

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»

Автоматизация сборки проекта с помощью Jenkins

ОТЧЕТ

по практической работе № 4

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

Я.В.Крашенинников

Преподаватель:
профессор, д.т.н.

В.В.Лавров

Екатеринбург
2024

СОДЕРЖАНИЕ

Практическая работа «Автоматизация сборки проекта с помощью Jenkins»	3
1.1 Цель работы.....	3
1.2 Ход проведения работы.....	3
1.2.1 Разработка тестового приложения Visual Studio .NET Core	3
1.2.2 Создание контейнера с использованием Docker	3
1.2.3 Создание файла docker-compose.yml	4
1.2.4 Создание файла Jenkinsfile	4
1.2.5 Размещение приложения в системе контроля версий GitHub.....	5
1.2.6 Настройка автоматической сборки проекта в Jenkins и интеграции с GitHub	5
1.2.7 Выполнение автоматической сборки проекта в Jenkins и анализ результатов	6
1.2.8 Демонстрация автоматической сборки и доставки проекта при корректировке функциональности веб-приложения.....	7
1.3 Выводы.....	8
Приложение А Листинг программного кода приложения «Калькулятор» (контроллер).....	9
Приложение Б Листинг программного кода приложения «Калькулятор» (appsettings.json)	11
Приложение В Листинг программного кода приложения «Калькулятор» (appsettings.Development.json).....	12
Приложение Г Листинг программного кода приложения «Калькулятор» (Program.cs).....	13
Приложение Д Листинг файла Dockerfile.....	15
Приложение Е Листинг файла docker-compose.yml	16
Приложение Ж Листинг файла Jenkinsfile	17

Практическая работа

«Автоматизация сборки проекта с помощью Jenkins»

