Πανεπιστήμιο Δυτικής Αττικής,
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Διπλωματική Εργασία

ISTIO. Ένα Πλέγμα Υπηρεσιών

Συγγραφέας: Δημήτριος Λεονταρίδης, 711171029
Εισηγητής: Νικόλαος Ψαρράς, Λέκτορας Εφαρμογών
Αθήνα, Οκτώβριος 2023

University of West Attica,

Department of Informatics and Computer Engineering

Diploma Thesis

ISTIO. A Service Mesh

Author: Dimitrios Leontaridis, 711171029

Supervisor: Nikolaos Psarras, Lecturer

Athens, October 2023

ISTIO. Ένα Πλέγμα Υπηρεσιών

Πανεπιστήμιο Δυτικής Αττικής,
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Διπλωματική Εργασία

ISTIO. Ένα Πλέγμα Υπηρεσιών

Μέλη τριμελούς επιτροπής εξέτασης

Α/Α	Όνομα Επώνυμο	Βαθμίδα	Ψηφιακή Υπογραφή
1	Βασίλειος Μάμαλης	Καθηγητής	Vasileios Mamalis Digitally signed by Vasileios Mamalis Date: 2023.10.17 00:26:03 +03'00'
2	Αντώνιος Μπόγρης	Καθηγητής	Antonios Bogris Digitally signed by Antonios Bogris Date: 2023.10.16 15:17:03 +03'00'
3	Νικόλαος Ψαρράς	Λέκτορας Εφαρμογών	Digitally signed by Nikolaos Psarras Date: 2023.10.12 00:25:58 +03'00'

Ημερομηνία Εξέτασης: 11/10/2023

Δήλωση συγγραφέα Διπλωματικής Εργασίας

Ο κάτωθι υπογεγραμμένος Δημήτριος Λεονταρίδης του Αθανασίου, με αριθμό μητρώου 711171029 φοιτητής 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. Ακόμη, 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

Δημήτριος Λεονταρίδης

Περίληψη

Η υιοθέτηση τεχνολογιών περιεκτών και εργαλείων ενορχήστρωσης όπως το Kubernetes, έχει διαμορφώσει τον τρόπο υλοποίησης και ανάπτυξης εφαρμογών ιστού. Ο ανταγωνισμός και οι αυξημένες δυνατότητες των σημερινών δικτύων, έχουν φέρει ένα νέο κύμα παροχών και υπηρεσιών που καλούνται οι εφαρμογές να αφομοιώσουν. Οι ανάγκες κλιμάκωσης, αρθρωτής διαχείρισης και επέκτασης της λειτουργικότητας μιας εφαρμογής έχουν οδηγήσει στην υιοθέτηση του μοντέλου των μικροϋπηρεσιών. Η φύση του μοντέλου αυτού, έφερε αλλαγές στον τρόπο ανάπτυξης εφαρμογών προσθέτοντας νέα επίπεδα λειτουργικότητας. Τεχνολογίες πλέγματος υπηρεσιών, όπως το Istio, συνδυάζονται με εργαλεία ενορχήστρωσης, προσφέροντας ένα σύνολο λειτουργιών επιπέδου εφαρμογής. Η συγγραφή της εργασίας αποσκοπεί στην διεξοδική ανάλυση του τρόπου λειτουργίας της πλατφόρμας Istio καθώς και των δυνατοτήτων που προσφέρονται μέσα από αυτή. Ακόμη, η εργασία στοχεύει στην κατανόηση των διαδικασιών ενορχήστρωσης μέσα από την διερεύνηση της πλατφόρμας Kubernetes, καθώς και των πόρων που μπορούν να δημιουργηθούν σε εκείνη. Έμφαση δόθηκε στην διαχείριση της κίνησης, την παρατηρησιμότητα και την ασφάλεια εντός ενός πλέγματος υπηρεσιών, αποδίδοντας πραγματικό σενάριο ανάπτυξης εφαρμογής με την χρήση των παραπάνω εργαλείων. Η εφαρμογή αποτελεί μία πλατφόρμα ιστού, με την οποία ένας χρήστης μπορεί να δημιουργήσει λίστες αναπαραγωγής καθώς και να προσθέσει μουσικά κομμάτια σε εκείνες. Η υλοποίησή της στηρίζεται στην αρχιτεκτονική μικροϋπηρεσιών, περιέχοντας ανάπτυξη υπηρεσιών βάσεων δεδομένων (MongoDB), APIs (NodeJS) και διεπαφής χρήστη (ReactJS). Επιπλέον, ακολούθησε πρακτική εφαρμογή των μηχανισμών δρομολόγησης επιπέδου εφαρμογής, ασφάλειας και παρατηρησιμότητας του Istio προσφέροντας ένα σύνολο από πολύτιμες γνώσεις.

Λέξεις Κλειδιά: Kubernetes, Istio, Τεχνολογίες Περιεκτών, Ενορχήστρωση, Μικροϋπηρεσίες, Πλέγμα Υπηρεσιών

Abstract

The adoption of containerization technologies and orchestration tools such as Kubernetes has shaped the way web applications are implemented and developed. The competition and increased capabilities of today's networks have brought a new wave of benefits and services that applications are being asked to assimilate. The needs for scalability, modularity and extension of an application's functionality have led to the adoption of the microservices model. The nature of this model has brought changes in the way applications are developed adding new layers of functionality. Service mesh technologies, such as Istio, are combined with orchestration tools, offering a set of application layer functionality. This paper aims to provide a thorough analysis of the way that Istio platform works and the functionality offered through it. Furthermore, the thesis aims to understand the orchestration processes through the exploration of the Kubernetes platform, as well as the resources that can be created within it. Emphasis was placed on traffic management, observability and security within a service mesh, yielding a real application development scenario using the above tools. The application is a web platform, with which a user can create playlists as well as add music tracks to them. Its implementation is based on the microservices architecture, including development of database services (MongoDB), APIs (NodeJS) and user interface (ReactJS). In addition, practical implementation of Istio's application layer routing, security and observability mechanisms followed, providing a set of valuable insights.

Key words: Kubernetes, Istio, Containerization Technologies, Orchestration, Microservices, Service Mesh

Πίνακας Περιεχομένων

Κεφάλαιο 1: Εισαγωγή 13
Κεφάλαιο 2: Πλατφόρμα Kubernetes 15
2.1 Χαρακτηριστικά και Ιστορικά στοιχεία 15
2.2 Αρχιτεκτονική 16
Master Node 16
Slave – Worker Node 17
2.3 Βασικοί πόροι 17
Pod 17
Deployment 18
Service 18
ClusterIP Service 19
Nodeport Service 19
StatefulSets 20
ConfigMap 22
Secrets 22
Role Based Access Control 22
Κεφάλαιο 3: Επίπεδο δεδομένων στο ISTIO 25
3.1 Η πορεία του Istio 25
3.2 Sidecar περιέκτης 26
3.3 Envoy Proxy 26
Κεφάλαιο 4: Επίπεδο ελέγχου στο ISTIO 29
4.1 Pilot 29
Pilot: Το μοντέλο διαμόρφωσης 29
Pilot: Διαδικασία προώθησης διαμορφώσεων 30
4.2 Citadel 34
Citadel: Χρήση εξωτερικής Αρχής Πιστοποίησης 35
Citadel: Υπηρεσία λήψης Αιτημάτων Υπογραφής Πιστοποιητικών 35
Citadel: Υπηρεσία Ανακάλυψης Μυστικών 36
4.3 Galley 37
Galley: Έλεγχος ορθότητας και κανονικοποίηση διαμορφώσεων 37
Galley: Απόδοση και Επεκτασιμότητα 38
4.4 Istiod 39

Κεφάλαιο 5: Χαρακτηριστικά του ISTIO 40
5.1 Διαχείριση Κίνησης 40
Δρομολόγηση κίνησης βάση HTTP επικεφαλίδας 40
Load Balancing 41
Traffic Shifting 42
Rate Limiting 43
Circuit Breaking 46
Fault Injection 47
5.2 Παρατηρησιμότητα 48
Prometheus 49
Grafana 52
Kiali 53
5.3 Ασφάλεια 54
SSL/TLS 55
Ασφαλείς Πύλες – Gateways 55
SSL/TLS και Υπηρεσίες πλέγματος 56
Υπηρεσία Cloudflare 57
Εξουσιοδότηση και Έλεγχος Πρόσβασης 58
Κεφάλαιο 6: Ανάπτυξη Εφαρμογής Μικροϋπηρεσιών 60
6.1 Υποδομή και Εργαλεία 60
ProxMox 60
Microk8s 61
6.2 Ingress Gateway 62
6.3 Βάσεις Δεδομένων 65
Βάση Δεδομένων Track 67
Βάση Δεδομένων Playlist 68
6.4 APIs 69
Υπηρεσία Track 69
Υπηρεσία Playlist 71
6.5 Διεπαφή Χρήστη 73
Υπηρεσία frontend 74
Canary Deployment 78
Κεφάλαιο 7: Συμπεράσματα και Προοπτικές 80

7.1 Συμπεράσματα 80
7.2 Προοπτικές Διπλωματικής Εργασίας 81
Κεφάλαιο 8: Παράρτημα 82
8.1 Βασικοί πόροι του Istio 82
DestinationRule 82
Virtual Service 82
8.2 JWT 83
Πηγές 84

Πίνακας Εικόνων

Εικόνα 1: Επίπεδο δεδομένων του Πλέγματος Υπηρεσιών 26
Εικόνα 2: Επίπεδο ελέγχου στο Istio 39
Εικόνα 3: Ρυθμός άφιξης των Bytes στις εκδόσεις v1, v2 της υπηρεσίας frontend 43
Εικόνα 4: Γράφημα απεικόνισης Ρυθμού Μετάδοσης Πακέτων των υπηρεσιών 52
Εικόνα 5: Μέσο Εύρος Ζώνης Μετάδοσης των mongodb περιεκτών 53
Εικόνα 6: Γράφημα του πλέγματος υπηρεσιών της μουσικής βιβλιοθήκης 54
Εικόνα 7: Client - Cloudflare Server TLS encryption 58
Εικόνα 8: ProxMox Virtual Environment 61
Εικόνα 9: Ενεργοποιημένα addons στην πλατφόρμα Microk8s 62
Εικόνα 10: VPN tunnel Hostnames 63
Εικόνα 11: Επιτυχής εγκατάσταση SSL πιστοποιητικού 64
Εικόνα 12: Ενδεικτικό ID μουσικού κομματιού 67
Εικόνα 13: Εγγραφές συλλογής tracks 68
Εικόνα 14: Εγγραφές συλλογής playlists 69
Εικόνα 15: UI Modals μουσικής βιβλιοθήκης 75
Εικόνα 16: UI μουσικής βιβλιοθήκης 76

Ευρετήριο

A	AES: Advanced Encryption Standard
	AKS: Azure Kubernetes Service
	API: Application Programmable Interface
C	CA: Certificate Authority
	CDS: Cluster Discovery Service
	CN: Common Name
	CPU: Central Processing Unit
	CRUD: Create Read Update Delete
	CSR: Certificate Signing Request
	CSRS: Certificate Signing Request Service
D	DB: Data Base
	DDoS: Distributed Denial of Service
	DES: Data Encryption Standard
	DevOps: Development Operations
	DS: Discovery Service
E	EDS: Endpoint Discovery Service
	EKS: Elastic Kubernetes Service
	ESP: Encapsulating Security Protocol
F	FQDN: Fully Qualified Domain Name
G	GB: Giga Byte
	GKS: Google Kubernetes Service
	gRPC: google Remote Procedure Call
H	HA: Hight Availability
	HEX: Hexadecimal
	HTML: Hyper Text Mark-up Language
	HTTP: Hypertext Transfer Protocol
	HTTPS: Hypertext Transfer Protocol Secure
I	IP: Internet Protocol
J	JS: JavaScript
	JSON: JavaScript Object Notation
	JWKS: JSON Web Token Key Set

	JWT: JSON Web Token
K	K8S: Kubernetes
	kB/s: kiloByte/second
L	LDS: Listener Discovery Service
M	MAC: Message Authentication Code
	MiB/s: MebiByte/second
	MIPS: Million Instructions Per Second
	mTLS: mutual Transport Layer Security
O	OSI: Open Systems Interconnection
P	PaaS: Platform as a Service
Q	QoS: Quality of Service
R	RAM: Random Access Memory
	RBAC: Role Based Access Control
	RDS: Route Discovery Service
S	SDN: Software Defined Networks
	SDS: Secret Discovery Service
	SQL: Structured Query Language
	SSL: Secure Socket Layer
T	TCP: Transmission Control Protocol
	TFA: Two Factor Authentication
U	URL: Uniform Resource Locator
	URI: Uniform Resource Identifier
V	VE: Virtual Environment
	VM: Virtual Machine
	VoIP: Voice over IP
	VPN: Virtual Private Network

ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ

Αντικείμενο μελέτης της εργασίας αποτελεί ο τρόπος λειτουργίας της πλατφόρμας ανοιχτού κώδικα Istio και η επεξήγηση του τρόπου εφαρμογής των διαφόρων λειτουργιών που εισάγονται σε ένα πλέγμα υπηρεσιών. Επίσης παρουσιάζεται πραγματικό παράδειγμα ανάπτυξης μιας εφαρμογής μουσικής βιβλιοθήκης, με στόχο την επίδειξη του τρόπου με τον οποίο το Istio και το Kubernetes μπορούν να συνδυαστούν, για την δημιουργία μιας ανθεκτικής, επεκτάσιμης και ασφαλούς εφαρμογής μικροϋπηρεσιών.

Η προσέγγιση της εργασίας χαρακτηρίζεται από την διεξοδική ανάλυση όλων των τεχνολογιών που χρησιμοποιήθηκαν καθώς και επιπρόσθετων λειτουργιών του Istio. Ακολουθούν σενάρια χρήσης ανάπτυξη πλέγματος υπηρεσιών, με σημείο αναφοράς την μεθοδολογία υλοποίησης της μουσικής βιβλιοθήκης. Η δημιουργία της εφαρμογής αποτέλεσε σημαντικό ρόλο στην εμπέδωση και τεκμηρίωση της θεωρητική πτυχής της προσέγγισης. Δίχως την ανάπτυξη της εφαρμογής, η παρουσίαση έγκυρου σεναρίου πραγματικού κόσμου δεν θα ήταν δυνατή.

Στο **Κεφάλαιο 2** της εργασίας παρουσιάζονται τεχνικές γνώσεις, που στοχεύουν στην κάλυψη του απαραίτητου θεωρητικού υπόβαθρου. Οι γνώσεις αυτές αφορούν την ενορχήστρωση υπηρεσιών στην πλατφόρμα Kubernetes, παρέχοντας τις απαραίτητες πληροφορίες και έννοιες για την ανάπτυξη ενός συμπλέγματος υπηρεσιών. Σημαντικό σημείο γνώσης αποτελούν οι βασικοί πόροι με του οποίους υλοποιήθηκε η εφαρμογή της μουσικής βιβλιοθήκης.

Στο **Κεφάλαιο 3** πραγματοποιείται ανάλυση του επιπέδου ελέγχου στην πλατφόρμα του Istio, εστιάζοντας στην εκτέλεση ειδικών και υψηλής απόδοσης διακομιστών μεσολάβησης στο πλέγμα. Εξετάζεται η λειτουργία τους ως sidecar περιέκτες καθώς και η χρήση τους ως δρομολογητές επιπέδου εφαρμογής.

Στο **Κεφάλαιο 4** διερευνάται ο τρόπος με τον οποίο πραγματοποιείται η κεντρικοποιημένη διαχείριση και διαμόρφωση του πλέγματος υπηρεσιών. Πραγματοποιείται ανάλυση των επιμέρους συνιστωσών, από τις οποίες αποτελείται το επίπεδο ελέγχου της πλατφόρμας Istio. Συγκεκριμένα το κεφάλαιο αφορά τις λειτουργίες που επιτελούν οι υπηρεσίες Pilot, Citadel και Galley στο control plane. Τέλος γίνεται αναφορά στην υιοθέτηση ενός μονολιθικού μοντέλου, για την υλοποίηση των μεταγενέστερων εκδόσεων του επιπέδου ελέγχου.

Το **Κεφάλαιο 5** εστιάζει στην απαρίθμηση των δυνατοτήτων της πλατφόρμας Istio, παρουσιάζοντας τις διάφορες λειτουργίες που εισάγονται στο πλέγμα. Οι λειτουργίες αφορούν την διαχείριση της κίνησης, την προηγμένη δρομολόγηση, την ασφάλεια, την παρατηρησιμότητα και την οπτικοποίηση. Γίνεται διερεύνηση του τρόπου ενεργοποίησης εκείνων των μηχανισμών, αξιοποιώντας ως σημείο αναφοράς

διάφορες YAML διαμορφώσεις. Συνολικά στις λειτουργίες περιέχονται: η δρομολόγηση της κίνησης, η κατανομή φόρτου, η μετατόπιση της κίνησης, η οριοθέτηση ρυθμού, η διακοπή κυκλώματος, η προσομοίωση σφαλμάτων, η ενεργοποίηση υλοποιήσεων του πρωτοκόλλου TLS κ.α.

Στο **Κεφάλαιο 6** της εργασίας πραγματοποιείται επεξήγηση της εφαρμογής η οποία επιλέχθηκε για την υλοποίηση του πρακτικού μέρους. Γίνεται ανάλυση των υπηρεσιών από τις οποίες αποτελείται η εφαρμογή, καθώς και επεξήγηση της λειτουργικότητάς τους. Οι υπηρεσίες υλοποιούνται μέσα από την ανάπτυξη βάσεων δεδομένων, APIs και διεπαφής χρήστη. Στην ανάλυση, περιέχονται οι διάφοροι πόροι του Kubernetes που αξιοποιήθηκαν στα πλαίσια της εφαρμογής, καθώς και οι μηχανισμοί διαχείρισης της κίνησης, δρομολόγησης και ενεργοποίησης του πρωτοκόλλου TLS που επιλέχθηκαν.

Στο **Κεφάλαιο 7** παρουσιάζονται τα συμπεράσματα που προέκυψαν από την διαδικασία διερεύνησης και ανάπτυξης της εφαρμογής μικροϋπηρεσιών. Ακόμη, συζητούνται οι προοπτικές για μελλοντική έρευνα, καθώς και προτάσεις επέκτασης της παρούσας εφαρμογής. Εκτός αυτού, προτείνονται και διάφορα άλλα projects, τα οποία αφορούν θεματικά την αρχιτεκτονική μικροϋπηρεσιών, την κατανεμημένη υλοποίηση βάσεων δεδομένων, την ανάπτυξη εφαρμογών με την χρήση PaaS λύσεων νέφους, καθώς και την υλοποίηση DevOps pipelines.

Τέλος στο **Κεφάλαιο 8** βρίσκεται το παράρτημα της εργασίας, στο οποίο περιέχονται οι βασικοί πόροι του Istio καθώς και η ανάλυση της διαδικασίας εξουσιοδότησης με την χρήση JSON Web Tokens.

ΚΕΦΑΛΑΙΟ 2: ΠΛΑΤΦΟΡΜΑ KUBERNETES

Σε αυτό το κεφάλαιο της εργασίας παρέχεται μια σύντομη επισκόπηση των θεμελιωδών εννοιών, που σχετίζονται με την αρχιτεκτονική της πλατφόρμας Kubernetes, καθώς και μία ολοκληρωμένη διερεύνηση των βασικών πόρων για την ανάπτυξη μιας εφαρμογής μικροϋπηρεσιών. Στην πρώτη ενότητα γίνεται επισκόπηση της ιστορικής πορείας της πλατφόρμας, δίνοντας έμφαση στην δημιουργία, εξέλιξη και ανακοίνωση του εγχειρήματος. Ύστερα αναφέρονται τα σημαντικότερα χαρακτηριστικά που διαθέτει η πλατφόρμα, καθιστώντας την, ως το πλέον δημοφιλέστερο και ευρέως χρησιμοποιούμενο εργαλείο ενορχήστρωσης μικροϋπηρεσιών. Στην δεύτερη ενότητα ορίζονται τα δύο επίπεδα της πλατφόρμας, ελέγχου και δεδομένων, καθώς επίσης και τα επιμέρους στοιχεία - διεργασίες που εκτελούνται στο καθένα. Τέλος στην τρίτη ενότητα αναλύονται οι πόροι της πλατφόρμας, με τους οποίους υλοποιούνται οι επιμέρους υπηρεσίες στα πλαίσια ανάπτυξης μιας εφαρμογής μικροϋπηρεσιών, όπως εκείνη της μουσικής βιβλιοθήκης.

2.1 Χαρακτηριστικά και Ιστορικά στοιχεία

Σε επίπεδο χρήσης, το Kubernetes αξιοποιείται ευρέως σε περιβάλλοντα παραγωγής, για την αποτελεσματική διαχείριση των υπηρεσιών εφαρμογών. Στις λειτουργίες του εντάσσεται η αρθρωτή υλοποίηση και οι δυνατότητες οριζόντιας κλιμάκωσης των εφαρμογών, που το καθιστούν πολύτιμο τόσο για on – premise εφαρμογές όσο και για εφαρμογές που βασίζονται σε υπηρεσίες νέφους. Παρουσιάζει δυνατότητες, όπως υψηλή διαθεσιμότητα, κλιμακοθεσιμότητα, κλιμάκωση, ανάκτηση δεδομένων και έλεγχο πρόσβασης, εξασφαλίζοντας σταθερή απόδοση, προηγμένη διαχείριση συμπλέγματος και ασφάλεια. Επιπλέον, υποστηρίζεται δυναμική προσαρμογή του πλήθους των περιεκτών (αυτόματη κλιμάκωση συμπλέγματος) που αντικατοπτρίζουν τη συστοιχία, ώστε να ανταποκρίνεται στις μεταβαλλόμενες απαιτήσεις, παρέχοντας βέλτιστη χρήση και εξοικονόμηση πόρων σε περιπτώσεις χαμηλής ζήτησης.

Όταν αναφερόμαστε στην πλατφόρμα Kubernetes, αναφερόμαστε κατ’ ουσία σε ένα εργαλείο ανοιχτού κώδικα ενορχήστρωσης μικροϋπηρεσιών. Η απαρχή του, εντοπίζεται στο σύστημα διαχείρισης συμπλεγμάτων "Borg", μία πρωτοβουλία της Google [1] στην απόπειρα της να διαχειριστεί τον τεράστιο όγκο υπηρεσιών και εφαρμογών της, έργο που παρείχε πολύτιμα μαθήματα. Υπό την ονομασία "Project Seven" το 2013, μια ομάδα της Google με επικεφαλής τους Joe Beda, Brendan Burns και Craig McLuckie βασιζόμενη στην τεχνογνωσία που είχε αποκτηθεί, εργάστηκε σε ένα σύστημα διαχείρισης περιεκτών ανοιχτού κώδικα, το οποίο εξελίχθηκε στο Kubernetes. Το 2014, κατά τη διάρκεια του DockerCon, η Google ανακοίνωσε το άνοιγμα του κώδικα του Kubernetes [2] καθιστώντας αυτό αλλά και την τεχνογνωσία του διαθέσιμα στην ευρύτερη κοινότητα. Από εκεί και έπειτα, με γρήγορους ρυθμούς, το Kubernetes έγινε το καθιερωμένο εργαλείο ενορχήστρωσης περιεκτών.

2.2 Αρχιτεκτονική

Προς βαθύτερη κατανόηση της αρχιτεκτονικής και της λειτουργίας του Kubernetes, θεμιτή είναι η ανάλυση των βασικών στοιχείων που το αποτελούν. Αρχικά, δύο είναι οι κατηγορίες στις οποίες ανήκουν τα στοιχεία αυτά, το επίπεδο ελέγχου και το επίπεδο δεδομένων. Το επίπεδο ελέγχου αποτελείται από Master Nodes και το επίπεδο δεδομένων από Slave (Worker) Nodes.

Master Node

Σε ένα master node, εκτελείται ένα σύνολο βασικών διεργασιών που είναι υπεύθυνες για τη διαχείριση και τον έλεγχο ολόκληρου του συμπλέγματος. Εμπεριέχει στοιχεία όπως το Kube – Apiserver, KubeCtl, KubeScheduler και τη βάση δεδομένων etcd. Τα εν λόγω στοιχεία αναλύονται ως εξής:

Kube – APIserver: Λειτουργεί ως το κύριο σημείο επικοινωνίας και διαχείρισης του Kubernetes. Μέσω αυτού, λαμβάνονται Restful αιτήματα από χρήστες και εξωτερικά προγράμματα. Αποτελεί μέσο επικοινωνίας μεταξύ των διαφορετικών μερών του επιπέδου ελέγχου. Για παράδειγμα αιτήματα προς εκείνο αφορούν τροποποιήσεις, δημιουργία, διαγραφές Pods ή υπηρεσιών. Μετά από την λήψη κάθε αιτήματος, ο API διακομιστής του Kubernetes πραγματοποιεί αυθεντικοποίηση και εξουσιοδότηση με την χρήση μη κεντρικοποιημένου μηχανισμού ελέγχου ταυτότητας σε unmanaged περιβάλλοντα και κεντρικοποιημένου ελέγχου σε managed περιβάλλον. Άλλοι τρόποι αλληλεπίδρασης με την πλατφόρμα Kubernetes, αποτελούν το Command Line Interface (CLI) και το User Interface (UI) με την χρήση εργαλείων, όπως το Kubernetes Dashboard.

KubeCtl: Αποτελεί ένα σύνολο από ελεγκτές (controllers), οι οποίοι επιτελούν λειτουργίες όπως επίβλεψη και παρακολούθηση του συμπλέγματος, χειρισμό των πόρων και διαχείριση των χώρων ονόματος (namespaces). Για παράδειγμα ένας από τους πολλούς controllers είναι υπεύθυνος σε περιπτώσεις καταστροφής κάποιου Pod ή περιέκτη, έτσι ώστε να προβεί σε επιδιόρθωση ή σε επανεκκίνηση.

Kube-Scheduler: Ο ρόλος του είναι ο καθορισμός της βέλτιστης τοποθέτησης των Pods σε κόμβους μέσα στο σύμπλεγμα. Χρησιμοποιεί διάφορες πολιτικές και αλγόριθμους για να λαμβάνει έξυπνες αποφάσεις, σχετικά με το ποιος κόμβος πρέπει να εκτελέσει ένα νεοσύστατο ή αναπρογραμματισμένο pod. Ο πρωταρχικός στόχος του kube-scheduler είναι η αποτελεσματική χρήση των πόρων και η κατανομή του φόρτου εργασίας στο σύμπλεγμα. Διαδραματίζει βασικό ρόλο στη διατήρηση της συνολικής υγείας και απόδοσης του συμπλέγματος.

Etcd: Αποτελεί την βάση δεδομένων στην οποία αποθηκεύονται δεδομένα σχετικά με τις πληροφορίες διαμόρφωσης και κατάστασης του συμπλέγματος. Τα δεδομένα αποθηκεύονται σε μορφή ζεύγους key –

value. Η βάση δεδομένων etcd μπορεί να κλιμακωθεί μαζί με τα master nodes για την διασφάλιση υψηλής διαθεσιμότητας, παραμένοντας συγχρονισμένη μεταξύ των διαφορετικών master κόμβων. Εναλλακτικά μπορεί να υλοποιηθεί και ως εξωτερική βάση δεδομένων (εκτός των master nodes) και να κλιμακωθεί αντίστοιχα, παρέχοντας ανεκτικότητα σε σφάλματα[3].

Slave – Worker Node

Container runtime: Κάθε slave κόμβος στο σύμπλεγμα πρέπει να έχει εγκατεστημένο ένα container runtime, προκειμένου να λειτουργήσουν τα εγκατεστημένα Pods και να εκτελεστούν οι αντίστοιχοι περιέκτες. Διαχειρίζεται ολόκληρο τον κύκλο ζωής των περιεκτών και μέσω αυτού φορτώνονται οι αντίστοιχες εικόνες στους περιέκτες. Στις εικόνες (container images) περιέχεται ο κώδικας της εφαρμογής μαζί με τα απαραίτητα dependencies, με τελικό σκοπό την εκτέλεση της προαναφερθείσας container εικόνας.

Kubelet: Η λειτουργία του kubelet αναφέρεται στην εφαρμογή των διαμορφώσεων στα αντίστοιχα pods. Δέχεται διαμορφώσεις κόμβου και αναμεταδίδει συμβάντα και καταστάσεις (κόμβου) μέσω του kube – api διακομιστή προς το επίπεδο ελέγχου. Η αποδοτική διαχείριση των πόρων του κόμβου είναι μία από τις εξίσου σημαντικές λειτουργίες, έτσι ώστε να μην ξεπερνιούνται τα όρια της διαθέσιμης χωρητικότητας κατά την εκτέλεση των pods. Ακόμη, είναι υπεύθυνο για την δημιουργία του κατάλληλου δικτυώματος εντός του συμπλέγματος, καθώς και την ανάθεση IP διευθύνσεων στα pods. Τέλος μέσω του kubelet φορτώνονται οι κατάλληλοι όγκοι προς διαμοίραση, στα αντίστοιχα pods.

Kube – Proxy: Επιτρέπει την απρόσκοπτη επικοινωνία μεταξύ διαφορετικών pods, κόμβων και υπηρεσιών εντός του συμπλέγματος. Το Kube Proxy διασφαλίζει ότι τα αιτήματα δικτύου φτάνουν στον σωστό προορισμό, βάση του πίνακα δρομολόγησης του κόμβου. Διαδραματίζει βασικό ρόλο στην δρομολόγηση αιτημάτων προς τις υπηρεσίες διατηρώντας συνδεσιμότητα κατά μήκος του δικτυώματος μεταξύ των pods. Σε περίπτωση όμως που ενσωματωθεί το Istio εντός ενός Kubernetes συμπλέγματος, ο Kube Proxy παρακάμπτεται και χρησιμοποιείται ο Envoy proxy.

2.3 Βασικοί πόροι

Σε αυτή την ενότητα αναλύονται οι βασικοί πόροι που παρέχονται από το εργαλείο ενορχήστρωσης Kubernetes. Διερευνάται ο τρόπος δήλωσης και διαμόρφωσης εκείνων των πόρων, καθώς και της συνολικής λειτουργικότητάς τους. Συγκεκριμένα οι πόροι Pod, Deployment, Service, ClusterIP, StaefulSet, ConfigMap και Secret χρησιμοποιήθηκαν στην ανάπτυξη της μουσικής βιβλιοθήκης.

Pod

Στο πλαίσιο της πλατφόρμας Kubernetes, το "Pod" αντιπροσωπεύει τη μικρότερη μονάδα ανάπτυξης, λειτουργώντας ως το βασικό οικοδομικό στοιχείο της πλατφόρμας. Κάθε υπηρεσία μπορεί να περιέχει ένα ή και περισσότερα όμοια pods, τα οποία ουσιαστικά λειτουργούν εξειδικευμένα για διακριτές λειτουργίες

εφαρμογών, όπως είναι οι διακομιστές ιστού ή οι βάσεις δεδομένων. Στα Pods, ταυτοχρόνως, μπορεί να περιλαμβάνονται συνήθως ένας αλλά κατά περίπτωση και περισσότεροι περιέκτες (containers), στους οποίους αποδίδονται συγκεκριμένοι πόροι (όπως RAM και CPU). Για τον ακριβή καθορισμό της κατανομής πόρων στον περιέκτη (container) εντός των Pods, χρησιμοποιείται μία YAML διαμόρφωση, γνωστή και ως διαμόρφωση Pod [4]. Αυτή η λεπτομερής προδιαγραφή εξασφαλίζει την ισομερή ανάθεση πόρων για κάθε περιέκτη, διασφαλίζοντας την αποτελεσματική λειτουργία τους. Η επικοινωνία, επιπρόσθετα, μεταξύ των containers που βρίσκονται εντός ενός Pod, εξασφαλίζεται μέσω της διεύθυνσης localhost loopback address - 127.0.0.1, διασφαλίζοντας με αυτό τον τρόπο την απρόσκοπτη αλληλεπίδραση τους. Επιπλέον, τα Pods έχουν σχεδιαστεί έχοντας έναν προσωρινό χαρακτήρα, καθιστώντας τα έτσι εύκολα διαχειρίσιμα και επιτρέποντας γρήγορες αλλαγές όπως τη δημιουργία, το τερματισμό ή την αντικατάστασή τους κατά απαίτηση. Η ομαδοποίηση των pods, βέβαια, επιτυγχάνεται με την χρήση υψηλού επιπέδου αφαιρέσεις όπως ReplicaSets, Deployments ή StatefulSets, με απώτερο στόχο την παροχή κλιμακούμενης (scalability) λειτουργικότητας, κυλιόμενων ενημερώσεων (rolling updates) και συνολικής διαχείρισης της κατάστασης μιας εφαρμογής.

Deployment

Αποτελεί πηγή διαμόρφωσης ενός ή περισσοτέρων Pods (`spec.replicas`), με την οποία καθορίζεται η λειτουργικότητα των pods καθώς και οι ρυθμίσεις τους. Αρχικά σε αυτή την YAML διαμόρφωση [5] πραγματοποιείται δήλωση του κύριου περιέκτη και της εικόνας που θα χρησιμοποιηθεί. Με την εφαρμογή του Deployment, εκείνος ο περιέκτης (`spec.template.spec.containers`) εγχέεται σε κάθε ένα από τα pods της διαμόρφωσης. Ακόμη γίνεται χρήση του πεδίου selector (`spec.selector.matchLabels.app`) για την διαμόρφωση πολλαπλών pods με το ίδιο `app` label (`spec.template.metadata.labels.app`). Αυτό μπορεί να είναι χρήσιμο σε περίπτωση, που έχουν εκτελεστεί και άλλα pods στο παρελθόν με την χρήση μεμονωμένων pod διαμορφώσεων. Η χρήση της ετικέτας `version` γίνεται για την διαμόρφωση υποσυνόλων (`subsets`, βλέπε Παράρτημα: Destination Rules) από pods, τα οποία αναφέρονται σε διαφορετικές εκδόσεις της εφαρμογής. Τέλος, με την ετικέτα `environment` (`template.spec.labels.environment`) καθορίζεται το περιβάλλον (π.χ. production, staging, deployment) ανάπτυξης εφαρμογής, ομαδοποιώντας τα subsets σε λογικές ομάδες (περιβάλλοντα).

Service

Τα Services [6] αποτελούν διαμόρφωση ομαδοποίησης συνήθως όμοιων Pods (replicas) μίας εφαρμογής. Με αυτό τον τρόπο, το σύνολο των ομαδοποιημένων pods αντιστοιχίζεται με μία Service IP. Όμως, θα ήταν παράλογο η αναφορά σε ένα service να πραγματοποιείται με την χρήση της IP, κατά την ανάπτυξη του κώδικα της εφαρμογής. Για τον λόγο αυτό στο Kubernetes, η αναφορά στα Services γίνεται με την χρήση του ονόματός τους. Το όνομα ύστερα γίνεται επίλυση (resolve) σε Service IP. Επομένως το

Service name (`metadata.name`), αντιστοιχίζεται πλήρως με την Service IP. Επιπλέον, σε μία διαμόρφωση Service, πραγματοποιείται δήλωση στοιχείων επιπέδου μεταφοράς για την προώθηση των αιτημάτων στις κατάλληλες pod πόρτες. Η θύρα του Service ορίζεται ως `port` (`spec. ports.port`), ενώ η θύρα του pod στην οποία προωθούνται τα αιτήματα, ονομάζεται `targetPort` (`spec.ports.targetPort`). Παράλληλα με τις πόρτες, δηλώνεται και το πρωτόκολλο (`spec.ports.protocol`) για την μεταφορά ων δεδομένων μεταξύ των pods και της υπηρεσίας (Service). Στο Kubernetes υποστηρίζονται τα πρωτόκολλα μεταφοράς, SCTP, TCP (default), και UDP [7].

