Модульность без микросервисов в ASP.NET из коробки

Стас Выщепан

DotNext 2026

О себе

- Создаю программы с 2007 года, с 2008 на .NET
- Занимаюсь корпоративным софтом и публичным вебом
- Microsoft MVP 2011-2018 (SharePoint, SQL Server, Windows Server)
- С 2018 года пишу на .NET Core, Linux и PostgreSql
- Руковожу командами и занимаюсь архитектурой 15 лет
- С 2021 года все проекты так или иначе связаны с микросервисами

Все хотят распилить монолит

🎯 **Масштабирование:** Ресурсы только под "горячие" зоны

🛡️ **Изоляция сбоев:** Упал модуль — система жива

🧠 **Управление сложностью:** Держим в голове только свой кусок

👥 **Продуктивность:** Параллельная работа без конфликтов

🚀 **Независимый деплой:** Деплой не договариваясь ни с кем

🛠️ **Технологии:** Бери лучший инструмент для каждой задачи

😉 **Тренд:** Микросервисы — это модно

📌 Запомните этот список — **в конце вернёмся к нему**.

Микросервисное бинго

- ✓ Микросервисов больше чем разработчиков
- ✓ Весь код сервисов на одном языке, а базы данных на одном сервере
- ✓ Стандартизированная архитектура - нельзя тащить любые серверы, библиотеки, и паттерны
- ✓ Нельзя запустить всё одной кнопкой (F5) или командой (dotnet run)
- ✓ Деплой не зависит от разработчиков, им занимаются отдельные специалисты *aka* DevOps-инженеры
- ✓ Каскадные отказы – ошибка\таймаут одного сервиса приводит к ошибкам в других

«Налог» на микросервисы

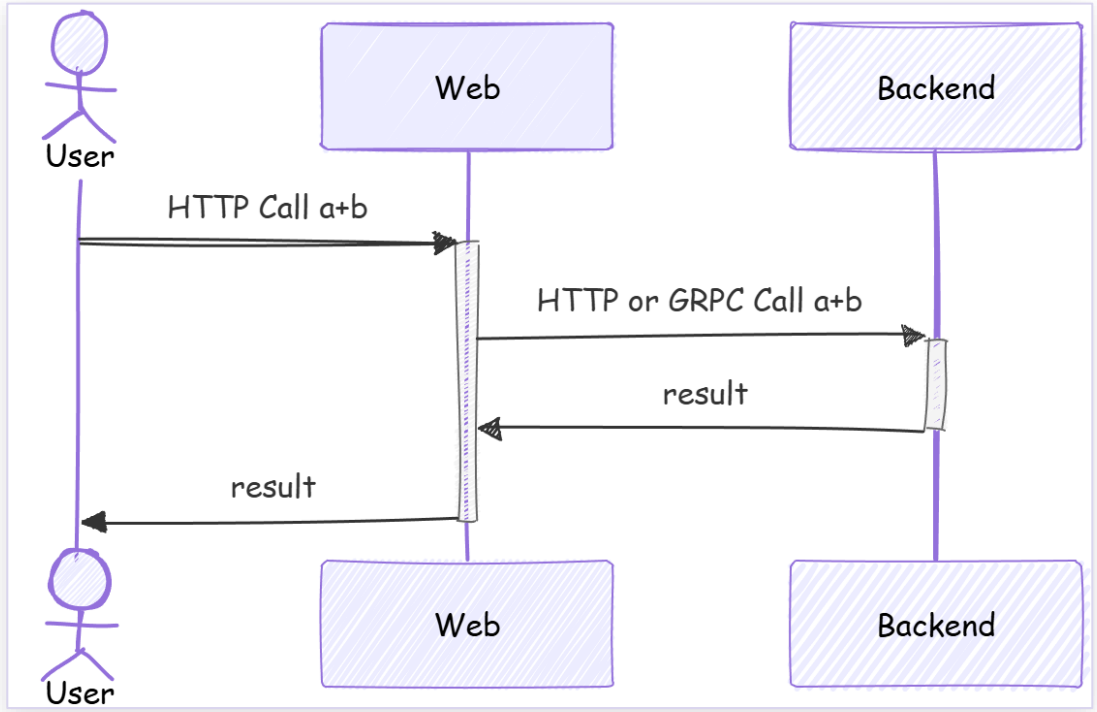
- Согласование изменений
  - Поменяли контракт сервиса – поменяйте контракт клиентов
  - Обновили общую библиотеку – обновляем все сервисы
- Eventual consistency
  - Саги, координаторы, ретраи, компенсирующие транзакции
  - Все равно не получается ACID – данные в сервисах расходятся
- Повышенная сложность
  - Вызов сервиса упал по таймауту – повторять или вернуть ошибку?
  - Нельзя написать JOIN в базе, нужно делать к коде программы
- Операционные расходы
  - Каждый сервис потребляет память даже без нагрузки 50 - 300 МБ на «голое» ASP.NET приложение
  - 25 сервисов по 2 экземпляра каждый – 2,5 - 15 ГБ впустую
- Быстродействие
  - Каждый вызов сервиса это сериализация и обращение по сети
  - Данные дублируются в разных сервисах, повышая нагрузку на диски
  - Ресурсы съедает взаимодействие, а не бизнес-логика

Быстродействие MCA – нагрузочный тест

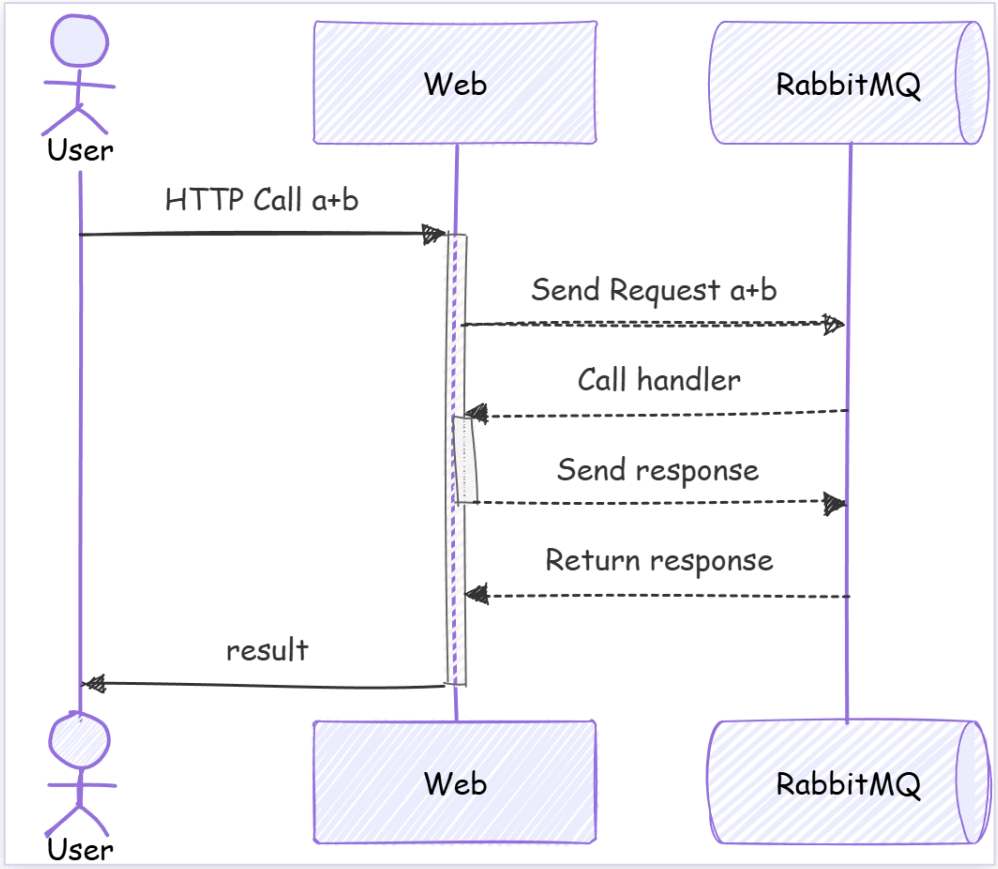
- Сценарий: вызов функции, которая складывает два числа
- Архитектуры:
  - Прямой вызов функции внутри приложения
  - Вызов по **HTTP** без сериализации в JSON
  - Вызов по **GRPC**
  - Вызов через **RabbitMQ** (EasyNetQ Rpc)
- Что измеряем: RPS, CPU и память
- HTTP и GRPC собраны с использованием NativeAOT

