Π.Μ.Σ. ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ & ΝΕΦΟΣ

ΤΕΧΝΟΛΟΓΙΕΣ ΥΠΟΛΟΓΙΣΤΙΚΗΣ ΝΕΦΟΥΣ (SD0105)

Εικονικοποίηση σε επίπεδο Λειτουργικού Συστήματος με το Docker

Αποστολάκης Ιωάννης, mai25004

Χρυσοχοΐδης Αναστάσιος, mai25067

Δεκέμβριος 2024

Πίνακας Περιεχομένων

1. Εισαγωγή

Τα τελευταία χρόνια, το cloud computing έχει σημειώσει ραγδαία ανάπτυξη, λόγω των αυξανόμενων αναγκών της βιομηχανίας για υπολογιστική ισχύ, αποθήκευση και ευέλικτη κλιμάκωση. Η Εικονικοποίηση (Virtualization) αποτελεί την τεχνολογία κλειδί στην εξέλιξη αυτή, επιτρέποντας την δημιουργία εικονικών πόρων, όπως Εικονικές Μηχανές (Virtual Machines - VMs) και Containers. Οι τεχνολογίες αυτές επιτρέπουν την καλύτερη αξιοποίηση των φυσικών πόρων, με την φιλοξενία πολλών εφαρμογών σε μια μόνο φυσική υποδομή, μειώνοντας έτσι το κόστος συντήρησης και κατ' επέκταση βελτιώνοντας την αποδοτικότητα.

Παρότι οι εικονικές μηχανές παραμένουν μια δημοφιλής λύση, ειδικά σε οργανισμούς που έχουν on premises υποδομές, παρουσιάζουν ορισμένα μειονεκτήματα σε σύγκριση με τα containers. Τα containers αποτελούν τεχνολογία εικονικοποίησης που επιτρέπει τη φιλοξενία και εκτέλεση εφαρμογών σε απομονωμένα περιβάλλοντα, διαχωρισμένα από το υπόλοιπο σύστημα, δίχως ωστόσο την ανάγκη ύπαρξης ενός λειτουργικού συστήματος όπως απαιτούν οι εικονικές μηχανές. Κάθε container περιέχει όλα τα αναγκαία εξαρτήματα για να τρέξει μια εφαρμογή και ΄έτσι διασφαλίζει την σταθερή λειτουργία των εφαρμογών ανεξάρτητα από το σε ποιο περιβάλλον θα εκτελείται. Ωστόσο, η διαχείριση πολλαπλών container αποτελεί μια σημαντική πρόκληση. Την πρόκληση αυτή την αντιμετωπίζει το Docker, μια πλατφόρμα που απλοποιεί τη δημιουργία, διανομή, ανάπτυξη και γενική διαχείριση των containers, προσφέροντας ένα εύχρηστο περιβάλλον για τους διαχειριστές συστημάτων.

Σε αυτήν την εργασία, θα αναλυθεί η τεχνολογία της εικονικοποίησης λειτουργικού συστήματος (Containers) με την open source πλατφόρμα Docker.

2. Τεχνολογίες Εικονικοποίησης

Εικονικοποίηση είναι η τεχνολογία που επιτρέπει το διαμοιρασμό των φυσικών πόρων ενός υπολογιστικού συστήματος σε πολλά εικονικά μηχανήματα. Αυτό προσφέρει πολλά οφέλη όπως:

- αποδοτική αξιοποίηση πόρων, καθώς αξιοποιούνται στο έπακρο οι πόροι ενός συστήματος ανάλογα με τις ανάγκες των εικονικών μηχανών αντί να υπάρχουν πολλά φυσικά συστήματα όπου καθένα συνήθως υπο-χρησιμοποιείται,
- ευκολότερη διαχείριση, αφού οι εικονικές μηχανές ελέγχονται αποκλειστικά μέσω λογισμικού και δεν απαιτείται αγορά φυσικών μηχανημάτων και εγκατάσταση τους
- ελαχιστοποίηση χρόνου εκτός λειτουργίας (downtime), εφόσον μπορεί να είναι ενεργοποιημένη εφεδρική εικονική μηχανή με ελάχιστο κόστος για να χρησιμοποιηθεί σε περίπτωση σφάλματος
- εύκολη κλιμάκωση, επειδή μπορούν να ενεργοποιούνται αυτόματα επιπλέον εικονικές μηχανές για να καλύψουν αυξημένη ζήτηση ή αντίστοιχα να μειώνεται ο αριθμός τους σε περιόδους χαμηλής ζήτησης για οικονομία.

Η εικονικοποίηση μπορεί να γίνει είτε σε επίπεδο υλικού (VM) είτε σε επίπεδο λειτουργικού συστήματος (Container).

2.1 Εικονικές Μηχανές

Στην πρώτη περίπτωση γίνεται πλήρης προσομοίωση του υλικού, έτσι ώστε η εικονική μηχανή να λειτουργεί χωρίς τροποποιήσεις, ανεξάρτητα από το σε ποιο σύστημα φιλοξενείται. Τα VMs που φιλοξενούνται σε έναν φυσικό εξοπλισμό διαμοιράζονται τους φυσικούς πόρους του εξοπλισμού (επεξεργαστική ισχύς, μνήμη, αποθηκευτικό χώρο κτλ). Κάθε VM τρέχει το δικό του λειτουργικό σύστημα και λειτουργεί ξεχωριστά από άλλα VMs παρόλο που μοιράζονται τους ίδιους φυσικούς πόρους. Οι εικονικές μηχανές αναπτύσσονται για να ικανοποιήσουν διαφορετικά επίπεδα αναγκών σε επεξεργαστική ισχύ, όπως εκτέλεση λογισμικού που απαιτεί συγκεκριμένο λειτουργικό σύστημα ή για την δοκιμή εφαρμογών σε ασφαλές περιβάλλον. Στην πλειονότητα των περιπτώσεων οι εικονικές μηχανές χρησιμοποιούνται για την εικονικοποίηση διακομιστών, βοηθώντας τους διαχειριστές συστημάτων να αξιοποιήσουν στο έπακρο τους φυσικούς τους πόρους. Επιπλέον, οι εικονικές μηχανές χρησιμοποιούνται για συγκεκριμένους σκοπούς που θεωρούνται επικίνδυνοι να εκτελούνται από φυσικά μηχανήματα όπως πρόσβαση σε μολυσμένα δεδομένα ή δοκιμή διάφορων λειτουργικών συστημάτων. Από τη στιγμή που η εικονική μηχανή είναι χωρισμένη από το υπόλοιπο σύστημα, το λογισμικό μέσα στην εικονική μηχανή δεν επηρεάζει τον φυσικό εξοπλισμό όπου φιλοξενείται. Η διαχείριση των εικονικών μηχανών γίνεται με την βοήθεια ενός λογισμικού που λέγεται Επόπτης (Hypervisor). Ο επόπτης αποτελεί ένα εύχρηστο εργαλείο διαχείρισης εικονικών μηχανών, και των πόρων ενός φυσικού μηχανήματος.

