

JMFLORESZAZO

software craftsman & digital creative

Arquitectura de Microservicios con Contenedores usando .NET5

Bienvenidos

Acerca de...



Jose María Flores Zazo, autor

*¡Hola! Gracias por entrar en
“Arquitectura de Microservicios con contenedores usando .NET5”.
Espero poder aportarte los conocimientos mínimos y necesarios
para que puedas ponerlo en práctica.*



Introducción

Resumen del workshop

CONTAINERS & MICROSERVICES – Contenedores & Microservicios

La arquitectura basada en microservicios hoy en día es una de las arquitecturas más populares para desarrollar nuevos productos y servicios.

Los contenedores nos permitirán desarrollar e implementar microservicios a la perfección.

Para lograr nuestro objetivo vamos a usar .NET Core y .NET 5 ya que soporta el desarrollo e implementación de microservicios gracias a una serie de características además de ofrecer un gran rendimiento en escenarios multiplataforma.

No solamente vamos a centrarnos en contenedores y microservicios. Aprovecharemos el workshop para ampliar sus habilidades. Tales como:

- ¿Qué es el protocolo de comunicación gRPC y cuales son sus beneficios y usos con .NET 5?
- ¿Qué es WSL?
- ¿Qué es Tye y cómo podemos usarlo ampliamente en nuestra aplicación de microservicios?
- ¿Qué es la orquestación de contenedores y sus conceptos con respecto a Kubernetes?
- Arquitectura e implementación de microservicios a través de un ejemplo.

Al final de este workshop va a poder implementar aplicaciones basadas en microservicios usando las ultimas herramientas de Microsoft con mucha facilidad.



Introducción

¿Por qué ver otros temas y no ir directamente a los microservicios?

Cuando se habla de aplicaciones de microservicios, podría preguntarse, ¿por qué no revelar la arquitectura y simplemente implementarla? Esto es natural y no hay nada de malo en hacer esto. Estamos aprendiendo sobre las otras tecnologías en este contexto y van a ayudarnos en nuestro viaje hacia el desarrollo de aplicaciones basadas en microservicios. Las tecnologías circundantes han sido mejoradas con la introducción de .NET 5.

Por tanto:

- gRPC nos ayudará a seleccionar la última y más rápida tecnología de comunicación entre los microservicios.
- WSL 2 hace nuestra vida más fácil: desarrollo y depuración de las aplicaciones de microservicios .NET en máquinas Windows.
- Tye nos ayudará a atar todo dentro de la máquina de desarrollo local y nos facilitará mover rápidamente nuestro proceso hacia una compleja infraestructura basada en la nube.



Introducción

Requisitos previos y herramientas

01

Windows 10, **Windows Subsystem for Linux (WSL)**

02

Entorno de desarrollo
Visual Studio Code





TEORÍA

gRPC

Protocolo de comunicación

gRPC – Remote Procedure Calls

gRPC es un marco de trabajo de código abierto que fue originalmente desarrollado por Google en 2015 y ahora forma parte de la [Cloud Native Computing Foundation \(CNCF\)](#), proporciona el estándar global para las ofertas nativas de la nube.

RPC significa **llamada de procedimiento remoto**, lo que significa que se hace una llamada de método a través de un límite de red, ya sea en la misma máquina, en una red privada o en una red pública. gRPC es un RPC moderno, multiplataforma y de alto rendimiento que está disponible en casi todos los lenguajes de programación más populares. Un servidor gRPC expone un método de llamada remota, que un cliente gRPC llama a través de una función local que invoca internamente una función en una máquina remota que sirve para alguna operación comercial.

Hay tecnologías RPC similares, tales como versiones muy antiguas de la RMI de Java, Thrift, JSON-RPC, WCF, y otras. Pero la comunidad de código abierto y la CNCF están más volcadas en gRPC, lo que se traduce en un amplio nivel de soporte multiplataforma.

El gRPC utiliza las tecnologías HTTP/2 y Protobuf.



gRPC

HTTP/2 y Protobuf

HTTP/2 habilita la comunicación cliente/servidor en modo bidireccional, además de habilitar la transmisión de llamadas. También permite la multiplexación, lo que permite realizar varias solicitudes en paralelo a través de la misma conexión. HTTP/2 también habilita el encuadre binario para el transporte de datos, a diferencia de HTTP/1.1, que solo se basaba en texto.

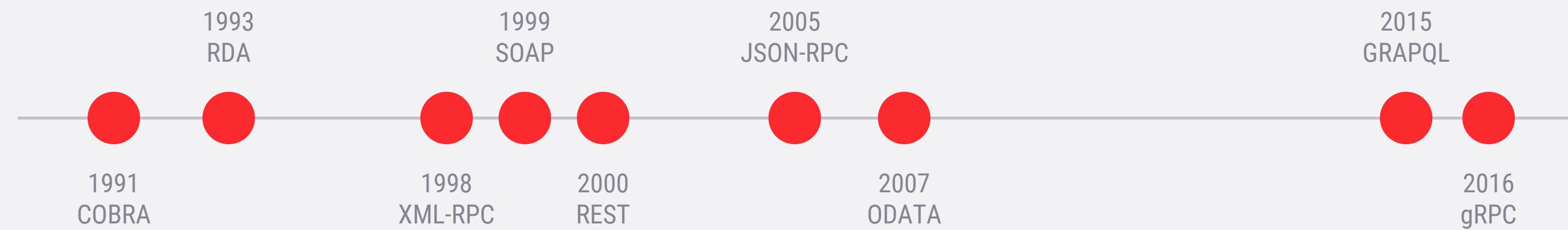
Protobuf o Protocol Buffers, es una tecnología multiplataforma que define el Interface Definition Language (IDL) para declarar las interfaces de servicio que se llaman de forma remota. gRPC utiliza un enfoque de contrato mediante el cual los contratos se definen en formato Protobuf, generalmente en archivos *.proto. Protobuf define el formato legible por humanos y su representación a nivel de máquina y maneja la traducción entre los dos. Las bibliotecas de gRPC para cada idioma/plataforma permiten la definición del IDL de Protobuf en el formato de tipo propio del idioma, por ejemplo, tipos C# para manejar objetos request y response.

Para la comunicación entre servicios, podemos utilizar mecanismos de comunicación distintos a RPC, como REST y Simple Object Access Protocol (SOAP), entre otros. Veamos rápidamente las principales diferencias entre las opciones más populares en el mundo .NET.



gRPC

Comparación (1/2)



gRPC

Comparación (2/2)

	RPC	SOAP	REST	GraphQL
Organizado en términos de	Llamada a procedimiento local	Mensaje con estructura de sobre	cumplimiento de seis restricciones arquitectónicas	Sistema esquemático y tipado
Formato	JSON, XML, Protobuf, Thrift, FlatBuffers	Solo XML	XML, JSON, HTML, texto plano	JSON
Curva de aprendizaje	Fácil	Complejo	Fácil	Medio
Soporte de la Comunidad	Amplio	Pequeño	Amplio	En crecimiento
Casos de Uso	API orientadas a comandos y acciones; comunicación interna de alto rendimiento en sistemas de microservicios masivos.	Pasarelas de pago, gestión de identidades, soluciones CRM, servicios financieros y de telecomunicaciones, soporte para sistemas heredados	API públicas, aplicaciones simples basadas en recursos	API para móviles, sistemas complejos, microservicios



gRPC

En detalle: gRPC vs REST APIs

gRPC	REST APIs
Basado en RPC	Usa verbos HTTP
Manda el contrato	Manda el contenido
Contrato gestionado por el fichero proto	Contrato es opcional y gestionado via OpenAPI (swagger). El esquema del contrato no es obligatorio
El fichero proto usa Protobuf IDL para la definición de la interface	Las especificaciones personalizadas de la interface están definidas via OpenAPI usando formato JSON
Protocolo Punto-Punto generalmente entre cliente y servidor con un contrato fijo	Protocolo Punto-Punto entre cliente y servidor siendo flexible o desconocido (ejemplo arquitectura HATEOAS)
Contenido binario	Contenido en JSON, fácilmente entendible por humanos y muy simple
Esconde complejidades remotas	Fuerza a usar verbos HTTP en los recursos del API
Protocolo binario, pequeño tamaño, serialización Protobuf, compresión HTTP/2 y bajo uso de red	JSON siempre, tamaño medio, serialización JSON.
Focalizado en alto rendimiento	Focalizado en el uso de un alto numero de consumidores
HTTP/2 streaming, HTTP/2 multiplexing, permite múltiples llamadas en la misma conexión TPC, alto rendimiento y bajo consumo	El estándar no especifica ningún beneficio de HTTP/2



gRPC

Novedades en .NET 5

gRPC ya estaba disponible con .NET Core, pero ahora con .NET 5, tiene más características y mejoras. Echemos un vistazo rápido a ellas:

- gRPC fue implementado como un envoltorio sobre su implementación de C++, pero ahora con .NET 5, ha sido completamente reescrito en C# administrado. Se traduce en más rendimiento evitando el cambio de contexto del código seguro a inseguro.
- Las mejoras en el servidor Kestrel que comúnmente alberga los servicios de gRPC son las siguientes: Reducción de las asignaciones, Mejora de la concurrencia y Compresión del encabezado de la respuesta
- gRPC en sistemas operativos Windows a través del servidor IIS, gracias a la nueva versión de HTTP.sys que permitió mejoras en el conjunto de características para HTTP/2 a nivel de sistema operativo que no estaban antes presentes.
- Inter-Process Communication (IPC) ahora puede usar el protocolo gRPC.
- gRPC-Web añade soporte para los navegadores para que las aplicaciones cliente basadas en JavaScript o Blazor WebAssembly puedan comunicarse a través del protocolo gRPC.



*.PROTO – [file_name].proto

Desde mi punto de vista para comprender la esencia de gRPC, es una buena idea tener una visión de cómo se ve un archivo proto usa el Protocolo de Buffers IDL.

El siguiente ejemplo es archivo proto que vamos a utilizar cuando desarrollemos la aplicación de microservicios en este workshop. Básicamente nos habla de la interfaz de un servicio con el nombre de PairCalculator. Tiene un método, a saber, IsItPair, con una petición y una respuesta:

```
paircalculator > Protos > pair.proto
 1 syntax = "proto3";
 2
 3 option csharp_namespace = "microservicesapp";
 4
 5 package pair;
 6
 7 //Service definition for the microservice: PairCalculator
 8 service PairCalculator {
 9   //Sends a true false if the number is pair or not
10   rpc IsItPair (PairRequest) returns (PairReply);
11 }
12
13 //Request message
14 message PairRequest {
15   int64 number = 1;
16 }
17
18 //Response message containing the result
19 message PairReply {
20   bool isPair = 1;
21 }
```



gRPC

Ejemplo de un fichero Proto (2/2)

gRPC tiene cuatro tipos diferentes de métodos de RPC. IsItPair en el ejemplo anterior es del tipo unario.

Los tipos de métodos RPC son los siguientes:

- RPC unario: Este tipo de métodos tienen una llamada de método única con una sola solicitud y una sola respuesta.
- RPC streaming de servidor: En este modo, el cliente envía una única solicitud y el servidor devuelve un flujo de mensajes.
- RPC streaming de cliente: En este modo, el cliente abre un flujo y envía una secuencia de mensajes al servidor. El servidor lee todos los mensajes y luego devuelve la respuesta para marcar el final de la llamada.
- RPC de transmisión bidireccional: En este modo, tanto el servidor como el cliente envían secuencias de mensajes en secuencia. En este caso, ambos tienen dos flujos de lectura-escritura diferentes. Los dos flujos son independientes y pueden ser leídos de cualquier manera. Por ejemplo, un servidor puede decidir leer primero todo el flujo de entrada y luego escribir en el flujo de salida, o leer un solo mensaje y escribir una sola respuesta. También puede hacer cualquier otra combinación de lectura y escritura de los mensajes a los respectivos flujos.



*.PROTO – [file_name].proto

Si estás depurando una aplicación de APS.NET (ya sea con REST o gRPC) en una máquina Windows, necesitarás aceptar e instalar un certificado SSL, que nos permitirá habilitar el desarrollo contra HTTPS.

Puedes ejecutar el siguiente comando:

```
dotnet dev-certs https –trust
```

O ir al siguiente sitio para ampliar más información:

<https://docs.microsoft.com/en-us/aspnet/core/security/enforcing-ssl>



WSL

¿Qué es?

WSL nos permite ejecutar un entorno Linux completo dentro de Windows sin usar una configuración de arranque dual o mediante una VM. Actualmente existen dos versiones: WSL1 y WSL2. Lógicamente WSL 2 es una versión mejorada; es la que se recomienda y la que vamos a usar. En **WSL 2** han incrementado el rendimiento del sistema de archivos y han añadido compatibilidad total con las llamadas al sistema.

Con WSL, por ejemplo, podemos depurar nuestras aplicaciones .NET que deban ejecutarse en un entorno de producción Linux en lugar de nuestra maquina Windows. También puedes depurar un contenedor basado en Linux dentro de una máquina Windows.

Además, con WSL, puedes instalar una serie de distribuciones de Linux disponibles en el Store, y la línea de comando Linux, como grep para nuestra unidad C. También podrás montar un MySQL, MongoDB y PostgreSQL dentro de WSL. Por supuesto podrás usar APT de Ubuntu y dpkg como administrado en un Debian.

