

## MassTransit

### MassTransit

#### Ответ

MassTransit — это платформа для реализации шаблона обмена сообщениями (message-driven architecture), написанная на .NET. Она позволяет создавать распределённые системы путём отправки сообщений между компонентами приложения. MassTransit поддерживает различные брокеры сообщений, такие как RabbitMQ, Azure Service Bus и Kafka, предоставляя простой API для публикации сообщений, обработки запросов и реагирования на события.

Основные возможности MassTransit включают:

#### 1. Шаблоны взаимодействия

- **Publish/Subscribe:** отправка сообщений многим подписчикам.
- **Request/Response:** синхронная обработка запросов с ожиданием ответа.
- **Command/Event:** разделение команд и событий для повышения гибкости архитектуры.

#### 2. Поддерживаемые транспортные механизмы

Поддерживает работу с несколькими транспортными механизмами:

- RabbitMQ
- Azure Service Bus
- ActiveMQ
- Amazon SQS

#### 3. Обработка ошибок и восстановление

Предоставляет встроенную поддержку механизмов восстановления после сбоев, повторных попыток доставки сообщений и использование отложенных очередей.

#### 4. Автоматическое управление конфигурациями

Автоматически создаёт очереди и обменники, упрощая настройку инфраструктуры.

#### Простота интеграции

MassTransit легко интегрируется с популярными фреймворками и контейнерами зависимостей, такими как ASP.NET Core и Autofac.

Пример простейшего использования MassTransit:// Подключение библиотеки  
var busControl = Bus.Factory.CreateUsingRabbitMq(cfg =>{  
cfg.Host("rabbitmq://localhost");});busControl.Start();// Публикация сообщения  
await bus.Publish(new MyMessage { Text = "Hello World!" });

Таким образом, MassTransit является мощным инструментом для разработки масштабируемых микросервисных приложений на платформе .NET.

## Как настроить MassTransit с RabbitMQ?

### Ответ

Настройка MassTransit с использованием RabbitMQ включает несколько шагов. Вот подробное руководство по быстрой настройке и запуску проекта:

### Шаг 1: Установка пакетов NuGet

Для начала установите необходимые пакеты NuGet в ваш проект. Откройте консоль диспетчера пакетов (Package Manager Console) или терминал и выполните следующие команды:

Install-Package MassTransit

Install-Package MassTransit.RabbitMQ

Или добавьте их вручную в файл .csproj вашего проекта:

```
<ItemGroup>
  <PackageReference Include="MassTransit" Version="8.*" />
  <PackageReference Include="MassTransit.RabbitMQ" Version="8.*" />
</ItemGroup>
```

### Шаг 2: Настройка подключения к RabbitMQ

Создайте класс конфигурации BusConfigurer, который будет настраивать соединение с RabbitMQ:

```
public class BusConfigurer
{
    public static IBusControl CreateBus()
    {
        return Bus.Factory.CreateUsingRabbitMq(cfg =>
        {
            var host = cfg.Host(new Uri("rabbitmq://localhost"), hst => {});

            // Регистрация обработчиков и потребителей
            cfg.ReceiveEndpoint(host, "my_queue_name", ep =>
            {
                ep.Consumer<MyConsumer>();
            });
        });
    }
}
```

Здесь мы создаем очередь "my\_queue\_name" и привязываем её к потребителю MyConsumer.

### Шаг 3: Реализация потребителя

Создайте класс-потребитель (consumer), который будет обрабатывать поступающие сообщения:

```
using System.Threading.Tasks;
```

```
using GreenPipes;
using MassTransit;

public class MyConsumer : IConsumer<MyMessage>
{
    public Task Consume(ConsumeContext<MyMessage> context)
    {
        var message = context.Message.Text;
        Console.WriteLine($"Received message: {message}");
        return Task.CompletedTask;
    }
}
```

Этот потребитель принимает сообщение типа **MyMessage** и выводит его содержимое в консоль.

#### Шаг 4: Создание класса модели сообщения

Определите структуру сообщения, которое будете отправлять:

```
public class MyMessage
{
    public string Text { get; set; }
}
```

#### Шаг 5: Запуск шину и публикация сообщения

Теперь создадим точку входа в приложение и запустим процесс:

```
class Program
{
    static async Task Main(string[] args)
    {
        var busControl = BusConfigurer.CreateBus();

        await busControl.StartAsync(); // Запускаем шину

        try
        {
            while (true)
            {
                Console.Write("Enter a message to send or 'exit' to quit: ");
                var input = Console.ReadLine();

                if (input == "exit")
                    break;

                await busControl.Publish(new MyMessage { Text = input });
            }
        }
        finally
    }
}
```

```
{  
    await busControl.StopAsync(); // Остановка шины  
}  
}  
}
```

## Итоги настройки

Мы настроили систему таким образом, что теперь при вводе текста в консoli ваше приложение публикует сообщения в RabbitMQ, которые обрабатываются вашим потребителем и выводятся обратно в консоль.

Это базовая настройка MassTransit с RabbitMQ. Для более сложных сценариев можно добавить дополнительные опции вроде конфигурирования маршрутов, журналирования и расширенной обработки ошибок.

Эти операции мы проделываем в одном едином проекте ?

## Ответ

Да, все указанные шаги выполняются внутри одного проекта. Однако часто MassTransit используется именно в архитектурных решениях с микросервисами, когда разные части функциональности реализуются отдельно друг от друга. Например, один сервис (проект) занимается публикацией сообщений (например, отправкой заказов), а другой сервис (другой проект) — их обработкой (например, сохранением заказа в базу данных). Поэтому приведенный пример демонстрирует минимальную реализацию всего в рамках одного проекта для понимания базовых принципов работы.

Однако, когда вы начнёте строить полноценную архитектуру с множеством сервисов, каждый из них станет отдельным проектом, выполняющим свою специфичную роль.

Например:

- Один проект («Сервис отправки») отправляет заказы клиентам.
- Другой проект («Логистика») получает заказанные товары и доставляет их клиенту.
- Третий проект («Учёт товаров») следит за наличием товара на складе.

Каждый из этих проектов может иметь свою собственную реализацию MassTransit, соединённую через общую очередь RabbitMQ.

Но для первоначального знакомства и быстрого старта вполне достаточно реализовать всё в пределах одного проекта, как показано ранее.

```
var host = cfg.Host(new Uri("rabbitmq://localhost"), hst => { });Severity Code  
Description Project File Line Suppression State DetailsError  
(active) CS0815 Cannot assign void to an implicitly-typed variable MassTransit01  
D:\VC2508\MassTransit\MassTransitSIn\MassTransit01\BusConfigurer.cs 16
```

## Ответ

Ошибка возникает потому, что метод `cfg.Host()` возвращает значение типа `void`. Это значит, что возвращаемое значение нельзя присвоить переменной. Исправьте код следующим образом:

Удалите ненужную переменную `host` и вызовите метод непосредственно:

```

public class BusConfigurer
{
    public static IBusControl CreateBus()
    {
        return Bus.Factory.CreateUsingRabbitMq(cfg =>
        {
            cfg.Host(new Uri("rabbitmq://localhost"), hst => {}); // Просто вызов метода без присваивания результата

            // Регистрация обработчиков и потребителей
            cfg.ReceiveEndpoint("my_queue_name", ep =>
            {
                ep.Consumer<MyConsumer>();
            });
        });
    }
}

```

Также обратите внимание, что аргумент метода `ReceiveEndpoint` изменился — теперь это строка с именем очереди, а не объект `IRabbitMqHost`. Так должно выглядеть правильно:

```

cfg.ReceiveEndpoint("my_queue_name", ep =>
{
    ep.Consumer<MyConsumer>();
});