Ενώ η τεχνολογία των εικονικών μηχανών αποτελεί μια βελτιωμένη μέθοδο αξιοποίησης πόρων έναντι της χρήσης φυσικών μηχανημάτων, δεν έρχεται και δίχως τα μειονεκτήματά της. Η λειτουργία των VM απαιτεί ένα πλήρη πυρήνα λειτουργικού συστήματος, που συνεπάγεται με μεγαλύτερη κατανάλωση πόρων και βαρύτερες απαιτήσεις σε αποθήκευση και χρήση μνήμης. Κάτι τέτοιο, επηρεάζει την απόδοση και να αυξήσει τον χρόνο εκκίνησης. Τα μειονεκτήματα αυτά τα λύνουν οι containers. Στην εικονικοποίηση σε επίπεδο λειτουργικού συστήματος, τα containers χρησιμοποιούν όλα κοινό λειτουργικό σύστημα.

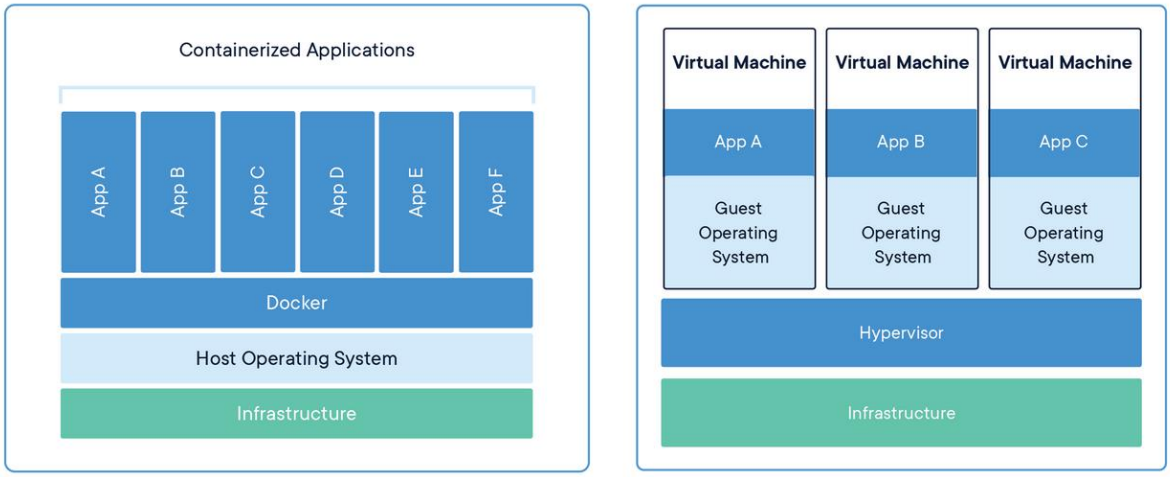
2.2 Containers

Η εικονικοποίηση σε επίπεδο λειτουργικού συστήματος έχει αρκετά πλεονεκτήματα σε σχέση με την εικονικοποίηση σε επίπεδο υλικού. Τα containers είναι μικρότερα σε μέγεθος καθώς δεν περιέχουν λειτουργικό σύστημα αλλά μόνο τις επιπλέον βιβλιοθήκες και υπηρεσίες που απαιτούν για την λειτουργία τους. Επίσης, χρειάζονται λιγότερους πόρους καθώς χρησιμοποιούν το ήδη υπάρχον λειτουργικό σύστημα αντί να έχουν ξεχωριστό. Ακόμα, είναι γρηγορότερα στην δημιουργία και τον τερματισμό καθώς δεν απαιτούν την εκκίνηση λειτουργικού συστήματος από την αρχή.

Τα containers προσφέρουν ευελιξία, καθώς μπορούν να εικονικοποιήσουν τόσο ελαφριές όσο και απαιτητικές εφαρμογές, έχουν εύκολη δυνατότητα κλιμάκωσης καθώς ο αριθμός των κλώνων μπορεί να μεταβληθεί εύκολα με αυτοματοποιημένες

διαδικασίες και προσφέρουν εύκολη φορητότητα καθώς είναι εύκολο να αναπτυχθούν σε οποιοδήποτε περιβάλλον.

Στην Εικόνα 1 αποτυπώνονται οι αρχιτεκτονικές των τεχνολογιών εικονικοποίησης.



Εικόνα 1 – Αρχιτεκτονική VMs & Containers

Υπάρχουν όμως και μειονεκτήματα στην εικονικοποίηση σε επίπεδο λειτουργικού συστήματος. Αρχικά, όλα τα containers που φιλοξενούνται σε ένα σύστημα πρέπει να χρησιμοποιούν το ίδιο λειτουργικό σύστημα. Επίσης, το γεγονός ότι είναι λιγότερο απομονωμένα αυξάνει την πιθανότητα ύπαρξης ευπαθειών.

Η εικονικοποίηση σε επίπεδο λειτουργικού συστήματος βασίζεται στην δημιουργία και εκτέλεση Containers. Τα containers δημιουργούνται ξεκινώντας από μια απεικόνιση (container image), δηλαδή ένα αμετάβλητο σύνολο αρχείων. Κάθε απεικόνιση προσδιορίζει μια άλλη απεικόνιση σαν βάση καθώς και πολλαπλές στρώσεις (layers). Κάθε στρώση αποτελείται από ένα σύνολο αρχείων και ουσιαστικά ορίζει τις διαφορές της προηγούμενης στρώσης από την τρέχουσα, όπως προσθήκη κάποιας βιβλιοθήκης ή τροποποίηση κάποιας ρύθμισης. Εάν στοιβαχτούν όλες οι στρώσεις πάνω στην απεικόνιση-βάση τότε δημιουργείται η ζητούμενη απεικόνιση.

Η στοίβαξη αυτή υλοποιείται με την λειτουργικότητα της προσάρτησης ένωσης (union mount), όπου γίνεται να προσαρτηθούν πολλαπλοί κατάλογοι στο ίδιο μονοπάτι και ο κατάλογος που θα προκύψει να φαίνεται σαν να αποτελείται από τα συνδυασμένα περιεχόμενά τους. Κάθε στρώση βρίσκεται σε ξεχωριστό κατάλογο και τα αρχεία της έχουν άδεια μόνο για ανάγνωση ώστε να είναι αμετάβλητη. Για τη δημιουργία της τελικής απεικόνισης γίνεται προσάρτηση ένωσης με όλους τους καταλόγους των στρώσεων.

Για κάθε καινούριο container δημιουργείται και προσαρτάται ένας επιπλέον κατάλογος (μία επιπλέον στρώση) με άδεια ανάγνωσης και εγγραφής. Ο κατάλογος αυτός χρησιμεύει για να αποθηκεύονται οι αλλαγές στα αρχεία που γίνονται όσο

εκτελείται το συγκεκριμένο container. Με αυτόν τον τρόπο, τα αρχεία της απεικόνισης μένουν αμετάβλητα και μπορούν να χρησιμοποιηθούν για τη δημιουργία κι άλλων containers.

Εκτός από το σύστημα αρχείων που περιγράφηκε, κάθε container περιέχει επίσης και ένα σύνολο διεργασιών (processes). Οι διεργασίες αυτές λειτουργούν απομονωμένα από άλλα containers και από τις διεργασίες του συστήματος που φιλοξενεί τα containers. Η απομόνωση επιτυγχάνεται με την λειτουργικότητα των χώρων ονομάτων (namespaces), που προσφέρει πολλαπλά επίπεδα απομόνωσης. Κάθε διεργασία έχει περιορισμένες δυνατότητες πρόσβασης ανάλογα με τον χώρο ονομάτων που ανήκει.

Τέλος, δεδομένου ότι όλα τα containers μοιράζονται κοινούς πόρους (CPU, μνήμη κλπ), απαιτείται ένα σύστημα ελέγχου των πόρων που χρησιμοποιούνται από κάθε container. Αυτό γίνεται με τις ομάδες ελέγχου (cgroups), μια λειτουργία του πυρήνα που φροντίζει να γίνεται σωστός διαμοιρασμός των πόρων μεταξύ διεργασιών και επιτρέπει τον προσδιορισμό κανόνων.

Το πιο διάσημο και ευρέως χρησιμοποιούμενο σύστημα σήμερα για την εικονικοποίηση σε επίπεδο λειτουργικού συστήματος είναι το Docker, αν και υπάρχουν εναλλακτικές όπως το Podman.

3. Docker

Το Docker περιλαμβάνει ένα σύνολο από προϊόντα τα οποία συνεργάζονται για να επιτευχθεί η εικονικοποίηση σε επίπεδο λειτουργικού συστήματος. Ξεκίνησε σαν εσωτερικό project της εταιρείας dotCloud Inc, η οποία το δημοσίευσε σαν λογισμικό ανοιχτού κώδικα το 2013 και μετονομάστηκε το ίδιο έτος σε Docker Inc. Τα επόμενα χρόνια υπήρξε ραγδαία ανάπτυξή του και δημιουργήθηκαν συνεργασίες με πολλές μεγάλες εταιρείες όπως Microsoft, Amazon, IBM.

3.1 Docker Registry

Ένα μητρώο απεικονίσεων Docker (Docker Registry) είναι ένα σύστημα για την αποθήκευση και διαμοιρασμό απεικονίσεων (images), από τις οποίες δημιουργούνται τα containers. Κάθε μητρώο περιέχει συλλογές από απεικονίσεις, τα αποθετήρια (repositories), και κάθε αποθετήριο περιέχει μία η περισσότερες απεικονίσεις. Οι απεικονίσεις σε ένα αποθετήριο συνήθως σχετίζονται μεταξύ τους, για παράδειγμα μπορεί να είναι διαφορετικές εκδόσεις ή διαφορετικά συστατικά ενός προϊόντος. Κάθε απεικόνιση χαρακτηρίζεται από το όνομά της και, προαιρετικά, κάποιες ετικέτες. Οι ετικέτες έχουν σκοπό συνήθως να προσδιορίσουν έναν συγκεκριμένο αριθμό έκδοσης της απεικόνισης ή μια παραλλαγή της (variant).

Το πιο γνωστό μητρώο απεικονίσεων είναι το Docker Hub. Είναι το μεγαλύτερο δημόσιο μητρώο αυτή την στιγμή, χρησιμοποιείται από το Docker σαν προεπιλεγμένο και προσφέρει ελεύθερα την δυνατότητα σε οποιονδήποτε να δημοσιεύει απεικονίσεις. Επίσης δίνει την δυνατότητα να αποθηκεύει ιδιωτικά απεικονίσεις επί πληρωμή. Επιπλέον προσφέρει διάφορους αυτοματισμούς για την δημιουργία και αποθήκευση

απεικονίσεων καθώς και ένα εργαλείο γραμμής εντολών (CLI) για την διαχείρισή του (το οποίο όμως βρίσκεται ακόμα σε πειραματικό στάδιο).

Εκτός από το Docker Hub υπάρχουν και άλλα δημόσια μητρώα όπως το Elastic Container Registry της Amazon, το Azure Container Registry και το Google Container Registry. Επίσης παρέχεται η δυνατότητα να εγκατασταθεί ένα μητρώο σε ιδιωτικό εξυπηρετητή.

3.2 Docker Engine

Το Docker Engine (ή αλλιώς Docker Daemon) είναι μια υπηρεσία που τρέχει στο παρασκήνιο και είναι υπεύθυνη για το κυριότερο έργο του Docker. Επικοινωνεί με μητρώα απεικονίσεων, δημιουργεί και αποθηκεύει τοπικά απεικονίσεις, στέλνει απεικονίσεις σε μητρώα και διαχειρίζεται τα containers κατά τη διάρκεια της ζωής τους. Αξίζει να σημειωθεί ότι για την εκκίνηση και τερματισμό των containers χρησιμοποιεί το containerd, εργαλείο το οποίο ξεκίνησε σαν κομμάτι του Docker Engine και το 2017 αποσπάστηκε και δόθηκε στο Cloud Native Computing Foundation.

Το Docker Engine επίσης παρέχει την δυνατότητα διατήρησης δεδομένων των containers. Όπως αναφέρθηκε, τα αρχεία που τροποποιούνται ή παράγονται κατά την λειτουργία ενός container αποθηκεύονται σε έναν ξεχωριστό κατάλογο (μια εγγράψιμη στρώση), η οποία διαγράφεται όταν διαγραφεί το container. Η διαχείριση του αποθηκευτικού χώρου γίνεται μέσω των storage drivers, που χειρίζονται τον τρόπο που οι απεικονίσεις και τα containers χρησιμοποιούν τον διαθέσιμο δίσκο. Για την διατήρηση κάποιων αρχείων παρέχεται η λειτουργία των τόμων (volumes). Ένας τόμος είναι ουσιαστικά ένας κατάλογος τον οποίο διαχειρίζεται το Docker. Αποθηκεύεται σε ξεχωριστό σημείο και μπορεί να προσαρτηθεί σε κάποιο μονοπάτι του container. Εκτός από τους τόμους, μπορεί επίσης να προσαρτηθεί στο container και οποιοσδήποτε άλλος κατάλογος ή αρχείο του συστήματος που φιλοξενεί τα containers (bind mount).

Το Docker Engine επιτρέπει ακόμα στα containers να χρησιμοποιούν το δίκτυο για να επικοινωνούν μεταξύ τους ή και με το ίντερνετ. Κάθε container έχει την δική του διεπαφή δικτύου με διεύθυνση IP, θύρες, πύλη δικτύου (gateway) και τα υπόλοιπα απαραίτητα στοιχεία για την επικοινωνία, σαν να ήταν ξεχωριστή μηχανή. Μπορεί επίσης να αντιστοιχηθεί κάποια θύρα του container σε θύρα του συστήματος που φιλοξενεί το container, έτσι ώστε κάθε αίτημα που φτάνει στην θύρα αυτή να προωθείται στην θύρα του container και να μπορεί για παράδειγμα το container να λειτουργεί σαν εξυπηρετητής.

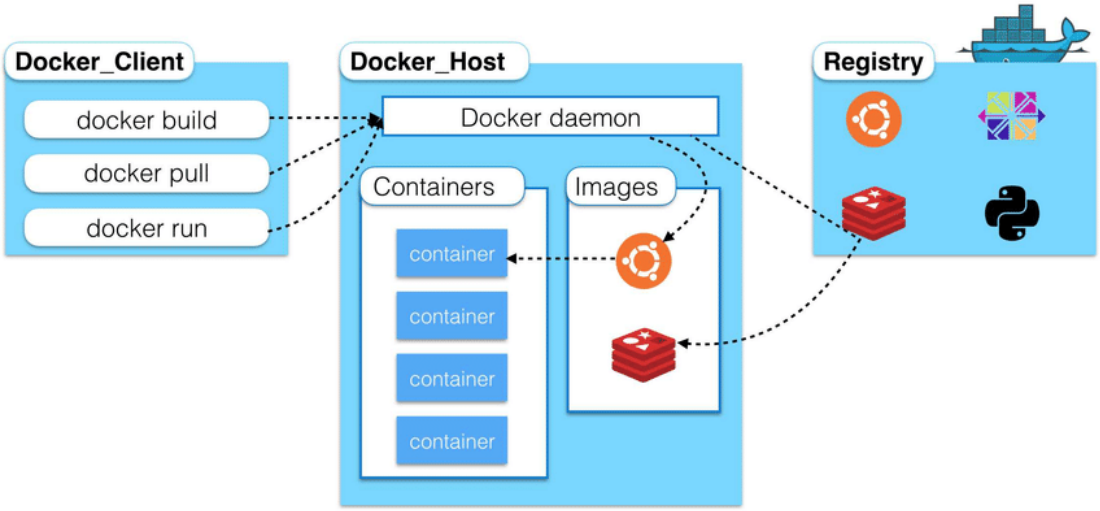
3.3 Docker Client

Το Docker Client είναι εργαλείο που χρησιμοποιείται για να επικοινωνήσει ο χρήστης με το Docker Engine. Ο Docker Client μπορεί να έχει την μορφή διεπαφής γραμμής εντολών (CLI) ή γραφικής διεπαφής. Είναι υπεύθυνος για την μετάφραση των εντολών του χρήστη στα κατάλληλα αιτήματα HTTP τα οποία στέλνει στον εξυπηρετητή που διαθέτει το Docker Engine ώστε να εκτελεστούν οι ζητούμενες λειτουργίες. Οι εντολές του Docker Client, όπως docker run, docker build, docker stop,

στέλνουν αιτήματα μέσω του Docker API στο Docker Engine, το οποίο εκτελεί τις απαραίτητες λειτουργίες.

Ένας Docker Client που αξίζει να αναφερθεί είναι το Docker Desktop, το οποίο, εκτός από γραφική διεπαφή για τον χειρισμό του Docker Engine, παρέχει και άλλες δυνατότητες όπως την δημιουργία τοπικής συστάδας Kubernetes. Το Docker Desktop παρέχεται από την Docker Inc δωρεάν σε όλους εκτός από μεγάλες επιχειρήσεις.

Τα κύρια στοιχεία της αρχιτεκτονικής του Docker που περιγράφηκαν παραπάνω, φαίνονται στην Εικόνα 2.



Εικόνα 2 – Αρχιτεκτονική Docker

4. Χρήση του Docker

Το Docker Engine μπορεί να εγκατασταθεί σε πληθώρα λειτουργικών συστημάτων και διανομών Linux. Στην επίσημη σελίδα του Docker βρίσκονται όλα τα διαθέσιμα πακέτα εγκατάστασης. Όπως αναφέρθηκε, υπάρχει πλήθος έτοιμων απεικονίσεων σε μητρώα όπως το Docker Hub που μπορούν να εκτελεστούν χωρίς να απαιτούν επιπλέον τροποποιήσεις. Πράγματι, ο χρήστης χρειάζεται μόνο να δώσει μια εντολή όπως η παρακάτω:

docker run --name myDB mongo

Αυτή η εντολή αρκεί ώστε το Docker Engine να αναζητήσει στο μητρώο την πιο πρόσφατη έκδοση της απεικόνισης για τη βάση δεδομένων MongoDB, να την κατεβάσει τοπικά, να δημιουργήσει ένα καινούριο container από αυτή την απεικόνιση με όνομα “myDB” και να το θέσει σε λειτουργία.

Το Docker παρέχει την δυνατότητα στους χρήστες να δημιουργήσουν προσαρμοσμένες απεικονίσεις, έτσι ώστε να μπορούν να εκτελέσουν την εφαρμογή που επιθυμούν. Η δημιουργία αυτή γίνεται με την προσθήκη στρώσεων πάνω σε ήδη υπάρχουσες απεικονίσεις. Οι στρώσεις αυτές ορίζονται σαν μια ακολουθία από εντολές σε ένα αρχείο κειμένου το οποίο ονομάζεται Dockerfile.

Κάθε εντολή ξεκινάει με μια δεσμευμένη λέξη και έπειτα παρατίθενται ορίσματα που προσδιορίζουν παραμέτρους. Οι εντολές που κάνουν οποιαδήποτε τροποποίηση στο σύστημα αρχείων της απεικόνισης δημιουργούν νέα στρώση, το οποίο μεταφράζεται σε νέο κατάλογο που θα προσαρτηθεί μαζί με τους υπόλοιπους όπως εξηγήθηκε προηγουμένως. Οι σημαντικότερες εντολές στο Dockerfile παρουσιάζονται παρακάτω:

- FROM: είναι συνήθως η πρώτη εντολή που εμφανίζεται στο Dockerfile και δηλώνει την απεικόνιση που θα χρησιμοποιηθεί σαν βάση για την προσαρμοσμένη απεικόνιση. Μπορεί επίσης αντί για απεικόνιση να δηλωθεί η δεσμευμένη λέξη “scratch” σαν όρισμα, που θα έχει σαν αποτέλεσμα να χρησιμοποιηθεί μια κενή απεικόνιση σαν βάση.

- COPY: χρησιμοποιείται για να αντιγράψει ένα ή περισσότερα αρχεία και καταλόγους από το σύστημα που φιλοξενεί το Docker στο σύστημα αρχείων της απεικόνισης. Με αυτόν τον τρόπο συνήθως προστίθενται τα αρχεία της εφαρμογής που θα εκτελεστεί στο container.

- ENV: ορίζει κάποιες μεταβλητές περιβάλλοντος στην απεικόνιση, οι οποίες θα είναι ορατές εντός του container που θα δημιουργηθεί.

- VOLUME: προσδιορίζει μονοπάτια εντός της απεικόνισης στα οποία θα προσαρτηθούν τόμοι κατά την εκτέλεση του container, ώστε να διατηρηθούν τα αρχεία που θα δημιουργηθούν σε αυτά τα μονοπάτια και μετά τον τερματισμό και την διαγραφή του container.

- RUN: εκτελεί εντολές γραμμής εντολών μέσα στην απεικόνιση. Χρησιμοποιείται συχνά για την εγκατάσταση βιβλιοθηκών ή απαιτούμενων εκτελέσιμων που δεν υπήρχαν ήδη στην απεικόνιση, για παράδειγμα χρησιμοποιώντας τον διαχειριστή πακέτων “apt” σε απεικονίσεις Ubuntu.

- ENTRYPOINT: προσδιορίζει μια εντολή τερματικού που θα χρησιμοποιηθεί για την εκκίνηση του container. Συνήθως είναι κάποιο script ή κάποιο εκτελέσιμο αρχείο που έχει μεταφερθεί στην απεικόνιση με την εντολή COPY.

- ARG: επιτρέπει την δήλωση ορισμάτων που θα δοθούν από τον χρήστη κατά την διάρκεια δημιουργίας (ή χτισίματος) της απεικόνισης σύμφωνα με το Dockerfile.

- USER: δηλώνει τον χρήστη εντός του container που θα χρησιμοποιηθεί για να ξεκινήσει η εκτέλεση από το σημείο εισόδου του container.

- WORKDIR: ορίζει το μονοπάτι εντός της απεικόνισης που θα χρησιμοποιείται σαν κατάλογος εργασίας (current working directory) για τις επόμενες εντολές του Dockerfile, μέχρι να προσδιοριστεί κάποιο άλλο μονοπάτι με τη χρήση πάλι της ίδιας εντολής.

