



[Download Free .NET & JAVA Files API](#)

[Try Free File Format APIs for Word/Excel/PDF](#)

Scheduling and monitoring background tasks is challenging work. Every big application needs to implement background tasks to accomplish background work such as data processing, email reminders, and processing SMS queues and email queues. Windows Service is the most typical approach to meet the necessity.

Today, we are going to setup Hangfire and write some code to schedule an initial job in the ASP.NET Core project.

[Hangfire](#) is an open source library to schedule and execute background jobs in .NET applications. You'll be able to create a simple background process inside the same application pool or thread without creating separate applications. Hangfire creates background jobs in persistence storage, like MS SQL Server, Redis, MongoDB, and others, that may prevent you from losing the job on recycling IIS pools or exception prevalence.

ASP.NET Core is now a common platform for MVC and Web API with no separate project creation needed. Let's create a new ASP.NET Core MVC project. After the project is created, install Hangfire from NuGet. You can install Hangfire either from Package Management Console or NuGet Package Manager.

Install via nuget

```
01. | PM> Install-Package HangFire
```

We are going to install Hangfire using NuGet Package Manager. Search the Hangfire stable version(current 1.6.7) and install in the project.

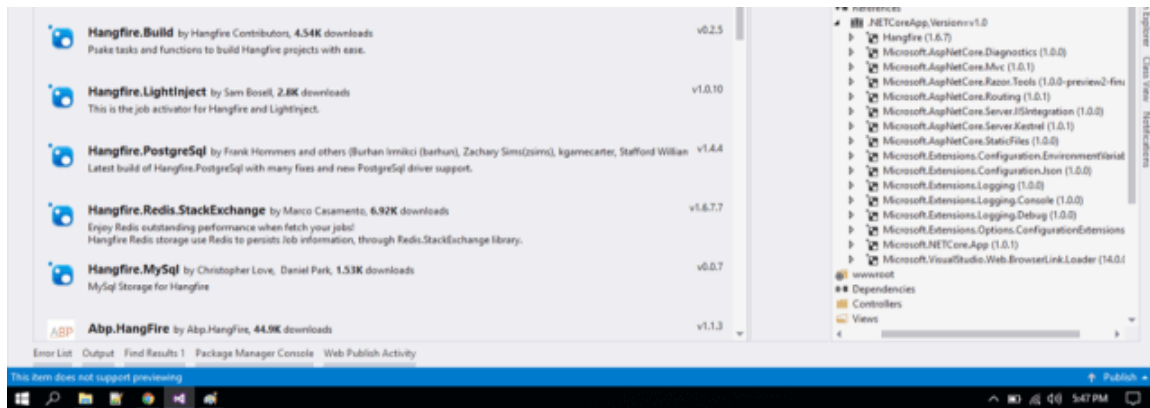
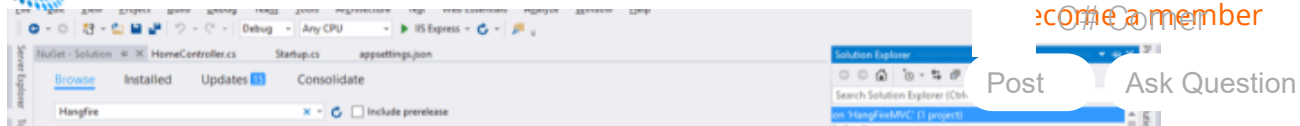


Image 1: Nuget package manager

After successful installation, let's configure Hangfire.

Configuration

Open Startup.cs class and locate a ConfigureServices method to register Hangfire as a service.

```

01. public void ConfigureServices(IServiceCollection services)
02. {
03.     // Add Hangfire services.
04.     services.AddHangfire(x => x.UseSqlServerStorage(Configuration.GetConnectionString("DefaultConnection")));
05.
06.     // Add framework services.
07.     services.AddMvc();
08. }

```

Here, we are registering Hangfire with SQL Server. We must provide a connection string to locate SQL Server database named HangFireDb. DefaultConnection is a connection string name, added to appsettings.json file.

```

01. "ConnectionStrings": {
02.     "DefaultConnection": "Data Source=.\SQLEXPRESS2014;Initial Catalog=HangFireDb;
03. }

```

Once the service is configured, navigate to Configure method to add below codes. Here, app.UseHangfireDashboard() will set up the dashboard. While http://<website-url>/hangfire, and app.UseHangfireServer() will register a new instance for [BackgroundJobServer](#).

```

01. public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
02. {
03.     loggerFactory.AddConsole(Configuration.GetSection("Logging"));
04.     loggerFactory.AddDebug();
05. }

```



```
08.
09.     app.UseDeveloperExceptionPage();
10.     app.UseBrowserLink();
11.

14.     {
15.         routes.MapRoute(
16.             name: "default",
17.             template: "{controller=Home}/{action=Index}/{id?}");
18.     });
19. }
```

Now, let's run the application. Hangfire dashboard is available in browser by hitting <http://<website-url>/hangfire> url.

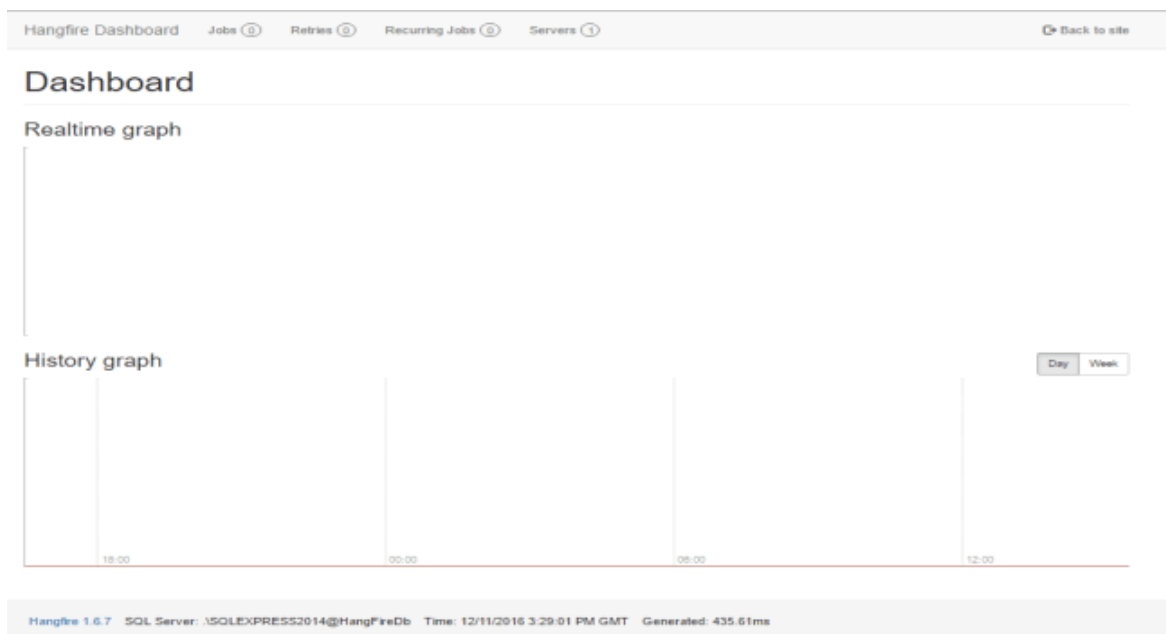


Image 2: Hangfire Dashboard.

Hangfire offers integrated web monitoring UI, which is used to control any aspect of background job processing, as well as statistics, exceptions, and background job history.

When the Hangfire Server starts, it'll look for a configured database and check for the required database schema. If the schema doesn't exist, it creates a schema. The below image shows a list of schemas created when running the application.

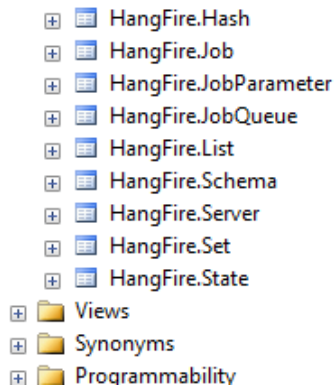
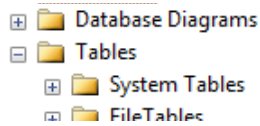


Image 3: Hangfire database schema

How does it work?

Hangfire handles different types of background jobs, and all of them are invoked in a separate execution context.

With Hangfire, you can create the following-

Fire and forget

Fire and forget jobs are executed once on an immediate basis after creation. Once you create a fire-and-forget job, it is saved to its queue ("default" by default, but multiple queues supported). The queue is listened to by a couple of dedicated workers that fetch a job and perform it.

```
01. | BackgroundJob.Enqueue(() => Console.WriteLine("Fire-and-forget Job Executed"));
```

Delayed

After the given delay, the job will be put in its queue and invoked as a regular fire-and-forget job.

```
01. | BackgroundJob.Schedule(() => Console.WriteLine("Delayed job executed"), TimeSpan.FromMinutes(1));
```

Recurring

Recurring jobs will recur on every defined interval. You can define intervals from milliseconds to years.

```
01. | RecurringJob.AddOrUpdate(() => Console.WriteLine("Minutely Job executed"), Cron.Minutely);
```

Continuations

Continuations allow you to define complex workflows by chaining multiple background jobs together. Here, the second job is queued with the first job.

