

# Taming Service Dependencies with Aspire

Alex Crome

# Legal Disclaimer

This presentation is provided for information purposes only. Nothing herein should be construed as legal, technical or other professional advice or be relied upon as such. Nothing contained herein should be construed as a representation or warranty.

Trayport and the names of Trayport's products are Trademarks and Service Marks of Trayport Limited, and where relevant have been registered as such.

Other products, services, or company names mentioned herein are the property of, and may be the service mark or trademark of, their respective owners.

© Trayport Limited 2025

# | Our Problem



# The Problem



- **Slow**
- **Brittle**
- **Lots of copy pasta**

- **Lots of copy pasta**
- **Worked in isolation**

# Aspire Integrations Why not Build our Own?

```
builder  
    .AddRedis("cache")  
    .WithDataVolume();
```

The screenshot shows the Aspire Integrations Gallery interface. The top navigation bar includes a search bar, a 'Ctrl K' keyboard shortcut, and system status icons. The left sidebar has a dark theme with purple highlights. It features a navigation menu with 'Docs', 'Integrations' (which is selected), 'CLI Reference', 'Community', and 'Samples'. Below this is an 'Explore' section with 'Integration gallery' (selected), 'Overview', 'Database' (with 'PostgreSQL' listed), and 'Messaging' (with 'RabbitMQ' listed). The main content area is titled 'Integrations Gallery' and displays several integration components:

- # 142 Number of Integrations**: Aspire.StackExchange.Redis. A generic Redis® client that integrates with Aspire, including health checks, logging, and telemetry. Tags: client, component, cloud, +3. Downloads: 2.1M, Version: 9.5.2, Package →.
- # 147 Unique Tags**: Aspire.Npgsql.EntityFrameworkCore.Postgre... A PostgreSQL® provider for Entity Framework Core that integrates with Aspire, including connection pooling, health checks, logging, and telemetry. Tags: client, component, cloud, +12. Downloads: 2M, Version: 9.5.2, Package →, Docs ↗.
- # 32,259,835 Total Downloads**: Aspire.Microsoft.EntityFrameworkCore.SqlS... A Microsoft SQL Server provider for Entity Framework Core that integrates with Aspire, including connection pooling, health check, logging, and telemetry. Tags: client, component, cloud, +10. Downloads: 2M, Version: 9.5.2, Package →, Docs ↗.
- # 1 Testing**: Aspire.Hosting.Testing. Testing support for the .NET Aspire application model. Tags: testing.

# Building an Integration

## Building an Integration The Resource Type

```
public class AccountsResource(string name)
: ContainerResource(name)
{}
```

## Initial Integration Add Extension Method

```
public static IResourceBuilder<AccountsResource> AddAccounts(  
    this IDistributedApplicationBuilder builder,  
    string name)  
{  
    var resource = new AccountsResource(name);  
  
    return builder  
        .AddResource(resource)  
        .WithImage("web/accounts", "latest")  
        .WithImageRegistry("registry.io")  
        .WithEnvironment("ASPNETCORE_ENVIRONMENT", "Development");  
}
```

## Initial Integration Builder Extension Method

```
public static IResourceBuilder<AccountsResource> WithMockUsers(  
    this IResourceBuilder<AccountsResource> builder,  
    string path)  
{  
    const string mountPath = "/mnt/MockUsers.json";  
  
    return builder  
        .WithBindMount(path, mountPath, isReadOnly: true)  
        .WithEnvironment("FakeUsers__Path", mountPath);  
}
```

## Initial Integration Use it

```
var builder = DistributedApplication  
.CreateBuilder(args)  
  
builder  
.AddAccounts("accounts")  
.WithMockUsers("./MockUsers.json");
```

## Expanding the Integration Open Telemetry

```
public static IResourceBuilder<AccountsResource> AddAccounts(...)  
{  
    return builder.AddResource(new AccountsResource(name));  
    //...  
    .WithOtlpExporter();  
}
```

# Enhancing the Integration Endpoints

```
public static IResourceBuilder<AccountsResource> AddAccounts(...)  
{  
    return builder.AddResource(new AccountsResource(name))  
    //...  
    .WithHttpEndpoint(targetPort: 8080,  
                      env: "ASPNETCORE_HTTP_PORTS");  
}
```

The screenshot shows the TMX TRAYPORT application interface. The left sidebar has icons for Resources, Console, Structured, Traces, and a TMX logo. The main area is titled 'Resources' and contains a table with the following data:

Name	State	Start time	Source	URLs	Actions
accounts	Running	5:19:18 PM	web/accounts:latest	<a href="http://localhost:57814">http://localhost:57814</a>	<span>Details</span> <span>...</span>

# Expanding the Integration Health Checks

```
public static IResourceBuilder<AccountsResource> AddAccounts(...)  
{  
    return builder.AddResource(new AccountsResource(name))  
    //...  
    .WithHttpHealthCheck(path: "/health");  
}
```

The screenshot shows the TMX TRAYPORT interface with the title "DotNetConf". On the left, there is a sidebar with icons for "Resources", "Console", "Structured", "Traces", and "Metrics". The "Resources" icon is highlighted with a purple bar. The main area is titled "Resources" and contains a table with the following data:

Name	State	Start time	Source	URLs	Actions
healthy	Running	5:35:19 PM	web/accounts:latest	<a href="http://localhost:59202">http://localhost:59202</a>	<span>Copy</span> <span>Open</span> <span>...</span>
unhealthy	Running (Un...)	5:35:19 PM	web/accounts:latest	<a href="http://localhost:59201">http://localhost:59201</a>	<span>Copy</span> <span>Open</span> <span>...</span>

# Expanding the Integration Test it

```
[Test]
public async Task GoesHealthy(CancellationToken ct)
{
    var builder = DistributedApplicationTestingBuilder.Create();
    builder.Services.AddLogging(logging => {
        logging.SetMinimumLevel(LogLevel.Debug)
            .AddFilter(builder.Environment.ApplicationName, LogLevel.Debug)
            .AddFilter("Aspire.", LogLevel.Debug);
    });

    builder.AddAccounts("accounts");

    await using var app = builder.Build();
    await app.StartAsync(cancellationToken);
    await app.ResourceNotifications.WaitForResourceHealthyAsync("accounts", ct);
}
```

# Build Process Publish Integration Package

Build Process	Trayport.Aspire.Hosting.Accounts.csproj
<pre>dotnet pack Foo.slnx dotnet nuget push artifacts/pack/**.nupkg</pre>	<pre>&lt;Project&gt;   &lt;PropertyGroup&gt;     &lt;IsPackable&gt;true&lt;/IsPackable&gt;   &lt;/PropertyGroup&gt; &lt;/Project&gt;</pre>
	<b>Directory.Build.props</b>
	<pre>&lt;Project&gt;   &lt;PropertyGroup&gt;     &lt;IsPackable&gt;false&lt;/IsPackable&gt;     &lt;UseArtifactsOutput&gt;true&lt;/UseArtifactsOutput&gt;     &lt;ArtifactsPath&gt;\$(MSBuildThisFileDirectory)artifacts&lt;/ArtifactsPath&gt;   &lt;/PropertyGroup&gt; &lt;/Project&gt;</pre>

# Build Process Publish Container

## Build Process

```
dotnet publish Foo.slnx  
/t:PublishContainer  
/p:ContainerRegistry=registry.io  
/p:ContainerImageTags="latest;$(Version)"
```

## Directory.Build.props

```
<Project>  
  <PropertyGroup>  
    <IsPublishable>false</IsPublishable>  
  </PropertyGroup>  
</Project>
```

## Trayport.Accounts.csproj

```
<Project>  
  <PropertyGroup>  
    <IsPublishable>true</IsPublishable>  
    <ContainerRepository>web/accounts</ContainerRepository>  
  </PropertyGroup>  
</Project>
```

# Introducing Dependencies

# Maging Dependencies Adding Dependencies

## Required Dependency - `IDistributedApplicationBuilder.AddXYZ()`

```
public static IResourceBuilder<AccountsResource> AddAccounts(  
    this IDistributedApplicationBuilder builder,  
    string name,  
    IResourceBuilder<PostgresDatabaseResource> db)
```

## Optional Dependency - `resource.WithXYZ()`

```
public static IResourceBuilder<AccountsResource> WithDatabase(  
    this IResourceBuilder<AccountsResource> builder,  
    IResourceBuilder<PostgresDatabaseResource> db)
```

## Dependencies Endpoints

```
var endpoint = accounts.GetEndpoint("http");
var host = endpoint.Property(EndpointProperty.Host);
var port = endpoint.Property(EndpointProperty.Port);

return resource
    .WithEnvironment("OIDC__Authority", endpoint)
    .WithArgs("--authority", endpoint)
    .WithEnvironment("OIDC__Host", host)
    .WithEnvironment("OIDC__Authority", $"http://{host}:{port}");

.WithEnvironment("OIDC__Port", endpoint.Property.Port)
```

