Приемы экономии памяти в .NET

Или дорогие файлы для S3

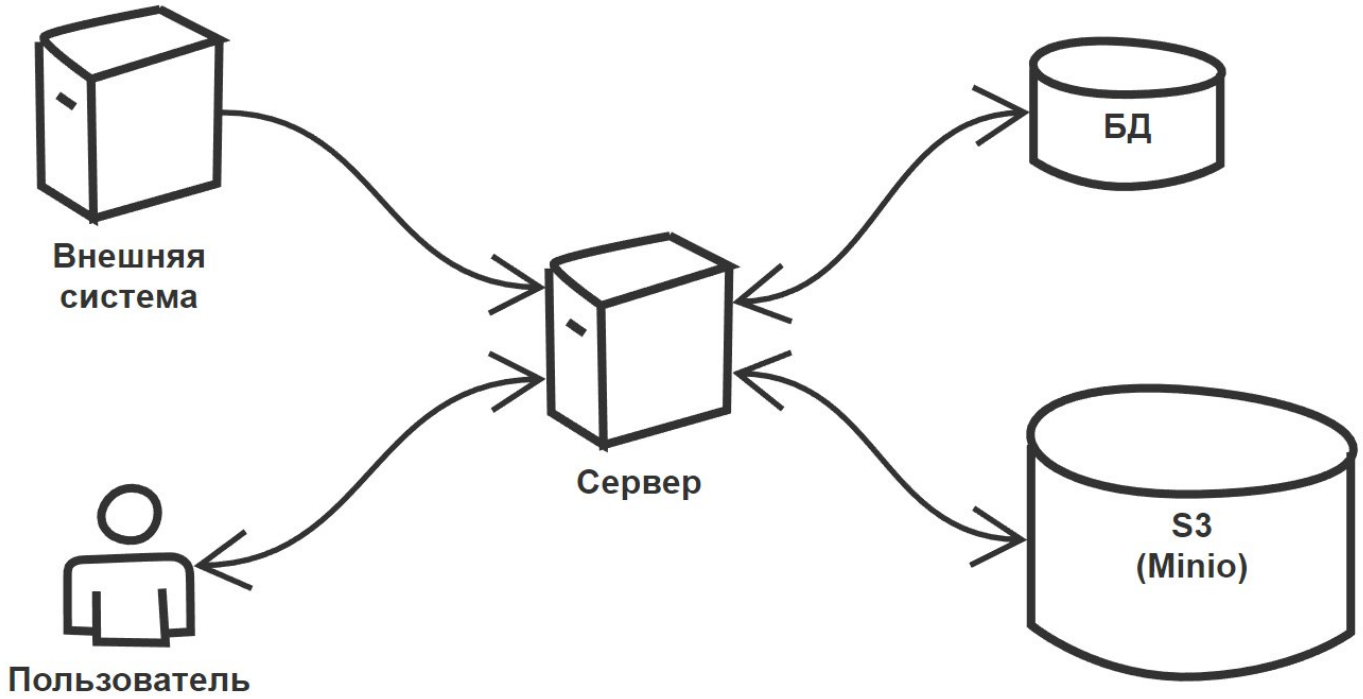
https://t.me/csharp_gepard

Что вы узнаете

- Техники написания кода с низкой аллокацией
- Как экономно работать с HttpClient
- Применение ArrayPool
- Использование stackalloc
- Зачем это всё в повседневной разработке?
- Где я про это узнал?

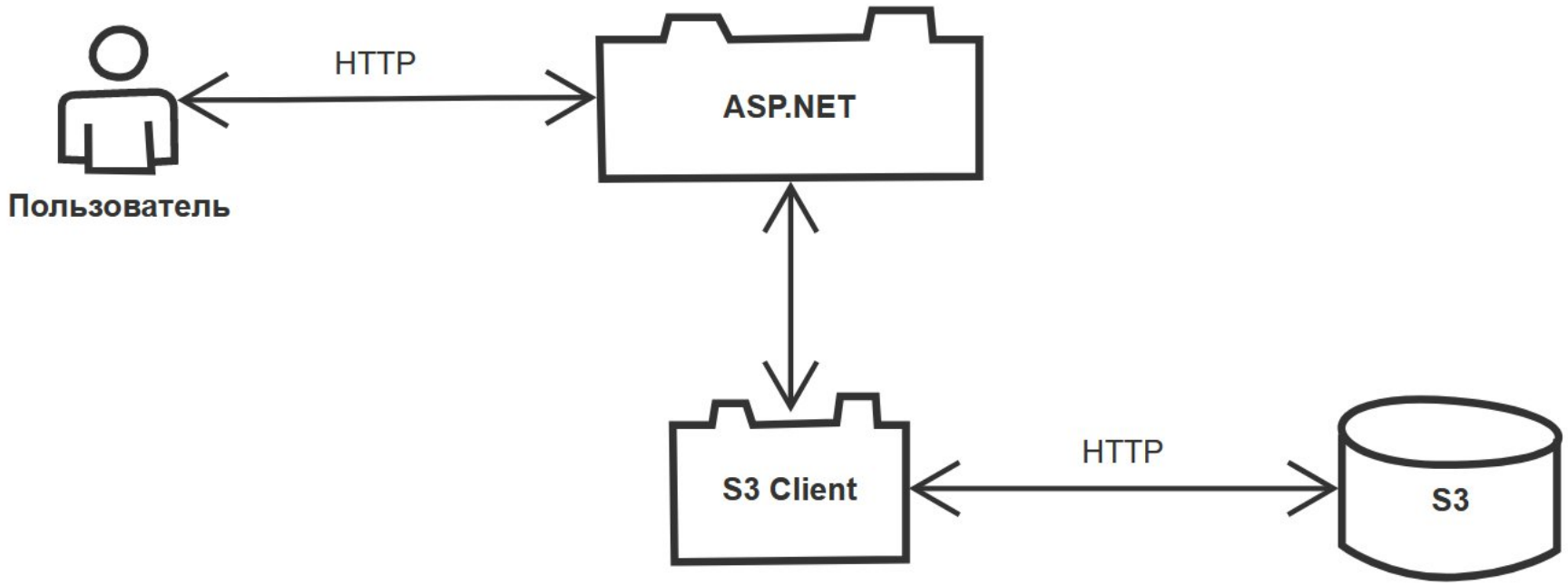
Есть система хранения документов

- Много читателей
- Есть писатели
- Физически файлы в S3
- Отдаём поток байт
- S3-клиент



Проблема: OutOfMemory

- Средний файл 123 МБ
- K8S
- .NET 7
- REST



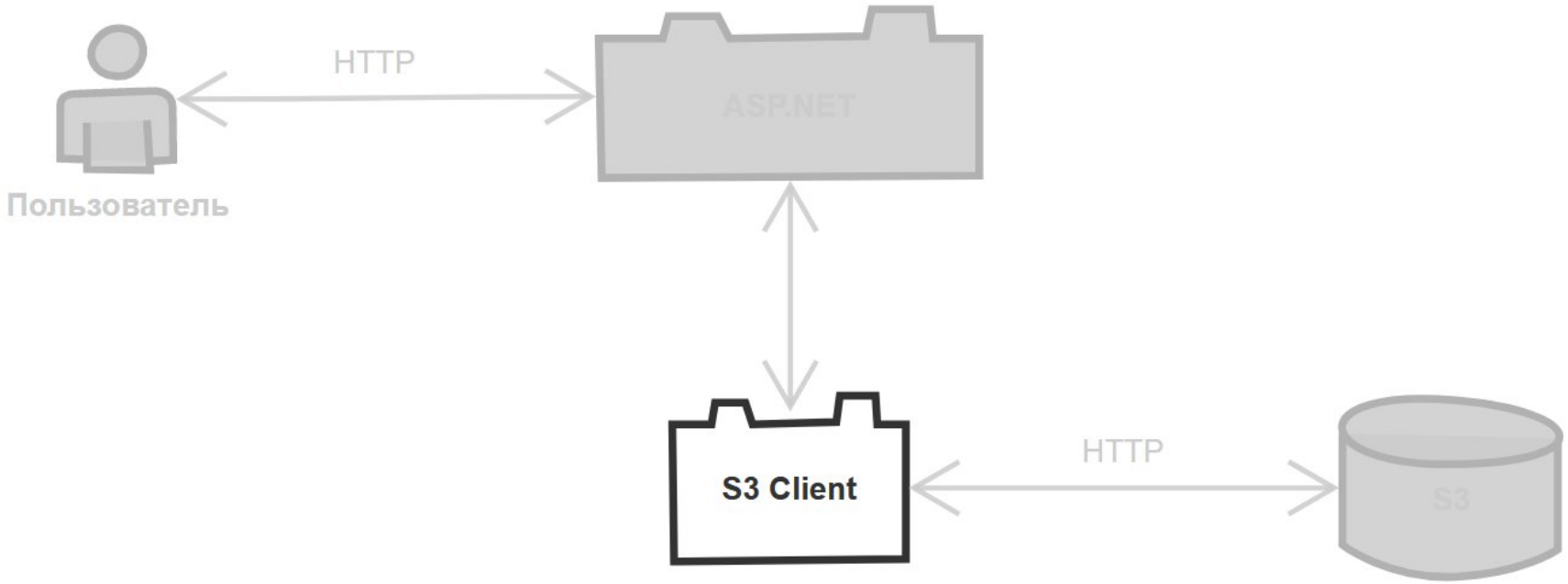
Проблема: OutOfMemory

Применяем все рекомендации от Microsoft:

- Настраиваем endpoint (DisableFormValueModelBinding)
- Используем MultipartReader
- Решаем вопрос с валидацией данных

Проблема: OutOfMemory

- Средний файл 123 МБ
- K8S
- .NET 7
- REST



Проблема: OutOfMemory

- Средний файл 123 МБ
- Загружаем
- Выгружаем
- Удаляем

```
BenchmarkDotNet = v0.13.5, OS=Windows 11 (10.0.22621.1265/22H2/2022Update/SunValley2)
AMD Ryzen 7 5800H with Radeon Graphics, 1 CPU, 16 logical and 8 physical cores
.NET SDK = 7.0.102
           [Host]   : .NET 7.0.2 (7.0.222.60605), X64 RyuJIT AVX2 DEBUG
           .NET 7.0 : .NET 7.0.2 (7.0.222.60605), X64 RyuJIT AVX2

Job = .NET 7.0  Runtime=.NET 7.0
```

Method	Mean	Gen0	Gen1	Allocated
Aws	2.173 s	25000.0000	8000.0000	207 341.8 KB
Minio	1.365 s	-	-	279 989.3 KB

Проблема: OutOfMemory

- Средний файл 123 МБ
- Загружаем
- Выгружаем
- Удаляем

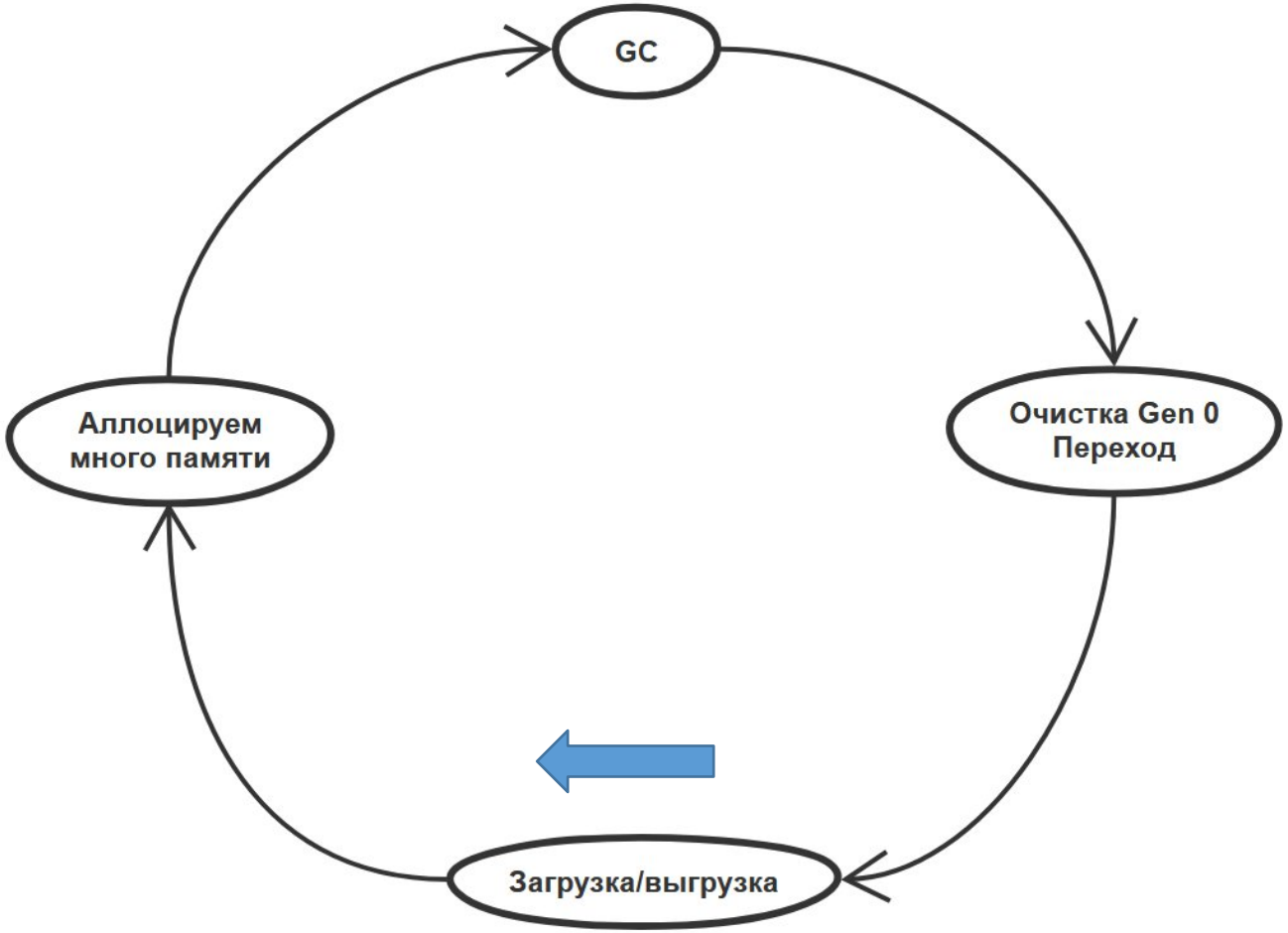
```
BenchmarkDotNet = v0.13.5, OS=Windows 11 (10.0.22621.1265/22H2/2022Update/SunValley2)
AMD Ryzen 7 5800H with Radeon Graphics, 1 CPU, 16 logical and 8 physical cores
.NET SDK = 7.0.102
           [Host]   : .NET 7.0.2 (7.0.222.60605), X64 RyuJIT AVX2 DEBUG
           .NET 7.0 : .NET 7.0.2 (7.0.222.60605), X64 RyuJIT AVX2

Job = .NET 7.0  Runtime=.NET 7.0
```

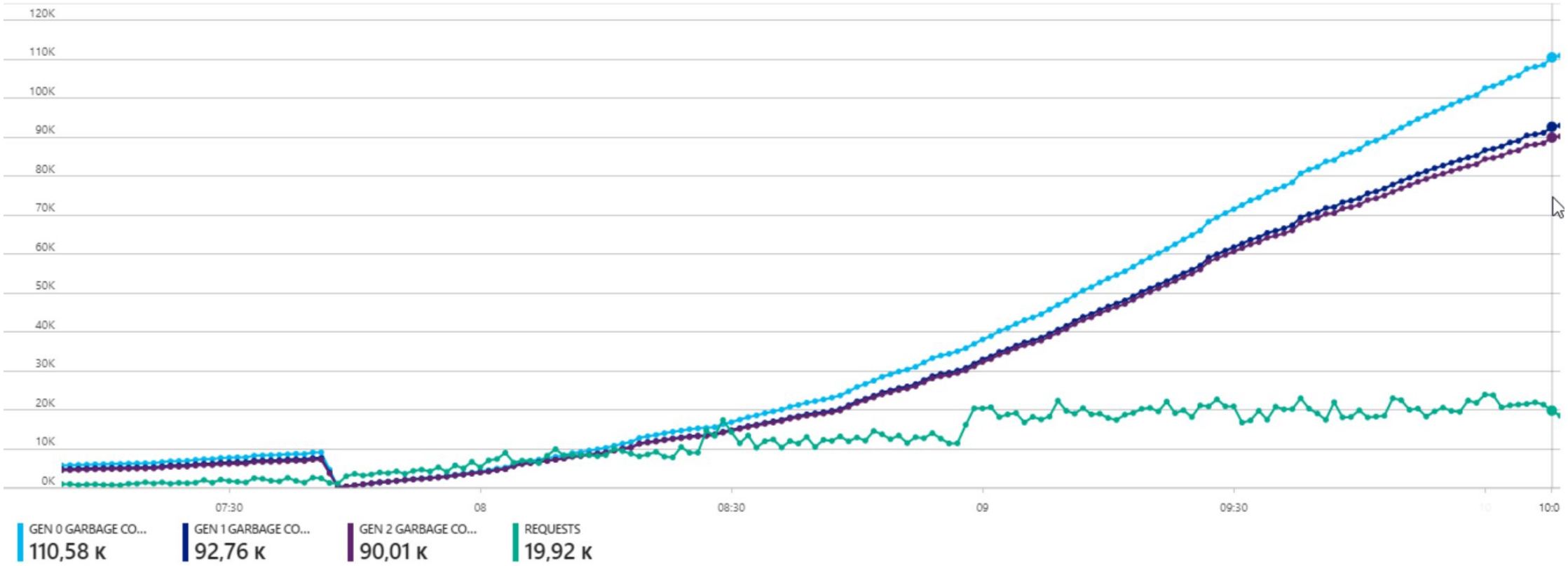
Method	Mean	Gen0	Gen1	Allocated
Aws	2.173 s	25000.0000	8000.0000	207 341.8 KB
Minio	1.365 s	-	-	279 989.3 KB

