Configuring Serilog through appsettings.json file

Mohsen Esmailpour · Follow
5 min read · Mar 7, 2023

30 1

In this post, I'm going to show you how to configure Serilog via the configuration file. Changing the logging configuration without touching the codebase is really helpful, especially in the production environment. First why Serilog? It is easy to set up, has a clean API, and is portable between recent .NET platforms. The big difference between Serilog and the other frameworks is that it is designed to do structured logging out of the box. Another thing I really like about Serilog is that it can be configured via the `appsetting.json` file alongside configuring through code.

Let’s start with creating a new project

Step 1 — New project

Create a new ASP.NET 7.0 Web API project.

Step 2 — Install package

Install Serilog.AspNetCore nuget package.

Step 3 — Add `UseSerilog` extension method

Open `Program.cs` file and add the following code:

```
builder.Host.UseSerilog((hostingContext, loggerConfiguration) =>
    loggerConfiguration.ReadFrom.Configuration(hostingContext.Configuration));
```

`UseSerilog` sets Serilog as the logging provider. We are going to config Serilog via the `appsettings.json` file.

Step 4 — Remove the default configuration

Open `appsettings.json` and `appsettings.Development.json` files and get rid of

th

```
"Logging": {
  "LogLevel": {
    "Default": "Information",
    "Microsoft": "Warning",
    "Microsoft.Hosting.Lifetime": "Information"
  }
}
```

Step 5: Add Serilog configuration

Add Serilog configuration section to `appsettings.Development.json` file:

```
"Serilog": {
  "MinimumLevel": {
    "Default": "Debug",
    "Override": {
      "Microsoft": "Information",
      "System": "Information"
    },
    "Using": [ ],
  },
  "WriteTo": [
    { }
    ]
```

`Serilog.AspNetCore` nuget package has a dependency on Serilog.Settings.Configuration nuget package and is a Serilog settings provider that reads from `Microsoft.Extensions.Configuration` sources. The above configuration is equivalent to this:

```
Log.Logger = new LoggerConfiguration()
    .MinimumLevel.Debug()
    .MinimumLevel.Override("Microsoft", LogEventLevel.Information)
    .MinimumLevel.Override("System", LogEventLevel.Information)
    .Enrich.FromLogContext()
    .CreateLogger();
```

Step 6 — Installing Sinks

Serilog uses sinks to write log events to storage, for example, database, file, etc. One of the most popular sinks for the debugging environment is the Console sink.

- Install `Serilog.Sinks.Console` nuget package

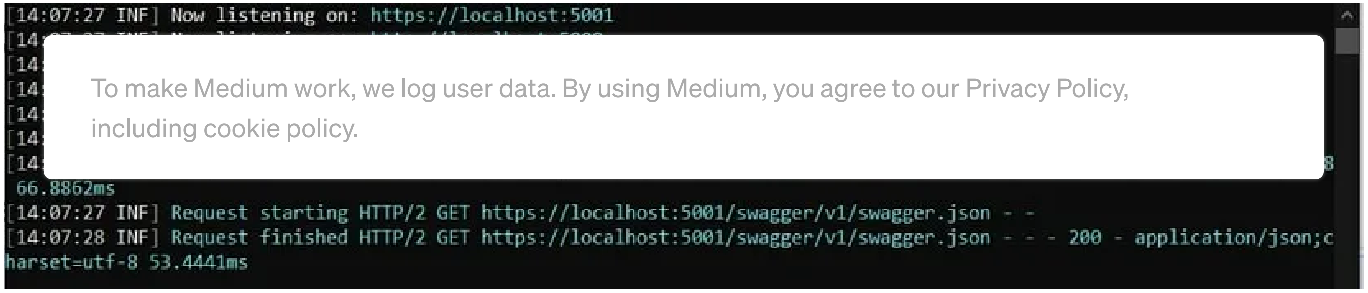
- Add the following configuration:

```
"Serilog": {
  "MinimumLevel": {
    "Default": "Debug",
    "Override": {
      "Microsoft": "Information",
      "System": "Information"
    }
  },
  "Using": [ "Serilog.Sinks.Console" ],
  "WriteTo": [
    { "Name": "Console" }
  ]
}
```

