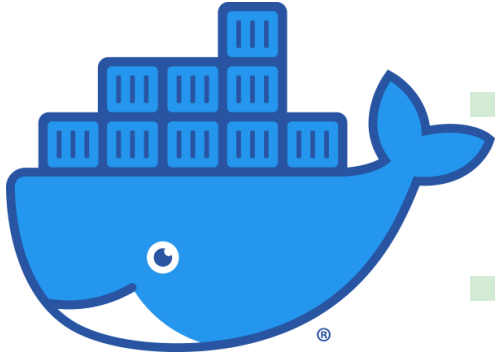
Arquitectura orientada a Microservicios

.NET Core



Por Geovani de León

Microsoft Dev Group {Cobán}

Comparte | Aprende | Desarrolla

Geovani de León (Yova)

Ingeniero de Software en Pensotec

Líder de Microsoft Dev Group Cobán

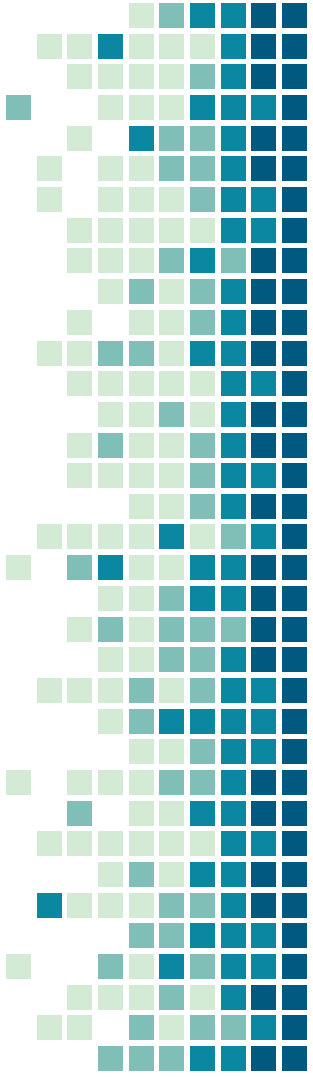
Docente Universitario

https://github.com/yovafree

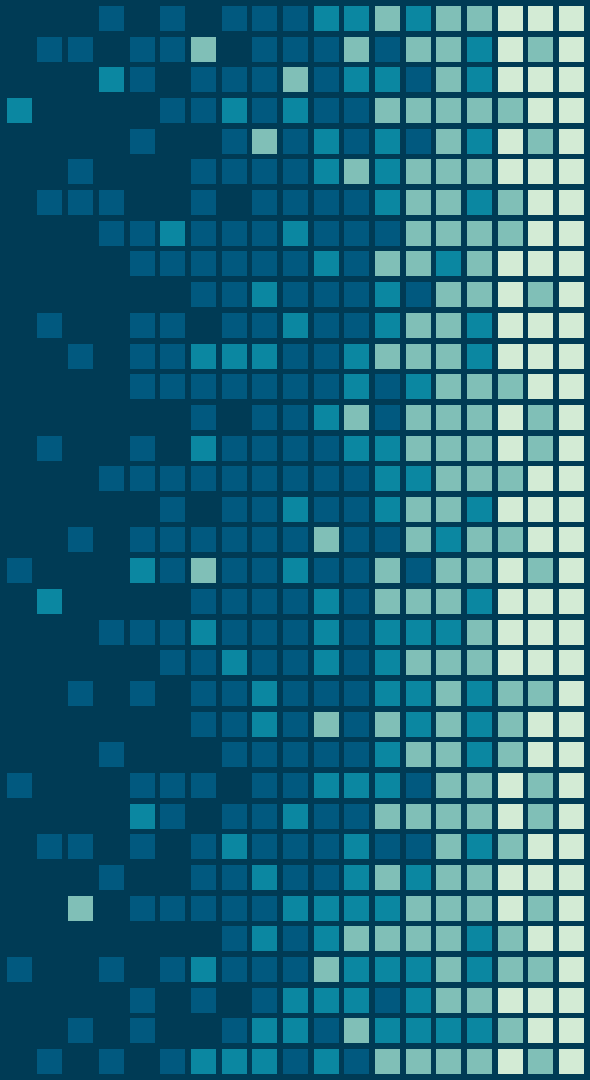
https://yovadeleon.dev

Agenda

- Arquitectura monolítica
- Arquitectura basada en microservicios
- Retos y Soluciones de Microservicios
- Arquitectura Monolítica vs Microservicios
- Características de Microservicios
- Buenas prácticas en Microservicios
- .NET Core
- Demo 1
- Docker
- Demo 2
- Demo 3
- Preguntas y respuestas



Arquitectura de Software



“Es un conjunto de patrones que proporcionan un marco de referencia necesario para guiar la construcción de un software.

Arquitectura Monolítica

En la ingeniería de software, una aplicación monolítica describe una única aplicación de software en niveles en los que la interfaz de usuario y código de acceso a datos se combinan en un solo programa de una plataforma única.

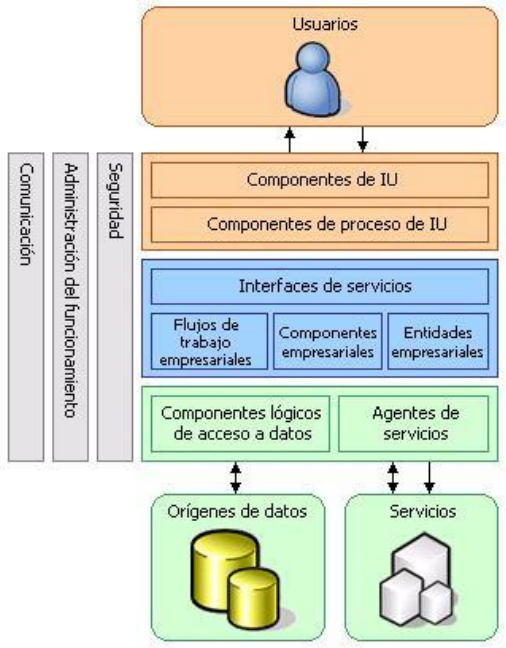
Arquitectura Monolítica

Pros

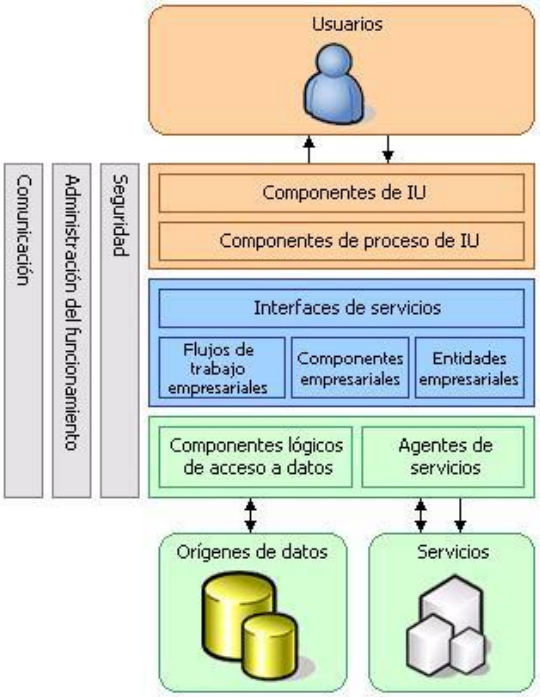
- Fácil de desarrollar
- Fácil de entender
- Baja complejidad

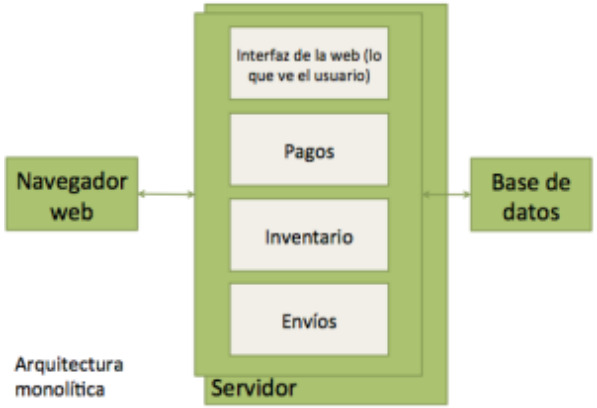
Contras

- Baja especialización
- Recursos altos para escalar
- Nuevos cambios, nueva versión!



Arquitectura Monolítica - Ejemplo





Arquitectura basada en Microservicios

Es un estilo de arquitectura y un modo de programar software. Las aplicaciones se dividen en componentes más pequeños, y son independientes entre sí.

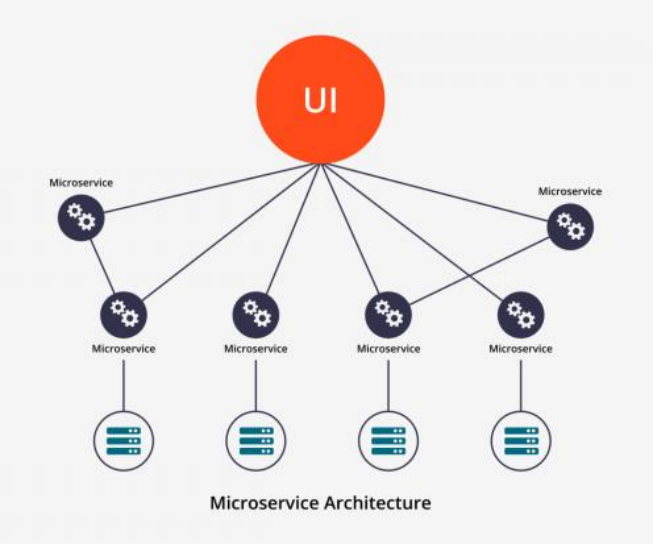
Cada uno de estos elementos es un microservicio. Este enfoque privilegia el nivel de detalle, la sencillez y la capacidad de compartir un proceso similar en varias aplicaciones.

Es un enfoque fundamental hacia un modelo nativo de la nube.

Arquitectura basada en Microservicios

Pros

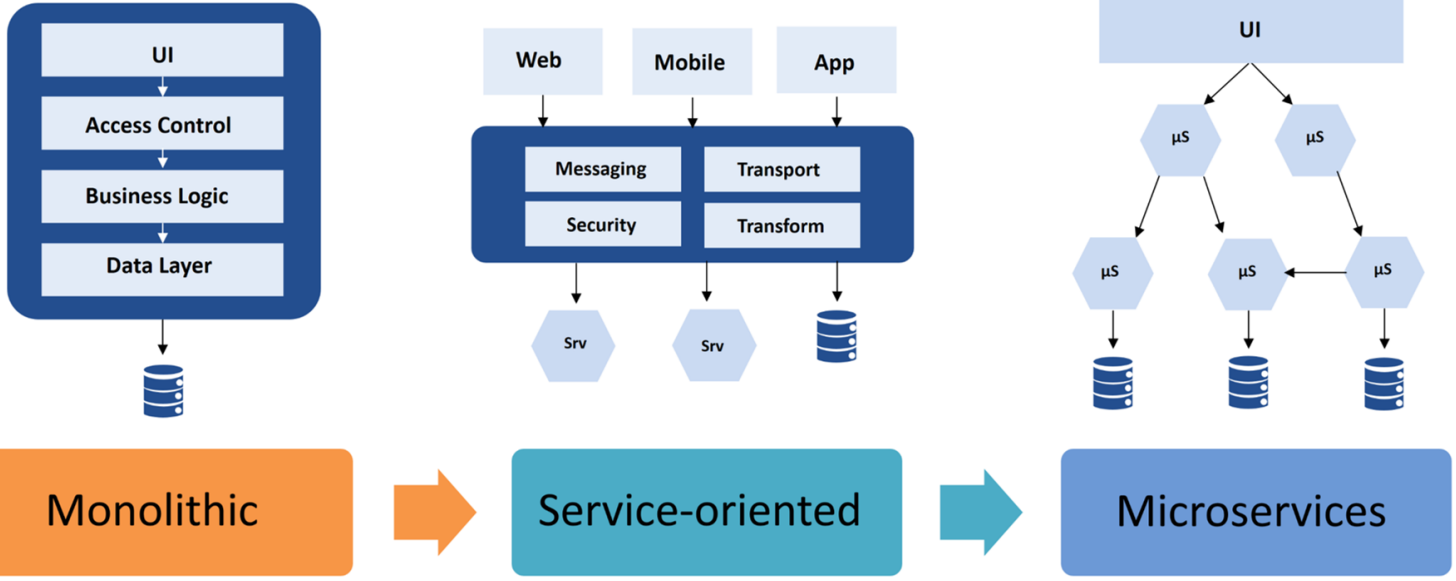
- Alta granularidad
- Alta especialización
- Escalabilidad
- Facilidad de combinar con diversas tecnologías
- Resiliencia a fallos



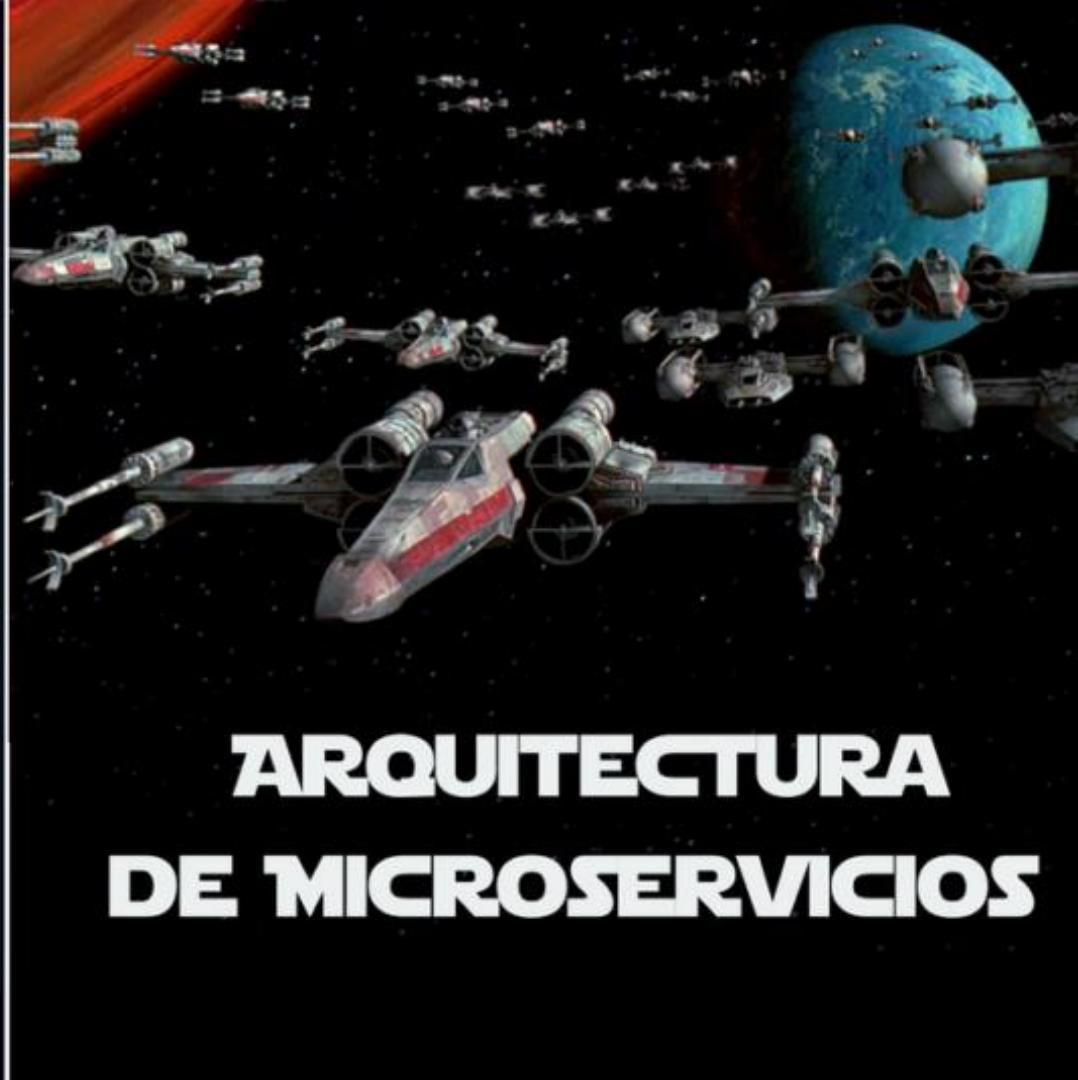
Contras

- Alta complejidad
- Requiere buena comunicación del equipo de desarrollo
- Requiere alto dominio de patrones de diseño y arquitectura