Προηγουμένως ορίστηκε ο τρόπος αναφοράς σε ομάδες – υπηρεσίες από pods. Όμως, με ποιο τρόπο ομαδοποιούνται τα pods; Για την ομαδοποίηση των επιθυμητών Pods σε μία διαμόρφωση Service, γίνεται χρήση ενός `selector` (`spec.selector.app`) με το οποίο επιλέγονται τα pods με το αντίστοιχο `app` label. Όμως, σε διαμορφώσεις, όπως DestinationRule και VirtualService, η αναφορά σε ένα Service γίνεται με την χρήση του FQDN, όταν π.χ. αποτελεί προορισμό σε έναν κανόνα δρομολόγησης. Η τιμή της Service IP γίνεται resolve, συντάσσοντας τον προορισμό με το FQDN: `reviews.prod.svc.cluster.local`. Όπου `reviews` είναι η ονομασία της υπηρεσίας, `prod` το περιβάλλον εφαρμογής και τέλος `svc.cluster.local` αποτελεί μία σειρά από attributes τα οποία περιγράφουν την τοπική IP διεύθυνση της υπηρεσίας συμπλέγματος.

ClusterIP Service

Εάν σε μία διαμόρφωση υπηρεσίας δεν δηλωθεί ο τύπος της, τότε από προεπιλογή χαρακτηρίζεται ως ClusterIP. Οι ClusterIP υπηρεσίες δεν μπορούν να δεχθούν εξωτερική κίνηση. Εναλλακτικά, είναι γνωστές και ως εσωτερικές υπηρεσίες (internal services) και δέχονται κίνηση μόνο εντός του cluster. Αυτό σημαίνει ότι τους αποδίδεται μια εσωτερική διεύθυνση IP, η οποία χρησιμοποιείται για τη δρομολόγηση των downstream αιτημάτων, προς εκείνη την υπηρεσία. Όταν άλλα στοιχεία (όπως pods ή υπηρεσίες) χρειάζεται να επικοινωνήσουν με Pods που ανήκουν σε μια συγκεκριμένη υπηρεσία, χρησιμοποιούν την διεύθυνση IP που την αντιπροσωπεύει. Κατά τη δημιουργία μίας υπηρεσίας δηλαδή, εκχωρείται μια τοπική διεύθυνση IP, η οποία χρησιμοποιείται κατά την δρομολόγηση αιτημάτων στο δικτύωμα του συμπλέγματος.

Nodeport Service

Μια Nodeport υπηρεσία, σε αντίθεση με την Cluster IP, μπορεί να δεχθεί εξωτερική κίνηση. Ο τρόπος με τον οποίο επιτυγχάνεται αυτό, είναι εκθέτοντας μία πόρτα κόμβου (Nodeport), από την οποία διέρχονται τα πακέτα επιπέδου μεταφοράς (segments, datagrams). Οποιαδήποτε εξωτερική κίνηση φτάνει σε αυτή τη πόρτα του κόμβου, κατευθύνεται προς την αντίστοιχη πόρτα της υπηρεσίας και ύστερα προωθείται στην πόρτα του Pod. Ωστόσο, η πρακτική αυτή αποτελεί έναν μη ασφαλή τρόπο χειρισμού της εξωτερικής κίνησης προς μία υπηρεσία. Για τον λόγο αυτό, χρησιμοποιείται κυρίως στο στάδιο ανάπτυξης μιας

εφαρμογής για την εκτέλεση γρήγορων δοκιμών. Καλύτερη πρακτική αποτελεί η δημιουργία ενός Ingress Gateway πόρου, ο οποίος θα μεσολαβεί στην εξωτερική εισερχόμενη κίνηση.

StatefulSets

Οι εφαρμογές με κατάσταση (stateful) αποτελούν μια κατηγορία εφαρμογών, στην οποία διατηρούνται μόνιμα δεδομένα ή κατάσταση καθ' όλη τη διάρκεια της λειτουργίας. Αντίθετα, στις εφαρμογές χωρίς κατάσταση, κάθε λειτουργία αντιμετωπίζεται ανεξάρτητα, δίχως να υπάρχει ανάγκη αποθήκευσης δεδομένων τοπικά. Για παράδειγμα, σε μια εφαρμογή χωρίς κατάσταση όπως ένας διακομιστής ιστού, εξυπηρετείται στατικό περιεχόμενο και δεν απαιτείται γνώση των προηγούμενων αιτήσεων, καθιστώντας εύκολη την κλιμάκωσή του ή την αντικατάσταση, δίχως ανησυχίες σχετικά με τη διατήρηση συγκεκριμένων καταστάσεων. Σε ένα σύστημα διαχείρισης βάσεων δεδομένων όπως η MySQL, όπου η ακεραιότητα των δεδομένων και οι σχέσεις μεταξύ των εγγραφών είναι κύριας σημασίας, εμφανίζονται σημαντικές προκλήσεις κλιμάκωσης. Όπως και στην περίπτωση μιας stateful εφαρμογής η οποία αποτελείται από ένα σύστημα ανταλλαγής μηνυμάτων μεταξύ υπηρεσιών όπως το RabbitMQ, που βασίζεται στη διατήρηση της σειράς αποστολής μηνυμάτων στις ουρές. Το ερώτημα το οποίο δημιουργείται σε αυτή την περίπτωση, έγκειται στον λόγο κλιμάκωσης των stateful εφαρμογών. Γιατί είναι σημαντική η οριζόντια κλιμάκωση μιας stateful εφαρμογής; Όπως και στις stateless εφαρμογές έτσι και στις stateful, η αύξηση του φόρτου, όπως και η διατήρηση υψηλής διαθεσιμότητας αποτελούν τις κύριες αιτίες κλιμάκωσης.

Η πρώτη πρόκληση που εμφανίζεται, αναφέρεται στην αποθήκευση των δεδομένων κατάστασης της εφαρμογής. Η αποθήκευση των δεδομένων στον σκληρό δίσκο του κόμβου συμπλέγματος καθίσταται ανεπαρκής, ειδικά σε εφαρμογές μεγάλης κλίμακας. Αυτή η προσέγγιση μπορεί να οδηγήσει σε απώλεια δεδομένων, σε περίπτωση αποτυχίας (failure) του κόμβου. Η λύση του προβλήματος έγκειται στην χρήση εξωτερικών κατανεμημένων περιβαλλόντων αποθήκευσης, με τα οποία παρέχονται μόνιμοι όγκοι στα pods μιας stateful εφαρμογής. Η πρόσβαση στα δεδομένα γίνεται, σαν να ήταν αποθηκευμένα τοπικά στο pod. Οι όγκοι προσαρτώνται στα pods είτε ως block devices, είτε ως filesystem [8] στην πλατφόρμα Kubernetes. Μερικές υπηρεσίες νέφους με τις οποίες προσφέρονται μόνιμοι όγκοι αποθήκευσης είναι η Amazon EBS, AzureDisk, GCEPersistentDisk και Ceph (opnesource). Με την χρήση μόνιμων όγκων αποθήκευσης τα δεδομένα δεν επηρεάζονται από τυχόν σφάλματα των pods, του κόμβου, αλλά ούτε και του συμπλέγματος. Για εφαρμογές μικρής κλίμακας ή δοκιμής, όπου η ανεκτικότητα σε σφάλματα δεν αποτελεί σημαντική προτεραιότητα, μπορούν να εφαρμοστούν λύσεις όπως εκείνης της δημιουργίας όγκων χρησιμοποιώντας τον αποθηκευτικό χώρο του κόμβου. Στην πλατφόρμα Kubernetes, η δημιουργία Stateful εφαρμογών πραγματοποιείται με την χρήση του πόρου StatefulSets. Η προσάρτηση πηγών αποθηκευτικού χώρου στα pods γίνεται με την δήλωση του pod μονοπατιού στο πεδίο volumeMounts.mountPath, ενώ η δημιουργία

μόνιμων όγκων γίνεται με την χρήση volumeClaimTemplates ή peisistentVolumeClaimTemplates. Η επιλογή volumeClaimTemplates παρέχει μία απλούστερη πηγή διαμόρφωσης των όγκων και χρησιμοποιείται κυρίως για χρήση τοπικού αποθηκευτικού χώρου του κόμβου. Ενώ στην επιλογή persistentVolumeClaimTemplates [9] γίνεται χρήση κλάσεων αποθηκευτικού χώρου (StorageClass) για χρήση απομακρυσμένων εξωτερικών πηγών αποθήκευσης νέφους. Στην εφαρμογή της μουσικής βιβλιοθήκης, η υλοποίηση των stateful βάσεων δεδομένων πραγματοποιήθηκε με την χρήση των volumeClaimTemplates, με τα οποία δημιουργήθηκε ένας χώρος αποθήκευσης σε κάθε βάση δεδομένων (track και playlist mongodb).

Μία άλλη πρόκληση που εμφανίζεται κατά την κλιμάκωση των stateful εφαρμογών, έγκειται στην δικτυακή επικοινωνία μεταξύ των διαφόρων στιγμιότυπων της εφαρμογής. Έστω ότι τα στιγμιότυπα μιας βάσης δεδομένων mongodb, αποτελούν τα μέλη σε ένα mongodb replication cluster. Σε μία τέτοια περίπτωση, η ταυτόχρονη πραγματοποίηση ενεργειών, όπως create, update, delete, patch, μπορεί να οδηγήσει σε καταστροφή δεδομένων, ειδικά όταν διαμοιράζεται ο ίδιος χώρος αποθήκευσης των δεδομένων σε όλα τα στιγμιότυπα. Σε ένα mongo σύμπλεγμα, κάθε στιγμιότυπο χρησιμοποιεί ξεχωριστό όγκο αποθήκευσης, στον οποίο αντιγράφονται τα κύρια δεδομένα από το primary στιγμιότυπο [10]. Ο primary κόμβος, εκτελεί λειτουργίες CRUD, ενώ οι secondary κόμβοι χρησιμοποιούνται για Read λειτουργίες. Επομένως, με την χρήση αυτού του μηχανισμού επιλύονται οι συγκρούσεις σε ένα κατανεμημένο σύμπλεγμα mongo. Όπως και στην περίπτωση του mongo, έτσι και σε άλλους τύπους βάσεων δεδομένων υποστηρίζεται κατανεμημένη υλοποίησης μιας βάσης δεδομένων (π.χ. Cassandra, DynamoDB, CockroachDB κ.α.). Όμως, ο μηχανισμός της κατανεμημένης υλοποίησης στις βάσεις δεδομένων δεν θα μπορούσε να λειτουργήσει, εάν δεν ήταν δυνατή η δικτυακή επικοινωνία μεταξύ των replica pods σε ένα Kubernetes σύμπλεγμα. Γνωρίζουμε ότι κατά την δημιουργία των pods στην πλατφόρμα Kubernetes, το όνομα τομέα εντός του δικτύου συμπλέγματος, περιέχει ένα τυχαίο hash. Κάτι τέτοιο καθιστά δύσκολη έως αδύνατη την επικοινωνία μεταξύ των replicas, όταν τα Pods εύκολα διαγράφονται ή αντικαθίστανται. Όπως οι διαμορφώσεις Service παρέχουν μία σταθερή ταυτότητα δικτύου ομαδοποιώντας τα pods μιας υπηρεσίας, έτσι και σε αυτή την περίπτωση κάτι αντίστοιχο χρειάζεται να υλοποιηθεί σε επίπεδο pod. Επομένως, όταν δημιουργείται ένα replica set από pods μίας stateful εφαρμογής, αποδίδεται ένα συγκεκριμένο όνομα τομέα σε κάθε pod [11]. Για παράδειγμα έστω ότι σε ένα Mongo σύμπλεγμα υπάρχουν N replica pods, τότε τα ονόματα τομέα θα ανήκουν στο σύνολο {mongo-0, mongo-1, …, mongo-N}, καθιστώντας εύκολη την επικοινωνία εντός του mongo συμπλέγματος. Συνεπώς, το όνομα τομέα των Pods ακολουθεί συγκεκριμένο μοτίβο στα StatefulSets διευκολύνοντας την επικοινωνία εντός των κατανεμημένων συμπλεγμάτων βάσεων δεδομένων, σε αντίθεση με τους πόρους Deployment της πλατφόρμας Kubernetes.

ConfigMap

Στην πλατφόρμα Kubernetes οι ConfigMap πόροι [12] παίζουν σημαντικό ρόλο στην αποθήκευση δεδομένων εκτός του κώδικα της εφαρμογής μιας υπηρεσίας. Συγκεκριμένα η χρήση τους εμπίπτει στον διαχωρισμό δεδομένων διαμόρφωσης από του περιέκτες εφαρμογών, παρέχοντας ευελιξία στην ανάπτυξη και συντήρηση. Για παράδειγμα σε περίπτωση που χρειάζεται κάποια αλλαγή ενός URL ή κάποιου μονοπατιού στον κώδικα της εφαρμογής, η πραγματοποίηση της αλλαγής περιορίζεται στην διαμόρφωση του ConfigMap πόρου και όχι στον κώδικα της εφαρμογής. Με αυτό τον τρόπο οι αλλαγές μπορούν να εφαρμόζονται κατά τον χρόνο εκτέλεσης (runtime), δίχως την ανάγκη δημιουργίας (build) νέας εικόνας περιέκτη. Τα δεδομένα του ConfigMap πόρου αποθηκεύονται σε μορφή key-value ζευγαριών είτε ως properties, είτε ως αρχεία, τα οποία προσαρτώνται στους περιέκτες ως όγκοι (volumes). Σε μία έρευνα που πραγματοποιήθηκε το 2019 [13], κατά την ανάπτυξη μίας εφαρμογής μικροϋπηρεσιών, η ομάδα ανάπτυξης με την χρήση του πόρου ConfigMap και ενός remote repository επωφελήθηκε από το την γρήγορη και ευέλικτη επεξεργασία των δεδομένων διαμόρφωσης. Παρόμοια χρήση ενός ConfigMap πόρου έγινε για την χρήση δεδομένων διαμόρφωσης στην υπηρεσία frontend της μουσικής βιβλιοθήκης που υλοποιήθηκε στα πλαίσια της εργασίας. Περισσότερα στοιχεία παρουσιάζονται στο Κεφάλαιο 6: «Ανάπτυξη Εφαρμογής Μικροϋπηρεσιών».

Secrets

Όπως και στην περίπτωση των ConfigMap πόρων, έτσι και οι Secret πόροι χρησιμοποιούνται ως πηγή αποθήκευσης δεδομένων. Όμως στους Secret πόρους [14], τα δεδομένα αποτελούν ευαίσθητες πληροφορίες διαμόρφωσης, όπως API tokens, cridentials και κλειδιά κρυπτογράφησης. Με παρόμοιο τρόπο επίσης, γίνεται η προσάρτηση των Secret πόρων στους περιέκτες των εφαρμογών ως όγκοι (volumes). Αρχικά, τα δεδομένα δηλώνονται σε base64 κωδικοποίηση εντός των Secret πόρων. Στην συνέχεια μπορεί να εφαρμοστεί κρυπτογράφηση των δεδομένων [15] για την αποθήκευσή τους εντός της etcd βάσης δεδομένων της πλατφόρμας Kubernetes. Με την κρυπτογράφηση εξασφαλίζεται εμπιστευτικότητα και μυστικότητα των ευαίσθητων πληροφοριών σε περιπτώσεις μη εξουσιοδοτημένης πρόσβασης. Όσον αφορά την εφαρμογή της μουσικής βιβλιοθήκης δημιουργήθηκε Secret πόρος για την αποθήκευση του ψηφιακού πιστοποιητικού και του ιδιωτικού κλειδιού του Ingress Gateway. Περισσότερα στοιχεία παρουσιάζονται στο Κεφάλαιο 6: «Ανάπτυξη Εφαρμογής Μικροϋπηρεσιών».

Role Based Access Control

Η Διαχείριση Πρόσβασης με βάση τον Ρόλο (Role-Based Access Control - RBAC) εφαρμόζεται στο Kubernetes παρέχοντας έναν αξιόπιστο και λεπτομερή μηχανισμό ασφάλειας, για τη διαχείριση της πρόσβασης σε διάφορους πόρους εντός του συμπλέγματος. Η λειτουργία RBAC επιτρέπει στους διαχειριστές να ορίζουν και να ελέγχουν τις άδειες μεμονωμένων χρηστών, ομάδων χρηστών και υπηρεσιών, εξασφαλίζοντας εξουσιοδοτημένη πρόσβαση στο Kubernetes API. Αυτό είναι ιδιαίτερα

σημαντικό σε περιβάλλοντα μεγάλης κλίμακας, με πολλαπλούς χρήστες και υπηρεσίες, στα οποία απαιτούνται διαφορετικά επίπεδα πρόσβασης. Το RBAC συνήθως εφαρμόζεται κατά τη διάρκεια της ανάπτυξης του συμπλέγματος ή όταν οι απαιτήσεις για τον έλεγχο της πρόσβασης εξελίσσονται, καθώς αυξάνονται οι υπηρεσίες και οι χρήστες.

Για τον έλεγχο πρόσβασης χρηστών στους διάφορους πόρους του συμπλέγματος γίνεται χρήση ψηφιακών πιστοποιητικών. Η ταυτοποίηση των χρηστών σε μη διαχειριζόμενα περιβάλλοντα, δεν γίνεται μέσω καθιερωμένων μηχανισμών Log-in, αλλά με την επικύρωση ψηφιακών πιστοποιητικών, τα οποία είναι υπογεγραμμένα από την Αρχή πιστοποίησης (CA - Certificate Authority) του Kubernetes. Για την δημιουργία ενός ψηφιακού πιστοποιητικού χρήστη, πρώτα σειρά έχει η παραγωγή του ιδιωτικού κλειδιού χρήστη και ενός Αιτήματος Υπογραφής Πιστοποιητικού (CSR – Certificate Signing Request) [16]. Το αίτημα περιέχει το δημόσιο κλειδί του χρήστη καθώς και το όνομά του (CN – Common Name) ή και την ομάδα (O – Organization) στην οποία ανήκει. Η υπογραφή του αιτήματος CSR, γίνεται με την χρήση του ιδιωτικού κλειδιού της Αρχής Πιστοποίησης του Kubernetes και του πιστοποιητικού της, έτσι ώστε τελικά να προκύψει το ψηφιακό πιστοποιητικό του χρήστη (.crt αρχείο). Σημαντικός είναι ο ορισμός διάρκειας ισχύος του πιστοποιητικού χρήστη, για την αποφυγή της κατάχρησής του. Το πιστοποιητικό και το ιδιωτικό κλειδί της Αρχής Πιστοποίησης συνήθως βρίσκεται στο μονοπάτι /etc/kubernetes/pki/ εντός του Master Node. Έπειτα, ακολουθεί η αποθήκευση του πιστοποιητικού χρήστη στο αρχείο config του Master Node. Το αρχείο αυτό περιέχει εγγραφές συμπλέγματος και εγγραφές χρηστών. Οι εγγραφές συμπλέγματος αποτελούνται από τα δεδομένα της Αρχής Πιστοποίησης (ψηφιακό πιστοποιητικό), την url διεύθυνση του Kubernetes API και το όνομα του cluster (π.χ. dev-cluster, prod-cluster). Ενώ, στις εγγραφές χρηστών περιέχεται το ψηφιακό πιστοποιητικό, το δημόσιο κλειδί ενός χρήστη και το όνομα της εγγραφής χρήστη. Οι αντιστοιχίσεις χρηστών με συμπλέγματα ορίζονται εντός του config αρχείου, στις εγγραφές context [17]. Οι εγγραφές context περιέχουν το όνομα της εγγραφής συμπλέγματος και το όνομα της εγγραφής χρήστη. Με αυτό τον τρόπο, οι σχέσεις μεταξύ χρηστών και συμπλεγμάτων, είναι αποθηκευμένες σε ένα τοπικό αρχείο config στο default μονοπάτι που ορίζεται από το Kubernetes API. Η ταυτοποίηση λαμβάνει χώρα, όταν ο χρήστης αποστείλει αίτημα στο Kubernetes API για την χρήση κάποιου context (σύμπλεγμα και χώρο ονόματος), μέσω της kubectl config utility. Επομένως με αυτό τον μηχανισμό, ο διαχειριστής, είναι σε θέση να διαμοιράσει τα config αρχεία στους χρήστες για την χρήση των αντίστοιχων contexts.

Στην προηγούμενη παράγραφο αναλύθηκε ο τρόπος, με τον οποίο ταυτοποιούνται οι χρήστες για την χρήση κάποιου context στην πλατφόρμα Kubernetes. Στα contexts περιέχονται διάφοροι πόροι του Kubernetes, όπως pods, ConfigMaps, Gateways κ.α. Για τον χειρισμό εκείνων των πόρων, θα πρέπει να έχουν αποδοθεί στους χρήστες οι κατάλληλες άδειες (permissions). Η παραπάνω λειτουργικότητα υλοποιείται μέσω του μηχανισμού RBAC, με τον οποίο επιτρέπεται στους χρήστες η εκτέλεση

συγκεκριμένων ενεργειών σε συγκεκριμένους πόρους. Ο μηχανισμός RBAC βασίζεται σε τέσσερις διαμορφώσεις, οι οποίες περιέχουν τον ορισμό των ενεργειών που επιτρέπεται να εκτελεστούν, καθώς και την αντιστοίχιση αυτών με συγκεκριμένους χρήστες. Η πρώτη διαμόρφωση ονομάζεται Role, με την οποία ορίζονται οι ενέργειες ή αλλιώς verbs και οι πόροι – resources, για τους οποίους θα ισχύσουν τα verbs. Στα verbs μπορούν να αποδοθούν τιμές όπως get, watch, list, create, delete. Αξίζει να σημειωθεί ότι η διαμόρφωση Role αναφέρεται στον καθορισμό ενεργειών για πόρους, σε εμβέλεια χώρου ονόματος. Αντιθέτως, η διαμόρφωση ClusterRole συντάσσεται με τον ίδιο τρόπο, μόνο που παραλείπεται το πεδίο του χώρου ονόματος. Με αυτό τον τρόπο η ισχύς των verbs επεκτείνεται σε επίπεδο συμπλέγματος, συμπεριλαμβάνοντας τους πόρους όλων των χώρων ονόματος. Τέλος, οι αντιστοιχίσεις των διαμορφώσεων Role και ClusterRole σε χρήστες, πραγματοποιούνται στις διαμορφώσεις RoleBinding και ClusterRoleBinding. Συγκεκριμένα, κάτω από το πεδίο subjects, αναφέρονται τα αντικείμενα (subjects) με τα οποία πραγματοποιείται η αναγνώριση ενός χρήστη. Τα subjects αποτελούν κριτήριο ταιριάσματος με εκείνα που βρίσκονται αποθηκευμένα στα δεδομένα του ψηφιακού πιστοποιητικού χρήστη (π.χ. Common Name και Organization). Αμέσως μετά, ορίζονται οι αναφορές (roleRef) στις διαμορφώσεις Role ή ClusterRole, οι οποίες θα ισχύσουν για τους χρήστες μετά το ταίριασμα. Ολοκληρώνοντας τις δύο προηγούμενες παραγράφους, έγιναν κατανοητές οι έννοιες της ταυτοποίησης και της εξουσιοδότησης χρηστών για τον χειρισμό συγκεκριμένων πόρων του Kubernetes. Στην επόμενη παράγραφο γίνεται αναφορά στον τρόπο εξουσιοδότησης υπηρεσιών για τον χειρισμό των Kubernetes πόρων μέσω του Kube-api.

Η εξουσιοδότηση υπηρεσιών για τον χειρισμό πόρων μέσω του kube-api, δεν αποτελεί τόσο συχνό φαινόμενο στην ανάπτυξη εφαρμογών με την αρχιτεκτονική των μικροϋπηρεσιών. Παρ' όλ' αυτά, πιθανά σενάριο μπορεί να αποτελέσει η χρήση opensource εργαλείων που απαιτούν σύνδεση με το kube-api, η ανάπτυξη εργαλείων αυτόματης κλιμάκωσης και η υλοποίηση προσαρμοσμένων Kubernetes webHooks. Σε τέτοιες περιπτώσεις η εξουσιοδότηση υπηρεσιών γίνεται με την χρήση των προηγούμενων Role / ClusterRole και RoleBinding / ClusterRoleBinding διαμορφώσεων [18]. Μόνο που σε αυτή την περίπτωση, η αναφορά σε υπηρεσίες κατά το ταίριασμα (binding) γίνεται με την χρήση ServiceAccount αντικειμένων. Επομένως για μία εφαρμογή η οποία απαιτεί πρόσβαση σε συγκεκριμένους πόρους, με σκοπό την εκτέλεση ορισμένων ενεργειών, θα πρέπει να δημιουργηθεί και η αντίστοιχη ServiceAccount διαμόρφωση. Στην συνέχεια, στις Binding διαμορφώσεις ορίζεται ένα subject τύπου (kind) SeviceAccount, με το αντίστοιχο όνομα (name) του ServiceAccount. Όσον αφορά το Istio, το RBAC χρησιμοποιείται επίσης για τον έλεγχο της πρόσβασης σε πόρους καθώς και τον έλεγχο των λειτουργιών που μπορούν να εκτελέσουν. Το Istio επεκτείνει την λειτουργικότητα του RBAC στο Kubernetes για να παρέχει επιπλέον δυνατότητες ελέγχου πρόσβασης σε υπηρεσίες, σε workloads καθώς και σε συγκεκριμένους χώρους ονόματος, όπως θα δούμε στο Κεφάλαιο 5: «Χαρακτηριστικά του Istio».

ΚΕΦΑΛΑΙΟ 3: ΕΠΙΠΕΔΟ ΔΕΔΟΜΕΝΩΝ ΣΤΟ ISTIO

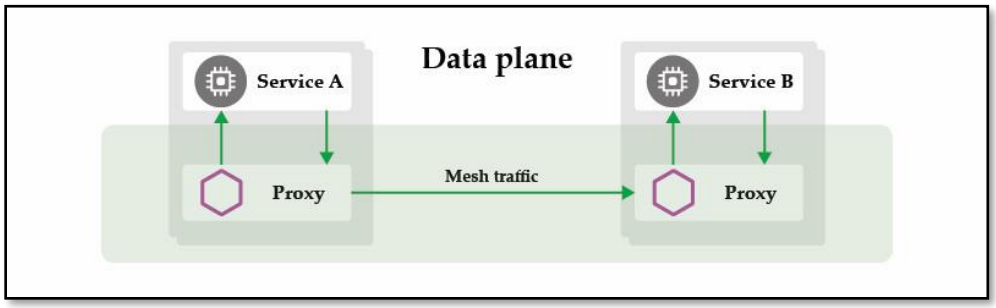
Η συγγραφή αυτού του κεφαλαίου στοχεύει στην κατανόηση της δομής του επιπέδου δεδομένων στην πλατφόρμα Istio. Λειτουργίες δρομολόγησης, μηχανισμοί ασφάλειας και πρόσθετες πολιτικές του Istio, εφαρμόζονται στο επίπεδο δεδομένων (data plane) αποτελεσματικά, μέσω ειδικών διακομιστών μεσολάβησης. Παρακάτω λοιπόν, ακολουθεί επεξήγηση των βασικών στοιχείων του επιπέδου δεδομένων, όπως Sidecar container και Envoy proxy, καθώς και ανάλυση της λειτουργικότητάς τους. Με αφορμή την εισαγωγή στην πτυχή του Istio, ακολουθεί μία μικρή αναδρομή της πορείας του.

3.1 Η πορεία του Istio

Το Istio είναι μια πλατφόρμα πλέγματος υπηρεσιών ανοικτού κώδικα που ανακοινώθηκε για πρώτη φορά τον Μάιο του 2017 από την Google [19], την IBM και τη Lyft. Το έργο αρχικά, εισήχθη για την απλοποίηση της αυξανόμενης πολυπλοκότητας στις αρχιτεκτονικές μικροϋπηρεσιών. Σε αντίθεση με τις μονολιθικές εφαρμογές, στην ανάπτυξη μικροϋπηρεσιών, λειτουργίες ασφάλειας, αυθεντικοποίησης, εξουσιοδότησης, διαχείρισης ρυθμού αφικνούμενων αιτημάτων και δημιουργίας αυτόνομων διαδρομών (routes) υλοποιούνται σε πλήθος υπηρεσιών. Όσο αυξάνονται οι υπηρεσίες, η ενσωμάτωση των λειτουργιών αυτών στον πηγαίο κώδικα κάθε υπηρεσίας ξεχωριστά καθίσταται αδύνατη. Με την χρήση των τεχνολογιών πλέγματος, όπως το Istio, η λογική της εφαρμογής διασπάται από εκείνες τις λειτουργίες, παρέχοντας ένα ξεχωριστό επίπεδο χειρισμού και ελέγχου. Οι λειτουργίες των τεχνολογιών πλέγματος βασίζονται στο επίπεδο μεταφοράς και εφαρμογής κατά OSI, παρέχοντας την δυνατότητα ομαδοποιημένης διαμόρφωσης πολλαπλών υπηρεσιών με την χρήση απλών μοντέλων. Από την ίδρυσή του, το Istio, έχει υποστεί σημαντική ανάπτυξη μέσα από τις διάφορες ενημερώσεις και εκδόσεις που έχουν ανακοινωθεί κατά την πορεία του. Για παράδειγμα, τον Ιούλιο του 2018, ανακοινώθηκε η πρώτη έκδοση 1.0, καθιστώντας το Istio έτοιμο για χρήση σε εφαρμογές μικροϋπηρεσιών. Στη συνέχεια, τον Νοέμβριο του 2020, κυκλοφόρησε η έκδοση 1.8 [20], στην οποία εισήχθησαν ενισχυμένες δυνατότητες διαχείρισης της κυκλοφορίας και βελτιωμένη τηλεμετρία. Το έργο συνέχισε να εξελίσσεται, περνώντας στην έκδοση 1.11 [21], η οποία κυκλοφόρησε τον Αύγουστο του 2021. Αυτή η έκδοση επικεντρώθηκε σε περαιτέρω βελτιώσεις ασφάλειας, σταθεροποίησης και εκτενής υποστήριξη της πλατφόρμας σε υβριδικά περιβάλλοντα cloud. Με τη συνεχή ανάπτυξη και την αυξανόμενη υποστήριξη της κοινότητας, το Istio παραμένει στην πρώτη γραμμή της τεχνολογίας πλέγματος, παρέχοντας μια ισχυρή λύση για τη διαχείριση αρθρωτών υπηρεσιών. Σήμερα το Istio βρίσκεται στην έκδοση 1.18 (Ιούνιος 2023) [22], όπου και συνεχώς να εξελίσσεται.

3.2 Sidecar περιέκτης

Στο πλαίσιο της ενορχήστρωσης υπηρεσιών, ένας sidecar περιέκτης αποτελεί ένα συμπληρωματικό περιέκτη (container) που εκτελείται παράλληλα με τον κύριο περιέκτη της εφαρμογής. Σε ένα pod μπορεί να εκτελούνται πολλαπλοί περιέκτες, παρ' όλ' αυτά το μέγιστο πλήθος των συνεκτελούμενων περιεκτών καθορίζεται βάση των πόρων που του έχουν ανατεθεί. Η YAML διαμόρφωση πολλαπλών pods, όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, καλείται Deployment. Μέσα από εκείνη καθορίζονται μεταξύ άλλων και οι πόροι ανάθεσης (όπως RAM, CPU). Ο ρόλος του sidecar περιέκτη είναι υποβοηθητικός και συνήθως εκτελεί λειτουργίες μεσολάβησης, διαχωρίζοντας τις διαδικασίες επικοινωνίας μεταξύ υπηρεσιών από την εφαρμογή, όπως φαίνεται παρακάτω.



Εικόνα 1: Επίπεδο δεδομένων του Πλέγματος Υπηρεσιών

Σε άλλη περίπτωση ένας sidecar περιέκτης (όπως Grafana container) μπορεί να υλοποιεί πρόσθετες λειτουργίες, όπως καταγραφή και παρακολούθηση της κατάστασης του περιέκτη εφαρμογής. Στην συνηθισμένη περίπτωση, ο κύριος περιέκτης του pod χειρίζεται τη λειτουργικότητα της εφαρμογής, ενώ ο περιέκτης sidecar λειτουργεί ως ενδιάμεσος διακομιστής (proxy) ή μεσάζων. Αυτό επιτρέπει την απομόνωση των δικτυακών εργασιών, όπως την δρομολόγηση επιπέδου εφαρμογής, καθιστώντας την υλοποίηση των υπηρεσιών απλούστερη. Η υιοθέτηση αυτού του μοντέλου παρέχει προηγμένες πολιτικές δικτύου (όπως rate limiting και circuit breaking), διευκολύνοντας τον χειρισμό της κίνησης εντός του πλέγματος.

3.3 Envoy Proxy

Ο περιέκτης Envoy χαρακτηρίζεται ως ένας διακομιστής μεσολάβησης (Layer 7 proxy) υψηλής απόδοσης, ανοικτού κώδικα που αναπτύχθηκε από τη Lyft (2016) [23] και υιοθετήθηκε ευρέως σε αρχιτεκτονικές πλέγματος υπηρεσιών όπως το Istio. Σε ένα Istio πλέγμα υπηρεσιών, από προεπιλογή, σε κάθε pod εγχέεται ένας διακομιστής μεσολάβησης Envoy ως sidecar περιέκτης (Envoy proxy sidecar injection). Η λειτουργία αυτή μπορεί να απενεργοποιηθεί αποδίδοντας στο πεδίο ISTIO_META_SIDECAR_INJECTION (meshconfig.yaml, istio-system namespace) την τιμή "disabled". Επίσης η ενεργοποίηση της έγχυσης των διακομιστών μεσολάβησης Envoy μπορεί να πραγματοποιηθεί σε

επίπεδο χώρου ονόματος μέσω της kubectl utility με την εκτέλεση της εντολής: `microk8s kubectl label namespace default istio-injection=enabled –overwrite`
Μετά την υλοποίηση της εφαρμογής στην πλατφόρμα Istio, ο έλεγχος της εκτέλεσης των Envoy διακομιστών μπορεί να γίνει με την εκτέλεση της εντολής: `microk8s istioctl proxy-status`
Συγκεκριμένα με εκείνη την εντολή εμφανίζονται όλοι οι Envoy sidecar περιέκτες οι οποίοι εκτελούνται παράλληλα με τους κύριους περιέκτες εφαρμογής στον default χώρο ονόματος (default namespace).

Μετά την εκτέλεση της εντολής, εμφανίζονται όλοι οι Envoy proxies της εφαρμογής που υλοποιήθηκε στο πλαίσιο της εργασίας.

NAME	CLUSTER	CDS	LDS	EDS	RDS
frontend-5fc5d9dd88-ktjsl.default	Kubernetes	SYNCED	SYNCED	SYNCED	SYNCED
istio-egressgateway-6b957f5b7d-cmv82.istio-system	Kubernetes	SYNCED	SYNCED	SYNCED	NOT SENT
istio-ingressgateway-5f57c68988-bh2jj.istio-system	Kubernetes	SYNCED	SYNCED	SYNCED	SYNCED
mongodb-playlist-0.default	Kubernetes	SYNCED	SYNCED	SYNCED	SYNCED
mongodb-track-0.default	Kubernetes	SYNCED	SYNCED	SYNCED	SYNCED
playlist-77cb78ccb5-kgxh7.default	Kubernetes	SYNCED	SYNCED	SYNCED	SYNCED
track-664cf89559-v8l7r.default	Kubernetes	SYNCED	SYNCED	SYNCED	SYNCED

Οι εγγραφές αποτελούνται από το όνομα του περιέκτη καθώς και τις x Υπηρεσίες Ανακάλυψης (xDS) με τις οποίες είναι συμβεβλημένοι οι Envoy διακομιστές. Στο επόμενο κεφάλαιο (Κεφάλαιο 4: Επίπεδο Ελέγχου στο ISTIO), πραγματοποιείται ανάλυση της λειτουργικότητας εκείνων των υπηρεσιών.

