Unlock the Power of .NET in the Cloud: Journey into the Future with .NET Aspire

Orestis Meikopoulos

https://linkedin.com/in/ormikopo

Agenda

- Introduction, Setup Prerequisites and Installation Steps
- Key Concepts
- Inner-Loop Networking
- Service Discovery
- Service Defaults
- Deploy to Azure

Introduction, Setup Prerequisites and Installation Steps

- **Purpose**: Build observable, production-ready, distributed applications.
- **Delivery**: Via a collection of NuGet packages.
- **Focus**: Cloud-native, distributed applications using microservices architecture.
- **Target**: Enhances building and managing .NET cloud-native apps.

Introduction, Setup Prerequisites and Installation Steps

- **Requirements:**
  - .NET 8.0
  - .NET Aspire workload
  - Docker Desktop or Podman for container support
  - IDE or code editor (e.g., Visual Studio 2022 Preview 17.10+ or Visual Studio Code)
- **Installation via .NET CLI**:
  - Update: `dotnet workload update`
  - Install: `dotnet workload install aspire`
  - Verify installation: `dotnet workload list`
- **Creating Projects**:
  - List templates: `dotnet new list aspire`
  - Create a basic project: `dotnet new aspire`
  - Create a project with UI and API: `dotnet new aspire-starter`

Key Concepts - Terminology

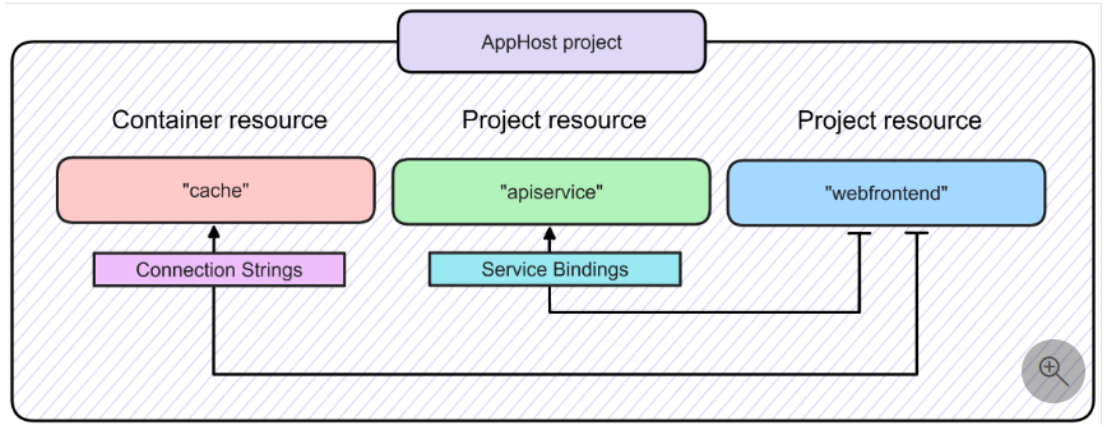
- **App Model**: A collection of interconnected resources making up your distributed application.
- **App Host/Orchestrator Project**: Orchestrates the app model, typically named with the *.AppHost suffix.
- **Resource**: Elements like projects, containers, executables, or services (e.g., databases, caches).
- **Reference**: Defines connections between resources as dependencies.

Key Concepts - Defining the App Model

- **Purpose**: Outline the resources in your app and their relationships.
- **Implementation**: Utilize `IDistributedApplicationBuilder` to configure resources and dependencies.
- **Example**: Use `AddProject` or `AddContainer` to include resources in your app model.

Key Concepts – App Host Project

- **Purpose**: Handles running all the projects that are part of the .NET Aspire aplication.
- **Example**: The current image describes an application with two projects and a Redis cache.