In the `Using` section add the sink's nuget package name `"Using": [ "Serilog.Sinks.Console" ]`

- In the `WriteTo` section add sink name and arguments `"WriteTo":[ { "Name": "Console" } ]`

Run the project and in the console window you should see logs like below



```
[14:07:27 INF] Now listening on: https://localhost:5001
[14
[14
[14
[14
[14
[14                                                                                              8
 66.8862ms
[14:07:27 INF] Request starting HTTP/2 GET https://localhost:5001/swagger/v1/swagger.json - -
[14:07:28 INF] Request finished HTTP/2 GET https://localhost:5001/swagger/v1/swagger.json - - - 200 - application/json;c
harset=utf-8 53.4441ms
```

Now we want to use SQL Server sink for other environments:

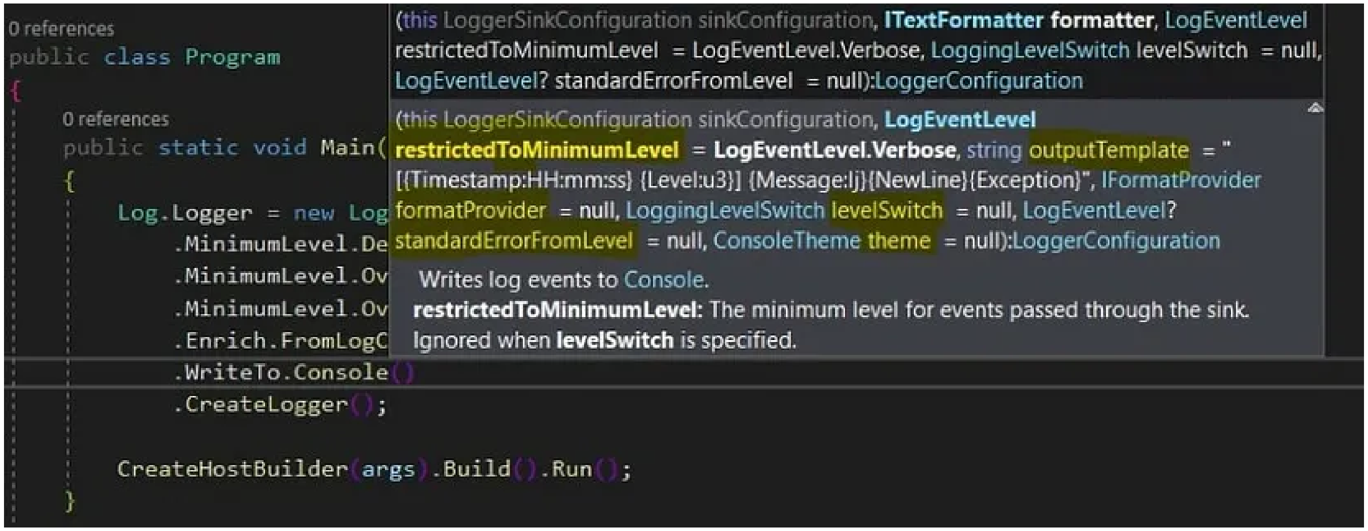
- Install Serilog.Sinks.MSSqlServer sink
- Copy Serilog setting form `appsettings.Development.json` to `appsettings.json` file

```
"Serilog": {
  "MinimumLevel": {
    "Default": "Information",
    "Override": {
      "Microsoft": "Error",
      "System": "Error"
    },
    "Using": [ "Serilog.Sinks.MSSqlServer" ]
  },
  "WriteTo": [
    {
      "Name": "MSSqlServer",
      "Args": {
        "connectionString": "ConnectionString",
        "tableName": "Logs",
        "autoCreateSqlTable": true
```

```
}
```

Step 7 — How to configure a sink

Well, configuring a sink via `appsettings.json` could be harder than configuring through the code, and for each sink, you might not be able to find a JSON configuration sample. Normally each sink accepts several parameters to configure the sink. For instance, the Console sink accepts the below parameters:



```
0 references
public class Program
{
    0 references
    public static void Main(
    {
        Log.Logger = new Log
            .MinimumLevel.De
            .MinimumLevel.Ov
            .MinimumLevel.Ov
            .Enrich.FromLogC
            .WriteTo.Console()
            .CreateLogger();

        CreateHostBuilder(args).Build().Run();
    }
```

(this LoggerSinkConfiguration sinkConfiguration, ITextFormatter formatter, LogEventLevel restrictedToMinimumLevel = LogEventLevel.Verbose, LoggingLevelSwitch levelSwitch = null, LogEventLevel? standardErrorFromLevel = null):LoggerConfiguration

(this LoggerSinkConfiguration sinkConfiguration, LogEventLevel restrictedToMinimumLevel = LogEventLevel.Verbose, string outputTemplate = "[{Timestamp:HH:mm:ss} {Level:u3}] {Message:lj}{NewLine}{Exception}", IFormatProvider formatProvider = null, LoggingLevelSwitch levelSwitch = null, LogEventLevel? standardErrorFromLevel = null, ConsoleTheme theme = null):LoggerConfiguration

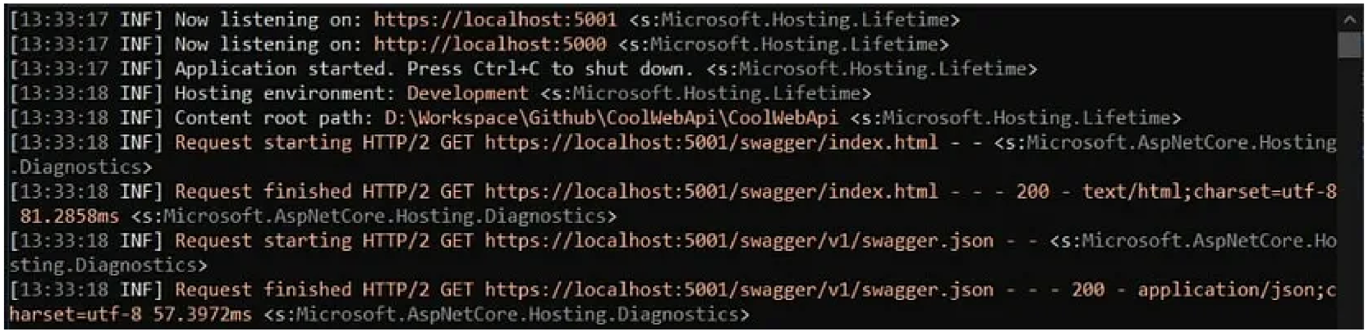
Writes log events to Console.

restrictedToMinimumLevel: The minimum level for events passed through the sink. Ignored when **levelSwitch** is specified.

Each one of these parameters can be configured through JSON:

```
"WriteTo": [
  {
    "Name": "Console",
    "Args": {
      "restrictedToMinimumLevel": "Verbose",
      "outputTemplate": "[{Timestamp:HH:mm:ss} {Level:u3}] {Message:lj} <s:{Sour
      "theme": "Serilog.Sinks.SystemConsole.Themes.AnsiConsoleTheme::Code, Seril
    }
  }
]
```

After setting the console sink parameters, logs should be like the following image:



```
[13:33:17 INF] Now listening on: https://localhost:5001 <s:Microsoft.Hosting.Lifetime>
[13:33:17 INF] Now listening on: http://localhost:5000 <s:Microsoft.Hosting.Lifetime>
[13:33:17 INF] Application started. Press Ctrl+C to shut down. <s:Microsoft.Hosting.Lifetime>
[13:33:18 INF] Hosting environment: Development <s:Microsoft.Hosting.Lifetime>
[13:33:18 INF] Content root path: D:\Workspace\Github\CoolWebApi\CoolWebApi <s:Microsoft.Hosting.Lifetime>
[13:33:18 INF] Request starting HTTP/2 GET https://localhost:5001/swagger/index.html - - <s:Microsoft.AspNetCore.Hosting
.Diagnostics>
[13:33:18 INF] Request finished HTTP/2 GET https://localhost:5001/swagger/index.html - - - 200 - text/html;charset=utf-8
 81.2858ms <s:Microsoft.AspNetCore.Hosting.Diagnostics>
[13:33:18 INF] Request starting HTTP/2 GET https://localhost:5001/swagger/v1/swagger.json - - <s:Microsoft.AspNetCore.Ho
sting.Diagnostics>
[13:33:18 INF] Request finished HTTP/2 GET https://localhost:5001/swagger/v1/swagger.json - - - 200 - application/json;c
harset=utf-8 57.3972ms <s:Microsoft.AspNetCore.Hosting.Diagnostics>
```

