ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη εφαρμογών σε απομονωμένες περιοχές χρηστών με την χρήση της πλατφόρμας Docker

Αύγουστος 2017

Αποκρεμιώτης Απόστολος, Λυπίτκας Κωνσταντίνος

Επιβλέπων καθηγητής
Παπαδάκης Νικόλαος

1. Εισαγωγή

Αυτή η πτυχιακή εργασία ασχολείται με το Docker μια καινούργια πλατφόρμα εικονικοποίησης (virtualization) που έχει ως στόχο να προσφέρει καινούργιες δυνατότητες και να βελτιώσει την ήδη υπάρχουσα δομή των τεχνολογιών εικονικοποίησης. Η "Εικονικοποίηση" αναφέρεται στην δημιουργία εικονικών μηχανών (virtual machines) οι οποίες έχουν ανεξάρτητα λειτουργικά συστήματα, ενώ η εκτέλεση των λογισμικών που τρέχουν σε εικονικές μηχανές διαχωρίζεται από τους πόρους του βασικού συστήματος. Θα αναφερθούμε στις διαφορές των κλασσικών εργαλείων και τεχνολογιών virtualization με την πλατφόρμα του Docker.

Στο κεφάλαιο 2 αναφέρονται οι χρήσεις της τεχνολογίας αυτής καθώς και οι δύο σημαντικότερες προσεγγίσεις της. Έπειτα αναλύονται τα εργαλεία και οι τεχνικές που χρησιμοποιούνται στις εικονικές μηχανές. Εν συνεχεία περιγράφεται μια απλή μέθοδος για την δημιουργία ενός απομονωμένου περιβάλλοντος (container), τα θετικά και τα αρνητικά της εικονικοποίησης βασισμένης σε "δοχεία" (containers) λογισμικού και διάφορα διαθέσιμα εργαλεία.

Στο τρίτο κεφάλαιο παρουσιάζεται το Docker, μια καινούργια virtualization πλατφόρμα και περιγράφεται η ιστορία του, η δομή του και αναλύεται η ασφάλεια, η απόδοση του και το μέλλον της πλατφόρμας. Στη συνέχεια δίνονται οδηγίες για την εγκατάσταση, τις κύριες λειτουργίες, τα εργαλεία του Docker και γίνεται σύγκριση του Docker με άλλες αντίστοιχες πλατφόρμες.

Στο τέλος προσφέρεται μια ματιά στην υλοποίηση, την χρήση του Docker API για την μεταφορά αρχείων μεταξύ δύο containers καθώς και την ενημέρωση του Docker.

Πίνακας περιεχομένων

Πίνακας εικόνων

2. Εικονικοποίηση

2.1 Περιπτώσεις χρήσης της εικονικοποίησης

Η εικονικοποίηση “Virtualization” χρησιμοποιείται σε πάρα πολλά συστήματα πληροφοριών. Εταιρείες όπως η Google, το Facebook, η Amazon και η Microsoft την χρησιμοποιούν σε διάφορα στάδια της παραγωγής τους, ωστόσο αυτή η τεχνολογία δεν χρησιμοποιείται μόνο σε συμπλέγματα διακομιστών (server clusters) εταιριών αλλά και σε εκατομμύρια προσωπικούς υπολογιστές.

Οι παρακάτω είναι κάποιες περιπτώσεις χρήσης της εικονικοποίησης:

- Στην περίπτωση μιας εφαρμογής που χρησιμοποιεί πολλούς πόρους του συστήματος όπως πχ. το Matlab, ενώ στους περισσότερους καινούριους προσωπικούς υπολογιστές δεν αντιμετωπίζει κάποιο πρόβλημα, τα δεδομένα αλλάζουν για ακαδημαϊκά και εταιρικά περιβάλλοντα λόγω παλιού υλικού (hardware) που δεν ανταποκρίνεται στις απαιτήσεις του προγράμματος και έτσι προτιμάται η αναπαραγωγή τέτοιων προγραμμάτων σε εικονικά περιβάλλοντα.

- Η διαδικασία των δοκιμών (testing) είναι ένα σημαντικό κεφάλαιο για κάθε κύκλο ανάπτυξης λογισμικού. Ειδικά σε μεγάλα projects συμβατά με διαφορετικές πλατφόρμες το testing περιλαμβάνει την δοκιμή του προϊόντος σε διαφορετικά διαμορφωμένα συστήματα. Διαφορετικά διαμορφωμένα συστήματα μπορεί να περιέχουν διαφορετική έκδοση λογισμικού και βιβλιοθηκών/εξαρτήσεων λογισμικού (dependencies) ή ακόμα και ένα εντελώς διαφορετικό λειτουργικό σύστημα. Οι διαθέσιμοι πόροι του συστήματος, το hardware και οι εγκατεστημένοι οδηγοί λογισμικού (drivers) είναι επιπλέον μεταβλητές που επηρεάζουν την συμπεριφορά του προγράμματος, οπότε η αλλαγή ή ο περιορισμός αυτών των πόρων μπορεί να αποτελεί ένα σημαντικό αντικείμενο του εκάστοτε testing. Αυτού του είδους οι δοκιμές δεν είναι πρακτικές πάνω σε φυσικούς υπολογιστές λόγω σπατάλης χρόνου και χρήματος.

- Οι εμπειρογνώμονες σε θέματα ασφαλείας συχνά έχουν την ανάγκη να τρέξουν μια εφαρμογή σε ένα απομονωμένο περιβάλλον (sandbox) προκειμένου να έχουν μια πλήρη εικόνα της συμπεριφοράς της, ειδικά όταν η στατική ανάλυση δεν προβαίνει σε σημαντικά αποτελέσματα. Επιπλέον, ακόμα και απλοί χρήστες μπορεί να είναι επιφυλακτικοί για το πως ένα μη αξιόπιστο πρόγραμμα μπορεί να επηρεάσει τον υπολογιστή τους. Τα πλεονεκτήματα που προσφέρονται από την δυνατότητα εκτέλεσης εφαρμογών σε απομονωμένα συστήματα κάνει τις τεχνολογίες εικονικοποίησης εξαιρετικά σημαντικές σε τέτοια σενάρια.
- Αρκετές εταιρίες ανάπτυξης λογισμικού δεν έχουν δικές τους υποδομές νέφους (cloud infrastructure) για διάφορους λόγους με τον κυριότερο να αποτελεί το κόστος συντήρησης, και έτσι βασίζονται σε τρίτους προκειμένου να μειώσουν αυτά τα κόστη και σε τεχνολογίες όπως η εικονικοποίηση. Από την άλλη πλευρά μια εταιρία που έχει παραπάνω υπολογιστική ισχύ από όση χρειάζεται μπορεί να νοικιάσει ένα μέρος της.

Υπάρχουν αρκετά προϊόντα cloud computing στην αγορά και κατηγοριοποιούνται ως εξής:

- **IaaS** – Η Infrastructure as a Service [1] αναφέρεται στην παροχή μεριδίων cloud υποδομών όπως servers, χώροι αποθήκευσης δεδομένων (data storages) ή δικτυακοί πόροι (network components). Ο πάροχος είναι υπεύθυνος για την συντήρηση του hardware, ενώ ο πελάτης πρέπει να φροντίσει για το software που θέλει να τρέξει. Κάποια ενδεικτικά παραδείγματα είναι τα παρακάτω: Google Compute Engine, Microsoft Azure ή Amazon Web Services, όπου τα δύο τελευταία χρησιμοποιούνται και σαν PaaS [1].
- **PaaS** – Όταν μια εφαρμογή ανεβαίνει στο cloud, είναι συχνή πρακτική το να χρησιμοποιούνται κομμάτια ενός ήδη υπάρχοντος περιβάλλοντος αντί να ξεκινάει από το μηδέν. Το Platform as a Service είναι βασισμένο στο IaaS, και αναφέρεται στο περιβάλλον που είναι ήδη σε μεγάλο βαθμό εγκατεστημένο και έτοιμο για να φιλοξενήσει (hosting) τις υπηρεσίες (services) του πελάτη. Σε αντίθεση με το IaaS, που επικεντρώνεται γύρω από την διαχείριση ολόκληρων υποδομών το PaaS συγκεντρώνεται σε ξεχωριστές εφαρμογές. Προϊόντα που χρησιμοποιούν συχνά το PaaS είναι τα OpenShift, Google App Engine και Heroku.
- **SaaS** – Βασισμένο στο PaaS, το Software as a Service [1] είναι ένα σύνολο εφαρμογών που έχουν ήδη εγκατασταθεί στο cloud και παρέχονται στους πελάτες. Σε αυτό το μοντέλο, ο πάροχος είναι υπεύθυνος για την αναβάθμιση και την διαθεσιμότητα των εφαρμογών. Τα Office 365, Gmail και άλλες εφαρμογές (apps) της Google είναι παραδείγματα του SaaS.

Η εικονικοποίηση παίρνει μέρος και στα 3 επίπεδα, ακόμα και στο IaaS καθώς συχνά δεν εκπροσωπεί απευθείας το hardware αλλά ένα σύνολο από πόρους απομονωμένους από το υπόλοιπο cloud με την χρήση εικονικοποίησης. Το περιβάλλον του λειτουργικού συστήματος στην περίπτωση του PaaS ή οι εφαρμογές στην περίπτωση του SaaS είναι επίσης απομονωμένα.

2.2 Εικονικοποίηση βασισμένη σε μια εικονική μηχανή

2.2.1 Εικονικοποίηση βασισμένη σε λογισμικό

Όταν αναφερόμαστε σε λογισμικό έχουμε πάντα στο πίσω μέρος του μυαλού μας την έννοια του χειροπιαστού υλικού από το οποίο αποτελείται κάποιος υπολογιστής, ωστόσο στα μοντέρνα υπολογιστικά συστήματα οι πόροι συνήθως διανέμονται ανάμεσα σε υπολογιστές που έχουν φυσική παρουσία αλλά και σε εικονικές μηχανές οι οποίες τρέχουν πάνω σε έναν υπολογιστή με φυσική παρουσία. Χωρίζουμε τα λειτουργικά συστήματα σε guest και host ανάλογα με τον τρόπο τον οποίο αυτά εκτελούνται και το εάν έχουν απευθείας πρόσβαση στο hardware του φυσικού μηχανήματος.

Τα guest λειτουργικά συστήματα συνήθως τρέχουν πάνω σε έναν hypervisor ο οποίος μπορεί να αποτελεί ένα κομμάτι software, firmware ή hardware το οποίο δημιουργεί, εκτελεί και διαχειρίζεται εικονοποιημένα μηχανήματα (virtual machines) που δεν έχουν απευθείας πρόσβαση στο hardware. Αυτό γίνεται κυρίως για λόγους ασφάλειας και απομόνωσης αφού διαφορετικά θα ήταν δυνατόν να βλάψουν τον host (φυσικό υπολογιστή) ή να πάρουν τον έλεγχο του.

Παρακάτω, αναλύονται οι δύο πιο σημαντικές επιπτώσεις στην επίδοση των εικονικοποιημένων μηχανών:

- **Επεξεργαστής** – Οι καινούριοι επεξεργαστές χρησιμοποιούν ένα σύστημα προστασίας των πόρων όπου πολλαπλοί κύκλοι δικαιωμάτων δημιουργούνται και κάθε ένας από αυτούς αντιπροσωπεύει ένα διαφορετικό επίπεδο εμπιστοσύνης. Η x86 αρχιτεκτονική επεξεργαστών προσφέρει τέσσερις κύκλους δικαιωμάτων (0-3) και συνήθως όσο μικρότερος είναι ο κύκλος τόσο πιο "έμπιστο" κώδικα αντιπροσωπεύει.

  Σε κάποιες περιπτώσεις η αρίθμηση των κύκλων δικαιωμάτων αναφέρεται στο τρέχων επίπεδο πρόσβασης – CPL. Το κομμάτι του λειτουργικού συστήματος που χρειάζεται τα περισσότερα δικαιώματα τρέχει στον κύκλο 0, τα drivers των συσκευών τρέχουν στον κύκλο 1 ή σε περισσότερους από έναν κύκλους και για τις περισσότερες εφαρμογές ο υψηλότερος βαθμός κύκλου δικαιωμάτων είναι αρκετός.

  Όταν μια εφαρμογή χρειαστεί τέτοιου είδους εντολές, καλεί μια συνάρτηση του λειτουργικού συστήματος το οποίο θέτει σε λειτουργία τον kernel κώδικα που τρέχει στον κύκλο 0. Όταν η εκτέλεση της συνάρτησης ολοκληρωθεί τότε γυρνάει το κατάλληλο αποτέλεσμα και συνεχίζει η εκτέλεση του κώδικα της εφαρμογής και τα δικαιώματα ανακτώνται. Αυτός ο μηχανισμός λέγεται system call και ο χαμηλότερος κύκλος δικαιωμάτων που τρέχουν

οι εφαρμογές αντιπροσωπεύουν το userspace στο οποίο έχει πρόσβαση ο εκάστοτε χρήστης, και το επίπεδο του kernel καλείται kernelspace.

Από την πλευρά της εικονικοποίησης αυτό προσφέρει έναν μηχανισμό ασφάλειας που διασφαλίζει την ασφάλεια της ιδιοκτησίας του virtual machine monitor. Με το να μην τρέχει ο κώδικας του guest στο κύκλο δικαιωμάτων 0, το host λειτουργικό σύστημα είναι το μόνο με τον πλήρη έλεγχο των πόρων του υπολογιστή. Ωστόσο όταν ο guest προσπαθήσει να εκτελέσει ένα system call θα αποτύχει καθώς δεν μπορεί να τρέξει στον 0 (πρώτο) κύκλο δικαιωμάτων.

Για να ξεπεραστεί αυτό το εμπόδιο πρέπει να χρησιμοποιηθεί μια τεχνική που ονομάζεται paravirtualization, η οποία αναλαμβάνει την εκτέλεση των εντολών οι οποίες είναι πολύ δύσκολο να τρέξουν μέσα σε εύλογο χρονικό διάστημα σε ένα virtualized περιβάλλον συγκριτικά με ένα non-virtualized περιβάλλον. Ακόμη πρέπει να εγκατασταθούν ειδικοί drivers για να επιτρέψουν την επικοινωνία του host (φυσικού μηχανήματος) με τον hypervisor.

Μια επιπλέον δυνατότητα είναι η αντικατάσταση των εντολών που απαιτούν επιπλέον δικαιώματα εκτέλεσης κατά το runtime ελέγχοντας τον τρέχων κώδικα και κάνοντας τις απαραίτητες αλλαγές όπου είναι αναγκαίο.

- **Μνήμη** – Οι μοντέρνοι επεξεργαστές χρησιμοποιούν την έννοια της εικονικοποιημένης μνήμης (virtual memory), με την οποία κάθε εφαρμογή έχει την ψευδαίσθηση ότι μπορεί να χρησιμοποιήσει ολόκληρη την μνήμη του συστήματος για τον εαυτό της. Για να λειτουργήσει αυτή η τεχνική, το λειτουργικό σύστημα πρέπει να καθορίσει ποια μέρη της φυσικής μνήμης αντιστοιχούν στην εικονική μνήμη. Εάν η συνολική μνήμη που χρησιμοποιήθηκε από τις εφαρμογές είναι μεγαλύτερη από την φυσική μνήμη, μέρη της virtual μνήμης μεταφέρονται σε ένα δευτερεύον μέσο αποθήκευσης (πχ. σκληρός δίσκος).

  Εφόσον το guest λειτουργικό σύστημα δεν γνωρίζει ότι ο host ήδη έχει τον έλεγχο της virtual μνήμης, παρουσιάζεται ένα σημαντική μείωση στην απόδοση των virtual machines, επειδή κάθε πρόσβαση στην μνήμη από τον guest πρέπει να καταγραφεί πρώτα στο address space του guest και ύστερα στο χώρο διευθύνσεων μνήμης του host. Η λύση είναι η απόκρυψη του page table (η δομή που καταγράφει τις αντιστοίχισεις της μνήμης), έτσι ώστε κάθε φορά που ο guest καταγράφει ένα στοιχείο στη μνήμη, ο hypervisor εκτελεί απευθείας μια καταγραφή της διεύθυνσης μνήμης που χρησιμοποιήθηκε στον host στο shadow table-ένας προσωρινός πίνακας αντιστοιχίσεων των διευθύνσεων μνήμης. Ύστερα όταν ο guest δοκιμάσει να πάρει πρόσβαση στην μνήμη, το shadow table χρησιμοποιείται αντί για ένα πίνακα καταγραφών ελεγχόμενο από τον guest.

2.2.2 Εικονικοποίηση υποβοηθούμενη από το υλικό

Μέχρι το 2005-2006 δεν υπήρχε υποστήριξη για virtualization από πλευράς hardware του υπολογιστή μέχρι την κυκλοφορία των AMD-V για τους AMD επεξεργαστές και του INTEL VT για τους Intel επεξεργαστές. Ενώ και οι δύο τεχνολογίες είναι παρόμοιες στην λειτουργία τους, η ορολογία τους διαφέρει και για το παρακάτω κεφάλαιο θα χρησιμοποιηθεί η ορολογία της AMD-V τεχνολογίας.

Επεξεργαστής

Μια νέα δομή παρουσιάστηκε η VMCB (Virtual Machine Control Block) [2], που αντιπροσωπεύει ένα virtual machine μέσα στον επεξεργαστή. Όταν ένα VMCB εκτελείται, ο επεξεργαστής εκτελεί τα ακόλουθα βήματα:

- Η κατάσταση (state) του host συστήματος αποθηκεύεται σε μια περιοχή της μνήμης που ορίζει το VMCB.
- Η κατάσταση (state) του guest συστήματος φορτώνεται από την περιοχή της μνήμης που ορίζει το VMCB.
- Ο κώδικας του guest αρχίζει να εκτελείται.

Η VCMB δομή περιέχει επίσης το επιθυμητό επίπεδο CPL, επιτρέποντας στο virtual machine να τρέξει ακόμα και στον κύκλο δικαιωμάτων 0. Ωστόσο μερικές λειτουργίες, όπως απευθείας πρόσβαση στις συσκευές εισόδου εξόδου απαγορεύονται και προκαλούν το virtual machine να αποχωρήσει από το guest state. Μετά την αποχώρηση του, το virtual machine monitor μπορεί να αποφασίσει να αλλάξει διάφορα πεδία στο VCMB, με αποτέλεσμα την εξομοίωση των συσκευών εισόδου εξόδου και την εκτέλεση του VMRUN για ακόμα μια φορά.

Μνήμη

Η Intel και η AMD έχουν ανακαλύψει έναν μηχανισμό που εξαλείφει την ανάγκη χρήσης των shadow tables. Τα Extended Page Tables [3] της Intel και το Rapid Virtualization Indexing [4] της AMD είναι παραδείγματα μετάφρασης διεύθυνσης δεύτερου επιπέδου, όπου οι εμφωλευμένοι πίνακες σελίδων δημιουργούνται και διατηρούνται από το υλικό. Σε αυτό το μοντέλο, το υλικό εκτελεί και τις μεταφράσεις από το virtual address space του guest στο φυσικό χώρο διευθύνσεων μνήμης του guest καθώς και από το φυσικό χώρο διευθύνσεων μνήμης του guest στο φυσικό χώρο διευθύνσεων του host.

2.2.3 VirtualBox

Το VirtualBox [5] είναι ένα ελεύθερο λογισμικό δευτέρου τύπου hypervisor, το οποίο είχε αναπτυχθεί αρχικά από την Innotek GmbH, μια γερμανική εταιρία, η οποία στην συνέχεια εξαγοράστηκε από την Sun Microsystems. Σήμερα, το VirtualBox είναι μέρος των προϊόντων της Oracle καθώς από το 2010, η Oracle εξαγόρασε την Sun Microsystems. Έτσι το επίσημο όνομα του προϊόντος είναι το Oracle VM Virtualbox.

Το VirtualBox υποστηρίζει όλα τα μεγάλα λειτουργικά συστήματα είτε σαν host ή guest με εξαίρεση το OS X σε συστήματα που δεν έχουν επίσημο Apple υλικό (αν και είναι δυνατή η εγκατάσταση "σπασμένου" image του OS X). Επίσης, υποστηρίζονται και οι software και οι υποβοηθούμενοι από το υλικό (hardware assisted) virtualization μεθόδοι. Οι 64-bit guest υποστηρίζονται εάν ο host είναι επίσης 64-bit ή επιτρέπεται το 64-bit virtualization στον επεξεργαστή.

Τα υπόλοιπα αξιοσημείωτα χαρακτηριστικά του Virtualbox είναι οι κοινόχρηστοι φάκελοι, ο κοινόχρηστος χώρος αποθήκευσης σε προσωρινή μνήμη (clipboard) και το cloning των virtual machines. Ένα ολοκληρωμένο state του κάθε ενεργού virtual machine (Snapshot) μπορεί να αποθηκευτεί σε ένα αρχείο για να χρησιμοποιηθεί στο μέλλον. Το VirtualBox επίσης προσφέρει ένα πλούσιο GUI interface, ένα command line interface, ένα διαδραστικό Python Shell καθώς και ένα web API. Μια φορητή έκδοση που δεν χρειάζεται εγκατάσταση είναι επίσης διαθέσιμη με το όνομα Portable-VirtualBox.

Figure 2.1: Το γραφικό περιβάλλον του VirtualBox σε Linux Ubuntu.

2.2.4 VMWare Player

Το VMWare Player [6] είναι ένα λογισμικό virtualization δημιουργημένο από την εταιρία VMWare. Ενώ η VMWare προσφέρει ένα μεγάλο εύρος λογισμικού virtualization, θα αναφερθούμε στο VMWare Player και εν ολίγοις στα υπόλοιπα προϊόντα της εφόσον το VMWare Player είναι το μοναδικό που μπορεί να χρησιμοποιηθεί δωρεάν, και έτσι θα είναι δυνατή μια καλύτερη σύγκριση του με το Docker που είναι επίσης δωρεάν αν και η άδεια του το περιορίζει σε μη-εμπορική χρήση.

Οι δυνατότητες του VMWare Player διαφέρουν ελάχιστα από αυτές του VirtualBox. μια σημαντική διαφορά του VMWare Player είναι ότι δεν υποστηρίζεται η εγκατάστασή του σε OS X λειτουργικά συστήματα ενώ προσφέρει την δυνατότητα εγκατάστασης του OS X σε guest συστήματα ακόμα και σε υλικό μη εξουσιοδοτημένο από την Apple. Άλλη μια διαφορά είναι η έλλειψη της δυνατότητας αποθήκευσης snapshots, που παρέχεται μόνο στα εμπορικά προϊόντα της VMWare. Για κάποιες δυνατότητες όπως τον κοινόχρηστο χώρο προσωρινής αποθήκευσης στην μνήμη (clipboard) και φακέλους, το VMWare Player απαιτεί την εγκατάσταση των VMWare εργαλείων στο guest σύστημα.

Όμοια με το API του VirtualBox το VIX API του VMWare δίνει την δυνατότητα ελέγχου των virtual machines προγραμματιστικά. Το VIX επίσης διαθέτει ένα εργαλείο γραμμής εντολών (command line interface - cli) το vmrun, το οποίο δίνει την δυνατότητα στο VMWare Player να λειτουργήσει χωρίς γραφικό περιβάλλον (GUI).

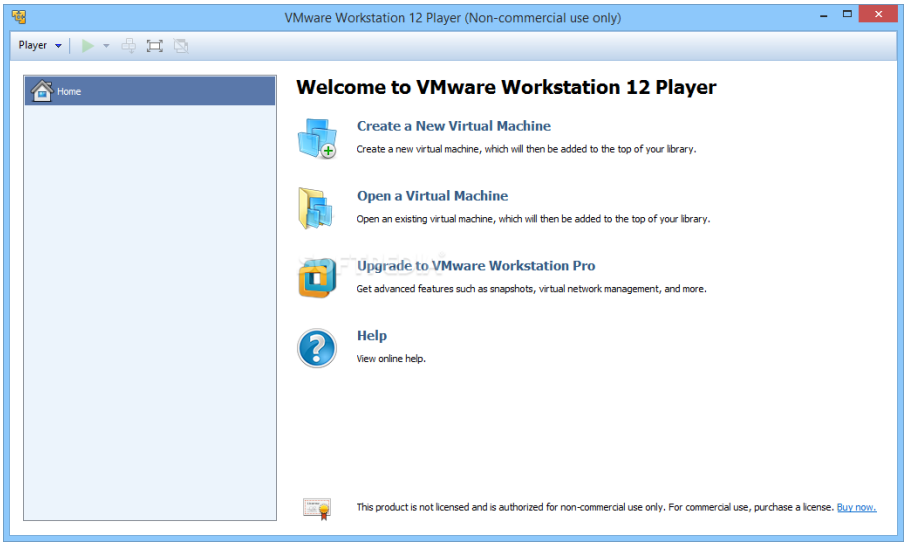

Figure 2.2: Το γραφικό περιβάλλον του VMWare Player 12.

2.2.5 Άλλα διαθέσιμα εργαλεία εικονικοποίησης

Το **Quick Emulator (Qemu)** [7] είναι ένα ελεύθερης χρήσης virtual machine monitor. Τρέχει σε Linux, Windows και OSX. Η λειτουργία του είναι το emulation ενός υλικού και έτσι επιτρέπει στον host να τρέχει λογισμικό γραμμένο για διαφορετική αρχιτεκτονική. Δίνεται η δυνατότητα εκτέλεσης μιας εφαρμογής ή ενός ολόκληρου λειτουργικού συστήματος. Όσον αφορά την απόδοση, το emulation του instruction set ενός επεξεργαστή είναι μια βαριά διαδικασία οπότε το Qemu συχνά χρησιμοποιείται μαζί με ένα άλλο λογισμικό virtualization το οποίο ελέγχει την εκτέλεση.

KVM – kernel-based virtual machine [8], είναι ένα extension του Linux Kernel, που του προσφέρει δυνατότητες τύπου 2 hypervisor. Το KVM λειτουργεί με το να εκθέτει ένα API, το οποίο χρησιμοποιείται για την διεπαφή με κάποιες λειτουργίες του kernel και του υλικού. Αυτό το API ύστερα χρησιμοποιείται από κάποιον client όπως τον Qemu. Αυτό δίνει την δυνατότητα στο λογισμικό που γίνεται host από το Qemu να τρέξει πολύ πιο γρήγορα. Χρησιμοποιείται το Hardware-assisted virtualization οπότε χρειάζεται ένας επεξεργαστής που να το υποστηρίζει.

Xen [9] – αρχικά κυκλοφόρησε από το πανεπιστήμιο του Cambridge το 2003 και είναι ένα παράδειγμα Hypervisor τύπου 1. Κατέληξε να είναι ο νούμερο 1 Open source hypervisor σύμφωνα με αναλυτές όπως το Gartner. Συντηρητικές εκτιμήσεις δείχνουν ότι το Xen έχει μια βάση χρηστών που ανέρχεται στα 10+ εκατομμύρια οι οποίοι είναι πραγματικοί χρήστες και όχι απλά εγκαταστάσεις του hypervisor, το οποίο κάνει αυτά τα νούμερα ακόμα ποιο σημαντικά. Η cloud υποδομή της Amazon (Amazon Web Services) [10] εκτελούν πάνω από μισό εκατομμύριο εικονικοποιημένες Xen υποστάσεις, σύμφωνα με μια πρόσφατη έρευνα ενώ και οι υπόλοιποι cloud providers όπως το Rackspace και εταιρίες hosting χρησιμοποιούν το hypervisor σε μεγάλη κλίμακα. Εταιρείες όπως η Google και η Yahoo χρησιμοποιούν τον hypervisor για τις εσωτερικές τους δομές. Στην ορολογία του Xen, οι εγκατεστημένοι host καλούνται domains, που από αυτά ακριβώς ένα είναι το προνομιούχο domain0. Το Domain0 χρησιμοποιείται για την διαχείριση και τον έλεγχο των υπόλοιπων μη προνομιούχων domain (DomainU guests) και περιέχει network και disc drivers για paravirtualized guests.

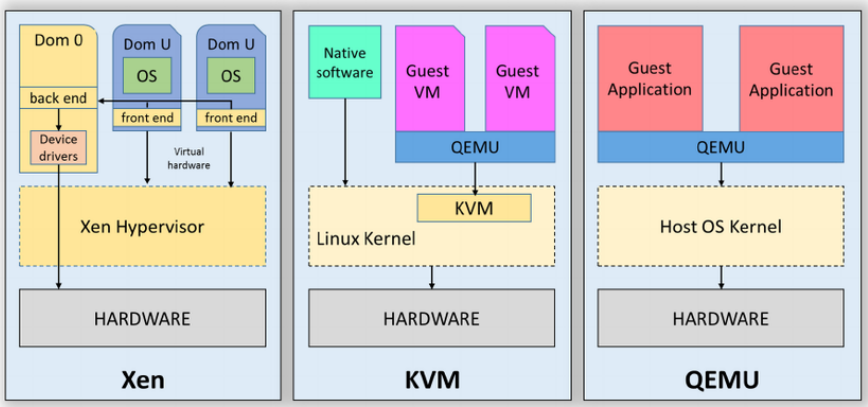

Figure 2.3: Δομή και σύγκριση των Qemu, KVM, Xen. [11]

2.3 Εικονικοποίηση βασισμένη σε containers λογισμικού

Η παροχή ενός απομονωμένου περιβάλλοντος μέσα στο λειτουργικό σύστημα φιλοξενίας ονομάζεται εικονικοποίηση επιπέδου λειτουργικού συστήματος και ένα τέτοιο απομονωμένο περιβάλλον ονομάζεται container. Ένα container είναι ένα αυτοδύναμο περιβάλλον εκτέλεσης λογισμικού το οποίο μοιράζεται τον ίδιο πυρήνα (kernel) του λειτουργικού συστήματος φιλοξενίας και το οποίο είναι προαιρετικά απομονωμένο από τα υπόλοιπα containers στο σύστημα. Οι πιο συχνά χρησιμοποιούμενες τεχνολογίες είναι οι Solaris Containers [12] ή Zones, OpenVZ [13], FreeBSD jails [14] και LXC [15].

2.3.1 Chroot και jail

Κατά την ανάπτυξη του Linux, προέκυψε η ιδέα για την απομόνωση μιας διεργασίας από το σύστημα αρχείων του host και έτσι δημιουργήθηκε η εντολή chroot ακριβώς για αυτό τον σκοπό. Το chroot που είναι τα αρχικά για το change root είναι ταυτόχρονα ένα λογισμικό βοήθειας αλλά και μια κλήση συστήματος που επιτρέπει τον προσδιορισμό ενός νέου root directory εκτός του "/". Έτσι η διεργασία δεν μπορεί να αποκτήσει πρόσβαση σε αρχεία υψηλότερης ιεραρχίας του καινούργιου root directory ενώ εξωτερικά προγράμματα μπορούν ακόμα να δουν μέσα στο καινούργιο root directory.

Αυτή την περίοδο το chroot χρησιμοποιείται για να παρέχει απομονωμένα περιβάλλοντα για την δοκιμή άγνωστων και ασταθών εφαρμογών ή για την ανακάλυψη ανεπιθύμητων εξαρτήσεων λογισμικού. Εργαλεία κατασκευής πακέτων λογισμικού όπως το Pbuilder για Debian ή το Mock για Fedora επίσης χρησιμοποιούν το chroot για να παρέχουν απομονωμένα περιβάλλοντα ελέγχου σε διάφορες διανομές Linux.

Ένας προχωρημένος μηχανισμός δημιουργημένος με βάση το chroot είναι ο jail, ο οποίος είναι διαθέσιμος στο FreeBSD από το 2000 και επιτρέπει την δημιουργία απομονωμένου εικονικού περιβάλλοντος το οποίο τρέχει στο host μηχάνημα και έχει τα δικά του αρχεία, διεργασίες καθώς και λογαριασμούς χρηστών (απλών αλλά και διαχειριστών-root accounts). Μέσα από μια jailed διεργασία το περιβάλλον είναι σχεδόν απροσδιόριστο από ένα αληθινό σύστημα. Το jail μπορεί επομένως να προσδιορίσει έναν νέο root χρήστη, ο οποίος έχει απόλυτο έλεγχο μέσα σε αυτό, αλλά δεν μπορεί να έχει πρόσβαση σε τίποτα εξωτερικά του.

Το jail ωστόσο, δεν επιτυγχάνει πραγματικό virtualization αφού δεν επιτρέπει στις εικονικές μηχανές να τρέχουν διάφορες εκδόσεις του πυρήνα εκτός από εκείνη του βασικού (host) συστήματος. Όλοι οι virtual servers μοιράζονται τον ίδιο πυρήνα και συνεπώς μπορούν να εκτεθούν στα ίδια σφάλματα και τις ίδιες πιθανές ευπάθειες ασφαλείας. Δεν υπάρχει υποστήριξη για clustering ή proccess migration, έτσι ο host υπολογιστής εξακολουθεί να αποτελεί ένα μοναδικό σημείο αποτυχίας (single point of failure) για όλους τους εικονικούς διακομιστές. Είναι δυνατόν να χρησιμοποιηθεί το jail για να δοκιμαστεί με ασφάλεια νέο λογισμικό αλλά όχι για να δοκιμαστούν με ασφάλεια νέοι πυρήνες (kernels).

2.3.2 Namespaces

Οι περιοχές ονομάτων ή αλλιώς **namespaces** είναι βασικά χαρακτηριστικά του πυρήνα των Linux για την υποστήριξη της ελαφριάς εικονικοποίησης μέσω της απομόνωσης των πόρων του συστήματος μιας συλλογής διεργασιών. Ο σκοπός του κάθε namespace είναι η κάλυψη ενός καθολικού πόρου του συστήματος σε ένα αφηρημένο αντικείμενο το οποίο παρουσιάζεται στις διεργασίες μέσα στο εκάστοτε namespace ως η δική του απομονωμένη εκδοχή του συνολικού πόρου.

Για την καλύτερη κατανόηση των namespaces, θα επικεντρωθούμε στους ατομικούς τύπους τους. Στο Linux υπάρχουν έξι διαθέσιμες περιοχές ονομάτων:

- **Mount (mnt)** – Για να κατανοήσουμε την έννοια του mount namespace πρέπει πρώτα να κατανοήσουμε τα mount points. Mount point στο Linux δεν είναι τίποτα αλλό παρά ένας φάκελος στο διαθέσιμο σύστημα (συνήθως κενός) μέσω του οποίου προσαρτάται σε αυτό ένα επιπλέον σύστημα αρχείων (συνήθως με φυσική παρουσία όπως πχ. ένας σκληρός δίσκος ή ένα usb stick). Το mount namespace επιτρέπει τον έλεγχο των mount points των συστημάτων αρχείων και δίνει σε καθένα από αυτά μια ξεχωριστή διεργασία η οποία σχετίζεται με αυτό και επιτρέπει πρόσβαση μόνο σε συγκεκριμένους πόρους.

- **Process ID (pid)** — Το PID namespace δίνει σε κάθε διεργασία του λειτουργικού συστήματος ένα πλήθος από μοναδικά αναγνωριστικά (PIDs) τα οποία είναι εμφωλευμένα κάτι που σημαίνει πως όταν μια νέα διεργασία ξεκινά θα έχει ένα ξεχωριστό PID για κάθε namespace από το υπάρχον namespace της έως το αρχικό namespace της διεργασίας που την ξεκίνησε. Έτσι το αρχικό PID namespace έχει πρόσβαση και μπορεί να "δει" όλες τις διεργασίες με διαφορετικά PIDs από ότι τα υπόλοιπα namespaces τα οποία μπορούν να δουν μόνο όλες τις διεργασίες "παιδιά" τους (child processes). Αυτό σημαίνει πως εάν τερματιστεί η πρώτη διεργασία του συστήματος (με PID = 1) τότε αυτό θα τερματίσει όλες τις διεργασίες στο namespace της συγκεκριμένης καθώς και τους απογόνους τους που πρακτικά σημαίνει τον τερματισμό του συστήματος.

- **Network (net)** — Τα δικτυακά namespaces πρακτικά εικονικοποιούν την στοίβα του δικτύου. Κατά την δημιουργία του, ένα network namespace περιέχει μόνο ένα loopback interface το οποίο ουσιαστικά είναι ένα καταληκτικό σημείο επικοινωνίας (communication endpoint). Κάθε διεπαφή δικτύου είτε εικονική είτε φυσική υπάρχει σε ακριβώς ένα namespace και μπορεί να μεταφερθεί ανάμεσα σε namespaces. Κάθε namespace δικτύου έχει ένα πλήθος από ιδιωτικές IP διευθύνσεις, τον δικό του πίνακα δρομολόγησης (routing table), μια λίστα με τα διαθέσιμα sockets, έναν πίνακα με τις υπάρχουσες συνδέσεις, τείχος προστασίας καθώς και άλλους δικτυακούς πόρους. Κατά την καταστροφή ενός τέτοιου namespace θα καταστραφούν και οποιεσδήποτε εικονικές διεπαφές δικτύου μέσα σε αυτό.

- **Interprocess Communication (ipc)** — Το IPC είναι ένας μηχανισμός ο οποίος επιτρέπει τον διαμοιρασμό πληροφορίας ανάμεσα σε 2 οντότητες μέσω των διεργασιών τους. Αυτές οι 2 οντότητες μπορεί να είναι μηχανήματα στο ίδιο είτε σε διαφορετικό δίκτυο ακόμη και διεργασίες στο ίδιο μηχάνημα, ενώ το μοντέλο σύνδεσης τους συνήθως είναι client-server. Τα IPC namespaces απομονώνουν κυρίως πόρους που αφορούν το System V [16] (μια πρώιμη μορφή του λειτουργικού συστήματος Unix). Όταν αναφερόμαστε στο ίδιο σύστημα τότε ο τρόπος απομόνωσης αφορά κυρίως τα σημεία της μνήμης στα οποία έχουν πρόσβαση δύο διαφορετικές διεργασίες, έτσι με τη χρήση του IPC namespace αποτρέπονται διεργασίες από διαφορετικά namespaces να έχουν πρόσβαση σε ένα σετ εντολών που τους επιτρέπει να επικοινωνήσουν μεταξύ τους διαμέσου της μνήμης (RAM).

- **UTS** — Τα UTS namespaces επιτρέπουν σε ένα ενιαίο σύστημα να εμφανίζει σε κάποιες διεργασίες διαφορετικό hostname καθώς και domain name.

- **User ID (user)** – Τα user namespaces είναι ένα χαρακτηριστικό του πυρήνα του Linux που επιτρέπει την απομόνωση των δικαιώματων του χρήστη καθώς και την αναγνώριση του χρήστη ανάμεσα σε πολλαπλά πλήθη διεργασιών. Η δομή του user namespace είναι παρόμοια με αυτή του PID namespace που σημαίνει ότι και αυτά τα namespaces είναι εμφωλευμένα καθώς και κάθε νέο namespace είναι απόγονος του user namespace που το δημιούργησε. Το user namespace περιέχει έναν πίνακα αντιστοίχισης (mapping table) που μετατρέπει user IDs από την πλευρά του container στο οποίο βρίσκεται στην πλευρά του συστήματος. Αυτό επιτρέπει για παράδειγμα στον root user να έχει uid 0 στο container αλλά το σύστημα να του συμπεριφέρεται με το id 1,400,000 όταν πραγματοποιεί ελέγχους ιδιοκτησίας. Για να πετύχουμε απομόνωση των δικαιωμάτων του χρήστη καθώς και των διαχειριστικών πράξεων που ίσως προσπαθήσει να πραγματοποιήσει κάθε τύπος namespace θεωρείται ότι ανήκει σε κάποιο user namespace με βάση τα δικαιώματα της διεργασίας που το δημιούργησε την στιγμή που το δημιούργησε. Έτσι το user namespace μπορεί να έχει πρόσβαση με δικαιώματα διαχειριστή σε άλλα namespaces του συστήματος ανάλογα την στιγμή δημιουργίας τους.

- **cgroup** – Το cgroup είναι ένα χαρακτηριστικό του πυρήνα του Linux το οποίο περιορίζει, ελέγχει και απομονώνει την χρήση των πόρων του συστήματος (CPU, μνήμη RAM, I/O λειτουργίες στον δίσκο, χρήση του δικτύου κ.ο.κ). Κάθε διεργασία του συστήματος ανήκει σε κάποιο control group και κάθε control group αντιστοιχεί σε μια διεργασία, με αυτόν τον τρόπο η διεργασία δεν έχει πρόσβαση σε άλλα control groups εξωτερικά του container στο οποίο ανήκει.

Τα namespaces αυτά δημιουργούνται και διαχειρίζονται με τις παρακάτω κλήσεις συστήματος (syscalls):

- **clone** – ορίζει σε ποιο νέο namespace θα μεταφερθεί μια νέα διεργασία.

- **unshare** – ορίζει σε ποιο νέο namespace θα μεταφερθεί μια υπάρχουσα διεργασία.

- **setns** – επιτρέπει τον επανορισμό κάποιου namespace με βάση κάποιο file descriptor το οποίο μπορεί να είναι για παράδειγμα κάποιο handle σε ένα αρχείο.

Εάν κάποιο namespace πλέον δεν ορίζεται ή κάποια διεργασία δεν παραπέμπει σε αυτό, τότε αυτό καταστρέφεται και ο χειρισμός των πόρων που υπάρχουν μέσα σε αυτό ή που εξαρτώνται από αυτό ποικίλει ανάλογα με το είδος του namespace.

2.3.3 Εργαλεία εικονικοποίησης βασισμένα σε container τεχνολογίες

LXC (Linux containers) [15] – Συνδυάζουν τις τεχνολογίες και τα χαρακτηριστικά που προσφέρονται από τα namespaces καθώς και τα control groups με σκοπό να δημιουργήσουν ένα πλήρως απομονωμένο περιβάλλον το οποίο μπορεί κάποιος εύκολα να διαχειριστεί - ένα container. Επειδή τα containers είναι η βασική μονάδα με την οποία ασχολείται το LXC, αυτό προσφέρει όλες τις βασικές λειτουργίες που χρειάζονται όπως την δημιουργία, την έναρξη, το σταμάτημα της εκτέλεσης ενός container καθώς και την εμφάνιση τους σε λίστα και την διαγραφή τους. Εφόσον ένα κενό container δεν χρήζει σημασίας για τον χρήστη, το LXC δίνει την δυνατότητα για δημιουργία προτύπων (templates) και containers βασισμένων σε αυτά τα πρότυπα έτσι ώστε να δημιουργούνται με κάποιες προεπιλεγμένες ρυθμίσεις όπως το σύστημα αρχείων και άλλες ρυθμίσεις-διαμορφώσεις του συστήματος. Επιπλέον το LXC δίνει την δυνατότητα στον χρήστη να "παγώσει" και να "ξεπαγώσει" κάποιο container, ουσιαστικά δηλαδή να σταματήσει την εκτέλεση του και να μπορεί να την συνεχίσει σε κάποιο μετέπειτα χρονικό σημείο. Επιτρέπεται επίσης η δημιουργία checkpoints το οποίο είναι ένα χρονικό σημείο στο οποίο καταγράφεται η παρούσα κατάσταση του container καθώς και πληροφορίες για αυτό και μπορεί κάποιος αργότερα να επαναφέρει το container στο συγκεκριμένο checkpoint. Η κλωνοποίηση ενός container είναι ένα ακόμη από τα χαρακτηριστικά του LXC καθώς και ο έλεγχος τους προγραμματιστικά με την χρήστη ενός API (Application Programming Interface) που διαθέτει το LXC.

Parallels OpenVZ (Virtuozzo) [13] – Είναι μια ακόμη τεχνολογία βασισμένη σε container εικονικοποίηση. Το συγκεκριμένο εργαλείο διατίθεται από την εταιρεία Parallels και είναι πρόγονος του LXC καθώς μεγάλο κομμάτι του κώδικα του LXC προέρχεται από το OpenVZ ή από μέλη της ομάδας του OpenVZ. Η μεγάλη διαφορά του LXC σε σχέση με το OpenVZ είναι ότι το OpenVZ χρησιμοποιεί δικό του kernel (πυρήνα) ο οποίος προέρχεται από την έκδοση 2.6 του πυρήνα του Linux. Παρόλα αυτά οι περισσότερες από τις αλλαγές που απαιτούσε για να τρέξει το OpenVZ από το kernel έχουν μεταφερθεί πλέον και στους επισήμους kernel του Linux οπότε μπορεί κάποιος να εκτελέσει μια έκδοση του OpenVZ σε επίσημο πυρήνα με μειωμένες ωστόσο δυνατότητες και λιγότερα χαρακτηριστικά. Παρόλο που το OpenVZ μοιάζει απαρχαιωμένο πολλοί διαχειριστές συστημάτων το προτιμούν αφού έχει αποδείξει την αξία του σε περιβάλλοντα παραγωγής αρκετά χρόνια τώρα, καθώς επίσης έχει και χαρακτηριστικά τα οποία δεν περιλαμβάνονται στο LXC όπως live migration (μια διαδικασία μεταφοράς του συστήματος ενώ αυτό εκτελείται από ένα μηχάνημα σε ένα άλλο) κάνωντας suspend στο ένα μηχάνημα και resume στο άλλο που είναι χαρακτηριστικά που χρησιμοποιούνται ιδιαίτερα σε VPS (Virtual Private Servers).

Υπάρχουν πολλά εργαλεία εικονικοποίησης, και επειδή μια εταιρεία συνήθως έχει παραπάνω από ένα εικονικά μηχανήματα και containers το καθένα δημιουργημένο με διαφορετική τεχνολογία, η διαχειρισή τους μπορεί να αποδειχθεί ένα δύσκολο και πολύπλοκο έργο. Για την αντιμετώπιση του προβλήματος αυτού, ένα εργαλείο με το όνομα **libvirt** παρέχει ένα ενοποιημένο API (Application Programming Interface) υποστηρίζοντας κάθε hypervisor που αναφέρθηκε καθώς και αρκετούς άλλους ακόμη και από τους container-based hypervisors. Είναι γραμμένο σε C αλλά παρέχει δυνατότητες σύνδεσης με όλες σχεδόν τις δημοφιλείς γλώσσες προγραμματισμού όπως η C, Python, Perl, Java και άλλες. Είναι cross-platform και δουλεύει σε συστήματα Linux, FreeBSD, Windows καθώς και OSX.

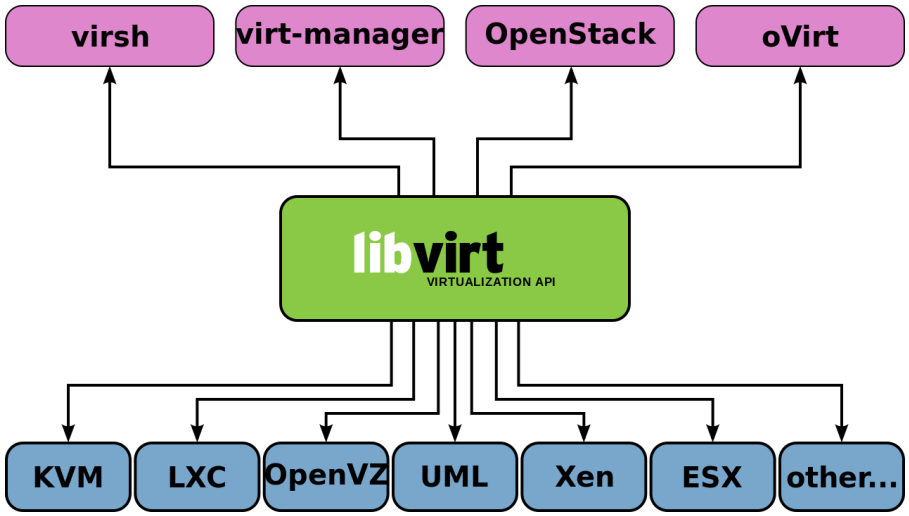

Figure 2.4: Το ενοποιημένο API του libvirt.
[17]

2.3.4 Σύγκριση των τεχνολογιών container με τα virtual machines

Τα εικονικοποιημένα μηχανήματα δημιουργούν μια νέα εκδοχή του λειτουργικού συστήματος για κάθε εκτέλεση του virtual machine. Αυτό έχει αρκετά οφέλη όπως η δυνατότητα να τρέξει κάποιος τελείως διαφορετικό guest λειτουργικό σύστημα σε σχέση με το host, ωστόσο έχει επίσης και αρκετά μειονεκτήματα αφού υπάρχει μεγάλη καθυστέρηση κατά την έναρξη και την εκτέλεση που προκαλείται από την επιδιόρθωση των εντολών μέσω του machine monitor καθώς και την μετάφραση εντολών που δεν μπορούν να εκτελεστούν για λόγους δικαιωμάτων (paravirtualization) όπως αναφερθήκαμε παραπάνω. Επιπλέον τα virtual machines καταναλώνουν πολύ μεγαλύτερο χώρο στο δίσκο και στη μνήμη και είναι πολύ δυσκολότερο να συντηρηθούν.

Προβλήματα που υπάρχουν με τα virtual machines όπως η καθυστέρηση και η κατάχρηση της μνήμης μπορούν να επιλυθούν

με την χρήση των containers. Το μόνο που απαιτούν τα containers είναι η εφαρμογή μαζί με τις απαιτήσεις της (dependencies) ώστε να λειτουργήσουν ενώ διαμοιράζονται τον πυρήνα μεταξύ τους. Επειδή το λειτουργικό σύστημα τρέχει ήδη (στον host) το να ξεκινήσει κάποιος ένα container είναι κατά πολύ γρηγορότερο σε σχέση με ένα virtual machine. Ωστόσο ο διαμοιρασμός του πυρήνα (kernel) δεν είναι πάντα ιδανική περίπτωση, για παράδειγμα, εάν κάποιος θέλει να τρέξει μια εφαρμογή Windows σε container του Linux τότε αυτό δεν μπορεί να γίνει.

3. Docker

3.1 Ο σκοπός ανάπτυξης του Docker

Ο κύριος στόχος του Docker [18] είναι οποιαδήποτε εφαρμογή να μπορέσει να αναπτυχθεί και να τρέξει σε οποιοδήποτε εξυπηρετητή οπουδήποτε. Το Docker στον πυρήνα του είναι μια πλατφόρμα ανοιχτού λογισμικού που επιτρέπει στις εφαρμογές να αναπτυχθούν μέσα σε containers λογισμικού. Αυτή η startup επιχείρηση που ξεκίνησε από την Silicon Valey τράβηξε την προσοχή πολλών μεγάλων εταιριών πληροφορικής. Η Amazon, Google, Microsoft και Red Hat υποστήριξαν το Docker στις πλατφόρμες τους και συνεχίζουν να συνεισφέρουν στο project του Docker.

Το Docker όμως είναι περισσότερο από μια απλή βιβλιοθήκη εικονικοποίησης, συνοψίζει τις διαφορές μεταξύ των λειτουργικών συστημάτων και δημιουργεί ένα προτυποποιημένο περιβάλλον για την ανάπτυξη εφαρμογών. Ένας μηχανικός λογισμικού μπορεί να δημιουργήσει μια προτυποποιημένη εφαρμογή η οποία γίνεται φορητή και μπορεί να τρέξει οπουδήποτε είναι εγκατεστημένο το Docker Engine. Αυτό γλυτώνει πολύ χρόνο από τον δημιουργό της εφαρμογής καθώς δεν χρειάζεται να υποστηρίξει πολλαπλές πλατφόρμες και διαφορετικά λειτουργικά συστήματα. Οι διαχειριστές συστημάτων χρειάζεται να αφιερώσουν λιγότερο χρόνο στην διαμόρφωση της εφαρμογής καθώς είναι πακεταρισμένη με όλες τις εξαρτήσεις και τις βιβλιοθήκες που μπορεί να χρειάζεται για την λειτουργία της. Το γεγονός ότι κάθε εφαρμογή τρέχει απομονωμένα στο δικό της container λύνει πολλά κοινά προβλήματα όπως η ανάγκη για κατάργηση ή αντικατάσταση μιας εφαρμογής με μια άλλη καθώς και την περίπτωση όπου δύο εφαρμογές χρειάζονται διαφορετική έκδοση της ίδιας εξάρτησης ή βιβλιοθήκης λογισμικού (software dependency).

Το Docker αρχικά περιοριζόταν μόνο σε εφαρμογές για περιβάλλοντα Linux αλλά μετά την συνεργασία του με την Microsoft και το Anniversary Update των Windows, τον Σεπτέμβριο του 2016, δόθηκε η δυνατότητα για την εκτέλεση containers του docker natively σε Windows 10 ή Windows Server 2016 VM.

3.2 Ιστορικά στοιχεία

Η αρχική έκδοση του project της εταιρίας dotCloud κυκλοφόρησε ως ελεύθερο λογισμικό τον Μάρτιο του 2013. Δύο μήνες αργότερα, κυκλοφόρησε το Docker. Στο δεύτερο μισό του 2013 οι Google, Yandex και Baidu (οι δημοφιλέστερες Ρωσικές και Κινέζικες μηχανές αναζήτησής αντίστοιχα) είχαν ήδη ενσωματώσει το Docker στις cloud υποδομές τους.

Το Docker εισήλθε στο 2014 ολοκληρώνοντας την έγκριση ενός κονδυλίου 15 εκατομμυρίων δολαρίων το οποίο του έδωσε την δυνατότητα να επενδύσει σε μεγάλο βαθμό στο πρότζεκτ του ανοικτού λογισμικού, στην υποστήριξη επιχειρήσεων καθώς και στην επέκταση της κοινοτικής πλατφόρμας. Τον Απρίλιο το LXC καταργήθηκε ως το προεπιλεγμένο περιβάλλον εκτέλεσης και αντικαταστάθηκε με το libcontainer του Docker. Τον επόμενο μήνα, η έκδοση 14.04 του Ubuntu ήταν η πρώτη επιχειρησιακή έκδοση του Linux που κυκλοφόρησε μαζί με το πακέτο του Docker, δίνοντας την δυνατότητα σε εκατομμύρια διακομιστές Ubuntu να χρησιμοποιήσουν τα Docker containers μόνο με 3 εντολές. Η έκδοση 1.0 κυκλοφόρησε τον Ιούνιο του 2014 στην πρώτη ομιλία με θέμα το Docker που ονομάστηκε DockerCon.

Τον Σεπτέμβριο ανακοινώθηκε ακόμα ένα μεγάλο κονδύλι αξίας 40 εκατομμυρίων, με την αξία του Docker να φτάνει τα 400 εκατομμύρια δολάρια. Ένα μήνα αργότερα, το Docker και η Microsoft ανακοίνωσαν την συνεργασία τους με σκοπό την δημιουργία μιας μηχανής Docker για τα Windows και ένα multi-Docker container μοντέλο, συμπεριλαμβανομένης της υποστήριξης εφαρμογών που αποτελούνται από Linux και Windows Docker containers. Τον Δεκέμβριο διεξήχθη η πρώτη επίσημη ομιλία στην Ευρώπη στο Amsterdam, ανακοινώνοντας διάφορα καινούργια projects που έχουν σχέση με το Docker καθώς και το Docker Hub Enterprise. Το Docker τελείωσε την χρονιά του 2014 με την κυκλοφορία της έκδοσης 1.4, που ήταν το 24ο πιο δημοφιλές project στην πλατφόρμα του Github.

Στο πρώτο μισό του 2015 κυκλοφόρησαν πολλαπλές καινούργιες εκδόσεις του Docker με διάφορες προσθήκες όπως η υποστήριξη IPv6, η πολυαναμενόμενη client έκδοση για Windows, τα εργαλεία ενορχήστρωσης Docker Compose, Docker Swarm, Docker Machine καθώς και το runC ένα εργαλείο γραμμής εντολών για την δημιουργία και εκτέλεση containers με βάση τις προδιαγραφές OCP. Τους επόμενους μήνες ανακοινώθηκε το Docker Content Trust με την έκδοση 1.8 όπου με την βοήθεια του Notary, ενός πρότζεκτ βασισμένου στο Update Framework, προσφέρει επιπλέον ασφάλεια στο Docker καθώς δίνει την δυνατότητα στους χρήστες να επιβεβαιώνουν την ταυτότητα του εκδότη ενός Docker Image, χρησιμοποιώντας το δημόσιο του κλειδί κρυπτογραφίας για να είναι σίγουροι ότι δεν έχει επέμβει κάποιος κακόβουλος χρήστης. Επίσης, ανακοινώθηκε και το Docker Toolbox δηλαδή τα όλα εργαλεία που χρειάζονται για την εγκατάσταση και λειτουργία του Docker σε ένα πακέτο.

Το 2016 υπήρχαν ακόμα αρκετές ανακοινώσεις και νέες προσθήκες στο Docker με σημαντικότερη από αυτές την διάθεση του καινούργιου

Docker client για Windows και Mac με βελτιώσεις στην ταχύτητα και την λειτουργία του καθώς τρέχει πάνω σε μια εικονική μηχανή xhyve, όσον αφορά τα Mac ή Hyper-V στα Windows και δεν βασίζεται πλέον στο VirtualBox. Επιπλέον περιέχει όλα τα εργαλεία από την έκδοση Docker Toolbox και προσφέρει εύκολη πρόσβαση στα containers του τοπικού δικτύου.

3.3 Docker Daemon

Εσωτερικά το Docker χρησιμοποιεί ένα μοντέλο πελάτη-εξυπηρετητή (client-server architecture), όπου ο εξυπηρετητής (server) είναι ένα daemon που μπορεί να εκτελείται σε εντελώς διαφορετικό σύστημα από τον πελάτη (client). Το daemon μπορεί να ξεκινήσει είτε με μια εντολή του docker χρησιμοποιώντας το -d flag, είτε σαν υπηρεσία του συστήματος (service) με τις εντολές systemctl start docker (για kernels που λειτουργούν με το systemd) ή service docker start για εκδόσεις του Linux kernel που δεν χρησιμοποιούν το systemd. Για την εκκίνηση του docker σε κατάσταση daemon χρειάζεται ένας εξουσιοδοτημένος λογαριασμός με δικαιώματα διαχειριστή (root) αν και έχουν γίνει σημαντικές προσπάθειες για να ξεπεραστεί αυτό το εμπόδιο έτσι ώστε μέχρι και απλοί χρήστες να μπορούν να τρέξουν containers.

Ο εξυπηρετητής εξ' ορισμού επικοινωνεί μόνο με ένα Unix socket, πράγμα που τον καθιστά απρόσιτο μέσα από το δίκτυο και αυτό γιατί όποιος έχει πρόσβαση στον daemon μπορεί να αποκτήσει τον έλεγχο ολόκληρου του host. Μια απλή επίθεση θα ήταν η εκκίνηση ενός container με προσαρτημένο τον root "/" κατάλογο αρχείων του host, έτσι ο κακόβουλος χρήστης θα μπορούσε πολύ εύκολα να αποκτήσει πρόσβαση όχι μόνο στο container αλλά και στο σύστημα αρχείων που υπάρχει στον host υπολογιστή καθως και την δυνατότητα να ξαναγράψει οποιοδήποτε απο τα αρχεία του host. Συνεπώς είναι άκρως σημαντικό όταν το Docker εκτελείται σε δημόσια IP, αυτό να γίνεται πάντα με την χρήση του πρωτοκόλλου κρυπτογράφησης TLS, όπου πιστοποιείται η γνησιότητα κάθε πελάτη από μια έμπιστη αρχή πιστοποιητικών.

3.4 Docker Libcontainer

Στην έκδοση 0.9 του Docker παρουσιάστηκε ένα καινούργιο API οδηγών εκτέλεσης, που έδωσε την δυνατότητα χρήσης διάφορων εργαλείων απομόνωσης, και εκτός από τον οδηγό LXC ο οποίος υποστηρίζεται σαν αυτόνομος οδηγός παρουσιάστηκε ο Libcontainer ο οποίος και έγινε ο προεπιλεγμένος οδηγός εκτέλεσης του Docker. Είναι γραμμένος αποκλειστικά στην γλώσσα Go και χειρίζεται την διαχείριση των containers, χρησιμοποιώντας τις δυνατότητες του πυρήνα που αναφέραμε και προηγουμένως όπως τα namespaces και τα control groups. Ο Libcontainer επίσης εξάλειψε την ανάγκη χρήσης πρόσθετων εργαλείων όπως το Boot2docker [19] με την έκδοση του Docker για Windows.

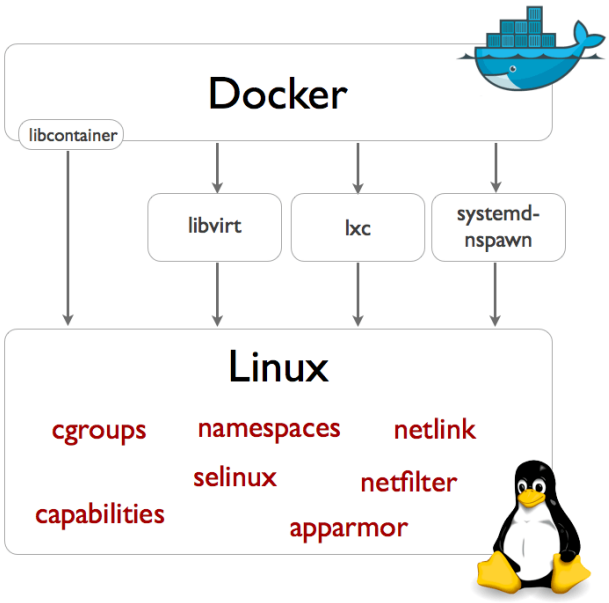

Figure 3.1: Οδηγοί εκτέλεσης του Docker.
[20]

3.5 Docker Layered Filesystem

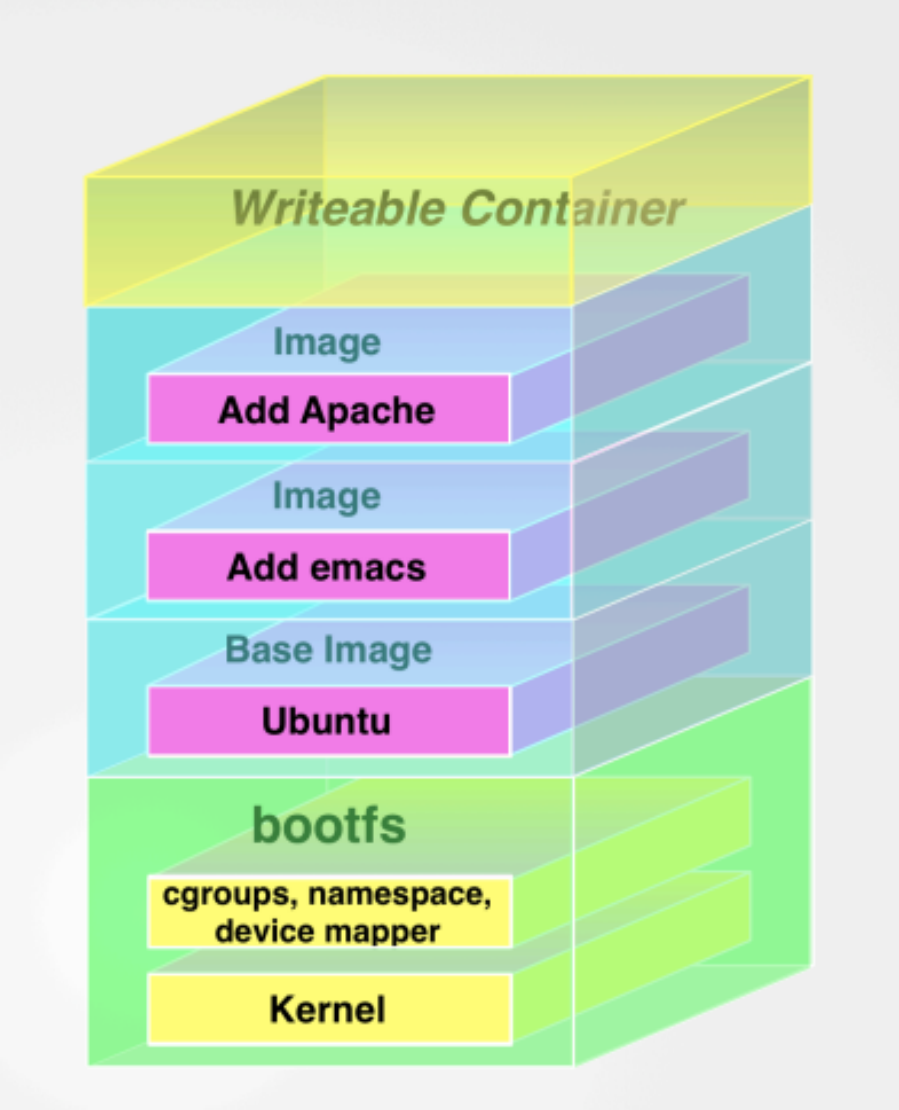

Figure 3.2: Το πολυεπίπεδο σύστημα αρχείων του Docker. [21]

Το Docker αποτελείται από δύο κύρια στοιχεία: τα Images και τα containers. Ένα Docker Image περιλαμβάνει ένα σύστημα αρχείων καθώς και τις παραμέτρους που χρειάζεται αυτό το σύστημα κατά την εκτέλεση του. Το Image δεν διαθέτει κατάσταση και δεν δέχεται αλλαγές. Σε αντίθεση με το Image, το container αποτελεί μια εκτελέσιμη εκδοχή του Image το οποίο έχει κατάσταση και μπορεί να κρατήσει δεδομένα αποθηκευμένα. Κατά την εκτέλεση ενός container, το Docker χρησιμοποιεί έναν μηχανισμό που ονομάζεται Union File System ο οποίος είναι θεμέλιος λίθος για την δημιουργία του container καθώς και ένας τρόπος να συνδυαστούν πολλά διαφορετικά συστήματα αρχείων ή φάκελοι σε ένα ενιαίο που φαίνεται να περιέχει συνδυαστικά τα περιεχόμενα τους. Έτσι, τα συστήματα αρχείων δεν βρίσκονται σε διαφορετικά σημεία, αλλά το ένα "πάνω" στο άλλο, δημιουργώντας την

αίσθηση μιας πολυεπίπεδης ιεραρχικής δομής.

Οι εφαρμογές που εκτελούνται μέσω του Docker προσδιορίζουν σχεδόν πάντα έναν γονέα ο οποίος είναι ένα Docker Image. Για παράδειγμα εάν έχουμε μια web εφαρμογή αυτή συνήθως εξαρτάται από έναν συγκεκριμένο web server (πχ. apache, nginx) ο οποίος με την σειρά του μπορεί να εξαρτάται από ένα συγκεκριμένο λειτουργικό σύστημα. Αυτό με την σειρά του σημαίνει πως κάθε Docker Image μπορεί να περιέχει πολλά επίπεδα συστημάτων αρχείων τα οποία βρίσκονται το ένα πάνω στο άλλο με ιεραρχική δομή και με δικαιώματα μόνο για διάβασμα. Έτσι, όταν η εφαρμογή ξεκινήσει σαν container, πλέον προστίθεται ένα επιπλέον επίπεδο στην κορυφή το οποίο έχει επίσης δικαιώματα εγγραφής. Όταν ένα αρχείο από τα κατώτερα μόνο-για-διάβασμα (read-only) επίπεδα χρειαστεί να υποστεί μια αλλαγή τότε αυτό αντιγράφεται στο επίπεδο στο οποίο επιτρέπεται η εγγραφή και η αλλαγή πραγματοποιείται εκεί. Είναι σημαντικό να κατανοήσουμε ότι αυτές οι αλλαγές εξακολουθούν να υφίστανται ακόμη και μετά την παύση της εκτέλεσης του container, οπότε και κατά την επόμενη εκτέλεσή του θα είναι είναι ενεργές.

Τα Data Volumes [22] αποτελούν έναν μηχανισμό διαχείρισης και διαμοιρασμού των δεδομένων ανάμεσα σε containers οποίος όμως δεν χρησιμοποιεί το Union File System που αναφέρθηκε. Τα Data Volumes είναι ειδικοί φάκελοι που αρχικοποιούνται κατά την δημιουργία ενός container και μπορούν να διαμοιραστούν και να επαναχρησιμοποιηθούν ανάμεσα σε διαφορετικά containers. Οι αλλάγες στα Data Volumes γίνονται άμεσα χωρίς την διαμεσολάβηση επιπλέον επιπεδού για λόγους δικαιωμάτων εγγραφής, όπως προαναφέρθηκε στα Union File Systems, οι αλλαγές ωστόσο στα Data Volumes δεν συμπεριλαμβάνονται κατά την ενημέρωση ενός Docker Image. Αξίζει να σημειωθεί ότι ο μηχανισμός αυτός είναι πολύ χρήσιμος για την μεταφορά δεδομένων, την δημιουργία αντιγράφων ασφαλείας αλλά και την επαναφορά τους.

3.6 Ασφάλεια στο Docker

Ένα από τα πιο σημαντικά ζητήματα που καλούνται να επιλύσουν οι μηχανικοί του Docker είναι η ασφάλεια του. Επί του παρόντος το Docker για να τρέξει σε ένα σύστημα και να είναι πλήρως λειτουργικό έχει κάποιες διαδικασίες οι οποίες χρειάζονται root δικαιώματα για να εκτελεστούν. Κάτι τέτοιο σημαίνει δυσμενείς επιπτώσεις για τον host εάν βρεθεί κάποιο είδος ευπάθειας στο Docker και για το λόγο αυτό γίνονται προσπάθειες να βρεθούν τρόποι ώστε να τρέξει ο daemon του Docker χωρίς root δικαιώματα. Έχει γίνει πρόοδος πάνω σε αυτόν τον τομέα και το Docker πλέον υποστηρίζει αρχιτεκτονική με 2 daemons ώστε να τρέχει τις περισσότερες λειτουργίες χωρίς root δικαιώματα και μόνο αυτές οι οποίες χρειάζονται εξ' ορισμού root πρόσβαση να τρέχουν με αυτό το επίπεδο πρόσβασης.

Το Docker φροντίζει επίσης και για την ασφάλεια των Docker Images. Από τον Δεκέμβριο του 2014 και μετά, το Docker προώθησε

ένα νέο χαρακτηριστικό του που ονομάζεται “image signing” και αφορά ψηφιακές υπογραφές που αποσκοπούν στην επαλήθευση της ταυτότητας και της ακεραιότητας του κάθε Docker Image. Ωστόσο τα πράγματα δεν πήγαν τόσο καλά αφού βρέθηκε πως ο τρόπος επαλήθευσης ενός Docker Image βασίστηκε αποκλειστικά και μόνο στην ύπαρξη ενός ψηφιακά πιστοποιημένου αρχείου manifest (είναι ένα αρχείο που περιέχει τα μεταδεδομένα για μια ομάδα αρχείων που συνοδεύει) και δεν γίνεται καθόλου έλεγχος του checksum του αρχείου εικόνας (Docker Image), κάτι που σημαίνει ότι ένας κακόβουλος χρήστης θα μπορούσε χρησιμοποιώντας ένα έγκυρο αρχείο manifest και ένα κακόβουλο αρχείο Docker Image να προσπεράσει αυτό το μηχανισμό ελέγχου.

Μια ακόμα πιθανή απειλή στην εικονικοποίηση βασισμένη σε containers λογισμικού αποτελούν τα Linux user namespaces αφού αποτελούν ένα σχετικά νέο χαρακτηριστικό του πυρήνα και δεν έχουν ακόμη δοκιμαστεί εκτενώς σε περιβάλλοντα παραγωγής. Ένα από τα προβλήματα ασφαλείας που βρέθηκαν αφορούσε την δυνατότητα μιας διεργασίας να αποκτήσει πρόσβαση σε ένα σημείο του σύστηματος αρχείων στο οποίο ακόμη και ο τρέχων χρήστης δεν είχε δικαίωμα πρόσβασης, και αυτό χρησιμοποιώντας το user namespace και αφαιρώντας ομάδα χρηστών από αυτό.

Εκτός από τα προβλήματα που αφορούν τον πυρήνα του Linux καθώς και το Docker σαν λογισμικό, υπάρχουν πάντα και τα προβλήματα που δημιουργούνται με βάση τον ανθρώπινο παράγοντα, π.χ. για παράδειγμα οι ρυθμίσεις που αφορούν την δημιουργία των containers μπορεί να αφήνουν κενά ασφαλείας εκμεταλλεύσιμα όπως για παράδειγμα μια ανοιχτή πόρτα σε κάποιο container που αφήνει εκτεθειμένη μια υπηρεσία.

3.7 Οι επιδόσεις του Docker

Σύμφωνα με συγκριτική έρευνα της IBM ανάμεσα σε virtual machines (KVM) και το Docker που πραγματοποιήθηκε το 2014 [23], το Docker φαίνεται να υπερισχύει του KVM σε όλα τα χαρακτηριστικά που δοκιμάστηκαν, όπως ταχύτητα προσπέλασης μνήμης, εγγραφή και ανάγνωση δεδομένων από το δίσκο καθώς και στα χαρακτηριστικά που αφορούν το δίκτυο. Οι δοκιμές έγιναν σε δύο δημοφιλή πακέτα λογισμικού, την MySQL και το Redis Cache. Έχοντας υπόψιν τις διαφορές των δύο τεχνολογιών εικονικοποίησης αποφασίστηκε ότι “και οι δυο τεχνολογίες (virtual machines και containers) είναι αρκετά ώριμες και έχουν επωφεληθεί από τις διαρκείς αναβαθμίσεις και βελτιστοποιήσεις των συστημάτων πληροφορικής τα τελευταία χρόνια, ωστόσο σε γενικές γραμμές το Docker, ισοδυναμεί ή ξεπερνά τις επιδόσεις του KVM σε όλες τις δοκιμασίες που πραγματοποιήθηκαν. Τα αποτελέσματα μας δείχνουν ότι και το KVM και το Docker εμφανίζουν αμελητέο επιπλέον βάρος στις επιδόσεις του επεξεργαστή και της μνήμης (εκτός από ακραίες περιπτώσεις).”

Οι δοκιμές που έγιναν απέδειξαν ότι οι επιδόσεις του Docker όσο αναφορά το δικτυακό κομμάτι εξαρτώνται κατά ένα μεγάλο βαθμό από τον τρόπο που χρησιμοποιεί το δίκτυο του host υπολογιστή και εάν αυτό γίνεται απευθείας ή μέσω μιας "γέφυρας NAT" (NAT bridge). Καθυστερήσεις καθώς και μικρότερες ταχύτητες εγγραφής ενδέχεται να υπάρχουν και όταν τα δεδομένα αποθηκεύονται σε ένα κοινό volume (shared volume) αντί του union filesystem που υποστηρίζει το Docker.

3.8 Η επίδραση του Docker σε αναπτυσσόμενα project

Το Docker έχει αποτελέσει πηγή έμπνευσης για πολλά καινούργια πρότζεκτ, που έχουν εκμεταλλευτεί τις λειτουργίες του. Ενώ πολλά από αυτά τα πρότζεκτ έχουν ολοκληρωθεί, πολλά ακόμα αναπτύσσονται καθώς όλοι θέλουν να καλύψουν ένα κενό στην αγορά όσο το δυνατόν γρηγορότερα. Απομένει να φανεί ποια από αυτά τα πρότζεκτ θα αναδυθούν σαν νικητές μέσα από αυτό το άκρως ανταγωνιστικό περιβάλλον και εμείς στην παρούσα εργασία θα αναφέρουμε τα σημαντικότερα από αυτά.

3.8.1 Kubernetes

Η Google είναι αδιαμφισβήτητα από τους μεγαλύτερους χειριστές κέντρων δεδομένων, και έχει παραδεχθεί ότι κάθε μία από τις υπηρεσίες της εκτελείται μέσα σε ένα Linux container. Ενώ ακόμα δεν έχουν αποκαλύψει τον βασικό εσωτερικό task scheduler "Omega", κυκλοφόρησαν έναν δεύτερο container manager: το Kubernetes [24], ως ένα πρότζεκτ ελεύθερου λογισμικού. Τα containers μπορούν να τρέξουν είτε μέσα στο Docker είτε στο Rkt (Rocket) [25], ένα container σύστημα παρόμοιο του Docker το οποίο θα καλύψουμε σε επόμενο κεφάλαιο.

Το Kubernetes παρουσιάζει την ιδέα των Pods - ομάδες από containers τα οποία είναι στενά συνδεδεμένα μεταξύ τους και αντιμετωπίζονται ως η μικρότερη μονάδα προς ανάπτυξη. Μια συνηθισμένη περίπτωση χρήσης είναι ένα κύριο container που εκτελεί μία εφαρμογή διακομιστή σελίδων χρησιμοποιώντας διάφορα άλλα containers με τις βοηθητικές του υπηρεσίες. Είναι λογικό αυτά τα containers να ξεκινούν και να σταματούν την ίδια στιγμή και να εκτελούνται στον ίδιο host. Τα Pods έχουν συνήθως επισυναπτόμενες ετικέτες (key,value pairs) που τους επιτρέπουν την αναζήτηση των cluster nodes που εκτελούν μια συγκεκριμένη ομάδα από pods.

Το επόμενο abstraction είναι ένα service - ομάδων με ports καθορισμένες από ετικέτες που εμφανίζουν το ίδιο port και εκτελούν την ίδια εφαρμογή. Ένα pod μπορεί να σταματήσει να εκτελείται σε ένα node και να αναπαραχθεί σε ένα άλλο, έτσι με την χρήση ενός service, τα υπόλοιπα pods δεν χρειάζεται να παρακολουθούν ποιο pod τρέχει σε ποιο Node, αντί αυτού μπορούν να χρησιμοποιήσουν μια εικονική IP του service.

Για να υποστηρίξει την κλιμάκωση, το Kubernetes παρουσίασε τα replication controllers. Ένα replication controller χρησιμοποιεί ένα πρότυπο, σύμφωνα με το οποίο τα pods δημιουργούνται και επιτρέπει τον ορισμό του αριθμού των pod κλώνων-αντιγράφων (replicas) προς εκτέλεση. Αυτό μπορεί να χρησιμοποιηθεί με ευκολία για να αυξηθεί ή να μειωθεί ο αριθμός των υποστάσεων (instances) μιας εκτελούμενης εφαρμογής. Μια άλλη περίπτωση χρήσης θα ήταν η ενημέρωση μιας εφαρμογής, χρησιμοποιώντας ένα replication controller για την παλαιά έκδοση και έναν για την καινούρια, χωρίς να χρειαστεί να σταματήσει η εφαρμογή (service disruption) παρέχοντας έτσι συνεχή ομαλή λειτουργία (uptime).

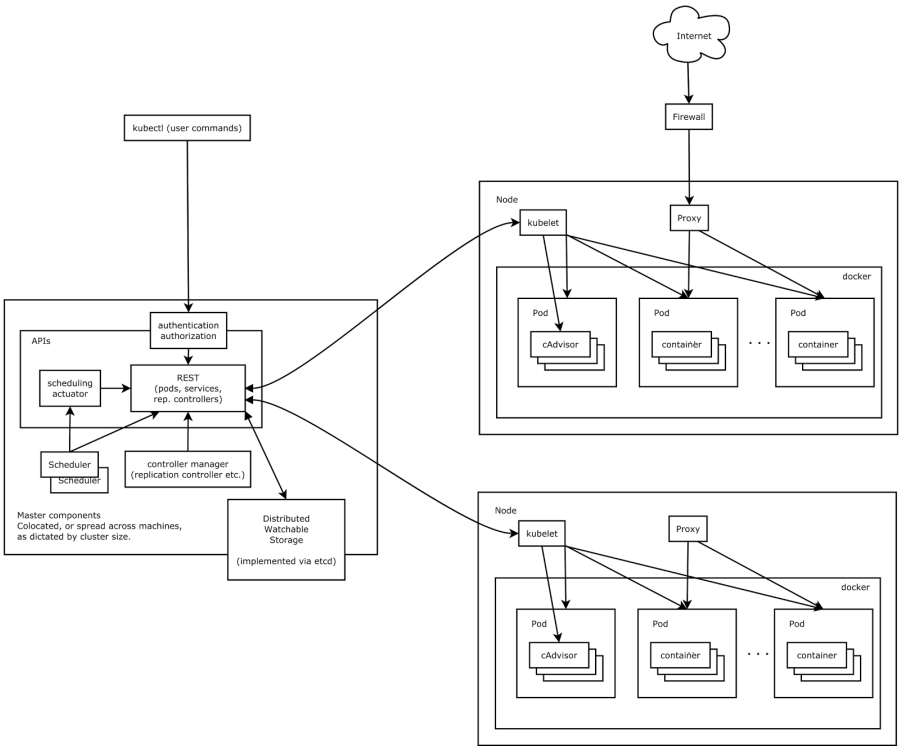

Figure 3.3: Η αρχιτεκτονική του Kubernetes.
[26]

3.8.2 Mesos

Το Apache Mesos [27] που κυκλοφόρησε το 2013, αποκρύπτει τους πόρους που είναι διαθέσιμοι σε ένα cluster, και τους παρουσιάζει σαν ένα μοναδικό τεράστιο υπολογιστικό σύστημα. Η αρχιτεκτονική του προσδιορίζει την έννοια ενός σκελετού (framework) - μια εφαρμογή γραμμένη για το Mesos, η οποία περιέχει έναν χρονοδρομολογητή (scheduler) που εκτελεί τις εργασίες της εφαρμογής. Το Kubernetes μπορεί να χρησιμοποιηθεί ως ένα τέτοιο framework έτσι ώστε μια πιθανή διεργασία να εκκινείται από ένα pod.

Ο κύκλος διεπαφής μεταξύ του cluster, του Mesos και ενός framework μπορεί να περιγραφεί ως εξής:

- Ένα slave daemon του Mesos εκτελείται σε κάθε node στο cluster.
- Ένα master daemon του Mesos χρησιμοποιείται για να ελέγξει τα slaves.
- Τα slaves παρουσιάζουν τους διαθέσιμους πόρους του node στον master.
- Το master node επιλέγει ένα framework και του προσφέρει ένα κομμάτι από τους διαθέσιμους πόρους μαζί με τα αναγνωριστικά των κόμβων (nodes) αλλά και τους κανόνες που αφορούν την χρήση των πόρων από αυτά.
- Ο scheduler του framework αποφασίζει ποιες διεργασίες θα εκτελεστούν σε ποια nodes και καθορίζει τον διαμοιρασμό των πόρων.
- Το master node καλεί τον εκτελεστή διεργασιών του framework στα επιλεγμένα nodes, με τους πόρους που ζητήθηκαν.

Το Mesos απέκτησε γρήγορα κοινό απο εταιρίες που χρησιμοποιούν τεράστια clusters όπως οι: Paypal, Vimeo, Twitter, Airbnb, Netflix. Ακόμα και η Apple αποκάλυψε ότι η Siri, ο προσωπικός βοηθός του iOS χρησιμοποιεί το Mesos στο backend της.

3.8.3 Mesosphere DC/OS

Το Mesosphere DC/OS [28] είναι ένα λειτουργικό σύστημα βασισμένο στο Mesos το οποίο απευθύνεται σε μεγάλες επιχειρήσεις που έχουν μεγάλο αριθμό χρηστών, επεξεργάζονται κλιμακωτά δεδομένα σε πραγματικό χρόνο και χρειάζονται να παραδίδουν υπηρεσίες σε σύντομο χρονικό διάστημα.

Αυτό που προσφέρει το Mesosphere DC/OS είναι μία ενιαία πλατφόρμα για την εκτέλεση κατανεμημένων συστημάτων με containers και υπηρεσίες Big Data με εφαρμογές είτε σε τοπικό επίπεδο είτε στο cloud απλουστεύοντας με αυτό τον τρόπο τις διαδικασίες και προσφέροντας ελαστικότητα στις απαιτήσεις σε πόρους συστήματος εξοικονομώντας με αυτόν τον τρόπο χρήματα και χρόνο.

3.8.4 CoreOS

Το CoreOS [30] είναι ένα λειτουργικό σύστημα ανοικτού κώδικα το οποίο βασίζεται στο ChromeOS της Google. Κυκλοφόρησε το φθινόπωρο του 2013 και άρχισε να προωθεί την ιδέα ενός λειτουργικού συστήματος βασισμένο σε containers. Αυτό σημαίνει ότι το CoreOS δεν περιέχει package manager, αλλά προσφέρει πρόσβαση σε εφαρμογές μέσω containers και χρησιμοποιεί το Docker σαν container manager.

Το CoreOS προσπαθεί να επικεντρωθεί σε cloud υποδομές με τις δύο κύριες εφαρμογές του τις Etcd και Fleet. Το Etcd είναι ένα daemon για την διαχείριση cloud υποδομών και προσφέρει ένα API για την

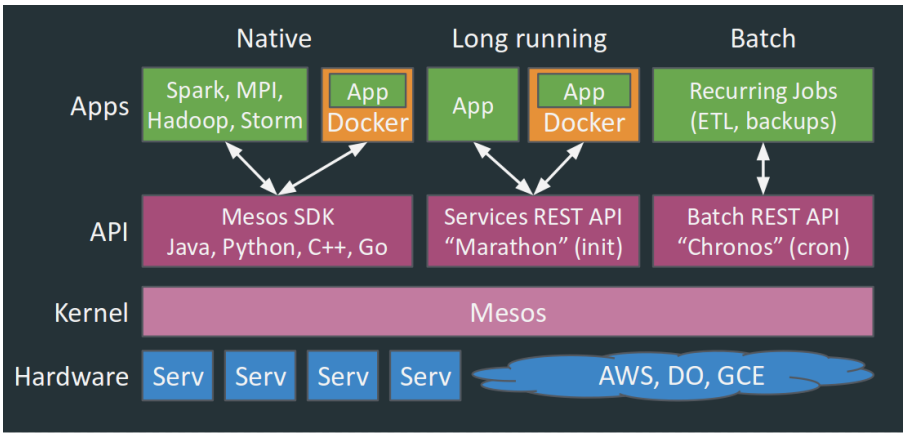

Figure 3.4: Η Αρχιτεκτονική του Mesosphere.
[29]

διάδοση των αλλαγών στις ρυθμίσεις μεταξύ ενός ολόκληρου cluster από etcd instances. Το Fleet είναι ένα επιπέδου cluster systemd daemon που επιτρέπει την ανάπτυξη container σε όλες τις μηχανές μέσα στο cluster ή σε ένα συγκεκριμένο μηχάνημα με υποστήριξη failover.

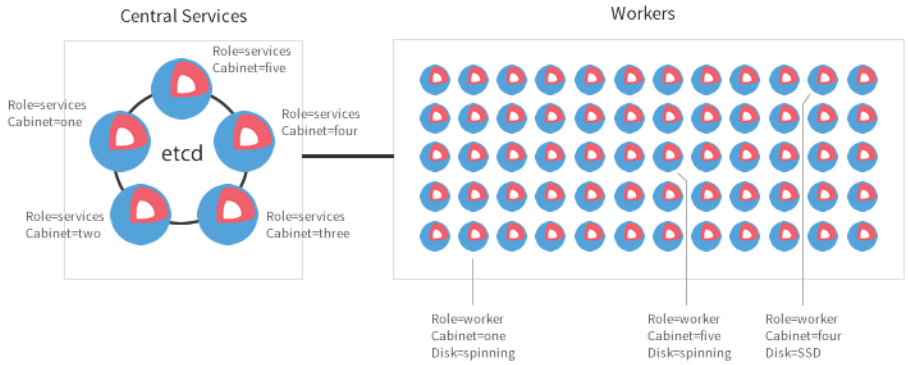

Figure 3.5: Ένα cluster από CoreOS hosts με containers.
[31]

3.8.5 Snappy Ubuntu Core και LXD

Το Snappy - Snaps [32] της Canonical είναι ένα ακόμα λειτουργικό σύστημα, με στόχο την εκτέλεση containerized εφαρμογών. Αυτό το project παρέχει το ελάχιστο δυνατό σε μέγεθος λειτουργικό σύστημα που μπορεί να υποστηρίζει εφαρμογές βασισμένες σε containers. Το Snappy υποστηρίζει πολλαπλούς παρόχους containers συμπεριλαμβανομένου και του Docker οι οποίοι αποκαλούνται frameworks.

Επίσης, περιλαμβάνει ένα atomic update σύστημα το οποίο εφαρμόζεται για τις εγκατεστημένες εφαρμογές αλλά και για το ίδιο το Snappy. Οι ενημερώσεις γίνονται με την μορφή συναλλαγών, έτσι ώστε να είναι πάντα δυνατό ένα rollback στην περίπτωση μίας αποτυχημένης

ή μη επιθυμητής ενημέρωσης. Το Snappy εσωτερικά διατηρεί όλες τις βασικές εκδόσεις όλων των packages και ενημερώνει μόνο τις διάφορες με την προηγούμενη έκδοση (incremental update).

Η Ubuntu έχει αναπτύξει επίσης το LXD (Linux Contained Daemon) [33], το οποίο βασίζεται στο LXC και ο στόχος του είναι το hosting containers μηχανών που λειτουργούν σαν εικονικές μηχανές Linux σε αντίθεση με το Docker που κάνει host containers εφαρμογών. Το LXD και το Docker μπορούν να δουλέψουν μαζί, καθώς το LXD προσφέρει στις εφαρμογές του Docker χαρακτηριστικά εικονικών μηχανών και την δυνατότητα διαχείρισης και εκτέλεσης εφαρμογών του Docker μέσα στα machine containers του LXD.

3.8.6 Project Atomic

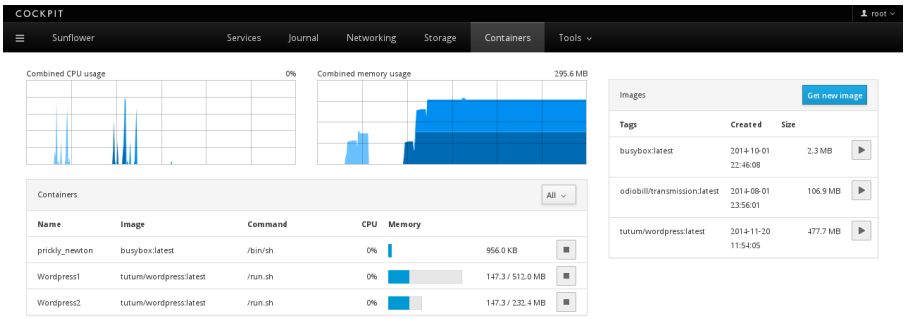

Figure 3.6: Το web-based γραφικό περιβάλλον (GUI) του Project Cockpit.

Το Project Atomic [34] είναι ένα σετ εργαλείων που προσφέρουν λύσεις για το deployment containerized εφαρμογών. Το κύριο αποτέλεσμα του Project Atomic είναι τα Project Atomic hosts, τα οποία είναι ελαφριά σε απαιτήσεις λειτουργικά συστήματα βασισμένα σε Red hat Enterprise Linux, Fedora ή CentOS. Τα σημαντικότερα εργαλεία που περιέχονται σε αυτά είναι τα Docker, Kubernetes, SELinux, Rpm-ostree και Project Cockpit.

Ένα από τα εργαλεία του Project Atomic: το Project Cockpit αποτελεί έναν Linux server manager απομακρυσμένης σύνδεσης με το δικό του web based GUI. Το κύριο πλεονέκτημα του είναι η παροχή απλών και ξεκάθαρων εικονικών στοιχείων για την εύκολη διεπαφή καινούριων System Administrators με τα στατιστικά της κατάστασης του server.

3.8.7 Rkt

Τον Δεκέμβριο του 2014, η ομάδα του CoreOS εξέφρασε τον προβληματισμό της για την κατεύθυνση που είχε πάρει το Docker project και κυκλοφόρησε ένα blog post το οποίο αμέσως τράβηξε την προσοχή του κόσμου. Εξέφρασαν την διαφωνία τους με το πόσο πλατύς είχε γίνει ο στόχος του Docker:

”Όταν το Docker παρουσιάστηκε για πρώτη φορά στις αρχές του 2013, η ιδέα ενός προτύπου container ήταν εκπληκτική και άμεσα ελκυστική: ένα απλό εργαλείο, μία συναρμολογούμενη μονάδα που μπορεί να χρησιμοποιηθεί σε διάφορα συστήματα. Το Docker repository περιείχε ένα μανιφέστο για το πως πρέπει καθορίζεται ένα container. Πιστέψαμε ότι το Docker θα γινόταν μία απλή μονάδα που θα μπορούσαμε όλοι να συμφωνήσουμε μαζί του. Το μανιφέστο που καθόριζε το container αφαιρέθηκε. Πρέπει να σταματήσουμε να μιλάμε για Docker containers και να αρχίσουμε να μιλάμε για την πλατφόρμα Docker. Δεν αναπτύσσεται σε αυτό το απλό συναρμολογούμενο κομμάτι που είχαμε οραματιστεί.” [35]

Αυτή η κριτική πιθανώς ήταν στοχευμένη στο τρίο των προτζεκτ Docker Machine, Docker Swarm και Docker Compose τα οποία κυκλοφόρησαν το 2015. Επίσης, υποστήριξαν ότι το αρχιτεκτονικό μοντέλο του Docker, όπου τα πάντα εκτελούνται μέσα από ένα κεντρικό daemon είναι ”κατά βάση προβληματικό” από την πλευρά της ασφάλειας. Αυτά τα επιχειρήματα απευθείας δημιούργησαν συζητήσεις αν το Docker προσπαθεί να επιτύχει υπερβολικά πολλούς στόχους και ότι έτσι πάει να γίνει μια μεγάλη μονολιθική πλατφόρμα, χωρίς να προσφέρει ουσιαστική τμηματοποίηση, ή ότι έχει επεκταθεί στις επαγγελματικές περιοχές άλλων επιχειρήσεων και start-ups και αυτό το blog post του CoreOS είναι ένα παράδειγμα επιχειρηματικής άμυνας.

Επιπλέον το post της ομάδας του CoreOS περιείχε και την ανακοίνωση του Rocket (Rkt) - ενός καινούργιου container runtime και άμεσου ανταγωνιστή του Docker. Το όραμα του CoreOS είναι η συγκέντρωση του γύρω από μία προδιαγραφή για containers και το “App Container executor”, που περιλάμβαναν και ενθάρρυναν τους developers να υποβάλουν τα σχόλιά τους πάνω σε αυτό. Αργότερα μετονομάστηκε σε Rkt - “Rock it”. Το Rkt ακόμα συνεχίζεται να αναπτύσσεται σε μεγάλο βαθμό, αλλά ήδη επιτρέπει την εκτέλεση native Docker Images.

Η ομάδα τους επίσης εξέφρασε, ότι είναι πιθανό για το Docker να περιλάβει το δικό τους App container specification, συνδέοντας έτσι τα δύο project. Επίσης δήλωσαν ότι το CentOS θα συνεχίζει να υποστηρίζει το Docker.

4. Χρησιμοποιώντας το Docker

4.1 Η εγκατάσταση του Docker

4.1.1 Σε λειτουργικό σύστημα Windows

Για την έκδοση των Windows 10 Professional ή Enterprise 64-bit είναι διαθέσιμη η εφαρμογή Docker for Windows με native Windows user interface, με αυτόματες αναβαθμίσεις και εικονικοποίηση με την χρήση του Hyper-V του native εργαλείου εικονικοποίησης των Windows.

Για παλαιότερες εκδόσεις των Windows είναι διαθέσιμο το Docker Toolbox που περιέχει τα εργαλεία docker, docker-machine, docker-compose, Kitematic, Boot2Docker και το VirtualBox.

Για τις εκδόσεις Windows Server 2016 οι λειτουργίες του Docker είναι διαθέσιμες natively με την εντολή "docker run" χάρη στην διετή συνεργασία μεταξύ της ομάδας του Docker και της Microsoft.

Εγκατάσταση Docker σε Windows 10

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

- Πλοηγούμαστε στην σελίδα https://docs.docker.com/docker-for-windows/ ώστε να κατεβάσουμε το αρχείο εγκατάστασης του Docker (InstallDocker.msi)
- Εκτελούμε το αρχείο InstallDocker.msi κάνοντας διπλό κλικ πάνω του.
- Ακολουθούμε τα βήματα του οδηγού εγκατάστασης για την αποδοχή της άδειας, εξουσιοδότησης και συνεχίζουμε με την εγκατάσταση.
- Κάνουμε click στο Finish για να ολοκληρώσουμε την εγκατάσταση

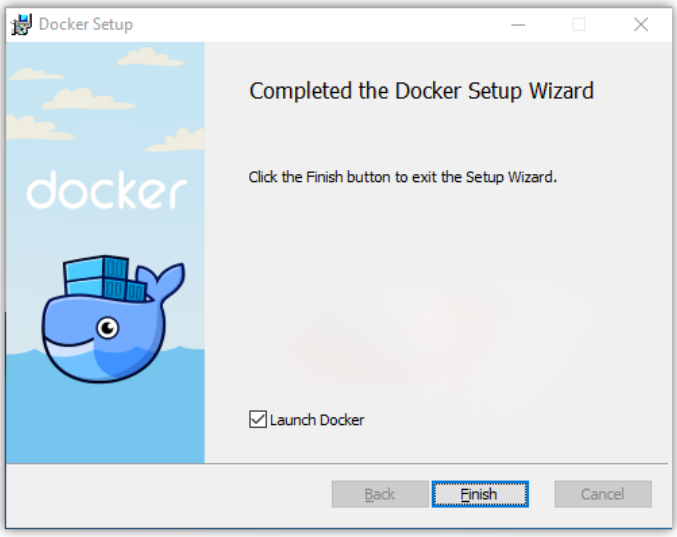

Figure 4.1: Τελειώνοντας την εγκατάσταση του Docker σε Windows 10.

Έναρξη Docker σε Windows 10

Όταν τελειώσει επιτυχώς η εγκατάσταση το Docker ξεκινάει αυτόματα. Το εικονίδιο της φάλαινας στο status bar δείχνει ότι το Docker είναι σε λειτουργία και είναι προσβάσιμο από ένα τερματικό.

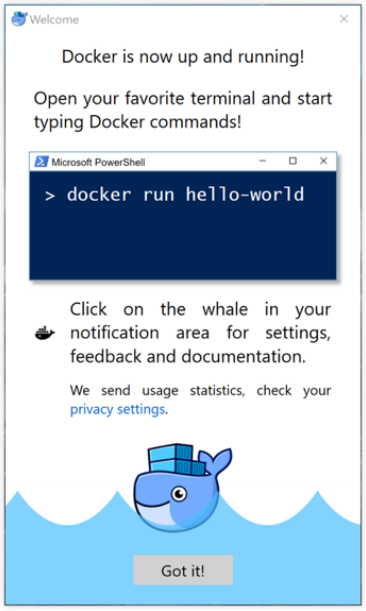

Figure 4.2: Έναρξη Docker σε Windows 10.

Τέλος επιλέγουμε το About Docker από το εικονίδιο των ειδοποιήσεων για να σιγουρευτούμε ότι έχουμε την τελευταία έκδοση.

Εξερεύνηση της εφαρμογής και εκτέλεση παραδειγμάτων Στα επόμενα βήματα θα δούμε μερικά παραδείγματα και τρόπους πειραματισμού με το Docker, τον έλεγχο των πληροφοριών της έκδοσης και θα βεβαιωθούμε ότι οι εντολές docker λειτουργούν κανονικά.

Ανοίγουμε ένα κέλυφος γραμμής εντολών όπως το cmd ή το Powershell και τρέχουμε εντολές του Docker όπως:

```
docker ps
docker version
docker info
```

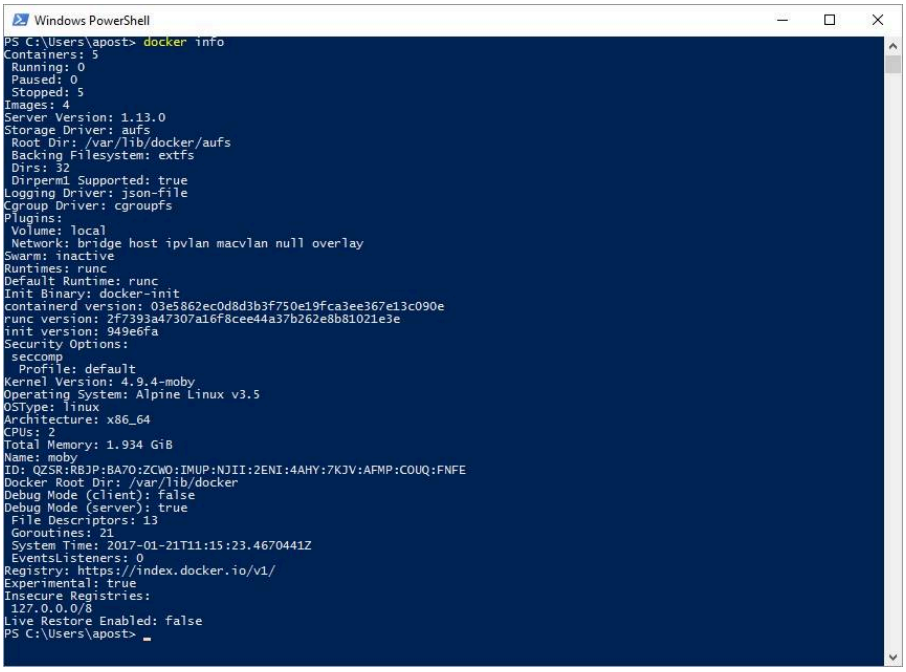

Figure 4.3: Παράδειγμα αποτελέσματος εντολής 'docker info' σε Windows 10.

Εδώ βλέπουμε την έξοδο της εντολής docker ps που εκτελείται σε ένα powershell. (Σε αυτό το παράδειγμα δεν εκτελείται ακόμα κανένα container.

```
CONTAINER ID    IMAGE    COMMAND          CREATED          STATU
```

Το παρακάτω είναι ένα παράδειγμα εξόδου της εντολής docker version:

```
Client:
Version:      1.13.0-rc3
API version:  1.25
Go version:   go1.7.3
Git commit:   4d92237
Built:        Tue Dec  6 01:15:44 2016
OS/Arch:      windows/amd64

Server:
```

```
Version:      1.13.0-rc3
API version:  1.25 (minimum version 1.12)
Go version:   go1.7.3
Git commit:   4d92237
Built:        Tue Dec  6 01:15:44 2016
OS/Arch:      linux/amd64
Experimental: true
```

Εκτελούμε την εντολή docker run hello-world για να ελέγξουμε το pull ενός image από το Docker hub και την εκκίνηση ενός container.

```
Hello from Docker.
This message shows that your installation
appears to be working correctly.

To generate this message, Docker took the
following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world"
image from the Docker Hub.
3. The Docker daemon created a new container
from that image which runs the executable
that produces the output you are currently
reading.
4. The Docker daemon streamed that output to
the Docker client, which sent it to your
terminal.
```

4.1.2 Σε λειτουργικό σύστημα Linux

Το Docker υποστηρίζεται σε διάφορα Linux distributions. Μεταξύ αυτών είναι και τα Arch Linux, CentOS, CRUX Linux, Debian, Fedora, Gentoo, Oracle Linux, Red Hat Enterprise Linux, openSUSE και SUSE Linux Enterprise καθώς και τα Ubuntu. Εμείς θα δούμε την εγκατάσταση του Docker για την τελευταία έκδοση LTS (Long Term Support) των Ubuntu (16.04.1 LTS).

Για αρχή θα πρέπει να γνωρίζουμε ότι το Docker λειτουργεί μόνο σε 64-bit εγκατάσταση του Linux και χρειάζεται έκδοση kernel 3.10 ή μεγαλύτερη. Σε εκδόσεις kernel μικρότερες της 3.10, μερικά χαρακτηριστικά του Docker δεν λειτουργούν σωστά και μπορεί να προκαλέσουν σφάλματα τα οποία με τη σειρά τους μπορεί να οδηγήσουν σε απώλεια δεδομένων σε κάποιες συγκεκριμένες περιπτώσεις.

Για να βρούμε την έκδοση του kernel σε Ubuntu 16.04.1 αρκεί να πληκτρολογήσουμε την παρακάτω εντολή σε ένα τερματικό (terminal):

```
$ uname -r
```

Η παραπάνω εντολή θα μας δώσει σαν έξοδο κάτι της μορφής:

```
4.4.0.58-generic
```

Αυτό σημαίνει ότι η έκδοση του kernel μας υποστηρίζει το Docker. Τώρα θα χρειαστεί να ρυθμίσουμε τον package manager του συστήματος που δεν είναι άλλος από τον aptitude έτσι ώστε να ενημερωθούν τα sources για την ύπαρξη των docker repositories για να κατεβάσουμε και να εγκαταστήσουμε τα ανάλογα packages.

Για το κατέβασμα και την ενημέρωση ενδείκνυται η χρήση https και για το λόγο αυτό κάνουμε μια ενημέρωση πριν προχωρήσουμε σε αλλαγές στα sources αρχεία του συστήματος μας, έτσι ώστε να βεβαιωθούμε ότι ο σύστημα μας έχει τις τελευταίες ενημερώσεις (up to date) και ότι έχουμε τα packages που απαιτούνται ώστε να μπορεί να λειτουργήσει το aptitude με τη χρήση https:

```
$ sudo apt-get update
$ sudo apt install apt-transport-https
ca-certificates
```

Στην συνέχεια, χρειάζεται να κατεβάσουμε το PGP key του Docker και να το περάσουμε στο τοπικό keychain του συστήματος. Τρέχουμε την εντολή:

```
    $ sudo apt-key adv \
--keyserver hkp://ha.pool.sks-keyservers.net:80 \
--recv-keys \
    58118E89F3A912897C070ADBF76221572C52609D
```

Τώρα χρειάζεται να προσθέσουμε σε ένα νέο αρχείο sources του συστήματος το repository από το οποίο θα κατεβάζει τις εκδόσεις του

Docker. Για Ubuntu 16.04-16.04.1 η εντολή είναι η παρακάτω:

```
$ echo "deb https://apt.dockerproject.org/repo \
ubuntu-xenial main" | sudo tee \
/etc/apt/sources.list.d/docker.list
```

Η παραπάνω εντολή γράφει ένα νέο αρχείο που ονομάζεται docker.list με μια γραμμή που αποτελεί την διεύθυνση του repository του Docker.

Τρέχουμε την παρακάτω εντολή για να ενημερώσουμε τα indexes του aptitude:

```
$ sudo apt-get update
```

Θα πρέπει να εγκαταστήσουμε επίσης κάποια πακέτα του kernel που επιτρέπουν την χρήση του οδηγού αποθήκευσης **aufs**.

```
$ sudo apt-get install \
linux-image-extra-$(uname -r) \
linux-image-extra-virtual
```

Για να εγκαταστήσουμε το Docker καθώς και για να ενημερώσουμε το σύστημα με τα τελευταία packages πλέον αρκεί να τρέξουμε:

```
$ sudo apt-get install \
docker-engine && apt-get upgrade
```

Αφού εγκατασταθεί το Docker μπορούμε να ελέγξουμε για το εάν λειτουργεί πληκτρολογώντας την παρακάτω εντολή:

```
$ service docker status
```

Εάν το docker daemon τρέχει, θα μας εμφανίσει ένα μήνυμα παρόμοιο με το παρακάτω:

```
 docker.service - Docker Application Container Engine
  Loaded: loaded (/lib/systemd/system/docker.service; enabled;
  vendor preset: enabled)
  Active: active (running) since 2017-01-08 10:42:43 EET;
  1min 22s ago
    Docs: https://docs.docker.com
Main PID: 20961 (dockerd)
   Tasks: 19
  Memory: 14.4M
     CPU: 378ms
  CGroup: /system.slice/docker.service
          20961 /usr/bin/dockerd -H fd://
          20968 docker-containerd -l
          unix:///var/run/docker/libcontainerd/docker-containerd.
```

Εάν το docker δεν καταφέρει να τρέξει, θα μας εμφανίσει κάποιο μήνυμα λάθους και αντί το process status να είναι running, θα είναι είτε failed είτε stopped. Στην περίπτωση που είναι stopped μπορούμε να ξεκινήσουμε το daemon (πάντα σαν root αφού το docker απαιτεί root privileges για να τρέξει) πληκτρολογώντας:

```
$ sudo service docker start
```

Για να επιβεβαιώσουμε ότι το Docker τρέχει σωστά, μπορούμε να κάνουμε μια δοκιμή με ένα default image που υπάρχει στο Docker repository για αυτό το λόγο, επομένως πληκτρολογούμε:

```
$ sudo docker run hello-world
```

Εάν το κάνουμε για πρώτη φορά αυτό τότε θα λάβουμε ένα τέτοιο μήνυμα γιατί το Docker ελέγχει πρώτα εάν υπάρχει το αρχείο εικόνας τοπικά πριν προσπαθήσει να το κατεβάσει εκ νέου από το repository:

```
Unable to find image 'hello-world:latest' locally
```

Έπειτα συνεχίζει με το παρακάτω μήνυμα:

```
latest: Pulling from library/hello-world
c04b14da8d14: Pull complete
Digest: sha256:0256e8a36e2070f7bf2d0b0763dbabdd...
Status: Downloaded newer image for hello-world:latest
Hello from Docker!
This message shows that your installation
appears to be working correctly.
To generate this message, Docker took
the following steps:
 1. The Docker client contacted
the Docker daemon.
 2. The Docker daemon pulled
the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new
container from that image which runs
the executable that prduces the output
you are currently reading.
 4. The Docker daemon streamed that
output to the Docker client, which
sent it to your terminal.
To try something more ambitious, you can run
an Ubuntu container with:
        $ docker run -it ubuntu bash
Share images, automate workflows, and more
with a free Docker Hub account:
        https://hub.docker.com
For more examples and ideas, visit:
        https://docs.docker.com/engine/userguide/
```

Εάν λάβουμε το παραπάνω μήνυμα ή κάτι παρόμοιο τότε το Docker λειτουργεί επιτυχώς στο σύστημα μας. Αυτό που συνέβη εν συντομία όταν τρέξαμε αυτή την εντολή ήταν ότι ο Docker client συνδέθηκε με τον Docker daemon (που τρέχει πλέον σαν service στο σύστημα) έπειτα ο Docker daemon κατέβασε την τελευταία έκδοση της εικόνας hello-world από το repository του Docker και δημιούργησε ένα νέο container από αυτή την εικόνα και πέρασε τα περιεχόμενα του στον client ο οποίος με τη σειρά του τα εμφάνισε στο τερματικό μας.

Επιπλέον παραμετροποιήσεις για λειτουργικό σύστημα Linux

Εκτέλεση Docker κατά την έναρξη του συστήματος

Για να ενεργοποιήσουμε την έναρξη του Docker κατά το boot του συστήματος αρκεί να τρέξουμε την παρακάτω εντολή (για συστήματα που χρησιμοποιούν το systemd σαν init σύστημα τους):

```
$ sudo systemctl enable docker
```

Για συστήματα που χρησιμοποιούν το upstart τότε το Docker είναι αυτόματα παραμετροποιημένο να λειτουργεί κατά την έναρξη του συστήματος.

OverlayFS driver

Εάν κάποιος δεν επιθυμεί να χρησιμοποιήσει το aufs σαν default storage driver ή έχει έκδοση kernel μεγαλύτερη του 4.6, για την οποία αυτή τη στιγμή δεν διατίθεται πακέτο linux-image-extra και linux-image-virtual, τότε μπορεί απλά να αλλάξει τις ρυθμίσεις με τις οποίες εκτελείται το Docker daemon δημιουργώντας το αρχείο /etc/docker/-daemon.json (είναι αναγκαίο να έχει root privileges ο χρήστης που θα δημιουγήσει αυτό το αρχείο). Χρησιμοποιώντας τον παρακάτω json κώδικα μπορούμε να παραμετροποιήσουμε το Docker και αντί να χρησιμοποιεί τον aufs driver να λειτουργεί με τον OverlayFS driver (διαθέσιμος από την έκδοση 3.18 του kernel και έπειτα).

```
{
  "storage-driver": "overlay"
}
```

Διαχείριση του Docker σαν non-root χρήστης

Εάν δεν επιθυμείτε να διαχειριστείτε το Docker σαν root χρήστης τότε υπάρχει αυτή η δυνατότητα, αφού το Docker daemon συνδέεται σε ένα unix socket αντί για TCP port. Σε αυτό το socket έχει πρόσβαση μόνο ο root χρήστης του μηχανήματος (σε αυτόν ανήκει) και οι χρήστες μπορούν να έχουν πρόσβαση σε αυτό μόνο με την χρήση του sudo. Το daemon τρέχει πάντα με δικαιώματα root. Επιπλέον, όταν ξεκινάει το Docker daemon, ελέγχει να δει εάν υπάρχει κάποιο group με όνομα docker και εάν το βρει τότε δίνει στο unix socket δικαιώματα εγγραφής και διαβάσματος από αυτό το group. Εάν δεν θέλουμε να δώσουμε δικαιώματα sudo σε κάποιον χρήστη μόνο για να χρησιμοποιεί το Docker, τότε αρκεί να δημιουργήσουμε ένα group με το όνομα docker και να κάνουμε τον χρήστη μέλος αυτού του group. Οπότε μπαίνουμε σαν

κάποιος χρήστης με δικαιώματα sudo και πληκτρολογούμε τις παρακάτω εντολές:

```
$ sudo groupadd docker
```

Δημιουργήσαμε το group με το όνομα docker και τώρα το μόνο που θα χρειαστεί να κάνουμε είναι να βάλουμε τον επιθυμητό χρήστη να γίνει μέλος αυτού του group:

```
$ sudo usermod -aG docker $USER
```

Αρκεί ο χρήστης να κάνει logout/login και θα είναι πλέον μέλος αυτού του group. Έτσι θα μπορεί να χρησιμοποιεί και να διαχειριστεί το Docker χωρίς δικαιώματα sudo.

Χρήση firewall (UFW) και docker στο ίδιο σύστημα

Εάν χρησιμοποιείτε το UFW (Απλό Τείχος προστασίας) [36] για το ίδιο σύστημα καθώς τρέχει το Docker, θα χρειαστεί να κάνετε επιπλέον ρυθμίσεις, αφού το Docker χρησιμοποιεί μια γέφυρα για τη διαχείριση της δικτύωσης των containers. Από προεπιλογή, το UFW απορρίπτει όλη την προωθούμενη κίνηση. Θα πρέπει να ορίσετε την κατάλληλη πολιτική προώθησης για το UFW.

Αν θέλετε να αποκτήσετε πρόσβαση στο Docker Remote API από άλλο host και έχετε ενεργοποιήσει απομακρυσμένη πρόσβαση, θα πρέπει να ρυθμίσετε το UFW να επιτρέπει εισερχόμενες συνδέσεις στη θύρα του Docker: η προεπιλογή είναι η θύρα 2376 όταν η κρυπτογράφηση με TLS έχει ενεργοποιηθεί ή η θύρα 2375 διαφορετικά. Από προεπιλογή, το Docker τρέχει χωρίς να είναι ενεργοποιημένο το TLS. Εάν δεν θέλετε να χρησιμοποιήσετε TLS, αποθαρρύνεται έντονα η πρόσβαση στο Docker Remote API από απομακρυσμένους υπολογιστές, για την αποτροπή επιθέσεων κλιμάκωσης-δικαιωμάτων (privilege escalation attacks). Θα πρέπει να ακολουθήσετε τα παρακάτω βήματα για να επιτρέψετε στο Docker να λειτουργήσει παράλληλα με το UFW.

Πραγματοποιήστε εισαγωγή στο σύστημα σαν χρήστης με δικαιώματα sudo και ελέγξτε εάν το UFW είναι ενεργοποιημένο:

```
$ sudo ufw status
```

Εάν το αποτέλεσμα της εντολής είναι κάτι τέτοιο:

```
Status: inactive
```

Τότε δεν θα χρειαστεί να προχωρήσετε στα επόμενα βήματα αφού το firewall είναι απενεργοποιημένο. Εναλλακτικά, θα πρέπει πρώτα να παραμετροποιηθεί το firewall ώστε να δέχεται την προωθούμενη κίνηση, οπότε ανοίγουμε το αρχείο /etc/default/ufw:

```
$ sudo vim /etc/default/ufw
```

Και κάνουμε αλλαγή στη γραμμή από:

```
DEFAULT_FORWARD_POLICY="DROP"
```

σε

```
DEFAULT_FORWARD_POLICY="ACCEPT"
```

Εν συνεχεία, αποθηκεύουμε τις αλλαγές και βγαίνουμε από το αρχείο. Εάν θέλουμε να επιτρέψουμε πρόσβαση από εξωτερικούς hosts στο Docker Remote API τότε ανοίγουμε την θύρα 2375 (εάν δεν χρησιμοποιούμε TLS) ή την θύρα 2376 αντίστοιχα.

```
$ sudo ufw allow 2376/tcp
```

Ενημερώνουμε το τρέχον UFW με τις αλλαγές πληκτρολογώντας:

```
$ sudo ufw reload
```

4.2 Διαχείριση Docker Images

4.2.1 Docker Registry

Το Docker Registry [37] είναι μία μη-καταστατική και άκρως επεκτάσιμη εφαρμογή εξυπηρετητή που αποθηκεύει και δίνει την δυνατότητα για τον διαμοιρασμό των Docker images. Το Registry είναι χρήσιμο για περιπτώσεις όπου χρειάζεται αυστηρός έλεγχος για το που αποθηκεύονται τα images, για πλήρη έλεγχο πάνω στην διανομή των images και για την ενσωμάτωση της αποθήκευσης και την διανομή σε μία εσωτερική επιχειρησιακή ροή εργασιών.

Για την δημιουργία ενός Registry χρειάζεται η έκδοση του Docker Engine 1.6 ή ανώτερη και οι εξής εντολές:

Εκκίνηση του registry

```
$ docker run -d -p 5000:5000 --name registry registry:2
```

Η παραπάνω εντολή ξεκινάει ένα container το οποίο τρέχει μια έκδοση εξυπηρετητή μητρώου του Docker στην οποία μπορούμε να έχουμε πρόσβαση μέσω της θύρας 5000 του host υπολογιστή (η θύρα 5000 του host υπολογιστή έχει αντιστοιχιθεί στην θύρα 5000 του container).

Έπειτα ξεκινάμε την διαδικασία μεταφόρτωσης μιας νέας εικόνας από το Docker Hub:

```
$ docker pull ubuntu
```

Αφού κατέβει το image, χρησιμοποιούμε την εντολή docker tag για να δώσουμε όνομα και να καθορίσουμε το repository/registry στο οποίο θα βρίσκουμε από εδώ και στο εξής το συγκεκριμένο docker image.

```
$ docker tag ubuntu \
localhost:5000/ubuntu-image-local-repo
```

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

```
$ docker push \
localhost:5000/ubuntu-image-local-repo
```

Με παρόμοιο τρόπο μπορούμε να παραλάβουμε πίσω το image:

```
$ docker pull \
localhost:5000/ubuntu-image-local-repo
```

Με αυτόν τον τρόπο μπορούμε να δημιουργήσουμε το δικό μας Registry και να μεταφορτώσουμε τα 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 (με την παράμετρο -v θα σβήσουμε επίσης και τα πιθανά volumes-αποθηκευτικούς χώρους που έχουν δημιουργηθεί για το container):

```
$ docker rm -v registry
```

Η υπηρεσία Docker Hub

Το Docker Hub [38] είναι μία 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 γίνεται δυνατό το deployment αυτών των images στο cloud. Προσφέρει μία κεντρική πηγή πόρων για την ανακάλυψη images, συνεργασία μεταξύ χρηστών ή ομάδων και την αυτοματοποίηση της ροής των εργασιών στον κύκλο ανάπτυξης εφαρμογών.

Για την χρήση του Docker Hub χρειάζεται ένας λογαριασμός στο Docker (Docker ID) και η αναζήτηση δημόσιων repositories ή images μπορεί να πραγματοποιηθεί με δύο τρόπους. Είτε απο το site του Docker Hub ή με την εντολή docker search, τα ιδιωτικά repositories δεν εμφανίζονται στα αποτελέσματα της αναζήτησης και η διαχείριση τους γίνεται από το Dashboard της σελίδας του Docker Hub.

Το Docker επίσης προσφέρει πρόσβαση 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)

4.2.2 Εμφάνιση και εκτέλεση Docker Images

Με την χρήση της παρακάτω εντολής μπορούμε να δούμε ποια διαθέσιμα docker images έχουμε τοπικά στο σύστημα μας:

```
$ docker images
```

Η εντολή θα δώσει σαν έξοδο το παρακάτω αποτέλεσμα (για το σύστημα μας):

```
REPOSITORY              TAG              IMAGE ID
CREATED         SIZE
ubuntu          16.04    1d073211c498
3 days ago      187.9 MB
busybox         latest   2c5ac3f849df
5 days ago      1.113 MB
training/webapp latest   54bb4e8718e8
5 months ago    348.7 MB
```

Με την λίστα των images μπορούμε να δούμε από ποιο repository προέρχονται πχ. ubuntu, τα tags που είναι ουσιαστικά πληροφορίες που αφορούν την έκδοση του κάθε image πχ. 14.04 και το μοναδικό αναγνωριστικό image ID του κάθε image.

Ένα repository τις περισσότερες φορές διατηρεί διάφορες εκδόσεις ενός Image. Στην περίπτωση του Ubuntu image υπάρχουν images που καλύπτουν διαφορετικές εκδόσεις όπως 12.04, 12.10, 13.04, 13.10, 14.04, 16.04, 16.10.

Κάθε έκδοση έχει ένα αναγνωριστικό tag στο οποίο γίνεται αναφορά με αυτό το τρόπο:

```
ubuntu:16.04 - repository:version
```

Έτσι με την εκτέλεση ενός container αναφερόμαστε σε ένα tagged image με αυτόν τον τρόπο:

```
$ docker run -t -i ubuntu:16.04 /bin/bash
```

Εάν θέλαμε να τρέξουμε την έκδοση 14.04 αντί για την 16.04 των Ubuntu θα χρησιμοποιούσαμε:

```
$ docker run -t -i ubuntu:14.04 /bin/bash
```

4.2.3 Αναζήτηση και εκτέλεση ενός Docker Image

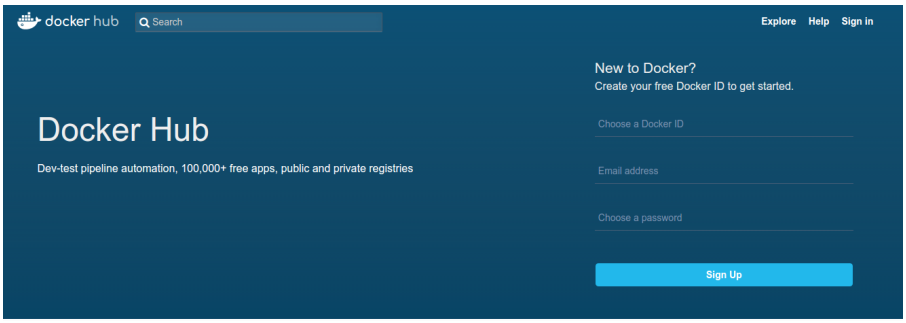

Figure 4.4: Κομμάτι ιστοσελίδας του Docker Hub.

Ανοίγουμε σε έναν browser την σελίδα του Docker Hub (https://hub.docker.com) και πληκτρολογούμε whalesay στην μπάρα αναζήτησης για να βρούμε το image που θα χρησιμοποιήσουμε σε αυτό το παράδειγμα. Επιλέγουμε το docker/whalesay image στα αποτελέσματα της αναζήτησης και βλέπουμε όλες τις σχετικές πληροφορίες και οδηγίες για το image που επιλέξαμε.

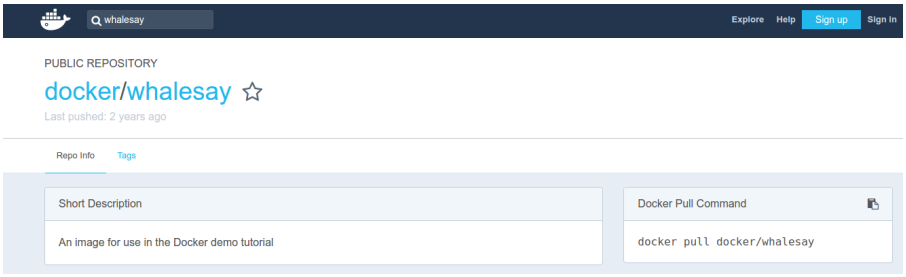

Figure 4.5: Η επίσημη σελίδα του docker/whalesay image.

Εφόσον σιγουρευτούμε ότι το Docker είναι σε λειτουργία ανοίγουμε ένα τερματικό και πληκτρολογούμε:

```
$ docker run docker/whalesay cowsay boo
```

Αυτή η εντολή εκτελεί το whalesay image σε ένα container και το αποτέλεσμα είναι το ακόλουθο:

```
$ docker run docker/whalesay cowsay boo

Unable to find image 'docker/whalesay:latest' locally
latest: Pulling from docker/whalesay
e9e06b06e14c: Pull complete
a82efea989f9: Pull complete
37bea4ee0c81: Pull complete
07f8e8c5e660: Pull complete
676c4a1897e6: Pull complete
5b74edbcaa5b: Pull complete
1722f41ddcb5: Pull complete
99da72cfe067: Pull complete
```

```
5d5bd9951e26: Pull complete
fb434121fc77: Already exists
Digest: sha256:d6ee73f978a366cf97974115abe9c40...
Status: Downloaded newer image for docker/whalesay:latest

 _____
< boo >
 -----
    \
     \
      \
                    ##         .
              ## ## ##        ==
           ## ## ## ##       ===
       /""""""""""""""""___/ ===
  ~~~ {~~ ~~~~ ~~~ ~~~~ ~~ ~ /  ===- ~~~
       \______ o          __/
        \    \        __/
          \____\______/
```

Επειδή είναι η πρώτη φορά που εκτελούμε αυτό το image το docker ψάχνει εάν υπάρχει αυτό το image τοπικά στον δίσκο και εάν το image δεν βρίσκεται τοπικά το κατεβάζει από το Docker Hub και το εκτελεί.

4.2.4 Δημιουργία ενός Docker Image

Για την δημιουργία ενός Docker Image, θα χρειαστεί να δημιουργηθεί ένα αρχείο **dockerfile** το οποίο είναι ένα αρχείο κειμένου που περιγράφει τα αρχεία, το περιβάλλον καθώς και τις εντολές οι οποίες συντελούν ένα Docker Image.

Με την χρήση ενός τερματικού δημιουργούμε ένα νέο κατάλογο αρχείων, και εισερχόμαστε με το τερματικό σε αυτόν:

```
$ mkdir -p my-docker && cd my-docker
```

Επεξεργαζόμαστε ένα νέο αρχείο κειμένου με όνομα dockerfile στο τρέχον κατάλογο αρχείων, χρησιμοποιώντας έναν επεξεργαστή κειμένου όπως nano ή vi σε Linux, ή notepad στα Windows.

```
$ vim dockerfile
```

Το τελικό αρχείο θα πρέπει να δείχνει όπως το παρακάτω:

```
FROM docker/whalesay:latest
RUN apt-get -y update && apt-get install -y fortunes
CMD /usr/games/fortune -a | cowsay
```

Η λέξη-κλειδί FROM στην πρώτη γραμμή λέει στο Docker σε ποιο image βασίζεται το νέο image που θα δημιουργήσουμε. Όλα τα images πρέπει να βασιστούν σε κάποιο αρχικό image.

Προσθέτουμε μια RUN εντολή η οποία προκαλεί την εκτέλεση της φράσης που ακολουθεί από το προεπιλεγμένο τερματικό του συστήματος που τρέχει το image, και επειδή το whalesay image είναι βασισμένο στα Ubuntu, που χρησιμοποιούν το apt-get για την

εγκατάσταση των packages αυτές οι δύο εντολές ανανεώνουν την λίστα των διαθέσιμων packages στο image και εγκαθιστούν σε αυτό το πρόγραμμα fortunes.

Προσθέτουμε μια CMD εντολή η οποία ενημερώνει το image για την τελευταία εντολή προς εκτέλεση αφότου στηθεί το περιβάλλον. Αυτή η εντολή εκτελεί το πρόγραμμα fortune -a και στέλνει το αποτέλεσμα στην εντολή cowsay.

Αποθηκεύουμε το αρχείο και κλείνουμε τον επεξεργαστή κειμένου. Σε αυτό το σημείο είμαστε έτοιμοι να δημιουργήσουμε ένα καινούριο image. Για να το κάνουμε αυτό, στον κατάλογο τον οποίο έχουμε το αρχείο dockerfile τρέχουμε την παρακάτω εντολή:

```
$ docker build -t fortunes-custom .
```

Η εντολή αυτή διαβάζει το Dockerfile [39] στον τρέχων κατάλογο αρχείων και κατεργάζεται τις εντολες μια προς μια για να δημιουργήσει ένα image ονομαζόμενο fortunes-custom στο τοπικό μας σύστημα και να το προσθέσει στα τοπικά images μας.

Τα βήματα που κάνει το Docker είναι τα εξής:

- Ελέγχει εάν υπάρχει το whalesay image τοπικά αλλιώς το κατεβάζει από το Docker Hub.
- Ξεκινά ένα προσωρινό container που τρέχει το whalesay image.
- Εκτελεί όλες τις εντολές RUN που έχουμε δώσει μέσα στο container.

Στο τέλος κάθε βήματος, εμφανίζεται ένα ID. Αυτό είναι το ID του επιπέδου που δημιουργήθηκε στο τρέχον βήμα. Κάθε γραμμή σε ένα Dockerfile αντιστοιχεί σε ένα επίπεδο μέσα στο image. Όταν η εντολή RUN τελειώσει, ένα νέο επίπεδο δημιουργείται και το προσωρινό container διαγράφεται.

Ένα νέο προσωρινό container δημιουργείται και το Docker προσθέτει ένα επίπεδο στην γραμμή CMD του Dockerfile και διαγράφεται το προσωρινό container.

Έτσι η δημιουργία του image fortunes-custom ολοκληρώθηκε. Εάν επιθυμούμε να εκτελέσουμε το image που δημιουργήσουμε τότε μπορούμε να το κάνουμε με την παρακάτω εντολή:

```
$ docker run fortunes-custom
```

Το αποτέλεσμα θα δείχνει παρόμοιο με το παρακάτω:

```
 ------------------------------------
/ Dirksen's Three Laws of Politics: \
|                                    |
| (1) Get elected.                   |
|                                    |
| (2) Get re-elected.                |
|                                    |
| (3) Don't get mad, get even.       |
|                                    |
\ -- Sen. Everett Dirksen            /
 ------------------------------------
    \
     \
      \
                    ##         .
              ## ## ##        ==
           ## ## ## ##       ===
       /""""""""""""""""___/ ===
  ~~~ {~~ ~~~~ ~~~ ~~~~ ~~ ~ /  ===- ~~~
       \______ o          __/
        \    \        __/
          \____\______/
```

4.3 Διαχείριση Docker Containers

4.3.1 Ξεκινώντας ένα Docker Container από κάποιο 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 παραπέμθηκε 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 είτε να φτιάξουμε 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.

Ας χρησιμοποιήσουμε την σελίδα του Docker Hub για να βρούμε ένα image:

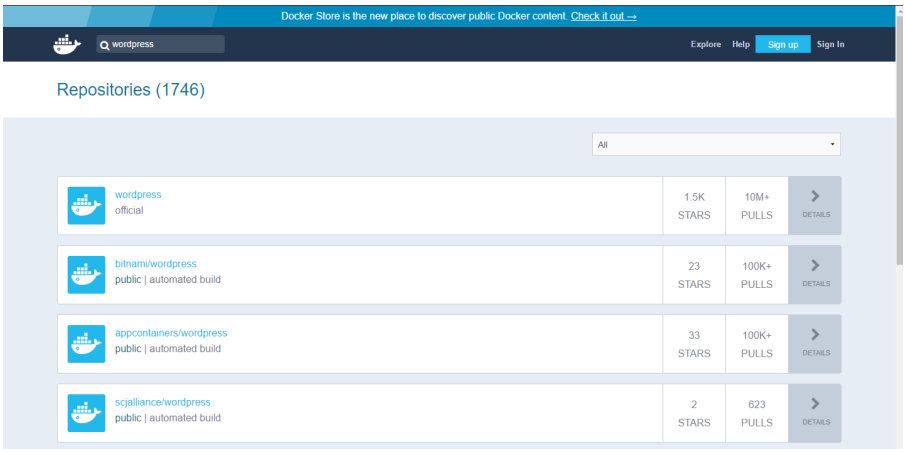

Figure 4.6: Αναζήτηση για wordpress image στο Docker Hub.

Χρησιμοποιούμε την εντολή:

```
$ docker pull wordpress
```

Έτσι το Docker θα κατεβάσει την τελευταία έκδοση του wordpress Image και θα την αποθηκεύσει μαζί με τις λεπτομέρειες της έκδοσης που κατέβασε τοπικά για μελλοντική χρήση. Η έξοδος στο τερματικό θα μοιάζει με την παρακάτω:

```
Using default tag: latest
latest: Pulling from library/wordpress
5040bd298390: Pull complete
568dce68541a: Pull complete
6a832068e64c: Pull complete
3b0f3d176a5b: Pull complete
20cc248a5690: Pull complete
6ff565538ee6: Pull complete
9f1077228581: Pull complete
57359f144a19: Pull complete
9754ef36b033: Pull complete
e156df35b624: Pull complete
df09daa2224a: Pull complete
bfa0d8031302: Pull complete
fe629d500af7: Pull complete
2761420c8e70: Pull complete
71580740e433: Pull complete
22ff3670a5e6: Pull complete
39f4427993e5: Pull complete
9ab2843f4934: Pull complete
```

```
Digest: sha256:9e397d900023f0ca04b8ce...
Status: Downloaded newer image for wordpress:latest
```

Αφού κατέβει μπορούμε να δούμε τα διαθέσιμα images που έχουμε τοπικά με την εντολή:

```
$ docker images
REPOSITORY TAG  IMAGE ID  CREATED  SIZE
wordpress  latest dfbefe759772
39 hours ago 400 MB
```

Για να τρέξουμε το wordpress ωστόσο χρειάζεται ακόμα ένα container που να αποτελεί μηχανισμό διαχείρισης βάσης δεδομένων MySQL.

Για να κατεβάσουμε το image αυτό πληκτρολογούμε την παρακάτω εντολή:

```
$ docker pull mysql:latest
```

Αφού κατέβει και τρέχοντας πλέον την εντολή:

```
$ docker images
```

Βλέπουμε και το image του MySQL Server στην λίστα με τα images:

```
REPOSITORY  TAG  IMAGE ID  CREATED  SIZE
wordpress latest dfbefe759772
39 hours ago    400 MB
mysql latest f3694c67abdb
8 days ago      400 MB
```

Για να ξεκινήσουμε το wordpress container θα πρέπει να του δώσουμε κάποια επιπλέον ορίσματα τα οποία αφορούν κυρίως τον τρόπο αλληλεπίδρασης με την βάση δεδομένων. Θα πρέπει πρώτα να ξεκινήσουμε μια εκδοχή (instance) της βάσης δεδομένων και για να το κάνουμε αυτό πληκτρολογούμε την παρακάτω εντολή:

```
$ docker run -d --name=wpdb -e \
"MYSQL_ROOT_PASSWORD=root" mysql
```

Η παραπάνω εντολή ξεκινάει ένα container σε detached mode με την παράμετρο -d έτσι αυτό τρέχει στο background και δεν μπλοκάρει το υπάρχον thread στο terminal με την εκτέλεση του. Επιπλέον ονομάζουμε το container με την παράμετρο –name σε κάτι πιο εύκολα αναγνωρίσιμο έτσι ώστε να μπορούμε να το καταλάβουμε πιο εύκολα κατά την χρήση της εντολής docker ps. Τέλος, του δίνουμε ένα επιπλέον όρισμα που αφορά τα environment variables με τα οποία εκκινείται το container, το συγκεκριμένο είναι απαιτούμενο και αφορά το password του root χρήστη της βάσης δεδομένων. Το password που χρησιμοποιήθηκε εδώ είναι κάτι πολύ απλό, ωστόσο σε περιβάλλον παραγωγής θα πρέπει να χρησιμοποιηθεί ένα πολύ πιο ασφαλές password.

Αφού εκκινηθεί το container της βάσης δεδομένων μπορούμε να εκκινήσουμε και το container του wordpress και να συνδέσουμε τα δυο μεταξύ τους με την χρήση της εντολής:

```
$ docker run --name wp --link wpdb:mysql \
-p 8080:80 -d wordpress
```

Η εντολή αυτή θα εκκινήσει ένα νέο container που θα τρέχει το λογισμικό του wordpress και θα το συνδέσει με την χρήση του –link με το container της βάσης που εκτελείται ήδη με όνομα wpdb. Πλέον τρέχουν και τα δύο containers και μπορούμε να δούμε το wordpress από έναν φυλλομετρητή στην διεύθυνση http://localhost:8080 με βάση την θύρα που δώσαμε στην παράμετρο -p (port).

4.3.2 Εμφάνιση των Docker Containers

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

```
$ docker ps
```

Η έξοδος της εντολής θα πρέπει να δείχνει όπως η παρακάτω:

```
CONTAINER ID  IMAGE  COMMAND  CREATED
STATUS  PORTS  NAMES
6e8ef536ad24  wordpress  "docker-entrypoint..."
6 minutes ago Up 6 minutes 0.0.0.0:8080->80/tcp
wp
3632687138b1  mysql  "docker-entrypoint..."
7 minutes ago  Up 7 minutes  3306/tcp  wpdb
```

Δηλαδή μια περιγραφή του αναγνωριστικού του container (container ID), την εικόνα από την οποία προήλθε η συγκεκριμένη εκδοχή του container καθώς και την εντολή αλλά και πότε δημιουργήθηκε το container καθώς και πόσο χρόνο εκτελείται. Επιπλέον φαίνεται και η παρούσα κατάσταση του container.

Εάν θέλουμε να δούμε όλα τα containers και όσα εκτελέστηκαν στο παρελθόν αλλά και όσα εκτελούνται την δεδομένη χρονική στιγμή μπορούμε να χρησιμοποιήσουμε την εντολή:

```
$ docker ps -a
```

Εδώ τα αποτελέσματα της εντολής είναι λίγο διαφορετικά παρόλο που το είδος της πληροφορίας εξακολουθεί να είναι το ίδιο.

```
CONTAINER ID  IMAGE  COMMAND  CREATED
STATUS  PORTS  NAMES
6e8ef536ad24  wordpress  "docker-entrypoint..."
17 minutes ago
Up 17 minutes 0.0.0.0:8080->80/tcp  wp
5f963b633408  wordpress  "docker-entrypoint..."
18 minutes ago  Exited (1) 18 minutes ago
dazzling_bassi
3632687138b1  mysql  "docker-entrypoint..."
18 minutes ago  Up 18 minutes  3306/tcp  wpdb
```

Παρατηρούμε ότι εδώ φαίνεται και ένα container το οποίο δεν εκτελείται πλέον αλλά πραγματοποίησε έξοδο πριν από το 18 λεπτά.

4.3.3 Κατανομή πόρων σε ένα Docker Container

Ο βασικότερος τρόπος για να ελέγξουμε την κατανομή πόρων είναι με την χρήση παραμέτρων στην εντολή:

```
$ docker run
```

Μερικές παράμετροι για να περιορίσουμε την χρήση της CPU είναι οι παρακάτω:

- **–cpu-shares** - Κύκλοι εκτέλεσης CPU όσο μεγαλύτερο το νούμερο τόσο μεγαλύτερο ποσοστό % παίρνει το container.
- **–cpu-count=0**- Ο μέγιστος αριθμός επεξεργαστών στους οποίους έχει πρόσβαση ένα container. Σε containers Windows Server, το flag είναι ορισμένο ως το ποσοστό του συνόλου της χρήσης CPU και οι έλεγχοι των πόρων του επεξεργαστή είναι αμοιβαία αποκλεισμένοι με βάση την σειρά προτεραιότητας η οποία είναι CPUCount πρώτη, τότε CPUShares και CPUPercent τελευταία.
- **–cpu-percent=0** - Αφορά Windows συστήματα και τον περιορισμό του συνολικού ποσοστού επί τοις εκατό του CPU που χρησιμοποιείται από ένα container.
- **–cpu-quota=0** - Ίδιο με το flag cpu-count, αφορά περιορισμό της χρήσης της CPU από το container. Από προεπιλογή, τα containers τρέχουν με την πλήρη χρήση των πόρων της CPU. Αυτό το flag ενημερώνει το kernel να περιορίσει τη χρήση της CPU του container σύμφωνα με την τιμή που έχει ορίσει ο χρήστης.
- **–cpus=0.0** - Ο αριθμός των CPU. Η προεπιλογή είναι 0,0 το οποίο σημαίνει ότι δεν υπάρχει όριο.

Πέρα από τον έλεγχο των πόρων του επεξεργαστή μπορούμε να ελέγξουμε και την ποσότητα της μνήμης RAM που διατίθεται για κάποιο container χρησιμοποιώντας τα ανάλογα flags κατά την εκτέλεση της εντολής docker run.

- **-m, –memory=’”** - Όριο διαθέσιμης μνήμης (μορφή: <αριθμός>[<μονάδα>], όπου μονάδα = b, k, m ή g) . Δίνει τη δυνατότητα να περιοριστεί η διαθέσιμη μνήμη σε ένα container. Αν ο host υποστηρίζει μνήμη swap, τότε η τιμή -m μνήμης μπορεί να είναι μεγαλύτερη από τη φυσική μνήμη RAM. Εάν προσδιοριστεί ένα μηδενικό όριο τότε δεν υπάρχει περιορισμός στην μνήμη που εμφανίζεται ως διαθέσιμη για χρήση από το container.
- **–memory-reservation=’”** - Ήπιο όριο μνήμης (μορφή: <αριθμός> [<μονάδα>], όπου μονάδα = b, k, m ή ζ) Μετά τη ρύθμιση της κράτησης μνήμης, όταν το σύστημα εντοπίσει απώλεια μνήμης ή χαμηλή μνήμη, αναγκάζει τα containers να περιορίσουν την κατανάλωσή τους με βάση την τιμή του memory-reservation. Έτσι, θα πρέπει πάντα να οριστεί μια τιμή κάτω από την τιμή της

παραμέτρου –memory, αλλιώς το όριο με την μικρότερη τιμή από τα δύο όρια θα υπερισχύσει. Από προεπιλογή, η τιμή του memory-reservation θα είναι το ίδιο με το όριο μνήμης –memory.

- **–memory-swap="LIMIT"** - Μια οριακή τιμή ίση με τη μνήμη swap. Πρέπει να χρησιμοποιείται με την flag -m (–memory). Το όριο swap θα πρέπει πάντα να είναι μεγαλύτερο από ό,τι το όριο -m (–memory). Από προεπιλογή, το όριο μνήμης swap "LIMIT" είναι το διπλάσιο της τιμής του –memory flag.

Παράδειγμα χρήσης της κατανομής πόρων

Θα χρησιμοποιήσουμε ένα από τα προηγούμενα containers και συγκεκριμένα το container που τρέχει ένα instance του MySQL server, αλλά αυτή τη φορά θα περιορίσουμε τους πόρους που μπορεί να χρησιμοποιήσει με την παρακάτω εντολή:

```
$ docker run --name="mini-resources-mysql" \
--memory="512m" \
--cpu-count="1" \
-e="MYSQL_ALLOW_EMPTY_PASSWORD=1"
-d mysql:latest
```

Αυτό που κάναμε εδώ ήταν να τρέξουμε το container με μερικές έξτρα παραμέτρους όπως –name για να του δώσουμε ενα αντιπροσωπευτικό όνομα, –cpu-count περιορίσαμε τον αριθμό των CPUs που μπορεί να χρησιμοποιήσει σε 1 και –memory για να περιορίσουμε την χρήση της μνήμης RAM σε 512mb το μέγιστο.

Μπορούμε να δούμε την χρήση πόρων ενός container με την εντολή docker inspect <container id ή όνομα> το αποτέλεσμα θα συμπίπτει με το παρακάτω (στο οποίο έχουμε κρατήσει μόνο τα κομμάτια που μας ενδιαφέρουν και αφορούν την κατανομή πόρων που αλλάξαμε):

```
"Memory": 536870912,
"CpuCount": 1,
```

Ο περιορισμός στην CPU είναι εμφανής ενώ η διαθέσιμη μνήμη εμφανίστηκε σε Bytes τα οποία κάνοντας την διαίρεση 536870912 / 1024 / 1024 = 512m που είναι ο αριθμός της διαθέσιμης μνήμης στην οποία περιορίσαμε το container.

4.3.4 Χρήση Volumes για διαχείριση των δεδομένων σε ένα container

Όπως έχει προαναφερεθεί και σε προηγούμενο κεφάλαιο τα containers δεν αποθηκεύουν πληροφορία, έτσι μετά την διαγραφή κάποιου container τα δεδομένα που υπήρχαν σε αυτό θα χαθούν. Για να μπορέσουμε να διατηρήσουμε τα δεδομένα μας θα πρέπει να συνδέσουμε το container με ένα volume.

Στο προηγούμενο container που εκτελέσαμε μπορούμε να πάμε να το συνδέσουμε με ένα volume έτσι ώστε να μπορούμε να κάνουμε αλλαγές στα αρχεία του οι οποίες θα αποθηκεύονται και στο τοπικό σύστημα.

Για να το κάνουμε αυτό θα πρέπει να σβήσουμε το ήδη υπάρχον container και να δημιουργήσουμε ένα νέο.

Με την παρακάτω εντολή θα σβήσουμε το container αφού το σταματήσουμε:

```
$ docker stop wp && docker rm wp
```

Το αποτέλεσμα αυτής της εντολής θα πρέπει να είναι το όνομα του container, έτσι υποδηλώνεται από το σύστημα ότι διαγράφηκε επιτυχώς. Έπειτα ξεκινάμε το container με ένα volume:

```
$ docker run --name wp --link wpdb:mysql \
-d -v /home/klipitkas/Desktop/wp:/var/www/html \
-p 8080:80 wordpress
```

Η παραπάνω εντολή κάνει αρκετά πράγματα: ξεκινάει δίνοντας το όνομα "wp" στο νέο container κάτι που θα αποτελεί το αναγνωριστικό του από εδώ και στο εξής, έπειτα το συνδέει με ένα ήδη υπάρχον container που τρέχει και ονομάστηκε νωρίτερα "wpdb" το οποίο αποτελεί την βάση δεδομένων, η παράμετρος -d προκαλεί την εκτέλεση του container στο παρασκήνιο σαν detached από το τερματικό στο οποίο την εκτελούμε.

Έπειτα με την παράμετρο -v δίνουμε το volume στην παρακάτω μορφή -v τοπικός-κατάλογος : κατάλογος-container και έπειτα με την παράμετρο -p την θύρα η οποία θα αντιστοιχηθεί από τον host υπολογιστή προς το container με την μορφή -p θύρα-host : θύρα-container και τέλος το όνομα του image από το οποίο θα δημιουργηθεί το container.

Πρέπει να σημειωθεί ότι ο τοπικός φάκελος που επιλέξαμε στο σύστημα μας δεν προϋπήρχε αλλά δημιουργήθηκε κατά την έναρξη του container με δικαιώματα read-write (rw) από προεπιλογή. Εάν θέλουμε να δώσουμε μόνο δικαιώματα διαβάσματος από το container θα πρέπει να αλλάξουμε την τιμή του -v και να δώσουμε την παρακάτω εντολή αντίστοιχα:

```
$ docker run --name wp --link wpdb:mysql \
-d -v /home/klipitkas/Desktop/wp:/var/www/html:ro \
-p 8080:80 wordpress
```

Αυτό θα δώσει δικαιώματα εγγραφής στον κατάλογο /var/www/html του container από το host σύστημα και θα δώσει μόνο δικαιώματα διαβάσματος στο σύστημα του container. Με αυτόν τον τρόπο μπορούμε από τον host υπολογιστή να γράψουμε αρχεία μέσα στον κατάλογο του container και το container να μπορεί να έχει πρόσβαση για διάβασμα σε αυτά τα αρχεία αλλά όχι εγγραφής.

4.3.5 Διαχείριση κατάστασης και έλεγχος των containers εσωτερικά

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 και συγκεκριμένα με το STDOUT της κεντρικής διεργασίας του container.
- **commit** - Δημιουργία ενός νέου image από τις αλλαγές ενός ήδη υπάρχοντος container.
- **cp** - Αντιγραφή αρχείων ανάμεσα στο σύστημα αρχείων του container και το τοπικό σύστημα.
- **create** - Δημιουργεί ένα νέο container, η διαφορά αυτής της εντολής με την εντολή run είναι πως η εντολή create απλά δημιουργεί το container χωρίς να το εκτελέσει-εκκινήσει και μετά θα πρέπει να τρέξει το container με την εντολή docker start, ενώ η εντολή docker run δημιουργεί και εκκινεί το container απευθείας.
- **diff** - Εμφανίζει τις αλλαγές ανάμεα σε ένα εκτελούμενο container και ένα container χωρίς αλλαγές όπως προήλθε από το image του.
- **exec** - Εκτελεί μια εντολή σε ένα ήδη εκτελούμενο container.
- **export** - Εξάγει το σύστημα αρχείων ενός container σε ένα αρχείο tar.
- **inspect** - Εμφανίζει μια λεπτομερή περιγραφή των συστήματος και των πόρων που απαρτίζουν ένα container.
- **kill** - Τερματίζει ένα ή περισσότερα container.
- **logs** - Εμφανίζει τα περιεχόμενα των logs ενός container.
- **ls** - Εμφανίζει μια λίστα με τα containers, λειτουργεί με τον ίδιο τρόπο που τρέχει η εντολή docker ps.
- **pause** - Προκαλεί την παύση όλων των διεργασιών που τρέχουν σε ένα ή περισσότερα container.
- **port** - Εμφανίζει μια λίστα με τις θύρες του container που έχουν χαρτογραφηθεί σε θύρες του host.
- **prune** - Αφαιρεί όλα τα container που έχουν σταματήσει να εκτελούνται.
- **rename** - Μετονομάζει ένα container.
- **restart** - Προκαλεί την επανεκκίνηση ενός ή περισσότερων container.
- **rm** - Αφαιρεί ένα ή περισσότερα container.

- **run** - Εκτελεί μια εντολή σε ένα νέο container.
- **start** - Εκκινεί ένα ή περισσότερα σταματημένα container.
- **stats**
  - Εμφανίζει μια ζωντανή ροή των πόρων που χρησιμοποιούνται από τα containers.
- **stop** - Σταματά ένα ή περισσότερα εκτελούμενα container.
- **top** - Εμφανίζει τις διεργασίες που εκτελούνται μέσα σε ένα container.
- **unpause** - Αναιρεί την παύση των διεργασιών που τρέχουν σε ένα container.
- **update** - Ενημερώνει τις παραμέτρους ενός ή περισσότερων container.
- **wait** - Παγώνει την εκτέλεση των εντολών στο τερματικό μέχρι ένα ή περισσότερα container σταματήσουν και μετά τυπώνει τον κωδικό εξόδου τους. Αυτός ο κωδικός περιγράφει την κατάσταση λόγω της οποίας πραγματοποίησε έξοδο ένα container η οποία μπορεί να είναι φυσιολογική ή να προήλθε από ένα σφάλμα (όπου τυπώνει ο κωδικός σφάλματος).

4.4 Docker Machine

Το Docker Machine είναι ένα εργαλείο που επιτρέπει την εγκατάσταση του Docker Engine σε εικονικούς hosts, και επιτρέπει την διαχείρισή τους μέσω των εντολών docker-machine. Με το Docker Machine είναι δυνατή η δημιουργία Docker hosts σε τοπικά Mac ή Windows μηχανήματα, σε εταιρικά δίκτυα, data centers ή σε cloud διακομιστές όπως η AWS ή Digital Ocean.

Οι κύριες λειτουργίες του Docker Machine είναι:

- Η εγκατάσταση και η εκτέλεση του Docker σε Mac και Windows.
- Η διαχείριση πολλαπλών απομακρυσμένων Docker hosts.
- Η διαχείριση Swarm clusters.

Με την χρήση των εντολών docker-machine δίνεται η δυνατότητα για εκκίνηση, επιθεώρηση, επανεκκίνηση και σταμάτημα ενός host, η αναβάθμιση του Docker client και daemon και η διαμόρφωση ενός Docker client έτσι ώστε να επικοινωνεί με τον host.

Το Docker Machine ήταν ο μόνος τρόπος για την λειτουργία του Docker σε Windows ή Mac πριν την έκδοση v1.12 του Docker και πλέον υποστηρίζονται οι εκδόσεις Docker for Windows, Docker for Mac που περιέχουν το Docker Machine και το Docker Compose.

Η χρήση του Docker Machine πλέον προορίζεται για την διαχείριση πολλαπλών απομακρυσμένων Docker hosts σε διάφορες εκδόσεις των Linux καθώς και δίνει την δυνατότητα για χρήση του Docker σε παλαιές εκδόσεις Mac, Windows που δεν υποστηρίζονται από τις εκδόσεις Docker for Windows/Mac που αναφέραμε προηγουμένως.

Διαφορές μεταξύ Docker Engine και Docker Machine

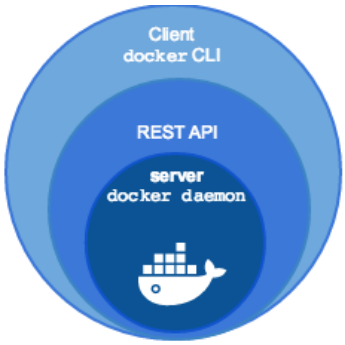

Figure 4.7: Η εσωτερική δομή του Docker Engine (Daemon-API-CLI). [40]

Όταν ο κόσμος αναφέρει το Docker τυπικά εννοεί το Docker Engine που είναι η client-server εφαρμογή αποτελούμενη από το Docker daemon, ένα REST API που προσδιορίζει την διεπαφή με το daemon και ένα CLI (command line interface) το οποίο "μιλάει" με το daemon μέσω του REST API wrapper. Το Docker Engine δέχεται εντολές docker από το CLI, όπως docker run <image>, docker ps για να παρουσιάσει τα ενεργά containers ή docker images για να παρουσιάσει τα ενεργά images.

Το Docker Machine είναι ένα εργαλείο για την διαχείριση των Dockerized hosts (host με εγκατεστημένο το Docker Engine). Τυπικά η εγκατάσταση του Docker Machine γίνεται στο τοπικό μας σύστημα. Το Docker Machine έχει τον δικό του command line client docker-machine και το Docker Engine client docker. Είναι δυνατή η χρήση του Docker machine για την εγκατάσταση του Docker Engine σε ένα ή περισσότερα εικονικά συστήματα. Αυτά τα εικονικά συστήματα μπορούν να λειτουργούν τοπικά ή απομακρυσμένα. Τα Dockerized hosts μπορούν να αναφερθούν ως διαχειριζόμενες μηχανές.

4.5 Docker Compose

Το Docker Compose [42] είναι ένα εργαλείο για τον καθορισμό και την εκτέλεση Docker εφαρμογών με πολλαπλά containers. Με την χρήση ενός αρχείου Compose διαμορφώνουμε τα services της εφαρμογής μας και έπειτα με μια εντολή, δημιουργούμε και ενεργοποιούμε όλες τις υπηρεσίες με βάση την παραμετροποίηση που έχουμε κάνει μέσω του docker-compose.

Το Compose είναι ένα εργαλείο κατάλληλο για περιβάλλοντα ανάπτυξης, δοκιμών και οργάνωσης λογισμικού καθώς και για ροές

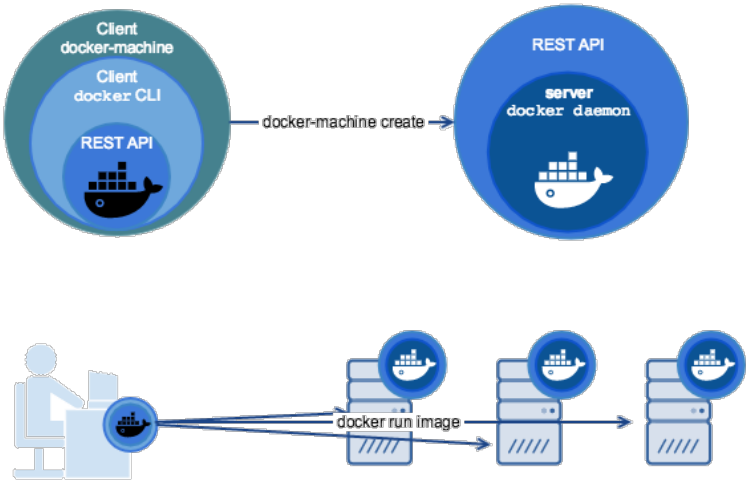

Figure 4.8: Πως λειτουργεί το Docker Machine.
[41]

εργασιών συνεχής ολοκλήρωσης (continuous integration workflows).

Για την χρήση του Docker Compose απαιτούνται 3 βήματα:

- Ορισμός του περιβάλλοντος της εφαρμογής με ένα Dockerfile έτσι ώστε να μπορεί να αναπαραχθεί οπουδήποτε.
- Ορισμός των services που συνθέτουν την εφαρμογή σε ένα docker-compose.yml έτσι ώστε να εκτελούνται μαζί σε ένα απομονωμένο περιβάλλον.
- Τέλος με την εντολή "docker-compose up -d" το Compose ξεκινάει και εκτελεί ολόκληρη την εφαρμογή στο παρασκήνιο.

Ένα αρχείο docker-compose.yml έχει την παρακάτω μορφή:

```
version: "2"

services:
      redis:
    image: redis
            volumes:
              - redis_log:/var/log/redis

volumes:
    - ./code:/code
    - logvolume01:/var/log
```

Στην κορυφή του αρχείου υπάρχει η έκδοση του Docker Compose για την οποία έχει δημιουργηθεί το αρχείο yaml. Έπειτα διακρίνουμε τις εικόνες των services που θα τρέξουν (redis) και κάποια volumes που έχουν δημιουργηθεί για να κρατήσουν αντίγραφα των logs, καθώς και κώδικα (το volume code).

4.6 Docker Swarm

Το Docker Swarm είναι ένα native clustering project για το Docker. Μετατρέπει μία ομάδα (pool) από Docker hosts σε ένα ενιαίο Docker host. Επειδή το Docker Swarm εξυπηρετεί το standard Docker API, κάθε εργαλείο που επικοινωνεί ήδη με το ένα Docker daemon μπορεί να χρησιμοποιήσει το Swarm για την διαφανή κλιμάκωση σε πολλαπλούς hosts.

Τα υποστηριζόμενα εργαλεία περιλαμβάνουν τα εξής:

- Dokku [43]
- Docker Compose
- Docker Machine
- Jenkins [44]

Σημειώνεται πως υποστηρίζεται και το Docker client και όπως και τα υπόλοιπα Docker projects, το Docker Swarm ακολουθεί την αρχή "τοποθέτηση, αλλαγή και άμεση λειτουργία".

4.6.1 Λειτουργία των Swarm clusters

Το πρώτο βήμα για την δημιουργία ενός Swarm cluster στο δίκτυο μας είναι το pull του Docker Swarm image. Έπειτα, χρησιμοποιώντας το Docker, ρυθμίζουμε το Swarm manager και όλα τα nodes να εκτελέσουν το Docker Swarm. Αυτή η μέθοδος απαιτεί τα εξής:

- Άνοιγμα ενός TPC port σε κάθε node για την επικοινωνία με το Swarm Manager.
- Εγκατάσταση του Docker σε κάθε node
- Δημιουργία και διαχείριση πιστοποιητικών TLS για την ασφάλεια του cluster.

Με την χρήση του Docker Machine, μπορούμε να εγκαταστήσουμε το Docker Swarm σε παρόχους cloud ή στο δικό μας data center. Έαν προϋπάρχει εγκατεστημένο το VirtualBox στο τοπικό μας μηχάνημα μπορούμε να κάνουμε build και να εξερευνήσουμε γρήγορα το Docker Swarm στο τοπικό μας περιβάλλον. Αυτή η μέθοδος παράγει αυτόματα ένα πιστοποιητικό για την ασφάλεια του cluster.

Το Docker Swarm με την έκδοση 1.12 του Docker ενσωματώθηκε στο Docker Engine με την μορφή του Swarm Mode για την natively διαχείριση ενός cluster από Docker engines που αποκαλούνται swarm. Είναι δυνατή η χρήση του Docker CLI για την δημιουργία ενός swarm, την ανάπτυξη services εφαρμογών σε ένα swarm και την διαχείριση της συμπεριφοράς του swarm.

4.7 Docker APIs

Τα διαθέσιμα Docker APIs (Application Programming Interfaces) είναι τα παρακάτω:

- Docker Engine API [45].
- Docker Cloud API [46].

4.7.1 Docker Engine API

Το Engine API είναι το API που προσφέρει το Docker Engine και μας δίνει τον έλεγχο του Docker μέσα από τις δικές μας εφαρμογές. Επίσης μας δίνει την δυνατότητα να δημιουργήσουμε εργαλεία για την διαχείριση και επίβλεψη εφαρμογών που λειτουργούν στο Docker, καθώς και να το χρησιμοποιήσουμε για να δημιουργήσουμε εφαρμογές στο ίδιο το Docker.

Το Docker client χρησιμοποιεί το API για να επικοινωνήσει με το Engine, έτσι ότι μπορεί να επιτευχθεί μέσω του Docker client μπορεί να να επιτευχθεί επίσης μέσω του API, για παράδειγμα:

- Εκτέλεση και διαχείριση containers.
- Διαχείριση κόμβων και υπηρεσιών Swarm.
- Πρόσβαση σε αρχεία καταγραφής και στατιστικά στοιχεία.
- Δημιουργία και διαχείριση Swarms.
- Πρόσβαση και διαχείριση σε images.
- Διαχείριση δικτύων και αποθηκευτικών μέσων.

Πρόσβαση στο API μπορεί να αποκτηθεί με οποιοδήποτε HTTP client, αλλά προσφέρονται επίσης SDKs σε Python και Go για να είναι εύκολη η χρήση του και από άλλες γλώσσες προγραμματισμού.

Για παράδειγμα, η εντολη docker run μπορεί να εφαρμοστεί και με διάφορες άλλες γλώσσες προγραμματισμού ή επικοινωνώντας απευθείας με το API μέσω της εντολής curl.

Παράδειγμα χρήσης του API σε Python (τρέχει ένα hello world με την χρήση ενός image που προέρχεται από την διανομή Linux Alpine):

```
import docker
client = docker.from_env()
print client.containers.run(
        "alpine",
        ["echo", "hello", "world"]
)
```

Παράδειγμα δημιουργίας ”hello world” container με την χρήση της εντολής curl:

```
$ curl --unix-socket /var/run/docker.sock \
-H "Content-Type: application/json" \
-d '{"Image": "alpine", "Cmd":["echo","hello world"]}' \
-X POST http:/v1.24/containers/create
```

4.7.2 Docker Cloud API

Το Docker Cloud προσφέρει ένα HTTP REST API και ένα Websocket Stream API τα οποία χρησιμοποιούνται από το Web UI και το CLI.

Για να είμαστε σε θέση να κάνουμε αιτήσεις στο Docker Cloud API, πρέπει πρώτα να αποκτήσουμε ένα API key για τον λογαριασμό μας. Για να το πετύχουμε αυτό κάνουμε Log in στο Docker Cloud και δημιουργούμε ένα API key.

Παράδειγμα χρήσης του Docker Cloud API με την εντολή curl:

```
$ curl -u username:api-key \
https://cloud.docker.com/api/app/v1/service/
```

Το αποτέλεσμα αυτής της εντολής θα δείχνει όπως το παρακάτω:

```
{
"meta": {
"limit":25,
"next":null,
"offset":0,
"previous":null,
"total_count":0
},
"objects":[]
}
```

Όπως φαίνεται και παραπάνω το Docker Cloud REST API είναι προσβάσιμο μέσα από την ακόλουθη διεύθυνση https://cloud.docker.com και όλες οι αιτήσεις προς αυτό απαιτούν πιστοποίηση του χρήστη με την χρήση του API key του. Θα πρέπει να εισάγουμε στα headers του κάθε αιτήματος τουλάχιστον τις ακόλουθες δύο πληροφορίες για να εξυπηρετηθεί το αίτημα μας:

```
Authorization: Basic api-key
Accept: application/json
```

Το πρώτο αφορά την πιστοποίηση μας το API key και το δεύτερο αφορά τον τύπο της πληροφορίας που περιμένουμε ως απάντηση ο οποίος είναι JSON.

Ένα παράδειγμα ενός τέτοιου αιτήματος είναι το παρακάτω:

```
GET /api/app/v1/service/ HTTP/1.1
Host: cloud.docker.com
Authorization: Basic 33d311c3-1e4a-*****
Accept: application/json
```

5. Υλοποίηση δικού μας Docker Image

Θα ξεκινήσουμε να δημιουργήσουμε την δική μας εικόνα (Image) για το Docker, που θα περιέχει μια παραμετροποιημένη εγκατάσταση wordpress έναν εξυπηρετητή για την εμφάνιση της εφαρμογής (apache) και ένα σύστημα βάσης δεδομένων (mysql). Για να το κάνουμε αυτό θα πρέπει να χρησιμοποιήσουμε κάποιο base Image που θα αποτελέσει την βάση του συστήματος μας.

Η επιλογή για το base image είναι το ubuntu 16.04 το οποίο μπορούμε να το βρούμε στο Docker Hub.

