REAL-TIME DATA INTEGRATION PROJECT FROM AN API TO SQL SERVER USING SSIS

ENG. Silva Parraguez Maximo

I. UNDERSTANDING THE DATA AND THE PROJECT

Real-time data will be obtained from the OpenWeatherMap API , which provides real-time weather data such as temperature, humidity, wind speed, etc.

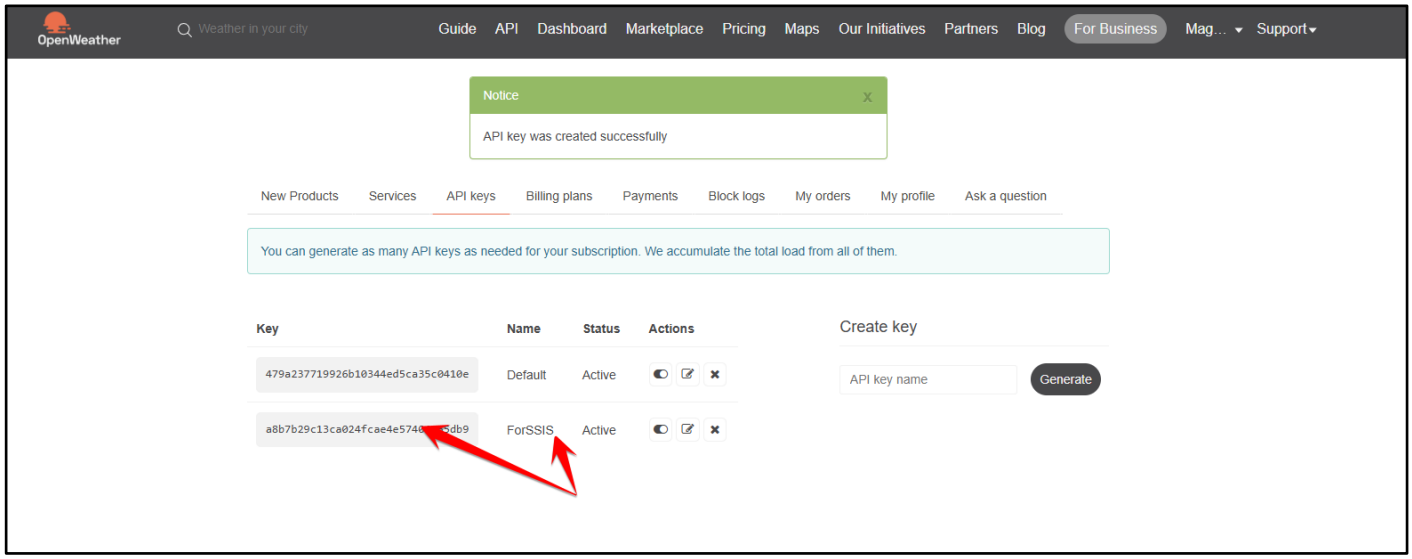
The purpose of this project is to create an ETL process in SSIS that captures, transforms, and loads real-time data from a continuous data source (such as the OpenWeatherMap API) for a specific city (or multiple cities), into a SQL Server database.

II. PROJECT DEVELOPMENT

Step 1: Obtaining an OpenWeatherMap API Key

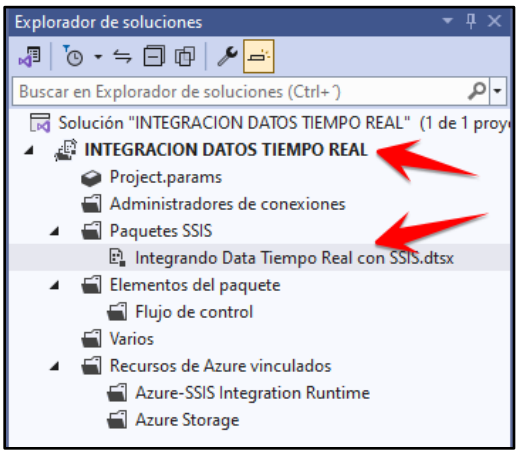
Since I will extract the data from an API that asks for an API Key, I will need to create an account on the Web that will provide it to me. If there were another API that did not ask for that key, for example later, in ***Step 5: Editing the Script with the C# code,*** instead of passing the APIKey to the URL that builds the API, only the API link would be passed directly (For example: https://api.coindesk.com/v1/bpi/currentprice.json) (Free API that provides real-time Bitcoin Price Index (BPI) information).

Once registered on the OpenWeatherMap website (https://openweathermap.org/api), we generate a new API Key that you will use to obtain the Data.



Step 2: Creating Project in SSIS

The project was created with the name: **"REAL-TIME DATA INTEGRATION"** and an SSIS package called: **"Integrating Real-Time Data with SSIS".**

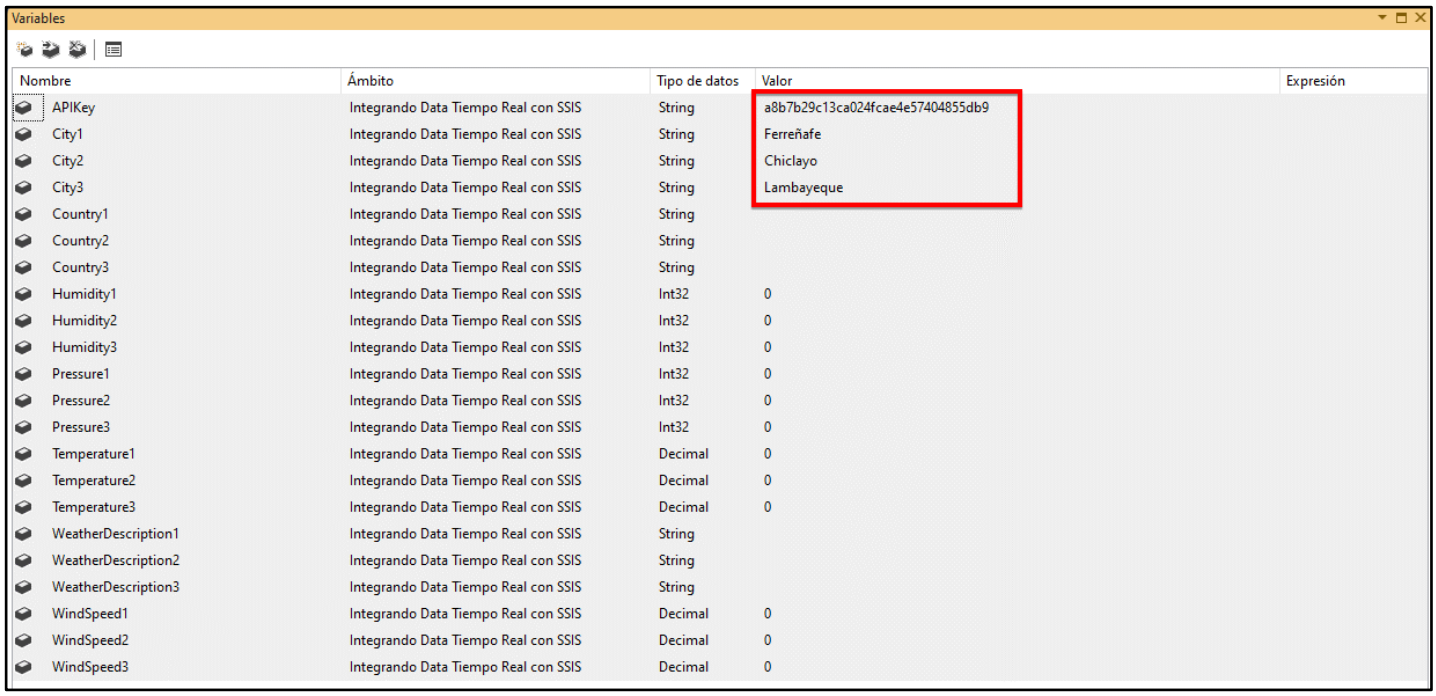


Step 3: Setting Up Variables in SSIS

I will create variables that will be responsible for extracting and storing the data obtained from the API. The variables are:

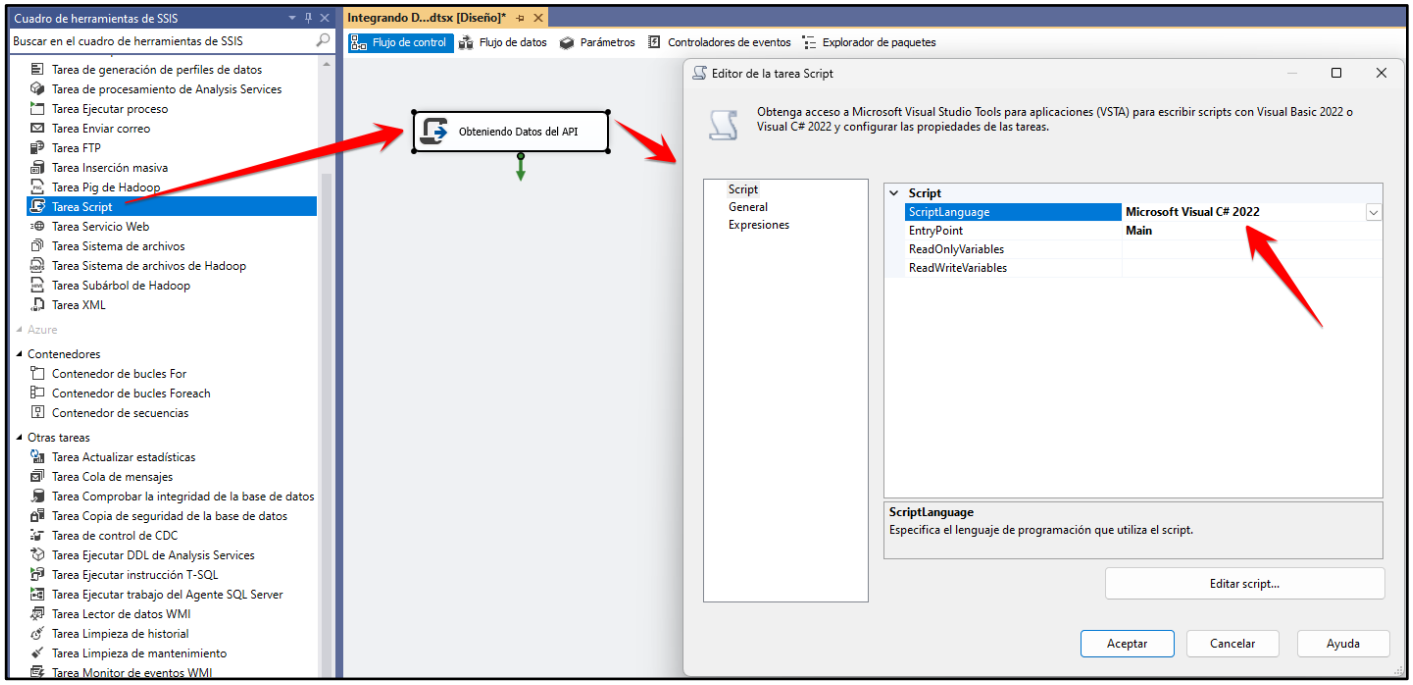
- **APIKey** (String): Will store the OpenWeatherMap API Key that I obtained when creating the account.
- **City1** (String): Will store the name of a specific city (for this project, "Ferreñafe").
- **City2** (String): Will store the name of a specific city (for this project, "Chiclayo").
- **City3** (String): Will store the name of a specific city (for this project, "Lambayeque").
- **Country1** (String): Will store the country to which city 1 belongs.
- **Country2** (String): Will store the country to which city 2 belongs.
- **Country3** (String): Will store the country to which city 3 belongs.
- **Humidity1** (Int): Will store the humidity obtained from city 1.
- **Humidity2** (Int): Will store the humidity obtained from city 2.
- **Humidity3** (Int): Will store the humidity obtained from city 3.
- **Pressure1** (Int): For atmospheric pressure obtained from city 1.
- **Pressure2** (Int): For atmospheric pressure obtained from city 2.
- **Pressure3** (Int): For atmospheric pressure obtained from city 3.
- **Temperature1** (Decimal): Will store the temperature obtained from city 1.
- **Temperature2** (Decimal): Will store the temperature obtained from city 2.
- **Temperature3** (Decimal): Will store the temperature obtained from city 3.
- **WeatherDescription1** (String): For the weather description obtained from city 1.
- **WeatherDescription2** (String): For the weather description obtained from city 2.
- **WeatherDescription3** (String): For the weather description obtained from city 3.
- **WindSpeed1** (Decimal): For the wind speed obtained from city 1.
- **WindSpeed2** (Decimal): For the wind speed obtained from city 2.
- **WindSpeed3** (Decimal): For the wind speed obtained from city 3.

