ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Διπλωματική Εργασία

Ανάπτυξη Εφαρμογών Έξυπνης Γεωργίας με χρήση Μικροϋπηρεσιών και Εικονικοποίησης βασισμένης σε Περιέκτες

Συγγραφέας: Κωνσταντίνος Χουτιούδης
Αριθμός Μητρώου: 711141296

Επιβλέπων Καθηγητής: Βασίλειος Μάμαλης
Συν-επιβλέπων: Απόστολος Αναγνωστόπουλος

ΑΘΗΝΑ, ΙΟΥΛΙΟΣ 2024

University of West Attica Department of Informatics and Computer Engineering

Thesis

Development of Smart Agriculture Applications with use of Microservices and Container-based Virtualization

Author: Konstantinos Choutioudis
Registration Number: 711141296

Supervising Professor: Vasileios Mamalis
Co-supervising Professor: Apostolos Anagnostopoulos

ATHENS, JULY 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ ΤΜΗΜΑ
ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Διπλωματική Εργασία

Ανάπτυξη Εφαρμογών Έξυπνης Γεωργίας με χρήση Μικροϋπηρεσιών και Εικονικοποίησης βασισμένης σε Περιέκτες

Συγγραφέας: Κωνσταντίνος Χουτιούδης
Αριθμός Μητρώου: 711141296
Επιβλέπων Καθηγητής: Βασίλειος Μάμαλης
Συν-επιβλέπων: Απόστολος Αναγνωστόπουλος

Η διπλωματική εργασία εξετάστηκε την Παρασκευή 26/07/2024 επιτυχώς από την κάτωθι Εξεταστική Επιτροπή:

Α/Α	Ονοματεπώνυμο	Βαθμίδα/Ιδιότητα	Ψηφιακή Υπογραφή
1	Καντζάβελου Ιωάννα	Αναπληρώτρια Καθηγήτρια Πανεπιστημίου Δυτικής Αττικής	IOANNA KANTZAVELOU Digitally signed by IOANNA KANTZAVELOU Date: 2024.07.29 11:33:27 +03'00'
2	Μάμαλης Βασίλειος	Καθηγητής Πανεπιστημίου Δυτικής Αττικής	Vasileios Mamalis Digitally signed by Vasileios Mamalis Date: 2024.07.29 03:03:48 +03'00'
3	Πάντζιου Γραμματή	Καθηγήτρια Πανεπιστημίου Δυτικής Αττικής	GRAMMATI PANTZIOU Digitally signed by GRAMMATI PANTZIOU Date: 2024.07.29 12:43:24 +03'00'

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Κωνσταντίνος Χουτιούδης του Χρήστου με αριθμό μητρώου 711141296 φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής Μηχανικών του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών, δηλώνω υπεύθυνα ότι: «Είμαι συγγραφέας αυτής της πτυχιακής/διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος. Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο Δηλών
(υπογραφή)

…….
Κωνσταντίνος Χουτιούδης

Copyright © Κωνσταντίνος Χουτιούδης, 2024.

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ' ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Πανεπιστημίου Δυτικής Αττικής

Περίληψη

Ο γεωργικός τομέας βρίσκεται σε μια σημαντική μεταμόρφωση που καθοδηγείται από την ενσωμάτωση προηγμένων τεχνολογιών. Αυτή η διπλωματική εργασία, με τίτλο "Ανάπτυξη Εφαρμογών Έξυπνης Γεωργίας με χρήση Μικροϋπηρεσιών και Εικονικοποίησης βασισμένης σε Περιέκτες", εξετάζει πώς οι σύγχρονες τεχνολογίες μπορούν να βελτιώσουν την αποτελεσματικότητα, την κλιμακωσιμότητα και τη βιωσιμότητα των γεωργικών λειτουργιών. Η έρευνα επικεντρώνεται στη χρήση της αρχιτεκτονικής μικροϋπηρεσιών και της εικονικοποίησης με βάση τα containers για την ανάπτυξη εφαρμογών έξυπνης γεωργίας.

Η αρχιτεκτονική μικροϋπηρεσιών επιτρέπει την ανάπτυξη μικρών, ανεξάρτητα αναπτυσσόμενων υπηρεσιών που μπορούν να διαχειριστούν και να κλιμακωθούν αυτόνομα, προάγοντας την ευκινησία και την ανθεκτικότητα στην ανάπτυξη εφαρμογών. Η εικονικοποίηση με βάση τα containers, που επιτρέπει τεχνολογίες όπως το Docker και το Kubernetes, παρέχει ένα ελαφρύ και συνεπές περιβάλλον για την εκτέλεση εφαρμογών σε διαφορετικά υπολογιστικά περιβάλλοντα, από συσκευές στο πεδίο μέχρι κέντρα δεδομένων στο cloud.

Η διπλωματική εργασία εμβαθύνει στις βασικές αρχές της εικονικοποίησης και των μικροϋπηρεσιών, εξετάζοντας τις αρχές, τα οφέλη και τις προκλήσεις τους. Παρουσιάζει επίσης μια ολοκληρωμένη ανασκόπηση των υφιστάμενων εφαρμογών έξυπνης γεωργίας, με ιδιαίτερη έμφαση στις λύσεις της Libelium. Η πλατφόρμα της Libelium ενσωματώνει ασύρματα δίκτυα αισθητήρων με IoT και αναλυτικά δεδομένα μεγάλου όγκου για την παρακολούθηση των περιβαλλοντικών συνθηκών, τη διαχείριση των πόρων και τη βελτίωση της υγείας και της παραγωγικότητας των καλλιεργειών.

Μελέτες περιπτώσεων αναδεικνύουν πρακτικές εφαρμογές των τεχνολογιών της Libelium, συμπεριλαμβανομένης της διαχείρισης άρδευσης ακριβείας, της παρακολούθησης και ελέγχου του κλίματος, της ανίχνευσης παρασίτων και ασθενειών και της περιβαλλοντικής παρακολούθησης. Αυτές οι μελέτες περιπτώσεων δείχνουν πώς η εικονικοποίηση με βάση τα containers και η αρχιτεκτονική μικροϋπηρεσιών μπορούν να αξιοποιηθούν για τη δημιουργία κλιμακωτών, ανθεκτικών και αποτελεσματικών συστημάτων έξυπνης γεωργίας.

Μέσω λεπτομερούς ανάλυσης και παραδειγμάτων, αυτή η διπλωματική εργασία παρέχει γνώσεις σχετικά με την ανάπτυξη και την υλοποίηση εφαρμογών έξυπνης γεωργίας, προσφέροντας έναν οδικό χάρτη για την αξιοποίηση των προηγμένων τεχνολογιών προς βελτίωση των γεωργικών πρακτικών. Η έρευνα υπογραμμίζει τη μετασχηματιστική δυνατότητα των μικροϋπηρεσιών και της εικονικοποίησης με βάση τα containers να οδηγήσουν στην καινοτομία και τη βιωσιμότητα στον γεωργικό τομέα.

Λέξεις-κλειδιά:

Έξυπνη Γεωργία, Αρχιτεκτονική Μικροϋπηρεσιών, Εικονικοποίηση με Βάση τους Περιέκτες, Docker, Kubernetes, Άρδευση Ακριβείας, Παρακολούθηση Κλίματος, Ανίχνευση Παρασίτων, IoT, Αναλυτικά Δεδομένα Μεγάλου Όγκου, Libelium

Abstract

The agricultural sector is undergoing a significant transformation driven by the integration of advanced technologies. This thesis, titled "Development of Smart Agriculture Applications with use of Microservices and Container-based Virtualization," explores how modern technologies can enhance the efficiency, scalability, and sustainability of agricultural operations. The research focuses on the use of microservices architecture and container-based virtualization to develop smart agriculture applications.

Microservices architecture allows for the development of small, independently deployable services that can be managed and scaled autonomously, promoting agility and resilience in application deployment. Container-based virtualization, enabled by technologies such as Docker and Kubernetes, provides a lightweight and consistent environment for running applications across diverse computing environments, from edge devices in the field to cloud-based data centers.

The thesis delves into the fundamentals of virtualization and microservices, examining their principles, benefits, and challenges. It also presents a comprehensive review of existing smart agriculture applications, with a particular focus on Libelium's solutions. Libelium's platform integrates wireless sensor networks with IoT and big data analytics to monitor environmental conditions, manage resources efficiently, and enhance crop health and productivity.

Case studies highlight practical applications of Libelium's technologies, including precision irrigation management, climate monitoring and control, pest and disease detection, and environmental monitoring. These case studies demonstrate how container-based virtualization and microservices architecture can be leveraged to create scalable, resilient, and efficient smart agriculture systems.

Through detailed analysis and examples, this thesis provides insights into the development and deployment of smart agriculture applications, offering a roadmap for leveraging advanced technologies to improve agricultural practices. The research underscores the transformative potential of microservices and container-based virtualization in driving innovation and sustainability in the agricultural sector.

Keywords:
Smart Agriculture, Microservices Architecture, Container-based Virtualization, Docker, Kubernetes, Precision Irrigation, Climate Monitoring, Pest Detection, IoT, Big Data Analytics, Libelium

Κεφάλαιο 1: Εισαγωγή

1.1 Υπόβαθρο

Η διασταύρωση της τεχνολογίας και της γεωργίας έχει οδηγήσει σε αυτό που είναι πλέον γνωστό ως έξυπνη γεωργία ή γεωργία ακριβείας. Αυτός ο τομέας με γνώμονα την καινοτομία αξιοποιεί προηγμένες τεχνολογίες όπως το Διαδίκτυο των πραγμάτων (IoT), την τεχνητή νοημοσύνη (AI) και την ανάλυση μεγάλων δεδομένων για να αυξήσει την ποσότητα και την ποιότητα των γεωργικών προϊόντων, βελτιστοποιώντας παράλληλα την απαιτούμενη ανθρώπινη εργασία. Μεταξύ των τεχνολογιών που έχουν καθοριστική σημασία για αυτή την επανάσταση, η αρχιτεκτονική μικρουπηρεσιών και η εικονικοποίηση με βάση τα εμπορευματοκιβώτια έχουν αναδειχθεί ως βασικοί παράγοντες για την ανάπτυξη κλιμακούμενων, ανθεκτικών και αποτελεσματικών γεωργικών εφαρμογών.

1.2 Η σημασία της έξυπνης γεωργίας

Η έξυπνη γεωργία δεν είναι απλώς μια τεχνολογική αναβάθμιση. Οι τρέχουσες εξελίξεις στην σύγχρονη κοινωνία αλλά και την τεχνολογία την καθιστούν αναγκαιότητα για την επιβίωση και τη βιωσιμότητα της γεωργίας παγκοσμίως. Με τον παγκόσμιο πληθυσμό να προβλέπεται να φτάσει τα 9,7 δισεκατομμύρια μέχρι το 2050, η παραγωγή τροφίμων πρέπει να αυξηθεί σημαντικά για να καλυφθεί η ζήτηση. Ωστόσο, η αύξηση αυτή πρέπει να γίνει εν όψει της κλιματικής αλλαγής, της εξάντλησης των φυσικών πόρων και της ανάγκης για βιώσιμες γεωργικές πρακτικές. Η έξυπνη γεωργία προσφέρει μια πορεία προς τα εμπρός, επιτρέποντας την ακριβή διαχείριση των πόρων, τη μείωση της σπατάλης και τη βελτίωση των αποδόσεων των καλλιεργειών μέσω αποφάσεων βάση δεδομένων που συλλέγονται. Οι σύγχρονες τεχνολογίες μπορούν να παρέχουν στους αγρότες εκτεταμένες δυνατότητες ελέγχου, παρακολούθησης, σχεδιασμού και εξερεύνησης. Δίνετε η δυνατότητα στους ιδιοκτήτες αγροτικών επιχειρήσεων/αγρότες να οργανώσουν την αποτελεσματική διαχείριση της παραγωγής, ικανοποιώντας τις αυξανόμενες απαιτήσεις των καταναλωτών και δημιουργώντας επίσης ένα φιλικό προς το περιβάλλον περιβάλλον. Οι συσκευές που οδηγούνται από το IoT, συμπεριλαμβανομένων των γεωργικών μηχανημάτων, μπορούν να χρησιμοποιηθούν για τη διαχείριση των τυποποιημένων γεωργικών καταστάσεων. Οι τεχνολογίες που χρησιμοποιούνται στην έξυπνη γεωργία επιτρέπουν επίσης στους αγρότες να ανταλλάσσουν πληροφορίες, να εγκαθιδρύουν συνεργασίες, να αξιολογούν ομότιμα και να αναπτύσσουν ακόμη και άτυπα συστήματα πληροφόρησης που μπορούν να συμπληρώσουν το επίσημο σύστημα πληροφόρησης των ελεγκτικών αρχών. Αυτή η ροή πληροφοριών μεταξύ των γεωργών και μεταξύ των γεωργών και των καταναλωτών θα είναι ανεξάρτητη από την κλίμακα χωρίς περιορισμούς. Επιπλέον οι σύγχρονοι δορυφόροι και τα μη επανδρωμένα αεροσκάφη

μπορούν να συμβάλλουν σημαντικά στη βελτιστοποίηση των γεωργικών εργασιών και να αποτελέσουν ακόμη και βασικό ρόλο στη λήψη αποφάσεων για την παραγωγή, οδηγώντας έτσι σε υψηλότερες αποδόσεις των καλλιεργειών.

1.3 Σκοπός & στόχοι της εργασίας

Ο πρωταρχικός στόχος της παρούσας εργασίας είναι να διερευνήσει την ανάπτυξη εφαρμογών έξυπνης γεωργίας μέσα από το πρίσμα των μικρουπηρεσιών και της container-based εικονικοποίησης. Πιο συγκεκριμένα η εργασία επιδιώκει να:

- Παρέχει μια ολοκληρωμένη κατανόηση της εικονικοποίησης και των τύπων της, με έμφαση στην container-based εικονικοποίηση, όπως εφαρμόζεται στη γεωργική τεχνολογία.
- Εμβαθύνει στην αρχιτεκτονική μικρουπηρεσιών, εξετάζοντας τις αρχές, τα οφέλη και τον ρόλο της στην ανάπτυξη ευέλικτων και κλιμακούμενων εφαρμογών έξυπνης γεωργίας.
- Αναλύσει τις υφιστάμενες εφαρμογές έξυπνης γεωργίας, με ιδιαίτερη έμφαση στις λύσεις έξυπνης γεωργίας της Libelium, για την κατανόηση της αρχιτεκτονικής, της ανάπτυξης και των επιπτώσεών τους.
- Προτείνει ένα πρωτότυπο σχεδιασμό που χρησιμοποιεί μικρουπηρεσίες και container-based εικονικοποίηση, επιδεικνύοντας τις πρακτικές εφαρμογές αυτών των τεχνολογιών στην έξυπνη γεωργία.

1.4 Δομή της εργασίας

Η παρούσα εργασία οργανώνεται σε διάφορα κεφάλαια, καθένα από τα οποία επικεντρώνεται σε μια συγκεκριμένη πτυχή της διασταύρωσης μεταξύ έξυπνης γεωργίας, μικρουπηρεσιών και εικονικοποίησης. Μετά από αυτή την εισαγωγή, η δομή εκτυλίσσεται ως εξής:

- *Κεφάλαιο 2: Βιβλιογραφική ανασκόπηση* - Το κεφάλαιο αυτό εξετάζει την υπάρχουσα βιβλιογραφία σχετικά με τις τεχνολογίες έξυπνης γεωργίας, με έμφαση στην εξέλιξη και την τρέχουσα κατάσταση της εικονικοποίησης και των μικρουπηρεσιών σε αυτόν τον τομέα.
- *Κεφάλαιο 3: Βασικές αρχές της εικονικοποίησης* - Συζητείται η έννοια της εικονικοποίησης, τα είδη της και η σημασία της, με ιδιαίτερη έμφαση στους περιέκτες και την εφαρμογή τους στη γεωργία.
- *Κεφάλαιο 4: Κατανόηση των μικρουπηρεσιών* - Εξετάζει την αρχιτεκτονική των μικρουπηρεσιών, αντιπαραβάλλοντάς την με τα μονολιθικά σχέδια, και αναλύει τα πλεονεκτήματά της για την έξυπνη γεωργία.
- *Κεφάλαιο 5*: Μικρουπηρεσίες στην έξυπνη γεωργία και μια λεπτομερή μελέτη περίπτωσης για τις λύσεις έξυπνης γεωργίας της Libelium, αντίστοιχα.
- *Κεφάλαιο 6:* Ανάλυση δημιουργίας εφαρμογής για τον σκοπό της Διπλωματικής

- *Κεφάλαιο 7:* Βιβλιογραφία
- *Κεφάλαιο 8:* Παράρτημα Α - Περιέχει τον κώδικα του project που έχει υλοποιηθεί στο κεφ. 6

Κεφάλαιο 2: Βιβλιογραφική ανασκόπηση

2.1 Εξέλιξη των γεωργικών τεχνολογιών

Η πορεία των γεωργικών τεχνολογιών χαρακτηρίζεται από συνεχή καινοτομία με στόχο την ενίσχυση της παραγωγικότητας και της βιωσιμότητας. Ιστορικά, η γεωργία εξελίχθηκε μέσα από διάφορες φάσεις, ξεκινώντας από τη χειρωνακτική εργασία και φτάνοντας στην εισαγωγή των μηχανικών εργαλείων, ακολουθούμενη από την Πράσινη Επανάσταση, η οποία έδωσε έμφαση στις ποικιλίες υψηλής απόδοσης και στα συνθετικά λιπάσματα. Σήμερα, γινόμαστε μάρτυρες της τέταρτης γεωργικής επανάστασης ή γεωργίας 4.0, η οποία ενσωματώνει τις ψηφιακές τεχνολογίες στις γεωργικές πρακτικές. Η παρούσα ενότητα διερευνά αυτή την εξέλιξη, εστιάζοντας στα ορόσημα που διαμόρφωσαν τη σύγχρονη γεωργία.

2.1.1 Μηχανικές καινοτομίες: Τρακτέρ, συνδυαστικές μηχανές και άλλα μηχανικά εργαλεία

Στις αρχές του $20_{ού}$ αιώνα σημειώθηκε ένας αξιοσημείωτος μετασχηματισμός στις γεωργικές πρακτικές με την εισαγωγή μηχανικών καινοτομιών που αντικατέστησαν τα παραδοσιακά εργαλεία χειρός και την εργασία των ζώων. Κάποια παραδείγματα τέτοιων μηχανικών καινοτομιών είναι τα τρακτέρ, οι θεριζοαλωνιστικές μηχανές, φυτευτικές και σπαρτικές μηχανές, συστήματα άρδευσης κ.α.

Τα τρακτέρ αποτελούν μια από τις σημαντικότερες μηχανικές εξελίξεις στη γεωργία. Αρχικά εισήχθησαν στα τέλη του 19ου αιώνα, ενώ η ευρεία υιοθέτησή τους έγινε στις αρχές και στα μέσα του 20ού αιώνα. Επέφεραν μια θεμελιώδη αλλαγή στη γεωργία, παρέχοντας μια πηγή ισχύος που δεν περιοριζόταν από την αντοχή της ανθρώπινης ή ζωικής εργασίας. Οι βασικές εξελίξεις που προσέφεραν στη γεωργία έχουν ως αποτέλεσμα τη καλύτερη αποδοτικότητα και τη μεγαλύτερη παραγωγικότητα καθώς αύξησαν την έκταση της γης που μπορούσε να καλλιεργηθεί σε μια ημέρα, μειώνοντας έτσι δραστικά την εργασία που απαιτούνταν για το όργωμα, τη φύτευση και τη συγκομιδή. Οι συνδυαστικές μηχανές ήταν μια άλλη πρωτοποριακή καινοτομία. Ενσωμάτωσαν την κοπή και το αλώνισμα των καλλιεργειών σε μια ενιαία εργασία και ολοκλήρωναν τη συγκομιδή των καλλιεργειών γρηγορότερα και αποτελεσματικότερα εκτελώντας πολλές εργασίες ταυτόχρονα. Με τη πάροδο του χρόνου έχουν υποστεί πολυάριθμες βελτιώσεις, συμπεριλαμβανομένων των αυτόματων συστημάτων διεύθυνσης, των τεχνολογιών παρακολούθησης της απόδοσης και των βελτιώσεων στο χειρισμό των σιτηρών και τη διαχείριση των υπολειμμάτων. Αυτά τα εργαλεία κατέστησαν δυνατή την εκτέλεση πολλαπλών γεωργικών εργασιών με ένα μόνο μηχάνημα, αυξάνοντας περαιτέρω την αποτελεσματικότητα.

2.1.2 Πράσινη ανάπτυξη

Στα μέσα του 20ου αιώνα συντελέστηκε μια καίρια μεταμόρφωση της παγκόσμιας γεωργίας, γνωστή και ως Πράσινη Επανάσταση. Η περίοδος αυτή χαρακτηρίστηκε από σημαντικές αλλαγές και προόδους, κυρίως σε τομείς όπως την αναπαραγωγή φυτών, την ευρεία υιοθέτηση χημικών λιπασμάτων και φυτοφαρμάκων, και την εισαγωγή προηγμένων γεωργικών πρακτικών και μηχανημάτων. Οι αλλαγές αυτές οδήγησαν συλλογικά σε πρωτοφανή αύξηση της φυτικής παραγωγής γενικότερα, ιδίως όμως στις αναπτυσσόμενες χώρες.

Υπήρξαν κάποια βασικά χαρακτηριστικά της που τη καθιέρωσαν τόσο σημαντική και στιγμάτισαν τη γεωργία παγκοσμίως. Ο ακρογωνιαίος λίθος της Πράσινης Επανάστασης ήταν η εισαγωγή γενετικά βελτιωμένων ποικιλιών σε σημαντικές βασικές καλλιέργειες όπως το σιτάρι, το ρύζι και ο αραβόσιτος. Οι ποικιλίες αυτές κατασκευάστηκαν έτσι ώστε να ανταποκρίνονται καλύτερα στα λιπάσματα και να μπορούν έτσι να αποδίδουν καλύτερα υπό βέλτιστες συνθήκες. Συνδυαστικά, μεγάλο ρόλο σε αυτή την ανάπτυξη έπαιξαν τα χημικά λιπάσματα και φυτοφάρμακα. Η χρήση συνθετικών λιπασμάτων αύξησε σημαντικά τη διαθεσιμότητα θρεπτικών στοιχείων για τις καλλιέργειες που εφαρμόστηκαν ενισχύοντας έτσι την ανάπτυξη και την παραγωγικότητα. Ομοίως, τα φυτοφάρμακα έλεγχαν ένα ευρύ φάσμα εντόμων και ασθενειών που αποτελούσαν απειλές για τις καλλιέργειες, μειώνοντας τις απώλειες αυτών και αυξάνοντας περαιτέρω την απόδοση τους. Επιπρόσθετα, τα οι υποδομές άρδευσης ήταν ζωτικής σημασίας για την πράσινη ανάπτυξη. Η επέκταση των υποδομών αυτών αποτέλεσαν μια αξιόπιστη πηγή νερού που επέτρεψε πολλαπλούς καλλιεργητικούς κύκλους ανά έτος, επιτρέποντας στους αγρότες να μεγιστοποιήσουν την παραγωγικότητα της γης τους.

Συνολικά γίνεται αντιληπτή η επίδραση της πράσινης ανάπτυξης σε παγκόσμια κλίμακα τόσο σε οικονομικό, περιβαλλοντολογικό και κοινωνικό τομέα. Η αύξηση της γεωργικής παραγωγής συνέβαλε στη βελτίωση της επισιτιστικής ασφάλειας και στη μείωση της πείνας σε πολλά μέρη του κόσμου. Επιπλέον προώθησε την οικονομική ανάπτυξη στους γεωργικούς τομείς και ενίσχυσε καταλυτικά την αγροτική ανάπτυξη καθώς και μετασχημάτισε τις οικονομίες από την παραγωγή βιοπορισμού σε παραγωγή προσανατολισμένη στην αγορά.

Όμως παρά τις επιτυχίες της, η πράσινη επανάσταση οδήγησε επίσης σε ακούσια περιβαλλοντικά και κοινωνικά ζητήματα. Συνέβαλε μαζί και με άλλους παράγοντες στην σταδιακή υποβάθμιση του εδάφους και στη μείωση της γενετικής ποικιλομορφίας, καθώς και στην αύξηση της ανθεκτικότητας στα φυτοφάρμακα, που αποτελούν απόρροια της αυξημένης παραγωγής. Υπήρξαν και κοινωνικά προβλήματα όπως η ανισότητα λόγω της διαφορετικής ικανότητας των αγροτών να επενδύουν σε νέες τεχνολογίες.

2.1.3 Γεωργία ακριβείας

Η γεωργία ακριβείας η οποία εμφανίστηκε περίπου στα τέλη του 20ού αιώνα αντιπροσωπεύει μια στροφή της γεωργίας που είναι περισσότερο καθοδηγούμενη από δεδομένα και στοχεύει στη διαχείριση των πρακτικών παραγωγής καλλιεργειών.

Οι κύριες τεχνολογίες που χρησιμοποιεί είναι το GPS, αισθητήρες διαφόρων ειδών, δορυφόροι, και μη επανδρωμένα αεροσκάφη (drones), και βοηθούν στο να δημιουργηθεί μια πιο ακριβή και ελεγχόμενη προσέγγιση στις γεωργικές εργασίες. Το παγκόσμιο σύστημα εντοπισμού θέσης ή αλλιώς GPS αποτελεί τεχνολογία ζωτικής σημασίας για τη γεωργία ακριβείας. Βοηθάει στον προγραμματισμό των γεωργικών εκμεταλλεύσεων, τη χαρτογράφηση των αγρών, την ανίχνευση των καλλιεργειών κ.α. Με αυτό το τρόπο οι αγρότες είναι σε θέση να εργάζονται σε συνθήκες χαμηλής ορατότητας, όπως βροχή, σκόνη, ομίχλη και σκοτάδι. Εξίσου σημαντική είναι και η αξιοποίηση των αισθητήρων και των συστημάτων παρακολούθησης στις καλλιέργειες. Διάφοροι αισθητήρες μετρούν την υγρασία του εδάφους, τα θρεπτικά συστατικά, την υγεία των καλλιεργειών και τις καιρικές συνθήκες. Αυτά τα δεδομένα έπειτα συμβάλλουν στη λήψη τεκμηριωμένων αποφάσεων σχετικά με το πότε πρέπει να φυτευτούν, να ποτιστούν, να λιπανθούν και να συγκομιστούν οι καλλιέργειες. Στην ίδια συχνότητα επίσης με το GPS υφίστανται και τα γεωγραφικά συστήματα πληροφοριών γνωστά και ως GIS. Τα GIS χρησιμοποιούνται για την ανάλυση χωρικών και γεωγραφικών δεδομένων. Η τεχνολογία αυτή έχει ως σκοπό τη χαρτογράφηση του εδάφους και την κατανόηση της χωρικής μεταβλητότητας εντός των αγρών, επιτρέποντας έτσι την ακριβέστερη εφαρμογή των εισροών.

Τα οφέλη που προσφέρει η γεωργία ακριβείας είναι αρκετά σημαντικά καθώς οι ανάγκες για τροφή σε παγκόσμιο επίπεδο αυξάνονται συνεχώς. Αυξάνει την παραγωγικότητα και μπορεί να μειώσει την ποσότητα των αποβλήτων, άρα κατ' επέκταση μειώνει το κόστος και τις περιβαλλοντολογικές επιπτώσεις. Τα δεδομένα που συλλέγονται από διάφορες πηγές που αναφέρθηκαν προηγουμένως βοηθούν τους αγρότες να λαμβάνουν καλύτερες αποφάσεις σχετικά με τη διαχείριση των καλλιεργειών τους και να μειώσουν τους κινδύνους που μπορεί να αντιμετωπίσουν. Επιπλέον με τη γεωργία ακριβείας διασφαλίζεται το γεγονός πως οι καλλιέργειες λαμβάνουν ακριβώς αυτό που χρειάζονται για τη βέλτιστη ανάπτυξη και απόδοση, οδηγώντας έτσι σε υψηλότερη παραγωγικότητα.

Παράλληλα με τα οφέλη της εμπίπτουν κάποιες εξίσου σημαντικές προκλήσεις. Η υιοθέτηση της τεχνολογίας της γεωργίας ακριβείας συνεπάγεται σε σημαντικές αρχικές επενδύσεις και συνεχές κόστος, το οποίο μπορεί να είναι απαγορευτικό για τους αγρότες μικρής κλίμακας. Έτσι γίνεται ολοένα και πιο δύσκολο οι αγρότες αυτοί να εξελιχθούν και να επιβιώσουν επαγγελματικά σε ένα άνισα ανταγωνιστικό περιβάλλον. Ο τεράστιος όγκος δεδομένων επίσης που παράγεται από τη γεωργία ακριβείας μπορεί να είναι συντριπτικός και απαιτεί εξελιγμένα εργαλεία και τεχνογνωσία για την αποτελεσματική ανάλυση και χρήση.

Γίνεται αντιληπτό πως τόσο η πράσινη ανάπτυξη όσο και η γεωργία ακριβείας αντιπροσωπεύουν κομβικές στιγμές στην ιστορία της γεωργίας, οι οποίες οφείλονται στην εισαγωγή νέων τεχνολογιών και πρακτικών. Συνδυαστικά κατάφεραν να επικεντρωθούν στην αύξηση των αποδόσεων των καλλιεργειών μέσω χημικών εισροών και ποικιλιών υψηλής απόδοσης, και να χρησιμοποιήσουν προηγμένες τεχνολογίες για τη βελτίωση της αποδοτικότητας και της βιωσιμότητα. Και τα δύο αυτά κινήματα έχουν αναμφισβήτητα διαμορφώσει τις σύγχρονες γεωργικές πρακτικές και την κουλτούρα, αφήνοντας το καθένα μια μοναδική κληρονομιά με οφέλη και προκλήσεις.

2.2 Επισκόπηση της έξυπνης γεωργίας

2.2.1 Ορισμός της έξυπνης γεωργίας

Η έξυπνη γεωργία είναι ένας σχετικά καινούργιος όρος που χρησιμοποιείται στην βιομηχανία των καλλιεργειών. Πρόκειται για την ενσωμάτωση προηγμένων τεχνολογιών - *όπως αναφέρθηκαν και παραπάνω* - στις γεωργικές πρακτικές και στη ρουτίνα των αγροτών, που στοχεύει στην βελτίωση της αποδοτικότητας, της παραγωγικότητας και της βιωσιμότητας των καλλιεργειών με βάση τα δεδομένα.

2.2.2 Βασικές τεχνολογίες

Η έξυπνη γεωργία χρησιμοποιεί μια σειρά καινοτόμων τεχνολογιών που την μετατρέπουν από μια παραδοσιακή σε μια έξυπνη, αποτελεσματική και πιο ελεγχόμενη διαδικασία.

2.2.2.1 Διαδίκτυο των πραγμάτων - IoT

Οι συσκευές IoT χρησιμοποιούνται εκτενώς στην έξυπνη γεωργία και βοηθούν στην παρακολούθηση και στον έλεγχο σε πραγματικό χρόνο. Πρόκειται για ένα δίκτυο φυσικών συσκευών, οχημάτων, και άλλων φυσικών αντικειμένων που είναι ενσωματωμένα με αισθητήρες, λογισμικό και συνδεσιμότητα δικτύου που τους επιτρέπει να συλλέγουν δεδομένα.

Στην περίπτωση της έξυπνης γεωργίας οι συσκευές IoT περιλαμβάνουν πολλά είδη αισθητήρων IoT, συμπεριλαμβανομένων αισθητήρων για την παρακολούθηση των καλλιεργειών, την παρακολούθηση του ζωικού κεφαλαίου και την παρατήρηση της κατάστασης του γεωργικού εξοπλισμού ανα πάσα στιγμή. Οι αισθητήρες αυτοί έχουν την ικανότητα να μετρούν διάφορες περιβαλλοντικές παραμέτρους όπως η θερμοκρασία, η υγρασία του εδάφους και τα επίπεδα θρεπτικών ουσιών. Ως αποτέλεσμα παρέχουν κρίσιμα δεδομένα που μπορούν να χρησιμοποιηθούν για την αυτοματοποίηση γεωργικών εργασιών όπως είναι η άρδευση, η λίπανση και η καταπολέμηση παρασίτων.

2.2.2.2 Τεχνολογία πληροφοριών και επικοινωνιών - ICT

Το Εθνικό Ινστιτούτο Προτύπων και Τεχνολογίας του Υπουργείου Εμπορίου των ΗΠΑ ορίζει την τεχνολογία πληροφοριών και επικοινωνιών (ΤΠΕ) ως τη σύλληψη, την αποθήκευση, την ανάκτηση, την επεξεργασία, την εμφάνιση, την αναπαράσταση, την παρουσίαση, την οργάνωση, τη διαχείριση, την ασφάλεια, τη μεταφορά και την ανταλλαγή δεδομένων και πληροφοριών. Η

συλλογή δεδομένων για οτιδήποτε μπορεί να συντελέσει θετικά, από την περιεκτικότητα του εδάφους έως τις καιρικές συνθήκες, έχει γίνει βασική πτυχή της έξυπνης γεωργίας και οι ΤΠΕ βοηθούν τους αγρότες να οργανώσουν και να μεταφέρουν αυτά τα δεδομένα.

2.2.2.3 Γεωγραφικά συστήματα πληροφοριών - GIS & Παγκόσμια συστήματα εντοπισμού θέσης - GPS

Τα γεωγραφικά συστήματα πληροφοριών ή αλλιώς GIS χρησιμοποιούνται για τη χαρτογράφηση και την ανάλυση της γης και των γεωργικών αποτελεσμάτων που παράγονται. Συλλέγουν, αποθηκεύουν, ελέγχουν και εμφανίζουν δεδομένα τα οποία σχετίζονται με θέσεις στην επιφάνεια της γής. Από την άλλη τα συστήματα εντοπισμού θέσης, εναλλακτικά GPS βοηθούν στην ακριβή χαρτογράφηση των αγρών, στην ανίχνευση των καλλιεργειών και στην αποτελεσματική συγκομιδή. Το GPS επιτρέπει επίσης τη φύτευση και τη λίπανση ακριβείας καθώς παρέχει ακριβείς τοποθεσίες για τη βελτιστοποιημένη εφαρμογή εισροών.

