

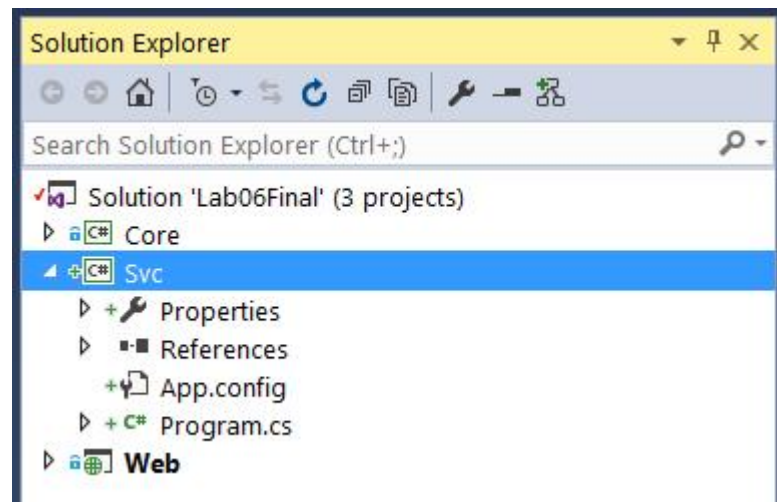
Using Background Processing to Build Scalable Applications with Hangfire

Lab 7

Part 1

In Lab 7 we will remove the Hangfire servers from our Web project and host them inside of a Windows Service. We will use TopShelf, a framework which eases the development of Windows Services.

1. To get started, we need to add a new project to our solution for the Service. In Solution Explorer, right-click on the solution and select Add, New Project.... Create a new Windows Console Application, called Svc.



2. Next, we need to install a number of NuGet Packages. To begin, open the NuGet Package Manager Console (Tools \ NuGet Package Manager \ Package Manager Console).
3. Change the Default project to the Svc project.



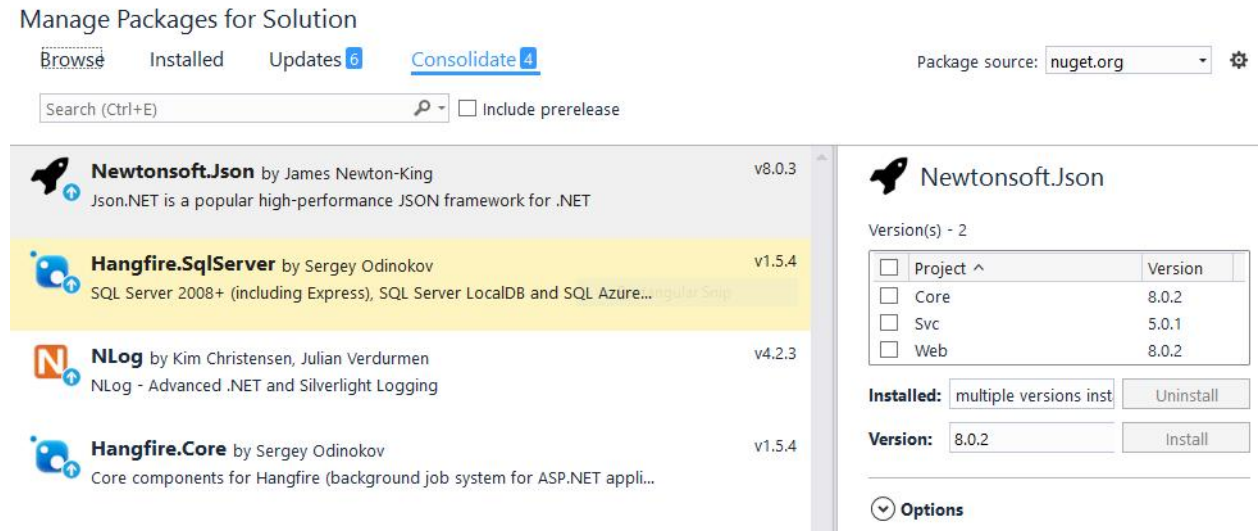
4. To install the necessary packages, execute the following commands at the PM> prompt:

```
PM> Install-Package TopShelf
PM> Install-Package TopShelf.NLog
PM> Install-Package Hangfire.Core
PM> Install-Package Hangfire.SqlServer
PM> Install-Package EntityFramework
```

Note: we are installing the *Hangfire.Core* and *Hangfire.SqlServer* packages separately, and not installing the *Hangfire* package as we have done previously. The *Hangfire* package is meant to

be a quick start, and includes both *Hangfire.Core* and *Hangfire.SqlServer* as well as a dependency on *Microsoft.Owin.Host.SystemWeb* and all of its dependencies. Since we're hosting our Hangfire servers in a Windows Server, none of the web dependencies are necessary.

- Due to some dependency issues, we must now Consolidate our NuGet packages. Consolidating NuGet packages upgrades each project in the solution so that each project uses the same version of the same package. To Consolidate NuGet packages, first open the NuGet extension (Tools \ NuGet Package Manager \ Manage NuGet Packages for Solution...)