Only APIKey , City1, City2 and City3 have a “Value” assigned , the other variables will be left empty since they will be filled in the Script. If it is necessary to change the cities, they can be changed by modifying this section.



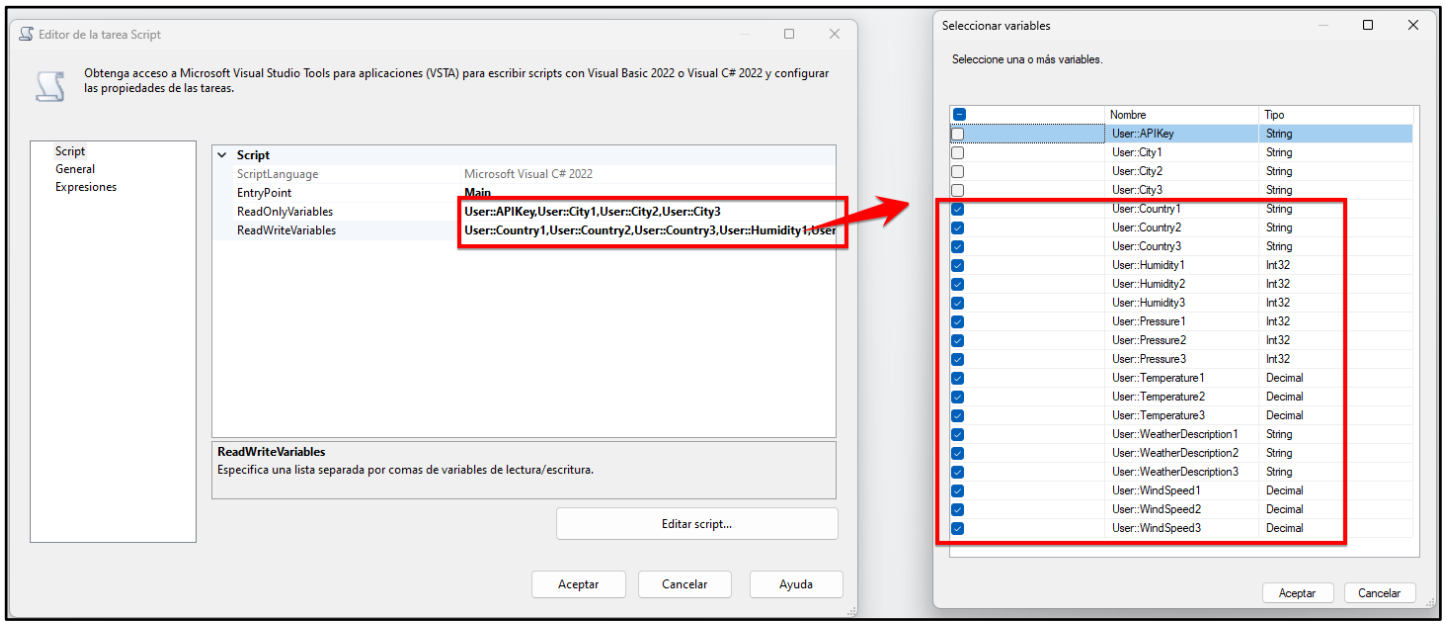
Step 4: Script Task Component.

I will use the “Script Task” component and use C# as the programming language to configure the data retrieval from the API.



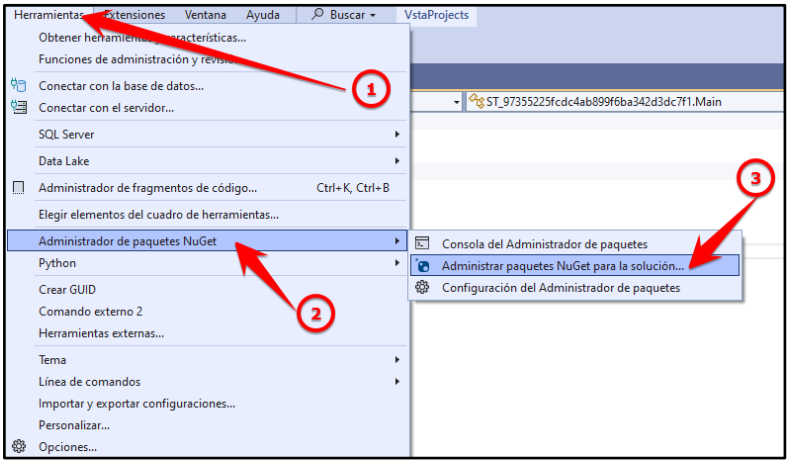
Variables are added to the Script:

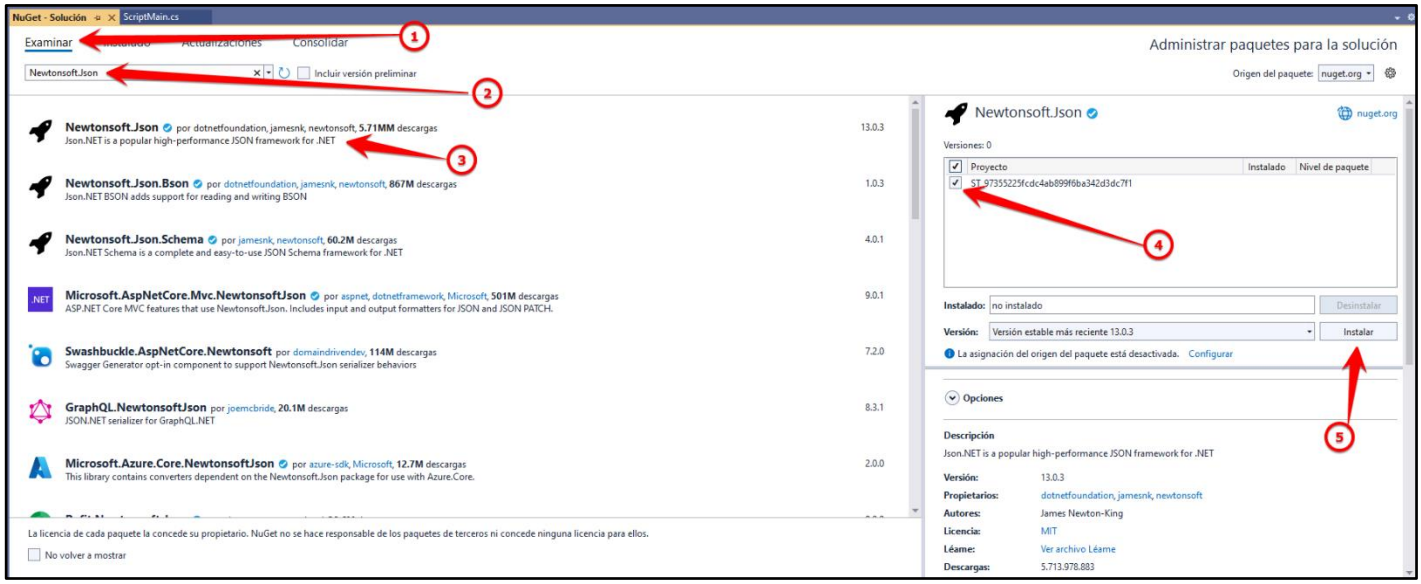
- **ReadOnlyVariables "** tab, I selected the variables User::APIKey , User::City1, User::City2, User::City3. (The variables I set a value for.)
- **ReadWriteVariables "** tab, I selected the other variables (the other variables that were not given any Value).

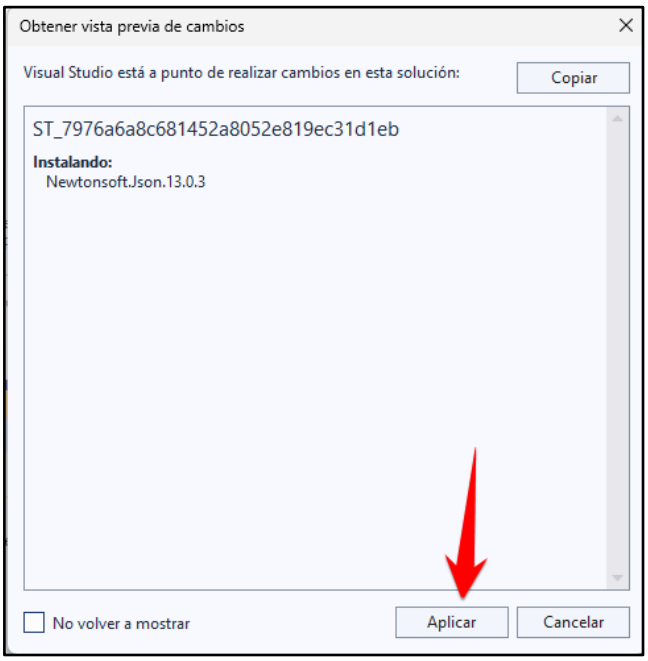


Step 5: Editing the Script with C# code.

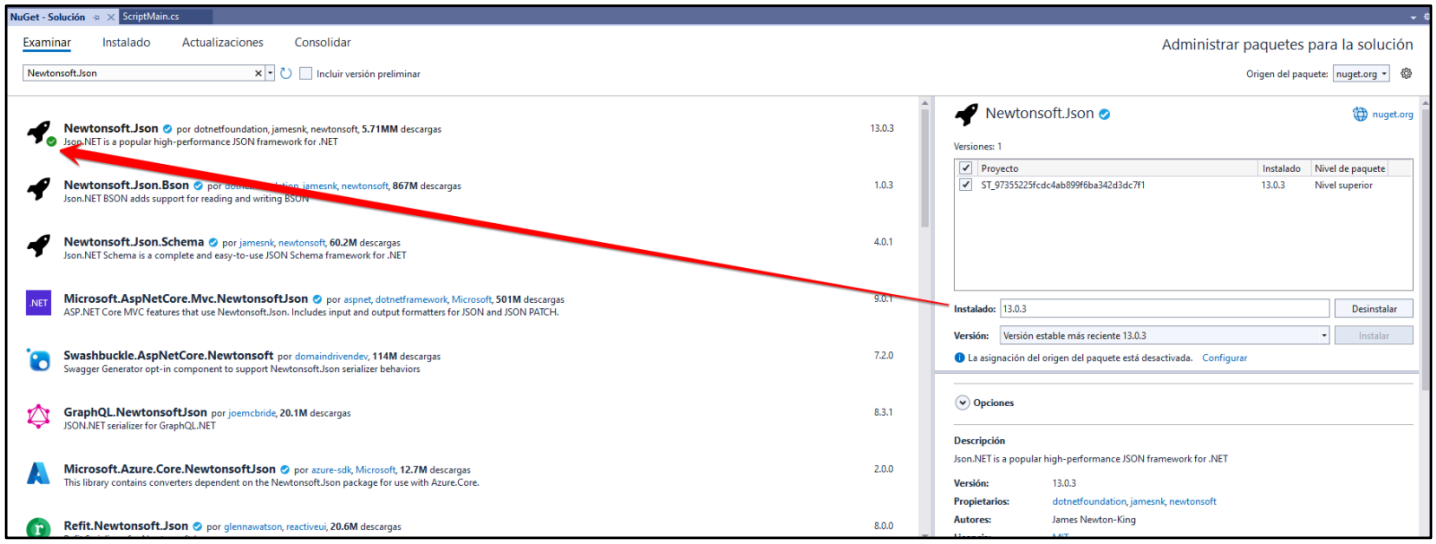
First I will install a library that I will use in the code, the library is **Newtonsoft.Json :**



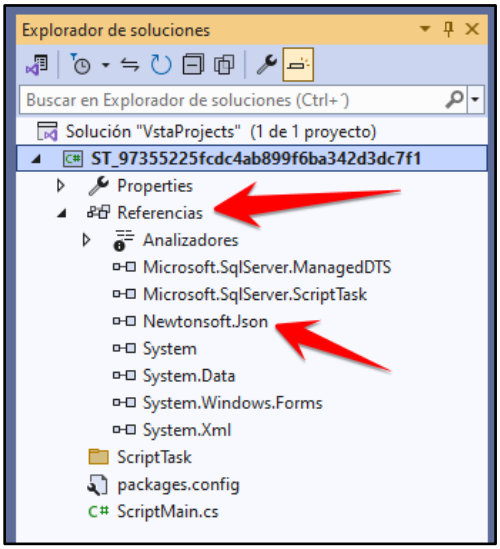




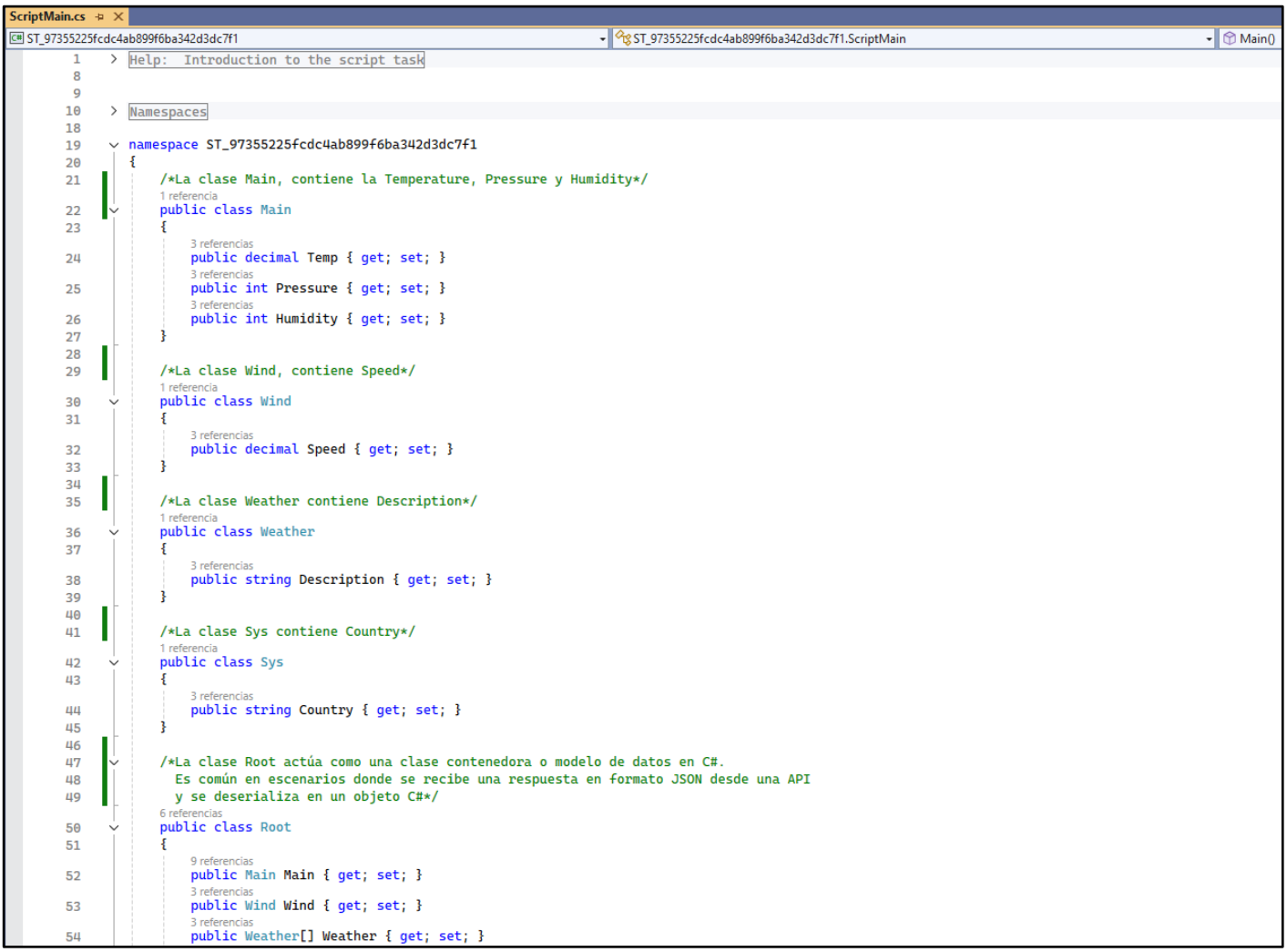
Successfully installed:



We verify that the library that was added appears referenced:



Then I wrote the code that calls the OpenWeatherMap API using the API Key and 3 Cities, we create the classes taking into account the JSON, then it extracts the temperature, humidity, atmospheric pressure, wind speed and weather description of each city provided from the response JSON and saves the values in the variables created in the SSIS.



```
ScriptMain.cs
ST_97355225fcdc4ab899f6ba342d3dc7f1 | ST_97355225fcdc4ab899f6ba342d3dc7f1.ScriptMain | Main()

Help:  Introduction to the script task


Namespaces

namespace ST_97355225fcdc4ab899f6ba342d3dc7f1
{
    /*La clase Main, contiene la Temperature, Pressure y Humidity*/
    1 referencia
    public class Main
    {
        3 referencias
        public decimal Temp { get; set; }
        3 referencias
        public int Pressure { get; set; }
        3 referencias
        public int Humidity { get; set; }
    }

    /*La clase Wind, contiene Speed*/
    1 referencia
    public class Wind
    {
        3 referencias
        public decimal Speed { get; set; }
    }

    /*La clase Weather contiene Description*/
    1 referencia
    public class Weather
    {
        3 referencias
        public string Description { get; set; }
    }

    /*La clase Sys contiene Country*/
    1 referencia
    public class Sys
    {
        3 referencias
        public string Country { get; set; }
    }

    /*La clase Root actúa como una clase contenedora o modelo de datos en C#.
      Es común en escenarios donde se recibe una respuesta en formato JSON desde una API
      y se deserializa en un objeto C#*/
    6 referencias
    public class Root
    {
        9 referencias
        public Main Main { get; set; }
        3 referencias
        public Wind Wind { get; set; }
        3 referencias
        public Weather[] Weather { get; set; }
        3 referencias
        public Sys Sys { get; set; }
    }

    /// <summary> ScriptMain is the entry point class of the script.  Do not change  ...
    [Microsoft.SqlServer.Dts.Tasks.ScriptTask.SSISScriptTaskEntryPointAttribute]
    0 referencias
    public partial class ScriptMain : Microsoft.SqlServer.Dts.Tasks.ScriptTask.VSTARTScriptObjectModelBase
    {
        Help:  Using Integration Services variables and parameters in a script

        Help:  Firing Integration Services events from a script

        Help:  Using Integration Services connection managers in a script


        /// <summary> This method is called when this script task executes in the contro ...
        0 referencias
        public void Main()
        {
            // Obtener variables de SSIS donde asignamos valores, la APIKey y los nombres de las ciudades
            string apiKey = Dts.Variables["User::APIKey"].Value.ToString();
            string city1 = Dts.Variables["User::City1"].Value.ToString();
            string city2 = Dts.Variables["User::City2"].Value.ToString();
            string city3 = Dts.Variables["User::City3"].Value.ToString();
```

```
137
138             // Construir la URL de la API que hace llamado a cada ciudad con el mismo APIKey
139             string url1 = $"http://api.openweathermap.org/data/2.5/weather?q={city1}&appid={apiKey}&units=metric";
140             string url2 = $"http://api.openweathermap.org/data/2.5/weather?q={city2}&appid={apiKey}&units=metric";
141             string url3 = $"http://api.openweathermap.org/data/2.5/weather?q={city3}&appid={apiKey}&units=metric";
142
143             //Try Catch para controlar errores
144             try
145             {
146                 /*Crea una instancia de la clase WebClient para realizar solicitudes HTTP (Descargar archivos de una API o un archivo de Internet)*/
147                 using (WebClient wc = new WebClient())
148                 {
149
150                     //Descarga datos en formato JSON desde una URL y convierte ese JSON  en un Objeto C# para cada ciudad.
151                     string json1 = wc.DownloadString(url1);
152                     Root root1 = JsonConvert.DeserializeObject<Root>(json1);
153
154                     string json2 = wc.DownloadString(url2);
155                     Root root2 = JsonConvert.DeserializeObject<Root>(json2);
156
157                     string json3 = wc.DownloadString(url3);
158                     Root root3 = JsonConvert.DeserializeObject<Root>(json3);
159
160
161                     // Extraer datos relevantes de Ciudad1
162                     string country1 = root1.Sys.Country.ToString();
163                     decimal temperature1 = root1.Main.Temp;
164                     int humidity1 = root1.Main.Humidity;
165                     int pressure1 = root1.Main.Pressure;
166                     decimal windSpeed1 = root1.Wind.Speed;
167                     string weatherDescription1 = root1.Weather[0].Description.ToString();
168
169
170                     // Guardar los datos en variables de SSIS de Ciudad1
171                     Dts.Variables["User::Country1"].Value = country1;
172                     Dts.Variables["User::Temperature1"].Value = temperature1;
173                     Dts.Variables["User::Humidity1"].Value = humidity1;
174                     Dts.Variables["User::Pressure1"].Value = pressure1;
175                     Dts.Variables["User::WindSpeed1"].Value = windSpeed1;
176                     Dts.Variables["User::WeatherDescription1"].Value = weatherDescription1;
177                     Dts.Variables["User::WeatherDescription1"].Value = weatherDescription1;
178
179                     // Extraer datos relevantes de Ciudad2
180                     string country2 = root2.Sys.Country.ToString();
181                     decimal temperature2 = root2.Main.Temp;
182                     int humidity2 = root2.Main.Humidity;
183                     int pressure2 = root2.Main.Pressure;
184                     decimal windSpeed2 = root2.Wind.Speed;
185                     string weatherDescription2 = root2.Weather[0].Description.ToString();
```

```
186
187                     // Guardar los datos en variables de SSIS de Ciudad2
188                     Dts.Variables["User::Country2"].Value = country2;
189                     Dts.Variables["User::Temperature2"].Value = temperature2;
190                     Dts.Variables["User::Humidity2"].Value = humidity2;
191                     Dts.Variables["User::Pressure2"].Value = pressure2;
192                     Dts.Variables["User::WindSpeed2"].Value = windSpeed2;
193                     Dts.Variables["User::WeatherDescription2"].Value = weatherDescription2;
194
195
196                     // Extraer datos relevantes de Ciudad3
197                     string country3 = root3.Sys.Country.ToString();
198                     decimal temperature3 = root3.Main.Temp;
199                     int humidity3 = root3.Main.Humidity;
200                     int pressure3 = root3.Main.Pressure;
201                     decimal windSpeed3 = root3.Wind.Speed;
202                     string weatherDescription3 = root3.Weather[0].Description.ToString();
203
204                     // Guardar los datos en variables de SSIS de Ciudad1
205                     Dts.Variables["User::Country3"].Value = country3;
206                     Dts.Variables["User::Temperature3"].Value = temperature3;
207                     Dts.Variables["User::Humidity3"].Value = humidity3;
208                     Dts.Variables["User::Pressure3"].Value = pressure3;
209                     Dts.Variables["User::WindSpeed3"].Value = windSpeed3;
210                     Dts.Variables["User::WeatherDescription3"].Value = weatherDescription3;
211
```

```
221                 }
222                 Dts.TaskResult = (int)ScriptResults.Success;
223             }
224             catch (Exception ex)
225             {
226                 // Manejar errores
227                 Dts.Events.FireError(0, "Script Task", ex.Message, string.Empty, 0);
228                 Dts.TaskResult = (int)ScriptResults.Failure;
229             }
230
231             Dts.TaskResult = (int)ScriptResults.Success;
232         }
233
234         #region ScriptResults declaration
235         /// <summary>
236         /// This enum provides a convenient shorthand within the scope of this class for setting the
237         /// result of the script.
238         ///
239         /// This code was generated automatically.
240         /// </summary>
            3 referencias
241         enum ScriptResults
242         {
243             Success = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Success,
244             Failure = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Failure
245         };
246         #endregion
247
248     }
249 }
```