Напоминание про память и процессор

- Тратим много памяти
- GC собирает
- Или перемещает
- Или сжимает

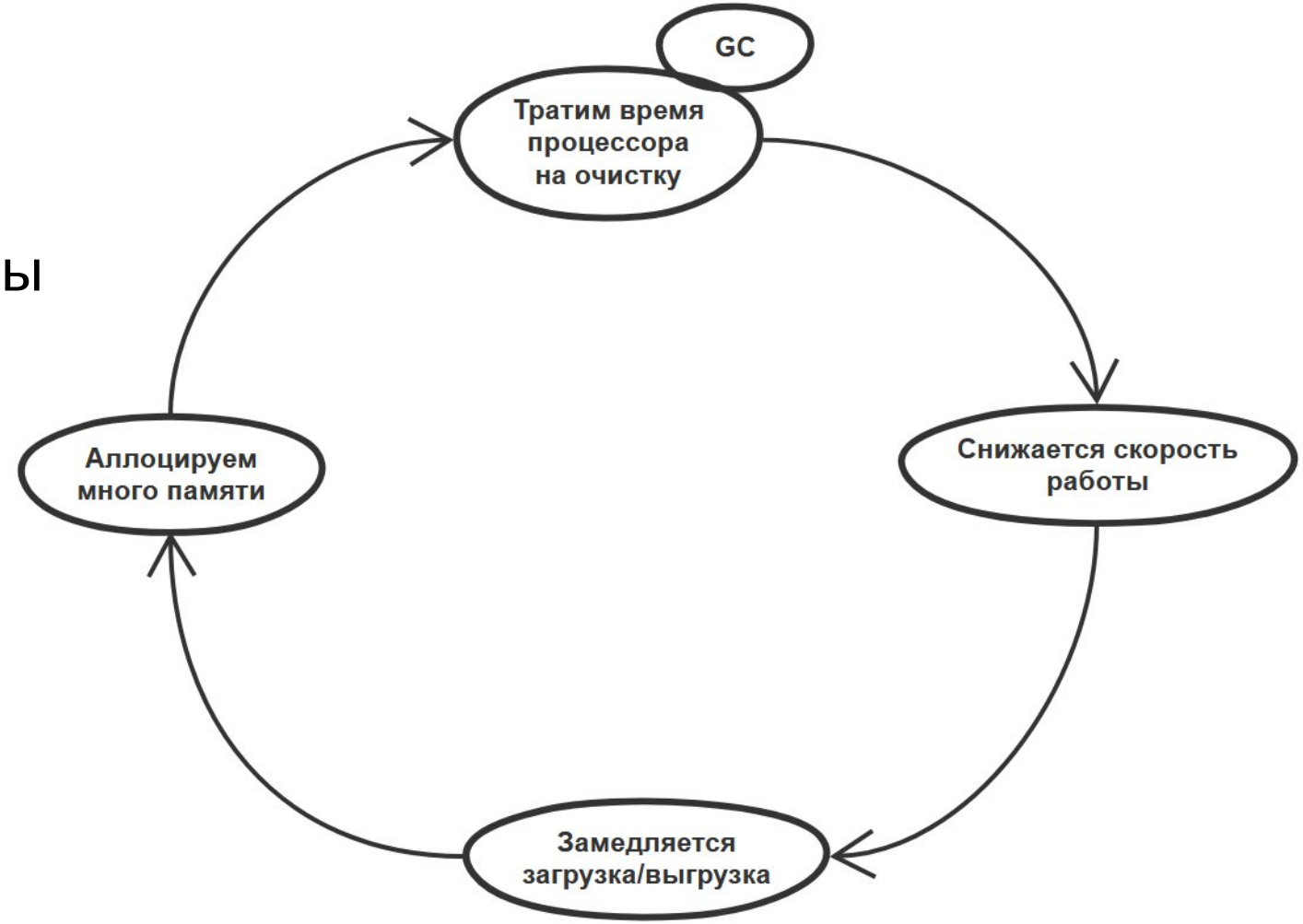


Напоминание про память и процессор



Напоминание про память и процессор

- Тратим много памяти
- Работает процессор
- Снижается скорость работы
- Очистка замедляется



Как же быть?

Сделать велосипед!

Как же быть?

Сделать велосипед!

Method	Mean	Ratio	Gen0	Gen1	Allocated	Alloc Ratio
Aws	2.173 s	1.73	25000.0000	8000.0000	207 341.8 KB	252.99
Minio	1.365 s	1.08	-	-	279 989.3 KB	341.64
Storage	1.282 s	1.00	-	-	819.5 KB	1.00

Клиент для S3

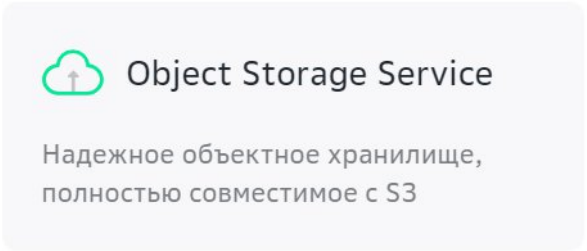
1. Понять, что такое S3
2. Изучить код клиента Minio
3. Написать свой клиент для S3
4. Написать свои вспомогательные классы
5. ???
6. PROFIT?

Клиент для S3

1. Понять, что такое S3
2. Изучить код клиента Minio
3. Написать свой клиент для S3
4. Написать свои вспомогательные классы
5. ???
6. PROFIT?

S3: что это такое?

- **Протокол** HTTP взаимодействия
- Но с особенностями (XML)
- Реализация S3-хранилища может быть любой
- Есть во всех облаках



S3: читаем документацию

- Основан на HTTP
- Обычные GET/POST/PUT
- Хорошее описание у Yandex

Общий вид запроса к API

Общий вид запроса к Object Storage API:

```
{GET|HEAD|PUT|DELETE} /<имя_бакета>/<ключ_объекта> HTTP/2
Host: storage.yandexcloud.net
Content-Length: length
Date: date
Authorization: authorization string (AWS Signature Version 4)

Request_body
```

Запрос содержит HTTP-метод, имя бакета и ключ объекта.