After running the application, we have interesting results in a dashboard

Id	Job	Total Duration	Succeeded
#16	Console.WriteLine	62ms	a few seconds ago
#15	Console.WriteLine	49ms	a minute ago
#14	Console.WriteLine	110ms	2 minutes ago
#13	Console.WriteLine	73ms	3 minutes ago
#12	Console.WriteLine	115ms	4 minutes ago
#11	Console.WriteLine	163ms	5 minutes ago
#10	Console.WriteLine	84ms	6 minutes ago
#9	Console.WriteLine	88ms	7 minutes ago
#8	Console.WriteLine	60ms	8 minutes ago
#7	Console.WriteLine	110ms	9 minutes ago

Image 4: A list of succeeded jobs

As you can see, dashboard gives meaningful information like a list of successive jobs with Id, name, executed time, and duration. Re-queuing jobs feature has been provided to re-fire selected job from the list.

Following are the screenshots for various kinds of background jobs.

Hangfire Dashboard Jobs Retries Recurring Jobs Servers Back to site

Enqueued 0/0 Scheduled 0 Processing 0 Succeeded 0 Failed 0 Deleted 0 Awaiting 0

Console.WriteLine

The job is finished. It will be removed automatically in a day.

```
// Job ID: #1
using System;

Console.WriteLine("Fire-and-forget Job Executed");
```

CurrentCulture: "en-US"
CurrentUICulture: "en-US"

State: Succeeded 2 minutes ago (+180ms)

Latency: 441ms
Duration: 88ms

Processing +320ms
Server: DESKTOP-3CAIJ7V
Worker: 7bbbd035

Enqueued +87ms
Queue: DEFAULT

Created 2 minutes ago

Image 5: Fire and forget job example



Enqueued	0/0
Scheduled	0
Processing	0

Console.WriteLine

Post

Ask Question

The job is finished. It will be removed automatically [in a day](#).

Deleted	0
Availing	0

```
Console.WriteLine("Delayed job executed");
```

CurrentCulture: "en-US"

State

Requeue

Delete

Succeeded

3 minutes ago (+23ms)

Latency: 1m 13.740s
Duration: 4ms

Processing

+13ms

Server: DESKTOP-3CANJ7V
Worker: 7bbbd935

Enqueued

+1m 13.707s

Triggered by DelayedJobScheduler

Queue: [DEFAULT](#)

Scheduled

+10ms

Enqueue at: 3 minutes ago

Created

4 minutes ago

Image 6: Delayed job example

Enqueued	0/0
Scheduled	0
Processing	0
Succeeded	10
Failed	0
Deleted	0
Availing	0

Console.WriteLine

The job is finished. It will be removed automatically [in a day](#).

```
// Job ID: #3  
using System;  
Console.WriteLine("Hello, ");
```

CurrentCulture

"en-US"

CurrentUICulture

"en-US"

Continuations

Id	Condition	State	Job	Created
#4	OnlyOnSucceededState	Succeeded	Console.WriteLine	6 minutes ago

State

Requeue

Delete

Succeeded

6 minutes ago (+656ms)

Latency: 73ms
Duration: 69ms

Processing

+24ms

Server: DESKTOP-3CANJ7V
Worker: e02b727e

Enqueued

+33ms

Queue: [DEFAULT](#)

Created

6 minutes ago

Image 7: Continuations job example



Post

Ask Question

Enqueued	(0/0)
Scheduled	(0)
Processing	(0)
Succeeded	(11)

Awaiting	(0)
----------	-----

Console.WriteLine

The job is finished. It will be removed automatically in a day.

