Net Core DI Truths

References:

NET Core DI Best Practices

Here’s a list of behaviors of the NET Core DI registry:

1. All dynamic access to services (from DI) are requested through a service provider.
   This is what people call a service locator. For example:
   `var foosvc = _serviceprovider.GetService<Foo>();`
2. Every service provider instance is created from a service scope.
   Meaning, every service provider instance has a service scope.
3. A service provider is, itself, a scoped service.
4. A service scope determines the lifetime of the generated service provider.
   Specifically, the service scope determines when the service provider will get disposed.
5. A unique service scope is created for each web request.
   This means, any retrieved services in the call stack of a web request, share the same scope.
   And, they will all get disposed when the web request closes.
6. A scoped service that is used in more than one place, while in the same service scope will be the same instance of that service. A data context is a good example of this.
7. Services that are registered as singleton, are held by the root-container.
   This gives singleton services a service scope of the root-container, which that lasts for the duration of the app’s runtime.
8. The root-container service scope gets disposed when the application runtime shuts down.
   This means, that created singleton instances are also disposed when the application runtime shuts down.
9. Service types that are registered as scoped services, will be disposed with the service scope gets disposed.
   An exception to this would be, DI-registered services that lack the IDisposable interface. This is discussed below.
10. Any transient service that we retrieve from a service provider will have the same scope as the provider. Specifically, when the provider’s service scope is disposed, all transient services will be disposed.
    An exception to this would be, DI-registered services that lack the IDisposable interface. This is discussed below.
11. The instance of a service provider given to an API controller, scoped/transient service, will not only have the same scope as the parent. But, it will be the same instance of service provider.
12. Since ever service provider reference is to the same provider instance, for a given service scope, any accessed scoped service will be the same instance, across the service scope.

13. By contrast, every transient service that is retrieved from a service provider, even within the same scope, will be a unique instance of the transient service.
14. A singleton service that is retrieved from a service provider, will always be provided by the root-container, regardless of the service provider's scope.
    The runtime DI registry ensures this by all service providers having a reference to their parent scope factory, which holds the root-scope provider and service descriptors of all singletons.

Non-Disposed Service (Exception Noted Above)

We noted an exception, above, where a DI-registered service may live beyond the service scope of the provider that gave it to us. This can occur if the service's class type lacks the IDisposable interface.

Just like designing any class type, we need to decide if the class type requires disposal (through IDisposable).
And, adding IDisposable to a type, allows the service scope to automatically dispose of it after use.

However. If we choose for a service type to not implement IDisposable, and we register it as transient or scoped, it could live beyond the service scope that provided its instance.

This may not be a problem, if a service doesn't require disposal. So, it might not be a real issue.
But, there could be interesting logging side-effects if the non-disposed service is used beyond its scoped parent.
For example: Passing a non-disposable, transient-registered service, like a message broker client, from an API controller, down to a spawned async task would mean, the passed broker client could be used beyond the duration and scope of the web request, and there would be no disposed exception to indicate the issue.

Revision #2
Created 8 October 2025 02:41:01 by glwhite
Updated 8 October 2025 02:53:43 by glwhite