

ΕΡΓΑΣΙΑ ΣΤΟ ΜΑΘΗΜΑ
«ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΛΟΓΙΣΜΙΚΟΥ»

Docker: Δημιουργία containers για ανάπτυξη
εφαρμογών

ΑΠΟΣΤΟΛΟΣ ΠΕΣΛΗΣ

Περιεχόμενα

Περίληψη.....	3
1 Εισαγωγή	4
2 Θεωρητική προσέγγιση του Docker	5
2.1 Ορισμός και σκοπός	5
2.2 Η έννοια των containers	5
2.3 Σύγκριση Docker με Εικονικές Μηχανές.....	6
2.4 Αρχιτεκτονική του Docker	7
2.5 Πώς λειτουργεί το Docker στον κύκλο ζωής μιας εφαρμογής.....	9
3 Docker Tutorial	10
3.1 Τα βασικά του Docker	11
3.2 Προετοιμασία αρχείων docker εφαρμογής.....	15
4 Συμπέρασμα	26
Βιβλιογραφία.....	27
Παράρτημα	28

Περίληψη

Η εργασία εστιάζει στη μελέτη και εφαρμογή του Docker, μιας πλατφόρμας containerization που έχει αναδειχθεί σε βασικό εργαλείο για την ανάπτυξη σύγχρονων εφαρμογών λογισμικού. Παρουσιάζεται η θεωρητική βάση του Docker, συμπεριλαμβανομένης της αρχιτεκτονικής του, των βασικών συστατικών (Docker Engine, Images, Containers, Dockerfile, Docker Hub) και της σύγκρισης με τις εικονικές μηχανές, αναδεικνύοντας τα πλεονεκτήματα της ελαφριάς και φορητής φύσης του.

Επιπλέον, περιγράφονται οι βασικές εντολές και η διαδικασία δημιουργίας containers για την ανάπτυξη μιας εφαρμογής με frontend (PHP/Apache), backend και βάση δεδομένων (MySQL), χρησιμοποιώντας αρχεία όπως το .env, το Dockerfile και το docker-compose.yml. Η πρακτική υλοποίηση καταδεικνύει την ικανότητα του Docker να εξασφαλίζει συνέπεια και αποδοτικότητα σε όλα τα στάδια του κύκλου ζωής μιας εφαρμογής. Τα αποτελέσματα υπογραμμίζουν την ευελιξία και την αξιοπιστία του Docker, καθώς και τις προκλήσεις που σχετίζονται με τη διαχείριση πολύπλοκων περιβαλλόντων.

Συμπερασματικά, το Docker αποτελεί μια κρίσιμη τεχνολογία για τη σύγχρονη αρχιτεκτονική λογισμικού, υποστηρίζοντας τις ανάγκες των προγραμματιστών για ταχύτητα, φορητότητα και σταθερότητα.

Λέξεις – κλειδιά: Docker, Containerization, Containers, Εικονικοποίηση, Docker Engine, Dockerfile, Docker Compose, Microservices, DevOps, Cloud Computing, Αρχιτεκτονική Λογισμικού, Φορητότητα, CI/CD, MySQL, PHP.

1 Εισαγωγή

Στην εποχή της ταχύτατης εξέλιξης της τεχνολογίας και της συνεχούς ανάγκης για ευέλικτες και αποδοτικές λύσεις στην ανάπτυξη λογισμικού, η έννοια της εικονικοποίησης (virtualization) έχει αναδειχθεί σε κρίσιμο εργαλείο για την υποστήριξη πολύπλοκων εφαρμογών. Η παραδοσιακή προσέγγιση με εικονικές μηχανές (Virtual Machines - VMs) παρείχε λύσεις σε ζητήματα απομόνωσης και επεκτασιμότητας, ωστόσο συνοδευόταν από υψηλές απαιτήσεις σε πόρους και δυσκολίες στην ενοποίηση των περιβαλλόντων ανάπτυξης και παραγωγής.

Η ανάγκη για πιο ευέλικτες, ελαφριές και φορητές λύσεις οδήγησε στην εμφάνιση των τεχνολογιών containerization, με το Docker να αποτελεί την πιο διαδεδομένη και ισχυρή πλατφόρμα αυτής της προσέγγισης. Το Docker επιτρέπει τη δημιουργία, διανομή και εκτέλεση εφαρμογών μέσα σε containers, τα οποία είναι απομονωμένα περιβάλλοντα που περιλαμβάνουν όλα τα απαραίτητα για τη λειτουργία της εφαρμογής (κώδικα, εξαρτήσεις, βιβλιοθήκες κ.ά.). Αυτή η προσέγγιση εξασφαλίζει συνέπεια σε όλα τα στάδια της ανάπτυξης λογισμικού και μειώνει τα σφάλματα που σχετίζονται με ασυμβατότητες μεταξύ περιβαλλόντων.

Στο πλαίσιο της παρούσας εργασίας, μελετάται εις βάθος το Docker, τόσο θεωρητικά όσο και πρακτικά. Αρχικά παρουσιάζεται η λειτουργία και η αρχιτεκτονική του, τα πλεονεκτήματα και τα μειονεκτήματά του, καθώς και η χρήση των βασικών εντολών που το συνοδεύουν. Στη συνέχεια, γίνεται εφαρμογή των εννοιών αυτών μέσω της υλοποίησης μιας απλής εφαρμογής με χρήση containers, η οποία περιλαμβάνει frontend, backend και βάση δεδομένων, επιτρέποντας την κατανόηση του τρόπου λειτουργίας του Docker σε ένα περιβάλλον πολλαπλών υπηρεσιών.

Στόχος της εργασίας είναι η κατανόηση των βασικών αρχών του Docker, η εξοικείωση με τη δημιουργία και διαχείριση containers, και η πρακτική εφαρμογή των θεωρητικών γνώσεων στην ανάπτυξη μιας σύγχρονης εφαρμογής λογισμικού. Μέσα από αυτή τη διαδικασία, επιχειρείται να αναδειχθούν τα οφέλη και οι προκλήσεις της τεχνολογίας containerization στην αρχιτεκτονική λογισμικού.

2 Θεωρητική προσέγγιση του Docker

2.1 Ορισμός και σκοπός

Το Docker είναι μία ανοιχτού κώδικα πλατφόρμα που επιτρέπει την αυτοματοποιημένη δημιουργία, ανάπτυξη και εκτέλεση εφαρμογών μέσα σε απομονωμένα περιβάλλοντα γνωστά ως containers. Τα containers αποτελούν ελαφριά, φορητά και απομονωμένα συστήματα που περιλαμβάνουν όλα τα απαραίτητα στοιχεία για την εκτέλεση μίας εφαρμογής, όπως ο πηγαίος κώδικας, βιβλιοθήκες, εξαρτήσεις και παραμετροποιήσεις.

Η κύρια αποστολή του Docker είναι να προσφέρει έναν τρόπο για τη συσκευασία (packaging) και διανομή λογισμικού, διασφαλίζοντας ότι η εφαρμογή θα εκτελείται με τον ίδιο τρόπο ανεξαρτήτως περιβάλλοντος. Αυτή η προσέγγιση λύνει ένα από τα πιο συχνά προβλήματα στην ανάπτυξη λογισμικού, γνωστό ως “it works on my machine” φαινόμενο [1].

Το Docker δημιουργήθηκε από τον Solomon Hykes και παρουσιάστηκε για πρώτη φορά το 2013 ως εσωτερικό εργαλείο της εταιρείας dotCloud. Από τότε έχει εξελιχθεί σε μία από τις βασικότερες τεχνολογίες στον χώρο του DevOps, του cloud computing και της σύγχρονης αρχιτεκτονικής λογισμικού [2].

2.2 Η έννοια των containers

Τα containers είναι απομονωμένα περιβάλλοντα εκτέλεσης εφαρμογών, τα οποία χρησιμοποιούν τον πυρήνα του λειτουργικού συστήματος του host (κοινή χρήση του kernel), αλλά διατηρούν πλήρη απομόνωση σε επίπεδο αρχείων συστήματος, βιβλιοθηκών και διεργασιών.