Para una comparación detallada de WSL 1 y WSL 2, consulte el siguiente enlace:

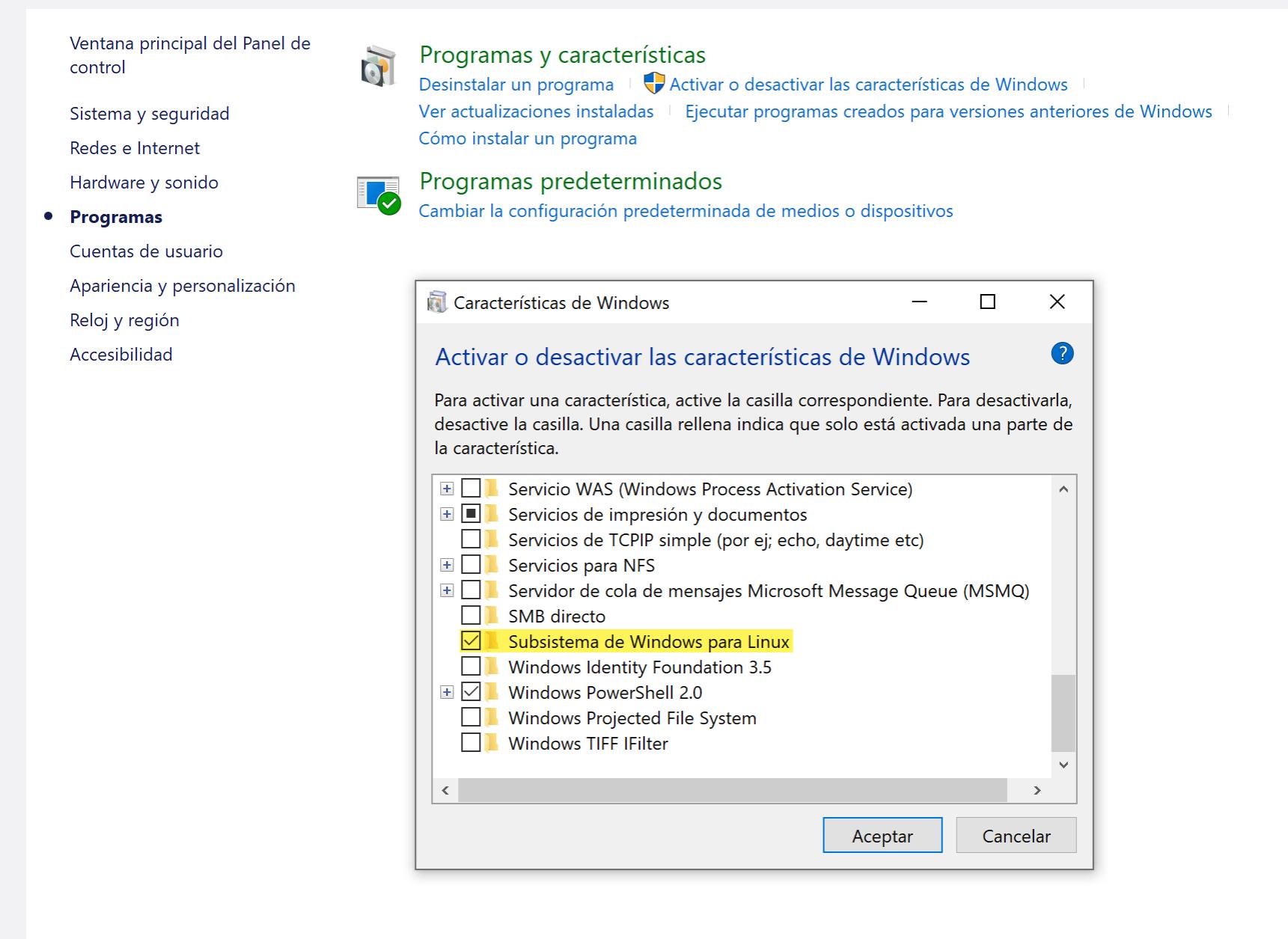
<https://docs.microsoft.com/en-us/windows/wsl/compare-versions>



WSL

Preparación de la máquina (1/3)

Paso 1. Instalar Windows Subsystem para Linux:



WSL

Preparación de la máquina (2/3)

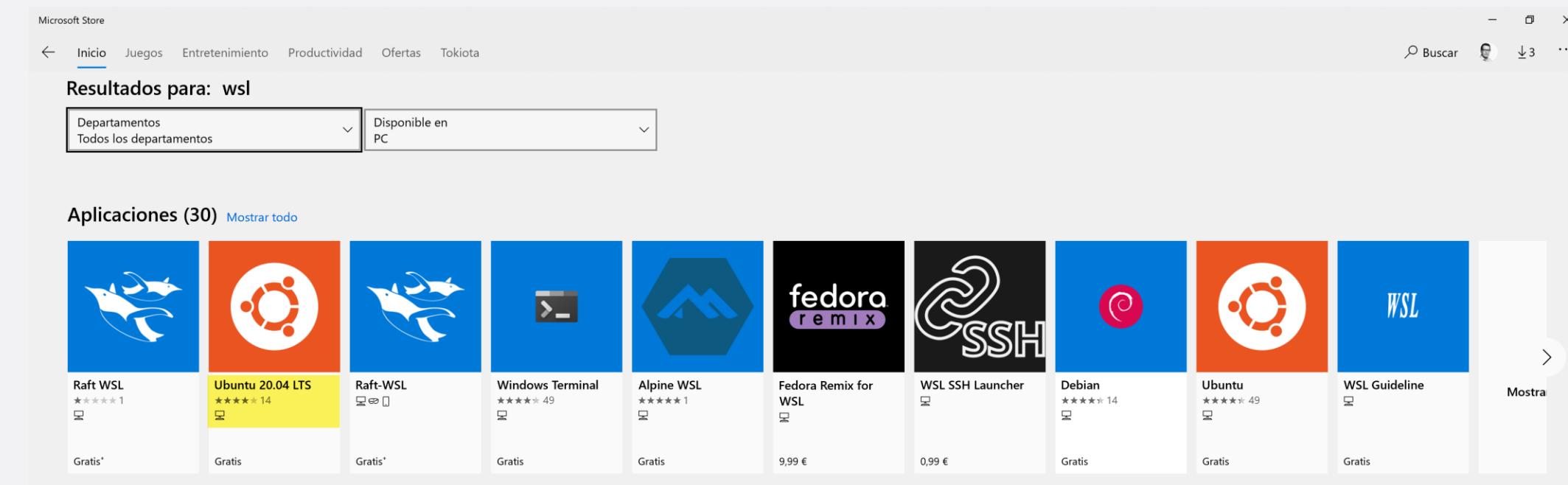
Paso 2. Instalar el Kernel de Linux a través de update package.

https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi

Paso 3. Establecer WSL 2 como la versión por defecto.

```
wsl --set-default-version 2
```

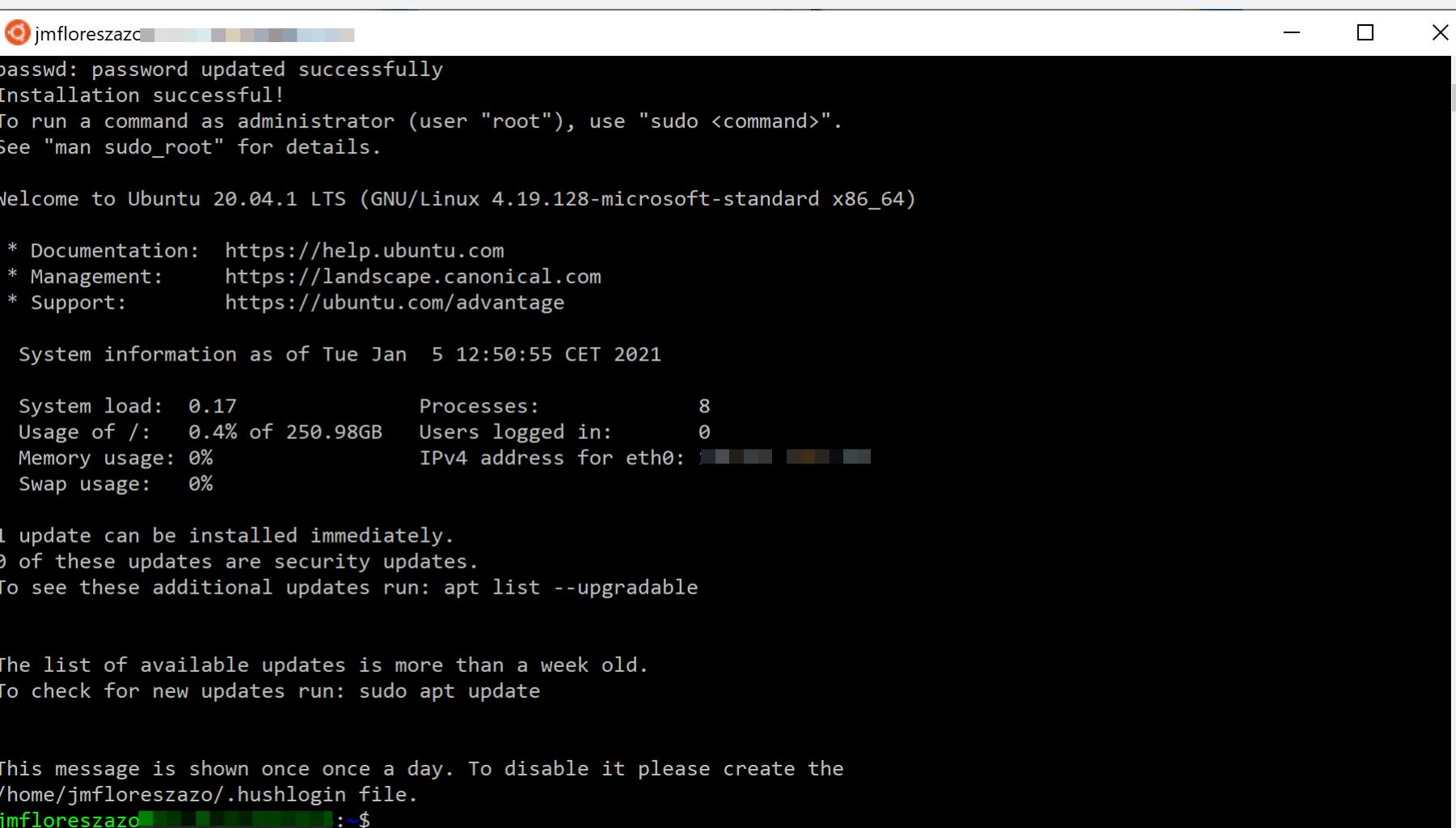
Paso 4. Seleccionar la distribución de Linux con la que te sientas más cómodo. Deberás usar la instalación desde Microsoft Store de tu Windows.



WSL

Preparación de la máquina (3/3)

Paso 5. En mi caso he instalado Ubuntu, y para ver si correctamente esta instalado en tu maquina escribe Ubuntu o WS en el menú de inicio de Windows:



The screenshot shows a terminal window titled "jmfloreszacz" with a dark background. It displays the following text:
passwd: password updated successfully
Installation successful!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 4.19.128-microsoft-standard x86_64)

* Documentation: <https://help.ubuntu.com>
* Management: <https://landscape.canonical.com>
* Support: <https://ubuntu.com/advantage>

System information as of Tue Jan 5 12:50:55 CET 2021

System load: 0.17 Processes: 8
Usage of /: 0.4% of 250.98GB Users logged in: 0
Memory usage: 0% IPv4 address for eth0: [REDACTED]
Swap usage: 0%

1 update can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

This message is shown once once a day. To disable it please create the
/home/jmfloreszazo/.hushlogin file.
jmfloreszazo [REDACTED] :~\$



VS Code

Algunas extensiones interesantes (1/3)

Para incrementar la productividad dentro de VS Code, sugiero que se añadan las siguientes extensiones:

- Docker Desktop para Windows
- Remote WSL
- Remote Containers
- Kubernetes

Si no estas familiarizado con Linux, para poder acceder a tu directorio de repos:



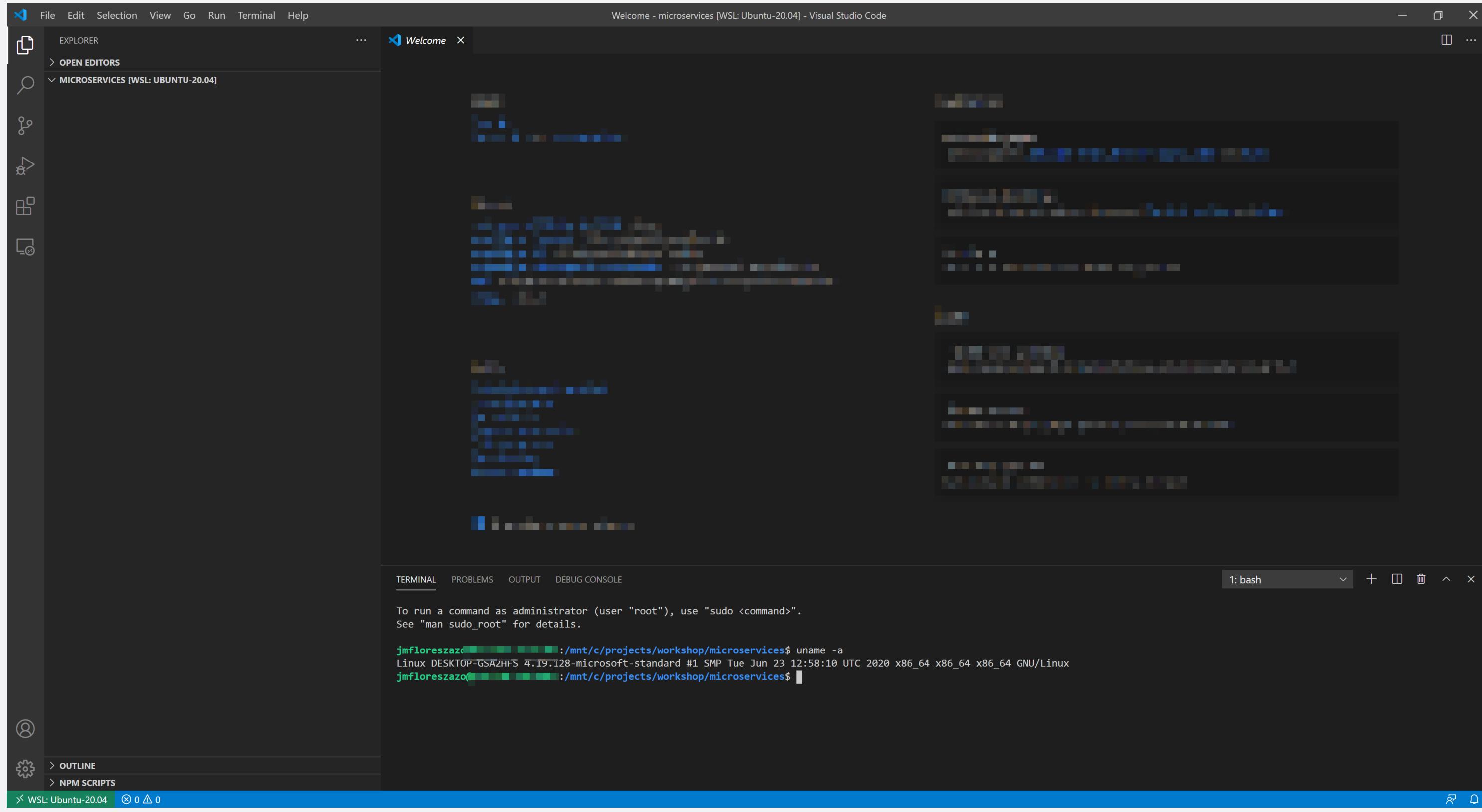
A screenshot of a terminal window within the VS Code interface. The window title bar shows the user's name 'jmfloreszazo' and the path '/mnt/c/projects'. The terminal itself has a dark background and displays the following command-line session:

```
jmfloreszazo@: ~ /mnt/c/projects
jmfloreszazo@: ~ /mnt/c/
jmfloreszazo@: ~ /mnt/c$ cd projects
jmfloreszazo@: ~ /mnt/c/projects$ ls
MyRepos  WorkRepos  Workshop
jmfloreszazo@: ~ /mnt/c/projects$
```



VS Code

Algunas extensiones interesantes (2/3)