Για την δημιουργία της προσαρμοσμένης απεικόνισης αρκεί ο χρήστης να εκτελέσει μια εντολή όπως η παρακάτω:

docker build -t myimage:latest

Η εντολή αυτή θα δημιουργήσει την απεικόνιση από το Dockerfile που βρίσκεται στον τρέχοντα κατάλογο εργασίας, θα την ονομάσει "myimage" και θα της δώσει επιπλέον την ετικέτα "latest", που τυπικά αναφέρεται στην πιο πρόσφατη έκδοση της συγκεκριμένης απεικόνισης. Έπειτα, ο χρήστης μπορεί με την εντολή "*docker run*" όπως περιγράφηκε προηγουμένως να δημιουργήσει και να εκτελέσει container από αυτή την απεικόνιση.

Παρέχεται ακόμα πληθώρα εντολών για την διαχείριση των containers και τις διάφορες λειτουργίες του Docker. Ενδεικτικά αναφέρονται κάποιες παρακάτω:

- *docker pause/unpause* : προσωρινή παύση/συνέχιση αντίστοιχα των διεργασιών ενός container
- *docker stop/start* : τερματισμός/εκκίνηση των διεργασιών ενός container
- *docker rm* : διαγραφή ενός container
- *docker logs* : εμφάνιση των αρχείων καταγραφής συμβάντων ενός container
- *docker exec* : εκτέλεση μιας εντολής τερματικού εντός του container
- *docker pull/push* : κατέβασμα/ανέβασμα μιας απεικόνισης από/σε ένα μητρώο αντίστοιχα

5. Docker Compose

Το Docker Compose αποτελεί εργαλείο για την ευκολότερη δημιουργία και διαχείριση εφαρμογών που αποτελούνται από πολλά containers. Η βασική του ιδέα είναι η αποθήκευση σε ένα αρχείο YAML όλης της πληροφορίας που αφορά σε παραμετροποίηση containers ώστε να είναι εύκολο για τον χρήστη να δημιουργήσει τα containers μόνο με την εκτέλεση αυτού του συγκεκριμένου αρχείου. Η πρώτη έκδοση του Docker Compose δημοσιεύτηκε το 2014.

Το βασικότερο πλεονέκτημα του Docker Compose όπως αναφέρθηκε είναι η δυνατότητά του να συγκεντρώσει σε ένα αρχείο την περιγραφή πολλαπλών containers που αποτελούν μια εφαρμογή. Έπειτα ο χρήστης μπορεί με μία απλή εντολή του Docker Compose να ξεκινάει ή να τερματίζει την εφαρμογή, αντί να απαιτούνται κάθε φορά πολλές εντολές ενός Docker Client, που καταλήγουν να γίνονται αρκετά πολύπλοκες και μακροσκελείς όταν απαιτείται πιο εξειδικευμένη παραμετροποίηση. Ακόμα, το Docker Compose εφαρμόζει κάποιες βελτιστοποιήσεις για οικονομικότερη και γρηγορότερη διαχείριση, όπως το να επαναχρησιμοποιεί τυχόν ήδη υπάρχοντα containers κατά την επανεκκίνηση μιας εφαρμογής αν δεν έχουν γίνει αλλαγές σε αυτά. Επιπλέον, το Docker Compose διευκολύνει την συνεργασία στην ανάπτυξη εφαρμογών, καθώς είναι πολύ απλό να διαμοιραστεί το αρχείο YAML μεταξύ προγραμματιστών.

Στα πλαίσια της παραμετροποίησης containers, το Docker Compose παρέχει όλες τις δυνατότητες που έχει και ένας Docker Client. Μπορεί να ορίσει containers από κάποιο image, να τους δηλώσει μεταβλητές περιβάλλοντος, να προωθήσει θύρες δικτύου τους σε θύρες του συστήματος που τα φιλοξενεί, να ορίσει τόμους που θα προσαρτηθούν σε container, να δημιουργήσει εσωτερικά δίκτυα και να τοποθετήσει τα containers σε αυτά ώστε να επικοινωνούν μεταξύ τους και άλλα.

Το Docker Compose προσφέρει όμως και επιπλέον δυνατότητες που δεν υλοποιούνται σε έναν απλό Docker Client, όπως τα “lifecycle hooks”. Αυτά δίνουν τη δυνατότητα στον χρήστη να ορίσει εντολές τερματικού που θα εκτελεστούν εντός του container είτε αμέσως μετά που θα ξεκινήσει είτε αμέσως πριν τον τερματισμό του. Ένα παράδειγμα χρήσης θα ήταν η πραγματοποίηση εγγραφής δεδομένων στο δίσκο ώστε να μη χαθούν κατά τον τερματισμό του container.

Μια άλλη δυνατότητα που προσφέρει το Docker Compose είναι ο προσδιορισμός εξαρτήσεων μεταξύ containers με την χρήση της λέξης-κλειδί “depends_on”. Μια τέτοια εξάρτηση σημαίνει ότι ένα container για να λειτουργήσει σωστά απαιτεί να λειτουργεί ήδη ένα άλλο container, όπως για παράδειγμα ένας εξυπηρετητής με μια βάση δεδομένων. Το Docker Compose θα φροντίσει να ξεκινήσει τα containers με την κατάλληλη σειρά ώστε κάθε container να ξεκινήσει μετά από αυτά από τα οποία εξαρτάται. Δίνεται μάλιστα η δυνατότητα να οριστούν και προσαρμοσμένοι έλεγχοι για την υγεία του container για να εξασφαλιστεί ότι θα είναι έτοιμα για λειτουργία.

Επίσης, το Docker Compose επιτρέπει τον ορισμό διάφορων προφίλ και την αντιστοίχιση κάποιων containers με αυτά. Με αυτόν τον τρόπο τα containers αυτά θα εκτελεστούν μόνο αν είναι ενεργό το προφίλ που τους έχει οριστεί. Αυτό μπορεί να φανεί χρήσιμο όταν η εφαρμογή πρέπει να μπορεί να εκτελείται σε πολλαπλά περιβάλλοντα, όπου να απαιτούνται διαφορετικά containers σε κάθε περιβάλλον.

Ακόμα, το Docker Compose παρέχει τη δυνατότητα ορισμού κάποιων μυστικών (secrets) ώστε να υπάρχει σχετική προστασία σε ευαίσθητα δεδομένα όπως κωδικοί. Το περιεχόμενο των μυστικών ορίζεται σε διαφορετικό αρχείο ή μεταβλητή περιβάλλοντος, και το Docker Compose αναλαμβάνει να μεταφέρει την τιμή στο container, έτσι ώστε να αποφευχθεί να γραφτεί το μυστικό απευθείας στον κώδικα ή στο αρχείο YAML.

Δίνεται επιπλέον η λειτουργικότητα του Compose Watch, η οποία έχει στόχο να διευκολύνει τον προγραμματιστή κατά τη διάρκεια της ανάπτυξης μιας εφαρμογής. Το Compose Watch είναι ένας μηχανισμός παρακολούθησης αρχείων, ο οποίος αναλαμβάνει να συγχρονίζει αυτόματα τα αρχεία εντός του container με τα αρχεία κώδικα που γράφει ο προγραμματιστής. Ο συγχρονισμός αυτός συμβαίνει κάθε φορά που ο προγραμματιστής αποθηκεύει τον κώδικα και μπορεί να είναι από απλή αντικατάσταση του αρχείου στο container μέχρι και επαναδημιουργία της απεικόνισης και δημιουργία νέου container από την νέα απεικόνιση.

Στην Εικόνα 3 φαίνεται ένα παράδειγμα Docker Compose yaml αρχείου, όπου στήνει μια ιστοσελίδα Wordpress χρησιμοποιώντας MySql για βάση δεδομένων με μια υποδομή που χρησιμοποιεί πολλαπλά containers. Ορίζει τις υπηρεσίες που θα τρέχουν

(mysql και wordpress), ορίζει ποιες απεικονίσεις θα χρησιμοποιηθούν, ορίζει τις εξαρτήσεις, καθορίζει μεταβλητές που θα χρησιμοποιήσει η κάθε υπηρεσία καθώς και τις πόρτες στις οποίες θα υπάρχει επικοινωνία.

```yaml
version: '3.3'

services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:C:\Users\User\Desktop\dcompose
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: rootwordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
volumes:
    db_data:
```

Εικόνα 3 Παράδειγμα YAML αρχείου

6. Ενορχήστρωση Docker Containers

Ο μεγάλος αριθμός των container που απαιτούνται για την λειτουργία πολλών συστημάτων κάνει την διαχείρισή τους πιο περίπλοκη και κατ επέκταση απαιτητική. Η ανάγκη για έλεγχο, συντονισμό , σχεδιασμό των containers οδήγησε στην δημιουργία των πρώτων εργαλείων ενορχήστρωσης. Με αυτά τα εργαλεία, είναι επίσης δυνατή η παρακολούθηση εφαρμογών που είναι εντός των containers, η εκτέλεσή τους, η ενημέρωσή τους, αποσφαλμάτωση και η επιδιόρθωση τυχών λαθών. Υπάρχουν διάφορα γνωστά εργαλεία ενορχήστρωσης όπως το Docker Swarm και το Kubernetes. Σημειώνεται ότι ενώ όλα τα εργαλεία ενορχήστρωσης κάνουν κατά βάση την ίδια δουλειά, έχουν διάφορες μεταξύ τους, το καθένα με τα πλεονεκτήματα και μειονεκτήματά του.

6.1 Swarm mode

Το Docker παρέχει την δυνατότητα για τη δημιουργία ενός «σμήνους», δηλαδή τη συνεργασία πολλών Docker Engines για την εκτέλεση διεργασιών. Η αρχική υλοποίηση έγινε αρχικά σαν ξεχωριστή επέκταση με όνομα Docker Swarm. Έπειτα, το 2016, δημοσιεύτηκε ένα νέο εργαλείο, το Swarmkit, το οποίο είναι ενσωματωμένο στο Docker Engine και αντικατέστησε το Docker Swarm που πλέον δεν υποστηρίζεται.

Ένα σμήνος αποτελείται από πολλαπλά συνδεδεμένα Docker Engines, όπου κάθε Engine συνήθως βρίσκεται σε ξεχωριστό υπολογιστικό σύστημα, και έχει σαν στόχο την εύκολη εκμετάλλευση των συνολικών πόρων από τα διάφορα συστήματα για την εκτέλεση διεργασιών. Κάθε συνδεδεμένο Engine αποτελεί έναν κόμβο (node) και έχει ρόλο είτε εργάτη (worker), είτε διευθυντή (manager), είτε και τα δύο. Ο ρόλος του εργάτη είναι να εκτελεί διεργασίες ενώ ο ρόλος του διευθυντή είναι να εκτελεί διαχειριστικές λειτουργίες για τη σωστή λειτουργικότητα του σμήνους.

Όπως αναφέρθηκε, ένα σμήνος μπορεί να αναλάβει την εκτέλεση διεργασιών. Ο χρήστης δηλώνει ποια υπηρεσία επιθυμεί να εκτελέσει και πόσα αντίγραφά της (replicas) επιθυμεί να εκτελούνται ταυτόχρονα. Έπειτα, το σμήνος αναλαμβάνει να μοιράσει τις διεργασίες, δηλαδή τα ζητούμενα containers, στους κόμβους έτσι ώστε να επιτευχθεί η επιθυμητή κατάσταση αξιοποιώντας όσο το δυνατόν καλύτερα όλους τους διαθέσιμους κόμβους.

Σε ένα σμήνος συγκρίνεται συνεχώς η πραγματική κατάσταση του σμήνους, δηλαδή οι διεργασίες που εκτελούνται στη πραγματικότητα στους κόμβους, με την επιθυμητή κατάσταση, δηλαδή τις ζητούμενες διεργασίες από τον χρήστη. Εάν υπάρξει κάποια διαφοροποίηση μεταξύ τους, γίνονται οι απαραίτητες ενέργειες για την επαναφορά του σμήνους στην επιθυμητή κατάσταση. Διαφοροποίηση μπορεί να υπάρξει είτε λόγω αλλαγής της επιθυμητής κατάστασης (αλλαγή απαιτήσεων του χρήστη) είτε λόγω αλλαγής της πραγματικής κατάστασης (για παράδειγμα απώλεια ενός κόμβου και των διεργασιών του λόγω σφάλματος).

Το σμήνος επίσης διαθέτει ένα δίκτυο δρομολόγησης αιτημάτων για τη διαχείριση των αιτημάτων που έχουν σαν προορισμό υπηρεσίες που εκτελούνται στο σμήνος. Η υπηρεσία που εκτελείται στο σμήνος και λειτουργεί ως εξυπηρετητής μπορεί να ανατεθεί σε οποιονδήποτε κόμβο του σμήνους. Για να μην χρειάζεται ένας τελικός χρήστης της υπηρεσίας να πρέπει να ξέρει ανά πάσα στιγμή σε ποιόν κόμβο βρίσκεται η υπηρεσία προκειμένου να της στείλει αίτημα, χρησιμοποιείται το δίκτυο δρομολόγησης που αναφέρθηκε. Η υπηρεσία δηλώνει σε ποια θύρα δικτύου επιθυμεί να λαμβάνει αιτήματα και το σμήνος δεσμεύει την θύρα αυτή σε όλους τους κόμβους του και αναλαμβάνει να προωθεί στην υπηρεσία όλα τα αιτήματα που φτάνουν σε αυτή τη θύρα σε οποιονδήποτε κόμβο. Επομένως, ο τελικός χρήστης αρκεί να στείλει αίτημα σε οποιονδήποτε κόμβο του σμήνους και αυτό θα προωθηθεί αυτόματα στην υπηρεσία. Αν υπάρχουν πολλαπλά αντίγραφα της υπηρεσίας, το σμήνος θα μοιράσει τα αιτήματα στα αντίγραφα με κάποιον εξισορροπητή φορτίου (load balancer).

