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 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.

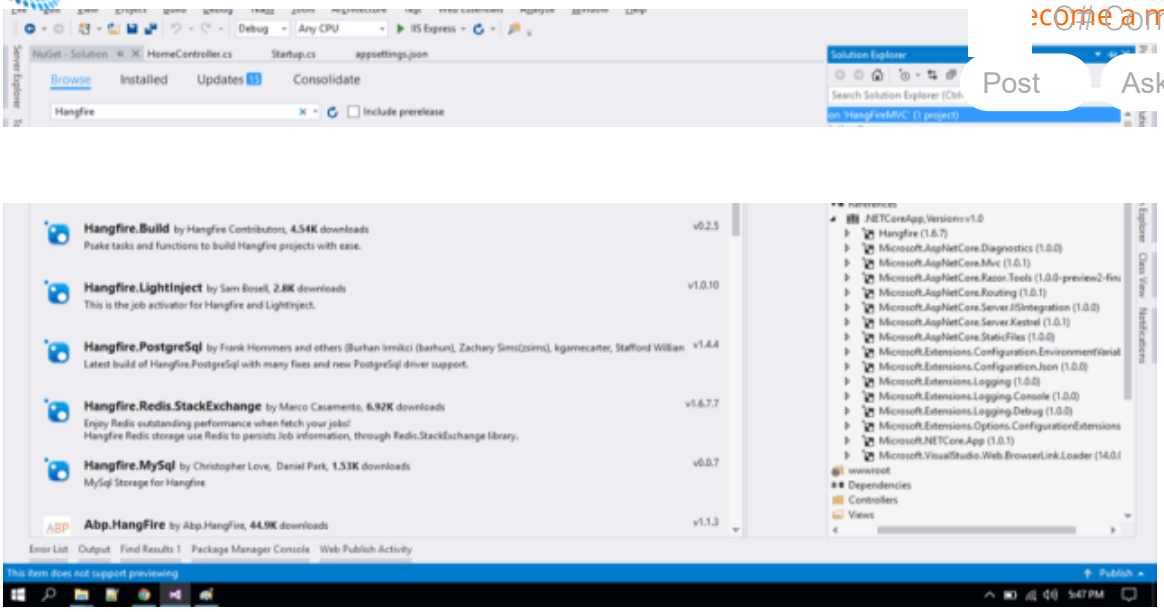

Image 1: Nuget package manager

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.GetConnectionStri
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, ILoggerFact
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.

