**Autofac**

1. **Adicionando o Autofac ao projeto:**
   1. **Adicionando o pacote Autofac:**

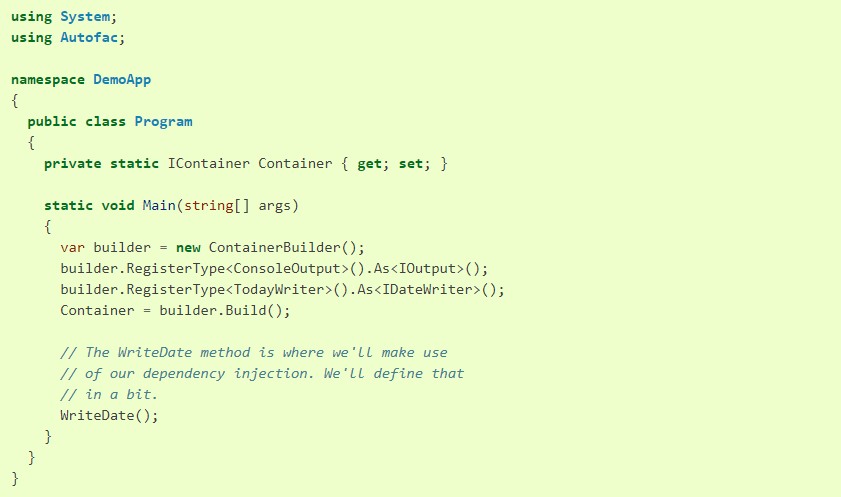
Podemos adicionar o pacote do Autofac com o gerenciador de pacotes NuGet:



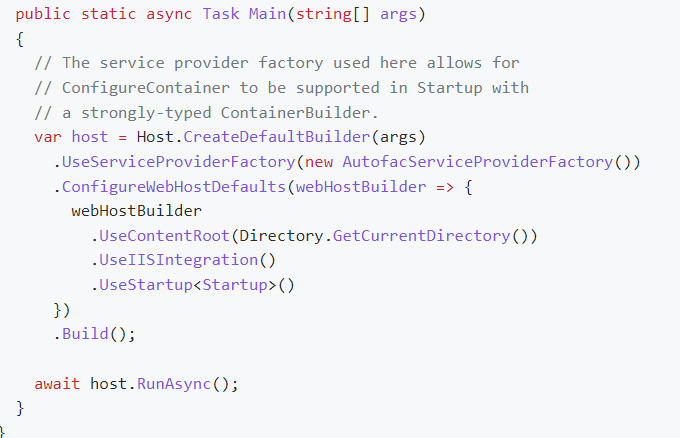
Podemos adicionar o pacote do Autofac através de comando também:

* Dotnet add package Autofac
  1. **Adicionando container do Autofac no builder:**

Para usarmos o Autofac devemos adicionar o container do Autofac no nosso builder, isso pode ser feito da seguinte forma:



Ou podemos adicionar assim:

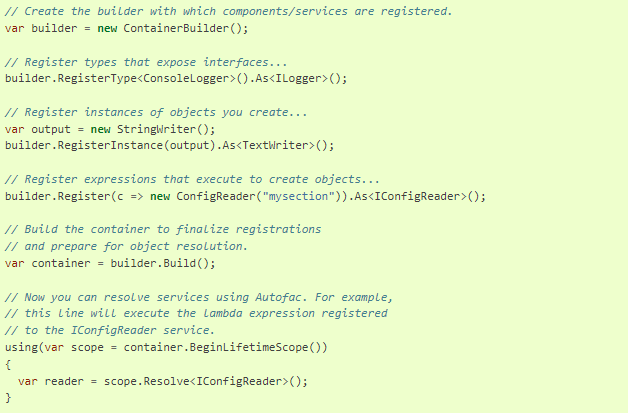


Porem dessa forma devemos ter o seguinte package: Autofac.Extensions.DependencyInjection

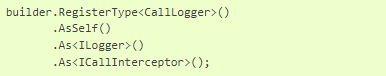
1. **Conceitos de registro:**

Você registra os componentes com o Autofac criando um **ContainerBuilder** e informando ao construtor quais **componentes** expôem quais **serviços**.

