MICROSOFT ARCHITECT
By Joydip Kanjilal, Columnist, InfoWorld
APR 29, 2019 3:00 AM PDT

How to use IHostedService in ASP.Net Core

Take advantage of the IHostedService interface to execute background tasks in your ASP.Net Core applications

We often need to execute background tasks and scheduled jobs in our applications. To implement background tasks in ASP.Net Core, you can take advantage of Azure WebJobs or any of a number of third-party task schedulers like Quartz or Hangfire.

In ASP.Net Core, you can implement background tasks as hosted services as well. A hosted service is a class that implements the IHostedService interface and includes the necessary code for running tasks in the background. This article presents a discussion of how we can build hosted services in ASP.Net Core.

[.Net roadmap: The new features you can expect in .Net Standard 2.1. | .Net Framework or .Net Core? Learn when to use which. | Keep up with hot topics in programming with InfoWorld's App Dev Report newsletter.]

At the time of this writing, Visual Studio 2019 is available for free download. If you don't already have a copy of Visual Studio 2019 installed in your system, you can download it from Microsoft's Visual Studio downloads page.

Create an ASP.Net Core project in Visual Studio

First off, let's create an ASP.Net Core project in Visual Studio. To do that, follow the steps given below to create a new ASP.Net Core project in Visual Studio 2019.

1. Launch the Visual Studio IDE.
2. Click on "Create new project."

3. In the “Create new project” window, select “ASP.Net Core Web Application” from the list of the templates displayed.
4. Click Next.
5. In the “Configure your new project” window, specify the name and location for the new project.
6. Click Create.
7. In the “Create New ASP.Net Core Web Application” window, select .Net Core as the runtime and ASP.Net Core 2.2 (or later) from the drop-down list at the top.
8. Select “Web Application” as the project template.
9. Ensure that the check boxes “Enable Docker Support” and “Configure for HTTPS” are unchecked as we won’t be using those features here.
10. Ensure that Authentication is set as “No Authentication” as we won’t be using authentication either.
11. Click Create.

This would create a new ASP.Net Core project in Visual Studio. We’ll use this project in the subsequent sections of this article to build our custom hosted service.

Create a hosted service in ASP.Net Core

To create our hosted service, we’ll implement the IHostedService interface. Here is what the IHostedService interface looks like:

```
public interface IHostedService
{
    Task StartAsync(CancellationToken cancellationToken);
    Task StopAsync(CancellationToken cancellationToken);
}
```

In this section we’ll create a minimalistic implementation of a hosted service in ASP.Net Core. To create a new hosted service, follow these steps.

1. Select the project in the Solution Explorer window.
2. Right-click and select “Add -> Class...”
3. In the “Add New Item” window, specify a name for the class (e.g. MyFirstHostedService).
4. Click Add when done.

This will create a new class named MyFirstHostedService in a file of the same name with a “.cs” extension. Now replace the default MyFirstHostedService class with the following code.

```
public class MyFirstHostedService : IHostedService
   {
       protected async override Task
        ExecuteAsync(CancellationToken token)
       {
           throw new NotImplementedException();
       }
   }
```

Use the BackgroundService class in ASP.Net Core

An abstract helper class called BackgroundService is available in .Net Core that already implements the IHostedService interface. Let’s rewrite our code to extend this class.

```
public class MyFirstHostedService : BackgroundService
    {
        protected async override Task
         ExecuteAsync(CancellationToken token)
        {
            throw new NotImplementedException();
        }
    }
```

You can examine the source code of the BackgroundService class on GitHub.

The following code snippet illustrates a simple method to log the current time in a text file. This method will be called from our hosted service.

```
private async Task Log()
        {
            using (StreamWriter sw = new
              StreamWriter(@"D:\log.txt",true))
            {
                await sw.WriteLineAsync
                (DateTime.Now.ToLongTimeString());
            }
        }
```

Use the ExecuteAsync method in ASP.Net Core

Here is the implementation of the ExecuteAsync method. Note how the Log method we created above is called from this method at regular intervals (every second to be precise).

```
protected async override Task ExecuteAsync
   (CancellationToken token)
        {
            while (!token.IsCancellationRequested)
            {
                await Log();
                await Task.Delay(1000, token);
            }
        }
```

And here is the complete source code of the MyFirstHostedService for your reference.

```
using Microsoft.Extensions.Hosting;
using System;
using System.IO;
using System.Threading;
using System.Threading.Tasks;
namespace HostedServicesApp
{
    public class MyFirstHostedService : BackgroundService
    {
        protected async override Task
         ExecuteAsync(CancellationToken token)
        {
            while (!token.IsCancellationRequested)
            {
                await Log();
                await Task.Delay(1000, token);
            }
        }
        private async Task Log()
        {
            using (StreamWriter sw = new
             StreamWriter(@"D:\log.txt",true))
            {
                await sw.WriteLineAsync
                (DateTime.Now.ToLongTimeString());
            }
        }
    }
}
```

Register the hosted service in ASP.Net Core

Lastly, you should register the hosted service in the Startup class as shown in the code snippet below.

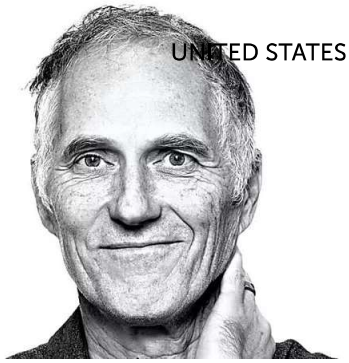
```
public void ConfigureServices(IServiceCollection services)
  {
    services.AddHostedService<MyFirstHostedService>();
    services.AddMvc().SetCompatibilityVersion
     (CompatibilityVersion.Version_2_2);
  }
```

Now when you run this ASP.Net Core application, a log file will be created at the location specified and the current time will be logged every second.

The IHostedService interface can be used to start or stop background tasks in an ASP.Net Core web application. You can take advantage of this interface to implement your own custom hosted service with the ability to cancel and shut down gracefully if need be. A typical use case for using a hosted service might be executing background tasks to update some content or data in your application. Offloading this activity to a background thread will ensure that the main request thread is not blocked.

Joydip Kanjilal is a Microsoft MVP in ASP.Net, as well as a speaker and author of several books and articles. He has more than 20 years of experience in IT including more than 16 years in Microsoft .Net and related technologies.

Follow



Tim O'Reilly: the golden age of the programmer is over

The best open source software of 2020

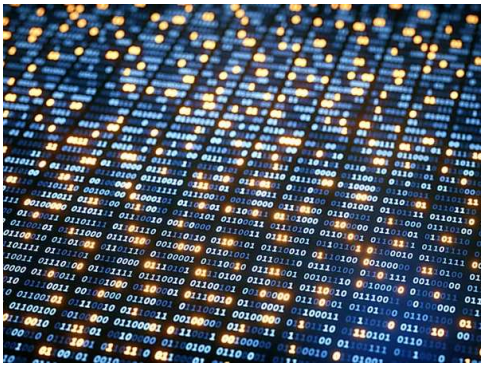
6 ways Alibaba Cloud challenges AWS, Azure, and GCP

How to use SortedDictionary, SortedList, and SortedSet in C#

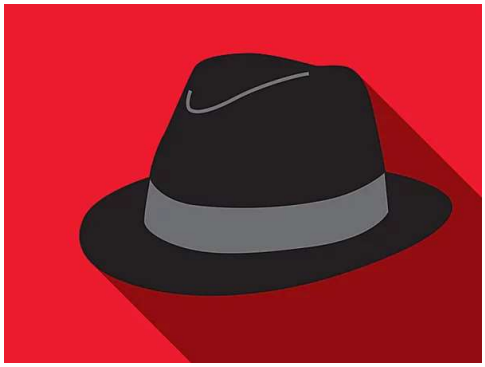
What's new in Kubernetes 1.20

Prisma ORM for Node.js is ready for production

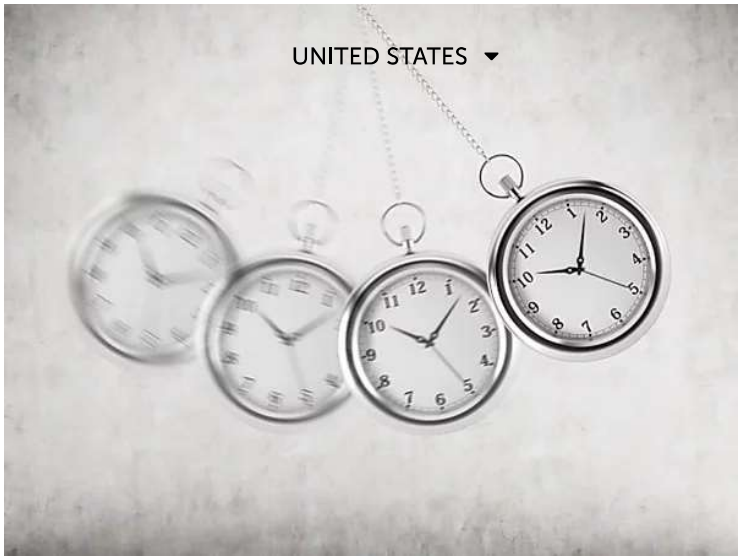
Edge computing archetypes are emerging,



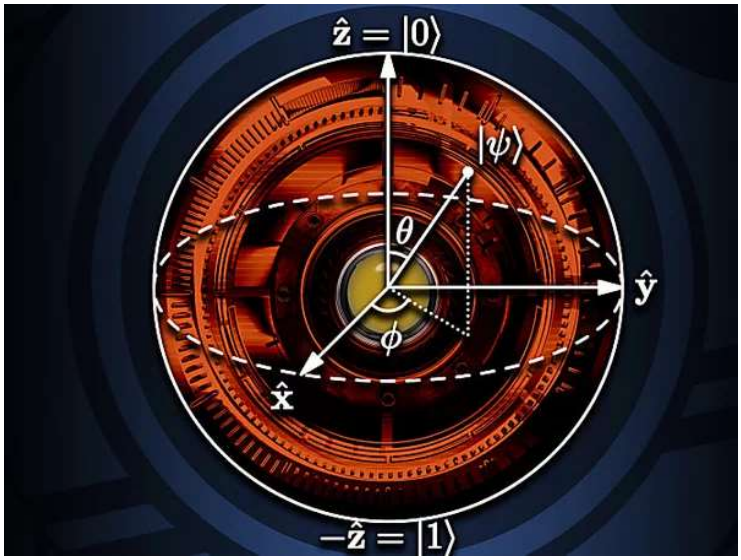
What is a computational storage drive? Much-



Red Hat OpenShift ramps up security and



When to use Task.WaitAll vs. Task.WhenAll in .NET

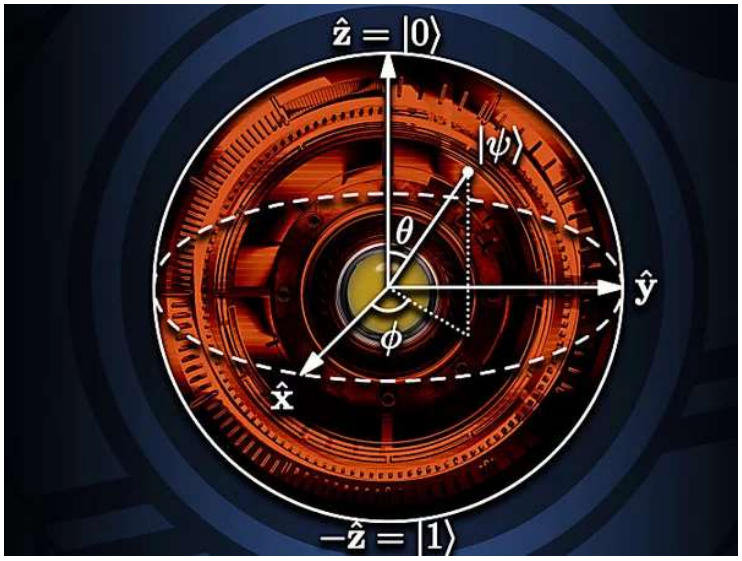


Amazon Braket: Get started with quantum computing

Deno Company forms to back Node.js rival

3 enterprise AI success stories

Edge computing archetypes are emerging,



Amazon Braket: Get started with quantum computing

When to use Task.WaitAll vs. Task.WhenAll in .NET