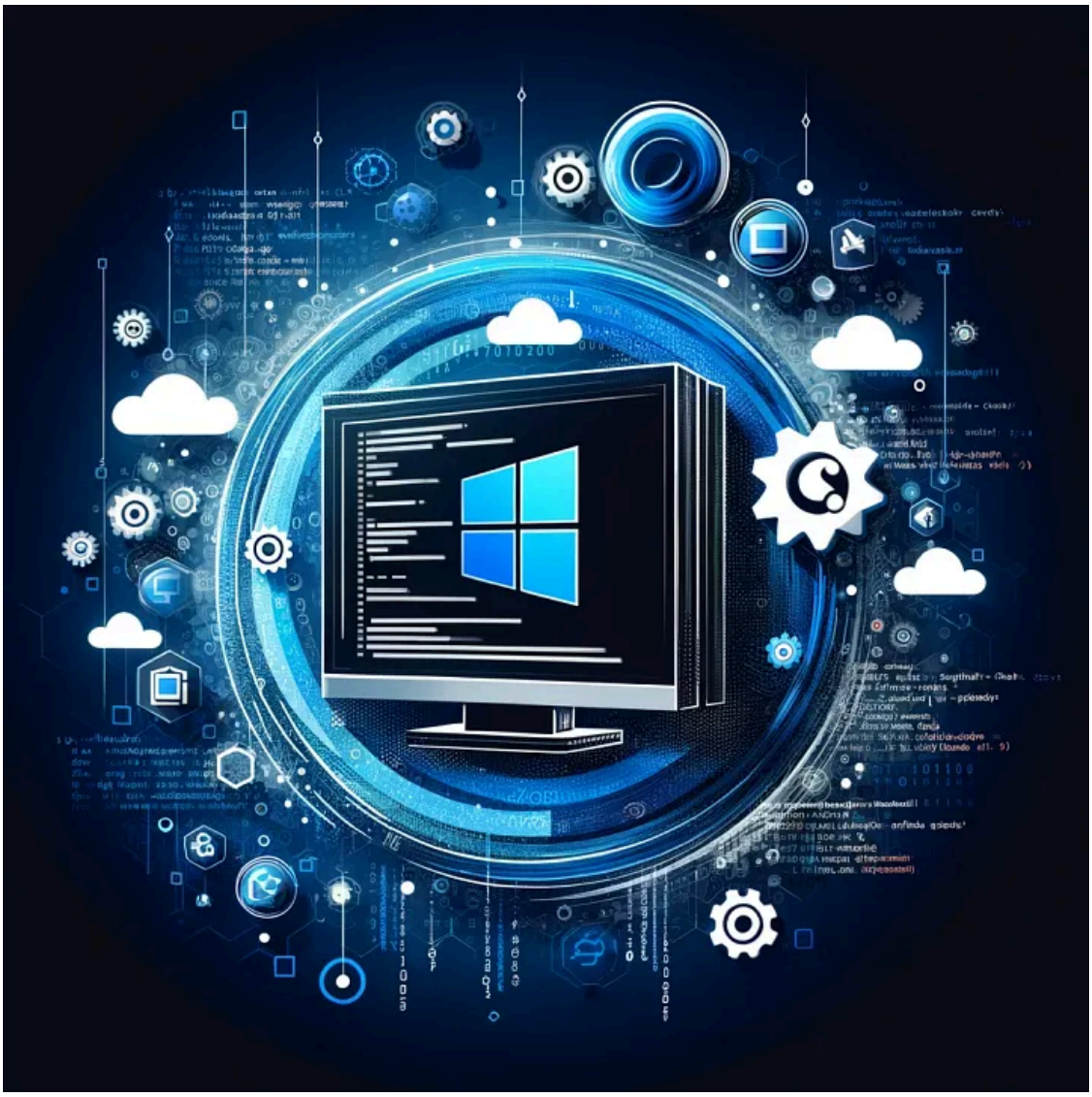
medium.com /@tivole/create-windows-services-using-net-8-0-worker-service-ea6b8f1f20a1

Create Windows Services using .NET 8.0 (Worker Service)

Kamran Asgarov ⁝ 5-6 minutes ⁝ 2024/3/3



Installing the Tools

Make sure you have installed .NET 8 in your machine:

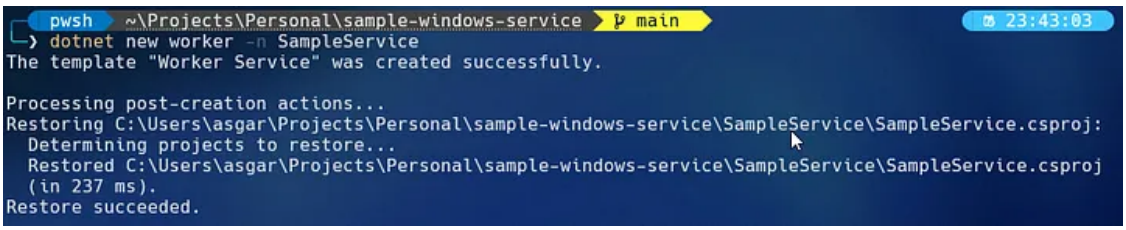
```
pwsh ~\Projects\Personal\sample-windows-service main
❯ dotnet --version
8.0.101
```

If you don't have it installed, you can download and install it from here: https://dotnet.microsoft.com/en-us/download/dotnet/8.0

Creating a Windows Service Project

We will user dotnet CLI to create a worker project:

```
dotnet new worker -n SampleService
```



Once project is created, the solution looks as following:

To interoperate with the Windows Service Control Manager (SCM), we need to add a reference to the `Microsoft.Extensions.Hosting.WindowsServices` package. Open a terminal and navigate to the project folder and run the following command:

```
dotnet add package Microsoft.Extensions.Hosting.WindowsServices
```

After adding the package, we need to modify the `Program.cs` file to use the *WindowsService*:

```
using SampleService;

var host = Host.CreateDefaultBuilder(args)
    .UseWindowsService(options => {
        options.ServiceName = "SampleService";
    })
    .ConfigureServices((hostContext, services) =>
    {
        services.AddHostedService<Worker>();
    })
    .Build();

host.Run();
```

Main logic of the windows service is in the `Worker.cs` file:

```
namespace SampleService;

public class Worker : BackgroundService
{
    private readonly ILogger<Worker> _logger;

    public Worker(ILogger<Worker> logger)
    {
        _logger = logger;
    }
```

```
    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            if (_logger.IsEnabled(LogLevel.Information))
            {
                _logger.LogInformation("Worker running at: {time}", DateTimeOffset.Now);
            }
            await Task.Delay(1000, stoppingToken);
        }
    }
}
```

It is a simple worker class that logs the current time every second.

Adding Serilog to the Service

We will add ***Serilog*** to our service to write logs to a file.

First, we need to install the Serilog package. Open a terminal and navigate to the project folder and run the following command:

```
dotnet add package Serilog.Sinks.File
```

To add *Serilog* to dotnet host builder we need to install following package:

```
dotnet add package Serilog.Extensions.Hosting
```

Now, let's add *Serilog* to the `Program.cs` file:

```
using SampleService;
using Serilog;

Log.Logger = new LoggerConfiguration()
    .WriteTo.File(
        Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "sample-service.log")
    )
    .CreateLogger();

var host = Host.CreateDefaultBuilder(args)
    .UseWindowsService(options => {
        options.ServiceName = "SampleService";
    })
    .UseSerilog()
    .ConfigureServices((hostContext, services) =>
    {
        services.AddHostedService<Worker>();
    })
    .Build();

host.Run();
```

Publishing the Service

First, we need to publish the service. Open a terminal and navigate to the project folder and run the following command:

```
dotnet publish -o .\publish -c Release -p:PublishSingleFile=true
```

This command will publish the service to the ***publish*** folder. The `-c` Release option specifies that the service should be built in release mode. The `-p:PublishSingleFile=true` option specifies that the service should be published as a single file.

```
pwsh  ~\Projects\Personal\sample-windows-service\SampleService  main ~2                    02:09:17
❯ dotnet publish -o .\publish -c Release -p:PublishSingleFile=true
MSBuild version 17.8.3+195e7f5a3 for .NET
  Determining projects to restore...
  All projects are up-to-date for restore.
  SampleService -> C:\Users\asgar\Projects\Personal\sample-windows-service\SampleService\bin\Release\n
  et8.0\win-x64\SampleService.dll
  SampleService -> C:\Users\asgar\Projects\Personal\sample-windows-service\SampleService\publish\
```

Deploying the Service

Now the we have a successfully compiler service, let's install it on a system and then run it. One of the well known tools for this is `sc.exe` , a built-in Windows tool for manipulating services. We'll use this tool to install and then run the service. Not that installation and running of services is a privileged operation and only allowed for administrators.

