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»

**Использование конвейера данных реального времени   
с Kafka**

**ОТЧЕТ**

**по практической работе № 3**

**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

СОДЕРЖАНИЕ

[Практическая работа «Использование конвейера данных реального времени с Kafka» 3](#_Toc152161890)

[1.1 Цель работы 3](#_Toc152161891)

[1.2 Ход проведения работы 3](#_Toc152161892)

[1.2.1 Разработка тестового приложения Visual Studio .NET Core 3](#_Toc152161893)

[1.2.2 Размещение приложения в системе контроля версий GitHub 5](#_Toc152161894)

[1.2.3 Сборка образов 5](#_Toc152161895)

[1.2.4 Установка программы HeidiSQL для сопровождения базы данных в СУБД MariaDB 6](#_Toc152161896)

[1.2.5 Настройка миграции базы данных в MariaDB 7](#_Toc152161897)

[1.2.6 Демонстрация работы приложения брокером сообщений Kafka 7](#_Toc152161898)

[1.3 Выводы 8](#_Toc152161899)

[Приложение А Листинг программного кода приложения «Калькулятор» (контроллер) 9](#_Toc152161900)

[Приложение Б Листинг программного кода приложения «Калькулятор» (KafkaConsumerService.cs) 11](#_Toc152161901)

[Приложение В Листинг программного кода приложения «Калькулятор» (appsettings.json) 13](#_Toc152161902)

[Приложение Г Листинг программного кода приложения «Калькулятор» (appsettings.Development.json) 14](#_Toc152161903)

[Приложение Д Листинг программного кода приложения «Калькулятор» (Program.cs) 15](#_Toc152161904)

# **Практическая работа «Использование конвейера данных реального времени с Kafka»**

1.1 Цель работы

Усовершенствовать структуру тестового приложения «Калькулятор», которое было разработано ранее, с использованием системы Apache Kafka. В частности, сформировать обмен сообщениями между веб-страницей ввода исходных данных и веб-страницей получения результатов, записи выполненной операции в базу данных MariaDB через брокер сообщений Kafka.

1.2 Ход проведения работы

### 1.2.1 Разработка тестового приложения Visual Studio .NET Core

Усовершенствуем приложение «11\_Calculator», которое было разработано в предыдущих лабораторных работах. В частности, создаем папку Services с файлами KafkaConsumerService.cs, KafkaProducerHandler.cs и KafkaProducerService.cs. Создаем файлы EnumExtensions.cs и MathOperation.cs , также частично модернизируем ранее созданные файлы. В проект дополнительно подключаем пакет расширения Confluent.Kafka 2.3.0.

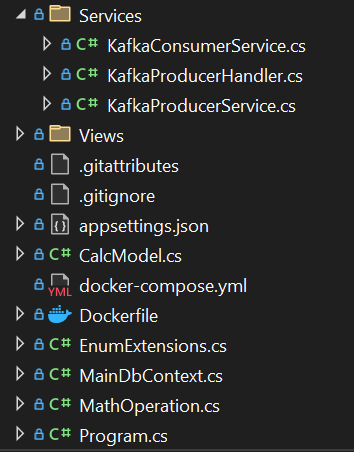


Рисунок 1 – Фрагмент обозревателя решений с новыми файлами проекта

Выполнена сборка и протестирована работа приложения, чтобы убедиться - оно выполняет требуемые функции.

Фрагменты Visual Studio c отображением листинга программного кода файла appsettings.json и appsettings.Development.json представлены на рисунках 2,3 соответственно. Листинг кода некоторых файлов представлен в приложениях А-Д.

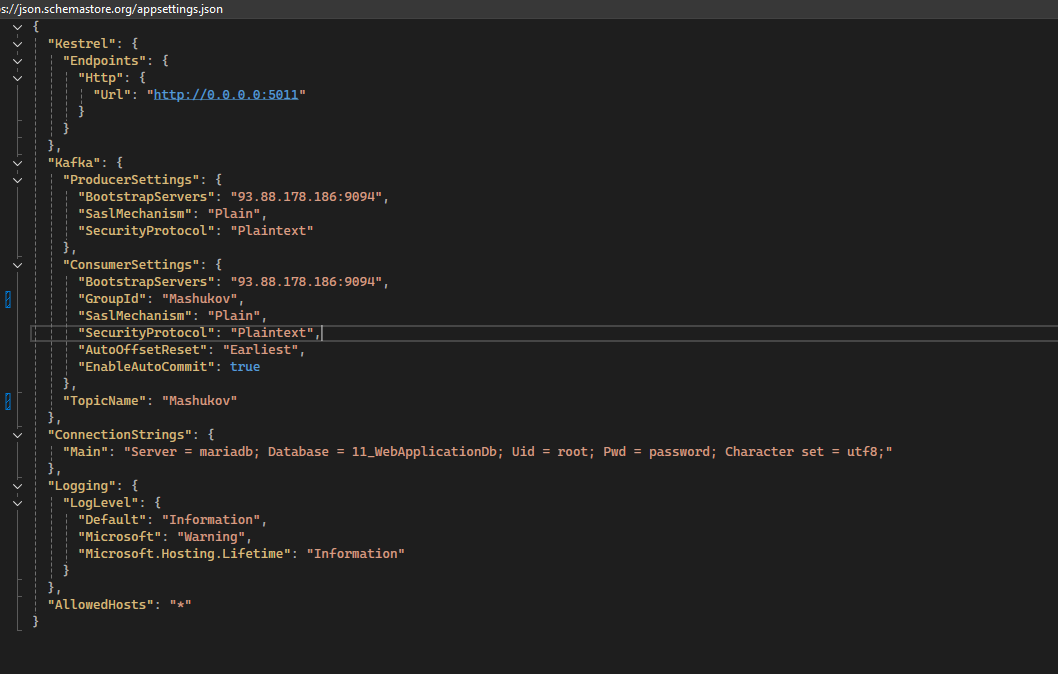


Рисунок 2 - Фрагмент Visual Studio c отображением листинга программного кода файла appsettings.json

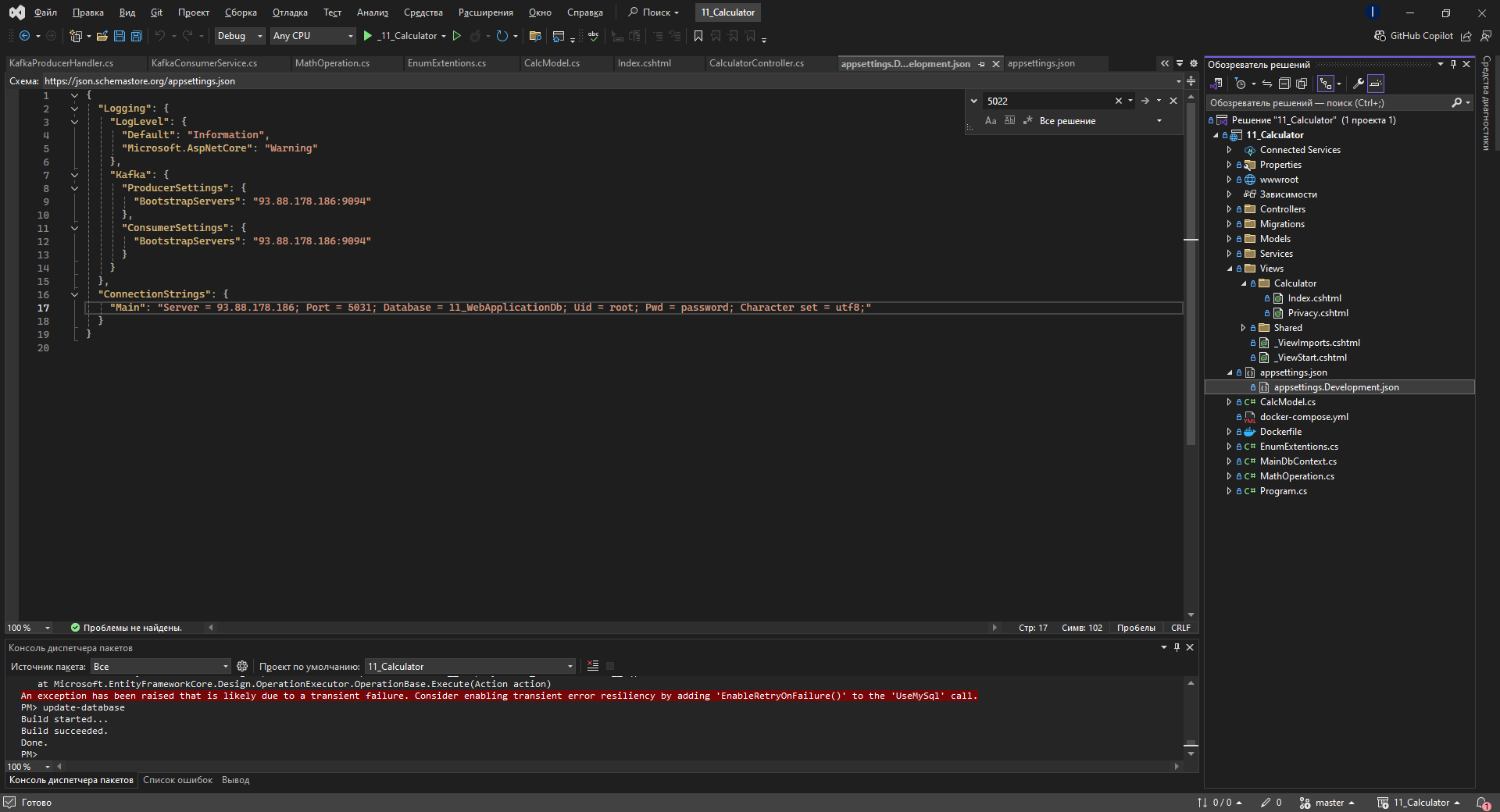


Рисунок 3 - Фрагмент Visual Studio c отображением листинга программного кода файла appsettings.Development.json

### 1.2.2 Размещение приложения в системе контроля версий GitHub

После того, как тестовое приложение работает без ошибок, загружаем его в личный репозиторий на GitHub.

### 1.2.3 Сборка образов

В терминальном режиме на удаленном сервере клонируем проект из личного репозитория GitHub. Для этого выполняем команду: git clone <HTTPS>

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

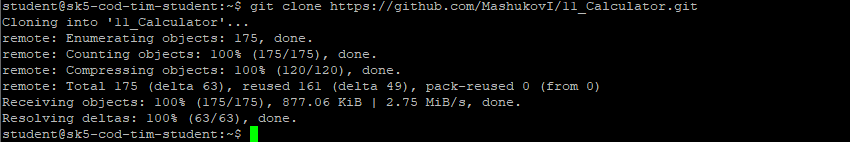


Рисунок 4 – Окно терминальной консоли с результатами клонирования тестового проекта

Далее запускаем многоконтейнерное приложение. В командной строке в папке с файлом docker-compose.yml, выполняем команду: docker compose up -d

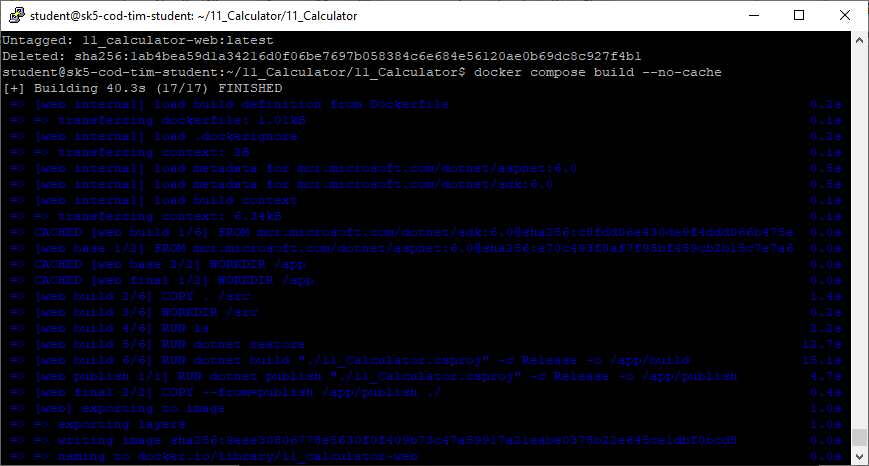


Рисунок 5 – Запуск многоконтейнерного приложения

Проверяем существование созданных образов в списке активных контейнеров и их идентификаторов с помощью команды: docker ps -a

