The Microsoft Dependency Injection Container

Steve Gordon
MICROSOFT DEVELOPER TECHNOLOGIES MVP

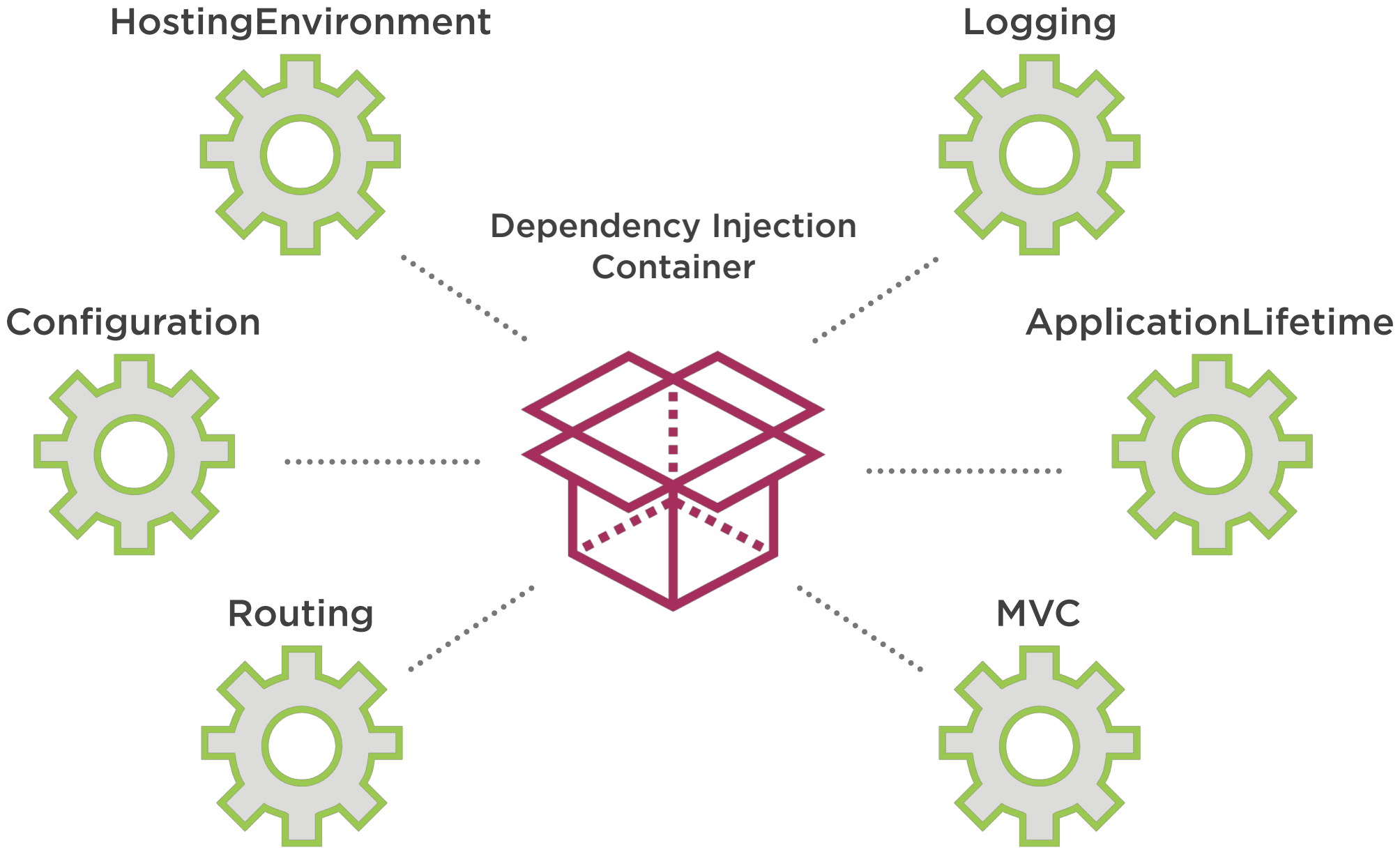
@stevejgordon www.stevejgordon.co.uk

Overview

How ASP.NET Core uses the container

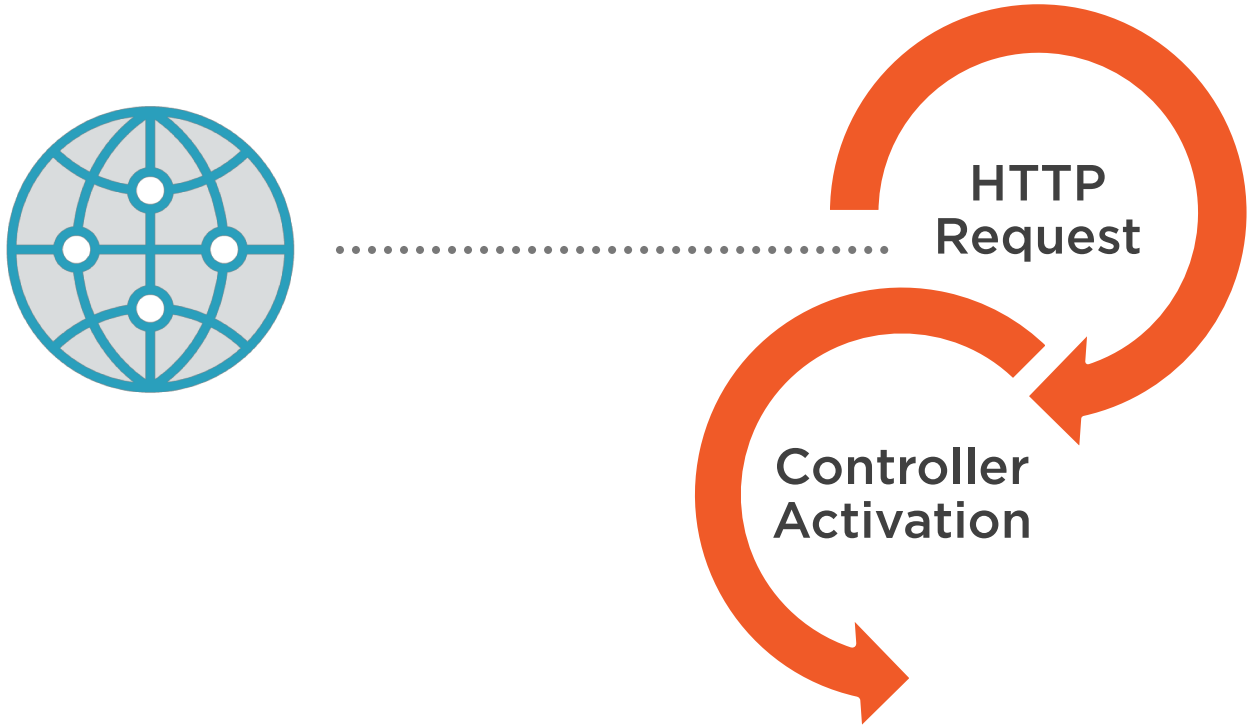
What to register with the D.I. container

