1. .NET Versions

The .NET platform has undergone significant evolution since its initial release by Microsoft in 2002. It is now divided into two primary product lines:

.NET Framework (Windows-only):

- Versions: 1.0 to 4.8.1 (latest)
- Platform: Windows
- Use Case: Legacy enterprise applications, Windows Forms, ASP.NET Web Forms
- Status: Still supported but in maintenance mode (no new major features)

.NET Core and .NET (Cross-platform, Modern):

- Versions:
  - .NET Core 1.x (2016)
  - .NET Core 2.x
  - .NET Core 3.x (last version under "Core" branding)
- Rebranded as .NET starting from version 5:
  - .NET 5 (2020)
  - .NET 6 (Long-Term Support - 2021)
  - .NET 7 (2022)
  - .NET 8 (Long-Term Support - 2023)
  - .NET 9 (Planned - 2024)

Version Comparison Table:

Version	Release Year	LTS	Cross-Platform	Notes
.NET Framework 4.8	2019	No	No	Windows-only
.NET Core 1.0	2016	Yes	Yes	Initial cross-platform release
.NET Core 3.1	2019	Yes	Yes	Introduced desktop support
.NET 5	2020	No	Yes	Start of platform unification
.NET 6	2021	Yes	Yes	Long-Term Support (LTS)
.NET 7	2022	No	Yes	Feature and performance updates
.NET 8	2023	Yes	Yes	LTS, cloud-native features

2. .NET Namespaces

Namespaces in .NET are used to organize code and prevent naming conflicts by logically grouping related classes, interfaces, enums, and other types.

Common .NET Namespaces:

Namespace	**Description**
System	Core classes like String, Math, DateTime
System.Collections	Generic and non-generic collections
System.IO	Input/output, file and stream manipulation
System.Net	Network communications (HTTP, FTP, etc.)
System.Linq	LINQ query syntax and operators
System.Threading	Multithreading, async, and concurrent programming
Microsoft.AspNetCore	ASP.NET Core web framework
System.Text.Json	JSON serialization and deserialization

Namespaces are implemented through assemblies (DLLs), which are referenced in project files or through NuGet packages.

3. .NET Core and .NET 5+

Key Features:

- Cross-platform (Windows, Linux, macOS)
- Modular architecture (via NuGet)
- High-performance runtime and JIT compiler
- Built with container support (Docker, Kubernetes)
- Unified platform for cloud, desktop, web, and mobile development

Supported Workloads:

- ASP.NET Core (REST APIs, MVC, Razor Pages)
- Blazor (C# in browser via WebAssembly)
- Windows Forms and WPF (Windows desktop)

- .NET MAUI (cross-platform mobile and desktop)
- Console applications
- Background services and microservices
- ML.NET for machine learning
- Entity Framework Core for data access

---

4. Solutions in .NET

A solution (.sln file) is a container that can hold one or more related projects, typically structured to separate responsibilities like web UI, business logic, data access, and tests.

Example Solution Structure:

scss

CopyEdit

```
MyApp.sln
├── MyApp.Web (ASP.NET Core Web Project)
├── MyApp.Core (Business Logic and Models)
├── MyApp.Data (EF Core and Repositories)
└── MyApp.Tests (Unit and Integration Tests)
```

Project Types:

- Class Library: Reusable business logic or helper code
- Console Application: For CLI tools or background tasks
- ASP.NET Core Application: Web APIs or full-stack web apps
- Unit Test Projects: Using XUnit, NUnit, or MSTest

Dependency Management:

- NuGet for third-party and Microsoft packages (e.g., Serilog, AutoMapper, Swashbuckle)
- Built-in Dependency Injection container
- Flexible configuration system via appsettings.json, environment variables, and command-line args