Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Δημιουργία συστήματος για παρατηρησιμότητα χρησιμοποίησης δικτύου, CPU, μνήμης και δίσκου εικονικών μηχανημάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Άννα Καλυψώ, Ι. Ποδηματά

Επιβλέπων : Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2023

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Δημιουργία συστήματος για παρατηρησιμότητα χρησιμοποίησης δικτύου, CPU, μνήμης και δίσκου εικονικών μηχανημάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Άννα Καλυψώ, Ι. Ποδηματά

Επιβλέπων : Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 18η Ιουλίου 2023.

.....	.....	.....
Π.Τσανάκας	Δ.Σούντρης	Α.Σταφυλοπάτης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2023

.....
Άννα Καλυψώ, Ι. Ποδηματά

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

Περίληψη

Η συνεχής τεχνολογική ανάπτυξη στην εποχή μας έχει επιφέρει την ανάγκη για συνεχή παρακολούθηση των υπολογιστικών συστημάτων, τόσο των φυσικών όσο και των εικονικών συστημάτων εικονικοποίησης.
Αυτό καθιστά επιτακτική την ανάγκη για δημιουργία συστημάτων παρακολούθησης σε πραγματικό χρόνο όλων των πόρων μιας δομής καθώς και την έγκαιρη και ορθή ενημέρωση σε περίπτωση σφάλματος ή προβλήματος σε οποιοδήποτε σημείο της δομής αυτής.
Στην παρούσα διπλωματική έγινε προσπάθεια για δημιουργία ενός τέτοιου συστήματος για την παρακολούθηση εικονικών μηχανημάτων (VM) και δημιουργία ειδοποιήσεων (alerts) σε πραγματικό χρόνο μέσω αναγνώρισης ανωμαλιών.

Λέξεις Κλειδιά: Prometheus, Thanos, Grafana, Real-Time Monitoring, Times series, Virtual Machine, Docker, Anomaly Detection

Abstract

The constant rise,in our days, in the technology field brings on the need for constant monitoring of computer systems, both physical and virtual systems and the containerized environments.
As a result, there is a need for creating monitoring systems in real time for all the resources of an infrastructure, accompanied with systems of accurate and on time alerting in case of errors or issues at any site of the infrastructure.
Aim of this diploma thesis is to try to create a system like the one described above for monitoring of virtual machines and alerting for any issues in real time through anomaly detection.

Keywords: Prometheus, Thanos, Grafana, Real-Time Monitoring, Times Series, Virtual Machine, Docker, Anomaly Detection

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Παναγιώτη Τσανάκα για την εμπιστοσύνη που μου έδειξε και την ευκαιρία που μου έδωσε να ασχοληθώ με το συγκεκριμένο θέμα. Στη συνέχεια θα ήθελα να ευχαριστήσω τους Βρεττό Μουλο και Άγγελο Κολαιτη για την άψογη συνεργασία που είχαμε καθόλη την διάρκεια εκπόνησης της διπλωματικής μου. Οι γνώσεις και η συνεχής διαθεσιμότητά τους ήταν καθοριστικές για την ολοκλήρωση της διπλωματικής αυτής.

Επίσης, θα ήθελα να ευχαριστήσω την οικογένειά μου που με στήριξαν καθ'όλη την διάρκεια των σπουδών μου καθώς και για την υπομονή και επιμονή που έδειξαν τις φορές που αντιμετώπισα δυσκολίες.

Τέλος, θα ήθελα να ευχαριστήσω τους φίλους μου για όλες τις όμορφες στιγμές που μου χάρισαν κατά την διάρκεια των φοιτητικών μου χρόνων.

Άννα

Πίνακας Περιεχομένων

Πίνακας Εικόνων

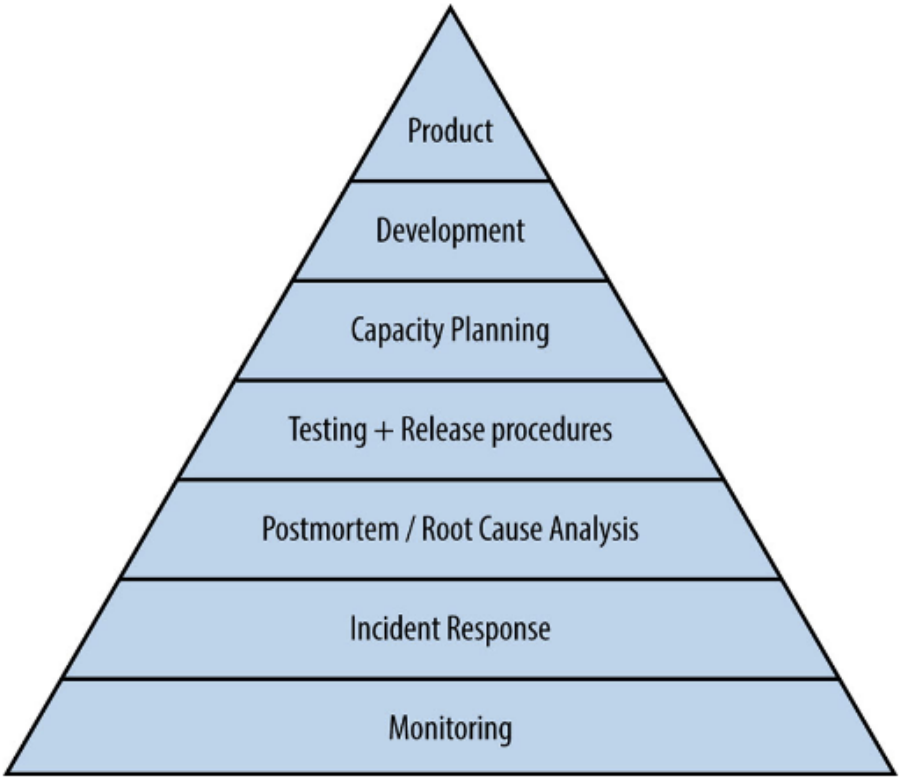
Κεφάλαιο 1

Εισαγωγή

1.1 Περιγραφή του προβλήματος

Στην σημερινή εποχή κάθε IT υποδομή αποτελείται από πολλά και διαφορετικά συστήματα και εφαρμογές. Για κάθε οργανισμό / εταιρεία είναι απαραίτητο να υπάρχει απόλυτη εποπτεία της υποδομής, συνεπώς όλων των διαφορετικών τμημάτων από τα οποία αυτή αποτελείται. Για παράδειγμα τις δικτυακές συσκευές αλλά και τη δικτυακή κίνηση, τα φυσικά ή τα εικονικά μηχανήματα που φιλοξενούν τις εφαρμογές, καθώς και τις ίδιες τις εφαρμογές. Σε αυτό το σημείο αξίζει να σημειωθεί πως με την ανάπτυξη των Cloud περιβαλλόντων η ανάγκη για παρατηρησιμότητα είναι μεγαλύτερη και πολυπλοκότερη.

Η παρακολούθηση της υποδομής αποτελεί τον ακρογωνιαίο λίθο για να καταστεί μια υπηρεσία ή μια εφαρμογή που φιλοξενείται σε αυτή αξιόπιστη.



Εικόνα 1.1: Service Reliability Hierarchy

Η παρούσα διπλωματική εστιάζει στην παρακολούθηση σε πραγματικό χρόνο (Real Time Monitoring) και στην ειδοποίηση σε περίπτωση προβλήματος, μέσω αναγνώρισης ανωμαλιών (Anomaly Detection), στη λειτουργία ενός εικονικού μηχανήματος (VM). Με τον όρο Real Time Monitoring εννοούμε την συλλογή μετρήσεων η ανάλυση των οποίων μπορεί να δώσει μια εικόνα για την κατάσταση του μηχανήματος σε πραγματικό χρόνο. Η παρακολούθηση σε πραγματικό προϋποθέτει την συνεχή συλλογή των δεδομένων από το εικονικό μηχάνημα καθώς και την διάθεση των δεδομένων για μεγάλο χρονικό διάστημα ώστε να είναι πιο σωστή η πρόβλεψη πάνω σε αυτα. Για την ολοκληρωτική παρακολούθηση ενός εικονικού μηχανήματος χρειάζονται μετρήσεις που να αφορούν την CPU, την μνήμη RAM, τον δίσκο κ.λ.π.

Μερικοί από τους λόγους για τους οποίους είναι σημαντικό όλα τα VMs μιας υποδομής να παρακολουθούνται παρουσιάζονται παρακάτω:

1. Η γρήγορη αναγνώριση ενός προβλήματος που παρουσιάζεται σε ένα VM βοηθάει στην αμεσότερη επίλυση του και με αυτό επιτυγχάνεται η μείωση της διακοπής λειτουργίας (downtime) του VM καθώς και της υπηρεσίας που αυτό φιλοξενεί.
2. Με την συνεχή παρακολούθηση ενός VM μπορούν να εντοπιστούν περιπτώσεις μειωμένης επίδοσης και να αναγνωριστούν πιθανά κωλύματα της υπηρεσίας / εφαρμογής και να διορθωθούν πριν προκαλέσουν κάποια σοβαρό καθολικό πρόβλημα
3. Μέσω των μετρήσεων που συλλέγονται και αξιολογούνται υπάρχει η δυνατότητα δημιουργίας ειδοποιήσεων (alerts) οι οποίες θα ενεργοποιούν αυτόματους μηχανισμούς αποκατάστασης προβλημάτων χωρίς να χρειάζεται η ανθρώπινη παρέμβαση.
4. Μέσω των μετρήσεων σε πραγματικό χρόνο δίνεται η δυνατότητα για καλύτερες και πιο σωστές αποφάσεις όσων αφορά την πρόβλεψη των αναγκών της δομής στο μέλλον. Κατα συνέπεια μειώνεται και το οικονομικό κόστος, καθώς υπολογίζονται μόνο τα απαραίτητα έξοδα.

Όλα τα παραπάνω γίνονται ακόμα πιο επωφελή όταν η παρακολούθηση ξεπερνάει τα πλαίσια των εικονικών μηχανημάτων και εφαρμόζεται σε ολόκληρη την υποδομή αλλα και τις εφαρμογές. Αξίζει να σημειωθεί πως πλέον η παρακολούθηση σε πραγματικό χρόνο γίνεται και περισσότερο επιτακτική ανάγκη και πάρα πολλές εταιρείες και οργανισμοί την υλοποιούν για να ακολουθούν τα πρότυπα συμβατότητας σε επίπεδο ασφάλειας και προστασίας δεδομένων που ορίζονται σε διεθνές επίπεδο καθώς οφείλουν να μοιράζουν αναφορές (reports) για τα συστήματα τους.

1.2 Οργάνωση Κειμένου

Η παρούσα διπλωματική εργασία αποτελείται από έξι (6) κεφάλαια. Στο Κεφάλαιο 1 παρουσιάζεται το πρόβλημα που προσπαθεί να αντιμετωπίσει η εργασία καθώς και τα οφέλη που υπάρχουν από την διαδικασία που εφαρμόζεται. Τέλος γίνεται μία σύντομη παρουσίαση του αντικειμένου της.

Στο Κεφάλαιο 2 παρουσιάζονται οι απαραίτητες έννοιες οι οποίες στοχεύουν στην δημιουργία του κατάλληλου θεωρητικού υπόβαθρου για την κατανόηση της εργασίας.

Συγκεκριμένα αναλύονται οι έννοιες των χρονοσειρών, της μηχανικής μάθησης και της ανίχνευσης ανωμαλιών.

Στο Κεφάλαιο 3 παρουσιάζεται θεωρητικά η αρχιτεκτονική του συστήματος παρακολούθησης που δημιουργήθηκε στα πλαίσια της εργασίας. Πιο συγκεκριμένα, γίνεται μια θεωρητική εισαγωγή στα εργαλεία ανοικτού κώδικα που χρησιμοποιήθηκαν και τον ρόλο που εξυπηρετούν στην εργασία. Τα εργαλεία αυτά είναι τα εξής: Docker, Prometheus, Node Exporter, Thanos, Grafana, Minio, Prometheus Anomaly Detector.

Στο Κεφάλαιο 4 γίνεται η διεξοδική ανάλυση της διαδικασίας που ακολουθήθηκε για την δημιουργία του συστήματος παρακολούθησης. Παρουσιάζεται ο τρόπος με τον οποίο έγιναν deploy τα διαφορετικά components του συστήματος και το πώς επικοινωνούν μεταξύ τους.

Στο Κεφάλαιο 5 παρουσιάζονται οι μετρικές οι οποίες επιλέχθηκαν και θεωρούνται σημαντικές για την παρακολούθηση ενός εικονικού μηχανήματος. Παρουσιάζονται τα διαγράμματα που σχεδιάστηκαν στο εργαλείο Grafana καθώς και το πώς αναγνωρίζονται οι “ανωμαλίες” σε κάποιο metric. Τέλος ελέγχεται και αξιολογείται η συνεχής και η μακροχρόνια διάθεση των δεδομένων που συλλέλονται.

Το Κεφάλαιο 6 είναι ο επίλογος της εργασίας. Στο κεφάλαιο αυτό συνοψίζονται τα συμπεράσματα της εργασίας και παρουσιάζονται πιθανές μελλοντικές επεκτάσεις μετά την ολοκλήρωσή της.

Στο τέλος της διπλωματικής εργασίας παρατίθεται η βιβλιογραφία που χρησιμοποιήθηκε κατά την συγγραφή της και το παράρτημα.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

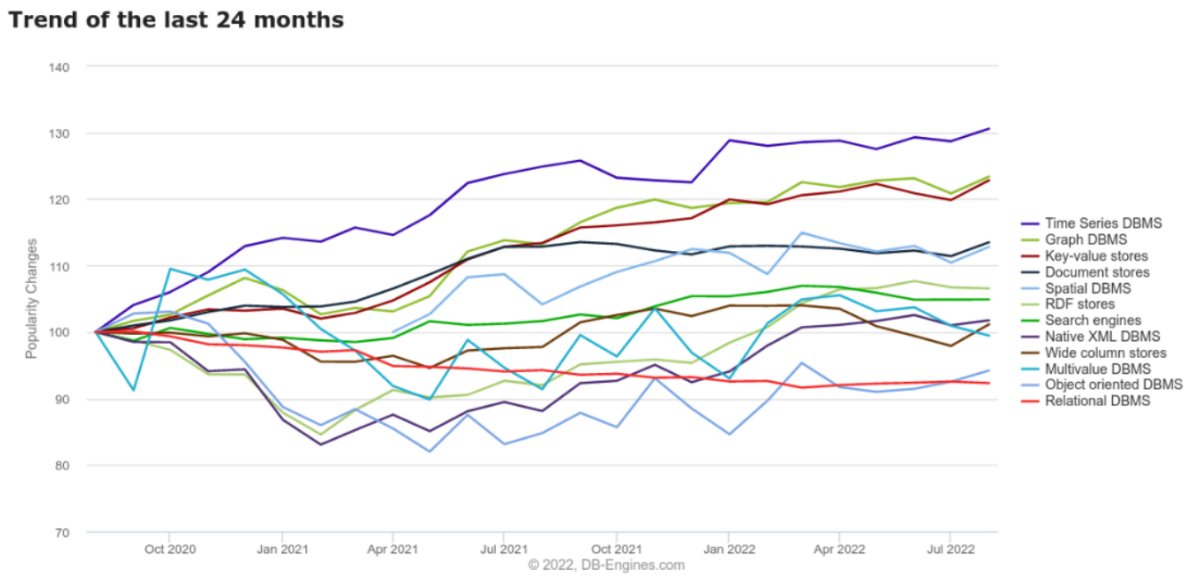
2.1 Χρονοσειρές

Τα δεδομένα χρονοσειρών (time series data) είναι μια μορφή δεδομένων που απαρτίζονται από μία αλληλουχία μετρήσεων (τιμών) στη διάρκεια του χρόνου. Για παράδειγμα ως δεδομένα χρονοσειρών μπορούν να θεωρηθούν οι καταγραφές της θερμοκρασίας ανά μία ώρα. Με τον ίδιο τρόπο θεωρούνται δεδομένα χρονοσειρών είναι και οι μετρήσεις που λαμβάνονται από ένα εικονικό μηχάνημα για την CPU, την RAM, τον ελεύθερο χώρο στο δίσκο καθώς και την δικτυακή κίνηση, τα οποία θα μελετηθούν και στην παρούσα διπλωματική.

Μια χρονοσειρά, δομικά, αποτελείται από μια σειρά αριθμητικών τιμών (value) όπου η κάθε μια συνοδεύεται από μια χρονική τιμή (timestamp). Ορίζεται με ένα όνομα ώστε να μπορεί να καλείται εύκολα όταν χρειαστεί και ένα σύνολο ετικετών (tags) που βοηθούν στο φιλτράρισμα των δεδομένων.

Οι χρονοσειρές σχετίζονται άρρηκτα με την παρακολούθηση συστημάτων (monitoring) καθώς η μορφή των συγκεκριμένων δεδομένων βοηθάει, αρχικά, στο να καταλάβει κάποιος το παρελθόν, αναλύοντας την κατάσταση του συστήματος οποιαδήποτε στιγμή. Μέσω των δεδομένων αυτών μπορεί να διαπιστωθεί κάποια στιγμή που το σύστημα ήταν εκτός λειτουργίας ή υπολειτουργούσε. Επιπλέον, ακόμα και σε πραγματικό χρόνο μπορεί να προληφθεί κάποιο πιθανό πρόβλημα της υποδομής καθώς οι χρονοσειρές θα αλλάξουν μορφή. Τέλος, βοηθούν στο πρόβλεψη του μέλλοντος αφού τα δεδομένα που συλλέγονται μπορούν να παρουσιάσουν κάποιο επαναλαμβανόμενο μοτίβο (trend) και έτσι μπορούν να συσχετιστούν με κάποιο άλλο γεγονός που μπορεί να συνέβαινε παράλληλα.

Παρά την μεγάλη χρηστικότητα τους αλλά και την αυξημένη ορατότητα που προσφέρουν στην παρακολούθηση μιας υποδομής, ένα από τα σημαντικότερα προβλήματα είναι η αποθήκευση των δεδομένων αυτών. Ο λόγος για τον οποίο συμβαίνει αυτό είναι ότι τα δεδομένα χρονοσειρών συσσωρεύονται με ταχείς ρυθμούς. Οι σχεσιακές βάσεις δεδομένων, δεν μπορούσαν να καλύψουν την αποθήκευση και την ανάκληση τέτοιων δεδομένων με αποτέλεσμα την δημιουργία βάσεων δεδομένων χρονοσειρών. Οι βάσεις δεδομένων χρονοσειρών έλυσαν πολλά από τα προβλήματα με την χρήση λειτουργιών και συναρτήσεων για την ανάκληση των δεδομένων συγκεντρωτικά (aggregated data), κυρίως όταν πρόκειται για δεδομένα του μακρινού παρελθόντος.



Εικόνα 2.1: Τάσεις σε σύγχρονες Βάσεις Δεδομένων

Για παράδειγμα, αν συλλέγονται δεδομένα για την CPU ενός εικονικού μηχανήματος ανά 10 δευτερόλεπτα τότε σε 1 λεπτο θα έχουν καταγραφεί 6 μετρήσεις. Η ίδια μέτρηση σε βάθος ενός μήνα είναι περίπου 260.000 μετρήσεις. Στο συγκεκριμένο παράδειγμα αν υπάρχει η ανάγκη για ανάκληση δεδομένων ενός χρόνου δεν είναι εύκολο να ανακληθούν όλες οι μετρήσεις μία προς μία. Στην περίπτωση αυτή τα δεδομένα ομαδοποιούνται με την χρήση συναρτήσεων, όπως η μέση τιμή. Τέλος, οι βάσεις δεδομένων χρονοσειρών προσφέρουν την δυνατότητα όχι μόνο να ανακαλούνται τα δεδομένα ομαδοποιημένα αλλά και να συμπιέζονται παλιά δεδομένα ώστε να γίνεται πιο εύκολη η αποθήκευση τους.

2.2 Μηχανική Μάθηση

Η Μηχανική Μάθηση, που είναι μια υποκατηγορία της επιστήμης της Τεχνητής Νοημοσύνης, επικεντρώνεται στην μελέτη και δημιουργία αλγορίθμων και στατιστικων μοντέλων, που χρησιμοποιούνται από υπολογιστικά συστήματα, ικανών να μαθαίνουν, με αυτόματο τρόπο, μοτίβα ώστε να κάνουν προβλέψεις και να παίρνουν αποφάσεις χωρίς να έχουν προγραμματιστεί για αυτό. Η διαδικασία δημιουργίας του μαθηματικού/στατιστικού μοντέλου ονομάζεται “εκπαίδευση” (training) του συστήματος και το δείγμα των δεδομένων πάνω στα οποία γίνεται η διαδικασία αυτή ονομάζονται “δεδομένα εκπαίδευσης” (training data).

Ο κλάδος της Μηχανικής Μάθησης έχει γίνει ευρέως γνωστός τα τελευταία χρόνια εξαιτίας της ικανότητας που έχει να αντιμετωπίζει περίπλοκα προβλήματα, να προσφέρει χρήσιμη πληροφορία μέσα από μεγάλης κλίμακας δεδομένα που μπορεί να αναλύσει και τέλος να κάνει στοχευμένες προβλέψεις μέσω αυτών για διάφορους τομείς της επιστήμης. Έχει εφαρμογή σε επιστήμες όπως η ιατρική καθώς μπορεί να βοηθήσει στην διάγνωση ασθενειών, σε στοχευμένες συστάσεις θεραπείας ανάλογα με το κάθε περιστατικό, στον κλάδο της οικονομίας και των επιχειρήσεων μέσω τον προβλέψεων και την βοήθεια στην

παίρνονται πιο σωστές και αποτελεσματικές αποφάσεις, στον κλάδο των μεταφορών με την χρήση του “αυτόματου πιλότου” αλλά και με ανάλυση και πρόβλεψη της κυκλοφορίας.

Υπάρχουν 3 υποκατηγορίες της Μηχανικής Μάθησης βάση του τρόπου με τον οποίο γίνεται η εκπαίδευση του συστήματος.

1. Επιτηρούμενη μάθηση (**Supervised Learning**): Το σύστημα δέχεται τα δεδομένα εκπαίδευσης συνοδευόμενα με μία τιμή (label) που θεωρείται σωστή ή αναμενόμενη έτσι ώστε να μπορεί στη συνέχεια κάθε νέα δεδομένο που δέχεται να μπορεί να το κατατάσει αντίστοιχα.
2. Μη επιτηρούμενη μάθηση (**Unsupervised Learning**): Το σύστημα δέχεται τα δεδομένα χωρίς κάποια άλλη τιμή (σωστή η αναμενόμενη όπως στο προηγούμενο παράδειγμα). Ο λόγος είναι για να ανακαλυφθούν πιθανώς κρυμμένα μοτίβα.
3. Ενισχυτική μάθηση (**Reinforcement Learning**): Για την εκπαίδευση του συστήματος εδώ ακολουθείται μία διαφορετική προσέγγιση που περιέχει την αλληλεπίδραση με ένα δυναμικό περιβάλλον για να επιτευχθεί ένας συγκεκριμένος σκοπός. (π.χ οδήγηση ενός οχήματος.

