

Arquitectura de Microservicios con Dapr & Tye en .NET

Bienvenidos
Acerca de...



Jose María Flores Zazo, autor

*¡Hola! Gracias por entrar en
“Arquitectura de Microservicios con Dapr & Tye”.
Espero poder aportarte los conocimientos mínimos y necesarios
para que puedas ponerlo en práctica.*



Introducción

Preámbulo

Esta es la tercera y última parte de un ciclo de tres workshops dedicados a Microservicios, donde:

- I. **Arquitectura de Microservicios con contenedores usando .NET5**, hablaba del Proyecto Tye y daba ciertas explicaciones sobre que son los microservicios, así como varias tecnológicas a parte de los microservicios, tales como gRPC o WSL. Si no habéis realizado ese workshop el contenido lo puedes encontrar en:

- <https://github.com/jmfloreszazo/microservicesapp>
- <https://speakerdeck.com/jmfloreszazo/arquitectura-de-microservicios-con-contenedores-usando-net5>

- II. **Arquitectura de Microservicios con DARP en .NET**, donde te encuentras actualmente, trataremos de esta tecnología que nos permitirá trabajar con componentes distribuidos. Además de tocar Azure APIM para ahondar un poco en Arquitectura propiamente dicha.

- <https://github.com/jmfloreszazo/microservicesappdapr>
- <https://speakerdeck.com/jmfloreszazo/arquitectura-de-microservicios-con-dapr-en-net-59d865c5-61b6-431e-b6c1-aa962b44c427>

- III. **Arquitectura de Microservicios con DARP y Tye en .NET**, donde vamos a ver la simbiosis existente entre estas dos tecnologías. Tal y como he realizado en los anteriores, añadido un par más de tecnológica para que amplíes tus conocimientos. En esta ocasión vamos a desplegar a Kubernetes y monitorizar microservicios y a depurar con VS Code. Si no has realizado los anteriores workshops es posible que parte de lo que aquí veas no lo entiendas, por tanto, sigue el orden de publicación antes de continuar con este workshop.

- <https://github.com/jmfloreszazo/microservicesappdaprtye>
- <https://speakerdeck.com/jmfloreszazo/arquitectura-de-microservicios-con-dapr-tye-en-net>

Con estos tres workshops, ya tienes una base de herramientas y tecnologías que son el futuro de los microservicios.

Nota:

Antes de entrar en código, voy a recapitular ambas tecnologías base, por si acaso ayuda tras el periodo entre publicaciones estas pequeñas explicaciones.



Recordando que es Dapr

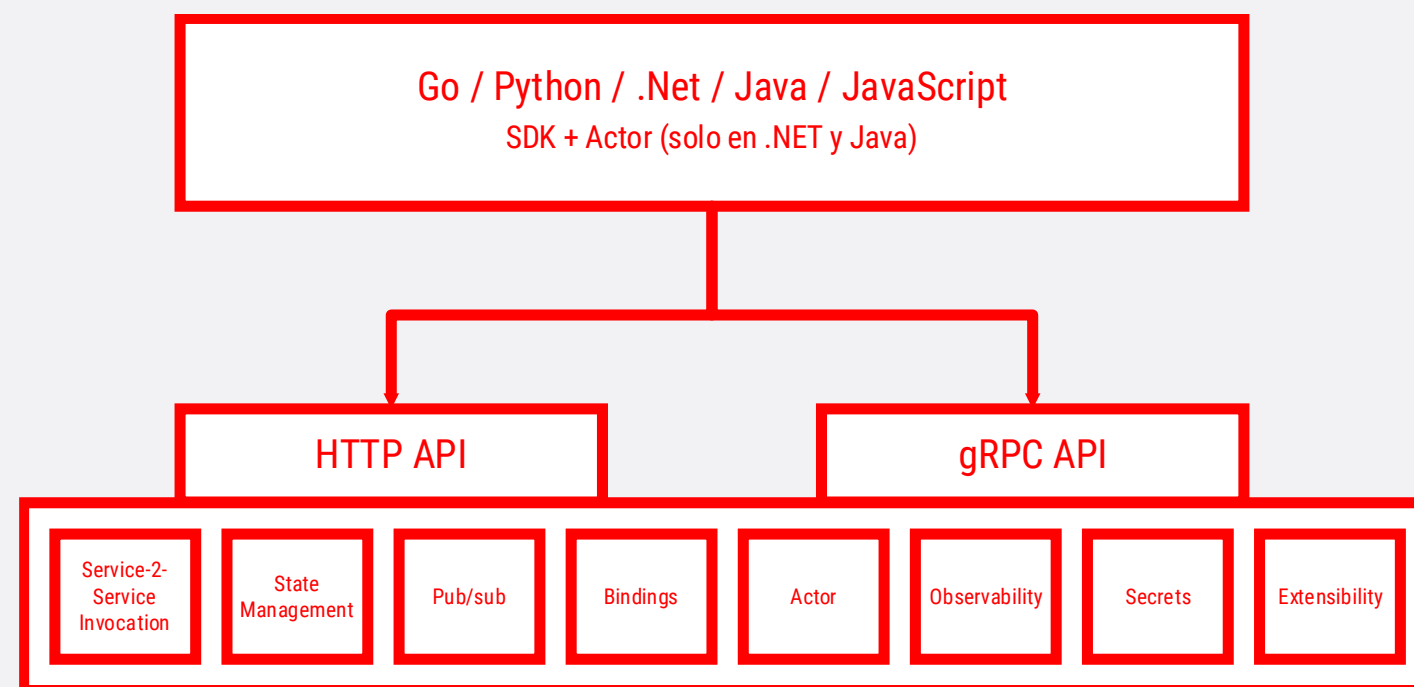
Sumario (1/4)

DAPR – Distributed Application Runtime

Dapr es una portable runtime orientado a event-driven, creado por Microsoft bajo open source que se encuentra ya en versión estable 1.0.0.

El estar dirigida por event-driven es clara su disposición a microservicios que reaccionan a eventos desde sistemas externos u otras piezas de la solución, también puede producir eventos para informar a otros servicios para continuar con el procesamiento de una etapa anterior.

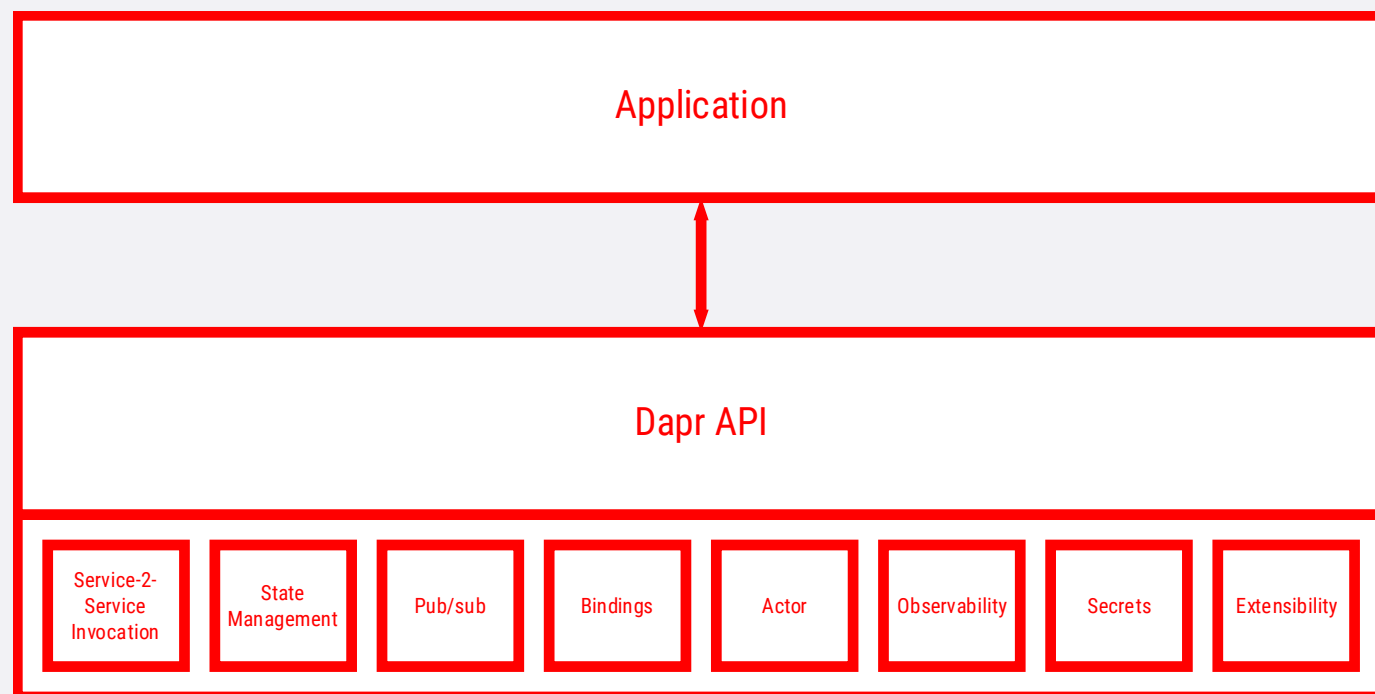
Dapr es portátil (portable) ya que puede ejecutarse de forma auto hospedada o en Kubernetes. El siguiente diagrama muestra la arquitectura de Dapr.



