλλά δεν μπορεί να διαγράψει αρχεία.
- **Viewer**: Ο viewer μπορεί να δει την λίστα με τα αρχεία και να κατεβάσει το αποκρυπτογραφημένο αρχείο που έχει δημιουργήσει.

Για το παραπάνω χρειαστήκαμε ένα middleware το οποίο είναι μια συνάρτηση που καλείται κάθε φορά που θέλουμε να κάνουμε ένα αίτημα που απαιτεί αυθεντικοποίηση και εξουσιοδότηση, το τόκεν μπαίνει σαν header (Bearer) και αφού επικυρωθεί ότι είναι απολύτως έγκυρο, γίνεται ένας έλεγχος αν ο χρήστης υπάρχει στη βάση δεδομένων ακολουθώντας πάντα τη μοναδικότητα του email, αν δε υπάρχει τότε γίνεται μια εγγραφή στη βάση και συνεχίζει με το να εξάγει τα claims, που περιέχει τον ρόλο που του έχουμε αναθέσει, στην δική μας υλοποίηση οι ρόλοι δένονται με συγκεκριμένες άδειες που έχει ο χρήστης στην εκάστοτε εφαρμογή ή μικρουπηρεσία, με αυτό το τρόπο αποκτάμε ευελιξία ώστε οι ρόλοι να μπορούν να είναι δυναμικοί και διαχειρίσιμοι από μία κεντρική αρχή και ταυτόχρονα για κάθε εφαρμογή να έχουν την δική τους σημασία. Έτσι, για λόγους ασφαλείας, πετυχαίνουμε να μην συμπεριλάβουμε πολύ πληροφορία στο τόκεν παρά μόνο την απαραίτητη που χρειάζεται. Στη παρακάτω συνάρτηση βλέπουμε ότι δημιουργήσαμε ένα γκρούπ από endpoints που χρειάζονται εξουσιοδότηση μέσω του middleware.

```
func RegisterRoutes(router *gin.Engine, pg *database.Postgres) {
  logger.Info("Register Routes")

  authGroup := router.Group("/", middleware.AuthMiddleware(pg))
  {
    authGroup.GET("/documents", func(ctx *gin.Context) {
      handlers.GetDocumentList(ctx, pg)
    })
```

```
        authGroup.POST("/document", func(ctx *gin.Context) {
          handlers.UploadDocument(ctx, pg)
        })

        authGroup.GET("/document/:id", func(ctx *gin.Context) {
          handlers.DownloadDocument(ctx, pg)
        })

        authGroup.PUT("/document/:id", func(ctx *gin.Context) {
          handlers.ReUploadDocument(ctx, pg)
        })

        authGroup.DELETE("/document/:id", func(ctx *gin.Context) {
          handlers.DeleteDocument(ctx, pg)
        })
      }
    }
```

7.4 Χρήση του Hashicorp Vault για πρόσβαση σε μυστικά κλειδιά

Με τη χρήση Helm Charts δημιουργήσαμε τοπικά την Hashicorp Vault υπηρεσία η οποία αποτελείται από ένα injector pod και ορισμένα vault pods, με το πρώτο να ενσωματώνει τα μυστικά κλειδιά στα pods που του έχουμε ρυθμίσει λειτουργώντας σαν sidecar [13]. Τα vault pods είναι και αυτά που διαχειρίζονται τα κλειδιά και την πρόσβαση σε αυτά. Εφόσον τα αρχικοποιήσαμε, έπειτα έγινε και η αποσφράγιση τους με την χρήση ενός root key ώστε να μπορούν να αποκρυπτογραφούν το περιεχόμενο που βρίσκεται σε αυτά.

Pods(vault)[4]

NAME↑	PF	READY	STATUS	RESTARTS	IP	NODE	AGE
vault-0	●	2/2	Running	0	10.244.226.135	minikube2	16h
vault-1	●	2/2	Running	0	10.244.226.139	minikube2	16h
vault-2	●	2/2	Running	0	10.244.226.190	minikube2	16h
vault-agent-injector-7f7f68d457-5jkf6	●	2/2	Running	0	10.244.226.138	minikube2	16h

Η πρόσβαση επιτυγχάνεται με την αυθεντικοποίηση του Kubernetes χρησιμοποιώντας ένα service account το οποίο έχει συνδεθεί με το ρόλο του vault και τις πολιτικές που του καθορίζουν την πρόσβαση. Έτσι μπορούμε να εισάγουμε μυστικά κλειδία στις εφαρμογές μας μέσω helm charts χωρίς να εξαρτόμαστε από τα λιγότερο ασφαλής Kubernetes secrets.

```
vault write auth/kubernetes/role/document-management \
bound_service_account_names=document-management-service-account \
bound_service_account_namespaces=backend \
policies=document-management \
ttl=1h
```