Evolución de la Arquitectura de Software



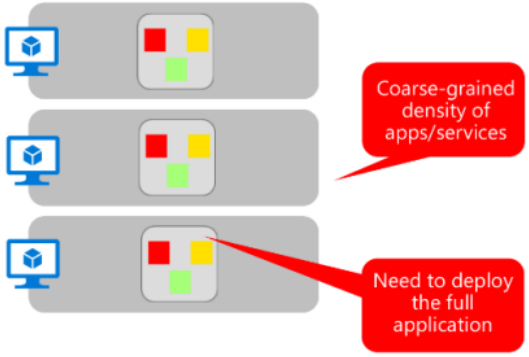

ARQUITECTURA
MONOLÍTICA


ARQUITECTURA
DE MICROSERVICIOS

Enfoque Monolítico vs Microservicios

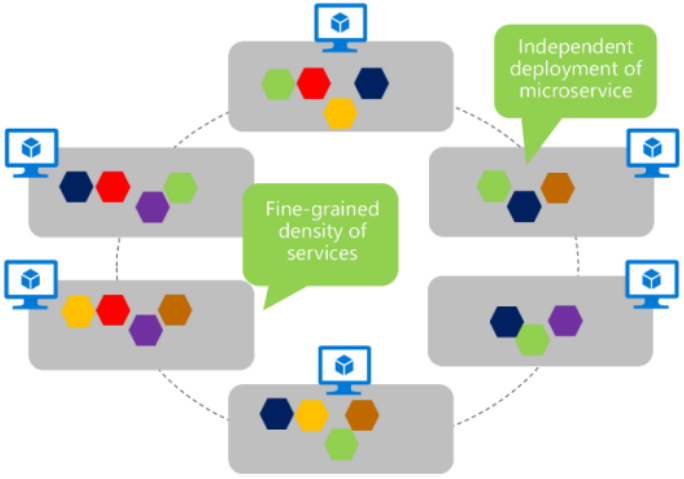
Monolithic deployment approach

- A traditional application has most of its functionality within a few processes that are componentized with layers and libraries.
- Scales by cloning the app on multiple servers/VMs



Microservices application approach

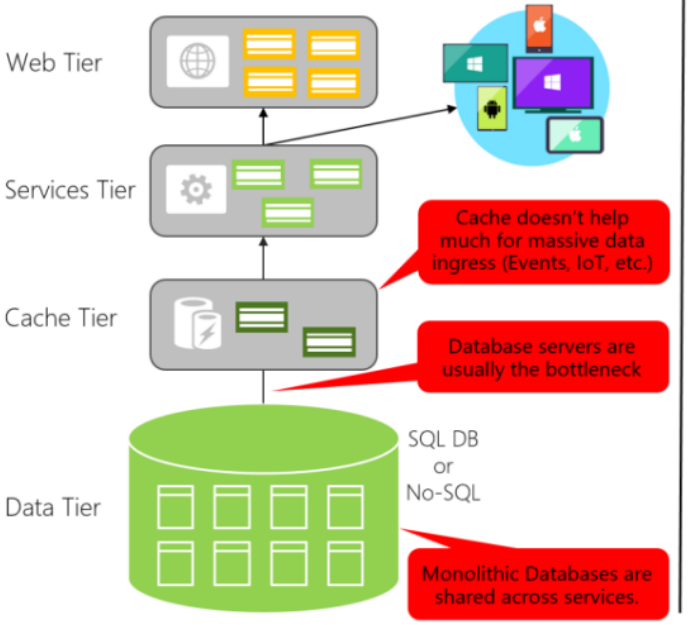
- A microservice application segregates functionality into separate smaller services.
- Scales out by **deploying each service independently** with multiple instances across servers/VMs



Enfoque Monolítico vs Microservicios

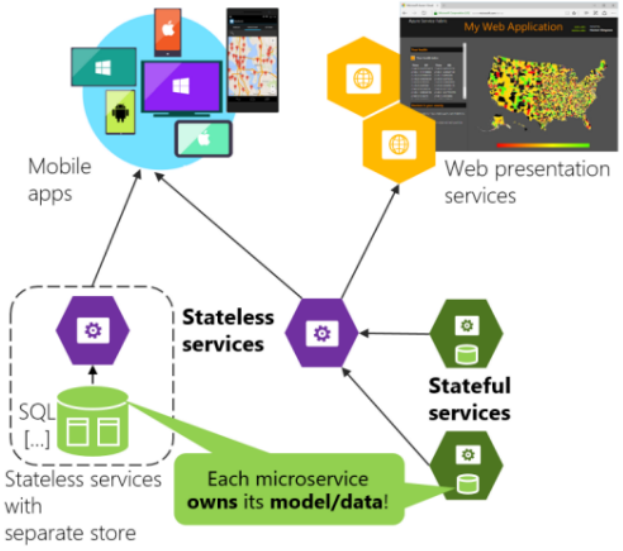
Data in Traditional approach

- Single monolithic database
- Tiers of specific technologies

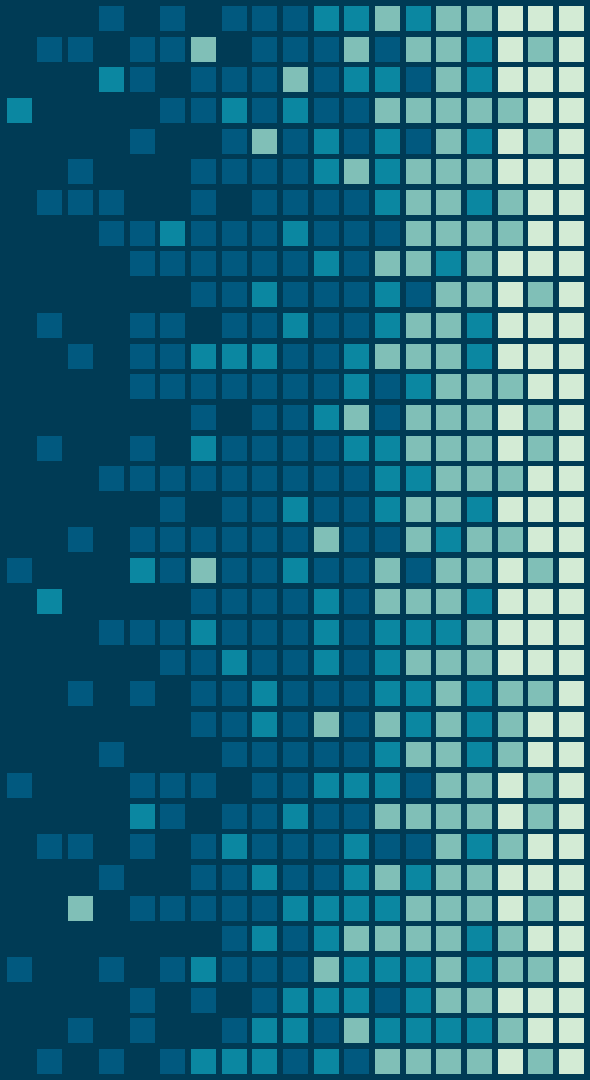


Data in Microservices approach

- Graph of interconnected microservices
- State typically scoped to the microservice
- Remote Storage for cold data



Retos al abordar el enfoque en Microservicios



Reto #1: Cómo definir los límites de cada microservicio

- Se debe intentar identificar las islas de datos desacopladas y los diferentes contextos dentro de la aplicación.

- “un Usuario puede referirse a un usuario en el contexto de identidad o membresía, a un cliente en el contexto de CRM, a un comprador en el contexto de pedidos y así sucesivamente”

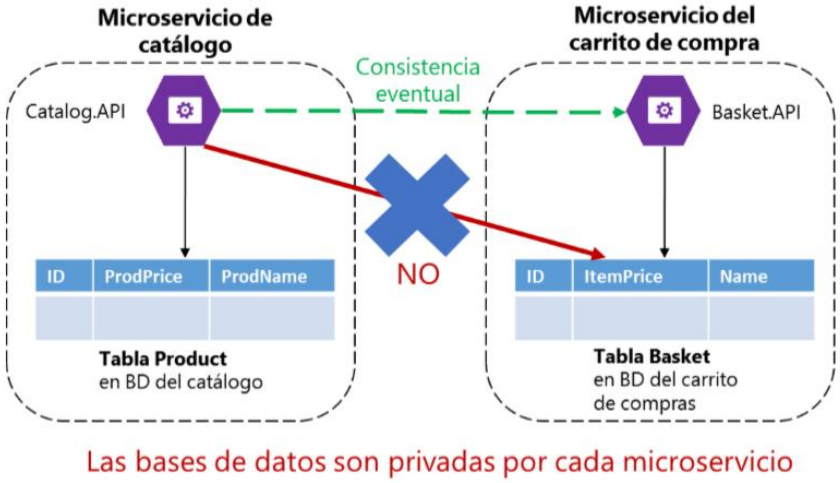
Reto #2: Cómo crear consultas que recuperan datos de varios microservicios

- Tomar en cuenta que esta base de datos centralizada, se usará sólo para consultas e informes que no necesitan datos en tiempo real. Las actualizaciones y transacciones originales, como su fuente original, tienen que estar en las bases de datos de los microservicios

- **API Gateway**
- **CQRS**
- **Datos fríos**

Reto #3: Cómo lograr consistencia entre múltiples microservicios

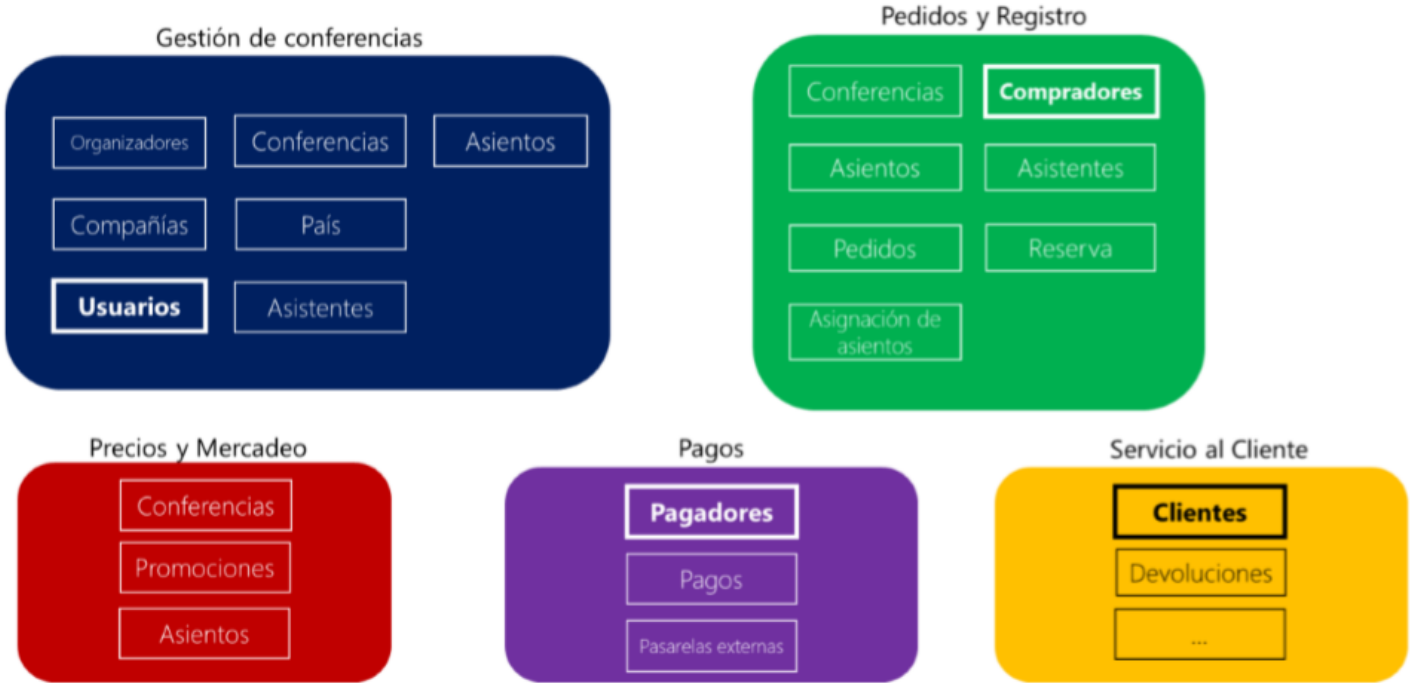
- los datos de cada microservicio deben ser privados y sólo se debe poder acceder a ellos usando el API del microservicio



Reto #4: Cómo diseñar comunicaciones a través de los límites de los microservicios

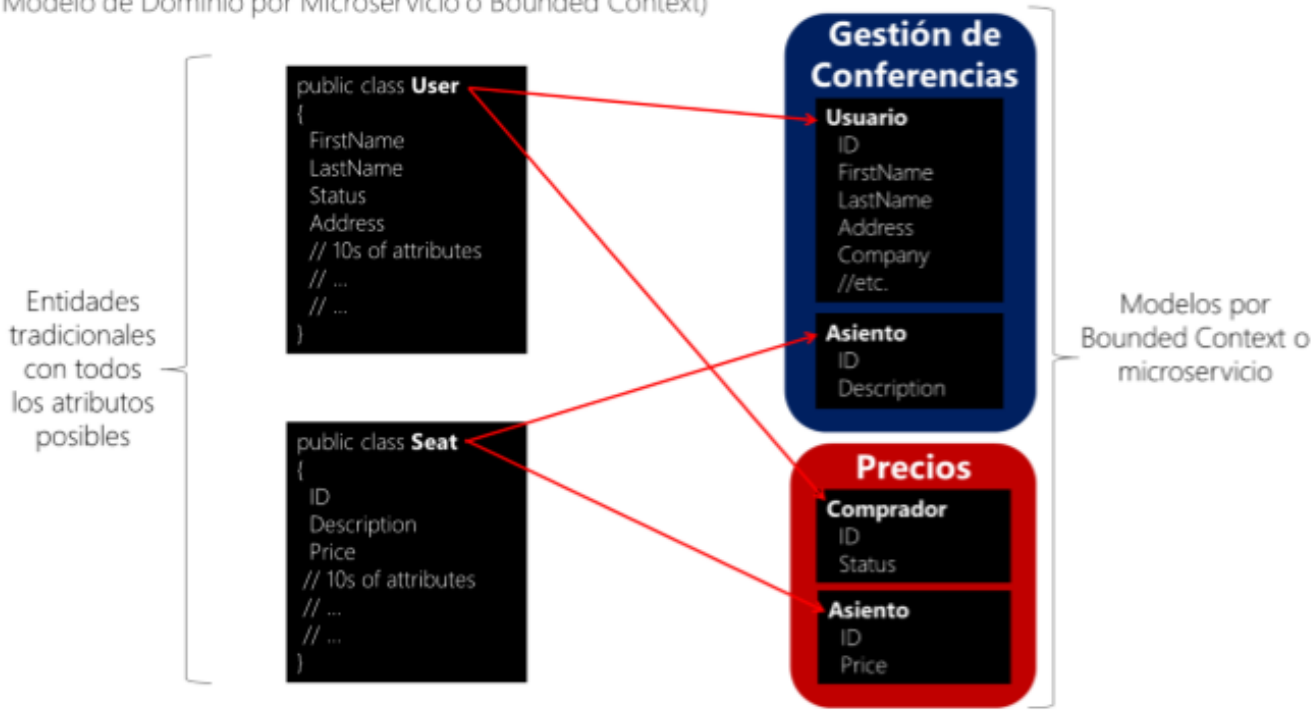
- La comunicación a través de los límites de los microservicios es un verdadero desafío
- La comunicación no se refiere a qué protocolo usar (HTTP y REST, AMQP, mensajes, etc.)
- Un enfoque popular es implementar microservicios basados en HTTP (REST), debido a su simplicidad

Identificando un modelo de dominio por microservicio o Bounded Context



Descomponiendo un modelo de datos tradicional en múltiples Modelos de Dominio

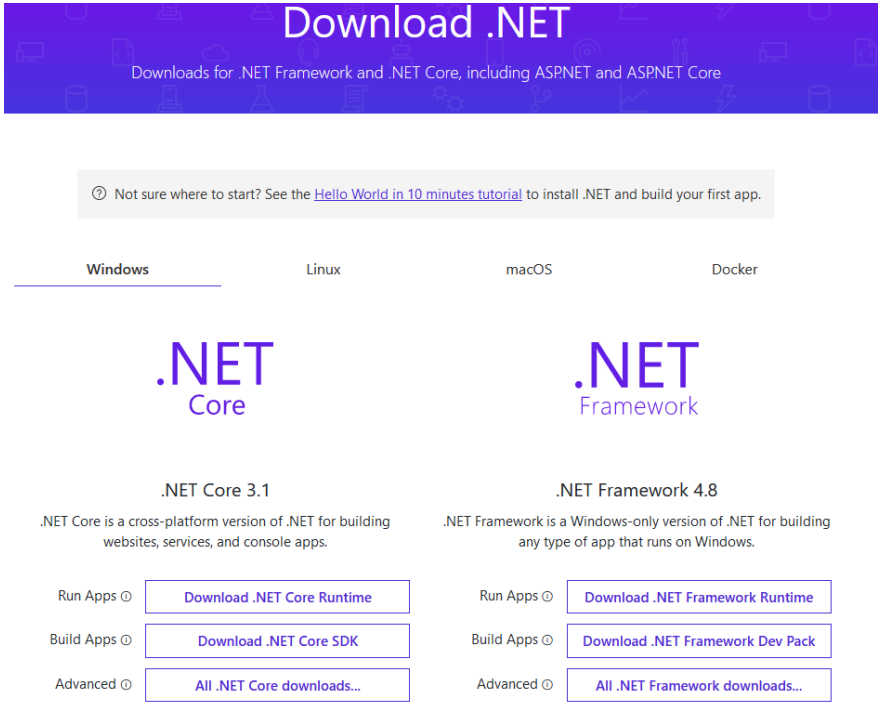
(Un Modelo de Dominio por Microservicio o Bounded Context)



.NET Core

- .NET Core es una implementación del estándar .NET que, al igual que otras implementaciones como .NET Framework o Mono, incluye todo lo necesario para crear y ejecutar aplicaciones como:

- Compiladores
- Bibliotecas de clases básicas o la máquina virtual o runtime que proporciona el entorno donde se ejecutan las aplicaciones.



https://dotnet.microsoft.com/download

¿Qué hay de nuevo en .NET Core 3?

- Language improvements C# 8.0
- .NET Standard 2.1
- Single-file executables

```
dotnet publish -r win10-x64 -p:PublishSingleFile=true
```

- Windows Native Interop
- Linux
  - GPIO Support for Raspberry Pi
  - ARM64 Linux support
  - Docker and cgroup memory Limits

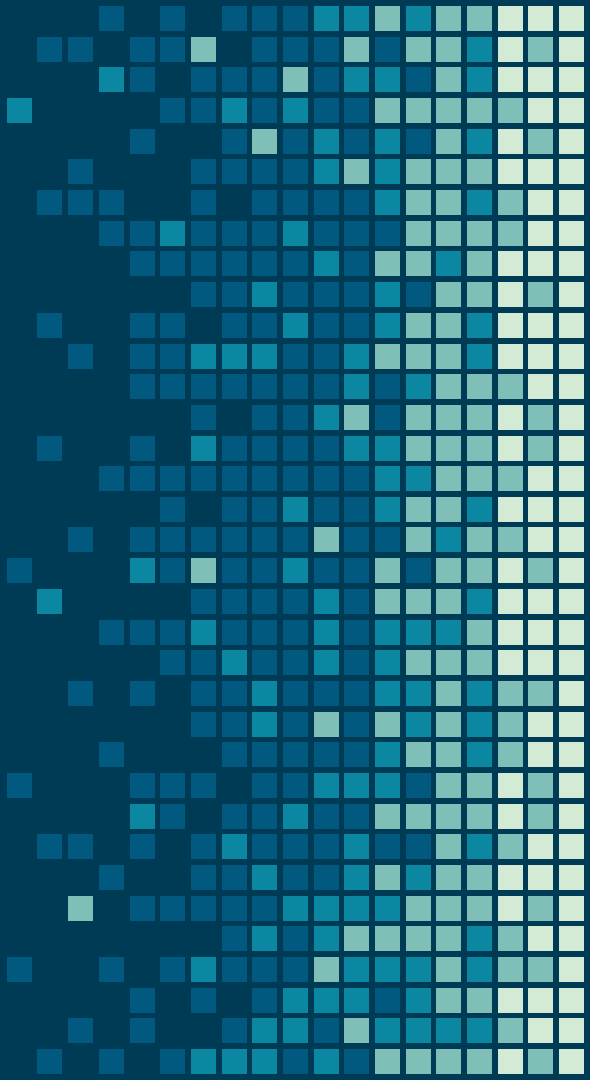
.NET Core 3.1

Long-term support

.NET Core 3.1 is an LTS release with support from Microsoft for the next three years. It's highly recommended that you move your apps to .NET Core 3.1. The current lifecycle of other major releases is as follows:

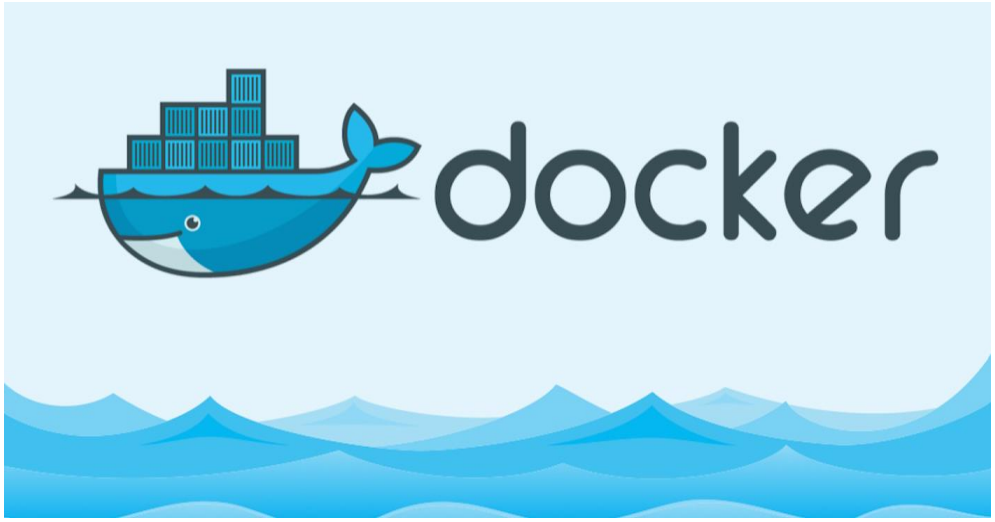
Release	Note
.NET Core 3.0	End of life on March 3, 2020.
.NET Core 2.2	End of life on December 23, 2019.
.NET Core 2.1	End of life on August 21, 2021.

DEMO 1



Docker:

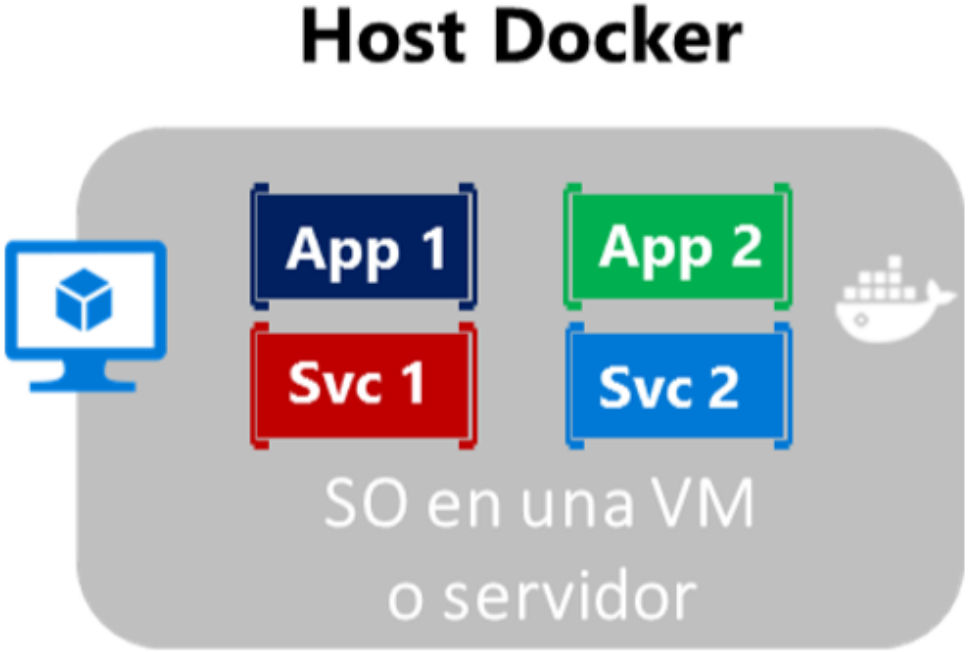
Estándar en la industria de contenedores, siendo soportado por las empresas más significativas en los ecosistemas de Windows y Linux

