Investigate Third Party Environment Working with HPCC Systems – Power BI

Document History:

Version	Change Description	Updated By	Updated On	Comments
1.0	Initial Draft	Girikratna	22nd March 2024	
1.1	Final Draft	Girikratna	-- March 2024	

Table of Contents

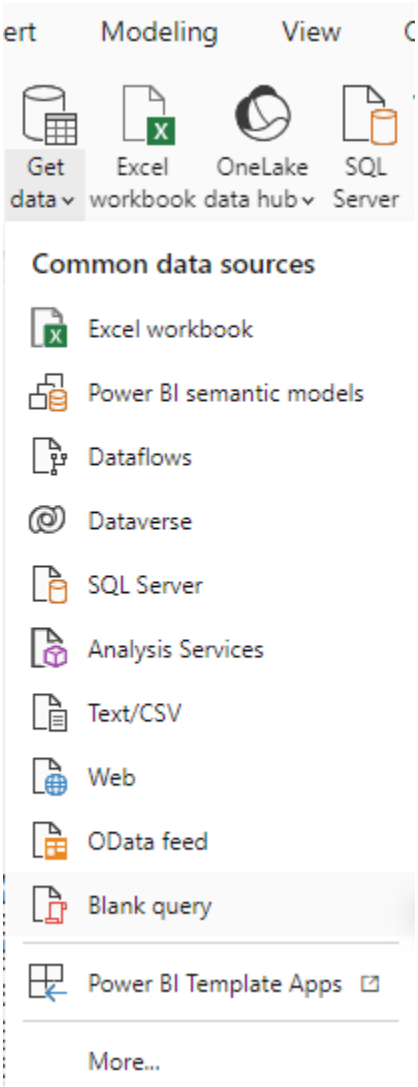
Power BI Connection with HPCC via WsSQL: Power Query M Code

1. Introduction:

For this Document, we are taking the example of the “Ratings” data, based on different Movies, this is a sample assignment as part of Onboarding and these Files are available on the HPCC Webpage. Once we have sprayed this data and made changes using ECL, we then note down the path where it resides, in my case the path is: “gps::movielens::stagingrating”

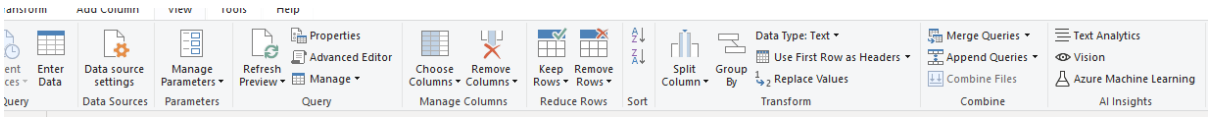
The code given below takes this file into account, also, the SOAP Envelope code can be generated for “your” specific case by going to WsSQL (As part of ESP) and going to “Execute SQL” where we can pass the parameters we need and perform the “SOAP Test”.

This Document contains the detailed steps and working code of connecting HPCC Systems with Power BI via WsSQL:



Once you’ve opened Power BI and created a pbix notebook, the next step is to select “Blank Query” in the “Get Data” dropdown.

That creates “Query 1” now we need to modify the “Advance Editor” which can be seen in the taskbar at the top.



The next step is to add the following Code in the “Advance Editor”.

2. Code Explanation:

M Code:

```
//START

let

url = ""&link&"",

SOAPEnvelope =

"<?xml version="&version&"?>
<soap:Envelope xmlns:soap="&soap&" xmlns:SOAP-ENC="&soapenc&" xmlns="&wssql&">
 <soap:Body>
  <ExecuteSQLRequest>
   <SqlText>"&query&" </SqlText>
   <UserName>"&name&"</UserName>
   <TargetCluster>"&cluster&"</TargetCluster>
   <AlternateClusters>
    <AlternateCluster/>
   </AlternateClusters>
   <TargetQuerySet/>
   <SuppressResults>0</SuppressResults>
```

```
  <SuppressXmlSchema>0</SuppressXmlSchema>
  <Wait>-1</Wait>
  <resultLimit>0</resultLimit>
  <ResultWindowStart>0</ResultWindowStart>
  <ResultWindowCount>0</ResultWindowCount>
  <IgnoreCache/>
 </ExecuteSQLRequest>
</soap:Body>
</soap:Envelope>",

   options = [
      #"Content-Type"="text/xml;charset=utf-8"
   ],

   responseBinary = Web.Contents(url, [Content=Text.ToBinary(SOAPEnvelope), Headers=options]),

   responseText = Text.FromBinary(responseBinary, TextEncoding.Ascii),

   startPos = Text.PositionOf(responseText, "Dataset name=&apos;WsSQLResult&apos; xmlSchema=&quot;WsSQLResultSchema&quot;&gt;") + Text.Length("Dataset name=&apos;WsSQLResult&apos; xmlSchema=&quot;WsSQLResultSchema&quot;&gt;"),
   endPos = Text.PositionOf(responseText, "&lt;/Dataset&gt"),
   resultXmlText = Text.Middle(responseText, startPos, endPos - startPos),

   // Decode the HTML entities
   decodedXmlContent = Text.Replace(Text.Replace(Text.Replace(Text.Replace(resultXmlText, "&lt;", "<"), "&gt;", ">"), "&quot;", """"), "&apos;", "'"),

   // Wrap the XML content with a single root element
   xmlWithRoot = "<Root>" & decodedXmlContent & "</Root>",

   // Parse the XML content into a table
```

```
    xmlTable = Xml.Tables(xmlWithRoot){0},
    // Parse the XML content into a table
    Table = xmlTable[Table]
in
    Table
```