- Next, click Consolidate at the top of the page.
- For each package listed, select the package, then select all of the projects, and then click Install.
- This will ensure that each project is using the proper version of the package specified.
- For convenience, we need to move the Configuration folder from the Web project to Core. To do so, simply drag-and-drop the folder in Solution Explorer.
- Next, add a reference to System.Configuration to the Core project.
- Clean up the Bootstrap.cs file by removing any unused using statements, and renaming the namespace from Web.Configuration to Core.Configuration.
- Delete the Configuration folder from the Web project.
- Open the Global.asax.cs file, located in the Web project.
- Cleanup any unused using statements (Web.Configuration).
- Modify the call to our Bootstrap class as follows:

```
Core.Configuration.Bootstrap.Start();
```

- Add a connection string to the App.config file in the Svc project.

```
<connectionStrings>
  <add name="DefaultConnection"
        connectionString="Data Source=(LocalDb)\MSSQLLocalDB;Initial Catalog=LemmingsS
chedulingSystem;Integrated Security=True"
```

```

        providerName="System.Data.SqlClient" />
</connectionStrings>

```

17. Add a reference to the Core project to the Svc project.
18. In the Svc project, add a new class called EventProcessor.cs.
19. Copy the following into the EventProcessor.cs file:

```

using System;
using System.Collections.Generic;
using System.Linq.Expressions;
using Core.Constants;
using Core.Jobs;
using Hangfire;

namespace Svc
{
    public class EventProcessor
    {
        private readonly Stack<BackgroundJobServer> _backgroundJobServers;

        public EventProcessor()
        {
            _backgroundJobServers = new Stack<BackgroundJobServer>();
        }

        public void Start()
        {
            Core.Configuration.Bootstrap.Start();
            GlobalConfiguration.Configuration.UseSqlServerStorage("DefaultConnection")
                .UseNLogLogProvider();

            var serverOptions = new BackgroundJobServerOptions()
            {
                WorkerCount = Environment.ProcessorCount * 5, // This is the amount o
f threads. Default is 10;
                Queues = new[] { QueuePriority.HighPriority, QueuePriority.Default }
            };

            var recurringOptions = new BackgroundJobServerOptions()
            {
                WorkerCount = 1,
                Queues = new[] { QueuePriority.Recurring }
            };

            _backgroundJobServers.Push(new BackgroundJobServer(serverOptions));
            _backgroundJobServers.Push(new BackgroundJobServer(recurringOptions));

            ConfigureRecurringJobs();
        }

        public void Stop()
        {
            while (_backgroundJobServers.Count > 0)

```

```

        _backgroundJobServers.Pop().Dispose();
    }

    private static void ConfigureRecurringJobs()
    {
        SetupRecurringJob<BuildHouseJob>(j => j.BuildHouseRecurring());
    }

    private static void SetupRecurringJob<T>(Expression<Action<T>> job) where T :
BaseLemmingJob
    {
        RecurringJob.AddOrUpdate(job, Cron.Minutely);
    }
}

```

A large majority of the EventProcessor class should look familiar. A handful of new pieces include:

BackgroundJobServers are now managed as part of a stack, initialized in the EventProcessor constructor.

A Start method contains the Hangfire server instantiation and configuration. Hangfire servers are then pushed onto the stack.

A Stop method pops all of the Hangfire servers off of the stack, and then disposed of each instance.

At a cursory glance, the EventProcessor class is simple and may not appear to offer much more than what we have already done. The true purpose of the EventProcessor is when we configure our Windows Service with TopShelf.

20. Open the Program.cs file in the Svc project and copy the following into the file:

```

using Topshelf;

namespace Svc
{
    class Program
    {
        static void Main(string[] args)
        {
            HostFactory.Run(x =>
            {
                x.UseNLog();

                x.SetDisplayName("CPL16-Hangfire Event Processing Host");
                x.SetServiceName("CPL16-HangfireEventProcessor");
                x.SetDescription("CPL16-Hangfire Event Processing Host");
                x.RunAsLocalSystem();
            });
        }
    }
}

```

```

        x.StartAutomatically();
        x.EnableServiceRecovery(r =>
        {
            r.RestartService(1);
            r.SetResetPeriod(1);
        });

        x.Service<EventProcessor>(s =>
        {
            s.ConstructUsing(name => new EventProcessor());

            s.WhenStarted(tc => tc.Start());
            s.WhenStopped(tc => tc.Stop());
        });
    }
}

```

Inside of the Program class, we are running the HostFactory, which is used by TopShelf to describe how the service should be installed and what actions are performed when it is started and stopped.

Notice that when our service is started, we call EventProcessor.Start(). When it is stopped, we call EventProcessor.Stop().

21. Before we try to launch both of our applications (Web and Svc), we need to perform some cleanup in the Web project.
22. Open the Startup.Hangfire.cs located in the App_Start folder in the Web project.
23. Copy the following into the Startup.Hangfire.cs file:

```

using Hangfire;
using Owin;

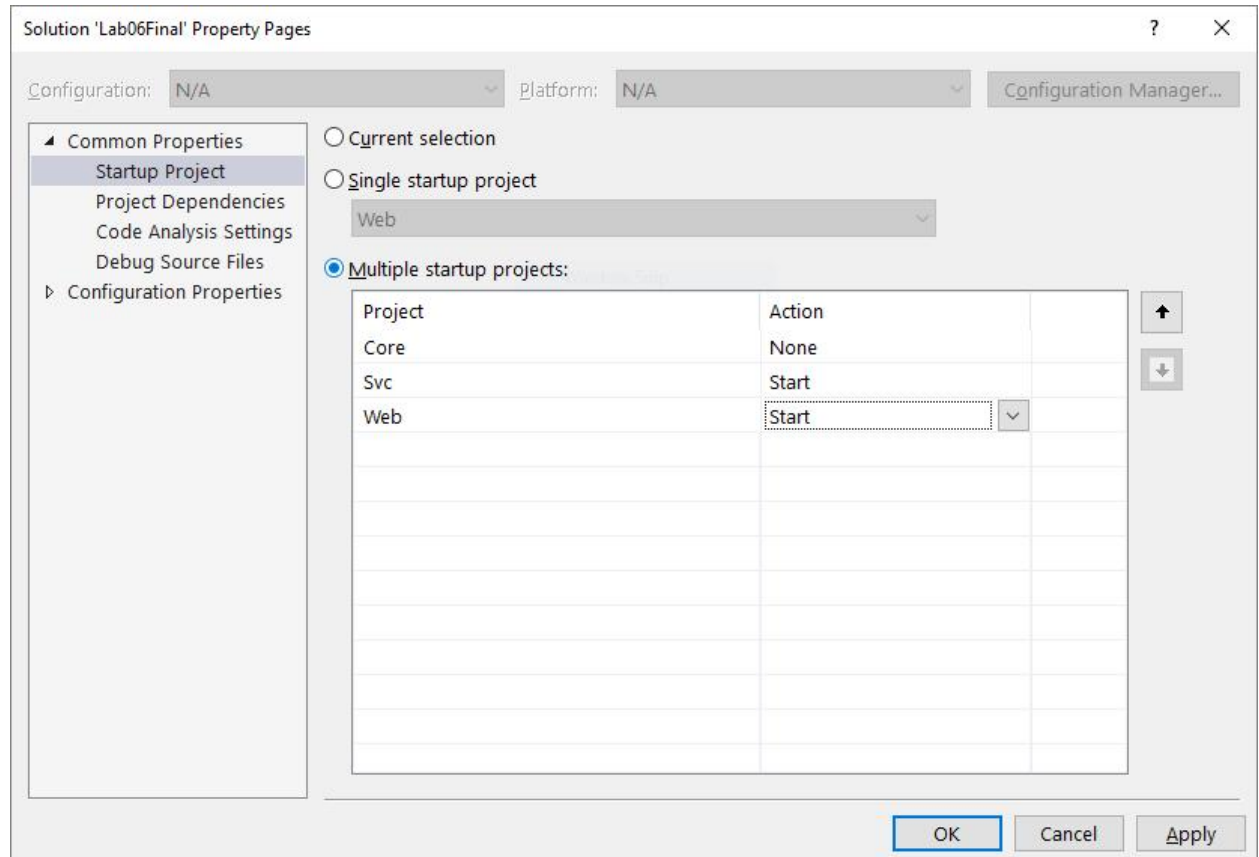
namespace Web
{
    public partial class Startup
    {
        public void ConfigureHangfire(IAppBuilder app)
        {
            GlobalConfiguration.Configuration.UseSqlServerStorage("DefaultConnection")
            .UseNLogLogProvider();

            app.UseHangfireDashboard(); // This will configure the current application
            // to run the Hangfire Dashboard with the configured connection.
        }
    }
}

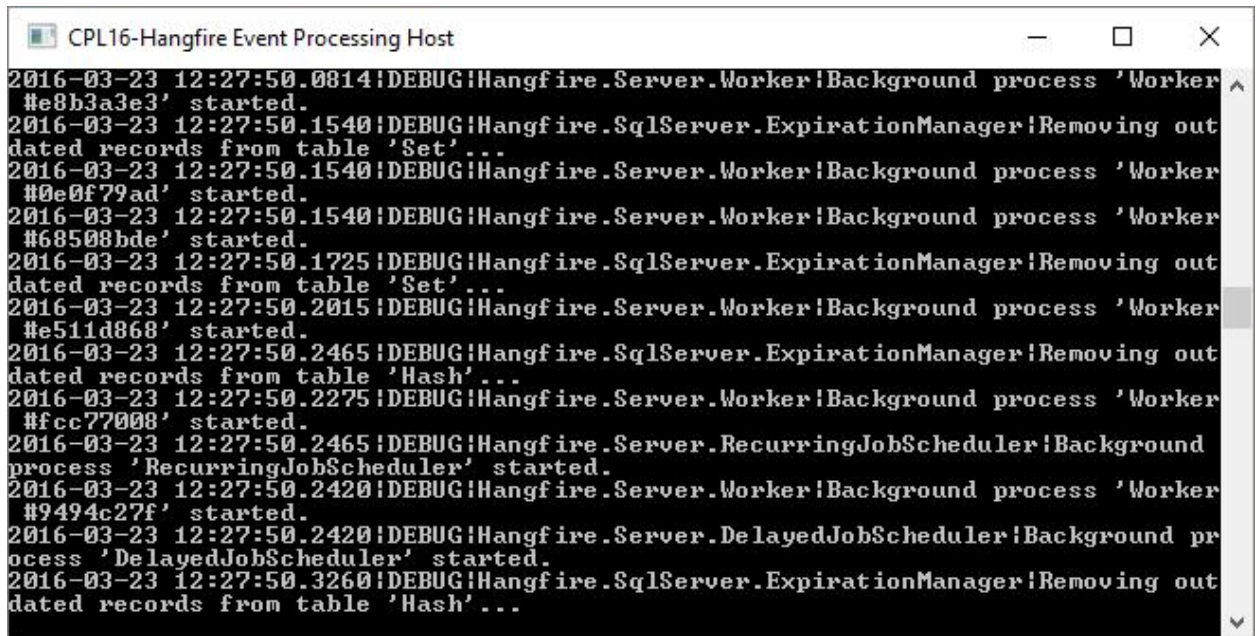
```

Notice that we've removed the majority of the contents of the Startup.Hangfire.cs file. We simply have to configure our persistent storage, and the dashboard. All of our Hangfire servers are now located in the Svc project.

24. Finally, we need to set our Startup Projects.
25. In Solution Explorer, right-click the solution, and select Set StartUp Projects...
26. Select Multiple startup projects, and then set the Action of both the Svc and Web projects to Start.

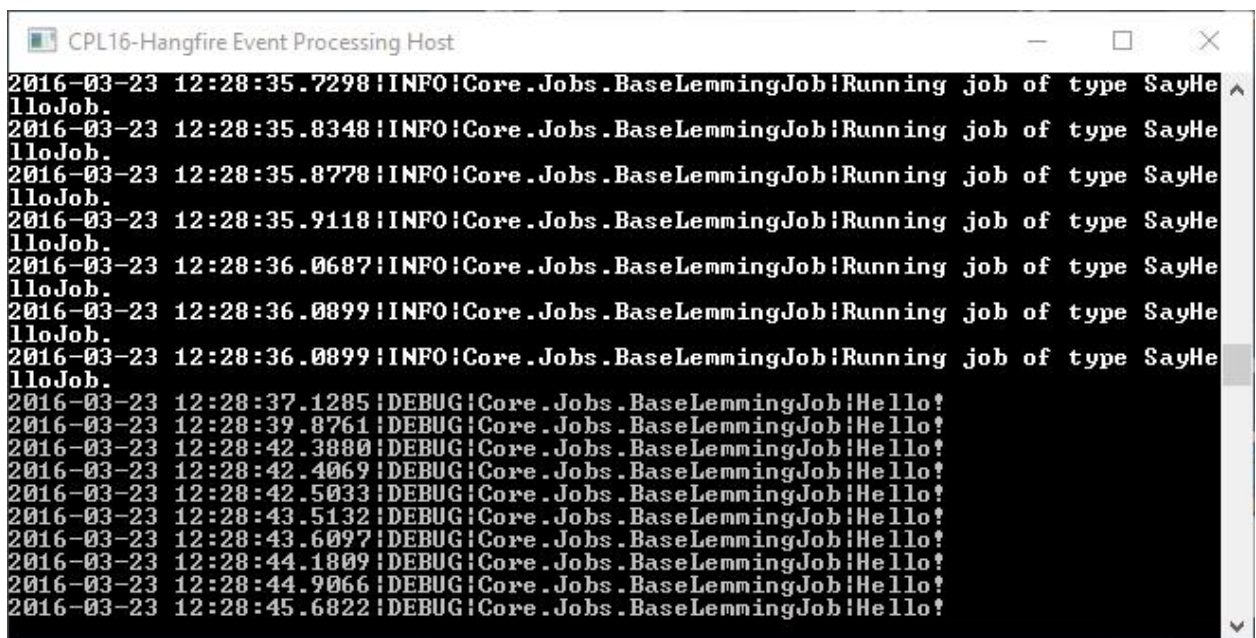


27. Click OK.
28. Now press F5 to start debugging. You should see both your web application and a console application running.
29. As the Hangfire servers are initialized, you will see a large amount of debugging messages written to the console.



```
2016-03-23 12:27:50.0814|DEBUG|Hangfire.Server.Worker!Background process 'Worker
#e8b3a3e3' started.
2016-03-23 12:27:50.1540|DEBUG|Hangfire.SqlServer.ExpirationManager!Removing out
dated records from table 'Set'...
2016-03-23 12:27:50.1540|DEBUG|Hangfire.Server.Worker!Background process 'Worker
#0e0f79ad' started.
2016-03-23 12:27:50.1540|DEBUG|Hangfire.Server.Worker!Background process 'Worker
#68508bde' started.
2016-03-23 12:27:50.1725|DEBUG|Hangfire.SqlServer.ExpirationManager!Removing out
dated records from table 'Set'...
2016-03-23 12:27:50.2015|DEBUG|Hangfire.Server.Worker!Background process 'Worker
#e511d868' started.
2016-03-23 12:27:50.2465|DEBUG|Hangfire.SqlServer.ExpirationManager!Removing out
dated records from table 'Hash'...
2016-03-23 12:27:50.2275|DEBUG|Hangfire.Server.Worker!Background process 'Worker
#fcc77008' started.
2016-03-23 12:27:50.2465|DEBUG|Hangfire.Server.RecurringJobScheduler!Background
process 'RecurringJobScheduler' started.
2016-03-23 12:27:50.2420|DEBUG|Hangfire.Server.Worker!Background process 'Worker
#9494c27f' started.
2016-03-23 12:27:50.2420|DEBUG|Hangfire.Server.DelayedJobScheduler!Background pr
ocess 'DelayedJobScheduler' started.
2016-03-23 12:27:50.3260|DEBUG|Hangfire.SqlServer.ExpirationManager!Removing out
dated records from table 'Hash'...
```

30. In addition to the CPL16-Hangfire Event Processing Host, you should also see a browser running with your application. Queue a number of Lemming jobs from the Dashboard. Below you see that a number of Hello Lemmings were queued.



```
2016-03-23 12:28:35.7298|INFO|Core.Jobs.BaseLemmingJob!Running job of type SayHe
lloJob.
2016-03-23 12:28:35.8348|INFO|Core.Jobs.BaseLemmingJob!Running job of type SayHe
lloJob.
2016-03-23 12:28:35.8778|INFO|Core.Jobs.BaseLemmingJob!Running job of type SayHe
lloJob.
2016-03-23 12:28:35.9118|INFO|Core.Jobs.BaseLemmingJob!Running job of type SayHe
lloJob.
2016-03-23 12:28:36.0687|INFO|Core.Jobs.BaseLemmingJob!Running job of type SayHe
lloJob.
2016-03-23 12:28:36.0899|INFO|Core.Jobs.BaseLemmingJob!Running job of type SayHe
lloJob.
2016-03-23 12:28:36.0899|INFO|Core.Jobs.BaseLemmingJob!Running job of type SayHe
lloJob.
2016-03-23 12:28:37.1285|DEBUG|Core.Jobs.BaseLemmingJob!Hello!
2016-03-23 12:28:39.8761|DEBUG|Core.Jobs.BaseLemmingJob!Hello!
2016-03-23 12:28:42.3880|DEBUG|Core.Jobs.BaseLemmingJob!Hello!
2016-03-23 12:28:42.4069|DEBUG|Core.Jobs.BaseLemmingJob!Hello!
2016-03-23 12:28:42.5033|DEBUG|Core.Jobs.BaseLemmingJob!Hello!
2016-03-23 12:28:43.5132|DEBUG|Core.Jobs.BaseLemmingJob!Hello!
2016-03-23 12:28:43.6097|DEBUG|Core.Jobs.BaseLemmingJob!Hello!
2016-03-23 12:28:44.1809|DEBUG|Core.Jobs.BaseLemmingJob!Hello!
2016-03-23 12:28:44.9066|DEBUG|Core.Jobs.BaseLemmingJob!Hello!
2016-03-23 12:28:45.6822|DEBUG|Core.Jobs.BaseLemmingJob!Hello!
```

In this part of the lab, we have moved our Hangfire servers into a separate process. To install your service, or learn more about TopShelf, visit <http://topshelf-project.com/>. Note: it is not necessary to install your service in order to continue.