Μετά την έγχυση (injection), ο Envoy διακομιστής είναι σε θέση να διαχειριστεί την εισερχόμενη/εξερχόμενη κίνηση ενός pod, ή ακόμη και ενός κόμβου - εικονικής μηχανής. Προσομοιώνει τη λειτουργικότητα των πυλών σε ένα πλέγμα υπηρεσιών, αλλά σε μικρότερη κλίμακα. Από προεπιλογή, το Istio επιτρέπει την κυκλοφορία μεταξύ pods εντός του ίδιου χώρου ονομάτων, αλλά και υπηρεσιών του χώρου ονόματος istio-system. Η συμπεριφορά των διαμεσολαβητών Envoy μπορεί να ρυθμιστεί συνολικά ανά χώρο ονομάτων, μέσω της YAML διαμόρφωσης Sidecar του χώρου ονόματος istio-system. Ακόμη, συγκεκριμένες διαμορφώσεις μπορούν να εφαρμοστούν σε ομάδες από proxies εντός ενός χώρου ονόματος, χρησιμοποιώντας ετικέτες φόρτου εργασίας (workload labels). Οι διακομιστές μεσολαβούν στην TCP κίνηση από και προς τον περιέκτη εφαρμογής. Η εισερχόμενη κυκλοφορία προς την εφαρμογή μεταφέρεται από την αντίστοιχη πόρτα του μεσάζοντα διακομιστή, προτού προωθηθεί στη θύρα της εφαρμογής. Η εξερχόμενη κυκλοφορία η οποία προέρχεται από την εφαρμογή, περνά από τον μεσάζοντα διακομιστή, ο οποίος τη δρομολογεί στο κατάλληλο τελικό σημείο (endpoint). Επιπλέον, ένας διακομιστής Envoy μπορεί να μεσολαβήσει στην κυκλοφορία που προορίζεται προς έναν κόμβο (VM). Αυτό επιτυγχάνεται με τον ορισμό ενός proxy listener για τη σύλληψη και προώθηση της TCP κίνησης προς κάποια διεργασία που εκτελείται σε ένα VM. Αυτό επιτρέπει στην κυκλοφορία που προέρχεται από το εξωτερικό του κόμβου και στοχεύει σε ένα συγκεκριμένο υποδίκτυο υπηρεσιών να κατευθυνθεί προς ένα

workload μέσω του μεσολαβητή. Αυτή η λειτουργικότητα θα ήταν χρήσιμη σε σενάρια επικοινωνίας εσωτερικών υπηρεσιών (που δεν εκτίθενται εκτός του κέντρου δεδομένων), οι οποίες υλοποιούνται σε διαφορετικούς κόμβους. Τέλος στις πύλες (gateways) εισόδου/εξόδου (ingress/egress), δεν απαιτείται η έγχυση κάποιου Envoy (Envoy injection), αφού λειτουργούν ως Istio proxy περιέκτες.

ΚΕΦΑΛΑΙΟ 4: ΕΠΙΠΕΔΟ ΕΛΕΓΧΟΥ ΣΤΟ ISTIO

Στο Κεφάλαιο: «Επίπεδο Ελέγχου» περιγράφονται με αναλυτικό τρόπο, οι επιμέρους συνιστώσες που απαρτίζουν εκείνο το επίπεδο του Istio. Γίνεται διερεύνηση των τρόπων διαχείρισης της συνολικής συμπεριφοράς μέσα από το πλήθος διαμορφώσεων που προσφέρονται. Οι διαμορφώσεις αυτές αφορούν κανόνες διαχείρισης της κίνησης, ασφάλειας και άλλες προηγμένες ρυθμίσεις της συμπεριφοράς του πλέγματος, όπως η διακοπή κυκλώματος (circuit breaking). Για τη δημιουργία, τον έλεγχο και τη προώθηση των διαμορφώσεων, το επίπεδο ελέγχου βασίζεται στην λειτουργία και συνεργασία των επιμέρους υπηρεσιών που το αποτελούν. Δηλαδή το Pilot, το Galley και το Citadel.

4.1 Pilot

Το Pilot λοιπόν, αντιπροσωπεύει τον διαχειριστή (manager) του πλέγματος. Μέσα από προτυποποιημένα μοντέλα διαμορφώσεων, είναι σε θέση να εφαρμόζει πολιτικές διαχείρισης της εσωτερικής καθώς και της εισερχόμενης και εξερχόμενης κίνησης στο πλέγμα. Θέτει όρους στην ενδιάμεση συνδεσιμότητα μεταξύ των υπηρεσιών, καθώς και των επιμέρους στιγμιότυπών τους. Η παραπάνω λειτουργικότητα υλοποιείται, συνδυάζοντας τρεις κύριες πηγές διαμόρφωσης: διαμόρφωση πλέγματος, διαμόρφωση δικτύου και ανακάλυψη υπηρεσίας. Η επιβολή των ρυθμίσεων στο επιπέδου δεδομένων πραγματοποιείται μέσα από την διαδικασία προώθησης των διαμορφώσεων προς τους sidecar διακομιστές μεσολάβησης (Envoy Proxies). Μέσα από υπηρεσίες ακρόασης οι διακομιστές είναι σε θέση να ανανεώνουν δυναμικά την κατάσταση της διαμόρφωσής τους. Η δυναμική εφαρμογή των ρυθμίσεων στο επίπεδο δεδομένων είναι αρκετά κρίσιμη, καθώς η συμπεριφορά του πλέγματος θα πρέπει με αποδοτικό και ταχύ τρόπο να προσαρμόζεται, για την διασφάλιση της διαθεσιμότητας των υπηρεσιών.

Pilot: Το μοντέλο διαμόρφωσης

Η διαμόρφωση πλέγματος περιλαμβάνει ρυθμίσεις που είναι στατικές κατά την εγκατάσταση του Istio. Αυτές οι καθολικές ρυθμίσεις επηρεάζουν τη συνολική συμπεριφορά και τα χαρακτηριστικά του πλέγματος υπηρεσιών. Περιλαμβάνει αντικείμενα διαμόρφωσης όπως MeshConfig, ProxyConfig και MeshNetworks. Το MeshConfig [24] είναι ένας προσαρμοσμένος πόρος στο Istio, με τον οποίο καθορίζονται ρυθμίσεις πολιτικών τηλεμετρίας, αλγόριθμους εξισορρόπησης φορτίου και πηγές διαμόρφωσης. Με το MeshConfig, μπορεί να καθοριστεί η λειτουργία mTLS, η κατάσταση επιβολής πολιτικών και η διαμόρφωση παραμέτρων ανίχνευσης, όπως τα ποσοστά δειγματοληψίας, οι κεφαλίδες ανίχνευσης και τα χρονικά όρια (timeouts). Ακόμη, παρέχει προεπιλογές για όλο το σύμπλεγμα που μπορούν να παρακαμφθούν σε επίπεδο χώρου ονόματος ή φόρτου εργασίας. Το ProxyConfig [25], από την άλλη πλευρά, είναι ένα αντικείμενο που χρησιμοποιείται για την προσαρμογή της default συμπεριφοράς των διακομιστών μεσολάβησης (Envoy sidecars) στο επίπεδο των δεδομένων. Επιτρέπει την προσαρμογή των χρονικών ορίων μεταξύ αποτυχημένων αιτημάτων (timeouts), ρυθμίσεις σύνδεσης στους χώρους συγκέντρωσης (pools),

καταγραφή πρόσβασης και διακόπτες κυκλώματος (circuit breakers) σε περίπτωση αποτυχίας κάποιας υπηρεσίας. Συνολικά με αυτό το αντικείμενο του Istio, ελέγχεται η συμπεριφορά του διακομιστή μεσολάβησης βάσει των επιθυμητών απαιτήσεων. Στην συνέχεια, το MeshNetworks [26] είναι ένα αντικείμενο διαμόρφωσης, με το οποίο καθορίζονται τα επιμέρους δίκτυα εντός του πλέγματος. Επίσης, δίνει την δυνατότητα ορισμού των πυλών (gateways) για τα υπάρχοντα δίκτυα του πλέγματος. Τόσο το ProxyConfig όσο και το MeshNetworks αποτελούν μέρος της διαμόρφωσης που παρέχεται στο αντικείμενο MeshConfig. Επομένως μέσω αυτών των αντικειμένων γίνεται ο καθολικός χειρισμός της συμπεριφοράς πλέγματος σε επίπεδο δικτύου, ασφάλειας, διακομιστών μεσολάβησης και άλλων πολύ συγκεκριμένων και προηγμένων ρυθμίσεων πλέγματος.

Ο καθορισμός των ρυθμίσεων δικτύου εφαρμόζεται ακολουθώντας τα μοντέλα διαμορφώσεων ServiceEntry, DestinationRule, VirtualService και Gateway, τα οποία είναι προτυποποιημένα. Συνολικά, αυτές οι διαμορφώσεις δικτύου παρέχουν λεπτομερή έλεγχο της δρομολόγησης της HTTP/TCP κυκλοφορίας, της εξισορρόπησης φορτίου και της ασφάλειας στο πλέγμα. Στο Istio, το μοντέλο VirtualService (v1alpha1) αντικαθιστά τις προηγούμενες ξεχωριστές διαμορφώσεις των RouteRules και DestinationPolicy (v1alpha3). Ομοίως, ο πόρος διαμόρφωσης DestinationRule (v1alpha3) αντικαθιστά τις διαμορφώσεις LoadBalancerSettings και CircuitBreaker (v1alpha1), ενοποιώντας τες σε μία κύρια πηγή διαμόρφωσης. Χαρακτηρίζεται ως ένας ενοποιημένος πόρος για τη διαχείριση των ρυθμίσεων εξισορρόπησης φορτίου, του διαχωρισμού της κυκλοφορίας (βάση προορισμού), και του εντοπισμού ακραίων τιμών. Τέλος, ο πόρος Gateway (v1alpha3) αντικαθιστά τις προηγούμενες διαμορφώσεις Ingress και Egress (v1alpha1), αποτελώντας την κύρια μέθοδο διαμόρφωσης της εισερχόμενης κυκλοφορίας στο πλέγμα. Οι παραπάνω αλλαγές χαρακτηρίζουν το τρέχον μοντέλο διαμόρφωσης δικτύου της πιο πρόσφατης έκδοσης (1.18, 2023) του Istio control plane και αποσκοπούν στην απλοποίηση του μοντέλου διαμόρφωσης και στην μείωση του πλήθους των ξεχωριστών μοντέλων που απαιτούνται για τη διαχείριση της κίνησης. Το Istio παρέχει επίσης διάφορα άλλα μοντέλα για αυθεντικοποίηση, έλεγχο πρόσβασης, εξουσιοδότησης, όπως θα δούμε παρακάτω στο Κεφάλαιο 5: «Χαρακτηριστικά του ISTIO»

Pilot: Διαδικασία προώθησης διαμορφώσεων

Μόλις κατασκευαστεί το μοντέλο διαμόρφωσης, το Pilot λειτουργεί ως το επίπεδο ελέγχου του πλέγματος. Είναι υπεύθυνο για την προώθηση των κατάλληλων διαμορφώσεων στο επίπεδο δεδομένων, δηλαδή στους διακομιστές μεσολάβησης Envoy ή τις πύλες πλέγματος. Η προώθηση των διαμορφώσεων σε εκείνα τα μέρη, επιτυγχάνεται μέσω των υπηρεσιών x Discovery Service (xDS. Οι διαμορφώσεις μεταφέρονται από τους διακομιστές των υπηρεσιών xDS στους Envoy διακομιστές με την χρήση του πρωτοκόλλου gRPC. Όταν ένας διακομιστής μεσολάβησης (π.χ. Envoy) συνδέεται με το Pilot, γνωστοποιεί την ταυτότητά του. Η ταυτοποίηση βασίζεται στις ετικέτες (labels) της σχετικής υπηρεσίας (Service) που

ανήκει ο διακομιστής. Ύστερα το Pilot ομαδοποιεί όλους τους συνδεδεμένους διακομιστές με κριτήριο την ταυτότητά τους και δημιουργεί συγκεκριμένες αποκρίσεις xDS προσαρμοσμένες σε κάθε ομάδα. Αυτές οι αποκρίσεις αντικατοπτρίζουν την τρέχουσα κατάσταση του περιβάλλοντος πλέγματος, καθώς περιέχουν τις απαραίτητες πληροφορίες διαμόρφωσης για κάθε ομάδα. Το Pilot χρησιμοποιεί τέσσερα κύρια αντικείμενα του πρωτοκόλλου xDS για την προώθηση της διαμόρφωσης στους διακομιστές.

Η Υπηρεσία Ανακάλυψης Ακροατών (Listener Discovery Service) είναι υπεύθυνη για τον χειρισμό των ακροατών (listeners) εντός της διαμόρφωσης των διακομιστών Envoy. Οι ακροατές αποτελούν εγγραφές με τις οποίες καθορίζεται ο χειρισμός της εισερχόμενης δικτυακής κίνησης. Με την LDS υπηρεσία παρέχεται ένας μηχανισμός προώθησης των αλλαγών, έτσι ώστε οι εγγραφές των ακροατών να παραμένουν ενημερωμένες στην διαμόρφωση των Envoy proxies. Συγκεκριμένα στις εγγραφές ακροατών, καθορίζονται οι IP διευθύνσεις των υπηρεσιών με τις οποίες εγκαθιδρύονται συνδέσεις, τα επιτρεπόμενα πρωτόκολλα επικοινωνίας που χρησιμοποιούνται για την αποστολή και λήψη αιτημάτων, οι θύρες στις οποίες αποστέλλονται τα αιτήματα προς τις υπηρεσίες και τέλος τα ονόματα τομέων των υπηρεσιών. Τα πρωτόκολλα εφαρμογής και μεταφοράς που χρησιμοποιούνται για την επικοινωνία μέσω ενός listener, μπορεί να είναι HTTP1, HTTP2 και TCP. Για να γίνει καλύτερα αντιληπτή η μορφή των εγγραφών ακροατών, παρακάτω παρουσιάζονται δύο εγγραφές ακροατών του playlist Envoy διακομιστή (εφαρμογή μουσικής βιβλιοθήκης). Η εμφάνιση των ακροατών πραγματοποιείται με την εκτέλεση της εντολής: `microk8s istioctl proxy-config listeners playlist-77cb78ccb5-kgxh7.default`

```
ADDRESS         PORT  MATCH       DESTINATION
10.1.120.147    27017 ALL         Cluster: outbound|27017||mongodb-track.default.svc.cluster.local
10.1.120.186    27017 ALL         Cluster: outbound|27017||mongodb-playlist.default.svc.cluster.local
```

Παρατηρούμε ότι το πεδίο `ADDRESS` αναφέρεται στην τοπική IP διεύθυνση της υπηρεσίας, το πεδίο `PORT` αναφέρεται στην πόρτα της υπηρεσίας, το πεδίο `MATCH` αναφέρεται στα πρωτόκολλα επικοινωνίας που χρησιμοποιούνται για την προώθηση των πακέτων μέσω του listener και το πεδίο `DESTINATION` αναφέρεται στο όνομα τομέα της υπηρεσίας. Επομένως ο playlist Envoy διακομιστής δέχεται αιτήματα οποιουδήποτε πρωτοκόλλου (`ALL`) από τις υπηρεσίες mongodb-track και mongodb-playlist και αποστέλλει αιτήματα προς εκείνες στην πόρτα 27017.

Επίσης, μία σημαντική λειτουργικότητα των listeners, εμπίπτει στην χρήση αλυσίδων Envoy φίλτρων. Τα Envoy φίλτρα αποτελούν στοιχεία με τα οποία εκτελούνται συγκεκριμένες λειτουργίες στις HTTP/TCP ροές, όπως έλεγχο ταυτότητας, δρομολόγηση, τερματισμό TLS, εξισορρόπηση φορτίου, οριοθέτηση ρυθμού ή μετασχηματισμό της κεφαλίδας των πακέτων. Με αυτό τον τρόπο φιλτράρεται η εισερχόμενη κίνηση προς τους Envoy διακομιστές ή τροποποιείται καταλλήλως. Η αλυσίδα φίλτρων αντιπροσωπεύει τη διαδοχική σειρά εφαρμογής αυτών των φίλτρων για κάθε εισερχόμενη σύνδεση. Επομένως μέσα από

την χρήση του LDS, το Pilot διασφαλίζει την κατάλληλη διαμόρφωση ακρόασης μεταξύ των Envoy διακομιστών και των υπηρεσιών, οι οποίες ανακαλύπτονται μέσω της υπηρεσίας CDS (Cluster Discovery Service), η οποία θα αναλυθεί παρακάτω.

Η Υπηρεσία Ανακάλυψης Διαδρομών (Routing Discovery Service) εστιάζει στη διαμόρφωση των πινάκων δρομολόγησης εντός των Envoy διακομιστών μεσολάβησης. Με την υπηρεσία RDS αποστέλλονται RDS αποκρίσεις στους διακομιστές Envoy, οι οποίες περιέχουν πληροφορίες σχετικά με τις διαθέσιμες διαδρομές (routes) προς τις οποίες οι Envoy διακομιστές μπορούν να αποστείλουν αιτήματα. Οι πληροφορίες κατασκευάζονται μέσω των Εικονικών Υπηρεσιών (Virtual Services) και των Κανόνων Προορισμού (Destination Rules) που ορίστηκαν προηγουμένως. Οι Envoy διακομιστές λαμβάνοντας τις αποκρίσεις RDS, τροποποιούν αναλόγως τις εγγραφές των διαδρομών. Οι εγγραφές των διαδρομών περιέχουν πληροφορίες όπως, η πόρτα προορισμού στην οποία αποστέλλονται αιτήματα καθώς και τα ονόματα τομέα της υπηρεσίας. Επίσης σε κάθε εγγραφή περιέχονται και οι διαδρομές (routes), οι οποίες αποτελούν κριτήριο ταιριάσματος για τα αιτήματα. Παρακάτω ακολουθούν μερικές εγγραφές διαδρομών του playlist Envoy διακομιστή. Η εμφάνιση των εγγραφών πραγματοποιείται με την χρήση της εντολής: `microk8s istioctl proxy-config routes playlist-77cb78ccb5-kgxh7.default`

```
NAME        DOMAINS                                     MATCH
3000        frontend, frontend.default + 1 more...      /*
4000        playlist, playlist.default + 1 more...      /*
5000        track, track.default + 1 more...            /*
```

Παρατηρούμε ότι, ο playlist Envoy proxy αποστέλλει προς οποιαδήποτε διαδρομή αιτήματα στις υπηρεσίες frontend, playlist και track. Το σύνολο των εγγραφών διαδρομών, μπορεί να προσομοιαστεί με έναν πίνακα δρομολόγησης επιπέδου εφαρμογής, με τον οποίο γίνεται ταίριασμα των πακέτων εφαρμογής, έτσι ώστε να αποσταλούν στην κατάλληλη πόρτα και υπηρεσία. Επομένως, με την χρήση της υπηρεσίας RDS, το Pilot διασφαλίζει ότι οι διακομιστές μεσολάβησης Envoy, διαθέτουν ενημερωμένες πληροφορίες δρομολόγησης των πακέτων προς της κατάλληλες διαδρομές (routes).

Η Υπηρεσία Ανακάλυψης Συμπλέγματος (Cluster Discovery Service) χειρίζεται την δυναμική ανακάλυψη των συμπλεγμάτων. Τα συμπλέγματα (clusters), στο πλαίσιο της υπηρεσίας CDS, αντιπροσωπεύουν λογικές ομάδες υπηρεσιών προορισμού, προς τις οποίες προωθείται η κίνηση μέσω των διακομιστών μεσολάβησης. Δηλαδή, η έννοια του όρου «σύμπλεγμα» αναφέρεται στις υπηρεσίες (Services) ομαδοποίησης όμοιων τελικών σημείων (endpoints) και όχι στην ομαδοποίηση φυσικών κόμβων (physicals hosts). Η υπηρεσία CDS συνθέτει πληροφορίες σχετικές με τα διαθέσιμα συμπλέγματα, όπως το όνομα τομέα της υπηρεσίας (Fully Qualified Domain Name), τα subsets της υπηρεσίας (βλ. Παράρτημα: Destination Rules), την πόρτα προώθησης αιτημάτων προς την υπηρεσία, την κατεύθυνση και τον τύπο

του cluster. Το CDS είναι υπεύθυνο για την ανακάλυψη τόσο στατικών όσο και δυναμικών συμπλεγμάτων. Τα στατικά συμπλέγματα είναι προκαθορισμένα και αντιπροσωπεύουν συνήθως τις υπηρεσίες που εκτελούνται στο πλέγμα Istio, όπως μία υπηρεσία ανακάλυψης ψηφιακών πιστοποιητικών ή στατιστικών στοιχείων. Ενώ τα δυναμικά συμπλέγματα προέρχονται από την ανακάλυψη υπηρεσίας και αντιπροσωπεύουν πόρους, στους οποίους αλλάζουν δυναμικά τα τελικά σημεία - pods, όπως για παράδειγμα συμβαίνει σε μία υπηρεσία διακομιστή ιστού ή βάσης δεδομένων. Παρακάτω ακολουθούν μερικές εγγραφές της λίστας συμπλεγμάτων που υπάρχουν στην διαμόρφωση του playlist Envoy διακομιστή. Η εμφάνιση των εγγραφών πραγματοποιήθηκε με την εκτέλεση της εντολής:

```
microk8s istioctl proxy-config cluster playlist-77cb78ccb5-kgxh7.default
```

SERVICE FQDN	PORT	SUBSET	DIRECTION	TYPE
mongodb-playlist.default.svc.cluster.local	27017	-	outbound	ORIGINAL_DST
mongodb-track.default.svc.cluster.local	27017	-	outbound	ORIGINAL_DST
playlist.default.svc.cluster.local	4000	-	outbound	EDS
track.default.svc.cluster.local	5000	-	outbound	EDS
prometheus_stats	-	-	-	STATIC
sds-grpc	-	-	-	STATIC

Παρατηρούμε ότι το πεδίο `SERVICE FQDN` αναφέρεται στο όνομα τομέα των υπηρεσιών προορισμού. Η τιμή `outbound` στο πεδίο `DIRECTION` αναφέρεται στην εξερχόμενη κατεύθυνση της κίνησης η οποία προωθείται εκτός του playlist pod. Τέλος, ο τύπος του συμπλέγματος αναφέρεται στο είδος του τελικού προορισμού. Η τιμή EDS δηλώνει ότι η IP/TCP κίνηση που φθάνει στην υπηρεσία, προωθείται σε τελικό σημείο - pod, το οποίο ανακαλύπτεται δυναμικά από την υπηρεσία EDS (βλ. επόμενη παράγραφο). Εάν η τιμή του τύπου είναι ίση με `ORIGINAL_DST`, όπως στην περίπτωση των mongo βάσεων δεδομένων, τότε η IP/TCP κίνηση κατευθύνεται απευθείας στο pod. Αυτό σημαίνει ότι το FQDN αντιστοιχίζεται με την IP διεύθυνση του pod και όχι της υπηρεσίας. Κάτι τέτοιο συμβαίνει στην περίπτωση που μία υπηρεσία δεν διαθέτει IP διεύθυνση, παραμόνο όνομα τομέα. Τέλος, ο τύπος `STATIC` δηλώνει την ύπαρξη στατικού συμπλέγματος. Συνολικά, με την χρήση της υπηρεσία CDS, διασφαλίζεται η ανανέωση του μητρώου συμπλεγμάτων, έτσι ώστε οι Envoys να διατηρούνται ενήμεροι. Με αυτό τον τρόπο οι διακομιστές είναι σε θέση να εφαρμόσουν αποδοτικά λειτουργίες, όπως εξισορρόπηση φορτίου στα διαθέσιμα pod μιας υπηρεσίας και να δημιουργήσουν listeners προς άλλες υπηρεσίες.

Η υπηρεσία Εντοπισμού Τελικού Σημείου (Endpoint Discovery Service) εστιάζει στη προώθηση της διαμόρφωσης των μεμονωμένων τελικών σημείων κάθε συμπλέγματος. Τα τελικά σημεία αντιπροσωπεύουν τα pods, τα οποία αποτελούν τελικούς προορισμούς όταν αιτήματα αποστέλλονται πρώτα στο σύμπλεγμα στο οποίο ανήκουν. Η υπηρεσία EDS ανανεώνει με δυναμικό τρόπο τα χαρακτηριστικά των τελικών σημείων στο μητρώο των Envoy διακομιστών. Με αυτό τον τρόπο

παρέχονται στους Envoy διακομιστές πληροφορίες σχετικά με τα διαθέσιμα τελικά σημεία, όπως την διεύθυνση IP, την κατάσταση και το σύμπλεγμα στο οποίο ανήκουν. Ακόμη, η υπηρεσία EDS υποστηρίζει λειτουργίες όπως ανίχνευση ακραίων τιμών και ελέγχους υγείας (health checks), με στόχο την διατήρηση των υγειών pods εντός του connection pool. Παρακάτω φαίνονται ορισμένες εγγραφές τελικών σημείων που υπάρχουν στην διαμόρφωση του playlist Envoy proxy. Για την εμφάνιση των εγγραφών εκτελέστηκε η εντολή: `microk8s istioctl proxy-config endpoints playlist-77cb78ccb5-kgxh7.default`

```
ENDPOINT            STATUS   OUTLIER CHECK     CLUSTER
10.1.120.141:4000   HEALTHY  OK                outbound|4000||playlist.default.svc.cluster.local
10.1.120.178:3000   HEALTHY  OK                outbound|3000||frontend.default.svc.cluster.local
10.1.120.184:5000   HEALTHY  OK                outbound|5000||track.default.svc.cluster.local
```

Παρατηρούμε ότι στο πεδίο `ENDPOINT` περιέχεται η IP διεύθυνση και η πόρτα της διεπαφής από την οποία δέχεται αιτήματα ένα τελικό σημείο. Στο πεδίο `STATUS` περιέχεται η τιμή της κατάστασης του endpoint και απ' ότι φαίνεται όλα τα pods είναι υγιή. Ακόμη φαίνεται πως όλα τα pods έχουν περάσει επιτυχώς τον έλεγχο υγείας (`OUTLIER CHECK`). Τέλος στο πεδίο `CLUSTER` περιέχονται τα στοιχεία των συμπλεγμάτων στα οποία ανήκουν τα pods, όπως η κατεύθυνση, η πόρτα και το όνομα τομέα. Επομένως με την χρήση του EDS, το Pilot είναι σε θέση να παρέχει μία αντιπροσωπευτική εικόνα για την κατάσταση των Pods, έτσι ώστε τα αιτήματα των Envoy proxies να προωθούν στα αντίστοιχα υγιή τελικά σημεία.

Συμπερασματικά, οι Υπηρεσίες Ανακάλυψης (Discovery Services) στο Istio Pilot παρουσιάζουν μια ποικιλία αντικειμένων προώθησης διαμορφώσεων για την επίτευξη της επιθυμητής συμπεριφοράς του πλέγματος υπηρεσιών. Μέσω της Υπηρεσία Ανακάλυψης Ακροατών (LDS) προωθούνται διαμορφώσεις ακροατών στους Envoy, περιλαμβάνοντας πρωτόκολλα μεταφοράς, πόρτες, ρυθμίσεις SSL/TLS και αλυσίδες φίλτρων. Η υπηρεσία εντοπισμού δρομολόγησης (RDS) πορωθεί εγγραφές δρομολόγησης αιτημάτων προς διαδρομές υπηρεσιών προορισμού. Μέσω της Υπηρεσίας Ανακάλυψης συμπλέγματος (CDS) συντίθενται διαμορφώσεις συμπλέγματος, καλύπτοντας στατικά και δυναμικά συμπλέγματα. Με την Υπηρεσία Ανακάλυψης Τελικού Σημείου (EDS) αποστέλλονται διαμορφώσεις τελικού σημείου, που σχετίζονται με την κατάσταση υγείας και της δικτυακής διαμόρφωσης των διαθέσιμων pods μιας υπηρεσίας. Μέσα από την χρήση εκείνων των Υπηρεσιών Ανακάλυψης διατηρείται ενήμερη η κατάσταση της διαμόρφωσης των Envoy proxies.

4.2 Citadel

Το Citadel αποτελεί ένα αντικείμενο ελέγχου πιστοποιητικών εντός του Istio πλέγματος. Η κύρια λειτουργία του βασίζεται στην δημιουργία, την επικύρωση και τη διανομή πιστοποιητικών προς τις υπηρεσίες. Αυτά τα πιστοποιητικά λειτουργούν ως μέσο ταυτοποίησης, επιτρέποντας στις υπηρεσίες να επαληθεύσουν τη γνησιότητα των ομότιμων. Έτσι, με αυτό το τρόπο δημιουργούνται μεταξύ των

υπηρεσιών ασφαλή κανάλια επικοινωνίας. Η υπηρεσία Citadel από μόνο της λειτουργεί ως αρχή πιστοποίησης (Certificate Authority), παρέχοντας αυτό-υπογραφόμενα πιστοποιητικά, τα οποία διανέμονται στις υπηρεσίες για την εξασφάλιση αμοιβαίας ασφαλούς επικοινωνίας. Σε άλλη περίπτωση, παρέχεται δυνατότητα ενσωμάτωσης κάποιας εξωτερικής αρχής πιστοποίησης για την επικύρωση των ψηφιακών πιστοποιητικών. Παρακάτω, θα αναλυθεί η ροή εκείνου του μηχανισμού, καθώς και των υπόλοιπων συμπληρωματικών λειτουργιών του Citadel.

Citadel: Χρήση εξωτερικής Αρχής Πιστοποίησης

Σε ένα πλέγμα υπηρεσιών η ενσωμάτωση μίας ήδη υπάρχουσας αρχής πιστοποιητικών (Certificate Authority) αποτελεί ένα από τα κύρια χαρακτηριστικά του Citadel. Με αυτόν τον τρόπο, τα κλειδιά και πιστοποιητικά της υπάρχουσας αρχής (CA) αξιοποιούνται για την δημιουργία ασφαλών συνδέσεων μεταξύ των υπηρεσιών. Για την παραπάνω διαμόρφωση απαιτείται η δημιουργία ενός μυστικού πόρου (Kubernetes Secret) στα πλαίσια του istio-system χώρου ονόματος, στον οποίο προσάπτεται ένας χώρος αποθήκευσης (volume), εντός του οποίου είναι αποθηκευμένα ως αρχείο .pem τα απαραίτητα πιστοποιητικά. Στα απαραίτητα πιστοποιητικά περιλαμβάνονται εκείνα της ρίζας (root cerificate), της αρχής (CA certificate) και η ενδιάμεση αλυσίδα πιστοποιητικών (chain certification), όπως για παράδειγμα το υπογεγραμμένο από το Citadel πιστοποιητικό. Η διάδοση των πιστοποιητικών αναλαμβάνεται πλήρως από το Citadel, πραγματοποιώντας διαμοίραση των πιστοποιητικών. Προσοχή, τα πιστοποιητικά δεν διαμοιράζονται απευθείας στους διακομιστές μεσολάβησης, αλλά σε αφιερωμένους όγκους αποθήκευσης μυστικών. Κάθε διακομιστής μέσω του ξεχωριστού όγκου που του έχει απονεμηθεί, έχει πρόσβαση στο υπογεγραμμένο πιστοποιητικό του, καθώς και δημόσιο κλειδί. Στην έκδοση v1.5 [27], η διαμοίραση των πιστοποιητικών (εξωτερικής αρχής) πραγματοποιείται με την χρήση χειριστών (operators). Οι χειριστές αποτελούν μία επέκταση του Kubernetes API μέσω της οποίας αυτοματοποιούνται διάφορες διαχειριστικές εργασίες στο επίπεδο ελέγχου όπως η προώθηση διαμορφώσεων στο επίπεδο δεδομένων (data plane), και η αυτοματοποιημένη ενημέρωση πόρων κ.α. Επομένως οι χειριστές, μέσω του Kubernetes API, ενημερώνουν τους αντίστοιχους μυστικούς όγκους με τα πιστοποιητικά και δημόσια κλειδιά. Η εγκυρότητα του πιστοποιητικού, μπορεί να ελεγχθεί με την χρήση εργαλείων όπως το OpenSSL, επιθεωρώντας το πιστοποιητικό ενός φόρτου εργασίας.

Citadel: Υπηρεσία λήψης Αιτημάτων Υπογραφής Πιστοποιητικών

Στην προηγούμενη παράγραφο αναλύθηκε ο τρόπος με τον οποίο πραγματοποιείται ενσωμάτωση μίας υπάρχουσας αρχής πιστοποιητικών σε ένα πλέγμα υπηρεσιών. Αφού γνωστοποιηθούν τα απαραίτητα πιστοποιητικά στο επίπεδο ελέγχου, το Citadel είναι σε θέση να διαμοιράσει υπογεγραμμένα πιστοποιητικά στις επιμέρους μονάδες υπηρεσιών (pods). Συγκεκριμένα στους αφιερωμένους όγκους αποθήκευσης μυστικών (secret volumes). Προτού γίνει η ενημέρωση πιστοποιητικού σε έναν όγκο (volume), ο sidecar

διακομιστής αποστέλλει ένα αίτημα υπογραφής πιστοποιητικού (Certificate Signing Request) [28]. Για τον λόγο αυτό, το Citadel, διαθέτει μία υπηρεσία χειρισμού Αιτημάτων Υπογραφής Πιστοποιητικών (CSR service) [29]. Η λειτουργία της CSR υπηρεσίας είναι αρκετά κρίσιμη για την ασφαλή επικοινωνία μεταξύ των υπηρεσιών, καθώς είναι υπεύθυνη για την λήψη αιτημάτων υπογραφής. Το Citadel, επιστρατεύει ένα μηχανισμό ελέγχου υγείας της CSR, με τον οποίο διασφαλίζεται η διαθεσιμότητα της υπηρεσίας. Η λειτουργία του μηχανισμού ελέγχου υγείας (Health Check) βασίζεται στην παρακολούθηση, τον εντοπισμό και την επίλυση βλαβών της CSRS. Εάν εντοπιστεί μια αποτυχία, η διεργασία Kubelet, επανεκκινεί αυτόματα το περιέκτη (container) Citadel. Όταν είναι ενεργοποιημένος ο έλεγχος υγείας, το Citadel χρησιμοποιεί έναν διερευνητή (prober) για την περιοδική αξιολόγηση της κατάστασης υγείας του διακομιστή CSR. Η λειτουργία του διερευνητή στηρίζεται στην αποστολή Αιτημάτων Υπογραφής Πιστοποιητικού (CSR), μέσω του πρωτοκόλλου gRPC στον διακομιστή, επικυρώνοντας στην συνέχεια τις απαντήσεις. Εάν η υπηρεσία CSR λειτουργεί σωστά, ο διερευνητής (prober) ενημερώνει τον χρόνο τροποποίησης ενός αρχείου κατάστασης υγείας. Από την άλλη πλευρά, εάν εντοπιστούν προβλήματα, δεν γίνονται τροποποιήσεις στο αρχείο κατάστασης υγείας.

