Mastering .NET Architecture: IL, CLR, JIT, and Class Libraries Explained

Introduction: What Really Happens When You Run a C# Program?

If you are writing C# code regularly, you probably don't think twice about pressing F5 or running **dotnet run.** Your application starts, prints some output, maybe opens a window or hits an API - and just works.

But under the hood, a lot is happening.

C# is part of the larger .NET ecosystem, which includes compilers, virtual machines, libraries, and runtime environments. When you run your code, it's not compiled directly to machine instructions right away like languages like C++. It goes through several stages like IL generation, just-in-time (JIT) compilation, and execution inside the CLR (Common Language Runtime).

This article breaks down what happens at each stage: from the moment you write C# code, to how it's compiled, how it runs, and how core components like the CLR, IL, and the Base Class Library (BCL) all fit together.

This guide will help you to understand the full picture.

The Components of .NET Explained

C# - The High-Level Language

- C# is modern, type-safe, object-oriented programming language.
- It is used to write application logic in human readable way.

```
1 Console.WriteLine("Hello, .NET");
```

Compilation to IL (Intermediate Language)

- Your C# code is compiled by the Roslyn compiler (csc.exe) into Intermediate Language (IL).
- IL is a low-level, CPU-agnostic instruction set. IL is the **only** language that the CLR understands.
- This compiled output, including IL code and metadata, is stored in an assembly (.exe or .dll).
  - Assemblies contains:
    - IL instructions
    - Type metadata
    - Resources
    - Security attributes

CLR - The Common Language Runtime

- Think of the CLR as the "virtual machine" of the .NET world.
- It loads your assemblies, manages memory, executes IL, and handles runtime services like:
  - Garbage collection
  - Type safety
  - Security enforcement

    - Thread management
    - Exception handling
- CLR is a virtual machine that executes IL (Intermediate Language) into CPU specific machine instructions.
- We can run C# code everywhere (Windows, Linux and MacOS) where we have installed CLR runtime. From 2016, the .NET is open source and cross-platform so we can use C# on different operating systems. Before that was .NET Framework which is only for Windows.

JIT - Just-in-Time Compilation

- When your application runs, the CLR hands IL code over to the JIT compiler.
- The JIT translates IL into native machine code specific to your CPU.
- This native code is then executed directly by the operating system.

> **When Does JIT Happen?**
>
> - The first time a method is called, the CLR invokes the JIT compiler to compile that method into native code.
> - The compiled code is cached in memory and reused for subsequent calls.
> - This means only the methods that are actually used get compiled - improving startup time.

Concept	Description
JIT Compiler	Convert IL to native code at runtime
Trigger	Runs when a method is called for the first time
Output	Native machine code stored in memory
Benefits	Fast development, runtime optimizations
Trade-offs	Startup delay, higher memory use
Optimization	Tiered compilation improves performance in long-running apps

Base Class Library (BCL) in .NET

The Base Class Library (BCL) is a core part of the .NET. It provides a wide range of foundational classes and types that every .NET application depends on - from simple data types like “string” and “Int32” to complex file I/O, networking, and collections.

Namespace	Description
System	Base types: object, String, Int32, DateTime, etc.
System.Collections	Data structures like: List, Dictionary, Queue, etc.

System.IO	File and stream operations: FileStream, ByteStream, etc.
System.Net	Networking: HTPP, FTP, Sockets.
System.Text	String handling, REGEX, encoding, etc.
System.Threading	Multithreading, tasks and async programming
System.Linq	Language-Integrated Query support
System.Reflection	Runtime type inspection and metadata
System.Security	Encryption, hashing and permissions

Runtime Cycle Summary:

1. Developer writes C# (human readable) code.
2. Compiler (Roslyn) turns it into Intermediate Language (IL).
3. IL + metadata are packaged into an assembly (.exe or .dll).
4. CLR loads the assembly at runtime.
5. JIT compiler translates IL to native instructions for the CPU of the machine.
6. Native machine code is executed on the target hardware.

Workflow diagram: