

# Serilog MSSQL Sink

May 21, 2024

ⓘ This conversation may reflect the link creator's personalized data, which isn't shared and can meaningfully change how the model responds.

👤 **Anonymous**

```
This is my code "using Autofac.Extensions.DependencyInjection;
using Autofac;
using DependencyInjectionPractice1.Data;
using DependencyInjectionPractice1.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.EntityFrameworkCore;
using DependencyInjectionPractice1;
using Serilog;
using Serilog.Events;

#region Bootstrap Logger Configuration
var configuration = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("appsettings.json")
    .Build();

Log.Logger = new LoggerConfiguration()
    .ReadFrom.Configuration(configuration)
    .CreateBootstrapLogger();
#endregion

try
{
    Log.Information("Application Starting...");

    var builder = WebApplication.CreateBuilder(args);

    #region Serilog General Configuration
builder.Host.UseSerilog((ctx, lc) => lc
    .MinimumLevel.Debug()
    .MinimumLevel.Override("Microsoft", LogEventLevel.Warning)
    .Enrich.FromLogContext()
    .ReadFrom.Configuration(builder.Configuration));
    #endregion

    // Add services to the container.
    var connectionString =
builder.Configuration.GetConnectionString("DefaultConnection") ??
throw new InvalidOperationException("Connection string
'DefaultConnection' not found.");
    builder.Services.AddDbContext<ApplicationContext>(options =>
        options.UseSqlServer(connectionString));
    builder.Services.AddDatabaseDeveloperPageExceptionFilter();

    #region Autofac Configuration
builder.Host.UseServiceProviderFactory(new
AutofacServiceProviderFactory());
    builder.Host.ConfigureContainer<ContainerBuilder>(containerBuilder
=>
{
    containerBuilder.RegisterModule(new WebModule());
});
    #endregion

    builder.Services.AddDefaultIdentity<IdentityUser>(options =>
        options.SignIn.RequireConfirmedAccount = true)
        .AddEntityFrameworkStores<ApplicationContext>();
    builder.Services.AddControllersWithViews();

    //Service Collection
    //builder.Services.AddScoped<IEmailSender, EmailSender>();
    //builder.Services.AddKeyedScoped<IEmailSender,
```

```

        HtmlEmailSender>("another");
        builder.Services.AddScoped<IMembership, Membership>();
        builder.Services.AddScoped<IReceiveEmail, ReceiveEmail>();

        var app = builder.Build();

        // Configure the HTTP request pipeline.
        if (app.Environment.IsDevelopment())
        {
            app.UseMigrationsEndPoint();
        }
        else
        {
            app.UseExceptionHandler("/Home/Error");
            // The default HSTS value is 30 days. You may want to change this for
            production scenarios, see https://aka.ms/aspnetcore-hsts.
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseStaticFiles();

        app.UseRouting();

        app.UseAuthorization();

        app.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
        app.MapRazorPages();

        app.Run();
    }

    catch(Exception ex)
    {
        Log.Fatal(ex, "Failed to start application.");
    }
    finally
    {
        Log.CloseAndFlush();
    }

    "
    "{
        "ConnectionStrings": {
            "DefaultConnection": "Server=\\(localdb)\\mssqllocaldb;Database=aspnet-
DependencyInjectionPractice1-9de3fafc-c6ff-47dd-9f4b-
69533ba0da83;Trusted_Connection=True;MultipleActiveResultSets=true",
            //"DefaultConnection":
            "Server=PLICL60\\SQLEXPRESS;Database=asptest1;Trusted_Connection=True;MultipleActiveResultSets=true"
        },
        "Logging": {
            "LogLevel": {
                "Default": "Information",
                "Microsoft.AspNetCore": "Warning"
            }
        },
        "Serilog": {
            "WriteTo": [
                {
                    "Name": "File",
                    "Args": {
                        "path": "Logs/web-log.log",
                        "rollingInterval": "Day"
                    }
                }
            ]
        },
        "AllowedHosts": "*"
    }
}

```

```
}
```

"

Now tell me the process to add serilog database(mssql) sink, step by step in this existing code

ChatGPT

To add the Serilog MSSQL sink to your existing code, follow these steps:

### 1. Install the necessary packages

You need to install the `Serilog.Sinks.MSSqlServer` package. Open the NuGet Package Manager Console and run:

```
sh
```

Install-Package Serilog.Sinks.MSSqlServer

### 2. Modify `appsettings.json`

Add the configuration for the MSSQL sink in your `appsettings.json` file. This should be added under the `Serilog` section:

```
json
```

{

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-DependencyIn",  
 "SerilogConnection": "Server=(localdb)\\mssqllocaldb;Database=SerilogLogs;Trusted  
 },

"Logging": {

"LogLevel": {

"Default": "Information",  
 "Microsoft.AspNetCore": "Warning"  
 }

},

"Serilog": {

"WriteTo": [

{

"Name": "File",  
 "Args": {  
 "path": "Logs/web-log-.log",  
 "rollingInterval": "Day"  
 }

},

{

"Name": "MSSqlServer",  
 "Args": {  
 "connectionString": "SerilogConnection",  
 "tableName": "Logs",  
 "autoCreateSqlTable": true,  
 "columnOptionsSection": {  
 "addStandardColumns": [ "LogEvent" ],  
 "removeStandardColumns": [ "MessageTemplate" ]  
 }  
 }

}

]

},

"AllowedHosts": "\*"

}

### 3. Modify `Program.cs` to use the MSSQL sink

Ensure that your `Program.cs` file reads the updated configuration and initializes Serilog with the MSSQL sink.

```
csharp
```

```
using Autofac.Extensions.DependencyInjection;
using Autofac;
using DependencyInjectionPractice1.Data;
using DependencyInjectionPractice1.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI.Services;
```

```

using Microsoft.EntityFrameworkCore;
using DependencyInjectionPractice1;
using Serilog;
using Serilog.Events;
using Serilog.Sinks.MSSqlServer; // Add this using directive

#region Bootstrap Logger Configuration
var configuration = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("appsettings.json")
    .Build();

Log.Logger = new LoggerConfiguration()
    .ReadFrom.Configuration(configuration)
    .CreateBootstrapLogger();
#endregion

try
{
    Log.Information("Application Starting...");

    var builder = WebApplication.CreateBuilder(args);

    #region Serilog General Configuration
    builder.Host.UseSerilog((ctx, lc) => lc
        .MinimumLevel.Debug()
        .MinimumLevel.Override("Microsoft", LogEventLevel.Warning)
        .Enrich.FromLogContext()
        .ReadFrom.Configuration(builder.Configuration));
    #endregion

    // Add services to the container.
    var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
    builder.Services.AddDbContext<ApplicationContext>(options =>
        options.UseSqlServer(connectionString));
    builder.Services.AddDatabaseDeveloperPageExceptionFilter();

    #region Autofac Configuration
    builder.Host.UseServiceProviderFactory(new AutofacServiceProviderFactory());
    builder.Host.ConfigureContainer<ContainerBuilder>(containerBuilder =>
    {
        containerBuilder.RegisterModule(new WebModule());
    });
    #endregion

    builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount()
        .AddEntityFrameworkStores<ApplicationContext>());
    builder.Services.AddControllersWithViews();

    //Service Collection
    //builder.Services.AddScoped<IEmailSender, EmailSender>();
    //builder.Services.AddKeyedScoped<IEmailSender, HtmlEmailSender>("another");
    builder.Services.AddScoped<IMembership, Membership>();
    builder.Services.AddScoped<IReceiveEmail, ReceiveEmail>();

    var app = builder.Build();

    // Configure the HTTP request pipeline.
    if (app.Environment.IsDevelopment())
    {
        app.UseMigrationsEndPoint();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        // The default HSTS value is 30 days. You may want to change this for production scenarios
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();
}

```

```

        app.UseAuthorization();

        app.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
        app.MapRazorPages();

        app.Run();
    }

    catch(Exception ex)
    {
        Log.Fatal(ex, "Failed to start application.");
    }
    finally
    {
        Log.CloseAndFlush();
    }
}

```

#### 4. Create the Log Table

The configuration `autoCreateSqlTable` is set to `true`, so the table will be created automatically when the application runs. If you prefer to create the table manually or need specific column settings, you can generate the SQL script using the [`Serilog.Sinks.MSSqlServer` package documentation](#).

#### 5. Run your application

Start your application, and the logs should be written to both the file and the MSSQL database as configured.

With these steps, you should have successfully integrated the Serilog MSSQL sink into your existing ASP.NET Core application.



##### Anonymous

what are the changes in program.cs? please tell me which lines did you change from my program.cs to your program.cs tell me all the changed lines in the program.cs file



##### ChatGPT

To integrate the Serilog MSSQL sink into your `Program.cs` file, you need to make a few specific changes. Below, I've outlined the changes line-by-line from your original `Program.cs` to the modified version.

##### Changes in `Program.cs`

1. Add the `Serilog.Sinks.MSSqlServer` using directive:

```
csharp
using Serilog.Sinks.MSSqlServer; // Add this using directive
```

2. Ensure the `var configuration` definition includes the `SerilogConnection` string:

This is already covered in your existing `Program.cs` file. Just make sure the `appsettings.json` includes the `SerilogConnection` string.

3. Modify the Serilog General Configuration to include the MSSQL sink:

```
csharp
#region Serilog General Configuration
builder.Host.UseSerilog((ctx, lc) => lc
    .MinimumLevel.Debug()
    .MinimumLevel.Override("Microsoft", LogEventLevel.Warning)
    .Enrich.FromLogContext()
    .ReadFrom.Configuration(builder.Configuration));
#endregion
```

#### Here's a complete comparison for clarity:

Original lines in your `Program.cs`:

```
csharp
```

```

        using Autofac.Extensions.DependencyInjection;
        using Autofac;
        using DependencyInjectionPractice1.Data;
        using DependencyInjectionPractice1.Models;
        using Microsoft.AspNetCore.Identity;
        using Microsoft.AspNetCore.Identity.UI.Services;
        using Microsoft.EntityFrameworkCore;
        using DependencyInjectionPractice1;
        using Serilog;
        using Serilog.Events;

#region Bootstrap Logger Configuration
var configuration = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("appsettings.json")
    .Build();

Log.Logger = new LoggerConfiguration()
    .ReadFrom.Configuration(configuration)
    .CreateBootstrapLogger();
#endregion

try
{
    Log.Information("Application Starting...");

    var builder = WebApplication.CreateBuilder(args);

    #region Serilog General Configuration
    builder.Host.UseSerilog((ctx, lc) => lc
        .MinimumLevel.Debug()
        .MinimumLevel.Override("Microsoft", LogEventLevel.Warning)
        .Enrich.FromLogContext()
        .ReadFrom.Configuration(builder.Configuration));
    #endregion

    // Add services to the container.
    var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
    builder.Services.AddDbContext<ApplicationContext>(options =>
        options.UseSqlServer(connectionString));
    builder.Services.AddDatabaseDeveloperPageExceptionFilter();

    #region Autofac Configuration
    builder.Host.UseServiceProviderFactory(new AutofacServiceProviderFactory());
    builder.Host.ConfigureContainer<ContainerBuilder>(containerBuilder =>
    {
        containerBuilder.RegisterModule(new WebModule());
    });
    #endregion

    builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true)
        .AddEntityFrameworkStores<ApplicationContext>();
    builder.Services.AddControllersWithViews();

    //Service Collection
    //builder.Services.AddScoped<IEmailSender, EmailSender>();
    //builder.Services.AddKeyedScoped<IEmailSender, HtmlEmailSender>("another");
    builder.Services.AddScoped<IMembership, Membership>();
    builder.Services.AddScoped<IReceiveEmail, ReceiveEmail>();

    var app = builder.Build();

    // Configure the HTTP request pipeline.
    if (app.Environment.IsDevelopment())
    {
        app.UseMigrationsEndPoint();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        // The default HSTS value is 30 days. You may want to change this for product
        app.UseHsts();
    }

    app.UseHttpsRedirection();
}

```

```

        app.UseStaticFiles();

        app.UseRouting();

        app.UseAuthorization();

        app.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
        app.MapRazorPages();

        app.Run();
    }
    catch(Exception ex)
    {
        Log.Fatal(ex, "Failed to start application.");
    }
    finally
    {
        Log.CloseAndFlush();
    }
}

```

Modified lines in your `Program.cs`:

```

csharp
Copy code

using Autofac.Extensions.DependencyInjection;
using Autofac;
using DependencyInjectionPractice1.Data;
using DependencyInjectionPractice1.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.EntityFrameworkCore;
using DependencyInjectionPractice1;
using Serilog;
using Serilog.Events;
using Serilog.Sinks.MSSqlServer; // Add this using directive

#region Bootstrap Logger Configuration
var configuration = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("appsettings.json")
    .Build();

Log.Logger = new LoggerConfiguration()
    .ReadFrom.Configuration(configuration)
    .CreateBootstrapLogger();
#endregion

try
{
    Log.Information("Application Starting...");

    var builder = WebApplication.CreateBuilder(args);

    #region Serilog General Configuration
    builder.Host.UseSerilog((ctx, lc) => lc
        .MinimumLevel.Debug()
        .MinimumLevel.Override("Microsoft", LogEventLevel.Warning)
        .Enrich.FromLogContext()
        .ReadFrom.Configuration(builder.Configuration));
    #endregion

    // Add services to the container.
    var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
    builder.Services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(connectionString));
    builder.Services.AddDatabaseDeveloperPageExceptionFilter();

    #region Autofac Configuration
    builder.Host.UseServiceProviderFactory(new AutofacServiceProviderFactory());
    builder.Host.ConfigureContainer<ContainerBuilder>(containerBuilder =>
    {
        containerBuilder.RegisterModule<new WebModule>());
    });
    #endregion
}

```

```

    .ConfigureServices((builder) =>
    {
        // ...
        #endregion

        builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedEmail = true)
            .AddEntityFrameworkStores<ApplicationDbContext>();
        builder.Services.AddControllersWithViews();

        //Service Collection
        //builder.Services.AddScoped<IEmailSender, EmailSender>();
        //builder.Services.AddKeyedScoped<IEmailSender, HtmlEmailSender>("another");
        builder.Services.AddScoped<IMembership, Membership>();
        builder.Services.AddScoped<IReceiveEmail, ReceiveEmail>();

        var app = builder.Build();

        // Configure the HTTP request pipeline.
        if (app.Environment.IsDevelopment())
        {
            app.UseMigrationsEndPoint();
        }
        else
        {
            app.UseExceptionHandler("/Home/Error");
            // The default HSTS value is 30 days. You may want to change this for production environments.
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseStaticFiles();

        app.UseRouting();

        app.UseAuthorization();

        app.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
        app.MapRazorPages();

        app.Run();
    })
    catch(Exception ex)
    {
        Log.Fatal(ex, "Failed to start application.");
    }
    finally
    {
        Log.CloseAndFlush();
    }
}

```

The key changes are the addition of the `Serilog.Sinks.MSSqlServer` directive and ensuring the Serilog configuration section includes the MSSQL sink. These changes enable logging to an MSSQL database while maintaining the existing logging functionality.

#### Anonymous

so in these 2 codes, my one and your one. there is only one change  
and that is "using Serilog.Sinks.MSSqlServer;"  
am I correct?  
or there is more changes ?

#### ChatGPT

You're correct that the main additional line in the code is the `using Serilog.Sinks.MSSqlServer;` directive. However, the crucial part of configuring the Serilog MSSQL sink is actually done in the `appsettings.json` file, and the code in `Program.cs` already supports reading this configuration. So, technically, the primary change in `Program.cs` is the addition of the `using` statement.

#### Full comparison of changes:

##### 1. Add the using directive:

```
csharp
using Serilog.Sinks.MSSqlServer;
```

2. Ensure `appsettings.json` includes the MSSQL sink configuration:

```
json Copy code  
  
{  
    "ConnectionStrings": {  
        "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-DependencyIn",  
        "SerilogConnection": "Server=(localdb)\\mssqllocaldb;Database=SerilogLogs;Trusted  
    },  
    "Logging": {  
        "LogLevel": {  
            "Default": "Information",  
            "Microsoft.AspNetCore": "Warning"  
        }  
    },  
    "Serilog": {  
        "WriteTo": [  
            {  
                "Name": "File",  
                "Args": {  
                    "path": "Logs/web-log-.log",  
                    "rollingInterval": "Day"  
                }  
            },  
            {  
                "Name": "MSSqlServer",  
                "Args": {  
                    "connectionString": "SerilogConnection",  
                    "tableName": "Logs",  
                    "autoCreateSqlTable": true,  
                    "columnOptionsSection": {  
                        "addStandardColumns": [ "LogEvent" ],  
                        "removeStandardColumns": [ "MessageTemplate" ]  
                    }  
                }  
            }  
        ]  
    },  
    "AllowedHosts": "*"  
}
```

Summary of changes in `Program.cs`:

- Addition of the `using` statement:

```
csharp Copy code  
  
using Serilog.Sinks.MSSqlServer;
```

Recap of all steps:

1. Install the necessary packages:

```
sh Copy code  
  
Install-Package Serilog.Sinks.MSSqlServer
```

2. Update `appsettings.json`:

```
json Copy code  
  
{  
    "ConnectionStrings": {  
        "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-Dependen",  
        "SerilogConnection": "Server=(localdb)\\mssqllocaldb;Database=SerilogLogs;Tru  
    },  
    "Logging": {  
        "LogLevel": {  
            "Default": "Information",  
            "Microsoft.AspNetCore": "Warning"  
        }  
    },  
    "Serilog": {  
        "WriteTo": [  
            {  
                "Name": "File",  
                "Args": {  
                    "path": "Logs/web-log-.log",  
                    "rollingInterval": "Day"  
                }  
            },  
            {  
                "Name": "MSSqlServer",  
                "Args": {  
                    "connectionString": "SerilogConnection",  
                    "tableName": "Logs",  
                    "autoCreateSqlTable": true,  
                    "columnOptionsSection": {  
                        "addStandardColumns": [ "LogEvent" ],  
                        "removeStandardColumns": [ "MessageTemplate" ]  
                    }  
                }  
            }  
        ]  
    },  
    "AllowedHosts": "*"  
}
```

```

    "WriteTo": [
      {
        "Name": "File",
        "Args": {
          "path": "Logs/web-log-.log",
          "rollingInterval": "Day"
        }
      },
      {
        "Name": "MSSqlServer",
        "Args": {
          "connectionString": "SerilogConnection",
          "tableName": "Logs",
          "autoCreateSqlTable": true,
          "columnOptionsSection": {
            "addStandardColumns": [ "LogEvent" ],
            "removeStandardColumns": [ "MessageTemplate" ]
          }
        }
      }
    ],
    "AllowedHosts": "*"
  }
}

```

### 3. Update `Program.cs`:

- Add the `using` directive:

```
csharp
using Serilog.Sinks.MSSqlServer;
```

With these updates, your application will be configured to log to both a file and an MSSQL database.



**Anonymous**

please explain this code "  
 "Name": "MSSqlServer",  
 "Args": {  
 "connectionString": "SerilogConnection",  
 "tableName": "Logs",  
 "autoCreateSqlTable": true,  
 "columnOptionsSection": {  
 "addStandardColumns": [ "LogEvent" ],  
 "removeStandardColumns": [ "MessageTemplate" ]  
 }  
 }  
 }"

step by step, every line like a software engineer, and tell me which lines can I add and which lines I can edit, I mean I don't know anything about this process



**ChatGPT**

Certainly! Let's break down the JSON configuration for the Serilog MSSQL sink step-by-step.

### JSON Configuration Breakdown

```
json
{
  "Name": "MSSqlServer",
  "Args": {
    "connectionString": "SerilogConnection",
    "tableName": "Logs",
    "autoCreateSqlTable": true,
    "columnOptionsSection": {
      "addStandardColumns": [ "LogEvent" ],
      "removeStandardColumns": [ "MessageTemplate" ]
    }
  }
}
```

### Explanation of Each Line

1. `"**Name**": "MSSqlServer"`
  - **Description:** Specifies the sink type. In this case, it's `MSSqlServer`.
  - **Edit:** Do not change this line. It tells Serilog to use the MSSQL sink.
2. `"**Args**": { ... }`
  - **Description:** This object contains the arguments (parameters) required to configure the MSSQL sink.
  - **Edit:** You will be configuring values within this object.
3. `"**connectionString**": "SerilogConnection"`
  - **Description:** Specifies the connection string to use for connecting to the MSSQL database.
  - **Edit:** You can change `"**SerilogConnection**"` to any connection string name defined in your `appsettings.json`. For example, if your connection string is named `MyLoggingDatabase`, you would use that instead.
  - **Example:**

`json` Copy code

```
"connectionString": "MyLoggingDatabase"
```
4. `"**tableName**": "Logs"`
  - **Description:** Specifies the name of the table in which logs will be stored.
  - **Edit:** You can change `"**Logs**"` to any valid table name you want to use in your database.
  - **Example:**

`json` Copy code

```
"tableName": "ApplicationLogs"
```
5. `"**autoCreateSqlTable**": true`
  - **Description:** Indicates whether Serilog should automatically create the SQL table if it does not exist.
  - **Edit:** You can set this to `false` if you want to create the table manually.
  - **Example:**

`json` Copy code

```
"autoCreateSqlTable": false
```
6. `"**columnOptionsSection**": { ... }`
  - **Description:** This object contains additional options for configuring the columns of the log table.
  - **Edit:** You can add, remove, or modify the columns based on your requirements.
7. `"**addStandardColumns**": [ "LogEvent" ]`
  - **Description:** Specifies which standard columns to add to the log table. `LogEvent` is a standard column that includes detailed information about the log event.
  - **Edit:** You can add other standard columns here. Common columns include `TimeStamp`, `Message`, `Level`, etc.
  - **Example:**

`json` Copy code

```
"addStandardColumns": [ "TimeStamp", "Message", "Level", "LogEvent" ]
```
8. `"**removeStandardColumns**": [ "MessageTemplate" ]`
  - **Description:** Specifies which standard columns to remove from the log table. `MessageTemplate` is a standard column that can be excluded if not needed.
  - **Edit:** You can specify other standard columns you wish to exclude.
  - **Example:**

```
json Copy code  
"removeStandardColumns": [ "Properties" ]
```

## Example Configuration

Here's an example of how you might customize the configuration:

```
json Copy code  
{  
  "Name": "MSSqlServer",  
  "Args": {  
    "connectionString": "MyLoggingDatabase",  
    "tableName": "ApplicationLogs",  
    "autoCreateSqlTable": true,  
    "columnOptionsSection": {  
      "addStandardColumns": [ "TimeStamp", "Message", "Level", "LogEvent" ],  
      "removeStandardColumns": [ "MessageTemplate" ]  
    }  
  }  
}
```

## Practical Steps

1. Identify or create a connection string in your `appsettings.json`:

```
json Copy code  
"ConnectionStrings": {  
  "DefaultConnection": "your_default_connection_string_here",  
  "MyLoggingDatabase": "Server=(localdb)\\mssqllocaldb;Database=SerilogLogs;Trust  
  "User=sa;Password=123456;MultipleActiveResultSets=true";  
}
```

2. Update the Serilog configuration in `appsettings.json` to use the new connection string and table name:

```
json Copy code  
"Serilog": {  
  "WriteTo": [  
    {  
      "Name": "File",  
      "Args": {  
        "path": "Logs/web-log-.log",  
        "rollingInterval": "Day"  
      }  
    },  
    {  
      "Name": "MSSqlServer",  
      "Args": {  
        "connectionString": "MyLoggingDatabase",  
        "tableName": "ApplicationLogs",  
        "autoCreateSqlTable": true,  
        "columnOptionsSection": {  
          "addStandardColumns": [ "TimeStamp", "Message", "Level", "LogEvent" ],  
          "removeStandardColumns": [ "MessageTemplate" ]  
        }  
      }  
    }  
  ]  
}
```

By following these steps and understanding each line of the configuration, you can customize the Serilog MSSQL sink to fit your needs.



Anonymous

give me total list of column which I can write in "addStandardColumns":



ChatGPT

When configuring the Serilog MSSQL sink, the `addStandardColumns` section allows you to include standard columns in the SQL table where your logs will be stored. Here's a list of standard columns you

can add:

1. `Id`: A unique identifier for each log entry (usually an `int` or `Guid`).
2. `Message`: The log message.
3. `MessageTemplate`: The template of the message before properties are applied.
4. `Level`: The log level (e.g., Information, Warning, Error).
5. `TimeStamp`: The timestamp of when the log entry was created.
6. `Exception`: Details of any exception that was logged.
7. `Properties`: A JSON object containing additional properties.
8. `LogEvent`: A serialized representation of the entire log event.

## Full List of Standard Columns

- `Id`: Automatically included if `autoCreateSqlTable` is true.
- `Message`: The main log message.
- `MessageTemplate`: The message template.
- `Level`: The severity level of the log (Debug, Information, Warning, Error, Fatal).
- `TimeStamp`: The date and time when the log event occurred.
- `Exception`: The exception details, if any.
- `Properties`: Additional properties associated with the log event, stored as JSON.
- `LogEvent`: The complete log event as a serialized JSON object.

## Example Usage in `appsettings.json`

To include all possible standard columns, your configuration might look like this:

```
json
Copy code

{
  "Name": "MSSqlServer",
  "Args": {
    "connectionString": "MyLoggingDatabase",
    "tableName": "ApplicationLogs",
    "autoCreateSqlTable": true,
    "columnOptionsSection": {
      "addStandardColumns": [ "Id", "Message", "MessageTemplate", "Level", "TimeStamp" ]
    }
  }
}
```

## Explanation of Each Column

- `Id`: A primary key for each log entry.
- `Message`: The main content of the log entry, which is what you see in your logs.
- `MessageTemplate`: The message format string, useful for understanding the structure of the log message.
- `Level`: Indicates the log level, such as Debug, Information, Warning, Error, or Fatal.
- `TimeStamp`: The exact time when the log was recorded.
- `Exception`: The full exception information, if an error occurred.
- `Properties`: Any additional data that is logged with the event, serialized as JSON.
- `LogEvent`: The entire log event, including all data, serialized into a JSON object.

By adding these columns to the `addStandardColumns` array, you can customize the schema of your logs table in the database to include all necessary information for your logging requirements.

[Sign up to chat](#)