**.NET PARA LOS QUE NO TENEMOS TARJETA DE CREDITO**



**-** **Hecho por:** German Uriel Evangelista Martinez.

*- Una guía de mi para todos los que no tenemos una millonada en el banco.*

**Contenido**

[INTRODUCCION 3](#_Toc200233362)

[1. Estrategias base para no gastar 4](#_Toc200233363)

[1.1 Usa las versiones gratuitas oficiales 4](#_Toc200233364)

[1.2 Sácale provecho a licencias estudiantiles y promociones 4](#_Toc200233365)

[1.3 Planea desde el principio sin dependencias caras 4](#_Toc200233366)

[1.4 Crea tu propio “Azure” local 5](#_Toc200233367)

[1.5 Nunca subestimes el poder del open source 5](#_Toc200233368)

[2. Stack local potente (sin tarjeta) 6](#_Toc200233369)

[2.1 Aspire (.NET 8+) 6](#_Toc200233370)

[🐳 2.2 Docker para servicios críticos 7](#_Toc200233371)

[2.3 Simulación de email y eventos 7](#_Toc200233372)

[2.5 Pruebas completas sin la nube 8](#_Toc200233373)

[Ejemplo de uso del stack de .NET sin gastar ni un peso. 9](#_Toc200233374)

[Paso 1: Crear la solución 9](#_Toc200233375)

[Paso 2: Crear proyectos base 9](#_Toc200233376)

[Paso 3: Configurar AppHost para orquestar 10](#_Toc200233377)

[Paso 4: Conectar la API a Redis y SQL Server 10](#_Toc200233378)

[🐳 Paso 5: Agregar servicios externos vía Docker (opcional, si no usas Aspire local) 11](#_Toc200233379)

[Paso 6: Levantar el stack 11](#_Toc200233380)

[Resultado esperado: 12](#_Toc200233381)

[4. Alternativas gratuitas reales 13](#_Toc200233382)

[4.1 Autenticación sin Azure AD B2C 13](#_Toc200233383)

[4.2 Hosting de aplicaciones web/API 13](#_Toc200233384)

[4.3 Base de datos en la nube 14](#_Toc200233385)

[4.4 Almacenamiento y archivos 14](#_Toc200233386)

[4.5 Otras joyitas 14](#_Toc200233387)

[5. Simulación de servicios Azure sin Azure 15](#_Toc200233388)

[5.1 Blob Storage → Azurite 15](#_Toc200233389)

[5.2 Azure Service Bus → RabbitMQ / NATS 15](#_Toc200233390)

[5.3 Azure Cache (Redis) → Redis oficial 16](#_Toc200233391)

[5.4 Application Insights → Jaeger + OpenTelemetry 16](#_Toc200233392)

[5.5 Azure App Configuration → appsettings.json + Aspire 16](#_Toc200233393)

[5.6 Azure Communication Services → MailHog (SMTP local) 17](#_Toc200233394)

[6. Consejos de arquitectura para ahorrar 18](#_Toc200233395)

[6.1 Evita el overengineering 18](#_Toc200233396)

[6.2 Diseña desacoplado desde el inicio 18](#_Toc200233397)

[7. Herramientas útiles y gratuitas 20](#_Toc200233398)

[7.1 Bases de datos y visualización 20](#_Toc200233399)

[7.3 CI/CD y automatización 20](#_Toc200233400)

[7.4 Observabilidad y monitoreo 20](#_Toc200233401)

[7.5 Desarrollo general 21](#_Toc200233402)

[8. Casos de ejemplo 22](#_Toc200233403)

[8.1 Desarrollo de una API educativa (como ACC) 22](#_Toc200233404)

[8.2 MVP para un sistema de reservas 23](#_Toc200233405)

[8.3 Microservicio de reportes y facturación 23](#_Toc200233406)

[8.4 Pruebas de rendimiento y arquitectura sin gastar 23](#_Toc200233407)

[9. Cierre 25](#_Toc200233408)

# INTRODUCCION

El ecosistema .NET ha evolucionado hasta convertirse en una de las plataformas más robustas, versátiles y productivas para desarrollar software moderno. Desde aplicaciones web hasta servicios en la nube, pasando por inteligencia artificial, APIs, videojuegos, dispositivos móviles y más. Si sabes usar .NET, puedes construir prácticamente cualquier cosa.

Sin embargo, hay una verdad incómoda: lo más potente del ecosistema muchas veces está detrás de una tarjeta de crédito.

Azure, la nube estrella de Microsoft, es un ejemplo claro. Tiene servicios impresionantes como Azure Functions, Service Bus, Cosmos DB, Application Insights o Azure SQL, pero usarlos de forma sostenida —incluso en proyectos personales o educativos— puede volverse impagable para quienes no tienen acceso a créditos, becas o presupuestos empresariales.

¿Significa esto que estamos condenados a desarrollar solo con lo básico?  
**No**.  
Significa que debemos ser más inteligentes.

Esta guía está pensada para los desarrolladores que quieren lo mejor de .NET sin vaciar la cartera. Aquí no vas a encontrar consejos como “usa menos recursos” o “limita tu cuenta gratuita”.

No. Te enseñare a cómo **emular, simular o reemplazar la mayoría de los servicios premium de Azure y .NET Enterprise con herramientas gratuitas, locales o de código abierto**, sin perder el potencial de diseño ni sacrificar escalabilidad futura.

Porque construir cosas grandes sin pagar no es un truco, es un arte.  
Y estás a punto de dominarlo.

# 1. Estrategias base para no gastar

## 1.1 Usa las versiones gratuitas oficiales

* **Visual Studio Community**: gratis para estudiantes, proyectos personales y pequeños equipos.
* **Visual Studio Code**: ligero, extensible, y suficiente para la mayoría de los proyectos .NET modernos.
* **JetBrains Rider (educativo)**: si eres estudiante, puedes pedir la versión completa gratis.

## 1.2 Sácale provecho a licencias estudiantiles y promociones

* **Azure for Students**: 100 USD en crédito + acceso a muchos servicios sin tarjeta de crédito.
* **GitHub Student Developer Pack**: incluye licencias gratuitas de herramientas premium, servicios en la nube y más.
* **Microsoft Learn for Students**: cursos, recursos y entornos de práctica sin costo.

## 1.3 Planea desde el principio sin dependencias caras

Diseñar sin pensar en el costo de infraestructura es un error común. Algunas recomendaciones:

* No uses Azure SQL desde el inicio si puedes usar **SQL Server LocalDB** o en contenedor.
* No uses Application Insights solo “por tener métricas”, si puedes usar **OpenTelemetry con un stack local**.
* No uses Azure Storage si puedes levantar **Azurite**, o usar almacenamiento local para desarrollo.

## 1.4 Crea tu propio “Azure” local

Tu PC puede ser tu nube si sabes cómo orquestar, más adelante lo profundizo más, pero por ahora ten en mente que es posible:

* Usar **Docker** para montar Redis, RabbitMQ, SQL Server y más.
* Usar **Aspire** para simular despliegue orquestado, logging, métricas y configuración centralizada.
* Usar herramientas como **Mailhog**, **Azurite**, o **PostgreSQL en contenedor** para reemplazar servicios de producción.

## 1.5 Nunca subestimes el poder del open source

El ecosistema .NET es más abierto que nunca:

* Muchas bibliotecas de Azure tienen versiones auto-hosted.
* La comunidad crea alternativas gratuitas para casi todo.
* Y si no existe, puedes simularlo tú con un poco de código y creatividad.

# 2. Stack local potente (sin tarjeta)

¿Quieres un entorno completo tipo Azure pero sin costo? Puedes tenerlo. Gracias a herramientas como **Docker** y el nuevo **.NET Aspire**, es posible simular un ecosistema profesional directamente en tu máquina de desarrollo, sin comprometer escalabilidad ni estructura.

Aquí te muestro cómo montar un stack local que puede emular lo esencial de Azure sin pagar un peso.

