MICROSERVICIOS EN NET 5

ARQUITECTURA PARA CONTENEDORES
SESION I

AFORO255 TRAINING CENTER

Instructor

Ivan Cuadros Altamirano
Lead Software Architect

Agenda

- Introducción a la Arquitectura de Microservicios
- Revisión de Arquitectura propuesta
- Introducción a Docker
- Creación Bases de Datos
- Hands ON – Construcción de Microservicios Base

Arquitectura de Microservicios

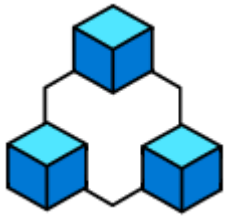
- Los microservicios son un estilo de arquitectura de software en que las aplicaciones se componen de **pequeños módulos independientes** que se comunican entre sí mediante contratos de API bien definidos

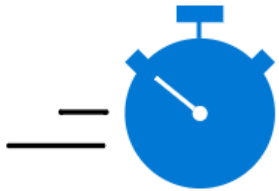
- Estos módulos de servicios son bloques de creación altamente desacoplados, lo suficientemente pequeños para implementar una única funcionalidad.

- **La finalidad de las arquitecturas de microservicios es facilitar el desarrollo y el escalado de aplicaciones. Las arquitecturas de microservicios fomentan la colaboración entre equipos independientes y les permite introducir nuevas funcionalidades en el mercado más rápido**.

Motivos para usar los microservicios

- Compilar servicios de forma independiente
- Escalar servicios de forma autónoma
- Usar el mejor enfoque
- Aislar puntos de error
- Entregar valor más rápido





Principios de una arquitectura basada en microservicios

- Escalabilidad
- Disponibilidad
- Resistencia
- Flexibilidad
- Independiente, autónomo
- Gobierno descentralizado
- Aislamiento de fallas
- Aprovisionamiento automático
- Entrega continua a través de DevOps

Retos de una arquitectura basada en microservicios

Aunque traen muchas ventajas, los microservicios son un concepto relativamente nuevo y, por lo cual, presentan bastante retos:

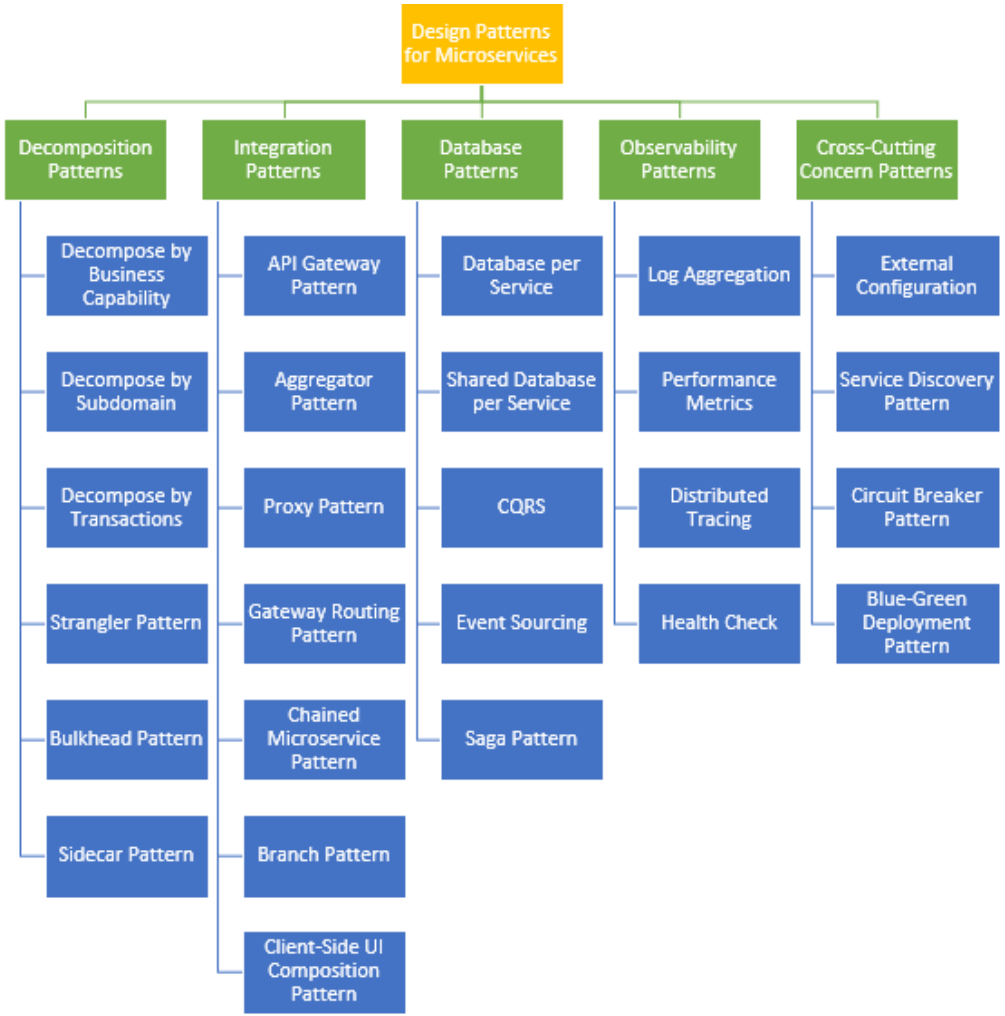
- En primer lugar, la **complejidad**: Una aplicación basada en microservicios es más compleja que un monolito, ya que está compuesta por muchos servicios distintos e independientes. Se necesita por lo cual de una política de gobernanza adecuada.
- Además, **manejar los fallos** es más complicado, ya que se necesita monitorizar distintas piezas para detectar los posibles problemas.
- Finalmente, no todos los profesionales de IT poseen los **conocimientos necesarios** para desarrollar y gestionar correctamente una arquitectura de microservicios.