Η επαλήθευση της λειτουργικότητας του υγειονομικού ελέγχου, μπορεί να γίνει μέσω των αρχείων καταγραφής του Citadel. Εκτελώντας μια συγκεκριμένη εντολή, τα αρχεία καταγραφής αποκαλύπτουν τα αποτελέσματα του ελέγχου υγείας που σχετίζονται με την υπηρεσία υπογραφής CSR. Η υγιής κατάσταση υποδεικνύεται από καταχωρίσεις στο αρχείο καταγραφής όπως "... CSR signing service is healthy (logged every 100 times)." Τα αρχεία καταγραφής διαβεβαιώνουν τους διαχειριστές, ότι ο μηχανισμός περιοδικού ελέγχου υγείας λειτουργεί όπως αναμένεται. Από προεπιλογή, ο υγειονομικός έλεγχος πραγματοποιείται κάθε 15 δευτερόλεπτα, με τις εγγραφές ημερολογίου να δημιουργούνται μία φορά κάθε 100 ελέγχους. Για την τροποποίηση των παραμέτρων υγείας, το Citadel παρέχει το αρχείο διαμόρφωσης "istio-citadel-with-health-check.yaml" [30] για τον ορισμό της επιθυμητής λειτουργίας του μηχανισμού. Παράμετροι όπως liveness-probe-path, liveness-probe-interval, probe-check-interval, initialDelaySeconds και periodSeconds μπορούν να προσαρμοστούν βάσει συγκεκριμένων απαιτήσεων. Ωστόσο, αυτές οι διαμορφώσεις θα πρέπει να προσαρμόζονται με προσοχή, λαμβάνοντας υπόψη τον επιπλέον φόρτο που προξενούν στο δίκτυο του πλέγματος οι συχνοί υγειονομικοί έλεγχοι. Συμπερασματικά, η λειτουργία ελέγχου υγείας του Citadel ενισχύει σημαντικά την αξιοπιστία της υπηρεσίας Αίτησης Υπογραφής Πιστοποιητικού (CSR) εντός του πλέγματος υπηρεσιών. Επιτρέποντας την αυτοματοποιημένη ανίχνευση και επίλυση αστοχιών για την διατήρηση ασφαλών καναλιών επικοινωνίας.

Citadel: Υπηρεσία Ανακάλυψης Μυστικών

Στην παράγραφο «Χρήση εξωτερικής Αρχής Πιστοποίησης», διερευνήθηκε ο τρόπος με τον οποίο γίνεται ενσωμάτωση εξωτερικών αρχών πιστοποίησης (CAs) στο Citadel. Τα πιστοποιητικά των

διακομιστών αποθηκεύονται σε αφιερωμένους μυστικούς όγκους αποθήκευσης (secret volumes) για κάθε sidecar διακομιστή ξεχωριστά. Η λύση αυτή όμως παρουσιάζει ορισμένες ευαισθησίες απόδοσης και ασφάλειας. Κάθε φορά που χρειάζεται να γίνει ανανέωση του πιστοποιητικού (π.χ. λόγω λήξης) σε έναν διακομιστή μεσολάβησης, απαιτείται επανεκκίνηση για την ολοκλήρωση της ανανέωσης. Ακόμη, η αποθήκευση των επιμέρους πιστοποιητικών σε αφιερωμένους μυστικούς όγκους των sidecar διακομιστών μπορεί να αποτελέσει ευπάθεια, καθώς τα υπογεγραμμένα πιστοποιητικά και τα δημόσια κλειδιά διασκορπίζονται σε διάφορους χώρους αποθήκευσης. Την λύση στο πρόβλημα, φέρνει η Υπηρεσία Ανακάλυψης Μυστικών (Secret Discovery Service). Με την χρήση ενός SDS εξυπηρετητή ανά κόμβο (node), το Citadel είναι σε θέση να ανανεώνει δυναμικά τα πιστοποιητικά σε κάθε διακομιστή μεσολάβησης ξεχωριστά. Με τον SDS μηχανισμό, το πιστοποιητικό αποστέλλεται στους sidecar διακομιστές, μέσω ενός Unix domain socket. Ωστόσο, η εγκαθίδρυση του καναλιού απαιτεί ενεργοποίηση του πρωτοκόλλου αμοιβαίας ασφάλειας επιπέδου μεταφοράς (mTLS) στο πλέγμα. Για την αποφυγή ανεπιθύμητης πρόσβασης στο socket, μπορούν να εφαρμοστούν πολιτικές RBAC (Role Based Access Control), με την χρήση Envoy φίλτρων. Με εκείνες τις πολιτικές, περιορίζεται η πρόσβαση στο socket, μόνο στο πλαίσιο επικοινωνίας μεταξύ του Envoy και της υπηρεσίας SDS. Η υπηρεσία SDS προστέθηκε στην λειτουργικότητα του επιπέδου ελέγχου στην έκδοση Istio 1.1.0 [31], καθώς στις προηγούμενες εκδόσεις η υπογραφή και διαμοίραση των πιστοποιητικών γινόταν αποκλειστικά από το Citadel με την χρήση όγκων. Ενώ, με την χρήση της SDS ροής, δεν υπάρχει ανάγκη για αφιερωμένους όγκους αποθήκευσης μυστικών, αλλά ούτε ανάγκη επανεκκίνησης των διακομιστών μεσολάβησης, στην περίπτωση ανανέωσης ενός πιστοποιητικού.

4.3 Galley

Σε προηγούμενο κεφάλαιο αναλύθηκε η λειτουργία του Pilot ως ένα βασικό μέρος του επιπέδου ελέγχου στην αρχιτεκτονική του Istio. Έγινε αναφορά για την ευθύνη του Pilot ως προς την διαχείριση και δυναμική προώθηση των διαμορφώσεων του επιπέδου δεδομένων στους αντίστοιχους διακομιστές μεσολάβησης. Όμως προτού γίνει η προώθηση των διαμορφώσεων μέσω των υπηρεσιών ανακάλυψης xDS, προηγείται ο έλεγχος ορθότητας των διαμορφώσεων. Το Galley είναι εκείνο το μέρος του Istio με το οποίο εκτελείται η επικύρωση των διαφόρων διαμορφώσεων που αφορούν το επίπεδο δεδομένων, καθώς και το επίπεδο ελέγχου.

Galley: Έλεγχος ορθότητας και κανονικοποίηση διαμορφώσεων

Σε ένα περιβάλλον πλέγματος όπως το Istio, η λειτουργικότητα μεταξύ των επιμέρους στοιχείων βασίζεται στην αποκεντρωμένη διαχείριση. Είναι τέτοιες οι συνθήκες εντός του πλέγματος, που η εφαρμογή ενός κατανεμημένου μοντέλου για την συνεργασία και λειτουργία των στοιχείων του πλέγματος θα οδηγούσε πολύ γρήγορα σε ανεπάρκειες. Το πλέγμα πρέπει να ανταποκρίνεται πολύ γρήγορα σε

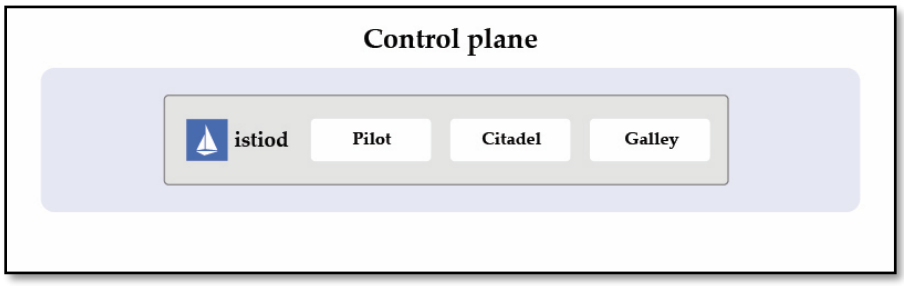
αλλαγές διαμορφώσεων, τοπολογίας δικτύου και πόρων. Η ύπαρξη του επιπέδου ελέγχου, παρέχει αυτή την δυνατότητα, μέσω των μερών που απαρτίζεται. Μία από τις κύριες λειτουργίες του επιπέδου ελέγχου, εμπίπτει στον έλεγχο ορθότητας των διαμορφώσεων που περιγράφουν τις επιθυμητές αλλαγές και συμπεριφορά του πλέγματος. Το Galley [32], λοιπόν επικυρώνει τη δομή, το σχήμα και τις σχέσεις μεταξύ διαφορετικών στοιχείων διαμόρφωσης, αποτρέποντας την εφαρμογή εσφαλμένων ή μη συμβατών διαμορφώσεων στο πλέγμα υπηρεσίας. Ένα βασικό χαρακτηριστικό του πλέγματος είναι η ετερογένεια μεταξύ των στοιχείων τα οποία μπορεί το καθένα να υιοθετεί διαφορετικό μοντέλο διαμόρφωσης. Το Galley κανονικοποιεί αυτές τις διαμορφώσεις σε μια συνεπή και ενοποιημένη μορφή που είναι συμβατή με το επίπεδο ελέγχου και τους διακομιστή μεσολάβησης (sidecar proxies) του επιπέδου δεδομένων. Ακόμη, χρησιμοποιεί βέλτιστες πρακτικές τυποποίησης των συμβάσεων ονομασίας επιλύοντας διενέξεις ή ασυνέπειες μεταξύ διαφορετικών πηγών διαμόρφωσης. Για παράδειγμα, μπορεί να μετατρέψει τις διαμορφώσεις μορφής YAML ή JSON σε μορφή Protocol Buffer (Protobuf). Αυτή είναι μια συμπαγής και αποτελεσματική μορφή δυαδικής σειριοποίησης δεδομένων που χρησιμοποιείται για εσωτερική επικοινωνία. Το Protobuf παρέχει μια δομημένη αναπαράσταση δεδομένων χρησιμοποιώντας έναν ορισμό σχήματος (schema), επιτρέποντας την αποτελεσματική κωδικοποίηση, αποκωδικοποίηση και χειρισμό των μηνυμάτων διαμόρφωσης. Με την κανονικοποίηση των διαμορφώσεων σε μια συνεπή μορφή, το Galley διασφαλίζει ότι το επίπεδο ελέγχου και δεδομένων, είναι σε θέση να επεξεργάζονται και να ερμηνεύουν αποτελεσματικά τις διαμορφώσεις, επιτρέποντας απρόσκοπτη επικοινωνία και συνεπή συμπεριφορά εντός του πλέγματος υπηρεσιών.

Galley: Απόδοση και Επεκτασιμότητα

Το Galley έχει σχεδιαστεί για να χειρίζεται μεγάλα δίκτυα πλέγματος με χιλιάδες διακομιστές μεσολάβησης. Το Galley μπορεί να χειριστεί αυξημένο όγκο φορτίων, δίχως απώλεια απόδοσης, αποτελώντας μία πλήρως επεκτάσιμη υπηρεσία. Η συνεργασία του Galley με τον αποτελεσματικό μηχανισμό παράδοσης διαμορφώσεων του Pilot, επιτρέπει την αξιόπιστη και γρήγορη διάδοση των αλλαγών διαμόρφωσης σε όλο το δίκτυο. Συνεπώς, όλοι οι διακομιστές μεσολάβησης λαμβάνουν εγκαίρως, έγκυρες ενημερώσεις. Ακόμη, το Galley παρέχει μια μεγάλη ποικιλία επιλογών προσαρμογής και επέκτασης, δίνοντας τη δυνατότητα στους χρήστες να προσαρμόσουν τη διαχείριση διαμόρφωσης του Istio στις συγκεκριμένες ανάγκες τους. Υποστηρίζει την ενοποίηση εξωτερικών συστημάτων μέσω προσαρμογέων (adapters), επιτρέποντας την επικοινωνία με τις υπάρχοντες διεργασίες ελέγχου. Αυτή η επέκταση δίνει τη δυνατότητα στους οργανισμούς να αξιοποιήσουν τις υπάρχουσες λύσεις ασφάλειας, ανάλυσης και συμμόρφωσης εντός του δικτύου Istio. Με την ενσωμάτωση εξωτερικών συστημάτων, το Galley παρέχει μια ενοποιημένη και ευέλικτη προσέγγιση στη διαχείριση των διαμορφώσεων του πλέγματος, διασφαλίζοντας συμμόρφωση (compliance) με τις υπάρχουσες οργανωτικές δομές.

4.4 Istiod

Όλα τα παραπάνω μέρη του επιπέδου ελέγχου, αποτελούσαν ξεχωριστές υπηρεσίες. Στα αρχικά στάδια της ανάπτυξης του Istio, το επιπέδου ελέγχου υλοποιήθηκε από την ομάδα ανάπτυξης με την αρχιτεκτονική των μικροϋπηρεσιών. Σε εκείνες τις εκδόσεις του control plane, υπήρχε και μία επιπλέον υπηρεσία, το Mixer. Η υπηρεσία εκείνη ήταν υπεύθυνη για την συλλογή δεδομένων τηλεμετρίας από το επίπεδο δεδομένων. Ακόμη μέσω εκείνης της υπηρεσίας πραγματοποιούταν επιβολή πολιτικών χρήσης και ελέγχου πρόσβασης [33]. Αργότερα η ομάδα ανάπτυξης του Istio, συμπέρανε πως υλοποίηση του επιπέδου ελέγχου σε ξεχωριστές υπηρεσίες δημιουργούσε υπερβολικό φόρτο στο πλέγμα. Σύμφωνα με μία έρευνα που πραγματοποιήθηκε το 2021 [34], οι κριτικές των χρηστών ήταν εκείνες οι οποίες οδήγησαν την ομάδα ανάπτυξης του Istio σε αναθεώρηση της αρχιτεκτονικής του επιπέδου ελέγχου. Συνεπώς το επόμενο βήμα ήταν η ανάπτυξη του control plane ως μία μονολιθική υπηρεσία. Συγκεκριμένα η υπηρεσία Mixer αφαιρέθηκε τελείως και οι επιμέρους συνιστώσες Pilot, Citadel, Galley ενοποιήθηκαν σε μία ενιαία υπηρεσία, την Istiod, όπως φαίνεται παρακάτω.



Εικόνα 2: Επίπεδο ελέγχου στο Istio

Επίσης με την χρήση της νέας έκδοσης του επιπέδου ελέγχου, τα δεδομένα τηλεμετρίας συλλέγονται κατευθείαν από την υπηρεσία Prometheus [35]. Αντίθετα στην microservice αρχιτεκτονική του επιπέδου ελέγχου, τα δεδομένα τηλεμετρίας συλλέγονταν από την υπηρεσία Mixer και ύστερα αποστέλλονταν στο Prometheus API. Και στις δύο αρχιτεκτονικές η πηγή δεδομένων τηλεμετρίας είναι οι Envoy διακομιστές μεσολάβισης.

ΚΕΦΑΛΑΙΟ 5: ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΟΥ ISTIO

Σε αυτό το κεφάλαιο της εργασίας διερευνώνται τα χαρακτηριστικά και οι λειτουργίες που προφέρονται από την πλατφόρμα Istio, δίνοντας έμφαση στην διαχείριση της κίνησης, την παρατηρησιμότητα και την ασφάλεια στις υπηρεσίες πλέγματος. Ακόμη, ορίζεται ο τρόπος εφαρμογής και ενεργοποίησης των συγκεκριμένων λειτουργιών, καθώς και τα οφέλη που μπορούν να προκύψουν από την χρήση τους.

5.1 Διαχείριση Κίνησης

Η παρούσα θεματική πραγματεύεται θεμελιώδεις μηχανισμούς και τεχνικές του Istio που συμβάλλουν σημαντικά στην αξιοπιστία και διαχείριση της κίνησης σε ένα πλέγμα, πράγμα που μπορεί να αποτελέσει πρόκληση κατά την ανάπτυξη εφαρμογών. Θα συζητηθούν συνολικά τέσσερις βασικές πτυχές που επιτρέπουν την μηχανίκευση της κίνησης στο επίπεδο εφαρμογής. Αυτές οι πτυχές περιλαμβάνουν το φιλτράρισμα της κίνησης βάση της HTTP επικεφαλίδας, τη εξισορρόπηση φορτίου, την ενημέρωση εφαρμογών με σιγουριά και την προσομοίωση σφαλμάτων ή καθυστερήσεων.

Δρομολόγηση κίνησης βάση HTTP επικεφαλίδας

Αυτός ο τύπος δρομολόγησης βασίζεται στον έλεγχο της HTTP επικεφαλίδας ενός αιτήματος. Με το μοντέλο διαμόρφωσης Virtual Service, ορίζονται τα κριτήρια ταιριάσματος - πεδία μίας προσαρμοσμένης (custom) HTTP επικεφαλίδας, στα οποία βασίζεται η προώθηση των HTTP αιτημάτων. Μία προσαρμοσμένη επικεφαλίδα μπορεί να περιέχει πληροφορίες σχετικά με την ταυτότητα του χρήστη, όπως το όνομά του (username). Ένα τέτοιο σενάριο αποτελεί η αποστολή ενός προσαρμοσμένου http request ενός API σε ένα άλλο, για μία συγκεκριμένη ομάδα χρηστών. Μετά το ταίριασμα της προσαρμοσμένης κεφαλίδας ενός http request, το αίτημα προωθείται προς μία συγκεκριμένη έκδοση της εφαρμογής, δηλαδή προς ένα υποσύνολο από pods (βλ. Παράρτημα: VirtualService, DestinationRule) Για παράδειγμα μετά την εφαρμογή της παρακάτω διαμόρφωσης VirtualService[36], ένα αίτημα προς την υπηρεσία (Service) reviews φτάνει στον αντίστοιχο Ingress gateway. Ο gateway, ελέγχει την προσαρμοσμένη επικεφαλίδα `end-user` εάν έχει την τιμή `jason`. Εάν γίνει ταίριασμα, τότε το αίτημα προωθείται προς το υποσύνολο `v1` της υπηρεσίας reviews, διαφορετικά προωθείται προς το υποσύνολο `v2`.

```
- match:
  - headers:
      end-user:
        exact: jason
  route:
  - destination:
      host: reviews
      subset: v2
- route:
  - destination:
      host: reviews
      subset: v1
```

Αντίστοιχο ταίριασμα τιμών μπορεί να γίνει και στα cookies μιας κεφαλίδας ενός http request, αντικαθιστώντας το πεδίο `end-user` με `cookie`, καθώς και το πεδίο `exact` με τα αντίστοιχα πεδία key και τιμές (`exact.value`) του cookie. Με αυτό τον τρόπο είναι δυνατός ο έλεγχος του session ID για περαιτέρω εξουσιοδότηση (authorization) του χρήστη. Ακόμη, εκτός από την χρήση των cookies, πλέον προτιμάται και η χρήση των web tokens. Η χρήση των web tokens (βλ. Παράρτημα: JWT) αποτελεί σήμερα μία από τις κύριες χρησιμοποιούμενες μεθόδους για την εξουσιοδότηση των χρηστών, καθώς παρέχει ευελιξία και αποδοτικότητα στην πλευρά των διακομιστών. Στο Istio, λοιπόν πρόσφατα εντάχθηκε και η δυνατότητα JWT (JSON Web Token) ελέγχου. Η δυνατότητα αυτή βρίσκεται ακόμη στην κατηγορία Alpha (όχι και τόσο σταθερή, δίχως υποστήριξη), παρ' όλ' αυτά προσφέρεται μόνο για προηγμένη χρήση και δοκιμή από προγραμματιστές. Περεταίρω ανάλυση και διερεύνηση του τρόπου λειτουργίας του μηχανισμού αυθεντικοποίησης και εξουσιοδότηση χρήστη με την χρήση JWT, πραγματοποιείται στο Κεφάλαιο 5: «Χαρακτηριστικά του ISTIO: Ασφάλεια».

Load Balancing

Μετά την ομαδοποίηση των pods σε υπηρεσίες και υποσύνολα, το Istio παρέχει μέσα από την DestinationRule διαμόρφωση την επιβολή στοχευμένων πολιτικών κατανομής φόρτου. Οι πολιτικές κατανομής φόρτου μεταξύ των Pods, υλοποιούνται με την χρήση των αλγορίθμων Least Required, Round Robin και Random. Ο καθορισμός της πολιτικής γίνεται εντός της διαμόρφωσης DestinationRule, με την δήλωση του YAML attribute `loadBalancer` (`trafficPolicy.loadBalancer.simple`). Οι τιμές που μπορεί να πάρει το attribute είναι `UNSPECIFIED`, `RANDOM`, `PASSTHROUGH`, `ROUND_ROBIN` και `LEAST_REQUEST`. Με την επιλογή της τιμής `UNSPECIFIED` δεν χρησιμοποιείται κάποιος συγκεκριμένος αλγόριθμος, με αποτέλεσμα να εφαρμοστεί μία προεπιλεγμένη κατανομή του φόρτου ορισμένη από το Istio. Η χρήση της τιμής `PASSTHROUGH` γίνεται μόνο σε περίπτωση που η εξισορρόπηση φορτίου, προτιμάται να εκτελεστεί από την Υπηρεσία (L4 Load Balancing) και όχι από τον Envoy proxy (L7 Load Balancing). Πολιτικές κατανομής φόρτου μπορούν να εφαρμοστούν σε ολόκληρες υπηρεσίες και subsets, όπως αναφέρθηκε. Εκτός αυτού, διαφορετικές πολιτικές εξισορρόπησης φορτίου μπορούν να οριστούν ξεχωριστά σε κάθε πόρτα που ακούει μία Υπηρεσία. Για παράδειγμα, μία υπηρεσία δέχεται HTTP αιτήματα στην πόρτα 80 και HTTP αιτήματα στην πόρτα 443. Σε εκείνη την υπηρεσία μπορεί να εφαρμοστεί πολιτική `LEAST_REQUEST` στην πόρτα 443 και `ROUND_ROBIN` στην πόρτα 80. Η ρύθμιση αυτή μπορεί να εφαρμοστεί με την χρήση του YAML attribute `portLevelSettings`, όπου καθορίζεται ο αριθμός της πόρτας (`portLevelSettings.port.number.`) και έπειτα η πολιτική της κίνησης (`traffivPolicy`), όπως στο παρακάτω ενδεικτικό παράδειγμα [37].

```
    loadBalancer:
      simple: ROUND_ROBIN
    portLevelSettings:
    - port:
        number: 443
      tls:
        credentialName: client-credential
        mode: MUTUAL
```

Traffic Shifting

Η έννοια της αλλαγής κίνησης (traffic shifting) στο Istio αναφέρεται στην δυνατότητα μετατόπισης της HTTP κίνησης, ανάμεσα σε διαφορετικές εκδόσεις μιας υπηρεσίας, δηλαδή σε διαφορετικά subsets. Η μετατόπιση της κίνησης πραγματοποιείται μέσα από τον ορισμό ποσοστών (βαρών) σε κάθε έκδοση. Αυτό επιτρέπει την δοκιμή νέων εκδόσεων της υπηρεσίας, ενώ παράλληλα διατηρείται ένα μέρος της κίνησης στην παλιά έκδοση. Στο παρακάτω παράδειγμα παρουσιάζεται μία VirtualService διαμόρφωση καθορισμού δύο διαφορετικών προορισμών δρομολόγησης των αιτημάτων της υπηρεσίας `frontend`.

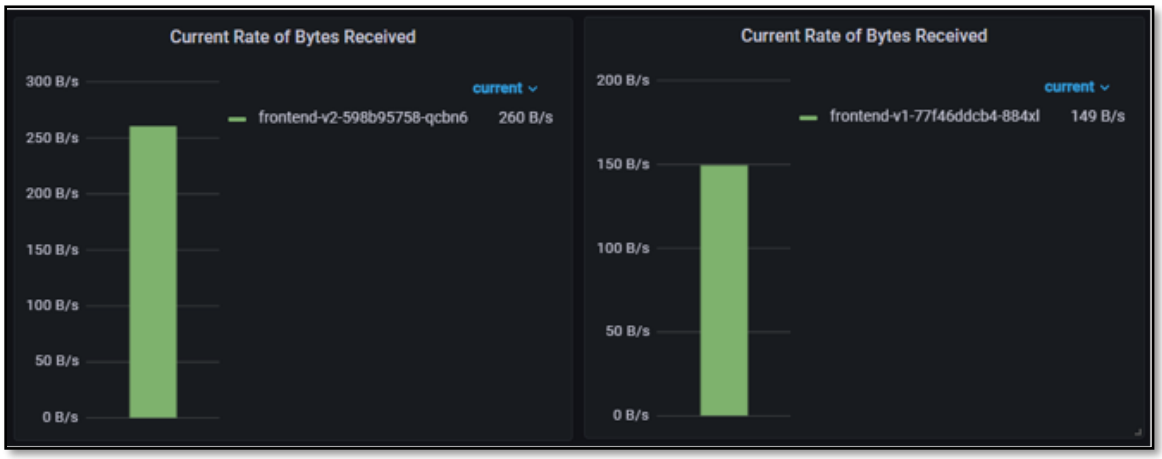
```
http:
  - route:
    - destination:
        host: frontend
        subset: v1
      weight: 30
    - destination:
        host: frontend
        subset: v2
      weight: 70
```

Υπάρχουν δύο δυνατοί προορισμοί (`destinations`) προς εκείνη την υπηρεσία, ο `v1` και ο `v2`. Τα βάρη έχουν οριστεί ως 30 για το subset `v1`, και 70 για το `v2`. Επομένως, το 30% της κίνησης προς την υπηρεσία `frontend` κατευθύνεται προς την έκδοση v1 και το 70% προς την έκδοση `v2`. Για την δημιουργία των δύο subsets υλοποιήθηκαν οι διαμορφώσεις frontend-v1-v2-Deployment.yaml και frontend-v1-v2-DestinationRule.yaml, ενώ για τον καθορισμό των βαρών δημιουργήθηκε η εικονική υπηρεσία frontend-v1-v2-VirualService.yaml. Για την επιβεβαίωση της τήρησης των βαρών δημιουργήθηκε το παρακάτω bash script, με το οποίο πραγματοποιείται ανά ένα δευτερόλεπτο HTTP αίτημα προς την υπηρεσία frontend, κάτω από TLS συνδέσεις.

```
#!/bin/bash

while true; do
    curl -s -o /dev/null -w "%{http_code}\n" https://homelabamplifier.com
    sleep 1
done
```

Το script εκτελέστηκε για 15 λεπτά συνολικά και έπειτα τερματίστηκε. Παρατηρώντας τα γραφήματα του ρυθμού άφιξης των Bytes στο UI της υπηρεσίας Grafana, φαίνονται τα παρακάτω αποτελέσματα για τις δύο εκδόσεις της υπηρεσίας frontend.



Εικόνα 3: Ρυθμός άφιξης των Bytes στις εκδόσεις v1, v2 της υπηρεσίας frontend

Ο μέσος ρυθμός άφιξης των Bytes για την έκδοσης v2 είναι ίσος με 260 B/s, ενώ για την έκδοση v1 είναι ίσος με 149 B/s στο χρονικό παράθυρο των 15 λεπτών. Σύμφωνα με τα βάρη, εάν ο ρυθμός 260B/s είναι το 0.7 του συνολικού ρυθμού, τότε ο ρυθμός 149 B/s είναι ίσος με $\frac{149 * 0.7}{260} = 0.4$ του συνολικού ρυθμού. Επομένως το ποσοστά μετατόπισης της κίνησης στις εκδόσεις v1 και v2 είναι περίπου ίδια με εκείνα που δηλώθηκαν στην Εικονική Υπηρεσία της υπηρεσίας frontend. Η απόκλιση ήταν της τάξης $0.4 - 0.3 = 0.1 = 10\ \%$.

Εν κατακλείδι με την μετατόπιση της κίνησης δίνεται η δυνατότητα ελέγχου της επίδοσης και συμπεριφοράς της νέας έκδοσης `v2` από την ομάδα ανάπτυξης, προτού γίνει εξολοκλήρου κατεύθυνση της κίνησης στην νέα έκδοση. Αυτή η μέθοδος ονομάζεται και αλλιώς “Canary Deployments”.

Rate Limiting

Το Istio δεν διαθέτει κάποιον μηχανισμό περιορισμού ρυθμού, αντιθέτως χρησιμοποιεί μία εξωτερική υπηρεσία, στην οποία αποστέλλεται κάθε HTTP αίτημα που φτάνει στον Envoy και ύστερα αποφασίζεται αν θα απορριφθεί. Εάν τηρείται το όριο ρυθμού άφιξης αιτημάτων, τότε η υπηρεσία αποστέλλει HTTP response 200 προς τον Envoy. Εάν όχι, τότε αποστέλλεται μόνο ο κωδικός 429 ως απάντηση στον Envoy. Η διάσπαση του μηχανισμού rate limit από την λειτουργικότητα του Envoy, αποφέρει ευελιξία καθώς δίνεται η δυνατότητα προσαρμογής ή ακόμη και δημιουργίας μίας εξωτερικής rate limiting υπηρεσίας, η οποία θα ταιριάζει σε συγκεκριμένες απαιτήσεις. Επίσης, δύο pods εκτελούν την RateLimit υπηρεσία, η οποία υλοποιήθηκε από την Envoy (Google, IBM, Lyft). Ο envoy διακομιστής σε κάθε Pod, εάν χρειαστεί να επικοινωνήσει με μία άλλη υπηρεσία, δημιουργεί ένα Envoy Cluster μέσω της CDS υπηρεσίας. Το

Envoy Cluster αντιστοιχεί σε εκείνη την υπηρεσία και ύστερα μέσω ενός Listener θα είναι σε θέση ο Envoy να αποστείλει αιτήματα προς εκείνη. Όμως, επειδή η RateLimit υπηρεσία είναι εξωτερική, θα πρέπει να δηλωθεί χειροκίνητα (manually) το αντίστοιχο Envoy Cluster. Ο φραγμός των αιτημάτων γίνεται μέσα από την καταμέτρηση του πλήθους των αιτημάτων και την σύγκριση αυτού με quotas εντός συγκεκριμένου χρονικού διαστήματος. Για τον λόγο αυτόν, τα pods της RateLimit υπηρεσίας ενσωματώνονται στο σύμπλεγμα μαζί με μία Redis βάση δεδομένων, στην οποία ανανεώνονται δυναμικά τα πλήθη των αιτημάτων για συγκεκριμένα Endpoints και υπηρεσίες. Τα pods της RateLimit υπηρεσίας αξιοποιούν τις εγγραφές της Redis βάσης δεδομένων για την τήρηση του ρυθμού άφιξης των αιτημάτων. Παραπάνω τονίστηκε η χρήση εξωτερικής υπηρεσίας για την οριοθέτηση της κίνησης. Η συγκεκριμένη λειτουργία απαιτεί την επέκταση της διαμόρφωσης του Envoy. Η επέκταση αυτή πραγματοποιείται με την χρήση ορισμένων Envoy φίλτρων (Envoy Filters), έτσι ώστε ένας Envoy proxy να μπορέσει να λειτουργήσει σε συνεργασία με την εξωτερική RateLimit υπηρεσία.

Η δημιουργία της RateLimit υπηρεσίας γίνεται μέσα από την διαμόρφωση rate-limit-service.yaml του Istio repository ή την τροποποίηση αυτής. Η δήλωση της rate limit πολιτικής πραγματοποιείται μέσω της χρήσης ενός Kubernetes ConfigMap πόρου. Σε εκείνο τον πόρο δηλώνεται το όριο του πλήθους των αιτημάτων για ένα μονοπάτι, όπως φαίνεται παρακάτω.

```
  config.yaml: |
    domain: productpage-ratelimit
    descriptors:
      - key: PATH
        value: "/productpage"
        rate_limit:
          unit: minute
          requests_per_unit: 1
```

Άρα η παραπάνω διαμόρφωση ορίζεται για το μονοπάτι `PATH` (uri) `/productpages` στον Ingress Gateway, οριοθετώντας την HTTP κίνηση προς την υπηρεσία.

Έπειτα, σειρά έχει η διαμόρφωση EnvoyFilter, με την οποία ορίζεται η προσθήκη φίλτρων στους listeners του Ingress Gateway, στοχεύοντας στην οριοθέτηση του ρυθμού άφιξης των HTTP πακέτων στις υπηρεσίες προορισμού.

```
spec:
  workloadSelector:
    labels:
      istio: ingressgateway
```

Με την χρήση του πεδίου `routeConfiguration` του `configPatches` (διαμόρφωση EnvoyFilter) καθορίζονται οι υπηρεσίες για τις οποίες θα εφαρμοστεί το HTTP φίλτρο. Αν δεν οριστεί κάποιο

routeConfiguration, τότε εκτελείται καθολικό rate limiting για όλες της υπηρεσίες, όπως φαίνεται παρακάτω.

```
  configPatches:
      match:
        context: GATEWAY
        listener:
          filterChain:
            filter:
              name: "envoy.filters.network.http_connection_manager"
              subFilter:
                name: "envoy.filters.http.router"
```

Αξίζει να σημειωθεί ότι σε αυτή την περίπτωση, θα πρέπει να έχει δημιουργηθεί στον ConfigMap πόρο ο αντίστοιχος rate limit κανόνας δίχως value στο πεδίο key.PATH του descriptor, για την εφαρμογή καθολικού rate limiting. Με την εφαρμογή της παραπάνω διαμόρφωσης φίλτρου, δημιουργείται μία αλυσίδα φίλτρων filterChain η οποία περιέχει το κύριο φίλτρο (filter) με όνομα envoy.filters.network.http_connection_manager και το υποφίλτρο (subfilter) με όνομα envoy.filters.http.router. Το κύριο φίλτρο είναι υπεύθυνο για την διαχείριση και επεξεργασία της HTTP κίνησης. Το subfilter αποτελεί ένα HTTP φίλτρο με το οποίο δρομολογούνται τα HTTP requests και responses βάση των εγγραφών δρομολόγησης. Τα παραπάνω φίλτρα εφαρμόζονται ούτως ή άλλως από προεπιλογή, παρ' όλ' αυτά η δήλωσή τους είναι προληπτική για την τήρηση της αλληλουχίας στην αλυσίδα φίλτρων. Τέλος, στην αλυσίδα φίλτρων προστίθεται ένα ακόμη φίλτρο με όνομα envoy.filters.http.ratelimit, το οποίο καλείται όταν HTTP πακέτα περνούν στο socket επικοινωνίας με την υπηρεσία προορισμού. Τότε με την εκτέλεση του φίλτρου καλούνται διάφορες callback συναρτήσεις (gRPC) στην RateLimit υπηρεσία και ύστερα η υπηρεσία γνωστοποιεί τον Envoy για την ενέργεια που θα επιβληθεί για το αντίστοιχο HTTP request.

```
  patch:
    operation: INSERT_BEFORE
    value:
      name: envoy.filters.http.ratelimit
        rate_limit_service:
          grpc_service:
            envoy_grpc:
              cluster_name: outbound|8081||ratelimit.default.svc.cluster.local
              authority: ratelimit.default.svc.cluster.local
```

Το φίλτρο οριοθέτησης ρυθμού εφαρμόζεται στην πόρτα 8081 του cluster service ratelimit, που ανήκει στην εξωτερική υπηρεσία. Υπενθυμίζεται ό,τι η υπηρεσία ratelimit δημιουργήθηκε με σκοπό οι Envoy proxies να έχουν την δυνατότητα επικοινωνίας με την Ratelimit επικοινωνία μέσω του connection pool.