Рисунок 6 - Окно терминальной консоли с демонстрацией созданного образа



### 1.2.4 Установка программы HeidiSQL для сопровождения базы данных в СУБД MariaDB

Этот шаг осуществлен в предыдущей работе. Окно программы HeidiSQL с отображением наполненной базы в MariaDB представлено на рисунке 8.

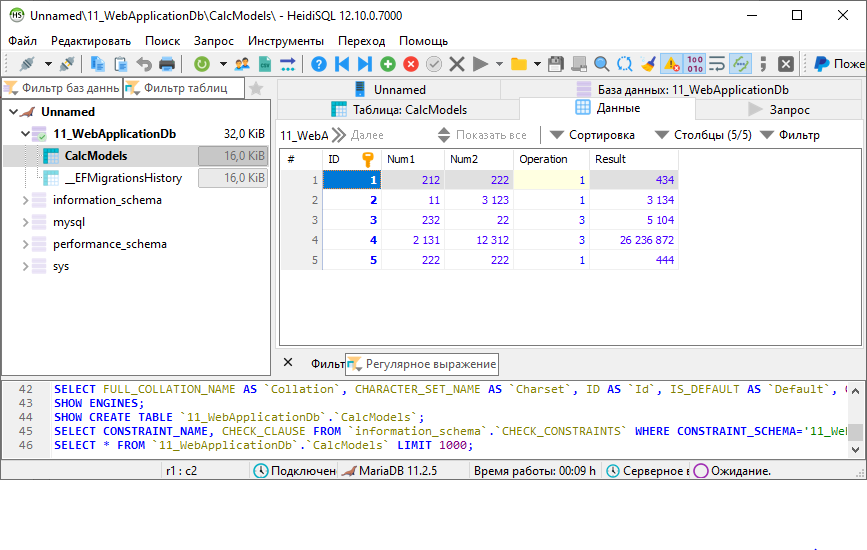


Рисунок 7 - Окно программы HeidiSQL с отображением наполненной базы в MariaDB

### 1.2.5 Настройка миграции базы данных в MariaDB

Данный шаг тоже был реализован в предыдущей лабораторной работе.

### 1.2.6 Демонстрация работы приложения брокером сообщений Kafka

Запускаем приложение «11\_Calculator» в браузере по ссылке http://93.88.178.186:5011/. Выполняем несколько тестовых расчетов и убеждаемся, что данные попадают в базу данных 11-WebCalculatorDb.

