

# Using Background Processing to Build Scalable Applications with Hangfire

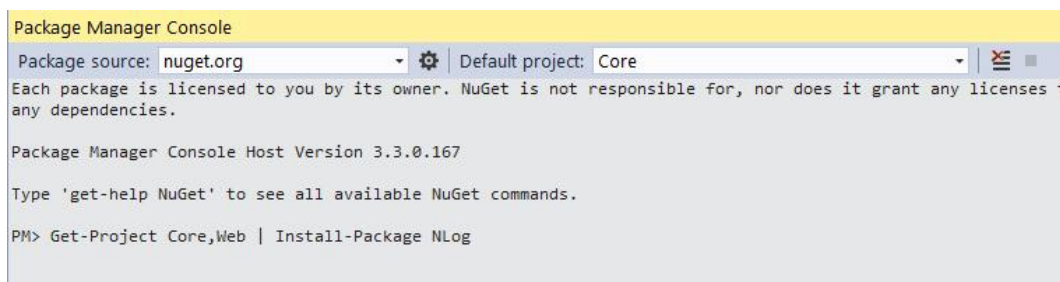
## Lab 3

### Goal

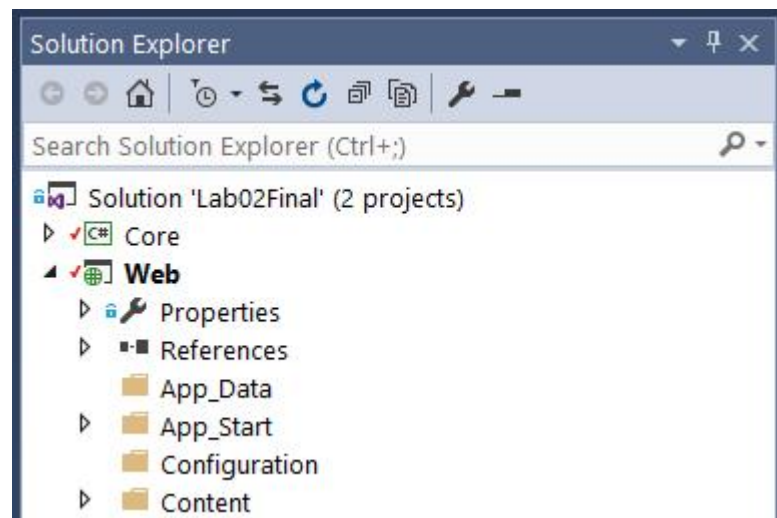
Since our jobs are now executing in the background, we must rely on logging to understand how they are performing. In this exercise, we will configure our application to use NLog to write log events to the console and to our database.

1. To install NLog, open the Package Manager Console from Visual Studio (Tools \ NuGet Package Manager \ Package Manager Console).
2. We need to install the NLog NuGet package to both the Core and Web projects. At the `PM>` prompt, type

```
PM> Get-Project Core,Web | Install-Package NLog.
```



3. Next, we need to configure NLog. In the Web project, create a folder named Configuration.



4. In the Configuration folder, add a new class called Bootstrap.cs.
5. Copy the following into Bootstrap.cs file:

```

using NLog;
using NLog.Config;
using NLog.Layouts;
using NLog.Targets;

namespace Web.Configuration
{
    public static class Bootstrap
    {
        public static void Start()
        {
            BootstrapNLog();
        }

        private static void BootstrapNLog()
        {
            var config = new LoggingConfiguration();

            var colorConsoleTarget = new ColoredConsoleTarget()
            {
                Name = "colorConsole"
            };

            config.AddTarget("colorConsole", colorConsoleTarget);
            config.LoggingRules.Add(new LoggingRule("*", LogLevel.Trace, colorConsole
Target));

            var databaseTarget = new DatabaseTarget()
            {
                ConnectionString =
System.Configuration.ConfigurationManager.ConnectionStrings["DefaultConnection"].Conn
ectionString,
                CommandText = "INSERT INTO EventLog (Logger, [TimeStamp], Level,
ServiceInstanceId, JobId, Message, Exception, StackTrace) " +
                    "VALUES (@logger, @timeStamp, @level,
@serviceinstanceid, @jobid, " +
                    "CASE WHEN LEN(@message) > 4000 THEN LEFT(@message,
3988) + '[truncated]' ELSE @message END," +
                    "CASE WHEN LEN(@exception) > 4000 THEN LEFT(@exception,
3988) + '[truncated]' ELSE @exception END," +
                    "CASE WHEN LEN(@stacktrace) > 4000 THEN
LEFT(@stacktrace, 3988) + '[truncated]' ELSE @stacktrace END" +
                    ")",
                Parameters =
                {
                    new DatabaseParameterInfo("@logger", new
SimpleLayout("${logger}")),
                    new DatabaseParameterInfo("@timestamp", new
SimpleLayout("${date}")),
                    new DatabaseParameterInfo("@level", new
SimpleLayout("${level}")),
                    new DatabaseParameterInfo("@serviceinstanceid", new
SimpleLayout("${gdc:item=ServiceInstanceId}")),

```

```

        new DatabaseParameterInfo("@jobid", new
SimpleLayout("${mdc:item=JobId}")),
        new DatabaseParameterInfo("@message", new
SimpleLayout("${message}")),
        new DatabaseParameterInfo("@exception", new
SimpleLayout("${exception}")),
        new DatabaseParameterInfo("@stacktrace", new
SimpleLayout("${exception:stacktrace}")),
    }
};

config.AddTarget("database", databaseTarget);

config.LoggingRules.Add(new LoggingRule("*", LogLevel.Info, databaseTarget));
LogManager.Configuration = config;
}
}
}

```

Inside of the static *BootstrapNLog* method, we are configuring NLog with both a Colored Console Target and a Database Target.

The Colored Console Target will be used later when debugging our Hangfire service. It is configured with a logging rule to direct all messages of Trace level, or higher, to the Colored Console Target.

The Database Target can be used not only when debugging our application, but also while running in production. It is configured with a logging rule to direct all message of Info level, or higher, to the Database Target.

Note: other logging targets can be configured according to your needs. See the NLog documentation for more information.

6. Once our bootstrapping class is complete, we must call the static *Start* method when our application starts. Open the *Global.asax.cs* file, and add the following using statement to the file:

```
using Web.Configuration;
```

7. Next, add the following line to the end of the *Application\_Start* method:

```
Bootstrap.Start();
```

8. Before we can take advantage of this logging configuration in our application, we must add a table to the database for our log messages. We will do this by adding an additional entity to our Entity Framework model. In the Core project, add a new class called *EventLogEntry.cs* to the Data folder.
9. Copy the following into the *EventLogEntry.cs* file.

```

using System;

namespace Core.Data
{
    public class EventLogEntry
    {
        public int Id { get; set; }
        public string Logger { get; set; }
        public DateTime Timestamp { get; set; }
        public string Level { get; set; }
        public string Message { get; set; }
        public string Exception { get; set; }
        public string StackTrace { get; set; }
        public string ServiceInstanceId { get; set; }
        public string JobId { get; set; }
    }
}

