Lifetime Management

Henry Been
Independent DevOps & Azure Architect

@henry_been www.henrybeen.nl

Overview

Explore different lifetimes

Dependency captivity

Demo

Reusing instances using the singleton lifetime

About the Different Lifetimes

The lifetime determines weather the DI container will create a new instance

The Different Lifetimes

Transient

A new instance is created every time a type is requested

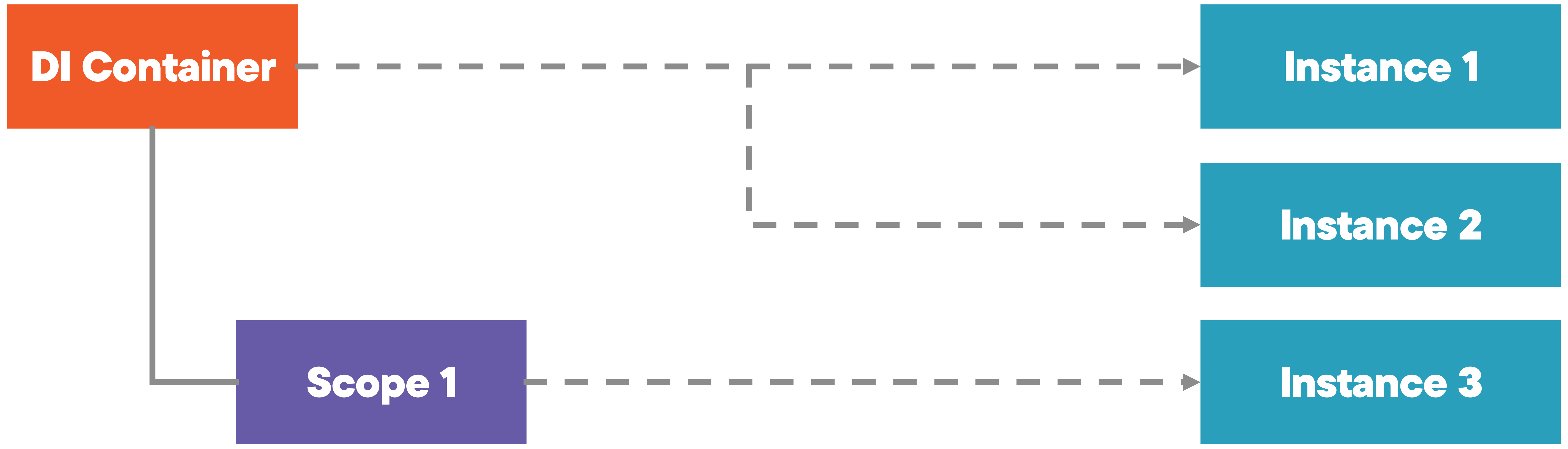
Singleton

A new instance is created once and reused from then onwards