Open an ***elevated*** command window and type the following:

```
sc create "SampleService" binPath="C:\\path\\to\\publish\\SampleService.exe"
```

If all goes well, the output should indicate success.

To run windows service we need to run following command:

```
sc start "SampleService"
```

To stop the service, you can run the following command:

```
sc stop "SampleService"
```

Testing the Service

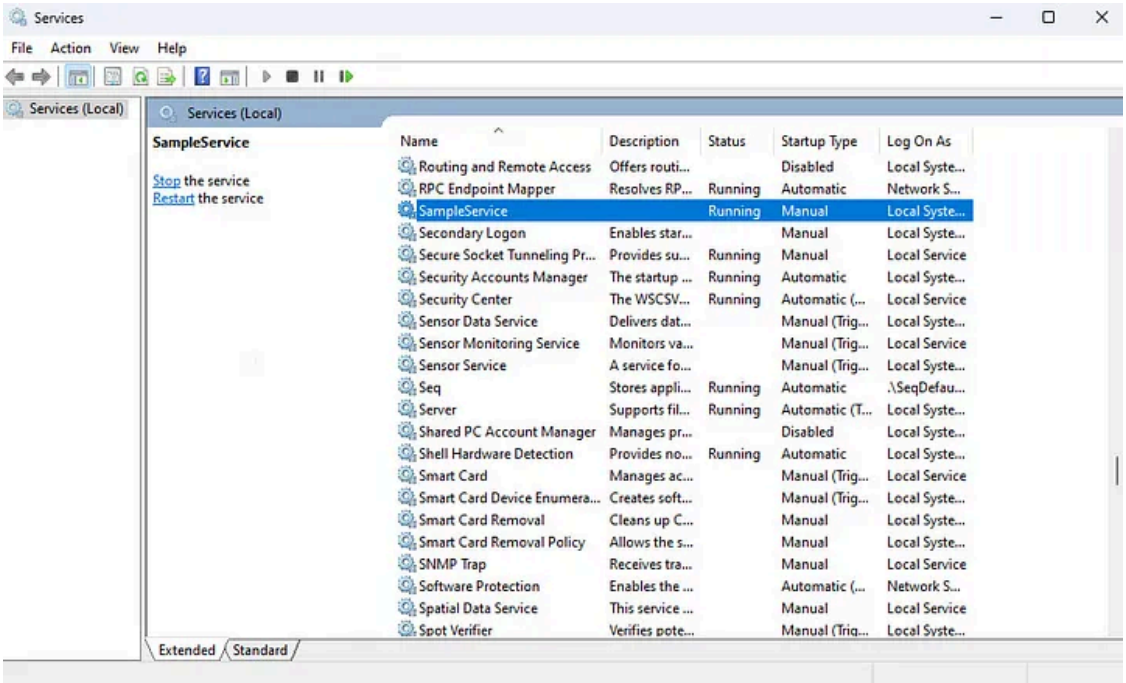
To test the service, open the `sample-service.log` file in the `publish` folder. You should see log entries that indicate the service is running:

```
2024-03-02 00:00:00 [INF] Application started. Hosting environment: Production; Content root path:
C:\path\to\publish
2024-03-02 00:00:01 [INF] Worker running at: "2024-03-02T00:00:01"
2024-03-02 00:00:02 [INF] Worker running at: "2024-03-02T00:00:02"
2024-03-02 00:00:03 [INF] Worker running at: "2024-03-02T00:00:03"
```

After stopping the service, the log entries should stop.

```
2024-03-02 00:00:03 [INF] Worker running at: "2024-03-02T00:00:03"
2024-03-02 00:00:04 [INF] Application is shutting down...
```

Also you can see ***SampleService*** in the list of services in the `services.msc`:



References

- Source Code GitHub repository
- Create a Windows service using .NET Core