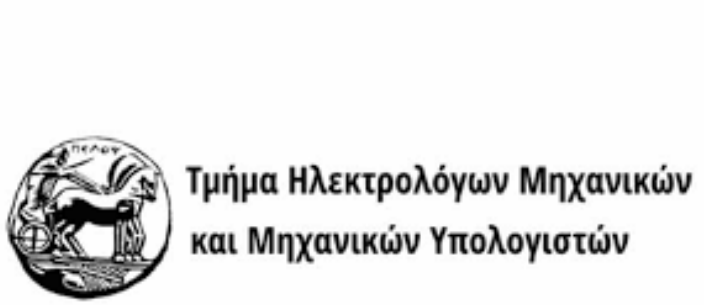


ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

" Web Based Platform for IoT data collection "

ΕΠΙΜΕΛΕΙΑ : ΝΙΚΟΛΑΚΗΣ ΜΙΧΑΗΛ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΡΙΑ : ΠΟΛΙΤΗ ΧΡΙΣΤΙΝΑ

ΑΘΗΝΑ
ΙΟΥΝΙΟΣ 2021

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

Αντικείμενο της πτυχιακής εργασίας

Αντικείμενο της παρούσας πτυχιακής εργασίας είναι η μελέτη των τεχνολογιών Docker, Flask, Python καθώς η υλοποίηση μίας εφαρμογής μέσω αυτών των τεχνολογιών. Η εφαρμογή έχει στόχο την δημιουργία μίας πλατφόρμας, στην οποία θα συνδέονται συσκευές για ιατρικούς σκοπούς. Μια από τις συσκευές αυτές θα είναι μια θερμική κάμερα η οποία θερμομετρά ασθενείς η εφαρμογή μας θα συλλέγει και θα αποθηκεύει τις θερμοκρασίες που καταγράφει η κάμερα.
Το διαδίκτυο των πραγμάτων (IoT) εξελίσσεται συνεχώς και γίνεται μια μεγάλη προσπάθεια για την πλήρη υιοθέτηση του στη βιομηχανία, στα συστήματα παραγωγής, συναρμολόγησης και γενικότερα στην καθημερινή ζωή του ανθρώπου. Σε γενικότερες γραμμές αναφέρεται ως ένα γρήγορο αναπτυσσόμενο δίκτυο συνδεδεμένων αντικειμένων που είναι σε θέση να συλλέγουν και να ανταλλάσσουν δεδομένα χρησιμοποιώντας ενσωματωμένους αισθητήρες, οι οποίοι βρίσκονται πάνω στις συσκευές και αντίστοιχα έχουν μια συγκεκριμένη λειτουργία ως προς το έργο που θα εκτελέσουν.
Επιλέξαμε να χρησιμοποιήσουμε τις τεχνολογίες Flask & Docker, διότι κάνουν τη εφαρμογή μας απλή, λειτουργική και σε μεγάλο βαθμό επεκτάσιμη. Ο στόχος του συγκεκριμένου project είναι η δημιουργία της πλατφόρμας στην οποία θα συνδέεται μια θερμική κάμερα και θα συλλέγονται οι θερμοκρασίες που αυτή καταγράφει.

Κεφαλαιο 1

1.1 IoT – Internet of Things (IoT)

Αρχικά θα αναλύσουμε το **διαδίκτυο των πραγμάτων** (**Internet of things**) και στη συνέχεια θα προχωρήσουμε αναλύοντας το Διαδίκτυο ιατρικών πραγμάτων(Internet of Medical Things (IoMT)).
Το διαδίκτυο των πραγμάτων (**Internet of things**) αποτελεί το δίκτυο επικοινωνίας συσκευών, οικιακών συσκευών, αυτοκινήτων καθώς και κάθε αντικειμένων που ενσωματώνουν ηλεκτρονικά μέσα, λογισμικό, αισθητήρες και συνδεσιμότητα σε δίκτυο ώστε να επιτρέπεται η σύνδεση και η ανταλλαγή δεδομένων. Απλούστερα, η φιλοσοφία του IoT είναι η σύνδεση όλων των ηλεκτρονικών συσκευών μεταξύ τους (σε ένα τοπικό δίκτυο) ή με δυνατότητα σύνδεσης στο διαδίκτυο.

1.2 Διαδίκτυο ιατρικών πραγμάτων (IoMT)

Το Διαδίκτυο των ιατρικών πραγμάτων (IoMT) είναι ένα σύνολο ιατρικών συσκευών και εφαρμογών που συνδέονται με συστήματα πληροφορικής υγειονομικής περίθαλψης ΜΕΣΩ ΔΙΑΔΥΚΤΥΟΥ. Οι ιατρικές συσκευές είναι εξοπλισμένες με τεχνολογίες ασύρματης επικοινωνίας και μπορούν να επικοινωνούν μεταξύ με κόμβους που απαρτίζουν το υγειονομικό πληροφοριακό σύστημα. Οι συσκευές IoMT συνδέονται με πλατφόρμες cloud, όπως τα Amazon web services, στις οποίες μπορούν να αποθηκευτούν και να αναλυθούν δεδομένα που συλλαμβάνονται. Το IoMT είναι επίσης γνωστό ως IoT υγειονομικής περίθαλψης. Ως παραδείγματα μπορούν να θεωρηθούν συσκευές οι οποίες έχουν εφαρμογές με στόχο να παρακολουθούν ασθενείς με μακροχρόνιες παθήσεις, ενημερώνοντας τόσο τους ίδιους καθώς και το αντίστοιχο ιατρικό προσωπικό, το οποίο επιβλέπει, την κατάσταση της υγείας τους.

1.3 Περιπτώσεις χρήσης του IoMT (Internet of Medical Things) και των εφαρμογών του

(1) Πάνω στο σώμα
Αισθητήρες σε συσκευές που βρίσκονται πάνω στο σώμα μπορούν να παρακολουθούν την υγεία του χρήστη ανά πάσα στιγμή και να καταγράφουν την θερμοκρασία τους, το οξυγόνο. Οι συσκευές αυτές μπορούν να ενσωματωθούν πάνω σε κάποια ενδυμασία, πάνω σε δέρμα ακόμα και κάτω από το δέρμα, παρέχοντας την πλήρη ελευθερία στους χρήστες.
(2) Νοσοκομεία
Στα νοσοκομεία γίνεται η μεγαλύτερη χρήση των IoMT συσκευών, με στόχο την βελτίωση της υγειονομικής περίθαλψης. Μερικές από τις εφαρμογές είναι η χρήση των μηχανημάτων ακτινογραφιών και οι εφαρμογές σχετικά με την σύνδεση των γιατρών με τους ασθενείς τους.
(3) Στα σπίτια
Οι εφαρμογές της ιατρικής δίνουν την δυνατότητα τόσο στους γιατρούς αλλά και στο αντίστοιχο προσωπικό να μπορούν να παρακολουθούν απομακρυσμένα τον ασθενή ο οποίος

πάσχει από σοβαρές παθήσεις με στόχο να αποφεύγει συνωστισμένους χώρους όπως τα νοσοκομεία, τα οποία θα μπορούσαν να βάλουν σε κίνδυνο την υγεία του ασθενή.

(4) Σε απομακρυσμένες περιοχές
Στην περίπτωση των απομακρυσμένων περιοχών πέρα από την δυνατότητα παρακολούθησης του ασθενούς λειτουργεί και ως μέσο σύνδεσης μεταξύ του γιατρού και του ασθενή.

1.4 Προκλήσεις του IoMT

Πολλά από τα μεγάλα νοσοκομεία σήμερα εξαρτώνται από το διαδίκτυο των ιατρικών πραγμάτων (IoMT) για την παροχή καλύτερων υπηρεσιών τόσο για τον ασθενή αλλά και για τον γιατρό. Σε σύγκριση με άλλους κλάδους οι διαχειριστές στην υγειονομική περίθαλψη αντιμετωπίζουν μοναδικές προκλήσεις με την χρήση αυτών των τεχνολογιών.

Πρόκληση 1

Ενίσχυση και διατήρηση των δεδομένων των ασθενών

Με βάση την έρευνα HIMSS Cybersecurity 2019, το 82% των οργανισμών υγειονομικής περίθαλψης παρουσίασαν σημαντικά περιστατικά ασφάλειας το 2018.
Στα νοσοκομεία όπου υπάρχουν τα περισσότερα μηχανήματα, για παράδειγμα τα μηχανήματα μαγνητικής τομογραφίας χρησιμοποιούνται κατά μέσο όρο για πάνω από μία δεκαετία καθώς και άλλες συσκευές είναι ιδιαίτερα ευάλωτες σε επιθέσεις ή κακή επικοινωνία με άλλες ιατρικές συσκευές λόγω της χρήσης τεχνολογίας παλαιού τύπου. Μάλιστα να αναφέρουμε εδώ πως το ένα τρίτο των συνδεδεμένων ιατρικών συσκευών δεν μπορούν συμβαδίσουν με όλες τις ήδη υπάρχουσες συνδεδεμένες συσκευές, συνεπώς αυτές οι συσκευές δεν μπορούν να ενημερωθούν για προστασία από νέους κινδύνους, επειδή δεν υποστηρίζονται πλέον από τον κατασκευαστή τους. Ενώ η παραβίαση της ασφάλειας είναι σοβαρή σε οποιονδήποτε κλάδο, μια επίθεση σε μηχανήματα τα οποία βρίσκονται σε νοσοκομεία έχουν τη δυνατότητα να επηρεάσουν σημαντικά τις ανθρώπινες ζωές οι οποίες στηρίζονται από αυτές τις συσκευές. Για παράδειγμα, ο οργανισμός FDA αρχικά ανακοίνωσε πρόσφατα ανακλήσεις για τους βηματοδότες και τις αντλίες ινσουλίνης με γνωστά ζητήματα ασφαλείας. Οι συμβιβασμοί στα κενά ασφάλειας σε αυτές τις κρίσιμες συσκευές θα μπορούσαν να οδηγήσουν σε σοβαρούς τραυματισμούς, αν όχι σε θανάτους.

Πρόκληση 2

Διατήρηση της συνδεσιμότητας όταν έχει μεγαλύτερη σημασία

Στα κρίσιμα περιστατικά στον τομέα της υγειονομικής περίθαλψης, το ιατρικό προσωπικό αλλά και οι ασθενείς θα πρέπει να βασίζονται πλήρως στην συνδεσιμότητα αυτών των συσκευών αλλά και του χρόνου απόκρισης. Συχνό φαινόμενο αποτελεί το γεγονός ότι οι αρχικές συνδέσεις χάνονται. Η αποτυχία των ασύρματων συνδέσεων μπορεί να χάνονται για διάφορους λόγους, όπως για παράδειγμα η υπερφόρτωση του δικτύου ή από φυσικά εμπόδια που διακόπτουν το σήμα. Στον τομέα της υγείας αυτό αποτελεί πολύ κρίσιμο σημείο. Η συνεχής συνδεσιμότητα απαιτεί μια αξιόπιστη υποδομή δικτύου. Για τις εφαρμογές ιατρικής απαιτείται ένα ευέλικτο, προσαρμοστικό

αλλά και ασφαλές δίκτυο για να μπορούν να συνδέονται χιλιάδες συσκευές πάνω σε αυτό. Μάλιστα το δίκτυο θα πρέπει να έχει μια ιεράρχηση προτεραιότητας διασφαλίζοντας την συνδεσιμότητα στις κρίσιμες συσκευές έναντι του μη κρίσιμου εξοπλισμού, παράδειγμα, εξοπλισμό που αφορά την ψυχαγωγία στο θάλαμο του ασθενή.

Πρόκληση 3

Μείωση του ανθρώπινου λάθους με χρήση συσκευών IoMT

Πρέπει να αναφέρουμε πως το περιβάλλον στα νοσοκομεία είναι αρκετά πιο περίπλοκο από όσο νομίζουμε και μάλιστα κάποιοι από τους ασθενείς εξαρτώνται από την τεχνολογία πιο πολύ από ποτέ. Σύμφωνα με την 30η έρευνα για την ηγεσία και το εργατικό δυναμικό των ΗΠΑ από το HIMSS, μόνο το 28% των νοσοκομείων ισχυρίστηκε πως το εργατικό δυναμικό της πληροφορικής για την υγεία τους είναι πλήρως εξοπλισμένο. Στην ίδια μελέτη, μόνο το 37% των ερωτηθέντων ανέμενε ότι θα αυξήσει τον εξοπλισμό στο εργατικό δυναμικό τους (προγραμματιστές) το 2019. Επομένως, οι ειδικοί της πληροφορικής έχουν την πρόκληση να επανεξετάσουν πώς να απλοποιήσουν τις διαδικασίες και τα συστήματα, βελτιώνοντας παράλληλα τη συνολική απόδοση και ασφάλεια. Πολλοί μάλιστα αναζητούν και την αυτοματοποίηση δικτύου.
Με τον αυτοματισμό στο δίκτυο του νοσοκομείου οι ομάδες IT μπορούν να τυποποιήσουν και να διευκολύνουν τις διαδικασίες του προσωπικού, να αυξήσουν την ποιότητα και την ποσότητα των πληροφοριών των ασθενών και να βελτιώσουν την αφοσίωση και την ικανοποίηση των ασθενών για καλύτερα αποτελέσματα στην υγειονομική τους περίθαλψη. Τελικά, ο αυτοματισμός μπορεί να βοηθήσει τους οργανισμούς υγειονομικής περίθαλψης να αφιερώσουν λιγότερο χρόνο στα διοικητικά καθήκοντα και περισσότερο χρόνο στη φροντίδα των ασθενών.
Η ποιοτική ιατρική περίθαλψη εξαρτάται τόσο από την τεχνολογία, όσο και από την ανθρώπινη νοημοσύνη που συνεργάζονται όσο το δυνατόν πιο αποτελεσματικά και αδιάκοπα γίνεται. Είτε πρόκειται για τη διασφάλιση συσκευών και την προστασία της ασφάλειας των ασθενών είτε για τη βελτίωση της συνδεσιμότητας ή τη βελτιστοποίηση των διαθέσιμων πόρων, οι προγραμματιστές που εξειδικεύονται στον τομέα υγείας είναι υπεύθυνοι για την πραγματοποίηση αυτού του οράματος και την δημιουργία για τις επόμενες εκπληκτικές εμπειρίες των ασθενών του αύριο.

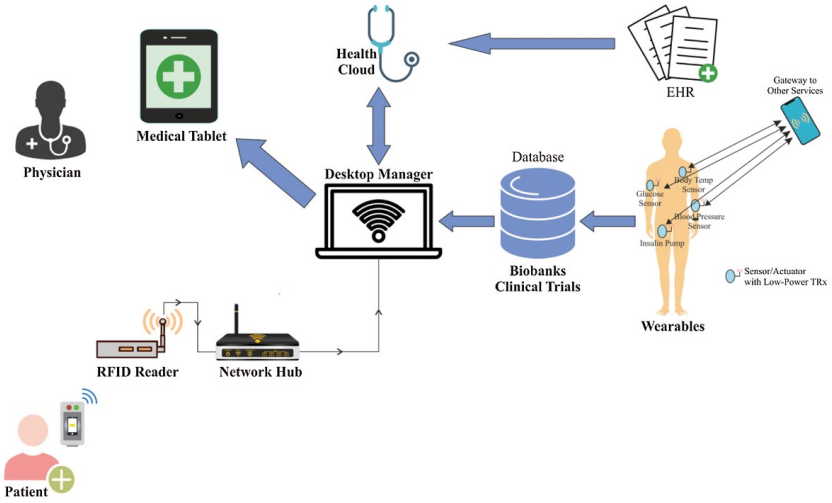
Συμπέρασμα

Οι προκλήσεις αυτές είναι αρκετά σημαντικές και πρέπει να πέρα από το να ληφθούν σοβαρά υπόψη, θα πρέπει τα μελλοντικά projects να συμβάλλουν στην ασφάλεια και επεκτασιμότητα τους. Η επεκτασιμότητα τους θα καταστήσει δυνατή την προσθήκη νέων τεχνολογιών και συσκευών για την βελτίωση παροχών τόσο στους εργαζόμενους όσο και στους ασθενείς. Στη συγκεκριμένη εφαρμογή μας έχει ληφθεί σοβαρά υπόψη η επεκτασιμότητα αλλά και η βελτιστοποίηση του κώδικα με στόχο την καλύτερη δυνατή λειτουργία της εφαρμογής με τις υπόλοιπες συσκευές που είναι συνδεδεμένες ή που πρόκειται να συνδεθούν.

