Asp.net core Web API микросервисы. Часть 1

Запросы между сервисами



# На этом уроке

На прошлом уроке мы полностью доделали приложение агент сбора-метрик (если выполнили домашнее задание). Наш менеджер метрик должен теперь как-то получить метрики с агента и положить полученное в свою базу данных. На этом уроке мы познакомимся с тем, как можно запрашивать данные у сервисов по протоколу HTTP. Наш менеджер метрик при помощи http клиента к сервису агенту сбора метрик будет забирать собранные метрики и сохранять их к себе в базу данных.

# 

Оглавление

[На этом уроке](#_4wfp8dsrmxhk)

[Вступление](#_n93g6yblf23c)

[HttpClient](#_hdudgc4rvpjg)

[Polly](#_vxosdy9w9kza)

[Клиент к сервису агенту сбора метрик](#_pcve4ib2cni5)

[Практическое задание](#_tf1198bb3721)

[Список используемой литературы](#_54hssoocdbld)

[Дополнительные материалы](#_oreyj64p9zko)

# Вступление

Микросервисы взаимодействуют между собой и передают различные данные. Чаще всего это запросы по протоколу HTTP. Микросервисы имеют свою выделенную зону ответственности (каждый сервис отвечает за что-то определенное). Например, один сервис может отвечать за хранение данных о пользователях системы, а другой при получении запроса спрашивать у сервиса с пользователями, действительно ли существует пользователь, который хочет что-то спросить. В нашем примере у нас есть микросервис агент сбора метрик, который мы доделали на прошлом уроке. И менеджер сбора метрик, который будет спрашивать у сервисов-агентов данные о собранных ими метриках. В этом уроке мы разберемся как можно выполнять такие запросы между сервисами по протоколу HTTP.

# HttpClient

Мы будем писать клиентскую библиотеку для сервиса агента сбора метрик. Эту библиотеку мы подключим в проект сбора метрик. Код, который будет выполнять запросы, будет работать при помощи HttpClient. Рассмотрим, как его можно подключить и настроить.

Создавать HttpClient напрямую не нужно, лучше воспользоваться IHttpClientFactory. IHttpClientFactory можно зарегистрировать и использовать для настройки и создания экземпляров HttpClient в приложении. IHttpClientFactory дает следующие преимущества:

* Управляет объединением и временем жизни базовых экземпляров HttpClientMessageHandler. Автоматическое управление позволяет избежать распространенных проблем с DNS (системой доменных имен), которые возникают при ручном управлении временем жизни HttpClient.
* Можно создавать именованные клиенты
* Логирование централизовано для клиентов

Добавить клиента к проекту просто. Достаточно вызвать в конфигурировании сервисов в DI следующий код

|  |
| --- |
| services.AddHttpClient(); |

Рассмотрим как можно получить ответ от сервиса при помощи httpClient

|  |
| --- |
| [HttpGet("agent/{agentId}/from/{fromTime}/to/{toTime}")] public IActionResult GetMetricsFromAgent([FromRoute] int agentId, [FromRoute] TimeSpan fromTime, [FromRoute] TimeSpan toTime) {  var request = new HttpRequestMessage(HttpMethod.Get,  "http://localhost:50343/api/cpumetrics/from/1/to/999999");  request.Headers.Add("Accept", "application/vnd.github.v3+json");  var client = clientFactory.CreateClient();  HttpResponseMessage response = client.SendAsync(request).Result;  if (response.IsSuccessStatusCode)  {  using var responseStream = response.Content.ReadAsStreamAsync().Result;  var metricsResponse = JsonSerializer.DeserializeAsync  <CpuMetricsResponse>(responseStream).Result;  }  else  {  *// ошибка при получении ответа*  }  return Ok(); } |

Мы сначала создали объект запроса, в котором указали по какому http методу и пути обращаться за ответом. После при помощи клиента отправили запрос и удостоверившись, что статус код ответа соответствует успешному, прочитали и десериализовали результат в объект.

# Polly

IHttpClientFactory интегрируется со сторонней библиотекой Polly. Polly - это библиотека для обеспечения отказоустойчивости и обработки временных сбоев для .NET. Она позволяет разработчикам создавать политики запросов, такие как Retry, Circuit Breaker, Timeout, Bulkhead Isolation и Fallback. Предоставляются методы расширения, позволяющие использовать политики Polly с настроенными экземплярами HttpClient. Расширения Polly поддерживают добавление к клиентам обработчиков на основе Polly. Polly требуется пакет NuGet Microsoft.Extensions.Http.Polly, установите этот пакет через менеджер пакетов nuget в проекте сервиса менеджера сбора метрик.

Например, мы можем настроить политику для повторного выполнения запроса, если сервер ответил ошибкой следующим образом

|  |
| --- |
| services.AddHttpClient<IMetricsAgentClient, MetricsAgentClient>()  .AddTransientHttpErrorPolicy(p =>   p.WaitAndRetryAsync(3, \_ => TimeSpan.FromMilliseconds(1000)))*;* |

Таким кодом регистрации клиента мы сообщаем, что клиент будет работать с политикой повторных запросов и если сервер ответит ошибкой на запрос, то клиент попробует до трех раз повторить запрос с задержкой между запросами в одну секунду.