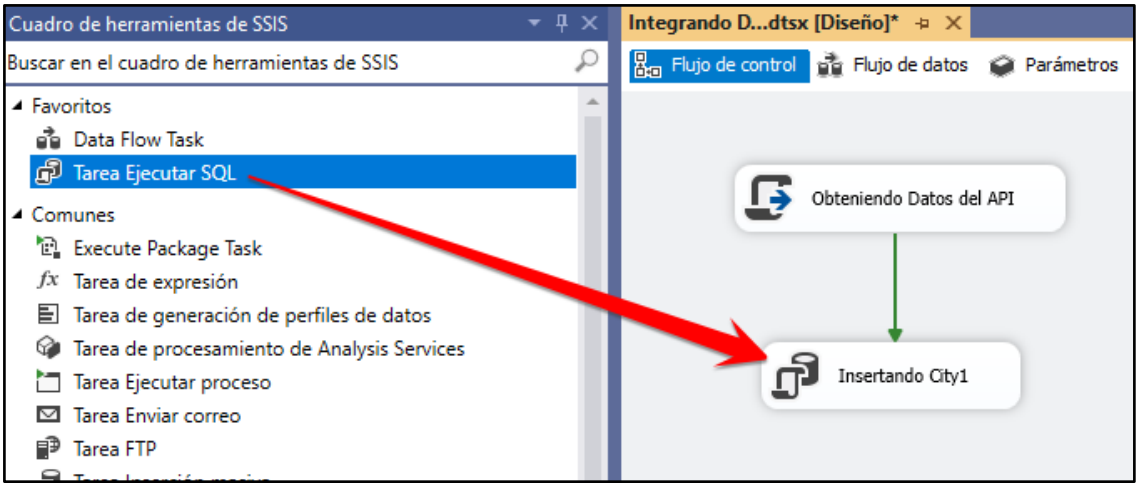
Step 6: Creating Table to store the Data.

I will create a database and a table in SQL Server where the data extracted from the API will be stored:

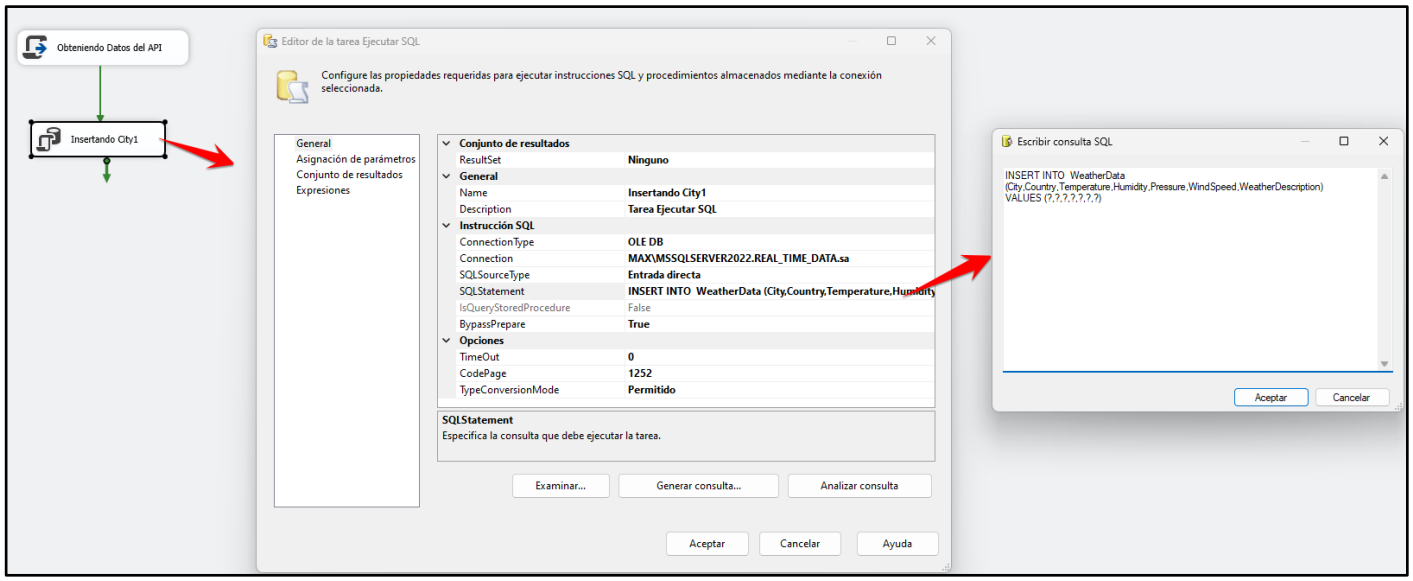
```sql
/*CREANDO LA BASE DE DATOS REAL_TIME_DATA*/

USE master
GO

IF EXISTS(SELECT NAME FROM SYS.databases WHERE NAME='REAL_TIME_DATA')
BEGIN
    DROP DATABASE REAL_TIME_DATA
END
GO

CREATE DATABASE REAL_TIME_DATA
GO

USE REAL_TIME_DATA
GO


SET ANSI_NULLS ON /*CONTROL Y MANEJO CORRECTO DE LOS VALORES NULL EN LAS COMPARACIONES*/
GO

SET QUOTED_IDENTIFIER ON /*PERMITE NOMBRES DE OBJETOS MAS FLEXIBLES Y EVITA PROBLEMAS CON PALABRAS RESERVADAS*/
GO

/*CREANDO TABLA DONDE SE GUARDARAN LOS DATOS DEL API*/
CREATE TABLE WeatherData (
    ID INT IDENTITY(1,1) PRIMARY KEY,
    City NVARCHAR(100),
    Country NVARCHAR(100),
    Temperature decimal(10,2),
    Humidity INT,
    Pressure INT,
    WindSpeed decimal(10,2),
    WeatherDescription NVARCHAR(200),
    Timestamp DATETIME DEFAULT GETDATE()
);
```

Step 7: Creating an Execute SQL Task to insert the data into the Database.

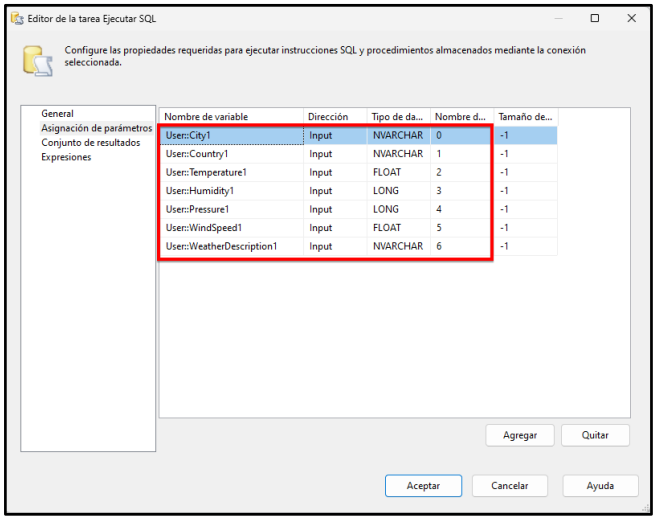
Now for the process of loading JSON data to a table in SQL Server I used the Execute SQL Task component to insert the data corresponding to each City.



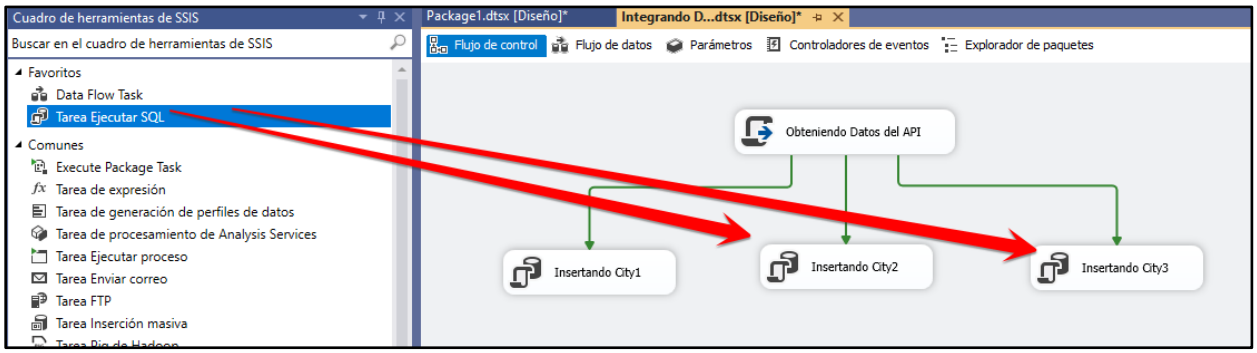
I added the connection to the Database and the SQL command where I will pass the variables.

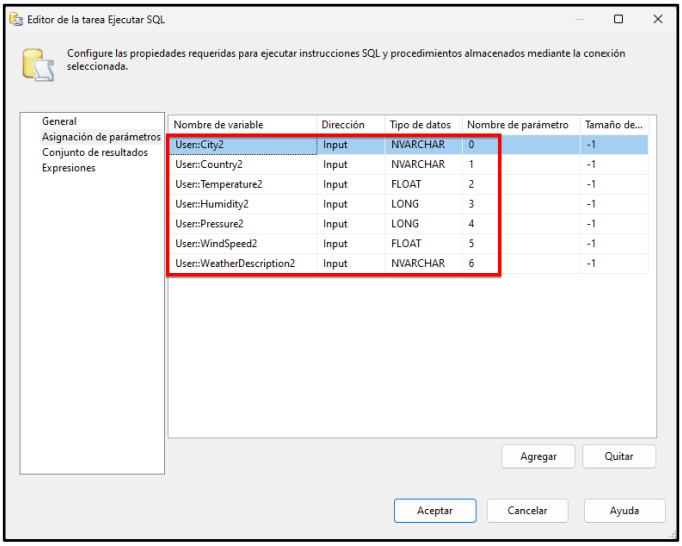


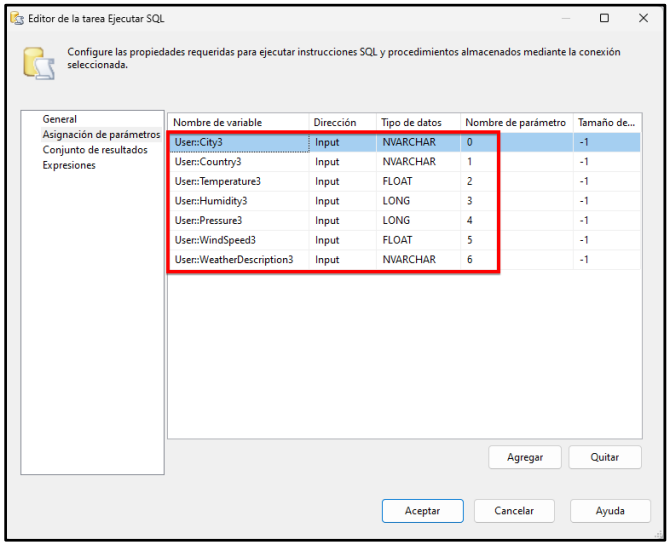
In Variable Assignment we place everything related to data obtained for city 1.



The same process was done for the other cities:

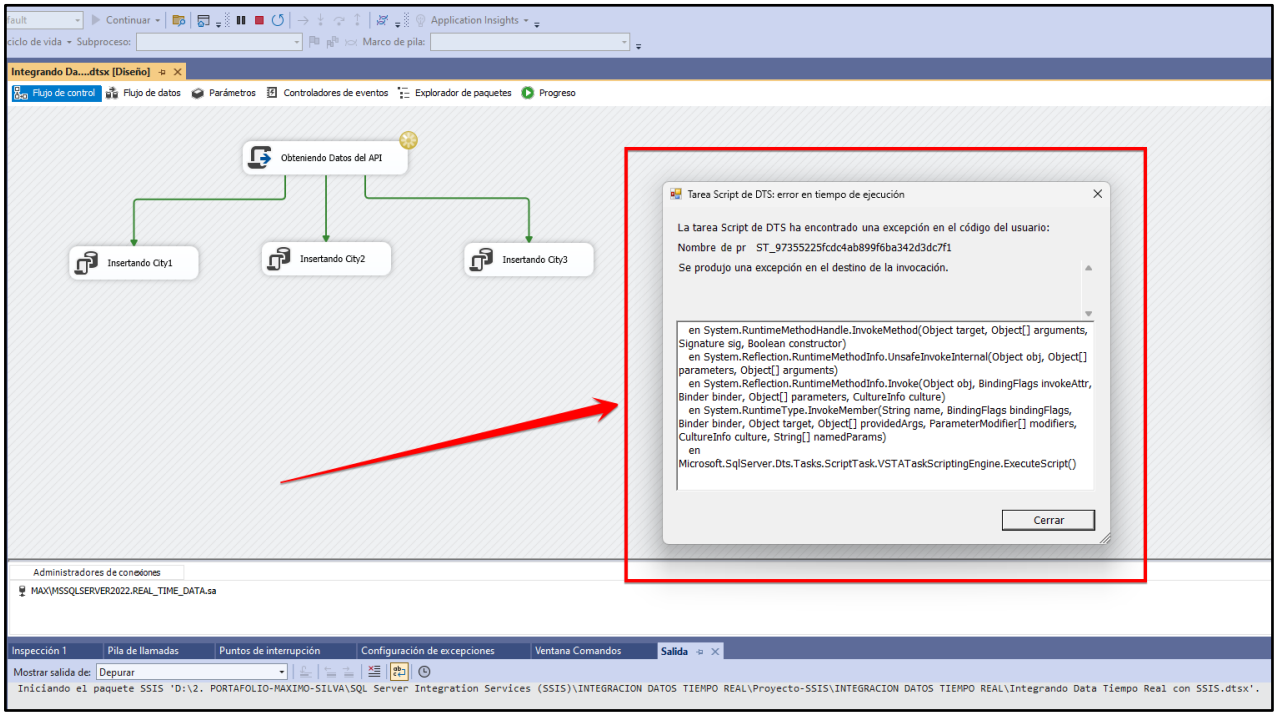


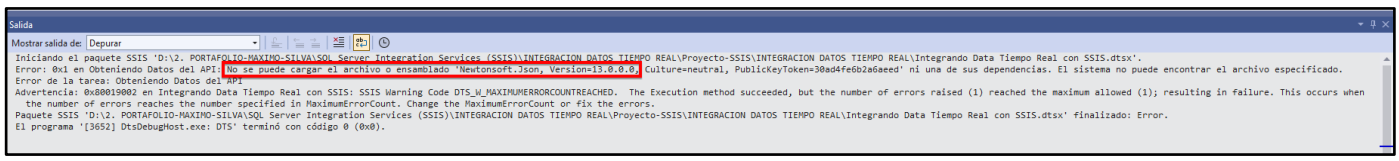




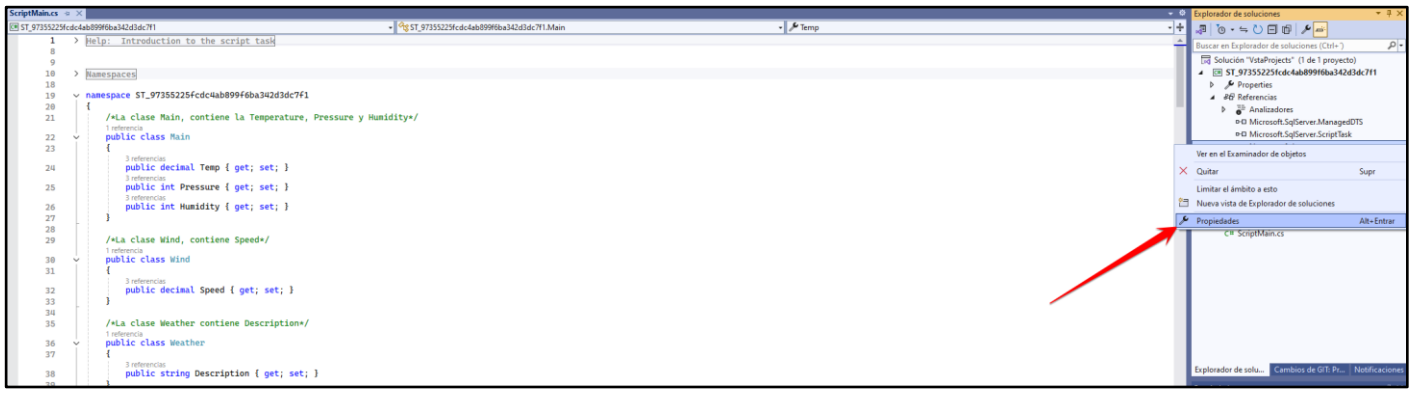
Step 8: Execution Error and Solution to the Error.

When starting the project execution **I got the following error** :

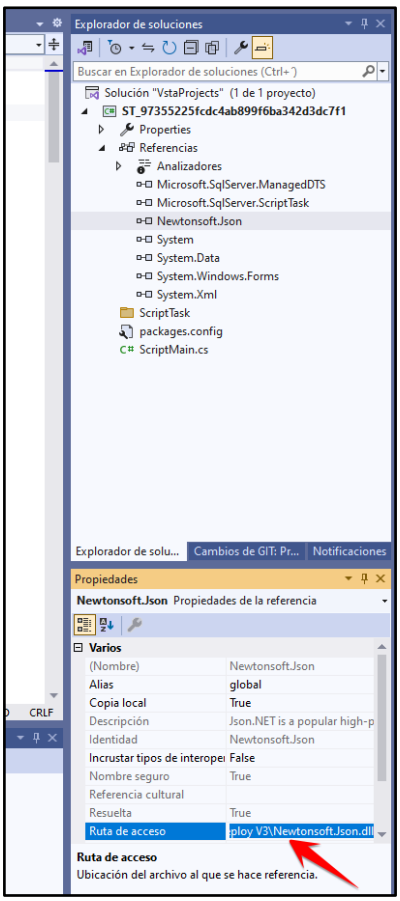




I was investigating and this error indicates that the reference to the **Newtonsoft.Json library** is not registered in the **GAC (Global Assembly Cache)** so it needs to be registered so that it can be located by the project and thus be used, for this we open again where the C# code was written to search for the path where the downloaded library is:



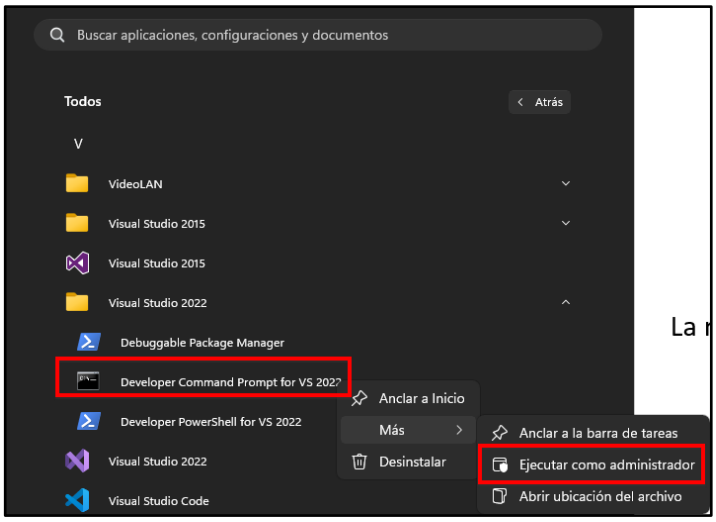
Identify the Library Path and copy it:



The route where it was downloaded was:

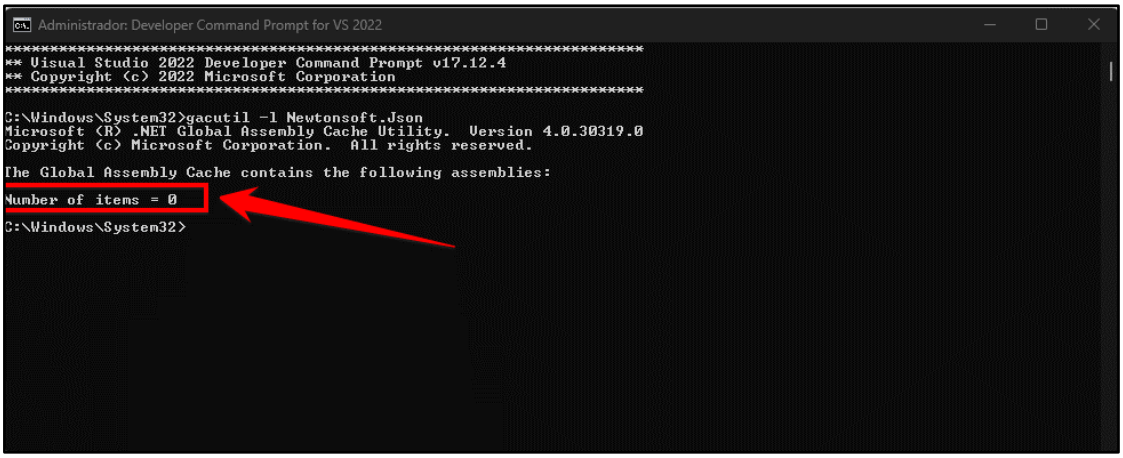
C:\Program Files\IIS\Microsoft Web Deploy V3\Newtonsoft.Json.dll

Then I opened **Developer Command Prompt for VS 2022** and ran it as Administrator.