Key Concepts – Resource Types

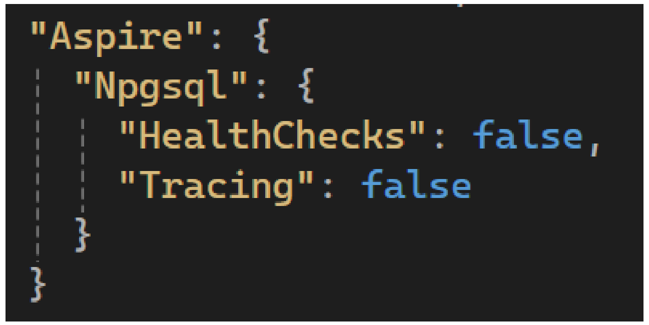
- **Resource Management**: .NET Aspire apps are made up of a set of resources:
  - `AddProject`: A .NET project, for example an ASP.NET Core web app. Project resources are .NET projects that are part of the app model
  - `AddContainer`: A container image, such as a Docker image.
  - `AddExecutable`: An executable file.
- **Example**: To add a project to the app model:
  - `var aspireDemoApp = builder.AddProject<Projects.GlobalAzure_NetAspire_Server>("aspiredemoapp")`

Key Concepts – Reference Management

- **Define Dependencies**: Use WithReference to establish dependencies among resources. For example:
    - `var customerDb = builder.AddSqlServer("aspiredemosqlserver");`
    - `builder.AddProject<Projects.GlobalAzure_NetAspire_Server>("aspiredemoapp").WithReference(customerDb);`:
        - ConnectionStrings__ aspiredemosqlserver="localhost:1433"
- **Connection Strings and Service Discovery**: Inject environment variables for dependencies and service discovery. For example:
    - `WithReference(aspiredemoapi)`:
        - services__aspiredemoapi__0=http://_http.localhost:5000
        - services__aspiredemoapi__1="http://localhost:5000"

Key Concepts – Components

- **Purpose**: Enhance integration with services like Redis and PostgreSQL through curated NuGet packages.
- **Resiliency**: Automatically integrated features such as connection retries and timeouts to maintain functionality during failures.
- **Setup**: Through JSON configuration files or directly via code using delegates.
- **Example**: Configuring PostgreSQL components using appsettings.json.



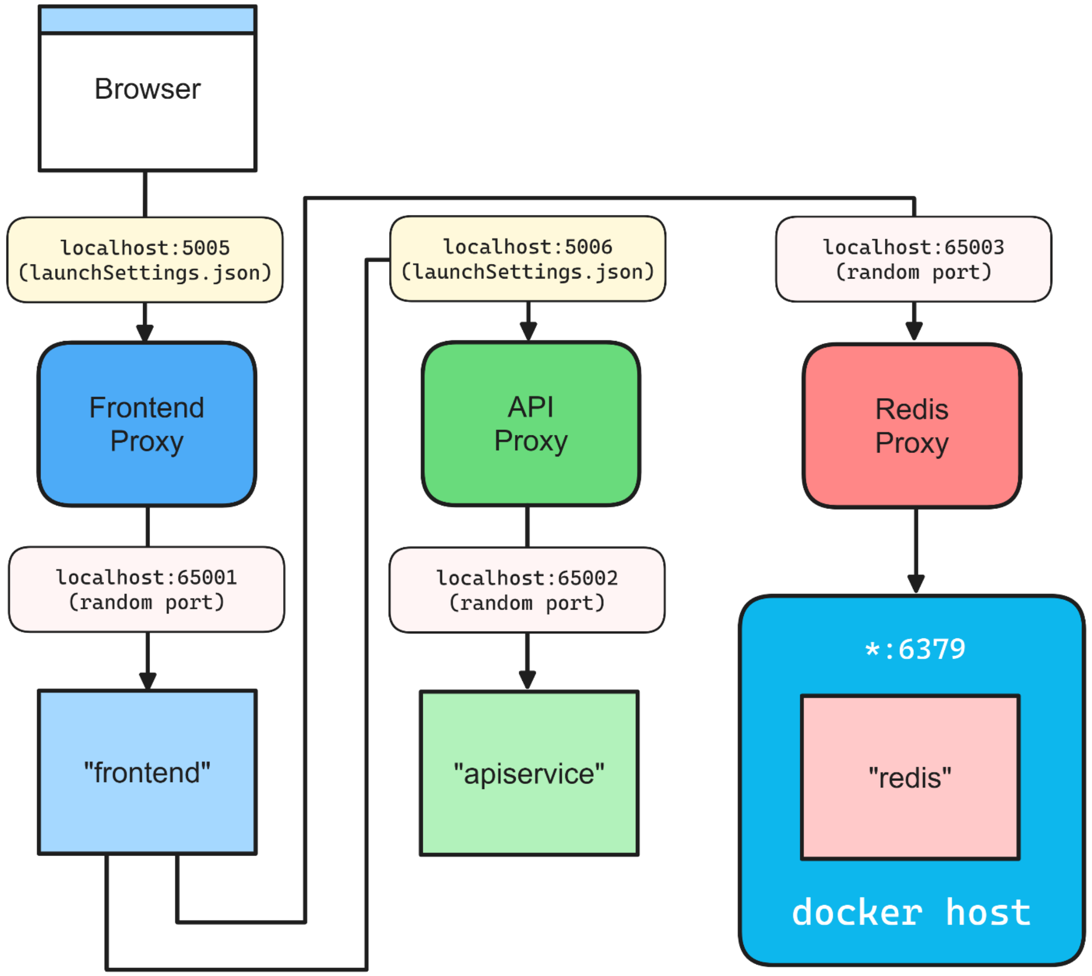
```
"Aspire": {
  "Npgsql": {
    "HealthChecks": false,
    "Tracing": false
  }
}
```