Service lifetimes

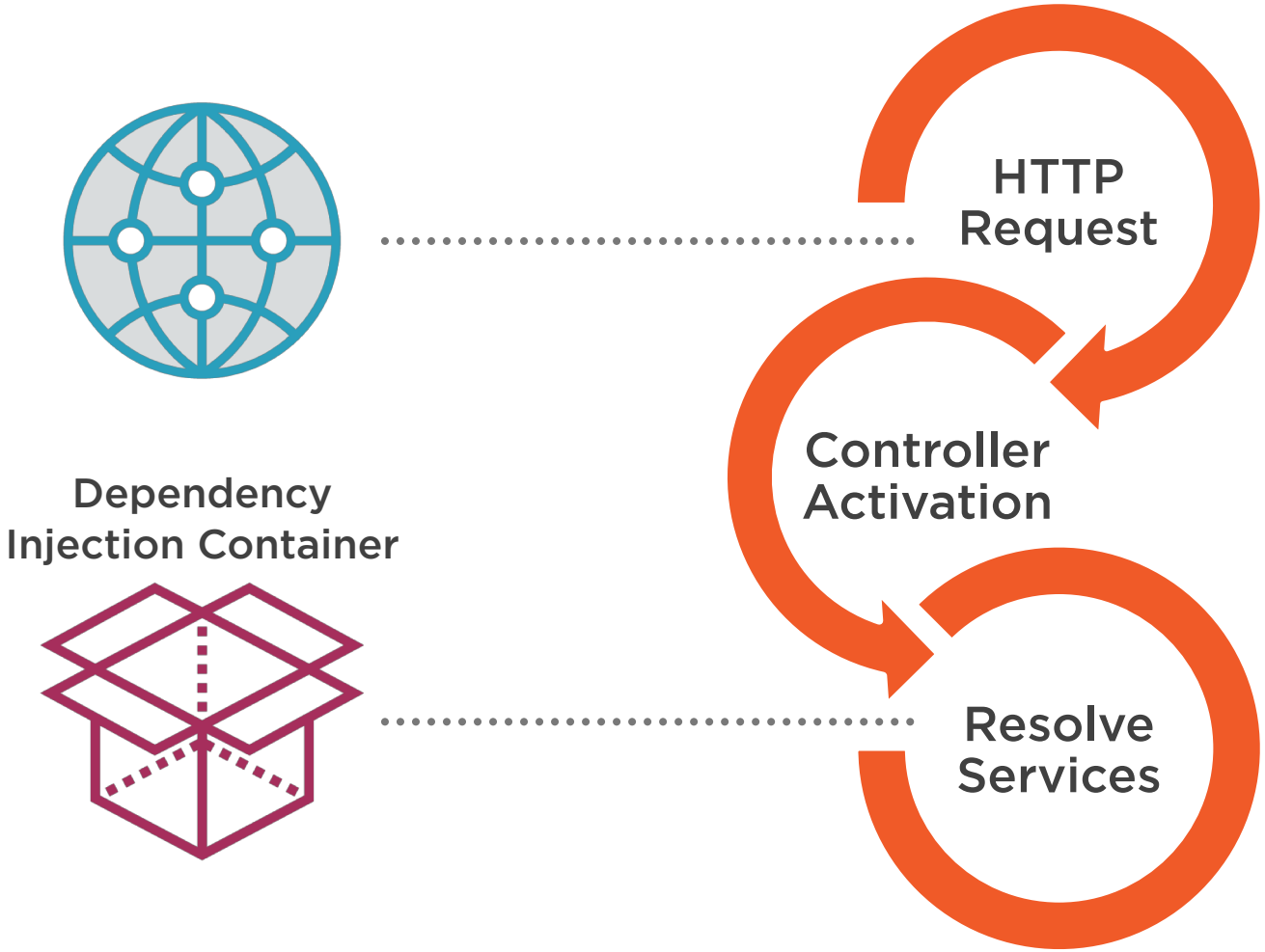

HostingEnvironment
Logging
Dependency Injection
Container
Configuration
ApplicationLifetime
Routing
MVC

ASP.NET Core and Dependency Injection

ASP.NET Core and Dependency Injection



ASP.NET Core and Dependency Injection



Microsoft.Extensions.DependencyInjection

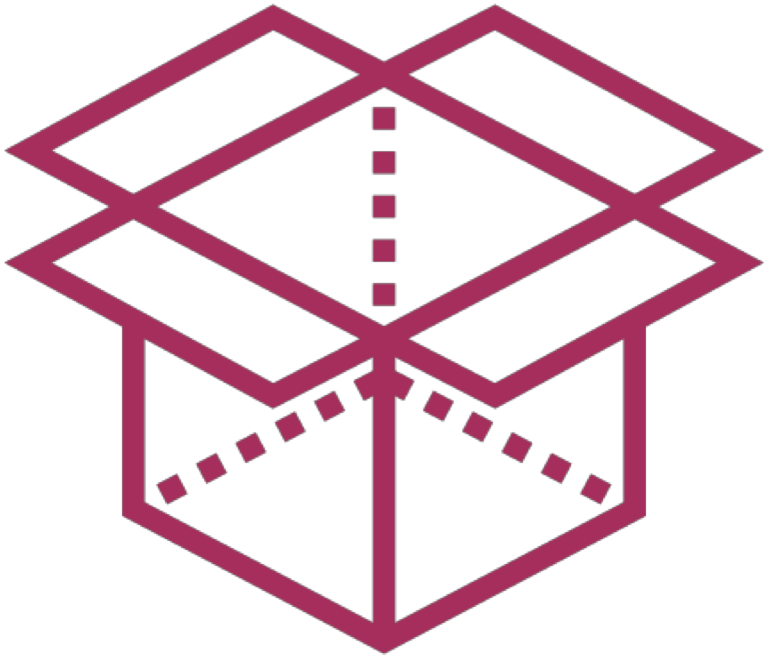
Dependency Injection Container

=

Inversion of Control Container

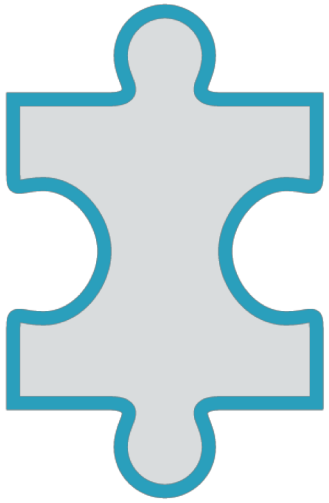
Dependency Injection Containers

Register



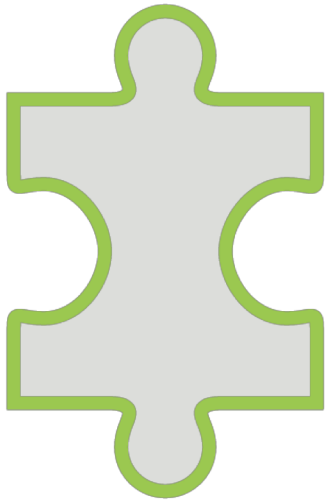
Resolve

Components



IServiceCollection

Register services



IServiceProvider

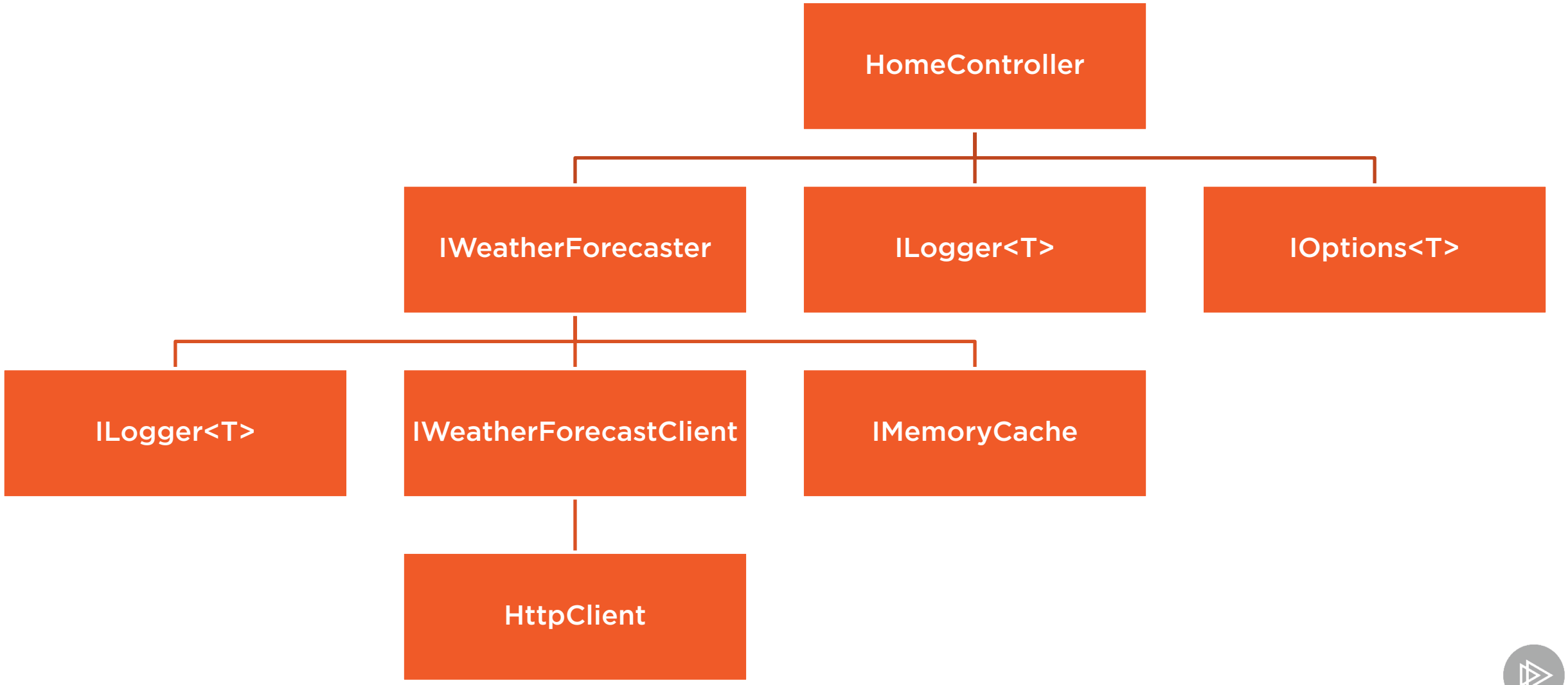
Resolve service instances

What to Register with the D.I. Container

Identifying Dependencies

- Locate ‘new’ keyword usage
- Is the object a dependency?
- Apply dependency inversion
- Register the service
- Rinse and repeat

Dependency Graph



Plain Old CLR Objects

Primitive Types and Strings

Should not be stored in a D.I. container

Typically required for configuration

Prefer the strongly-typed options pattern

Demo

Configuration and options injection

- ASP.NET Core configuration
- Binding and injecting IOptions<T>

ASP.NET Core Fundamentals

Scott Allen

https://app.pluralsight.com/library/courses/aspdotnet-core-fundamentals

Service Lifetimes

Registering Lifetimes on the IServiceCollection

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddTransient<IServiceA, ServiceA>();
    services.AddSingleton<IServiceB, ServiceB>();
    services.AddScoped<IServiceC, ServiceC>();
}
```

Service Lifetimes

Transient

Created each time they are requested

Singleton

Created once for the lifetime of the application

Scoped

Created once per request

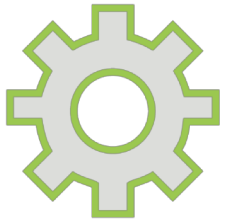
The chosen lifetime affects the creation and reuse of service instances.

Transient Services

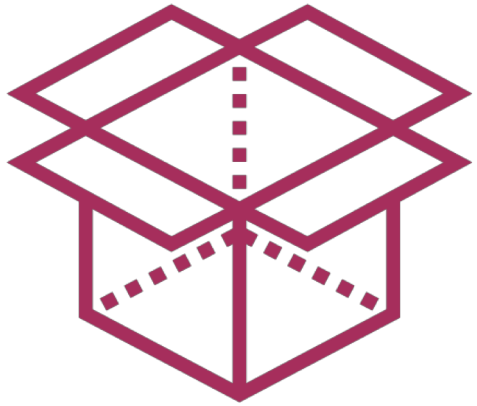
A new instance every time the service is resolved.

Resolving Transient Services

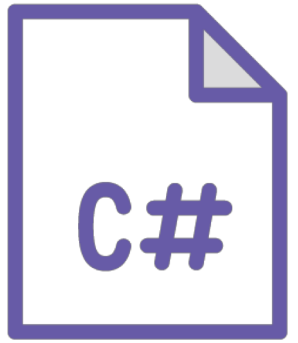
IServiceA



