

2021

# Netcoreconf

Introducción al Proyecto Tye

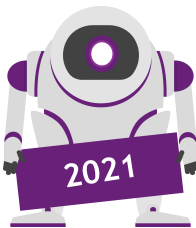


**Jose María Flores Zazo**

*Development & Cloud Consultan in Tokiota*

@jmfloreszazo

# SPONSORS



# Acerca de...



## Jose María Flores Zazo

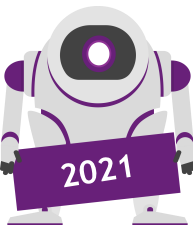
Tengo más de 25 años de experiencia en el campo de desarrollo, análisis, diseño y entrega de aplicaciones con diversas tecnologías.

Actualmente desempeño mi actividad laboral en TOKIOTA.

## Cosas que me motivan

A partir de mi profesión, hacer la vida más fácil a las personas.

Disfrutar de la familia, leer y hacer deporte.



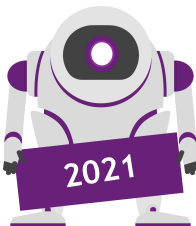
# ¿Qué es el Proyecto Tye?

El Proyecto Tye (**Project Tye**) es una herramienta experimental que facilita el desarrollo, prueba e implementación de microservicios y aplicaciones distribuidas.

Independientemente del nombre que reciba tu arquitectura, indudablemente se tratará de muchos servicios creados a partir de muchos proyectos y es complicado hacer que todo funcione.

La complejidad que supone configurarlo es algo que todos hemos sufrido, no es tan sencillo como parece: debemos configurar servicios, mapear puertos, etc.

La simple idea de escribir, ejecutar y probar el código, objetivo final de cualquier desarrollador se convierte en algo tedioso, complejo y frustrante. Aquí es donde entra **Project Tye**, eliminar esos escollos que hacen que nuestra productividad se vea mermada.



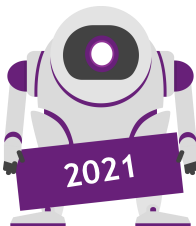
# ¿Dónde se aloja este proyecto?



<https://github.com/dotnet/tye>

<https://github.com/dotnet/tye/tree/master/docs>

<https://github.com/dotnet/tye/blob/master/docs/reference/commandline/README.md>



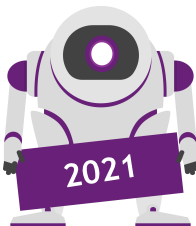
# ¿Por qué os presento el Proyecto Tye?

Siempre he sido un fiel detractor de la filosofía *early adopter* para proyectos en producción, nos soy un temerario, pero con esta herramienta he realizado una excepción debido a que la injerencia en un proyecto es mínima, ya lo veremos más adelante.

En mi caso Tye a venido a resolver una serie de situaciones reales que describo a continuación, por ejemplo:

- Probar un microservicio con la técnica de subir a un entorno DEV, utilizado por muchos desarrolladores, donde lo habitual es que choquen tus errores con los míos y nos impiden llegar al punto que deseamos probar. Puedes usar tecnicas como [Feature Toggles](#), desplegar en otro slot y apuntar a el, ... pero para probar un simple *fix* estamos gastando recursos y añadiendo complejidad. Y esto provoca frustración en el equipo de desarrollo.
- Cuando ves que no puedes disponer de un entorno aislado para hacer unas pruebas de un bug debido a que no eres capaz de montar todo lo necesario para emular el entorno hostil al 100%.
- O que simplemente no soportas tener que lanzar 10 soluciones (no proyectos) por muy bien que los puertos estén bien mapeados (cosa poco habitual), además debes añadir una cola RabbitMQ y cambiar la cadena de conexión de *SQL Server*.

Supongo que muchos podrán añadir unas cuantas más a esta lista.

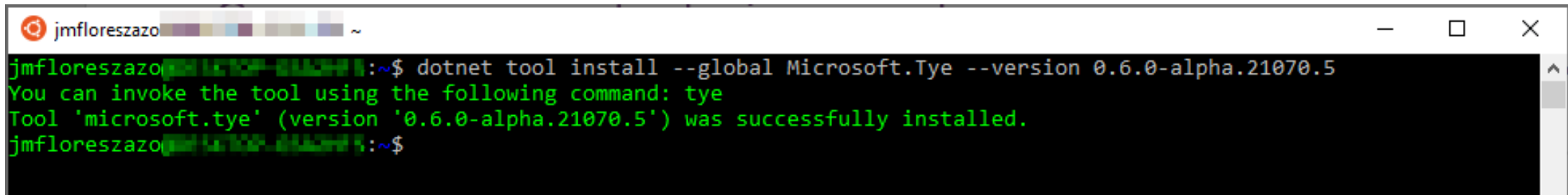


# Trabajando con Tye (1/11)

Para poder usar **Tye** debemos tener el SDK de .NET CORE instalado en nuestro equipo y es tan sencillo como ejecutar el siguiente comando, para disponer de ella de forma global:

<https://www.nuget.org/packages/Microsoft.Tye/>

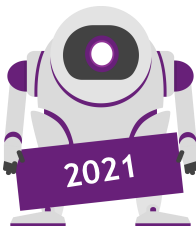
```
dotnet tool install -g Microsoft.Tye --version 0.6.0-alpha.21070.5
```

