How to Configure Rolling File Logging With Serilog

Posted by **Code Maze** | Updated Date Aug 10, 2023 | **0**

Want to build **great APIs?** Or become **even better** at it? Check our **Ultimate ASP.NET Core Web API program** and learn how to create a full production-ready ASP.NET Core API using only the latest .NET technologies. Bonus materials (Security book, Docker book, and other bonus files) are included in the Premium package!

Logging holds a crucial role in software development, as it provides software engineers valuable insights into the behavior and health of their applications. Moreover, Serilog, a popular logging framework for .NET applications, offers a wide range of features. One particular feature is its ability to log into various sinks. Among these, the file sink is definitely a good default option. In this article, we will explore how to configure rolling file logging with Serilog.

For a more detailed guide on Serilog, refer to the article **Structured Logging in ASP.NET Core With Serilog**.

> To download the source code for this article, you can visit our **GitHub repository**.

Let's dig in.

Getting Started With Serilog File Sink

First, let's set up a simple .NET Core Web API application. We'll need to install the `Serilog.Sinks.File` NuGet package:

```
PM> Install-Package Serilog.Sinks.File
```


To continue, let's inspect how we can configure rolling file logging with Serilog. We do this in the main file – `Program.cs`:

```
var log = new LoggerConfiguration()
    .WriteTo.File("logs/log.txt",
    rollingInterval: RollingInterval.Day,
    rollOnFileSizeLimit: true)
    .CreateLogger();
```

We configure the `rollingInterval` parameter to use `RollingInterval.Day`, ensuring that a new log file is created every day. Additionally, we enable rolling based on file size by setting `rollOnFileSizeLimit` to `true`.

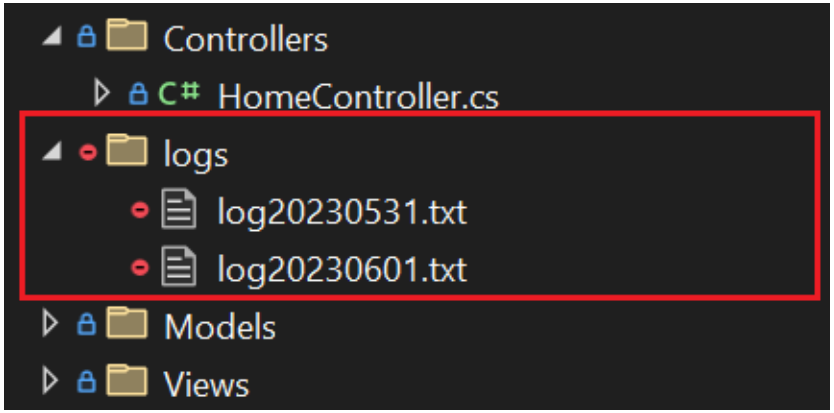
With our configuration in place, let's navigate into the `Controllers` directory and then open the `HomeController` class. There, we can update the code in the `Index` method:

```
public IActionResult Index()
{
    _logger.LogInformation("TESTING MESSAGE 123..");

    return View();
}
```

The `Index` method in our `HomeController` class logs an informational-level message using the `_logger.LogInformation` method.

Upon running the application and browsing around, we can expect to observe the creation of a new logs directory at the project root:



The logs directory contains log files, for example, `log20230531.txt`. It's important to note that, based on our configuration, our app will create a new log file on a daily basis. Upon further inspection of the latest created log file, such as `log20230601.txt`, we shall observe our informational log entry:

```
2023-06-01 10:29:34.571 +03:00 [INF] TESTING MESSAGE 123..
```

Rolling Policies in Serilog File Sink

The `Serilog.Sinks.File` NuGet package supports several rolling policies to control new log file creation. That said, let's inspect the ones developers commonly use:

- `RollingInterval`
- `FileSizeLimitBytes`
- `RetainedFileCountLimit`
- `RollOnFileSizeLimit`

RollingInterval

This policy specifies the time interval after which a new log file should be created. Available options are: `RollingInterval.Year`, `RollingInterval.Month`, `RollingInterval.Day`, `RollingInterval.Hour`, `RollingInterval.Minute` and `RollingInterval.Second.`

Please refer to the sample implementation we have in the **Getting Started section**.

FileSizeLimitBytes

The purpose of setting the `FileSizeLimitBytes` is to control the maximum size of individual log files:

```
var log = new LoggerConfiguration()
.WriteTo.File("logs/log.txt",
  fileSizeLimitBytes: 524288000,
  rollOnFileSizeLimit: true)
  .CreateLogger();
```

By specifying a file size limit using `fileSizeLimitBytes` parameter and additionally setting `rollOnFileSizeLimit` parameter to `true`, we ensure that log files do not grow indefinitely and become too large to manage and consume effectively. By default, the file size limit is 1 GB. Software developers often find smaller log files easier to search, analyze, and troubleshoot issues using a viewer.

Once we reach the limit of 500 megabytes (524,288,000 bytes), Serilog will create a new log file and continue writing logs into it.

RetainedFileCountLimit

Developers consider this to be an important policy, we use it to specify the maximum number of log files to keep:

```
var log = new LoggerConfiguration()
    .WriteTo.File("logs/log.txt",
     retainedFileCountLimit: 21,
     rollingInterval: RollingInterval.Day)
```

```
        .CreateLogger();
```

With this configuration, we ensure that the old log files will be cleaned up in accordance with `retainedFileCountLimit` parameter. The default value is `31`. The oldest log file will be deleted if the number of files exceeds this limit.

RollOnFileSizeLimit

In regard to the `FileSizeLimitBytes` rolling policy, it is essential to note that further setting the `rollOnFileSizeLimit` to `true` triggers the creation of a new log file when we reach the specified `fileSizeLimitBytes`.

On the other hand, if we don't enable file size-based rollover by setting the property to `false`, the sink stops writing any new events to the file once the limit is reached. This is important to be aware of, as it may result in potential loss of information.

content and **get paid?** **>> JOIN US! <<**

Conclusion

In this article, we explored how to configure rolling file logging using `Serilog.Sinks.File`. By utilizing rolling policies such as `FileSizeLimitBytes` and `RetainedFileCountLimit`, we can effectively manage log files and ensure that they don't grow too large or clutter the file system. With Serilog we can flexibly and efficiently handle log files in a .NET application.

Remember to fine-tune the rolling policies according to the application's logging needs and disk space constraints. Additionally, Serilog and its rich ecosystem of sinks and extensions empower us to enhance our logging capabilities and gain valuable insights into the application's behavior.

Subscribe Login

Be the First to Comment!

0 COMMENTS

Privacy