Circuit Breaking

Σε ένα πλέγμα υπηρεσιών, η ανθεκτικότητα σε σφάλματα είναι ένα από τα βασικά χαρακτηριστικά με το οποίο εξασφαλίζεται η ομαλή λειτουργία των μικροϋπηρεσιών. Η εμφάνιση αστοχίας σε μία υπηρεσία μπορεί να δημιουργήσει μία αλυσίδα διαδοχικών σφαλμάτων και σε άλλες υπηρεσίες, οι οποίες είναι στενά συνδεδεμένες. Ακόμη, σε συνθήκες υψηλής ζήτησης η ανταπόκριση μίας υπηρεσίας μπορεί να περιοριστεί, με αποτέλεσμα ενδεχομένως να επηρεαστούν και άλλες υπηρεσίες. Σε περιβάλλοντα μικροϋπηρεσιών το φαινόμενο αυτό δημιουργεί επιπλέον δικτυακή κίνηση σε ένα σύμπλεγμα, καθώς η αποτυχία μίας υπηρεσίας οδηγεί σε επαναλαμβανόμενες προσπάθειες επανασύνδεσης από άλλες υπηρεσίες. Επίσης είναι γεγονός ό,τι λάθη μπορούν να συμβούν στα πλαίσια διαμόρφωσης των υπηρεσιών, αλλά ακόμη και επιθέσεις DDoS, προκαλώντας αστάθειες και υπερφόρτωση του δικτύου. Η πρόληψη τέτοιων ανεπιθύμητων φαινομένων είναι σημαντική καθώς το αντίκτυπο τους μπορεί να επιφέρει μεγάλο κόστος, ειδικά σε περιβάλλοντα υπολογιστικής νέφους. Ένας από τους σημαντικότερους μηχανισμούς του Istio είναι η διακοπή κυκλώματος (circuit breaking), με το οποίο παρέχεται λύση στο πρόβλημα. Η ύπαρξη του μηχανισμού στην πλατφόρμα Istio προσφέρει ελαστικότητα καθώς η ρύθμισή του μπορεί να γίνει εύκολα για πλήθος προορισμών, σε αντίθεση με την μεμονωμένη υλοποίηση παρόμοιου μηχανισμού στον κώδικα της εφαρμογής κάθε υπηρεσίας.

Στα πλαίσια υλοποίησης του μηχανισμού διακοπής κυκλώματος, όταν εμφανίζονται σφάλματα ή υψηλή καθυστέρηση σε μία υπηρεσία, ενεργοποιείται ο διακόπτης κυκλώματος, με αποτέλεσμα να τερματίζεται η επικοινωνία μεταξύ οποιασδήποτε υπηρεσίας προς αυτή για ένα συγκεκριμένο χρονικό διάστημα. Η εφαρμογή αυτής της δυνατότητας πραγματοποιείται μέσω πολιτικών κίνησης (trafficPolicy) εντός της διαμόρφωσης DestinationRule. Με αυτό τον τρόπο καθορίζεται το μέγιστο πλήθος των http αιτημάτων, το μέγιστο πλήθος tcp συνδέσεων και το μέγιστο πλήθος αναμενόντων http αιτημάτων σε έναν προορισμό. Από προεπιλογή, η τιμή των παραπάνω attributes είναι ίση με $2^{32} - 1$. Για παράδειγμα, παρακάτω φαίνεται η ρύθμιση των μέγιστων TCP συνδέσεων στην διαμόρφωση DestinationRule μίας υπηρεσίας.

```
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 6
```

Το μέγιστο πλήθος παράλληλων TCP συνδέσεων για την υπηρεσία θα είναι τρείς. Οι υπόλοιπες αιτούμενες συνδέσεις υπεισέρχονται σε ουρά αναμονής.

Ο περιορισμός των αναμενόντων TCP συνδέσεων πραγματοποιείται μέσω του περιορισμού των αναμενόντων HTTP αιτημάτων. Ο περιορισμός διακρίνεται για αιτήματα τύπου HTTP1 και HTTP2. Η διαφορά τους έγκειται στο γεγονός ότι στο πρωτόκολλο HTTP, η αποστολή ενός τμήματος, πρέπει να έχει

ολοκληρωθεί εντός μίας TCP σύνδεσης, πριν σταλεί το επόμενο. Επομένως για παράλληλα αιτήματα δημιουργούνται διαφορετικές TCP συνδέσεις. Ενώ στην επιλογή HTTP2, μπορεί να αποστέλλονται, παράλληλα, πλήθος HTTP αιτημάτων από μία TCP σύνδεση. Εφόσον στο προηγούμενο YAML snapshot της διαμόρφωση DestinationRule ορίστηκε το attribute `tcp.maxConnections`, τότε θα χρησιμοποιηθεί το attribute `http.http1MaxPendingRequest`, όπως φαίνεται παρακάτω για την οριοθέτηση των αναμενόντων HTTP1 αιτημάτων.

```
http:
  http1MaxPendingRequests: 2
```

Με αυτό τον τρόπο το πλήθος των HTTP1 αναμενόντων αιτημάτων προς την υπηρεσία δεν θα ξεπεράσει τα δύο. Εάν δεν έχει οριστεί το attribute `tcp.maxConnections`, προτείνεται χρήση μόνο του `http.http2MaxRequests`.

Αξίζει να σημειωθεί ότι σύμφωνα με μία εκτενή διερεύνηση του μηχανισμού διακοπής κυκλώματος που πραγματοποιήθηκε από μία ομάδα μηχανικών του οργανισμού OLX Engineering [38], διαπιστώθηκε το εξής: τα όρια των αιτημάτων και των συνδέσεων ισχύουν ξεχωριστά για κάθε proxy που έχει δημιουργήσει Listener προς την υπηρεσία προορισμού. Επομένως εάν η υπηρεσία A διαθέτει 3 pods και στην συνέχεια εφαρμοστεί άνω όριο στην υπηρεσία B ίσο με δέκα HTTP2 αιτήματα, τότε ο συνολικός αριθμός των μέγιστων αιτημάτων που μπορεί να πραγματοποιήσει η υπηρεσία A στην B, θα είναι 30. Όμως το συνολικό όριο των 30 requests εφαρμόζεται καθολικά για την υπηρεσία προορισμού B. Επομένως, στην υπηρεσία A το συνολικό πλήθος των επιτρεπόμενων αιτημάτων δεν διαμοιράζεται ομοιόμορφα σε κάθε Pod, κάτι που είναι απρόσμενα θετικό. Το επίπεδο δεδομένων δεν λειτουργεί με κατανεμημένο τρόπο, παρ' όλ' αυτά με κάποιο τρόπο οι envoy proxies αποστέλλουν άνισο αριθμό αιτημάτων ανεξάρτητα του ατομικού ορίου, διατηρώντας την τήρηση του συνολικού ορίου αιτημάτων. Η χρήση μηχανισμού διακοπής κυκλώματος μπορεί να συνδυαστεί με την εφαρμογή ορίου ρυθμού αφικνούμενων πακέτων (rate limiting) παρέχοντας προηγμένο έλεγχο της HTTP κίνησης του πλέγματος.

Fault Injection

Στα πλαίσια της ανάπτυξης υπηρεσιών στην πλατφόρμα Istio παρέχεται ένας πολύ σημαντικός μηχανισμός προσομοίωσης σεναρίων αποτυχίας, ο οποίος ονομάζεται προσομοίωση σφαλμάτων (fault injection). Αποτελεί έναν τρόπο επικύρωσης της ορθής και αναμενόμενης λειτουργίας που ορίστηκε στην εφαρμογή. Ορισμένα πραγματικά σενάρια προσομοίωσης σφαλμάτων περιλαμβάνουν την προσομοίωση αργών ή αποτυχημένων συνδέσεων δικτύου, την εισαγωγή υψηλών καθυστερήσεων στις αποκρίσεις των υπηρεσιών, ή ακόμη και την προσωρινή διακοπή της επικοινωνίας μεταξύ των υπηρεσιών. Πραγματοποιώντας δοκιμές, οι προγραμματιστές αποκτούν γνώσεις σχετικά με τον χειρισμό διαφόρων σεναρίων αποτυχίας, εντοπίζοντας πιθανά αδύνατα σημεία. Ύστερα εκείνες οι γνώσεις μπορούν να

αξιοποιηθούν για την επιβολή κατάλληλων μέτρων και την αντιμετώπιση τυχόν λαθών, βελτιώνοντας την συνολική αξιοπιστία και σταθερότητα της εφαρμογής.

Στο πλαίσιο της προσομοίωσης υπάρχουν δύο ειδών σφάλματα που μπορούν να προσομοιωθούν σε ορισμένα τελικά σημεία και υπηρεσίες. Ο ένας τύπος αναφέρεται σε σφάλματα καθυστέρησης (delay), με τον οποίο προστίθεται ένα διάστημα καθυστέρησης κατά την απόπειρα επικοινωνίας με μία υπηρεσία. Με αυτό τον τρόπο προσομοιώνονται περιπτώσεις υπερφόρτωσης του δικτύου ή της υπηρεσίας. Ο άλλος τύπος χαρακτηρίζεται διακοπή (abort) της υπηρεσίας, που προσομοιώνεται εξ' ολοκλήρου αποτυχία της λειτουργίας. Με αυτό τον τρόπο μπορεί να επαληθευτεί ο αναμενόμενος χειρισμός αποτυχίας μιας υπηρεσίας, αποστέλλοντας στον χρήστη το κατάλληλο error page ως response. Η προσομοίωση σφαλμάτων δηλώνεται στην VirtualService διαμόρφωση κάτω από το `http.fault` attribute. Στο παρακάτω παράδειγμα φαίνεται ο τρόπος δήλωσης μίας χρονικής καθυστέρησης για μία εικονική υπηρεσία.

```
http:
- fault:
    delay:
      fixedDelay: 4s
      percentage:
        value: 100
```

Μετά την εφαρμογή της παραπάνω διαμόρφωσης, για κάθε αίτημα που αποστέλλεται στην αντίστοιχη υπηρεσία, η αποστολή της απάντησης θα καθυστερήσει τέσσερα δευτερόλεπτα. Καθυστέρηση δημιουργείται για κάθε αίτημα, διότι ο κανόνας εφαρμόζεται για το 100% της κίνησης.

Αντίστοιχα για την προσομοίωση της διακοπής μίας υπηρεσίας, θα πρέπει να οριστεί κάτω από το `fault` attribute, το ποσοστό της κίνησης για το οποίο θα μιμηθεί η προσωρινή διακοπή, καθώς και τον HTTP κωδικό που θα επιστραφεί ως απάντηση.

```
http:
- fault:
    abort:
      httpStatus: 500
      percentage:
        value: 50
```

Επομένως στο συγκεκριμένο παράδειγμα η διακοπή εφαρμόζεται για το 50% της κίνησης, επιστρέφοντας τον HTTP κωδικό 500.

5.2 Παρατηρησιμότητα

Στην ενότητα διερευνώνται τα χαρακτηριστικά παρατηρησιμότητας που παρέχει το Istio, με ιδιαίτερη έμφαση στην υπηρεσία Prometheus και Grafana . Εξετάζεται ο τρόπος με τον οποίο το Grafana παρέχει

δυνατότητες παρακολούθησης, καθώς και δυνατότητες οπτικοποίησης - ανάλυσης (σε πραγματικό χρόνο) της απόδοσης και υγείας του πλέγματος υπηρεσιών. Ακόμη γίνεται επισκόπηση των λειτουργιών του εργαλείου Kiali. Συνολικά στην ενότητα αυτή επισημαίνονται τα οφέλη που προκύπτουν από τη χρήση των εργαλείων συλλογής μετρικών και παρατηρησιμότητας για την αναπαράσταση κρίσιμων στατιστικών στοιχείων.

Prometheus

Σε ένα πλέγμα υπηρεσιών, η διατήρηση μετρήσεων και στατιστικών έχει ιδιαίτερη σημασία για την απόκτηση γνώσεων σχετικά με την απόδοση, την υγεία και τη συνολική συμπεριφορά των εφαρμογών, οι οποίες έχουν υλοποιηθεί με την αρχιτεκτονική μικροϋπηρεσιών. Οι μετρικές συμβάλλουν στην παρακολούθηση και την κατανόηση της συμπεριφοράς του πλέγματος, τον εντοπισμό πιθανών σημείων συμφόρησης και τη διασφάλιση της αποδοτικής χρήσης των πόρων. Το Istio λοιπόν, παρέχει έναν ισχυρό μηχανισμό για την αποθήκευση μετρικών, αξιοποιώντας την third-party υπηρεσία Prometheus (οργανισμός Cloud Native Computing Foundation). Η υπηρεσία Prometheus στο Istio λειτουργεί ως μια ειδική βάση δεδομένων, η οποία είναι υπεύθυνη για την αποθήκευση δεδομένων υπό την μορφή χρονοσειρών, οι οποίες παράγονται από διάφορες μικροϋπηρεσίες και στοιχεία εντός του πλέγματος. Η συλλογή των δεδομένων γίνεται από τους Envoy διακομιστές μεσολάβησης στο επίπεδο δεδομένων, συμπεριλαμβανομένων των σε τακτά χρονικά διαστήματα.

Συγκεκριμένα η υπηρεσία Prometheus παρέχει χρήσιμες μετρικές για το επίπεδο ελέγχου και επίπεδο δεδομένων ξεχωριστά. Η ενεργοποίηση αυτής της λειτουργικότητας γίνεται με την δήλωση αντίστοιχων ServiceMonitor διαμορφώσεων. Για την συλλογή πληροφοριών από το επίπεδο ελέγχου γίνεται χρήση του attribute `spec.selector.matchExpressions`, με το οποίο δηλώνονται οι υπηρεσίες, όπως citadel, pilot, galley και sidecar-injector.

```
spec:
  jobLabel: istio
  selector:
    matchExpressions:
      - {key: istio, operator: In, values:
[pilot,galley,citadel,sidecar-injector]}
```

Οι υπηρεσίες επιπέδου ελέγχου δηλώνονται με σειριακό τρόπο σε πίνακα ενός JSON αντικειμένου. Κάθε υπηρεσία επιπέδου ελέγχου διαθέτει μία πόρτα, από την οποία αποστέλλονται τα απαραίτητα δεδομένα προς συλλογή. Η πόρτα αυτή ονομάζεται `http-monitoring`, στην οποία αποστέλλει το επίπεδο ελέγχου δεδομένα προς την υπηρεσία Prometheus από το monitoring API endpoint του. Τα δεδομένα που προωθούνται, αναφέρονται σε πληροφορίες σχετικές με σφάλματα και γενικότερα απόδοσης. Για παράδειγμα ρυθμό εμφάνισης σφαλμάτων, ρυθμό άφιξης αιτημάτων και καθυστερήσεις στα pod μίας

υπηρεσίας. Ακόμη, εκτός από την παραπάνω πόρτα χρησιμοποιείται και η `http-policy-monitoring` πόρτα, για την αποστολή επιπλέον στοιχείων στην υπηρεσία Prometheus.

```
endpoints:
- port: http-monitoring
interval: 15s
- port: http-policy-monitoring
interval: 15s
```

Συγκεκριμένα προωθούνται πληροφορίες σχετικές με την επιβολή πολιτικών δρομολόγησης (DestinationRules) στο επίπεδο δεδομένων, όπως την συχνότητα εκτέλεσης εκείνων των πολιτικών και τυχών συγκρούσεων κατά την δρομολόγηση αιτημάτων προς ορισμένες υπηρεσίες προορισμού. Για παράδειγμα, ένα σενάριο σύγκρουσης αποτελεί η δημιουργία δύο εικονικών υπηρεσιών για την ίδια υπηρεσία. Τα δεδομένα για την σύνθεση μετρικών αποστέλλονται κάθε 15 δευτερόλεπτα από το επιπέδου ελέγχου, σύμφωνα με την τιμή του attribute `interval`.

Οι μετρικές που μπορούν να συλλεχθούν από το Pilot είναι οι εξής. Η μετρική `pilot_xds` αναφέρεται στο πλήθος των τελικών σημείων, τα οποία είναι συνδεδεμένα με το Pilot, μέσω των υπηρεσιών `xDS`. Επίσης, στον μετρητή `pilot_xds_pushes` αποθηκεύεται ο αριθμός των σφαλμάτων και των μηνυμάτων, που έχουν σταλεί στις υπηρεσίες `xDS`. Τέλος, η συνολική χρονική διάρκεια η οποία απαιτείται για την προώθηση μηνυμάτων προς τις `xDS` υπηρεσίες, αποθηκεύεται στην μετρική `pilot_xds_push_time`.

Για το Galley οι μετρικές που μπορούν να συλλεχθούν από την υπηρεσία Prometheus είναι οι εξής. Με την `galley_validation_failed`, μετριέται το συνολικό πλήθος των αποτυχημένων διαμορφώσεων, που αφορούν πόρους του Istio και του Kubernetes εντός του πλέγματος. Αντίστοιχα η μετρική `galley_validation_passed`, χρησιμοποιείται για την μέτρηση των επικυρωμένων διαμορφώσεων που επιβεβαιώθηκαν από το Galley.

Τέλος για το Citadel με την μετρική `citadel_server_csr_count`, μετριέται το συνολικό πλήθος των αιτημάτων υπογραφής ψηφιακών πιστοποιητικών (CSRs). Με την μετρική `citadel_server_root_cert_expiry_timestamp`, σημαίνεται η χρονική στιγμή (σε δευτερόλεπτα) κατά την οποία λήγει το ψηφιακό πιστοποιητικό της root Αρχής Πιστοποίησης (CA)

Για το επίπεδο δεδομένων ορίζεται αντίστοιχη διαμόρφωση ServiceMonitor, στην οποία δηλώνονται διάφορα endpoints από τα οποία η υπηρεσία Prometheus συλλέγει δεδομένα. Για το επίπεδο δεδομένων λαμβάνονται δεδομένα από τους envoy proxies για την παρακολούθηση των αντίστοιχων pods εφαρμογής. Συγκεκριμένα στην διαμόρφωση ServiceMonitor δηλώνεται το Envoy endpoint από το οποίο προωθούνται στατιστικά δεδομένα στην υπηρεσία Prometheus, όπως φαίνεται παρακάτω.

```
endpoints:
- path: /stats/prometheus
  targetPort: http-envoy-prom
  interval: 15s
```

Εκτός από το τελικό σημείο του Envoy, δηλώνεται και η πόρτα από την οποία προωθούνται τα δεδομένα των μετρικών, καθώς και το διάστημα αποστολής των δεδομένων

Επίσης με την χρήση του attribute `spec.selector.matchExpressions`, γίνεται φιλτράρισμα των Envoy διακομιστών, από τους οποίους θα συλλεχθούν δεδομένα. Στο συγκεκριμένο παράδειγμα επιλέγονται οι Envoy διακομιστές, οι οποίοι έχουν τιμή `DoesNotExist` στο αντίστοιχο attribute `istio-prometheus-ignore`.

```
spec:
selector:
matchExpressions:
- {key: istio-prometheus-ignore, operator: DoesNotExist}
```

Δηλαδή επιλέγονται οι envoys οι οποίοι δεν «αγνοούν» την υπηρεσία Prometheus.

Ακόμη στα πλαίσια κατανάλωσης των μετρικών από την υπηρεσία Prometheus, θα πρέπει να πραγματοποιηθεί κατάλληλη επανασημασιοδότηση (relabeling) των τίτλων στις μετρικές. Επομένως οι τίτλοι μετρικών που παράγονται από τους Envoys, ανανεώνονται δυναμικά. Για την αποθήκευση πραγματοποιείται η συγκεκριμένη επεξεργασία, με στόχο την εκτεταμένη οργάνωση και δημιουργία εγγραφών με ετικέτες που βασίζονται σε συγκεκριμένες υπηρεσίες και pods. Παρακάτω φαίνεται και ο τρόπος με τον οποίο γίνεται το relabeling εντός της διαμόρφωσης ServiceMonitor, για το label `__meta_kubernetes_namespace`, το οποίο τελικά μετατρέπεται σε `namespace`.

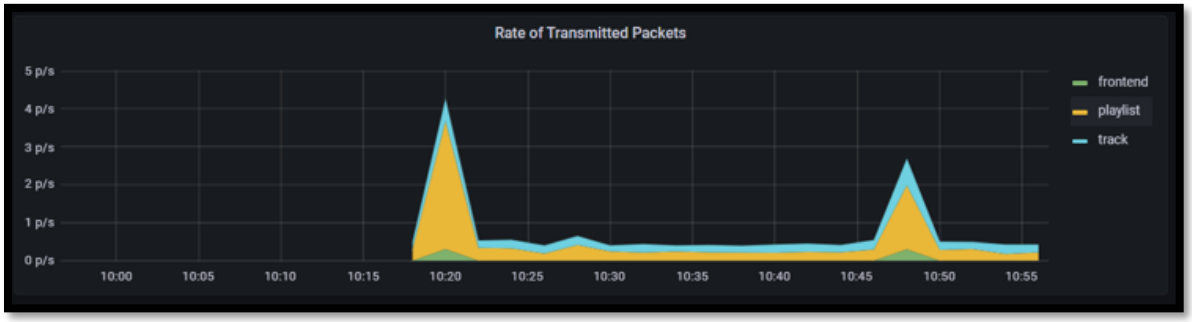
```
relabelings:
- sourceLabels: [__meta_kubernetes_namespace]
  action: replace
  targetLabel: namespace
```

Οι πιθανές μετρικές τις οποίες μπορεί να συλλέξει η υπηρεσία Prometheus από τους Envoys αναφέρονται στα ληφθέντα HTTP1, HTTP2 και gRPC αιτήματα. Συγκεκριμένα, το `istio_request_total` αποτελεί έναν μετρητή στον οποίο αποθηκεύονται τα συνολικά αιτήματα όλων των παραπάνω τύπων. Η μετρική `istio_request_duration_miliseconds` εκφράζει το χρονικό διάστημα που απαιτείται για την επεξεργασία των αιτημάτων κάθε τύπου. Στην μετρική `istio_request_bytes` αποθηκεύεται ο συνολικός αριθμός των bytes σώματος (body) που λήφθηκαν. Τέλος, με την `istio_response_bytes` μετρούνται τα συνολικά bytes σώματος των αιτημάτων, που αποστάλθηκαν από τους Envoys.

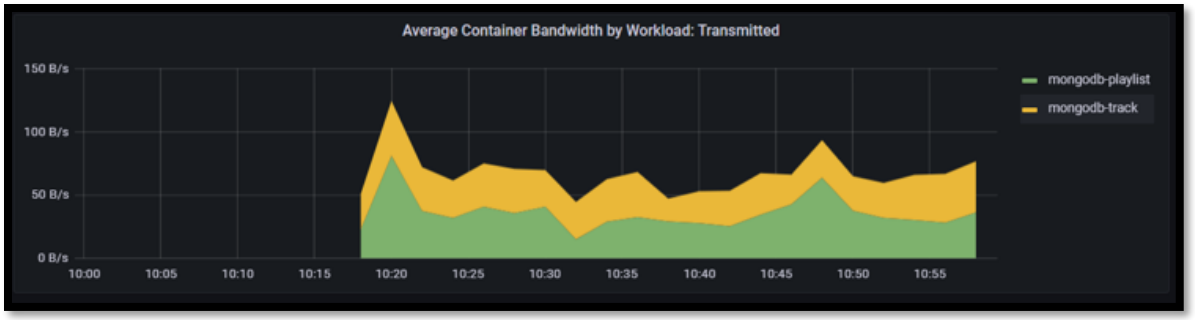
Grafana

Καθώς η υπηρεσία Prometheus πραγματοποιεί συλλογή μετρικών από τα διάφορα τελικά σημεία του του επιπέδου ελέγχου και δεδομένων, η οπτικοποίηση των αποτελεσμάτων είναι εξίσου σημαντική. Η λειτουργικότητα αυτή υλοποιείται με την χρήση της υπηρεσίας Grafana, η οποία συνδυάζεται με την υπηρεσία Prometheus, παρέχοντας μία διαισθητική αναπαράσταση των μετρικών. Επομένως, η υπηρεσία Prometheus λειτουργεί ως το backend της πλατφόρμας Grafana, με το οποίο γίνονται fetch τα δεδομένα στο Grafana UI. Η οπτικοποίηση των μετρικών γίνεται με την χρήση γραφημάτων, μετρητών, ιστογραμμάτων και πινάκων, παρέχοντας μία φιλική αίσθηση προς τον χρήστη. Οι τρόποι που μπορεί να εγκατασταθεί η υπηρεσία Grafana σε ένα σύμπλεγμα είναι είτε ως πρόσθετο add-on της πλατφόρμας Kubernetes, είτε ως υλοποίηση του Istio. Στην συγκεκριμένη περίπτωση έγινε εγκατάσταση του Grafana ως add-on της πλατφόρμας Microk8s στον χώρο ονόματος observability.

Στο UI της υπηρεσίας Grafana παρέχεται δυνατότητα αναπαράστασης γραφημάτων για κάθε χώρο ονόματος ξεχωριστά. Οι διαθέσιμοι χώροι ονόματος είναι οι cert-manager, default, ingress, istio-system, kube-system, metallb-system, observability και portainer στα πλαίσια υλοποίησης του πλέγματος της εφαρμογής μουσικής βιβλιοθήκης. Για κάθε χώρο ονόματος εμφανίζονται διάφορα είδη γραφημάτων στα οποία παρουσιάζονται τιμές μεγεθών στην μονάδα του χρόνου. Σε κάθε γράφημα εμφανίζονται τιμές για όλα τα deployments εκείνου του χώρου ονόματος. Για παράδειγμα, στον χώρο ονόματος cert-manager έχουν δημιουργηθεί οι εξής πόροι cert-manager-deployment, cert-manger-cainjector-deployment και cert-manager-webhook-deployment. Εκείνα τα deployments είναι υπεύθυνα για την διαχείριση και προώθηση των ψηφιακών πιστοποιητικών στους Envoys. Εξερευνώντας το UI της υπηρεσίας Grafana, διαπιστώνεται ότι τα είδη των γραφημάτων αναφέρονται σε χρήση επεξεργαστικής ισχύς (CPU usage, MIPS), χρήση μνήμης (Memory Usage, MiB/s) και τρέχουσας δικτυακής χρήσης (current Network Usage, packets/s, kB/s, B/s). Ακόμη παρουσιάζονται γραφήματα σχετικά με το εύρος ζώνης λήψης και μετάδοσης (kB/s) για κάθε workload – υπηρεσία, το μέσο εύρος ζώνης λήψης και μετάδοσης σε επίπεδο περιέκτη (kB/s), τον ρυθμό άφιξης και μετάδοσης πακέτων (packets/sec) και τέλος, τον ρυθμό απόρριψης ληφθέντων και μεταδιδόμενων πακέτων (packets/sec), όπως φαίνεται παρακάτω.



Εικόνα 4: Γράφημα απεικόνισης Ρυθμού Μετάδοσης Πακέτων των υπηρεσιών

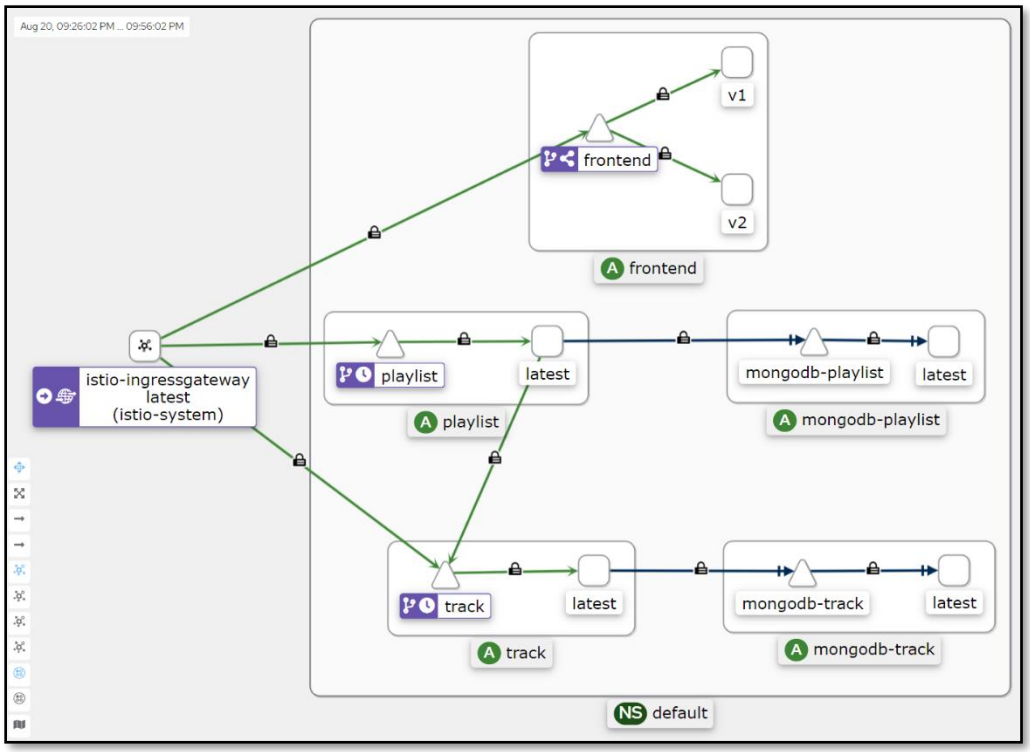


Εικόνα 5: Μέσο Εύρος Ζώνης Μετάδοσης των mongodb περιεκτών

Επίσης για διαθέσιμες βάσεις δεδομένων παρέχονται γραφήματα σχετικά με τα reads και writes ανά δευτερόλεπτο. Εκτός από αυτά τα γραφήματα, η υπηρεσία προσφέρει διάφορες build-in βιβλιοθήκες για την αλλαγή του γραφήματος σε άλλου είδους panel. Μερικά από τα διαθέσιμα είδη panel είναι bar chart, bar gauge, candle stick, canvas, dashboard list και gauge. Ακόμη προφέρεται δυνατότητα αποθήκευσης ολόκληρου του dashboard σε μορφή JSON. Ίσως, όμως η πιο σημαντική ιδιότητα του Grafana, εμπίπτει στην δημιουργία συναγερμών ή ειδοποιήσεων. Οι κανόνες εκτέλεσης κάποιου συναγερμού προωθούνται στην υπηρεσία Prometheus, μέσω του αντίστοιχου API. Στους κανόνες ορίζεται η τιμή κατωφλιού για πλήθος μετρικών, που παρέχονται από την υπηρεσία Prometheus, καθώς και ο τρόπος ειδοποίησης σε περίπτωση εκτέλεσης του κανόνα. Ειδοποιήσεις μπορούν να αποσταλούν μέσω Email, Slack και PagerDuty. Τέλος προσφέρεται δυνατότητα ομαδοποίησης κανόνων για στοχευμένη ρύθμιση του διαστήματος αξιολόγησης του κανόνα.

Kiali

Σε αντίθεση με την υπηρεσία Grafana, στην υπηρεσία Kiali προσφέρονται επιπρόσθετες λειτουργίες οπτικοποίησης του πλέγματος υπηρεσιών, δίνοντας την δυνατότητα ελέγχου ολόκληρου του οικοσυστήματος. Η αναπαράσταση του πλέγματος γίνεται με την χρήση γράφων και την αναπαράσταση των μετρικών με την χρήση dashboards. Συγκεκριμένα η υπηρεσία αναδεικνύει τις σχέσεις μεταξύ των μικροϋπηρεσιών που έχουν υλοποιηθεί στο πλέγμα, τις πολιτικές δρομολόγησης, τα φίλτρα που έχουν εφαρμοστεί στους Envoy διακομιστές μεσολάβησης για την συλλογή στατιστικών στοιχείων και διάφορα άλλα στοιχεία απόδοσης. Παρακάτω φαίνεται ο γράφος του πλέγματος υπηρεσιών της μουσικής βιβλιοθήκης από το UI της υπηρεσίας Kiali. Περαιτέρω ανάλυση των υπηρεσιών της εφαρμογής πραγματοποιείται στο Κεφάλαιο 6: «Ανάπτυξη Εφαρμογής Μικροϋπηρεσιών».



Εικόνα 6: Γράφημα του πλέγματος υπηρεσιών της μουσικής βιβλιοθήκης

Στην αναπαράσταση του πλέγματος φαίνονται οι υπηρεσίες, οι περιέκτες, ο Ingress Gateway, καθώς και οι μεταξύ τους σχέσεις. Ακόμα φαίνεται ότι για την επικοινωνία μεταξύ των υπηρεσιών είναι ενεργοποιημένο το πρωτόκολλο mTLS για την ασφαλή μεταφορά των δεδομένων μεταξύ των υπηρεσιών.

Η συλλογή πληροφοριών και μετρικών πραγματοποιείται από τις υπηρεσίες Prometheus και Grafana, οι οποίες αποτελούν το backend της υπηρεσίας. Η διαισθητική διεπαφή της υπηρεσίας Kiali μεταφράζει το περίπλοκο δίκτυο υπηρεσιών και τις αλληλεπιδράσεις τους σε οπτικές αναπαραστάσεις, δίνοντας τη δυνατότητα στους χειριστές του πλέγματος να αντιμετωπίζουν αποτελεσματικά προβλήματα, με στόχο την βελτιστοποίηση της απόδοσης του πλέγματος. Επιπλέον λειτουργίες καλύπτουν την ανίχνευση της κυκλοφορίας, την οπτικοποίηση κυκλωμάτων και την παρακολούθηση της κατάστασης υγείας, προσφέροντας μια ολιστική προσέγγιση για τη διατήρηση της ζωτικότητας, της ευρωστίας και της συνολικής αποδοτικότητας των εφαρμογών σε περιβάλλοντα ανάπτυξης πλέγματος. Η ενσωμάτωσή της υπηρεσίας απλοποιεί την κατανόηση εξελιγμένων λειτουργιών δικτύωσης του Istio, προσφέροντας εξελιγμένες μεθόδους παρακολούθησης σύνθετων αρχιτεκτονικών μικροϋπηρεσιών.

5.3 Ασφάλεια

Στο πλαίσιο ανάπτυξης εφαρμογών μικροϋπηρεσιών, το πρωτόκολλο TLS εφαρμόζεται για την κρυπτογράφηση των TCP ροών, τόσο εσωτερικά όσο και εξωτερικά του πλέγματος. Στο κεφάλαιο αυτό θα

διερευνηθούν και τα δύο σενάρια, καθώς και μία άλλη εναλλακτική επιλογή εγκαθίδρυσης εξωτερικών ασφαλών συνδέσεων με την χρήση μιας τρίτης υπηρεσίας (third-party).

SSL/TLS

Στην ταχύρρυθμα εξελισσόμενη ψηφιακή εποχή, η ιδιωτικότητα των δεδομένων και η ασφαλής μετάδοσή τους πάνω από το διαδίκτυο, αποτελεί ύψιστης σημασία προϋπόθεση. Η αποστολή ευαίσθητων πληροφοριών προς έναν διακομιστή, η επικοινωνία μεσώ ηλεκτρονικού ταχυδρομείου, η πραγματοποίηση ηλεκτρονικών συναλλαγών, η επικοινωνία μέσω πρωτοκόλλου VoIP, η περιήγηση στον ιστό και η μεταφορά αρχείων αποτελούν αναπόσπαστες διαδικασίες της καθημερινότητάς μας. Το Secure Socket Layer (SSL) και ο διάδοχός του, το Transport Layer Security (TLS), έχουν αναδειχθεί ως ο ακρογωνιαίος λίθος των πρωτοκόλλων ασφαλούς επικοινωνίας, παρέχοντας ισχυρή κρυπτογράφηση (εμπιστευτικότητα), ακεραιότητα δεδομένων και ταυτοποίηση. Η εμπιστευτικότητα των δεδομένων επιτυγχάνεται με την χρήση συμμετρικών αλγόριθμων κρυπτογράφησης, για την κρυπτογράφηση της ωφέλιμης πληροφορίας των πακέτων επιπέδου μεταφοράς. Οι αλγόριθμοι κρυπτογράφησης που χρησιμοποιούνται συνήθως είναι ο AES ή 3DES (κωδικοποίηση τμήματος). Για την αυθεντικότητα και ακεραιότητα των δεδομένων, εκτελείται κώδικας αυθεντικοποίησης μνήματος (MAC – Message Authentication Code) στο κρυπτογράφημα, με τον οποίο προκύπτει η MAC ετικέτα. Τέλος, η ταυτοποίηση ενός διακομιστή πραγματοποιείται μέσα από την λήψη και επικύρωση του ψηφιακού πιστοποιητικού από την πλευρά του πελάτη (client), χρησιμοποιώντας το δημόσιο κλειδί του διακομιστή. Το κλειδί του MAC και της συμμετρικής κρυπτογράφησης δημιουργείται από τον πελάτη, κατά την διαδικασία της TLS χειραψίας όπου αποστέλλεται κρυπτογραφημένο στον διακομιστή (κρυπτογράφηση με το δημόσιο κλειδί του διακομιστή). Η παραπάνω διαδικασία αποτελεί τον μηχανισμό του SSL/TLS πρωτοκόλλου, με την οποία δημιουργούνται ασφαλείς συνδέσεις μεταξύ πελατών και διακομιστών.