Cada componente expõe um ou mais **serviços** conectados usando os **As()** métodos em **ContainerBuilder**.



Se você deseja expor um componente como um conjunto de serviços, bem como usar o serviço padrão, use o AsSelfmétodo:



É importante destacar que o autofac possui métodos para registro de diferentes tipos de ações como métodos, classes, construtores, interfaces etc. Você pode encontrar mais detalhes sobre eles na documentação: <https://autofac.readthedocs.io/en/latest/register/index.html>

1. **Tempos de vida de componentes:**

Diferente do container padrão da Microsoft o Autofac não possui 3 tempos de vida e sim dezenas, são eles:

**Transient**: Esta é a vida útil padrão e cria uma nova instância do componente sempre que é resolvido.

**Singleton**: Cria uma única instância do componente que é usada sempre que o componente é resolvido.

**InstancePerLifetimeScope**: Cria uma única instância do componente por escopo de tempo de vida e reutiliza essa instância sempre que o componente é resolvido dentro desse escopo.

**InstancePerMatchingLifetimeScope**: Cria uma única instância do componente para cada escopo de tempo de vida que corresponde a um escopo específico fornecido durante a resolução.

**InstancePerDependency**: Cria uma única instância do componente para cada solicitação de resolução.

**InstancePerHttpRequest**: Cria uma única instância do componente para cada solicitação HTTP.

**InstancePerOwned**: Cria uma única instância do componente por proprietário e libera essa instância quando o proprietário é liberado.

**InstancePerLifetimeScopeNamed**: Cria uma única instância do componente para cada escopo de tempo de vida com um nome específico fornecido durante a resolução.

**InstancePerMatchingLifetimeScopeNamed**: Cria uma única instância do componente para cada escopo de tempo de vida com um nome e um escopo específico fornecido durante a resolução.

**InstancePerMatchingLifetimeScopeTagged**: Cria uma única instância do componente para cada escopo de tempo de vida que corresponde a uma tag específica e um escopo específico fornecidos durante a resolução.

**InstancePerMatchingLifetimeScopeTyped**: Cria uma única instância do componente para cada escopo de tempo de vida que corresponde a um tipo específico e um escopo específico fornecidos durante a resolução.

**InstancePerOwned**<T>: Cria uma única instância do componente por proprietário e libera essa instância quando o proprietário é liberado. A diferença em relação ao **InstancePerOwned** é que este tempo de vida especifica o tipo do proprietário.

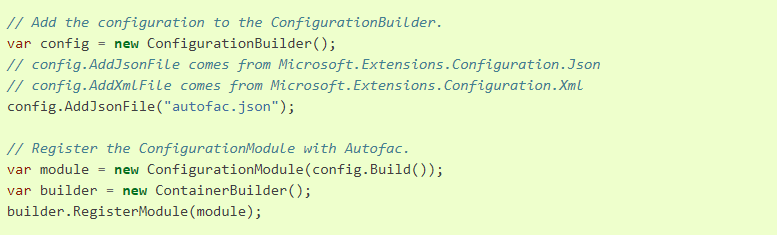
**InstancePerMatchingLifetimeScope**<T>: Cria uma única instância do componente para cada escopo de tempo de vida que corresponde a um tipo específico fornecido durante a resolução.

**InstancePerMatchingLifetimeScope<TLimit, TActivatorData, TRegistrationStyle>:** Cria uma única instância do componente para cada escopo de tempo de vida que corresponde a um tipo específico, informações de ativação e estilo de registro específicos fornecidos durante a resolução.

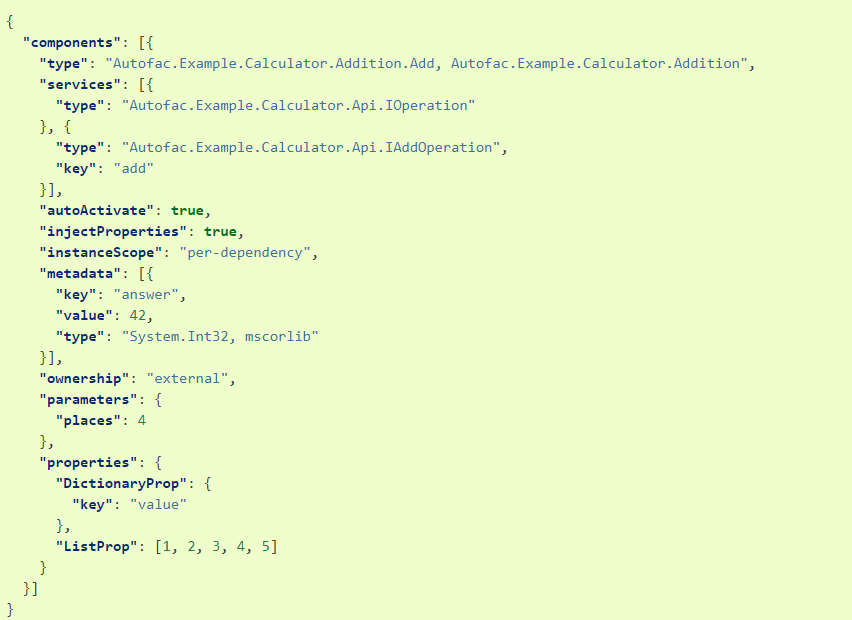
O tempo de vida padrão é **Transient**., que significa que uma nova instância do componente é criada cada vez que ele é resolvido pelo contêiner Autofac.

1. **Configuração via JSON/XML:**

É importante entender que o Autofac pode ser configurado via arquivo e método ConfigurationBuilder():



O arquivo JSON de configuração poder ser algo parecido com isso:



É importante entender que o Autofac possui diversas propriedades de configuração e é importante sempre olhar a documentação onde contêm todas essas configurações:

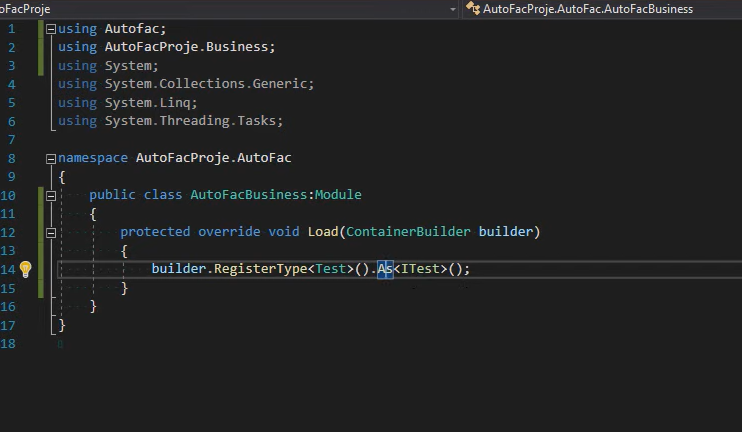
[https://autofac.readthedocs.io/en/latest/configuration/xml.html#](https://autofac.readthedocs.io/en/latest/configuration/xml.html)

1. **Módulos:**

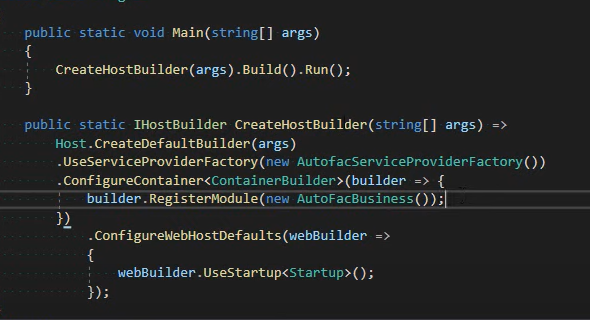
O Autofac traz o conceito de módulos que basicamente são uma forma de você organizar a injeção de dependências no container através da localização classes ou arquivos, isso possibilitada separar a logica de injeção de dependências ao longo do código e não apenas no método main().

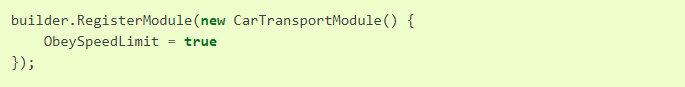
* 1. **Módulos via classe:**

Os módulos via classe são os mais comuns e consistem em separar a logica de injeção de dependências em classes:



É importante entender que devemos declarar os módulos no builder:





* 1. **Via configuração:**

Nos podemos declarar os módulos via arquivo de configuração também:

