Sesión 01

Introducción y evolución a Net 9

Instructor:

ERICK ARÓSTEGUI

earostegui@galaxy.edu.pe

GALAXY
TRAINING

ÍNDICE

GALAXY
TRAINING

01

Historia y evolución de NET

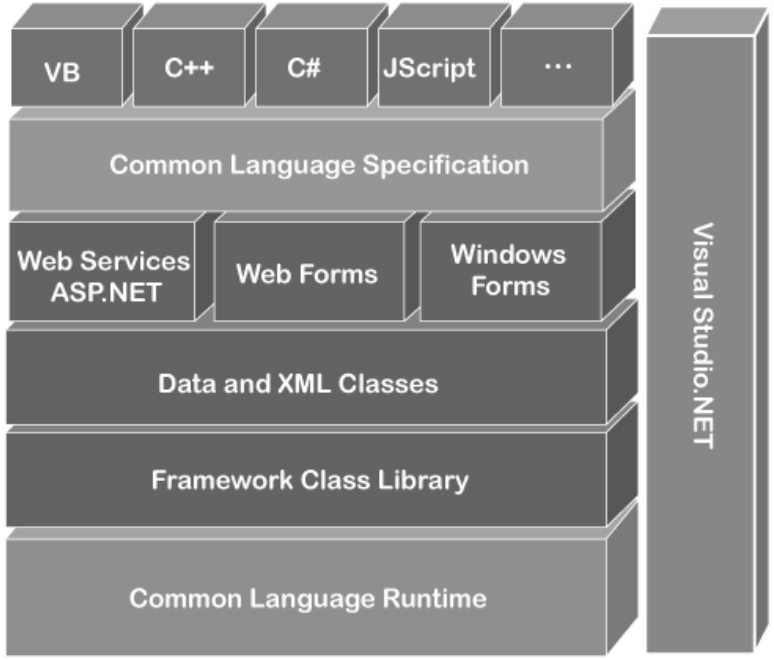
La historia detrás de .NET

Microsoft comenzó a trabajar en el **framework .NET a finales de los 90**. La idea era crear una **plataforma basada en el llamado código administrado**, código que se puede ejecutar bajo un entorno de ejecución.

Esto era necesario para mejorar la experiencia de desarrollo y aliviar a los ingenieros del manejo de las operaciones de seguridad, la administración activa de la memoria y otros esfuerzos de bajo nivel con los que los desarrolladores de C / C ++ tenían que molestarse.

GALAXY TRAINING

.NET Framework

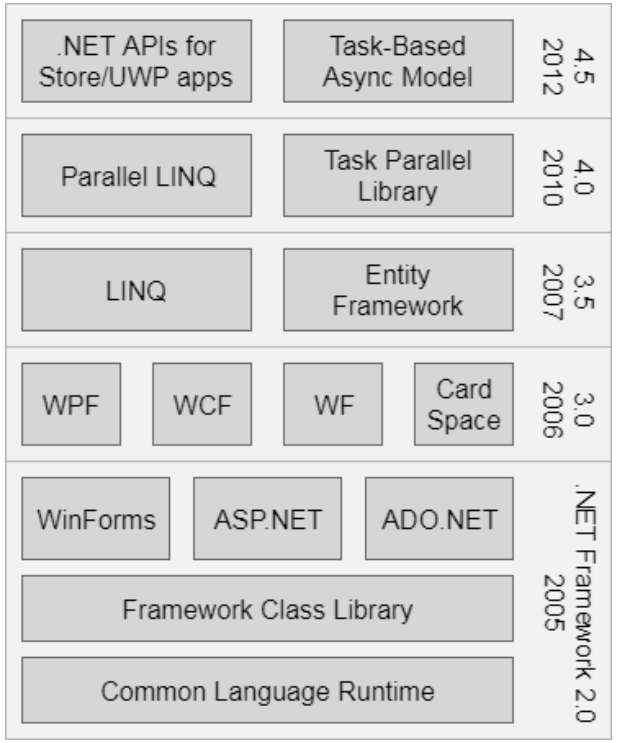


La primera versión de .NET **Framework en 2002** introdujo C#, un lenguaje para escribir código administrado que tenía un diseño similar a C++.

El marco en sí estaba dirigido a computadoras y servidores basados en Windows. **Tenía WinForms, una biblioteca GUI para aplicaciones de escritorio; ASP.NET, un framework para Web; y ADO.NET para el acceso a los datos. Common Language Runtime (CLR)** controló todos estos elementos para compilar y ejecutar código administrado.

GALAXY TRAINING

.NET Framework



Para unir varias funciones, **.NET ofrecía un Framework Class Library (FCL)** que incluía la **Base Class Library (BCL)**, la biblioteca de red, una biblioteca numérica y otras.

Desde entonces, el framework ha sufrido múltiples iteraciones que abarcan actualizaciones en tiempo de ejecución, nuevos **sistemas gráficos de escritorio (WPF),** API para aplicaciones orientadas a **servicios (WCF)** y más.

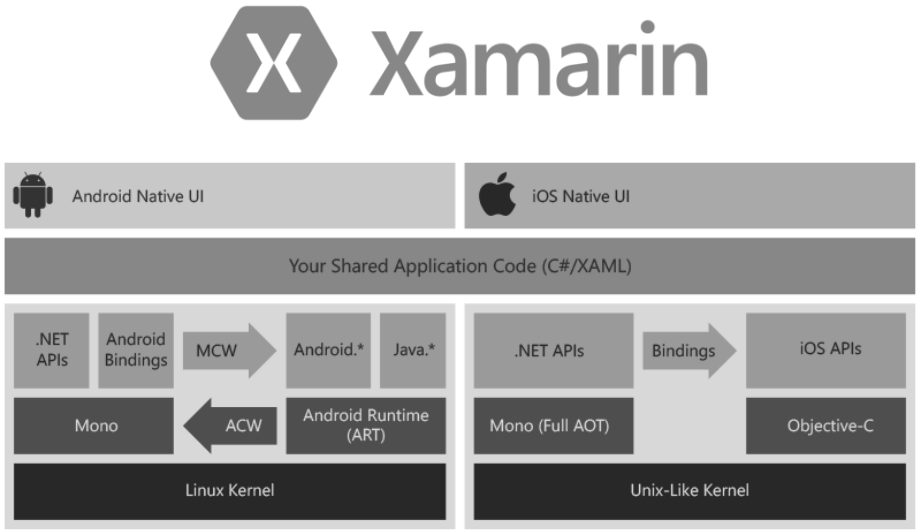
GALAXY TRAINING

.NET CORE

En 2014, Microsoft anunció un cambio dramático en la forma en que existe .NET al presentar **.NET Core**, una nueva versión multiplataforma, compatible con la nube y de código abierto

.NET Core llegó a su **lanzamiento en 2016**, convirtiéndose en la principal tecnología a considerar para los nuevos proyectos. **Microsoft comenzó a portar los servicios existentes** para trabajar con Core. Algunos no recibieron portabilidad oficial, como **Windows Communication Foundation (WCF)** y fueron sustituidos por alternativas procedentes de la comunidad.

.NET CORE



En 2016, **Microsoft adquirió Xamarin**, anteriormente una tecnología patentada para el desarrollo móvil multiplataforma, lo que también la convierte en código abierto.

Microsoft continuó avanzando hacia la **"transparencia entre el equipo del producto y la comunidad"** y los Frameworks Windows Presentation Foundation (WPF), Windows Forms y WinUI en diciembre de 2018 se convirtieron en proyectos de código abierto .

NET

En mayo de 2019, Microsoft anunció el lanzamiento que uniría el ecosistema: se suponía que todos los elementos de .NET se incluirían en la plataforma de desarrollo .NET 5. Si bien se realizaron cambios en el cronograma debido a COVID-19, la plataforma de desarrollo unificado .NET 5 finalmente se introdujo en noviembre de 2020.

El sucesor de .NET Core 3.1 y .NET Framework 4.8, .NET 5 pone orden en la fragmentación del mundo .NET y proporciona muchas características para crear aplicaciones en Windows, Linux, macOS, iOS, watchOS, Android, tvOS o mediante WebAssembly. La plataforma viene con nuevas API, características de lenguaje y capacidades de tiempo de ejecución. Además, .NET 5 incluye ASP.NET Core, Xamarin, Entity Framework Core, WPF, WinForms y ML.NET.