Οι πιο συνήθεις εφαρμογές της Μηχανικής Μάθησης είναι η *ταξινόμηση (classification)* και η *παλινδρόμηση (regression)*, που χρησιμοποιούν την μέθοδο της επιτηρούμενης μάθησης, η *ομαδοποίηση (clustering)* , που χρησιμοποιεί την μέθοδο της μη επιτηρούμενης μάθησης, και η *ανίχνευση ανωμαλιών (anomaly detection)*, για την οποία υπάρχουν διάφορα μοντέλα που ακολουθούνται όπως η επιτηρούμενη μάθηση, η ημι-επιτηρούμενη μάθηση και η μη επιτηρούμενη μάθηση.

2.3 Ανίχνευση Ανωμαλιών

2.3.1 Ορισμός και εφαρμογές

Η Ανίχνευση Ανωμαλιών (Anomaly Detection), είναι κλάδος της μηχανικής μάθησης και χρησιμοποιείται από τα υπολογιστικά συστήματα για να την παρατήρηση και τον εντοπισμό ιδιόμορφων χαρακτηριστικών στα δεδομένα, που τα κάνουν να ξεχωρίζουν από την πλειοψηφία αυτών. Η ανίχνευση τέτοιων ανωμαλιών βοηθάει στον προσδιορισμό μη αναμενόμενων συμπεριφορών και προβλημάτων. Οι εφαρμογές της ανίχνευσης ανωμαλιών είναι πολλές σε ποικίλους κλάδους και αναφέρονται στη συνέχεια κάποιες από αυτές.

1. Ανίχνευση απάτης (**Fraud Detection**): Ανίχνευση, κατά κύριο λόγο, σε μη φυσιολογικη συμπεριφορά σε διάφορου είδους συναλλαγές (π.χ πληρωμές με κάρτα).
2. Ανίχνευση Εισβολών (**Intrusion Detection**): Ανίχνευση ανωμαλιών στην δικτυακή κίνηση μιας υποδομής που μπορεί να υποδεικνύει κάποιο περιστατικό ρήξης της ασφάλειας της υποδομής.
3. Παρακολούθηση Συστημάτων (**System Monitoring**): Ανίχνευση ανωμαλιών στα συστήματα, στο δίκτυο αλλά και στις εφαρμογές μια υποδομής προκειμένου να

γίνει πρόληψη για οποιοδήποτε πρόβλημα μπορεί να προκληθεί στην εύρυθμη λειτουργία της.

4. **IoT** στις Βιομηχανίες: Ανίχνευση ανωμαλιών στο δεδομένα από αισθητήρες και από τα ίδια τα μηχανήματα μιας βιομηχανικής μονάδας με σκοπό την σωστή λειτουργία της παραγωγικής διαδικασίας.

Σε όλες τις παραπάνω περιπτώσεις αλλά και στα πλαίσια της παρούσας διπλωματικής που επικεντρώνεται στην παρακολούθηση των χαρακτηριστικών εικονικών μηχανημάτων, για να επιτευχθεί η ανίχνευση ανωμαλιών πραγματοποιείται μια ανάλυση στα δεδομένα που εκπαιδεύουν το σύστημα μας και δημιουργείται ένα μοντέλο που θεωρείται η φυσιολογική λειτουργία του συστήματος ή της εφαρμογής που παρακολουθείται. Οποιαδήποτε απόκλιση από τα πλαίσια του μοντέλου φυσιολογικής λειτουργίας θεωρείται ως ανωμαλία.

2.3.2 Prophet

Το Prophet είναι ένα μοντέλο ανοιχτού κώδικα που σχεδιάστηκε από το Facebook (Meta) και χρησιμοποιείται για την ανάλυση δεδομένων χρονοσειρών και πρόβλεψη ανωμαλιών στα δεδομένα αυτά. Το σημαντικότερο πλεονέκτημα που προσφέρει στην ανάλυση και στις προβλέψεις που κάνει είναι ότι μπορεί να χειριστεί ελλιπή δεδομένα, μεγάλες αλλαγές στις τιμές των δεδομένων και γενικά ακραίες τιμές που μπορεί να παρουσιάσουν αυτά ενώ παράλληλα μπορεί να τροφοδοτηθεί και με άλλες εξωγενείς μεταβλητές πριν γίνει η πρόβλεψη. Μπορεί να χειριστεί, ακόμα, δεδομένα που λαμβάνονται με μεγαλύτερη χρονική διαφορά μεταξύ της λήψης των δειγμάτων όπως για παράδειγμα μηνιαία ή ετήσια. Επιπλέον, το Prophet μπορεί να λάβει υπόψη του στις προβλέψεις του πολύ σημαντικά χαρακτηριστικά των δεδομένων όπως:

1. Τασεις (**Trends**): Εφήμερες αλλαγές των δεδομένων ανά διαστήματα βάση και άλλων εξωγενών παραγόντων.
2. Εποχικότητα (**Seasonality**): Προσαρμόζεται στα μοτίβα που παρουσιάζουν τα δεδομένα ημερήσια ή εβδομαδιαία ακόμα και ετήσια και τα συμπεριλαμβάνει στην διαδικασία των προβλέψεων.
3. Αυτόματο Σημείο Αλλαγής (**Changepoint**): Ανιχνεύει αυτόματα ολοκληρωτικές αλλαγές στο μοτίβο που ακολουθούν τα δεδομένα και προσαρμόζεται στην νέα φυσιολογική κατάσταση.

Το Prophet είναι συμβατό με τις γλώσσες προγραμματισμού Python και R, που είναι ευρέως γνωστές τα τελευταία χρόνια για την χρήση τους στην ανάλυση δεδομένων ενώ παράλληλα είναι αρκετά εύκολο στην χρήση του.

2.3.3 Μετασχηματισμός Fourier

Ο μετασχηματισμός Fourier είναι ένας μαθηματικός μετασχηματισμός που χρησιμοποιείται στην επεξεργασία σημάτων και βοηθάει στην ανάλυση της συχνότητας των σημάτων. Παρά το γεγονός ότι ο μετασχηματισμός Fourier δεν έχει σχεδιαστεί για την

ανίχνευση ανωμαλιών, μπορεί να χρησιμοποιηθεί σε μεθόδους και μοντέλα ανίχνευσης ανωμαλιών. Έχει εφαρμογή σε δεδομένα χρονοσειρών για την αναγνώρισης ασυνήθιστων μοτίβων. Στην ανίχνευση ανωμαλιών χρησιμοποιείται ως εξής:

1. Με την χρήση του μετασχηματισμού Fourier τα δεδομένα κατατάσσονται σε κατηγορίες με βάση το πόσο συχνά εμφανίζεται μια τιμή. Με τον τρόπο αυτό μπορεί να βρεθεί ποιες τιμές είναι κυρίαρχες στα δεδομένα αυτά.
2. Έπειτα με την ανάλυση των συχνοτήτων είναι εφικτή η εύρεση ακραίων τιμών που πιθανότητα σηματοδοτούν καποια είδος μη συμβατότητας αλλά και η γενική ταξινόμηση των νέων δεδομένων σε αποδεκτά και μη.
3. Αναδημιουργία: Από την στιγμή που έχουν ανακαλυφθεί πιθανές ανωμαλίες, με τον αντίστροφο μετασχηματισμό μπορούν να δημιουργηθούν ξανά τα δεδομένα χρονοσειρών απομακρύνοντας τα δεδομένα που έχουν χαρακτηριστεί ως μη αποδεκτά και δημιουργώντας έτσι ένα μοντέλο “φυσιολογικής συμπεριφοράς”.

Αξίζει να σημειωθεί πως ο μετασχηματισμός Fourier δεν είναι κατάλληλος για την ανάλυση δεδομένων που παρουσιάζουν trends και εποχικότητα και για αυτό συνιστάται να συνδυάζεται και με άλλους αλγορίθμους και μοντέλα ανίχνευσης ανωμαλιών.

Στην παρούσα διπλωματική χρησιμοποιήθηκε μια υλοποίηση ανοιχτού κώδικα, το οποίο θα αναλυθεί εκτενώς στην επόμενη ενότητα, που συνδυάζει το μοντέλου του Prophet και τον μετασχηματισμό Fourier ώστε να επιτύχει την καλύτερη δυνατή πρόβλεψη στα δεδομένα χρονοσειρών, τα οποία προσφέρονται από την παρακολούθηση των εικονικών μηχανημάτων μιας υποδομής.

Κεφάλαιο 3

Αρχιτεκτονική

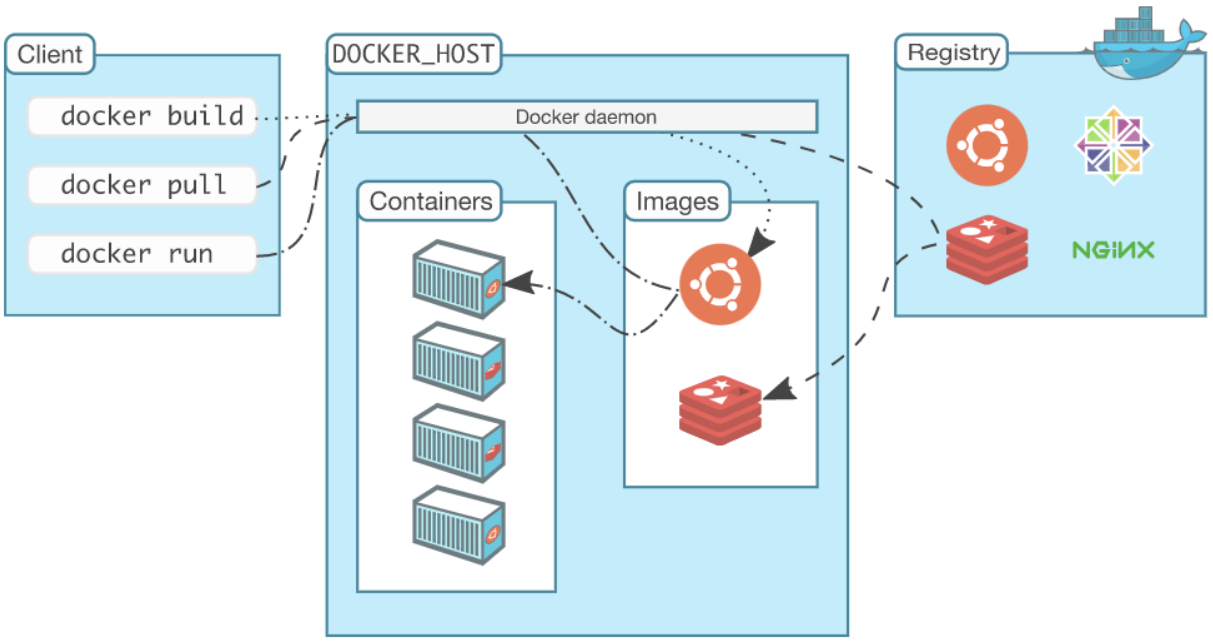
3.1 Docker

Το Docker είναι μια πλατφόρμα ανοικτού κώδικα, γραμμένη στη γλώσσα προγραμματισμού Go χρησιμοποιώντας παράλληλα αρκετά στοιχεία του πυρήνα του Linux, που εξυπηρετεί την ανάπτυξη και εκτέλεση εφαρμογών δίνοντας την δυνατότητα να διαχωρίζονται οι εφαρμογές από την υποδομή. Με την χρήση του Docker μια εφαρμογή πακετάρεται και τρέχει σε ένα απομονωμένο περιβάλλον που ονομάζεται **container**. Με τον τρόπο αυτό υπάρχει η δυνατότητα να τρέχουν ταυτόχρονα πολλαπλά containers σε ένα μηχάνημα έχοντας το καθένα ό,τι χρειάζεται για την εφαρμογή που εξυπηρετεί, ανεξάρτητα από τι είναι εγκατεστημένο στο μηχάνημα αυτό.
Παρακάτω παρουσιάζονται μερικά από τα πλεονεκτήματα της χρήση του Docker:

1. *Γρήγορη και συνεπής υλοποίηση εφαρμογών:* Ο κώδικας της εφαρμογής μπορεί να δοκιμαστεί σε ένα απομονωμένο περιβάλλον το οποίο αποτελείται και αυτό από containers. Στο περιβάλλον αυτό θα εντοπιστούν τυχόν προβλήματα της εφαρμογής και θα διορθωθούν, με σκοπό τελικά να δημιουργηθεί μία εικόνα (**image**) της εφαρμογής έτοιμη για να ενημερωθεί με αυτή το περιβάλλον της παραγωγής. Η διαδικασία αυτή είναι σημαντικά πιο γρήγορη με την χρήση των containers συγκριτικά με τη χρήση Virtual Machines (**VM**).
2. *Ευκολότερη κλιμάκωση των αναγκών μιας εφαρμογής:* Με την χρήση των containers μια εφαρμογή μπορεί να τρέξει σε έναν προσωπικό υπολογιστή, σε ένα φυσικό ή εικονικό μηχάνημα σε ένα Data Center, σε ένα cloud περιβάλλον ή σε ένα συνδυασμό όλων των παραπάνω. Επιπλέον τα container, λόγω της φορητότητας που προσφέρουν καθώς και το γεγονός ότι επιβαρύνουν λιγότερο το μηχάνημα το οποίο τα φιλοξενεί, μπορούν δυναμικά να διαχειριστούν το φόρτο και την κίνηση μιας εφαρμογής αναθέτοντας ή αποδεσμευοντας πόρους με βάση τις ανάγκες της εφαρμογής κατά την πάροδο του χρόνου.

Το Docker χρησιμοποιεί την αρχιτεκτονική Πελάτη-Εξυπηρετητή (Client-Server). Ο Docker client επικοινωνεί με τον Docker daemon, ο οποίος είναι υπεύθυνος για όλη την δημιουργία και την υλοποίηση των containers. Οι Docker client και Docker daemon μπορούν να φιλοξενούνται στο ίδιο σύστημα ή σε διαφορετικό και να επικοινωνούν απομακρυσμένα χρησιμοποιώντας ένα API. Στην παρούσα διπλωματική χρησιμοποιήθηκε το Docker

Compose, το οποίο αποτελεί ένα είδος Docker Client, με το οποίο γίνεται πιο εύκολη η διαχείριση πολλαπλών containers.



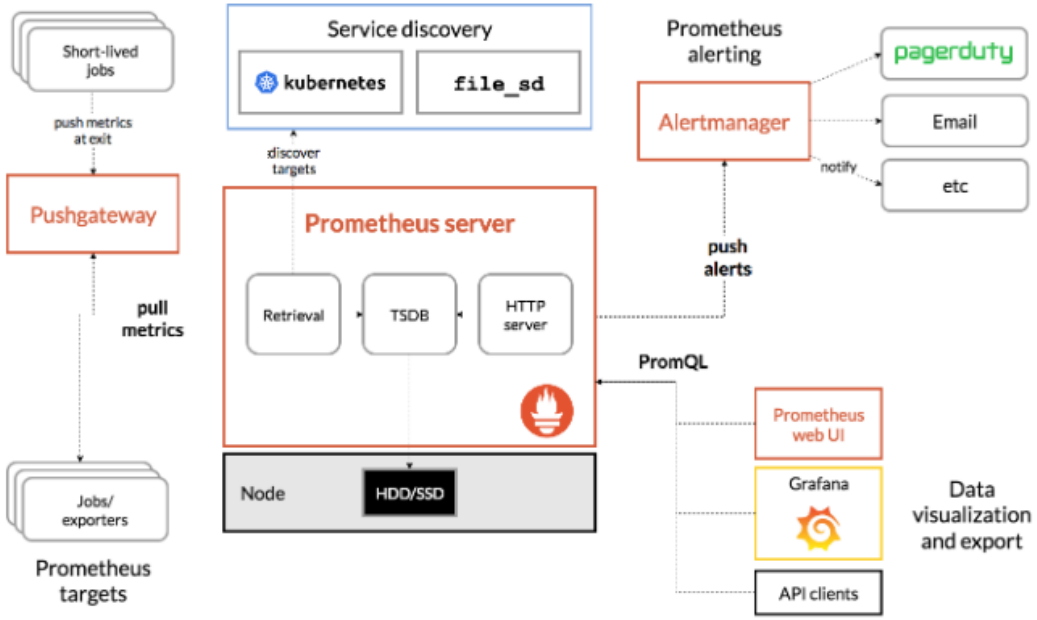
Εικόνα 3.1: Αρχιτεκτονική του Docker

Ο Docker daemon (*dockerd*) δέχεται τα Docker API αιτήματα και βάση αυτών διαχειρίζεται τα containers, τα images και το δίκτυο με το οποίο επικοινωνούν τα containers. Μέσω του Docker client (*docker*) μπορούν οι χρήστες να αλληλεπιδράσουν με το Docker daemon και να τρέξουν τις απαραίτητες εντολές για να δημιουργηθούν τα containers και να τρέξει μια εφαρμογή. Ένας χρήστης μπορεί να βρει έτοιμα images σε ενα Docker registry. Στην παρούσα διπλωματική χρησιμοποιήθηκε το Docker Hub, το οποίο είναι ένα Docker registry διαθέσιμο για όλους, καθώς αποτελεί το προκαθορισμένο registry του Docker. Ένα image είναι αρχείο που περιέχει ένα σύνολο από οδηγίες για την δημιουργία ενός container. Συνήθως ένα image μιας εφαρμογής βασίζεται σε κάποιο άλλο image που έχει υποστεί κάποια παραμετροποίηση. Ένα container είναι μια εκτελέσιμη μορφή ενός image. Ένας χρήστης μπορεί να δημιουργήσει, να ξεκινήσει, να σταματήσει ή να διαγράψει ένα container, να το συνδέσει σε ένα ή περισσότερα δίκτυα, να του παραχωρήσει επιπλέον χώρο χρησιμοποιώντας το Docker API. Τέλος μπορεί να δημιουργήσει ένα νέο image βασισμένο σε αυτό το container και τις διεργασίες που έχουν γίνει πάνω του.

3.2 Prometheus

Το Prometheus είναι ένα εργαλείο εποπτείας και ειδοποίησης ανοικτού κώδικα, γραμμένο σε γλώσσα προγραμματισμού Go. Αρχικά σχεδιάστηκε από την SoundCloud το 2012 αλλά σήμερα αποτελεί ένα πλήρως αυτόνομο λογισμικό το οποίο συντηρείται από κάθε εταιρεία/χρήστη ξεχωριστά, έχοντας μια τεράστια κοινότητα προγραμματιστών σε όλο τον κόσμο.

Στην εικόνα 3.2 παρουσιάζεται η αρχιτεκτονική του Prometheus όπως αυτό παρατίθεται στην επίσημη ιστοσελίδα.



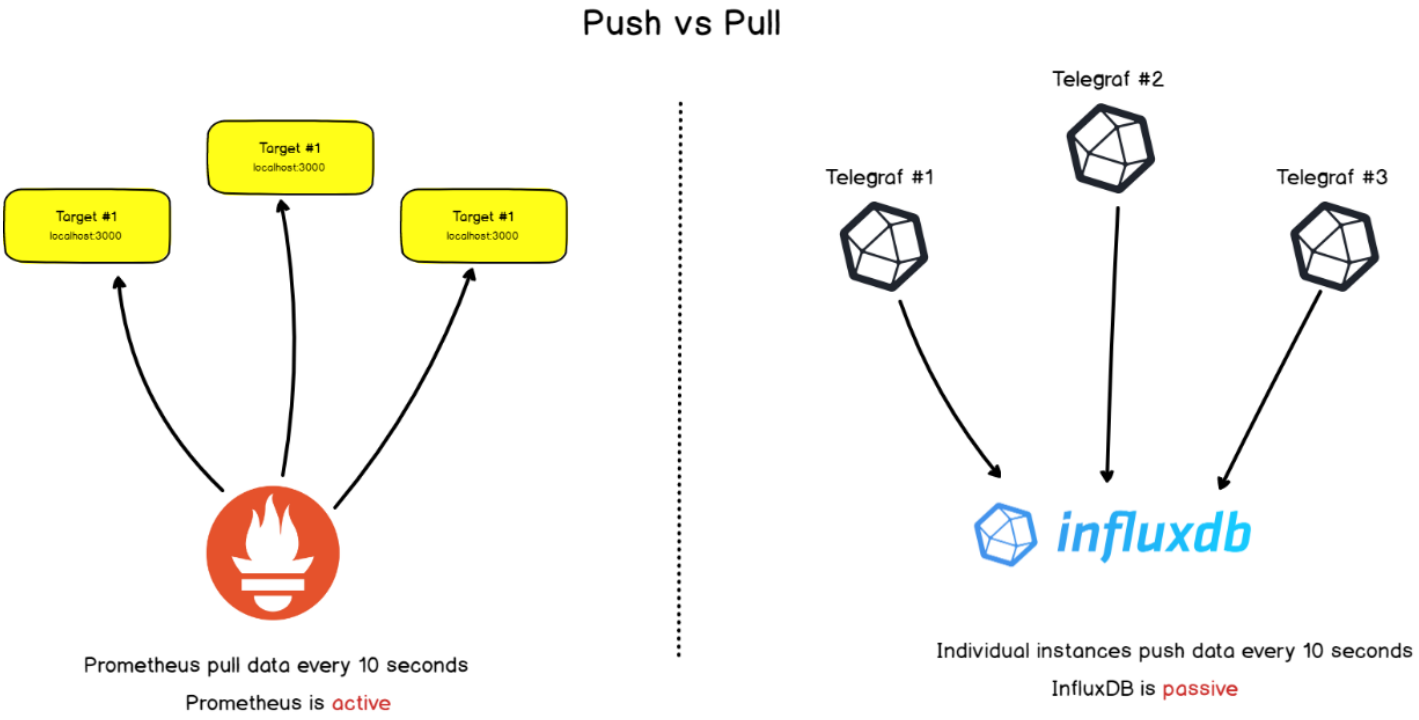
Εικόνα 3.2: Αρχιτεκτονική Prometheus

Τα κύρια δομικά στοιχεία του Prometheus αναλύονται στην συνέχεια.

1. ***Prometheus Server****:* Είναι το βασικότερο κομμάτι του οικοσυστήματος καθώς εκεί συλλέγονται τα metrics (δεδομένα χρονοσειρών) και αποθηκεύονται τοπικά. Συλλέγει τα δεδομένα με την μέθοδο του scraping, εκτελεί δηλαδή HTTP calls προς τα τελικά σημεία τα οποία εποπτεύει ανά συγκεκριμένα χρονικά διαστήματα.
2. ***Push gateway:*** Είναι μια ενδιάμεση υπηρεσία η οποία συλλέγει metrics από εφήμερες εργασίες καθώς αυτές δεν είναι εφικτό να γίνονται scrape για πάντα αφού δεν είναι μόνιμα διαθέσιμες.
3. ***Exporters:*** Είναι βιβλιοθήκες που βοηθούν στην εξαγωγή δεδομένων από συστήματα και εφαρμογές σε μορφή metric ώστε να είναι συμβατά με τον Prometheus. (Node Exporter, PostgreSQL Exporter κ.λ.π)

4. ***Alertmanager:*** Είναι υπηρεσία που προσφέρει το Prometheus για να διαχειρίζεται ο χρήστης alerts σε περίπτωση που κάποιο από τα metric που συλλέγει βρεθεί σε κάποιο επίπεδο τιμών που δεν είναι επιθυμητό. Δίνει επιπλέον την δυνατότητα να στέλνονται οι ειδοποιήσεις σε εξωτερικά μέσα όπως email, SMS, Slack κ.λ.π.
5. ***Data Dashboard:*** Το Prometheus παρέχει μία διεπαφή χρήστη με γραφήματα σχετικά με τα metric που συλλέγει. Αναφέρεται σαν component παρόλου που συνήθως χρησιμοποιείται από τους χρήστες δευτερεύον εργαλείο για την παρουσίαση των γραφημάτων, το οποίο θα αναλυθεί στην συνέχεια και είναι το Grafana.