2.2.2.4 Μη επανδρωμένα αεροσκάφη και εναέρια απεικόνιση

Τα μη επανδρωμένα αεροσκάφη (drones) που χρησιμοποιούνται στην έξυπνη γεωργία είναι εξοπλισμένα με διάφορους αισθητήρες συμπεριλαμβανομένων και πολυφασματικών καμερών. Προσφέρουν λήψη εικόνων υψηλής ανάλυσης που είναι ζωτικής σημασίας για την παρακολούθηση της υγείας των καλλιεργειών, τη διαχείριση της άρδευσης και την ανίχνευση ασθενειών. Επιπλέον αυτή η εναέρια όψη που προσφέρουν επιτρέπει την αξιολόγηση της ευρωστίας των καλλιεργειών και την έγκαιρη ανίχνευση πιθανών προβλημάτων που δεν είναι εύκολα ορατά από το έδαφος.

2.2.2.5 Ρομποτική και αυτοματισμοί

Ο αυτοματισμός και η ρομποτική κατέχουν μια εξέχουσα θέση στις σύγχρονες πρακτικές της έξυπνης γεωργίας. Χρησιμοποιούνται όλο και περισσότερο για εργασίες όπως η φύτευση, η συγκομιδή, το βοτάνισμα και ο ψεκασμός, με τρόπο που μπορεί να είναι πιο αποτελεσματικός και ακριβής από τις παραδοσιακές μεθόδους. Ειδικότερα, με την ακριβέστερη και πιο περιορισμένη εφαρμογή λιπασμάτων που πετυχαίνουμε χάρη αυτών οδηγούμαστε στο να έχουμε ένα αξιοσημείωτο περιβαλλοντικό αντίκτυπο.

2.2.2.6 Τεχνητή νοημοσύνη - AI & Μηχανική μάθηση -ML

Η τεχνητή νοημοσύνη και η μηχανική μάθηση (ML) μπορούν να βοηθήσουν τους αγρότες να αντλήσουν πληροφορίες από τα μεγάλα δεδομένα - *μεγάλα και πολύπλοκα σύνολα δεδομένων* -

που συλλέγονται από συσκευές IoT, drones και άλλους αισθητήρες. Είναι σε θέση να κάνουν προγνωστικές αναλύσεις σχετικά με την υγεία των καλλιεργειών, τις προβλέψεις απόδοσης και τους βέλτιστους χρόνους φύτευσης. Επιπλέον Οι αλγόριθμοι μηχανικής μάθησης μαθαίνουν από ιστορικά δεδομένα για να βελτιώνουν συνεχώς τις προβλέψεις και τις συστάσεις για τις γεωργικές εργασίες.

2.3 Εφαρμογές στην έξυπνη γεωργία

2.3.1 Μελέτες περιπτώσεων

Μέχρι τώρα το Industrial Internet of Things (IIoT) έχει διαταράξει πολλές βιομηχανίες και η βιομηχανία της γεωργίας δεν αποτελεί εξαίρεση. Ωστόσο, στη γεωργία, το IoT έχει φέρει το μεγαλύτερο αντίκτυπο. Πρόσφατα στατιστικά στοιχεία αποκαλύπτουν ότι ο παγκόσμιος πληθυσμός πρόκειται να φτάσει τα 9,6 δισεκατομμύρια μέχρι το 2050. Έτσι για να τροφοδοτήσει αυτόν τον τεράστιο πληθυσμό, η αγροτική βιομηχανία είναι υποχρεωμένη να υιοθετήσει το Διαδίκτυο των Πραγμάτων. Μεταξύ των προκλήσεων όπως οι ακραίες καιρικές συνθήκες, κλιματικές αλλαγές και περιβαλλοντικές επιπτώσεις, το IoT εξαλείφει αυτές τις προκλήσεις και μας βοηθά να ανταποκριθούμε στη ζήτηση για περισσότερα τρόφιμα.

Η εισαγωγή αισθητήρων στις αγροτικές εργασίες είναι μια συζήτηση του παρελθόντος. Ωστόσο, το πρόβλημα με αυτήν την παραδοσιακή προσέγγιση της τεχνολογίας αισθητήρων ήταν ότι δεν έδινε ζωντανά δεδομένα. Αυτοί οι αισθητήρες χρησιμοποιήθηκαν για την αποθήκευση των δεδομένων στη συνδεδεμένη μνήμη και χρησιμοποιήθηκαν αργότερα.

Όταν μιλάμε για γεωργία που βασίζεται στο IoT, εξετάζουμε ένα σύστημα που έχει κατασκευαστεί για την παρακολούθηση του αγρού με τη βοήθεια αισθητήρων. Αυτοί οι αισθητήρες παρακολουθούν κάθε απαραίτητο στοιχείο για την παραγωγή των καλλιεργειών όπως την υγρασία του εδάφους, το φως, τη θερμοκρασία κλπ., και για παράδειγμα αυτοματοποιούν το σύστημα άρδευσης. Αυτό το σύστημα επιτρέπει στους αγρότες να παρακολουθούν τις συνθήκες του αγρού από οπουδήποτε και οποιαδήποτε χρονική στιγμή. Η γεωργία που βασίζεται στο IoT είναι πολύ αποτελεσματική συγκριτικά με τη συμβατική γεωργία.

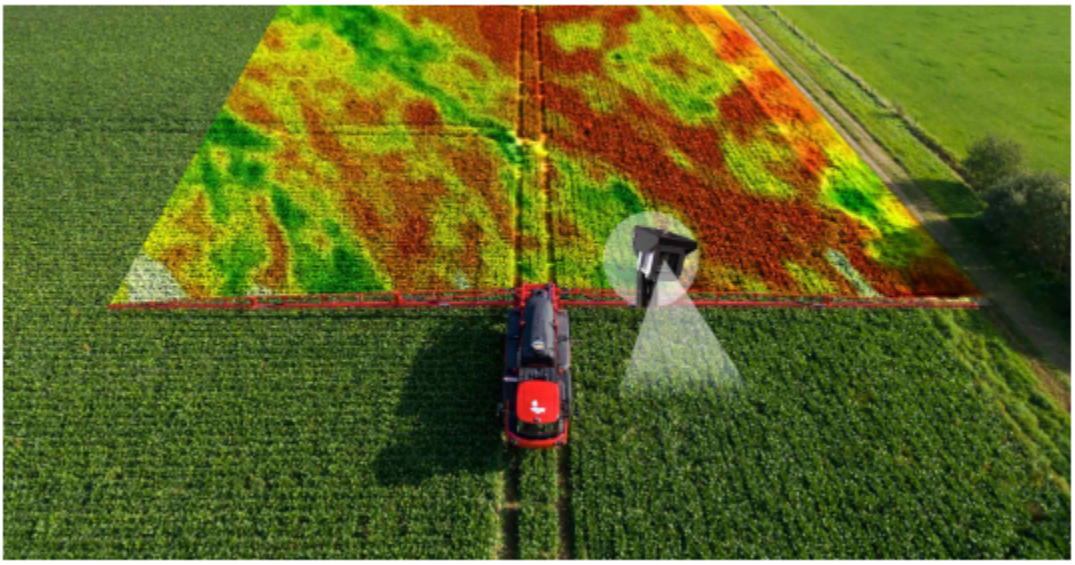
Με την εισαγωγή του βιομηχανικού IoT στη Γεωργία, οι σύγχρονοι αισθητήρες είναι πλέον διαθέσιμοι για χρήση. Αυτοί οι αισθητήρες συνδέονται με το νέφος μέσω ενός κυψελοειδούς/δορυφορικού δικτύου. Αυτό το σύστημα μας βοηθά να λαμβάνουμε δεδομένα σε πραγματικό χρόνο και σε πραγματικό χρόνο και να λαμβάνουμε αποτελεσματικές αποφάσεις. Το IoT έχει προσφέρει διπλά οφέλη στους αγρότες. Μπορούν πλέον να εκτελούν τον ίδιο αριθμό εργασιών σε μικρότερο χρονικό διάστημα και επίσης να αυξάνουν τις αποδόσεις των καλλιεργειών με τη βοήθεια ακριβών δεδομένων που λαμβάνονται από το IoT.

Η έξυπνη γεωργία που βασίζεται στο IoT όχι μόνο βοηθά στον εκσυγχρονισμό των συμβατικών μεθόδων γεωργίας, αλλά στοχεύει και άλλες μεθόδους γεωργίας όπως η βιολογική γεωργία, η οικογενειακή γεωργία (σύνθετοι ή μικροί χώροι, συγκεκριμένα βοοειδή και/ή καλλιέργειες, διατήρηση ιδιαίτερων ή υψηλής ποιότητας ποικιλιών κ. .), και ενισχύει τη γεωργία με υψηλή διαφάνεια.

Η έξυπνη γεωργία που βασίζεται στο IoT είναι επίσης επωφελής όσον αφορά τα περιβαλλοντικά ζητήματα. Για παράδειγμα μπορεί να βοηθήσει τους αγρότες να χρησιμοποιούν αποτελεσματικά το νερό, να βελτιστοποιήσουν τις εισροές και τις επεξεργασίες.

Τώρα, έχοντας κατανοήσει την έννοια της έξυπνης γεωργίας, θα εξετάσουμε τις κύριες εφαρμογές της έξυπνης γεωργίας που βασίζεται στο IoT που φέρνουν επανάσταση στον τομέα της γεωργίας.

2.3.1.1 Καλλιέργεια ακριβείας (Precision farming)



Η γεωργία ακριβείας είναι μια προσέγγιση διαχείρισης που εστιάζει στην παρατήρηση (σχεδόν σε πραγματικό χρόνο), τη μέτρηση και τις απαντήσεις στη μεταβλητότητα στις καλλιέργειες, τα χωράφια και τα ζώα. Μπορεί να βοηθήσει στην αύξηση των αποδόσεων των καλλιεργειών και της απόδοσης των ζώων, στη μείωση του κόστους, συμπεριλαμβανομένου του κόστους εργασίας, και στη βελτιστοποίηση των εισροών της διαδικασίας. Όλα αυτά μπορούν να βοηθήσουν στην αύξηση της κερδοφορίας. Ταυτόχρονα, η γεωργία ακριβείας μπορεί να αυξήσει την ασφάλεια στην εργασία και να μειώσει τις περιβαλλοντικές επιπτώσεις της γεωργίας και των γεωργικών πρακτικών, συμβάλλοντας έτσι στη βιωσιμότητα της γεωργικής παραγωγής.

Το βασικό συστατικό αυτής της τεχνικής καλλιέργειας είναι η χρήση της Πληροφορικής και διαφόρων άλλων τεχνολογιών όπως αισθητήρες, ρομποτική, οχήματα αυτοματισμού, συστήματα ελέγχου, αυτοματοποιημένο υλικό, τεχνολογία μεταβλητού ρυθμού κ.λπ. Το βασικό χαρακτηριστικό της γεωργίας ακριβείας είναι η υιοθέτηση της πρόσβασης σε internet υψηλής ταχύτητας, φορητές συσκευές και αξιόπιστους, χαμηλού κόστους δορυφόρους (για εικόνες και εντοπισμό θέσης) από τους κατασκευαστές.

Η γεωργία ακριβείας θεωρείται μια από τις πιο διάσημες εφαρμογές του IoT στον αγροτικό τομέα και αξιοποιείται παγκοσμίως από αρκετούς οργανισμούς. Δύο από τα παραδείγματα είναι η *CropMetrics* και η *Libelium*. Πρόκειται για οργανισμούς γεωργίας ακριβείας που εστιάζουν σε υπερσύγχρονες αγρονομικές λύσεις.

2.3.1.2 Γεωργικά Drones (Agricultural Drones)

Τα γεωργικά drone είναι μη επανδρωμένα εναέρια οχήματα (UAV) που χρησιμοποιούνται για να βοηθήσουν στη βελτιστοποίηση των γεωργικών λειτουργιών, στην αύξηση της παραγωγής καλλιεργειών και στην παρακολούθηση της ανάπτυξης των καλλιεργειών. Οι αισθητήρες και οι δυνατότητες ψηφιακής απεικόνισης μπορούν να δώσουν στους αγρότες μια πιο πλούσια εικόνα των αγρών τους. Η χρήση ενός γεωργικού drone και η συλλογή πληροφοριών από αυτό μπορεί να αποδειχθεί χρήσιμη για τη βελτίωση των αποδόσεων των καλλιεργειών και της αποδοτικότητας

των αγροκτημάτων. Οι δύο τύποι drone - επίγειων και εναέριων - χρησιμοποιούνται στη γεωργία για:

- Αξιολόγηση της υγείας των καλλιεργειών
- Παρακολούθηση των καλλιεργειών
- Ψεκασμός φυτοφαρμάκων
- Άρδευση
- Φύτευση
- Ανάλυση του χωραφιού

Αυτά τα drones καταγράφουν πολυφασματικές, θερμικές και οπτικές εικόνες κατά τη διάρκεια της πτήσης τους.

Η χρήση drones προσφέρει πολλά οφέλη, όπως απεικόνιση της υγείας των καλλιεργειών, ενσωματωμένη χαρτογράφηση GIS, εξοικονόμηση χρόνου, ευκολία χρήσης και επίσης αύξηση της απόδοσης των καλλιεργειών. Όταν συνδυάζουμε την τεχνολογία των drone με τη σωστή στρατηγική και σχεδιασμό που βασίζεται στη συλλογή δεδομένων σε πραγματικό χρόνο, μπορούμε να δώσουμε μια ανανέωση υψηλής τεχνολογίας στον αγροτικό τομέα.

Από τα δεδομένα που συλλέγονται από drones, οι αγρότες μπορούν να αντλήσουν πληροφορίες σχετικά με τους δείκτες φυτοϋγειονομικού ελέγχου, την καταμέτρηση και την πρόβλεψη της απόδοσης των φυτών, τη μέτρηση του ύψους των φυτών, τη χαρτογράφηση της κάλυψης του θόλου, τη χαρτογράφηση λιμνάζουσας ποσότητας νερού, τις αναφορές ανίχνευσης, τη μέτρηση αποθεμάτων, τη μέτρηση χλωροφύλλης, την περιεκτικότητα σε άζωτο στο σιτάρι , χαρτογράφηση αποστράγγισης, χαρτογράφηση πίεσης ζιζανίων και ούτω καθεξής.

2.3.1.3 Παρακολούθηση ζώων (Livestock monitoring)

Οι ιδιοκτήτες μεγάλων αγροκτημάτων χρησιμοποιούν ασύρματες εφαρμογές IoT για να παρακολουθούν την τοποθεσία, την υγεία και την ευημερία των βοοειδών τους. Αυτές οι πληροφορίες τους βοηθούν να αναγνωρίζουν άρρωστα ζώα και στο εξής να τα διαχωρίζουν από το κοπάδι, να τα φροντίζουν και επίσης να περιορίζουν την εξάπλωση της νόσου μεταξύ άλλων ζώων. Είναι επίσης χρήσιμο για τη μείωση του κόστους εργασίας, καθώς οι ιδιοκτήτες μπορούν να εντοπίσουν τα βοοειδή τους με τη βοήθεια αισθητήρων που βασίζονται στο IoT.

Οι JMB North America και Allflex για παράδειγμα είναι σύλλογοι που προσφέρουν απαντήσεις ελέγχου αγελάδων για παραγωγούς αγελάδων. Μία από τις ρυθμίσεις βοηθά τους ιδιοκτήτες αγελάδων να παρατηρήσουν αγελάδες που είναι έγκυες και πρόκειται να συλλάβουν απογόνους. Από τη γάμπα, ένας αισθητήρας που τροφοδοτείται από μια μπαταρία αφαιρείται όταν σπάσει το νερό του. Αυτό στέλνει δεδομένα στον ιδιοκτήτη ή στον αγρότη. Στο χρόνο που αφιερώνεται με τα βοοειδή που γεννούν, οι αισθητήρες επιτρέπουν στους αγρότες να είναι πιο συγκεντρωμένοι.

2.3.1.4 Έξυπνα θερμοκήπια (Smart greenhouses)

Η καλλιέργεια του θερμοκηπίου ασχολείται με την αύξηση των αποδόσεων λαχανικών, καλλιεργειών, φρούτων κλπ. Τα θερμοκήπια ελέγχουν τους περιβαλλοντικούς παράγοντες μέσω της χειρωνακτικής παρέμβασης ή ενός αναλογικού μηχανισμού ελέγχου. Ωστόσο, η χειρωνακτική παρέμβαση οδηγεί σε απώλεια παραγωγής, απώλεια ενέργειας και κόστος εργασίας. Αυτό καθιστά αναποτελεσματική όλη την έννοια των θερμοκηπίων. Έτσι, τα έξυπνα θερμοκήπια είναι μια καλύτερη εναλλακτική λύση. Ένα έξυπνο θερμοκήπιο μπορεί να δημιουργηθεί με τη βοήθεια του IoT. Αυτά τα έξυπνα θερμοκήπια παρακολουθούν και ελέγχουν έξυπνα το κλίμα χωρίς να απαιτούν χειροκίνητη παρέμβαση.

Σε ένα έξυπνο θερμοκήπιο χρησιμοποιούνται διαφορετικοί τύποι αισθητήρων που μετρούν τους περιβαλλοντικούς παράγοντες και αξιολογούν την καταλληλότητά τους για τα φυτά. Μια απομακρυσμένη πρόσβαση δημιουργείται συνδέοντας το σύστημα σε ένα σύννεφο με τη βοήθεια του IoT. Αυτό εξαλείφει την ανάγκη για συνεχή χειροκίνητη παρακολούθηση. Ο διακομιστής cloud ελέγχει την επεξεργασία δεδομένων και εφαρμόζει μια ενέργεια ελέγχου μέσα στο θερμοκήπιο.

Οι αισθητήρες IoT που είναι εγκατεστημένοι μέσα στο θερμοκήπιο παρέχουν κρίσιμες πληροφορίες για τη θερμοκρασία, την υγρασία, την πίεση και τα επίπεδα φωτός. Αυτοί οι αισθητήρες ελέγχουν τα πάντα, από το άναμμα των φώτων και το άνοιγμα ενός παραθύρου έως τον έλεγχο της θερμοκρασίας και την ψύξη, όλα μέσω ενός σήματος WiFi. Τα πλεονεκτήματα εγκαθίδρυσης τους είναι τα παρακάτω:

- Διατήρηση ιδανικών συνθηκών μικροκλίματος: Οι αισθητήρες IoT επιτρέπουν στους αγρότες να συλλέγουν διάφορα σημεία δεδομένων με πρωτοφανή ευαισθησία. Παρέχουν πληροφορίες σε πραγματικό χρόνο για κρίσιμους κλιματικούς παράγοντες, όπως η θερμοκρασία, η υγρασία, η έκθεση στο φως και το διοξείδιο του άνθρακα σε όλο το θερμοκήπιο. Αυτά τα δεδομένα προτρέπουν τις σχετικές προσαρμογές στις ρυθμίσεις HVAC και φωτισμού για τη διατήρηση των βέλτιστων συνθηκών για την ανάπτυξη των φυτών με παράλληλη αύξηση της ενεργειακής απόδοσης. Παράλληλα, οι αισθητήρες κίνησης/επιτάχυνσης βοηθούν στον εντοπισμό θυρών που μένουν ακούσια ανοιχτές για να διασφαλιστεί ένα αυστηρά ελεγχόμενο περιβάλλον.

- Ενίσχυση των πρακτικών άρδευσης και λίπανσης: Εκτός από τις παραμέτρους περιβάλλοντος, τα έξυπνα θερμοκήπια επιτρέπουν στους αγρότες να τηρούν τις συνθήκες καλλιέργειάς τους. Αυτό διασφαλίζει ότι οι δραστηριότητες άρδευσης και λίπανσης είναι στο ίδιο επίπεδο με τις πραγματικές ανάγκες των καλλιεργούμενων φυτών για μεγιστοποιημένες αποδόσεις. Για παράδειγμα, οι μετρήσεις σχετικά με την ογκομετρική περιεκτικότητα σε νερό του εδάφους υποδεικνύουν εάν οι καλλιέργειες βρίσκονται υπό υδατική καταπόνηση. Ομοίως, οι μετρήσεις της αλατότητας του εδάφους δίνουν χρήσιμες πληροφορίες για τις απαιτήσεις λίπανσης. Με βάση αυτά τα δεδομένα, τα συστήματα καταιονισμού και ψεκασμού μπορούν να ενεργοποιηθούν αυτόματα για την αντιμετώπιση των απαιτήσεων των καλλιεργειών σε πραγματικό χρόνο, ελαχιστοποιώντας παράλληλα τη χειροκίνητη παρέμβαση

- Έλεγχος μολύνσεων και αποφυγή ξεσπάσματος ασθενειών: Η μόλυνση των καλλιεργειών είναι μια επίμονη γεωργική πρόκληση, με κάθε ξέσπασμα να έχει μεγάλο αντίκτυπο στα περιθώρια των καλλιεργειών. Αγροχημικές επεξεργασίες είναι διαθέσιμες, αλλά οι αγρότες συχνά δεν γνωρίζουν την καλύτερη στιγμή για να τις εφαρμόσουν. Οι εφαρμογές που γίνονται πολύ συχνά εγείρουν οικολογικές ανησυχίες, ανησυχίες για την ασφάλεια και το κόστος, ενώ η αποτυχία χρήσης θεραπειών θα μπορούσε να οδηγήσει σε επιζήμιες εστίες ασθενειών. Με τη βοήθεια μιας πλατφόρμας μηχανικής μάθησης, δεδομένα για περιβάλλοντα θερμοκηπίου, εξωτερικές καιρικές συνθήκες και χαρακτηριστικά του εδάφους αποκαλύπτουν πολύτιμες γνώσεις σχετικά με τους υπάρχοντες κινδύνους παρασίτων και μυκήτων. Αξιοποιώντας αυτές τις πληροφορίες, οι αγρότες μπορούν να εφαρμόσουν θεραπείες ακριβώς όταν χρειάζεται για να εξασφαλίσουν μια υγιή καλλιέργεια με τη χαμηλότερη χημική δαπάνη.

- Αποτροπή κλοπών και βελτιωμένη προστασία: Τα θερμοκήπια με καλλιέργειες υψηλής αξίας αποτελούν ευάλωτο στόχο για τους κλέφτες. Καθώς τα παραδοσιακά δίκτυα επιτήρησης με κλειστά κυκλώματα τηλεόρασης είναι ακριβά στην εφαρμογή τους, πολλοί καλλιεργητές δεν διαθέτουν αποτελεσματικό σύστημα ασφαλείας. Σε αυτό το πλαίσιο, οι αισθητήρες IoT σε έξυπνα θερμοκήπια παρέχουν μια προσιτή υποδομή για την παρακολούθηση της κατάστασης της πόρτας και τον εντοπισμό ύποπτων

δραστηριοτήτων. Συνδεδεμένα με ένα αυτοματοποιημένο σύστημα συναγερμού, ειδοποιούν αμέσως τους καλλιεργητές όταν προκύψει ένα ζήτημα ασφάλειας.

2.3.1.5 Παρακολούθηση των κλιματικών συνθηκών (Climate conditions monitoring)

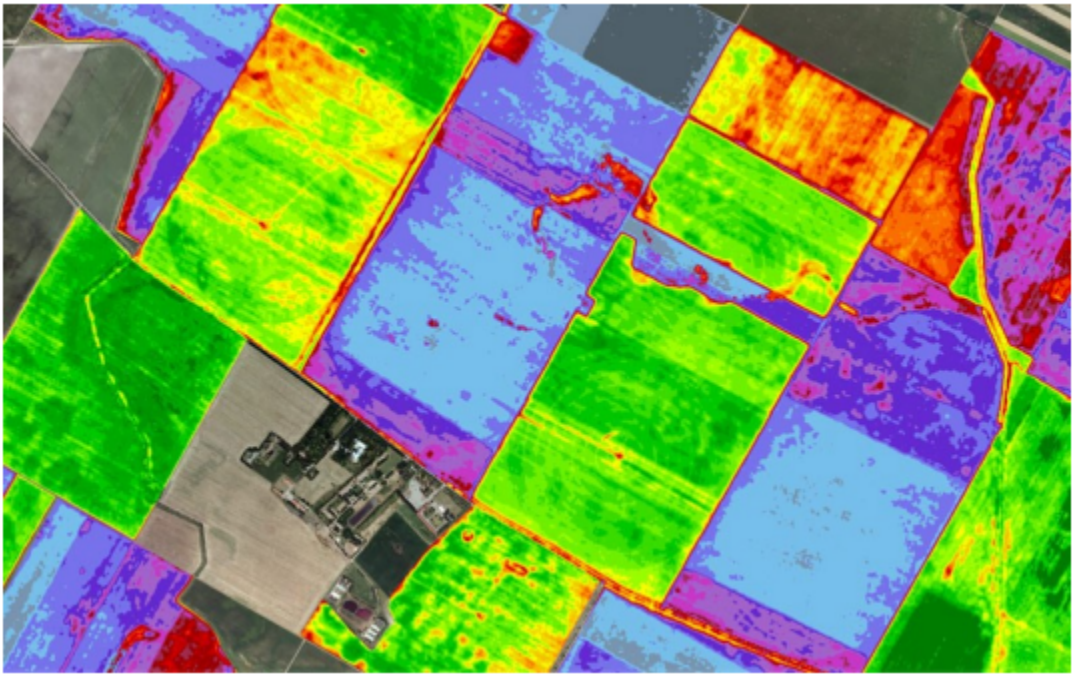
Το κλίμα διαδραματίζει σημαντικό ρόλο στη φυτική παραγωγή. Διαφορετικές καλλιέργειες απαιτούν διαφορετικές κλιματικές συνθήκες για να αναπτυχθούν και οποιαδήποτε μικρή γνώση σχετικά με το κλίμα βελτιώνει σημαντικά την ποσότητα και την ποιότητα της φυτικής παραγωγής. Οι λύσεις IoT επιτρέπουν στους αγρότες να γνωρίζουν τις καιρικές συνθήκες σε πραγματικό χρόνο.

Οι αισθητήρες που τοποθετούνται στα γεωργικά χωράφια συλλέγουν δεδομένα από το περιβάλλον που χρησιμοποιούνται από τους αγρότες για να επιλέξουν μια καλλιέργεια που μπορεί να αναπτυχθεί σε συγκεκριμένες κλιματικές συνθήκες.

Ένα σύστημα παρακολούθησης της κλιματικής αλλαγής ενσωματώνει δορυφορικές παρατηρήσεις, επίγεια δεδομένα και μοντέλα προβλέψεων για την παρακολούθηση και την πρόβλεψη αλλαγών στον καιρό και το κλίμα. Με την πάροδο του χρόνου δημιουργείται ένα ιστορικό αρχείο σημείων μετρήσεων, το οποίο παρέχει τα δεδομένα που επιτρέπουν τη στατιστική ανάλυση και τον 8 προσδιορισμό των μέσων τιμών, τάσεων και διακυμάνσεων. Όσο καλύτερη είναι η διαθέσιμη πληροφορία, τόσο περισσότερο μπορεί να γίνει κατανοητό το κλίμα και τόσο ακριβέστερα μπορούν να αξιολογηθούν οι μελλοντικές συνθήκες, σε τοπικό, περιφερειακό, εθνικό και παγκόσμιο επίπεδο. Αυτό έχει καταστεί ιδιαίτερα σημαντικό στο πλαίσιο της κλιματικής αλλαγής, καθώς η μεταβλητότητα του κλίματος αυξάνεται και τα ιστορικά πρότυπα αλλάζουν.

Ολόκληρο το οικοσύστημα IoT αποτελείται από αισθητήρες που ανιχνεύουν σε πραγματικό χρόνο καιρικές συνθήκες όπως υγρασία, βροχοπτώσεις, θερμοκρασία, όλα πολύ σημαντικά για την παραγωγή καλλιεργειών. Αυτοί οι αισθητήρες είναι σε θέση να προβλέψουν οποιαδήποτε δραστική αλλαγή στις κλιματικές συνθήκες που μπορεί να επηρεάσει την παραγωγή. Αποστέλλεται μια ειδοποίηση στον διακομιστή σχετικά με την αλλαγή του κλίματος που βοηθά στην εξάλειψη της ανάγκης για φυσική παρουσία. Αυτό οδηγεί τελικά σε υψηλότερες αποδόσεις.

2.3.1.6 Η τηλεπισκόπηση (Remote sensing)



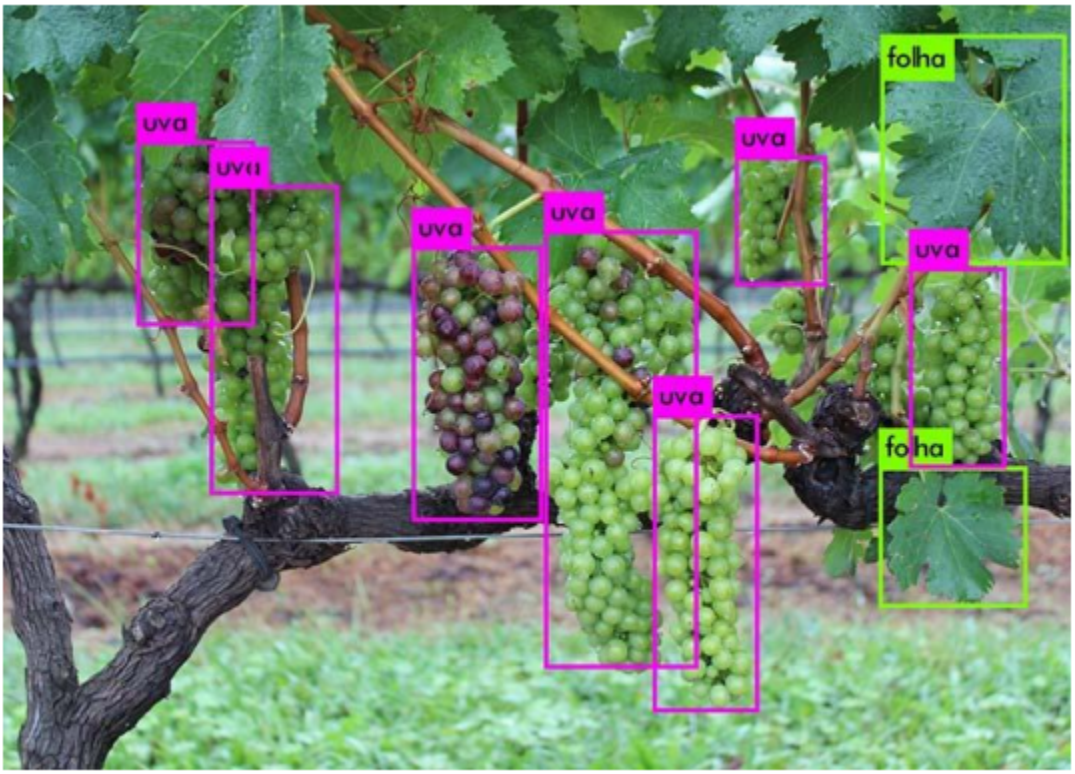
Με βάση το IoT χρησιμοποιεί αισθητήρες που τοποθετούνται κατά μήκος των αγροκτημάτων, όπως μετεωρολογικούς σταθμούς για τη συσσώρευση δεδομένων που μεταφέρονται σε αναλυτικά εργαλεία για ανάλυση. Οι καλλιέργειες μπορούν να παρακολουθούνται από τους αγρότες μέσω αναλυτικών πινάκων εργαλείων και μπορούν να ληφθούν μέτρα από τις πληροφορίες που προκύπτουν ανάλογα.

- Αξιολόγηση καλλιεργειών: Αυτοί οι αισθητήρες τοποθετημένοι σε διαφορετικές γωνίες των αγροκτημάτων αξιολογούν τις καλλιέργειες για να παρακολουθούν τυχόν αλλαγές στο σχήμα, το μέγεθος, το φως, την υγρασία και τη θερμοκρασία. Οποιαδήποτε απόκλιση σημειώνεται από τους αισθητήρες αξιολογείται και ενημερώνεται ο αγρότης. Ως αποτέλεσμα, η τηλεπισκόπηση βοηθά στην πρόληψη της εξάπλωσης ασθενειών καθώς και στην παρακολούθηση της εξέλιξης των καλλιεργειών.

- Καιρικές συνθήκες: Τα δεδομένα που συγκεντρώνονται από τους αισθητήρες στην περίπτωση θερμοκρασίας, υγρασίας, βροχόπτωσης υγρασίας και ανίχνευσης δρόσου βοηθούν στο συμπέρασμα του καιρού στις φάρμες, ώστε η καλλιέργεια να εκτελείται για τις κατάλληλες καλλιέργειες.

- Ποιότητα εδάφους: Η ανάλυση της ποιότητας του εδάφους βοηθά στον καθορισμό της θρεπτικής αξίας και των ξεραμένων τμημάτων των αγροκτημάτων, της ικανότητας αποστράγγισης του εδάφους ή της οξύτητας, που επιτρέπει την προσαρμογή του επιπέδου του νερού που απαιτείται για την άρδευση και την επιλογή ενός πλεονεκτικού τύπου καλλιέργειας.

2.3.1.7 Απεικόνιση / όραση υπολογιστή (Computer imaging / vision)



Αυτή η μορφή απεικόνισης περιλαμβάνει κυρίως τη χρήση των καμερών αισθητήρων που τοποθετούνται σε διάφορες γωνίες του αγροκτήματος για τη δημιουργία εικόνων που περνούν από ψηφιακή επεξεργασία εικόνας.