Diseño de la arquitectura de microservicio

Los microservicios ofrecen grandes ventajas, pero también generan nuevos desafíos enormes. Los patrones de arquitectura de microservicios son los pilares fundamentales a la hora de crear una aplicación basada en microservicios.

Podemos clasificar estos patrones de la siguiente manera:

- Patrones de descomposición
- Patrones de Integración
- Patrones de Base de datos
- Patrones de obserbavilidad
- Patrones de Cross-Outing



Los patrones de diseño de la arquitectura de microservicios es una colección de patrones para aplicar la arquitectura de microservicios. Tiene dos objetivos:

- Le permite decidir si los microservicios son adecuados para su aplicación.
- Le permite utilizar la arquitectura de microservicios correctamente.

Patrones de descomposición

- **Descomposición por capacidad empresarial**: Definir los servicios correspondientes a las capacidades comerciales. Una capacidad empresarial es un concepto del modelado de arquitectura empresarial. Es algo que hace una empresa para generar valor. Una capacidad comercial a menudo se corresponde con un objeto comercial (La gestión de pedidos es responsable de los pedidos, La gestión de clientes es responsable de los clientes)

- **Descomposición por subdominio:** Definir los servicios correspondientes a los subdominios de Diseño controlado por dominio (DDD). DDD se refiere al espacio problemático de la aplicación, el negocio, como dominio. Un dominio consta de varios subdominios. Cada subdominio corresponde a una parte diferente del negocio. Los subdominios pueden ser Catalogo de producto, gestión de pedido, gestión de entrega, etc.

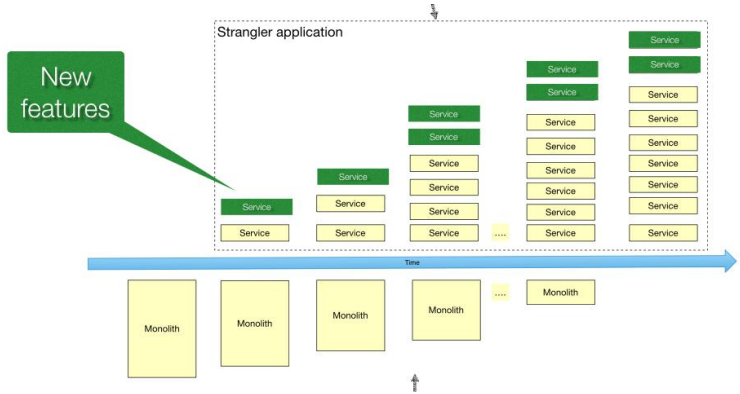
Patrones de descomposición

- **Descomposición autónoma**: Diseñar un servicio para que pueda responder a una solicitud síncrona sin esperar la respuesta de ningún otro servicio. Se puede aplicar utilizando los patrones CQRS y Saga. Un servicio autónomo utiliza el patrón Saga para mantener de forma asincrónica la coherencia de los datos. Utiliza el patrón CQRS para mantener una réplica de los datos que pertenecen a otros servicios.