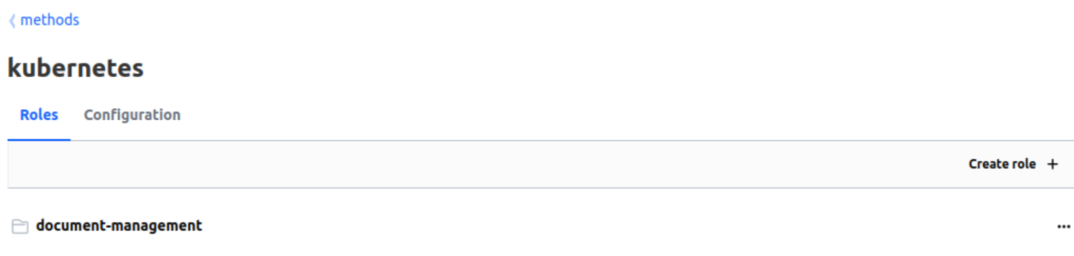
Με το παρακάτω τρόπο μπορούμε να τα ενσωματώνουμε στο deployment YAML μαζί με το serviceAccount που έχου δημιουργίσει για να γίνεται το Kubernetes authentication

```
vault.hashicorp.com/agent-inject: "true"
vault.hashicorp.com/role: "document-management"
vault.hashicorp.com/agent-inject-secret-database-config: "secret/data/document-management/config"
```

έπειτα να είναι διαθέσιμο στο

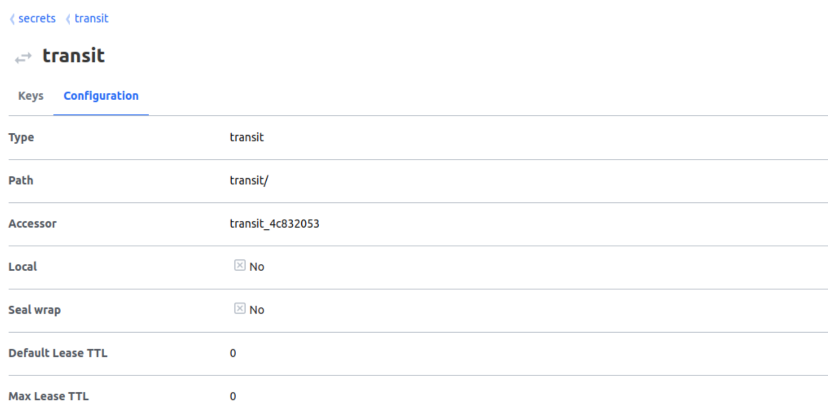
```
/vault/secrets/database-config
```

που θα μπορεί να αντλείται κατευθείαν από το pod της εφαρμογής.



7.4.1 Δημιουργία κρυπτογραφικού κλειδιού

Στη περίπτωση του Transit Engine απαιτείτε ένας διαφορετικό τρόπος, συνδεθήκαμε στη διεπαφή του vault και δημιουργήσαμε ένα νέο κρυπτογραφικό κλειδί που χρησιμοποιείται από την εφαρμογή για κρυπτογράφηση και αποκρυπτογράφηση των αρχείων. Η αυθεντικοποίηση γίνεται πάλι μέσω service account αλλά δεν ενσωματώνουμε κάποιο κλειδί αλλά η εφαρμογή που θα κάνει χρήση της κρυπτογράφησης/αποκρυπτογράφησης, επικοινωνεί μέσω API.



Στην περίπτωσή μας, χρησιμοποιήσαμε το επίσημο SDK του Hashicorp Vault (*github.com/hashicorp/vault/api*) για την γλώσσα προγραμματισμού Golang. Δημιουργούμε ένα client που θα διαχειρίζεται τα αιτήματα στο vault.

```
func InitializeVaultClient() (*api.Client, error) {
  cfg := config.Get()
  conf := vault.DefaultConfig()
  conf.Address = "http://vault.vault:8200"

  client, err := vault.NewClient(conf)
  if err != nil {
    return nil, fmt.Errorf("unable to initialize Vault client: %w", err)
  }

  k8sAuth, err := auth.NewKubernetesAuth(
  cfg.VaultRoleName,
  auth.WithServiceAccountTokenPath("/var/run/secrets/kubernetes.io/serviceaccount/
      token"),
  )
```