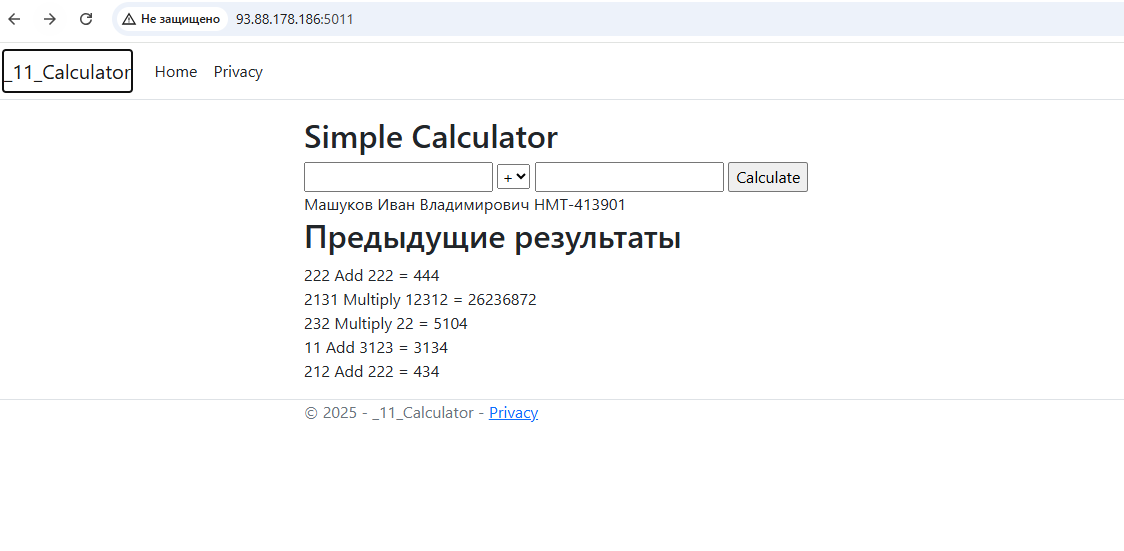


Рисунок 8 - Окно веб-браузера с демонстрацией работающего тестового приложения на выбранном порту

Проверяем запись в Kafka в созданном топике по ссылке http://93.88.178.186:10025/.