## Dependencies Endpoints Gotcha

```
resource.WithEnvironment("OIDC_Url", $"http://{host}:{port}");
```

```
var url = $"http://{host}:{port}"
resource.WithEnvironment("OIDC_Url", url);
```

```
// Or ReferenceExpressionBuilder (like StringBuilder)
var url = ReferenceExpression.Create($"http://{host}:{port}")
resource.WithEnvironment("OIDC_Url", url);
```

## Expanding the Integration Connection Strings

```
var server = db.Resource.Parent;

resource
    .WithEnvironment("DATABASE_HOST", server.Host)
    .WithEnvironment("DATABASE_PORT", server.Port)
    .WithEnvironment("DATABASE_USERNAME", server.UserNameReference)
    .WithEnvironment("DATABASE_PASSWORD", server.PasswordParameter)
    .WithEnvironment("DATABASE_NAME", db.Resource.DatabaseName);

resource
    .WithEnvironment("CS", db.Resource.ConnectionStringExpression)
    .WithEnvironment("JDBC", db.Resource.JdbcConnectionString);
```

## Dependencies Waiting

```
resource.WaitFor(db);  
  
resource.WaitForCompletion(migrations);
```

**Be careful of long chains**

**Persistent Containers can mitigate**

```
resource.WithLifetime(ContainerLifetime.Persistent)
```

# Expanding the Integration Config Files

```
resource.OnBeforeResourceStarted(async (resource, evt, ct) =>
{
    var endpoint = accounts.GetEndpoint("http");
    await endpoint.GetValueAsync(ct);

    var connectionString = await db.GetConnectionStringAsync(ct);

    File.WriteAllText("./config.txt",
        $"""
        endpoint={endpoint.Url}
        connectionString={connectionString}
        """
    );
}) ;
```

## Dependencies Sub Resources

```
public static IResourceBuilder<AccountsResource> WithDatabaseMigrations(
    this IResourceBuilder<AccountsResource> accounts,
    IResourceBuilder<PostgresDatabaseResource> db,
    Action<IResourceBuilder<ExecutableResource>> configure = null)
{
    var migrations = db.ApplicationBuilder
        .AddExecutable($" {db.Resource.Name}-Migrations",
            "dotnet", "./Trayport.Accounts")
        .WithArgs("ef", "database", "update", "--connection", db.Resource)
        .WaitFor(db)
        .WithParentRelationship(db);
    configure?.Invoke(migrations);
    return accounts.WaitForCompletion(migrations);
}
```

A wide-angle photograph of a massive solar power facility. The foreground is filled with numerous rows of dark blue solar panels, which are angled towards the sun. The ground is a light-colored, sandy desert soil. In the background, there are some industrial buildings, a tall water tower, and a clear blue sky.

# Helpers

## Helper Conventions

```
public static IDistributedApplicationBuilder WithTrayportDefaults(  
    this IDistributedApplicationBuilder builder)  
{  
    return builder  
        .WithRegistryMirrors()  
        .WithHostCertificates();  
}
```

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

## Helpers Registry Mirrors

```
public static IDistributedApplicationBuilder WithRegistryMirror(  
    this IDistributedApplicationBuilder builder)  
{  
    builder.Eventing.Subscribe<BeforeStartEvent>((evt, _) => {  
        foreach (var resource in evt.Model.Resources)  
        foreach (var annotation in resource.Annotations.OfType<ContainerImageAnnotation>())  
        {  
            if (annotation.Registry == "docker.io") {  
                annotation.Registry = "mirror.registry.io";  
                annotation.Image = $"cache/docker.io/{annotation.Image}";  
            }  
        }  
        return Task.CompletedTask;  
    });  
    return builder;  
}
```

## Helpers Certificates

```
public static IDistributedApplicationBuilder WithHostCertificates(
    this IDistributedApplicationBuilder builder)
{
    builder.Eventing.Subscribe<BeforeStartEvent>((evt, ct) => {
        foreach (var resource in evt.Model.GetContainerResources())
        {
            builder.CreateResourceBuilder(resource)
                .WithCertificateTrustScope(CertificateTrustScope.System);
        }
        return Task.CompletedTask;
    });

    return builder;
}
```