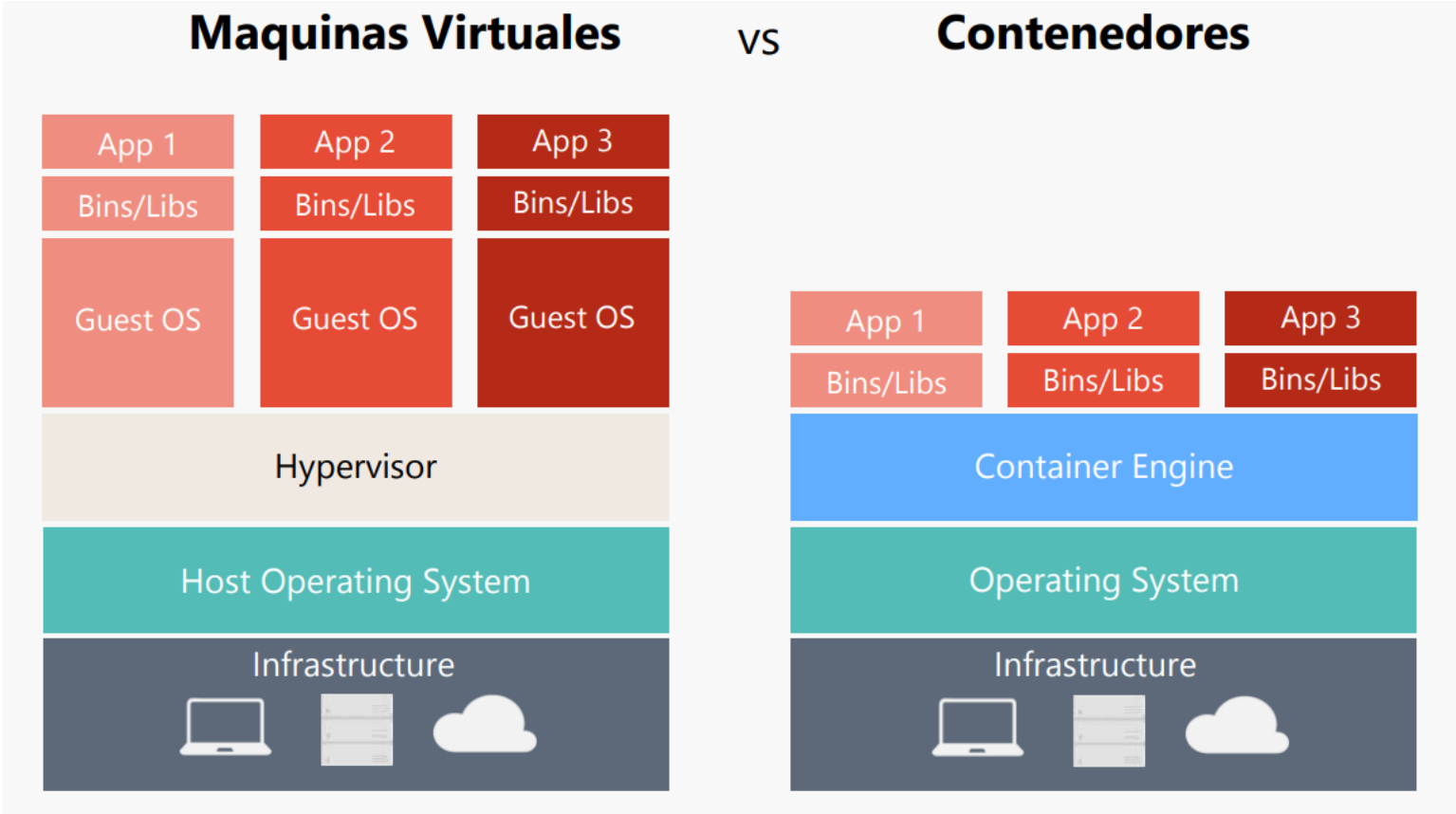


Contenerización

Es un enfoque de desarrollo de software en el que una aplicación o servicio, sus dependencias y su configuración (en forma de ficheros de manifiesto para despliegue) se empaquetan juntos como una imagen de contenedor (o simplemente imagen para simplificar, cuando no haya ambigüedad).

Una aplicación basada en contenedores se puede probar como una unidad y desplegar como una instancia de la imagen, en el sistema operativo que funciona como host


Host Docker
App 1
App 2
Svc 1
Svc 2
SO en una VM
o servidor


Maquinas Virtuales
vs
Contenedores
App 1
App 2
App 3
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Host Operating System
Infrastructure
App 1
App 2
App 3
Bins/Libs
Bins/Libs
Bins/Libs
Container Engine
Operating System
Infrastructure

Taxonomía básica en Docker

Docker Registry Alojado

Docker Trusted Registry interno

Interno
('N' organizaciones privadas)

Docker Hub Registry

Docker Trusted Registry en-la-nube

Azure Container Registry

AWS Container Registry

Google Container Registry

Quay Registry

Other Cloud

Nube pública
(proveedores específicos)

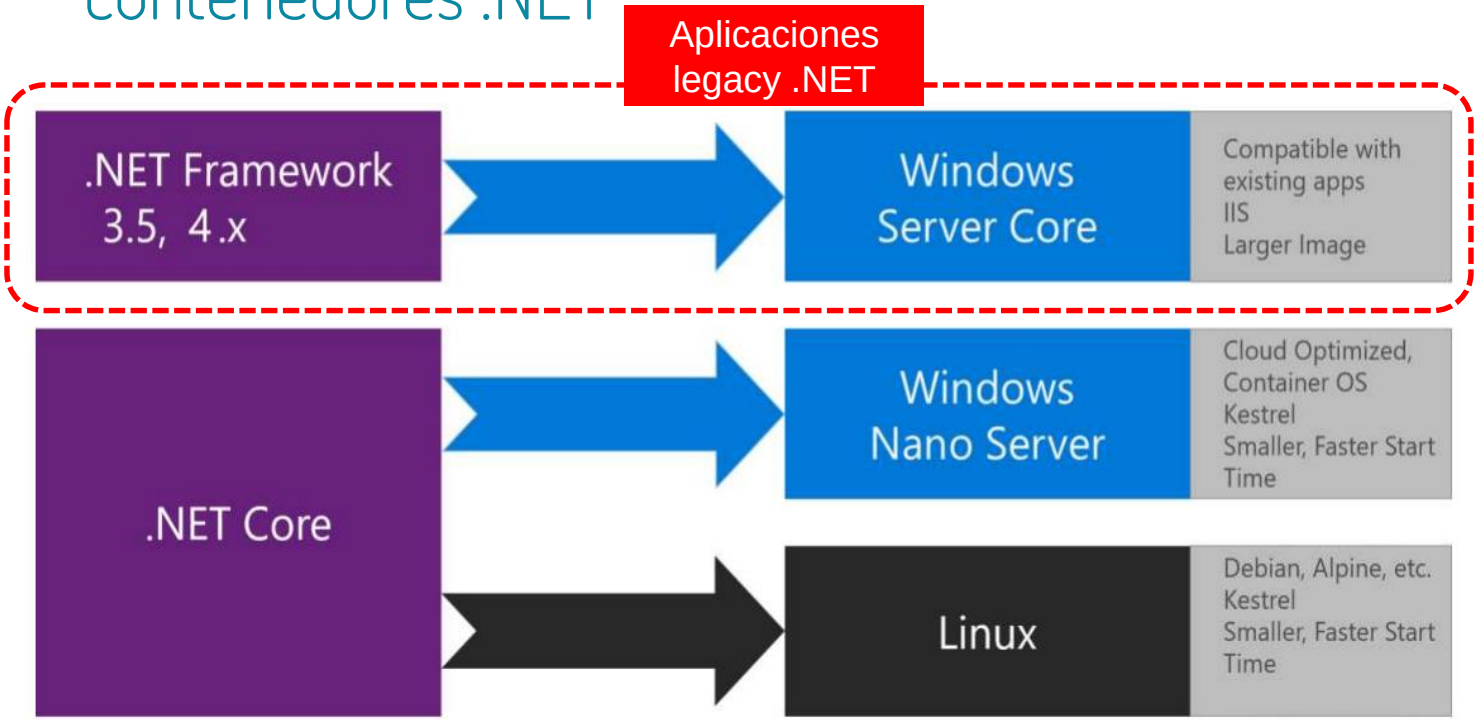
A qué sistema operativo apuntar con contenedores .NET

La diversidad de sistemas operativos soportados por Docker y las diferencias entre .NET Framework y .NET Core, debe apuntar a un sistema operativo específico y versiones específicas según el framework que esté utilizando.

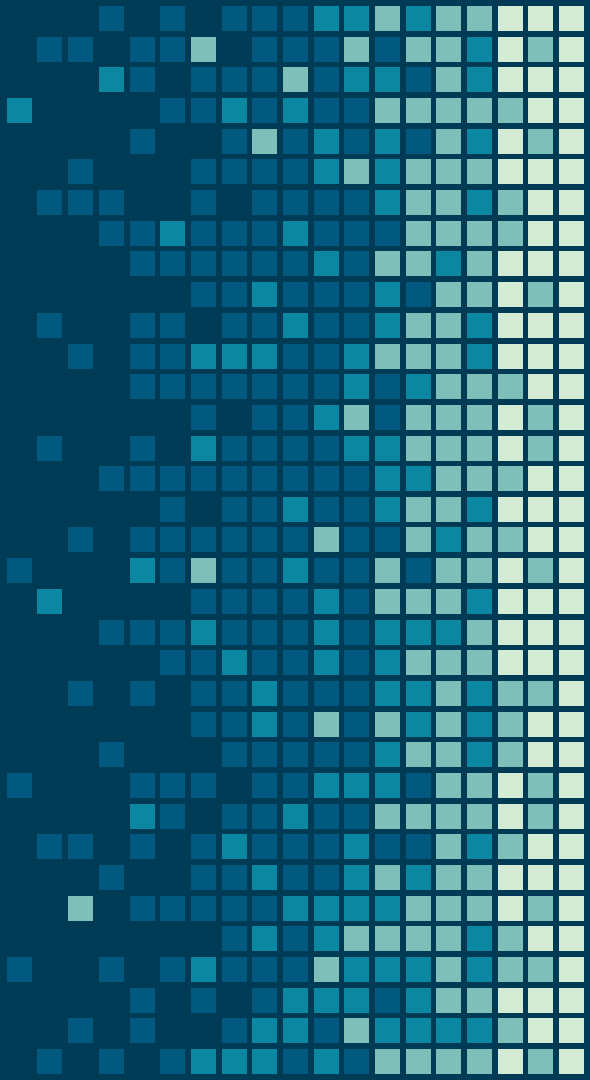
Para Windows, puede usar Windows Server Core o Windows Nano Server.

Para Linux, están disponibles múltiples distribuciones y están soportadas en imágenes oficiales de .NET en Docker (como Debian o Alpine).

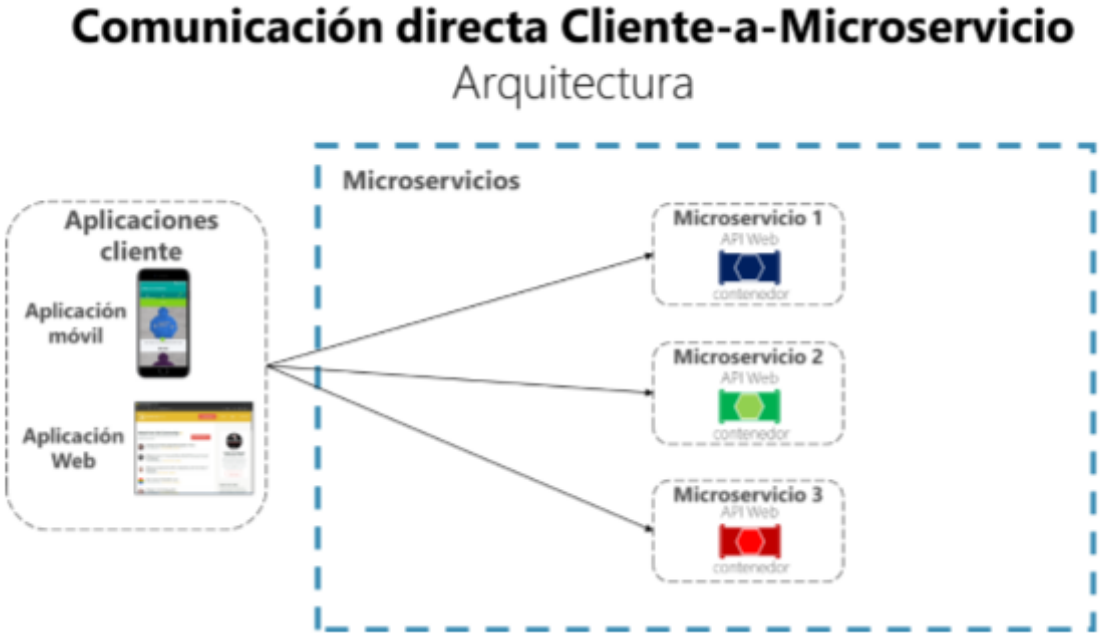
Qué sistema operativo desplegar en contenedores .NET



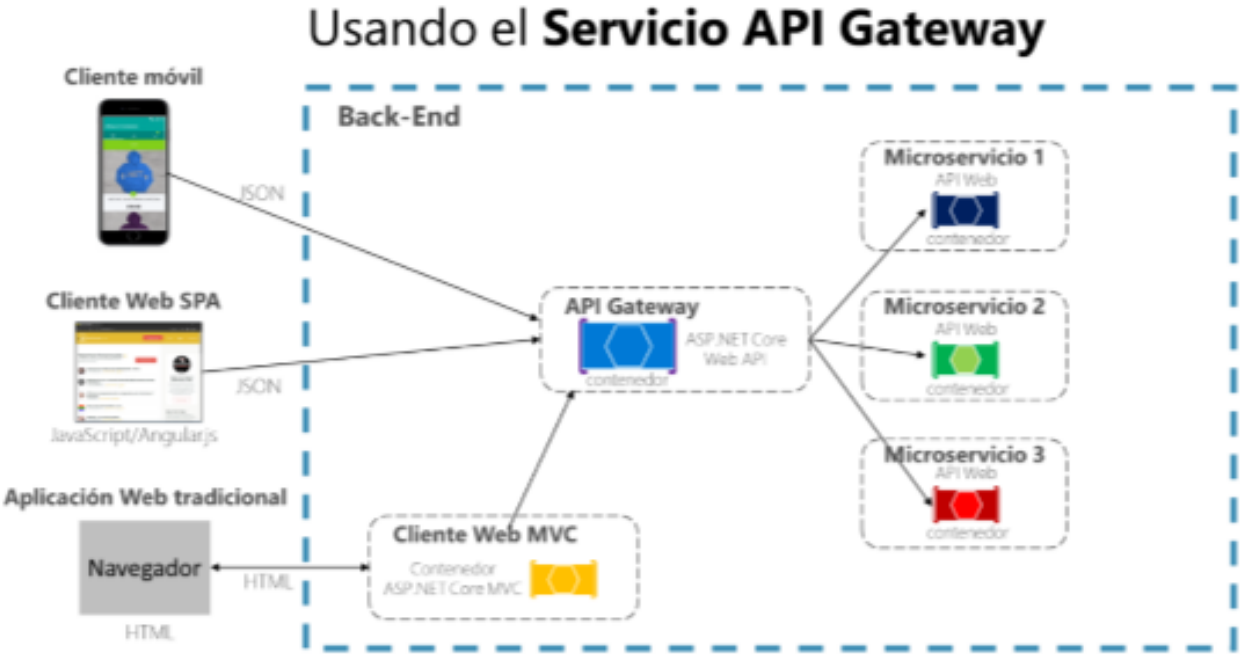
DEMO 2



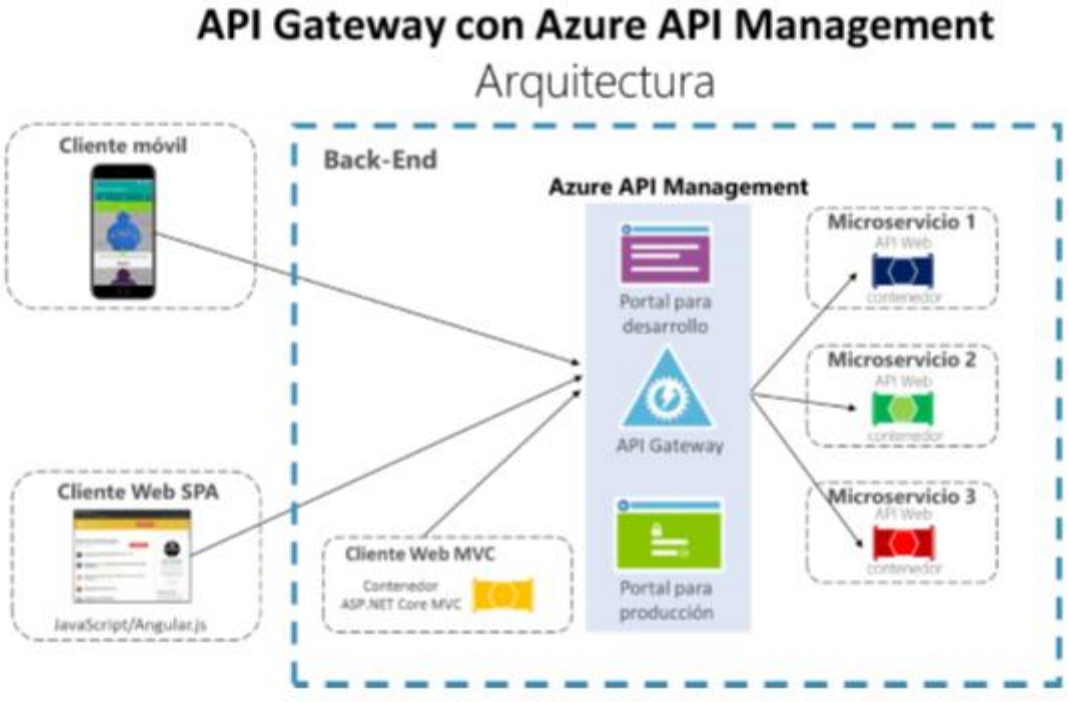
Enfoque de comunicación entre Microservicios