Запрос

```
PUT /{bucket}/{key} HTTP/2
```

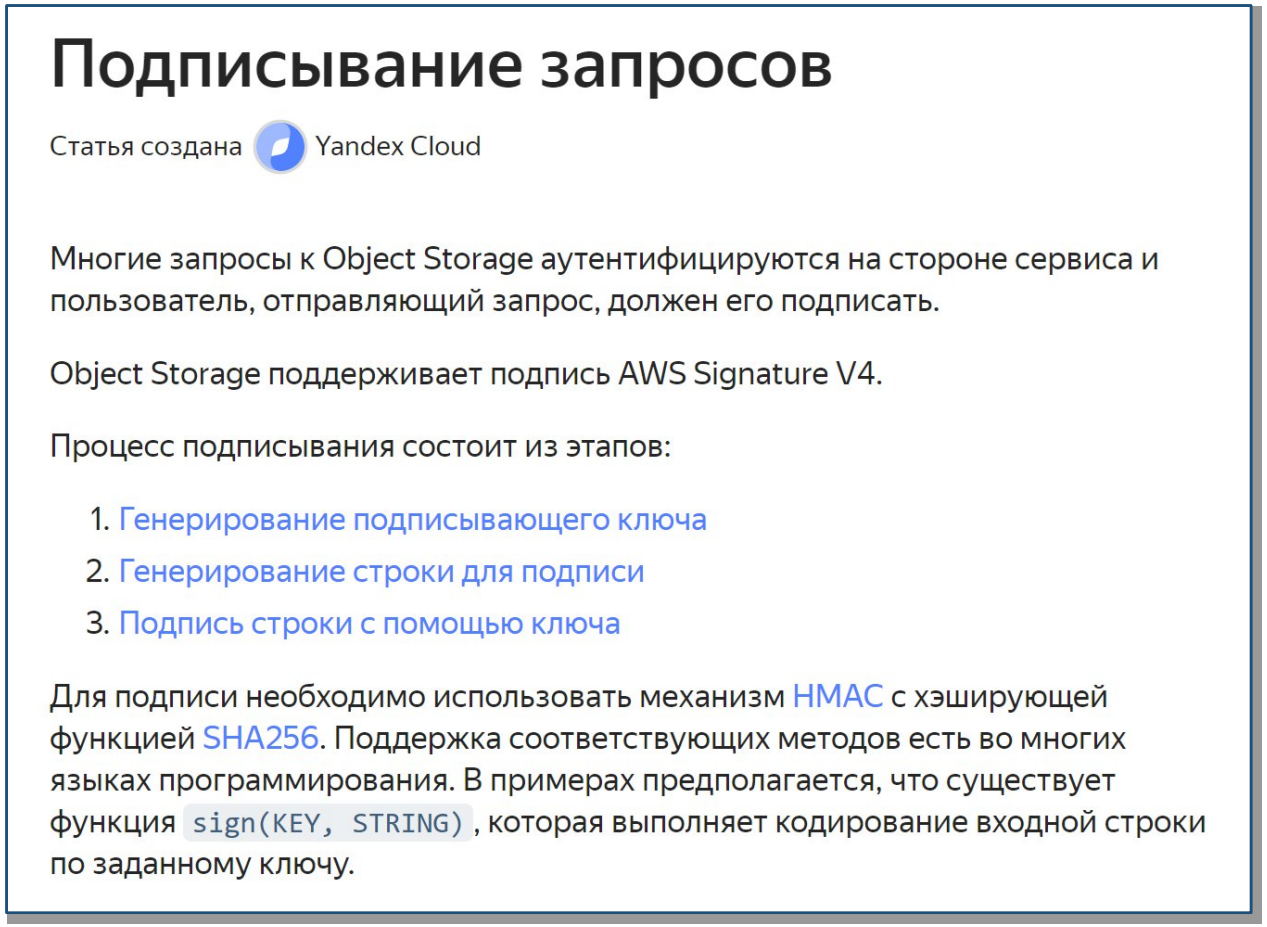
Path параметры

Параметр	Описание
`bucket`	Имя бакета.
`key`	Ключ объекта. Идентификатор, под которым объект будет сохранен в Object Storage.

S3: это просто контракт

То есть мы **действительно** можем сделать:

- Обёртку над HttpClient
- С подписыванием
- И новым .NET 7



Подписывание запросов

Статья создана Yandex Cloud

Многие запросы к Object Storage аутентифицируются на стороне сервиса и пользователь, отправляющий запрос, должен его подписать.

Object Storage поддерживает подпись AWS Signature V4.

Процесс подписывания состоит из этапов:

1. Генерирование подписывающего ключа
2. Генерирование строки для подписи
3. Подпись строки с помощью ключа

Для подписи необходимо использовать механизм HMAC с хэширующей функцией SHA256. Поддержка соответствующих методов есть во многих языках программирования. В примерах предполагается, что существует функция `sign(KEY, STRING)`, которая выполняет кодирование входной строки по заданному ключу.

Клиент для S3

1. ~~Понять, что такое S3~~
2. Изучить код клиента Minio
3. Написать свой клиент для S3
4. Написать свои вспомогательные классы
5. ???
6. PROFIT?

Minio: код клиента Minio

- Код открыт: https://github.com/minio/minio-dotnet

Minio: сохраняем файл

- Разберём сохранение файла в клиенте Minio
- Это показательно для демонстрации расхода памяти

Minio: сохраняем файл

- На каждый запрос создаётся объект PutObjectArgs

```
await _minioClient.PutObjectAsync(args: new PutObjectArgs()
    .WithBucket(_bucket)
    .WithObject(_fileId)
    .WithObjectSize(_inputData.Length)
    .WithStreamData(_inputData) // PutObjectArgs
    .WithContentType("application/pdf"), _cancellation); // Task
```

Minio: сохраняем файл

- На каждый запрос создаётся объект PutObjectArgs
- И массив byte[], куда перекладывается часть файла

```
internal async Task<byte[]> ReadFullAsync(Stream data, int currentPartSize)
{
    var result = new byte[currentPartSize];
    var totalRead = 0;
```

Чтобы по его содержимому вычислить hash, который нужен для подписи

Minio: сохраняем файл

- На каждый запрос создаётся объект PutObjectArgs
- И массив byte[], куда перекладывается часть файла
- И тяжелый RequestMessageBuilder

```
private async Task<string> PutObjectSinglePartAsync(PutObjectArgs args,   IL code
    CancellationToken cancellationToken = default)
{
    //Skipping validate as we need the case where stream sends 0 bytes
    var requestMessageBuilder = await CreateRequest(args).ConfigureAwait(false);
```

Minio: сохраняем файл

- На каждый запрос создаётся объект PutObjectArgs
- И массив byte[], куда перекладывается часть файла
- И тяжелый RequestMessageBuilder
- И ещё много много строк

```
if (!usePathStyle)
{
    var suffix:string = bucketName ≠ null ? bucketName + "/" : "";
    host = host + "/" + suffix;
}

var scheme:string = secure ? "https" : "http";
var endpointURL:string = string.Format("{0}://{1}", scheme, host);
var uri = new Uri(endpointURL, UriKind.Absolute);
```

```
var query:NameValueCollection = HttpUtility.ParseQueryString(requestUriBuilder.Query);
requestUriBuilder.Query = query.ToString();
```

```
if (objectName ≠ null) resource += utils.EncodePath(objectName

// Append query string passed in
if (resourcePath ≠ null) resource += resourcePath;
```

C# HttpRequestMessageBuilder.cs

```
63
64      public HttpRequestMessage Request  IL code
65      {
66          get
67          {
68              var requestUriBuilder = new UriBuilder(RequestUri);
69
70              foreach (var queryParameter:KeyValuePair<string,string> in QueryParameters)
71              {
72                  var query:NameValueCollection = HttpUtility.ParseQueryString(requestUriBuilder.Query);
73                  requestUriBuilder.Query = query.ToString();
74              }
75
76              var requestUri = requestUriBuilder.Uri;
77              var request = new HttpRequestMessage(Method, requestUri);
78
79              if (Content != null) request.Content = new ByteArrayContent(Content);
80
```

Minio: большие накладные расходы

- Средний файл 123 МБ
- Загружаем
- Выгружаем
- Удаляем

```
BenchmarkDotNet = v0.13.5, OS=Windows 11 (10.0.22621.1265/22H2/2022Update/SunValley2)
AMD Ryzen 7 5800H with Radeon Graphics, 1 CPU, 16 logical and 8 physical cores
.NET SDK = 7.0.102
           [Host]   : .NET 7.0.2 (7.0.222.60605), X64 RyuJIT AVX2 DEBUG
           .NET 7.0 : .NET 7.0.2 (7.0.222.60605), X64 RyuJIT AVX2

Job = .NET 7.0  Runtime=.NET 7.0
```

Method	Mean	Gen0	Gen1	Allocated
Aws	2.173 s	25000.0000	8000.0000	207 341.8 KB
Minio	1.365 s	-	-	279 989.3 KB

Клиент для S3

1. ~~Понять, что такое S3~~
2. ~~Изучить код клиента Minio~~
3. Написать свой клиент для S3
4. Написать свои вспомогательные классы
5. ???
6. PROFIT?