- Ποιοτικός έλεγχος: Η επεξεργασία εικόνας σε συνδυασμό με τη μηχανική εκμάθηση χρησιμοποιεί εικόνες από τη βάση δεδομένων για σύγκριση με εικόνες καλλιεργειών για

συμπέρασμα για το μέγεθος, το σχήμα, το χρώμα και την ανάπτυξη, με αποτέλεσμα την προσαρμογή της ποιότητας.

- Ταξινόμηση και ταξινόμηση: Η απεικόνιση με υπολογιστή μπορεί να βοηθήσει στην ταξινόμηση και ταξινόμηση των προϊόντων με βάση το χρώμα, το σχήμα και το μέγεθος.

- Παρακολούθηση Άρδευσης: Η άρδευση για μια χρονική περίοδο βοηθά στη χαρτογράφηση των αρδευόμενων εκτάσεων. Αυτό βοηθά στη λήψη απόφασης στην προ συγκομιδής περίοδο συγκομιδής ή μη συγκομιδής.

Κεφάλαιο 3: Βασικές αρχές της εικονικοποίησης

3.1 Εισαγωγή στην εικονικοποίηση

Η τεχνολογία της εικονικοποίησης διαδραματίζει καθοριστικό ρόλο στη σύγχρονη πληροφορική. Πρόκειται για τη διαδικασία δημιουργίας μιας εικονικής έκδοσης κάποιου στοιχείου, όπως για παράδειγμα ένα πραγματικό υλικό (hardware), ένα λειτουργικό λογισμικό (software) κ.α. Επιτρέπει την ταυτόχρονη εκτέλεση πολλαπλών εικονικών συστημάτων, πλατφορμών, εφαρμογών ή πόρων σε έναν μόνο φυσικό υπολογιστή, αυξάνοντας την αποτελεσματικότητα και την ευελιξία του υλικού που χρησιμοποιείται.

Η έννοια της εικονικοποίησης πιστεύεται γενικά ότι έχει τις ρίζες της στις ημέρες των κεντρικών υπολογιστών (mainframes) στα τέλη της δεκαετίας του 1960 και στις αρχές της δεκαετίας του 1970, όταν η IBM επένδυσε πολύ χρόνο και προσπάθεια στην ανάπτυξη ισχυρών λύσεων διαμοιρασμού του χρόνου. Ο διαμοιρασμός του χρόνου αναφέρεται στην κοινή χρήση των πόρων ενός υπολογιστικού συστήματος από μια ομάδα χρηστών, με στόχο την αύξηση της αποδοτικότητας τόσο των χρηστών όσο και των πόρων του υπολογιστή που αυτοί μοιράζονται. Το μοντέλο αυτής της τεχνολογίας κατάφερε να μειώσει το κόστος παροχής υπολογιστικών δυνατοτήτων και έτσι κατέστη δυνατό για οργανισμούς, ακόμη και για μεμονωμένα άτομα, να χρησιμοποιούν έναν υπολογιστή χωρίς να τον κατέχουν στην πραγματικότητα. Παρόμοιοι λόγοι έχουν οδηγήσει και σήμερα την εικονικοποίηση να είναι ένα από τα στάνταρ πρότυπα της βιομηχανίας. Η χωρητικότητα σε έναν μόνο διακομιστή είναι τόσο μεγάλη που είναι σχεδόν αδύνατο για τα περισσότερα φορτία εργασίας να τη χρησιμοποιήσουν αποτελεσματικά. Έτσι, ο καλύτερος τρόπος για να βελτιωθεί η αξιοποίηση των πόρων και ταυτόχρονα να απλοποιηθεί η διαχείριση του κέντρου δεδομένων, είναι μέσω της εικονικοποίησης.

Τα κέντρα δεδομένων (datacenters) χρησιμοποιούν σήμερα τεχνικές εικονικοποίησης για να κάνουν αφαίρεση (abstraction) του φυσικού υλικού, να δημιουργήσουν μεγάλες συγκεντρωτικές δεξαμενές λογικών πόρων, και να προσφέρουν αυτούς τους πόρους στους χρήστες ή τους πελάτες με τη μορφή ευέλικτων, κλιμακούμενων, ενοποιημένων εικονικών μηχανών. Παρόλο που η τεχνολογία και οι περιπτώσεις χρήσης έχουν εξελιχθεί, το βασικό νόημα της εικονικοποίησης παραμένει το ίδιο: να επιτρέπει σε ένα υπολογιστικό περιβάλλον να εκτελεί ταυτόχρονα πολλά ανεξάρτητα συστήματα.

Βασικές έννοιες στην εικονικοποίηση:

- *Hypervisor:* Ο hypervisor είναι ένα λογισμικό, υλικολογισμικό ή υλικό, που δημιουργεί και εκτελεί εικονικές μηχανές (VMs). Διαχειρίζεται την εκτέλεση των φιλοξενούμενων λειτουργικών συστημάτων και κατανέμει πόρους σε αυτά. Οι hypervisors μπορούν να ταξινομηθούν σε δύο τύπους:
    - Τύπος 1 (Native ή Bare-Metal Hypervisors): Αυτοί εκτελούνται απευθείας στο υλικό του κεντρικού υπολογιστή για τον έλεγχο του υλικού και τη διαχείριση των φιλοξενούμενων λειτουργικών συστημάτων. Παραδείγματα περιλαμβάνουν το VMware ESXi, το Microsoft Hyper-V και το Xen.
    - Τύπος 2 (Hosted Hypervisors): Αυτοί εκτελούνται σε ένα συμβατικό λειτουργικό σύστημα όπως ακριβώς και άλλα προγράμματα υπολογιστών. Παραδείγματα περιλαμβάνουν το VMware Workstation και το Oracle VirtualBox.

- *Εικονικές μηχανές (VM)*: Ένα VM είναι ένα στενά απομονωμένο εμπορευματοκιβώτιο λογισμικού που μπορεί να εκτελεί τα δικά του λειτουργικά συστήματα και εφαρμογές σαν να ήταν ένας φυσικός υπολογιστής. Ένα VM συμπεριφέρεται πανομοιότυπα με έναν φυσικό υπολογιστή και περιέχει την εικονική του CPU, RAM, σκληρό δίσκο και διασύνδεση δικτύου.

- *Εικονικοποίηση έναντι εξομοίωσης:* Αν και συχνά χρησιμοποιούνται εναλλακτικά, η εικονικοποίηση και η εξομοίωση είναι διαφορετικές έννοιες. Η εξομοίωση περιλαμβάνει την αναπαραγωγή του πλήρους περιβάλλοντος υλικού ενός μηχανήματος σε ένα άλλο, επιτρέποντας στο λογισμικό που έχει γραφτεί για ένα μηχάνημα να εκτελείται σε ένα άλλο. Η εξομοίωση, από την άλλη πλευρά, περιλαμβάνει τη δημιουργία μιας εικονικής έκδοσης μιας συσκευής ή ενός πόρου, όπως ένας διακομιστής, μια συσκευή αποθήκευσης, ένα δίκτυο ή ακόμη και ένα ολόκληρο δίκτυο συσκευών.

3.2 Τύποι εικονικοποίησης

Η τεχνολογία της εικονικοποίησης κυκλοφορεί σε διάφορους τύπους, οπου ο καθένας έχει σχεδιαστεί για την βελτιστοποίηση των διαφόρων πτυχών στις υποδομές που υπάρχουν στην επιστήμη της πληροφορικής. Παρακάτω αναφέρονται οι κυριότεροι τύποι εικονικοποίησης που χρησιμοποιούνται σήμερα.

3.2.1 Εικονικοποίηση υλικού

Η εικονικοποίηση υλικού επιτρέπει σε ένα μόνο φυσικό μηχάνημα να λειτουργεί ως πολλαπλά μηχανήματα δημιουργώντας προσομοιωμένα περιβάλλοντα. Ο φυσικός κεντρικός υπολογιστής χρησιμοποιεί τον hypervisor ο οποίος δημιουργεί και διαχειρίζεται εικονικές μηχανές (VMs). Αυτά τα VM χρησιμοποιούν τους πόρους του φυσικού κεντρικού υπολογιστή, συμπεριλαμβανομένων της CPU, της μνήμης και της αποθήκευσης, οι οποίοι κατανέμονται στους επισκέπτες ανάλογα με τις ανάγκες που έχουν. Επιπλέον αυτά τα VM είναι σε θέση να εκτελούν ανεξάρτητα λειτουργικά συστήματα και εφαρμογές, απομονωμένες μεταξύ τους. Τα οφέλη που προκύπτουν από την εικονικοποίηση υλικού είναι πως αυξάνεται η χρήση του υλικού, μειώνεται το κόστος αφου ελαχιστοποιούνται οι απαιτήσεις φυσικού υλικού και ενισχύονται οι στρατηγικές ανάκαμψης από καταστροφές και κακόβουλο λογισμικό. Επίσης επιτρέπεται η παράλληλη εκτέλεση πολλαπλών λειτουργικών συστημάτων σε μία μόνο φυσική συσκευή.

Στην περίπτωση της έξυπνης γεωργίας η τεχνική της εικονικοποίησης του υλικού μπορεί να χρησιμοποιηθεί για την ενοποίηση πολλών διαφορετικών λειτουργιών που εκτελούν διάφοροι διακομιστές σε έναν μόνο φυσικό διακομιστή. Κάποιες από αυτές τις λειτουργίες μπορούν να είναι η ανάλυση δεδομένων,η μοντελοποίηση των καλλιεργειών και η διαχείριση των logistics. Έτσι μειώνεται το η κατανάλωση ενέργειας στις γεωργικές επιχειρήσεις.

3.2.2 Εικονικοποίηση λογισμικού

Η εικονικοποίηση λογισμικού είναι η δημιουργία ενός εικονικού περιβάλλοντος που μπορεί να εκτελεί εφαρμογές λογισμικού ανεξάρτητα από το υποκείμενο υλικό. Επιτρέπει την εκτέλεση εφαρμογών σε ένα περιορισμένο και ασφαλές εικονικό περιβάλλον που μιμείται μια ξεχωριστή φυσική συσκευή. Μπορεί όχι μόνο να περιλαμβάνει εφαρμογές αλλά και ολόκληρα λειτουργικά συστήματα -*μπορεί και διαφορετικά μεταξύ τους*-. Ένα από τα σημαντικότερα πλεονεκτήματα της εικονικοποίησης του λογισμικού είναι πως παρέχει συμβατότητα για παλαιότερες εφαρμογές και ασφαλέστερα περιβάλλοντα δοκιμών για νέο λογισμικό, χωρίς να διακινδυνεύεται το κύριο

λειτουργικό σύστημα. Επιπλέον επιτρέπει την ύπαρξη καλύτερου ελέγχου και διανομή των πόρων του εκάστοτε λογισμικού.

Ως τύπος εικονικοποίησης μπορεί να βοηθήσει τις γεωργικές επιχειρήσεις στο να εκτελούν παλαιά συστήματα λογισμικού (legacy software) παράλληλα με νεότερες εφαρμογές χωρίς την ανάγκη για δαπανηρές ενημερώσεις υλικού. Μπορεί επίσης να διευκολύνει τη δοκιμή νέων λύσεων λογισμικού για τη διαχείριση καλλιεργειών και την ανάλυση δεδομένων χωρίς να διαταράσσονται οι τρέχουσες λειτουργίες.

3.2.3 Εικονικοποίηση δικτύου

Η εικονικοποίηση δικτύου χρησιμοποιεί λογισμικό για τη δημιουργία μιας «προβολής» του δικτύου, την οποία μπορεί να χρησιμοποιήσει ένας διαχειριστής για τη διαχείριση του δικτύου από μια ενιαία κονσόλα. Αποσπά στοιχεία και λειτουργίες υλικού (π.χ. συνδέσεις, μεταγωγείς και δρομολογητές) και τα ενσωματώνει σε λογισμικό που εκτελείται σε έναν hypervisor. Ο διαχειριστής δικτύου μπορεί να τροποποιεί και να ελέγχει αυτά τα στοιχεία χωρίς να αγγίζει τα υποκείμενα φυσικά στοιχεία, γεγονός που απλοποιεί δραματικά τη διαχείριση του δικτύου. Οι τύποι εικονικοποίησης δικτύου περιλαμβάνουν τη δικτύωση που ορίζεται από το λογισμικό (SDN), η οποία εικονικοποιεί το υλικό που ελέγχει τη δρομολόγηση της κυκλοφορίας του δικτύου, το οποίο ονομάζεται επίπεδο ελέγχου. Ένας άλλος τύπος είναι η εικονικοποίηση λειτουργιών δικτύου, η οποία εικονικοποιεί μία ή περισσότερες συσκευές υλικού που παρέχουν μια συγκεκριμένη λειτουργία δικτύου (για παράδειγμα ένα τείχος προστασίας, έναν εξισορροπιστή φορτίου ή έναν αναλυτή κίνησης), διευκολύνοντας τη διαμόρφωση, την παροχή και τη διαχείριση αυτών των συσκευών.

Όσον αφορά την εφαρμογή της στην έξυπνη γεωργία μπορούμε να κατανοήσουμε την σημαντικότητα της. Η εικονικοποίηση δικτύου μπορεί να διευκολύνει τη διαχείριση πολύπλοκων ροών δεδομένων από συσκευές IoT που βρίσκονται διάσπαρτες σε εκτεταμένους γεωργικούς αγρούς. Επιπλέον είναι σε θέση να επιτρέπει τη δημιουργία τμηματοποιημένων δικτύων για ευαίσθητα δεδομένα, ενισχύοντας έτσι την ασφάλεια και την αποδοτικότητα.

3.2.4 Εικονικοποίηση χώρου αποθήκευσης

Η εικονικοποίηση αποθήκευσης είναι μια τεχνική που συγκεντρώνει φυσικό αποθηκευτικό χώρο από πολλαπλές συσκευές αποθήκευσης δικτύου σε αυτό που φαίνεται να είναι μια ενιαία συσκευή αποθήκευσης, διαχειριζόμενη από μια κεντρική κονσόλα. Επιτρέπει ουσιαστικά την πρόσβαση σε όλες τις συσκευές αποθήκευσης στο δίκτυο -*είτε είναι εγκατεστημένες σε μεμονωμένους διακομιστές είτε σε αυτόνομες μονάδες αποθήκευσης*- και τη διαχείρισή τους ως μία ενιαία συσκευή αποθήκευσης. Πιο συγκεκριμένα, η εικονικοποίηση αποθήκευσης ομαδοποιεί όλα τα μπλοκ αποθήκευσης σε ένα ενιαίο κοινόχρηστο σύνολο, από το οποίο μπορούν να ανατεθούν σε

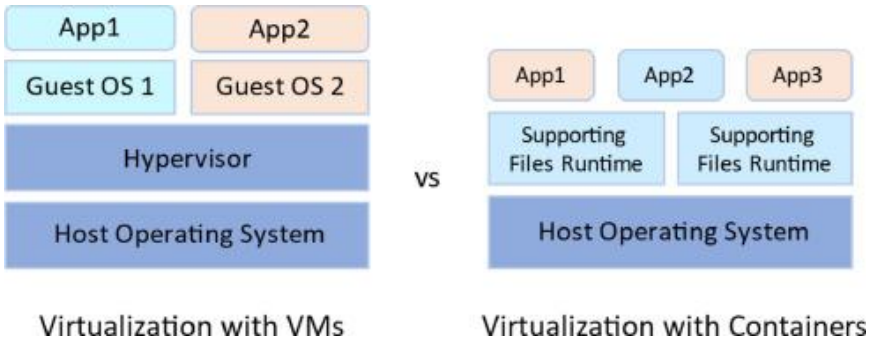
οποιοδήποτε VM στο δίκτυο ανάλογα με τις ανάγκες. Η τεχνική αυτή βελτιώνει τη χρήση των πόρων, ενισχύει την κινητικότητα των δεδομένων και απλοποιεί τις διαδικασίες δημιουργίας αντιγράφων ασφαλείας και ανάκτησης. Μπορεί επίσης να βελτιώσει δραματικά την απόδοση των εφαρμογών με τη δυναμική κατανομή των πόρων αποθήκευσης βάσει της ζήτησης.

Από τη στιγμή που η έξυπνη γεωργία βασίζετε κατα κύριο λόγο στην εισροή τεράστιου όγκου δεδομένων μπορούμε να θεωρήσουμε την εικονικοποίηση χώρου αποθήκευσης απαραίτητη για τη διαχείριση τους. Μπορεί να εξασφαλίσει ότι τα δεδομένα αυτά είναι εύκολα προσβάσιμα και αποτελεσματικά αποθηκευμένα, διευκολύνοντας έτσι την έγκαιρη ανάλυση και λήψη αποφάσεων.

3.2.5 Εικονικοποίηση βασισμένη σε περιέκτες

3.2.5.1 Βασικά στοιχεία

Η εικονικοποίηση με βάση τα περιέκτες έχει αναδειχθεί ως μια δημοφιλής εναλλακτική λύση στην παραδοσιακή εικονικοποίηση με VM τα τελευταία χρόνια. Επιτρέπει την εκτέλεση πολλαπλών απομονωμένων user-space στιγμιοτύπων, ή αλλιώς περιέκτες, σε ένα μόνο λειτουργικό σύστημα κεντρικού υπολογιστή. Σε αντίθεση με τις παραδοσιακές τεχνικές εικονικοποίησης, οι οποίες βασίζονται σε hypervisors και ολοκληρωμένα guest λειτουργικά συστήματα, οι περιέκτες αξιοποιούν τον πυρήνα του host λειτουργικού συστήματος για να παρέχουν απομόνωση πόρων και διαχωρισμό διεργασιών. Ενθυλακώνουν σε ένα πακέτο όλα τα απαραίτητα στοιχεία για την εκτέλεση μιας εφαρμογής, όπως αρχεία, μεταβλητές περιβάλλοντος, dependencies και βιβλιοθήκες το οποίο μπορεί να εκτελεστεί σε οποιαδήποτε υποδομή, και καταργούν την ανάγκη για πλεονάζουσες εγκαταστάσεις στο λειτουργικό σύστημα. Τα περιέκτες είναι ελαφριά, ξεκινούν γρήγορα, χρησιμοποιούν λιγότερους πόρους από τα παραδοσιακά VM και παρέχουν ένα σταθερό περιβάλλον για την ανάπτυξη, τη δοκιμή και την ανάπτυξη εφαρμογών.



Εικ.1: VM & Container Εικονικοποίηση

3.2.5.2 Οφέλη της εικονικοποίησης με περιέκτες

Η εικονικοποίηση με περιέκτες προσφέρει σημαντικά οφέλη στους προγραμματιστές και τις ομάδες ανάπτυξης. Μεταξύ αυτών είναι τα εξής:

- Φορητότητα: Ένα container δημιουργεί ένα εκτελέσιμο πακέτο λογισμικού που είναι αποκομμένο από το λειτουργικό σύστημα του host. Το καθιστά έτσι φορητό και ικανό να εκτελείται ομοιόμορφα και με αξιοπιστία σε οποιαδήποτε πλατφόρμα ή νέφος.

- Ευελιξία: Η open source Docker Engine για την εκτέλεση container ξεκίνησε το βιομηχανικό πρότυπο για περιέκτες με απλά εργαλεία ανάπτυξης και μια καθολική προσέγγιση πακεταρίσματος που λειτουργεί τόσο σε λειτουργικά συστήματα Linux όσο και σε Windows. Το οικοσύστημα των περιέκτες έχει μετατοπιστεί σε μηχανές που διαχειρίζεται η Open Container Initiative (OCI). Οι προγραμματιστές λογισμικού μπορούν να συνεχίσουν με τη χρήση ευέλικτων ή DevOps εργαλείων και διαδικασιών για την ταχεία ανάπτυξη και βελτίωση εφαρμογών.

- Ταχύτητα: Τα περιέκτες μοιράζονται τον πυρήνα του λειτουργικού συστήματος (OS) του μηχανήματος. Αυτό όχι μόνο οδηγεί σε υψηλότερη αποδοτικότητα του διακομιστή, αλλά μειώνει επίσης το κόστος του διακομιστή και των αδειών χρήσης, ενώ επιταχύνει τους χρόνους εκκίνησης, καθώς δεν υπάρχει λειτουργικό σύστημα για εκκίνηση.

- Απομόνωση σφαλμάτων: Η κάθε εφαρμογή σε container είναι απομονωμένη και λειτουργεί ανεξάρτητα από τις υπόλοιπες. Η αποτυχία ενός container δεν επηρεάζει τη συνέχιση της λειτουργίας οποιουδήποτε άλλου container. Οι προγραμματιστές μπορούν να εντοπίζουν και να διορθώνουν τυχόν τεχνικά προβλήματα σε ένα container χωρίς να υπάρχει διακοπή λειτουργίας σε άλλα. Επίσης, η container engine μπορεί να χρησιμοποιήσει οποιεσδήποτε τεχνικές απομόνωσης ασφαλείας του λειτουργικού συστήματος για την απομόνωση σφαλμάτων εντός των περιέκτες.

- Αποδοτικότητα: Όταν το λογισμικό που εκτελείται σε containerized περιβάλλοντα χρησιμοποιεί το ίδιο λειτουργικό σύστημα (OS) με τον υπολογιστή στον οποίο βρίσκεται, μπορεί να μοιράζεται τμήματα του εαυτού του με άλλα περιέκτες. Αυτό καθιστά τα περιέκτες μικρότερα και ταχύτερα στην εκκίνηση σε σύγκριση με τις εικονικές μηχανές (VM). Εξαιτίας αυτού, είναι δυνατόν να χωρέσουν περισσότερα περιέκτες σε έναν υπολογιστή απ' ό,τι VM, πράγμα που σημαίνει ότι μπορούμε να χρησιμοποιήσουμε τους πόρους του διακομιστή μας πιο αποτελεσματικά και να εξοικονομήσουμε χρήματα από το κόστος του διακομιστή και των αδειών χρήσης.

- Ασφάλεια: Η απομόνωση των εφαρμογών ως περιέκτες αποτρέπει εγγενώς την εισβολή κακόβουλου κώδικα από το να επηρεάσει άλλα περιέκτες ή το κεντρικό σύστημα. Επίσης, μπορούν να οριστούν δικαιώματα ασφαλείας για τον αυτόματο αποκλεισμό ανεπιθύμητων στοιχείων από την είσοδο σε περιέκτες ή τον περιορισμό της επικοινωνίας με περιττούς πόρους.
- Ευκολία διαχείρισης: Μια πλατφόρμα ενορχήστρωσης περιεκτών αυτοματοποιεί την εγκατάσταση, την κλιμάκωση και τη διαχείριση των φορτίων εργασίας και των υπηρεσιών που έχουν ενσωματωθεί σε περιέκτες. Οι πλατφόρμες ενορχήστρωσης περιεκτών (Docker swarm, Kubernetes) μπορούν να διευκολύνουν τα καθήκοντα διαχείρισης, όπως η κλιμάκωση εφαρμογών που έχουν ενσωματωθεί σε περιέκτες, η διάθεση νέων εκδόσεων εφαρμογών και η παροχή παρακολούθησης, καταγραφής και αποσφαλμάτωσης, μεταξύ άλλων λειτουργιών

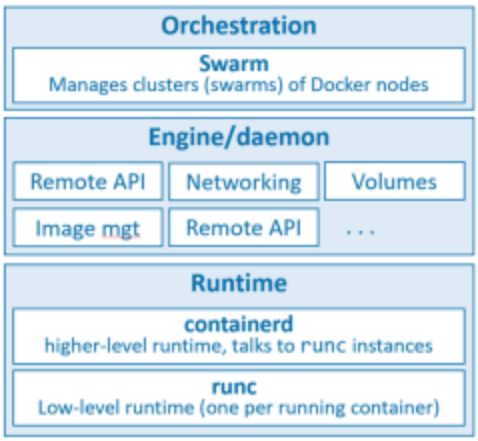
3.2.5.3 Τεχνολογία Docker και ενορχηστρωτές Docker Swarm & Kubernetes

Τεχνολογία Docker

Το Docker είναι μια ισχυρή πλατφόρμα που χρησιμοποιεί την τεχνολογία των περιεκτών έτσι ώστε να επιτρέπει στους προγραμματιστές να συσκευάζουν εφαρμογές σε περιέκτες και να τα διαθέτουν στην αγορά. Όπως έχουμε προαναφέρει ένας περιέκτης είναι ένα τυποποιημένο εκτελέσιμο στοιχείο που συνδυάζει τον πηγαίο κώδικα της εφαρμογής με τις βιβλιοθήκες του λειτουργικού συστήματος (OS) και τα dependencies που απαιτούνται για την εκτέλεση αυτού του κώδικα σε οποιοδήποτε περιβάλλον. Αυτή ακριβώς είναι και η προσέγγιση του Docker στην εικονικοποίηση. Ενθυλακώνει μια εφαρμογή που μπορεί να εκτελεστεί σε οποιαδήποτε μηχανή Docker, ανεξάρτητα από το υποκείμενο λειτουργικό σύστημα ή το υλικό.

Η αρχιτεκτονική της τεχνολογίας Docker έχει τρία κύρια τμήματα:

1. Το χρόνο εκτέλεσης (runtime)
2. Τη μηχανή (daemon, engine)
3. Τον ενορχηστρωτή



Εικ 2.: Αρχιτεκτονική Docker

1. Το runtime λειτουργεί στο χαμηλότερο επίπεδο και είναι υπεύθυνο για την εκκίνηση και τη διακοπή των περιεκτών (αυτό περιλαμβάνει τη δημιουργία όλων των δομών του λειτουργικού συστήματος, όπως τα namespaces και τα cgroups). Το Docker υλοποιεί μια πολυεπίπεδη runtime αρχιτεκτονική με runtimes υψηλού και χαμηλού επιπέδου που συνεργάζονται μεταξύ τους.
    - Το χαμηλού επιπέδου runtime ονομάζεται runc και είναι η υλοποίηση αναφοράς του Open Containers Initiative (OCI) runtime-spec. Η δουλειά του είναι να διασυνδέεται με το υποκείμενο λειτουργικό σύστημα και να εκκινεί και να σταματάει τους περιέκτες. Κάθε τρέχον περιέκτης σε έναν κόμβο Docker έχει μια παρουσία runc που το διαχειρίζεται.
    - Το runtime ανώτερου επιπέδου ονομάζεται containerd. Το containerd κάνει πολύ περισσότερα από το runc. Διαχειρίζεται ολόκληρο τον κύκλο ζωής ενός περιέκτη, συμπεριλαμβανομένου του τραβήγματος εικόνων, της δημιουργίας διασυνδέσεων δικτύου και της διαχείρισης των instances runc χαμηλότερου επιπέδου.
    - Μια τυπική εγκατάσταση Docker έχει μια ενιαία διεργασία containerd (docker-containerd) που ελέγχει τις περιπτώσεις runc (dockerrunc) που σχετίζονται με κάθε τρέχον περιέκτη.
2. Ο δαίμονας Docker (dockerd) βρίσκεται πάνω από το containerd και εκτελεί εργασίες υψηλότερου επιπέδου όπως:
    - έκθεση του απομακρυσμένου API του Docker,
    - διαχείριση εικόνων, τόμων, δικτύων και άλλα

3. Το Docker διαθέτει επίσης εγγενή υποστήριξη για τη διαχείριση συστάδων κόμβων που εκτελούν το Docker. Αυτές οι συστάδες ονομάζονται σμήνη και η εγγενής τεχνολογία ονομάζεται Docker Swarm. Το Docker Swarm είναι εύχρηστο και πολλές εταιρείες το χρησιμοποιούν σε πραγματικές παραγωγικές εφαρμογές.

Ενορχηστρωτής Docker Swarm

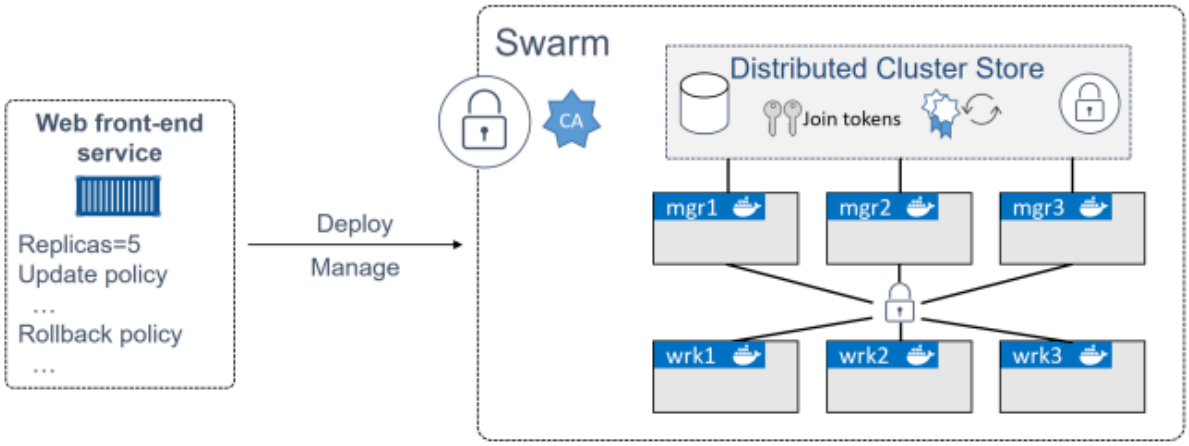
Το Docker Swarm ανήκει στα εργαλεία ενορχήστρωσης περιεκτών και επιτρέπει τη διαχείριση συστάδων μηχανών που χρησιμοποιούν Docker. Δίνει τη δυνατότητα στους προγραμματιστές να αναπτύσσουν, να κλιμακώνουν, και να διαχειρίζονται με ευκολία εφαρμογές περιεκτών σε πολλαπλές μηχανές.

Όσον αφορά τις συστάδες, πρόκειται για έναν ή περισσότερους ομαδοποιημένους κόμβους Docker που το Docker Swarm διαχειρίζεται. Παρέχει ένα κρυπτογραφημένο κατανεμημένο αποθηκευτικό χώρο συστάδας, κρυπτογραφημένα δίκτυα, αμοιβαίο TLS, ασφαλή σύνδεση σε συστάδα tokens και ένα PKI που κάνει τη διαχείριση και την εναλλαγή πιστοποιητικών αρκετά εύκολη. Επιπλέον επιτρέπει την προσθαφαίρεση κόμβων χωρίς τη διακοπή του. Ειδικότερα, Το TLS είναι τόσο στενά ενσωματωμένο που είναι αδύνατο να δημιουργηθεί ένα σμήνος χωρίς αυτό. Το Docker Swarm χρησιμοποιεί το TLS για την κρυπτογράφηση των επικοινωνιών, την αυθεντικοποίηση των κόμβων και την εξουσιοδότηση ρόλων, και όλα αυτά συμβαίνουν τόσο ομαλά που ο χρήστης δεν αντιλαμβάνεται καν ότι υπάρχουν.

Οι κόμβοι διαμορφώνονται ως διαχειριστές (managers) ή εργάτες (workers). Οι διαχειριστές φροντίζουν το επίπεδο ελέγχου της συστάδας, δηλαδή πράγματα όπως η κατάσταση της συστάδας και η αποστολή εργασιών στους εργάτες. Οι εργάτες έπειτα με τη σειρά τους δέχονται εργασίες από τους διαχειριστές και τις εκτελούν. Η διαμόρφωση και η κατάσταση του σμήνους διατηρούνται σε μια κατανεμημένη βάση δεδομένων που ονομάζεται etcd, και βρίσκεται σε όλους τους διαχειριστές. Διατηρείται στη μνήμη, είναι πάντα ενημερωμένη και απαιτεί μηδενική διαμόρφωση καθώς εγκαθίσταται ως μέρος του σμήνους και απλά φροντίζει τον εαυτό της.

Στο Docker Swarm, το θεμελιώδες δομικό στοιχείο για την ενορχήστρωση εφαρμογών ονομάζεται «υπηρεσία». Ο όρος αυτός αντιπροσωπεύει μια νέα έννοια που εισήχθη παράλληλα με το Swarm και χρησιμεύει ως ένα ανώτερο επίπεδο αφαίρεσης που περιλαμβάνει διάφορα προηγμένα χαρακτηριστικά που σχετίζονται με τους περιέκτες.Μια υπηρεσία αποτελεί μια εμπλουτισμένη έκδοση ενός περιέκτη (container). Ενώ ένας περιέκτη είναι μια ελαφριά, απομονωμένη μονάδα που ενθυλακώνει μια εφαρμογή και τις εξαρτήσεις της, μια υπηρεσία επεκτείνει αυτή την έννοια παρέχοντας πρόσθετες δυνατότητες. Ένα βασικό χαρακτηριστικό μιας υπηρεσίας είναι η ικανότητά της να χειρίζεται την κλιμάκωση. Μπορεί να ορίζει πόσα instances, ή αντίγραφα, μιας εφαρμογής σε container θα πρέπει να εκτελούνται ανά πάσα στιγμή. Το Docker Swarm διαχειρίζεται αυτόματα την κατανομή αυτών των αντιγράφων στη συστάδα των μηχανών. Μια άλλη σημαντική

πτυχή μιας υπηρεσίας είναι η υποστήριξή της για κυλιόμενες ενημερώσεις (rolling updates). Όταν χρειάζεται να ενημερωθεί η εφαρμογή που εκτελείται μέσα σε περιέκτες, αυτό μπορεί να συμβεί χωρίς διακοπή λειτουργίας. Το Docker Swarm ενορχηστρώνει τη διαδικασία ενημέρωσης με ελεγχόμενο τρόπο, αντικαθιστώντας σταδιακά τους παλιούς περιέκτες με νέους, διασφαλίζοντας παράλληλα ότι η εφαρμογή παραμένει διαθέσιμη και ανταποκρινόμενη. Επιπλέον, οι υπηρεσίες προσφέρουν απλή λειτουργία επαναφοράς (rollback). Εάν μια νέα έκδοση της εφαρμογής εισάγει προβλήματα ή σφάλματα, μπορεί εύκολα να επανέλθει σε μια προηγούμενη έκδοση της υπηρεσίας. Αυτή η δυνατότητα επιτρέπει τη γρήγορη ανάκαμψη σε περίπτωση απροσδόκητων προβλημάτων κατά τη διάρκεια των ενημερώσεων.