Audience Question

Is .NET Aspire reminding you of some other technology?

Inner Loop Networking – Service Bindings

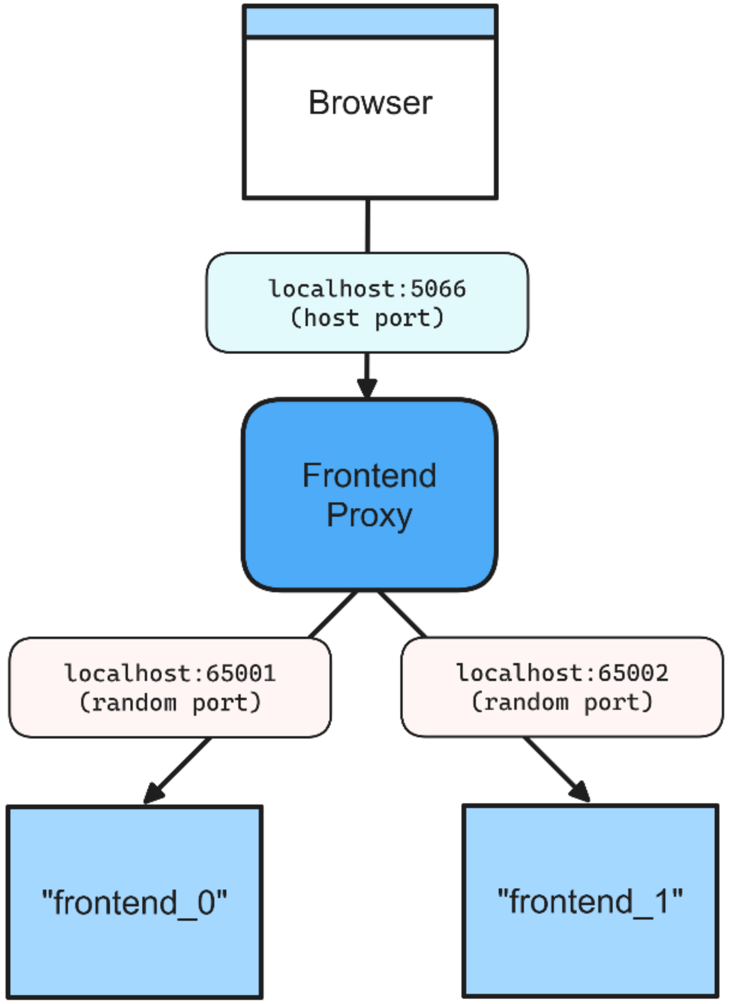
- **Role**: Connect your app to external services required (databases, queues, APIs).
- **Types**:
  - Implicit: Automatically created from launch profiles.
  - Explicit: Manually created using WithEndpoint.
- **Proxy Function**: Handles routing and load balancing, launched for each service binding.



Browser
localhost:5005 (launchSettings.json)
Frontend Proxy
localhost:65001 (random port)
"frontend"
localhost:5006 (launchSettings.json)
API Proxy
localhost:65002 (random port)
"apiservice"
localhost:65003 (random port)
Redis Proxy
*:6379
"redis"
docker host

Inner Loop Networking – Ports and Proxies

- **Configuration**: Host port is assigned to a proxy, which manages connections to services.
- **Example**: Using `AddProject` with `WithHttpEndpoint` and `WithReplicas`:
  - Creates multiple service replicas, each listening on a unique port.
  - Proxies route traffic to appropriate service replica based on the configuration.


Browser
localhost:5066 (host port)
Frontend Proxy
localhost:65001 (random port)
localhost:65002 (random port)
"frontend_0"
"frontend_1"

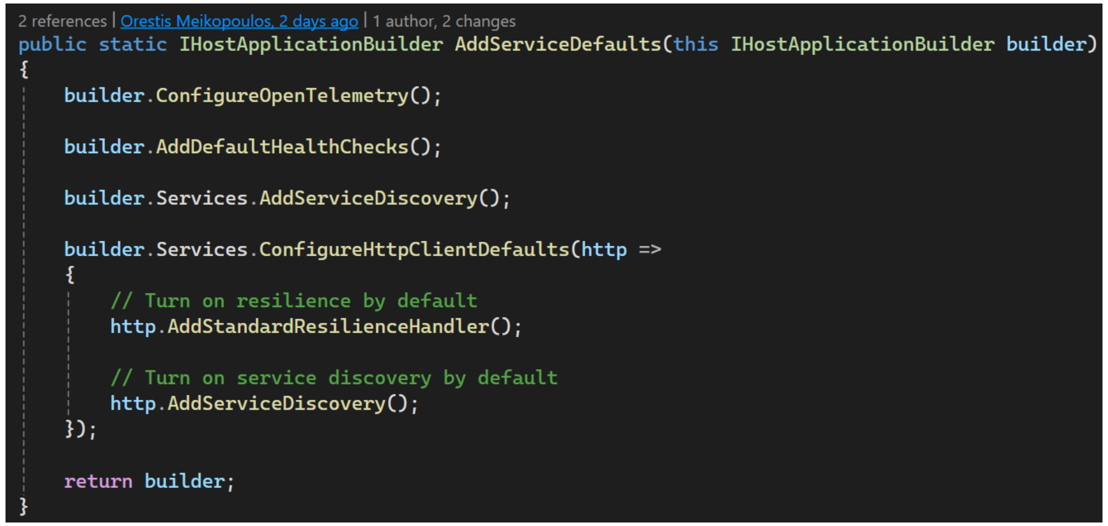
Service Discovery

- **Purpose**: Facilitate configuration of service discovery for development and testing environments.
- **Functionality**: Allows apps within the .NET Aspire framework to automatically discover and connect with each other.
- **Implementation**: Service discovery settings are provided to individual services within the application model based on their references.
- **Example**:

```
`var builder = DistributedApplication.CreateBuilder(args);
var catalog = builder.AddProject<Projects.CatalogService>("catalog");
var basket = builder.AddProject<Projects.BasketService>("basket");
var frontend =
builder.AddProject<Projects.MyFrontend>("frontend").WithReference(basket).WithReference(catalog)`
```

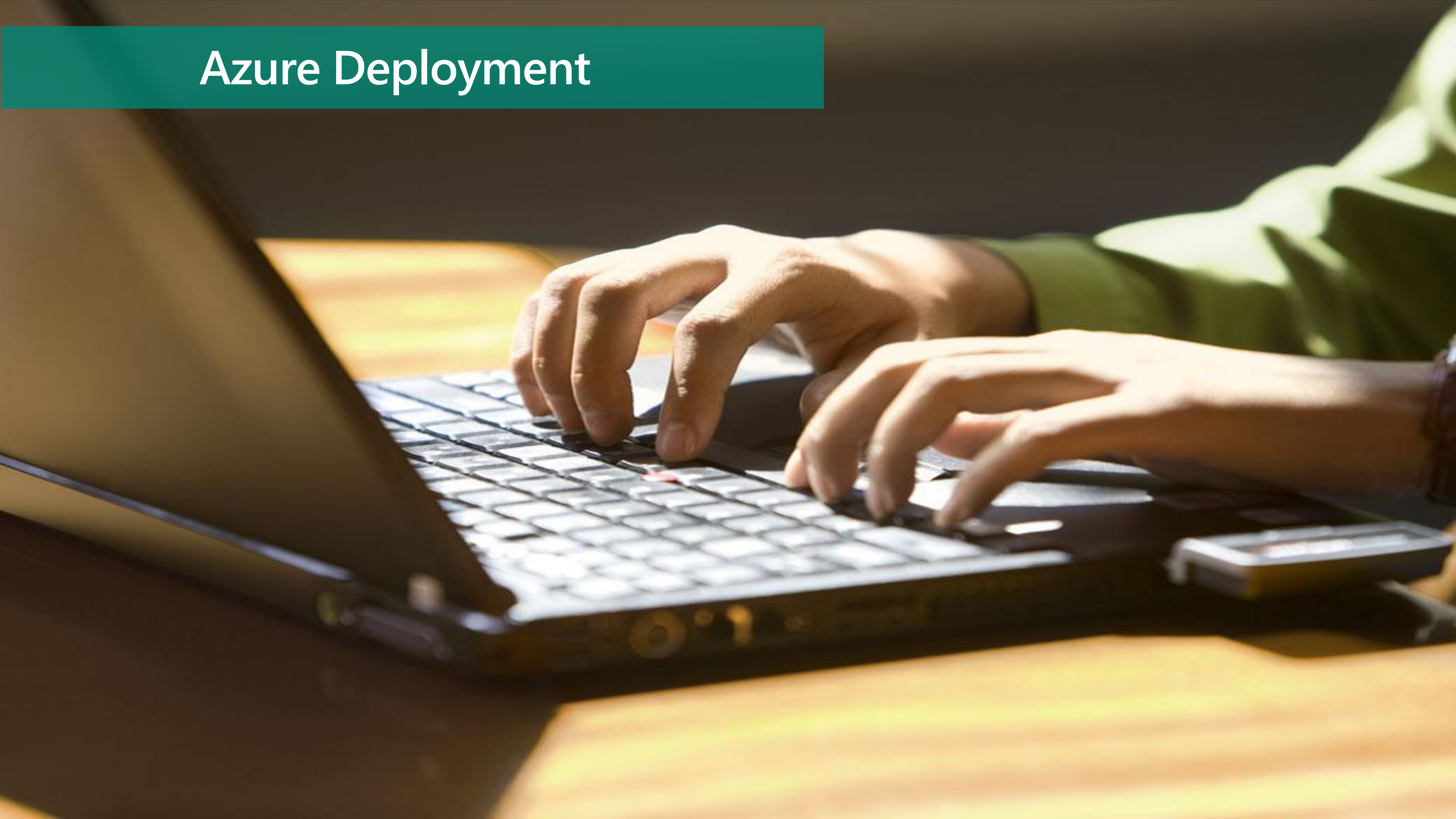
Service Defaults

- **Purpose**: Manage extensive configurations for cloud-native applications across various environments.
- **Key Methods:**
  - `ConfigureOpenTelemetry`: Sets up metrics and tracing.
  - `AddDefaultHealthChecks`: Incorporates default health check endpoints.
  - `AddServiceDiscovery`: Adds service discovery functionality.
  - `ConfigureHttpClientDefaults`: Sets up HttpClient defaults



2 references | Orestis Meikopoulos, 2 days ago | 1 author, 2 changes

```
public static IHostApplicationBuilder AddServiceDefaults(this IHostApplicationBuilder builder)
{
    builder.ConfigureOpenTelemetry();

    builder.AddDefaultHealthChecks();

    builder.Services.AddServiceDiscovery();

    builder.Services.ConfigureHttpClientDefaults(http =>
    {
        // Turn on resilience by default
        http.AddStandardResilienceHandler();

        // Turn on service discovery by default
        http.AddServiceDiscovery();
    });

    return builder;
}
```



Azure Deployment

Thank You

- For the opportunity
- For participating
- For listening