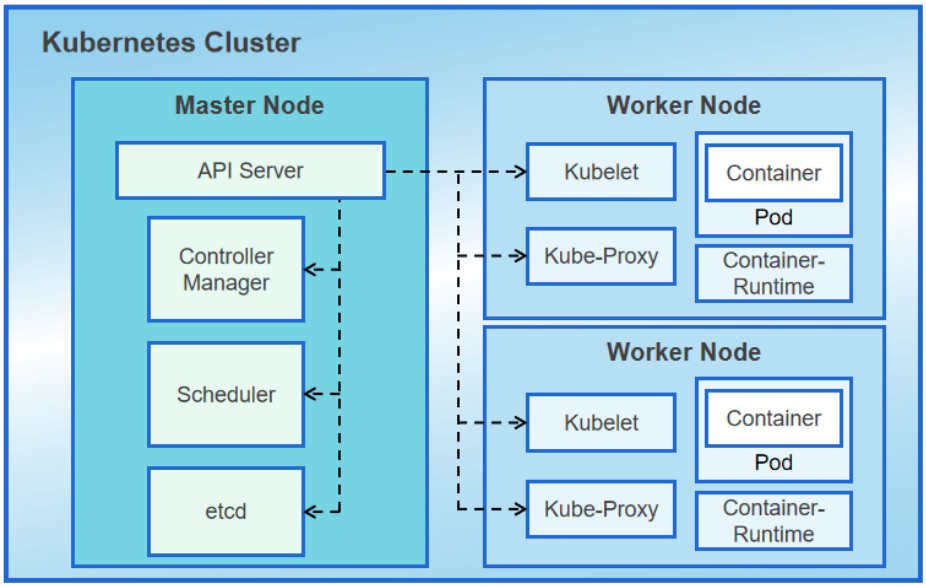
6.2 Kubernetes

Το Kubernetes αποτελεί μια δημοφιλής μηχανή ενορχήστρωσης container, γνωστή και ως k8s. Η μηχανή ενορχήστρωσης αρχικά είχε αναπτυχθεί από την google, αλλά μετέπειτα ανατέθηκε στην CloudF Native Computing Foundation, την οποία διαχειρίζεται η Linux Foundation. Οι πρώτες εκδόσεις της μηχανής βγήκαν το 2014, και σήμερα αποτελεί εργαλείο ανοικτού λογισμικού. Όπως και το Docker Swarm, έτσι και το Kubernetes για την ανάπτυξη των containers βασίζεται στην χρήση αρχείων yaml.

Το Kubernetes αποτελείται από πολλά συστατικά που επικοινωνούν μεταξύ τους μέσω API, όπου το καθένα έχει τον ρόλο του. Διαφέρει σε σχέση με άλλες

μηχανές ενορχήστρωσης κυρίως στο πως ομαδοποιούνται τα container για την παροχή μιας υπηρεσίας, τα λεγόμενα pods.

Δημιουργούνται συστάδες (clusters) που τις διαχειρίζεται το Kubernetes και αποτελούνται από τα συστατικά στοιχεία όπως φαίνεται στην Εικόνα 4.



Εικόνα 4 - Αρχιτεκτονική Kubernetes cluster

Ο Master Node αποτελεί στην ουσία τον πυρήνα του Kubernetes, που ελέγχει την διαδικασία της ενορχήστρωσης. Περιλαμβάνει και αυτό με την σειρά του βασικά στοιχεία όπως ο API Server, κεντρικό συστατικό όπου λειτουργεί ως API και είναι υπεύθυνο για την επικοινωνία με τα άλλα συστατικά του cluster και το etcd, που έχει όλα τα δεδομένα παραμετροποίησης. Ο Scheduler ελέγχει την διαθεσιμότητα, τις επιδόσεις και την χωρητικότητα του cluster αποφασίζοντας σε ποιους κόμβους θα τοποθετηθούν τα pods. Ο Controller Manager, στοιχείο που ελέγχει το etcd μέσω του API Server και διαρκώς συγκρίνει την τρέχουσα κατάσταση του cluster με την επιθυμητή. Περιλαμβάνει και αυτός με την σειρά του διάφορους controllers για τον έλεγχο των επιμέρους στοιχείων του cluster.

Οι Worker nodes σε ένα cluster λαμβάνουν αιτήματα από τον Master και αποτελούν τη βασική υπολογιστική ισχύ του cluster, όπου εκτελούνται τα containers. Συνήθως πρόκειται για ξεχωριστούς φυσικούς servers και περιλαμβάνουν τα εξής βασικά στοιχεία: το Kubelet, έναν agent που εκτελείται σε κάθε κόμβο, το Kubeproxy, έναν απλό διακομιστή proxy για εξισορρόπηση φορτίου, και τα Labels, τα οποία αποδίδονται ως ζεύγη κλειδιού-τιμής σε υπηρεσίες. Τα pods αποτελούν το χαμηλότερο επίπεδο του cluster και περιέχουν τα containers. Συνήθως ένα pod περιλαμβάνει ένα μόνο container, αλλά μπορεί να περιέχει και περισσότερα, τα οποία λειτουργούν ως ομάδα. Στην περίπτωση αυτή, ο προγραμματισμός των δραστηριοτήτων επικεντρώνεται στην ομάδα συνολικά. Κατά τη δημιουργία ενός container cluster, το pod προκαθορίζει πόσους πόρους, CPU και μνήμη, θα χρησιμοποιήσει. Ο scheduler χρησιμοποιεί αυτά τα δεδομένα για να αποφασίσει σε ποιον κόμβο θα τοποθετήσει το

pod. Αν τα containers αναπτύσσονταν ξεχωριστά και όχι ως ομάδα, θα ήταν δύσκολο να καθοριστεί η απαραίτητη χρήση πόρων, με αποτέλεσμα ο scheduler να μην μπορεί να τοποθετήσει σωστά ένα pod σε έναν συγκεκριμένο κόμβο.

Όπως και το Docker Swarm, το Kubernetes επιτρέπει την παρουσία πολλαπλών Masters σε ένα cluster, σε ενδεχόμενο πιθανής αστοχίας σε κάποιον από αυτούς, αλλά μόνο ένας μπορεί να είναι ενεργός κάθε φορά. Σε περίπτωση αποτυχίας του ενεργού Master, ενεργοποιείται ένας εφεδρικός Master.

7. Πλεονεκτήματα Docker

Τα containers υπερτερούν της παραδοσιακής εικονικοποίησης με την χρήση VM με πολλούς τρόπους, ιδίως όσον αφορά την απόδοση και την δυνατότητα επέκτασης.

Επειδή τα Docker containers δεν χρησιμοποιούν επόπτη (hypervisor), οι διαθέσιμοι πόροι μπορούν να αξιοποιηθούν πιο αποδοτικά. Αυτό σημαίνει ότι μπορούν να υπάρχουν περισσότερα containers σε μία μηχανή από ό,τι εικονικές μηχανές. Με αυτήν την υψηλότερη πυκνότητα και την απουσία σπατάλης πόρων τα Docker containers μπορούν να έχουν καλύτερη απόδοση. Επίσης, τα containers είναι πολύ ελαφριά, και έτσι οι χρόνοι κατασκευής είναι γρήγοροι. Αυτό μειώνει τον χρόνο δοκιμών, ανάπτυξης και παραγωγής. Αφού κατασκευαστεί το container, μπορεί να προωθηθεί σε περιβάλλοντα δοκιμών και στη συνέχεια στο περιβάλλον παραγωγής.

Οι περισσότερες cloud-based πλατφόρμες χρειάζονται εύκολες λύσεις επεκτασιμότητας, και τα Docker containers αποτελούν εξαιρετική λύση. Σε αντίθεση με την λειτουργία πολλών εικονικών μηχανών που καταναλώνουν πολύτιμους πόρους λόγω της χρησιμοποίησης ολόκληρου του λειτουργικού συστήματος, το Docker μπορεί να εκκινεί πολύ γρηγορότερα περισσότερα container δίχως επιπλέον κόστος. Τα Docker containers μπορούν να λειτουργούν σε οποιοδήποτε σύστημα Linux, ενώ μπορούν επίσης να αναπτυχθούν σε διάφορα cloud περιβάλλοντα, data centers, φυσικούς servers κ.λπ. Οι χρήστες μπορούν εύκολα να μεταφέρουν containers από το cloud σε τοπικό περιβάλλον και πίσω στο cloud με γρήγορους ρυθμούς. Η αύξηση και μείωση της κλίμακας είναι επίσης απλή, καθώς οι χρήστες μπορούν να προσαρμόσουν την κλίμακα από ένα σε χιλιάδες και πάλι πίσω στο ένα, αν δεν χρειάζονται.