Εικ 3.: Υψηλό επίπεδο ενός σμήνους

Ενορχηστρωτής Kubernetes

Το Kubernetes, συχνά συντομογραφούμενο ως K8s (όπου το «8» αντιπροσωπεύει τα οκτώ γράμματα μεταξύ του «K» και του «s» στο «Kubernetes»), είναι μια πλατφόρμα ενορχήστρωσης περιεκτών ανοικτού κώδικα που αναπτύχθηκε αρχικά από την Google, για τη διαχείριση μικρουπηρεσιών ή εφαρμογών που έχουν κατασκευαστεί με περιέκτες σε μια κατανεμημένη συστάδα κόμβων. Παρέχει μια εξαιρετικά ανθεκτική υποδομή με δυνατότητες ανάπτυξης μηδενικού χρόνου διακοπής λειτουργίας, αυτόματη επαναφορά (automatic rollback), κλιμάκωση (scaling) και αυτοθεραπεία (self-healing) των περιεκτών. Κύριος στόχος του Kubernetes είναι να κρύψει την πολυπλοκότητα της διαχείρισης ενός στόλου περιεκτών παρέχοντας REST APIs για τις απαιτούμενες λειτουργίες. Είναι από τη φύση του φορητό, πράγμα που σημαίνει ότι μπορεί να

τρέξει σε διάφορες δημόσιες ή ιδιωτικές πλατφόρμες cloud, όπως το AWS, το Azure, το OpenStack κ.α.

Ακολουθεί μια αρχιτεκτονική πελάτη-εξυπηρετητή (client-server). Είναι δυνατή η εγκατάσταση πολλαπλών master κόμβων -*για υψηλή διαθεσιμότητα*-, αλλά από προεπιλογή υπάρχει ένας μόνο master server που λειτουργεί ως κόμβος ελέγχου και σημείο επαφής. Ο κεντρικός διακομιστής αποτελείται από διάφορα στοιχεία, όπως έναν kube-apiserver, έναν αποθηκευτικό χώρο etcd, έναν kube-controller-manager, έναν cloud-controller-manager, έναν kube-scheduler και έναν διακομιστή DNS για τις υπηρεσίες Kubernetes. Τα συστατικά του κόμβου περιλαμβάνουν το kubelet και το kube-proxy πάνω από το Docker. Στην αρχιτεκτονική του επίσης υπάρχουν και τα pods όπου είναι τα μικρότερα και απλούστερα αντικείμενα του Kubernetes. Ένα pod ενθυλακώνει έναν ή περισσότερους περιέκτες που μοιράζονται τον ίδιο namespace δικτύου και τους ίδιους τόμους αποθήκευσης, και αποτελεί τη βασική μονάδα ανάπτυξης.

Όπως αναφέρθηκε ο master κόμβος είναι υπεύθυνος για τη διαχείριση της συστάδας Kubernetes. Επιβλέπει τον χρονοπρογραμματισμό των φορτίων εργασίας, τη διατήρηση της κατάστασης της συστάδας και την υλοποίηση αλλαγών. Βασικά στοιχεία ενός master κόμβου είναι:

- etcd : Πρόκειται για έναν απλό, κατανεμημένο αποθηκευτικό χώρο τιμών κλειδιών (key-value), ο οποίος χρησιμοποιείται για την αποθήκευση των δεδομένων του Kubernetes cluster (όπως ο αριθμός των pods, η κατάστασή τους, ο χώρος ονομάτων κ.α.), των αντικειμένων API και των λεπτομερειών ανακάλυψης υπηρεσιών. Είναι προσβάσιμο μόνο από τον διακομιστή API για λόγους ασφαλείας. Το etcd επιτρέπει ειδοποιήσεις στο cluster σχετικά με αλλαγές στις ρυθμίσεις με τη βοήθεια των watchers. Οι ειδοποιήσεις είναι API αιτήματα σε κάθε κόμβο συστάδας etcd για να προκαλέσουν την ενημέρωση των πληροφοριών στον αποθηκευτικό χώρο του κόμβου.

- API Server : Ο διακομιστής API του Kubernetes είναι η κεντρική οντότητα διαχείρισης που λαμβάνει όλα τα αιτήματα REST για τροποποιήσεις (σε pods, υπηρεσίες, σύνολα/ελεγκτές αντιγραφής και άλλα), λειτουργώντας ως frontend για τη συστάδα. Επίσης, αυτό είναι το μόνο στοιχείο που επικοινωνεί με τη συστάδα etcd, διασφαλίζοντας ότι τα δεδομένα αποθηκεύονται σε αυτή και είναι σε συμφωνία με τις λεπτομέρειες των υπηρεσιών των deployed pods.

- Scheduler : Βοηθάει στον προγραμματισμό των pods (μια ομάδα περιεκτών μέσα στην οποία εκτελούνται διεργασίες μιας εφαρμογής) στους διάφορους κόμβους με βάση τη χρήση των πόρων. Διαβάζει τις λειτουργικές απαιτήσεις της υπηρεσίας και την προγραμματίζει στον κόμβο που ταιριάζει καλύτερα. Για παράδειγμα, αν η εφαρμογή χρειάζεται 1GB μνήμης και 2 πυρήνες CPU, τότε τα pods για τη συγκεκριμένη εφαρμογή θα προγραμματιστούν σε έναν κόμβο με τουλάχιστον αυτούς τους πόρους. Ο scheduler εκτελείται κάθε φορά που υπάρχει ανάγκη προγραμματισμού pods. Επιπλέον πρέπει να

γνωρίζει τους συνολικούς διαθέσιμους πόρους καθώς και τους πόρους που διατίθενται σε υφιστάμενους φόρτους εργασίας σε κάθε κόμβο.

- Controller manager : Εκτελεί στο παρασκήνιο έναν αριθμό διαφορετικών controller διεργασιών (για παράδειγμα, ο controller αντιγραφής ελέγχει τον αριθμό των αντιγράφων σε ένα pod, ο controller τελικών σημείων συμπληρώνει αντικείμενα τελικών σημείων, όπως υπηρεσίες και pods, κ.α.) για να ρυθμίζει την κοινή κατάσταση της συστάδας και να εκτελεί συνήθεις εργασίες. Όταν συμβαίνει μια αλλαγή στη διαμόρφωση μιας υπηρεσίας (αντικατάσταση της εικόνας από την οποία εκτελούνται τα pods ή αλλαγή παραμέτρων στο αρχείο διαμόρφωσης yaml), ο controller εντοπίζει την αλλαγή και αρχίζει να εργάζεται προς τη νέα επιθυμητή κατάσταση.

Ένα άλλο στοιχείο του είναι οι κόμβοι εργάτες - worker nodes. Δεν είναι τίποτα παραπάνω από τα μηχανήματα στα οποία εκτελούνται οι εφαρμογές που έχουν ενσωματωθεί σε περιέκτες. Κύρια χαρακτηριστικά των worker κόμβων είναι:

- Kubelet : Είναι η κύρια υπηρεσία που τρέχει σε κάθε worker κόμβο. Λαμβάνει τακτικά νέες ή τροποποιημένες προδιαγραφές pod (κυρίως μέσω του API Server) και διασφαλίζει ότι τα pod και οι περιέκτες τους είναι υγιή και λειτουργούν στην επιθυμητή κατάσταση.
- Kube-proxy: Μια υπηρεσία μεσολάβησης που εκτελείται σε κάθε worker κόμβο για να χειρίζεται την υποδικτύωση μεμονωμένων κεντρικών υπολογιστών και να εκθέτει υπηρεσίες στον εξωτερικό κόσμο. Πραγματοποιεί την προώθηση αιτήσεων στα σωστά pods/περιέκτες μέσω των διαφόρων απομονωμένων δικτύων σε μία συστάδα.

Σύγκριση του Docker Swarm και του Kubernetes

Το Docker Swarm και το Kubernetes είναι αμφότερα πλατφόρμες ενορχήστρωσης περιεκτών- διαχειρίζονται τον κύκλο ζωής των περιεκτών, συμπεριλαμβανομένης της ανάπτυξης, της κλιμάκωσης, της δικτύωσης και της διαθεσιμότητας. Ενώ το Docker Swarm είναι το εγγενές εργαλείο ομαδοποίησης και χρονοπρογραμματισμού του Docker, το Kubernetes είναι μια πλατφόρμα ανοιχτού κώδικα που προέρχεται από την Google. Παρακάτω παρατίθεται ένας συγκριτικός πίνακας που υπογραμμίζει τα πλεονεκτήματα και τα μειονεκτήματά τους:

Χαρακτηριστικό	Docker Swarm	Kubernetes
Ευκολία εγκατάστασης	Το Docker Swarm είναι ευκολότερο στη διαμόρφωση και απλούστερο στην εγκατάσταση σε σύγκριση με το Kubernetes.	Το Kubernetes έχει μια πιο απότομη καμπύλη εκμάθησης και απαιτεί περισσότερες αρχικές ρυθμίσεις, αλλά είναι πιο πλούσιο σε δυνατότητες.
Ενσωμάτωση	Απρόσκοπτη ενσωμάτωση με το οικοσύστημα και τα εργαλεία Docker.	Εκτεταμένες δυνατότητες API και εργαλείων που ενσωματώνονται καλά με διάφορες υπηρεσίες cloud και εργαλεία λογισμικού.
Επεκτασιμότητα	Ταχύτερη ανάπτυξη και κλιμάκωση, η οποία είναι πιο απλή αλλά λιγότερο ευέλικτη από το Kubernetes.	Παρέχει εξαιρετικά προηγμένο και πιο λεπτομερή έλεγχο της επεκτασιμότητας, αλλά με πρόσθετη πολυπλοκότητα.
Εξισορρόπηση φορτίου	Παρέχει βασική εξισορρόπηση φορτίου και απαιτεί χειροκίνητη διαμόρφωση για πιο προηγμένα σενάρια.	Προσφέρει πιο εξελιγμένη εξισορρόπηση φορτίου και διαχειρίζεται αυτόματα την κατανομή της κυκλοφορίας.
Υψηλή διαθεσιμότητα	Το Docker Swarm υποστηρίζει υψηλή διαθεσιμότητα σε επίπεδο περιεκτών, αλλά λιγότερο σε επίπεδο ενορχήστρωσης.	Το Kubernetes προσφέρει υψηλή διαθεσιμότητα για τους περιέκτες και για το ίδιο το επίπεδο διαχείρισης συστάδων.

Χρήση πόρων	Γενικά, έχει χαμηλότερη επιβάρυνση και μικρότερη κατανάλωση πόρων σε σύγκριση με το Kubernetes.	Συχνά απαιτεί περισσότερους πόρους για να λειτουργήσει αποτελεσματικά λόγω της σύνθετης και πολύπλοκης φύσης του.
Χαρακτηριστικά διαχείρισης	Απλούστερες και πιο απλές λειτουργίες διαχείρισης, ιδανικές για μικρότερες, λιγότερο σύνθετες εγκαταστάσεις.	Προσφέρει ένα ευρύτερο σύνολο χαρακτηριστικών για σύνθετες αναπτύξεις, συμπεριλαμβανομένης της καλύτερης υποστήριξης για stateful εφαρμογές και χαρακτηριστικών αυτοματοποίησης όπως η αυτο-ίαση, η ανακάλυψη υπηρεσιών και η διαχείριση μυστικών.
Κοινοτική υποστήριξη	Έχει μεγάλη κοινότητα, αλλά μικρότερη από το Kubernetes, με λιγότερες ενημερώσεις και προσθήκες λειτουργιών.	Πολύ μεγάλη υποστήριξη από την κοινότητα με συχνές ενημερώσεις και ένα τεράστιο οικοσύστημα συνεργατών και εργαλείων.

Η επιλογή μεταξύ του Docker Swarm και του Kubernetes εξαρτάται συχνά από συγκεκριμένες ανάγκες. Το Docker Swarm είναι κατάλληλο για προγραμματιστές και μικρές ομάδες που επιθυμούν να αναπτύξουν γρήγορα εφαρμογές σε ένα περιβάλλον με επίκεντρο το Docker και με λιγότερη πολυπλοκότητα. Από την άλλη πλευρά, το Kubernetes είναι καταλληλότερο για μεγαλύτερες επιχειρήσεις ή εφαρμογές που απαιτούν ισχυρή κλιμάκωση, εκτεταμένη αυτοματοποίηση και προηγμένες στρατηγικές ανάπτυξης. Και οι δύο πλατφόρμες προσφέρουν ισχυρά εργαλεία για τη διαχείριση των περιεκτών, αλλά το Kubernetes προηγείται όσον αφορά τα χαρακτηριστικά και την υιοθέτηση από τον κλάδο, καθιστώντας το προτιμότερη επιλογή για πολύπλοκες και μεγάλης κλίμακας αναπτύξεις.

Κεφάλαιο 4: Κατανόηση των μικρουπηρεσιών

4.1 Εισαγωγή στις μικροϋπηρεσίες

Οι μικροϋπηρεσίες είναι ανεξάρτητα μικρές και αρθρωτές αναπτύξιμες υπηρεσίες που στο σύνολο τους αποτελούν μία ενιαία εφαρμογή ακολουθώντας ένα είδος αρχιτεκτονικής προσανατολισμένης στις υπηρεσίες (SOA). Κάθε μικροϋπηρεσία υποστηρίζει ένα συγκεκριμένο σκοπό και χρησιμοποιεί μία απλή, σαφώς καθορισμένη διεπαφή για να επικοινωνεί με τις υπόλοιπες.

Τα πλεονεκτήματα των μικρουπηρεσιών είναι πολλά και ποικίλα. Ο ανεξάρτητος χαρακτήρας των αναπτύξεων ανοίγει νέα μοντέλα για τη βελτίωση της κλίμακας και της ευρωστίας των συστημάτων, και επιτρέπει τον συνδυασμό των διαφόρων τεχνολογιών. Παρέχουν δηλαδή τη δυνατότητα να διαφοροποιήσουμε τις τεχνολογικές επιλογές που κάνουμε, ίσως αναμειγνύοντας διαφορετικές γλώσσες προγραμματισμού, στυλ προγραμματισμού, πλατφόρμες ανάπτυξης ή βάσεις δεδομένων για να βρούμε τον κατάλληλο συνδυασμό. Επιπλέον μπορούν να δουλεύονται παράλληλα, οπότε περισσότεροι προγραμματιστές εστιάζουν σε ένα πρόβλημα χωρίς να μπλέκονται ο ένας στο δρόμο του άλλου.

4.2 Αρχιτεκτονική των μικρουπηρεσιών

Η αρχιτεκτονική των μικρουπηρεσιών επικεντρώνεται στη δημιουργία εφαρμογών ως μια σουίτα μικρών, ανεξάρτητα αναπτύξιμων υπηρεσιών. Κάθε υπηρεσία εκτελεί τη δική της διαδικασία και επικοινωνεί με άλλες υπηρεσίες μέσω ενός καλά καθορισμένου, ελαφρού μηχανισμού, συνήθως ενός API βασισμένου στο HTTP. Εδώ εμβαθύνουμε στις θεμελιώδεις αρχές και τα στοιχεία που καθορίζουν την αρχιτεκτονική μικρουπηρεσιών, παρέχοντας μια λεπτομερή κατανόηση του τρόπου με τον οποίο αυτά τα στοιχεία συνεργάζονται για τη δημιουργία ισχυρών, κλιμακούμενων και ευέλικτων εφαρμογών.

4.2.1 Αρχές σχεδιασμού

Οι αρχές σχεδιασμού της αρχιτεκτονικής μικρουπηρεσιών είναι απαραίτητο να κατανοηθούν, καθώς αποτελούν το θεμέλιο πάνω στο οποίο χτίζονται οι μικρουπηρεσίες. Διασφαλίζουν ότι οι μικρουπηρεσίες δεν είναι μόνο αποτελεσματικές αλλά και αποδοτικές στη λειτουργία και την αλληλεπίδρασή τους.

- Αποκέντρωση : Μια από τις βασικές αρχές των μικρουπηρεσιών είναι η αποκέντρωση. Σε αντίθεση με τις μονολιθικές αρχιτεκτονικές όπου η διακυβέρνηση και η διαχείριση δεδομένων είναι συγκεντρωτικές, οι μικροϋπηρεσίες προωθούν την αποκεντρωμένη διαχείριση και διακυβέρνηση δεδομένων. Αυτή η προσέγγιση επιτρέπει μεγαλύτερη

ευελιξία, καθώς κάθε ομάδα μπορεί να λαμβάνει αποφάσεις ανεξάρτητα χωρίς να περιμένει κεντρική έγκριση. Αυτή η αυτονομία επιτρέπει την ταχύτερη λήψη αποφάσεων και την καινοτομία, η οποία είναι ζωτικής σημασίας σε ένα ταχέως μεταβαλλόμενο περιβάλλον όπως η έξυπνη γεωργία.

- Ανεξαρτησία : Οι μικρουπηρεσίες έχουν σχεδιαστεί ώστε να είναι ανεξάρτητα αναπτύξιμες και επεκτάσιμες. Κάθε υπηρεσία ενθυλακώνει μια συγκεκριμένη επιχειρησιακή λειτουργία και μπορεί να αναπτυχθεί και να κλιμακωθεί ανεξάρτητα από άλλες υπηρεσίες. Αυτή η ανεξαρτησία επιτρέπει τη συνεχή ανάπτυξη και παράδοση, μειώνοντας το χρόνο διάθεσης στην αγορά για νέα χαρακτηριστικά και ενημερώσεις. Σημαίνει επίσης ότι μια αποτυχία σε μια υπηρεσία δεν επηρεάζει άμεσα τη διαθεσιμότητα άλλων υπηρεσιών, ενισχύοντας τη συνολική ανθεκτικότητα της εφαρμογής.

- Κάνε ένα πράγμα καλά : Εμπνευσμένες από τη φιλοσοφία του UNIX, οι μικρουπηρεσίες ακολουθούν την αρχή του να κάνεις ένα πράγμα καλά. Κάθε υπηρεσία επικεντρώνεται σε μία μόνο εργασία ή επιχειρησιακή δυνατότητα, γεγονός που απλοποιεί την ανάπτυξη, τον έλεγχο και τη συντήρηση. Αυτή η εξειδίκευση διασφαλίζει ότι οι υπηρεσίες βελτιστοποιούνται για τη συγκεκριμένη λειτουργία τους, οδηγώντας σε καλύτερες επιδόσεις και αξιοπιστία.

4.2.2 Στοιχεία των μικρουπηρεσιών

Η αρχιτεκτονική μικρουπηρεσιών αποτελείται από διάφορα κρίσιμα συστατικά που επιτρέπουν την απρόσκοπτη αλληλεπίδραση και διαχείριση των υπηρεσιών. Τα συστατικά αυτά παρέχουν την απαραίτητη υποδομή για την υποστήριξη της modular και δυναμικής φύσης των μικρουπηρεσιών.

- Ανακάλυψη υπηρεσιών: Σε μια αρχιτεκτονική μικρουπηρεσιών, οι υπηρεσίες πρέπει να ανακαλύπτουν και να επικοινωνούν δυναμικά μεταξύ τους. Οι μηχανισμοί ανακάλυψης υπηρεσιών, όπως η Eureka, η Consul ή η ενσωματωμένη ανακάλυψη υπηρεσιών του Kubernetes, επιτρέπουν στις υπηρεσίες να εγγράφονται και να ανακαλύπτουν άλλες υπηρεσίες. Αυτή η δυναμική ανακάλυψη είναι ζωτικής σημασίας σε ένα περιβάλλον όπου οι υπηρεσίες ενημερώνονται, κλιμακώνονται ή μετακινούνται συχνά

- API Gateway : Ένα API Gateway λειτουργεί ως ένα ενιαίο σημείο εισόδου για όλους τους clients για την αλληλεπίδραση με τις διάφορες υπηρεσίες. Χειρίζεται τα αιτήματα δρομολογώντας τα στην κατάλληλη υπηρεσία, διαχειρίζεται μεταφράσεις πρωτοκόλλων, συγκεντρώνει απαντήσεις και επιβάλλει πολιτικές ασφαλείας. Οι πύλες API, όπως η Zuul ή η NGINX, συμβάλλουν στην απλοποίηση των αλληλεπιδράσεων των clients και στην κεντρική διαχείριση οριζόντιων προβλημάτων, όπως ο έλεγχος ταυτότητας, το logging και το rate limiting.

- Circuit Breaker : Οι Circuit Breakers είναι ένα πρότυπο σχεδίασης που χρησιμοποιείται για την ανίχνευση αποτυχιών και την αποτροπή αλυσιδωτών προβλημάτων στην αρχιτεκτονική μικρουπηρεσιών. Εργαλεία όπως το Hystrix ή το Resilience4j υλοποιούν διακόπτες κυκλώματος που παρακολουθούν τις κλήσεις υπηρεσιών και, εάν μια υπηρεσία αποτύχει ή αργήσει να ανταποκριθεί, σταματούν τις κλήσεις προς τη συγκεκριμένη υπηρεσία. Με τον τρόπο αυτό αποτρέπεται η υπερφόρτωση του συστήματος και επιτρέπεται η ομαλή υποβάθμιση της λειτουργικότητας, διασφαλίζοντας ότι το συνολικό σύστημα παραμένει λειτουργικό.

- Διαχείριση ρυθμίσεων: Η διαχείριση των ρυθμίσεων για πολλαπλές υπηρεσίες μπορεί να είναι πολύπλοκη. Τα εργαλεία διαχείρισης διαμόρφωσης, όπως το Spring Cloud Config ή το Consul, εξωτερικεύουν τις διαμορφώσεις των υπηρεσιών, επιτρέποντας την κεντρική διαχείριση και την ενημέρωσή τους χωρίς να απαιτείται αναδιανομή των υπηρεσιών. Αυτή η συγκεντρωτική διαχείριση απλοποιεί τη διαδικασία διαμόρφωσης και διασφαλίζει τη συνοχή σε όλα τα περιβάλλοντα.

- Διαχείριση δεδομένων: Αυτό σημαίνει ότι κάθε υπηρεσία διαχειρίζεται τη δική της βάση δεδομένων. Αυτή η απομόνωση βοηθά στη διατήρηση της ανεξαρτησίας των υπηρεσιών, αλλά εισάγει προκλήσεις που σχετίζονται με τη συνέπεια των δεδομένων και τις συναλλαγές. Για τη διαχείριση της συνέπειας των δεδομένων σε όλες τις υπηρεσίες χρησιμοποιούνται τεχνικές όπως το event sourcing και το CQRS (Command Query Responsibility Segregation).

- Επικοινωνία μεταξύ υπηρεσιών: Η επικοινωνία μεταξύ υπηρεσιών μπορεί να είναι σύγχρονη ή ασύγχρονη. Η σύγχρονη επικοινωνία χρησιμοποιεί συνήθως HTTP/REST ή gRPC, επιτρέποντας στις υπηρεσίες να καλούν απευθείας η μία την άλλη. Η ασύγχρονη επικοινωνία χρησιμοποιεί συστήματα ανταλλαγής μηνυμάτων όπως το RabbitMQ, το Kafka ή το Amazon SNS/SQS, επιτρέποντας στις υπηρεσίες να επικοινωνούν μέσω μηνυμάτων χωρίς να περιμένουν άμεση απάντηση. Η ασύγχρονη επικοινωνία είναι ιδιαίτερα χρήσιμη για την αποσύνδεση υπηρεσιών και την ενίσχυση της ανθεκτικότητας του συστήματος.

- Καταγραφή και παρακολούθηση: Η παρατηρησιμότητα είναι κρίσιμη σε μια αρχιτεκτονική μικρουπηρεσιών. Τα κεντρικά εργαλεία καταγραφής και παρακολούθησης, όπως το ELK Stack (Elasticsearch, Logstash, Kibana), το Prometheus και το Grafana, συλλέγουν και αναλύουν τα αρχεία καταγραφής και τις μετρήσεις από όλες τις υπηρεσίες. Αυτά τα εργαλεία παρέχουν πληροφορίες σχετικά με την υγεία και την απόδοση της εφαρμογής, βοηθώντας στον γρήγορο εντοπισμό και την αντιμετώπιση προβλημάτων.

4.2.3 Dockerized μικροϋπηρεσίες

Το Dockerizing μικροϋπηρεσιών περιλαμβάνει τη δημιουργία κάθε μικροϋπηρεσίας σε ένα περιέλτη Docker, επιτρέποντας τη συνεπή και απομονωμένη εκτέλεση σε διαφορετικά περιβάλλοντα. Αυτή η διαδικασία είναι απαραίτητη για την επίτευξη των πλεονεκτημάτων της αρχιτεκτονικής μικροϋπηρεσιών, όπως η επεκτασιμότητα, η φορητότητα και η αποδοτική χρήση των πόρων καθώς απλοποιεί την ανάπτυξη και μειώνει τις πιθανότητες εμφάνισης προβλημάτων που σχετίζονται με το περιβάλλον.

Τα βήματα που πρέπει να ακολουθηθούν για να γίνουν οι μικρουπηρεσίες Dockerized είναι:

- Δημιουργία ενός αρχείου Docker:
  Ένα αρχείο Docker είναι ένα σενάριο που περιέχει μια σειρά από οδηγίες για το πώς να δημιουργηθεί μια εικόνα Docker για μια συγκεκριμένη μικροϋπηρεσία. Κάθε Dockerfile θα πρέπει να είναι προσαρμοσμένο στις απαιτήσεις της μικροϋπηρεσίας που πακετάρει.

- Κατασκευή της εικόνας Docker:
  Χρησιμοποιώντας το Dockerfile, κατασκευάζεται η εικόνα Docker για τη μικροϋπηρεσία. Αυτή η εικόνα περιέχει όλα όσα απαιτούνται για την εκτέλεση της μικροϋπηρεσίας, συμπεριλαμβανομένου του κώδικα της εφαρμογής, του χρόνου εκτέλεσης, των βιβλιοθηκών και των εξαρτήσεων.

- Εκτέλεση του περιέκτη Docker:
  Μόλις κατασκευαστεί το image, εκτελείται ο περιέκτης. Αυτό το βήμα περιλαμβάνει την εκκίνηση της μικροϋπηρεσίας σε περιέκτη, επιτρέποντάς της να λειτουργεί σε ένα απομονωμένο περιβάλλον.

- Χρησιμοποίηση του Docker Compose για εφαρμογές πολλαπλών περιεκτών:
  Σε μια αρχιτεκτονική μικροϋπηρεσιών, πολλαπλές υπηρεσίες πρέπει να συντονίζονται. Το Docker Compose είναι ένα εργαλείο για τον ορισμό και την εκτέλεση εφαρμογών Docker με πολλαπλούς περιέκτες. Με ένα αρχείο docker-compose.yml, καθρίζεται ο τρόπος με τον οποίο αλληλεπιδρούν πολλαπλοί περιέκτες και η διαχείρηση τους ως ενιαία εφαρμογή.

Το dockerization μικροϋπηρεσιών τυποποιεί τις διαδικασίες ανάπτυξης και διάθεσης, διασφαλίζοντας ότι οι μικροϋπηρεσίες εκτελούνται με συνέπεια σε διαφορετικά περιβάλλοντα. Αυτή η προσέγγιση ενισχύει τη φορητότητα, καθώς οι Docker περιέκτες μπορούν εύκολα να μετακινηθούν μεταξύ περιβαλλόντων ανάπτυξης, δοκιμών και παραγωγής χωρίς αλλαγές. Βελτιώνει επίσης τη χρήση των πόρων απομονώνοντας τις υπηρεσίες σε ελαφριούς περιέκτες, επιτρέποντας την αποτελεσματική κλιμάκωση και διαχείριση.

4.3 Πλεονεκτήματα των μικρουπηρεσιών

Η αρχιτεκτονική των μικρουπηρεσιών προσφέρει πολλά επιτακτικά πλεονεκτήματα που την καθιστούν ελκυστική επιλογή για την ανάπτυξη σύγχρονων εφαρμογών, ιδίως σε δυναμικά και πολύπλοκα περιβάλλοντα όπως η έξυπνη γεωργία. Η παρούσα ενότητα διερευνά τα βασικά πλεονεκτήματα της υιοθέτησης μιας προσέγγισης μικρουπηρεσιών, εστιάζοντας στην επεκτασιμότητα, την ευελιξία στην ανάπτυξη, την ανθεκτικότητα και την τεχνολογική ποικιλομορφία.

4.3.1 Επεκτασιμότητα

Ένα από τα σημαντικότερα πλεονεκτήματα της αρχιτεκτονικής μικρουπηρεσιών είναι η ικανότητά της να κλιμακώνει αποτελεσματικά τις εφαρμογές. Κάθε μικρουπηρεσία μπορεί να κλιμακωθεί ανεξάρτητα με βάση τις συγκεκριμένες απαιτήσεις φορτίου και επιδόσεων. Για παράδειγμα, εάν μια υπηρεσία παρακολούθησης καλλιεργειών παρουσιάζει υψηλή ζήτηση κατά τη διάρκεια της καλλιεργητικής περιόδου, μπορεί να κλιμακωθεί χωρίς να επηρεάσει άλλες υπηρεσίες, όπως η πρόγνωση καιρού ή η ανάλυση εδάφους. Με την κλιμάκωση μόνο των υπηρεσιών που απαιτούν πρόσθετους πόρους, η αρχιτεκτονική μικρουπηρεσιών διασφαλίζει τη βέλτιστη χρήση του υλικού και της υποδομής, μειώνοντας το κόστος και βελτιώνοντας τη συνολική απόδοση του συστήματος. Σε περιβάλλοντα νέφους, οι μικρουπηρεσίες μπορούν να επωφεληθούν από τα χαρακτηριστικά αυτόματης κλιμάκωσης που παρέχουν οι πλατφόρμες νέφους. Αυτό επιτρέπει στις εφαρμογές να χειρίζονται δυναμικά τα ποικίλα φορτία, κάτι που είναι ιδιαίτερα χρήσιμο στη γεωργία, όπου οι απαιτήσεις σε πόρους μπορεί να αυξομειώνονται με βάση τις εποχικές δραστηριότητες.

4.3.2 Ευελιξία στην ανάπτυξη

Η αρχιτεκτονική των μικρουπηρεσιών προωθεί μια ευέλικτη και αποτελεσματική διαδικασία ανάπτυξης, επιτρέποντας στις ομάδες να καινοτομούν και να παραδίδουν λειτουργίες ταχύτερα. Διαφορετικές ομάδες μπορούν να εργάζονται ταυτόχρονα σε διαφορετικές υπηρεσίες, χρησιμοποιώντας τα καταλληλότερα εργαλεία και τεχνολογίες για τα συγκεκριμένα καθήκοντά τους. Αυτή η αποκέντρωση επιταχύνει τους κύκλους ανάπτυξης και μειώνει τις εξαρτήσεις μεταξύ των ομάδων. Οι μικρουπηρεσίες υποστηρίζουν πρακτικές συνεχούς ολοκλήρωσης και συνεχούς παράδοσης (CI/CD), επιτρέποντας πιο συχνές και αξιόπιστες εκδόσεις. Κάθε υπηρεσία μπορεί να δοκιμαστεί, να αναπτυχθεί και να ενημερωθεί ανεξάρτητα, διασφαλίζοντας ότι νέα χαρακτηριστικά ή διορθώσεις σφαλμάτων μπορούν να κυκλοφορήσουν γρήγορα χωρίς να διαταραχθεί ολόκληρο το σύστημα. Επιπλέον, οι ομάδες έχουν την ελευθερία να επιλέγουν τις καλύτερες γλώσσες προγραμματισμού και πλαίσια για τις υπηρεσίες τους, επιτρέποντας στους προγραμματιστές να αξιοποιούν τις πιο πρόσφατες τεχνολογίες και τις βέλτιστες πρακτικές προσαρμοσμένες στις συγκεκριμένες απαιτήσεις των υπηρεσιών τους.

4.3.3 Ανθεκτικότητα

Η ανθεκτικότητα είναι ένα κρίσιμο χαρακτηριστικό της αρχιτεκτονικής μικρουπηρεσιών, διασφαλίζοντας ότι οι εφαρμογές παραμένουν εύρωστες και αξιόπιστες ακόμη και σε περίπτωση αποτυχιών. Η αποτυχία μιας υπηρεσίας δεν προκαλεί την αποτυχία ολόκληρης της εφαρμογής. Για παράδειγμα, εάν μια υπηρεσία που είναι υπεύθυνη για την αποστολή ειδοποιήσεων αντιμετωπίσει κάποιο πρόβλημα, άλλες υπηρεσίες, όπως η συλλογή και η ανάλυση δεδομένων, μπορούν να συνεχίσουν να λειτουργούν κανονικά. Οι αρχιτεκτονικές μικρουπηρεσιών έχουν σχεδιαστεί για να χειρίζονται τις αποτυχίες με αξιοπρέπεια. Χρησιμοποιώντας πρότυπα όπως οι Circuit Breakers και τα Bulkheads, τα συστήματα μπορούν να ανιχνεύουν και να απομονώνουν αποτυχίες, ανακατευθύνοντας την κυκλοφορία ή υποβαθμίζοντας τη λειτουργικότητα με ελεγχόμενο τρόπο. Σε περίπτωση αποτυχίας, οι μικρουπηρεσίες μπορούν να επανεκκινήσουν ή να επανατοποθετηθούν γρήγορα, ελαχιστοποιώντας τον χρόνο διακοπής λειτουργίας. Οι αυτοματοποιημένες διαδικασίες αποκατάστασης, όπως οι δυνατότητες αυτοθεραπείας που παρέχονται από εργαλεία ενορχήστρωσης όπως το Kubernetes, ενισχύουν περαιτέρω την ανθεκτικότητα του συστήματος.

