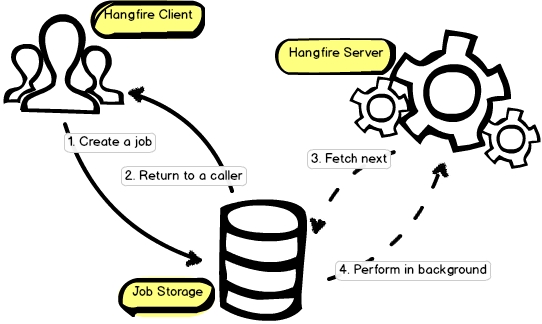
Hangfire

1. Hangfire allow us to kick off method call outside of the request processing pipeline.
2. Job are performed in a background thread called background jobs.
3. It has three main components
   1. Hangfire client
   2. Job Storage
   3. Hangfire server



1. Requirements
   1. .Net Framework / .Net Framework Core
   2. Persistent Storage
      1. Sql Server
      2. Redis
2. We can create
   1. Recurring Job
   2. Fire and forget (Immediate)
   3. Scheduled job
   4. Job Continuation (Completion of parent job trigger next job)
   5. Batch Jobs (Paid version)
   6. Batch Continuation (Paid Version)
3. It has Event Filter to track of the job status
4. Authorization Filter to track protect its security
5. Easy Load Balancing, Hangfire servers take care of Load Balance

Steps:

1. Install Nuget Packages
   1. Hangfire.Core
   2. Hangfire.AspNetCore
   3. Hangfir.SqlServer
2. Configure it in Startup.cs

**ConfigureServices:**

Services.AddHangfire((provider, config) => config

.SetDataCompatibilityLevel(CompatibilityLevel.Version\_170)

.UseSimpleAssemblyNameTypeSerializer()

.UseUseRecommendedSerializerSettings()

.UseFilter(new [Hanfire filter that catches job status and can inject dependency]()

.UseSqlServerStorage(connectionstring, new Hangfire.SqlServerStorageOptions

{

CommandBatchMaxTimeout = TimeSpan.FromMinutes(5),

SlidingInvisibilityTimeout = TimeSpan.FromMinutes(5),

QueuePollInterval = TimeSpan.Zero,

useRecommendedIsolationLevel = true,

DisableGlobalLocks = true,

PrepareSchemaIfNecessary = false

}));

Services.AddHangfireServer();

**Configure:**

App.UseHanfiredashboard(“/hangfire”, new DashboardOptions

{

DashboardTitle = “Title of the dash board”,

Authorization = new []

{

New <hangefire authorization filter>

}

}

**Event Filter:**

We can keep tracking the job status using Event filters

Ex.:

public class LogEverythingAttribute : JobFilterAttribute,

IClientFilter, IServerFilter, IElectStateFilter, IApplyStateFilter

{

private static readonly ILog Logger = LogProvider.GetCurrentClassLogger();

public void OnCreating(CreatingContext context)

{

Logger.InfoFormat("Creating a job based on method `{0}`...", context.Job.Method.Name);

}

public void OnCreated(CreatedContext context)

{

Logger.InfoFormat(

"Job that is based on method `{0}` has been created with id `{1}`",

context.Job.Method.Name,

context.BackgroundJob?.Id);

}

public void OnPerforming(PerformingContext context)

{

Logger.InfoFormat("Starting to perform job `{0}`", context.BackgroundJob.Id);

}

public void OnPerformed(PerformedContext context)

{

Logger.InfoFormat("Job `{0}` has been performed", context.BackgroundJob.Id);

}

public void OnStateElection(ElectStateContext context)

{

var failedState = context.CandidateState as FailedState;

if (failedState != null)

{

Logger.WarnFormat(

"Job `{0}` has been failed due to an exception `{1}`",

context.BackgroundJob.Id,

failedState.Exception);

}

}

public void OnStateApplied(ApplyStateContext context, IWriteOnlyTransaction transaction)

{

Logger.InfoFormat(

"Job `{0}` state was changed from `{1}` to `{2}`",

context.BackgroundJob.Id,

context.OldStateName,

context.NewState.Name);

}

public void OnStateUnapplied(ApplyStateContext context, IWriteOnlyTransaction transaction)

{

Logger.InfoFormat(

"Job `{0}` state `{1}` was unapplied.",

context.BackgroundJob.Id,

context.OldStateName);

}

}

Apply It:

public class EmailService

{

[LogEverything]

public static void Send() { }

}

GlobalJobFilters.Filters.Add(new LogEverythingAttribute());