Using Background Processing to Build Scalable Applications with Hangfire

Lab 6

In Lab 6 we will explore the many ways in which Hangfire can be configured to scale to accommodate workloads of all sizes.

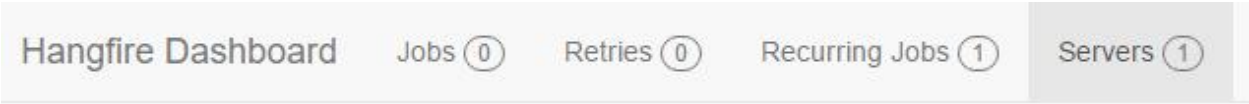
Part 1

In Part 1 of this lab, we will explore how Hangfire can be configured to run multiple queues. Additionally, we will configure our background jobs to be enqueued in specific queues, when multiple queues exist.

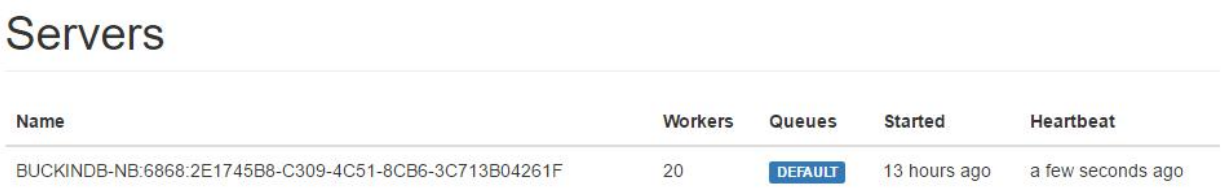
1. Before we configure additional queues, let's first take a look at the default configuration of our Hangfire server. From inside of Visual Studio, press F5 to debug your application.
2. Once you application is running, open a new tab and browse to the following URL to open the Hangfire dashboard:

   http://localhost:13161/hangfire

3. Once the Hangfire dashboard is loaded, click Servers from the navigation bar at the top of the dashboard.



4. In the list of Servers, you should see your server listed (your name will be different). In the Queues column, you should see a single queue called Default.



   By default, Hangfire is configured to use a single queue, simply called Default.

5. Return to Visual Studio and stop debugging.
6. Before we configure more queues for our Hangfire server, let's first create some constants to help. Add a folder to the Core project called Constants.
7. In the Constants folder, add a class called QueuePriority and paste the following code into the file:

```
namespace Core.Constants
{
    public static class QueuePriority
    {
        public const string Default = "default";
        public const string HighPriority = "aahighpriority";
        public const string Recurring = "recurring";
    }
}
```

8. Open the Startup.Hangfire.cs file located in the App_Start folder in the Web project.
9. Add the following using statement to the top of the file:

```
using Core.Constants;
```

10. Next, paste the following code after the call to the GlobalConfiguration.

```
var serverOptions = new BackgroundJobServerOptions()
{
    Queues = new[] {QueuePriority.HighPriority, QueuePriority.Default}
};
```

In this snippet, we are instantiating an instance of the BackgroundJobServerOptions class which is used to configure a Hangfire server. Using the BackgroundJobServerOptions, we are adding two queues: a High Priority queue and the Default Queue. Note: there is nothing special about our High Priority queue. It's treated as "high priority" by name alone, and we decide what jobs are treated as high priority.

11. Next, add serverOptions as a parameter to the call to UseHangfireServer.

```
app.UseHangfireServer(serverOptions);  // This will configure the current application
 to run the Hangfire Server with the configured connection.
```

12. Run your application again by pressing F5.
13. Browse to the Hangfire dashboard, and navigate to the list of Servers.
14. You should now see two queues listed for your server: AAHIGHPRIORITY and DEFAULT.

15. Return to Visual Studio, and stop debugging.
16. To take advantage of our new queue, we will configure the BuildHouseJob.BuildHouseWithFriendWithName method to use the high priority queue.
17. First, we have to install the Hangfire NuGet package into the Core project.
18. Open the NuGet Package Manager Console (Tools \ NuGet Package Manager \ Package Manager Console).
19. Type the following command at the PM> prompt:

```
PM> Install-Package Hangfire –ProjectName Core
```

20. Open the BuildHouseJob.cs file in the Jobs folder, in the Core project.
21. Next, add the following using statements to the top of the file:

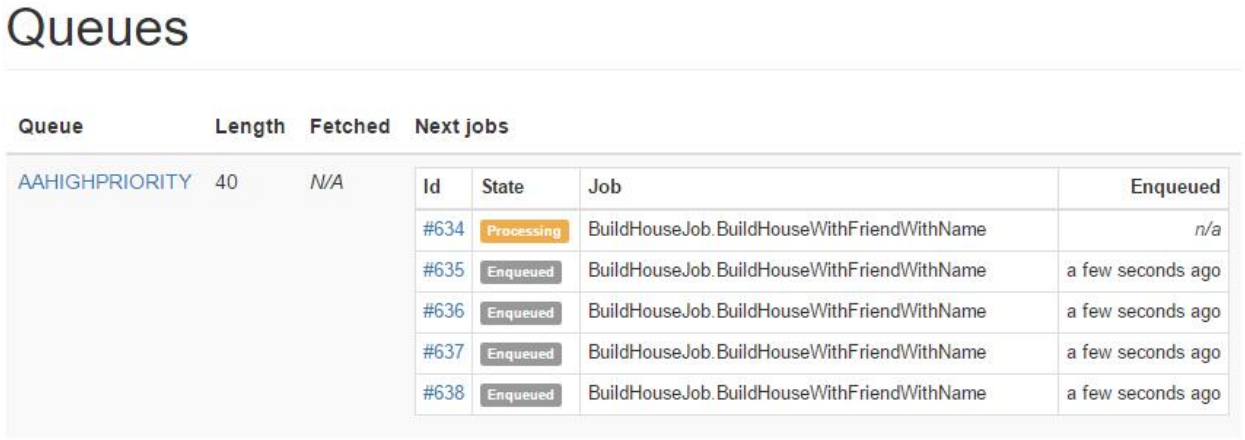
```
using Hangfire;
using Core.Data;
```

22. Next, decorate the BuildHouseWithFriendWithName method with the following attribute:

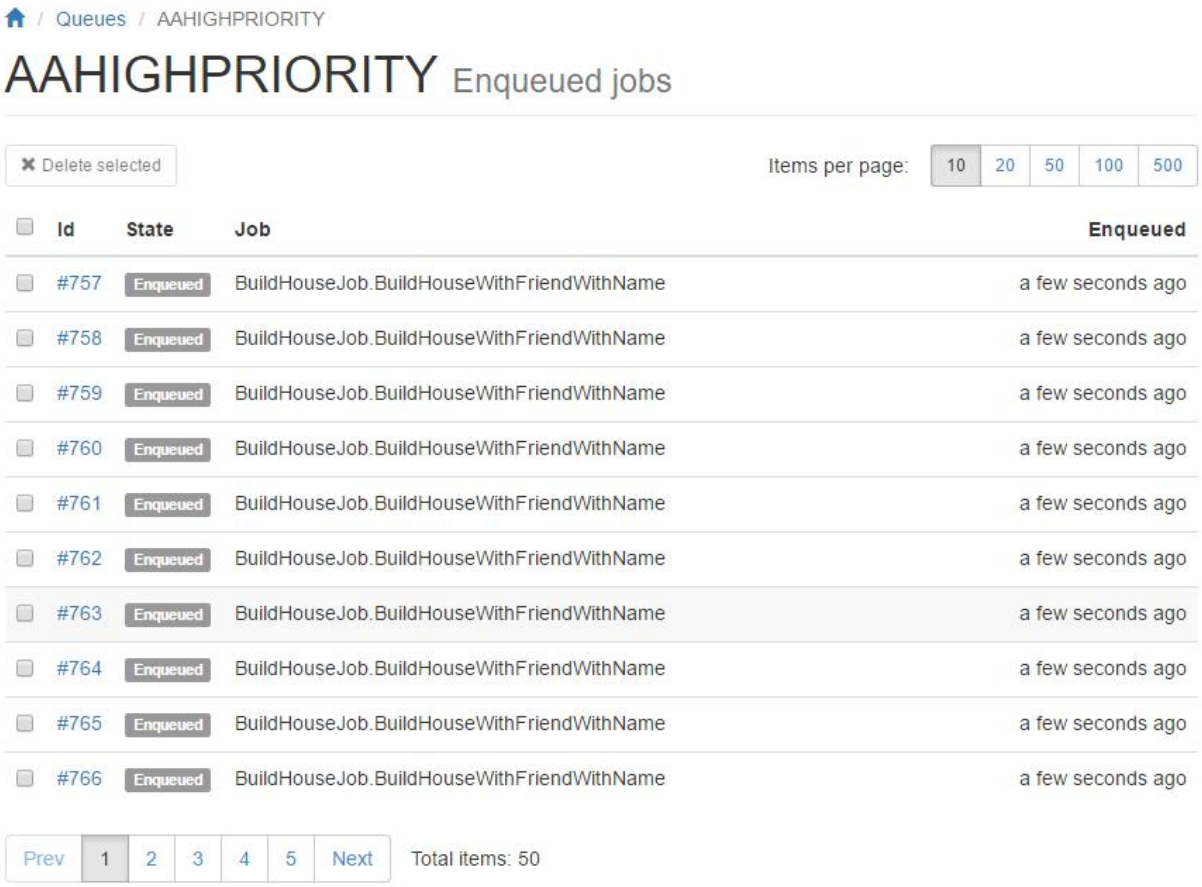
```
[Queue(QueuePriority.HighPriority)]
```

The Queue attribute instructs Hangfire in which queue the job should be placed whenever it is enqueued.

23. Press F5 to run your application.
24. Once the application is running, request a large number (500 or so) of Home Builder Lemmings.
25. Open a new tab in your browser, and browse the to the Hangfire Dashboard.
26. Once the Hangfire Dashbord is loaded, select Jobs from the Navigation Bar.
27. Here you'll see the list of all Queues which contain enqueued jobs.



28. Next, click the *AAHIGHPRIORITY* queue name.
29. Now you can see the details for each job currently in the *AAHIGHPRIORITY* queue.



In this part of the lab we have explored how Hangfire can be configured to support multiple queues inside of a single Hangfire server. With multiple queues, we have seen that we can partition our jobs so that particular jobs are only enqueued in the proper queues.

Part 2

In Part 2 of this lab, we will configure our Hangfire server to run specified number of Worker Threads. Hangfire uses this pool of dedicated threads for servicing the queue, or queues.

1. Before we configure additional worker threads, let's first take a look at our Hangfire server and find out how many threads it currently uses. If your application is not running, from Visual Studio press F5 to start your application.
2. Next, open a tab in your browser and browse to the Hangfire dashboard.
3. Once the Hangfire dashboard is loaded, click Servers in the navigation bar at the top of the page.
4. In the list of Servers, you should see your server listed (your name will be different). In the Workers column, you will see a number indicating the number of worker threads currently running in your Hangfire server.

Note: you may see a different value of worker threads for your Hangfire server. By default, Hangfire servers are configured to use

```
WorkerCount = Environment.ProcessorCount * 5
```

Now let's change this value to better suit our needs.

5. Return to Visual Studio and stop debugging.
6. Open the Startup.Hangfire.cs file located in the App_Start folder in the Web project.
7. Inside of the BackgroundJobServerOptions, configure the WorkerCount property and set the number of worker threads to some specific value.

```
var serverOptions = new BackgroundJobServerOptions()
{
    WorkerCount = 10,
    Queues = new[] { QueuePriority.HighPriority, QueuePriority.Default }
};
```

Here we are changing the number of worker threads from the default (20, in my case) to 10.