1.1 Цель работы

Познакомиться с инструментом автоматизации сборки Jenkins и научить их создавать и настраивать простой процесс автоматизированной сборки проекта.

1.2 Ход проведения работы

1.2.1 Разработка тестового приложения Visual Studio .NET Core

На этом этапе я откатился до версии, где у меня не было ни БД, ни кафки

1.2.2 Создание контейнера с использованием Docker

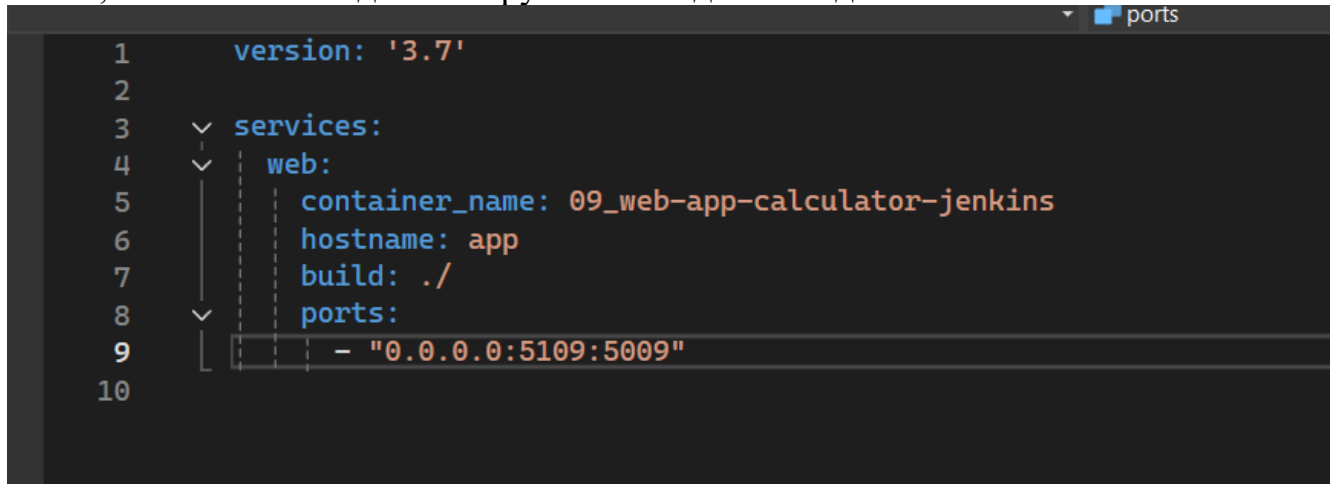
Dockerfile уже содержит все необходимые инструкции для корректной сборки контейнера, поэтому никаких изменений в его конфигурацию вносить не требуется. Текущая настройка полностью соответствует требованиям и позволяет создать контейнер с нужными параметрами без дополнительных модификаций.

```
1  # Используем базовый образ с ASP.NET 6.0
2  FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
3
4  # Устанавливаем рабочую директорию внутри контейнера
5  WORKDIR /app
6
7  # Используем SDK
8  FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
9
10 # Копируем файлы в контейнер
11 COPY . /src
12
13 # Устанавливаем рабочую директорию внутри контейнера
14 WORKDIR /src
15
16 # Устанавливаем зависимости приложения и параметры компиляции
17 RUN ls
18 RUN dotnet restore
19 RUN dotnet build ".\09_Calculate.csproj" -c Release -o /app/build
20
21 FROM build AS publish
22 RUN dotnet publish ".\09_Calculate.csproj" -c Release -o /app/publish
23
24 FROM base AS final
25 WORKDIR /app
26 COPY --from=publish /app/publish ./
27
28 # Определяем команду запуска контейнера
```

Рисунок 1 - Dockerfile

1.2.3 Создание файла docker-compose.yml

Docker-compose.yml - это конфигурационный файл, который позволяет единожды описать все необходимые параметры контейнеров (порты, зависимости и другие настройки) и в дальнейшем использовать их для автоматического развертывания, избегая необходимости ручного ввода команд

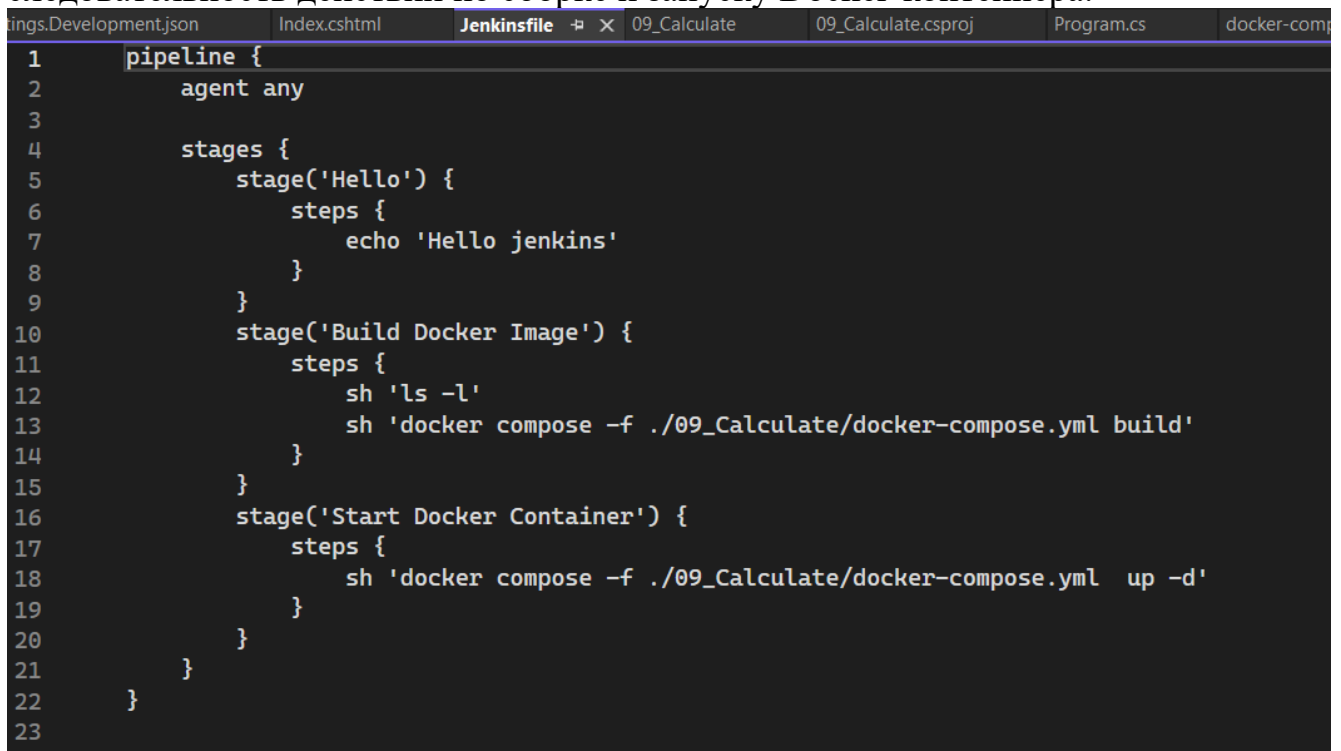


```
1 version: '3.7'
2
3 services:
4   web:
5     container_name: 09_web-app-calculator-jenkins
6     hostname: app
7     build: ./
8     ports:
9     - "0.0.0.0:5109:5009"
10
```