Το Prometheus είναι ένα pull-based σύστημα παρακολούθησης. Μπορεί να συλλέξει δεδομένα (metrics) όπως αναφέρθηκε προηγουμένως με διάφορους τρόπους , είτε να κάνει η ίδια εφαρμογή ή το σύστημα που επρόκειτο να παρακολουθείται τα metrics διαθέσιμα σε μορφή συμβατή με το Prometheus με την βοήθεια βιβλιοθηκών είτε με τους ήδη διαθέσιμους Exporters που δένουν στο εκάστοτε σύστημα είτε μέσω του push gateway. Με εξαίρεση τον τελευταίο τρόπο η pull based μέθοδος χρησιμοποιείται από το Prometheus και είναι αυτή που το ξεχωρίζει από άλλα συστήματα που συλλέγουν και αποθηκεύουν δεδομένα χρονοσειρών.



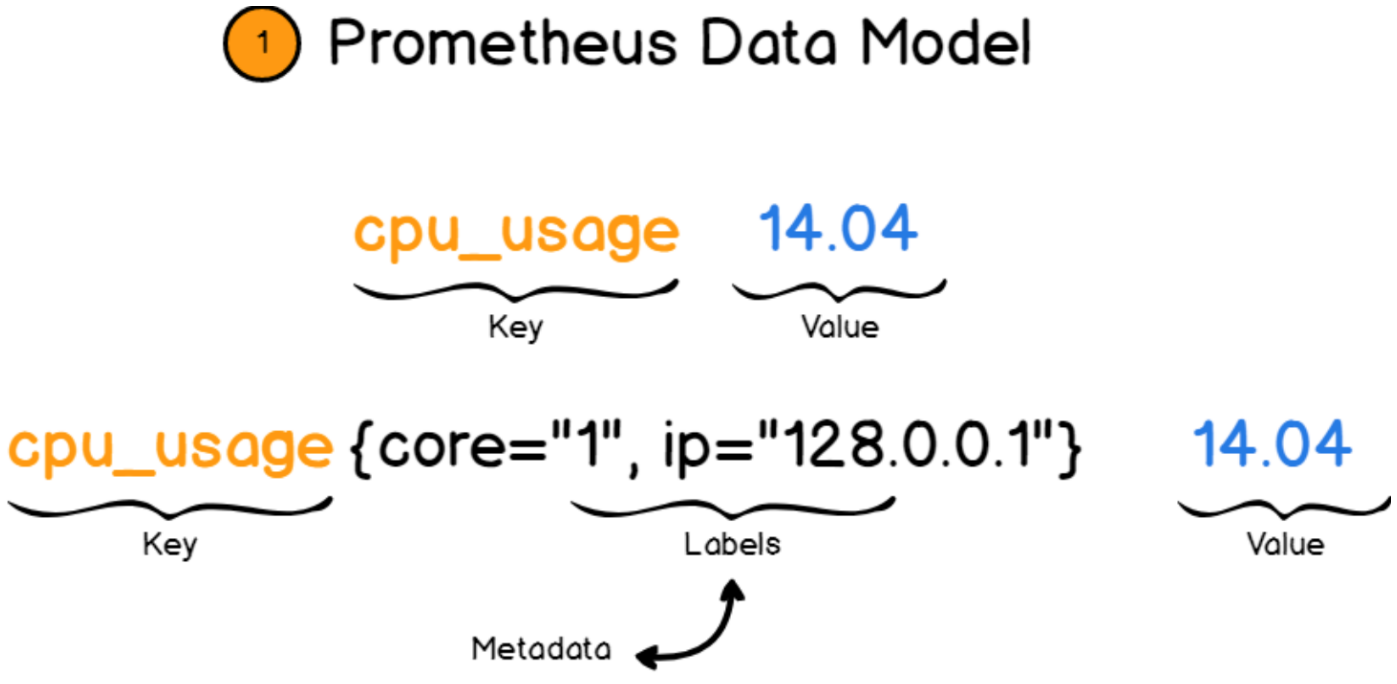
Εικόνα 3.3: Pull vs Push method Example.

Τα κυριότερα πλεονεκτήματα της αρχιτεκτονικής αυτής και ο λόγος που το Prometheus χρησιμοποιείται από πολλούς για την παρακολούθηση των υπολογιστικών συστημάτων είναι δύο.

Το πρώτο είναι ότι μπορεί ο χρήστης του συστήματος να έχει μια κεντρικοποιημένη διαχείριση, το Prometheus ξεκινάει την επικοινωνία με τα συστήματα "στόχους" (targets), με αυτόν τον τρόπο ο χρήστης έχει μόνο να διαχειριστεί την μεριά του Prometheus Server και όχι κάθε target ξεχωριστά. Στην μεριά του Prometheus αποφασίζεται ποιον target θα παρακολουθεί και πόσο συχνά και έτσι προστατεύεται η υλοποίηση από την πιθανότητα να σταλούν υπερβολικά δεδομένα στον server με αποτέλεσμα να τον υπερφορτώσουν, κάτι που θα μπορούσε να συμβεί σε ένα push based σύστημα. Το δεύτερο πλεονέκτημα είναι ότι το Prometheus αποθηκεύει συνολικά (aggregated) metrics για το εκάστοτε σύστημα κάτι που υποστηρίζεται από την pull based υλοποίηση. Δεν είναι ένα event-based σύστημα που θα λάβει μία ειδοποίηση σε περίπτωση σφάλματος σε κάποιο σημείο της υποδομής αλλά θα λάβει την συνολική εικόνα ότι σε ένα χρονικό διάστημα η υποδομή σε κάποιο σημείο δεν ανταποκρινόταν με τον τρόπο που θα έπρεπε.

Το Prometheus καθότι αποθηκεύει τα δεδομένα που συλλέγει προσφέρει και την δική του ενσωματωμένη γλώσσα για να μπορεί ο χρήστης να έχει πρόσβαση σε δεδομένα αυτά, την **PromQL**. Το Prometheus αποθηκεύει τα δεδομένα με την μορφή **key-value** ζευγαριών. Το key (κλειδί) ονομάζεται metric και περιγράφει με λόγια το τι είναι αριθμός που έχει το value (τιμή). Στην περίπτωση που ο χρήστης θέλει περισσότερη πληροφορία να αποθηκεύεται για μια τιμή (value) τότε μπορεί να χρησιμοποιήσει labels. Με την PromQL ο χρήστης μπορεί να διαχειριστεί τα δεδομένα του Prometheus έχοντας στην διάθεση του 2 είδη διανυσμάτων:

1. Instant Vectors: Οι πιο πρόσφατες τιμές των μετρήσεων.
2. Time range vectors: Η εξέλιξη μιας μέτρησης στο πέρασμα του χρόνου.



Εικόνα 3.4: Prometheus Data Model

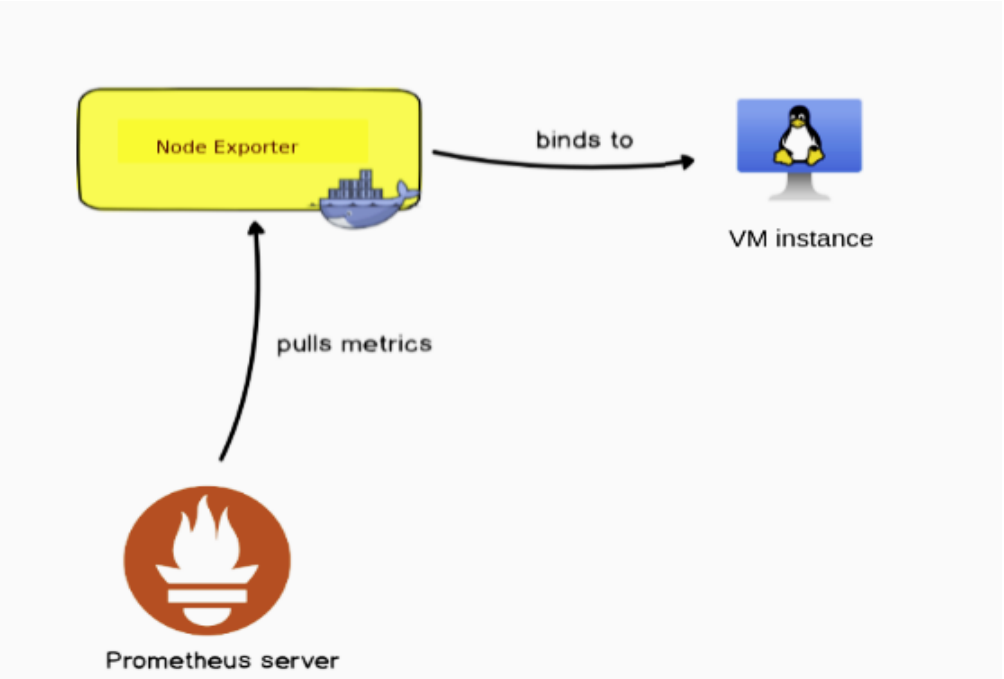
3.3 Node Exporter

Ο Node Exporter είναι ένας από τους διαθέσιμους και συμβατούς με το Prometheus, γραμμένος σε Go, εξαγωγείς δεδομένων για το υλικό (hardware) και το λογισμικό (software) ενός UNIX συστήματος (π.χ Linux). Τα δεδομένα (metrics) που συλλέγει είναι διαθέσιμα στην πόρτα 9100 από την στιγμή που εγκατασταθεί ο Node Exporter σε ένα μηχάνημα. Στη συνέχεια παρουσιάζονται όλες οι κατηγορίες δεδομένων που συλλέγονται με την εγκατάσταση του στο μηχάνημα που το φιλοξενεί. Στην παρούσα διπλωματική δίνεται έμφαση κυρίως στα metrics που συλλέγονται για τον επεξεργαστή (CPU), για την μνήμη RAM, για τον δίσκο και για την δικτυακή κίνηση από και προς το συγκεκριμένο μηχάνημα.

Name	Description	Name	Description
arp	Exposes ARP statistics from /proc/net/arp.	netstat	Exposes network statistics from /proc/net/netstat. This is the same information as netstat -s.
bcache	Exposes bcache statistics from /sys/fs/bcache/.	nfs	Exposes NFS client statistics from /proc/net/rpc/nfs. This is the same information as nfsstat -c.
bonding	Exposes the number of configured and active slaves of Linux bonding interfaces.	nfsd	Exposes NFS kernel server statistics from /proc/net/rpc/nfsd. This is the same information as nfsstat -s.
btrfs	Exposes btrfs statistics	nvme	Exposes NVMe info from /sys/class/nvme/
boottime	Exposes system boot time derived from the kern.boottime sysctl.	os	Expose OS release info from /etc/os-release or /usr/lib/os-release
conntrack	Shows conntrack statistics (does nothing if no /proc/sys/net/netfilter/ present).	powersupplyclass	Exposes Power Supply statistics from /sys/class/power_supply
cpu	Exposes CPU statistics	pressure	Exposes pressure stall statistics from /proc/pressure/.
cpufreq	Exposes CPU frequency statistics	rapl	Exposes various statistics from /sys/class/powercap.
diskstats	Exposes disk I/O statistics.	schedstat	Exposes task scheduler statistics from /proc/schedstat.
dmi	Expose Desktop Management Interface (DMI) info from /sys/class/dmi/id/	selinux	Exposes SELinux statistics.
edac	Exposes error detection and correction statistics.	sockstat	Exposes various statistics from /proc/net/sockstat.
entropy	Exposes available entropy.	softnet	Exposes statistics from /proc/net/softnet_stat.
exec	Exposes execution statistics.	stat	Exposes various statistics from /proc/stat. This includes boot time, forks and interrupts.
fibrechannel	Exposes fibre channel information and statistics from /sys/class/fc_host/.	tapestats	Exposes statistics from /sys/class/scsi_tape.
filefd	Exposes file descriptor statistics from /proc/sys/fs/file-nr.	textfile	Exposes statistics read from local disk. The --collector.textfile.directory flag must be set.

filesystem	Exposes filesystem statistics, such as disk space used.	thermal	Exposes thermal statistics like pmset -g therm.
hwmon	Expose hardware monitoring and sensor data from /sys/class/hwmon/.	thermal_zone	Exposes thermal zone & cooling device statistics from /sys/class/thermal.
infiniband	Exposes network statistics specific to InfiniBand and Intel OmniPath configurations.	time	Exposes the current system time.
ipvs	Exposes IPVS status from /proc/net/ip_vs and stats from /proc/net/ip_vs_stats.	timex	Exposes selected adjtimex(2) system call stats.
loadavg	Exposes load average.	udp_queues	Exposes UDP total lengths of the rx_queue and tx_queue from /proc/net/udp and /proc/net/udp6.
mdadm	Exposes statistics about devices in /proc/mdstat (does nothing if no /proc/mdstat present).	uname	Exposes system information as provided by the uname system call.
meminfo	Exposes memory statistics.	vmstat	Exposes statistics from /proc/vmstat.
netclass	Exposes network interface info from /sys/class/net/	xfs	Exposes XFS runtime statistics.
netdev	Exposes network interface statistics such as bytes transferred.	zfs	Exposes ZFS performance statistics.

Στην συνέχεια παρουσιάζεται η αρχιτεκτονική με την οποία έχει υλοποιηθεί η συλλογή metrics απο το εικονικό μηχάνημα (VM) στον Prometheus με την αξιοποίηση του Node Exporter.

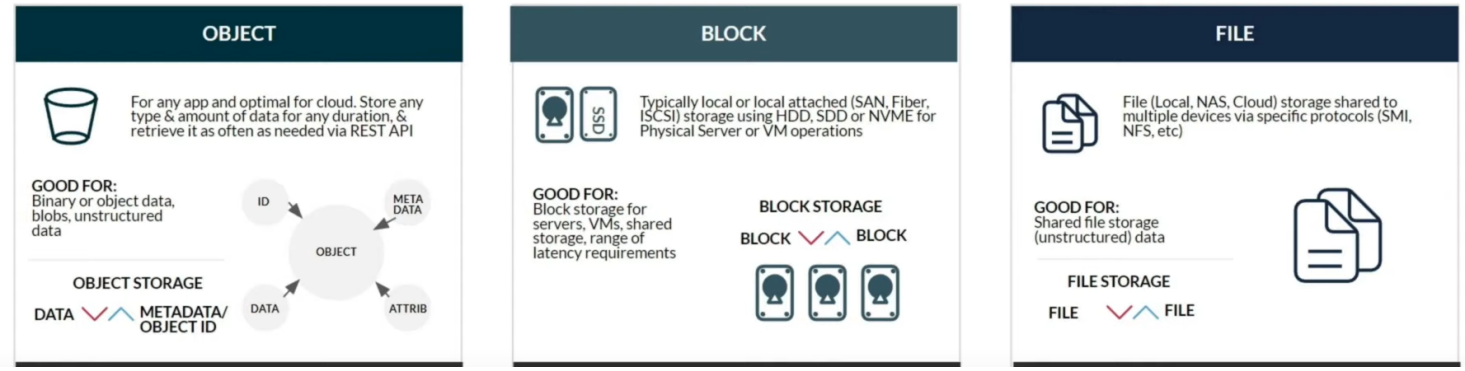


Εικόνα 3.5: Αρχιτεκτονική Prometheus-Node Exporter

3.4 Minio

Για την αποθήκευση (storage) δεδομένων μπορούν να χρησιμοποιηθούν οι παρακάτω αρχιτεκτονικές:

1. File Storage: Τα δεδομένα αποθηκεύονται όλα μαζί σε ένα αρχείο με μια συγκεκριμένη κατάληξη που προσδιορίζει την εφαρμογή που χρησιμοποιήθηκε για να παραχθούν τα συγκεκριμένα δεδομένα (.jpg, .txt). Είναι μια ιεραρχική δομή που αποτελείται από φακέλους.
2. Object storage: Τα δεδομένα χωρίζονται σε μικρότερες μονάδες και αποθηκεύονται όλα στο ίδιο επίπεδο χωρίς να υπάρχουν φάκελοι και υποκαταλογοι όπως σε μία file storage υποδομή. Επιπλέον τα objects έχουν δευτερεύοντα χαρακτηριστικά (metadata) που αποθηκεύονται μαζί τους. Είναι μια υλοποίηση που βοηθάει στην αποθήκευση δεδομένων χωρίς συγκεκριμένη μορφή.
3. Block Storage: Τα δεδομένα χωρίζονται σε block και αποθηκεύονται ξεχωριστά μαζί με ένα μοναδικό αναγνωριστικό. Το κάθε block μπορεί να αποθηκευτεί σε διαφορετικό περιβάλλον από κάποιο άλλο block των ίδιων δεδομένων. Όταν ο χρήστης ρωτήσει για τα δεδομένα αυτά το σύστημα συλλέγει όλα τα block τα συναρμολογεί και τα προσφέρει στον χρήστη.



Εικόνα 3.6: Συγκριση object / file / block storage

Το Minio είναι μια object storage λύση και σχεδιάστηκε για να ικανοποιήσει την ανάγκη για αποθήκευση δεδομένων που δεν έχουν συγκεκριμένη δομή και παράλληλα να μπορεί να υλοποιηθεί είτε σε τοπική υποδομή είτε σε μία cloud υποδομή είτε ακόμα και σε ένα υβριδικό μοντέλο. Η αρχιτεκτονική και ο σχεδιασμός του Minio προσφέρει την συνεχή διάθεση των δεδομένων (high availability), την ανοχή σε σφάλματα και προβλήματα στην υποδομή σε επίπεδο υλικού (hardware) καθώς χρησιμοποιεί erasure coding, το οποίο είναι μια μέθοδος για την διασφάλιση της ακεραιότητας των δεδομένων. Το Minio είναι ιδανικό για λύσεις που χρειάζεται τα δεδομένα να αποθηκεύονται γρήγορα και με ασφαλή τρόπο καθώς και να είναι προσβάσιμα εξίσου γρήγορα. Τα δεδομένα στην συγκεκριμένη υλοποίηση είναι διαθέσιμα και μπορούν να γίνουν διεργασίες πάνω σε αυτά με την χρήση

API και αυτό είναι ένας από τους λόγους που επιλέχθηκε απο το Thanos, το οποίο είναι το αντικείμενο μελέτης της επόμενης ενότητας.

3.5 Thanos

Το Thanos είναι ένα εργαλείο ανοιχτού κώδικα το οποίο σχεδιάστηκε για να εξυπηρετήσει την οριζόντια επέκταση (horizontal scaling) του Prometheus. Το Prometheus διαπιστώθηκε πως αντιμετώπιζε προβλήματα διαχείρισης μεγάλου αριθμού metrics, ειδικά σε κατανεμημένα συστήματα, καθώς παρείχε περιορισμένο χρόνο διατήρησης (retention) των metrics. Στην συνέχεια παρουσιάζονται τα πλεονεκτήματα της χρήσης του Thanos σε μία υλοποίηση monitoring μιας υποδομής:

1. Σφαιρική οπτική από όλους τους διαθέσιμους Prometheus Servers μέσα στην υποδομή, επιτρέποντας να γίνονται ΄ερωτήματα΄ σε παραπάνω από έναν Prometheus Server ταυτόχρονα.
2. Μακροχρόνια αποθήκευση των metrics, περισσότερο από 15 μέρες πού είναι το μέγιστο που διαθέτει ο Prometheus.
3. Συνεχής διαθεσιμότητα των metrics ακόμα και αν κάποιο component καταστεί μη διαθέσιμο, κάποια χρονική στιγμή. Με την χρήση πολλαπλών αντιγράφων (replicas), όταν το ένα δεν είναι διαθέσιμο το άλλο συνεχίζει να παρέχει την υπηρεσία χωρίς διακοπή.
4. Deduplication των δεδομένων, μειώνοντας τον χώρο που χρειάζεται για να αποθηκευτούν τα metrics διαγράφοντας διπλότυπες εγγραφές.

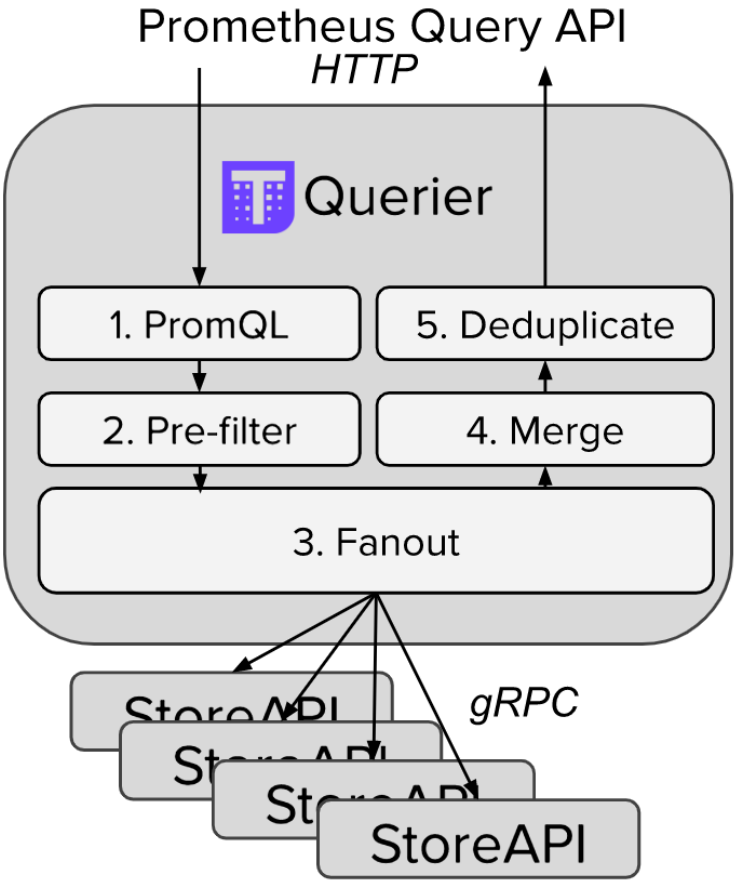
Στο σημείο αυτό αξίζει να σημειωθεί πως το Thanos χρησιμοποιεί σύστημα αποθήκευσης αντικειμένων (object storage). Στην παρούσα διπλωματική χρησιμοποιήθηκε το Minio, όπως αναφέρθηκε στην προηγούμενη ενότητα. Στην συνέχεια παρουσιάζονται τα δομικά στοιχεία του Thanos και ο ρόλος τους.

Sidecar

Το Thanos Sidecar ενοποιείται στους υπάρχοντες Prometheus Servers σαν μια υποστηρικτική διεργασία (sidecar process). Το Thanos Sidecar πρέπει να βρίσκεται στο ίδιο μηχάνημα ή γενικά στο ίδιο σημείο που είναι ο Prometheus Server. Ο ρόλος που εξυπηρετεί είναι να αποθηκεύει τα δεδομένα του Prometheus στο object storage σύστημα που χρησιμοποιείται στην υλοποίηση. Επιπλέον παρέχει, στα υπόλοιπα στοιχεία του Thanos, πρόσβαση στα metrics του Prometheus μέσω ενός gRPC (Remote Procedure Call) API. Το gRPC είναι ανοιχτού κώδικα λογισμικό το οποίο επιτρέπει την επικοινωνία πελάτη (client) και εξυπηρετητή (server) χρησιμοποιώντας protocol buffers, ένα είδος δυαδικής σειριοποίησης των δεδομένων που δεν στηρίζεται σε κάποια συγκεκριμένη γλώσσα προγραμματισμού. Είναι ένα πρωτόκολλο που είναι πολύ αποδοτικό και ταιριάζει σε αρχιτεκτονικές που χρησιμοποιούν μικρουπηρεσίες (microservices), όπως ακριβώς το Thanos που αποτελείται από διαφορετικά και ανεξάρτητα στοιχεία (components).

Querier/Query

Το Thanos Query χρησιμοποιείται για να προσφέρει γενικευμένη εικόνα όλων των Thanos Sidecar , επομένως και των Prometheus, από τα οποία απαρτίζεται η υλοποίηση. Με αυτόν τον τρόπο ο χρήστης έχει την δυνατότητα να εκτελεί PromQL ερωτήματα τα οποία απευθύνονται σε διαφορετικούς Prometheus Servers, και το Thanos Query αφότου συνδεθεί στα Thanos Sidecars αυτόματα ανιχνεύει με ποιους Prometheus Servers πρέπει να επικοινωνήσει για να φέρει τα αποτελέσματα του εκάστοτε ερωτήματος. Το Thanos Query εκτελεί επιπλέον ένα πανομοιότυπο API με Prometheus HTTP API με αποτέλεσμα να μπορεί να χρησιμοποιηθεί σαν πηγή δεδομένων για εργαλεία απεικόνισης όπως το Grafana. Ένα από τα πιο σημαντικά πρόσθετα που προσφέρει το Thanos Query είναι το data deduplication προσφέροντας την συνεχή διαθεσιμότητα του Prometheus. Έστω ότι έχουμε δύο Prometheus Servers που παρακολουθούν τα ίδια μηχανήματα/εφαρμογές στόχους. Με το Thanos Query μπορούμε να βλέπουμε το αποτέλεσμα ενός PromQL ερωτήματος χωρίς να έχουμε 'διπλά' δεδομένα αλλά παράλληλα να συνεχίσουμε να έχουμε δεδομένα ακόμα και αν κάποια από τα 2 Prometheus Servers καταστεί μη διαθέσιμο κάποια χρονική στιγμή.



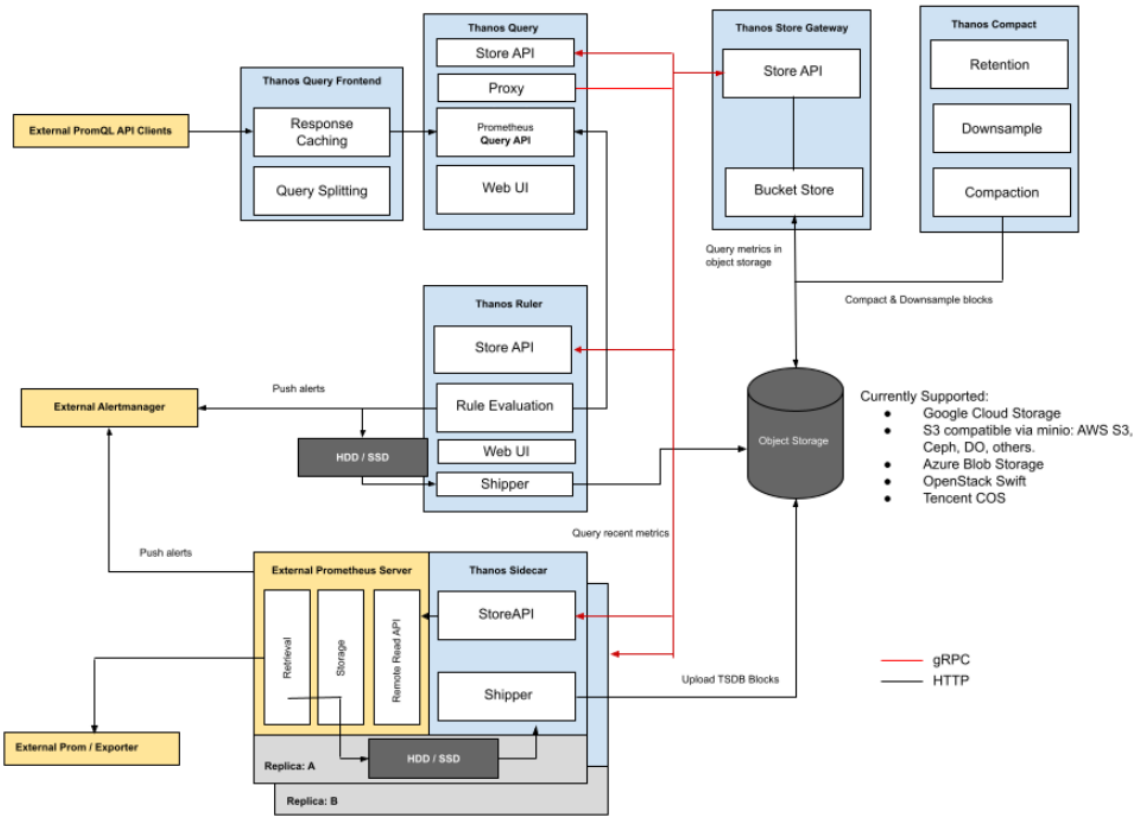
Εικόνα 3.7: Διαδρομή ενός PromQL ερωτήματος με χρήση Thanos Query

Store Gateway

Όπως ήδη αναφέρθηκε το Thanos Sidecar δημιουργεί αντίγραφα των δεδομένων στο σύστημα αποθήκευσης αντικειμένων που έχει επιλεχθεί, με αποτέλεσμα να μπορεί το Prometheus να παραμετροποιηθεί ώστε να κρατάει τοπικά για λιγότερο χρονικό διάστημα τα δεδομένα. Παρόλα αυτά υπάρχει η ανάγκη για πρόσβαση στα ιστορικά δεδομένα σε μεταγενέστερο χρόνο. Τον σκοπό αυτό εξυπηρετεί στο Store Gateway καθώς υλοποιεί το ίδιο gRPC API όπως το Sidecar το οποίο εντοπίζεται απο το Thanos Querier.

Compactor

Το Prometheus περιοδικά συμπιέζει τα πιο παλιά δεδομένα για να είναι πιο αποδοτικά και γρήγορα τα ερωτήματα που πραγματοποιούνται. Δεδομένου ότι το Sidecar μεταφέρει τα δεδομένα στο object storage το γρηγορότερο δυνατό υπάρχει η ανάγκη να γίνεται η διαδικασία της συμπίεσης (compaction) σε δεύτερο χρόνο. Το Thanos Compactor εξυπηρετεί αυτό ακριβώς τον σκοπό. Την ίδια στιγμή κάνει υποδειγματοληψία(downsampling) των δεδομενων για εξυπηρετούνται γρηγορότερα τα διάφορα ερωτήματα (queries). Αξίζει να σημειωθεί το γεγονός ότι Thanos Compactor είναι μοναδικό, δηλαδή δεν μπορεί να λειτουργούν 2 ή περισσότεροι compactor στα ίδια δεδομένα.



Εικόνα 3.7: Αρχιτεκτονική των microservices του Thanos

3.6 Grafana

Η Grafana είναι μια πλατφόρμα ανοιχτού κώδικα που χρησιμοποιείται για την παρακολούθηση συστημάτων καθώς παρέχει έναν ευέλικτο και εύκολο τρόπο οπτικοποίησης και ανάλυσης δεδομένων χρονοσειρών. Είναι μια ευρέως διαδεδομένη πλατφόρμα των τελευταίων χρόνων και χρησιμοποιείται από πολλούς οργανισμούς και επιχειρήσεις εξαιτίας της πληθώρας δυνατοτήτων που προσφέρει, της εύκολης στην χρήση διεπαφής χρήστη και του μεγάλου εύρους διασυνδέσεων με άλλες πλατφόρμες που έχει.

Η Grafana μπορεί να προσφέρει παρακολούθηση και παρατηρησιμότητα σε μεγάλα και περίπλοκα υπολογιστικά περιβάλλοντα αφού επιτρέπει στους χρήστες να συλλέξουν δεδομένα από πολλαπλές πηγές όπως μετρήσεις (metrics) ή δεδομένα από αρχεία καταγραφής (logs) και στην συνέχεια να τα οπτικοποιήσουν σε μια μορφή που θα δώσει πολύτιμη προφορία για την κατάσταση και την απόδοση ενός συστήματος. Με την Grafana, οι χρήστες μπορούν να δημιουργούν γραφήματα με βάση τις ανάγκες του κάθε οργανισμού, να χρησιμοποιούν την λειτουργία ειδοποιήσεων σε περίπτωση που εντοπιστεί κάποιο πρόβλημα στο σύστημα που παρακολουθείται και έτσι να γνωρίζουν σε κάθε χρονική στιγμή για την απόδοση των συστημάτων και των εφαρμογών.

Στην παρούσα διπλωματική χρησιμοποιήθηκε εξαιτίας των δυνατοτήτων που προσφέρει και γιατί είναι συμβατή με το Prometheus και το Thanos. Το Prometheus ή το Thanos μπορούν να συλλέγουν τα δεδομένα από τα εικονικά μηχανήματα ή οποιοδήποτε σύστημα μιας υποδομής και στην συνέχεια να τα προωθούν στην Grafana για να την δημιουργία των γραφημάτων.

Η συμβολή της πλατφόρμας αυτής στην παρακολούθηση σε πραγματικό χρόνο μιας υποδομής είναι πολύ σημαντική καθώς με το εύκολο στην χρήση περιβάλλον και τη διαδραστική διεπαφή χρήστη γίνεται ευκολότερη και γρηγορότερη η αναγνώριση, από τον άνθρωπο, μέσω των γραφημάτων, οποιοδήποτε είδος μη αναμενόμενης συμπεριφοράς όπως εξάντληση των πόρων σε ένα εικονικό μηχάνημα (π.χ μνήμη RAM, ελεύθερος χώρος στο δίσκο κ.λ.π). Επιπλέον, οι δυνατότητες για αποστολή ειδοποιήσεων (alerts) σε περίπτωση που διαπιστωθεί κάποιο πρόβλημα είναι εξαιρετικά χρήσιμες δεδομένου ότι μπορούν να σταλούν αυτόματα με email στο αντίστοιχο άτομο που είναι υπεύθυνο να διερευνήσει το πρόβλημα περεταίρω. Να σημειωθεί σε αυτό το σημείο πως οι ειδοποιήσεις μπορούν να σταλούν και σε άλλες πλατφόρμες όπως Slack, Microsoft Teams κ.λ.π.

3.7 Prometheus Anomaly Detector

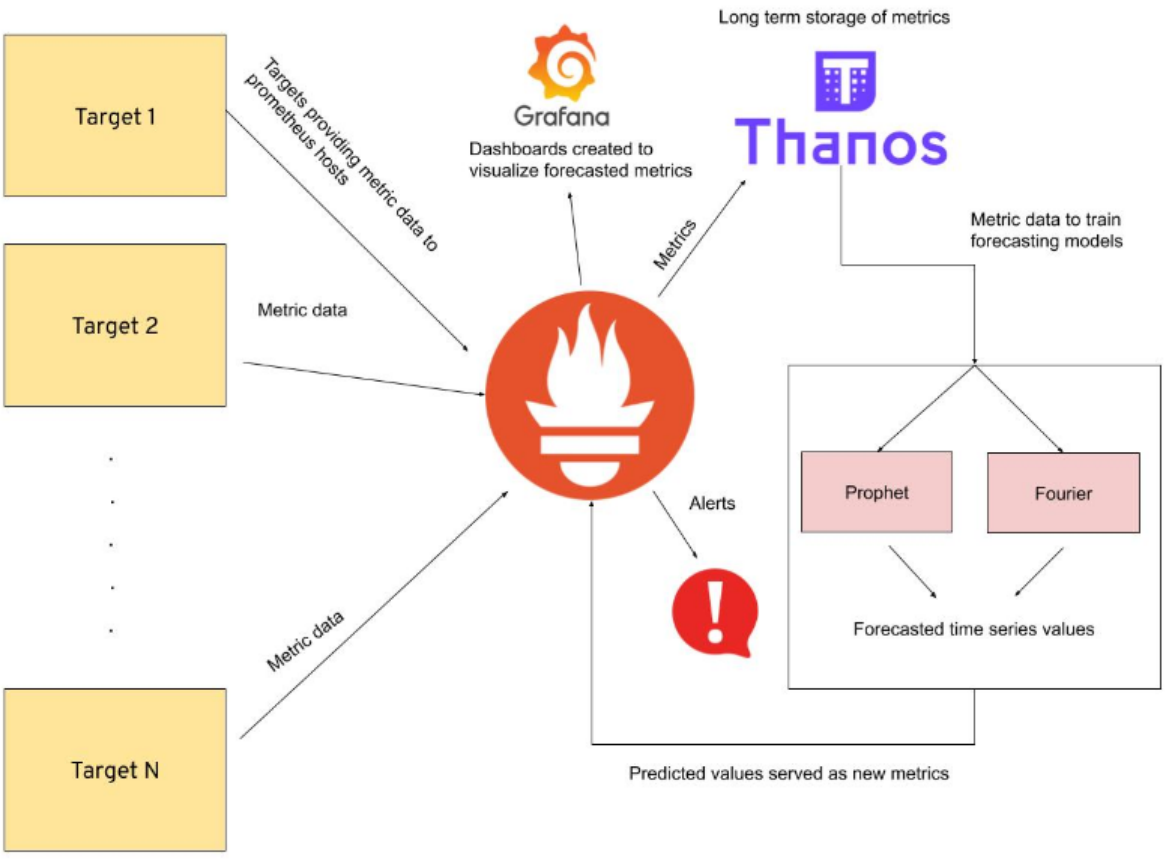
Το Prometheus Anomaly Detector (PAD) είναι ένα πλαίσιο για την υλοποίηση ενός μοντέλου πρόβλεψης και ανίχνευσης ανωμαλιών σε πραγματικό χρόνο από δεδομένα

χρονοσειρών που συλλέγονται από το Prometheus. Για την πρόβλεψη και την ανίχνευση ανωμαλιών το PAD κάνει χρήση των αλγορίθμων μηχανικής μάθησης:

1. Fourier: Χρησιμοποιείται για την μετατροπή των δεδομένων από την πεδίο του χρόνου στο πεδίο της συχνότητας. Μετατρέπει τα δεδομένα χρονοσειρών σε άθροισμα ημιτόνων και συνημιτόνων.
2. Prophet: Χρησιμοποιείται για να συμπεριληφθούν στην πρόβλεψη μη γραμμικές τάσεις που παρουσιάζουν εποχικότητα (ημερήσια, εβδομαδιαία ή ετήσια) καθώς και holiday effects.

Το αποτέλεσμα του PAD αποτελείται από την τιμή yhat που είναι η πραγματική πρόβλεψη, την τιμή yhat_lower που είναι η χαμηλότερη τιμή του διαστήματος αβεβαιότητας και την τιμή yhat_upper που είναι η υψηλότερη τιμή του διαστήματος αβεβαιότητας.

Συνολικά η αρχιτεκτονική της υλοποίησης παρουσιάζεται στο παρακάτω διάγραμμα:



Εικόνα 3.8: Αρχιτεκτονική του συστήματος παρακολούθησης και πρόβλεψης

Κεφάλαιο 4

Υλοποίηση - Εγκατάσταση

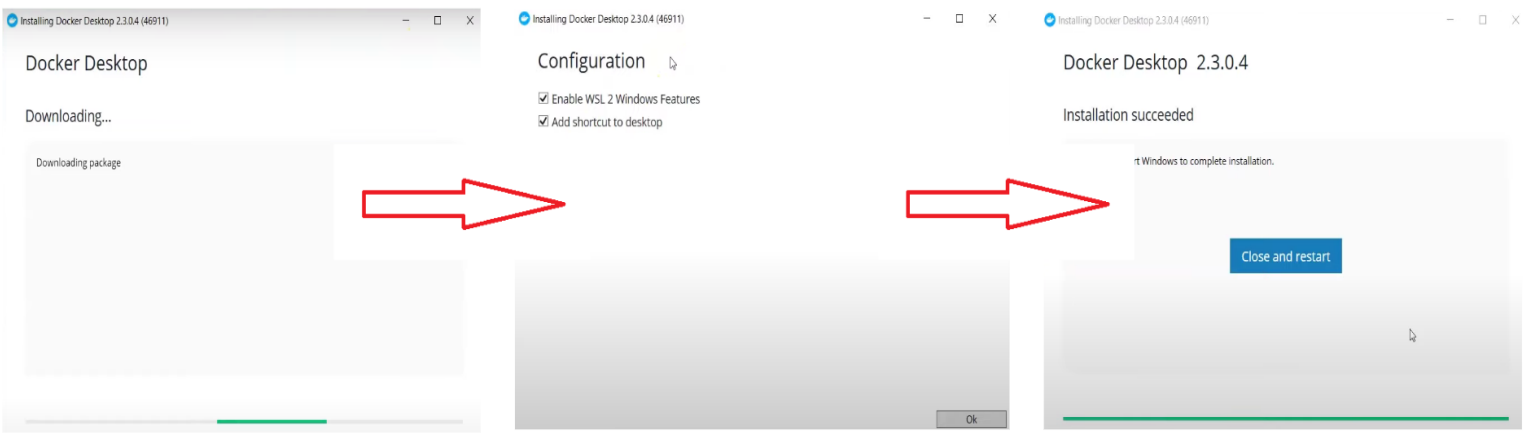
Όπως ήδη αναφέρθηκε στα προηγούμενα κεφάλαια, στα πλαίσια της διπλωματικής έγινε χρήση πολλών τεχνολογιών και στη συνέχεια παρουσιάζεται η εγκατάσταση και η παραμετροποίηση αυτών για τους σκοπούς της διπλωματικής. Καθώς έγινε χρήση του Docker χρειάστηκε μόνο η εγκατάσταση αυτού και οι υπόλοιπες τεχνολογίες χρησιμοποιήθηκαν μέσω docker images, εκτός του Thanos που χρειάστηκε επιπλέον η εγκατάσταση της Go για να μπορεί να εκτελεστεί ο πηγαίος κώδικας της εφαρμογής.
Στα πλαίσια της διπλωματικής έγινε η χρήση:

1. Ενός εικονικού μηχανήματος (Windows OS) το οποίο αποτέλεσε τον host για τα container των 2 Prometheus , του Thanos που συνδέθηκε και στα 2 Prometheus, του Minio, της Grafana και του PAD. Έγινε χρήση ενός docker-compose.yml αρχείου για την δημιουργία των containers για τα όλα τα παραπάνω.
2. Ενός εικονικού μηχανήματος (Linux OS) που αποτέλεσε το μηχάνημα στόχο για την συλλογή δεδομένων και την πρόβλεψη πάνω σε αυτά.

Στην συνέχεια παρουσιάζεται η εγκατάσταση και η παραμετροποίηση που χρειάστηκε για κάθε εφαρμογή/πλατφόρμα που χρησιμοποιήθηκε.

4.1 Docker

Για να γίνει χρήση του Docker σε ένα μηχάνημα με λειτουργικό Windows είναι απαραίτητη η εγκατάσταση της εφαρμογής Docker Desktop. Το Docker Desktop εγκαθιστά αυτόματα το Docker Engine για την δημιουργία containers, το Docker Compose καθώς και όλα τα υπόλοιπα απαραίτητα εργαλεία. Το μονο που απαιτείται είναι να έχουμε τοπικά το ".exe" αρχείο (το οποίο υπάρχει στο σύνδεσμο) και να ακολουθηθούν οι οδηγίες εγκατάστασης όπως παρουσιάζονται στην συνέχεια.



Εικόνα 4.1: Βήματα εγκατάστασης Docker Desktop

Στο τέλος επιβεβαιώνουμε την ολοκλήρωση της εγκατάστασης:

```
C:> docker --version
Docker version 20.10.17, build 100c701
C:> docker-compose --version
Docker Compose version v2.10.2
```

Σε μηχάνημα με Linux λειτουργικό σύστημα η εγκατάσταση του docker, του docker-compose και των αντίστοιχων εργαλείων η εγκατάσταση γίνεται μέσω εντολών:

Docker

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg

$ echo "deb [arch=$(dpkg --print-architecture)
signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null

$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io

$ docker --version
Docker version 20.10.14, build a224086
```

Docker-compose

```
$ sudo curl -L
"https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(unam
e -s)-$(uname -m)" -o /usr/local/bin/docker-compose

$ sudo chmod +x /usr/local/bin/docker-compose

$ docker-compose --version
docker-compose version 1.29.2, build 5becea4c
```

Στην συνέχεια για την δημιουργία των containers των αντίστοιχων εφαρμογών που χρησιμοποιήθηκαν στο μηχάνημα με Windows λειτουργικό σύστημα, δημιουργήθηκε το αρχείο docker-compose.yml το οποίο περιέχει όλη την απαραίτητη παραμετροποίηση για τις εφαρμογές καθώς και την γενική παραμετροποίηση ώστε οι εφαρμογές να επικοινωνούν μεταξύ τους με την δημιουργία του αντίστοιχου δικτύου. Στην συνέχεια παρουσιάζεται το κομμάτι που αφορά την δημιουργία του δικτύου, ενώ οι σχετικές με τις εφαρμογές παραμετροποιήσεις μελετούνται σε επόμενο κεφάλαιο.

docker-compose.yml (στο μηχάνημα που χρησιμοποιήθηκε για την εγκατάσταση των εφαρμογών παρακολούθησης και πρόβλεψης)

```
version: '3.2'

networks:
  monitoring-net:
    driver: bridge
    ipam:
     config:
       - subnet: 10.1.0.0/28

services:
# Παρουσιάζονται σε επόμενο κεφάλαιο
```

Στο πεδίο 'networks' ορίζεται το όνομα του δικτύου που θα χρησιμοποιηθεί (*monitoring-net*), το είδος του δικτύου (*bridge*), το οποίο επιτρέπει στα containers να χρησιμοποιούν την κάρτα δικτύου το μηχανήματος που τα φιλοξενεί και για να είναι διαθέσιμα οι εφαρμογές στον host πρέπει να γίνει ειδική παραμετροποίηση για να προωθούν τις πόρτες (ports) που χρησιμοποιούν. Τέλος ορίζεται το *subnet* των διαθέσιμων IP διευθύνσεων που πρόκειται να δοθούν στα containers (*10.1.0.0/28*). Αξίζει να σημειωθεί πως με την χρήση του συγκεκριμένου είδους δικτύου υπάρχει DNS που υλοποιείται αυτόματα και τα container μπορούν να επικοινωνούν μεταξύ τους και με τα ονόματά τους όπως φαίνεται στα επόμενα κεφάλαια.

Το είδος δικτύου που χρησιμοποιήθηκε για το container του node-exporter στο εικονικό μηχάνημα στόχος είναι 'host'. Στην περίπτωση αυτή το εικονικό μηχάνημα μοιράζεται τα πάντα σε επίπεδο δικτύου με το container. Δεν είναι απαραίτητη η προώθηση κάποια πόρτας στο μηχάνημα που φιλοξενεί το container, καθώς είναι ήδη διαθέσιμη σε αυτό όπως ακριβώς μια εφαρμογή που θα είχε γίνει κανονική εγκατάσταση σε αυτό και όχι μέσω docker container.

4.2 Prometheus

Για την δημιουργία των containers των Prometheus instances χρειάστηκαν τα εξής:

1. Τα αρχεία prometheus.yml για το κάθε Prometheus instance.
2. Την παραμετροποίηση του αρχείου docker-compose.yml αντίστοιχα.

Το 1ο Prometheus instance θα παρακολουθεί το εικονικό μηχάνημα στόχο της υποδομής καθώς και όλη την υπόλοιπη υποδομή ενώ το 2ο Prometheus instance θα παρακολουθεί μόνο το εικονικό μηχάνημα, το 1ο Prometheus instance καθώς και τα δεδομένα που προκύπτουν από την πρόβλεψη.

prometheus.yml

```
global:
  scrape_interval:     30s
  external_labels:
    monitor: 'prometheus_one'
scrape_configs:
  - job_name: 'prometheus-two'
    static_configs:
      - targets: ['prometheus_two:9002']
  - job_name: 'minio'
    metrics_path: /minio/prometheus/metrics
    static_configs:
      - targets: ['minio:9000']
  - job_name: 'thanos-sidecar-one'
    static_configs:
      - targets: ['thanos_sidecar_one:10902']
  - job_name: 'thanos-sidecar-two'
    static_configs:
      - targets: ['thanos_sidecar_two:10908']
  - job_name: 'thanos-query'
    static_configs:
      - targets: ['thanos_querier:10904']
  - job_name: 'thanos-compact'
    static_configs:
      - targets: ['thanos_compactor:10912']
  - job_name: 'thanos-store'
    static_configs:
      - targets: ['thanos_store:10906']
  - job_name: "VM"
    scrape_interval: 15s
    static_configs:
    - targets: ["62.217.83.162:9100"]
```

prometheus-two.yml

```
global:
  scrape_interval:     30s
  external_labels:
    monitor: 'prometheus_two'
scrape_configs:
  - job_name: 'prometheus-one'
    static_configs:
      - targets: ['prometheus_one:9001']
  - job_name: 'thanos-sidecar_one'
    static_configs:
      - targets: ['thanos_sidecar_one:10908']
  - job_name: "VM"
    scrape_interval: 15s
```

```
    static_configs:
    - targets: ["62.217.83.162:9100"]
  - job_name: "PAD"
    scrape_interval: 60s
    static_configs:
    - targets: ["pad:8080"]
```

Και στα 2 αρχεία στο πεδίο 'global' ορίζονται οι γενικές παραμετροποιήσεις που θα ισχύουν για το εκάστοτε instance. Πιο συγκεκριμένα:

1. scrape_interval: Αφορά το διάστημα που θα μεσολαβεί μεταξύ 2 τιμών που θα διαβάζει το Prometheus από τα μηχανήματα στόχους που παρακολουθεί.
2. external_labels: Αφορά πρόσθετες πληροφορίες που συνοδεύουν τα δεδομένα που συλλέγονται από το από τα εικονικά μηχανήματα στόχους.

Στο πεδίο 'scrape_configs' ορίζονται οι απαραίτητες λεπτομέρειες για τα επιμέρους μηχανήματα/εφαρμογές στόχους από τα οποία θα συλλέγει δεδομένα το εκάστοτε Prometheus. Πιο συγκεκριμένα:

1. job_name: Αφορά επιπρόσθετη πληροφορία που συνοδεύει τα metrics και προσδιορίζει το μηχάνημα από τα οποία συλλέγονται τα διάφορα δεδομένα.
2. scrape_interval: Στο πεδίο global έχει οριστεί ο χρόνος που θα συλλέγονται τα δεδομένα θα είναι κάθε 30 δευτερόλεπτα. Παρόλα αυτά, στην περίπτωση που είναι αναγκαία η χρήση διαφορετικής τιμής για κάποιο μηχάνημα στόχο τότε ορίζεται αποκλειστικά στο συγκεκριμένο πεδίο. Στην περίπτωση της παρούσας υλοποίησης ο χρόνος μεταξύ 2 μετρήσεων για το εικονικό μηχάνημα προσδιορίστηκε στην τιμή 15s για να υπάρχει περισσότερη πληροφορία.
3. static_configs (targets): Στο πεδίο αυτό ορίζεται η διεύθυνση IP και η πόρτα (port) στην οποία το Prometheus θα πρέπει να στείλει το εκάστοτε ερώτημα (request) για το κάθε μηχάνημα/εφαρμογή στόχο.

Στην συνέχεια προστέθηκαν στο docker-compose.yml οι παμετροποιήσεις που χρειάζονται για τα 2 Prometheus instances κάτω από το πεδίο "services":

docker-compose.yml

```
prometheus_one:
    image: prom/prometheus:latest
    container_name: prometheus_one
    user: root
    volumes:
      - ./prometheus:/etc/config/
      - ./data/prometheus/one:/data
    command:
      - '--config.file=/etc/config/prometheus.yml'
      - '--storage.tsdb.path=/data'
      - '--web.console.libraries=/etc/prometheus/console_libraries'
      - '--web.console.templates=/etc/prometheus/consoles'
```

```
    - '--storage.tsdb.retention.time=2h'
    - '--web.enable-lifecycle'
    - '--web.enable-admin-api'
    - '--web.listen-address=:9001'
    - '--storage.tsdb.min-block-duration=5m'
    - '--storage.tsdb.max-block-duration=5m'
  restart: unless-stopped
  networks:
    monitoring-net:
      ipv4_address: 10.1.0.2
  expose:
    - 9001
  ports:
    - "9001:9001"
prometheus_two:
  image: prom/prometheus:latest
  container_name: prometheus_two
  user: root
  volumes:
    - ./prometheus:/etc/config/
    - ./data/prometheus/two:/data
  command:
    - '--config.file=/etc/config/prometheus_two.yml'
    - '--storage.tsdb.path=/data'
    - '--web.console.libraries=/etc/prometheus/console_libraries'
    - '--web.console.templates=/etc/prometheus/consoles'
    - '--storage.tsdb.retention.time=2h'
    - '--web.enable-lifecycle'
    - '--web.enable-admin-api'
    - '--web.listen-address=:9002'
    - '--storage.tsdb.min-block-duration=5m'
    - '--storage.tsdb.max-block-duration=5m'
  restart: unless-stopped
  networks:
    monitoring-net:
      ipv4_address: 10.1.0.10
  expose:
    - 9002
  ports:
    - "9002:9001"
```

Στην συνέχεια αναλύεται η παραμετροποίηση του docker-compose.yml αρχείου:

1. image: Ορίζεται το αρχικό image που θα χρησιμοποιηθεί για το καινούριο container και το οποίο είναι διαθέσιμο στο Docker Hub. Στην προκειμένη περίπτωση το “prom/prometheus” το οποίο διαθέτει το tag “latest”, που σημαίνει το τελευταίο διαθέσιμο και stable version.
2. container_name: Ορίζεται το όνομα του εκάστοτε container με το οποίο είναι προσπελάσιμο.

3. volumes: Ορίζονται πιθανή αντιστοίχιση φακέλων (directory) στο μηχάνημα που φιλοξενεί τα container με φακέλους εντός του container. Με αυτό τον τρόπο αποθηκεύονται τα δεδομένα και δεν χάνονται ακόμα και αν διαγραφεί το container.
4. command: Ορίζονται συγκεκριμένες εντολές που αφορούν συγκεκριμένα την παραμετροποίηση της εφαρμογής που θα τρέχει στο container. Πιο συγκεκριμένα:
    a. config.file: Μέσω της αντιστοίχισης που έχει γίνει μέσω των volumes ορίζεται που βρίσκεται ακριβώς το αρχείο prometheus.yml μέσα στο container που περιέχει την πληροφορία που δώσαμε προηγουμένως.
    b. storage.tsdb.path: Ορίζεται σε ποιό ακριβώς σημείο (φάκελο) θα αποθηκεύονται τα δεδομένα που συλλέγει ο Prometheus.
    c. web.console.libraries: Ορίζεται σε ποιό ακριβώς σημείο (φάκελο) θα αποθηκεύονται αρχεία βιβλιοθήκης που αφορούν την διεπαφή χρήστη (web interface) του Prometheus. (Javascript files etc)
    d. web.console.templates: Ορίζεται σε ποιό ακριβώς σημείο (φάκελο) θα αποθηκεύονται τα πρότυπα αρχεία που αφορούν την εμφάνιση της διεπαφής χρήστη του Prometheus.
    e. storage.tsdb.retention.time: Ορίζεται για πόσο διάστημα θα αποθηκεύονται δεδομένα στον Prometheus στην προκειμένη περίπτωση 2 ώρες. Δεδομένα με χρονική τιμή μεγαλύτερη των 2 ωρών θα διαγράφονται αυτόματα.
    f. web.enable-lifecycle: Ενεργοποιείται το HTTP API του Prometheus με αποτελέσματα να μπορούν να γίνουν κάποιες διαχειριστικές ενέργειες μέσω αυτού. (reloads and shutdowns)
    g. web.enable-admin-api: Ενεργοποιείται το HTTP API του Prometheus που αφορά διαχειριστικές ενέργειες πάνω στα δεδομένα χρονοσειρών.
    h. storage.tsdb.min-block-duration & storage.tsdb.max-block-duration: Ορίζεται το ελάχιστο και το μέγιστο μέγεθος ενός block δεδομένων που θα αποθηκεύονται από το Prometheus. Οι τιμες εδώ ορίστηκαν ίσες με σκοπό να απενεργοποιηθεί η συμπίεση (compaction) που κάνει το Prometheus αφού θα χρησιμοποιηθεί το Thanos Compactor για αυτό.
5. restart: Ορίζεται η πολιτική για την επανεκκίνηση του container. Θα πρέπει να επανεκκινεί ξανά εκτός αν το σταματήσει κάποιος χρήστης.
6. networks: Ορίζεται η διεύθυνση IP του εκάστοτε container από το subnet που έχει οριστεί νωρίτερα.
7. expose: Ορίζεται ποια πόρτα θα είναι ορατή στα υπόλοιπα container.
8. ports: Ορίζεται ποιά πόρτα του container θα αντιστοιχιστεί σε ποιά πόρτα του μηχανήματος που το φιλοξενεί με σκοπό η εφαρμογή να είναι προσβάσιμη από εξωτερικά δίκτυα. Δεδομένου ότι το Prometheus χρησιμοποιεί τη πόρτα 9001 και υπάρχουν 2 instances του Prometheus και ένα μηχάνημα που τα φιλοξενεί χρησιμοποιήθηκαν 2 πόρτες του host , η 9001 και η 9002.

4.3 Node Exporter

Στο μηχάνημα στόχο το οποίο θα παρακολουθείται έπρεπε να είναι διαθέσιμα τα δεδομένα που αφορούν το συγκεκριμένο VM. Για τον λόγο αυτό στην συνέχεια γίνεται ανάλυση της παραμετροποίησης του Node Exporter, που είναι υπεύθυνος να προσφέρει αυτά τα δεδομένα για ένα Linux μηχάνημα.

docker-compose.yml

```yaml
version: '3.8'
services:
  node-exporter:
    image: prom/node-exporter:latest
    container_name: node-exporter
    restart: unless-stopped
    volumes:
      - /proc:/host/proc:ro
      - /sys:/host/sys:ro
      - /:/rootfs:ro
    command:
      - '--path.procfs=/host/proc'
      - '--path.rootfs=/rootfs'
      - '--path.sysfs=/host/sys'
      - '--collector.filesystem.mount-points-exclude=^/(sys|proc|dev|host|etc)($$|/)'
    network_mode: "host"
```

Πιο συγκεκριμένα:

1. Χρησιμοποιήθηκε το image “prom/node-exporter” με το tag “latest” από το Docker Hub.
2. Το container ονομάστηκε node_exporter.
3. Η πολιτική για την επανεκκίνηση του container ορίστηκε ώστε να γίνεται πάντα restart εκτός αν το σταματήσει ο χρήστης.
4. Η αντιστοιχία των directories στο μηχάνημα που φιλοξενεί το container έγινε με τρόπο ώστε το node exporter εντός του container να παρακολουθεί στην πραγματικότητα το εικονικό μηχάνημα που είναι και αυτό που φιλοξενεί το container.
5. Στο πεδίο “command” δηλώθηκαν σχετικές εντολές για την σωστή λειτουργία του node exporter δηλώντας σε ποιό σημείο θα βρίσκονται δεδομένα σχετικά τις διεργασίες (*procfs*) που τρέχουν στο εικονικό μηχάνημα, τα χαρακτηριστικά του υλικού-hardware (*sysfs*) του εικονικού μηχανήματος και καθώς και τα δεδομένα για όλο το εικονικό μηχάνημα κάτω από το αρχικό directory. (root)
6. Τέλος ορίστηκε όπως αναφέρθηκε στο προηγούμενο κεφάλαιο η δικτυακή σχέση που έχουν το container και το εικονικό μηχάνημα που το φιλοξενεί ώστε το container να μοιράζεται τα πάντα σε επίπεδο δικτύου με τον host.

4.4 Minio

Το Minio χρησιμοποιήθηκε για να μπορούν να αποθηκεύονται μακροχρόνια, σε συνδυασμό με το Thanos, τα δεδομένα που συλλέγονται από το Prometheus. Σύμφωνα με την επίσημη ιστοσελίδα του Minio η υλοποίηση ενός container σε μια λύση δεν είναι η βέλτιστη αλλά για τα πλαίσια της διπλωματικής εξυπηρετεί τον σκοπό. Στην συνέχεια, αναλύεται η παραμετροποίηση το docker-compose.yml για το Minio.

docker-compose.yml

```
minio:
    image: minio/minio:latest
    container_name: minio
    volumes:
      - ./data/minio:/data
    networks:
      monitoring-net:
        ipv4_address: 10.1.0.4
    ports:
      - "9000:9000"
   expose:
     -9000
   environment:
     MINIO_PROMETHEUS_AUTH_TYPE: public
     MINIO_ROOT_USER: myrootuser
     MINIO_ROOT_PASSWORD: myrootpassword
   command: server /data
   healthcheck:
     test: ["CMD", "curl", "-f", "http://localhost:9000/minio/health/live"]
     interval: 30s
     timeout: 20s
     retries: 2
   restart: unless-stopped
```

Πιο αναλυτικά:

1. Χρησιμοποιήθηκε το image: minio/minio με το tag latest, δηλαδή την τελευταια σταθερή έκδοση.
2. Το container ονομάστηκε minio.
3. Έγινε αντιστοιχία του φακέλου που αποθηκεύονται τα δεδομένα στο container με ένα φάκελο στο μηχάνημα που το φιλοξενεί ώστε τα δεδομένα να μην χάνονται όταν το container επανεκκινεί.
4. Δόθηκε η IP 10.1.04 από το δίκτυο που έχει φτιαχτεί για τα containers.
5. Η πόρτα 9000 του container έγινε διαθέσιμη στην πόρτα 9000 του μηχανήματος που το φιλοξενεί.

6. Το container επιτρέπει στα υπόλοιπα container να επικοινωνούν με αυτό στην πόρτα 9000.
7. Στο πεδίο “environment” ορίστηκαν οι απαραίτητες μεταβλητές για την ομαλή λειτουργία του minio.
    a. MINIO_PROMETHEUS_AUTH_TYPE: Ορίστηκε ως public με αποτέλεσμα να μην χρειάζεται το Prometheus “όνομα χρήστη” και κωδικό για να επικοινωνήσει με το minio.
    b. MINIO_ROOT_USER: Ορίστηκε το username με το οποίο μπορούν οι χρήστες να συνδέονται στο minio.
    c. MINIO_ROOT_PASSWORD: Ορίστηκε το password του παραπάνω χρήστη.
8. Στο πεδίο “command” ορίστηκε η εντολή που χρειάζεται να τρέξει μόλις εκκινήσει το container ώστε να λειτουργήσει το minio.
9. Στο πεδίο healthcheck ορίστηκε ο έλεγχος που θα γίνεται για να επιβεβαιώνεται ότι το container τρέχει ομαλά.
    a. Θα ελέγχεται ότι επιστρέφει έγκυρη απάντηση (200 OK) το url: http://localhost:9000/minio/health/live
    b. Ο έλεγχος θα γίνεται κάθε 30 δευτερόλεπτα.
    c. Αν το συγκεκριμένο url δεν απαντήσει σε 20 δευτερόλεπτα τότε θεωρείται ότι δεν απάντησε επιτυχώς.
    d. Θα συνεχίζεται να γίνεται ο έλεγχος εώς ότου δεν υπάρχει έγκυρη απάντηση για 3η φορά. Στην περίπτωση αυτή το container θα μαρκαριστεί ως μη υγιές και δεν θα επανεκκινήσει.
10. Η πολιτική επανεκκίνησης του container είναι να επανεκκινεί συνεχώς εκτός αν ο χρήστης το σταματήσει , ή στην προκειμένη περίπτωση αν δεν απαντήσει για 3 φορές το url που επιβεβαιώνει ότι είναι ενεργό το minio ή όχι.

4.5 Thanos

Το Thanos προσέφερε στην λύση συνεχή διάθεση δεδομένων ακόμα και όταν κάποιο από τα Prometheus instances είναι εκτός λειτουργίας καθώς και δυνατότητα εποπτείας των μηχανημάτων/εφαρμογων στόχων από το χρονικό διάστημα των 15 ημερών που προσφέρει το Prometheus. Για να μπορέσουν να δημιουργηθούν τα container για τα διάφορα thanos components υπήρξε η ανάγκη πρώτα να υπάρχει τοπικά στο μηχάνημα ο πηγαίος κώδικας του Thanos και να δημιουργηθεί το binary αρχείο του (compiling).

Προτού γίνει η ανάλυση του docker-compose.yml για το Thanos πρέπει να αναφερθούν τα εξής:

1. Δημιουργήθηκε αρχικά ένας ξεχωριστός φάκελος (thanos-source-code), στο ίδιο σημείο (directory) που βρίσκεται το docker-compose.yml, και αντιγράφηκε ο πηγαίος κώδικας του Thanos χρησιμοποιώντας την εντολή:
```
git clone https://github.com/thanos-io/thanos
```
2. Στο ίδιο σημείο (directory) με το docker-compose.yml δημιουργήθηκε επιπλέον το αρχείο Makefile το οποίο συνέβαλε στο να αυτοματοποιηθεί η διαδικασία για την

δημιουργία του binary του Thanos καθώς και το να γίνεται η δημιουργία, η διαγραφή και η επανεκκίνηση όλων των containers.

Makefile

```
THANOS_SOURCE ?= ../thanos-source-code
COMPOSE_FILE ?= docker-compose.yml

GOPATH            ?= $(shell go env GOPATH)
GOBIN             ?= $(firstword $(subst :, ,${GOPATH}))/bin

THANOS_BINARY ?= $(GOBIN)/thanos

.PHONY: help
help: ## Displays help.
        @awk 'BEGIN {FS = ":.*##"; printf "\nUsage:\n  make
\033[36m<target>\033[0m\n\nTargets:\n"} /^[a-z0-9A-Z_-]+:.*?##/ { printf "
\033[36m%-10s\033[0m %s\n", $$1, $$2 }' $(MAKEFILE_LIST)

.PHONY: $(THANOS_BINARY)
$(THANOS_BINARY): ## Builds the Thanos binary from source code
$(THANOS_BINARY):
        @echo ">> building the Thanos binary"
        @make -C "$(THANOS_SOURCE)" build

.PHONY: up
up: ## Bootstraps a docker-compose setup for local development/demo
up: $(THANOS_BINARY)
        @echo ">> copying binaries to development env"
        @rm -f ./thanos/thanos || true
        cp "$(THANOS_BINARY)" ./thanos/
        docker-compose -f "$(COMPOSE_FILE)" up -d --build

.PHONY: restart
restart: ## Rebuilds and restarts the container without building the binary
restart:
        docker-compose -f "$(COMPOSE_FILE)" up -d --build

.PHONY: down
down: ## Brings down the docker-compose setup
down:
        docker-compose -f "$(COMPOSE_FILE)" down
```

Με την δημιουργία του αρχείου αυτού δίνεται η δυνατότητα για τις παρακάτω επιλογές:

```
$ make help
Usage:
  make <target>
Targets:
  help     Displays help.
  up       Brings up the docker-compose setup and builds thanos binary
  restart  Rebuilds and restarts the container without building the thanos binary
  down     Brings down the docker-compose setup
```

3. Από τα components του Thanos όπως αναφέρθηκε στην προηγούμενη ενότητα χρησιμοποιήθηκαν στην παρούσα διπλωματική τα: Sidecar, Store, Querier, Compactor. Δεδομένου ότι τα container φιλοξενούνται στο ίδιο μηχάνημα ακολουθήθηκε η ανάθεση πορτών (ports) επικοινωνίας με το κάθε ένα από αυτά σύμφωνα με τον επίσημο οδηγό που βρίσκεται στην ιστοσελίδα του Thanos και παρουσιάζεται παρακάτω.

Component	Interface	Port
Sidecar	gRPC	10901
Sidecar	HTTP	10902
Query	gRPC	10903
Query	HTTP	10904
Store	gRPC	10905
Store	HTTP	10906
Receive	gRPC (store API)	10907
Receive	HTTP (remote write API)	10908
Receive	HTTP	10909
Rule	gRPC	10910
Rule	HTTP	10911
Compact	HTTP	10912

Εικόνα 4.2: Thanos on Single Host

Στην συνέχεια αναλύεται η παραμετροποίηση του docker-compose.yml για τα διάφορα components του Thanos που χρησιμοποιήθηκαν καθώς και το Dockerfile που χρειάστηκε για να δημιουργηθεί το αρχικό image των containers:

Dockerfile.thanos

```
FROM alpine:3.6 as alpine
RUN apk add -U --no-cache ca-certificates bash

ENTRYPOINT []
WORKDIR /

COPY ./thanos /
```

Ως βάση για τα container χρησιμοποιήθηκε σαν αρχικό image το alpine:3.6 το οποίο είναι έκδοση των Linux. Στην συνέχεια, γίνεται η εγκατάσταση των πακέτων “ca-certificates” και “bash” στο κάθε container. Ορίζεται ότι δεν χρειάζεται να εκτελεστούν εντολές μόλις δημιουργηθεί το image θα εκτελεστεί μόνο ό,τι οριστεί στην εντολή “docker run” μέσω του docker-compose.yml. Τέλος ορίζεται το directory μέσα στο container που θα εκτελούνται όλες οι εντολές και γίνεται αντιγραφή του binary του Thanos σε αυτό.

docker-compose.yml