Ασφαλείς Πύλες – Gateways

Αρχικά υποθέτουμε ό,τι σε ένα πλέγμα, οι υπηρεσίες είναι προσβάσιμες από τον διαδίκτυο μέσω της πύλης Gateway. Επομένως κάθε εισερχόμενο αίτημα προς τις υπηρεσίες, διέρχεται πρώτα από εκείνο το σημείο εισόδου. Για την εγκαθίδρυση ασφαλούς σύνδεσης μεταξύ των πελατών (clients) και της πύλης του πλέγματος, θα πρέπει να δημιουργηθεί το απαραίτητο πιστοποιητικό και κλειδί του Gateway. Το ψηφιακό πιστοποιητικό είναι τύπου X.509 και το κλειδί αποτελεί το ιδιωτικό κλειδί του διακομιστή, με το οποίο κρυπτογραφούνται τα δεδομένα του ψηφιακού πιστοποιητικού. Για την παραγωγή των παραπάνω στοιχείων γίνεται χρήση εργαλείων παραγωγής ψηφιακών πιστοποιητικών, όπως για παράδειγμα το certbot και το openssl. Στην συνέχεια δημιουργείται ένας Secret πόρος στο Kubernetes, όπου αποθηκεύονται τα παραπάνω αρχεία. Αξίζει να σημειωθεί ό,τι για κάθε όνομα τομέα (domain name), θα πρέπει να δημιουργηθεί διαφορετικό πιστοποιητικό, ιδιωτικό κλειδί και μυστικός πόρος (Secret). Μετά την

δημιουργία των μυστικών πόρων (Secrets) στο Kubernetes, σειρά έχει η δήλωσή τους στην διαμόρφωση του Gateway. Κάτω από το πεδίο `servers.tls`, γίνεται αναφορά του ονόματος του μυστικού πόρου, στον οποίο θα έχει πρόσβαση ο Gateway, για την παρουσίαση του ψηφιακού πιστοποιητικού κατά την TLS χειραψία. Το όνομα τομέα της υπηρεσίας στην οποία αντιστοιχεί το πιστοποιητικό, δηλώνεται στο πεδίο `tls.hosts`. Η παραπάνω περιγραφή της διαδικασίας αναφέρεται στην ταυτοποίηση μόνο του διακομιστή (`tls.mode.SIMPLE`). Άλλη περίπτωση ταυτοποίησης αποτελεί εκείνη της αμφίδρομης ταυτοποίησης του πελάτη και του διακομιστή. Για την αμφίδρομη ταυτοποίηση, χρησιμοποιείται το πρωτόκολλο mTLS (mutualTLS) και για την ενεργοποίησης αυτής της λειτουργίας, θα πρέπει στο πεδίο `tls.mode` (διαμόρφωση Gateway), να αποδοθεί η τιμή `MUTUAL` αντί για `SIMPLE`. Επομένως και ο πελάτης θα είναι υποχρεωμένος να παρουσιάσει έγκυρο ψηφιακό πιστοποιητικό κατά την TLS χειραψία.

SSL/TLS και Υπηρεσίες πλέγματος

Στην προηγούμενη παράγραφο διερευνήθηκε η υλοποίηση του πρωτοκόλλου TLS στην πύλη πλέγματος, με τις λειτουργίες `SIMPLE` και `MUTUAL`. Και στις δύο περιπτώσεις, η ασφαλής TLS σύνδεση τερματίζεται στην πύλη και ύστερα δημιουργείται εκ νέου, μία ασφαλής σύνδεση μεταξύ της πύλης και της υπηρεσίας προορισμού. Εκτός και αν η υλοποίηση του πρωτοκόλλου TLS έχει γίνει με την λειτουργία `PASSTHROUGHT`. Τότε η TLS σύνδεση δεν τερματίζεται στην πύλη, αλλά στο αντίστοιχο τελικό σημείο της υπηρεσίας. Για την υλοποίηση του πρωτοκόλλου TLS, με στόχο την εγκαθίδρυση ασφαλών συνδέσεων μεταξύ της πύλης και των υπηρεσιών καθώς και μεταξύ διαφορετικών υπηρεσιών, υπάρχουν τρεις επιλογές: `SIMPLE`, `MUTUAL` και `ISTIO_MUTUAL`. Όταν λοιπόν, σε μία υπηρεσία προορισμού είναι ενεργοποιημένο το πρωτόκολλο TLS σε λειτουργία `SIMPLE`, τότε θα πρέπει να δημιουργηθεί και ο αντίστοιχος Secret πόρος στο Kubernetes, στον οποίο θα βρίσκονται αποθηκευμένα το ψηφιακό πιστοποιητικό και ιδιωτικό κλειδί της υπηρεσίας. Ύστερα ακολουθεί η δήλωση του Secret πόρου στον αντίστοιχο κανόνα προορισμού (DestinationRule), κάτω από το πεδίο `trafficPolicy.tls.credentialName`. Υπενθυμίζεται ότι η λειτουργία `SIMPLE` αναφέρεται μόνο στην ταυτοποίηση της υπηρεσίας προορισμού, στην συγκεκριμένη περίπτωση.

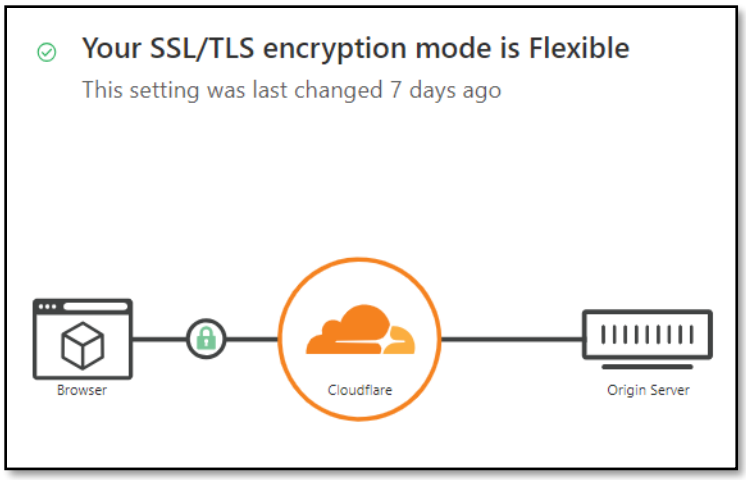
Στην περίπτωση της `MUTUAL` λειτουργίας (αμφίδρομη ταυτοποίηση ομότιμων), θα μελετηθούν δύο σενάρια χρήσης, για την απλοποίηση των πραγμάτων. Στο πρώτο σενάριο, έστω ότι υλοποιείται το πρωτόκολλο TLS σε `MUTUAL` λειτουργία μεταξύ της πύλης και μίας υπηρεσίας προορισμού. Τότε σε αυτή την περίπτωση, το πιστοποιητικό και δημόσιο κλειδί της υπηρεσίας αποθηκεύεται εντός ενός Secret πόρου, του οποίου το όνομα προστίθεται στην διαμόρφωση DestinatioRule, για την προσκόμιση του ψηφιακού πιστοποιητικού. Από την άλλη μεριά, για την επικύρωση του ψηφιακού πιστοποιητικού της πύλης, θα πρέπει τα τελικά σημεία (pods) της υπηρεσίας να έχουν πρόσβαση στον αντίστοιχο Secret πόρο της πύλης (πιστοποιητικό και δημόσιο κλειδί της πύλης). Με αυτό τον τρόπο, κατά την TLS χειραψία, τα τελικά

σημεία της υπηρεσίας, θα είναι σε θέση να επικυρώσουν το αντίστοιχο ψηφιακό πιστοποιητικό της πύλης. Επομένως απαραίτητη είναι, στην διαμόρφωση DestinationRule της υπηρεσίας, η δήλωση του μονοπατιού στο οποίο έγινε mount ο κατάλογος των Gateway Credentials. Τα πεδία στα οποία δηλώνεται το μονοπάτι είναι τα `clientCertificate`, `privateKey`, `caCerificates` (DestinationRule διαμόρφωση της υπηρεσίας). Το δεύτερο σενάριο είναι απλούστερο και αφορά την λειτουργία `MUTUAL` του πρωτοκόλλου TLS μεταξύ δύο υπηρεσιών. Σε αυτή την περίπτωση δεν υπάρχει ανάγκη δημιουργίας και προσάρτησης Secret πόρων στην υπηρεσία προορισμού. Τα credentials αποθηκεύονται τοπικά σε κάθε περιέκτη και προσκομίζονται δυναμικά από το επίπεδο δεδομένων του Istio.

Η τελευταία λειτουργία TLS που υλοποιείται στο Istio ονομάζεται `ISTIO_MUTUAL` και αναφέρεται στην αμοιβαία ταυτοποίηση μεταξύ υπηρεσιών. Αποτελεί προεπιλογή του Istio πλέγματος, με την οποία δημιουργούνται και διανέμονται από το επίπεδο ελέγχου τα απαραίτητα πιστοποιητικά στους διακομιστές μεσολάβησης Envoy των pods. Ως αρχή πιστοποίησης λειτουργεί στο επίπεδο ελέγχου το Citadel, με το οποίο αποστέλλονται τα απαραίτητα πιστοποιητικά στα τελικά σημεία. με την χρήση της υπηρεσίας SDS (Secret Discovery Service). Αρχικά οι Envoy διακομιστές μεσολάβησης αποστέλλουν αιτήματα υπογραφής πιστοποιητικού στο Citadel. Ύστερα το Citadel προωθεί τα υπογεγραμένα ψηφιακά πιστοποιητικά στο επίπεδο δεδομένων, έσω του gRPC server της υπηρεσίας SDS.

Υπηρεσία Cloudflare

Οι παραπάνω μηχανισμοί αποτελούν έναν εύκολο τρόπο διαχείρισης των TLS συνδέσεων, εσωτερικά και εξωτερικά του πλέγματος, δίχως την ανάγκη υλοποίησης εκείνης της λειτουργικότητας στην εφαρμογή. Όμως παρ’ όλ’ αυτά, απαιτείται διαχείριση και ανανέωση των ιδιωτικών κλειδιών και των πιστοποιητικών ειδικότερα στην πύλη του πλέγματος. Μία εναλλακτική επιλογή αποτελεί η υπηρεσία Cloudflare, με την οποία επιστρατεύονται μηχανισμοί IPSec, Caching, προστασίας από DDoS, ανακατεύθυνσης κίνησης, κανόνων ασφάλειας κ.α. Η ανάγκη δημιουργίας και απόδοσης credentials στην πύλη του πλέγματος, μπορεί να αντικατασταθεί με την χρήση της Zero Trust λειτουργικότητας, που παρέχεται δωρεάν από την υπηρεσία Cloudflare. Μέσω εκείνης της δυνατότητας, δημιουργώντας ένα VPN tunnel μεταξύ κάποιου Cloudflare edge διακομιστή και της πύλης πλέγματος, τα δεδομένα μεταφέρονται με ασφάλεια. Συγκεκριμένα, η εξωτερική κίνηση προς την πύλη πλέγματος, πρώτα κατευθύνεται προς τον Cloudflare edge διακομιστή, μέσω ασφαλής σύνδεσης (χρήση πρωτοκόλλου TLS) και ύστερα τερματίζεται, όπως φαίνεται παρακάτω.



Εικόνα 7: Client - Cloudflare Server TLS encryption

Στην συνέχεια, η κίνηση διέρχεται από το tunnel, όπου και τελικά καταφθάνει στην πύλη του πλέγματος. Σε αυτή την περίπτωση δεν υπάρχει ανάγκη δημιουργίας και ανανέωσης ψηφιακών πιστοποιητικών στην πύλη, αλλά εκτός αυτού παρέχεται ένα επιπλέον επίπεδο ασφάλειας καθώς ολόκληρο το αρχικό IP πακέτο κρυπτογραφείται, σε αντίθεση με το πρωτόκολλο SSL/TLS όπου κρυπτογραφείται μόνο η ωφέλιμη πληροφορία (payload) του TCP segment. Η δημιουργία VPN tunnel μπορεί να συνδυαστεί και με την χρήση πιστοποιητικού στον Gateway (TLS πρωτόκολλο), καθιστώντας ακόμη πιο ασφαλή την μετάδοση των δεδομένων από τον Cloudflare server στον Gateway του πλέγματος (διπλή κρυπτογράφηση).

Εξουσιοδότηση και Έλεγχος Πρόσβασης

Η εξουσιοδότηση ενός χρήστη αποτελεί μία αναπόσπαστη λειτουργία των εφαρμογών, που στοχεύει στην απόδοση ελέγχου πρόσβασης σε υπηρεσίες. Στην ανάπτυξη εφαρμογών μικροϋπηρεσιών, η λειτουργία της εξουσιοδότησης χρήστη, μπορεί να αφαιρεθεί από την εφαρμογή και να προστεθεί στην λειτουργικότητα του πλέγματος. Μέσω της πλατφόρμας Istio δίνεται η δυνατότητα ελέγχου του JWT ενός χρήστη από τις υπηρεσίες πλέγματος, για την εξουσιοδότηση των αιτημάτων του. Ας εξεταστεί το σενάριο όπου σε ένα πλέγμα δύο υπηρεσίες (A και B) δέχονται αιτήματα, μέσω ενός Ingress Gateway. Η υπηρεσία A αποτελείται από τον διακομιστή ταυτοποίησης – αυθεντικοποίησης, από την οποία ελέγχονται τα στοιχεία του χρήστη (credentials), όπως username και password. Μόλις ταυτοποιηθεί ο χρήστης από την υπηρεσία αυθεντικοποίησης, αποστέλλεται το JWT στον χρήστη με μία HTTPS απόκριση. Σημαντική είναι η ενεργοποίηση πρωτοκόλλου TLS στον Ingress καθώς και του mTLS στα πλαίσια του πλέγματος, για την ασφαλής μετάδοση του JWT εντός και εκτός του πλέγματος. Μετά την λήψη του JWT, ο χρήστης ύστερα επιχειρεί την αποστολή αιτήματος προς την υπηρεσία B, συμπεριλαμβάνοντας στην HTTP κεφαλίδα το JWT. Η εξουσιοδότηση του χρήστη για πρόσβαση στην υπηρεσία B, πραγματοποιείται στον Ingress Gateway με τον έλεγχο της υπογραφής του JWT. Για τον έλεγχο όμως της υπογραφής, ο Gateway

χρειάζεται το δημόσιο κλειδί του εκδότη, δηλαδή της υπηρεσίας Α. Η δυνατότητα πρόσβασης στο δημόσιο κλειδί του εκδότη γίνεται μέσω της διαμόρφωσης RequestAuthentication, όπου δηλώνεται ο εκδότης (issuer) του JWT και το uri (jwksUri) του jwks (json web token key set), όπου βρίσκεται το δημόσιο κλειδί αποθηκευμένο για την επαλήθευση της ψηφιακής υπογραφής του JWT. Σε αυτό το σημείο πρέπει να τονιστεί ότι, η επαλήθευση του JWT πραγματοποιείται στον Ingress, μέσω του αντικειμένου VirtualService της υπηρεσίας Β. Συγκεκριμένα, στην διαμόρφωση VirtualService πραγματοποιείται η δήλωση του κανόνα για την δρομολόγηση των εξουσιοδοτημένων αιτημάτων προς την επιθυμητή διαδρομή (route). Κριτήριο ενός τέτοιου κανόνα δρομολόγησης μπορεί να αποτελέσουν τα διάφορα claim πεδία του payload ενός JWT, όπως groups, name, sub κ.α.

Στην προηγούμενη παράγραφο έγινε αναφορά στον τρόπο με τον οποίο υλοποιείται η εξουσιοδότηση χρήστη βάσει JWT. Σε αυτή την παράγραφο παρουσιάζεται η επέκταση αυτού του μηχανισμού, η οποία αναφέρεται στον καθορισμό κανόνων ελέγχου πρόσβασης. Συγκεκριμένα, με την χρήση του JWT ελέγχεται εάν ο χρήστης έχει την πρόσβαση προς μία διαδρομή (route), μόνο που σε αυτή την περίπτωση μπορεί να εφαρμοστεί και πολιτική πρόσβασης μεταξύ υπηρεσιών, εκδόσεων εφαρμογών (workloads), χώρων ονόματος (namespaces) ή ακόμη και σε επίπεδο πλέγματος. Σε αυτή την περίπτωση, δηλώνεται ρητά ο τύπος κανόνα (ALLOW, DENY), καθώς και η HTTP μέθοδος αποστολής (π.χ. GET, POST) με την οποία γίνεται διάκριση των αιτημάτων. Επίσης εκτός από την διαδρομή, μπορεί να δηλωθεί και η πόρτα στην οποία απορρίπτονται ή εγκρίνονται τα αιτήματα. Η σύνταξη των κανόνων ελέγχου πρόσβασης πραγματοποιείται στις διαμορφώσεις τύπου AuthorizationPolicy. Στο τμήμα action καθορίζεται η ενέργεια, ενώ στο τμήμα rules, καθορίζεται η πηγή (from) και τα κριτήρια ταιριάσματος (to) των HTTP αιτημάτων, για τα οποία θα εκτελεστεί η ενέργεια. Η πηγή και ο προορισμός όπως αναφέρθηκε, μπορεί να είναι είτε υπηρεσία, είτε χώρος ονόματος, είτε συγκεκριμένα pods, είτε τίποτα προκειμένου η ισχύς του κανόνα να επεκταθεί σε όλους τους χώρους ονόματος του πλέγματος. Ύστερα, στο πεδίο operations δηλώνεται η HTTP μέθοδος, η πόρτα ή και το HTTP μονοπάτι προορισμού. Με αυτό τον τρόπο παρέχεται έλεγχος πρόσβασης για όλα τα επίπεδα του πλέγματος, εφαρμόζοντας στοχευμένες AuthorizationPolicy διαμορφώσεις. Σε χαμηλότερο επίπεδο, η εφαρμογή των πολιτικών γίνεται στους Envoy διακομιστές με την δημιουργία διάφορων http.rbac φίλτρων, τα οποία τοποθετούνται αμέσως μετά από τα jwt.authn φίλτρα στην αλυσίδα των φίλτρων. Επομένως, οι πολιτικές ελέγχου πρόσβασης και εξουσιοδότησης στο Istio, αποτελούν μία επέκταση της λειτουργικότητας του μηχανισμού RBAC (Kubernetes), πάνω σε http φίλτρα.

ΚΕΦΑΛΑΙΟ 6: ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ΜΙΚΡΟΫΠΗΡΕΣΙΩΝ

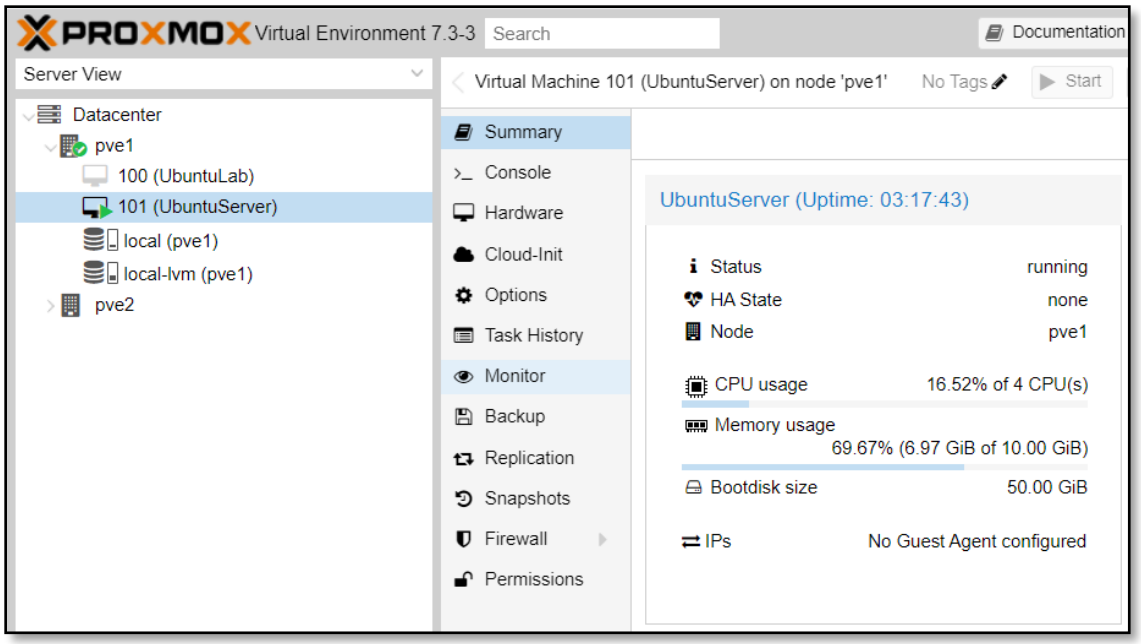
Σε αυτό το κεφάλαιο της εργασίας γίνεται επισκόπηση και περιγραφή του τρόπου λειτουργίας των διαφόρων στοιχείων που έχουν αναπτυχθεί στην εφαρμογή μουσικής βιβλιοθήκης. Το κεφάλαιο χωρίζεται σε πέντε ενότητες. Αρχικά γίνεται αναφορά των εργαλείων και της υποδομής που χρησιμοποιήθηκαν. Ακολουθεί επεξήγηση των υπηρεσιών, στις οποίες περιέχονται οι βάσεις δεδομένων για την αποθήκευση των δεδομένων, τα Playlist και Track APIs για την εκτέλεση λειτουργιών CRUD. Τέλος γίνεται ανάλυση της frontend υπηρεσίας ως την διεπαφή, η οποία αναπτύχθηκε με την χρήση του ReactJS framework. Συνολικά καλύπτονται οι λειτουργίες, η σχεδίαση των MongoDB βάσεων δεδομένων, η ανάπτυξη των backend APIs και η υλοποίηση της διεπαφής χρήστη (UI). Μέσα από αυτή την επισκόπηση, ο αναγνώστης θα είναι σε θέση να κατανοήσει την αρχιτεκτονική και την λειτουργία της εφαρμογής. Ο κώδικας των υπηρεσιών της εφαρμογής και τα αρχεία διαμόρφωσης YAML βρίσκονται στον σύνδεσμο: https://github.com/amplifier19/Kubernetes-Istio-MusicApp

6.1 Υποδομή και Εργαλεία

Παρακάτω γίνεται ανάλυση των εργαλείων που χρησιμοποιήθηκαν για την ανάπτυξη και υλοποίηση του πλέγματος υπηρεσιών της μουσικής βιβλιοθήκης, ως μία εφαρμογή μικροϋπηρεσιών. Ακόμη αναφέρονται οι λόγοι που επιλέχθηκαν τα εργαλεία εκείνα, δεδομένου του διαθέσιμου υλικού – hardware.

ProxMox

Η υλοποίηση της εφαρμογής έγινε σε bare-metal περιβάλλον, το οποίο συνολικά αποτελείται από ένα σύμπλεγμα δύο φυσικών μηχανημάτων. Η δημιουργία του συμπλέγματος έγινε με την χρήση του εργαλείου ανοιχτού κώδικα ProxMox, το οποίο αποτελεί μία πλήρης πλατφόρμα εικονικοποίησης. Το εργαλείο υποστηρίζει λειτουργίες όπως υψηλή διαθεσιμότητα (HA), δημιουργία εικονικών μηχανών (VMs) και περιεκτών (Containers), δημιουργία συμπλέγματος (Clustering), κατανεμημένου συστήματος αρχείων, VM migration και άλλων. Επίσης παρέχονται εξειδικευμένες ρυθμίσεις ασφάλειας και διαχείρισης των εικονικών περιβάλλοντων (VE), όπως έλεγχος πρόσβασης, έλεγχος ταυτότητας δύο παραγόντων (TFA) και build-in τοίχος προστασίας. Για την ανάπτυξη του κώδικα της εφαρμογής δημιουργήθηκε η εικονική μηχανή 100 (UbuntuLab) με χρήση εικόνας λειτουργικού συστήματος Ubuntu έκδοσης 18.04.6, ενώ για την υλοποίηση του πλέγματος υπηρεσιών δημιουργήθηκε η εικονική μηχανή 101 (UbuntuServer) με την χρήση εικόνας λειτουργικού συστήματος Ubuntu Server έκδοσης 22.04.



Εικόνα 8: ProxMox Virtual Environment

Στην εικονική μηχανή 101 αποδόθηκαν 4 CPU και 10 GB RAM για την υλοποίηση του πλέγματος υπηρεσιών. Στην αρχή είχαν αποδοθεί 6 GB μνήμης αλλά δεν ήταν αρκετά, καθώς το utilization ξεπερνούσε το 100% της χωρητικότητας με την εγκατάσταση του Istio. Επομένως ο φόρτος που εισάγει η εγκατάσταση της πλατφόρμας ήταν σημαντικός. Σε μία έρευνα που πραγματοποιήθηκε του 2020 [39] παρατηρήθηκε ότι ο φόρτος είναι σχεδόν διπλάσιος σε σχέση με εγγενή περιβάλλοντα Kubernetes.

Microk8s

Για την ενορχήστρωση των υπηρεσιών της μουσικής βιβλιοθήκης επιλέχθηκε η πλατφόρμα Microk8s, η οποία αποτελεί μία από τις διάφορες υλοποιήσεις του εργαλείου Kubernetes. Η επιλογή του Microk8s οφείλεται στην ευκολία χρήσης, αποτελώντας μία από τις ελαφρότερες εκδόσεις. Η εγκατάσταση της πλατφόρμας ήταν αρκετά απλή και γρήγορη, με αποτέλεσμα να εξοικονομηθεί χρόνος ο οποίος αφιερώθηκε στην ανάπτυξη της εφαρμογής και την διερεύνηση του Istio. Ακόμη, η εγκατάσταση του Istio έγινε με αντίστοιχη διευκόλυνση με την χρήση του Istio add-on της πλατφόρμας Microk8s. Ένα από τα βασικότερα χαρακτηριστικά του λοιπόν, είναι η επέκταση της λειτουργικότητας με την απλοϊκή και γρήγορη ενεργοποίηση των διαθέσιμων addons. Παρακάτω ακολουθούν τα ενεργοποιημένα addons της πλατφόρμας Microk8s στην εικονική μηχανή 101.

```
ubuntu@k8cluster:~$ microk8s status
microk8s is running
high-availability: no
  datastore master nodes: 127.0.0.1:19001
  datastore standby nodes: none
addons:
  enabled:
    istio                # (community) Core Istio service mesh services
    portainer            # (community) Portainer UI for your Kubernetes cluster
    cert-manager         # (core) Cloud native certificate management
    community            # (core) The community addons repository
    dns                  # (core) CoreDNS
    ha-cluster           # (core) Configure high availability on the current node
    helm                 # (core) Helm - the package manager for Kubernetes
    helm3                # (core) Helm 3 - the package manager for Kubernetes
    hostpath-storage     # (core) Storage class; allocates storage from host directory
    ingress              # (core) Ingress controller for external access
    metallb              # (core) Loadbalancer for your Kubernetes cluster
    metrics-server       # (core) K8s Metrics Server for API access to service metrics
    rbac                 # (core) Role-Based Access Control for authorisation
    storage              # (core) Alias to hostpath-storage add-on, deprecated
```

Εικόνα 9: Ενεργοποιημένα addons στην πλατφόρμα Microk8s

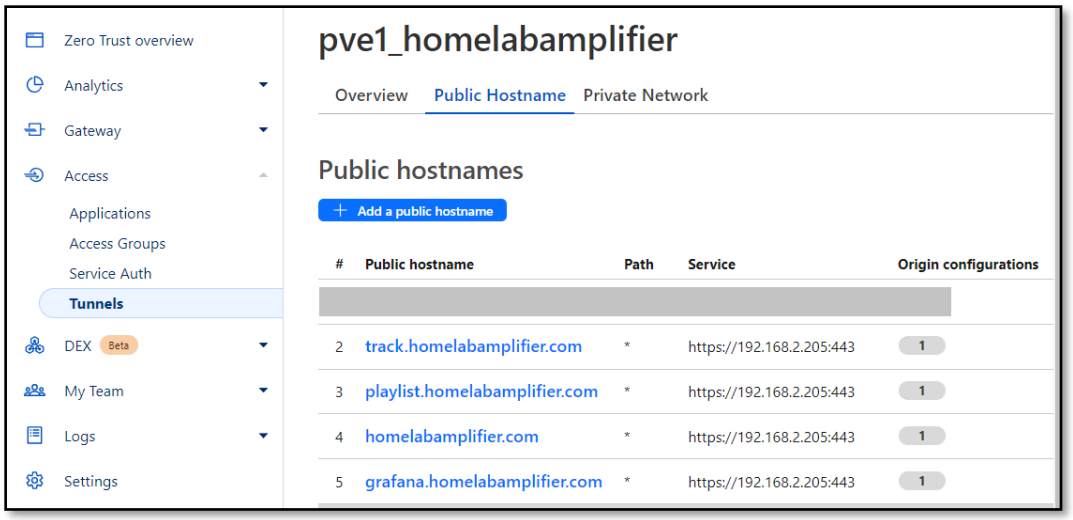
Τα παρακάτω addons προστέθηκαν μετά την εγκατάσταση της πλατφόρμας:

- Istio: Λογισμικό πλέγματος υπηρεσιών
- Portainer: Λογισμικό για την παροχή γραφικής διεπαφής του Kubernetes συμπλέγματος
- Metallb: Load Balancer λογισμικό του Kubernetes για την απόδοση External IP διεύθυνσης (192.168.2.205) στον Ingress Gateway του πλέγματος.

6.2 Ingress Gateway

Στα πλαίσια υλοποίησης των υπηρεσιών της μουσικής βιβλιοθήκης, πρώτα δημιουργήθηκε ο πόρος piplib-gateway, ο οποίος αποτελεί τον Ingress Gateway του πλέγματος. Ο Gateway δέχεται αιτήματα πρωτοκόλλου HTTPS στην πόρτα 443 και ύστερα βάση του πεδίου host στην HTTP κεφαλίδα, προωθούνται στην αντίστοιχη υπηρεσία. Παρακάτω στην υλοποίηση κάθε υπηρεσίας διαπιστώνεται ότι, για την αντιστοίχιση των υπηρεσιών με ονόματα τομέα (domain names), δημιουργήθηκαν οι αντίστοιχες Εικονικές Υπηρεσίες (VirtualServices). Με αυτό τον τρόπο για παράδειγμα, κάθε αίτημα με διεύθυνση προορισμού “homelabamplifier.com” δρομολογείται προς την ClusterIP υπηρεσία frontend και ούτω καθ΄ εξής.

Αρχικό βήμα αποτέλεσε η δημιουργία ενός VPN tunnel με την χρήση της υπηρεσίας Cloudflare για τα παρακάτω ονόματα τομέα.

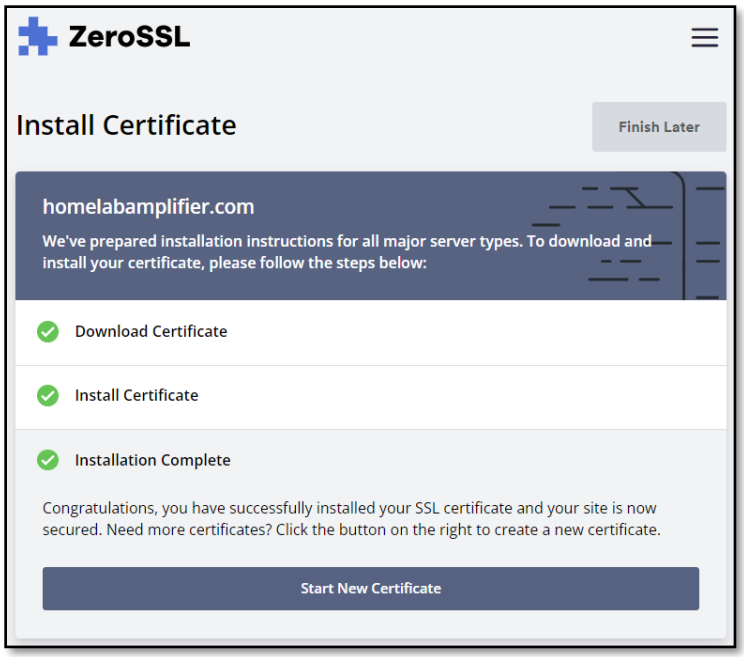


Εικόνα 10: VPN tunnel Hostnames

Με αυτό τον τρόπο τα HTTPS αιτήματα αποστέλλονται στον edge server της υπηρεσίας Cloudflare και ύστερα αποστέλλονται μέσω του VPN tunnel στον Ingress Gateway (πόρτα 443) του συμπλέγματος. Επομένως ο Cloudflare server λειτουργεί ως revers proxy, ο οποίος μεσολαβεί στην κίνηση προς τον τοπικό server του συμπλέγματος. Η λύση αυτή παρέχει τα παρακάτω πλεονεκτήματα.

- Ασφαλής σύνδεση μεταξύ πελάτη και του Cloudflare server, με την χρήση TLS.
- Χρήση της IP διεύθυνσης του Cloudflare reverse proxy και όχι της δημόσιας IP του Ingress Gateway.
- Τερματισμός TLS στον Cloudflare server και εγκαθίδρυση νέας ασφαλούς TLS σύνδεσης με τον Ingress Gateway.
- Ενθυλάκωση IP πακέτων σε νέο και κρυπτογράφηση ολόκληρου του αρχικού IP πακέτου. Συνεπώς τα HTTP πακέτα κρυπτογραφούνται δύο φορές, κατά την διαδρομή μεταξύ του Cloudflare Server και του Ingress Gateway. Μία με την χρήση του πρωτοκόλλου TLS και άλλη μία με την χρήση του ESP πρωτοκόλλου (IPSec Tunnel mode).

Για την εγκαθίδρυση ασφαλών TCP συνδέσεων μεταξύ του Cloudflare server και του Ingress Gateway, εκδόθηκε το απαραίτητο πιστοποιητικό για το όνομα τομέα “homelabamplifier.com” από την υπηρεσία ZeroSSL, όπως φαίνεται παρακάτω.



Εικόνα 11: Επιτυχής εγκατάσταση SSL πιστοποιητικού

Μαζί με το πιστοποιητικό του site συμπεριλαμβάνεται και το πιστοποιητικό της Αρχής Πιστοποίησης (CA) καθώς και το ιδιωτικό κλειδί του site. Τα παραπάνω credentials αποθηκεύτηκαν τοπικά στον κόμβο του συμπλέγματος στον φάκελο certs, όπως φαίνεται παρακάτω.

```
ubuntu@k8cluster:~/certs$ ls -l
total 12
-rw-rw-r-- 1 ubuntu ubuntu 2431 Aug 14 10:39 ca_bundle.crt
-rw-rw-r-- 1 ubuntu ubuntu 2338 Aug 14 10:39 certificate.crt
-rw-rw-r-- 1 ubuntu ubuntu 1706 Aug 14 10:39 private.key
```

Ύστερα για να είναι σε θέση ο Ingress Gateway να υπογράψει το πιστοποιητικό με το ιδιωτικό κλειδί του site, δημιουργήθηκε ο Secret πόρος gateway-credentials στην πλατφόρμα Kubernetes με την παρακάτω εντολή.

