Laboratorium 5: . NET Aspire

Technologie Aplikacji Rozproszonych

Materiały do kolokwium

7 grudnia 2025

Spis treści

1 Teoria: .NET Aspire

1.1 Czym jest .NET Aspire?

Definicja

.NET Aspire to stack technologiczny zaprojektowany do budowy obserwowalnych, gotowych na produkcję aplikacji rozproszonych. Jest dostarczany poprzez zestaw pakietów NuGet, które obsługują konkretne potrzeby aplikacji chmurowych.

1.1.1 Główne cele . NET Aspire

- **Orkiestracja** – Uproszczone zarządzanie wieloma projektami i zależnościami
- **Integracje** – Łączenie z różnymi usługami (bazy danych, messaging, storage)
- **Narzędzia** – Wsparcie dla lokalnego developmentu i testowania
- **Telemetria** – Wbudowane logowanie, metryki i tracing
- **Deployment** – Gotowe do wdrożenia w środowisku produkcyjnym

1.2 Kluczowe komponenty

1.2.1 1. App Host (Orkiestrator)

App Host

App Host to projekt orkiestracyjny, który definiuje i konfiguruje wszystkie komponenty aplikacji rozproszonej.

Charakterystyka:

- Centralny punkt konfiguracji całej aplikacji
- Definiuje zależności między serwisami
- Zarządza cyklem życia komponentów
- Automatycznie konfiguruje Service Discovery
- Generuje manifesty dla deployment

Typowa struktura App Host:

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

// Dodanie zasobow
var cache = builder.AddRedis("cache");
var db = builder.AddPostgres("postgres")
               .AddDatabase("mydb");

// Dodanie serwisow
var apiService = builder.AddProject<Projects.ApiService>("apiservice")
                       .WithReference(cache)
```

```
                          .WithReference(db);

builder.AddProject<Projects.Web>("webfrontend")
       .WithReference(apiService);

builder.Build().Run();
```

1.2.2 2. Service Defaults

Service Defaults

Wspólna konfiguracja dla wszystkich serwisów w aplikacji, zapewniająca spójne ustawienia telemetrii, health checks i resiliency.

Co zawierają Service Defaults:

- Konfiguracja OpenTelemetry (logging, metrics, tracing)
- Health checks
- Service Discovery
- Standardowe middleware
- HTTP client factory z resilience patterns

1.2.3 3. Service Discovery

Service Discovery

Automatyczny mechanizm wykrywania i komunikacji między serwisami bez hard-codowania adresów URL.

Jak działa:

1. App Host rejestruje wszystkie serwisy
2. Każdy serwis otrzymuje unikalną nazwę
3. Komunikacja odbywa się przez nazwę serwisu, nie przez URL
4. W runtime nazwy są rozwiązywane do rzeczywistych adresów

Przykład użycia:

```
// W App Host
var apiService = builder. AddProject<Projects.ApiService>("apiservice");
builder.AddProject<Projects.Web>("webfrontend")
       .WithReference(apiService);

// W serwisie Web
var response = await httpClient.GetAsync("http://apiservice/api/data");
// "apiservice" zostanie automatycznie rozwiazane
```

Kategoria	Przykłady
Bazy danych	PostgreSQL, MySQL, SQL Server, MongoDB
Cache	Redis, Valkey
Messaging	RabbitMQ, Azure Service Bus, Kafka
Storage	Azure Blob Storage, AWS S3
Cloud Services	Azure, AWS, GCP

1.3 Integracje (Components)

. NET Aspire dostarcza gotowe integracje z popularnymi technologiami:

1.4 Hosting Integrations vs Client Integrations

1.4.1 Hosting Integrations

- Używane w **App Host**
- Zarządzają cyklem życia zasobów
- Prefix: `Aspire.Hosting.*`
- Przykład: `Aspire.Hosting.Redis`

```
var redis = builder.AddRedis("cache");
```

1.4.2 Client Integrations

- Używane w **serwisach aplikacyjnych**
- Konfigurują klientów do łączenia się z zasobami
- Prefix: `Aspire.*`
- Przykład: `Aspire. StackExchange.Redis`

```
builder.AddRedisClient("cache");
```

1.5 Telemetria i Observability

OpenTelemetry

. NET Aspire ma wbudowane wsparcie dla OpenTelemetry, zapewniając kompletną obserwowalność aplikacji.

1.5.1 Trzy filary observability:

1. **Logging** – Strukturalne logi ze wszystkich serwisów
2. **Metrics** – Metryki wydajności i biznesowe
3. **Tracing** – Rozproszone śledzenie requestów przez wszystkie serwisy

1.5.2 Dashboard

Aspire dostarcza wbudowany dashboard dostępny podczas developmentu:

- Widok wszystkich serwisów i ich statusu
- Live logs z wszystkich komponentów
- Metryki i wykresy
- Distributed tracing
- Console output

1.6 Deployment i Manifesty

1.6.1 Manifest

. NET Aspire może wygenerować manifest opisujący całą aplikację:

```
dotnet run --project AppHost --publisher manifest --output-path manifest
    .json
```

Manifest zawiera:

- Definicje wszystkich serwisów
- Zależności między nimi
- Wymagane zasoby (bazy danych, cache, itp.)
- Zmienne środowiskowe
- Konfigurację portów

1.6.2 Opcje deployment:

- **Azure Container Apps** – Natywne wsparcie
- **Kubernetes** – Przez konwersję manifestu
- **Docker Compose** – Lokalne środowisko produkcyjne
- **Azure Developer CLI (azd)** – Szybki deployment do Azure

1.7 Dlaczego używać .NET Aspire?

Korzyści

Zalety używania .NET Aspire:

1. **Uproszczony development lokalny**
    - Jeden punkt startowy dla całej aplikacji
    - Automatyczne zarządzanie zależnościami

- Live reload i hot reload

2. **Production-ready z pudełka**
    - Wbudowana telemetria
    - Health checks
    - Resilience patterns
3. **Ujednolicona konfiguracja**
    - Service Defaults dla konsystencji
    - Centralna konfiguracja w App Host
    - Łatwe zarządzanie sekretami
4. **Ecosystem integracji**
    - Gotowe komponenty dla popularnych technologii
    - Spójne API dla różnych providerów
    - Łatwa wymiana implementacji
5. **Developer Experience**
    - Wspaniały dashboard
    - Intelligent IntelliSense
    - Visual Studio integration

1.8 Kiedy używać .NET Aspire?

Idealnie dla:

- Aplikacji mikroservices
- Aplikacji cloud-native
- Projektów z wieloma serwisami
- Zespołów potrzebujących spójnej observability

Możliwe overkill dla:

- Prostych aplikacji monolitycznych
- Single-service projektów
- Prototypów bez planów skalowania

2 Materiały do nauki

2.1 Filmy wideo

2.1.1 Seria wprowadzająca

1. Why .NET Aspire

Tytuł: Why .NET Aspire
Link: `https://www.youtube.com/watch?v=8w1Q-J8-vJE`
Opis: Wprowadzenie do problematyki aplikacji rozproszonych i jak .NET Aspire je rozwiązuje. Omówienie głównych celów i filozofii frameworka.

2. What is .NET Aspire

Tytuł: What is .NET Aspire
Link: `https://www.youtube.com/watch?v=tAMYEJpWjAs`
Opis: Szczegółowe omówienie komponentów .NET Aspire: App Host, Service Defaults, Dashboard, integracje.

3. Get started with .NET Aspire

Tytuł: Get started with . NET Aspire
Link: `https://www.youtube.com/watch?v=bRMGEiq53pI`
Opis: Praktyczny tutorial - utworzenie pierwszej aplikacji Aspire, konfiguracja, podstawowe operacje.

2.1.2 Tematy zaawansowane

4. Service Discovery

Tytuł: . NET Aspire Service Discovery
Link: `https://www.youtube.com/watch?v=2ggVL-WsidI`
Opis: Jak działa Service Discovery w Aspire, konfiguracja, best practices dla komunikacji między serwisami.

5. Service Defaults

Tytuł: . NET Aspire Service Defaults
Link: `https://www.youtube.com/watch?v=Vp9iJYKxX-A`
Opis: Konfiguracja Service Defaults, co jest zawarte domyślnie, jak dostosować do własnych potrzeb.

6. Telemetry with OpenTelemetry

Tytuł: . NET Aspire Telemetry with OpenTelemetry
Link: `https://www.youtube.com/watch?v=01VGGrJcxMQ`
Opis: Integracja z OpenTelemetry, distributed tracing, metryki, eksport do różnych backendów.

7. Components and Integrations

Tytuł: .NET Aspire Components and Integrations
Link: https://www.youtube.com/watch?v=dJ1aaT6bU5s
Opis: Przegląd dostępnych komponentów, jak ich używać, różnice między hosting i client integrations.

8. Inner Loop Development

Tytuł: .NET Aspire Inner Loop Development
Link: https://www.youtube.com/watch?v=Jz5eH1q5cvE
Opis: Workflow podczas developmentu, debugging, hot reload, praca z dashboard.

9. Deployment and Manifest

Tytuł: .NET Aspire Deployment and Manifest
Link: https://www.youtube.com/watch?v=z1M-7Bms1Jg
Opis: Generowanie manifestów, deployment do różnych środowisk, Azure Container Apps, Kubernetes.

2.2 Dokumentacja oficjalna

2.2.1 Microsoft Learn

- **. NET Aspire Overview**
  https://learn.microsoft.com/en-us/dotnet/aspire/get-started/aspire-overview
- **Build your first . NET Aspire app**
  https://learn.microsoft.com/en-us/dotnet/aspire/get-started/build-your-first-asp
- **. NET Aspire orchestration**
  https://learn.microsoft.com/en-us/dotnet/aspire/fundamentals/app-host-overview
- **.NET Aspire integrations**
  https://learn.microsoft.com/en-us/dotnet/aspire/fundamentals/integrations-overvie
- **.NET Aspire service discovery**
  https://learn.microsoft.com/en-us/dotnet/aspire/service-discovery/overview
- **.NET Aspire dashboard**
  https://learn.microsoft.com/en-us/dotnet/aspire/fundamentals/dashboard

2.2.2 Samples i przykłady

- **Official . NET Aspire Samples Repository**
  https://github.com/dotnet/aspire-samples
- **eShop Reference Application**
  https://github.com/dotnet/eshop
- **Community Toolkit**
  https://github.com/CommunityToolkit/Aspire

2.3 Polecane artykuły i blogi

- **.NET Blog - Aspire announcements**
  `https://devblogs.microsoft.com/dotnet/`
- **Introducing .NET Aspire: Simplifying Cloud-Native Development**
  Artykuł wprowadzający od Microsoft
- **Building Microservices with .NET Aspire**
  Praktyczny przewodnik po architekturze mikroservices

2.4 Narzędzia i wymagania

2.4.1 Wymagania systemowe

- **.NET 8 SDK** lub nowszy
- **Visual Studio 2022** (17.9+) lub **Visual Studio Code**
- **Docker Desktop** (dla kontenerów)
- **Workload Aspire**:

```
dotnet workload install aspire
```

2.4.2 Template'y projektów

```
# Lista template'ow Aspire
dotnet new list aspire

# Utworzenie nowego projektu
dotnet new aspire-starter -n MyAspireApp
```

2.5 Plan nauki

1. **Tydzień 1: Podstawy**
   - Obejrzenie filmów 1-3
   - Przeczytanie Aspire Overview
   - Utworzenie pierwszej aplikacji
2. **Tydzień 2: Konfiguracja**
   - Filmy 4-5
   - Praca z Service Discovery i Service Defaults
   - Dodanie pierwszych integracji
3. **Tydzień 3: Observability**
   - Film 6
   - Konfiguracja telemetrii

- Praca z dashboard

4. **Tydzień 4: Production**
    - Filmy 7-9
    - Deployment i manifesty
    - Best practices

3 Cheat Sheet - . NET Aspire

3.1 Instalacja i setup

3.1.1 Instalacja workload

```
# Instalacja . NET Aspire workload
dotnet workload install aspire

# Aktualizacja do najnowszej wersji
dotnet workload update

# Lista zainstalowanych workloads
dotnet workload list
```

3.1.2 Tworzenie nowego projektu

```
# Starter template (podstawowa aplikacja z Web + API)
dotnet new aspire-starter -n MyAspireApp

# Pusty App Host
dotnet new aspire-apphost -n MyAppHost

# Service Defaults
dotnet new aspire-servicedefaults -n MyServiceDefaults

# Przejscie do katalogu
cd MyAspireApp

# Uruchomienie
dotnet run --project MyAspireApp. AppHost
```

3.2 Struktura projektu Aspire

```
MyAspireApp/
|-- MyAspireApp. AppHost/          # Projekt orkiestracyjny
|    |-- Program.cs                # Definicja aplikacji
|    |-- appsettings.json
|-- MyAspireApp.ServiceDefaults/   # Wspolna konfiguracja
|    |-- Extensions. cs
|-- MyAspireApp. ApiService/        # Przykladowy serwis API
|-- MyAspireApp.Web/               # Przykladowy frontend
```

3.3 App Host - podstawowe operacje

3.3.1 Szablon Program.cs

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

// Dodawanie serwisow i zasobow

builder.Build(). Run();
```

3.3.2 Dodawanie projektów

```
// Dodanie projektu . NET
var apiService = builder.AddProject<Projects.ApiService>("apiservice");

// Z referencja do innego serwisu
var webApp = builder.AddProject<Projects. Web>("webapp")
                    .WithReference(apiService);

// Z zmienna srodowiskowa
var api = builder.AddProject<Projects. Api>("api")
                 .WithEnvironment("FEATURE_FLAG", "enabled");

// Z konkretnym portem
var service = builder.AddProject<Projects. Service>("service")
                     .WithHttpEndpoint(port: 5001);
```

3.3.3 Dodawanie kontenerów

```
// Kontener z Docker Hub
var redis = builder.AddContainer("redis", "redis")
                   .WithEndpoint(targetPort: 6379);

// Z woluminem
var postgres = builder.AddContainer("postgres", "postgres")
                      .WithEnvironment("POSTGRES_PASSWORD", "secret")
                      .WithVolume("postgres-data", "/var/lib/postgresql/
    data");

// Z custom image
var myapp = builder.AddContainer("myapp", "myregistry/myimage")
                   .WithTag("latest");
```

3.4 Popularne integracje

3.4.1 Redis

```
# Hosting integration (w AppHost)
dotnet add package Aspire. Hosting.Redis

# Client integration (w serwisie)
dotnet add package Aspire.StackExchange.Redis
```

```
// W AppHost
var cache = builder.AddRedis("cache");
var api = builder.AddProject<Projects. Api>("api")
                 . WithReference(cache);

// W serwisie API - Program.cs
builder.AddRedisClient("cache");

// W serwisie API - uzycie
public class MyService
{
    private readonly IConnectionMultiplexer _redis;

```

```
    public MyService(IConnectionMultiplexer redis)
    {
        _redis = redis;
    }

    public async Task SetValue(string key, string value)
    {
        var db = _redis.GetDatabase();
        await db.StringSetAsync(key, value);
    }
}
```

3.4.2 PostgreSQL

```
# Hosting
dotnet add package Aspire. Hosting.PostgreSQL

# Client
dotnet add package Aspire. Npgsql
```

```
// W AppHost
var postgres = builder.AddPostgres("postgres")
                      .AddDatabase("mydb");

var api = builder.AddProject<Projects.Api>("api")
                 .WithReference(postgres);

// W serwisie
builder.AddNpgsqlDataSource("mydb");

// Uzycie
public class MyDbContext : DbContext
{
    public MyDbContext(DbContextOptions<MyDbContext> options)
        : base(options) { }
}

// Rejestracja
builder.Services.AddDbContext<MyDbContext>();
```

3.4.3 RabbitMQ

```
# Hosting
dotnet add package Aspire. Hosting.RabbitMQ

# Client
dotnet add package Aspire.RabbitMQ. Client
```

```
// W AppHost
var messaging = builder.AddRabbitMQ("messaging");

var worker = builder.AddProject<Projects. Worker>("worker")
                    .WithReference(messaging);

// W serwisie
builder. AddRabbitMQClient("messaging");
```

3.4.4 SQL Server

```
dotnet add package Aspire.Hosting.SqlServer
dotnet add package Aspire.Microsoft.Data.SqlClient
```

```
// W AppHost
var sql = builder.AddSqlServer("sql")
                 .AddDatabase("mydb");

// W serwisie
builder.AddSqlServerClient("mydb");
```

3.5 Service Discovery

3.5.1 Konfiguracja HttpClient

```
// W AppHost - definiowanie zaleznosciwith
var api = builder.AddProject<Projects.Api>("apiservice");
var web = builder.AddProject<Projects.Web>("webfrontend")
                 .WithReference(api);

// W Web projekcie - Program.cs
builder.Services.AddHttpClient<ApiClient>(client =>
{
    client.BaseAddress = new("http://apiservice");
});

// W Web projekcie - uzycie
public class ApiClient
{
    private readonly HttpClient _httpClient;

    public ApiClient(HttpClient httpClient)
    {
        _httpClient = httpClient;
    }

    public async Task<Data[]> GetDataAsync()
    {
        return await _httpClient
            .GetFromJsonAsync<Data[]>("/api/data");
    }
}
```

3.6 Service Defaults

3.6.1 Dodawanie Service Defaults

```
// W kazdym serwisie - Program.cs (na samym poczatku)
var builder = WebApplication.CreateBuilder(args);

builder.AddServiceDefaults(); // <-- Dodaj to

// ...  reszta konfiguracji
```

3.6.2 Co zawierają Service Defaults

```
// Service Defaults automatycznie konfiguruja:
// 1. OpenTelemetry (logging, metrics, tracing)
// 2.  Health checks
// 3. Service discovery
// 4. HTTP client resiliency

// W Extensions.cs mozna dostosowac:
public static IHostApplicationBuilder AddServiceDefaults(
    this IHostApplicationBuilder builder)
{
    builder.ConfigureOpenTelemetry();
    builder.AddDefaultHealthChecks();
    builder. Services.AddServiceDiscovery();
    builder. Services.ConfigureHttpClientDefaults(http =>
    {
        http. AddStandardResilienceHandler();
        http.AddServiceDiscovery();
    });

    return builder;
}
```

3.7 Konfiguracja i zmienne środowiskowe

```
// Dodawanie zmiennych srodowiskowych
var api = builder.AddProject<Projects. Api>("api")
    .WithEnvironment("MY_SETTING", "value")
    .WithEnvironment("ASPNETCORE_ENVIRONMENT", "Development");

// Przekazywanie sekretow
var api = builder.AddProject<Projects.Api>("api")
    .WithEnvironment("ApiKey", builder.Configuration["ApiKey"]);

// Referencja do connectionString
var db = builder.AddPostgres("postgres"). AddDatabase("mydb");
var api = builder.AddProject<Projects. Api>("api")
    .WithReference(db); // Automatycznie dodaje connection string
```

3.8 Dashboard

3.8.1 Dostęp do Dashboard

```
# Dashboard jest automatycznie dostepny po uruchomieniu AppHost
# URL wyswietla sie w konsoli, typowo:
# https://localhost:17123
```

Funkcje Dashboard:

- Widok wszystkich zasobów i serwisów
- Live logs
- Console output

- Metryki
- Distributed tracing
- Environment variables
- Endpoints

3.9 Telemetria i Observability

3.9.1 Custom metrics

```
// Definicja custom metric
public class OrderMetrics
{
    private readonly Counter<int> _ordersProcessed;

    public OrderMetrics(IMeterFactory meterFactory)
    {
        var meter = meterFactory.Create("MyApp.Orders");
        _ordersProcessed = meter.CreateCounter<int>("orders. processed")
    ;
    }

    public void RecordOrderProcessed()
    {
        _ordersProcessed.Add(1);
    }
}

// Rejestracja
builder.Services.AddSingleton<OrderMetrics>();
```

3.9.2 Custom tracing

```
using System.Diagnostics;

public class MyService
{
    private static readonly ActivitySource _activitySource =
        new("MyApp.MyService");

    public async Task DoWorkAsync()
    {
        using var activity = _activitySource
            .StartActivity("ProcessOrder");

        activity?.SetTag("order. id", "12345");

        // ...  logika biznesowa

        activity?.SetStatus(ActivityStatusCode.Ok);
    }
}
```