Свой клиент: подходы

- Собираем все строки заранее
- Избегаем лишних аллокаций (zero-allocation)
- Использует ValueStringBuilder
- ArrayPool для промежуточного хранения

Своя клиент: строки-строки-строки

- URL можно собрать заранее

```
_bucket = $"{scheme}://{settings.EndPoint}{port}/{bucket}";
_endpoint = $"{settings.EndPoint}{port}";
```

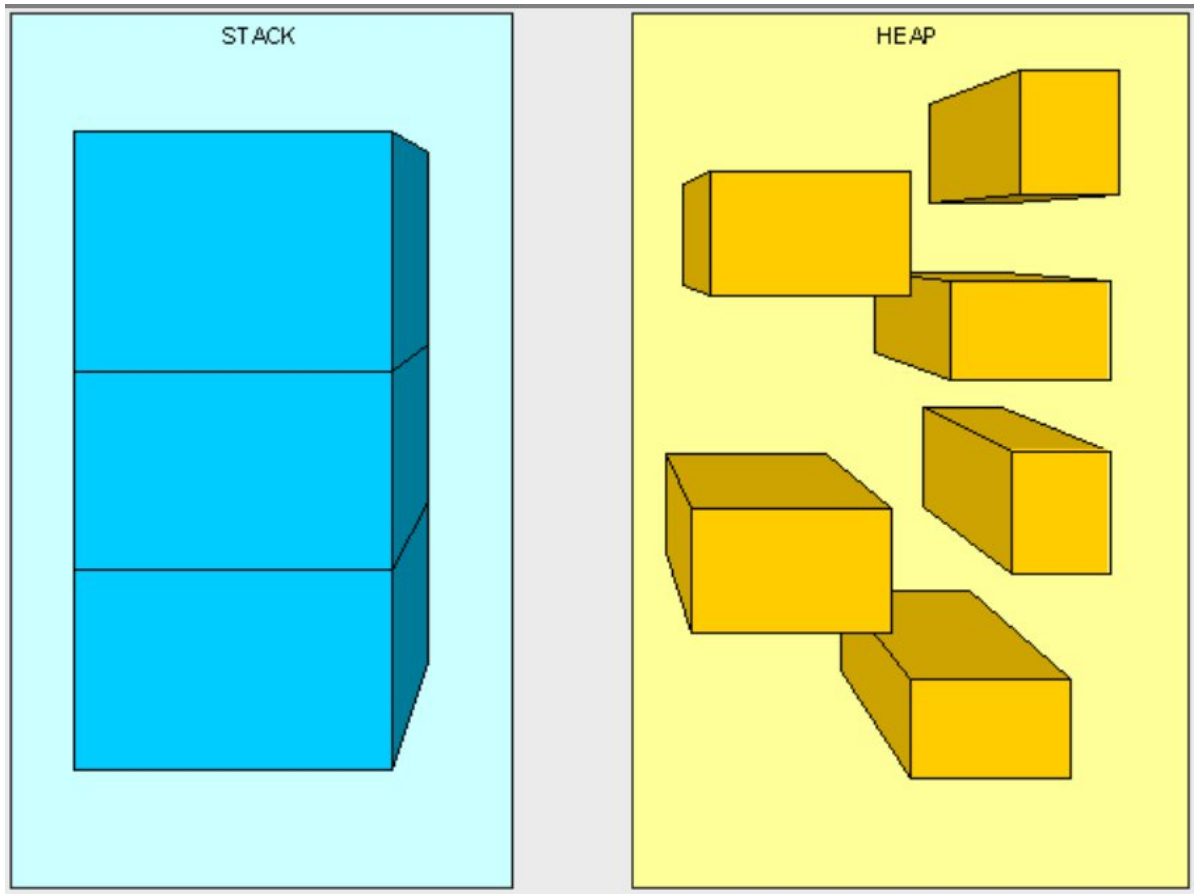
Своя клиент: строки-строки-строки

- URL можно собрать заранее
- Как и подпись (нужна для работы с файлами)

```
public Signature(string secretKey, string region, string service)
{
    _region = region;
    _secretKey = Encoding.UTF8.GetBytes(s:$"AWS4{secretKey}");
    _scope = $"/{region}/{service}/aws4_request\n";
    _service = service;
}
```

Своя клиент: аллокация на стеке

- Память состоит из стека и кучи
- Стек быстрый
- Куча медленная



Свой клиент: аллокация на стеке

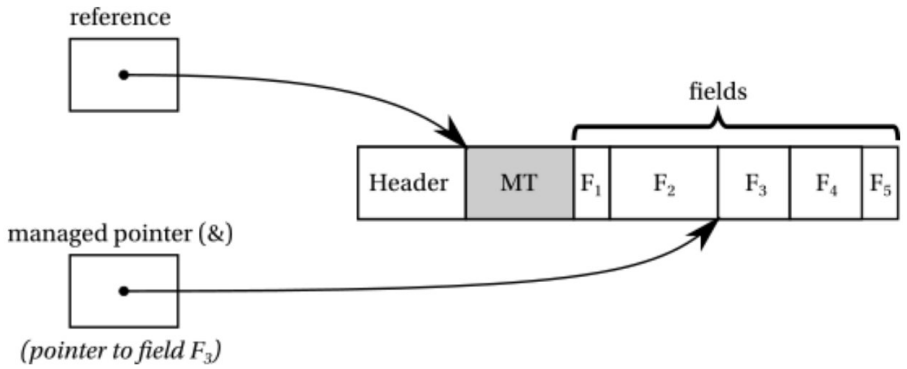
- Память состоит из стека и кучи
- Стек быстрый (очистка по длине)
- Куча медленная (граф, GC)
- Stackalloc - тот самый zero-allocation

```
Span<byte> buffer = stackalloc byte[64];
```

Свой клиент: аллокация на стеке

- Span это указатель на область памяти
- Ref struct

```
public ref struct Span<T>
{
   internal ref T _pointer;
   private int _length;
   ...
}
```



Свой клиент: аллокация на стеке

- Span это указатель на область памяти
- Ref struct

```
var array = new int[64];
Span<int> span1 = new Span<int>(array);
Span<int> span2 = new Span<int>(array, start: 8, length: 4);
```

```
Span<int> span3 = stackalloc[] { 1, 2, 3, 4, 5 };
```

```
IntPtr memory = Marshal.AllocHGlobal(64);
void* ptr = memory.ToPointer();
Span<byte> span4 = new Span<byte>(ptr, 64);
```

```
string str = "Hello world";
ReadOnlySpan<char> span5 = str.AsSpan();
```

Свой клиент: аллокация на стеке

- Span это указатель на область памяти
- Ref struct
- Соревнуемся с List

Method	Runtime	Mean	Error	StdDev	Ratio	Gen 0	Allocated
ListWithCapacity	.NET 6.0	1,601.2 ns	11.79 ns	11.03 ns	1.00	0.1335	1,120 B
ListWithoutCapacity	.NET 6.0	1,675.1 ns	6.06 ns	5.37 ns	1.05	0.2708	2,272 B
StackAlloc	.NET 6.0	275.7 ns	2.61 ns	2.32 ns	0.17	-	-

Свой клиент: аллокация на стеке

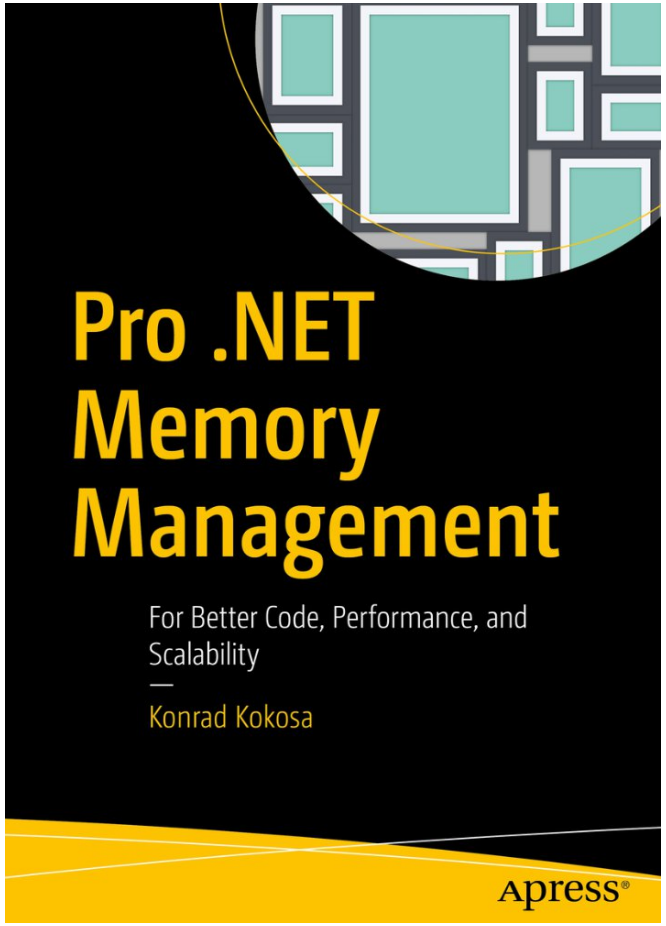
- Span это указатель на область памяти
- Ref struct
- Соревнуемся с List

Method	Runtime	Mean	Error	Gen 0	Allocated
ListWithCapacity	.NET 6.0	1,601.2 ns	11.79 ns	0.1335	1,120 B
ListWithoutCapacity	.NET 6.0	1,675.1 ns	6.06 ns	0.2708	2,272 B
StackAlloc	.NET 6.0	275.7 ns	2.61 ns	-	-

Свой клиент: аллокация на стеке

- Span это указатель на область памяти
- Ref struct

- Конрад Кокоса



Свой клиент: TryFormat

- TryFormat делает «ToString» в Span
- Есть почти у всех примитивных типов

```
Span<char> numberString = stackalloc char[10];
var pos:int = _length;
if (number.TryFormat(numberString, out var written:int))
{
    var writtenChars:Span<char> = numberString[..written];
    writtenChars.CopyTo(_otherSpan[pos..]);
}
else Errors.CantFormatToString(number);
```