```
thanos_sidecar_one:
    build:
      context: ./thanos
      dockerfile: Dockerfile.thanos
    container_name: thanos_sidecar_one
    volumes:
      - ./prometheus:/etc/config/
      - ./data/prometheus:/data
    command:
      - "/thanos"
      - "sidecar"
      - "--log.level=debug"
      - "--tsdb.path=/data"
      - "--prometheus.url=http://prometheus:9001"
      - "--reloader.config-file=/etc/config/prometheus.yml"
      - |
          --objstore.config=type: S3
          config:
            bucket: thanos-data-bucket
            access_key: myrootuser
            secret_key: myrootpassword
            endpoint: minio:9000
            insecure: true
```

```
    networks:
      monitoring-net:
        ipv4_address: 10.1.0.5
    expose:
      - 10902
      - 10901
    depends_on:
      - minio
      - prometheus

thanos_sidecar_two:
    build:
      context: ./thanos
      dockerfile: Dockerfile.thanos
    container_name: thanos_sidecar_two
    volumes:
      - ./prometheus:/etc/config/
      - ./data/prometheus_two:/data
    command:
      - "/thanos"
      - "sidecar"
      - "--log.level=debug"
      - "--tsdb.path=/data"
      - "--grpc-address=0.0.0.0:10907"
      - "--http-address=0.0.0.0:10908"
      - "--prometheus.url=http://prometheus_two:9002"
      - "--reloader.config-file=/etc/config/prometheus_two.yml"
      - |
          --objstore.config=type: S3
          config:
            bucket: thanos-data-bucket
            access_key: myrootuser
            secret_key: myrootpassword
            endpoint: minio:9000
            insecure: true
    networks:
      monitoring-net:
        ipv4_address: 10.1.0.7
    expose:
      - 10907
      - 10908
    depends_on:
      - minio
      - prometheus_two

thanos_querier:
    build:
      context: ./thanos
      dockerfile: Dockerfile.thanos
    container_name: thanos_querier
    command:
```

```
      - "/thanos"
      - "query"
      - "--log.level=debug"
      - "--grpc-address=0.0.0.0:10903"
      - "--http-address=0.0.0.0:10904"
      - "--log.format=logfmt"
      - "--store=thanos_sidecar_one:10901"
      - "--store=thanos_sidecar_two:10907"
      - "--store=thanos_store:10905"
      - "--store.sd-interval=5m"
      - "--query.replica-label=monitor"
    networks:
      monitoring-net:
        ipv4_address: 10.1.0.6
    expose:
      - 10903
      - 10904
    ports:
      - "10904:10904"
    depends_on:
      - minio

thanos_compactor:
    build:
      context: ./thanos
      dockerfile: Dockerfile.thanos
    container_name: thanos_compactor
    volumes:
      - ./data/compactor:/data
    command:
      - "/thanos"
      - "compact"
      - "--log.level=debug"
      - "--log.format=logfmt"
      - "--http-address=0.0.0.0:10912"
      - |
        --objstore.config=type: S3
        config:
            bucket: thanos-data-bucket
            access_key: myrootuser
            secret_key: myrootpassword
            endpoint: minio:9000
            insecure: true
      - "--data-dir=/data"
      - "--consistency-delay=30m"
      - "--retention.resolution-raw=30d"
      - "--retention.resolution-5m=120d"
      - "--retention.resolution-1h=1y"
      - "--compact.concurrency=1"
      - "--delete-delay=15m"
      - "--wait"
```

```
      - "--wait-interval=3m"
    networks:
      monitoring-net:
        ipv4_address: 10.1.0.8
    expose:
      - 10912
    ports:
      - "10912:10912"
    depends_on:
      - minio

thanos_store:
    build:
      context: ./thanos
      dockerfile: Dockerfile.thanos
    container_name: thanos_store
    volumes:
      - ./data/store:/data
    command:
      - "/thanos"
      - "store"
      - "--log.level=debug"
      - "--grpc-address=0.0.0.0:10905"
      - "--http-address=0.0.0.0:10906"
      - |
        --objstore.config=type: S3
        config:
            bucket: thanos-data-bucket
            access_key: myrootuser
            secret_key: myrootpassword
            endpoint: minio:9000
            insecure: true
      - "--data-dir=/data"
      - "--log.format=logfmt"
      - "--index-cache-size=250MB"
      - "--chunk-pool-size=1GB"
      - "--store.grpc.series-max-concurrency=20"
      - "--sync-block-duration=3m"
      - "--block-sync-concurrency=20"
    restart: unless-stopped
    networks:
      monitoring-net:
        ipv4_address: 10.1.0.9
    expose:
      - 10905
      - 10906
    depends_on:
      - minio
    ports:
      - "10906:10906"
```

Thanos Sidecar

1. Τα Thanos Sidecar container ονομάστηκαν με βάση το Prometheus instance που θα συνοδεύουν.
2. Για να φτιαχτούν τα container ορίστηκε ότι πρέπει να δημιουργηθούν με βάση το image που περιγράφεται στο Dockerfile.thanos.
3. Έγινε αντιστοίχιση των directories εκτός του μηχανήματος που φιλοξενεί τα container με directories μέσα στα container ώστε να έχουν πρόσβαση στα δεδομένα του Prometheus που το καθένα συνοδεύει.
4. Στο πεδίο command ορίστηκε η εντολή που είναι απαραίτητη για να εκκινήσει το Thanos Sidecar μαζί με τις ειδικές εντολές παραμετροποίησης του.
    a. */thanos sidecar*: Εκκινεί το sidecar component του Thanos.
    b. *--log.level=debug*: Ορίζεται το επίπεδο των logs που θα εμφανίζονται, με τον όρο debug ορίζεται πιο λεπτομερής καταγραφή.
    c. *--tsdb.path=/data*: Ορίζεται το σημείο όπου βρίσκονται τα δεδομένα χρονοσειρών που συλλέγονται από το Prometheus.
    d. *--prometheus.url=http://prometheus:9001*: Ορίζεται το url του Prometheus που συνοδεύουν.
    e. *--reloader.config-file=/etc/config/prometheus.yml*: Ορίζεται το αρχείο παραμετροποίησης του Prometheus που συνοδεύουν.
    f. *--objstore.config=type: S3*: Ορίζεται το είδος του bucket που αποθηκεύονται τα δεδομένα χρονοσειρών, στην παρούσα διπλωματική του minion, καθώς και τα δεδομένα που χρειάζονται για να έχει πρόσβαση το sidecar σε αυτό (username,password κ.λ.π).
5. Στην συνεχεια στο πεδίο network ορίζεται η IP του συγκεκριμένου container.
6. Στο πεδίο ports ορίζονται οι πόρτες του HTTP και gPRC API όπου μπορούν τα υπόλοιπα container να επικοινωνούν.
7. Στο πεδίο depends_on ορίζεται ότι το τα thanos sidecar πρέπει να περιμένουν πριν δημιουργηθούν τα container τους να δημιουργηθούν τα container του minio και του αντίστοιχου Prometheus που συνοδεύουν.

Thanos Store

1. Για το container του Thanos Store χρησιμοποιήθηκε το image που περιγράφεται στο Dockerfile.thanos.
2. Δημιουργήθηκε αντιστοιχία directories ώστε τα δεδομένα του Store να μην χάνονται όταν το container διαγραφεί ή επανεκκινήσει.
3. Για το Thanos Store έγινε η εξής παραμετροποίηση:
    a. */thanos store*: Εκκινεί το store component του Thanos.
    b. *--log.level=debug*: Ορίζεται το επίπεδο των logs που θα εμφανίζονται, με τον όρο debug ορίζεται πιο λεπτομερής καταγραφή.

c. *--grpc-address=0.0.0.0:10905 & --http-address=0.0.0.0:10906*: Ορίστηκαν οι πόρτες για το HTTP και gRPC API του store.
d. *--data-dir=/data*: Ορίζεται το σημείο όπου βρίσκονται τα δεδομένα χρονοσειρών του store.
e. *--log.format=logfmt*: Ορίζεται η μορφή με την οποία θα εμφανίζονται τα logs ως key-value (κλειδί-τιμή).
f. *--index-cache-size=250MB*: Ορίζεται ο αριθμός bytes από δεδομένα τύπου index (σελιδοδείκτες) και μεταδεδομένα (metadata) των δεδομένων χρονοσειρών που θα αποθηκεύονται στην προσωρινή μνήμη για να είναι πιο γρήγορα προσπελάσιμα τα πραγματικά δεδομένα.
g. *--chunk-pool-size=1GB*: Ορίζεται ο αριθμός των δεδομένων χρονοσειρών που θα αποθηκεύονται στη προσωρινή μνήμη.
h. *--store.grpc.series-max-concurrency=20*: Ορίζεται ο μέγιστος αριθμός παράλληλων ερωτημάτων στο gRPC API.
i. *--sync-block-duration=3m*: Ορίζεται ανά πόσο χρονικό διάστημα θα γίνεται συγχρονισμός των δεδομένων που βρίσκονται τοπικά στο Thanos Store με τα υπόλοιπα components.
j. *--block-sync-concurrency=20*: Ορίζεται ο μέγιστος αριθμός από πακέτα δεδομένων (block) που θα συγχρονίζονται παράλληλα.

Thanos Querier

1. Για το container του Thanos Store χρησιμοποιήθηκε το image που περιγράφεται στο Dockerfile.thanos.
2. Για το Thanos Querier έγινε η εξής παραμετροποίηση:
    a. */thanos query*: Εκκινεί το query component του Thanos.
    b. *--log.level=debug*: Ορίζεται το επίπεδο των logs που θα εμφανίζονται, με τον όρο debug ορίζεται πιο λεπτομερής καταγραφή.
    c. *--grpc-address=0.0.0.0:10903 & --http-address=0.0.0.0:10904*: Ορίστηκαν οι πόρτες για το HTTP και gRPC API του store.
    d. --log.format=logfmt: Ορίζεται η μορφή με την οποία θα εμφανίζονται τα logs ως key-value (κλειδί-τιμή).
    e. *--store=thanos_sidecar_one:10901 & --store=thanos_sidecar_two:10907 & --store=thanos_store:10905*: Ορίζονται τα gRPC API των υπόλοιπων components ώστε το Query να μπορεί να επικοινωνήσει μαζί τους.
    f. *--store.sd-interval=5m:* Ορίστηκε το χρονικό διάστημα κατά το οποίο το Query θα ελέγχει για την ύπαρξη νέων components εκτός από αυτά που ορίστηκαν στο προηγούμενο στάδιο.
    g. *--query.replica-label=monitor*: Ορίστηκε η τιμή την οποία πρέπει να ελέγχει το Query στα δεδομένα ώστε να φέρνει αποτέλεσμα που έχει προκύψει από αφαίρεση των διπλών δεδομένων λόγω της ύπαρξης 2 Prometheus. Η τιμή

αυτή έχει οριστεί σαν label στην παραμετροποίηση του Prometheus όπως φαίνεται στην Ενότητα 2 του κεφαλαίου.

3. Τέλος, όπως σε όλα τα container, ορίστηκε η IP του, η πόρτες στις οποίες μπορεί να επικοινωνεί με τα υπόλοιπα container, η πόρτα την οποία "καθρεπτίζει" στο μηχάνημα που το φιλοξενεί κ.λ.π.

Thanos Compactor

1. Για το container του Thanos Store χρησιμοποιήθηκε το image που περιγράφεται στο Dockerfile.thanos.
2. Αντιστοιχίστηκε το directory εντός του container με ένα στο μηχάνημα που το φιλοξενεί ώστε τα δεδομένα να μείνουν ανεπηρέαστα από την λειτουργία του container.
3. Για το Thanos Compactor έγινε η εξής παραμετροποίηση:
    a. */thanos compact*: Εκκινεί το compact component του Thanos.
    b. *--log.level=debug*: Ορίζεται το επίπεδο των logs που θα εμφανίζονται, με τον όρο debug ορίζεται πιο λεπτομερής καταγραφή.
    c. *--http-address=0.0.0.0:10912*: Ορίστηκε η πόρτα για το HTTP API του compactor.
    d. --log.format=logfmt: Ορίζεται η μορφή με την οποία θα εμφανίζονται τα logs ως key-value (κλειδί-τιμή).
    e. Ορίστηκαν οι απαραίτητες μεταβλητές για να έχει πρόσβαση στα δεδομένα του minio.
    f. *--data-dir=/data*: Ορίζεται το directory στο οποίο αποθηκεύονται τα δεδομένα του compactor.
    g. *--consistency-delay=30m*: Τα δεδομένα στο minio που έχουν συλλεχθεί τα τελευταία 30 λεπτά δεν θα γίνεται διαδικασία για συμπίεσης τους.
    h. *--retention.resolution-raw=30d:* Δεδομένα που έχουν συλλεχθεί τις τελευταίες 30 ημέρες θα παραμένουν στην μορφή που είναι χωρίς να δεχθούν downsampling.
    i. *--retention.resolution-5m=120d*: Δεδομένα μεγαλύτερα των 120 ημερών θα υφίστανται downsampling με ανάλυση (resolution) στα 5 λεπτά.
    j. *--retention.resolution-1h=1y*: Δεδομένα μεγαλύτερα του 1 έτους θα υφίστανται downsampling με ανάλυση (resolution) στη 1 ώρα.
    k. *--compact.concurrency=1*: Μια διαδικασία την φορά θα εκτελείται και για να γίνει η συμπίεση των δεδομένων.
    l. *--delete-delay=15m*: Όσα block δεδομένων μαρκάρονται για διαγραφή θα διαγράφονται μετά από 15 λεπτά.
    m. *--wait & --wait-interval=3m*: Επιβάλλουν στον compactor να μην σταματήσει να λειτουργεί αλλά να περιμένει για 3 λεπτά αφότου τελειώσει όσες διαδικασίες για συμπίεση έχει και στην συνέχεια να προσπαθήσει ξανά προκειμένου να βρει νέα block δεδομένων που χρήζουν συμπίεσης.

4. Τέλος, όπως σε όλα τα container, ορίστηκε η IP του, η πόρτες στις οποίες μπορεί να επικοινωνεί με τα υπόλοιπα container, η πόρτα την οποία “καθρεπτίζει” στο μηχάνημα που το φιλοξενεί κ.λ.π.

4.6 Grafana

Το Grafana χρησιμοποιήθηκε για την αναπαράσταση και οπτικοποίηση των δεδομένων που συλλέγονται από την υποδομή.

Για την δημιουργία του container του Grafana παραμετροποιήθηκε το docker-compose.yml αρχείο όπως φαίνεται στην συνέχεια.

docker-compose.yml

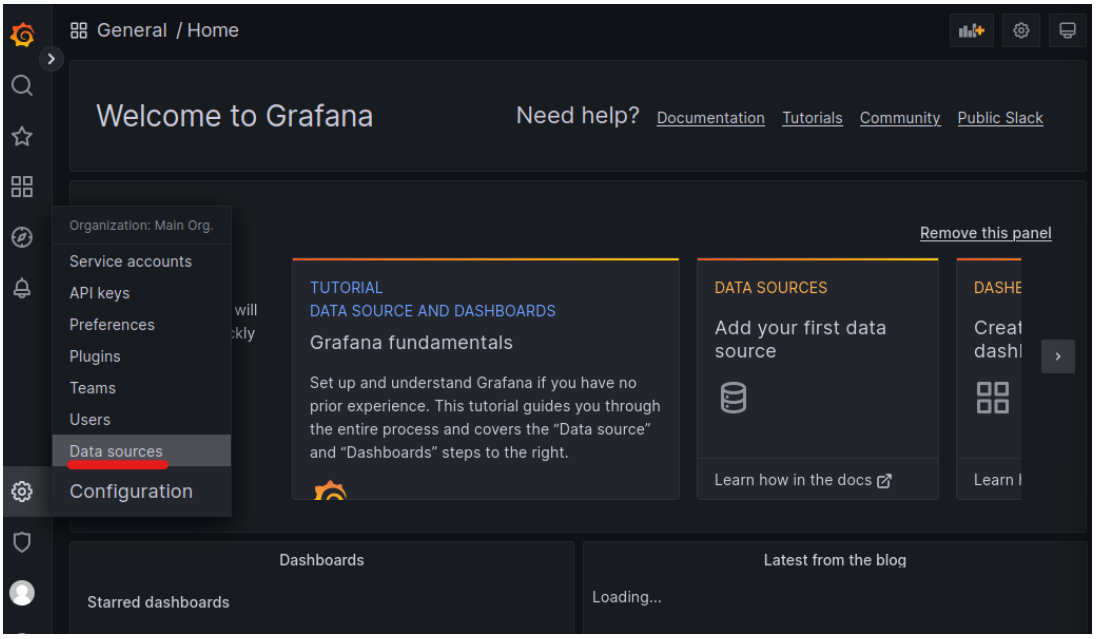
```
grafana:
    image: grafana/grafana:latest
    container_name: grafana
    user: root
    volumes:
     - ./data/grafana:/var/lib/grafana
    environment:
     - GF_SECURITY_ADMIN_USER=${ADMIN_USER:-admin}
     - GF_SECURITY_ADMIN_PASSWORD=${ADMIN_PASSWORD:-admin}
     - GF_USERS_ALLOW_SIGN_UP=false
     - GF_PATHS_DATA=/var/lib/grafana
    restart: unless-stopped
    networks:
      monitoring-net:
        ipv4_address: 10.1.0.3
    ports:
      - "3000:3000"
```

1. Ορίστηκε το image που χρησιμοποιήθηκε από το docker hub, το οποίο είναι η τελευταία σταθερή έκδοση. (*grafana/grafana:latest*)
2. Έγινε αντιστοιχια directories του host με το directories στο container για την διατήρηση των δεδομένων ανεξάρτητα από το container.
3. Ορίστηκαν βασικές μεταβλητές για την ομαλή λειτουργία της Grafanas:
    a. Ορίστηκε το username και το password του διαχειριστή (admin user)
    b. Ορίστηκε ως μη επιτρεπτό νέοι χρήστες να δημιουργούν username και password.
    c. Ορίστηκε το directory που θα αποθηκεύονται τα δεδομένα.
4. Τέλος, όπως σε όλα τα container, ορίστηκε η IP του και η πόρτα την οποία “καθρεπτίζει” στο μηχάνημα που το φιλοξενεί καθώς και η πολιτική επανεκκίνησης του container.

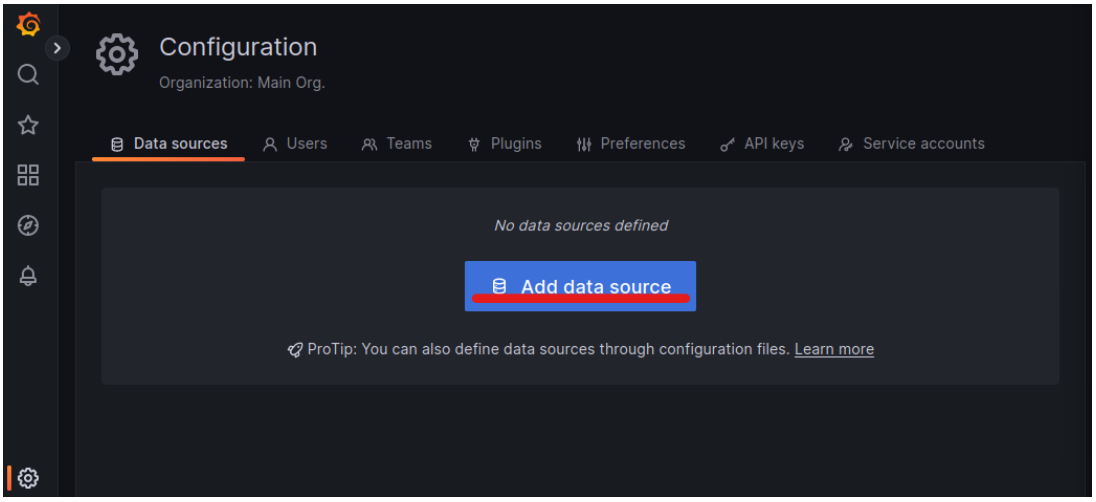
Αφότου δημιουργηθεί το container είναι δυνατή η πρόσβαση στο γραφικό περιβάλλον στο με την χρήση του url: http://10.1.0.3:3000. Στην συνέχεια παρουσιάζεται η διαδικασία για τον ορισμό ενός νέου Data Source, στην προκειμένη περίπτωση του Prometheus και ενός Dashboard. Με τον ίδιο τρόπο προστέθηκε και το Thanos Querier ως Data source.

Προσθήκη Data Source

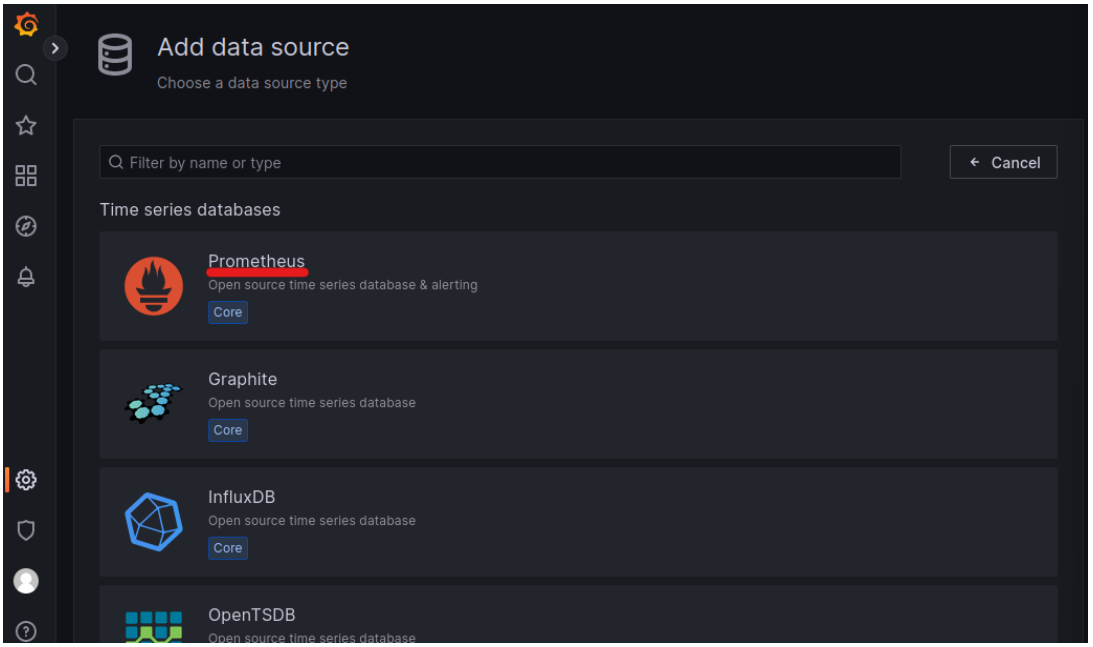
1. Στην αρχική σελίδα στο εικονίδιο με το γρανάζι επιλέγουμε “Data Sources”.



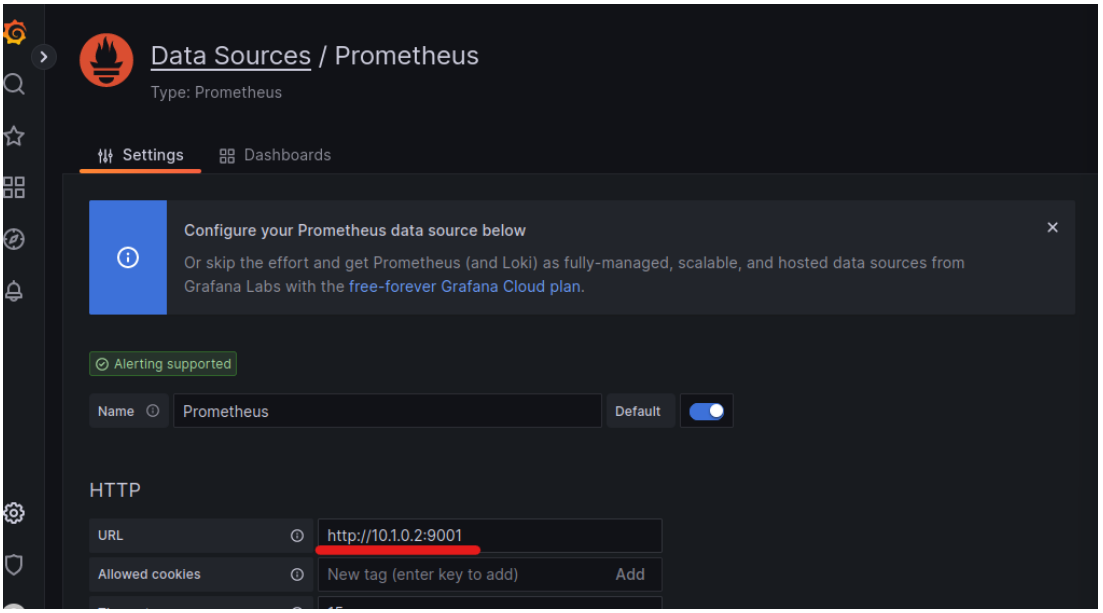
2. Στην συνέχεια επιλέγουμε “Add data source”.



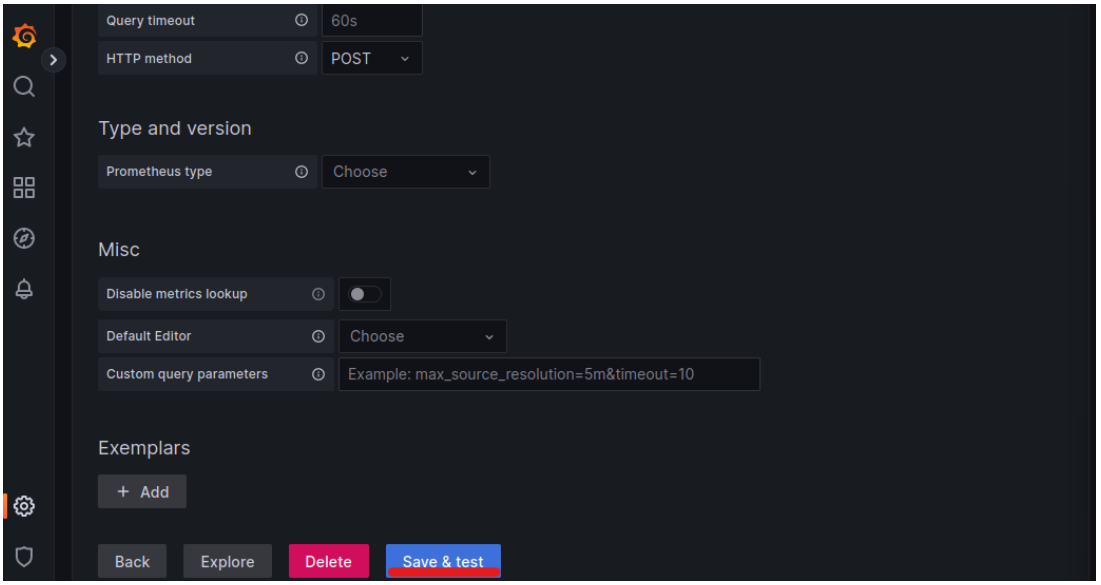
3. Στην παρούσα διπλωματική ως βάση δεδομένων χρονοσειρών χρησιμοποιήθηκε το Prometheus συνεπώς επιλέγουμε “Prometheus” από τις διαθέσιμες επιλογές.



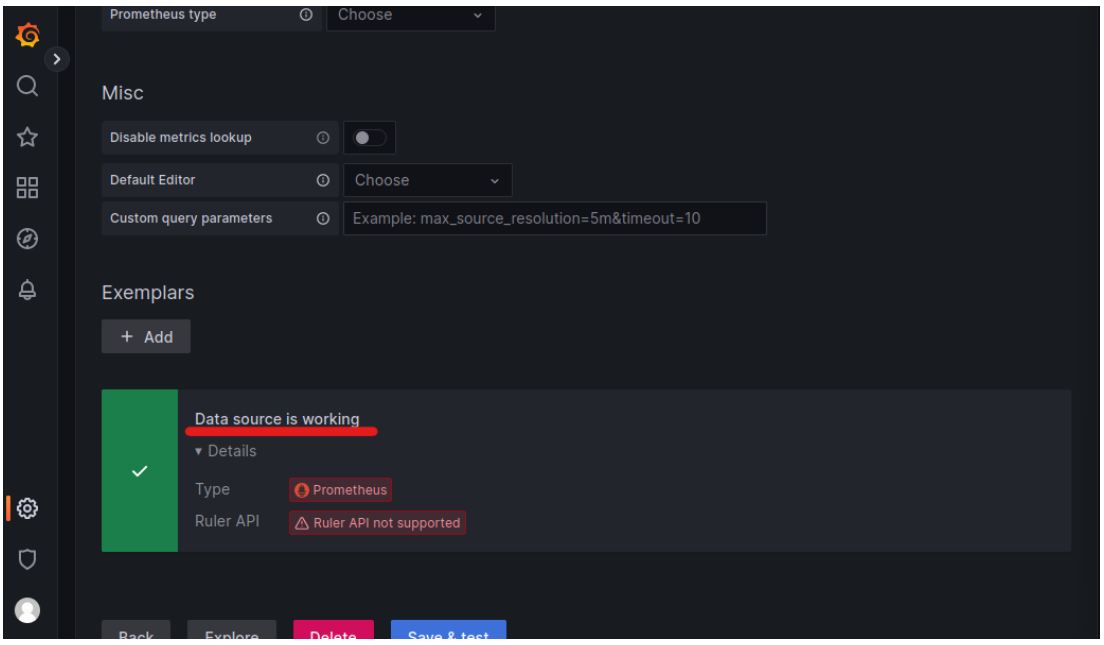
4. Στην συνέχεια ορίζουμε το URL που έχουμε δηλώσει για το Prometheus όπως φαίνεται στην εικόνα παρακάτω.



5. Αφήνουμε τις υπόλοιπες επιλογές χωρίς να τις παραμετροποιήσουμε και επιλέγουμε στο τέλος της σελίδας “Save and Test”.

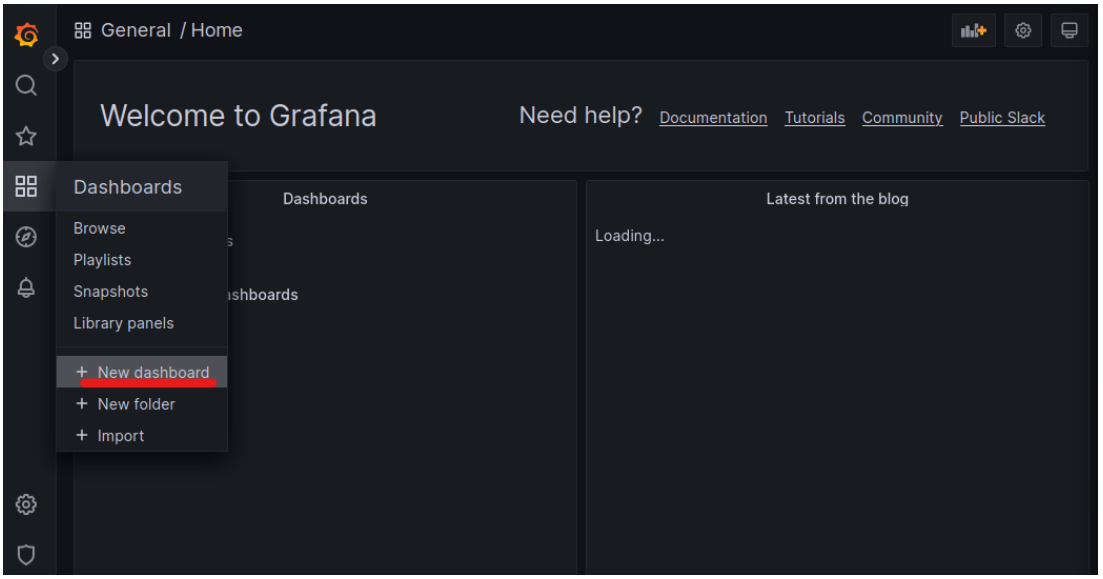


6. Θα πρέπει να εμφανιστεί το ακόλουθο μήνυμα που θα επιβεβαιώνει ότι έχει προστεθεί επιτυχώς το Prometheus ως Data Source στην Grafana.

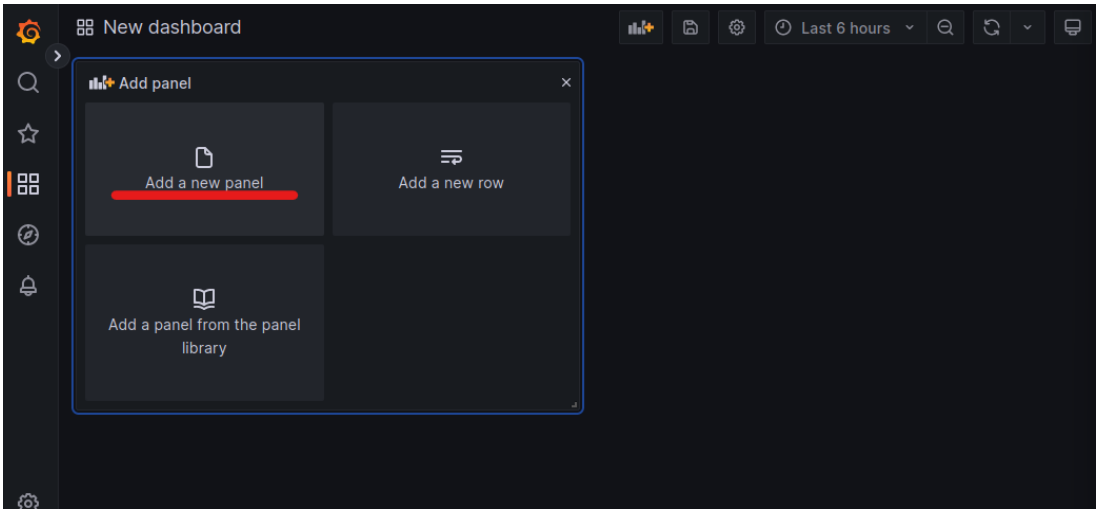


Δημιουργία ενός Dashboard

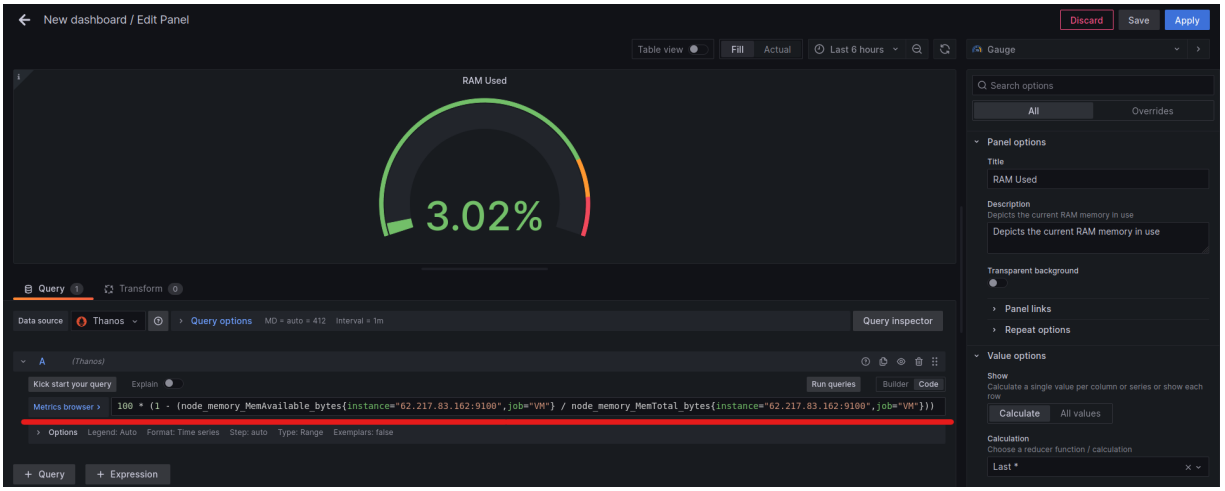
1. Στην αρχική σελίδα επιλέγουμε το εικονίδιο  και στην συνέχεια την επιλογή “New Dashboard”.



2. Στην συνέχεια δημιουργούμε τα γραφήματα που επιθυμούμε επιλέγοντας “Add a new panel”. Δείχνεται στην συνέχεια ενδεικτικά πως δημιουργήθηκε το γράφημα για την χρήση της μνήμης RAM του εικονικού μηχανήματος VM.



3. Στην συνέχεια γίνεται η παραμετροποίηση του εκάστοτε γραφήματος. Το σημαντικότερο πεδίο στην διαδικασία αυτή είναι το σημείο στο οποίο συμπληρώνεται το PromQL ερώτημα που πρέπει να γίνει όπως φαίνεται παρακάτω.



4.7 Prometheus Anomaly Detector

Το Prometheus Anomaly Detector χρησιμοποιήθηκε για να γίνονται προβλέψεις πάνω στα δεδομένα χρονοσειρών που συλλέγονται από το Prometheus. Στην συνέχεια παρουσιάζεται η παραμετροποίηση του docker-compose.yml για την δημιουργία του container του PAD.

docker-compose.yml