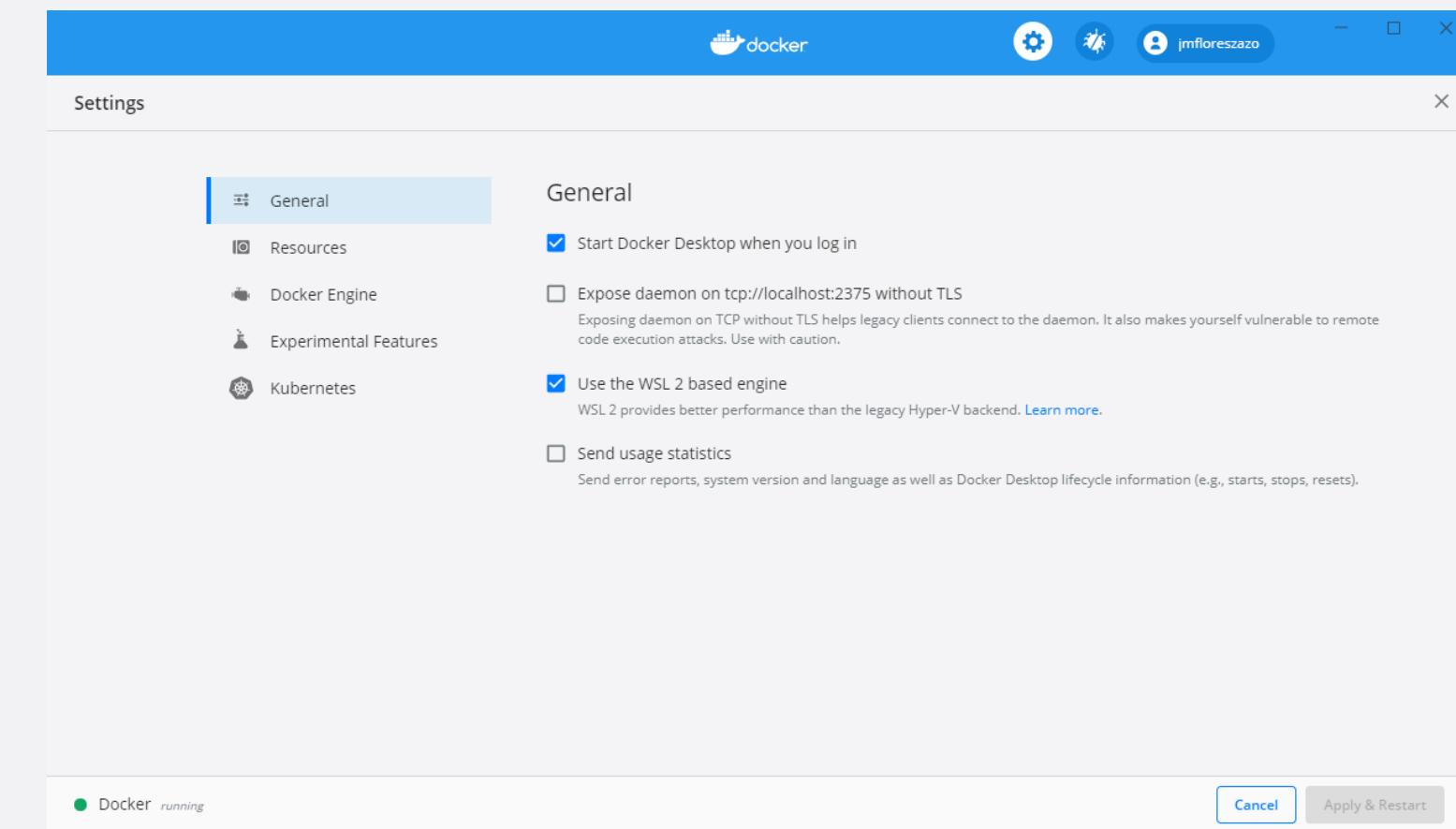
VS Code

Algunas extensiones interesantes (3/3)

Seguir los pasos que se indican para instalar .NET SDK o .NET Runtime en la versión que tengas de Ubuntu:

<https://docs.microsoft.com/en-us/dotnet/core/install/linux-ubuntu>

Y comprobar que Docker Desktop esta presente en Windows, recuerda marcar “Use the WSL 2 base engine”:



Tye

¿Qué es?

Microsoft está continuamente recibiendo comentarios de los desarrolladores, y más aún después de convertirse en uno de los principales valedores del open source. Cuando se dieron cuenta del quebradero de cabeza que sufrimos los desarrolladores de .NET que construimos aplicaciones basadas en microservicios, depurándolas y desplegándolas en entornos de prueba y producción. Decidieron formar un equipo para construir una herramienta que ayudara a simplificar todo esto, y de ahí nació una nueva herramienta llamada Tye (también conocida como Proyecto Tye).

Desde Noviembre de 2020 podemos comenzar a usar esta herramienta experimental.

Tye es una herramienta de línea de comandos desde la cual podemos ejecutar, probar y depurar con facilidad aplicaciones en contenedores en nuestra maquina de desarrollo local sin preocuparnos de otros servicios y dependencias entre otros contenedores o Docker, o incluso un orquestador en nuestra maquina. Además, permite desplegar microservicios en un cluster de Kubernetes, si lo necesita.

Ya sabes que es Tye y para que sirve. Más adelante la usaremos en nuestra aplicación de microservicios.

Si desea conocer más sobre este proyecto: <https://github.com/dotnet/tye>

Y aquí tienes un walkthrough: <https://github.com/dotnet/tye/tree/master/docs/tutorials/hello-tye>

De momento dejamos este punto en suspenso.



Contenedores y orquestación

Introducción

Los contenedores proporcionan aislamiento de los procesos, lo que significa que se ejecutan en un espacio aislado. También proporcionan un control efectivo de los recursos de I/O de una forma que en realidad es una especie de sistema suboperativo alojado en un sistema operativo existente con límites virtuales.

El sistema operativo que alberga el contenedor o contenedores puede estar funcionando en una máquina física o en sí mismo dentro de una máquina virtual.

Un contenedor es un lugar aislado que está virtualmente aislado utilizando el proceso y la tecnología de aislamiento del espacio de nombres que proporciona el sistema operativo anfitrión, de modo que una aplicación que se ejecuta dentro del contenedor se ejecuta sin afectar al resto del sistema fuera del límite del contenedor.

La idea de este tipo de entorno virtual fue propuesta inicialmente por un profesor del MIPT en 1999. Fue básicamente una mejora del modelo **chroot** que tenía tres componentes principales:

- Grupos de procesos aislados por espacios de nombres.
- Un sistema de archivos para compartir parte del código.
- Proporcionar el aislamiento y la gestión de recursos clave como las redes.

Algunas de las tecnologías de contenedores del pasado incluyen chroot de los años 80, las Jails FreeBSD del 2000, Solaris Zones de 2004, OpenVZ de 2005, cgroups de 2006, LXC de 2008, Imctfy de 2013 y Docker de 2013, entre otras.

La propia Docker utiliza internamente el **containerd**, que es un estándar del sector, también utilizado por Kubernetes.



Contenedores y orquestación

Algunas definiciones fundamentales de contenedores (1/3)

Con la popularidad de los contenedores en estos últimos tiempos, algunos conceptos han adoptado una definición formal. Vemos alguno de estas definiciones:

- **Container host**: es el sistema operativo que funciona en una máquina física o una máquina virtual. Este sistema operativo en realidad alberga contenedores en su interior.
- **Container operating system image**: Los contenedores se despliegan en forma de capas de imágenes y se apilan. La capa base proporciona los fundamentos de un entorno de sistema operativo para las aplicaciones deseadas eventualmente dentro del contenedor.
- **Container image**: La imagen de un contenedor está destinada a las aplicaciones. Contiene el sistema operativo de base, la aplicación y todas las dependencias de la aplicación en forma de capas de contenedor, junto con todas las configuraciones necesarias para desplegar y ejecutar el contenedor.
- **Container registry**: es el lugar donde se guardan las imágenes de los contenedores que pueden ser descargados bajo demanda. Los registros de contenedores más populares son el Docker Hub y el Azure Container Registry. Se puede tener un registro de contenedores on-premises o temporalmente en una máquina de desarrollo local.



Contenedores y orquestación

Algunas definiciones fundamentales de contenedores (2/3)

- **Configuration:** el archivo de configuración del contenedor se utiliza para crear imágenes del contenedor. Puede, por ejemplo, especificar las imágenes de base, las asignaciones de directorio, la red y los archivos necesarios antes de que se inicie el contenedor en cuestión. En el caso de los contenedores basados en Docker, se trata de un archivo Docker.
- **Container orchestration:** cuando se despliegan muchos de los contenedores que componen una aplicación, el rastreo, el seguimiento y la gestión, su despliegue y ejecución requieren una gestión y orquestación sofisticadas. Los orquestadores de contenedores asignan un conjunto de servidores (nodos) en un determinado clúster y el respectivo programa para desplegar los contenedores en esos nodos. Estos orquestadores configuran la conexión en red entre los contenedores, el equilibrio de la carga, las actualizaciones continuas, la extensibilidad y otros aspectos. Entre los ejemplos de orquestadores de contenedores más populares figuran los siguientes: Docker Compose/Docker Swarm, Kubernetes, y Mesos/DCOS. Kubernetes es el orquestador de contenedores más popular y existen varias implementaciones basadas en Kubernetes tanto en la nube como en las instalaciones.



Contenedores y orquestación

Algunas definiciones fundamentales de contenedores (3/3)

¿Cuáles son los componentes fundamentales de un motor Docker?

El motor Docker es un componente fundamental en un sistema de eco basado en Docker. Es una aplicación central que se despliega en el sistema operativo y permite alojar los contenedores Docker. Tiene tres componentes principales:

- Dockerd: Un demonio que es un programa de servidor de larga duración.
- REST API: Expone la interfaz para que los programas puedan hablar con este demonio.
- CLI: Nos permite interactuar con Docker usando el comando docker. Básicamente habla con el motor Docker a través de sus APIs de REST.

Ya hemos cubierto los fundamentos de los contenedores. Ahora entendamos los fundamentos del orquestador de contenedores. Elegimos a Kubernetes como nuestro orquestador ya que es el más popular y con más características del mercado.



Contenedores y orquestación

Lo básico de Kubernetes (1/5)

Vamos a cubrir los conceptos básicos de los Kubernetes, también conocidos como **K8s**. Voy a usar el término Kubernetes y K8s indistintamente.

Kubernetes es una plataforma de orquestación de contenedores extensible y de código abierto para: configurar, gestionar y automatizar el despliegue, la ejecución y la orquestación de contenedores.

Fue desarrollada originalmente por Google y de código abierto desde 2014; ahora está mantenida por el CNCF.

Kubernetes en sí es un tema enorme que merece ser tratado de forma independiente y con un libro, un workshop se que. Aquí, se explicará lo básico de algunos de sus componentes más utilizados.



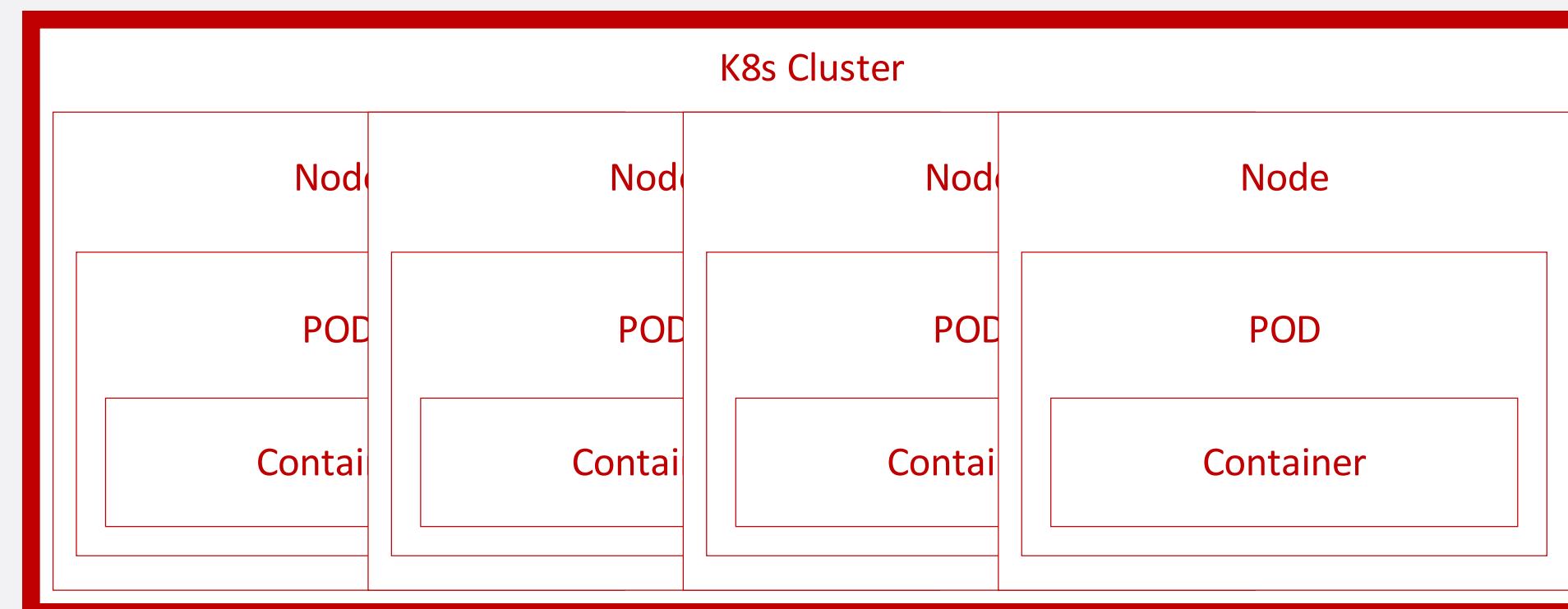
Contenedores y orquestación

Lo básico de Kubernetes (1/5)

Algunos de los componentes de K8s son esenciales y requeridos en la mayoría de las aplicaciones desplegadas en un clúster de K8s.

Vamos a usarlos estos componentes esenciales vitales para cualquier despliegue a través de la aplicación de microservicios.

Este diagrama muestra unos conceptos básicos a un alto nivel de detalle:



Contenedores y orquestación

Lo básico de Kubernetes (2/5)

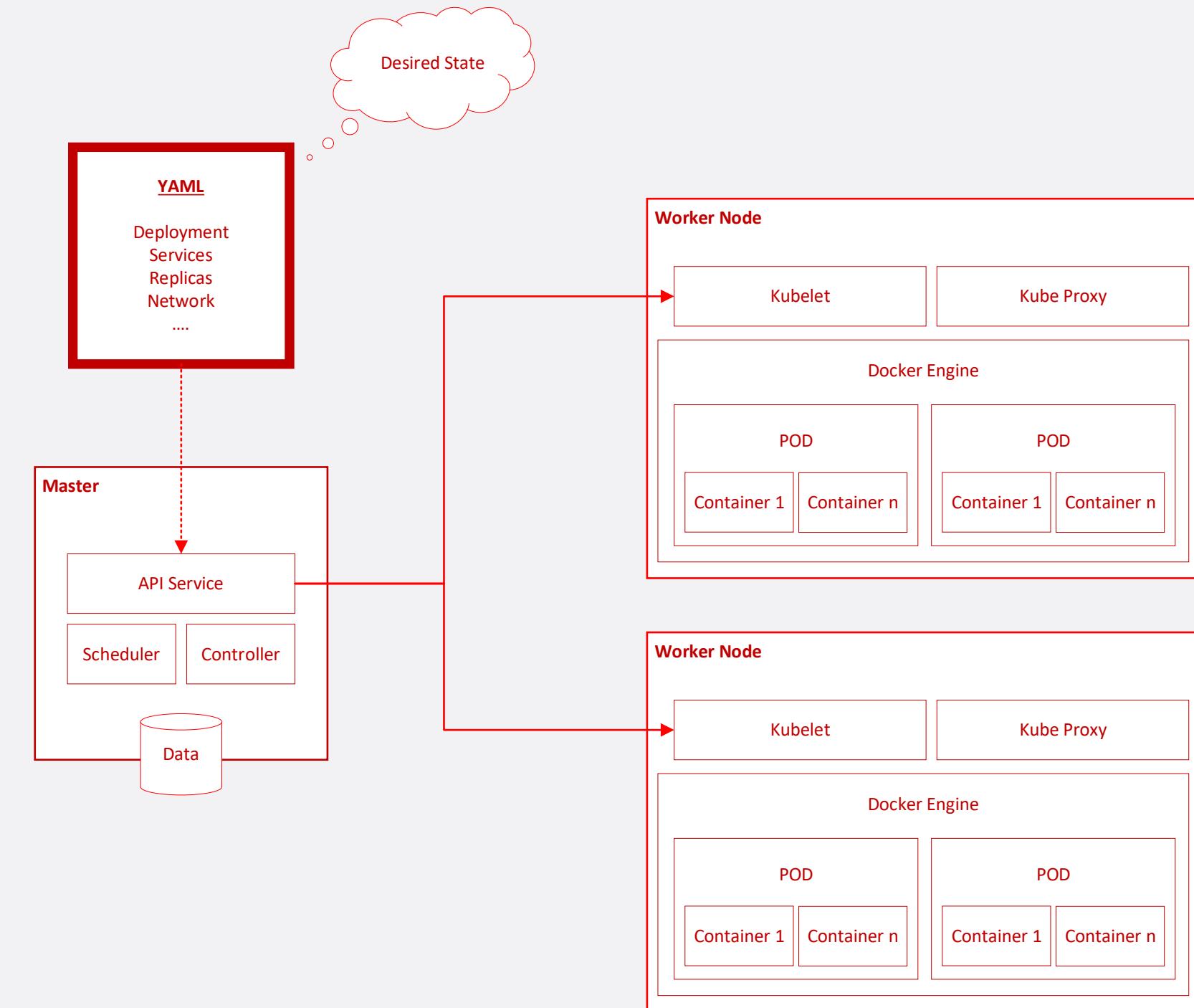
La figura anterior representa un cluster de K8s, donde tenemos nodos (node), pods y contenedores (containers).

- **Node**: especie de máquina o una máquina virtual que proporciona todos los recursos de computación a los componentes subyacentes.
- **Pod**: un componente básico desplegable que se despliega en un determinado grupo de K8s. Tiene un conjunto de contenedores en su interior.
- **Container**: cualquier imagen de un contenedor para nuestra aplicación que queramos desplegar en un cluster K8s se despliega y corre bajo una entidad administrada por K8s que es un Pod.



Contenedores y orquestación

Lo básico de Kubernetes (3/5)



Contenedores y orquestación

Lo básico de Kubernetes (4/5)

El despliegue de una aplicación en un clúster de Kubernetes se realiza mediante un archivo basado en YAML, que básicamente le dice a los K8 que activen y mantengan un estado deseado de los servicios. Es decir, si un servicio en una imagen de un contenedor o un pod o un nodo está caído, K8s automáticamente activará una nueva instancia y mantendrá el estado deseado tal y como lo especificamos con el YAML.

A través de la anterior figura continuamos con las definiciones:

- **Container runtime:** Kubernetes soporta varios containers runtimes para facilitar la ejecución del contenedor de la aplicación. Incluye Docker, containerd, CRI-O, y cualquier implementación Kubernetes Container Runtime Interface (CRI).
- **Master:** Un nodo maestro es un nodo con el que interactuamos y proporcionamos el descriptor de despliegue basado en YAML. Este es un componente necesario para un cluster K8s.
- **API Service:** Como su nombre lo indica, expone la API de Kubernetes.
- **Scheduler:** Es un componente de master que se encarga de asignar los mejores nodos a las pod en un momento dado.

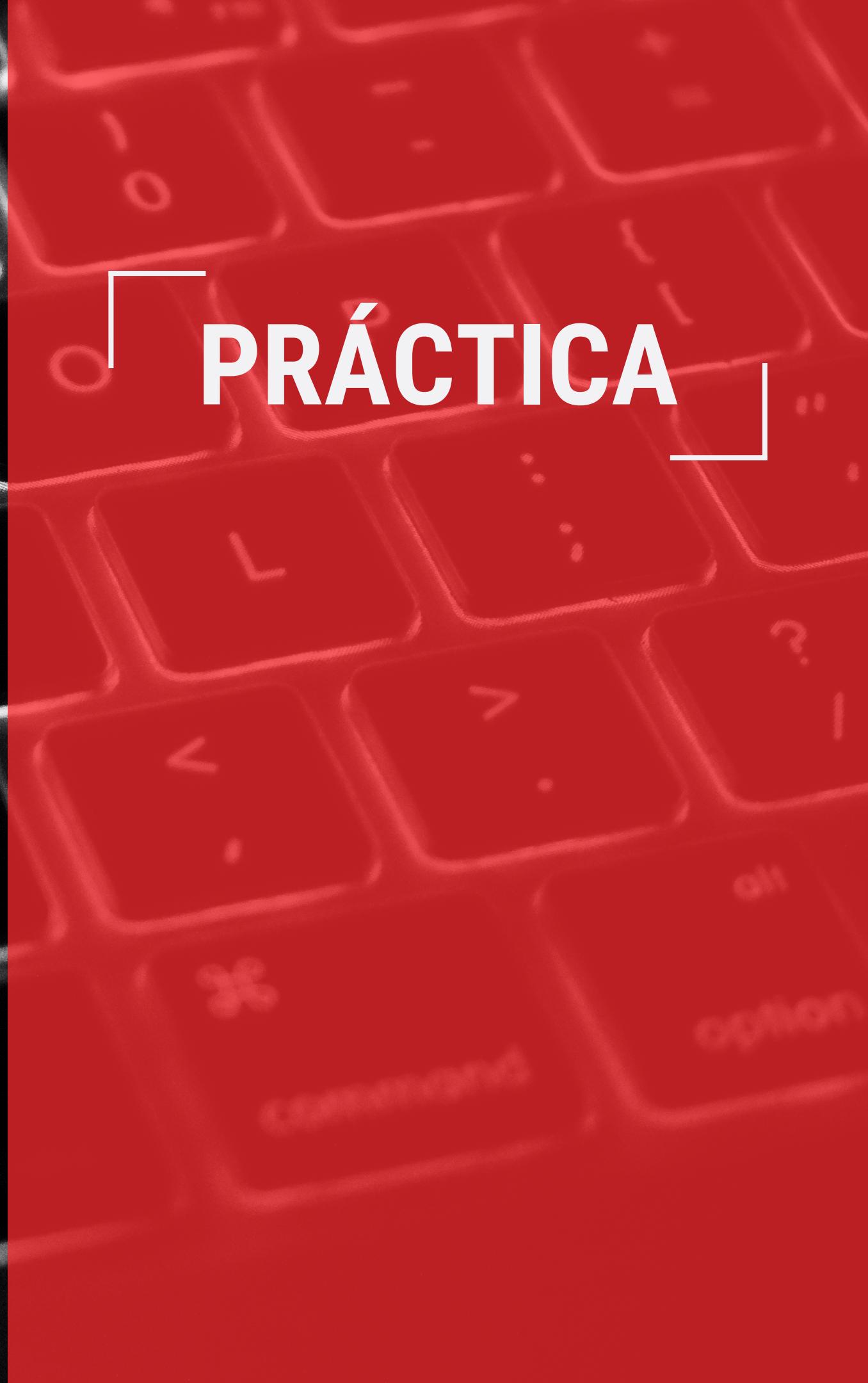


Contenedores y orquestación

Lo básico de Kubernetes (5/5)

- **Controller**: Este componente tiene la mayor parte de la lógica de control de un clúster K8s. Controla los nodos, la replicación, los endpoints y las secrets/tokens.
- **Data Store**: Básicamente es un almacenamiento basado en un etc. que mantiene la información de la configuración.
- **Service**: Permite exponer un servicio que funciona en un pod determinado o pods accesibles fuera de un contenedor como un servicio de red.
- **Kubelet**: Agente que ejecuta cada nodo de un cluster. Asegura que los contenedores están funcionando en sus respectivos pods.
- **Kube proxy**: Es un proxy de red que se ejecuta en cada nodo de un clúster. Implementa parte del servicio de Kubernetes, permitiendo la conexión en red entre varios pods y contenedores, así como permitiendo la accesibilidad fuera del clúster K8s.
- **kubectl**: Por último, pero no menos importante, kubectl es la herramienta de línea de comandos que permite controlar los clusters de Kubernetes mediante el API Service. Utilizaremos kubectl para configurar el clúster de K8s.





Workshop

Introducción

HANDS OF LABS – Manos a la obra

Todos los temas tratados con anterioridad nos han proporcionado una base sólida que necesitamos para construir una aplicación de microservicios en el mundo real y que van a ser útiles de ahora en adelante en nuestra carrera profesional.

¿Cuál es la aplicación?

No estamos construyendo una aplicación del mundo real, aunque tiene componentes que se ajustan a cualquier escenario del mundo real. Estos pequeños componentes son lo suficientemente sencillos para encajar en un workshop y para ayudar a entender los conceptos de construcción de aplicaciones de microservicios usando las últimas herramientas ofrecidas por Microsoft junto con .NET 5.

Nuestra aplicación simplemente trata sobre cálculo de números pares.

Habrá un servicio capaz de “calcular números pares”. Y otros tres servicios clientes que solicitan este servicio dando un número y esperan saber si es par o no. Cada uno de los tres servicios pregunta por un número desde su propio lado, de modo que el servicio principal recibe las solicitudes en varios órdenes (sólo para simular una carga aleatoria). Los clientes enviarán solicitudes una por una continuamente con un retraso de 10 milisegundos entre ellas.

Es una idea básica de la aplicación; en lenguaje profesional, es una especie de caso para una aplicación del mundo real.

Ahora construyamos la arquitectura.



Workshop

Arquitectura de la aplicación (1/5)

Arquitectura de la aplicación.

Cualquier arquitectura de aplicación está incompleta si no satisface todos los requisitos y criterios. También tenemos ciertas limitaciones. En primer lugar, que la aplicación sea lo suficientemente corta como para caber en el contexto de un workshop y lo suficientemente larga como para poder tocar todos los conceptos clave de una aplicación de microservicios en el contexto de la tecnología .NET 5 de Microsoft.

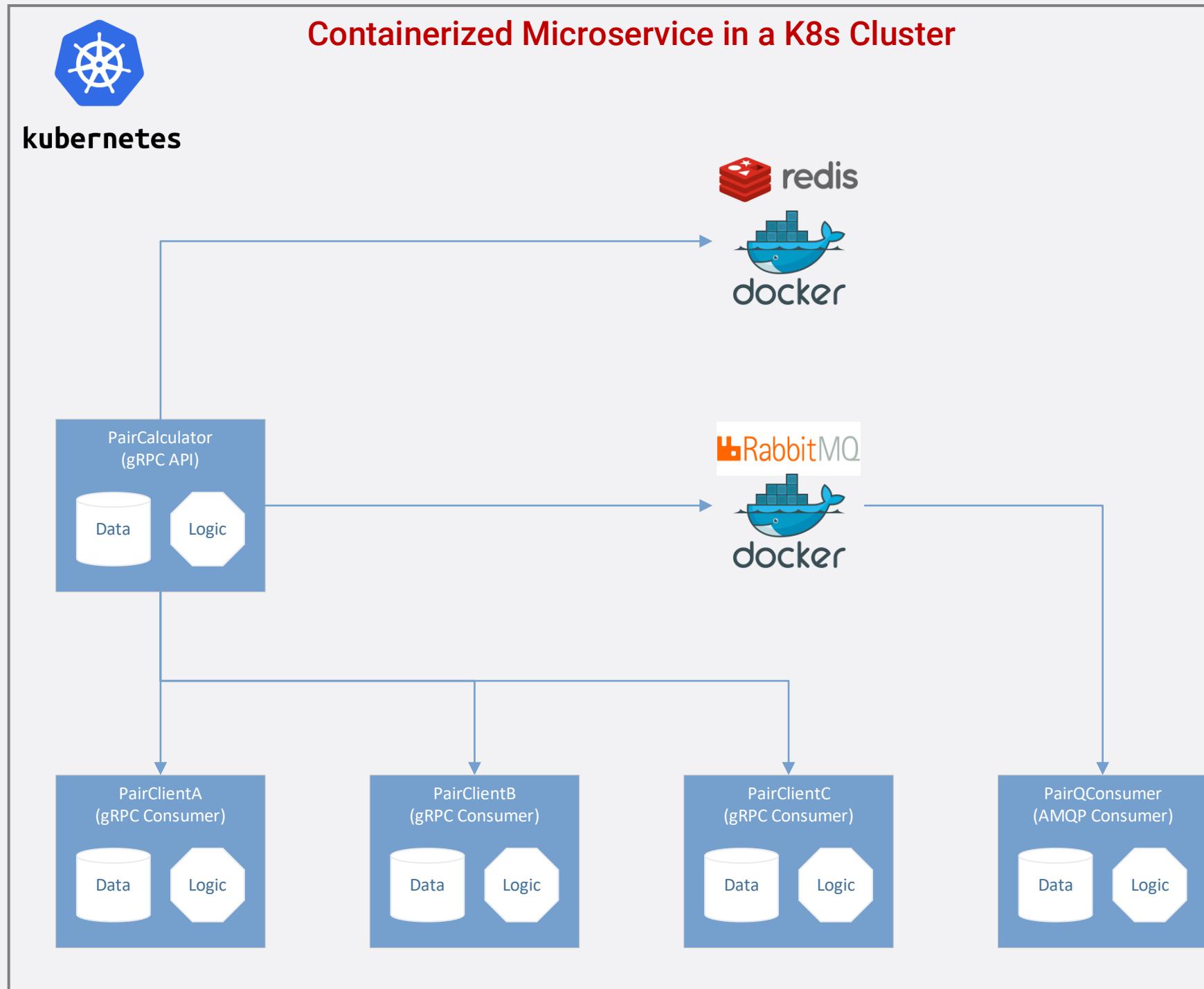
Al construir la arquitectura de esta aplicación, ya tenemos ciertos requisitos conocidos. Un hecho ya conocido es que tiene que ser una arquitectura de aplicación basada en microservicios; otros son que tiene que usar tecnología basada en .NET 5 y que usaremos Tye para construir, depurar y también eventualmente desplegarla en un clúster de Kubernetes y ejecutarla desde allí.

Teniendo en cuenta estos requisitos y restricciones, el siguiente diagrama muestra nuestra arquitectura:



Workshop

Arquitectura de la aplicación (2/5)



Nota:

Los microservicios de nuestra aplicación no tienen datos como tales, pero si tiene que haber algunos datos almacenados en un servicio, esos datos deben residir exclusivamente junto con ese microservicio y no deben ser compartidos como tales.

Para que un microservicio sea completamente independiente, debe llevarse todos sus recursos al terminar, crear o replicarse.

En el diagrama de arquitectura, se pueden ver en total siete microservicios mientras que en nuestra declaración de aplicación, hablamos principalmente de cuatro servicios, donde el primero es principalmente un servicio de servidor que hace el cálculo con números y los otros tres son básicamente sus consumidores.



Workshop

Arquitectura de la aplicación (3/5)

Puntos para tener en cuenta a medida que avanzamos

Todos los servicios están en contenedores usando Docker y se despliegan con la ayuda de Tye. Hay cinco servicios personalizados en la arquitectura; todos ellos están desarrollados usando .NET 5.

Los desplegaremos en un simple clúster de Kubernetes. No crearemos un archivo Dockerfile y archivos de despliegue YAML de Kubernetes para nuestros servicios .NET 5; en su lugar, utilizaremos Tye para hacerlo por nosotros.

Tye no crea el descriptor de despliegue para los servicios/contenedores externos cuando se despliegan al clúster de K8s; por lo tanto, los crearemos y los desplegaremos al clúster nosotros mismos. Cuando no usemos un clúster K8s, todo el trabajo también lo hará Tye usando los contenedores Docker para servicios externos en nuestra máquina, que actúa como un orquestador de contenedores en tiempo de desarrollo.

Vamos a definir ahora los componentes que estamos usando en esta arquitectura.



Workshop

Arquitectura de la aplicación (4/5)

¿Cuáles son los componentes de esta arquitectura?

- **PairCalculator:** Es un microservicio que hace el cálculo para determinar si un número de entrada es Par o no. Por lo tanto, expone un API que se puede llamar a través de otros servicios. Utiliza gRPC.
- **PairClientA:** Este es un microservicio que utiliza el API del microservicio de la PairCalculator para obtener sus respuestas. Ejecuta el código en un bucle infinito empezando por el número 1 como entrada y pide continuamente números. Naturalmente, se comunica usando gRPC.
- **PairClientB** y **PairClientC:** Estos servicios son similares a los PairClientA, excepto que piden números van en una secuencia diferente. Por ejemplo, uno de ellos enviará números múltiplos de 2 (pares) y el otro impares.
- **Redis:** Es un caché bien conocida e implantada. En nuestra aplicación de microservicios de demostración, se utiliza como caché en un contenedor de Redis Docker. PairCalculator y escribe sus resultados en el caché durante 30 minutos; así, si el mismo número es solicitado por cualquier cliente, devuelve los resultados del caché en lugar de calcularlo de nuevo. Se utiliza aquí como en ciertos escenarios similares del mundo real.



Workshop

Arquitectura de la aplicación (5/5)

- **RabbitMQ**: Agente de mensajes. PairCalculator calcula un resultado y si es un número par, pone ese número en una cola de mensajes nombrados como paris. RabbitMQ se utiliza aquí como un contenedor de Docker. Por lo que debes conocer los conceptos de message brokers y queues.
- **PairQConsumer**: Este es un servicio que se suscribe a una cola de mensajes llamada Pars en RabbitMQ. Por lo tanto, sólo recibe todos los números pares generados por la PairCalculator. Una vez más, aquí se utiliza sólo para replicar escenarios del mundo real. También pueden notar que este es el único microservicio asíncrono en nuestra arquitectura y que usa el protocolo AMQP para recibir mensajes de la cola de mensajes.
- **Cluster**: Finalmente, todos los microservicios de nuestra arquitectura están presentes en un solo clúster de Kubernetes.

Nuestra arquitectura está completa, con todos los componentes definidos y con su funcionalidad es clara. Los mecanismos de comunicación también están visibles, así como las tecnologías a utilizar.

Principalmente, vamos a utilizar código basado en .NET 5 con contenedores Docker, utilizando Tye para probar, así como para desplegar en un clúster de Kubernetes.

Y ahora sí, **¡comencemos el desarrollo!**



Workshop

Construyendo la aplicación de microservicios (1/20)

Antes de comenzar el desarrollo hemos establecidos los elementos de infraestructura requeridos.

Asumo que está en Windows 10 y que ya ha instalado Docker Desktop, el SDK de .NET 5 y ha habilitado WSL 2 con Ubuntu. Si no es así es el momento de hacerlo.

También asumo que tiene VS Code o Visual Studio 2019 instalado, puedes usar cualquier otro editor, como Notepad++. Pero me encanta VS Code y VS 2019 y es lo que voy a usar.

Además de lo que ya tenemos instalado en la máquina, necesitaremos configurar algunos componentes más. Vamos a configurarlos.



Workshop

Construyendo la aplicación de microservicios (2/20)

Instalar – Tye

Necesitarás instalar la herramienta Tye. Requiere .NET Core 3.1. Tye puede ser instalada hoy ejecutando el siguiente comando (si no quieres seguir el workshop sin sobre saltos, instala exactamente las versiones indicadas):

```
dotnet tool install -g Microsoft.Tye --version "0.5.0-alpha.20555.1"
```

Instalar – Docker Container Registry

Para el desarrollo y las pruebas locales, vamos a utilizar un registro local de contenedores antes de publicar una versión de nuestros microservicios. Esto acelerará los despliegues y las pruebas en múltiples ocasiones.

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

Instalar – Docker Desktop for Kubernetes

Existen muchas herramientas tales como: Minikube, MicroK8s, K3S, K3D, KinD, ... pero nosotros vamos a usar la opción que nos da Docker Desktop:

Setting >> Kuberenetes >> Enable Kuberenetes

Una vez instalado debemos verificar que funciona, lanza el siguiente comando: kubectl config current-context deberás obtener una salida como esta: docker-desktop



Workshop

Construyendo la aplicación de microservicios (3/20)

Descarga la solución desde:

Entra en VS Code y comienza a editar la solución: code .

Esta solución la podrás usar de base por si te quedas atascado durante la explicación y así poder continuar.

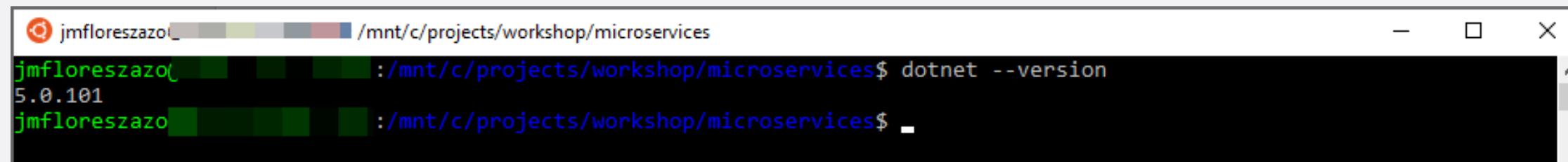
La solución esta separada en ramas, la rama “final” contiene el proyecto completo.



Workshop

Construyendo la aplicación de microservicios (4/20)

Casi comienza el desarrollo. Primero debemos construir la solución y los proyectos. Antes de nada, vamos a ver que tenemos el entorno correcto:



A screenshot of a terminal window titled 'jmfloreszazo_'. The window shows the command 'dotnet --version' being run in the directory '/mnt/c/projects/workshop/microservices'. The output of the command '5.0.101' is displayed. The terminal has a dark background with light-colored text.

```
jmfloreszazo_ /mnt/c/projects/workshop/microservices
jmfloreszazo_ :/mnt/c/projects/workshop/microservices$ dotnet --version
5.0.101
jmfloreszazo_ :/mnt/c/projects/workshop/microservices$ -
```

Y seguimos los siguientes pasos:

1. dotnet new grpc -n paircalculator
2. dotnet new sln -n microservicesapp
3. dotnet sln microservicesapp.sln add paircalculator/paircalculator.csproj



Workshop

Construyendo la aplicación de microservicios (5/20)

Entra en el proyecto PairCalculator, y busca el fichero con extensión proto, cambia el nombre a pair.proto y escribe el siguiente código:

```
src > PairCalculator > Protos > pair.proto
1 syntax = "proto3";
2
3 option csharp_namespace = "PairCalculator";
4
5 package pair;
6
7 // The service definition.
8 service PairCalculator {
9 | rpc IsItPair (PairRequest) returns (PairReply);
10 }
11
12 // The request message
13 message PairRequest {
14 | int64 number = 1;
15 }
16
17 // The response message
18 message PairReply {
19 | bool isPair = 1;
20 }
21
```

El archivo proto nos dice que se trata de una operación gRPC unaria en la que el cliente llama al servidor con el método IsItPair pasando una solicitud que contiene un número y espera una respuesta como valor booleano para confirmar si el número de entrada era un número par o no. El valor csharp_namespace le dice al generador de código que use este nombre como el espacio de nombres para el código C#.



Workshop

Construyendo la aplicación de microservicios (6/20)

Hora toca implementar el método en C# IsItPair. Apóyate en las fuentes que te has descargado para completar el proyecto.

Tras esta codificación vamos a trabajar sobre el proyecto PairClientA. Ejecuta los siguientes comandos:

```
dotnet new worker -n pairclienta
```

```
dotnet sln microservicesapp.sln add pairclienta/pairclienta.csproj
```

```
dotnet add pairclienta/pairclienta.csproj package Grpc.Net.Client
```

```
dotnet add pairclienta/pairclienta.csproj package Grpc.Net.ClientFactory
```

```
dotnet add pairclienta/pairclienta.csproj package Google.Protobuf
```

```
dotnet add pairclienta/pairclienta.csproj package Grpc.Tools
```

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



Workshop

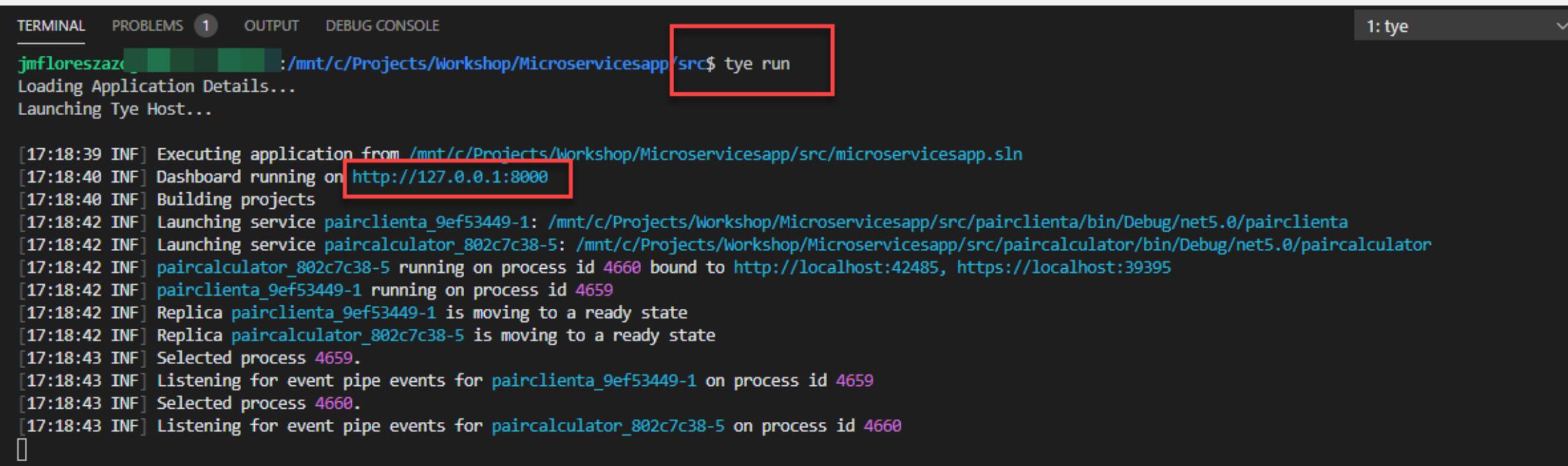
Construyendo la aplicación de microservicios (7/20)

Si estás usando VS 2019, puedes añadir un Connected Service de tipo gRPC, directamente auto genera code behind en C#.

Tras este inciso entramos en Program.cs y añadimos las siguientes líneas de código en ConfigureServices:

```
        services.AddGrpcClient<PrimeCalculator.PrimeCalculatorClient>(o =>
{
    //primecalculator will be inject to configuration via Tye
    o.Address = configurationFromHostBuilderContext.GetServiceUri("primecalculator") ?? MANUAL_DEBUGTIME_URI;
});
```

Ya tenemos el cliente gRPC listo y ahora podrás ejecutar Tye:



The screenshot shows a terminal window with the following output:

```
TERMINAL PROBLEMS 1 OUTPUT DEBUG CONSOLE
jmfloreszazc_ [~] :/mnt/c/Projects/Workshop/Microservicesapp/src$ tye run
Loading Application Details...
Launching Tye Host...

[17:18:39 INF] Executing application from /mnt/c/Projects/Workshop/Microservicesapp/src/microservicesapp.sln
[17:18:40 INF] Dashboard running on http://127.0.0.1:8000
[17:18:40 INF] Building projects
[17:18:42 INF] Launching service pairclienta_9ef53449-1: /mnt/c/Projects/Workshop/Microservicesapp/src/pairclienta/bin/Debug/net5.0/pairclienta
[17:18:42 INF] Launching service paircalculator_802c7c38-5: /mnt/c/Projects/Workshop/Microservicesapp/src/paircalculator/bin/Debug/net5.0/paircalculator
[17:18:42 INF] paircalculator_802c7c38-5 running on process id 4660 bound to http://localhost:42485, https://localhost:39395
[17:18:42 INF] pairclienta_9ef53449-1 running on process id 4659
[17:18:42 INF] Replica pairclienta_9ef53449-1 is moving to a ready state
[17:18:42 INF] Replica paircalculator_802c7c38-5 is moving to a ready state
[17:18:43 INF] Selected process 4659.
[17:18:43 INF] Listening for event pipe events for pairclienta_9ef53449-1 on process id 4659
[17:18:43 INF] Selected process 4660.
[17:18:43 INF] Listening for event pipe events for paircalculator_802c7c38-5 on process id 4660
```



Workshop

Construyendo la aplicación de microservicios (8/20)

Si entras en la URL que he marcado:

The screenshot shows the Tye Dashboard interface. At the top, there's a header with a gear icon, the text "Tye Dashboard", and buttons for "Tell us what you think!" and "About". Below the header, there's a breadcrumb navigation bar with "Home". The main area is titled "Services" and contains a table with two rows. The columns are labeled "Name", "Type", "Source", "Bindings", "Replicas", "Restarts", and "Logs". The first row represents the "paircalculator" service, which is a "Project" type with source code at "/mnt/c/Projects/Workshop/Microservicesapp/src/paircalculator/paircalculator.csproj". It has bindings to "http://localhost:42485" and "https://localhost:39395", 1 replica, 0 restarts, and a "View" link for logs. The second row represents the "pairclienta" service, also a "Project" type with source code at "/mnt/c/Projects/Workshop/Microservicesapp/src/pairclienta/pairclienta.csproj". It has no bindings, 1 replica, 0 restarts, and a "View" link for logs.

Name	Type	Source	Bindings	Replicas	Restarts	Logs
paircalculator	Project	/mnt/c/Projects/Workshop/Microservicesapp/src/paircalculator/paircalculator.csproj	http://localhost:42485https://localhost:39395	1/1	0	View
pairclienta	Project	/mnt/c/Projects/Workshop/Microservicesapp/src/pairclienta/pairclienta.csproj	none	1/1	0	View

Podrás observar el funcionamiento de Tye y ver los logs correspondientes (independientemente del código que he puesto para ver si es par o impar, más adelante lo arreglaremos);

The screenshot shows the "Logs for pairclienta" section of the Tye dashboard. There is a "Clear Log" button at the top left. The log output is displayed in a black box with white text. The log entries show the flow of an HTTP request from the client to the service and back again. The log includes timestamps, service names, and informational messages about the processing of the request.

```
[pairclienta_9ef53449-1]: info: System.Net.Http.HttpClient.PairCalculatorClient.LogicalHandler[100]
[pairclienta_9ef53449-1]: Start processing HTTP request POST http://localhost:42485/pair.PairCalculator/IsItPair
[pairclienta_9ef53449-1]: info: System.Net.Http.HttpClient.PairCalculatorClient.ClientHandler[100]
[pairclienta_9ef53449-1]: Sending HTTP request POST http://localhost:42485/pair.PairCalculator/IsItPair
[pairclienta_9ef53449-1]: info: System.Net.Http.HttpClient.PairCalculatorClient.ClientHandler[101]
[pairclienta_9ef53449-1]: Received HTTP response after 2.4181ms - OK
[pairclienta_9ef53449-1]: info: System.Net.Http.HttpClient.PairCalculatorClient.LogicalHandler[101]
[pairclienta_9ef53449-1]: End processing HTTP request after 2.8707ms - OK
[pairclienta_9ef53449-1]: info: pairclienta.Worker[0]
[pairclienta_9ef53449-1]: Is 14387 a Pair number? Service tells us: True
[pairclienta_9ef53449-1]: info: System.Net.Http.HttpClient.PairCalculatorClient.LogicalHandler[100]
[pairclienta_9ef53449-1]: Start processing HTTP request POST http://localhost:42485/pair.PairCalculator/IsItPair
```



Workshop

Construyendo la aplicación de microservicios (9/20)

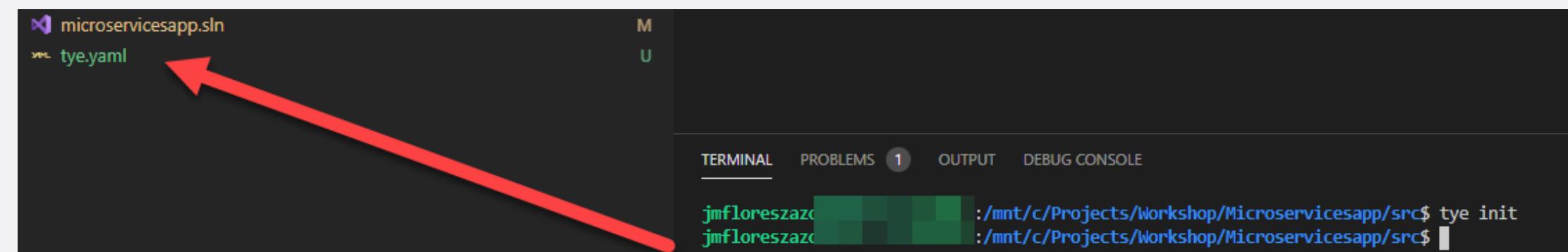
Para parar el proceso, nada más sencillo que Ctrl+C y el servicio de Tye deja de ejecutarse.

Ya hemos creado un servicio gRPC de forma muy sencilla, ejecutado via Tye y observado los logs de ambos procesos.

De momento vamos a refactorizar el código para que calcule si el valor es par o impar y continuamos con el siguiente paso, en la rama “Step2”.

Ahora toca el momento de introducir Redis Cache para ello vamos a seguir estos pasos:

Necesitamos **Redis** como un contenedor, y para ellos usaremos Tye. Vamos a generar un fichero de descripción de Tye en formato YAML, mediante el comando tye init. Este comando nos creará tye.yaml (revisar su contenido inicial):



Workshop

Construyendo la aplicación de microservicios (10/20)

Al contenido inicial debemos añadir ahora nuestra nueva opción de Redis:

```
*~ tye.yaml > [ ] services > {} 1 > ↵ args
 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: microservicesapp
 8 services:
 9 - name: paircalculator
10   project: paircalculator/paircalculator.csproj
11 - name: pairclienta
12   project: pairclienta/pairclienta.csproj
13 services:
14 - name: redis
15   image: redis
16   bindings:
17     - port: 6379
18       connectionString: ${host}:${port}
19     - name: redis-cli
20       image: redis
21       args: "redis-cli -h redis MONITOR"
```

Por supuesto, esto no me lo invento yo, si no que me ayudo del proyecto:

<https://github.com/dotnet/tye/blob/master/samples/redis/tye.yaml>



Workshop

Construyendo la aplicación de microservicios (11/20)

Vemos que tenemos un redis-cli, se usa únicamente para logs, es un componente opcional que podemos quitarlo si lo deseamos.

Vamos a crear el proyecto de Redis:

1. dotnet add paircalculator/paircalculator.csproj package Microsoft.Extensions.Configuration
2. dotnet add paircalculator/paircalculator.csproj package Microsoft.Extensions.Caching.StackExchangeRedis

Vamos a poner el código necesario para implementar la cache en el startup.cs y en el servicio. Puedes revisar como implementar la cache desde el ejercicio que llevamos.

Y lo ejecutamos:

```
jmfloreszazo@DESKTOP-1D9C9E8:~$ cd /mnt/c/Projects/Workshop/Microservicesapp/src$ tye run --debug primecalculator primecliente
Loading Application Details...
Launching Tye Host...

[18:34:45 INF] Executing application from /mnt/c/Projects/Workshop/Microservicesapp/src/tye.yaml
[18:34:45 INF] Dashboard running on http://127.0.0.1:8000
[18:34:46 INF] Docker image redis already installed
[18:34:46 INF] Creating docker network tye_network_6cha7343-0
```



Workshop

Construyendo la aplicación de microservicios (12/20)

Vamos al punto 3 (step 3 de nuestras ramas). En la que vamos a meter una cola RabbitMQ en la aplicación, y poco a poco estamos implementando nuestra arquitectura.

Para ello vamos a añadir a nuestro fichero YAML el componente y la interface MUI:

```
- name: rabbitmq
  image: rabbitmq:3-management
  bindings:
    - name: mq_binding
      port: 5672
      protocol: rabbitmq
    - name: mui_binding
      port: 15672
```

MUI será accesible desde <https://localhost:15672> y usando las credenciales guest/guest.

Una vez más debemos añadir las referencias:

```
dotnet add paircalculator/paircalculator.csproj package RabbitMQ.Client
```

Introducir el código (ver rama “Step3”) y ya podemos usar los mensajes en nuestro proyecto.



Workshop

Construyendo la aplicación de microservicios (13/20)

Gracias a nuestro ya conocido comando `tye run`, podemos ver nuestros servicios corriendo y además acceder a MUI para gestionar RabbitMQ.

RabbitMQ™ RabbitMQ 3.8.9 Erlang 23.2.1 Refreshed 2021-01-10 17:58:50 Refresh every 5 seconds Virtual host All Cluster rabbit@3cc207768ec9 User guest Log out

Overview Connections Channels Exchanges Queues Admin

Overview

Totals

Queued messages last minute ?

Ready: 1,727 Unacked: 0 Total: 1,727

Message rates last minute ?

Publish: 32/s Publisher confirm: 0.00/s Unroutable (drop): 0.00/s Disk read: 0.00/s Unroutable (return): 0.00/s Disk write: 0.00/s

Global counts ?

Connections: 1 Channels: 1 Exchanges: 7 Queues: 1 Consumers: 0

Nodes

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats	+/-
rabbit@595cc9581928	98 1048576 available	1 943629 available	575 1048576 available	113 MiB 4.9 GiB high watermark	233 GiB 48 MiB low watermark	1m 32s	basic disc 2 rss	This node All nodes	+/-

Churn statistics
Ports and contexts
Export definitions
Import definitions

HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

Workshop

Construyendo la aplicación de microservicios (14/20)

Lo que estamos haciendo es que cada numero par encontrado lo enviamos a través de la cola pair. De esta manera, la cola de RabbitMQ se queda con todos los pares calculados. Esto nos sirve para ver el correcto funcionamiento de nuestro proceso. Así de simple, se trata de aprender con cosas sencillas, añadir más complejidad es perdernos en puntos que no interesan.

Podrás observar que los mensajes son un flujo asíncrono, por tanto, nuestra cola esta desacoplada y desconectada de los demás microservicios pero conectada con un message broker.

Hasta ahora hemos creado una caché y una cola y ejecutado con éxito todos los servicios declarados en el YAML usando Tye, además de comprobar que MUI también se ejecuta correctamente.

Lo que nos falta es añadir dos servicios más que envían datos para calcular los pares y un lector de mensajes.

Vamos a ello.



Workshop

Construyendo la aplicación de microservicios (15/20)

Recapitulamos: lo que estamos haciendo es que cada numero par encontrado lo enviamos a través de la cola pair. De esta manera, la cola de RabbitMQ se queda con todos los pares calculados. Esto nos sirve para ver el correcto funcionamiento de nuestro proceso. Así de simple, se trata de aprender con cosas sencillas, añadir más complejidad es perdernos en puntos que no interesan.

Podrás observar que los mensajes son un flujo asíncrono, por tanto, nuestra cola esta desacoplada y desconectada de los demás microservicios, pero conectada con un message broker.

Hasta ahora hemos creado una caché, una cola y ejecutado con éxito todos los servicios declarados en el YAML usando Tye; además de comprobar que MUI también se ejecuta correctamente y podemos ver su interface.

Lo que nos falta es añadir dos servicios más que envían datos para calcular los pares y un lector de mensajes de la cola de RabbitMQ.

Vamos a ello. Aun nos queda tarea antes de llegar a otro punto más interesante.



Workshop

Construyendo la aplicación de microservicios (15/20)

Añadimos os proyectos similares a pairclienta; los llamaremos pairclientb y pairclienc. Uno enviará exclusivamente valores pares y el otro impares.

Además de los servicios debemos agregarlos a nuestro fichero YAML de Tye.

Vamos a la rama “Step4”.

Y vamos a introducir el concepto de replica:

```
name: microservicesapp
services:
  - name: paircalculator
    project: paircalculator/paircalculator.csproj
    replicas: 2
  - name: pairclienta
    project: pairclienta/pairclienta.csproj
    replicas: 4
  - name: redis
```

Es lo mismo que se usa en K8s, es decir, vamos a mantener un número de Pods ejecutándose en todo momento, de esta forma garantizamos la disponibilidad de un número idénticos de Pods. Esto que osuento es en resumen un ReplicaSet, que también admite Tye.



Workshop

Construyendo la aplicación de microservicios (16/20)

Crear tanto el proyecto pairclientb, como pairclientc, como copia de pairclientb y añadir los proyectos a la solución. Nada que no sepamos realizar ya. Y recordar crear la entrada de los proyectos en el YAML.

Services							
Name	Type	Source	Bindings	Replicas	Restarts	Logs	
paircalculator	Project	/mnt/c/Projects/Workshop/Microservicesapp/src/paircalculator/paircalculator.csproj	http://localhost:40895 https://localhost:44105	2/2	0	View	
pairclienta	Project	/mnt/c/Projects/Workshop/Microservicesapp/src/pairclienta/pairclienta.csproj	none	4/4	0	View	
pairclientb	Project	/mnt/c/Projects/Workshop/Microservicesapp/src/pairclientb/pairclientb.csproj	none	4/4	0	View	
pairclientc	Project	/mnt/c/Projects/Workshop/Microservicesapp/src/pairclientc/pairclientc.csproj	none	4/4	0	View	

Tras esta pequeña modificación vamos a crear el ultimo de ellos, el consumidor de las colas que llamaremos pairqconsumer.



Workshop

Construyendo la aplicación de microservicios (17/20)

Desde la línea de comandos vamos a ejecutar:

1. dotnet new worker -n pairqconsumer
2. dotnet add pairqconsumer/pairqconsumer.csproj package RabbitMQ.Client
3. dotnet add pairqconsumer/pairqconsumer.csproj package --prerelease Microsoft.Tye.Extensions.Configuration

Añadimos el código necesario para consumir la cola, el proyecto el fichero YAML y podemos observar su consumo:



The screenshot shows the Tye Dashboard interface. At the top, there's a dark header bar with the Tye logo, a "Tye Dashboard" button, and two blue buttons: "Tell us what you think!" and "About". Below the header, a navigation bar has a "Home" button (which is highlighted in blue) and a "Logs for pairqconsumer" button. Underneath the navigation bar is a "Clear Log" button. The main area is a large black text box containing log entries from the "pairqconsumer" service. The log entries are as follows:

```
[pairqconsumer_d9bcf7a2-c]: Received a new calculated pair number: 100866
[pairqconsumer_d9bcf7a2-c]: info: pairqconsumer.Worker[0]
[pairqconsumer_d9bcf7a2-c]: Received a new calculated pair number: 526
[pairqconsumer_d9bcf7a2-c]: info: pairqconsumer.Worker[0]
[pairqconsumer_d9bcf7a2-c]: Received a new calculated pair number: 100868
[pairqconsumer_d9bcf7a2-c]: info: pairqconsumer.Worker[0]
[pairqconsumer_d9bcf7a2-c]: Received a new calculated pair number: 100868
[pairqconsumer_d9bcf7a2-c]: info: pairqconsumer.Worker[0]
[pairqconsumer_d9bcf7a2-c]: Received a new calculated pair number: 100870
```



Workshop

Construyendo la aplicación de microservicios (18/20)

Dentro del fichero YAML podemos establecer tags en cada name, para poder usar características del comando tye. Ver rama “Step5”.

He añadido “bs” como tag que indica “base service”, “cl” como “client” y “mw” como “middleware”. De esta forma podemos si lo deseamos ejecutar tye de la siguiente forma:

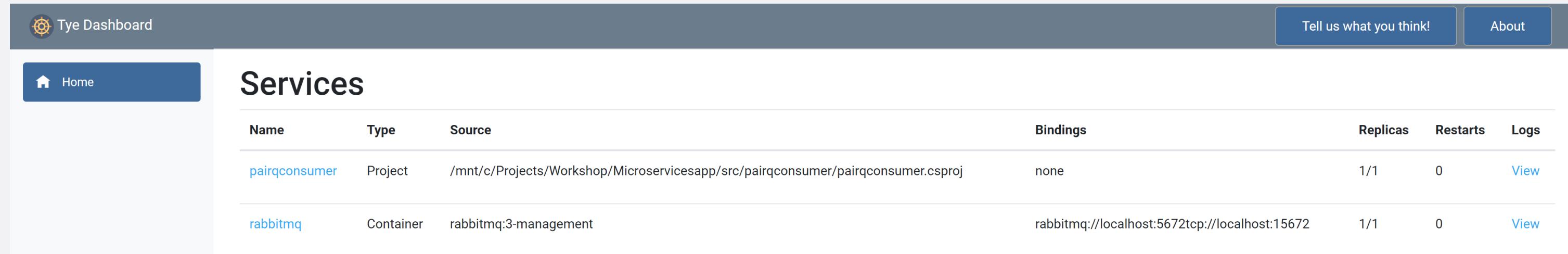
```
jmfloreszazo@DESKTOP-1QHJF0A:~/mnt/c/Projects/Workshop/Microservicesapp/src$ tye run --tags mw --debug primeqconsumer
Loading Application Details...
Launching Tye Host...

[22:03:31 INF] Executing application from /mnt/c/Projects/Workshop/Microservicesapp/src/tye.yaml
[22:03:32 INF] Dashboard running on http://127.0.0.1:8000
[22:03:32 INF] Docker image rabbitmq:3-management already installed
[22:03:32 INF] Running image rabbitmq:3-management for rabbitmq_161b3f83-5
[22:03:32 INF] Building projects
[22:03:33 INF] Running container rabbitmq_161b3f83-5 with ID f5e21ed91248
[22:03:33 INF] Replica rabbitmq_161b3f83-5 is moving to a ready state
[22:03:33 INF] Collecting docker logs for rabbitmq_161b3f83-5.
[22:03:35 INF] Launching service pairqconsumer_df686bc7-5: /mnt/c/Projects/Workshop/Microservicesapp/src/pairqconsumer/bin/Debug/net5.0/pairqconsumer
[22:03:35 INF] pairqconsumer_df686bc7-5 running on process id 31292
[22:03:36 INF] Replica pairqconsumer_df686bc7-5 is moving to a ready state
[22:03:36 INF] Selected process 31292.
[22:03:36 INF] Listening for event pipe events for pairqconsumer_df686bc7-5 on process id 31292
```



Workshop

Construyendo la aplicación de microservicios (19/20)



The screenshot shows the Tye Dashboard interface. At the top, there's a navigation bar with 'Tye Dashboard' on the left, 'Tell us what you think!' in the center, and 'About' on the right. Below the navigation bar, there's a breadcrumb menu with 'Home' selected. The main area is titled 'Services' and contains a table with two rows. The columns are labeled 'Name', 'Type', 'Source', 'Bindings', 'Replicas', 'Restarts', and 'Logs'. The first row represents a 'pairqconsumer' project, which is a 'Project' type with a source path of '/mnt/c/Projects/Workshop/Microservicesapp/src/pairqconsumer/pairqconsumer.csproj'. It has no bindings, 1/1 replicas, 0 restarts, and a 'View' link for logs. The second row represents a 'rabbitmq' container, which is a 'Container' type with a source path of 'rabbitmq:3-management'. It has bindings to 'rabbitmq://localhost:5672tcp://localhost:15672', 1/1 replicas, 0 restarts, and a 'View' link for logs.

Name	Type	Source	Bindings	Replicas	Restarts	Logs
pairqconsumer	Project	/mnt/c/Projects/Workshop/Microservicesapp/src/pairqconsumer/pairqconsumer.csproj	none	1/1	0	View
rabbitmq	Container	rabbitmq:3-management	rabbitmq://localhost:5672tcp://localhost:15672	1/1	0	View

Y ejecutar las partes de nuestros microservicios que deseemos. Usar tags es una buena táctica de depuración.

Puedes ampliar información en el proyecto:

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



Workshop

Construyendo la aplicación de microservicios (20/20)

Y tal y como ya conoces ejecutar `tye run`, ejecutará todos los servicios para comprobar la ejecución y logs de los mismos ya sea con el panel de tye o mediante los registros del contenedor Docker cuando se ejecuta desde el cluster K8s.

Tye Dashboard

Tell us what you think!

About

Home

Services

Name	Type	Source	Bindings	Replicas	Restarts	Logs
paircalculator	Project	/mnt/c/Projects/Workshop/Microservicesapp/src/paircalculator/paircalculator.csproj	http://localhost:45459 https://localhost:41807	2/2	0	View
pairclienta	Project	/mnt/c/Projects/Workshop/Microservicesapp/src/pairclienta/pairclienta.csproj	none	4/4	0	View
pairclientb	Project	/mnt/c/Projects/Workshop/Microservicesapp/src/pairclientb/pairclientb.csproj	none	4/4	0	View
pairclientc	Project	/mnt/c/Projects/Workshop/Microservicesapp/src/pairclientc/pairclientc.csproj	none	4/4	0	View
pairqconsumer	Project	/mnt/c/Projects/Workshop/Microservicesapp/src/pairqconsumer/pairqconsumer.csproj	none	1/1	0	View
redis	Container	redis	tcp://localhost:6379	1/1	0	View
redis-cli	Container	redis	none	1/1	0	View
rabbitmq	Container	rabbitmq:3-management	rabbitmq://localhost:5672tcp://localhost:15672	1/1	0	View



Workshop

Desplegar microservicios al cluster Kubernetes (1/10)

Con anterioridad hemos hablado de la infraestructura necesaria y hemos configurado un registro de contenedores local.

El registro de contenedores es necesario cuando implementamos un cluster de K8s. Por tanto, es ahora cuando vamos a implementar nuestros microservicios en el cluster K8s.

Recuerda que tenemos que tener habilitado el cluster K8s de Docker y su nombre es docker-desktop:

```
jmfloreszazc@: /mnt/c/Projects/Workshop/Microservicesapp/src$ kubectl config current-context  
docker-desktop
```

Ese será nuestro contexto para kubectl.

Una vez verificado, ya podemos implementar las cargas de trabajo en K8s. Vamos a usar tye para implementar los servicios. Tye admite el lanzamiento de contenedores externos como Redis ejecutando una especie de orquestador en tiempo de desarrollo; no implementa esos servicios externos automáticamente en K8s.

Para nuestro ejemplo tanto Redis como RabbitMQ, los vamos a implementar nosotros, para ello vamos a crear un archivo de implementación de K8s. Por tanto, necesitamos dos archivos de implementación YAML: redis.yaml y rabbitmq.yaml, los podéis ver a continuación.

Lo tendremos todo en la ultima rama, llamada “final”.



Workshop

Desplegar microservicios al cluster Kubernetes (2/10)

redis.yaml

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: redis
5    labels:
6      app.kubernetes.io/name: redis
7      app.kubernetes.io/part-of: microservicesapp
8  spec:
9    selector:
10   matchLabels:
11     app.kubernetes.io/name: redis
12   replicas: 1
13   template:
14     metadata:
15       labels:
16         app.kubernetes.io/name: redis
17         app.kubernetes.io/part-of: microservicesapp
18     spec:
19       containers:
20         - name: redis
21           image: redis
22           resources:
23             requests:
24               cpu: 100m
25               memory: 100Mi
26             ports:
27               - containerPort: 6379
29   ---
30   apiVersion: v1
31   kind: Service
32   metadata:
33     name: redis
34     labels:
35       app.kubernetes.io/name: redis
36       app.kubernetes.io/part-of: microservicesapp
37   spec:
38     ports:
39       - port: 6379
40         targetPort: 6379
41     selector:
42       app.kubernetes.io/name: redis
43   
```



Workshop

Desplegar microservicios al cluster Kubernetes (3/10)

rabbitmq.yaml

```
29
30  ---
31  apiVersion: v1
32  kind: Service
33  metadata:
34    name: rabbitmq
35    labels:
36      app.kubernetes.io/name: rabbitmq
37      app.kubernetes.io/part-of: microservicesapp
38  spec:
39    ports:
40      - port: 5672
41        protocol: TCP
42        targetPort: 5672
43    selector:
44      app.kubernetes.io/name: rabbitmq
45  ---
46  apiVersion: v1
47  kind: Service
48  metadata:
49    name: rabbitmq-mui
50    labels:
51      app.kubernetes.io/name: rabbitmq
52      app.kubernetes.io/part-of: microservicesapp
53  spec:
54    type: NodePort
55    ports:
56      - port: 15672
```

```
29
30  ---
31  apiVersion: v1
32  kind: Service
33  metadata:
34    name: rabbitmq
35    labels:
36      app.kubernetes.io/name: rabbitmq
37      app.kubernetes.io/part-of: microservicesapp
38  spec:
39    ports:
40      - port: 5672
41        protocol: TCP
42        targetPort: 5672
43    selector:
44      app.kubernetes.io/name: rabbitmq
45  ---
46  apiVersion: v1
47  kind: Service
48  metadata:
49    name: rabbitmq-mui
50    labels:
51      app.kubernetes.io/name: rabbitmq
52      app.kubernetes.io/part-of: microservicesapp
53  spec:
54    type: NodePort
55    ports:
56      - port: 15672
```

```
57    protocol: TCP
58    targetPort: 15672
59    nodePort: 30072
60  selector:
61    app.kubernetes.io/name: rabbitmq
62
```



Workshop

Desplegar microservicios al cluster Kubernetes (4/10)

Lo más importante de estos archivos YAML es que le dicen a Kuberentes que implementen estos pods con el nombre que hemos dado y use la imagen del contenedor dentro del pod. Además, expone la interface de comunicación, como la URL HTTO de un pod a otros pods, utilizando el valor de la etiqueta kind del archivo YAML.

Una vez revisado esto, vamos a ejecutar los archivos YAML:

```
jmfloreszazo@DESKTOP-1D9C9E8:~/mnt/c/Projects/Workshop/Microservicesapp/src$ kubectl apply -f rabbitmq.yaml
deployment.apps/rabbitmq created
service/rabbitmq created
service/rabbitmq-mui created
```

```
jmfloreszazo@DESKTOP-1D9C9E8:~/mnt/c/Projects/Workshop/Microservicesapp/src$ kubectl apply -f redis.yaml
deployment.apps/redis created
service/redis created
```

Con estas acciones hemos desplegado las imágenes instantáneamente y entrar en funcionamiento unos segundos (RabbitMQ tarda de 30 segundos a un minuto en estar listo para su uso). En general muchos servicios tardan algo de tiempo: han de levantarse y calentarse.



Workshop

Desplegar microservicios al cluster Kubernetes (4/10)

También puedes observar que en rabbitmq.yaml hemos puesto el puerto adicional 30072, con el puerto interno 15672 con un tipo de NodePort. Básicamente esta exponiendo MUI de RabbitMQ al exterior del cluster K8s. De esta forma podremos acceder a MUI en nuestra máquina desde <http://localhost:30072> y ver el flujo de mensajes que veíamos con tye run.

Desplegar nuestros microservicios desarrollados en .NET5 al cluster tye, es algo trivial. Ejecuta el siguiente comando:

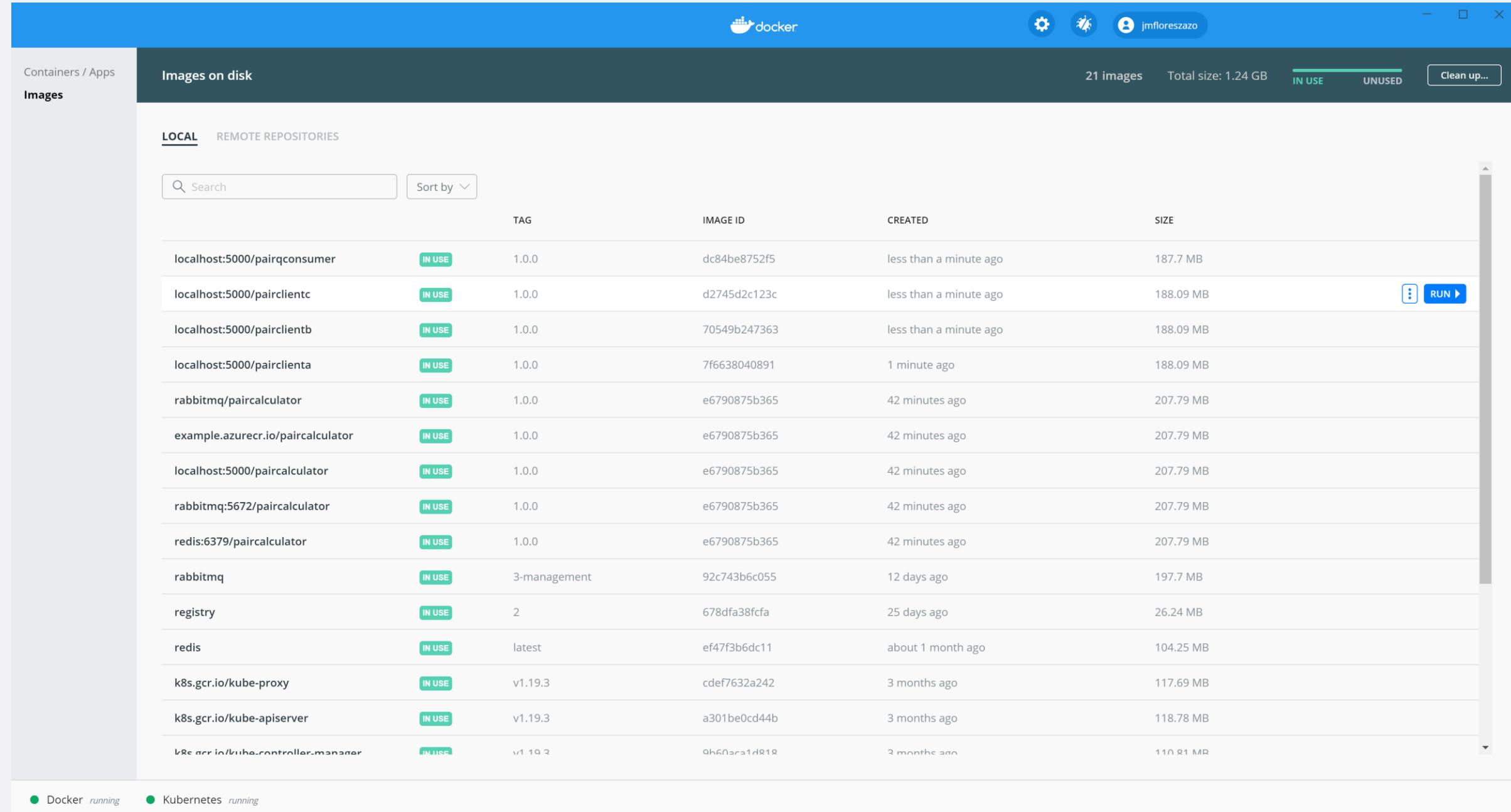
```
jmfloreszazo@DESKTOP-GSA2HFS:/mnt/c/Projects/Workshop/Microservicesapp/src$ tye deploy --interactive
Loading Application Details...
Verifying kubectl installation...
Verifying kubectl connection to cluster...
Enter the Container Registry (ex: 'example.azurecr.io' for Azure or 'example' for dockerhub): []
```

Si en tye.yaml. No hemos especificado usar un registro tipo, por ejemplo: localhost:5000 (revisar el fichero tye por qué deberás añadirlo), nos va a pedir que proporcionemos la URI de redis y rabbitmq. En la anterior captura lo que nos está pidiendo es que indiquemos los respectivos servicios que hemos definido en los YAML de K8s:



Workshop

Desplegar microservicios al cluster Kubernetes (5/10)

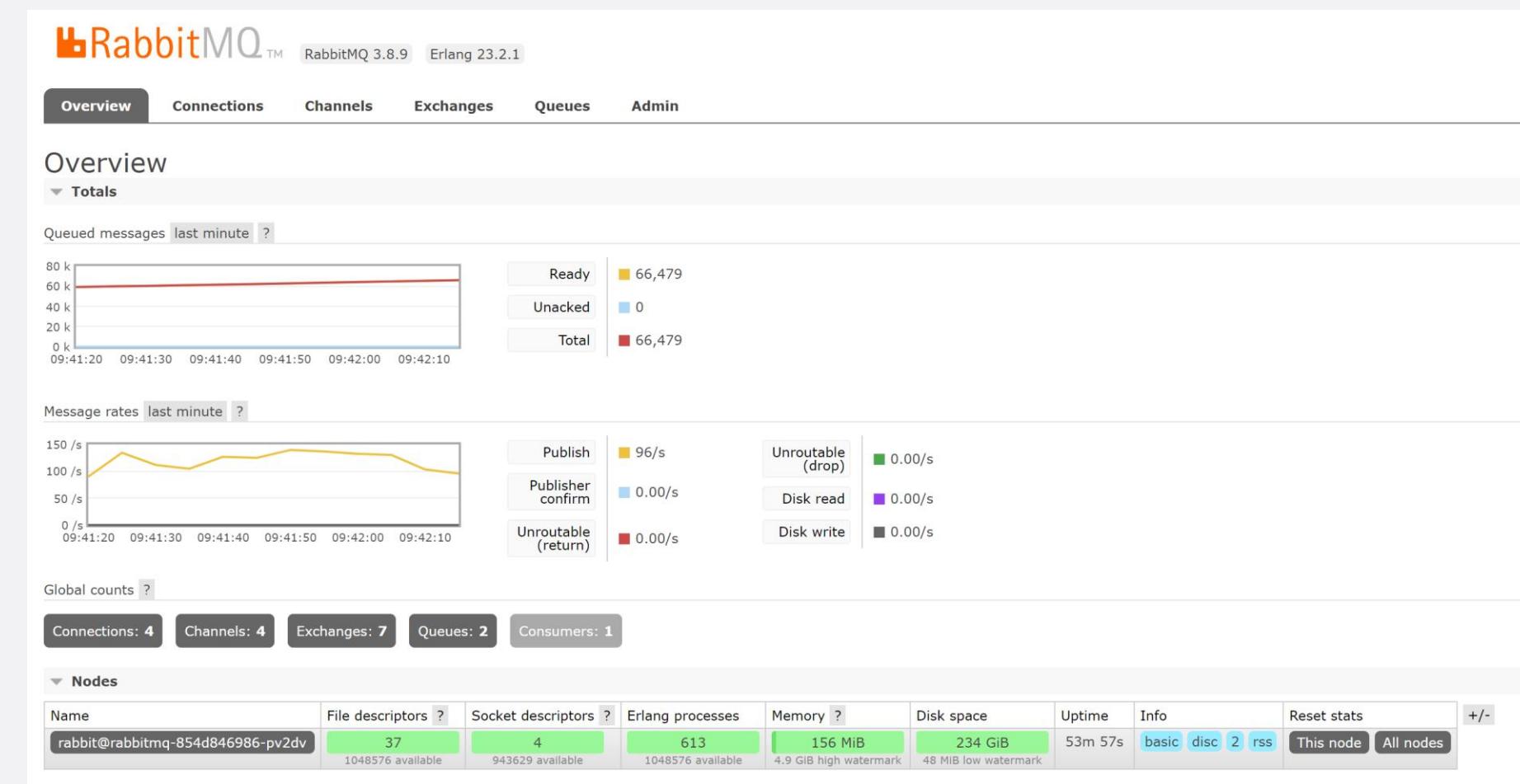


Workshop

Desplegar microservicios al cluster Kubernetes (6/10)

Ten en cuenta localhost:5000, lo que ya he indicando anteriormente. Esto creará automáticamente imágenes Docker y, por tanto, los Dockerfiles de nuestros microservicios. Los cargaré en el registro del contenedor local que hemos especificado, mientras que redis y rabbitmq son nombre de imágenes que hemos obtenido del Docker Hub publico.

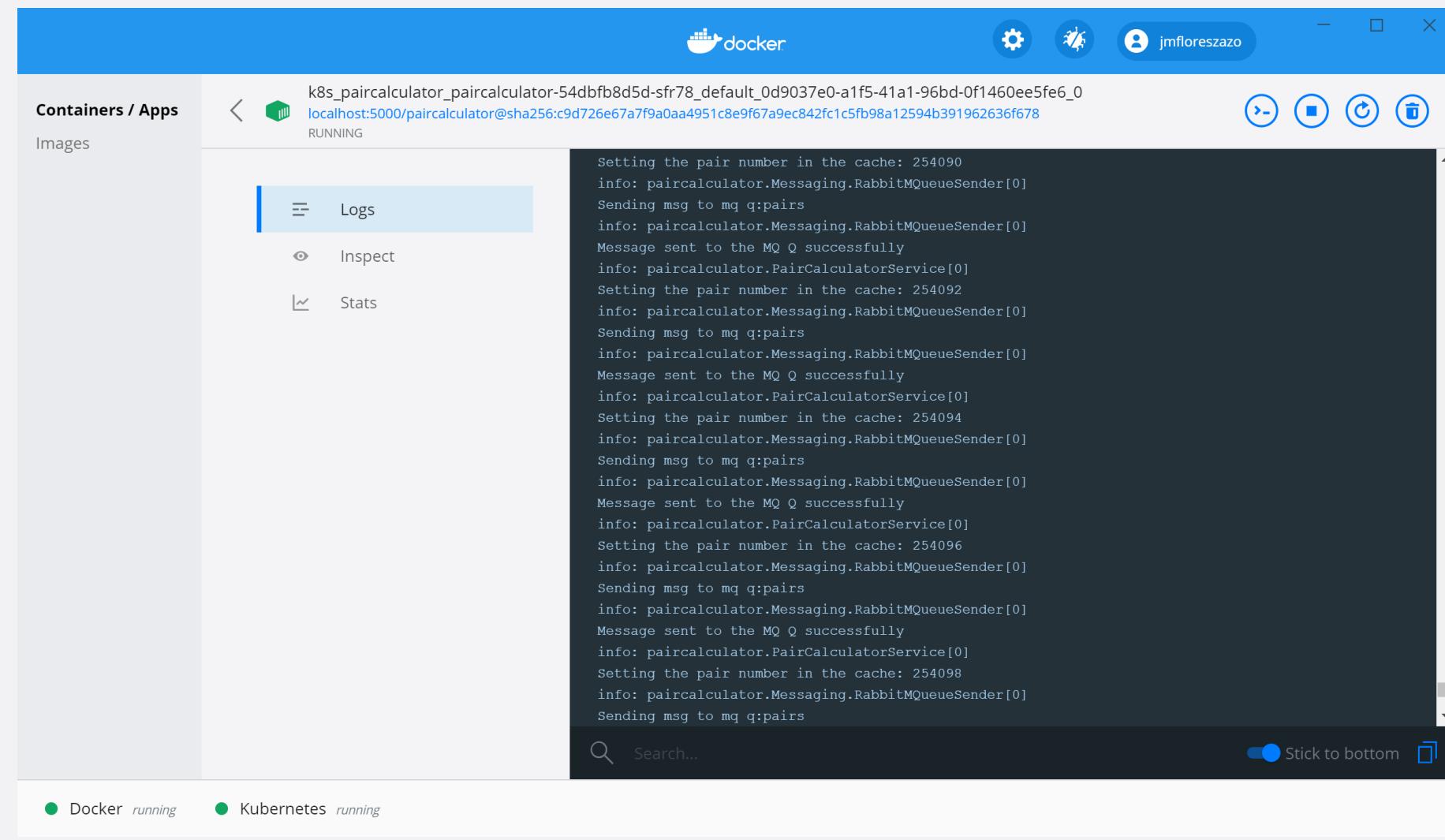
Ahora puedes usar MUI desde <http://localhost:30072> y ver le flujo de mensajes a medida que se ejecutan en el cluster de Kuberentes. Debido que nuestro cluster de K8s de un solo nodo esta basado en Docker, también podrás ver la lista de pods en ejecución en la lista de imágenes que Docker ejecuta en la máquina de desarrollo.



Workshop

Desplegar microservicios al cluster Kubernetes (8/10)

Podrás observar que para cada microservicio hoy un contenedor que se ejecuta para el pod y un contenedor que se ejecuta para la imagen del microservicio. Esto se debe al cluster de nodo único de Docker Desktop, K8s se ejecuta en el contenedor Docker y, por tanto, los pods también se ejecutan en su propio contenedor. Si deseas ver los registros de un microservicio determinado, puedes verlos desde su contenedor y no desde el contenedor del pod. Por ejemplo:



Workshop

Desplegar microservicios al cluster Kubernetes (9/10)

También puedes ver la lista de todo lo que se ejecuta en el cluster K8s en nuestro contexto de la aplicación. Ejecutando los siguientes comandos de consola:

```
kubectl get deployment  
kubectl get svc  
kubectl get secrets  
kubectl get pods
```



Workshop

Desplegar microservicios al cluster Kubernetes (10/10)

```
jmfloreszazo@DESKTOP-GSA2HFS:/mnt/c/Projects/Workshop/Microservicesapp/src$ kubectl get deployment
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
paircalculator   3/3     3           3           27m
pairclienta     3/3     3           3           27m
pairclientb     3/3     3           3           27m
pairclientc     3/3     3           3           27m
pairqconsumer   1/1     1           1           27m
rabbitmq        1/1     1           1          104m
redis           1/1     1           1          100m
jmfloreszazo@DESKTOP-GSA2HFS:/mnt/c/Projects/Workshop/Microservicesapp/src$ kubectl get svc
NAME            TYPE      CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
kubernetes      ClusterIP  10.96.0.1    <none>       443/TCP   123m
paircalculator   ClusterIP  10.96.50.30  <none>       80/TCP    28m
rabbitmq         ClusterIP  10.105.180.252 <none>      5672/TCP   105m
rabbitmq-mui    NodePort   10.101.102.184 <none>      15672:30072/TCP 105m
redis            ClusterIP  10.109.187.145 <none>      6379/TCP   102m
jmfloreszazo@DESKTOP-GSA2HFS:/mnt/c/Projects/Workshop/Microservicesapp/src$ kubectl get secrets
NAME              TYPE           DATA   AGE
binding-production-rabbitmq-secret  Opaque          3      31m
binding-production-redis-secret    Opaque          1      32m
default-token-q25qg                kubernetes.io/service-account-token  3      125m
jmfloreszazo@DESKTOP-GSA2HFS:/mnt/c/Projects/Workshop/Microservicesapp/src$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
paircalculator-54dbfb8d5d-s7jxb  1/1    Running   0          31m
paircalculator-54dbfb8d5d-sfr78  1/1    Running   0          31m
paircalculator-54dbfb8d5d-zrw7n  1/1    Running   0          31m
pairclienta-867bc785b-rdk47    1/1    Running   0          31m
pairclienta-867bc785b-sp4jt    1/1    Running   0          31m
pairclienta-867bc785b-thshs    1/1    Running   0          31m
pairclientb-5cd55d7b75-h6dbq   1/1    Running   0          31m
pairclientb-5cd55d7b75-mq5r4   1/1    Running   0          31m
pairclientb-5cd55d7b75-x4sg7   1/1    Running   0          31m
pairclientc-84667cd5d-hfh7t    1/1    Running   0          31m
pairclientc-84667cd5d-qr2vk    1/1    Running   0          31m
pairclientc-84667cd5d-rjcbz    1/1    Running   0          31m
pairqconsumer-7875db655-cflf1  1/1    Running   0          31m
rabbitmq-854d846986-pv2dv     1/1    Running   1          107m
redis-5957ddd4f4-bw89g        1/1    Running   1          104m
jmfloreszazo@DESKTOP-GSA2HFS:/mnt/c/Projects/Workshop/Microservicesapp/src$
```



Workshop

Resumen

¡Enhorabuena y muchas gracias por llegar al final!

Espero, deseo, que hayas aprendido muchas cosas y que puedas aprovechar las para poder ponerlo en práctica.

Hemos cubierto con este proyecto los siguientes puntos:

- Teoría, práctica e implementación de comunicación a través del protocolo gRPC.
- Has podido montar WSL2 en Windows.
- Has aprendido conceptos básicos de orquestación de contenedores y aplicar conceptos de K8s.
- Hemos construido una aplicación de microservicios con .NET haciendo un uso extensivo de Tye. Junto a Redis y RabbitMQ.

Me he dejado una parte muy importante que es poder mover todo esto a una nube, ya lo trataré más adelante, ya que mi objetivo ahora es crear un workshop de [ML.NET](#).



Workshop

Enlaces de interés.

gRPC:

- .NET: <https://docs.microsoft.com/es-es/aspnet/core/grpc/?view=aspnetcore-5.0>
- Conceptos: <https://grpc.io/docs/what-is-grpc/core-concepts/>
- Protocol Buffer: <https://developers.google.com/protocol-buffers/docs/overview>
- eBook de arquitectura: <https://docs.microsoft.com/en-us/dotnet/architecture/grpc-for-wcf-developers/>

WSL:

- Sitio de Microsoft: <https://docs.microsoft.com/en-us/windows/wsl/wsl-config>

Contenedores:

- <https://containerd.io/>

Y más información sobre **kubectl** CLI:

- <https://kubernetes.io/docs/reference/kubectl/cheatsheet/>





¡Gracias!

Puedes encontrarme buscando por **jmfloreszazo** en