8. Run your application again by pressing F5.
9. Browse to the Hangfire dashboard, and navigate to the list of Servers.
10. You should now see a new value (reflective of the value that you specified in the BackgroundJobServerOptions) for the number of Workers.

In this part of the lab, we have explored how the number of Worker Threads inside of the Hangfire server can be configured. The number of Worker Threads can determine how quickly our enqueued jobs are serviced. If jobs take a large amount of time to execute, adding more Worker Threads can help service the Queues in a more expedient manner, at the expense of using more resources (CPU and RAM).

Part 3

In the final part of this lab, we will configure a second Hangfire server to run in our application and assign specific queues to this new instance of Hangfire.

1. To configure a new Hangfire server, open the Startup.Hangfire.cs file located in the App_Start folder in the Web project.

2. Add the following instance of the BackgroundJobServerOptions immediately following our declaration of the serverOptions variable:

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

In this snippet, we are creating a new instance of the BackgroundJobServerOptions class to be used to configure our new Hangfire server instance. This instance will be configured to have a single worker thread, and will only have a single queue to be serviced.

3. Next, add the following line immediately after the call to app.UseHangfireServer(serverOptions):

```
app.UseHangfireServer(recurringOptions);
```

In this snippet, we have added a second Hangfire server, configured with the options that we declared previously.

4. At this point, the ConfigureHangfire method should look like the following:

```
public void ConfigureHangfire(IAppBuilder app)
{
    GlobalConfiguration.Configuration.UseSqlServerStorage("DefaultConnection").UseNLo
gLogProvider();

    var serverOptions = new BackgroundJobServerOptions()
    {
        Queues = new[] {QueuePriority.HighPriority, QueuePriority.Default}
    };

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

    app.UseHangfireDashboard(); // This will configure the current application to run
 the Hangfire Dashboard with the configured connection.
    app.UseHangfireServer(serverOptions);  // This will configure the current applica
tion to run the Hangfire Server with the configured connection.
    app.UseHangfireServer(recurringOptions);

    ConfigureRecurringJobs();
}
```

5. Next, open the BuildHouseJob.cs file located in the Jobs folder in the Core project.
6. Add the following method to the BuildHouseJob class:

```
[Queue(QueuePriority.Recurring)]
public void BuildHouseRecurring()
{
    DoWork();
}
```

This method simply wraps our DoWork() method, and decorates the method with the Queue attribute, specifying the Recurring queue. Note: the Recurring property was previously created when we created the QueuePriority class.

7. Next, we need to update the recurring job registration. In the Startup.Hangfire.cs file, modify the ConfigureRecurringJobs method as follows:

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

Here we simply changed the method to be called from Run() to BuildHouseRecurring(). This is necessary since our Run() method is declared in our BaseLemmingJob class and we now wish to assign this specific job to a specific queue.

8. Press F5 to start the application.
9. Once the application is running, open a new tab in your browser and browse to the Hangfire dashboard.
10. Select the Servers from the navigation bar at the top of the page. Notice that it now shows 2 servers running.
11. In the list of servers, you should now see two server instances listed, each configured appropriately:

12. Now, select Recurring Jobs from the navigation bar at the top of the page. You should see two recurring jobs configured: our original recurring job (created in a previous lab), and our latest recurring job.

13. Click the checkbox next to the recurring job with Id BuildHouseJob.Run, and click the Remove button at the top of the list. The previous recurring job has now been removed.

In this lab, we have learned multiple strategies for scaling a Hangfire application: multiple queues, configurable number of worker threads, and partitioning queues among multiple Hangfire servers. With these strategies, work loads of all sizes can be configured to run efficiently.

This completes Lab 06.