Ζάκυνθος 29/08/2022
Συγγραφέας Γεώργιος Μυλωνάς

Παραδοτέο 2 (Τμήμα Β)

Στα πλαίσια Σύμβασης Έργου με Αρ. Πρωτοκόλλου **80383/24806/β1.α**

Περιγραφή Δράσης

Ασφάλεια και διαχείριση δεδομένων, οργάνωση και διαδικτυακή διαθεσιμότητα της βάσης δεδομένων που τροφοδοτείται από τις συνεχώς παραγόμενες μετρήσεις του δικτύου σταθμών, των αριθμητικών αποτελεσμάτων που παράγονται από την Μαθηματική και Στατιστική τους επεξεργασία, και των δεδομένων περιβαλλοντικού κινδύνου που θα τροφοδοτούνται σε πραγματικό χρόνο, στα πλαίσια του ΠΕ 2.1.1 : Επιχειρησιακή διάγνωση Μετεωρολογικών συνθηκών σε πραγματικό χρόνο"

Στα πλαίσια του υποέργου (με κωδικό ΕΛΚΕ Ιονίου Πανεπιστημίου 80383) “Τρέχουσες Μετεωρολογικές Συνθήκες, Κλιματική Μεταβλητότητα, και Κίνδυνος Δασικών Πυρκαγιών στα Επτάνησα”, που αποτελεί διακριτό υποέργο της Πράξης "ΛΑΕΡΤΗΣ - ΚΑΙΝΟΤΟΜΟ ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΣΥΣΤΗΜΑ ΔΙΑΧΕΙΡΙΣΗΣ ΦΥΣΙΚΩΝ ΚΙΝΔΥΝΩΝ ΣΤΗΝ ΠΕΡΙΦΕΡΕΙΑ ΙΟΝΙΩΝ ΝΗΣΩΝ” MIS 5010951 / ΕΣΠΑ 2014-2020

Περιεχόμενα

1 Περιγραφή Έργου

Στα πλαίσια της σχετικής με το υποέργο 2: “Επιχειρησιακή διάγνωση Μετεωρολογικών συνθηκών σε πραγματικό χρόνο" της πράξης ΛΑΕΡΤΗΣ / MIS 5010951 σύμβασης 80383/24806/β1.α, μελετήθηκε και θα αναβαθμίστηκε το διαδικτυακό σύστημα “IonianWeather” του δικτύου 14 υπαίθριων επιστημονικών (Μετεωρολογικών – Περιβαλλοντικών) σταθμών υπαίθρου που το Εργαστήριο Φυσικής Περιβάλλοντος, Ενέργειας, και Περιβαλλοντικής Βιολογίας του Τμήματος Περιβάλλοντος του Ιονίου Πανεπιστημίου διαθέτει κατά μήκος των Επτανήσων.

Στόχοι της αναβάθμισης είναι οι εξής:

1. Βελτιστοποίηση της ασφάλειας του διαδικτυακού συστήματος IonianWeather
2. Μείωση της έλλειψης δεδομένων από τυχών βλάβες των παρελκόμενων συστημάτων
3. Βελτιστοποίηση της ικανότητας της συνεχής λειτουργίας του διαδικτυακού συστήματος IonianWeather

Βασιζόμενοι στους προηγούμενους στόχους και στην μελέτη που εκπονήθηκε (όπως παρουσιάστηκε στο πρώτο τμήμα του παραδοτέου) υλοποιήθηκαν και αναβαθμίστηκαν οι παρακάτω μονάδες και υποδομές της πλατφόρμας

1. Ανάπτυξη περιβάλλον λειτουργίας και φιλοξενίας του διαδικτυακού συστήματος IonianWeather με τεχνολογία DOCKER
2. Εγκατάσταση μονάδων και πιστοποιητικών διασφάλισης μετάδοσης και λήψης δεδομένων τύπου HTTPS
3. Εξασφάλιση βέλτιστης προστασίας από κακόβουλες επιθέσεις πχ. spyware, ddos attacks κ.α.
4. Εγκατάσταση/Παραμετροποίηση ασφάλειας περιβάλλοντος λειτουργίας

διαδικτυακού συστήματος IonianWeather

5. Ανάπτυξη μονάδος παρακολούθησης και καταγραφής ενεργειών διασυνδεδεμένων χρηστών. Η συγκεκριμένη μονάδα αναπτύχθηκε βάση των προτύπων όπως καθορίζονται από τους κανονισμούς GDPR
6. Εγκατάσταση/Παραμετροποίηση μονάδος παρακολούθησης της ορθής λειτουργίας του διαδικτυακό σύστημα IonianWeather. Η συγκεκριμένη μονάδα διαθέτει μηχανισμούς έγκαιρης ενημέρωσης του λογισμικού Diameson το οποίο αποτελεί την γέφυρα μεταφοράς δεδομένων από τους μετεωρολογικούς σταθμούς.
7. Ανάπτυξη διαδικασίας διασφάλισης του κώδικα του έργου όσον αφορά την μελλοντική χρήση και επεκτασιμότητα του.

Στα επόμενα κεφάλαια παρουσιάζεται η μεθοδολογία και το πλάνο των εργασιών που ακολουθήθηκε για την πραγματοποίηση των προαναφερθέντων εργασιών.

2 Βασικά χαρακτηριστικά του δικτύου επιστημονικών (Μετεωρολογικών – Περιβαλλοντικών) σταθμών υπαίθρου IonianWather

Σύμφωνα με την μελέτη του προηγούμενου παραδοτέου 1 τα βασικά χαρακτηριστικά του δικτύου είναι τα εξής

- Παράμετροι που καταγράφονται από τα όργανα του επιχειρισιακού δικτύου Ionian Weather είναι οι εξής:
  - Μέση Ταχύτητα και Κατεύθυνση Ανέμου
  - Ριπή Ανέμου
  - Βροχόπτωση
  - Θερμοκρασία
  - Υγρασία
  - Πίεση
  - Ηλιακή Ακτινοβολία
  - Υπεριώδης Ακτινοβολία
  - Αιωρούμενα Σωματίδια
- Γεωγραφική διάταξη σταθμών δικτύου:
  - ΚΕΡΚΥΡΑ : Αυλιώτες (CRF-1), Τεμπλόνι (CRF-2), Λίμνη Κορισίων (CRF-3), πόλη Κέρκυρας (CRF-A)
  - ΠΑΞΟΙ : Αγ. Ισαυρος (PAX-1)
  - ΛΕΥΚΑΔΑ : Λιμνοθάλασσα Λευκάδας (LFK-1)
  - ΚΕΦΑΛΟΝΙΑ : Αντυπάτα Ερισού (KEF-1), Κηπούρια Παλικής (KEF-2), Σκάλα Πόρου (KEF-3)
  - ΖΑΚΥΝΘΟΣ : Αγαλάς (ZKT-1), Αεροδρόμιο (ZKT-2), Σκινάρι (ZKT-3), την πόλη Ζακύνθου (ZKT-A)
  - ΗΛΕΙΑ : Κατάκολο (KTL-1)

3 Ανάπτυξη περιβάλλοντος φιλοξενίας του συστήματος με τεχνολογία Docker

Όπως παρουσιάστηκε στο εισαγωγικό κείμενο, 2 από τους βασικούς στόχους αυτού του έργου είναι η ασφαλής λειτουργία διαδικτυακού συστήματος IonianWeather και η βελτιστοποίηση της συνεχής λειτουργίας του.

Στα πλαίσια αυτού του έργου κρίθηκε ότι η τεχνολογία Docker μπορεί να εξυπηρετήσει και τους δυο στόχους. Όπως θα παρουσιάσουμε στις επόμενες παραγράφους, η τεχνολογία Docker μπορεί να παρέχει ασφαλή "κυτία" (Containers) τα οποία έχουν την δυνατότητα να φιλοξενήσουν τις μονάδες που απαρτίζουν το διαδικτυακό σύστημα IonianWeather τα οποία το απομονώνουν από διάφορους τύπους επιθέσεων.

Επίσης η τεχνολογία Docker μπορεί να παρέχει "εικόνες" (images) του περιβάλλοντος λειτουργίας διαδικτυακού συστήματος IonianWeather προς άμεση εγκατάσταση σε νέον υπολογιστή. Αυτό επιτρέπει να μειωθεί ο χρόνος επαναλειτουργίας του συστήματος σε περίπτωση που υπάρξει βλάβη στον υπολογιστή που φιλοξενεί το διαδικτυακό σύστημα IonianWeather

3.1 Βασικές Έννοιες

Όπως είχε παρουσιαστεί στα πορίσματα του 1ου παραδοτέου, η έννοια του containerization ("εγκλεισμού") υπάρχει εδώ και πολύ καιρό. Το containerized λογισμικό, παρέχει την δυνατότητα διανομής του λογισμικού χωρίς εξαρτήσεις από τα τεχνικά χαρακτηριστικά του περιβάλλοντος που θα εγκατασταθεί. Αυτό δίνει τη δυνατότητα στους δημιουργούς λογισμικού (δηλαδή προγραμματιστές και οργανισμούς) να δημιουργία και να διανέμουν πακέτα που δεν έχουν εξαρτήσεις. Αυτό σημαίνει ότι αν θέλουμε να τρέξουμε μια συγκεκριμένη εφαρμογή, χρειάζεται μόνο να κατεβάσουμε το πακέτο που οι προγραμματιστές της εφαρμογής έχουν δημιουργήσει.

Το συγκεκριμένο έργο, βάση προδιαγραφών έχει την ανάγκη της

απρόσκοπτης λειτουργίας και συνεχής λειτουργίας. Σε περίπτωση μηχανικής βλάβης (hardware failure) υπάρχει σοβαρός κίνδυνος απώλειας δεδομένων. Εφαρμόζοντας την τεχνολογία Docker στο περιβάλλον του Διαδικτυακού Συστήματος Ionian Weather, μας δίνεται η δυνατότητα της άμεσης επαναλειτουργίας του συστήματος σε νέο υπολογιστή μέσα σε σύντομο χρόνο.

Το Docker αποτελείται από μερικές έννοιες: δαίμονες, εικόνες, containers και Dockerfiles.

3.1.1 Υπηρεσίες παρασκηνίου ("Δαίμονες")

Οι υπηρεσίες παρασκηνίου μπορούν να συμμετάσχουν στην "αυτό επιδιόρθωση" του διαδικτυακού συστήματος IonianWeather σε περιπτώσεις που το σύστημα έχει την ανάγκη αναγκαστικής επανεκκίνησης.

Ο δαίμονας είναι ένας τύπος προγράμματος υπολογιστή που εκτελείται στο παρασκήνιο, συνήθως ως διεργασία παρασκηνίου, αντί να βρίσκεται στο προσκήνιο και να είναι ορατός στο χρήστη. Οι δαίμονες χρησιμοποιούνται συχνά για να εκτελούν εργασίες που δεν βλέπουν το χρήστη, όπως η διαχείριση των πόρων του συστήματος, η εκτέλεση επεξεργασίας στο παρασκήνιο ή η παροχή υπηρεσιών σε άλλα προγράμματα.

Ένας δαίμονας είναι συνήθως σχεδιασμένος να εκτελείται συνεχώς, εκτελώντας τις καθορισμένες εργασίες του με συνεχή τρόπο, ακόμη και όταν ο χρήστης δεν αλληλεπιδρά με το σύστημα. Αυτό έρχεται σε αντίθεση με άλλους τύπους προγραμμάτων, όπως οι εφαρμογές γραφικής διεπαφής χρήστη (GUI), οι οποίες εκκινούνται και εκτελούνται μόνο όταν ο χρήστης τις ζητήσει ρητά.

Ο δαίμονας είναι μια υπηρεσία (ένα πρόγραμμα με δικαιώματα που εκτελείται στο παρασκήνιο) που εκτελείται (ως root) στον κεντρικό υπολογιστή. Διαχειρίζεται ό,τι σχετίζεται με το Docker σε αυτό το μηχάνημα. Για παράδειγμα, εάν ένας χρήστης χρειάζεται να επανεκκινεί ένα container, τότε ο δαίμονας του Docker είναι η διαδικασία που επανεκκινεί το container. Αυτό είναι αξιοσημείωτο, καθώς όλα όσα σχετίζονται με το Docker τα χειρίζεται ο δαίμονας και το Docker έχει

πρόσβαση σε όλους τους πόρους του κεντρικού υπολογιστή (επειδή εκτελείται ως root), συνεπώς η δυνατότητα χρήσης του Docker ισοδυναμεί με την πρόσβαση root στον κεντρικό υπολογιστή.

Στην περίπτωση του συγκεκριμένου έργου οι "δαίμονες" ελέγχονται από προγραμματισμένες διεργασίες του συστήματος έτσι ώστε να παραμένει σε συνεχή λειτουργία το περιβάλλον Docker

3.1.2 Εικόνες (Docker Images)

Μια εικόνα Docker είναι ένα στιγμιότυπο ενός περιβάλλοντος λογισμικού που έχει ληφθεί σε μια συγκεκριμένη χρονική στιγμή. Οι εικόνες Docker είναι τα δομικά στοιχεία των κυτίων Docker και περιέχουν όλα όσα απαιτούνται για την εκτέλεση ενός λογισμικού, συμπεριλαμβανομένου του κώδικα, του χρόνου εκτέλεσης, των βιβλιοθηκών, των μεταβλητών περιβάλλοντος και των εργαλείων συστήματος. Οι εικόνες Docker δημιουργούνται χρησιμοποιώντας ένα Dockerfile, το οποίο είναι ένα σενάριο που περιέχει ένα σύνολο οδηγιών που αυτοματοποιούν τη διαδικασία δημιουργίας μιας εικόνας.

Οι εικόνες Docker έχουν σχεδιαστεί ώστε να είναι φορητές, πράγμα που σημαίνει ότι μπορούν να τρέξουν σε οποιοδήποτε σύστημα που υποστηρίζει την πλατφόρμα Docker, χωρίς να χρειάζεται να γίνουν αλλαγές στις ρυθμίσεις. Αυτό καθιστά εύκολη τη μετακίνηση των εικόνων από ένα σύστημα σε άλλο, γεγονός που τις καθιστά ιδανική λύση για υπολογιστικό νέφος, όπου οι πόροι συχνά παρέχονται και αφαιρούνται κατά παραγγελία.

Οι εικόνες Docker είναι επίσης εκδόσεις, πράγμα που σημαίνει ότι πολλαπλές εκδόσεις μιας εικόνας μπορούν να αποθηκευτούν και να διαχειριστούν σε ένα μητρώο Docker. Αυτό καθιστά δυνατή την επαναφορά σε μια προηγούμενη έκδοση μιας εικόνας, εάν είναι απαραίτητο, ή τη διατήρηση διαφορετικών εκδόσεων μιας εικόνας για διαφορετικές περιπτώσεις χρήσης.

Ένα πλεονέκτημα της χρήσης εικόνων Docker είναι ότι είναι εύκολη η κοινή χρήση τους. Οι εικόνες Docker μπορούν να μεταφορτωθούν ένα κεντρικό

αποθετήριο όπου οι εικόνες μπορούν να αποθηκευτούν και να έχουν απομακρυσμένη πρόσβαση. Αυτό δίνει τη δυνατότητα στους προγραμματιστές να μοιράζονται εικόνες με άλλους, διευκολύνοντας τη συνεργασία σε έργα ανάπτυξης λογισμικού.