Historia y evolución de NET

GALAXY TRAINING

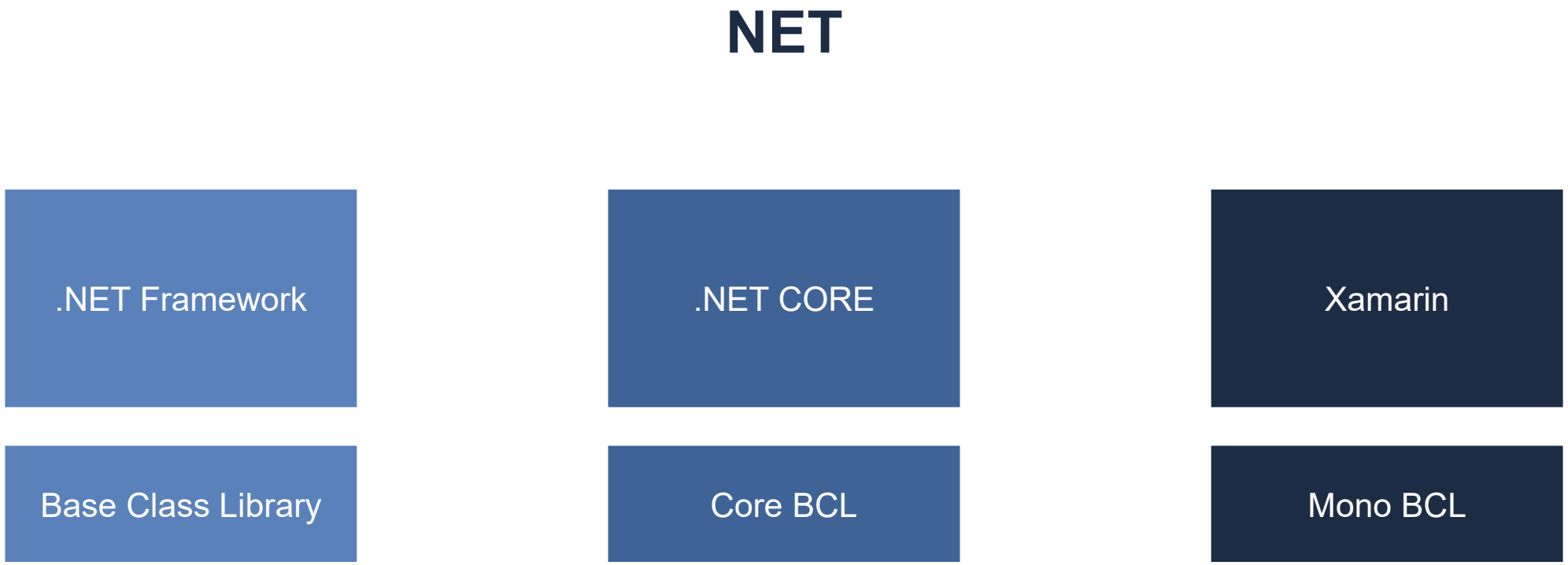
NET

.NET Framework

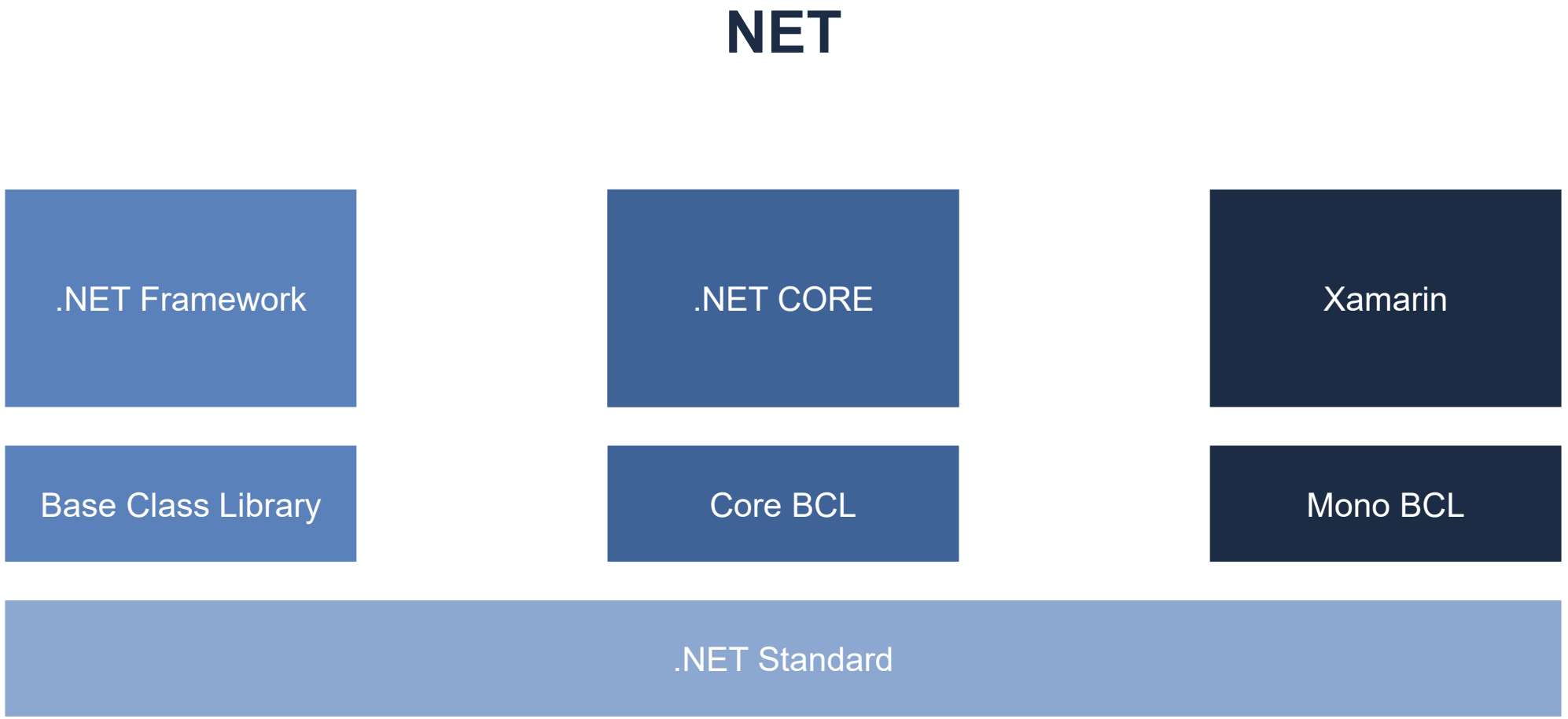
.NET CORE

Xamarin

Historia y evolución de NET



Historia y evolución de NET

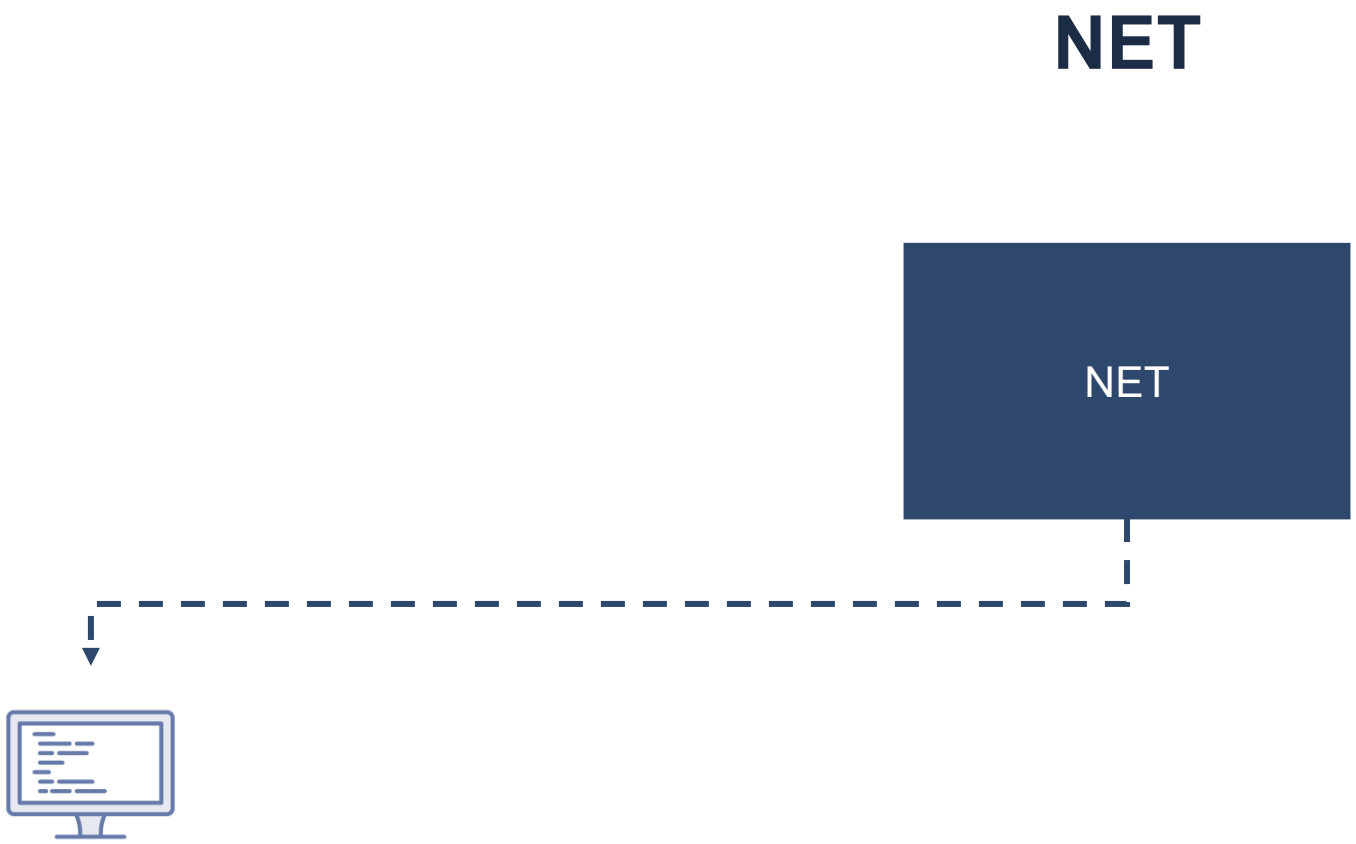


Historia y evolución de NET

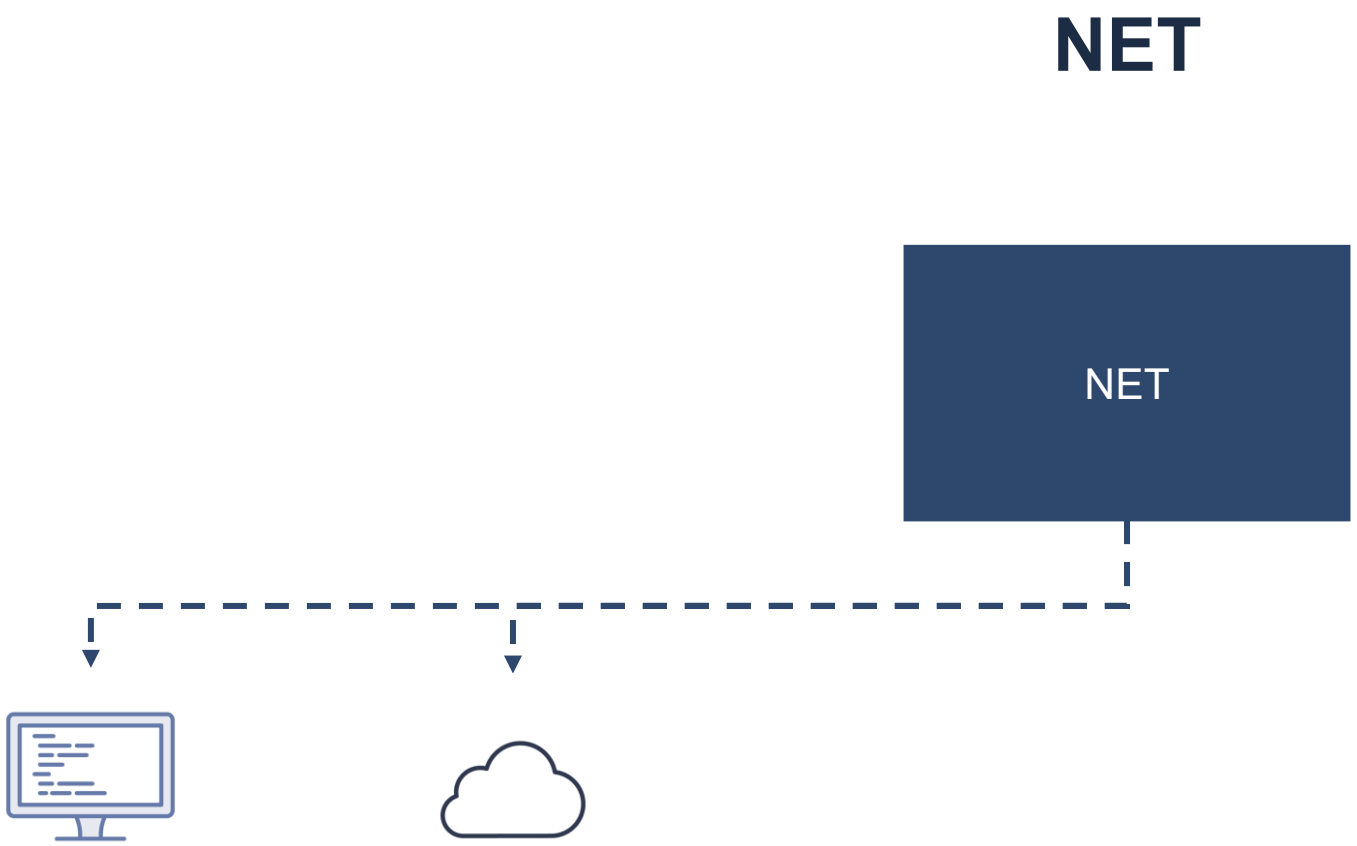
GALAXY TRAINING

NET

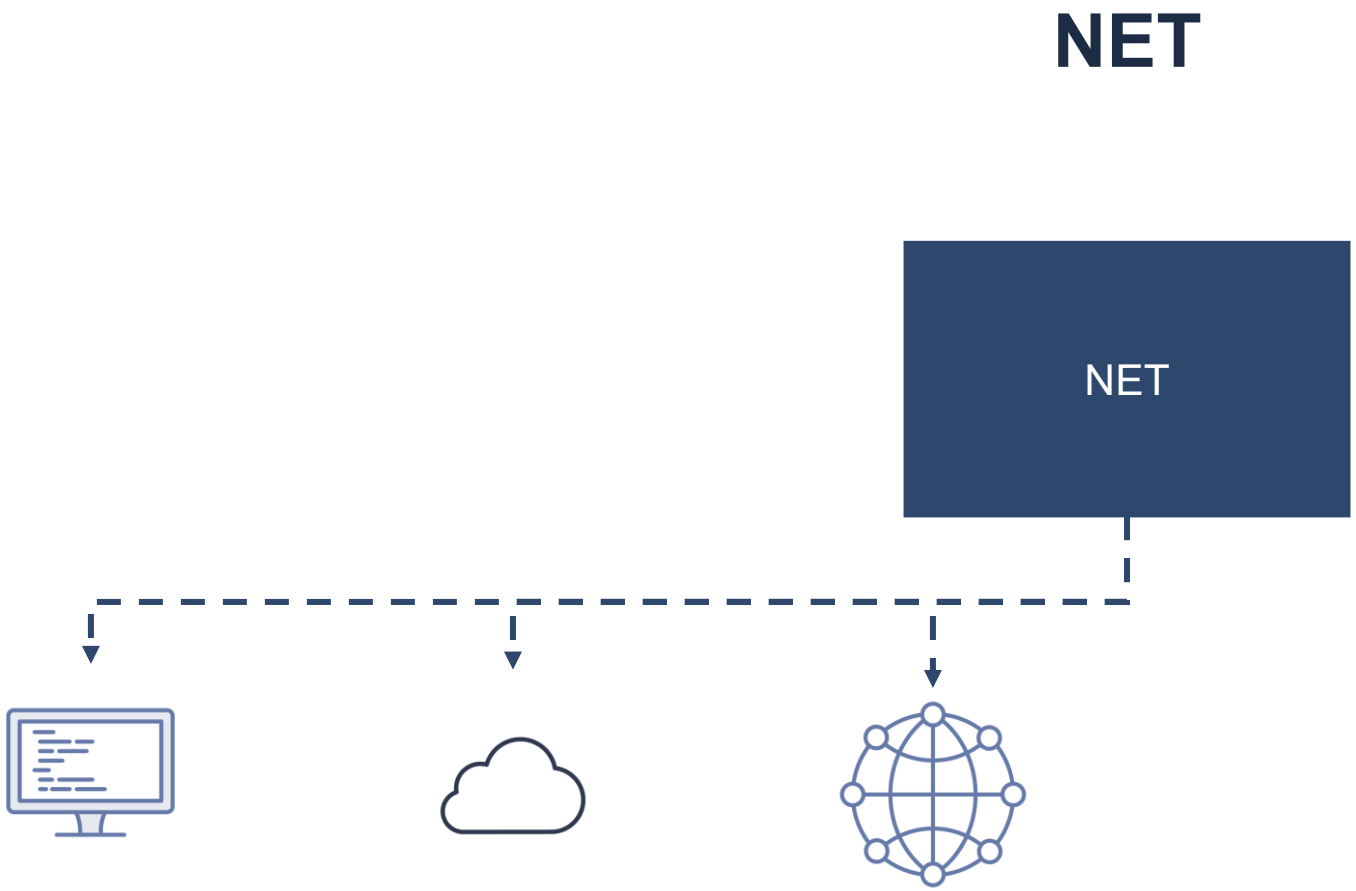
Historia y evolución de NET



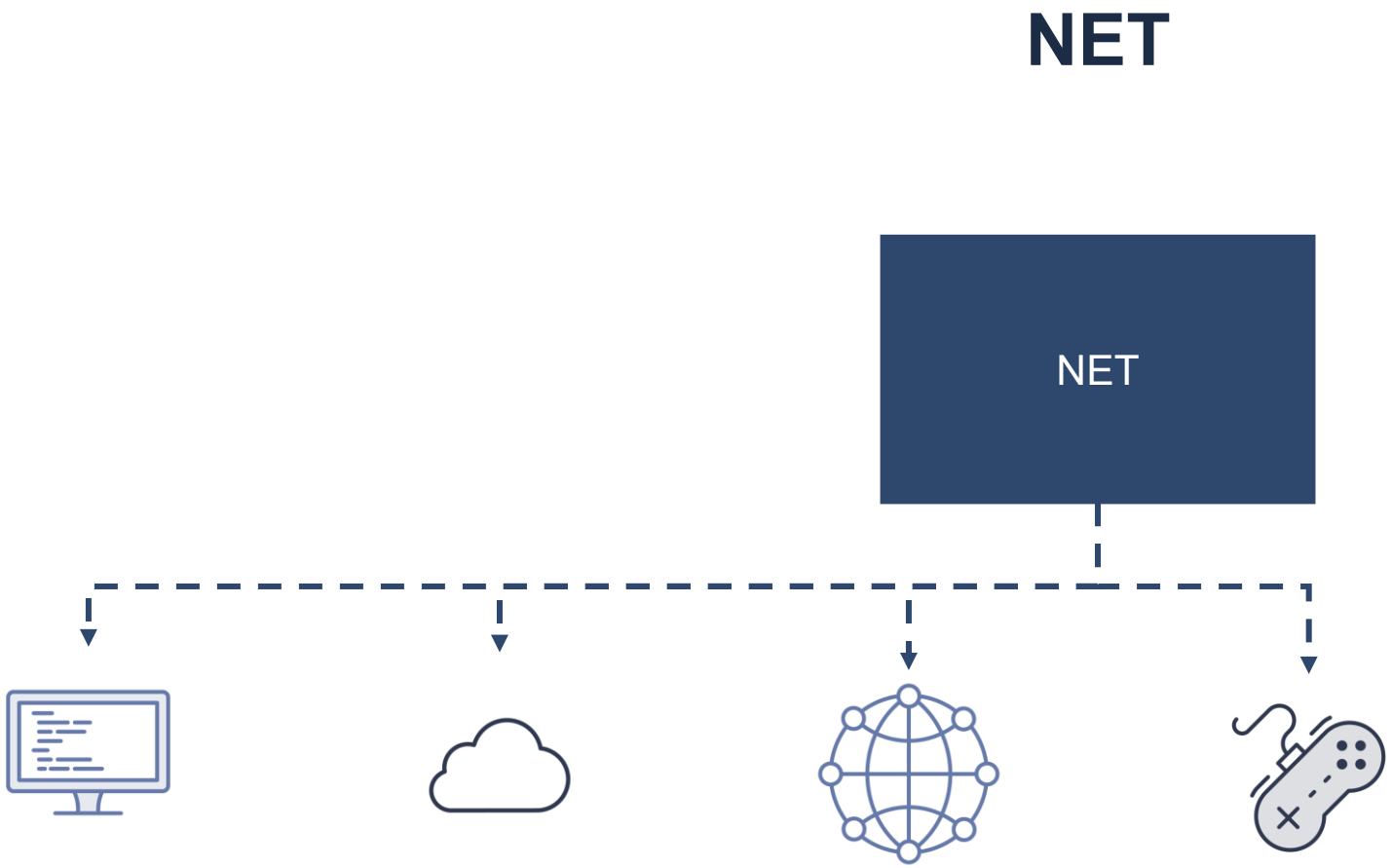
Historia y evolución de NET



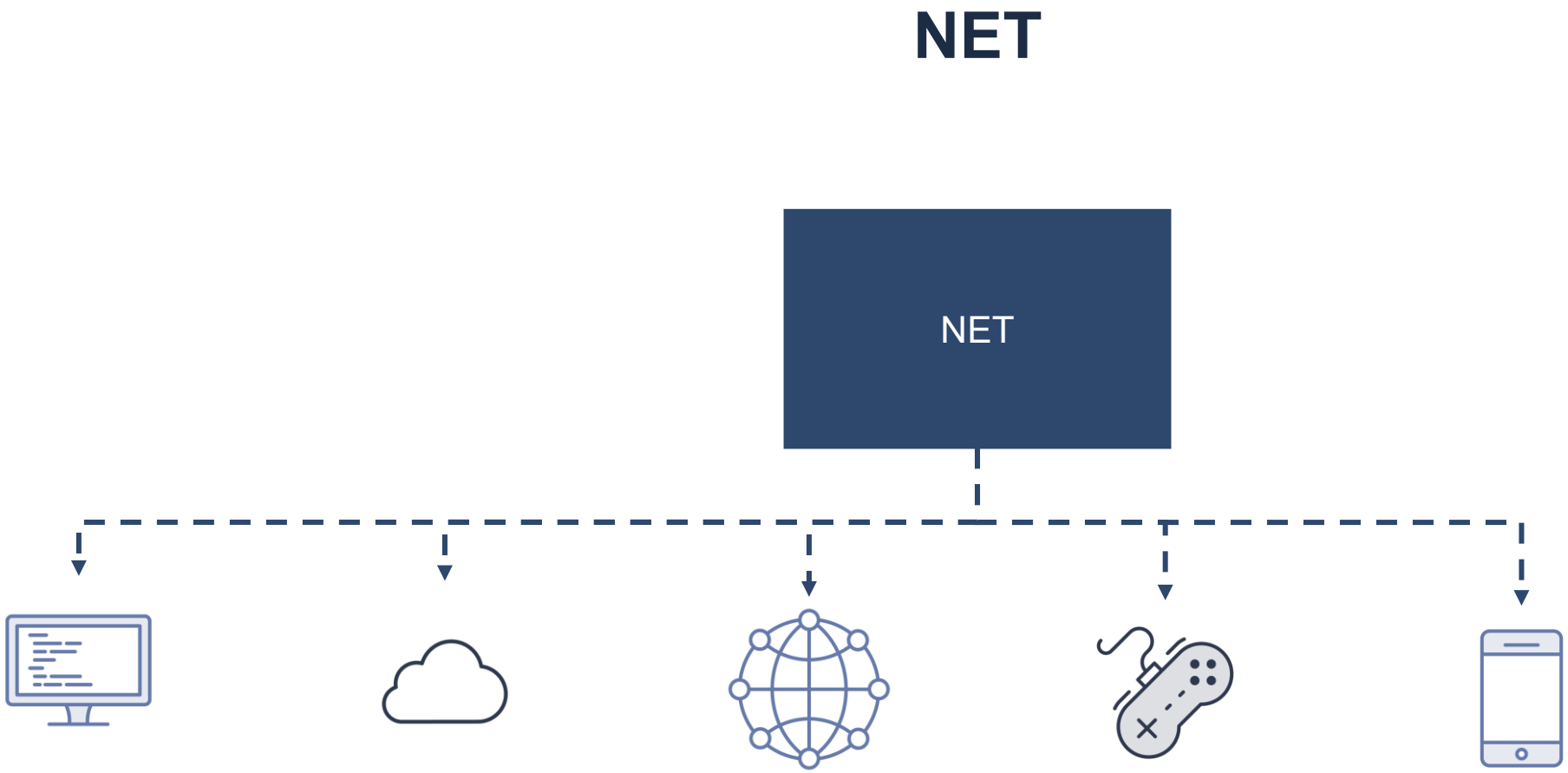
Historia y evolución de NET



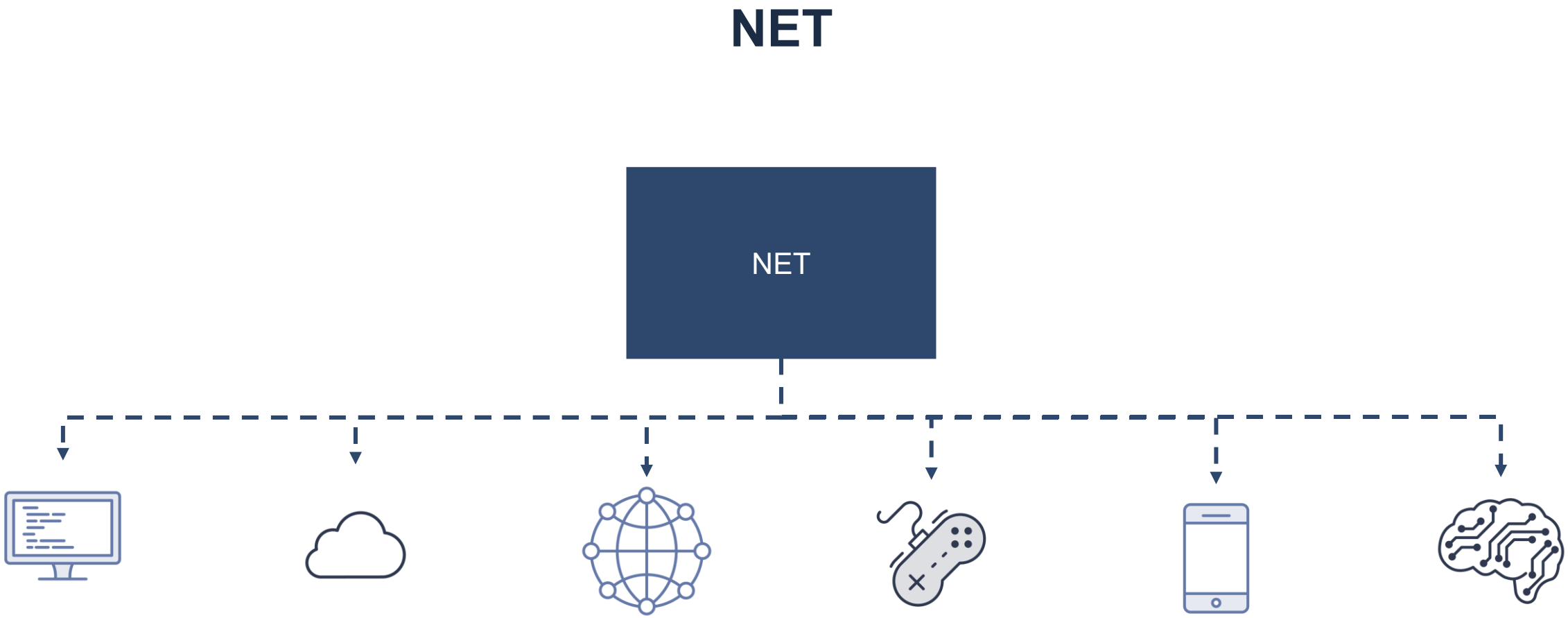
Historia y evolución de NET



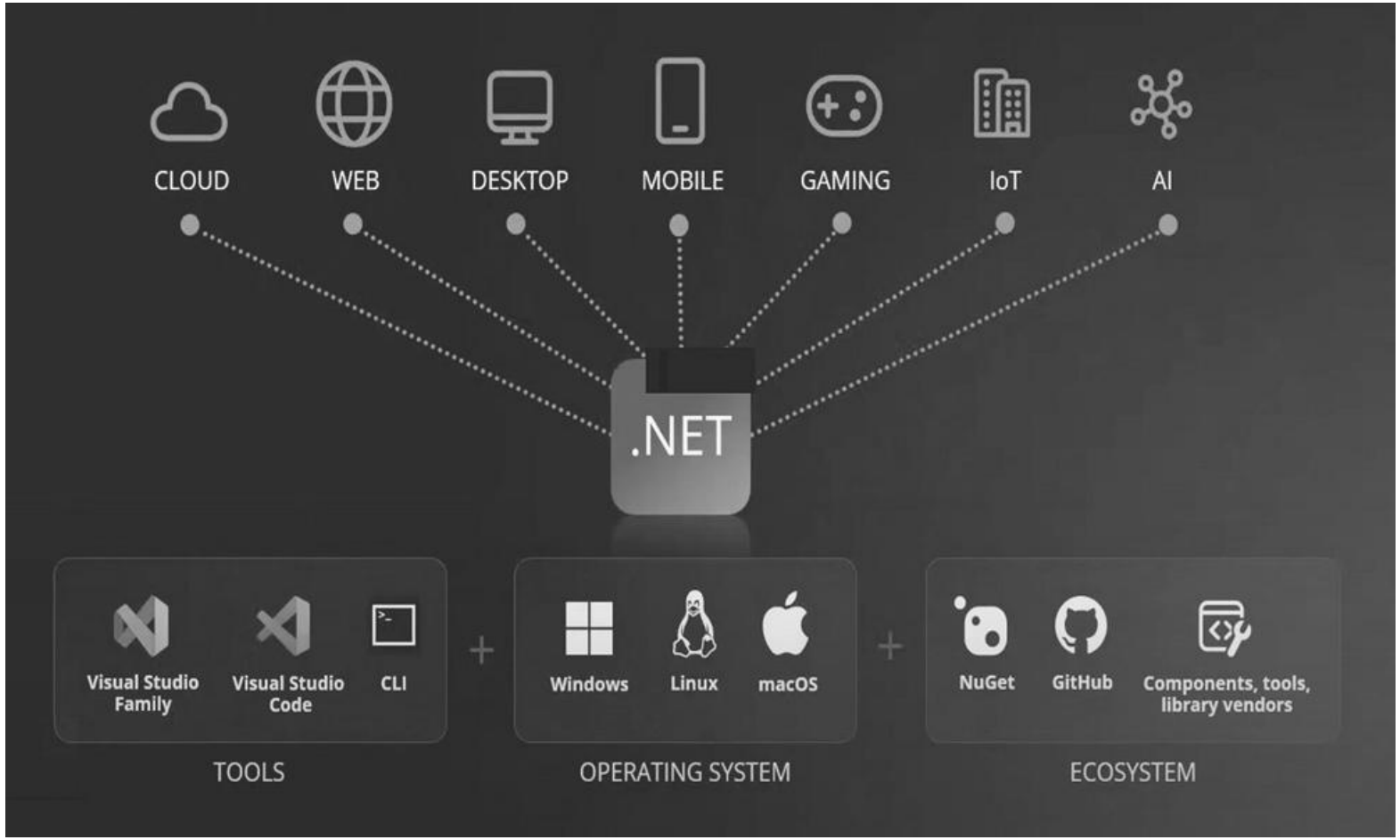
Historia y evolución de NET



Historia y evolución de NET



Historia y evolución de NET



.NET 5 estableció las bases de unificación, la versión de .NET 6 se entregó las partes finales en noviembre de 2021, y Visual Studio 2022 se lanzó el mismo día. **NET 8 es una plataforma unificada** para crear proyectos en entornos de nube, explorador, IoT, móviles y de escritorio, lo que permite a todos usar las mismas bibliotecas .NET, SDK y tiempo de ejecución.

NET

Historia y evolución de NET

GALAXY TRAINING

NET

2016 – 2018

.NET Core
1.0 - 1.1 - 2.0 - 2.1 - 2.2

Historia y evolución de NET

GALAXY TRAINING

NET

2016 – 2018

.NET Core
1.0 - 1.1 - 2.0 - 2.1 - 2.2

2019

.NET Core 3.1 LTS
End of support (Dic 22)