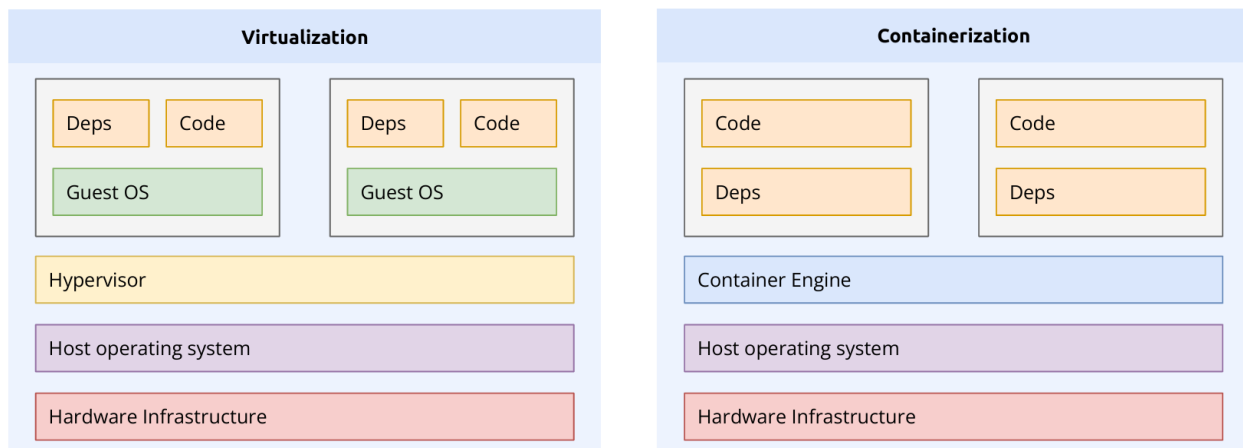
Αυτό τα καθιστά σημαντικά πιο ελαφριά και γρήγορα σε σύγκριση με τις εικονικές μηχανές. Η δημιουργία και εκκίνηση ενός container μπορεί να γίνει μέσα σε δευτερόλεπτα, καθώς δεν απαιτείται φόρτωση πλήρους λειτουργικού συστήματος.

Η λειτουργία των containers βασίζεται σε τεχνολογίες του Linux όπως τα namespaces και τα cgroups, οι οποίες επιτρέπουν την απομόνωση πόρων και διεργασιών, ενώ διατηρείται η αποτελεσματική χρήση του hardware [3].

2.3 Σύγκριση Docker με Εικονικές Μηχανές

Παραδοσιακά, η ανάπτυξη εφαρμογών σε διαφορετικά περιβάλλοντα γινόταν με τη χρήση εικονικών μηχανών (VMs). Παρότι οι VMs προσφέρουν πλήρη απομόνωση μέσω της εξομοίωσης ενός ολόκληρου λειτουργικού συστήματος, συνοδεύονται από υψηλό υπολογιστικό κόστος, καθυστερήσεις στην εκκίνηση, και δυσκολίες στη διανομή [4].

Ο παρακάτω πίνακας συνοψίζει τις βασικές διαφορές:



Εικόνα 1 - Διαφορά Virtualization και Containerization [4]

Στα αριστερά όπου παρουσιάζεται η εικονικοποίηση δύο εφαρμογών σε έναν υπολογιστή με την χρήση των εικονικών συσκευών, παρατηρούμε ότι κάθε μία από τις εικονικές μηχανές απαιτεί ξεχωριστή εγκατάσταση λειτουργικού συστήματος. Αντιθέτως, στην δεξιά αρχιτεκτονική που χρησιμοποιεί το Docker, παρατηρείται ότι κάθε container που περιέχει μία εφαρμογή χρειάζεται μόνο τις εξαρτήσεις και τον κώδικά της.

Το Docker προσφέρει μία πιο αποδοτική προσέγγιση, ειδικά για εφαρμογές μικροϋπηρεσιών (microservices) και σύγχρονα DevOps pipelines.

Μία ακόμα συγκριτική ανάλυση μεταξύ των Εικονικών Μηχανών και των Docker Containers είναι στην σύγκρισή τους στις εξής βασικές πτυχές:: την απομόνωση (Isolation), το μέγεθος/υπερκάλυψη (Size/Overhead), τη φορητότητα (Portability) και τις προτεινόμενες περιπτώσεις χρήσης (When to use). Οι Εικονικές Μηχανές προσφέρουν ισχυρή απομόνωση, καθώς κάθε VM διαθέτει το δικό της λειτουργικό σύστημα, αλλά έχουν μεγαλύτερο μέγεθος λόγω της ανάγκης για πλήρη virtualized hardware και guest OS. Αντιθέτως, τα Docker Containers παρέχουν απομόνωση σε επίπεδο διεργασιών, μοιράζονται τον πυρήνα του host OS, και είναι πιο ελαφριά, με ελάχιστη υπερκάλυψη. Επιπλέον, τα Containers ξεχωρίζουν για την υψηλή τους φορητότητα, καθώς είναι ανεξάρτητα από την πλατφόρμα και εκτελούνται με συνέπεια σε διαφορετικά περιβάλλοντα, ενώ οι VMs είναι λιγότερο φορητές λόγω της εξάρτησής τους από συγκεκριμένους hypervisors και διαμορφώσεις OS. Όσον αφορά τις περιπτώσεις χρήσης, οι VMs συνιστώνται για εφαρμογές που απαιτούν ισχυρή απομόνωση ή όταν

αντιμετωπίζονται παλαιότερες εφαρμογές που δεν μπορούν εύκολα να γίνουν containerized, καθώς και για περιβάλλοντα δοκιμών ή ανάπτυξης που απαιτούν πλήρη συστήματα. Αντιθέτως, τα Containers προτείνονται για σύγχρονες, cloud-native εφαρμογές μικροϋπηρεσιών (microservices), όπου η γρήγορη κλιμάκωση και η αποδοτικότητα αποτελούν προτεραιότητα. Η παρακάτω εικόνα αποτελεί ένα χρήσιμο εργαλείο για την κατανόηση των διαφορών και των κατάλληλων εφαρμογών των δύο τεχνολογιών στο πλαίσιο της διαχείρισης υποδομών πληροφορικής [4].

Feature	Virtual Machines (VMs)	Docker Containers
Isolation	Strong isolation: Each VM has its own OS, providing complete isolation.	Process-level isolation: Containers share the host OS kernel.
Size/Overhead	Larger: VMs have a larger footprint due to the guest OS and virtual hardware.	Lightweight: Containers have minimal overhead, as they share the kernel.
Portability	Less portable: VMs can be tied to specific hypervisors and guest OS configurations.	Highly portable: Containers are platform-agnostic and run consistently.
When to use	<ul style="list-style-type: none"> You need strong isolation between different environments. You're dealing with legacy applications that might not be easily containerized. You want to replicate a complete system environment for testing or development. 	<ul style="list-style-type: none"> You're building modern, cloud-native applications using microservices architecture. You need to scale your applications quickly and efficiently. Portability across different environments is a top priority.

Εικόνα 2 - Βασικές πτυχές σύγκρισης VMs – Containers [4]

2.4 Αρχιτεκτονική του Docker

Η αρχιτεκτονική του Docker βασίζεται στα εξής βασικά συστατικά:

Docker Engine

Πρόκειται για τη βασική μηχανή που επιτρέπει τη δημιουργία και διαχείριση containers. Αποτελείται από:

- Docker Client: Δέχεται εντολές από τον χρήστη (π.χ. docker build, docker run) και τις προωθεί στον daemon.
- Docker Daemon: Υλοποιεί τις εντολές και διαχειρίζεται images, containers, δίκτυα και volumes.
- REST API: Επιτρέπει την επικοινωνία μεταξύ client και daemon.

Docker Image

Είναι ένα στιγμιότυπο μόνο για ανάγνωση (read-only) που περιλαμβάνει όλα τα απαραίτητα για την εκτέλεση της εφαρμογής. Τα images είναι πολύ-επίπεδα, γεγονός που επιτρέπει την επαναχρησιμοποίηση και αποδοτική αποθήκευση.

Docker Container

Είναι μια εκτελέσιμη διεργασία ενός Docker image. Ο container δημιουργείται από το image και εκτελείται ως απομονωμένη διεργασία με δικό του σύστημα αρχείων, δίκτυο και πόρους.

Dockerfile

Αποτελεί αρχείο οδηγιών για την αυτόματη δημιουργία ενός image. Περιγράφει βήμα προς βήμα τι χρειάζεται να εγκατασταθεί ή να ρυθμιστεί (π.χ. ποια έκδοση Node.js, ποιος φάκελος να αντιγραφεί κ.λπ.).