To see the complete SQL Server sink JSON configuration check out this

ap

Step 8 — Enrichers

Log events can be enriched with properties in various ways. You can add additional data by enrichers. For instance, in the production environment, we want to add the IP of the client to the log events.

- Install Serilog.Enrichers.ClientInfo package
- Add enriched package name to `Using` section
- Add `Enrich` section with `WithClientIp` value (enriched name normally starts with `With` word)

```
"Using": [ "Serilog.Sinks.MSSqlServer", "Serilog.Enrichers.ClientInfo" ],
"Enrich": [ "WithClientIp" ]
```

All events written through the logger will carry a property `ClientIp` with the IP of the client. Check out the list of available enrichers here.

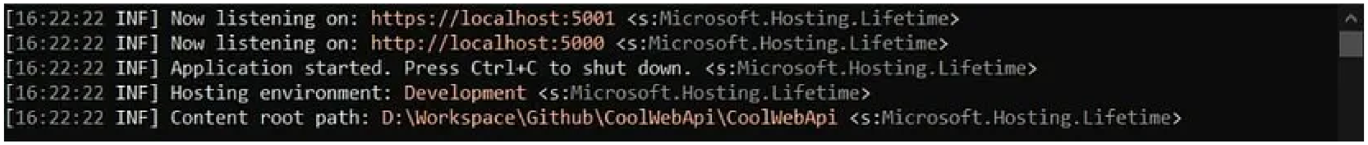
Step 9 — Filters

By

- Install `Serilog.Expressions` nuget package
- Add the “Filter” section to Serilog settings

```
"Filter": [
  {
    "Name": "ByExcluding",
    "Args": {
      "expression": "RequestPath like '%swagger%'"
    }
  }
]
```

All log events that contain `swagger` will be excluded.



```
[16:22:22 INF] Now listening on: https://localhost:5001 <s:Microsoft.Hosting.Lifetime>
[16:22:22 INF] Now listening on: http://localhost:5000 <s:Microsoft.Hosting.Lifetime>
[16:22:22 INF] Application started. Press Ctrl+C to shut down. <s:Microsoft.Hosting.Lifetime>
[16:22:22 INF] Hosting environment: Development <s:Microsoft.Hosting.Lifetime>
[16:22:22 INF] Content root path: D:\Workspace\Github\CoolWebApi\CoolWebApi <s:Microsoft.Hosting.Lifetime>
```

Step 10 — HTTP requests logging

Moreover, you can log the HTTP requests. `UseSerilogRequestLogging` extension

ad

re

details in several events, this middleware collects information during the request (including from `Serilog.IDiagnosticContext`), and writes a single event at request completion.

- In `Program.cs` file, add the following code:

```
var app = builder.Build();
app.UseSerilogRequestLogging();
```

- In `MinimumLevel.Override` section add `"Microsoft.AspNetCore": "Warning"`:

```
"MinimumLevel": {
  "Default": "Information",
  "Override": {
    "Microsoft": "Error",
    "Microsoft.AspNetCore": "Warning",
    "System": "Error"
  },
```

Step 11 — Overriding configuration in docker

La

setting by the Docker environment variable. Consider the following configuration:

```
"Serilog": {
  "MinimumLevel": {
    "Default": "Information",
    "Override": {
      "Microsoft": "Error",
      "System": "Error"
    },
```

```
    {
      "Name": "MSSqlServer",
      "Args": {
        "connectionString": "",
        "tableName": "Logs",
        "autoCreateSqlTable": true
      }
    }
  ]
}
```

Now in the docker-compose file, we want to pass the actual connection

str

```
my-api:
    environment:
      - ASPNETCORE_ENVIRONMENT=Production
      - Serilog__MinimumLevel__Default=Warning
      - Serilog__WriteTo__0__Args__connectionString="Your connection string"
```

The value of each section can be accessed by `__` , for instance, `Serilog__MinimumLevel__Default` is equivalent to:

```
"Serilog": {
  "MinimumLevel": {
    "Default": "",
```

In a section to access an item inside the array, use the item index number. `"WriteTo"` section accepts an array of sinks configuration. If you are using two sinks use `Serilog__WriteTo__0__` to access the first sink and `Serilog__WriteTo__1__` to access the second sink.

Step 12 -Test

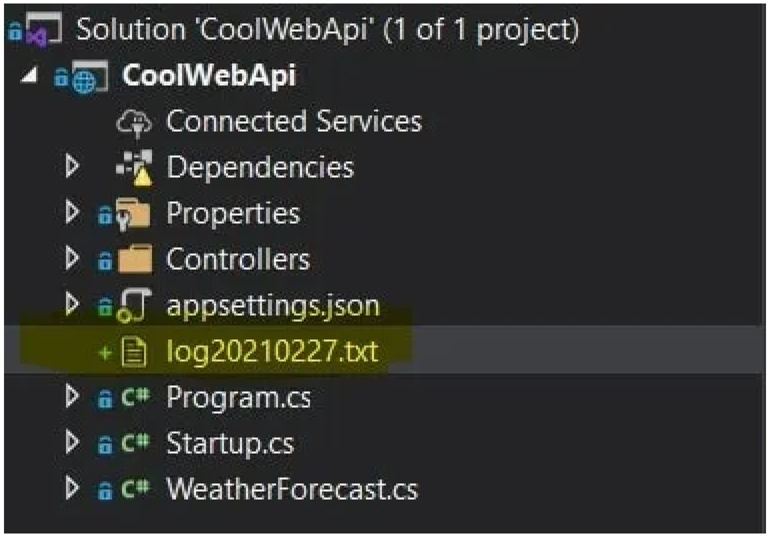
Le

- Type `dotnet add package Serilog.Sinks.File` to install File sink
- Open `appsettings.josn` file and change the logging configuration like this:

```
"Serilog": {
  "MinimumLevel": {
    "Default": "Debug",
    "Override": {
      "Microsoft": "Information",
      "System": "Information"
    }
  },
  "Using": [ "Serilog.Sinks.Console", "Serilog.Sinks.File" ],
  "WriteTo": [
    { "Name": "Console" },
    {
      "Name": "File",
      "Args": {
        "path": "log.txt",
        "rollingInterval": "Day"
      }
    }
  ]
}
```

- Type `dotnet build` then `dotnet run`

- directory



Solution 'CoolWebApi' (1 of 1 project)
CoolWebApi
Connected Services
Dependencies
Properties
Controllers
appsettings.json
log20210227.txt
Program.cs
Startup.cs
WeatherForecast.cs

The source code for this walkthrough could be found on GitHub.

Aspnetcore · Serilog · Logging · Dotnet Core

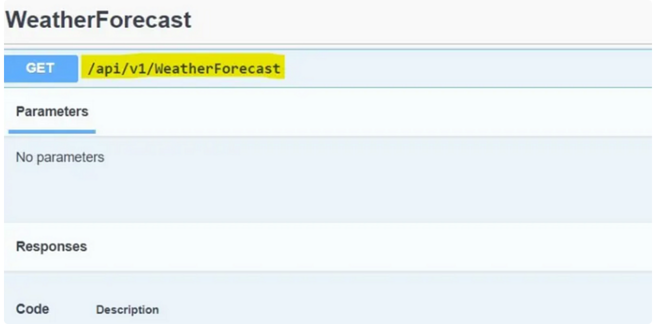
Written by Mohsen Esmailpour

Follow

24 Followers

A software developer who turns coffee into codes

More from Mohsen Esmailpour



WeatherForecast

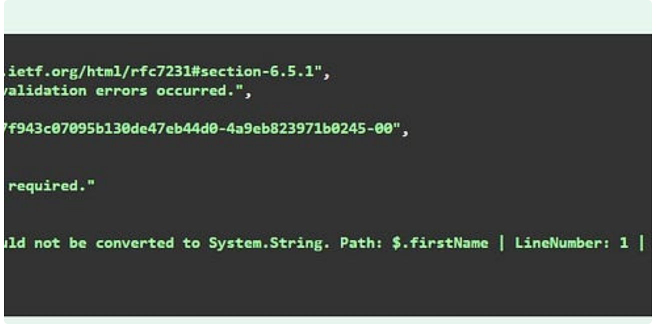
GET /api/v1/WeatherForecast

Parameters

No parameters

Responses

Code Description



```
.ietf.org/html/rfc7231#section-6.5.1",
validation errors occurred.",

7f943c07095b130de47eb44d0-4a9eb823971b0245-00",


 required."


uld not be converted to System.String. Path: $.firstName | LineNumber: 1 |
```

Mohsen Esmailpour

API versioning and Swagger in ASP.NET Core 7.0

In this article, I'm going to show how to add the API version and adjust Swagger to show...

Mohsen Esmailpour

Override System.Text.Json.JsonSerializer...

In this blog post, I'm going to show how to override the default error message when...

8 min read · May 10, 2023

2 min read · Mar 9, 2023

See all from Mohsen Esmailpour

Recommended from Medium

Winson Yau

How to Use the Serilog in .Net Core

Brucy Centeio

Adding Serilog to ASP.NET Core .NET 7 & 8

In t

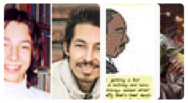
cre

9 m

16 1

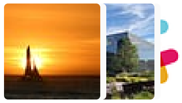
65

Lists



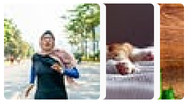
Staff Picks

557 stories · 641 saves



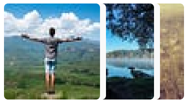
Stories to Help You Level-Up at Work

19 stories · 418 saves



Self-Improvement 101

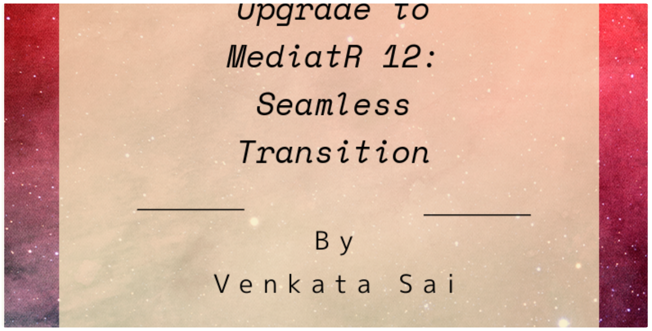
20 stories · 1211 saves



Productivity 101

20 stories · 1107 saves

Divyansh Bhatia



Saisiva

Implement Logging in .NET Core us

In t

logging plays a pivotal role in understanding...

3 min read · Aug 23, 2023

Upgrading MediatR from Version

for clean, maintainable, and scalable code...

3 min read · Aug 18, 2023

11

LOGGING WITH ELASTICSEARCH + KIBANA + DOCKER AND ASP.NET CORE
elasticsearch
kibana
docker
.NET Core

Tohid haghighi

Logging with ElasticSearch, Kibana, ASP.NET Core and Docker

A Step by Step Guide to Logging with ElasticSearch, Kibana, ASP.NET Core 3.1 and...

10 min read · Jul 21, 2023

61

Dushyantha Kalehewatte

Best Practices for Developing Middleware in ASP.NET Core

Middleware in ASP.NET Core plays a crucial role in processing HTTP requests and...

3 min read · Oct 5, 2023

106 2

Help Status About Careers Blog Privacy Terms Text to speech Teams