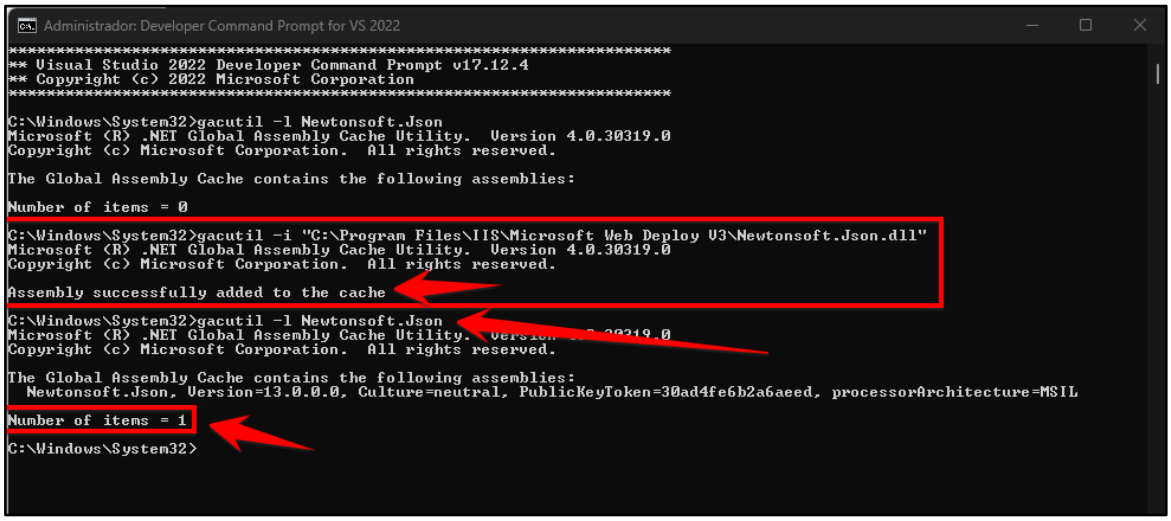
In the command screen we verify that the Library is not registered:

gacutil -l Newtonsoft.Json



The number 0 indicates that there are no libraries registered in the **GAC**, to reference it **globally** (Not just for the project, that's why **gacutil** is used):

gacutil -i "C:\Program Files\IIS\Microsoft Web Deploy V3\Newtonsoft.Json.dll"



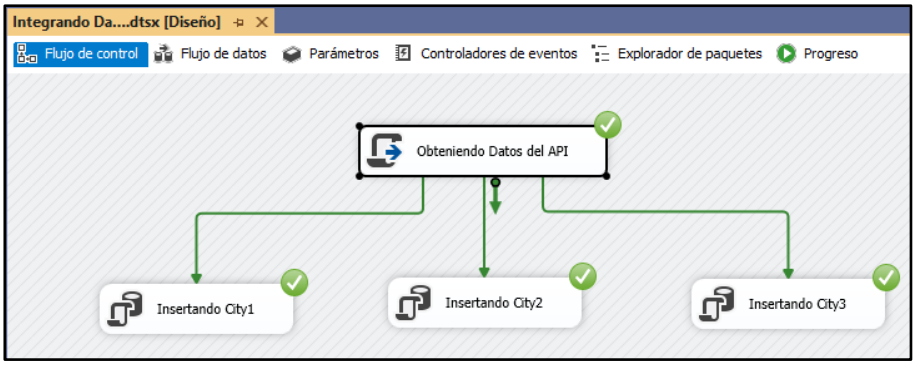
When checking the library again, the number 1 appears, indicating that a library is registered, in this case Newtonsoft.Json . **The problem should now be solved.**

If for some reason we need to delete reference to the library, it is done with the following command:

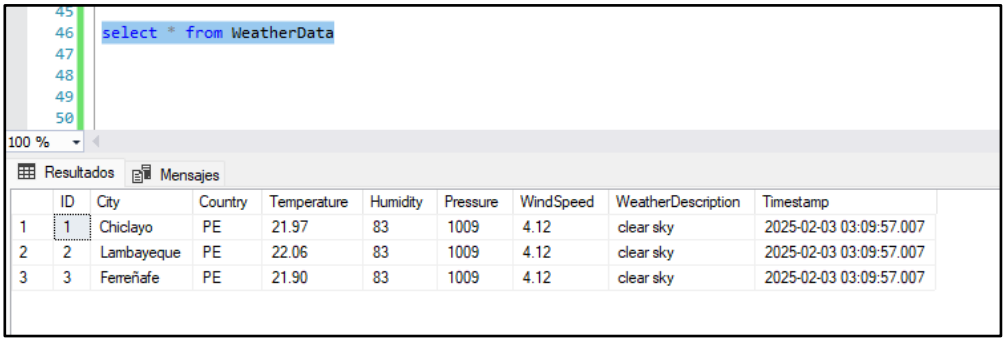
gacutil -u Newtonsoft.Json

Step 9: Running the project.

Once the error is resolved, when running the project, it does so without any problem:



I verify that the data was recorded in the Database:



```
select * from WeatherData
```

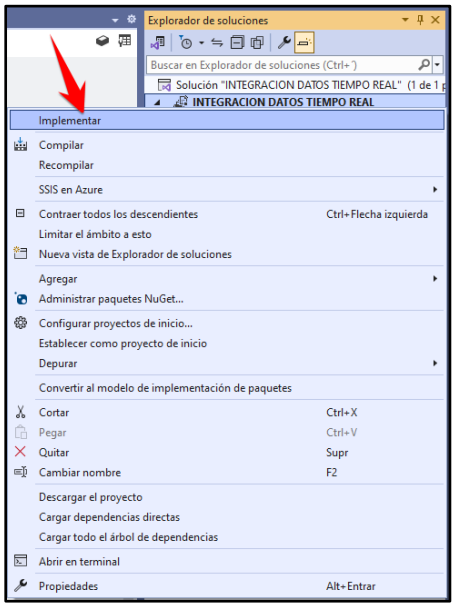
100 %

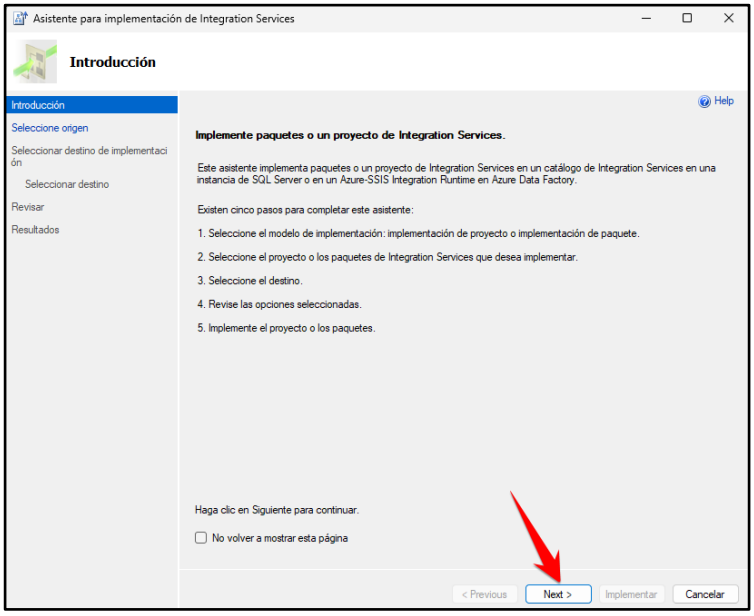
Resultados | Mensajes

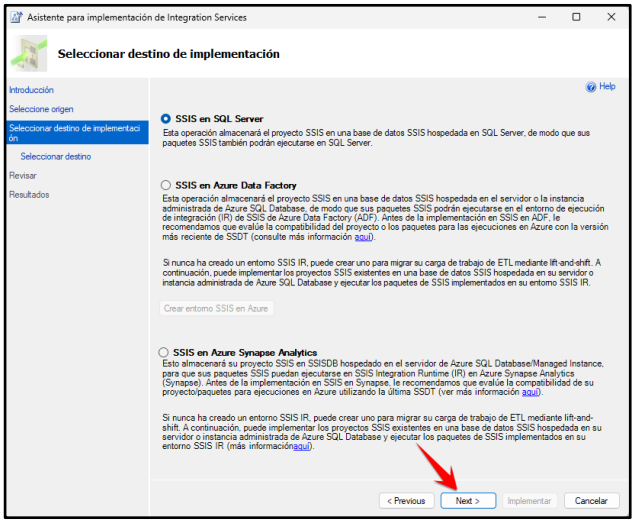
	ID	City	Country	Temperature	Humidity	Pressure	WindSpeed	WeatherDescription	Timestamp
1	1	Chiclayo	PE	21.97	83	1009	4.12	clear sky	2025-02-03 03:09:57.007
2	2	Lambayeque	PE	22.06	83	1009	4.12	clear sky	2025-02-03 03:09:57.007
3	3	Ferreñafe	PE	21.90	83	1009	4.12	clear sky	2025-02-03 03:09:57.007

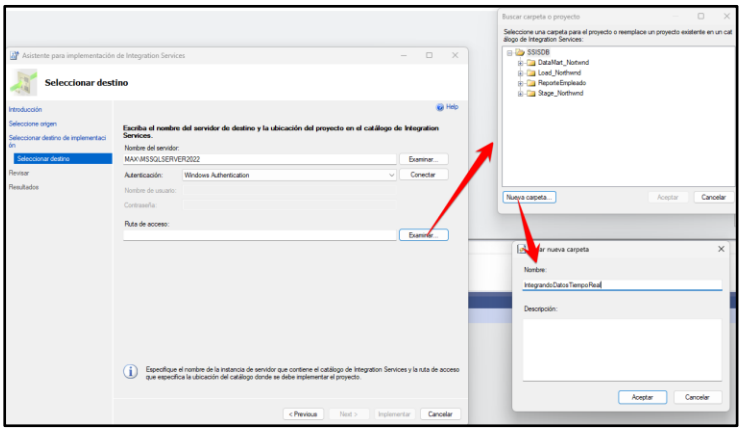
Step 10: Implementing the Integration project Services .

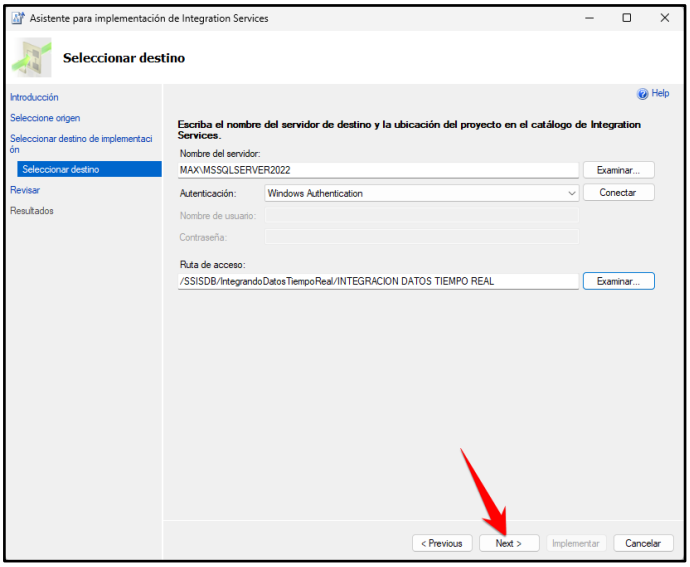
Once we have the project, I will start with the deployment so that it appears in SQL Server and I can create a Scheduled Task in the SQL Server Agent, for this I implemented it:

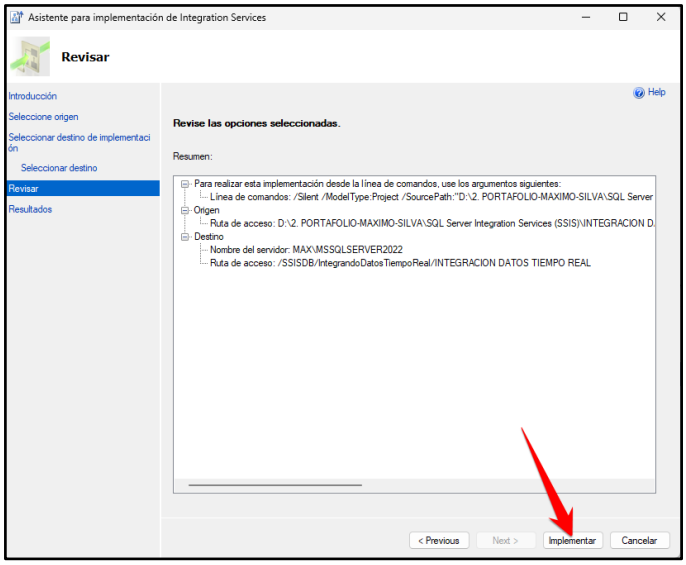

Explorador de soluciones
Buscar en Explorador de soluciones (Ctrl+ ')
Solución "INTEGRACION DATOS TIEMPO REAL" (1 de 1 p
INTEGRACION DATOS TIEMPO REAL
Implementar
Compilar
Recompilar
SSIS en Azure
Contraer todos los descendientes Ctrl+Flecha izquierda
Limitar el ámbito a esto
Nueva vista de Explorador de soluciones
Agregar
Administrar paquetes NuGet...
Configurar proyectos de inicio...
Establecer como proyecto de inicio
Depurar
Convertir al modelo de implementación de paquetes
Cortar Ctrl+X
Pegar Ctrl+V
Quitar Supr
Cambiar nombre F2
Descargar el proyecto
Cargar dependencias directas
Cargar todo el árbol de dependencias
Abrir en terminal
Propiedades Alt+Entrar

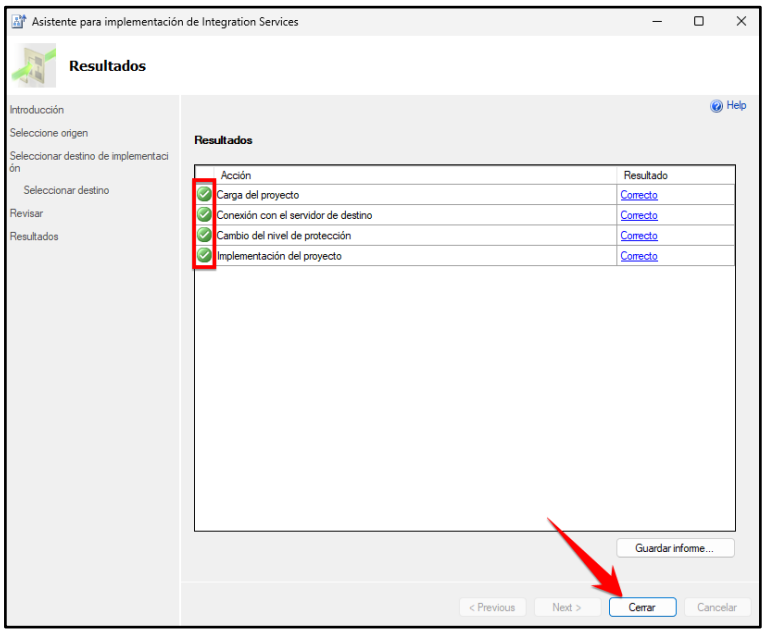

Asistente para implementación de Integration Services
Introducción
Introducción
Seleccione origen
Seleccionar destino de implementación
Seleccionar destino
Revisar
Resultados
Help
Implemente paquetes o un proyecto de Integration Services.
Este asistente implementa paquetes o un proyecto de Integration Services en un catálogo de Integration Services en una instancia de SQL Server o en un Azure-SSIS Integration Runtime en Azure Data Factory.
Existen cinco pasos para completar este asistente:
1. Seleccione el modelo de implementación: implementación de proyecto o implementación de paquete.
2. Seleccione el proyecto o los paquetes de Integration Services que desea implementar.
3. Seleccione el destino.
4. Revise las opciones seleccionadas.
5. Implemente el proyecto o los paquetes.
Haga clic en Siguiente para continuar.
No volver a mostrar esta página
< Previous
Next >
Implementar
Cancelar


Asistente para implementación de Integration Services
Seleccionar destino de implementación
Introducción
Seleccione origen
Seleccionar destino de implementación
Seleccionar destino
Revisar
Resultados
Help
SSIS en SQL Server
Esta operación almacenará el proyecto SSIS en una base de datos SSIS hospedada en SQL Server, de modo que sus paquetes SSIS también podrán ejecutarse en SQL Server.
SSIS en Azure Data Factory
Esta operación almacenará el proyecto SSIS en una base de datos SSIS hospedada en el servidor o la instancia administrada de Azure SQL Database, de modo que sus paquetes SSIS podrán ejecutarse en el entorno de ejecución de integración (IR) de SSIS de Azure Data Factory (ADF). Antes de la implementación en SSIS en ADF, le recomendamos que evalúe la compatibilidad del proyecto o los paquetes para las ejecuciones en Azure con la versión más reciente de SSDT (consulte más información aquí).
Si nunca ha creado un entorno SSIS IR, puede crear uno para migrar su carga de trabajo de ETL mediante lift-and-shift. A continuación, puede implementar los proyectos SSIS existentes en una base de datos SSIS hospedada en su servidor o instancia administrada de Azure SQL Database y ejecutar los paquetes de SSIS implementados en su entorno SSIS IR.
Crear entorno SSIS en Azure
SSIS en Azure Synapse Analytics
Esto almacenará su proyecto SSIS en SSISDB hospedado en el servidor de Azure SQL Database/Managed Instance, para que sus paquetes SSIS puedan ejecutarse en SSIS Integration Runtime (IR) en Azure Synapse Analytics (Synapse). Antes de la implementación en SSIS en Synapse, le recomendamos que evalúe la compatibilidad de su proyecto/paquetes para ejecuciones en Azure utilizando la última SSDT (ver más información aquí).
Si nunca ha creado un entorno SSIS IR, puede crear uno para migrar su carga de trabajo de ETL mediante lift-and-shift. A continuación, puede implementar los proyectos SSIS existentes en una base de datos SSIS hospedada en su servidor o instancia administrada de Azure SQL Database y ejecutar los paquetes de SSIS implementados en su entorno SSIS IR (más información aquí).
< Previous
Next >
Implementar
Cancelar

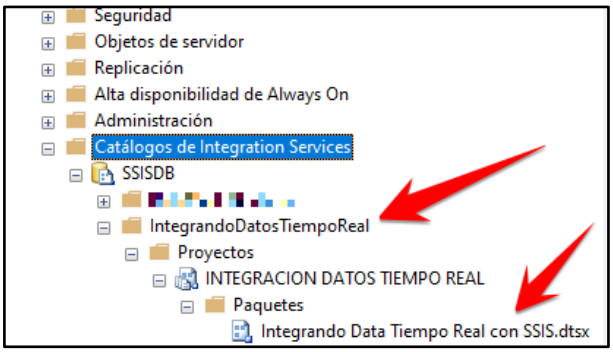

Seleccionar destino
Escriba el nombre del servidor de destino y la ubicación del proyecto en el catálogo de Integration Services.
Nombre del servidor:
MAX\MSSQLSERVER2022
Autenticación:
Windows Authentication
Nombre de usuario:
Contraseña:
Ruta de acceso:
Examinar...
Conectar
Seleccione una carpeta para el proyecto o reemplace un proyecto existente en un catálogo de Integration Services:
SSISDB
DataMart_Nothwind
Load_Nothwind
ReporteEmpleado
Stage_Nothwind
Nueva carpeta...
Aceptar
Cancelar
Crear nueva carpeta
Nombre:
IntegrandoDatosTiempoReal
Descripción:
Especifique el nombre de la instancia de servidor que contiene el catálogo de Integration Services y la ruta de acceso que especifica la ubicación del catálogo donde se debe implementar el proyecto.


Asistente para implementación de Integration Services
Seleccionar destino
Introducción
Seleccione origen
Seleccionar destino de implementación
Seleccionar destino
Revisar
Resultados
Help
Escriba el nombre del servidor de destino y la ubicación del proyecto en el catálogo de Integration Services.
Nombre del servidor:
MAX\MSSQLSERVER2022
Examinar...
Autenticación:
Windows Authentication
Conectar
Nombre de usuario:
Contraseña:
Ruta de acceso:
/SSISDB/IntegrandoDatosTiempoReal/INTEGRACION DATOS TIEMPO REAL
Examinar...
< Previous
Next >
Implementar
Cancelar


Asistente para implementación de Integration Services
Revisar
Introducción
Seleccione origen
Seleccionar destino de implementación
Seleccionar destino
Revisar
Resultados
Help
Revise las opciones seleccionadas.
Resumen:
Para realizar esta implementación desde la línea de comandos, use los argumentos siguientes:
Línea de comandos: /Silent /ModelType:Project /SourcePath:"D:\2. PORTAFOLIO-MAXIMO-SILVA\SQL Server
Origen
Ruta de acceso: D:\2. PORTAFOLIO-MAXIMO-SILVA\SQL Server Integration Services (SSIS)\INTEGRACION D
Destino
Nombre del servidor: MAX\MSSQLSERVER2022
Ruta de acceso: /SSISDB/IntegrandoDatosTiempoReal/INTEGRACION DATOS TIEMPO REAL
< Previous
Next >
Implementar
Cancelar

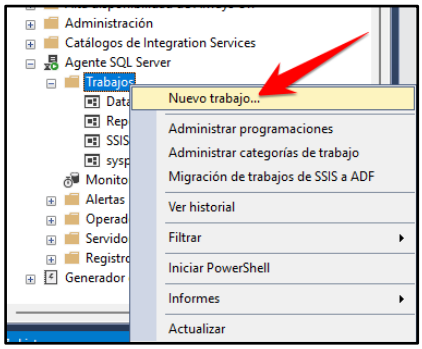


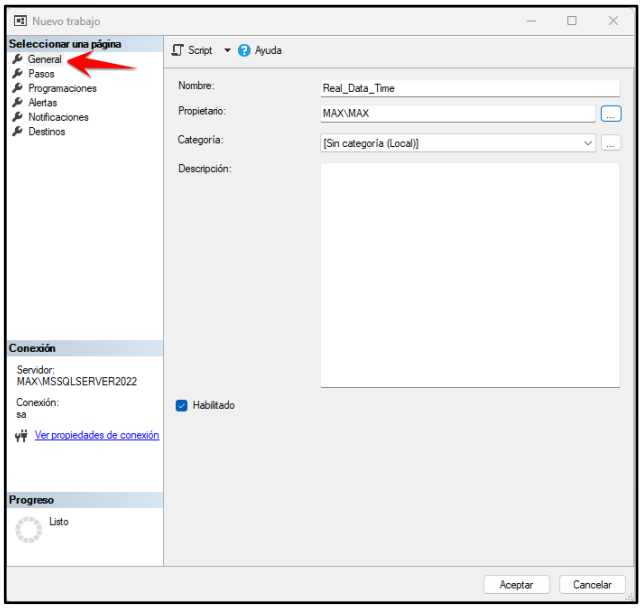
In SQL Server, in the Integration Services Catalogs section (a new catalog had to be created previously), the project now appears:

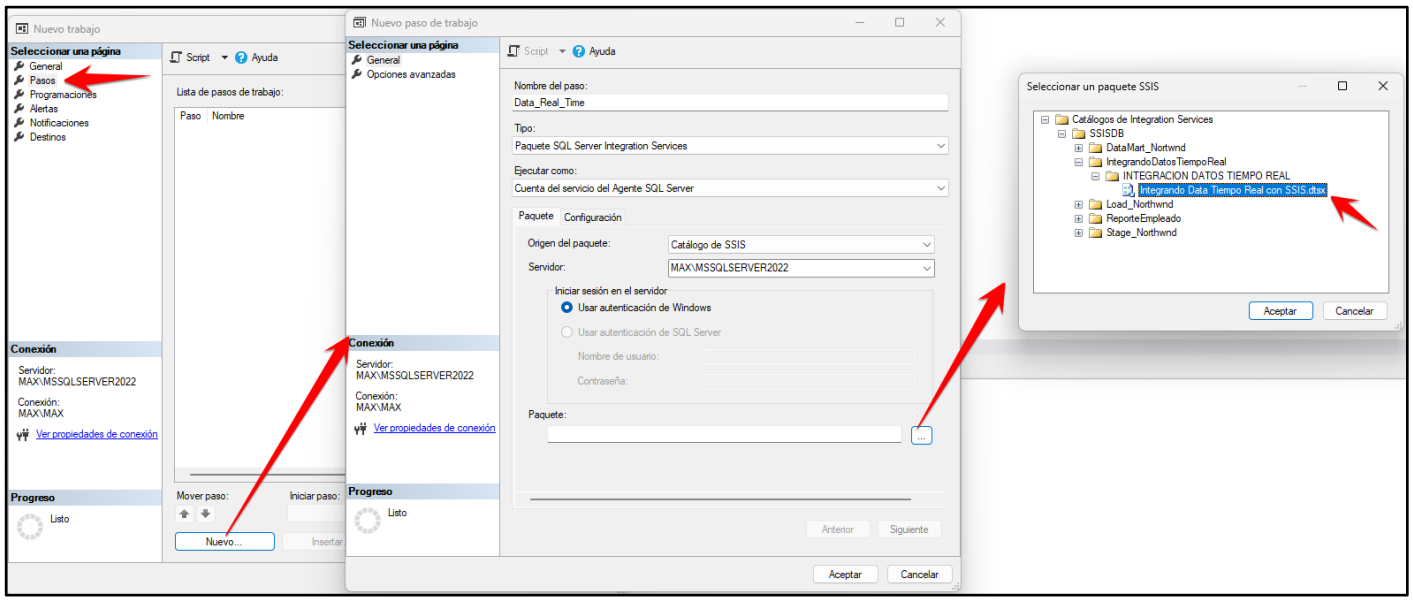


Step 11: Creating an automatic scheduled task with SQL Server Agent.

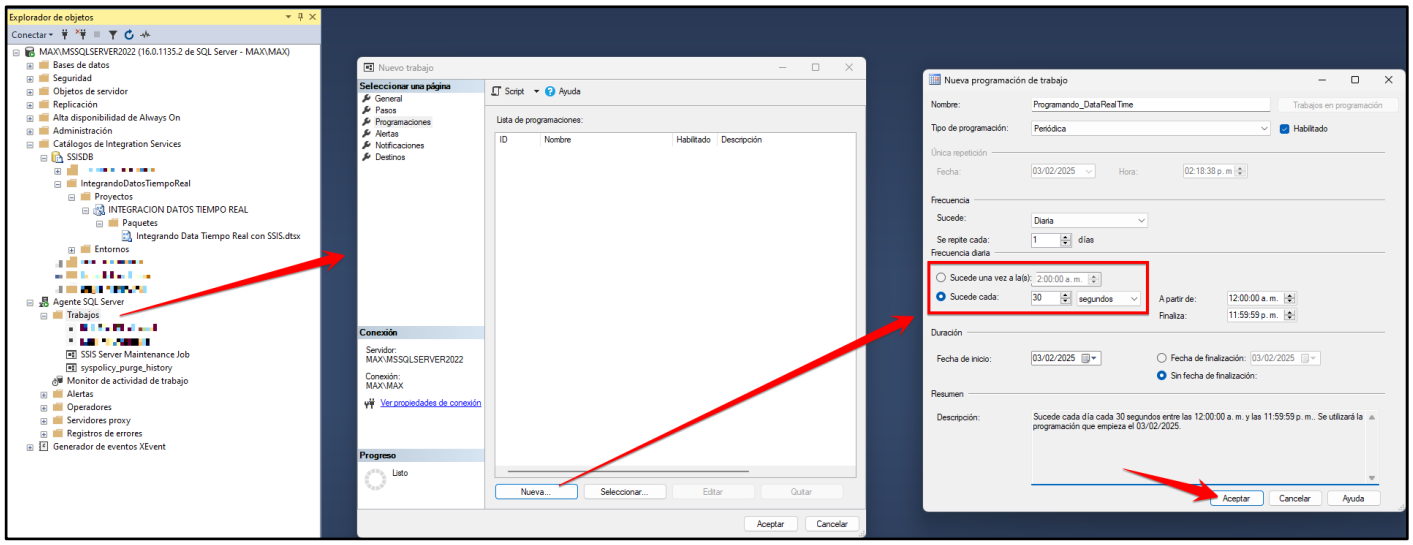
Create a new Job in SQL Server that runs the previously deployed project:

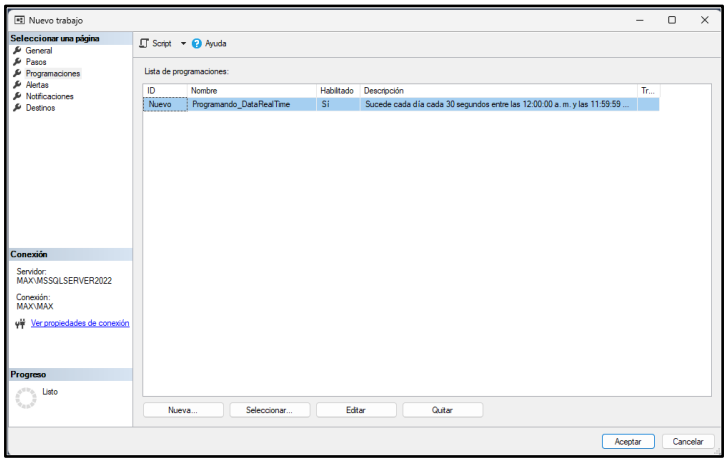




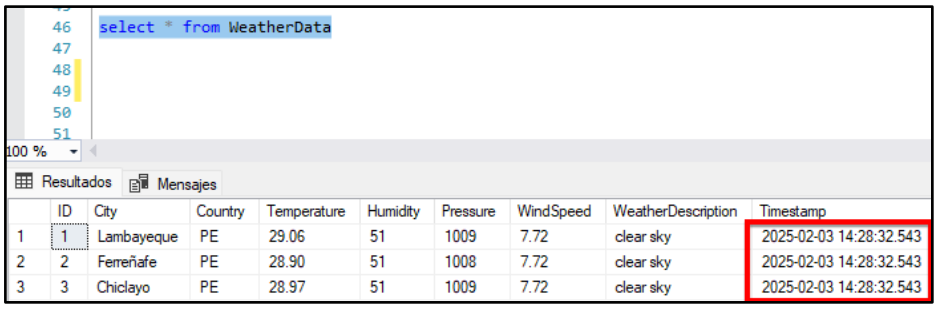


Scheduling to run automatically every 30 seconds:





It will be executed automatically, the time of the first execution is observed:



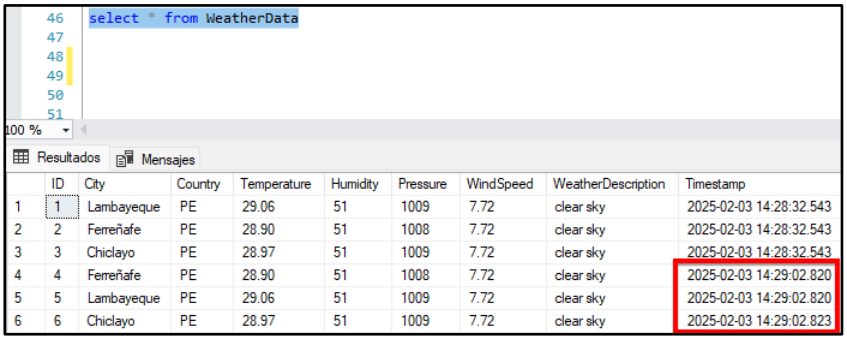
```
select * from WeatherData
```

100 %

Resultados | Mensajes

	ID	City	Country	Temperature	Humidity	Pressure	WindSpeed	WeatherDescription	Timestamp
1	1	Lambayeque	PE	29.06	51	1009	7.72	clear sky	2025-02-03 14:28:32.543
2	2	Ferreñafe	PE	28.90	51	1008	7.72	clear sky	2025-02-03 14:28:32.543
3	3	Chiclayo	PE	28.97	51	1009	7.72	clear sky	2025-02-03 14:28:32.543

On the second run it shows that it happened 30 seconds later just as I programmed it, but you can reduce that time if you want:



```
select * from WeatherData
```

100 %

Resultados | Mensajes

	ID	City	Country	Temperature	Humidity	Pressure	WindSpeed	WeatherDescription	Timestamp
1	1	Lambayeque	PE	29.06	51	1009	7.72	clear sky	2025-02-03 14:28:32.543
2	2	Ferreñafe	PE	28.90	51	1008	7.72	clear sky	2025-02-03 14:28:32.543
3	3	Chiclayo	PE	28.97	51	1009	7.72	clear sky	2025-02-03 14:28:32.543
4	4	Ferreñafe	PE	28.90	51	1008	7.72	clear sky	2025-02-03 14:29:02.820
5	5	Lambayeque	PE	29.06	51	1009	7.72	clear sky	2025-02-03 14:29:02.820
6	6	Chiclayo	PE	28.97	51	1009	7.72	clear sky	2025-02-03 14:29:02.823

In this way I conclude the project where information was extracted from an API and inserted into a database in SQL Server using SSIS and which will be executed every 30 seconds automatically.