## 🐳 ****Пункт 1: Що таке Docker?****

### 🔥 ****1.1 Проблема без Docker****

#### 💣 Типова ситуація:

👨‍💻 Розробник А: «В мене все працює»  
👨‍💻 Розробник Б: «А в мене все ламається…»  
👨‍💻 DevOps: «А на продакшені це навіть не стартує» 😤

Чому так?

* Різні версії бібліотек
* Різні налаштування середовища
* Різні операційні системи
* Складна інсталяція залежностей

**Docker вирішує ці проблеми через ізоляцію середовищ.**

### 📦 ****1.2 Що таке Docker?****

**Docker** — це платформа (або інструмент), яка дозволяє:

* **упакувати додаток з усіма залежностями** в один образ
* **запускати його в ізольованому середовищі** (контейнері)
* **переносити цей контейнер будь-куди**: на інший комп, сервер, хмару — і він завжди працює однаково

🧠 **Docker = Лего для софту:**  
Збираєш все необхідне в один контейнер → запускаєш → працює незалежно від ОС і "хаосу на машині"

### ⚙️ ****1.3 Чим Docker відрізняється від віртуальної машини?****

| Віртуальні машини (VM) | Docker |
| --- | --- |
| Має свою повну ОС | Ділить ядро ОС |
| Повільний старт | Швидкий старт |
| Більше RAM/CPU | Менше ресурсів |
| Важкий у використанні | Простий і гнучкий |

🔍 Docker **швидше**, **легше**, і **масштабується краще**.

### 🧱 ****1.4 Компоненти Docker (огляд)****

| Компонент | Пояснення |
| --- | --- |
| **Image** | "Шаблон" або "рецепт" програми |
| **Container** | "Запущений екземпляр" образу — робоча програма |
| **Dockerfile** | Файл, де ти пишеш, як створити образ |
| **Docker Hub** | Онлайн-репозиторій з готовими образами (як GitHub) |

### 💬 ****1.5 Мета Docker'а простими словами****

"Зроби один раз — запускай будь-де"

Ти створив образ свого додатку — і можеш віддати його:

* іншому розробнику
* тестеру
* DevOps'у
* або запустити на сервері

💥 І він буде працювати так само, **без ніякого "воно не збирається!"**

### 📌 ****Що варто наголосити студентам:****

* Docker — не заміна CI/CD або віртуалки, а **інструмент для ізольованого запуску додатків**
* Його використовують Google, Netflix, Spotify, PayPal, і майже всі в IT
* Docker — must-have для сучасного розробника

## 📦 ****Пункт 2: Основні поняття Docker****

Це база, яку студент має вивчити, щоб зрозуміти, що і як відбувається в Docker. Ось докладно, з аналогіями та прикладами:

### 🔹 ****2.1 Образ (Image)****

**Образ** — це інструкція, з якої створюється контейнер.  
Це як **рецепт піци**: у ньому вказано, які інгредієнти (бібліотеки, код) потрібні й як їх "змішати".

📌 Образ:

* Статичний
* Нічого не змінюється, поки ти не запустиш контейнер
* Створюється з Dockerfile

🔧 Приклад:

docker pull python:3.10

Це команда завантажує образ з офіційного репозиторію.

### 🔹 ****2.2 Контейнер (Container)****

**Контейнер** — це **живий, запущений екземпляр образу**.

Контейнер:

* Має свою ізольовану файлову систему, процеси, мережу
* Може читати з образу
* Може змінювати стан під час роботи (але зміни тимчасові)

📌 Аналогія:

* Образ — це рецепт піци
* Контейнер — це вже спечена піца, яку ти зараз їси

🔧 Приклад:

bash

1. docker run -it python:3.10

Ти запускаєш контейнер з Python-ом і заходиш всередину.

### 🔹 ****2.3 Dockerfile****

**Dockerfile** — це текстовий файл із командами, які Docker виконує, щоб створити образ.

📌 Як сценарій, де ти пишеш:

* яку базову ОС використати (FROM)
* які залежності встановити (RUN)
* що скопіювати (COPY)
* яку команду запускати (CMD)

🔧 Приклад Dockerfile:

Dockerfile

FROM python:3.10

WORKDIR /app

COPY . .

RUN pip install -r requirements.txt

CMD ["python", "main.py"]

### 🔹 ****2.4 Docker Hub****

**Docker Hub** — це GitHub для Docker-образів.

Там зберігаються:

* Офіційні образи (nginx, python, mysql…)
* Образи від компаній
* Твої власні (якщо хочеш залити)

📌 Посилання: https://hub.docker.com

🔧 Приклад:

docker pull nginx

docker run -d -p 80:80 nginx

### 🔹 ****2.5 Volume (Том)****

**Volume** — це спосіб зберегти дані, щоб вони не зникли після зупинки контейнера.

Бо за замовчуванням усе в контейнері — **тимчасове**.

📌 Volume дозволяє:

* Зберігати БД
* Передавати файли між хостом і контейнером
* Робити бекапи

🔧 Приклад:

docker volume create mydata

docker run -v mydata:/app/data myapp

### 🔹 ****2.6 Network (Мережа)****

**Network** — це спосіб об’єднувати кілька контейнерів у локальну мережу

📌 Docker автоматично створює:

* bridge — дефолтна
* host — спільна з хост-машиною
* none — повна ізоляція

🔧 Приклад:

docker network create mynetwork

docker run -d --network=mynetwork --name db mysql

docker run -d --network=mynetwork --name app myapp

## 📋 ****Коротке резюме (для дошки/слайду):****

| Поняття | Що це таке? | Аналогія |
| --- | --- | --- |
| Image | Шаблон для контейнера | Рецепт |
| Container | Запущений додаток з образу | Готова страва |
| Dockerfile | Інструкції для створення образу | Сценарій або рецепт |
| Docker Hub | Репозиторій образів | GitHub для контейнерів |
| Volume | Сховище для даних | USB-флешка |
| Network | З’єднання контейнерів | Віртуальний LAN |

## ⚙️ ****Пункт 3: Установка Docker****

## 🪟 ****3.1 Установка Docker на Windows****

👉 Рекомендується **Windows 10/11 Pro або Enterprise** (бо там є Hyper-V)

#### 🔸 Кроки:

1. Зайти на https://www.docker.com/products/docker-desktop
2. Завантажити **Docker Desktop for Windows**
3. Встановити (під час інсталяції увімкнути Hyper-V)
4. Перезавантажити ПК (іноді обов’язково)
5. Запустити Docker Desktop

📌 Після запуску:  
В системному треї має з’явитися значок 🐳, і він має бути зеленим.

🔍 **Перевірка:**

docker --version

## 🍏 ****3.2 Установка Docker на macOS****

#### 🔸 Кроки:

1. Зайти на https://www.docker.com/products/docker-desktop
2. Завантажити **Docker Desktop for Mac** (для Apple Silicon або Intel — залежно від моделі)
3. Перетягнути Docker у папку Applications
4. Запустити Docker

🔍 **Перевірка:**

docker --version

## 🐧 ****3.3 Установка Docker на Linux (Ubuntu приклад)****

#### 🔸 Кроки:

sudo apt update

sudo apt install \

ca-certificates \

curl \

gnupg \

lsb-release

Додати ключ і репозиторій:

sudo mkdir -m 0755 -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \

sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \

"deb [arch=$(dpkg --print-architecture) \

signed-by=/etc/apt/keyrings/docker.gpg] \

https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

Встановлення Docker:

sudo apt update

sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

🔍 Перевірити:

bash

docker --version

📌 Додати поточного користувача в docker-групу (щоб не писати sudo):

sudo usermod -aG docker $USER

Після цього треба перелогінитись або перезавантажити сесію.

