.net aspire

by Sinan Nar

who am I

consultant

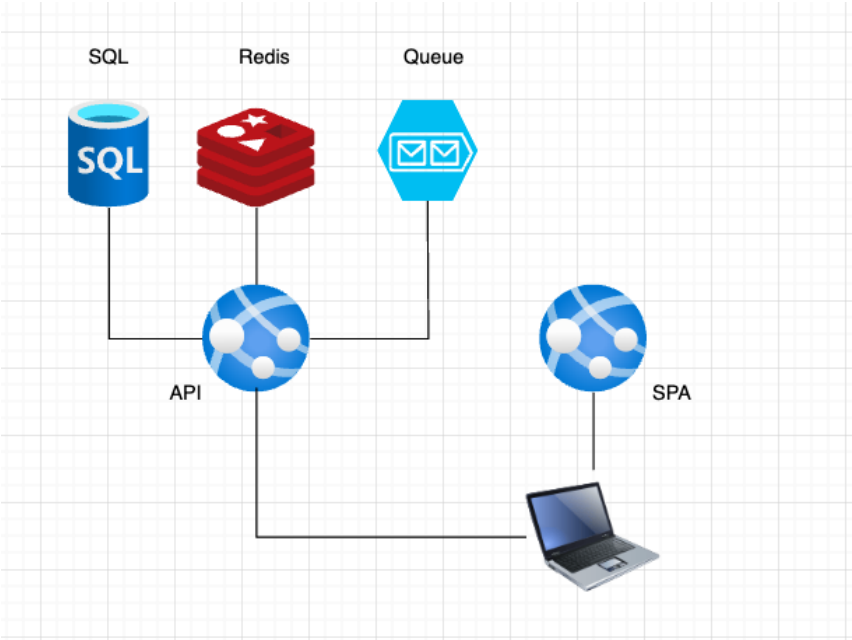
have 8+ yrs exp

have 3 rabbits

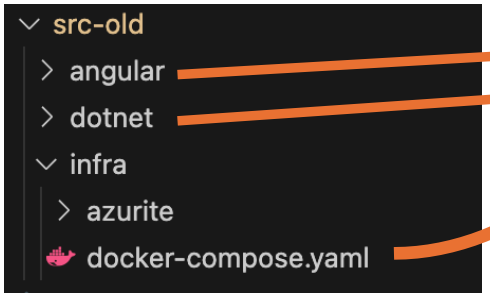
- **Microsoft Certified: AI Fundamentals** — June 2025
- **GitHub Copilot** — May 2025
- **MongoDB Associate Developer** — Nov 2024
- **Microsoft Certified: Azure AI Engineer Associate** — Jun 2024
- **Microsoft Certified: Azure Solution Architect Expert** — Mar 2023
- **Microsoft Certified: DevOps Engineer Expert** — Dec 2022
- **Microsoft Certified: Azure Developer Associate** — Mar 2022
- **Microsoft Certified: Azure Administrator Associate** — Feb 2022
- **Microsoft Certified: Azure Fundamentals** — Jul 2020
- **Microsoft Certified Solutions Developer: App Builder** — Aug 2019
- **Microsoft Certified Solutions Associate: Web Applications** — Mar 2019
- **Microsoft Certified Professional** — Sep 2018

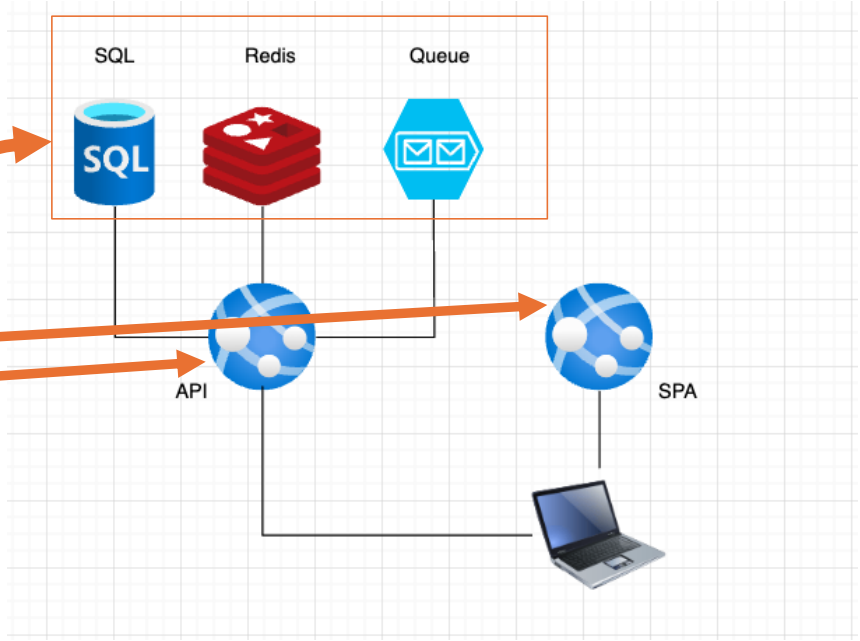

Lettuce hat... That you can eat later

agenda



before aspire(folders)

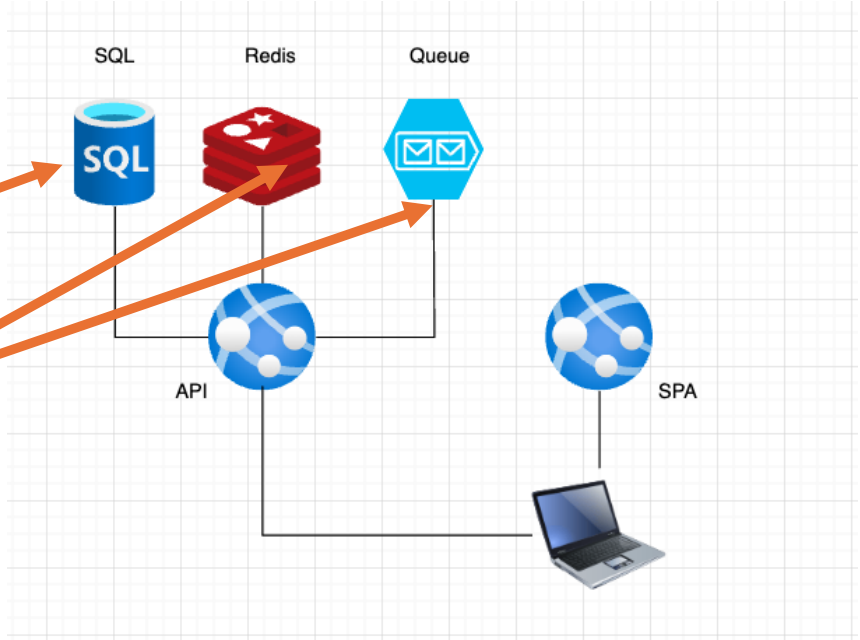




before aspire(w docker)

src-old > infra > docker-compose.yaml > name

```yaml
name: 'demo-infra'
▷Run All Services
services:

  ▷Run Service
  mssql:
    image: mcr.microsoft.com/mssql/server:2022-latest
    container_name: mssql
    environment: ⋯
    ports: ⋯
    volumes:
      - mssql_data:/var/opt/mssql


  ▷Run Service
  azurite:
    image: mcr.microsoft.com/azure-storage/azurite
    container_name: azurite
    ports: ⋯
    volumes: ⋯
    command: "azurite --blobHost 0.0.0.0 --queueHost 0.0.0.0 --tableHost 0.0.0.0 --skipApiVersionCheck"

  ▷Run Service
  redis:
    container_name: redis
    image: redis:latest
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data

volumes:
  mssql_data:
  redis_data:
  azurite_data:
```

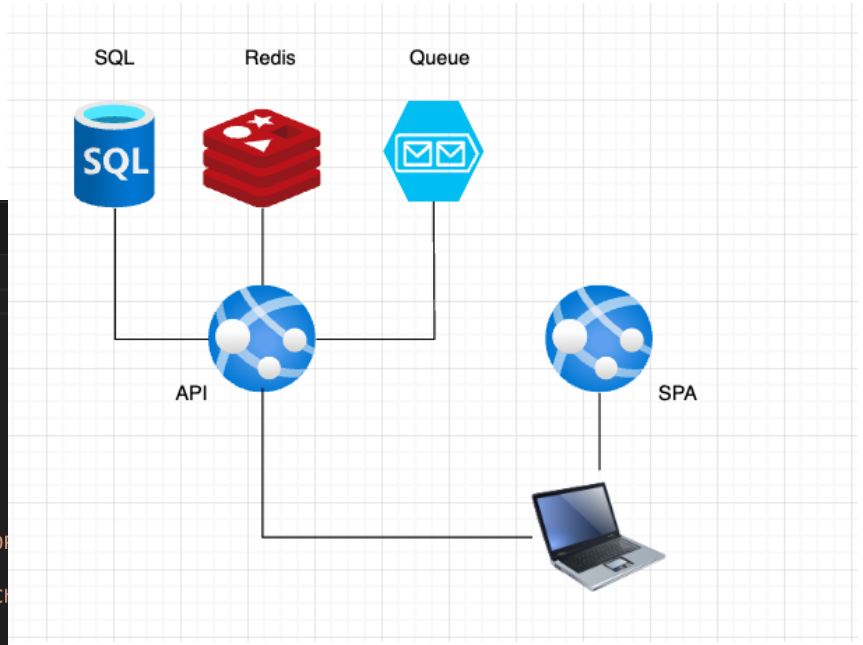


before aspire(conn str)