Historia y evolución de NET

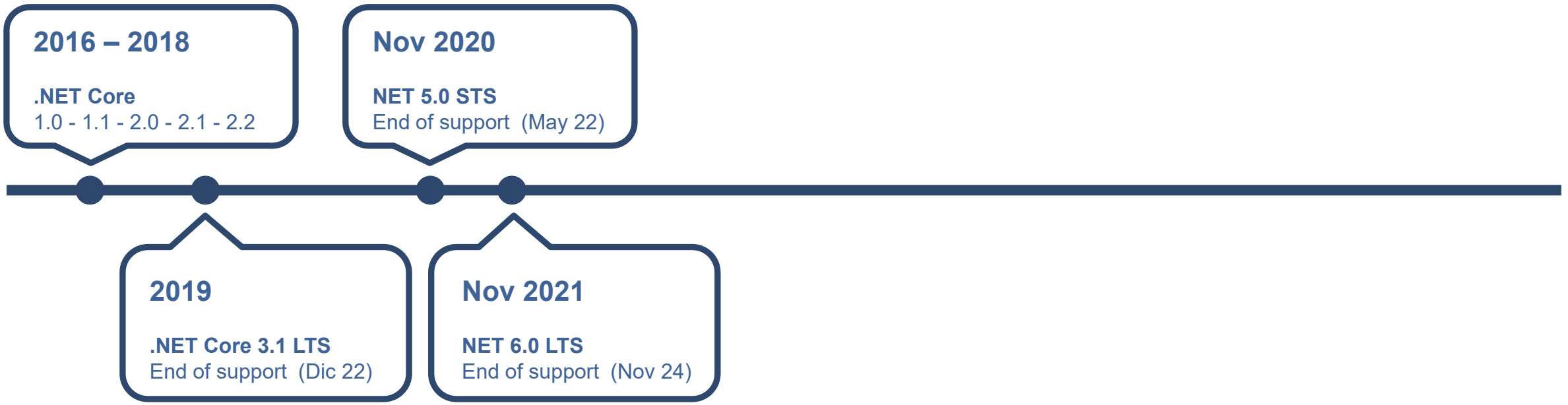
GALAXY TRAINING

NET

Historia y evolución de NET

GALAXY TRAINING

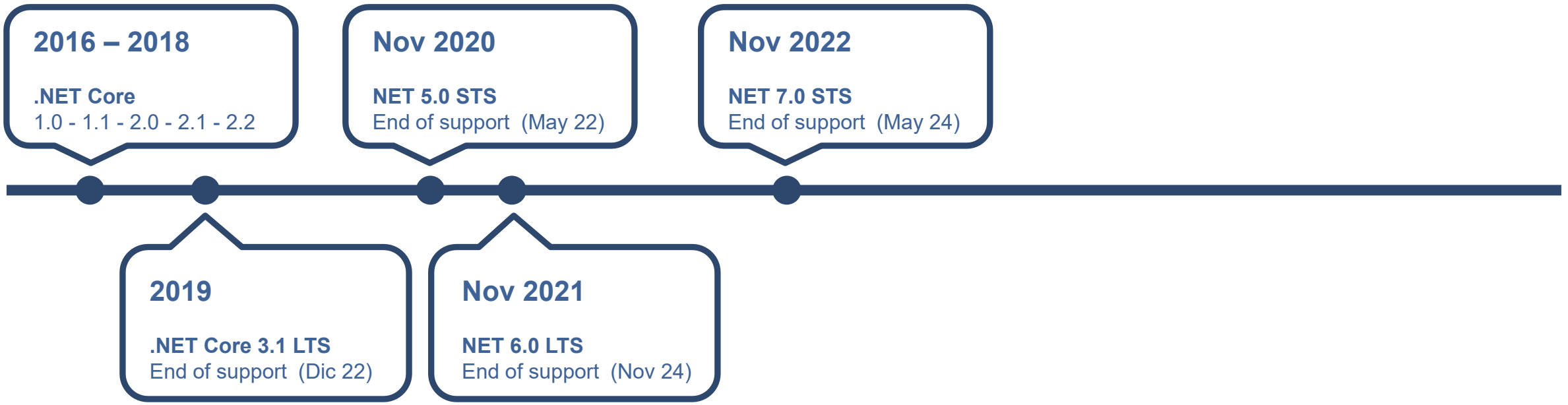
NET



Historia y evolución de NET

GALAXY TRAINING

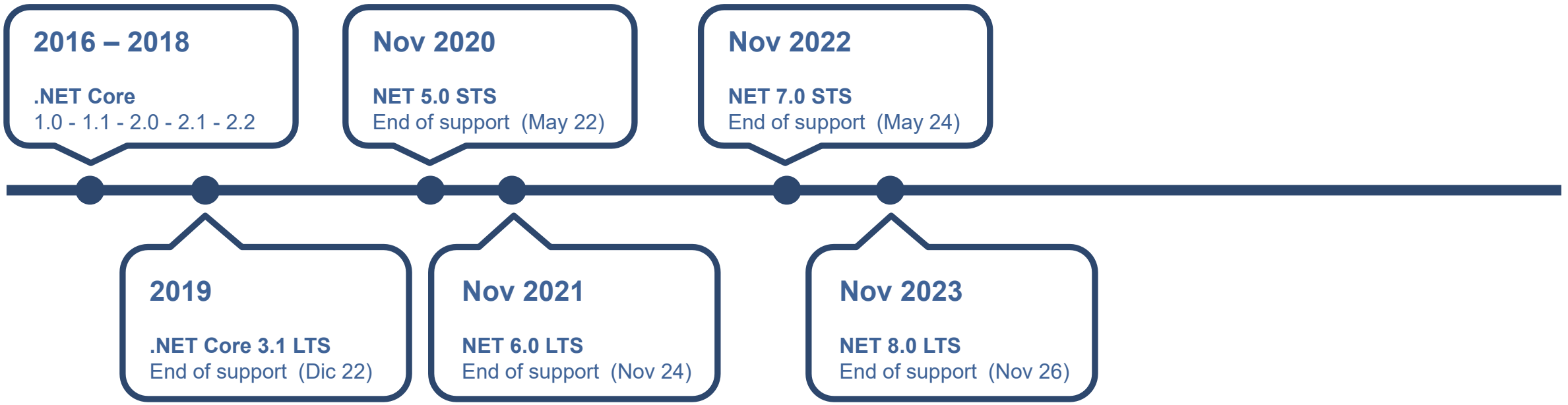
NET



Historia y evolución de NET

GALAXY TRAINING

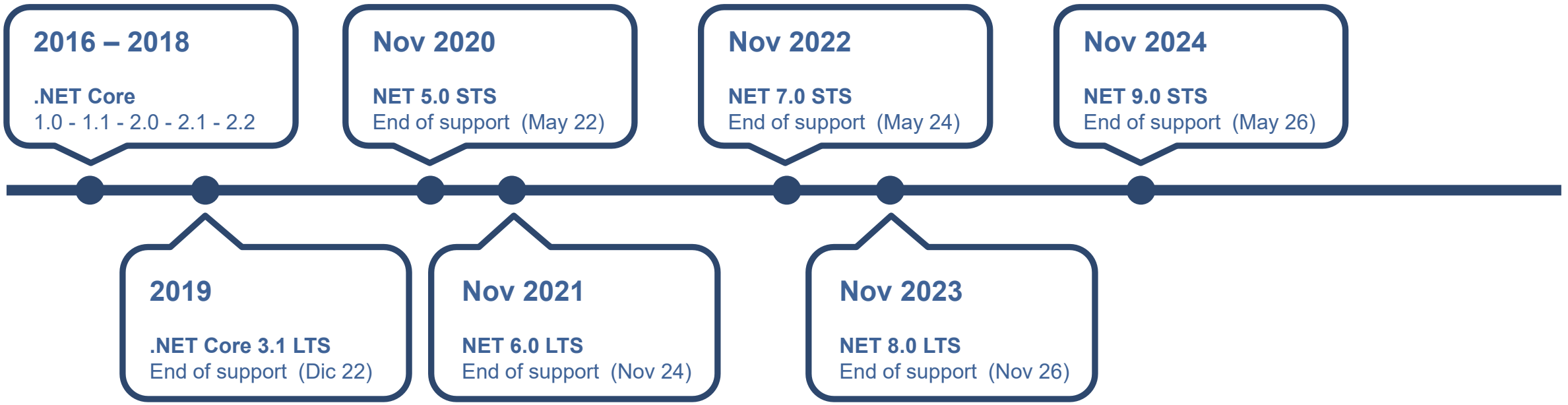
NET



Historia y evolución de NET

GALAXY TRAINING

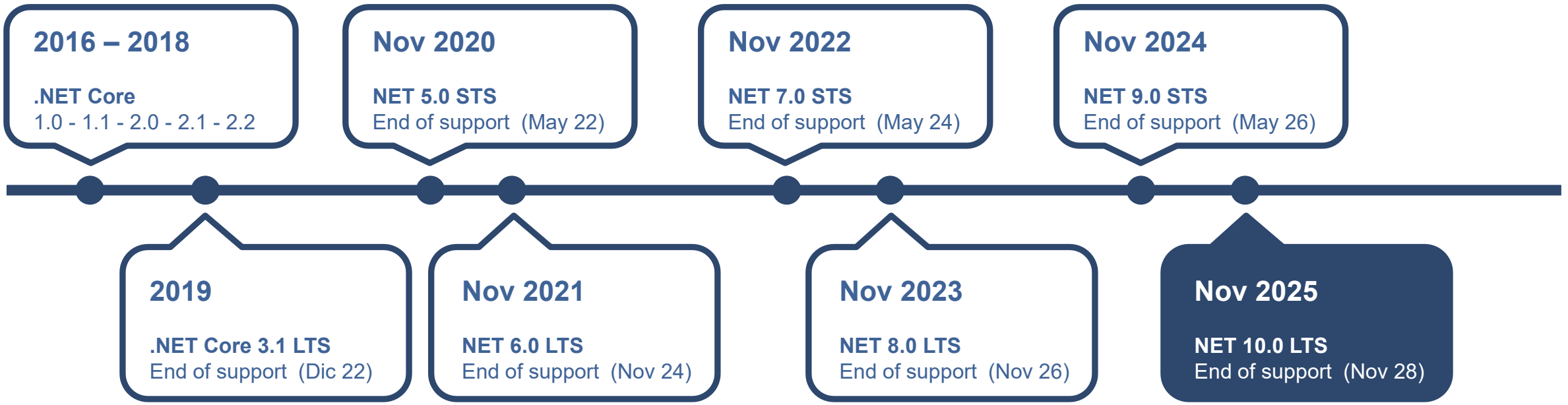
NET



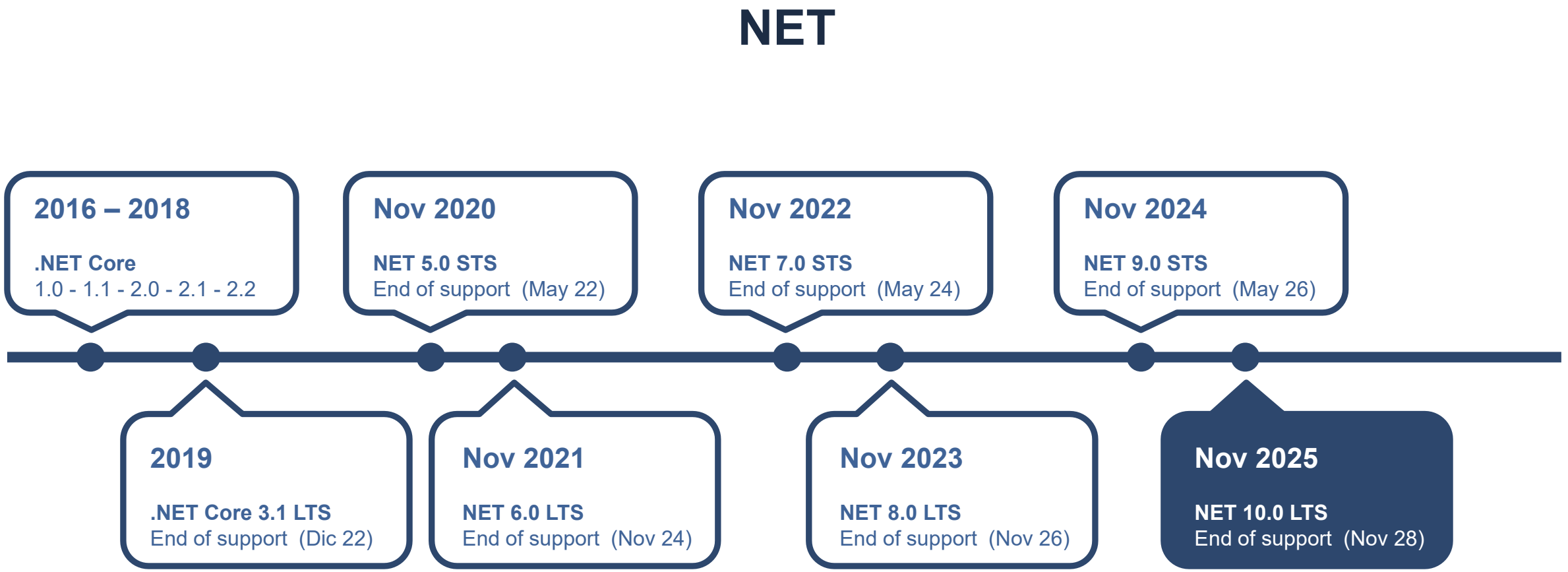
Historia y evolución de NET

GALAXY TRAINING

NET

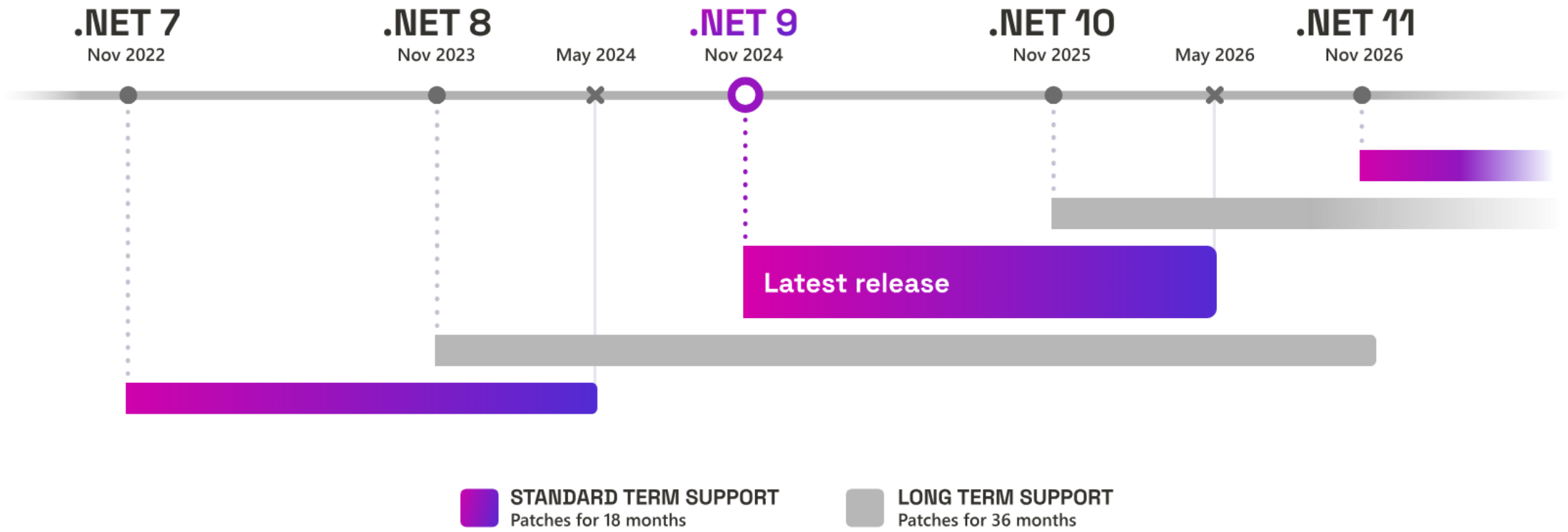


Historia y evolución de NET



https://dotnet.microsoft.com/en-us/platform/support/policy/dotnet-core

Historia y evolución de NET



Long Term Support (LTS)

Las versiones de LTS tendrán soporte durante tres años después de la versión inicial.

Standard Term Support (STS)

Las versiones STS tendrán soporte durante seis meses después de una versión STS o LTS posterior. Las versiones se producen cada 12 meses, por lo que el período de soporte técnico para STS es de 18 meses.

https://dotnet.microsoft.com/en-us/platform/support/policy/dotnet-core

02

GALAXY
TRAINING

¿Qué es NET?,
Ventajas que ofrece NET

¿Qué es NET?, Ventajas que ofrece NET

NET Es una plataforma gratuita popular que se utiliza actualmente para muchos tipos diferentes de aplicaciones, ya que proporciona el entorno de programación para la mayoría de las fases de desarrollo de software.

.**NET se adapta mejor a las empresas que buscan una amplia gama de características**, como servicios basados en web, software de escritorio y compatibilidad con infraestructura en la nube.

¿Qué es NET?, Ventajas que ofrece NET

GALAXY TRAINING

.NET foundation

Openness

Community

Rapid innovation

.NET Compiler Platform ("Roslyn")
LLILC
MVVM Light Toolkit
MSBuild
ASP.NET MVC
MEF
IdentityManager
ASP.NET Core
.NET SDK for Hadoop
Kudu
.NET Core
.NET Micro Framework
Mono
Mailkit
Cecil
Xamarin.Auth
Mimekit
Umbraco
ASP.NET AJAX Control Toolkit
Open Live Writer
NuGet
Cake
WorldWide Telescope
Couchbase Lite for .NET
ASP.NET SignalR
ASP.NET Web Pages
Microsoft Azure SDK for .NET
WCF
Xamarin SDK
Entity Framework
Open XML SDK
IdentityServer
Microsoft Azure WebJobs SDK
OWIN Authentication Middleware
Microsoft Web Protection Library
ASP.NET Web API
Prism
System.Drawing
Orchard CMS
ProtoBuild
Xamarin.Mobile
Salesforce Toolkits for .NET
Orleans

Features

Protection
- Licenses
- Copyrights
- Trademarks
- Patents

Practices
- Mentorship
- Governance
- Feedback
- Co-ordination

Visibility
- Media
- Branding
- Events

Support
- Hosting
- Code signing
- CLA Management
- Swag

¿Qué es NET?, Ventajas que ofrece NET

Ventajas

Modelo de desarrollo de orientado a objetos

Monitorización automática en ASP.NET

IDE Visual Studio

Popularidad y comunidad de .NET

Potentes compiladores Roslyn y RyuJIT

Soporte técnico activo de Microsoft

Implementación flexible y de fácil mantenimiento

Diseño multiplataforma

¿Qué es NET?, Ventajas que ofrece NET

Ventajas Net 8

Mejoras en el rendimiento

Soporte para C# 12

.NET MAUI

ASP.NET Core y EF Core mejoras

Windows Forms y Windows Presentation Foundation (WPF) actualizaciones

Mejor manejo de recursos, lo que lleva a una mayor velocidad y estabilidad.

.NET Aspire

Una nueva pila preparada para la nube que facilita la construcción de aplicaciones observables, distribuidas y listas para producción, con paquetes NuGet que abordan problemas específicos nativos de la nube.

GALAXY TRAINING

Ventajas Net 9

Ventajas Net 9

- **Ejecución más rápida**: Gracias a optimizaciones en el runtime y el compilador JIT, las aplicaciones se ejecutan de manera más eficiente.
- **Menor uso de memoria**: La recolección de basura (GC) adaptativa permite un ajuste dinámico del consumo de memoria según la carga de la aplicación.
- **Optimización para hardware moderno:** Compatibilidad ampliada con plataformas ARM64 y mejoras para hardware de alto rendimiento.

Ventajas Net 9

- **Nuevas características de lenguaje**:
  - Colecciones params para simplificar la manipulación de listas.
  - Nuevas secuencias de escape (\e) para mayor control en strings.
  - Acceso implícito a índices en inicializadores de colecciones.
- ***Programación* avanzada**:
  - Soporte para *ref locals*, *unsafe* y *ref struct* en contextos asíncronos y genéricos.
- **Modularidad mejorada**:
  - Propiedades parciales y priorización de sobrecargas en bibliotecas.

Ventajas Net 9

- **Desarrollo unificado**: Permite escribir una sola base de código para aplicaciones que funcionan en Windows, macOS, Android y iOS.
- **Optimización para ARM64**: Mejor rendimiento en dispositivos modernos como Mac M1/M2 y Raspberry Pi.
- **Interfaz enriquecida**: Mejoras en el diseño y rendimiento de las interfaces de usuario.

Ventajas Net 9

- **Rendimiento web superior:**
  - Optimización en protocolos modernos como HTTP/3 y gRPC.
  - Tiempos de respuesta más rápidos en aplicaciones web.
- **Manejo de datos eficiente:**
  - Mejoras en consultas LINQ y soporte para bases de datos avanzadas.
- **Escalabilidad optimizada:**
  - Capacidad para manejar mayor cantidad de usuarios y cargas de trabajo.

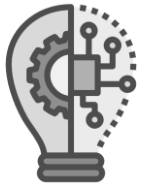
Ventajas Net 9

Rendimiento Excepcional

Soporte para C# 13

Desarrollo Multiplataforma con .NET MAUI

Mejoras en ASP.NET Core y EF Core



- **Integración simplificada**: Nuevos bloques de construcción para trabajar con IA mediante *Microsoft.Extensions.AI*.
- **Procesamiento de datos eficiente:**
  - Tipos tensoriales (*Tensor<T>*) para cálculos multidimensionales.
  - Herramientas mejoradas para IA y modelos de lenguaje.
- **Listo para el futuro**: Facilita el uso de IA en aplicaciones modernas, incluyendo servicios basados en nube.

¿Qué es NET?, Ventajas que ofrece NET

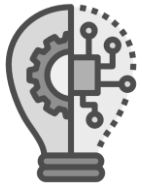
Ventajas Net 9

Rendimiento Excepcional

Soporte para C# 13

Desarrollo Multiplataforma con .NET MAUI

Mejoras en ASP.NET Core y EF Core



- **Actualizaciones significativas:**
  - Mejoras en la compatibilidad con hardware actual y plataformas modernas.
  - Optimización para pantallas de alta resolución.
- **Mayor estabilidad y velocidad:**
  - Rendimiento optimizado para aplicaciones de escritorio tradicionales.

¿Qué es NET?, Ventajas que ofrece NET

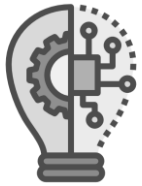
Ventajas Net 9

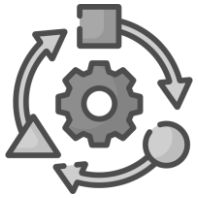
Rendimiento Excepcional

Soporte para C# 13

Desarrollo Multiplataforma con .NET MAUI

Mejoras en ASP.NET Core y EF Core





- **Recolección de basura dinámica**: Ajuste automático del consumo de memoria para mejorar la estabilidad.
- **Menor consumo de CPU**: Uso eficiente de recursos incluso bajo cargas elevadas.
- **Mayor estabilidad**: Ideal para aplicaciones de larga duración o con cargas variables.

¿Qué es NET?, Ventajas que ofrece NET

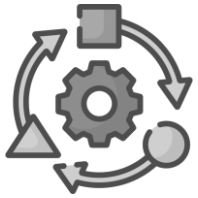
Ventajas Net 9

Rendimiento Excepcional

Soporte para C# 13

Desarrollo Multiplataforma con .NET MAUI

Mejoras en ASP.NET Core y EF Core



- **Capacidades avanzadas para la nube:**
  - Creación de aplicaciones distribuidas y observables.
  - Optimizado para trabajar con contenedores y microservicios.
- **Integración con herramientas modernas:**
  - Compatible con NuGet y workloads especializados.
- **Listo para producción:** Despliegues simplificados y soporte para aplicaciones listas para operar en entornos de producción.

¿Qué es NET?, Ventajas que ofrece NET

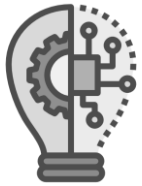
Ventajas Net 9

Rendimiento Excepcional

Soporte para C# 13

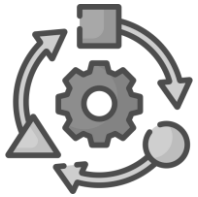
Desarrollo Multiplataforma con .NET MAUI

Mejoras en ASP.NET Core y EF Core



Innovaciones en IA

Windows Forms y WPF Modernizados

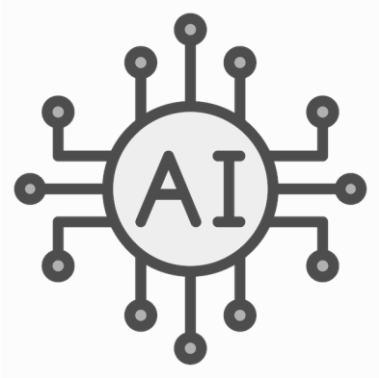


Gestión Adaptativa de Recursos

Preparado para la Nube con .NET Aspire

GALAXY TRAINING

Desafíos



Complejidad en la Implementación de IA

Costo de licencia

La brecha entre la liberación y la estabilidad

Migración y Compatibilidad

03

Principales características de NET

GALAXY
TRAINING

Principales características de NET

Características

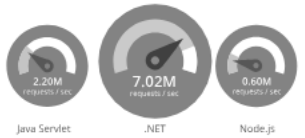
Multi-lenguaje

Any app, any platform

Compatibilidad Mejorada

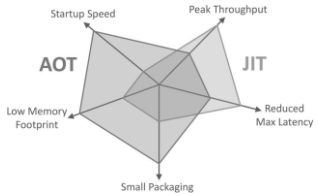
Diseñado para la Nube (Cloud Native)

Rendimiento Web Superior



On-stack replacement (OSR), Permite que el tiempo de ejecución cambie el código ejecutado

Soporte para ARM64



Generación de Código Nativo (AOT)

04

Características de C# 13

Características de C# 11

- **Literales de cadena sin formato**. Un literal de cadena sin formato comienza con al menos tres caracteres de comillas dobles (""")
- **Literales de cadena de UTF-8**. Puede especificar el sufijo u8 en un literal de cadena para especificar la codificación de caracteres UTF-8
- **Miembros requeridos**. Puede agregar el modificador **required** a propiedades y campos para aplicar constructores y llamadores para inicializar esos valores
- **Estructuras predeterminadas automáticas.** Este cambio significa que el compilador inicializa automáticamente cualquier campo o propiedad automática no inicializados por un constructor.
- **Tipos locales de archivo.** El modificador de acceso **file** se puede usar para crear un tipo cuya visibilidad esté limitada al archivo de origen en el que se declara
- **Patrones de lista.** Amplía la coincidencia de patrones para buscar coincidencias con secuencias de elementos de una lista o una matriz

La lista completa en https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-11

Características de C# 12

- **Constructores principales**. Los parámetros del constructor principal están en el ámbito en toda la definición de clase. Es importante ver los parámetros del constructor principal como parámetros. Todos los demás constructores de una clase deben llamar al constructor principal, directa o indirectamente, a través de una invocación de constructor **this()**.
- **Alias de cualquier tipo**. Permitirle apuntar a cualquier tipo de tipo, no solo a los tipos con nombre (namespace). Esto admitiría tipos que no se permiten hoy en día, como: tipos de tupla, tipos de puntero, tipos de matriz, etc.
- **Expresiones de colección**. Puede usar una expresión de colección para crear valores de colección comunes. Una expresión de colección es una sintaxis tersa que, cuando se evalúa, se puede asignar a muchos tipos de colección diferentes.
- **Parámetros lambda predeterminados.** Ahora puede definir valores predeterminados para parámetros en expresiones lambda. La sintaxis y las reglas son las mismas que agregar valores predeterminados para los argumentos a cualquier método o función local.
- **Interceptores.** Los interceptores proporcionan una instalación limitada para cambiar la semántica del código existente agregando código nuevo a una compilación, por ejemplo, en un generador de origen.

La lista completa en https://learn.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-12

Características de C# 13

- **Colecciones params.** el modificador **params** se ha ampliado para admitir tipos de colección adicionales, no solo arreglos tradicionales. Esto incluye interfaces de colección genéricas e incluso tipos como **System.Span<T>** y **System.ReadOnlySpan<T>.**
  - Simplifica el código cuando trabajas con métodos que aceptan múltiples parámetros.
  - Mejora la legibilidad y reduce errores en código repetitivo.

Características de C# 13

- **Colecciones params.** el modificador **params** se ha ampliado para admitir tipos de colección… de colección genéricas e incluso tipos como **System.Span<T>** y **System.ReadOnlySpan<T>.**
  - Simplifica el código cuando trabajas con métodos que aceptan múltiples parámetros.
  - Mejora la legibilidad y reduce errores en código repetitivo.

Soporte para más tipos de colección:

- Se pueden usar colecciones reconocidas, como **System.Span<T>, System.ReadOnlySpan<T>**, y colecciones que implementen **IEnumerable<T>** con un método **Add**.
- Interfaces soportadas:
  - **IEnumerable<T>**
  - **IReadOnlyCollection<T>**
  - **IReadOnlyList<T>**
  - **ICollection<T>**
  - **IList<T>**

Características de C# 13

- **Colecciones params.** el modificador **params** se ha ampliado para admitir tipos de colección adicionales, no solo arreglos tradicionales. Esto incluye interfaces de colección genéricas e incluso tipos como **System.Span<T>** y **System.ReadOnlySpan<T>.**
  - Simplifica el código cuando trabajas con métodos que aceptan múltiples parámetros.
  - Mejora la legibilidad y reduce errores en código repetitivo.
- **Nuevo Tipo y Semántica para lock**. Introduce un nuevo tipo y reglas revisadas para el uso de **lock**, lo que mejora la sincronización multihilo.
  - Aumenta la claridad al escribir código que utiliza sincronización.
  - Reduce la posibilidad de errores al trabajar con recursos compartidos.

Características de C# 13

- **Colecciones params.** el modificador **params** se ha ampliado para admitir tipos de colección a[illegible] de colección genéricas e incluso tipos como **System.Span<T>** y **System.ReadOnlySpan<T>.**
  - Simplifica el código cuando trabajas con métodos que aceptan múltiples parámetros.
  - Mejora la legibilidad y reduce errores en código repetitivo.
- **Nuevo Tipo y Semántica para lock**. Introduce un nuevo tipo y reglas revisadas para el
  - Aumenta la claridad al escribir código que utiliza sincronización.
  - Reduce la posibilidad de errores al trabajar con recursos compartidos.

System.Threading.Lock

1. Nuevo Modelo Basado en Scope:
   - Usa el método **Lock.EnterScope()** para crear un contexto de exclusión mutua.
   - Devuelve una estructura **ref struct** que implementa el patrón **Dispose()**, lo que garantiza que el bloqueo se libere correctamente al salir del alcance (scope).
2. Compatibilidad con la Palabra Clave **lock**:
   - Si la palabra clave **lock** se utiliza con un objeto del tipo **Lock**, el compilador usa automáticamente la nueva API de **System.Threading.Lock** en lugar del enfoque tradicional basado en **Monitor**.
3. Compatibilidad Retroactiva:
   - Si un objeto **Lock** se convierte a otro tipo, el código basado en **Monitor** será generado, manteniendo compatibilidad con versiones anteriores.
4. Sin Cambios de Código Extra:
   - Solo necesitas cambiar el tipo de objeto de bloqueo para aprovechar esta mejora. No se requieren otros cambios en el código.

Características de C# 13

- **Colecciones params.** el modificador **params** se ha ampliado para admitir tipos de colección adicionales, no solo arreglos tradicionales. Esto incluye interfaces de colección genéricas e incluso tipos como **System.Span<T>** y **System.ReadOnlySpan<T>.**
  - Simplifica el código cuando trabajas con métodos que aceptan múltiples parámetros.
  - Mejora la legibilidad y reduce errores en código repetitivo.
- **Nuevo Tipo y Semántica para lock**. Introduce un nuevo tipo y reglas revisadas para el uso de **lock**, lo que mejora la sincronización multihilo.
  - Aumenta la claridad al escribir código que utiliza sincronización.
  - Reduce la posibilidad de errores al trabajar con recursos compartidos.
- **Nueva Secuencia de Escape \e.** Soporte para la secuencia de escape **\e**, que representa el carácter ESC (ASCII 27).
  - Útil para escenarios que requieren caracteres de control, como terminales o protocolos de red.
  - Facilita la manipulación de cadenas para sistemas heredados.

Características de C# 13

- **Colecciones params.** el modificador **params** se ha ampliado para admitir tipos de colección a[illegible] de colección genéricas e incluso tipos como **System.Span<T>** y **System.ReadOnlySpan<T>.**
  - Simplifica el código cuando trabajas con métodos que aceptan múltiples parámetros.
  - Mejora la legibilidad y reduce errores en código repetitivo.
- **Nuevo Tipo y Semántica para lock**. Introduce un nuevo tipo y reglas revisadas para el u[illegible]
  - Aumenta la claridad al escribir código que utiliza sincronización.
  - Reduce la posibilidad de errores al trabajar con recursos compartidos.
- **Nueva Secuencia de Escape \e.** Soporte para la secuencia de escape **\e**, que represe[illegible]
  - Útil para escenarios que requieren caracteres de control, como terminales o protocolos c[illegible]
  - Facilita la manipulación de cadenas para sistemas heredados.

¿Qué problema resuelve \e?

1. Métodos anteriores para el carácter ESCAPE:
   - Antes de C# 13, podías usar:
     - **\u001b**: Secuencia Unicode explícita.
     - **\x1b**: Secuencia hexadecimal.
   - Sin embargo, **x1b** no era la mejor opción, porque si había caracteres válidos de dígitos hexadecimales después de 1b, estos se convertían en parte de la secuencia. Esto podía llevar a errores difíciles de depurar.
2. Mejora con **\e**:
   - **\e** es una forma directa y no ambigua de representar el carácter ESCAPE.
   - Hace el código más legible y menos propenso a errores.

Características de C# 13

- **Colecciones params.** el modificador **params** se ha ampliado para admitir tipos de colección adicionales, no solo arreglos tradicionales. Esto incluye interfaces de colección genéricas e incluso tipos como **System.Span<T>** y **System.ReadOnlySpan<T>.**
  - Simplifica el código cuando trabajas con métodos que aceptan múltiples parámetros.
  - Mejora la legibilidad y reduce errores en código repetitivo.
- **Nuevo Tipo y Semántica para lock**. Introduce un nuevo tipo y reglas revisadas para el uso de **lock**, lo que mejora la sincronización multihilo.
  - Aumenta la claridad al escribir código que utiliza sincronización.
  - Reduce la posibilidad de errores al trabajar con recursos compartidos.
- **Nueva Secuencia de Escape \e.** Soporte para la secuencia de escape **\e**, que representa el carácter ESC (ASCII 27).
  - Útil para escenarios que requieren caracteres de control, como terminales o protocolos de red.
  - Facilita la manipulación de cadenas para sistemas heredados.
- **Inicializadores de Índices Implícitos.** Ahora es posible inicializar colecciones utilizando índices directamente en la definición, sin métodos adicionales.
  - Hace el código más limpio y conciso al inicializar colecciones.
  - Facilita la lectura y el mantenimiento de estructuras complejas.

Características de C# 13

- **Colecciones params.** el modificador **params** se ha ampliado para admitir tipos de colección a[illegible] de colección genéricas e incluso tipos como **System.Span<T>** y **System.ReadOnlySpan<T>.**
  - Simplifica el código cuando trabajas con métodos que aceptan múltiples parámetros.
  - Mejora la legibilidad y reduce errores en código repetitivo.
- **Nuevo Tipo y Semántica para lock**. Introduce un nuevo tipo y reglas revisadas para el us[illegible]
  - Aumenta la claridad al escribir código que utiliza sincronización.
  - Reduce la posibilidad de errores al trabajar con recursos compartidos.
- **Nueva Secuencia de Escape \e.** Soporte para la secuencia de escape **\e**, que representa e[illegible]
  - Útil para escenarios que requieren caracteres de control, como terminales o protocolos c[illegible]
  - Facilita la manipulación de cadenas para sistemas heredados.
- **Inicializadores de Índices Implícitos.** Ahora es posible inicializar colecciones utilizando í[illegible]
  - Hace el código más limpio y conciso al inicializar colecciones.
  - Facilita la lectura y el mantenimiento de estructuras complejas.

¿Qué es el operador ^?

El operador **^** es un índice relativo que cuenta los elementos desde el final de una colección. Por ejemplo:

- **^1** se refiere al último elemento.
- **^2** se refiere al penúltimo elemento, y así sucesivamente.

En versiones anteriores, este operador solo podía usarse en expresiones, no en inicializadores de objetos.

Ahora puedes usar el operador **^** dentro de un inicializador de objeto para asignar valores desde el final de un arreglo. Esto permite un código más conciso y claro cuando necesitas llenar arreglos en orden inverso.

Características de C# 13

- **Colecciones params.** el modificador **params** se ha ampliado para admitir tipos de colección adicionales, no solo arreglos tradicionales. Esto incluye interfaces de colección genéricas e incluso tipos como **System.Span<T>** y **System.ReadOnlySpan<T>.**
  - Simplifica el código cuando trabajas con métodos que aceptan múltiples parámetros.
  - Mejora la legibilidad y reduce errores en código repetitivo.
- **Nuevo Tipo y Semántica para lock**. Introduce un nuevo tipo y reglas revisadas para el uso de **lock**, lo que mejora la sincronización multihilo.
  - Aumenta la claridad al escribir código que utiliza sincronización.
  - Reduce la posibilidad de errores al trabajar con recursos compartidos.
- **Nueva Secuencia de Escape \e.** Soporte para la secuencia de escape **\e**, que representa el carácter ESC (ASCII 27).
  - Útil para escenarios que requieren caracteres de control, como terminales o protocolos de red.
  - Facilita la manipulación de cadenas para sistemas heredados.
- **Inicializadores de Índices Implícitos.** Ahora es posible inicializar colecciones utilizando índices directamente en la definición, sin métodos adicionales.
  - Hace el código más limpio y conciso al inicializar colecciones.
  - Facilita la lectura y el mantenimiento de estructuras complejas.
- **Mejoras en Grupos de Métodos.** Los grupos de métodos ahora tienen inferencia de tipos más precisa y natural, facilitando su uso en expresiones lambda y delegados.
  - Reduce la necesidad de conversiones explícitas de tipos.
  - Mejora la compatibilidad con delegados.