## 🧪 ****3.4 Перевірка Docker після встановлення****

Незалежно від ОС, команда для тесту:

docker run hello-world

💬 Очікуваний результат:

Hello from Docker!

This message shows that your installation appears to be working correctly.

...

## 💡 ****3.5 Корисні поради****

* Docker Desktop може використовувати багато ресурсів — його варто вимикати, якщо не користуєшся
* Образи займають багато місця — періодично чистити через:

docker system prune

(⚠️ очищає всі непотрібні контейнери, образи й мережі)

* Якщо щось не працює — завжди перевірати:

docker info

## 🧰 ****3.6 Альтернативи для студентів з проблемами інсталяції****

* **Play with Docker** — онлайн-пісочниця: https://labs.play-with-docker.com/
* **Replit / GitPod / Codespaces** — можна налаштувати dockerized середовища для практики
* **WSL2 (Windows Subsystem for Linux)** + Docker CLI — полегшена альтернатива на Windows Home

## 🔧 ****Пункт 4: Базові команди Docker****

## 📋 ****4.1 Перевірка Docker після встановлення****

docker --version

✅ Перевіряємо, що Docker встановлено й доступно у консолі.

## 🧪 ****4.2 Перше знайомство: запуск**** hello-world

docker run hello-world

📌 Що відбувається:

* Docker шукає образ hello-world локально
* Якщо не знаходить — качає з Docker Hub
* Створює контейнер і запускає
* Виводить в консоль повідомлення

🧠 Поясни:

* run = "створи контейнер з образу і запусти його"
* hello-world = ім’я образу

## 🧊 ****4.3 Завантаження образу (без запуску)****

docker pull ubuntu

📌 Качає образ Ubuntu з Docker Hub.  
Можна використовувати без запуску, наприклад, для створення свого контейнера пізніше.

## 📦 ****4.4 Перегляд образів****

docker images

🔍 Побачимо:

* REPOSITORY — назва образу
* TAG — версія (наприклад, python:3.10)
* IMAGE ID — унікальний ID образу
* SIZE — скільки займає

## 🚀 ****4.5 Запуск контейнера****

docker run -it ubuntu ls bash

📌 Що робить ця команда:

* -it — інтерактивний режим + термінал
* ubuntu — образ
* bash — команда всередині контейнера (shell)

👉 Це відкриває консоль **всередині контейнера** з Ubuntu!

Вийти:

exit

## 🧠 ****4.6 Перегляд активних контейнерів****

docker ps

📋 Показує всі **запущені** контейнери:

* CONTAINER ID
* IMAGE
* STATUS
* PORTS (якщо є)
* NAMES (автоматичне або задане ім’я)

## 👀 ****4.7 Перегляд усіх контейнерів (включно з зупиненими)****

docker ps -a

📋 Показує історію: що запускалось, що впало, що зупинилось.

## ⛔ ****4.8 Зупинка контейнера****

docker stop <container\_id>

📌 Зупиняє активний контейнер.

Наприклад:

docker stop 9a3bcf842b57

## 🧹 ****4.9 Видалення контейнера****

docker rm <container\_id>

🧠 Пояснення:

* stop лише зупиняє
* rm видаляє (повністю)
* Зручно запускати в парі:

docker

docker stop <id> && docker rm <id>

## 🧽 ****4.10 Видалення образу****

docker rmi <image\_id>

⚠️ Якщо образ ще використовується запущеним або зупиненим контейнером — буде помилка.  
В такому разі — видалити контейнер спочатку.

## 📌 ****4.11 Очищення всього сміття (обережно!)****

docker system prune

Це:

* видаляє зупинені контейнери
* непотрібні образи
* неактивні мережі
* "все зайве"

🧨 Попередження: **все, що не використовується — зникне.**

## 🐚 ****4.12 Зайти в уже запущений контейнер****

docker exec -it <container\_id> bash

🔧 Приклад:

docker exec -it 9a3bcf842b57 bash

* exec — запуск команди **всередині** контейнера
* -it — інтерактивно
* bash — відкрити консоль

## 🧠 ****4.13 Корисні додаткові команди****

docker inspect <container\_id> # детальна інфа

docker logs <container\_id> # лог контейнера

docker stats # ресурси в реальному часі

docker top <container\_id> # процеси в контейнері

## 🌐 ****Пункт 5: Практика — запуск простого вебсервера в Docker****

## 🔹 ****5.1 Образ Nginx — що це?****

**Nginx** — це високопродуктивний вебсервер.  
Образ nginx в Docker Hub — офіційний, стабільний, маленький.

📌 Ми використаємо його, щоб:

* Показати, як працює контейнер
* Продемонструвати проброс портів
* Дати учням зворотний зв’язок в браузері

## 🐳 ****5.2 Команда запуску контейнера****

docker run -d -p 8080:80 nginx

🔍 Пояснення параметрів:

| Параметр | Що він робить |
| --- | --- |
| run | Запускає контейнер |
| -d | **Detach mode** – у фоні |
| -p 8080:80 | Проброс порту: з хосту (8080) у контейнер (80, стандарт Nginx) |
| nginx | Ім’я образу |

## 💻 ****5.3 Перевірка: браузер****

Відкриваємо браузер:

http://localhost:8080

🧠 Якщо бачиш: **"Welcome to nginx!"** — все працює. Контейнер запущено, порт проброшено, вебсервер на ходу.

## 🔍 ****5.4 Перевірка статусу****

docker ps

Там буде:

* контейнер з образом nginx
* STATUS: Up...
* PORTS: 0.0.0.0:8080->80/tcp

## 🔄 ****5.5 Зупинка та видалення контейнера****

docker stop <container\_id>

docker rm <container\_id>

Або в одній команді:

docker rm -f <container\_id>

🔍 Щоб знайти ID:

docker ps -a

## 📁 ****5.6 Додатково: запустити зі своєю HTML-сторінкою****

Покажи студентам, що можна “підкинути” свою сторінку в контейнер за допомогою volume.

#### Структура папки:

/my-website

└── index.html

#### Команда запуску:

docker run -d -p 8081:80 -v $(pwd)/my-website:/usr/share/nginx/html nginx

🔍 Що відбувається:

* -v $(pwd)/my-website:/usr/share/nginx/html — підключає локальну папку як "корінь сайту" в контейнері
* Відкриваємо: http://localhost:8081

## 🧾 ****Пункт 6: Dockerfile — створення власного образу (Java версія)****

### 🎯 ****Мета:****

* Навчити студентів створювати Dockerfile для Java-додатків
* Побудувати образ із .jar-файлом
* Запустити Java-додаток у контейнері

## 🧠 ****6.1 Що таке Dockerfile?****

Dockerfile — це **рецепт**, за яким Docker "пакує" додаток у образ.

📦 Тут ми самі створюємо образ для Java-додатку:

* Вказуємо базовий образ (openjdk)
* Копіюємо .jar-файл
* Вказуємо, як його запускати

## 📁 ****6.2 Структура простого Java-проєкту****

Для прикладу уявімо, що в тебе є вже зібраний .jar файл (наприклад, з Maven або Gradle).

/my-java-app

├── app.jar

└── Dockerfile

## 🔸 ****6.3 Dockerfile для Java-додатку****

Dockerfile

FROM openjdk:17-jdk-slim

# Створюємо робочу директорію

WORKDIR /app

# Копіюємо jar-файл

COPY app.jar app.jar

# Запускаємо додаток

CMD ["java", "-jar", "app.jar"]

🔍 Що тут відбувається:

* openjdk:17-jdk-slim — мінімальний, але повнофункціональний образ JDK 17
* COPY переносить app.jar у контейнер
* CMD вказує, що саме запускати

## 🛠 ****6.4 Побудова образу****

docker build -t my-java-app .

📌 Поясни:

* my-java-app — це ім’я образу, яке ми самі задали
* . — поточна директорія з Dockerfile

## 🚀 ****6.5 Запуск контейнера****

docker run -d -p 8080:8080 my-java-app

📌 Якщо додаток слухає порт 8080, він буде доступний за адресою:  
http://localhost:8080

(або інший порт, залежно від твоєї Java-програми)

## 🔍 ****6.6 Перевірка статусу****

docker ps

Побачиш активний контейнер, який працює на базі образу my-java-app.

Переглянути лог:

docker logs <container\_id>

## 🧹 ****6.7 Зупинка та видалення****

docker stop <container\_id>

docker rm <container\_id>

Образ видалити:

docker rmi my-java-app

## 🧺 ****Пункт 7: Volumes — збереження даних у Docker****

### 🎯 ****Мета:****

* Пояснити, чому дані в контейнері тимчасові
* Навчити використовувати **volume** для збереження даних
* Показати на прикладі бази даних (MySQL)

## 💡 ****7.1 Проблема: Куди зникають дані?****

За замовчуванням, усі зміни всередині контейнера зникають після його зупинки або видалення.

Приклад:

1. Створили файл всередині контейнера
2. Зупинили контейнер
3. Запустили новий — **файла більше немає**

## 📦 ****7.2 Що таке Volume?****

**Volume** — це спеціальне сховище, яке Docker створює на хості.  
Воно **існує незалежно від життєвого циклу контейнера**.

📌 Volume можна:

* створити вручну
* монтувати до будь-якого контейнера
* спокійно зберігати дані між запусками
* ділити між кількома контейнерами

## 🔧 ****7.3 Команди для роботи з Volumes****

# Створити volume

docker volume create mydata

# Подивитися всі volumes

docker volume ls

# Інформація про конкретний volume

docker volume inspect mydata

# Видалити volume (тільки якщо не використовується)

docker volume rm mydata

## 🧪 ****7.4 Приклад: запуск MySQL з volume****

docker run -d \

--name mysql-db \

-e MYSQL\_ROOT\_PASSWORD=toha \

-v mysql\_data:/var/lib/mysql \

-p 3306:3306 \

mysql:5.7

🔍 Пояснення:

| Параметр | Що робить |
| --- | --- |
| -e | Задає змінну середовища (пароль root) |
| -v | Прив'язує volume mysql\_data до папки в контейнері |
| /var/lib/mysql | Де MySQL зберігає свої дані |
| -p | Проброс порту на хост |

📌 Volume mysql\_data буде зберігати БД навіть після зупинки або видалення контейнера.

## 🧠 ****7.5 Перевірка****

docker volume ls # бачимо mysql\_data

docker volume inspect mysql\_data

🔍 У inspect побачимо:

* де фізично зберігаються дані (на хості)
* які контейнери використовують volume

## 🧹 ****7.6 Як показати, що volume зберігає дані****

1. Запускаєш MySQL із volume
2. Створюєш нову таблицю або додаєш дані
3. Зупиняєш і видаляєш контейнер:

docker stop mysql-db

docker rm mysql-db

1. Запускаєш **новий контейнер** з тим же volume:

docker run -d \

--name mysql-db-new \

-e MYSQL\_ROOT\_PASSWORD=toha \

-v mysql\_data:/var/lib/mysql \

-p 3306:3306 \

mysql:5.7

1. Дані залишилися ✅

## 📁 ****7.7 Альтернатива: використання bind-монту (локальна папка)****

docker run -d \

-v $(pwd)/data:/data \

busybox \

sh -c "echo 'Hello from Docker!' > /data/hello.txt"

➡️ Файл hello.txt буде створений у вашій локальній папці ./data.

## 🧾 ****Пункт 8: docker-compose — багатоконтейнерна система****

### 🎯 ****Мета:****

* Пояснити, що таке docker-compose
* Навчити писати docker-compose.yml
* Запустити одразу кілька контейнерів у зв'язці (наприклад, **Java API + БД**)
* Навчити керувати стеком з кількох сервісів

## 🧠 ****8.1 Що таке docker-compose?****

docker-compose — це інструмент для **опису й запуску декількох контейнерів** за допомогою **єдиного YAML-файлу**.

📌 Це:

* зручно для проєктів, де багато сервісів (API, БД, кеш, фронт…)
* автоматично створює спільну мережу між контейнерами
* дозволяє запускати все однією командою

## 🧩 ****8.2 Типовий стек: Java + БД****

**Мета прикладу** — запустити:

* Spring Boot додаток (готовий .jar)
* PostgreSQL
* Автоматичне з’єднання через мережу

## 📁 ****8.3 Структура проєкту****

/my-compose-java-app

├── app.jar

├── Dockerfile

├── docker-compose.yml

## 🔨 ****8.4 Dockerfile (Java)****

Dockerfile

FROM openjdk:17

WORKDIR /app

COPY app.jar app.jar

CMD ["java", "-jar", "app.jar"]

## ⚙️ ****8.5 docker-compose.yml****

yaml

version: '3.9'

services:

app:

build: .

ports:

- "8080:8080"

depends\_on:

- db

environment:

SPRING\_DATASOURCE\_URL: jdbc:postgresql://db:5432/mydb

SPRING\_DATASOURCE\_USERNAME: postgres

SPRING\_DATASOURCE\_PASSWORD: postgres

db:

image: postgres:14

environment:

POSTGRES\_DB: mydb

POSTGRES\_USER: postgres

POSTGRES\_PASSWORD: postgres

volumes:

- pgdata:/var/lib/postgresql/data

volumes:

pgdata:

## 🔍 ****8.6 Пояснення структури****

| Елемент | Пояснення |
| --- | --- |
| version | Версія синтаксису docker-compose |
| services | Всі сервіси, які ми запускаємо (у нас app і db) |
| build | Побудувати образ із Dockerfile |
| image | Образ, який буде використано (Postgres) |
| ports | Проброс портів з хосту в контейнер |
| environment | Змінні середовища (для Spring Boot і Postgres) |
| depends\_on | Вказує, що спочатку треба запустити db |
| volumes | Використання volume для збереження даних Postgres |

## 🚀 ****8.7 Запуск проєкту****

docker-compose up

📌 Це:

* будує образ з Dockerfile
* запускає обидва сервіси
* створює спільну мережу
* зв’язує API з БД автоматично

Виведе лог в консоль для обох контейнерів.

## 🔚 ****8.8 Завершення роботи****

docker-compose down

📌 Зупиняє та видаляє всі контейнери, які були створені через up.

## 🚀 ****8.9 Фоновий запуск****

docker-compose up -d

📌 Запускає сервіси **у фоні**. Перевірити можна так:

docker ps

## 🛠 ****8.10 Додаткові команди****

docker-compose build # Перебудувати образ

docker-compose logs # Подивитись логи

docker-compose stop # Зупинити сервіси, але не видаляти

docker-compose restart # Перезапустити

## ✅ ****Міні-завдання для студентів:****

1. Створити структуру my-compose-java-app
2. Покласти app.jar і написати Dockerfile
3. Створити docker-compose.yml з Postgres
4. Запустити:

docker-compose up

1. Перевірити, що обидва сервіси працюють (docker ps)
2. Подивитися логи:

docker-compose logs app

## 💡 ****Бонус-ідеї, якщо є час:****

* Показати, як додати adminer для перегляду БД в браузері:

yaml

adminer:

image: adminer

ports:

- 8081:8080

→ http://localhost:8081 — вебінтерфейс до Postgres