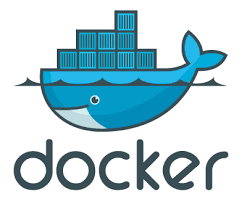
Посилання на github: <https://github.com/Ilya-Olenchenko/Script-programming.git>

**Лабораторна робота № 5.**



Використання Docker, для розгортання додатку Node.JS

Короткі теоретичні відомості

Docker являє собою систему управління контейнерами. Вона дозволяє «упакувати» додаток або веб-сайт з усім його оточенням і залежностями в контейнер, яким в подальшому можна легко і просто керувати: переносити на інший сервер, масштабувати, оновлювати.

Docker був написаний на мові програмування Go і випущений в 2013 році. Спочатку він працював тільки з Linux-системами, проте на даний момент його можна використовувати також в Windows і macOS. Незважаючи на те, що проект є відносно новим, Докер широко використовується багатьма фахівцями і продовжує завойовувати популярність.

Контейнери (containers) являють собою засоби інкапсуляції додатку разом з його залежностями. На перший погляд контейнери можуть здаватися усього лише спрощеною формою віртуальних машин (virtual machines - VM) - як і віртуальна машина, контейнер містить ізольований екземпляр операціонної системи (ОС), який можна використовувати для запуску додатків. Але контейнери мають деякі переваги, що забезпечують такі варіанти використання, які важко або неможливо реалізувати в звичайних віртуальних машинах:

* контейнери спільно використовують ресурси основної ОС, що робить їх напорядок більш ефективними. Контейнери можна запускати і зупиняти за долі секунди. Для додатків, що запускаються в контейнерах, накладні витрати мінімальні або взагалі відсутні, в порівнянні з додатками, що запускаються безпосередньо під управлінням основної ОС;
* переносимість контейнерів забезпечує потенційну можливість усунення цілого класу програмних помилок, що викликаються незначнимними змінами робочого середовища, це позбавляє обґрунтування позиції розробника: «*але це працює на моєму комп'ютері*»;
* спрощена сутність контейнера означає, що розробники можуть одночасно запускати десятки контейнерів, що дає можливість імітації роботи промислової розподіленої системи. Інженери по експлуатації можуть запустити на одному хості набагато більше контейнерів, ніж при використанні окремих віртуальних машин;
* окрім того, контейнери надають переваги кінцевим користувачам і розробникам без необхідності розгортання програми в хмарі. Користувачі можуть завантажувати і запускати складні додатки без багатогодинної метушні з конфігурацією і проблемами при установціі, при цьому не турбуватися про будь-які зміни в їх локальних системах. У свою чергу, розробники подібних додатків можуть уникнути проблем, пов'язаних з відмінностями в конфігураціях призначених для користувача середовищ і з доступністю залежностей для цих додатків.

І що більш важливо, існують принципові відмінності у меті використання віртуальних машин і контейнерів.

Метою застосування ***віртуальної машини*** є повна емуляція чужорідного програмного (операційного) середовища, тоді як мета застосування ***контейнера*** - зробити додатки такими, що переносяться і самодостатніми.

Розглянемо три основні терміни в екосистемі Docker:

**Образ (image)** - шаблон, який використовується для створення контейнерів. Являє собою зліпок файлової системи, в якому розташований код програми та його оточення.

**Реєстр (registry)** - репозиторій образів. Docker Hub, - це публічний репозиторій, де зберігається величезна кількість образів.

**Контейнер (container)** - запущений додаток, тобто сукупність процесів і образу.

Хід роботи

**Результати виконання роботи**

Лістинг 1.1 файл app.js.

const mongoose = require("mongoose");

const express = require("express");

const Schema = mongoose.Schema;

const app = express();

const jsonParser = express.json();

const {

    MONGO\_DB\_HOSTNAME,

    MONGO\_DB\_PORT,

    MONGO\_DB

} = process.env

const options = {

    useUnifiedTopology: true,

    useNewUrlParser: true,

    useFindAndModify: false

}

const url = `mongodb://${MONGO\_DB\_HOSTNAME}:${MONGO\_DB\_PORT}/${MONGO\_DB}`;

const studentScheme = new Schema(

    {

        name: String,

        lastname: String,

        age: Number,

        group: String,

    },

    { versionKey: false }

);

const Student = mongoose.model("Student", studentScheme);

app.use(express.static(\_\_dirname + "/public"));

mongoose.connect(url, options, function (err) {

    if (err) return console.log(err);

    app.listen(3000, function () {

        console.log("Сервер запущено...");

    });

});

app.get("/api/students", function (req, res) {

    Student.find({}, function (err, students) {

        if (err) return console.log(err);

        res.send(students)

    });

});

app.get("/api/students/:id", function (req, res) {

    const id = req.params.id;