Рисунок 2 – Docker-compose

1.2.4 Создание файла Jenkinsfile

Jenkinsfile содержит набор инструкций для сервера Jenkins, определяющих последовательность действий по сборке и запуску Docker-контейнера.



```
1 pipeline {
2   agent any
3
4   stages {
5     stage('Hello') {
6       steps {
7         echo 'Hello jenkins'
8       }
9     }
10    stage('Build Docker Image') {
11      steps {
12        sh 'ls -l'
13        sh 'docker compose -f ./09_Calculate/docker-compose.yml build'
14      }
15    }
16    stage('Start Docker Container') {
17      steps {
18        sh 'docker compose -f ./09_Calculate/docker-compose.yml up -d'
19      }
20    }
21  }
22 }
23
```

Рисунок 3 - JenkinsFile

1.2.5 Размещение приложения в системе контроля версий GitHub

Во время выполнения работы каждый новый шаг отмечался сохранением в системе контроля версий GitHub

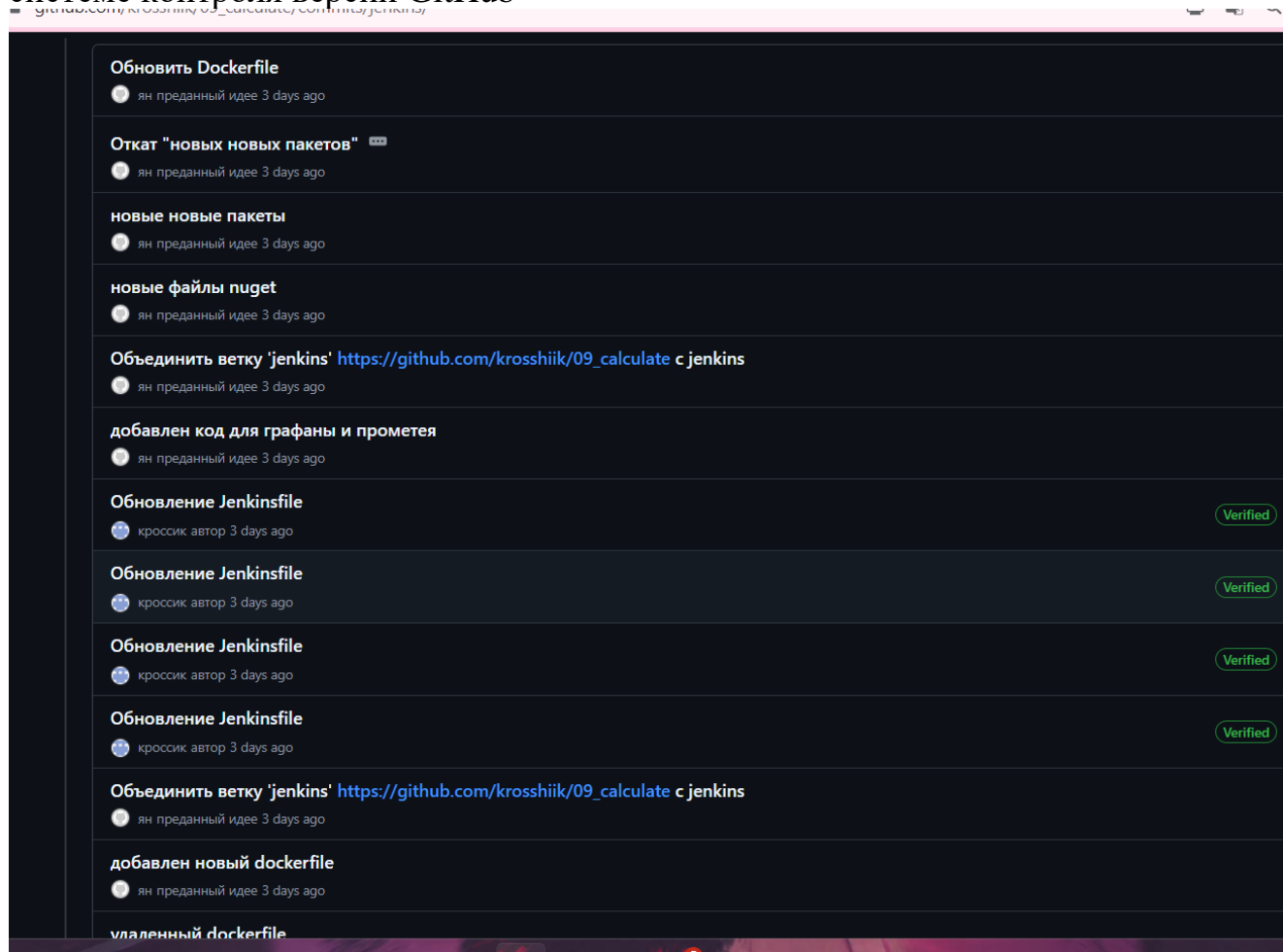


Рисунок 4 - Коммиты

1.2.6 Настройка автоматической сборки проекта в Jenkins и интеграции с GitHub

Завершив подготовительный этап проекта для Jenkins, я приступил к ключевому этапу - конфигурации инструмента для его эффективного функционирования. Используя порт 8080, я осуществил вход в веб-интерфейс Jenkins через серверный адрес. Далее я сформировал новый элемент, взяв за основу имеющийся шаблон, и адаптировал настройки в соответствии с требованиями моего проекта