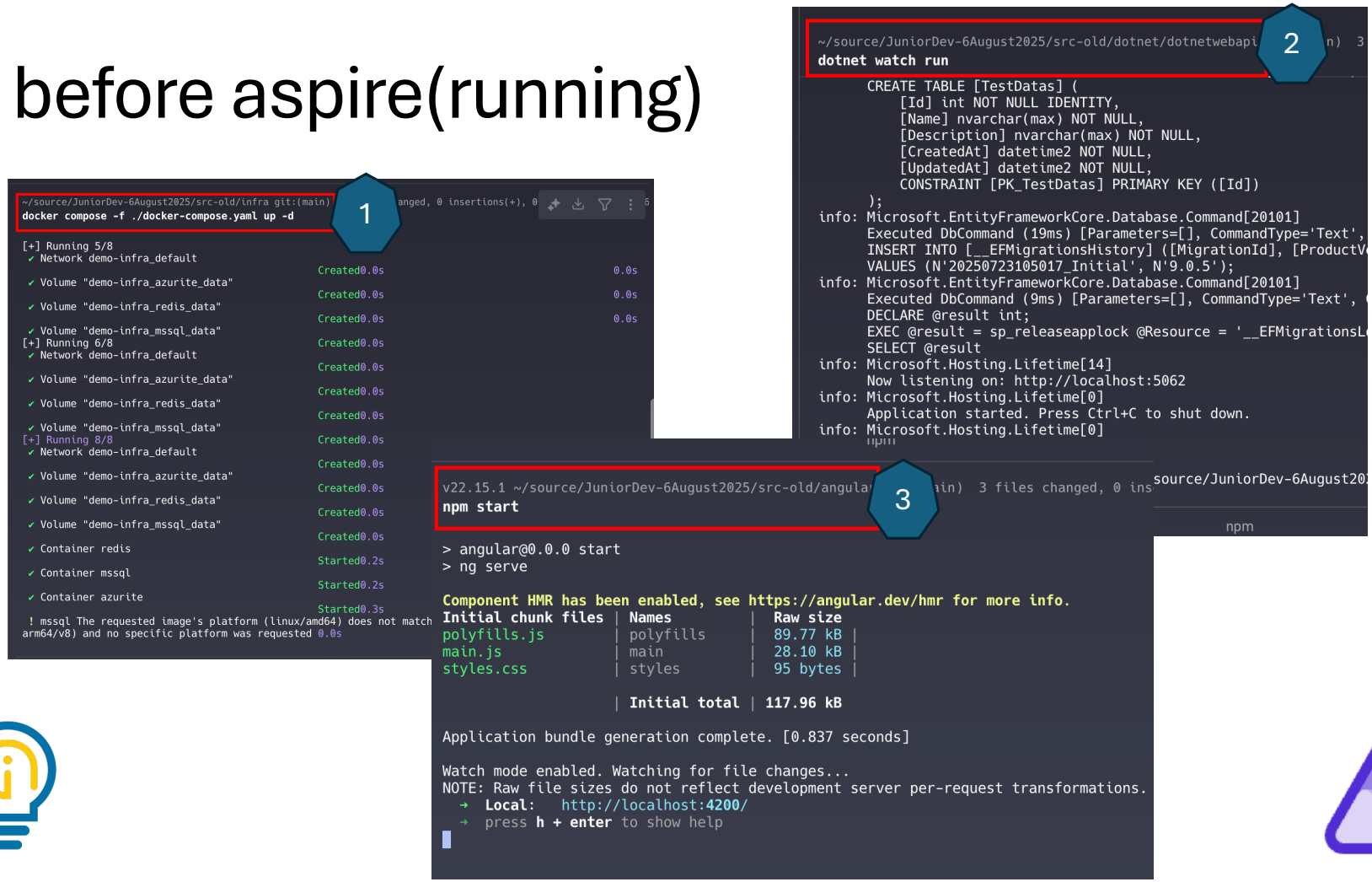
appsettings.json

src-old > dotnet > dotnetwebapi > {} appsettings.json > ...

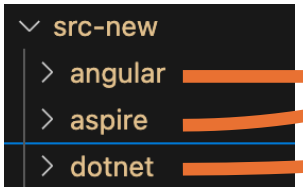
```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "Sql": "Server=localhost,1433;Database=AppDbContext;User Id=sa;Password=d3m0
    "Redis": "localhost:6379",
    "Storage": "SharedAccessSignature=sv=2023-01-03&ss=btqf&srt=sco&spr=https%2C
  }
}
```

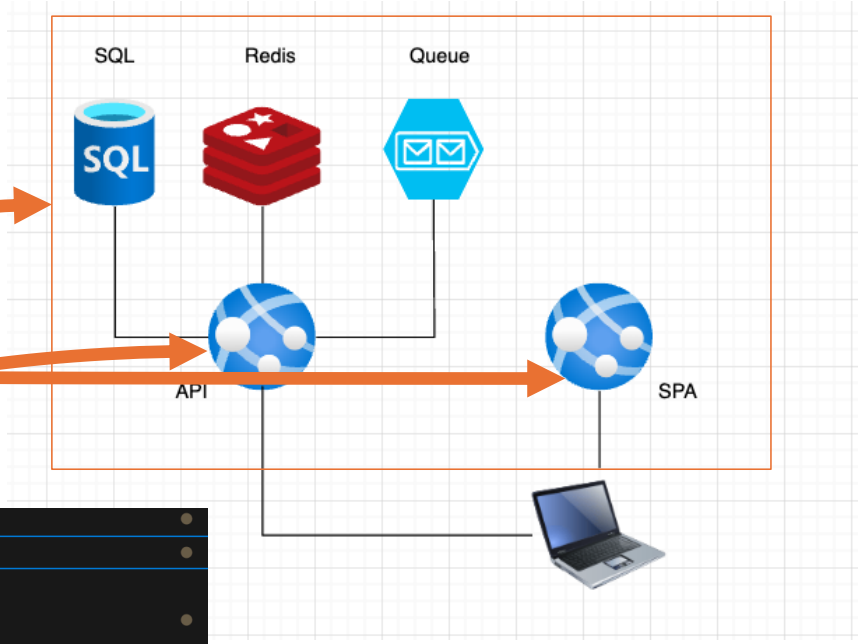


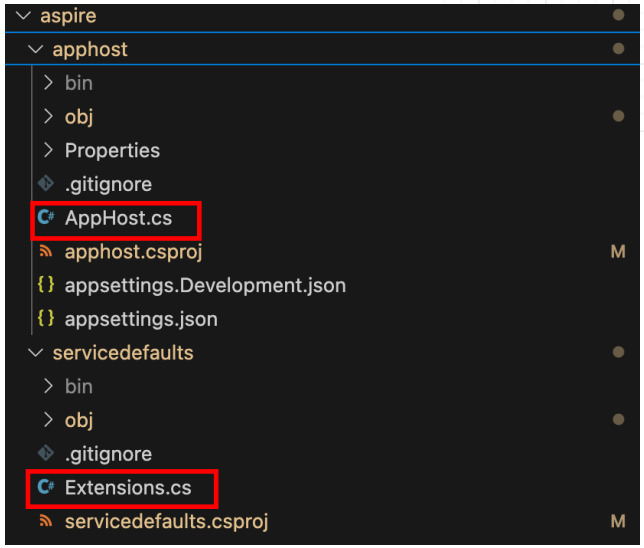
before aspire(running)



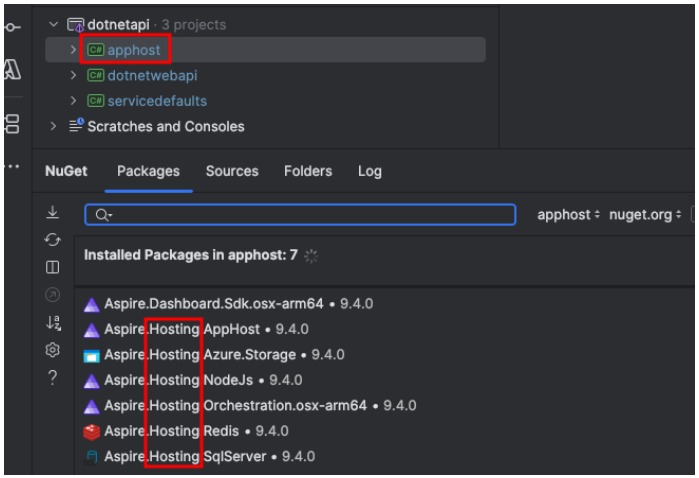
after aspire(folders)

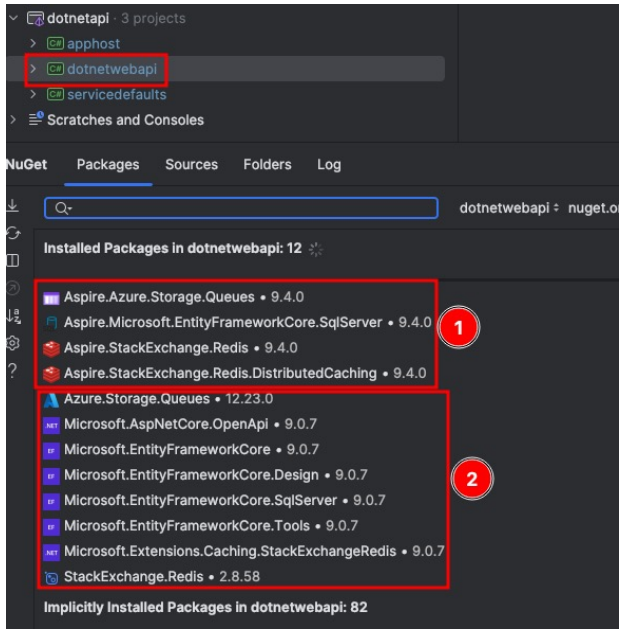






after aspire(nuget)





after aspire(apphost)

AppHost.cs | Program.cs | TestController.cs

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

var sqlserver :IResourceBuilder<SqlServerServerResource> = builder.AddSqlServer( name: "sqlserver");
var appDb :IResourceBuilder<SqlServerDatabaseResource> = sqlserver.AddDatabase( name: "appDb");
var redis :IResourceBuilder<RedisResource> = builder.AddRedis( name: "redis");
var storage :IResourceBuilder<AzureStorageResource> = builder.AddAzureStorage( name: "storage").RunAsEmulator();
var queues :IResourceBuilder<AzureQueueStorageResource> = storage.AddQueues("test-queue");

var api :IResourceBuilder<ProjectResource> = builder.AddProject<dotnetwebapi>( name: "webapi")
    .WithEnvironment( name: "ASPIRED_BOOTSTRAP", "true")
    .WithReference(appDb).WaitFor(appDb)
    .WithReference(queues).WaitFor(queues)
    .WithReference(redis).WaitFor(redis);

var spa :IResourceBuilder<NodeAppResource> = builder.AddNpmApp( name: "spa", workingDirectory: "../../angular")
    .WithReference(api).WaitFor(api)
    .WithUrl("http://localhost:4200")
    .WithHttpEndpoint(env: "PORT");

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

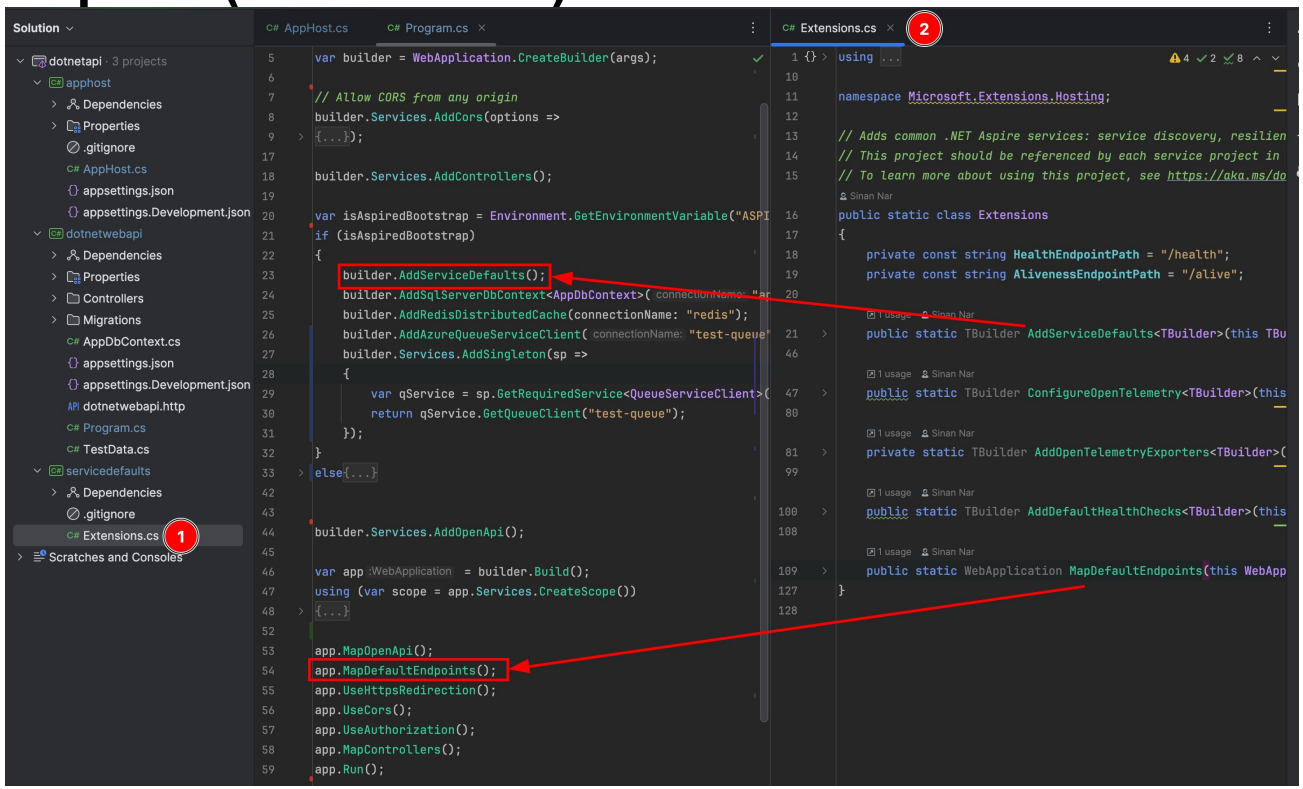
after aspire(conn str)

```
var sqlserver :IResourceBuilder<SqlServerServerResource> = builder.AddSqlServer( name: "sqlserver");
var appDb :IResourceBuilder<SqlServerDatabaseResource> = sqlserver.AddDatabase( name: "appDb");
var redis :IResourceBuilder<RedisResource> = builder.AddRedis( name: "redis");
var storage :IResourceBuilder<AzureStorageResource> = builder.AddAzureStorage( name: "storage").RunAsEmulator();
var queues :IResourceBuilder<AzureQueueStorageResource> = storage.AddQueues("test-queue");

var api :IResourceBuilder<ProjectResource> = builder.AddProject<dotnetwebapi>( name: "webapi")
    .WithEnvironment( name: "ASPIRED_BOOTSTRAP", "true")
    .WithReference(appDb).WaitFor(appDb)
    .WithReference(queues).WaitFor(queues)
    .WithReference(redis).WaitFor(redis);
```

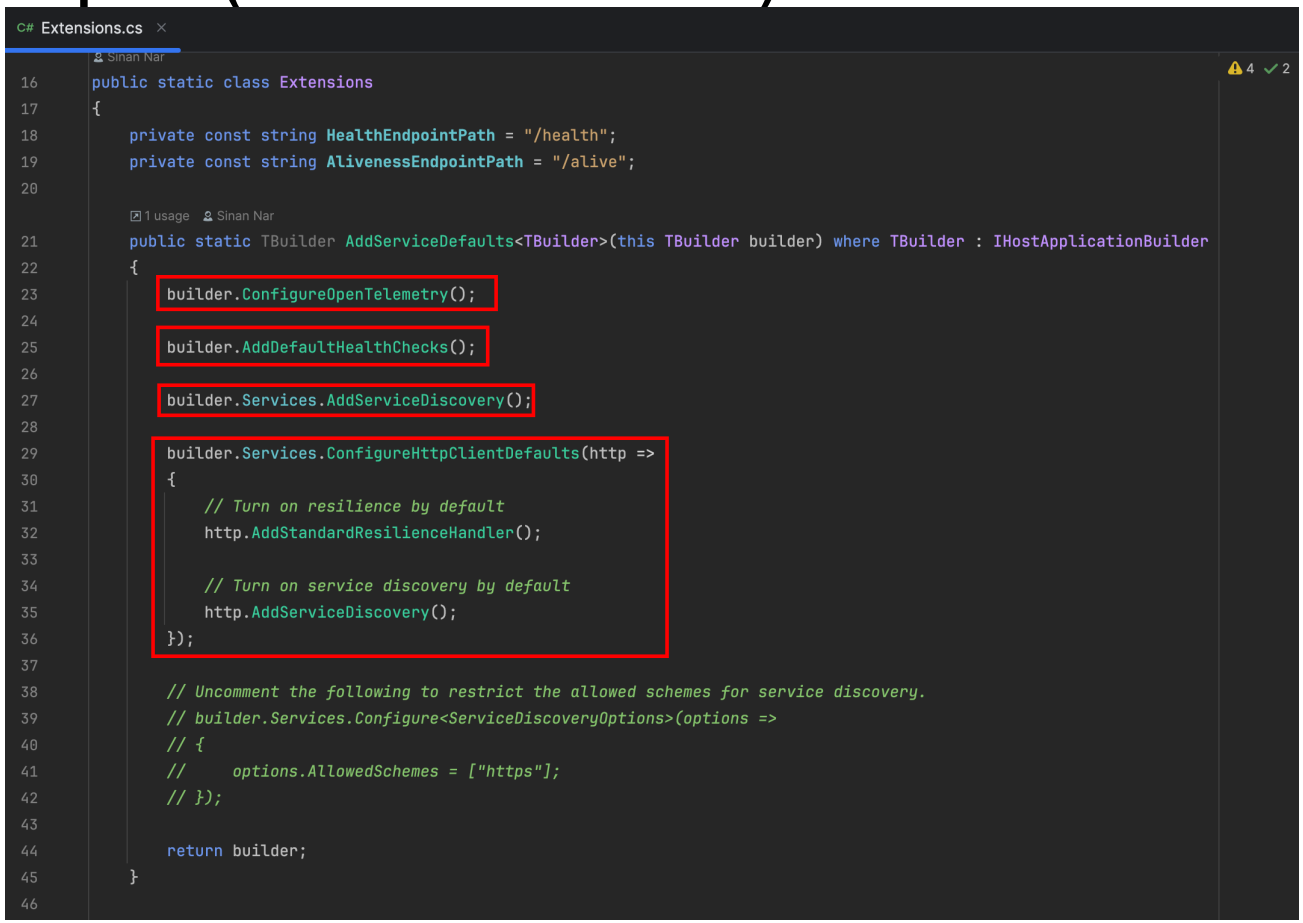
after aspire(conn str cont.)

```
var isAspiredBootstrap = Environment.GetEnvironmentVariable("ASPIRED_BOOTSTRAP") == "true";
if (isAspiredBootstrap)
{
    builder.AddServiceDefaults();
    builder.AddSqlServerDbContext<AppDbContext>( connectionName: "appDb");
    builder.AddRedisDistributedCache(connectionName: "redis");
    builder.AddAzureQueueServiceClient( connectionName: "test-queue");
    builder.Services.AddSingleton(sp =>
    {
        var qService = sp.GetRequiredService<QueueServiceClient>();
        return qService.GetQueueClient("test-queue");
    });
}
else
{
    var sqlConnectionString = builder.Configuration.GetConnectionString( name: "Sql");
    var redisConnectionString = builder.Configuration.GetConnectionString( name: "Redis");
    var storageConnectionString = builder.Configuration.GetConnectionString( name: "Storage");
    builder.Services.AddDbContext<AppDbContext>(options => options.UseSqlServer(sqlConnectionString));
    builder.Services.AddStackExchangeRedisCache(options => { options.Configuration = redisConnectionString; });
    builder.Services.AddScoped(_ => new QueueClient(storageConnectionString, queueName: "test-queue"));
}
```

after aspire(defaults)



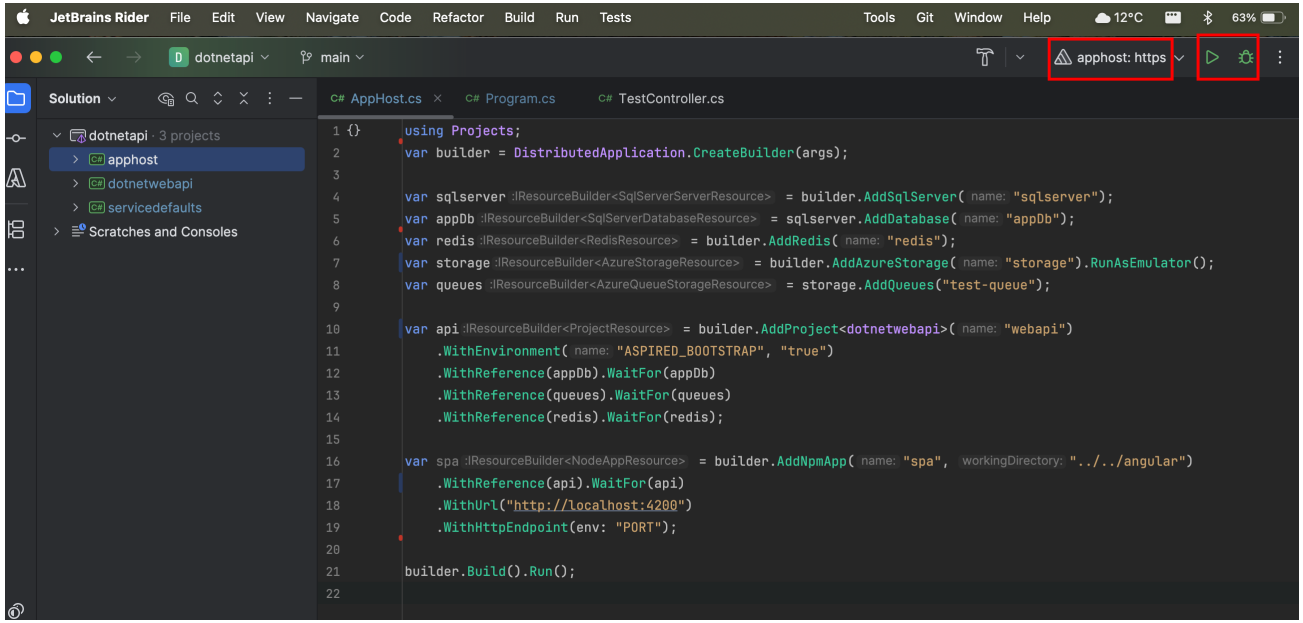
after aspire(defaults cont.)



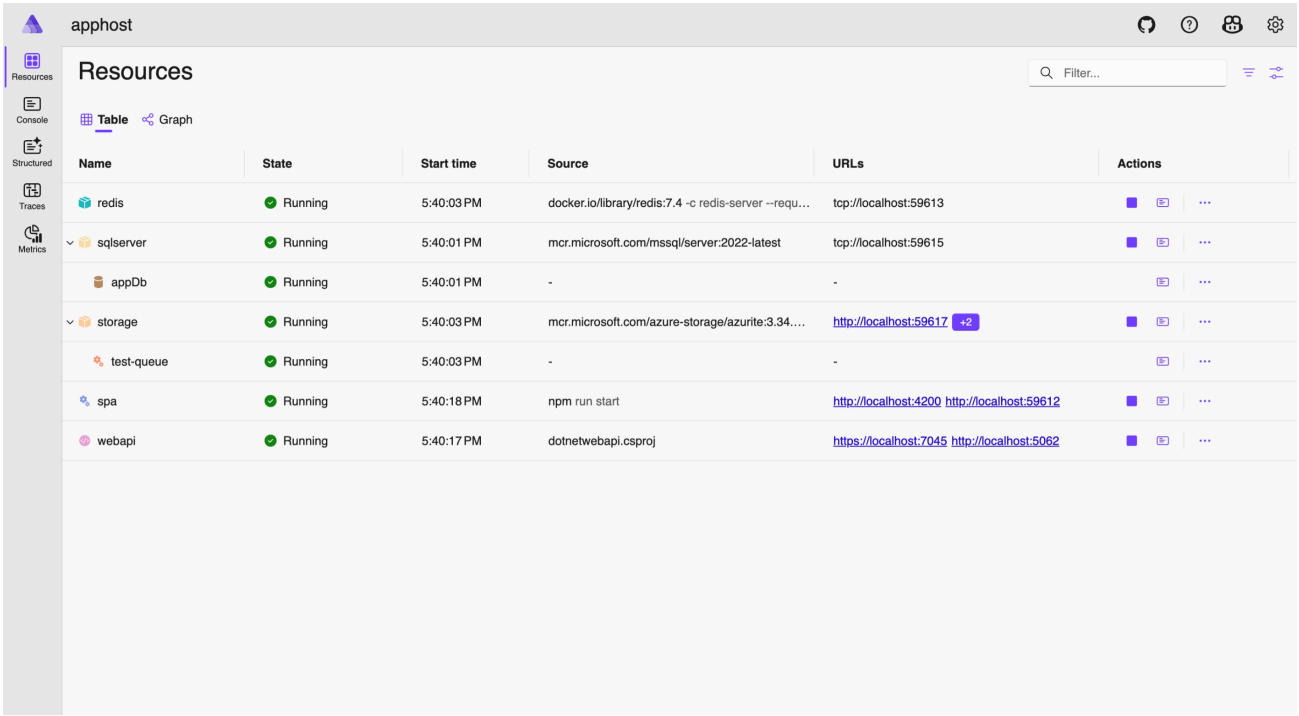
after aspire(defaults cont.)

```
1 usage  Sinan Nar
public static WebApplication MapDefaultEndpoints(this WebApplication app)
{
    // Adding health checks endpoints to applications in non-development environments has security implications.
    // See https://aka.ms/dotnet/aspire/healthchecks for details before enabling these endpoints in non-development environments.
    if (app.Environment.IsDevelopment())
    {
        // All health checks must pass for app to be considered ready to accept traffic after starting
        app.MapHealthChecks( pattern: HealthEndpointPath);

        // Only health checks tagged with the "live" tag must pass for app to be considered alive
        app.MapHealthChecks( pattern: AlivenessEndpointPath, new HealthCheckOptions
        {
            Predicate = r :HealthCheckRegistration => r.Tags.Contains("live")
        });
    }

    return app;
}
```