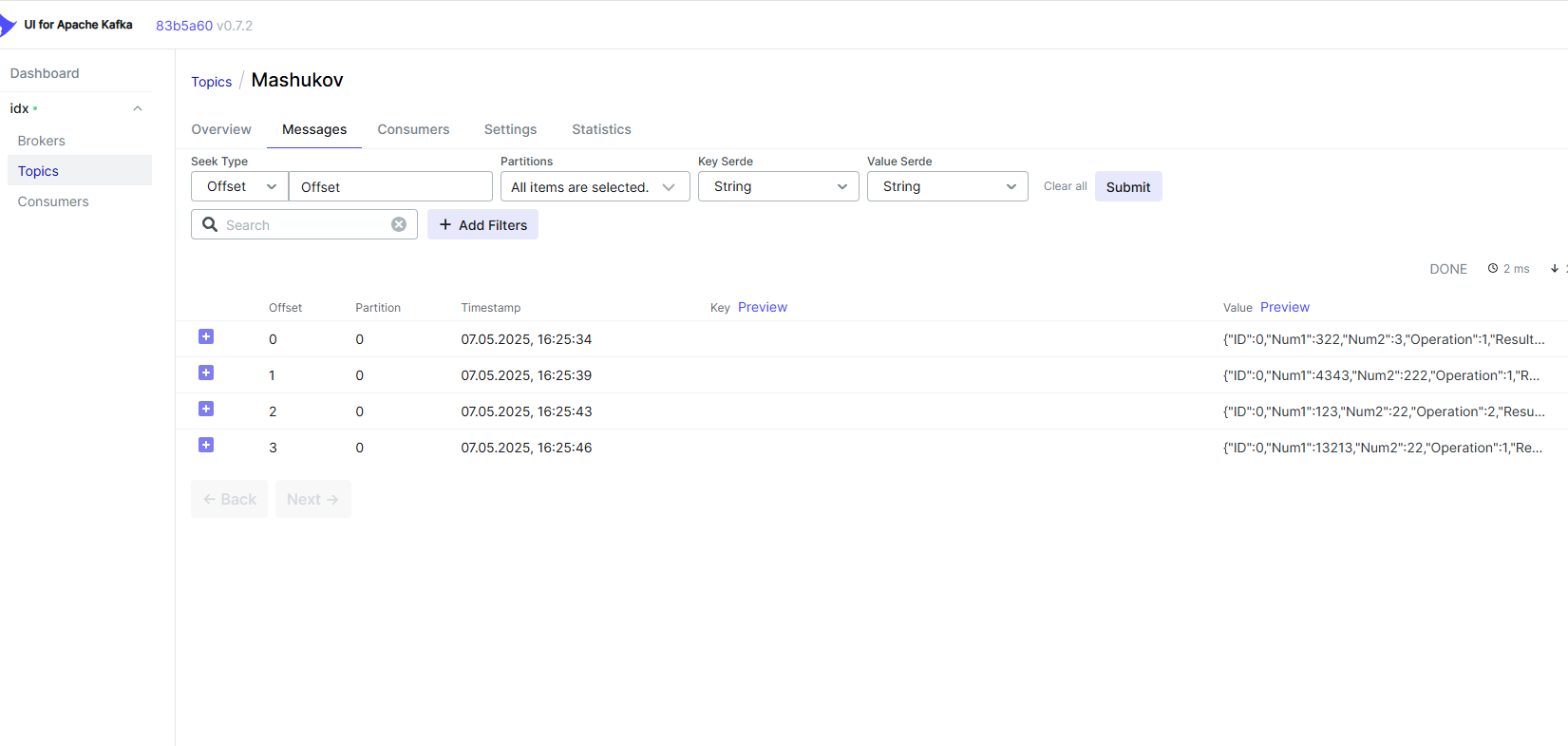


Рисунок 10 - Окно веб-браузера с демонстрацией записей в Kafka в созданном топике

1.3 Выводы

В ходе выполнения работы осуществлено усовершенствование структуры тестового приложения «Калькулятор», которое было разработано ранее, с использованием системы Apache Kafka. В частности, сформирован обмен сообщениями между веб-страницей ввода исходных данных и веб-страницей получения результатов, записи выполненной операции в базу данных MariaDB через брокер сообщений Kafka.

Приложение А Листинг программного кода приложения «Калькулятор» (контроллер)

using \_11\_Calculator.Kafka;

using Confluent.Kafka;

using Microsoft.AspNetCore.Mvc;

using System.Text.Json;

namespace \_11\_Calculator.Controllers

{

public class CalculatorController : Controller

{

private readonly MainDbContext \_context;

private readonly KafkaProducerService<Null, string> \_producer;

public CalculatorController(MainDbContext context, KafkaProducerService<Null, string> producer)

{

\_context = context;

\_producer = producer;

}

/// <summary>

/// Отображение страницы Index.

/// </summary>

public IActionResult Index()

{

var data = \_context.CalcModels.OrderByDescending(x => x.ID).ToList();

return View(data);

}

/// <summary>

/// Обработка запроса на вычисление.

/// </summary>

/// <param name="num1">Первый операнд.</param>

/// <param name="num2">Второй операнд.</param>

/// <param name="operation">Тип операции (сложение, вычитание, умножение, деление).</param>

[HttpPost]

[ValidateAntiForgeryToken]

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

{

// Подготовка объекта для расчета

var dataInputVariant = new CalcModel

{

Num1 = num1,

Num2 = num2,

Operation = operation,

};

// Отправка данных в Kafka

await SendDataToKafka(dataInputVariant);

// Перенаправление на страницу Index

return RedirectToAction(nameof(Index));

}

public IActionResult Callback([FromBody] CalcModel inputData)

{

// Сохранение данных и результата в базе данных

SaveDataAndResult(inputData);

return Ok();

}

/// <summary>

/// Сохранение данных и результата в базе данных.

/// </summary>

/// <param name="num1">Первый операнд.</param>

/// <param name="num2">Второй операнд.</param>

/// <param name="operation">Тип операции (сложение, вычитание, умножение, деление).</param>

/// <param name="result">Результат математической операции.</param>

/// <returns>Объект с данными и результатом.</returns>

private void SaveDataAndResult(CalcModel inputData)

{

\_context.CalcModels.Add(inputData);

\_context.SaveChanges();

}

/// <summary>

/// Отправка данных в Kafka.