```

10. Next, add a new class called EventLogEntryMapping.cs to the Mapping folder in the Core project.

11. Copy the following into the EventLogEntryMapping.cs file.

```

using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity.ModelConfiguration;
using Core.Data;

namespace Core.Mapping
{
    public class EventLogEntryMapping : EntityTypeConfiguration<EventLogEntry>
    {
        public EventLogEntryMapping()
        {
            ToTable("EventLog");

            HasKey(p => p.Id);

            Property(p => p.Id).HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);

            Property(p => p.Logger).HasMaxLength(50);
            Property(p => p.Level).HasMaxLength(10);
            Property(p => p.Message).HasMaxLength(4000);
            Property(p => p.Exception).HasMaxLength(4000);
            Property(p => p.StackTrace).HasMaxLength(4000);
            Property(p => p.ServiceInstanceId).HasMaxLength(32);
            Property(p => p.JobId).HasMaxLength(32);
        }
    }
}

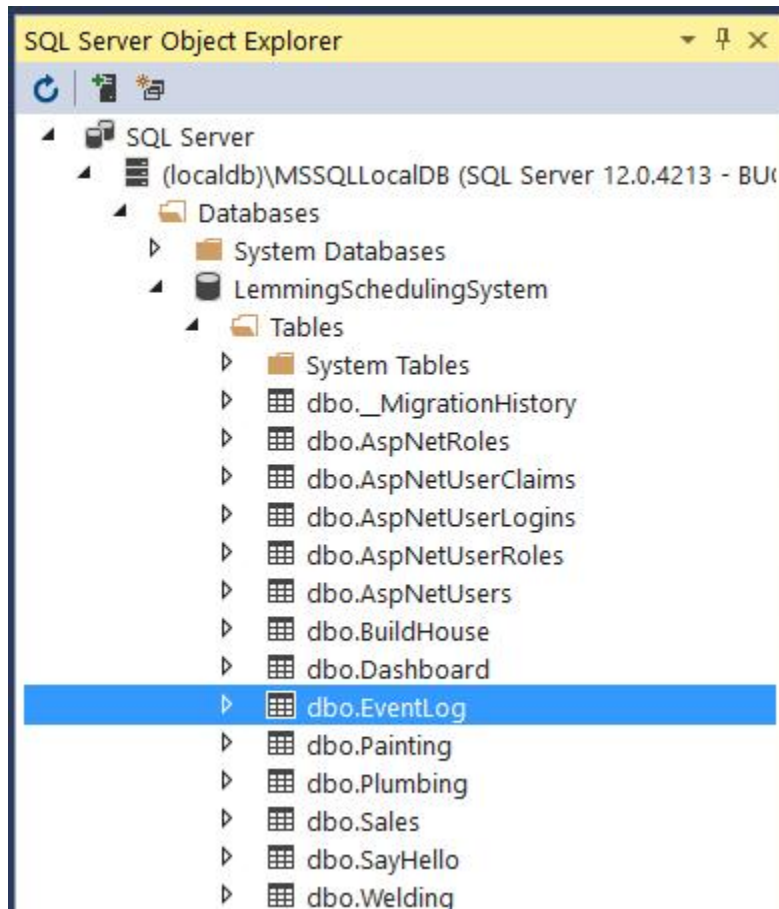
```

12. To apply our changes, we must run the Update-Database cmdlet. Open the Package Manager Console (Tools \ NuGet Package Manager \ Package Manager Console).

13. At the PM> prompt, type:

```
PM> Update-Database -ProjectName Core
```

14. To verify that the table was properly added, open Visual Studio's SQL Server Data Explorer (View \ SQL Server Data Explorer).
15. Navigate to the (localdb)\MSSQLLocalDB server, listed under SQL Servers.
16. Navigate to the LemmingSchedulingSystem database.
17. Expand Tables, and verify that the dbo.EventLog table exists.



18. Now that the database is ready to receive log messages, it is time to update our application to take advantage of our logging framework. In the Core project, open the BaseLemmingJob class in the Jobs folder.
19. Add the following using statement to the file:

```
using NLog;
```

20. Next, add the following member to the BaseLemmingJob class:

```
protected readonly Logger Logger = LogManager.GetCurrentClassLogger();
```

21. Next, replace the following line in the Run method:

```
System.Diagnostics.Debug.WriteLine($"Running job of type {GetType().Name}.")
```

;

With the following line:

```
Logger.Info($"Running job of type {GetType().Name}.");
```

Here we have refactored our BaseLemmingJob to use NLog to log our messages, rather than simply writing to the Output window.

