.NET Aspire overview

Article • 01/26/2025

.NET Aspire is a set of tools, templates, and packages for building observable, production ready apps. .NET Aspire is delivered through a collection of NuGet packages that bootstrap or improve specific challenges with modern app development. Today's apps generally consume a large number of services, such as databases, messaging, and caching, many of which are supported via .NET Aspire Integrations. For information on support, see the .NET Aspire Support Policy.

Why .NET Aspire?

.NET Aspire improves the experience of building apps that have a variety of projects and resources. With dev-time productivity enhancements that emulate deployed scenarios, you can quickly develop interconnected apps. Designed for flexibility, .NET Aspire allows you to replace or extend parts with your preferred tools and workflows. Key features include:

- **Dev-Time Orchestration**: .NET Aspire provides features for running and connecting multi-project applications, container resources, and other dependencies for local development environments.
- **Integrations**: .NET Aspire integrations are NuGet packages for commonly used services, such as Redis or Postgres, with standardized interfaces ensuring they connect consistently and seamlessly with your app.
- **Tooling**: .NET Aspire comes with project templates and tooling experiences for Visual Studio, Visual Studio Code, and the .NET CLI to help you create and interact with .NET Aspire projects.

Dev-time orchestration

In .NET Aspire, "orchestration" primarily focuses on enhancing the *local development* experience by simplifying the management of your app's configuration and interconnections. It's important to note that .NET Aspire's orchestration isn't intended to replace the robust systems used in production environments, such as Kubernetes. Instead, it's a set of abstractions that streamline the setup of service discovery, environment variables, and container configurations, eliminating the need to deal with low-level implementation details. With .NET Aspire, your code has a consistent bootstrapping experience on any dev machine without the need for complex manual steps, making it easier to manage during the development phase.

.NET Aspire orchestration assists with the following concerns:

- **App composition**: Specify the .NET projects, containers, executables, and cloud resources that make up the application.
- **Service discovery and connection string management**: The app host injects the right connection strings, network configurations, and service discovery information to simplify the developer experience.

For example, using .NET Aspire, the following code creates a local Redis container resource, waits for it to become available, and then configures the appropriate connection string in the `"frontend"` project with a few helper method calls:

C#

```
// Create a distributed application builder given the command line arguments.
var builder = DistributedApplication.CreateBuilder(args);

// Add a Redis server to the application.
var cache = builder.AddRedis("cache");

// Add the frontend project to the application and configure it to use the
// Redis server, defined as a referenced dependency.
builder.AddProject<Projects.MyFrontend>("frontend")
       .WithReference(cache)
       .WaitFor(cache);
```

For more information, see .NET Aspire orchestration overview.

> **Important**
>
> The call to AddRedis creates a new Redis container in your local dev environment. If you'd rather use an existing Redis instance, you can use the AddConnectionString method to reference an existing connection string. For more information, see Reference existing resources.

.NET Aspire integrations

.NET Aspire integrations are NuGet packages designed to simplify connections to popular services and platforms, such as Redis or PostgreSQL. .NET Aspire integrations handle cloud resource setup and interaction for you through standardized patterns, such as adding health checks and telemetry. Integrations are two-fold - "hosting" integrations represents the service you're connecting to, and "client" integrations represents the client or consumer of that service. In other words, for many hosting packages there's a corresponding client package that handles the service connection within your code.

Each integration is designed to work with the .NET Aspire app host, and their configurations are injected automatically by referencing named resources. In other words, if *Example.ServiceFoo* references *Example.ServiceBar*, *Example.ServiceFoo* inherits the integration's required configurations to allow them to communicate with each other automatically.

For example, consider the following code using the .NET Aspire Service Bus integration:

C#

```
builder.AddAzureServiceBusClient("servicebus");
```

The AddAzureServiceBusClient method handles the following concerns:

- Registers a ServiceBusClient as a singleton in the DI container for connecting to Azure Service Bus.
- Applies ServiceBusClient configurations either inline through code or through configuration.
- Enables corresponding health checks, logging, and telemetry specific to the Azure Service Bus usage.

A full list of available integrations is detailed on the .NET Aspire integrations overview page.

Project templates and tooling

.NET Aspire provides a set of project templates and tooling experiences for Visual Studio, Visual Studio Code, and the .NET CLI. These templates are designed to help you create and interact with .NET Aspire projects, or add .NET Aspire into your existing codebase. The templates include a set of opinionated defaults to help you get started quickly - for example, it has boilerplate code for turning on health checks and logging in .NET apps. These defaults are fully customizable, so you can edit and adapt them to suit your needs.

.NET Aspire templates also include boilerplate extension methods that handle common service configurations for you:

C#

```
builder.AddServiceDefaults();
```

For more information on what `AddServiceDefaults` does, see .NET Aspire service defaults.

When added to your *Program.cs* file, the preceding code handles the following concerns:

- **OpenTelemetry**: Sets up formatted logging, runtime metrics, built-in meters, and tracing for ASP.NET Core, gRPC, and HTTP. For more information, see .NET Aspire telemetry.
- **Default health checks**: Adds default health check endpoints that tools can query to monitor your app. For more information, see .NET app health checks in C#.
- **Service discovery**: Enables service discovery for the app and configures HttpClient accordingly.

Next steps

Quickstart: Build your first .NET Aspire project