Part 2

What happens when a request is made to stop a Hangfire server while a worker thread is in the middle of executing a long running job? In this part of the lab we will find out how Hangfire handles shutting down when a job is in the middle of execution, and we will also learn about *Cancellation Tokens*, which are used to signal a worker thread that it is time to shutdown.

1. To begin, open BaseLemmingJob.cs located in the Jobs folder, in the Core project.
2. Change the SleepyTime method from private to protected:

```
protected void SleepyTime()
{
    var random = new Random();

    Thread.Sleep(random.Next(1000, 10000));
}
```

3. Next, open the BuildHouseJob.cs file located in the Jobs folder in the Core project.
4. Update the BuildHouseRecurring method so that it looks like the following:

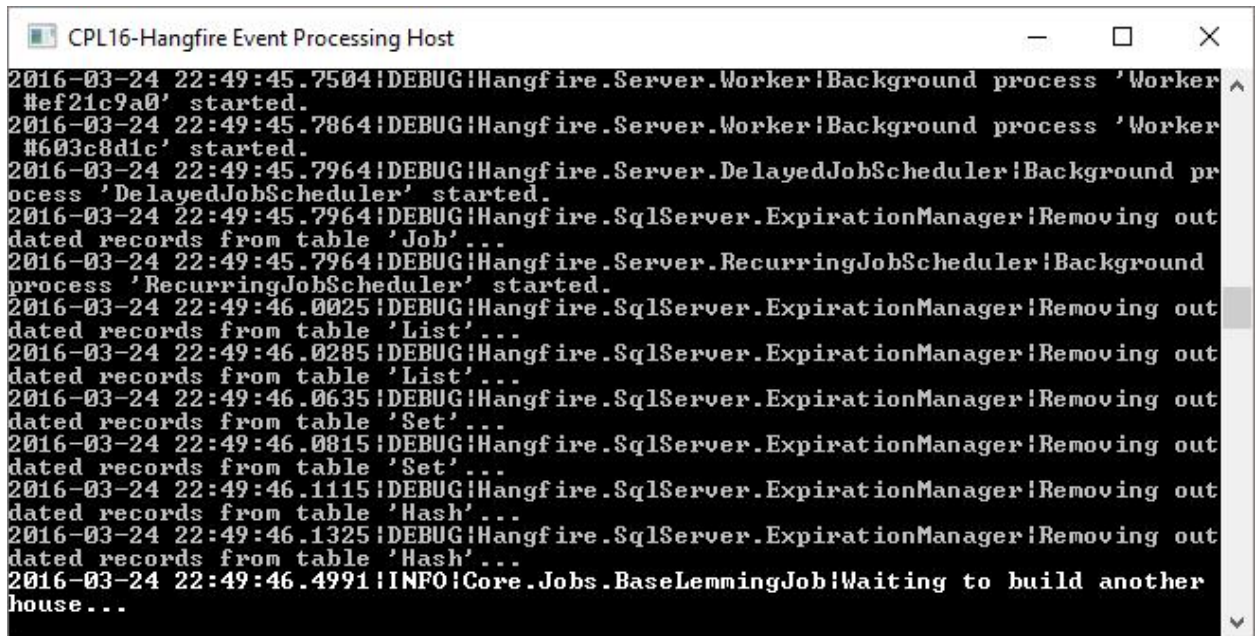
```
[Queue(QueuePriority.Recurring)]
public void BuildHouseRecurring()
{
    Logger.Info($"Waiting to build another house...");

    for (var i = 0; i < 10; i++)
    {
        SleepyTime();
    }

    DoWork();
}
```

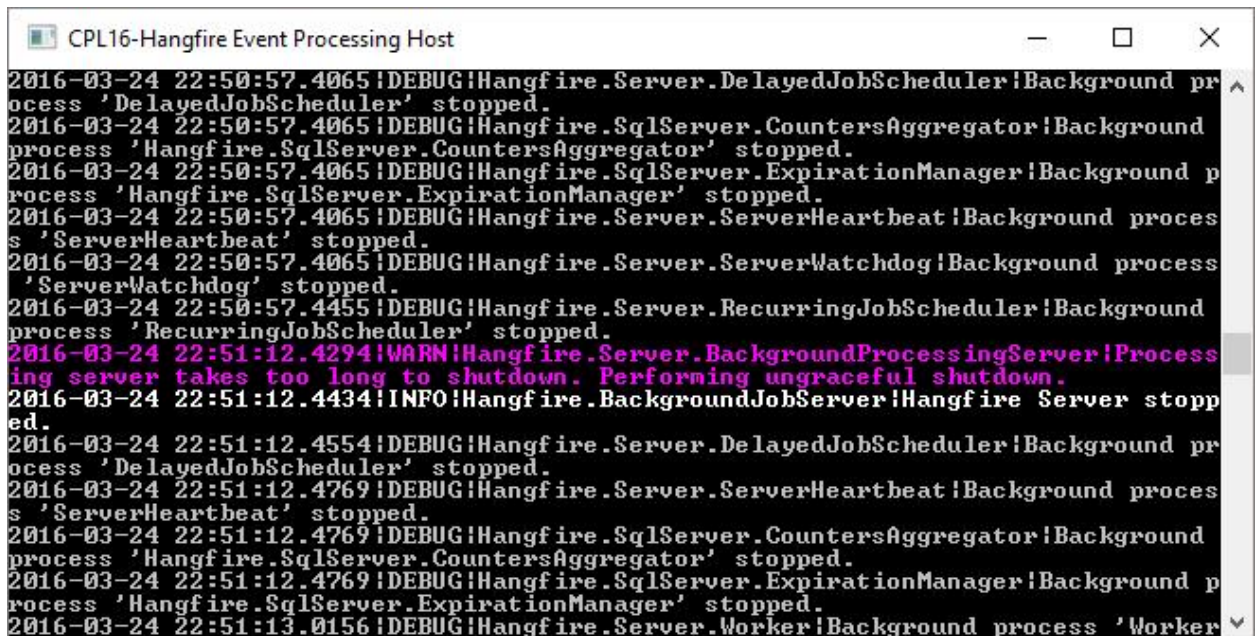
Here we have updated the BuildHouseRecurring method so that it takes a bit longer to execute.

5. Open the EventProcessor.cs file in the Svc project.
6. Place a breakpoint on the closing brace ('}') of the Stop() method. Our goal here is to stop at a breakpoint after our service has completed its shutdown procedure.
7. Press F5 to run both the service and your application.
8. In the console window, when you see the message "Waiting to build another house...", press CTRL+C to stop the service.



```
CPL16-Hangfire Event Processing Host
2016-03-24 22:49:45.7504!DEBUG!Hangfire.Server.Worker!Background process 'Worker
#ef21c9a0' started.
2016-03-24 22:49:45.7864!DEBUG!Hangfire.Server.Worker!Background process 'Worker
#603c8d1c' started.
2016-03-24 22:49:45.7964!DEBUG!Hangfire.Server.DelayedJobScheduler!Background pr
ocess 'DelayedJobScheduler' started.
2016-03-24 22:49:45.7964!DEBUG!Hangfire.SqlServer.ExpirationManager!Removing out
dated records from table 'Job'...
2016-03-24 22:49:45.7964!DEBUG!Hangfire.Server.RecurringJobScheduler!Background
process 'RecurringJobScheduler' started.
2016-03-24 22:49:46.0025!DEBUG!Hangfire.SqlServer.ExpirationManager!Removing out
dated records from table 'List'...
2016-03-24 22:49:46.0285!DEBUG!Hangfire.SqlServer.ExpirationManager!Removing out
dated records from table 'List'...
2016-03-24 22:49:46.0635!DEBUG!Hangfire.SqlServer.ExpirationManager!Removing out
dated records from table 'Set'...
2016-03-24 22:49:46.0815!DEBUG!Hangfire.SqlServer.ExpirationManager!Removing out
dated records from table 'Set'...
2016-03-24 22:49:46.1115!DEBUG!Hangfire.SqlServer.ExpirationManager!Removing out
dated records from table 'Hash'...
2016-03-24 22:49:46.1325!DEBUG!Hangfire.SqlServer.ExpirationManager!Removing out
dated records from table 'Hash'...
2016-03-24 22:49:46.4991!INFO!Core.Jobs.BaseLemmingJob!Waiting to build another
house...
```

9. Once your breakpoint gets hit, return to the console window, and scroll up until you see the magenta message.



```
CPL16-Hangfire Event Processing Host
2016-03-24 22:50:57.4065!DEBUG!Hangfire.Server.DelayedJobScheduler!Background pr
ocess 'DelayedJobScheduler' stopped.
2016-03-24 22:50:57.4065!DEBUG!Hangfire.SqlServer.CountersAggregator!Background
process 'Hangfire.SqlServer.CountersAggregator' stopped.
2016-03-24 22:50:57.4065!DEBUG!Hangfire.SqlServer.ExpirationManager!Background p
rocess 'Hangfire.SqlServer.ExpirationManager' stopped.
2016-03-24 22:50:57.4065!DEBUG!Hangfire.Server.ServerHeartbeat!Background proces
s 'ServerHeartbeat' stopped.
2016-03-24 22:50:57.4065!DEBUG!Hangfire.Server.ServerWatchdog!Background process
'ServerWatchdog' stopped.
2016-03-24 22:50:57.4455!DEBUG!Hangfire.Server.RecurringJobScheduler!Background
process 'RecurringJobScheduler' stopped.
2016-03-24 22:51:12.4294!WARN!Hangfire.Server.BackgroundProcessingServer!Process
ing server takes too long to shutdown. Performing ungraceful shutdown.
2016-03-24 22:51:12.4434!INFO!Hangfire.BackgroundJobServer!Hangfire Server stopp
ed.
2016-03-24 22:51:12.4554!DEBUG!Hangfire.Server.DelayedJobScheduler!Background pr
ocess 'DelayedJobScheduler' stopped.
2016-03-24 22:51:12.4769!DEBUG!Hangfire.Server.ServerHeartbeat!Background proces
s 'ServerHeartbeat' stopped.
2016-03-24 22:51:12.4769!DEBUG!Hangfire.SqlServer.CountersAggregator!Background
process 'Hangfire.SqlServer.CountersAggregator' stopped.
2016-03-24 22:51:12.4769!DEBUG!Hangfire.SqlServer.ExpirationManager!Background p
rocess 'Hangfire.SqlServer.ExpirationManager' stopped.
2016-03-24 22:51:13.0156!DEBUG!Hangfire.Server.Worker!Background process 'Worker
```

The message reads *"Processing server takes too long to shutdown. Performing ungraceful shutdown."*. This message tells us that a job was executing when the server shutdown and didn't respond when signal to stop. This job will be requeued once the server is restarted, and executed again.

Additionally, we would prefer that our Hangfire server respond in a timely manner when requested to shutdown, rather than shutting down ungracefully.