4.3.4 Τεχνολογική ποικιλομορφία

Οι μικρουπηρεσίες επιτρέπουν τη χρήση ποικίλων τεχνολογιών, επιτρέποντας στους οργανισμούς να υιοθετήσουν τα καλύτερα εργαλεία και πρακτικές για κάθε συγκεκριμένη υπηρεσία. Οι ομάδες μπορούν να χρησιμοποιούν διαφορετικές γλώσσες προγραμματισμού για διαφορετικές υπηρεσίες, βελτιστοποιώντας την απόδοση και αξιοποιώντας τα πλεονεκτήματα κάθε γλώσσας. Για παράδειγμα, μια υπηρεσία μηχανικής μάθησης για την πρόβλεψη καλλιεργειών μπορεί να χρησιμοποιεί Python, ενώ μια υπηρεσία επεξεργασίας δεδομένων σε πραγματικό χρόνο μπορεί να χρησιμοποιεί Go ή Node.js. Οι μικρουπηρεσίες μπορούν να ενσωματώσουν διάφορα εργαλεία και πλαίσια που ταιριάζουν καλύτερα στις συγκεκριμένες ανάγκες τους, είτε πρόκειται για μια συγκεκριμένη τεχνολογία βάσης δεδομένων, ένα σύστημα ανταλλαγής μηνυμάτων ή ένα πλαίσιο καταγραφής. Η αρθρωτή φύση των μικρουπηρεσιών ενθαρρύνει τον πειραματισμό με νέες τεχνολογίες και προσεγγίσεις. Οι ομάδες μπορούν να δοκιμάζουν νέες λύσεις σε απομονωμένες υπηρεσίες χωρίς να διακινδυνεύουν τη σταθερότητα ολόκληρης της εφαρμογής, προωθώντας μια κουλτούρα καινοτομίας.

4.4 Εφαρμογή των μικρουπηρεσιών

Η υλοποίηση μικρουπηρεσιών περιλαμβάνει μια σειρά από στρατηγικές αποφάσεις και πρακτικές που διασφαλίζουν την αποτελεσματικότητα, την επεκτασιμότητα και τη συντηρησιμότητα της αρχιτεκτονικής. Αυτή η ενότητα εξετάζει τις βασικές πτυχές της υλοποίησης μικρουπηρεσιών, συμπεριλαμβανομένης της επιλογής μιας στρατηγικής ανάπτυξης, της διαχείρισης δεδομένων και του χειρισμού της επικοινωνίας μεταξύ των υπηρεσιών.

4.4.1 Επιλογή στρατηγικής ανάπτυξης

Κατά την υλοποίηση μικρουπηρεσιών, μία από τις πρώτες αποφάσεις είναι η επιλογή μιας κατάλληλης στρατηγικής ανάπτυξης. Οι περιέκτες χρησιμοποιούνται συχνά για την ενθυλάκωση των μικρουπηρεσιών, παρέχοντας ένα απομονωμένο περιβάλλον για κάθε υπηρεσία. Αυτή η απομόνωση διασφαλίζει ότι οι υπηρεσίες δεν αλληλεπιδρούν μεταξύ τους, επιτρέποντας μεγαλύτερη σταθερότητα και προβλεψιμότητα. Μπορούν εύκολα να μετακινηθούν σε διαφορετικά περιβάλλοντα, από την ανάπτυξη στην παραγωγή, εξασφαλίζοντας συνέπεια. Τα εργαλεία ενορχήστρωσης, όπως το Docker Swarm και το Kubernetes, διαχειρίζονται αυτές τις υπηρεσίες που περιέχουν περιέκτες, χειριζόμενα την ανάπτυξη, την κλιμάκωση και τη δικτύωση. Αυτοματοποιούν πολλές λειτουργικές εργασίες, όπως η εξισορρόπηση φορτίου, η αυτοθεραπεία και η διαχείριση μυστικών, οι οποίες είναι απαραίτητες για τη διατήρηση μιας στιβαρής και ανθεκτικής αρχιτεκτονικής εφαρμογών

4.4.2 Διαχείριση δεδομένων

Η διαχείριση δεδομένων σε μια αρχιτεκτονική μικρουπηρεσιών μπορεί να είναι πολύπλοκη λόγω της αποκεντρωμένης φύσης των υπηρεσιών. Μια προσέγγιση είναι η υλοποίηση ενός προτύπου "βάση δεδομένων ανά υπηρεσία", όπου κάθε μικρουπηρεσία διαχειρίζεται τη δική της βάση δεδομένων. Αυτή η απομόνωση εξασφαλίζει ότι οι υπηρεσίες είναι αποσυνδεδεμένες στο επίπεδο δεδομένων, επιτρέποντας την ανεξάρτητη κλιμάκωση και ανάπτυξη. Ωστόσο, αυτό εισάγει επίσης προκλήσεις που σχετίζονται με τη συνέπεια των δεδομένων και τις συναλλαγές μεταξύ των υπηρεσιών. Τεχνικές όπως το event sourcing και ο διαχωρισμός ευθύνης ερωτήσεων εντολών (Command Query Responsibility Segregation - CQRS) μπορούν να βοηθήσουν στη διαχείριση αυτών των επιπλοκών. Το event sourcing περιλαμβάνει την καταγραφή όλων των αλλαγών στην κατάσταση της εφαρμογής ως μια ακολουθία γεγονότων, τα οποία μπορούν να αναπαραχθούν για την ανακατασκευή της τρέχουσας κατάστασης. Το CQRS διαχωρίζει τις λειτουργίες που διαβάζουν δεδομένα από εκείνες που γράφουν δεδομένα, επιτρέποντας πιο ευέλικτες και κλιμακούμενες αρχιτεκτονικές.

4.4.3 Χειρισμός της επικοινωνίας μεταξύ υπηρεσιών

Η αποτελεσματική επικοινωνία μεταξύ των υπηρεσιών είναι ζωτικής σημασίας σε μια αρχιτεκτονική μικρουπηρεσιών. Η επικοινωνία αυτή μπορεί να είναι σύγχρονη ή ασύγχρονη, ανάλογα με τις απαιτήσεις του συστήματος. Η σύγχρονη επικοινωνία χρησιμοποιεί συνήθως HTTP/REST ή gRPC, επιτρέποντας στις υπηρεσίες να καλούν απευθείας η μία την άλλη και να λαμβάνουν άμεσες απαντήσεις. Αυτή η μέθοδος είναι απλή και εύκολη στην υλοποίηση, αλλά μπορεί να οδηγήσει σε στενή σύζευξη μεταξύ των υπηρεσιών και σε πιθανά σημεία συμφόρησης των επιδόσεων. Η ασύγχρονη επικοινωνία, από την άλλη πλευρά, χρησιμοποιεί συστήματα ανταλλαγής μηνυμάτων όπως το RabbitMQ, το Kafka ή το Amazon SNS/SQS. Αυτά τα συστήματα επιτρέπουν στις υπηρεσίες να επικοινωνούν μέσω μηνυμάτων χωρίς να περιμένουν άμεση απάντηση, γεγονός που αποσυνδέει τις υπηρεσίες και βελτιώνει την ανθεκτικότητα και την επεκτασιμότητα του συστήματος. Η ασύγχρονη επικοινωνία είναι ιδιαίτερα χρήσιμη για εργασίες που δεν απαιτούν άμεσες απαντήσεις, όπως η καταγραφή, η παρακολούθηση ή η επεξεργασία δέσμης.

4.4.4 Καταγραφή και παρακολούθηση

Η παρατηρησιμότητα είναι ζωτικής σημασίας σε μια αρχιτεκτονική μικρουπηρεσιών για να διασφαλιστεί ότι το σύστημα λειτουργεί ομαλά και ότι τυχόν προβλήματα εντοπίζονται και επιλύονται γρήγορα. Τα κεντρικά εργαλεία καταγραφής και παρακολούθησης, όπως το ELK Stack (Elasticsearch, Logstash, Kibana), το Prometheus και το Grafana, συλλέγουν και αναλύουν τα αρχεία καταγραφής και τις μετρήσεις από όλες τις υπηρεσίες. Αυτά τα εργαλεία παρέχουν μια ολοκληρωμένη εικόνα της υγείας και των επιδόσεων της εφαρμογής, βοηθώντας στον εντοπισμό σημείων συμφόρησης των επιδόσεων, στον εντοπισμό ανωμαλιών και στην αντιμετώπιση προβλημάτων. Η αποτελεσματική καταγραφή και παρακολούθηση επιτρέπουν την προληπτική συντήρηση και βελτιώνουν τη συνολική αξιοπιστία του συστήματος.

4.5 Προκλήσεις των μικρουπηρεσιών

Ενώ η αρχιτεκτονική μικρουπηρεσιών προσφέρει πολλά πλεονεκτήματα, εισάγει επίσης αρκετές προκλήσεις που πρέπει να αντιμετωπίσουν οι οργανισμοί για να διασφαλίσουν την επιτυχή υλοποίηση και λειτουργία. Εδώ εξετάζουμε τις βασικές προκλήσεις που σχετίζονται με τις

μικρουπηρεσίες, συμπεριλαμβανομένης της πολυπλοκότητας, της ακεραιότητας των δεδομένων, των δοκιμών και της παρακολούθησης και των οργανωτικών αλλαγών.

4.5.1 Πολυπλοκότητα

Η αρχιτεκτονική μικρουπηρεσιών αυξάνει εγγενώς την πολυπλοκότητα του συστήματος λόγω του μεγάλου αριθμού υπηρεσιών που πρέπει να διαχειριστούν και να ενορχηστρωθούν. Κάθε υπηρεσία είναι μια ξεχωριστή οντότητα με τη δικό της codebase, deployment pipeline και απαιτήσεις υποδομής. Αυτός ο αποκεντρωμένος χαρακτήρας απαιτεί εξελιγμένα εργαλεία διαχείρισης και ενορχήστρωσης για τον χειρισμό της ανακάλυψης υπηρεσιών, της εξισορρόπησης φορτίου, της διαχείρισης ρυθμίσεων και πολλά άλλα. Η αυξημένη πολυπλοκότητα μπορεί να οδηγήσει σε υψηλότερο λειτουργικό κόστος και απαιτεί μια ισχυρή υποδομή για τη διαχείριση των εξαρτήσεων και των αλληλεπιδράσεων μεταξύ των υπηρεσιών. Οι ομάδες πρέπει να υιοθετήσουν νέες διαδικασίες και εργαλεία για την αποτελεσματική διαχείριση αυτής της πολυπλοκότητας, διασφαλίζοντας ότι οι υπηρεσίες μπορούν να κλιμακώνονται και να λειτουργούν αξιόπιστα.

4.5.2 Ακεραιότητα δεδομένων

Η διατήρηση της ακεραιότητας των δεδομένων σε πολλαπλές μικρουπηρεσίες αποτελεί σημαντική πρόκληση. Σε μια μονολιθική αρχιτεκτονική, μια μεμονωμένη συναλλαγή μπορεί να διασφαλίσει τη συνοχή των δεδομένων. Ωστόσο, σε μια αρχιτεκτονική μικρουπηρεσιών, κάθε υπηρεσία συνήθως διαχειρίζεται τη δική της βάση δεδομένων, γεγονός που καθιστά δύσκολη τη διατήρηση των ιδιοτήτων ACID (Atomicity, Consistency, Isolation, Durability) σε όλες τις υπηρεσίες. Μπορούν να χρησιμοποιηθούν τεχνικές όπως η ενδεχόμενη συνέπεια (eventual consistency), όπου οι ενημερώσεις των δεδομένων διαδίδονται ασύγχρονα και η συνέπεια επιτυγχάνεται με την πάροδο του χρόνου. Ωστόσο, αυτή η προσέγγιση απαιτεί προσεκτικό σχεδιασμό και χειρισμό για να διασφαλιστεί ότι το σύστημα παραμένει αξιόπιστο και ακριβές. Η εφαρμογή προτύπων όπως το Saga, το οποίο συντονίζει τις συναλλαγές σε πολλαπλές υπηρεσίες, μπορεί να βοηθήσει στη διαχείριση των κατανεμημένων συναλλαγών, αλλά προσθέτει άλλο ένα επίπεδο πολυπλοκότητας στο σύστημα.

4.5.3 Δοκιμές και παρακολούθηση

Η δοκιμή μικρουπηρεσιών είναι πιο πολύπλοκη από τη δοκιμή μονολιθικών εφαρμογών λόγω του αριθμού των ανεξάρτητων υπηρεσιών και των αλληλεπιδράσεών τους. Οι παραδοσιακές στρατηγικές δοκιμών πρέπει να προσαρμοστούν για να φιλοξενήσουν δοκιμές ολοκλήρωσης που καλύπτουν πολλαπλές υπηρεσίες, διασφαλίζοντας ότι λειτουργούν μαζί όπως αναμένεται. Αυτό

απαιτεί μια ολοκληρωμένη στρατηγική δοκιμών που περιλαμβάνει δοκιμές μονάδας, δοκιμές ολοκλήρωσης και δοκιμές από άκρο σε άκρο. Επιπλέον, η παρακολούθηση μιας αρχιτεκτονικής μικρουπηρεσιών αποτελεί πρόκληση, επειδή περιλαμβάνει την παρακολούθηση της απόδοσης και της υγείας πολυάριθμων υπηρεσιών. Οι κεντρικές λύσεις καταγραφής και παρακολούθησης είναι απαραίτητες για τη συλλογή και ανάλυση δεδομένων από όλες τις υπηρεσίες. Εργαλεία όπως το Prometheus, το Grafana και το ELK Stack (Elasticsearch, Logstash, Kibana) παρέχουν την απαραίτητη υποδομή για την παρακολούθηση της απόδοσης των υπηρεσιών, τον εντοπισμό ανωμαλιών και την αντιμετώπιση προβλημάτων, διασφαλίζοντας την ομαλή λειτουργία του συστήματος.

4.5.4 Οργανωτικές αλλαγές

Η υιοθέτηση μιας αρχιτεκτονικής μικρουπηρεσιών συχνά απαιτεί σημαντικές οργανωτικές αλλαγές. Οι ομάδες πρέπει να μετακινηθούν από μια παραδοσιακή, συγκεντρωτική προσέγγιση σε ένα πιο αποκεντρωμένο, αυτόνομο μοντέλο. Αυτή η μετάβαση απαιτεί αλλαγές στη δομή της ομάδας, στις πρακτικές ανάπτυξης και στα πρωτόκολλα επικοινωνίας. Οι ομάδες πρέπει να είναι διαλειτουργικές, περιλαμβάνοντας όλες τις δεξιότητες που απαιτούνται για την ανάπτυξη, και τη συντήρηση των υπηρεσιών τους ανεξάρτητα. Αυτή η αυτονομία προάγει την ευελιξία και την καινοτομία, αλλά απαιτεί μια πολιτισμική στροφή προς τη συνεργασία και την κοινή ευθύνη. Οι οργανισμοί πρέπει να επενδύσουν στην εκπαίδευση και την υποστήριξη για να βοηθήσουν τις ομάδες να προσαρμοστούν σε αυτούς τους νέους ρόλους και διαδικασίες, διασφαλίζοντας ότι μπορούν να διαχειριστούν αποτελεσματικά την αυξημένη πολυπλοκότητα και τις αλληλεξαρτήσεις μιας αρχιτεκτονικής μικρουπηρεσιών.

Κεφάλαιο 5: Έξυπνη γεωργία Libelium

5.1 Εισαγωγή

Η Libelium είναι κορυφαίος πάροχος ασύρματων δικτύων αισθητήρων που έχουν σχεδιαστεί για να βελτιώνουν την αποδοτικότητα και την παραγωγικότητα των γεωργικών εργασιών μέσω λύσεων έξυπνης γεωργίας. Ιδρύθηκε το 2006 από την Alicia Asín και τον David Gascón. Η πλατφόρμα τους ενσωματώνει διάφορες τεχνολογίες για την παρακολούθηση των περιβαλλοντικών συνθηκών, τη βελτίωση της διαχείρισης των καλλιεργειών και τη βελτιστοποίηση της χρήσης των πόρων.

5.2 Μελέτες περιπτώσεων με προϊόντα της Libelium

5.2.1 Διαχείριση άρδευσης ακριβείας

Μια από τις πιο κρίσιμες εφαρμογές της πλατφόρμας έξυπνης γεωργίας της Libelium είναι η διαχείριση άρδευσης ακριβείας. Σε περιοχές όπου το νερό είναι ένας σπάνιος και πολύτιμος πόρος, η αποδοτική χρήση του νερού είναι ζωτικής και υψίστης σημασίας. Το σύστημα της Libelium χρησιμοποιεί ασύρματα δίκτυα αισθητήρων που είναι στρατηγικά τοποθετημένα σε όλους τους γεωργικούς αγρούς για την παρακολούθηση της εδαφικής υγρασίας, της θερμοκρασίας και της υγρασίας σε πραγματικό χρόνο. Αυτοί οι αισθητήρες παρέχουν ακριβή δεδομένα σχετικά με τις τρέχουσες συνθήκες του εδάφους, επιτρέποντας στους αγρότες μια λεπτομερή ανάλυση των δεδομένων έτσι ώστε να είναι σε θέση να λαμβάνουν τεκμηριωμένες αποφάσεις σχετικά με την άρδευση.

Η τεχνολογία πίσω από αυτό το σύστημα περιλαμβάνει την εικονικοποίηση με βάση τους περιέκτες χρησιμοποιώντας το Docker και το Kubernetes. Το Docker χρησιμοποιείται για την εμπορευματοποίηση των υπηρεσιών συλλογής και ανάλυσης δεδομένων, διασφαλίζοντας ότι μπορούν να εκτελούνται με συνέπεια σε διάφορες συσκευές, από κόμβους υπολογισμού ακραίων σημείων στον αγρό μέχρι διακομιστές cloud για πιο εντατική ανάλυση δεδομένων. Με τη σειρά του το Kubernetes ενορχηστρώνει αυτούς τους περιέκτες, παρέχοντας την απαραίτητη επεκτασιμότητα και ανοχή σε σφάλματα. Κατά τη διάρκεια των περιόδων αιχμής της άρδευσης, το Kubernetes μπορεί να κλιμακώσει αυτόματα τις υπηρεσίες επεξεργασίας δεδομένων για να χειριστεί το αυξημένο φορτίο, διασφαλίζοντας ότι οι συστάσεις άρδευσης είναι έγκαιρες και ακριβείς. Αυτό το σύστημα μειώνει τη χρήση νερού έως και 30%, διασφαλίζοντας ότι οι καλλιέργειες λαμβάνουν τη

βέλτιστη ποσότητα νερού, γεγονός που όχι μόνο εξοικονομεί νερό αλλά και βελτιώνει την απόδοση και την ποιότητα των καλλιεργειών.

5.2.2 Παρακολούθηση και έλεγχος του κλίματος

Η πλατφόρμα της Libelium υπερέχει επίσης στην παρακολούθηση και τον έλεγχο του κλίματος, ιδίως σε περιβάλλοντα θερμοκηπίων, όπου η διατήρηση των βέλτιστων κλιματικών συνθηκών είναι απαραίτητη για την υγεία και την παραγωγικότητα των καλλιεργειών. Το σύστημα περιλαμβάνει αισθητήρες που παρακολουθούν διάφορες περιβαλλοντικές συνθήκες, όπως η θερμοκρασία του αέρα, η υγρασία, η ηλιακή ακτινοβολία και τα επίπεδα CO2. Αυτή η ολοκληρωμένη παρακολούθηση επιτρέπει τον ακριβή έλεγχο του περιβάλλοντος του θερμοκηπίου.

Η υποκείμενη τεχνολογία αξιοποιεί τους περιέκτες Docker για την εκτέλεση εφαρμογών παρακολούθησης και ελέγχου του κλίματος. Αυτές οι εφαρμογές επεξεργάζονται δεδομένα από τους αισθητήρες και ελέγχουν ανάλογα τα συστήματα HVAC (θέρμανσης, αερισμού και κλιματισμού). Το Kubernetes χρησιμοποιείται για τη διαχείριση της ανάπτυξης αυτών των περιεκτών, εξασφαλίζοντας υψηλή διαθεσιμότητα και ανθεκτικότητα. Εάν ένας κόμβος εντός του συστήματος αποτύχει, το Kubernetes ανακατανείμει αυτόματα το φόρτο εργασίας για να διατηρηθεί η συνεχής παρακολούθηση και ο έλεγχος. Αυτός ο αυτοματοποιημένος κλιματικός έλεγχος διασφαλίζει ότι οι καλλιέργειες καλλιεργούνται σε βέλτιστες συνθήκες, οδηγώντας σε υψηλότερες αποδόσεις και καλύτερη ποιότητα προϊόντων. Επιπλέον, μειώνει την κατανάλωση ενέργειας βελτιστοποιώντας τη λειτουργία των συστημάτων HVAC, συμβάλλοντας στη συνολική βιωσιμότητα των γεωργικών πρακτικών.

5.2.3 Ανίχνευση παρασίτων και ασθενειών

Μια άλλη σημαντική εφαρμογή της πλατφόρμας έξυπνης γεωργίας της Libelium είναι η ανίχνευση παρασίτων και ασθενειών στις καλλιέργειες. Η έγκαιρη ανίχνευση και αντιμετώπιση παρασίτων και ασθενειών είναι υψίστης σημασίας για τη διατήρηση υγιών καλλιεργειών και την ελαχιστοποίηση των απωλειών των γεωργών. Η Libelium ενσωματώνει την πλατφόρμα της με συσκευές απεικόνισης και μοντέλα μηχανικής μάθησης για τον εντοπισμό επιβλαβών οργανισμών και ασθενειών σε πρώιμο στάδιο, έτσι ώστε να μην χάνεται πολύτιμος χρόνος για τον εντοπισμό τους. Καταγράφονται και αναλύονται εικόνες υψηλής ανάλυσης για τον εντοπισμό αυτών των ενδείξεων προσβολής ή μόλυνσης πριν αυτές επεκταθούν.

Η τεχνολογική υλοποίηση περιλαμβάνει τη χρήση περιεκτών Docker για την ενθυλάκωση της επεξεργασίας εικόνας και των μοντέλων μηχανικής μάθησης. Οι περιέκτες αυτοί διασφαλίζουν ότι τα μοντέλα μπορούν να εκτελούνται σε διάφορες διαμορφώσεις υλικού, από συσκευές άκρων στις καλλιέργειες μέχρι ισχυρούς διακομιστές cloud. Το Kubernetes ενορχηστρώνει αυτές τις υπηρεσίες,

κλιμακώνοντας τους υπολογιστικούς πόρους ανάλογα με τις ανάγκες για τη γρήγορη επεξεργασία μεγάλου όγκου δεδομένων εικόνας. Αυτό το σύστημα έγκαιρης ανίχνευσης επιτρέπει έγκαιρες παρεμβάσεις, μειώνοντας τις ζημιές στις καλλιέργειες και ελαχιστοποιώντας τη χρήση φυτοφαρμάκων. Αυτό όχι μόνο οδηγεί σε υγιέστερες καλλιέργειες, αλλά μειώνει και το κόστος παραγωγής μειώνοντας την εξάρτηση από χημικές επεξεργασίες.

5.2.4 Περιβαλλοντική παρακολούθηση και συμμόρφωση

Η πλατφόρμα της Libelium χρησιμοποιείται επίσης για την περιβαλλοντική παρακολούθηση και συμμόρφωση, βοηθώντας τους αγρότες να τηρούν τους περιβαλλοντικούς κανονισμούς και να βελτιστοποιούν τις γεωργικές τους πρακτικές. Το σύστημα περιλαμβάνει αισθητήρες που παρακολουθούν την ποιότητα του νερού, την ατμοσφαιρική ρύπανση και τη μόλυνση του εδάφους, παρέχοντας ολοκληρωμένα δεδομένα σχετικά με τις περιβαλλοντικές συνθήκες. Τα δεδομένα αυτά βοηθούν στη διασφάλιση ότι οι γεωργικές πρακτικές δεν επηρεάζουν αρνητικά το περιβάλλον με καμία επιβάρυνση και συμμορφώνονται με τα κανονιστικά πρότυπα που ισχύουν.

Οι υπηρεσίες περιβαλλοντικής παρακολούθησης είναι συγκεντρωμένες σε περιέκτες με χρήση Docker, επιτρέποντας τη συνεπή και αξιόπιστη λειτουργία σε διαφορετικές συσκευές και περιβάλλοντα. Το Kubernetes ενορχηστρώνει αυτούς τους περέκτες, διασφαλίζοντας ότι οι υπηρεσίες παρακολούθησης είναι επεκτάσιμες και ανθεκτικές. Αυτή η ρύθμιση επιτρέπει τη συνεχή παρακολούθηση και τις ειδοποιήσεις σε πραγματικό χρόνο, στη περίπτωση που οι περιβαλλοντικές συνθήκες υπερβαίνουν τα προκαθορισμένα όρια. Με τη στενή παρακολούθηση των περιβαλλοντικών παραγόντων, οι αγρότες μπορούν να λαμβάνουν προληπτικά μέτρα για τον μετριασμό τυχόν αρνητικών επιπτώσεων, προωθώντας έτσι βιώσιμες γεωργικές πρακτικές και βοηθώντας στην διατήρηση της ισορροπίας του περιβάλλοντος.

5.2.5 Ενσωμάτωση με IoT και Big Data Analytics

Η πλατφόρμα έξυπνης γεωργίας της Libelium ενσωματώνεται απρόσκοπτα με το IoT και τις αναλύσεις μεγάλων δεδομένων, παρέχοντας μια ολιστική προσέγγιση στη διαχείριση των γεωργικών εκμεταλλεύσεων. Η πλατφόρμα συλλέγει τεράστιες ποσότητες δεδομένων από διάφορους αισθητήρες και συσκευές, τα οποία στη συνέχεια αναλύονται για την εξαγωγή ουσιαστικών πληροφοριών και συμπερασμάτων. Αυτή η ενσωμάτωση επιτρέπει την προγνωστική ανάλυση, βοηθώντας τους αγρότες να προβλέπουν και να ανταποκρίνονται στις αλλαγές των περιβαλλοντικών συνθηκών, των πληθυσμών των παρασίτων και της υγείας των καλλιεργειών.

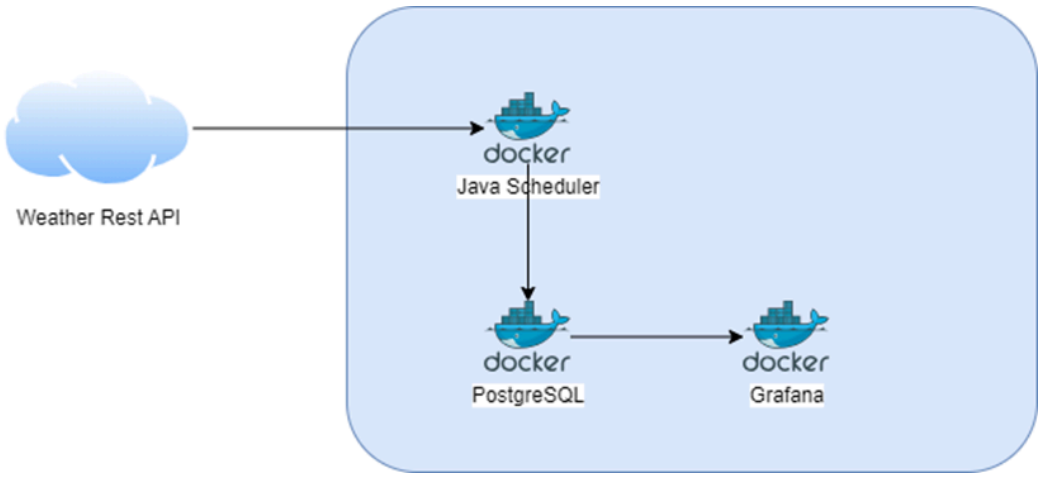
Η ενσωμάτωση τροφοδοτείται από το Docker και το Kubernetes, τα οποία παρέχουν την απαραίτητη υποδομή για τη διαχείριση μεγάλου όγκου δεδομένων και σύνθετων εργασιών ανάλυσης.Οι περιέκτες Docker διασφαλίζουν ότι οι εφαρμογές συλλογής και επεξεργασίας

δεδομένων μπορούν να εκτελούνται αξιόπιστα σε διαφορετικά περιβάλλοντα. Το Kubernetes με τη σειρά του διαχειρίζεται την ενορχήστρωση αυτών των εμπορευματοκιβωτίων, διασφαλίζοντας ότι το σύστημα μπορεί να κλιμακώνεται ώστε να διαχειρίζεται αυξημένα φορτία δεδομένων κατά τη διάρκεια κρίσιμων περιόδων, όπως οι εποχές φύτευσης ή συγκομιδής των καλλιεργειών. Έτσι αξιοποιώντας το IoT και την ανάλυση μεγάλων δεδομένων, η πλατφόρμα της Libelium βοηθά τους αγρότες να λαμβάνουν αποφάσεις βάσει δεδομένων, βελτιώνοντας την αποδοτικότητα και την παραγωγικότητα των εργασιών τους.

Κεφάλαιο 6: Ανάλυση εφαρμογής

6.1 Σχεδιασμός αρχιτεκτονικής συστήματος.

Σκοπός είναι η δημιουργία τριών περιεκτών (Docker containers) καθένα από τα οποία θα πραγματοποιεί ανεξάρτητες υπηρεσίες:



Docker 1 (Java Scheduler): Πραγματοποιήθηκε η ανάπτυξη ενός Scheduler σε Java και Spring Boot ο οποίος κάνει ανάκτηση δεδομένων για συγκεκριμένες πόλεις ανά τακτά χρονικά διαστήματα (σε χρονική περίοδο μιας ώρας) και αποθηκεύει τις μετρήσεις για κάθε πόλη στη βάση δεδομένων PostgreSQL.

Docker 2 (PostgreSQL): Πραγματοποιήθηκε η εγκατάσταση βάσης δεδομένων PostgreSQL στην οποία αποθηκεύονται τα δεδομένα που θα συλλέγονται από τον Scheduler (Docker 1). Έγινε σχεδιασμός του σχεσιακού μοντέλου (ER-Diagram) βάσει του οποίου δημιουργήθηκαν οι αντίστοιχοι πίνακες στη βάση δεδομένων.

Docker 3 (Grafana): Πραγματοποιήθηκε η εγκατάσταση του Grafana για τη γραφική απεικόνιση των δεδομένων των οποίων έχουν συλλεχθεί και αποθηκευτεί στη βάση δεδομένων.

6.2 Εγκατάσταση του Docker σε λειτουργικό Windows.

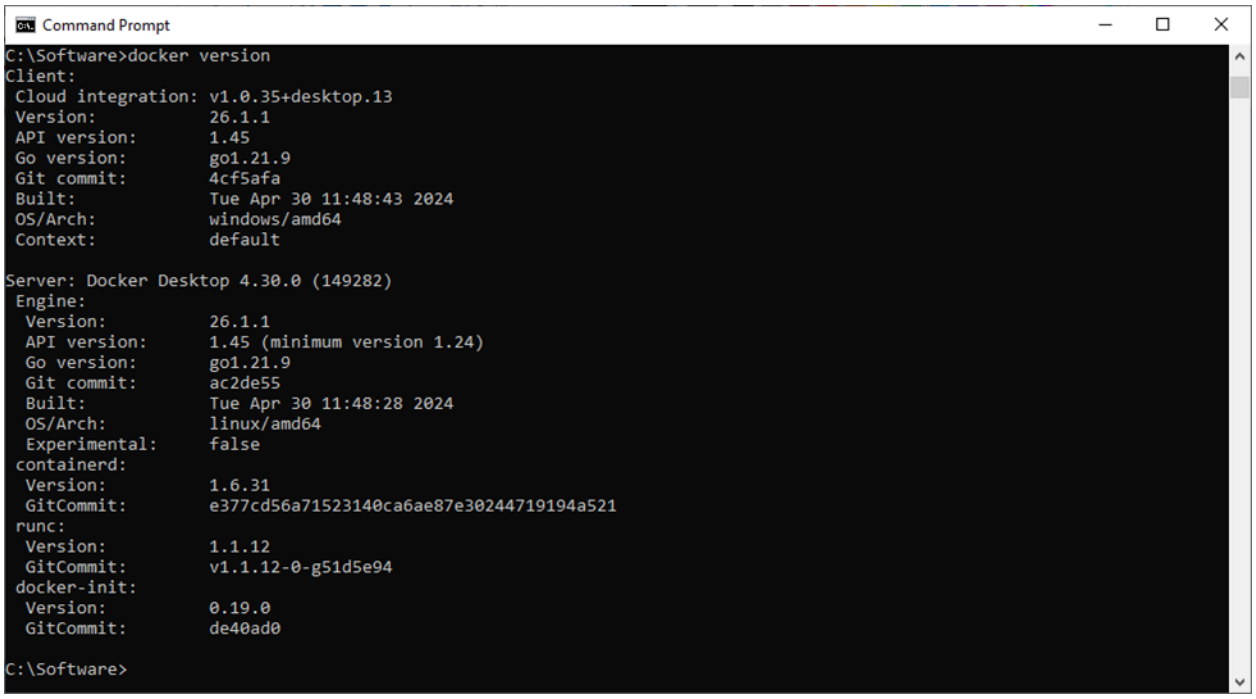
Το Docker είναι μια πλατφόρμα λογισμικού ανοιχτού κώδικα με την οποία μπορούμε να κάνουμε εικονικοποίηση (virtualization) σε επίπεδο λειτουργικού συστήματος. Δίνει τη δυνατότητα να εγκατασταθεί μόνο η επιθυμητή εφαρμογή / υπηρεσία (χωρίς την εγκατάσταση επιπλέον λειτουργικού συστήματος), σε ένα απομονωμένο περιβάλλον από το κανονικό σου σύστημα.