```
 pad:
    image: quay.io/aicoe/prometheus-anomaly-detector:latest
    container_name: pad
    environment:
      FLT_PROM_URL: "http://10.1.0.10:9002"
      FLT_RETRAINING_INTERVAL_MINUTES: 10
      FLT_METRICS_LIST:
node_network_receive_packets_total{instance="62.217.83.162:9100",job="VM"};node_n
etwork_transmit_packets_total{instance="62.217.83.162:9100",job="VM",device="ens3
"}
      APP_FILE: app.py
      FLT_ROLLING_TRAINING_WINDOW_SIZE: 15d
    restart: unless-stopped
    networks:
      monitoring-net:
        ipv4_address: 10.1.0.11
    expose:
      - 8080
    ports:
```

```
    - "8080:8080"
```

Πιο αναλυτικά:

1. Χρησιμοποιήθηκε το image *quay.io/aicoe/prometheus-anomaly-detector.*
2. Ορίστηκαν οι μεταβλητές για την σωστή λειτουργία του PAD:
    a. *FLT_PROM_URL*: Ορίστηκε το url του Prometheus.
    b. *FLT_RETRAINING_INTERVAL_MINUTES*: Ορίζεται το χρονικό διάστημα που τα δεδομένα θα εκπαιδεύονται χρησιμοποιώντας τα καινούρια δεδομένα. (10 λεπτά)
    c. *FLT_METRICS_LIST*: Ορίζεται η λίστα με metrics για το οποία θα γίνεται εκπαίδευση του μοντέλου και πρόβλεψη.
    d. *APP_FILE*: Ορίζεται το αρχείο που περιέχει την εφαρμογή του PAD.
    e. *FLT_ROLLING_TRAINING_WINDOW_SIZE*: Ορίζεται το χρονικό παράθυρο από δεδομένα που θα χρησιμοποιούνται για την εκπαίδευση του μοντέλου (15 ημέρες).
3. Τέλος, ορίστηκε η IP του container και οι πόρτες που χρησιμοποιεί.

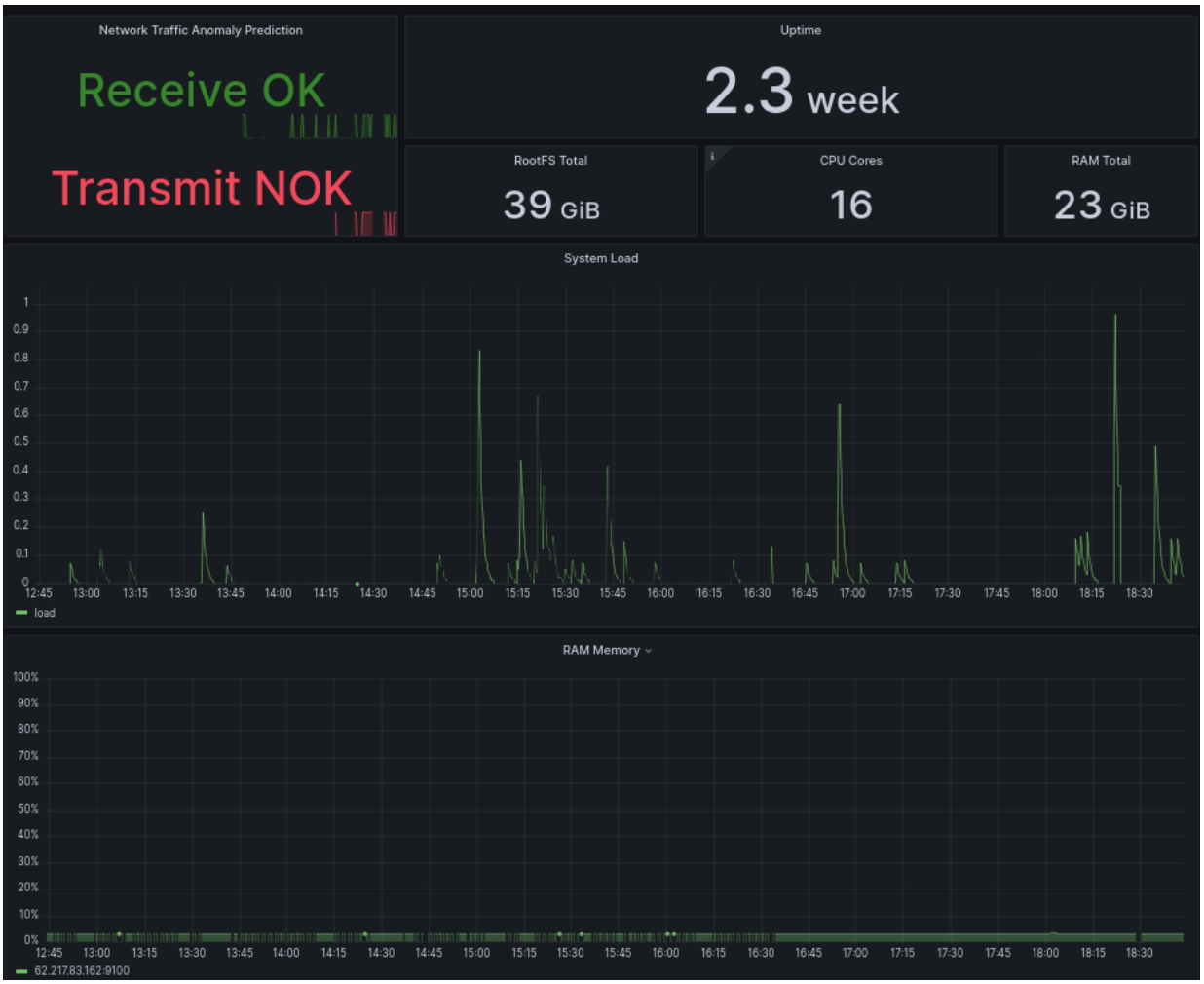
Κεφάλαιο 5

Ανάλυση Αποτελεσμάτων

5.1 Ανάλυση Γραφημάτων

Για την παρακολούθηση του εικονικού μηχανήματος δημιουργήθηκαν γραφήματα σχετικά με την χρήση της CPU, την χρήση της μνήμης RAM, τον ελεύθερο χώρο στο δίσκο και τέλος την δικτυακή κίνηση (τα πακέτα που στέλνει το εικονικό μηχάνημα καθώς και τα πακέτα που λαμβάνει το εικονικό μηχάνημα). Η χρήση του Prometheus Anomaly Detector έγινε στα metrics που αφορούν την δικτυακή κίνηση.

Εικόνα 5.1: Γραφήματα παρακολούθησης εικονικού μηχανήματος - 1



Εικόνα 5.2: Γραφήματα παρακολούθησης εικονικού μηχανήματος - 2

Για το εικονικό μηχάνημα δημιουργήθηκαν τα εξής γραφήματα με τα αντίστοιχα PromQL ερωτήματα:

1. **RAM Used**

```
100 * (1 - (node_memory_MemAvailable_bytes{instance="62.217.83.162:9100",job="VM"} / 
node_memory_MemTotal_bytes{instance="62.217.83.162:9100",job="VM"}))
```

2. **CPU Busy**

```
(sum by(instance) (irate(node_cpu_seconds_total{instance="62.217.83.162:9100",job="VM",
mode!="idle"}[$__rate_interval])) / on(instance) group_left sum by
(instance)((irate(node_cpu_seconds_total{instance="62.217.83.162:9100",job="VM"}[$__rate_i
nterval])))) * 100
```

3. **Root FS Used**

```
100 -
((node_filesystem_avail_bytes{instance="62.217.83.162:9100",job="VM",mountpoint="/",fstype
!="rootfs"} * 100) /
node_filesystem_size_bytes{instance="62.217.83.162:9100",job="VM",mountpoint="/",fstype!="
rootfs"})
```

4. **Sys Load (5m avg)**

```
avg(node_load5{instance="62.217.83.162:9100",job="VM"}) /
count(count(node_cpu_seconds_total{instance="62.217.83.162:9100",job="VM"}) by (cpu)) *
100
```

5. **Uptime**

```
node_time_seconds{instance="62.217.83.162:9100",job="VM"} -
node_boot_time_seconds{instance="62.217.83.162:9100",job="VM"}
```

6. **RootFS Total**

```
node_filesystem_size_bytes{instance="62.217.83.162:9100",job="VM",mountpoint="/",fstype!="
rootfs"}
```

7. **CPU Cores**

```
count(count(node_cpu_seconds_total{instance="62.217.83.162:9100",job="VM"}) by (cpu))
```

8. **RAM Total**

```
node_memory_MemTotal_bytes{instance="62.217.83.162:9100",job="VM"}
```

9. **CPU Analysis**