`microk8s kubectl create -n istio-system secret tls gateway-credential --key=private.key --cert=certificate.crt`

Στην συνέχεια, προστέθηκαν στην διαμόρφωση του Gateway τα παρακάτω:

```
servers:
   - port:
       number: 443
       name: https
       protocol: HTTPS
     tls:
       mode: SIMPLE
       credentialName: gateway-credential
     hosts:
       - '*'
```

Όπως φαίνεται στο πεδίο `tls.credentialName` δηλώνεται ο Secret πόρος `gateway-credential`, έτσι ώστε να προσκομιστεί το ιδιωτικό του κλειδί του site, για την υπογραφή του πιστοποιητικού από τον Ingress Gateway.

Τέλος, για την δημιουργία του piplib-gateway εκτελέστηκε η παρακάτω εντολή στον κατάλογο kubernetes/Gateway.

```
microk8s kubectl apply -f piplib-gateway.yaml
```

Ακολουθεί η επιβεβαίωση της δημιουργίας του Gateway με την εκτέλεση της εντολής:

```
microk8s kubectl get gateways -n istio-system
```

```
NAME              AGE
piplib-gateway    8d
```

6.3 Βάσεις Δεδομένων

Στην ενότητα αυτή περιγράφονται οι τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση και σύνδεση του track και playlist API με τις βάσεις δεδομένων. Ύστερα ακολουθεί απαρίθμηση των βημάτων που ακολουθήθηκαν για την ανάπτυξη των NoSQL βάσεων δεδομένων στην πλατφόρμα Kubernetes, με την χρήση mongo εικόνων, StatefulSets και Services. Στην συνέχεια γίνεται περιγραφή του σχήματος των συλλογών κάθε βάσης και των μεταξύ τους σχέσεων, δίνοντας έμφαση στην περιγραφή των πεδίων του σχήματος.

Για την ανάπτυξη των βάσεων δεδομένων έγινε χρήση της υπηρεσίας MongoDB με την οποία δημιουργήθηκαν οι δύο βάσεις δεδομένων της εφαρμογής, track και playlist. Η υπηρεσία εγκαταστάθηκε με την χρήση περιεκτών, στους οποίους υλοποιήθηκαν οι βάσεις δεδομένων. Οι εικόνα mongo που εκτελείται στους περιέκτες είναι της έκδοσης 4.0.17, η οποία αποτέλεσε την περισσότερο σταθερή έκδοση της υπηρεσίας mongo. Η αλληλεπίδραση μεταξύ των APIs και των βάσεων δεδομένων πραγματοποιήθηκε με την χρήση της JavaScript βιβλιοθήκης Mongoose, με την οποία τα σχήματα τω βάσεων δεδομένων δηλώνονται στον κώδικα. Στον φάκελο models του κώδικα των APIs δηλώθηκαν τα σχήματα (schemas) των βάσεων δεδομένων μέσω της βιβλιοθήκης Mongoose. Συγκεκριμένα, η δήλωση των σχημάτων έγινε με την χρήση της συνάρτησης mongoose.schema(), η οποία δέχεται ως παράμετρο το σχήμα σε μορφή JSON. Ύστερα η συνάρτηση επιστρέφει τον δείκτη σχήματος, ο οποίος αποθηκεύεται στην αντίστοιχη const μεταβλητή. Στην συνέχεια, για κάθε σχήμα δηλώθηκε το αντίστοιχο μοντέλο συλλογής (collection model), στο οποίο θα εκτελούνται οι διάφορες CRUD λειτουργίες της εφαρμογής. Η δήλωση του μοντέλου το οποίο θα χρησιμοποιηθεί, έγινε με την χρήση της συνάρτησης mongoose.model(), η οποία δέχεται ως παράμετρο το όνομα του μοντέλου και τον δείκτη του σχήματος. Εάν δεν υπάρχει στην βάση δεδομένων κάποια συλλογή με το όνομα του μοντέλου, δημιουργείται η αντίστοιχη συλλογή. Η συλλογή αποτελείται

από έγγραφα (documents) τύπου JSON, με τα αντίστοιχα ζευγάρια key-value που ορίστηκαν στο σχήμα της βάσης δεδομένων. Τέλος μετά την δήλωση του σχήματος και του μοντέλου, η σύνδεση του API με την βάση δεδομένων πραγματοποιείται στο αρχείο server.js, με την χρήση της συνάρτησης mongoose.connect(), η οποία δέχεται ως παράμετρο το URL της βάσης δεδομένων.

Με αφορμή την επεξήγηση της σύνδεσης των APIs με τις αντίστοιχες βάσεις δεδομένων, ας γίνει αναφορά πρώτα της αρχιτεκτονικής που ακολουθήθηκε και των πόρων που χρησιμοποιήθηκαν στην πλατφόρμα Kubernetes για την υλοποίηση των βάσεων δεδομένων. Στην προηγούμενη παράγραφο, αναλύθηκε η δομή των MongoDB βάσεων δεδομένων οι οποίες υλοποιούνται στους περιέκτες, καθώς και του τρόπου αλληλεπίδρασης των APIs με εκείνες. Για την εκτέλεση αυτών των περιεκτών, στο πλαίσιο της πλατφόρμας Kubernetes, δημιουργήθηκαν και οι αντίστοιχοι StatefulSets πόροι, με τους οποίους σε κάθε monogdb Pod παρέχεται και μόνιμος χώρος αποθήκευσης δεδομένων της τάξης του 1GB. Όμως για να είναι δυνατή η σύνδεση των APIs με εκείνα τα pods, θα πρέπει κάθε pod να κατέχει σταθερό όνομα τομέα, έτσι ώστε το URL που θα χρησιμοποιείται στην συνάρτηση mongoose.connect() να μην χρειάζεται αλλαγή κατά την αντικατάσταση ενός pod . Επειδή στα pods δεν διατίθεται σταθερό όνομα τομέα από την πλατφόρμα Kubernetes, δημιουργήθηκαν και οι κατάλληλοι Service πόροι οι οποίοι αντιστοιχούν σε εκείνα τα Pods, παρέχοντας μία σταθερή ταυτότητα δικτύου στα pods των βάσεων δεδομένων. Τα Services εκείνα διαμορφώθηκαν με τέτοιο τρόπο, έτσι ώστε να μην αποδοθεί ClusterIP διεύθυνση από το Kubernetes (`ClusterIP.none`). Η ενέργεια αυτή στοχεύει στον περιορισμό της προσβασιμότητας των υπηρεσιών, εκτός του πλέγματος, μέσω του Ingress Gateway. Επομένως η επικοινωνία με εκείνα τα Services των βάσεων δεδομένων γίνεται μόνο μέσω του ονόματος τομέα, καθιστώντας τά αυστηρώς εσωτερικές υπηρεσίες.

Για την δημιουργία των mongodb-track, mongodb-playlist StatefulSets, εκτελείται η παρακάτω εντολή στον κατάλογο kubernetes/StatefulSets.

```
microk8s kubectl apply -f .
```

Αντίστοιχα με την εκτέλεση των παρακάτω εντολων στον κατάλογο kubernetes/Services δημιουργούνται οι υπηρεσίες mongodb-track, mongodb-playlist.

```
microk8s kubectl apply -f mongodb-track-Service.yaml
microk8s kubectl apply -f mongodb-playlist-Service.yaml
```

Για να επιβεβαιώσουμε ότι η υπηρεσίες των βάσεων δεδομένων δεν διαθέτουν IP διεύθυνση αρκεί να εκτελέσουμε την εντολή: `microk8s kubectl get services | grep 'mongo'`

```
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)     AGE
mongodb-track      ClusterIP   None             <none>        27017/TCP   84d
mongodb-playlist   ClusterIP   None             <none>        27017/TCP   84d
```

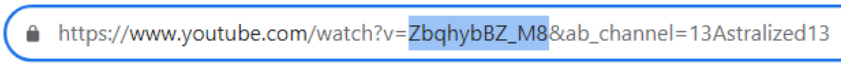
Στην έξοδο του τερματικού φαίνεται ότι οι υπηρεσίες mongo δεν διαθέτουν IP διεύθυνση, διότι η τιμή του πεδίου `CLUTER-IP` είναι ίση με `None`.

Με αυτό τον τρόπο, για παράδειγμα, το URL σύνδεσης του track API μη την υπηρεσία mongodb-track θα είναι της μορφής "mongodb://user:password@mongodb-track:27017", χρησιμοποιώντας το σταθερό όνομα τομέα mongodb-track της υπηρεσίας.

Βάση Δεδομένων Track

Το μοντέλο της συλλογής tracks δημιουργήθηκε βάση του σχήματος trackSchema, όπως αναφέρθηκε παραπάνω. Το σχήμα περιγράφει τα πεδία, καθώς και τον τύπο των τιμών που αποθηκεύονται σε κάθε έγγραφο track. Παρακάτω ακολουθούν τα πεδία για την αποθήκευση των μουσικών κομματιών:

- Στο πεδίο name αποθηκεύεται το όνομα του μουσικού κομματιού, ως αλφαριθμητικό (string). Κατά την αποθήκευση ενός μουσικού κομματιού, η απόδοση τιμής στο πεδίο name είναι απαραίτητη.
- Το όνομα του καλλιτέχνη αποθηκεύεται στο πεδίο artist, ως αλφαριθμητικό και δεν είναι απαραίτητη η απόδοση τιμής σε εκείνο το πεδίο κατά την εγγραφή.
- Το link αποτελεί το πεδίο στο οποίο αποθηκεύεται το id του μουσικού κομματιού που περιέχεται στο youtube link, ως αλφαριθμητικό και είναι απαραίτητη η απόδοση τιμής. Σε οποιοδήποτε youtube link ενός μουσικού κομματιού ή βίντεο, το id είναι η πρώτη παράμετρος με την οποία πραγματοποιείται query στην πλατφόρμα youtube, όπως φαίνεται παρακάτω.

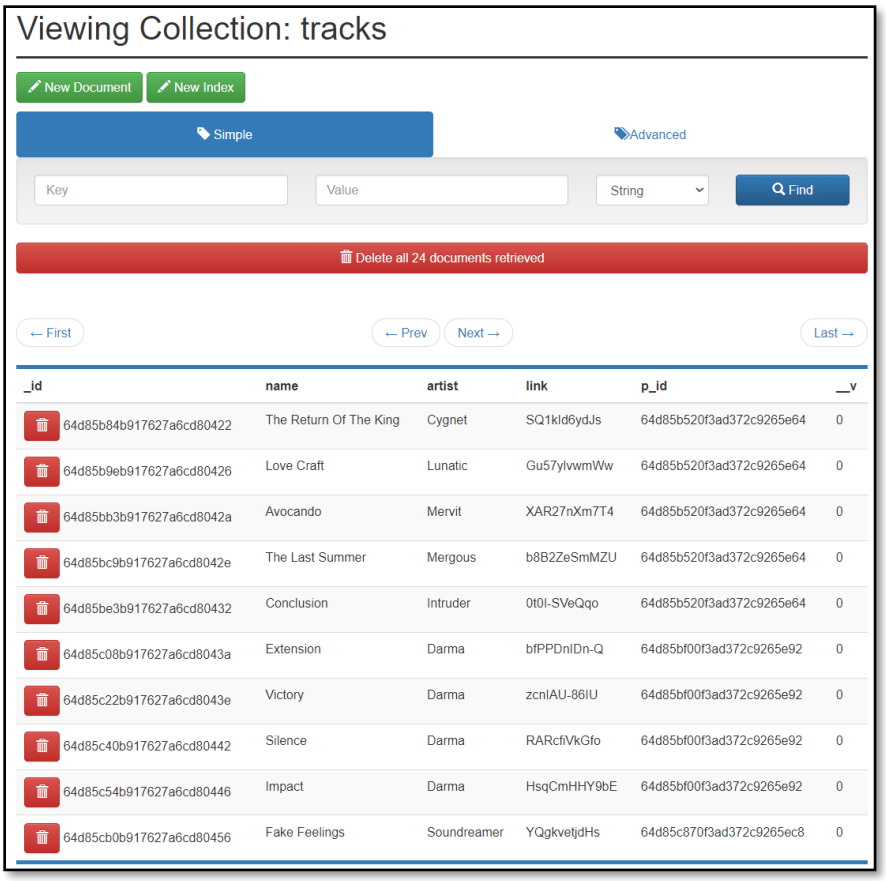


Εικόνα 12: Ενδεικτικό ID μουσικού κομματιού

Επομένως το id βρίσκεται μεταξύ του ‘watch?v=’ και του ‘&ab_channel’, σε κάθε youtube σύνδεσμο.

- Το τελευταίο πεδίο είναι το p_id του μουσικού κομματιού, το οποίο αναφέρεται στην τιμή _id του playlist, στο οποίο είναι αποθηκευμένο το μουσικό κομμάτι. Εάν το μουσικό κομμάτι βρίσκεται στην Main playlist, τότε πεδίο έχει τιμή null.

Παρακάτω φαίνονται μερικές εγγραφές μουσικών κομματιών της συλλογής tracks. Οι τιμές εμφανίζονται στο UI της εφαρμογής mongo-express, με την οποία παρέχεται εικόνα της κατάστασης στην βάση δεδομένων. Η εφαρμογή mongo-express προσομοιώθηκε προσωρινά σε pod εντός του cluster για την οπτικοποίηση των εγγραφών.



_id	name	artist	link	p_id	__v
64d85b84b917627a6cd80422	The Return Of The King	Cygnet	SQ1kld6ydJs	64d85b520f3ad372c9265e64	0
64d85b9eb917627a6cd80426	Love Craft	Lunatic	Gu57ylvwmWw	64d85b520f3ad372c9265e64	0
64d85bb3b917627a6cd8042a	Avocando	Mervit	XAR27nXm7T4	64d85b520f3ad372c9265e64	0
64d85bc9b917627a6cd8042e	The Last Summer	Mergous	b8B2ZeSmMZU	64d85b520f3ad372c9265e64	0
64d85be3b917627a6cd80432	Conclusion	Intruder	0t0l-SVeQqo	64d85b520f3ad372c9265e64	0
64d85c08b917627a6cd8043a	Extension	Darma	bfPPDnlDn-Q	64d85bf00f3ad372c9265e92	0
64d85c22b917627a6cd8043e	Victory	Darma	zcnlAU-86lU	64d85bf00f3ad372c9265e92	0
64d85c40b917627a6cd80442	Silence	Darma	RARcfiVkGfo	64d85bf00f3ad372c9265e92	0
64d85c54b917627a6cd80446	Impact	Darma	HsqCmHHY9bE	64d85bf00f3ad372c9265e92	0
64d85cb0b917627a6cd80456	Fake Feelings	Soundreamer	YQgkvetjdHs	64d85c870f3ad372c9265ec8	0

Εικόνα 13: Εγγραφές συλλογής tracks

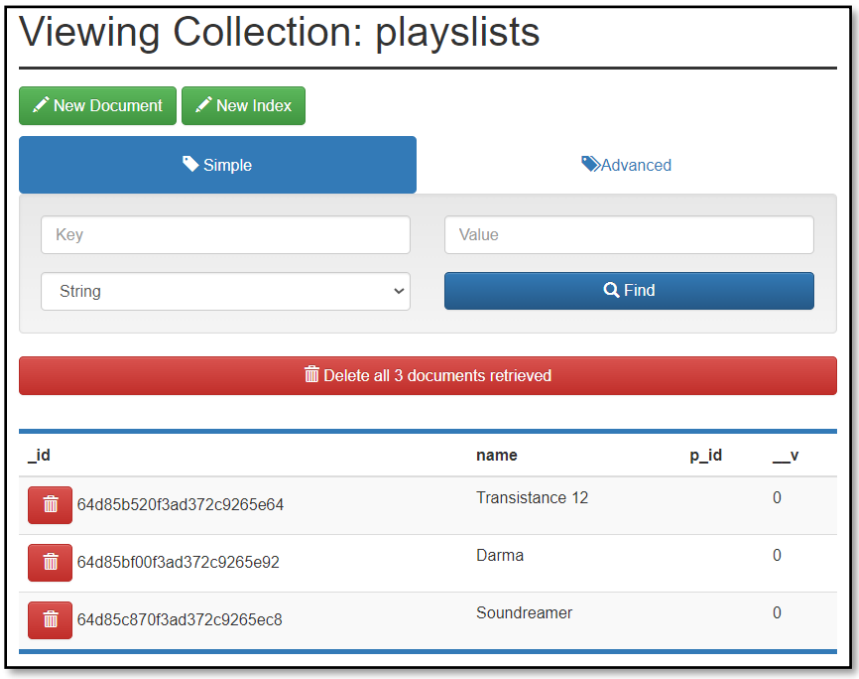
Παρατηρούμε επίσης ότι σε κάθε εγγραφή που προστίθεται, αποδίδεται αυτόματα ένα _id, στο οποίο περιέχεται μία μοναδική δεκαεξαδική (hex) τιμή μεγέθους 12 bits.

Βάση Δεδομένων Playlist

Η βάση δεδομένων αποτελείται από την συλλογή playlists, η οποία δημιουργήθηκε βάση του σχήματος playlistSchema. Το σχήμα της συλλογής playlists περιέχει τα παρακάτω πεδία:

- Στο πεδίο name αποθηκεύεται το όνομα της playlist ως αλφαριθμητικό (string). Η απόδοση τιμής είναι απαραίτητη κατά την δημιουργία μιας playlist.
- Στο πεδίο link αποθηκεύεται το link της playlist, με σκοπό την αυτόματη ενσωμάτωση μιας εξωτερικής λίστας αναπαραγωγής στην εφαρμογή της μουσικής βιβλιοθήκης. Πραγματοποιήθηκε προσπάθεια για την υλοποίηση αυτής της λειτουργικότητας με την χρήση του Mega Drive, ως εξωτερική πηγή ανακάλυψης λιστών αναπαραγωγής. Παρ' όλ' αυτά η προσπάθεια ήταν ανεπιτυχής, λόγω παρωχημένων πηγών τεκμηρίωσης του API της υπηρεσίας Mega Drive.

- Τέλος στο πεδίο p_id αποθηκεύεται το _id της γονικής λίστας αναπαραγωγής στην οποία περιέχεται το playlist. Εάν η γονική λίστα είναι η Main, τότε η τιμή του πεδίο p_id είναι ίση με null, όπως φαίνεται παρακάτω στο UI της υπηρεσίας mongo-express.



Εικόνα 14: Εγγραφές συλλογής playlists

6.4 APIs

Στην ενότητα "APIs" αναλύεται ο τρόπος υλοποίησης του backend της εφαρμογής. Συγκεκριμένα πραγματοποιείται απαρίθμηση των διαδρομών (routes) που αναπτυχθήκαν στο playlist και track API για την εκτέλεση λειτουργιών όπως δημιουργία, ανάγνωση, ενημέρωση και διαγραφή λιστών αναπαραγωγής καθώς μουσικών κομματιών. Στην συνέχεια γίνεται αναφορά στα βήματα που ακολουθήθηκαν για την δημιουργία και αποθήκευση των εικόνων περιεκτών σε μητρώο, καθώς και την υλοποίηση των υπηρεσιών στην πλατφόρμα Kubernetes, με τους αντίστοιχους πόρους.

Υπηρεσία Track

Η υπηρεσία track, αποτελεί το API από το οποίο γίνονται fetch τα δεδομένα των μουσικών κομματιών. Διαθέτει πέντε διαδρομές (routes) από τις οποίες μπορούν να πραγματοποιηθούν HTTP αιτήματα με την μέθοδο GET και POST. Οι διαδρομές υλοποιήθηκαν στο αρχείο track/routes/trackRoute.js, ενώ η λογική χειρισμού των δεδομένων και αποστολής HTTP response για κάθε διαδρομή, υλοποιήθηκε στο αρχείο track/controllers/trackController.js του κώδικα. Παρακάτω αναφέρονται οι διαθέσιμες διαδρομές του track API, καθώς και η λογική που υλοποιείται στην κάθε μία.

- Συγκεκριμένα πραγματοποιώντας αίτημα στο root μονοπάτι “/”, αποστέλλεται HTTP response με όλα τα μουσικά κομμάτια. Πριν αποσταλούν στον χρήστη τα κομμάτια, το API πραγματοποιεί το κατάλληλο query στην βάση δεδομένων mongodb-track. Ως response, αποστέλλεται πίνακας με όλα τα μουσικά κομμάτια, όπου το κάθε κομμάτι αποτελεί ένα JSON αντικείμενο.
- Με την επόμενη διαδρομή “/getByPlaylist” του track API, πραγματοποιείται HTTP αίτημα, για την αποστολή μουσικών κομματιών ενός συγκεκριμένου playlist. Τα αιτήματα αποστέλλονται με την μέθοδο GET και στο σώμα περιέχεται το _id του επιθυμητού playlist. Το track API ύστερα, αποστέλλει HTTP response με όλα τα μουσικά κομμάτια του αιτούμενου playlist, αφού πρώτα πραγματοποιηθεί το αντίστοιχο query στην βάση δεδομένων mongodb-track.
- Στην διαδρομή “/create”, αποστέλλεται HTTP αίτημα δημιουργίας ενός μουσικού κομματιού. Τα αιτήματα περιέχουν στο σώμα τους, το όνομα (name), τον καλλιτέχνη (artist), το link και το p_id του κομματιού και πραγματοποιούνται με την HTTP μέθοδο POST.
- Για την ενημέρωση (update) μιας εγγραφής κομματιού στην βάση δεδομένων mongodb-track, αποστέλλεται HTTP αίτημα με την μέθοδο POST στο μονοπάτι “/update”. Στο σώμα του HTTP πακέτου περιέχονται οι τιμές των attributes προς ενημέρωση, καθώς και το _id του κομματιού.
- Τέλος για την διαγραφή ενός μουσικού κομματιού αποστέλλεται αντίστοιχο HTTP αίτημα (POST) στο μονοπάτι “/delete”. Στο σώμα του HTTP πακέτου περιέχεται το _id του μουσικού κομματιού προς διαγραφή.

Η λογική εξυπηρέτησης των αιτημάτων, υλοποιήθηκε στο JavaScript αρχείο server.js, με την χρήση της βιβλιοθήκης express server, με την οποία υλοποιείται η εφαρμογή του διακομιστή. Η μετατροπή της εφαρμογής σε εικόνα περιέκτη πραγματοποιήθηκε με την εκτέλεση της παρακάτω εντολής στον κατάλογο /app/playlist, όπου περιέχεται το αντίστοιχο Dockerfile.

```
docker build -t dimleo/track .
```

Η εικόνα ύστερα αποθηκεύτηκε στο μητρώο dimleo της υπηρεσίας DockerHub με την εκτέλεση της παρακάτω εντολής.

```
docker push dimleo/track
```

Για την δημιουργία του playlist pod εκτελείται η παρακάτω εντολή στον κατάλογο kubernetes/Deployments.

```
mikro8s kubectl apply -f  track-Deployment.yaml
```

Στην συνέχεια, ακολουθεί η δημιουργία της υπηρεσίας track με την παρακάτω εντολή στον κατάλογο kubernetes/Services.

```
microk8s kubectl apply -f track-Service.yaml
```

Η υπηρεσία track δέχεται HTTP αιτήματα εντός του συμπλέγματος από την πόρτα 5000, τα οποία προωθούνται στην πόρτα 5000 του pod και τελικά καταλήγουν στην πόρτα 5000 του περιέκτη track. Η

υπηρεσία track είναι τύπου ClusterIP, επομένως διαθέτει τοπική IP διεύθυνση όπως φαίνεται παρακάτω στην έξοδο του τερματικού, μετά την εκτέλεση της εντολής:

```
microk8s kubectl get services | grep '^track'
```

```
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)     AGE
track               ClusterIP   10.152.183.138   <none>        5000/TCP    84d
```

Επομένως με αυτό τον τρόπο, η κίνηση προς την εσωτερική υπηρεσία track μπορεί να διαμοιράζεται στα διάφορα τελικά σημεία της υπηρεσίας.

Μετά από την ενορχήστρωση της υπηρεσίας track, σειρά έχει η δημιουργία της Εικονικής Υπηρεσίας track-virtual-service στον χώρο ονόματος istio-system. Στην διαμόρφωση της Εικονικής Υπηρεσίας, η διεύθυνση host της υπηρεσίας αντιστοιχίζεται στο όνομα τομέα “track.homelabamplifier.com”. Με αυτό τον τρόπο τα HTTP αιτήματα που διέρχονται από τον piplib-gateway και έχουν τιμή “track.homelabamplifier.com” στο πεδίο host της HTTP επικεφαλίδας, θα προωθούνται στην πόρτα 4000 της υπηρεσίας track. Για την δημιουργία της Εικονικής Υπηρεσίας track-virtual-service, εκτελείται η παρακάτω εντολής στον κατάλογο kubernetes/VirtualServices.

```
microk8s kubectl apply -f trackVirtualService.yaml
```

Για την επαλήθευση της δημιουργίας εκτελείται η παρακάτω εντολή, με σκοπό να εμφανιστεί στην έξοδο του τερματικού η εγγραφή της Εικονικής Υπηρεσίας.

```
microk8s kubectl get virtualservices -n istio-system | grep 'track'
```

```
NAME                      GATEWAYS              HOSTS                              AGE
track-virtual-service     ["piplib-gateway"]    ["track.homelabamplifier.com"]     83d
```

Τέλος, δημιουργήθηκε ο κανόνας προορισμού track-mtls, με τον οποίο ενεργοποιείται το πρωτόκολλο mTLS για την υπηρεσία track. Η εφαρμογή του κανόνα προορισμού πραγματοποιήθηκε με την εκτέλεση της παρακάτω εντολής στον κατάλογο kubernetes/DestinationRules.

```
microk8s kubectl apply -f track-DestinationRule.yaml
```

Υπηρεσία Playlist

Η υπηρεσία playlist αποτελεί το API από το οποίο γίνονται fetch τα δεδομένα των λιστών αναπαραγωγής (playlists) της μουσικής βιβλιοθήκης. Το API δέχεται HTTP αιτήματα προς οκτώ διαδρομές. Οι διαδρομές υλοποιήθηκαν στο αρχείο playlist/routes/playlistRoute.js και η λογική η οποία εκτελείται με την αποστολή αιτημάτων προς εκείνες, στο αρχείο playlist/controllers/playlistController.js. Παρακάτω ακολουθεί η λειτουργία που εκτελείται σε κάθε διαδρομή.

- Με την αποστολή HTTP αιτήματος στο root μονοπάτι "/" του playlist API αποστέλλονται όλα τα playlists, του τρέχοντος playlist στο οποίο περιηγείται ο χρήστης στον browser. Στο σώμα του HTTP αιτήματος περιέχεται το _id του τρέχοντος playlist και στο σώμα του HTTP response περιέχονται τα playlists, σε μορφή JSON.
- Η αποστολή όλων των playlist γίνεται με την αποστολή αιτήματος στο μονοπάτι "/all" του playlist API.
- Ενώ, για την αποστολή των προηγούμενων playlists πραγματοποιείται αίτημα στο μονοπάτι "/prev". Στο σώμα του αιτήματος περιέχεται το p_id και το _id του τρέχοντος playlist.
- Για την αποστολή όλων τον μουσικών κομματιών ενός playlist αποστέλλεται αίτημα στο μονπάτι "/getTracks". Στο σώμα του αιτήματος περιέχεται το _id του τρέχοντος playlist. Στην συνέχεια μέσω του playlistController αποστέλλεται GET HTTP αίτημα στο μονοπάτι "/getByplaylist" του track API.

Όλα τα αιτήματα προς τις παραπάνω διαδρομές πραγματοποιούνται με την μέθοδο GET, καθώς αφορούν διάβασμα (read) των δεδομένων της βάσης. Ενώ, τα αιτήματα προς τις υπόλοιπες τρεις διαδρομές γίνονται μέσω της μεθόδου POST.

- Συγκεκριμένα για την δημιουργία ενός playlist αποστέλλεται αίτημα στο μονοπάτι "/create". Στο σώμα του αιτήματος περιέχονται όλα τα απαραίτητα δεδομένα, όπως name και p_id, για την δημιουργία ενός playlist στην συλλογή (collection) της βάσης δεδομένων mongodb-playlist.
- Για την ενημέρωση των στοιχείων ενός playlist αποστέλλεται αίτημα προς το μονοπάτι "/update", στο οποίο περιέχονται τα στοιχεία _id, p_id, name και προαιρετικά link.
- Τέλος για την διαγραφή ενός playlist πραγματοποιείται αίτημα στο μονοπάτι "/delete", στο οποίο περιέχεται το _id του συγκεκριμένου playlist.

Όπως και στην υπηρεσία track, έτσι και στην υπηρεσία playlist η λογική της εξυπηρέτησης αιτημάτων υλοποιήθηκε στο αρχείο server.js, με την χρήση της βιβλιοθήκης express server. Η μετατροπή της εφαρμογής σε εικόνα περιέκτη πραγματοποιήθηκε με την εκτέλεση της παρακάτω εντολής στον κατάλογο /app/playlist, όπου περιέχεται το αντίστοιχο Dockerfile.

```
docker build -t dimleo/playlist .
```

Η εικόνα ύστερα αποθηκεύτηκε στο μητρώο dimleo της υπηρεσίας DockerHub με την εκτέλεση της παρακάτω εντολής.

```
docker push dimleo/playlist
```

Για την δημιουργία του playlist pod εκτελείται η παρακάτω εντολή στον κατάλογο kubernetes/Deployments.

```
mikro8s kubectl apply -f playlist-Deployment.yaml
```

Έπειτα, ακολουθεί η δημιουργία της υπηρεσίας playlist με την παρακάτω εντολή στον κατάλογο kubernetes/Services.

```
microk8s kubectl apply -f playlist-Service.yaml
```

Η υπηρεσία playlist δέχεται HTTP αιτήματα εντός του συμπλέγματος από την πόρτα 4000, τα οποία στην συνέχεια προωθούνται στην πόρτα 4000 του pod και τελικά καταλήγουν στην πόρτα 4000 του περιέκτη playlist. Η υπηρεσία playlist είναι και αυτή τύπου ClusterIP, επομένως διαθέτει τοπική IP διεύθυνση όπως φαίνεται παρακάτω στην έξοδο του τερματικού, μετά την εκτέλεση της εντολής:

```
microk8s kubectl get services | grep '^playlist'
```

```
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)     AGE
playlist            ClusterIP   10.152.183.211   <none>        4000/TCP    84d
```

Ύστερα, το επόμενο βήμα είναι η δημιουργία της Εικονικής Υπηρεσίας playlist-virtual-service στον χώρο ονόματος istio-system. Στην διαμόρφωση της Εικονικής Υπηρεσίας, η διεύθυνση host της υπηρεσίας αντιστοιχίζεται στο όνομα τομέα “playlist.homelabamplifier.com”. Με αυτό τον τρόπο τα HTTP αιτήματα που διέρχονται από τον piplib-gateway και έχουν τιμή “playlist.homelabamplifier.com” στο πεδίο host της HTTP επικεφαλίδας, θα προωθούνται στην πόρτα 4000 της υπηρεσίας playlist. Για την δημιουργία της Εικονικής Υπηρεσίας playlist-virtual-service, εκτελείται η παρακάτω εντολής στον κατάλογο kubernetes/VirtualServices.

```
microk8s kubectl apply -f playlist-VirtualService.yaml
```

Για την επαλήθευση της δημιουργίας εκτελείται η παρακάτω εντολή, με σκοπό να εμφανιστεί στην έξοδο του τερματικού η εγγραφή της Εικονικής Υπηρεσίας.

```
microk8s kubectl get virtualservices -n istio-system | grep 'playlist'
```

```
NAME                        GATEWAYS              HOSTS                                  AGE
playlist-virtual-service    ["piplib-gateway"]    ["playlist.homelabamplifier.com"]      83d
```

Τέλος, δημιουργήθηκε ο κανόνας προορισμού playlist-mtls, με τον οποίο ενεργοποιείται το πρωτόκολλο mTLS για την υπηρεσία playlist. Η εφαρμογή του κανόνα προορισμού πραγματοποιήθηκε με την εκτέλεση της παρακάτω εντολής στον κατάλογο kubernetes/DestinationRules.

```
microk8s kubectl apply -f playlist-DestinationRule.yaml
```

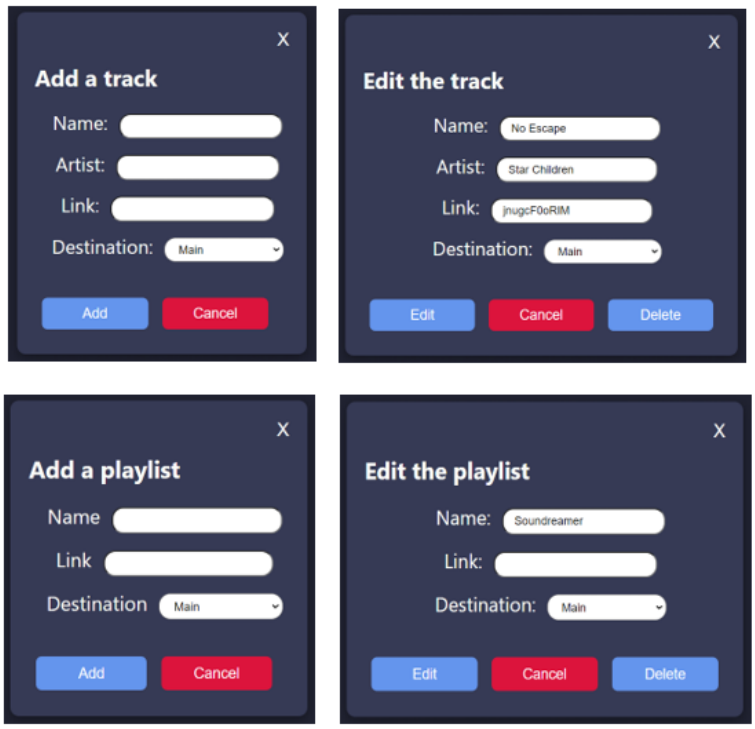
6.5 Διεπαφή Χρήστη

Η υλοποίηση της γραφικής διεπαφής της εφαρμογής έγινε μέσα από το React.js framework. Επεξηγείται ο τρόπος με τον οποίο έγινε αξιοποίηση των React Components και Web Hooks για μία διαισθητική και διαδραστική διεπαφή χρήστη. Η ενότητα εξετάζει τον τρόπο που εκτελούνται λειτουργίες διαχείρισης λιστών αναπαραγωγής και κομματιών στον browser του client. Τέλος, ακολουθεί ανάλυση των βημάτων

για την υλοποίηση της εφαρμογής στην πλατφόρμα Kubernetes, χρησιμοποιώντας τους πόρους Deployment, Service και Virtual Service.