Έπειτα μέσω του client και τη χρήση της συνάρτησης Logical().Write κρυπτογραφούμε/αποκρυπτογραφούμε αρχεία και αποθηκεύουμε το κρυπτογραφημένο κείμενο στη βάση.

```
  secret, err := client.Logical().Write(path, data)
if err != nil {
  logger.Error("error encrypting data", "error", err)
  return "", err
}
```

7.5 Διαχείριση των υπηρεσιών

Η διεπαφή ξεκινάει ένα αίτημα εξουσιοδότησης και ανακατευθύνει τον χρήστη τον χρήστη στο keycloak ώστε να συμπληρώσει τα στοιχεία του, με τη σειρά του το keycloak ανακατευθύνει το χρήστη πίσω στην εφαρμογή μαζί με το authorization code το οποίο και θα χρησιμοποιεί η εφαρμογή (για όσο διαρκεί το session) για την εξουσιοδότηση στην backend εφαρμογή, αυτό σημαίνει ότι τόσο η διεπαφή αλλά και υπηρεσία του keycloak θα πρέπει να είναι προσπελάσιμη και εκτός συστοιχίας, για αυτό το λόγο θα χρειαστούμε ένα Ingress Gateway και τα ανάλογα VirtualService που θα κατευθύνουν την κίνηση εσωτερικά της συστοιχίας. Επίσης, θα χρειαστούμε ένα πιστοποιητικό ώστε η Ingress να μπορεί να υπογράφει με βάση το ιδιωτικό κλειδί πιστοποιώντας την εγκυρότητα της web εφαρμογής (χειραψία TLS). Αφού το εκδώσουμε θα το εισάγουμε στη συστοιχία μας ως secret για να μπορεί να χρησιμοποιηθεί από το Istio.

```
kubectl create secret tls istio-ca-secret \
 --cert=cert.pem --key=cert-key.pem --namespace istio-system
```

Τα εξουσιοδοτημένα αιτήματα για δεδομένα από το backend, πραγματοποιούνται από την μεριά του πελάτη (client side), συνεπώς, στις υπηρεσίες που θα είναι προσπελάσιμες και εκτός συστοιχίας θα προστεθεί και υπηρεσία του backend.

Έπειτα δημιουργήσαμε την Gateway με την οδηγία να κάνει χρήση το πιστοποιητικό και επιλέγοντας ρητά του συγκεκριμένους hosts που θα επιτρέπουμε ώστε να μην αφήνουμε να εισέλθει ανεπιθύμητη κίνηση μέσα στην συστοιχία μας ακόμα και αν δεν υπάρχει διαθέσιμη υπηρεσία να ανταποκριθεί.

```
          istio: ingressgateway
        servers:
          - port:
              number: 443
              name: https
              protocol: HTTPS
            tls:
              mode: SIMPLE
              credentialName: my-tls-secret
        hosts:
          - "frontend.services.com"
          - "backend.services.com"
          - "keycloak.local.services"
```

Και τέλος δημιουργούμε τις VirtualService, ώστε να καθορίσουμε που θα καθοδηγείται η κίνηση στο εσωτερικό της συστοιχίας.

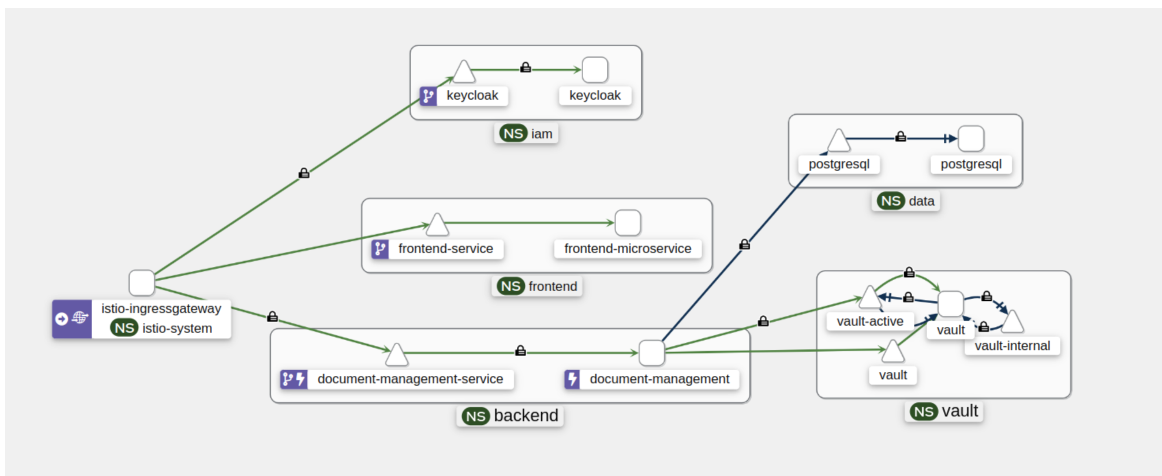
```
        tls:
        mode: SIMPLE
        credentialName: keycloak-tls-secret
        hosts:
        - "keycloak.local.services"
```

7.5.1 Προσθέτοντας ασφάλεια στις υπηρεσίες

Σε επίπεδο εφαρμογής πάντα ακολουθώντας την προσέγγιση ενός δικτύου μηδενικής εμπιστοσύνης, το πρώτο βήμα ήταν να προσθέσουμε mTLS στις υπηρεσίες που μιλάνε εσωτε-

ρικά της συστοιχίας, με διαμορφώσεις όπως η παρακάτω στα namespaces που διαθέτουμε sidecars, βάλαμε αυστηρή(strict) λειτουργία που σημαίνει ότι οποιαδήποτε κίνηση δεν είναι κρυπτογραφημένη θα απορρίπτεται.

```
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: backend
spec:
  mtls:
    mode: STRICT
```



Σχήμα 7.4 Διάγραμμα του πλέγματος της εφαρμογής

7.5.2 Δοκιμή load balancer

Σε αυτή την φάση χρησιμοποιήσαμε το εργαλείο δοκιμών φορτίου Fortio για την συγχρονισμένη και ταυτόχρονη αποστολή μεγάλου όγκου αιτημάτων. Επιλέξαμε να δοκιμάσουμε πάνω στον GET /documents endpoint που επιστρέφει την λίστα με τα αρχεία που διαθέτει η βάση. Αρχικά, σιγουρευτήκαμε ότι το connection pool της βάσης που κάνει χρήση το backend, δεν επηρεάζει τον χρόνο απόκρισης των αιτημάτων μιας και με πάνω από 15 συνδέσεις οι χρόνοι παρέμεναν στα ίδια όρια ακόμα και με 1000 ταυτόχρονα αιτήματα το δευτερόλεπτο.

```
func CreateConnectionPool(ctx context.Context) (*Postgres, error) {
  pgOnce.Do(func() {
    logger.Debug("Connection Pool Creation")
    cnf := config.Get()
    cfg, err := pgxpool.ParseConfig("postgres://" + cnf.PsqlUsername + ":" + cnf.
        PsqlPassword + "@" + cnf.PsqlHost + ":" + cnf.PsqlPort + cnf.PsqlDatabase)
    if err != nil {
```

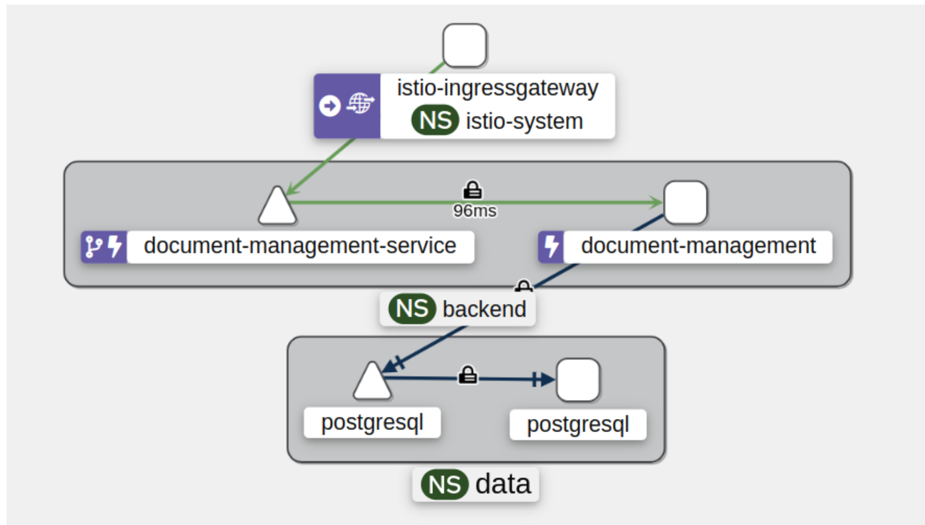
```
        logger.Fatal("Unable to parse config: %v\n", err)
        os.Exit(1)
      }
      cfg.MaxConns = cnf.MaxConns
      cfg.MinConns = cnf.MinConns
      cfg.MaxConnLifetime = 5 * time.Minute
      cfg.MaxConnIdleTime = 2 * time.Minute

      db, err := pgxpool.NewWithConfig(ctx, cfg)
      if err != nil {
        logger.Fatal("Unable to create connection pool: %v\n", err)
        os.Exit(1)
      }
      pgInstance = &Postgres{db}
    })

    return pgInstance, nil
  }
```

Στη συνέχεια, πραγματοποιήσαμε μια σειρά δοκιμών, ξεκινώντας με ένα μόνο αντίγραφο (replica) του backend και έπειτα επεκτείναμε τις δοκιμές μας για να συμπεριλάβουμε πολλαπλά αντίγραφα μαζί με την χρήση load balancer. Οπότε με την εντολή

```
fortio load -k -c 1000 -qps 1000 -t 1s https://backend.localdomain.com/documents
```

πραγματοποιήσαμε μια πληθώρα δοκιμών και συνδυασμών με αριθμό αντιγράφων και αλγορίθμων εξισορρόπησης του φόρτου. Με τη χρήση ενός DestinatioRule ορίζουμε ρητά τον αλγόριθμο εξισορρόπησης φόρτου.

```
kind: DestinationRule
metadata:
  name: backend-load-balancing
  namespace: backend
spec:
  host: document-management-service.backend.svc.cluster.local
  trafficPolicy:
    loadBalancer:
      simple: LEAST_CONN
```



Πίνακας 7.2 Πίνακας αποτελεσμάτων δοκιμών εξισορρόπησης φόρτου

Αριθμός αντιγράφων	Αλγόριθμος	Αριθμός αιτημάτων	Ποσοστό επιτυχίας	Μέσος χρόνος απόκρισης
1	-	1000	100%	96ms
2	ROUND_ROBIN	1000	100%	35ms
2	LEAST_CONN	1000	100%	45ms
3	ROUND_ROBIN	1000	100%	19ms
3	LEAST_CONN	1000	100%	27ms
5	ROUND_ROBIN	1000	100%	20ms
5	LEAST_CONN	1000	100%	29ms

Τα δεδομένα δείχνουν ότι το σύστημα φτάνει το όριο κλιμάκωσης του σε τρία αντίγραφα, όπου τα επιπλέον αντίγραφα δεν βελτιώνουν τους χρόνους μέσης απόκρισης. Ο αλγόριθμος εξισορρόπησης φορτίου Round Robin υπερτερεί του αλγορίθμου Least Connection, προσφέροντας καλύτερη κατανομή της κίνησης και αποδοτικότητα. Σημαντικό να αναφέρουμε είναι ότι οι δοκιμές έγιναν σε συστοιχία που τρέχει τοπικά, σε κάθε άλλη περίπτωση τα αποτελέσματα θα ήταν διαφορετικά αν υπολογίζαμε και την καθυστέρηση μιας πιο απομακρυσμένης επικοινωνίας.

7.5.3 Πρόσθετες ρυθμίσεις αξιοπιστίας

Στο ίδιο DestionationRule με τον load balancer, εισάγαμε και ρυθμίσεις για circuit braking εξασφαλίζοντας ότι το σύστημα παραμένει αξιόπιστο σε περιπτώσεις όπου κάποιο pod δεν λειτουργήσει σωστά και επιστρέψει 5 διαδοχικά σφάλματα λόγο εσωτερικού προβλήματος της εφαρμογής, σε περίοδο ενός λεπτού, τότε το απομονώνουμε για δεκαπέντε λεπτά και μέχρι η λειτουργία του να είναι και η αναμενόμενη.

```
trafficPolicy:
  outlierDetection:
   consecutive5xxErrors: 5
   interval: 1m
   baseEjectionTime: 15m
   maxEjectionPercent: 50
```

Σε επόμενη φάση συνεχίσαμε εφαρμόζοντας περιορισμό στον ρυθμό που θα δεχόμαστε αιτήματα στα εκτεθειμένα endpoints. Ξεκινήσαμε με δημιουργία ενός ConfigMap, το οποίο περιέχει τις ρυθμίσεις περιορισμού όπως για παράδειγμα τον περιορισμό στο endpoint /documents στα 1000 αιτήματα το λεπτό.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: ratelimit-config
  namespace: istio-system
data:
  config.yaml: |
    domain: "ratelimit"
    descriptors:
      - key: PATH
        value: "/documents"
        rate_limit:
          unit: minute
          requests_per_unit: 1000
```

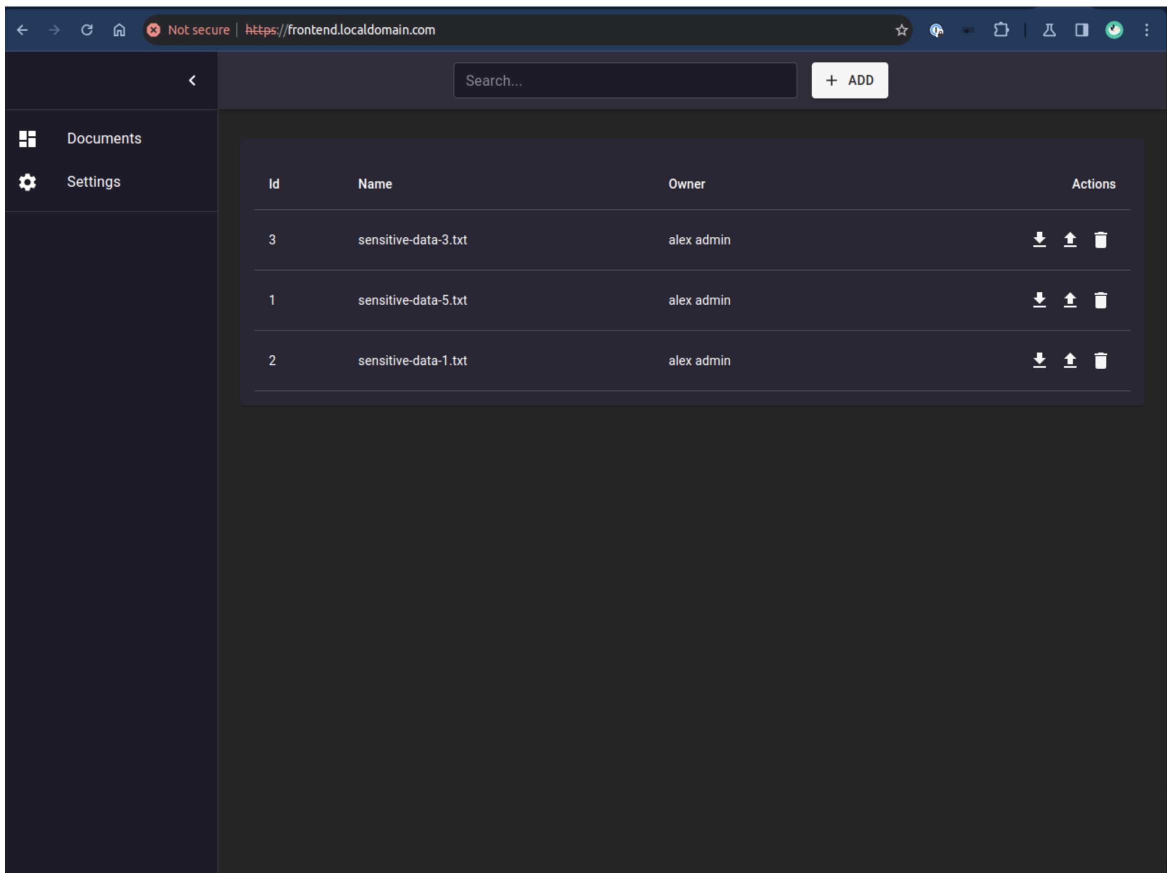
Έπειτα δημιουργήσαμε μια υπηρεσία βασισμένη στο image του envoyproxy/ratelimit που έχει υλοποιηθεί από την ομάδα του Envoy και την συνδέσαμε με μια βάση δεδομένων Redis που θα αποθηκεύει τον αριθμό των αιτημάτων που έχει δεχτεί το endpoint, συγκρίνοντας τον αριθμό που έχουμε βάλει στον ratelimit-config (το οποίο δυναμικά μπορεί να αλλάζει με βάση τις ανάγκες) αναστέλει την διαθεσιμότητα της υπηρεσίας και έτσι διατηρεί την σταθερότητα του συστήματος. Αυτό συμβαίνει όταν ένα αίτημα φτάσει στην gateway, αυτή επικοινωνεί με την υπηρεσία του ratelimit και αν έχει περάσει το όριο που έχει καθοριστεί, τότε αναχαιτίζει το αίτημα και δεν το αφήνει να συνεχίσει. Εδώ φαίνεται και η ανάγκη για πολύ γρήγορη επικοινωνία μεταξύ της gateway και τη υπηρεσίας ratelimit για αυτό και χρησιμοποιείται πρωτόκολλο gRPC. Ολοκληρώνουμε την ρύθμιση με ένα EnvoyFilter όπου στοχεύουμε την ingress για να εφαρμόσουμε και την όλη υλοποίηση,

μέσω του HTTP_FILTER δηλώνουμε ότι θέλουμε να επεξεργαστούμε την HTTP κίνηση στο επίπεδο της gateway προσθέτοντας ένα νέο φίλτρο ακριβώς πριν από τα δύο υφιστάμενα του filterchain μέσω της τιμής INSERT_BEFORE. Τέλος ορίζουμε ποιο είναι αυτό το νέο φίλτρο και τις ιδιαιτερότητές του, όπως και ποια υπηρεσία θα χρησιμοποιεί.

```
     istio: ingressgateway
   configPatches:
     - applyTo: HTTP_FILTER
     match:
       context: GATEWAY
       listener:
         filterChain:
           filter:
             name: "envoy.filters.network.http_connection_manager"
           subFilter:
             name: "envoy.filters.http.router"
   patch:
     operation: INSERT_BEFORE
     value:
       name: envoy.filters.http.ratelimit
       typed_config:
         "@type": type.googleapis.com/envoy.extensions.filters.http.ratelimit.v3.RateLimit
         domain: "ratelimit"
         failure_mode_deny: true
         timeout: 10s
         rate_limit_service:
           grpc_service:
             envoy_grpc:
               cluster_name: ratelimit
               authority: ratelimit.istio-system.svc.cluster.local
```

7.6 Διεπαφή εφαρμογής

Για διεπαφή της υλοποίησής του frontend εστιάσαμε στην αξιοποίηση της επέκτασης PKCE για το OAuth 2.0 στη διαδικασία εξουσιοδότησης με το Keycloak που αναλύσαμε στο προηγούμενο κεφάλαιο, εξασφαλίζοντας έτσι μια πιο ασφαλή διαδικασία ανακατεύθυνσης και πιστοποίησης.



Σχήμα 7.5 Διεπαφή εφαρμογής

Κεφάλαιο 8

Συμπεράσματα και μελλοντική εργασία

Η διπλωματική εργασία διερεύνησε το τοπίο του υπολογιστικού νέφους, εστιάζοντας στον καθοριστικό ρόλο των περιεκτών και των μικρουπηρεσιών στην ανάπτυξη κλιμακούμενων και ασφαλών εφαρμογών. Μέσω ολοκληρωμένης ανάλυσης και πρακτικής εφαρμογής, προσπαθήσαμε να ορίσουμε ένα πλαίσιο μέσα από βέλτιστες πρακτικές και τεχνολογίες που αντιμετωπίζει τις εγγενείς προκλήσεις ασφάλειας και αξιοπιστίας των περιβαλλόντων νέφους, αλλά και αξιοποιεί τα πλεονεκτήματα των περιεκτών και της ενορχήστρωσης για την ενίσχυση της λειτουργικής αποδοτικότητας.

Σαν μελλοντική εργασία θα μπορούσε μπορούμε να καθορίσουμε πιο εκτενής ρυθμίσεις του Istio, όπως fault injection και υλοποιώντας μια πιο κατανεμημένη σχεδίαση χρησιμοποιώντας πολλαπλές συστοιχίες αναλύοντας τα ευρήματα και τις προκλήσεις στην επικοινωνία μεταξύ τους. Με βάση το keycloak θα μπορούσε να υλοποιηθεί ένα πιο Context-based access control αντί του RBAC με υλοποίηση κάποια Service Provider Interface (SPI) επέκταση με χρήση γλώσσας προγραμματισμού Java που υποστηρίζει το Keycloak. Τέλος θα μπορούσε να υλοποιηθεί ένα pipeline πάνω σε jenkins jobs που θα σάρωνε τις λανθασμένες ρυθμίσεις και τις ευπάθειες των εικόνων.

Βιβλιογραφία

[1] P. Mell and T. Grance, ”The NIST Definition of Cloud Computing,” *NIST Special Publication 800-145*, 2011.

[2] National Institute of Standards and Technology. (2024). The NIST Cybersecurity Framework (CSF) 2.0. https://csrc.nist.gov/pubs/cswp/29/the-nist-cybersecurity-framework-csf-20/final

[3] Docker official documentation, https://docs.docker.com/ , *Docker, Inc.*, 2023.

[4] Gitlab DevOps definition, https://about.gitlab.com/topics/devops/ , *GitLab Inc.*, 2023.

[5] Shivakumar R Goniwada, ”Cloud Native Architecture and Design” , *Apress Berkeley, CA*, 2023.

[6] Rose, S., Borchert, O., Mitchell, S., and Connelly, S. (2020), ”Zero Trust Architecture,” *Special Publication (NIST SP)*, National Institute of Standards and Technology, Gaithersburg, MD, [online], https://doi.org/10.6028/NIST.SP.800-207

