## References

* SignalR

<https://www.asp.net/signalr>

* Tutorial: Getting Started with SignalR 2 and MVC 5

<https://www.asp.net/signalr/overview/getting-started/tutorial-getting-started-with-signalr-and-mvc>

* Using SignalR with Web Apps in Azure App Service

<https://www.asp.net/signalr/overview/deployment/using-signalr-with-azure-web-sites>

* Event Hubs programming guide

<https://docs.microsoft.com/en-us/azure/Event-Hubs/event-hubs-programming-guide>

* Get started with Event Hubs and Receive messages with EventProcessorHost

<https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-csharp-ephcs-getstarted>

* Event Processor Host Best Practices

<https://blogs.msdn.microsoft.com/servicebus/2015/01/16/event-processor-host-best-practices-part-1/>

* Getting Started with Azure IoT services: Event Processor Host

<https://www.linkedin.com/pulse/getting-started-azure-iot-services-event-processor-host-rob-tiffany>

## Requirements

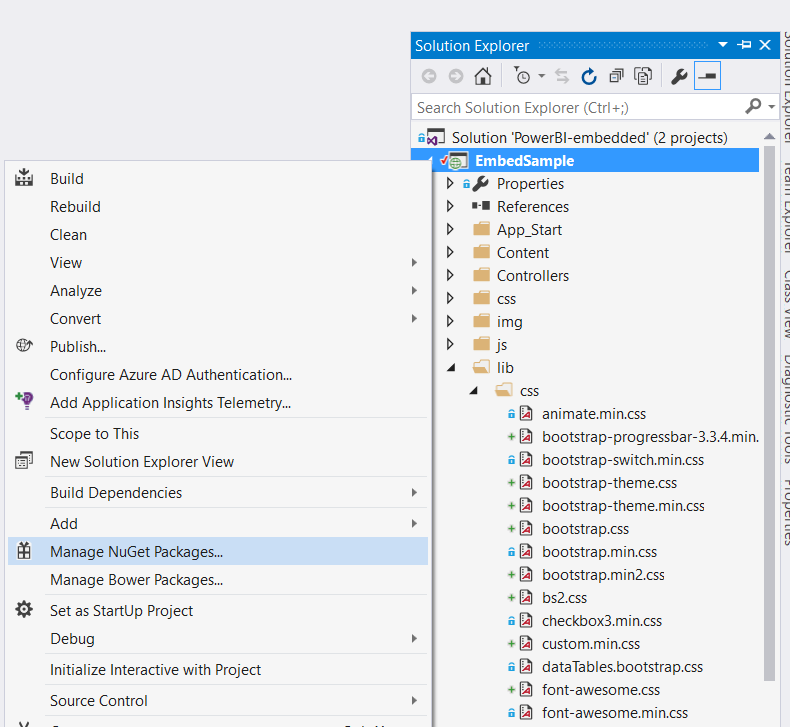
* Finished the part 9 of HOL
* Simulated Linux & Windows Wind turbines
* NuGet packages
  + Newtonsoft.Json for JSON in C#
  + Microsoft.AspNet.SignalR for SignalR
  + WindowsAzure.ServiceBus and Microsoft.Azure.ServiceBus.EventProcessorHost for Event Processor Host

## Goals

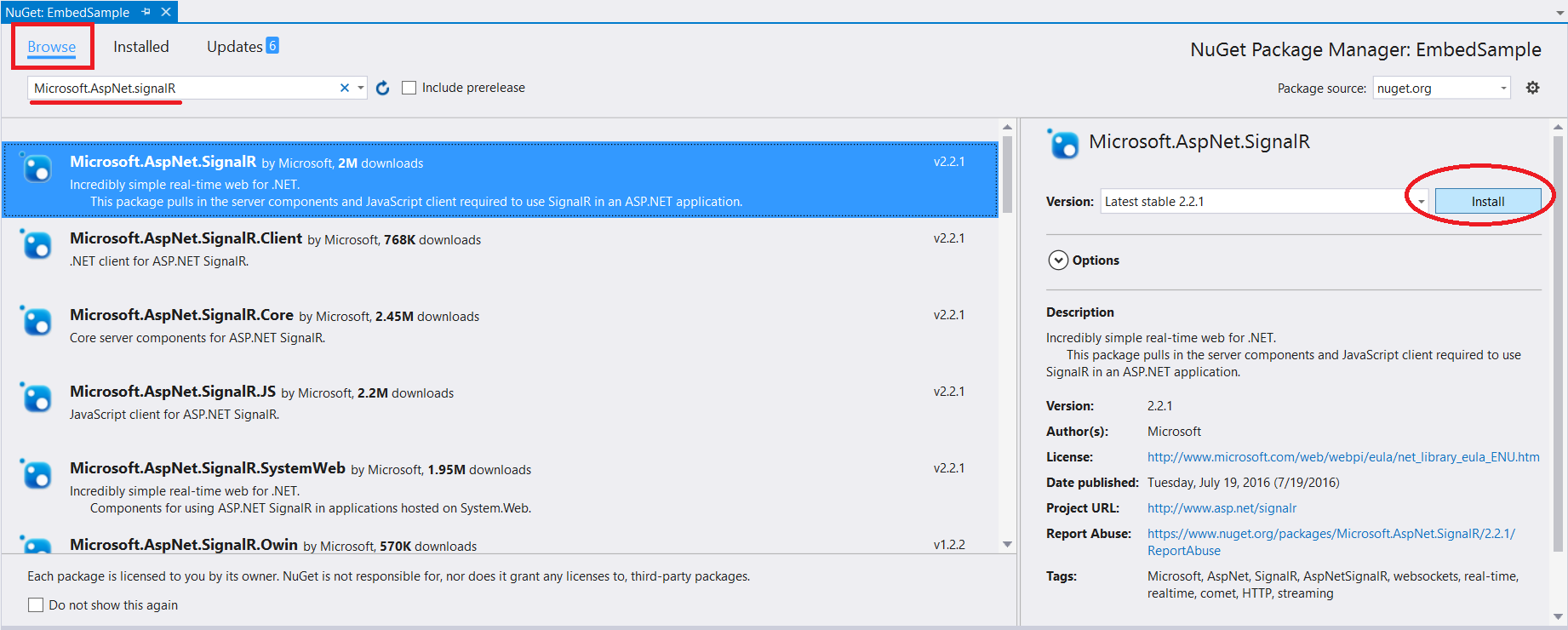
* Use the **Signal R** to communication between Web Server and Browser Client.
* Use the **Event Processor Host** to push the telemetry data to Web Server.
* Show the real time dashboard in the web portal.

## Step 1: Add the SignalR Library

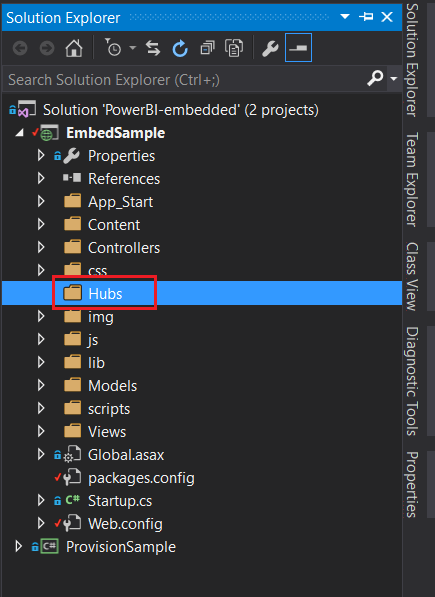
* Open the **Solution Explorer** | **EmbedSample** | **Manage NuGet Packages** and get the SignalR library.