- **Descomposición por equipo:** Cada servicio es propiedad de un equipo, que es el único responsable de realizar cambios. Idealmente, cada equipo tiene un solo servicio. Cada equipo es responsable de una o más funciones comerciales. Un equipo debe tener exactamente un servicio a menos que exista una necesidad comprobada de tener varios servicios.

Patrones de descomposición

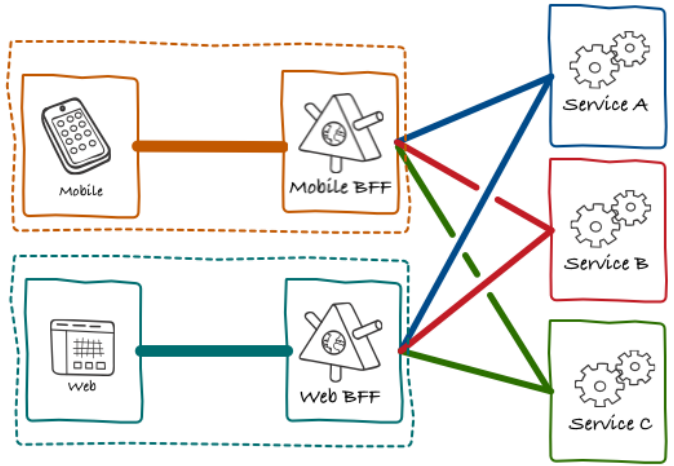
- **Descomposición estranguladora**: Modernizar una aplicación desarrollando gradualmente una nueva aplicación (estranguladora) en torno a la aplicación heredada. En este escenario, la aplicación estranguladora tiene una arquitectura de microservicio. La aplicación estranguladora consta de dos tipos de servicios. Primero, hay servicios que implementan funcionalidades que anteriormente residían en el monolito. En segundo lugar, hay servicios que implementan nuevas funciones. Estos últimos son particularmente útiles ya que demuestran a la empresa el valor de usar microservicios.



Patrones de integración

- **API Gateway**: Implementar una puerta de enlace API que sea el punto de entrada único para todos los clientes.
  La puerta de enlace API también puede implementar seguridad, por ejemplo, verificar que el cliente está autorizado para realizar la solicitud.

- **Backends for Frontends:** Definir una puerta de enlace API separada para cada tipo de cliente.



Patrones de base de datos

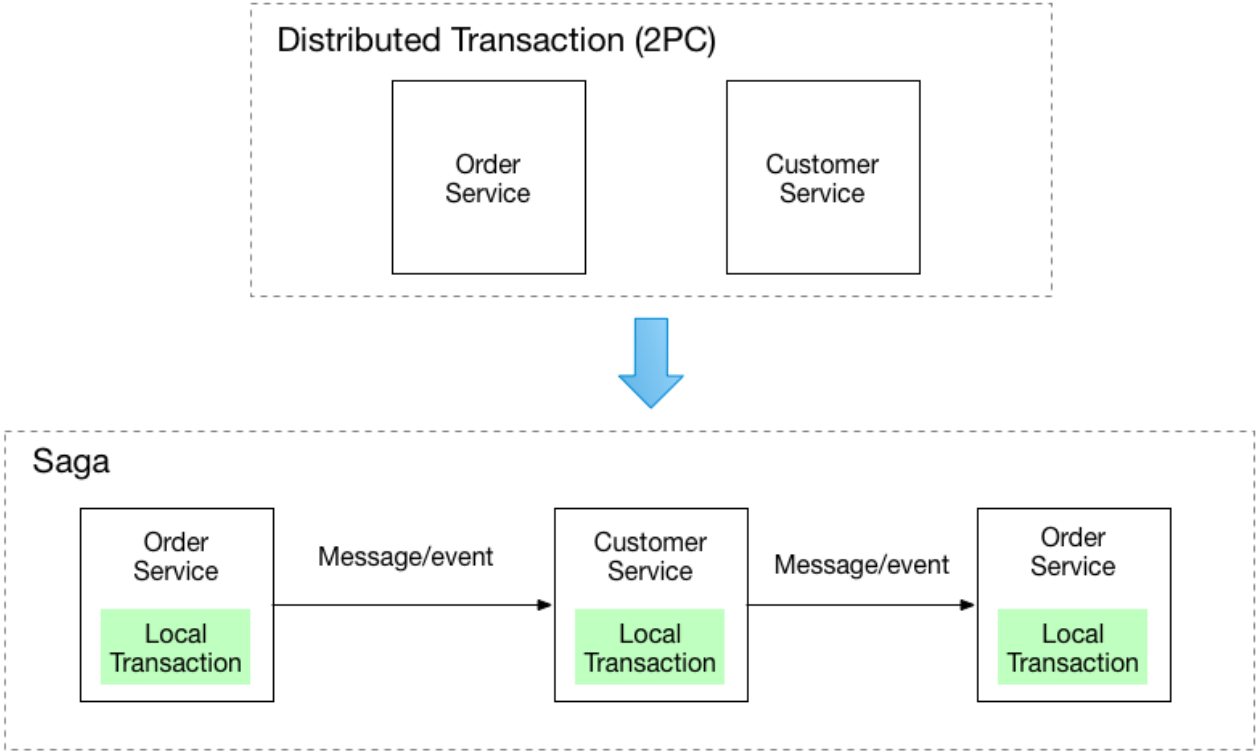
- **Base de datos por servicio**: La base de datos del servicio es efectivamente parte de la implementación de ese servicio. Otros servicios no pueden acceder directamente. Hay algunas formas diferentes de mantener la privacidad de los datos persistentes de un servicio. No es necesario que proporcione un servidor de base de datos para cada servicio. Por ejemplo, si está utilizando una base de datos relacional, las opciones son: Tablas privadas por servicio, esquemas por servicio. Las tablas privadas por servicio y el esquema por servicio tienen la sobrecarga más baja. El uso de un esquema por servicio es atractivo porque aclara la propiedad. Algunos servicios de alto rendimiento pueden necesitar su propio servidor de base de datos., etc.

Patrones de base de datos

- **Base de datos compartidas**: Utilizar una base de datos (única) compartida por varios servicios. Cada servicio accede libremente a los datos que pertenecen a otros servicios.

- **Saga:** Una saga es una secuencia de transacciones locales. Cada transacción local actualiza la base de datos y publica un mensaje o evento para activar la siguiente transacción local de la saga. Si una transacción local falla porque viola una regla comercial, la saga ejecuta una serie de transacciones compensatorias que deshacen los cambios realizados por las transacciones locales anteriores.

Patrones de base de datos



Patrones de base de datos

- **Composición de API**: Implementar una consulta definiendo un API Composer , que invoca los servicios que poseen los datos y realiza una unión en memoria de los resultados.

- **Segregación de responsabilidad de consultas de comandos (CQRS):** Definir una base de datos de vista, que es una réplica de solo lectura diseñada para admitir esa consulta. La aplicación mantiene la réplica actualizada al suscribirse a eventos de dominio publicados por el servicio que posee los datos.

Patrones de base de datos

- **Eventos de dominio**: Organizar la lógica empresarial de un servicio como una colección de agregados DDD que emiten eventos de dominio cuando se crean o actualizan. El servicio publica estos eventos de dominio para que puedan ser consumidos por otros servicios.

- **Event sourcing:** Se encarga de capturar todos los cambios que se pueden producir en nuestra aplicación como una secuencia de eventos. El código de la aplicación que genera los eventos está desacoplado de los sistemas que se suscriben a los eventos. La tienda de eventos generalmente publica estos eventos para que los consumidores puedan ser notificados y puedan manejarlos si es necesario.

Patrones de observabilidad

- **Log aggregation:** Utilizar un servicio de registro centralizado que agregue registros de cada instancia de servicio. Los usuarios pueden buscar y analizar los registros. Pueden configurar alertas que se activan cuando aparecen determinados mensajes en los registros.

- **Application metrics:** Instrumentar un servicio para recopilar estadísticas sobre operaciones individuales. Métricas agregadas en el servicio de métricas centralizadas, que proporciona informes y alertas.

Patrones de observabilidad

- **Distributed tracing:** Servicios de instrumentos con código que: Asigna a cada solicitud externa una identificación de solicitud externa única, pasa el ID de solicitud externa a todos los servicios que participan en el manejo de la solicitud, incluye la identificación de solicitud externa en todos los mensajes de registro, Registra información (por ejemplo, hora de inicio, hora de finalización) sobre las solicitudes y operaciones realizadas al manejar una solicitud externa en un servicio centralizado.

- **Health Check API:** Un servicio tiene un punto final de API de verificación de estado (por ejemplo, HTTP /health) que devuelve el estado del servicio.

Patrones de observabilidad

- **Exception tracking:** Informa todas las excepciones a un servicio de seguimiento de excepciones centralizado que agrega y rastrea las excepciones y notifica a los desarrolladores.

- **Audit logging:** Registra la actividad del usuario en una base de datos.

Patrones Cross-Outing

- **Configuración externalizada:** Externalizar o centralizar toda la configuración de la aplicación, incluidas las credenciales de la base de datos y la ubicación de la red. Al inicio, un servicio lee la configuración de una fuente externa, por ejemplo, variables de entorno del sistema operativo, etc.

- **Circuit Breaker:** Puede determinar que un determinado servicio no esta disponible al detectar que ha respondido con error de forma consecutiva un número determinado de veces.

Patrones Cross-Outing

- **Service Registry and Discovery:** La idea central de esta patron es que todos los servicios cuando inician, se registran ante una entidad llamada Service Registry, el cual lleva el control de todos los servicios activos. De tal forma que cuando nosotros queremos consumir un servicio, buscamos en el Service Registry las instancias disponibles

Comunicación entre Microservicios

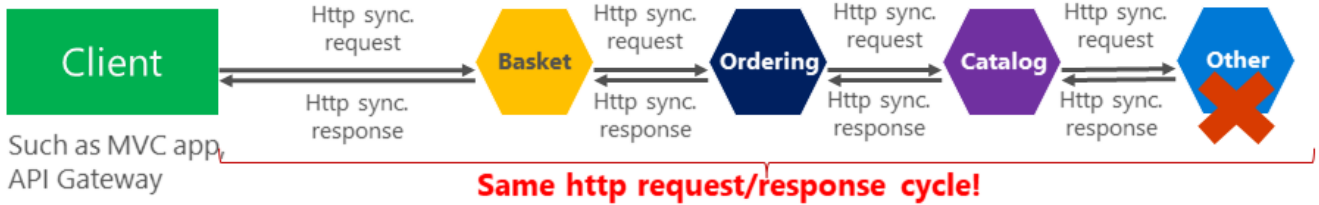
- Una aplicación o un software basado en microservicios es un sistema que, a su vez, se ejecuta en diferentes procesos, servicios y, la mayoría de las veces, también en varios servidores o hosts. Como lo habitual es que cada instancia de servicio funcione como un proceso diferenciado, los servicios tienen que actuar mediante un protocolo de comunicación entre procesos. En función de la naturaleza de cada servicio, estos protocolos pueden ser HTTP, AMQP o un protocolo binario TCP.

- En general, existen dos criterios para clasificar estos sistemas de comunicación:
  - **Por clase de protocolo:** sincrónico o asincrónico.
  - **Por número de receptores:** uno o varios.

Synchronous vs. async communication across microservices

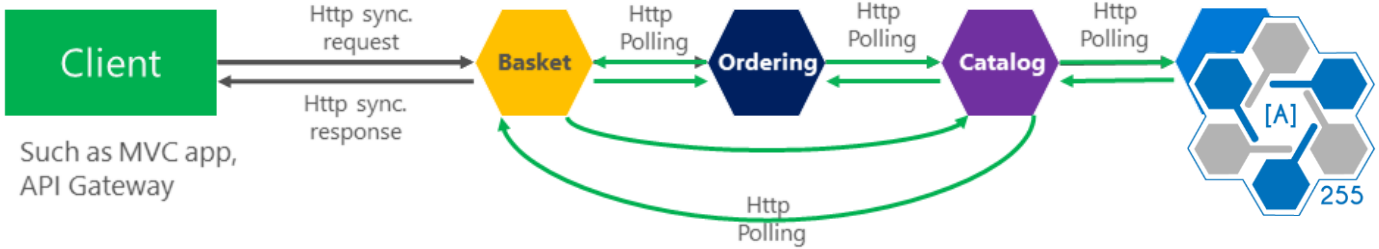
Anti-pattern

Synchronous
all request/response cycle



Asynchronous
Comm. across internal microservices
(EventBus: like **AMQP**)

"Asynchronous"
Comm. across internal microservices
(Polling: **Http**)



Complementando …

Los desarrolladores y arquitectos de software usan muchos modelos arquitectónicos. Los siguientes son algunos de ellos (se combinan estilos y modelos arquitectónicos):

- CRUD simple, de un nivel y una capa.
- Tradicional de N capas.
- Diseño controlado por dominios de N capas.
- Arquitectura limpia
- Segregación de responsabilidades de consultas de comandos (CQRS).
- Arquitectura controlada por eventos (EDA).

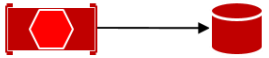
Complementando …

- También se pueden compilar microservicios con **muchas tecnologías y lenguajes**, como las API web de ASP.NET Core, NancyFx, ASP.NET Core SignalR (disponible con .NET Core 2), F#, Node.js, Python, Java, C++, GoLang y muchos más.

- Lo **importante** es que **ningún modelo o estilo arquitectónico** determinado, ni ninguna tecnología concreta, es **adecuado para todas las situaciones.**

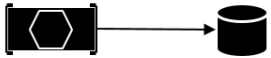
The Multi-Architectural-Patterns and polyglot microservices world

Microservice 1



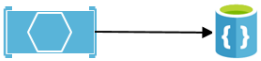
- **ASP.NET Core**
- Simple CRUD Design
- Entity Framework Core

Microservice 2



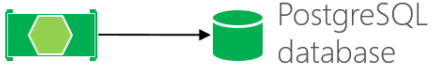
- **ASP.NET Core**
- DDD & CQRS patterns
- EF Core + Dapper

Microservice 3



- **ASP.NET Core**
- Queries projection
- DocDB/MongoDB API

Microservice 4

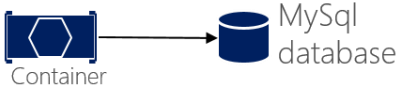


- **NancyFX (.NET Core)**
- Simple CRUD Design
- Massive

Microservice 5

- **ASP.NET Core**
- Simple CRUD Design
- Redis API

Microservice 6



- **Node.js**
- Simple CRUD Design

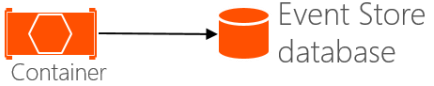
Microservice 7

- **Python**
- Simple CRUD Design

Microservice 8

- **Java**
- DDD patterns

Microservice 9



- **ASP.NET Core**
- Event Sourcing patterns
- Event Store API

Microservice 10

- **SignalR (.NET Core 2)**
- Hub for Real Time comm.