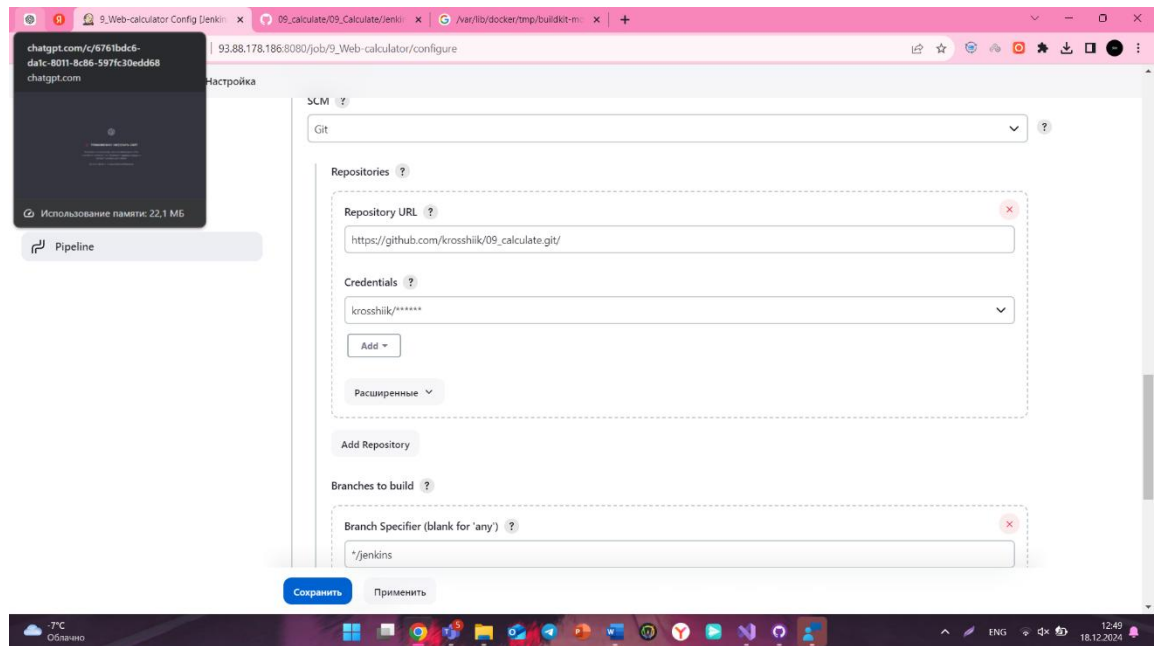


Рисунок 5 - настройки Jenkins

1.2.7 Выполнение автоматической сборки проекта в Jenkins и анализ результатов

При настройке команд для работы с Docker, я добавил флаг `-f` с указанием пути до конфигурационного файла. Это обеспечивает точное указание расположения файла `docker-compose.yml` и гарантирует корректное выполнение команд независимо от текущей директории.

```

1 pipeline {
2   agent any
3
4   stages {
5     stage('Hello') {
6       steps {
7         echo 'Hello jenkins'
8       }
9     }
10    stage('Build Docker Image') {
11      steps {
12        sh 'ls -l'
13        sh 'docker compose -f ./09_Calculate/docker-compose.yml build'
14      }
15    }
16    stage('Start Docker Container') {
17      steps {
18        sh 'docker compose -f ./09_Calculate/docker-compose.yml up -d'
19      }
20    }
21  }
22 }
23

```

Рисунок 6 - изменения в Jenkinsfile

1.2.8 Демонстрация автоматической сборки и доставки проекта при коррек- тировке функциональности веб-приложения

После внесения необходимых изменений в конфигурационный файл, Jenkins продемонстрировал стабильную работу (как показано на Рисунке 9). Система теперь успешно выполняет все стадии автоматической сборки, результатом чего является корректный запуск приложения на сервере.

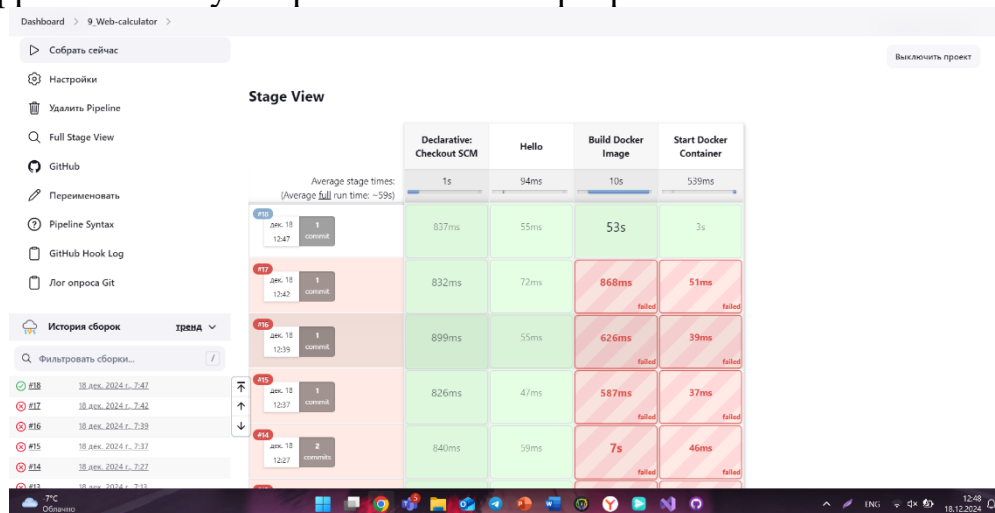


Рисунок 7 - история сборок в Jenkins

История сборок отражает внесенные изменения в код, включая новые коммиты. После добавления заголовка страницы, Jenkins автоматически выполнил повторную успешную сборку. В итоге была создана рабочая конфигурация, которая обеспечивает развертывание текущей версии приложения, синхронизированной с веткой Jenkins в репозитории GitHub.

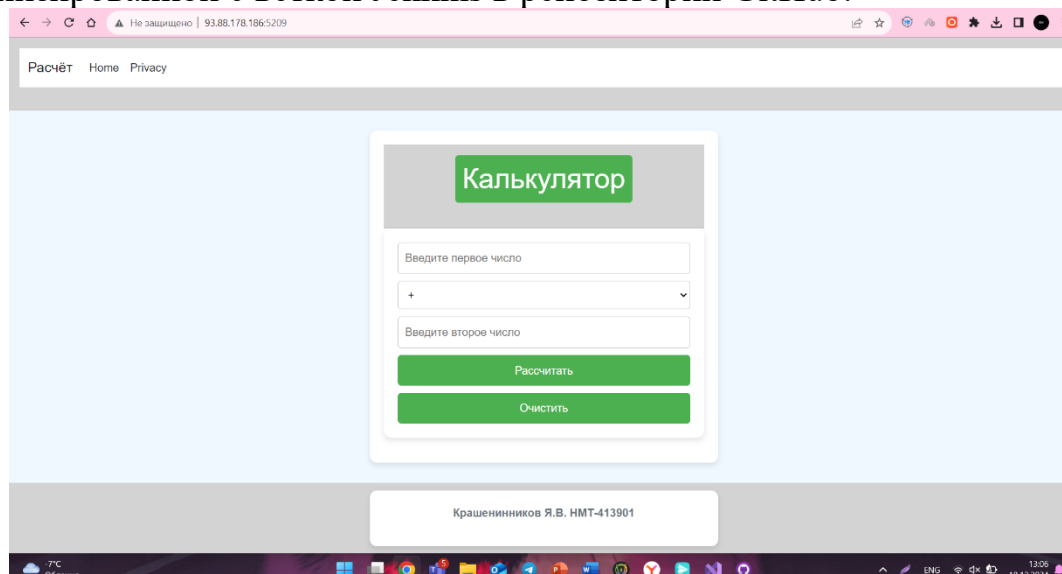


Рисунок 8 - интерфейс приложения

1.3 Выводы

В процессе исследования был освоен Jenkins - система непрерывной интеграции, которая показала высокую результативность в организации CI/CD процессов. В результате был создан автоматизированный пайплайн, охватывающий весь цикл от синхронизации с GitHub до запуска приложения в Docker-контейнере. Такой подход обеспечил надежное автоматическое обновление программы при модификации кода.

Ключевым элементом настройки стал Jenkinsfile, определяющий структуру и последовательность этапов сборки. Решение технических проблем, включая вопросы иерархии файлов, подчеркнуло важность адаптивного подхода к конфигурации и детального понимания архитектуры проекта. Оптимизация процесса сборки помогла предотвратить потенциальные сложности.

Практическое применение Jenkins продемонстрировало ключевую роль автоматизации в современной разработке. Этот инструмент существенно ускоряет процессы, минимизирует риск человеческих ошибок и упрощает развертывание. Интеграция с Docker подчеркивает значимость контейнеризации для обеспечения совместимости приложений в различных средах.

Проведенная работа подтверждает, что автоматизация сборки является неотъемлемой частью современных методологий разработки. Созданная конфигурация может масштабироваться для более комплексных проектов, что делает её ценным инструментом в профессиональной деятельности.