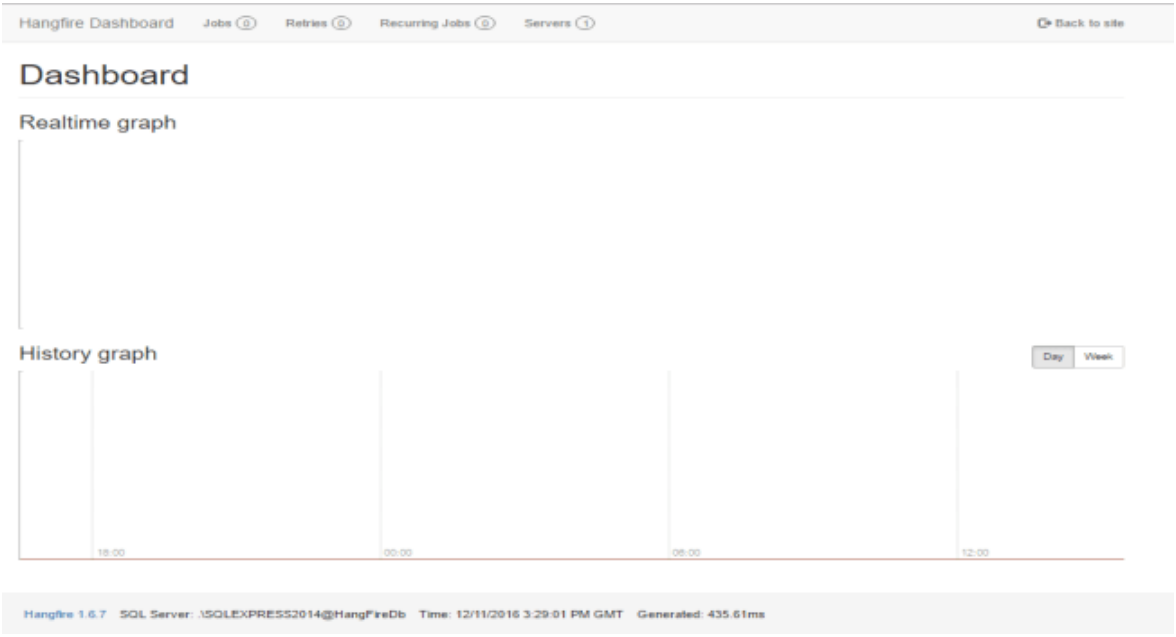

Image 2: Hangfire Dashboard.

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.

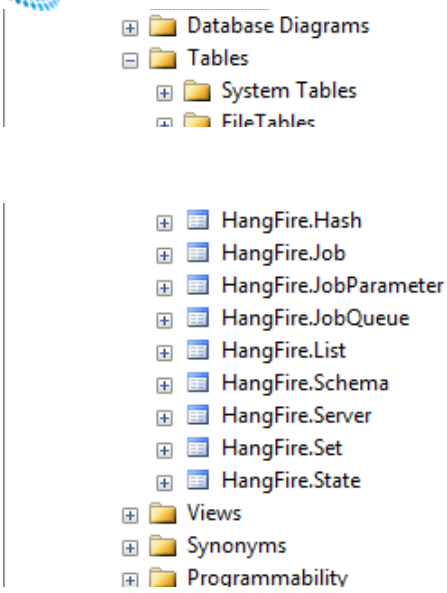

Image 3: Hangfire database schema

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.Fr
```

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.Mir
```

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

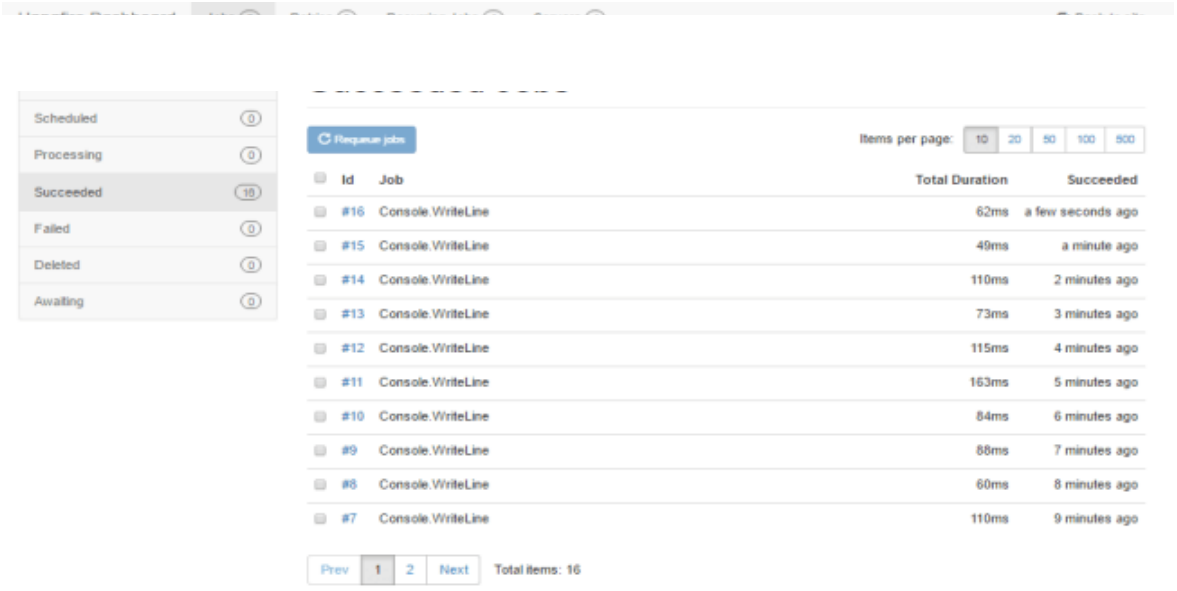

Image 4: A list of succeeded jobs

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.

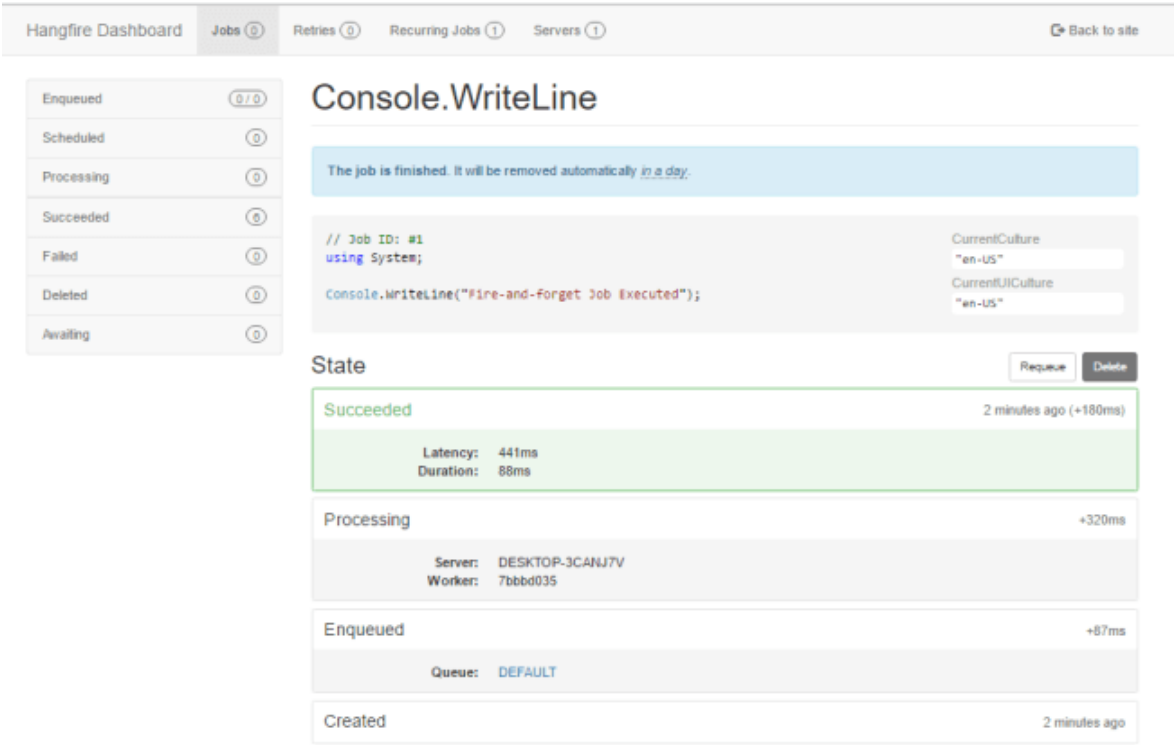

Image 5: Fire and forget job example

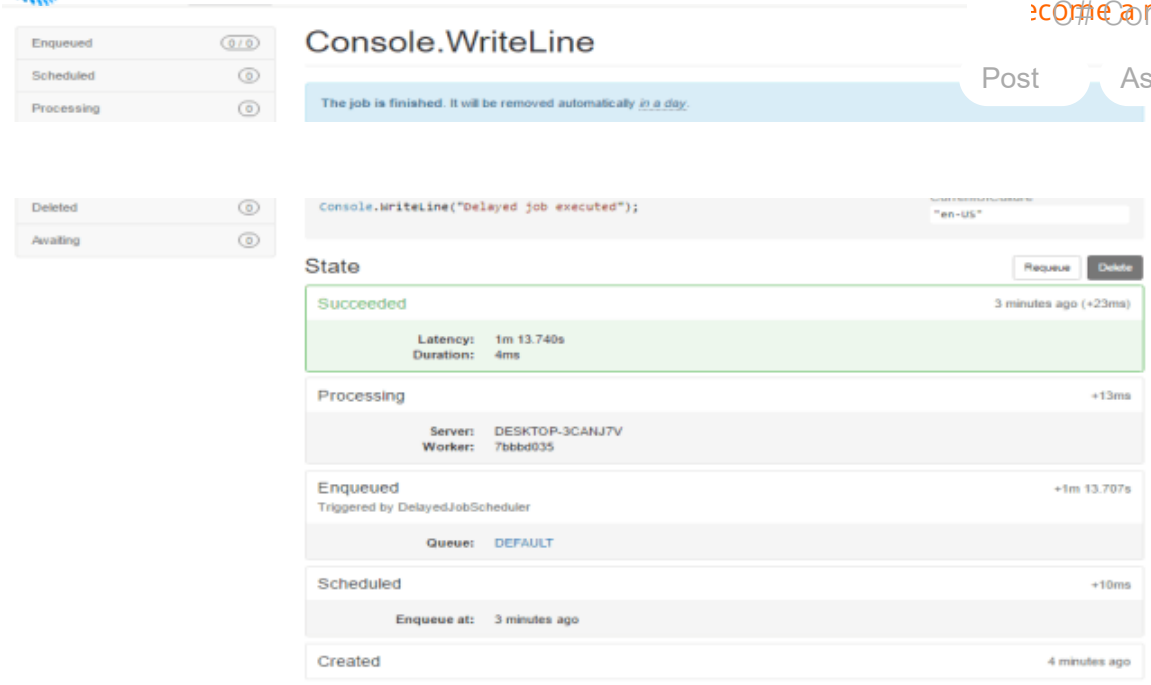

Image 6: Delayed job example

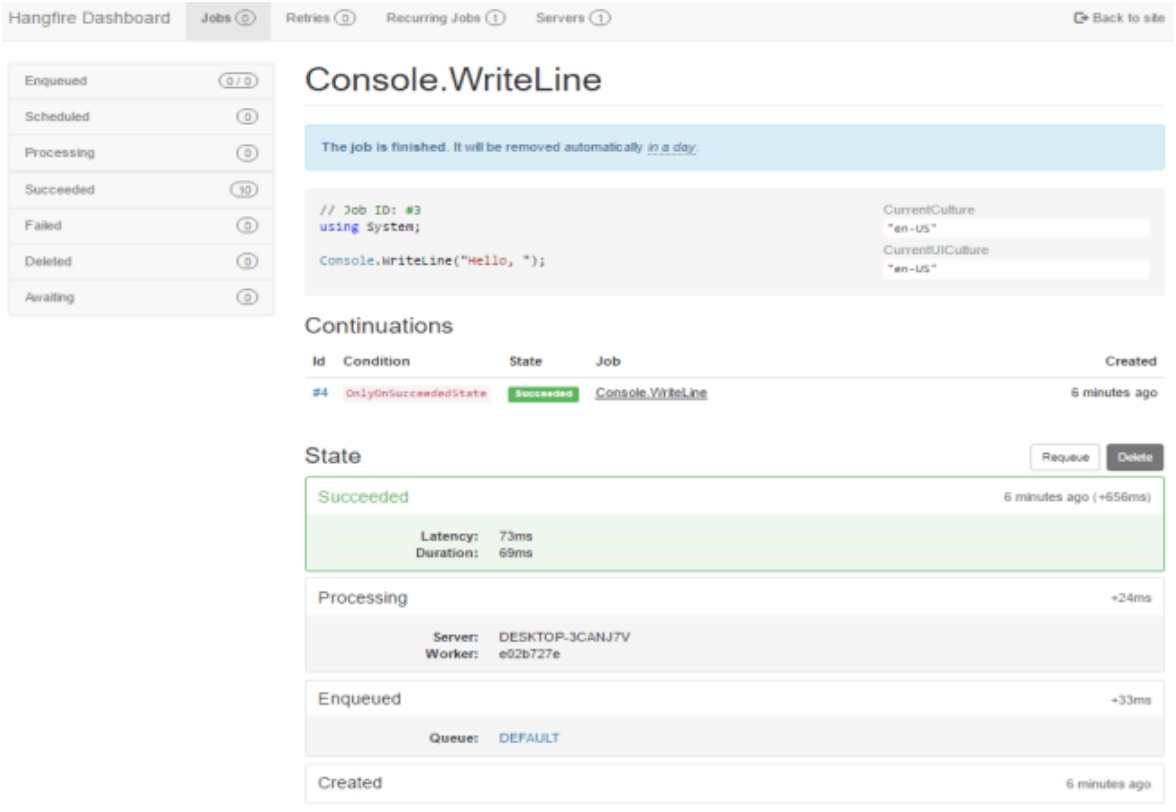

Image 7: Continuations job example

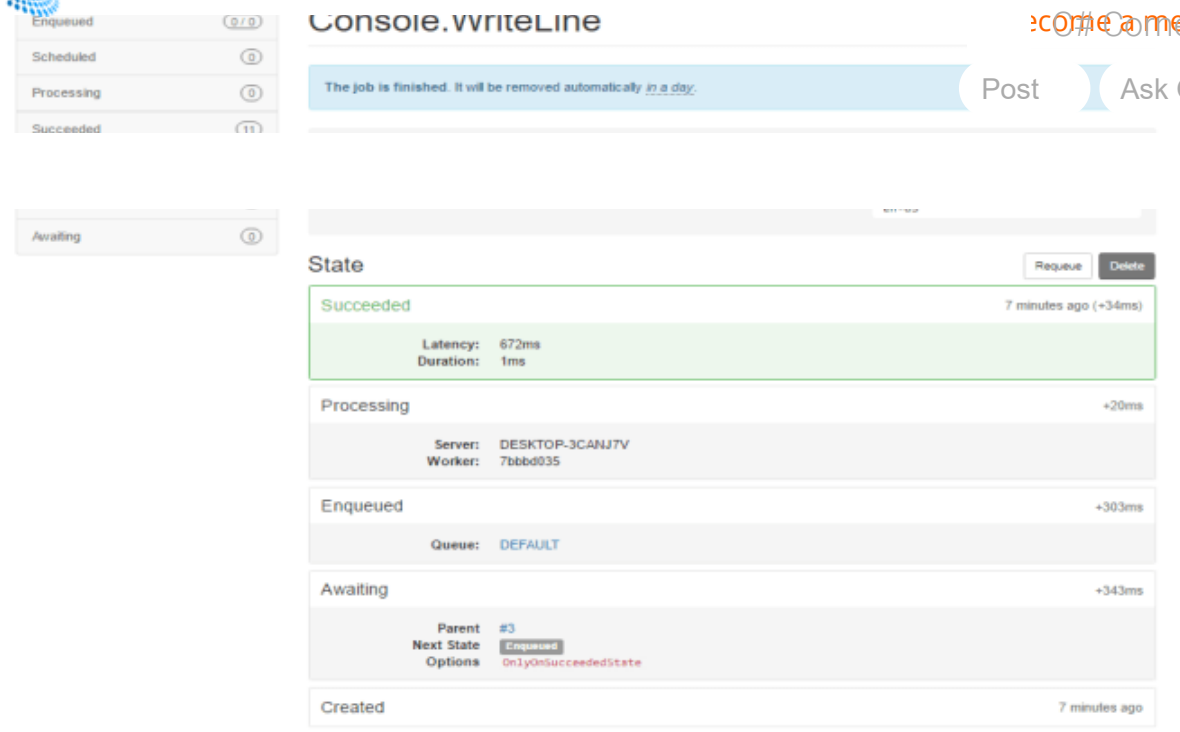

Image 8: Continuations dependent job example

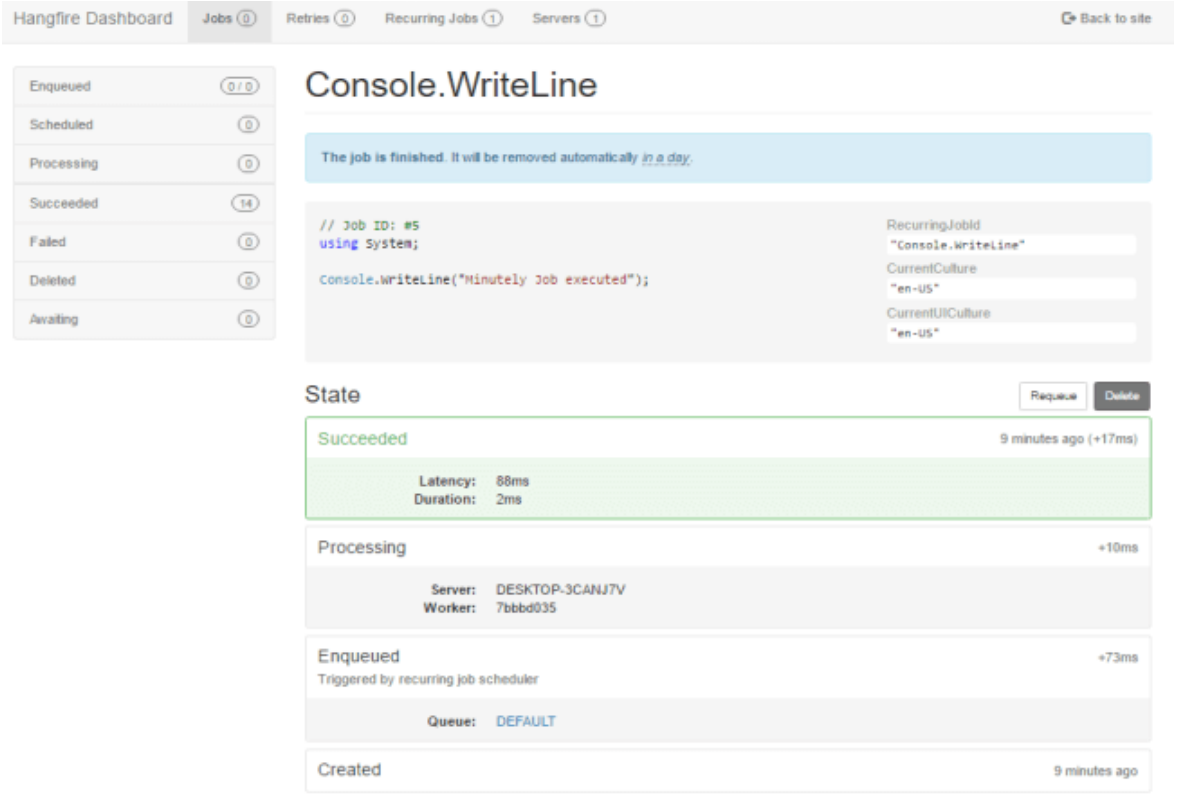

Image 9: Recurring job example

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

```
app.UseHangfireDashboard("/dashboard", new DashboardOptions
{
    Authorization = new [] { new HangireAuthorizationFilter() }
});
```

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

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