Στην παρακάτω εικόνα, παρουσιάζεται ένα Dockerfile το οποίο χρησιμοποιεί ως βασική εικόνα, από το Docker Hub, την έκδοση 8.2 της PHP και έναν Apache web server που έχει ρυθμιστεί να εκτελεί εφαρμογές PHP και επιπλέον εκτελεί μία εντολή κατά την διαδικασία δημιουργίας του image εγκαθιστά δύο επεκτάσεις, το pdo και το pdo_mysql για διεπαφή με βάσεις δεδομένων και συγκεκριμένα με βάσεις δεδομένων σε MySQL.

```
1 FROM php:8.2-apache
2
3 RUN docker-php-ext-install pdo pdo_mysql
```

Εικόνα 3 - Dockerfile

Docker Hub / Registry

Αποτελεί τον "χώρο αποθήκευσης" των images. Το Docker Hub είναι η επίσημη δημόσια registry, αλλά μπορούν να δημιουργηθούν και ιδιωτικά registries για επιχειρησιακή χρήση [5].

2.5 Πώς λειτουργεί το Docker στον κύκλο ζωής μιας εφαρμογής

Το Docker ενσωματώνεται ιδανικά στον κύκλο ζωής ανάπτυξης λογισμικού, από την ανάπτυξη (development) έως την παραγωγή (production), υποστηρίζοντας μοντέρνες πρακτικές όπως το Continuous Integration/Continuous Deployment (CI/CD).

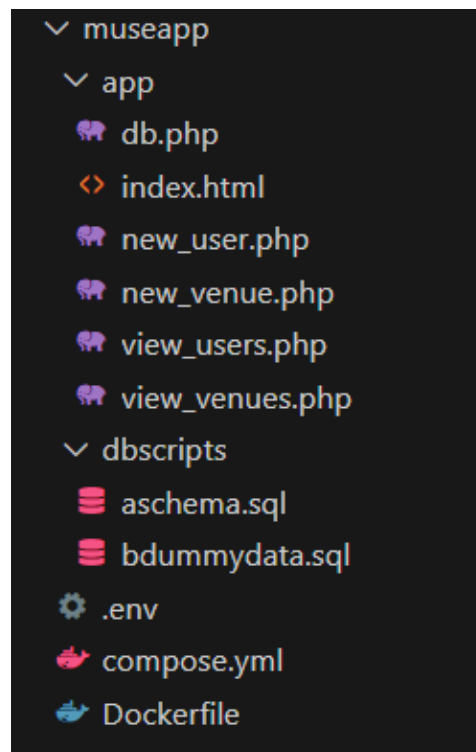
Οι προγραμματιστές μπορούν να δημιουργούν και να τεστάρουν εφαρμογές τοπικά σε containers, διασφαλίζοντας ότι το περιβάλλον εκτέλεσης είναι ακριβώς το ίδιο με αυτό που θα χρησιμοποιηθεί στην παραγωγή. Αυτό μειώνει τα σφάλματα, διευκολύνει την ομαδική εργασία και επιταχύνει τον κύκλο ανάπτυξης.

Επιπλέον, το Docker υποστηρίζει τη φιλοσοφία “Infrastructure as Code”, μέσω αρχείων όπως το Dockerfile και το docker-compose.yml, καθιστώντας τον καθορισμό και την επανάληψη περιβαλλόντων εύκολο και διαφανή [6].

3 Docker Tutorial

Αφού έχουν αναλυθεί οι λόγοι για τους οποίους είναι σημαντική η χρήση των Docker containers, στην συνέχεια της εργασίας θα παρουσιαστεί ένα παράδειγμα για το πως μπορεί να δημιουργηθούν containers για την ανάπτυξη μίας εφαρμογής.

Τα περιεχόμενα των αρχείων της λειτουργικότητας της εφαρμογής αναλύονται στο [παράρτημα](#) της παρούσας εργασίας, ώστε να γίνει αντιληπτή. Παρακάτω ακολουθεί η δομή του φακέλου της εφαρμογής:



Συγκεκριμένα, εντός του φακέλου της εφαρμογής υπάρχουν τα αρχεία `.env` `compose.yml` και `Dockerfile` που αναλύονται στην εργασία. Επιπλέον εντός του φακέλου της εφαρμογής υπάρχουν οι υποφάκελοι `app` και `dbscripts`. Στον φάκελο `app` περιέχονται όλα τα απαραίτητα αρχεία της εφαρμογής τα οποία αναλύονται στο [παράρτημα](#) της εργασίας. Στον φάκελο `dbscripts` περιέχονται δύο αρχεία `sql`, τα οποία, το ένα δημιουργεί το σχήμα της βάσης και τους απαραίτητους πίνακες και το δεύτερο έχει κάποια δοκιμαστικά δεδομένα για την παρουσίαση της εφαρμογής.

3.1 Τα βασικά του Docker

Σε αυτό το σημείο θα παρουσιαστούν βασικές εντολές στο docker.

Αφού ξεκινήσουμε το Docker engine στον υπολογιστή μας και ανοίξουμε το τερματικό μπορούμε να πάρουμε πληροφορίες τις παρακάτω εντολές.

docker version ή docker -v: Μας δίνει πληροφορίες με την έκδοση του docker client και του docker engine που χρησιμοποιούμε.

```
PS C:\Users\apostolos> docker version
Client:
Version:      28.1.1
API version:  1.49
Go version:   go1.23.8
Git commit:   4eba377
Built:        Fri Apr 18 09:53:24 2025
OS/Arch:      windows/amd64
Context:      desktop-linux

Server: Docker Desktop 4.41.2 (191736)
Engine:
Version:      28.1.1
API version:  1.49 (minimum version 1.24)
Go version:   go1.23.8
Git commit:   01f442b
Built:        Fri Apr 18 09:52:57 2025
OS/Arch:      linux/amd64
Experimental: false
containers:
```

docker-compose version ή docker-compose -v: Μας δίνει πληροφορίες με την έκδοση του docker-compose που χρησιμοποιούμε.

```
PS C:\Users\apostolos> docker-compose -v
Docker Compose version v2.35.1-desktop.1
```

docker --help: Μας δίνει όλες τις διαθέσιμες εντολές που μπορούμε να εκτελέσουμε με την περιγραφή τους.

```

PS C:\Users\apostolos> docker --help
Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:
  run       Create and run a new container from an image
  exec      Execute a command in a running container
  ps        List containers
  build     Build an image from a Dockerfile
  bake      Build from a file
  pull      Download an image from a registry
  push      Upload an image to a registry
  images    List images
  login     Authenticate to a registry
  logout    Log out from a registry
  search    Search Docker Hub for images
  version   Show the Docker version information
  info      Display system-wide information

```

docker ps: Μας εμφανίζει όλα τα container που είναι σε λειτουργία.

docker ps -a: Μας εμφανίζει όλα τα container, ακόμα και αν δεν λειτουργούν.

```

PS C:\Users\apostolos> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
PS C:\Users\apostolos>
PS C:\Users\apostolos>
PS C:\Users\apostolos> docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
88089f2c97fd   erasmapp-web   "docker-php-entrypoi..."   24 hours ago   Exited (0) 23 hours ago   erasmapp_web
13093fcaa4be   mysql:8.0     "docker-entrypoint.s..."   24 hours ago   Exited (0) 23 hours ago   erasmapp_db

```

docker start <container_id/name>: Θέτει σε λειτουργία το container που υποδεικνύουμε.

```

PS C:\Users\apostolos> docker start 88089f2c97fd
88089f2c97fd
PS C:\Users\apostolos> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
88089f2c97fd   erasmapp-web   "docker-php-entrypoi..."   24 hours ago   Up 7 seconds   0.0.0.0:8080->80/tcp   erasmapp_web

```

docker stop <container_id/name>: Τερματίζει την λειτουργία του container που υποδεικνύουμε, τερματίζοντας σταδιακά όλα τα services.

```

PS C:\Users\apostolos> docker stop 88089f2c97fd
88089f2c97fd
PS C:\Users\apostolos> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES

```

`docker kill <container_id/name>`: Τερματίζει την λειτουργία του container με βίαιο τρόπο.

```
PS C:\Users\apostolos> docker kill competent_khayyam
competent_khayyam
PS C:\Users\apostolos> docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
bbd10c798bcd	nginx:latest	"/docker-entrypoint..."	About a minute ago	Exited (137) 8 seconds ago		competent_khayyam
e781ca6dcca5	nginx:latest	"/docker-entrypoint..."	5 minutes ago	Exited (0) About a minute ago		serene_feynman

`docker run <image:tag>`: Δημιουργεί και ξεκινάει ένα container βασισμένο σε nginx. Αν δεν βάλουμε το flag `-d`, τότε στο terminal που εκτελέσαμε την εντολή, θα ξεκινήσουν να τρέχουν τα services του container. Με το flag `-d`, τρέχει σε detached mode, οπότε μπορούμε να συνεχίσουμε στο ίδιο terminal, γιατί το container τρέχει στο background.