10. In Visual Studio, press F5 to continue past the breakpoint and allow our service to stop.
11. Next, stop debugging in Visual Studio.
12. Return to BuildHouseJob.cs and update the BuildHouseRecurring method as below:

```
[Queue(QueuePriority.Recurring)]
public void BuildHouseRecurring(IJobCancellationToken cancellationToken)
{
    Logger.Info($"Waiting to build another house...");

    for (var i = 0; i < 10; i++)
    {
        cancellationToken.ThrowIfCancellationRequested();

        SleepyTime();
    }

    DoWork();
}
```

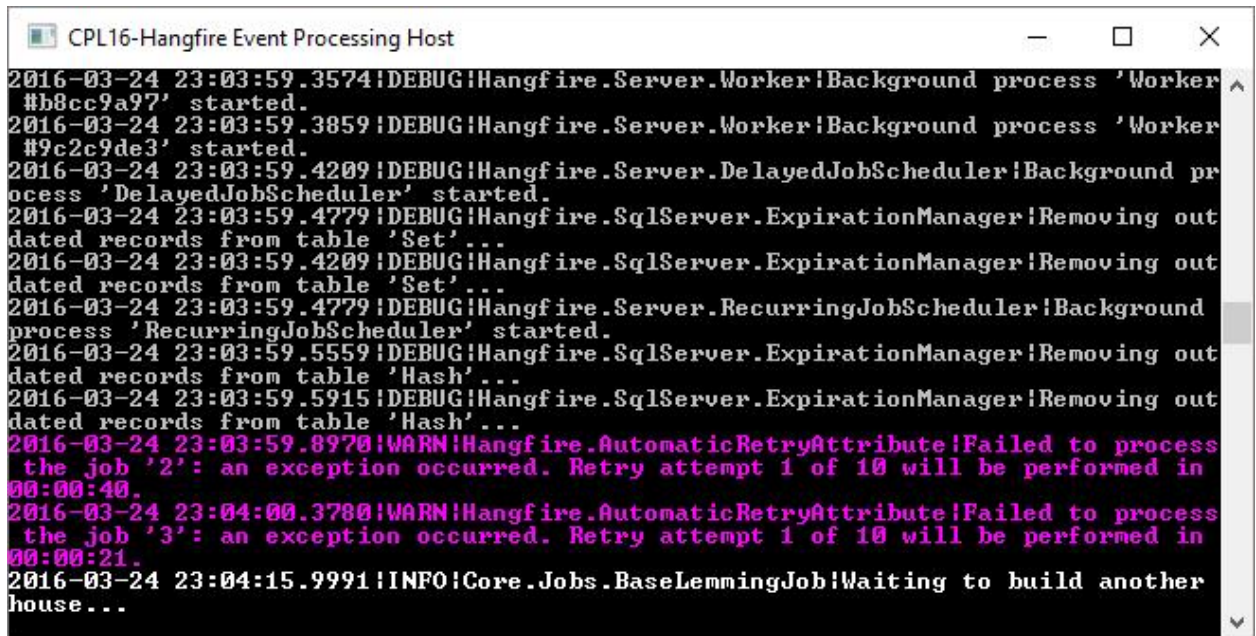
Note: we are knowingly violating a best practice in this step by updating the method signature of our BuildHouseRecurring job. While not ideal in a true application, it does not affect this lab.

13. Open EventProcessor.cs and add the IJobCancellationToken parameter to the call to BuildHouseRecurring:

```
private static void ConfigureRecurringJobs()
{
    SetupRecurringJob<BuildHouseJob>(j => j.BuildHouseRecurring(JobCancellationToken.
Null));
}
```

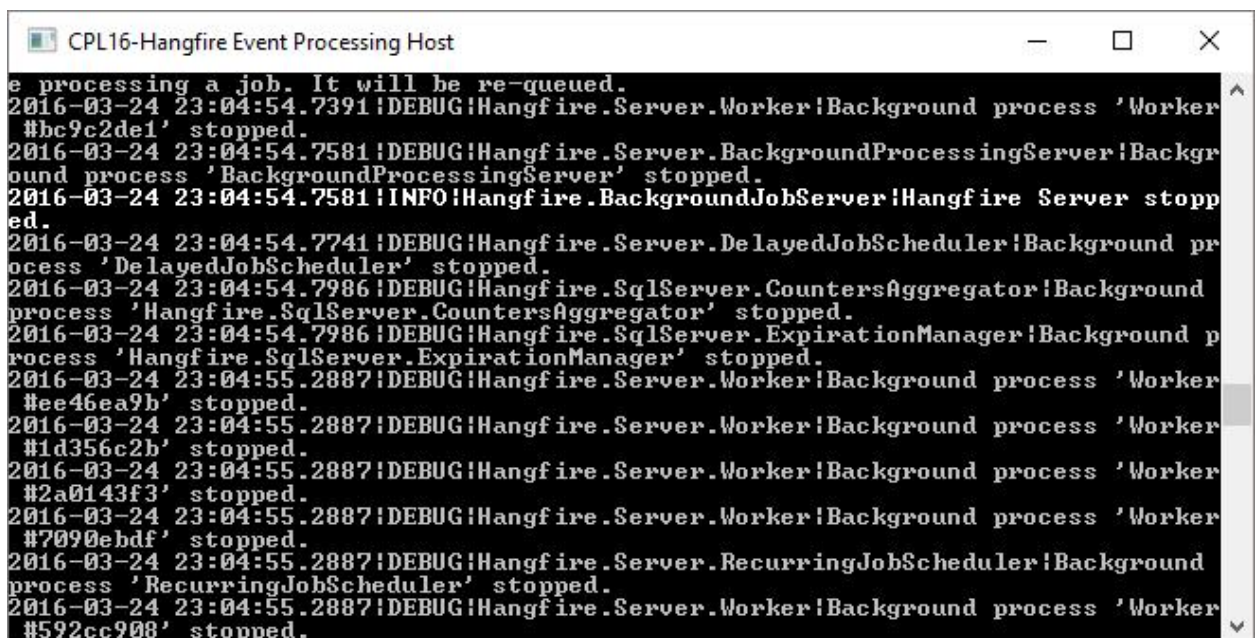
The JobCancellationToken.Null property is simply just a Special Case object which indicates that we are intending to use a job cancellation token. Hangfire will properly pass along a non-null instance of an actual cancellation token during execution at runtime.

