Docker

Веб-орієнтовані системи і технології

Лекція 2

**Проблеми, що можуть виникати при розробці**

* Велика кількість інструментів. Необхідно пам'ятати їх версії, налаштовувати середовище розробки, застаріла документація
* Розгортання (деплой): потрібні точні версії, декілька проектів можуть вимагати різних версій, складно видаляти інструменти, неможливість підтримувати різні версії

[**Why Docker?**](https://www.docker.com/why-docker)

**Докер** – це платформа для керування інфраструктурою додатку, що розробляється. В інфраструктуру входять *мови розробки*, *фреймворки*, *архітектури*, *API* тощо.

**Докер** вирішує такі задачі:

* Не потрібно пам'ятати, які інструменти яких версій потрібно встановити (що актуально для тривалих термінів розробки, участі декількох розробників та наявності локальних версій)
* Розробка і розгортання: консистентне оточення, легке розгортання, одні і ті ж інструменти
* Велика інфраструктура

**Інструменти, що будуть нами використовуватись**

* Docker – низькорівневий набір інструментів
* Docker-compose – високорівневий інструмент для керування докер-контейнерами. В ньому ми декларативно описуємо правила контейнера

Інструменти, що не будуть нами використовуватись (використовуються для великих проектів)

* Docker-machine
* Docker-swarm
* Kubernets

**Планування проекту**

* **Сервіс #1** – Фронтенд (React, CreateReactApp)
* **Сервіс #2** – API (Node)
* **Сервіс #3** – AuthAPI (авторизація, Node)
* **Сервіс #4** – API БД (MongoDB, для зберігання сутностей проекту – публікації, категорії)
* **Сервіс #5** – API авторизації в БД (окрема БД для користувачів та авторизації
* **Сервіс #6** – Mailer
* **Сервіс #7** – Nginx (проксірування запитів, кешування)

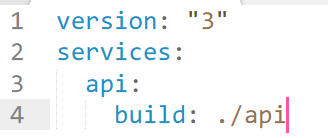
**Установка інструментів**

* Установити Docker Desktop
* Перевірити виконання команд docker, docker-compose (docker --help, docker --version, docker-compose --version, docker-compose --help)
* Для наочності (міграції проекту всередину докер-контейнера) встановити NodeJS (node -v)

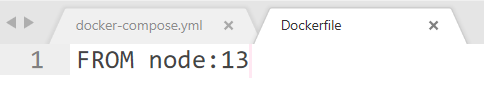
**Створення docker-compose.yml, Dockerfile**

Створити папку з проектом (realworld-docker)

Створимо файл з конфігурацією для наших сервісів ***docker-compose.yml***



* В папці api створимо Dockerfile – текстовий документ, що містить всі команди, що користувач може використовувати для збірки образу

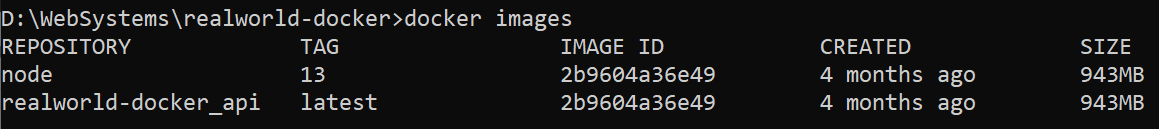


**Побудова образу Docker image**

* Для побудови образу Docker image виконати команду docker-compose build
* Після цього створиться образ Linux із встановленою NodeJS для нашого сервісу API

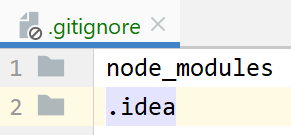
Команда docker images

* Realworld-docker\_api – Докер-образ, який був створений на основі Node:13



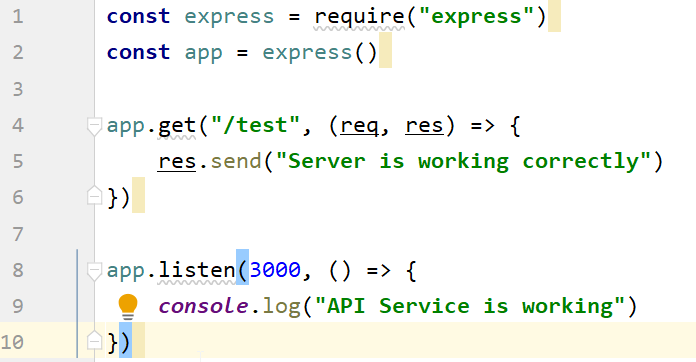
**Створення демонстраційного API-додатку**