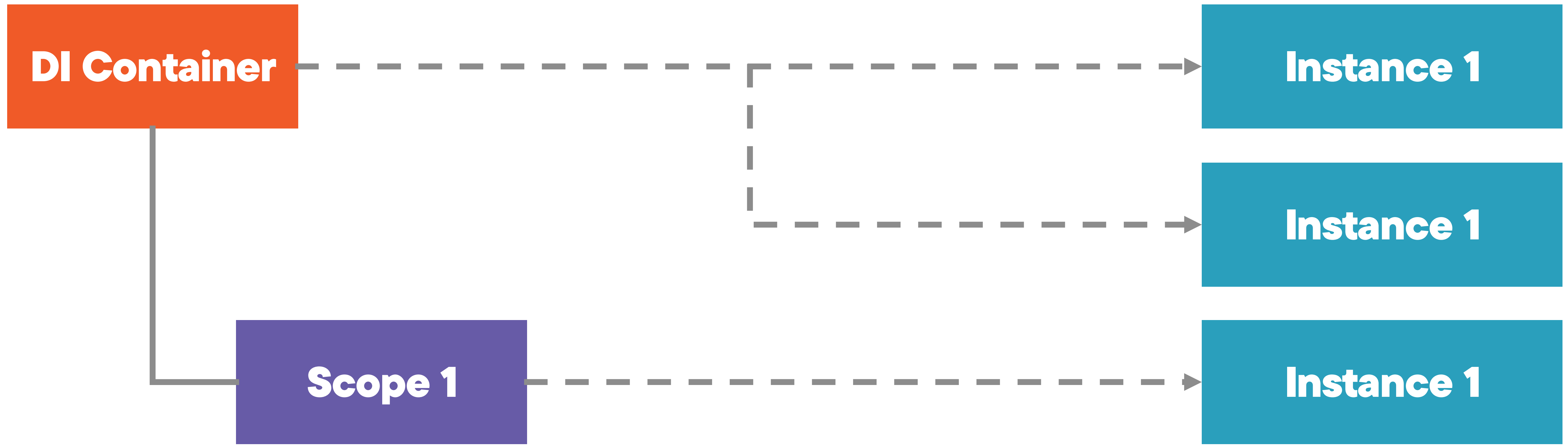
Scoped

A new instance is created once per scope, and then reused in the scope

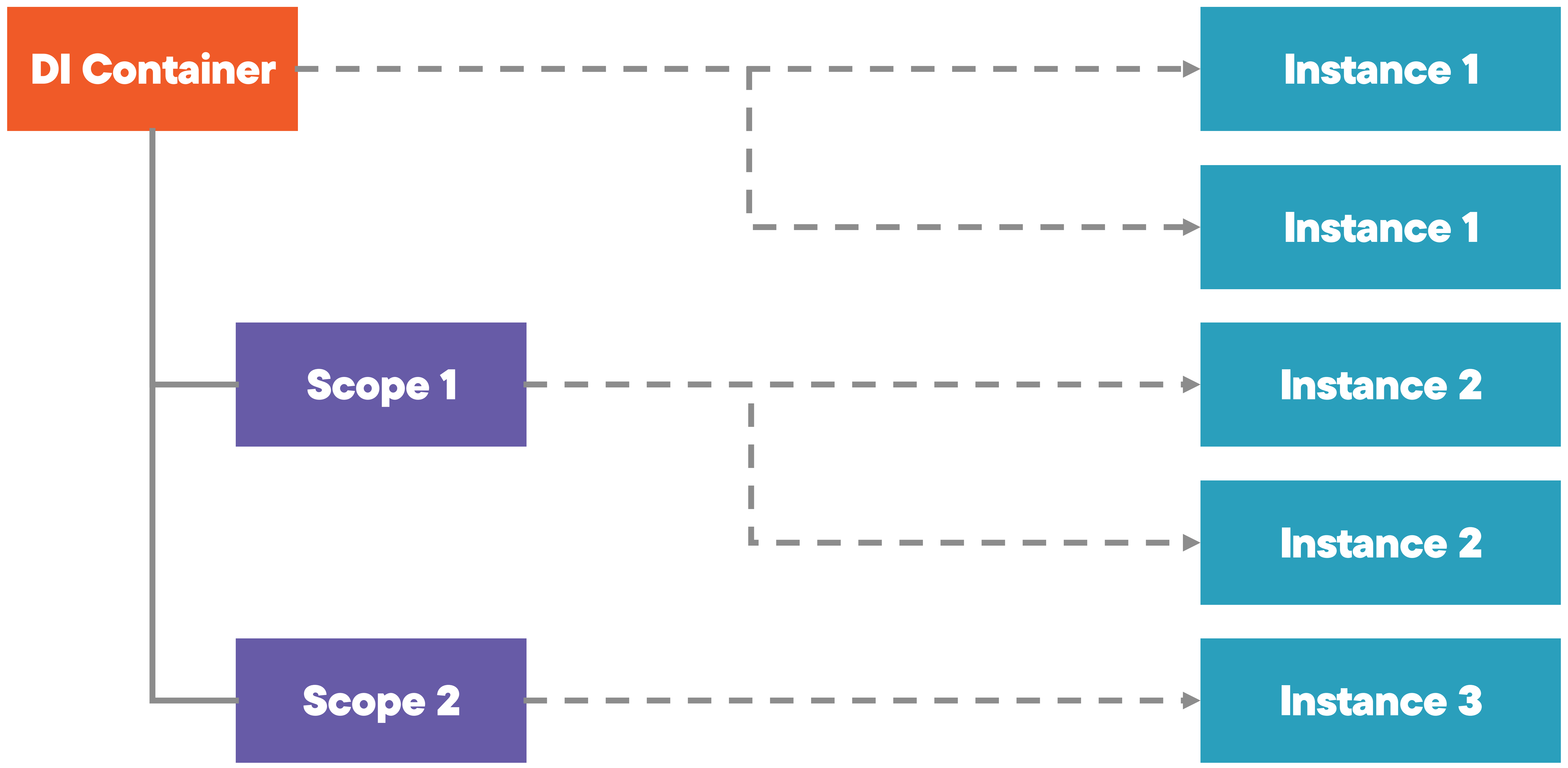
The Different Lifetimes: Transient



The Different Lifetimes: Singleton



The Different Lifetimes: Scoped



```
services.AddTransient<IProductImporter, ProductImporter>();

...


var resolvedOnce = host.Services.GetService<IProductImporter>();

var resolvedTwice = host.Services.GetService<IProductImporter>();

var areSameInstance = Object.ReferenceEquals(resolvedOnce, resolvedTwice); // f
```

Transient Lifetime

When resolving the same type multiple times, a new instance will be returned every time.

```
services.AddSingleton<IProductImporter, ProductImporter>();

...


var resolvedOnce = host.Services.GetService<IProductImporter>();

var resolvedTwice = host.Services.GetService<IProductImporter>();

var areSameInstance = Object.ReferenceEquals(resolvedOnce, resolvedTwice); // t
```

Singleton Lifetime

When resolving the same type multiple times on the same container, the same instance will be returned every time.

```
services.AddScoped<IProductImporter, ProductImporter>();

...
```

```
services.AddScoped<IProductImporter, ProductImporter>();
...


using var firstScope = host.Services.CreateScope()
var resolvedOnce = firstScope.ServiceProvider.GetRequiredService<IProductImporter>();
var resolvedTwice = firstScope.ServiceProvider.GetRequiredService<IProductImporter>();
var isSameInFirstScope = Object.ReferenceEquals(resolvedOnce, resolvedTwice); // true
```

```
services.AddScoped<IProductImporter, ProductImporter>();
...



using var firstScope = host.Services.CreateScope()
var resolvedOnce = firstScope.ServiceProvider.GetRequiredService<IProductImporter>();
var resolvedTwice = firstScope.ServiceProvider.GetRequiredService<IProductImporter>();
var isSameInFirstScope = Object.ReferenceEquals(resolvedOnce, resolvedTwice); // true

using var secondScope = host.Services.CreateScope()
var resolvedThrice = secondScope.ServiceProvider.GetRequiredService<IProductImporter>();
var resolvedFourth = secondScope.ServiceProvider.GetRequiredService<IProductImporter>();
var isSameCrossScope = Object.ReferenceEquals(resolvedOnce, resolvedFourth); // false
var isSameInSecondScope = Object.ReferenceEquals(resolvedThrice, resolvedFourthTime); //t
```

Demo

Transforming products while importing them

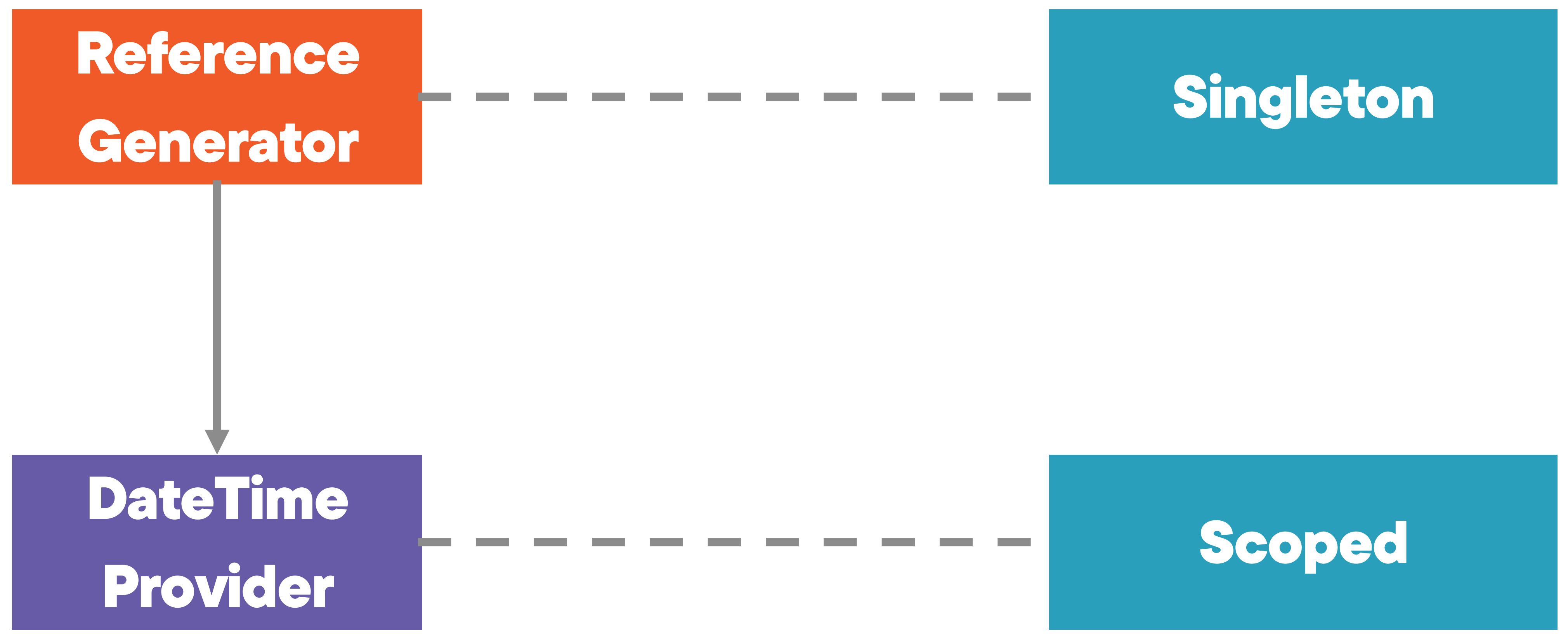
- Using the scoped lifetime
- Creating scopes

Demo

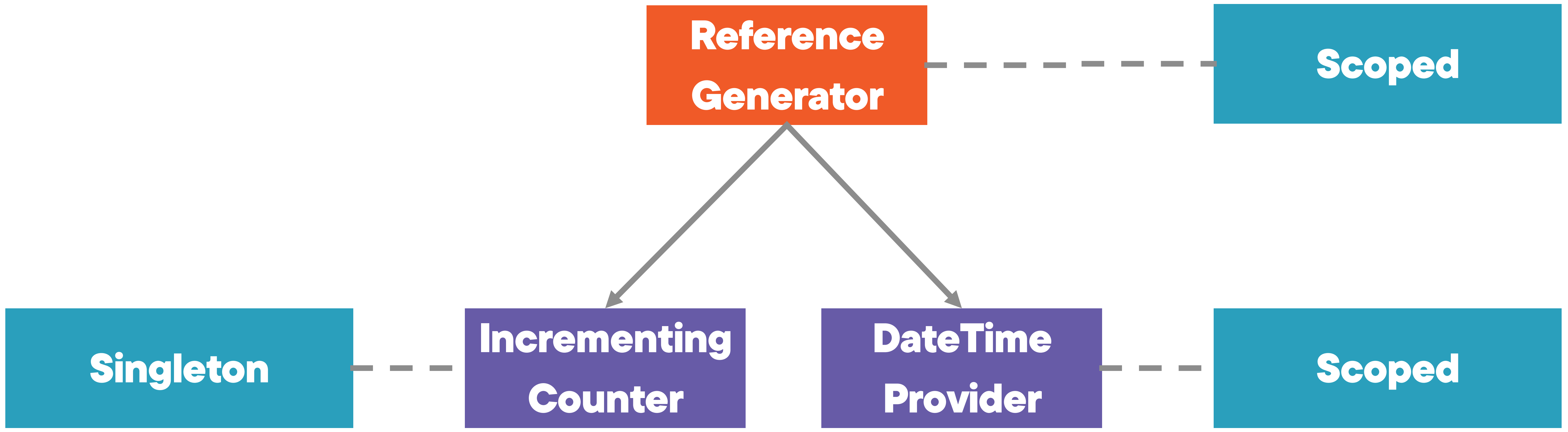
Dependency captivity

Avoiding dependency captivity

Dependency Captivity



Dependency Captivity - Fixed



Choosing the Correct Lifetime

How an *implementing type* handles state is the best indicator for choosing a **lifetime.**

Some Pointers

If there is no state at all, choose transient

If the state is derived or can be calculated on the fly, choose transient

If the state relates to a single item, request or context, and needs to be shared between depending classes, choose scoped

If the state relates to everything in the application, choose singleton

Shy away from (too many) custom scopes; leave that to your framework

Lifetime-related bugs

Choosing an incorrect lifetime can lead to unexpected behaviors, also known as bugs.

When in doubt, err on the side of transient.

Check Your Framework

EF Core DbContext

The lifetime should correspond to the unit of work (transaction). In practice: scoped

CosmosClient

Is thread-safe and intended for re-use. A single instance is recommended, hence: singleton

Demo

Selecting the correct lifetime

- For classes in the Product Importer
- Adding a connection to a SQL DB

More Information

Entity Framework Core 6 Fundamentals

Julie Lerman

Summary

Different lifetimes

- Transient
- Scoped
- Singleton

Dependency captivity and avoiding it

Design recommendations

Up Next:
Expanding the Product Importer