Hangfire is an open-source framework for .NET that makes it easy to perform background processing of long-running tasks, such as sending emails or updating databases. Here's how you can use Hangfire in your .NET application:

**Background process Types**

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(

"myrecurringjob",

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

Cron.Daily);

Continuations

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

BackgroundJob.ContinueJobWith(

jobId,

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

Batches [**Pro**](https://www.hangfire.io/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**](https://www.hangfire.io/pro/)

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

BatchJob.ContinueBatchWith(batchId, x =>

{

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

});

**Installations**

Install the Hangfire package in your .NET project using NuGet.

Add Hangfire services to your ASP.NET Core application using Startup.cs:

public void ConfigureServices(IServiceCollection services)

{

services.AddHangfire(configuration => configuration

.SetDataCompatibilityLevel(CompatibilityLevel.Version\_170)

.UseSimpleAssemblyNameTypeSerializer()

.UseDefaultTypeSerializer()

.UseSqlServerStorage(Configuration.GetConnectionString("HangfireConnection")));

}

Add Hangfire middleware to your ASP.NET Core application using Startup.cs:

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

app.UseHangfireDashboard();

}

Create a background job:

BackgroundJob.Enqueue(() => Console.WriteLine("Hello, Hangfire!"));

Start the Hangfire background process:

app.UseHangfireServer();

**Disable Concurrent Execution**

// Add Hangfire SqlServerStorageOptions.

services.AddHangfire(configuration => configuration

.SetDataCompatibilityLevel(CompatibilityLevel.Version\_170)

.UseSimpleAssemblyNameTypeSerializer()

.UseRecommendedSerializerSettings()

.UseSqlServerStorage(Configuration.GetConnectionString("HangfireConnection"), new SqlServerStorageOptions

{

CommandBatchMaxTimeout = TimeSpan.FromMinutes(5),

SlidingInvisibilityTimeout = TimeSpan.FromMinutes(5),

QueuePollInterval = TimeSpan.Zero,

UseRecommendedIsolationLevel = true,

DisableGlobalLocks = true

}));

**SlidingInvisibilityTimeout** is used to indicate how long a BackgroundJob execution is allowed to run for without status change (success/failure) before Hangfire decides the BackgroundJob execution was not successful and needs to be made visible to the HangfireServer for processing again. The idea being that if a HangfireServer starts processing a BackgroundJob and then gets killed without being able to report back that the BackgroundJob failed then there would need to be a retry of that BackgroundJob.

You probably shouldn’t change SlidingInvisibilityTimeout unless you have a really compelling reason. For example, if you set it to 5 minutes and then have a BackgroundJob that runs for 6 minutes, Hangfire is going to end up queuing up multiple instances of that BackgroundJob because it hasn’t completed fast enough.

**QueuePollInterval** is how long the server is going to wait in between checking the database for new BackgroundJobs to process. Setting this value will result in Jobs being picked up off the queue faster, but also more load on your SQL Server.

This will disable global locks, which will prevent multiple Hangfire servers from processing the same job simultaneously. By setting QueuePollInterval to TimeSpan.Zero, the servers will not poll the queue at the same time, effectively preventing concurrent execution.

Also, you can use the **DisableConcurrentExecution** attribute to prevent multiple instances of the same job from running simultaneously. To use this attribute, simply apply it to the method that represents the background job:

[DisableConcurrentExecution(timeoutInSeconds: 3600)]

public static void MyJob()

{

// Job logic goes here.

}

In this example, the MyJob method is decorated with the **DisableConcurrentExecution** attribute and set to have a timeout of 3600 seconds (1 hour). This means that if another instance of MyJob is already running, any subsequent calls to this job will be placed in a separate queue and will wait for the first instance to finish before starting. The timeout value ensures that the lock is automatically released if the first instance takes longer than 1 hour to complete.