ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΕΛΟΠΟΝΝΗΣΟΥ
UNIVERSITY of the PELOPONNESE

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

"ΕΥΕΛΙΞΙΑ ΣΤΗΝ ΜΕΛΕΤΗ ΚΑΙ ΕΞΕΛΙΞΗ ΜΙΑΣ DATABASE ΜΕ ΤΗΝ ΧΡΗΣΗ MICROSERVICES-DOCKER"

ΛΥΚΟΥΡΓΙΩΤΗΣ ΜΙΧΑΗΛ

Α.Μ 2619

ΥΠΕΥΘΥΝΟΣ ΚΑΘΗΓΗΤΗΣ:

Ιωάννης Τζήμας, Καθηγητής

ΕΠΙΒΛΕΠΩΝ:

Σπυρίδων Ντουρούκης

ΠΑΤΡΑ 2023

ΠΕΡΙΛΗΨΗ

Το Docker (https://docs.docker.com/) παρέχει τη δυνατότητα τρεξίματος εφαρμογών σε ένα container, το οποίο περιέχει όλες τις εξαρτήσεις και τις βιβλιοθήκες που είναι απαραίτητες για το τρέξιμο της εφαρμογής. Με τον τρόπο αυτό, με μια μόνο εντολή μπορούν να ξεκινήσουν όλες οι υπηρεσίες που απαιτούνται για να ξεκινήσει η εφαρμογή.

Στόχος της συγκεκριμένης διπλωματικής είναι αρχικά η μελέτη του Docker, εστιάζοντας στα βασικά του στοιχεία και δυνατότητες. Σε δεύτερη φάση, στόχος της διπλωματικής είναι η υλοποίηση μιας Database σε Docker στην οποία θα καταχωρούνται δεδομένα από ένα API (Application Programming Interface) και όλη αυτή η διαδικασία θα γίνετε τρέχοντας ένα docker-compose script.

ΠΡΟΛΟΓΟΣ

Η παρούσα διπλωματική εργασία αποτελεί τον επίλογο των προπτυχιακών μου σπουδών στο Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών. Το περιεχόμενο της εργασίας σχετίζεται με την μελέτη συστημάτων containerization και την εφαρμογή του Docker καθώς και την δημιουργία μιας database με την χρήση τους. Η εκπόνηση της διπλωματικής αυτής εργασίας αποτελεί το επιστέγασμα των πολύτιμων γνώσεων που είχα την τύχη και τη χαρά να αποκτήσω όλο αυτό το χρονικό διάστημα.

Πριν την παρουσίαση των αποτελεσμάτων της διπλωματικής εργασίας, αισθάνομαι την ανάγκη να ευχαριστήσω θερμά όλους όσους με στήριξαν και μου συμπαραστάθηκαν σε όλη τη διάρκεια των προπτυχιακών μου σπουδών.

Αρχικά, θα ήθελα να ευχαριστήσω τον Καθηγητή Ιωάννη Τζήμα, Καθηγητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, ο οποίος ήταν και ο υπεύθυνος καθηγητής της διπλωματικής μου εργασίας. Θέλω να τον ευχαριστήσω θερμά για τις συμβουλές, την καθοδήγηση του καθώς και για την ευκαιρία που μου έδωσε να ασχοληθώ εκτενώς με αυτό το σύγχρονο και ενδιαφέρον ερευνητικό τομέα.

Επιπλέον, αισθάνομαι την ανάγκη να ευχαριστήσω το Σπυρίδων Ντουρούκη, πρώην φοιτητή του τμήματος μου και πλέον μεταπτυχιακού και διδακτορικού, για την άψογη συνεργασία που είχαμε και την ουσιαστική βοήθεια του στην υλοποίηση της διπλωματικής μου εργασίας.

Κλείνοντας, θα ήθελα να απευθύνω ένα μεγάλο ευχαριστώ στους γονείς μου, Ιωάννη και Βαρβάρα καθώς και στην αδερφή μου, Δήμητρα, για τη ψυχική και υλική στήριξη τους στην προσπάθεια των προπτυχιακών μου σπουδών.

Πάτρα, Φεβρουάριος 2023

Λυκουργιώτης Μιχαήλ

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1 : ΕΙΣΑΓΩΓΗ

Στόχος της παρούσας εργασίας είναι η μελέτη του συστήματος containerization Docker εστιάζοντας στα βασικά στοιχεία και χαρακτηριστικά ώστε να κατανοήσουμε καλύτερα τις δυνατότητες που αυτό μας προσφέρει.

Στην συνέχεια θα δούμε τόσο θεωρητικά όσο και πρακτικά τις κυριότερες διαδικασίες που μας παρέχει το Docker όπως : η εκτέλεση ενός docker image, η δημιουργία πολλαπλών container, η επικοινωνία και σύνδεση των container μεταξύ τους, δημιουργία docker compose file.

Τέλος θα υλοποιηθεί σε Docker μια βάση δεδομένων η οποία θα καταχωρεί data τα οποία θα παίρνει από ένα API. Πρωταρχικός στόχος αυτής της υλοποίησης είναι να μπορεί να γίνετε πιο εύκολα η καταχώρηση δεδομένων σε μια βάση δεδομένων. Θα φροντίσουμε έτσι ώστε όλα τα απαραίτητα εργαλεία για την υλοποίηση του συστήματος να εγκαθίστανται αυτόματα με την χρήση του Docker καθώς και ότι ολόκληρο το σύστημα να μπορεί να εκκινείτε με μόνο μία εντολή.

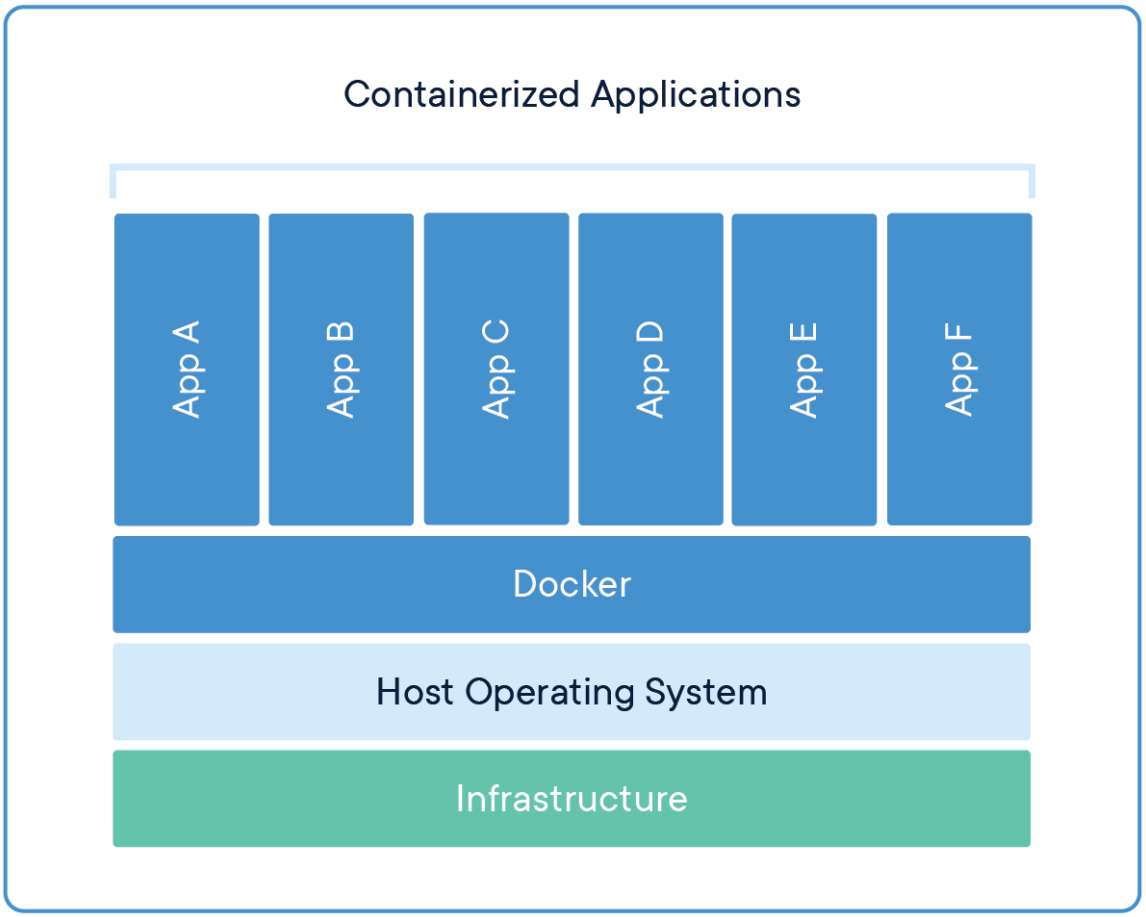
Συγκεκριμένα στα Κεφάλαια 2,3,4,5 γίνεται μια θεωρητική μελέτη σε κάποιες βασικές έννοιες (Container, VM, Docker, Kubernetes) που θα μας χρειαστούν για την επίτευξη του τελικού στόχου οι οποίες θα μας κάνουν να κατανοήσουμε τον λόγο-σκοπό εξέλιξης του Docker.

Στα Κεφάλαια 6,7,8 παρουσιάζεται το πρακτικό κομμάτι της εργασίας στα οποία θα μάθουμε πώς γίνετε η εγκατάσταση του Docker σε διάφορα λειτουργικά συστήματα καθώς και την δημιουργία images, container,docker compose files. Περνώντας στην εγκατάσταση της βάσης δεδομένων και την επικοινωνία της με τα υπόλοιπα container.

ΚΕΦΑΛΑΙΟ 2 : CONTAINER

2.1 ΤΙ ΕΙΝΑΙ ΕΝΑ CONTAINER

Ένα container[1] είναι μια μονάδα λογισμικού που συσκευάζει κώδικα και όλες τις εξαρτήσεις του, ώστε η εφαρμογή να εκτελείται γρήγορα και αξιόπιστα από το ένα υπολογιστικό περιβάλλον στο άλλο. Μια εικόνα container Docker είναι ένα ελαφρύ, αυτόνομο, εκτελέσιμο πακέτο λογισμικού που περιλαμβάνει όλα όσα χρειάζονται για την διαδικασία εκτέλεσης μιας εφαρμογής όπως κώδικα, χρόνο εκτέλεσης, εργαλεία συστήματος, βιβλιοθήκες συστήματος και ρυθμίσεις.



[1] https://www.docker.com/resources/what-container/

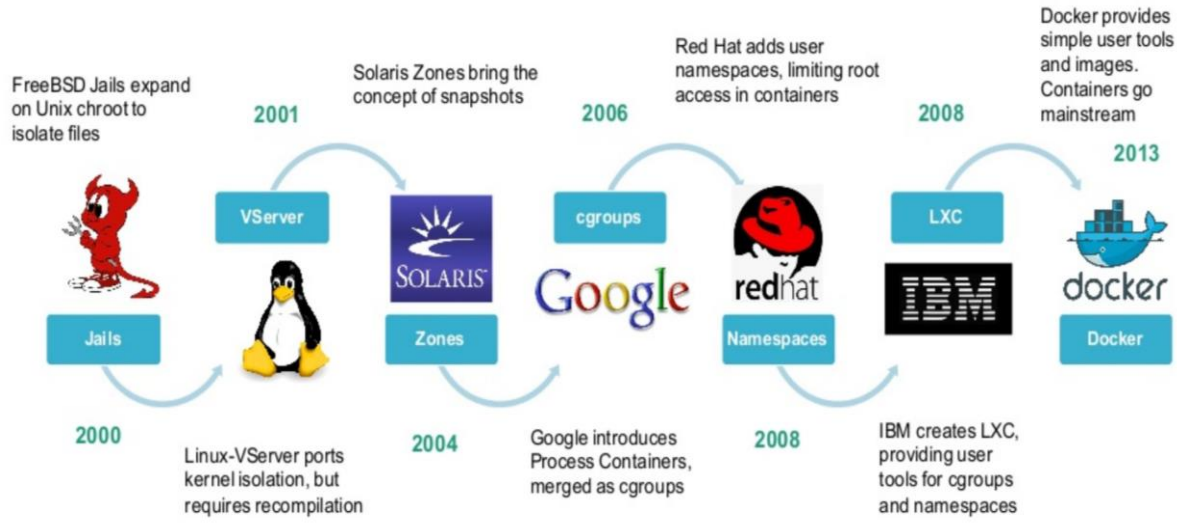
Οι εικόνες container γίνονται container κατά το χρόνο εκτέλεσης και στην περίπτωση των container Docker, οι εικόνες γίνονται container όταν εκτελούνται στο Docker Engine. Το οποίο είναι διαθέσιμο τόσο για εφαρμογές που βασίζονται σε Linux όσο και σε Windows, το λογισμικό με container θα λειτουργεί πάντα το ίδιο, ανεξάρτητα από την υποδομή.

Τα container απομονώνουν το λογισμικό από το περιβάλλον του και διασφαλίζουν ότι λειτουργεί ομοιόμορφα παρά τις διαφορές, για παράδειγμα, μεταξύ ανάπτυξης και σταθεροποίησης. Τα Docker Containers είναι παντού: Linux, Windows, Data Center, Cloud, Server less κλπ.

2.2 ΙΣΤΟΡΙΑ ΠΙΣΩ ΑΠΟ CONTAINER

Η τεχνολογία Docker container κυκλοφόρησε το 2013 ως ανοιχτού κώδικα Docker Engine. Αξιοποίησε υπάρχουσες υπολογιστικές έννοιες γύρω από container και συγκεκριμένα στον κόσμο του Linux, πρωτόγονα γνωστά ως c groups και name spaces.

Η τεχνολογία του Docker είναι μοναδική επειδή εστιάζει στις απαιτήσεις των προγραμματιστών και των χειριστών συστημάτων για διαχωρισμό των εξαρτήσεων εφαρμογών από την υποδομή. Η επιτυχία στον κόσμο του Linux οδήγησε σε μια συνεργασία με τη Microsoft που έφερε τα container Docker και τη λειτουργικότητα του στον Windows Server (μερικές φορές αναφέρονται ως δοχεία Docker Windows). Η τεχνολογία που διατίθεται από το Docker και το έργο ανοιχτού κώδικα, το Moby έχει αξιοποιηθεί από όλους τους μεγάλους προμηθευτές κέντρων δεδομένων και παρόχους cloud. Πολλοί από αυτούς τους παρόχους αξιοποιούν το Docker για τις εγγενείς προσφορές Iaas τους σε container. Επιπλέον, τα κορυφαία πλαίσια ανοιχτού κώδικα χωρίς διακομιστή χρησιμοποιούν την τεχνολογία Docker container.



2.3 ΕΝΟΡΧΗΣΤΡΩΣΗ CONTAINER

2.3.1 ΣΦΑΙΡΙΚΗ ΕΙΚΟΝΑ

Η ενορχήστρωση container ορίζεται ως ο αυτοματισμός της ανάπτυξης, της διαχείρισης, την κλιμάκωσης και της δικτύωσης των container. Οι επιχειρήσεις που πρέπει να αναπτύξουν και να διαχειριστούν εκατοντάδες ή χιλιάδες container και κεντρικούς υπολογιστές Linux μπορούν να επωφεληθούν από την ενορχήστρωση container.

Η ενορχήστρωση container μπορεί να χρησιμοποιηθεί σε οποιοδήποτε περιβάλλον όπου χρησιμοποιείτε container. Μας δίνει την δυνατότητα να αναπτύσουμε την ίδια εφαρμογή σε διαφορετικά περιβάλλοντα χωρίς να χρειάζεται να την επανασχεδιάσουμε. Και οι μικροϋπηρεσίες σε container διευκολύνουν την ενορχήστρωση υπηρεσιών, συμπεριλαμβανομένης της αποθήκευσης, της δικτύωσης και της ασφάλειας.

Τα container δίνουν στις εφαρμογές που βασίζονται σε microservice μια ιδανική μονάδα ανάπτυξης εφαρμογών και αυτόνομο περιβάλλον εκτέλεσης. Καθιστούν δυνατή την εκτέλεση πολλαπλών τμημάτων μιας εφαρμογής ανεξάρτητα σε μικροϋπηρεσίες, στο ίδιο υλικό, με πολύ μεγαλύτερο έλεγχο σε μεμονωμένα κομμάτια και κύκλους ζωής.

Η διαχείριση του κύκλου ζωής των container με ενορχήστρωση υποστηρίζει επίσης ομάδες DevOps που το ενσωματώνουν σε ροές εργασίας CI/CD. Μαζί με τις διεπαφές προγραμματισμού εφαρμογών (API) και τις ομάδες DevOps, οι μικροϋπηρεσίες με container αποτελούν τη βάση για τις εγγενείς εφαρμογές του cloud.

2.3.2 ΣΕ ΤΙ ΧΡΗΣΙΜΕΥΕΙ Η ΕΝΟΡΧΗΣΤΡΩΣΗ CONTAINER

Χρησιμοποιήστε την ενορχήστρωση container[2] για να αυτοματοποιήσετε και να διαχειριστείτε εργασίες όπως :

- Προμήθεια και ανάπτυξη.
- Διαμόρφωση και προγραμματισμός.
- Κατανομή των πόρων.

[2] https://www.redhat.com/en/topics/containers/what-is-container-orchestration

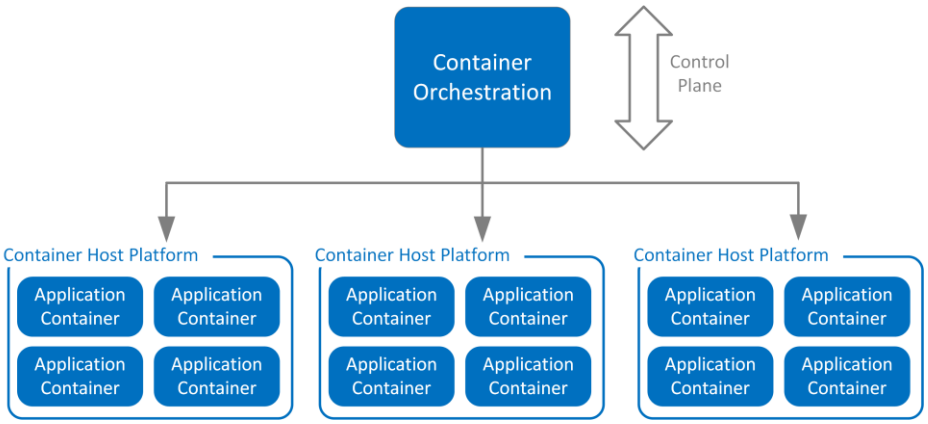
- Διαθεσιμότητα container.
- Κλιμάκωση ή αφαίρεση container με βάση την εξισορρόπηση του φόρτου εργασίας σε όλη την υποδομή σας.
- Εξισορρόπηση φορτίου και δρομολόγηση κυκλοφορίας.
- Παρακολούθηση της υγείας του δοχείου.
- Διαμόρφωση εφαρμογών με βάση το container στο οποίο θα εκτελούνται.
- Διατήρηση ασφαλών αλληλεπιδράσεων μεταξύ δοχείων.

2.3.2 ΠΩΣ ΛΕΙΤΟΥΡΓΕΙ Η ΕΝΟΡΧΗΣΤΡΩΣΗ CONTAINER

Όταν χρησιμοποιείτε ένα εργαλείο ενορχήστρωσης container όπως το Kubernetes, περιγράφετε η διαμόρφωση μιας εφαρμογής χρησιμοποιώντας είτε ένα αρχείο YAML είτε JSON. Το αρχείο διαμόρφωσης λέει στο εργαλείο διαχείρισης διαμόρφωσης που να βρεί τις εικόνες container, πως να δημιουργήσει ένα δίκτυο και που να αποθηκεύσει αρχεία καταγραφής. Κατά την ανάπτυξη ενός νέου container, το εργαλείο διαχείρισης container προγραμματίζει αυτόματα την ανάπτυξη σε ένα σύμπλεγμα και βρίσκει τον κατάλληλο κεντρικό υπολογιστή, λαμβάνοντας υπόψη τυχόν καθορισμένες απαιτήσεις ή περιορισμούς. Στη συνέχεια, το εργαλείο ενορχήστρωσης διαχειρίζεται τον κύκλο ζωής του container με βάση τις προδιαγραφές που καθορίστηκαν στο αρχείο σύνθεσης.

Μπορείτε να χρησιμοποιήσετε μοτίβα Kubernetes για να διαχειριστείτε τη διαμόρφωση, τον κύκλο ζωής και την κλίμακα των εφαρμογών και υπηρεσιών που βασίζονται σε container. Αυτά τα επαναλαμβανόμενα μοτίβα είναι τα εργαλεία που χρειάζονται ένας προγραμματιστής της Kubernetes για την κατασκευή ολοκληρωμένων συστημάτων.

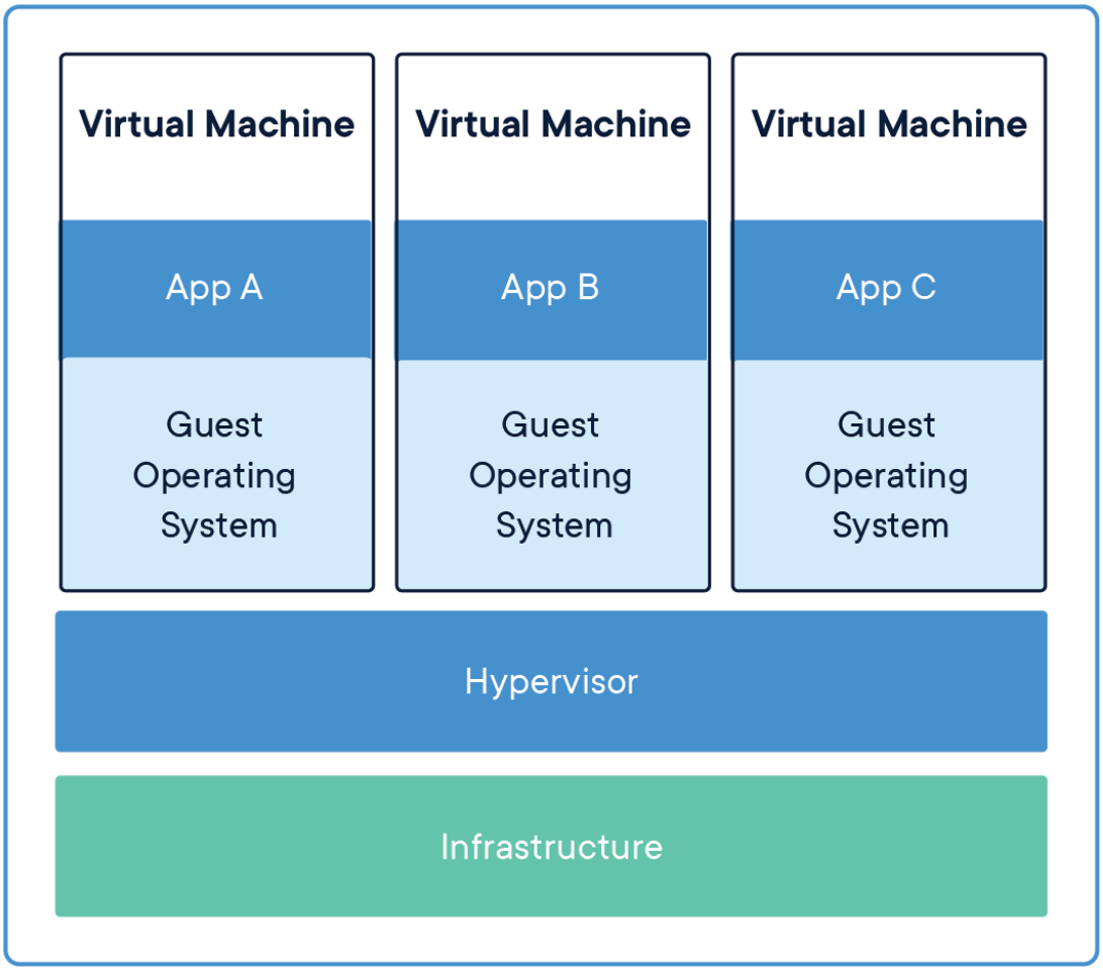
Η ενορχήστρωση container μπορεί να χρησιμοποιηθεί σε οποιοδήποτε περιβάλλον που εκτελεί container, συμπεριλαμβανομένων των διακομιστών εσωτερικής εγκατάστασης και των δημόσιων ή ιδιωτικών περιβαλλόντων cloud.



ΚΕΦΑΛΑΙΟ 3:VIRTUAL MACHINE

3.1 ΤΙ ΕΙΝΑΙ VIRTUAL MACHINE

Μια εικονική μηχανή [3]που συνήθως συντομεύεται σε VM, δεν διαφέρει από οποιονδήποτε άλλο φυσικό υπολογιστή, όπως φορητός υπολογιστής, έξυπνο τηλέφωνο ή διακομιστής. Διαθέτει CPU, μνήμη, δίσκους για την αποθήκευση των αρχείων σας και μπορεί να συνδεθεί στο διαδίκτυο αν χρειαστεί. Ενώ τα μέρη που συνθέτουν τον υπολογιστή σας (που ονομάζονται υλικό) είναι φυσικά και από αυτά τα VM συχνά θεωρούνται ως εικονικοί υπολογιστές ή υπολογιστές που καθορίζονται από λογισμικό εντός φυσικών διακομιστών, που υπάρχουν μόνο ως κώδικας.



[3] https://www.docker.com/resources/what-container/

3.2 ΤΡΟΠΟΙ ΧΡΗΣΗΣ VMs

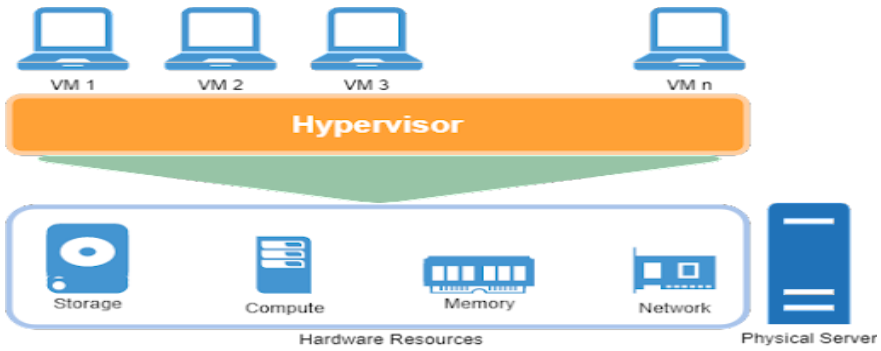
- Δημιουργία και ανάπτυξη εφαρμογών στο cloud. Δοκιμάζοντας ένα νέο λειτουργικό σύστημα (OS), συμπεριλαμβανομένων των εκδόσεων beta.
- Αναπτύσσοντας ένα νέο περιβάλλον για να είναι απλούστερη και ταχύτερη για τους προγραμματιστές η εκτέλεση σεναρίων δοκιμών προγραμματισμού.
- Δημιουργία αντιγράφων ασφαλείας του υπάρχοντος λειτουργικού συστήματος.
- Πρόσβαση σε δεδομένα που έχουν μολυνθεί από ιούς ή εκτέλεση μιας παλιάς εφαρμογής εγκαθιστώντας ένα παλαιότερο λειτουργικό σύστημα.
- Εκτέλεση λογισμικού ή εφαρμογών σε λειτουργικά συστήματα για τα οποία δεν προορίζονταν αρχικά.

3.3 ΟΦΕΛΗ ΤΩΝ VMs

Ενώ οι εικονικές μηχανές λειτουργούν όπως μεμονωμένοι υπολογιστές με μεμονωμένα λειτουργικά συστήματα και εφαρμογές, έχουν το πλεονέκτημα να παραμένουν εντελώς ανεξάρτητες η μία από την άλλη και από τη φυσική μηχανή υποδοχής.

Ένα κομμάτι λογισμικού που ονομάζεται hypervisor ή εικονικός διαχειριστής μηχανών, σας επιτρέπει να εκτελείτε διαφορετικά λειτουργικά συστήματα σε διαφορετικές εικονικές μηχανές ταυτόχρονα. Αυτό καθιστά δυνατή την εκτέλεση εικονικών μηχανών Linux για παράδειγμα, σε λειτουργικό σύστημα Windows ή την εκτέλεση μιας παλαιότερης έκδοσης των Windows σε πιο πρόσφατο λειτουργικό σύστημα Windows.

Και επειδή τα VM είναι ανεξάρτητα το ένα με το άλλο είναι επίσης εξαιρετικά φορητά. Μπορείτε να μετακινήσετε ένα VM σε έναν hypervisor σε έναν άλλον hypervisor σε ένα εντελώς διαφορετικό μηχάνημα σχεδόν αμέσως.



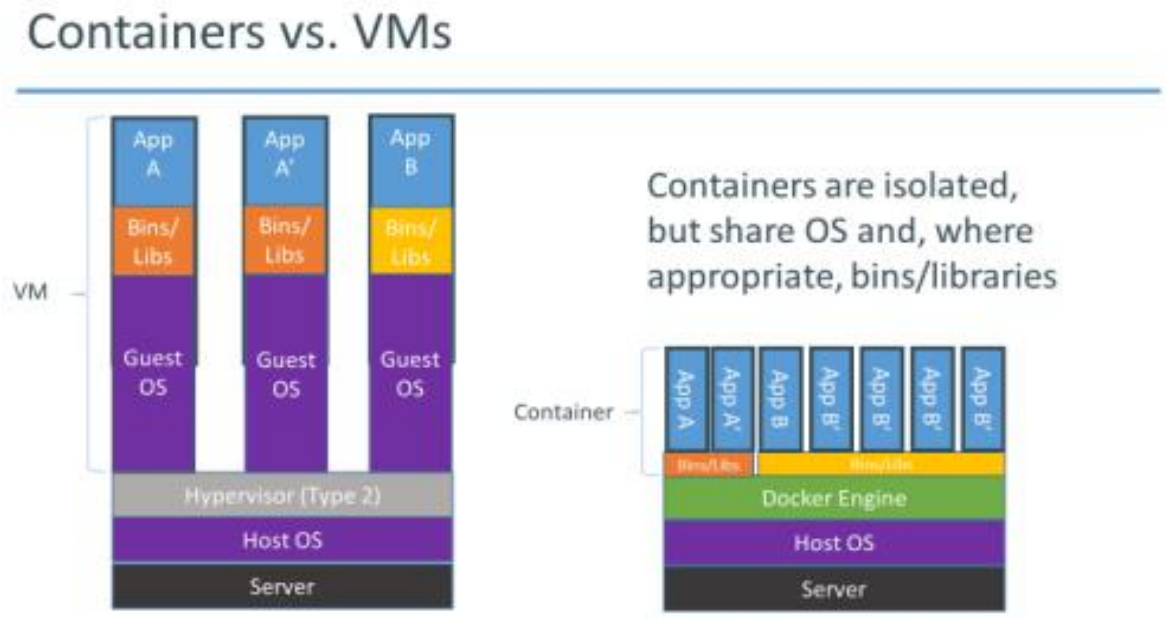
3.4 ΣΥΓΚΡΙΣΗ CONTAINER – VIRTUAL MACHINE

Στον παρακάτω Πίνακα γίνεται μια σύγκριση ανάμεσα σε εικονικές μηχανές και container σε πολλαπλούς παράγοντες όπως η απόδοση, η ασφάλεια, ο χρόνος εκκίνησης και ο χώρος αποθήκευσης.

Παράμετρος	**Virtual Machine**	**Container**
Guest OS	Κάθε εικονική μηχανή τρέχει σε εικονικό υλικό και το kernel φορτώνεται στην δικιά του περιοχή μνήμης.	Όλα τα container μοιράζονται το ίδιο kernel. Το kernel φορτώνεται στην φυσική μνήμη.
Επικοινωνία	Μέσω συσκευών Ethernet.	Στάνταρ IPC μηχανισμοί όπως σήματα, σωληνώσεις (pipes), υποδοχές (sockets) κ.τ.λ.
Ασφάλεια	Ανάλογα με την υλοποίηση του hypervisor.	Ο υποχρεωτικός έλεγχος πρόσβασης μπορεί να χρησιμοποιηθεί
Απόδοση	Οι εικονικές μηχανές υποφέρουν από μια μικρή επιβάρυνση καθώς οι εντολές μηχανής μεταφράζονται από το Guest στο Host λειτουργικό σύστημα.	Τα container έχουν σχεδόν την ίδια απόδοση που έχει το υποκείμενο Host λειτουργικό σύστημα.
Απομόνωση	Ο διαμοιρασμός βιβλιοθηκών, αρχείων, κ.τ.λ. ανάμεσα σε εικονικές μηχανές ή ανάμεσα στην εικονική μηχανή και το host λειτουργικό σύστημα δεν είναι δυνατός.	Τα αρχεία και οι βιβλιοθήκες μπορούν εύκολα να διαμοιραστούν ανάμεσα στα container ή μεταξύ του container και του host λειτουργικού συστήματος.

Χρόνος εκκίνησης	Η εκκίνηση στις εικονικές μηχανές χρειάζεται λίγα λεπτά.	Τα Containers μπορούν να εκκινηθούν σε λίγα δευτερόλεπτα δηλαδή σε σημαντικά λιγότερο σε σχέση με τις εικονικές μηχανές.
Χώρος αποθήκευσης	Οι εικονικές μηχανές χρειάζονται αρκετά περισσότερο αποθηκευτικό χώρο αφού ολόκληρο το kernel του λειτουργικού συστήματος καθώς και τα σχετικά προγράμματα χρειάζονται να εγκατασταθούν.	Τα container χρειάζονται λιγότερο αποθηκευτικό χώρο αφού το βασικό λειτουργικό σύστημα είναι κοινόχρηστο.

Στην Εικόνα μπορούμε να δούμε μερικές από τις σημαντικές διαφορές ανάμεσα σε εικονικές μηχανές και container[4]. Όπως φαίνεται στην εικόνα, το Docker, σε αντίθεση με τις εικονικές μηχανές, δεν χρησιμοποιεί κάποιο hypervisor, κάτι που μειώνει σημαντικά το overhead. Κάθε εικονική μηχανή έχει ένα πλήρες αντίγραφο του λειτουργικού συστήματος καθώς και των βιβλιοθηκών που χρησιμοποιεί. Σε αντίθεση στο Docker το kernel μοιράζεται με το host λειτουργικό σύστημα και τα άλλα container. Επιπροσθέτως τα container μοιράζονται όσες βιβλιοθήκες είναι κοινές. Αυτό έχει ως αποτέλεσμα τα container να έχουν σημαντικά μικρότερο μέγεθος σε σχέση με τις εικονικές μηχανές.



[4] https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm

ΚΕΦΑΛΑΙΟ 4 : DOCKER

4.1 ΤΙ ΕΙΝΑΙ DOCKER

ΣΦΑΙΡΙΚΗ ΕΙΚΟΝΑ

Η λέξη “Docker” [5] (Jarostaw, 2016) αναφέρεται σε πολλά πράγματα συμπεριλαμβανομένου ενός κοινοτικού έργου ανοιχτού κώδικα, εργαλεία από το έργο ανοιχτού κώδικα, Docker Inc, η εταιρεία που υποστηρίζει κατά κύριο λόγο αυτό το έργο και τα εργαλεία που υποστηρίζει επίσημα η εταιρεία. Το γεγονός ότι οι τεχνολογίες και η εταιρεία μοιράζονται το ίδιο όνομα μπορεί να προκαλέσει σύγχυση.

Εδώ είναι μια σύντομη επεξήγηση:

Το λογισμικό πληροφορικής "Docker" (Turnbull, 2014) είναι τεχνολογία container που επιτρέπει τη δημιουργία και χρήση container Linux. Η κοινότητα ανοιχτού κώδικα Docker εργάζεται για να βελτιώσει αυτές τις τεχνολογίες προς όφελος όλων των χρηστών. Η εταιρεία, Docker Inc., βασίζεται στο έργο της κοινότητας Docker, την κάνει πιο ασφαλή και μοιράζεται αυτές τις εξελίξεις στην ευρύτερη κοινότητα. Στη συνέχεια, υποστηρίζει τις βελτιωμένες και σκληρυμένες τεχνολογίες για εταιρικούς πελάτες. Με το Docker, μπορείτε να χειρίζεστε τα container σαν εξαιρετικά ελαφριές, αρθρωτές εικονικές μηχανές. Και έχετε ευελιξία με αυτά τα container, μπορείτε να τα δημιουργήσετε, να τα αναπτύξετε, να τα αντιγράψετε και να τα μετακινήσετε από περιβάλλον σε περιβάλλον, κάτι που βοηθά στη βελτιστοποίηση των εφαρμογών σας για το cloud.

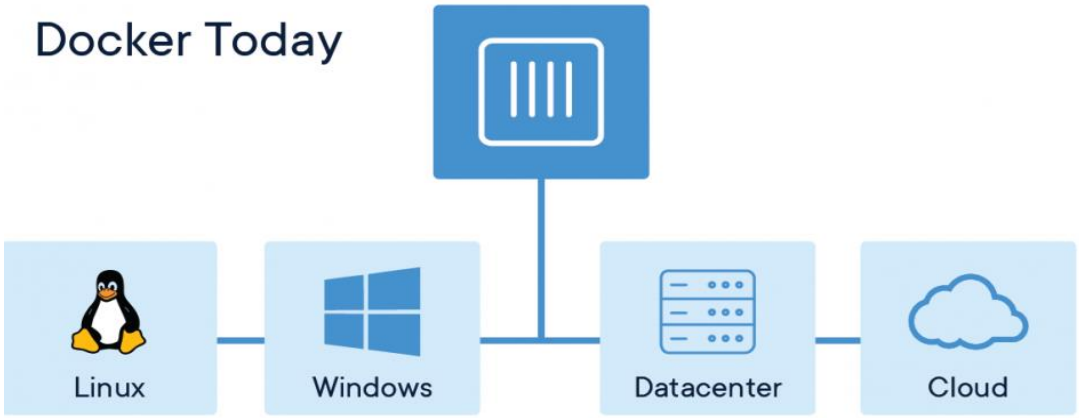
[5] https://www.doctorandroid.gr/2020/05/docker-install.html

4.2 ΠΩΣ ΛΕΙΤΟΥΡΓΕΙ ΤΟ DOCKER

Το Docker [6] συσκευάζει, προμηθεύει και εκτελεί container. Η τεχνολογία container είναι διαθέσιμη μέσω του λειτουργικού συστήματος. Ένα container συσκευάζει την υπηρεσία ή τη λειτουργία εφαρμογής με όλες τις βιβλιοθήκες, τα αρχεία διαμόρφωσης, τις εξαρτήσεις και άλλα απαραίτητα μέρη και παραμέτρους για τη λειτουργία. Κάθε container μοιράζεται τις υπηρεσίες ενός υποκείμενου λειτουργικού συστήματος. Οι εικόνες Docker περιέχουν όλες τις εξαρτήσεις που απαιτούνται για την εκτέλεση κώδικα μέσα σε ένα container, έτσι τα container που μετακινούνται μεταξύ περιβαλλόντων Docker με το ίδιο λειτουργικό σύστημα λειτουργούν χωρίς αλλαγές.

Το Docker χρησιμοποιεί απομόνωση πόρων στον πυρήνα του λειτουργικού συστήματος για την εκτέλεση πολλών container στο ίδιο λειτουργικό σύστημα. Αυτό είναι διαφορετικό από τις εικονικές μηχανές (VM), οι οποίες ενσωματώνουν ένα ολόκληρο λειτουργικό σύστημα με εκτελέσιμο κώδικα πάνω από ένα αφηρημένο επίπεδο φυσικών πόρων υλικού.

Το Docker δημιουργήθηκε για να λειτουργεί στην πλατφόρμα Linux, αλλά έχει επεκταθεί για να προσφέρει μεγαλύτερη υποστήριξη για λειτουργικά συστήματα εκτός Linux, συμπεριλαμβανομένων των Microsoft Windows και Apple OS X. Διατίθενται εκδόσεις του Docker για τις υπηρεσίες Web Amazon (AWS) και το Microsoft Azure.



[6] https://www.techtarget.com/searchitoperations/definition/Docker

4.3 ΠΛΕΟΝΕΚΤΗΜΑΤΑ DOCKER

- **Συνοχή**[7] **:** Η χρήση μιας πλατφόρμας που λειτουργεί με τον ίδιο τρόπο σε πολλά περιβάλλοντα εξαλείφει τόσο μεγάλο άγχος. Ολόκληρη η ομάδα λειτουργεί με τον ίδιο τρόπο, ανεξάρτητα από τον διακομιστή, το μηχάνημα ή το λειτουργικό σύστημα που χρησιμοποιεί. Δεν υπάρχει ανταλλαγή απόψεων μεταξύ του προσωπικού που εργάζεται σε θέματα πλατφόρμας. Απλά δημιουργήστε εικόνες που θα μετατραπούν σε container όταν αναπτυχθούν σε οποιαδήποτε συσκευή.

- **Αυτοματοποίηση :** Υπάρχουν τόσες πολλές εργασίες που, ως προγραμματιστής, μπορεί να γίνουν επαναλαμβανόμενες και μονότονες όταν γίνονται χειροκίνητα. Τα εμπορευματοκιβώτια Docker σάς επιτρέπουν να προγραμματίσετε μια σειρά εργασιών που θα πραγματοποιηθούν όταν χρειάζονται, χωρίς χειροκίνητη παρέμβαση από άνθρωπο. Αυτό εξοικονομεί χρόνο, προσπάθεια και ελαφρύνει τον φόρτο εργασίας για τους προγραμματιστές.

- **Σταθερότητα:** Το Docker βασίζεται στο Linux και, ως εκ τούτου, έχει τον πυρήνα του Linux σε κάθε container, ανεξάρτητα από το σύστημα στο οποίο εκτελείται. Στο παρελθόν, αυτό μπορεί να είχε προκαλέσει κάποια δευτερεύοντα προβλήματα σταθερότητας κατά την εκτέλεση των container σε συστήματα Mac ή Windows. Αυτές τις μέρες, παρόλο που το Docker ενημερώνεται συχνά, το περιβάλλον παραμένει σταθερό σε οποιοδήποτε σύστημα ή συσκευή. Δεν χρειάζεται να επιστρέψετε ξαφνικά σε μια προηγούμενη ενημέρωση ή να πανικοβληθείτε λόγω απρόβλεπτων προβλημάτων συμβατότητας.

- **Εξοικονομεί χώρο:**. Ο πρόδρομος των container ήταν η Εικονική Μηχανή (VM). Τα VM λειτουργούν με παρόμοιο τρόπο με τα container, αλλά λαμβάνουν φυσικούς διακομιστές και τους φτύνουν σε εικονικά περιβάλλοντα, χρησιμοποιώντας τεράστιες ποσότητες φυσικού χώρου διακομιστή και τόνους μνήμης. Τα container Docker χρησιμοποιούν μόνο τον κώδικα για την εφαρμογή και τις εξαρτήσεις της και μπορούν να εκτελούνται εξ ολοκλήρου στο Cloud, που σημαίνει ότι είναι πολύ μικρότερα και αναιρούν την απαίτηση για μεγάλους, φυσικούς διακομιστές.

[7] https://blog.iron.io/docker-containers-the-pros-and-cons-of-docker/

4.4 ΣΚΟΠΟΣ ΑΝΑΠΤΥΞΗΣ DOCKER

Ο κύριος στόχος του Docker (Rad, Bhatti, & Ahmadi, 2017) είναι ότι οποιαδήποτε εφαρμογή μπορεί να αναπτυχθεί και να εκτελεστεί σε οποιονδήποτε διακομιστή, οπουδήποτε. Το Docker είναι μια ανοιχτή πλατφόρμα λογισμικού που υποστηρίζει την ανάπτυξη εφαρμογών εντός container λογισμικού. Αυτή η start-up εταιρεία της Silicon Valley έχει προσελκύσει την προσοχή πολλών μεγάλων εταιρειών πληροφορικής. Η Amazon, η Google, η Microsoft και η Red Hat υποστηρίζουν το Docker στις πλατφόρμες τους και συνεχίζουν να συνεισφέρουν στο έργο Docker. Αλλά το Docker είναι κάτι περισσότερο από μια απλή βιβλιοθήκη εικονικοποίησης, αφαιρεί τις διαφορές μεταξύ λειτουργικών συστημάτων και δημιουργεί ένα τυποποιημένο περιβάλλον για την ανάπτυξη εφαρμογών. Οι μηχανικοί λογισμικού μπορούν να δημιουργήσουν μια φορητή πρωτότυπη εφαρμογή που μπορεί να εκτελεστεί οπουδήποτε είναι εγκατεστημένο το Docker Engine. Αυτό εξοικονομεί πολύ χρόνο για τους δημιουργούς εφαρμογών επειδή δεν χρειάζεται να υποστηρίζουν πολλές πλατφόρμες και διαφορετικά λειτουργικά συστήματα. Οι διαχειριστές συστήματος πρέπει να αφιερώνουν λιγότερο χρόνο στη ρύθμιση παραμέτρων της εφαρμογής, επειδή είναι συσκευασμένη με όλες τις εξαρτήσεις και τις βιβλιοθήκες που μπορεί να απαιτούνται για την εκτέλεση. Το γεγονός ότι κάθε εφαρμογή εκτελείται ανεξάρτητα στο δικό της container επιλύει πολλά κοινά προβλήματα, όπως την ανάγκη αφαίρεσης μιας εφαρμογής ή αντικατάστασής της με μια άλλη, και δύο εφαρμογές που απαιτούν τις ίδιες εξαρτήσεις λογισμικού ή βιβλιοθήκες. Η περίπτωση διαφορετικών εκδόσεων. Το Docker αρχικά περιοριζόταν σε εφαρμογές για περιβάλλοντα Linux, αλλά μετά από μια συνεργασία με τη Microsoft και το Windows Anniversary Update τον Σεπτέμβριο του 2016, ήταν δυνατή η εγγενής εκτέλεση container docker σε εικονικά μηχανήματα Windows 10 ή Windows Server 2016.

4.5 ΙΣΤΟΡΙΑ ΠΙΣΩ ΑΠΟ DOCKER

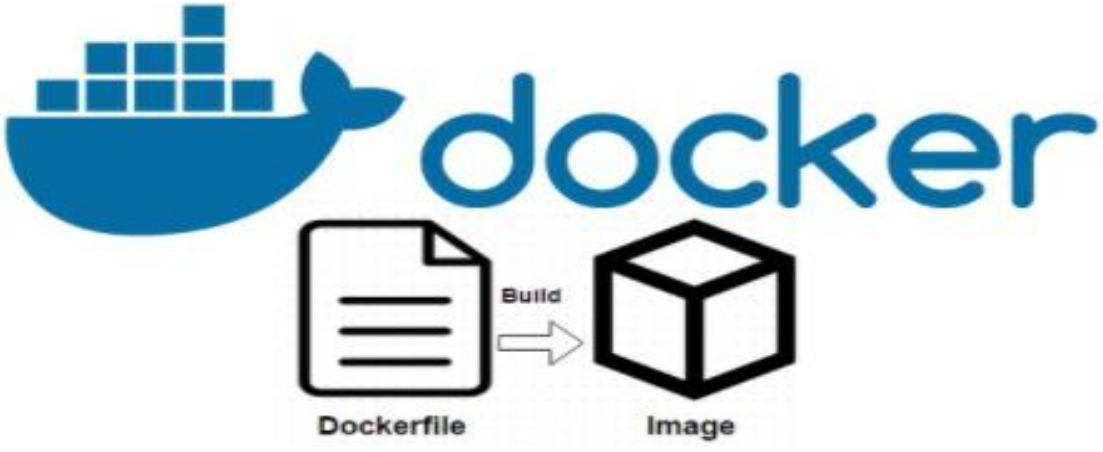
Η πρώτη έκδοση[8] του έργου dotCloud κυκλοφόρησε ως δωρεάν λογισμικό τον Μάρτιο του 2013. Δύο μήνες αργότερα κυκλοφόρησε το Docker. Μέχρι τα τέλη του 2013, η Google, η Yandex και η Baidu (οι πιο δημοφιλείς ρωσικές και κινεζικές μηχανές αναζήτησης αντίστοιχα) είχαν ήδη ενσωματώσει το Docker στην υποδομή τους στο cloud. Ο Docker μπήκε στο 2014 αφού ολοκλήρωσε τον κύκλο χρηματοδότησής του 15 εκατομμυρίων δολαρίων, επιτρέποντάς του να επενδύσει βαριά σε έργα ανοιχτού κώδικα, επιχειρηματική υποστήριξη και να επεκτείνει την κοινοτική του πλατφόρμα. Τον Απρίλιο το LXC καταργήθηκε ως ο προεπιλεγμένος χρόνος εκτέλεσης και αντικαταστάθηκε από το libcontainer του Docker. Τον επόμενο μήνα, η έκδοση 14.04 του Ubuntu έγινε η πρώτη έκδοση παραγωγής του Linux που κυκλοφόρησε με πακέτα Docker, επιτρέποντας σε εκατομμύρια διακομιστές Ubuntu να χρησιμοποιούν container Docker με τις τρεις μόνο εντολές του. Η έκδοση 1.0 κυκλοφόρησε τον Ιούνιο του 2014 στην πρώτη ομιλία του Docker που ονομάζεται DockerCon. Τον Σεπτέμβριο, ανακοινώθηκε ένα μεγάλο fund αξίας 40 εκατομμυρίων δολαρίων, ανεβάζοντας την αξία του Docker στα 400 εκατομμύρια δολάρια. Ένα μήνα αργότερα, η Docker και η Microsoft ανακοίνωσαν μια συνεργασία για τη δημιουργία ενός μοντέλου container πολλαπλών Docker που περιλαμβάνει Docker Engine για Windows και υποστήριξη για εφαρμογές που αποτελούνται από container Docker σε Linux και Windows. Τον Δεκέμβριο, η πρώτη επίσημη ευρωπαϊκή κεντρική ομιλία πραγματοποιήθηκε στο Άμστερνταμ, όπου ανακοινώθηκαν αρκετά νέα έργα που σχετίζονται με το Docker και το Docker Hub Enterprise. Το Docker έκλεισε το 2014 με την κυκλοφορία της έκδοσης 1.4, του 24ου πιο δημοφιλούς έργου του στην πλατφόρμα Github. Το πρώτο εξάμηνο του 2015, αρκετές νέες εκδόσεις του Docker προστέθηκαν διάφορα χαρακτηριστικά, όπως υποστήριξη IPv6, πολυαναμενόμενη έκδοση πελάτη για Windows, εργαλεία ενορχήστρωσης Docker Compose, Docker Swarm, Docker Machine, εργαλείο γραμμής εντολών runC κ.λπ. Δημιουργήστε και εκτελέστε container με βάση τις προδιαγραφές OCP. Τους επόμενους μήνες, το Docker Content Trust ανακοινώθηκε με την έκδοση 1.8. Παρέχει πρόσθετη ασφάλεια στο Docker με τη βοήθεια του συμβολαιογράφου, ένα έργο που βασίζεται στο Update Framework. Έκθεση κλειδιών κρυπτογράφησης για να διασφαλιστεί ότι δεν έχουν παραβιαστεί από κακόβουλους χρήστες. Επίσης, ανακοινώθηκε το

[8] https://www.businesswire.com/news/home/20131029005746/en/dotCloud-Inc.-is-Now-Docker-Inc.

Docker Toolbox. Έτσι όλα τα εργαλεία που χρειάζεστε για να εγκαταστήσετε και να λειτουργήσετε το Docker βρίσκονται σε ένα πακέτο για αυτόν. Το 2016, υπήρξαν αρκετές νέες ανακοινώσεις για το Docker, η πιο σημαντική από τις οποίες ήταν η κυκλοφορία του νέου προγράμματος-πελάτη Docker για Windows και Mac. Αυτός ο πελάτης έχει βελτιωμένη ταχύτητα και λειτουργία και βασίζεται πλέον σε εικονικές μηχανές xhyve. Επιπλέον, περιλαμβάνει όλα τα εργαλεία από την έκδοση Docker Toolbox και διευκολύνει την πρόσβαση σε container τοπικού δικτύου.

4.6 ΑΣΦΑΛΕΙΑ ΣΤΟ DOCKER

Η ασφάλεια (Bui, 2015) του Docker είναι ένα από τα πιο κρίσιμα προβλήματα που πρέπει να αντιμετωπίσουν οι μηχανικοί. Επί του παρόντος, ορισμένες διεργασίες στο Docker απαιτούν δικαιώματα root για να εκτελεστούν, προκειμένου να είναι πλήρως λειτουργικά σε ένα σύστημα. Αυτό θέτει τον κεντρικό υπολογιστή σε κίνδυνο εάν ανακαλυφθεί μια ευπάθεια Docker, επομένως γίνονται προσπάθειες να βρεθούν τρόποι εκτέλεσης του daemon Docker χωρίς δικαιώματα root. Λόγω των προόδων που έγιναν σε αυτόν τον τομέα, το Docker υποστηρίζει τώρα μια αρχιτεκτονική 2-daemon που επιτρέπει την εκτέλεση των περισσότερων λειτουργιών χωρίς την ανάγκη πρόσβασης root, με εξαίρεση αυτές που έχουν σχεδιαστεί να λειτουργούν με αυτό το επίπεδο πρόσβασης από προεπιλογή. Την ασφάλεια για τις εικόνες Docker διαχειρίζεται επίσης το Docker.

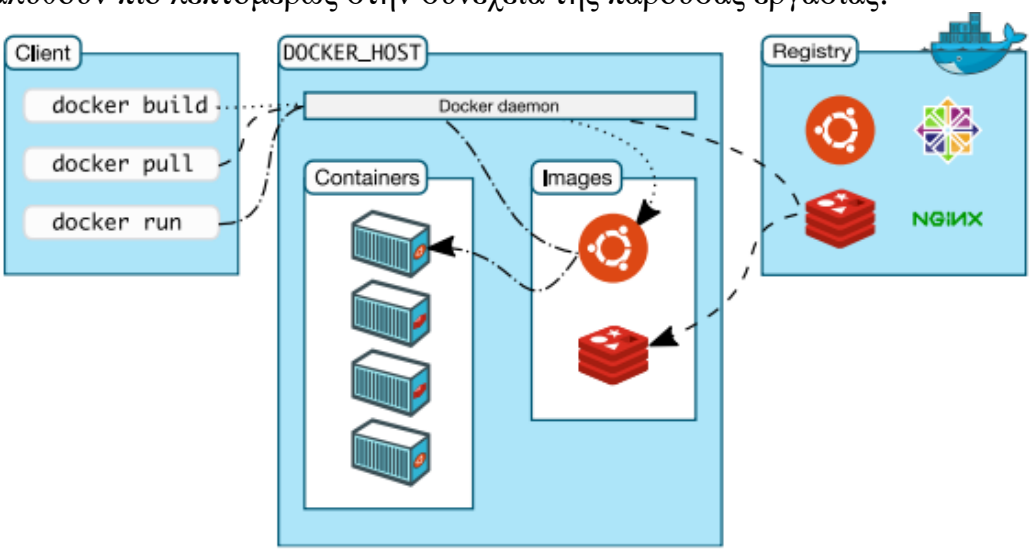


Προκειμένου να επαληθεύσει τη νομιμότητα και την ακεραιότητα κάθε Εικόνας Docker, το Docker άρχισε να προωθεί μια νέα δυνατότητα που ονομάζεται "υπογραφή εικόνας" τον Δεκέμβριο του 2014. Αυτή η δυνατότητα χρησιμοποιεί ψηφιακές υπογραφές. Ωστόσο, τα πράγματα δεν πήγαν όπως είχαν προγραμματιστεί όταν ανακαλύφθηκε ότι ο μόνος τρόπος επικύρωσης

μιας εικόνας Docker βασίστηκε στην ύπαρξη ενός ψηφιακά πιστοποιημένου αρχείου δήλωσης (ένα αρχείο που περιέχει τα μεταδεδομένα για μια συλλογή αρχείων που συνοδεύει) και ότι το άθροισμα ελέγχου του αρχείου εικόνας (Docker Image) δεν ελέγχεται καθόλου. Αυτό σημαίνει ότι ένας κακός χρήστης θα μπορούσε να χρησιμοποιήσει ένα έγκυρο αρχείο δήλωσης και ένα κακόβουλο αρχείο Docker Image για να παρακάμψει αυτόν τον μηχανισμό ελέγχου. Οι χώροι ονομάτων χρήστη Linux, μια σχετικά πρόσφατη προσθήκη στον πυρήνα, είναι μια άλλη πιθανή απειλή για την εικονικοποίηση που βασίζεται σε container λογισμικού, επειδή δεν έχουν υποβληθεί ακόμη σε εκτεταμένες δοκιμές σε ρυθμίσεις πραγματικού κόσμου. Ένα από τα ζητήματα ασφαλείας που ανακαλύφθηκαν αφορούσε τη δυνατότητα μιας διεργασίας να χρησιμοποιεί τον χώρο ονομάτων χρήστη και να αφαιρεί μια ομάδα χρηστών από αυτόν για να αποκτήσει πρόσβαση σε μια τοποθεσία στο σύστημα αρχείων στην οποία ούτε ο τρέχων χρήστης είχε πρόσβαση. Εκτός από τα προβλήματα με τον πυρήνα του Linux και το Docker ως λογισμικό, υπάρχουν πάντα προβλήματα που προκαλούνται από τον ανθρώπινο παράγοντα. Για παράδειγμα, οι ρυθμίσεις δημιουργίας container θα μπορούσαν να έχουν εκμεταλλεύσιμα ελαττώματα ασφαλείας, όπως μια ανοιχτή πόρτα σε ένα container που εκθέτει μια υπηρεσία.

4.7 ΒΑΣΙΚΑ ΣΤΟΙΧΕΙΑ ΤΟΥ DOCKER

Η αρχιτεκτονική του Docker φαίνεται στην Εικόνα. Στην συνέχεια παρουσιάζονται μερικά από τα βασικά στοιχεία του Docker[9], Τα οποία θα αναλυθούν πιο λεπτομερώς στην συνέχεια της παρούσας εργασίας.



[9] https://docs.docker.com/get-started/overview/

- **Docker client:**

  Το Docker client είναι ένας από τους τρόπους για να αλληλοεπιδράσουμε με το docker. Όταν χρησιμοποιούμε εντολές όπως η docker run, το client στέλνει αυτές τις εντολές στο daemon, το οποίο τις εκτελεί. Οι εντολές που ξεκινούν με “docker” χρησιμοποιούν το docker api.

- **Docker daemon:**

  Το Docker daemon ακούει για docker api αιτήσεις και διαχειρίζεται αντικείμενα του docker όπως είναι οι εικόνες, τα container και τα δίκτυα. Ένας daemon μπορεί επίσης να επικοινωνήσει με άλλους daemon για να διαχειριστεί υπηρεσίες του docker

- **Docker image:**

  Μία docker εικόνα (Docker image) είναι ένα read-only πρότυπο που περιέχει οδηγίες για την δημιουργία ενός Docker container. Συνήθως, κάθε εικόνα βασίζεται σε κάποια άλλη εικόνα με μερικές επιπρόσθετες αλλαγές. Για παράδειγμα, θα μπορούσαμε να βασιστούμε σε μια εικόνα των Ubuntu για να δημιουργήσουμε μια νέα εικόνα η οποία θα εγκαθιστά το postgres καθώς και θα δημιουργεί τους πίνακες που θέλουμε σε αυτό.

  Μπορούμε είτε να δημιουργήσουμε δικές μας εικόνες (δημιουργώντας τες με την χρήση ενός Docker file) είτε να χρησιμοποιήσουμε εικόνες που έχουν δημιουργήσει άλλοι χρήστες και έχουν ανεβάσει σε κάποιο registry.

- **Docker container:**

  Ένα Docker container είναι ένα απομονωμένο περιβάλλον που εκτελεί εφαρμογές. Μπορούμε να διαχειριστούμε ένα container χρησιμοποιώντας το docker api.

  Από προεπιλογή, ένα container είναι απομονωμένο τόσο από τα άλλα container όσο και από την host μηχανή. Μπορούμε να κάνουμε έναν έλεγχο για το πόσο απομονωμένο θα είναι το container.

Ένα container ορίζεται από την docker εικόνα του καθώς και ότι ρυθμίσεις του παρέχουμε κατά την δημιουργία του. Τα container μοιράζονται το kernel με το host λειτουργικό σύστημα (όπως φαίνεται στην Εικόνα).

- **Docker Registry:**

  Ένα Docker Registry περιέχει docker εικόνες. Ένα registry μπορεί να είτε ιδιωτικό είτε δημόσιο. Όταν χρησιμοποιούμε την εντολή docker run ή docker pull, το docker θα τραβήξει τις απαιτούμενες εικόνες από το προεπιλεγμένο registry. Με την εντολή docker push μπορούμε να ανεβάσουμε μια εικόνα στο registry.

  Το προεπιλεγμένο απομακρυσμένο registry είναι το docker hub το οποίο είναι ένα δημόσιο registry που διαχειρίζεται η Docker Inc. και το οποίο μπορεί να χρησιμοποιήσει ο οποιοσδήποτε. Στο docker hub μπορούμε να βρούμε εικόνες για πολλές δημοφιλείς εφαρμογές.

- **Docker file:**

  Ένα Docker file είναι ένα αρχείο κειμένου που περιέχει οδηγίες για την δημιουργία μιας docker εικόνας. Το Docker file περιέχει όλες τις γραμμές κώδικα που θα τρέχαμε στο τερματικό προκειμένου να σχηματίσουμε την εικόνα. Με την εντολή docker build δημιουργούμε μια εικόνα από ένα docker file.

- **Docker Compose:**

  Το Compose είναι ένα εργαλείο για να ορίσουμε και να τρέξουμε εφαρμογές στο Docker που χρησιμοποιούν πολλαπλά container. Με το compose, μπορούμε να χρησιμοποιήσουμε ένα YAML αρχείο για να ρυθμίσουμε τις υπηρεσίες της εφαρμογής μας και έπειτα με μόνο μία εντολή να τις εκκινήσουμε όλες μαζί.

- **Docker Service:**

  Μια υπηρεσία (service) μας επιτρέπει να ορίσουμε το πως θέλουμε να τρέξουμε τα container των εφαρμογών μας σε ένα σμήνος (swarm). Μια υπηρεσία μας επιτρέπει να ορίσουμε την επιθυμητή κατάσταση, όπως το πόσα αντίγραφα (replicas) της εφαρμογής θέλουμε να τρέξουμε συγχρόνως. Από προεπιλογή γίνεται εξισορρόπηση φορτίου (load-

balancing) ανάμεσα σε όλους τους κόμβους-εργάτες (worker nodes) του σμήνους. Στο χρήστη η υπηρεσία φαίνεται σαν να είναι μία μόνο εφαρμογή.

Συχνά μια υπηρεσία είναι μια μικρουπηρεσία στο πλαίσιο μιας μεγαλύτερης εφαρμογής. Παραδείγματα υπηρεσιών ίσως περιλαμβάνουν έναν HTTP server, μια βάση δεδομένων ή οποιοδήποτε άλλο είδος εκτελέσιμου προγράμματος θα επιθυμούσαμε να τρέξουμε σε ένα κατανεμημένο περιβάλλον.

- **Swarm mode:**

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

4.8 ΑΝΑΤΟΜΙΑ ΜΙΑΣ ΕΙΚΟΝΑΣ DOCKER

Μια εικόνα Docker αποτελείται [10]από μια συλλογή αρχείων που συνδυάζουν όλα τα απαραίτητα όπως εγκαταστάσεις , κώδικα εφαρμογής και εξαρτήσεις που απαιτούνται για τη διαμόρφωση ενός πλήρως λειτουργικού περιβάλλοντος container. Μπορείτε να δημιουργήσετε μια εικόνα Docker χρησιμοποιώντας μία από τις δύο μεθόδους:

- **Διαδραστικό**: Εκτελώντας ένα container από μια υπάρχουσα εικόνα Docker, αλλάζοντας με μη αυτόματο τρόπο αυτό το περιβάλλον container μέσω μιας σειράς ενεργών βημάτων και αποθηκεύοντας την κατάσταση που προκύπτει ως νέα εικόνα.
- **Docker file**: Κατασκευάζοντας ένα αρχείο απλού κειμένου, γνωστό ως Docker file, το οποίο παρέχει τις προδιαγραφές για τη δημιουργία μιας εικόνας Docker.

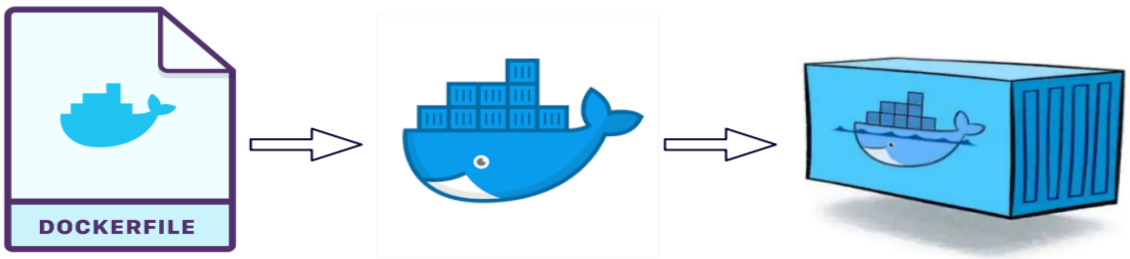
Θα καλύψουμε αυτές τις δύο μεθόδους με περισσότερες λεπτομέρειες αργότερα σε αυτόν τον οδηγό. Προς το παρόν, όμως, ας εστιάσουμε στις πιο σημαντικές έννοιες της εικόνας του Docker.

[10] https://jfrog.com/knowledge-base/a-beginners-guide-to-understanding-and-building-docker-images/

4.8.1 ΕΠΙΠΕΔΑ ΕΙΚΟΝΑΣ

Κάθε ένα από τα αρχεία που συνθέτουν μια εικόνα Docker είναι γνωστό ως στρώμα . Αυτά τα στρώματα σχηματίζουν μια σειρά από ενδιάμεσες εικόνες , χτισμένες η μία πάνω στην άλλη σε στάδια, όπου κάθε στρώμα εξαρτάται από το στρώμα ακριβώς κάτω από αυτό.

Η ιεραρχία των επιπέδων σας είναι το κλειδί για την αποτελεσματική διαχείριση του κύκλου ζωής των εικόνων Docker. Έτσι, θα πρέπει να οργανώσετε επίπεδα που αλλάζουν πιο συχνά όσο πιο ψηλά γίνεται στη στοίβα. Αυτό συμβαίνει επειδή, όταν κάνετε αλλαγές σε ένα επίπεδο στην εικόνα, το Docker όχι μόνο αναδομεί αυτό το συγκεκριμένο επίπεδο, αλλά και όλα τα επίπεδα που δημιουργούνται από αυτό. Επομένως, μια αλλαγή σε ένα επίπεδο στο πάνω μέρος μιας στοίβας περιλαμβάνει τη μικρότερη υπολογιστική εργασία για την αναδόμηση ολόκληρης της εικόνας.



4.8.2 ΣΤΡΩΜΑ CONTAINER

Κάθε φορά που το Docker εκκινεί ένα container από μια εικόνα, προσθέτει ένα λεπτό εγγράψιμο στρώμα, γνωστό ως στρώμα container, το οποίο αποθηκεύει όλες τις αλλαγές στο container κατά τη διάρκεια εκτέλεσης του. Δεδομένου ότι αυτό το επίπεδο είναι η μόνη διαφορά μεταξύ ενός ζωντανού λειτουργικού container και της ίδιας της εικόνας του Docker προέλευσης, οποιοσδήποτε αριθμός container παρόμοια για παρόμοια μπορεί ενδεχομένως να μοιράζεται πρόσβαση στην ίδια υποκείμενη εικόνα, διατηρώντας τη δική τους μεμονωμένη κατάσταση.

4.8.3 ΓΟΝΙΚΗ ΕΙΚΟΝΑ

Στις περισσότερες περιπτώσεις, το πρώτο επίπεδο μιας εικόνας Docker είναι γνωστό ως γονική εικόνα. Είναι το θεμέλιο πάνω στο οποίο χτίζονται όλα τα άλλα στρώματα και παρέχει τα βασικά δομικά στοιχεία για τα περιβάλλοντα container. Μπορείτε να βρείτε μια μεγάλη ποικιλία από έτοιμες εικόνες για χρήση ως γονική εικόνα στο δημόσιο μητρώο container Docker Hub. Μπορείτε επίσης να τα βρείτε σε έναν μικρό αριθμό υπηρεσιών τρίτων, όπως το Μητρώο container της Google . Εναλλακτικά, μπορείτε να χρησιμοποιήσετε μία από τις υπάρχουσες εικόνες σας ως βάση για τη δημιουργία νέων.

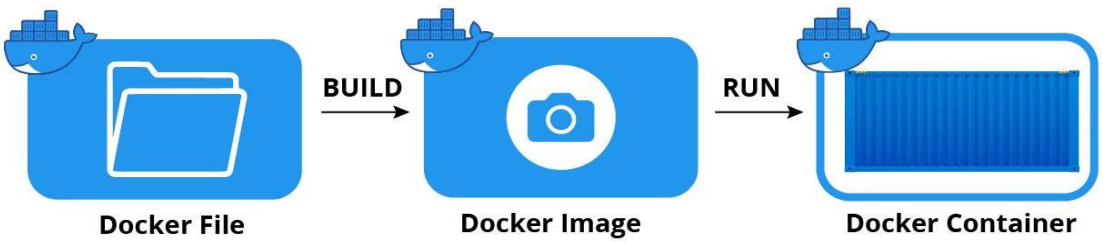
Μια τυπική γονική εικόνα μπορεί να είναι μια απογυμνωμένη διανομή Linux ή να συνοδεύεται από μια προεγκατεστημένη υπηρεσία, όπως ένα σύστημα διαχείρισης βάσης δεδομένων (DBMS) ή ένα σύστημα διαχείρισης περιεχομένου (CMS) .

4.8.4 ΕΙΚΟΝΑ ΒΑΣΗΣ

Με απλά λόγια, μια εικόνα βάσης είναι ένα κενό πρώτο επίπεδο, το οποίο επιτρέπει να δημιουργήσετε τις εικόνες Docker από την αρχή. Οι εικόνες βάσης δίνουν τον πλήρη έλεγχο των περιεχομένων των εικόνων, αλλά γενικά προορίζονται για πιο προχωρημένους χρήστες Docker.

4.8.5 DOCKER MANIFEST

Μαζί με ένα σύνολο αρχείων μεμονωμένων επιπέδων, μια εικόνα Docker περιλαμβάνει επίσης ένα πρόσθετο αρχείο γνωστό ως δήλωση. Αυτή είναι ουσιαστικά μια περιγραφή της εικόνας σε μορφή **JSON** και περιλαμβάνει πληροφορίες όπως ετικέτες εικόνας, μια ψηφιακή υπογραφή και λεπτομέρειες σχετικά με τον τρόπο διαμόρφωσης του container για διαφορετικούς τύπους πλατφορμών κεντρικού υπολογιστή.



ΚΕΦΑΛΑΙΟ 5 : KUBERNETES

5.1 ΤΙ ΕΙΝΑΙ KUBERNETES

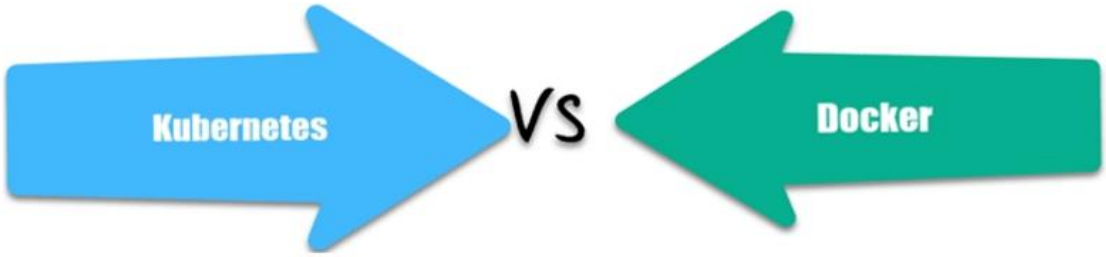
Το Kubernetes[11] (Melendez, 2005)είναι μια φορητή, επεκτάσιμη πλατφόρμα ανοιχτού κώδικα για τη διαχείριση φόρτου εργασίας και υπηρεσιών με εμπορευματοκιβώτια, που διευκολύνει τόσο τη δηλωτική διαμόρφωση όσο και τον αυτοματισμό. Έχει ένα μεγάλο, ταχέως αναπτυσσόμενο οικοσύστημα. Τα εργαλεία Kubernetes, η υποστήριξη και οι υπηρεσίες είναι ευρέως διαθέσιμα.

Το όνομα Kubernetes προέρχεται από τα ελληνικά, που σημαίνει πιλότος ή τιμονιέρης. Το K8 ως συντομογραφία προκύπτει από την καταμέτρηση των οκτώ γραμμάτων μεταξύ του "K" και του "s". Η Google δημιούργησε το έργο Kubernetes ανοιχτού κώδικα το 2014. Το Kubernetes συνδυάζει πάνω από 15 χρόνια εμπειρίας της Google στη διαχείριση φόρτου εργασίας παραγωγής σε κλίμακα με τις καλύτερες ιδέες και πρακτικές από την κοινότητα.

[11] https://kubernetes.io/docs/concepts/overview/

5.2 KUBERNETES vs DOCKER

Η συζήτηση[12] γύρω από Kubernetes εναντίον Docker συχνά πλαισιώνεται ως είτε-ή: πρέπει να χρησιμοποιήσω Kubernetes ή Docker; Αυτό είναι σαν να συγκρίνετε τα μήλα με τη μηλόπιτα και είναι μια κοινή παρανόηση ότι πρέπει να επιλέξετε το ένα ή το άλλο.



Η διαφορά μεταξύ Kubernetes και Docker γίνεται πιο εύκολα κατανοητή όταν πλαισιώνεται ως ερώτηση "και-και". Το γεγονός είναι ότι δεν χρειάζεται να επιλέξετε - Το Kubernetes και το Docker είναι θεμελιωδώς διαφορετικές τεχνολογίες που λειτουργούν καλά μαζί για τη δημιουργία, την παράδοση και την κλιμάκωση εφαρμογών με container.

5.3 ΔΙΑΦΟΡΑ ΜΕΤΑΞΥ KUBERNETES ΚΑΙ DOCKER

Αν και είναι σύνηθες να συγκρίνουμε το Kubernetes με το Docker, μια πιο κατάλληλη σύγκριση είναι το Kubernetes εναντίον του Docker Swarm. Το Docker Swarm είναι η τεχνολογία ενορχήστρωσης του Docker που εστιάζει στη ομαδοποίηση για container Docker ενσωματωμένο στενά στο οικοσύστημα Docker και χρησιμοποιώντας το δικό του API.

Μια θεμελιώδης διαφορά μεταξύ Kubernetes και Docker είναι ότι το Kubernetes προορίζεται να τρέχει σε ένα σύμπλεγμα ενώ το Docker εκτελείται σε έναν μόνο κόμβο. Το Kubernetes είναι πιο εκτεταμένο από το Docker Swarm και προορίζεται να συντονίζει συμπλέγματα κόμβων σε κλίμακα στην παραγωγή με αποτελεσματικό τρόπο. Οι ομάδες Kubernetes μονάδες προγραμματισμού που μπορούν να περιέχουν ένα ή περισσότερα container στο οικοσύστημα Kubernetes κατανέμονται μεταξύ των κόμβων για να παρέχουν υψηλή διαθεσιμότητα.

[12] https://azure.microsoft.com/en-us/solutions/kubernetes-vs-docker/

5.4 ΣΥΝΔΥΑΣΜΟΣ KUBERNETES ΚΑΙ DOCKER

Οι Kubernetes και τα Docker συνεργάζονται. Το Docker παρέχει ένα ανοιχτό πρότυπο για τη συσκευασία και τη διανομή εφαρμογών σε εμπορευματοκιβώτια.

Χρησιμοποιώντας το Docker, μπορείτε να δημιουργήσετε και να εκτελέσετε container, καθώς και να αποθηκεύσετε και να μοιραστείτε εικόνες container. Κάποιος μπορεί εύκολα να εκτελέσει μια κατασκευή Docker σε ένα σύμπλεγμα Kubernetes, αλλά το ίδιο το Kubernetes δεν είναι μια πλήρης λύση.

Για να βελτιστοποιήσετε το Kubernetes στην παραγωγή, εφαρμόστε πρόσθετα εργαλεία και υπηρεσίες για τη διαχείριση της ασφάλειας, της διακυβέρνησης, της ταυτότητας και της πρόσβασης μαζί με ροές εργασίας συνεχούς ενοποίησης/συνεχούς ανάπτυξης (CI/CD) και άλλες πρακτικές DevOps.

Χρησιμοποιώντας Kubernetes με Docker πετυχαίνουμε :

- Η υποδομή γίνεται πιο ισχυρή και η εφαρμογή πιο διαθέσιμη. Η εφαρμογή θα παραμείνει συνδεδεμένη, ακόμα κι αν ορισμένοι από τους κόμβους είναι εκτός σύνδεσης.
- Η εφαρμογή πιο επεκτάσιμη. Εάν η εφαρμογή αρχίσει να φορτώνει πολύ περισσότερο και πρέπει να κλιμακωθείτε για να μπορέσετε να παρέχετε καλύτερη εμπειρία χρήστη, είναι απλό να περιστρέψετε περισσότερα container ή να προσθέσετε περισσότερους κόμβους στο σύμπλεγμα Kubernetes.

ΚΕΦΑΛΑΙΟ 6 :

ΕΓΚΑΤΑΣΤΑΣΗ DOCKER

6.1 ΣΕ WINDOWS

Για την εγκατάσταση του Docker σε windows, ακολουθούμε τα παρακάτω βήματα:

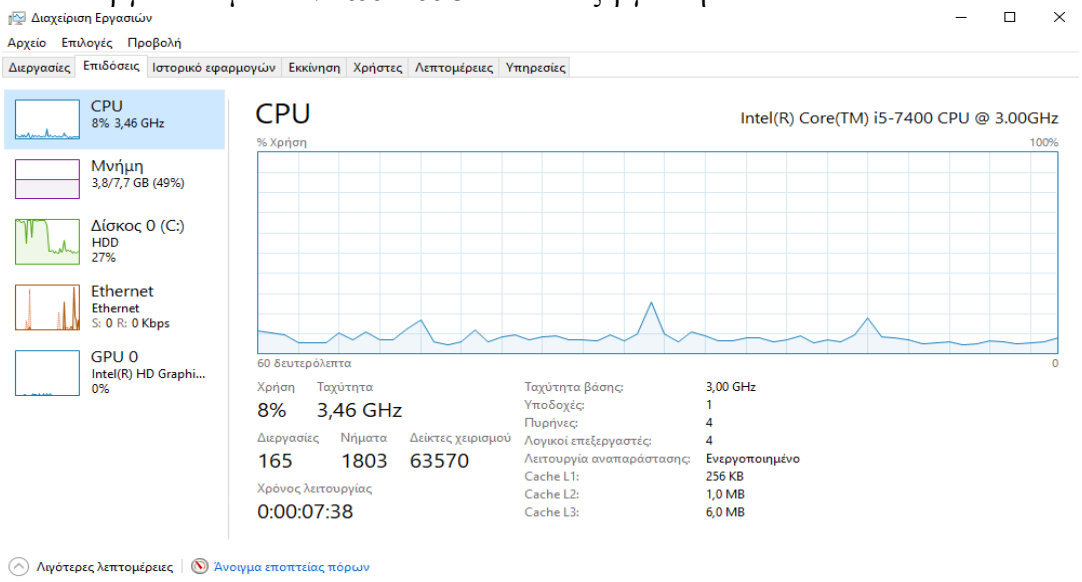
- **ΒΗΜΑ 1ο :**

  Αρχικά εγκαθιστούμε το WSL2 Linux Kernel από την σελίδα της Microsoft:

  https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi

- **ΒΗΜΑ 2ο :**

  Ανοίγουμε την διαχείριση εργασιών στην επιλογή Επιδόσεις και ελέγχουμε εάν είναι ενεργοποιημένη η λειτουργία αναπαράστασης (Virtualization) του επεξεργαστή. Σε περίπτωση που δεν είναι ανοίγουμε τις ρυθμίσεις των BIOS και ενεργοποιούμε το Virtualization του επεξεργαστή.

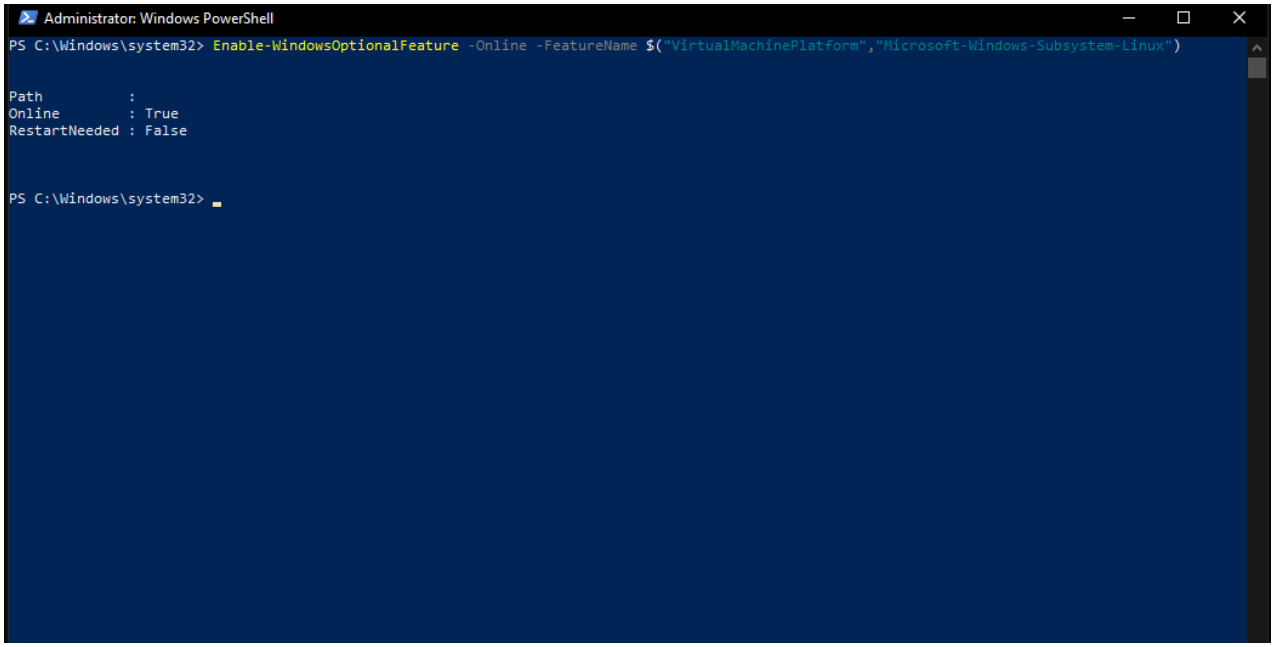


- **ΒΗΜΑ 3ο :**

Στο επόμενο βήμα ενεργοποιούμε το Virtual Machine Platform και το Windows Subsystem for Linux τα οποία μπορούμε να ενεργοποιήσουμε είτε από τις ρυθμίσεις είτε ανοίγοντας CMD(Command Promt) ως Διαχειριστής και γράφοντας το παρακάτω command.

```
Enable-WindowsOptionalFeature -Online -
FeatureName
$("VirtualMachinePlatform","Microsoft-Windows-
Subsystem-Linux")
```

Στην συνέχεια ξανατρέχουμε την εντολή για να δούμε εάν είναι ενεργοποιημένα, σε περίπτωση που δεν είναι θα χρειαστεί να κάνουμε επανεκκίνηση στον υπολογιστή μας.

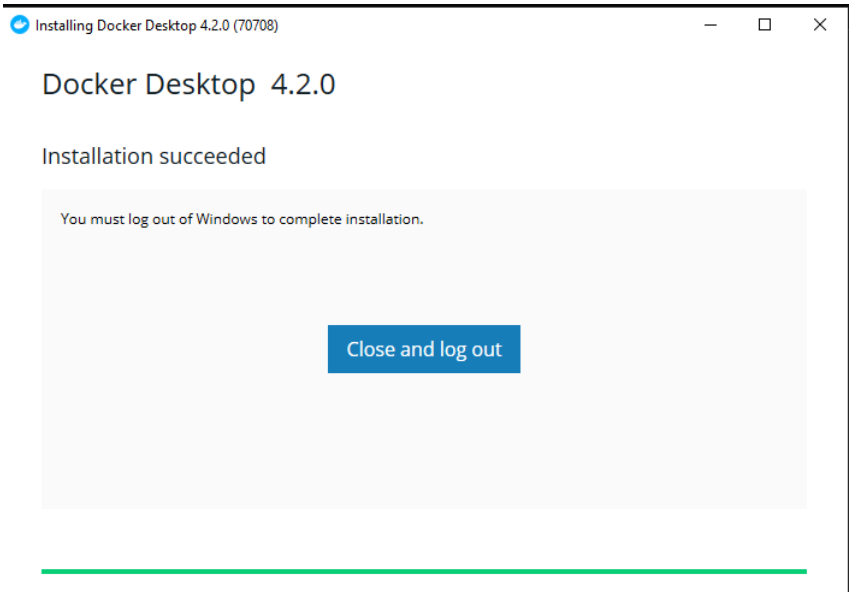


- **ΒΗΜΑ 4ο :**

Πλοηγούμαστε στην σελίδα https://docs.docker.com/desktop/windows/install/ ώστε να κατεβάσουμε το αρχείο εγκατάστασης του Docker (InstallDocker.msi).

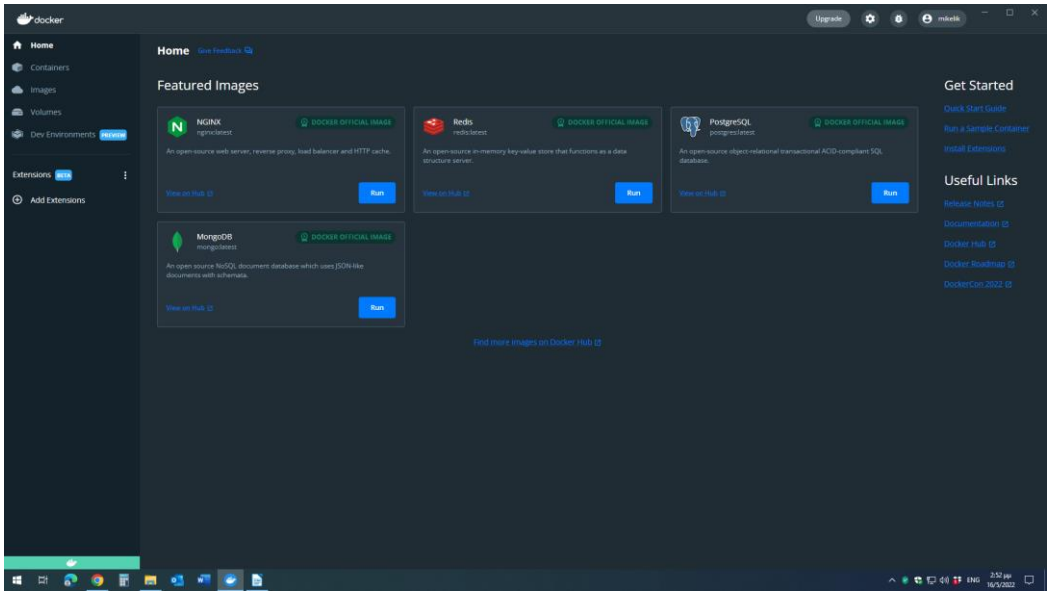
- **ΒΗΜΑ 5ο :**

  Ακολουθούμε τα βήματα του οδηγού εγκατάστασης για την αποδοχή της άδειας, εξουσιοδότησης και συνεχίζουμε με την εγκατάσταση.



Τελευταίο στάδιο είναι να κάνουμε accept στα terms of service του Docker. Αφού το κάνουμε αυτό θα αναδυθεί η εφαρμογή Docker for Desktop όπου κάτω αριστερά το πράσινο εικονίδιο μας διαβεβαιώνει ότι η εγκατάσταση του προγράμματος μας έχει γίνει με επιτυχία.

Στην περίπτωση που το εικονίδιο δεν είναι πράσινο θα πρέπει να ανατρέξουμε στα βήματα : 1, 2, 3



6.1 ΣΕ LINUX

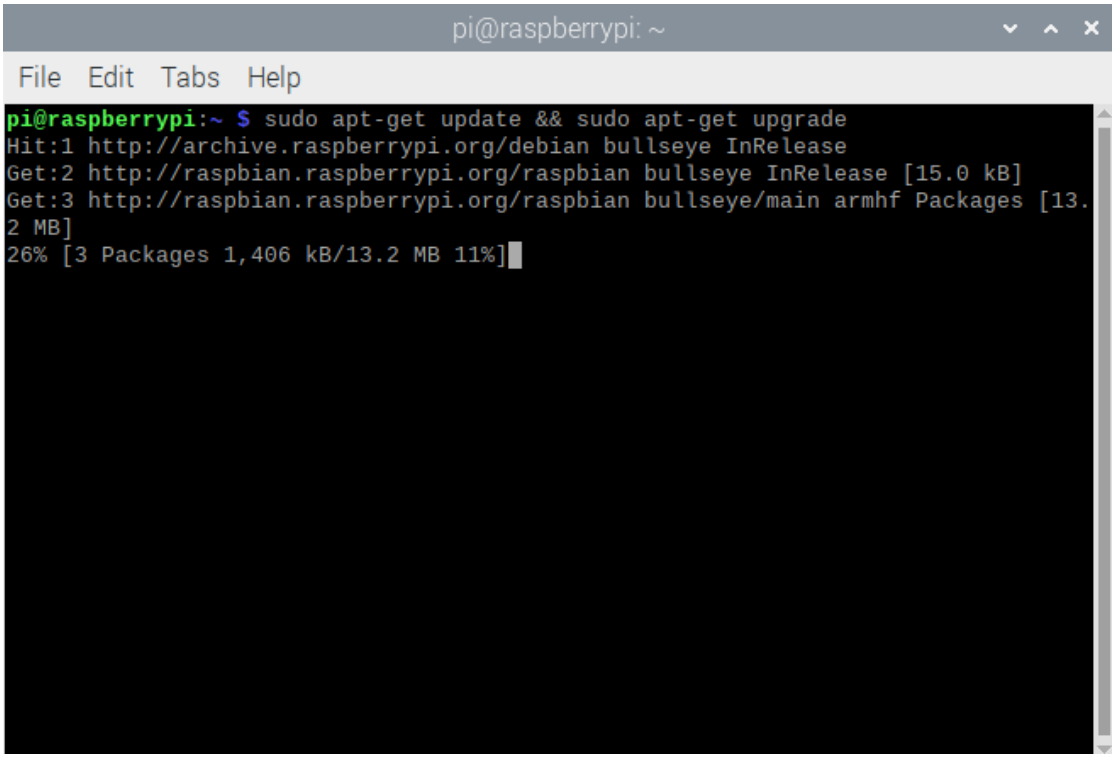
Για την εγκατάσταση του Docker σε linux, χρησιμοποιήθηκε ένα Raspberry pi 3 (linux περιβάλλον) στο οποίο ακολουθήσαμε τα παρακάτω βήματα:

- **ΒΗΜΑ 1ο :**

  Ξεκινάμε με την ενημέρωση και την αναβάθμιση του συστήματος. Αυτό διασφαλίζει ότι θα εγκαταστήσουμε την πιο πρόσφατη έκδοση του λογισμικού.

  Ανοίγουμε ένα παράθυρο τερματικού και εκτελούμε την εντολή:

```
sudo apt-get update && sudo apt-get upgrade
```

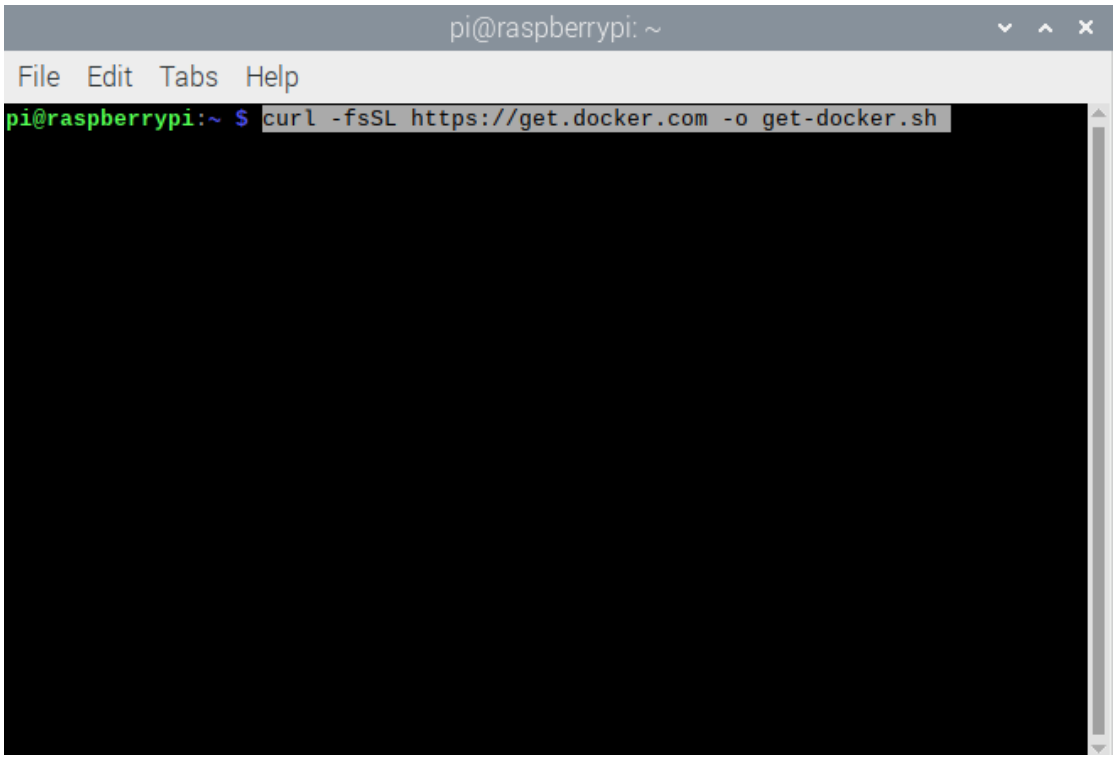


- **ΒΗΜΑ 2^{ο} :**

Στην συνέχεια προχωράμε στην λήψη του script εγκατάστασης.

Ανοίγουμε ένα παράθυρο τερματικού και εκτελούμε την εντολή:

```
curl -fsSL https://get.docker.com -o get-docker.sh
```

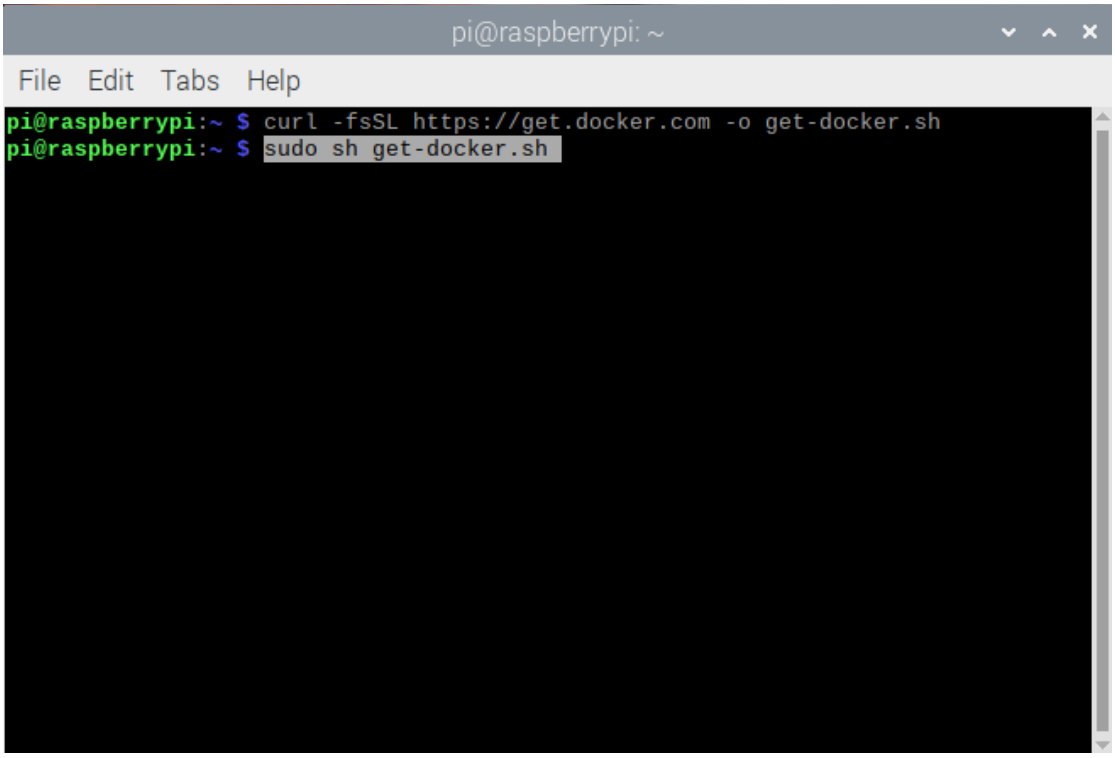


- **ΒΗΜΑ 3ο :**

Τρέχουμε το script.

Ανοίγουμε ένα παράθυρο τερματικού και εκτελούμε την εντολή:

```
sudo sh get-docker.sh
```



Η έξοδος της εντολής θα μας πει ποια έκδοση του Docker εκτελείται τώρα στο σύστημά μας.

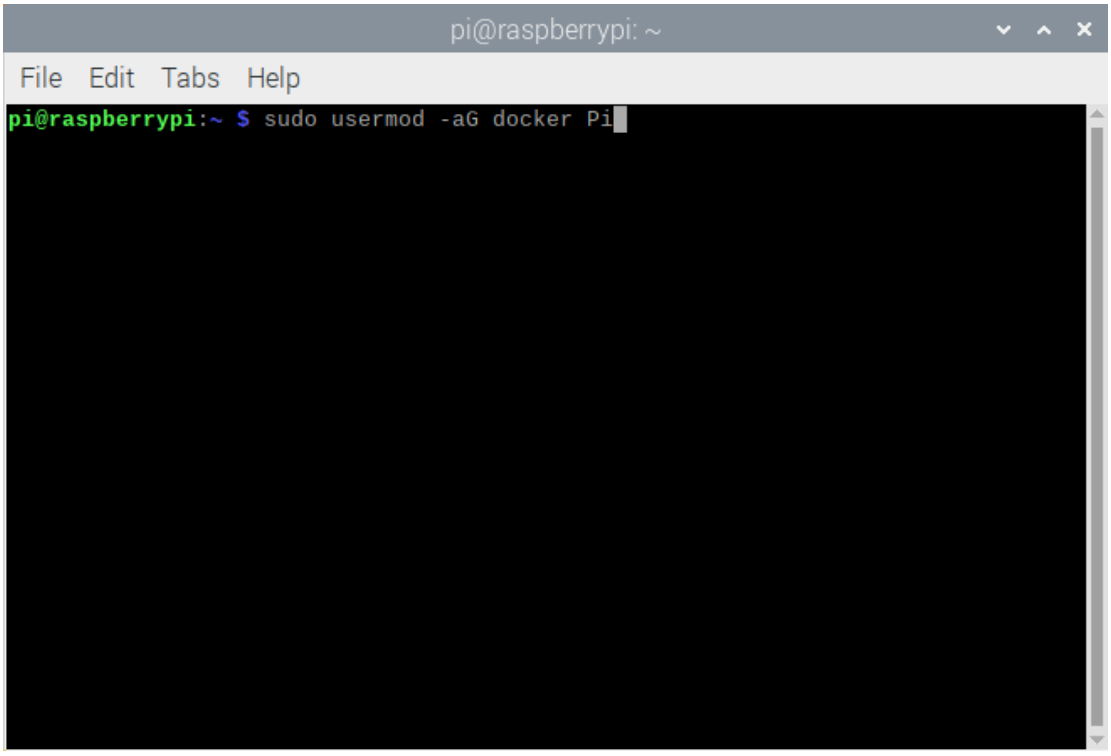
```
Server: Docker Engine - Community
 Engine:
  Version:          20.10.6
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.13.15
  Git commit:       8728dd2
  Built:            Fri Apr  9 22:44:17 2021
  OS/Arch:          linux/arm
  Experimental:     false
 containerd:
  Version:          1.4.4
  GitCommit:        05f951a3781f4f2c1911b05e61c160e9c30eaa8e
 runc:
  Version:          1.0.0-rc93
  GitCommit:        12644e614e25b05da6fd08a38ffa0cfe1903fdec
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0

================================================================================

To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:

    dockerd-rootless-setuptool.sh install

Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.


To run the Docker daemon as a fully privileged service, but granting non-root
users access, refer to https://docs.docker.com/go/daemon-access/

WARNING: Access to the remote API on a privileged Docker daemon is equivalent
         to root access on the host. Refer to the 'Docker daemon attack surface'
         documentation for details: https://docs.docker.com/go/attack-surface/
```

- **ΒΗΜΑ 4ο :**

  Στην συνέχεια για να προσθέσουμε τον Pi χρήστη σαν default user.

  Ανοίγουμε ένα παράθυρο τερματικού και εκτελούμε την εντολή:

```
s udo usermod -aG docker Pi
```

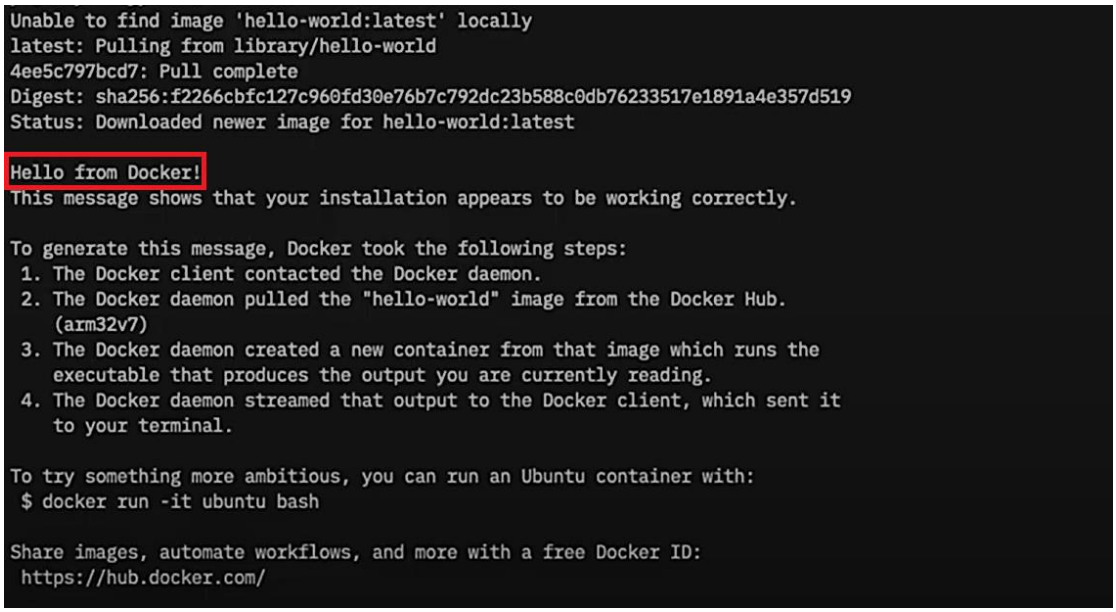


- **ΒΗΜΑ 5ο :**

Τέλος ο καλύτερος τρόπος για να ελέγξουμε εάν το Docker έχει ρυθμιστεί σωστά είναι τρέχοντας το Hello World container.

Ανοίγουμε ένα παράθυρο τερματικού και εκτελούμε την εντολή:

```
docker run hello-world
```



ΚΕΦΑΛΑΙΟ 7 : DOCKER IMAGES

7.1 DOCKER REGISTRY

Το Docker Registry (Anwar, et al., 2018) είναι μία μη καταστατική και άκρως επεκτάσιμη εφαρμογή εξυπηρετητή που δίνει την δυνατότητα στους χρήστες να αποθηκεύουν και να διαμοιράζουν τα ονομασμένα **Docker images**. Ένα ιδιωτικό Registry είναι χρήσιμο για περιπτώσεις όπου χρειάζεται αυστηρός έλεγχος για το που αποθηκεύονται τα images, αλλά και για πλήρη έλεγχο πάνω στην διανομή τους. Τα ιδιωτικά Registry χρησιμοποιούνται κυρίως σε επιχειρησιακές ροές εργασιών για την αποθήκευση και την διανομή images σε μια ομάδα ανάπτυξης λογισμικού. Για την δημιουργία ενός Registry χρειάζεται η version του Docker Engine 1.6 ή επόμενες vresion. Με την παρακάτω εντολή εκκινούμε το ιδιωτικό μας Registry:

```
$ docker run -d -p 6000:6000 --name my_own_registry
registry:2
```

Με την παραπάνω εντολή εκτελούμε ένα container το οποίο τρέχει μια έκδοση εξυπηρετητή Registry του Docker στην οποία μπορούμε να έχουμε πρόσβαση μέσω της θύρας 6000 του host υπολογιστή (η θύρα 6000 του host υπολογιστή έχει αντιστοιχηθεί στην θύρα 6000 του container).

Έπειτα ξεκινάμε την διαδικασία μεταφόρτωσης μιας νέας εικόνας από το

Docker Hub:

```
$ docker pull ubuntu
```

Αφού κατέβει η πιο πρόσφατη έκδοση του image, χρησιμοποιούμε την εντολή docker tag για να δώσουμε όνομα και να καθορίσουμε το registry στο οποίο θα βρίσκουμε από εδώ και στο εξής το συγκεκριμένο docker image.

```
$ docker tag ubuntu_latest \
localhost :6000/ ubuntu_latest -image -local -repo
```

Για να στείλουμε το τοπικό docker image στο registry που τρέχουμε αρκεί να δώσουμε την παρακάτω εντολή:

```
$ docker push \
localhost :6000/ ubuntu_latest -image -local -repo
```

Με παρόμοιο τρόπο μπορούμε να παραλάβουμε πίσω το image:

```
$ docker pull \
localhost :6000/ ubuntu_latest -image -local -repo
```

Ακολουθώντας τα παραπάνω βήματα μπορούμε να δημιουργήσουμε το δικό μας Registry και να μεταφορτώσουμε docker images πάνω σε αυτό. Εάν θέλουμε σε οποιαδήποτε στιγμή να σταματήσουμε τον εξυπηρετητή δίνουμε την εντολή:

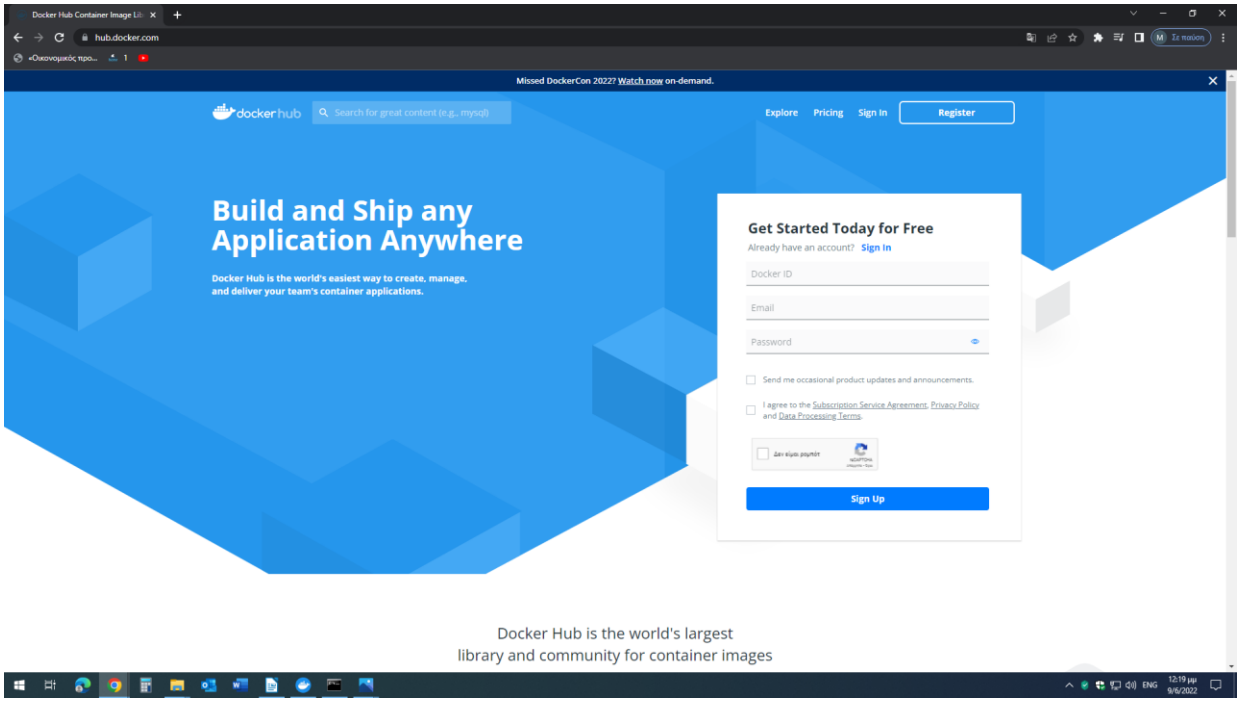
```
$ docker stop registry
```

Εάν επίσης θέλουμε να διαγράψουμε τα δεδομένα και να σβήσουμε το container, τότε αφού το έχουμε σταματήσει με την παραπάνω εντολή δίνουμε το παρακάτω: (χρησιμοποιώντας την παράμετρο -v θα σβήσουμε τα πιθανά volumes αποθηκευτικούς χώρους που έχουν δημιουργηθεί από αυτό το container):

```
$ docker rm –v my_own_registry
```

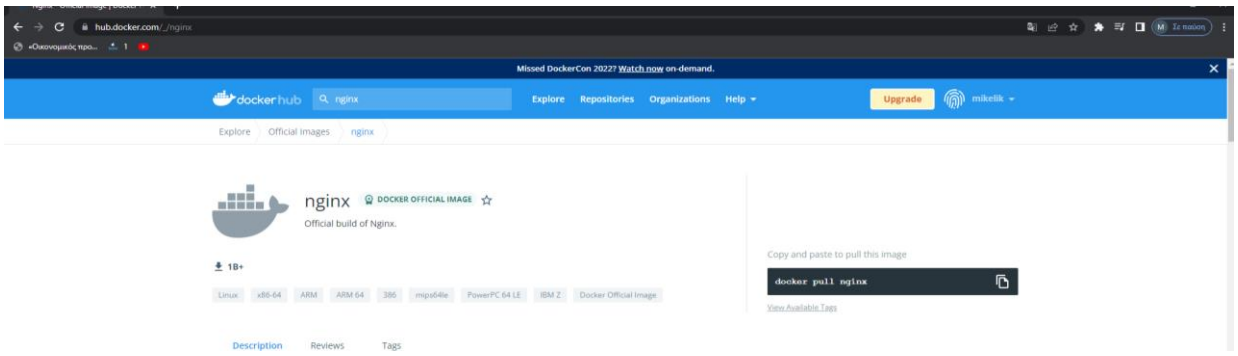
7.2 ΥΠΗΡΕΣΙΑ DOCKERHUB

Το Docker Hub είναι μια υπηρεσία Docker Registry βασισμένη στο cloud η οποία επιτρέπει την σύνδεση code repositories, την δημιουργία και τον έλεγχο images και συνδέεται με το Docker Cloud έτσι ώστε να γίνετε δυνατό το deployment αυτών των images στο cloud. Προσφέρει μια κεντρική πηγή πόρων για την ανακάλυψη διαφόρων images, συνεργασία μεταξύ ομάδων ή χρηστών και την αυτοματοποίηση της ροής των εργασιών στην διαδικασία ανάπτυξης εφαρμογών. Για την χρήση του Docker Hub χρειάζεται η δημιουργία ενός λογαριασμού στο Docker (Docker ID) και η αναζήτηση δημόσιων repositories ή images μπορεί να πραγματοποιηθεί με δύο τρόπους. Είτε με την εντολή docker search ή από το site του Docker Hub. To Docker επίσης παρέχει δυνατότητα πρόσβασης στις υπηρεσίες του Docker Hub με τις εντολές docker search (για αναζήτηση images), pull (για μεταφόρτωση images), login (για είσοδο με τα στοιχεία του λογαριασμού μας), και push (για ανέβασμα images).



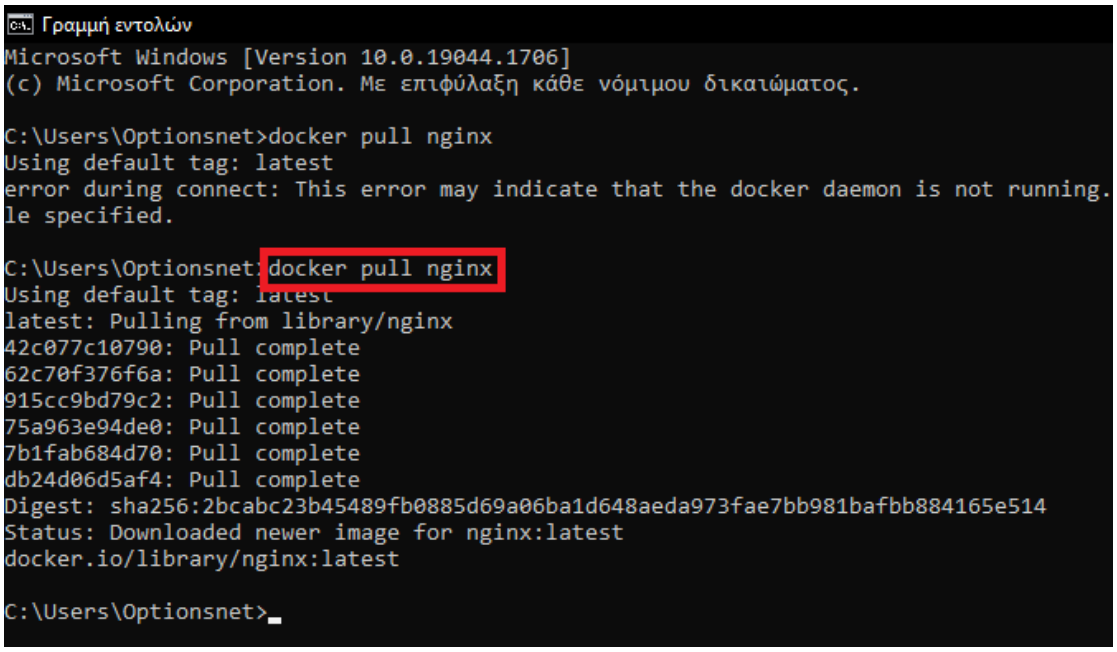
7.3 ΑΝΑΖΗΤΗΣΗ ΚΑΙ ΕΚΤΕΛΕΣΗ DOCKER IMAGE

Ανοίγουμε σε έναν browser την σελίδα του Docker Hub (https://hub.docker.com/) και στην μπάρα αναζήτησης πληκτρολογούμε το image που θέλουμε να χρησιμοποιήσουμε καθώς και τις σχετικές πληροφορίες και οδηγίες του. Στο συγκεκριμένο παράδειγμα το image που χρησιμοποιείται είναι το nginx.



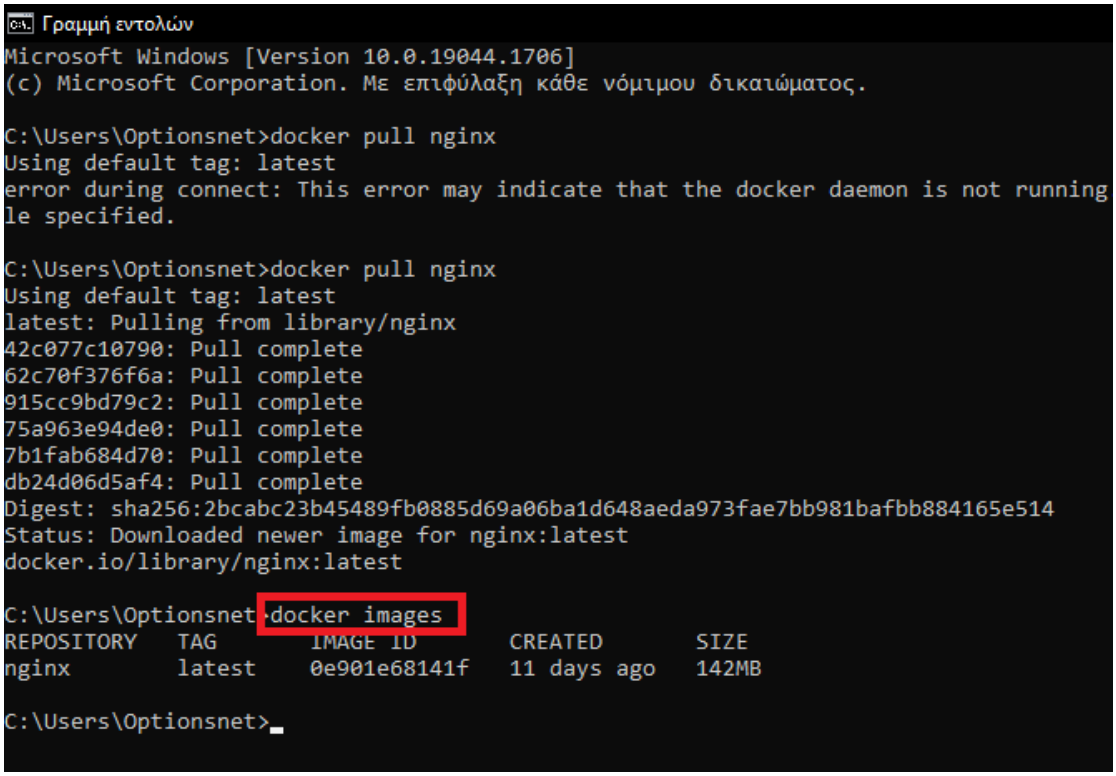
Στην συνέχεια ο τρόπος με τον οποίο κατεβάζουμε το image που έχουμε επιλέξει είναι με την εντολή:

```
docker pull nginx
```



Για να ελέγξουμε εάν το image που έχουμε επιλέξει έχει εγκατασταθεί με επιτυχία πληκτρολογούμε την παρακάτω εντολή στο τερματικό:

```
docker images
```



7.4 ΔΙΑΧΕΙΡΙΣΗ CONTAINER

7.4.1 ΕΚΚΙΝΗΣΗ DOCKER CONTAINER ΑΠΟ ΚΑΠΟΙΟ DOCKER IMAGE

Για την δημιουργία και την έναρξη ενός Docker container απαιτείται η χρήση κάποιου Docker Image, το οποίο όπως έχουμε προαναφέρει αποτελεί ουσιαστικά ένα σύστημα αρχείων (filesystem) με τις παραμέτρους που θα χρειαστούν για την εκτέλεση του. Το image δεν έχει κατάσταση και τα δεδομένα του παραμένουν πάντα σταθερά. Μια εκδοχή του image αποτελεί ένα container, για παράδειγμα όταν κατά την εγκατάσταση του Docker τρέξαμε την εντολή:

```
docker run hello-world
```

Το Docker έψαξε το σύστημα για να δει εάν έχουμε ήδη τοπικά το image με όνομα “hello-world” και δεν το βρήκε, για το λόγο αυτό το Docker παραπέμφθηκε στο Docker Hub με σκοπό να ψάξει να βρει την τελευταία έκδοση αυτού του image. Τέλος, αφού κατέβασε την τελευταία έκδοση την “φόρτωσε” σε ένα container και την εκτέλεσε. Το συγκεκριμένο image ήταν πολύ απλό και περιείχε μόνο μια εντολή, την οποία αφού εκτέλεσε πραγματοποίησε έξοδο και έκλεισε. Ο λόγος της απλότητας του Image ήταν απλά για να δοκιμάσουμε εάν όλα τα βασικά χαρακτηριστικά του Docker λειτουργούν σωστά στο σύστημα μας. Υπάρχουν πολύ πιο περίπλοκα Docker Images που μπορούν να κάνουν πολύ περισσότερα πράγματα από την εκτέλεση μιας εντολής.

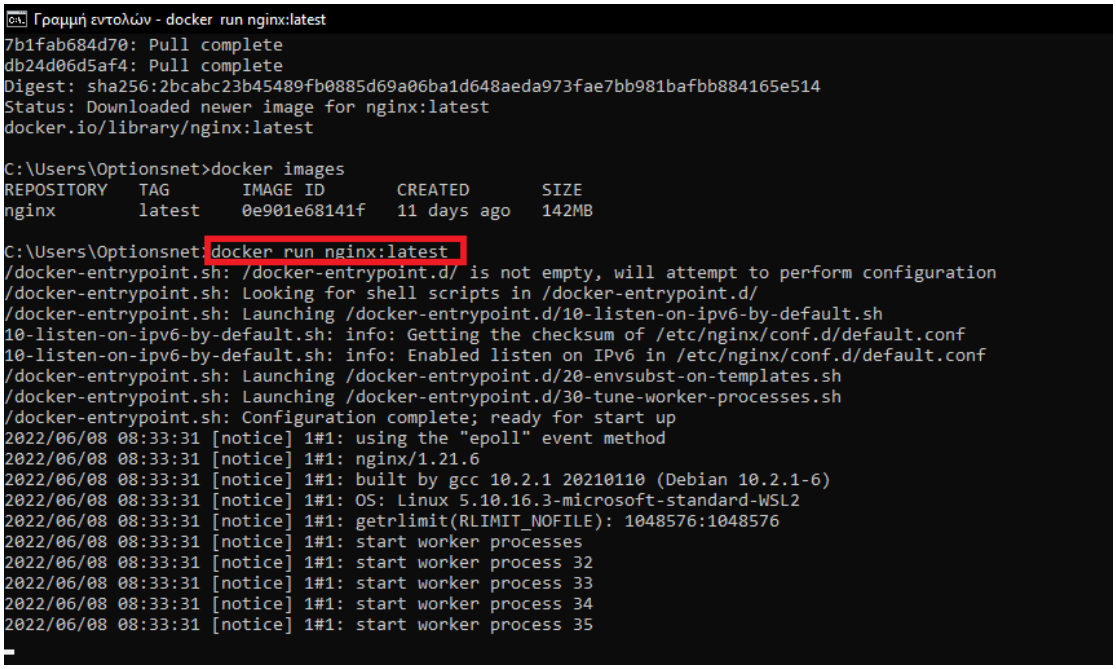
Για να τρέξουμε ένα νέο Docker Container από κάποιο Image θα πρέπει να φτιάξουμε είτε ένα δικό μας Image, είτε να χρησιμοποιήσουμε κάποιο private Docker registry server ή να κατεβάσουμε κάποιο Image από το Official Docker Registry το Docker Hub. Κάτι το οποίο το προαναφέραμε στην ενότητα 7.3, με τις εντολές:

```
docker pull nginx
```

```
docker images
```

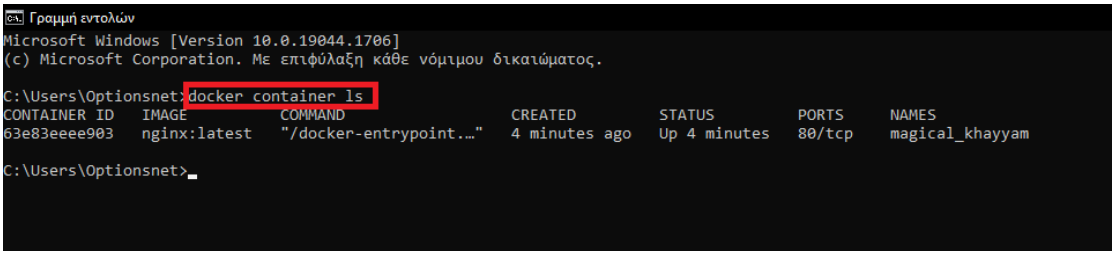
Για να τρέξουμε το nginx ωστόσο χρειάζεται ακόμα ένα container που να αποτελεί μηχανισμό διαχείρισης βάσης δεδομένων MySQL. Για να κατεβάσουμε το image αυτό πληκτρολογούμε την παρακάτω εντολή:

```
docker run nginx:latest
```



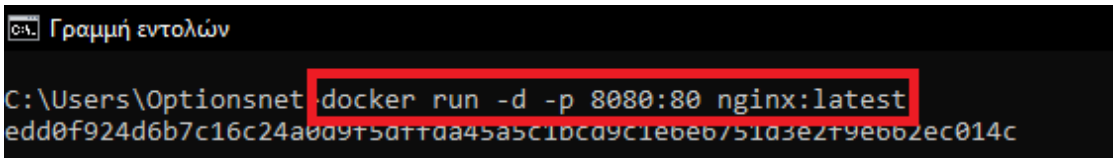
Για να ελέγξουμε πόσα container "τρέχουν" στο σύστημα μας μία χρήσιμη εντολή είναι:

```
docker container ls
```



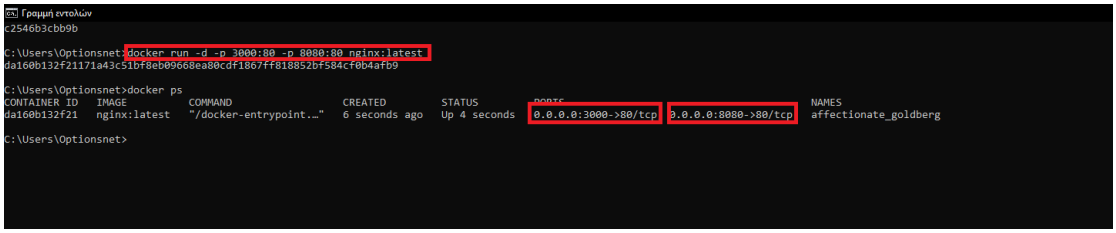
Πλέον παρατηρούμε ότι το image που έχουμε επιλέξει υπάρχει εσωτερικά στο σύστημα μας. Οπότε για να δούμε τις υπηρεσίες που παρέχει το συγκεκριμένο image πληκτρολογούμε την παρακάτω εντολή στο τερματικό μας χρησιμοποιώντας ένα φυλλομετρητή στην διεύθυνση **http://localhost:8080** με βάση την θύρα που δώσαμε στην παράμετρο -p (port).

```
docker run -d -p 8080:80 nginx:latest
```



Επιπλέον μας δίνετε η δυνατότητα να εκτελούμε το image που έχουμε επιλέξει σε φυλλομετρητές διαφορετικών διευθύνσεων. Αυτό επιτυγχάνετε με την παρακάτω εντολή:

```
docker run -d -p 3000:80 -p 8080:80 nginx:latest
```



Για να βρούμε τα containers που εκτελούνται στο σύστημα μια δεδομένη χρονική στιγμή μπορούμε να χρησιμοποιήσουμε την εντολή:

```
docker ps
```

Στην περίπτωση που θέλουμε να σταματήσουμε το container πληκτρολογούμε την παρακάτω εντολή:

```
docker stop <<container id>>
```

7.4.2 ΕΝΤΟΛΕΣ ΔΙΑΧΕΙΡΙΣΗΣ ΤΩΝ CONTAINER ΕΣΩΤΕΡΙΚΑ

Το Docker παρέχει μεγάλο πλήθος εντολών για την διαχείριση των containers, και θα δούμε μερικές από τις πιο βασικές που χρησιμοποιούνται:

- **attach:** Σύνδεση με ένα container που εκτελείται αυτήν τη στιγμή, πιο συγκεκριμένα με το STDOUT της κύριας διαδικασίας του container.
- **commit:** Μια εικόνα δημιουργείται αλλάζοντας ένα υπάρχον container για την παραγωγή μιας νέας εικόνας.
- **cp:** Μεταφέρετε αρχεία από το τοπικό σύστημα στο σύστημα αρχείων του container.
- **create:** Δημιουργεί ένα νέο container. Η διαφορά μεταξύ αυτής της εντολής και της εντολής εκτέλεσης είναι ότι η εντολή δημιουργίας δημιουργεί απλώς το container χωρίς να το εκτελεί. Στη συνέχεια, το container θα πρέπει να εκτελεστεί με την εντολή start docker, ενώ η εντολή εκτέλεσης δημιουργεί και εκκινεί απευθείας το container.

- **diff:** Εμφανίζει τις εντολές που αποστέλλονται σε ένα container που τρέχει και σε ένα container που δεν έχει αλλάξει από τότε που δημιουργήθηκε από την εικόνα του.

- **exec:** Εκτελεί μια εντολή σε ένα container που λειτουργεί.

- **export:** Δημιουργεί ένα αρχείο tar εξάγοντας το σύστημα αρχείων ενός container.

- **inspect:** Παρέχει μια λεπτομερή ανάλυση των στοιχείων που συνθέτουν το σύστημα και τους πόρους ενός container.

- **kill:** Βάζει τέλος σε ένα ή περισσότερα container.

- **logs:** Δείχνει τα αρχεία καταγραφής των περιεχομένων ενός container.

- **ls:** Λειτουργεί παρόμοια με την εκτέλεση της εντολής docker ps και εμφανίζει μια λίστα με container.

- **pause:** Θέτει σε παύση όλες τις διεργασίες που εκτελούνται αυτήν τη στιγμή σε ένα ή περισσότερα container.

- **port:** Παραθέτει τις θύρες κεντρικού υπολογιστή που αντιστοιχίζονται στις θύρες container.

- **prune:** Την εξάλειψη όλων των δοχείων που δεν λειτουργούν.

- **rename:** Αλλάζει το όνομα ενός container.

- **restart:** Προκαλεί την επανεκκίνηση ενός ή περισσότερων container.

- **rm:** Δυνατότητα αφαίρεσης container.
- **run:** Στέλνει μια εντολή που θα εκτελεστεί σε νέο container.

- **start:** Επανεκκινεί ένα ή περισσότερα σταματημένα container.

- **stats:** Δείχνει μια ζωντανή ροή των πόρων που χρησιμοποιούν τα container.

- **stop:** Σταματά ένα ή περισσότερα container που τρέχουν.

- **top:** Δείχνει τις ενεργές διεργασίες μέσα σε ένα container.

- **unpause:** Καταργεί την παύση τυχόν διεργασιών που είναι ενεργές αυτήν τη στιγμή σε ένα container.

- **update:** Ένα container που τροποποιεί τις παραμέτρους του

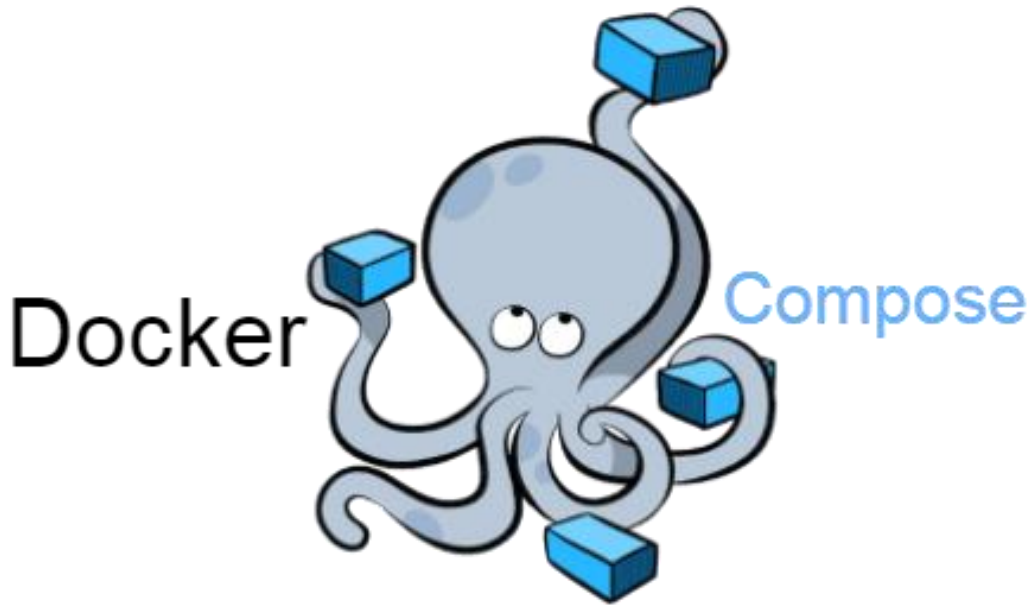
- **wait:** Παγώνει την εκτέλεση εντολών στο τερματικό έως ότου σταματήσουν ένα ή περισσότερα container και μετά εκτυπώνει τον κωδικό εξόδου του καθενός. Αυτός ο κωδικός περιγράφει την περίσταση που οδήγησε στην έξοδο ενός container, η οποία μπορεί να ήταν φυσιολογική ή αποτέλεσμα σφάλματος (όπου εκτυπώνει τον κωδικό σφάλματος).

7.5 DOCKER COMPOSE

Το Docker Compose είναι λογισμικό που χρησιμοποιείται για τον καθορισμό και την εκτέλεση εφαρμογών Docker πολλαπλών container. Μπορεί να χειριστεί πολλαπλά container ταυτόχρονα στο περιβάλλον παραγωγής, τοποθέτησης, ανάπτυξης, δοκιμής και CI. Για να χρησιμοποιήσουμε αυτό το εργαλείο πρέπει να φτιάξουμε ένα .yml αρχείο όπου θα έχουμε διατυπώσει τις απαιτήσεις που χρειάζεται η εφαρμογή μας και έπειτα με μια εντολή δημιουργούμε και ενεργοποιούμε όλες τις υπηρεσίες με βάση την παραμετροποίηση που έχουμε κάνει στο .yml αρχείο.

Πλεονεκτήματα του Docker Compose:

- **Ανάπτυξη ενός κεντρικού υπολογιστή:** Αυτό σημαίνει ότι μπορείτε να εκτελέσετε τα πάντα σε ένα μόνο κομμάτι υλικού.
- **Γρήγορη και εύκολη διαμόρφωση:** Λόγω των σεναρίων YAML.
- **Υψηλή παραγωγικότητα:** Το Docker Compose μειώνει τον χρόνο που απαιτείται για την εκτέλεση εργασιών.
- **Ασφάλεια:** Όλα τα container είναι απομονωμένα μεταξύ τους, μειώνοντας το τοπίο απειλής.



7.5.1 ΔΗΜΙΟΥΡΓΙΑ DOCKER COMPOSE FILE

Σε αυτό το κομμάτι του κεφαλαίου θα ακολουθηθεί βήμα προς βήμα η διαδικασία δημιουργίας ενός docker compose αρχείου η οποία απαρτίζεται από την δομή ενός .yml αρχείου στο οποίο θα καθορίσουμε τα services που χρειάζονται για την δημιουργία του image που έχουμε επιλέξει. Για την συγκεκριμένη αναπαράσταση έχουμε επιλέξει το RabbitMQ για image.

- **ΒΗΜΑ 1ο :**

Δημιουργούμε ένα φάκελο στον οποίο θα περιέχεται το .yaml αρχείο μας, και στην συνέχεια ανοίγουμε ένα source code editor της επιλογής μας για να γράψουμε τις ιδιότητες του .yml αρχείου μας.

docker-compose.yml

```
version: '3.2'
services:
  rabbitmq:
    image: rabbitmq:3-management-alpine
    container_name: 'rabbitmq'
    ports:
      - 5672:5672
      - 15672:15672
    volumes:
      - ~/.docker-conf/rabbitmq/data/:/var/lib/rabbitmq
      - ~/.docker-conf/rabbitmq/log/:/var/log/rabbitmq
    networks:
      - rabbitmq_go_net

networks:
  rabbitmq_go_net:
    driver: bridge
```

Η τελική εικόνα του .yml αρχείου μας θα πρέπει να είναι κάπως έτσι. Με καθορισμένη version του image και έπειτα με την διαδικασία services οι ιδιότητες του.

- **ΒΗΜΑ 2º :**

Αφού αποθηκεύσουμε το .yml αρχείο ανοίγουμε ένα terminal πλοηγούμαστε στον φάκελο που περιέχει το .yml αρχείο και πληκτρολογούμε την παρακάτω εντολή:

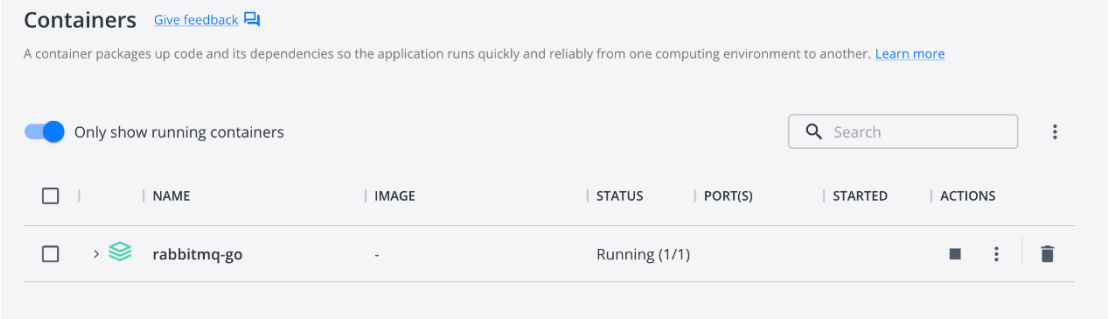
```
docker compose up
```

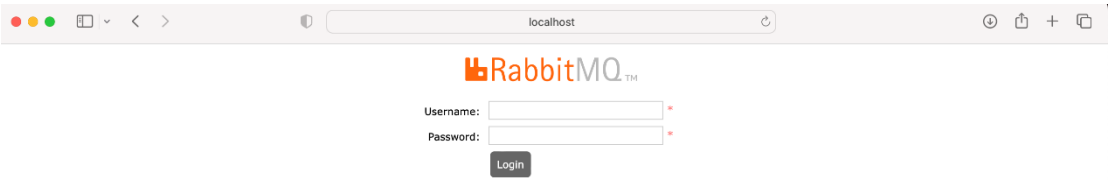
```
geokarou@Air-Giorgos rabbitmq-go % docker-compose up
[+] Running 2/11
 ⠸ rabbitmq Pulling                                                              19.4s
   ⠿ 261da4162673 Pull complete                                                   2.4s
   ⠿ cff7551a4e41 Pull complete                                                   2.7s
   ⠼ 539c3454f133 Downloading  [=================================>        ...    16.4s
   ⠼ 209f27449c9f Download complete                                              16.4s
   ⠼ 874da42f996e Download complete                                              16.4s
   ⠼ 16477288985b Download complete                                              16.4s
   ⠼ 90640758ba85 Download complete                                              16.4s
   ⠼ b2b5de31642e Download complete                                              16.4s
   ⠼ 43aa8717dd0f Download complete                                              16.4s
   ⠼ 85e3fbbc296f Downloading  [==============>                          ...    16.4s
```

```
rabbitmq  | 2022-12-03 17:02:19.824495+00:00 [info] <0.229.0> Will seed default virtual host and user...
rabbitmq  | 2022-12-03 17:02:19.824539+00:00 [info] <0.229.0> Adding vhost '/' (description: 'Default virtual host', tags: [])
rabbitmq  | 2022-12-03 17:02:19.825534+00:00 [info] <0.229.0> Applying default limits to vhost '<<"/">>': []
rabbitmq  | 2022-12-03 17:02:19.832239+00:00 [info] <0.674.0> Making sure data directory '/var/lib/rabbitmq/mnesia/rabbit@0dc96cd17aa1/msg_stores/vhosts/628WB79CIFDYO9LJI6DKMI09L' for vhost '/' exists
rabbitmq  | 2022-12-03 17:02:19.852843+00:00 [info] <0.674.0> Setting segment_entry_count for vhost '/' with 0 queues to '2048'
rabbitmq  | 2022-12-03 17:02:19.861099+00:00 [info] <0.674.0> Starting message stores for vhost '/'
rabbitmq  | 2022-12-03 17:02:19.861250+00:00 [info] <0.679.0> Message store "628WB79CIFDYO9LJI6DKMI09L/msg_store_transient": using rabbit_msg_store_ets_index to provide index
rabbitmq  | 2022-12-03 17:02:19.876005+00:00 [info] <0.674.0> Started message store of type transient for vhost '/'
rabbitmq  | 2022-12-03 17:02:19.876245+00:00 [info] <0.683.0> Message store "628WB79CIFDYO9LJI6DKMI09L/msg_store_persistent": using rabbit_msg_store_ets_index to provide index
rabbitmq  | 2022-12-03 17:02:19.883397+00:00 [warning] <0.683.0> Message store "628WB79CIFDYO9LJI6DKMI09L/msg_store_persistent": rebuilding indices from scratch
rabbitmq  | 2022-12-03 17:02:19.889193+00:00 [info] <0.674.0> Started message store of type persistent for vhost '/'
rabbitmq  | 2022-12-03 17:02:19.889320+00:00 [info] <0.674.0> Recovering 0 queues of type rabbit_classic_queue took 33ms
rabbitmq  | 2022-12-03 17:02:19.889346+00:00 [info] <0.674.0> Recovering 0 queues of type rabbit_quorum_queue took 0ms
rabbitmq  | 2022-12-03 17:02:19.889373+00:00 [info] <0.674.0> Recovering 0 queues of type rabbit_stream_queue took 0ms
rabbitmq  | 2022-12-03 17:02:19.892201+00:00 [info] <0.229.0> Created user 'guest'
rabbitmq  | 2022-12-03 17:02:19.893195+00:00 [info] <0.229.0> Successfully set user tags for user 'guest' to [administrator]
rabbitmq  | 2022-12-03 17:02:19.894150+00:00 [info] <0.229.0> Successfully set permissions for 'guest' in virtual host '/' to '.*', '.*', '.*'
rabbitmq  | 2022-12-03 17:02:19.894201+00:00 [info] <0.229.0> Running boot step rabbit_observer_cli defined by app rabbit
rabbitmq  | 2022-12-03 17:02:19.894236+00:00 [info] <0.229.0> Running boot step rabbit_looking_glass defined by app rabbit
rabbitmq  | 2022-12-03 17:02:19.894253+00:00 [info] <0.229.0> Running boot step rabbit_core_metrics_gc defined by app rabbit
rabbitmq  | 2022-12-03 17:02:19.894316+00:00 [info] <0.229.0> Running boot step background_gc defined by app rabbit
rabbitmq  | 2022-12-03 17:02:19.894429+00:00 [info] <0.229.0> Running boot step routing_ready defined by app rabbit
rabbitmq  | 2022-12-03 17:02:19.894466+00:00 [info] <0.229.0> Running boot step pre_flight defined by app rabbit
rabbitmq  | 2022-12-03 17:02:19.894485+00:00 [info] <0.229.0> Running boot step notify_cluster defined by app rabbit
rabbitmq  | 2022-12-03 17:02:19.894501+00:00 [info] <0.229.0> Running boot step networking defined by app rabbit
rabbitmq  | 2022-12-03 17:02:19.894523+00:00 [info] <0.229.0> Running boot step definition_import_worker_pool defined by app rabbit
rabbitmq  | 2022-12-03 17:02:19.894562+00:00 [info] <0.286.0> Starting worker pool 'definition_import_pool' with 4 processes in it
rabbitmq  | 2022-12-03 17:02:19.894755+00:00 [info] <0.229.0> Running boot step cluster_name defined by app rabbit
rabbitmq  | 2022-12-03 17:02:19.894802+00:00 [info] <0.229.0> Initialising internal cluster ID to 'rabbitmq-cluster-id-pHeg50esB_E2Z-_ameuvkA'
rabbitmq  | 2022-12-03 17:02:19.896201+00:00 [info] <0.229.0> Running boot step direct_client defined by app rabbit
rabbitmq  | 2022-12-03 17:02:19.896279+00:00 [info] <0.229.0> Running boot step rabbit_maintenance_mode_state defined by app rabbit
rabbitmq  | 2022-12-03 17:02:19.896300+00:00 [info] <0.229.0> Creating table rabbit_node_maintenance_states for maintenance mode status
rabbitmq  | 2022-12-03 17:02:19.914160+00:00 [info] <0.229.0> Running boot step rabbit_management_load_definitions defined by app rabbitmq_management
rabbitmq  | 2022-12-03 17:02:19.914277+00:00 [info] <0.720.0> Resetting node maintenance status
rabbitmq  | 2022-12-03 17:02:19.922251+00:00 [info] <0.779.0> Management plugin: HTTP (non-TLS) listener started on port 15672
rabbitmq  | 2022-12-03 17:02:19.922321+00:00 [info] <0.807.0> Statistics database started.
rabbitmq  | 2022-12-03 17:02:19.922357+00:00 [info] <0.806.0> Starting worker pool 'management_worker_pool' with 3 processes in it
rabbitmq  | 2022-12-03 17:02:19.927452+00:00 [info] <0.821.0> Prometheus metrics: HTTP (non-TLS) listener started on port 15692
rabbitmq  | 2022-12-03 17:02:19.927539+00:00 [info] <0.720.0> Ready to start client connection listeners
rabbitmq  | 2022-12-03 17:02:19.929605+00:00 [info] <0.865.0> started TCP listener on [::]:5672
rabbitmq  | 2022-12-03 17:02:20.034123+00:00 [info] <0.720.0> Server startup complete; 4 plugins started.
rabbitmq  | 2022-12-03 17:02:20.034123+00:00 [info] <0.720.0>  * rabbitmq_prometheus
rabbitmq  | 2022-12-03 17:02:20.034123+00:00 [info] <0.720.0>  * rabbitmq_management
rabbitmq  | 2022-12-03 17:02:20.034123+00:00 [info] <0.720.0>  * rabbitmq_web_dispatch
rabbitmq  | 2022-12-03 17:02:20.034123+00:00 [info] <0.720.0>  * rabbitmq_management_agent
rabbitmq  |  completed with 4 plugins.
```

➢ **ΒΗΜΑ 3ο :**

Με την ολοκλήρωση της εντολής που τρέξαμε ανοίγοντας το Dockerhub παρατηρούμε ότι δημιουργήθηκε το container του RabbitMQ. Στην συνέχεια ανοίγουμε έναν φυλλομετρητή και πλοηγούμαστε στην διεύθυνση που έχουμε ορίσει **http://localhost:15672** Εάν έχουν πάει όλα όπως πρέπει τα αποτελέσματα που πρέπει να έχουμε στην οθόνη μας είναι τα εξής.





Για να τερματίσουμε την διαδικασία του docker compose πληκτρολογούμε την εντολή:

```
docker compose down
```

ΚΕΦΑΛΑΙΟ 8 :

ΑΝΑΠΤΥΞΗ ΒΑΣΗΣ ΔΕΔΟΜΕΝΩΝ ΜΕΣΩ DOCKER

Ο σκοπός αυτού του κεφαλαίου σε συνδυασμό με τις θεωρητικές πληροφορίες που αναφέραμε στα προηγούμενα κεφάλαια είναι η δημιουργία και η ανάπτυξη δικών μας microservices όπου η διαχείριση τους θα γίνετε μέσω Docker. Σε πρώτο στάδιο θα αναπτύξουμε την εφαρμογή μας με ξεχωριστά images συνδέοντας τα χειροκίνητα ένα-ένα και στην πορεία του κεφαλαίου θα προσπαθήσουμε να αυτοματοποιήσουμε αυτή την διαδικασία χρησιμοποιώντας εργαλεία διαχείρισης όπως το docker-compose. Προτού προχωρήσουμε σε αυτή τη διαδικασία θα γίνει μια θεωρητική αναφορά σε κάποιες λέξεις- όρους που θα μας χρειαστούν για την κατανόηση και εξέλιξη των υπηρεσιών μας.

8.1 API

Τα API[13] (Application Programming Interface) είναι μηχανισμοί που επιτρέπουν σε δύο στοιχεία λογισμικού να επικοινωνούν μεταξύ τους χρησιμοποιώντας ένα σύνολο ορισμών και πρωτοκόλλων. Για παράδειγμα, το σύστημα λογισμικού του μετεωρολογικού γραφείου περιέχει καθημερινά δεδομένα καιρού. Η εφαρμογή καιρού στο τηλέφωνό σας «μιλάει» σε αυτό το σύστημα μέσω API και σας δείχνει καθημερινές ενημερώσεις καιρού στο τηλέφωνό σας.

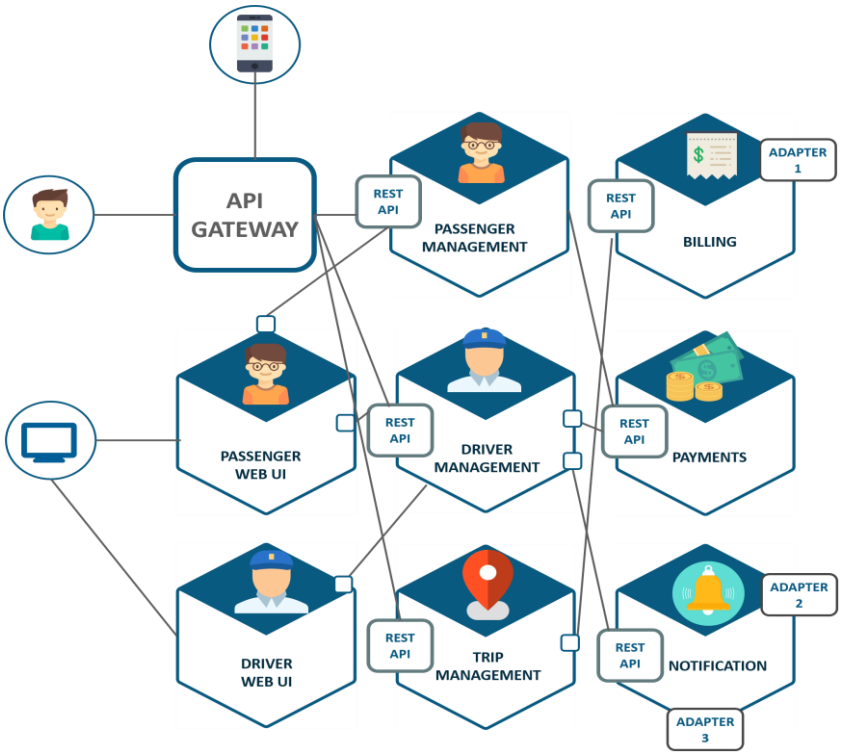
[13] https://aws.amazon.com/what-is/api/

8.2 MICROSERVICES

Τα microservices [14] είναι μια αρχιτεκτονική και οργανωτική προσέγγιση στην ανάπτυξη λογισμικού όπου το λογισμικό αποτελείται από μικρές ανεξάρτητες υπηρεσίες που επικοινωνούν μέσω καλά καθορισμένων API. Αυτές οι υπηρεσίες ανήκουν σε μικρές, αυτόνομες ομάδες.

Οι αρχιτεκτονικές microservices διευκολύνουν την κλίμακα και ταχύτερη ανάπτυξη των εφαρμογών, επιτρέποντας την καινοτομία και επιταχύνοντας την κυκλοφορία νέων χαρακτηριστικών στην αγορά. Κάποια από τα πλεονεκτήματα των microservices είναι:

- **Πιο απλό στην ανάπτυξη:** Αναπτύξτε σε κυριολεκτικά κομμάτια χωρίς να επηρεάζετε άλλες υπηρεσίες.
- **Πιο απλό στην κατανόηση:** Ο κώδικας είναι πιο εύκολο να ακολουθηθεί, καθώς η συνάρτηση είναι απομονωμένη και λιγότερο εξαρτημένη.
- **Επαναχρησιμοποιήσιμο σε όλες τις επιχειρήσεις:** Μοιραστείτε μικρές υπηρεσίες, όπως συστήματα πληρωμής ή σύνδεσης σε όλη την επιχείρηση.
- **Ταχύτερη απομόνωση ελαττώματος:** Όταν μια δοκιμή αποτύχει ή η υπηρεσία πέσει, απομονώστε την γρήγορα.
- **Ελαχιστοποίηση του κινδύνου από την αλλαγή:** Αποφύγετε το κλείδωμα σε τεχνολογίες ή γλώσσες - αλλάξτε εν κινήσει με ελάχιστο κίνδυνο.



[14] https://smartbear.com/solutions/microservices/

8.3 ΕΓΚΑΤΑΣΤΑΣΗ ΒΑΣΗΣ ΔΕΔΟΜΕΝΩΝ ΧΕΙΡΟΚΙΝΗΤΑ

Σε αυτό το κομμάτι του κεφαλαίου θα ακολουθηθεί βήμα προς βήμα η διαδικασία εγκατάστασης της βάσης δεδομένων και ενός database administration system μέσω Docker η οποία θα μας χρειαστεί για την αποθήκευση και την πρόσβαση στα δεδομένα που θα μας δίνει το API που θα επιλέξουμε για την αναπαράσταση της επικοινωνίας των container μέσω docker χειροκίνητα-'μη αυτοματοποιημένα'.

- **ΒΗΜΑ 1ο :**

Ανοίγουμε ένα terminal και πληκτρολογούμε την παρακάτω εντολή.

```
docker pull mariadb:latest
```

```
(latest=latest version of database)
```

```
geokarou@Air-Giorgos ~ % docker pull mariadb:latest
latest: Pulling from library/mariadb
0509fae36eb0: Pull complete
cf568f815bdc: Pull complete
4ba9b40f35bf: Pull complete
e0343c2247ca: Pull complete
2ede61e3bdcf: Pull complete
104d70805c98: Pull complete
c90746ee41c2: Pull complete
68df37186922: Pull complete
Digest: sha256:940985c1cf37812ffb3bb6c7b34b4e40233e0907fc786ec7d63c49553d7d1454
Status: Downloaded newer image for mariadb:latest
docker.io/library/mariadb:latest
geokarou@Air-Giorgos ~ %
```

- **ΒΗΜΑ 2ο :**

Στην συνέχεια πατώντας την παρακάτω εντολή φτιάχνουμε το container της βάσης και ορίζουμε τα credentials της. Δηλαδή όνομα βάσης και κωδικό.

```
docker run --name myDB-docker -e
MARIADB_ROOT_PASSWORD=mike2619 -d mariadb:latest
```

```
geokarou@Air-Giorgos ~ % docker run --name myDB-docker -e MARIADB_ROOT_PASSWORD=mike2619 -d mariadb:latest
4bc35b5849a585d15e560461de329ab962989386038593341 1f27501d0cf2449
geokarou@Air-Giorgos ~ %
```

- **ΒΗΜΑ 3ο :**

Αφού φτιάξαμε το container της βάσης με την παρακάτω εντολή θα κατεβάσουμε το database administration system που στην προκειμένη περίπτωση έχουμε επιλέξει το phpMyAdmin.

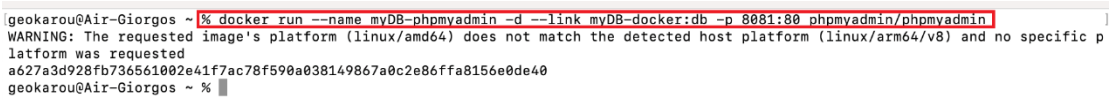
```
docker pull phpmyadmin/phpmyadmin:latest
```

```
geokarou@Air-Giorgos ~ % docker pull phpmyadmin/phpmyadmin:latest
latest: Pulling from phpmyadmin/phpmyadmin
214ca5fb9032: Pull complete
cd813a1b2cb8: Pull complete
63cf7574573d: Pull complete
54c27146d16e: Pull complete
078f4450f949: Pull complete
5f145e355bc4: Pull complete
fdc797cb9eea: Pull complete
af45e7153a31: Pull complete
b546fbaf263b: Pull complete
16dd2cabbcd2: Pull complete
30a426b49280: Pull complete
c94e73d5f13e: Pull complete
2f5a3464a278: Pull complete
a4f9f723c297: Pull complete
5b04d16a8086: Pull complete
2a3d1fa22772: Pull complete
ef56affc4552: Pull complete
9b9b44731108: Pull complete
Digest: sha256:ae6dadd9cf3c158e42937788f7255fa820ea3daef0349226d8d43f32e76535e1
Status: Downloaded newer image for phpmyadmin/phpmyadmin:latest
docker.io/phpmyadmin/phpmyadmin:latest
geokarou@Air-Giorgos ~ %
```

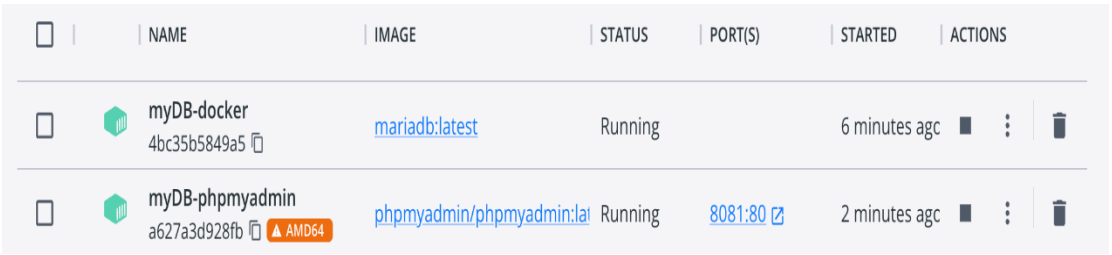
- **ΒΗΜΑ 4ο :**

Τελευταίο βήμα είναι η υλοποίηση του container του database administration system και ο καθορισμός της διεύθυνσης που θα οδηγεί στην βάση τοπικά.

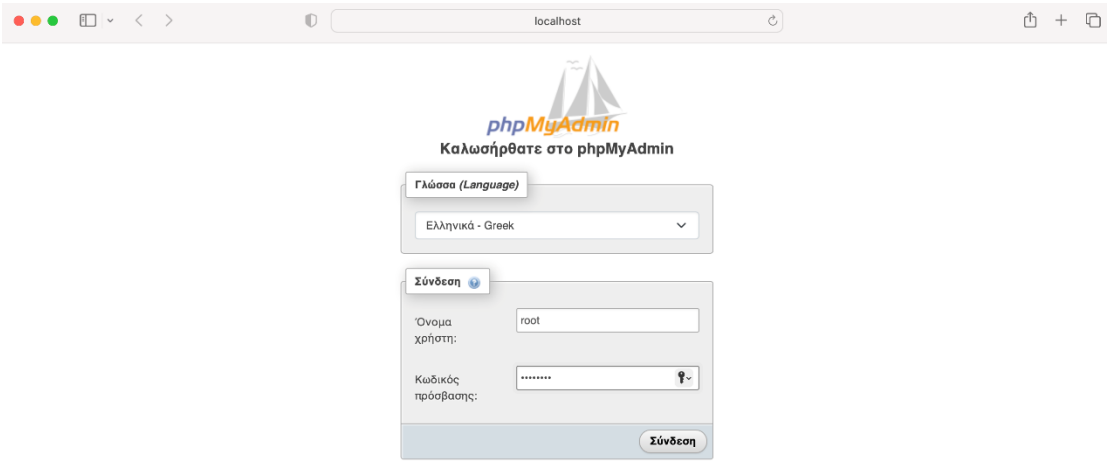
```
docker run --name myDB-phpmyadmin -d --link
myDB-docker:db -p 8081:80 phpmyadmin/phpMyAdmin
```

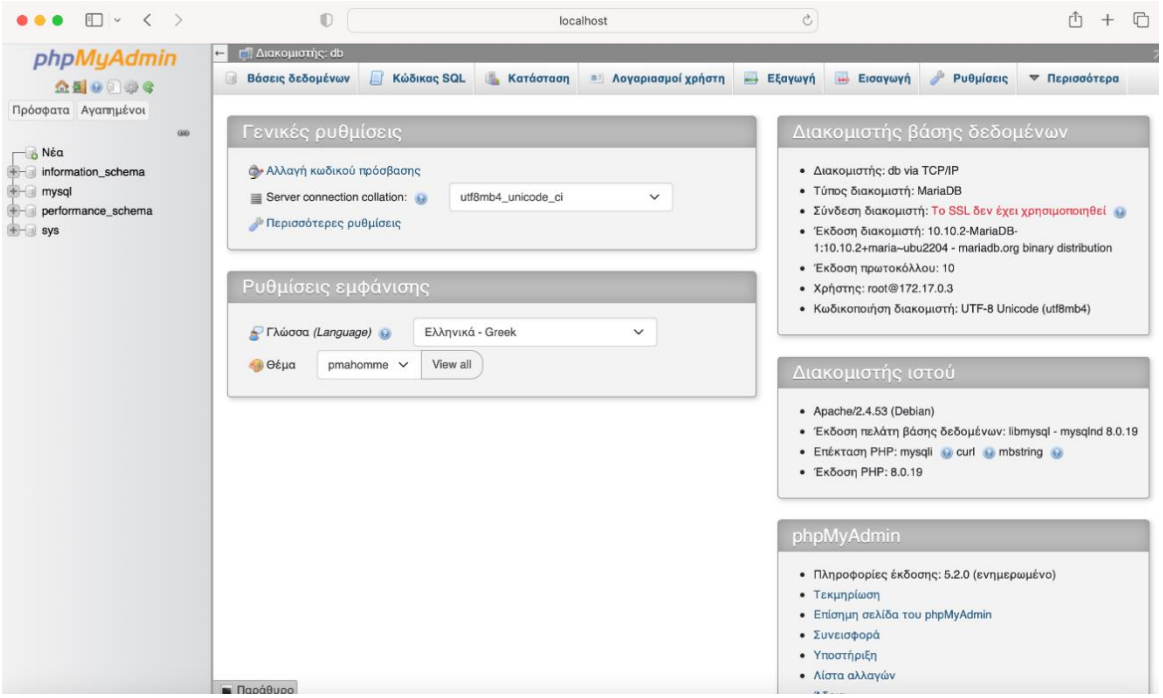


Παρατηρούμε ότι έχουν δημιουργηθεί και τα δύο containers που χρειαζόμαστε.



Επομένως εφόσον έχουμε εκτελέσει τα προαναφερθέντα βήματα σωστά θα πρέπει ανοίγοντας έναν φυλλομετρητή και πατώντας **http://localhost:8081** αναμένουμε να δούμε αυτό το αποτέλεσμα.

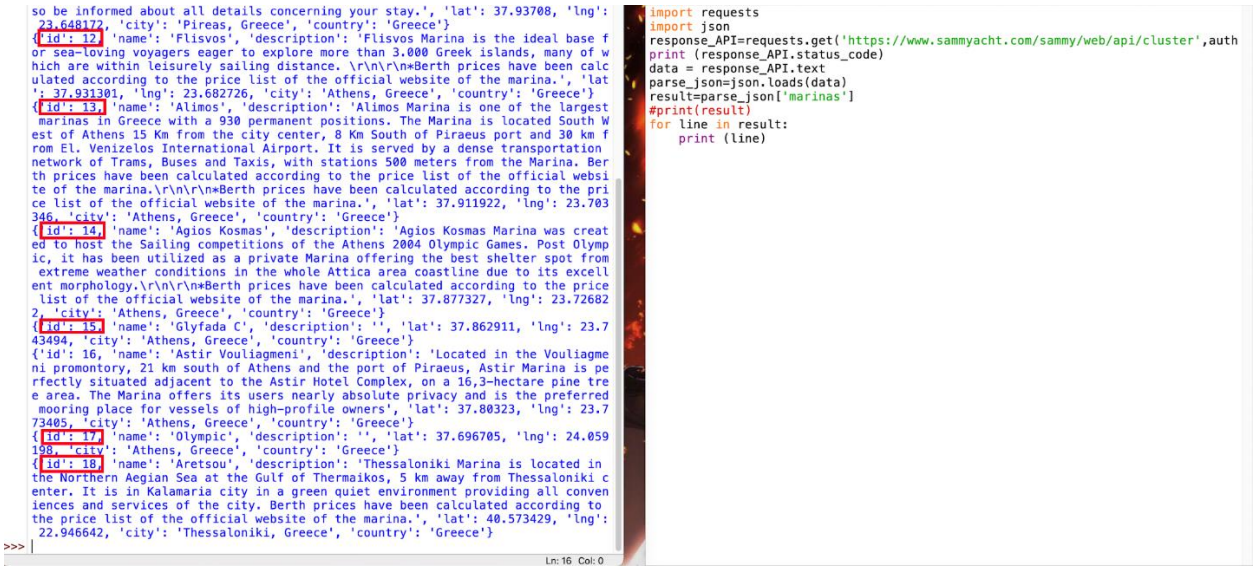




- **ΒΗΜΑ 5ο :**

Η διαδικασία που μας μένει είναι η δημιουργία του container όπου θα περιέχει το python script το οποίο θα συλλέγει τα δεδομένα από το API που θα επιλέξουμε και θα τα καταχωρεί στην βάση. Το API που έχουμε επιλέξει είναι ενδεικτικό αναπαριστώντας κάποιες μαρίνες της Ελλάδας με κάποιες πληροφορίες τους όπως (όνομα, πόλη, γεωγραφική θέση, σύντομη περιγραφή).

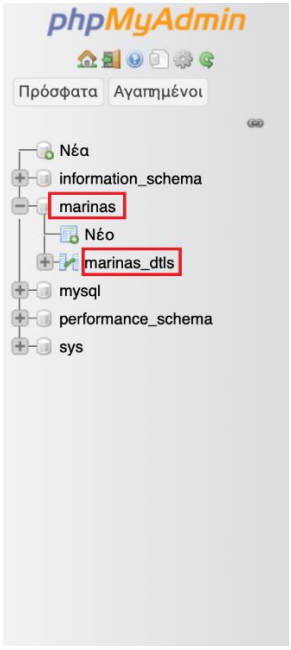
Python Script: Αρχικά εξασφαλίσαμε ότι λαμβάνουμε τα δεδομένα που μας παρέχει το API.



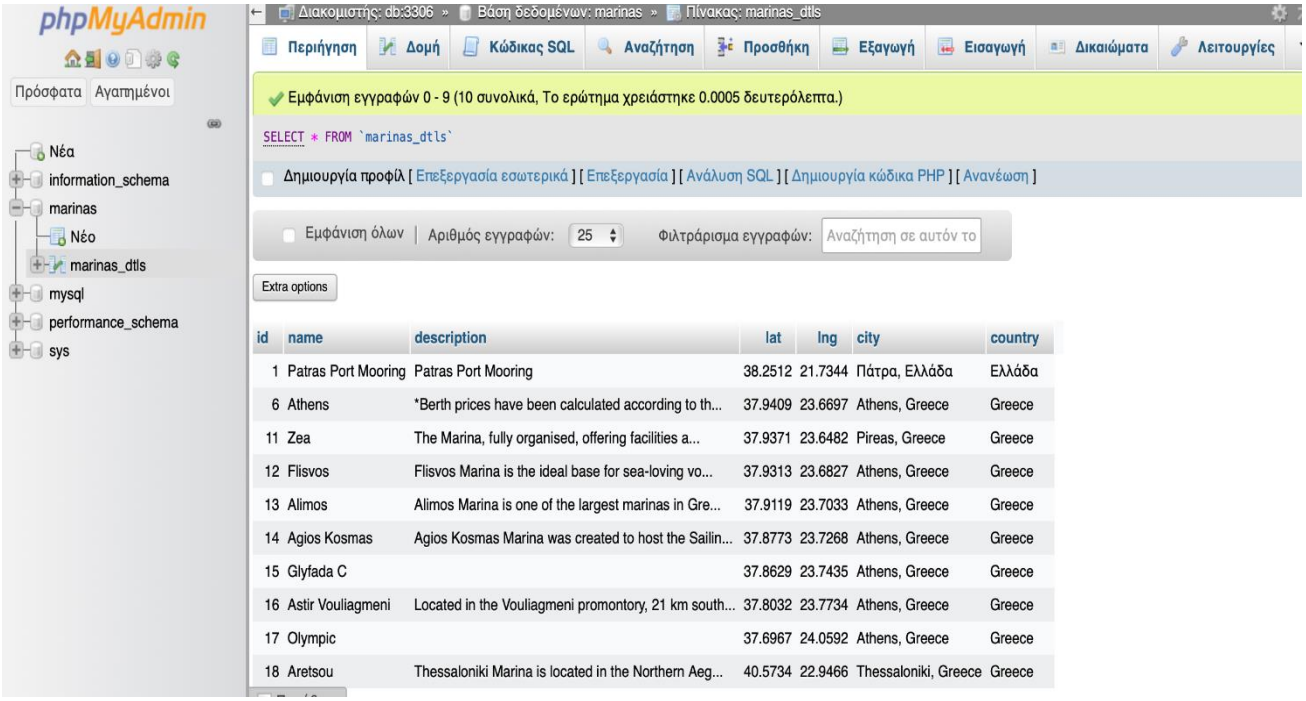
Στην συνέχεια αφού είχαμε την απάντηση του API φτιάξαμε τον πίνακα που θα καταχωρούνται οι πληροφορίες των μαρίνων αποθηκεύοντας τες στην βάση δεδομένων που δημιουργήσαμε.

```
import requests
import json
import os
import mysql.connector

response_API=requests.get('https://www.sammyacht.com/sammy/web/api/cluster',auth=('Basic', 'bWFyaW5hc2Nhbjp0N1U7OSVGTQ'))
print (response_API.status_code)
data = response_API.text
parse_json=json.loads(data)
result=parse_json['marinas']

try:
    connection=mysql.connector.connect(
        user='root',
        password="mike2619",
        host="db",
        database="marinas")
    print("success")
except Exception as inst:
    print("fail",inst)


cursor=connection.cursor()
cursor.execute("CREATE TABLE marinas_dtls (id INT(11) NOT NULL , name VARCHAR(255) NOT NULL , description VARCHAR(650) NOT
print("table created")
for row in result:
    for key,val in row.items():
        exec(key + '=val')
    print (id,name, len(description),lat,lng,city,country)
    sql = "INSERT INTO marinas_dtls (id, name, description, lat, lng, city, country) VALUES (%s, %s, %s, %s, %s, %s, %s)"

    val = (id,name,description,lat,lng,city,country)
    cursor.execute(sql,val)
    print("row inserted")
connection.commit()

connection.close()

    |
```

Αφού ολοκληρώθηκε και η δημιουργία του python script που μας εξασφαλίζει την επικοινωνία της βάσης δεδομένων με το API ανοίγουμε έναν φυλλομετρητή πατώντας **http://localhost:8081** εισάγουμε τα credentials που έχουμε ορίσει στο phpMyAdmin interface και το αποτέλεσμα που θα πρέπει να αντικρίζουμε θα είναι το παρακάτω.



Παρατηρούμε το όνομα της βάσης μας (marinas) και ο πίνακας (marinas_dtls) έχουν δημιουργηθεί με επιτυχία. Πατώντας στον πίνακα διαπιστώνουμε ότι τα δεδομένα του API περάστηκαν στον πίνακα με επιτυχία.



8.4 ΑΥΤΟΜΑΤΟΠΟΙΗΣΗ ΒΑΣΗΣ ΔΕΔΟΜΕΝΩΝ

Στο προηγούμενο κομμάτι αυτού του κεφαλαίου δημιουργήσαμε χειροκίνητα τα container που χρειάζονται για την αναπαράσταση της σύνδεσης της βάσης δεδομένων με το API που είχαμε επιλέξει.

Αυτός ο τρόπος είναι λειτουργικός αλλά δεν είναι αυτοματοποιημένος για δυο λόγους:

- Αναγκαζόμαστε να ανοίγουμε τα container ένα-ένα κάθε φορά που θέλουμε να εισέλθουμε στην βάση.
- Δεν έχουμε την δυνατότητα πρόσβασης στην βάση σε περίπτωση που είμαστε συνδεδεμένοι από άλλο υπολογιστή. Με αποτέλεσμα να στερούμαστε την κύρια υπηρεσία που μας παρέχει το docker.

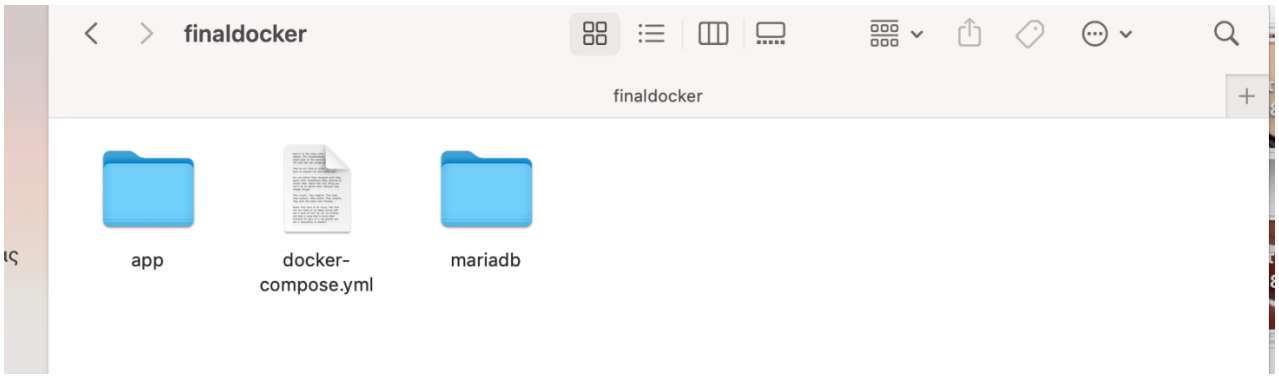
Επομένως για να επιλύσουμε αυτό το ‘πρόβλημα’ θα πρέπει να αυτοματοποιήσουμε την βάση δεδομένων. Αυτό το πετυχαίνουμε με την χρήση του docker-compose που έχουμε αναφέρει στο κεφάλαιο 7 κάνοντας την εφαρμογή μας δυναμική.

- **ΒΗΜΑ 1^{ο} :**

  Δημιουργία ενός φακέλου που περιέχει όλα τα αρχεία που απαιτούνται για την εφαρμογή μας.

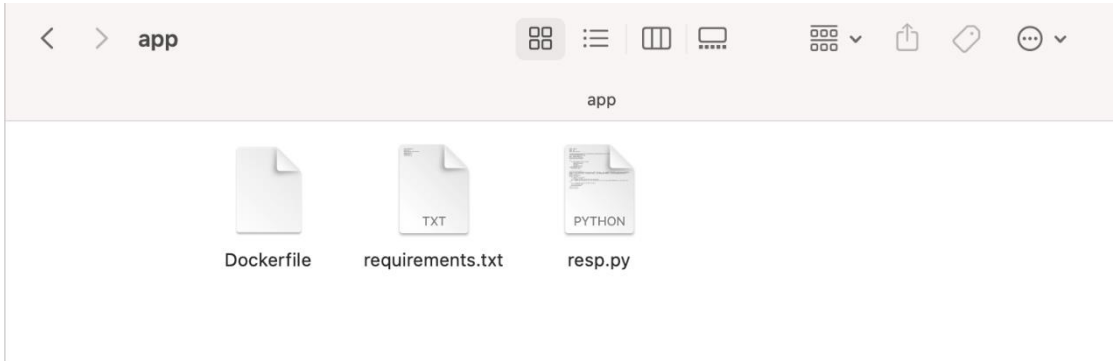
- **ΒΗΜΑ 2ο :**

Στον κύριο φάκελο που υλοποιήσαμε θα δημιουργήσουμε άλλους δυο φακέλους που θα αναπαριστούν τα container που χρειάζονται για την αυτοματοποίηση της εφαρμογής.



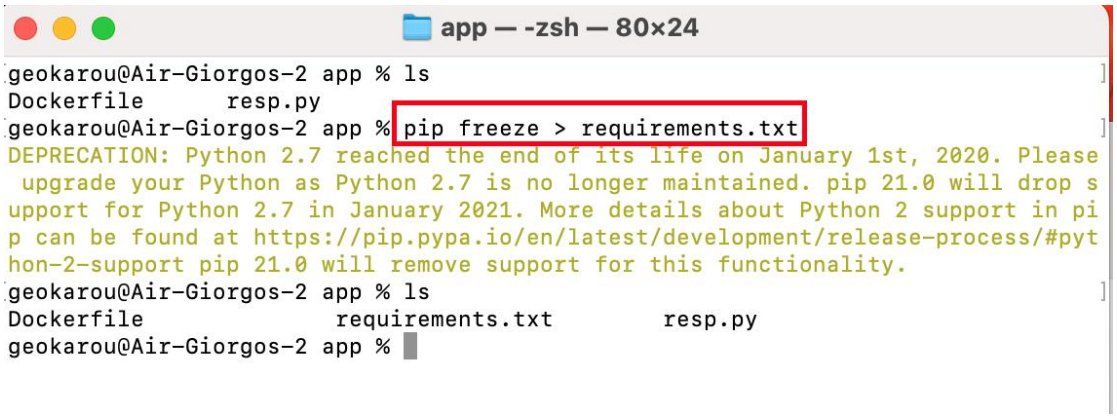
Οι δυο φάκελοι που δημιουργήσαμε οι οποίοι αναπαριστούν τα container της εφαρμογής περιέχουν τα dockerfiles και λοιπά scripts που απαιτούνται για την σωστή λειτουργία της τα οποία θα τα δούμε αναλυτικά παρακάτω.

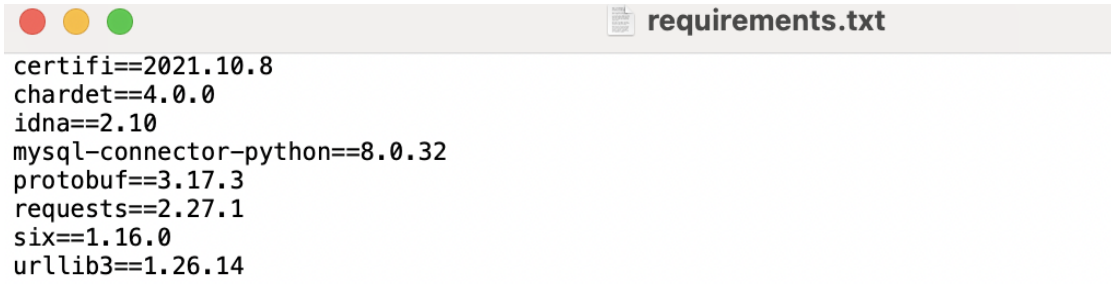
Περιεχόμενα φάκελου app:



- **requirements.txt**

  Το παραπάνω αρχείο μας χρειάζεται για την αναγνώριση των απαιτούμενων και το δημιουργήσαμε εκτελώντας την παρακάτω εντολή στο τερματικό.





- **resp.py**

  Το παραπάνω αρχείο είναι το python script που εισάγει τα δεδομένα του API στην βάση δεδομένων το οποίο χρησιμοποιήσαμε και στην ενότητα 8.3 βήμα 5ο.

- **Dockerfile:**

  Στο παραπάνω αρχείο καθορίζονται η python version που χρησιμοποιείται, ο φάκελος και αντιγράφεται το αρχείο requirements.txt στο project μας εγκαθιστώντας όλα τα requirements που αναφέραμε στο αντίστοιχο αρχείο.

Dockerfile — Έ

```
# syntax=docker/dockerfile:1

FROM python:3.11.0a1-slim

WORKDIR /app

COPY requirements.txt requirements.txt

RUN pip3 install -r requirements.txt


COPY . .
```

Περιεχόμενα φάκελου mariadb:

- **Dockerfile:**

  Στο παραπάνω αρχείο αντιγράφονται αρχεία και ρυθμίσεις που αφορούν την βάση δεδομένων.

Dockerfile

```
FROM Mariadb:latest

ADD init.sql /docker-entrypoint-initdb.d

COPY my.cnf /etc/mysql/my.cnf

HEALTHCHECK CMD ["healthcheck.sh", "--su=mysql", "--connect", "--innodb_initialized"]
```

- **ΒΗΜΑ 3ο :**

Τελευταίο βήμα για να ολοκληρώσουμε την αυτοματοποίηση της εφαρμογής μας είναι να δημιουργήσουμε ένα docker-compose.yml αρχείο.

Καθορίζουμε τις ρυθμίσεις του phpMyAdmin όπως (όνομα, image, port).

```yaml
version: '3'
services:
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    container_name: phpdocker
    links:
      - db
    restart: on-failure
    ports:
      - "8081:80"
    environment:
      PMA_HOST: db
      PMA_PORT: 3306
      PMA_ARBITRARY: 1
    depends_on:
      - db
    hostname: phpmyadmin
```

Αντίστοιχα καθορίζουμε τις ρυθμίσεις της βάσης όπως (όνομα, image, port, credentials).

```
19    db:
20      image: mariadb:latest
21      build:
22        context: ./mariadb
23        dockerfile: Dockerfile
24      container_name: db
25      ports:
26        - "13306:3306"
27      restart: unless-stopped
28      environment:
29        MARIADB_ROOT_PASSWORD: mike2619
30        MARIADB_DATABASE: marinas
31        MARIADB_USER: root
32        MARIADB_PASSWORD: mike2619
33      hostname: db
```

Τέλος καθορίζουμε τις ρυθμίσεις του φακέλου app όπως (όνομα, εκτέλεση αρχείου resp.py με Python, επανεκκίνηση σε περίπτωση αποτυχίας).

```
35    app:
36      build:
37       context: ./app
38       dockerfile: Dockerfile
39      hostname: app
40      container_name: app
41      depends_on:
42        - db
43      command: ["python", "resp.py"]
44      restart: on-failure
45
46
47
```

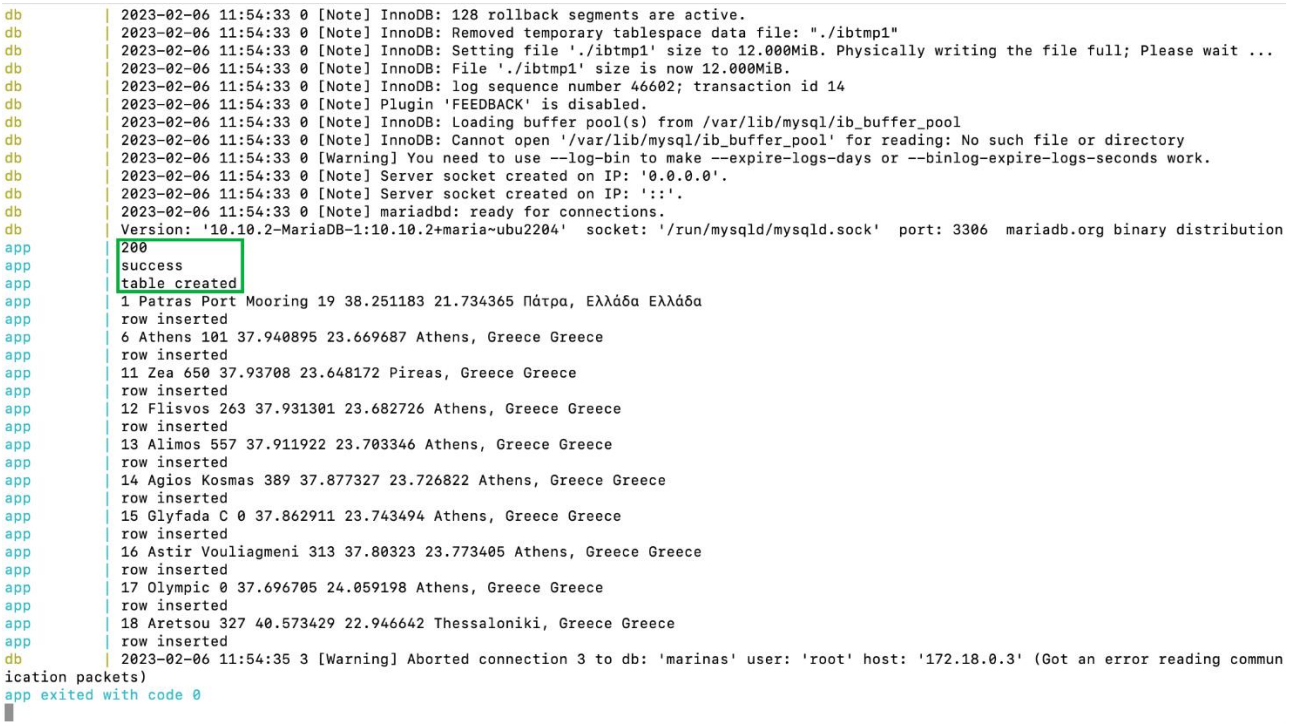
Κατά συνέπεια εφόσον έχουμε ολοκληρώσει τα προαναφερθέντα βήματα σωστά θα πρέπει ανοίγοντας ένα τερματικό και εκτελώντας τις παρακάτω εντολές θα πρέπει να έχουμε αυτό το αποτέλεσμα στην οθόνη μας:

```
docker-compose build

docker-compose up
```

```
geokarou@Air-Giorgos-2 finaldocker % sudo docker-compose up
[+] Running 4/2
 ⠿ Network finaldocker_default
              Created 0.0s                                                    0.0s
 ⠿ Container db
              Created 0.0s
 ⠿ Container app
              Created 0.0s
 ⠿ Container phpdocker
              Created 0.0s
 ⠿ phpmyadmin The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no specific platfor
was requested  0.0s
Attaching to app, db, phpdocker
db          | 2023-02-06 11:54:29+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.10.2+maria~ubu2204 started.
db          | 2023-02-06 11:54:30+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
db          | 2023-02-06 11:54:30+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.10.2+maria~ubu2204 started.
db          | 2023-02-06 11:54:30+00:00 [Note] [Entrypoint]: Initializing database files
phpdocker   | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.18.0.4. Set the 'ServerName'
rective globally to suppress this message
phpdocker   | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.18.0.4. Set the 'ServerName'
rective globally to suppress this message
db          |
db          |
db          | PLEASE REMEMBER TO SET A PASSWORD FOR THE MariaDB root USER !
db          | To do so, start the server, then issue the following command:
db          |
db          | '/usr/bin/mysql_secure_installation'
db          |
db          | which will also give you the option of removing the test
db          | databases and anonymous user created by default.  This is
db          | strongly recommended for production servers.
db          |
db          | See the MariaDB Knowledgebase at https://mariadb.com/kb
db          |
db          | Please report any problems at https://mariadb.org/jira
db          |
db          | The latest information about MariaDB is available at https://mariadb.org/.
db          |
db          | Consider joining MariaDB's strong and vibrant community:
db          | https://mariadb.org/get-involved/
```

```
db          |
db          | 2023-02-06 11:54:30+00:00 [Note] [Entrypoint]: Database files initialized
db          | 2023-02-06 11:54:30+00:00 [Note] [Entrypoint]: Starting temporary server
db          | 2023-02-06 11:54:31+00:00 [Note] [Entrypoint]: Waiting for server startup
db          | 2023-02-06 11:54:31 0 [Note] mariadbd (server 10.10.2-MariaDB-1:10.10.2+maria~ubu2204) starting as process 96 ...
db          | 2023-02-06 11:54:31 0 [Note] InnoDB: Compressed tables use zlib 1.2.11
db          | 2023-02-06 11:54:31 0 [Note] InnoDB: Number of transaction pools: 1
db          | 2023-02-06 11:54:31 0 [Note] InnoDB: Using ARMv8 crc32 + pmull instructions
db          | 2023-02-06 11:54:31 0 [Note] mariadbd: O_TMPFILE is not supported on /tmp (disabling future attempts)
db          | 2023-02-06 11:54:31 0 [Note] InnoDB: Using liburing
db          | 2023-02-06 11:54:31 0 [Note] InnoDB: Initializing buffer pool, total size = 128.000MiB, chunk size = 2.000MiB
db          | 2023-02-06 11:54:31 0 [Note] InnoDB: Completed initialization of buffer pool
db          | 2023-02-06 11:54:31 0 [Note] InnoDB: File system buffers for log disabled (block size=512 bytes)
db          | 2023-02-06 11:54:31 0 [Note] InnoDB: 128 rollback segments are active.
db          | 2023-02-06 11:54:31 0 [Note] InnoDB: Setting file './ibtmp1' size to 12.000MiB. Physically writing the file full; Please wait
db          | 2023-02-06 11:54:31 0 [Note] InnoDB: File './ibtmp1' size is now 12.000MiB.
db          | 2023-02-06 11:54:31 0 [Note] InnoDB: log sequence number 45452; transaction id 14
db          | 2023-02-06 11:54:31 0 [Note] Plugin 'FEEDBACK' is disabled.
db          | 2023-02-06 11:54:31 0 [Warning] 'user' entry 'root@db' ignored in --skip-name-resolve mode.
db          | 2023-02-06 11:54:31 0 [Warning] 'proxies_priv' entry '@% root@db' ignored in --skip-name-resolve mode.
db          | 2023-02-06 11:54:31 0 [Note] mariadbd: ready for connections.
db          | Version: '10.10.2-MariaDB-1:10.10.2+maria~ubu2204'  socket: '/run/mysqld/mysqld.sock'  port: 0  mariadb.org binary distributi
phpdocker   | [Mon Feb 06 11:54:31.079790 2023] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.53 (Debian) PHP/8.0.19 configured -- resum
ormal operations
phpdocker   | [Mon Feb 06 11:54:31.081543 2023] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'
app         | 200
app         | fail 2003: Can't connect to MySQL server on 'db:3306' (111 Connection refused)
app         | Traceback (most recent call last):
app         |   File "/app/resp.py", line 23, in <module>
app         |     cursor=connection.cursor()
app         |           ^^^^^^^^^^
app         | NameError: name 'connection' is not defined
app exited with code 1
```

Όπως παρατηρούμε στην παραπάνω εικόνα μας επιστρέφει ένα fail 2003. Ο λόγος που συμβαίνει αυτό είναι γιατί δεν έχει γίνει εκκίνηση της βάσης με αποτέλεσμα να γίνετε επανεκκίνηση του app συνεχώς μέχρι να εκκινήσει. Όταν η βάση είναι έτοιμη γίνετε η σύνδεση και έχουμε το παρακάτω αποτέλεσμα στην εικόνα.

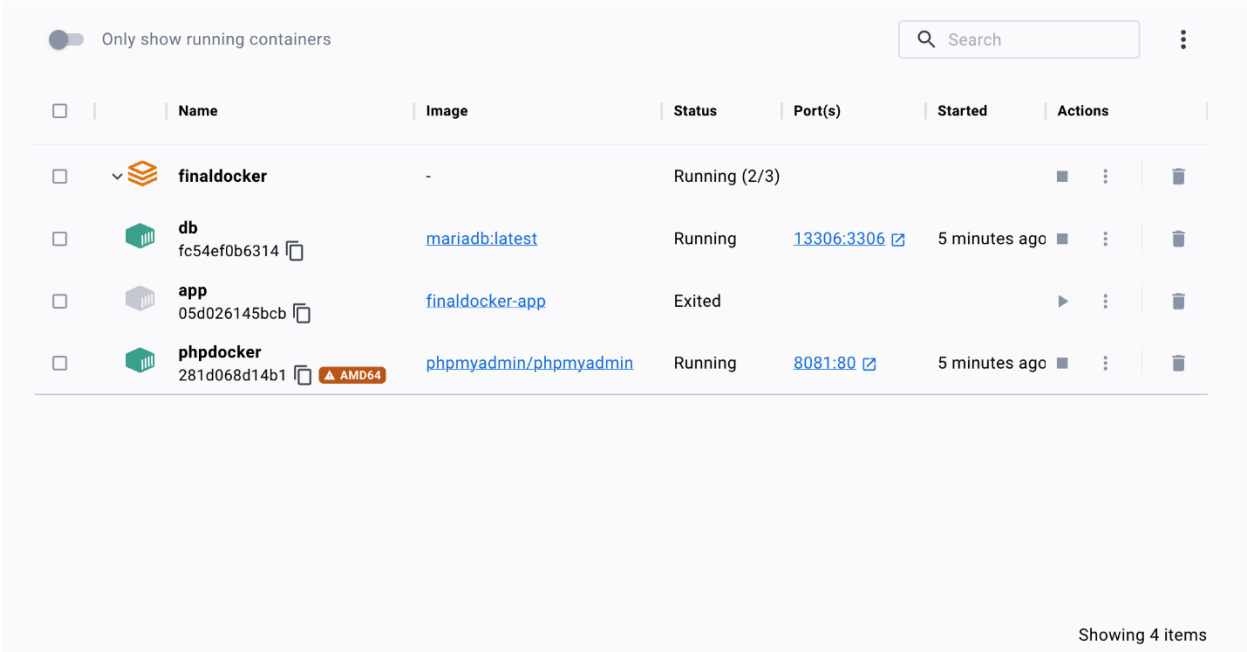


Παρατηρούμε ότι μόλις γίνετε η σύνδεση της βάσης έχουμε τα παρακάτω αποτελέσματα:

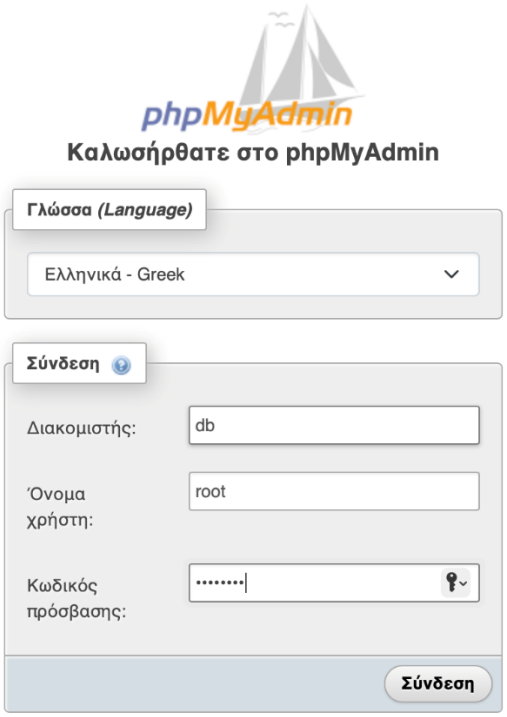
- **200:** Κατάφερε να τραβήξει δεδομένα από το API.
- **Success:** Πέτυχε η σύνδεση με την βάση δεδομένων.
- **Table created:** Δημιουργήθηκε ο πίνακας.
- Για κάθε γραμμή που κάνει insert επιστρέφει μήνυμα για την επιτυχή εισαγωγή καθώς και πληροφορίες που τράβηξε από το API. Λόγω μεγάλου μεγέθους για το description επιστρέφει μόνο το μέγεθος του string ώστε να έχουμε και εικόνα τι μέγεθος να δώσουμε στο πεδίο της βάσης κατά τη δημιουργία της.

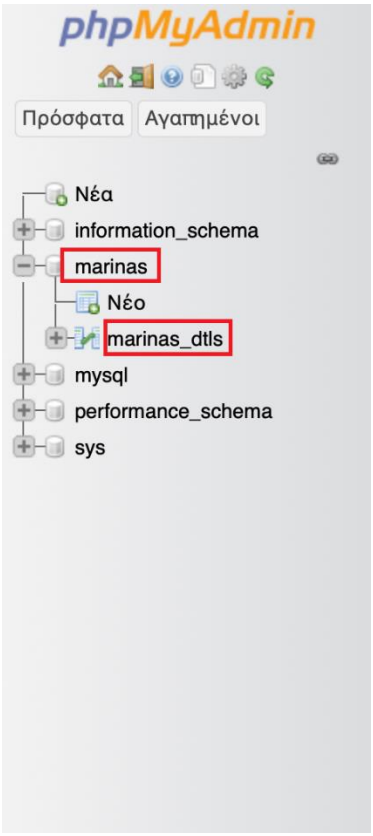
Tip: Ένας τρόπος για να μην βγάζει το fail 2003 είναι να τρέξουμε ένα script (wait-for-it.sh) το οποίο αναμένει μέχρι να ενεργοποιηθεί το κατάλληλο port και μετά εκτελεί τον python κώδικα. Ακολουθεί παράδειγμα παρακάτω.

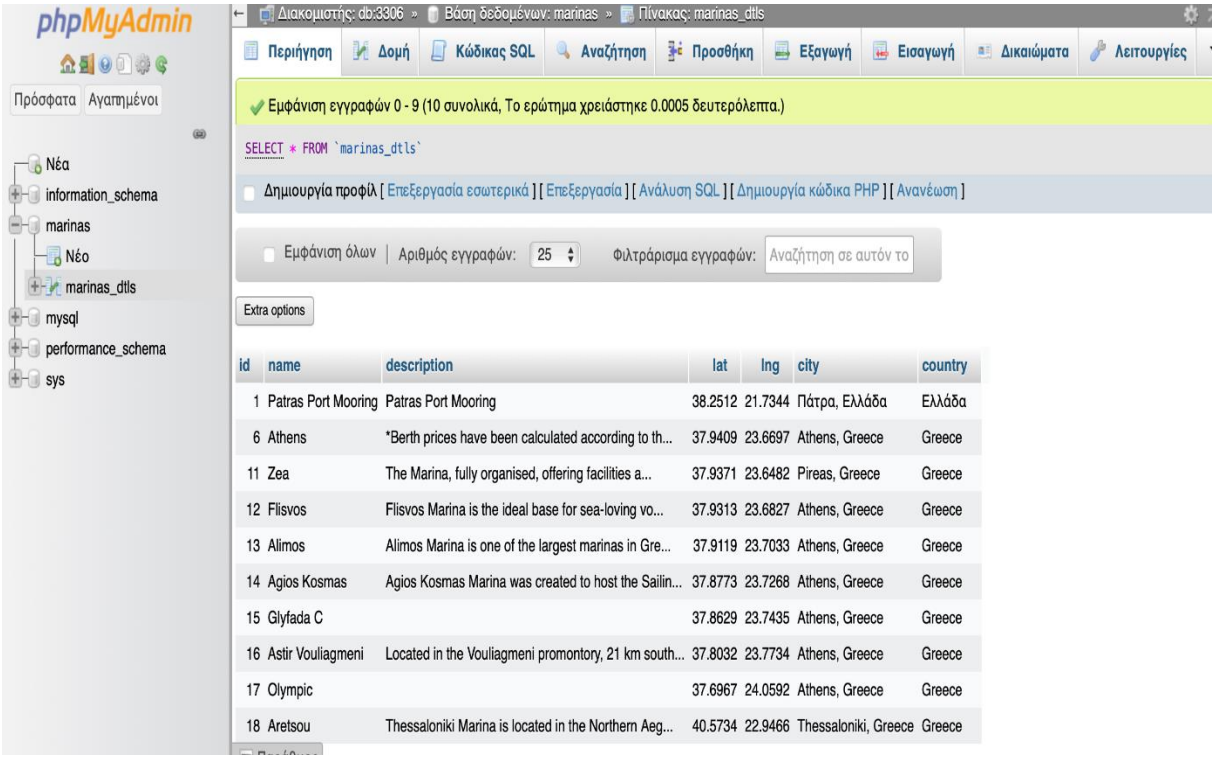
Επιπροσθέτως με την εκτέλεση της εντολής **docker-compose build** βλέπουμε ότι δημιουργήθηκαν τα container που θέλαμε για την αυτοματοποίηση της εφαρμογής μας.



Οπότε για να σιγουρευτούμε πλήρως ότι η εφαρμογή μας πλέον είναι αυτοματοποιημένη-δυναμική ανοίγουμε έναν φυλλομετρητή και πατώντας **http://localhost:8081** έχουμε το παρακάτω αποτέλεσμα.




phpMyAdmin
Πρόσφατα
Αγαπημένοι
Νέα
information_schema
marinas
Νέο
marinas_dtls
mysql
performance_schema
sys


phpMyAdmin
Διακομιστής: db:3306 » Βάση δεδομένων: marinas » Πίνακας: marinas_dtls
Περιήγηση
Δομή
Κώδικας SQL
Αναζήτηση
Προσθήκη
Εξαγωγή
Εισαγωγή
Δικαιώματα
Λειτουργίες
Εμφάνιση εγγραφών 0 - 9 (10 συνολικά, Το ερώτημα χρειάστηκε 0.0005 δευτερόλεπτα.)
SELECT * FROM `marinas_dtls`
Δημιουργία προφίλ [Επεξεργασία εσωτερικά] [Επεξεργασία] [Ανάλυση SQL] [Δημιουργία κώδικα PHP] [Ανανέωση]
Εμφάνιση όλων | Αριθμός εγγραφών: 25 Φιλτράρισμα εγγραφών: Αναζήτηση σε αυτόν το
Extra options
id name description lat lng city country
1 Patras Port Mooring Patras Port Mooring 38.2512 21.7344 Πάτρα, Ελλάδα Ελλάδα
6 Athens *Berth prices have been calculated according to th... 37.9409 23.6697 Athens, Greece Greece
11 Zea The Marina, fully organised, offering facilities a... 37.9371 23.6482 Pireas, Greece Greece
12 Flisvos Flisvos Marina is the ideal base for sea-loving vo... 37.9313 23.6827 Athens, Greece Greece
13 Alimos Alimos Marina is one of the largest marinas in Gre... 37.9119 23.7033 Athens, Greece Greece
14 Agios Kosmas Agios Kosmas Marina was created to host the Sailin... 37.8773 23.7268 Athens, Greece Greece
15 Glyfada C 37.8629 23.7435 Athens, Greece Greece
16 Astir Vouliagmeni Located in the Vouliagmeni promontory, 21 km south... 37.8032 23.7734 Athens, Greece Greece
17 Olympic 37.6967 24.0592 Athens, Greece Greece
18 Aretsou Thessaloniki Marina is located in the Northern Aeg... 40.5734 22.9466 Thessaloniki, Greece Greece

Βιβλιογραφία

Anwar, A., Mohamed, M., Tarasov, V., Littley, M., Rupprecht, L., & Cheng, Y. e. (2018). Improving Docker Registry Design based on Production Workload Analysis. σσ. 265-278.

Bui, T. (2015). Analysis of Docker Security.

Jarostaw, K. (2016). *Developing with Docker.* Packt Publishing Ltd.

Melendez, C. (2005). Getting Started with Kubernetes. σσ. 1-32.

Rad, B. B., Bhatti, H. J., & Ahmadi, M. (2017). An Introduction to Docker and Analysis of its Performance Electrification of Transportation System View project An Introduction to Docker and Analysis of its Performance. *IJCSNS International Journal of Computer Science and Network Security*.

Turnbull, J. (2014). *The Docker Book.*