Συμπερασματικά, οι εικόνες Docker είναι ένα σημαντικό και ισχυρό εργαλείο για την κατασκευή και την ανάπτυξη εφαρμογών λογισμικού. Είναι φορητές, εκδόσεις, προσαρμόσιμες και εύκολες στην κοινή χρήση, καθιστώντας τις ιδανική λύση για ένα ευρύ φάσμα περιπτώσεων χρήσης, όπως υπολογιστικό νέφος, αρχιτεκτονικές μικρουπηρεσιών και δοκιμές και ανάπτυξη. Με τη σωστή γνώση και εμπειρία, οι εικόνες Docker μπορούν να αποτελέσουν πολύτιμο πόρο για την αυτοματοποίηση της ανάπτυξης και της διάθεσης εφαρμογών λογισμικού.

Στην περίπτωση του συγκεκριμένου έργου έχει παραχθεί Docker Image έτσι ώστε σε περίπτωση κρίσης όπου απαιτείται η ενεργοποίηση νέου υπολογιστή, θα χρησιμοποιηθεί για την άμεση επαναλειτουργία του συστήματος.

3.1.3 Κυτία "Containers"

Ένα κυτίο Docker (Docker container) είναι ένα αυτόνομο εκτελέσιμο πακέτο που περιλαμβάνει όλα όσα απαιτούνται για την εκτέλεση ενός λογισμικού, συμπεριλαμβανομένου του κώδικα, του χρόνου εκτέλεσης, των βιβλιοθηκών, των μεταβλητών περιβάλλοντος και των εργαλείων συστήματος. Τα κυτία Docker δημιουργούνται από εικόνες Docker, οι οποίες είναι στιγμιότυπα ενός περιβάλλοντος λογισμικού που έχουν ληφθεί σε μια συγκεκριμένη χρονική στιγμή. Οι εικόνες δημιουργούνται χρησιμοποιώντας ένα Dockerfile, το οποίο είναι ένα σενάριο που περιέχει ένα σύνολο οδηγιών που αυτοματοποιούν τη διαδικασία δημιουργίας μιας εικόνας.

Τα κυτία Docker έχουν σχεδιαστεί ώστε να είναι φορητά, πράγμα που σημαίνει ότι μπορούν να τρέξουν σε οποιοδήποτε σύστημα που υποστηρίζει την πλατφόρμα Docker, χωρίς να χρειάζεται να γίνουν αλλαγές στις ρυθμίσεις. Αυτό καθιστά εύκολη τη μετακίνηση των κυτίων από το ένα σύστημα στο άλλο, γεγονός

που τα καθιστά ιδανική λύση για υπολογιστικό νέφος, όπου οι πόροι συχνά παρέχονται και αφαιρούνται κατά παραγγελία.

Ακόμα είναι απομονωμένα από το λειτουργικό σύστημα του κεντρικού υπολογιστή και από άλλα κυτία, πράγμα που σημαίνει ότι κάθε κυτίο εκτελείται στο δικό του απομονωμένο περιβάλλον. Αυτό παρέχει μια σειρά από πλεονεκτήματα, συμπεριλαμβανομένης της καλύτερης ασφάλειας, καθώς το εμπορευματοκιβώτιο προστατεύεται από άλλες διεργασίες που εκτελούνται στο κεντρικό σύστημα. Διευκολύνει επίσης τη διαχείριση τους, καθώς κάθε κυτίο μπορεί να ενημερώνεται και να διαχειρίζεται ανεξάρτητα από άλλα και από το λειτουργικό σύστημα του κεντρικού υπολογιστή.

Επίσης τα κυτία Docker είναι επίσης ελαφριά, γεγονός που τα καθιστά ιδανικά για χρήση σε αρχιτεκτονικές μικρουπηρεσιών, όπου πολλαπλά εμπορευματοκιβώτια χρησιμοποιούνται για τη δημιουργία και τη λειτουργία μιας μεγάλης, σύνθετης εφαρμογής. Κάθε εμπορευματοκιβώτιο μπορεί να επικεντρωθεί σε μία μόνο, συγκεκριμένη εργασία, καθιστώντας εύκολη τη διαχείριση και τη συντήρηση της εφαρμογής.

Ένα από τα κύρια πλεονεκτήματα της χρήσης των κυτίων Docker είναι ότι μπορούν να δημιουργηθούν και να καταστραφούν γρήγορα και εύκολα. Αυτό καθιστά δυνατή την ταυτόχρονη εκτέλεση και δοκιμή πολλαπλών εκδόσεων μιας εφαρμογής ή τη γρήγορη κλιμάκωση μιας εφαρμογής για να ανταποκριθεί στην αυξημένη ζήτηση. Αυτό οφείλεται στο γεγονός ότι τα containers δημιουργούνται από εικόνες, οι οποίες είναι στιγμιότυπα ενός περιβάλλοντος λογισμικού που έχουν καταγραφεί σε μια συγκεκριμένη χρονική στιγμή. Αυτό σημαίνει ότι τα κοντέινερ μπορούν να δημιουργηθούν γρήγορα από την εικόνα και ότι η εικόνα μπορεί να ενημερωθεί με νέο κώδικα και να αναπτυχθεί στα κοντέινερ, χωρίς να χρειάζεται να ξαναχτιστεί ολόκληρη η εφαρμογή.

Ένα άλλο πλεονέκτημα της χρήσης των κυτίων Docker είναι ότι είναι εξαιρετικά προσαρμόσιμα. Το περιβάλλον στο εσωτερικό ενός εμπορευματοκιβωτίου μπορεί να προσαρμοστεί στις συγκεκριμένες απαιτήσεις

μιας εφαρμογής, καθιστώντας δυνατή την εκτέλεση πολλαπλών εκδόσεων μιας εφαρμογής, η καθεμία με διαφορετικό περιβάλλον, στον ίδιο κεντρικό υπολογιστή.

Συμπερασματικά, τα κυτία Docker είναι ένα σημαντικό και ισχυρό εργαλείο για την κατασκευή και την ανάπτυξη εφαρμογών λογισμικού. Είναι φορητά, απομονωμένα, ελαφριά και προσαρμόσιμα, γεγονός που τα καθιστά ιδανική λύση για ένα ευρύ φάσμα περιπτώσεων χρήσης, όπως η υπολογιστική νέφους, οι αρχιτεκτονικές μικροϋπηρεσιών και οι δοκιμές και η ανάπτυξη. Με τη σωστή γνώση και εμπειρία, μπορούν να αποτελέσουν πολύτιμο πόρο για την αυτοματοποίηση της ανάπτυξης και της διάθεσης εφαρμογών λογισμικού.

Στην περίπτωση του συγκεκριμένου έργου έχουν υλοποιηθεί 3 containers:

1. **Mysql container για την φιλοξενία της βάσης δεδομένων**
2. **Application container για την φιλοξενία και λειτουργία του διαδικτυακού συστήματος Ionian Weather**
3. **Node.js container για την φιλοξενία και λειτουργία εξειδικευμένων λογισμικών που αναπτύχθηκε στα πλαίσια αυτού του έργου.**

3.1.4 Dockerfiles

Ένα αρχείο Dockerfile είναι ένα σενάριο που περιέχει ένα σύνολο οδηγιών που αυτοματοποιούν τη διαδικασία δημιουργίας μιας εικόνας Docker. Αυτές οι εικόνες χρησιμοποιούνται στη συνέχεια για τη δημιουργία κυτίων Docker. Δημιουργείται σε απλή μορφή κειμένου και περιλαμβάνει μια σειρά εντολών, καθεμία από τις οποίες καθορίζει μια συγκεκριμένη ενέργεια που πρέπει να εκτελεστεί κατά τη διαδικασία δημιουργίας της εικόνας. Το Dockerfile πρέπει να είναι γραμμένο με συγκεκριμένη σύνταξη και σειρά, καθώς κάθε εντολή εκτελείται διαδοχικά και βασίζεται στην προηγούμενη εντολή.

Η χρήση ενός Dockerfile είναι ο πιο αποτελεσματικός τρόπος για τη δημιουργία μιας εικόνας Docker, καθώς αυτοματοποιεί τη διαδικασία και εξαλείφει την ανάγκη για χειροκίνητη παρέμβαση. Το αρχείο Docker μπορεί να ελεγχθεί ως

προς την έκδοση και να μοιραστεί εύκολα με άλλους, καθιστώντας το έναν βολικό τρόπο για την τυποποίηση του περιβάλλοντος ανάπτυξης σε διαφορετικές ομάδες και έργα.

Ένα από τα μεγαλύτερα πλεονεκτήματα της χρήσης των Dockerfiles είναι η δυνατότητα εύκολης δημιουργίας και διαχείρισης κυτία. Με τα Dockerfiles, μπορείτε να ορίσετε ένα επαναλαμβανόμενο, συνεπές και προβλέψιμο περιβάλλον για τις εφαρμογές σας, το οποίο βοηθά στην ελαχιστοποίηση των προβλημάτων συμβατότητας και των προβλημάτων διαμόρφωσης.

Ένα άλλο πλεονέκτημα της χρήσης των Dockerfiles είναι η δυνατότητα γρήγορης και εύκολης δημιουργίας νέων κυτίων από μια υπάρχουσα εικόνα. Αυτό μπορεί να είναι ιδιαίτερα χρήσιμο όταν εργάζεστε με μεγάλες, σύνθετες εφαρμογές, καθώς μπορείτε να δημιουργήσετε νέα εμπορευματοκιβώτια με τις τελευταίες ενημερώσεις και αλλαγές, χωρίς να χρειάζεται να περάσετε ολόκληρη τη διαδικασία δημιουργίας εικόνας από την αρχή.

3.1.5 Διατήρηση δεδομένων

Χωρίς πρόσθετη διαμόρφωση, ένα κυτίο Docker δεν έχει μόνιμη αποθήκευση. Η αποθήκευσή του διατηρείται όταν σταματά το κυτίο, αλλά όχι όταν αφαιρείται το κυτίο. Είναι δυνατή η προσάρτηση ενός καταλόγου στον host μέσα σε ένα κυτίο Docker. Αυτό επιτρέπει στο κυτίο να έχει πρόσβαση σε αρχεία του κεντρικού υπολογιστή και να αποθηκεύει σε αυτόν τον προσαρτημένο κατάλογο.

Στην περίπτωση του συγκεκριμένου έργου βάση των παραμετροποιήσεων που έχουν εφαρμοστεί, η πηγαίος κώδικας της εφαρμογής αναμένεται να είναι τοποθετημένος μέσα στο ίδιο root φάκελο μέσα στον οποίο βρίσκεται και το docker image

3.1.6 Δικτύωση

Μέσα στα πλαίσια της ασφάλειας που παρέχει η τεχνολογία Docker είναι ότι ένα Docker-κυτίο δεν έχει άμεση πρόσβαση στο δίκτυο διασφαλίζοντας την λειτουργία του από διαδικτυακές επιθέσεις.

Για να μπορέσει να έρθει σε επαφή με το διαδίκτυο, όταν δημιουργείται ένα κυτίο Docker, ο δαίμονας Docker δημιουργεί μία δικτυακή γέφυρα (“network sandbox”) για αυτό το Docker-κυτίο και (από προεπιλογή) το συνδέει σε ένα εσωτερικό δίκτυο. Αυτό δίνει πόρους δικτύωσης στο Docker-κυτίο (π.χ. διεύθυνση IPv4, διαδρομές και καταχωρήσεις DNS) που είναι ξεχωριστοί σε σχέση με τον κεντρικό υπολογιστή.

΄Όλη η εισερχόμενη και η εξερχόμενη κίνηση στο Docker-κυτίο δρομολογείται μέσω μιας διεπαφής (από προεπιλογή) η οποία γεφυρώνεται σε μια διεπαφή στον κεντρικό υπολογιστή (Host).

Η εισερχόμενη κίνηση (που δεν αποτελεί μέρος μιας υπάρχουσας σύνδεσης) είναι δυνατή μέσα από δρομολόγηση της κυκλοφορίας για συγκεκριμένες πόρτες από τον κεντρικό υπολογιστή στο Docker-κυτίο. Ο προσδιορισμός των θυρών του κεντρικού υπολογιστή από όπου δρομολογείται η κυκλοφορία και των θυρών του Docker-κυτίου όπου αυτή οδηγείται γίνεται κατά την δημιουργία ενός Docker-κυτίο.

Στο συγκεκριμένο έργο η εντολή εκκίνησης του docker εφαρμόζεται ως:

docker run –rm -p 80:80 –name=ionianweather ionianweather

Η πρώτη δημιουργεί μια εικόνα Docker χρησιμοποιώντας το αρχείο Dockerfile και στη συνέχεια δημιουργούμε (και ξεκινάμε) ένα Docker-κυτίο από αυτήν την εικόνα. «Δημοσιεύουμε» την θύρα 80 του κεντρικού υπολογιστή στη θύρα 80 του Docker-κυτίου. Αυτό σημαίνει ότι, ενώ το Docker-κυτίο εκτελείται, όλη η κίνηση από τη θύρα 80 στον κεντρικό υπολογιστή δρομολογείται προς τη θύρα 80 του Docker-κυτίου.

Από προεπιλογή, όλα τα Docker-κυτία προστίθενται στο ίδιο εσωτερικό δίκτυο. Αυτό σημαίνει ότι (από προεπιλογή) όλα τα κυτίο Docker μπορούν να επικοινωνήσουν μεταξύ τους μέσω του δικτύου. Αυτό διαφέρει από την απομόνωση την οποία χρησιμοποιεί το Docker για τα namespaces του.

Σε άλλα namespaces (προγραμματιστικοί χώροι), το Docker απομονώνει κυτία από τον οικοδεσπότη καθώς και από άλλα Docker-κυτία. Αυτή η διαφορά στο σχεδιασμό μπορεί να οδηγήσει σε επικίνδυνες εσφαλμένες διαμορφώσεις (misconfigurations), επειδή οι προγραμματιστές μπορεί να πιστεύουν ότι τα Docker κυτία είναι εντελώς απομονωμένα μεταξύ τους (συμπεριλαμβανομένου και του δικτύου).

3.1.7 Μηχανισμοί Προστασίας

Για να μειωθούν σημαντικά οι κίνδυνοι που θέτουν τα (μελλοντικά) τρωτά σημεία διαδικτυακού συστήματος IonianWeather , σε ένα σύστημα με Docker, υπάρχουν πολλαπλοί μηχανισμοί προστασίας ενσωματωμένοι στο Docker και στον ίδιο τον πυρήνα Linux.

Προκειμένου να επιτρέπεται ή να μην επιτρέπεται σε μια διαδικασία να χρησιμοποιεί μια συγκεκριμένη λειτουργικότητα η οποία απαιτεί επαυξημένα δικαιώματα, ο πυρήνας του Linux διαθέτει ένα χαρακτηριστικό που ονομάζεται ¨δυνατότητες¨. Μια δυνατότητα είναι ένας λεπτομερής τρόπος παροχής ορισμένων

προνομίων σε διαδικασίες. Μια δυνατότητα επιτρέπει σε μια διαδικασία να εκτελέσει μια προνομιακή ενέργεια χωρίς να δίνει στη διαδικασία πλήρη δικαιώματα ρίζας (root privileges). Για παράδειγμα, αν θέλουμε μια διεργασία να μπορεί να δημιουργήσει τα δικά της πακέτα δικτύου, της δίνουμε μόνο τη δυνατότητα CAP_NET_RAW.

Από προεπιλογή, κάθε κυτίο Docker ξεκινά μόνο με τις ελάχιστες απαραίτητες δυνατότητες. Οι προεπιλεγμένες δυνατότητες βρίσκονται στο κώδικα Docker. Μπορούμε να προσθέσουμε ή αφαιρέσουμε δυνατότητες κατά το χρόνο εκτέλεσης χρησιμοποιώντας το ορίσματα –cap-add και –cap-drop.

Secure Computing Mode

Η λειτουργία Ασφαλούς Υπολογισμού (seccomp), όπως και οι δυνατότητες, είναι ένα ενσωματωμένο χαρακτηριστικό που περιορίζει την προνομιακή λειτουργικότητα που επιτρέπεται σε μια διαδικασία. Οπου οι δυνατότητες

περιορίζουν τη λειτουργικότητα (όπως η ανάγνωση προνομιακών αρχείων), η λειτουργία Ασφαλούς Υπολογισμού περιορίζει συγκεκριμένες κλήσεις συστήματος (syscalls). Αυτό επιτρέπει τον διεξοδικό έλεγχο ασφαλείας που επιτυγχάνεται μέσα από whitelists (που ονομάζονται προφίλ) των syscalls.

Στο συγκεκριμένο έργο χρησιμοποιήθηκε το προφιλ –security-opt seccomp.

Μη-διαχειριστές (root) χρήστες σε Docker-κυτίο

Για την μεγιστοποίηση της ασφάλειας του διαδικτυακού συστήματος IonianWeather, στο συγκεκριμένο έργο περιορίστηκαν οι χρήστες οι οποίοι δεν ανήκουν στο group των διαχειριστών.

Εκτός από τους μηχανισμούς προστασίας στον κεντρικό υπολογιστή, υπάρχουν και μηχανισμοί προστασίας σε επίπεδο εικόνας Docker. Ο πιο σημαντικός μηχανισμός προστασίας που μπορούν να εφαρμόσουν οι δημιουργοί εικόνων Docker είναι να μην εκτελούν διεργασίες εντός ενός Docker-κυτίου ως root.

Από προεπιλογή, οι διεργασίες στα κυτία Docker εκτελούνται ως root, επειδή η διαδικασία είναι απομονωμένη από τον κεντρικό υπολογιστή. Ωστόσο, όπως θα δούμε, υπάρχουν πολλοί τρόποι διαφυγής από Docker-κυτίο. Οι περισσότεροι από αυτούς τους τρόπους απαιτούν δικαιώματα root (μέσα στο Docker-κυτίο).

Για αυτό το λόγο συνιστάται η εκτέλεση διεργασιών σε Docker-κυτίο χωρίς χρήση root. Αν κάποιος καταφέρει να παραβιάσει το Docker-κυτίο δεν θα καταφέρει να διαφύγει από αυτό διότι δεν έχει δικαιώματα root.

3.1.8 docker-compose

Στο συγκεκριμένο έργο για την υλοποίηση των κυτίων και εικόνων Docker χρησιμοποιήθηκε το λογισμικό docker-compose.

Το Docker Compose είναι ένα εργαλείο για τον ορισμό και την εκτέλεση εφαρμογών Docker με πολλά κοντέινερ. Επιτρέπει στους προγραμματιστές και

τους διαχειριστές να ορίζουν τις υπηρεσίες που συνθέτουν την εφαρμογή τους και να εκτελούν αυτές τις υπηρεσίες χρησιμοποιώντας μία μόνο εντολή. Με το Docker Compose, είναι δυνατή η διαχείριση ολόκληρου του κύκλου ζωής μιας εφαρμογής πολλαπλών εμπορευματοκιβωτίων, από την ανάπτυξη έως την παραγωγή, χρησιμοποιώντας ένα μόνο αρχείο ρυθμίσεων.

Το Docker Compose χρησιμοποιεί ένα αρχείο YAML για να ορίσει τις υπηρεσίες που συνθέτουν μια εφαρμογή, καθώς και τις σχέσεις μεταξύ αυτών των υπηρεσιών. Το αρχείο YAML μπορεί να περιλαμβάνει πληροφορίες όπως η εικόνα που θα χρησιμοποιηθεί για κάθε υπηρεσία, οι θύρες που πρέπει να εκτεθούν και οι μεταβλητές περιβάλλοντος που πρέπει να οριστούν για κάθε υπηρεσία.

Ένα από τα κύρια πλεονεκτήματα της χρήσης του Docker Compose είναι ότι διευκολύνει τη διαχείριση σύνθετων εφαρμογών με πολλά εμπορευματοκιβώτια. Με το Docker Compose, οι προγραμματιστές και οι διαχειριστές μπορούν να ορίσουν όλες τις υπηρεσίες που συνθέτουν μια εφαρμογή σε ένα μόνο αρχείο και μπορούν να εκτελέσουν αυτές τις υπηρεσίες χρησιμοποιώντας μια μόνο εντολή. Αυτό καθιστά εύκολη τη διαχείριση ολόκληρου του κύκλου ζωής μιας εφαρμογής, από την ανάπτυξη έως την παραγωγή, χρησιμοποιώντας ένα μόνο αρχείο ρυθμίσεων.

Ένα άλλο πλεονέκτημα της χρήσης του Docker Compose είναι ότι διευκολύνει την κλιμάκωση και τη διαχείριση των εφαρμογών στην παραγωγή. Με το Docker Compose, είναι δυνατό να καθορίσετε τον αριθμό των αντιγράφων μιας υπηρεσίας που πρέπει να εκτελείται και να επεκτείνετε ή να μειώσετε εύκολα την κλίμακα ανάλογα με τις ανάγκες. Αυτό καθιστά δυνατή τη διαχείριση της απόδοσης και της διαθεσιμότητας των εφαρμογών στην παραγωγή, χωρίς να χρειάζεται να γίνονται χειροκίνητες αλλαγές στην υποκείμενη υποδομή.

Εν κατακλείδι, το Docker Compose είναι ένα ισχυρό εργαλείο για τον ορισμό και την εκτέλεση εφαρμογών Docker με πολλά εμπορευματοκιβώτια. Διευκολύνει τη διαχείριση σύνθετων εφαρμογών πολλαπλών κοντέινερ και διευκολύνει την κλιμάκωση και τη διαχείριση εφαρμογών στην παραγωγή.

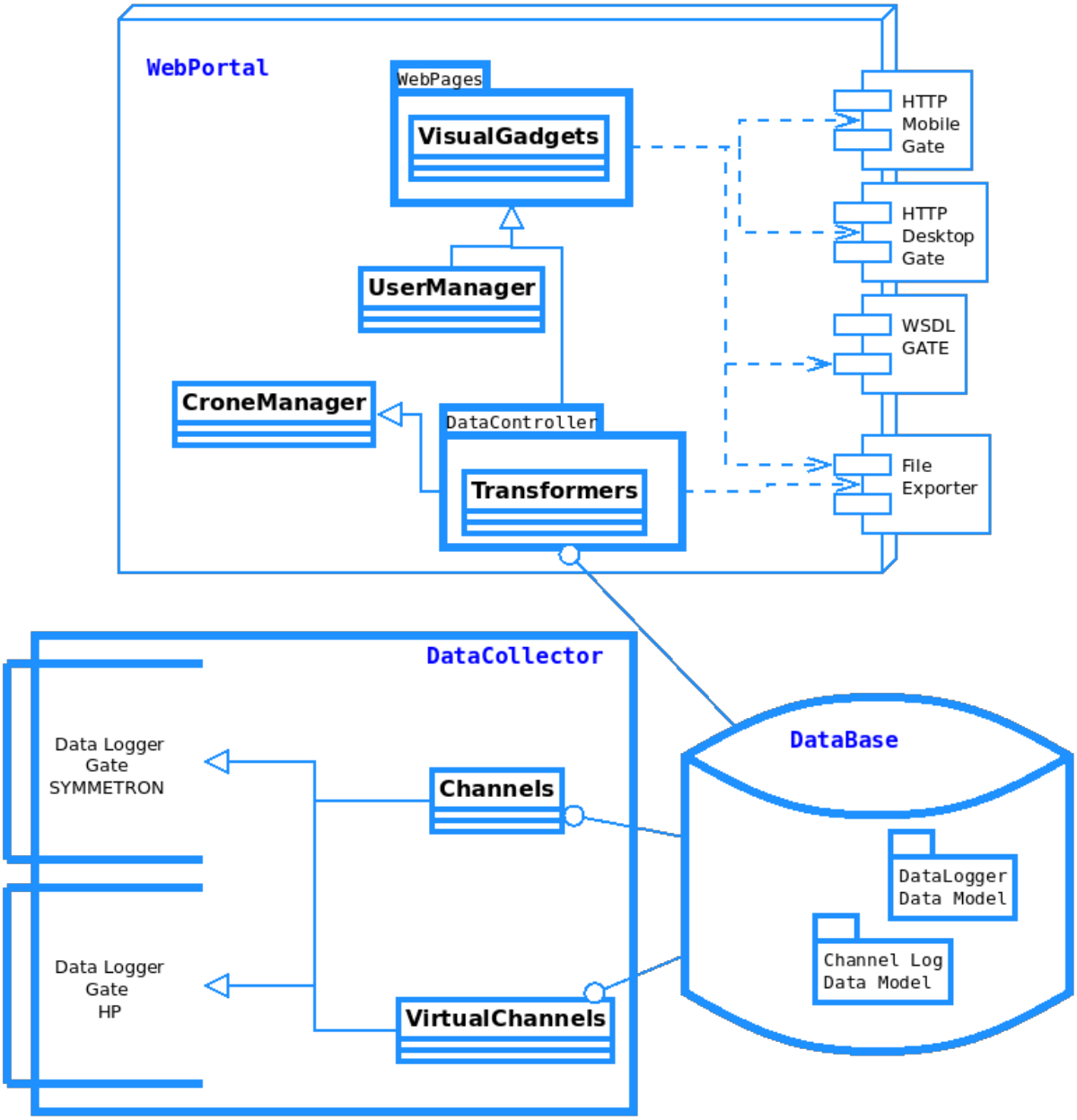
4 Εφαρμοσμένη αρχιτεκτονική περιβάλλοντος φιλοξενίας

Σύμφωνα με τα πορίσματα της μελέτης του παραδοτέου 1 αναπτύχθηκαν και αναβαθμίστηκαν μονάδες στο σύστημα IonianWeather έτσι ώστε να συμμορφωθούν με την αρχιτεκτονική η οποία παρουσιάζεται συνοπτικά στην επόμενη παράγραφο.

4.1 Αποτύπωση υπάρχουσας αρχιτεκτονικής

Αρχιτεκτονικό διάγραμμα συστήματος

Οι βασικές μονάδες οι οποίες υλοποιούν το διαδικτυακό σύστημα είναι οι εξής:



4.1.1 Data Collector

Η συγκεκριμένη μονάδα αναλαμβάνει την συλλογή δεδομένων απο διάφορες διαδικτυακές πηγές και την καταχώρηση των δεδομένων στην βάση δεδομένων.

Τα εργαλεία που ενσωματώνει η συγκεκριμένη μονάδα για να πετύχει το σκοπό της είναι τα παρακάτω:

- Πύλες σύνδεσης. Οι πύλες σύνδεσης υλοποιούν υπηρεσίες προσαρμοσμένες στις ανάγκες των διαφορετικών παροχών μετεωρολογικών δεδομένων και αναλαμβάνουν την λήψη των δεδομένων. Η σημερινή υλοποίηση εφαρμόζει τις εξής πύλες σύνδεσης:
  - Symmetron. Η συγκεκριμένη πύλη υλοποιεί WSDL πρωτοκόλλου το οποίο λαμβάνει μετεωρολογικά δεδομένα από SYMMETRON Data Logger με διαμεσολαβητή το λοιμικό Diameson
- RAW channels. Τα κανάλια σύνδεσης αναλαμβάνουν την καταγραφή των διαφορετικών μετεωρολογικών παραμέτρων στην βάση δεδομένων. Η καταγραφή των δεδομένων εμπλουτίζεται με διάφορα μεταδομένα όπως GIS, Timestamps κ.α.
- Virtual channels. Τα εικονικά κανάλια αναλαμβάνουν την παραγωγή νέων μετεωρολογικών παραμέτρων από την μαθηματική επεξεργασία άλλων μετεωρολογικών παραμέτρων. Μετά την postprocess διαδικασία καταχωρούν τα εικονικά δεδομένα στην βάση δεδομένων.
- Crone Manager. Η συγκεκριμένη υπομονάδα αναλαμβάνει σε τακτά χρονικά διαστήματα να ενεργοποιεί τα κανάλια που την ενσωματώνουν έτσι ώστε να εκτελέσουν προγραμματιζόμενες εργασίες. Π.χ. υπολογισμό ημερήσιας βροχόπτωσης.

4.1.2 Web Portal

Η συγκεκριμένη μονάδα έχει δύο βασικούς σκοπούς:

1. Υλοποίηση διαδικτυακού τόπου για την προβολή των μετεωρολογικών δεδομένων στο κοινό και στους ερευνητές. Η προβολή των δεδομένων γίνεται μέσο διαβάθμισης έτσι ώστε να υπάρχει περιορισμός πρόσβασης ανάλογα με τον ρόλο του χρήστη
2. Υλοποίηση WSDL υπηρεσίας για την διάχυση των μετεωρολογικών δεδομένων σε άλλες μετεωρολογικές διαδικτυακές υπηρεσίες οι οποίες έχουν την ανάγκη χρήσης των συλλεγόντων δεδομένων του IonianWeather π.χ. Εθνική μετεωρολογική Υπηρεσία (Ε.Μ.Υ.)