Быстродействие MCA – архитектура тестов

Вызовы по HTTP и GRPC



Вызов через RabbitMQ



Тестовый стенд

- **Хост** – 12 CPU, 64гб RAM, M2 SSD, Windows 11, WSL
- Все приложения вместе с rabbit в **одном docker compose**
- Тестовая нагрузка – k6 с хоста, 250 VUs, 30 сек

Результаты забега: полные таблицы

Тест	RPS	CPU % Web	Mem MiB Web	CPU % Back	Mem MiB Back
In process	20 243.36	214.44	69.27	0	0
Remote HTTP	14 397.56	445.40	125.90	124.90	37.93
Remote GRPC	14 819.93	458.88	149.50	96.04	21.00
Remote RabbitMQ	10 649.60	365.36	106.20	300.18	137.4

Тест	Ratio RPS/sec	Ratio CPU	Ratio MEM	(CPU+MEM) / (2 * RPS)
In process	1.00	1.00	1.00	**1.00**
Remote HTTP	0.71	2.66	2.37	**3.54**
Remote GRPC	0.73	2.59	2.46	**3.52**
Remote RabbitMQ	0.53	3.10	3.52	**6.25**

Замеры: k6 v2.2.0 · .NET 10 · коммит b934693ed61960d5b3006910cbb715edbb83ef8c · стенд — на предыдущем слайде

Ресурсы на один запрос относительно вызова в процессе

3,5×

Вы платите в три с половиной раза больше ресурсов за запрос.

Чтобы вызвать функцию, которая складывает два числа.

HTTP
3,54×

gRPC
3,52×

RabbitMQ
6,25×

(CPU+MEM) / (2 * RPS) · In process = 1,00×

Можно ли не платить «налог»?

- Иметь один контейнер или процесс
- Внутри отдельные модули, не знающие об устройстве друг друга
- Использовать синхронные вызовы внутри процесса
- Запускать все одной кнопкой\командой

Модульный монолит!

Модульный монолит

- Проблема: Никто не знает что такое модульный монолит
- Что обычно называют модульным монолитом:

  ❌ **Разложить по папкам\сборкам** — internal классы и статические ссылки всё равно протекают, ссылки между модулями никак не контролируются.

  ❌ **Распределенный монолит** — когда (микро-)сервисы запускаются в одном процессе и деплоятся вместе, но используют все паттерны MCA.

  ❌ **Свой плагинный движок** — написание велосипедов на Reflection или поверх готовых библиотеках типа MediatR, обязательно с интерфейсом **IModule**.
- В ASP.NET уже давно есть все средства для модульности
- Я сделал нагрузочный тест модульным чтобы показать это

Хост не должен знать, что модульность существует — иначе миграция монолита начинается с правки хоста, то есть с того же «платить вперёд».

docker-compose.yml нагрузочного теста

```yaml
services:
  web-local:
    build: &build
      context: .
      dockerfile: Web/Dockerfile
    ports:
      - 5001:8080
    environment:
        HOSTINGSTARTUPASSEMBLIES: Local

  web-remote-rest:
    build: *build
    ports:
      - 5002:8080
    environment:
        HOSTINGSTARTUPASSEMBLIES: RemoteRest
        Services__rest__http__0: rest-backend:8080
    depends_on:
      - rest-backend

  web-remote-grpc:
    build: *build
    ports:
      - 5003:8080
    environment:
        HOSTINGSTARTUPASSEMBLIES: RemoteGrpc
        Services__grpc__http__0: grpc-backend:8080
    depends_on:
      - grpc-backend

  web-remote-rmq:
    build: *build
    ports:
      - 5004:8080
    environment:
        HOSTINGSTARTUPASSEMBLIES: RemoteRmq
        ConnectionStrings__Rabbit: Host=rabbit
    depends_on:
      rabbit:
        condition: service_started
        restart: true
```

- Все 4 тестовых сайта запускаются из одного и того же образа
- Разница только в переменной окружения ***HOSTINGSTARTUPASSEMBLIES***
- Загружаются ***только*** указанные модули
- Внутри свой плагинный движок?

Program.cs: найдите здесь модули

```
var builder = WebApplication.CreateBuilder(args);

var services = builder.Services;
services.AddServiceDiscovery();
services.ConfigureHttpClientDefaults(static http ⇒
{
    // Turn on service discovery by default
    http.AddServiceDiscovery();
});
services.AddMvc();

var app = builder.Build();
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

await app.RunAsync();
```

Никаких упоминании IModule

…и совершенно обычный контроллер

```
[Route(template: "[controller]")]
[ApiController]
public class TestController(ICalculator calculator) : ControllerBase
{
    [HttpGet]
    public Task<int> Get(int a, int b) ⇒ calculator.Add(a, b);
}
```

Контроллер живёт в хосте и ничего не знает о модулях

Модули - отдельные

```
[assembly: HostingStartup(typeof(RemoteRmq.Module))]
namespace RemoteRmq;

class Module : IHostingStartup
{
    public void Configure(IWebHostBuilder builder)
    {
        builder.ConfigureServices((WebHostBuilderContext ctx, IServiceCollection services) => services
            .AddTransient<ICalculator,Calculator>()
            .AddHostedService<CalculatorBackend>()
            .AddEasyNetQ(ctx.Configuration.GetConnectionString(name: "Rabbit")).UseSystemTextJson()
        );

    }
}

public readonly record struct AddRequest(int A, int B);

public class Calculator(IRpc rpc) : ICalculator
{
    public Task<int> Add(int a, int b) =>
        rpc.RequestAsync<AddRequest, int>(new AddRequest(a, b),
            IRequestConfiguration c => c.WithExpiration(TimeSpan.FromSeconds(60)));
}

internal class CalculatorBackend(IRpc rpc) : IHostedService
{
    IAsyncDisposable? subscription = null;
    public async Task StartAsync(CancellationToken cancellationToken)
    {
        subscription = await rpc.RespondAsync<AddRequest, int>(
            AddRequest x => x.A + x.B, cancellationToken: cancellationToken);
    }

    public async Task StopAsync(CancellationToken cancellationToken)
    {
        await subscription!.DisposeAsync();
    }
}
```

```
[assembly: HostingStartup
namespace RemoteGrpc;

class Module : ModuleBase
{
    protected override vo
    {
        services
            .AddTransient
            .AddGrpcClien
                c.Address
            );
    }
}

class Calculator(Grpc.Ser
{
    public async Task<int
        A = a,
        B = b
    })).Result;
}
```

```
[assembly: HostingStartu
nam
cla
{
}
pub
{
}
```

```
[assembly: HostingS
namespace RemoteRes

class Module : IHos
{
    public void Con
    {
        builder.Con
            service
        );
    }
}

public class Calcul
{
    public async Ta
}
```

Ссылка между модулями — это утверждение о деплое

Модуль A использует из модуля B один DTO → **любая топология с A обязана содержать B**

Компилятор эмитит ссылку только на сборку, чьи типы реально использованы — список ссылок точен

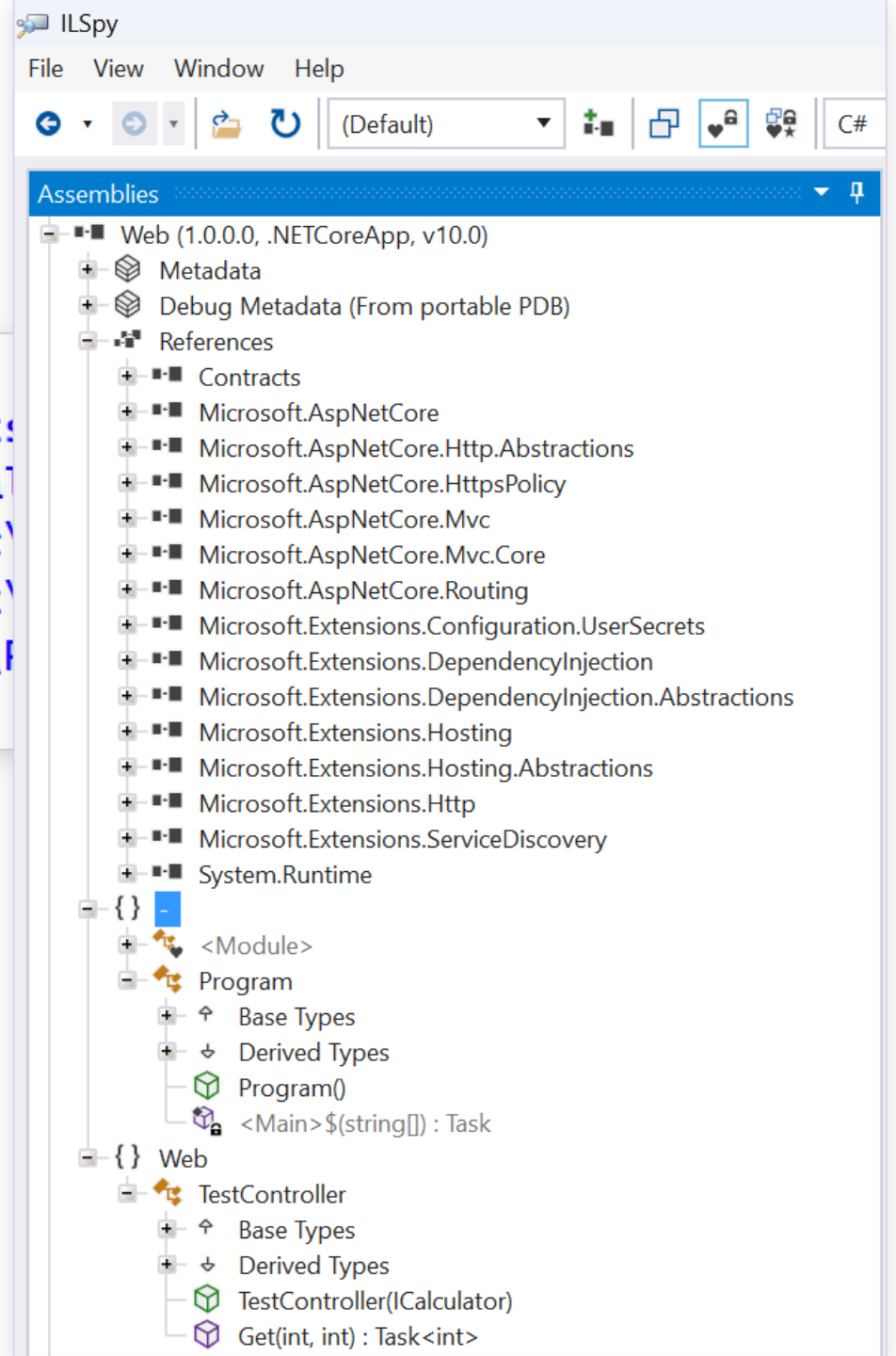
Рантайм уронит старт, если B не активирован

Выход: тип, нужный двоим, переезжает в **contracts-библиотеку** — обычный `Microsoft.NET.Sdk`, без `[HostingStartup]` — не модуль. Типы — публичные.

Как модули подключаются

```
<ItemGroup>
  <ProjectReference Include="..\Contracts\Contracts
  <ProjectReference Include="..\Modules\Local\Local
  <ProjectReference Include="..\Modules\RemoteGrpc\
  <ProjectReference Include="..\Modules\RemoteRest\
  <ProjectReference Include="..\Modules\RemoteRmq\R
</ItemGroup>
```

- Reference в сборке не будет, если вы не будете использовать типы модуля в хосте



Где же магия?

Загрузка модулей происходит внутри `WebApplication.CreateBuilder`

1. Читает переменные окружения
   - `hostingStartupAssemblies`
   - `hostingStartupExcludeAssemblies`
2. В переменных имена сборок, разделенные точкой с запятой
3. Убирает из первого списка второй и добавляет текущую сборку в начало
4. Для каждой сборки в списке
   a. Вызывает `Assembly.Load`
   b. Получает `HostingStartupAttribute`
   c. Создает экземпляр типа, указанного в качестве параметра атрибута
   d. Приводит к `IHostingStartup` и вызывает метод `Configure`

https://learn.microsoft.com/en-us/aspnet/core/fundamentals/host/platform-specific-configuration

Где магия молчит

У механизма три следствия. Два из них молчат — и оба роняли прод.

🔇

Нет [HostingStartup]

Сборка загружается и игнорируется. Ни ошибки, ни лога, ни единого признака.

🔇

Опечатка в имени

Имя модуля в переменной с опечаткой: `crit` в лог — и старт продолжается. Readiness-проба зелёная, на всё 404.

✅

Порядок = порядок в переменной

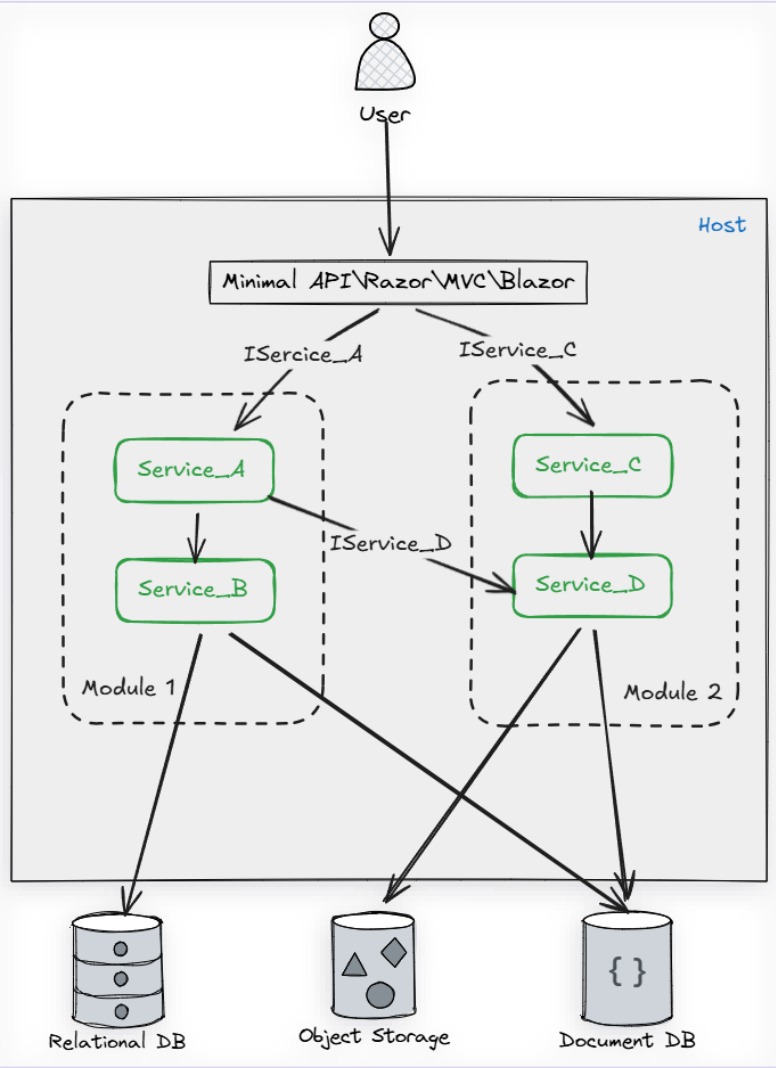
Entry assembly идёт первой. Это контракт — на него опирается связка сущностей между модулями.

Сервисы и конфигурация в модулях

- Внутри `IHostingStartup.Configure` можно вызывать:
  - `IWebHostBuilder.ConfigureServices`
    - для регистрации любых сервисов в Service Provider
    - Включая EF Core, API клиенты и Hosted-сервисы
  - `IWebHostBuilder.ConfigureAppConfiguration`
    - для добавления или параметров в IConfiguration

- Не нужно вызывать `IWebHostBuilder.Configure` – это полностью заменит существующий пайплайн

Какая архитектура получилась



- Мы можем загружать независимые модули и сервисы в них
- Каждый модуль может иметь свою архитектуру и свои хранилища
- Модули могут общаться через интерфейсы

- Если не будет хватать зависимостей, то эндпоинты будут падать

Модульные эндпоинты

Для регистрации эндпоинтов в модуле используем IStartupFilter

IStartupFilter в Service Provider вызвает свой код перед app.Run

```
class Module : IHostingStartup, IStartupFilter
{
    public void Configure(IWebHostBuilder builder)
    {
        builder.ConfigureServices(IServiceCollection services ⇒
        {
            services.AddSingleton<IStartupFilter>(this);
        });
    }

    public Action<IApplicationBuilder> Configure(Action<IApplicationBuilder> next) ⇒
    IApplicationBuilder builder ⇒ {
        next(builder); // Не забываем вызвать следующий модуль

        builder.UseEndpoints(IEndpointRouteBuilder app ⇒...);
    };
}
```

Модульность ASP.NET MVC и Razor Pages

Они уже модульные: добавляете ссылку на проект с контроллерами или страницами и можете делать HTTP запросы к ним

- **ProjectReference** есть
- В коде не используются типы или неймспейсы из другого проекта

… и это работает, но как?

Application Part Discovery

При сборке:

- Если собираемый проект использует **Microsoft.NET.Sdk.Web**
- Для всех **ProjectReference** из собираемого проекта
  - Которые сами ссылаются на сборки **Microsoft.AspNetCore.Mvc.***
  - Создается **ApplicationPartAttribute** для сборки

В рантайме при вызове **AddRazorPages\AddMvc\AddControllers**:

1. Берет все **ApplicationPartAttribute** текущей сборки
2. Загружает сборки и добавляет **ApplicationPart**

https://learn.microsoft.com/en-us/aspnet/core/mvc/advanced/app-parts

Как подружить с Hosting Startup

- *Application Part Discovery* не знает о *Hosting Startup* и не запускает код модуля
- Отключается свойством `GenerateMvcApplicationPartsAssemblyAttributes=false` в свойствах проекта
- Контроллеры и страницы *модуль* подключает сам: `ApplicationPart` в коде модуля

Цена, если не отключить

Хост получает контроллеры модуля, **не запустив код модуля** — в каждой топологии, включая те, что этот модуль намеренно исключили. Ни один тест на маршруты внутри модуля этого не увидит.

Application Part Discovery

По умолчанию

```
[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: Debuggable(/*Could not decode attribute arguments.*/)]
[assembly: TargetFramework(".NETCoreApp,Version=v10.0", FrameworkDisplayName = ".NET 10.0")]
[assembly: AssemblyCompany("Host")]
[assembly: AssemblyConfiguration("Debug")]
[assembly: AssemblyFileVersion("1.0.0.0")]
[assembly: AssemblyInformationalVersion("1.0.0+c46fd782bebe7d3294c8f3ed5206f366dadbc60f")]
[assembly: AssemblyProduct("Host")]
[assembly: AssemblyTitle("Host")]
[assembly: ApplicationPart("Microsoft.AspNetCore.OpenApi")]
[assembly: ApplicationPart("MvcModule")]
[assembly: ApplicationPart("RazorModule")]
[assembly: AssemblyVersion("1.0.0.0")]
[module: RefSafetyRules(11)]
```

После отключения

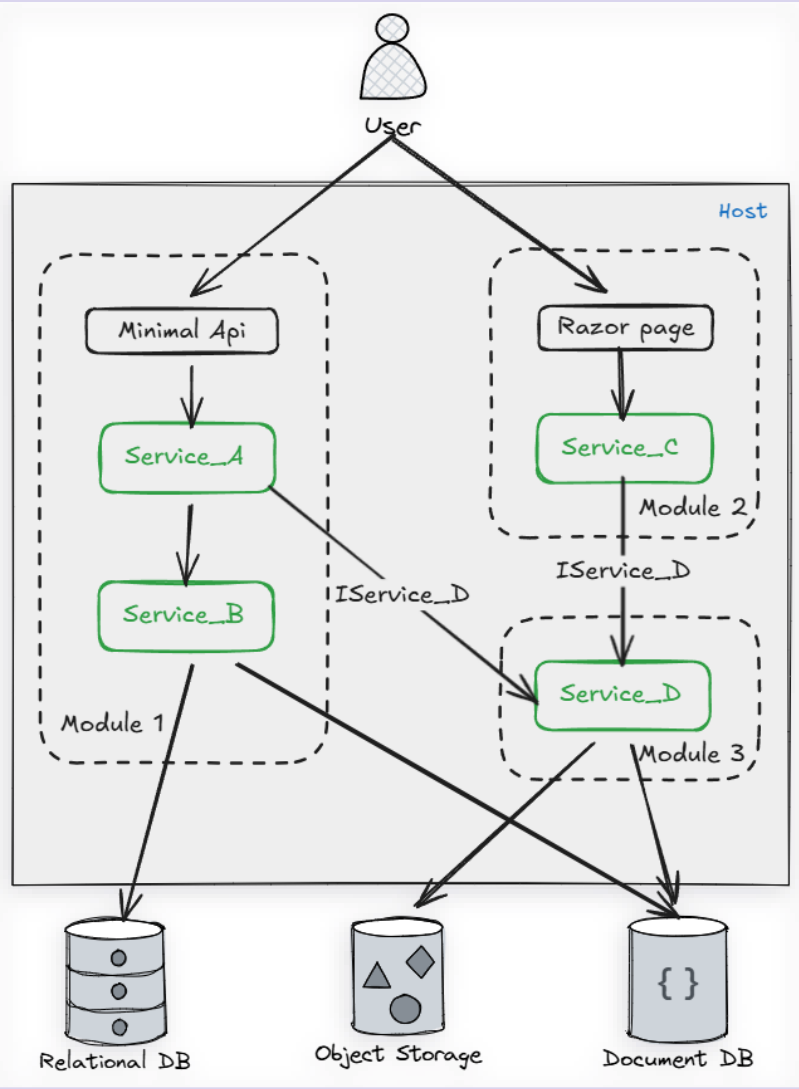
```
[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: Debuggable(/*Could not decode attribute arguments.*/)]
[assembly: TargetFramework(".NETCoreApp,Version=v10.0", FrameworkDisplayName = ".NET 10.0")]
[assembly: AssemblyCompany("Host")]
[assembly: AssemblyConfiguration("Debug")]
[assembly: AssemblyFileVersion("1.0.0.0")]
[assembly: AssemblyInformationalVersion("1.0.0+c46fd782bebe7d3294c8f3ed5206f366dadbc60f")]
[assembly: AssemblyProduct("Host")]
[assembly: AssemblyTitle("Host")]
[assembly: AssemblyVersion("1.0.0.0")]
[module: RefSafetyRules(11)]
```

▶ LIVE

Демо

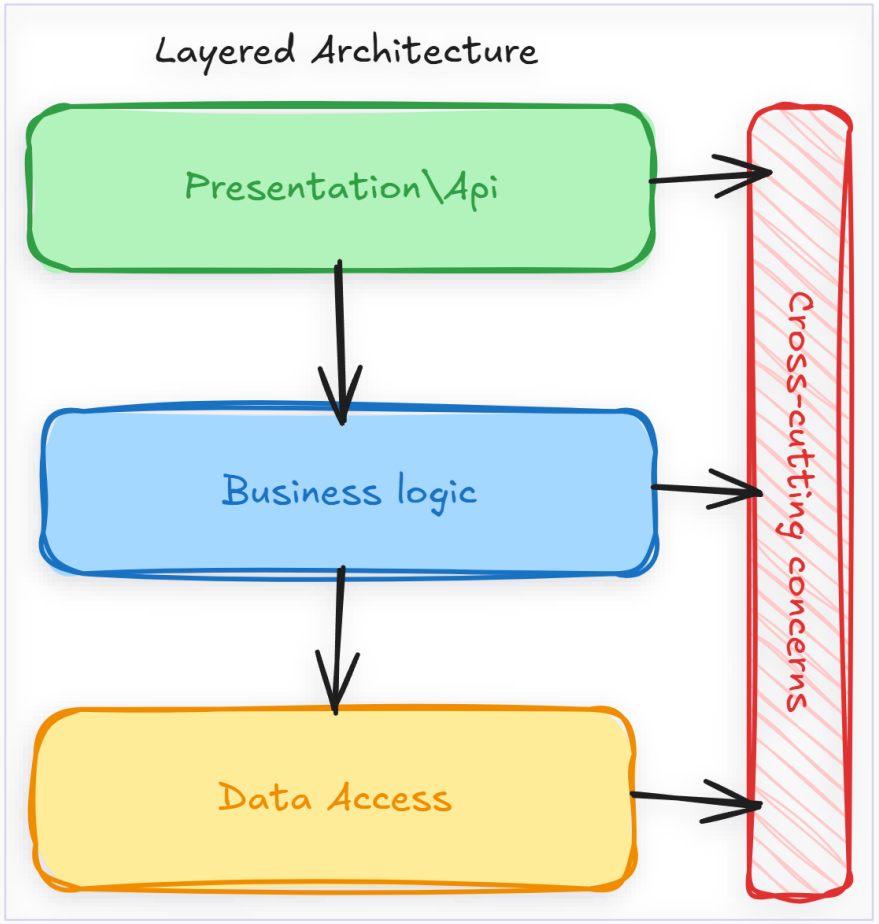
Модульность для Minimal API, Razor pages и MVC

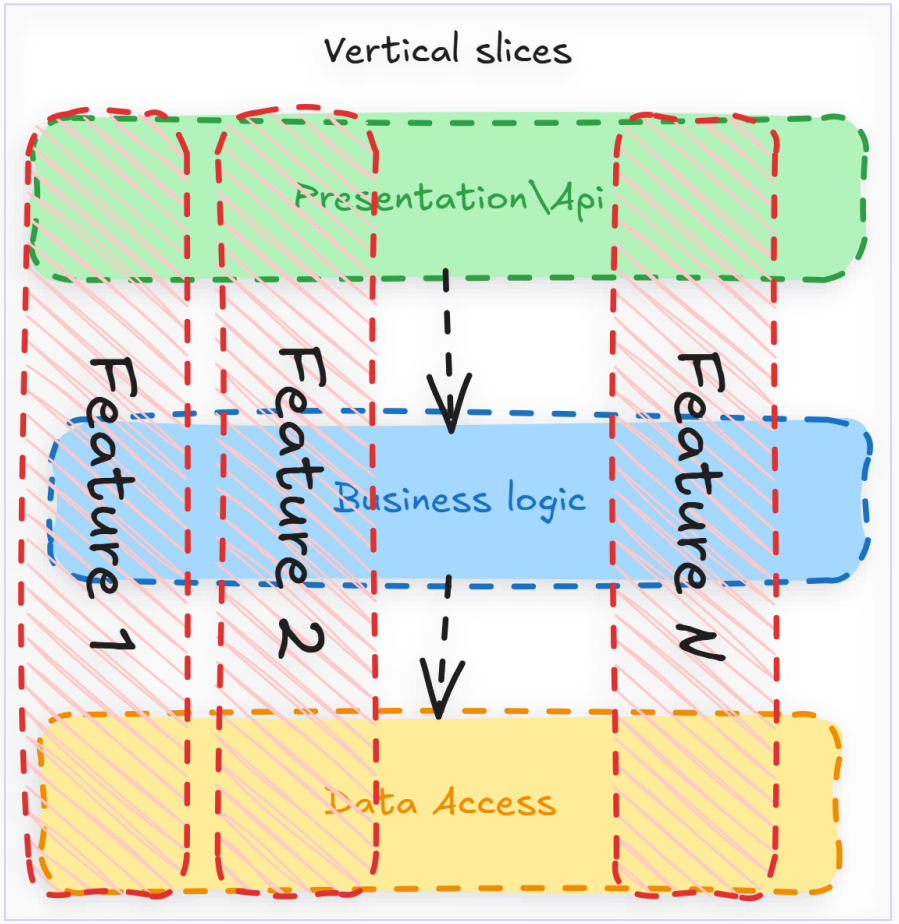
Модульный монолит ASP.NET



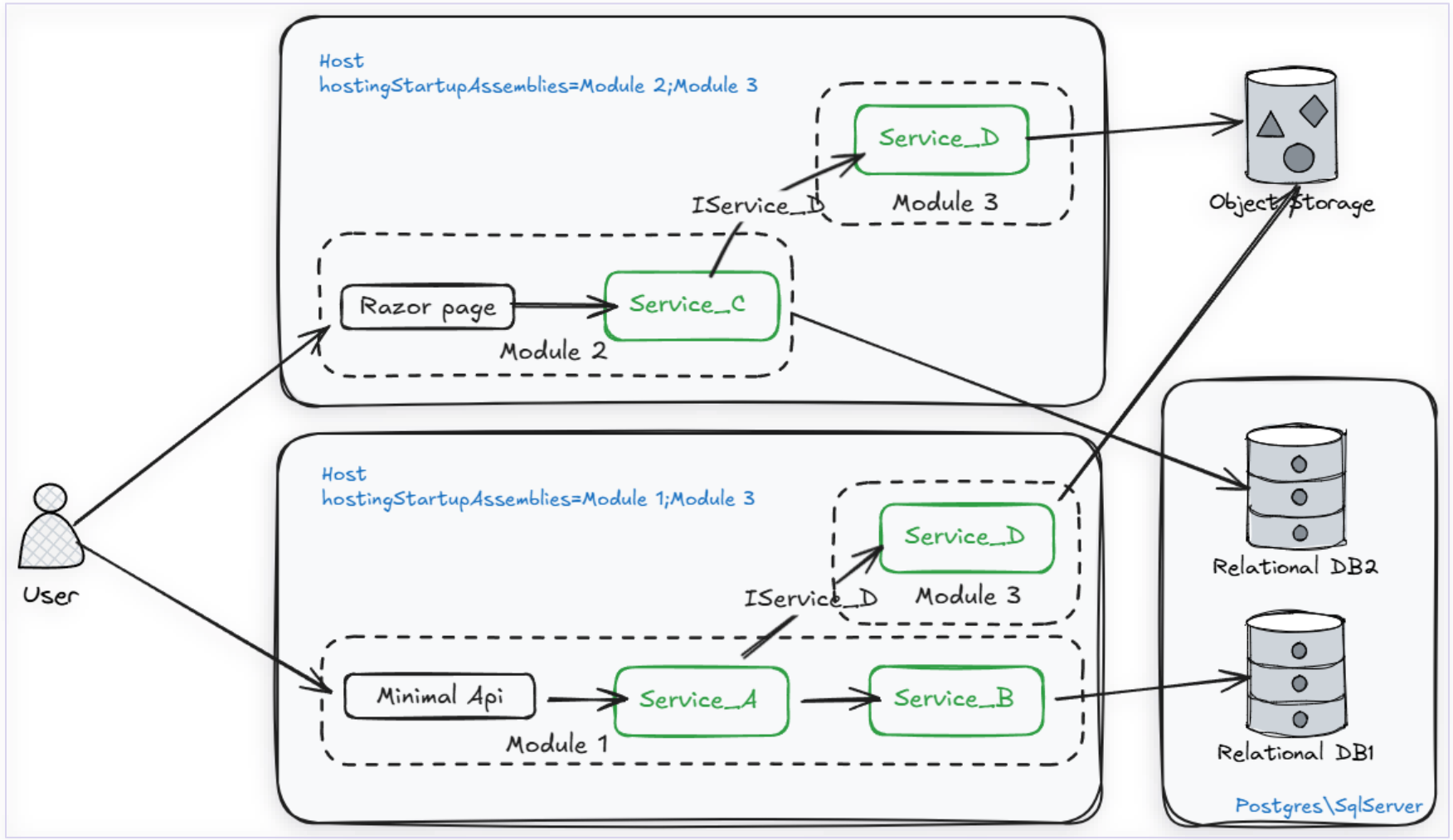
- Можем при загрузке включать или отключать части приложения
- Модули могут вызывать друг друга через публичные интерфейсы
- Модуль может быть vertical slice

Layered vs Vertical Slices





Масштабирование модульного монолита



- Один образ
- У каждого экземпляра свой набор модулей
- Для отладки можно запустить все по F5

- Экземпляры могут вызывать друг друга
  - по API в отдельном модуле
  - Через очередь

Модульная модель данных

- В EF Core можно собрать модель по частям
- Надо реализовать `IEntityTypeConfiguration<T>` в модулях
- В хосте собрать модель из модулей

```
class AppContext(DbContextOptions options) : DbContext(options)
{
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        // Сканируем все загруженные сборки на наличие атрибута HostingStartup
        var IEnumerable<Assembly>? hostingStartupAssemblies = AppDomain.CurrentDomain.GetAssemblies()
            .Where(Assembly a ⇒ a.GetCustomAttributes<HostingStartupAttribute>().Any());

        foreach (var Assembly assembly in hostingStartupAssemblies)
        {
            modelBuilder.ApplyConfigurationsFromAssembly(assembly);
        }
    }
}
```

⚠ До второго хоста

`AppDomain` верен, пока в процессе один хост.

Модульная модель данных

Customer.Entity

```
public class Customer
{
    public int Id { get; set; }

    [MaxLength(length: 50)]
    public string Name { get; set; } = null!;

    [MaxLength(length: 50)]
    [EmailAddress]
    public string Email { get; set; } = null!;
}
```

```
class CustomerEntityConfiguration : IEntityTypeConfiguration<Customer>
{
    public void Configure(EntityTypeBuilder<Customer> entity)
    {
        entity.ToTable(nameof(Customers), schema: "Marketing");
    }
}
```

Order.Entity

```
public class Order
{
    public int Id { get; set; }
    public int CustomerId { get; set; }
    public DateTime OrderDate { get; set; }
    public DateTime? PaymentDate { get; set; }
    public DateTime? ShippedDate { get; set; }
    public decimal Amount { get; set; }
}
```

```
class OrderEntityConfiguration : IEntityTypeConfiguration<Order>
{
    public void Configure(EntityTypeBuilder<Order> entity)
    {
        entity.ToTable(nameof(Orders), schema: "Sales");
    }
}
```

Модуль зависит от DbContext, а не от вашего контекста

- В модуле

```
builder.UseEndpoints(IEndpointRouteBuilder endpoints ⇒ endpoints.MapGet(pattern: "/customers",
    (int[] id, DbContext ctx) ⇒ ctx.Set<Customer>().Where(Customer c ⇒ id.Contains(c.Id)))
);
```

- В хосте

```
services.AddTransient<DbContext>(IServiceProvider sp ⇒ sp.GetRequiredService<AppContext>());
```

- **Модулю нужно место для таблиц, а не полная модель данных**
- **Цена:** теряются `DbSet`-свойства и конвенции уровня контекста. Взамен модуль работает в приложении, для которого не писался

Миграции модульной модели

- Для создания миграций надо загружать все модули конфигураций

```
dotnet ef migrations add [name] `
   --project Web `
   -- `
   HOSTINGSTARTUPASSEMBLIES="Orders.Entities;Customers.Entities"
```

- `DbContext.Database.MigrateAsync` по умолчанию падает когда модель отличается от снэпшота
  - Понижать до лога, **а не подавлять:**

```
services.ConfigureDbContext<AppContext>(DbContextOptionsBuilder b ⇒
    b.ConfigureWarnings(WarningsConfigurationBuilder o ⇒
        o.Log(RelationalEventId.PendingModelChangesWarning))
);
```

На полной модели предупреждение всё ещё что-то значит.

Миграции: пустая миграция без единой ошибки

`dotnet ef` поднимает хост и честно активирует модули из `HOSTINGSTARTUPASSEMBLIES` — **той оболочки, из которой запустили команду**:

Переменная не задана	→ **Пустая миграция**	**0 ошибок**
Задана подмножеством	→ **Миграция для подмножества**	**0 ошибок**

Схема базы зависит от того, на чьей машине сгенерировали миграцию.

Лечится design-time фабрикой

Состав модулей перечислен в коде, а не взят из окружения.

Связи между сущностями

Модуль `OrderProcessing` зависит от `Customer.Entity` и `Order.Entity`

```
using Orders;
using Customers;

class OrderEntityConfiguration : IEntityTypeConfiguration<Order>
{
    public void Configure(EntityTypeBuilder<Order> entity)
    {
        entity.HasOne<Customer>()
              .WithMany()
              .HasForeignKey(Order e ⇒ e.CustomerId);
    }
}
```

Порядок обязателен

Composing-модуль — **последним** в `HOSTINGSTARTUPASSEMBLIES`: EF Core применяет конфигурации в порядке активации.

Не антипаттерн ли это?

Если не делать связи, то придется

1. Или передавать коллекции сущностей между сервисами и делать джоины в приложении
2. Или держать для каждого модуля копию свою данных
   - Делать джоин с копией
   - Синхронизировать данные из источника во все копии
   - Учитывать неконсистентность копии

Если нужно делать **JOIN**, то его придется делать, вопрос где и как

JOIN в базе vs JOIN в приложении

Monolith · Module · Configure(Action<IApplicationBuilder> next)

```
25  group.MapGet("/unpaid", (DbContext ctx) =>
26  {
27      return from o in ctx.Set<Order>()
28             join c in ctx.Set<Customer>() on o.CustomerId equals c.Id
29             where o.PaymentDate == null
30             where o.OrderDate < DateTime.UtcNow.AddDays(-3)
31             select new
32             {
33                 OrderId = o.Id,
34                 o.OrderDate,
35                 Customer = new
36                 {
37                     c.Id,
38                     c.Name,
39                     c.Email
40                 }
41             };
42  })
43  .WithName("Unpaid");
44
45  endpoints.MapPost("/seed", async (DbContext ctx, CancellationToken ct) =>...);
82
83
84
```

OrdersMicroservice · Module · Configure(IWebHostBuilder builder)

```
27  group.MapGet("/unpaid", async (DbContext ctx, HttpClient http, CancellationToken ct) =>
28  {
29      var ordersQuery = from o in ctx.Set<Order>()
30                        where o.PaymentDate == null
31                        where o.OrderDate < DateTime.UtcNow.AddDays(-3)
32                        select new
33                        {
34                            o.Id,
35                            o.OrderDate,
36                            o.CustomerId
37                        };
38      var orders = await ordersQuery.ToListAsync(ct);
39      var customerIds = orders.Select(o => o.CustomerId).Distinct();
40      var customers = await http.GetFromJsonAsync<Customer[]>(
41          $"http://customers/customers?{string.Join('&', customerIds.Select(i => "id="+i))}",
42          JsonSerializerOptions.Web, ct);
43      var d = customers!.ToFrozenDictionary(c => c.Id);
44      return orders.Select(o =>
45      {
46          var haveCustomer = d.TryGetValue(o.CustomerId, out var customer);
47          return new
48          {
49              OrderId = o.Id,
50              o.OrderDate,
51              Customer = haveCustomer ? new
52              {
53                  customer!.Id,
54                  customer!.Name,
55                  customer!.Email,
56              } : null
57          };
58      });
59
60  })
61  .WithName("Unpaid");
```

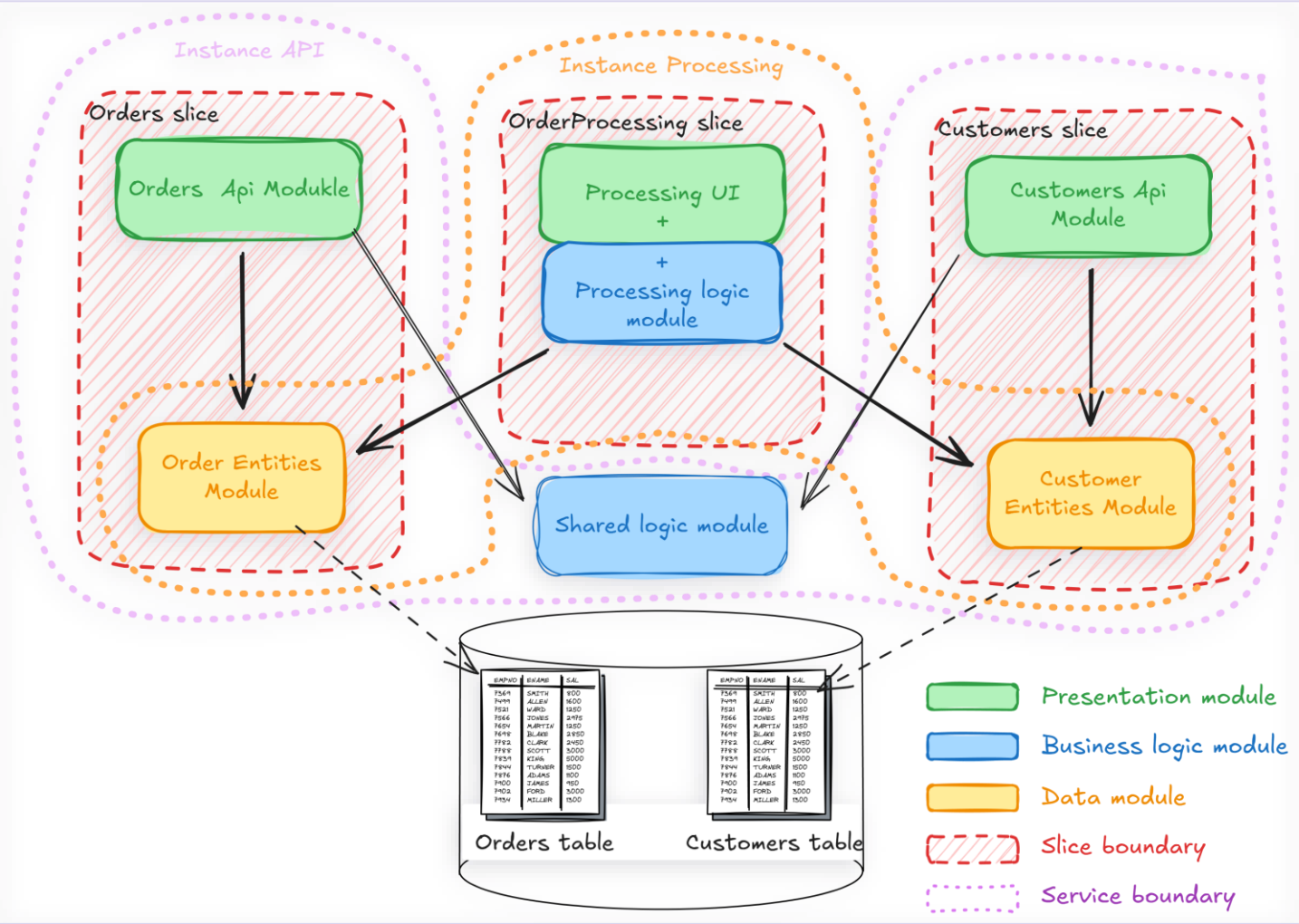
JOIN в базе: на 12% быстрее и вдвое дешевле по памяти

- Вызовы методов с предыдущего слайда
- Приложение и база Postgres в Docker
- 50 VUs, 30 сек

Тест	Request/sec	CPU % Web	Mem MiB Web	CPU % PG	Mem MiB PG
Монолит	2011	509	143	235	126
MCA	1795	224+413	119+149	206	183
Соотношение	1.12	0.8	0.53	1.14	0.69

- Монолит на 12% быстрее, тратя на **пятую часть меньше CPU** и **вдвое меньше памяти**

Модули, слайсы, слои и сервисы



- Один бинарный образ
- Сервис загружает только необходимое подмножество модулей
- Связи между модулями
  1. Ссылка в csproj
  2. Переменная окружения

Модульность можно отменить. Микросервисы — нет

Модульный монолит

Тот же интерфейс, другая реализация. Топология откатывается на remote **одной переменной окружения** — без единой правки бизнес-логики.

