**Dependency Injection :**

Asp.net core provides built-in dependency injection container. The framework takes a responsibility of creating and managing lifetime of dependency objects.

Dependency Injection is used to configure in **ConfigureServices** **method**() on startup class

Here you add all the services what your application required

In Startup class the startup constructor already takes a dependency **IConfiguration**

That’s configuration object which represents a key value pair. This configuration can be loaded from anywhere including your json files, environment variables,or azure keyword etc.

This dependency is injected in by the asp.net core runtime and there are a few limitations on the types that can be injected in here we can inject in a **iwebhostenvironment, ihostenvironment** which is basically the same as in the web host. We cant inject in any other services through this startup class constructor.

In ConfigureService() we have IServiceCollection services which adds services like AddController()

This method configures the MVC services for the commonly used features with controllers.

The Asp.net runtime also injects in a lot of framework provided services

(You state the need that you need a logger and we are not worried whether it logs to file or console or to an external logging services all we need is a logger to which you can log to by default the asp.net code logger is configured to log to console and a few other providers. With DI we have inverted the dependencies weather forcast controller no longer needs to know how the logger is implemented all it cares about it needs a functionality to log which it gets in through the ilogger interface)

IServiceCollection : it is collection of services descriptor and

A services descriptor describes a service with its service type, implemented and lifetime. It will be used to initializes a new instance of ServiceDescriptor with the specified implementationType. A Service type is usually the interface or an abstract based class it could also be a class in itself The implementation is the class that implements the abstract class or the interface or when it is a class it says the class itself again and also there is a lifetime. At a high level there three life time in asp.net

Transient : if you had written a new and called in the constructor explicitly anywhere you needed an external dependency so with transient the services are created each time they are requested for from the service container scoped lifetime services are created once in the entire request

Scope : Scoped lifetime services are created once in the entire request scope. so within the context of a request if you ask for the same type twice you will get the same instance

Singleton :

////