## 2.1 Aspire (.NET 8+)

Aspire es un stack de desarrollo moderno para .NET que te permite levantar y orquestar múltiples servicios y aplicaciones desde un solo punto. Lo puedes ver como un mini Azure local:

* Usa AppHost como orquestador.
* Define proyectos, servicios, conexiones, y dependencias en segundos.
* Permite incluir Redis, PostgreSQL, SQL Server, Jaeger (tracing), etc.

**Ventajas:**

* Te da observabilidad (dashboards, tracing, logs) sin Application Insights.
* Maneja secretos, configuración compartida, health checks y más.
* Es modular y muy productivo, incluso en equipos grandes.

## 🐳 2.2 Docker para servicios críticos

Docker es la clave para simular servicios de Azure localmente. Estos son los más comunes que puedes levantar con un simple docker-compose.yml o desde Aspire:

|  |  |
| --- | --- |
| **Servicio en Azure** | **Alternativa local con Docker** |
| Azure SQL Database | SQL Server / PostgreSQL en contenedor |
| Azure Redis Cache | Redis (oficial image) |
| Azure Blob Storage | Azurite (simulación local de Blobs) |
| Azure Service Bus | RabbitMQ / NATS |
| Azure Functions | Worker services + Quartz.NET |
| Azure Monitor | Jaeger + OpenTelemetry + Zipkin |
| Azure App Configuration | appsettings.json + Aspire |

- Todos estos servicios se pueden levantar con imágenes gratuitas y usar desde tus proyectos como si estuvieras en la nube.

## 2.3 Simulación de email y eventos

* **MailHog**: un servidor SMTP local para capturar correos. Ideal para probar funcionalidades de registro y recuperación sin enviar correos reales.
* **RabbitMQ + Management UI**: simula colas, eventos y pub/sub.
* **Azurite**: emula Azure Blob, Queue y Table Storage.

**2.4 Observabilidad sin Azure Monitor**

* **OpenTelemetry** (integrado en Aspire): permite trazas distribuidas sin costo.
* **Jaeger o Zipkin**: viewers de trazas que puedes montar como contenedores.
* **Grafana + Prometheus**: dashboard local para métricas avanzadas.

## 2.5 Pruebas completas sin la nube

Puedes correr pruebas de integración contra estos servicios locales, asegurando que cuando llegues a producción, simplemente cambies la cadena de conexión y estés listo para escalar.

**Extra: Todo esto sin exponer datos ni gastar recursos de red.**

* Tus datos no salen de tu equipo.
* Puedes trabajar sin conexión a internet.
* Ideal para entornos educativos, prototipos o desarrollo offline.

# Ejemplo de uso del stack de .NET sin gastar ni un peso.

Primero hay que definir una estructura de carpetas, para tener un ecosistema mas organizado, te recomiendo la siguiente en la imagen:

Texto

El contenido generado por IA puede ser incorrecto.

A continuación, los pasos:

## Paso 1: Crear la solución

Texto

El contenido generado por IA puede ser incorrecto.

## Paso 2: Crear proyectos base

Texto

El contenido generado por IA puede ser incorrecto.

Después de crearlos deberás agregar los proyectos a la solución:

Texto

El contenido generado por IA puede ser incorrecto.

## Paso 3: Configurar AppHost para orquestar

Dentro del proyecto Aspire de la solución, encontraras un archivo de código llamado Program.cs, este contiene la información sobre cómo están configuradas algunas cosas en tu proyecto aspire, en este caso(si es que seguiste el tutorial de manera exacta), seria en:  
**MyAspireDemo.AppHost/Program.cs:**

**Texto

El contenido generado por IA puede ser incorrecto.**

## Paso 4: Conectar la API a Redis y SQL Server

En appsettings.json del proyecto Api, no necesitas hardcodear nada. Usamos IConfiguration:  
Texto

El contenido generado por IA puede ser incorrecto.

En el código de Api.csproj agrega el siguiente bloque de código siguiente:

Texto

El contenido generado por IA puede ser incorrecto.

**Nota:** Asegúrate de ponerlo en el sitio correspondiente, ya que los archivos de proyecto suelen romperse si no se les tiene cuidado.

## 🐳 Paso 5: Agregar servicios externos vía Docker (opcional, si no usas Aspire local)

Texto

El contenido generado por IA puede ser incorrecto.

## Paso 6: Levantar el stack

Si estas usando aspire directamente ejecuta este comando en la terminal de la solución:  


( si no sabes cual es la terminal de la solución haz clic derecho en la solución, y da en “abrir en terminal” ).

Una alternativa si prefieres levantar sin aspire y prefieres Docker de manera directa, simplemente ejecuta este comando en la terminal:



# Resultado esperado:

Tienes un **entorno de desarrollo profesional en tu máquina** que simula lo siguiente:

|  |  |
| --- | --- |
| **Azure Real** | **Alternativa Local Aquí** |
| Azure SQL | SQL Server en contenedor |
| Azure Redis Cache | Redis local |
| Application Insights | Jaeger + Aspire observabilidad |
| Azure App Services | dotnet run o IIS Express local |

**Nota:**   
algunas cosas podrían estar desactualizadas, espero que no la verdad, pero aspire y .NET evolucionan rápido así que verifica las versiones, este manual es en .NET 8. Con aspire 9.

# 4. Alternativas gratuitas reales

Aunque Aspire y Docker te permiten montar una infraestructura potente en tu máquina, hay ocasiones donde **sí necesitas servicios en la nube accesibles desde internet** (como para desplegar, autenticar usuarios reales o almacenar datos). La buena noticia es que **existen servicios gratuitos y potentes que puedes aprovechar** sin necesidad de una tarjeta de crédito.

Aquí van algunas alternativas confiables:

## 4.1 Autenticación sin Azure AD B2C

|  |  |  |
| --- | --- | --- |
| **Servicio** | **Descripción breve** | **Gratis ¿hasta que limite?** |
| **Firebase Auth** | Fácil de integrar en Blazor, WebAPI o apps móviles. | Tienes Usuarios ilimitados y reglas gratis |
| **Supabase Auth** | Alternativa open source a Firebase, con roles y JWT. | Un limite de 500 usuarios activos/mes |
| **Auth0 (plan Dev)** | Integración avanzada con C#, pero limitado en el plan free. | Un límite de 7,000 usuarios activos/mes |
| **Clerk.dev** | Ideal para proyectos modernos (como Next.js/SPA). | Un límite de 5,000 usuarios activos/mes |

## 4.2 Hosting de aplicaciones web/API

|  |  |  |
| --- | --- | --- |
| **Plataforma** | **Características** | **El Plan gratuito incluye** |
| **Render** | Ideal para apps .NET, PostgreSQL, Redis, WebSockets. | 750h/mes (equiv. a 1 app fulltime) |
| **Railway.app** | Despliegue simple con base de datos y secretos incluidos. | 500h/mes + 1GB RAM |
| **Glitch** | Perfecto para prototipos rápidos o APIs pequeñas. | Siempre gratis (con limitaciones) |
| **Vercel** | Más orientado a frontend, pero sirve para Blazor WASM estático. | 100 GB bandwidth / 12 funciones |

## 4.3 Base de datos en la nube

|  |  |  |
| --- | --- | --- |
| **DBaaS** | **Tipo** | **Plan gratuito** |
| **PlanetScale** | MySQL sin servidor, autoescalable. | 5 GB + 1 producción gratuita |
| **Supabase** | PostgreSQL full con autenticación, storage y REST. | 500 MB + 2 proyectos |
| **ElephantSQL** | PostgreSQL clásico. | 20 MB (Tiny Turtle plan) |
| **MongoDB Atlas** | MongoDB (NoSQL) | 512 MB en clúster compartido |

## 4.4 Almacenamiento y archivos

|  |  |  |
| --- | --- | --- |
| **Opción** | **Descripción** | **Gratis hasta...** |
| **Cloudinary** | Imágenes/videos optimizados | 25 créditos/día (~300 MB/mes) |
| **Firebase Storage** | Archivos y blobs | 5 GB + 1 GB descarga/día |
| **Supabase Storage** | Directamente ligado a su plataforma | 1 GB incluidos |

## 4.5 Otras joyitas

* **UptimeRobot**: monitoreo gratuito de tus servicios (hasta 50 como límite).
* **Freenom / DuckDNS**: dominios gratuitos y dinámicos.
* **Tailscale**: acceso seguro entre dispositivos sin exponer puertos.

**Consejo realista:** Puedes tener tu aplicación completa con API, autenticación, base de datos y almacenamiento completamente online **sin pagar un peso**, si eliges bien las herramientas.

# 5. Simulación de servicios Azure sin Azure

Uno de los mayores obstáculos al migrar desde un entorno gratuito o local a Azure es la dependencia de sus servicios gestionados. Pero ¿y si pudieras **simularlos con herramientas open source o contenedores** hasta que realmente necesites pagar?

Aquí te muestro cómo recrear la mayoría de los **servicios clave de Azure** sin pagar ni un centavo.

## 5.1 Blob Storage → Azurite

¿Qué es Azurite?

- Es el emulador oficial de Microsoft para Azure Blob, Queue y Table Storage.

Para usarlo ejecuta este comando en la terminal que tengas con acceso a Docker:



Para integrarlo en .NET agrega en la cadena de conexión que tengas con la base de datos lo siguiente:

“ UseDevelopmentStorage=true; ” evidentemente sin las comillas.

## 5.2 Azure Service Bus → RabbitMQ / NATS

¿Qué se hace exactamente con RabbitMQ en este caso?

* Se simulan colas y modelos de Publicación/Suscripción con Exchange y bindings
* Un Amin web en http://localhost:15672.

Ejecutando la siguiente línea en la terminal:

**docker run -d --hostname rabbit --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-management**

**Alternativa ligera:** NATS, el cual es muy conveniente si no necesitas demasiado por el momento, puedes simular de manera rápida el modelo Publicacion/Suscripcion, además de que se cuenta con una integración más rápida con microservicios de .NET.

## 5.3 Azure Cache (Redis) → Redis oficial

Para levantar un contenedor con Redis, el cual por cierto es oficial, ejecuta este comando en la terminal:

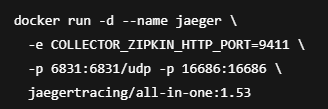


Para visualizar de mejor manera algunos aspectos de Redis, te recomiendo **RedisInsight**, este te servirá para tus keys, TTLs y comandos.

## 5.4 Application Insights → Jaeger + OpenTelemetry

Usa Opentelemetry y Jaeger para tener mejores métricas de tus servicios y aplicaciones de .NET, para cuando busques un rastreo distribuido.

Para levantar Jaeger con docker ejecuta este comando en la terminal:



* **Nota:** aspire ya configura por si solo opentelemetry en tus aplicaciones, solo agrega los servicedefaults en los program.cs de tus proyectos relevantes, como APIs, o aplicaciones relacionadas al servidor a las cuales veas necesario seguirles el rastro.

## 5.5 Azure App Configuration → appsettings.json + Aspire

* Usa appsettings.{Environment}.json + secretos en desarrollo.
* Con Aspire, puedes centralizar la configuración entre proyectos y servicios fácilmente.

## 5.6 Azure Communication Services → MailHog (SMTP local)

Simula envío de correos sin enviar nada.

* Para levantar el servicio ejecuta este comando en la terminal:



Y para acceder a tu inbox de prueba ve a http://localhost:8025.

**Recapitulando, los servicios de Azure y sus alternativas:**

|  |  |
| --- | --- |
| **Servicio Azure** | **Alternativa local / gratuita** |
| Azure Blob Storage | Azurite |
| Azure Service Bus | RabbitMQ o NATS |
| Azure Redis Cache | Redis + RedisInsight |
| Azure Monitor | Jaeger + OpenTelemetry |
| Azure App Configuration | Aspire + appsettings.json |
| Azure Email / Notify | MailHog |

# 6. Consejos de arquitectura para ahorrar

Desarrollar sin tarjeta no es sinónimo de hacer las cosas a medias. Es todo lo contrario: significa **diseñar de forma inteligente, modular y sostenible**, para que tu aplicación funcione bien hoy sin limitar su futuro.

Aquí van algunos consejos clave para lograrlo:

## 6.1 Evita el overengineering

* No apliques patrones complejos “por si acaso”.
* Si estás en local o prototipo, **no necesitas CQRS, DDD, Event Sourcing y microservicios a la vez**.
* En lugar de escalar horizontalmente desde el inicio, **escala tu código, tu lógica y tus decisiones**.
* *Hazlo funcionar, luego hazlo elegante, y después hazlo escalable.*

## 6.2 Diseña desacoplado desde el inicio

* Usa **interfaces (DI)** para todo servicio externo (email, storage, cache).
* Asegúrate de poder cambiar Redis por MemoryCache, o Azure SQL por SQLite, **sin tocar tu lógica de negocio**.
* Esto te permite probar barato hoy y conectar con Azure el día que tengas presupuesto.

**6.3 Usa CQRS solo cuando se justifique**

* CQRS = separación entre comandos (modificar) y queries (leer).
* Útil en aplicaciones con **alta carga de lectura y escritura** o **muchos consumidores simultáneos**.
* Pero si tu aplicación tiene bajo volumen, podría duplicar el trabajo innecesariamente.
* *Simplifica primero. Refactoriza cuando lo necesites.*

**6.4 Cuidado con los logs**

* Usar **Application Insights** desde el inicio puede parecer tentador, pero no es gratis.
* En local, usa **Serilog + consola + archivos**.
* Si quieres monitoreo real, **OpenTelemetry + Jaeger o Zipkin** te dan todo lo necesario sin costo.

**6.5 Usa entornos separados por perfil**

* Usa appsettings.Development.json, appsettings.Production.json, etc.
* Aplica la regla: **nunca apuntes a servicios en la nube desde desarrollo**.
* Asegúrate de que todos los entornos se comporten igual, solo cambie la infraestructura

**6.6 Tu código debe ignorar la infraestructura**

* No pongas lógica específica de Azure en tus controladores o servicios si no es necesario, recuerda que tu código no debe depender al 100% de algo nunca, menos si es en una capa lógica que podrías hacer sin mucho esfuerzo.
* Usa patrones como **Adapter** o **Proxy** para abstraer lo que necesites.
* El objetivo: *tu aplicación debería poder correr en tu PC o en Azure sin cambiar nada* más que la configuración para el escenario.

# 7. Herramientas útiles y gratuitas

El stack de desarrollo .NET moderno se puede potenciar muchísimo con herramientas gratuitas, open source o con licencias generosas. Aquí te presento una lista curada que alguna ventajita sin tanto cache:

## 7.1 Bases de datos y visualización

|  |  |
| --- | --- |
| **Herramienta** | **¿Para qué sirve?** |
| **DBeaver** | Administra y explora bases de datos SQL y NoSQL. |
| **pgAdmin** | Interfaz para bases de datos PostgreSQL. |
| **RedisInsight** | Interfaz visual para monitorear Redis como un pro. |

## 7.3 CI/CD y automatización

|  |  |
| --- | --- |
| **Herramienta** | **¿Para qué sirve?** |
| **GitHub Actions** | Automatiza tests, builds y deploys desde tus repos. |
| **Azure DevOps Pipelines** | Muy completo, aunque más pesado. Plan gratuito útil. |
| **Taskfile.dev** | Alternativa tipo Makefile, pero multiplataforma. |

## 7.4 Observabilidad y monitoreo

|  |  |
| --- | --- |
| **Herramienta** | **¿Para qué sirve?** |
| **Grafana** | Visualización de métricas y logs personalizados. |
| **Prometheus** | Sistema de métricas y scraping para servicios. |
| **Jaeger** | Trazas distribuidas para detectar cuellos de botella. |

## 7.5 Desarrollo general

|  |  |
| --- | --- |
| **Herramienta** | **¿Para qué sirve?** |
| **Docker Desktop** | Orquestación local de contenedores y redes. |
| **Visual Studio Community** | IDE oficial para .NET con todo lo necesario. |
| **VS Code** | Editor liviano pero extensible con C#, Docker y más. |
| **Rider (educativo)** | IDE avanzado de JetBrains si tienes acceso estudiantil. |

Por si usas Visual estudio code, te recomiendo las siguientes extensiones para tener tu entorno bien preparado:

* C# Dev Kit
* Docker
* GitLens
* Thunder Client
* SQLTools

Como consejo te recomiendo combinar estas herramientas con buenas prácticas, ya que muchas veces las olvidamos al tener un entorno de desarrollo más potente.

