Національний університет “Острозька академія”

Економічний факультет

Кафедра економіко-математичного моделювання та інформаційних технологій

Довгострокова робота з курсу

з курсу “WCF”

**Docker для .NET проєктів**

Виконала:

студентка 3-го курсу

групи КН-32

Киричок Ю. Я.

Оцінка \_\_\_\_\_ \_\_\_\_\_\_

Острог, 2022

**Зміст**

[Теоретична частина](#_heading=h.gjdgxs) **3**

[Практична частина](#_heading=h.3znysh7) **4**

[Висновки](#_heading=h.tyjcwt) **7**

[Список використаних джерел](#_heading=h.dx1awkmi48t) **7**

# Теоретична частина

Тема мого проєкту – Docker для .NET проєктів. Почнемо з теорії і з бази: що з взагалі таке Docker і для чого його використовують?

Docker – це ПЗ (програмне забезпечення) для управління контейнерами, коли контейнер – це певне ізольоване середовище, яке працює на тому ж сервері, що і основне середовище.

Механіка дуже схожа на віртуальну машину, але якщо віртуальна машина робить образ окремого комп’ютера, то контейнер – тільки ОС. Це дозволяє займати менше пам’яті та ефективніше використовувати ресурси комп’ютера. Якщо у вас проблема з кількістю оперативної пам’яті на комп’ютері, то контейнери у цьому випадку будуть кращим рішенням проблеми.

Яку проблему взагалі вирішує Docker? Він дозволяє перевірити та виріщити запуск вашого проєкту на іншій ОС. Тобто, якщо ви спостерігаєте ситуацію, що все добре працює на Windows чи на macOS, але на Linux все крашиться – у вас є змога створити віртуальний імейдж (образ) в Докері вашого проєкту та налаштувати його на проблемну оперативну систему.

Розберемо, як це робиться в теорії:

* Перш за все, встановлюєте Docker Dekstop. Тут проблем не має бути, інструкцію по встановленню можна знайти на офіційному сайті Докера.
* Створюєте Dockerfile. Там описуєте те, як буде створюватись ваш образ.
* Створюєте Image – образ для контейнеру.
* Далі створення самого контейнеру – запущений образ, в якому вже відпрацьовує ваш image так, як ви його налаштували.

Прикріплю схему для візуалізації описаного вище:



Рис1. Схема запуску Docker контейнеру

Основні переваги Docker:

* мінімально використовує ресурси;
* зручно приховує фонові процеси;
* просто масштабує;
* зменшує час між написанням і запуском коду;
* швидше тестує;
* швидко розгортає;
* швидше створює додатки.

Можна сказати, що контейнеризація дає нам можливість дуже гнучко підганяти необхідні налаштування під наші програми, при цьому не поламати глобальні налаштування нашого сервера.

# Практична частина

Перш за все, у мене є проєкт, написаний на C# з технологією API та базою даних SQLite – менеджер іграшок в дитсадку. Також має технологію Swagger в собі.

Він написаний за стандартами, тому сам код тут розписувати не буду. Задача – огорнути його в Docker.

Йдемо по теоретичних кроках, що я описувала вище, але вже на практиці:

1. Встановлення Docker Dekstop. Тут проблем не було, ось так він виглядає:

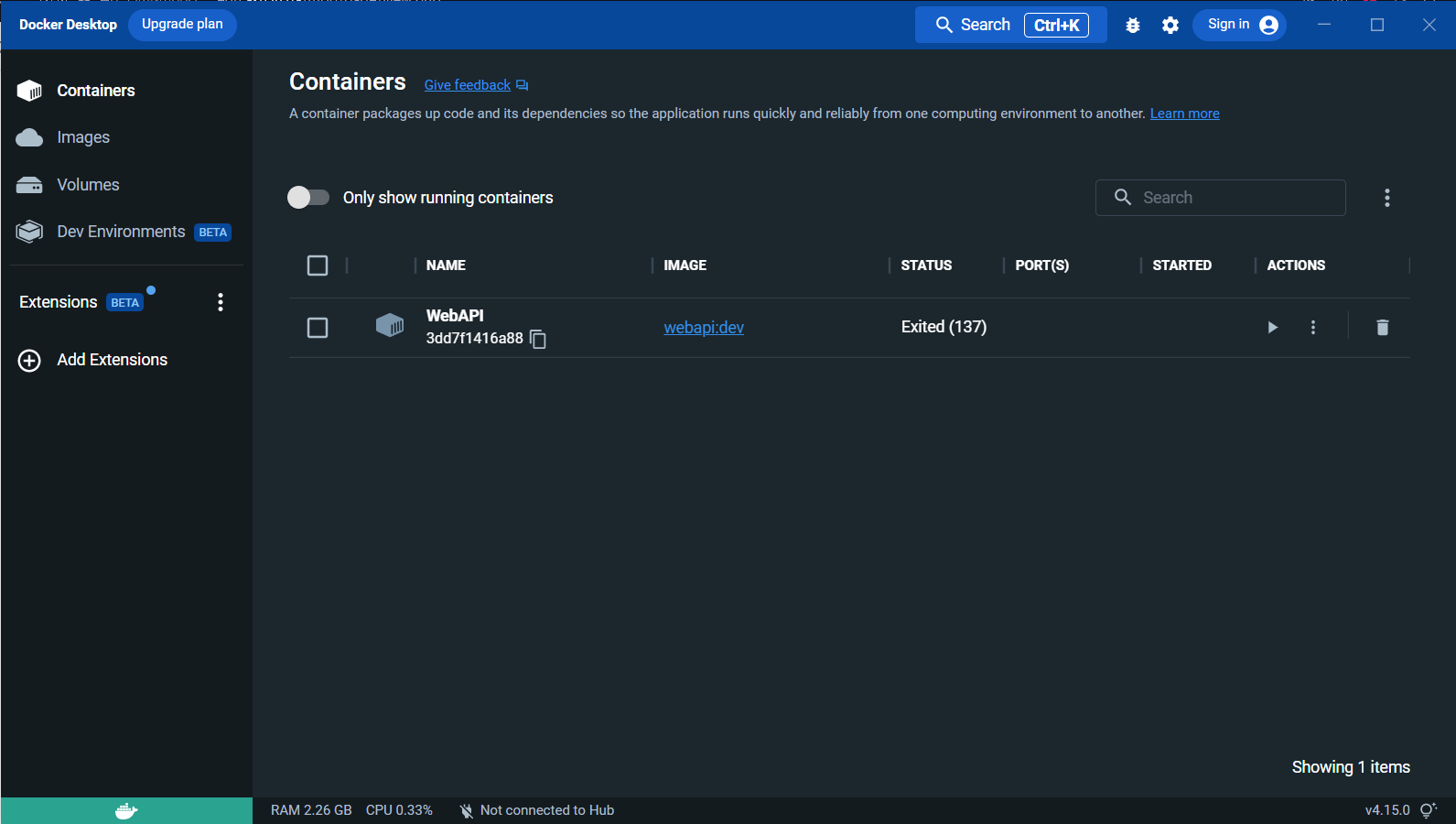


Рис2. Результат встановлення Docker Dekstop

1. Створення Dockerfile. Налаштовуємо його попередньо на Linux ОС через Add > Docker support > Linux.

Лістинг 1. Код Dockerfile

| #See https://aka.ms/containerfastmode to understand how Visual Studio uses this Dockerfile to build your images for faster debugging.  FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base  WORKDIR /app  EXPOSE 80  EXPOSE 443  FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build  WORKDIR /src  COPY ["WebAPI.csproj", "."]  RUN dotnet restore "./WebAPI.csproj"  COPY . .  WORKDIR "/src/."  RUN dotnet build "WebAPI.csproj" -c Release -o /app/build  FROM build AS publish  RUN dotnet publish "WebAPI.csproj" -c Release -o /app/publish /p:UseAppHost=false  FROM base AS final  WORKDIR /app  COPY --from=publish /app/publish .  ENTRYPOINT ["dotnet", "WebAPI.dll"] |
| --- |

Покажу ще не менш важливу частину – json файл підключення до БД.

Лістинг 2. Код appsettings.json

| {  "ConnectionStrings": {  "data": "Data source = KinderGardenDb.db"  },  "Logging": {  "LogLevel": {  "Default": "Information",  "Microsoft.AspNetCore": "Warning"  }  },  "AllowedHosts": "\*"  } |
| --- |

1. Створення Image:

Прописуємо команди в PowerShell:

docker build -t web\_api .

Після цієї команди створиться ваш образ. Перевірити його наявність можна через команду:

docker image ls



Рис3. Результат виконання команди

1. Створення контейнеру:

Компілюємо проєкт та запускаємо WebAPI через Docker. З’являється контейнер:

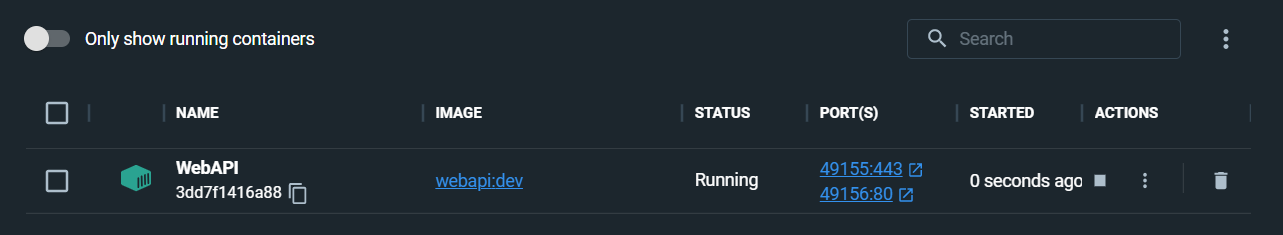


Рис4. Результат створення контейнеру

1. Результат:

Вигляд проєкту через WebAPI:

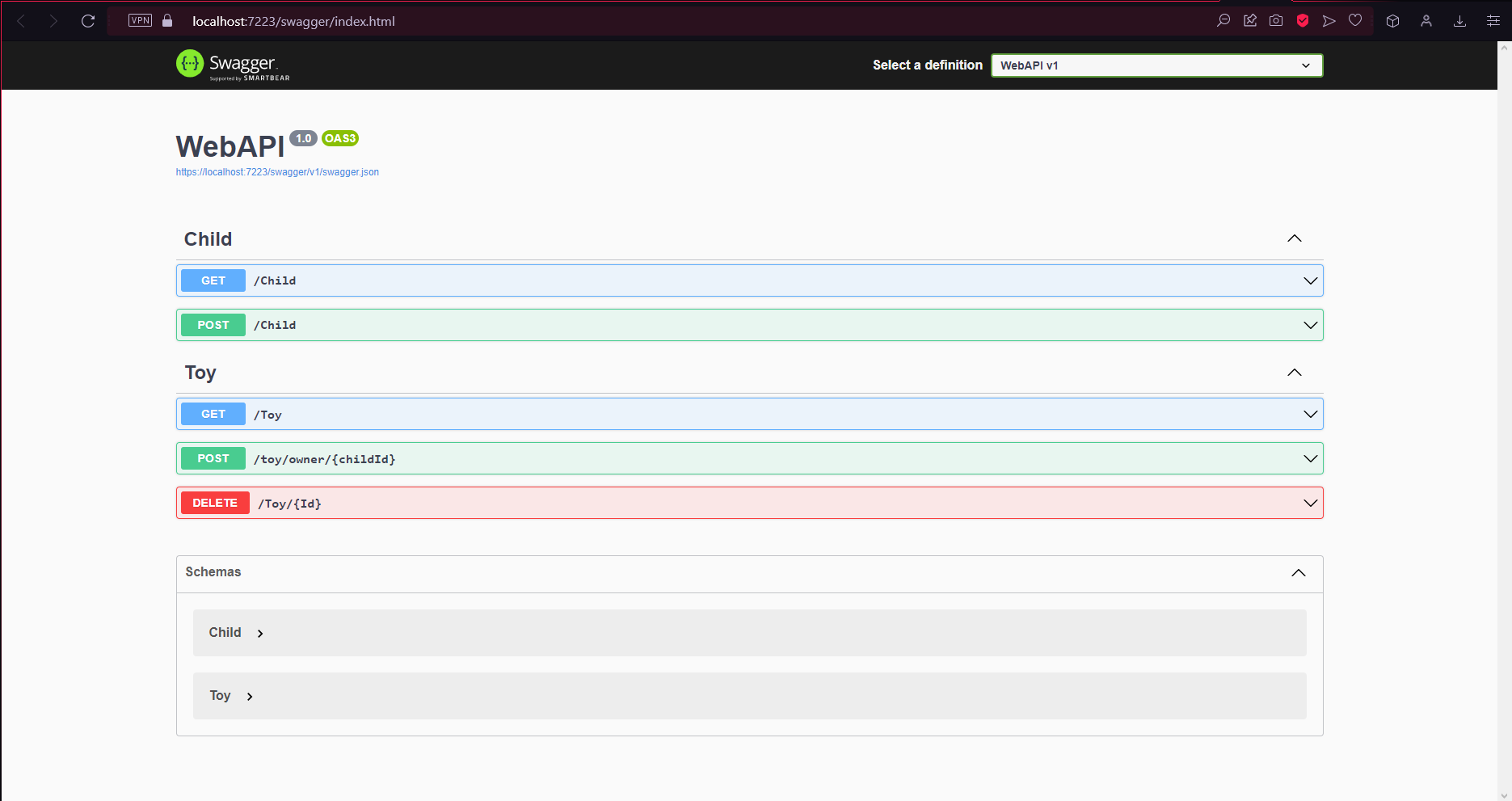


Рис5. Результат виконання програми через WebAPI

Вигляд проєкту через Docker:

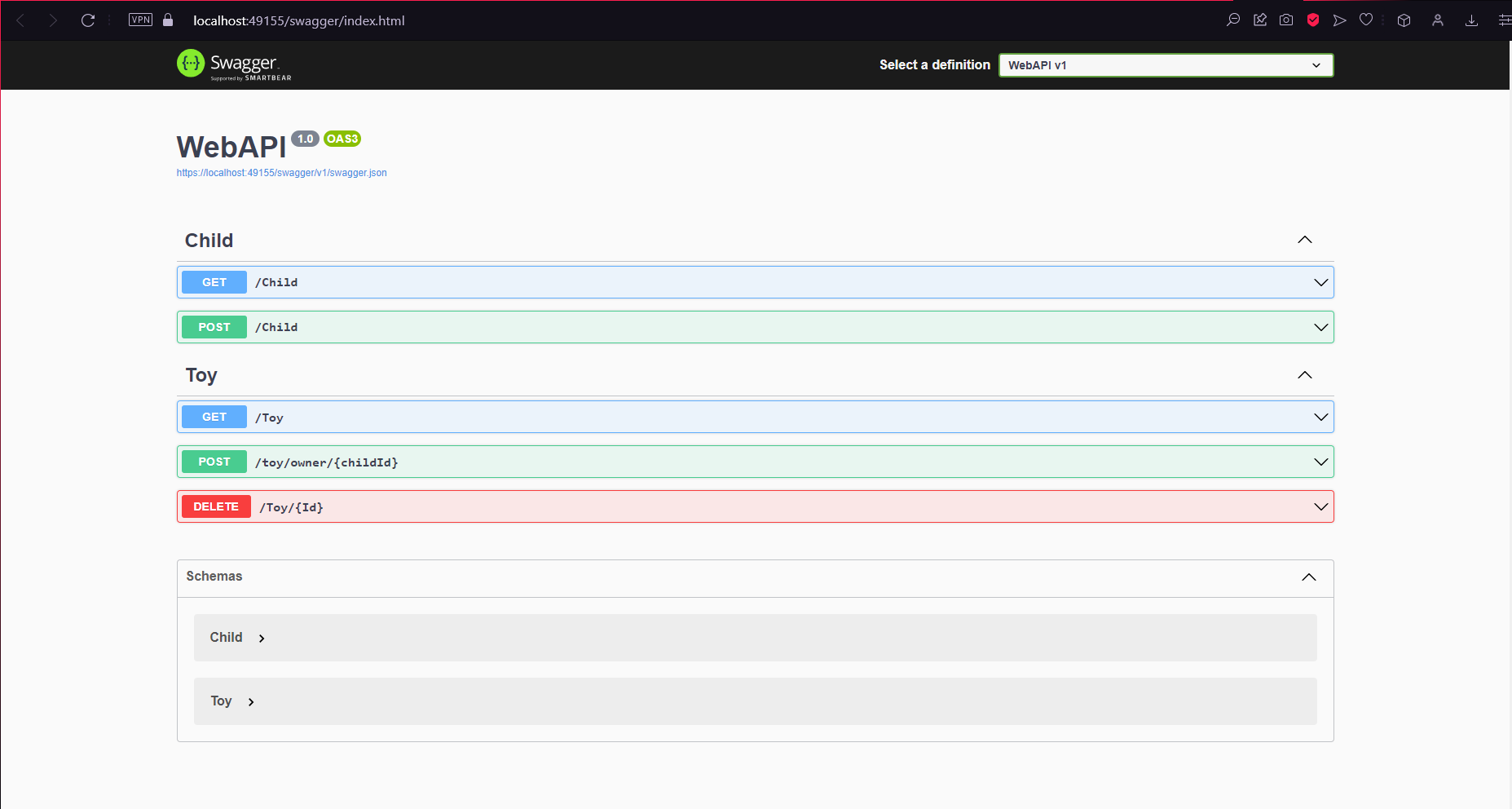


Рис6. Результат виконання програми через Docker

Працює!

# Висновки

Робота з Docker – не завжди проста, але корисна. Основними перевагами є можливість перевірити та налаштувати свою програми на іншій ОС та мале використання ресурсів комп’ютера при цьому відносно віртуальної машини.

З якими складнощами я зіткнулась під час роботи:

1. Не білдився імейдж. Переробила Dockerfile – все пофіксилось.
2. Не ставився Docker на мій проєкт, де БД на SQL Server, а не на SQLite. На жаль, тут так і не розібралась, не зрозуміла, як правильно налаштувати Dockerfile тут, щоб всі шляхи працювали правильно. Тому ставила на SQLite на проєкт, показаний вище.

В підсумку, було цікаво та корисно. Docker можна використовувати у багатьох проєктах та на багатьох мовах програмування. Він корисний як і для розгортання мікросервісів на різних ОС, так і на таких невеликих проєктах, як в мене.

# Список використаних джерел

* [Що таке Docker: простими словами про контейнеризацію](https://blog.ithillel.ua/articles/shcho-take-docker-prostimi-slovami-pro-konteynerizatsiyu)
* [Відмінності між ВМ та контейнерами](http://edu.asu.in.ua/mod/page/view.php?id=127)