//END

NOTE: The SOAP Envelope has all the code Parameterized, so anyone can just change the value of one parameter without needing to read the XML Request, I have added the SOAP Request from WsSQL below to understand what the Envelope looks like, the next section will explain the code.

SOAP Request from WsSQL: First Part of the Code, whereas the Second Part deals with Decoding the Response.

//START : WITH REMOVING THE ZERO'S AND EMPTY TAGS

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns="urn:hpccsystems:ws:wssql">
 <soap:Body>
  <ExecuteSQLRequest>
   <SqlText>SELECT * FROM gps::movielens::stagingrating</SqlText>
   <UserName>gps</UserName>
   <TargetCluster>thor</TargetCluster>
   <SuppressXmlSchema>1</SuppressXmlSchema>
   <Wait>-1</Wait>
  </ExecuteSQLRequest>
 </soap:Body>
</soap:Envelope>
```

//END

Power BI Connection with HPCC via WsSQL: Power Query M Code

Explanation of the Code after the SOAP Envelope starting from “Options”:

CODE as shown before:

```
    options = [
        #"Content-Type"="text/xml;charset=utf-8"
    ],

    responseBinary = Web.Contents(url, [Content=Text.ToBinary(SOAPEnvelope), Headers=options]),

    responseText = Text.FromBinary(responseBinary, TextEncoding.Ascii),

    startPos = Text.PositionOf(responseText, "Dataset name=&apos;WsSQLResult&apos; xmlSchema=&quot;WsSQLResultSchema&quot;&gt;") + Text.Length("Dataset name=&apos;WsSQLResult&apos; xmlSchema=&quot;WsSQLResultSchema&quot;&gt;"),
    endPos = Text.PositionOf(responseText, "&lt;/Dataset&gt"),
    resultXmlText = Text.Middle(responseText, startPos, endPos - startPos),

    // Decode the HTML entities
    decodedXmlContent = Text.Replace(Text.Replace(Text.Replace(Text.Replace(resultXmlText, "&lt;", "<"), "&gt;", ">"), "&quot;", """), "&apos;", "'"),

    // Wrap the XML content with a single root element
    xmlWithRoot = "<Root>" & decodedXmlContent & "</Root>",

    // Parse the XML content into a table
    xmlTable = Xml.Tables(xmlWithRoot){0},
    // Parse the XML content into a table
    Table = xmlTable[Table]
in
    Table
```

- Starting from the “responseBinary” variable, the code sends a POST request to the specified URL using the Web.Contents function.
- This request includes the SOAP envelope converted to binary using Text.ToBinary.
- The request headers are set to specify that the content type is XML encoded in UTF-8. The response from the request is received as binary data.
- The binary response is converted to text using Text.FromBinary, specifying ASCII encoding. This step is necessary to perform text manipulation operations on the response.
- The “startPos” and “endPos” variables locate the start and end positions of the XML content within the response text. These positions are found using the Text.PositionOf function, which searches for specific substrings within the text.
- The “resultXmlText” variable extracts the XML content from the response text using the Text.Middle function. This function extracts a substring from the response text, starting from the position indicated by “startPos” and ending at the position indicated by “endPos”.
- Subsequently, the code decodes any HTML entities present in the extracted XML content. This is done using nested Text.Replace functions to replace HTML entity codes with their corresponding characters.
- The XML content is then wrapped with a root element and to ensure it forms a valid XML structure. This is stored in the “xmlWithRoot” variable.
- The code uses the Xml.Tables function to parse the XML content into a table. The {0} index is used to select the first table from the result, as there may be multiple tables returned and we can decide which one to return. The resulting table is stored in the “xmlTable” variable.
- The “Table” variable extracts the parsed XML table from the “xmlTable” variable. This table contains the structured data extracted from the XML response and is the final output of the code.

3. Result:

Once the code above is written and executed, it will send a Request to WsSQL which in turn passes the Envelope to said destination, in this case it's a Thor Cluster file, then the response is received in Power BI in XML form. The code then automatically decoded this and converts this into Tabular form.

For the example shown above, the output data is shown in the screenshot below: