Министерство науки и высшего образования Российской Федерации

ФГАОУ ВО «Уральский федеральный университет   
имени первого Президента России Б.Н. Ельцина»

Институт новых материалов и технологий

Кафедра «Теплофизика и информатика в металлургии»

**Контейнеризация программных приложений и брокер сообщений**

**ОТЧЕТ**

**по практической работе №1 по дисциплине**

**«Основы методологии Development Operation»**

Направление 09.03.02 «Информационные системы и технологии» (уровень бакалавриата)

Образовательная программа  
09.03.02/33.02 «Информационные системы и технологии» (СУОС)

Студент:

группы НМТ-413901 И.В.Машуков

Преподаватель:

профессор, д.т.н В.В.Лавров

Екатеринбург

2024

**Практическая работа №1**

**«Сборка и запуск контейнера в Docker»**

Ход работы:

1. Разработка тестового приложения Visual Studio .NET Core

Для начала необходимо создать новый проект в Visual Studio и выбрать шаблон, соответствующий типу приложения, который необходимо разработать – веб-приложение ASP.NET Core MVC (рис. 1).

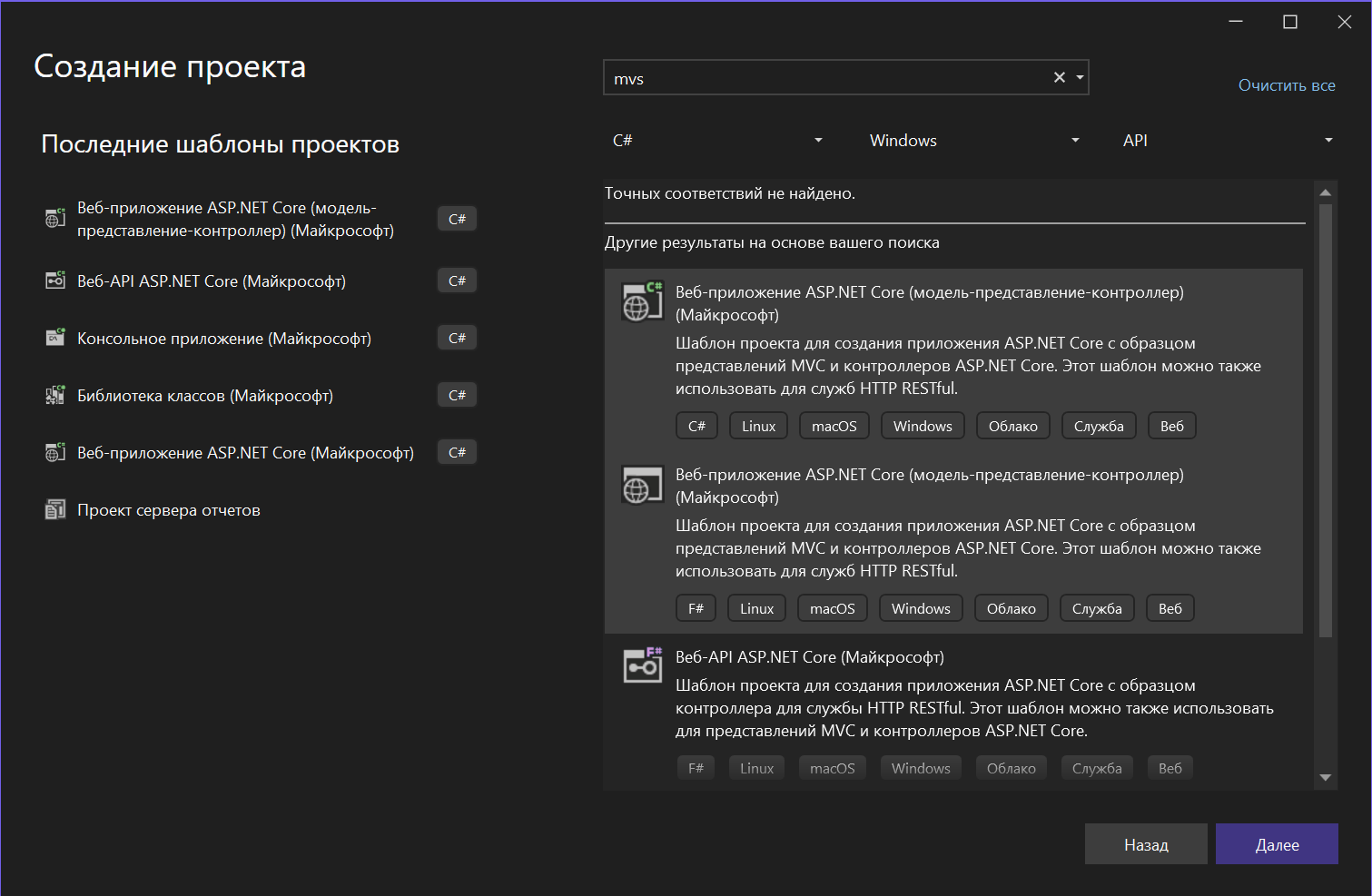


Рисунок 1 - Создание проекта в Visual Studio

Далее необходимо написать программный код приложения «Калькулятор» – простой калькулятор с веб-интерфейсом, который будет выполнять основные арифметические операции (сложение, разность, умножение и деление) и отображать результат. На главной странице веб-приложения, в футере должны отображаться ФИО и номер группы студента. Архитектура приложения «Калькулятор» представлена на рисунке 2.



Рисунок 2 - Архитектура приложения «Калькулятор»

Фрагмент Visual Studio c отображением листинга программного кода файла appsettings.json представлен на рисунке 3.



Рисунок 3 - Листинг программного кода файла appsettings.json

2.Создание контейнера с использованием Docker для тестового приложения ASP.NET Core Web Application, определение и настройка окружение контейнера с помощью Dockerfile

Создаем файл с именем "Dockerfile" в корневой папке тестового проекта (рис. 4).

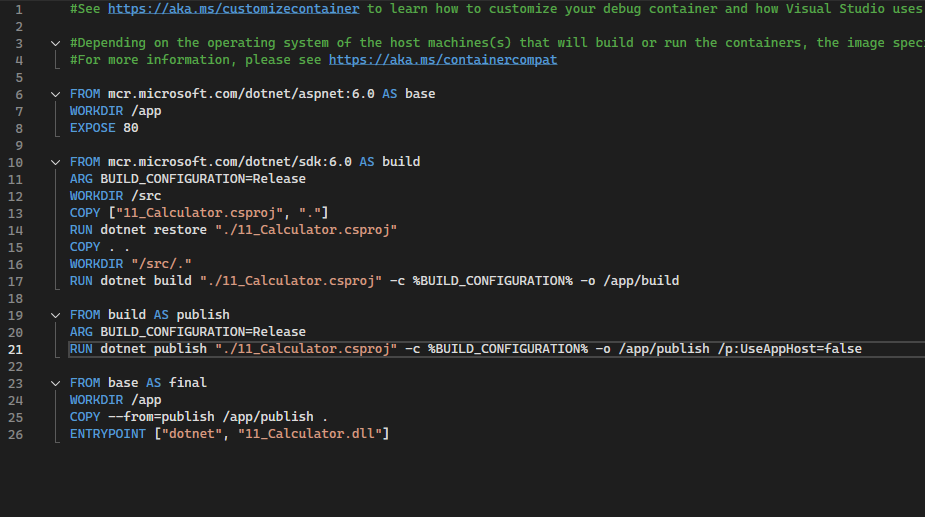


Рисунок 4 - Созданный Dockerfile

3. Размещение приложение в системе контроля версий GitHub

Когда тестовое приложение работает без ошибок, создаем личный репозиторий на GitHub. Выгруженный проект представлен на рисунке 5.

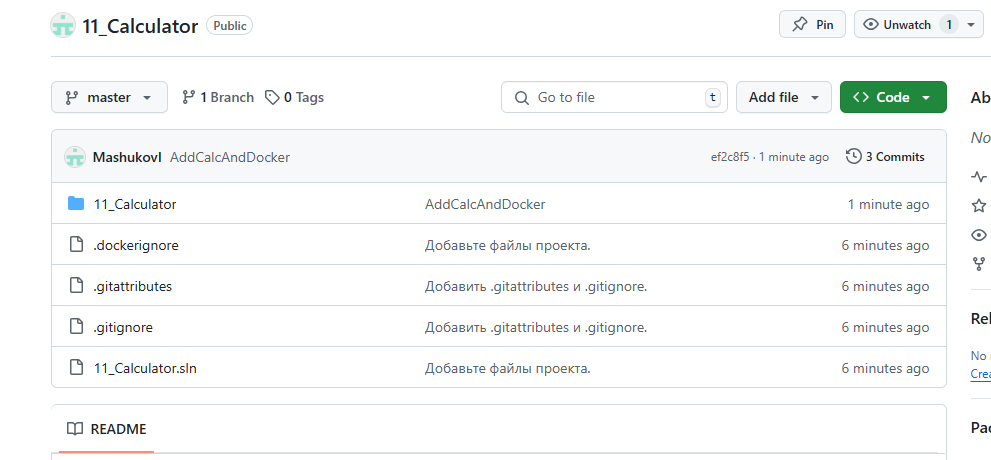


Рисунок 5 - Окно личного репозитория GitHub с размещением тестового проекта

4. Выполнение сборки образа с использованием команды docker build

Сначала в терминальном режиме на удаленном сервере необходимо клонировать проект из личного репозитория GitHub (рис. 6). Для этого выполняем команду:

git clone <HTTPS>

Здесь <HTTPS> – это HTTPS-адрес проекта в GitHub.

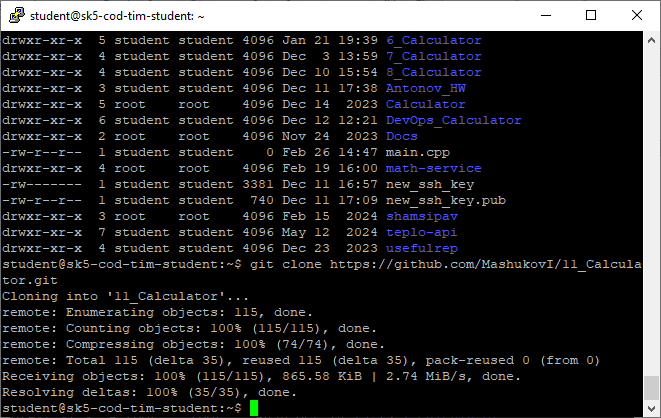


Рисунок 6 - Клонирование проекта

Далее осуществляем сборку Docker-образа (рис.7).

docker build -t <IMAGE\_NAME>:latest .

Здесь <IMAGE\_NAME> – это имя образа.

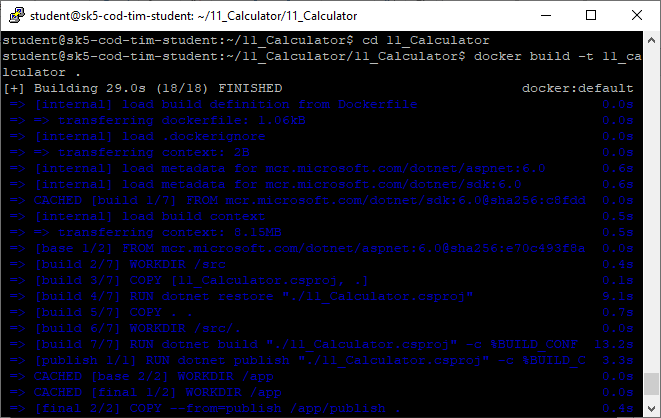


Рисунок 7 - Сборка Docker-образа

Проверяем существование созданного образа в списке активных контейнеров и их идентификаторов (рис. 8) с помощью команды:

docker ps -a

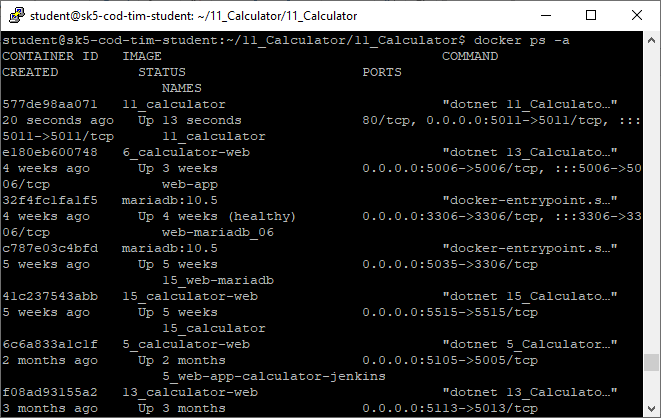


Рисунок 8 - Окно терминальной консоли с демонстрацией созданного образа

5. Создание контейнера с использованием Docker-образа и его выполнение

Для создания контейнера с использованием Docker-образа используем командой docker run (рис. 9):

docker run -d -p 5001:5001 --name <CONTAINER\_NAME> <IMAGE\_ID | IMAGE\_NAME>

Здесь:

-d означает, что контейнер будет запущен в фоновом режиме (daemon mode);

-p 5001:5001 пробрасывает порт 5001 на вашем хосте к порту 5001 в контейнере (на котором запущено ваше ASP.NET Core приложение).

<CONTAINER\_NAME> – это уникальное в рамках системы имя контейнера, по которому в дальнейшем его можно будет остановить, запустить или удалить;

<IMAGE\_ID> – это идентификатор или имя образа. Используем команду docker images -a для получения списка всех образов.

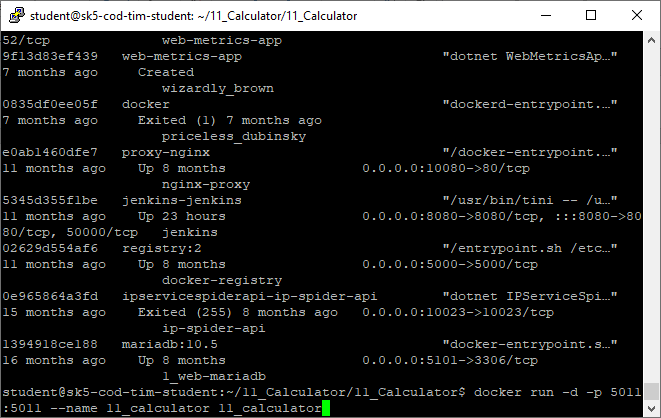


Рисунок 9 - Создание контейнера с использованием Docker-образа

Используем команду docker ps -a для получения списка активных контейнеров и их идентификаторов:

docker ps -a

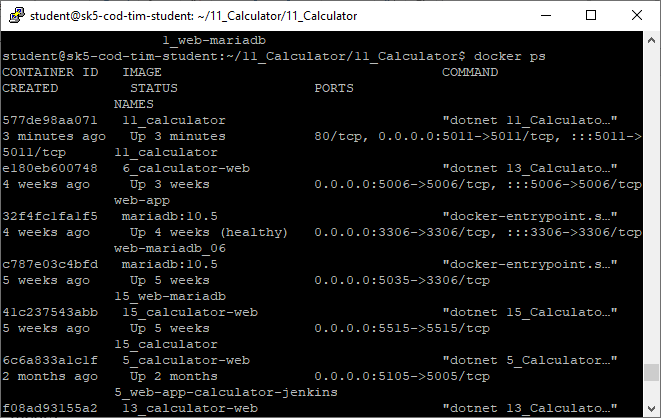


Рисунок 10 - Список активных контейнеров и их идентификаторов

По завершении работы в контейнере необходимо выйти из него, используя команду exit.

5. Демонстрация возможность публикации и остановки контейнера

Запускаем контейнер (рис. 11) с помощью команды:

docker start <CONTAINER\_ID или CONTAINER\_NAME>

Здесь <CONTAINER\_ID> – это уникальный идентификатор контейнера, или вы можете использовать <CONTAINER\_NAME>, если вы задали имя контейнеру при его запуске.

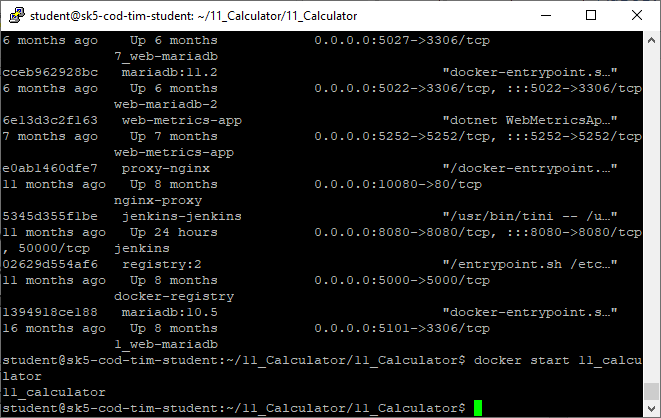


Рисунок 11 - Запуск контейнера

Убедимся в браузере, что веб-приложение работает (рис. 12).

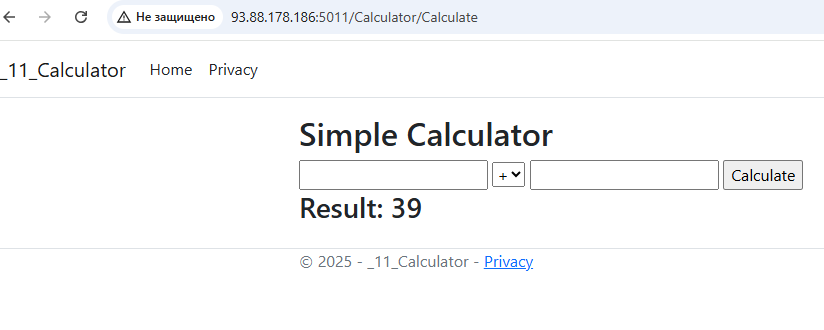


Рисунок 12 - Окно веб-браузера с демонстрацией работающего тестового приложения на выбранном порту

Остановим работающий контейнер (рис. 13) с помощью команды:

docker stop <CONTAINER\_ID или CONTAINER\_NAME>



Рисунок 13 - Остановка контейнера

Убедимся в браузере, что веб-приложение перестало работать (рис. 14).

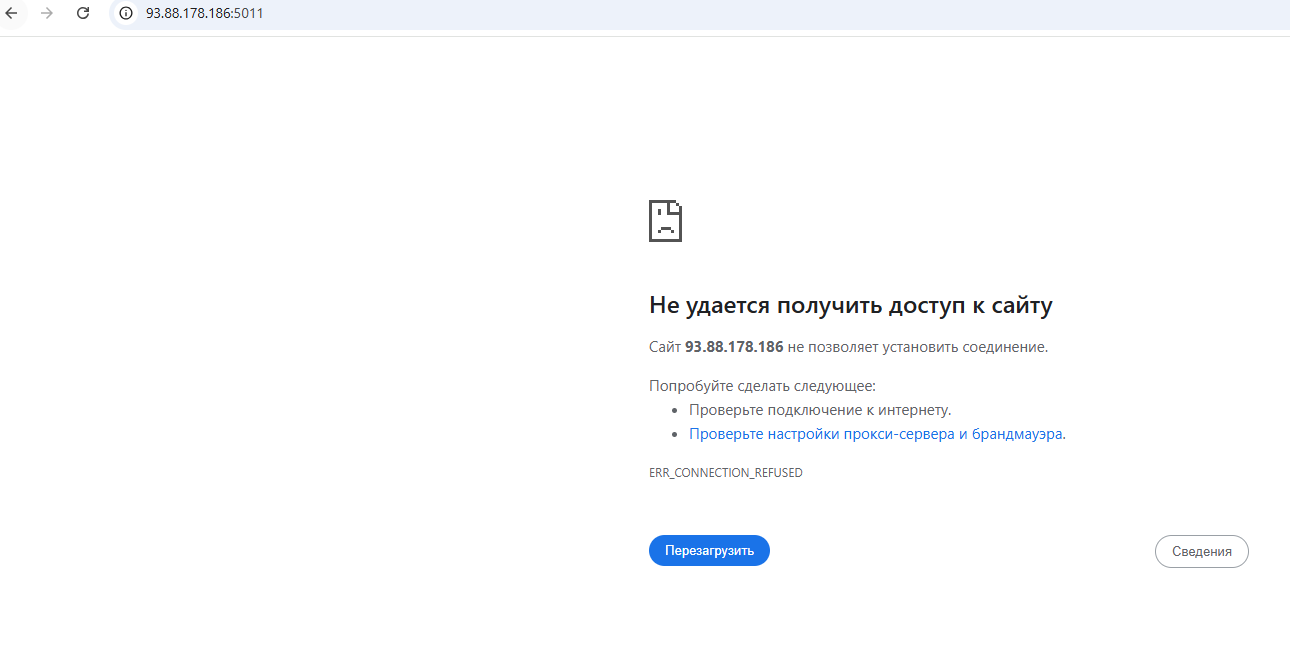


Рисунок 14 - Веб-приложение не работает

**Приложение А Листинг программного кода файла Calculator.Controllers**

using Microsoft.AspNetCore.Mvc;

namespace \_11\_Calculator.Controllers

{

public enum Operation { Add, Subtract, Multiply, Divide }

public class CalculatorController : Controller

{

[HttpGet]

public IActionResult Index()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Calculate(double num1, double num2, Operation operation)

{

double result = 0;

switch (operation)

{

case Operation.Add:

result = num1 + num2;

break;

case Operation.Subtract:

result = num1 - num2;

break;

case Operation.Multiply:

result = num1 \* num2;

break;

case Operation.Divide:

result = num1 / num2;

break;

}

var newRecord = new CalcModel

{

Num1 = num1,

Num2 = num2,

Operation = operation,

Result = result

};

await \_dbContext.CalcModels.AddAsync(newRecord);

await \_dbContext.SaveChangesAsync();

ViewBag.Result = result;

return View("Index");

}

}

}

**Приложение Б Листинг программного кода файла appsettings.json**

{

"Kestrel": {

"Endpoints": {

"Http": {

"Url": "http://0.0.0.0:5011"

}

}

},

"ConnectionStrings": {

"Main": "Server = mariadb; Database = 11\_WebApplicationDb; Uid = root; Pwd = password; Character set = utf8;"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

**Приложение В Листинг программного кода файла appsettings.Development.json**

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

}

}

**Приложение Г Листинг программного кода файла Program.cs**

using Microsoft.EntityFrameworkCore;

namespace \_11\_Calculator

{

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

}

app.UseStaticFiles();

app.UseRouting();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Calculator}/{action=Index}/{id?}");

app.Run();

}

}

}