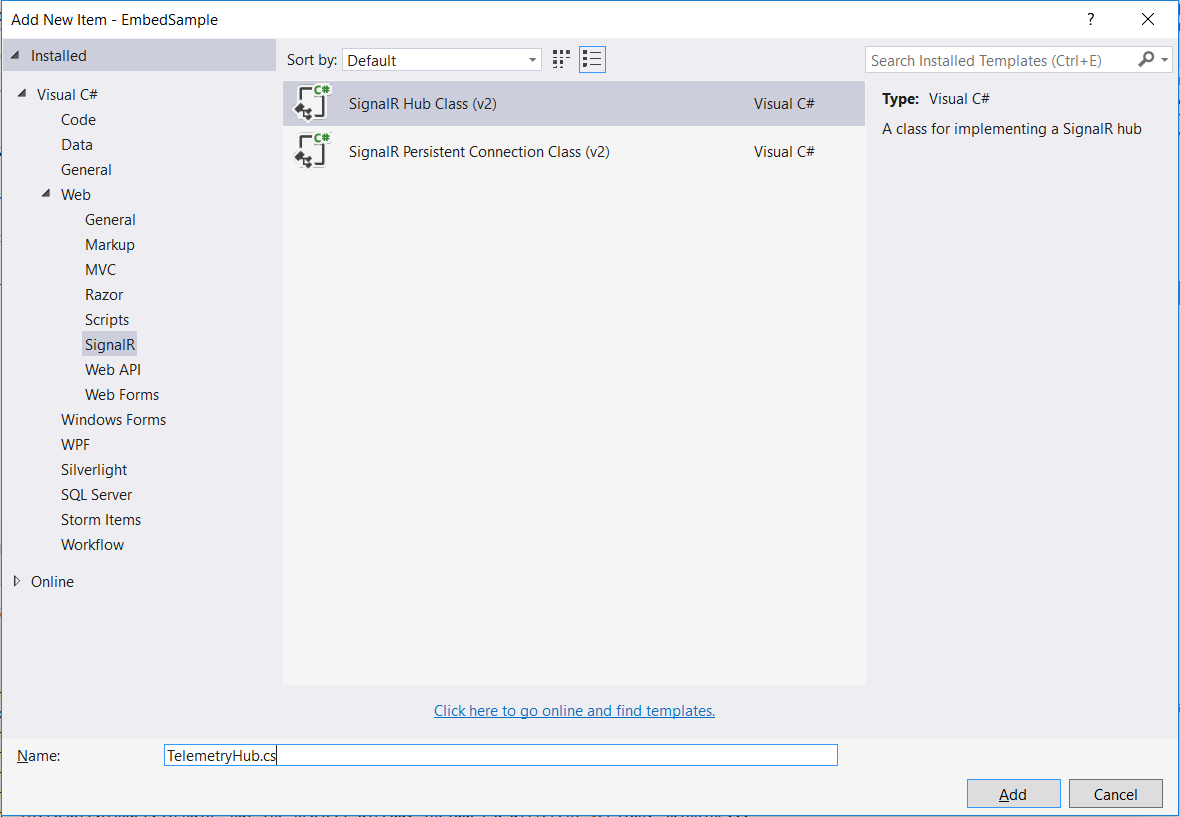
* + **Browse** to find **Microsoft.AspNet.signalR**, then install this package.

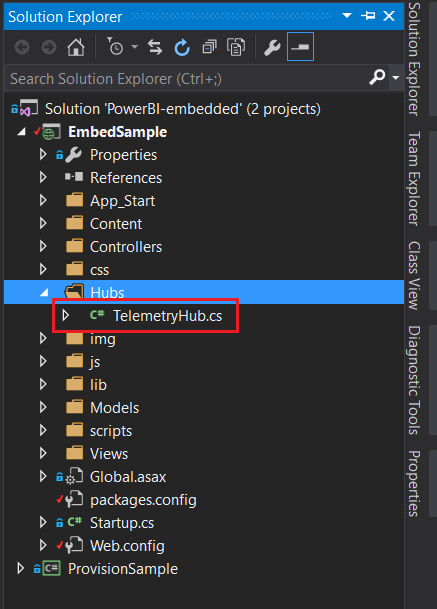


* In **Solution Explorer**, right-click the project **EmbedSample**, select **Add** | **New Folder**, and add a new folder named **Hubs**.



* Right-click the **Hubs** folder, click **Add** | **New Item**, select the **Visual C#** | **Web** | **SignalR** node in the **Installed** pane, select **SignalR Hub Class (v2)** from the center pane, and create a new hub named **TelemetryHub.cs**. You will use this class as a SignalR server hub that sends messages to all clients.





* Replace the code in the **TelemetryHub** class with the following code.

using Microsoft.AspNet.SignalR;

namespace paas\_demo.Hubs

{

public class TelemetryHub : Hub

{

public void Hello()

{

System.Diagnostics.Debug.WriteLine("Hello!");

}

}

}

* Update the **Startup.cs** of Root directory. Change the contents of the file to the following.

using Microsoft.Owin;

using Owin;

[assembly: OwinStartupAttribute(typeof(paas\_demo.Startup))]

namespace paas\_demo

{

public partial class Startup

{

public void Configuration(IAppBuilder app)

{

// Any connection or hub wire up and configuration should go here

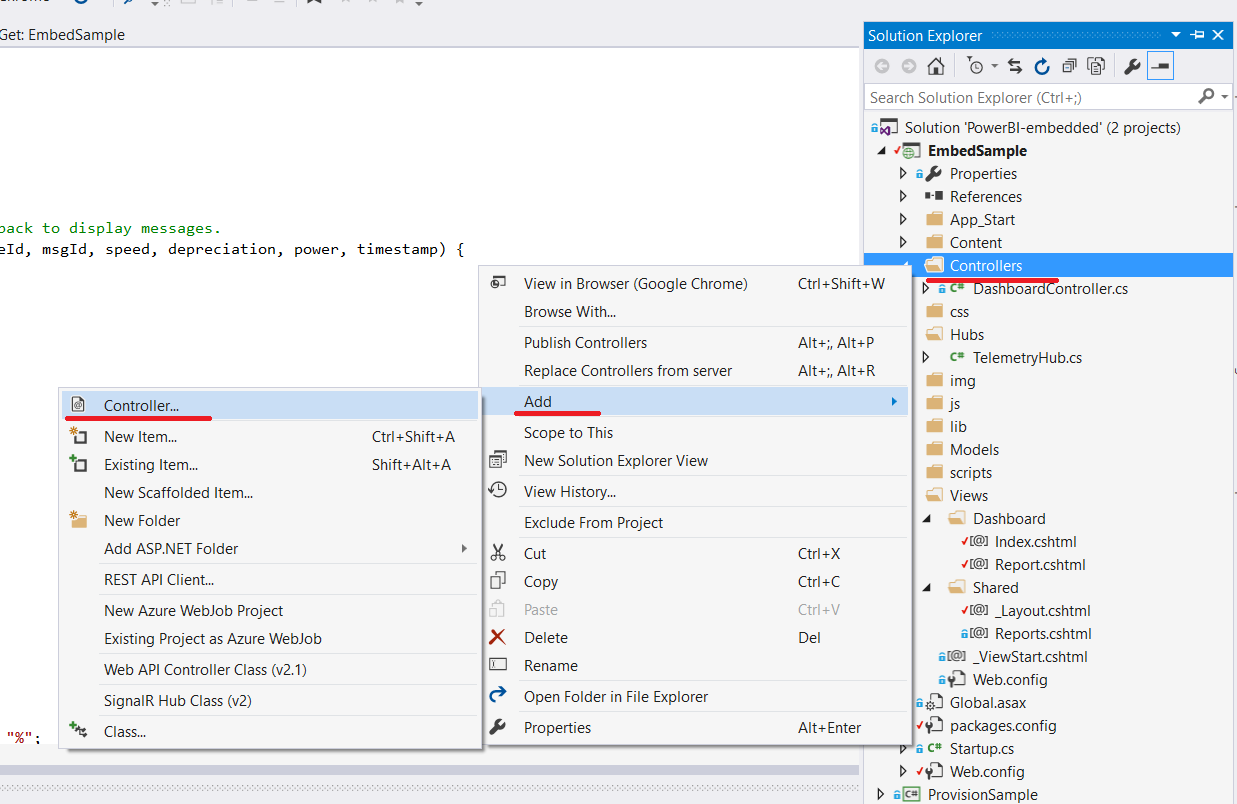
app.MapSignalR();

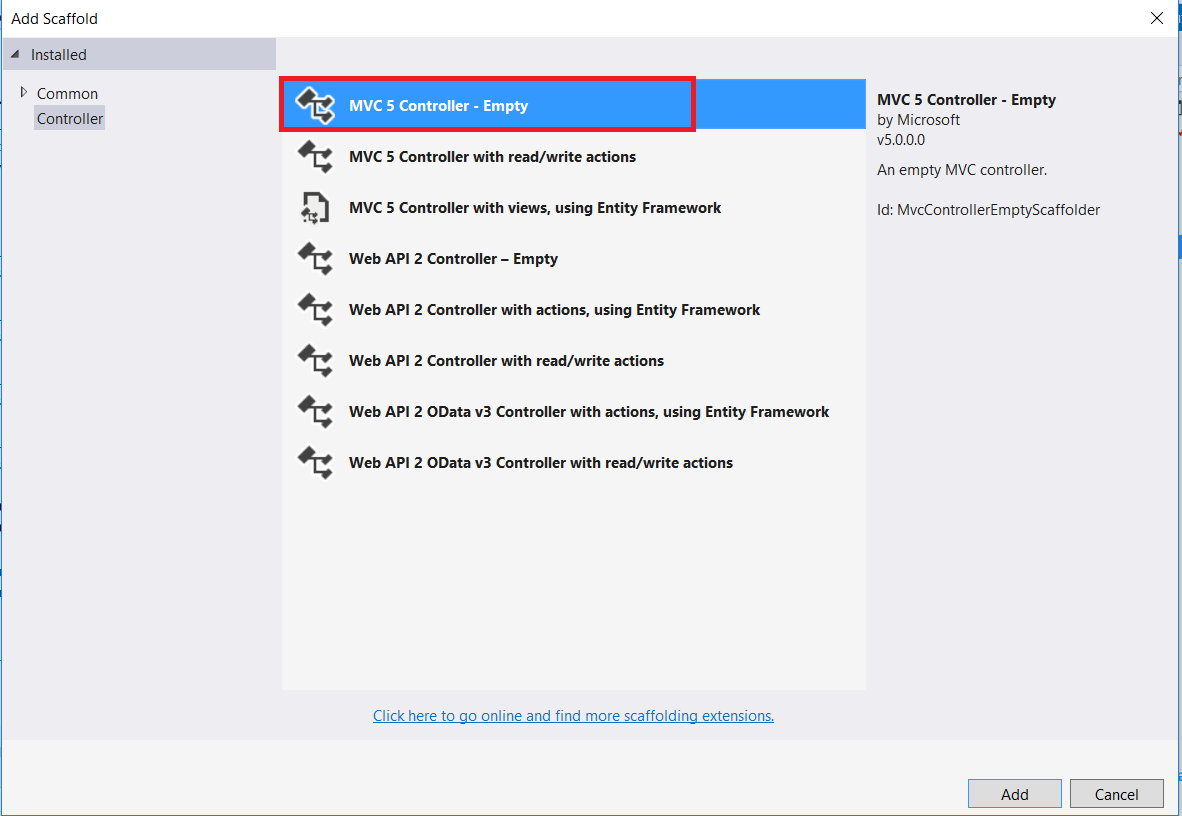
}

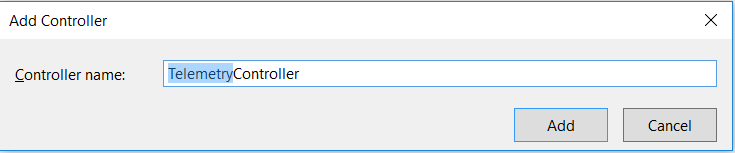
}

}

* Add the **TelemetryController.cs** class in Controllers.







* Update the **TelemetryController.cs**. Change the contents of the file to the following. This method returns the Empty view that you will create in a later step.

using Microsoft.AspNet.SignalR;

using paas\_demo.Hubs;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace paas\_demo.Controllers

{

public class TelemetryController : Controller

{

// POST Request from Event Processor Host

[HttpPost]

public ActionResult PutTelemetry(string deviceId, string msgId, double speed, double depreciation, double power, string time)

{

System.Diagnostics.Debug.WriteLine("deviceId = {0}, msgId = {1}, speed = {2}, depreciation = {3}, power = {4}, time = {5}",

deviceId,

msgId,

speed,

depreciation,

power,

time);

DateTime eventTime = DateTime.Parse(time);

long epoch = (eventTime.Ticks - 621355968000000000) / 10000;

var context = GlobalHost.ConnectionManager.GetHubContext<TelemetryHub>();

context.Clients.All.sendTelemetry(deviceId, msgId, speed, depreciation, power, epoch);

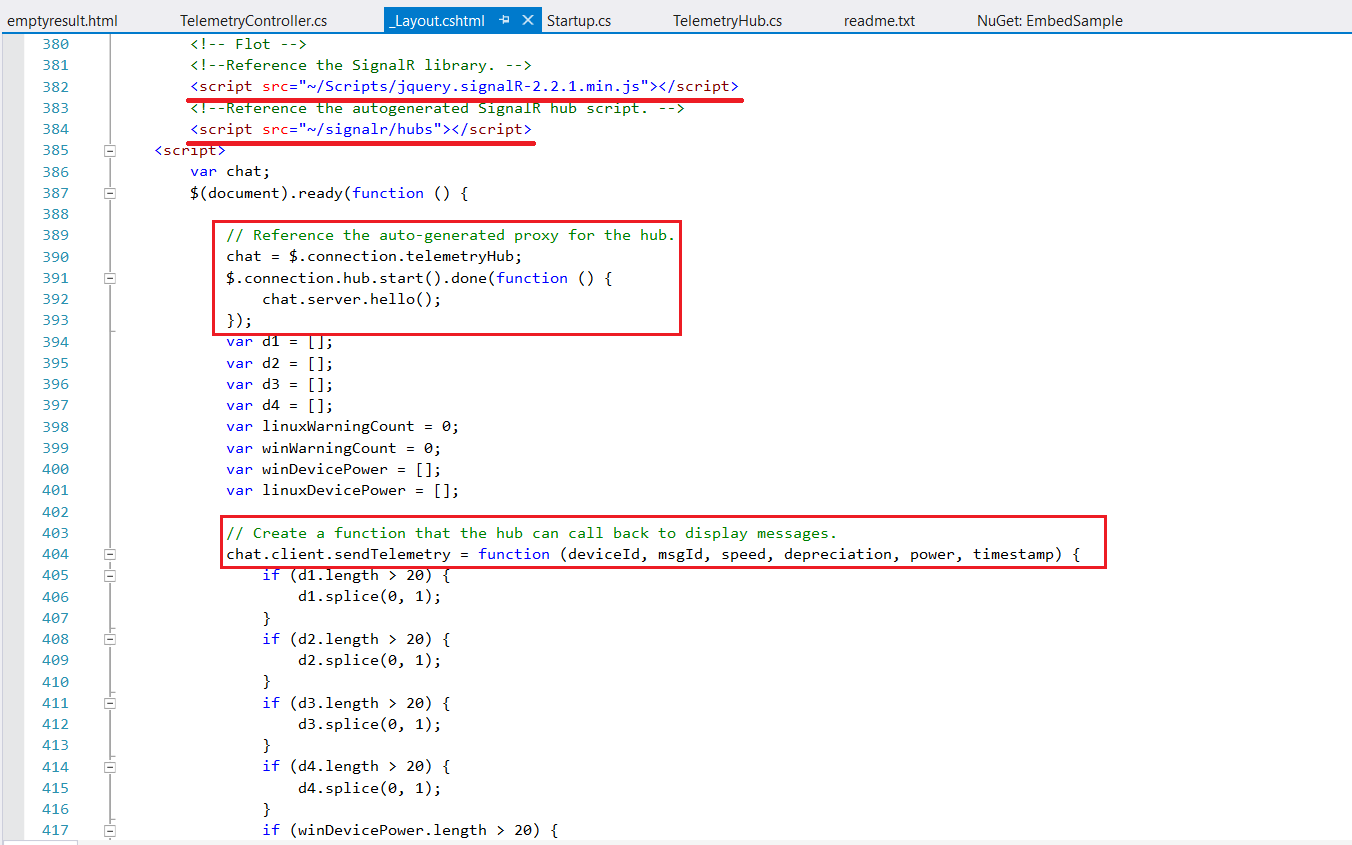
return this.Content("");

