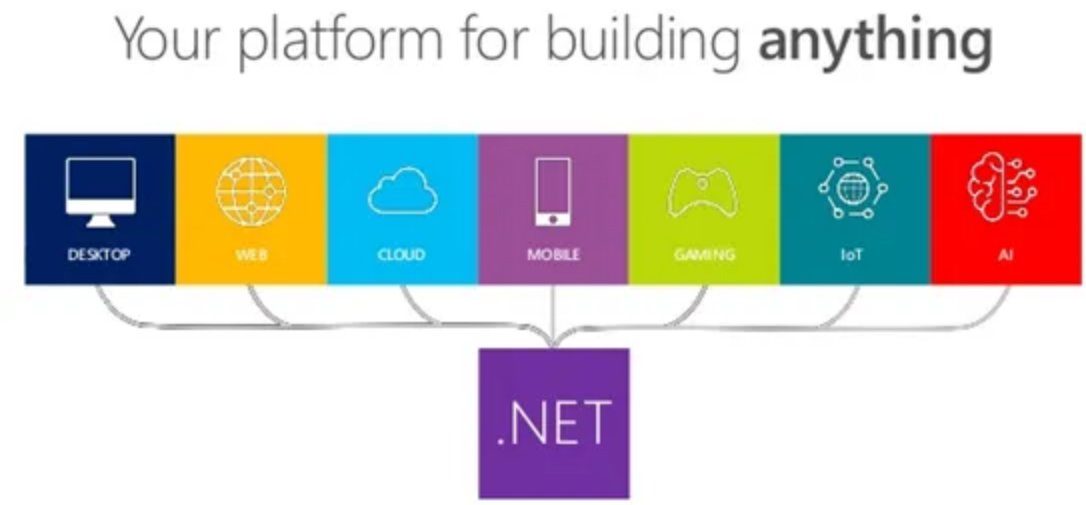
.NET behind the scene

What is it .NET, How it works, and Why it built that way?

Ofir Elarat · Follow
Published in C# Programming · 5 min read · Nov 12, 2020



.NET is a developer platform with tools and libraries for building any type of app.

With .NET, you can use multiple languages, editors, and libraries to build applications for web, mobile, desktop, games, and IoT.

In the late 90's, Microsoft developed .NET framework as a response to the JAVA platform that was developed a few years earlier.

.Net is part of Windows operating system and gets updated on regular basis, and in May 2019 .NET 5.0 was announced.

The .NET platform includes many different parts and a combination of them let you develop and run applications.

The main different parts include:

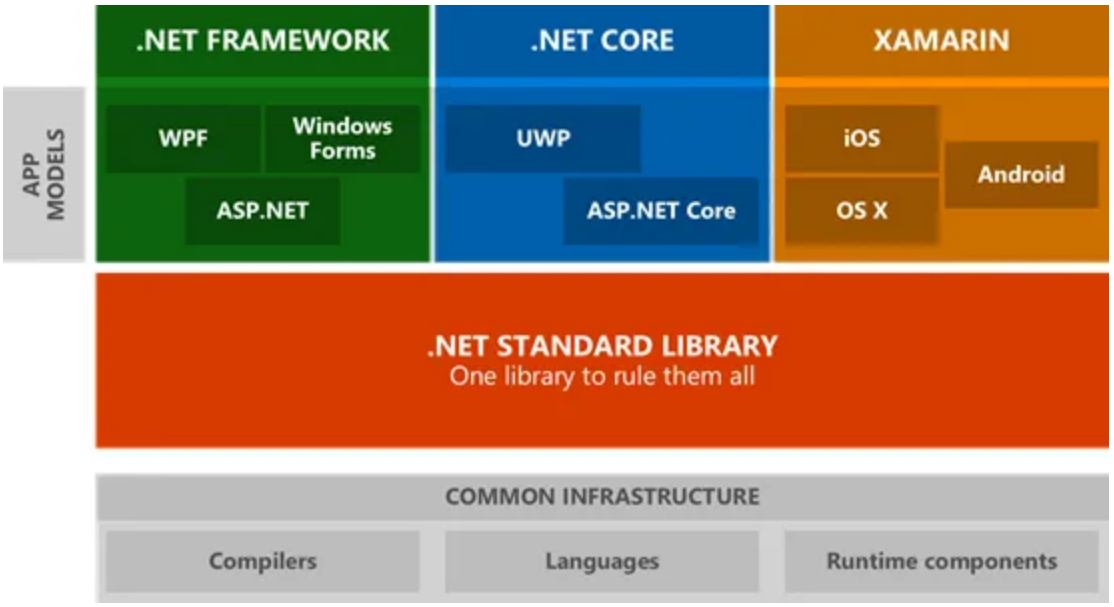
· Programming languages (C#, F#, VB)

· Common libraries (.NET standard)

· Compiler for each programming language (that compile to CIL)

· CIL (Common Infrastructure Language)

· CLR (Common Language Runtime)



Let's explain each one of the .NET components

Programming languages

.NET platform includes many different programming languages, such as C#, F#, and Visual Basic.

Common libraries

.NET Standard is a shared set of libraries that suitable for the various .NET languages and platforms (.NET Core, .NET Framework, Xamarin).

Each .net platform includes class libraries, compilers, and CLR.
The main difference between .NET framework .NET core and Xamarin is the platform each one supports. .NET framework support Windows, Xamarin (based on the Mono project) support Linux, IOS, and Android. And .NET core, the latest one created, supports all of those platforms. The new .NET 5.0 will replace .NET core and will be the only .NET platform in development.

Compilers

Each language in .NET has **Common Language Compiler,** which compiles the code written in some programming language to intermediate language — CIL.

CIL (Common Infrastructure Language)

The CIL is the intermediate language in .NET platform, which means that code in one of .NET languages compiles to an artifact with CIL code instead of machine code.

Also called MSIL.

CLR (Common Language Runtime)

The CLR Is the virtual machine component of .NET, manages the execution of .NET programs. Just-in-time compilation converts the CIL code (compiled intermediate language code), into native code which then executed on the CPU of the computer.

The CLR provides additional services including memory management, type safety, exception handling, garbage collection, security, and thread management. All programs are written for the .NET framework, regardless of programming language, are executed by the CLR.

JIT compiler: as we already understood the JIT compiler converts the intermediate code into native code that runs on the machine.

Advantages:

- The JIT compiler requires less memory usage as only the methods that are required at runtime are compiled into native code by the JIT

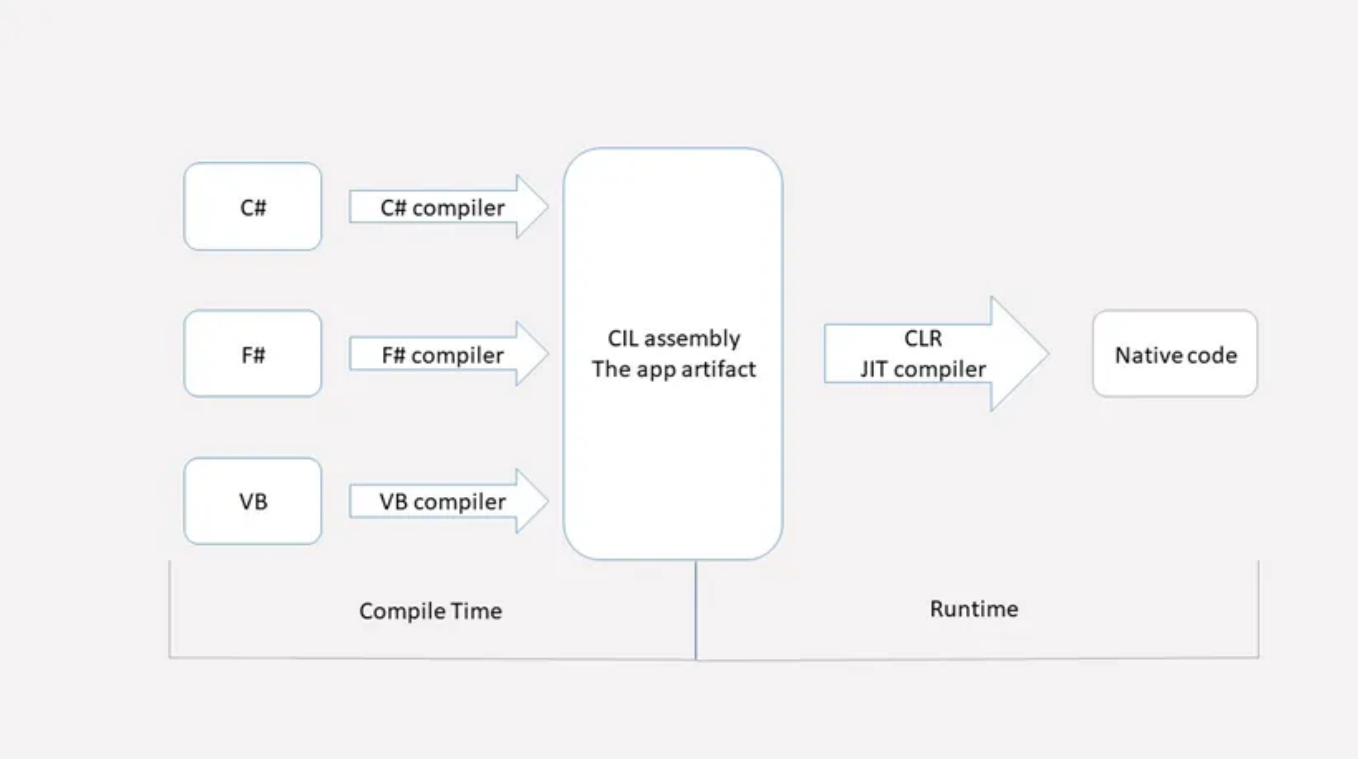
compiler.

- Page faults are reduced by using the JIT compiler as the methods required together are most probably in the same memory page.
- Code optimization based on statistical analysis can be performed by the JIT compiler while the code is running.

Disadvantages:

- The JIT compiler requires more startup time while the application is executed initially.
- The cache memory is heavily used by the JIT compiler to store the source code methods that are required at runtime.

Every time we build and run an application it going through this process.



Let assume we created a new console application in c#.

The computer cannot run the C# code directly and can only run native code.

After we finish writing the app itself with C#, we can build the project to create an artifact using the C# compiler. The artifact we created (DLL or exe) contains CIL code, CIL is an **intermediate language. All .NET language compilers compile the code to CIL artifact.**

When we want to run the artifact **the CLR converts the CIL code to native code** in runtime.

The different between DLL and exe

DLL and exe files are the output artifacts of the common language compilers. They contain the CIL assembly (the CIL code) and CIL metadata. Unlike exe, DLL is not runnable because of one simple reason: every .NET executable file must have an entry point to start from, the main function. DLL doesn't contain this function and therefore not runnable.

Windows run PE (Portable executable) files, such as exe, that contains either native code or CIL code. Because Microsoft is the creator of windows and .NET not like JAVA, they could use the same PE files for .NET assemblies and native assemblies. They accomplished this by adding a header in the assembly that indicates if either or not the CLR needs to run the assembly.

Why compiling to CIL and not to native code

The use of the CIL artifact in .NET platform empowers us for many reasons:

· **Cross Platform** — Native code can run only on a specific machine — the one compiled for. The use of the CIL artifact and CLR means that we can run the

artifact in any machine that contains the CLR engine.

· **Cross Language Integration** — The ability to easily use components developed in other languages as long they are part of .NET languages.

· **Garbage Collector** — The CLR engine manages the application memory, it's enabled because the CLR is a runtime engine. This reason helps full, makes the code simple decreasing chances for bugs, this kind of code calls managed code.

.NET's garbage collector manages the allocation and release of memory for your application. Each time you create a new object, the CLR allocates memory for the object from the managed heap. As long as address space is available in the managed heap, the CLR continues to allocate space for new objects. However, memory is not infinite. Eventually, the garbage collector must perform a collection in order to free some memory.

· **Error handling** — Thanks to the JIT compiler, .NET platform can support easy error handling in your application code. When an error occurs, instead of terminate the process the CLR throw error event.

· **Thread management** — In the .Net platform, the CLR is responsible for allocate resources for running applications. In particular, the CLR thread pool determines when threads are to be added or taken away.

· **Performance optimization** — When converting the CIL code to native code The JIT compiler optimizes the code for the machine platform and CPU, and therefore the application performance improved.

Conclusion

The .NET platform architecture is very innovative and let us write the code once and run it on many platforms. This idea was implemented similarly in different frameworks such as JAVA. I believe that knowing the process and why it works that way makes you a better developer that understand the process and use its advantages better.

Hope you enjoy reading this article.

Thanks.

For more information and extra reading:

· *Assemblies in .NET*

· *CLR overview*

Managed Execution Process

Dotnet | Dotnet Core | Dotnet Framework | Jit Compilers

Written by Ofir Elarat

Follow

26 Followers · Writer for C# Programming

Experienced software engineer, eager to learn more technologies and become better developer.

More from Ofir Elarat and C# Programming

Ofir Elarat

Developing inside a Container

In Container development even in an OFFLINE environment

5 min read · May 2, 2022

4

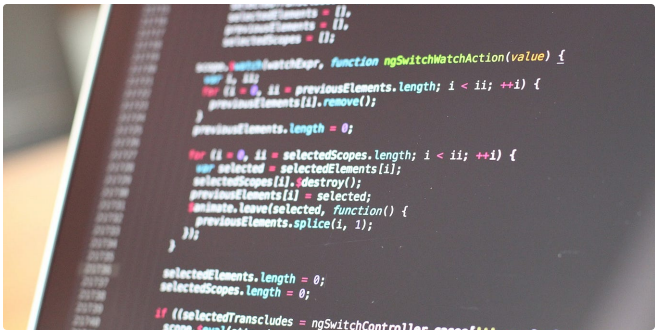
Abnoan Muniz in C# Programming

Serializing objects with Protobuf in .NET 7

Efficient Binary Serialization in .NET with Protobuf

✦ · 6 min read · Nov 1

81 1



Moritz Kross in C# Programming

Ofir Elarat in CodeX

The design behind Express

Understanding the Difference Between Task and Thread in .NET

Spotting the difference in TPL and Thread in .NET

✦ · 3 min read · May 21

500 5

How express routing & middleware work from the design perspective

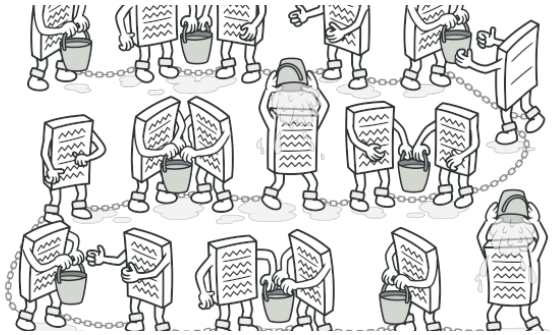
6 min read · Nov 11, 2022

103 1

See all from Ofir Elarat

See all from C# Programming

Recommended from Medium



Mohamed Hendawy

Chain of Responsibility Design Pattern with C# Examples

leveraging proven design patterns can significantly enhance the efficiency and...

10 min read · Jul 15

9

Atakan Kul

List and IEnumerable in C#

Hi! In C#, a List is a class that represents a strongly typed collection of objects that can...

2 min read · Jun 20

30

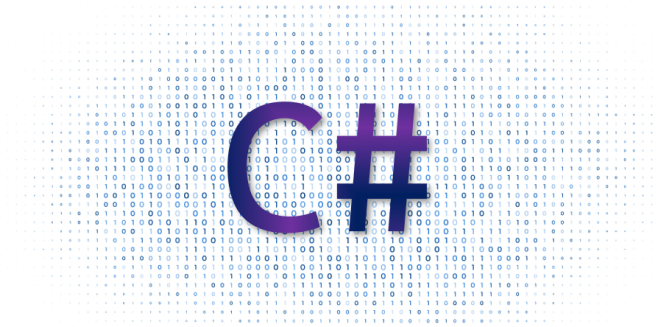
Lists

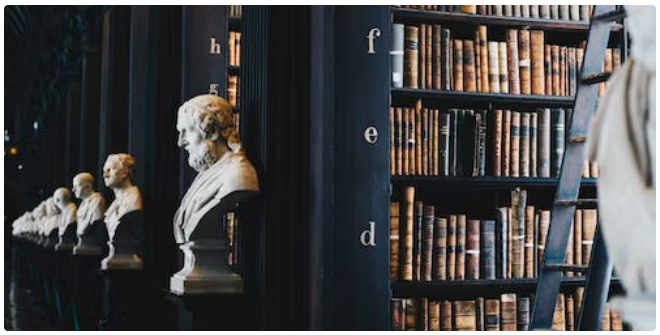
Staff Picks
516 stories · 469 saves

Stories to Help You Level-Up at Work
19 stories · 322 saves

Self-Improvement 101
20 stories · 949 saves

Productivity 101
20 stories · 864 saves





Juldhais Hengkyawan

Some New Features in C# 12 and .NET 8 You Will (Probably) Us...

This post will explore new C# 12 and .NET 8 features that can improve our ASP.NET Core...

3 min read · 5 days ago

153 2

Nikhil Jha

Understanding ConfigureAwait in C#: Managing Asynchronous...

Explore the powerful concept of ConfigureAwait in C#, and learn how to...

· 4 min read · Sep 22

7 2

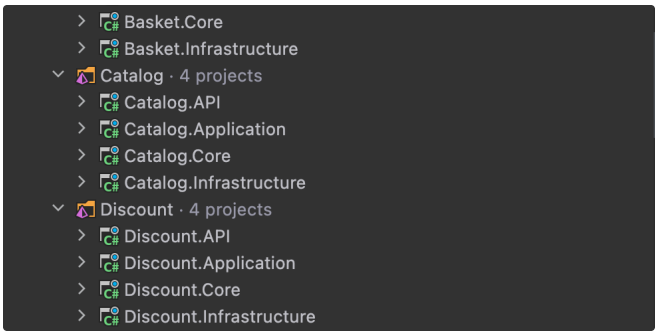
Roko Kovač

Server-Sent Events in .NET

Implementing The Most Elegant HTTP-Based Push Mechanism

4 min read · Nov 18

52 1



rahul sahay in Stackademic

Creating .Net Core Microservices using Clean Architecture

Complete guide to build enterprise edition application end to end

6 min read · Oct 16

437 3

See more recommendations