# Клиент к сервису агенту сбора метрик

Теперь мы можем спроектировать клиенты к сервису агенту сбора метрик. Начнем с его интерфейсов.

|  |
| --- |
| using MetricsAgent.Responses;  namespace MetricsAgent.Client {  public interface IMetricsAgentClient  {  AllRamMetricsApiResponse GetAllRamMetrics(GetAllRamMetricsApiRequest request);    AllHddMetricsApiResponse GetAllHddMetrics(GetAllHddMetricsApiRequest request);  DonNetMetricsApiResponse GetDonNetMetrics(DonNetHeapMetrisApiRequest request);  AllCpuMetricsApiResponse GetCpuMetrics(GetAllCpuMetricsApiRequest request);  }  } |

Такими простыми выглядят интерфейсы клиента для нашего агента сбора метрик. В его реализацию мы инъектируем HttpClient. Приведем пример реализации одного из наших клиентов

|  |
| --- |
| using MetricsAgent.Responses; using MetricsAgent.Requests; using Microsoft.Extensions.Logging; using System; using System.Net.Http; using System.Text.Json;  namespace MetricsAgent.Client  {  public class MetricsAgentClient : IMetricsAgentClient  {  private readonly HttpClient \_httpClient;  private readonly ILogger \_logger;  public MetricsAgentClient(HttpClient httpClient, ILogger logger)  {  \_httpClient = httpClient;  \_logger = logger;  }   public AllHddMetricsApiResponse GetAllHddMetrics(GetAllHddMetricsApiRequest request)  {  var fromParameter = request.FromTime.TotalSeconds;  var toParameter = request.ToTime.TotalSeconds;  var httpRequest = new HttpRequestMessage(HttpMethod.Get, $"{request.ClientBaseAddress}/api/hddmetrics/from/{fromParameter}/to/{toParameter}");  try  {  HttpResponseMessage response = httpClient.SendAsync(httprequest).Result;   using var responseStream = response.Content.ReadAsStreamAsync().Result;  return JsonSerializer.DeserializeAsync<AllHddMetricsApiResponse>(responseStream).Result;  }  catch (Exception ex)  {  logger.LogError(ex.Message);  }  }   return null;  }  // остальные методы реализовать самим } |

Классы с суффиксом ApiRequest (такие как GetAllHddMetricsApiRequest) содержат три поля - адрес агента (ведь у нас их много может быть зарегистрировано через AgentContoller), дату начала и дату конца выборки. В конструктор нашей реализации инжектирован логер, при помощи которого мы будем писать в файл лога все сообщения об ошибках, а также httpClient. Таким образом, для того, чтобы использовать приложение агента сбора метрик через его клиент в приложении менеджере сбора метрик необходимо зарегистрировать IHttpClientFactory и реализацию нашего клиента, которому из IHttpClientFactory будет предоставлен httpClient. Регистрация клиента может выглядеть следующим образом

|  |
| --- |
| services.AddHttpClient<IMetricsAgentClient, MetricsAgentClient>().AddTransientHttpErrorPolicy(p => p.WaitAndRetryAsync(3, \_ => TimeSpan.FromMilliseconds(1000))); |

Мы зарегистрировали в DI наш и наш клиент, и HttpClient одной командой.

Клиент для сбора метрик можно инжектировать в контроллеры менеджера метрик и использовать, либо (в домашнем задании) инъектировать в периодическую задачу опроса агентов. Рассмотрим на примере одного из методов контроллера обращение к клиенту.

|  |
| --- |
| [HttpGet("agent/{agentId}/from/{fromTime}/to/{toTime}")] public IActionResult GetMetricsFromAgent([FromRoute] int agentId, [FromRoute] TimeSpan fromTime, [FromRoute] TimeSpan toTime) {  *// логируем, что мы пошли в соседний сервис*  \_logger.LogInformation($"starting new request to metrics agent");  *// обращение в сервис*  var metrics = metricsAgentClient.GetCpuMetrics(new GetAllCpuMetricsRequest  {  FromTime = fromTime,  ToTime = toTime  });    *// возвращаем ответ*  return Ok(metrics); } |

# Практическое задание

* Подключите работу с базой данных к приложению менеджера сбора метрик. Все таблицы Metrics Manager должны повторять таблицы Metrics Agent, но иметь дополнительный столбец agentId (для связи метрики с агентом). Таблица Agents Должна иметь два столбца - AgentId и AgentUrl.
* Реализуйте в Metrics Quarz Job. Сделайте Job на каждый типа метрик (как и для агента). Инъектируйте в Job репозиторий работы с агентами, репозиторий работы с соответствующей метрикой, и IMetricsAgentClient. Читайте из базы всех агентов и для каждого в цикле вызывайте методы получения метрик с агента. ApiRequest будет содержать URL агента (вы возьмете его из свойства AgentUrl в базе) и дату начала выборки и дату конца. В качестве даты конца можно брать DateTime.UtcNow. В качестве даты начала - максимальную из дат, сохраненных в БД метрик на менеджере (сделайте соответствующий запрос к базе метрик к нужной таблице). Прочитанные данные из IMetricsAgentClient сохраните в базе данных менеджера, используя репозиторий метрики.
* Реализуйте методы GET контроллеров MetricsManager так, чтобы они по аналогии с MetricsAgent, ходили в свою базу данных и возвращали метрики.

# Список используемой литературы

* <https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/console-webapiclient>
* <https://docs.microsoft.com/en-us/aspnet/web-api/overview/advanced/calling-a-web-api-from-a-net-client>
* <https://habr.com/ru/post/424873/>
* <https://zetcode.com/csharp/httpclient/>
* <https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient>
* <https://github.com/App-vNext/Polly>
* <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/implement-resilient-applications/implement-http-call-retries-exponential-backoff-polly>
* <https://medium.com/@therealjordanlee/retry-circuit-breaker-patterns-in-c-with-polly-9aa24c5fe23a>

# Дополнительные материалы

* <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/implement-resilient-applications/implement-circuit-breaker-pattern>
* <https://habr.com/ru/company/dododev/blog/503376/>