ΚΕΦΑΛΑΙΟ 2

Αρχιτεκτονική Λογισμικού

Αυτή η αρχιτεκτονική λογισμικού ενός συστήματος περιγράφει τα κύρια συστατικά του, τις σχέσεις μεταξύ τους καθώς και τις κύριες αλληλεπιδράσεις μεταξύ τους.
Από τα κυριότερα χαρακτηριστικά της αρχιτεκτονικής λογισμικού είναι να δημιουργηθεί με επιτυχία μια σωστή βάση, για την εφαρμογή που αναπτύσσουμε. Πολλές φορές μια λάθος αρχιτεκτονική μπορεί να μας οδηγήσει να αρχίσουμε από την αρχή το project το οποίο και κατασκευάζουμε. Η αρχιτεκτονική λογισμικού συμβάλλει στο να αντιμετωπιστούν όλες οι απαραίτητες ανάγκες που μπορούν να προκύψουν (τεχνικές και λειτουργικές απαιτήσεις) με στόχο την βελτιστοποίηση της απόδοσης και της ασφάλειας του συστήματος. Κάθε απόφαση επιφέρει σημαντικές επιπτώσεις στην ποιότητα, την συντήρηση καθώς και στην απόδοση του συστήματος.
Στο Σχήμα 1: Αρχιτεκτονική μιας IoMT εφαρμογής μπορεί να δει κανείς την αρχιτεκτονική του IoMT



Σχήμα 1: Αρχιτεκτονική μιας IoMT εφαρμογής

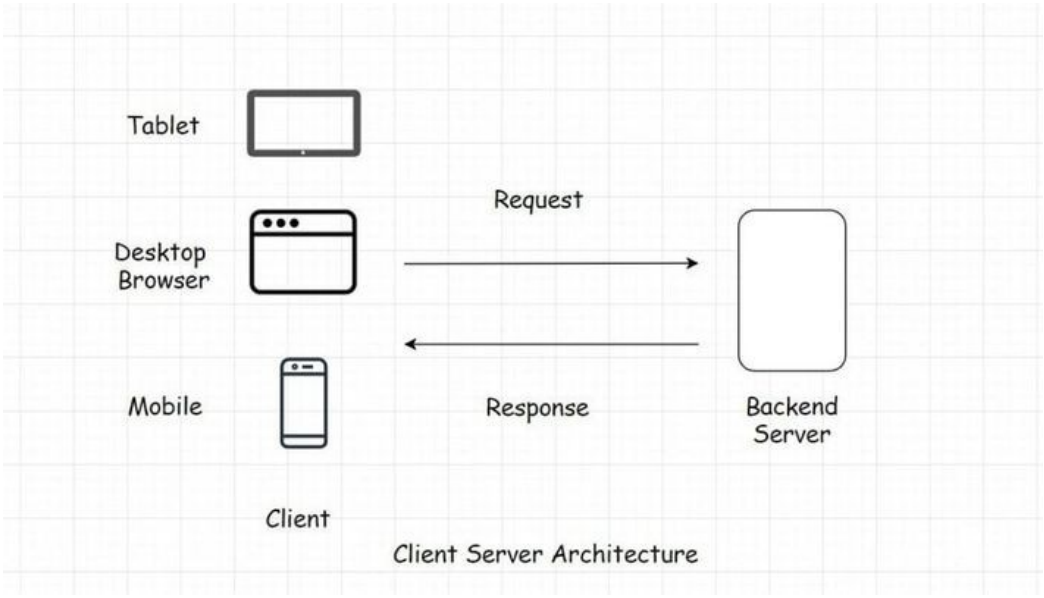
2.1 Διαφορές μεταξύ των ορισμών του σχεδιασμού λογισμικού & της αρχιτεκτονικής λογισμικού

Ο σχεδιασμός λογισμικού αναφέρεται στην σχεδίαση του κώδικα, ως προς τις λειτουργίες, τις ενότητες του κώδικα αλλά και το εύρος των κλάσεων. Αυτό δίνει την δυνατότητα στον προγραμματιστή να γλυτώνει επαναλαμβανόμενες ενέργειες με σκοπό να επιτυγχάνει υψηλότερες αποδόσεις. Η αρχιτεκτονική λογισμικού χρησιμοποιείται για την δημιουργία του "σκελετού", για τα υψηλού επιπέδου συστατικά, καθώς και την επιτυχή λειτουργία τους.

2.2 Παραδείγματα μοτίβων αρχιτεκτονικής λογισμικού

Client – Server

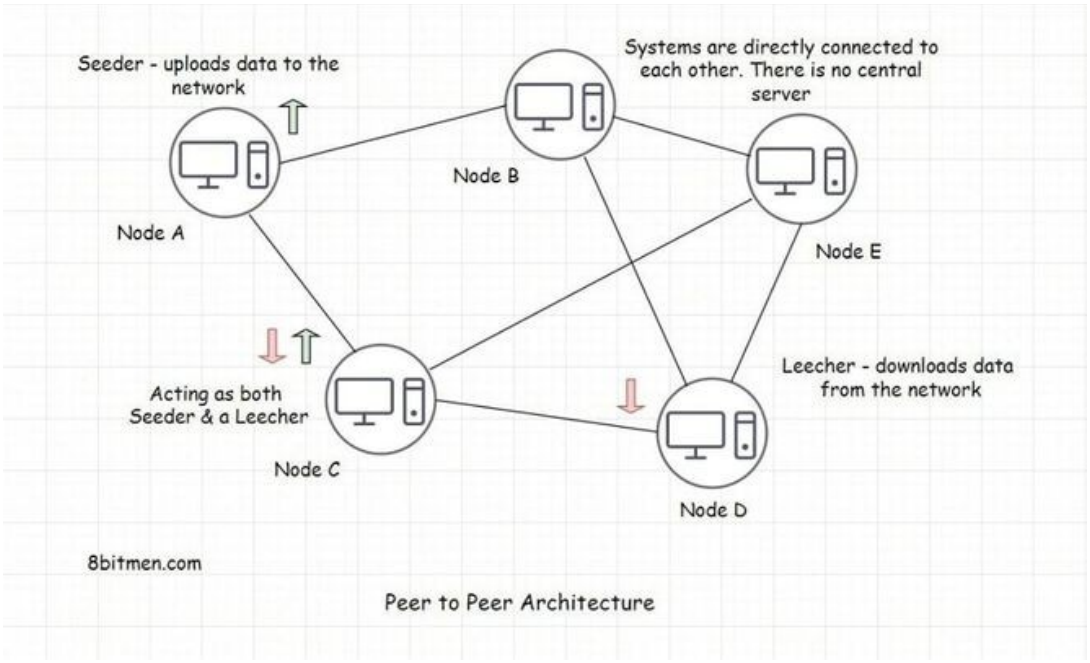
Στο συγκεκριμένο μοτίβο ο χρήστης στέλνει το "αίτημα" του στον server με τον οποίο επικοινωνεί για να πάρει πληροφορίες και έτσι ο server ανταποκρίνεται μετά από το αίτημά του. (Σχ. 2)



Σχήμα 2 : Client Server Αρχιτεκτονική

Peer to peer

Ένα δίκτυο peer to peer (Σχ. 3) διαφέρει από το προηγούμενο μοντέλο που αναλύσαμε, καθώς όλοι οι κόμβοι στο δίκτυο αυτό μπορούν και επικοινωνούν μεταξύ τους χωρίς να είναι απαραίτητη η ύπαρξη του κεντρικού διακομιστή (Server). Όλοι οι κόμβοι στο δίκτυο αυτό έχουν ίσα δικαιώματα . Σημαντικό είναι να πούμε πως ακόμα και αν μερικοί κόμβοι φύγουν από το δίκτυο, το δίκτυο θα συνεχίσει να είναι σε λειτουργία.



Σχήμα 3: Peer to Peer Αρχιτεκτονική

2.3 Μοτίβα σχεδίασης

Στις μέρες μας ένας προγραμματιστής μπορεί να γίνει πολύ πιο αποτελεσματικός όταν μπορεί να αποφύγει κοινά προβλήματα που μπορεί να παρουσιαστούν. Τα μοτίβα σχεδίασης είναι πολύ χρήσιμα όταν πρόκειται να χρησιμοποιηθούν για τους σωστά και στις κατάλληλες περιπτώσεις. Επίσης περιλαμβάνουν μια κοινή γλώσσα για τα επαναλαμβανόμενα προβλήματα τα οποία μπορούν να συζητηθούν από άλλους ή από άλλες ομάδες. Ωστόσο αποτελεί αντικείμενο διαμάχης από τους προγραμματιστές λόγω της μεγάλης χρήσης τους τα τελευταία χρόνια και αποτελεί μια πρόκληση η αποσφαλμάτωση, καθώς και το διάβασμα του κώδικα με αρκετά διαφορετικά μοτίβα σχεδίασης. Τέλος ο καλύτερος τρόπος για την αποφυγή προβλημάτων είναι να γνωρίζει ο προγραμματιστής "πού" αλλά και "γιατί" χρησιμοποιείται το κάθε μοτίβο σχεδίασης.

2.4 Παραδείγματα μοτίβων σχεδίασης

Ακολουθούν μερικά από τα μοτίβα σχεδίασης είναι τα εξής:

- Factory-method
  Σε αυτό το μοτίβο λογισμικού παράγουμε αντικείμενα. Και όχι μόνο αυτό - αυτό γίνεται χωρίς να καθοριστεί η ακριβής κατηγορία του αντικειμένου που θα δημιουργηθεί. Για να

επιτευχθεί αυτό, τα αντικείμενα δημιουργούνται καλώντας την μέθοδο αυτή αντί να καλέσουν έναν κατασκευαστή.

- Singleton
  Το μοτίβο “singleton” χρησιμοποιείται για τον περιορισμό της δημιουργίας μιας κλάσης σε ένα μόνο αντικείμενο. Αυτό είναι ωφέλιμο όταν απαιτείται ένα (και μόνο ένα) αντικείμενο για το συντονισμό των ενεργειών σε όλο το σύστημα. Υπάρχουν πολλά παραδείγματα όπου πρέπει να υπάρχει μόνο ένα στιγμιότυπο μιας κλάσης, συμπεριλαμβανομένων των caches, των ομαδοποιήσεων νημάτων και των μητρώων.

- Strategy
  Το μοτίβο στρατηγικής επιτρέπει την ομαδοποίηση σχετικών αλγορίθμων υπό μια αφαίρεση, η οποία επιτρέπει την εναλλαγή ενός αλγορίθμου ή πολιτικής για έναν άλλο χωρίς τροποποίηση του πελάτη. Ωστόσο αντί να εφαρμοστούν άμεσα σε έναν μόνο αλγόριθμο, ο κώδικας λαμβάνει τις οδηγίες χρόνου εκτέλεσης καθορίζοντας ποια ομάδα αλγορίθμων θα εκτελεστεί.

- Builder
  Το μοτίβο “builder” χρησιμοποιείται για την κατασκευή αντικειμένων. Μερικές φορές, τα αντικείμενα που δημιουργούμε μπορεί να είναι περίπλοκα, να αποτελούνται από πολλά υπο-αντικείμενα ή να απαιτούν μια περίπλοκη διαδικασία κατασκευής. Η δημιουργία σύνθετων τύπων μπορεί να απλοποιηθεί χρησιμοποιώντας το “builder” μοτίβο. Ένα σύνθετο ή ένα συγκεντρωτικό αντικείμενο είναι αυτό που γενικά ο “builder” δημιουργεί.

- Observer
  Σε αυτό το μοτίβο έχουμε μια εξάρτηση από ένα προς πολλά μεταξύ των αντικειμένων, με αποτέλεσμα όταν ένα αντικείμενο αλλάζει κατάσταση, ειδοποιούνται όλα τα εξαρτώμενα από αυτό. Αυτό γίνεται συνήθως καλώντας μια από τις μεθόδους τους. Για λόγους απλότητας, μπορούμε να χρησιμοποιήσουμε το παράδειγμα σχετικά με το τι συμβαίνει όταν παρακολουθούμε κάποιον στο Twitter. Ζητάμε ουσιαστικά από το Twitter να μας στείλει **(ο παρατηρητής)** ενημερώσεις tweet για το άτομο **(το θέμα)** που παρακολουθούμε. Το μοτίβο αποτελείται από δύο φορείς (actors), τον παρατηρητή που ενδιαφέρεται για τις ενημερώσεις και το θέμα που δημιουργεί τις ενημερώσεις. Ένα θέμα μπορεί να έχει πολλούς παρατηρητές και είναι μια σχέση “μία προς πολλές”. Ωστόσο, ένας παρατηρητής είναι ελεύθερος να εγγραφεί σε ενημερώσεις και από άλλα θέματα. Βασική θεώρηση: Στην περίπτωση πολλών θεμάτων και λίγων παρατηρητών, εάν κάθε θέμα αποθηκεύει τους παρατηρητές του ξεχωριστά, θα αυξήσει το κόστος αποθήκευσης καθώς ορισμένα θέματα θα αποθηκεύουν τον ίδιο παρατηρητή πολλές φορές.

- Adapter
  Αυτό το μοτίβο επιτρέπει στις ασυμβίβαστες κλάσεις να συνεργάζονται μετατρέποντας τη διεπαφή μιας κλάσης σε μια άλλη. Μπορούμε να το σκεφτούμε με ένα απλό παράδειγμα: Μερικές φορές όταν δύο κλάσεις επικοινωνούν η μία με την άλλη τότε μια συσχέτιση

έρχεται και λειτουργεί ως διερμηνέας μεταξύ των δύο κλάσεων, επιτρέποντας έτσι την επικοινωνία μεταξύ τους.

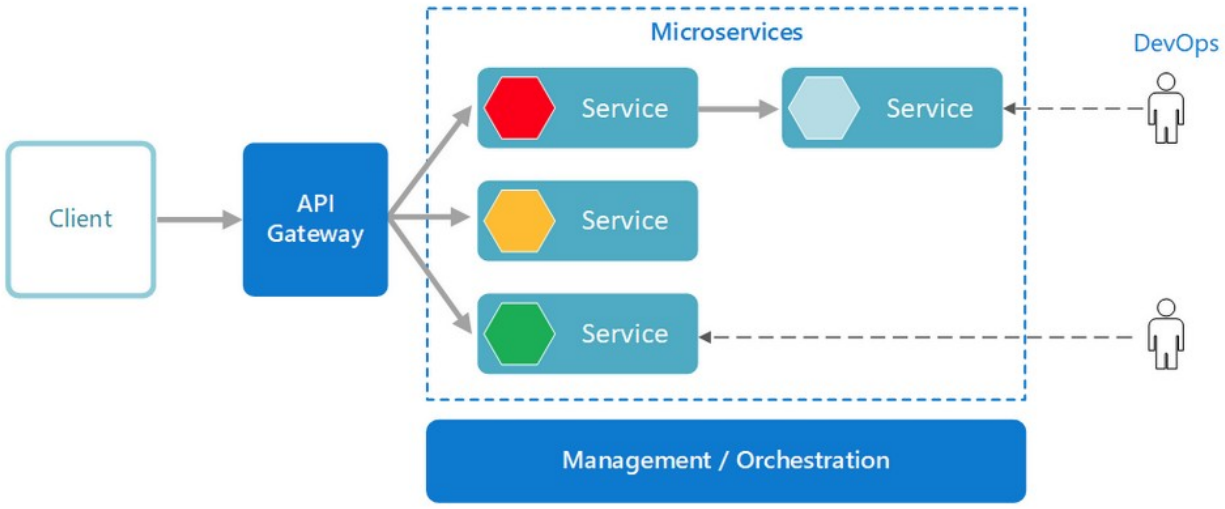
- State
  Το μοτίβο κατάστασης ενσωματώνει τις διάφορες καταστάσεις στις οποίες μπορεί να βρίσκεται μια μηχανή και επιτρέπει σε ένα αντικείμενο να αλλάξει τη συμπεριφορά του όταν αλλάζει η εσωτερική του κατάσταση. Το μηχάνημα ή το πλαίσιο, μπορεί να έχει δράσεις σε αυτό που το ωθούν σε διαφορετικές καταστάσεις. Χωρίς τη χρήση του μοτίβου, ο κώδικας γίνεται άκαμπτος και γεμάτος με όρους if-else.