```
avg without(cpu)
(irate(node_cpu_seconds_total{instance="62.217.83.162:9100",job="VM"}[$__rate_interval]))
* 100
```

10. **System Load**

```
avg(node_load1{instance="62.217.83.162:9100",job="VM"})
```

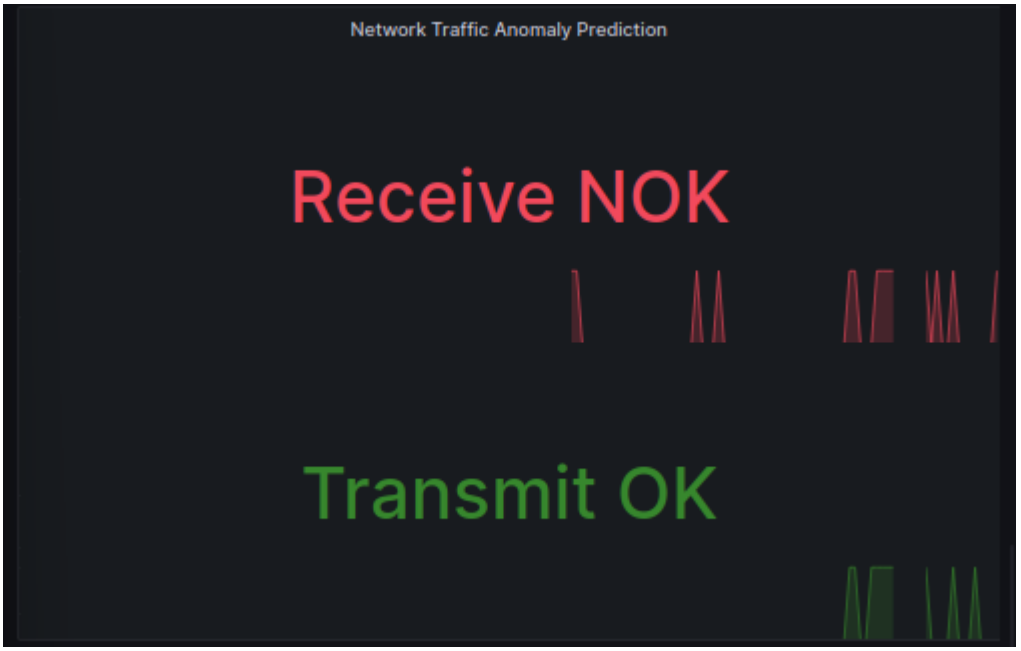
11. **Network Traffic**

```
irate(node_network_receive_packets_total{instance="62.217.83.162:9100",job="VM",device="en
s3"}[$__rate_interval])

&

irate(node_network_transmit_packets_total{instance="62.217.83.162:9100",job="VM",de
vice="ens3"}[$__rate_interval])
```

12. **RAM Memory**

```
100 * (1 - (node_memory_MemAvailable_bytes{instance="62.217.83.162:9100",job="VM"} /
node_memory_MemTotal_bytes{instance="62.217.83.162:9100",job="VM"}))
```

Όσον αφορά την πρόβλεψη στα δεδομένα χρονοσειρών της δικτυακής κίνησης, το PAD εκτός από την τιμή της πρόβλεψης για το σχετικό metric, επιστρέφει και μία τιμή η οποία ονομάζεται “anomaly” και ειναι δυαδική. Αν η τιμή πρόβλεψης είναι διαφορετική από ό,τι θεωρείται φυσιολογικό για το συγκεκριμένο εικονικό μηχάνημα τότε η τιμή “anomaly” μαρκαρεται ως True, σε αντίθετη περίπτωση μαρκάρεται ως False.
Έχοντας υπόψη τα παραπάνω δημιουργήθηκε το εξής γράφημα που ενημερώνει για το αν δικτυακή κίνηση στο εικονικό μηχάνημα προβλέπεται ότι είναι φυσιολογική ή όχι.



Εικόνα 5.3: Γραφήματα πρόβλεψη δικτυακής κίνησης

Για το παραπάνω χρησιμοποιήθηκαν τα εξής PromQL ερωτήματα:

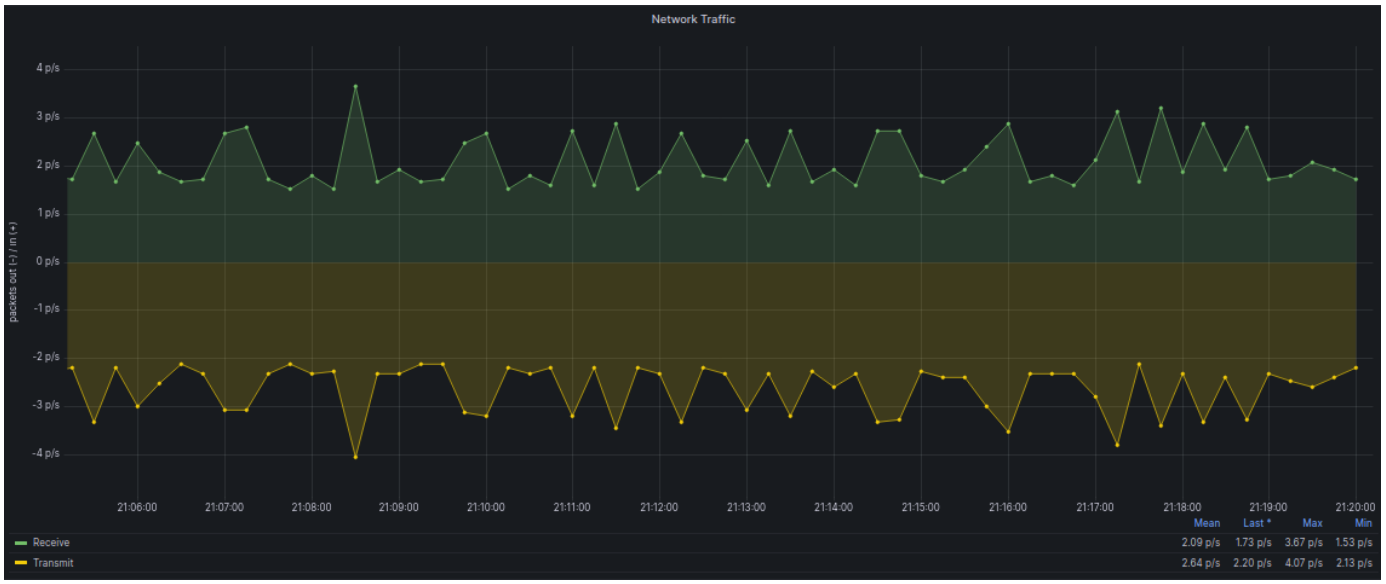
```
node_network_receive_packets_total_prophet{instance="pad:8080",device="ens3",value_type="a
nomaly",exported_instance="62.217.83.162:9100",exported_job="VM",job="PAD"}

&

node_network_transmit_packets_total_prophet{instance="pad:8080",device="ens3",value_type="
anomaly",exported_instance="62.217.83.162:9100",exported_job="VM",job="PAD"}
```

5.2 Ελεγχος High Availability / Long Term Metrics με το Thanos

Αρχικά έγινε ο έλεγχος για την συνεχή διάθεση δεδομένων (high availability) από το εικονικό μηχάνημα. Υπό κανονικές συνθήκες όλα τα container της λύσης μας είναι σε λειτουργία όπως φαίνεται στην συνέχεια.

```
~$ docker ps --format "table {{.ID}}\t{{.Image}}\t{{.Names}}\t{{.Status}}"
CONTAINER ID   IMAGE                                                NAMES                STATUS
5c0ffb7b425e   quay.io/aicoe/prometheus-anomaly-detector:latest     pad                  Up 23 minutes
33f4b57b54b0   monitoring-components_thanos_sidecar_two             thanos_sidecar_two   Up 22 minutes
ded32bef7967   monitoring-components_thanos_sidecar_one             thanos_sidecar_one   Up 22 minutes
4cef26c391ec   monitoring-components_thanos_compactor               thanos_compactor     Up 22 minutes
686104c298d7   monitoring-components_thanos_querier                 thanos_querier       Up 22 minutes
260e582b665f   monitoring-components_thanos_store                   thanos_store         Up 22 minutes
2d4a5f07e145   grafana/grafana:latest                               grafana              Up 23 minutes
8ff7dfba7d92   minio/minio:latest                                   minio                Up 22 minutes
c602018b6fe7   prom/prometheus:latest                               prometheus_two       Up 23 minutes
7f4fbd7ac545   prom/prometheus:latest                               prometheus           Up 23 minutes
```

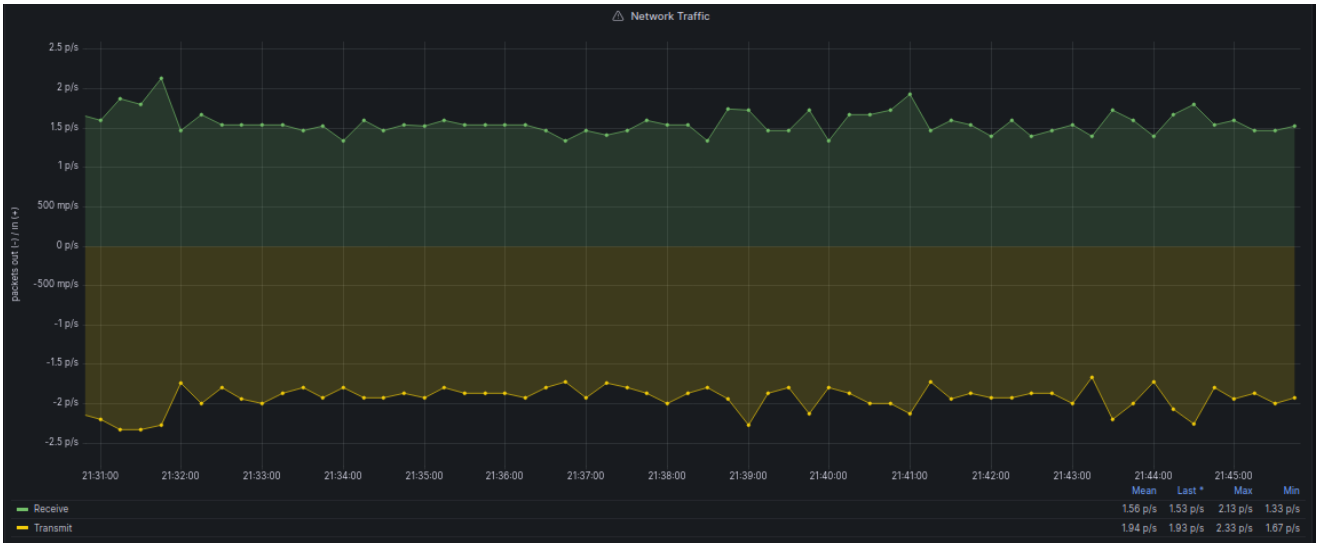
Την ίδια στιγμή, στο γράφημα της δικτυακής κίνηση στη Grafana έχουμε συνέχεια στα δεδομένα που εμφανίζονται χωρίς διακοπές. Υπενθυμίζεται ότι ως πηγή για τα δεδομένα αυτά είναι το Thanos Querier το οποίο έχει πρόσβαση και στα 2 Prometheus καθώς και στα δεδομένα που βρίσκονται στο minio.



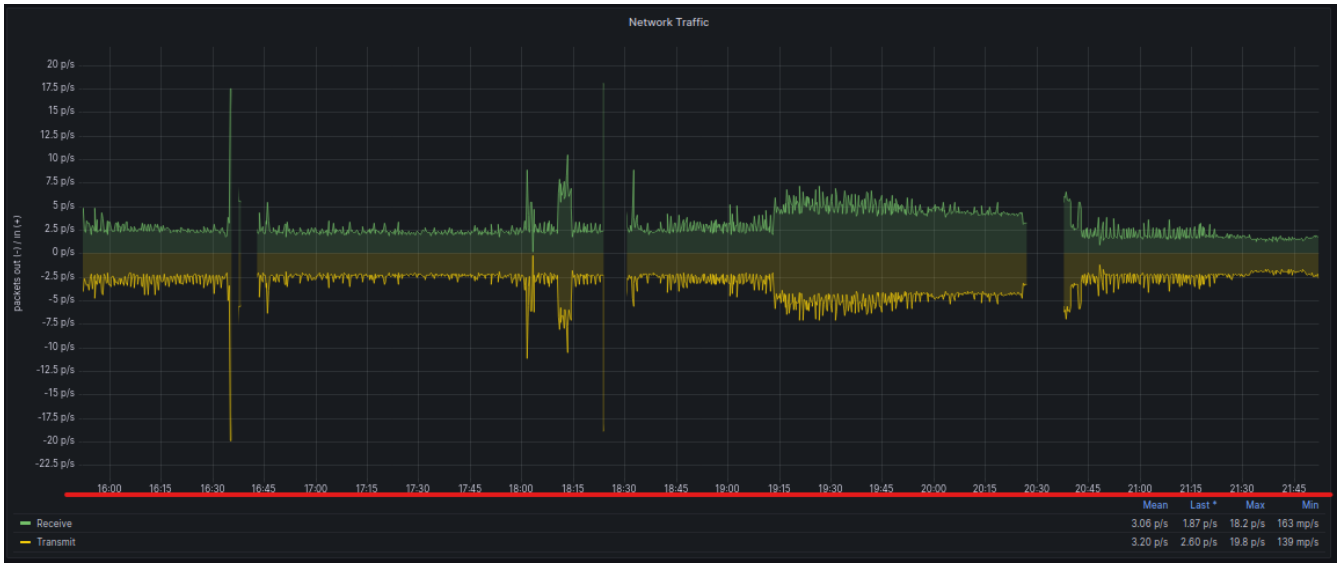
Έπειτα , διαγράφουμε το container του 2ου Prometheus χρησιμοποιώντας την εντολή:

```
~$ docker container stop prometheus_two

~$ $ docker ps --format "table {{.ID}}\t{{.Names}}\t{{.Status}}" --all | grep prometheus
CONTAINER ID   NAMES                STATUS
c602018b6fe7   prometheus_two       Exited (0) 10 minutes ago
7f4fbd7ac545   prometheus           Up About an hour
```

Παρόλα αυτά, το γράφημα συνεχίζει να έχει δεδομένα καθώς τα ερωτήματα πλέον απευθύνονται στο 1ο Prometheus με αποτέλεσμα να μην επηρεάζεται η παρακολούθηση του εικονικού μηχανήματος.



Στην συνέχεια, η επιβεβαίωση ότι υπάρχουν δεδομένα για περισσότερο καιρό από το χρονικό διάστημα που διατηρούνται από το Prometheus είναι απλή. Στην παραμετροποίηση των Prometheus containers ορίστηκε ο χρόνος για την διατήρηση των δεδομένων του Prometheus στις 2 ώρες. Παρόλα αυτά αν παρατηρηθεί το διάγραμμα της δικτυακής κίνησης για περισσότερο χρονικό διάστημα (π.χ 1 ημέρα) τότε τα δεδομένα παρουσιάζονται κανονικά όπως βλέπουμε στην συνέχεια. Το Thanos Querier επικοινωνεί με το Thanos Store και κατά συνέπεια με τα δεδομένα που είναι αποθηκευμένα στο Minio.



Κεφάλαιο 6

Επίλογος

6.1 Σύνοψη

Σκοπός της παρούσας διπλωματικής ήταν ο σχεδιασμός και η ανάπτυξη μιας υλοποίησης για την παρακολούθηση σε πραγματικό χρόνο ενός εικονικού μηχανήματος και η πρόβλεψη με την χρήση μηχανικής μάθησης κάποιας δυσλειτουργίας στο εικονικό μηχάνημα αυτό. Η υλοποίηση βασίστηκε σε λογισμικό ανοιχτού κώδικα και πιο συγκεκριμένα στα παρακάτω εργαλεία: Prometheus για την συλλογή των δεδομένων χρονοσειρών, Node Exporter για την διάθεση των δεδομένων απο το εικονικό μηχάνημα σε μορφή συμβατή με το Prometheus, Minio για την μακροχρόνια αποθήκευση των δεδομένων χρονοσειρών, Thanos για την εξασφάλιση μιας λύσης που θα προσφέρει συνεχή διαθεσιμότητα των δεδομένων, Grafana για την οπτικοποίηση των αποτελεσμάτων, Prometheus Anomaly Detector για την πρόβλεψη σε δεδομένα χρονοσειρών. Με τον τρόπο αυτό υπήρχε η δυνατότητα να υπάρχουν συνεχώς δεδομένα για την ομαλή ή όχι λειτουργία του εικονικού μηχανήματος και ο χρήστης να είναι ενήμερος για το πότε το εικονικό μηχάνημα τείνει να έχει μια μη αναμενόμενη συμπεριφορά. Η αξία του εγχειρήματος είναι ιδιαίτερα σημαντική καθώς όλοι οι οργανισμοί πλέον, λόγω της πολυπλοκότητας και του μεγέθους των υπολογιστικών συστημάτων που χρησιμοποιούν, θέλουν να έχουν πλήρη εποπτεία για την ομαλή λειτουργία τους και γιατί για την πρόβλεψη προβλημάτων και έγκαιρη αντιμετώπιση αυτών. Η ομαλή λειτουργία των εικονικών μηχανημάτων εξασφαλίζει σε ένα μεγάλο βαθμό την ομαλή λειτουργία της εφαρμογής που φιλοξενείται σε αυτά, με αυτό τον τρόπο διασφαλίζεται η επιχειρηματική δραστηριότητα και η ακεραιότητα ενός οργανισμού.

6.2 Μελλοντικές επεκτάσεις

Στην παρούσα διπλωματική, ο σχεδιασμός και η υλοποίηση βασίστηκε στην παρακολούθηση ενός μόνο εικονικού μηχανήματος και της συνολικής αρχιτεκτονικής και στην πρόβλεψη κάποιον δεδομένων που αφορούσαν το εικονικό μηχάνημα. Για την βελτίωση και την επέκταση της υλοποίησης αυτής θα μπορούσαν να προστεθούν περισσότεροι στόχοι για την παρακολούθηση από την λύση, όπως για παράδειγμα περισσότερα εικονικά μηχανήματα, φυσικοί εξυπηρετητές (servers), δικτυακές συσκευές (firewalls, switches, load balancers) αλλά και οι ίδιες οι εφαρμογές με την χρήση επιπλέον exporters που υπάρχουν διαθέσιμοι ή ακόμα και με την δημιουργία ειδικών προσαρμοσμένων exporters από τον εκάστοτε οργανισμό. Παράλληλα θα μπορούσαν να προστεθούν επιπλέον δεδομένα (metrics), είτε από το εικονικό μηχάνημα είτε από άλλα

συστήματα, που θα γινόταν πρόβλεψη πάνω σε αυτά. Με τον τρόπο αυτό θα υπήρχε μια πιο ολοκληρωμένη εικόνα για ολόκληρη την υποδομή. Επιπλέον θα μπορούσε η αρχιτεκτονική της λύσης να επεκταθεί ως προς τις δυνατότητες της. Μία από αυτές είναι η διασύνδεση της λύσης με άλλες εφαρμογές και αυτοματισμούς ενός οργανισμού για την λήψη αποφάσεων μέσω των ειδοποιήσεων που μπορεί να στέλνει το σύστημα σε περίπτωση προβλήματος. (αύξηση του χώρου στο δίσκο σε ένα εικονικό μηχάνημα μέσω μιας αυτοματοποίησης μόλις λάβει ειδοποίηση ότι προβλέφθηκε μείωση του διαθέσιμου χώρου στο εικονικό μηχάνημα). Τέλος η λύση προσφέρει διαθεσιμότητα των δεδομένων για μεγάλο χρονικό διάστημα με την χρήση του Minio, παρόλα αυτά δεν είναι πλήρως highly available λύση καθώς χρησιμοποιείται μόνο ένα container για την υλοποίηση του. Για την υλοποίηση του Minio σε ένα παραγωγικό περιβάλλον μεγάλης υποδομής για να εξυπηρετηθεί η συνεχής αποθήκευση των δεδομένων και για να αποφευχθεί η απώλεια δεδομένων προτείνεται να υλοποιηθεί σε εικονικά μηχανήματα, πιο συγκεκριμένα σε ένα ελάχιστο cluster τεσσάρων (4) VM με 4 δίσκους το καθένα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Site Reliability Engineering, edited by Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy (O'Reilly). (Πρόσβαση Μάρτιος 2022)
2. https://prometheus.io/docs/introduction/overview/ (Πρόσβαση Μάρτιος 2022)
3. https://github.com/prometheus/prometheus (Πρόσβαση Μάρτιος 2022)
4. https://github.com/prometheus/pushgateway (Πρόσβαση Μάρτιος 2022)
5. https://prometheus.io/docs/instrumenting/exporters/ (Πρόσβαση Μάρτιος 2022)
6. https://github.com/prometheus/node_exporter (Πρόσβαση Μάρτιος 2022)
7. https://devconnected.com/the-definitive-guide-to-prometheus-in-2019/ (Πρόσβαση Μάρτιος 2022)
8. https://grpc.io/docs/what-is-grpc/introduction/ (Πρόσβαση Μάρτιος 2022)
9. https://thanos.io/tip/thanos/quick-tutorial.md/ (Πρόσβαση Μάρτιος 2022)
10. https://thanos.io/tip/components/sidecar.md/ (Πρόσβαση Μάρτιος 2022)
11. https://thanos.io/tip/components/query.md/ (Πρόσβαση Μάρτιος 2022)
12. https://thanos.io/tip/components/store.md/ (Πρόσβαση Μάρτιος 2022)
13. https://thanos.io/tip/components/compact.md/ (Πρόσβαση Μάρτιος 2022)
14. https://www.youtube.com/playlist?list=PLFOIsHSSYIK3WitnqhqfpeZ6fRFKHxIr7 (Πρόσβαση Μάρτιος 2022)
15. https://grafana.com/docs/ (Πρόσβαση Μάρτιος 2022)
16. https://github.com/AICoE/prometheus-anomaly-detector (Πρόσβαση Μάρτιος 2022)
17. https://docs.docker.com/ (Πρόσβαση Μάρτιος 2022)
18. https://medium.com/data-science-engineering/using-time-series-forecasting-library-prophet-for-anomaly-detection-55fe36588f2f (Πρόσβαση Μάρτιος 2022)

19. https://facebook.github.io/prophet/ (Πρόσβαση Μάρτιος 2022)
20. https://medium.com/@mail2ramunakerikanti/thanos-for-prometheus-f7f111e3cb75 (Πρόσβαση Μάρτιος 2022)
21. https://www.youtube.com/watch?v=YUabB_7H710 (Πρόσβαση Μάρτιος 2022)
22. https://coralogix.com/blog/scale-your-prometheus-metrics-indefinitely-with-thanos/ (Πρόσβαση Μάρτιος 2022)
23. https://min.io/docs/minio/container/index.html (Πρόσβαση Μάρτιος 2022)
24. https://min.io/docs/minio/linux/operations/install-deploy-manage/deploy-minio-multi-node-multi-drive.html#deploy-minio-distributed (Πρόσβαση Μάρτιος 2022)