```
PS C:\Users\apostolos> docker run nginx:latest
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
```

```
PS C:\Users\apostolos> docker run -d nginx:latest
bbd10c798bcd4e4880d9864ab31b131e8ff2d11da160a6ecaa6fffc9384636a
PS C:\Users\apostolos> docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
bbd10c798bcd	nginx:latest	"/docker-entrypoint..."	9 seconds ago	Up 9 seconds	80/tcp	competent_khayyam

```
PS C:\Users\apostolos>
```

`docker rm <container_id/name>`: Διαγράφει το container που υποδεικνύουμε εάν δεν είναι σε λειτουργία.

```
bbd10c798bcd  nginx:latest  "/docker-entrypoint..."  6 minutes ago  Exited (137) 5 minutes ago  competent_khayyam
e781ca6dcca5  nginx:latest  "/docker-entrypoint..."  10 minutes ago  Up 16 seconds              80/tcp serene_feynman
PS C:\Users\apostolos> docker rm serene_feynman
Error response from daemon: cannot remove container "/serene_feynman": container is running: stop the container before removing or force remove
PS C:\Users\apostolos> docker rm competent_khayyam
competent_khayyam
PS C:\Users\apostolos> docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e781ca6dcca5	nginx:latest	"/docker-entrypoint..."	10 minutes ago	Up 49 seconds	80/tcp	serene_feynman

```
PS C:\Users\apostolos>
```

`docker rm -f <container_id/name>`: Διαγράφει το container που υποδεικνύουμε με force τρόπο, ακόμα και αν είναι σε λειτουργία.

```
PS C:\Users\apostolos> docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e781ca6dcca5	nginx:latest	"/docker-entrypoint..."	12 minutes ago	Up 2 minutes	80/tcp	serene_feynman

```
PS C:\Users\apostolos> docker rm -f serene_feynman
serene_feynman
PS C:\Users\apostolos> docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

```
PS C:\Users\apostolos>
```

docker image ls: Μας εμφανίζει όλα τα image που έχουμε αποθηκευμένα στο docker engine του υπολογιστή μας.

```
PS C:\Users\apostolos> docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
erasmapp-web	latest	0935b2e7c0c7	27 hours ago	709MB
nginx	latest	c15da6c91de8	5 weeks ago	279MB
mysql	8.0	51d7ec709cde	5 weeks ago	1.06GB
mysql	5.7	4bc6bc963e6d	17 months ago	689MB

docker rmi <image_name/image_id>: Διαγράφει την εικόνα που του υποδεικνύουμε και δεν χρησιμοποιείται από κάποιο container.

```
PS C:\Users\apostolos> docker rmi erasmapp-web
Untagged: erasmapp-web:latest
Deleted: sha256:0935b2e7c0c70689394b905a9e93282a1b9f7a6e9057a51860138dcf089bbfd5
PS C:\Users\apostolos> docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginx	latest	c15da6c91de8	5 weeks ago	279MB
mysql	8.0	51d7ec709cde	5 weeks ago	1.06GB
mysql	5.7	4bc6bc963e6d	17 months ago	689MB

docker rmi -f <image_name/image_id>: Διαγράφει βιαίως την εικόνα που του υποδεικνύουμε ακόμα και αν χρησιμοποιείται από κάποιο container.

```
PS C:\Users\apostolos> docker rmi c15da6c91de8
Error response from daemon: conflict: unable to delete c15da6c91de8 (must be forced) - image is being used by stopped container 9970757c2393
PS C:\Users\apostolos> docker rmi -f c15da6c91de8
Untagged: nginx:latest
Deleted: sha256:c15da6c91de8d2f436196f3a768483ad32c258ed4e1beb3d367a27ed67253e66
PS C:\Users\apostolos> docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mysql	8.0	51d7ec709cde	5 weeks ago	1.06GB
mysql	5.7	4bc6bc963e6d	17 months ago	689MB

docker volume ls: Μας εμφανίζει όλα τα volumes που έχουν δημιουργηθεί στο docker engine του υπολογιστή μας.

```
PS C:\Users\apostolos> docker volume ls
```

DRIVER	VOLUME NAME
local	erasmapp_db_data

`docker volume rm <volume_name>`: Διαγράφει το docker volume που του υποδεικνύουμε.

```
PS C:\Users\apostolos> docker volume ls
DRIVER      VOLUME NAME
local       erasmapp_db_data
PS C:\Users\apostolos> docker volume rm erasmapp_db_data
erasmapp_db_data
PS C:\Users\apostolos> docker volume ls
DRIVER      VOLUME NAME
PS C:\Users\apostolos>
```

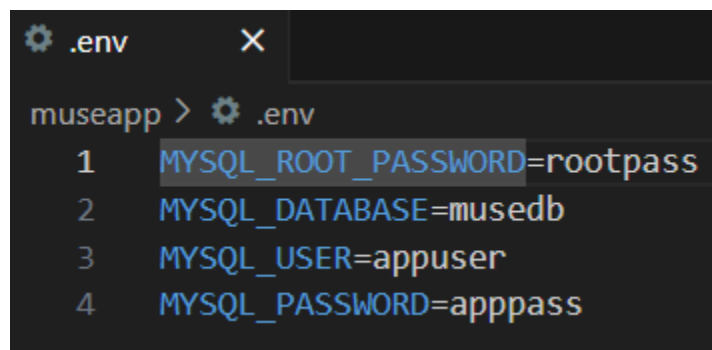
3.2 Προετοιμασία αρχείων docker εφαρμογής

Εφόσον έχουν παρουσιαστεί οι βασικές εντολές του docker, σε αυτό το σημείο θα αναλυθούν τα αρχεία του docker, που θα δημιουργήσουν τα container της εφαρμογής που θέλουμε να κάνουμε ανάπτυξη.

Η εφαρμογή βασίζεται στο απλό μοντέλο client-server. Από την πλευρά των servers, χρησιμοποιούνται ένας server `mySQL` για την βάση δεδομένων και ένας server `apache` με εγκατεστημένη την `PHP` για την επικοινωνία του frontend με την βάση. Αναλυτικά τα αρχεία που χρησιμοποιούνται έχουν όπως παρακάτω.

`.env`

Το αρχείο `.env` περιέχει μεταβλητές περιβάλλοντος που χρησιμοποιούνται από το `compose.yml` για να παραμετροποιήσουν τις ρυθμίσεις των containers. Είναι ένα απλό αρχείο κειμένου με ζεύγη `key=value`.



```
.env
museapp > .env
1 MYSQL_ROOT_PASSWORD=rootpass
2 MYSQL_DATABASE=musedb
3 MYSQL_USER=appuser
4 MYSQL_PASSWORD=apppass
```

```
1 MYSQL_ROOT_PASSWORD=rootpass
```

Ο κωδικός για τον root χρήστη της MySQL. Χρησιμοποιείται για τη διαχείριση της βάσης δεδομένων.

```
2 MYSQL_DATABASE=musedb
```

Το όνομα της βάσης δεδομένων που θα δημιουργηθεί αυτόματα από τη MySQL κατά την εκκίνηση του container.

```
3 MYSQL_USER=appuser
```

Ένας επιπλέον χρήστης (εκτός του root) που θα δημιουργηθεί για τη σύνδεση στην βάση δεδομένων. Τον χρησιμοποιούμε ώστε η PHP να μην χρησιμοποιεί τον χρήστη root για την επικοινωνία με την βάση για ενίσχυση της ασφάλειας της εφαρμογής.

```
4 MYSQL_PASSWORD=apppass
```

Ο κωδικός για τον χρήστη appuser.

Οι τιμές αυτές εισάγονται στο compose.yml μέσω της σύνταξης \${VARIABLE_NAME}. Το αρχείο .env φορτώνεται αυτόματα από το Docker Compose.

Dockerfile

Το αρχείο Dockerfile περιγράφει πώς θα χτιστεί το image για την υπηρεσία php. Είναι ένα αρχείο οδηγιών που λέει στο Docker πώς να δημιουργήσει ένα προσαρμοσμένο container image.

```
Dockerfile 1 X
museapp > Dockerfile > ...
1 FROM php:8.2-apache
2
3 RUN docker-php-ext-install pdo pdo_mysql
```



```
1 FROM php:8.2-apache
```

Βασικό Image: Χρησιμοποιεί το επίσημο Docker image php:8.2-apache από το Docker Hub. Αυτό το image περιλαμβάνει την PHP 8.2 και έναν Apache web server, έτοιμο να εξυπηρετήσει PHP εφαρμογές.

```
3 RUN docker-php-ext-install pdo pdo_mysql
```

Εγκατάσταση Επεκτάσεων: Εκτελεί την εντολή docker-php-ext-install για να εγκαταστήσει δύο επεκτάσεις της PHP:

pdo: Η επέκταση PDO (PHP Data Objects) που επιτρέπει τη σύνδεση με βάσεις δεδομένων μέσω ενός γενικού API.

pdo_mysql: Η υποστήριξη για τη MySQL μέσω PDO, που επιτρέπει στην PHP εφαρμογή να συνδεθεί στη βάση δεδομένων MySQL.

Το Dockerfile είναι αρκετά απλό, καθώς επεκτείνει το βασικό image με τις απαραίτητες επεκτάσεις για σύνδεση με τη MySQL. Αν η εφαρμογή χρειάζεται επιπλέον επεκτάσεις ή βιβλιοθήκες, θα προστεθούν περισσότερες εντολές εδώ.

compose.yml

Το αρχείο compose.yml (ή docker-compose.yml) είναι ένα αρχείο ρυθμίσεων για το Docker Compose. Είναι ένα εργαλείο που χρησιμοποιείται για τη διαχείριση πολλαπλών containers. Περιγράφει τις υπηρεσίες (services), τα δίκτυα (networks), και τους τόμους (volumes) που απαιτούνται για την εκτέλεση της εφαρμογής.

```
compose.yml X
museapp > compose.yml > YAML > version
docker-compose.yml - The Compose specification establishes a standard for the definition of multi-container platform-agnostic applications (compose-spec.json)
1 version: '3.8'
2
3 >Run All Services
services:
4 >Run Service
php:
5 build: .
6 ports:
7 - "8000:80"
8 volumes:
9 - ./app:/var/www/html
10 depends_on:
11 - db
12 environment:
13 MYSQL_DATABASE: ${MYSQL_DATABASE}
14 MYSQL_USER: ${MYSQL_USER}
15 MYSQL_PASSWORD: ${MYSQL_PASSWORD}
16
17 >Run Service
db:
18 image: mysql:8.0
19 restart: unless-stopped
20 ports:
21 - "3306:3306"
22 environment:
23 MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
24 MYSQL_DATABASE: ${MYSQL_DATABASE}
25 MYSQL_USER: ${MYSQL_USER}
26 MYSQL_PASSWORD: ${MYSQL_PASSWORD}
27 command: --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci --skip-character-set-client-handshake
28 volumes:
29 - db_data:/var/lib/mysql
30 - ./dbscripts:/docker-entrypoint-initdb.d
31
32 volumes:
33 db_data:
```

```
1 version: '3.8'
```

Αφορά την έκδοση του Docker Compose. Ορίζει ότι το αρχείο χρησιμοποιεί τη σύνταξη της έκδοσης 3.8 του Docker Compose. Αυτή η έκδοση καθορίζει τη δομή και τις δυνατότητες που υποστηρίζονται.

```
3 services:
```

Ορίζει τις υπηρεσίες (containers) που θα τρέξουν. Στην δοκιμαστική εφαρμογή που παρουσιάζεται έχουμε δύο υπηρεσίες: php και db.

```
4 php:
```

Ανάλυση του container της php.

```
5 build: .
```

Λέει στο Docker Compose να δημιουργήσει ένα image για την υπηρεσία php χρησιμοποιώντας το Dockerfile που βρίσκεται στον ίδιο φάκελο (.). Το Dockerfile περιγράφει, όπως αναλύσαμε παραπάνω, πώς θα χτιστεί το container.

```
6   ports:
7     - "8000:80"
```

Συνδέει τη θύρα 8000 του host (του υπολογιστή που τρέχει τον Docker Engine) με τη θύρα 80 του container. Αυτό σημαίνει ότι αν θέσουμε σε λειτουργία το container και μεταβούμε στην διαδρομή `http://localhost:8000` από τον browser του host, θα δούμε την PHP εφαρμογή που τρέχει στο container.

Η θύρα 80 είναι η προεπιλεγμένη θύρα για τον Apache web server που χρησιμοποιείται στο container.

```
8   volumes:
9     - ./app:/var/www/html
```

Συνδέει τον φάκελο `./app` από το host με τον φάκελο `/var/www/html` μέσα στο container. Ο φάκελος `/var/www/html` είναι ο προεπιλεγμένος φάκελος του Apache για την αποθήκευση αρχείων της εφαρμογής.

Αυτό σημαίνει ότι οποιαδήποτε αλλαγή γίνει στον φάκελο `./app` στον υπολογιστή του host, θα εμφανιστεί αυτόματα στο container, και αντίστροφα.

```
10  depends_on:
11    - db
```

Δηλώνει ότι η υπηρεσία `php` εξαρτάται από την υπηρεσία `db`. Το Docker Compose θα εξασφαλίσει ότι το container της βάσης δεδομένων (`db`) θα ξεκινήσει πριν από το container της PHP.

```
12  environment:
13    MYSQL_DATABASE: ${MYSQL_DATABASE}
14    MYSQL_USER: ${MYSQL_USER}
15    MYSQL_PASSWORD: ${MYSQL_PASSWORD}
```

Ορίζει μεταβλητές περιβάλλοντος για το container `php`. Αυτές οι μεταβλητές χρησιμοποιούνται από την PHP εφαρμογή για να συνδεθεί στη βάση δεδομένων.

Οι τιμές (`${MYSQL_DATABASE}`, `${MYSQL_USER}`, `${MYSQL_PASSWORD}`) αντλούνται από το αρχείο `.env` που περιγράψαμε παραπάνω.

```
17  db:
```

Ανάλυση του container της `db`.

```
18  image: mysql:8.0
```

Χρησιμοποιεί το επίσημο Docker image της MySQL έκδοσης 8.0 από το Docker Hub.

```
19  restart: unless-stopped
```

Ορίζει ότι το container της βάσης δεδομένων θα κάνει επανεκκίνηση αυτόματα αν σταματήσει (π.χ. λόγω σφάλματος), εκτός αν το σταματήσει ο χρήστης.

```
20     ports:
21     - "3306:3306"
```

Συνδέει τη θύρα 3306 του host με τη θύρα 3306 του container. Η θύρα 3306 είναι η προεπιλεγμένη θύρα της MySQL, επιτρέποντας σύνδεση στη βάση δεδομένων από το host (π.χ. μέσω εργαλείου όπως το MySQL Workbench).

```
22     environment:
23       MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
24       MYSQL_DATABASE: ${MYSQL_DATABASE}
25       MYSQL_USER: ${MYSQL_USER}
26       MYSQL_PASSWORD: ${MYSQL_PASSWORD}
```

Ορίζει μεταβλητές περιβάλλοντος για τη ρύθμιση της MySQL:

MYSQL_ROOT_PASSWORD: Ο κωδικός για τον root χρήστη της MySQL.

MYSQL_DATABASE: Το όνομα της βάσης δεδομένων που θα δημιουργηθεί.

MYSQL_USER: Ένας επιπλέον χρήστης για τη βάση δεδομένων (εκτός του root).

MYSQL_PASSWORD: Ο κωδικός για τον παραπάνω χρήστη.

Οι τιμές αυτών των μεταβλητών αντλούνται από το αρχείο .env.

```
27     command: --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci --skip-character-set-client-handshake
```

Απαραίτητη εντολή για κωδικοποιήσει σωστά τους χαρακτήρες utf8mb4, κατά την εισαγωγή του αρχείου bdummysdata.sql. Προσαρμόζει τις ρυθμίσεις της MySQL για να χρησιμοποιεί τον κωδικοποιητή χαρακτήρων utf8mb4 και το collation utf8mb4_unicode_ci. Αυτό εξασφαλίζει ότι η βάση δεδομένων υποστηρίζει πλήρως Unicode χαρακτήρες (π.χ. emoji, ελληνικά).

Η επιλογή --skip-character-set-client-handshake εξασφαλίζει ότι οι πελάτες (clients) που συνδέονται στη βάση δεδομένων δεν μπορούν να αλλάξουν τον κωδικοποιητή.

```
28     volumes:
```

```
29     - db_data:/var/lib/mysql
```

Συνδέει έναν named volume (db_data) με τον φάκελο /var/lib/mysql του container, όπου η MySQL αποθηκεύει τα δεδομένα της βάσης. Αυτό εξασφαλίζει ότι τα δεδομένα παραμένουν ακόμα και αν το container διαγραφεί ή σταματήσει.

```
30     - ./dbscripts:/docker-entrypoint-initdb.d
```

Συνδέει τον φάκελο ./dbscripts του host με τον φάκελο /docker-entrypoint-initdb.d του container. Η MySQL εκτελεί αυτόματα όλα τα .sql ή .sh αρχεία σε αυτόν τον φάκελο, με αλφαριθμητική σειρά, κατά την αρχικοποίηση της βάσης δεδομένων, επιτρέποντας τη δημιουργία πινάκων ή την εισαγωγή δεδομένων.

```
32 volumes:
33 db_data:
```

Δηλώνει έναν named volume (db_data) που χρησιμοποιείται για την αποθήκευση των δεδομένων της MySQL. Αυτός ο τόμος παραμένει ανεξάρτητος από το container και διατηρεί τα δεδομένα ακόμα και αν το container καταστραφεί.

Συμπερασματικά, τα τρία αρχεία συνεργάζονται για να δημιουργήσουν και να τρέξουν μια PHP εφαρμογή με MySQL βάση δεδομένων σε Docker containers. Για να εκτελέσουμε της εντολές στο Docker Engine. Αφού ξεκινήσουμε τον engine, από ένα τερματικό, μεταβαίνουμε στην διαδρομή του φακέλου.

```
PS C:\Users\apostolos> cd Desktop/museapp
PS C:\Users\apostolos\Desktop\museapp> |
```

Σε αυτό το σημείο, θα εκτελέσουμε την παρακάτω εντολή για να τρέξουμε το αρχείο compose.yml, που θα δημιουργήσει και τρέξει τα container.

```
PS C:\Users\apostolos\Desktop\museapp> docker-compose up --build
```

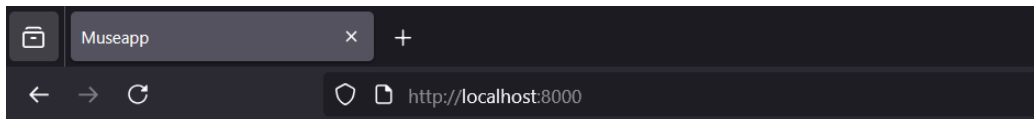
```
PS C:\Users\apostolos\Desktop\museapp> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
48ba547eb5e7   museapp-php   "docker-php-entrypoi..." About a minute ago Up 59 seconds  0.0.0.0:8000->80/tcp                museapp-php-1
ba99b3e1e7cf   mysql:8.0     "docker-entrypoint.s..." About a minute ago Up 59 seconds  0.0.0.0:3306->3306/tcp, 33060/tcp   museapp-db-1

PS C:\Users\apostolos\Desktop\museapp> docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
museapp-php   latest   61ef69a1f6dc   About a minute ago  709MB
mysql         8.0      51d7ec709cde   5 weeks ago     1.06GB
mysql         5.7      4bc6bc963e6d   17 months ago    689MB

PS C:\Users\apostolos\Desktop\museapp> docker volume ls
DRIVER    VOLUME NAME
local     museapp_db_data
```

Με τις εντολές του docker που είδαμε παραπάνω, παρατηρούμε ότι, έχουν δημιουργηθεί τα δύο container (museapp-php-1 και museapp-db-1). Βλέπουμε ότι δημιουργήθηκε ένα image (museapp-php) που είναι το image που είχαμε συντάξει στο Dockerfile και το δημιούργησε το docker-compose. Τέλος παρατηρούμε ότι δημιουργήθηκε το volume (museapp_db_data) που ζητήσαμε να δημιουργήσει το docker-compose, όπου θα αποθηκεύονται οι αλλαγές στην βάση.

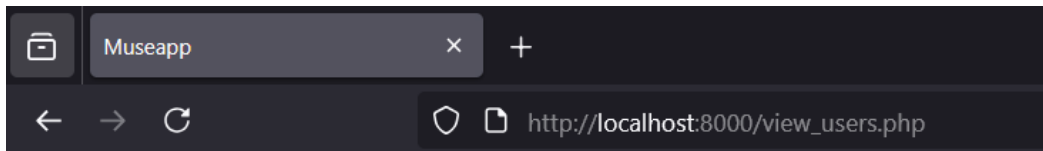
Αρχικά, μπορούμε να δούμε την εφαρμογή από την διεύθυνση <http://localhost:8000>, και να χρησιμοποιήσουμε την εφαρμογή.



Αρχική σελίδα εφαρμογής κράτησης αιθουσών

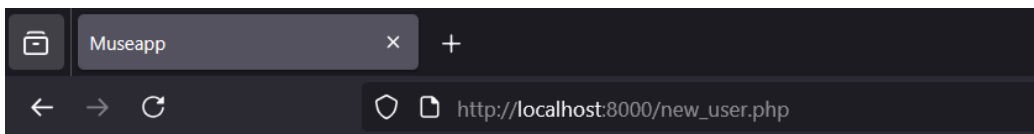
[Δείτε τους χρήστες.](#)

[Δείτε τις αίθουσες.](#)

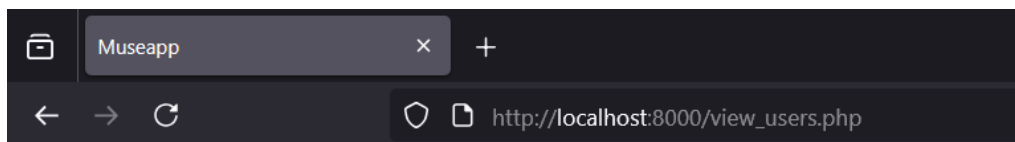


ID	Όνομα	Επώνυμο	Email
1	Αθανάσιος	Πέτρου	apetrou@mail.com
2	Νικόλαος	Ιωάννου	niwannou@mail.com
3	Μαρία	Χατζή	mchatzi@mail.com

[Προσθέστε νέο χρήστη.](#)



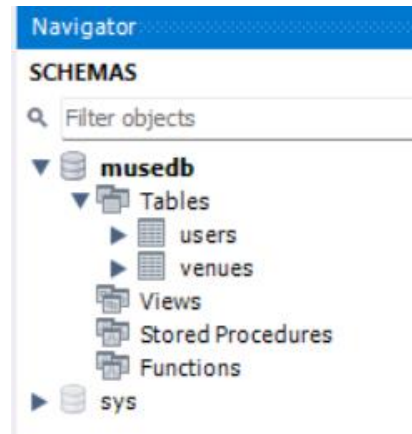
Όνομα:
Επώνυμο:
Email:



ID	Όνομα	Επώνυμο	Email
1	Αθανάσιος	Πέτρου	apetrou@mail.com
2	Νικόλαος	Ιωάννου	niwannou@mail.com
3	Μαρία	Χατζή	mchatzi@mail.com
4	Απόστολος	Πεσλής	apo@mail.com

[Προσθέστε νέο χρήστη.](#)

Επειδή έχουμε κάνει expose την port 3306 στον container της mysql, μπορούμε να συνδεθούμε με τα διαπιστευτήρια που έχουμε ορίσει στο .env από workbench.



```
1 • SELECT * FROM musedb.users;
```

Result Grid				
Filter Rows:				
	userid	firstname	lastname	email
▶	1	Αθανάσιος	Πέτρου	apetrou@mail.com
	2	Νικόλαος	Ιωάννου	niwannou@mail.com
	3	Μαρία	Χατζή	mchatzi@mail.com
	4	Απόστολος	Πεσλής	apo@mail.com
*	NULL	NULL	NULL	NULL

```
1 • SELECT * FROM musedb.venues;
```