22. Now we are ready to run our application and see that our logging is in place. Press F5 to start debugging.


23. When the application appears in your browser, queue some Lemming jobs.

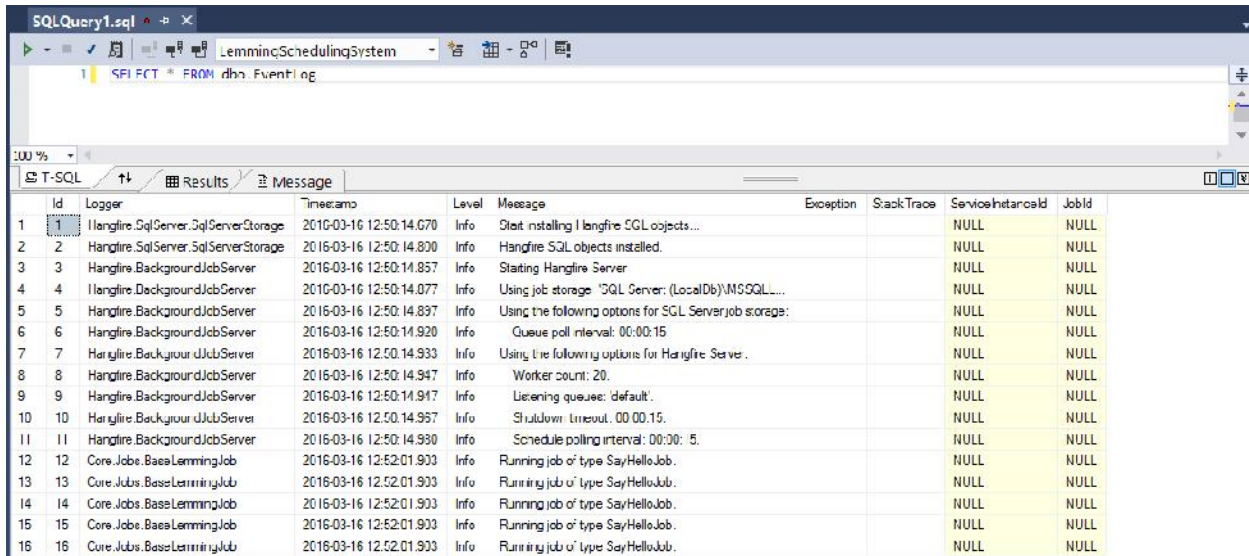
24. To verify the log messages, open the SQL Server Object Explorer in Visual Studio. Navigate to SQL Server \ (localdb)\MSSQLLocalDB \ Databases \ LemmingSchedulingSystem.

25. Right-click on the LemmingSchedulingSystem database, and select New Query...

26. In the Query Editor, type the query:

```
SELECT * FROM dbo.EventLog
```

27. Click the Execute  button. The results pane should show a list of log messages.



Id	Logger	TimeStamp	Level	Message	Exception	StackTrace	ServiceName	JobId
1	Hangfire.SqlServer.SqlServerStorage	2016-03-16 12:50:14.670	Info	Start installing Hangfire SQL objects...			NULL	NULL
2	Hangfire.SqlServer.SqlServerStorage	2016-03-16 12:50:14.800	Info	Hangfire SQL objects installed.			NULL	NULL
3	Hangfire.BackgroundJobServer	2016-03-16 12:50:14.857	Info	Starting Hangfire Server			NULL	NULL
4	Hangfire.BackgroundJobServer	2016-03-16 12:50:14.877	Info	Using job storage 'SQL Server: (LocalDb)\MSSQL...			NULL	NULL
5	Hangfire.BackgroundJobServer	2016-03-16 12:50:14.897	Info	Using the following options for SQL Server job storage:			NULL	NULL
6	Hangfire.BackgroundJobServer	2016-03-16 12:50:14.920	Info	Queue poll interval: 00:00:15			NULL	NULL
7	Hangfire.BackgroundJobServer	2016-03-16 12:50:14.933	Info	Using the following options for Hangfire Server:			NULL	NULL
8	Hangfire.BackgroundJobServer	2016-03-16 12:50:14.947	Info	Worker count: 20.			NULL	NULL
9	Hangfire.BackgroundJobServer	2016-03-16 12:50:14.947	Info	Listening queues: default.			NULL	NULL
10	Hangfire.BackgroundJobServer	2016-03-16 12:50:14.957	Info	Shutdown timeout: 00:00:15.			NULL	NULL
11	Hangfire.BackgroundJobServer	2016-03-16 12:50:14.980	Info	Schedule polling interval: 00:00:05.			NULL	NULL
12	Core.Jobs.BaseLemmingJob	2016-03-16 12:52:01.903	Info	Running job of type SayHelloJob.			NULL	NULL
13	Core.Jobs.BaseLemmingJob	2016-03-16 12:52:01.903	Info	Running job of type SayHelloJob.			NULL	NULL
14	Core.Jobs.BaseLemmingJob	2016-03-16 12:52:01.903	Info	Running job of type SayHelloJob.			NULL	NULL
15	Core.Jobs.BaseLemmingJob	2016-03-16 12:52:01.903	Info	Running job of type SayHelloJob.			NULL	NULL
16	Core.Jobs.BaseLemmingJob	2016-03-16 12:52:01.903	Info	Running job of type SayHelloJob.			NULL	NULL

As you start to redesign your applications to take advantage of asynchronous background processing, it is important to understand the importance of logging. In this lab we implemented our logging framework that will be used in future labs.

This completes Lab 03.