A screenshot of a terminal window with a dark background. The window title bar shows the user 'jmfloreszazo' and a home icon. The terminal text shows the command 'dotnet tool install --global Microsoft.Tye --version 0.6.0-alpha.21070.5' being executed. The output indicates that the tool 'microsoft.tye' (version '0.6.0-alpha.21070.5') was successfully installed and provides the command 'tye' to invoke it. The prompt returns to the user's shell.

```
jmfloreszazo@jmfloreszazo:~$ dotnet tool install --global Microsoft.Tye --version 0.6.0-alpha.21070.5
You can invoke the tool using the following command: tye
Tool 'microsoft.tye' (version '0.6.0-alpha.21070.5') was successfully installed.
jmfloreszazo@jmfloreszazo:~$
```

Necesitaremos tener Dockers y Docker Desktop for Kubernetes:

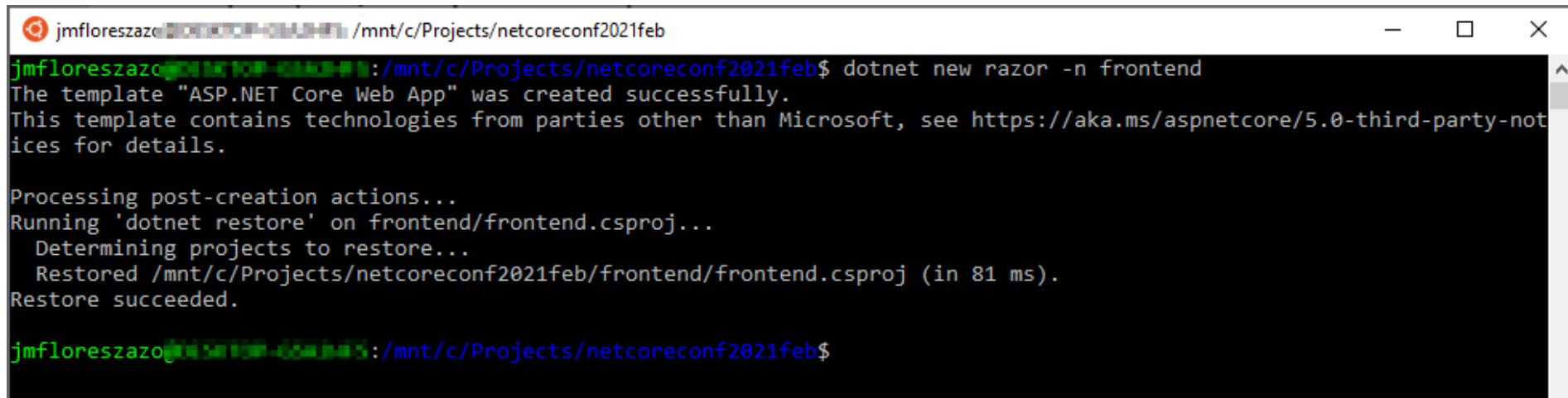
```
docker run -d -p 5000:5000 --restart=always --name registry registry:2
```



# Trabajando con Tye (2/11)

Desde una consola vamos a crear una aplicación .NET CORE, sin más complicación, ya que la idea es aprender que Tye.

```
dotnet new razor -n frontend
```

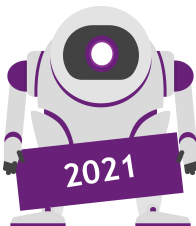


```
jmfloreszazo@DESKTOP-000000: /mnt/c/Projects/netcoreconf2021feb$ dotnet new razor -n frontend
The template "ASP.NET Core Web App" was created successfully.
This template contains technologies from parties other than Microsoft, see https://aka.ms/aspnetcore/5.0-third-party-notices for details.

Processing post-creation actions...
Running 'dotnet restore' on frontend/frontend.csproj...
  Determining projects to restore...
  Restored /mnt/c/Projects/netcoreconf2021feb/frontend/frontend.csproj (in 81 ms).
Restore succeeded.

jmfloreszazo@DESKTOP-000000: /mnt/c/Projects/netcoreconf2021feb$
```

Una de las razones por las que buscas una herramienta como Tye es poder ejecutar múltiples servicios con un solo comando, aunque hasta el momento resulte trivial.

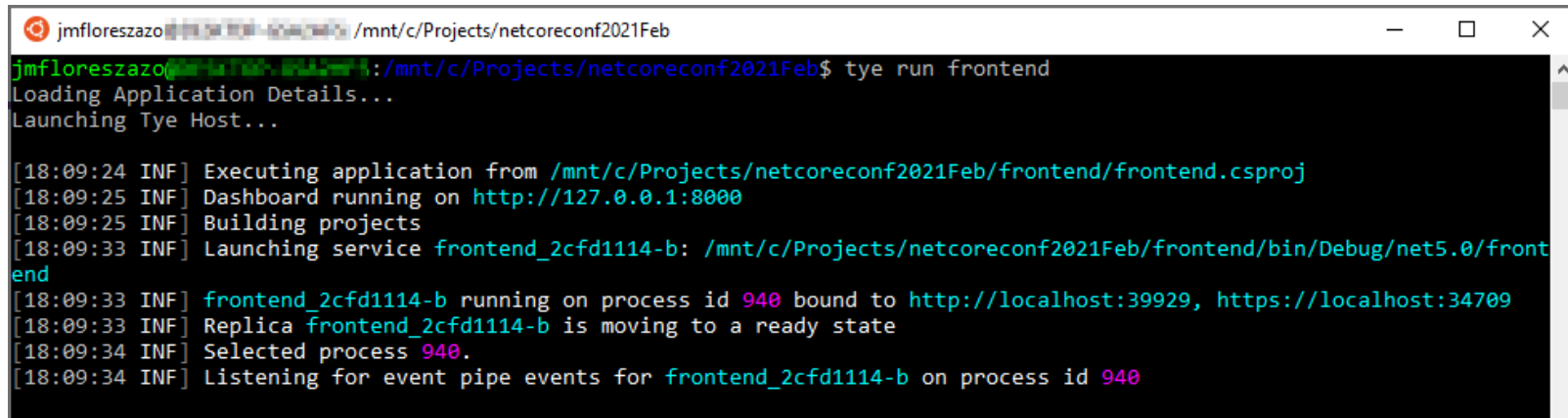




# Trabajando con Tye (3/11)

Tenemos un proyecto que podemos ejecutar con Visual Studio Code o Visual Studio 2019. Usaremos el tutorial básico que podéis encontrar en el proyecto oficial. Vamos a empezar:

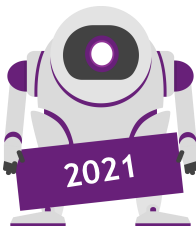
```
tye run frontend
```



```
jmfloreszazo@jmfloreszazo: /mnt/c/Projects/netcoreconf2021Feb$ tye run frontend
Loading Application Details...
Launching Tye Host...

[18:09:24 INF] Executing application from /mnt/c/Projects/netcoreconf2021Feb/frontend/frontend.csproj
[18:09:25 INF] Dashboard running on http://127.0.0.1:8000
[18:09:25 INF] Building projects
[18:09:33 INF] Launching service frontend_2cfd1114-b: /mnt/c/Projects/netcoreconf2021Feb/frontend/bin/Debug/net5.0/frontend
[18:09:33 INF] frontend_2cfd1114-b running on process id 940 bound to http://localhost:39929, https://localhost:34709
[18:09:33 INF] Replica frontend_2cfd1114-b is moving to a ready state
[18:09:34 INF] Selected process 940.
[18:09:34 INF] Listening for event pipe events for frontend_2cfd1114-b on process id 940
```

El comando identifica las aplicaciones y las construye, automáticamente establece los puertos, los puertos pueden estar en uso o entrar en conflicto si lanzáramos la aplicación de forma tradicional.



# Trabajando con Tye (4/11)

Y si entramos en:

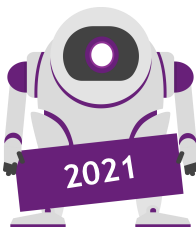
`http://127.0.0.1:8000`

Llegamos al Dashboard de Tye:



The screenshot shows the Tye Dashboard interface. At the top, there's a header bar with the 'Tye Dashboard' logo on the left and two buttons, 'Tell us what you think!' and 'About', on the right. Below the header, on the left side, is a sidebar with a 'Home' button. The main content area is titled 'Services' and contains a table with the following data:

Name	Type	Source	Bindings	Replicas	Restarts	Logs
<a href="#">frontend</a>	Project	/mnt/c/Projects/netcoreconf2021Feb/frontend/frontend.csproj	<a href="http://localhost:39929">http://localhost:39929</a> <a href="https://localhost:34709">https://localhost:34709</a>	1/1	0	<a href="#">View</a>



# Trabajando con Tye (5/11)

Y si entramos en:

`http://127.0.0.1:8000`

Entramos en el Dashboard de Tye:

 Tye Dashboard

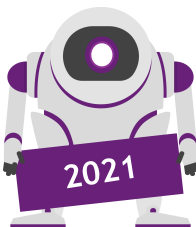
[Tell us what you think!](#)

[About](#)

[Home](#)

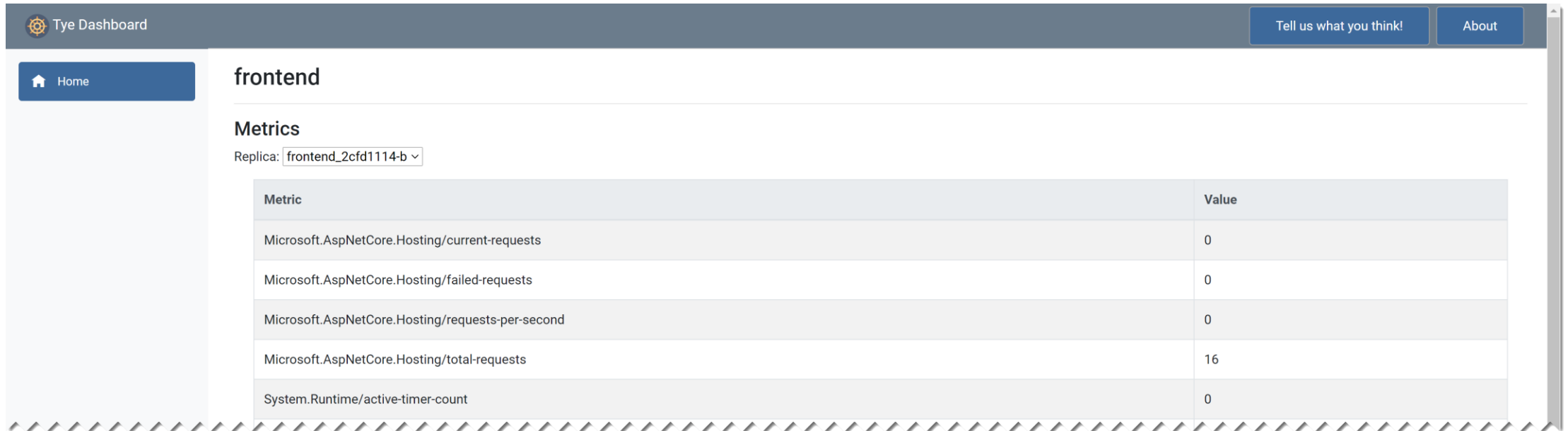
## Services

Name	Type	Source	Bindings	Replicas	Restarts	Logs
<a href="#">frontend</a>	Project	/mnt/c/Projects/netcoreconf2021Feb/frontend/frontend.csproj	<a href="http://localhost:39929">http://localhost:39929</a> <a href="https://localhost:34709">https://localhost:34709</a>	1/1	0	<a href="#">View</a>



# Trabajando con Tye (6/11)

Y este Dashboard nos permitirá ver algunos datos de interés de nuestra aplicación:

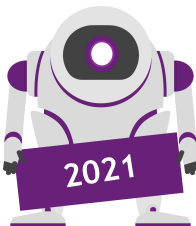


The screenshot shows the Tye Dashboard interface. At the top, there's a header with the 'Tye Dashboard' logo and two buttons: 'Tell us what you think!' and 'About'. On the left, a sidebar contains a 'Home' button. The main content area is titled 'frontend' and displays a 'Metrics' section. Below the title, a dropdown menu shows 'Replica: frontend\_2cfd1114-b'. A table lists various metrics and their current values.

Metric	Value
Microsoft.AspNetCore.Hosting/current-requests	0
Microsoft.AspNetCore.Hosting/failed-requests	0
Microsoft.AspNetCore.Hosting/requests-per-second	0
Microsoft.AspNetCore.Hosting/total-requests	16
System.Runtime/active-timer-count	0



veamos todo esto en acción ...



# Trabajando con Tye (7/11)

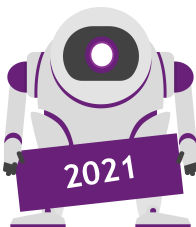
Ya estas más o menos familiarizado con el Dashboard. Es momento de añadir más servicios, ya que hasta ahora no hemos visto nada que nos anime a usar Tye.

Ahora es el momento de entrar de lleno en los microservicios, más o menos ya he dado una definición pero podemos volver a definirlo como: una aplicación que se compone de pequeñas piezas que trabajan juntas para llevar a cabo una tarea general.

La **orquestación** de estos componentes cuando trabajas de forma local puede implicar múltiples instancias de depuradores, IDEs o consolas para que todo funcione.

Añadamos un nuevo proyecto:

```
dotnet new webapi -n backend
```



# Trabajando con Tye<sup>(8/11)</sup>

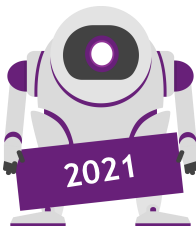
Y como Tye puede trabajar con proyectos y soluciones de .NET, creemos una solución, un punto de vista más obvio para cualquier desarrollador de .NET:

```
dotnet new sln -n netcoreconf2021feb
```

```
dotnet sln add frontend backend
```

Llegados a este punto podríamos ejecutar Tye y ver en el Dashboard las aplicaciones en ejecución, pero sin conectar.

¿Qué hacemos normalmente para conectar las aplicaciones? Pues crear un archivo de configuración o variable de entorno la URI que conecta el front con el API, y recordar que esto cambia cuando nos movemos de los entornos de DEV a PROD por ejemplo.



# Trabajando con Tye<sub>(9/11)</sub>

Aquí es donde tenemos la injerencia en nuestro proyecto, aquella que hablaba al principio de la presentación.

Tye necesita un NuGet de configuración llamado:

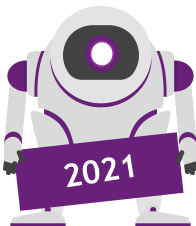
`Microsoft.Tye.Extensions.Configuration`

Esta referencia permite que nuestra aplicación acceda de forma sencilla a la URI y los puertos no solo cuando desarrollamos en nuestro entorno local, si no que también será capaz de hacerlo en cuando despliegues en Kubernetes. En resumen, no vas a tener que ir añadiendo ese mapeo de puertos o URI en cada entorno.

Sorprendente, un problema más resuelto y un quebradero de cabeza menos.

Añadamos la referencia:

```
dotnet add frontend/frontend.csproj package --prerelease Microsoft.Tye.Extensions.Configuration
```



# Trabajando con Tye<sub>(10/11)</sub>

Llegados a este punto, un inciso, no me gustan las demos en las que un paso se da por conocido y las personas que lo están siguiendo (independientemente del nivel) puedan verse ralentizadas.

Es posible que tengáis problemas de certificados:

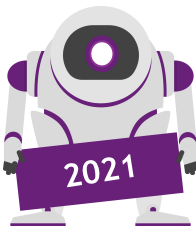
<https://aka.ms/dev-certs-trust>

Debéis seguir los pasos que nos indican. Por ejemplo, os copio y pego como establecer los certificados en Windows:

```
dotnet dev-certs https --clean  
dotnet dev-certs https --trust
```

Para WSL son más pasos y en el enlace anterior lo tenéis muy detallado o bien:

[https://github.com/dotnet/tye/blob/master/docs/tutorials/hello-tye/00\\_run\\_locally.md](https://github.com/dotnet/tye/blob/master/docs/tutorials/hello-tye/00_run_locally.md)



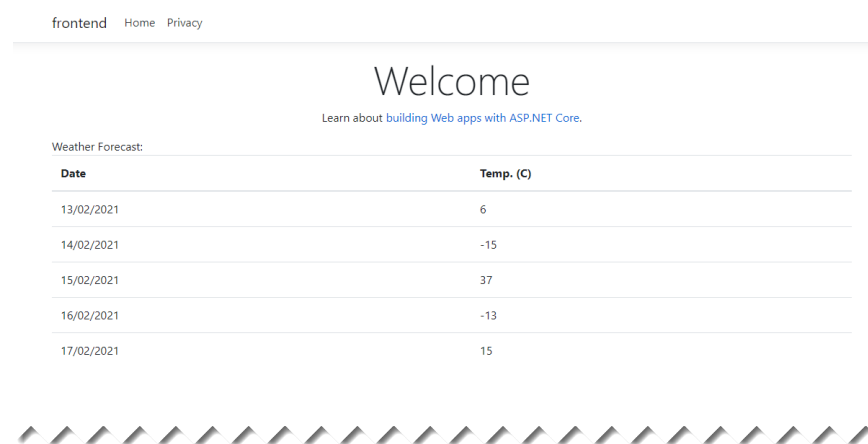


# Trabajando con Tye (11/11)

Ejecutamos:

```
tye run --dashboard
```

Y ya tenemos nuestra aplicación conectada:

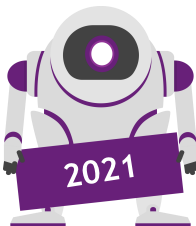


The screenshot shows a web application interface. At the top, there is a navigation bar with links: "frontend", "Home", and "Privacy". Below this, the main heading is "Welcome", followed by a subtitle: "Learn about building Web apps with ASP.NET Core." The main content area is titled "Weather Forecast:" and contains a table with two columns: "Date" and "Temp. (C)". The table lists five dates from 13/02/2021 to 17/02/2021 with corresponding temperatures: 6, -15, 37, -13, and 15. The bottom of the page features a decorative border of small, repeating geometric shapes.

Date	Temp. (C)
13/02/2021	6
14/02/2021	-15
15/02/2021	37
16/02/2021	-13
17/02/2021	15



veamos todo esto en acción ...



# Dependencias Externas (1/4)

Añadir más proyectos y ejecutarlos es como habéis visto muy sencillo. Es repetir el mismo proceso que ya os he presentado.

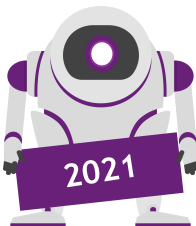
Sin embargo, el mundo de los microservicios, las aplicaciones se usan otros servicios: bases de datos, colas, otros procesos, etc.

Para poder hacer uso de estos servicios y tener más control sobre ellos, Tye usa un archivo de configuración llamado:

```
tye.yaml
```

Para generar el fichero debemos ejecutar un comando que automáticamente crear un archivo basado en los proyectos del directorio en el que nos encontramos:

```
tye init
```



# Dependencias Externas (2/4)

Para nuestro ejemplo:

```
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: netcoreconf2021feb
8 services:
9 - name: frontend
10   project: frontend/frontend.csproj
11 - name: backend
12   project: backend/backend.csproj
13
```

Doy un paso para atrás y os muestro como se está gestionando el lanzar varias soluciones, esto de momento, debemos escribirlo de forma manual y esta en fase de trabajo:

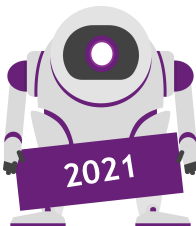
```
#The Name must be the same in each referenced tye.yaml
name: MultipileSolutions

#define multiple services here
services:

- name: supportingServices
  include: tye-services.yaml

- name: solution1
  include: /solution1/tye-micro1.yaml

- name: solution2
  include: /solution2/tye-micro2.yaml
```



# Dependencias Externas (3/4)

Volvemos sobre nuestros pasos y vamos a ampliar el fichero de configuración estándar de Tye para añadirle la extensión Redis:

```
- name: redis
  image: redis
  bindings:
    - port: 6379
      connectionString: "${host}:${port}"
- name: redis-cli
  image: redis
  args: "redis-cli -h redis MONITOR"
```

```
cd backend
dotnet add package Microsoft.Extensions.Caching.StackExchangeRedis
```

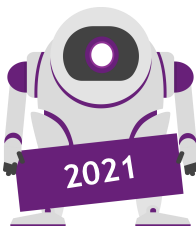
Podéis ver que este servicio esta usando un contenedor para cargar Redis:

```
PS C:\Projects\netcoreconf2021Feb> tye run --dashboard
Loading Application Details...
Launching Tye Host...

[13:03:35 INF] Executing application from C:\Projects\netcoreconf2021Feb\tye.yaml
[13:03:36 INF] Dashboard running on http://127.0.0.1:8000
[13:03:38 INF] Running docker command pull redis
[13:03:39 INF] redis: Using default tag: latest
```

## Services

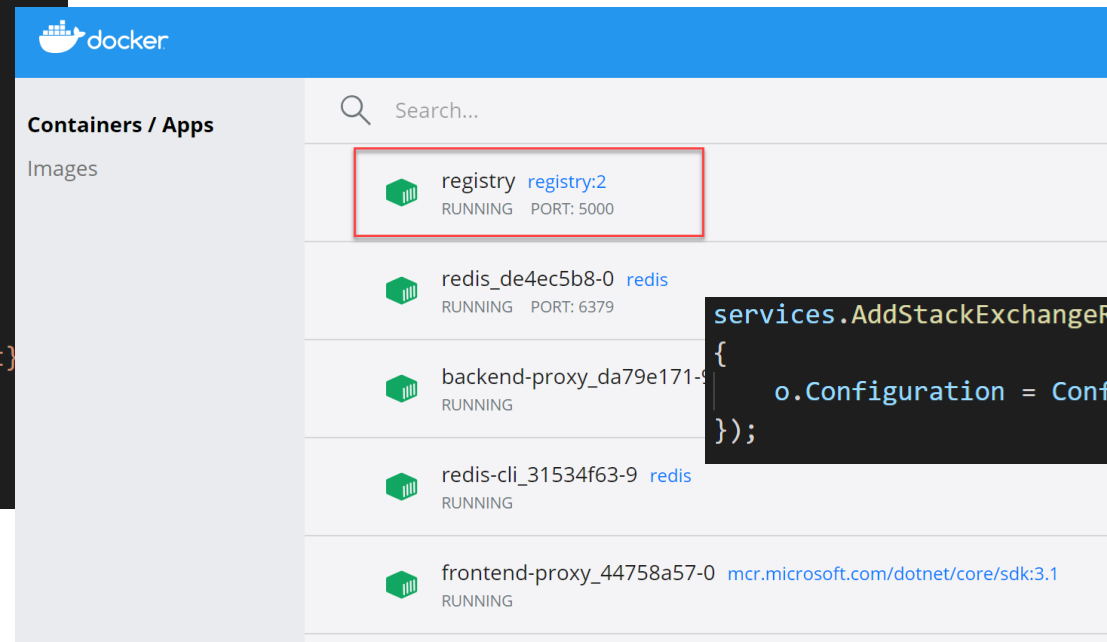
Name	Type	Source	Bindings	Replicas
frontend	Project	C:\Projects\netcoreconf2021Feb\frontend\frontend.csproj	http://localhost:64563https://localhost:64564	1/1
backend	Project	C:\Projects\netcoreconf2021Feb\backend\backend.csproj	http://localhost:64565https://localhost:64566	1/1
redis	Container	redis	tcp://localhost:6379	1/1
redis-cli	Container	redis	none	1/1



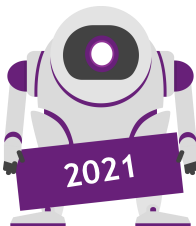
# Dependencias Externas (4/4)

Como podéis observar esta haciendo el binding automáticamente a Redis y solo hemos modificado código del backend, además de aprovechar para registrar el contenedor:

```
name: netcoreconf2021feb
registry: localhost:5000
services:
- name: frontend
  project: frontend/frontend.csproj
- name: backend
  project: backend/backend.csproj
- name: redis
  image: redis
  bindings:
  - port: 6379
    connectionString: "${host}:${port}"
- name: redis-cli
  image: redis
  args: "redis-cli -h redis MONITOR"
```



```
services.AddStackExchangeRedisCache(o =>
{
    o.Configuration = Configuration.GetConnectionString("redis");
});
```



# Ejecutando las aplicaciones en contenedores

Nuestro siguiente paso meter en un contenedor tanto el backend como el frontend, con este sencillo comando:

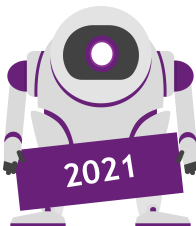
```
tye run --docker
```

Tye hace un pull de las imágenes base, construye el proyecto y la imagen, para posteriormente ejecutar las imágenes, creando también las redes para descubrir los servicios. Ahora mismo todos los servicios están ejecutándose en imágenes basadas en Linux. Nos hemos evitado conocer la sintaxis de dockerfile o docker-compose.

```
[15:31:41 INF] mcr.microsoft.com/dotnet/aspnet:5.0: 513626bd05cb: Download complete
[15:31:44 INF] mcr.microsoft.com/dotnet/aspnet:5.0: 081e89d42aee: Verifying Checksum
[15:31:44 INF] mcr.microsoft.com/dotnet/aspnet:5.0: 081e89d42aee: Download complete
[15:31:46 INF] mcr.microsoft.com/dotnet/aspnet:5.0: 5ccb476d6b8b: Verifying Checksum
[15:31:46 INF] mcr.microsoft.com/dotnet/aspnet:5.0: 5ccb476d6b8b: Download complete
[15:31:57 INF] mcr.microsoft.com/dotnet/aspnet:5.0: 752dcc4c3a04: Verifying Checksum
[15:31:57 INF] mcr.microsoft.com/dotnet/aspnet:5.0: 752dcc4c3a04: Download complete
```

## Services

Name	Type	Source
<a href="#">frontend</a>	Container	mcr.microsoft.com/dotnet/aspnet:5.0
<a href="#">backend</a>	Container	mcr.microsoft.com/dotnet/aspnet:5.0
<a href="#">redis</a>	Container	redis
<a href="#">redis-cli</a>	Container	redis



# Desplegando tus aplicaciones (1/4)

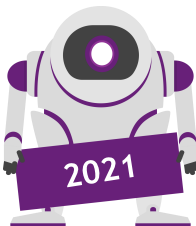
Tye despliega únicamente a K8S por tres razones:

- Es un estándar de facto.
- Es neutral con respecto a los proveedores.
- Y esta respaldada por una comunidad muy solida.

El despliegue de microservicios no es una tarea sencilla por diversas razones:

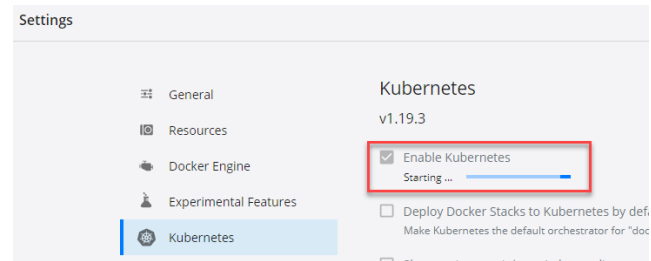
- Creación de contenedores.
- Almacenar las imágenes en un repositorio.
- Los ficheros de configuración de K8S.
- Las cadenas de conexión (secrets).
- Y seguramente en tu caso personal algún punto más.

Tye esta aquí para ayudar, al menos es mi percepción.



# Desplegando tus aplicaciones (2/4)

En esta presentación no voy a desplegar a AKS ni a ninguna otra nube, vamos a trabajar en local:



Primero debemos montar Redis en K8S, Tye no lo va a montar:

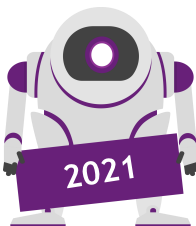
```
kubectl apply -f redis.yaml
```

Una forma de ver que hace Tye es usar el comando de despliegue de forma interactiva:

```
tye deploy --interactive
```

```
Validating Secrets...
```

```
Enter the connection string to use for service 'redis': redis:6379
```





# Desplegando tus aplicaciones (3/4)

Vamos a comprobar que efectivamente nuestros pods y secrets están:

```
PS C:\Projects\netcoreconf2021Feb> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
backend-78cb47c79-mvx24             1/1     Running   0           2m44s
frontend-6446b789cf-744mm          1/1     Running   0           2m44s
redis-784d86867f-skr88              1/1     Running   0           3m42s
```

```
PS C:\Projects\netcoreconf2021Feb> kubectl get secrets
NAME                                TYPE                                DATA   AGE
binding-production-redis-secret     Opaque                              1       3m5s
default-token-42kn1                 kubernetes.io/service-account-token 3       132m
```

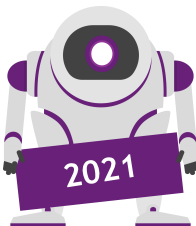
Lanzamos:

```
kubectl port-forward svc/frontend 5000:80
```

Ahora podemos acceder a nuestra web entrando en: `http://127.0.0.1:5000/`

Y quitamos la aplicación desplegada de nuestro K8S:

```
tye undeploy
```



# Desplegando tus aplicaciones (4/4)

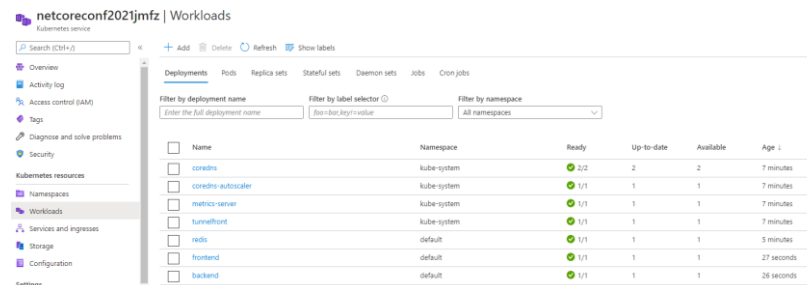
Si eliminamos la línea de `tye.yaml` en vez de dejarla apuntando a local:

```
registry: localhost:5000
```

Nos pedirá donde desplegar de forma interactiva, podréis observar que podemos indicar un despliegue a AKS, por ejemplo:

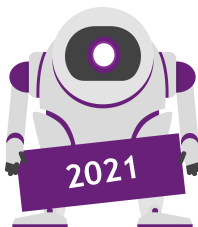
```
Verifying kubectl installation...
Verifying kubectl connection to cluster...
Enter the Container Registry (ex: 'example.azurecr.io' for Azure or 'example' for dockerhub):
```

Para ello he creado un Container Registry y una instancia AKS y hacemos el deploy con los nuevos endpoints.



The screenshot shows the 'netcoreconf2021jmfz | Workloads' page in the Azure portal. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Kubernetes resources, Namespaces, Workloads (selected), Services and Ingress, Storage, Configuration, and Settings. The main area displays a table of deployments with filters for deployment name, label selector, and namespace. The table lists several deployments, including 'coredns', 'coredns-autoscaler', 'metrics-server', 'tunnelford', 'redis', 'frontend', and 'backend', all in a 'Ready' state.

Name	Namespace	Ready	Up-to-date	Available	Age
coredns	kube-system	2/2	2	2	7 minutes
coredns-autoscaler	kube-system	1/1	1	1	7 minutes
metrics-server	kube-system	1/1	1	1	7 minutes
tunnelford	kube-system	1/1	1	1	7 minutes
redis	default	1/1	1	1	9 minutes
frontend	default	1/1	1	1	27 seconds
backend	default	1/1	1	1	28 seconds



# Conclusiones

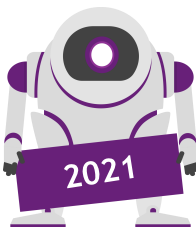
---

Como podréis observar, hemos podido probar todos nuestros microservicios en local y de forma aislada, hemos podido desplegar sin mayores complicaciones, también en cuestión de minutos podemos llevarlo a un AKS.

Es cierto que estamos ante una herramienta experimental, que le faltan módulos y que aun queda por madurar ciertos aspectos.

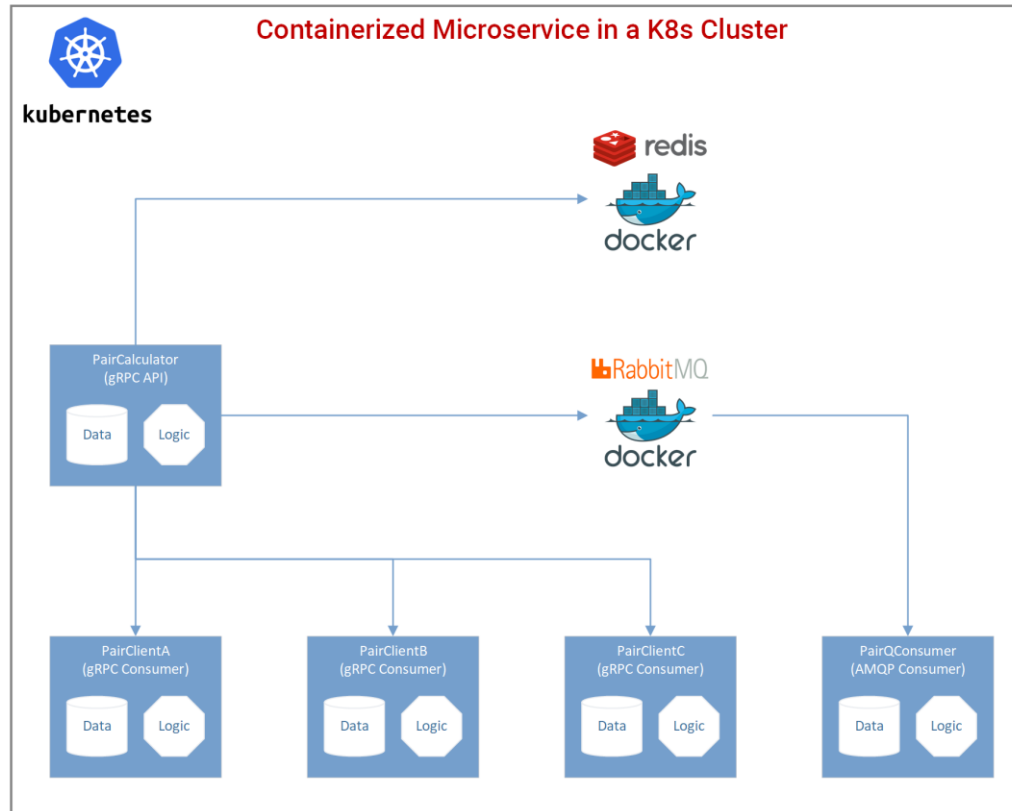
Pero lo que esta claro es que el objetivo de hacernos la vida más fácil cuando trabajamos con microservicios, lo esta logrando.

Sin lugar a dudas es un proyecto a seguir y que augura felicidad a los desarrolladores.

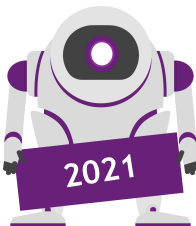


# Y si aun quieres más...

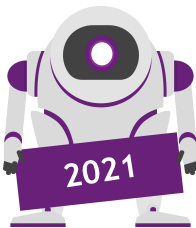
También os dejo un workshop:



[“Arquitectura de Microservicios con contenedores usando .NET5”](#)



# SPONSORS





More information:  
[info@netcoreconf.com](mailto:info@netcoreconf.com)  
@Netcoreconf

Visit on:  
[netcoreconf.com](http://netcoreconf.com)