Η εγκατάσταση μπορεί να πραγματοποιηθεί είτε μέσω του παραδοσιακού installer είτε μέσω της γραμμής εντολών. Πραγματοποιήθηκε εγκατάσταση του Docker (μέσω του installer) σε περιβάλλον Windows 11.

Για να δούμε τις πληροφορίες της εγκατάστασης εκτελούμε την ακόλουθη εντολή:

```
> docker version
```

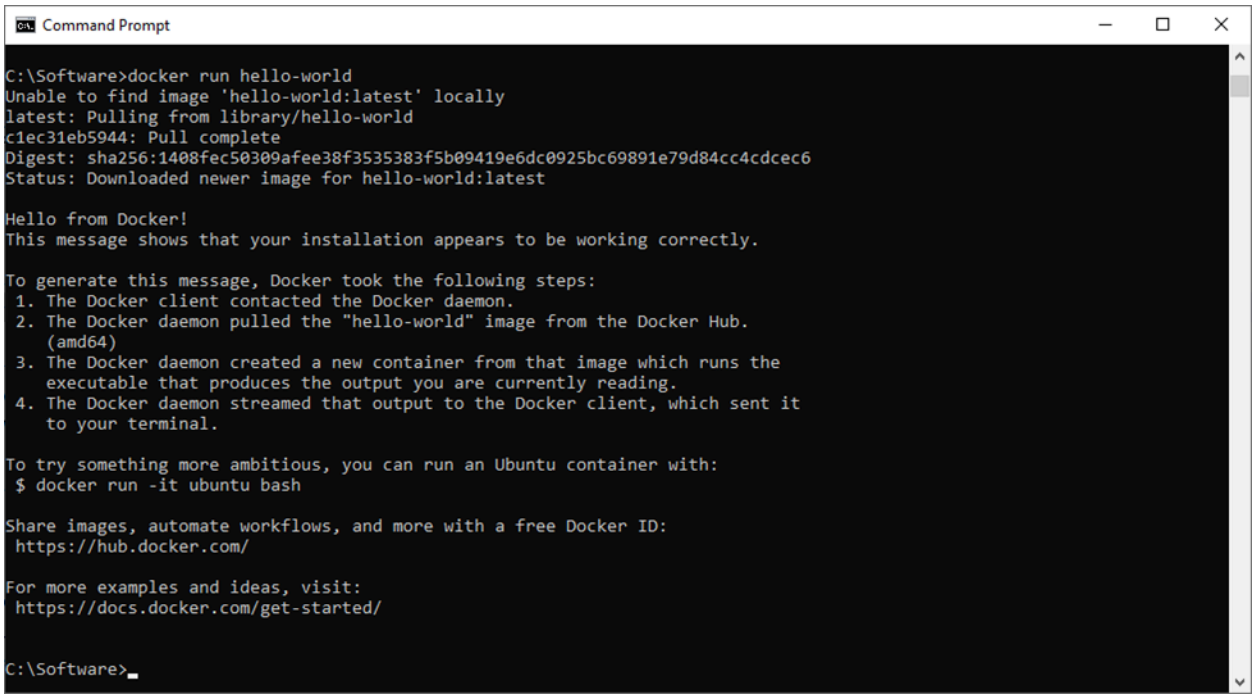
Στο τερματικό (console) εμφανίζονται πληροφορίες της εγκατάστασης.



Επιπλέον, για να ελέγξουμε ότι η εγκατάσταση έγινε σωστά, τρέχουμε ένα δοκιμαστικό container με την ακόλουθη εντολή:

```
> docker run hello-world
```

Στο τερματικό (console) εμφανίζονται το μήνυμα αδυναμίας εύρεσης του συγκεκριμένου container, βρίσκει και κατεβάζει από το Docker Hub το επιθυμητό container και το εκτελεί με επιτυχία.



6.3 Δημιουργία project για συλλογή και αποθήκευση δεδομένων καιρού.

6.3.1 Επιλογή Weather API για την ανάκτηση δεδομένων καιρού

Η εφαρμογή επεξεργάζεται δεδομένα καιρικών συνθηκών που λαμβάνει από την υπηρεσία ιστού (web service) της ιστοσελίδας wttr.in: https://wttr.in/ η οποία έχει άδεια χρήσης λογισμικού Apache License (v2.0).

Το wttr.in είναι μια υπηρεσία πρόγνωσης καιρού προσανατολισμένη στην κονσόλα που υποστηρίζει διάφορες μεθόδους αναπαράστασης πληροφοριών, όπως ακολουθίες ANSI προσανατολισμένες σε τερματικά για πελάτες HTTP κονσόλας (curl, httpie ή wget), HTML για προγράμματα περιήγησης ιστού ή PNG για προγράμματα προβολής γραφικών. (Apache-2.0 license)

Για τη χρήση των web services από το https://wttr.in δεν απαιτείται η δημιουργία λογαριασμού ή/και API KEY (σε αντίθεση με άλλα συστήματα όπως το OpenWeather, κτλ).

Λαμβάνουμε και αποθηκεύουμε δεδομένα για τις παρακάτω πόλεις:

- Αγρίνιο
- Αθήνα
- Αλεξανδρούπολη
- Άργος
- Βόλος
- Πύργος
- Καρδίτσα
- Τρίκαλα
- Θεσσαλονίκη
- Λάρισα
- Πάτρα
- Κόρινθος
- Χαλκίδα
- Χανιά

6.3.2 Δημιουργία scheduler για ανάκτηση προγνωστικών δεδομένων καιρού

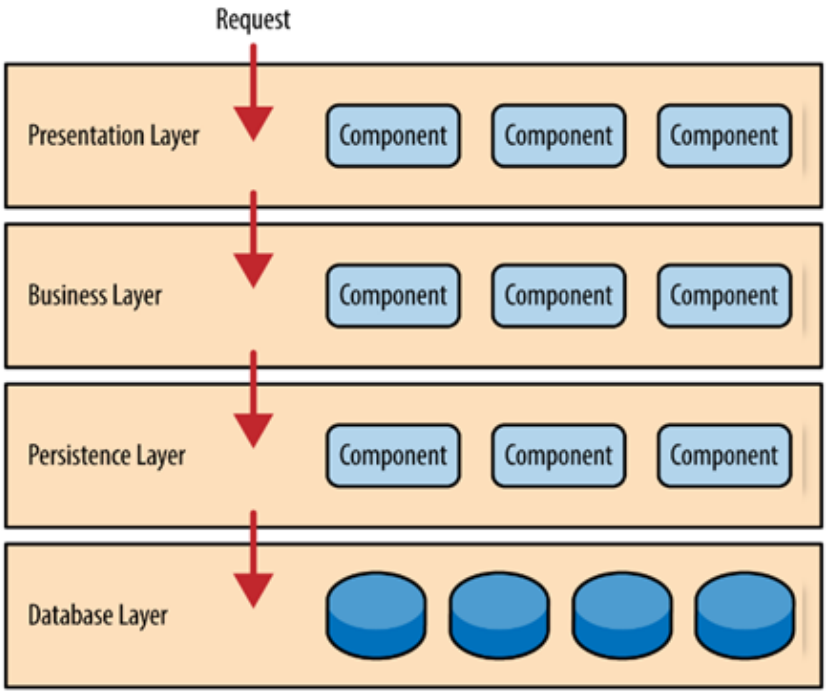
Δημιουργήθηκε το Project **telemetry-scheduler** το οποίο εκτελείται κάθε ώρα και αντλεί προγνωστικά δεδομένα καιρού από μια λίστα από πόλεις που έχουν οριστεί. Για κάθε πόλη ελέγχει αν ήδη υπάρχουν καταχωρημένες μετρήσεις βάσει της «Τοπικής Ώρας Παρατήρησης» που έχει καταχωρηθεί. Σε περίπτωση που έρθει νέα μέτρηση, τότε καταχωρείται στην βάση, εναλλακτικά αγνοείται.

Τεχνολογίες που χρησιμοποιήθηκαν:

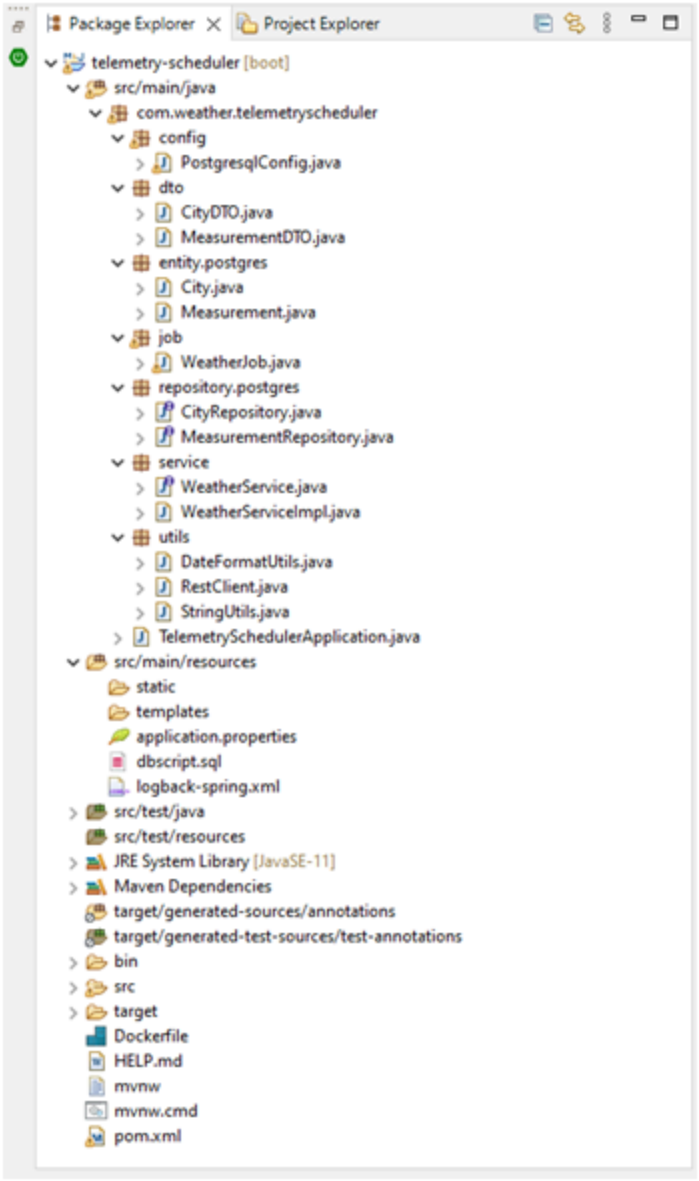
- Java – Spring Boot Framework
- Quartz scheduler
- Rest API
- GSON
- Hibernate
- Maven
- Lombok
- Logback
- Dockerfile

6.3.3 Δομή Java Scheduler project

Η ανάπτυξη του του scheduler project δημιουργήθηκε σε Java και Spring Boot Framework βασισμένοι στην δομή όπως ορίζουν τα Java projects ακολουθώντας την layered αρχιτεκτονική.



Ακολουθώντας την layered αρχιτεκτονική (όπως παρουσιάζεται στο παραπάνω σχήμα), δημιουργήθηκε η ακόλουθη δενδροειδής δομή (tree structure) του project το οποίο καλύπτει κάθε layer.


Package Explorer
Project Explorer
telemetry-scheduler [boot]
src/main/java
com.weather.telemetryscheduler
config
PostgresqlConfig.java
dto
CityDTO.java
MeasurementDTO.java
entity.postgres
City.java
Measurement.java
job
WeatherJob.java
repository.postgres
CityRepository.java
MeasurementRepository.java
service
WeatherService.java
WeatherServiceImpl.java
utils
DateFormatUtils.java
RestClient.java
StringUtils.java
TelemetrySchedulerApplication.java
src/main/resources
static
templates
application.properties
dbscript.sql
logback-spring.xml
src/test/java
src/test/resources
JRE System Library [JavaSE-11]
Maven Dependencies
target/generated-sources/annotations
target/generated-test-sources/test-annotations
bin
src
target
Dockerfile
HELP.md
mvnw
mvnw.cmd
pom.xml

Η κεντρική κλάση της εφαρμογής είναι η WeatherJob η οποία υλοποιεί τον Quartz scheduler. Εχει οριστεί να εκτελείται κάθε ώρα και κάνει ανάκτηση δεδομένων μέσα από μια λίστα από συγκεκριμένες πόλεις που έχουν οριστεί. Συγκεκριμένα, για κάθε πόλη ελέγχει αν υπάρχουν νεότερα δεδομένα μετρήσεων τα οποία δεν έχουν καταγραφεί στην βάση δεδομένων.

WeatherJob.java

Δημιουργούμε τη βοηθητική κλάση RestClient η οποία διαχειρίζεται τις Rest κλήσεις και κάνει χρήση του Rest API το οποίο προσφέρει ο weather server. Συγκεκριμένα, εκτελεί μια REST κλήση για κάποια πόλη, παίρνει σαν απάντηση δεδομένα σε μορφή JSON και περνάει τα δεδομένα που μας ενδιαφέρουν στη κλάση DTO (Data Transfer Object class).

RestClient.java

Δημιουργούμε το Business ή Service layer της αρχιτεκτονικής μας.

WeatherService.java
WeatherServiceImpl.java

Δημιουργούμε το Persistence Layer της αρχιτεκτονικής μας το οποίο χρησιμοποιείται για την αποθήκευση δεδομένων στην βάση δεδομένων. Αποθηκεύονται δεδομένα για την πόλη (City) και οι μετρήσεις αυτής.

CityRepository.java
MeasurementRepository.java

Οι παρακάτω κλάσεις City και Measurement αποτελούν Entities, κλάσεις οι οποίες αντιστοιχούν στους πίνακες CITY και MEASUREMENT (που δημιουργήθηκαν στην PostgreSQL) σε Java classes. Κάθε στήλη του πίνακα ισοδυναμεί με ένα πεδίο στην αντίστοιχη κλάση.

City.java
Measurement.java

Βοηθητικές DTO (ή **Data Transfer Objects**) κλάσεις είναι αντικείμενα που μεταφέρουν δεδομένα μεταξύ διεργασιών προκειμένου να μειωθεί ο αριθμός των κλήσεων μεθόδων.

Σε αυτές τις κλάσεις κάνουμε χρήση της βιβλιοθήκης **Lombok**. Η Lombok είναι μια δημοφιλής βιβλιοθήκη Java που απλοποιεί τη διαδικασία ανάπτυξης κώδικα μειώνοντας την ποσότητα του κώδικα boilerplate που χρειάζεται να γραφεί. Αυτό το επιτυγχάνει δημιουργώντας αυτόματα κοινές κατασκευές κώδικα Java, όπως κατασκευάστριες μεθόδους (constructor methods), μεθόδους getter και setter, κ.α, χρησιμοποιώντας annotations. Ο πρωταρχικός στόχος της Lombok είναι να αυξήσει την αναγνωσιμότητα του κώδικα και να μειώσει την πολυγλωσσία του χωρίς να θυσιάζει τη δυνατότητα συντήρησης.

CityDTO.java
MeasurementDTO.java

Δημιουργούμε την κλάση PostgresqlConfig που αποτελεί το Database Configuration file το οποίο θα ενώσει το project μας με τη βάση δεδομένων και θα εκτελούνται τα SQL queries. Ουσιαστικά εξασφαλίζεται η επικοινωνία του Java project με την PostgreSQL.

PostgresqlConfig.java

Δημιουργούμε το Property file (application.properties) το οποίο περιέχει παραμετροποιήσεις της εφαρμογής μέσω παραμέτρων για την αλλαγή της συμπεριφοράς της. Στην τυπική περίπτωση, μόνο οι διαχειριστές συστήματος και/ή οι διαχειριστές εφαρμογών μπορούν να διαβάζουν και να γράφουν ιδιότητες εφαρμογής.

Στο παρακάτω αρχείο, μεταξύ άλλων, ορίζονται η περίοδος εκτέλεσης ανάκτησης δεδομένων από τον Quartz Scheduler καθώς και στοιχεία σύνδεσης της εφαρμογής με τη βάση δεδομένων.

application.properties

Αξίζει να σημειωθεί ότι σημαντικό στοιχείο του project είναι και το pom.xml. Το Project Object Model (ή POM όπως αλλιώς αναφερόμαστε σε αυτό) είναι ένα κρίσιμο στοιχείο του Maven και χρησιμοποιείται για τον καθορισμό της διαμόρφωσης και των εξαρτήσεων ενός έργου Maven.

Το Maven είναι ένα εργαλείο αυτοματισμού κατασκευής (build automation tool) που χρησιμοποιείται για τη διαχείριση και την κατασκευή έργων Java.

6.4 Σχεδιασμός βάσης δεδομένων PostgreSQL και δημιουργία πινάκων (database tables) για αποθήκευση δεδομένων.

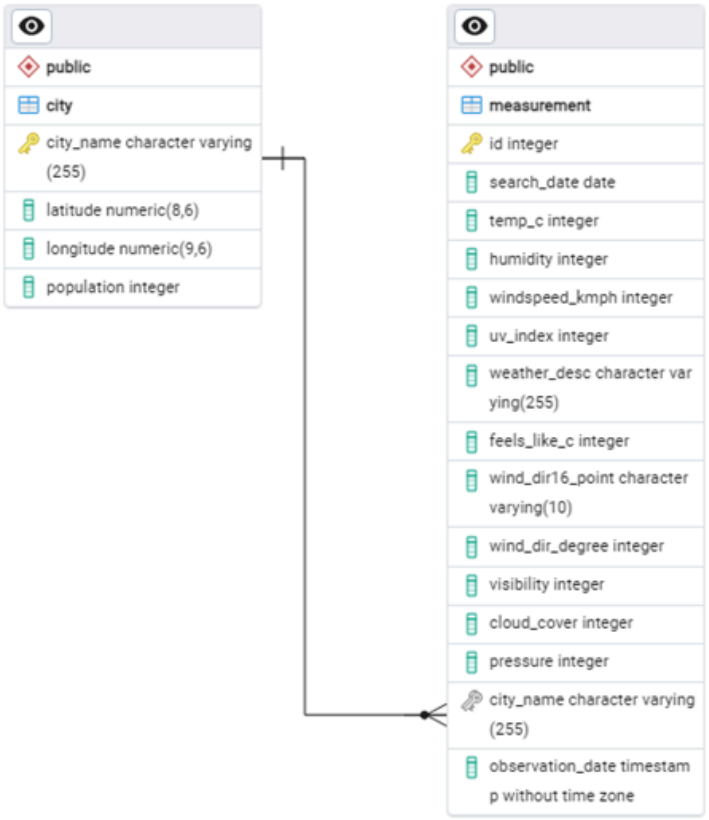
Τα δεδομένα που αποθηκεύονται για κάθε πόλη είναι:

Περιγραφή Δεδομένων	Όνομα πεδίου (Rest κλήσης)
Όνομα Πόλης	city_name
Γεωγραφικό πλάτος	latitude
Γεωγραφικό μήκος	longitude
Πληθυσμός	population

Τα δεδομένα καιρού που αποθηκεύονται για κάθε πόλη είναι:

Περιγραφή Δεδομένων	Όνομα πεδίου (Rest κλήσης)
Τοπική Ώρα Παρατήρησης	localObsDateTime
Θερμοκρασία (°C)	temp_c
Αισθητή Θερμοκρασία (°C)	FeelsLikeC
Υγρασία	humidity
Ταχύτητα ανέμου (km/h)	windspeedKmph
Διεύθυνση Ανέμου	winddir16Point
Βαθμός Διεύθυνσης Ανέμου	winddirDegree
Δείκτης UV	uvIndex
Λεκτική πρόγνωση καιρού	weatherDesc
Ορατότητα (km)	visibility
Νεφοκάλυψη	cloudcover
Πίεση	pressure

Για τα παραπάνω δεδομένα προκύπτει το ακόλουθο διάγραμμα οντοτήτων συσχετίσεων (ER Diagram) της βάσης δεδομένων.



Ορίζουμε στους παραπάνω πίνακες τη συσχέτιση “ένα-προς-πολλά” (one-to-many) καθώς επιθυμούμε για μια Πόλη (City) να έχουμε πολλές μετρήσεις (Measurement).

Δημιουργούμε τα αντίστοιχα SQL scripts για τη δημιουργία των πινάκων στη βάση δεδομένων βασισμένοι στο παραπάνω διάγραμμα οντοτήτων συσχετίσεων (ER Diagram) της βάσης δεδομένων.

```
CREATE TABLE CITY
(
        city_name  character varying(255) NOT NULL,
        latitude   numeric(8, 6)      NOT NULL,
        longitude  numeric(9, 6)              NOT NULL,
        population integer            NOT NULL,
        PRIMARY KEY (city_name)
);

CREATE TABLE MEASUREMENT
(
        id          INTEGER GENERATED BY DEFAULT AS IDENTITY,
        search_date         DATE                NOT NULL,
        observation_date timestamp without time zone NOT NULL,
        temp_c              integer             NOT NULL,
        humidity            integer             NOT NULL,
        windspeed_kmph   integer            NOT NULL,
        wind_dir16_point character varying(10)  NOT NULL,
        wind_dir_degree  integer            NOT NULL,
        uv_index            integer             NOT NULL,
        feels_like_c        integer             NOT NULL,
        visibility          integer             NOT NULL,
        cloud_cover         integer             NOT NULL,
        pressure            integer             NOT NULL,
        weather_desc        character varying(255) NOT NULL,
        city_name           character varying(255) NOT NULL,
        PRIMARY KEY (id)
);

ALTER TABLE IF EXISTS public.MEASUREMENT
        ADD FOREIGN KEY (city_name)
    REFERENCES public.city (city_name) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION NOT VALID;
```

6.5. Δημιουργία Docker container για το Java Project.

Για τη δημιουργία του Docker image επιλέγουμε την εγκαταστήσει του λειτουργικού σύστημα Linux Alpine. Το Alpine Linux είναι μια δημοφιλής επιλογή λειτουργικού συστήματος Linux για την εκτέλεση περιεκτών. Έχει σχεδιαστεί να είναι μικρή σε μέγεθος (περίπου 5,5 MB) και ασφαλής (secure).

Περιγραφή αρχείου Dockerfile και δημιουργία περιέκτη:

Το αρχείο "Dockerfile" είναι ένα απλό αρχείο οδηγιών, στο οποίο γράφουμε σε κάθε γραμμή τις εντολές που θέλουμε να εκτελεστούν με τη σειρά. Δημιουργεί τον περιέκτη (container) της Java εφαρμογής λαμβάνοντας από το Docker Hub την εικόνα (image) του OpenJDK έκδοσης 17, εγκαθιστά το Java OpenJDK στον περιέκτη και αντιγράφει στον περιέκτη τα complied αρχεία (.jar) του πηγαίου κώδικα.

Επομένως πρώτα γίνεται compile το Java project χρησιμοποιώντας την γραμμή εντολών του Maven (στο root φάκελο του project).

```
> mvn clean package install
```

Με την παραπάνω εντολή πραγματοποιούνται οι παρακάτω ενέργειες:

- clean: διαγράφει το περιεχόμενο του φακέλου /target
- package: μετατρέπει τον πηγαίο κώδικα .java σε αρχείο .jar (όπως έχει οριστεί στο pom.xml) και τον τοποθετεί στον φάκελο /target.
- install: Πρώτα δημιουργεί ένα πακέτο και στη συνέχεια, παίρνει αυτό το αρχείο .jar και το τοποθετεί στο τοπικό σας αποθετήριο Maven, το οποίο βρίσκεται στο ~/.m2/repository.

Επίσης στον root φάκελο του project, βρίσκεται το Dockerfile μέσω του οποίου θα δημιουργήσουμε το image που θα εκτελεστεί στο Docker.

Αρχείο Dockerfile

```
FROM openjdk:17-jdk-alpine
ARG JAR_FILE=target/telemetry-scheduler.jar
COPY ./target/telemetry-scheduler.jar app.jar

ENTRYPOINT ["java", "-jar", "/app.jar"]
```

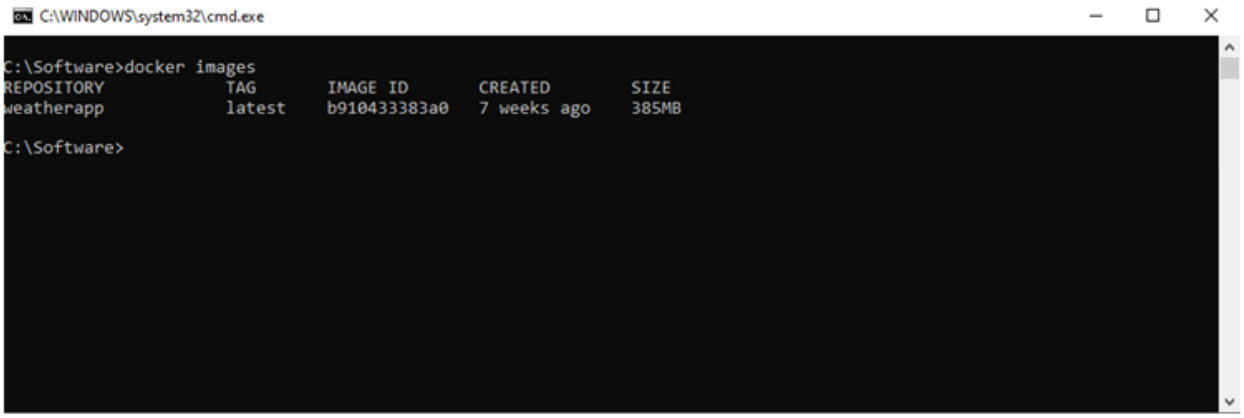
Εκτελούμε την παρακάτω εντολή για να δημιουργηθεί το Docker image:

```
> docker build -t weatherapp .
```

Μέσω της επιλογής παραμέτρου –t (ή και –tag) προσθέτουμε την ετικέτα weatherapp στην εικόνα όταν θα δημιουργηθεί. Η χρήση της ετικέτας διευκολύνει την εργασία με τις εικόνες που δημιουργούνται καθώς γίνεται αναφορά στην εικόνα με ένα όνομα που εμείς ορίζουμε.

Επιβεβαιώνουμε ότι έχει δημιουργηθεί το image εκτελώντας την παρακάτω εντολή:

```
> docker images
```



Εκτελούμε την εντολή “docker run” η οποία εκτελεί μια εντολή σε ένα νέο container, τραβώντας την εικόνα (pull image) εάν χρειάζεται και εκκινεί τον περιέκτη (container).

```
> docker run weatherapp
```

6.6. Δημιουργία αποθηκευτικού χώρου (Docker Volume) για χρήση από την PostgreSQL.

Οι τόμοι Docker (Docker Volumes) είναι μια μέθοδος που προσφέρει το Docker για τη διασφάλιση της μονιμότητας των δεδομένων κατά την εργασία των περιεκτών (containers). Οι τόμοι Docker είναι συστήματα αρχείων που είναι προσαρτημένα σε περιέκτες για τη διατήρηση των δεδομένων που δημιουργούνται από αυτούς. Όταν σταματήσει ένας περιέκτης (container), όλα τα δεδομένα που έχουν δημιουργηθεί αλλά δεν έχουν αποθηκευτεί σε έναν τόμος δεδομένων (Data Volume), διαγράφονται μαζί με τον περιέκτη

Τα Docker Volumes δημιουργούνται και διαχειρίζονται από το Docker. Αποθηκεύονται ως μέρος του συστήματος αρχείων του κεντρικού υπολογιστή. Ένα Docker Volume μπορεί να δημιουργηθεί είτε χρησιμοποιώντας την εντολή `docker volume create` είτε κατά τη δημιουργία ενός container μέσα από τη γραμμή εντολών.

Δημιουργούμε έναν τόμο δεδομένων (κάτι σαν αποθηκευτικό μέσον), έναν κατάλογο στο λειτουργικό σύστημα το οποίο είναι προσαρτημένο σαν συσκευή αποθήκευσης στο Docker Container.

Δημιουργούμε τον ακόλουθο φάκελο στον θα αποθηκεύσουμε τα αρχεία της βάσης δεδομένων τα οποία θα δημιουργηθούν κατά τη δημιουργία του Postgres image.

C:\dev\docker\pgdev

Με αυτόν τον τρόπο θα έχουμε διαθέσιμο το database schema και τα δεδομένα που θα έχουμε αποθηκεύσει κάθε φορά που θα χρειαστεί να κάνουμε restart το image (ή να κλείσουμε τον υπολογιστή που κάνουμε το development).

6.7. Δημιουργία Docker container για την PostgreSQL.

Δημιουργούμε το Docker image για την Postgres (έχοντας πρώτα δημιουργήσει Docker Volume).

```
> docker run -d --name project-pg -p 5432:5432 -e
POSTGRES_PASSWORD=adminadmin -v
C:\dev\docker\pgdev:/var/lib/postgresql/data postgres
```

Η ερμηνεία των παραπάνω εντολών είναι η ακόλουθη:

- -d: ενεργοποιούμε το Docker να εκτελεί τον περιέκτη στο παρασκήνιο (background process)
- --name: ορίζουμε το όνομα του περιέκτη που θα δημιουργηθεί (κάτι που μας βοηθά να το αναγνωρίζουμε και να αναφερόμαστε σε αυτό).
- -p και ακολουθούν οι αριθμοί θύρας (port numbers) επιτρέπει να αντιστοιχίσουμε τη θύρα του περιέκτη 5432 στην εξωτερική θύρα 5432. Με αυτόν τον τρόπο επιτρέπεται η σύνδεση σε αυτήν από το εξωτερικές εφαρμογές
- -e POSTGRES_PASSWORD: ορίζει τον κωδικό πρόσβασης στο docker. Αυτός είναι ο κωδικός πρόσβασης στη βάση δεδομένων
- -v ορίζει το Docker Volume που θα χρησιμοποιήσουμε, συγκεκριμένα ορίζουμε τον φάκελο C:\dev\docker\pgdev να αντιστοιχεί στον φάκελο /var/lib/postgresql/data του περιέκτη που θα περιέχει τα αρχεία της βάσης δεδομένων.
- Το τελευταίο τμήμα της εντολής (postgres) παίρνει την πιο πρόσφατη εικόνα της Postgres από το Docker Hub.

Εναλλακτικός τρόπος είναι να κατεβάσουμε πρώτα την τελευταία επίσημη εικόνα της Postgres από το Docker Hub μέσω της εντολής:

```
> docker pull postgres
```

Και έπειτα να ξεκινήσουμε την Postgres μέσω της γραμμής εντολών (docker run).

Σε περίπτωση που επιθυμούμε να ξεκινήσουμε τον περιέκτη ο οποίος έχει σταματήσει, χρησιμοποιούμε την εντολή:

```
> docker start project-pg
```

Πλέον η βάση δεδομένων έχει εγκατασταθεί και είναι έτοιμη για χρήση. Μπορούμε να συνδεθούμε με κάποιο εργαλείο που διαθέτουμε (όπως pgAdmin ή DBeaver) χρησιμοποιώντας τις ακόλουθες λεπτομέρειες σύνδεσης.

Στοιχεία Σύνδεσης:

- Host Name: localhost
- Port: 5432
- Username: postgres
- Password: adminadmin

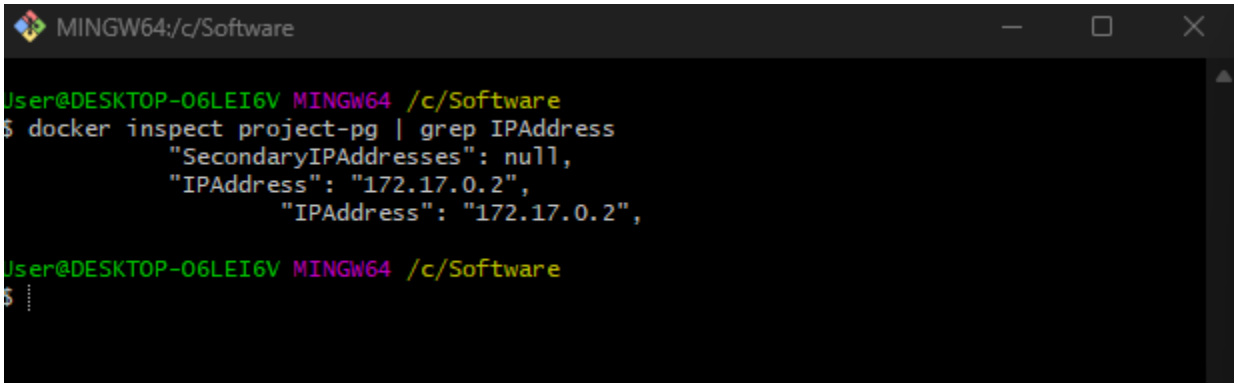
Μπορούμε να συνδεθούμε και απευθείας στην βάση δεδομένων χρησιμοποιώντας την ακόλουθη γραμμή μέσω της εντολής psql.

```
> psql -h localhost -U postgres
```

Για να βρούμε την διεύθυνση (IP Address) της Postgres που τρέχει πλέον μέσα στο Docker χρησιμοποιούμε την εντολή:

```
> docker inspect project-pg | grep IPAddress
```

Εκτελούμε την παραπάνω εντολή και βλέπουμε την IP την οποίοι μπορούμε να χρησιμοποιήσουμε και είναι η: 172.17.0.2.



6.8. Δημιουργία Docker container για το Grafana (για ανάλυση και παρακολούθηση δεδομένων) και σύνδεση με τη βάση δεδομένων.

Για την οπτικοποίηση των μετρήσεων που έχουμε συλλέξει από συγκεκριμένες πόλεις μέσω του Java Scheduler, θα χρησιμοποιήσουμε το Grafana.

Η Grafana είναι μια πλατφόρμα παρατηρησιμότητας για οπτικοποίηση μετρήσεων. Επιτρέπει τη δημιουργία ειδικά προσαρμοσμένων πινάκων εργαλείων για την εμφάνιση ουσιαστικών πληροφοριών από τις ροές δεδομένων σε πραγματικό χρόνο. Το λογισμικό ανοιχτού κώδικα Grafana δίνει τη δυνατότητα (μεταξύ πολλών λειτουργιών που προσφέρει) να πραγματοποιηθούν ερωτήσεις, να γίνει γραφική απεικόνιση αυτών και να δημιουργηθούν ειδοποιήσεις.

Δημιουργούμε το Docker image για το Grafana μέσω της ακόλουθης γραμμής εντολών:

```
> docker run -d --name grafana -p 3000:3000 grafana/grafana-oss
```

Συγκεκριμένα ο περιέκτης που θα δημιουργηθεί θα περιέχεις την έκδοση Grafana Open Source: *Grafana v10.4.3 (0bfd547800)*.

Η ερμηνεία των παραπάνω εντολών είναι η ακόλουθη:

- -d: ενεργοποιούμε το Docker να εκτελεί τον περιέκτη στο παρασκήνιο (background process)
- --name: ορίζουμε το όνομα του περιέκτη που θα δημιουργηθεί (κάτι που μας βοηθά να το αναγνωρίζουμε και να αναφερόμαστε σε αυτό).
- -p και ακολουθούν οι αριθμοί θύρας (port numbers) επιτρέπει να αντιστοιχίσουμε τη θύρα του περιέκτη 3000 στην εξωτερική θύρα 3000. Με αυτόν τον τρόπο επιτρέπεται η σύνδεση σε αυτήν από το εξωτερικές εφαρμογές
- Το τελευταίο τμήμα της εντολής (grafana/grafana-oss) παίρνει την πιο πρόσφατη εικόνα του Grafana από το Docker Hub.

Έχοντας εκτελέσει την παραπάνω εντολή είμαστε έτοιμοι να συνδεθούμε στο Grafana στην πόρτα 3000 χρησιμοποιώντας το παρακάτω URL:

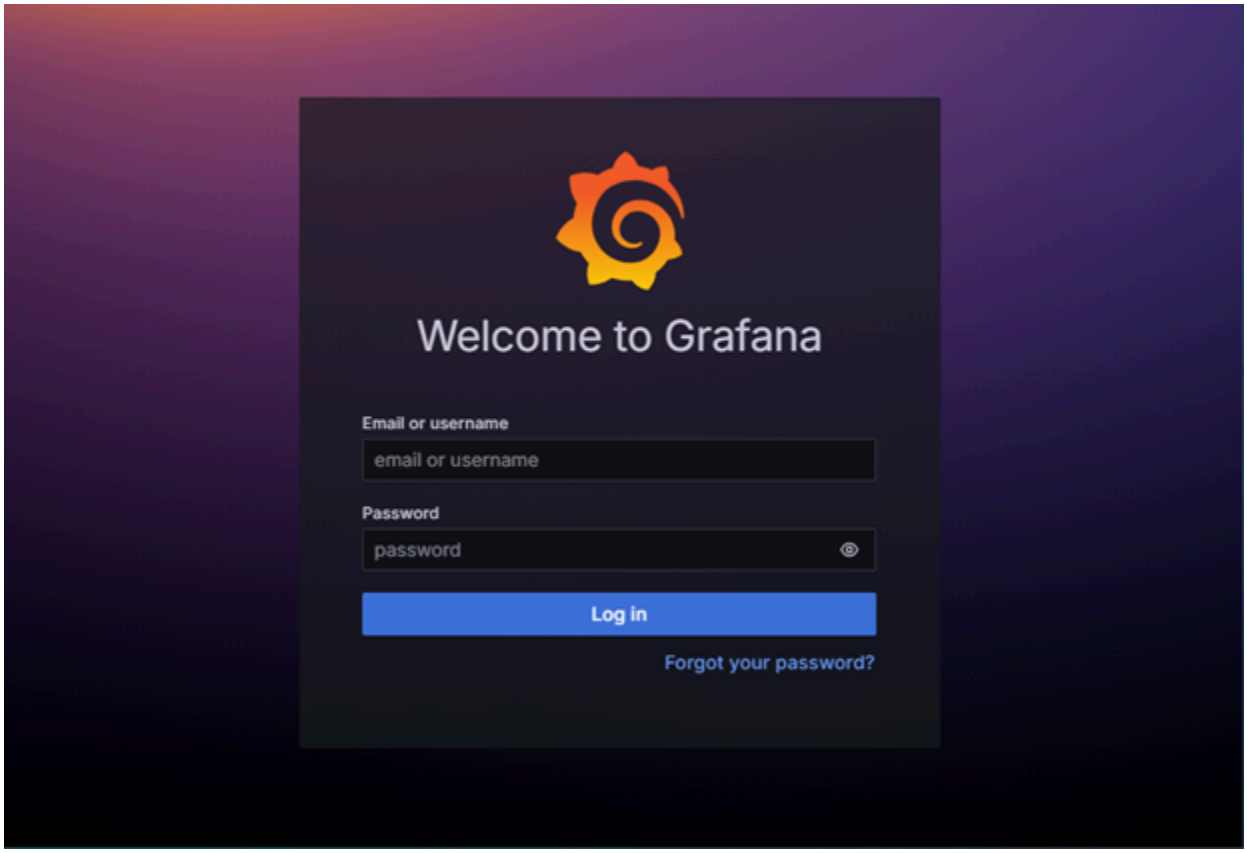
http://localhost:3000

Κατά την εγκατάσταση του έχει προκαθοριστεί το όνομα χρήστη και ο κωδικός τα οποία είναι:

Username: admin

Password: admin

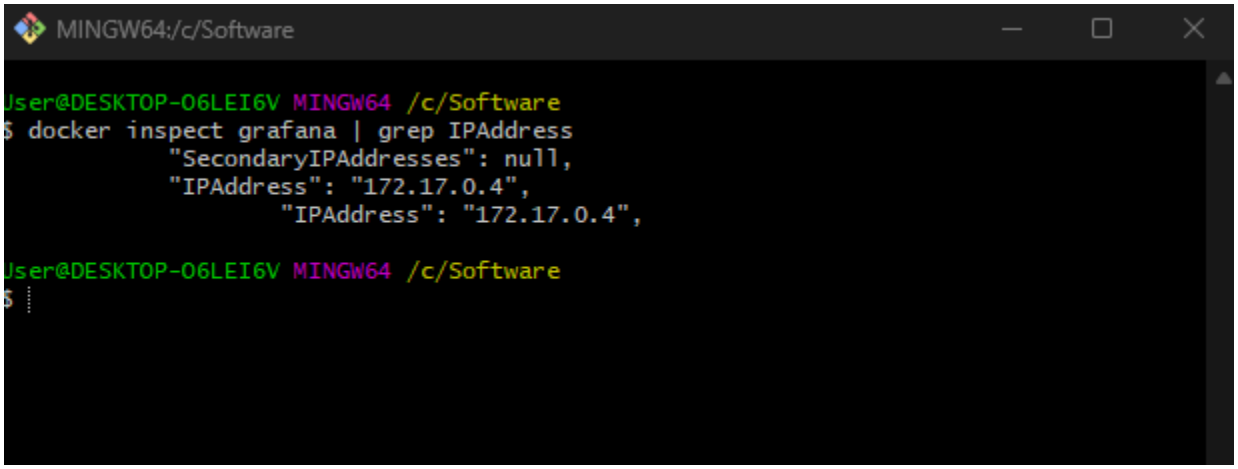
Μετά την πρώτη είσοδο στην εφαρμογή, ζητείται να ορίσουμε νέο κωδικό εισόδου για τον χρήστη admin.



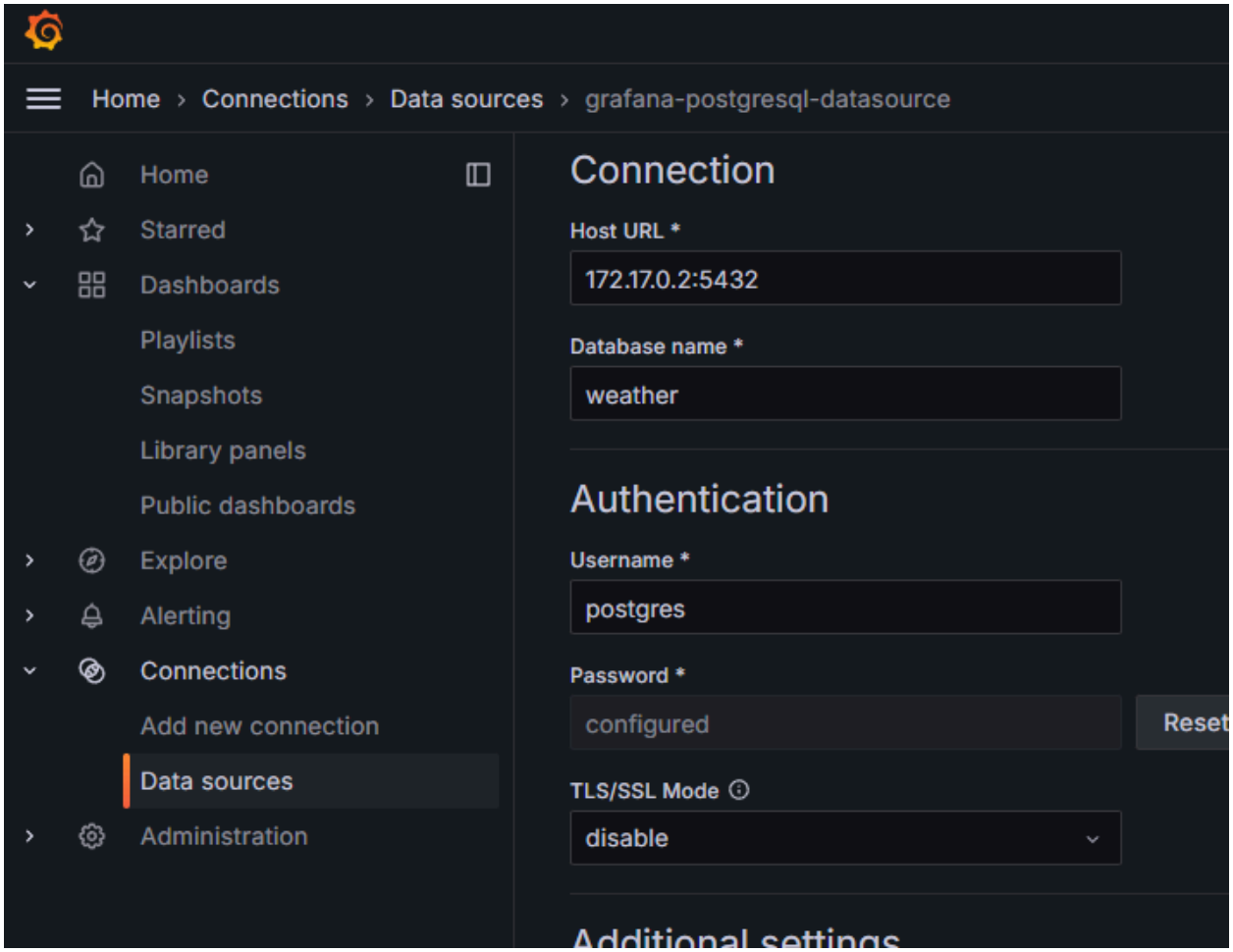
Για να βρούμε την διεύθυνση (IP Address) του Grafana που τρέχει στο Docker χρησιμοποιούμε την εντολή:

```
> docker inspect grafana | grep IPAddress
```

Εκτελούμε την παραπάνω εντολή και βλέπουμε την IP την οποίοι μπορούμε να χρησιμοποιήσουμε και είναι η: 172.17.0.4.



Κατά την είσοδο στο Grafana, το πρώτο πράγμα που πρέπει να κάνουμε είναι να το συνδέσουμε με το docker της Postgres που έχει ήδη δημιουργηθεί και τρέχει. Από το κεντρικό Menu επιλέγουμε " Add new connection" και ορίζουμε τα στοιχεία της σύνδεσης:


Home › Connections › Data sources › grafana-postgresql-datasource
Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Public dashboards
Explore
Alerting
Connections
Add new connection
Data sources
Administration
Connection
Host URL *
172.17.0.2:5432
Database name *
weather
Authentication
Username *
postgres
Password *
configured
Reset
TLS/SSL Mode
disable
Additional settings

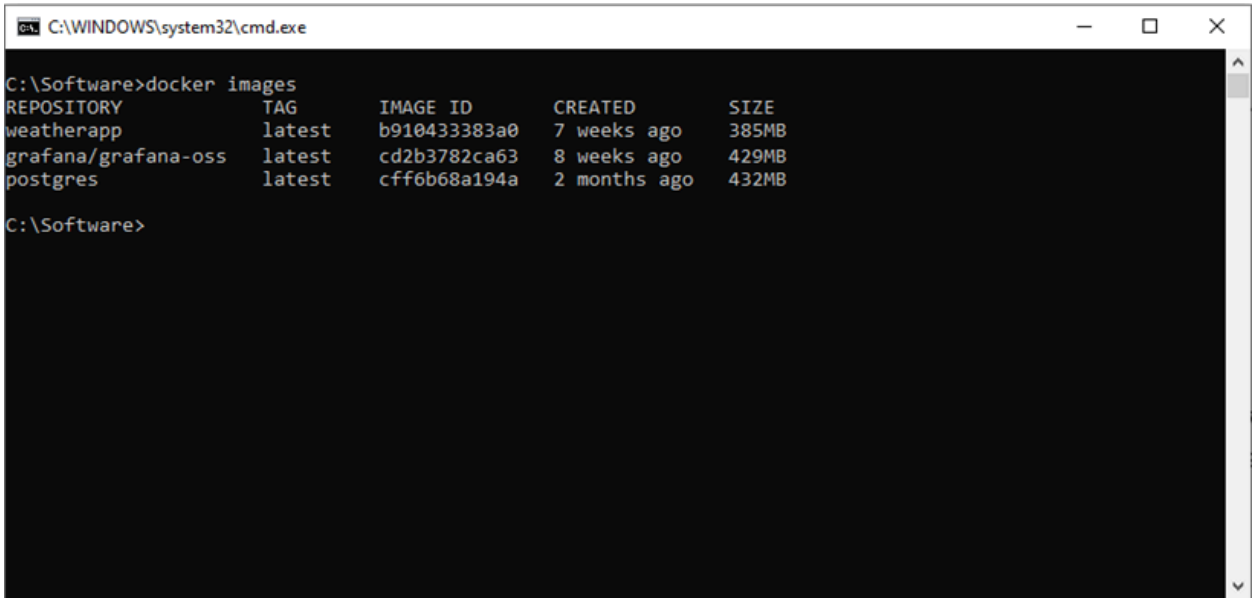
Με την επιτυχή σύνδεση στην Postgres δημιουργούμε νέο Dashboard (με όνομα Weather Forecast Dashboard) με τα δεδομένα που είναι αποθηκευμένα μέσω της συλλογής δεδομένων από τον scheduler και αποθηκεύονται στην βάση δεδομένων. Ενδεικτικά εμφανίζουμε δεδομένα από Αθήνα, Θεσσαλονίκη, Πύργο και Αλεξανδρούπολη.

6.9. Συνολικός έλεγχος της λύσης (end-to-end testing) για να βεβαιωθούμε ότι όλα τα containers επικοινωνούν σωστά μεταξύ τους.

Για να βεβαιωθούμε ότι όλα τα Docker Images είναι διαθέσιμα, εκτελούμε την παρακάτω εντολή η οποία μας εμφανίζει μια λίστα με τα διαθέσιμα images:

```
> docker images
```

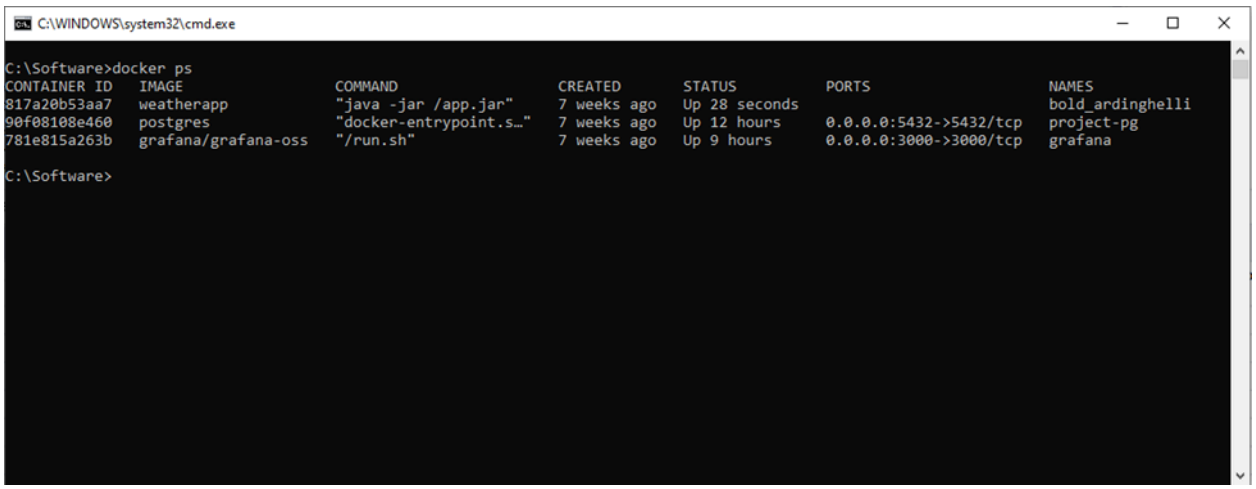
Εκτελούμε την παραπάνω εντολή και προκύπτει η ακόλουθη εικόνα:



Για να δούμε τα διαθέσιμα Docker Containers που τρέχουν, εκτελούμε την παρακάτω εντολή:

```
> docker ps
```

Εκτελούμε την παραπάνω εντολή και προκύπτει η ακόλουθη εικόνα:



6.10. Ο Ενορχηστρωτής Docker Swarm

Όπως έχει αναφερθεί σε προηγούμενο κεφάλαιο, το Docker Swarm είναι εργαλείο ενορχήστρωσης και ομαδοποίησης container που βοηθά στη διαχείριση κεντρικών υπολογιστών Docker Containers και αποτελεί μέρος του Docker Engine.

Στόχος του Docker Swarm είναι να ομαδοποιήσει πολλούς κεντρικούς Docker υπολογιστές σε έναν ενιαίο λογικό εικονικό διακομιστή. Το Docker Swarm παρέχει αρκετά πλεονεκτήματα για την εκτέλεση φόρτου εργασίας υπηρεσιών. Σημαντικά οφέλη που προσφέρει είναι Μεγάλη Διαθεσιμότητα (High Availability), Αποκεντρωμένο σχεδιασμό (Decentralized design), Εξισορρόπηση φορτίου (Load Balancing), Κλιμάκωση (Scaling) και άλλα.

6.10.1 Πως δουλεύει το Docker Swarm

Υπάρχουν δύο τύποι κόμβων στο Docker Swarm:

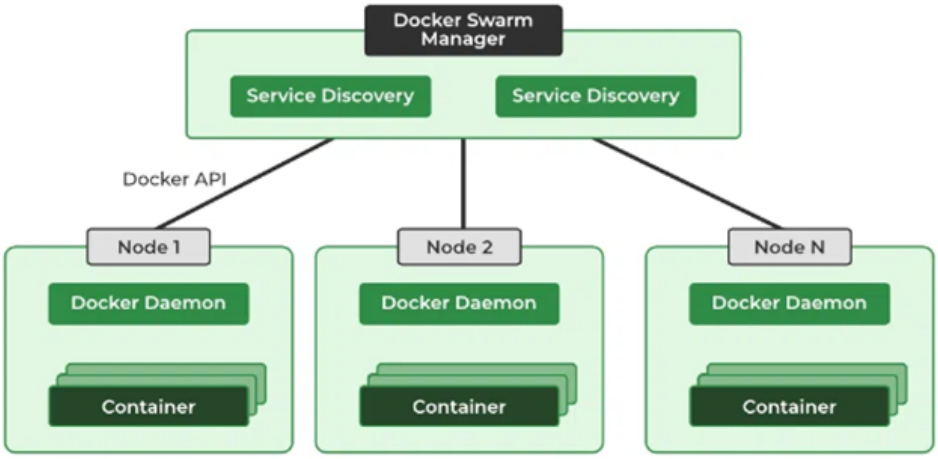
- Ο Κόμβος Διαχειριστή: (Manager node) ο οποίος εκτελεί και επιβλέπει καθήκοντα σε επίπεδο συμπλέγματος (cluster).

- Ο Εργαζόμενος Κόμβος (Worker node) ο οποίος λαμβάνει και ολοκληρώνει τις εργασίες που ορίζονται από τον κόμβο διαχειριστή (manager node).

Για να αναπτυχθεί (deploy) ένα container στο σμήνος (swarm), πρέπει πρώτα να γίνει εκκίνηση των υπηρεσιών (services). Η υπηρεσία αποτελείται από πολλά containers της ίδιας εικόνας (container image). Αυτές οι υπηρεσίες αναπτύσσονται μέσα σε έναν κόμβο (node), επομένως για να αναπτυχθεί ένα σμήνος πρέπει να αναπτυχθεί τουλάχιστον ένας κόμβος.

Ο κόμβος διαχειριστή (manager node) είναι υπεύθυνος για την κατανομή της εργασίας, την αποστολή και τον χρονοπρογραμματισμό (scheduling) αυτών. Επιπλέον ο κόμβος διαχειριστή (manager node) επικοινωνεί με κάθε κόμβο εργαζομένου (worker node) χρησιμοποιώντας το πρωτόκολλο HTTP.

Όλη η εκτέλεση της εργασίας εκτελείται από τον κόμβο εργάτη (worker node).



Σύμφωνα με το παραπάνω διάγραμμα, ένας κόμβος διαχειριστή (manager node) μπορεί να δημιουργηθεί μεμονωμένα σε αντίθεση με τον κόμβο εργαζόμενο (worker node) ο οποίος δεν μπορεί να δημιουργηθεί χωρίς έναν κόμβο διαχειριστή. Ο ιδανικός αριθμός για την καταμέτρηση του κόμβου διαχειριστή είναι επτά. Η επιπλέον αύξηση του αριθμού του κόμβου διαχειριστή δεν σημαίνει ότι θα αυξηθεί η επεκτασιμότητα (scalability).

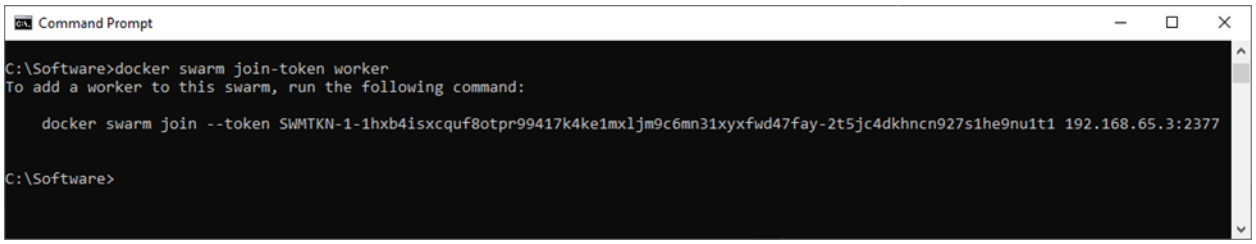
6.10.2 Δημιουργία Docker Swarm (manager) και εργάτες (worker nodes)

Ένα σμήνος (swarm) αποτελείται από πολλαπλούς Docker nodes που τρέχουν σε swarm mode και λειτουργούν ως managers ή/και εργάτες (workers). Ένα Docker node μπορεί να είναι manager, εργάτης ή και τα δύο.

Για να δημιουργηθεί περιβάλλον Docker Swarm πραγματοποιούνται τα παρακάτω βήματα:

- Εγκαθιστούμε το Docker Engine σε όλους τους κόμβους που θα αποτελούν μέρος του σμήνους (swarm).
- Αρχικοποιούμε το σμήνος στον κόμβο διαχειριστή (manager node) εκτελώντας την εντολή:
  `> docker swarm init`
    - Η παραπάνω εντολή κατά την δημιουργία του κόμβου διαχειριστή παράγει και ένα μοναδικό κλειδί το οποίο θα πρέπει να χρησιμοποιηθεί σε κάθε φορά που απαιτείται η δημιουργία νέου κόμβου εργαζομένου (worker node)

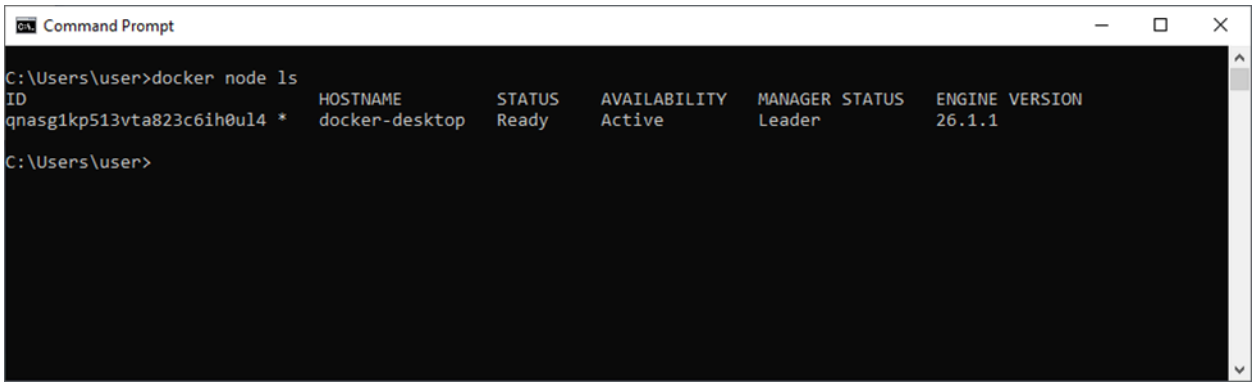
- Προσθέσουμε κόμβους εργαζομένων (worker node) στο σμήνος χρησιμοποιώντας την εντολή join που παρέχεται από τον κόμβο διαχειριστή
    - Για να ανακτήσουμε την εντολή join συμπεριλαμβανομένου του διακριτικού σύνδεσης (joint token) για κόμβους εργαζομένων, εκτελούμε την εντολή:
      `> docker swarm join-token worker`



Μετά την ενεργοποίηση του swarm (μέσω του swarm init), εμφανίζουμε τα διαθέσιμα swarm nodes:

```
> docker node ls
```

Παρατηρούμε ότι ο κόμβος (node) ο οποίος αρχικοποιήσαμε το swarm είναι και ο κόμβος διαχειριστής (manager node) .



Όπως αναφέραμε και παραπάνω, για να ενεργοποιήσουμε κόμβους εργαζομένων (worker nodes) θα πρέπει να συνδεθούμε σε κάθε έναν από αυτούς και να γίνει εγκατάσταση του Docker Engine. Δημιουργώντας ένα παράδειγμα σμήνους, δημιουργούμε δυο κόμβους εργαζομένων (worker nodes) και επιπλέον ορίζουμε το hostname σε node-work1 και node-work2 αντίστοιχα.

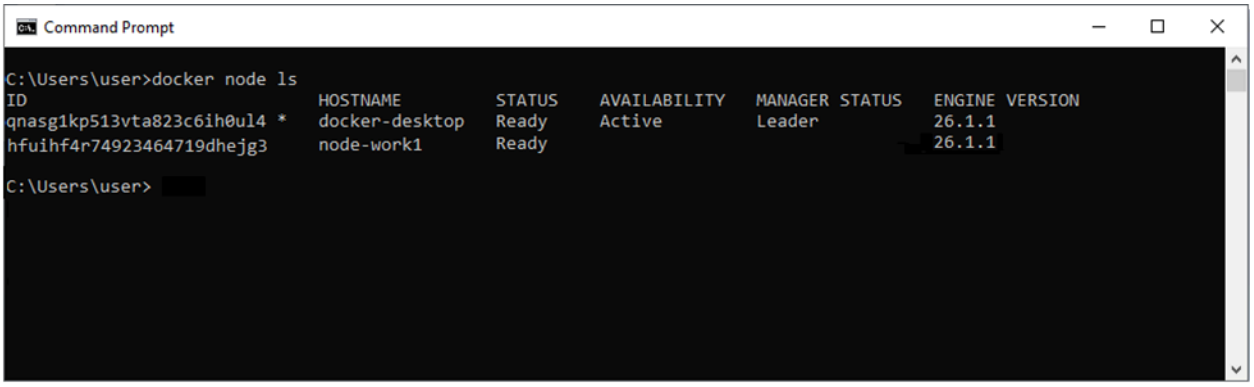
Εκτελούμε την ακόλουθη εντολή στον node-work1:

```
> docker swarm join --token
SWMTKN-1-1hxb4isxcquf8otpr99417k4ke1mxljm9c6mn31xyxfwd47fay-2t5jc4dkhn
cn927s1he9nu1t1 192.168.65.3:2377
```

Εκτελούμε ξανά την ακόλουθη εντολή να εμφανίζουμε τα διαθέσιμα swarm nodes:

```
> docker node ls
```

Παρατηρούμε ότι πλέον στη λίστα με τα εμφανίζουμε τα διαθέσιμα swarm nodes εμφανίζεται και το worker node: node-work1:



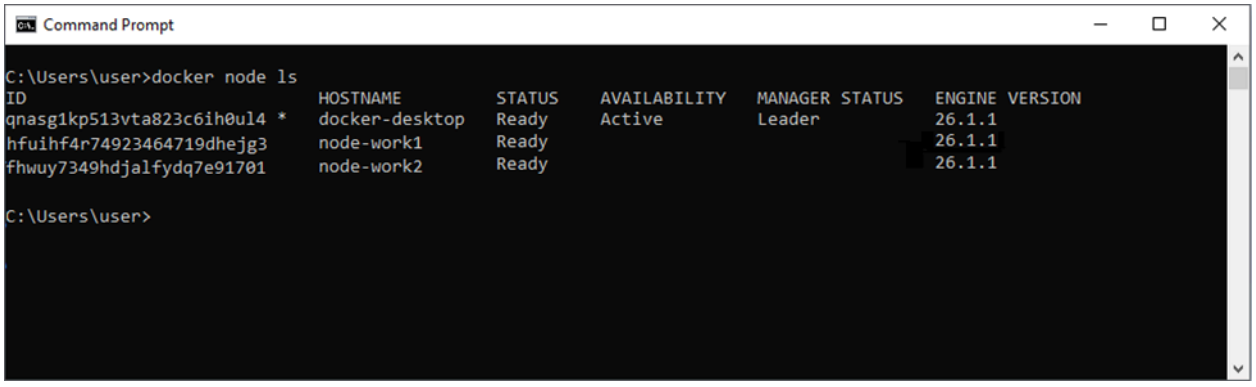
Εκτελούμε ξανά την ακόλουθη εντολή στον node-work2:

```
> docker swarm join --token
SWMTKN-1-1hxb4isxcquf8otpr99417k4ke1mxljm9c6mn31xyxfwd47fay-2t5jc4dkhn
cn927s1he9nu1t1 192.168.65.3:2377
```

Εμφανίζουμε τα διαθέσιμα swarm nodes:

```
> docker node ls
```

Παρατηρούμε ότι πλέον στη λίστα με τα εμφανίζουμε τα διαθέσιμα swarm nodes εμφανίζεται και το worker node: node-work2.



Για να διαγράψουμε κάποιον worker από το σμήνος, θα πρέπει να συνδεθούμε στο node το οποίο περιέχει τον worker και να εκτελέσουμε την εντολή:

```
> docker swarm leave
```

6.10.3 Docker Swarm scaling

Για να αναπτύξουμε (deploy) μια εικόνα εφαρμογής (application image) όταν το Docker Engine βρίσκεται σε λειτουργία Swarm, δημιουργείτε μια υπηρεσία (service).
Μόλις δημιουργηθεί μια υπηρεσία (service) σε ένα σμήνος (swarm), μέσω του Docker CLI (command line) μπορούμε να αυξήσουμε ή να μειώσουμε τον αριθμό των περιεκτών (containers) στην υπηρεσία (service). Οι περιέκτες που εκτελούνται σε μια υπηρεσία ονομάζονται εργασίες (tasks).

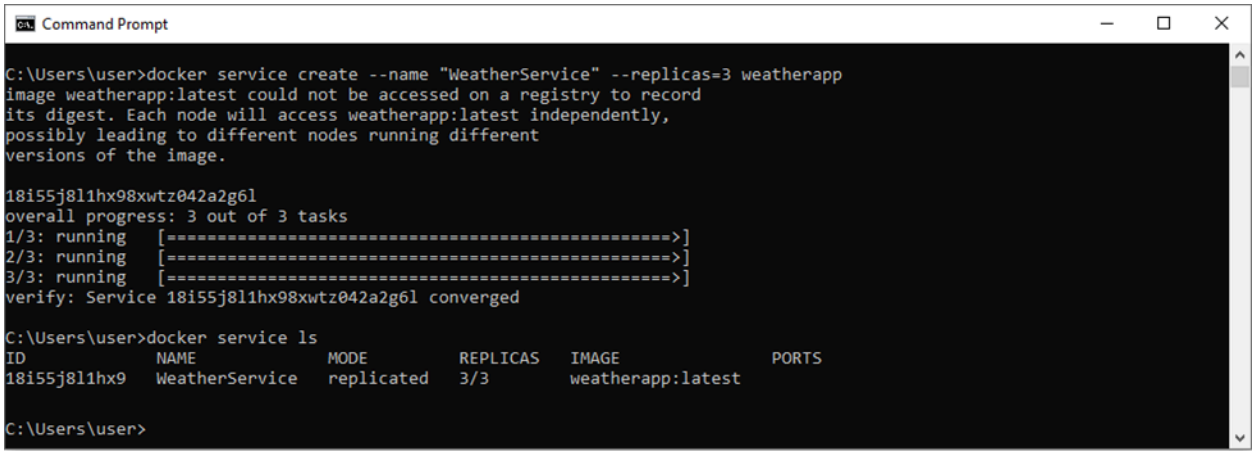
Αυτή η κλιμάκωση μίας ή περισσότερων από τις αναπαραγόμενες υπηρεσίες (είτε προς τα πάνω είτε προς τα κάτω) στον επιθυμητό αριθμό αντιγράφων χρησιμοποιούμε την εντολή scale (ορίζοντας το μέγεθος πχ σε 5):

```
> docker service scale <SERVICE NAME>=5
```

Η παραπάνω εντολή για παράδειγμα ορίζει τον αριθμό των αναπαραγόμενων υπηρεσιών σε 5.
Αυτή η εντολή δεν μπορεί να εφαρμοστεί σε υπηρεσίες που είναι καθολική λειτουργία (global mode). Η εντολή θα επιστρέψει αμέσως, αλλά η πραγματική κλιμάκωση της υπηρεσίας μπορεί να πάρει κάποιο χρόνο. Για να σταματήσουν όλα τα αντίγραφα μιας υπηρεσίας ενώ θα πρέπει διατηρείτε η υπηρεσία ενεργή στο σμήνος, θα πρέπει να οριστεί η κλίμακα στο 0.