# 8. Casos de ejemplo

Para cerrar el círculo, aquí van algunos **casos reales y aplicables** donde puedes utilizar todo lo que vimos en esta guía. Estas situaciones representan escenarios comunes tanto en proyectos personales como escolares o incluso prototipos profesionales.

## 8.1 Desarrollo de una API educativa (como ACC)

**Escenario:**  
Quieres crear una plataforma educativa como **Aprendiendo C# con Charp (ACC)**, por si no lo sabias, un proyecto mío el cual lleve a la competencia nacional de prototipos en México(que presumido) con autenticación, tareas y contenido interactivo.

**Stack sugerido sin tarjeta:**

* **API** en ASP.NET Core.
* **SQL Server en Docker** (datos de usuarios, lecciones, progreso).
* **Redis en Docker** (caché de respuestas rápidas, sesiones).
* **MailHog** para correos de recuperación.
* **Swagger** y **Thunder Client** para probar endpoints.
* **GitHub Actions** para CI/CD.
* *Curiosamente, este stack es muy similar al que verdaderamente se uso para la creación de ACC, así que puedo confirmar que sirve muy bien.*

## 8.2 MVP para un sistema de reservas

**Escenario:**  
Debes presentar un prototipo funcional de una app de reservas sin pagar hosting ni servicios.

**Stack sugerido:**

* **Blazor WebAssembly + .NET API.**
* **Render.com** para desplegar la API sin costo.
* **PlanetScale** para tener base de datos MySQL gratuita.
* **Firebase Auth** para login simple sin manejar contraseñas.
* **GitHub Pages o Vercel** para el frontend Blazor.

## 8.3 Microservicio de reportes y facturación

**Escenario:**  
Simulas un sistema donde distintos microservicios se comunican para generar reportes o facturas.

**Stack local:**

* **RabbitMQ en Docker** como bus de eventos.
* **Worker Services** con IHostedService en .NET.
* **Jaeger + OpenTelemetry** para trazas distribuidas.
* **DBeaver** para revisar la base de datos local.

## 8.4 Pruebas de rendimiento y arquitectura sin gastar

**Escenario:**  
Estás evaluando qué arquitectura escalaría mejor, pero no puedes pagar Azure App Services.

**Solución gratuita:**

* Orquestación local con **Aspire**.
* Logs, trazas y health checks integrados.
* Pruebas con **k6**, **wrk**, o herramientas de stress testing.
* Visualización con **Grafana** y métricas de Prometheus.

En fin.

Todos estos escenarios **se pueden montar sin gastar un peso**, sin comprometer calidad ni arquitectura. Y cuando llegue el momento de escalar a producción, solo cambias las conexiones y despliegas en Azure (o cualquier nube).

# 9. Cierre

Decir que .NET es caro es un error. Lo que **cuesta dinero no es la tecnología, sino las decisiones mal planeadas**. En realidad, .NET es uno de los ecosistemas más poderosos y accesibles del mundo si sabes cómo moverte.

En esta guía te demostré que se puede:

* Construir APIs robustas.
* Simular servicios empresariales como Azure SQL, Redis o Service Bus.
* Desplegar en la nube.
* Tener trazabilidad, monitoreo, métricas y CI/CD.

Y hacerlo todo **sin una tarjeta de crédito**.

Lo único que necesitas es **criterio técnico, algo de creatividad y muchas ganas de construir**. Evidentemente no pude cubrir cada detalle en esta pequeña guía, pero es una base para que te adentres más, y veas que .NET no es solo para el sector empresarial.

Porque al final del día, ser desarrollador no es tener acceso a lo más caro, sino **hacer que cada línea de código cuente, incluso cuando el presupuesto es cero**.

Así que sigue creando, sigue optimizando, y nunca subestimes el poder de una buena idea… ni el de una cartera vacía bien organizada 😉