Κεφάλαιο 3

Η Αρχιτεκτονική των Micro services στο IoT

3.1 Εισαγωγή

Η αρχιτεκτονική Microservices είναι μια από τις πιο συζητημένες αρχιτεκτονικές αλλά και πρώτη στις τάσεις αρχιτεκτονικής λογισμικού αυτή τη στιγμή και έχει αλλάξει για πάντα τον τρόπο κατασκευής των εταιρικών εφαρμογών. Αντί της αργής, περίπλοκης μονολιθικής προσέγγισης του παρελθόντος, οι προγραμματιστές και οι εταιρείες στρέφονται στην αρχιτεκτονική μικροϋπηρεσιών για να απλοποιήσουν και να κλιμακώσουν τις αρχιτεκτονικές δομές τους. Μάλιστα πολλές δημοφιλείς εταιρείες όπως το Amazon, το Netflix, το Spotify αλλά και το Uber έχουν κάνει ήδη την μετάβαση. Παρακάτω θα αναφερθούμε στη μονολιθική προσέγγιση, στα microservices αλλά και στα πλεονεκτήματα και μειονεκτήματα των microservices.



Σχήμα 4 : Αρχιτεκτονική των microservices

3.2 Μονολιθική προσέγγιση ή microservices

Η **μονολιθική αρχιτεκτονική** είναι ο παραδοσιακός τρόπος κατασκευής και ανάπτυξης εφαρμογών. Αυτή η δομή βασίζεται στην έννοια μιας ενιαίας μονάδας συμπεριλαμβανομένης της πλευράς του διακομιστή, της πλευράς του πελάτη και της βάσης δεδομένων. Όλες οι πτυχές ενοποιούνται και διαχειρίζονται ως ενιαία μονάδα στον κώδικα. Αυτό σημαίνει ότι οποιεσδήποτε ενημερώσεις πρέπει να γίνουν στον κώδικα, τότε πρέπει ολόκληρη η δομή να αλλάξει. Η επεκτασιμότητα και η αναβάθμιση εφαρμογών, είναι μια αρκετά περίπλοκη διαδικασία.

Η **αρχιτεκτονική των μικροϋπηρεσιών**, από την άλλη πλευρά, σε αυτή την αρχιτεκτονική δεν έχουμε τον όρο της ενιαίας μονάδας, όλα όσα χρειαζόμαστε ή πρόκειται να χρησιμοποιήσουμε είναι ή θα κατασκευαστούν ως ανεξάρτητες μονάδες που λειτουργούν και ως ξεχωριστές υπηρεσίες. Αυτό σημαίνει ότι κάθε υπηρεσία έχει τη δική της λογική και βάση κώδικα και επικοινωνούν μεταξύ τους μέσω API (Σχ. 4).

Κάνοντας πιο επεκτάσιμη μια εφαρμογή, η χρήση των microservices προσφέρει νέες δυνατότητες, καθώς νέες λειτουργικές μονάδες μπορούν να προστεθούν με μεγαλύτερη ευκολία και ταχύτητα.

3.3 Πλεονεκτήματα & μειονεκτήματα

Παρακάτω θα αναφερθούμε στα πλεονεκτήματα που προκύπτουν από τα microservices, τα οποία είναι και τα εξής.

- *Τα microservices βελτιώνουν την επεκτασιμότητα και την παραγωγικότητα*

Οι μεγάλες ομάδες συχνά πρέπει να συνεργάζονται σε σύνθετα έργα. Με τις μικροϋπηρεσίες, τα έργα μπορούν να χωριστούν σε μικρότερες, ανεξάρτητες μονάδες. Επιπλέον, οι ομάδες που είναι υπεύθυνες για κάθε μικροϋπηρεσία μπορούν να λάβουν τις δικές τους τεχνολογικές αποφάσεις ανάλογα με τις ανάγκες τους.

- *Ενσωματώνονται καλά με παλαιά συστήματα*

Τα μονολιθικά συστήματα είναι δύσκολο να συντηρηθούν. Πολλά παλαιά συστήματα είναι κακώς δομημένα, κακώς δοκιμασμένα ή εξαρτώνται από ξεπερασμένες τεχνολογίες. Οι μικροϋπηρεσίες μπορούν να λειτουργήσουν παράλληλα με παλαιότερα συστήματα για τη βελτίωση του κώδικα και την αντικατάσταση παλαιών τμημάτων του συστήματος. Η ολοκλήρωση θα είναι εύκολη και μπορεί να λύσει πολλά από τα προβλήματα που καθιστούν τα μονολιθικά συστήματα κομμάτια του παρελθόντος.

- *Υπάρχει βιώσιμη ανάπτυξη*

Οι αρχιτεκτονικές μικροϋπηρεσιών δημιουργούν συστήματα που παραμένουν διατηρήσιμα μακροπρόθεσμα, καθώς τα διάφορα μέρη μπορούν να αντικατασταθούν. Αυτό σημαίνει ότι μια μικροϋπηρεσία μπορεί εύκολα να ξαναγραφεί χωρίς να διακυβεύεται ολόκληρο το σύστημα.

- *Διαλειτουργικότητα*

Οι μικροϋπηρεσίες είναι οι καλύτερες για κατανεμημένες ομάδες, ομάδες σπαρμένες σε όλο τον κόσμο ή σε διάφορα τμήματα ενός οργανισμού ή μιας επιχείρησης. Οι μικροϋπηρεσίες παρέχουν την απαραίτητη ελευθερία και ευελιξία για να λειτουργούν αυτόνομα οι διαφορετικές ομάδες προγραμματιστών. Οι τεχνικές αποφάσεις μπορούν να ληφθούν γρήγορα, σχεδόν αμέσως μετά την ενσωμάτωση των νέων μικροϋπηρεσιών στο περιβάλλον του εργου.

Ωστόσο υπάρχουν και μειονεκτήματα όσον αφορά την χρήση τους, τα οποία πρέπει να ληφθούν υπόψη.

- *Η ανάπτυξη του κώδικα απαιτεί περισσότερη προσπάθεια*

Η λειτουργία ενός συστήματος με μικρουπηρεσίες απαιτεί συχνά περισσότερη προσπάθεια, καθώς υπάρχουν περισσότερες μονάδες που να αναπτύσσονται και να πρέπει να παρακολουθούνται. Πρέπει να πραγματοποιηθούν αλλαγές στις διεπαφές (APIs), έτσι ώστε να είναι δυνατή η ανεξάρτητη ανάπτυξη μεμονωμένων μικροϋπηρεσιών.

- *Οι δοκιμές πρέπει να είναι ανεξάρτητες*

Όλες οι μικρουπηρεσίες πρέπει να δοκιμαστούν όλες μαζί, αφού μια μικρουπηρεσία μπορεί να μπλοκάρει το στάδιο δοκιμής και να αποτρέψει την ανάπτυξη των άλλων μικροσυστημάτων. Υπάρχουν περισσότερες διεπαφές για δοκιμή και οι δοκιμές πρέπει να είναι ανεξάρτητες και για τις δύο πλευρές της διεπαφής.

- *Δύσκολο να αλλαχθούν πολλά microservices*

Οι αλλαγές που επηρεάζουν πολλές μικροϋπηρεσίες μπορεί να είναι πιο δύσκολο να εφαρμοστούν. Σε ένα σύστημα μικροϋπηρεσιών, οι αλλαγές απαιτούν αρκετές συντονισμένες αναπτύξεις.

3.5 Τελικά συμπεράσματα

Η χρήση των microservices αποτελούν κλειδί για πολλές εφαρμογές, καθώς τις κάνουν επεκτάσιμες και προσφέρουν όλα τα παραπάνω πλεονεκτήματα στα οποία αναφερθήκαμε. Ωστόσο χρειάζεται ιδιαίτερη προσοχή διότι μπορούν να οδηγήσουν μακροπρόθεσμα σε άλλα προβλήματα τα οποία είναι δύσκολο να διορθωθούν. Δεν υπάρχει απόλυτη ανάγκη και ούτε είναι πάντα απαραίτητο όλες οι εφαρμογές να δημιουργηθούν με microservices καθώς υπάρχουν και εφαρμογές που είναι απλές από τη φύση και τη λειτουργικότητά τους. Μετά την παραπάνω ανάλυση καταλήγουμε πως τα microservices είναι μια καλή επιλογή ως προς την χρήση στο έργο που αναπτύσσουμε καθώς προσφέρουν όλα τα παραπάνω πλεονεκτήματα.

ΚΕΦΑΛΑΙΟ 4

4.1 DOCKER

Οι εικόνες container γίνονται containers κατά την εκτέλεση και στην περίπτωση των Docker containers οι εικόνες γίνονται containers όταν εκτελούνται στο Docker Engine. Το Docker Engine είναι διαθέσιμο τόσο για Linux όσο και σε Windows, το λογισμικό πακεταρισμένο σε container θα λειτουργεί πάντα με τον ίδιο τρόπο, ανεξάρτητα από την υποδομή και το λειτουργικό σύστημα. Η τεχνολογία Docker containers ξεκίνησε το 2013 ως container Engine ανοιχτού κώδικα. Αξιοποίησε υπάρχουσες υπολογιστικές έννοιες γύρω από τα containers στον κόσμο του Linux, θεμελιώδες έννοιες των cgroups και namespaces. Η επιτυχία στον κόσμο του Linux οδήγησε σε μια συνεργασία με τη Microsoft που έφερε τα Docker containers και τη λειτουργικότητά τους στον κόσμο των Windows. Το Docker καθιστά την ανάπτυξη αποτελεσματική και προβλέψιμη και χρησιμοποιείται καθ 'όλη τη διάρκεια του κύκλου ζωής ανάπτυξης λογισμικού για μια γρήγορη, εύκολη και φορητή ανάπτυξη εφαρμογών – για τον επιτραπέζιο υπολογιστή και το cloud. Το Docker χρησιμοποιείται τουλάχιστον από έντεκα εκατομμύρια προγραμματιστές, υπάρχει σε εφτά εκατομμύρια εφαρμογές.

4.1.2 Containers or Images

Ένα container ξεκινά με την εκτέλεση μιας εικόνας. Μια εικόνα είναι ένα εκτελέσιμο πακέτο που περιλαμβάνει όλα όσα χρειάζονται για την εκτέλεση μιας εφαρμογής, δηλαδή ο κώδικας, ένας χρόνο εκτέλεσης, βιβλιοθήκες, μεταβλητές περιβάλλοντος και αρχεία διαμόρφωσης. Ένα container είναι ένα εκτελέσιμο στιγμιότυπο μιας εικόνας.

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

Σχήμα 5 : Παράδειγμα Dockerfile

Οι εικόνες δημιουργούνται από το Docker Engine με βάση τις οδηγίες που είναι γραμμένες σε ένα Dockerfile που καθορίζει τι συμβαίνει στο περιβάλλον μέσα στο container. Η πρόσβαση σε πόρους, όπως διεπαφές δικτύου και μονάδες δίσκου εικονοποιείται μέσα σε αυτό το περιβάλλον, το οποίο είναι απομονωμένο από το υπόλοιπο σύστημα. Για παράδειγμα, ο χρήστης μπορεί να χρειαστεί να δώσει εντολή στο Docker να αντιστοιχίσει ορισμένες θύρες από το container στη διεπαφή δικτύου του λειτουργικού συστήματος κεντρικού υπολογιστή ή να αντιγράψει ορισμένα αρχεία από το λειτουργικό σύστημα κεντρικού υπολογιστή στο container (π.χ. αρχεία διαμόρφωσης, δέσμες ενεργειών κ.λπ.), ένα παράδειγμα φαίνεται στο σχήμα (Σχ. 5).

4.1.3 docker-compose

Το docker compose (ονομάζεται επίσης docker-compose) είναι ένα εργαλείο για το σχεδιασμό και την εκτέλεση πολλαπλών container εφαρμογών. Ακολουθώντας το μοτίβο μιας εφαρμογής ανά Docker container, είναι δύσκολο να δημιουργηθούν ή να αποθηκευτούν εφαρμογές που έχουν πολλά στοιχεία ή διακομιστές. Χάρη στο docker-compose, ο χρήστης μπορεί εύκολα να ορίσει πολλαπλές υπηρεσίες (σε αρχείο YAML), να τις συνδέσει μεταξύ τους και να τις εκτελέσει. Στο αρχείο YAML, ο χρήστης ορίζει κάθε υπηρεσία, καθώς επίσης και την αλληλεξάρτηση των υπηρεσιών μεταξύ τους. Ενημερώνει το Docker για σημαντικά χαρακτηριστικά κάθε υπηρεσίας, όπως αν θα χρησιμοποιηθεί μια εικόνα container από ένα μητρώο ή θα δημιουργηθεί τοπικά από ένα Dockerfile. Ενημερώνει επίσης την μηχανή docker για το εάν υπάρχουν θύρες που πρέπει να

