Dependency Injection Containers

Scobie Smith, December 2021

This is information relating to choosing an optimal DI container, based on features, performance, etc.

Current container frameworks for .NET...

There are many DI container libraries available for .NET. Some are more legacy now, with more features (though this is not necessarily good/safe), and some are more light-weight, simple, or performant. Most now use type-safe, fluent notation, rather than, say, string names. Most are free. Here is a list of most, which are also still active.

Newer, lighter, growing:

Simple Injector: https://simpleinjector.org/ (New. Lightweight, fast. Concurrency. ~22M downloads.)

LightInject: https://www.lightinject.net/ (Relatively new. Lightweight, fast. ~4+M downloads.)

Autofac: https://autofac.org/ (Popular. Not fast, plenty of features. ~115M downloads.)

Grace: https://github.com/ipjohnson/Grace (New. Fast). Some features worth investigating. Small user base, ~1M downloads. But API is unintuitive and difficult.

Older, lots of features, but not lightweight or fast, yet still popular:

Unity: http://unitycontainer.org/ (Fatter. Slow. Created by MS. #2 in downloads. ~45M downloads.)

Ninject: http://www.ninject.org/ (Mature, 2008. Slow. Ease of use, popular. ~23M downloads.)

Castle Windsor: https://www.castleproject.org/projects/windsor/ (Mature. Feature rich. Slow. ~21M downloads.)

Lamar: https://jasperfx.github.io/lamar/ (New. Slow. Aimed as successor to StructureMap. ~1.7M downloads.)

Old:

Spring.NET: http://springframework.net/

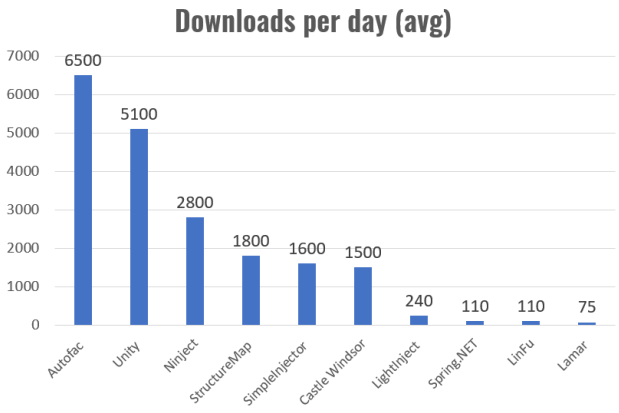
StructureMap: http://structuremap.github.io/ (Old, 2004. Ended in 2018. Was popular. Slow.)

Reviews and performance

There are several sites with lists or reviews of some or many DI frameworks for .NET. For example:

https://www.claudiobernasconi.ch/2019/01/24/the-ultimate-list-of-net-dependency-injection-frameworks/

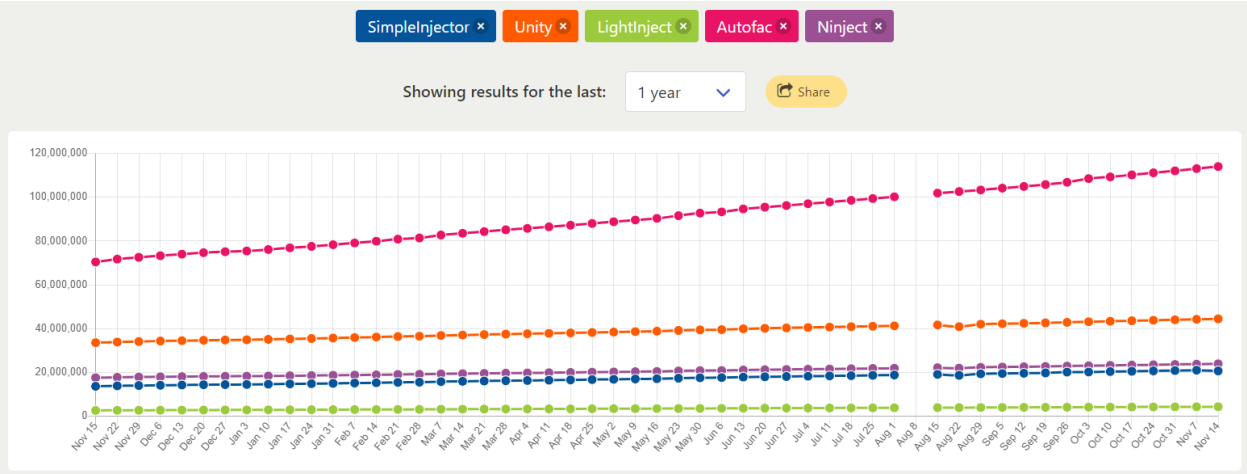
Nuget 1/24/2019 (from this article):

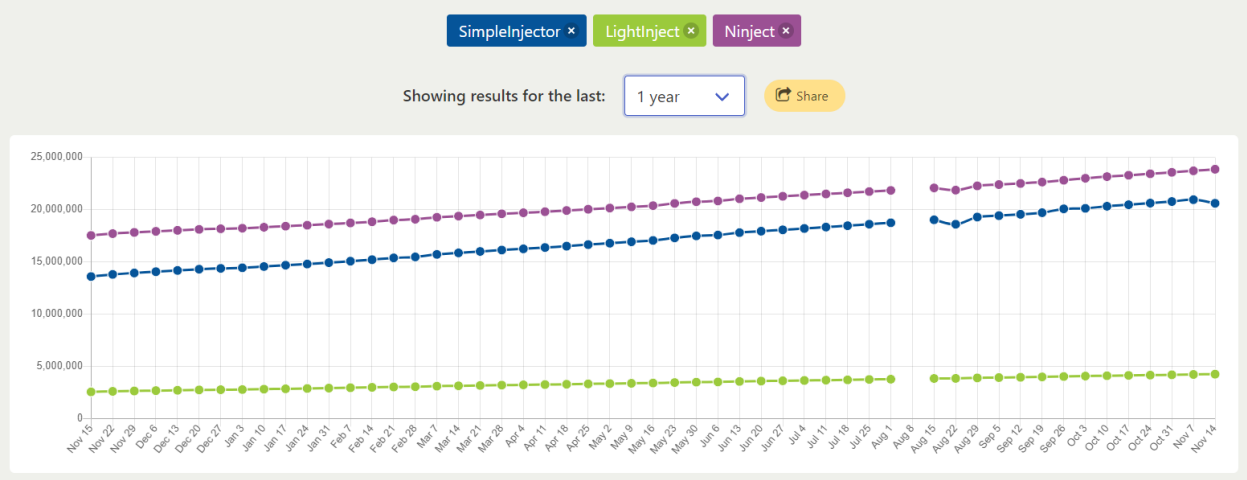


My current check...

https://www.nugettrends.com/

11/17/2021:





Another details benchmark listing (useful, but no date given, except versions):

https://danielpalme.github.io/IocPerformance/

What is driving popularity?? Probably no single factor. Features, ease of use, decoupling, support, documentation, performance....

It seems Autofac is the most popular, at least among newer frameworks.

Unity still seems strong, maybe from MS origins, integration as part of larger framework.

Ninject continues to be popular, but maybe no longer dominating, after its age. Not being updated much.

Simple Injector, though new and smaller in market share, is rising: 22m downloads so far is not bad. It promotes SOLID principles. It also supports concurrency.

LightInject—it seems a little surprising that it has not been growing noticeably in popularity. Similar to Simple Injector and in some ways easier.

Grace is up and coming, but still small. But I ended up finding the API difficult, and there is a lack of documentation and virtually no community to ask.

Examples of feature issues

Some features areas to consider

What kinds of injection: constructor, property, method. Emphasis is always on constructor injection.

What kinds of types can be registered: interfaces, classes, etc.

Object lifetime management (lifestyles): singleton, transient, etc., also scoping.

Concurrency support.

Fluent syntax, type-safe (not magic strings).

Basic design decisions about choice of framework

A choice is to be made between having a framework that offers more automatic and implicit functionality ("magic"), but which might then undermine careful SOLID principles or decoupling, or even allow run-time errors to creep in. This is in contrast to a framework that is more strict with the pattern, enforcing decoupling, not letting developers veer off into writing poor quality code.

Property injection (implicit, explicit)

Some containers provide for implicit property injection, but this can lead to run-time errors. Another approach is to be explicit, but require an attribute that references the container—but this puts a dependency on the container library among the classes, which is undesirable.

Simple Injector requires defining this explicitly in code, which then is safer. One way is an initializer lambda given at type-registration-time, which sets the property in question.

https://docs.simpleinjector.org/en/latest/advanced.html#property-injection

An example problem: what if the property is read-only? Then the container cannot give it a value when the type is resolved. Maybe then it will be passed over, but this opens the door to an unexpected undefined value at run-time.

Generally, constructor injection is preferable. If property injection is required, it has to be explicitly written. So, it seems acceptable if a DI container does *not* provide implicit property injection as a feature.

Another approach is to provide empty constructor arguments when the dependency is optional and not needed.

Some frameworks, like LightInject, let you turn off property injection—but have the option to turn it on if you really need it and accept the risks.

Multiple bindings

Sometimes there are multiple possible implementation to register for a given interface. So, you can get all the implementations through requesting the IEnumerable<IMyInterface>. However, it can then easily happen that you want just one of these implementations in some contexts. So, how do you specify that you want to resolve for only the one type, instead of an IEnumerable? Each framework handles this differently.

Many dependencies and constructor injection...

What if there are a lot of dependencies, say 100? It is not workable to provide them all as constructor parameters. First, this may be a clear indication of violation of single responsibility. But it might also be solved by injecting a factory, for example, if the dependencies are all related in some way. The factory would take some other parameter values and be used in the object in question to create the specific dependency.

Decorator pattern (and aspect-oriented programming)

It is possible to use dependency injection to set up a decorator pattern, which can then facilitate add new functionality or cross-cutting features to existing classes, without modifying the original classes. This support O (open-close principle) of SOLID. Simple Injector discusses this at some length in their documentation.

More issues to compare...

Concurrency support.

Scoping, such lifetime limited to a thread.

Decorator pattern and adding aspects (or cross-cutting features).

Evaluative Thoughts

I went into this from an earlier background of experimenting (years ago) with Ninject, autofac, and LightInject. At that time, I came to prefer LightInject a lot and standardized my own coding on that, though I did like Autofac then too. If LightInject faster and simpler. So, I was figuring I would still prefer LightInject. But since then, Simple Injector has risen in the ranks. Autofac also increased a lot in popularity, #2 to Unity.

Simple Injector has an easy-to-use API, is indeed simple, and is safe (regarding types, run-time bugs, etc.). I found the API one of the most intuitive, quicker to use directly without a lot of explanation. But there are some situations where a little knowledge is required to know how to do it, such as factory delegates, passing in a constructor parameter, etc.

The designers of Simple Injector also seem to put the most thought of all the frameworks into making their approach safer, steering developing in line with SOLID principles. For example, they do not allow property injection be default; it requires explicit implementation to use. They explain in some details the dangers (e.g., run-time bugs) of property injection. So, this is also even safer than LightInject, which does provide a way for implicit property injection, as well as explicit—though they have a way to disable property injection.

SimpleInjector has a mechanism to support the **decorator pattern**, which makes it easy to use an aspect-oriented approach (open-close principle of SOLID). (Cf. LightInject also below.)

Unity has an API that is similar to Simple Injector (or we should say vice versa). That is, the names of methods, the way the fluent syntax is set up—it seems more straightforward to me. But the API is overall more complex. It sometimes requires using a string name for a parameter (override), etc., which breaks type safety, requires keeping track of the name actually used in the code, etc.—which I find unacceptable now.

There was also an important place where it was necessary to construct manually the dependency to be injected, because the default type in a collection of implementations (of ILogger) had to be specified by string name—and this could be passed into the Transaction type. So, there was no way to autowire this

dependency on the named (my default) implementation of the interface. This problem follows from the design flaw in which a collection of implementations (of ILogger) treats IEnumerable<T> differently from T[]. So, it is apparently impossible to define some implementation as the default in a way that will resolve automatically when only one instance (TextLogger) is desired. There seems no way (as in LightInject) to sort the list or leave one unnamed as the default, allowing some way to autowire.

Autofac, on the other hand, has a noticeably less intuitive syntax, though it use a fluent style. You *register* a target type *as* the interface or type that is the "source", as it were... which I normally thing of the thing being registered to the target. So, it feels backwards. There is an IComponentContext that can come into the picture, which also can have some gotchas (I hit one even in this small sample). And there seems to be plenty more opportunity for run-time errors. But I did not need to use some string name to refer to a parameter or object, so it avoids that, which came up in Unity.

LightInject has an API fairly similar to Simple Injector, even in some distinctive names (e.g., GetInstance() instead of Resolve()). In fact, quite a few registration lines carry over as-is from the Simple Injector example. It is quick and intuitive to start using and follows the fluent syntactic approach. The documentation is one of the better ones, so that you don't have to go hunting for answers as much. It has a better approach to collections of types registered to the same interface. For example, it does not treat IEnumerable<T> different from T[], as Unity—so this is definitely better. It also seems to offer more capability in this area, e.g., ordering the dependencies listed in the collection using a string name.

LightInject allows for both implicit and explicit property injection, they also have an option to disable all property injection, which they recommend. So, this seems to be basically as safety-minded as Simple Injector, but does also give you an easy way to do property injection in your project if you really do want it.

In addition, implementing my project with LightInject was one of the quickest, as far as time writing the code—though LightInject was the fifth framework I tested. It just did not require a lot of hunting for answers. I got everything from their documentation. I still don't see why LightInject is not more popular in comparison with Simple Injector. There seem to be some other nice features, like a fallback mechanism for unresolved types (the RegisterFallback() method). LightInject also has excellent performance times, about equal with Simple Injector.

LightInject, like SimpleInjector, also has a mechanism to support the **decorator pattern**, so it is also possible to use aspect-oriented programming here too. It is worth digging into the differences with SimpleInjector to see which framework is more robust in this area.

Ninject has its own syntax, which is ok. It is not know for speed. It makes use of some magic strings. But it seems feature-sufficient and was a standard for a long time, though now it seems like it has not been updated for a while. The newer options are more type-safe and cleaner.

Grace was one I began to try out. However, just trying to set the lifetime style, to singleton, was strangely difficult. Documentation is inadequate—did not easily answer this basic question. At <1M downloads so far, there is too small a user base to find discussion or answers online. So, I actually gave up on this one, even though the initial descriptions sounded attractive. Also, the API is someone like Autofac, but they made it even more complicated, in that you generally have to pass in a lambda to

specify the target to be resolved—on top of the register-as approach of Autofac. So, that was also weirdly bad, contrary to their advertised simplicity.

Castle Windsor is still a big one to consider. My impression from the past and today is that it is large, not so simple, with many features, but more vulnerable to introducing bugs in the code. So, I did not experiment with it here.

Thus, of these, I would favor in order:

1. Simple Injector. Basically a tie with LightInject, but maybe somewhat more documentation, more type-safe, and larger community.

2. LightInject. This is a very close call between these top two. LightInject seems just as fast, able to be just as safe, but providing ease of use when some more flexibility is desired. Syntax is similar to Simple Injector.

3. Autofac. Syntax is less intuitive, but could be gotten used to. More complex, but maybe more of the favorite today. Seems feature-rich. But added complexity brings more risk of introducing subtle run-time bugs. Not fast.

So far I would not choose:

Unity: A few little but painful desire problems, with a real problem of using unsafe magic strings. Most frameworks have moved away from allowing this danger.

Ninject. Ok, but there are better options now.

Grace: Surprisingly less intuitive API, harder to use, with significantly less documentation and community.

Castle Windsor (but not tested here).