Свой клиент: TryFormat

- TryFormat делает «ToString» в Span
- Надо угадать с размером (InvariantCulture)

```
Span<char> numberString = stackalloc char[10];
var pos:int = _length;
if (number.TryFormat(numberString, out var written:int, format: default, _culture))
{
    var writtenChars:Span<char> = numberString[..written];
    writtenChars.CopyTo(_otherSpan[pos..]);
}
else Errors.CantFormatToString(number);
```

Свой клиент: TryFormat

- TryFormat делает «ToString» в Span
- Тот самый zero-allocation

```
public void Append(DateTime value, string format)  ◇  5 usages
{
    Span<char> buffer = stackalloc char[16];
    var pos:int = _length;
    if (value.TryFormat(buffer, out var written:int, format))
    {
        if (pos > _buffer.Length - written)+2 Grow(written);
        buffer.CopyTo(_buffer[pos..]);

        _length = pos + written;
    }
    else+1 Errors.CantFormatToString(value);
}
```

```
public void Append(int value)  ◇  1 usage  kirly +1
{
    Span<char> buffer = stackalloc char[10];
    var pos:int = _length;
    if (value.TryFormat(buffer, out var written:int ))+1
    {
        if (pos > _buffer.Length - written)+2 Grow(written);
        buffer.CopyTo(_buffer[pos..]);

        _length = pos + written;
    }
    else+1 Errors.CantFormatToString(value);
}
```

Свой клиент: ValueStringBuilder

- Из «недр» .NET
- Задаём первоначальный размер через stackalloc
- Ref struct – new ничего не стоит
- Обёртка над Span<char>

Свой клиент: ValueStringBuilder

```
internal ref struct ValueStringBuilder  18 usages
{
    private Span<char> _buffer;
    private int _length;
    private char[]? _array;

    public ValueStringBuilder(Span<char> buffer)
    {
        _array = null;
        _buffer = buffer;
        _length = 0;
    }
```

Свой клиент: ValueStringBuilder

- Это просто обёртка над Span<char>

```
var builder = new ValueStringBuilder(stackalloc char[512]);

builder.Append(_headerStart); // TryFormat
builder.Append(now, format: Signature.Iso8601Date); // TryFormat
builder.Append(_headerEnd);   // TryFormat
builder.Append(signature);    // TryFormat
```

Свой клиент: ValueStringBuilder

- Это просто обёртка над Span<char>
- Для расширения используется ArrayPool

```
var poolArray:char[] = ArrayPool<char>.Shared.Rent(newCapacity);

_buffer[.._length].CopyTo(poolArray);

var toReturn:char[]? = _array;
_buffer = _array = poolArray;
if (toReturn ≠ null)+1
{
    ArrayPool<char>.Shared.Return(toReturn);
}
```

Свой клиент: ValueStringBuilder

- Это просто обёртка над Span<char>
- Для расширения используется ArrayPool
- Его можно передавать в методы

```
var builder = new ValueStringBuilder(stackalloc char[512]);

AppendStringToSign(ref builder, requestDate);
AppendCanonicalRequestHash(ref builder, request, signedHeaders, payloadHash);
```

Свой клиент: ValueStringBuilder

- Это просто обёртка над Span<char>
- Для расширения используется ArrayPool
- Его можно передавать в методы
- Преобразовывать в ReadOnlySpan

```
var builder = new ValueStringBuilder(stackalloc char[512]);

AppendStringToSign(ref builder, requestDate);
AppendCanonicalRequestHash(ref builder, request, signedHeaders, payloadHash);

ReadOnlySpan<char> internalBuffer = builder.AsReadonlySpan();

builder.Dispose();
```

Свой клиент: ValueStringBuilder

- Сейчас много где «принимают» ReadOnlySpan.

```
private static string Sha256ToHex(ReadOnlySpan<char> value)  ◇  2 usages  Kirill Bazhay
{
    var count:int = Encoding.UTF8.GetByteCount(value);

    var pool = ArrayPool<byte>.Shared;
    var byteBuffer:byte[] = pool.Rent(count);

    var encoded:int = Encoding.UTF8.GetBytes(value, byteBuffer);
    Span<byte> hashBuffer = stackalloc byte[64];
    var source:Span<byte> = byteBuffer.AsSpan(start: 0, length: encoded);
    var result:string = SHA256.TryHashData(source, hashBuffer, out var written:int )
        ? ToHex(hashBuffer[..written])
        : string.Empty;
```

Свой клиент: загружаем файл

- Мы победили строки, пора загружать
- Получаем Stream из IFormFile или из Multipart

Свой клиент: загружаем файл

- Мы победили строки, пора загружать
- Получаем Stream из IFormFile или из Multipart
- И перекладываем содержимое Stream в массив

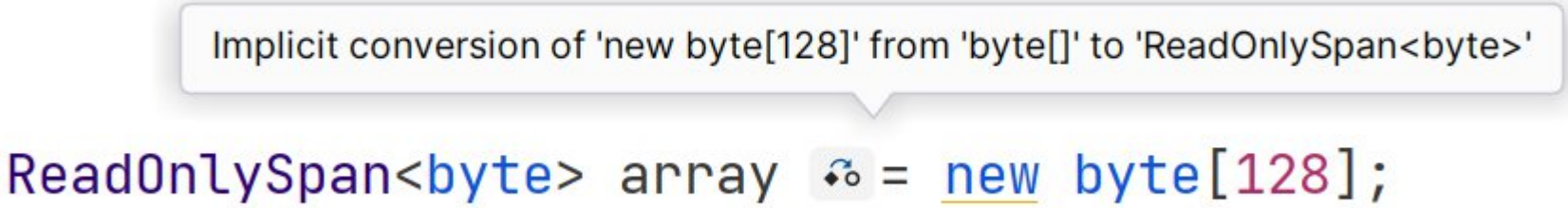
Свой клиент: загружаем файл

- В массив, потому что нужно вычислить хэш содержимого
- Чтобы подписать запрос в S3 (особенность)

```
public static string GetPayloadHash(ReadOnlySpan<byte> data) ◇ ↗5
{
    Span<byte> hashBuffer = stackalloc byte[32];
    return SHA256.TryHashData(data, hashBuffer, out var written:int)
        ? ToHex(hashBuffer[..written])
        : string.Empty;
}
```

Свой клиент: загружаем файл

- В массив, потому что нужно вычислить хэш содержимого
- Чтобы подписать запрос в S3 (особенность)
- Массив это тоже ReadOnlySpan



Свой клиент: ArrayPool

- Пулинг – эффективный способ оптимизации
- Для работы с массивами есть ArrayPool

```
var pool = ArrayPool<byte>.Shared;
var byteBuffer:byte[] = pool.Rent(length);
```

Свой клиент: ArrayPool

- Пуллинг – эффективный способ оптимизации
- Для работы с массивами есть ArrayPool
- После использования не забываем вернуть

```
var pool = ArrayPool<byte>.Shared;
var byteBuffer:byte[] = pool.Rent(length);

// ...

pool.Return(byteBuffer);
```

Свой клиент: ArrayPool

- Для работы с массивами есть ArrayPool
- После использования не забываем вернуть
- ArrayPool подходит для копирования byte[] из Stream

```
private async Task<bool> PutFile(  ◇  1 usage  kirly +1 *
    string fileName, string contentType, Stream data,
    CancellationToken ct)
{
    var pool = ArrayPool<byte>.Shared;

    var byteBuffer:byte[] = pool.Rent((int) data.Length);
    var dataSize:int = await data.ReadAsync( byteBuffer, ct).ConfigureAwait(false);
```

Свой клиент: ArrayPool

- ArrayPool подходит для копирования byte[] из Stream
- Нам очень нужно посчитать hash

```
var buffer:byte[] = bufferPool.Rent((int) data.Length);
var dataSize:int = await data.ReadAsync( buffer, ct).ConfigureAwait(false);

var payloadHash:string = GetPayloadHash( data: buffer.AsSpan( start: 0, length: dataSize));

HttpResponseMessage response;
using (var request = CreateRequest(HttpMethod.Put, fileName))
{
    using (var content = new ByteArrayContent(buffer, offset: 0, count: dataSize))
```