* На рівні папки *realworld-docker* виконати npm-init
* Встановити *Express* – міні-фреймворк для NodeJS
* Ініціювати репозиторій git init
* Створити файл .gitignore і додати папки *node\_modules* та *.idea* для їх ігнорування в системі керування версіями

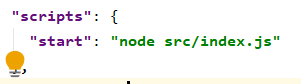


Створення проекту

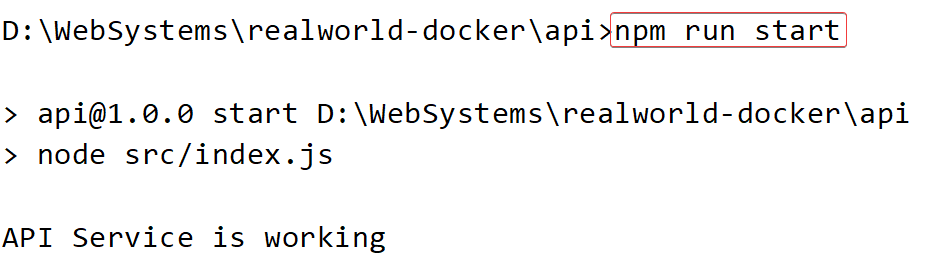
* Виконавчий файл src/index.js

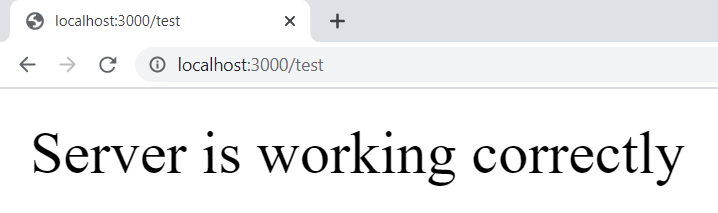


* Скрипт для запуску проекту в режимі продакшина



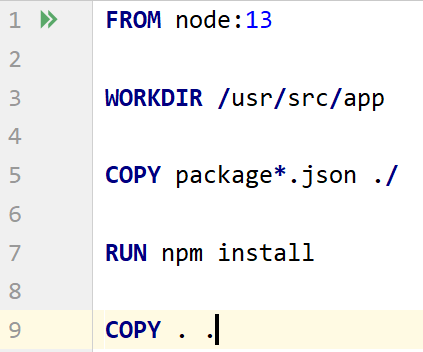
* Запуск проекту (перевірка)





**Підготовка і побудова Докер-образу для API сервісу**

* Задаємо робочу директорію для файлів проекту
* Копіюємо файли package та package-lock.json всередину Докер-образу
* Встановлюємо всі залежності
* Копіюємо файли проекту

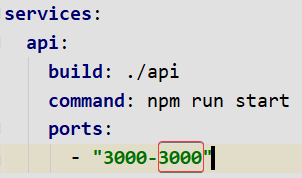


Після підготовки можна побудувати образ (docker-compose build)

* Якщо змін в конфігурації образу не відбувається, то повторне виконання команди побудови образу відбувається швидко (без змін)

**Налаштування і запуск API сервісу**

* В docker-compose.yml можна записувати порти і команди для запуску сервера:



Зліва записується зовншіній порт (порт localhost), справа – внутрішній порт (порт контейнера)

Запуск API сервера (docker-compose up)

* Для запуску використовуємо команду **docker-compose up**

Якщо потрібно побудувати образ і запустити сервіс виконуємо

**docker-compose up --build**

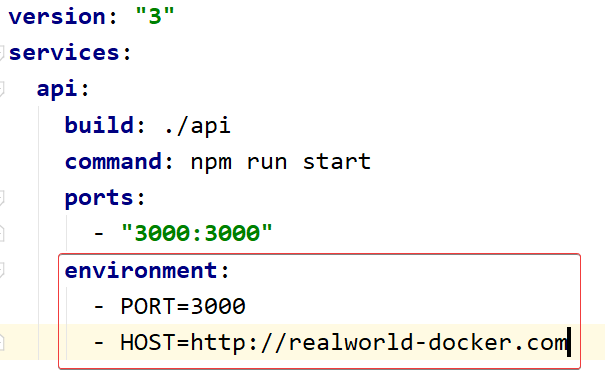
Docker Hub

* Docker Hub – репозиторій (веб-сайт) для докер-образів
* Alpine-версія для докер-образа NodeJS

**Створення і використання змінних оточення**

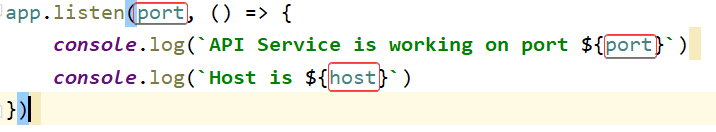
Такі змінні містять інформацію, що можуть використовувати виконавчі файли програм. Наприклад, порт, налаштування БД, конфігурація поштового серверу тощо.

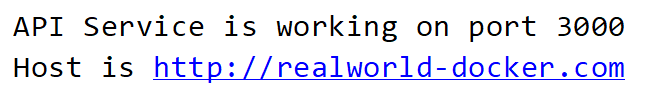
Для створення змінних оточення в docker-compose.yml потрібно записати:



Використання змінних оточення в коді

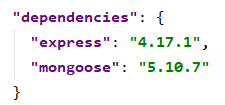






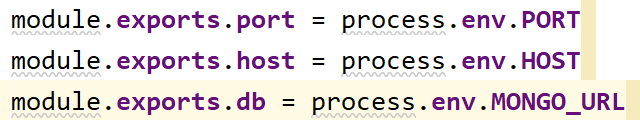
**Додаємо базу даних**

* MongoDB – нереляційна база даних
* Mongoose – модуль для роботи з БД
* Встановити Mongoose – npm install mongoose
* package.json:

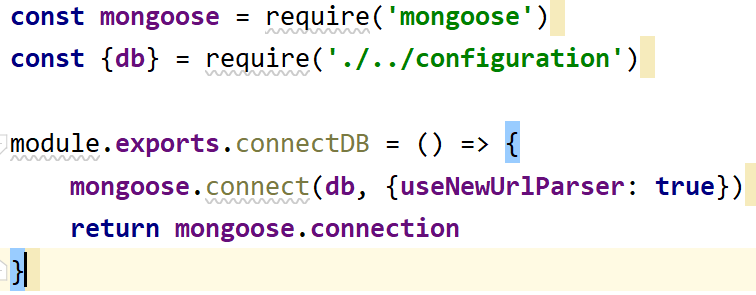


**Підключення до БД потрібно робити до запуску сервера**

* Виносимо конфігурацію в окремий файл configuration/index:



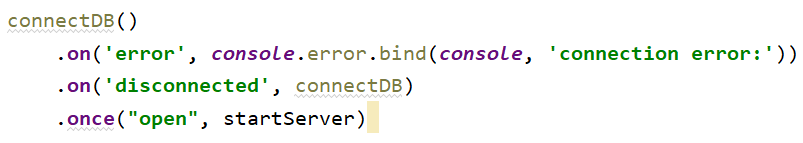
Створюємо допоміжний файл – хелпер для з'єднання з БД:



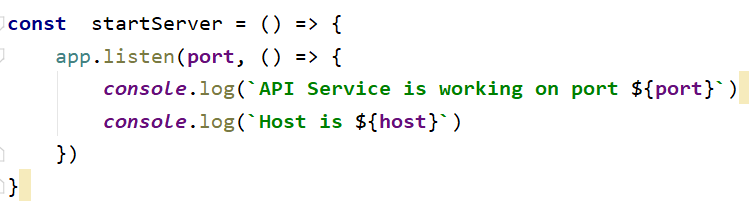
* Підключаємо конфігурацію



Коннектимось до БД



При успішному з'єднанні запускаємо сервер



**Додаємо новий сервіс api\_db**

* Нам не потрібно будувати новий образ, а лише використати існуючий (з mongoDB)
* Сервіс api запустити лише після запуску api\_db
* В змінній оточення MONGO\_URL використати адресу сервісу api\_db