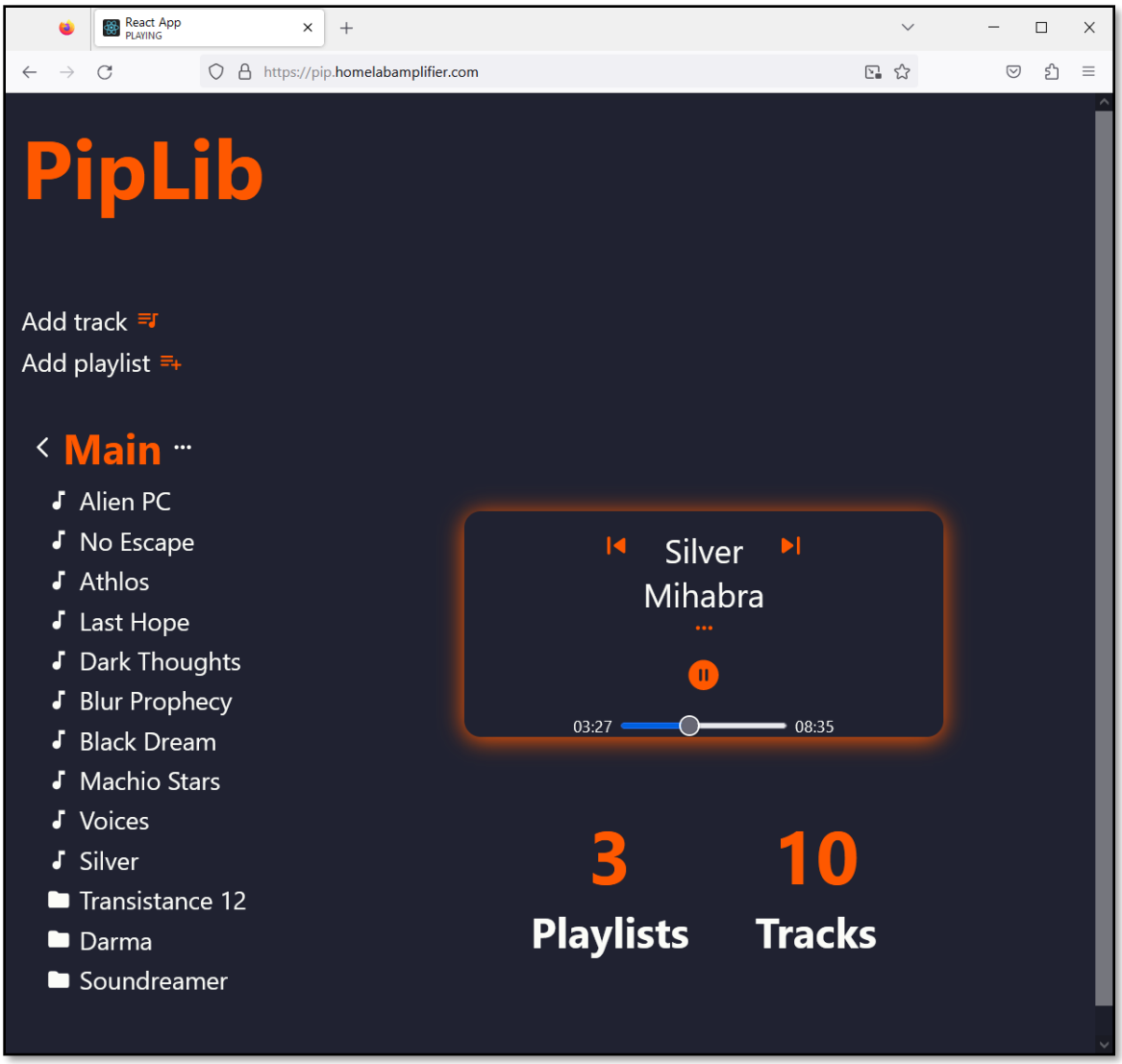
Υπηρεσία frontend

Η υπηρεσία frontend αποτελεί την διεπαφή με την οποία αλληλεπιδρά ο χρήστης. Μέσω του φυλλομετρητή (browser) δίνεται η δυνατότητα χειρισμού της μουσικής βιβλιοθήκης με γραφικό τρόπο. Η διεπαφή χρήστη έχει υλοποιηθεί σε μία μόνο σελίδα (single page website), καθώς ο χρήστης μπορεί να περιηγηθεί σε πλήθος από playlists, χωρίς να μεταβεί σε άλλο μονοπάτι. Με αυτό τον τρόπο, η δικτυακή διεπαφή χρήστη καθίσταται φιλική και προς τις φορητές συσκευές, διατηρώντας πλήρη ανταπόκριση στις δυναμικές αλλαγές της βάσης δεδομένων. Η χρήση των built-in webhooks (useState, useEffect) του ReactJS framework συντέλεσαν στην δυναμική εμφάνιση των αλλαγών που λαμβάνουν χώρα. Για παράδειγμα, όποτε πραγματοποιούταν μία προσθήκη ενός μουσικού κομματιού, ακολουθούσε η αλλαγή της τιμής μίας λογικής (boolean) μεταβλητής trackAdded με την χρήση της useState webhook. Η ανάθεση νέας τιμής στην μεταβλητή trackAdded, εκτελούσε με την σειρά της την αντίστοιχη useEffect webhook, έτσι ώστε να γίνουν ξανά fetch τα μουσικά κομμάτια, κατά την κλίση του track API. Η δομή της ιστοσελίδας υλοποιήθηκε με την χρήση των components, τα οποία επιτρέπουν την αρθρωτή ανάπτυξη μεμονωμένων τμημάτων μίας ιστοσελίδας. Ορισμένα τμήματα εμφανίζονταν στον browser μόνο σε συγκεκριμένες συνθήκες. Για παράδειγμα, κατά την ενημέρωση των στοιχείων ενός playlist εμφανίζεται ένα modal. Η εμφάνισή του γινόταν δυναμικά με το πάτημα του αντίστοιχου κουμπιού. Συγκεκριμένα η εφαρμογή διαθέτει τέσσερα modal components (PlaylistAddModal, PlaylistEditModal, TrackAddModal, TrackEditModal) για την προσθήκη και ενημέρωση/διαγραφή των μουσικών κομματιών και playlists, όπως φαίνεται παρακάτω.



Εικόνα 15: UI Modals μουσικής βιβλιοθήκης

Με αυτό τον τρόπο κάθε component εξυπηρετεί διαφορετική λειτουργικότητα στο UI της εφαρμογής. Η ροή του ήχου μεταδίδεται μέσω του δημόσιου API της Youtube, από το οποίο στέλνονται τα δεδομένα ήχου απευθείας στον browser. Η διαχείριση των λειτουργιών αναπαραγωγής μουσικού κομματιού όπως play, pause, next, previous και seek, υλοποιήθηκαν σε custom TrackPlayer component το οποίο εμφανίζεται κατά την επιλογή κομματιού, όπως φαίνεται παρακάτω στην ολοκληρωμένη εικόνα του UI.



Εικόνα 16: UI μουσικής βιβλιοθήκης

Η αναπαραγωγή των μουσικών κομματιών σε ένα Playlist γίνεται κυκλικά με συνεχή τρόπο, δίχως να υπάρχει ανάγκη επιλογής του επόμενου κομματιού, μετά το πέρας του τρέχοντος. Τα μουσικά κομμάτια παίζονται χωρίς την ύπαρξη διαφημίσεων, με την σειρά που προστέθηκαν.

Η λογική εξυπηρέτησης των αιτημάτων, υλοποιήθηκε με την χρήση ενός nginx περιέκτη, ενώ για το build της εφαρμογής frontend έγινε χρήση μιας node εικόνας περιέκτη. Η nginx εικόνα περιέκτη είναι της έκδοσης 1.15.2-alpine, ενώ η εικόνα node της έκδοσης 17-alpine.

Η δημιουργία της κύριας εικόνας περιέκτη της εφαρμογής πραγματοποιήθηκε με την εκτέλεση της παρακάτω εντολής στον κατάλογο /app/frotntend, όπου περιέχεται το αντίστοιχο Dockerfile.

```
docker build -t dimleo/frontend .
```

Ακολουθεί η αποθήκευση της εικόνας dimleo/frontend στο μητρώο dimleo της υπηρεσίας DockerHub.

```
docker push dimleo/frontend
```

Κατά την δημιουργία της κύριας εικόνας, τα αρχεία του build της εφαρμογής αντιγράφονται από τον κατάλογο app/build του node περιέκτη στον κατάλογο /usr/share/nginx/html του nginx περιέκτη, έτσι ώστε ο διακομιστής nginx να σερβίρει (serve) τον html κώδικα της εφαρμογής στον browser του χρήστη. Επίσης η διαμόρφωση του nginx περιέκτη αντικαθίσταται με εκείνη που βρίσκεται στον κατάλογο app/frontend/conf/conf.d του project. Η διαμόρφωση έχει οριστεί στο αρχείο default.conf εκείνου του καταλόγου, έτσι ώστε ο περιέκτης - διακομιστής να «ακούει» στην πόρτα 3000 για HTTP αιτήματα. Η χρήση του nginx περιέκτη έγινε με σκοπό την πραγματοποίηση δυναμικών αλλαγών στον HTML κώδικα της εφαρμογής κατά το runtime. Συγκεκριμένα στον κατάλογο usr/share/nginx/html/public βρίσκεται το αρχείο config.js, το οποίο περιέχει τα url των track και playlist API. Στο αρχείο config.js περιέχονται οι παρακάτω τιμές.

```
window.playlistURL = "https://playlist.homelabamplifier.com";
window.trackURL = "https://track.homelabamplifier.com";
```

Προσθέτοντας το παρακάτω <script/> tag στην σελίδα index.html, οι τιμές των API urls είναι διαθέσιμες σε όλο τον κώδικα του project μέσω του καθολικού αντικειμένου `window`. Επομένως αλλάζοντας το περιεχόμενο του αρχείου public/config.js, είναι δυνατή η αλλαγή των urls κατά την εκτέλεση του περιέκτη της εφαρμογής. Η επεξεργασία του αρχείου config.js γίνεται με την δημιουργία του ConfiMap πόρου frontend-config. Με αυτό το τρόπο, το περιεχόμενο του αρχείου config.js γίνεται overwrite με τα περιεχόμενα του ConfigMap πόρου. Για την δημιουργία του frontend-config εκτελείται η παρακάτω εντολή στον κατάλογο kubernetes/ConfigMap

```
microk8s kubectl apply -f frontend-ConfigMap.yaml
```

Για την δημιουργία του frontend pod εκτελείται η παρακάτω εντολή στον κατάλογο kubernetes/Deployments.

```
mikro8s kubectl apply -f  frontend-Deployment.yaml
```

Ύστερα ακολουθεί η δημιουργία της ClusterIP υπηρεσίας frontend με την εκτέλεση της παρακάτω εντολής στον κατάλογο kubernetes/Services.

```
microk8s kubectl apply -f frontend-Service.yaml
```

Η υπηρεσία frontend δέχεται HTTP αιτήματα εντός του συμπλέγματος από την πόρτα 3000, τα οποία προωθούνται στην πόρτα 3000 του pod και τελικά καταλήγουν στην πόρτα 3000 του περιέκτη frontend. Η υπηρεσία track είναι τύπου ClusterIP, επομένως διαθέτει τοπική IP διεύθυνση όπως φαίνεται παρακάτω στην έξοδο του τερματικού, μετά την εκτέλεση της εντολής:

```
microk8s kubectl get services | grep '^frontend'
```

```
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)     AGE
frontend            ClusterIP   10.152.183.156   <none>        3000/TCP    7d23h
```

Έπειτα από την δημιουργία της υπηρεσίας frontend, σειρά έχει η δημιουργία της Εικονικής Υπηρεσίας frontend-virtual-service στον χώρο ονόματος istio-system. Στην διαμόρφωση της Εικονικής Υπηρεσίας, η διεύθυνση host της υπηρεσίας αντιστοιχίζεται στο όνομα τομέα “homelabamplifier.com”. Με αυτό τον τρόπο τα HTTP αιτήματα που διέρχονται από τον piplib-gateway και έχουν τιμή “pip.homelabamplifier.com” στο πεδίο host της κεφαλίδας, θα προωθούνται στην πόρτα 3000 της υπηρεσίας frontend. Για την δημιουργία της Εικονικής Υπηρεσίας frontend-virtual-service, εκτελείται η παρακάτω εντολή στον κατάλογο kubernetes/VirtualServices.

```
microk8s kubectl apply -f frontend-VirtualService.yaml
```

Για την επαλήθευση της δημιουργίας εκτελείται η παρακάτω εντολή, με σκοπό να εμφανιστεί στην έξοδο του τερματικού η εγγραφή της Εικονικής Υπηρεσίας.

```
microk8s kubectl get virtualservices -n istio-system | grep 'frontend'
```

```
NAME                       GATEWAYS             HOSTS                          AGE
frontend-virtual-service   ["piplib-gateway"]   ["pip.homelabamplifier.com"]   84d
```

Τέλος, δημιουργήθηκε ο κανόνας προορισμού frontend-mtls, με τον οποίο ενεργοποιείται το πρωτόκολλο mTLS για την υπηρεσία frontend. Η εφαρμογή του κανόνα προορισμού πραγματοποιήθηκε με την εκτέλεση της παρακάτω εντολής στον κατάλογο kubernetes/DestinationRules.

```
microk8s kubectl apply -f frontend-DestinationRule.yaml
```

Συνολικά ο γραφικός χειρισμός της βιβλιοθήκης γίνεται μέσα από την υπηρεσία frontend. Ενώ ο χειρισμός λειτουργιών αποθήκευσης και επεξεργασίας των κομματιών και playlists πραγματοποιείται από τις υπηρεσίες playlist, track. Τέλος η μετάδοση του ήχου πραγματοποιείται με την χρήση του εξωτερικού Youtube API.

Canary Deployment

Προκειμένου να προσομοιωθεί η χρήση πολλαπλών εκδόσεων της υπηρεσίας frontend, δημιουργήθηκαν οι παρακάτω YAML διαμορφώσεις:

- frontend-v1-v2-Deployment.yaml: Αποτελείται από δύο Deployments με ένα pod στο καθένα. Στον περιέκτη του frontend-v1 pod χρησιμοποιείται η docker εικόνα έκδοσης frontend:v1. Ενώ στον περιέκτη frontend-v2, η docker εικόνα frontend:v2. Και οι δύο εικόνες βρίσκονται αποθηκευμένες στο μητρώο dimleo του DockerHub. Επίσης κάθε Deployment χαρακτηρίζεται από το αντίστοιχο version label για την δημιουργία των κατάλληλων subsets.
- frontend-v1-v2-DestinatioRule.yaml: Δημιουργία του v1 και v2 subset βάση του version label, για την ανάθεση βαρών στην εικονική υπηρεσία και ενεργοποίηση πρωτοκόλλου mTLS.

- frontend-v1-v2-VirtualService.yaml: Απόδοση βαρών σε κάθε subset για την εξυπηρέτηση του αντίστοιχου ποσοστού της εισερχόμενης κίνησης προς την υπηρεσία frontend. Συγκεκριμένα το v1 subset εξυπηρετεί το 30%, ενώ το subset v2 το 70% του συνολικού φόρτου.

Για την υλοποίηση αυτής της προσομοίωσης απαιτείται η διαγραφή των προηγούμενων πόρων με τις παρακάτω εντολές.

```
microk8s kubectl delete deployment frontend
microk8s kubectl delete destinationrule frontend-mtls
microk8s kubectl delete virtualservice frontend-virtual-service -n=istio-system
```

Έπειτα μπορούν να εφαρμοστούν οι νέες YAML διαμορφώσεις με την χρήση του microk8s kubectl utility.

ΚΕΦΑΛΑΙΟ 7: ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΠΡΟΟΠΤΙΚΕΣ

Σε αυτή την κεφάλαιο της εργασίας παρουσιάζονται τα συμπεράσματα που προέκυψαν από τη διερεύνηση της πλατφόρμας Istio καθώς και από την ανάπτυξη εφαρμογής μικροϋπηρεσιών. Επιπλέον, συζητούνται οι προοπτικές για μελλοντική έρευνα, παραθέτοντας ορισμένα ενδεικτικά αντικείμενα ανάπτυξης και μελέτης.

7.1 Συμπεράσματα

Όπως και με την τεχνολογία SDN παρέχεται ένα επιπλέον επίπεδο χειρισμού για την μηχανίκευσης της δικτυακής κίνησης στα δίκτυα των κέντρων δεδομένων. Έτσι και με τις τεχνολογίες πλέγματος όπως το Istio, προστίθεται ένα επιπλέον στρώμα λειτουργικότητας, παρέχοντας προηγμένη δρομολόγησης και διαχείρισης της κίνησης σε περιβάλλοντα μικροϋπηρεσιών. Οι πλατφόρμες πλέγματος, λειτουργούν στο επίπεδο μεταφοράς και εφαρμογής (μοντέλο OSI), ενώ η τεχνολογία SDN κυρίως σε επίπεδο δικτύου, με στόχο την παροχή ποιότητας υπηρεσίας (QoS). Στην εργασία καλύφθηκε ένα μεγάλο σύνολο από τις παροχές της πλατφόρμας Istio, οι οποίες οφείλονται στην ύπαρξη κεντρικοποιημένης διαχείρισης του πλέγματος. Με την χρήση τεχνολογιών πλέγματος, ένα μεγάλο πλήθος λειτουργιών όπως δρομολόγηση, έλεγχος πρόσβασης, εξουσιοδότηση, διακοπή κυκλώματος και οριοθέτηση ρυθμού αφαιρούνται από τον κώδικα της εφαρμογής, καθιστώντας την ανάπτυξη εφαρμογών απλούστερη. Η ενσωμάτωση του Istio σε περιβάλλοντα ανάπτυξης και παραγωγής ενός οργανισμού, επιτρέπει την πλήρης ενασχόληση της ομάδας ανάπτυξης με την επιχειρησιακή λογική των εφαρμογών. Είναι γεγονός ότι η πλατφόρμα Istio εισάγει επιπλέον φόρτο σε ένα σύμπλεγμα υπηρεσιών υπό την ενορχήστρωση του εργαλείου Kubernetes. Το φαινόμενο αυτό ελαχιστοποιήθηκε με την χρήση της πλατφόρμας Microk8s, η οποία αποτελεί μία από τις ελαφρότερες υλοποιήσεις του εργαλείου Kubernetes.

Η εφαρμογή της μουσικής βιβλιοθήκης αποτέλεσε ένα αντιπροσωπευτικό παράδειγμα ανάπτυξης ενός πλέγματος υπηρεσιών. Η αρθρωτή φύση της εφαρμογής διευκόλυνε τον καθορισμό και την ανάπτυξη των επιμέρους συνιστωσών, καθιστώντας αυτό το εγχείρημα υλοποιήσιμο. Η συνολική διαδικασία ανάπτυξης της εφαρμογής και του πλέγματος υπηρεσιών αποδείχτηκε πολύτιμη, καθώς δόθηκε η δυνατότητα αξιοποίησης πλήθους τεχνολογιών και εργαλείων αιχμής, όπως ProxMox, Cloudflare, Docker, Docker Compose, Kubernetes, Istio, Prometheus, Grafana, Kiali, React, NodeJS και MongoDB. Όσον αφορά την εφαρμογή της μουσικής βιβλιοθήκης, μέσω της πλατφόρμας Istio επιτεύχθηκε η δημιουργία μίας Istio Ingress πύλης και η απόδοση έγκυρου ψηφιακού πιστοποιητικού σε αυτήν. Ακόμη με την χρήση των εικονικών υπηρεσιών αποδόθηκαν ξεχωριστά ονόματα τομέα σε κάθε υπηρεσία, έτσι ώστε τα αιτήματα να δρομολογούνται καταλλήλως, βάση του host πεδίου (HTTP επικεφαλίδα), στις αντίστοιχες υπηρεσίες. Επίσης με την χρήση του build-in Istio πρωτοκόλλου mTLS, ήταν δυνατή η δημιουργία ασφαλών συνδέσεων μεταξύ των εσωτερικών υπηρεσιών. Τέλος αξιοποιήθηκε ο μηχανισμός Canary Deployments

του Istio, για την μετατόπισης της κίνησης μεταξύ διαφορετικών εκδόσεων μιας υπηρεσίας. Τα ποσοστά της δρομολογούμενης κίνησης προς τις διαφορετικές εκδόσεις διαπιστώθηκαν ίσα με τις προκαθορισμένες τιμές των βαρών στις εικονικές υπηρεσίες. Η επαλήθευση των τιμών ήταν δυνατή χάρη στον επιπλέον μηχανισμό τηλεμετρίας που εισάγει η πλατφόρμα Istio στο πλέγμα, για την συλλογή μετρικών και αναπαράσταση στατιστικών στοιχείων μέσω των υπηρεσιών Prometheus και Grafana.

7.2 Προοπτικές Διπλωματικής Εργασίας

Η μουσική βιβλιοθήκη ως μία εφαρμογή μικροϋπηρεσιών διαθέτει ξεχωριστές δυνατότητες βελτίωσης σε κάθε συνιστώσα. Η παρούσα έκδοση της εφαρμογής μπορεί να αξιοποιηθεί σε οποιοδήποτε ακαδημαϊκό έργο ή διατριβή που σχετίζεται θεματικά με τις υπηρεσίες νέφους, τον κλάδο DevOps και γενικότερα τις εφαρμογές μικροϋπηρεσιών. Παρακάτω ακολουθούν ορισμένα προτεινόμενα αντικείμενα μελέτης, με τα οποία μπορεί να επεκταθεί η λειτουργικότητα της παρούσας εφαρμογή της μουσικής βιβλιοθήκης ή και να χρησιμοποιηθεί ως αντικείμενο μελέτης για την διεκπεραίωση άλλων projects. Η χρήση server-side rendering framework για την υλοποίηση του UI, η σύγκριση με το client-side rendering μοτίβο και τα πλεονεκτήματα που προκύπτουν, όπως καλύτερο Search Engine Optimization κ.α. Η προσθήκη αυθεντικοποίησης και εξουσιοδότησης για την χρήση διαφορετικών εκδόσεων της εφαρμογής βάση της κατηγορίας χρήστη. Για παράδειγμα ένας admin χρήστης θα διαθέτει δυνατότητες χειρισμού του περιεχομένου της μουσικής βιβλιοθήκης από την admin έκδοση της εφαρμογής, ενώ οι χρήστες θα έχουν πρόσβαση στην κανονική έκδοση της βιβλιοθήκης, όπου θα του δίνεται μόνο η δυνατότητα περιήγησης και αναπαραγωγής (play) των μουσικών κομματιών. Άλλο αντικείμενο είναι η δημιουργία ενός πλήρως λειτουργικού κατανεμημένου replica set των ήδη υπάρχοντων mongodb βάσεων δεδομένων της εφαρμογής. Στο αντικείμενο αυτό μπορεί να γίνει υλοποίηση του replica set εξωτερικά του πλέγματος υπηρεσιών, δημιουργώντας και τον αντίστοιχο Egress Gateway πόρο. Άλλο αντικείμενο προς μελέτη, αποτελεί η υλοποίηση της εφαρμογής με την χρήση PaaS λύσεων νέφους, όπως EKS, AKS, GKS. Ακόμη, σε αυτή την προσέγγιση, θα μπορούσε να γίνει προσθήκη από DevOps pipelines για την αυτόματη δημιουργία εικόνων περιεκτών και αποθήκευσή τους σε μητρώο, όταν πραγματοποιείται κάποια ενημέρωση στο remote repository του κώδικα, όπως το GitHub. Αντίστοιχα, όταν το μητρώο περιεκτών ενημερώνεται, να αυτοματοποιείται εξίσου και η διαδικασία του deployment στην πλατφόρμα Kubernetes.

ΚΕΦΑΛΑΙΟ 8: ΠΑΡΑΡΤΗΜΑ

8.1 Βασικοί πόροι του Istio

DestinationRule

Αποτελεί κύρια πηγή διαμόρφωσης και δημιουργίας υποσυνόλων (subsets) βάση ετικετών (labels) στις διαμορφώσεις deployments. Ως host (`spec.host`) ορίζεται το κύριο σύνολο από pods, δηλαδή η τοπική IP `(svc.cluster.local)` του Service. Το αρχικό σύνολο (διαμόρφωση Service) διαιρείται σε υποσύναλα (subsets), τα οποία καλούνται και workloads. Η δημιουργία υποσυνόλων βασίζεται στην χρήση της ετικέτας έκδοσης (`spec.subsets.labels.version`) που αποδόθηκε κατά την διαμόρφωση Deployment, καθώς και από την ετικέτα περιβάλλοντος (`environment`). Ακόμη με την διαμόρφωση DestinationRule, ορίζονται πολιτικές κατανομής φόρτου ανάμεσα στα pods ενός subset ή μιας υπηρεσίας και ρυθμίσεις που αφορούν την λειτουργία του TLS πρωτοκόλλου.

Virtual Service

Αποτελεί διαμόρφωση με την οποία καθορίζονται κανόνες δρομολόγησης της κίνησης μεταξύ των υπηρεσιών. Ένας κανόνας δρομολόγησης αποτελείται κυρίως από δύο μέρη. Τα κριτήρια ταιριάσματος και τις ενέργειες. Ως κριτήρια, στους κανόνες χρησιμοποιούνται διάφορα YAML attributes ανάλογα την περίπτωση, π.χ. εξουσιοδότηση ή δρομολόγηση επιπέδου εφαρμογής. Για την δημιουργία ενός κανόνα, αρχικά δηλώνεται το όνομα τομέα ενός DNS registrar στο πεδίο `spec.hosts`. Το mapping της υπηρεσίας με το όνομα τομέα γίνεται με την χρήση του FQDN της υπηρεσίας στο πεδίο `http.route.destination.host`. Εκτός από την διεύθυνση host, μπορεί να γίνει ταίριασμα και με την διαδρομή (route) προς την οποία αποστέλλονται τα αιτήματα (πεδίο `http.match.uri.prefix`). Ύστερα, για έναν κανόνα ταιριάσματος διαδρομής (route), δηλώνεται και η ενέργεια που εκτελείται. Συνήθως, αυτή η ενέργεια ορίζεται ως η προώθηση του αιτήματος προς ένα συγκεκριμένο subset. Η δήλωση των υποσυνόλων γίνεται μέσω της ετικέτας subset στο πεδίο `spec.http.route.destination.subset`.

8.2 JWT

Στις σύγχρονες εφαρμογές, η αυθεντικοποίηση των χρηστών αποτελεί πυλώνα για την προσφορά υπηρεσιών στους αντίστοιχους χρήστες και γενικότερα για την υλοποίηση εμπορικής και επιχειρηματικής λογικής σε μία εφαρμογή. Η αυθεντικοποίηση, χαρακτηρίζεται από την ταυτοποίηση του χρήστη η οποία πραγματοποιείται με την αποστολή των credentials (όπως email, password) στην μορφή ενός http request και την επαλήθευση αυτών από την μεριά του διακομιστή. Μόλις ένας χρήστης ταυτοποιηθεί, ακολουθεί η εξουσιοδότησή του προκειμένου να αποκτήσει πρόσβαση στους αντίστοιχους πόρους. Μία από τις περισσότερο διαδεδομένες μεθόδους εξουσιοδότησης είναι η χρήση των JWT (JSON Web Tokens). Σε αυτή την μέθοδο, μετά την αυθεντικοποίηση, ο διακομιστής αποστέλλει στην μορφή ενός HTTP response ένα Json Web Token. Από εκεί και έπειτα, σε κάθε αίτημα του χρήστη συμπεριλαμβάνεται στην επικεφαλίδα, εκείνο το JWT. Ένα JWT λοιπόν, αποτελείται από 3 μέρη τα οποία σειριοποιούνται και διαχωρίζονται μεταξύ τους με " . " σε ένα ενιαίο string. Πριν την ενοποίηση των επιμέρους στοιχείων του JWT εκτελείται κωδικοποίηση των μερών σε Base64. Τα επιμέρους στοιχεία που αποτελείται ένα JWT, είναι η επικεφαλίδα (header), η ωφέλιμη πληροφορία (payload) και η ψηφιακή υπογραφή. Στην επικεφαλίδα περιέχεται ο αλγόριθμος κατακερματισμού με τον οποίο παράγεται το αποτύπωμα (signature) της ωφέλιμης πληροφορίας (payload). Το payload περιέχει διάφορα claims τα οποία ενδείκνυται να ανήκουν σε κάποια από τις κατηγορίες registered, public και private. Συνήθως χρησιμοποιούνται registered claims όπως subject, group, expiration date, issuer και άλλα. Σε αυτό το σημείο πρέπει να τονιστεί ότι η χρήση ημερομηνίας λήξης είναι απαραίτητη προϋπόθεση, καθώς με αυτό τον τρόπο αποφεύγεται η επαναλαμβανόμενη χρήση ενός υποκλεμμένου web token. Ακόμη, η χρήση ευαίσθητων πληροφοριών δεν θα πρέπει να συμπεριλαμβάνονται σε JWT, ειδικά σε περιπτώσεις όπου δεν υπάρχει VPN σύνδεση μεταξύ των άκρων. Τέλος, το τρίτο μέρος ενός JWT είναι η ψηφιακή υπογραφή. Η ψηφιακή υπογραφή προκύπτει ως η έξοδος του αλγορίθμου κατακερματισμού, με είσοδο το "header.payload". Στην περίπτωση του JWT, ο κατακερματισμός μπορεί να πραγματοποιηθεί και με συμμετρική αλλά και με ασύμμετρη κρυπτογράφηση. Το JWT εκτός από εξουσιοδότηση, παρέχει και ακεραιότητα εφόσον αποστέλλεται και το κρυπτογράφημα του "header.payload". Άρα ο διακομιστής είναι σε θέση να συγκρίνει το απεσταλμένο κρυπτογράφημα με εκείνο που υπολογίστηκε από το JWT του αιτήματος. Η χρήση του JWT για αυθεντικοποίηση και εξουσιοδότηση των χρηστών είναι προτιμότερη από άλλες μεθόδους όπως εκείνη των cookies και session IDs, καθώς δεν υπάρχει ανάγκη αποθήκευσης κάποιας πληροφορίας σχετικής με την κατάσταση του χρήστη σε διαφορετικούς διακομιστές. Το μόνο που χρειάζεται, είναι το JWT για την ταυτοποίηση ενός χρήστη σε οποιονδήποτε διακομιστή και αν αποσταλεί το αίτημα. Η χρήση των web tokens προτιμάται, χάρη στην απλότητα χειρισμού του token ως JSON αντικείμενο (προγραμματιστικές γλώσσες ιστού) και στην ευελιξία της ταυτοποίησης.

Πηγές

[1] Verma, A., et al. (2015). Large-Scale Cluster Management at Google with Borg. Proceedings of the Tenth European Conference on Computer Systems, April 17, 2015. doi:10.1145/2741948.2741964

[2] Brewe, E., (2014, June 10). An update on container support on Google Cloud Platform. https://cloudplatform.googleblog.com/2014/06/an-update-on-container-support-on-google-cloud-platform.html

[3] Khatami, A. A., Purwanto, Y., & Ruriawan, M. F. (2020). High Availability Storage Server with Kubernetes. In 2020 International Conference on Information Technology Systems and Innovation (ICITSI) (pp. 74-78). Bandung, Indonesia. doi: 10.1109/ICITSI50517.2020.9264928

[4] The Kubernetes Authors. (n.d.). Pod. https://kubernetes.io/docs/concepts/workloads/pods/

[5] The Kubernetes Authors. (n.d.). Deployments. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

[6] The Kubernetes Authors. (n.d.). Service. https://kubernetes.io/docs/concepts/services-networking/service/

[7] The Kubernetes Authors. (n.d.). Protocols for Services. https://kubernetes.io/docs/reference/networking/service-protocols/

[8] The Kubernetes Authors. (n.d.). Volumes. https://kubernetes.io/docs/concepts/storage/volumes/

[9] The Kubernetes Authors. (n.d.). Persistent Volumes. https://kubernetes.io/docs/concepts/storage/persistent-volumes/

[10] Gu, Y., Zhang, Y., Li, K., & Yu, H. (2015). Analysis of Data Replication Mechanism in NoSQL Database MongoDB. IEEE Xplore.

[11] MongoDB Inc. (n.d.). Deploy a Replica Set https://www.mongodb.com/docs/kubernetes-operator/stable/tutorial/deploy-replica-set/

[12] The Kubernetes Authors. (n.d.). ConfigMaps. https://kubernetes.io/docs/concepts/configuration/configmap/

[13] Wang, Y., & Ma, D. (2019). Developing a Process in Architecting Microservice Infrastructure with Docker, Kubernetes, and Istio. ArXiv, abs/1911.02275.

[14] The Kubernetes Authors. Secrets. (n.d.). https://kubernetes.io/docs/concepts/configuration/secret/

[15] The Kubernetes Authors. Encrypting Confidential Data at Rest. (n.d.). https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/

[16] The Kubernetes Authors. (n.d.). Certificates and Certificate Signing Requests, https://kubernetes.io/docs/reference/access-authn-authz/certificate-signing-requests/

[17] The Kubernetes Authors. Authenticating. (n.d). https://kubernetes.io/docs/reference/access-authn-authz/authentication/

[18] The Kubernetes Authors. Using RBAC Authorization. (n.d.). https://kubernetes.io/docs/reference/access-authn-authz/rbac/

[19] Talwar, V. (2017, May 25). Istio: a modern approach to developing and managing microservices. https://cloud.google.com/blog/products/gcp/istio-modern-approach-to-developing-and

[20] IBM Cloud. (n.d.). Istio add-on change log. https://cloud.ibm.com/docs/containers?topic=containers-istio-changelog#v18

[21] IBM Cloud. (n.d.). Istio add-on change log. https://cloud.ibm.com/docs/containers?topic=containers-istio-changelog#v111

[22] Istio. (n.d.). Supported Releases. https://istio.io/latest/docs/releases/supported-releases/

[23] Matt Klein. Lyft Engineering. Announcing Envoy: C++ L7 Proxy and communication bus. (2016, September 14). https://eng.lyft.com/announcing-envoy-c-l7-proxy-and-communication-bus-92520b6c8191

[24] Istio. Global Mesh Options. (n.d.). https://istio.io/latest/docs/reference/config/istio.mesh.v1alpha1/

[25] Istio. ProxyConfig. (n.d.). https://istio.io/latest/docs/reference/config/networking/proxy-config/#ProxyConfig

[26] Istio. MeshNetworks. (n.d.). https://istio.io/latest/docs/reference/config/istio.mesh.v1alpha1/#MeshNetworks

[27] Istio. Mutual TLS Migration. (n.d.). https://istio.io/v1.5/docs/tasks/security/authentication/mtls-migration/

[28] Wikipedia. Certificate signing request. (n.d.). https://en.wikipedia.org/wiki/Certificate_signing_request

[29] Istio. Security. (n.d.). https://istio.io/latest/docs/concepts/security/

[30] Istio. Citadel Health Checking. (n.d.). https://istio.io/v1.1/docs/tasks/security/health-check/

[31] Istio. Provisioning Identity through SDS. (n.d.). https://istio.io/v1.1/docs/tasks/security/auth-sds/

[32] Istio. Architecture. (n.d.). https://istio.io/v1.4/docs/ops/deployment/architecture/#galley

[33] Istio. Mixer Configuration Model. (n.d.). https://istio.io/v1.4/docs/reference/config/policy-and-telemetry/mixer-overview/

[34] Mendonca, N. C., Box, C., Manolache, C., & Ryan, L. (2021). The Monolith Strikes Back: Why Istio Migrated from Microservices to a Monolithic Architecture. IEEE Software, 38(5), 17–22. doi:10.1109/ms.2021.3080335

[35] Zsolt Varga. Cisco Tech Blog. Istio Mixerless Telemetry. (2020, April 12) https://techblog.cisco.com/blog/istio-mixerless-telemetry

[36] Istio. (n.d.). Request Routing. https://istio.io/latest/docs/tasks/traffic-management/request-routing/

[37] Istio. (n.d.). Destination Rule. https://istio.io/latest/docs/reference/config/networking/destination-rule/#TrafficPolicy-PortTrafficPolicy

[38] Kumar, V. (2020, Oct 12). Demystifying Istio Circuit Breaking. https://tech.olx.com/demystifying-istio-circuit-breaking-27a69cac2ce4

[39] Larsson, L., Tärneberg, W., Klein, C., Elmroth, E., & Kihl, M. (2020). Impact of etcd deployment on Kubernetes, Istio, and application performance. Software: Practice and Experience, 50(10), 1986-2007.