/// </summary>

/// <param name="dataInputVariant">Объект с данными и результатом.</param>

/// <returns>Task.</returns>

private async Task SendDataToKafka(CalcModel dataInputVariant)

{

var json = JsonSerializer.Serialize(dataInputVariant);

await \_producer.ProduceAsync("Mashukov", new Message<Null, string> { Value = json });

}

}

}

Приложение Б Листинг программного кода приложения «Калькулятор» (KafkaConsumerService.cs)

using \_11\_Calculator;

using Confluent.Kafka;

using System.Data;

using System.Text.Json;

namespace \_11\_Calculator.Kafka

{

public class KafkaConsumerService : BackgroundService

{

private readonly string \_httpCallbackPath = "http://localhost:5002/Calculator/callback";

private readonly string \_topic;

private readonly IConsumer<Null, string> \_kafkaConsumer;

private readonly IServiceProvider \_serviceProvider;

public KafkaConsumerService(IConfiguration configuration, IServiceProvider serviceProvider)

{

\_serviceProvider = serviceProvider;

var consumerConfig = new ConsumerConfig();

configuration.GetSection("Kafka:ConsumerSettings").Bind(consumerConfig);

\_topic = configuration.GetValue<string>("Kafka:TopicName");

\_kafkaConsumer = new ConsumerBuilder<Null, string>(consumerConfig).Build();

}

protected override Task ExecuteAsync(CancellationToken stoppingToken)

{

return Task.Run(async () =>

{

await StartConsumerLoop(stoppingToken);

}, stoppingToken);

}

private async Task StartConsumerLoop(CancellationToken stoppingToken)

{

\_kafkaConsumer.Subscribe(\_topic);

while (!stoppingToken.IsCancellationRequested)

{

try

{

var cr = \_kafkaConsumer.Consume(stoppingToken);

var ip = cr.Message.Value;

var inputData = JsonSerializer.Deserialize<CalcModel>(ip);

inputData.CalculateOperation();

var httpClient = new HttpClient();

\_ = await httpClient.PostAsJsonAsync(\_httpCallbackPath, inputData);

Console.WriteLine($"Message key: {cr.Message.Key}, value: {cr.Message.Value}");

}

catch (OperationCanceledException)

{

break;

}

catch(ConsumeException e)

{

if (e.Error.IsFatal)

break;

}

catch (Exception)

{

break;

}

}

}

public override void Dispose()

{

\_kafkaConsumer.Close();

\_kafkaConsumer.Dispose();

base.Dispose();

}

}

}

Приложение В Листинг программного кода приложения «Калькулятор» (appsettings.json)

{

"Kestrel": {

"Endpoints": {

"Http": {

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

}

}

},

"Kafka": {

"ProducerSettings": {

"BootstrapServers": "93.88.178.186:9094",

"SaslMechanism": "Plain",

"SecurityProtocol": "Plaintext"

},

"ConsumerSettings": {

"BootstrapServers": "93.88.178.186:9094",

"GroupId": "Mashukov",

"SaslMechanism": "Plain",

"SecurityProtocol": "Plaintext",

"AutoOffsetReset": "Earliest",

"EnableAutoCommit": true

},

"TopicName": "Mashukov"

},

"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 Г Листинг программного кода приложения «Калькулятор» (appsettings.Development.json)

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"Kafka": {

"ProducerSettings": {

"BootstrapServers": "93.88.178.186:9094"

},

"ConsumerSettings": {

"BootstrapServers": "93.88.178.186:9094"

}

},

"ConnectionStrings": {

"Main": "Server = 93.88.178.186; Port = 5031; Database = 11\_WebApplicationDb; Uid = root; Pwd = password; Character set = utf8;"

}

}

Приложение Д Листинг программного кода приложения «Калькулятор» (Program.cs)

using \_11\_Calculator.Kafka;

using Confluent.Kafka;

using Microsoft.EntityFrameworkCore;

namespace \_11\_Calculator

{

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

var connStr = builder.Configuration.GetConnectionString("Main");

builder.Services.AddDbContext<MainDbContext>(opt => opt.UseMySql(connStr, new MySqlServerVersion(new Version(11,2))));

// Add services to the container.

builder.Services.AddControllersWithViews();

builder.Services.AddHostedService<KafkaConsumerService>();

builder.Services.AddSingleton<KafkaProducerHandler>();

builder.Services.AddSingleton<KafkaProducerService<Null, string>>();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

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

}

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(

name: "default",

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

app.Run();

}

}

}