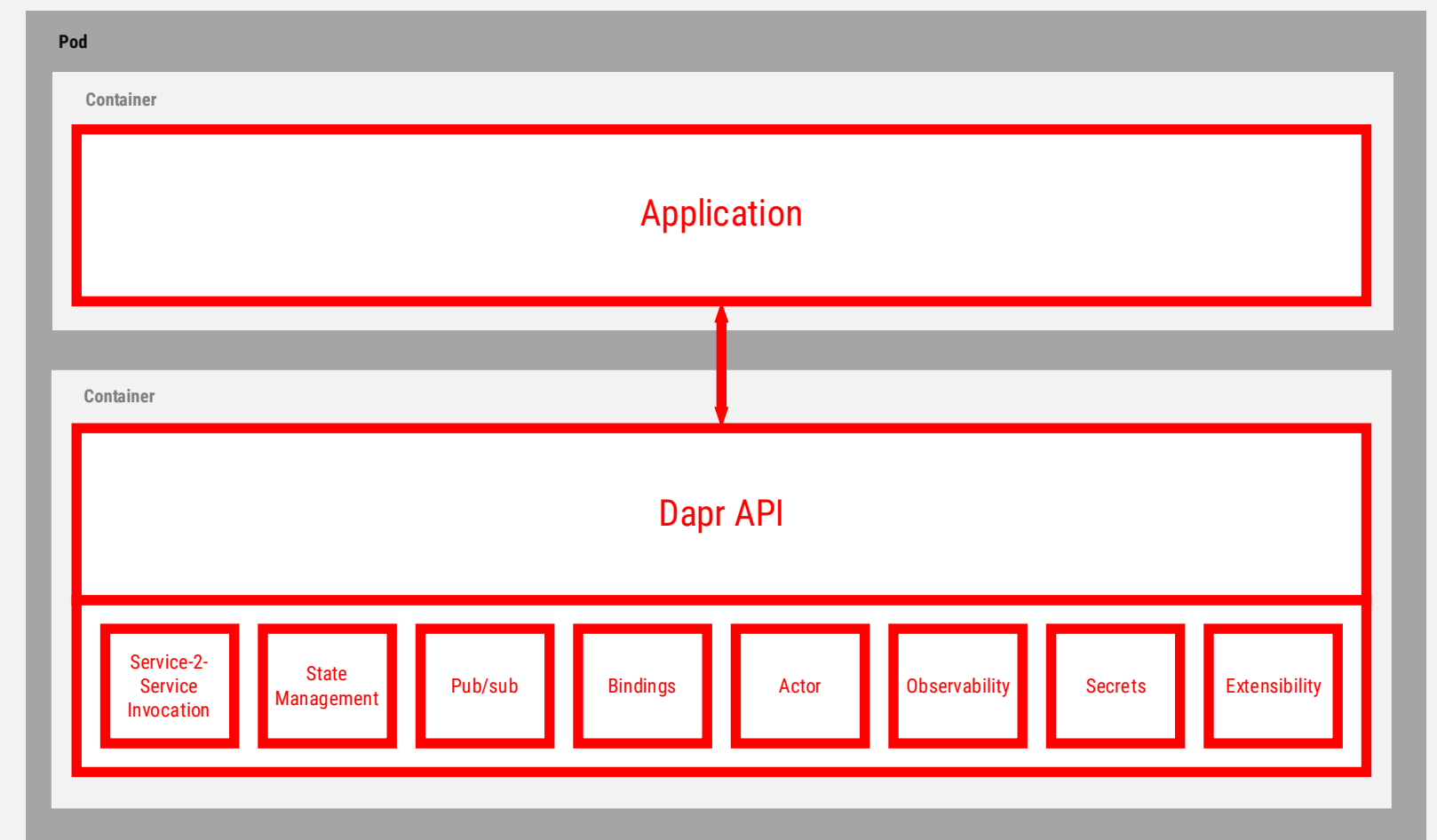
Recordando que es Dapr

Sumario (2/4)

La portabilidad nos permite modos de hospedaje:



Auto hospedado



Kubernetes



Recordando que no es Dapr

Sumario (3/4)

- El objetivo de Dapr no es obligar a un desarrollador a adoptar un modelo de programación con reglas y restricciones estrictas. Todo lo contrario, nos libera de complejidad en la arquitectura del microservicio, como son las conexiones a bases de datos (eso se convierte en responsabilidad de Dapr).
- Dapr no es una maya de servicios. Podrás encontrar similitudes entre services mesh y Dapr, pero la diferencia fundamental es que Dapr opera a nivel de servicio mientras que una Service mesh opera a nivel de infraestructura. Para vuestra información con Istio podemos integrar Dapr en una Service Mesh. En <https://docs.dapr.io/concepts/faq/> podéis ver que *"Istio is not a programming model and does not focus on application level features such as state management, pub-sub, bindings etc. That is where Dapr comes in"*.
- Y Dapr no es un servicio en la nube de Microsoft; ayuda a desarrollar aplicaciones de microservicios en la nube y proporciona muchas integraciones con servicios de Azure, pero de igual forma los tienes para AWS y GPC (ver tabla del workshop anterior).

Con este breve resumen de Dapr, mas toda la información del workshop anterior, ya tienes una visión más global de Dapr.



Recordando que es el Project Tye

Sumario (4/4)

Project Tye – <https://github.com/dotnet/tye>

Los desarrolladores de aplicaciones basadas en microservicios tenemos una complejidad más con la que enfrentarnos: lograr que la aplicación se ejecute.

Trabajar con muchos módulos con múltiples dependencias con diversas configuraciones, etc. implica un gasto de tiempo significativo.

Project Tye nos facilita el desarrollo, la prueba y la implementación de microservicios / aplicaciones distribuidas. Más o menos es un orquestador local que gracias a una mínima configuración nos ayuda incluso a desplegarlos en Kubernetes.

- Descubre los servicios a través de una convención de configuración, lo que nos permite exponer API con facilidad.
- Añade dependencias (Redis, MySQL, etc.) sin escribir ficheros de Docker.
- Ejecuta y depura de forma local usando contenedores o K8s.
- Dispone de un Dashboard para métricas, logging y depuración.
- Automáticamente dockeriza y despliega a Azure Kubernetes Service (AKS).

Y lo que realmente es interesante de Tye es que es muy liviano y la injerencia en el código es mínima.



Ejemplo E2E

Hello World! con Dapr (1/9)

En esta ocasión no vamos a montar una arquitectura sofisticada, esto lo podréis ver en el anterior workshop. Aquí solamente vamos a recordar como montar un proyecto con Dapr, la novedad será que solamente voy a usar VS Code.

Voy a suponer que ya tienes instalado el Dapr CLI, en caso contrario, por favor ve al anterior Workshop.

Comprobamos los requisitos iniciales:

```
Símbolo del sistema

C:\temp\microservicesappdaprt>dapr --version
CLI version: 1.0.1
Runtime version: 1.0.0

C:\temp\microservicesappdaprt>dotnet --info
SDK de .NET (que refleje cualquier global.json):
Version: 5.0.201
Commit: a09bd5c86c

Entorno de tiempo de ejecución:
OS Name: Windows
OS Version: 10.0.19042
OS Platform: Windows
RID: win10-x64
Base Path: C:\Program Files\dotnet\sdk\5.0.201\
```

Si esta todo en orden, podremos comenzar con el ejemplo.



Ejemplo E2E

Hello World! con Dapr (2/9)

Como vamos a trabajar con VS Code, debemos tener la siguiente extensión instalada:

<https://docs.dapr.io/developing-applications/ides/vscode>

Si debemos instalar Dapr en modo auto hospedado, ejecutar:

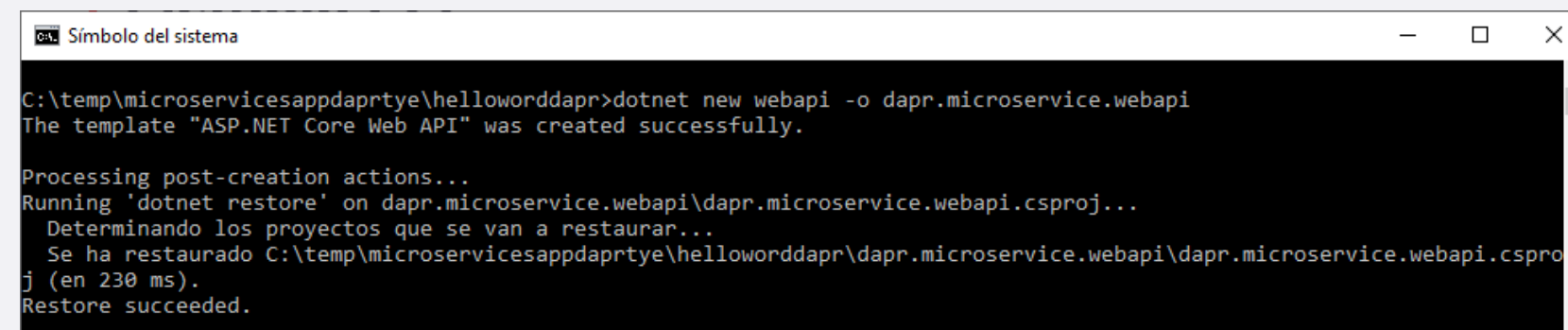
```
dapr init
```

Y para hacerlos en K8s:

```
dapr init -k
```

Vamos a construir el ejemplo:

```
dotnet new webapi -o dapr.microservice.webapi
```



```
Símbolo del sistema
C:\temp\microservicesappdaprt\helloworlddapr>dotnet new webapi -o dapr.microservice.webapi
The template "ASP.NET Core Web API" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on dapr.microservice.webapi\dapr.microservice.webapi.csproj...
  Determinando los proyectos que se van a restaurar...
  Se ha restaurado C:\temp\microservicesappdaprt\helloworlddapr\dapr.microservice.webapi\dapr.microservice.webapi.csproj (en 230 ms).
Restore succeeded.
```



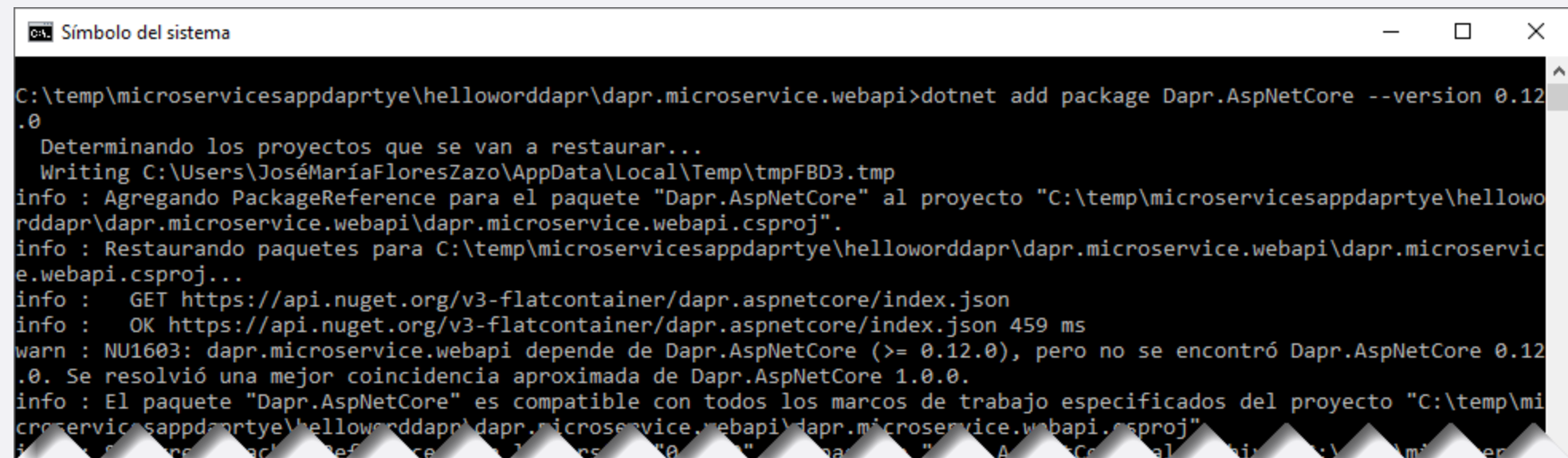
Ejemplo E2E

Hello World! con Dapr (3/9)

Vamos con el primer ejemplo (marcado en el repo como ejemplo1).

Añadimos la referencia:

```
dotnet add package Dapr.AspNetCore --version 1.0.0
```



```
C:\temp\microservicesappdaprt\helloworlddapr\dapr.microservice.webapi>dotnet add package Dapr.AspNetCore --version 0.12.0
Determinando los proyectos que se van a restaurar...
Writing C:\Users\JoséMaríaFloresZazo\AppData\Local\Temp\tmpFBD3.tmp
info : Agregando PackageReference para el paquete "Dapr.AspNetCore" al proyecto "C:\temp\microservicesappdaprt\helloworlddapr\dapr.microservice.webapi\dapr.microservice.webapi.csproj".
info : Restaurando paquetes para C:\temp\microservicesappdaprt\helloworlddapr\dapr.microservice.webapi\dapr.microservice.webapi.csproj...
info :   GET https://api.nuget.org/v3-flatcontainer/dapr.aspnetcore/index.json
info :   OK https://api.nuget.org/v3-flatcontainer/dapr.aspnetcore/index.json 459 ms
warn : NU1603: dapr.microservice.webapi depende de Dapr.AspNetCore (>= 0.12.0), pero no se encontró Dapr.AspNetCore 0.12.0. Se resolvió una mejor coincidencia aproximada de Dapr.AspNetCore 1.0.0.
info : El paquete "Dapr.AspNetCore" es compatible con todos los marcos de trabajo especificados del proyecto "C:\temp\microservicesappdaprt\helloworlddapr\dapr.microservice.webapi\dapr.microservice.webapi.csproj".
info : El paquete "Dapr.AspNetCore" se agregó al proyecto "C:\temp\microservicesappdaprt\helloworlddapr\dapr.microservice.webapi\dapr.microservice.webapi.csproj".
```

Y entramos en VS Code.



Ejemplo E2E

Hello World! con Dapr (4/9)

Para añadir Dapr a nuestro proyecto solamente debemos agregar las siguientes instrucciones:

```
// This method gets called by the runtime. Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers().AddDapr();
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new OpenApiInfo { Title = "dapr.microservice.webapi", Version = "v1" });
    });
}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
0 references
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseSwagger();
        app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "dapr.microservice.webapi v1"));
    }

    app.UseHttpsRedirection();

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapSubscribeHandler();
        endpoints.MapControllers();
    });
}
```



Ejemplo E2E

Hello World! con Dapr (5/9)

Y vamos a escribir nuestra API:

```
C# Program.cs C# Startup.cs C# HelloWorldController.cs X
Controllers > C# HelloWorldController.cs > {} dapr.microservice.webapi.Controllers
1  using Dapr;
2  using Microsoft.AspNetCore.Mvc;
3  using System;
4
5  namespace dapr.microservice.webapi.Controllers
6  {
7      [ApiController]
8      0 references
9      public class HelloWorldController : ControllerBase
10     {
11         [HttpGet("hello")]
12         0 references
13         public ActionResult<string> Get()
14         {
15             var message = "Hello, World!";
16             Console.WriteLine(message);
17             return message;
18         }
19     }
```

El atributo `[HttpGet("hello")]` es un requerimiento de Dapr, le servirá para identificar el nombre del método.



Ejemplo E2E

Hello World! con Dapr (6/9)

Ejecutamos nuestra aplicación en Dapr:

```
dapr run -a hello-world -p 5000 -H 5010 dotnet run
```

```
Selecionar Símbolo del sistema - dapr run -a hello-world -p 5000 -H 5010 dotnet run

C:\temp\microservicesapp\daprt\helloworld\dapr.microservice.webapi>code .

C:\temp\microservicesapp\daprt\helloworld\dapr.microservice.webapi>dapr run -a hello-world -p 5000 -H 5010 dotnet run
Starting Dapr with id hello-world. HTTP Port: 5010. gRPC Port: 50351
time="2021-03-26T09:14:43.1009598+01:00" level=info msg="starting Dapr Runtime --
time="2021-03-26T09:14:43.1119647+01:00" level=info msg="log level set to: info" a
time="2021-03-26T09:14:43.1229641+01:00" level=info msg="metrics server started on
time="2021-03-26T09:14:43.1879609+01:00" level=info msg="standalone mode configure
time="2021-03-26T09:14:43.1879609+01:00" level=info msg="app id: hello-world" app_
time="2021-03-26T09:14:43.189963+01:00" level=info msg="mTLS is disabled. Skipping
time="2021-03-26T09:14:43.252962+01:00" level=info msg="local service entry announ
time="2021-03-26T09:14:43.2549618+01:00" level=info msg="Initialized name resoluti
time="2021-03-26T09:14:43.3369645+01:00" level=info msg="component loaded. name: p
time="2021-03-26T09:14:43.3369645+01:00" level=info msg="waiting for all outstandi
time="2021-03-26T09:14:43.351963+01:00" level=info msg="component loaded. name: st
time="2021-03-26T09:14:43.351963+01:00" level=info msg="all outstanding components
time="2021-03-26T09:14:43.3619603+01:00" level=info msg="enabled gRPC tracing midd
time="2021-03-26T09:14:43.3639607+01:00" level=info msg="enabled gRPC metrics midd
time="2021-03-26T09:14:43.366959+01:00" level=info msg="API gRPC server is running
time="2021-03-26T09:14:43.3809614+01:00" level=info msg="enabled metrics http midd
time="2021-03-26T09:14:43.3819593+01:00" level=info msg="enabled tracing http midd
time="2021-03-26T09:14:43.3829624+01:00" level=info msg="http server is running on
time="2021-03-26T09:14:43.3839616+01:00" level=info msg="The request body size par
time="2021-03-26T09:14:43.3859648+01:00" level=info msg="enabled gRPC tracing midd
time="2021-03-26T09:14:43.3909605+01:00" level=info msg="enabled gRPC metrics midd
time="2021-03-26T09:14:43.391963+01:00" level=info msg="internal gRPC server is ru
time="2021-03-26T09:14:43.391963+01:00" level=info msg="application protocol: http
pe=dapr.runtime type=log ver=1.0.0-rc.3
Updating metadata for app command: dotnet run
You're up and running! Both Dapr and your app logs will appear here.

== APP == info: Microsoft.Hosting.Lifetime[0]
== APP ==      Now listening on: https://localhost:5001
== APP == info: Microsoft.Hosting.Lifetime[0]
== APP ==      Now listening on: http://localhost:5000
== APP == info: Microsoft.Hosting.Lifetime[0]
```



Ejemplo E2E

Hello World! con Dapr (7/9)

Y probamos la aplicación:

Dapr - Demo / http://localhost:5010/v1.0/invoke/hello-world/method/hello

GET http://localhost:5010/v1.0/invoke/hello-world/method/hello Send

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

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text

1 Hello, World

⌚ Status: 200 OK Time: 691 ms Size: 216 B Save Response

```
time="2021-03-26T09:14:54.3742075+01:00" level=info msg="application discovered on port 5000" app_id=hello-world instance=DES
time="2021-03-26T09:14:54.5346137+01:00" level=info msg="application configuration loaded" app_id=hello-world instance=DES
time="2021-03-26T09:14:54.5778917+01:00" level=info msg="actor runtime started. actor idle timeout: 1h0m0s. actor scan int
ver=1.0.0-rc.3
time="2021-03-26T09:14:54.704891+01:00" level=info msg="dapr initialized. Status: Running. Init Elapsed 11494.9304ms" app_
time="2021-03-26T09:14:54.949075+01:00" level=info msg="placement tables updated, version: 0" app_id=hello-world instance=
== APP == Hello, World.
```



Ejemplo E2E

Hello World! con Dapr (8/9)

Dapr nos proporciona un Dashboard:

<http://localhost:8080/overview>

The screenshot displays the Dapr Dashboard interface. The top section shows the 'daprdashboard' logo and the title 'Dashboard'. Below this, a sidebar on the left contains navigation icons: a home icon, a grid icon, and a key icon. The main content area is divided into two sections. The top section, titled 'Application: hello-world', shows a 'Summary' tab with the following details:

Property	Value
App ID	hello-world
App Port	5000
Dapr HTTP Port	5010
Dapr gRPC Port	57505
Command	dotnet run
Replicas	1
Address	localhost:5010
PID	5392
Created	53s
Age	53s

The bottom section, titled 'Dapr Components', shows a table with the following data:

Name	Type	Age	Created
pubsub	pubsub.redis	3d	
statestore	state.redis	3d	



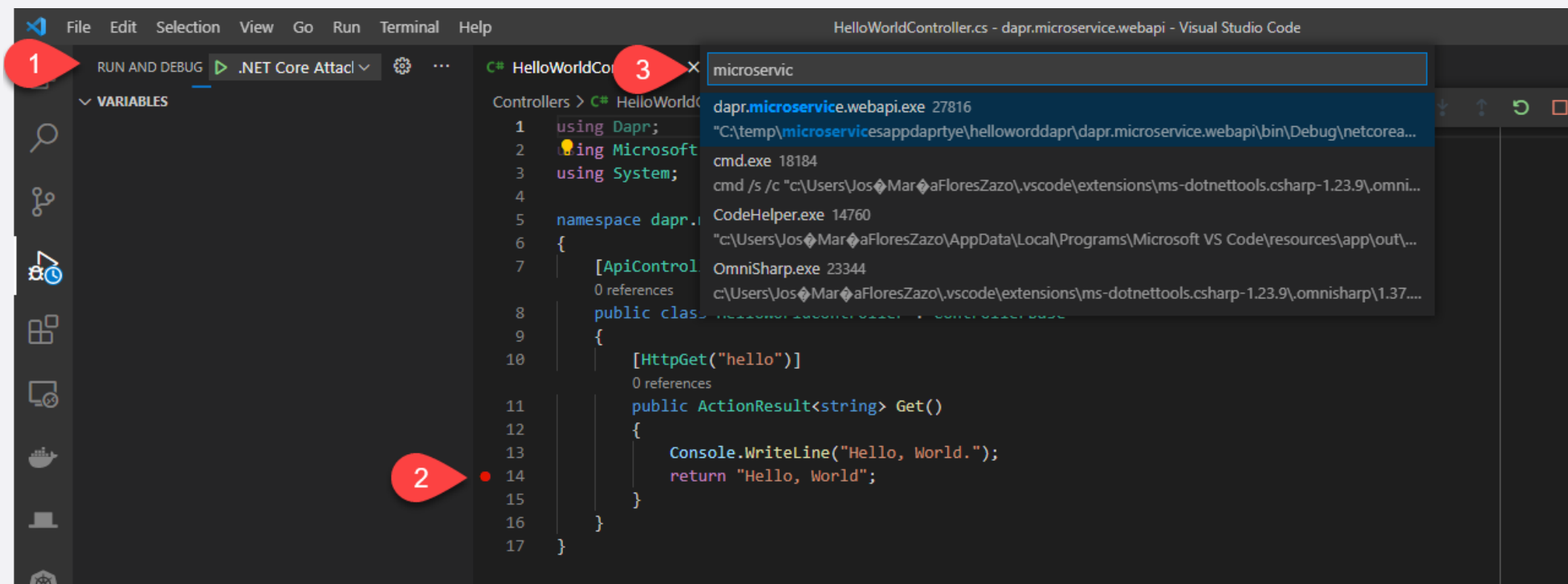
Depurar el ejemplo E2E

Hello World! con Dapr (8/9)

Vamos a enlazar el ejemplo contra el debugger de VS Code. Primero revisamos el fichero `launch.json`:

```
},  
{  
  "name": ".NET Core Attach",  
  "type": "coreclr",  
  "request": "attach",  
  "processId": "${command:pickProcess}"  
}  
]
```

Y podemos enlazarlo, con el proyecto en ejecución:



Depurar el ejemplo E2E

Hello World! con Dapr (9/9)

Podrás depurar la aplicación:

The screenshot displays two applications side-by-side. The top application is Visual Studio Code, showing the source code for `HelloWorldController.cs`. The code is as follows:

```
1 using Dapr;
2 using Microsoft.AspNetCore.Mvc;
3 using System;
4
5 namespace dapr.microservice.webapi.Controllers
6 {
7     [ApiController]
8     public class HelloWorldController : ControllerBase
9     {
10         [HttpGet("hello")]
11         public ActionResult<string> Get()
12         {
13             Console.WriteLine("Hello, World.");
14             return "Hello, World";
15         }
16     }
17 }
```

The bottom application is Postman, showing a GET request to `http://localhost:5010/v1.0/invoke/hello-world/method/hello`. The request is in the "Query Params" tab, and the response is "Sending request...".



Ejemplo E2E

Como depurar varios con Dapr (1/)

Lógicamente estamos hablando de microservicios y el ejemplo anterior no es para nada un microservicio. Para ello vamos con el ejemplo (marcado en el repo como ejemplo2).

La arquitectura de este ejemplo es muy sencilla, dos servicios al que invocamos desde un endpoint con Postman. Algo sencillo de interpretar.

Os dejo tarea: ver como he montado los archivos `tasks.json` y `launch.json` para ejecutar estos WebApi.

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "build-webapi",
      "command": "dotnet",
      "type": "process",
      "args": [
        "build",
        "${workspaceFolder}/hello.microservice.webapi/hello.microservice.webapi.csproj",
        "/property:GenerateFullPaths=true",
        "/consoleloggerparameters:NoSummary"
      ],
      "problemMatcher": "$msCompile"
    },
    {
      "label": "build-webapi2",
      "command": "dotnet",
      "type": "process",
      "args": [
        "build",
        "${workspaceFolder}/hi.microservice.webapi/hi.microservice.webapi.csproj",
        "/property:GenerateFullPaths=true",
        "/consoleloggerparameters:NoSummary"
      ],
      "problemMatcher": "$msCompile"
    },
    {
      "label": "publish",
      "command": "dotnet",
      "type": "process",
      "args": [
        "publish",
        "${workspaceFolder}/hello.microservice.webapi.csproj",
        "/property:GenerateFullPaths=true",
        "/consoleloggerparameters:NoSummary"
      ],
      "problemMatcher": "$msCompile"
    },
    {
      "label": "watch",
      "command": "dotnet",
      "type": "process",
      "args": [
        "watch",
        "run",
        "${workspaceFolder}/hello.microservice.webapi.csproj",
        "/property:GenerateFullPaths=true",
        "/consoleloggerparameters:NoSummary"
      ],
      "problemMatcher": "$msCompile"
    },
    {
      "appId": "hello",
      "appPort": 5001,
      "httpPort": 5010,
      "url": "http://localhost:5010"
    }
  ],
  "compounds": [
    {
      "name": "Launch Hello & Hi",
      "configurations": [
        "Launch hello",
        "Launch hi"
      ]
    }
  ]
}
```

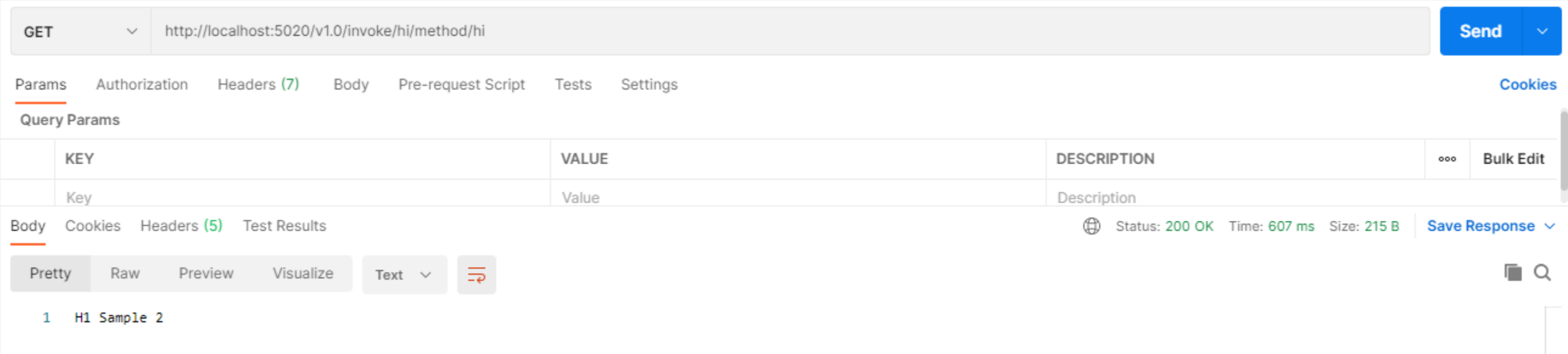
```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": ".NET Core Attach",
      "type": "coreclr",
      "request": "attach",
      "processId": "${command:pickProcess}"
    },
    {
      "name": "Launch hello",
      "type": "coreclr",
      "request": "launch",
      "preLaunchTask": "daprd-debug-webapi",
      "program": "${workspaceFolder}/hello.microservice.webapi/bin/Debug/netcoreapp3.1/hello.microservice.webapi.dll",
      "args": [],
      "cwd": "${workspaceFolder}/hello.microservice.webapi",
      "stopAtEntry": false,
      "env": {
        "ASPNETCORE_ENVIRONMENT": "Development",
        "ASPNETCORE_URLS": "http://*:5001"
      },
      "sourceFileMap": {
        "/Views": "${workspaceFolder}/Views"
      },
      "postDebugTask": "daprd-down-webapi"
    },
    {
      "name": "Launch hi",
      "type": "coreclr",
      "request": "launch",
      "preLaunchTask": "daprd-debug-webapi2",
      "program": "${workspaceFolder}/hi.microservice.webapi/bin/Debug/netcoreapp3.1/hi.microservice.webapi.dll",
      "args": [],
      "cwd": "${workspaceFolder}/hi.microservice.webapi",
      "stopAtEntry": false,
      "env": {
        "ASPNETCORE_ENVIRONMENT": "Development",
        "ASPNETCORE_URLS": "http://*:5002"
      },
      "sourceFileMap": {
        "/Views": "${workspaceFolder}/Views"
      },
      "postDebugTask": "daprd-down-webapi2"
    }
  ],
  "compounds": [
    {
      "name": "Launch Hello & Hi",
      "configurations": [
        "Launch hello",
        "Launch hi"
      ]
    }
  ]
}
```



Ejemplo E2E

Como depurar varios con Dapr (2/2)

Solamente debéis ejecutar la llamada desde Postman a cada endpoint, no entra en este workshop una comunicación entre ellos (ya lo vimos en el workshop anterior).



Ejemplo E2E

Como usar Tye con Dapr ^(1/3)

Partiendo del ejemplo anterior vamos a utilizar Tye.

Para poder trabajar un poco con servicios, solo con estos dos servicios, hemos optado por dos vias:

- O bien línea de comandos y enlazar manualmente los servicios para poder depurarlos. En total 2 ventanas de CLI para poder ver el log y nuestra solución de VS Code.
- O bien tener que crearnos en VS Code unos ficheros de configuración donde debemos prestar atención para que podamos lanzar los servicios y salga todo en el IDE. Esta configuración no es nada trivial.

VS Code nos sin lugar a dudas es capaz de ofrecernos un entorno aceptable con diversas vías para depurar microservicios.

Pero gracias a Tye podemos tener otra opción que como mínimo nos esta ahorrando algo con respecto a las anteriores formas de trabajar en depuración.



Ejemplo E2E

Como usar Tye con Dapr (2/3)

Veamos como funciona con nuestro ejemplo de Dapr.

El primer paso es ejecutar:

```
tye init
```

Nos generará este ymal:

```
tye.yaml > ...
1  # tye application configuration file
2  # read all about it at https://github.com/dotnet/tye
3  #
4  # when you've given us a try, we'd love to know what you think:
5  #   https://aka.ms/AA7q20u
6  #
7  name: sample2
8  services:
9  - name: hello-microservice-webapi
10 |   project: hello.microservice.webapi/hello.microservice.webapi.csproj
11 - name: hi-microservice-webapi
12 |   project: hi.microservice.webapi/hi.microservice.webapi.csproj
13
```

Al que debemos añadir la extensión de dapr (ver el ejemplo de GitHub).

```
name: sample2
extensions:
- name: dapr
services:
```



Ejemplo E2E

Como usar Tye con Dapr (3/3)

Y ejecutamos:

```
tye run
```

Os muestro la información que proporciona el Dashboard y solo nos quedaría hacer un test del endpoint con Postman:

Tye Dashboard

[Tell us what you think!](#)[About](#)

Home

Services

Name	Type	Source	Bindings	Replicas	Restarts	Logs
hello-microservice-webapi	Project	C:\temp\microservicesapp\daprtye\sample2\hello.microservice.webapi\hello.microservice.webapi.csproj	http://localhost:57494 https://localhost:57495	1/1	0	View
hi-microservice-webapi	Project	C:\temp\microservicesapp\daprtye\sample2\hi.microservice.webapi\hi.microservice.webapi.csproj	http://localhost:57496 https://localhost:57497	1/1	0	View
placement	Container	daprio/dapr	http://localhost:57493	1/1	0	View
hello-microservice-webapi-dapr	Executable		https://localhost:57498 http://localhost:57499 http://localhost:57500	1/1	0	View
hi-microservice-webapi-dapr	Executable		https://localhost:57501 http://localhost:57502 http://localhost:57503	1/1	0	View



Moviéndonos a la nube

Desplegar Dapr en AKS (1/7)

Es fundamental conocer su funcionamiento de **Azure CLI**.

El portal es una herramienta muy potente, pero debido al cambio continuo que sufre: cambios de nombre, situación de la opción que necesitamos, etc. Rápidamente se queda obsoleto, cosa que no ocurre con Azure CLI.

El procedimiento que os explicaré a continuación no es más que una adaptación de:

<https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough>

A nuestro ejemplo.

Vamos a lanzar una consola para poder trabajar con Azure CLI, si no lo tienes instalado, ese será el primer paso a seguir antes de adentrarse en el proceso:

<https://docs.microsoft.com/es-es/cli/azure/install-azure-cli>

Y hacer login:

```
az login
```



Moviéndonos a la nube

Desplegar Dapr en AKS (2/7)

Tras poner nuestras credenciales en la ventana emergente de tu navegador web que tengas por defecto.

Debemos situarnos en la suscripción que desees trabajar. Lista las suscripciones:

```
az account list
```

Busca, cual es la que vas a utilizar y establece el contexto:

```
az account set --subscription "YOUR_SUSSCRIPTION_NAME"
```

Creamos un resource group para nuestro trabajo:

```
az group create -l westus -n darpk8srgwork3
```

Ya solo nos queda **crear el cluster de AKS**:

```
az aks create --resource-group darpk8srgwork3 --name darpk8sakswork3 --node-count 3  
--node-vm-size Standard_D2s_v3 --enable-addons monitoring  
--vm-set-type VirtualMachineScaleSets --generate-ssh-keys
```



Moviéndonos a la nube

Desplegar Dapr en AKS (3/7)

Tras la espera necesaria para crear los recursos, tardará un poco, podemos verificar que el estado de los mismo:

```
az aks show --name daprk8sakswork3 --resource-group daprk8srgwork3
```

Instalar **Kubectl** CLI:

```
az aks install-cli
```

Nos queda añadir las credenciales para poder tener acceso al cluster:

```
az aks get-credentials --name daprk8sakswork3 --resource-group daprk8srgwork3
```

command session.

```
2. Update system PATH environment variable by following "Control Panel->System->Advanced->Environment Variables"
PS C:\temp\microservicesapp\daprtty> az aks get-credentials --name daprk8sakswork3 --resource-group daprk8srgwork3
Merged "daprk8sakswork3" as current context in C:\Users\JoséMaríaFloresZazo\.kube\config
```



Moviéndonos a la nube

Desplegar Dapr en AKS (4/7)

Y ya solo nos queda comprobar que ya tienes acceso a Kubectl CLI, herramienta que nos permitirá controlar cualquier aspecto del K8s:

Kubectl get nodes

```
PS C:\temp\microservicesappdaprtty> Kubectl get nodes
NAME                                STATUS    ROLES    AGE   VERSION
aks-nodepool1-12606866-vmss000000  Ready    agent    4m58s v1.18.14
aks-nodepool1-12606866-vmss000001  Ready    agent    5m1s  v1.18.14
aks-nodepool1-12606866-vmss000002  Ready    agent    4m46s v1.18.14
```

Todo el anterior trabajo nos a servido para generar nuestro clúster de K8s, así como las herramientas locales que nos dan acceso al mismo.

Con estos pasos previos ya podemos **instalar Dapr en nuestro clúster de AKS**.

Para ello vamos a usar nuestro ejemplo uno y lo vamos a copiar en una nueva carpeta: ejemplo tres. La intención es añadir la menor complejidad posible a la explicación.



Moviéndonos a la nube

Desplegar Dapr en AKS (5/7)

Para ello vamos a usar nuestro ejemplo dos y lo vamos a copiar en una nueva carpeta: ejemplo tres. Y ejecutamos el siguiente comando:

```
dapr init -k
```

```
PS C:\temp\microservicesapp\daprtypes\sample3> dapr init -k
Making the jump to hyperspace...
Note: To install Dapr using Helm, see here: https://docs.dapr.io/getting-started/install-dapr-kubernetes/#install-with-helm-advanced

Deploying the Dapr control plane to your cluster...
Success! Dapr has been installed to namespace dapr-system. To verify, run `dapr status -k` in your terminal. To get started, go here: https://aka.ms/dapr-getting-started
```

Con esta acción hemos instalado e inicializamos Dapr en el clúster correspondiente:

```
dapr status -k
```

```
PS C:\temp\microservicesapp\daprtypes\sample3> dapr status -k
```

NAME	NAMESPACE	HEALTHY	STATUS	REPLICAS	VERSION	AGE	CREATED
dapr-operator	dapr-system	True	Running	1	1.1.2	3m	2021-04-17 12:55.34
dapr-dashboard	dapr-system	True	Running	1	0.6.0	3m	2021-04-17 12:55.34
dapr-sidecar-injector	dapr-system	True	Running	1	1.1.2	3m	2021-04-17 12:55.34
dapr-sentry	dapr-system	True	Running	1	1.1.2	3m	2021-04-17 12:55.34
dapr-placement-server	dapr-system	True	Running	1	1.1.2	3m	2021-04-17 12:55.35



Moviéndonos a la nube

Desplegar Dapr en AKS ^(6/7)

Podemos ver los servicios:

Kubectl get services -n dapr-system -w

```
PS C:\temp\microservicesappdaprt\sample3> dapr status -k
NAME          NAMESPACE   HEALTHY  STATUS  REPLICAS  VERSION  AGE   CREATED
dapr-operator  dapr-system  True     Running 1          1.1.2    3m    2021-04-17 12:55:34
dapr-dashboard dapr-system  True     Running 1          0.6.0    3m    2021-04-17 12:55:34
dapr-sidecar-injector dapr-system True     Running 1          1.1.2    3m    2021-04-17 12:55:34
dapr-sentry    dapr-system  True     Running 1          1.1.2    3m    2021-04-17 12:55:34
dapr-placement-server dapr-system True     Running 1          1.1.2    3m    2021-04-17 12:55:35
```

O desde el portal:

Home > daprk8sakswork3

daprk8sakswork3

Kubernetes service

Search (Ctrl+ /)

«

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Security

Kubernetes resources

Namespaces

Workloads

Services and ingresses

Storage

Configuration

Settings

Node pools

Cluster configuration

+ Add

Delete

Refresh

Show labels

Services

Ingresses

Filter by service name

Filter by namespace

Enter the full service name

All namespaces


<input type="checkbox"/>	Name	Namespace	Status	Type	Cluster IP	External IP	Ports	Age ↓
<input type="checkbox"/>	kubernetes	default	✓ Ok	ClusterIP	10.0.0.1		443/TCP	21 minutes
<input type="checkbox"/>	healthmodel-replicaset-service	kube-system	✓ Ok	ClusterIP	10.0.114.204		25227/TCP	20 minutes
<input type="checkbox"/>	kube-dns	kube-system	✓ Ok	ClusterIP	10.0.0.10		53/UDP,53/TCP	20 minutes
<input type="checkbox"/>	metrics-server	kube-system	✓ Ok	ClusterIP	10.0.181.150		443/TCP	20 minutes
<input type="checkbox"/>	dapr-api	dapr-system	✓ Ok	ClusterIP	10.0.55.117		80/TCP	8 minutes
<input type="checkbox"/>	dapr-dashboard	dapr-system	✓ Ok	ClusterIP	10.0.123.7		8080/TCP	8 minutes
<input type="checkbox"/>	dapr-placement-server	dapr-system	✓ Ok	ClusterIP	None		50005/TCP,820...	8 minutes
<input type="checkbox"/>	dapr-sentry	dapr-system	✓ Ok	ClusterIP	10.0.99.181		80/TCP	8 minutes
<input type="checkbox"/>	dapr-sidecar-injector	dapr-system	✓ Ok	ClusterIP	10.0.112.215		443/TCP	8 minutes

Moviéndonos a la nube

Desplegar Dapr en AKS (7/7)

Ver el Dashboard de Dapr:

```
dapr Dashboard -k
```

Dashboard

>

Application: hello-world

SummaryActors

Summary

App IDhello-world

App Port5000

Dapr HTTP Port5010

Dapr gRPC Port57505

Commanddotnet run


Replicas1

Addresslocalhost:5010

PID5392



Created

Age53s

Dashboard

ScopeAll

Dapr Components

	Name	Type	Age	Created
	pubsub	pubsub.redis	3d	<div></div>
	statestore	state.redis	3d	<div></div>



Crear imágenes de Docker y exponer la solución

Un paso más con AKS ^(1/2)

Aunque se escapa en parte de la intención de este workshop os voy a dejar un mini-howto para que podáis continuar con el despliegue mediante Docker.

1. Crear el fichero `Dockerfile` correspondiente para generar la imagen
2. Construir la imagen con `docker build`
3. Crear un repositorio de imágenes por ejemplo a Azure Container Registry
4. Subir con `docker push`
5. Generar los secretos con Azure Key Vault e introducirlos en nuestro `.yaml` de Dapr.
6. Desplegar la aplicación con

```
az aks update --resource-group daprk8srgwork3 --name daprk8sakswork3 --attach-acr  
daprk8sacr
```

7. Y si recuerdas de anteriores workshops, solo te queda lanzar:

```
dapr run --app-id "XXX" --app-port "5004" --dapr-grpc-port "50040" -- ...
```



Crear imágenes de Docker y exponer la solución

Un paso más con AKS (2/2)

Lo que nos quedaría sería exponer nuestra aplicación a clientes externos.

La opción más conocida es establecer una NGINX como controlador de Ingreso:

<https://docs.microsoft.com/es-es/azure/aks/ingress-basic>

Y otra no tanto, es usar la potencia que ofrece Azure API Management, de aquí que hablara de el en anteriores workshops y comentara de sistema auto-hospedado de APIM:

<https://docs.microsoft.com/en-us/azure/api-management/api-management-dapr-policies>

Si deseas conocer más sobre este tema del que os he contado unas pinceladas, no queda más remedio que investigues por tu parte.

Y hasta aquí AKS.



Observar

Herramientas y procedimientos ^(1/2)

Vamos a ver como podemos observar una aplicación Dapr. En esta ocasión iré también rápido, mi intención es que tengas el contexto y el conocimiento básico para que puedas aplicarlo. No se trata de un step-by-step, con los workshops anteriores y un poco de hand-on-labs por tu parte, podrás aplicar todo lo que explico en esta sección.

Para ello vamos puedes usar dos acciones:

- Distributed tracing.
- OpenTelemetry collector.

Son las herramientas que han crecido y evolucionado casi al mismo tiempo que K8s. Si bien es cierto que Microsoft esta haciendo un gran esfuerzo para que Application Insight este acompañando a AKS, estas herramientas lo superan con creces y si crees en el termino “agnóstico de la nube”, no vas a usar Application Insight, evidentemente.

Y también mencionaré una herramienta muy útil para test de cargas.



Observar

Herramientas y procedimientos ^(2/2)

¿Cómo se reparten el trabajo?

1. Tracing: con Azure Insight, Zipkin, New Relic, Jaegger y OpenTelemetry..
2. Metricas: usando Prometheus + Grafana y Azure Monitor.
3. Logs: con Fluentd, Elastic search y Kibana.
4. Salud: con Health API de Dapr y usando el endpoint healthz.
5. Test de cargas: con Locust (directamente relacionado con el autoescalado con KEDA, por ejemplo).

Si sigues el movimiento **Open Telemetry** (<https://opentelemetry.io/> de la CNCF), muchas de estas herramientas te sonaran.

Os dejo un articulo de Microsoft sobre el tema, ya que, aunque tengas sus sistemas siempre están a favor de OS:

<https://devblogs.microsoft.com/dotnet/opentelemetry-net-reaches-v1-0/>



Observar una aplicación Dapr

Monitorizar aplicaciones: seguimiento, métricas, logs y salud (1/11)

¿Qué es Zipkin?

Es un sistema de rastreo distribuido Open Source. Nos ofrece la capacidad de buscar por ID, por servicios, operaciones o etiquetas, además nos muestra la dependencia entre servicios.

Puedes ampliar la información en: <https://zipkin.io>

Pasos:

1. Configurar Zipkin.
2. Configurar el exportador de Dapr.
3. Habilitar el seguimiento con Dapr.
4. Investigar con Zipkin.

Vamos a instalar Zipkin en el cluster de AKS.

Para ello puedes ir al archivo: `\microservicesappdaprttye\sample3\step1_zipkin.yaml`

Como nota importante es el puerto que usamos: 9411.



Observar una aplicación Dapr

Monitorizar aplicaciones: seguimiento, métricas, logs y salud (2/11)

Ahora lanzamos:

```
kubectl apply -f .\microservicesappdaprtypesample3\step1_zipkin.yaml
```

Para acceder a Zipkin o bien ponemos un NGINX de ingress o bien un forward:

```
kubectl port-forward svc/zipkin 9412:9411
```

Ya podemos acceder al portal de Zipkin:

<https://localhost:9412/>

Llega el momento de integrar el componente de Zipkin para Dapr:

```
\microservicesappdaprtypesample3\step2_component_zipkin.yaml
```

Y aplicamos:

```
kubectl apply -f .\microservicesappdaprtypesample3\step2_component_zipkin.yaml
```



Observar una aplicación Dapr

Monitorizar aplicaciones: seguimiento, métricas, logs y salud (3/11)

Establecemos el tracing de Zipkin:

```
\microservicesappdaprt\sample3\step3_configuration_zipkin.yaml
```

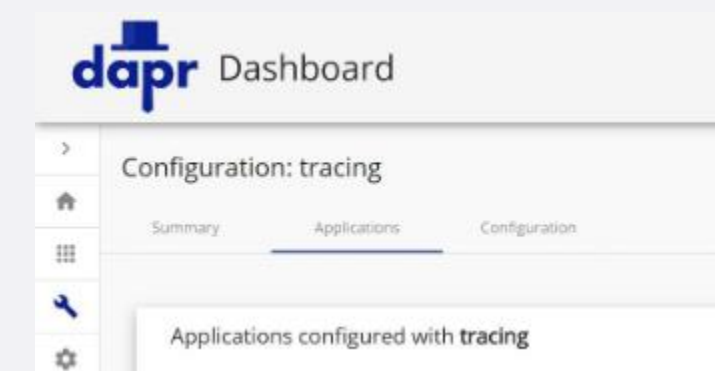
Aplicamos:

```
kubectl apply -f .\microservicesappdaprt\sample3\step3_configuration_zipkin.yaml
```

Modificamos los yaml de los servicios para añadir:

```
template:
  metadata:
    labels:
      app: order-service
    annotations:
      dapr.io/enabled: "true"
      dapr.io/id:
      dapr.io/port: "80"
      dapr.io/config: "tracing"
      dapr.io/sidecar-liveness-probe-period-seconds: "20"
      dapr.io/sidecar-readiness-probe-period-seconds: "20"
  spec:
```

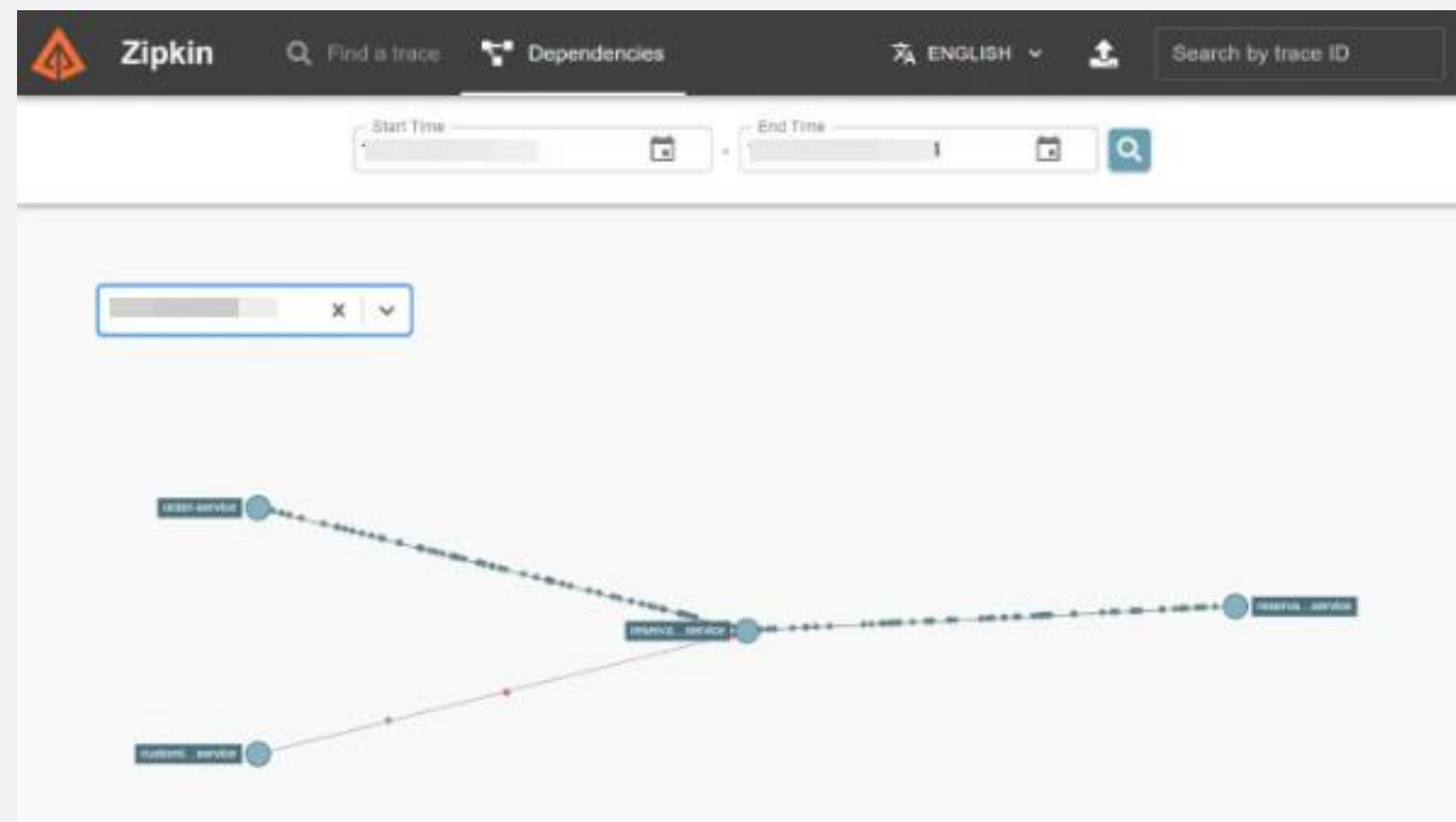
Y ya podemos ver que estamos traceando los servicios en el dashboard de Dapr:



Observar una aplicación Dapr

Monitorizar aplicaciones: seguimiento, métricas, logs y salud (4/11)

Un proceso importante es investigar un incidente, para ello Zipkin nos ofrece un Dashboard donde podemos ver el mapa de componentes y dependencias:



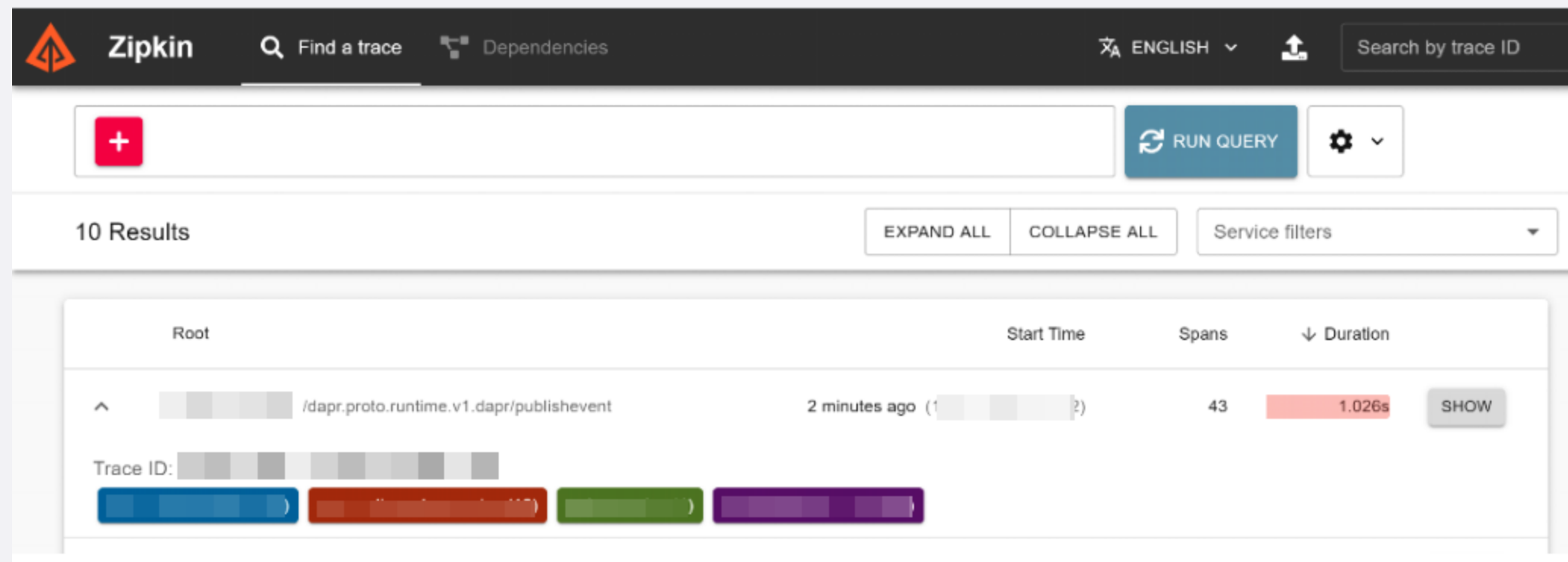
Observar una aplicación Dapr

Monitorizar aplicaciones: seguimiento, métricas, logs y salud (5/11)

En Zipkin existen dos conceptos que debemos manejar:

- Span, que es la unidad de trabajo (uow) ejecutada en un componente o servicio.
- Trace, una colección de Span.

Cuando entramos en un servicio, haciendo clic en una de las cajas anteriores:



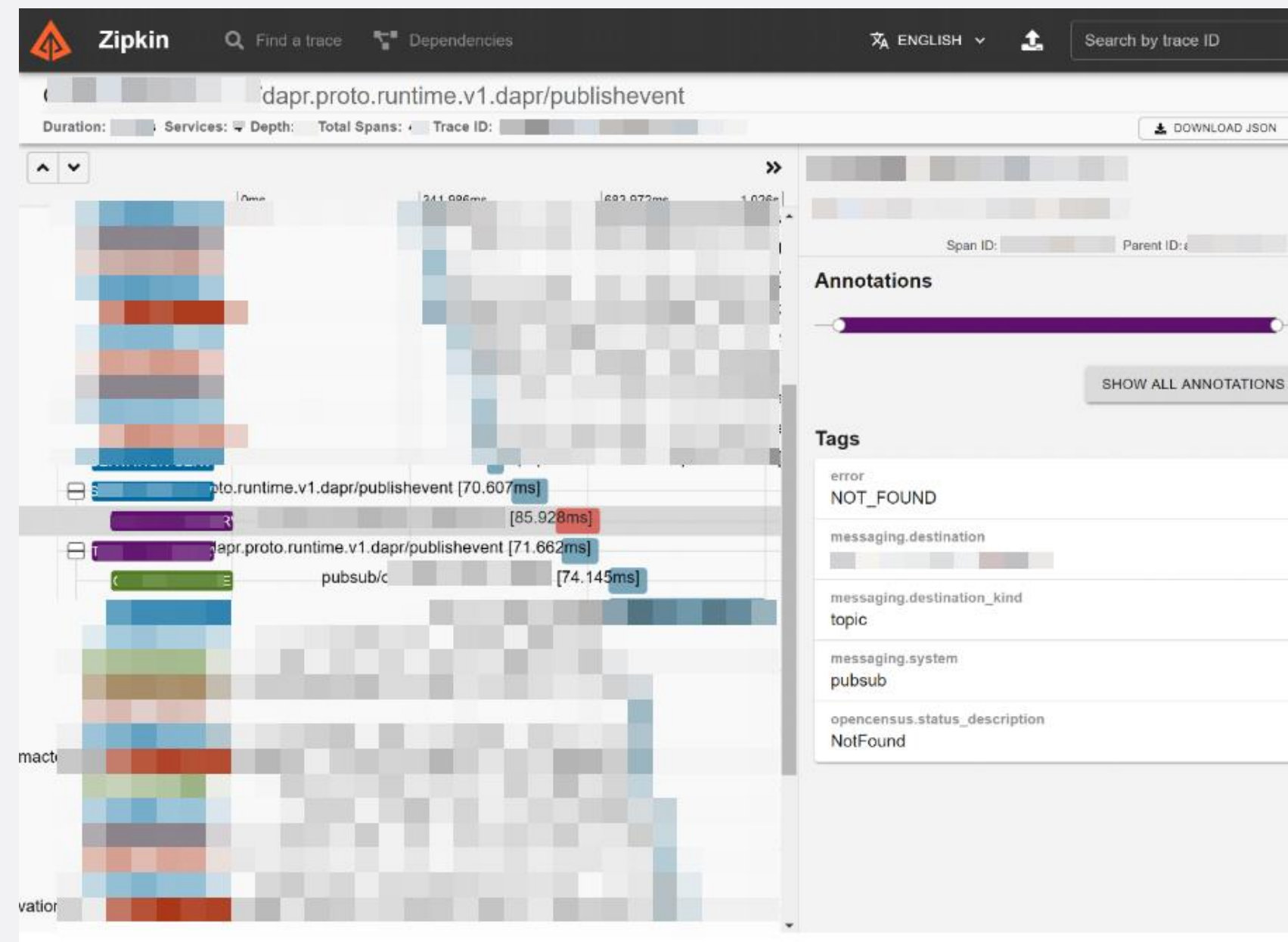
¡Nota! Disculpar la ofuscación de las imágenes, ya que son capturas de un proyecto real



Observar una aplicación Dapr

Monitorizar aplicaciones: seguimiento, métricas, logs y salud (6/11)

Y pulsando en show, veremos las trazas:



Observar una aplicación Dapr

Monitorizar aplicaciones: seguimiento, métricas, logs y salud (6/11)

Para poder analizar las métricas, recordar, vamos a usar Prometheus y Grafana.

Prometheus es Open Source y son un conjunto de herramientas para monitorizar, un proyecto que se inició en 2012 que ahora forma parte de la CNCF.

En nuestro caso vamos a sacar métricas expuestas por los Pods de Dapr y almacenarlas como series de tiempo. Estos serán los datos para los paneles de Grafana (nuestro visualizador).

Grafana es una herramienta de análisis y visualización de datos Open Source, la puedes usar para más cosas, como por ejemplo contra IoT.

En Grafana, vamos a usar unas plantillas generadas en el proyecto Dapr: <https://github.com/dapr/dapr/Releases>.

Y los pasos son:

1. Instalar Prometheus.
2. Instalar Grafana.
3. Importar paneles.



Observar una aplicación Dapr

Monitorizar aplicaciones: seguimiento, métricas, logs y salud (7/11)

Para instalar Prometheus, nada mejor como leer esta documentación:

<https://docs.dapr.io/operations/monitoring/prometheus>

Pero con esto nos sobrará:

```
kubectl create namespace dapr-monitoring
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
helm install dapr-prom prometheus-community/prometheus -n dapr-monitoring
```

Observamos que todo fue bien con:

```
kubectl get pods -n dapr-monitoring -w
```

Y comprobar que los pods de Prometheus estan corriendo:

```
kubectl get svc -n dapr-monitoring
```



Observar una aplicación Dapr

Monitorizar aplicaciones: seguimiento, métricas, logs y salud (8/11)

Para instalar Grafana, nada mejor como leer esta documentación:

<https://docs.dapr.io/operations/monitoring/grafana/>

Pero con esto nos sobrar :

```
helm repo add grafana https://grafana.github.io/helm-charts
helm repo update
helm install grafana grafana/grafana -n dapr-monitoring
```

Obener la password de Grafana:

```
$base64secret = kubectl get secret --namespace dapr-monitoring grafana -o jsonpath="{.data.admin-password}"
$password = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($base64secret))
$password
[this_is_your_password]
```

Importamos los dashboards:

```
kubectl port-forward svc/grafana 8090:80 -n dapr-monitoring
```



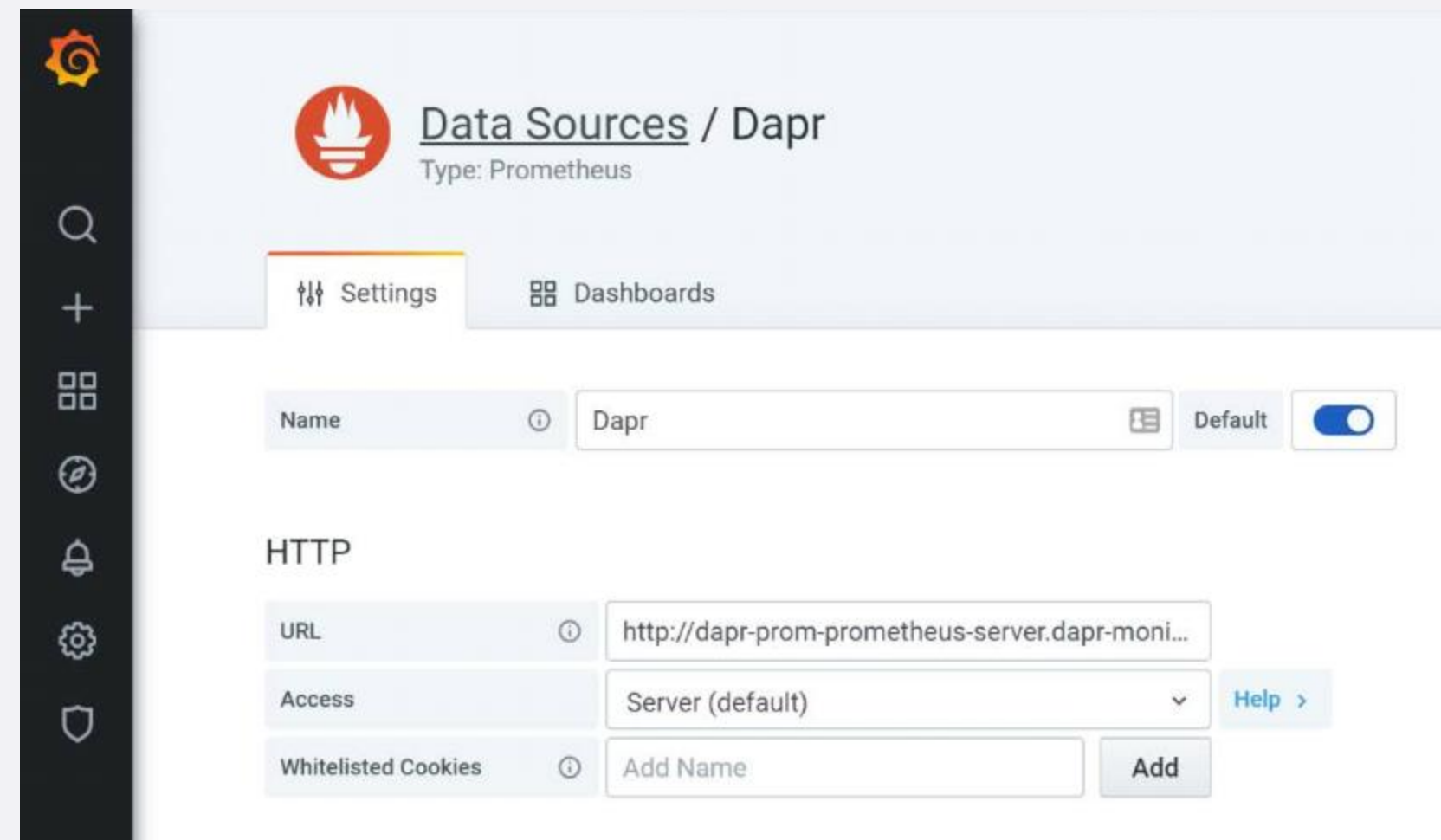
Observar una aplicación Dapr

Monitorizar aplicaciones: seguimiento, métricas, logs y salud (9/11)

Entramos en:

<https://localhost:8090>

Y:



The screenshot shows the 'Data Sources / Dapr' configuration page in a monitoring tool. The page has a dark sidebar on the left with icons for search, add, view, notifications, settings, and security. The main content area has a header with the Dapr logo and the title 'Data Sources / Dapr' with 'Type: Prometheus' below it. There are two tabs: 'Settings' (active) and 'Dashboards'. Under the 'Settings' tab, there is a 'Name' field set to 'Dapr' with a 'Default' toggle switch turned on. Below this is an 'HTTP' section with a 'URL' field containing 'http://dapr-prom-prometheus-server.dapr-moni...', an 'Access' dropdown menu set to 'Server (default)', and a 'Whitelisted Cookies' section with an 'Add Name' input field and an 'Add' button. A 'Help >' link is also present next to the 'Access' dropdown.



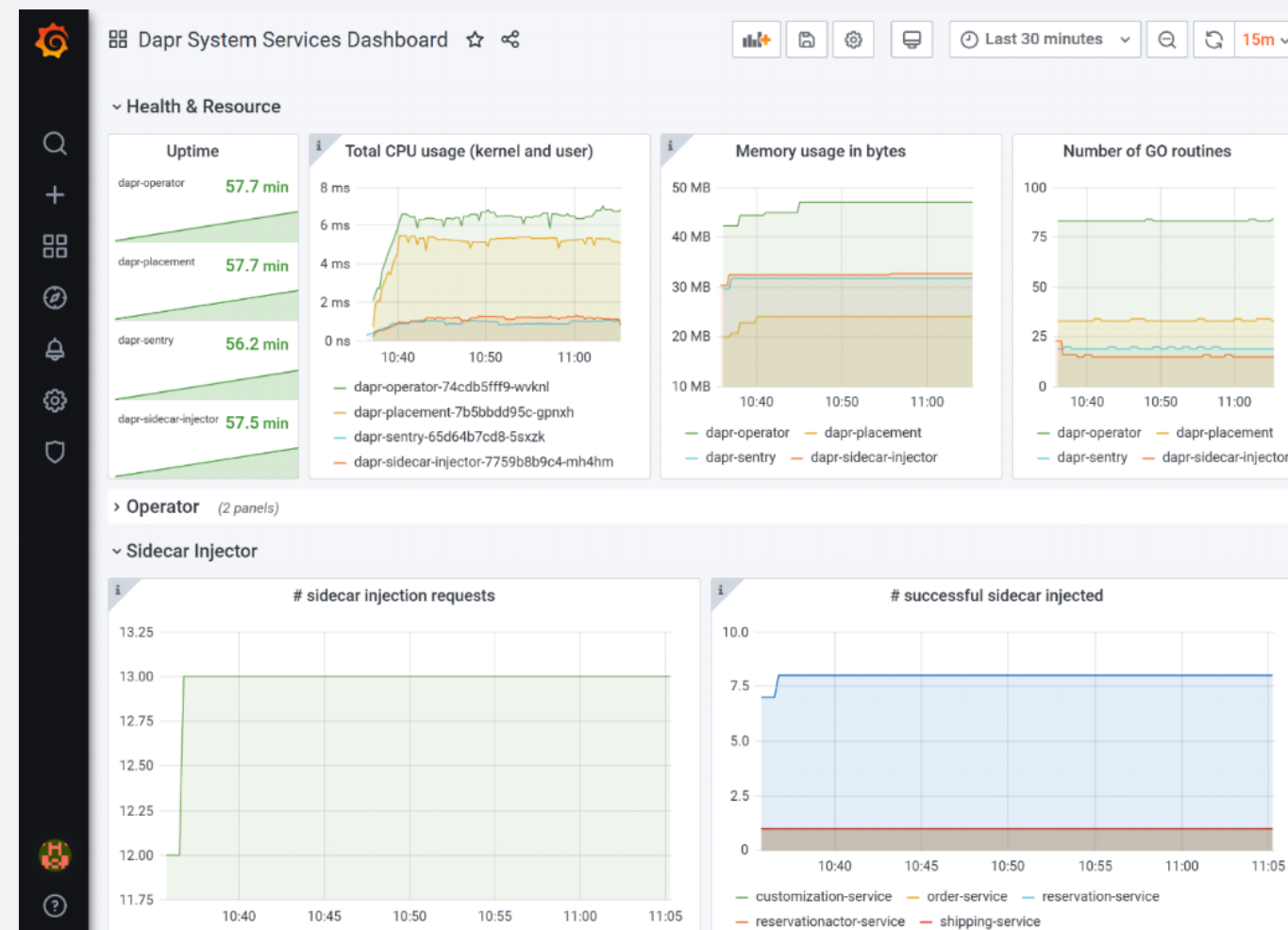
Observar una aplicación Dapr

Monitorizar aplicaciones: seguimiento, métricas, logs y salud (10/11)

Añadimos la url: `http://dapr-prom-prometheus-server.dapr-monitoring`

E importamos los 3 dashboards de: <https://github.com/dapr/dapr/tree/master/grafana>

Para que podamos ver:

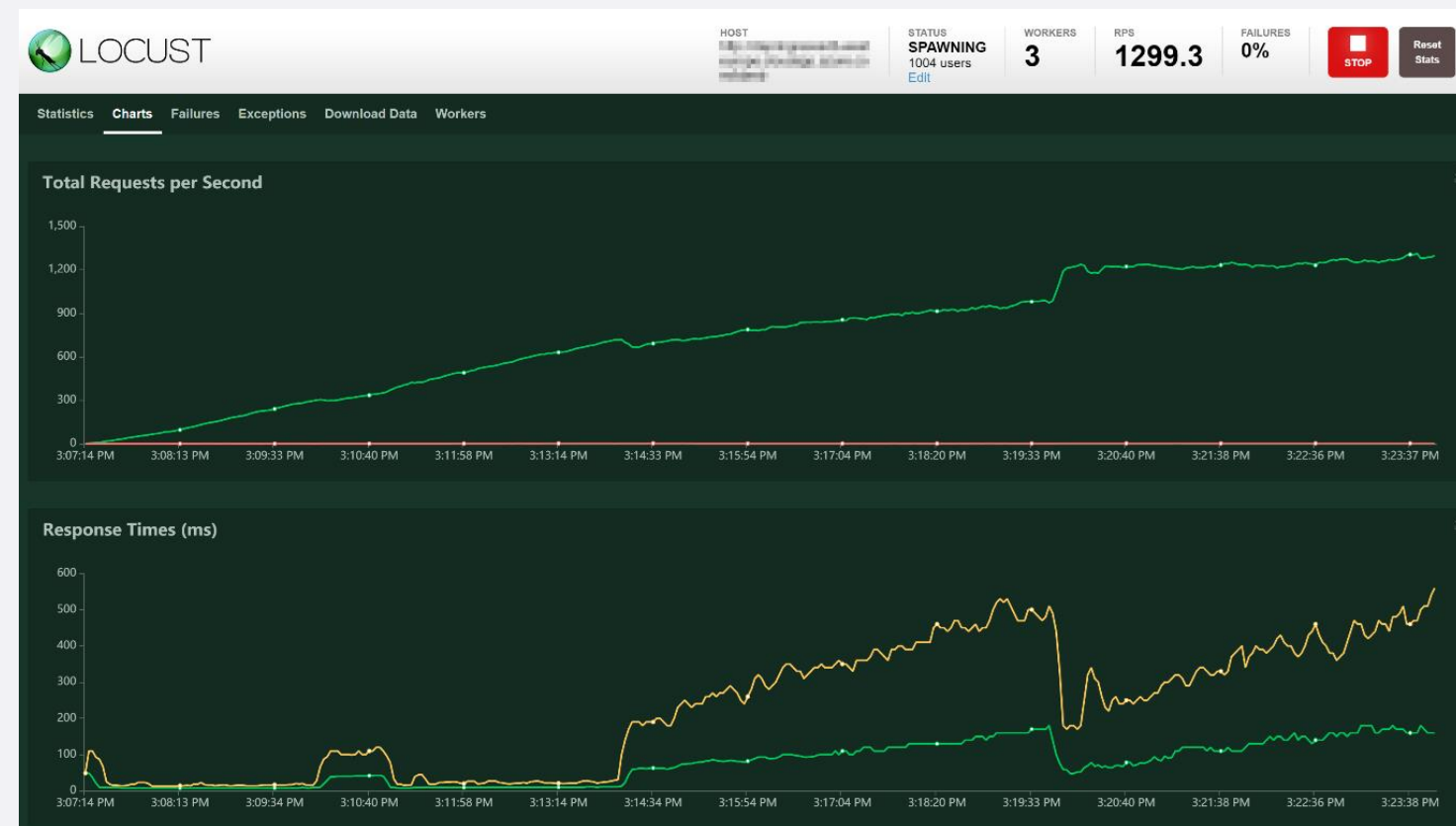


Observar una aplicación Dapr

Monitorizar aplicaciones: seguimiento, métricas, logs y salud (11/11)

Como bonus y que deberás investigar por tu parte, para hacer test de carga os recomiendo que reviséis:

- Locust, para los test: <https://docs.locust.io>



- Y el auto escalado con KEDA en el contexto de Dapr:
<https://docs.dapr.io/developing-applications/integrations/autoscale-keda/>



Workshop Resumen

¡Enhorabuena y muchas gracias por llegar al final!

Espero que este ciclo de 3 workshops de microservicios te pueda aportar el conocimiento suficiente para afrontarlos de una forma más moderna y con herramientas que te ayuden en tu día día.

Mi siguiente workshop será:



Spark y Azure Databrick para desarrolladores de .NET



¡Gracias!

Puedes encontrarme buscando por **jmfloreszazo** en

