Payoneer

.NET Aspire

Tamir Dresher
Head of Architecture @ Payoneer

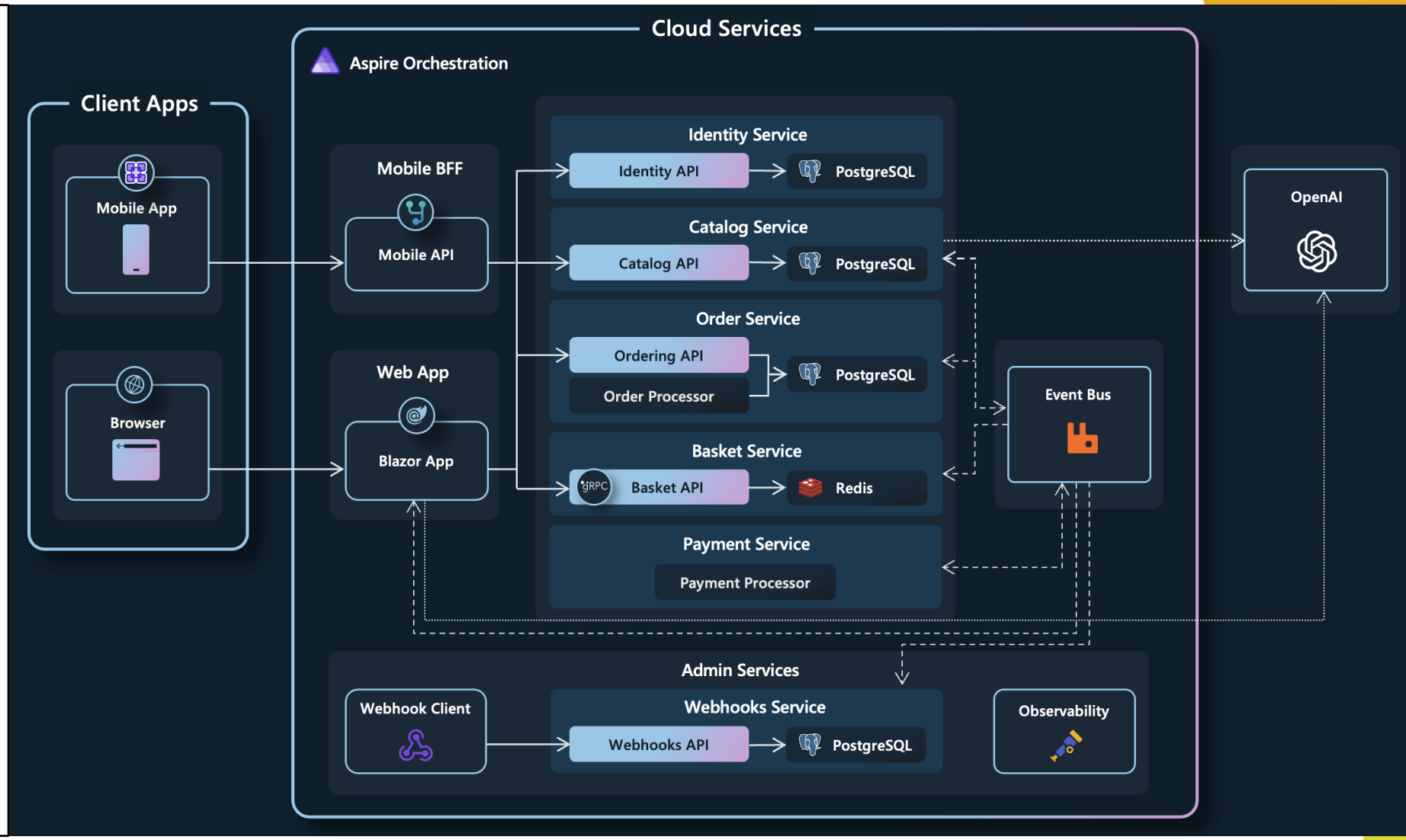
31/01/2024

Distributed Applications / Microservices are hard!

Creating a local dev env for
Distributed Applications / Microservices
is hard!

Payoneer

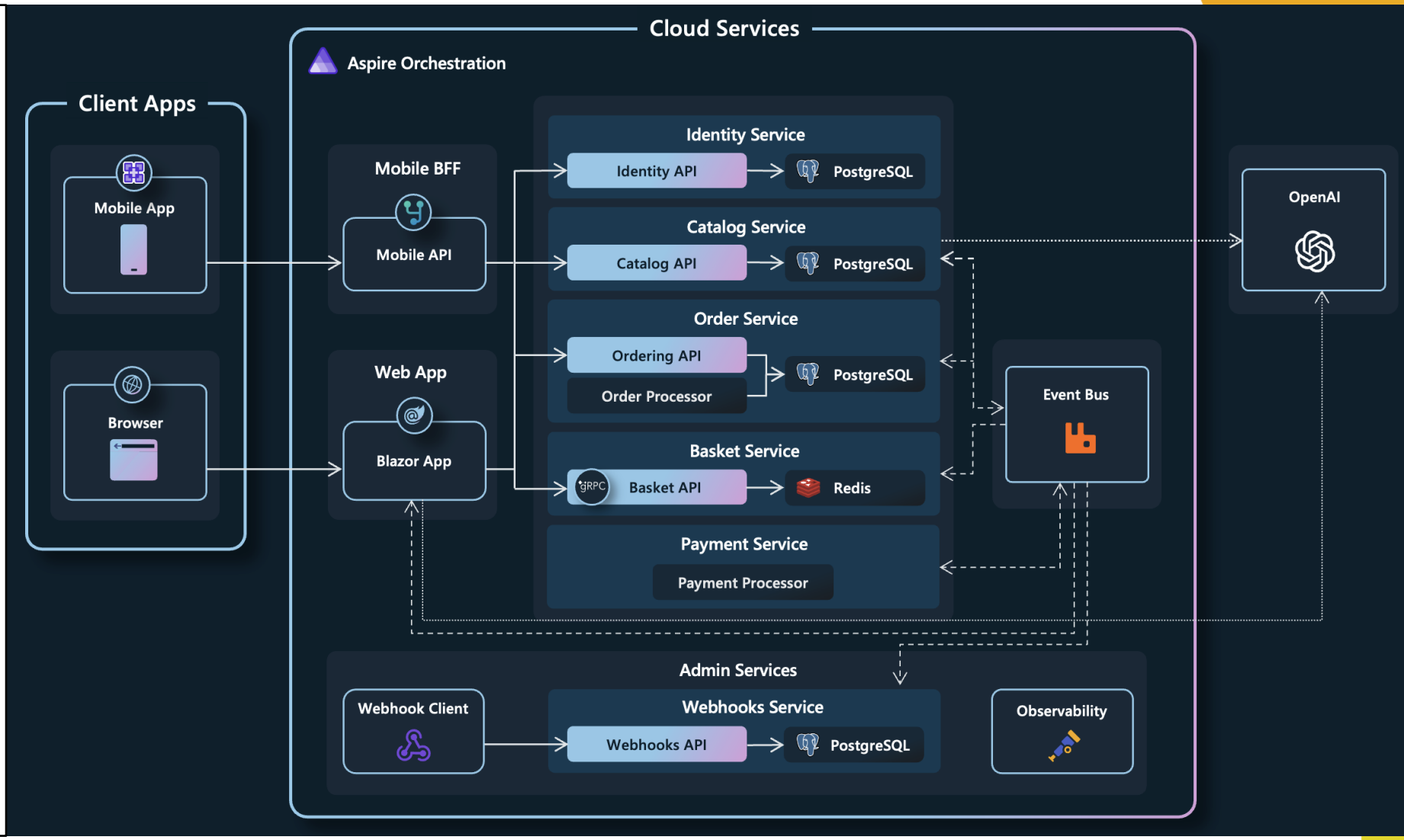
eShop as an example



https://github.com/dotnet/eshop

eShop as an example

- Running multiple interconnected services
  - Expressing the system structure
  - Configuring service discovery
  - Multiple instances
- Making it production grade: reliability, health checks, logging, telemetry etc.
- Local experience for observability and debuggability

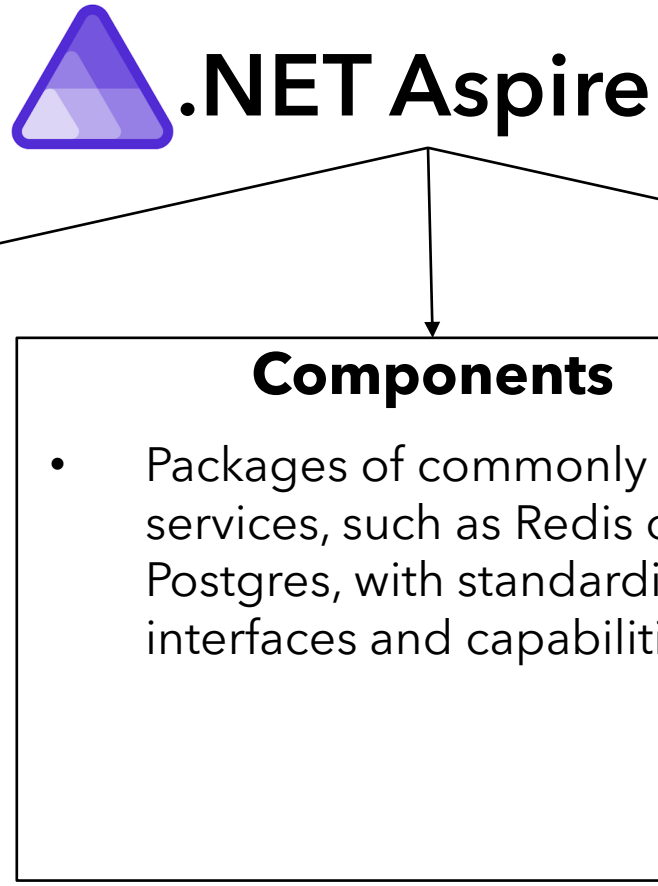


Payoneer

https://github.com/dotnet/eshop

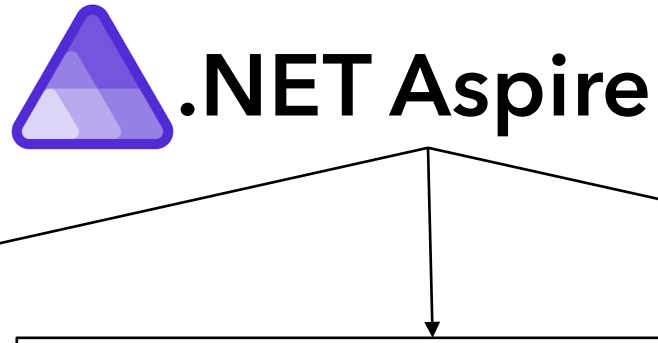
.NET Aspire

Opinionated, cloud ready stack for building observable, production ready, distributed applications. .NET



.NET Aspire

Opinionated, cloud ready stack for building observable, production ready, distributed applications. .NET



Composition & Orchestration

- Modeling language for defining the system resources and their dependencies
- A runtime to for running and connecting multi-project applications and their dependencies

Components

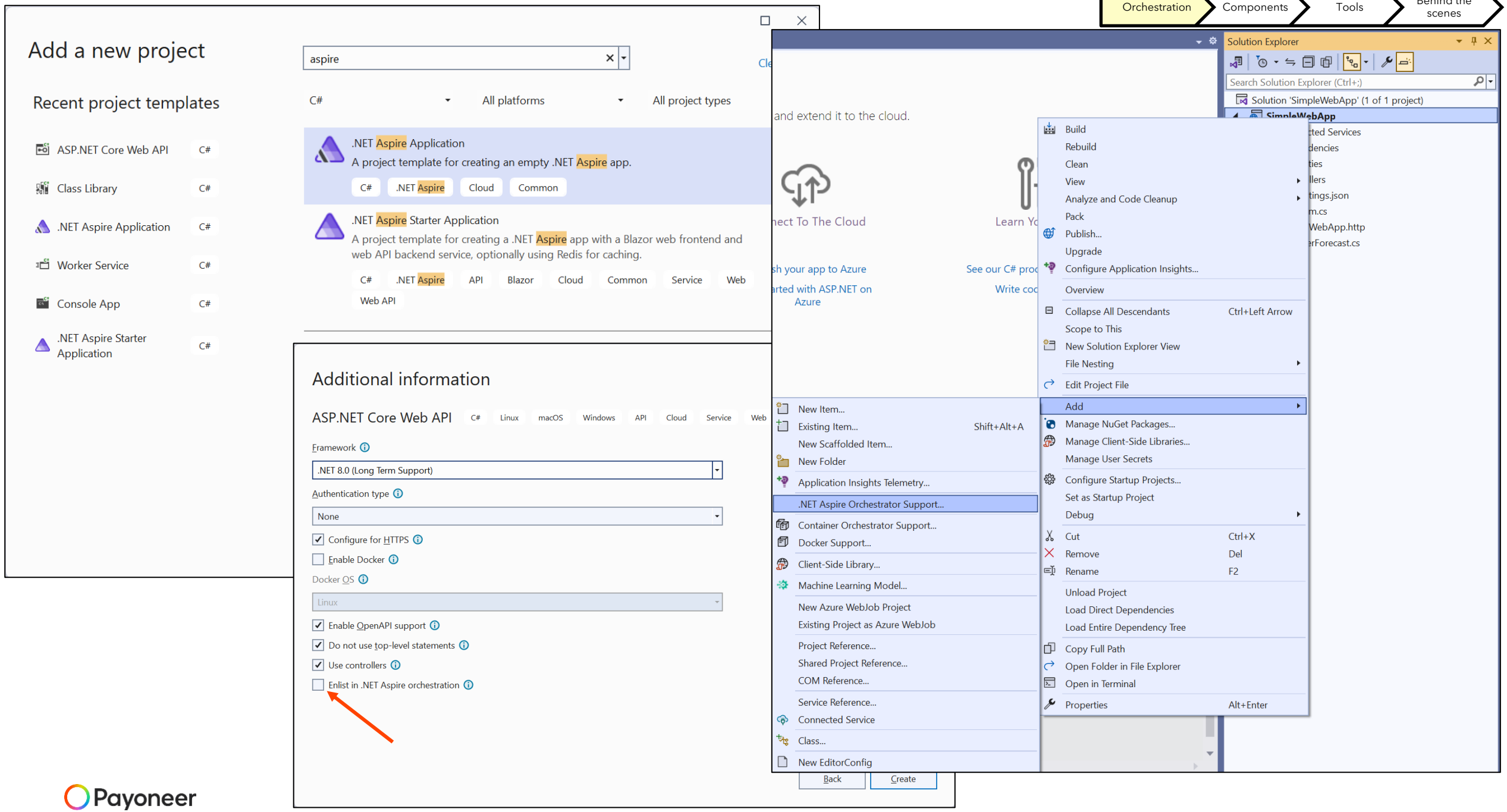
- Packages of commonly used services, such as Redis or Postgres, with standardized interfaces and capabilities