Οι βασικότερες υπομονάδες του Web Portal είναι οι εξής:

- User Manager. Η συγκεκριμένη υπομονάδα αναλαμβάνει τα εξής:
  - ✔ Διαβάθμιση των χρηστών που συνδέονται στο σύστημα.
  - ✔ Παραγωγή πολιτικών πρόσβασης στα δεδομένα και στις υπομονάδες του WebPortal ανάλογα με το ρόλο του χρήστη
- Data Controller. Η συγκεκριμένη υπομονάδα αναλαμβάνει την συλλογή των μετεωρολογικών δεδομένων από την βάση δεδομένων ανάλογα με την πολιτική πρόσβασης του χρήστη.
- Visual Gadgets. Η συγκεκριμένη υπομονάδα αναλαμβάνει την ενσωμάτωση στον Client εργαλείων που οπτικοποιούν τα μετεωρολογικά δεδομένα π.χ. Ιστόγραμμα.
- HTTP πύλες. Οι συγκεκριμένες πύλες υλοποιούν τους διάφορους client πρόσβασης στο IonianWeather. Οι πύλες οι οποίες έχουν υλοποιηθεί είναι οι εξής:
  - ✔ Web. Αναλαμβάνει την παραγωγή ιστοσελίδων πλοήγησης και προβολής μετεωρολογικών δεδομένων

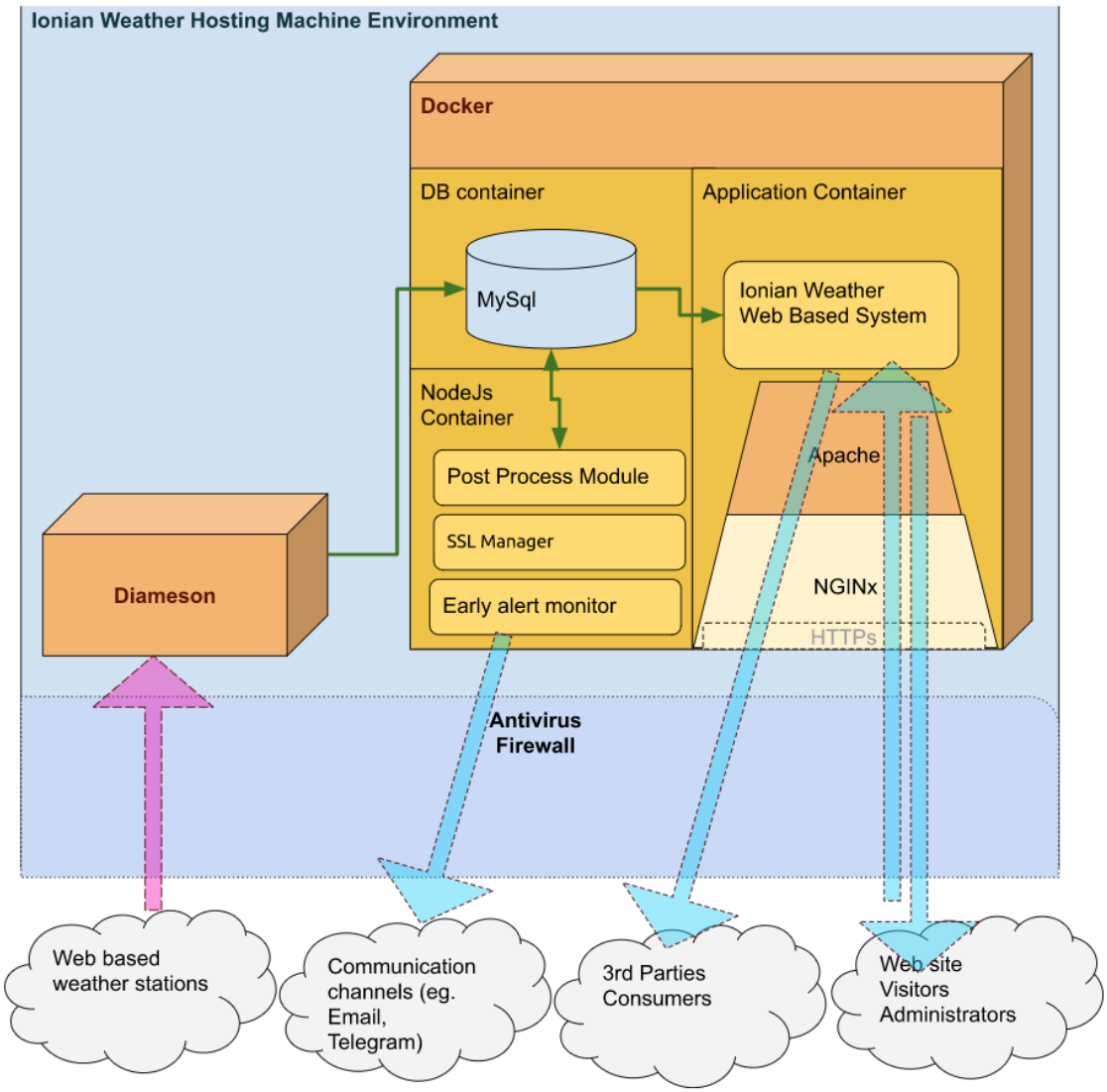
✔ File Exporter. Αναλαμβάνει την παραγωγή αρχείων (π.χ. CSV) με επιλεγμένα μετεωρολογικά δεδομένα.

✔ WSDL. Αναλαμβάνει την διάχυση των μετεωρολογικών δεδομένων σε άλλες μετεωρολογικές διαδικτυακές υπηρεσίες

4.2 Παρεμβάσεις βελτιστοποίησης του συστήματος Ionian Weather

Σαν στόχο της ευρύτερης ανάπτυξης/αναβάθμισης των μονάδων είναι η συνεχής και ασφαλής λειτουργία του συστήματος Ionian Weather. Όπως έχει προαναφερθεί η τεχνολογία Docker προσφέρει την υποδομή για την επίτευξη αυτών των στόχων.

Στα πλαίσια αυτού του έργου οι μονάδες του διαδικτυακού συστήματος IonianWeather εγκαταστάθηκαν σε παραμετροποιημένα docker containers (κυτία) τα οποία ακολουθούν το παρακάτω αρχιτεκτονικό διάγραμμα.



Εννοιολογική αρχιτεκτονική περιβάλλοντος φιλοξενίας

Οι βασικές διατάξεις οι οποίες υλοποιούν το διαδικτυακό σύστημα είναι οι εξής (όπως παρουσιάζονται στο παραπάνω διάγραμμα) αναλύονται παρακάτω:

4.2.1 Ionian Weather Hosting Machine Environment

Ως “Hosting Machine Environment”, ουσιαστικά, αναφερόμαστε στο ειδικά παραμετροποιημένο λειτουργικό σύστημα.

Η παρούσα παραμετροποίηση αφορά λειτουργικό σύστημα Windows εγκατεστημένα σε υπολογιστή με τις εξής προδιαγραφές:

- RAM : 8 GB
- CPU : Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
- HD : 500 GB

Για την διασφάλιση της ορθής και ασφαλής λειτουργίας του διαδικτυακού συστήματος IonianWeather εγκαταστάθηκαν και παραμετροποιήθηκαν τα εξής λογισμικά:

- Docker με πυρήνα Linux
- Antivirus Norton
- Windows/Norton Firewall με τις εξής ρυθμίσεις:
    - Black List αμφίδρομα όλες οι πόρτες εκτός των Whitelist
    - Whitelist πόρτες
        - Πόρτα 80 εισερχόμενη/εξερχόμενη για την κάλυψη αναγκών επίσκεψης του συστήματος απο browser
        - Πόρτα 1023 Diameson μόνο για εισερχόμενη κίνηση απο μετεωρολογικούς σταθμούς
        - Πόρτα 8053 για τις ανάγκες απομακρυσμένης διαχείρισης και συντήρησης του συστήματος.

4.2.2 Diameson

Η μονάδα Diameson αποτελεί αυτόνομο λογισμικό το οποίο είναι ανεπτυγμένο σε περιβάλλον windows και σκοπό έχει να συλλέγει δεδομένα από τους μετεωρολογικούς σταθμού του δικτύου IonianWeather με τεχνολογία Socket.

Η αρχική παραμετροποίηση του Diameson τα συλλεγμένα δεδομένα τα προωθούσε στην βάση δεδομένων μέσω πύλης Http. Η συγκεκριμένη αρχιτεκτονική κρίθηκε επισφαλής και επικίνδυνη για DOS attacks.

Βάση των προδιαγραφών ασφαλείας που έχουν προσδιοριστεί για την ασφαλή λειτουργία των μονάδων αυτού του έργου κρίθηκε απαραίτητο τα εισερχόμενα δεδομένα να καταχωρούνται απευθείας μέσα στην Βάση Δεδομένων. Αυτή η φιλοσοφία καθιστά την διαδικασία εξαιρετικά ασφαλής καθότι η βάση δεδομένων (MySQL) έχει παραμετροποιηθεί έτσι ώστε να επιτρέπει την

εκτέλεση διαδικασιών μόνο από συγκεκριμένες εφαρμογές που βρίσκονται στο ίδιο περιβάλλον.

Υπακουώντας στην συγκεκριμένη ανάγκη το Diameson παραμετροποιήθηκε κατάλληλα έτσι ώστε να συνδέεται στην βάση δεδομένων η οποία βρίσκεται σε λειτουργία σε περιβάλλον Docker-κυτίο.

4.2.3 DB Container

Το συγκεκριμένο container έχει παραμετροποιηθεί για να παρέχει τα εξής:

- Βάση δεδομένων MySQL 5.7
- Εξυπηρέτηση κλήσεων μόνο από:
  - Διαβαθμισμένες εφαρμογές οι οποίες φέρουν το κατάλληλο username/password
  - Δίαυλο mysql (εσωτερικός δίαυλος Docker) στην πόρτα 3036
  - Δίαυλο διαμέσου localhost (εσωτερικός δίαυλος εφαρμογών που βρίσκονται στο ίδιο μηχάνημα) στην πόρτα 3036 για να την κάλυψη των αναγκών του λογισμικού Diameson

4.2.4 Application Container

Το συγκεκριμένο container αποτελεί τον πυρήνα του συστήματος το οποίο ενσωματώνει:

- Η γλώσσα προγραμματισμού PHP 8
- Το Framework MODx Revolution σε έκδοση 2.8
- Πηγαίος κώδικας μονάδων διαδικτυακού συστήματος IonianWeather
- Server (Εξυπηρετητές) Nginx/Apache. Η συγκεκριμένη διάταξη αναλύεται διεξοδικά παρακάτω

4.2.5 NodeJs Container

Το NodeJs αποτελεί ένα προηγμένο προγραμματιστικό περιβάλλον το οποίο

χρησιμοποιείται για την κάλυψη αναγκών διαχείρισης των υπόλοιπων μονάδων του συστήματος. Οι βασικές υπομονάδες NodeJs που έχουν υλοποιηθεί στα πλαίσια αυτού του έργου είναι οι εξής:

4.2.5.1 Lets Encrypt Auto certificate management agent

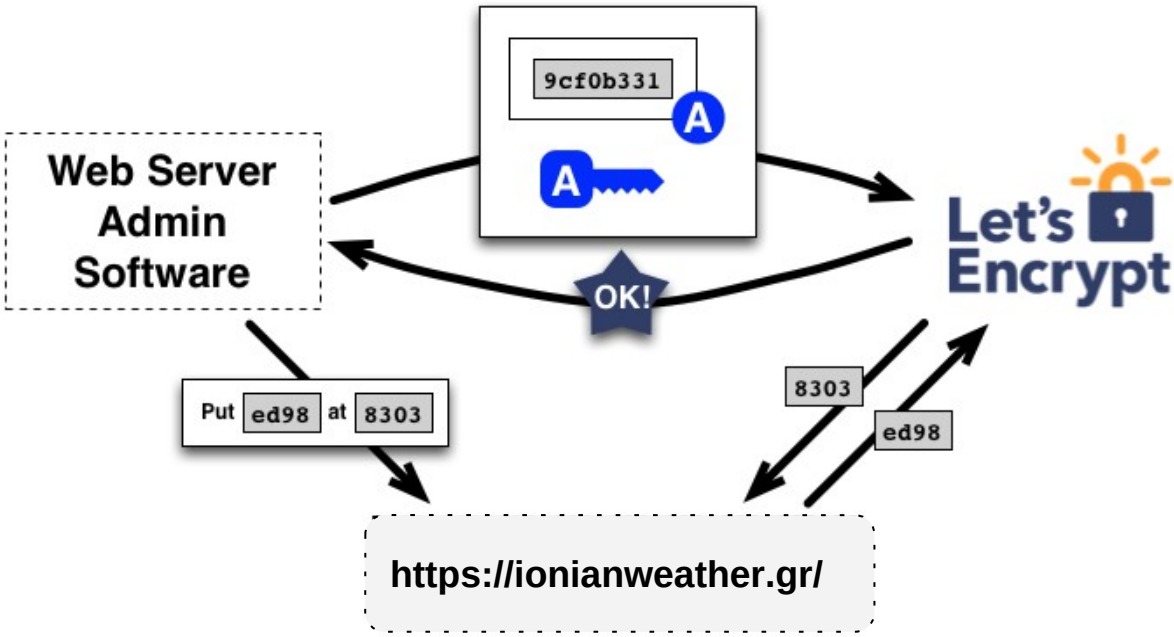
Όπως είχε αναφερθεί στο 1ο τμήμα του Παραδοτέου για την ασφαλή σύνδεση στο διαδικτυακό σύστημα IonianWeather στο διαδίκτυο (Internet) υλοποιήθηκε το πρωτόκολο HTTPS. Για την υλοποίηση αυτού του πρωτοκόλλου απαιτήθηκε και απαιτείτε συνεχόμενα η εξασφάλιση κατοχής διαδικτυακού πιστοποιητικού SSL (αναλύθηκε στο πρώτο τμήμα του παραδοτέου). Στο συγκεκριμένο έργο η εξασφάλιση πιστοποιητικού SSl πραγματοποιήθηκε βασισμένη στο πιστοποιητικό Let’s Encrypt.

Ο στόχος της Let’s Encrypt και του πρωτοκόλλου ACME είναι να καταστήσει δυνατή τη δημιουργία ενός διακομιστή HTTPS να αποκτήσει αυτόματα ένα πιστοποιητικό αξιόπιστου προγράμματος περιήγησης, χωρίς ανθρώπινη παρέμβαση έτσι ώστε να παρέχεται συνεχόμενα και αδιάλειπτα η ασφαλή λειτουργία του διαδικτυακού συστήματος IonianWeather. Αυτό επιτυγχάνεται εκτελώντας (χρησιμοποιώντας το Nodejs) έναν πράκτορα διαχείρισης πιστοποιητικών (certificate management agent) στον διακομιστή ιστού (web server).

Ο πράκτορας αναλαμβάνει την εκτέλεση δύο βημάτων:

1. Πρώτον, ο πράκτορας αποδεικνύει στην CA (αρχή έκδοσης πιστοποιητικών) ότι ο διακομιστής ιστού ελέγχει έναν τομέα.
2. Στη συνέχεια, ο πράκτορας μπορεί να ζητήσει, να ανανεώσει και να ανακαλέσει πιστοποιητικά για αυτόν τον τομέα.

Όταν ο πράκτορας έχει ένα εξουσιοδοτημένο ζεύγος κλειδιών, το αίτημα, η ανανέωση και η ανάκληση πιστοποιητικών είναι απλό - απλώς στέλνονται μηνύματα διαχείρισης πιστοποιητικών και υπογράφονται τα με το εξουσιοδοτημένο ζεύγος κλειδιών.



Για να αποκτήσει ένα πιστοποιητικό για τον τομέα, ο πράκτορας δημιουργεί ένα αίτημα υπογραφής πιστοποιητικού PKCS#10 που ζητά από την Let's Encrypt CA να εκδώσει ένα πιστοποιητικό για το ionianweather.gr με ένα καθορισμένο δημόσιο κλειδί. Ως συνήθως, η CSR περιλαμβάνει μια υπογραφή από το ιδιωτικό κλειδί που αντιστοιχεί στο δημόσιο κλειδί στην CSR. Ο πράκτορας υπογράφει επίσης ολόκληρη την CSR με το εξουσιοδοτημένο κλειδί για το example.com, έτσι ώστε η Let's Encrypt CA να γνωρίζει ότι είναι εξουσιοδοτημένο.

4.2.5.2 Post Process Module

Η συγκεκριμένη υπομονάδα στόχο έχει να “κανονικοποιεί” τα φυσικά δεδομένα όπως αυτά παρέχονται από τον λογισμικό Diameson έτσι ώστε να είναι συμβατά με τις ανάγκες των μονάδων του συστήματος.

Το συγκεκριμένο module έχει παραμετροποιηθεί να ενεργοποιείται αυτόματα όταν παρατηρείτε νέα εισαγωγή δεδομένων στην βάση δεδομένων.

Για λόγους ασφαλείας τα δεδομένα που καταχωρούνται από το Diameson εισάγονται σε προσωρινό πίνακα τύπου buffer.

Σε συνέχεια η συγκεκριμένη μονάδα αναλαμβάνει να τα διαμορφώσει κατάλληλα και να τα εισάγει στους ανάλογους πίνακες στην βάση δεδομένων.

5 Προηγμένη ασφαλής τοπολογία εξυπηρετητή

Για να διασφαλίσουμε την ορθή λειτουργία του διαδικτυακού συστήματος IonianWeather πέρα επιθέσεων τύπου Dos υλοποιήθηκε προηγμένη τοπολογία διάταξης server (διακομιστής) έτσι ώστε να επιτυγχάνεται ασφαλή λήψη/μετάδοση προς τους διασυνδεδεμένους clients. Client (πελάτης) θεωρείτε οποιοδήποτε λογισμικό (π.χ. browser) ή οποιαδήποτε άλλη διαδικτυακή εφαρμογή έχει ανάγκη διεπαφής με το διαδικτυακό σύστημα IonianWeather.

Για να λειτουργούν αποτελεσματικά, ο client και ο server ανταλλάσσουν πληροφορίες τακτικά. Ένας διαδικτυακός server συνήθως χρησιμοποιεί αντίστροφους server μεσολάβησης. Ένας client βλέπει έναν αντίστροφο server μεσολάβησης ή πύλη σαν να ήταν ένας κανονικός server ιστού και δεν απαιτούνται επιπλέον διαμορφώσεις. Ο client στέλνει τυπικά αιτήματα στον αντίστροφο server μεσολάβησης, ο οποίος στη συνέχεια καθορίζει πού θα σταλούν τα δεδομένα, παρέχοντας το τελικό αποτέλεσμα στον πελάτη σαν να ήταν η προέλευση.

Ο πιο συνηθισμένος διαδικτυακός server, στην περίπτωση αυτού του έργου ο Apache , μπορεί να εκτελείται σε ένα ευρύ φάσμα λειτουργικών συστημάτων, συμπεριλαμβανομένων των UNIX/Linux, Microsoft Windows και OpenVMS.

Ο NGINX είναι ένας διαδικτυακός server υψηλής απόδοσης που χρησιμοποιείται κυρίως για στατικά αρχεία και ως αντίστροφος server μεσολάβησης, αλλά μπορεί να χρησιμοποιηθεί σε οποιοδήποτε πλαίσιο λόγω της προσαρμοστικότητας του.

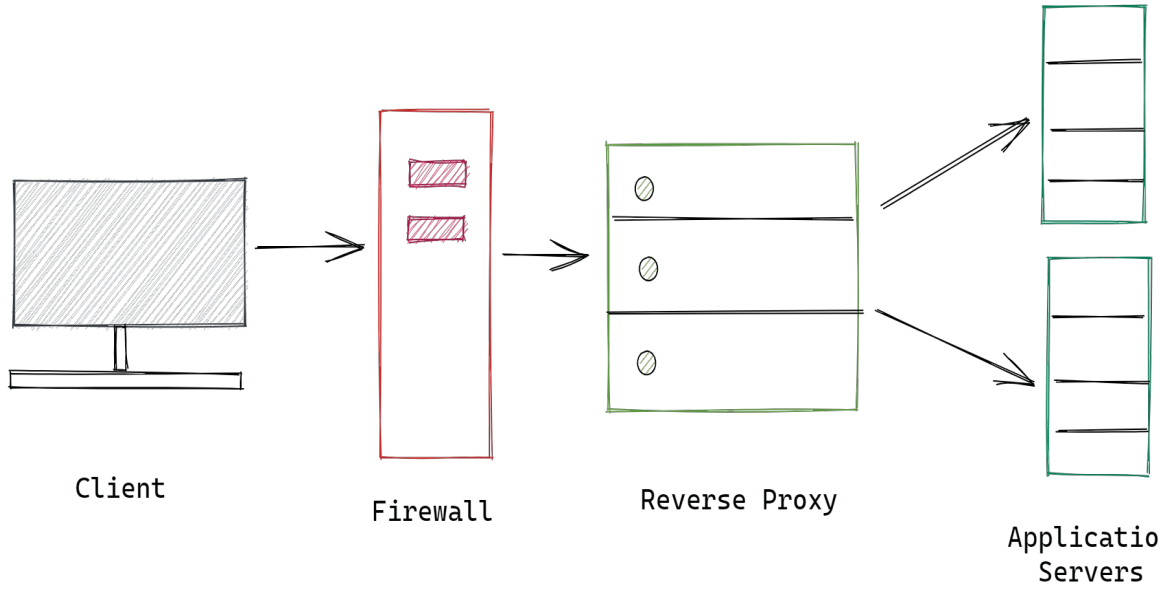
5.1 Διάταξη αντίστροφου διακομιστή(server) μεσολάβησης

Ένας αντίστροφος διακομιστής μεσολάβησης βρίσκεται μεταξύ εσωτερικών εφαρμογών και εξωτερικών πελατών, μεταφέροντας αιτήματα πελατών στον κατάλληλο διακομιστή. Η υπηρεσία αντίστροφου διακομιστή μεσολάβησης χρησιμεύει ως διεπαφή, που χειρίζεται όλα τα αιτήματα client και τα προωθεί στον ιστό του back-end, στη βάση δεδομένων ή σε άλλους διακομιστές και, στη συνέχεια, ο client λαμβάνει την απάντηση.

Ένας τυπικός διακομιστής μεσολάβησης ενεργεί για λογαριασμό των client, συνήθως προσφέροντας απόρρητο ή έλεγχο περιεχομένου. Ένας αντίστροφος διακομιστής μεσολάβησης παρεμποδίζει την κυκλοφορία για λογαριασμό ενός διακομιστή και τη δρομολογεί σε διαφορετικό διακομιστή.

Στο συγκεκριμένο έργο υλοποιήθηκε διάταξη αντίστροφου διακομιστή μεσολάβησης για την βελτιστοποίηση της ασφάλειας των διασυνδέσεων.

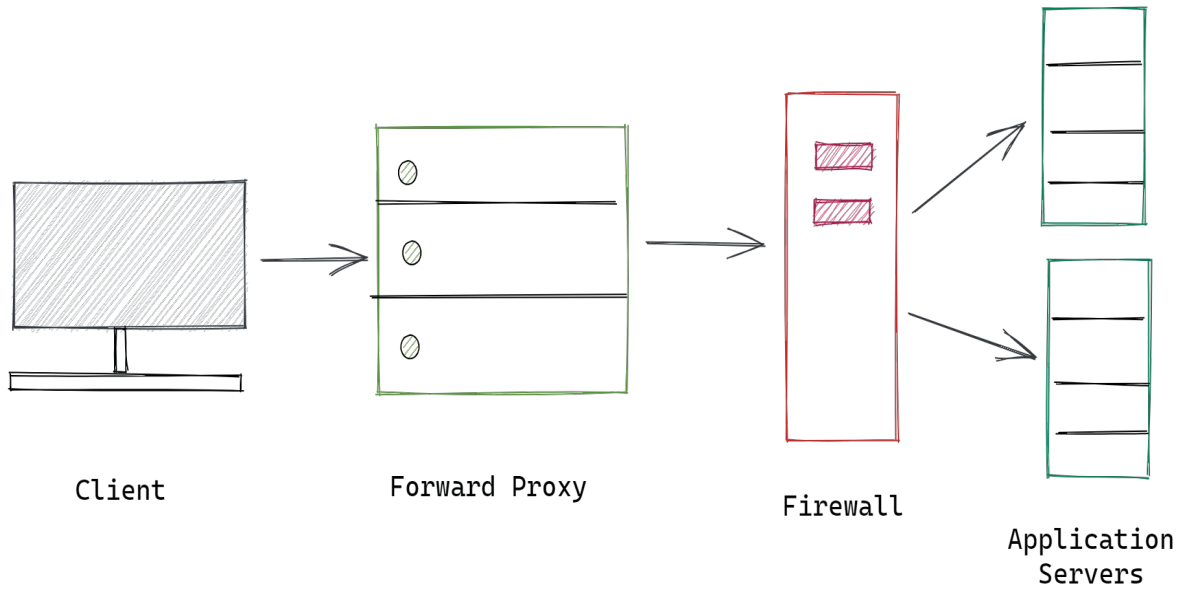
Ο αντίστροφος διακομιστής μεσολάβησης κατά την υποστήριξη του IonianWeather βοηθά στη συγκέντρωση της καταγραφής και της αναφοράς σε πολλούς διακομιστές, καθώς παρέχει ένα ενιαίο σημείο επαφής για τους client.



Το πρόγραμμα περιήγησης/συσκευή (browser) συνδέεται απευθείας με τον διακομιστή του ιστότοπου και ξεκινά τη λήψη των πόρων του όταν περιηγείστε κανονικά από το IonianWeather, κατα την επίσκεψή του.

Στο διαδικτυακό σύστημα IonianWeather χρησιμοποιήθηκε ένα διακομιστής μεσολάβησης για να προωθήσετε πρώτα όλα τα αιτήματά σας σε αυτόν. Σε συνέχεια τα αιτήματά των διασυνδεδεμένων client προωθούνται στο λογισμικό επίλυσης DNS , το οποίο στη συνέχεια θα ανακτήσει τους πόρους του IonianWeather από τον **διακομιστή προέλευσης (Application server)** ο οποίος φιλοξενεί τις μονάδες του διαδικτυακού συστήματος IonianWeather.

Στη συνέχεια, ο διακομιστής παρέχει τους πόρους του IonianWeather στο client. **Αυτή η διάταξη επιτρέπει το διαδικτυακό σύστημα IonianWeather να επικοινωνεί μόνο με τον διακομιστή διαμεσολαβητή και όχι απευθείας με τον client ο οποίος ενδέχεται να είναι επισφαλής ή κάποιο κακόβουλο λογισμικό.**



Σημείωση: *Σύμφωνα με την ταξινόμηση στις προηγούμενες ενότητες, όταν το NGINX χρησιμοποιείται ως διακομιστής μεσολάβησης HTTPS, λειτουργεί ως διαφανής διακομιστής μετάδοσης (σήραγγα), που σημαίνει ότι δεν αποκωδικοποιεί ή ανιχνεύει την επικοινωνία ανώτερου επιπέδου.*

Μερικά από τα σημαντικά οφέλη της συγκεκριμένης αρχιτεκτονικής παρουσιάζονται στις επόμενες παραγράφους.

5.1.1 Προηγμένη ασφάλεια

Η διεύθυνση IP και άλλα χαρακτηριστικά των διακομιστών προέλευσης μπορούν να κρυφτούν μέσω αντίστροφων διακομιστών μεσολάβησης. Ως αποτέλεσμα, ο διακομιστής προέλευσης του ιστοτόπου του IonianWeather μπορεί να διατηρήσει καλύτερα την ανωνυμία του, ενισχύοντας σημαντικά την ασφάλειά του.

Δεδομένου ότι ο αντίστροφος διακομιστής μεσολάβησης λαμβάνει όλη την επισκεψιμότητα πριν φτάσει στον κύριο διακομιστή, τυχόν εισβολείς ή χάκερ που θέλουν να στοχεύσουν το σύστημα με ανησυχίες ασφαλείας, όπως επιθέσεις DDoS, θα δυσκολευτούν να το κάνουν.

5.1.2 Εξισορρόπηση φορτίου

Μελλοντικά το IonianWeather ενδέχεται να παρουσιαστεί ανάγκη εξυπηρέτησης μεγάλου φόρτου διασυνδέσεων client. Ένα διαδικτυακό σύστημα όπως το IonianWeather έχει την δυνατότητα εξυπηρέτησης πεπερασμένου αριθμού καθημερινών μοναδικών χρήστών. Εφόσον παρατηρηθεί ότι υπάρχει καθυστέρηση στην εξυπηρέτηση όλων των διασυνδέσεων η συγκεκριμένη αρχιτεκτονική έχει την δυνατότητα να επεκταθεί έτσι ώστε να μπορεί να κατανείμει έξυπνα την κυκλοφορία μεταξύ μιας δεξαμενής πολλών διακομιστών που φιλοξενούν το IonianWeather και οι οποίοι μοιράζονται κοινούς πόρους όπως τα δεδομένα.

Ο αντίστροφος διακομιστής μεσολάβησης λαμβάνει την εισερχόμενη κίνηση προτού φτάσει στον διακομιστή προέλευσης. Εάν ο διακομιστής προέλευσης του IonianWeather υπερφορτωθεί, η κίνηση μπορεί να διανεμηθεί σε άλλους διακομιστές χωρίς να διακυβεύεται η λειτουργία του ιστότοπου.

5.1.3 Παρακολούθηση και καταγραφή της κυκλοφορίας

Τυχόν αιτήματα που περνούν από τον αντίστροφο διακομιστή μεσολάβησης του IonianWeather καταγράφονται. Ως αποτέλεσμα, υπάρχει η δυνατότητα παρακολούθησης της διαδικτυακής κίνησης για τις ανάγκες αναγνώρισης τυχών προβλημάτων που ενδέχεται να παρουσιαστούν κατά την διάρκεια διασυνδέσεων από διάφορους client.

5.1.4 Ισχυρή προσωρινή αποθήκευση

Ο αντίστροφος διακομιστής μεσολάβησης του IonianWeather χρησιμοποιείτε για την επιτάχυνση των ιστοσελίδων που παρέχει μέσω της προσωρινής αποθήκευσης στατικού και δυναμικού περιεχομένου. Αυτό μπορεί να μειώσει την επιβάρυνση του διακομιστή προέλευσης, κάνοντας τον ιστότοπο να επικοινωνεί πιο γρήγορα.

5.1.5 Βελτιστοποιημένη κρυπτογράφηση SSL

Ο διακομιστής προέλευσης μπορεί να επιβαρυνθεί υπερβολικά από την κρυπτογράφηση και την αποκρυπτογράφηση αιτημάτων SSL/TLS για κάθε client. Ο αντίστροφος διακομιστής μεσολάβησης του IonianWeather αναλαμβάνει να κάνει αυτήν τη λειτουργία, ελευθερώνοντας πόρους στον διακομιστή προέλευσης για άλλα κρίσιμα καθήκοντα, όπως η παροχή του περιεχομένου.

Ένα άλλο πλεονέκτημα της εκφόρτωσης κρυπτογράφησης και αποκρυπτογράφησης SSL/TSL είναι ότι μειώνει τον λανθάνοντα χρόνο για πελάτες που βρίσκονται πολύ μακριά από τον διακομιστή προέλευσης.

5.2 Ασφάλεια της επεξεργασίας προσωπικών δεδομένων

5.2.1 Αξιολόγηση επιπέδου κινδύνου - Μέτρα ασφαλείας

Μετά την αξιολόγηση του επιπέδου κινδύνου, μπορούμε προχωρήσουμε στην επιλογή των κατάλληλων μέτρων ασφαλείας για την προστασία των προσωπικών δεδομένων.

Υπάρχουν δύο κύριες κατηγορίες μέτρων: οργανωτικές και τεχνικές. Αυτές οι ευρείες κατηγορίες έχουν περαιτέρω χωριστεί σε υποκατηγορίες με σύντομη περιγραφή, εξηγώντας πώς κάθε υποκατηγορία σχετίζεται με συγκεκριμένες διατάξεις του GDPR.

