**What does CLR stand for in .NET?**

CLR stands for Common Language Runtime. It forms to be the most vital component of .NET as it provides the foundation for many applications to run on.

If a user writes an application in C#, it gets compiled and converted to intermediate code. After this, CLR takes up the code and works on it with respect to the following aspects:

* Memory management
* Security Protocols
* Libraries for loading
* [Thread management](https://intellipaat.com/blog/thread-in-java/)

**What is the difference between managed code and unmanaged code in .NET?**

|  |  |
| --- | --- |
| **Managed Code** | **Unmanaged Code** |
| Managed by CLR | Not managed by any entity |
| Garbage collection is used to manage memory | Runtime environment takes care of the management |
| The .NET framework is necessary for the execution | Not dependant on the .NET framework to run |

**How does managed code execute in the .NET framework?**

There are four main steps that include in the execution of the managed code. They are as follows:

1. Choosing a compiler that can execute the code written by a user
2. Conversion of the code into Intermediate Language (IL) using a compiler
3. IL gets pushed to CLR, which converts it into native code using JIT
4. Native code is now executed using the .NET runtime

**What is the difference between system.stringbuilder and system.string in .NET?**

|  |  |
| --- | --- |
| **system.**[**stringbuilder**](https://intellipaat.com/blog/stringbuilder-in-java/) | **system.string** |
| Mutable | Immutable |
| Supports using append | Cannot use the append keyword |

**What is the difference between a function and a stored procedure in .NET?**

|  |  |
| --- | --- |
| **Function** | **Stored Procedure** |
| Can only return one value | Can return any number of values |
| No support for exception handling using try-catch blocks | Supports the usage of try-catch blocks for exception handling |
| The argument consists of only one input parameter | Both input and output parameters are present |
| A function can be called from a stored procedure | The [stored procedure](https://intellipaat.com/blog/stored-procedure-in-sql/) cannot be called from a function |

**What is the meaning of boxing and unboxing in .NET?**

Boxing is the process that is used when a user wishes to convert a value type into a reference type directly.

Unboxing is the opposite of boxing, where the reference type is converted back into a value type.

**What is the meaning of garbage collection?**

Garbage collection is a process that is used to maintain various aspects of memory to prevent memory leaks during program execution.

An entity called the garbage collector is used to allocate and de-allocate memory as and when required by an application. This is done by performing checks on the references of [variables](https://intellipaat.com/blog/tutorial/python-tutorial/python-variables/) and objects used by the application. If an object is no longer required by the application, the memory is de-allocated and freed up.

**Generation 0**: This generation is the youngest and contains short-lived objects. An example of a short-lived object is a temporary variable. Garbage collection occurs most frequently in this generation.

**Generation 1**: This generation contains short-lived objects and serves as a buffer between short-lived objects and long-lived objects.

**Generation 2**: This generation contains long-lived objects. An example of a long-lived object is an object in a server application that contains static data that's live for the duration of the process.

**What is the difference between an abstract class and an interface in .NET?**

|  |  |
| --- | --- |
| **Abstract Class** | **Interface** |
| Provides the partial implementation of functionalities that are done by [inheriting classes](https://intellipaat.com/blog/inheritance-in-cpp/) | Used to declare the behavior of an implementing class |
| Used to declare properties, methods, events, and fields as well | Fields cannot be declared using interfaces |

| **Abstract Class** | **Interface** |
| --- | --- |
| Used to declare properties, events, methods, and fields as well. | Fields cannot be declared using interfaces. |
| Provides the partial implementation of functionalities that must be implemented by inheriting classes. | Used to declare the behavior of an implementing class. |
| Different kinds of access modifiers like private, public, protected, etc. are supported. | Only public access modifier is supported. |
| It can contain static members. | It does not contain static members. |
| Multiple inheritances cannot be achieved. | Multiple inheritances are achieved. |

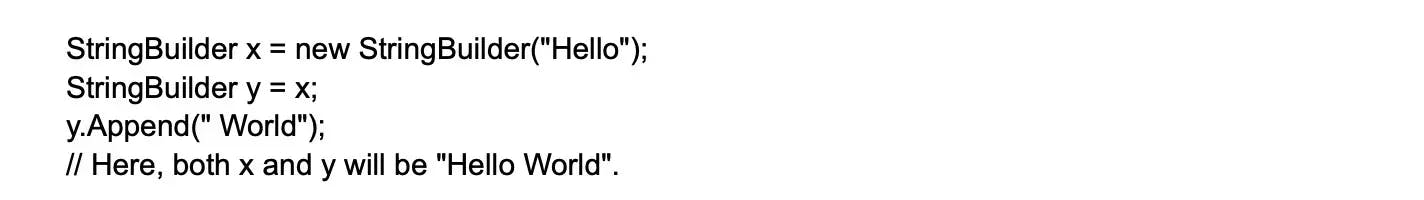
**Explain the difference between value types and reference types in .NET.**

In .NET, data types are divided into two categories: value types and reference types. The primary difference between them lies in how they store their data and how they are handled in memory.

Value types directly contain their data and are stored on the stack. They include primitive types such as int, bool, float, double, char, decimal, enum, and struct. When a value type is assigned to a new variable, a copy of the value is made. Therefore, changes made to one variable do not affect the other.



Reference types, on the other hand, store a reference to the actual data, which is stored on the heap. They include types such as class, interface, delegate, string, and array. When a reference type is assigned to a new variable, the reference is copied, not the actual data. Therefore, changes made to one variable will affect the other, as they both point to the same data.



Understanding the difference between value types and reference types is crucial for efficient memory management and performance optimization in .NET applications.

**What are extension methods in C# and how are they used?**

Extension methods in C# allow developers to add new methods to existing types without modifying their source code. They are defined as static methods within a static class, and the first parameter of the extension method specifies the type being extended, preceded by the 'this' keyword. Extension methods enable adding functionality to types without inheritance or modifying the type hierarchy, making it easier to extend third-party or framework classes.

**Explain the concept of reflection in .NET and its practical applications.**

Reflection in .NET allows for introspection of types, methods, properties, and other members at runtime. It provides the ability to examine and manipulate metadata, dynamically invoke methods, and create instances of types. This is done through the System.Reflection namespace.

Here's a simple example of using reflection to get information about a type:



In this example, we're using reflection to get information about the string type, such as its full name, namespace, and whether it's public.

Reflection is commonly used in scenarios such as dependency injection, serialization, custom attribute processing, and building extensible frameworks. While reflection is a powerful tool, it's also worth noting that it can be slower than using statically-typed code, and can potentially expose sensitive information or methods if used improperly.

**Explain the concept of microservices and how they can be implemented in .NET.**

[Microservices](https://www.turing.com/blog/microservices-best-practices/) is an architectural style that structures an application as a collection of small, autonomous, and loosely coupled services. Each service corresponds to a specific business functionality and can be developed, deployed, and scaled independently. This approach promotes modularity, making the system easier to understand, develop, and test. It also enhances scalability since each service can be scaled individually based on demand. Furthermore, it improves fault isolation: if one service fails, the others can continue to function.

In the .NET ecosystem, microservices can be implemented using ASP.NET Core, a cross-platform, high-performance framework for building modern, cloud-based, internet-connected applications. ASP.NET Core provides features like lightweight APIs, support for containerization (which is crucial for microservices), service discovery mechanisms, and options for synchronous (like HTTP/REST) and asynchronous (like message queues or gRPC) communication between services.

For instance, consider an e-commerce application broken down into several microservices such as User Management, Product Catalog, Order Processing, and Payment. Each of these can be an ASP.NET Core Web API project, developed and deployed independently.

**What are the advantages of using Docker containers for deploying .NET applications?**

Docker containers provide a lightweight and portable runtime environment for deploying applications. Docker offers several advantages, including:

Consistent deployment across different environments: A Docker container bundles the .NET application and its dependencies into a single unit, ensuring it works the same in every environment — whether it's a developer's machine, a test environment, or a cloud infrastructure.

Isolation of dependencies: Each .NET application in a Docker container runs in its own isolated environment. This prevents conflicts between different versions of dependencies used in other applications.

Improved scalability: Docker containers can be quickly started, stopped, and replicated as per the demand. This adaptability enables .NET applications to handle varying traffic loads efficiently.

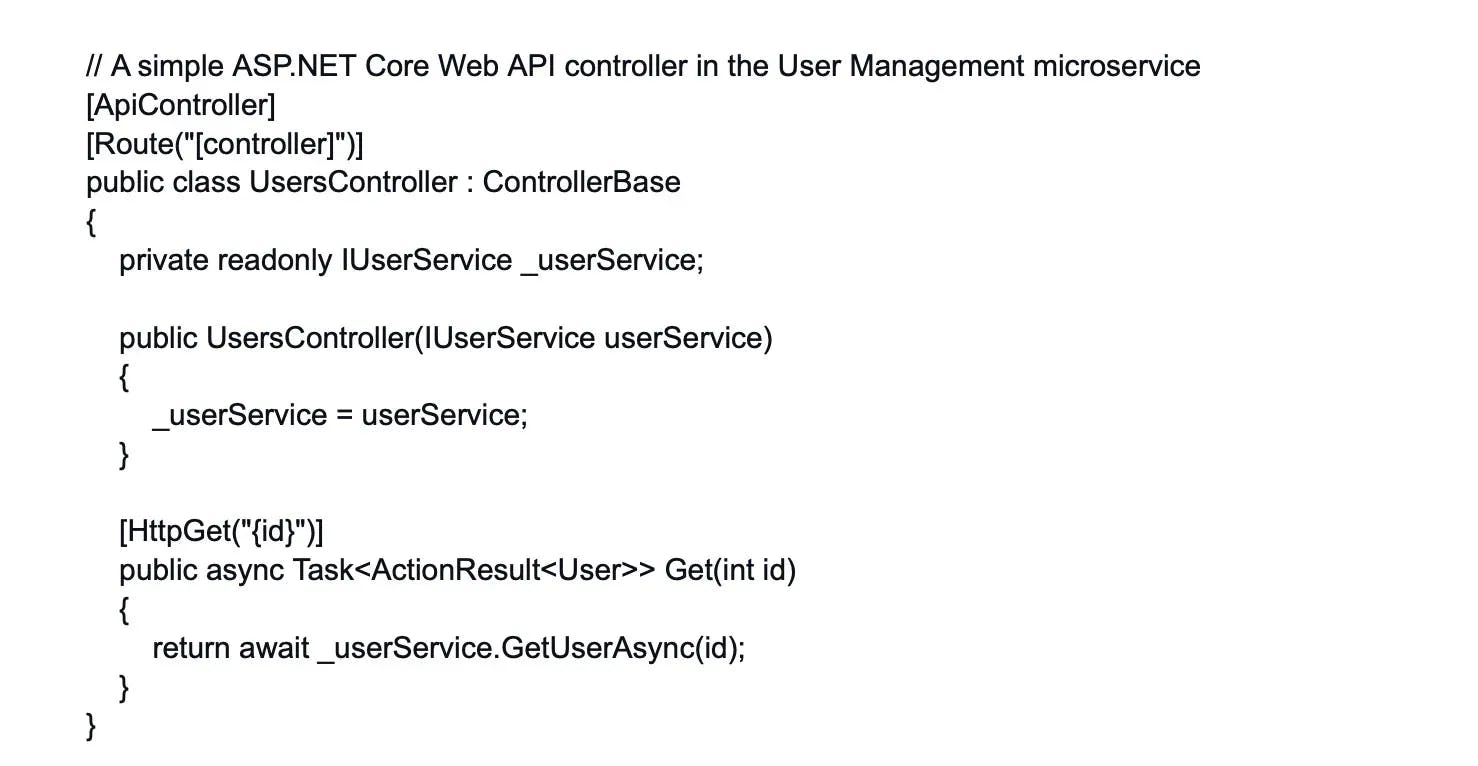
Easy versioning and rollback: Docker image versions facilitate easy application versioning. If a new version has an issue, you can quickly rollback to a previous healthy version.

Simplified deployment automation: With Docker, you can automate the creation of containerized .NET applications by writing a Dockerfile. This feeds into a CI/CD pipeline, simplifying deployment automation.

Efficient resource utilization: Docker containers share the host system OS kernel, making them far less resource-intensive compared to running full-fledged virtual machines.

Facilitates microservices architecture: Docker containers are great for the microservices architecture because they enable each service (like a .NET service) to run in its own container, aiding the independent development, deployment, and scaling of each service.

In conclusion, Docker containers improve the manageability and performance of .NET applications, especially in complex distributed systems and microservices-based architectures.



In this code snippet, we have a simple UsersController in the User Management microservice. It uses dependency injection to get an instance of IUserService, which would contain the business logic for user-related operations.

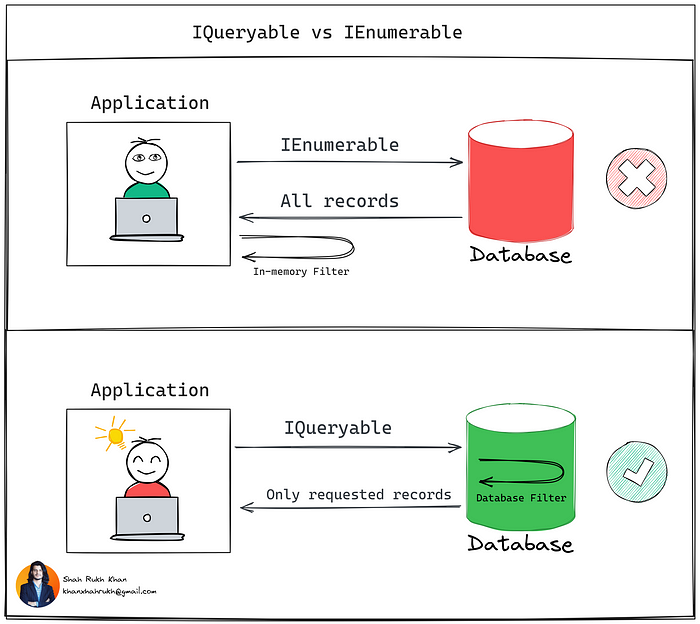
For deploying and managing these microservices, .NET integrates well with containerization tools like Docker and orchestration platforms like Kubernetes, which handle service discovery, load balancing, and scaling.

**Explain the differences between value type and reference type.**

The main differences between value type and reference type are given below:

* A Value Type holds the actual data directly within the memory location and a reference type contains a pointer which consists of the address of another memory location that holds the actual data.
* Value type stores its contents on the stack memory and reference type stores its contents on the heap memory.
* Assigning a value type variable to another variable will copy the value directly and assigning a reference variable to another doesn’t copy the value, instead, it creates a second copy of the reference.
* Predefined data types, structures, enums are examples of value types. Classes, Objects, Arrays, Indexers, Interfaces, etc are examples of reference types.

**IEnumerable and IQueryable in C#**



The primary difference lies in where the query is executed. IEnumerable executes the query in the client's memory, making it suitable for in-memory collections but less efficient for large datasets. In contrast, IQueryable executes the query on the server, allowing for more efficient data handling, especially with large or complex data sets.

Another key difference is their respective use cases. IEnumerable is more straightforward and is used for in-memory data, while IQueryable is more complex and is better suited for remote data sources or situations where performance optimization is crucial.

**When to Use Each**

***Use IEnumerable:***

* For in-memory data collections.
* When working with small to medium datasets.
* When the data source is not a database.

**Use IQueryable:**

* For large datasets or databases.
* When querying data from remote sources like a web service or a database.
* When performance optimization and efficient data handling are required.

**What Is C# Reflection?**

Accessing metadata at runtime is known as a reflection in C#. Using reflection, you may dynamically access and invoke a type's methods, attributes, and events. In addition to reading and calling, reflection is used to construct new types at runtime. The "System.Reflection" namespace contains the [classes](https://www.simplilearn.com/tutorials/asp-dot-net-tutorial/what-is-class-in-c-sharp) that provide access to the program's metadata during runtime.

Classes in the "System.Reflection" namespace are used to learn more about an app's structure and functionality and dynamically add new data types, properties, and objects.

Classes essential for reflection are found in the System.Reflection namespace, including:

* Assembly

An assembly is a recyclable, updatable, and self-describing component of a common language runtime program, and the Assembly class describes it.

* AssemblyName

It Describes the assembly's nth distinguished name.

* MemberInfo

Retrieves details about a member's properties and provides member's metadata.

* MethodInfo

It explains the method class and provides its metadata.

* ConstructorInfo

It explains the constructor class and provides its metadata.

* EventInfo

It explains the event info and provides its metadata.

* PropertyInfo

It identifies a property's features and makes its metadata available.