## Helpers Container Image

```
public static IResourceBuilder<T> WithTrayportImage<T>(
    this IResourceBuilder<T> resourceBuilder,
    string image,
    string tag = "latest")
    where T : ContainerResource
{
    var registry = imageTag.Contains("-") ? "NonProd.io" : "Prod.io";

    return resourceBuilder.WithImage(image)
        .WithImageTag(tag)
        .WithImageRegistry(registry);
}
```



# | Did it work?

# Today Internal Integrations

33

Internal Integrations

The screenshot shows a software package manager interface with a dark theme. At the top, there are tabs: 'Browse' (which is selected), 'Installed', 'Updates', and 'Consolidate 1'. Below the tabs is a search bar containing the text 'Trayport.Aspire.Hosting' with a clear button ('X') and a refresh icon. There is also a checkbox labeled 'Include prerelease'. The main area displays a list of 33 packages, each with a blue circular icon, the package name in bold, the author's name, a brief description, and the version number. The packages listed are:

- Trayport.Aspire.Hosting.Dawn** by Trayport.Aspire.Hosting.Dawn 1.0.54  
Support for Dawn containers as an Aspire resource
- Trayport.Aspire.Hosting.EplexConnector** by Trayport.Aspire.Hosting.EplexConnector 3.2.75  
Aspire Hosting package for Trayport Aggregation Database
- Trayport.Aspire.Hosting.ExchangeOrderDistribution** by Trayport.Aspire.Hosting.ExchangeOrderDistr 1.0.154  
Aspire Hosting package for Trayport Aggregation Database
- Trayport.Aspire.Hosting.ExchangeRequestRouter** by Trayport.Aspire.Hosting.ExchangeRequestRouter 1.0.73  
Aspire Hosting package for Trayport Theseus Exchange RequestRouter
- Trayport.Aspire.Hosting.Execution.Configuration** by Trayport.Aspire.Hosting.Execution.Configuration 0.133.0  
Support for Execution Configuration container as an Aspire resource

# Today Internal Integrations

```
builder.AddDawn();
builder.AddUnleash();

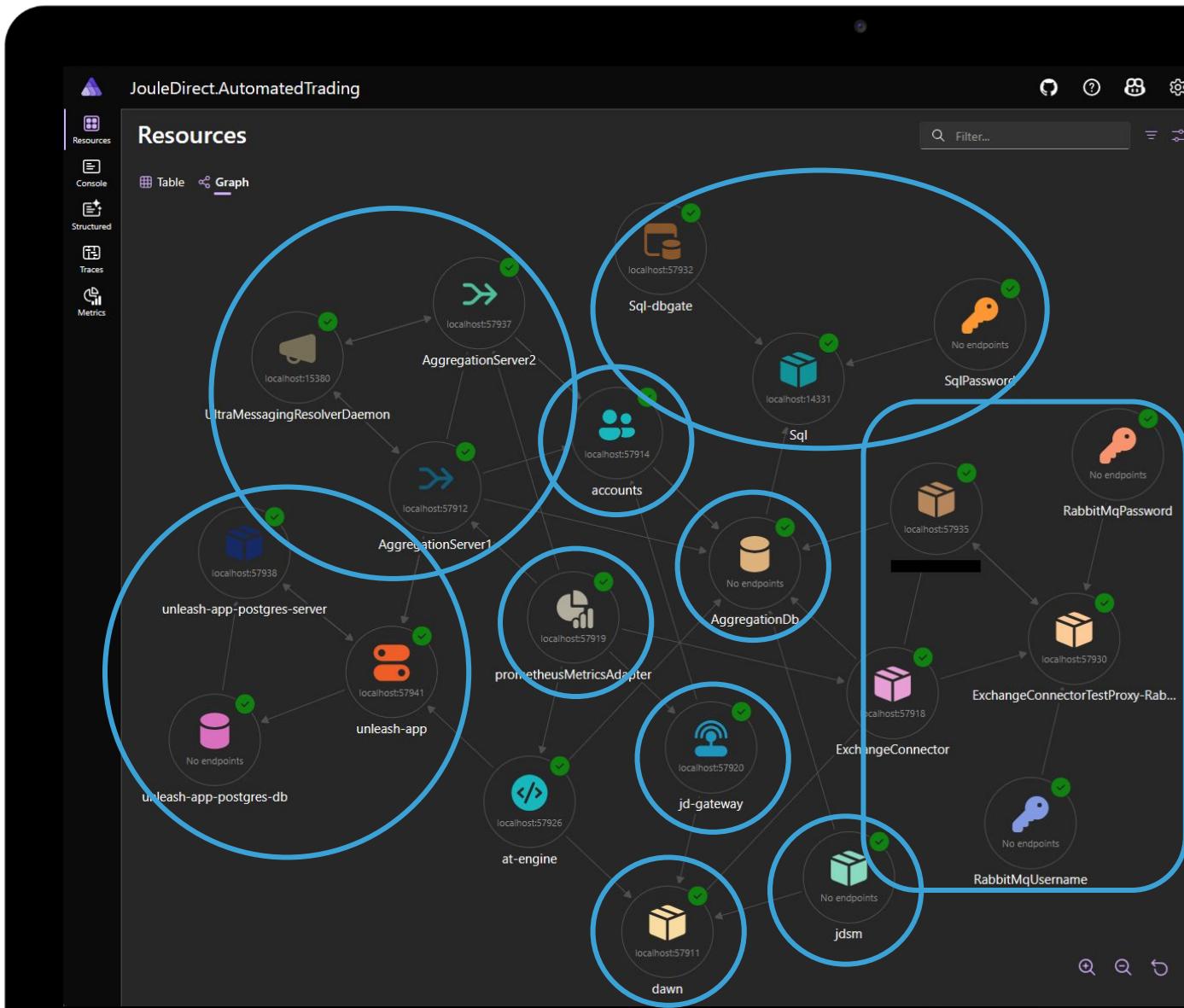
var aggDb = builder
    .AddSqlServer()
    .AddAggregationDatabase();

var accounts = builder
    .AddAccounts()
    .WithAggregationDb(aggDb);

builder
    .AddAggregationServer(aggDb)
    .WithAccounts(accounts);

// ...

