**Worker Service in ASP .NET Core**

By Shahed C on June 10, 2019

[3 Replies](https://wakeupandcode.com/worker-service-in-asp-net-core/#comments)

This is the **twenty-third**of a [series of posts](https://wakeupandcode.com/aspnetcore/#aspnetcore2019) on ASP .NET Core in 2019. In this series, we’ll cover 26 topics over a span of 26 weeks from January through June 2019, titled **A-Z of ASP .NET Core!**

** A – Z of ASP .NET Core!**

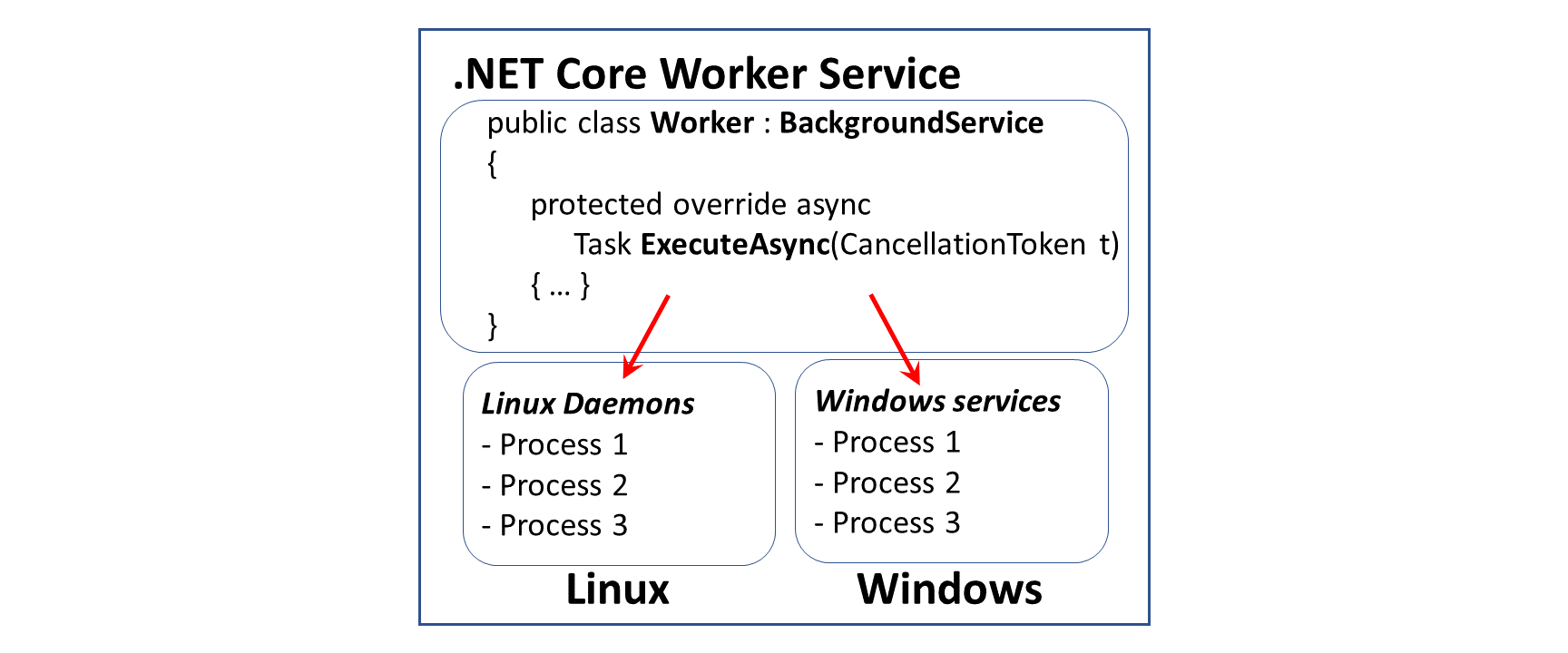
**In this Article:**

* W is for Worker Service
* New Worker Service Project
* Program and BackgroundService
* Logging in a Worker Service
* Running the Worker Service
* References

**W is for Worker Service**

When you think of ASP .NET Core, you probably think of web application backend code, including MVC and Web API. MVC Views and Razor Pages also allow you to use backend code to generate frontend UI with HTML elements. The all-new Blazor goes one step further to allow client-side .NET code to run in a web browser, using WebAssembly. And finally, we now have a template for Worker Service applications.