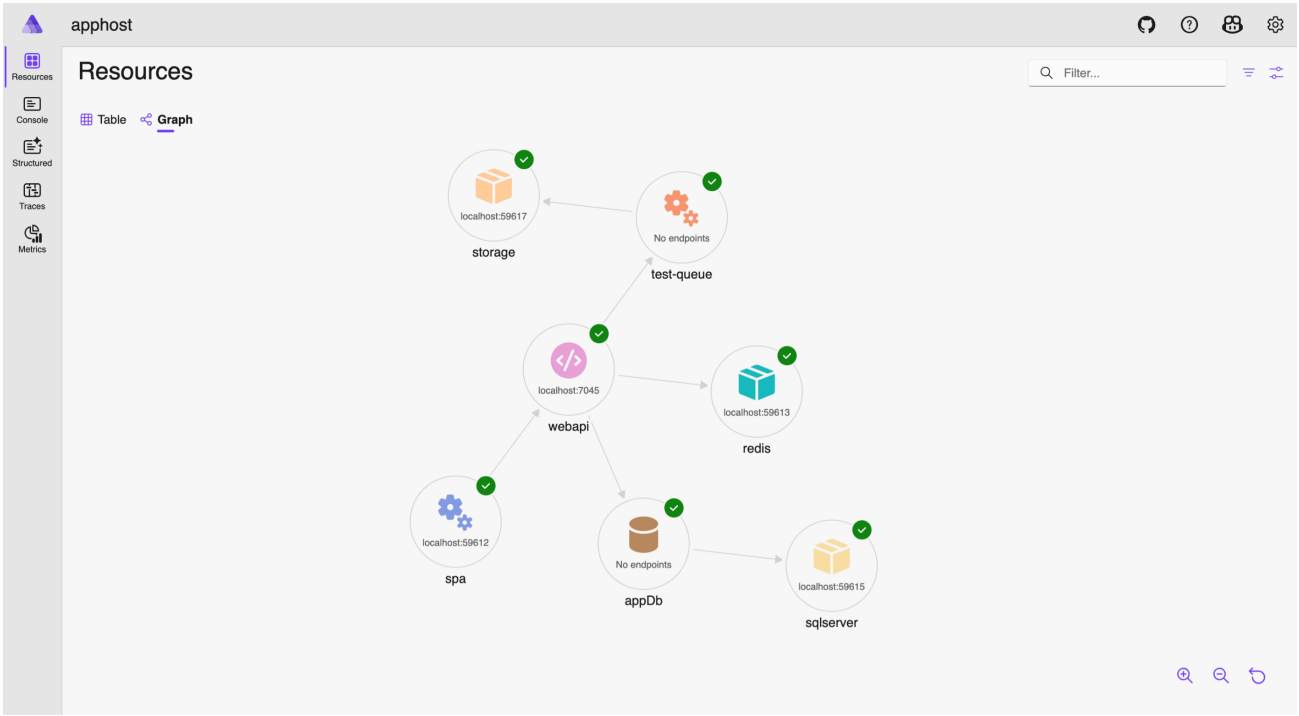
after aspire(running)



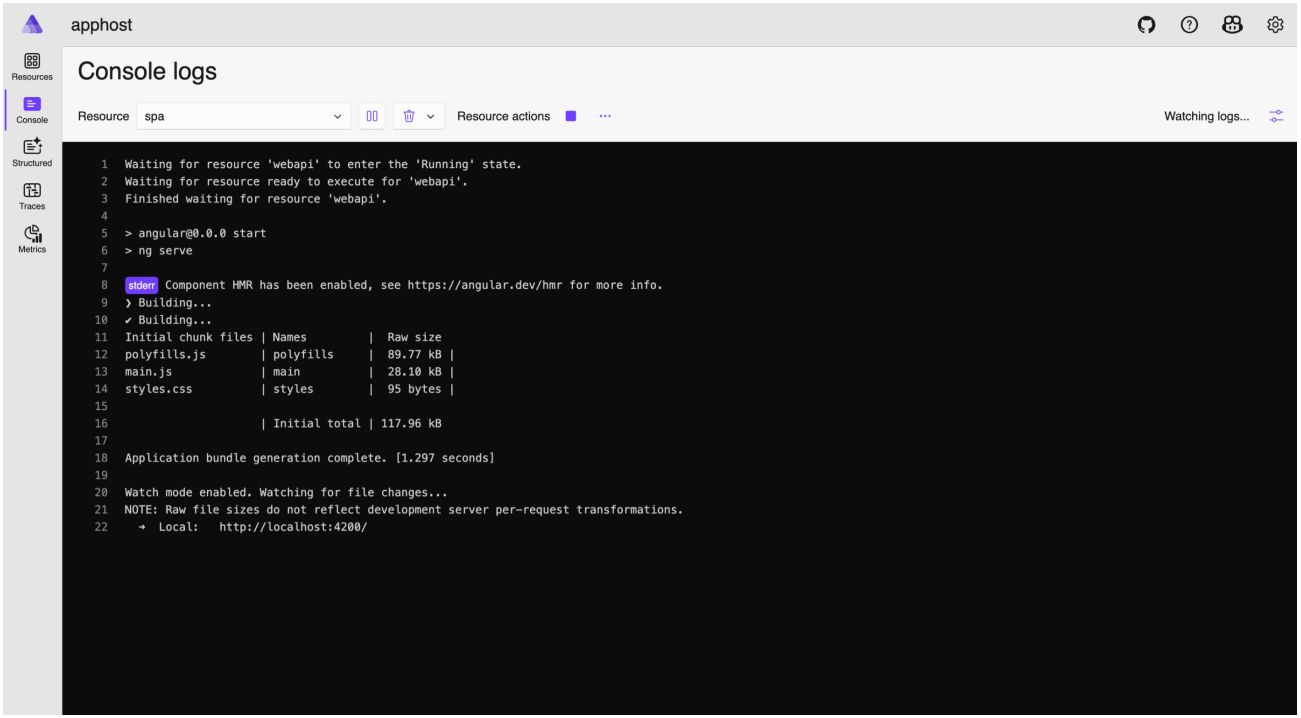
aspire dashboard



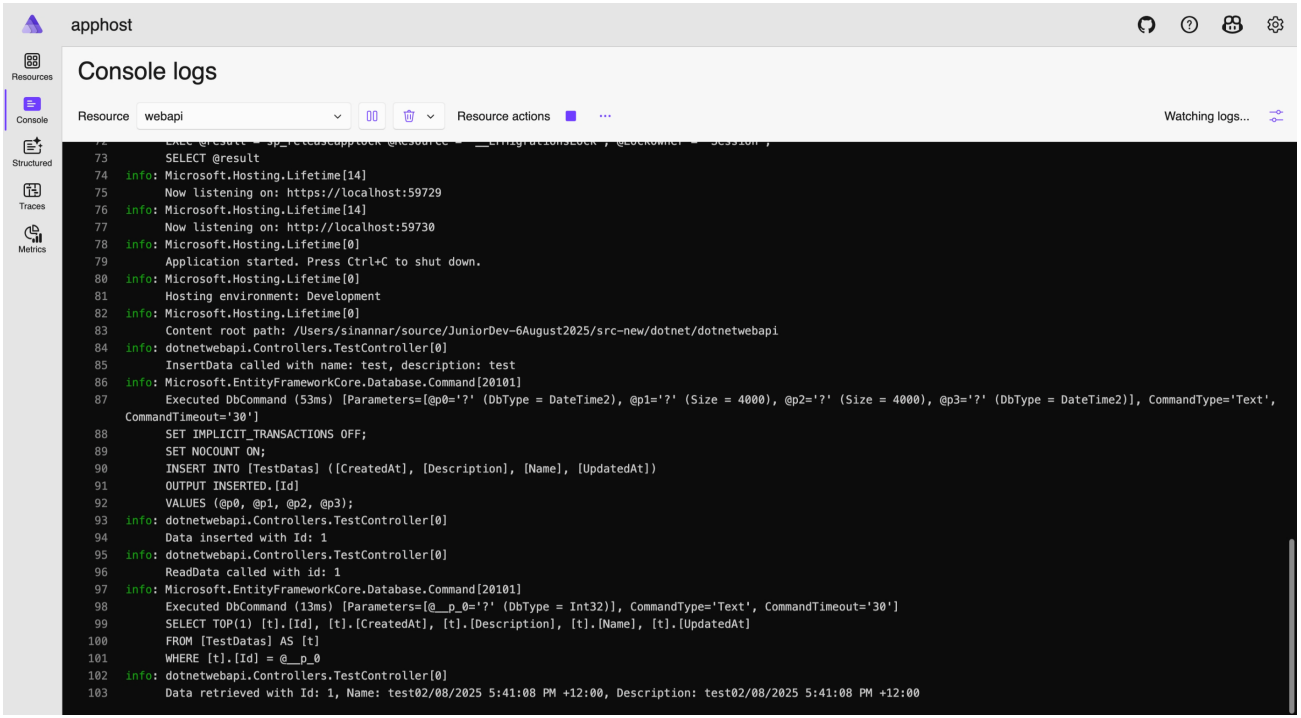
aspire dashboard (cont.)