Ένα άλλο βασικό ζήτημα που πλεονεκτεί το docker είναι στον τομέα της ανάπτυξής λογισμικού και συγκεκριμένα στο ζήτημα των εξαρτήσεων (dependencies). Καθώς τα docker containers περιέχουν όλες τις απαραίτητες εξαρτήσεις για την λειτουργία ενός προγράμματος, μπορούν να αναπτυχθούν σε οποιοδήποτε σύστημα. Η φορητότητα αυτή προσφέρει μεγάλο πλεονέκτημα ευελιξίας λύνοντας σε μεγάλο βαθμό προβλήματα συμβατότητας των εφαρμογών με τα λειτουργικά συστήματα.

Το Docker αποτελεί επίσης ένα είδος "συμβολαίου" μεταξύ Προγραμματιστών και Διαχειριστών. Πολλές ομάδες λογισμικού αντιμετωπίζουν προβλήματα στην επιλογή της τεχνολογίας που θα χρησιμοποιηθεί, καθώς οι προγραμματιστές θέλουν να χρησιμοποιούν τις πιο νέες τεχνολογίες, ενώ οι διαχειριστές προτιμούν τεχνολογίες που λειτουργούν αξιόπιστα ή έχουν χρησιμοποιηθεί στο παρελθόν. Σε τέτοιες περιπτώσεις, το Docker είναι πολύ χρήσιμο, καθώς οι προγραμματιστές μπορούν να χρησιμοποιούν νέες τεχνολογίες μέσα σε containers, και οι διαχειριστές μπορούν να τα αναπτύξουν εύκολα.

8. Επίλογος

Η εικονικοποίηση μέσω Docker έχει φέρει επανάσταση στον τρόπο διαχείρισης και ανάπτυξης εφαρμογών, προσφέροντας μια ελαφριά και αποδοτική εναλλακτική λύση σε σχέση με τις παραδοσιακές τεχνολογίες εικονικοποίησης, όπως τις εικονικές μηχανές. Τα Docker containers εξασφαλίζουν απομόνωση εφαρμογών, συμβατότητα και φορητότητα σε διαφορετικά περιβάλλοντα, εξαλείφοντας κοινά προβλήματα που σχετίζονται με εξαρτήσεις και διαφορές μεταξύ συστημάτων. Με τη λεπτομερή ανάλυση των στοιχείων του Docker, όπως το Docker Engine, το Docker Registry και το Docker Client, γίνεται σαφές ότι το Docker προσφέρει ένα ολοκληρωμένο οικοσύστημα για την ανάπτυξη και τη διαχείριση εφαρμογών με τρόπο που εξυπηρετεί τόσο τους προγραμματιστές όσο και τις επιχειρήσεις.

Η χρήση του Docker, με τις βασικές εντολές και τη λειτουργικότητα που παρέχει, διευκολύνει τη διαδικασία της ανάπτυξης λογισμικού, μειώνοντας παράλληλα το χρόνο και το κόστος διαχείρισης. Μέσω εργαλείων όπως το Docker Compose, η δημιουργία σύνθετων περιβαλλόντων γίνεται πιο απλή, επιτρέποντας στους προγραμματιστές να διαχειρίζονται εύκολα πολλαπλά containers ως ένα ενοποιημένο σύστημα. Παράλληλα, οι τεχνολογίες ενορχήστρωσης, όπως το Docker Swarm και το Kubernetes, προσφέρουν ισχυρές δυνατότητες διαχείρισης για μεγαλύτερα, κατανεμημένα περιβάλλοντα, διασφαλίζοντας τη διαθεσιμότητα, την κλιμάκωση και την αποδοτικότητα των υποδομών. Αυτές οι τεχνολογίες καθιστούν το Docker ιδανικό για σύγχρονες αρχιτεκτονικές.

Το Docker δεν αποτελεί απλώς ένα εργαλείο, αλλά μια ολόκληρη φιλοσοφία στον τρόπο σχεδίασης, ανάπτυξης και λειτουργίας εφαρμογών. Οι δυνατότητές του έχουν καταστήσει δυνατή την ευελιξία και την προσαρμοστικότητα που απαιτείται στο σύγχρονο, ανταγωνιστικό τεχνολογικό περιβάλλον. Με τα πολλαπλά πλεονεκτήματα που προσφέρει, από τη φορητότητα και την εξοικονόμηση πόρων μέχρι τη δυνατότητα διαχείρισης σε κλίμακα, το Docker έχει αναδειχθεί σε θεμέλιο λίθο της σύγχρονης πληροφορικής, παρέχοντας στους οργανισμούς τα μέσα να ανταποκριθούν στις αυξανόμενες απαιτήσεις της αγοράς και να επιτύχουν τους στρατηγικούς τους στόχους.

Βιβλιογραφία – Πηγές

[1]. Joy, A. M. (2015). Performance comparison between Linux containers and virtual machines. *International Conference on Advances in Computer Engineering and Applications*. https://doi.org/10.1109/icacea.2015.7164727

[2]. Kropp, A., & Torre, R. (2021). *Docker: containerize your application. Computing in Communication Networks, 231–244.* https://doi.org/10.1016/B978-0-12-820488-7.00026-8

[3]. Moravcik, M., & Kontsek, M. (2020b). *Overview of Docker container orchestration tools. 2022 20th International Conference on Emerging eLearning Technologies and Applications (ICETA).* https://doi.org/10.1109/iceta51985.2020.9379236

[4]. Pahl, C. (2015). Containerization and the PaaS Cloud. *IEEE Cloud Computing*, *2*(3), 24-31. https://doi.org/10.1109/MCC.2015.51

[5]. Potdar, A. M., G, N. D., Kengond, S., & Mulla, M. M. (2020). Performance evaluation of docker container and virtual machine. *Procedia Computer Science*, *171*, 1419–1428. https://doi.org/10.1016/j.procs.2020.04.152

[6]. Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A., & Peterson, L. (2007). Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. *ACM SIGOPS Operating Systems Review*, *41*(3), 275-287. https://doi.org/10.1145/1272998.1273025

[7]. IBM, What is Virtualization. https://www.ibm.com/topics/virtualization

[8]. Official Docker Documentation. https://docs.docker.com/

[9]. Official VMware Documentation, VMs vs Containers. https://www.vmware.com/topics/vms-vs-containers

[10]. Official VMware Documentation, VMs vs Containers. https://www.vmware.com/topics/virtual-machine

[11]. Ostrowski S., (2024, March 23). containerd vs. Docker: Understanding Their Relationship and How They Work Together. https://www.docker.com/blog/containerd-vs-docker/

[12].TRLogic, (2018, July 4). Docker Compose. https://medium.com/@trlogic/docker-compose-57a51a34526c

[13].Türkal F., (2024, June 4). How does Docker ACTUALLY work? The Hard Way: A Comprehensive Technical Deep Diving. https://medium.com/@furkan.turkal/how-does-docker-actually-work-the-hard-way-a-technical-deep-diving-c5b8ea2f0422