State

Requeue

Delete

Succeeded

7 minutes ago (+34ms)

Latency: 672ms

Duration: 1ms

Processing

+20ms

Server: DESKTOP-3CANJ7V

Worker: 7bbbd035

Enqueued

+303ms

Queue: DEFAULT

Awaiting

+343ms

Parent: #3

Next State: Enqueued

Options: OnlyOnSucceededState

Created

7 minutes ago

Image 8: Continuations dependent job example

Hangfire Dashboard	Jobs (0)	Retries (0)	Recurring Jobs (1)	Servers (1)	Back to site
--------------------	----------	-------------	--------------------	-------------	--------------

Enqueued	(0/0)
Scheduled	(0)
Processing	(0)
Succeeded	(14)
Failed	(0)
Deleted	(0)
Awaiting	(0)

Console.WriteLine

The job is finished. It will be removed automatically in a day.

```
// Job ID: #5
using System;

Console.WriteLine("Minutely Job executed");
```

RecurringJobId

"Console.WriteLine"

CurrentCulture

"en-US"

CurrentUICulture

"en-US"

State

Requeue

Delete

Succeeded

9 minutes ago (+17ms)

Latency: 88ms

Duration: 2ms

Processing

+10ms

Server: DESKTOP-3CANJ7V

Worker: 7bbbd035

Enqueued

+73ms

Triggered by recurring job scheduler

Queue: DEFAULT

Created

9 minutes ago

Image 9: Recurring job example

Change the Dashboard URL

By default, Hangfire is set on `http://<website-url>/hangfire` to access the dashboard. It's easy to configure your custom URL.



Default Hangfire configuration sets up for local environment only. To make Hangfire work on production

```
01. app.UseHangfireDashboard("/dashboard", new DashboardOptions
02. {
03.     Authorization = new [] { new HangfireAuthorizationFilter() }
04. });
```

Everything will work fine now.

That's all. You can find the detailed [documentation](#) from the official [Hangfire](#) website. You can fork the [Hangfire project](#) and make contributions on GitHub.

Download [source code](#) or fork on [github](#)

Summary

In this article, we've learned how to configure Hangfire in ASP.NET Core. We composed some code to queue the background jobs using Hangfire in ASP.NET Core. I emphatically favor Hangfire, as it is very easy to implement and the dashboard allows you to monitor and control any aspect of background job processing, as well as statistics, exceptions, and background job history.

References

- [Hangfire official website](#)
- [Hangfire overview](#)
- [Hangfire quick start](#)
- [Hangfire - GitHub](#)

Author link: [Scheduling background jobs using Hangfire in ASP.NET Core](#)

Next Recommended Article

[Job Scheduling In ASP.NET MVC With Quartz.NET](#)

In this article we will learn how to assign later task using Quartz.NET in Asp.Net MVC.

.NET Core

ASP.NET Core

ASP.NET Core MVC

HangFire



Bhavik Patel *TOP 1000*

I am Bhavik Patel from Ahmedabad. I am C# lover, tech enthusiastic, Life time learner, occasional blogger and Proud Indian. I build and break product and rich client applications

<http://www.dotnetspan.com>

716

470.4k



Type your comment here and press Enter Key (Minimum 10 characters)

Post

Ask Question

[Saktiprasad Swain](#)

1857 27 0

Jan 05, 2020

0 0 Reply



Nice article about HangFire...

[Humayun Kabir Mamun](#)

358 5.4k 0

Dec 18, 2016

1 0 Reply



Good one i will try this...

[Debendra Dash](#)

153 13.3k 5.1m

Dec 17, 2016

1 0 Reply



Usefull Info Bhavik, TQ :)

[Rathrola Prem Kumar](#)

301 6.5k 849.2k

Dec 15, 2016

1 0 Reply

FEATURED ARTICLES

[Introduction To MongoDB Atlas](#)[Why Is My SQL Server Query Slow](#)[Null Value And Null Reference Handling - C#6 To C# 9 New Features - Day One](#)[Learn To Draw Simple ASP.NET Core Blazor Bar Chart Using Canvas Extensions](#)[Host ASP.NET Core Web API On Linux Azure VM](#)[View All](#)

TRENDING UP

[01 Sealed Class Explained In C#](#)[02 Learn Angular 8 Step By Step In 10 Days - HttpClient Or Ajax Call - Day Nine](#)[03 How to Prevent Memory Leak and Profiling in Xamarin Applications](#)[04 How To Manage Our Blob Storage Account Using Logic Apps](#)[05 How To Upload A File To Amazon S3 Using AWS SDK In MVC](#)[06 Top 10 Social Media Influencers](#)[07 Future Ready Blazor Application Architecture](#)[08 What Can Be Done To Make Code Quality Better](#)[09 How To Create SSIS Catalog](#)



[Post](#)

[Ask Question](#)

[C# Tutorials](#) [Common Interview Questions](#) [Stories](#) [Consultants](#) [Ideas](#) [Certifications](#)

©2020 C# Corner. All contents are copyright of their authors.