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

**Why should I use Hangfire?**

There are many job scheduling frameworks available today. Why then should you use Hangfire instead of, say, Quartz.Net, which is another popular framework that has long been in use? Well, one of the major drawbacks of Quartz.Net is that it needs a Windows Service. On the contrary, you don't need a Windows Service to use Hangfire in your application. The ability to run without a Windows Service makes Hangfire a good choice over Quartz.Net. Hangfire takes advantage of the request processing pipeline of ASP.Net for processing and executing jobs.

Note that Hangfire is not limited to Web applications; you can also use it in your Console applications. The [documentation for Hangfire](http://docs.hangfire.io/en/latest/) is very detailed and well structured, and the best feature is its built-in dashboard. The Hangfire dashboard shows detailed information on jobs, queues, status of jobs, and so on.

For respective schedulars

<https://www.hangfire.io/>

<https://github.com/icsharp/Hangfire.RecurringJobExtensions>

### Fire-and-forget jobs

Fire-and-forget jobs are executed **only once** and almost **immediately** after creation.

var jobId = BackgroundJob.Enqueue(

() => Console.WriteLine("Fire-and-forget!"));

### Delayed jobs

Delayed jobs are executed **only once** too, but not immediately, after a certain **time interval**.

var jobId = BackgroundJob.Schedule(

() => Console.WriteLine("Delayed!"),

TimeSpan.FromDays(7));

### Recurring jobs

Recurring jobs fire **many times** on the specified **CRON schedule**.

RecurringJob.AddOrUpdate(

() => Console.WriteLine("Recurring!"),

Cron.Daily);

### Continuations

Continuations are executed when its parent job **has been finished**.

BackgroundJob.ContinueWith(

jobId,

() => Console.WriteLine("Continuation!"));

### Batches

### Pro

Batch is a group of background jobs that is **created atomically** and considered as a single entity.

var batchId = BatchJob.StartNew(x =>

{

x.Enqueue(() => Console.WriteLine("Job 1"));

x.Enqueue(() => Console.WriteLine("Job 2"));

});

### Batch Continuations

### Pro

Batch continuation is fired **when all** background jobs in a parent batch **finished**.

BatchJob.ContinueWith(batchId, x =>

{

x.Enqueue(() => Console.WriteLine("Last Job"));

});

<https://www.c-sharpcorner.com/article/schedule-background-jobs-using-hangfire-in-asp-net-core/>

Install HangFire using -> npm

Startup.cs

using Hangfire;

using Hangfire.Console;

using Hangfire.RecurringJobExtensions;

public void ConfigureServices(IServiceCollection services)

{

services.AddHangfire(configuration => {

configuration.UseSqlServerStorage("Data Source=125.62.198.183;Initial Catalog=3nad;User ID=IT-TekDB;Password=u$t@y0ut");

});

}

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

app.UseHangfireDashboard();

app.UseHangfireServer();

}