Δημιουργούμε μια νέα υπηρεσία με όνομα "WeatherService" από την εικόνα (image) weatherapp και με κλιμάκωση 3 και εμφανίζουμε τα διαθέσιμες υπηρεσίες (services):

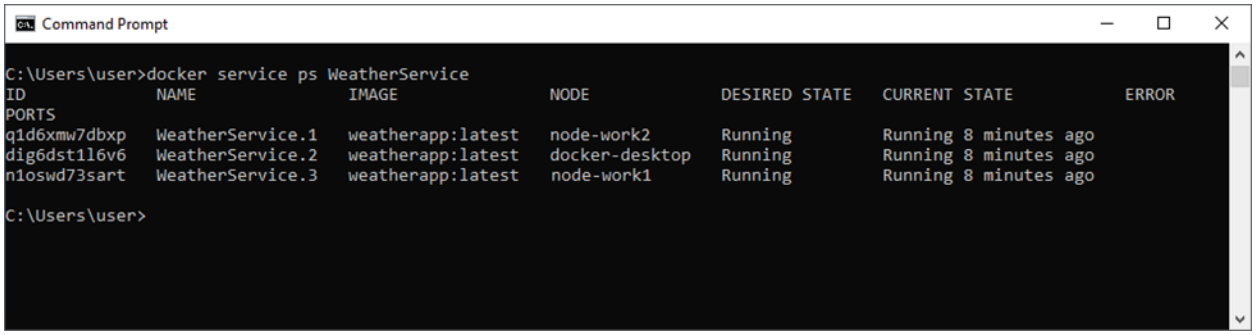
```
> docker service create --name "WeatherService" --replicas=3
weatherapp
```



Η γενική εντολή για να εμφανίσουμε τη λίστα με τις διαθέσιμες εργασίες (tasks) είναι η ακόλουθη:
```
> docker service ps <SERVICE-ID>
```

Για να εμφανίσουμε τη λίστα με τις διαθέσιμες εργασίες που δημιουργήθηκαν στην περίπτωση μας εκτελούμε την εντολή:

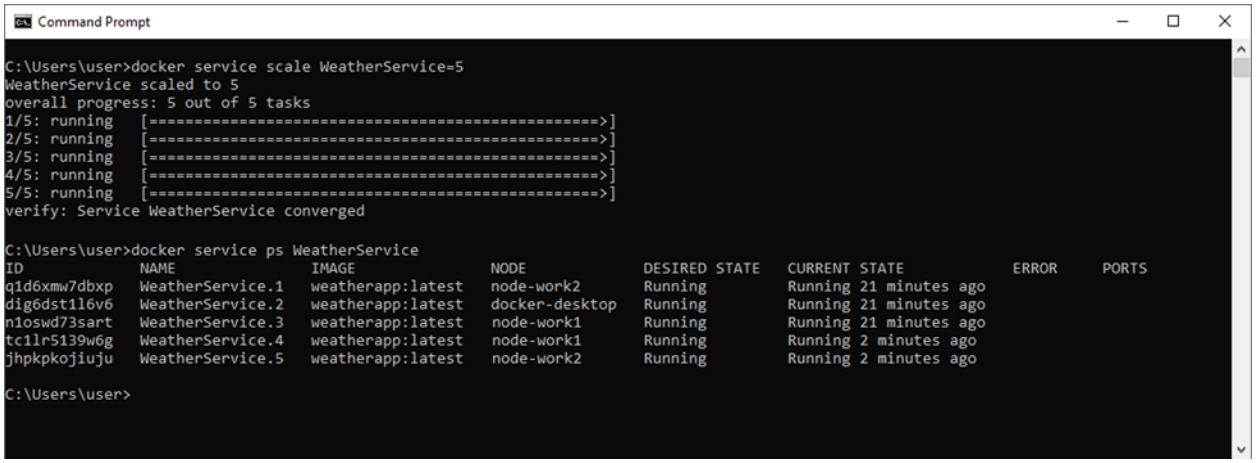
```
> docker service ps WeatherService
```



Παρατηρούμε ότι το swarm έχει δημιουργήσει 3 νέες εργασίες για κλιμάκωση σε συνολικά 3 εκτελούμενες παρουσίες του weatherapp. Οι εργασίες κατανέμονται μεταξύ των τριών κόμβων του σμήνους, μία από αυτές τρέχει στο manager node.

Αυξάνουμε την κλιμάκωση σε 5:

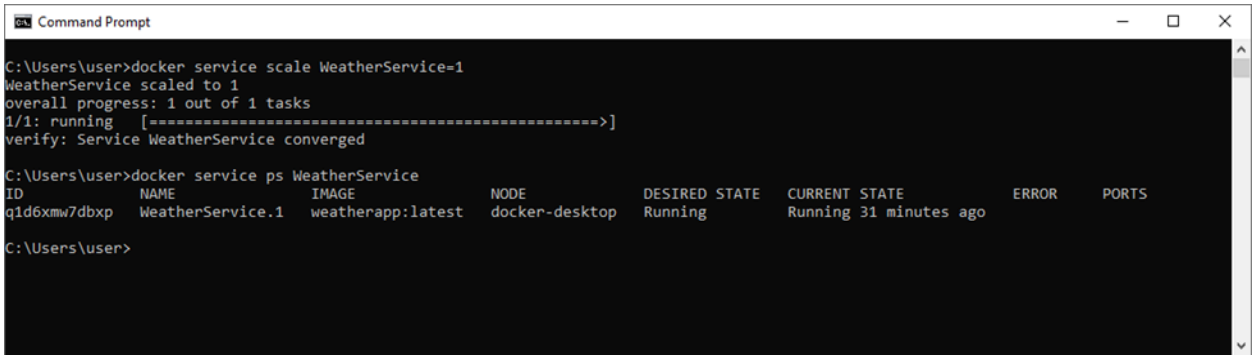
```
> docker service scale WeatherService=5
```



Παρατηρούμε αντίστοιχα πως διαμορφώνονται οι εργασίες.
Μπορούμε επίσης να μειώσουμε την κλιμάκωση σε 1:

```
> docker service scale WeatherService=1
```

Πλέον οι εργασίες έχουν περιοριστεί στον manager node.



Το “weatherapp” τρέχει ένα Cron Job (όπως ορίσαμε στο αρχείο application.properties) και δεν χρησιμοποιεί clustered αρχιτεκτονική στην οποία θα πρέπει να οριστεί ανεξάρτητη βάση δεδομένων η οποία θα περιέχει τα Jobs και τους προγραμματισμούς αυτών που θα πρέπει να εκτελεστούν. Επομένως σε περίπτωση που ένας worker αποτύχει, αμέσως ο manager μπορεί να ενεργοποιήσει τη συνέχεια της εργασίας σε άλλον worker.

Κεφάλαιο 7: Βιβλιογραφία

[1] IBM. (n.d.). *Smart farming*. Retrieved July 23, 2024, from https://www.ibm.com/topics/smart-farming#:~:text=Smart%20farming%2C%20also%20known%20as,Internet%20of%20Things%20(IoT).

[2] National Geographic Society. (n.d.). *Geographic information system (GIS)*. Retrieved July 23, 2024, from https://education.nationalgeographic.org/resource/geographic-information-system-gis/

[3] Jiva. (n.d.). *Why do we need smart agriculture?* Retrieved July 23, 2024, from https://www.jiva.ag/blog/why-do-we-need-smart-agriculture#:~:text=Smart%20farming%20allows%20agribusiness%20owners,manage%20the%20standard%20farming%20situations.

[4] Zhang, L. (2019). *Application of smart agriculture in food safety*. *Journal of Ethnic Foods*, 6(1), 1-10. https://doi.org/10.1186/s42779-019-0011-9

[5] Oracle. (n.d.). *Virtualization technology overview*. Retrieved July 23, 2024, from https://docs.oracle.com/cd/E26996_01/E18549/html/VMUSG1010.html

[6] Citrix. (n.d.). *What is hardware virtualization?* Retrieved July 23, 2024, from https://www.citrix.com/glossary/what-is-hardware-virtualization.html

[7] IBM. (n.d.). *Virtualization*. Retrieved July 23, 2024, from https://www.ibm.com/topics/virtualization

[8] ScienceDirect. (n.d.). *Container-based virtualization*. Retrieved July 23, 2024, from https://www.sciencedirect.com/topics/computer-science/container-based-virtualization

[9] Kubernetes. (n.d.). *Kubernetes documentation*. Retrieved July 23, 2024, from https://kubernetes.io/

[10] Aqua Security. (n.d.). *Kubernetes architecture*. Retrieved July 23, 2024, from https://www.aquasec.com/cloud-native-academy/kubernetes-101/kubernetes-architecture/

[11] Poulton, N. (2020). *Docker deep dive*. Leanpub.

[12] Poulton, N. (2021). *The Kubernetes book*. Leanpub.

[13] Hightower, K., Burns, B., & Beda, J. (2017). *Kubernetes up & running: Dive into the future of infrastructure*. O'Reilly Media.

[14] Newman, S. (2015). *Monolith to microservices: Evolutionary patterns to transform your monolith*. O'Reilly Media.

Κεφάλαιο 8: Παράρτημα Α

8.1 WeatherJob.java

```
package com.weather.telemetryscheduler.job;

import com.weather.telemetryscheduler.dto.CityDTO;
import com.weather.telemetryscheduler.service.WeatherService;
import com.weather.telemetryscheduler.utils.RestClient;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.scheduling.annotation.EnableScheduling;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Arrays;
import java.util.Date;
import java.util.List;


@Component
@EnableScheduling
public class WeatherJob {

	private static final Logger logger =
LoggerFactory.getLogger(WeatherJob.class);

	private static final SimpleDateFormat dateFormat = new
SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

	private WeatherService weatherService;


	public WeatherJob(WeatherService weatherService) {
	this.weatherService = weatherService;
	}


	@Scheduled(cron = "${cron.schedule.period}")
	public void performTaskUsingCron() throws IOException {

	logger.info("*** Executing job at: {}", dateFormat.format(new Date()));
```

```
	 /*
	* 1. Get data from web weather service
	*/

	RestClient client = new RestClient();

	for (String cityName : getCities()) {

		logger.info("--- >>> Fetching records for city: " + cityName);

		try {
		CityDTO cityData = client.getCityDetails(cityName);
		System.out.println(cityData);

		/*
			* 2. Store records to db
			*/
		weatherService.saveCityDetails(cityData);
		} catch (Exception ex) {
		logger.error("Network Error: Cannot fetch records for city name: "
+ cityName);
		}
	}
	}


	/**
	* @return List of city names
	*/
	private List<String> getCities() {

	return Arrays.asList("Athina",
		"Volos",
		"Pyrgos",
		"Karditsa",
		"Trikala",
		"Thessaloniki",
		"Larisa",
		"Patras",
		"Korinthos",
		"Chalcis",
		"Alexandroupoli",
		"Agrinio",
		"Argos",
		"Chania");
	}
```

```
}
```

8.2 RestClient.java

```
package com.weather.telemetryscheduler.utils;

import com.google.gson.*;

import java.io.IOException;
import java.math.BigDecimal;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;
import java.util.concurrent.TimeUnit;
import java.util.logging.Level;
import java.util.logging.Logger;

import com.weather.telemetryscheduler.dto.CityDTO;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;
import org.slf4j.LoggerFactory;

public class RestClient {

	private static final org.slf4j.Logger logger =
LoggerFactory.getLogger(RestClient.class);

	// Use the URL below to construct the necessary address to fetch data
records
	private static final String SERVER_URL = "https://wttr.in/";

	private final SimpleDateFormat restDateFormat = new
SimpleDateFormat("yyyy-MM-dd hh:mm a", Locale.US);
	private final SimpleDateFormat appDateFormat = new
SimpleDateFormat("dd-MM-yyyy");

	private final OkHttpClient client = new
OkHttpClient.Builder().connectTimeout(90, TimeUnit.SECONDS)
			.writeTimeout(90, TimeUnit.SECONDS).readTimeout(90,
TimeUnit.SECONDS).build();

	/**
	 * Εκτελούμε την REST κλάση και αντλούμε τα δεδομένα για την πόλη
```

```
	 *
	 * @param city το όνομα της πόλης
	 * @return δεδομενα για την πόλη
	 */
	public CityDTO getCityDetails(String city) {
		String url = SERVER_URL + city + "?format=j1&lang=el";

		String weatherData = getJsonData(url);

		// Create an object GsonBuilder
		GsonBuilder builder = new GsonBuilder();
		builder.setPrettyPrinting();
		Gson gson = builder.create();

		// Get the response as JsonObject and create a JsonArray
		JsonObject json = null;

		try {
			json = gson.fromJson(weatherData, JsonObject.class);

			// Create a JsonArray for nearest_area attribute
			JsonArray array = json.get("nearest_area").getAsJsonArray();
			System.out.println(array);
			String name = null;
			BigDecimal latitude = BigDecimal.ZERO;
			BigDecimal longitude = BigDecimal.ZERO;
			int population = 0;

			// Iterate array for nearest_area
			for (JsonElement jsonElement : array) {
				JsonObject object = jsonElement.getAsJsonObject();
				JsonArray areaName =
object.get("areaName").getAsJsonArray();
				// Iterate array for areaName
				for (JsonElement jsonElement2 : areaName) {
					// Get the value of city as (key:value)
					JsonObject object1 =
jsonElement2.getAsJsonObject();
					name = object1.get("value").getAsString();
				}

				// Get values
				latitude =
jsonElement.getAsJsonObject().get("latitude").getAsBigDecimal();
				longitude =
jsonElement.getAsJsonObject().get("longitude").getAsBigDecimal();
```

```
                        population =
jsonElement.getAsJsonObject().get("population").getAsInt();

                  }

                  if (name.length() == 0) {
                        return null;
                  }

                  // Create an object with that will contain response data
                  CityDTO cityDTO = new CityDTO();
                  cityDTO.setCityName(name);
                  cityDTO.setLatitude(latitude);
                  cityDTO.setLongitude(longitude);
                  cityDTO.setPopulation(population);

                  JsonArray currentConditions =
json.getAsJsonArray("current_condition");
                  System.out.println(currentConditions);

                  // Get values for measurements
                  for (JsonElement jsonElement : currentConditions) {

                        try {
                              int tempC =
jsonElement.getAsJsonObject().get("temp_C").getAsInt();
                              cityDTO.setTempC(tempC);
                              int humidity =
jsonElement.getAsJsonObject().get("humidity").getAsInt();
                              cityDTO.setHumidity(humidity);
                              int windSpeedKmph =
jsonElement.getAsJsonObject().get("windspeedKmph").getAsInt();
                              cityDTO.setWindSpeedKmph(windSpeedKmph);
                              int uvIndex =
jsonElement.getAsJsonObject().get("uvIndex").getAsInt();
                              cityDTO.setUvIndex(uvIndex);

                              int feelsLikeC =
jsonElement.getAsJsonObject().get("FeelsLikeC").getAsInt();
                              cityDTO.setFeelsLikeC(feelsLikeC);

                              String windDir16Point =
jsonElement.getAsJsonObject().get("winddir16Point").getAsString();
                              cityDTO.setWindDir16Point(windDir16Point);

                              int windDirDegree =
jsonElement.getAsJsonObject().get("winddirDegree").getAsInt();
```

```
                                cityDTO.setWindDirDegree(windDirDegree);

                                int visibility =
jsonElement.getAsJsonObject().get("visibility").getAsInt();
                                cityDTO.setVisibility(visibility);

                                int cloudCover =
jsonElement.getAsJsonObject().get("cloudcover").getAsInt();
                                cityDTO.setCloudCover(cloudCover);

                                int pressure =
jsonElement.getAsJsonObject().get("pressure").getAsInt();
                                cityDTO.setPressure(pressure);

                                String localObsDateTime =
jsonElement.getAsJsonObject().get("localObsDateTime").getAsString();
                                Date convertedDate =
restDateFormat.parse(localObsDateTime);
                                System.out.println("-------- Date: " +
convertedDate);
                                String dateAsString =
appDateFormat.format(convertedDate);
                                cityDTO.setSearchDate(dateAsString);
                                cityDTO.setObservationDate(convertedDate);

                                JsonArray weatherDescArray =
jsonElement.getAsJsonObject().get("weatherDesc").getAsJsonArray();
                                for (JsonElement jElement : weatherDescArray) {
                                        JsonObject object1 =
jElement.getAsJsonObject();
                                    // Read weatherDesc add it to response
object
                                        String weatherDesc =
object1.get("value").getAsString();
                                        cityDTO.setWeatherDesc(weatherDesc);
                                }

                        } catch (ParseException ex) {
                                logger.error("Problem with data parsing");
                        }
                }

                return cityDTO;

        } catch (JsonSyntaxException e) {
                logger.error("No records found. Data Error");
                throw new RuntimeException(e);
```

```
            }

        }

        /**
         * @param url to hit
         * @return records returned by calling Rest service, null otherwise
         */
        private String getJsonData(String url) {

            // Create a Request object using URL that will be called
            Request request = new Request.Builder().url(url).build();

            // Check if we get response records
            try (Response response = client.newCall(request).execute()) {
                if (response.isSuccessful() && response.body() != null) {
                    String responseString = response.body().string();
                    // System.out.println(responseString);
                    return responseString;
                }
            } catch (IOException e) {

Logger.getLogger(RestClient.class.getName()).log(Level.SEVERE, null, e);
                e.printStackTrace();
            }

            return null;
        }
}
```

8.3 WeatherService.java

```
package com.weather.telemetryscheduler.service;

import com.weather.telemetryscheduler.dto.CityDTO;

public interface WeatherService {

	void saveCityDetails(CityDTO cityDTO);

}
```

8.4 WeatherServiceImpl.java

```
package com.weather.telemetryscheduler.service;

import com.weather.telemetryscheduler.dto.CityDTO;
import com.weather.telemetryscheduler.entity.postgres.City;
import com.weather.telemetryscheduler.entity.postgres.Measurement;
import com.weather.telemetryscheduler.repository.postgres.CityRepository;
import
com.weather.telemetryscheduler.repository.postgres.MeasurementRepository;
import lombok.RequiredArgsConstructor;
import org.jetbrains.annotations.NotNull;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;

@RequiredArgsConstructor
@Service
public class WeatherServiceImpl implements WeatherService {

	private static final Logger logger =
LoggerFactory.getLogger(WeatherServiceImpl.class);

	private final CityRepository cityRepository;
```

```
	private final MeasurementRepository measurementRepository;

	private final SimpleDateFormat appDateFormat = new
SimpleDateFormat("dd-MM-yyyy");

	@Override
	public void saveCityDetails(CityDTO cityDTO) {

		try {
			// Search based on cityName

			List<City> resultList =
cityRepository.findByCityName(cityDTO.getCityName());

			if (resultList.isEmpty()) {
				logger.info("Add New City");

				City city = new City();
				city.setCityName(cityDTO.getCityName());
				city.setLatitude(cityDTO.getLatitude());
				city.setLongitude(cityDTO.getLongitude());
				city.setPopulation(cityDTO.getPopulation());
				cityRepository.save(city);
				resultList =
cityRepository.findByCityName(cityDTO.getCityName());
			}

			if (!resultList.isEmpty()) {

				logger.info("Check Update measurements");

				City city = resultList.get(0);

				Date searchDate =
appDateFormat.parse(cityDTO.getSearchDate());

				// Get measurements for this city
				List<Measurement> measurementList =
measurementRepository.findAllByCityNameAndObservationDate(city,
							cityDTO.getObservationDate());

				// Update measurement table
				if (measurementList.isEmpty()) {
					logger.info("New Measurement ---------- INSERT
--------------");
```

```
                                Measurement measurement =
getMeasurement(cityDTO, city, searchDate);

                                measurementRepository.save(measurement);
                            }
                        }

                } catch (ParseException ex) {
                        logger.error("Error");
                        ex.printStackTrace();
                }

        }

        @NotNull
        private static Measurement getMeasurement(CityDTO cityDTO, City city,
Date searchDate) {
                Measurement measurement = new Measurement();
                measurement.setCityName(city);
                measurement.setSearchDate(searchDate);
                measurement.setObservationDate(cityDTO.getObservationDate());

                measurement.setTempC(cityDTO.getTempC());
                measurement.setHumidity(cityDTO.getHumidity());
                measurement.setWindspeedKmph(cityDTO.getWindSpeedKmph());
                measurement.setUvIndex(cityDTO.getUvIndex());
                measurement.setWeatherDesc(cityDTO.getWeatherDesc());

                measurement.setFeelsLikeC(cityDTO.getFeelsLikeC());
                measurement.setWindDir16Point(cityDTO.getWindDir16Point());
                measurement.setWindDirDegree(cityDTO.getWindDirDegree());
                measurement.setVisibility(cityDTO.getVisibility());
                measurement.setCloudCover(cityDTO.getCloudCover());
                measurement.setPressure(cityDTO.getPressure());
                return measurement;
        }
}
```

```java
package com.weather.telemetryscheduler.repository.postgres;

import com.weather.telemetryscheduler.entity.postgres.City;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface CityRepository extends JpaRepository<City, String> {

	List<City> findByCityName(String cityName);

}
```

8.5 CityRepository.java

```java
package com.weather.telemetryscheduler.repository.postgres;

import com.weather.telemetryscheduler.entity.postgres.City;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface CityRepository extends JpaRepository<City, String> {

	List<City> findByCityName(String cityName);

}
```

8.6 MeasurementRepository.java

```
package com.weather.telemetryscheduler.repository.postgres;

import com.weather.telemetryscheduler.entity.postgres.City;
import com.weather.telemetryscheduler.entity.postgres.Measurement;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Date;
import java.util.List;

@Repository
public interface MeasurementRepository extends JpaRepository<Measurement,
Integer> {

      List<Measurement> findAllByCityNameAndObservationDate(City city, Date
observationDate);

}
```

8.7 City.java

```
package com.weather.telemetryscheduler.entity.postgres;

import java.io.Serializable;
import java.math.BigDecimal;
import java.util.List;
import javax.persistence.Basic;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name = "CITY")
public class City implements Serializable {

	private static final long serialVersionUID = 1L;

	@Id
	@Basic(optional = false)
	@Column(name = "city_name")
	private String cityName;

	@Column(name = "latitude")
	private BigDecimal latitude;

	@Column(name = "longitude")
	private BigDecimal longitude;

	@Column(name = "population")
	private Integer population;

	@OneToMany(cascade = CascadeType.ALL, mappedBy = "cityName")
	private List<Measurement> µeasurementList;

	public City() {
	}

	public City(String cityName) {
		this.cityName = cityName;
	}
```

```
	public City(String cityName, BigDecimal latitude, BigDecimal longitude,
Integer population) {
		this.cityName = cityName;
		this.latitude = latitude;
		this.longitude = longitude;
		this.population = population;
	}

	public String getCityName() {
		return cityName;
	}

	public void setCityName(String cityName) {
		this.cityName = cityName;
	}

	public List<Measurement> getMeasurementList() {
		return µeasurementList;
	}

	public void setMeasurementList(List<Measurement> µeasurementList) {
		this.µeasurementList = µeasurementList;
	}

	public BigDecimal getLatitude() {
		return latitude;
	}

	public void setLatitude(BigDecimal latitude) {
		this.latitude = latitude;
	}

	public BigDecimal getLongitude() {
		return longitude;
	}

	public void setLongitude(BigDecimal longitude) {
		this.longitude = longitude;
	}

	public Integer getPopulation() {
		return population;
	}

	public void setPopulation(Integer population) {
		this.population = population;
	}
```

```
	@Override
	public int hashCode() {
		int hash = 0;
		hash += (cityName != null ? cityName.hashCode() : 0);
		return hash;
	}

	@Override
	public boolean equals(Object object) {
		if (!(object instanceof City)) {
			return false;
		}
		City other = (City) object;
		if ((this.cityName == null && other.cityName != null)
				|| (this.cityName != null &&
!this.cityName.equals(other.cityName))) {
			return false;
		}
		return true;
	}

	@Override
	public String toString() {
		return "weatherapp.com.entities.City[ cityName=" + cityName + "
]";
	}

}
```

8.8 Measurement.java

```
package com.weather.telemetryscheduler.entity.postgres;

import java.io.Serializable;
import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
```

```
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

@Entity
@Table(name = "MEASUREMENT")
public class Measurement implements Serializable {

	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "id")
	private Integer id;

	@Column(name = "search_date")
	@Temporal(TemporalType.DATE)
	private Date searchDate;

	@Column(name = "observation_date")
	private Date observationDate;

	@Column(name = "temp_c")
	private int tempC;

	@Column(name = "humidity")
	private int humidity;

	@Column(name = "windspeed_kmph")
	private int windspeedKmph;

	@Column(name = "uv_index")
	private int uvIndex;

	@Column(name = "weather_desc")
	private String weatherDesc;

	@Column(name = "feels_like_c")
	private Integer feelsLikeC;

	@Column(name = "wind_dir16_point")
	private String windDir16Point;

	@Column(name = "wind_dir_degree")
```

```
	private Integer windDirDegree;

	@Column(name = "visibility")
	private Integer visibility;

	@Column(name = "cloud_cover")
	private Integer cloudCover;

	@Column(name = "pressure")
	private Integer pressure;

	@JoinColumn(name = "city_name", referencedColumnName = "city_name")
	@ManyToOne(optional = false)
	private City cityName;

	public Measurement() {
	}

	public Measurement(Integer id) {
		this.id = id;
	}

	public Measurement(Integer id, Date searchDate, int tempC, int humidity,
int windspeedKmph, int uvIndex,
			String weatherDesc) {
		this.id = id;
		this.searchDate = searchDate;
		this.tempC = tempC;
		this.humidity = humidity;
		this.windspeedKmph = windspeedKmph;
		this.uvIndex = uvIndex;
		this.weatherDesc = weatherDesc;
	}

	public Integer getId() {
		return id;
	}

	public void setId(Integer id) {
		this.id = id;
	}

	public Date getSearchDate() {
		return searchDate;
	}

	public void setSearchDate(Date searchDate) {
```

```
        this.searchDate = searchDate;
}

public int getTempC() {
        return tempC;
}

public void setTempC(int tempC) {
        this.tempC = tempC;
}

public int getHumidity() {
        return humidity;
}

public void setHumidity(int humidity) {
        this.humidity = humidity;
}

public int getWindspeedKmph() {
        return windspeedKmph;
}

public void setWindspeedKmph(int windspeedKmph) {
        this.windspeedKmph = windspeedKmph;
}

public int getUvIndex() {
        return uvIndex;
}

public void setUvIndex(int uvIndex) {
        this.uvIndex = uvIndex;
}

public String getWeatherDesc() {
        return weatherDesc;
}

public void setWeatherDesc(String weatherDesc) {
        this.weatherDesc = weatherDesc;
}

public City getCityName() {
        return cityName;
}
```

```
public Integer getFeelsLikeC() {
	return feelsLikeC;
}

public void setFeelsLikeC(Integer feelsLikeC) {
	this.feelsLikeC = feelsLikeC;
}

public String getWindDir16Point() {
	return windDir16Point;
}

public void setWindDir16Point(String windDir16Point) {
	this.windDir16Point = windDir16Point;
}

public Integer getWindDirDegree() {
	return windDirDegree;
}

public void setWindDirDegree(Integer windDirDegree) {
	this.windDirDegree = windDirDegree;
}

public Integer getVisibility() {
	return visibility;
}

public void setVisibility(Integer visibility) {
	this.visibility = visibility;
}

public Integer getCloudCover() {
	return cloudCover;
}

public void setCloudCover(Integer cloudCover) {
	this.cloudCover = cloudCover;
}

public Integer getPressure() {
	return pressure;
}

public void setPressure(Integer pressure) {
	this.pressure = pressure;
}
```

```
	public void setCityName(City cityName) {
		this.cityName = cityName;
	}

	public Date getObservationDate() {
		return observationDate;
	}

	public void setObservationDate(Date observationDate) {
		this.observationDate = observationDate;
	}

	@Override
	public int hashCode() {
		int hash = 0;
		hash += (id != null ? id.hashCode() : 0);
		return hash;
	}

	@Override
	public boolean equals(Object object) {
		if (!(object instanceof Measurement)) {
			return false;
		}
		Measurement other = (Measurement) object;
		if ((this.id == null && other.id != null) || (this.id != null &&
!this.id.equals(other.id))) {
			return false;
		}
		return true;
	}

	@Override
	public String toString() {
		return "Measurement{" + "id=" + id + ", searchDate=" + searchDate
+ ", tempC=" + tempC + ", humidity="
				+ humidity + ", windspeedKmph=" + windspeedKmph + ",
uvIndex=" + uvIndex + ", weatherDesc="
				+ weatherDesc + ", cityName=" + cityName + '}';
	}

}
```

8.9 CityDTO.java

```
package com.weather.telemetryscheduler.dto;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.math.BigDecimal;
import java.util.Date;

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class CityDTO {

	private String cityName;
	private BigDecimal latitude;
	private BigDecimal longitude;
	private Integer population;

	private String searchDate;
	private Date observationDate;
	private int tempC;
	private int humidity;
	private int windSpeedKmph;
	private int uvIndex;
	private String weatherDesc;
	private int feelsLikeC;
	private String windDir16Point;
	private int windDirDegree;
	private int visibility;
	private int cloudCover;
	private int pressure;

	// We use lombok to create necessary constructors, getter and setter
methods

	@Override
	public String toString() {
	return "CityDTO{" + "cityName=" + cityName +
		", latitude=" + latitude +
		", longitude=" + longitude +
		", population=" + population +
```

```
                ", ----  searchDate=" + searchDate +
                ", tempC=" + tempC +
                ", humidity=" + humidity +
                ", windSpeedKmph=" + windSpeedKmph +
                ", uvIndex=" + uvIndex +
                ", weatherDesc=" + weatherDesc + '}';
        }

}
```

8.10 MeasurementDTO.java

```
package com.weather.telemetryscheduler.dto;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class MeasurementDTO {

        private String searchDate;
        private String tempC;
        private String humidity;
        private String windspeedKmph;
        private String uvIndex;
        private String weatherDesc;

}
```

8.11 PostgresqlConfig.java

```
package com.weather.telemetryscheduler.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.boot.jdbc.DataSourceBuilder;
import org.springframework.boot.orm.jpa.EntityManagerFactoryBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.annotation.EnableTransactionManagement;

import javax.persistence.EntityManagerFactory;
import javax.sql.DataSource;

@Configuration
@EnableTransactionManagement
@EnableJpaRepositories(
      entityManagerFactoryRef = "postgresEntityManagerFactory",
      transactionManagerRef = "postgresTransactionManager",
      basePackages = {"com.weather.telemetryscheduler.repository.postgres"})
public class PostgresqlConfig {

      @Bean(name = "postgresDataSource")
      @ConfigurationProperties(prefix = "postgres.datasource")
      public DataSource dataSource() {
      return DataSourceBuilder.create().build();
      }

      @Bean(name = "postgresEntityManagerFactory")
      public LocalContainerEntityManagerFactoryBean
postgresEntityManagerFactory(
            EntityManagerFactoryBuilder builder,
            @Qualifier("postgresDataSource") DataSource dataSource) {

      return builder
            .dataSource(dataSource)
            .packages("com.weather.telemetryscheduler.entity.postgres")
            .persistenceUnit("postgresPU")
            .build();
```

```
	}

	@Bean(name = "postgresTransactionManager")
	public PlatformTransactionManager postgresTransactionManager(
		@Qualifier("postgresEntityManagerFactory") EntityManagerFactory
postgresEntityManagerFactory) {
	return new JpaTransactionManager(postgresEntityManagerFactory);
	}

	@Bean(name = "postgresJDBC")
	@Autowired
	public JdbcTemplate
postgresJdbcTemplate(@Qualifier("postgresDataSource") DataSource
postgresDataSource) {
	return new JdbcTemplate(postgresDataSource);
	}

}
```

8.12 application.properties

```
# Cron job period
# Run every hour
cron.schedule.period=* 0 * * * *

# Application Name
spring.application.name=TelemetryScheduler

#===============================================================================
# Database configuration
#===============================================================================
# Spring JPA
spring.jpa.open-in-view=false
spring.jpa.database=default
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.dialect =
org.hibernate.dialect.PostgreSQLDialect

# Docker Postgres
postgres.datasource.jdbc-url=jdbc:postgresql://172.17.0.2:5432/weather
postgres.datasource.username=postgres
postgres.datasource.password=adminadmin
postgres.datasource.driver-class-name= org.postgresql.Driver
```