14. Press F5 to run both the service and your application.
15. Notice that previous jobs which were ungracefully stopped attempt to be executed again, however fail since we changed our method signature:



```
CPL16-Hangfire Event Processing Host
2016-03-24 23:03:59.3574:DEBUG!Hangfire.Server.Worker!Background process 'Worker
#b8cc9a97' started.
2016-03-24 23:03:59.3859:DEBUG!Hangfire.Server.Worker!Background process 'Worker
#9c2c9de3' started.
2016-03-24 23:03:59.4209:DEBUG!Hangfire.Server.DelayedJobScheduler!Background pr
ocess 'DelayedJobScheduler' started.
2016-03-24 23:03:59.4779:DEBUG!Hangfire.SqlServer.ExpirationManager!Removing out
dated records from table 'Set'...
2016-03-24 23:03:59.4209:DEBUG!Hangfire.SqlServer.ExpirationManager!Removing out
dated records from table 'Set'...
2016-03-24 23:03:59.4779:DEBUG!Hangfire.Server.RecurringJobScheduler!Background
process 'RecurringJobScheduler' started.
2016-03-24 23:03:59.5559:DEBUG!Hangfire.SqlServer.ExpirationManager!Removing out
dated records from table 'Hash'...
2016-03-24 23:03:59.5915:DEBUG!Hangfire.SqlServer.ExpirationManager!Removing out
dated records from table 'Hash'...
2016-03-24 23:03:59.8970:WARN!HangfireAutomaticRetryAttribute!Failed to process
the job '2': an exception occurred. Retry attempt 1 of 10 will be performed in
00:00:40.
2016-03-24 23:04:00.3780:WARN!HangfireAutomaticRetryAttribute!Failed to process
the job '3': an exception occurred. Retry attempt 1 of 10 will be performed in
00:00:21.
2016-03-24 23:04:15.9991:INFO!Core.Jobs.BaseLemmingJob!Waiting to build another
house...
```

16. Once you see the “Waiting to build another house...” message, press CTRL+C.
17. When your breakpoint is hit, return to the console application and scroll through the log messages to find that the Hangfire server did indeed shutdown properly.



```
CPL16-Hangfire Event Processing Host
e processing a job. It will be re-queued.
2016-03-24 23:04:54.7391:DEBUG!Hangfire.Server.Worker!Background process 'Worker
#bc9c2de1' stopped.
2016-03-24 23:04:54.7581:DEBUG!Hangfire.Server.BackgroundProcessingServer!Backgr
ound process 'BackgroundProcessingServer' stopped.
2016-03-24 23:04:54.7581:INFO!Hangfire.BackgroundJobServer!Hangfire Server stopp
ed.
2016-03-24 23:04:54.7741:DEBUG!Hangfire.Server.DelayedJobScheduler!Background pr
ocess 'DelayedJobScheduler' stopped.
2016-03-24 23:04:54.7986:DEBUG!Hangfire.SqlServer.CountersAggregator!Background
process 'Hangfire.SqlServer.CountersAggregator' stopped.
2016-03-24 23:04:54.7986:DEBUG!Hangfire.SqlServer.ExpirationManager!Background p
rocess 'Hangfire.SqlServer.ExpirationManager' stopped.
2016-03-24 23:04:55.2887:DEBUG!Hangfire.Server.Worker!Background process 'Worker
#ee46ea9b' stopped.
2016-03-24 23:04:55.2887:DEBUG!Hangfire.Server.Worker!Background process 'Worker
#1d356c2b' stopped.
2016-03-24 23:04:55.2887:DEBUG!Hangfire.Server.Worker!Background process 'Worker
#2a0143f3' stopped.
2016-03-24 23:04:55.2887:DEBUG!Hangfire.Server.Worker!Background process 'Worker
#7090ebdf' stopped.
2016-03-24 23:04:55.2887:DEBUG!Hangfire.Server.RecurringJobScheduler!Background
process 'RecurringJobScheduler' stopped.
2016-03-24 23:04:55.2887:DEBUG!Hangfire.Server.Worker!Background process 'Worker
#592cc908' stopped.
```

18. Next, open a new tab in your browser and navigate to the Hangfire dashboard.
19. Navigate to the list of Processing Jobs (<http://localhost:13161/hangfire/jobs/processing>)
20. You should find a BuildHouseJob.BuildHouseRecurring job with a small warning icon:

Processing Jobs

🔄 Requeue jobs

✖ Delete selected

Items per page:

102050100500

<input type="checkbox"/>	Id	Server	Job	Started
<input type="checkbox"/>	#4	BUCKINDB-NB:15768:F0F072BE-4605-4EB8-9053-E39C904D29EC	<div>⚠ BuildHouseJob.BuildHouseRecurring</div>	4 minutes ago

The icon indicates that the job may have been aborted.

21. Click on the job Id to browse to the job details.
22. Here we see that the job was indeed aborted:

BuildHouseJob.BuildHouseRecurring

The job was aborted – it is processed by server `buckindb-nb:15768:f0f072be-4605-4eb8-9053-e39c904d29ec` which is not in the [active servers](#) list for now. It will be retried automatically after invisibility timeout, but you can also re-queue or delete it manually.

Job ID: #4

[Requeue](#) [Delete](#)

```
using Core.Jobs;

BuildHouseJob buildHouseJob = Activate<BuildHouseJob>();
buildHouseJob.BuildHouseRecurring(null);
```

Created 6 minutes ago

CurrentCulture
"en-US"

CurrentUICulture
"en-US"

In this lab we successfully offloaded our Hangfire servers from our web application, and into a separate hosting process which can be freely installed on a separate machine with it's own resources. Additionally, we also explored the use of Cancellation Tokens to assist in shutting down when long running tasks are currently executing.

This completes Lab 07.