    Student.findOne({ \_id: id }, function (err, students) {

        if (err) return console.log(err);

        res.send(students);

    });

});

app.post("/api/students", jsonParser, function (req, res) {

    if (!req.body) return res.sendStatus(400);

    const studentName = req.body.name;

    const studentLastName = req.body.lastname;

    const studentAge = req.body.age;

    const studentGroup = req.body.group;

    const student = new Student({ name: studentName, lastname: studentLastName, age: studentAge, group: studentGroup });

    student.save(function (err) {

        if (err) return console.log(err);

        res.send(student);

    });

});

app.delete("/api/students/:id", function (req, res) {

    const id = req.params.id;

    Student.findByIdAndDelete(id, function (err, student) {

        if (err) return console.log(err);

        res.send(student);

    });

});

app.put("/api/students", jsonParser, function (req, res) {

    if (!req.body) return res.sendStatus(400);

    const id = req.body.id;

    const studentName = req.body.name;

    const studentLastName = req.body.lastname;

    const studentAge = req.body.age;

    const studentGroup = req.body.group;

    const newStudent = { age: studentAge, lastname: studentLastName, name: studentName, group: studentGroup };

    Student.findOneAndUpdate({ \_id: id }, newStudent, { new: true }, function (err, student) {

        if (err) return console.log(err);

        res.send(student);

    });

});

Лістинг 1.1 файл docker-compose.yaml.

version: "3"

volumes:

  mongo\_db\_data:

services:

  my\_mongodb:

    image: mongo:latest

    volumes:

      - mongo\_db\_data:/data/db

  app:

    build: .

    ports:

      - 3000:3000

    environment:

      - MONGO\_DB\_HOSTNAME=my\_mongodb

      - MONGO\_DB\_PORT=27017

      - MONGO\_DB=usersdb

На рисунку 1 зображено завантажені образи та список запущених контейнерів.

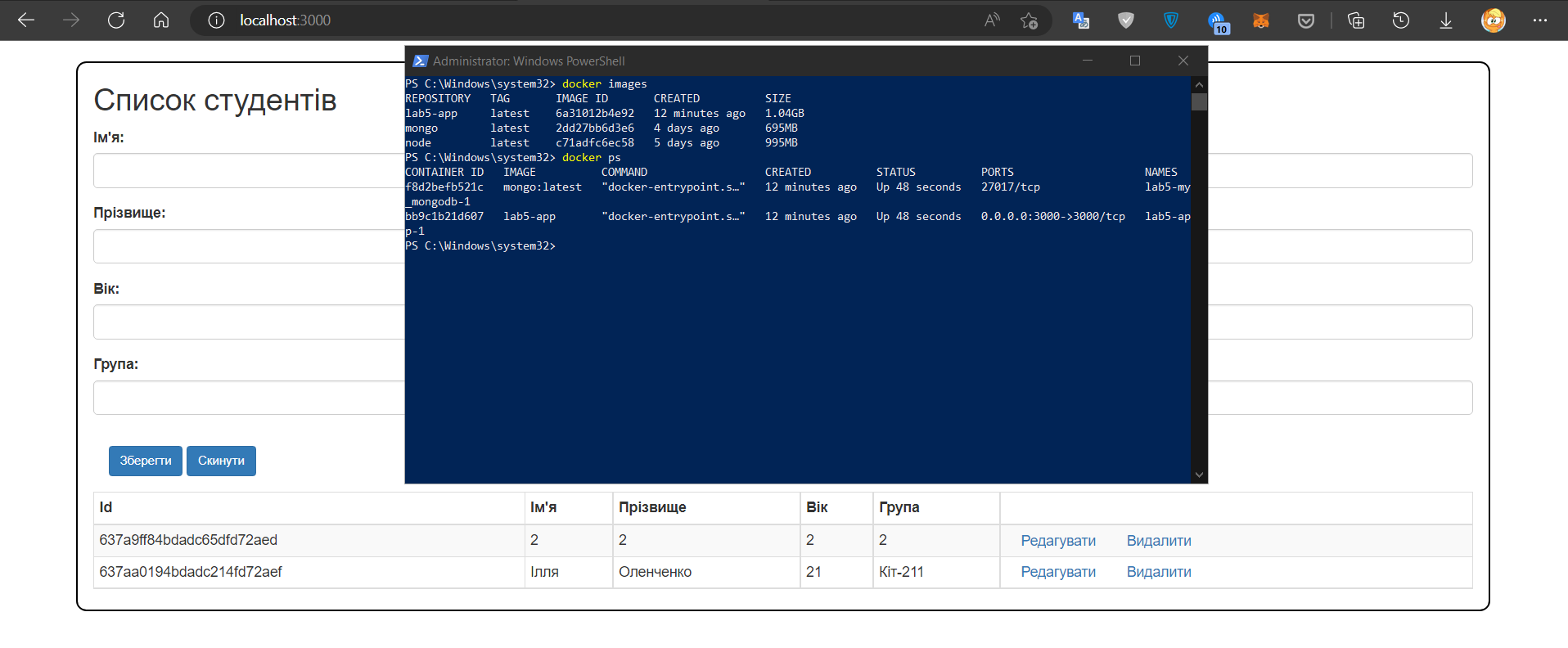


Рисунок 1 - Зображено завантажені образи, список запущених контейнерів

На рисунку 2 зображено запущений localhost:3000.

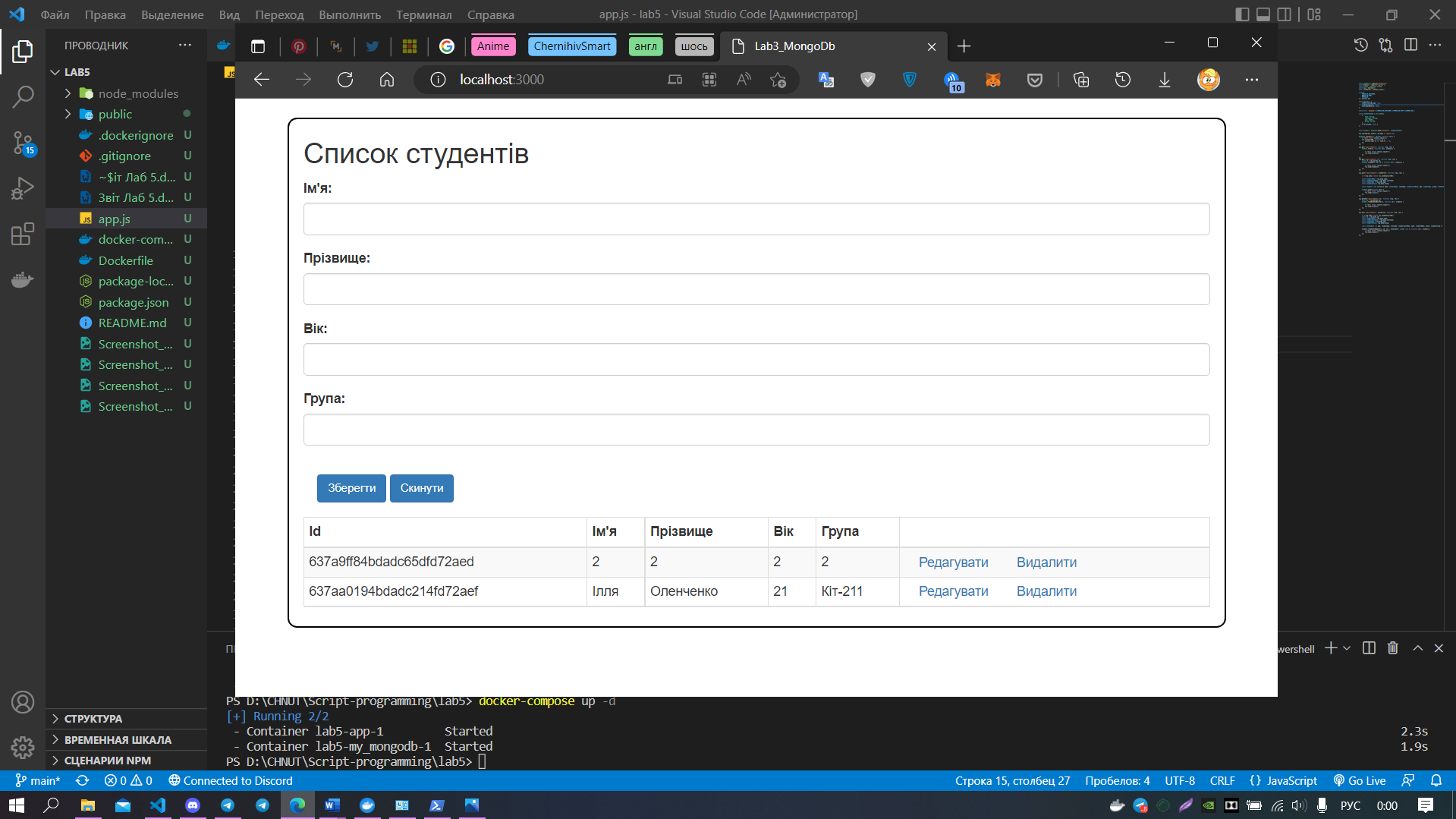


Рисунок 2 - Зображено запущений localhost:3000.

**Висновок:** Ми навчились отримувати повністю ізольовані додатоки, якиі створений на базі контейнерів Docker