|  |  |
| --- | --- |
| Routing in .Net Core |  |

In a **.NET Core API**, the term **"container"** typically refers to the **Dependency Injection (DI) container**.

### What is a Container in .NET Core API?

The **DI container** is a **built-in feature** in .NET Core (now .NET 5/6/7/8) that manages **dependency injection**. It is responsible for:

* **Registering** services and their lifetimes
* **Resolving** and **injecting** dependencies at runtime

This container is used to build and manage object lifetimes and dependencies efficiently and consistently.

### Why Use a Container?

* Promotes **loose coupling**
* Makes code **easier to test**
* Supports **Inversion of Control (IoC)**
* Reduces **boilerplate** code

### How it Works

1. **Register Services** in Startup.cs or Program.cs (depending on .NET version):

|  |
| --- |
| services.AddTransient<IMyService, MyService>();  services.AddScoped<IOtherService, OtherService>();  services.AddSingleton<ILogger, Logger>(); |

1. **Inject Dependencies** via constructor:

|  |
| --- |
| public class MyController : ControllerBase  {  private readonly IMyService \_myService;  public MyController(IMyService myService)  {  \_myService = myService;  }  } |

1. The **container automatically resolves** IMyService to MyService when the controller is instantiated.

### Lifetimes in the Container

* **Transient** – A new instance is created every time.
* **Scoped** – One instance per request.
* **Singleton** – One instance for the entire application lifetime.