Microservice 11

- **F# .NET Core**
- i.e. Calculus focused

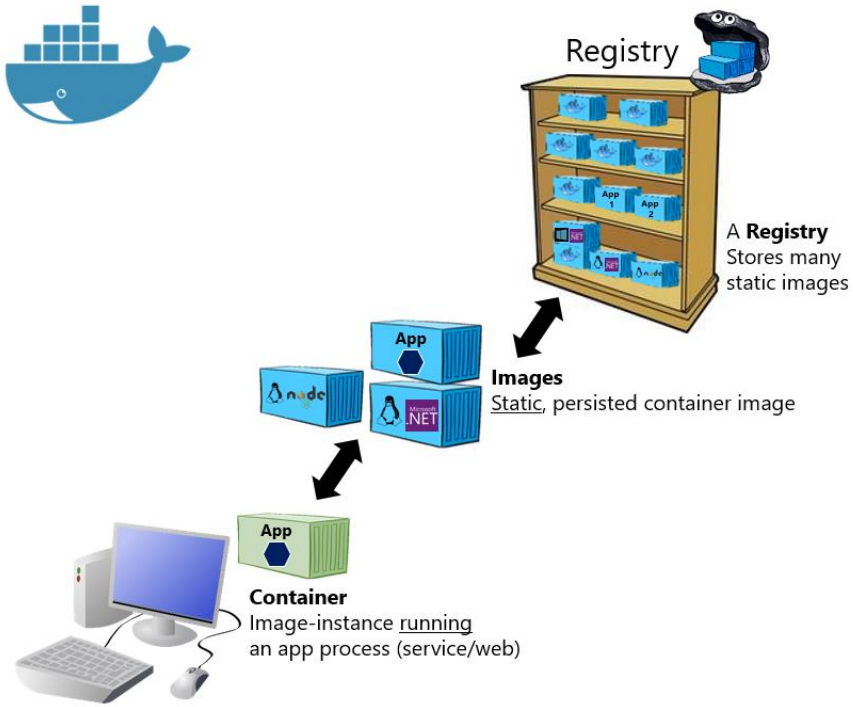
Microservice 10

- **GoLang**
- Stateless process

Docker

- Docker es una plataforma de software que le permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas **contenedores** que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución. Con Docker, puede implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que su código se ejecutará.

Taxonomía en Docker



A **Registry**
Stores many
static images

Images
Static, persisted container image

Hosted Docker Registry

Docker Trusted Registry on-prem.

On-premises
('n' private organizations)

Docker Hub Registry

Docker Trusted Registry on-cloud

Azure Container Registry

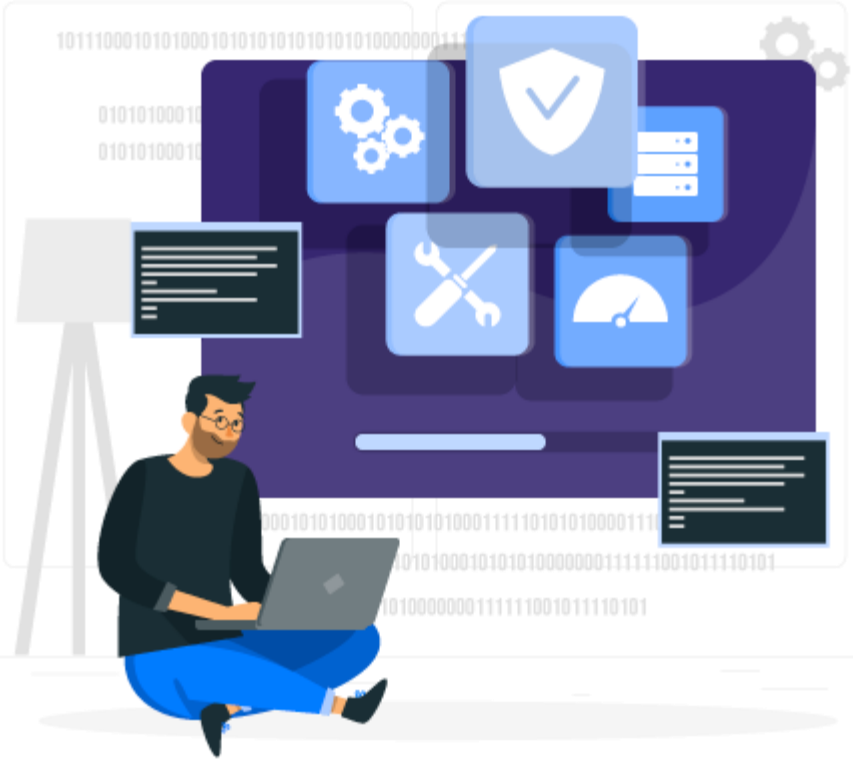
AWS Container Registry

Google Container Registry

Quay Registry

Other Cloud

Public Cloud
(specific vendors)



HANDS ON

¡Muchas gracias!