Κάτω από κάθε υποκατηγορία παρουσιάζονται τα μέτρα ανά επίπεδο κινδύνου. Προκειμένου να επιτευχθεί επεκτασιμότητα, θεωρείται ότι όλα τα μέτρα που περιγράφονται στο χαμηλό επίπεδο ισχύουν για όλα τα επίπεδα. Ομοίως, τα μέτρα που παρουσιάζονται κάτω από το μεσαίο επίπεδο ισχύουν και για υψηλό επίπεδο κινδύνου. Τα μέτρα που παρουσιάζονται κάτω από το υψηλό επίπεδο δεν ισχύουν για κανένα άλλο επίπεδο κινδύνου.

Θα πρέπει να σημειωθεί ότι η αντιστοίχιση των μέτρων με συγκεκριμένα επίπεδα κινδύνου δεν θα πρέπει να εκλαμβάνεται ως απόλυτη. Ανάλογα με το πλαίσιο της επεξεργασίας των προσωπικών δεδομένων, ο οργανισμός μπορεί να εξετάσει το ενδεχόμενο υιοθέτησης πρόσθετων μέτρων, ακόμη και αν αυτά αποδίδονται σε υψηλότερο επίπεδο κινδύνου. Επιπλέον, ο προτεινόμενος κατάλογος μέτρων δεν λαμβάνει υπόψη άλλες πρόσθετες ειδικές τομεακές απαιτήσεις ασφάλειας, καθώς και ειδικές ρυθμιστικές υποχρεώσεις, που απορρέουν, για παράδειγμα, από την Οδηγία για την προστασία της ιδιωτικής ζωής στις ηλεκτρονικές επικοινωνίες (σύνδεση προς ένα νέο παράθυρο) ή την Οδηγία NIS5 (σύνδεση προς ένα νέο παράθυρο). Σε μια προσπάθεια περαιτέρω διευκόλυνσης αυτής της διαδικασίας περιλαμβάνεται επίσης μια χαρτογράφηση της ..προτεινόμενης ομάδας μέτρων με ελέγχους ασφαλείας.

Πολιτική και διαδικασίες ασφαλείας για την προστασία των προσωπικών δεδομένων

Η πολιτική ασφαλείας είναι ένα έγγραφο υψηλού επιπέδου που θέτει τις βασικές αρχές για την ασφάλεια και την προστασία των προσωπικών δεδομένων σε έναν οργανισμό. Αποτελεί, λοιπόν, τη βάση για την εφαρμογή όλων των ειδικών τεχνικών και οργανωτικών μέτρων, σύμφωνα με το άρθ. 32 GDPR, όπως επίσης συμπληρώνεται από το άρθρο. 24 GDPR (εφαρμογή πολιτικών προστασίας δεδομένων).

Με βάση την πολιτική ασφαλείας, τα συγκεκριμένα τεχνικά και οργανωτικά μέτρα περιγράφονται σε ένα σύνολο λεπτομερέστερων πολιτικών/διαδικασιών (π.χ. σχετικά με τον έλεγχο πρόσβασης, τη διαχείριση συσκευών, τη διαχείριση πόρων κ.λπ.).

Η πολιτική ασφαλείας δείχνει τη συνολική δέσμευση της διοίκησης του οργανισμού για την ασφάλεια και την προστασία δεδομένων. Μπορεί να βασίζεται ή να αποτελεί μέρος της γενικής πολιτικής ασφάλειας πληροφορικής του οργανισμού. σε κάθε περίπτωση, θα πρέπει να αφορά ρητά και την προστασία των προσωπικών δεδομένων.

Η κατάταξη των επιπέδων πολιτικής βάση των επιπέδων κινδύνου κατηγοριοποιούνται ως εξής:

1. Ο οργανισμός θα πρέπει να τεκμηριώνει την πολιτική του όσον αφορά την επεξεργασία προσωπικών δεδομένων ως μέρος της πολιτικής του για την ασφάλεια των πληροφοριών.

2. Η πολιτική ασφαλείας θα πρέπει να επανεξετάζεται και να αναθεωρείται, εάν χρειάζεται, σε ετήσια βάση.

3. Ο οργανισμός θα πρέπει να τεκμηριώνει μια ξεχωριστή ειδική πολιτική ασφάλειας όσον αφορά την επεξεργασία προσωπικών δεδομένων. Η πολιτική θα πρέπει να εγκρίνεται από τη διοίκηση και να κοινοποιείται σε όλους τους υπαλλήλους και τα σχετικά εξωτερικά μέρη

4. Η πολιτική ασφάλειας θα πρέπει τουλάχιστον να αναφέρεται: στους ρόλους και τις ευθύνες του προσωπικού, στα βασικά τεχνικά και οργανωτικά μέτρα που έχουν ληφθεί για την ασφάλεια των δεδομένων προσωπικού χαρακτήρα, στους εκτελούντες την επεξεργασία δεδομένων ή σε άλλα τρίτα μέρη που εμπλέκονται στην επεξεργασία δεδομένων προσωπικού χαρακτήρα. Α.5 Θα πρέπει να δημιουργηθεί και να τηρηθεί κατάλογος συγκεκριμένων πολιτικών/διαδικασιών που σχετίζονται με την ασφάλεια των προσωπικών δεδομένων, με βάση τη γενική πολιτική ασφαλείας.
5. Η πολιτική ασφαλείας θα πρέπει να επανεξετάζεται και να αναθεωρείται, εάν είναι απαραίτητο, σε εξαμηνιαία βάση.

Σχετίζεται με το ISO 27001:2013 - A.5 Πολιτική ασφαλείας

5.2.2 Ρόλοι και ευθύνες

Σύμφωνα με το άρθ. 32 (4) GDPR, «ο υπεύθυνος επεξεργασίας και ο εκτελών την επεξεργασία λαμβάνουν μέτρα για να διασφαλίσουν ότι οποιοδήποτε φυσικό πρόσωπο που ενεργεί υπό την εξουσία του υπευθύνου επεξεργασίας ή του εκτελούντος την επεξεργασία που έχει πρόσβαση σε προσωπικά δεδομένα δεν τα επεξεργάζεται παρά μόνο με οδηγίες του υπεύθυνου επεξεργασίας, εκτός εάν αυτός ή υποχρεούται να το πράξει από το δίκαιο της Ένωσης ή του κράτους μέλους».

Ως εκ τούτου, ως πρώτος και βασικός έλεγχος για την ασφάλεια των προσωπικών δεδομένων, όλες οι θέσεις του οργανισμού με πρόσβαση σε προσωπικά δεδομένα θα πρέπει να έχουν σαφώς καθορισμένες και τεκμηριωμένες αρμοδιότητες, ρόλους και βάση ανάγκης γνώσης (τα οποία εξετάζονται και βελτιώνονται τακτικά).

Ιδιαίτερη σημασία έχει ο ρόλος του Υπεύθυνου Ασφαλείας, ο οποίος είναι υπεύθυνος για την παρακολούθηση της ορθής εφαρμογής της πολιτικής ασφαλείας. Ένας άλλος σημαντικός ρόλος είναι ο Υπεύθυνος Προστασίας

Δεδομένων (DPO), ο οποίος παρακολουθεί τη συμμόρφωση με τον

GDPR και, επομένως, είναι σαφές ότι χρειάζεται επίσης να συνεργάζεται με τον Υπεύθυνο

Ασφαλείας για την κατάλληλη εφαρμογή μέτρων ασφαλείας. Σημειώνεται ότι βάσει του GDPR (άρθρο 37) ο ορισμός DPO είναι υποχρεωτικός για ορισμένους τύπους εργασιών επεξεργασίας δεδομένων (δραστηριότητες παρακολούθησης μεγάλης κλίμακας, επεξεργασία ειδικών κατηγοριών δεδομένων κ.λπ.).

Η κατάταξη των επιπέδων ασφάλειας βάση των επιπέδων κινδύνου κατηγοριοποιούνται ως εξής:

1. Οι ρόλοι και οι ευθύνες που σχετίζονται με την επεξεργασία δεδομένων προσωπικού χαρακτήρα θα πρέπει να ορίζονται σαφώς και να κατανέμονται σύμφωνα με την πολιτική ασφαλείας.

2. Κατά τη διάρκεια εσωτερικών αναδιοργανώσεων ή απολύσεων και αλλαγής εργασίας, η ανάκληση δικαιωμάτων και ευθυνών με αντίστοιχες διαδικασίες παράδοσης θα πρέπει να ορίζεται με σαφήνεια.

3. Θα πρέπει να πραγματοποιείται σαφής διορισμός ατόμων που είναι επιφορτισμένα με συγκεκριμένα καθήκοντα ασφαλείας, συμπεριλαμβανομένου του διορισμού αξιωματικού ασφαλείας.

4. Ο υπεύθυνος ασφαλείας θα πρέπει να διοριστεί επίσημα (τεκμηριωμένο). Τα καθήκοντα και οι ευθύνες του αξιωματικού ασφαλείας θα πρέπει επίσης να ορίζονται και να τεκμηριώνονται με σαφήνεια.

5. Τα αντικρουόμενα καθήκοντα και οι τομείς ευθύνης, για παράδειγμα, οι ρόλοι του υπεύθυνου ασφαλείας, του ελεγκτή ασφαλείας και του DPO, θα πρέπει να θεωρούνται διαχωρισμένοι για να μειωθούν οι ευκαιρίες για μη εξουσιοδοτημένη ή ακούσια τροποποίηση ή κατάχρηση προσωπικών δεδομένων.

5.2.3 Πολιτική ελέγχου πρόσβασης

Μετά τον ορισμό των ρόλων και των ευθυνών, είναι σημαντικό να καθοριστεί μια πολιτική ελέγχου πρόσβασης στα συστήματα που χρησιμοποιούνται για την επεξεργασία δεδομένων προσωπικού χαρακτήρα. Αυτό θα πρέπει να βασίζεται στην αρχή της «ανάγκης γνώσης», δηλαδή κάθε ρόλος/χρήστης θα πρέπει να έχει μόνο το επίπεδο πρόσβασης στα προσωπικά δεδομένα που είναι απολύτως απαραίτητο για την εκτέλεση των σχετικών καθηκόντων του. Αυτή είναι μια κεντρική έννοια και στον GDPR και σχετίζεται στενά με την αρχή της ελαχιστοποίησης δεδομένων (άρθρο 5(γ) GDPR).

Η πολιτική ελέγχου πρόσβασης θα εφαρμοστεί με επακόλουθα τεχνικά μέτρα (βλ. επίσης 4.2.1 σε αυτό το έγγραφο).

Η κατάταξη των επιπέδων πρόσβασης βάση των επιπέδων κινδύνου κατηγοριοποιούνται ως εξής:

1. Θα πρέπει να εκχωρούνται συγκεκριμένα δικαιώματα ελέγχου πρόσβασης σε κάθε ρόλο (που εμπλέκεται στην επεξεργασία δεδομένων προσωπικού χαρακτήρα) σύμφωνα με την αρχή της ανάγκης γνώσης.

2. Μια πολιτική ελέγχου πρόσβασης πρέπει να είναι λεπτομερής και τεκμηριωμένη. Ο οργανισμός θα πρέπει να καθορίσει σε αυτό το έγγραφο τους κατάλληλους κανόνες ελέγχου πρόσβασης, δικαιώματα πρόσβασης και περιορισμούς για συγκεκριμένους ρόλους χρηστών στις διαδικασίες και διαδικασίες που σχετίζονται με προσωπικά δεδομένα.

3. Ο διαχωρισμός των ρόλων ελέγχου πρόσβασης (π.χ. αίτημα πρόσβασης, εξουσιοδότηση πρόσβασης, διαχείριση πρόσβασης) θα πρέπει να ορίζεται με σαφήνεια και να τεκμηριώνεται. Γ.4 Οι ρόλοι με υπερβολικά δικαιώματα πρόσβασης θα πρέπει να ορίζονται σαφώς και να ανατίθενται σε περιορισμένα συγκεκριμένα μέλη του προσωπικού.

Σχετικό με το ISO 27001:2013 - A.9.1.1 Πολιτική ελέγχου πρόσβασης

5.2.4 Ρόλοι χρηστών και δικαιώματα στο διαδικτυακό σύστημα IonianWeather

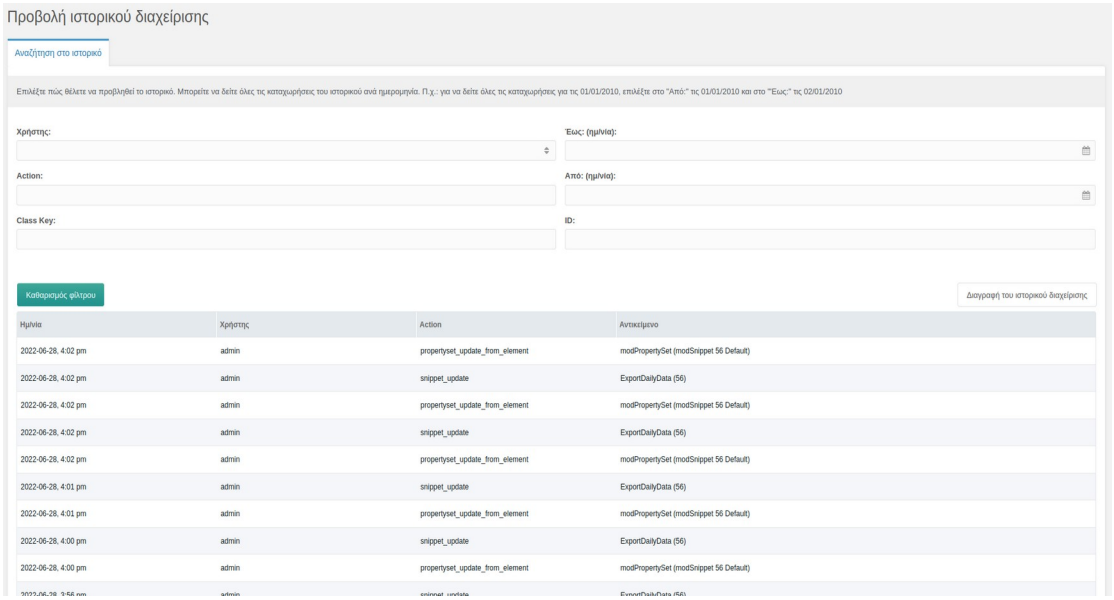
Στο παρόν σύστημα υπάρχουν 3εις ρόλοι με ανάλογες ανάγκες διαχείρισης δικαιωμάτων και προσωπικών δεδομένων.

1. Απλός επισκέπτης. Για τον απλό επισκέπτη το σύστημα δεν απαιτεί την συλλογή δεδομένων που άπτονται των κανόνων διαχείρισης προσωπικών δεδομένων
2. Εγγεγραμμένοι επισκέπτες. Οι εγγεγραμμένοι επισκέπτες παρέχουν ελάχιστα προσωπικά δεδομένα όπως: Email, και όνομα
3. Διαχειριστής. Ο διαχειριστής έχει πρόσβαση στα δεδομένα των εγγεγραμμένων χρηστών.

5.2.5 Μονάδα παρακολούθησης ενεργειών χρήστη

Από την παραπάνω ανάλυση των χρηστών απαιτείται η καταγραφή των κινήσεων του διαχειριστή έτσι ώστε το σύστημα να υπακούει στους κανονισμούς GTPR. Ένας άλλος λόγος ανάγκης καταγραφής ενεργειών του κάθε διαβαθμισμένου χρήστη είναι η αναγνώριση συμπεριφορών που συνιστούν παραβίαση της ασφάλειας του συστήματος.

Για την κάλυψη αυτών των αναγκών ενεργοποιήθηκε και παραμετροποιήθηκε μονάδα “Προβολής ιστορικού διαχείρισης”.



Στιγμιότυπο από το περιβάλλον της μονάδας “Προβολής ιστορικού διαχείρισης”

Η συγκεκριμένη μονάδα σου δίνει την δυνατότητα να παρακολουθήσεις:

- ποιες ενέργειες εφαρμόστηκαν
- σε ποια μονάδα (Αντικείμενο) του συστήματος εφαρμόστηκαν
- ποιος χρήστης της εφάρμοσε
- πότε εφαρμόστηκαν

5.3 Τι είναι η επίθεση DDos και DoS

Προαναφέρθηκε στις προηγούμενες παραγράφους οτι τα συστήματα ασφαλείας που εφαρμόστηκαν εξασφαλίζουν (σε μεγάλο βαθμό) το σύστημα απο επιθέσεις τύπου DDos και DoS. Αξίζει τον κόπο να περιγράψουμε την φύση αυτών των επιθέσεων

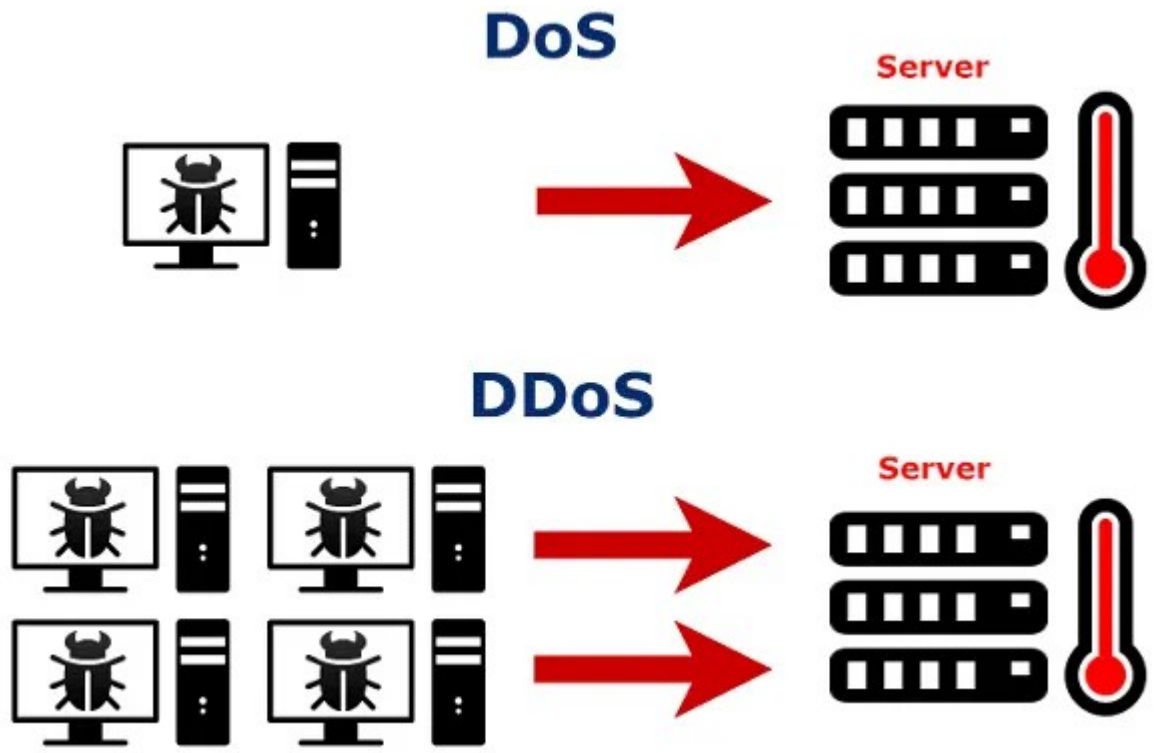
Τα DDoS και DoS είναι συντομογραφίες της φράσης (Distributed) Denial of Service. Στα Ελληνικά μεταφράζεται ως “Κατανεμημένη Επίθεση Άρνησης Εξυπηρέτησης”.

Μια επίθεση DDoS έχει σαν στόχο ένα σύστημα, συνήθως έναν server ή ένα ολόκληρο κέντρο δεδομένων (datacenter). Η επίθεση “βομβαρδίζει” το σύστημα με ένα τεράστιο όγκο δεδομένων από διαφορετικές πηγές.

Το σύστημα, στην προσπάθειά του να ανταποκριθεί, καταλήγει να εξαντλήσει τους πόρους του, όπως CPU, RAM, κτλ, με αποτέλεσμα να υπερφορτώνεται και στο τέλος να καταρρεύσει.

Οι επιθέσεις DDoS πρόκειται για αυτοματοποιημένες ενέργειες μέσω software.

Οι ενέργειες αυτές δεν έχουν σκοπό να αξιοποιήσουν τον server, αλλά να τον μπλοκάρουν. Έτσι, εμποδίζουν τους πραγματικούς χρήστες από το να έχουν πρόσβαση στο δίκτυο.



Μάλιστα, επειδή οι ενέργειες έχουν σκοπό να μπλοκάρουν, χρησιμοποιούν μεγαλύτερο όγκο δεδομένων σε σχέση με έναν απλό χρήστη.

Ένας υπολογιστής που συμμετέχει σε μια επίθεση DDoS μπορεί να προξενεί επιβάρυνση ίση με εκατοντάδες ή χιλιάδες κανονικούς χρήστες.

Οι επιθέσεις DDoS και DoS αναφέρονται ουσιαστικά στο ίδιο πράγμα, με μοναδική διαφορά την κλίμακα.

Η επίθεση DoS είναι ένα μικρής κλίμακας κρούσμα, συνήθως από ένα σύστημα-θύτη προς ένα σύστημα-θύμα.

Στις DDoS επιθέσεις η κλίμακα είναι σαφώς μεγαλύτερη, αφού βασίζονται μέχρι και σε εκατοντάδες χιλιάδες συσκευές. Αρκετά μεγάλες επιθέσεις DDoS και μπορούν να ρίξουν ολόκληρα κέντρα δεδομένων με πολλαπλούς server.

6 Αποθετήριο κώδικα και images

6.1 Git

Το συγκεκριμένο έργο έχει 2 βασικές μελλοντικές ανάγκες:

1. Ανάγκη επεκτασιμότητας του κώδικα
2. Ανάγκη άμεσης αποκατάστασης του διαδικτυακού συστήματος IonianWeather

Για την κάλυψη αυτών των αναγκών εφαρμόστηκε το μοντέλο παρακολούθησης συσσώρευσης κώδικα βασισμένο στο λογισμικό GIT

Ένα από τα βασικά χαρακτηριστικά της mellontik;hw ανάπτυξης λογισμικού είναι ότι γράφουμε τον πηγαίο κώδικα σε στάδια. Ξεκινάμε αρχικά με μια συγκεκριμένη έκδοση κώδικα και στην συνέχεια προσθέτουμε, διορθώνουμε και γράφουμε νέο κώδικα αναβάθμισης ή επέκτασης.

Ιδιαίτερη ανάγκη για την παρακολούθηση του κώδικα έχουν συστήματα όπως το σύστημα του IonianWeather λόγω της έκτασης του.

Ένα από τα συστήματα διαχείρισης εκδόσεων λογισμικού είναι το Git, που γράφτηκε αρχικά από τον Linus Torvalds και στηρίζει την ανάπτυξη του πυρήνα Linux, του GNOME και πολλών άλλων έργων ΕΛ/ΛΑΚ. Το Git είναι ένα κατανεμημένο σύστημα διαχείρισης εκδόσεων λογισμικού (Distributed Version Control System, DVCS).

Κάποια από τα σημαντικά χαρακτηριστικά του Git είναι:

- Εύκολο branching
- Εύκολο merging
- Έλεγχος εγκυρότητας των δεδομένων (SHA1)
- Παρακολουθεί το περιεχόμενο συνολικά και όχι ανά αρχείο

- Πολύ γρήγορο

Ορισμοί

- Repository: Το φυσικό σημείο στο οποίο βρίσκονται αποθηκευμένα τα αρχεία μαζί με όλες τις πληροφορίες του RCS (local ή remote).
- Commit: Ένα σύνολο λογικά συσχετισμένων αλλαγών στα παρακολουθούμενα αρχεία (edit, add, delete) μαζί με πληροφορίες για το ποιός τις έκανε,πότε και γιατί.
- Branch: Όταν υπάρχει ανάγκη για απόσπαση από το “κεντρικό σώμα” του κώδικα (π.χ. για ένα experimental feature ή bug fix), ένα branch κρατάει τα νέα commits χωρίς να επηρεάζει το σώμα του κώδικα.
- Tag: Δείκτης σε μια συγκεκριμένη έκδοση του κώδικα.
- Merge: Η διαδικασία ενσωμάτωσης ενός branch σε ένα άλλο.

7 Αρχιτεκτονική συστήματος

Η βασική δομή του συστήματος αναπτύχθηκε με την γλώσσα προγραμματισμού Typescript μέσω του εργαλείου NestJS.

7.1 Βασικές κλάσεις συστήματος

7.1.1 Κλάσεις Μετεωρολογικών δεδομένων

Η κλάση **class WeatherDatum** είναι η κύρια κλάση των μετρολογικών δεδομένων και αναπτύχθηκε με σκοπό την αναπαράσταση των μετρολογικών δεδομένων αυτών στην βάση δεδομένων. Στην κλάση αυτή τα δεδομένα αποθηκευονται μετά από επεξεργασία των δεδομένων όπως λαμβάνονται από τους αισθητήρες των μετεωρολογικών σταθμών (Παράρτημα 8.1).

Μέσω της κλάση **class WeatherDataService** αποθηκεύονται και ανακτούνται αυτά τα μετεωρολογικά πρωτογενή και δευτερογενή δεδομένα (Παράρτημα 8.2). Η επεξεργασία των δευτερογενών δεδομένων γίνονται μέσα από την κλάση αυτή.

Η κλάση **class Channel** αναπτύχθηκε με σκοπό την αναπαράσταση των μετρολογικών δεδομένων στην βάση δεδομένων όπως αυτά λαμβάνονται από τα καταγραφικά των μετεωρολογικών σταθμών μέσω του Diameson χωρίς καμία απολύτως επεξεργασία (Παράρτημα 8.3).

Η κλάση **class ChannelService** αναπτύχθηκε με σκοπό την επεξεργασία των πρωτογενών δεδομένων από την κλάση Channel και την αποθήκευση τους στην κλάση WeatherDatumμέσω της κλάσης WeatherDataService (Παράρτημα).

7.1.2 Κλάση Μετεωρολογικών σταθμών

Η κλάση **class Station** αναπτύχθηκε με σκοπό την αναπαράσταση των

μετρολογικών σταθμών στην βάση δεδομένων (Παράρτημα 8.5). Συνδέεται άμεσα με τις κλάσεις μετεωρολογικών δεδομένων με σκοπό το σύστημα να γνωρίζει από ποιο καταγραφικό έχουν συλλεχθεί τα δεδομένα τα οποία θα αποθηκευτούν στην βάση δεδομένων.

8 Παράρτημα

Στο συγκεκριμένο παράρτημα παρουσιάζονται κάποια (βασικά) επιλεγμένα βασικά κομμάτια του κώδικα που αναπτύχθηκαν στα πλαίσια του συγκεκριμένου παραδοτέου.

Τα ολοκληρωμένα πακέτα κώδικα (όλος ο πηγαίος κώδικας) έχουν παραδοθεί μέσο του αποθετηρίου GIT έτοιμα προς εγκατάσταση.

Διεύθυνση GIT

https://gitlab.com/sofargr/ionian_weather

8.1 Κλάση Αναπαράστασης Μετεωρολογικών δεδομένων

- Type script κώδικας περιγραφής της βασικής κλάσης δεδομένων που χρησιμοποιείτε για την καταγραφή/καταχώρηση μετεωρολογικών δεδομένων μετά την λήψη απο το Buffer μετεωρολογικών δεδομένων

```
@Entity({ name: 'iw_weatherdata2' })
export class WeatherDatum {

    @PrimaryGeneratedColumn()
    id: number;

    @Column({
        name: 'software_channel_sn',
        length: 20,
    })
    sn: string;

    @Column({
        name: 'software_channel_datetime',
    })
    date: Date;

    @Column({
        name: 'software_channel_name',
        length: 10,
    })
    name: string;

    @Column({
        name: 'software_channel_value',
        type: 'float',
        precision: 10,
        scale: 6,
    })
    value: number;

    @ManyToOne(type => Station )
    @JoinColumn({
        name : 'software_channel_sn',
        referencedColumnName : 'sn'
    })
    station : Station
}
```