```

**Итоговый исправленный вариант:**

```

public class BusConfigurer
{
    public static IBusControl CreateBus()
    {
        return Bus.Factory.CreateUsingRabbitMq(cfg =>
        {
            cfg.Host(new Uri("rabbitmq://localhost"), hst => {}); // Метод Host() не возвращает значения, удаляем присвоение

            cfg.ReceiveEndpoint("my_queue_name", ep => // Стока с названием очереди
            {
                ep.Consumer<MyConsumer>();
            });
        });
    }
}

```

**Попробуйте снова собрать проект — ошибка должна исчезнуть!**

```

using MassTransit01;using System;using System.Collections.Generic;using
System.Linq;using System.Text;using System.Threading.Tasks;
namespace MassTransit01{ public class BusConfigurer { 12 public static
IBusControl CreateBus() { 14     return Bus.Factory.CreateUsingRabbitMq(cfg =>
{
    cfg.Host(new Uri("rabbitmq://localhost"), hst => {});
    // Регистрация обработчиков и потребителей

```

```
cfg.ReceiveEndpoint("my_queue_name", ep =>
{
    ep.Consumer<MyConsumer>();
});
});
```

Severity	Code	Description	Project File	Line	Suppression State
DetailsError (active)	CS0246	The type or namespace name 'IBusControl' could not be found (are you missing a using directive or an assembly reference?)	MassTransit01	D:\VC2508\MassTransit\MassTransitSln\MassTransit01\BusConfigurer.cs	12
Severity	Code	Description	Project File	Line	Suppression State
Suppression State	DetailsError (active)	CS0103 The name 'Bus' does not exist in the current context	MassTransit01	D:\VC2508\MassTransit\MassTransitSln\MassTransit01\BusConfigurer.cs	14

## Ответ

**Ошибки возникают потому, что классы и интерфейсы MassTransit не импортируются автоматически. Вам необходимо подключить пространство имен, содержащее нужные типы и методы.**

**Исправьте начало файла следующим образом:**

## **using MassTransit**

### **using System;**

```
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;
```

## **namespace MassTransit01**

```
public class BusConfigurer
{
    public static IBusControl CreateBus()
    {
        return Bus.Factory.CreateUsingRabbitMq(cfg =>
        {
            cfg.Host(new Uri("rabbitmq://localhost"), hst => { });

            // Регистрация обработчиков и потребителей
            cfg.ReceiveEndpoint("my_queue_name", ep =>
            {
                ep.Consumer<MyConsumer>();
            });
        });
    }
}
```

## **Что изменилось:**

- 1. Добавлено пространство имен MassTransit, благодаря которому становятся доступными интерфейс IBusControl и статический класс Bus.**

**После внесения изменений соберите проект заново. Если вы добавили правильные зависимости NuGet, то сборка должна пройти успешно.**

**Смотрите весь код:// BusConfigurer.csusing MassTransit;using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading.Tasks;**

```
namespace MassTransit01{ public class BusConfigurer { public static IBusControl CreateBus() { return Bus.Factory.CreateUsingRabbitMq(cfg => { cfg.Host(new Uri("rabbitmq://localhost"), hst => { });
```

## // Регистрация обработчиков и потребителей

```
cfg.ReceiveEndpoint("my_queue_name", ep =>
```

{

```
ep.Consumer<MyConsumer>();
```

} );

} );

1

```
// MyConsumer.cs using MassTransit; using System; using System.Collections.Generic; using  
System.Linq; using System.Text; using System.Threading.Tasks;  
using System.Threading.Tasks; // using GreenPipes; using MassTransit01;
```

```
public class MyConsumer : IConsumer<MyMessage>{ public Task
```

```
Consume(ConsumeContext<MvMessage> context) { var message =
```

```
context.Message.Text; Console.WriteLine($"Received message: {message}");
```

```
return Task.CompletedTask; } }
```

```
// MyMessage.csusing System;using System.Collections.Generic;using System.Linq;using  
System.Text;using System.Threading.Tasks;
```

```
namespace MassTransit01{ public class MyMessage { public string Text { get;  
set; } }}
```

// See <https://aka.ms/new-console-template> for more information using MassTransit01:

```
class Program { static async Task Main(string[] args) { var busControl =
```

### **BusConfigurer.CreateBus():**

```
await busControl.StartAsync(); // Запускаем шину
```

**try**

```

{
    while (true)
    {
        Console.Write("Enter a message to send or 'exit' to quit: ");
        var input = Console.ReadLine();

        if (input == "exit")
            break;

        await busControl.Publish(new MyMessage { Text = input });
    }
}
finally
{
    await busControl.StopAsync(); // Остановка шины
}

}

```

**Все компилируется Но при запуске Консольного Приложения ничего не происходит.**

#### Ответ

Причин, почему ничего не происходит при запуске, может быть несколько. Давайте разберем возможные проблемы и способы их устранения:

#### Возможные причины и пути решения:

##### 1. Отсутствие запущенного сервера RabbitMQ

Убедитесь, что сервер RabbitMQ запущен и доступен по адресу `localhost`. Проверьте наличие работающего экземпляра RabbitMQ и убедитесь, что порт открыт и доступен.

Проверить состояние RabbitMQ можно командой:

`docker ps` # Если используете Docker

либо зайдите в веб-интерфейс RabbitMQ по адресу `http://localhost:15672` (если установлен плагины управления).

##### 2. Проблема с созданием виртуального хоста (vhost)

Иногда проблема связана с отсутствием или неправильностью настроек виртуальных хостов в RabbitMQ. Убедитесь, что задан правильный vhost. По умолчанию RabbitMQ создает / (корневой виртуальный хост), но если настроены другие, убедитесь, что они указаны правильно.

Можно попробовать изменить подключение к хосту следующим образом:

`cfg.Host(new Uri("rabbitmq://localhost/%2f"), hst => { }); // %2f означает '/'`

### 3. Проверка доступности очереди

Проверьте доступность очереди, которую вы пытаетесь создать. Возможно, ваша очередь была создана неправильно или возникла ошибка при попытке подключиться к существующей очереди. Попробуйте сначала проверить наличие очереди в RabbitMQ Management UI.

---

### 4. Правильность имени consumer-а

Убедитесь, что имя consumer-а соответствует ожиданиям RabbitMQ. Обычно MassTransit автоматически генерирует уникальные имена, но иногда лучше задать понятное имя очереди явно:

```
cfg.ReceiveEndpoint("my_queue_name", ep =>
{
    ep.Consumer<MyConsumer>();
});
```

Имя "my\_queue\_name" должно совпадать с ожидаемой вами очередью в RabbitMQ.

---

### 5. Отладка с включенным логгингом

Вы можете включить логирование для выявления проблем:

- Добавьте библиотеку Serilog или NLog для вывода логов.
- Включите трассировку сообщений в MassTransit:

```
log.Logger = new LoggerConfiguration()
    .WriteTo.Console()
    .CreateLogger();
```

```
Bus.Factory.ConfigureWith(loggerFactory);
```

---

### 6. Проблемы с правами доступа

Убедитесь, что учетная запись, используемая для подключения к RabbitMQ, имеет достаточные права для чтения и записи в указанную очередь.

---

Как диагностировать проблему?

Чтобы выяснить причину неполадки, попробуйте запустить приложение с включенными логами. Используйте следующий код для запуска с логированием:

```
static async Task Main(string[] args)
{
    Log.Logger = new LoggerConfiguration()
        .MinimumLevel.Debug()
        .WriteTo.Console()
```

```

.CreateLogger();

var busControl = BusConfigurer.CreateBus();

await busControl.StartAsync(); // Запускаем шину

try
{
    while (true)
    {
        Console.Write("Enter a message to send or 'exit' to quit: ");
        var input = Console.ReadLine();

        if (input == "exit")
            break;

        await busControl.Publish(new MyMessage { Text = input });
    }
}
catch(Exception ex)
{
    Log.Error(ex, "An error occurred.");
}
finally
{
    await busControl.StopAsync(); // Остановка шины
}
}

```

**После добавления логирования запустите приложение вновь и внимательно изучите выводимые логи. Они помогут выявить потенциальные проблемы, вызвавшие отсутствие реакции программы.**

#### Дополнительные советы:

- Проверьте статус RabbitMQ-сервер и очереди в Web UI.
- Перезапустите приложение после изменения конфигурации.
- Обновите версию пакета MassTransit, если используются устаревшие версии.

**Следуя указанным рекомендациям, вы сможете устраниТЬ проблему и убедиться, что MassTransit взаимодействует с RabbitMQ должным образом.**

**Правильно ли я понял, что нужно еще установить Docker-Контейнер RabbitMQ ?**

#### Ответ

**Нет, устанавливать Docker-контейнер RabbitMQ необязательно, хотя это популярный способ быстро развернуть RabbitMQ локально для тестирования и разработки.**

**Вот два основных способа установки и запуска RabbitMQ:**

**Вариант 1: Локальная установка RabbitMQ (без Docker)**

- 1. Скачайте и установите RabbitMQ на своей операционной системе (Windows, Linux, macOS):**
  - Официальная страница загрузки: [Download RabbitMQ](#)
- 2. После установки откройте терминальное окно и запустите службу**  
`RabbitMQ:rabbitmq-server start`
- 3. Установите веб-интерфейс для мониторинга и администрирования:**  
`rabbitmq-plugins enable rabbitmq_management`
- 4. Теперь перейдите в браузер по адресу `http://localhost:15672` и войдите с дефолтными учётными данными (login: guest / password: guest).**

## Вариант 2: Использование Docker-контейнера RabbitMQ

Docker — удобный инструмент для быстрого развёртывания среды разработки. Чтобы воспользоваться контейнером RabbitMQ через Docker, следуйте инструкции ниже:

- 1. Установите Docker, если ещё не сделали этого.**
- 2. Создайте контейнер RabbitMQ:**  
`docker run -d --name rabbitmq-container 5672:5672 -p 15672:15672 rabbitmq:management` Здесь:
  - 5672: стандартный порт AMQP, через который MassTransit общается с RabbitMQ.
  - 15672: порт для веб-интерфейса управления RabbitMQ.
- 3. Перейдите в браузере по адресу `http://localhost:15672` и введите стандартные учетные данные (guest/guest).**

## Что выбрать?

- **Без Docker:** подойдёт, если вам удобно управлять RabbitMQ напрямую, имеете опыт администрирования или хотите использовать встроенные инструменты вашей ОС.
- **С Docker:** отлично подходит для быстрых тестов и временных сред разработки, когда вы хотите избежать сложной установки или желаете сохранить окружение изолированным.

Итак, выбор остаётся за вами, оба варианта работают одинаково хорошо для тестирования и разработки MassTransit + RabbitMQ.

Опишите подробно установку с DockerContainer У меня из средств Docker есть только Программа DockerDesktop Что еще нужно сделать для установки ?

## Ответ

Установка RabbitMQ с помощью Docker Desktop проста и удобна. Если у вас уже установлена программа Docker Desktop, то никаких дополнительных инструментов больше не потребуется. Рассмотрим пошагово, как установить и запустить RabbitMQ в Docker-контейнере.

 Предварительные требования:

- Docker Desktop установлен и работает.

## Инструкция по установке:

### **1 Запустите Docker Desktop**

Откройте программу Docker Desktop на вашем компьютере. Она обеспечит работу с Docker-образами и контейнерами.

### **2 Загрузите образ RabbitMQ**

Запустите терминал или командную строку Windows PowerShell и выполните следующую команду для скачивания официального образа RabbitMQ с поддержкой веб-интерфейса управления:

`docker pull rabbitmq:management`

Эта команда загрузит последнюю стабильную версию RabbitMQ с графическим интерфейсом управления.

### **3 Запустите контейнер RabbitMQ**

После завершения загрузки образа запустите контейнер с необходимыми параметрами:

```
docker run -d --name rabbitmq-container \
  -p 5672:5672 \
  -p 15672:15672 \
  rabbitmq:management
```

Разберём подробнее каждую часть команды:

- `-d`: запускает контейнер в фоновом режиме.
- `--name rabbitmq-container`: задаёт имя вашему контейнеру (можно поменять на любое другое удобное название).
- `-p 5672:5672`: отображает внутренний порт 5672 (AMQP-порт) на внешний порт 5672 вашего компьютера.
- `-p 15672:15672`: отображает порт веб-интерфейса RabbitMQ (15672) на этот же порт на вашем ПК.
- `rabbitmq:management`: использует загруженный образ RabbitMQ с управлением.

Контейнер начнет работу сразу после выполнения команды.

### **4 Проверьте работоспособность контейнера**

Вы можете проверить, запущен ли контейнер, выполнив команду:

`docker ps`

Она покажет список активных контейнеров. Вы увидите строку с именем вашего контейнера ("rabbitmq-container").

### **5 Перейдите в веб-интерфейс RabbitMQ**

**Чтобы увидеть интерфейс RabbitMQ и начать его использовать, откройте браузер и перейдите по адресу:**

**[http://localhost:15672/](http://localhost:15672)**

**Вас попросят ввести логин и пароль. Используемые по умолчанию учётные данные:**

- **Логин: guest**
- **Пароль: guest**

**Войдя в веб-интерфейс, вы получите возможность просматривать очереди, обменники, соединения и другую важную информацию.**

#### **Полезные советы:**

- **Если вы захотите остановить контейнер, выполните команду:**

**`docker stop rabbitmq-container`**

- **Чтобы удалить контейнер (при желании), выполните:**

**`docker rm rabbitmq-container`**

#### **Всё готово!**

**Теперь у вас есть работающий экземпляр RabbitMQ в контейнере Docker, готовый к использованию совместно с MassTransit. Можете приступать к тестированию и разработке ваших приложений.**

#### **Успехов в освоении технологий!**

**Я нахожусь на странице Search results for RabbitMQ я выбрал rabbitmq, щелкнул мышью. Что делать дальше? Справа напротив rabbitmq есть кнопки Pull и Run. Какую кнопку нажимать?**

#### **Ответ**

**Давайте детально рассмотрим дальнейшие шаги после выбора образа RabbitMQ в Docker Hub.**

#### **Когда выбираешь пункт "Pull":**

**Кнопка Pull предназначена для того, чтобы скачать выбранный образ RabbitMQ на ваш компьютер. Нажав на неё, вы инициируете процедуру загрузки образа, но сам контейнер не будет запущен автоматически.**

**После нажатия на Pull, откроется всплывающее окно с инструкцией по команде, которая позволит загрузить образ. Эта команда выглядит примерно так:**

**`docker pull rabbitmq:management`**

**Выполните данную команду в своём терминале или окне PowerShell. Образ скачается, и вы сможете видеть его среди доступных образов Docker, проверив командой:**

**`docker images`**

#### **Когда выбираешь пункт "Run":**

**Кнопка Run открывает инструкцию по быстрому запуску контейнера на основе выбранного образа. То есть, нажимая на Run, вы получаете готовую команду для немедленного запуска контейнера прямо из браузера.**

**Нажмите на кнопку Run. Появится форма, позволяющая заполнить некоторые параметры перед запуском. Важно обратить внимание на пару моментов:**

- **Port mappings:** там указывается, какой порт внешнего мира будет перенаправлен внутрь контейнера. Стандартные порты для RabbitMQ — это 5672 (для связи по протоколу AMQP) и 15672 (порт веб-интерфейса управления).
- **Name:** выберите имя для своего контейнера (например, rabbitmq-container).