εκτεθούν (οι οποίες μπορούν επίσης να οριστούν σε ένα Dockerfile), καθώς και για το αν ορισμένα containers μοιράζονται συστήματα αρχείων (χρησιμοποιώντας την παράμετρο τόμων) ή εάν μια υπηρεσία πρέπει να περιμένει πριν ξεκινήσει. Ένα παράδειγμα αρχείου docker compose yaml μπορεί να δει κανείς στο σχήμα έξι και περαιτέρω εξέταση της σύνταξης μπορεί να αναζητηθεί στα επίσημα έγγραφα του docker-compose

```
version: '3'
services:
  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
    - .:/code
    - logvolume01:/var/log
    links:
    - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Σχήμα 6 : Παράδειγμα docker-compose.yaml

Συμπερασματικά, το Dockerfile ορίζει μια ενιαία μικροϋπηρεσία και το δοχείο του, ενώ το dockercompose καθορίζει ολόκληρη την εφαρμογή και τις σχέσεις μεταξύ των δοχείων. Υπάρχουν χαρακτηριστικά κοντέινερ που μπορούν να οριστούν από οποιοδήποτε αρχείο, η επιλογή βασίζεται σε διάφορες παραμέτρους της ανάπτυξης της εφαρμογής.

4.2 Στοιχεία από έρευνες

- Τα 2/3 από τις εταιρίες που προσπάθησαν να χρησιμοποιήσουν το Docker, τελικά και το υιοθέτησαν.
- Η υιοθέτηση του Docker αυξήθηκε κατά 30% τον τελευταίο χρόνο.
- Τα PHP, Ruby, Java και Node είναι τα κύρια πλαίσια προγραμματισμού που χρησιμοποιούνται σε containers.

4.3 Πλεονεκτήματα με την χρήση του Docker

Τα Docker contrainers εξασφαλίζουν τη συνοχή σε πολλούς κύκλους ανάπτυξης, τυποποιώντας το περιβάλλον τους. Ένα από τα μεγαλύτερα πλεονεκτήματα μιας αρχιτεκτονικής που βασίζεται στο

Docker είναι στην πραγματικότητα η τυποποίηση. Το Docker προσφέρει αναπαραγόμενα περιβάλλοντα κατασκευής, δοκιμών, ανάπτυξης και παραγωγής. Η τυποποίηση της υποδομής υπηρεσιών σε ολόκληρο το project επιτρέπει σε κάθε μέλος της ομάδας να εργάζεται σε ένα περιβάλλον ισοτιμίας παραγωγής. Με αυτόν τον τρόπο, οι προγραμματιστές είναι περισσότερο εξοπλισμένοι για την αποτελεσματική ανάλυση και διόρθωση σφαλμάτων εντός της εφαρμογής. Αυτό μειώνει τον χρόνο που θα σπαταλούσε κάποιος για τα ελαττώματα της εφαρμογής και αυξάνει τον διαθέσιμο χρόνο για την ανάπτυξη λειτουργιών. Όπως αναφέραμε, τα Docker containers μας επιτρέπουν να αλλάξουμε ρυθμίσεις και συστατικά στις εικόνες (images) του Docker , καθώς και να ελέγχουμε την έκδοση τους. Για παράδειγμα, εάν κάνουμε μια αναβάθμιση κάποιου στοιχείου το οποίο θα προκαλούσε μια καταστροφική βλάβη σε ολόκληρο το περιβάλλον μας, θα είναι πολύ εύκολο να κάνουμε μια επαναφορά σε κάποια προηγούμενη έκδοση της εικόνας του Docker. Επίσης ολόκληρη αυτή η διαδικασία μπορεί να ελεγχθεί και να δοκιμαστεί σε μόλις λίγα λεπτά. Μάλιστα, η εκκίνηση εικόνων Docker είναι τόσο γρήγορη όσο και η εκκίνηση μιας διαδικασίας.

Συμβατότητα και συντήρηση

Με το Docker, όπως προαναφερθήκαμε, μπορούμε να εξαλείψουμε πολύ γρήγορα το πρόβλημα με στόχο την επαναλειτουργία του μηχανήματος μας. Ένα από τα οφέλη που συνήθως οι περισσότερες ομάδες εκτιμούν είναι η ισοτιμία. Με την ισοτιμία, από την άποψη του Docker, εννοούμε πως οι εικόνες μας τρέχουν ταυτόχρονα, ανεξάρτητα από τον διακομιστή ή τον φορητό υπολογιστή στον οποίο λειτουργούν. Για τους προγραμματιστές, αυτό σημαίνει ακόμα λιγότερος χρόνος που αφιερώνεται από αυτούς για τη ρύθμιση περιβάλλοντων, για τον εντοπισμό σφαλμάτων σε συγκεκριμένα περιβάλλοντα και για μια πιο φορητή και εύχρηστη βάση κώδικα. Η ισοτιμία σημαίνει επίσης ότι η υποδομή της παραγωγής του κώδικα μας θα είναι πιο αξιόπιστη και πιο εύκολη στη συντήρηση.

Απλότητα και γρηγορότερες διαμορφώσεις

Ένα από τα βασικότερα οφέλη του Docker είναι ο τρόπος που απλοποιεί τα πράγματα. Οι χρήστες μπορούν να πάρουν τη δική τους διαμόρφωση, να τη βάλουν στον κώδικα τους και να την αναπτύξουν χωρίς προβλήματα που μπορεί να τους καθυστερήσουν ή να τους κουράσουν. Καθώς το Docker μπορεί να χρησιμοποιηθεί σε πολλά είδη περιβαλλόντων, οι απαιτήσεις της υποδομής δεν συνδέονται πλέον με το περιβάλλον της εφαρμογής.

Ταχεία ανάπτυξη

Το Docker καταφέρνει να μειώσει την ανάπτυξη σε μηδαμινό χρόνο. Αυτό οφείλεται στο γεγονός ότι δημιουργεί ένα container για κάθε διαδικασία και δεν εκκινεί εκ νέου ένα λειτουργικό σύστημα. Τα δεδομένα μπορούν να δημιουργηθούν και να καταστραφούν χωρίς να υπάρχει κάποια ανησυχία για το κόστος σχετικά με την ανάκτηση τους.

Συνεχής ανάπτυξη και δοκιμή

Το Docker εξασφαλίζει σταθερά περιβάλλοντα από την ανάπτυξη έως την παραγωγή. Τα Docker containers είναι διαμορφωμένα, έτσι ώστε να διατηρούν όλες τις διαμορφώσεις και τις εσωτερικές εξαρτήσεις. Μπορούμε να χρησιμοποιήσουμε το ίδιο container από την ανάπτυξη έως την παραγωγή, διασφαλίζοντας έτσι πως δεν θα υπάρχουν ασυμφωνίες ή χειροκίνητη παρέμβαση. Εάν πρέπει να πραγματοποιηθεί κάποια αναβάθμιση κατά τη διάρκεια του κύκλου ζωής ενός προϊόντος, τότε μπορούμε εύκολα να κάνουμε τις απαραίτητες αλλαγές στα Docker containers, να τα δοκιμάσουμε και να εφαρμόσουμε τις ίδιες αλλαγές στα ήδη υπάρχοντα containers. Αυτό το είδος ευελιξίας είναι άλλο ένα βασικό πλεονέκτημα της χρήσης του Docker. Το Docker μας επιτρέπει εύκολα να δημιουργήσουμε, να δοκιμάσουμε και να αφαιρέσουμε εικόνες (images) που μπορούν να αναπτυχθούν σε πολλούς διακομιστές. Επίσης από την στιγμή που θα υπάρξει μια νέα ενημέρωση στην έκδοση του κώδικα ασφαλείας, η διαδικασία παραμένει η ίδια. Μπορούμε να εφαρμόσουμε την έκδοση που θέλουμε, να τη δοκιμάσουμε και να την αφήσουμε στην παραγωγή.

Απομόνωση

Το Docker μας διασφαλίζει ότι οι εφαρμογές και οι πόροι θα είναι απομονωμένοι και διαχωρισμένοι. Το Docker μάλιστα διασφαλίζει ότι κάθε container έχει τους δικούς του πόρους που είναι απομονωμένοι από άλλα containers. Μάλιστα μπορούμε να έχουμε διάφορα containers για ξεχωριστές εφαρμογές που εκτελούν εντελώς διαφορετικές στοίβες. Το Docker επίσης μας βοηθά στο να διασφαλίσουμε την καθαρή αφαίρεση εφαρμογών, καθώς και πως κάθε εφαρμογή εκτελείται στο δικό της container. Εάν δεν χρειαζόμαστε πλέον μια εφαρμογή, μπορούμε απλώς να διαγράψουμε το container της. Αυτό δεν θα αφήσει προσωρινά αρχεία ή αρχεία διαμόρφωσης στο λειτουργικό μας σύστημα. Εκτός από αυτά τα οφέλη, το Docker διασφαλίζει επίσης ότι κάθε εφαρμογή χρησιμοποιεί μόνο πόρους που του έχουν ανατεθεί. Μια συγκεκριμένη εφαρμογή δεν θα χρησιμοποιήσει όλους τους διαθέσιμους πόρους, κάτι που συνήθως οδηγεί σε υποβάθμιση της απόδοσης ή σε πλήρη διακοπή λειτουργίας.

Ασφάλεια

Από άποψη ασφάλειας, το Docker διασφαλίζει ότι οι εφαρμογές που εκτελούνται σε container είναι πλήρως διαχωρισμένες και απομονωμένες μεταξύ τους, παρέχοντάς στον προγραμματιστή τον πλήρη έλεγχο της ροής και της διαχείρισης της κυκλοφορίας. Κανένα Docker container δεν μπορεί να εξετάσει διαδικασίες που εκτελούνται μέσα σε άλλο container. Από αρχιτεκτονική άποψη, κάθε container αποκτά το δικό του σύνολο πόρων, από την επεξεργασία έως τις στοίβες δικτύου.

4.4 Διαφορές μεταξύ Docker containers & Εικονικών μηχανών (Virtual Machines)

Οι εφαρμογές Dockerized είναι απλώς διαδικασίες που εκτελούνται στο σύστημά που χρησιμοποιούμε. Δεν απαιτεί την εκτέλεση Hypervisor (όπως το VMWare ή το VirtualBox),

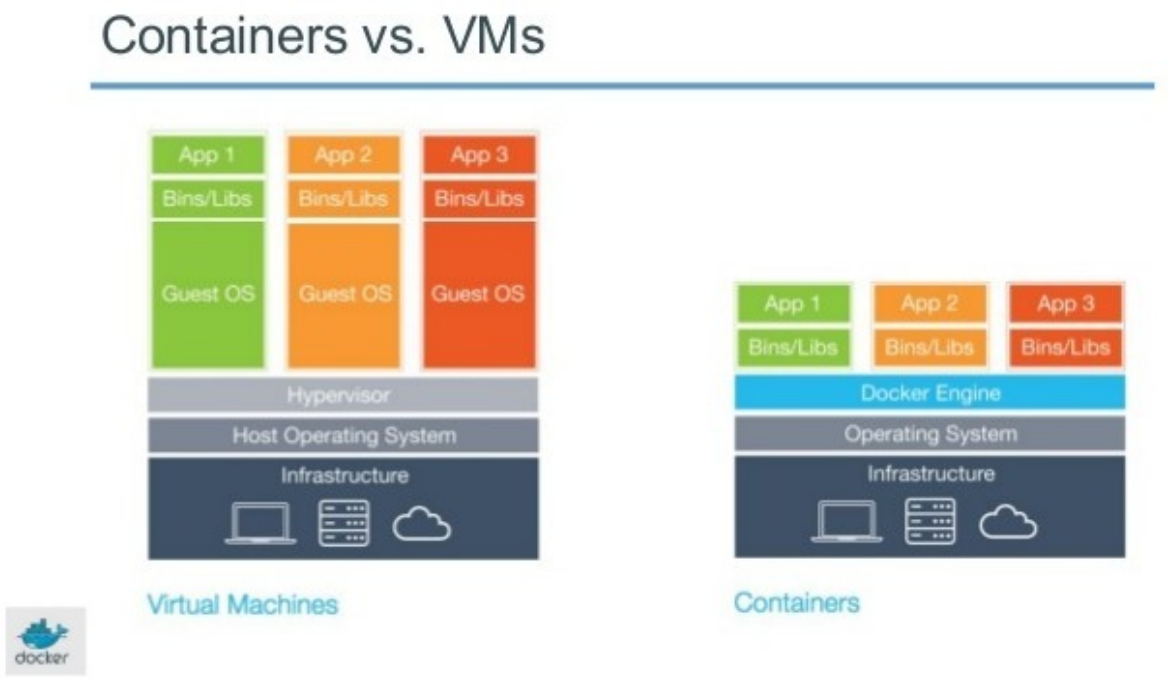
πράγμα που σημαίνει ότι δεν υπάρχει κανένας χρήστης (guest) στο λειτουργικό σύστημα που χρησιμοποιούμε. Υπάρχουν λόγοι για τη χρήση των Virtual Machines στις μέρες μας, αλλά λύνουν ένα διαφορετικό σύνολο προβλημάτων από το Docker. Μπορούμε να χρησιμοποιήσουμε το Docker για να απομονώσουμε μεμονωμένες εφαρμογές και να χρησιμοποιήσουμε τα Virtual Machines για να απομονώσουμε ολόκληρο το σύστημα μας. Ένα container που λειτουργεί ως προνομιακό μέσο μέσα σε μια εικονική μηχανή, χωρίς περιορισμούς πόρων, προφίλ ασφαλείας και ούτω καθεξής, είναι ένα είδος έξυπνου tarball και τίποτα περισσότερο. Η εκκίνηση μιας εικονικής μηχανής είναι ακριβότερη από την άποψη του χρόνου από την εκκίνηση ενός container.

4.4.1 Containers & VMs

Τα VMs είναι εικονικοί servers, οι οποίοι λειτουργούν σαν πραγματικοί servers. Σε μια εικονική μηχανή βασίζεται το συστήμα μας για να μπορέσει να μιμηθεί το ίδιο ακριβώς περιβάλλον στο οποίο εγκαθιστούμε την εφαρμογή μας. Οι εικονικές μηχανές εκτελούν ολόκληρο το λειτουργικό σύστημα, επιτρέποντάς μας να αντικαταστήσουμε μια πραγματική μηχανή με μια εικονική μηχανή. Τα containers βρίσκουν επί του παρόντος εκτεταμένη χρήση σε πλατφόρμες νέφους (Cloud) καθώς αποτελούν μία από τις δυο κύριες μεθόδους φιλοξενίας πλατφορμών ως υπηρεσία Platform as a Service (PaaS), ενώ η άλλη είναι η κοινή εικονικοποίηση του λειτουργικού συστήματος (OS). Η τεχνολογία των containers παρουσιάστηκε το 1979 από τους [Bukhary Ikhwan]

4.4.2 Παραδοσιακή εικονοποίηση (traditional virtualization) και εικονοποίηση βασισμένη σε containers(Container-based)

Τα containers είναι μια αφαίρεση στο επίπεδο εφαρμογής που πακετάρει τον κώδικα και τις εξαρτήσεις μαζί. Πολλαπλά containers μπορούν να λειτουργούν στον ίδιο υπολογιστή και να μοιράζονται τον πυρήνα του λειτουργικού συστήματος με άλλα containers, καθένα από τα οποία λειτουργεί ως μεμονωμένη διεργασία στο χώρο του χρήστη (user space). Τα containers καταλαμβάνουν λιγότερο χώρο από τα VMs (οι εικόνες container έχουν συνήθως δεκάδες MB μέγεθος). Οι εικονικές μηχανές είναι μια περίληψη φυσικού υλικού που μετατρέπει έναν διακομιστή σε πολλούς διακομιστές. Ο hypervisor επιτρέπει την εκτέλεση πολλών εικονικών μηχανών σε ένα μόνο μηχάνημα. Κάθε VM περιλαμβάνει ένα πλήρες αντίγραφο ενός λειτουργικού συστήματος, την εφαρμογή, τα απαραίτητα δυαδικά αρχεία και βιβλιοθήκες - που καταλαμβάνουν δεκάδες GB. Οι εικονικές μηχανές έχουν αρκετά μεγάλο χρόνο εκκίνησης. Το σχήμα εφτά δείχνει τη διαφορά στα γενικά υπολογιστικά κόστη όσον αφορά την παραδοσιακή εικονικοποίηση και την εικονικοποίηση βάσει containers. Η ελαφριά προσέγγιση των containers επιτρέπει στο Host OS να υποστηρίζει εκατοντάδες containers στο ίδιο υλικό. Έτσι, τα containers μπορούν να θεωρηθούν ως μια τυπική μονάδα λογισμικού, ένα πακέτο που ενσωματώνει όλο τον κώδικα και τις εξαρτήσεις σε μια μονάδα που μπορεί να λειτουργήσει σε οποιοδήποτε υπολογιστικό σύστημα, με γρήγορο και αξιόπιστο τρόπο.



Σχήμα 7 : Containers & VMs

4.4.3 Πλεονεκτήματα των Containers

Τα containers είναι απομονωμένες διεργασίες και δεν απαιτούν έναν hardware hypervisor. Αυτό σημαίνει πως τα containers είναι πολύ μικρότερα και απαιτούν πολύ λιγότερους πόρους από μια εικονική μηχανή. Επίσης με την ταχύτητα που μας παρέχει το Docker, τα containers χρειάζονται μερικά δευτερόλεπτα για να ξεκινήσουν από μια εικόνα ενός container σε σύγκριση με τις εικονικές μηχανές που θέλουν συνήθως μερικά λεπτά για την εκκίνηση τους ώστε να είναι έτοιμες για προγραμματισμό. Τέλος ένα από τα σημαντικότερα που πρέπει να αναφέρουμε είναι πως τα containers μπορούν να δοθούν σε πολλά μέλη μιας ομάδας, προσφέροντας φορητότητα, η οποία διευκολύνει την ανάπτυξη της εφαρμογής.

ΚΕΦΑΛΑΙΟ 5

5.1.1 Εισαγωγή στο Flask

Το Flask είναι ένα web micro framework γραμμένο σε γλώσσα Python για την δημιουργία εφαρμογών ιστού. Το Flask αναπτύχθηκε από τον Armin Ronacher και ονομάζεται micro framework της Python επειδή ο βασικός πυρήνας παραμένει απλός αλλά και σημαντικά επεκτάσιμος αφού επιτρέπει την προσθήκη νέων modules, ώστε η λειτουργικότητα και το σύνολο δυνατοτήτων να μπορούν να επεκταθούν σε νέα επίπεδα.Το Flask χρησιμοποιεί το Jinja Template Engine και την Werkzeug εργαλειοθήκη WSGI(Web Server Gateway Interface).

5.1.2 Τα οφέλη του Flask

Σύμφωνα με έρευνα που έγινε από προγραμματιστές στην Python, το Flask χαρακτηρίστηκε ως το πιο δημοφιλές πλαίσιο, με το 47% να δείχνει ιδιαίτερη προτίμηση ως προς αυτό. Μερικά από τα οφέλη που έχει το Flask είναι:

(1) Εύκολη η μάθηση του

Το Flask ως πλαίσιο χρήσης παραμένει απλό και για αυτό το λόγο η μάθηση του παραμένει ευχάριστη από τον πιο έμπειρο προγραμματιστή μέχρι τον πιο αρχάριο. Οφείλουμε να αναφέρουμε πως δεν κατακλύζει τους προγραμματιστές με πολλές και περίπλοκες λεπτομέρειες.

(2) Ιδιαίτερα επεκτάσιμη

Σε περιβάλλοντα που επεκτείνονται με γρήγορους ρυθμούς είναι προτιμότερο να αρχίζουμε από μικρά κομμάτια και να προχωράμε στα μεγαλύτερα, καθώς προχωράει η εφαρμογή μας. Το Flask μπορεί να συμβάλει σημαντικά σε αυτό, καθώς είναι εξαιρετικά επεκτάσιμο και μπορεί να επεξεργάζεται μεγάλο αριθμό αιτημάτων καθώς αυξάνεται η εφαρμογή.

(3) Διευκολύνει τον πειραματισμό

Να προσθέσουμε πως το Flask είναι το ιδανικό πλαίσιο για να πειραματιστεί ένας προγραμματιστής σε εφαρμογές ιστού. Σε σύγκριση με τη μονολιθική δομή ενός πλαισίου όπως το Django, το Flask είναι πολύ πιο προσαρμόσιμο με νέες αναδυόμενες τεχνολογίες στην αγορά. Οι προγραμματιστές μπορούν να βελτιώνουν συνεχώς το προϊόν τους, έτσι το Flask μπορεί να διευκολύνει την ταχύτερη δημιουργία της εφαρμογής. Έτσι, εάν ένας προγραμματιστής επιθυμεί να προσθέσει περισσότερες δυνατότητες για να εμπλουτίσει την εφαρμογή του, μπορεί να χρησιμοποιήσει το Flask για να πετύχει την γρήγορη ενσωμάτωση.

(4) Ευέλικτη διάταξη του έργου

Η ευελιξία είναι το βασικό χαρακτηριστικό του Flask. Είναι ιδιαίτερα χρήσιμο όταν οι προγραμματιστές πρέπει να συνεργάζονται με διαφορετικές ροές εργασίας και συστήματα. Το Flask έχει ένα επεκτάσιμο πλαίσιο ιστού. Δίνει στους προγραμματιστές την ευελιξία να προσαρμόσουν τις εφαρμογές τους με τον τρόπο που θέλουν. Αυτό τους επιτρέπει να μειώσουν τα ζητήματα που θα μπορούσαν να προκύψουν λόγω της ακαμψίας άλλων πλαισίων. Τους απαλλάσσει από τη χρήση συγκεκριμένων τεχνικών στοιχείων. Αντ 'αυτού,

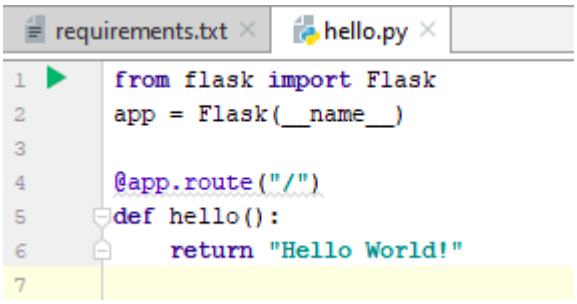
οι προγραμματιστές μπορούν να χρησιμοποιήσουν τα στοιχεία που θέλουν να δημιουργήσουν για τις εφαρμογές ιστού τους. Αυτό το επίπεδο ελευθερίας είναι ένα από τα πιο σημαντικά πλεονεκτήματα του Flask.

(5) Παρέχει καλύτερο έλεγχο στους κώδικες και στις επεκτάσεις

Λαμβάνοντας υπόψη ότι η βάση κώδικα είναι μικρή, οι προγραμματιστές έχουν μεγαλύτερο έλεγχο στους κώδικες τους. Ομοίως, οι προγραμματιστές προτιμούν το Flask επειδή τους επιτρέπει να επιλέγουν τα στοιχεία που θέλουν. Το Flask δίνει στους προγραμματιστές τη δυνατότητα να λαμβάνουν περισσότερες αποφάσεις επιλέγοντας τα στοιχεία για την εφαρμογή ιστού τους. Έχουν την δυνατότητα επίσης να έχουν τον πλήρη έλεγχο των επεκτάσεων τους. Για παράδειγμα, εάν μια εφαρμογή ιστού δεν απαιτεί πρόσβαση στη βάση δεδομένων ή επικύρωση φόρμας, τότε ο προγραμματιστής μπορεί να επιλέξει να εξαιρέσει αυτές τις επεκτάσεις και αντ' αυτού να χρησιμοποιήσει αυτές που σχετίζονται με την εφαρμογή.

5.1.3 Παράδειγμα κώδικα

Στο πρώτο μας παράδειγμα θα αναφερθούμε σε μια απλή και συνηθισμένη εκτύπωση ενός προγράμματος για να κατανοήσουμε το τρόπο λειτουργίας του Flask. Αρχικά στην (Σχ. 8) βλέπουμε πώς γίνεται η εισαγωγή του Flask. Αυτή η κλάση αντιπροσωπεύει μια μεμονωμένη WSGI εφαρμογή καθώς είναι και το κεντρικό αντικείμενο σε κάθε έργο Flask. Στην γραμμή δύο (app = Flask(__name__)), δημιουργούμε ένα παράδειγμα εφαρμογής με την μεταβλητή app και της περνάμε το όνομα της ενότητας μας, η μεταβλητή app μπορεί μάλιστα να είναι οτιδήποτε. Στη συνέχεια η επόμενη γραμμή κώδικα (@app.route("/")) προσδιορίζει τις προβολές που θα έχει η σελίδα, τις διαδρομές των συνδέσμων URL, τη διαμόρφωση του προτύπου και άλλα ακόμα.
Τέλος βλέπουμε μια απλή συνάρτηση που επιστρέφει το γνωστό μήνυμα "Hello World!".



Σχήμα 8 : Παράδειγμα εφαρμογής Flask

5.2: Blueprints

Μπορούμε να οργανώσουμε τις εφαρμογές του FLask μέσω μιας ενσωματωμένης έννοιας που ονομάζονται blueprints, τα οποία είναι ουσιαστικά ισοδύναμα με τα Python modules. Τα blueprints προορίζονται να εγκλείσουν τμήματα χαρακτηριστικών της εφαρμογής μας, όπως οι ροές οθόνης, το AUTH του χρήστη, τα προφίλ χρηστών κ.λπ. Τα blueprints διατηρούν τη σχετική λογική και τα στοιχεία ομαδοποιημένα και χωρισμένα μεταξύ τους, η οποία είναι απαραίτητη για τον σχεδιασμό ενός συντηρητικού έργου. Ενεργοποιούν επίσης τα οφέλη απόδοσης που σχετίζονται με τον διαχωρισμό κώδικα

5.2.1 Bluprints & Views

Η λειτουργία προβολής είναι ο κώδικας που γράφουμε για να ανταποκριθούν τα αιτήματα στην εφαρμογή μας. Το Flask χρησιμοποιεί μοτίβα για να ταιριάξει τη διεύθυνση URL του εισερχόμενου αιτήματος με την αναλυτική προβολή που πρέπει χειριστεί. Η προβολή επιστρέφει δεδομένα τα οποία το Flask τα μετατρέπει σε εξερχόμενη απόκριση. Το Flask έχει την δυνατότητα επίσης να στραφεί κι ως προς την αντίθετη κατεύθυνση και να δημιουργήσει μια διεύθυνση URL σε μια προβολή με βάση το όνομα και τα ορίσματά της.

5.2.2 Δημιουργία ενός blueprint

Ένα blueprint είναι ένας τρόπος οργάνωσης μιας ομάδας σχετικών προβολών και κώδικα. Αντί να γίνει η εγγραφή των προβολών και του κώδικα απευθείας σε μια εφαρμογή, έχουν καταχωριστεί με ένα blueprint. Στη συνέχεια, το blueprint καταγράφεται στην εφαρμογή όταν είναι διαθέσιμο στο μοτίβο Factory του Flask.

5.2.3 Αρχιτεκτονική εφαρμογής με blueprints

Όπως προαναφερθήκαμε προηγουμένως τα σχεδιαγράμματα του Flask μας επιτρέπουν να οργανώσουμε τις εφαρμογές μας ομαδοποιώντας τη λογική σε υποκαταλόγους. Για να το απεικονίσουμε αυτό, θα δημιουργήσουμε ένα παράδειγμα εφαρμογής με τρία σχεδιαγράμματα: home, profile και products (Σχ. 12).

Flask structure with Blueprints

```
/flask-blueprint-tutorial
├── /flask_blueprint_tutorial
│   ├── __init__.py
│   ├── assets.py
│   ├── api.py
│   ├── /home
│   │   ├── /templates
│   │   ├── /static
│   │   └── routes.py
│   ├── /profile
│   │   ├── /templates
│   │   ├── /static
│   │   └── routes.py
│   ├── /products
│   │   ├── /templates
│   │   ├── /static
│   │   └── routes.py
│   ├── /static
│   └── /templates
├── README.md
├── config.py
├── requirements.txt
└── wsgi.py
```

Σχήμα 12 : Δομή με blueprints

Ένα πολύ σημαντικό πλεονέκτημα των blueprints είναι η δυνατότητα διαχωρισμού των προτύπων σελίδας και των στατικών στοιχείων σε συγκεκριμένους καταλόγους templates και / static . Τώρα η σελίδα προορισμού μας δεν θα βρεθεί να φορτώνει άσχετο CSS, το οποίο σχετίζεται με σελίδες προϊόντων ή και το αντίστροφο. Ένα σημαντικό πράγμα που πρέπει να επισημάνουμε είναι η παρουσία κορυφαίων επιπέδων / templates και / static *εκτός* από τα αντίστοιχα συγκεκριμένα blueprints. Παρόλο που τα blueprints δεν μπορούν να έχουν πρόσβαση στα πρότυπα ή στατικά αρχεία των συνομηλίκων τους, μπορούν *να* χρησιμοποιούν κοινά στοιχεία για κοινή χρήση σε όλα τα blueprints (όπως ένα πρότυπο layout.html ή ένα γενικό αρχείο style.css που ισχύει για όλα τα μέρη της εφαρμογής).

5.3 Jinja

5.3.1 Πρότυπα – Templates

Το κλειδί για την σωστή συντήρηση μιας εφαρμογής ή μιας ιστοσελίδας αντίστοιχα είναι η συγγραφή καθαρού και καλά δομημένου κώδικα. Χρησιμοποιώντας τα πρότυπα μπορούμε να ορίσουμε την βασική διάταξη για τις σελίδες που θα χρησιμοποιήσουμε και να αναφερθούμε ποιο στοιχείο θα είναι αυτό που θα αλλαχτεί. Με αυτόν τον τρόπο μπορούμε να ορίσουμε την επικεφαλίδα της σελίδας μας μία φορά και να τη διατηρήσουμε σε όλες τις σελίδες του, ακόμα και εάν πρέπει να αλλάξουμε την επικεφαλίδα, θα πρέπει να ενημερωθεί μόνο σε ένα μέρος, δηλαδή εκεί που την έχουμε ορίσει. Η χρήση προτύπου - template μας εξοικονομεί πολύ χρόνο κατά τη δημιουργία της εφαρμογής, αλλά και κατά την ενημέρωση και συντήρησή της.

5.3.2 Τι είναι το Jinja

Το Jinja είναι μια μηχανή προτύπου ιστού για τη γλώσσα προγραμματισμού Python. Δημιουργήθηκε από τον Armin Ronacher και έχει πάρει άδεια από την BSD License. Το Jinja είναι παρόμοιο με τη μηχανή προτύπου Django, αλλά παρέχει εκφράσεις (expresssions) τύπου Python διασφαλίζοντας παράλληλα ότι τα πρότυπα αξιολογούνται σε ένα sandbox Είναι μια γλώσσα προτύπου ιστοσελίδων σε μορφή κειμένου και έτσι μπορεί να χρησιμοποιηθεί για τη δημιουργία οποιασδήποτε σήμανσης, καθώς και πηγαίου κώδικα. Η μηχανή προτύπου Jinja επιτρέπει την προσαρμογή ετικετών, φίλτρων, δοκιμών και σφαιρικών (globals). Επίσης, σε αντίθεση με τη μηχανή προτύπων Django, η Jinja επιτρέπει στον σχεδιαστή προτύπων να καλεί συναρτήσεις με ορίσματα σε αντικείμενα. Το Jinja είναι η προεπιλεγμένη μηχανή προτύπου του Flask.

5.3.3 Εισαγωγή στο Jinja

Το Jinja είναι μια εκφραστική, γρήγορη, επεκτάσιμη μηχανή για templating. Ειδικά σύμβολα κράτησης θέσης στο πρότυπο επιτρέπουν τη σύνταξη κώδικα παρόμοιου με τη σύνταξη Python. Στη συνέχεια, το πρότυπο διαβιβάζει δεδομένα για την απόδοση του τελικού εγγράφου.
Αυτό περιλαμβάνει:

- Το πρότυπο κληρονομιάς και συμπερίληψης.
- Ορίζει και εισάγει macros εντός προτύπων.
- Τα πρότυπα HTML μπορούν να χρησιμοποιούν αυτόματη διαμόρφωση για να αποτρέψουν το Cross site scripting (XSS) από την μη αξιόπιστη είσοδο από τον χρήστη.
- Ένα sandboxed περιβάλλον μπορεί να αποδώσει με ασφάλεια σε μη αξιόπιστα πρότυπα.
- Υποστήριξη Async για τη δημιουργία προτύπων που χειρίζονται αυτόματα τις λειτουργίες συγχρονισμού χωρίς κάποια επιπλέον σύνταξη.
- Υποστήριξη I18N με τον Babel.
- Τα πρότυπα μεταγλωττίζονται για να βελτιστοποιήσουν τον κώδικα Python just-in-time και να αποθηκευτούν προσωρινά ή για να μπορούν να μεταγλωττιστούν εκ των προτέρων.

- Οι εξαιρέσεις δείχνουν τη σωστή γραμμή στα πρότυπα για να διευκολύνουν τον εντοπισμό σφαλμάτων.
- Επεκτάσιμα φίλτρα, δοκιμές, λειτουργίες ακόμη και σύνταξη.

Η φιλοσοφία της Jinja είναι ότι ενώ η λογική της εφαρμογής ανήκει στην Python, εάν είναι δυνατόν, δεν θα πρέπει να κάνει τη δουλειά του σχεδιαστή προτύπου δύσκολη περιορίζοντας πάρα πολύ τη λειτουργικότητα.

ΚΕΦΑΛΑΙΟ 6: Secure Shell (SSH)

6.1 Εισαγωγή στο Secure Shell (SSH)

Το SSH, είναι επίσης γνωστό ως Secure Shell ή Secure Socket Shell, είναι ένα πρωτόκολλο δικτύου που παρέχει στους χρήστες, ιδιαίτερα στους διαχειριστές συστήματος, έναν ασφαλή τρόπο πρόσβασης σε έναν υπολογιστή μέσω ενός μη ασφαλούς δικτύου. Το SSH αναφέρεται στη σουίτα βοηθητικών προγραμμάτων – εργαλείων που εφαρμόζουν το πρωτόκολλο SSH. Το Secure Shell παρέχει ισχυρό έλεγχο ταυτότητας κωδικού πρόσβασης και έλεγχο ταυτότητας του δημόσιου κλειδιού, καθώς και της κρυπτογραφημένης επικοινωνίας των δεδομένων μεταξύ δύο υπολογιστών που συνδέονται μέσω ανοικτού δικτύου, όπως το Διαδίκτυο. Εκτός από την παροχή ισχυρής κρυπτογράφησης, το SSH χρησιμοποιείται ευρέως από διαχειριστές δικτύου για τη διαχείριση συστημάτων και εφαρμογών από απόσταση, επιτρέποντάς τους να συνδεθούν σε άλλον υπολογιστή μέσω δικτύου, να εκτελέσουν εντολές και να μετακινήσουν αρχεία από έναν υπολογιστή σε άλλο. Το SSH χρησιμοποιεί το μοντέλο πελάτη - διακομιστή, συνδέοντας μια εφαρμογή πελάτη Secure Shell, η οποία είναι το άκρο όπου εμφανίζεται η περίοδος σύνδεσης, με έναν διακομιστή SSH, ο οποίος είναι το άκρο όπου εκτελείται η περίοδος λειτουργίας. Οι εφαρμογές SSH περιλαμβάνουν συχνά υποστήριξη για πρωτόκολλα εφαρμογών που χρησιμοποιούνται για εξομοίωση μέσω του τερματικού ή για μεταφορά αρχείων.

6.2 Χρησιμότητα

Το SSH χρησιμοποιείται συνήθως για την σύνδεση σε απομακρυσμένα μηχανήματα και για την εκτέλεση εντολών, αλλά υποστηρίζεται επίσης στις σήραγγες (tunnels), για προώθηση των θυρών TCP και για τις συνδέσεις X11. Το πρόγραμμα πελάτη (client) SSH χρησιμοποιείται συνήθως για τη δημιουργία συνδέσεων σε έναν daemon SSH που δέχεται απομακρυσμένες συνδέσεις. Συνήθως και τα δύο παρουσιάζονται στα πιο κοινά λειτουργικά συστήματα όπως (Windows, Linux, MacOS). Το SSH είναι σημαντικό στο cloud computing τόσο για την επίλυση των προβλημάτων της συνδεσιμότητας που πιθανόν μπορεί να παρουσιαστούν, όσο και για την αποφυγή σχετικά με τα ζητήματα ασφάλειας της έκθεσης μιας εικονικής μηχανής που βασίζεται στο cloud, η οποία συνδέεται απευθείας στο Διαδίκτυο. Μια σήραγγα SSH μπορεί να παρέχει μια ασφαλή διαδρομή μέσω του Διαδικτύου, δηλαδή μέσω ενός τείχους προστασίας σε μια εικονική μηχανή. Τέλος, το SSH μπορεί επίσης να εκτελεστεί χρησιμοποιώντας SCTP αντί για TCP ως πρωτόκολλο σύνδεσης.

6.3 Εισαγωγή στο docker.sock

Το docker.sock είναι ένα UNIX socket που ακούει ο docker daemon. Είναι το κύριο σημείο εισόδου για το Docker API. Μπορεί να υπάρχουν διαφορετικοί λόγοι για τους οποίους μπορεί να χρειαστούμε να τοποθετήσουμε το docker socket μέσα σε ένα container. Όπως όταν σηκώνουμε νέα containers μέσα από ένα άλλο container ή ακόμα για σκοπούς αυτόματης ανακάλυψης και καταγραφής υπηρεσιών. Αυτό αυξάνει το κενό για επιθέσεις, οπότε θα πρέπει να είμαστε προσεκτικοί εάν κάνουμε mount το docker.socket μέσα σε ένα container, υπάρχουν αξιόπιστοι

κώδικες που εκτελούνται μέσα σε αυτό το container, αντιθέτως είναι πολύ πιθανόν να θέσουμε σε κίνδυνο τον κεντρικό υπολογιστή μας, δηλαδή αυτόν που εκτελεί το docker daemon, ενώ το docker εκκινεί από προεπιλογή όλα τα κοντέινερ ως root.

6.4 Οι κίνδυνοι του docker.sock

Το docker.sock θεωρείται πως μπορεί να αυξήσει τον κίνδυνο επίθεσης σε ένα container και έτσι να θέσει σε κίνδυνο και τον κεντρικό υπολογιστή. Εάν ένας μη εξουσιοδοτημένος χρήστης μπορεί αποκτήσει πρόσβαση σε ένα container που εκτελεί το docker.sock, με αυτό τον τρόπο θα αποκτήσει δικαιώματα πρόσβασης root στον κεντρικό υπολογιστή αλλά και στο Docker. Ένας χρήστης root μπορεί να κάνει όποιες αλλαγές θέλει πάνω στον κεντρικό υπολογιστή. Επιπλέον, επειδή το docker.sock επιτρέπει στους χρήστες να ξεκινήσουν, να σταματήσουν και να χειριστούν με άλλο τρόπο τα containers, ένας μη εξουσιοδοτημένος χρήστης θα μπορούσε να εκτελέσει οποιαδήποτε εντολή Docker. Τέλος, εάν αυτός ο χρήστης έχει πρόσβαση στην υποδοχή Docker, μπορεί να ακούσει ή να συνδεθεί με το API.

6.5 Χρησιμοποίηση του docker.sock με ασφάλεια

Για να πετύχουμε την μέγιστη ασφάλεια και να αποφύγουμε όλους τους κινδύνους ασφαλείας καλό θα είναι να μην χρησιμοποιήσουμε καθόλου το docker.sock. Αντιθέτως αν κάποια εταιρεία, οργανισμός ή κάποια ομάδα προγραμματιστών πρέπει να χρησιμοποιεί το docker.sock, θα πρέπει να χρησιμοποιήσει καλές και σωστές τεχνικές για να πετύχει με τον καλύτερο δυνατόν τρόπο αυτό που επιθυμεί. Πριν αρχίσουμε να τοποθετούμε το docker.sock στα containers μας, πρέπει να επανεξετάσουμε προσεκτικά τις πληροφορίες που θα μοιραστούν τα container μεταξύ τους - όσο πιο ευαίσθητες είναι οι πληροφορίες στην εικόνα του Docker, τόσο μεγαλύτερος είναι ο κίνδυνος που πιθανόν θα αντιμετωπίσουμε στο μέλλον. Μια καλή τεχνική είναι αρχικά να παραχωρήσουμε στο container πρόσβαση μόνο στις πληροφορίες που χρειάζεται όταν χρησιμοποιείται το docker.sock. Τουλάχιστον, μπορούμε να ορίσουμε τις πληροφορίες σε ένα container μόνο με δικαιώματα ανάγνωσης (read – only), καθώς αυτό θα μας αποτρέψει απο το να δώσουμε σε έναν μη εξουσιοδοτημένο χρήστη το δικαίωμα να αλλάξει οποιοδήποτε περιεχόμενο μέσα στα containers μας. Τέλος πρέπει να αναφέρουμε πως αυτό δεν θα εμπόδισει τον χρήστη να αποκτήσει πρόσβαση root στον κεντρικό υπολογιστή. Για να περιορίσουμε αυτές τις λειτουργίες από το χρήστη, είναι πολύ σημαντικό να ενεργοποιήσουμε το Transport Layer Security ή τα authorization proxies (τους διακομιστές μεσολάβησης εξουσιοδότησης). Το Docker μπορεί επίσης να αποκλείσει την πρόσβαση σε ένα socket μέσω του HTTP και να παρέχει πρόσθετες δυνατότητες ελέγχου ταυτότητας και εξουσιοδότησης μέσω διαμορφώσεων του role - based access control. Επιπροσθέτως, μπορούμε να χρησιμοποιήσουμε το Docker bench for security, ένα αυτοματοποιημένο σενάριο ελέγχου από την Docker Inc., το οποίο ελέγχει την ανάπτυξη σε σχέση με τα σημεία αναφοράς του Center for Internet Security και τις βέλτιστες πρακτικές σχετικά με την ανάπτυξη container.

ΚΕΦΑΛΑΙΟ 7: Pipe

7.1 Εισαγωγή στο Pipe

Ο Σωλήνας (pipe) είναι μια προγραμματιστική τεχνική που χρησιμοποιείται ώστε να πετύχουμε την επικοινωνία μεταξύ των διεργασιών. Η βασική του λειτουργία αναφέρεται στον μηχανισμό του, ο οποίος παίρνει την έξοδο (output) από τη μια από τις διεργασίες που εκτελούνται και την στέλνει ως είσοδο (input) σε μια άλλη διεργασία. Αυτό έχει ως αποτέλεσμα να υπάρχει μια συγκεκριμένη ροή δεδομένων μεταξύ δύο σχετικών διεργασιών. Μάλιστα στον σωλήνα (pipe) κάποιος μπορεί να έχει πρόσβαση μέσω ένος συνηθισμένου αρχείου και το σύστημα μας να το διαχειρίζεται όπως την FIFO queue. Δημιουργείται ένα αρχείο σωλήνα (pipe) χρησιμοποιώντας την κλήση συστήματος σωλήνων. Ένας σωλήνας έχει ένα άκρο εισόδου και ένα άκρο εξόδου. Κάποιος έχει την δυνατότητα να μπορεί να γράψει σε ένα σωλήνα από το άκρο εισόδου και να το διαβάσει από το άκρο εξόδου του. Το pipe desciptor, έχει έναν πίνακα που αποθηκεύει δύο δείκτες, ένας δείκτης είναι για το άκρο εισόδου του και ο άλλος δείκτης είναι για το άκρο εξόδου του.