Микросервисы

Распил в обратную сторону — **проект на квартал**, а не переменная окружения.

Это не обязательство. Не надо «платить вперёд» — решение можно пересмотреть в любой момент.

Шесть способов сломаться молча

Забыт [assembly: HostingStartup]

Модуль загружается и не делает ничего

Ошибка в переменной окружения

Забыли, опечатались, не указали – health есть, но 404 вместо страниц

Лишние контроллеры

Модуль виден в топологии, которая его исключила

ReferenceOutputAssembly="false"

Модуль не попадает в output — а выглядит как проблема роутинга

Hosted-сервис

Работал в одной реплике — теперь в каждой реплике каждой топологии

Вызвал не то в HostingStartup

Ошибки регистрации middleware, пустой пайплайн, не тот порядок

Как сделать чтобы не ломалось

- ~~Тесты~~
- Roslyn Analyzers
  - Модули не содержат public классов, кроме
    - Контроллеров и их параметров
    - Сущностей модели, указанных в качестве `IEntityTypeConfiguration`
  - Хост-проект
    - Не ссылается на модули
    - Не содержит `ApplicationPartAttribute`, указывающий на модули
- Базовый класс модуля, который проверяет что зависимости загружены и ведет реестр модулей

▶ LIVE

Демо

Как не сломать модульный монолит

Dockerfile модульного монолита

```
# This stage is used to build the service project
FROM mcr.microsoft.com/dotnet/sdk:10.0 AS build
ARG BUILD_CONFIGURATION
WORKDIR /src
COPY --parents **/*.csproj **/*.props *.sln* **/NuGet.config* .
RUN --mount=type=cache,id=nuget,target=/root/.nuget/packages \
    --mount=type=cache,id=modular-data,target=/app/build \
    dotnet restore --artifacts-path /app/build
COPY . .
WORKDIR "/src/Web"
RUN --mount=type=cache,id=nuget,target=/root/.nuget/packages \
    --mount=type=cache,id=modular-data,target=/app/build \
    dotnet build \
    --no-restore \
    --artifacts-path /app/build \
    -c $BUILD_CONFIGURATION

# This stage is used to publish the service project to be copied to the final stage
FROM build AS publish
ARG BUILD_CONFIGURATION
RUN --mount=type=cache,id=nuget,target=/root/.nuget/packages \
    --mount=type=cache,id=modular-data,target=/app/build \
    dotnet publish \
    --no-restore \
    --no-build \
    --artifacts-path /app/build \
    -c $BUILD_CONFIGURATION \
    /p:UseAppHost=false \
    -o /app/publish
```

- Один контейнер для всех сервисов
- Multi-stage build
- Не делаем работу дважды
- Cache mounts
  - NuGet пакетов
  - Артефактов для инкрементальной сборки
- Сброс кэшей одним флагом --no-cache
- Или используйте Container Publishing ☺

⚠ **AOT и trimming для модульного хоста недоступны** — модули грузятся по имени, тримминг их не видит.

Тестирование

- Модульные тесты через InternalsVisibleTo
- Интеграционные тесты модулей

```
public class WebApplicationFactory : WebApplicationFactory<Program>, IAsyncInitializer
{
    public Task InitializeAsync()...

    protected override void ConfigureWebHost(IWebHostBuilder builder)
    {
        builder.UseSetting(WebHostDefaults.HostingStartupAssembliesKey, value: "ApiModule");
        base.ConfigureWebHost(builder);
    }
}
```

⚠ Оба вида тестов упираются в **кэш модели EF Core**, как только собирают вторую модель из того же типа контекста.

Второй хост ломает модель

AppDomain.GetAsseblies() и фильтр по `HostingStartupAttribute` даёт ровно активированные модули в порядке активации во всем процессе. Ломается, когда хостов в процессе несколько — в тестах

Топология	Настройка	Тест
ApiModule	**Host;ApiModule**	**все три модуля**
RazorModule	**RazorModule**	**все три модуля**
MvcModule	**MvcModule**	**все три модуля**

Не только ваши модули

В хосте приложения попадают и чужие hosting startup: IISIntegration, а в живом приложении — Application Insights, OpenTelemetry, browser refresh из `dotnet watch`.

Замена — реестр

Модуль при активации пишет себя в `Modulith:Modules:<AssemblyName>` в конфигурации, а после build пролехост читает обратно в порядке активации. Хост по-прежнему не знает о модулях, а design-time получает тот же механизм вместо второго.

Кэш модели EF Core

EF Core кэширует построенную модель **по типу контекста** — во внутреннем сервис-провайдере, общем для всех контекстов с одинаковыми опциями

Два хоста с разными наборами модулей в одном процессе — то есть **любая сборка интеграционных тестов** — молча делят модель первого

Ничего не падает.

Просто не те таблицы.

Лечится IModelCacheKeyFactory

Ключ кэша включает отпечаток набора модулей. Проверено от обратного: убираешь фабрику — три теста краснеют.

Заключение: не исходники, а инструмент

- В ASP.NET и EF Core модульный монолит собирается из коробки
- Нет «налога» на микросервисы
- Автоматический контроль модульности в решении анализатором
- Локальная отладка по F5
- **Главное**: Модули - это не «*первый шаг к микросервисам*» Модули в монолите превосходят MCA в большинстве случаев

- Примеры — https://github.com/gandjustas/dotnext-2026
- Пакет и скилл — https://github.com/gandjustas/FusionModules

📦 NuGet-пакет

Одна `PackageReference`: базовый класс, девять правил анализатора, MSBuild-конвенции.

🤖 ИИ-скилл для миграции

Запилить модульный монолит из микросервисов. Качаете, добавляете в своей проект превращаете сервисы в модули.

Табло: шесть обещаний микросервисов

✓	**Масштабирование**	Топология — переменная окружения, а не новый сервис
✓	**Управление сложностью**	Анализаторы обеспечивают границы, которые распил только обещает
✓	**Продуктивность**	F5 запускает всё целиком
✓	**Независимый деплой**	Один образ, разные наборы модулей
✗	**Изоляция сбоев**	OOM в модуле роняет процесс целиком
✗	**Полиглот**	Только .NET: C#, F#, Python (CSnakes), JS (много всего)

4 : 2

Два честных проигрыша делают четыре победы убедительнее, а не слабее.

Шесть настоящих причин сделать отдельный сервис

1 **Другой TFM**
…или вообще не .NET

2 **Отдельная команда**
со своим релизным циклом

3 **Изоляция как смысл**
комплаенс, blast radius, мультитенантность

4 **Другой профиль нагрузки**
Уже сейчас, а не «в будущем»

5 **Другой SLA**
и другой change-management

6 **Хранилище**
Которое решает бизнес-задачу – durable очередь, поисковый индекс, итд

Ваша причина в этом списке?

Make Modules, Not Microservices

Станислав Выщепан

@gandjustas

me@stanislav-v.ru