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

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace _09_Calculate.Controllers
{
    public enum Operation { Add, Subtract, Multiply, Divide }

    public class CalculatorController : Controller
    {
        [HttpGet]
        public IActionResult Index()
        {
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public 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;
            }
            ViewBag.Result = result;
            return View("Index");
        }
    }
}
```

}
}

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

```
{  
  "Kestrel": {  
    "Endpoints": {  
      "Http": {  
        "Url": "http://0.0.0.0:5009" //  
      }  
    }  
  },  
  "Logging": {  
    "LogLevel": {  
      "Default": "Information",  
      "Microsoft.AspNetCore": "Warning"  
    }  
  },  
  "AllowedHosts": "*"   
}
```

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

```
{  
  "DetailedErrors": true,  
  "Logging": {  
    "LogLevel": {  
      "Default": "Information",  
      "Microsoft": "Warning",  
      "Microsoft.Hosting.Lifetime": "Information"  
    }  
  }  
}
```

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

```
using Microsoft.EntityFrameworkCore;
using OpenTelemetry.Metrics;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

builder.Services.AddOpenTelemetry()
    .WithMetrics(meterProviderBuilder =>
    {
        meterProviderBuilder.AddPrometheusExporter();

        meterProviderBuilder.AddMeter("Microsoft.AspNetCore.Hosting",
            "Microsoft.AspNetCore.Server.Kestrel");

        // Status code
        meterProviderBuilder.AddMeter("Microsoft.AspNetCore.Http.Connections");

        meterProviderBuilder.AddView("http.server.request.duration",
            new ExplicitBucketHistogramConfiguration
            {
                Boundaries =
                [
                    0,
                    0.005,
                    0.01,
                    0.025,
                    0.05,
                    0.075,
                    0.1,
                    0.25,
                    0.5,
                    0.75,
                    1,
                    2.5,
                    5,
                    7.5,
                    10
                ]
            })
    });
```

```
    ]  
    });  
});
```

```
builder.Services.AddRazorPages();  
  
var app = builder.Build();  
  
// Configure the HTTP request pipeline.  
if (!app.Environment.IsDevelopment())  
{  
    app.UseExceptionHandler("/Home/Error");  
    // The default HSTS value is 30 days. You may want to change this for produc-  
tion scenarios, see https://aka.ms/aspnetcore-hsts.  
    app.UseHsts();  
}  
  
app.UseHttpsRedirection();  
app.UseStaticFiles();  
  
app.MapPrometheusScrapingEndpoint();  
  
app.UseRouting();  
  
app.UseAuthorization();  
  
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller=Calculator}/{action=Index}/{id?}");  
  
app.Run();
```

Приложение Д Листинг файла Dockerfile

```
# Используем базовый образ с ASP.NET 6.0
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base

# Устанавливаем рабочую директорию внутри контейнера
WORKDIR /app

# Используем SDK
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build

# Копируем файлы в контейнер
COPY . /src

# Устанавливаем рабочую директорию внутри контейнера
WORKDIR /src

# Устанавливаем зависимости приложения и параметры компиляции
RUN ls
RUN dotnet restore
RUN dotnet build ".\09_Calculate.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish ".\09_Calculate.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish ./

# Определяем команду запуска контейнера
ENTRYPOINT ["dotnet", "09_Calculate.dll"]
```

Приложение Е Листинг файла docker-compose.yml

version: '3.7'

services:

web:

container_name: 09_web-app-calculator-jenkins

hostname: app

build: ./

ports:

- "0.0.0.0:5109:5009"

Приложение Ж Листинг файла Jenkinsfile

```
pipeline {
  agent any

  stages {
    stage('Hello') {
      steps {
        echo 'Hello jenkins'
      }
    }
    stage('Build Docker Image') {
      steps {
        sh 'ls -l'
        sh 'docker compose -f ./09_Calculate/docker-compose.yml build'
      }
    }
    stage('Start Docker Container') {
      steps {
        sh 'docker compose -f ./09_Calculate/docker-compose.yml up -d'
      }
    }
  }
}
```