Briefly [mentioned in a previous post](https://wakeupandcode.com/net-core-3-vs2019-and-csharp-8/) in this series, the new project type was introduced in ASP .NET Core early previews. Although the project template is currently listed under the Web templates, it is expected to be relocated one level up in the New Project Wizard. This is a great way to create potentially long-running cross-platform services in .NET Core. This article covers the Windows operating system.

[](https://wakeupandcode.com/wp-content/uploads/2019/06/WorkerService-Linux-Windows.png)

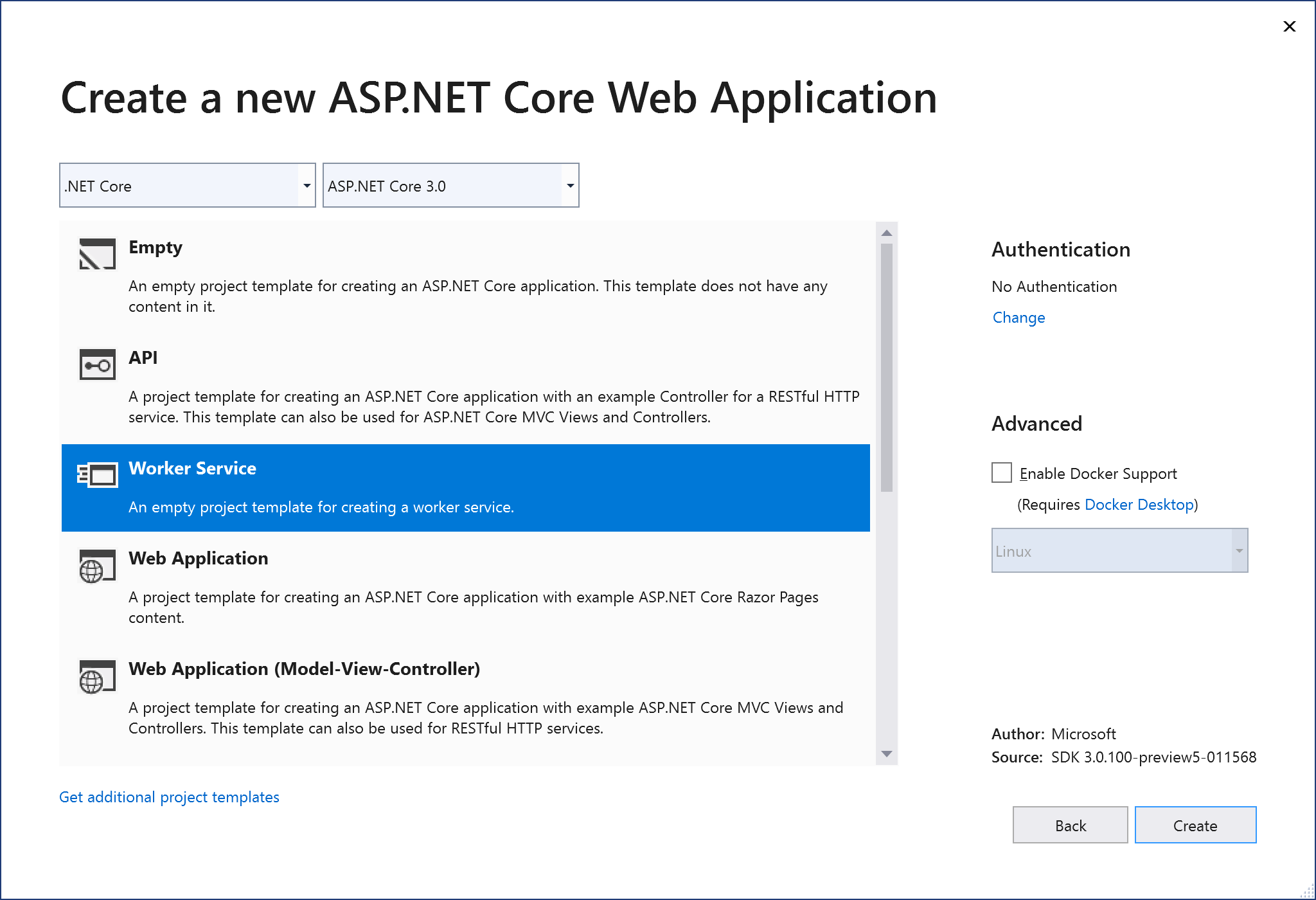
This article will refer to the following sample code on GitHub:

 Worker Service Sample: <https://github.com/shahedc/WorkerServiceSample>

**New Worker Service Project**

The quickest way to create a new Worker Service project in Visual Studio 2019 is to use the latest template available for ASP .NET Core 3.0. You may also use the appropriate dotnet CLI command.

Launch Visual Studio and select the Worker service template as shown below:

[](https://wakeupandcode.com/wp-content/uploads/2019/06/WorkerService-NewProject.png)

To use the Command Line, simply use the following command:

> dotnet new worker -o myproject

where -o is an optional flag to provide the output folder name for the project.

You can learn more about the new template at the following location:

* dotnet worker template: <https://dotnetnew.azurewebsites.net/template/Microsoft.DotNet.Web.ProjectTemplates.3.0/Microsoft.Worker.Empty.CSharp.3.0>

**Program and BackgroundService**

The [Program.cs](https://github.com/shahedc/WorkerServiceSample/blob/master/WorkerServiceSample/Program.cs) class contains the usual **Main()** method and a familiar **CreateHostBuilder()** method. This can be seen in the snippet below:

public class Program  
{  
 public static void Main(string[] args)  
 {  
 CreateHostBuilder(args).Build().Run();  
 }  
  
 public static IHostBuilder CreateHostBuilder(string[] args) =>  
 Host.CreateDefaultBuilder(args)  
 .UseWindowsService()  
 .ConfigureServices(services =>  
 {  
 services.AddHostedService<Worker>();  
 });  
 }

Things to note:

1. The Main method calls the **CreateHostBuilder**() method with any passed parameters, builds it and runs it.
2. As of ASP .NET Core 3.0, the *Web* Host Builder is being replaced by a *Generic* Host Builder. The so-called Generic Host Builder was covered in an [earlier blog post](https://wakeupandcode.com/generic-host-builder-in-asp-net-core/#generic3x) in this series.
3. **CreateHostBuilder**() creates the host and configures it by calling **AddHostService**<T>, where T is an **IHostedService**, e.g. a worker class that is a child of **BackgroundService**

The worker class, [Worker.cs](https://github.com/shahedc/WorkerServiceSample/blob/master/WorkerServiceSample/Worker.cs), is defined as shown below:

public class Worker : BackgroundService  
{  
 // ...  
   
 protected override async Task ExecuteAsync(CancellationToken stoppingToken)  
 {  
 // do stuff here  
 }  
}

Things to note:

1. The worker class implements the **BackgroundService** class, which comes from the namespace Microsoft.Extensions.Hosting
2. The worker class can then override the **ExecuteAsync()** method to perform any long-running tasks.

In the sample project, a utility class ([DocMaker.cs](https://github.com/shahedc/WorkerServiceSample/blob/master/WorkerServiceSample/Utils/DocMaker.cs)) is used to convert a web page (e.g. a blog post or article) into a Word document for offline viewing. Fun fact: when this A-Z series wraps up, the blog posts will be assembled into a free ebook, by using this EbookMaker, which uses some 3rd-party NuGet packages to generate the Word document.

**Logging in a Worker Service**

Logging in ASP .NET Core has been covered in great detail in an earlier blog post in this series. To get a recap, take a look at the following writeup:

* Logging in ASP .NET Core: <https://wakeupandcode.com/logging-in-asp-net-core/>

To use Logging in your Worker Service project, you may use the following code in your [Program.cs](https://github.com/shahedc/WorkerServiceSample/blob/master/WorkerServiceSample/Program.cs) class:

using Microsoft.Extensions.Logging; public static IHostBuilder CreateHostBuilder(string[] args) =>  
 Host.CreateDefaultBuilder(args)  
 .UseWindowsService()  
 .ConfigureLogging(loggerFactory => loggerFactory.AddEventLog())  
 .ConfigureServices(services =>  
 {  
 services.AddHostedService<Worker>();  
 });

1. Before using the extension method, add its NuGet package to your project:

* Microsoft.Extensions.Logging.EventLog

1. Add the appropriate namespace to your code:

* using Microsoft.Extensions.Logging;

1. Call the method **ConfigureLogging**() and call the appropriate logging method, e.g. **AddEventLog**()

The list of available loggers include:

* AddConsole()
* AddDebug()
* AddEventLog()
* AddEventSourceLogger()

The [Worker class](https://github.com/shahedc/WorkerServiceSample/blob/master/WorkerServiceSample/Worker.cs) can then accept an injected **ILogger<Worker>**object in its constructor:

private readonly ILogger<Worker> \_logger;  
  
public Worker(ILogger<Worker> logger)  
{  
 \_logger = logger;  
}

**Running the Worker Service**

**NOTE**: Run Powershell in Administrator Mode before running the commands below.

Before you continue, add a call to **UseWindowsService**() in your [Program class](https://github.com/shahedc/WorkerServiceSample/blob/master/WorkerServiceSample/Program.cs), or verify that it’s already there. The official announcement and initial document referred to **UseServiceBaseLifetime**() in an earlier preview. This method has been renamed to **UseWindowsService**() in the most recent version.

public static IHostBuilder CreateHostBuilder(string[] args) =>  
 Host.CreateDefaultBuilder(args)  
 .UseWindowsService()  
 .ConfigureServices(services =>  
 {  
 services.AddHostedService<Worker>();  
 });

According to the code documentation, **UseWindowsService**() does the following:

1. Sets the host lifetime to WindowsServiceLifetime
2. Sets the Content Root
3. Enables logging to the event log with the application name as the default source name

You can run the Worker Service in various ways:

1. Build and Debug/Run from within Visual Studio.
2. Publish to an exe file and run it
3. Run the sc utility (from Windows\System32) to create a new service

To publish the Worker Service as an exe file with dependencies, run the following **dotnet** command:

dotnet publish -o C:\path\to\project\pubfolder

The -o parameter can be used to specify the path to a folder where you wish to generate the published files. It could be the path to your project folder, followed by a new subfolder name to hold your published files, e.g. pubfolder. Make a note of your EXE name, e.g. MyProjectName.exe but omit the pubfolder from your source control system.

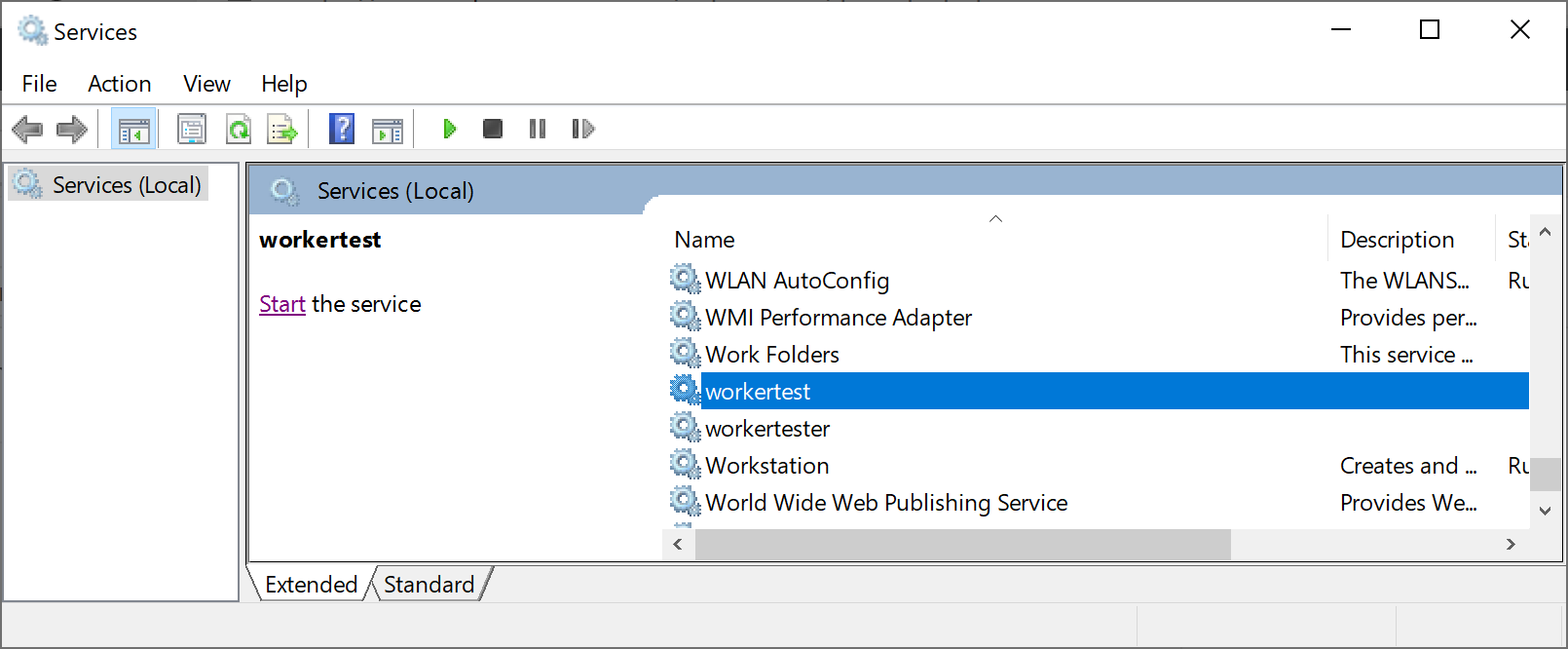
To create a new service, run **sc.exe** from your System32 folder and pass in the name of the EXE file generated from the publish command.

> C:\Windows\System32\sc create MyServiceName binPath=C:\path\to\project\pubfolder\MyProjectName.exe

When running the service manually, you should see some logging messages, as shown below:

info: WorkerServiceSample.Worker[0]  
 Making doc 1 at: 06/09/2019 00:09:52 -04:00  
Making your document...  
info: WorkerServiceSample.Worker[0]  
 Making doc 2 at: 06/09/2019 00:10:05 -04:00  
Making your document...  
info: Microsoft.Hosting.Lifetime[0]  
 Application started. Press Ctrl+C to shut down.  
info: Microsoft.Hosting.Lifetime[0]  
 Hosting environment: Development

After the service is installed, it should show up in the operating system’s list of Windows Services:

[](https://wakeupandcode.com/wp-content/uploads/2019/06/WorkerService-WindowsServices.png)

**NOTE**: When porting to other operating systems, the call to **UseWindowsService**() is safe to leave as is. It doesn’t do anything on a non-Windows system.

**References**

* .NET Core Workers as Windows Services: <https://devblogs.microsoft.com/aspnet/net-core-workers-as-windows-services/>
* Worker Service template in .NET Core 3.0: <https://gunnarpeipman.com/net/worker-service/>
* Worker Service Template in .NET Core 3.0 – DZone Web Dev: <https://dzone.com/articles/worker-service-template-in-net-core-30>
* ASP.NET Blog | .NET Core Workers in Azure Container Instances: <https://devblogs.microsoft.com/aspnet/dotnet-core-workers-in-azure-container-instances/>