[7] ”A brief history of DevOps,” https://everythingdevops.dev/a-brief-history-of-devops-and-its-impact-on-software-development/, *everythingdevops.dev*

[8] https://nvd.nist.gov/ National Vulnerability Database *NIST*

[9] Nigel Poulton, ”The Kubernetes Book” , *Nigel Poulton Ltd* , 2023.

[10] ”The history of Kubernetes” https://www.ibm.com/blog/kubernetes-history/ *ibm.com*

[11] Kubernetes docs, https://kubernetes.io/docs/home/ , *Cloud Native Computing Foundation*.

[12] https://docs.kubelinter.io/ Kubernetes YAML files and Helm charts scanning tool

[13] https://developer.hashicorp.com/vault/docs/platform/k8s/injector Hashicorp Agent sidecar injector

[14] https://trivy.dev/ Trivy Security Scanner *Aqua Security*

[15] Matthew Portnoy, ”Virtualization Essentials, 3rd Edition”, *Wiley* , 2023.

[16] State of Kubernetes security report 2023, https://www.redhat.com/en/resources/state-kubernetes-security-report-2023 , *Red Hat, Inc.*, 2023.

[17] Helm Charts Official docs, https://helm.sh/docs/, *Cloud Native Computing Foundation*.

Παράρτημα Α

Συμπληρωματικές τεχνολογίες που χρησιμοποιήθηκαν

Γλώσσα προγραμματισμού Golang

Για την υλοποίηση του REST API χρησιμοποιήσαμε την γλώσσα προγραμματισμού Golang ή Go η οποία είναι μια στατικά τυποποιημένη γλώσσα προγραμματισμού που σχεδιάστηκε από την Google. Κυκλοφόρησε για πρώτη φορά το 2009 με έμφαση στην απλότητα και την αποτελεσματικότητα. Η Go διαθέτει ένα καθαρό συντακτικό που μαθαίνεται εύκολα και προσφέρει ισχυρές τυποποιημένες βιβλιοθήκες και ισχυρά εργαλεία. Οι μηχανισμοί συγχρονισμού της, όπως οι goroutines την καθιστούν ιδιαίτερα κατάλληλη για την κατασκευή εξυπηρετητών δικτύου υψηλής απόδοσης και ταυτόχρονων εφαρμογών.

React Framework

Για την διεπαφή της υλοποίησης χρησιμοποιήσαμε React η οποία αναπτύχθηκε από το Facebook για την κατασκευή διεπαφών χρήστη. Επιτρέπει στους προγραμματιστές να δημιουργούν μεγάλες διαδικτυακές εφαρμογές και να είναι απόλυτα δυναμικές στην μεταφόρτωση δεδομένων χωρίς επαναφόρτωση της σελίδας. Επιπλέον, το εικονικό DOM του React βελτιστοποιεί την απόδοση, καθιστώντας το πολύ αποτελεσματικό στην ενημέρωση του UI σε απόκριση σε ενέργειες του χρήστη ή αλλαγές δεδομένων.

PostgreSQL

Για την βάση δεδομένων του backend επιλέξαμε την PostgreSQL ή Postgres που είναι μια σχεσιακή βάση δεδομένων ανοικτού κώδικα. Αναπτύχθηκε στο Πανεπιστήμιο της Καλιφόρνιας, Berkeley, και κυκλοφόρησε το 1996. Η Postgres είναι ιδιαίτερα επεκτάσιμη και συμβατή με τα πρότυπα, υποστηρίζοντας μια μεγάλη ποικιλία τύπων δεδομένων, συμπεριλαμβανομένων των JSON, XML και πινάκων, καθώς και προηγμένα χαρακτηριστικά όπως η κληρονομικότητα πινάκων και τα ξένα κλειδιά. Η Postgres προσφέρει επίσης ισχυρή ακεραιότητα συναλλαγών, εξασφαλίζοντας τη συμμόρφωση με τα πρότυπα ACID (Atomicity, Consistency, Isolation, Durability), γεγονός που την καθιστά δημοφιλή επιλογή για επιχειρήσεις και εφαρμογές που απαιτούν σύνθετες συναλλαγές και υψηλά επίπεδα ακεραιότητας δεδομένων.