```



# Challenges Keeping Containers Up To Date

```
.WithImageTag("latest")
```

- ✓ Simple
- ✗ Different developers will end up with different dependency versions
- ✗ Dependencies can get out of date (potentially years!)
- ⚠ Can be updated by running docker pull.

```
.WithImageTag("latest")
.WithImagePullPolicy(ImagePullPolicy.Always)
```

- ✓ Always up to date
- ✗ Fail to start if network unavailable

```
var version = Assembly.GetExecutingAssembly().GetName().Version;
.WithImageTag(version.ToString())
```

- ✓ Stable dependency version
- ✗ Images updates require nuget package updates

## Challenges Non Containerisable Executables

- Containers handle both acquisition and execution
- To run an executable, how do you acquire it?
  - Initially: Cheat and assume path from old deployment start
  - Exploring packing as dotnet tool
    - dnx / dotnet tool exec
    - Can we smuggle non dotnet tools through native AOT tool packages?

# Challenges Project Substitution

- Containers simple for others to use
- Projects better for debugging

No common base type between Projects and Containers!

1. Remove existing Container Resource
2. Add Project resource with same name
3. Copy annotations from container to project resource
4. Fix up differences for container vs project
  - Endpoints
  - Bind Mounts
5. Forward event handlers

```
builder
    .AddAccounts()
    .AsProject<AccountsResource, Projects.Trayport_Accounts>()
```

The background of the slide is a dark, atmospheric photograph of a renewable energy facility. In the foreground, several solar panels are visible, their surfaces reflecting some light. Behind them, several wind turbines stand tall against a dark sky. One prominent wind turbine on the left has a bright, radial lens flare effect emanating from its base, suggesting either a rising or setting sun. The overall mood is one of industrial progress and environmental sustainability.

# The Future

## More Integrations

- More Hosting Integrations
- Explore Client Integrations

## Ephemeral Environments

- PRs
- A la carte
- Exploratory Testing

## Reduce Dependencies

## Testing

- Exploratory Testing
- Chaos Testing

# Thank You



[info@trayport.com](mailto:info@trayport.com)

[www.trayport.com](http://www.trayport.com)

**United Kingdom (Head Office)**

**Austria**

**Germany**

**Singapore (Asia Pacific)**

Trayport Limited, 3rd Floor, 2 Gresham Street, London, EC2V 7AD, United Kingdom

Trayport Austria GmbH, Euro Plaza 2E/1 OG, Technologiestraße 10, 1120 Wien, Austria

Trayport Germany GmbH, Linzer Straße 11, 28359 Bremen, Germany

Trayport Pte Ltd, One Raffles Place, Office Tower 1, #31-02, Singapore, 048616

+44 20 7960 5500

+43 1 609 2290

+49 421 20109-0

+65 6411 4700

Feedback