Enfoque de comunicación entre Microservicios

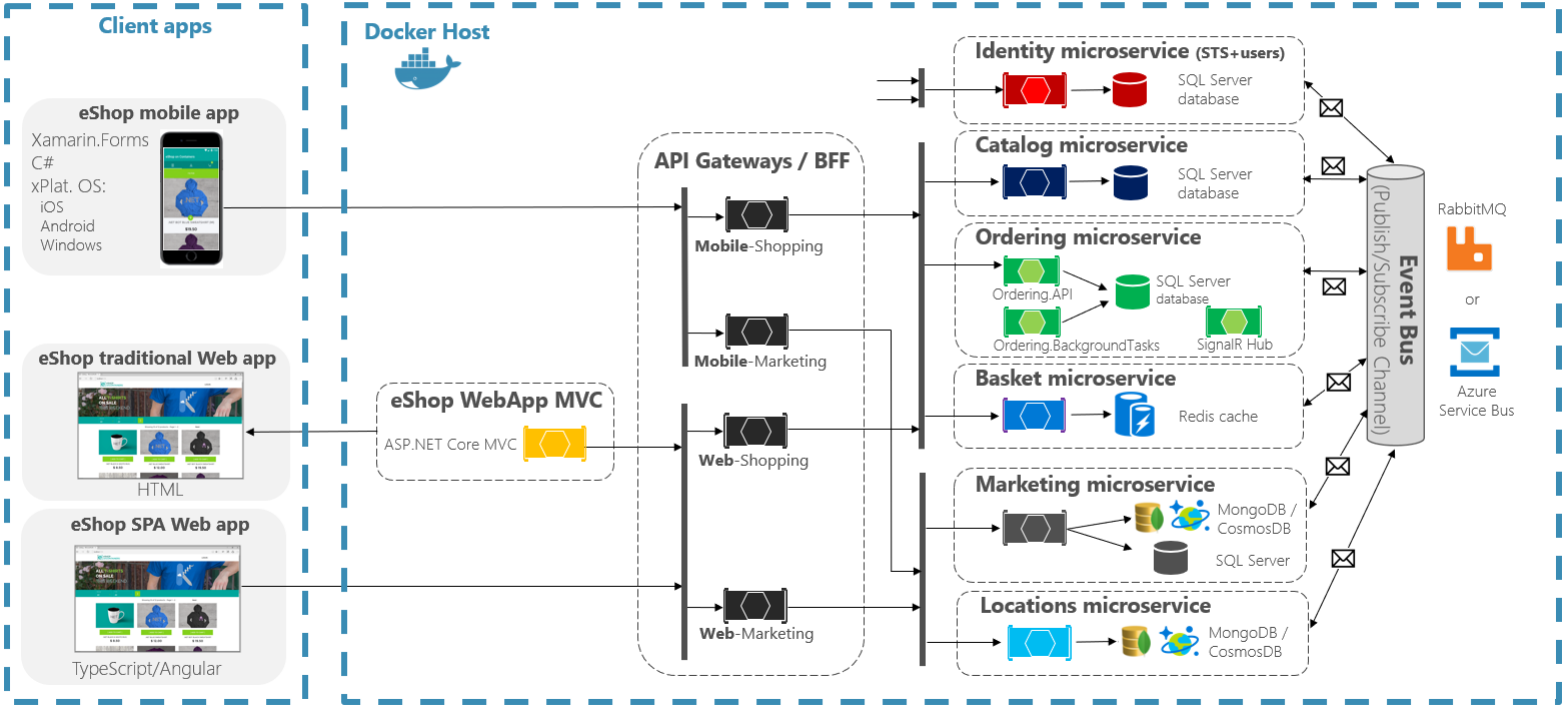


Enfoque de comunicación entre Microservicios



eShopOnContainers reference application

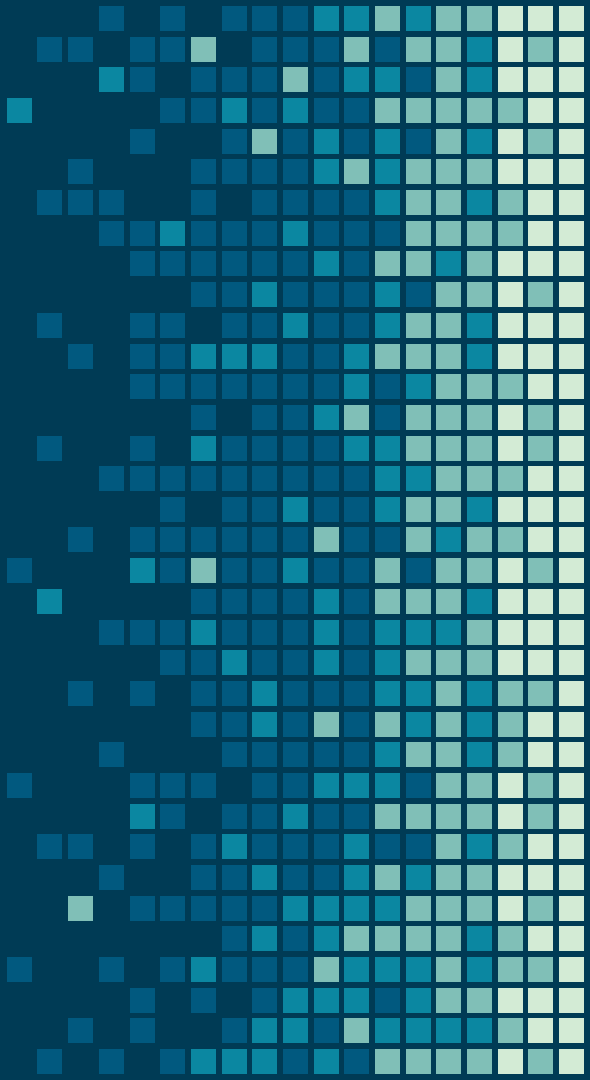
(Development environment architecture)



Enfoque de comunicación entre Microservicios

- Async pub/subs communication
  - Event Bus, RabbitMQ <- Mecanismos de cola y mensajería para manejar solicitudes.
- Health Check
  - Docker <- Sanidad del contenedor. Traces
- Resilent Cloud Applications ->
  - HttpClientFactory Core 2.1 <- Permite crear políticas,
  - Polly <- Es un proyecto que permite realizar políticas.
- API Gateways vs Direct Communication
  - Ocelot <- Framework para .NET que permite realizar un API Gateway
- Orchestrators: Scale-out & dev Collaboration
  - AKS, Docker Compose, Azure Dev Spaces
  - Consul <- permite realizar descubrimiento de servicios (Service Discovery)

DEMO 3

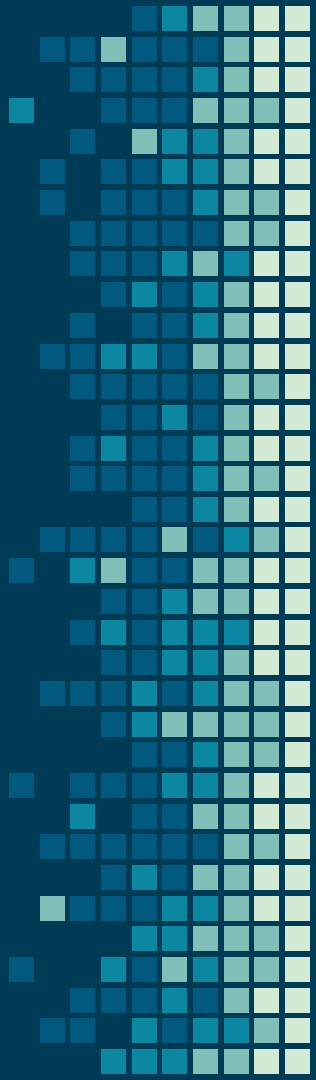


https://github.com/yovafree/demo-microservicios-dotnet-core

GRACIAS!

¿Preguntas?

Puedes comunicarte conmigo:
egdeleon@pensotec.com
Github.com/YovaFree



Microsoft
Dev Group
{Cobán}

Comparte | Aprende | Desarrolla

.NET Core

Christian Sánchez

Informatics Engineer

Co-Organizador Flutter Honduras

Creando API Rest en .NET Core usando CodeFirst

Octubre 3, 18:00 CST.

Referencias

https://docs.microsoft.com/es-es/dotnet/architecture/microservices/
[Microsoft]

https://microservices.io/
[Chris Richardson]