}

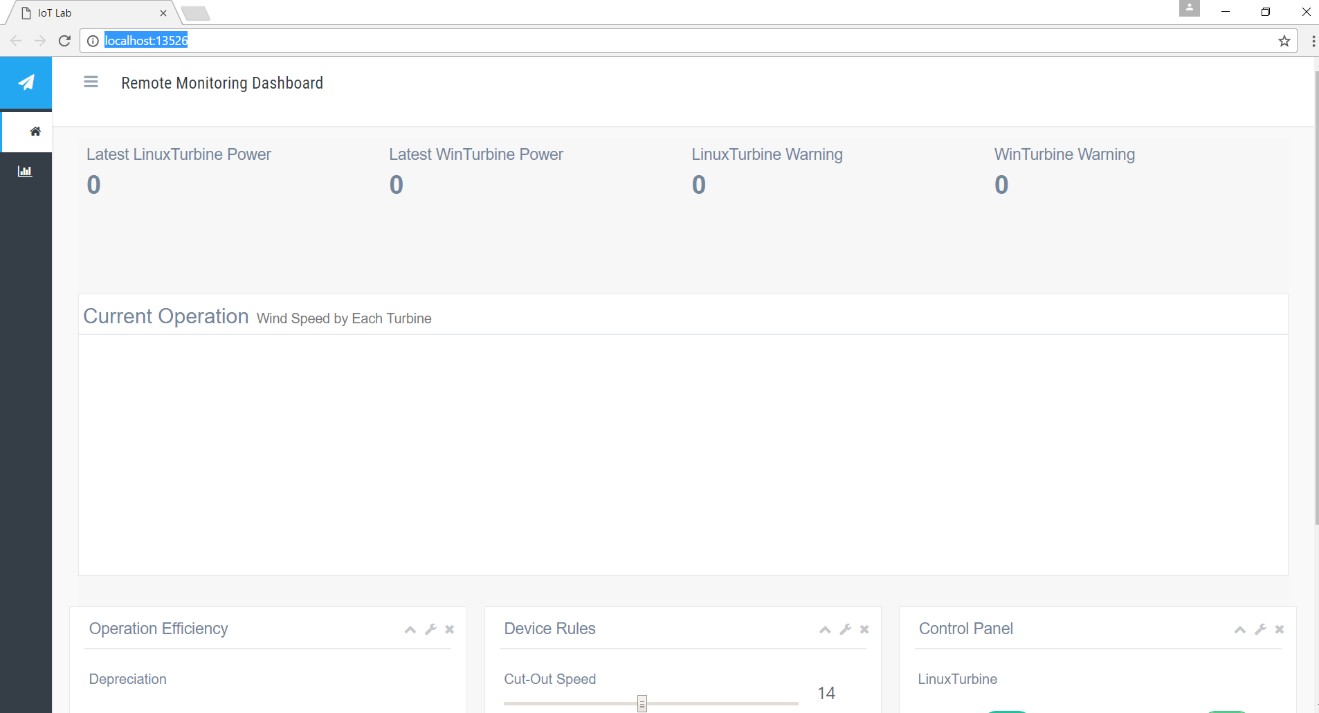
}

}

* View the java script of layout file (Views/Shared/\_layout.cshtml)

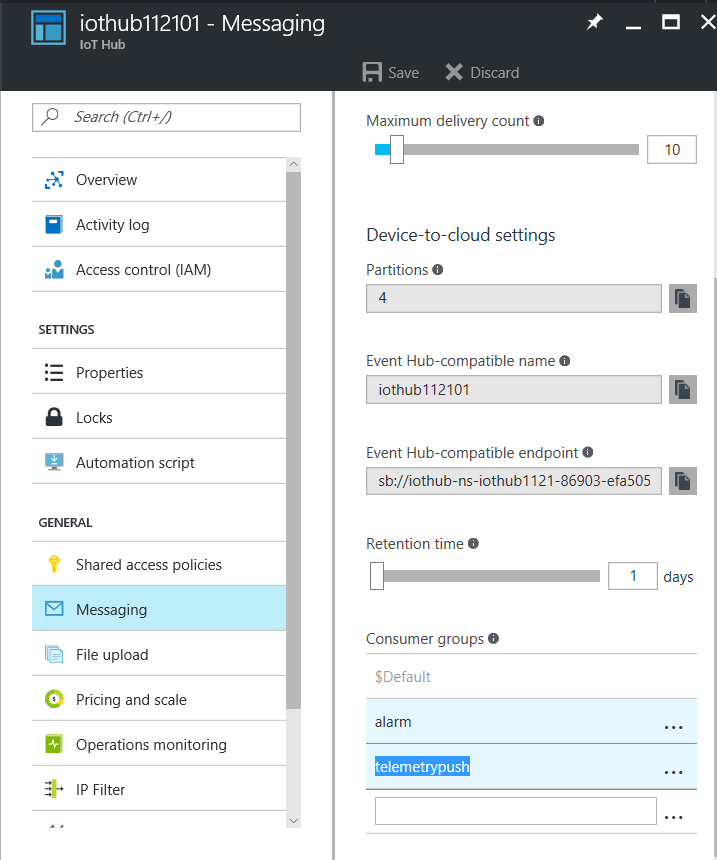


* Build and run the project. For now, it is ready for the real time data. Next, let’s implement the Event Processor Host.



## Step 2: Build the Event Processor Host to push the telemetry data

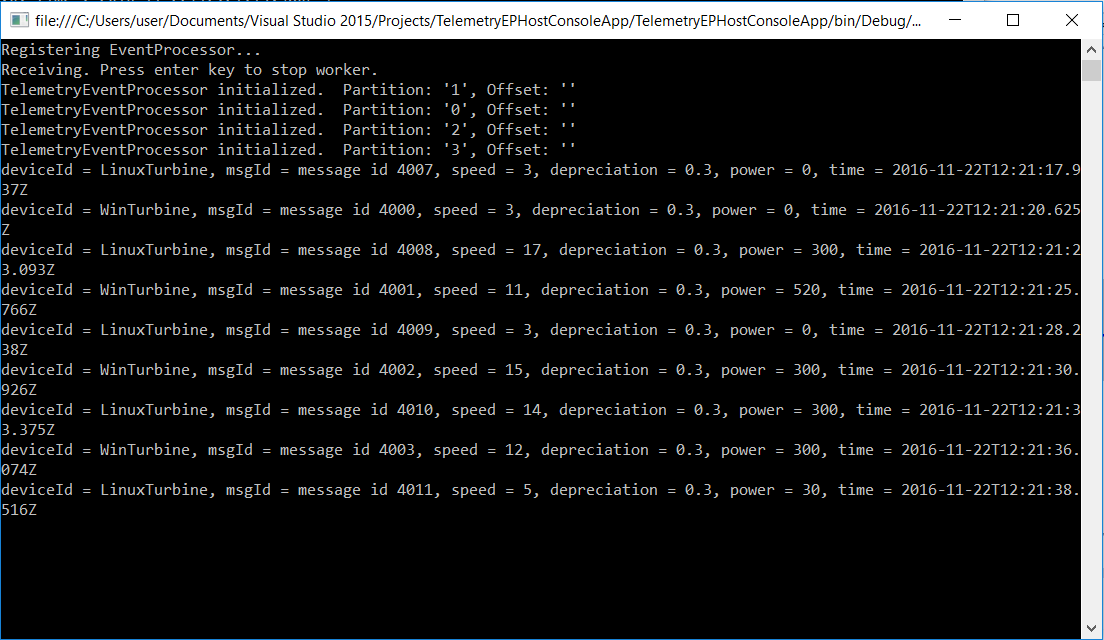
* Unzip the **TelemetryProcessor.zip** and open the solution in VS.
* Confirm the Consumer groups of IoT Hub for Event Processor Host as **telemetrypush**.



* Update the **App.config**. Change the appSettings as the following.
  + **WebServer.isProduction**: 0 means it will be connected to local host; 1 means it will be connected to online web.
  + **WebServer.Localhost**: the url of local host.
  + **WebServer.Production**: the url of online website.
  + **IoTHub.ConnectionString**: the connection string of IoT Hub
  + **StorageAccount.Name**: the name of Storage Account
  + **StorageAccount.Key**: the key of Storage Account



* Run the Event Processor Host and check the output of console App



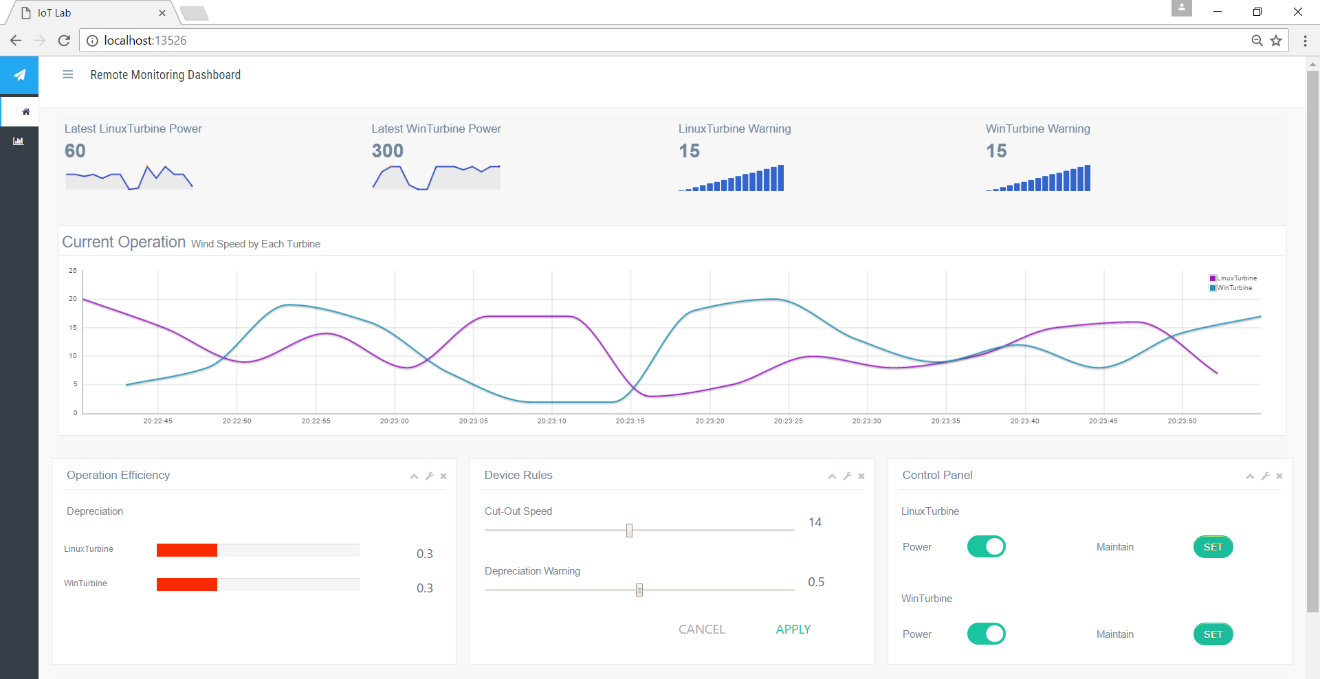
## Step 3: Show the dashboard in the portal

* Run the simulated wind turbines (Linux & Windows)
  + LinuxTurbine - Run C Simple Sample of AMQP in SDK

cd ~/azure-iot-sdk-c/cmake/iotsdk\_linux/serializer/samples/simplesample\_amqp/

./simplesample\_amqp

* + WinTurbine - Run the executable file (SimulatedWindTurbine.exe) or debugging in the VS
* Watch the real time data in the web portal



* *The HOL 10 has been completed. Now we can read the telemetry data from IoT Hub through the Event Processor Host, and feed-in the data to the backend web server by Http API. Finally, we use SignalR technology to communicate between web server and browser to get the real-time dashboard.*