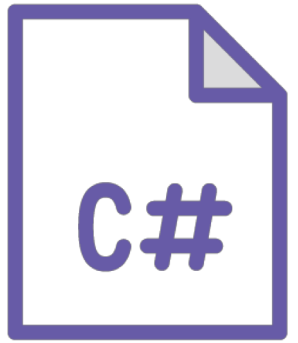
D.I. container



new ServiceA()



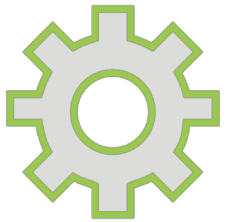
ClassOne



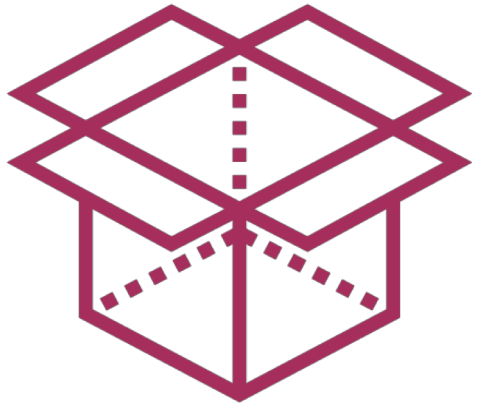
ClassTwo

Resolving Transient Services

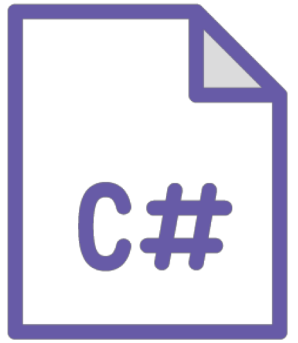
IServiceA



D.I. container

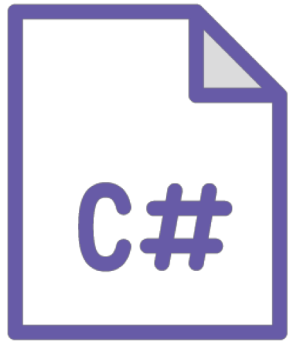


new ServiceA()



ClassOne

new ServiceA()



ClassTwo

Transient Services

Not required to be thread-safe

Potentially less efficient

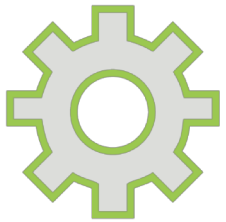
Easiest to reason about

Singleton Services

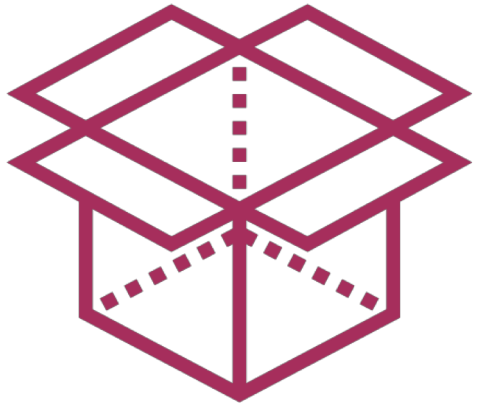
One shared instance for the lifetime of the application.

Resolving Singleton Services

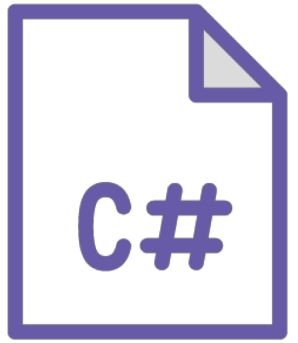
IServiceA



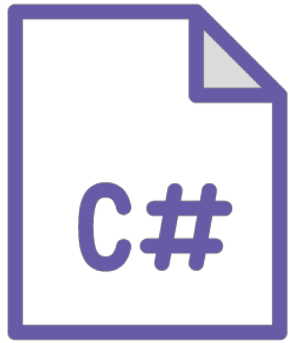
D.I. container



var instance1 = new ServiceA()



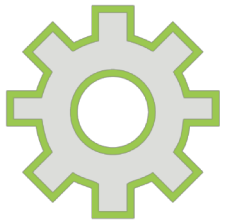
ClassOne



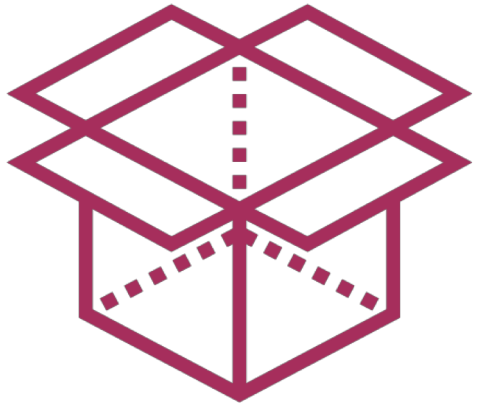
ClassTwo

Resolving Singleton Services

IServiceA

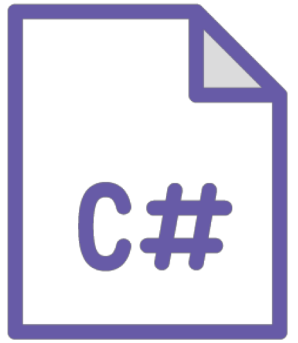


D.I. container



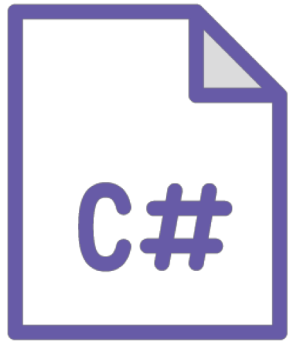
var instance1 = new ServiceA()

instance1



ClassOne

instance1



ClassTwo

Singleton Services

Generally more performant

- Allocates less objects
- Reduces load on GC

Must be thread-safe

Suited to functional stateless services

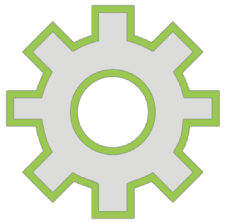
Consider frequency of use vs. memory consumption

Scoped Services

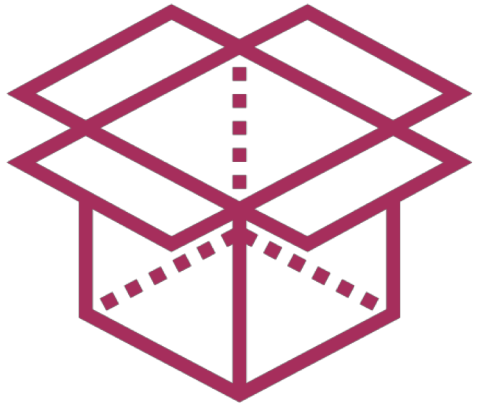
An instance per scope (request).

Resolving Scoped Services

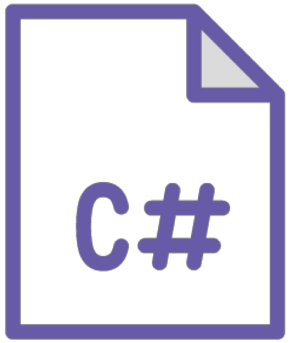
IServiceA



D.I. container

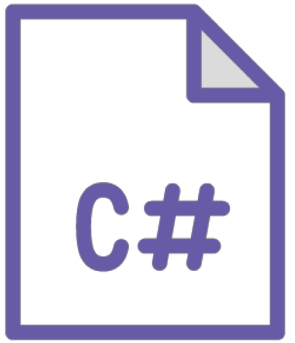


var instance1 = new ServiceA()



ClassOne

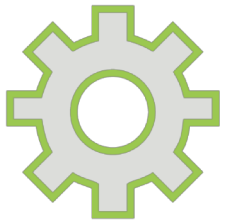
Request #1



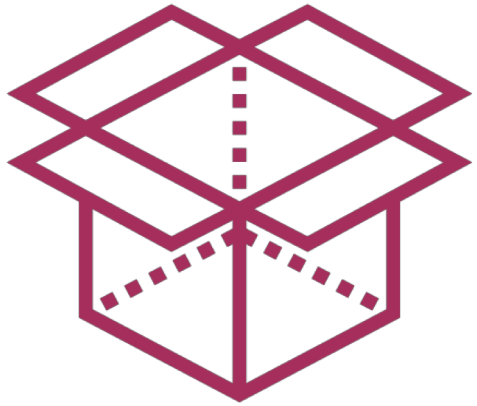
ClassTwo

Resolving Scoped Services

IServiceA

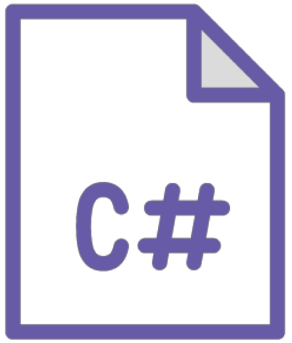


D.I. container



var instance1 = new ServiceA()

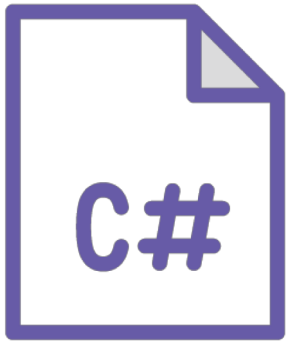
instance1



ClassOne

Request #1

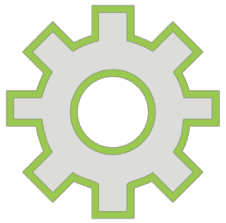
instance1



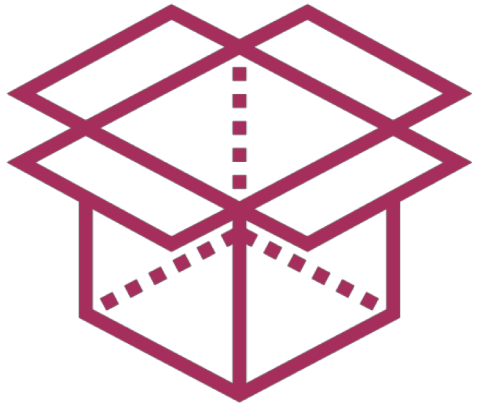
ClassTwo

Resolving Scoped Services

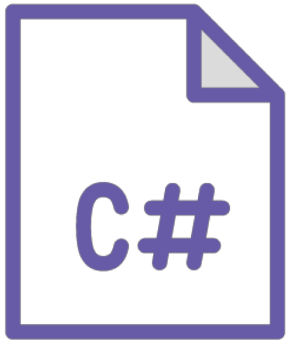
IServiceA



D.I. container

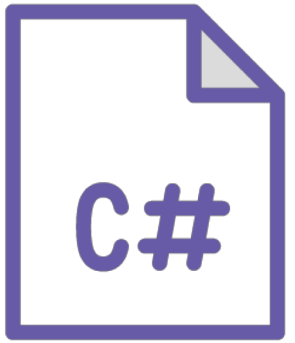


var instance2 = new ServiceA()



ClassOne

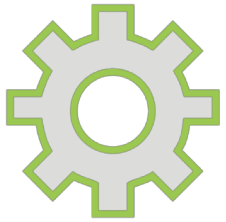
Request #2



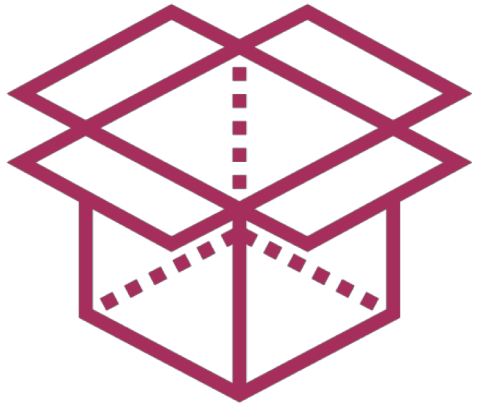
ClassTwo

Resolving Scoped Services

IServiceA

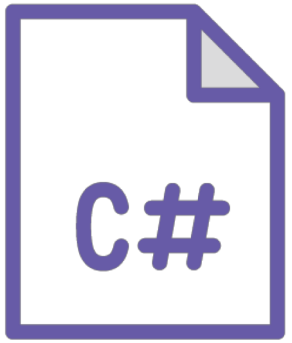


D.I. container



var instance2 = new ServiceA()

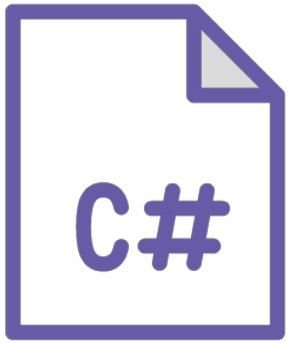
instance2



ClassOne

Request #2

instance2



ClassTwo

Scoped Services

New instance per scope (request)

Not required to be thread-safe

Useful if a service may be required by multiple consumers per request

Should not be captured into singleton services

Avoiding Captive Dependencies

A service should not depend on a service with a lifetime shorter than its own.

Side Effects of Captured Dependencies

Accidental sharing of non-thread-safe services between threads

Objects living longer than their expected lifetime

Safe Dependencies

	Transient	Scoped	Singleton
Transient	✓	✓	✓
Scoped	✗	✓	✓
Singleton	✗	✗	✓

Scope Validation

Introduced in ASP.NET Core 2.0

Enabled by default in development

Validates container scopes at startup

Review

The Microsoft dependency injection container

Types which should be registered

Inject configuration with IOptions<T>

Service lifetimes