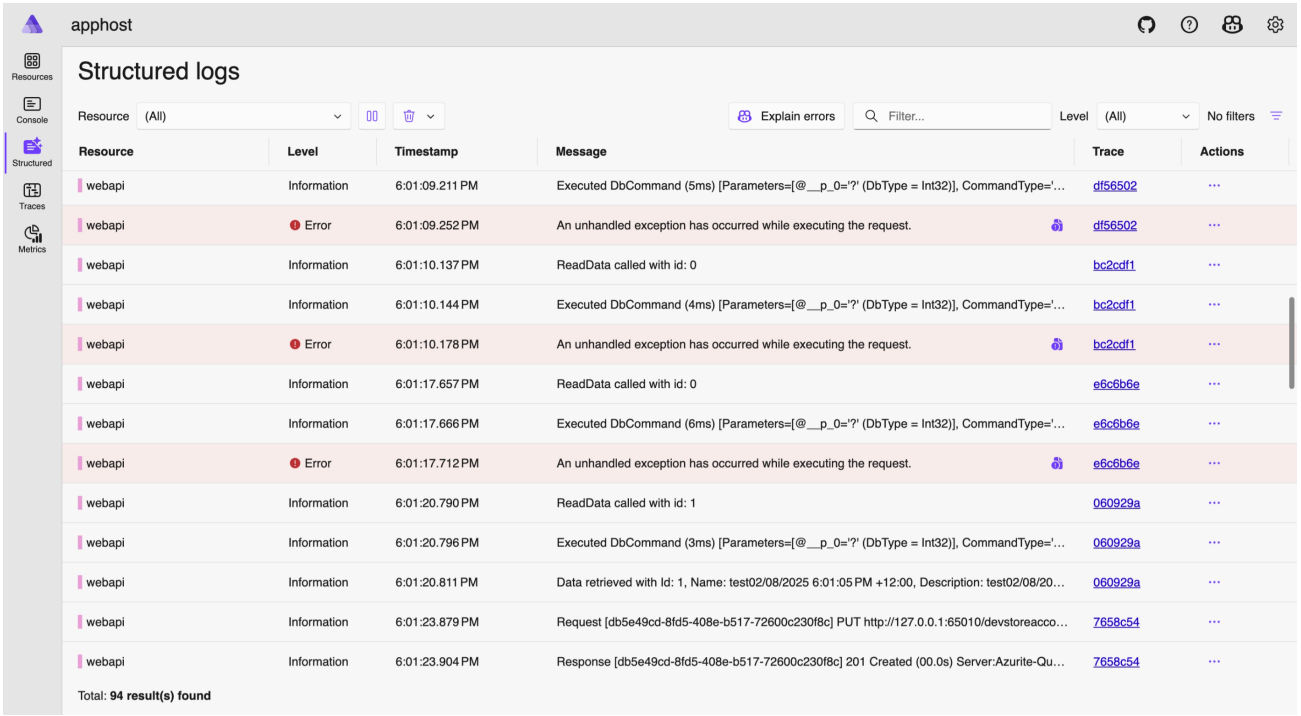
aspire dashboard (cont.)



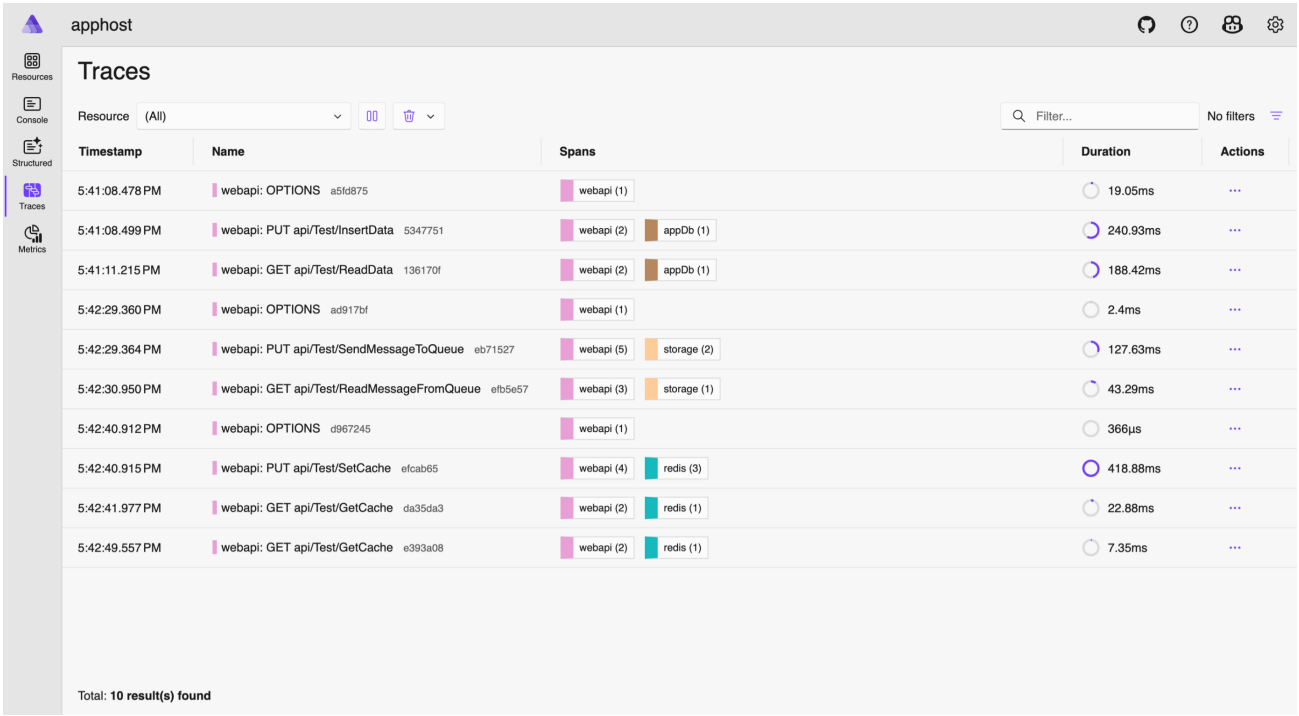
aspire dashboard (cont.)



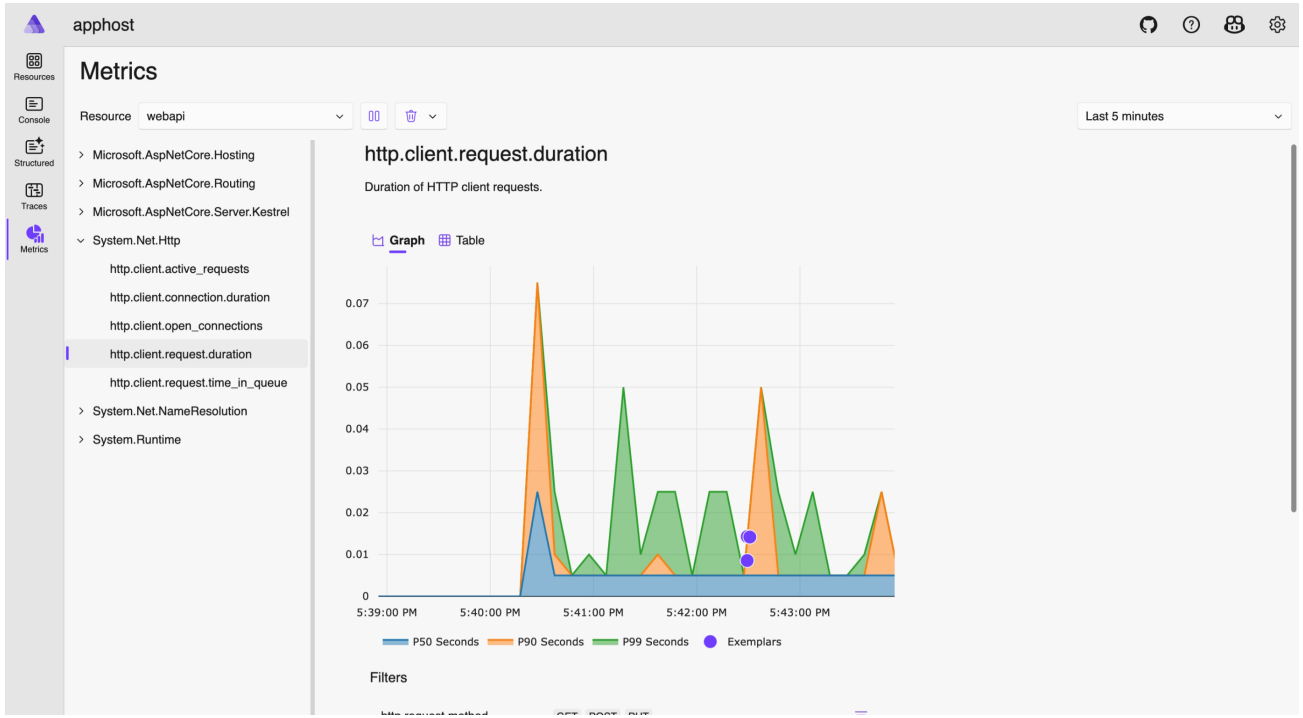
aspire dashboard (cont.)



aspire dashboard (cont.)



aspire dashboard (cont.)



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

var postgres = builder.AddPostgres("postgres")
    .WithPgAdmin()
    .WithLifetime(ContainerLifetime.Persistent);

if (builder.ExecutionContext.IsRunMode)
{
    // Data volumes don't work on ACA for Postgres so only add when running
    postgres.WithDataVolume();
}

var catalogDb = postgres.AddDatabase("catalogdb");

var basketCache = builder.AddRedis("basketcache")
    .WithDataVolume()
    .WithRedisCommander();

var catalogDbManager = builder.AddProject<Projects.AspireShop_CatalogDbManager>("catalogdbmanager")
    .WithReference(catalogDb)
    .WaitFor(catalogDb)
    .WithHttpHealthCheck("/health")
    .WithHttpCommand("/reset-db", "Reset Database", commandOptions: new() { IconName = "DatabaseLightning" });

var catalogService = builder.AddProject<Projects.AspireShop_CatalogService>("catalogservice")
    .WithReference(catalogDb)
    .WaitFor(catalogDbManager)
    .WithHttpHealthCheck("/health");

var basketService = builder.AddProject<Projects.AspireShop_BasketService>("basketservice")
    .WithReference(basketCache)
    .WaitFor(basketCache);

builder.AddProject<Projects.AspireShop_Frontend>("frontend")
    .WithExternalHttpEndpoints()
    .WithUrlForEndpoint("https", url => url.DisplayText = "Online Store (HTTPS)")
    .WithUrlForEndpoint("http", url => url.DisplayText = "Online Store (HTTP)")
    .WithHttpHealthCheck("/health")
    .WithReference(basketService)
    .WithReference(catalogService)
    .WaitFor(catalogService);

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

1
2
3
4

```
builder.Services.AddHttpServiceReference<CatalogServiceClient>("https+http://catalogservice", healthRelativePath: "health");

var isHttps = builder.Configuration["DOTNET_LAUNCH_PROFILE"] == "https";

builder.Services.AddSingleton<BasketServiceClient>()
    .AddGrpcServiceReference<Basket.BasketClient>($"{(isHttps ? "https" : "http")}://basketservice", failureStatus: HealthStatus.Degraded);
```

frontend/program.cs

examples

```
using Azure.Provisioning.Storage;

var builder = DistributedApplication.CreateBuilder(args);

builder.AddAzureContainerAppEnvironment("env");

var storage = builder.AddAzureStorage("storage").RunAsEmulator()
    .ConfigureInfrastructure((infrastructure) =>
    {
        var storageAccount = infrastructure.GetProvisionableResources().OfType<StorageAccount>().FirstOrDefault(r => r.BicepIdentifier == "storage")
            ?? throw new InvalidOperationException($"Could not find configured storage account with name 'storage'");

        // Ensure that public access to blobs is disabled
        storageAccount.AllowBlobPublicAccess = false;
    });
var blobs = storage.AddBlobs("blobs");
var queues = storage.AddQueues("queues");

var functions = builder.AddAzureFunctionsProject<Projects.ImageGallery_Functions>("functions")
                       .WithReference(queues)
                       .WithReference(blobs)
                       .WaitFor(storage)
                       .WithRoleAssignments(storage,
                            // Storage Account Contributor and Storage Blob Data Owner roles are required by the Azure Functions host
                            StorageBuiltInRole.StorageAccountContributor, StorageBuiltInRole.StorageBlobDataOwner,
                            // Queue Data Contributor role is required to send messages to the queue
                            StorageBuiltInRole.StorageQueueDataContributor)
                       .WithHostStorage(storage);

builder.AddProject<Projects.ImageGallery_FrontEnd>("frontend")
       .WithReference(queues)
       .WithReference(blobs)
       .WaitFor(functions)
       .WithExternalHttpEndpoints();

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

examples

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

var weatherApi = builder.AddProject<Projects.AspireJavaScript_MinimalApi>("weatherapi")
    .WithExternalHttpEndpoints();

builder.AddNpmApp("angular", "../AspireJavaScript.Angular")
    .WithReference(weatherApi)
    .WaitFor(weatherApi)
    .WithHttpEndpoint(env: "PORT")
    .WithExternalHttpEndpoints()
    .PublishAsDockerFile();

builder.AddNpmApp("react", "../AspireJavaScript.React")
    .WithReference(weatherApi)
    .WaitFor(weatherApi)
    .WithEnvironment("BROWSER", "none") // Disable opening browser on npm start
    .WithHttpEndpoint(env: "PORT")
    .WithExternalHttpEndpoints()
    .PublishAsDockerFile();

builder.AddNpmApp("vue", "../AspireJavaScript.Vue")
    .WithReference(weatherApi)
    .WaitFor(weatherApi)
    .WithHttpEndpoint(env: "PORT")
    .WithExternalHttpEndpoints()
    .PublishAsDockerFile();

builder.AddNpmApp("reactvite", "../AspireJavaScript.Vite")
    .WithReference(weatherApi)
    .WithEnvironment("BROWSER", "none")
    .WithHttpEndpoint(env: "VITE_PORT")
    .WithExternalHttpEndpoints()
    .PublishAsDockerFile();

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

1
2
3
4

examples

```
using Microsoft.Extensions.Hosting;

var builder = DistributedApplication.CreateBuilder(args);

#pragma warning disable ASPIREHOSTINGPYTHON001 // Type is for evaluation purposes only and is subject to change or removal in future updates. Suppress this diagnostic to proceed.
var pythonapp = builder.AddPythonApp("instrumented-python-app", "../InstrumentedPythonProject", "app.py")
       .WithHttpEndpoint(env: "PORT")
       .WithExternalHttpEndpoints();
#pragma warning restore ASPIREHOSTINGPYTHON001

if (builder.ExecutionContext.IsRunMode && builder.Environment.IsDevelopment())
{
    pythonapp.WithEnvironment("DEBUG", "True");
}

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

examples

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

var apiService = builder.AddProject<Projects.ClientAppsIntegration_ApiService>("apiservice");

if (OperatingSystem.IsWindows())
{
    builder.AddProject<Projects.ClientAppsIntegration_WinForms>("winformsclient")
        .WithReference(apiService)
        .WaitFor(apiService)
        .WithExplicitStart()
        .ExcludeFromManifest();

    builder.AddProject<Projects.ClientAppsIntegration_WPF>("wpfclient")
        .WithReference(apiService)
        .WaitFor(apiService)
        .WithExplicitStart()
        .ExcludeFromManifest();
}

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

examples

```
using Microsoft.Extensions.Hosting;

var builder = DistributedApplication.CreateBuilder(args);

var goVersion = builder.AddParameter("goversion", "1.24.2", publishValueAsDefault: true);

var ginapp = builder.AddDockerfile("ginapp", "../ginapp")
    .WithBuildArg("GO_VERSION", goVersion)
    .WithHttpEndpoint(targetPort: 5555, env: "PORT")
    .WithExternalHttpEndpoints();

if (builder.ExecutionContext.IsPublishMode || builder.Environment.IsProduction())
{
    ginapp
        .WithEnvironment("GIN_MODE", "release")
        // Trust all proxies when running behind a reverse proxy. If deploying to an environment
        // without a reverse proxy that ensures X-Forwarded-* headers are not forwarded from clients,
        // this should be removed.
        .WithEnvironment("TRUSTED_PROXIES", "all");
}

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

examples

```
// Licensed to the .NET Foundation under one or more agreements.
// The .NET Foundation licenses this file to you under the MIT license.

var builder = DistributedApplication.CreateBuilder(args);

var sqlserver = builder.AddSqlServer("sqlserver")
    .WithLifetime(ContainerLifetime.Persistent);

var db1 = sqlserver.AddDatabase("db1");

var migrationService = builder.AddProject<Projects.DatabaseMigrations_MigrationService>("migration")
    .WithReference(db1)
    .WaitFor(db1);

builder.AddProject<Projects.DatabaseMigrations_ApiService>("api")
    .WithReference(db1)
    .WaitForCompletion(migrationService);

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

main

aspire-samples / samples / DatabaseMigrations / DatabaseMigrations.MigrationService / DatabaseMigrations.MigrationService.csproj

DamianEdwards Updated to 9.4 (#873)

Code | Blame | 21 lines (18 loc) · 916 Bytes

```
<Project Sdk="Microsoft.NET.Sdk.Worker">

  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
    <!-- Make build happy with generated migrations -->
    <NoWarn>$(NoWarn);IDE0161;IDE0005</NoWarn>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="8.0.13">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
    </PackageReference>
    <PackageReference Include="Aspire.Microsoft.EntityFrameworkCore.SqlServer" Version="9.4.0" />
    <ProjectReference Include="..\DatabaseMigrations.ApiModel\DatabaseMigrations.ApiModel.csproj" />
    <ProjectReference Include="..\DatabaseMigrations.ServiceDefaults\DatabaseMigrations.ServiceDefaults.csproj" />
  </ItemGroup>

</Project>
```

examples

```
using MetricsApp.AppHost.OpenTelemetryCollector;

var builder = DistributedApplication.CreateBuilder(args);

var prometheus = builder.AddContainer("prometheus", "prom/prometheus", "v3.2.1")
       .WithBindMount("../prometheus", "/etc/prometheus", isReadOnly: true)
       .WithArgs("--web.enable-otlp-receiver", "--config.file=/etc/prometheus/prometheus.yml")
       .WithHttpEndpoint(targetPort: 9090, name: "http");

var grafana = builder.AddContainer("grafana", "grafana/grafana")
                     .WithBindMount("../grafana/config", "/etc/grafana", isReadOnly: true)
                     .WithBindMount("../grafana/dashboards", "/var/lib/grafana/dashboards", isReadOnly: true)
                     .WithEnvironment("PROMETHEUS_ENDPOINT", prometheus.GetEndpoint("http"))
                     .WithHttpEndpoint(targetPort: 3000, name: "http");

builder.AddOpenTelemetryCollector("otelcollector", "../otelcollector/config.yaml")
       .WithEnvironment("PROMETHEUS_ENDPOINT", $"{prometheus.GetEndpoint("http")}/api/v1/otlp");

builder.AddProject<Projects.MetricsApp>("app")
       .WithEnvironment("GRAFANA_URL", grafana.GetEndpoint("http"));

using var app = builder.Build();

await app.RunAsync();
```

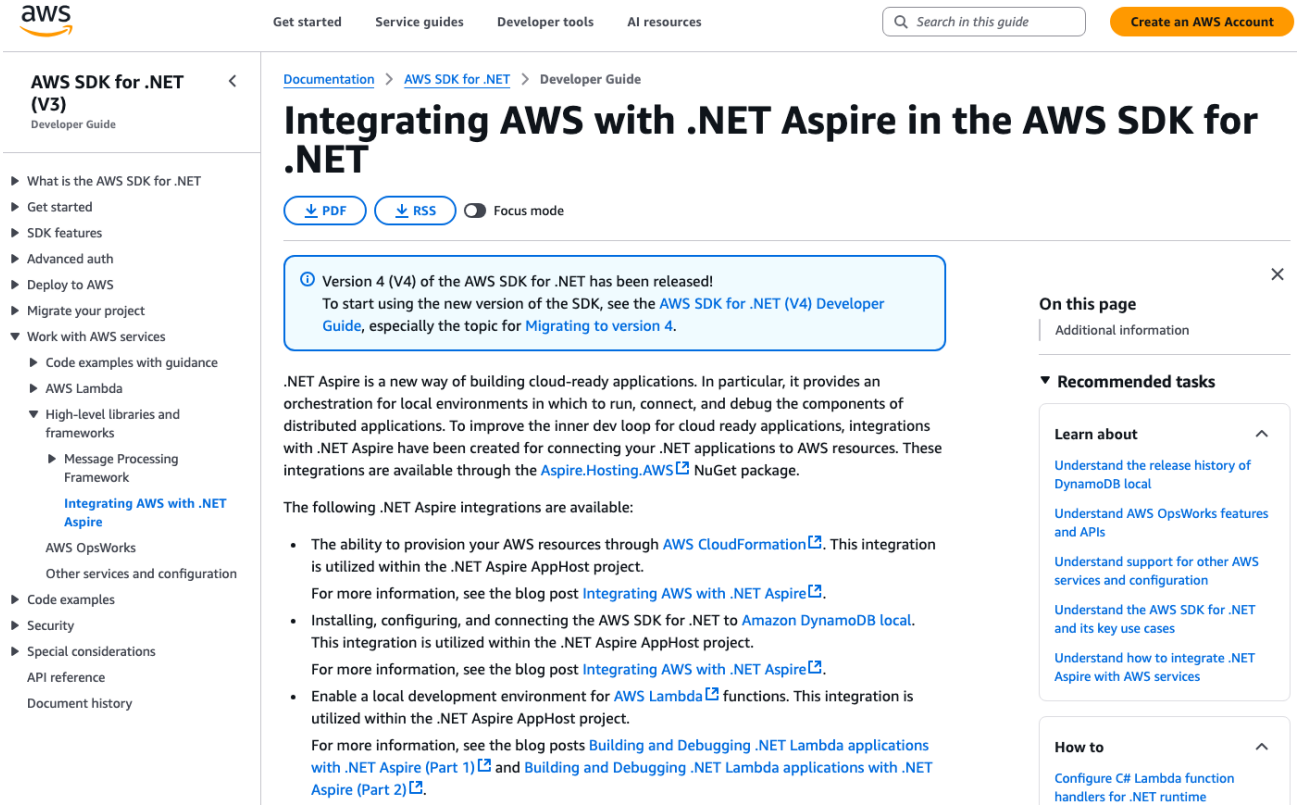
examples

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

var sqlserver = builder.AddSqlServer("sqlserver")
    .WithDataVolume()
    .WithLifetime(ContainerLifetime.Persistent);

var sqlDatabase = sqlserver.AddDatabase("sqldb");

var postgresServer = builder.AddPostgres("postgresserver")
    .WithDataVolume()
    .WithLifetime(ContainerLifetime.Persistent);

var postgresDatabase = postgresServer.AddDatabase("postgres");

var blobs = builder.AddAzureStorage("Storage")
    // Use the Azurite storage emulator for local development
    .RunAsEmulator(emulator => emulator.WithDataVolume())
    .AddBlobs("BlobConnection");

builder.AddProject<Projects.VolumeMount_BlazorWeb>("blazorweb")
    .WithReference(sqlDatabase)
    .WaitFor(sqlDatabase)
    .WithReference(postgresDatabase)
    .WaitFor(postgresDatabase)
    .WithReference(blobs)
    .WaitFor(blobs);

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

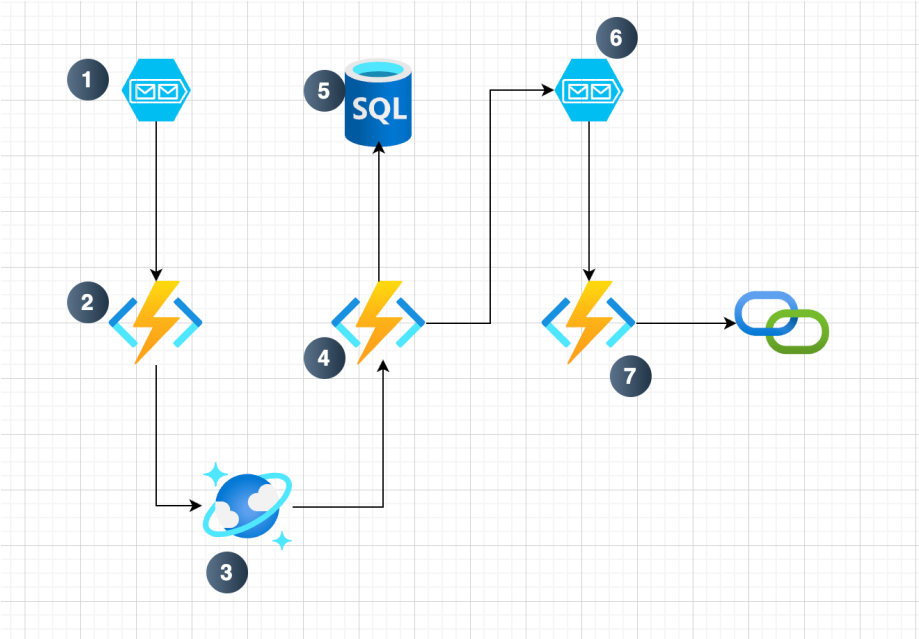
aspire integrations

- Integrations
  - **Overview**
  - Tutorials
  - Apache Kafka
  - Azure
  - Elasticsearch
  - Entity Framework Core
  - GitHub Models
  - Keycloak (Preview)
  - Milvus
  - MongoDB
  - MySQL
  - NATS
  - Orleans
  - PostgreSQL
  - Qdrant
  - RabbitMQ service broker
  - Redis
  - Seq
  - SQL Server
  - YARP (Yet Another Reverse Proxy)
  - Community Toolkit
  - Aspire.Hosting API reference
- Azure
  - Overview
  - Customize Azure resources
  - Local Azure provisioning
  - Configure Azure Container Apps environments
  - User-assigned managed identity
  - Manage role assignments
  - Azure App Configuration
  - Azure AI Foundry (Preview)
  - Azure AI Inference (Preview)
  - Azure AI Search
  - Azure Cache for Redis
  - Azure Container Registry (Preview)
  - Azure Cosmos DB
  - Azure Event Hubs
  - Azure Functions (Preview)
  - Azure Key Vault
  - Azure PostgreSQL
  - Azure OpenAI (Preview)
  - Azure SignalR Service
  - Azure Service Bus
  - Azure SQL Database
  - Azure Storage
  - Azure Web PubSub
  - Aspire.Hosting.Azure API reference
  - Aspire.Azure API reference
- Community Toolkit
  - Overview
  - Bun apps
  - Dapr
  - Deno apps
  - Go apps
  - Java/Spring
  - Node.js extensions
  - Python extensions
  - Ollama
  - Meilisearch
  - Rust apps
  - SQL Database Projects
  - Data API Builder
  - EventStore
  - SQLite
  - SQL Server Extensions
  - MongoDB Extensions
  - Redis Extensions
  - PostgreSQL Extensions
  - MySql Extensions
  - RavenDB

aspire integrations(cont)



into to .net aspire blog post



thank you for listening

- github.com/dotnet/aspire
- github.com/dotnet/aspire-samples/tree/main/samples
  - AspireShop/AspireShop.AppHost/AppHost.cs
  - AspireWithAzureFunctions/ImageGallery.AppHost/AppHost.cs
  - AspireWithJavaScript/AspireJavaScript.AppHost/AppHost.cs
  - AspireWithPython/AspireWithPython.AppHost/AppHost.cs
  - ClientAppsIntegration/ClientAppsIntegration.AppHost/AppHost.cs
  - ContainerBuild/ContainerBuild.AppHost/AppHost.cs
  - DatabaseMigrations/DatabaseMigrations.AppHost/AppHost.cs
- learn.microsoft.com/en-us/dotnet/aspire/
- github.com/**sinannar**/JuniorDev-6August2025

ARINCO