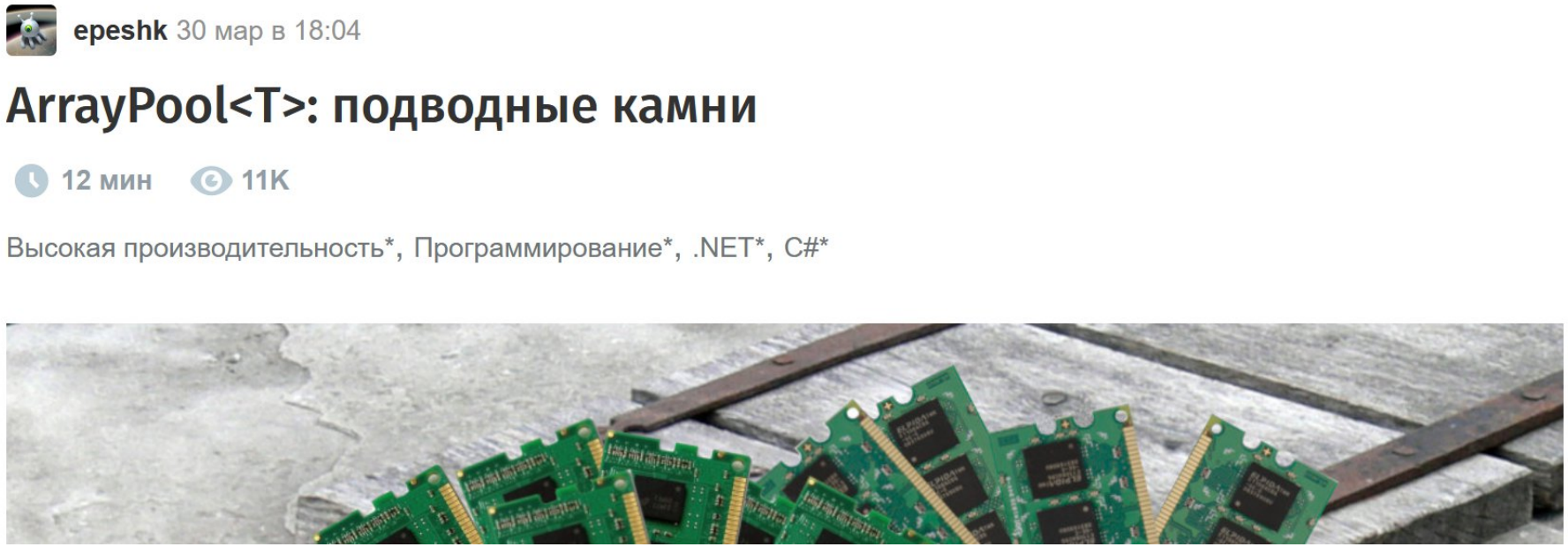
Свой клиент: ArrayPool

- ArrayPool подходит для копирования byte[] из Stream
- Нам очень нужно посчитать hash
- Но лучше использовать StreamContent (не с S3)

```
private async Task<bool> PutFile(  ◇  1 usage  kirly +1 *
    string fileName, string contentType, Stream data,
    CancellationToken ct)
{
    HttpResponseMessage response;
    using (var request = CreateRequest(HttpMethod.Put, fileName))
    {
        using (var content = new StreamContent(data,  bufferSize: 2048))
```

Свой клиент: ArrayPool

- ArrayPool подходит для копирования byte[] из Stream
- Нам очень нужно посчитать hash
- Есть особенности ArrayPool (статья Евгения Пешкова)



Свой клиент: возвращать нужное

- Получение данных от S3 также можно оптимизировать
- В большинстве случаев у нас есть все данные

Свой клиент: возвращать нужное

- Самый частый случай – перекладывание в Stream

```
Stream outputData = new MemoryStream();
await _minioClient.GetObjectAsync(args: new GetObjectArgs()
        .WithBucket(_bucket)
        .WithObject(_fileId) // GetObjectArgs
        .WithCallbackStream((file :Stream , ct :CancellationToken ) ⇒
        {
            return file.CopyToAsync(outputData, ct);
        }),
    _cancellation); // Task<ObjectStat>
```

Свой клиент: возвращать нужное

- Самый частый случай – перекладывание в Stream
- Но ведь у нас уже есть Stream

```
HttpResponseMessage response;
using (var request = CreateRequest(HttpMethod.Get, fileName))
{
    response = await Send(request, EmptyPayloadHash, cancellation).ConfigureAwait(false);
}

switch (response.StatusCode)+1
{
    case HttpStatusCode.OK:
        return await response.Content.ReadAsStreamAsync(cancellation).ConfigureAwait(false);
```

Свой клиент: возвращать нужное

- ContentType предлагают брать отдельным запросом

```
ObjectStat stats = await minioClient.StatObjectAsync(statObjectArgs, cancellation);
return stats.ContentType;
```

Свой клиент: возвращать нужное

- ContentType предлагают брать отдельным запросом
- Но и он у нас есть в ответе S3

```
public string? ContentType ⇒ _response.Content.Headers.ContentType?.MediaType;

private readonly HttpResponseMessage _response;

internal S3File(HttpResponseMessage response)  ◇  3 usages  kirly
{
    _response = response;
}
```

Клиент для S3

1. ~~Понять, что такое S3~~
2. ~~Изучить код клиента Minio~~
3. ~~Написать свой клиент для S3~~
4. Написать свои вспомогательные классы
5. ???
6. PROFIT?

Вспомогательные классы: File

- Скрывает сложность
- Отложенное чтение данных из HttpResponseMessage

```
internal S3File(HttpResponseMessage response){...}

public async Task<Stream> GetStream(CancellationToken cancellation)  ◇  6 usages  Kirill Bazhaykin +1
{
    if (!_response.IsSuccessStatusCode)+1 return Stream.Null;

    var stream = await _response.Content.ReadAsStreamAsync(cancellation).ConfigureAwait(false);
    return new S3Stream(_response, stream);
}
```

Вспомогательные классы: Stream

- Дружественный для ASP.NET
- Мы можем управлять моментом Dispose

```
public S3Stream(HttpResponseMessage response, Stream stream)
{
    _response = response;
    _stream = stream;
}

protected override void Dispose(bool disposing)  ◇  Kirill Bazha
{
    _stream.Dispose();
    _response.Dispose();
}
```

Вспомогательные классы: XmlReader

- S3 иногда посылает XML, где хранятся важные данные
- Но нам нужно только значение одного поля

```
internal static class XmlStreamReader  2 usages  Kirill Bazhaykin *
{
    public static string ReadString(Stream stream, ReadOnlySpan<char> elementName, int valueBufferLength)
    {
        Span<char> buffer = stackalloc char[valueBufferLength];

        var written:int = ReadTo(stream, elementName, ref buffer);
        return written == -1
            ? string.Empty
            : buffer[..written].ToString();
    }

    private static int ReadTo(Stream stream, ReadOnlySpan<char> elementName, ref Span<char> valueBuffer)
```

Клиент для S3

1. ~~Понять, что такое S3~~
2. ~~Изучить код клиента Minio~~
3. ~~Написать свою обёртку над HttpClient~~
4. ~~Написать свои вспомогательные классы~~
5. ???
6. PROFIT?

Ура!

```
BenchmarkDotNet = v0.13.5, OS=Windows 11 (10.0.22621.1265/22H2/2022Update/SunValley2)
AMD Ryzen 7 5800H with Radeon Graphics, 1 CPU, 16 logical and 8 physical cores
.NET SDK = 7.0.102
           [Host]   : .NET 7.0.2 (7.0.222.60605), X64 RyuJIT AVX2 DEBUG
           .NET 7.0 : .NET 7.0.2 (7.0.222.60605), X64 RyuJIT AVX2

Job = .NET 7.0  Runtime=.NET 7.0
```

Method	Mean	Ratio	Gen0	Gen1	Allocated	Alloc Ratio
Aws	2.173 s	1.73	25000.0000	8000.0000	207 341.8 KB	252.99
Minio	1.365 s	1.08	-	-	279 989.3 KB	341.64
Storage	1.282 s	1.00	-	-	819.5 KB	1.00

Ура?

```
BenchmarkDotNet = v0.13.5, OS=Windows 11 (10.0.22621.1265/22H2/2022Update/SunValley2)
AMD Ryzen 7 5800H with Radeon Graphics, 1 CPU, 16 logical and 8 physical cores
.NET SDK = 7.0.102
           [Host]   : .NET 7.0.2 (7.0.222.60605), X64 RyuJIT AVX2 DEBUG
           .NET 7.0 : .NET 7.0.2 (7.0.222.60605), X64 RyuJIT AVX2

Job = .NET 7.0  Runtime=.NET 7.0
```

Method	Mean	Ratio	Gen0	Gen1	Allocated	Alloc Ratio
Aws	2.173 s	1.73	25000.0000	8000.0000	207 341.8 KB	252.99
Minio	1.365 s	1.08	-	-	279 989.3 KB	341.64
Storage	1.282 s	1.00	-	-	819.5 KB	1.00