8.2 Κλάση επεξεργασίας δευτερογενών μετεωρολογικών δεδομένων

- Type script κώδικας περιγραφής της κλάσης δεδομένων που χρησιμοποιείτε για την επεξεργασία και καταγραφή/καταχώρηση μετεωρολογικών δεδομένων. Οι βασικές λειτουργίες της συγκεκριμένης κλάσης είναι να επεξεργαστή τα πρωτογενή δεδομένα και να δημιουργήσει νέα δευτερογενή δεδομένα (π.χ. παραγωγή παραμέτρου ημερήσιας βροχόπτωσης).
- Η κλάση τις απαραίτητες λειτουργίες (μετά την επεξεργασία των δεδομένων) παροχής του ανάλογου σχήματος για την εισαγωγή των δεδομένων στην βάση δεδομένων.

```
export class WeatherDataService {
  private readonly logger = new Logger(WeatherDataService.name);

  constructor(
    @InjectRepository(WeatherDatum)
    public weatherDataRepository: Repository<WeatherDatum>,

    @InjectRepository(Station)
    public stationRepository: Repository<Station>,
  ) {}

  create(weatherDatumDto: UpdateWeatherDatumDto): WeatherDatum {
    const entity = this.weatherDataRepository.create(weatherDatumDto);

    return this.preprocess(entity);
  }

  async getSecondaryParams(primaryData): Promise<WeatherDatum[]> {
    const secondaryParams: UpdateWeatherDatumDto[] = [];

    const data = primaryData.reduce(
      (prev, curr) => ({ ...prev, [curr.name]: curr.value }),
      {},
    );
    const record: UpdateWeatherDatumDto = {
      sn: primaryData[0].sn,
```

```
      date: primaryData[0].date,
    };

    const station = await this.stationRepository.findOne({
      where: { sn: record.sn },
    });

    if (!station) return [];

    for (const param of ['WD', 'AP', 'RH']) {
      if (!data[param]) {
        const v = await this.getRelatedStationsValue(station, param);
        if (v !== null) {
          data[param] = v;
          secondaryParams.push({
            ...record,
            name: param,
            value: data[param],
          });
        }
      }
    }

    if (data['AP'] !== undefined) {
      const FP =
        1.0016 + 3.15 * Math.pow(10, -6) * data['AP'] - 0.074 / data['AP'];

      const a = 7.5;
      const b = 237.3;
      const exp = (a * data['AT']) / (b + data['AT']);

      const ES = 6.112 * FP * Math.pow(10, exp);
      const PVAP = (data['RH'] * ES) / 100;

      const AH = (216.688 * PVAP) / (data['AT'] + 273);

      if (!Number.isNaN(AH)) {
        secondaryParams.push({
          ...record,
          name: 'AH',
          value: AH,
        });
      }
    }

    return secondaryParams.map((e) => this.create(e));
  }
```

```
async getRelatedStationsValue(
  station: Station,
  param: string,
): Promise<number> {
  const wd = await this.getLatestData(param, 'ZKT-3');

  const zkt3 = wd !== undefined;
  let stations = {};

  switch (param) {
    case 'WD':
      switch (station.code) {
        case 'KTL-1':
          stations = zkt3
            ? { 'ZKT-1': 0.74, 'ZKT-4': 0.18, 'ZKT-3': 0.08 }
            : { 'ZKT-1': 0.74, 'ZKT-4': 0.26 /*, 'ZKT-3' : 0.08*/ };
          break;
        default:
          stations = [];
          break;
      }
      break;
    case 'AP':
      switch (station.code) {
        case 'KTL-1':
          stations = zkt3
            ? { 'ZKT-3': 0.62, 'ZKT-1': 0.24, 'KEF-3': 0.14 }
            : { 'ZKT-1': 0.24, 'KEF-3': 0.76 };
          break;
        case 'KEF-1':
          stations = { 'KEF-2': 0.35, 'KEF-3': 0.35, 'LFK-1': 0.3 };
          break;
        case 'CRF-1':
          stations = { 'CRF-2': 0.35, 'CRF-4': 0.35, 'CRF-3': 0.3 };
          break;
        case 'ZKT-2':
        case 'ZKT-4':
          stations = zkt3
            ? { 'ZKT-1': 0.59, 'ZKT-3': 0.32, 'KEF-3': 0.09 }
            : { 'ZKT-1': 0.59, 'KEF-3': 0.41 };
          break;
        default:
          stations = [];
          break;
      }
      break;
    case 'RH':
      switch (station.code) {
```

```
        case 'KEF-2':
          stations = zkt3
            ? { 'KEF-1': 0.72, 'ZKT-1': 0.21, 'ZKT-3': 0.07 }
            : { 'KEF-1': 0.72, 'ZKT-1': 0.28 };
          break;
        case 'CRF-2':
          stations = { 'CRF-3': 0.58, 'CRF-4': 0.28, 'CRF-1': 0.14 };
          break;
        default:
          stations = [];
          break;
      }
      break;
    default:
      stations = [];
      break;
  }

  const values = [];
  for (const [code, weight] of Object.entries(stations)) {
    const wd = await this.getLatestData(param, code);
    if (wd !== undefined) {
      values.push(wd.value * (weight as number));
    }
  }

  if (
    Object.keys(stations).length > 0 &&
    values.length === Object.keys(stations).length
  ) {
    return values.reduce((a, b) => a + b, 0).toFixed(2);
  }

  return null;
}

async getLatestData(param, code) {
  const [wd] = await this.weatherDataRepository.find({
    relations: ['station'],
    where: {
      date: MoreThan(sub(new Date(), { hours: 1 })),
      name: param,
      station: { code },
    },
    order: {
      date: 'DESC',
    },
    take: 1,
```

```
    });

    return wd;
  }

  preprocess(entity: WeatherDatum) {
    const { sn, name } = entity;

    if (name === 'Ch2' && sn === '041A0264') {
      entity.name = 'RH';
    } else if (
      name === 'Ch6' &&
      ['041A0258', '041A0263', '020A0022', '041A0270'].includes(sn)
    ) {
      entity.name = 'UVB';
    } else {
      entity.name = this.getChannelName(name);
    }

    return entity;
  }

  getChannelName(name) {
    return (
      {
        Ch1: 'RT',
        Ch2: 'UVA',
        Ch3: 'AT',
        Ch4: 'RH',
        Ch5: 'SR',
        Ch6: 'AP',
        Ch7: 'WD',
        Ch8: 'WS',
        Ch9: 'UB',
        'Ch8-max': 'MaxWS',
      }[name] || name
    );
  }
}
```

8.3 Κλάση πρωτογενών μετεωρολογικών δεδομένων

- Type script κώδικας περιγραφής της κλάσης δεδομένων που χρησιμοποιείτε για την γεφύρωση της βάσης μετεωρολογικών δεδομένων με την βάση Buffer συλλογής δεδομένων (Diameson) η οποία αποθηκεύει προσωρινά τα δεδομένα όπως αυτά παράγονται απο τους data loggers των μετεωρολογικών σταθμών

```
@Entity()
export class Channel {

    @PrimaryGeneratedColumn()
    channel_id: number;

    @Column({ length: 8 })
    channel_sn: string;

    @Column({ length: 8 })
    channel_pwd: string;

    @Column({ length: 8 })
    channel_name: string;

    @Column({ length: 8 })
    channel_description: string;

    @Column({ type : 'double' })
    channel_value: number;

    @Column({ length: 16 })
```

```
    channel_unit: string;

    @Column({ length: 32 })
    channel_tag: string;

    @Column({ type : 'datetime' })
    channel_datetime: Date;

    @Column()
    channel_type: number;
}
```

8.4 Κλάση επεξεργασίας πρωτογενών μετεωρολογικών δεδομένων

- Type script κώδικας περιγραφής της κλάσης που χρησιμοποιείτε για την μεταεπεξεργασία των δεδομένων οπως αυτά παρέχονται απο το Buffer Dimeson έτσι ώστε να παραχθούν οντότητες τύπου `WeatherDatum`

```
export class ChannelService {
  private readonly logger = new Logger(ChannelService.name);

  constructor(
    private readonly httpService: HttpService,

    @InjectRepository(Channel, 'diameson')
    private readonly channelsRepository: Repository<Channel>,

    private readonly weatherDataService: WeatherDataService,

    private readonly dataSource: DataSource,

    @InjectDataSource('diameson')
    private readonly dataSource2: DataSource,
  ) {}

  findAll(): Promise<Channel[]> {
    return this.channelsRepository.find({
      skip: 0,
      take: 20,
    });
  }
```

```
async remove(id: string): Promise<void> {
  await this.channelsRepository.delete(id);
}

@Cron('0/2 * * * * *')
async handleCron() {
  const queryRunner = this.dataSource.createQueryRunner();
  await queryRunner.connect();
  await queryRunner.startTransaction();

  const queryRunner2 = this.dataSource2.createQueryRunner();
  await queryRunner2.connect();
  await queryRunner2.startTransaction();

  try {
    const [root] = await this.channelsRepository.find({
      take: 1,
      order: { channel_id: 'ASC' },
    });

    if (!root) throw new Error('No Diameson Data');

    const data = await this.channelsRepository.find({
      where: {
        channel_sn: root.channel_sn,
        channel_datetime: root.channel_datetime,
      },
      order: { channel_id: 'ASC' },
    });
```

```
    const primaryData = data.map((d) => {
      return this.weatherDataService.create({
        sn: d.channel_sn,
        date: d.channel_datetime,
        name: d.channel_name,
        value: d.channel_value,
      });
    });

    const secondaryData = await this.weatherDataService.getSecondaryParams(
      primaryData,
    );

    await queryRunner.manager.save([...primaryData, ...secondaryData])
    await queryRunner2.manager.remove(data);

    await queryRunner.commitTransaction();
    await queryRunner2.commitTransaction();
  } catch (err) {

    this.logger.error(err);

    await queryRunner.rollbackTransaction();
    await queryRunner2.rollbackTransaction();

  } finally {

    await queryRunner.release();
    await queryRunner2.release();
```

```
        }
      }
    }
```

8.5 Κλάση μετεωρολογικών σταθμών

- Type script κώδικας περιγραφής της κλάσης σχήματος αποτύπωσης μετωρολογικών σταθμών

```
@Entity({ name: 'iw_stations' })
export class Station {

    @PrimaryGeneratedColumn()
    id: number;

    @Column({ type : 'text'})
    name: string;

    @Column({ type : 'int'})
    latitude: number

    @Column({ type : 'int'})
    longtitude: number

    @Column({ length: 20 })
    sn: string

    @Column({ length: 5 })
    code: string

    @Column({ type : 'text' })
    info: string

    @Column({ type : 'int'})
```

```
        sort: number
    }
```

8.6 Server Dockerfile

- Κώδικας παραγωγής container φιλοξενίας και λειτουργίας της μονάδας server appication IonianWeather

```
#----------------development-------------------------#

FROM node:17-alpine AS development

WORKDIR /app

COPY package*.json ./

RUN npm ci


COPY . .

RUN npm run build


#----------------production--------------------------#

FROM node:17-alpine AS production

WORKDIR /app

ARG NODE_ENV=production

ENV NODE_ENV=${NODE_ENV}

COPY package*.json ./

RUN npm ci --production

COPY --from=development /app/dist ./dist

COPY .env .env

CMD ["node", "dist/main"]
```

8.7 Nginx Dockerfile

- Κώδικας παραγωγής container φιλοξενίας και λειτουργίας της μονάδας web server Nginx. Η συγκεκριμένη μονάδα είναι απαρίτητη για την παροχή ασφαλής πρόσβαση στο σύστημα IonianWeather.

```
FROM nginx:alpine as builder


ARG ENABLED_MODULES


RUN set -ex \
    && if [ "$ENABLED_MODULES" = "" ]; then \
        echo "No additional modules enabled, exiting"; \
        exit 1; \
    fi


RUN set -ex \
    && apk update \
    && apk add linux-headers openssl-dev pcre2-dev zlib-dev openssl abuild \
               musl-dev libxslt libxml2-utils make mercurial gcc unzip git \
               xz g++ coreutils \
    # allow abuild as a root user \
    && printf "#!/bin/sh\\nSETFATTR=true /usr/bin/abuild -F \"\$@\"\\n" >
/usr/local/bin/abuild \
    && chmod +x /usr/local/bin/abuild \
    && hg clone -r ${NGINX_VERSION}-${PKG_RELEASE} https://hg.nginx.org/pkg-
oss/ \
    && cd pkg-oss \
```

```
        && mkdir /tmp/packages \
        && for module in $ENABLED_MODULES; do \
            echo "Building $module for nginx-$NGINX_VERSION"; \
            if [ -d /modules/$module ]; then \
                echo "Building $module from user-supplied sources"; \
                # check if module sources file is there and not empty
                if [ ! -s /modules/$module/source ]; then \
                    echo "No source file for $module in modules/$module/source,
exiting"; \
                    exit 1; \
                fi; \
                # some modules require build dependencies
                if [ -f /modules/$module/build-deps ]; then \
                    echo "Installing $module build dependencies"; \
                    apk update && apk add $(cat /modules/$module/build-deps |
xargs); \
                fi; \
                # if a module has a build dependency that is not in a distro,
provide a
                # shell script to fetch/build/install those
                # note that shared libraries produced as a result of this script
will
                # not be copied from the builder image to the main one so build
static
                if [ -x /modules/$module/prebuild ]; then \
                    echo "Running prebuild script for $module"; \
                    /modules/$module/prebuild; \
```

```
                fi; \

                /pkg-oss/build_module.sh -v $NGINX_VERSION -f -y -o /tmp/packages
-n $module $(cat /modules/$module/source); \

                BUILT_MODULES="$BUILT_MODULES $(echo $module | tr '[A-Z]' '[a-z]'
| tr -d '[/_\-\.\t ]')"; \

            elif make -C /pkg-oss/alpine list | grep -E "^$module\s+\d+" >
/dev/null; then \

                echo "Building $module from pkg-oss sources"; \

                cd /pkg-oss/alpine; \

                make abuild-module-$module BASE_VERSION=$NGINX_VERSION
NGINX_VERSION=$NGINX_VERSION; \

                apk add $(. ./abuild-module-$module/APKBUILD; echo $makedepends;);
\

                make module-$module BASE_VERSION=$NGINX_VERSION
NGINX_VERSION=$NGINX_VERSION; \

                find ~/packages -type f -name "*.apk" -exec mv -v {}
/tmp/packages/ \;; \

                BUILT_MODULES="$BUILT_MODULES $module"; \

            else \

                echo "Don't know how to build $module module, exiting"; \

                exit 1; \

            fi; \

        done \

        && echo "BUILT_MODULES=\"$BUILT_MODULES\"" > /tmp/packages/modules.env


    FROM nginx:alpine

    COPY --from=builder /tmp/packages /tmp/packages

    RUN set -ex \
```

```
        && . /tmp/packages/modules.env \
        && for module in $BUILT_MODULES; do \
               apk add --no-cache --allow-untrusted /tmp/packages/nginx-module-${module}-${NGINX_VERSION}*.apk; \
           done \
        && rm -rf /tmp/packages


    RUN apk update \
        && apk add --no-cache --virtual .build-deps openssl \
        && openssl dhparam -dsaparam -out /etc/ssl/dhparam.pem 4096 \
        && apk del .build-deps


    WORKDIR /var/www
```

8.8 Php Dockerfile

- Κώδικας παραγωγής container φιλοξενίας του προγραμματιστικού περιβάλλοντος PHP

```
FROM php:8.1-fpm-alpine


COPY --from=mlocati/php-extension-installer /usr/bin/install-php-extensions
/usr/local/bin/


RUN apk add --no-cache --virtual .build-deps $PHPIZE_DEPS \
    && pecl install uploadprogress \
    && docker-php-ext-enable uploadprogress \
    && apk del .build-deps $PHPIZE_DEPS \
    && install-php-extensions bcmath \
            bz2 \
            calendar \
            curl \
            exif \
            fileinfo \
            ftp \
            gd \
            gettext \
            imagick \
            imap \
            intl \
            ldap \
```

```
                mbstring \
                mcrypt \
                # memcached \
                # mongodb \
                mysqli \
                opcache \
                openssl \
                pdo \
                pdo_mysql \
                # redis \
                soap \
                sodium \
                sysvsem \
                sysvshm \
                xmlrpc \
                xsl \
                zip
    RUN apk add --no-cache --virtual .shadow-deps \
            --repository http://dl-3.alpinelinux.org/alpine/edge/community/ --
allow-untrusted \
            shadow \
        && usermod -u 1000 www-data \
        && groupmod -g 1000 www-data \
        && apk del .shadow-deps
```

8.9 Docker compose

- Κώδικας ανάπτυξης των docker container

```
version: "3"

services:

  web:

    container_name: nginx

    build:

      context: .

      dockerfile: nginx/Dockerfile

      args:

          ENABLED_MODULES: brotli

    volumes:

      - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro

      - ./nginx/proxy.conf:/etc/nginx/proxy.conf:ro

      - ./nginx/conf.d/:/etc/nginx/conf.d/:ro

      - ./letsencrypt/conf/:/etc/nginx/ssl/:ro

      - ./ionianweather/:/var/www/ionianweather/

      - ./logs/nginx:/var/logs/nginx

    ports:

      - "80:80"

      - "443:443"

    restart: always

    depends_on:

      - php

    networks:
```

```
      - ionianweather
  php:
    build:
      context: .
      dockerfile: php/Dockerfile
    volumes:
      - ./ionianweather/:/var/www/ionianweather/
      - ./logs/php.log:/var/log/fpm-php.www.log
    restart: always
    networks:
      - ionianweather

networks:
  ionianweather:
    driver: bridge
```