Result Grid		
Filter Rows:		
	venueid	venuecapacity
▶	1	Μεγάλο Αμφιθέατρο
	2	Μικρό Αμφιθέατρο
	3	Εκπαιδευτική Αίθουσα
*	NULL	NULL

Τέλος, για να σταματήσουμε τα container εκτελούμε την παρακάτω εντολή:

```
PS C:\Users\apostolos\Desktop\museapp> docker-compose stop
time="2025-05-22T12:32:30+03:00" level=warning msg="C:\\Users\\apostolos\\Desktop\\museapp\\compose.yml: the attribute `version` is obsolete, it
will be ignored, please remove it to avoid potential confusion"
[+] Stopping 2/2
 ✓ Container museapp-php-1   Stopped                               1.7s
 ✓ Container museapp-db-1    Stopped                               1.9s
PS C:\Users\apostolos\Desktop\museapp> docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED   STATUS    PORTS    NAMES
PS C:\Users\apostolos\Desktop\museapp>
```

Για να τα θέσουμε εκ νέου σε λειτουργία:

```
PS C:\Users\apostolos\Desktop\museapp> docker-compose start
time="2025-05-22T12:33:27+03:00" level=warning msg="C:\\Users\\apostolos\\Desktop\\museapp\\compose.yml: the attribute `version` is obsolete, it
will be ignored, please remove it to avoid potential confusion"
[+] Running 2/2
 ✓ Container museapp-db-1    Started                               0.6s
 ✓ Container museapp-php-1   Started                               0.5s
PS C:\Users\apostolos\Desktop\museapp> docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED   STATUS    PORTS    NAMES
48ba547eb5e7   museapp-php  "docker-php-entrypoi..."  21 minutes ago  Up 24 seconds  0.0.0.0:8000->80/tcp  museapp-php-1
ba99b3e1e7cf   mysql:8.0    "docker-entrypoint.s..."  21 minutes ago  Up 24 seconds  0.0.0.0:3306->3306/tcp, 33060/tcp  museapp-db-1
PS C:\Users\apostolos\Desktop\museapp>
```

Με το docker-compose down, σταματάμε και διαγράφουμε τα container (χωρίς όμως να διαγραφεί το volume που έχει αποθηκευμένα τα δεδομένα της βάσης).

```
PS C:\Users\apostolos\Desktop\museapp> docker-compose down
time="2025-05-22T12:38:25+03:00" level=warning msg="C:\\Users\\apostolos\\Desktop\\museapp\\compose.yml: the attribute `version` is obsolete, it
will be ignored, please remove it to avoid potential confusion"
[+] Running 3/3
 ✓ Container museapp-php-1   Removed                               1.7s
 ✓ Container museapp-db-1    Removed                               1.6s
 ✓ Network museapp_default   Removed                               0.8s
PS C:\Users\apostolos\Desktop\museapp> docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED   STATUS    PORTS    NAMES
PS C:\Users\apostolos\Desktop\museapp>
```

```
PS C:\Users\apostolos\Desktop\museapp> docker volume ls
DRIVER      VOLUME NAME
local       museapp_db_data
PS C:\Users\apostolos\Desktop\museapp> docker volume rm museapp_db_data
museapp_db_data
PS C:\Users\apostolos\Desktop\museapp> docker volume ls
DRIVER      VOLUME NAME
PS C:\Users\apostolos\Desktop\museapp>
```

Η τεχνολογία του Docker αναδεικνύεται ως μια πρωτοποριακή λύση για την ανάπτυξη και διαχείριση σύγχρονων εφαρμογών, προσφέροντας ελαφριά και φορητά περιβάλλοντα. Εξασφαλίζει ευελιξία, ταχύτητα και αξιοπιστία, καθιστώντας το ιδανικό για περιβάλλοντα DevOps και cloud-native εφαρμογές. Σε σύγκριση με τις εικονικές μηχανές, το Docker υπερέχει σε αποδοτικότητα πόρων και ταχύτητα εκκίνησης, ενώ η πρακτική εφαρμογή του μέσω της ανάπτυξης μιας εφαρμογής με frontend, backend και βάση δεδομένων αποδεικνύει την ικανότητά του να απλοποιεί τη δημιουργία σύνθετων συστημάτων. Η εξοικείωση με τις βασικές εντολές και τα αρχεία όπως το Dockerfile και το docker-

compose.yml επιτρέπει στους προγραμματιστές να εκμεταλλεύονται πλήρως τις δυνατότητες του Docker, εξασφαλίζοντας συνέπεια σε όλα τα στάδια του κύκλου ζωής του λογισμικού.

4 Συμπέρασμα

Η παρούσα εργασία ανέδειξε τη σημασία και την αξία του Docker ως εργαλείου στην ανάπτυξη σύγχρονων εφαρμογών λογισμικού, προσφέροντας μια ολοκληρωμένη θεωρητική και πρακτική προσέγγιση. Μέσα από την ανάλυση της αρχιτεκτονικής του Docker, διαπιστώθηκε ότι η τεχνολογία containerization επιτρέπει τη δημιουργία ελαφρών, φορητών και απομονωμένων περιβαλλόντων, τα οποία εξασφαλίζουν συνέπεια μεταξύ των σταδίων ανάπτυξης, δοκιμής και παραγωγής.

Η σύγκριση με τις εικονικές μηχανές κατέδειξε την υπεροχή του Docker σε όρους αποδοτικότητας πόρων, ταχύτητας εκκίνησης και ευελιξίας, καθιστώντας το ιδανικό για εφαρμογές μικροϋπηρεσιών και πρακτικές CI/CD. Η πρακτική εφαρμογή, μέσω της δημιουργίας μιας εφαρμογής με frontend, backend και βάση δεδομένων, επιβεβαίωσε την ευκολία χρήσης του Docker, καθώς και την ικανότητά του να αντιμετωπίζει το πρόβλημα της ασυμβατότητας περιβαλλόντων.

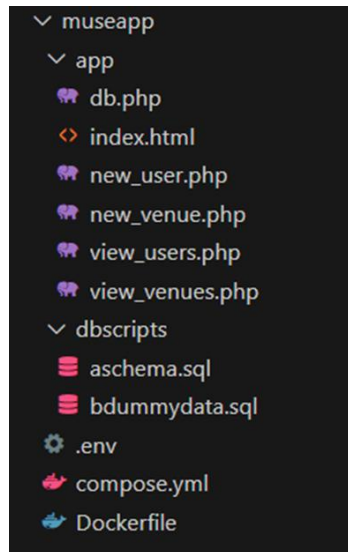
Παρά τα πλεονεκτήματα, όπως η φορητότητα και η ταχύτητα, το Docker παρουσιάζει προκλήσεις, όπως η ανάγκη για κατανόηση σύνθετων εννοιών δικτύωσης και διαχείρισης πόρων, καθώς και η εξάρτηση από την υγεία του host OS. Συνολικά, το Docker αποτελεί ένα ισχυρό εργαλείο που μεταμορφώνει την αρχιτεκτονική λογισμικού, προσφέροντας ευελιξία και αξιοπιστία, ενώ η συνεχής εξέλιξή του υπόσχεται ακόμα μεγαλύτερες δυνατότητες στο μέλλον.

Βιβλιογραφία

- [1] Docker, «docker.com,» Docker, [Ηλεκτρονικό]. Available: <https://www.docker.com/why-docker/>. [Πρόσβαση Μάι 2025].
- [2] S. Johnston, «docker.com,» 21 Μαρ 2024. [Ηλεκτρονικό]. Available: <https://www.docker.com/blog/docker-11-year-anniversary/>. [Πρόσβαση 20 Μαΐ 2025].
- [3] Docker, «docker.com,» Docker, [Ηλεκτρονικό]. Available: <https://www.docker.com/resources/what-container/>. [Πρόσβαση 20 Μαΐ 2025].
- [4] L. Muller, «udemy.com,» Udemy, [Ηλεκτρονικό]. Available: <https://www.udemy.com/course/complete-docker-kubernetes/?kw=docker&src=sac&couponCode=CP130525>. [Πρόσβαση Απρ 2025].
- [5] Docker, «docker.com,» Docker, [Ηλεκτρονικό]. Available: <https://docs.docker.com/get-started/docker-overview/>. [Πρόσβαση 21 Μαΐ 2025].
- [6] D. Merkel, «Docker: lightweight Linux containers for consistent development and deployment,» *Linux Journal*, p. 2, 1 Μαρ 2014.

Παράρτημα

Δομή εφαρμογής



Αρχεία φακέλου app

```
<> index.html X
museapp > app > <> index.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Museapp</title>
7  </head>
8  <body>
9      <h1>Αρχική σελίδα εφαρμογής κράτησης αιθουσών</h1>
10     <p><a href="view_users.php">Δείτε τους χρήστες.</a></p>
11     <p><a href="view_venues.php">Δείτε τις αίθουσες.</a></p>
12 </body>
13 </html>
```

Η αρχική σελίδα της εφαρμογής, περιέχει δύο συνδέσμους για ανακατεύθυνση στην σελίδα προβολής χρηστών και στην σελίδα προβολής αιθουσών.

```
db.php ×
museapp > app > db.php
1  <?php
2      $host = 'db';
3      $db = getenv('MYSQL_DATABASE');
4      $user = getenv('MYSQL_USER');
5      $pass = getenv('MYSQL_PASSWORD');
6      $charset = 'utf8mb4';
7
8      $dsn = "mysql:host=$host;dbname=$db;charset=$charset";
9      $options = [
10         PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8mb4",
11         PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
12         PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
13         PDO::ATTR_EMULATE_PREPARES => false,
14     ];
15
16     try {
17         $pdo = new PDO($dsn, $user, $pass, $options);
18     } catch (PDOException $e) {
19         die("DB Connection failed: " . $e->getMessage());
20     }
21  ?>
```

Η σελίδα για την διασύνδεση με την βάση δεδομένων. Δεν έχει κάποιο περιεχόμενο για εμφάνιση. Η λειτουργία της είναι η σύνδεση με την βάση δεδομένων. Χρησιμοποιεί το αρχείο περιβάλλοντος .env για να πάρει το όνομα της βάσης, το username και το password. Κάνει προσπάθεια σύνδεσης με την βάση για έλεγχο και εμφανίζει μήνυμα σφάλματος σε περίπτωση αποτυχίας.

```

view_users.php X
museapp > app > view_users.php
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Museapp</title>
7 </head>
8 <body>
9     <?php
10         require 'db.php';
11
12         $stmt = $pdo->query("SELECT * FROM users");
13         $users = $stmt->fetchAll();
14     ?>
15
16     <table border="1">
17         <tr>
18             <th>ID</th><th>Όνομα</th><th>Επώνυμο</th><th>Email</th>
19         </tr>
20         <?php foreach ($users as $user): ?>
21             <tr>
22                 <td><?= $user['userid'] ?></td>
23                 <td><?= $user['firstname'] ?></td>
24                 <td><?= $user['lastname'] ?></td>
25                 <td><?= $user['email'] ?></td>
26             </tr>
27         <?php endforeach; ?>
28     </table>
29     <p><a href="new_user.php">Προσθέστε νέο χρήστη.</a></p>
30 </body>
31 </html>

```

Η σελίδα προβολής χρηστών. Χρησιμοποιεί την σελίδα σύνδεσης με την βάση για να λάβει τα δεδομένα. Εμφανίζει όλους τους χρήστες με τα στοιχεία τους, εκτελώντας το αντίστοιχο query στην βάση. Περιέχει έναν σύνδεσμο για την μετάβαση στην σελίδα προσθήκης νέου χρήστη.

```

view_venues.php X
museapp > app > view_venues.php
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Museapp</title>
7 </head>
8 <body>
9     <?php
10         require 'db.php';
11
12         $stmt = $pdo->query("SELECT * FROM venues");
13         $venues = $stmt->fetchAll();
14     ?>
15
16     <table border="1">
17         <tr>
18             <th>ID</th><th>Όνομα Χώρου</th><th>Χωρητικότητα</th>
19         </tr>
20         <?php foreach ($venues as $venue): ?>
21             <tr>
22                 <td><?= $venue['venueid'] ?></td>
23                 <td><?= $venue['venueName'] ?></td>
24                 <td><?= $venue['capacity'] ?></td>
25             </tr>
26         <?php endforeach; ?>
27     </table>
28     <p><a href="new_venue.php">Προσθέστε νέα αίθουσα.</a></p>
29 </body>
30 </html>

```

Η σελίδα για την προβολή αιθουσών. Έχει ακριβώς την ίδια λειτουργία με την σελίδα προβολής χρηστών αλλά σε ότι αφορά τις αίθουσες.

```

new_user.php X
museapp > app > new_user.php
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>Museapp</title>
7 </head>
8 <body>
9 <?php
10     require 'db.php';
11
12     if ($_SERVER["REQUEST_METHOD"] == "POST") {
13         $stmt = $pdo->prepare("INSERT INTO users (firstname, lastname, email) VALUES (?, ?, ?)");
14         $stmt->execute([$POST['firstname'], $POST['lastname'], $POST['email']]);
15         echo "Ο χρήστης προστέθηκε!";
16     }
17 }>
18
19 <form method="post">
20     Όνομα: <input type="text" name="firstname"><br>
21     Επώνυμο: <input type="text" name="lastname"><br>
22     Email: <input type="email" name="email"><br>
23     <input type="submit" value="Προσθήκη">
24 </form>
25 </body>
26 </html>

```

Η σελίδα προσθήκης νέου χρήστη. Περιέχει μία φόρμα με τα αντίστοιχα πεδία για την καταχώρηση νέου χρήστη. Χρησιμοποιεί την σελίδα db.php για να κάνει σύνδεση με την βάση. Αν η σελίδα κληθεί με POST, εκτελεί ένα query για την εισαγωγή νέου χρήστη στην βάση δεδομένων.

```

new_venue.php X
museapp > app > new_venue.php
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>Museapp</title>
7 </head>
8 <body>
9 <?php
10     require 'db.php';
11
12     if ($_SERVER["REQUEST_METHOD"] == "POST") {
13         $stmt = $pdo->prepare("INSERT INTO venues (venueName, capacity) VALUES (?, ?)");
14         $stmt->execute([$POST['venueName'], $POST['capacity']]);
15         echo "Ο χώρος προστέθηκε!";
16     }
17 }>
18
19 <form method="post">
20     Όνομα Χώρου: <input type="text" name="venueName"><br>
21     Χωρητικότητα: <input type="number" name="capacity"><br>
22     <input type="submit" value="Προσθήκη">
23 </form>
24 </body>
25 </html>

```

Η σελίδα προσθήκης νέας αίθουσας. Έχει ακριβώς την ίδια λειτουργία με την σελίδα προσθήκης χρηστών αλλά σε ότι αφορά τις αίθουσες.

Αρχεία φακέλου dbscripts

```
aschema.sql X
museapp > dbscripts > aschema.sql
1 drop database if exists musedb;
2 create database musedb;
3 use musedb;
4
5 drop table if exists users;
6 create table users (
7     userid integer not null unique auto_increment,
8     firstname varchar(100) not null,
9     lastname varchar(100) not null,
10    email varchar(100) not null
11 );
12
13 drop table if exists venues;
14 create table venues (
15     venueid integer not null unique auto_increment,
16     venuename varchar(100) not null,
17     capacity integer not null
18 );
```

Το παραπάνω sql αρχείο δημιουργεί την βάση δεδομένων της εφαρμογής. Δημιουργεί τους πίνακες των χρηστών και των αιθουσών με τα αντίστοιχα γνωρίσματά τους.

```
bdummydata.sql X
museapp > dbscripts > bdummydata.sql
1 insert into venues (venueid, venueid, venueid) values
2 ('Μεγάλο Αμφιθέατρο', 500),
3 ('Μικρό Αμφιθέατρο', 300),
4 ('Εκπαιδευτική Αίθουσα', 300);
5
6 insert into users (userid, userid, userid) values
7 ('Αθανάσιος', 'Πέτρου', 'apetrou@mail.com'),
8 ('Νικόλαος', 'Ιωάννου', 'niwannou@mail.com'),
9 ('Μαρία', 'Χατζή', 'mchatzi@mail.com');
```

Το παραπάνω αρχείο sql, περιέχει δοκιμαστικά δεδομένα. Συγκεκριμένα προσθέτει μερικές τιμές στα περιεχόμενα των πινάκων της βάσης δεδομένων.