Неожиданно: проблемы на Linux

- Скачивание на Linux заметно дороже
- На Windows меньше 1 МБ, на Debian - 3 МБ

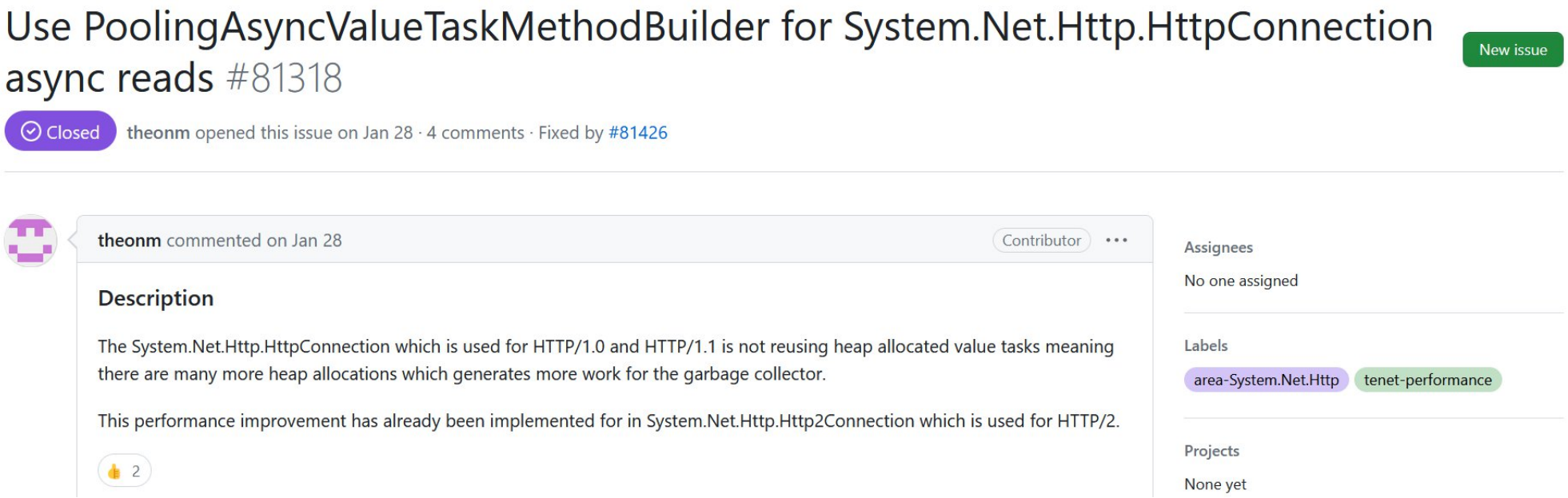
```
BenchmarkDotNet=v0.13.5, OS=debian 11 (container)
AMD Ryzen 7 5800H with Radeon Graphics, 1 CPU, 16 logical and 8 physical cores
.NET SDK=7.0.401
  [Host]   : .NET 7.0.11 (7.0.1123.42427), X64 RyuJIT AVX2
  .NET 7.0 : .NET 7.0.11 (7.0.1123.42427), X64 RyuJIT AVX2

Job=.NET 7.0  Runtime=.NET 7.0
```

Method	Mean	Error	StdDev	Ratio	RatioSD	Gen0	Gen1	Allocated	Alloc Ratio
Aws	5.021 s	0.0905 s	0.0969 s	0.99	0.03	25000.0000	8000.0000	200 MB	85.52
Minio	5.622 s	0.1103 s	0.1932 s	1.09	0.03	-	-	274.96 MB	117.58
Storage	5.099 s	0.1004 s	0.1341 s	1.00	0.00	-	-	2.34 MB	1.00

Неожиданно: проблемы на Linux

- Скачивание на Linux заметно дороже
- На Windows меньше 1 МБ, на Debian - 3 МБ



Docker: бенчмарки в Linux

- Скачивание на Linux заметно дороже
- На Windows меньше 1 МБ, на Debian - 3 МБ
- Запускайте бенчмарки в Docker

Неожиданно: проблемы на Linux

- Запускайте бенчмарки в Docker
- Пишем Dockerfile

```
FROM mcr.microsoft.com/dotnet/sdk:7.0
WORKDIR /src
COPY . .
RUN dotnet restore "./src/Storage.Benchmark/Storage.Benchmark.csproj"  \
    && dotnet publish "./src/Storage.Benchmark/Storage.Benchmark.csproj" \
    -c Release -o "./src/publish"
ENTRYPOINT ["dotnet", "./src/publish/Storage.Benchmark.dll"]
```

Неожиданно: проблемы на Linux

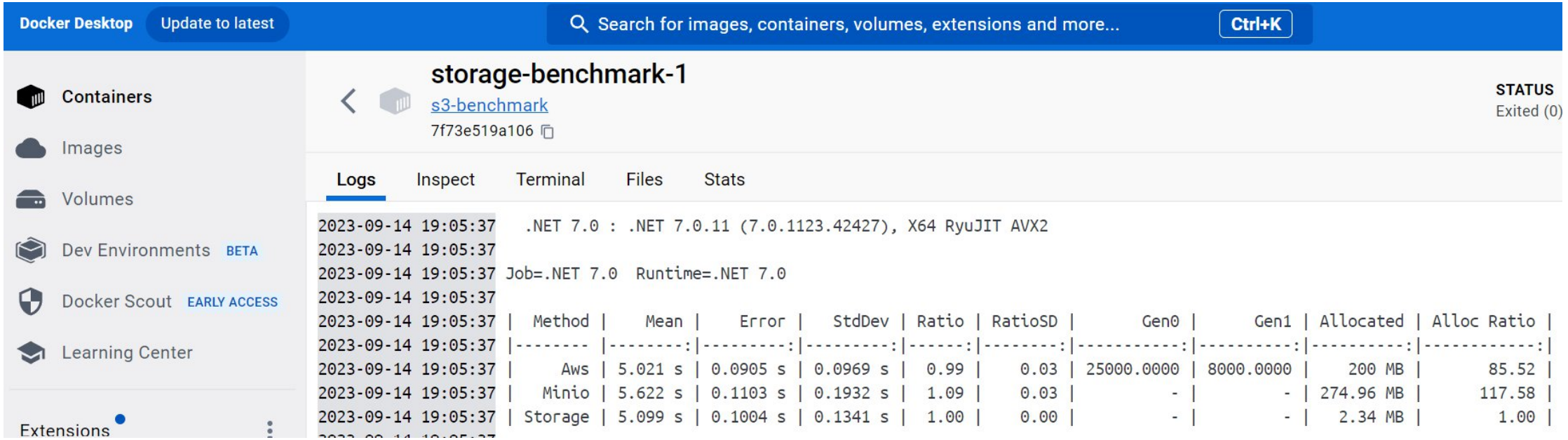
- Запускайте бенчмарки в Docker
- Пишем Dockerfile
- И docker-compose

```
  benchmark:
    image: s3-benchmark
    depends_on:
      - minio
    build:
      context: .
      dockerfile: Dockerfile

volumes:
  minio-data:
```

Неожиданно: проблемы на Linux

- Запускайте бенчмарки в Docker
- Пишем Dockerfile
- И docker-compose



Что мы узнали

- Лишние аллокации могут стать проблемой

Что мы узнали

- Лишние аллокации могут стать проблемой
- При высоких нагрузках – особенно

Что мы узнали

- Лишние аллокации могут стать проблемой
- При высоких нагрузках – особенно
- Строчки лучше собирать через ValueStringBuilder

Что мы узнали

- Лишние аллокации могут стать проблемой
- При высоких нагрузках – особенно
- Строчки лучше собирать через ValueStringBuilder
- Span в современном .NET – обычное дело

Что мы узнали

- Лишние аллокации могут стать проблемой
- При высоких нагрузках – особенно
- Строчки лучше собирать через ValueStringBuilder
- Span в современном .NET – обычное дело
- ArrayPool наш друг, если надо переиспользовать память

Что мы узнали

- Лишние аллокации могут стать проблемой
- При высоких нагрузках – особенно
- Строчки лучше собирать через ValueStringBuilder
- Span в современном .NET – обычное дело
- ArrayPool наш друг, если надо переиспользовать память
- Иногда проще написать велосипед

Что мы узнали

- Лишние аллокации могут стать проблемой
- При высоких нагрузках – особенно
- Строчки лучше собирать через ValueStringBuilder
- Span в современном .NET – обычное дело
- ArrayPool наш друг, если надо переиспользовать память
- Иногда проще написать велосипед
- Реализации .NET под Windows и Linux немного разные

Спасибо!

Код находится вот тут
https://github.com/teoadal/Storage

Можно сканировать QR.