Características de C# 13

- **Colecciones params.** el modificador **params** se ha ampliado para admitir tipos de colección a de colección genéricas e incluso tipos como **System.Span<T>** y **System.ReadOnlySpan<T>.**
  - Simplifica el código cuando trabajas con métodos que aceptan múltiples parámetros.
  - Mejora la legibilidad y reduce errores en código repetitivo.
- **Nuevo Tipo y Semántica para lock**. Introduce un nuevo tipo y reglas revisadas para el us
  - Aumenta la claridad al escribir código que utiliza sincronización.
  - Reduce la posibilidad de errores al trabajar con recursos compartidos.
- **Nueva Secuencia de Escape \e.** Soporte para la secuencia de escape **\e**, que representa e
  - Útil para escenarios que requieren caracteres de control, como terminales o protocolos c
  - Facilita la manipulación de cadenas para sistemas heredados.
- **Inicializadores de Índices Implícitos.** Ahora es posible inicializar colecciones utilizando í
  - Hace el código más limpio y conciso al inicializar colecciones.
  - Facilita la lectura y el mantenimiento de estructuras complejas.
- **Mejoras en Grupos de Métodos.** Los grupos de métodos ahora tienen inferencia delegados.
  - Reduce la necesidad de conversiones explícitas de tipos.
  - Mejora la compatibilidad con delegados.

Method Group Natural Type

Esta característica introduce mejoras en la resolución de sobrecargas relacionadas con grupos de métodos, optimizando el comportamiento del compilador al determinar el tipo natural de un grupo de métodos. Un grupo de métodos es un conjunto de métodos que comparten el mismo nombre, incluyendo todas sus sobrecargas. Antes de C# 13, la resolución de sobrecargas involucraba construir un conjunto completo de métodos candidatos antes de determinar el tipo natural, lo cual podía ser ineficiente y generar confusión en ciertos escenarios.

¿Qué cambia?

1. Optimización del Proceso de Resolución:
   - El compilador prunea (descarta) métodos no aplicables en cada ámbito (scope).
   - Métodos genéricos con aridad incorrecta o restricciones insatisfechas son eliminados temprano.
   - Si no se encuentra un candidato en el ámbito actual, se pasa al siguiente ámbito externo.
2. Comportamiento más Consistente:
   - Sigue más de cerca el algoritmo general de resolución de sobrecargas.
   - Si no hay coincidencia en un ámbito dado, se considera que el grupo de métodos no tiene un tipo natural.

¿Qué es un Tipo Natural?

Un tipo natural en este contexto es el tipo de delegado o expresión lambda que puede inferirse automáticamente para un grupo de métodos. Por ejemplo, al pasar un grupo de métodos a un delegado, el compilador necesita determinar cuál de los métodos del grupo (si existe) es aplicable y compatible.

Características de C# 13

- **Colecciones params.** el modificador **params** se ha ampliado para admitir tipos de colección adicionales, no solo arreglos tradicionales. Esto incluye interfaces de colección genéricas e incluso tipos como **System.Span<T>** y **System.ReadOnlySpan<T>.**
  - Simplifica el código cuando trabajas con métodos que aceptan múltiples parámetros.
  - Mejora la legibilidad y reduce errores en código repetitivo.
- **Nuevo Tipo y Semántica para lock**. Introduce un nuevo tipo y reglas revisadas para el uso de **lock**, lo que mejora la sincronización multihilo.
  - Aumenta la claridad al escribir código que utiliza sincronización.
  - Reduce la posibilidad de errores al trabajar con recursos compartidos.
- **Nueva Secuencia de Escape \e.** Soporte para la secuencia de escape **\e**, que representa el carácter ESC (ASCII 27).
  - Útil para escenarios que requieren caracteres de control, como terminales o protocolos de red.
  - Facilita la manipulación de cadenas para sistemas heredados.
- **Inicializadores de Índices Implícitos.** Ahora es posible inicializar colecciones utilizando índices directamente en la definición, sin métodos adicionales.
  - Hace el código más limpio y conciso al inicializar colecciones.
  - Facilita la lectura y el mantenimiento de estructuras complejas.
- **Mejoras en Grupos de Métodos.** Los grupos de métodos ahora tienen inferencia de tipos más precisa y natural, facilitando su uso en expresiones lambda y delegados.
  - Reduce la necesidad de conversiones explícitas de tipos.
  - Mejora la compatibilidad con delegados.
- *ref locals* **y Contextos** *unsafe* **en Métodos Asíncronos.** Permite el uso de **ref locals** y contextos **unsafe** dentro de iteradores y métodos **async**.
  - Habilita escenarios avanzados de programación de alto rendimiento.
  - Mejora la flexibilidad en métodos asíncronos.

Características de C# 13

- **Colecciones params.** el modificador **params** se ha ampliado para admitir tipos de colección a de colección genéricas e incluso tipos como **System.Span<T>** y **System.ReadOnlySpan<T>.**
  - Simplifica el código cuando trabajas con métodos que aceptan múltiples parámetros.
  - Mejora la legibilidad y reduce errores en código repetitivo.
- **Nuevo Tipo y Semántica para lock**. Introduce un nuevo tipo y reglas revisadas para el us
  - Aumenta la claridad al escribir código que utiliza sincronización.
  - Reduce la posibilidad de errores al trabajar con recursos compartidos.
- **Nueva Secuencia de Escape \e.** Soporte para la secuencia de escape **\e**, que representa e
  - Útil para escenarios que requieren caracteres de control, como terminales o protocolos c
  - Facilita la manipulación de cadenas para sistemas heredados.
- **Inicializadores de Índices Implícitos.** Ahora es posible inicializar colecciones utilizando í
  - Hace el código más limpio y conciso al inicializar colecciones.
  - Facilita la lectura y el mantenimiento de estructuras complejas.
- **Mejoras en Grupos de Métodos.** Los grupos de métodos ahora tienen inferencia d delegados.
  - Reduce la necesidad de conversiones explícitas de tipos.
  - Mejora la compatibilidad con delegados.
- *ref locals* **y Contextos** *unsafe* **en Métodos Asíncronos.** Permite el uso de **ref locals** y c
  - Habilita escenarios avanzados de programación de alto rendimiento.
  - Mejora la flexibilidad en métodos asíncronos.

En C# 13, se han relajado algunas restricciones sobre el uso de variables locales de tipo **ref** y contextos **unsafe** en métodos que utilizan **async** o **yield**. Estas mejoras permiten un uso más flexible y seguro de estructuras como **System.Span<T> y System.ReadOnlySpan<T>**, que son esenciales para la programación de alto rendimiento.

¿Qué Cambia?

1. Antes de C# 13:
   - Los métodos iteradores (**yield return**) y los métodos asíncronos (**async**) no podían declarar:
     - Variables locales **ref**.
     - Variables locales de tipo **ref struct** (por ejemplo, **Span<T>** o **ReadOnlySpan<T>**).
   - Los contextos **unsafe** no eran permitidos en métodos iteradores.
2. Con C# 13:
   - Ahora puedes:
     - Declarar variables locales **ref** en métodos iteradores y asíncronos.
     - Usar tipos de estructuras **ref struct** como **Span<T>** y **ReadOnlySpan<T>.**
     - Usar contextos **unsafe** en métodos iteradores.
   - Restricciones:
     - Las variables locales **ref** no pueden cruzar un límite **await** o un límite **yield return.**
     - Las instrucciones **yield return** y **yield break** deben estar en contextos seguros (no **unsafe**).

Características de C# 13

- **Colecciones params.** el modificador **params** se ha ampliado para admitir tipos de colección adicionales, no solo arreglos tradicionales. Esto incluye interfaces de colección genéricas e incluso tipos como **System.Span<T>** y **System.ReadOnlySpan<T>.**
  - Simplifica el código cuando trabajas con métodos que aceptan múltiples parámetros.
  - Mejora la legibilidad y reduce errores en código repetitivo.
- **Nuevo Tipo y Semántica para lock**. Introduce un nuevo tipo y reglas revisadas para el uso de **lock**, lo que mejora la sincronización multihilo.
  - Aumenta la claridad al escribir código que utiliza sincronización.
  - Reduce la posibilidad de errores al trabajar con recursos compartidos.
- **Nueva Secuencia de Escape \e.** Soporte para la secuencia de escape **\e**, que representa el carácter ESC (ASCII 27).
  - Útil para escenarios que requieren caracteres de control, como terminales o protocolos de red.
  - Facilita la manipulación de cadenas para sistemas heredados.
- **Inicializadores de Índices Implícitos.** Ahora es posible inicializar colecciones utilizando índices directamente en la definición, sin métodos adicionales.
  - Hace el código más limpio y conciso al inicializar colecciones.
  - Facilita la lectura y el mantenimiento de estructuras complejas.
- **Mejoras en Grupos de Métodos.** Los grupos de métodos ahora tienen inferencia de tipos más precisa y natural, facilitando su uso en expresiones lambda y delegados.
  - Reduce la necesidad de conversiones explícitas de tipos.
  - Mejora la compatibilidad con delegados.
- *ref locals* **y Contextos** *unsafe* **en Métodos Asíncronos.** Permite el uso de **ref locals** y contextos **unsafe** dentro de iteradores y métodos **async**.
  - Habilita escenarios avanzados de programación de alto rendimiento.
  - Mejora la flexibilidad en métodos asíncronos.

La lista completa en https://learn.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-13

05

Análisis comparativo Net Framework y NET

Análisis comparativo Net Framework y NET

Características	NET	.NET Framework
Plataforma o Framework	.NET es una plataforma modular y moderna que soporta múltiples frameworks como ASP.NET Core y MAUI, ampliando funcionalidades de desarrollo.	.NET Framework es un framework monolítico y completo, que incluye todas las APIs necesarias en un solo paquete.
Código Abierto	.NET es completamente de código abierto, con soporte activo de la comunidad y Microsoft en GitHub.	.NET Framework incluye solo componentes parciales de código abierto y está más cerrado en términos de evolución.
Multiplataforma	Compatible con Windows, Linux y macOS, además de otros dispositivos como iOS y Android.	Solo funciona en Windows, limitando el desarrollo a una única plataforma.
Modelos de Aplicación	Soporta aplicaciones modernas como web, microservicios, cloud y móviles (Xamarin y MAUI).	Diseñado para aplicaciones de escritorio tradicionales como WinForms y WPF.
Soporte de Microservicios	Ideal para microservicios con herramientas como Docker y Kubernetes.	Difícil de integrar en arquitecturas de microservicios debido a su enfoque monolítico.
Soporte REST	Compatible con gRPC y APIs REST modernas, pero no incluye WCF.	Incluye soporte completo para WCF, aunque limitado para APIs REST modernas.
Desempeño y Escalabilidad	Diseñado para alto rendimiento con compilación AOT y optimización JIT.	Menor escalabilidad debido a limitaciones de diseño arquitectónico.

Análisis comparativo Net Framework y NET

Características	NET	.NET Framework
Compatibilidad	Compatible con bibliotecas modernas y migración simplificada desde versiones anteriores de .NET Core.	Difícil de migrar aplicaciones debido a la incompatibilidad con versiones modernas.
Seguridad	Mejoras en criptografía moderna y soporte para autenticación avanzada en aplicaciones distribuidas.	Incluye mecanismos de seguridad básicos como acceso a código y autenticación Windows.
Herramientas de CLI	Incluye una interfaz de línea de comandos ligera y multiplataforma para desarrolladores.	Pesado y centrado en IDE como Visual Studio; limitado soporte para CLI.
Enfoque en Dispositivos	Soporte para dispositivos IoT, móviles, juegos y machine learning con bibliotecas avanzadas.	Limitado a aplicaciones Windows y no compatible con dispositivos IoT o móviles.
Desarrollo Móvil	Compatible con MAUI y Xamarin para aplicaciones móviles multiplataforma.	No soporta desarrollo móvil, lo que limita su utilidad en proyectos modernos.
Modelo de Despliegue	Flexible y modular, permite despliegues independientes con actualizaciones rápidas.	Centralizado, requiere despliegue en IIS o servidores específicos.
Empaquetado y Entrega	Distribuye bibliotecas como paquetes NuGet, lo que permite actualizaciones granulares.	Todas las bibliotecas se empaquetan juntas, limitando la flexibilidad.

¿Cuándo elegir .NET o .NET Framework?

Elige .NET si:

- Necesitas compatibilidad multiplataforma para Windows, Linux, macOS o dispositivos móviles.
- Estás desarrollando aplicaciones modernas como microservicios, aplicaciones en la nube o APIs REST.
- Requieres alto rendimiento y escalabilidad en contenedores o entornos distribuidos.
- Prefieres usar herramientas modernas de desarrollo, como CLI ligera y NuGet.
- Necesitas desarrollo móvil con Xamarin o MAUI.
- Planeas trabajar con tecnologías emergentes como IoT, IA o machine learning.

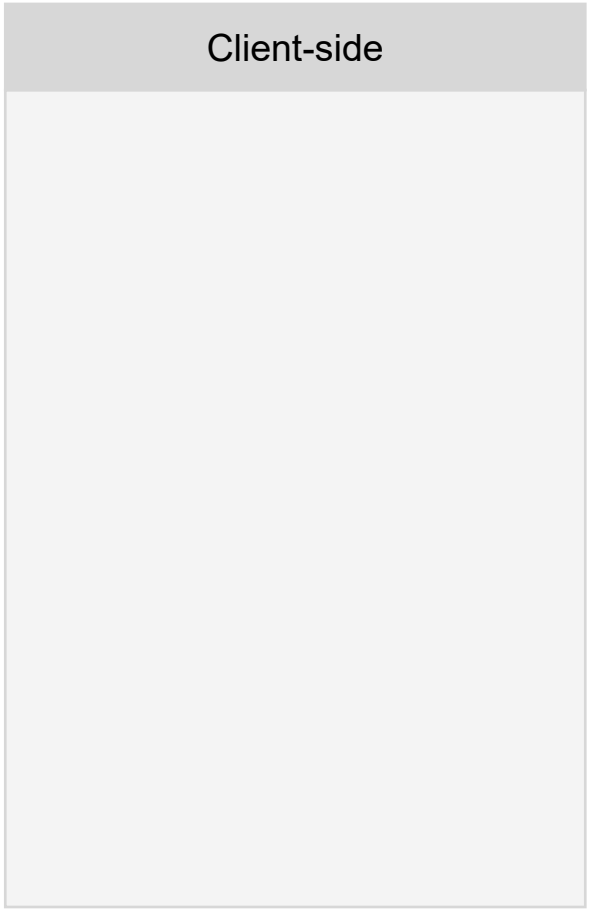
Elige .NET Framework si:

- Tu aplicación debe ejecutarse exclusivamente en Windows.
- Estás trabajando con un proyecto existente que utiliza WinForms, WPF o WCF.
- Necesitas un entorno estable y probado para aplicaciones empresariales **legacy**.
- Requieres soporte completo para Windows Communication Foundation (WCF).
- No planeas migrar a una arquitectura moderna o multiplataforma.

06

Introducción a Full-Stack Architecture con NET

GALAXY
TRAINING

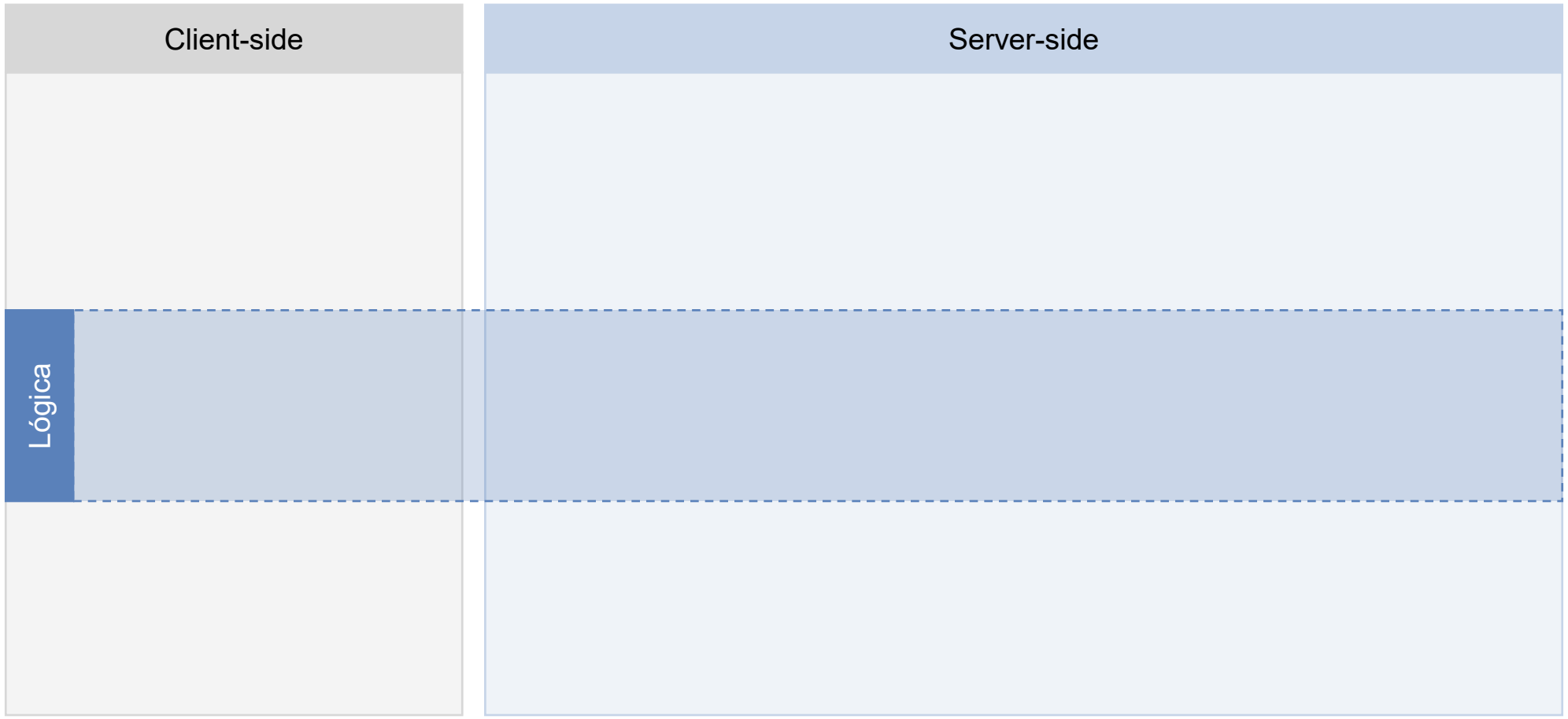

Client-side

Introducción a Full-Stack Architecture con NET 8

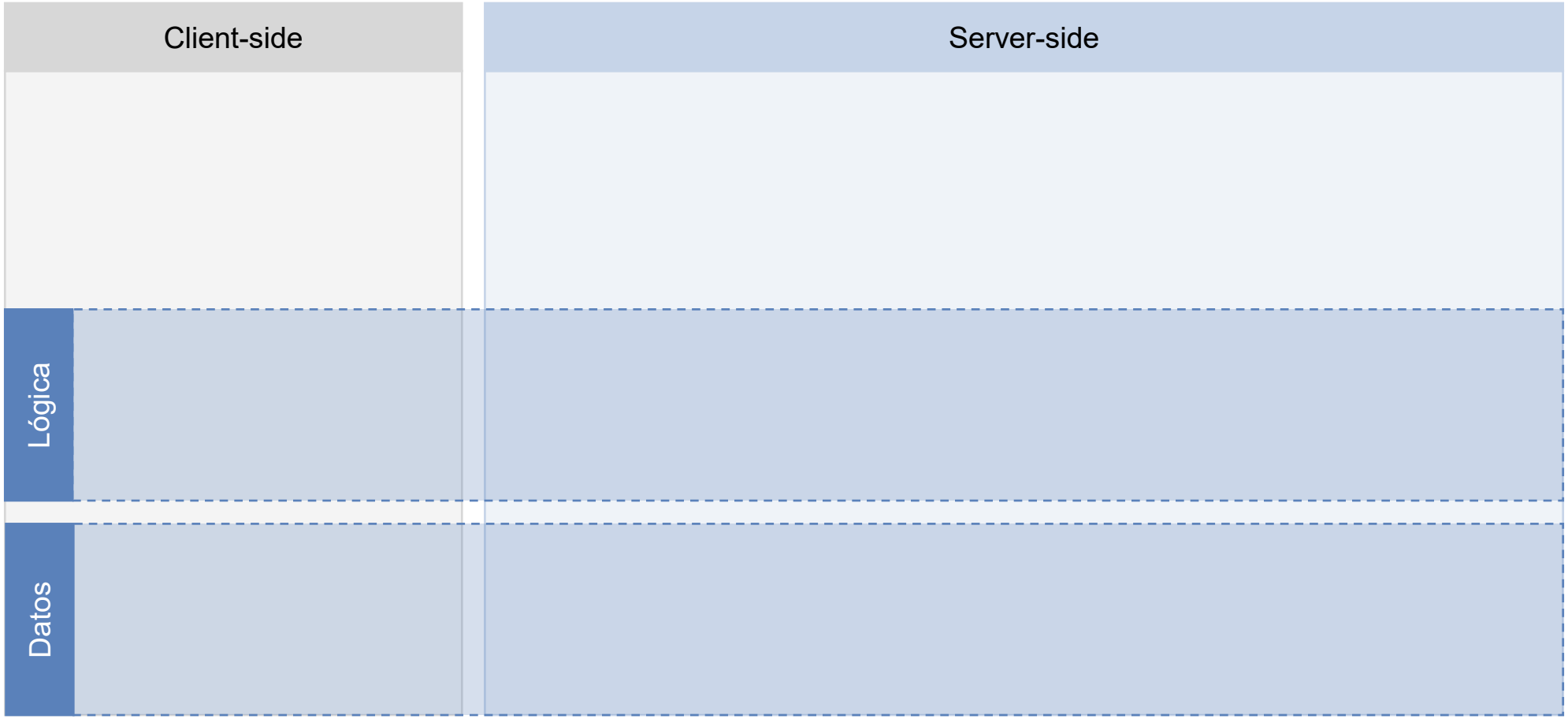
GALAXY TRAINING

Client-side	Server-side

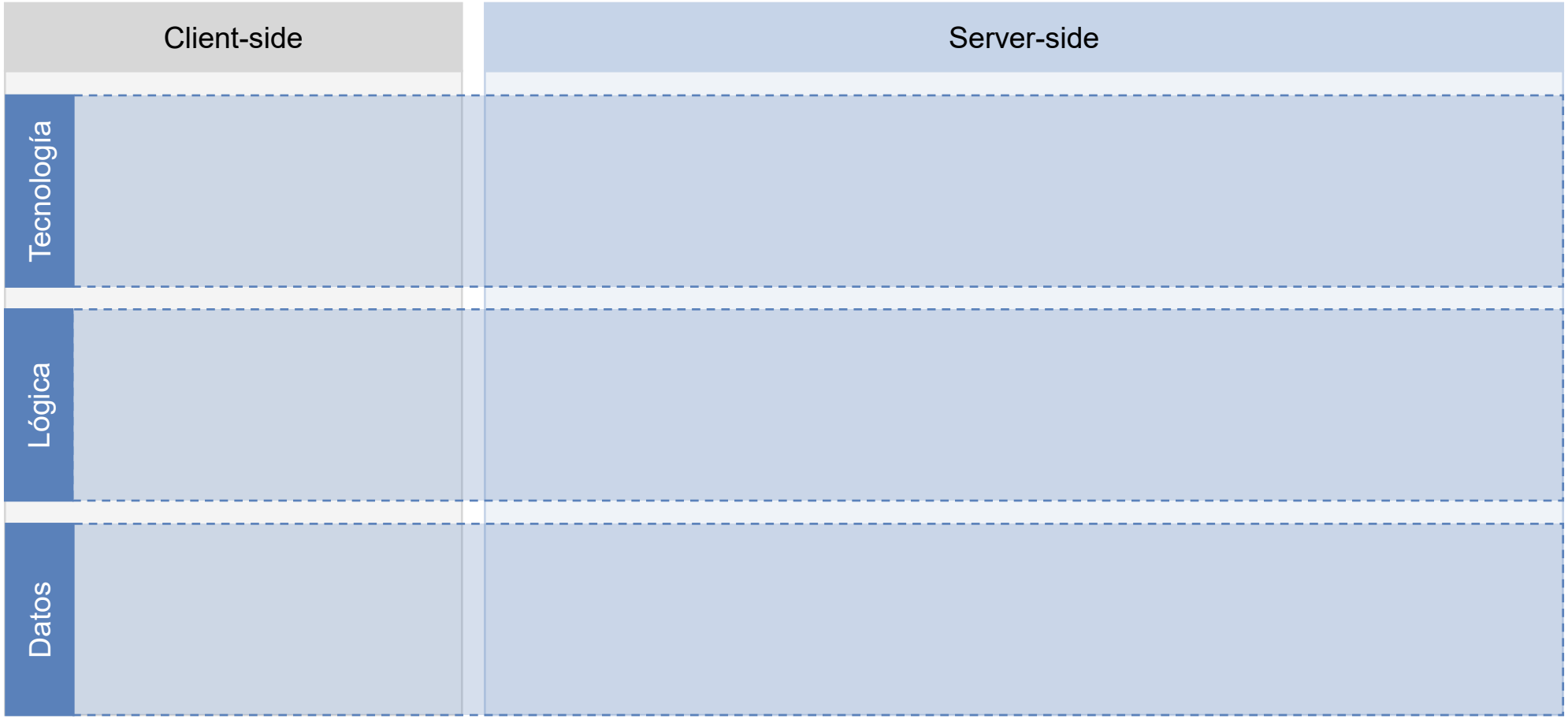
Introducción a Full-Stack Architecture con NET 8

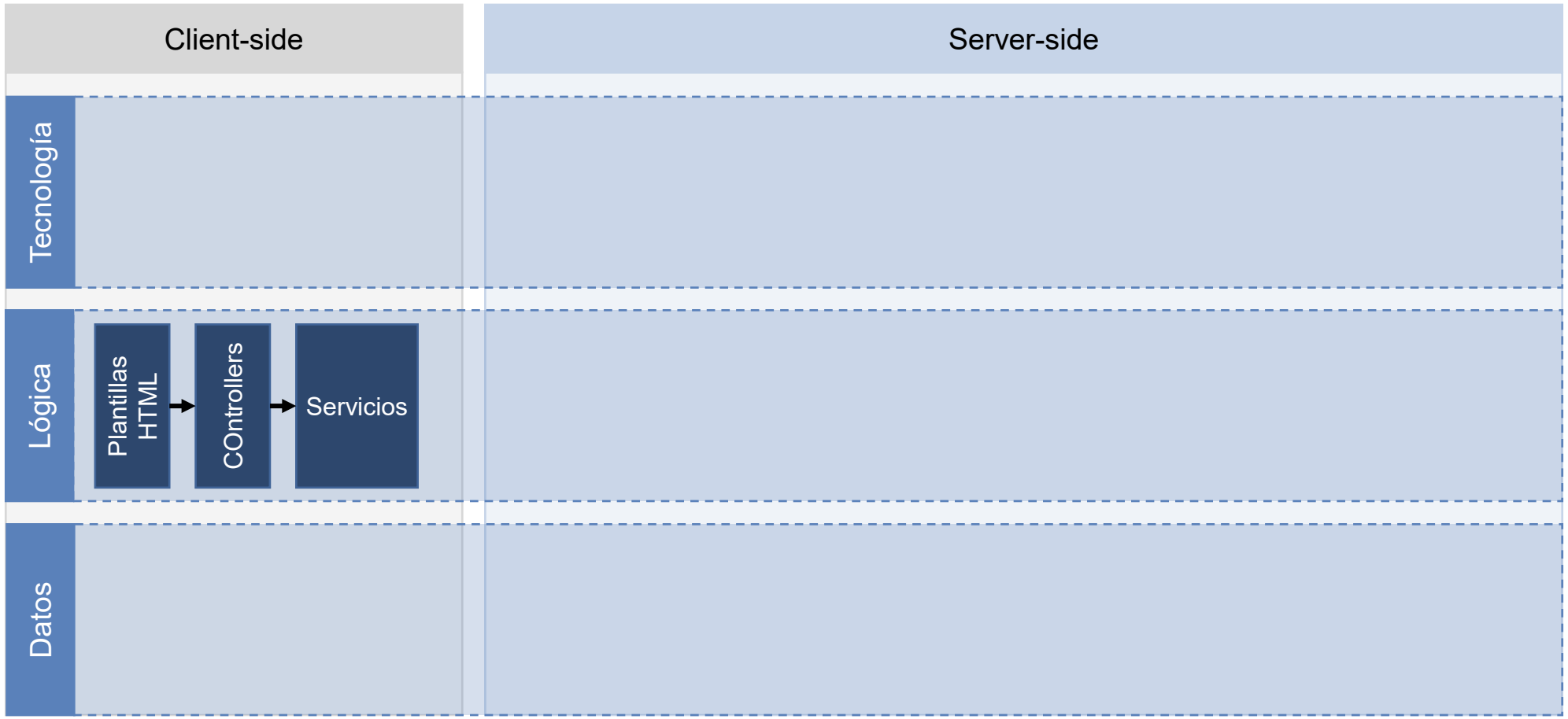


Introducción a Full-Stack Architecture con NET 8

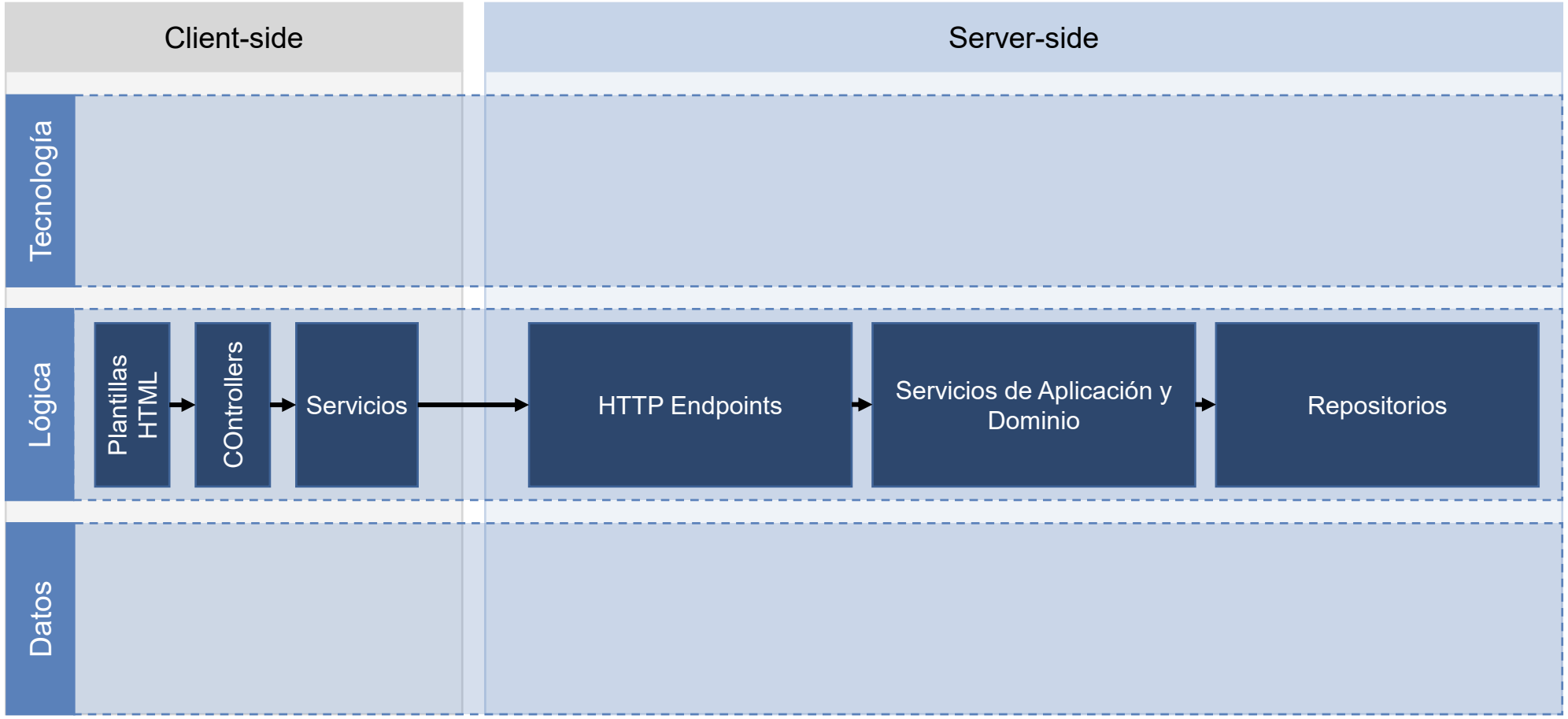


Introducción a Full-Stack Architecture con NET 8

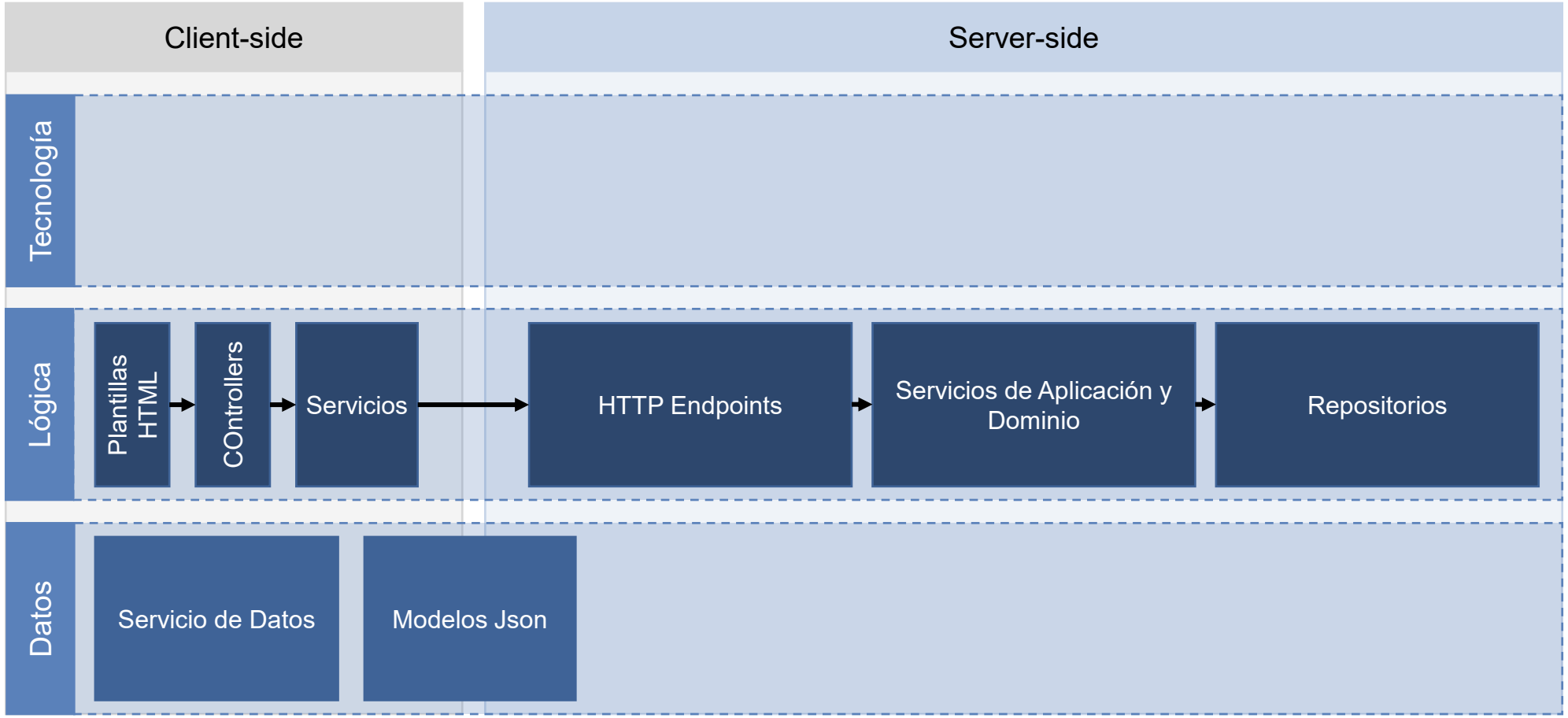



Client-side
Server-side
Tecnología
Lógica
Plantillas HTML
COntrollers
Servicios
Datos

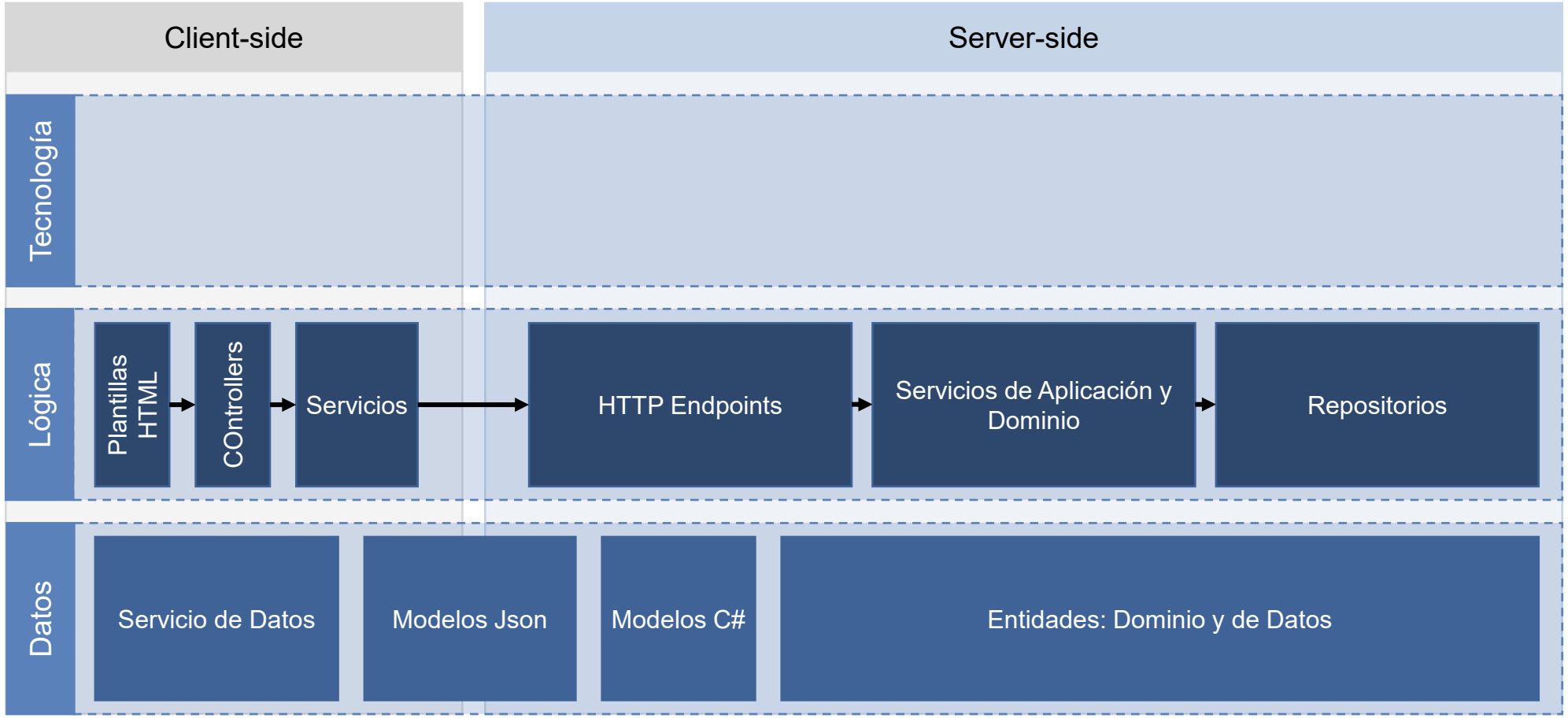
Introducción a Full-Stack Architecture con NET 8



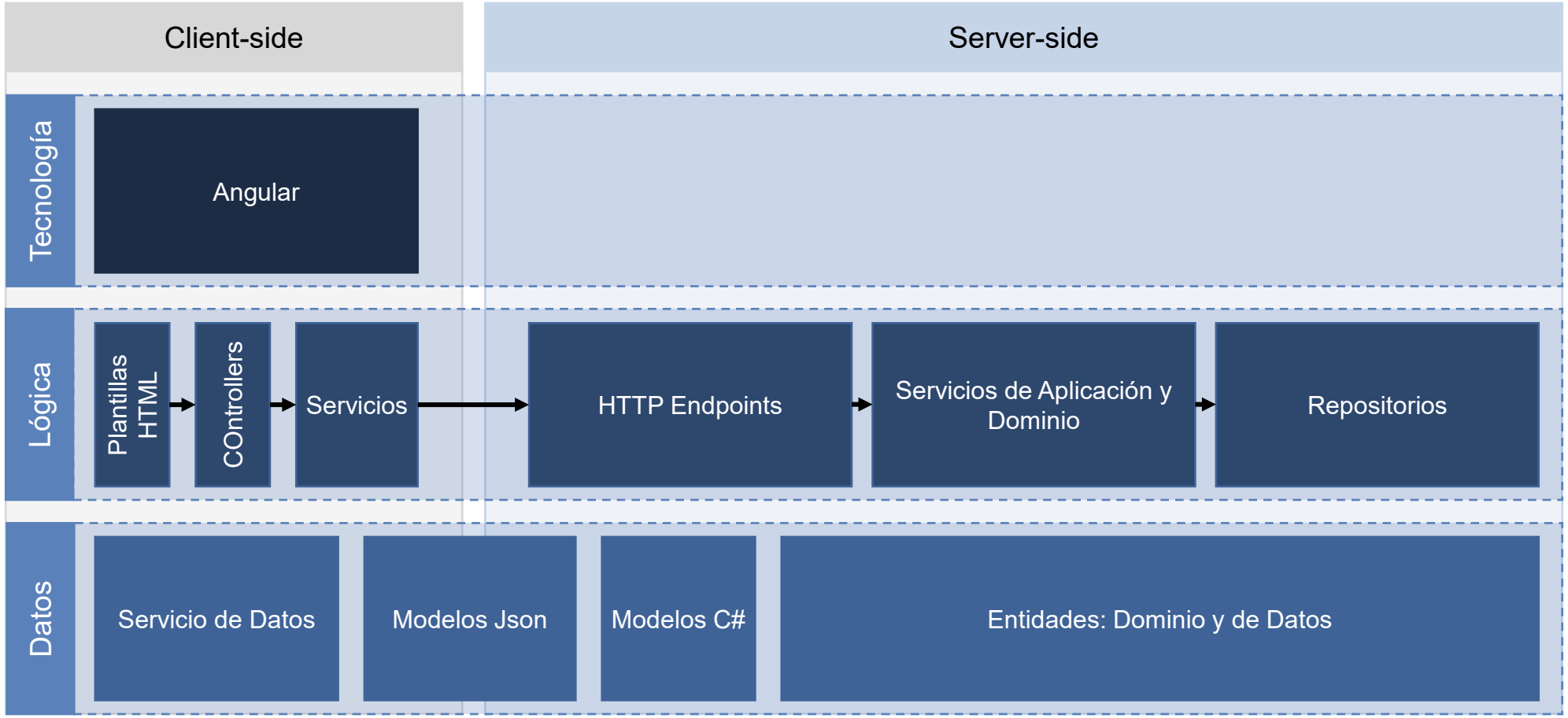
Introducción a Full-Stack Architecture con NET 8



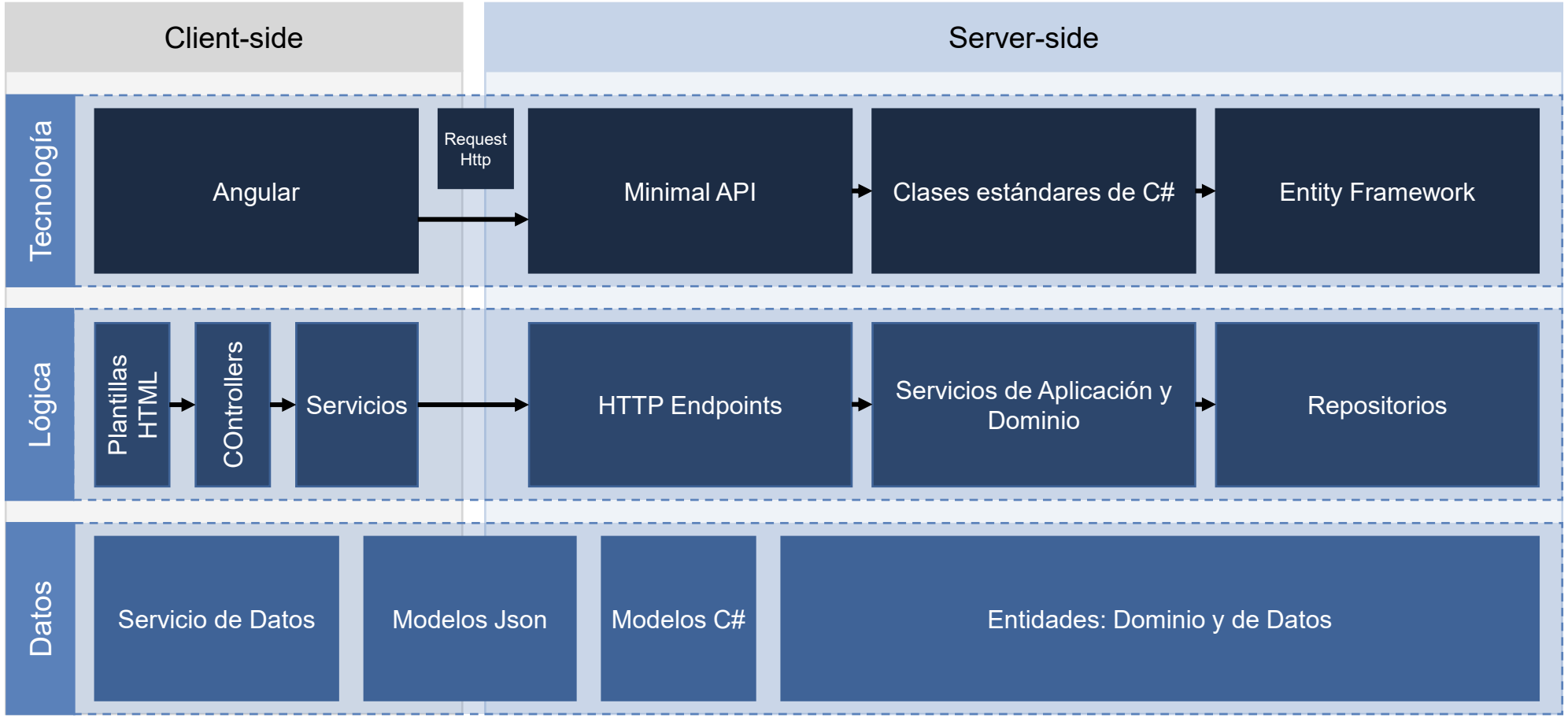
Introducción a Full-Stack Architecture con NET 8



Introducción a Full-Stack Architecture con NET 8



Introducción a Full-Stack Architecture con NET 8



GRACIAS

POR SU PREFERENCIA

GALAXY
TRAINING