Ξεκινάμε την διαδικασία κατεβάζοντας το image:

```
$ docker pull ubuntu:16.04
```

Το αποτέλεσμα θα πρέπει να είναι παρόμοιο με το παρακάτω:

```
16.04: Pulling from library/ubuntu
8aec416115fd: Pull complete
695f074e24e3: Pull complete
946d6c48c2a7: Pull complete
bc7277e579f0: Pull complete
2508cbcde94b: Pull complete
Digest: sha256:71cd81252a3563a03ad...
Status: Downloaded newer image for ubuntu:16.04
```

Με την χρήση της παρακάτω εντολής μπορούμε να επιβεβαιώσουμε την επιτυχή λήψη της εικόνας του ubuntu 16.04:

```
$ docker images | grep ""16.04
ubuntu 16.04 f49eec89601e 13 days ago 129 MB
```

Θα χρησιμοποιήσουμε το παραπάνω image με κάποιες αλλαγές και προσθήκη κάποιων πακέτων λογισμικού για να το ολοκληρώσουμε στην τελική μορφή του. Για τον λόγο αυτό θα χρειαστεί να γράψουμε ένα dockerfile. Το dockerfile περιέχει οδηγίες για την δημιουργία ενός image, τέτοιες οδηγίες συνήθως είναι το base image στο οποίο βασίστηκε το νέο image που θα δημιουργηθεί, καθώς και εντολές που θα πρέπει να εκτελεστούν για την εγκατάσταση πακέτων λογισμικού και την παραμετροποίηση τους. Ξεκινάμε δημιουργώντας έναν νέο κατάλογο στον οποίο θα βάλουμε το αρχείο dockerfile.

```
$ mkdir -p Docker && cd Docker && touch dockerfile
```

Πλέον μέσα στον κάταλογο /home/klipitkas/Documents/Docker υπάρχει ένα κενό αρχείο dockerfile. Το ανοίγουμε για επεξεργασία με κάποιον επεξεργαστή κειμένου όπως το nano ή το vi.

```
$ vim dockerfile
```

Η πρώτη γραμμή και η εντολή **FROM** αφορά το base image που θα βασίσουμε το image μας:

```
FROM ubuntu:16.04
```

Έπειτα μπορούμε να συμπληρώσουμε τα στοιχεία του συντηρητή του συγκεκριμένου docker image:

```
MAINTAINER Lypitkas Konstantinos
<klipitkas@gmail.com>
```

Επίσης δηλώνουμε στο σύστημα σε μια μεταβλητή περιβάλλοντος πως το σύστημα θα εκτελεί τις εντολές χωρίς αλληλεπίδραση με το χρήστη και αυτό θα μας χρειαστεί αργότερα κατά την εγκατάσταση κάποιων πακέτων λογισμικού.

```
ENV DEBIAN_FRONTEND noninteractive
```

Έπειτα έχουμε τις εντολές για τα πακέτα λογισμικού που θα χρειαστούμε τις οποίες εκτελεί το docker με την χρήση της εντολής RUN:

```
RUN apt-get update && apt-get upgrade -y && \
apt-get install -y \
curl \
wget \
apache2 \
php \
libapache2-mod-php \
php-mcrypt \
php-mysql \
supervisor \
debconf-utils
```

Πρώτα θα χρειαστεί να μεταφορτώσουμε στο σύστημα μας τις τελευταίες αλλαγές που υπάρχουν στα επίσημα αποθετήρια κώδικα (official code repositories) του ubuntu 16.04 και τις εγκαθιστούμε στο σύστημα, έπειτα εγκαθιστούμε ό,τι διαθέσιμο πακέτο θα χρειαστούμε στην συνέχεια για να τρέξουμε το wordpress. Αυτά τα πακέτα λογισμικού συμπεριλαμβάνουν τον apache web server και την γλώσσα προγραμματισμού php καθώς και κάποιες απαραίτητες βιβλιοθήκες για να λειτουργεί με τον apache αλλά και την βάση δεδομένων. Επίσης, θα χρειαστούμε ένα πακέτο λογισμικού, το supervisor, το οποίο θα ξεκινά τις υπηρεσίες της βάσης δεδομένων αλλά και τον apache.

Επειδή το docker λειτουργεί με τέτοιο τρόπο έτσι ώστε κανονικά μόνο μια κεντρική διεργασία θα πρέπει να τρέχει σε κάθε container και όταν ολοκληρωθεί αυτή η διεργασία το container σταματά να λειτουργεί αυτόματα αφού η λειτουργεία του είναι ταυτόσημη με την λειτουργία

της διεργασίας. Ωστόσο εμείς θέλουμε να τρέξουμε παραπάνω από μια διεργασίες και αυτές να λειτουργούν αρμονικά μεταξύ τους, για το λόγο αυτό η κεντρική διαδικασία θα είναι ο daemon του supervisor και μέσα από αυτό θα εκκινούνται οι διεργασίες που αφορούν την βάση δεδομένων και τον web server. Έτσι το container θα λειτουργεί όσο η διεργασία του supervisor είναι ενεργή και μόνο όταν αυτή σταματήσει θα πραγματοποιήσει έξοδο το container.

Έπειτα θα χρειαστεί να εγκαταστήσουμε το σύστημα της βάσης δεδομένων, ωστόσο κατά το τέλος της εγκατάστασης πάντα ζητείται από το χρήστη να εισάγει τον κωδικό για τον root χρήστη του MySQL server, κάτι που εμείς θέλουμε να αποφύγουμε αφού κατά την έναρξη του container αυτό θα πρέπει να γίνει αυτόματα. Έτσι πρέπει να ορίσουμε στις ρυθμίσεις του συστήματος που αφορούν τα πακέτα λογισμικού και συγκεκριμένα το mysql-server να πάρει ως κωδικό τον "root" από προεπιλογή.

```
RUN ["/bin/bash", "-c", "debconf-set-selections
\
<<< 'mysql-server mysql-server/root_password \
password root'"]
RUN ["/bin/bash", "-c", "debconf-set-selections \
<<< 'mysql-server mysql-server/root_password_again \
password root'"]
```

Και τώρα μπορούμε να κατεβάσουμε το πακέτο του MySQL server και να πάρει αυτόματα τον κωδικό του root χρήστη.

```
RUN apt-get -y install mysql-server
```

Το επόμενο βήμα είναι να κατεβάσουμε και να εγκαταστήσουμε την τελευταία έκδοση του wordpress.

```
RUN cd /var/www/html && wget \
https://wordpress.org/latest.tar.gz
RUN cd /var/www/html && rm index.html
RUN cd /var/www/html && tar -xf latest.tar.gz
RUN cd /var/www/html && mv wordpress/* . \
&& rm latest.tar.gz
```

Κατεβάζουμε την τελευταία έκδοση του wordpress και την τοποθετούμε στον φάκελο /var/www/html που αποτελεί τον προεπιλεγμένο φάκελο από τον οποίο ο apache εμφανίζει την ιστοσελίδα μας. Σβήνουμε το προεπιλεγμένο αρχείο index.html και αποσυμπιέζουμε το αρχείο του wordpress σε έναν κατάλογο με το ίδιο όνομα μέσα στο /var/www/html. Έπειτα μεταφέρουμε τα αρχεία που βρίσκονται μέσα στον φάκελο /var/www/html/wordpress στον φάκελο /var/www/html και σβήνουμε τον κενό φάκελο wordpress.

Ενεργοποιούμε κάποιες επεκτάσεις του apache που χρειάζονται από το wordpress και αρχίζουμε να δημιουργούμε τα αρχεία παραμετροποίησης του apache για το supervisor και δίνουμε την εντολή μέσω της οποίας το supervisor θα ξεκινά τον apache καθώς και ποιες ακόμη λειτουργίες θα αναλάβει, όπως για παράδειγμα την αυτόματη

έναρξη του apache αλλά και την επανεκκίνηση του σε περίπτωση σφάλματος.

```
RUN a2enmod rewrite #enable mod_rewrite
RUN echo "[program:apache2]" >>
 /etc/supervisor/conf.d/apache2.conf
RUN echo "command=/usr/sbin/apache2ctl -DFOREGROUND" >>
/etc/supervisor/conf.d/apache2.conf
RUN echo "autostart=true" >>
/etc/supervisor/conf.d/apache2.conf
RUN echo "autorestart=true" >>
/etc/supervisor/conf.d/apache2.conf
```

Τα αρχεία .conf που δημιουργήθηκαν θα τα διαβάσει αυτόματα το supervisor κατά την έναρξη του και θα πραγματοποιήσει τις ανάλογες ενέργειες. Παρακάτω θα φροντίσουμε κάποιες ρυθμίσεις που απαιτούνται για να τρέξει ομαλά ο MySQL server.

```
RUN service mysql stop
RUN mkdir -p /var/run/mysqld
RUN chown -R mysql:mysql /var/run/mysqld
```

Για αρχή σταματάμε την διεργασία του MySQL server που ξεκινάει αυτόματα μετά την εγκατάσταση του αφού θέλουμε να εκκινηθεί μέσα από το supervisor και αυτό να αναλάβει όλη την διαχείριση, και έπειτα διορθώνουμε κάποια πιθανά προβλήματα με δικαιώματα πρόσβασης αλλά και φακέλους που μπορεί να μην υπάρχουν κατά την πρώτη εκτέλεση του container.

Συνεχίζουμε με την παραμετροποίηση των αρχείων ρυθμίσεων του supervisor για τον MySQL server, και ορίζουμε το εκτελέσιμο αρχείο το οποίο θα τρέξει ο supervisor καθώς και την αυτόματη εκκίνηση και επανεκκίνηση της υπηρεσίας αλλά και τον χρήστη από τον οποίο θα ξεκινήσει η διεργασία με τα ανάλογα δικαιώματα.

```
RUN echo "[program:mysqld]" >>
/etc/supervisor/conf.d/mysql.conf
RUN echo "command=/usr/bin/mysqld_safe" >>
/etc/supervisor/conf.d/mysql.conf
RUN echo "autostart=true" >>
/etc/supervisor/conf.d/mysql.conf
RUN echo "autorestart=true" >>
/etc/supervisor/conf.d/mysql.conf
RUN echo "user=mysql" >>
/etc/supervisor/conf.d/mysql.conf
```

Θα χρησιμοποιήσουμε την εντολή EXPOSE για να εκθέσουμε κάποιες πόρτες τις οποίες θα μπορεί να χρησιμοποιήσει ο host υπολογιστής ή και άλλα container για να έχουν πρόσβαση. Οι δυο πρώτες θύρες αφορούν τον apache και τα πρωτόκολλα http (80) και https (443) και η επόμενη τον MySQL server (3306) και κυρίως την διαχείριση του από τον host υπολογιστή.

```
EXPOSE 80 443 3306
```

Έπειτα θα ορίσουμε κάποια volumes για να μπορούμε να κρατήσουμε τις ρυθμίσεις που θα πρέπει να παραμένουν στο σύστημα όπως τις αλλαγές στα αρχεία του wordpress καθώς και τις αλλαγές, δημιουργίες βάσεων δεδομένων και εισαγωγή πληροφοριών σε αυτές.

```
VOLUME ["/var/www/html", "/var/mysql"]
```

Τέλος θα ορίσουμε το αρχείο entrypoint.sh που θα εκτελεστεί μετά την εκκίνηση του container και είναι γνωστό στο docker σαν entrypoint το σημείο εισόδου δηλαδή ξεκινά την διεργασία την οποία τρέχει το container μας, το οποίο υπάρχει στον ίδιο φάκελο με το dockerfile αρχείο και αντιγράφεται στο container κατά την διαδικασία του build.

```
COPY entrypoint.sh /usr/local/bin/
RUN ln -s usr/local/bin/entrypoint.sh /entrypoint.sh
RUN chmod +x /usr/local/bin/entrypoint.sh
ENTRYPOINT ["entrypoint.sh"]
```

Τα περιεχόμενα του αρχείου entrypoint.sh είναι τα παρακάτω:

```
#!/bin/bash
set -e
# Since we stopped the service before we want to
# start it now.
service mysql start
# Wait for mysql to start and then execute command
is_mysql_ready() {
        mysqladmin ping --user=root --password=root
}
# We want to execute the create database command only
# when the mysql service is up and running without errors.
while (! is_mysql_ready)
do
sleep 3
done
# Connect and execute the command.
mysql -uroot -proot -e "CREATE DATABASE IF NOT EXISTS wp";
# Stop the service because it should be started by
# the supervisor daemon.
service mysql stop
# Start all the services and run in the foreground.
supervisord -n
```

Μέσα στο αρχείο entrypoint.sh τρέχουμε ουσιαστικά τον κώδικα ο οποίος δημιουργεί μια βάση δεδομένων με το όνομα 'wp' εάν δεν υπάρχει και έπειτα τρέχουμε το supervisor το σύστημα διαχείρισης των υπόλοιπων διεργασιών με την παράμετρο -n που σημαίνει –nodaemon δηλαδή τρέχει η εφαρμογή στο προσκήνιο και με αυτόν τον τρόπο παραμένει ενεργό το container μας και δεν τερματίζει μετά την έναρξη των υπόλοιπων διεργασιών.

Αφού αποθηκεύσουμε τις αλλαγές στο dockerfile θα χρειαστεί να τρέξουμε την παρακάτω εντολή μέσα στον κατάλογο που βρίσκεται το

dockerfile για να δημιουργήσουμε το docker image:

```
klipitkas@foxacid:~/Documents/Docker$ docker build .
```

Αφού δούμε το docker image μέσα στα διαθέσιμα images του συστήματος, μας θα το ονοματίσουμε κατάλληλα έτσι ώστε να μπορέσουμε να το ανεβάσουμε στο δημόσιο αποθετήριο του docker το docker hub. Δίνουμε την εντολή:

```
$ docker tag f142e5f442cd klipitkas/wp-custom:latest
```

Πληκτρολογούμε την παρακάτω εντολή για να συνδεθούμε με τον docker ID λογαριασμό μας στο docker hub.

```
$ docker login
Login with your Docker ID to push and
pull images from Docker Hub.
If you don't have a Docker ID, head over
to https://hub.docker.com to create one.
Username (klipitkas): klipitkas
Password:
Login Succeeded
```

Ανεβάζουμε το νέο docker image χρησιμοποιώντας την παρακάτω εντολή:

```
klipitkas@foxacid:~/Documents/Docker$ docker
push klipitkas/wp-custom
The push refers to a repository
[docker.io/klipitkas/wp-custom]
latest: digest: sha256:bd12ea05f9756f5689c6
13242b7b9a694a9fc423eeecec009dec93f92d4
c7e1f size: 6565
```

Το νέο docker image που δημιουργήσαμε υπάρχει πλέον διαθέσιμο σε όλους στην διεύθυνση https://hub.docker.com/r/klipitkas/wp-custom. Μπορούμε να ξεκινήσουμε ένα νέο container από το image που μόλις δημιουργήσαμε:

```
klipitkas@foxacid:~$ docker run --name custom-wp-container
-p 8080:80 -d klipitkas/wp-custom
```

Figure 5.1: Ανοίγοντας έναν φυλλομετρητή στον host υπολογιστή και πηγαίνοντας στην διεύθυνση http://localhost:8080 βλέπουμε την αρχική οθόνη εγκατάστασης του wordpress.

Figure 5.2: Μπορούμε να συνεχίσουμε την εγκατάσταση εισάγοντας τα παρακάτω στοιχεία:

Figure 5.3: Συμπληρώνουμε τις επιπλέον πληροφορίες που απαιτούνται και έχουμε μια πλήρως λειτουργική εγκατάσταση word-press

6. Επίλογος

Σε αυτή την πτυχιακή εργασία περιγράψαμε τις δύο πτυχές της εικονικοποίησης, τις εικονικές μηχανές και τα containers. Στην περίπτωση των εικονικών μηχανών αναπτύξαμε τις αρχές στις οποίες βασίζονται και αναφέραμε διάφορα τα εργαλεία που είναι διαθέσιμα. Οι εικονικές μηχανές είναι μία ευρέως χρησιμοποιημένη και ανεπτυγμένη τεχνολογία που με την πάροδο του χρόνου έχει περάσει από εκτενές δοκιμές, παρ' όλα ταύτα όμως έχουν αρκετά μειονεκτήματα. Είναι δύσκολες στο στήσιμο και την συντήρηση τους, δεσμεύουν πολύ χώρο στον δίσκο και έχουν προβλήματα με την απόδοση τους παρότι την απευθείας υποστήριξη που έχουν από το hardware.

Η εικονικοποίηση βασισμένη σε containers λειτουργεί με το να μοιράζεται τον πυρήνα του συστήματος εξαλείφοντας τα προβλήματα που αναφέραμε στις εικονικές μηχανές. Ωστόσο είναι σημαντικό να επισημάνουμε ότι τα containers δεν είναι ακόμα σε θέση να αντικαταστήσουν πλήρως τις εικονικές μηχανές. Η εκτέλεση ενός τελείως διαφορετικού λειτουργικού συστήματος στον guest είναι μία δημοφιλής λειτουργία των εικονικών μηχανών που δεν υποστηρίζεται από τα containers. Επιπλέον όσο περισσότερο χρησιμοποιούνται τα containers τόσα περισσότερα κενά ασφαλείας θα ανακαλύπτονται στις επιμέρους τεχνολογίες και ενώ επιδιορθώνονται αρκετά γρήγορα, θα περάσει αρκετός χρόνος μέχρι τα containers να φτάσουν την αξιοπιστία και ασφάλεια των εικονικών μηχανών. Ακόμα και έτσι όμως αρκετές μεγάλες εταιρίες έχουν αποφασίσει ότι τα containers μπορούν να βελτιώσουν δραματικά την απόδοση και την προσβασιμότητα των cloud τεχνολογιών τους και έχουν αρχίσει να τα υιοθετούν

Το Docker είναι ένα project το οποίο φέρνει τα containers ποιο κοντά στους προγραμματιστές και στους διαχειριστές υπολογιστικών συστημάτων. Προσφέρει ένα εύκολο στην χρήση περιβάλλον διεπαφής για την διαχείριση των containers καθώς και ένα υγιές οικοσύστημα για τον διαμοιρασμό containerized εφαρμογών - images. Το Docker είναι στο δρόμο για να γίνει πρότυπο διαχείρισης containers, ενώ εισάγει υψηλού επιπέδου εργαλεία για εφαρμογές και χρήστες. Το Docker αναπτύσσεται με ραγδαίους ρυθμούς προσθέτοντας καινούργιες λειτουργίες ενώ παράλληλα προσπαθεί να διατηρήσει μία διαρκής σχέση με την βάση χρηστών του, έτσι ώστε να είναι σίγουρο ότι τα containers θα φτάσουν κάποια στιγμή την φήμη των εικονικών μηχανών. Επιπλέον, πληθώρα από project δημιουργούνται με βάση το Docker, και κυμαίνονται από απλές επεκτάσεις μέχρι ολόκληρα cloud-based λειτουργικά συστήματα.

Το πρακτικό κομμάτι της πτυχιακής μας αναδεικνύει την πραγματική χρησιμότητα του Docker μέσω της δημιουργίας ενός εξατομικευμένου image με μια πλήρως λειτουργική εγκατάσταση wordpress που μπορεί χρησιμοποιηθεί σε πληθώρα από περιβάλλοντα χωρίς την ανάγκη περαιτέρω ρυθμίσεων, αποφεύγοντας το περιττό overhead των εικονικών μηχανών.

7. Βιβλιογραφία

[1] **Amazon AWS**: *Types of cloud computing*.
https://aws.amazon.com/types-of-cloud-computing

[2] **David Kaplan**, *Protecting VM Register State with SEV-ES*, 2017.
http://support.amd.com/TechDocs/Protecting-VM-Register-State-with-20SEV-ES.pdf

[3] **VMWare Inc.**, *Performance Evaluation of Intel EPT Hardware Assist*, 2008-2009.
http://www.vmware.com/pdf/Perf_ESX_Intel-EPT-eval.pdf

[4] **D. Padua**, *Encyclopedia of Parallel Computing*, 2011.

[5] **Oracle Corporation**, *VirtualBox technical documentation*.
https://www.virtualbox.org/wiki/VirtualBox

[6] **VMWare Inc.**, *VMWare player*.
https://www.vmware.com/products/player

[7] **QEmu contributors, Linux Foundation** *QEmu Wiki documentation*, 2017. https://wiki.qemu.org/Main_Page

[8] **KVM contributors**, *KVM Wiki documentation*, November 2016.
https://www.linux-kvm.org/page/Main_Page

[9] **Xen Project contributors**, *Xen Project Software Overview*, March 2016. https://wiki.xen.org/wiki/Xen_Project_Software_Overview

[10] **Amazon Web Services, Inc**, *Amazon Web Services (AWS) - Cloud Computing Services*, July 2017. https://aws.amazon.com

[11] **Song, Yang and Wang, Haoliang and Soyata, Tolga**, *Hardware and Software Aspects of VM-Based Mobile-Cloud Offloading*, August 2015.

[12] **Oracle Inc**, *Best Practices for Running Oracle Databases in Oracle Solaris Containers*, June 2011.
http://www.oracle.com/technetwork/server-storage/solaris/documentation/oracle-containers-163576.pdf

[13] **Parallels International GMBH**, *OpenVZ Main Wiki*, November 2016. https://openvz.org/Main_Page

[14] **Riondato M.**, *Chapter 14, Jails*.
https://www.freebsd.org/doc/handbook/jails.html

[15] **Canonical Ltd.**, *Introduction to LXC*.
https://linuxcontainers.org/lxc/introduction

[16] **The Linux Documentation Project contributors**, *6.4 System V IPC*. http://www.tldp.org/LDP/lpg/node21.html

[17] **Libvirt project contributors**, *Introduction*. http://libvirt.org

[18] **Babak Bashari Rad, Harrison John Bhatti, Mohammad Ahmadi**, *An Introduction to Docker and Analysis of its Performance*, March 2017.

[19] **Boot2docker contributors**, *boot2docker*, June 2014.
http://boot2docker.io

[20] **Docker Inc.**, *Docker 0.9: introducing execution drivers and libcontainer*, March 2014. https://blog.docker.com/2014/03/docker-0-9-introducing-execution-drivers-and-libcontainer

[21] **Prabhaker Mateti**, *File Systems*.
http://cecs.wright.edu/ pmateti/Courses/3900/Lectures/Internals/file-systems.html

[22] **Docker Inc.**, *Use volumes*.
https://docs.docker.com/engine/admin/volumes/volumes

[23] **Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio**, *An Updated Performance Comparison of Virtual Machines and Linux Containers*, July 21, 2014.

[24] **The Kubernetes Authors**, *What is Kubernetes?*.
https://kubernetes.io/docs/concepts/overview/what-is-kubernetes

[25] **CoreOS, Inc**, *Rkt Project Overview*. https://coreos.com/rkt

[26] **Nine Internet Solutions AG**, *Manage the Minions with Kubernetes*, February 2016.
https://blog.nine.ch/en/2016/02/01/manage-the-minions-with-kubernetes

[27] **Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, Ion Stoica**, *Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center*, September 2010.

[28] **Mesosphere, Inc.** *Architecture*, July 2017.
https://docs.mesosphere.com/1.9/overview/architecture

[29] **Alexander Rukletsov, Jörg Schad** *Apache Mesos (In a Nutshell)*, 2015. http://mesosphere.github.io/presentations/mesos-nutshell-2015

[30] **CoreOS, Inc**, *About CoreOS*. https://coreos.com/about

[31] **CoreOS, Inc**, *CoreOS Container Linux cluster architectures*.
https://coreos.com/os/docs/latest/cluster-architectures.html

[32] **Canonical Ltd.** *A 'snap' is a universal Linux package*, June 2017.
https://www.ubuntu.com/desktop/snappy

[33] **Canonical Ltd.**, *Introduction to LXD*.
https://linuxcontainers.org/lxd/

[34] **Red Hat, Inc**, *Introduction to Project Atomic*, 2014-2017.
http://www.projectatomic.io/docs/introduction

[35] **Alex Polvi**, *CoreOS is building a container runtime, rkt*, December 2014. https://coreos.com/blog/rocket.html

[36] **Ubuntu wiki contributors**, *UncomplicatedFirewall*.
https://wiki.ubuntu.com/UncomplicatedFirewall

[37] **Docker Inc**, *Docker Registry*. https://docs.docker.com/registry

[38] **Docker Inc**, *Docker Hub*, 2016. https://hub.docker.com

[39] **Docker Inc**, *Dockerfile reference*.
https://docs.docker.com/engine/reference/builder

[40] **Docker Inc**, *What's the difference between Docker Engine and Docker Machine?*. https://docs.docker.com/machine/overview/#whats-the-difference-between-docker-engine-and-docker-machine

[41] **Docker Inc**, *Docker Machine Overview*.
https://docs.docker.com/machine/overview

[42] **Docker Inc**, *Overview of Docker Compose*.
https://docs.docker.com/compose/overview

[43] **Jeff Lindsay**, *Getting Started with Dokku*.
http://dokku.viewdocs.io/dokku/getting-started/installation

[44] **Software in the Public Interest, Inc.**, *Jenkins Documentation*.
https://jenkins.io/doc

[45] **Docker Inc**, *Docker Engine API and SDKs*.
https://docs.docker.com/engine/api

[46] **Docker Inc**, *Introduction*
https://docs.docker.com/apidocs/docker-cloud