**Готовая команда для запуска контейнера появится внизу формы, и её можно**

**скопировать и вставить в терминал. Команда будет похожа на такую:**

```
docker run -d --name rabbitmq-container -p 5672:5672 -p 15672:15672  
rabbitmq:management
```

**Выполнение этой команды запустит контейнер, который немедленно заработает.**

#### ❖ Рекомендация:

**Лучше сначала нажать Pull, чтобы убедиться, что образ скачан на вашу машину.**

**Затем уже нажмите Run, чтобы запустить контейнер.**

**Таким образом, последовательность действий следующая:**

1. Кликните на Pull → выполните предложенную команду.
2. Затем кликните на Run → выполните предложенную команду.

**После этого ваш контейнер с RabbitMQ заработает, и вы сможете обращаться к нему по адресам:**

- Для подключения клиента (например, MassTransit): amqp://localhost:5672
- Веб-интерфейс: http://localhost:15672

**👉 Ваш контейнер RabbitMQ теперь готов к работе!**

**я вижу здесь Host port :5672/tcp - это правильный порт ?**

#### Ответ

**Да, это правильный порт для RabbitMQ. Разберемся детальнее:**

#### Порты в Docker:

- **5672/TCP:** Этот порт предназначен для протокола AMQP (Advanced Message Queuing Protocol), который используется для общения клиентских приложений с сервером RabbitMQ. Именно через этот порт ваши приложения будут отправлять и получать сообщения.
- **15672/TCP:** Порт веб-интерфейса управления RabbitMQ. Через него открывается административный интерфейс, где можно отслеживать очереди, обменники, пользователей и прочие элементы инфраструктуры RabbitMQ.

#### Почему важно отображение порта 5672?

**При создании контейнера Docker отображает внутренние порты RabbitMQ наружу, чтобы внешние клиенты могли общаться с сервером. Формат отображения порта выглядит так:**

**Хост-порт:TCP-протокол <- Внутренний порт контейнера**

**Поэтому запись вида 5672/tcp показывает, что:**

- **Хост-порт (5672) будет использоваться приложениями вне контейнера для подключения к RabbitMQ.**
- **Контейнер ожидает входящие запросы на своем внутреннем порте 5672, которые направляются снаружи через указанный хост-порт.**

### **Проверка портов:**

**Чтобы убедиться, что порты открыты и доступны, можно использовать команду:**

`docker ps`

**Результатом будет таблица, показывающая активные контейнеры и отображаемые порты. Рядом с вашим контейнером RabbitMQ вы должны увидеть строки вроде:**

**PORTS**

**0.0.0.0:5672->5672/tcp, :::5672->5672/tcp, 0.0.0.0:15672->15672/tcp, :::15672->15672/tcp**

**Это подтверждает, что порты правильно сопоставлены и готовы к работе.**

### **Итог:**

**Да, 5672/TCP — это абсолютно верный порт для работы с RabbitMQ, используемый для передачи сообщений через протокол AMQP.**

**давайте сделаем некоторые проверки. Тут что-то много строк выдалось, но вроде написано rabbitmq:latest**

### **Ответ**

**Давай проверим, правильно ли у тебя установлено и запущено всё необходимое для работы с RabbitMQ через Docker. Я перечислю ключевые шаги, чтобы убедиться, что всё настроено верно.**

- **Основные моменты для проверки:**

#### **1. Список доступных образов Docker**

**Сначала убедимся, что образ RabbitMQ действительно скачан. Выполнни команду:**

`docker images`

**Ты увидишь таблицу, похожую на такую:**

REPOSITORY	TAG	IMAGE ID	CREATED
rabbitmq	latest	abcdefghijklmn	X days ago

**Важно убедиться, что образ RabbitMQ присутствует и отмечен тегом latest (это обозначает самую свежую версию).**

#### **2. Проверка работающих контейнеров**

**Затем узнаем, запущен ли контейнер с RabbitMQ. Выполните команду:**  
`docker ps`

**Таблица результатов должна включать строку, подобную следующей:**

CONTAINER ID	IMAGE	COMMAND	PORTS	NAMES
abcdefghi	rabbitmq:latest	"/docker-entrypoint.s..."	0.0.0.0:5672->5672/tcp...	rabbitmq-container

**Обратите внимание на колонку PORTS: там должно быть указано, что порт 5672 (TCP) и 15672 (HTTP для web-интерфейса) отображаются на твоё устройство.**

### 3. Тестирование доступа к веб-интерфейсу

**Теперь попробуй перейти в веб-интерфейс RabbitMQ. Открой браузер и введи адрес:**  
`http://localhost:15672/`

**Ты должен попасть на страницу авторизации. Учётные данные по умолчанию:**

- **Username: guest**
- **Password: guest**

### 4. Проверка сети и соединений

**Используй команду для проверки состояния сетевых связей:**

`docker network ls`

**Там должен присутствовать default bridge (сетевое подключение по умолчанию).**

**Дополнительно можешь проверить сеть, к которой присоединён твой контейнер:**  
`docker inspect rabbitmq-container`

**Ищи раздел "NetworkSettings", чтобы убедиться, что контейнер находится в правильной сети.**

### 5. Пробуем отправить простое сообщение

**Чтобы удостовериться, что связь с RabbitMQ проходит normally, напиши простую программу или используй уже существующий скрипт, чтобы отправить сообщение в очередь и прочитать его. Если сообщение прошло успешно, значит, инфраструктура настроена корректно.**

**! Твоя задача сейчас:**

1. Выполните вышеуказанные команды и проверьте выход.
2. Опишите любые ошибки или несоответствия.
3. Попробуйте зайти в веб-интерфейс и отправить тестовое сообщение через интерфейс или собственный код.

**Если возникнут трудности на каком-то этапе, поделись деталями, и мы продолжим разбираться вместе.**

```
(base) PS C:\Users\Administrator> docker images
REPOSITORY TAG IMAGE ID
CREATED SIZE
rabbitmq latest 8d4336d67d34 10 days ago 238MB
(base) PS C:\Users\Administrator> docker ps
CONTAINER ID IMAGE COMMAND
CREATED STATUS PORTS NAMES
rabbitmq:latest "docker-entrypoint.s..." 7 minutes ago Up 7 minutes 4369/tcp, 5671-5672/tcp, 15691-15692/tcp, 25672/tcp rabbitmq
```

### Ответ

**Судя по результатам, у тебя всё практически настроено правильно, однако есть одна важная вещь, которую стоит исправить:**

**! Ты указал неверные порты для отображения. Сейчас видно, что порты 5672 и 15672 закрыты и не отображаются наружу. Причина в том, что ты использовал стандартную команду запуска контейнера без переопределения портов.**

### Решение:

**Нужно перезапустить контейнер с правильными портами. Сделай следующее:**

**1. Останавливаем текущий контейнер:**

```
docker stop rabbitmq
```

**2. Удаляем старый контейнер (если хочешь оставить образ, то удалять контейнер безопасно):**

```
docker rm rabbitmq
```

**3. Запускаем новый контейнер с правильным отображением портов:**

```
docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:latest
```

### Объяснение аргументов:

- **-d:** запускает контейнер в фоновым режиме.
- **--name rabbitmq:** задаёт имя контейнера.
- **-p 5672:5672:** отображает порт 5672 контейнера наружу.
- **-p 15672:15672:** отображает порт 15672 (web-интерфейс) наружу.
- **rabbitmq:latest:** указывает, что запускаем последний официальный образ RabbitMQ.

### Проверка:

**После повторного запуска снова проверь доступные порты командой:**

```
docker ps
```

**Теперь ты должен увидеть что порты отображаются корректно:**

**POR**T

**0.0.0.0:5672->5672/tcp, 0.0.0.0:15672->15672/tcp**

### Финальный шаг:

**Зайди в браузер и открой админ-панель RabbitMQ по ссылке:**

<http://localhost:15672>

Авторизируйся с логином и паролем:

- Username: guest
- Password: guest

Если интерфейс открылся, значит, всё настроено корректно, и можно начинать тестирование отправки сообщений через MassTransit.

```
(base) PS C:\Users\Administrator> docker stop rabbitmqrabbitmq(base) PS  
C:\Users\Administrator> docker rm rabbitmqrabbitmq(base) PS C:\Users\Administrator>  
docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672  
rabbitmq:latest7d316c2112787959bbad043531a29202bacc1c913313315aa3d97ffbf47b  
78(base) PS C:\Users\Administrator>
```

Ответ

Всё получилось отлично! Давай подведём итог наших действий и проведём финальную проверку.

#### ● Что сделано:

1. Останавливали и удаляли предыдущий контейнер:
  - Использовали команды `docker stop rabbitmq` и `docker rm rabbitmq`.
2. Перезапустили контейнер с корректными портами:
  - Применили правильную команду:  
`docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:latest`
3. Получили успешный запуск нового контейнера:
  - Видим уникальный идентификатор контейнера, что означает, что контейнер стартовал корректно.

#### 🔍 Последняя проверка:

##### Шаг 1: Просмотр статуса контейнера

Проверь состояние контейнера командой:

```
docker ps
```

Результат должен показать нечто подобное:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
NAMES					
7d316c211278	rabbitmq:latest	"docker-entrypoint.s..."	2 minutes ago	Up 2 minutes	
0.0.0.0:5672->5672/tcp, 0.0.0.0:15672->15672/tcp				rabbitmq	

Здесь самое важное — это отображение портов 5672 и 15672, которые соответствуют стандартам RabbitMQ.

##### Шаг 2: Доступность веб-интерфейса

Теперь зайди в браузер и попробуй войти в административную панель RabbitMQ:

<http://localhost:15672>

Авторизация по умолчанию:

- Имя пользователя: guest

- Пароль: guest

После успешного входа ты увидишь панели с информацией о состоянии узлов, обменниках, очередях и пользователях.

### 🏁 Готово!

Поздравляю, теперь у тебя всё настроено правильно. RabbitMQ запущен, порты отображаются корректно, и ты можешь приступить к экспериментам с MassTransit и RabbitMQ.

Продолжай развивать свои проекты, и если появятся новые вопросы — обращайся сюда!

<http://localhost:15672/>

### Ответ

Похоже, ты перешел по указанному адресу и ждёшь дальнейших указаний.

Поздравляю, если веб-интерфейс открылся и ты увидел экран авторизации!

Далее вот твои действия:

### 👤 Авторизация в RabbitMQ:

- Username: guest
- Password: guest

По умолчанию эти учётные данные разрешают полный доступ ко всей инфраструктуре RabbitMQ. Войди с ними.

### 📋 Интерфейс RabbitMQ:

После успешной авторизации ты попадёшь на главную страницу RabbitMQ. Там видны общие метрики, такие как количество соединений, очередей, обменников и прочее.

Посмотри на верхнюю навигационную панель:

- Overview: общая статистика по узлам и производительности.
- Queues: списки существующих очередей.
- Exchanges: существующие обменники (exchanges).
- Connections: активные соединения клиентов.
- Channels: каналы коммуникаций.
- Consumers: действующие потребители сообщений.

### ⌚ Проверка работоспособности:

Ты можешь провести небольшой тест:

#### 1. Создай новую очередь:

- Перейди в раздел Queues.
- Нажми на кнопку Add queue и заполни форму с произвольным именем очереди, например, test\_queue.

#### 2. Отправь тестовое сообщение:

- Вернись на вкладку **Overview** и выбери нужную очередь (например, недавно созданную `test_queue`).
- Найди блок **Messages**, нажми на ссылку рядом с надписью **Publish message**.
- Напечатай произвольное сообщение и нажми **Publish**.

### 3. Просмотри доставленное сообщение:

- Посмотри в списке очереди, увеличилось ли число сообщений.

Если сообщение дошло, значит, твоя система готова к полноценной работе с RabbitMQ и MassTransit.

#### 🔥 Следующие шаги:

Теперь, когда RabbitMQ установлен и работает, ты можешь переходить к созданию собственного сервиса с использованием MassTransit. Тебе предстоит написать приложение, которое сможет публиковать и получать сообщения через RabbitMQ. Дальше планируешь двигаться самостоятельно или нужны конкретные рекомендации по MassTransit?

This page isn't working localhost didn't send any data.

#### Ответ

Сообщение "This page isn't working" свидетельствует о том, что сервер RabbitMQ либо не ответил, либо возникли проблемы с подключением. Попробуем разобраться и решить эту ситуацию.

#### 🛠️ Причины возникновения проблемы:

1. Проблемы с отображением портов: несмотря на успешный запуск контейнера, порты могли быть отображены некорректно.
2. Неправильная версия образа: возможно, последняя версия RabbitMQ привела к ошибкам совместимости.
3. Firewall или антивирус: программное обеспечение безопасности могло заблокировать входящий трафик.
4. Сеть Docker: иногда Docker неправильно формирует сеть между контейнером и хост-машиной.

#### 🤔 Диагностика и решение:

##### Шаг 1: Проверка текущих состояний контейнеров

Проверь, активен ли контейнер и отображаются ли порты правильно:  
`docker ps`

Если видишь такое:

**PORTS**

**0.0.0.0:5672->5672/tcp, 0.0.0.0:15672->15672/tcp**

Значит, порты открыты и отображаются правильно.

## **Шаг 2: Проверка логов контейнера**

**Проверь логи контейнера на предмет возможных ошибок:**

**docker logs rabbitmq**

**Если заметишь какие-нибудь критические ошибки или предупреждения, обрати на них внимание.**

## **Шаг 3: Переустановка контейнера с конкретной версией**

**Возможно, самая новая версия RabbitMQ конфликтует с твоей системой. Попробуй запустить предыдущую версию:**

**docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-management**

**Здесь мы используем стабильную версию 3.x, которая обычно меньше подвержена проблемам.**

## **Шаг 4: Проверка firewall и антивирусов**

**Проверь, не блокирует ли фаервол или антивирус входящий трафик на портах 5672 и 15672. Иногда временные отключения помогают определить источник проблемы.**

## **Шаг 5: Перегрузка Docker Engine**

**Иногда помогает простая перегрузка Docker Engine:**

**docker system prune**

**docker restart**

**Это очистит неиспользуемые ресурсы и перезапустит Docker.**

### **Итоговые рекомендации:**

- 1. Повторно проконтролируй отображение портов.**
- 2. Используй стабильную версию RabbitMQ.**
- 3. Проверь безопасность и настройки сетей.**

### **Дальнейшие действия:**

**Если проблема сохраняется, пожалуйста, дай знать, какие шаги предпринял и что вышло в результате диагностики. Мы будем искать решение дальше.**

```
(base) PS C:\Users\Administrator> docker ps
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS              PORTS
NAMES              7d316c211278    rabbitmq:latest "docker-entrypoint.s..."   6 minutes ago   Up 6
minutes            0.0.0.0:5672->5672/tcp, [::]:5672->5672/tcp, 0.0.0.0:15672->15672/tcp, [::]:15672->15672/tcp
rabbitmq(base) PS C:\Users\Administrator>
```

### **Ответ**

**Отличные новости! Результаты показывают, что порты отображаются корректно и контейнер запущен:**

**PORTS**

**0.0.0.0:5672->5672/tcp, [::]:5672->5672/tcp,  
0.0.0.0:15672->15672/tcp, [::]:15672->15672/tcp**

Это значит, что порт 15672, через который осуществляется доступ к веб-интерфейсу, открыт и доступен для внешних обращений.

### 🔥 Действия дальше:

- Попробуй снова открыть браузер и обратиться по адресу:

**<http://localhost:15672>**

- Введи стандартные учётные данные:
  - Username: guest
  - Password: guest

Если страницы всё равно не открываются, возможны две ситуации:

1. Антивирус или Firewall блокируют доступ. Временное отключение защиты или создание исключения для указанных портов может помочь.
2. Ошибка в конфигурации Docker или RabbitMQ. Тогда рекомендую попробовать следующие шаги:

- Переустановить контейнер с предыдущей версией:

`docker stop rabbitmq`

`docker rm rabbitmq`

`docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-management`

- Проверить журналы Docker:

`docker logs rabbitmq`

Они покажут, были ли ошибки при старте или процессе работы.

👉 Если всё сработало и ты попал в веб-интерфейс, поздравляю! Теперь у тебя полностью рабочая среда для экспериментов с RabbitMQ и MassTransit. Продолжай осваиваться и развиваться!

**<http://localhost:15672/>**

The connection was reset

The connection to the server was reset while the page was loading.

The site could be temporarily unavailable or too busy. Try again in a few moments.

If you are unable to load any pages, check your computer's network connection.

If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the web.

Ответ

Ошибка "The connection was reset" говорит о том, что соединение было прервано, вероятно, из-за блокировки трафика или внутренней ошибки сервера. Рассмотрим наиболее вероятные причины и пути их устранения.

### 🔍 Возможные причины:

1. Фаервол или антивирус блокирует доступ: защитные программы могут считать трафик подозрительным и прерывать соединение.
2. Конфликт сети Docker: проблемы с настройками сети Docker могут привести к невозможности подключения.

- 3. Некорректная работа DNS: иногда DNS-сервер не может разрешить имя localhost.**
- 4. Сервер временно недоступен: временная нагрузка на сервер или сбой могут вызвать сброс соединения.**

#### ❖ План действий:

##### **Шаг 1: Проверка журналов Docker**

**Проанализируем логи текущего контейнера:**

**docker logs rabbitmq**

**Ищи сообщения об ошибках, связанных с отказом в доступе или сбоях в сети.**

##### **Шаг 2: Отключить временное ПО безопасности**

**Проверь, не блокирует ли антивирус или фаервол обращение к порту 15672. Можно временно отключить защиту или добавить исключение для порта.**

##### **Шаг 3: Проверка DNS**

**Проверь, разрешает ли DNS имя localhost:**

**ping localhost**

**Убедись, что IP-адрес совпадает с ожидаемым значением (обычно 127.0.0.1).**

##### **Шаг 4: Перезапуск Docker и контейнера**

**Иногда простая перезагрузка решает многие проблемы:**

**docker stop rabbitmq**

**docker rm rabbitmq**

**docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:latest**

##### **Шаг 5: Альтернативный путь к интерфейсу**

**Если локальное имя не работает, попробуй прямой доступ по IPv4:**

**<http://127.0.0.1:15672>**

#### ❖ Анализ ситуации:

**Если проблема не решается простыми методами, постарайся проанализировать журнал Docker и обрати внимание на поведение системы при обращении к другим портам. Может оказаться полезным взглянуть на мониторинг ресурсов системы (загруженность CPU, RAM, диска) при обращении к интерфейсу.**

#### ❖ Заключение:

**Чаще всего эта ошибка вызвана проблемами с сетевыми настройками или защитой системы. Убедившись, что всё работает, приступай к дальнейшим действиям. Если ситуация не изменится, продолжим углубляться в диагностику.**

```
2025-08-15 00:29:57.510895+00:00 [info] <0.10.0> Time to start RabbitMQ: 11729 ms
2025-08-15 00:38:06.636786+00:00 [info] <0.708.0> accepting AMQP connection
172.17.0.1:49000 -> 172.17.0.2:5672
2025-08-15 00:38:06.642451+00:00 [error] <0.708.0> closing AMQP connection
172.17.0.1:49000 -> 172.17.0.2:5672 (duration: '7ms')
2025-08-15 00:38:06.642451+00:00 [error] <0.708.0> TLS client detected on non-TLS AMQP port.
Ensure the client is connecting to the correct port.
2025-08-15 00:38:06.664301+00:00 [info] <0.715.0> accepting AMQP connection
172.17.0.1:49016 -> 172.17.0.2:5672
2025-08-15 00:38:06.664492+00:00 [error] <0.715.0> closing AMQP connection
172.17.0.1:49016 -> 172.17.0.2:5672 (duration: '0ms')
2025-08-15 00:38:06.664492+00:00 [error] <0.715.0> TLS client detected on non-TLS AMQP port.
Ensure the client is connecting to the correct port.
```

## Ответ

Анализируя логи, видим ключевое сообщение об ошибке:

**TLS client detected on non-TLS AMQP port. Ensure the client is connecting to the correct port.**

Это значит, что клиент пытается подключаться по защищенному каналу (TLS), хотя порт 5672 по умолчанию предназначен для обычного, незашифрованного соединения (незащищённый режим).

### 🔍 Причина ошибки:

Вероятно, у тебя включены SSL/TLS сертификаты на стороне клиента, хотя RabbitMQ сконфигурирован для работы в открытом виде (non-TLS). Это вызывает ошибку аутентификации и закрытие соединения.

### ⭐ Решения:

Есть два подхода к решению проблемы:

#### Вариант 1: Отключить шифрование на клиентской стороне

Самый быстрый способ — выключить шифрование в клиентском приложении. Для этого нужно убрать конфигурацию SSL/TLS из клиентского приложения, которое обращается к RabbitMQ.

- Если это приложение на Python или JavaScript, удостоверься, что не использовано свойство `ssl=True` или аналогичное.
- Если ты пользуешься MassTransit (.NET), удостоверься, что отсутствуют настройки `SecureSocketOptions` или другие SSL-конфигурации.

#### Вариант 2: Включить SSL/TLS на стороне RabbitMQ

Если обязательно нужен зашифрованный канал, нужно дополнительно настроить RabbitMQ для поддержки TLS. Но это требует определённых усилий и тщательной настройки сертификатов.

## Вот упрощённая инструкция для включения TLS в RabbitMQ:

1. Подготовь сертификат и ключ для TLS.
2. Скопируй файлы сертификата и ключа в папку контейнера (например, с помощью docker cp).
3. Измени конфигурацию RabbitMQ, добавив секции TLS (использовать переменные окружения или config-файл).

Пример простого шага (предварительно подготовив сертификаты):

```
docker exec -it rabbitmq bash
```

# Внутри контейнера

```
rabbitmq-plugins enable rabbitmq_amqp_tls  
rabbitmqctl set_permissions guest "[$][$]$" "[$][$]$" "[$][$]$\nrabbitmq-plugins list
```

Затем обнови конфигурационные файлы и настрой правила NAT для TCP/IP (если применимо).

### 💡 Резюме:

Скорее всего, причина ошибки кроется в неправильном взаимодействии клиента с портом 5672, предназначенному для незашифрованного канала. Нужно согласовать стороны и гарантировать, что обе используют одинаковый тип соединения (TLS или обычный).

Начни с первого варианта — убери лишние настройки SSL/TLS на клиентской стороне. Если это не решит проблему, обсудим второй вариант.

Давайте лучше у меня на компьютере выключим эти TLS

## Ответ

Отключение TLS на клиентской стороне — хороший выбор, особенно если защита каналов пока не обязательна. Ниже приведены шаги для разных популярных языков программирования и платформ, как отключить TLS и обеспечить обычное, незашифрованное соединение с RabbitMQ.