3.10 Deployment

3.10.1 Generowanie manifestu

```
# Generuj manifest JSON
dotnet run --project AppHost \
  --publisher manifest \
  --output-path ../manifest.json

# Manifest opisuje cala aplikacje i moze byc uzywany
# do deployment na rozne platformy
```

3.10.2 Azure Developer CLI (azd)

```
# Inicjalizacja
azd init

# Deploy do Azure
azd up

# Monitoring
azd monitor

# Cleanup
azd down
```

3.10.3 Docker Compose

```
# Wygeneruj manifest, a nastepnie:
# Uzyj narzedzi do konwersji manifest -> docker-compose.yml
```

3.11 Debugging i Development

3.11.1 Uruchamianie w Visual Studio

- Ustaw AppHost jako startup project
- F5 uruchomi wszystkie serwisy i dashboard
- Mozesz debugowac wszystkie projekty jednoczesnie

3.11.2 Selective startup

```
// Uruchamianie tylko wybranych serwisow
var api = builder.AddProject<Projects.Api>("api")
    . ExcludeFromManifest(); // Nie bedzie w manifescie

// Warunkowe dodawanie
if (builder.Environment.IsDevelopment())
{
    builder.AddProject<Projects.TestService>("testservice");
}
```

3.12 Najczęstsze komendy CLI

```
# Instalacja
dotnet workload install aspire

# Nowy projekt
dotnet new aspire-starter -n MyApp

# Dodanie integracji do AppHost
dotnet add package Aspire. Hosting.Redis

# Dodanie client integration
dotnet add package Aspire.StackExchange.Redis

# Uruchomienie
dotnet run --project MyApp.AppHost

# Watch mode (auto-reload)
dotnet watch --project MyApp.AppHost

# Build
dotnet build

# Generowanie manifestu
dotnet run --project MyApp.AppHost \
  --publisher manifest \
  --output-path manifest.json

# Czyszczenie
dotnet clean
```

3.13 Best Practices

1. **Zawsze używaj Service Defaults** w każdym serwisie
2. **Nie hardcoduj URLs** – używaj Service Discovery
3. **Nazywaj zasoby konsekwentnie**

```
// Dobra praktyka
var cache = builder.AddRedis("cache");
builder.AddRedisClient("cache"); // Ta sama nazwa!

```

4. **Używaj WithReference** dla zależności
5. **Grupuj konfigurację** w extension methods
6. **Monitoruj telemetrię** od początku projektu
7. **Testuj lokalnie z prawdziwymi zależnościami** (Redis, PostgreSQL w kontenerach)
8. **Używaj parametrów** dla secrets w produkcji

3.14 Troubleshooting

Częste problemy

Problem: Serwis nie może połączyć się z bazą danych
Rozwiązanie: Sprawdź czy:

- Używasz `WithReference()` w AppHost
- Dodałeś client integration w serwisie
- Nazwy się zgadzają

Problem: Dashboard nie wyświetla logów
Rozwiązanie: Upewnij się że wywołujesz `builder.AddServiceDefaults()`
Problem: Service Discovery nie działa
Rozwiązanie: Sprawdź czy HttpClient jest skonfigurowany z Service Discovery:

```
builder.Services.ConfigureHttpClientDefaults(http =>
{
    http.AddServiceDiscovery();
});
```

3.15 Kluczowe pakiety NuGet

Pakiet	Przeznaczenie
Aspire. Hosting	Podstawa dla AppHost
Aspire. Hosting.AppHost	AppHost runtime
Aspire. Hosting.Redis	Redis hosting integration
Aspire. Hosting.PostgreSQL	PostgreSQL hosting
Aspire.Hosting.RabbitMQ	RabbitMQ hosting
Aspire.Hosting.SqlServer	SQL Server hosting
Aspire. StackExchange.Redis	Redis client
Aspire. Npgsql	PostgreSQL client
Aspire.RabbitMQ.Client	RabbitMQ client
Aspire.Microsoft.Data. SqlClient	SQL Server client