Tools

- Dashboard
- Distributed App VS support
- Service Defaults
- Publishing utils

Composition & Orchestration

Demo


Add a new project
Recent project templates
ASP.NET Core Web API C#
Class Library C#
.NET Aspire Application C#
Worker Service C#
Console App C#
.NET Aspire Starter Application C#
aspire
C#
All platforms
All project types
.NET Aspire Application
A project template for creating an empty .NET Aspire app.
C# .NET Aspire Cloud Common
.NET Aspire Starter Application
A project template for creating a .NET Aspire app with a Blazor web frontend and web API backend service, optionally using Redis for caching.
C# .NET Aspire API Blazor Cloud Common Service Web Web API
Additional information
ASP.NET Core Web API C# Linux macOS Windows API Cloud Service Web
Framework
.NET 8.0 (Long Term Support)
Authentication type
None
Configure for HTTPS
Enable Docker
Docker OS
Linux
Enable OpenAPI support
Do not use top-level statements
Use controllers
Enlist in .NET Aspire orchestration
Back
Create
and extend it to the cloud.
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'SimpleWebApp' (1 of 1 project)
SimpleWebApp
Build
Rebuild
Clean
View
Analyze and Code Cleanup
Pack
Publish...
Upgrade
Configure Application Insights...
Overview
Collapse All Descendants Ctrl+Left Arrow
Scope to This
New Solution Explorer View
File Nesting
Edit Project File
Add
Manage NuGet Packages...
Manage Client-Side Libraries...
Manage User Secrets
Configure Startup Projects...
Set as Startup Project
Debug
Cut Ctrl+X
Remove Del
Rename F2
Unload Project
Load Direct Dependencies
Load Entire Dependency Tree
Copy Full Path
Open Folder in File Explorer
Open in Terminal
Properties Alt+Enter
New Item...
Existing Item... Shift+Alt+A
New Scaffolded Item...
New Folder
Application Insights Telemetry...
.NET Aspire Orchestrator Support...
Container Orchestrator Support...
Docker Support...
Client-Side Library...
Machine Learning Model...
New Azure WebJob Project
Existing Project as Azure WebJob
Project Reference...
Shared Project Reference...
COM Reference...
Service Reference...
Connected Service
Class...
New EditorConfig

Payoneer

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

var cache = builder.AddRedisContainer("cache");

var apiservice =
    builder.AddProject<Projects.AspireApp1_ApiService>("apiservice");

builder.AddProject<Projects.AspireApp1_Web>("webfrontend")
    .WithReference(cache)
    .WithReference(apiservice);

builder.AddProject<Projects.WorkerService1>("workerservice1");

var distributedAppModel =
    new DistributedApplicationModel(builder.Resources)
    {
        Name = "My Distributed System"
    };

DistributedApplication app = builder.Build();
app.Run();
```

QuickWatch

Expression: distributedAppModel

Reevaluate

Add Watch

Value:

Name	Value
distributedAppModel	Name = "My Distributed System", Resources = 4
Name	"My Distributed System" View
Resources	Count = 4 View
[0]	Type = RedisContainerResource, Name = "cache"
[1]	Type = ProjectResource, Name = "apiservice"
[2]	Type = ProjectResource, Name = "webfrontend"
[3]	Type = ProjectResource, Name = "workerservice1"
Raw View	

Close

Payoneer

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

var cache = builder.AddRedisContainer("cache");

var apiservice =
    builder.AddProject<Projects.AspireApp1_ApiService>("apiservice");

builder.AddProject<Projects.AspireApp1_Web>("webfrontend")
    .WithReference(cache)
    .WithReference(apiservice);

builder.AddProject<Projects.WorkerService1>("workerservice1");

var distributedAppModel =
    new DistributedApplicationModel(builder.Resources)
    {
        Name = "My Distributed System"
    };

DistributedApplication app = builder.Build();
app.Run();
```

QuickWatch

Expression: distributedAppModel

Reevaluate | Add Watch

Value:

Name	Value
distributedAppModel	Name = "My Distributed System", Resources = 4
Name	"My Distributed System" View
Resources	Count = 4 View
[0]	Type = RedisContainerResource, Name = "cache"
Annotations	Count = 4 View
[0]	{Aspire.Hosting.ApplicationModel.ManifestPublishingCallbackAnnotation}
[1]	Type = ServiceBindingAnnotation, Name = "tcp"
[2]	Type = ContainerImageAnnotation, Image = "redis", Tag = "latest"
[3]	Type = AllocatedEndpointAnnotation, Name = "tcp", UriString = "tcp://localhost:59404",...
Raw View	
Name	"cache" View
[1]	Type = ProjectResource, Name = "apiservice"
[2]	Type = ProjectResource, Name = "webfrontend"
[3]	Type = ProjectResource, Name = "workerservice1"
Raw View	

Close

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

var cache = builder.AddRedisContainer("cache");

var apiservice =
    builder.AddProject<Projects.AspireApp1_ApiService>("apiservice");

builder.AddProject<Projects.AspireApp1_Web>("webfrontend")
    .WithReference(cache)
    .WithReference(apiservice);

builder.AddProject<Projects.WorkerService1>("workerservice1");

var distributedAppModel =
    new DistributedApplicationModel(builder.Resources)
    {
        Name = "My Distributed System"
    };

DistributedApplication app = builder.Build();
app.Run();
```

QuickWatch

Expression:

new System.Collections.Generic.ICollectionDebugView<Aspire.Hosting.ApplicationModel.IResourceAnnotation>(((A

Reevaluate

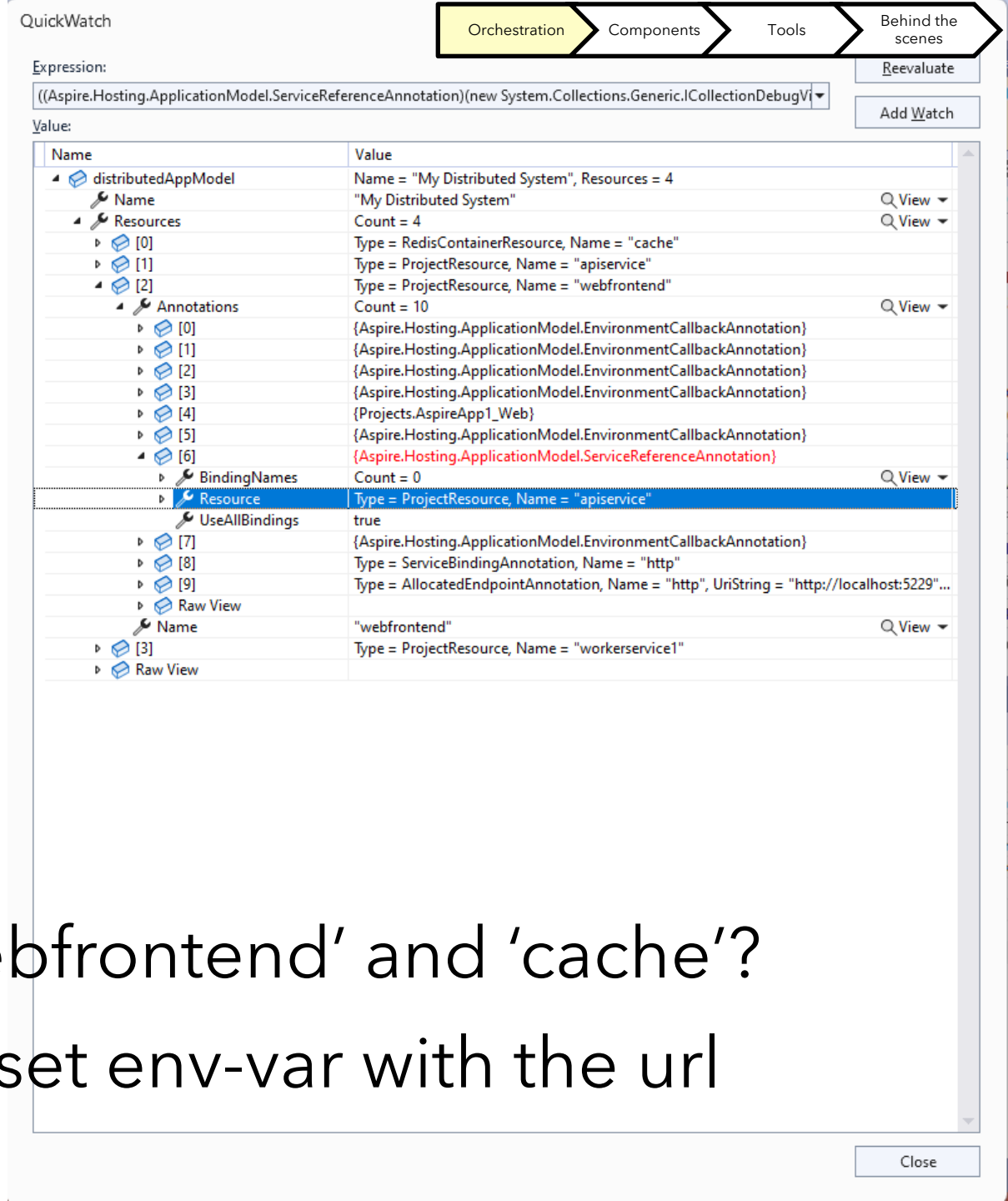
Add Watch

Value:

Name	Value
distributedAppModel	Name = "My Distributed System", Resources = 4
Name	"My Distributed System" View
Resources	Count = 4 View
[0]	Type = RedisContainerResource, Name = "cache"
[1]	Type = ProjectResource, Name = "apiservice"
Annotations	Count = 7 View
[0]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[1]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[2]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[3]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[4]	{Projects.AspireApp1_ApiService}
ProjectPath	"C:\\Users\\tamirdr\\source\\repos\\AspireApp1\\AspireApp1.ApiService\\A... View
[5]	Type = ServiceBindingAnnotation, Name = "http"
[6]	Type = AllocatedEndpointAnnotation, Name = "http", UriString = "http://localhost:5457"...
Raw View	
Name	"apiservice" View
[2]	Type = ProjectResource, Name = "webfrontend"
[3]	Type = ProjectResource, Name = "workerservice1"
Raw View	

Close

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

var cache = builder.AddRedisContainer("cache");

var apiservice =
    builder.AddProject<Projects.AspireApp1_ApiService>("apiservice");

builder.AddProject<Projects.AspireApp1_Web>("webfrontend")
    .WithReference(cache)
    .WithReference(apiservice);

builder.AddProject<Projects.WorkerService1>("workerservice1");

var distributedAppModel =
    new DistributedApplicationModel(builder.Resources)
    {
        Name = "My Distributed System"
    };

DistributedApplication app = builder.Build();
app.Run();
```



Q: Where is the link between ‘webfrontend’ and ‘cache’?

A: Aspire set the env callback to set env-var with the url

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

var cache = builder.AddRedisContainer("cache");

var apiservice =
    builder.AddProject<Projects.AspireApp1_ApiService>("apiservice");

builder.AddProject<Projects.AspireApp1_Web>("webfrontend")
    .WithReference(cache)
    .WithReference(apiservice);

builder.AddProject<Projects.WorkerService1>("workerservice1");

var distributedAppModel =
    new DistributedApplicationModel(builder.Resources)
    {
        Name = "My Distributed System"
    };

DistributedApplication app = builder.Build();
app.Run();
```

QuickWatch

Expression: distributedAppModel

Reevaluate | Add Watch

Value:

Name	Value
distributedAppModel	Name = "My Distributed System", Resources = 4
Name	"My Distributed System"
Resources	Count = 4
[0]	Type = RedisContainerResource, Name = "cache"
Annotations	Count = 4
[0]	{Aspire.Hosting.ApplicationModel.ManifestPublishingCallbackAnnotation}
[1]	Type = ServiceBindingAnnotation, Name = "tcp"
[2]	Type = ContainerImageAnnotation, Image = "redis", Tag = "latest"
[3]	Type = AllocatedEndpointAnnotation, Name = "tcp", UriString = "tcp://localhost:59404",...
Raw View	
Name	"cache"
[1]	Type = ProjectResource, Name = "apiservice"
Annotations	Count = 7
[0]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[1]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[2]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[3]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[4]	{Projects.AspireApp1_ApiService}
[5]	Type = ServiceBindingAnnotation, Name = "http"
[6]	Type = AllocatedEndpointAnnotation, Name = "http", UriString = "http://localhost:5457"...
Raw View	
Name	"apiservice"
[2]	Type = ProjectResource, Name = "webfrontend"
Annotations	Count = 10
[0]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[1]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[2]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[3]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[4]	{Projects.AspireApp1_Web}
[5]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[6]	{Aspire.Hosting.ApplicationModel.ServiceReferenceAnnotation}
[7]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[8]	Type = ServiceBindingAnnotation, Name = "http"
[9]	Type = AllocatedEndpointAnnotation, Name = "http", UriString = "http://localhost:5229"...
Raw View	
Name	"webfrontend"
[3]	Type = ProjectResource, Name = "workerservice1"
Annotations	Count = 5
[0]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[1]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[2]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[3]	{Aspire.Hosting.ApplicationModel.EnvironmentCallbackAnnotation}
[4]	{Projects.WorkerService1}
Raw View	
Name	"workerservice1"

Close

A few interesting examples

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

var catalogDb = builder.AddPostgres("postgres")
                       .WithPgAdmin()
                       .AddDatabase("catalogdb");

builder.AddProject<Projects.CatalogDb>("catalogdbapp")
       .WithReference(catalogDb);

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

Resource specific declarative config

```
var catalogService = builder.AddProject<Projects.CatalogService>("catalogservice")
                            .WithReference(catalogDb)
                            .WithReplicas(2);
```

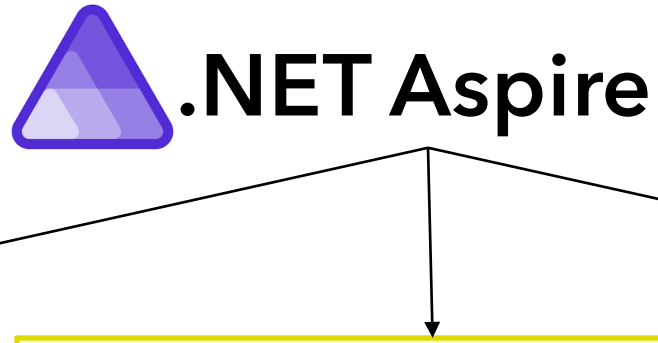
Multiple instances

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

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

var weatherapi = builder.AddProject<Projects.AspireWithNode_AspNetCoreApi>("weatherapi");

builder.AddNpmApp("frontend", "../NodeFrontend", "watch")
    .WithReference(weatherapi)
    .WithReference(cache)
    .WithServiceBinding(containerPort: 3000, scheme: "http", env: "PORT")
    .AsDockerfileInManifest();

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

External JS app

.NET Aspire

Opinionated, cloud ready stack for building observable, production ready, distributed applications. .NET



Composition & Orchestration

- Modeling language for defining the system resources and their dependencies
- A runtime to for running and connecting multi-project applications and their dependencies

Components

- Packages of commonly used services, such as Redis or Postgres, with standardized interfaces and capabilities

Tools

- Dashboard
- Distributed App VS support
- Service Defaults
- Publishing utils

Aspire Components

Standardizing how we use the system resources

- Each system resource our app depends on should be consumed by standardized client
- The client should provide
  - Observability and telemetry
    - > Logs
    - > Trace
    - > Metrics
  - Health checks
  - Resiliency - the ability of your system to react to failure and still remain functional
    - > Retries
    - > Timeouts
    - > Circuit breakers
- All the above should conform to the same conventions (configuration, API surface etc) stated in https://github.com/dotnet/aspire/blob/main/src/Components/README.md

Sta

- Eac
  clie
- The
  -
  -
  -
- All
  etc
  htt

Payc

aspire / src / Components / README.md

Preview | Code | Blame | 213 lines (158 loc) · 14.6 KB

Naming

- Each component's name must have the prefix `Aspire.`.
- When component is built around `ABC` client library, it should contain the client library name in its name. Example: `Aspire.ABC`. Where the technology has a particular casing we have preferred that: for example `Aspire.RabbitMQ` rather than `Aspire.RabbitMq`.
- When the client library is just one of many libraries that allows to consume given service, the names that refer to component need to be specific, not generic. Example: Npgsql is not the only db driver for PostgreSQL database, so the extension method should be called `AddNpgsql` rather than `AddPostgreSQL`. The goal here is to help app authors make an informed decision about which to choose.

Public API

- Each component should have an `AddXXX` extension method extending `IHostApplicationBuilder`
  - Consider adding support for keyed DI, if applicable.
  - A component can have more APIs, if necessary, but it is an explicit goal of Aspire Components to **not** wrap the underlying client library's APIs in new, convenient, higher-level APIs.
- Each component should provide it's own public and `sealed` `Settings` type.
  > [!NOTE] This type does not use the name `Options` because it is not an IOptions. `IOptions` objects can be configured through dependency injection. These settings need to be read before the DI container is built, so they can't be `IOptions`.
- The settings type name should be unique (no generic names like `ConfigurationOptions`), not contain an `Aspire` prefix and follow the client-lib name. Example: when a component wraps an `ABC` client library, the package is called `Aspire.ABC` and the settings type is named `ABCSettings` and is either in the `Aspire.ABC` namespace or a sub namespace.

Configuration

- When a new instance of the settings type is created, its properties should return the recommended/default values (so when they are bound to an empty config they still return the right values).
- Settings should be bound to a section of `IConfiguration` exposed by `IHostApplicationBuilder.Configuration`.
- Each component should determine a constant configuration section name for its settings under the `Aspire` config section.
- All configuration knobs exposed by the settings type should be public and mutable, so they can be changed in the config and applied without a need for re-compiling the application.
- Each component should expose an optional lambda that accepts an instance of given settings type. By doing that, we provide the users with a possibility to override the bound config values (make final changes).
- When a mandatory config property is missing, an exception should be thrown, and it should contain information about the config path that was used to read it.

Sta

- Eac
  clie
- The
  -
  -
  -
- All
  etc
  htt

Payo

aspire / src / Components / README.md

Preview | Code | Blame | 213 lines (158 loc) · 14.6 KB

```
        }
      }
    }
  }
```

Health Checks

Aspire components expose health checks enabling applications to track and respond to the remote service's health.

- Health checks should be enabled by default, but the users should be able to disable them via configuration.
- AddHealthChecks(this IServiceCollection) should be used to register health checks.
- If the client library provides an integration with `HealthCheckService` (example: Microsoft.Extensions.Diagnostics.HealthChecks.EntityFrameworkCore) it should be used.
- If there is an established open-source health check (example: AspNetCore.HealthChecks.Redis) it should be used. If the existing health check library doesn't meet our requirements, efforts should be made to add the necessary functionality to the existing library.
- Otherwise we need to implement IHealthCheck and register it via HealthCheckRegistration.
- Consider whether the Health Check should reuse the same client object registered in DI by the component or not. Reusing the same client object has the advantages of getting the same configuration, logging, etc during the health check.
- Calling MapHealthChecks is outside of the scope of a Component.

Resilience

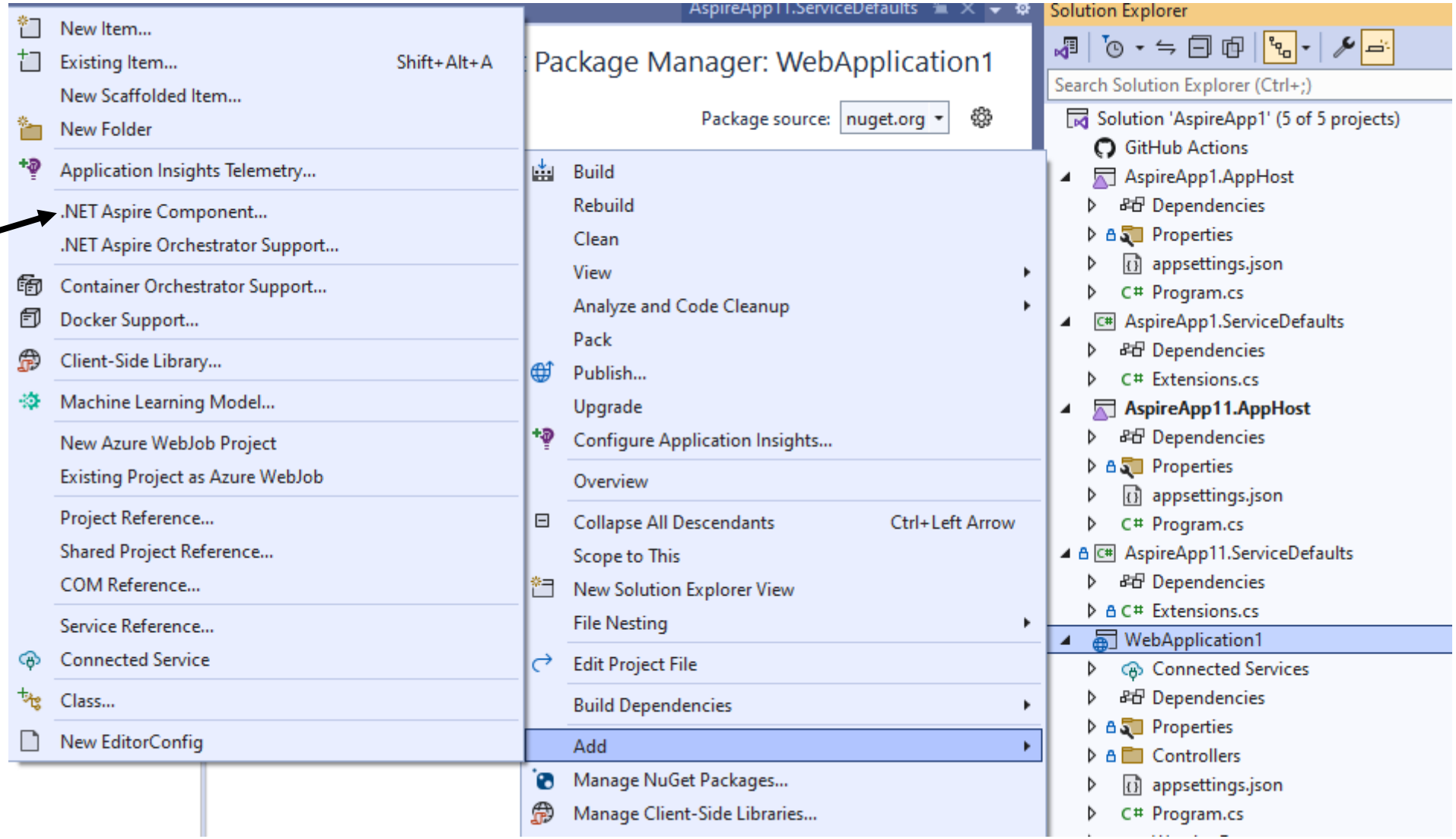
Aspire components leverage configurable resilience patterns such as retries, timeouts, and circuit breakers to maximize availability. This functionality is configurable and seamlessly integrates with higher level resilience strategies implemented at the application level.

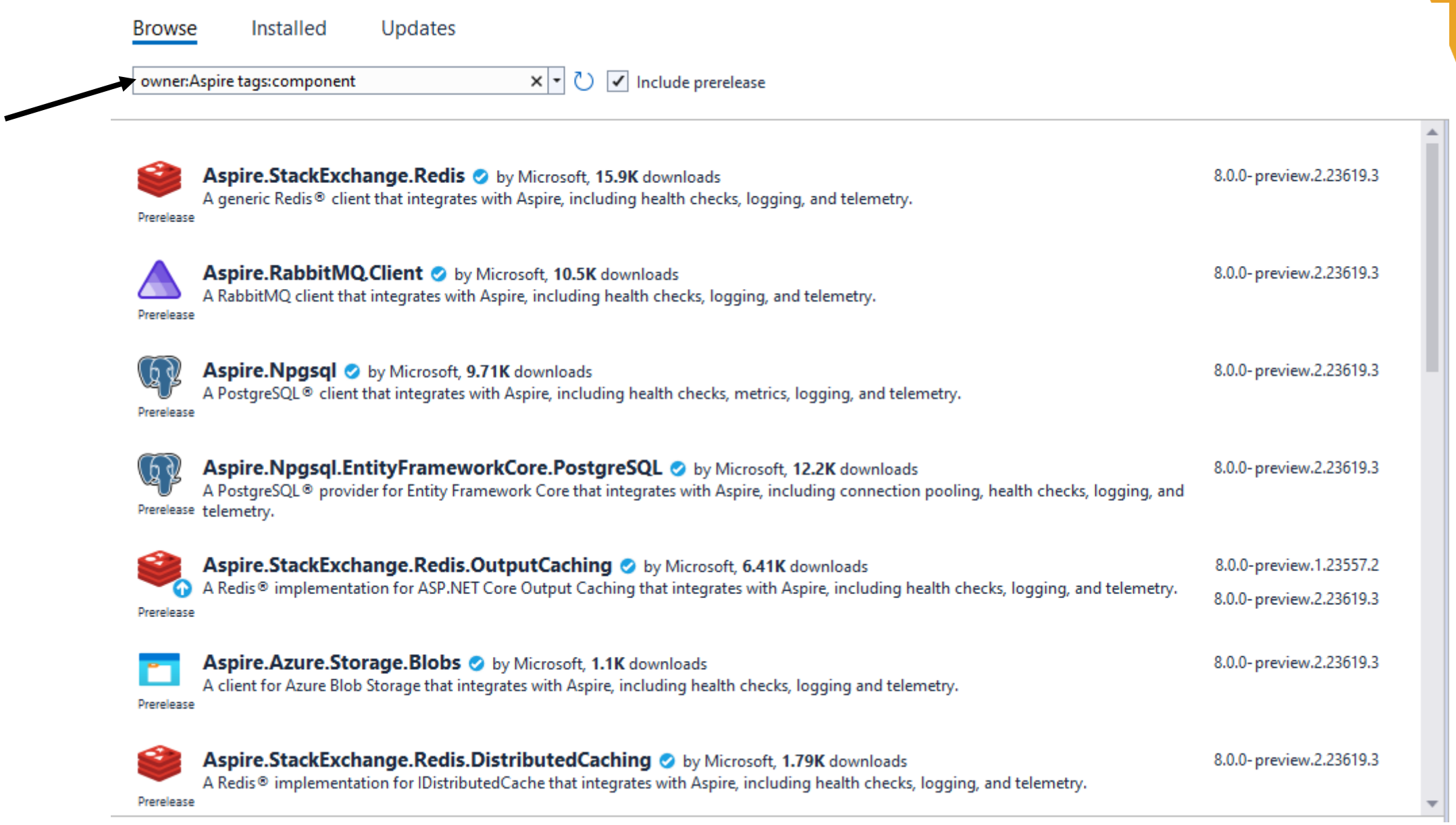
- Each component must ensure that by default reasonable timeouts are enabled. It should be possible to configure the timeouts.
- If the client library provides connection pooling, it should be enabled by default (to scale proportionally). It should be possible to disable it via configuration.
- If given client library provides built in mechanism for retries, it should be enabled by default and configurable.
- It's not always possible to implement retries. Example: A raw db driver does not know whether the currently executed command is part of a transaction or not. If it is, re-trying a failed command won't help as the whole transaction has already failed.

Telemetry

Aspire components offer integrated logging, metrics, and tracing using modern .NET abstractions (ILogger, Meter, Activity). Telemetry is schematized and part of a component's contract, ensuring backward compatibility across versions of the component.

- The component's telemetry names should conform to OpenTelemetry's Semantic Conventions when available.


New Item...
Existing Item... Shift+Alt+A
New Scaffolded Item...
New Folder
Application Insights Telemetry...
.NET Aspire Component...
.NET Aspire Orchestrator Support...
Container Orchestrator Support...
Docker Support...
Client-Side Library...
Machine Learning Model...
New Azure WebJob Project
Existing Project as Azure WebJob
Project Reference...
Shared Project Reference...
COM Reference...
Service Reference...
Connected Service
Class...
New EditorConfig
AspireApp11.ServiceDefaults
Package Manager: WebApplication1
Package source: nuget.org
Build
Rebuild
Clean
View
Analyze and Code Cleanup
Pack
Publish...
Upgrade
Configure Application Insights...
Overview
Collapse All Descendants Ctrl+Left Arrow
Scope to This
New Solution Explorer View
File Nesting
Edit Project File
Build Dependencies
Add
Manage NuGet Packages...
Manage Client-Side Libraries...
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'AspireApp1' (5 of 5 projects)
GitHub Actions
AspireApp1.AppHost
Dependencies
Properties
appsettings.json
Program.cs
AspireApp1.ServiceDefaults
Dependencies
Extensions.cs
AspireApp11.AppHost
Dependencies
Properties
appsettings.json
Program.cs
AspireApp11.ServiceDefaults
Dependencies
Extensions.cs
WebApplication1
Connected Services
Dependencies
Properties
Controllers
appsettings.json
Program.cs


Browse
Installed
Updates
owner:Aspire tags:component
Include prerelease
Aspire.StackExchange.Redis by Microsoft, 15.9K downloads
A generic Redis® client that integrates with Aspire, including health checks, logging, and telemetry.
8.0.0-preview.2.23619.3
Prerelease
Aspire.RabbitMQ.Client by Microsoft, 10.5K downloads
A RabbitMQ client that integrates with Aspire, including health checks, logging, and telemetry.
8.0.0-preview.2.23619.3
Prerelease
Aspire.Npgsql by Microsoft, 9.71K downloads
A PostgreSQL® client that integrates with Aspire, including health checks, metrics, logging, and telemetry.
8.0.0-preview.2.23619.3
Prerelease
Aspire.Npgsql.EntityFrameworkCore.PostgreSQL by Microsoft, 12.2K downloads
A PostgreSQL® provider for Entity Framework Core that integrates with Aspire, including connection pooling, health checks, logging, and telemetry.
8.0.0-preview.2.23619.3
Prerelease
Aspire.StackExchange.Redis.OutputCaching by Microsoft, 6.41K downloads
A Redis® implementation for ASP.NET Core Output Caching that integrates with Aspire, including health checks, logging, and telemetry.
8.0.0-preview.1.23557.2
8.0.0-preview.2.23619.3
Prerelease
Aspire.Azure.Storage.Blobs by Microsoft, 1.1K downloads
A client for Azure Blob Storage that integrates with Aspire, including health checks, logging and telemetry.
8.0.0-preview.2.23619.3
Prerelease
Aspire.StackExchange.Redis.DistributedCaching by Microsoft, 1.79K downloads
A Redis® implementation for IDistributedCache that integrates with Aspire, including health checks, logging, and telemetry.
8.0.0-preview.2.23619.3
Prerelease

.NET Aspire Component Name	Contains README	Public API	Configuration Schema	DI Services	Logging	Tracing	Metrics	Health Checks
Npgsql	✅	✅	✅	✅	✅	✅	✅	✅
Npgsql.EntityFrameworkCore.PostgreSQL	✅	✅	✅	✅	✅	✅	✅	✅
Microsoft.Azure.Cosmos	✅	✅	✅	✅	✅	✅	❌	❌
Microsoft.Data.SqlClient	✅	✅	✅	✅	❌	✅	✅	✅
Microsoft.EntityFramework.Cosmos	✅	✅	✅	✅	✅	✅	❌	❌
Microsoft.EntityFrameworkCore.SqlServer	✅	✅	✅	✅	✅	✅	✅	✅
MongoDB.Driver	✅	✅	✅	✅	✅	✅	❌	✅
Azure.AI.OpenAI	✅	✅	✅	✅	✅	✅	❌	❌
Azure.Data.Tables	✅	✅	✅	✅	✅	✅	❌	✅
Azure.Messaging.ServiceBus	✅	✅	✅	✅	✅	✅	❌	✅
Azure.Security.KeyVault	✅	✅	✅	✅	✅	✅	❌	✅
Azure.Storage.Blobs	✅	✅	✅	✅	✅	✅	❌	✅
Azure.Storage.Queues	✅	✅	✅	✅	✅	✅	❌	✅
StackExchange.Redis	✅	✅	✅	✅	✅	✅	❌	✅
Azure Redis								
StackExchange.Redis.DistributedCaching	✅	✅	N/A	✅	✅	✅	❌	✅
StackExchange.Redis.OutputCaching	✅	✅	N/A	✅	✅	✅	❌	✅
RabbitMQ	✅	✅	✅	✅			❌	✅
MySqlConnector	✅	✅	✅	✅	✅	✅	✅	✅
Oracle.EntityFrameworkCore	✅	✅	✅	✅	✅	✅	✅	✅
Confluent.Kafka	✅		✅	✅	✅	❌	✅	✅
Pomelo.EntityFrameworkCore.MySql	✅	✅	✅	✅	✅	✅	✅	✅

https://github.com/dotnet/aspire/blob/main/src/Components/Aspire_Components_Progress.md

Kafka as an example

```
dotnet add package Aspire.Confluent.Kafka
```

```
builder.AddKafkaProducer<string, string>(“purchase-events");
```

```
{
  "ConnectionStrings": {
    "purchase-events": “..."
  },
  "Aspire": {
    "Confluent": {
      "Kafka": {
        "Producer": {
          "HealthChecks": true,
          "Config": {
            "Acks": "All"
          }
        }
      }
    }
  }
}
```

appsettings.json

Kafka as an example

Basket Service

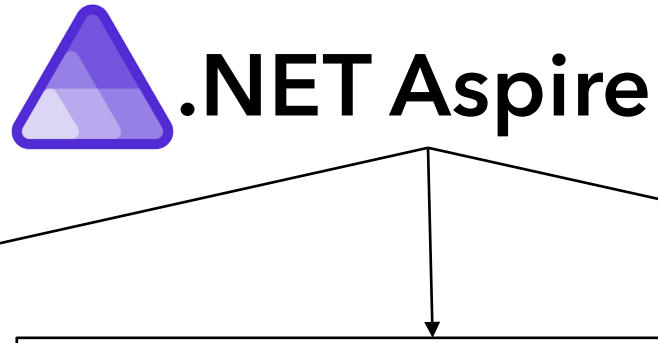
```
class MyWorker(IProducer<string, string> producer) : BackgroundService
{

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        var message = new Message<string, string>
        {
            Key = Guid.NewGuid().ToString(),
            Value = $"Hello, World!"
        };
        producer.Produce("topic", message);
        await Task.Delay(1000, stoppingToken).ConfigureAwait(false);
    }
}
```

AppHost

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

var kafka = appbuilder.AddKafka("purchase-events");

var basketService = appbuilder.AddProject("basketservice", @"..\BasketService\BasketService.csproj")
                              .WithReference(kafka);
appbuilder.Build().Run();
```

.NET Aspire

Opinionated, cloud ready stack for building observable, production ready, distributed applications. .NET



Composition & Orchestration

- Modeling language for defining the system resources and their dependencies
- A runtime to for running and connecting multi-project applications and their dependencies

Components

- Packages of commonly used services, such as Redis or Postgres, with standardized interfaces and capabilities

Tools

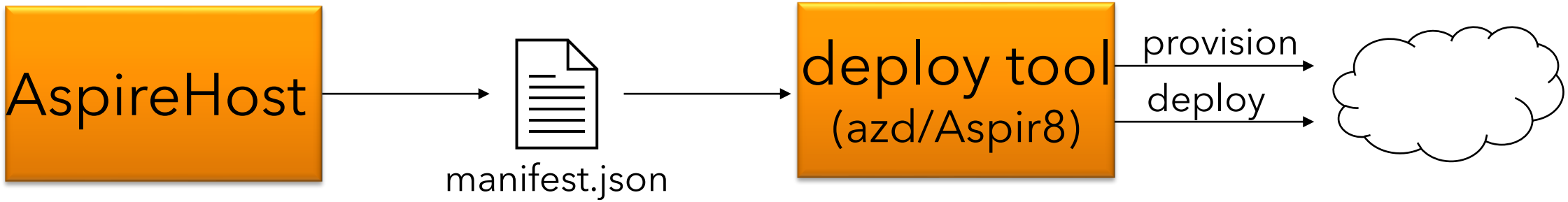
- Dashboard
- Distributed App VS support
- Service Defaults
- Publishing utils

Publishing & Deploying

Aspire Manifest

- Aspire project can generate a manifest that can later be used by other tools to deploy into an environment
  - Think of it as the IL for a distributed .NET app

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



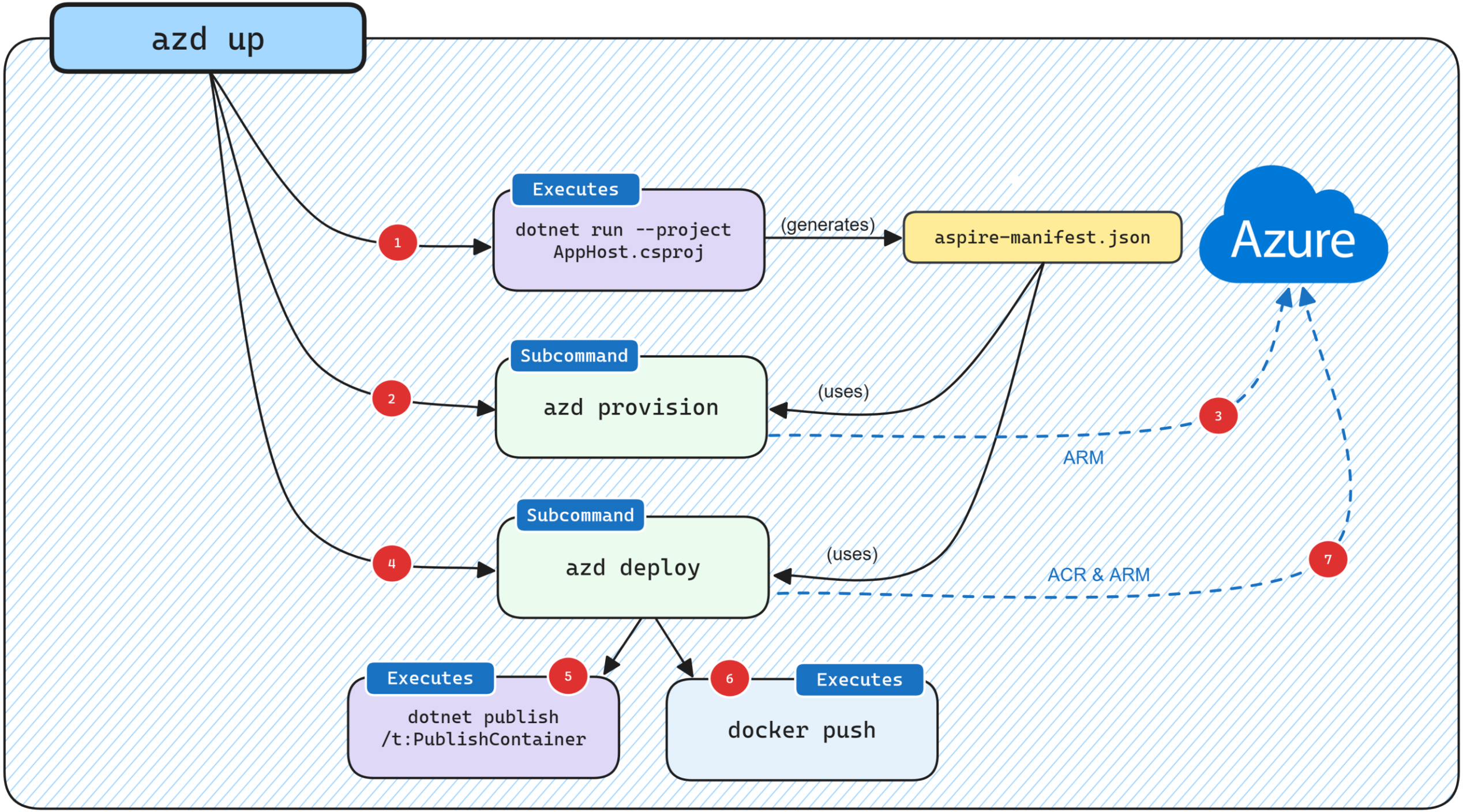
Aspire Manifest

- Aspire project can gene
  deploy into an environn
  - Think of it as the IL for a

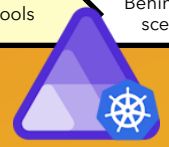
```
dotnet run --pr
--publisher mar
--output-path a
```

manifest.json

```
{
  "resources": {
    "cache": {
      "type": "container.v0",
      "image": "redis:latest",
      "bindings": {
      "connectionString": "{cache.bindings.tcp.host}:{cache.bindings.tcp.port}"
    },
    "apiservice": {
      "type": "project.v0",
      "path": "../AspireApp1.ApiService/AspireApp1.ApiService.csproj",
      "env": {
        "OTEL_DOTNET_EXPERIMENTAL_OTLP_EMIT_EXCEPTION_LOG_ATTRIBUTES": "true",
        "OTEL_DOTNET_EXPERIMENTAL_OTLP_EMIT_EVENT_LOG_ATTRIBUTES": "true"
      },
      "bindings": {
        "http": {
        "https": {
      }
    },
    "webfrontend": {
      "type": "project.v0",
      "path": "../AspireApp1.Web/AspireApp1.Web.csproj",
      "env": {
        "OTEL_DOTNET_EXPERIMENTAL_OTLP_EMIT_EXCEPTION_LOG_ATTRIBUTES": "true",
        "OTEL_DOTNET_EXPERIMENTAL_OTLP_EMIT_EVENT_LOG_ATTRIBUTES": "true",
        "ConnectionStrings__cache": "{cache.connectionString}",
        "services__apiservice__0": "{apiservice.bindings.http.url}",
        "services__apiservice__1": "{apiservice.bindings.https.url}"
      },
      "bindings": {
        "http": {
        "https": {
      }
    }
  }
}
```


azd up
1
Executes
dotnet run --project AppHost.csproj
(generates)
aspire-manifest.json
Azure
2
Subcommand
azd provision
(uses)
3
ARM
4
Subcommand
azd deploy
(uses)
7
ACR & ARM
5
Executes
dotnet publish /t:PublishContainer
6
Executes
docker push

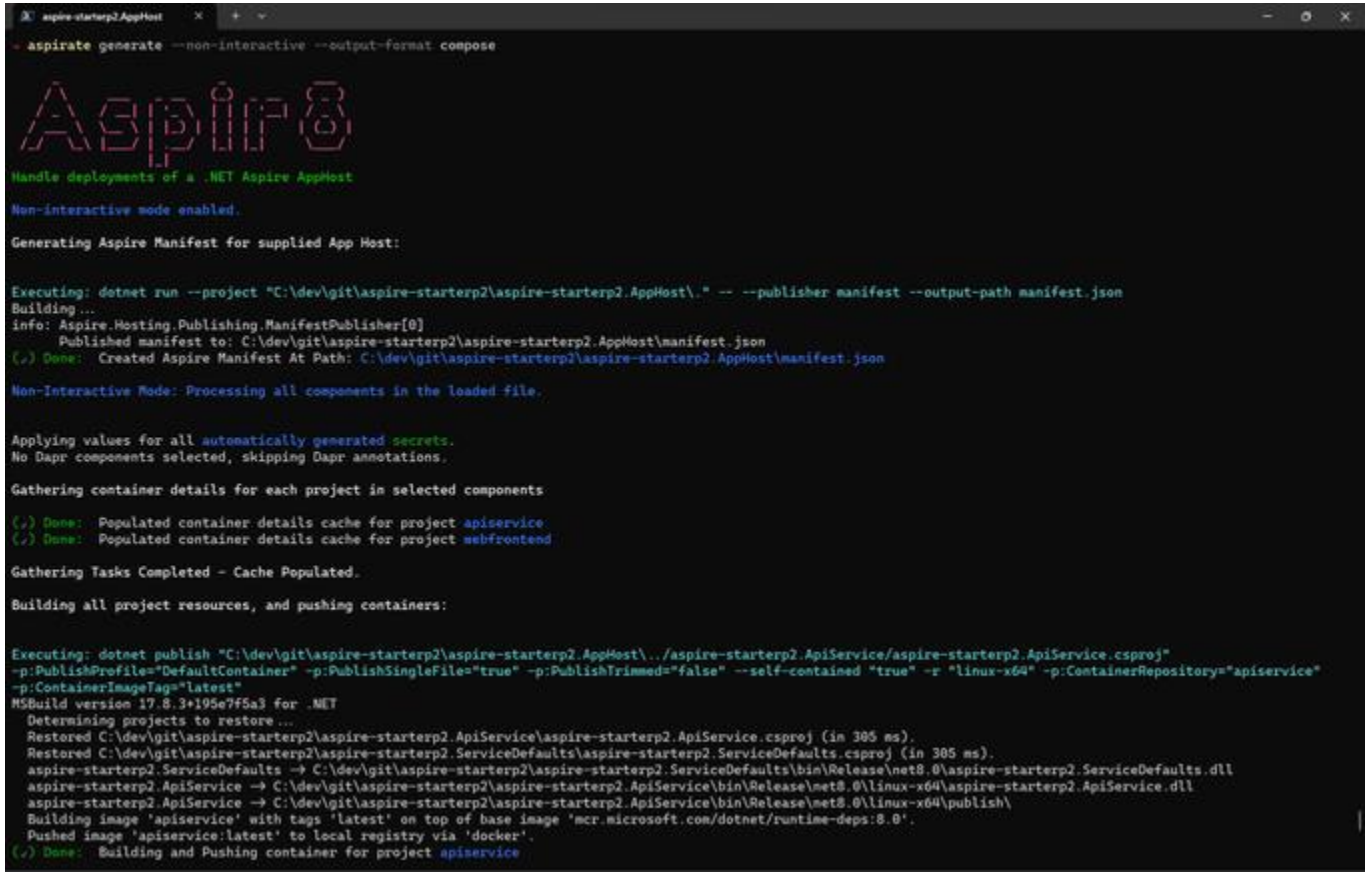
Deploy the manifest to K8s or Compose with Aspir8



- https://github.com/prom3theu5/aspirational-manifests

```
dotnet tool install -g aspirate --prerelease
```

Kustomize: `aspirate generate` Compose: `aspirate generate --output-format compose`



Service Defaults

Service Defaults - Standardizing services behavior

- Every service enlisted with Aspire will get automatic reference and call to
  - AddServiceDefaults()
  - MapDefaultEndpoints() – if a web project
- Out of the box services gets these capabilities
  - OpenTelemetry
  - Default HealthChecks
  - Service Discovery
  - Standard Resiliency configuration for HttpClient

Standardizing how our service should behave

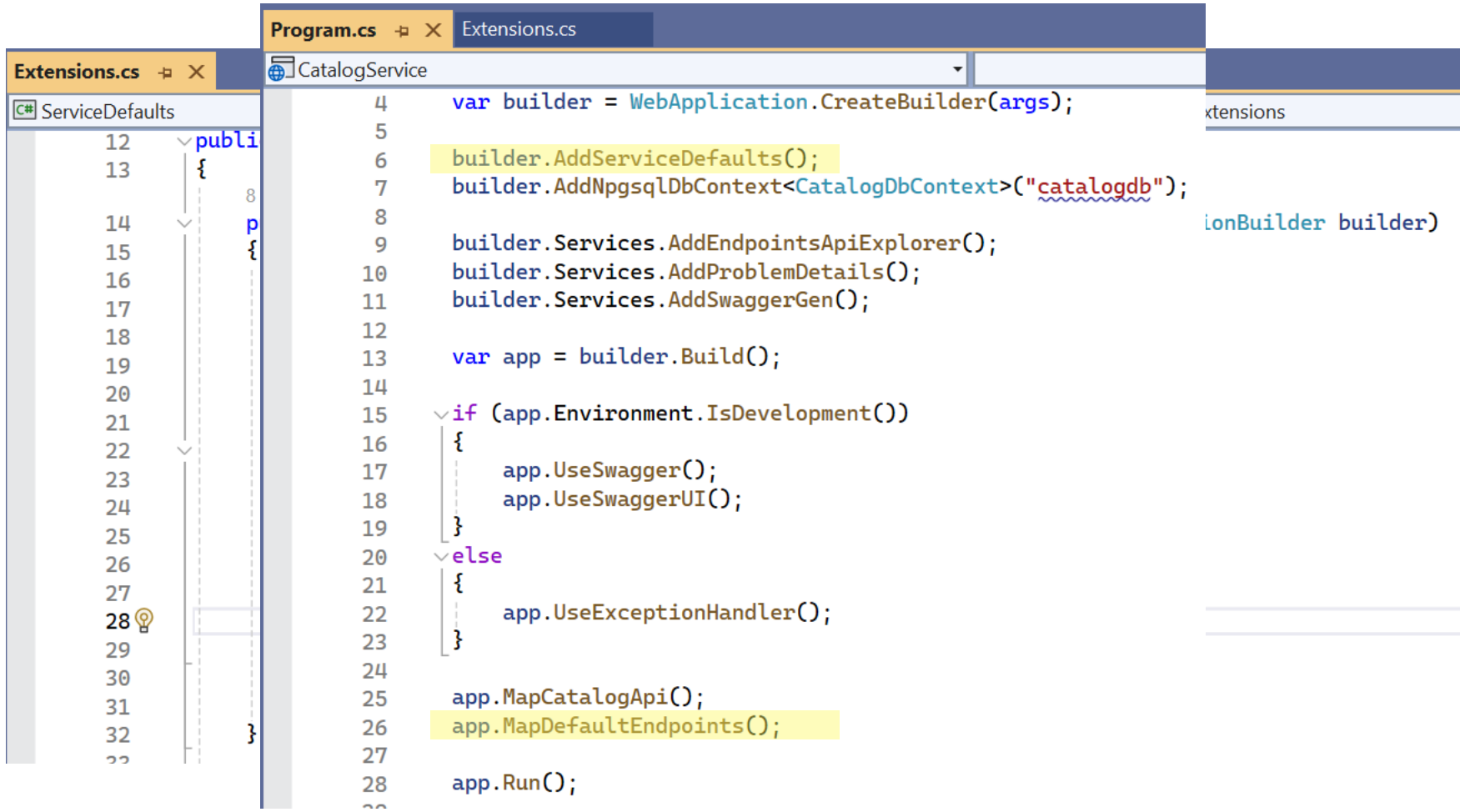
AspireApp1.ServiceDefaults

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

  <PropertyGroup>
    <OutputType>Library</OutputType>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <IsAspireSharedProject>true</IsAspireSharedProject>
  </PropertyGroup>

  <ItemGroup>
    <FrameworkReference Include="Microsoft.AspNetCore.App" />

    <PackageReference Include="Microsoft.Extensions.Http.Resilience" Version="8.1.0" />
    <PackageReference Include="Microsoft.Extensions.ServiceDiscovery" Version="8.0.0-preview.2.23619.3" />
    <PackageReference Include="OpenTelemetry.Exporter.OpenTelemetryProtocol" Version="1.7.0" />
    <PackageReference Include="OpenTelemetry.Extensions.Hosting" Version="1.7.0" />
    <PackageReference Include="OpenTelemetry.Instrumentation.AspNetCore" Version="1.7.0" />
    <PackageReference Include="OpenTelemetry.Instrumentation.GrpcNetClient" Version="1.6.0-beta.3" />
    <PackageReference Include="OpenTelemetry.Instrumentation.Http" Version="1.7.0" />
    <PackageReference Include="OpenTelemetry.Instrumentation.Runtime" Version="1.7.0" />
  </ItemGroup>

</Project>
```

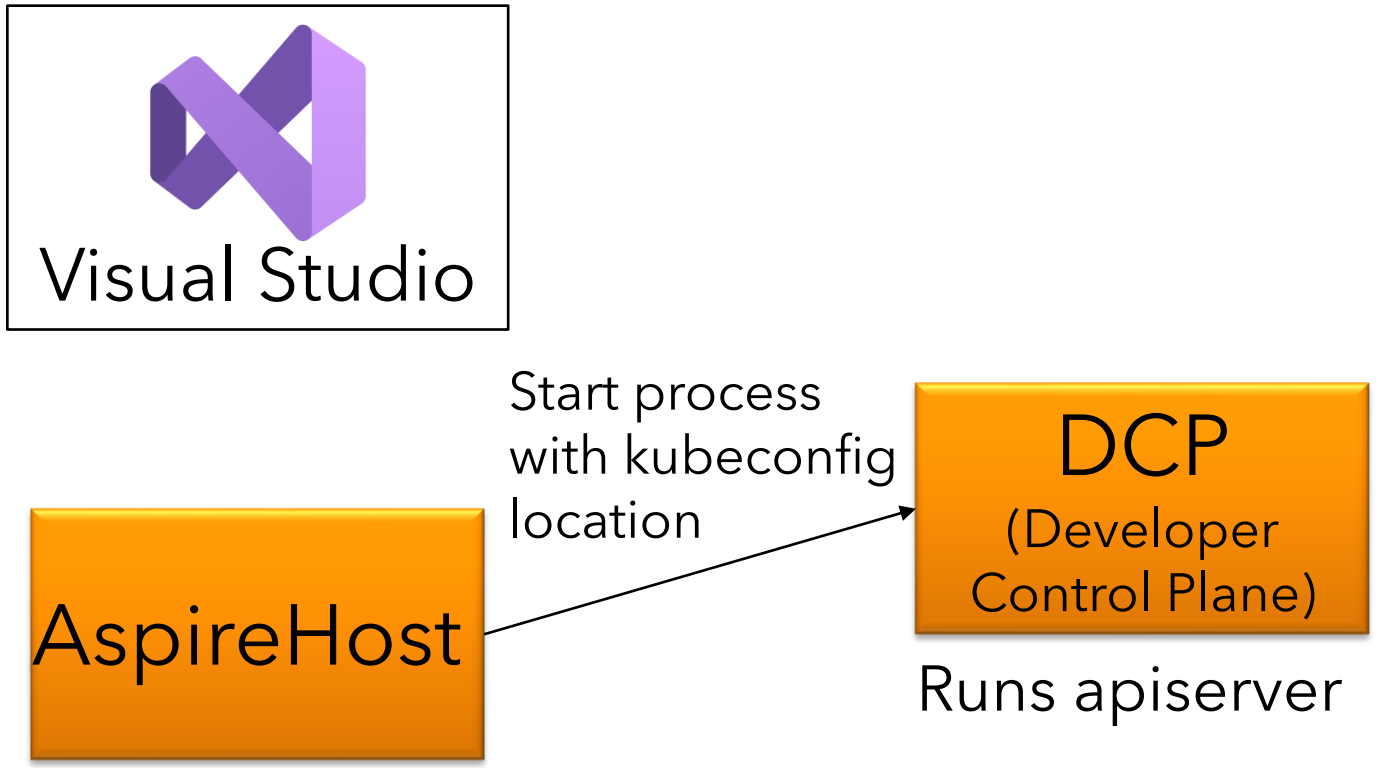
Extensions.cs

ServiceDefaults — Microsoft.Extensions.Hosting.Extensions

```
12  public static class Extensions
13  {
        8 references | Drew Noakes, 10 days ago | 1 author, 1 change
14      public static IHostApplicationBuilder AddServiceDefaults(this IHostApplicationBuilder builder)
15      {
16          builder.ConfigureOpenTelemetry();
17
18          builder.AddDefaultHealthChecks();
19
20          builder.Services.AddServiceDiscovery();
21
22          builder.Services.ConfigureHttpClientDefaults(http =>
23          {
24              // Turn on resilience by default
25              http.AddStandardResilienceHandler();
26
27              // Turn on service discovery by default
28              http.UseServiceDiscovery();
29          });
30
31          return builder;
32      }
```

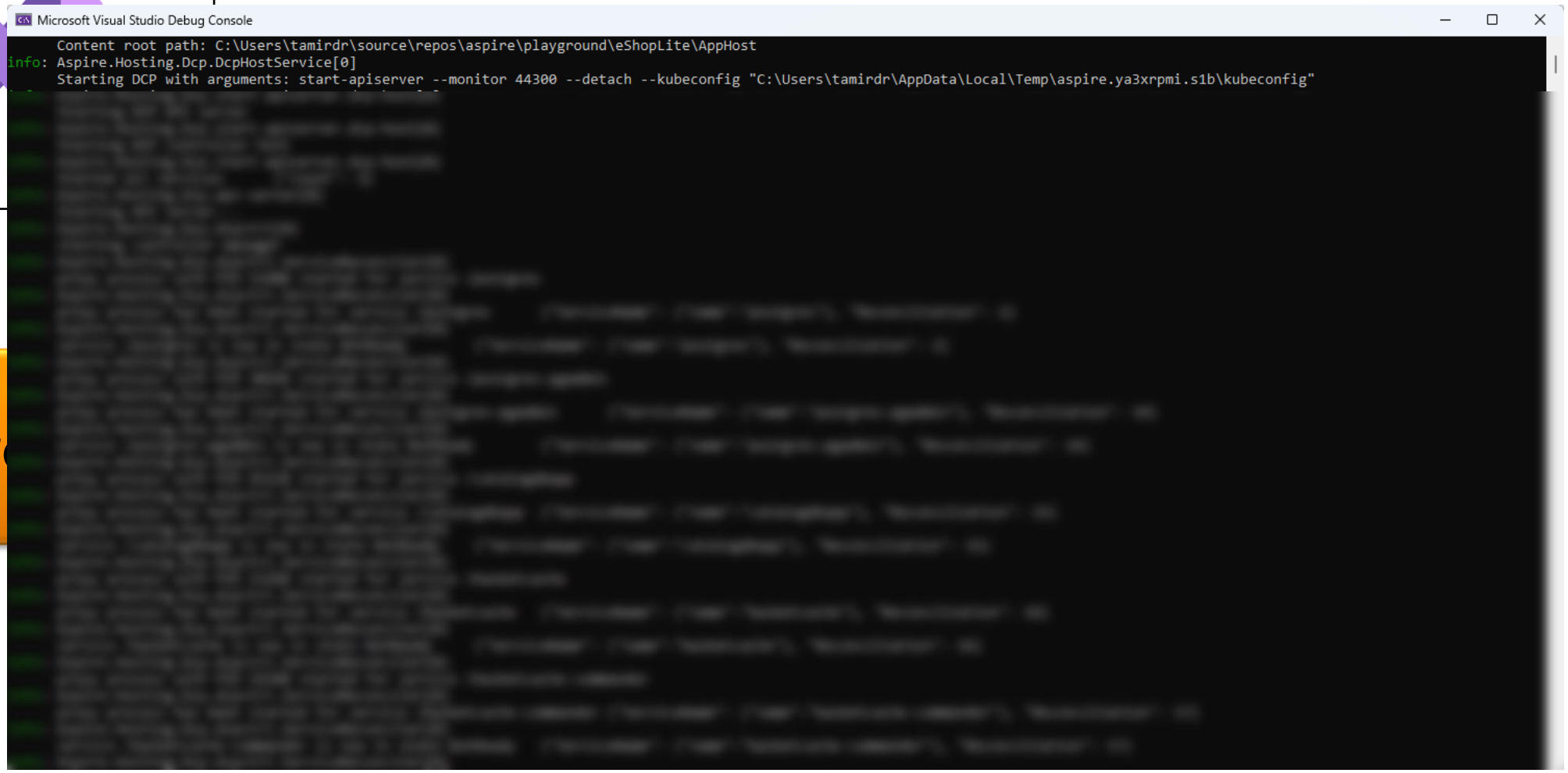

Extensions.cs
ServiceDefaults
Program.cs
Extensions.cs
CatalogService
var builder = WebApplication.CreateBuilder(args);
builder.AddServiceDefaults();
builder.AddNpgsqlDbContext<CatalogDbContext>("catalogdb");
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddProblemDetails();
builder.Services.AddSwaggerGen();
var app = builder.Build();
if (app.Environment.IsDevelopment())
{
app.UseSwagger();
app.UseSwaggerUI();
}
else
{
app.UseExceptionHandler();
}
app.MapCatalogApi();
app.MapDefaultEndpoints();
app.Run();

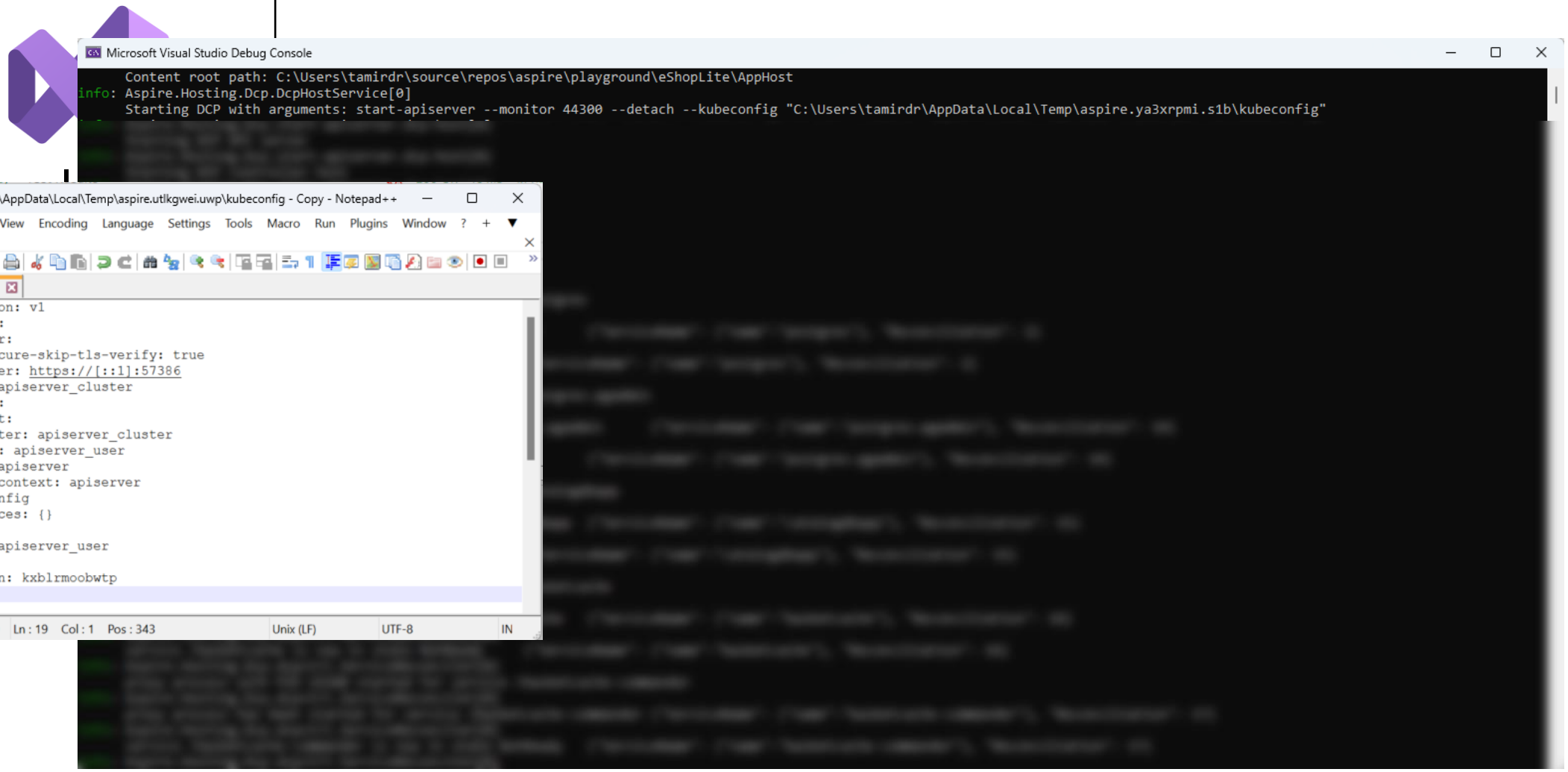
Behind the scenes & Under the covers


Visual Studio
Start process with kubeconfig location
DCP
(Developer Control Plane)
AspireHost
Runs apiserver

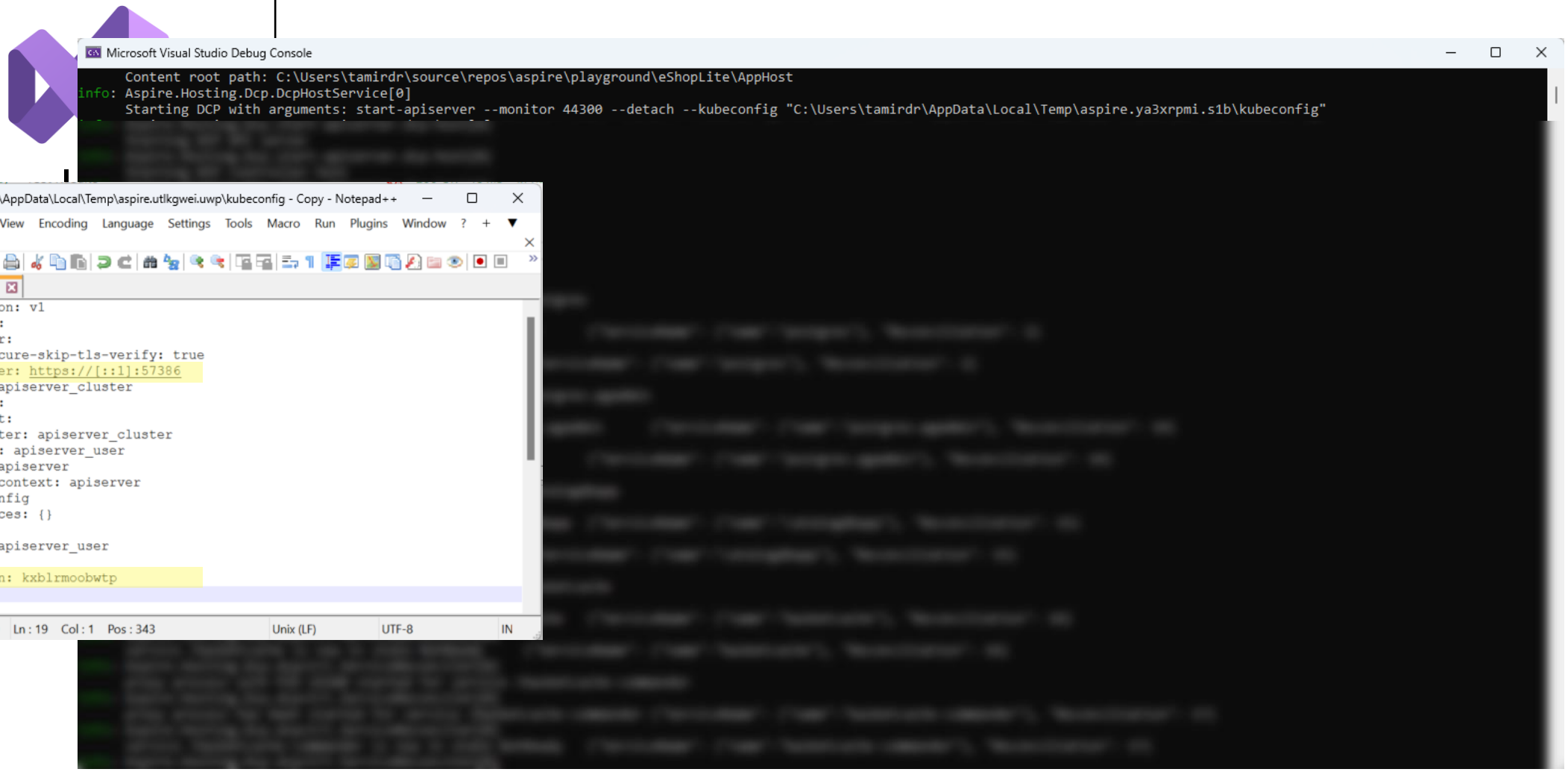
Visual

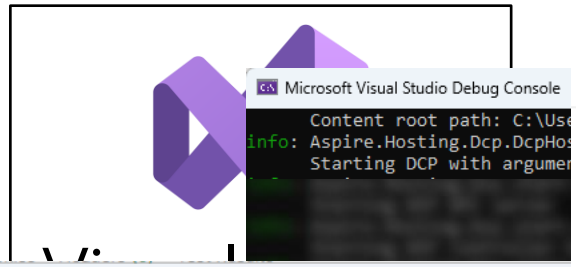
Aspir

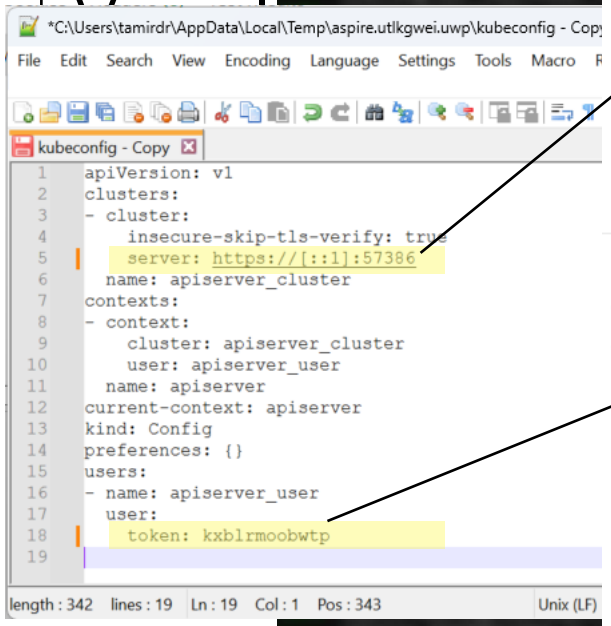



Microsoft Visual Studio Debug Console
Content root path: C:\Users\tamirdr\source\repos\aspire\playground\eShopLite\AppHost
info: Aspire.Hosting.Dcp.DcpHostService[0]
Starting DCP with arguments: start-apiserver --monitor 44300 --detach --kubeconfig "C:\Users\tamirdr\AppData\Local\Temp\aspire.ya3xrpmi.s1b\kubeconfig"
*C:\Users\tamirdr\AppData\Local\Temp\aspire.utlkgwei.uwp\kubeconfig - Copy - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? +
kubeconfig - Copy
apiVersion: v1
clusters:
- cluster:
insecure-skip-tls-verify: true
server: https://[::1]:57386
name: apiserver_cluster
contexts:
- context:
cluster: apiserver_cluster
user: apiserver_user
name: apiserver
current-context: apiserver
kind: Config
preferences: {}
users:
- name: apiserver_user
user:
token: kxblrmoobwtp
length : 342 lines : 19 Ln : 19 Col : 1 Pos : 343 Unix (LF) UTF-8 IN

Payoneer


Microsoft Visual Studio Debug Console
Content root path: C:\Users\tamirdr\source\repos\aspire\playground\eShopLite\AppHost
info: Aspire.Hosting.Dcp.DcpHostService[0]
Starting DCP with arguments: start-apiserver --monitor 44300 --detach --kubeconfig "C:\Users\tamirdr\AppData\Local\Temp\aspire.ya3xrpmi.s1b\kubeconfig"
*C:\Users\tamirdr\AppData\Local\Temp\aspire.utlkgwei.uwp\kubeconfig - Copy - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? +
kubeconfig - Copy
apiVersion: v1
clusters:
- cluster:
insecure-skip-tls-verify: true
server: https://[::1]:57386
name: apiserver_cluster
contexts:
- context:
cluster: apiserver_cluster
user: apiserver_user
name: apiserver
current-context: apiserver
kind: Config
preferences: {}
users:
- name: apiserver_user
user:
token: kxblrmoobwtp
length : 342 lines : 19 Ln : 19 Col : 1 Pos : 343 Unix (LF) UTF-8 IN





https://localhost:57386/

Save

GET https://localhost:57386/

Send

Params | Authorization | Headers (7) | Body | Pre-request Script | Tests | Settings | Cookies

Type: Bearer ...

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment,we recommend using variables. Learn more about variables.

Token: kxblrmoobwtp

Body | Cookies | Headers (5) | Test Results | 200 OK | 16 ms | 1.1 KB | Save as example

Pretty | Raw | Preview | Visualize | JSON


```
{
  "paths": [
    "/apis",
    "/apis/usvc-dev.developer.microsoft.com",
    "/apis/usvc-dev.developer.microsoft.com/v1",
    "/healthz",
    "/healthz/log",
    "/healthz/ping",
    "/healthz/poststarthook/max-in-flight-filter",
    "/healthz/poststarthook/start-tilt-server-informers",
    "/healthz/poststarthook/storage-object-count-tracker-hook",
    "/livez",
    "/livez/log",
    "/livez/ping",
    "/livez/poststarthook/max-in-flight-filter",
    "/livez/poststarthook/start-tilt-server-informers",
    "/livez/poststarthook/storage-object-count-tracker-hook",
    "/metrics",
    "/metrics/slis",
    "/openapi/v2",
    "/openapi/v3",
```


Payoneer

https://localhost:57386/

Save

https://localhost:57386/apis/usvc-dev.developer.microsoft.com/v1

Save

GET https://localhost:57386/apis/usvc-dev.developer.microsoft.com/v1 Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Type: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment,we recommend using variables. Learn more about variables.

Token: kxblrmoobwtp

*C:\Users\tamirdr\AppData\Loc

File Edit Search View Encod

kubeconfig - Copy

```
apiVersion: v1
clusters:
- cluster:
    insecure-skip
    server: https
  name: apiserver
contexts:
- context:
    cluster: apis
    user: apiserv
  name: apiserver
current-context:
kind: Config
preferences: {}
users:
- name: apiserver
  user:
    token: kxblrm
```

length : 342 lines : 19 Ln : 19

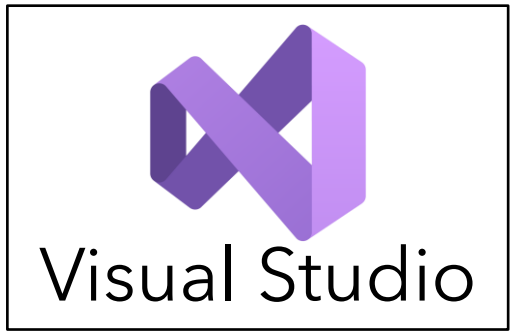
Body Cookies Headers (5) Test Results Status: 200 OK Time: 5 ms Size: 2 KB Save as example

Pretty Raw Preview Visualize JSON


```
{
    "kind": "APIResourceList",
    "apiVersion": "v1",
    "groupVersion": "usvc-dev.developer.microsoft.com/v1",
    "resources": [
        {
            "name": "containers",
            "singularName": "",
            "namespaced": false,
            "kind": "Container",
            "verbs": [
                "create",
                "delete",
                "deletecollection",
                "get",
                "list",
                "patch",
                "update",
                "watch"
            ],
            "shortNames": [
                "ctr"
            ]
        },
        {
            "name": "containers/status",
            "singularName": "",
            "namespaced": false,
            "kind": "Container",
            "verbs": [
                "get",
```

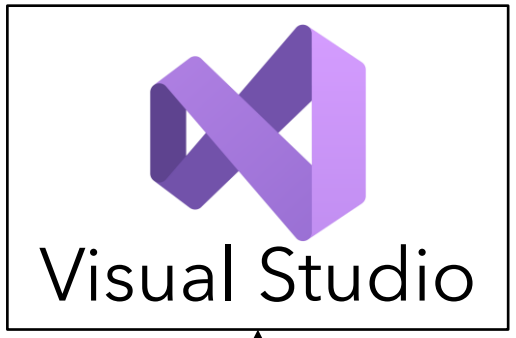

```
    "/livez/poststarthook/storage-object-count-tracker-hook",
    "/metrics",
    "/metrics/slis",
    "/openapi/v2",
    "/openapi/v3",
```

Payoneer



Details

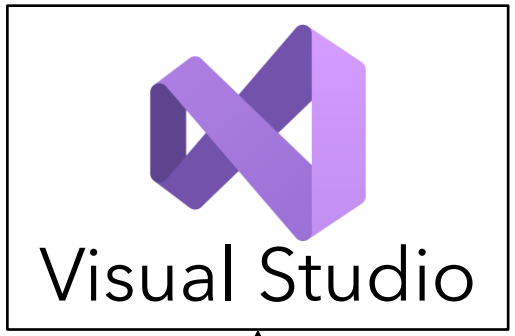
Name	PID	Command line
dcp.exe	33328	,dcp.exe" start-apiserver --kubeconfig C:\Users\tamirdr\AppData\Local\Temp\aspire.ml41mnfo.b0l\kubeconfig --monitor 51840 -v=debug
dcpctrl.exe	47616	,ext\dcpctrl.exe" run-controllers --kubeconfig C:\Users\tamirdr\AppData\Local\Temp\aspire.ml41mnfo.b0l\kubeconfig --monitor 33328 -v=debug
dcpd.exe	17688	,ext\dcpd.exe" --kubeconfig C:\Users\tamirdr\AppData\Local\Temp\aspire.ml41mnfo.b0l\kubeconfig --monitor 33328 -v=debug



Read projects info
+ launchProfiles

AspireHost

DCP
(Developer Control Plane)



Read pro
+ launch

AspireHost

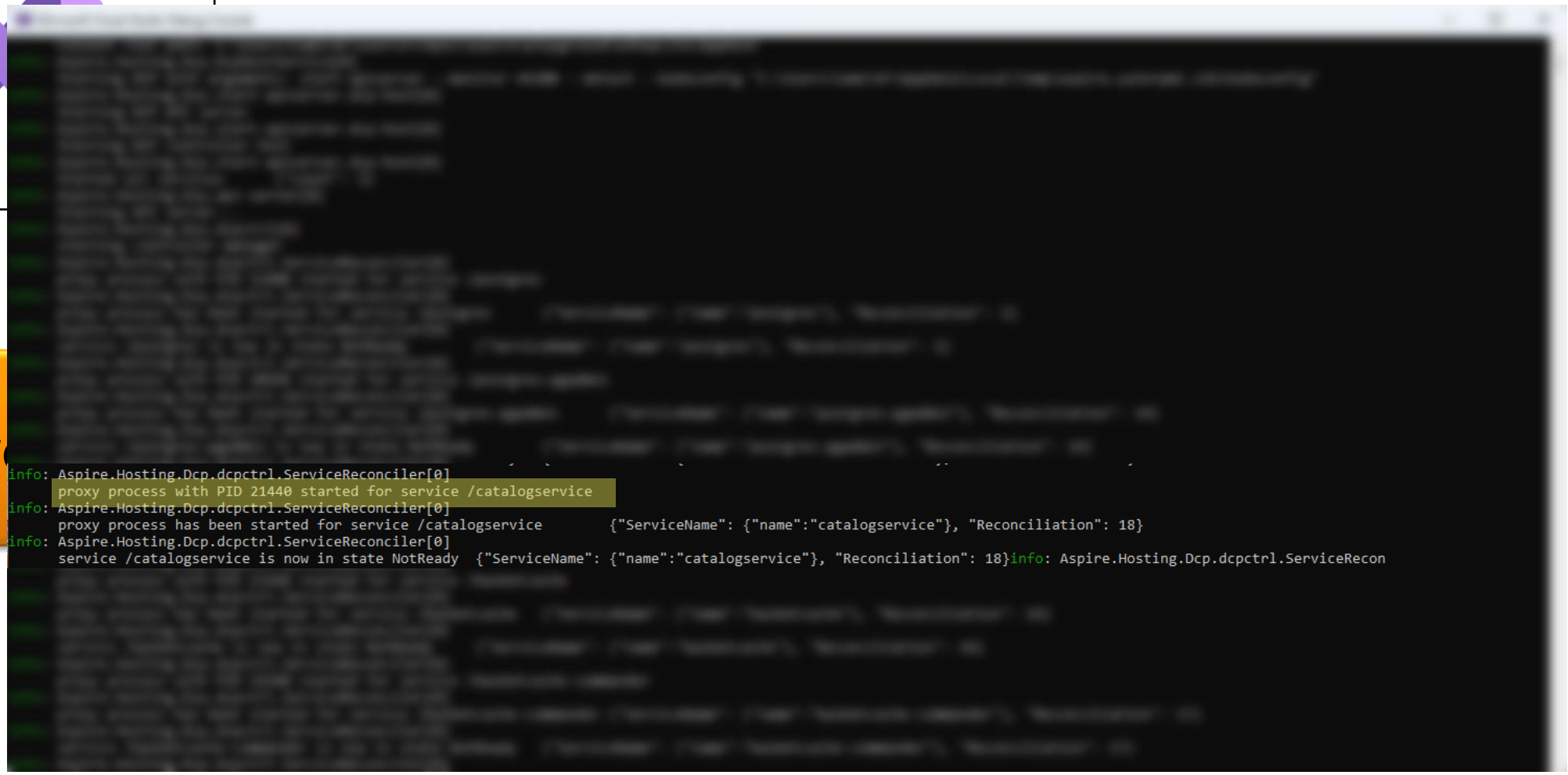
launchSettings.json

Schema: http://json.schemastore.org/launchsettings.json

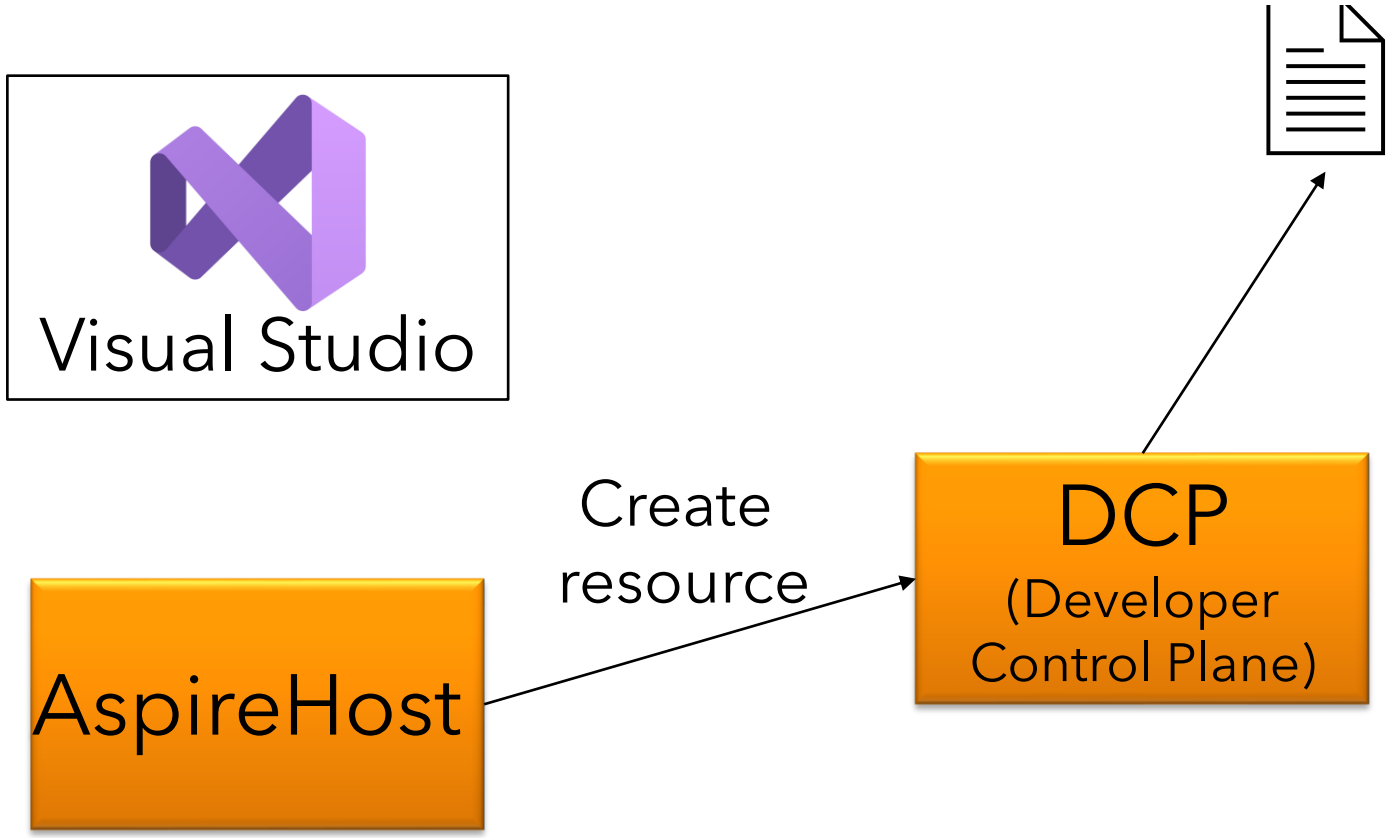
```
1  {
2    "$schema": "http://json.schemastore.org/launchsettings.json",
3    "profiles": {
4      "http": {
5        "commandName": "Project",
6        "dotnetRunMessages": true,
7        "launchBrowser": false,
8        "launchUrl": "swagger",
9        "applicationUrl": "http://localhost:5237",
10       "environmentVariables": {
11         "ASPNETCORE_ENVIRONMENT": "Development"
12       }
13     },
14     "https": {
15       "commandName": "Project",
16       "dotnetRunMessages": true,
17       "launchBrowser": false,
```

Visual

Aspir



%TEMP%\usvc-servicecontroller-serviceconfig*[resource]*.yaml



%TEMP%\usvc-servicecontroller-serviceconfig*[resource]*.yaml

Visual Stu

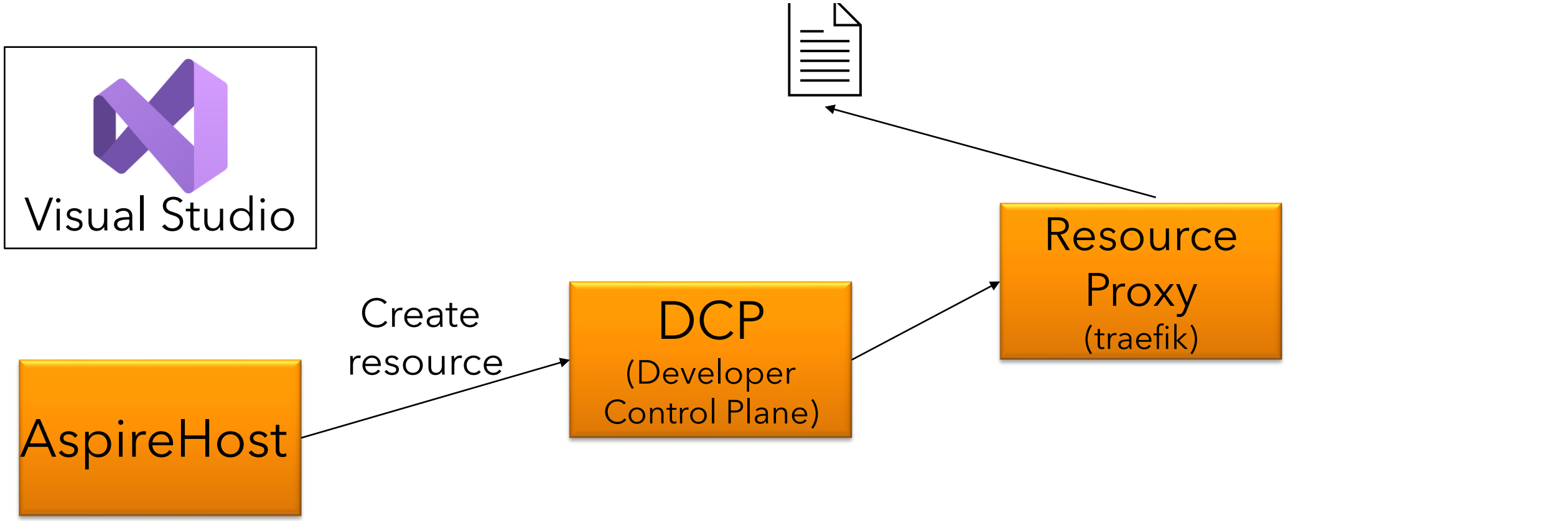
AspireHo

catalogservice - Copy (2).yaml

```yaml
tcp:
    routers:
        catalogservice:
            entryPoints:
                - web
                - webipv6
            rule: HostSNI(`*`)
            service: catalogservice
    services:
        catalogservice:
            loadBalancer:
                servers:
                    - address: localhost:51011
                    - address: localhost:51012
```

length : 365 lines : 15 Ln : 1 Col : 1 Pos : 1 Unix (LF) UTF-8 IN

%TEMP%\usvc-servicecontroller-serviceconfig\resource.yaml



%TEMP%\usvc-servicecontroller-serviceconfig\resource.yaml

Visual Studio

Resource Proxy

Create

DCP

Details

Name	PID	Command line
traefik.exe	21440	"C:\Program Files\dotnet\packs\Aspire.Hosting.Orchestration.win-x64\8.0.0-preview.1.23557.2\tools\ext\bin\traefik.exe" --entryPoints.web.address=127.0.0.1:5237 --entryPoints.webipv6.address=[::1]:5237 --providers.file.filename=C:\

AspireHost

%TEMP%\usvc-servicecontroller-serviceconfig\resource.yaml

Visual Studio

launchSettings.json

Schema: http://json.schemastore.org/launchsettings.json

```
1  {
2    "$schema": "http://json.schemastore.org/launchsettings.json",
3    "profiles": {
4      "http": {
5        "commandName": "Project",
6        "dotnetRunMessages": true,
7        "launchBrowser": false,
8        "launchUrl": "swagger",
9        "applicationUrl": "http://localhost:5237",
10       "environmentVariables": {
```

Details

Name	PID	Command line
traefik.exe	21440	"C:\Program Files\dotnet\packs\Aspire.Hosting.Orchestration.win-x64\8.0.0-preview.1.23557.2\tools\ext\bin\traefik.exe" --entryPoints.web.address=127.0.0.1:5237 --entryPoints.webipv6.address=[::1]:5237 --providers.file.filename=C:\

AspireHost

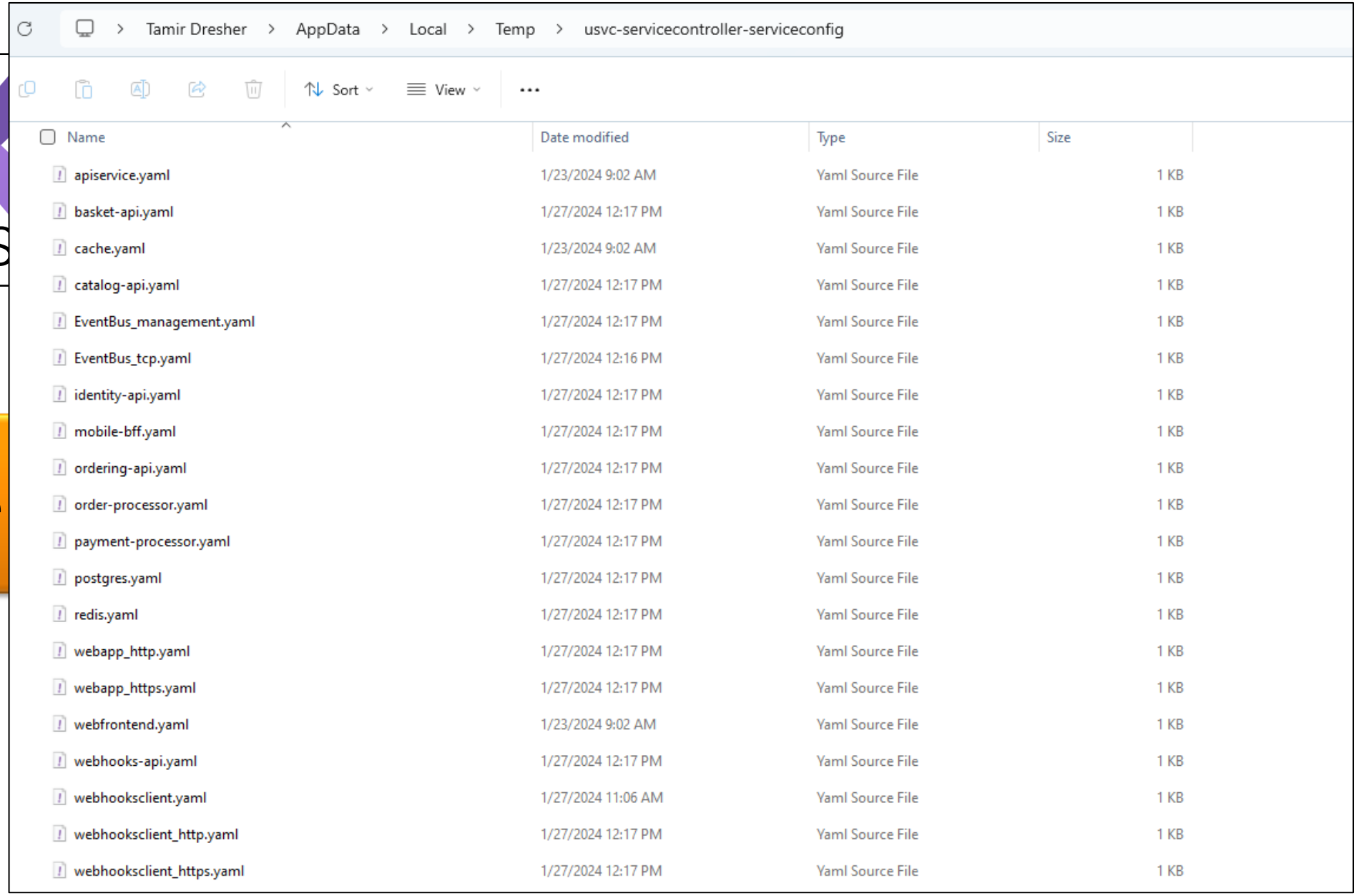
```
16       "dotnetRunMessages": true,
17       "launchBrowser": false,
```

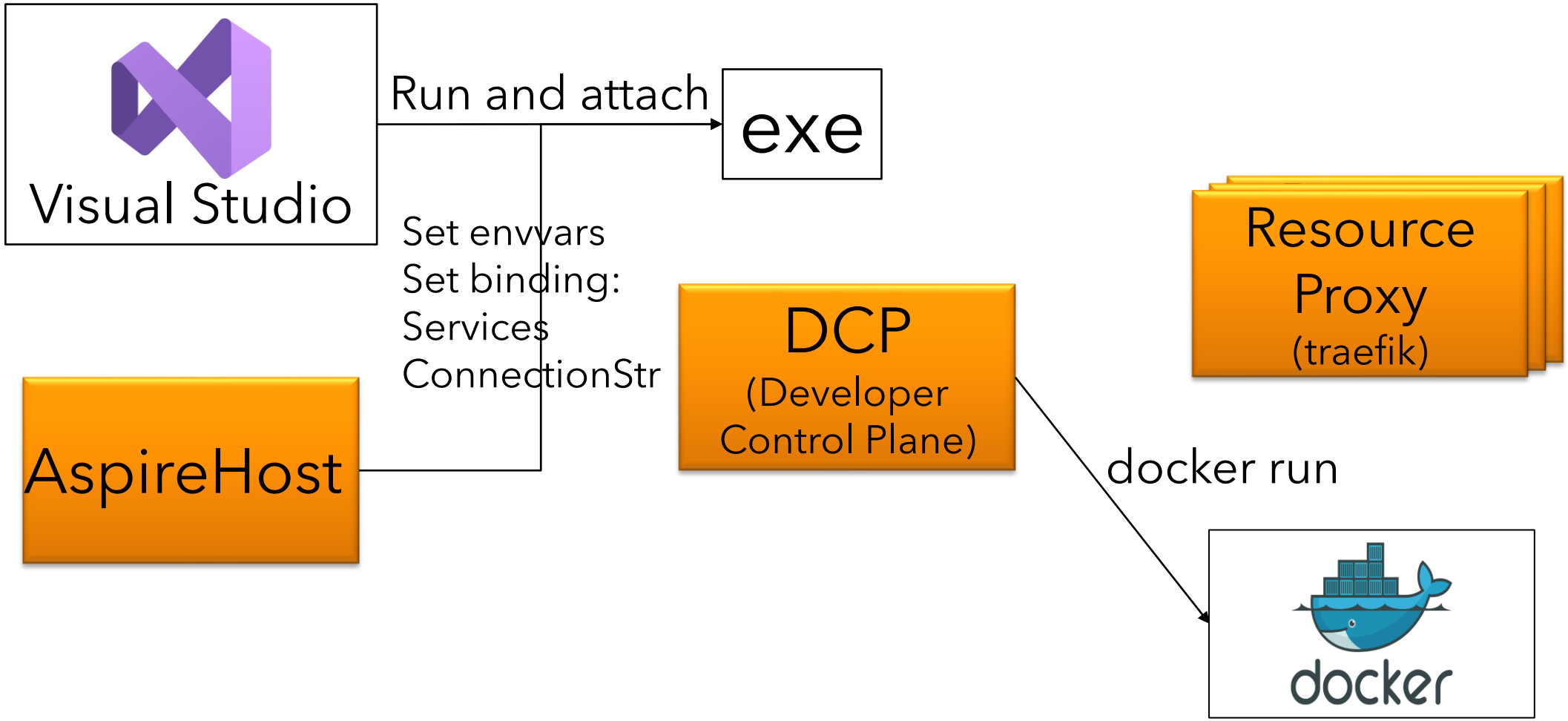
Payoneer

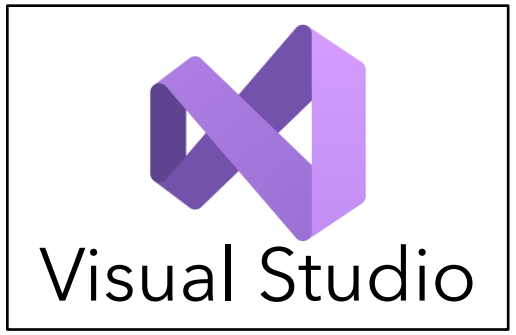
%TEMP%\usvc-servicecontroller-serviceconfig\resource.yaml

Visual S

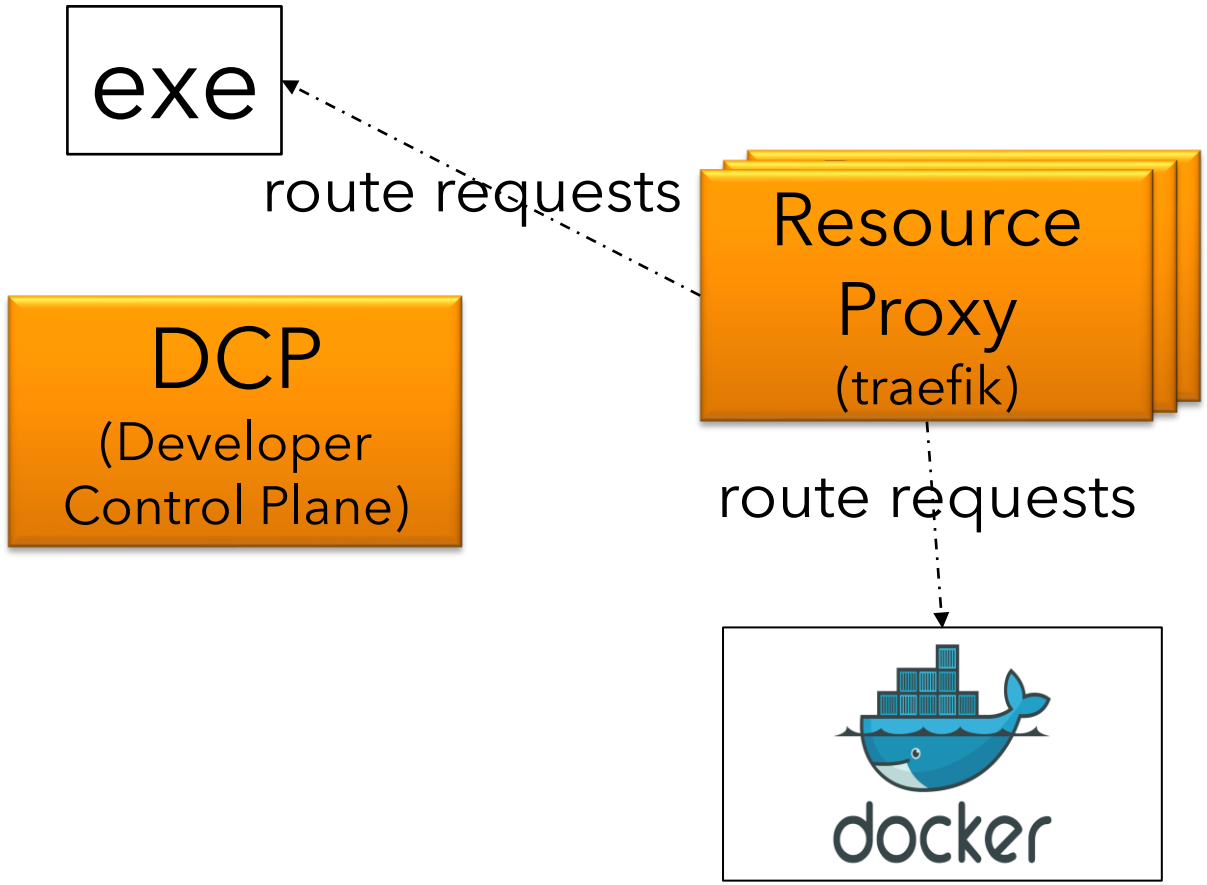
Aspire



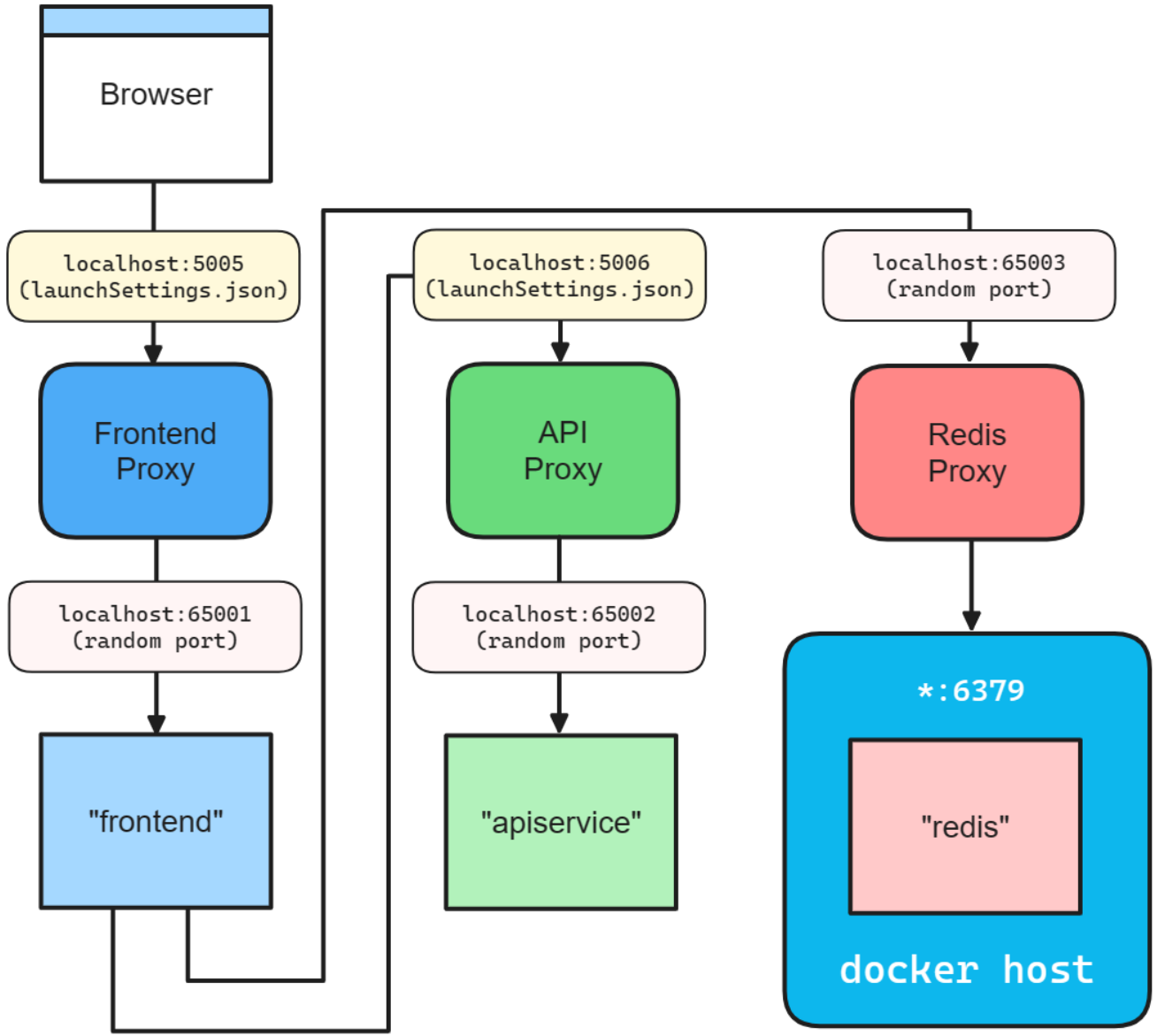

Visual Studio
Run and attach
exe
Set envvars
Set binding:
Services
ConnectionStr
AspireHost
DCP
(Developer
Control Plane)
Resource
Proxy
(traefik)
docker run
docker


Visual Studio

AspireHost


exe
route requests
Resource Proxy
(traefik)
DCP
(Developer Control Plane)
route requests
docker

Aspire networking



Payoneer

https://learn.microsoft.com/en-us/dotnet/aspire/fundamentals/networking-overview

.NET Aspire - Summary

Opinionated, cloud ready stack for building observable, production ready, distributed applications. .NET

Composition & Orchestration

- Modeling language for defining the system resources and their dependencies
- A runtime to for running and connecting multi-project applications and their dependencies

Components

- Packages of commonly used services, such as Redis or Postgres, with standardized interfaces and capabilities

Tools

- Dashboard
- Distributed App VS support
- Service Defaults
- Publishing utils