Microsoft .NET: Arquitectura para Aplicaciones .NET Contenerizadas

Víctor Iranzo

18 de junio de 2018

1. Introducción

La contenerización es un enfoque de desarrollo de software en el que un servicio, sus dependencias y su configuración se empaquetan juntos como una imagen de contenedor. Este enfoque facilita desplegar un servicio en distintos entornos sin apenas cambios. Además, actúan como una unidad que aísla un servicio de otros.

Entre los beneficios que ofrece este enfoque están el aislamiento, portabilidad, agilidad y escalabilidad.

Docker es un proyecto open-source para automatizar el despliegue de aplicaciones como contenedores independientes y autosuficientes, que se pueden ejecutar en la nube o en servidores internos. Las imágenes Docker se pueden ejecutar en Windows y Linux de forma nativa.

Comparando los contenedores con las máquinas virtuales, las seguras requieren más recursos que los primeros porque necesitan un sistema operativo completo. Los contenedores se ejecutan como procesos independientes y comparten el kernel del sistema operativo de la máquina donde se ejecutan. Esto hace que sean fáciles de desplegar y de arranque rápido, por lo que permiten una mayor densidad de servicios. Sin embargo, un contenedor tiene menos aislamiento que una máquina virtual. Un caso particular son los contenedores Hyper-V, donde cada contenedor se ejecuta en una máquina virtual optimizada.

1.1. Terminología Docker

- Imagen de contenedor: es un paquete con todas las dependencias e información necesaria para crear un contenedor. La imagen contiene

todas las dependencias así como toda la configuración de despliegue. Una imagen se puede derivar de otras imágenes base. Las imágenes permiten empaquetar un servicio para desplegarlo de forma confiable y reproducible. Son inmutables, es decir, no se pueden modificar después de crearlas.

- Contenedor: una instancia de una imagen que representa la ejecución de un servicio.
- Dockerfile: un fichero de texto que contiene instrucciones sobre cómo construir una imagen Docker.
- Compose: una herramienta de línea de comandos y un fichero en formato YAML con metadata, para definir y ejecutar aplicaciones de múltiples contenedores. Se define una aplicación basada en múltiples imágenes con uno o más ficheros .yml, que crea un contenedor en el host Docker por cada imagen.
- Orquestador: una herramienta que simplifica la gestión de clusters y hosts Docker. Un orquestador es responsable de ejecutar, distribuir, escalar y distribuir la carga de trabajo en una colección de nodos. También gestionan el descubrimiento de servicios y sus configuraciones.

2. .NET Core vs .NET Framework para contenedores Docker

.NET Core debe ser el framework preferido para crear aplicaciones .NET en contenedores porque es compatible con las plataformas Linux y Windows. Solo se debe emplear .NET Framework para un servicio contenerizado si este tiene fuertes dependencias con Windows y APIs, librerías o paquetes NuGets que no están disponibles en .NET Core.

3. Principios de diseño de contenedores

Cada contenedor representa un proceso único. Cuando se diseña una imagen, verá la definición de un ENTRYPOINT en el Dockerfile. Esto define el proceso cuya vida determinará la vida útil del contenedor. Cuando el proceso finaliza, también termina la vida del contenedor.

3.1. Arquitectura orientada a servicios (SOA)

En una arquitectura orientada a servicios una aplicación se descompone en múltiples servicios (por lo general, HTTP). Cada servicio tiene necesidades de escalabilidad y disponibilidad diferentes.

Los microservicios se derivan de SOA, pero SOA es diferente de la arquitectura de microservicios. Las características como grandes agentes centrales (central brokers), orquestadores centrales a nivel de organización y el Enterprise Service Bus (ESB) son típicos en SOA. Pero en la mayoría de los casos, estos son anti patrones en la comunidad de microservicios. De hecho, algunas personas argumentan que "la arquitectura de microservicio es SOA implementada correctamente".

3.2. Arquitectura de microservicios

Una arquitectura de microservicios es un enfoque para construir una aplicación de servidor como un conjunto de pequeños servicios. Cada servicio se ejecuta en su propio proceso y se comunica con otros utilizando protocolos como HTPP, WebSockets o AMQP. Cada microservio implementa una funcionalidad dentro de los límites de cierto contexto. Cada microservicio se debe poder desarrollar y desplegar de forma independiente. La tecnología empleada en cada microservicio puede ser diferente. Además, cada microservicio debe tener su modelo de datos y lógica de dominio relacionados.

Aunque por su nombre parezca que lo más relevante sea el tamaño del servicio, lo más importante debe ser crear servicios poco acoplados y con límites bien definidos.

Entre los beneficios de una arquitectura basada en microservicios:

- Agilidad: facilitan el mantenimiento de sistemas complejos, grandes y que requieren una alta escalabilidad gracias al despliegue autónomo de cada servicio. Además facilitan los cambios, que se encuentran más localizados, y permiten iterar más rápido en desarrollos ágiles de servicios.
- Escalabilidad: en lugar de escalar todo el monolito como una única unidad, se pueden escalar microservicios específicos de forma individual. Esto permite un mejor aprovechamiento de los recursos.
- Facilita las prácticas de integración y entrega continuas (CI/CD): las nuevas funcionalidades se entregan más rápido y es más sencillo probar y ejecutar servicios de forma aislada y en diferentes entornos.

4. Contenerizando aplicaciones monolíticas

Un único servicio monolítico ejecutándose en un contenedor no permite escalar solo aquellos componentes de la aplicación que son realmente un cuello de botella. Aunque se ejecute en un contenedor que se puede replicar, el escalado es forma gruesa porque se replica toda la aplicación. El enfoque monolítico se puede emplear inicialmente porque es más fácil que un enfoque basado en microservicios.