### ⌚ Настройки для различных клиентов:

#### 1. Python (Pika):

Убедись, что при создании соединения ты не используешь TLS:

```
import pika
```

```
connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))  
channel = connection.channel()
```

Если раньше использовали SSL/TLS, то уберите соответствующие аргументы:

```
parameters = pika.ConnectionParameters(  
    host='localhost',  
    ssl_options=None # Убираем TLS  
)
```

#### 2. JavaScript (Amqplib):

**Убедись, что нет дополнительного параметра для TLS:**

```
const amqp = require('amqplib');
```

```
amqp.connect('amqp://localhost')
.then((conn) => console.log('Connected successfully'));
```

**Если раньше использовалось TLS, удали соответствующий блок:**

```
amqp.connect({
  protocol: 'amqp', // Без префикса s
  hostname: 'localhost'
})
```

### 3. C#/.NET (MassTransit):

**Массовая библиотека MassTransit обычно не требует дополнительной конфигурации TLS. Тем не менее, удостоверьтесь, что не используете никакие специальные настройки безопасного сокета (SecureSocketOptions):**

```
cfg.Host(new Uri("rabbitmq://localhost"));
```

**Убери или закомментируй любую секцию, относящуюся к настройке TLS:**

```
// Нет необходимости в специальных настройках для TLS
```

### 4. Node.js (node-amqp):

**Отредактируйте конфигурацию подключения, исключив использование TLS:**

```
const amqp = require('amqplib/callback_api');
```

```
amqp.connect('amqp://localhost', function(err, conn) {
  if (!err) console.log('Connected!');
});
```

### 5. Go (amqp):

**Измените параметры подключения, убрав необходимость в TLS:**

```
conn, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
if err != nil {
  log.Fatalf("%s", err)
}
```

#### 🔧 Проверка:

**После удаления любых настроек TLS перезапустите своё приложение и попытайтесь повторно подключиться к RabbitMQ. Должно появиться уведомление о подключении в журнале Docker:**

```
docker logs rabbitmq
```

**Ищите сообщения типа:**

```
accepting AMQP connection ...
```

**Если удастся подключиться без TLS-сообщений об ошибках, значит, задача выполнена успешно.**

## ❖ Важное замечание:

**Хотя отключение TLS делает разработку проще, помни, что передача данных становится открытой и уязвимой для перехвата. Всегда подумай дважды, прежде чем разворачивать production-версию без TLS-шифрования.**

где мне запускать эту команду `cfg.Host(new Uri("rabbitmq://localhost"));`?

## Ответ

**Команда `cfg.Host(new Uri("rabbitmq://localhost"));` выполняется в исходном коде вашего приложения на .NET, которое использует MassTransit для подключения к RabbitMQ.**

**Допустим, у вас есть отдельный класс, отвечающий за инициализацию шины MassTransit (например, класс BusConfigurer). Обычно в таком классе содержится логика построения и запуска шины.**

**Приведу общий шаблон того, как такая конфигурация может выглядеть в коде:**

`using MassTransit;`

```
public class BusConfigurer
{
    public static IBusControl CreateBus()
    {
        return Bus.Factory.CreateUsingRabbitMq(cfg =>
        {
            // Указываем хост для подключения к RabbitMQ
            cfg.Host(new Uri("rabbitmq://localhost"));

            // Другие настройки (очереди, точки приёма и т.п.)
            cfg.ReceiveEndpoint("your_queue_name", endpointConfigurator =>
            {
                endpointConfigurator.Consumer<YourConsumer>();
            });
        });
    }
}
```

**Где разместить этот код?**

**Обычно этот фрагмент размещается в специальном месте в вашем проекте, например:**

- **В файле конфигурации Program.cs или Startup.cs в проектах ASP.NET Core.**
- **В отдельном классе-настроечнике (например, тот самый BusConfigurer).**

**Если у вас проект, созданный на основе шаблона MassTransit, скорее всего, у вас уже есть подобный класс, где определяется точка подключения к RabbitMQ.**

**Пример размещения:**

**Допустим, у вас имеется файл Program.cs или Startup.cs, где нужно вызвать метод создания шины:**

```
class Program
{
    static async Task Main(string[] args)
    {
        var busControl = BusConfigurer.CreateBus();
        await busControl.StartAsync();

        // Основная логика приложения...

        await busControl.StopAsync();
    }
}
```

#### ❖ Итог:

- Код cfg.Host(new Uri("rabbitmq://localhost")); помещаете в метод, ответственный за построение и настройку шины MassTransit.
- Далее, этот метод вызывается в основном потоке приложения (чаще всего в точке входа — Main или эквивалентной).

Убедитесь, что у вас отсутствует дополнительная конфигурация TLS, и тогда соединение должно произойти без проблем.

Enter a message to send or 'exit' to quit: 12345Enter a message to send or 'exit' to quit:  
Received message: 12345

#### Ответ

**Великолепно! Сообщение прошло успешно, и вы получили подтверждение о приёме.**

Значит, вся цепочка MassTransit → RabbitMQ → Consumer работает безупречно.

Давайте обобщим пройденный путь и выделим важные этапы:

#### ❖ Основные шаги:

- 1. Установили Docker Container с RabbitMQ:**
  - Скачали и запустили контейнер с последней версией RabbitMQ.
- 2. Создали инфраструктуру для MassTransit:**
  - Написали конфигурацию для шины и потребителя.
  - Определили модель сообщения и реализовали обработку.
- 3. Тестировали передачу сообщений:**
  - Отправляли сообщения через консольное приложение.
  - Получали подтверждения о доставке.

#### 💻 Что делать дальше?

- 🚀 **Расширяйте функциональность:** создавайте новые типы сообщений, увеличивайте количество очередей и обменников.

- **Изучайте шаблоны проектирования:** знакомьтесь с паттернами Publish/Subscribe, Request/Reply и Command/Event.
- **Документация и поддержка:** регулярно обращайтесь к официальной документации MassTransit и RabbitMQ для глубокого изучения возможностей.

### ⌚ Ваши достижения:

Сегодня вы успешно установили и настроили RabbitMQ с MassTransit, создали собственное приложение и отправили первое сообщение. Это большой шаг вперёд в изучении асинхронных систем и работы с очередями сообщений.

Желаю успехов в дальнейшем развитии и обучении! Если возникнут вопросы — обращайтесь.

Но яничего не делал с портами. Я просто запустил нашу Программу

```
public class BusConfigurer{    public static IBusControl CreateBus() {        return Bus.Factory.CreateUsingRabbitMq(cfg => {            cfg.Host(new Uri("rabbitmq://localhost"), hst => {});  
            // Регистрация обработчиков и потребителей  
            cfg.ReceiveEndpoint("my_queue_name", ep =>  
            {  
                ep.Consumer<MyConsumer>();  
            });  
        });  
    }  
}
```

### Ответ

Действительно, если вы указали адрес "rabbitmq://localhost", MassTransit автоматически выберет стандартные порты для RabbitMQ, которые соответствуют обычному (незашифрованному) соединению. Таким образом, вручную указывать порт не требуется, если вы пользуетесь стандартной схемой адресации.

### ❖ Разбор ситуаций:

#### 1. Почему вообще работали порты?

- При запуске контейнера RabbitMQ с указанием портов (-p 5672:5672 -p 15672:15672), Docker обеспечивает автоматическое перенаправление стандартных портов (5672 для AMQP и 15672 для веб-интерфейса).
- Даже если вы не прописываете порт явно в коде MassTransit, библиотека сама выбирает стандартный порт 5672, поскольку схема "rabbitmq://" подразумевает именно этот порт.

#### 2. Зачем мы говорили о проблемах с портами? Ранее мы столкнулись с проблемой, связанной с попыткой установления TLS-подключения на обычном порте. После исправления клиентской стороны (отключения принудительного TLS), вы смогли продолжить работу без указания конкретных портов в коде.

#### 3. Рекомендации по практике:

- Если вы решите позже использовать TLS (шифрование), придется явно указать порт (например, 5671 для TLS) и настроить соответствующие параметры безопасности.
  - Пока вы разрабатываете локально и без особых требований к защите, стандартного подхода хватит.