7.2 Παραδειγμα του Pipe

Έστω ότι έχουμε δύο διεργασίες την Α και την Β, οι οποίες προσπαθούν να επικοινωνήσουν μεταξύ τους. Είναι πολύ σημαντικό να οριστεί το ποια διεργασία είναι αυτή που έχει το άκρο εισόδου και ποια το άλλο άκρο εξόδου, το οποίο είναι και το τέλος του σωλήνα (pipe).

1) Η διεργασία Α πρέπει να διατηρεί ανοιχτό το άκρο της εγγραφής και να κλείνει το άκρο του σωλήνα ανάγνωσης.

2) Η διαδικασία Β πρέπει να διατηρεί ανοιχτό το τέλος ανάγνωσης και να κλείνει το τέλος εγγραφής. Όταν ο σωλήνας δημιουργηθεί, έχει καταλάβει ένα σταθερό μέγεθος στην μνήμη.

Όταν μια διεργασία επιχειρεί να γράψει μέσα στον σωλήνα, το αίτημα εγγραφής εκτελείται άμεσα αν ο σωλήνας δεν είναι γεμάτος. Ωστόσο, εάν ο σωλήνας είναι πλήρης, η διεργασία δεν εκτελείται μέχρις ότου να αλλάξει η κατάσταση του σωλήνα. Παρομοίως, μια διαδικασία ανάγνωσης είναι αποκλεισμένη, εάν προσπαθεί να διαβάσει περισσότερα bytes από αυτά που ήδη βρίσκονται εκείνη τη στιγμή στον σωλήνα, διαφορετικά η διαδικασία ανάγνωσης εκτελείται. Τέλος μόνο μία διαδικασία μπορεί να έχει πρόσβαση σε έναν σωλήνα κάθε φορά.

7.2.1 Περιορισμοί:

Το κανάλι επικοινωνίας ενός σωλήνα λειτουργεί μόνο ως προς μία κατεύθυνση. Οι σωλήνες δεν μπορούν να υποστηρίξουν τη μετάδοση, δηλαδή την αποστολή μηνυμάτων σε πολλές διαδικασίες ταυτόχρονα.
Το άκρο ανάγνωσης ενός σωλήνα διαβάζει με οποιονδήποτε τρόπο. Δεν έχει σημασία ποια διαδικασία συνδέεται με το άκρο εγγραφής του σωλήνα. Επομένως, αυτός είναι πολύ ανασφαλής τρόπος επικοινωνίας. Ορισμένα κλεισίματα των άκρων απαιτούν τη σωστή δημιουργία ενός κατευθυνόμενου σωλήνα.

7.3 Η λειτουργία του Pipe

Μέσω ενός bash script το αρχείο pipe ρυθμίζεται, έτσι ώστε να περιμένει την λήψη μηνυμάτων σε μορφή string και να τα εκτελεί σαν εντολές μέσω της ειδικής εντολής eval. Το directory του pipe συνδέεται με το container που ελέγχει μέσω του bind mounting (volume) συνεπώς μπορεί να δεχτεί μηνύματα από το container και να τα εκτελέσει στο επίπεδο που είναι ο host. Το container που ελέγχει ρυθμίζεται έτσι ώστε να μπορεί να "σπρώξει" τα επιθυμητά μηνύματα – εντολές στο pipe μέσω της εντολής echo.

7.4 Διαφορές μεταξύ ssh & pipe

Η εγκατάσταση του ssh απαιτεί τόσο στο container που ελέγχει όσο και στα containers που θα ελέγχονται να γίνεται 150MB ανά container installation. Συνεπώς η εφαρμογή γίνεται πολύ βαριά και δημιουργούνται καθυστερήσεις κατά την εκκίνηση του προγράμματος μας. Επίσης απαιτεί ένα set up στο root password του κάθε ελεγχόμενου container ή τουλάχιστον την δημιουργία super user (το ssh pass απαιτεί την γνώση του password ώστε να συνδεθεί στο κάθε container). Μάλιστα μπορεί να επικοινωνήσει με άλλα containers μόνο αν γνωρίζει την ip τους, η οποία σε κάθε εκκίνηση μπορεί να αλλάζει (δεν μπορεί να κανει resolve naming) εκτός αν έχουμε δημιουργήσει custom docker network (προς το παρόν η εφαρμογή μας δουλεύει με το default bridge network). Αφήνει περιθώριο για security issues αφού θεωρητικά έχει την δυνατότητα να συνδεθεί με ssh μέχρι και στον host, ενώ όλα τα passwords αναγράφονται είτε σε plain text είτε σε unencrypted files. Χρησιμοποιώντας το pipe επιτυγχάνουμε να μην απαιτείται η εγκατάσταση οποιασδήποτε βιβλιοθήκης ή άλλης εγκατάστασης στα containers, έτσι γίνεται πιο ελαφριά η εφαρμογή μας. Μάλιστα εκμεταλλεύεται την "ανώτερη" εντολή docker exec του host η οποία μπορεί να επικοινωνήσει με οποιοδήποτε container και μάλιστα αναφερομένη σε αυτό με το όνομα του αντί για την ip η οποία αλλάζει κάθε φορά.

Μερικά από τα μειονεκτήματα που έχει το pipe είναι οτι:

1) Απαιτεί την εκκίνηση του pipe πριν απο το docker-compose up. καθώς και η εκκίνηση του χρειάζεται 2 εντολές αντί για μια, δηλαδή, το docker-compose up.

2) Αφήνει ένα σημαντικό περιθώριο για security issues αφού αποτελεί ένα μόνιμο backdoor σε terminal του host. Τα πιθανά security issues του pipe δεν είναι απαραίτητα περισσότερα από αυτά του ssh pass ειδικά από την στιγμή που δεν είναι δυνατόν κάποιος να βάλει sudo password μέσω αυτού.

7.5 Τελικό συμπέρασμα

Με βάση αυτά που αναφέρθηκαν παραπάνω θεωρούμε πως το pipe αποτελεί μια πολύ καλή επιλογή στο εν λόγω πρόγραμμα που προσπαθούμε να αναπτύξουμε καθώς και μια πολύ καλή λύση που συμβάλει στην επικοινωνία μεταξύ των container που έχουμε.

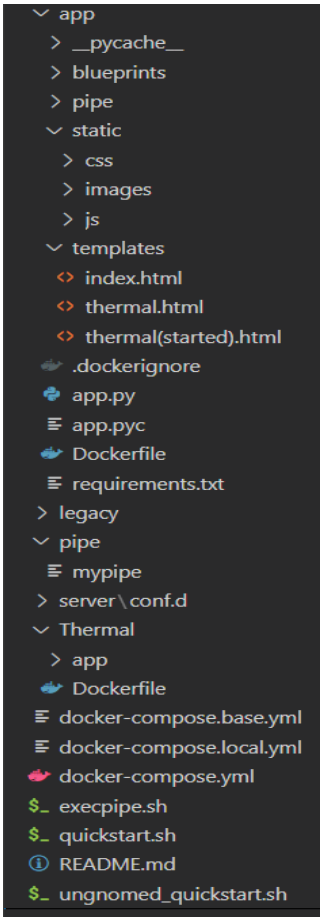
ΚΕΦΑΛΑΙΟ 8 : Αναλύοντας την εφαρμογή

8.1 Ο σκοπός και τα πρώτα βήματα στην εφαρμογής μας

Όπως αναφέραμε και στην εισαγωγή μας, η υλοποίηση της εφαρμογής έχει στόχο την δημιουργία μίας πλατφόρμας από την οποία θα συνδέονται συσκευές για ιατρικούς σκοπούς, στην προκειμένη περίπτωση, η συσκευή μας θα είναι μια θερμική κάμερα που θερμομετρά ασθενείς και η εφαρμογή μας θα συλλέγει και θα αποθηκεύει τις θερμοκρασίες. Με την δημιουργία της εφαρμογής θα χρειαστούμε να εγκαταστήσουμε στο Linux την python3, το Flask και το Docker και το sublime (κειμενογράφος) για την επεξεργασία του κώδικα μας.

8.2 Αρχιτεκτονική της εφαρμογής

Αρχικά βλέπουμε την αρχιτεκτονική της εφαρμογής μας και παρακάτω θα αναλύσουμε τα πιο βασικά αρχεία της εφαρμογής καθώς και τον τρόπο λειτουργίας τους. Βλέπουμε πως μέσα στον φάκελο app βρίσκονται κάποια αρχεία όπως και κάποιοι φάκελοι, ανάμεσα σε αυτούς βρίσκεται η εφαρμογή μας (app.py) καθώς και οι φάκελοι που περιλαμβάνουν τα αρχεία για την ιστοσελίδα που σηκώνουμε . Τα html τα βλέπουμε στο φάκελο templates και τα css, images αλλά και την javascript στο φάκελο static. Στη συνέχεια βλέπουμε το Dockerfile καθώς και το αρχείο requirments.txt. Μέσα στο φάκελο legacy βλέπουμε το αρχείο extractTemps. Με το τέλος της εκτέλεσης του προγράμματος μας, οι θερμοκρασίες αποθηκεύονται στον host, στο path Thermal/app/data.



Σχήμα 13 : Αρχιτεκτονική της εφαρμογής

8.3 Λειτουργία της εφαρμογής

Στο κύριο directory της εφαρμογής κάνουμε δεξί κλικ και open in terminal, στη συνέχεια στο terminal εκτελούμε την εντολή ./quickstart.sh . Έπειτα στην οθόνη εμφανίζονται δύο terminals ένα για το pipe και ένα για το docker-compose up. Στο broswer ανοίγω την ιστοσελίδα στην διεύθυνση 0.0.0.0 και κάνω κλικ στο thermal (το terminal του docker όπου καταγράφει τα GET requests) .
Χωρίς να συμπληρώσω κάποιο interval (στον χρόνο) αφού ο default χρόνος είναι 15sec (για το app.py) πατάω το κουμπί start. Το terminal του docker καταγράφει το request και το terminal του pipe κάνει execute το extractTemps (στο τεστ που κάναμε έκανε συνολικα δυο φορές αφού το αφήσαμε ανοιχτό για 30 δευτερόλεπτα). Μετά την δεύτερη επανάληψη πατήσαμε το κουμπί “στοπ”, το terminal του docker καταγράφει το request και το terminal του pipe σταματάει να αντιδράει και περιμένει για περιαιρετό εντολές. Στα τελευταία δευτερόλεπτα του βίντεο γίνεται μεγέθυνση στο παράθυρο του pipe όπου φαίνεται καθαρά ότι το extractTemps έτρεξε μόνο δυο φορές (όπως ήταν και το επιθυμητό στο συγκεκριμένο παράδειγμα).

8.4 Επεξήγηση του κώδικα

Στην πρώτη γραμμή του κώδικα της εφαρμογής μας γίνονται import καποιες μεθόδοι από το Flask. Στη συνέχεια κάνουμε import το subproceess το οποίο είναι για την εκτέλεση των εντολών σε επίπεδο container. Το import time το χρειαζόμαστε για το time sleep
Κάνουμε import το μοντέλο του Thread για να μπορούμε να κάνουμε το multiThreadig.
Στην πέμπτη γραμμή του κώδικα γίνεται initialize ένα αντικείμενο, το οποίο είναι το Flask application μας.
Στην έκτη γραμμή του κώδικα έχουμε τις εντολές " pipe_command = "echo 'docker exec thermalcamera ./extractTemps' > ./pipe/mypipe"
pipe_empty = "echo " > ./pipe/mypipe" " που θα σταλούν στο pipe για να εκτελεστούν σε επίπεδο host σε συνεργασία με ένα συγκεκριμένο container (thermal).

Στην γραμμή έντεκα δημιουργούμε μια κλάση η οποία αναλαμβάνει τις επαναλήψεις Στη συνέχεια ορίζω μια μεταβλητή boolean με το όνομα loop, η οποία μας ορίζει αν θα έχουμε άλλες επαναλήψεις ή όχι.
Στην δεκάτη τρίτη γραμμή έχουμε την interv μεταβλητή που κρατάει το χρόνο και αρχίζει με 15 secs, επίσης ορίζει το ιντερβαλ των επαναλήψεων.
Στην γραμμή δεκατέσσερα ορίζουμε μια συνάρτηση που αναλαμβάνει το ξεκίνημα των επαναλήψεων.
try:
Οτιδήποτε υπάρχει κάτω από το try προσπαθεί το πρόγραμμα να το εκτελέσει αλλιώς περνάει στην συνθήκη except.
Μάλιστα μετατρέπει την είσοδο του χρήστη σε ακέραιο αριθμό γιατί δίνεται από τον χρήστη ως string και επίσης ελέγχει αν είναι αρνητικός αριθμός και αν όντως εκπληρώνει την συνθήκη στην οποία είναι αρνητικός αριθμός τότε περνάει στην except όπου επιστρέφει το default χρόνο ο οποίος είναι τα δεκαπέντε δευτερόλεπτα. Στην γραμμή εικοσιδύο σιγουρεύουμε ότι η παράμετρος loop της κλάσης είναι True για να γίνουν οι επαναλήψεις, Ωστόσο αν πατήσω το κουμπί stop αυτό θα γίνει False με αποτέλεσμα στη συνέχεια να μην μπορέσουμε να πατήσουμε το start για να ξαναγίνει η επανάληψη με επιτυχία. Στην γραμμή εικοσιτρία όσο είναι true.loop στέλνουμε την εντολή για να τραβηχτούν οι θερμοκρασίες και ύστερα για να αδειάσουμε το pipe για να μην τραβηχτούν παραπάνω θερμοκρασίες από όσες θέλουμε και τέλος περιμένουμε τον χρόνο για την επομένη επανάληψη. Στην γραμμή εικοσιοκτώ η συνάρτηση σταματάει τις επαναλήψεις, μετατρέπει την μεταβλητή σε False και αδειάζει το pipe σε περίπτωση που δεν έχει προλάβει να γίνει στην πιο πάνω επανάληψη. Στην γραμμή τριάντα δύο ενεργοποιούμε την κλάση loopobject στο instance ob. Στην γραμμή τριάντα τέσσερα δηλώνουμε σε ποιο directory της εφαρμογής μας αντιστοιχεί ο παρακάτω κώδικας. Μια συνάρτηση με το όνομα helloword το μόνο που κάνει είναι να επιστρέφει την εισαγωγική σελίδα index.html. Γραμμή τριάντα εννιά δηλώνουμε σε ποιο directory της εφαρμογής μας αντιστοιχεί ο παρακάτω κώδικας (start button) και ορίζουμε την μέθοδο εκτέλεσης ότι θα είναι [POST]. Στη συνέχεια ορίζουμε την συνάρτηση startthread(), Ορίζουμε ένα thread object το οποίο το "δένουμε" με την μέθοδο start loop και από την ιστοσελίδα παίρνει τα δεόμενα που έχει συμπληρώσει ο χρήστης και τα αποθηκεύει σε μια τοπική μεταβλητή με το όνομα interv. Η μεταβλητή interv παίρνει την τιμή που έδωσε ο χρήστης πριν τις επαναλήψεις Στη συνέχεια ξεκιναει το thread και συνεπώς οι επαναλήψεις και τέλος επιστρέφει την ιστοσελίδα "thermal started".

Γραμμή σαράντα εννιά δηλώνουμε σε ποιο directory της εφαρμογής μας αντιστοιχεί ο παρακάτω κώδικας (stop button) και ορίζουμε την μέθοδο εκτέλεσης ότι θα είναι [POST].
Στη συνέχεια ορίζουμε την συνάρτηση thread(), Ορίζουμε ένα thread object το οποίο το "δένουμε" με την μέθοδο stop loop και ενεργοποιώ το thread και κατά επέκταση σταματάω τις επαναλήψεις και τέλος γυρίζουμε στην αρχική σελίδα (index.html).

Γραμμή εξήντα ένα είναι ένας τυπικός έλεγχος ο οποίος θεωρείται καλή προγγραμματιστική τεχνική πριν την εκκίνηση μιας εφαρμογής – ενός προγράμματος.
Τέλος γινεται η εκκίνηση της εφαρμογής στο 0.0.0.0 και εκεί είναι που πραγματικό ξεκινάει το πρόγραμμα μας.

Παρακάτω βλέπουμε τον κώδικα.

```
from flask import Flask
from blueprints.thermal_blueprint import thermal_web
from blueprints.index_blueprint import start_page

app = Flask(__name__)
app.register_blueprint(start_page)
app.register_blueprint(thermal_web)

if __name__ == '__main__':
    app.run(host="0.0.0.0", debug=True)
```

8.5 Απαραίτητες διευκρινήσεις ως προς τις λειτουργίες

Dockerfile

Στην πρώτη γραμμή έχουμε την python σαν default image. Στην γραμμή δύο αντιγράφει το requirments.txt αρχείο του host στο container. Στην τρίτη γραμμή το run εκτελεί αυτή την εντολή, διαβάζει το αρχείο requirments.txt για να δει τι libraries χρειαζόμαστε και τα εγκαταστεί μέσω του pip, το οποίο pip εγκαθιστά libraries. Στην τέταρτη γραμμή έχουμε το VOLUME το οποίο είναι το αντίστοιχο mkdir, δηλαδή να δημιουργήσουμε τον φάκελο pipe και τον φάκελο blueprints.
Να αναφέρουμε πως όταν δημιουργήσουμε αυτούς τους φακέλους αρχικά είναι άδειοι αλλά μετά λόγο του bind mounting αυτοί γεμίζουν με τα ίδια αρχεία που έχουν οι αντίστοιχοι φάκελοι του host.

Κώδικας

```
FROM python
COPY requirements.txt .
RUN pip install -r ./requirements.txt
VOLUME /pipe
```

VOLUME /blueprints

Docker-compose.base.yml

Στη πρώτη γραμμή του κώδικα γίνεται η τυπική ονομασία του service (web) Παρακάτω η εντολή restart σημαίνει πως αν κλείσει ή κρασάρει κάνει restart από την αρχή. Στην τρίτη γραμμή ψάχνει τον φάκελο app και κάνει build με το Docker file που θα βρει. Στην τέταρτη γραμμή ο φάκελος app του container ορίζεται ως το working directory – default path. Στην πέμπτη γραμμή τα έχουμε τα volumes. Αντιστοιχώ έναν φάκελο του host σε ένα φάκελο του container (η αντιστοίχηση είναι το bind mounting) Ο φάκελος app του host αντιστοιχίζεται στο φάκελο app του container. Αντίστοιχα Ο φάκελος pipe του host αντιστοιχίζεται στο φάκελο app/pipe του container
Ο φάκελος app/blueprints του host αντιστοιχίζεται στο φάκελο app/blueprints του container

Κώδικας

```
web:
  restart: always
  build: ./app
  working_dir: /app
  volumes:
    - ./app:/app
    - ./pipe:/app/pipe
    - ./app/blueprints:/app/blueprints
```

Docker.compose.local.yml

Στη πρώτη γραμμή ονομάζουμε το service web. Στη συνέχεια στη δεύτερη γραμμή γίνεται το extends, δηλαδή επεκτείνει τις ιδιότητες ενός άλλου service στις γραμμές πέντε και εφτά. Στη γραμμή τρία ορίζεται το αρχείο το οποίο πρέπει να το ψάξει στο docker.compose.base.yml και στην από κάτω γραμμή το όνομα του service που θα επεκταθεί είναι το “web”.
Στη γραμμή πέντε “ports” αντιστοιχίζεται το port 80 του host στο port 5000 του container (ο λόγος είναι οτι το port 80 χρησιμοποιείται πάντα από το host για http services και συνεπώς μας επιτρέπει να χειριζόμαστε containers από έναν web broswer στο επίπεδο του host, υπάρχει δηλαδή αντιστοίχηση). Τέλος στην γραμμή εφτά ορίζουμε ποιο θα είναι το default command στο terminal του container. (Είναι αυτό που θα εκτελεστεί πρώτα αφού δημιουργηθεί το container).

Κώδικας

```
web:
  extends:
    file: docker-compose.base.yml
    service: web
  ports:
    - "80:5000"
  command: python app.py
```

Docker-compose.yml

Αρχικά στο αρχείο docker-compose.yml βλέπουμε τα τρία cointainers μας (Το πρώτο για το nginx, το άλλο για το flask και τέλος το thermal). Στο Container του thermal θα αναλύσουμε το τι κάνουν οι παρακάτω εντολές.
Στην αρχή παίρνει τυπικά ένα όνομα, στην συγκεκριμένη περίπτωση το container_name είναι nginx. Το restart βοηθά στο αν βγεί κάποιο σφάλμα και δεν σηκωθεί η εφαρμογη τότε γίνεται rebuild. Στη γραμμή τέσσερα γίνεται η τυπική ονομασία. Στη γραμμή πέντε έχουμε τα volumes. Ο φάκελος server/conf.d του host αντιστοιχίζεται στο φάκελο etc/nginx/conf.d του container. Ο φάκελος app/static του host αντιστοιχίζεται στο φάκελο app/static του container. (προκειμένου να κάνουμε live αλλαγές χωρίς να σκοτώνουμε και να ξανα σηκώνουμε τα containers). Στη γραμμή οχτώ το web service γίνεται αντιστοίχηση του host σε κάποιο φάκελο του container. Στη γραμμή εννια έχουμε τα ports http services για το container nginx (Η αντιστοίχηση πάει και από τις δύο μεριές έτσι ώστε ο broswer να διαβάζει τις νέες πληροφορίες που έρχονται και απο τα δύο containers ταυτόχρονα) -λειτουργούν αλληλοεξαρτώμενα -
Παρακάτω έχουμε το web Στη γραμμή δέκα τέσσερα έχουμε την τυπική ονομασία, στην κάτω γραμμή κάνει extends στο ίδιο service την εντολή expose kai command. Η εντολή file δηλώνει σε ποιο αρχείο service θα κοιτάξει και από κάτω έχουμε το όνομα του service που θα κάνει extend. Στη γραμμή δέκα οχτώ, η εντολή expose, ανοίγει το port 8000 και το καθιστά διαθέσιμο για οποιαδήποτε χρήση και τέλος στην γραμμή είκοσι με την δημιουργία του container θα εκτελεστεί η συγκεκριμένη εντολή, όπου δημιουργεί ένα instance του gunicorn app.
Στο Thermal
Έχουμε αρχικά την τυπική ονομασία στη συνέχεια όπως είπαμε παραπάνω το
restart βοηθά στο αν βγεί κάποιο σφάλμα και δεν σηκωθεί η εφαρμογη τότε γίνεται rebuild. Το container name δηλώνει με ποιο container επικοινωνεί Το build δηλώνει από ποιο directory τραβάει το image που θα κανει build μέσω του dockerfile (local directory). Το tty είναι true για να παραμένει πάντα ενεργό και να μην κάνει extract code 0, ακόμα και αν φαινομενικά δεν έχει κάποια δουλεία να κάνει. Τα Volumes είναι υπεύθυνα για την σύνδεση των directory του host με τα directory των container.

Στο server ολα τα αρχεία ανταποκρίνονται

Μάλιστα το gunicornapp αναλαμβάνει το τρόπο επικοινωνίας και λειτουργίας του flask και του application μας, συνεπώς αναλαμβανει το εκτελεστικό κομμάτι.

Κώδικας

```
server:
  container_name: nginxserver
  restart: always
  image: nginx
  volumes:
    - ./server/conf.d:/etc/nginx/conf.d
    - ./app/static:/app/static
  links:
    - web:web
  ports:
    - "80:80"
```

```
web:
  container_name: gunicornapp
  extends:
    file: docker-compose.base.yml
    service: web
  expose:
    - "8000"
  command: gunicorn app:app -b :8000 --name app --log-level=debug --log-file=-

thermal:
  restart: always
  container_name: thermalcamera
  build: ./Thermal
  tty: true
  ports:
    - "1567:1567"
  volumes:
    - ./Thermal/app/data:/data
```

Execpipe.sh

Το execpipe.sh εκτελεί μια επαναληπτική διαδικασία η οποία περιμένει μήνυμα απο το pipe file και το εκτελεί σαν μια εντολή με το eval.

Κώδικας

```
#!/bin/bash

echo "This is the PIPE Terminal - Do not close!"
while true; do eval "$(cat ./pipe/mypipe)"; done
```

Quickstart.sh

Αρχικά αυτό το αρχείο εμφανίζει το μήνυμα "Initializing Pipe" με σκοπό να ενημερώσει τον χρήστη. Στη συνέχεια ανοίγει ένα νέο terminal και εκτελεί την εντολή "./execpipe.sh". Πρέπει να είναι σε διαφορετικά terminals το pipe με το terminal του docker compose .-Εκτυπώνει στο terminal (τερματικό) το μήνυμα “done” και το “Initializing docker Multi – container Application ” . Και τέλος εκτελεί και την εντολή για να σηκωθεί ο server μαζί με την σελίδα μας.

```
#!/bin/bash
#A quickstarting bashscript for our application
#requirements: gnome-terminal

echo "Initializing Pipe"
gnome-terminal -- "./execpipe.sh"
echo "done"
echo "Initializing Docker Multi-Container Application"
```

docker-compose up

[Σχήμα 17] : code

Ως προς το pipe πρέπει να τονίσουμε πως το execpipe.sh πρέπει να ξεκινάει σε ένα ξεχωριστό terminal με την εκκίνηση της εφαρμογής μας (γίνεται αυτόματα μέσω της εντολής ./quickstart.sh) και να κλείνει χειροκίνητα με το κλείσιμο της εφαρμογής μας. Παρόλο που βρίσκεται σε infinite loop (While True) δεν περνάει από την μια επανάληψη στην άλλη αν δεν λάβει κάποια εντολή, συνεπώς δεν εκτελεί παλιές εντολές επ άπειρον αλλά κάθε φορά περιμένει για μια νέα εντολή.
Τέλος να αναφέρουμε πως δεν έχει καμιά λειτουργική σχέση με το interval των θερμοκρασιών ή το stop loop, διότι αυτα ελέγχονται αποκλειστικά από το app.py .

Ο ρόλος του είναι να δέχεται οποιαδήποτε εντολή και να την εκτελεί στο επίπεδο του

host, συνεπώς σε κάποιο ενδεχόμενο app scaling, θα μπορούσε πολύ εύκολα να εξυπηρετεί και άλλα containers με διαφορετικές εντολές.
[Σχήμα 18] : code

ungnomed_quickstart.sh

Κώδικας

```
#!/bin/bash

#A quickstarting bashscript for our application

#Does not require Gnome Terminal


echo "Initializing Pipe"

./execpipe.sh &

echo "done"

echo "Initializing Docker Multi-Container Application"

docker-compose up
```

Χρήση Jinja & blueprints στην εφαρμογή μας

Η index σελίδα μας περιέχει όλο το βασικό template της σελίδας, καθώς κάνουμε πλοήγηση από σελίδα σε σελίδα το μόνο που θέλουμε να αλλάζει είναι ένα block κώδικα στη μέση, το οποίο ανάλογα τη σελίδα έχει και συγκεκριμένη λειτουργία.

Index

Στο block της σελίδας index έχουμε το κουμπί “Go to Thermal”. Ως στόχος είναι σε μελλοντικό up-scaling να επεκταθεί σε λίστα με κουμπιά που το καθένα θα παραπέμπει σε ένα διαφορετικό microservice.

Thermal html

Το block είναι ένα text input για να βάζει ο χρήστης το interval και το κουμπί start, για λόγους αισθητικής έχουμε και το κουμπί stop που είναι disable.

Thermal started html

Το block έχει ένα μήνυμα για να ενημερώνει το χρήστη ότι το microservice ξεκίνησε να λειτουργεί και το κουμπί stop, για λόγους αισθητικής έχουμε το κουμπί start και το text input για το interval που είναι disabled.

Η λειτουργία τους

Στην εφαρμογή μας στη σελίδα index ορίζουμε το block κώδικα που θέλουμε να αλλάζει από σελίδα σε σελίδα και του δίνουμε ένα όνομα, του δώσαμε το όνομα midmodule.
Στις σελίδες Thermal & Thermal started αρχικά χρησιμοποιώ την εντολή “index.extends” για να ορίσουμε ότι κατά το rendering της σελίδας αυτής θα χρησιμοποιηθεί ως βασικό template το index.html και ο υπόλοιπος κώδικας θα προστεθεί επί αυτόν. Όμως επειδή παρακάτω ορίζουμε ένα block με το ίδιο όνομα (midmodule) αλλά διαφορετικό κώδικα το αποτέλεσμα είναι κατά το rendering να γίνεται over ride του αρχικού block με αυτό που δώσαμε.

Extra note

Από την στιγμή που χρησιμοποιούμε την εντολή render template του flask δεν απαιτείται να κάνουμε import στον κώδικα μας το Jinja2, ο λόγος είναι οτι το Jinja2 αποτελεί dependency του render_template για το λόγο αυτό το μόνο που χρειάστηκε να κάνουμε ήταν να ορίσουμε τα απαραίτητα blocks στα html αρχεία μας.

Index_blueprint

Μετά την εφαρμογή των blueprints ο κώδικας μας άλλαξε λίγο. Αρχικά έχουμε το blueprint για το index και το κρατάμε σε ένα ξεχωριστό blueprint για τη μελλοντική αναβάθμιση της εφαρμογής μας. Βλέπουμε πως στη πρώτη γραμμη του κώδικα το render_tempalate επιστρέφει την σελίδα index.html. Στη συνέχεια κάνουμε το tempalate, object blueprint. Στη γραμμή πέντε δηλώνουμε ένα instance το οποίο το ορίζουμε start_page (‘start page’, __name__) (το __name__δηλώνει το path και θα ανταποκρίνεται στην λειτουργία του index page) . Συνεπώς ο παρακάτω κώδικας δένεται με το blueprint start page.

Κώδικας

```
from flask import Blueprint, render_template

#Blueprint for Index Page

start_page = Blueprint('start_page', __name__)
```

```
@start_page.route('/')
def hello_world():
    return render_template('index.html')
```

Thermal_blueprint

Εδώ επίσης έχουμε το blueprint για το thermal και όπως προαναφερθήκαμε παραπάνω το κρατάμε σε ένα ξεχωριστό blueprint για την μελλοντική επέκταση της εφαρμογής μας.. Κάνοντας τα ίδια import. Στη γραμμή τριάντα τέσσερα δηλώνω ένα καινούργιο instance του blueprint object το οποίο θα αντιστοιχήσουμε στις σελίδες που ελέγχουν το container thermal camera/
Στη γραμμή τριάντα έξι στο συγκεκριμένο url (start button) δένουμε τα παρακάτω fuctions. Αυτά τα functions λειτουργούν με POST methods.
Στην γραμμή σαράντα έξι αντίστοιχα το συγκεκριμένο url (stop button) δένουμε τα παρακάτω fuctions.
Το τελευταίο function είναι αυτό που πηγαίνει στην αρχική σελίδα.

Κώδικας

```
from flask import Blueprint, render_template, request
import subprocess
import time
from threading import Thread

pipe_command = "echo 'docker exec thermalcamera ./extractTemps' > ./pipe/mypipe"
pipe_empty = "echo " > ./pipe/mypipe"

class loopobject():
    loop = True
    interv = 15
    def startloop(self):
        try:
            self.interv = int(self.interv) #if user did not enter number we go to except
            if self.interv < 0: #if user typed a number but not a valid one
                raise Exception() #we still go to except
        except:
            self.interv = 15 #except returns to the default 15 sec

        self.loop = True
        while self.loop:
            subprocess.Popen(pipe_command,shell=True)
            subprocess.Popen(pipe_empty,shell=True)
            time.sleep(self.interv)
```

```
    def stoploop(self):
        self.loop = False
        subprocess.Popen(pipe_empty,shell=True)

ob = loopobject()

#Blueprint for Thermal App Control

thermal_web = Blueprint('thermal_web',__name__)

@thermal_web.route('/ThermalCamera', methods=['POST'])
def create_container():
    return render_template('thermal.html')

@thermal_web.route('/startbutton', methods=['POST'])
def startthread():
    t1 = Thread(target = ob.startloop)
    #print("start button pressed")
    interval = request.form['interval']
    #print("you selected an interval of:",interval)
    ob.interv = interval
    t1.start()
    return render_template('thermal(started).html')

@thermal_web.route('/stopbutton', methods=['POST'])
def stopthread():
    t2 = Thread(target = ob.stoploop)
    #print("stop button pressed!")
    t2.start()
    return render_template('index.html')
```

App.py

Στη συνέχεια θα αναλύσουμε τον κώδικα του αρχείου app.py .
Βλέπουμε πως στην πρώτη γραμμή κάνουμε import μόνο το flask object καθώς όλα τα υπόλοιπα modules θα κληθούν αυτόματα όταν κάνουμε register τα blueprints, εκεί έχουμε ένα από τα πλεονεκτήματα των blueprints, δηλαδή να κρατήσουμε και τον κώδικα μας πιο καθαρό. Στην παρακάτω γραμμή κάνουμε import τα blueprints (να αναφέρουμε πως το from blueprints.thermal_blueprint έχει γραφτεί με αυτό το τρόπο διότι οι “ . “ έχουν το ρόλο των “ / “ για να πηγαίνουν πιο βαθιά σε ένα path). Στη συνέχεια στη γραμμή πέντε ορίζω το flask application ως ένα flask object και συνεχίζοντας παρακάτω δένουμε τα απαραίτητα blueprints τα οποία καθορίζουν την συμπεριφορά και την λειτουργία του application μας. Τέλος στην τελευταία γραμμή είναι που ξεκινάει να η εφαρμογή μας να τρέχει.

Κώδικας

```
from flask import Flask
from blueprints.thermal_blueprint import thermal_web
from blueprints.index_blueprint import start_page

app = Flask(__name__)
app.register_blueprint(start_page)
app.register_blueprint(thermal_web)

if __name__ == '__main__':
    app.run(host="0.0.0.0", debug=True)
```

Βιβλιογραφία

1. Bitsadmin. “Here Is Why We Are Big Fans of Python’s Flask Framework.” *Benchmark IT Solutions*, 2 June 2020, www.benchmarkit.solutions/blog/here-is-why-we-are-big-fans-of-pythons-flask-framework.

2. Coursera. “What Are Microservices?” *Coursera*, 2021, www.coursera.org/lecture/applications-development-microservices-serverless-openshift/what-are-microservices-k5dLG.

3. DelVecchio, Alex. “IoMT (Internet of Medical Things) or Healthcare IoT.” *IoT Agenda*, 31 Aug. 2015, www.internetofthingsagenda.techtarget.com/definition/IoMT-Internet-of-Medical-Things

4. Educative. “The 7 Most Important Software Design Patterns.” *Educative: Interactive Courses for Software Developers*, 7 Nov. 2018, www.educative.io/blog/the-7-most-important-software-design-patterns.

5. Fawcett, Amanda. “Microservices Architecture Tutorial: All You Need to Get Started.” *Educative: Interactive Courses for Software Developers*, 17 Mar. 2020, www.educative.io/blog/microservices-architecture-tutorial-all-you-need-to-get-started.

6. Fuzon. “Internet of Medical Things (IoMT): New Era in Healthcare Industry.” *Fuzon*, 19 Dec. 2019, www.fuzon.io/insight/iomt-in-healthcare-industry.

7. GeeksforGeeks. “Python | Introduction to Web Development Using Flask.” *GeeksforGeeks*, 18 May 2020, www.geeksforgeeks.org/python-introduction-to-web-development-using-flask.

8. Wilson, Cameron. “How to Design a Web Application: Software Architecture 101.” *Educative: Interactive Courses for Software Developers*, 4 Feb. 2020, www.educative.io/blog/how-to-design-a-web-application-software-architecture-101.

9. Schlosser, H. (2020, January 30). *Docker vs. Virtual Machine: Where are the differences?* DevOps Conference. https://devopscon.io/blog/docker/docker-vs-virtual-machine-where-are-the-differences/

10. Loshin, P., & Cobb, M. (2020, February 14). *Secure Shell (SSH)*. SearchSecurity. https://searchsecurity.techtarget.com/definition/Secure-Shell

11. Wikipedia contributors. (2021, June 21). *Secure Shell Protocol*. Wikipedia. https://en.wikipedia.org/wiki/Secure_Shell_Protocol

12. GeeksforGeeks. (2020, May 14). *IPC technique PIPES*. https://www.geeksforgeeks.org/ipc-technique-pipes/

13. Birchard, T. (2020, December 9). *Organizing Flask Apps with Blueprints*. Hackers and Slackers. https://hackersandslackers.com/flask-blueprints/

14. *Blueprints and Views — Flask Documentation (2.0.x)*. (2021, July 5). Blueprints and Views — Flask Documentation. https://flask.palletsprojects.com/en/2.0.x/tutorial/views/

15. *Can anyone explain docker.sock*. (2016, January 31). Stack Overflow. https://stackoverflow.com/questions/35110146/can-anyone-explain-docker-sock

16. Extreme Marketing Team. (2021, April 6). *The Top 3 IoMT Challenges Keeping Healthcare IT Up at Night*. Extreme Networks. https://www.extremenetworks.com/extreme-networks-blog/the-top-3-iomt-challenges-keeping-healthcare-it-up-at-night/

17. *Introduction — Jinja Documentation (3.0.x)*. (2021, July 5). Introduction — Jinja Documentation (3.0.x). https://jinja.palletsprojects.com/en/3.0.x/intro/#dependencies

18. *Modular Applications with Blueprints — Flask Documentation (2.0.x)*. (2021, July 4). Modular Applications with Blueprints. https://flask.palletsprojects.com/en/2.0.x/blueprints/

19. Wikipedia contributors. (2021, June 28). *Jinja (template engine)*. Wikipedia. https://en.wikipedia.org/wiki/Jinja_(template_engine)

20. Gillis, A. S. (2019, January 25). *Get informed of the risks associated with docker.sock*. SearchITOperations. https://searchitoperations.techtarget.com/tip/Get-informed-of-the-risks-associated-with-dockersock

21. Shah, H. (2020, December 23). *Best of 2020: When To Use - and Not To Use - Microservices*. Container Journal. https://containerjournal.com/topics/container-ecosystems/when-to-use-and-not-to-use-microservices/

22. *Docker Containers vs. Virtual Machines*. (2021, March 7). Aqua. https://www.aquasec.com/cloud-native-academy/docker-container/docker-containers-vs-virtual-machines/

23. Sethi, P. (2017, January 26). *Internet of Things: Architectures, Protocols, and Applications*. Internet of Things: Architectures, Protocols, and Applications. https://www.hindawi.com/journals/jece/2017/9324035/

24. O. Lamtzidis, “An IoT Edge-as-a-service (Eaas) Distributed Architecture & Reference Implementation,” University of Patras, Department of Electrical Engineering and Computer Technology, 2019

Available: https://nemertes.lis.upatras.gr/jspui/bitstream/10889/12992/6/Lamtzidis%28ele%29.pdf

25. Grinberg, Miguel. *Flask Web Development: Developing Web Applications with Python*. 1st ed., O’Reilly Media, 2014.
26. “Introduction to Flask — Python for You and Me 0.4.Beta1 Documentation.” *Introduction ToFlask*, 2008, pymbook.readthedocs.io/en/latest/flask.html

27. “Learning Flask Framework.” *Google Books*, 2015, books.google.gr/books?hl=el&lr=&id=HPGoCwAAQBAJ&oi=fnd&pg=PP1&dq=flask+framework+&ots=A__2fouL_2&sig=nGoPU4MvHGNQwILaznuE_NxpPys&redir_esc=y#v=onepage&q=flask%20framework&f=false.

28. Novoseltseva, Ekaterina. “Top 10 Benefits of Docker.” *Dzone.Com*, 27 Apr. 2017, dzone.com/articles/top-10-benefits-of-using-docker.

29. Wikipedia contributors. “Διαδίκτυο των πραγμάτων.” *Βικιπαίδεια*, 13 Oct. 2020, el.wikipedia.org/wiki/%CE%94%CE%B9%CE%B1%CE%B4%CE%AF%CE%BA%CF%84%CF%85%CE%BF_%CF%84%CF%89%CE%BD_%CF%80%CF%81%CE%B1%CE%B3%CE%BC%CE%AC%CF%84%CF%89%CE%BD

30. “Efficient Way of Web Development Using Python and Flask.” *CASE STUDY AND REPORT*, core.ac.uk/download/pdf/55305148.pdf