## **Результат:**

**Ваш текущий подход рабочий и оптимальный для начальной стадии разработки. Пока вы не планируете использовать дополнительное шифрование или сложную топологию сети, можете спокойно пользоваться простым способом подключения, указанным в примере.**

**Спасибо за продуктивную сессию и отличные результаты! Если возникнут вопросы или идеи по улучшению — смело делитесь ими.**

**Сейчас я Вам выложу весь код нашего Приложения.Давайте разобьем этот код, чтобы было похоже на реальные Приложения.То есть надо написать два Приложения.В начале Первое Приложение пишет сообщение Второму Приложению, которое читает это сообщение и логирует его.А затем Второе сообщение делает паузу и затем наоборот Втоое Приложение Пишет сообщение Первому Прилоржению, а Первое Приложение читает логирует это сообщенте у себя.И так в цикле 20 раз.Тогда можно будет сказать, что у нас будет реализован дуплексный обмен.**

```
Вот код нашего Приложения. Нам нужно из одного сделать два Приложения.//  
BusConfigurer.csusing MassTransit;using System;using System.Collections.Generic;using  
System.Linq;using System.Text;using System.Threading.Tasks;  
namespace MassTransit01{ public class BusConfigurer { public static IBusControl  
CreateBus() { return Bus.Factory.CreateUsingRabbitMq(cfg => {  
cfg.Host(new Uri("rabbitmq://localhost"), hst => { });  
    // Регистрация обработчиков и потребителей  
    cfg.ReceiveEndpoint("my_queue_name", ep =>  
    {  
        ep.Consumer<MyConsumer>();  
    });  
});  
}}  
}
```

```

// MyConsumer.cs using MassTransit; using System; using System.Collections.Generic; using
System.Linq; using System.Text; using System.Threading.Tasks;
using System.Threading.Tasks; // using GreenPipes; using MassTransit01;

public class MyConsumer : IConsumer<MyMessage> { public Task
Consume(ConsumeContext<MyMessage> context) { var message =
context.Message.Text; Console.WriteLine($"Received message: {message}");
return Task.CompletedTask; }

// MyMessage.cs using System; using System.Collections.Generic; using System.Linq; using
System.Text; using System.Threading.Tasks;
namespace MassTransit01 { public class MyMessage { public string Text { get; set;
} } } // See https://aka.ms/new-console-template for more information using MassTransit01;
class Program { static async Task Main(string[] args) { var busControl =
BusConfigurer.CreateBus();
await busControl.StartAsync(); // Запускаем шину

try
{
    while (true)
    {
        Console.Write("Enter a message to send or 'exit' to quit: ");
        var input = Console.ReadLine();

        if (input == "exit")
            break;

        await busControl.Publish(new MyMessage { Text = input });
    }
}
finally
{
    await busControl.StopAsync(); // Остановка шины
}
}
}

```

## Ответ

Идея разделить одно приложение на два отдельных сервиса звучит разумно и приближенно к реальной жизни. Дуплексный обмен (обмен сообщениями туда-обратно) будет хорошим примером двустороннего взаимодействия между двумя независимыми сервисами.

Давайте распланируем архитектуру и реализуем нужный сценарий:

- Архитектура двух приложений:

- Приложение А: отправляет сообщение Приложению В и ждёт ответ.
- Приложение В: принимает сообщение от А, обрабатывает его и отправляет ответ обратно.
- Циклический обмен продолжается до завершения (например, до 20 итераций).

 **Реализуем поэтапно:**

**Этап 1: Определение моделей сообщений**

**Нам понадобятся две модели сообщений: одна для отправки от А к В, вторая — для ответа от В к А.**

```
// MyMessage.cs
using System;
```

```
namespace MassTransitAppA
```

```
{
    public class MyMessage
    {
        public string Text { get; set; }
    }
}
```

```
// ResponseMessage.cs
```

```
using System;
```

```
namespace MassTransitAppB
```

```
{
    public class ResponseMessage
    {
        public string ReplyText { get; set; }
    }
}
```

**Этап 2: Конфигурация шины и регистрация точек приема**

**Приложение А (отправляет и слушает ответ):**

```
// AppABusConfigurer.cs
using MassTransit;
```

```
namespace MassTransitAppA
```

```
{
    public class AppABusConfigurer
    {
        public static IBusControl CreateBus()
        {
            return Bus.Factory.CreateUsingRabbitMq(cfg =>
            {

```

```

cfg.Host(new Uri("rabbitmq://localhost"), hst => { });

// Регистрация приёмника сообщений от Б
cfg.ReceiveEndpoint("app_a_response_endpoint", ep =>
{
    ep.Consumer<ResponseConsumer>();
});

}

}

// ResponseConsumer.cs
using MassTransit;

namespace MassTransitAppA
{
    public class ResponseConsumer : IConsumer<ResponseMessage>
    {
        public Task Consume(ConsumeContext<ResponseMessage> context)
        {
            var reply = context.Message.ReplyText;
            Console.WriteLine($"Received response from B: {reply}");
            return Task.CompletedTask;
        }
    }
}

```

**Приложение В (слушает сообщение от А и отправляет ответ):**

```

// AppBBusConfigurer.cs
using MassTransit;

namespace MassTransitAppB
{
    public class AppBBusConfigurer
    {
        public static IBusControl CreateBus()
        {
            return Bus.Factory.CreateUsingRabbitMq(cfg =>
            {
                cfg.Host(new Uri("rabbitmq://localhost"), hst => { });

                // Регистрация приёмника сообщений от А
                cfg.ReceiveEndpoint("app_b_request_endpoint", ep =>
                {
                    ep.Consumer<MessageConsumer>();
                });
            });
        }
    }
}

```

```

    }

}

// MessageConsumer.cs
using MassTransit;

namespace MassTransitAppB
{
    public class MessageConsumer : IConsumer<MyMessage>
    {
        private readonly IRequestClient<ResponseMessage> _responseClient;

        public MessageConsumer(IRequestClient<ResponseMessage> responseClient)
        {
            _responseClient = responseClient;
        }

        public async Task Consume(ConsumeContext<MyMessage> context)
        {
            var originalMsg = context.Message.Text;
            Console.WriteLine($"Received request from A: {originalMsg}");

            // Формируем ответ
            var response = new ResponseMessage { ReplyText = $"Processed by B: {originalMsg}" };

            // Отправляем ответ назад
            await _responseClient.RespondAsync(response);
        }
    }
}

```

### Этап 3: Реализация главного цикла обмена

#### Главный цикл Приложения А:

```

// Program.cs (AppA)
using MassTransit;
using System;
using System.Threading.Tasks;

namespace MassTransitAppA
{
    class Program
    {
        static async Task Main(string[] args)
        {
            var busControl = AppABusConfigurer.CreateBus();

```

```
await busControl.StartAsync();

try
{
    int count = 0;
    while (count++ < 20)
    {
        // Отправляем сообщение Приложению В
        await busControl.Publish(new MyMessage { Text = $"Ping #{count}" });

        // Пауза для ожидания ответа
        await Task.Delay(TimeSpan.FromSeconds(1));
    }
}

finally
{
    await busControl.StopAsync();
}
```

## **Главный цикл Приложения В:**

```
// Program.cs (AppB)
using MassTransit;
using System;
using System.Threading.Tasks;

namespace MassTransitAppB
{
    class Program
    {
        static async Task Main(string[] args)
        {
            var busControl = AppBBusConfigurer.CreateBus();
            await busControl.StartAsync();

            try
            {
                // Ждем поступления сообщений
                Console.WriteLine("Waiting for messages...");
                await Task.Delay(-1); // бесконечно ждем
            }
            finally
            {
                await busControl.StopAsync();
            }
        }
    }
}
```

```
}
```

### ⌚ Запуск обоих приложений:

1. Запустите сначала Приложение В (оно будет ждать входящих сообщений).
2. Затем запустите Приложение А, оно начнёт отправлять сообщения каждые секунды и ожидать ответ.

### ⭐ Итог:

Мы разбили первоначальное приложение на два независимых сервиса, реализовав полноценный дуплексный обмен сообщениями с RabbitMQ через MassTransit. Каждый сервис способен передавать и принимать сообщения, обеспечивая двустороннюю коммуникацию.

Теперь у вас есть готовая основа для дальнейшего развития реальных микросервисных решений.

[Провести тест-драйв GigaChat](#)

### Источники

### Источники