

# Investigate Third Party Environment Working with HPCC Systems – Power BI and GitPod

Procedural and Technical Document

Document Code:

Version: 1.0

Date: {04/02/2024}

CONFIDENTIAL & PROPRIETARY

The recipient of this material (hereinafter "the Material") acknowledges that it contains confidential and proprietary data the disclosure to, or use of which by, third parties will be damaging to LexisNexis Risk Solutions Inc. and its affiliated companies (hereinafter "LexisNexis Risk Solutions "). Therefore, recipient agrees to hold the Material in strictest confidence, not to make use of it other than for the purpose for which it is being provided, to release it only to employees requiring such information, and not to release or disclose it to any other party. Upon request, recipient will return the Material together with all copies and modifications, if any.

All names in the text, or on the sample reports and screens shown in this document, are of fictitious persons and entities. Any similarity to the name of any real person, address, school, business or other entity is purely coincidental.

© Copyright 2022 LexisNexis Risk Solutions Inc.

All rights reserved.

“LexisNexis Risk Solutions” is a registered trademark, and the LexisNexis Risk Solutions logo is a trademark of LexisNexis Risk Solutions Asset Company.

## HISTORY OF REVISIONS

Version#	Date	Author	Description of Changes	Reviewer
1	02/04/2024	Girikratna Sharma	Initial	Srinivasan Kothandam

## TABLE OF CONTENTS

<b>1</b>	<b>Introduction:</b>	<b>5</b>
<b>1.1</b>	<b>Purpose of this Document:</b>	<b>5</b>
1.1.1	Scope:	5
1.1.2	Prerequisites:	5
<b>2</b>	<b>Bare Metal HPCC Deployment:</b>	<b>6</b>
<b>2.1</b>	<b>WsSQL setup on Bare Metal:</b>	<b>9</b>
2.1.1	Bare Metal WsSQL Configuration and Working:	9
<b>3</b>	<b>Connection of Power BI with HPCC Systems:</b>	<b>11</b>
<b>3.1</b>	<b>Testing WsSQL with HPCC Systems:</b>	<b>12</b>
<b>3.2</b>	<b>Connection of Power BI with HPCC Systems:</b>	<b>14</b>
3.2.1	Power Query Code for Connection:	16
3.2.2	Query Code with Larger Data:	21
3.2.3	Results generated and Visualization:	23
<b>4</b>	<b>Case Study on GitPod:</b>	<b>25</b>
<b>4.1</b>	<b>GitPod Integration with HPCC Platform:</b>	<b>26</b>
4.1.1	GitPod Roadblocks faced with potential alternatives:	27
<b>5</b>	<b>5 REFERENCE:</b>	<b>30</b>

# 1 Introduction:

This Procedural and Technical document talks about the Integration of third-party software's with the HPCC Platform, mainly taking 2 into consideration, Power BI and GitPod.

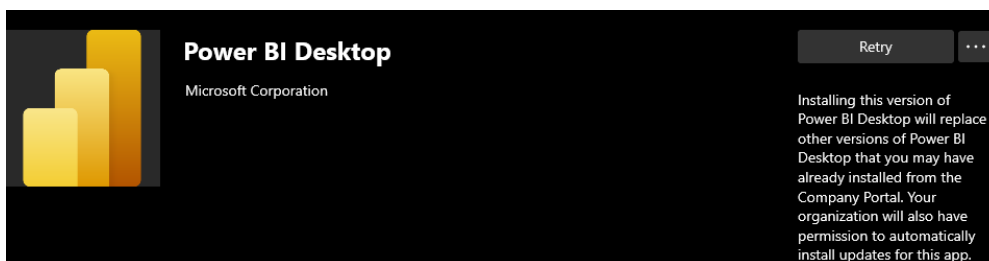
## 1.1 Purpose of this Document:

### 1.1.1 Scope:

This document does a deep dive into all the required software's and installations with a details explanation of the steps needed to be performed with information on the output, such as the data used and the information it generated.

### 1.1.2 Prerequisites:

A Basic understanding of SOAP Requests and Response, Installing and setting up Power BI in your local or company machine, the next sections will talk about the installation and setup of A Local Instance of the HPCC Platform as a single node cluster. We also need to have a development code on GitHub which we will need later for the integration of GitPod with the HPCC Platform, wherein this code has a decent amount of dependencies as we can then test this out in the Gitpod side, ideally we can take the HPCC Platform's GitHub page and fork that out into the persons personal account, as later on we will use that to build the whole gitpod project.



## 2 Bare Metal HPCC Deployment:

**NOTE:** *We have been provided with a Windows machine, we need to install WSL(Windows Subsystem for Linux) and then follow the procedure through WSL & direct Linux installations are one and the same.*

The official documentation of the Boca Raton Documentation Team is used as a reference here and will be linked below in the references section. Titled “[Installing &Running the HPCC Systems Platform](#)”. We have also setup ECL IDE to create a new preference called “Local” which runs the local build of the HPCC Platform which we installed on a single node on our local machine.

Once, the steps written in the above documentation are followed we then have a Single Node HPCC Platform set up on our Local machine inside the Windows Subsystem of Linux (WSL) which in our case is Ubuntu 22.

The next thing to do is to start the HPCC Platform which we have installed, and we do that by running the following commands on the WSL Terminal:

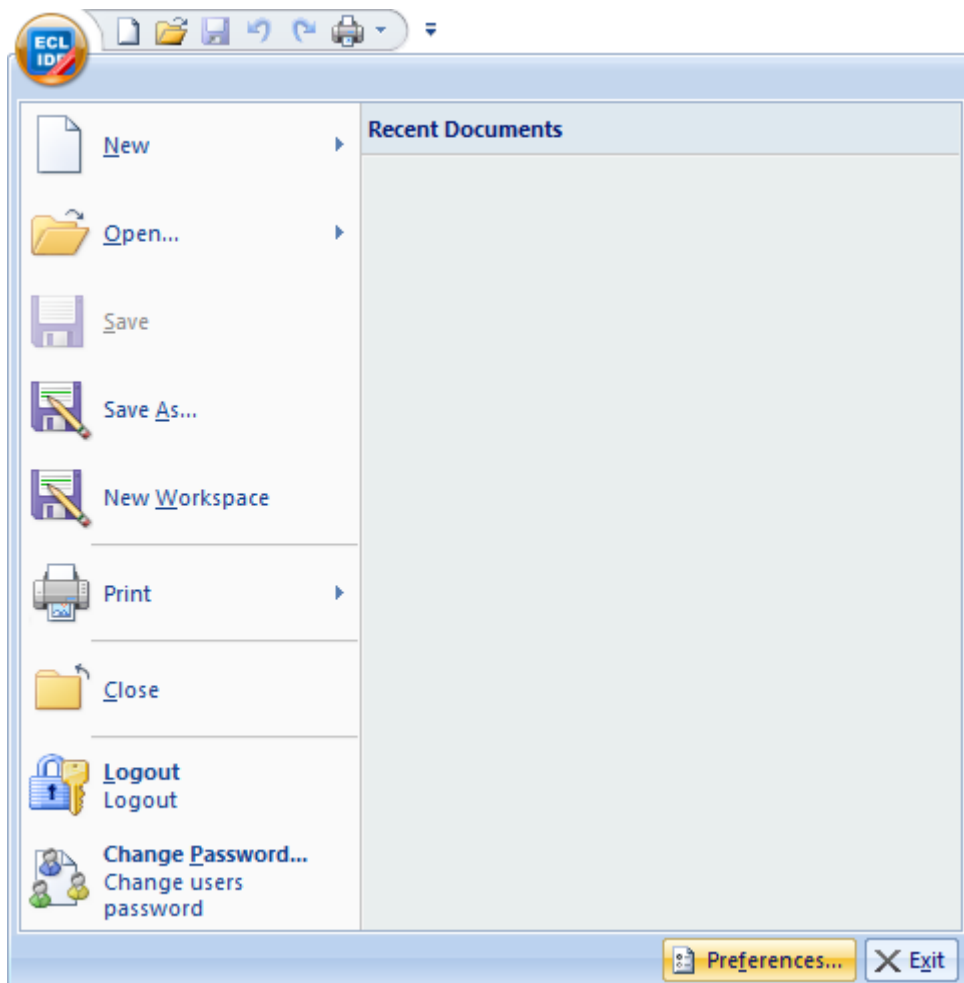
```
root@RIS-qsJ63Rr167Y: ~  
gsharma@RIS-qsJ63Rr167Y:~$ sudo su  
root@RIS-qsJ63Rr167Y:/home/gsharma# cd /root/  
root@RIS-qsJ63Rr167Y:~# systemctl start hpccsystems-platform.target
```

Now, this “start” command will take some time to execute and even if you see a warning called:

```
root@RIS-qsJ63Rr167Y:~# systemctl start hpccsystems-platform.target  
A dependency job for hpccsystems-platform.target failed. See 'journalctl -xe' for details.  
root@RIS-qsJ63Rr167Y:~#
```

The next step would be to open your preferred browser and open “localhost:8010” and that would open ECL Watch for our Installed HPCC Platform, out here we can “Spray” files and perform other actions but a bit easier to use if we can configure ECL IDE with the same Platform.

On Opening ECL IDE (Which we can download from Company Portal for LexisNexis employees) we go to “Preferences” which can we viewed by:



And add the following in a “new” Preference we create:

Preferences

Configurations: Local

Locate New... Delete

Server Editor Colors Results Compiler Other

Server localhost ☐ SSL ☒ Advanced

Topology Server: http://localhost:8010/WsTopology

Workunit Server: http://localhost:8010/WsWorkunits

Attribute Server:

Account Server: http://localhost:8010/Ws\_Account

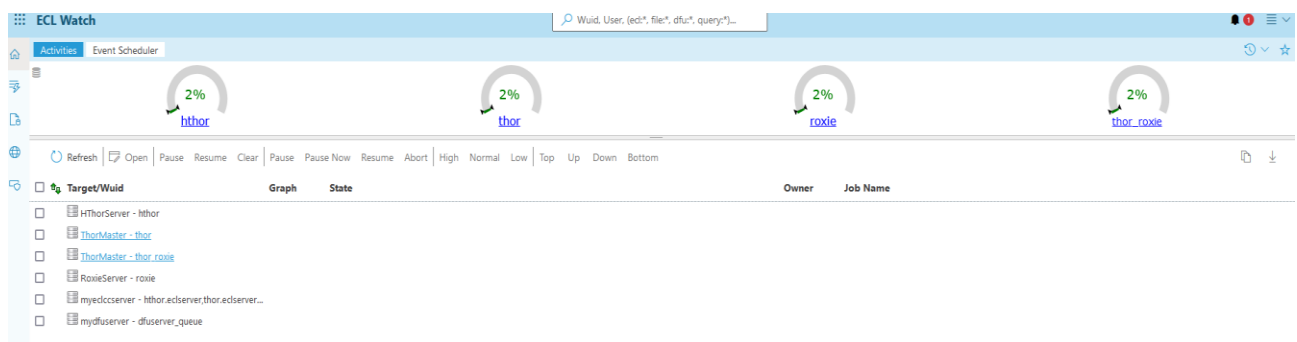
SMC Server: http://localhost:8010/WsSMC

Spray Server: http://localhost:8010/FileSpray

DFU Server: http://localhost:8010/WsDfu

Ok Cancel Apply

Now, we have ECL IDE working with our Local Platform, we have ECL Watch working as well and can even access and see ESP at Port: 8002, and to configure we can go to port 8015.





## 2.1 WsSQL setup on Bare Metal:

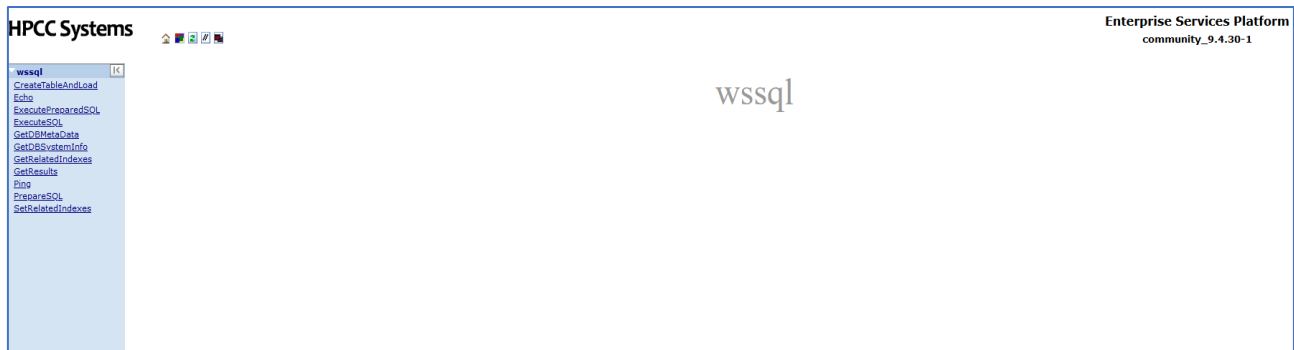
Next, we need to install and setup WsSQL, it is worth making a note that most current versions of the HPCC Platform have WsSQL installed out of the box but in some cases of Single Node Install it is needed that we configure WsSQL.

The Boca Raton Documentation of "[WsSQL ESP Web Service Guide](#)" follows all the steps to do the same wherein we can install it manually or using an installation wizard.

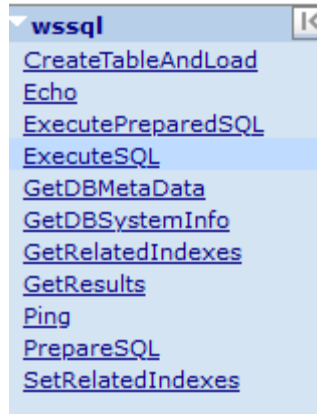
Once those steps are completed, we can follow 2.1.1

### 2.1.1 Bare Metal WsSQL Configuration and Working:

The next step is to go to "localhost:8510" wherein we have configured "WsSQL" now this is a part of the Enterprise Service Platform as a Web Service.



WsSQL :This is a service provided as part of ESP with the main function of converting a SQL text into an output valid on the HPCC Platform, so this means that one does not need to learn ECL Language to write code and retrieve information from a file or large database which is present on the HPCC Platform.



Let's Look at "ExecuteSQL" in more detail as that would help us later.

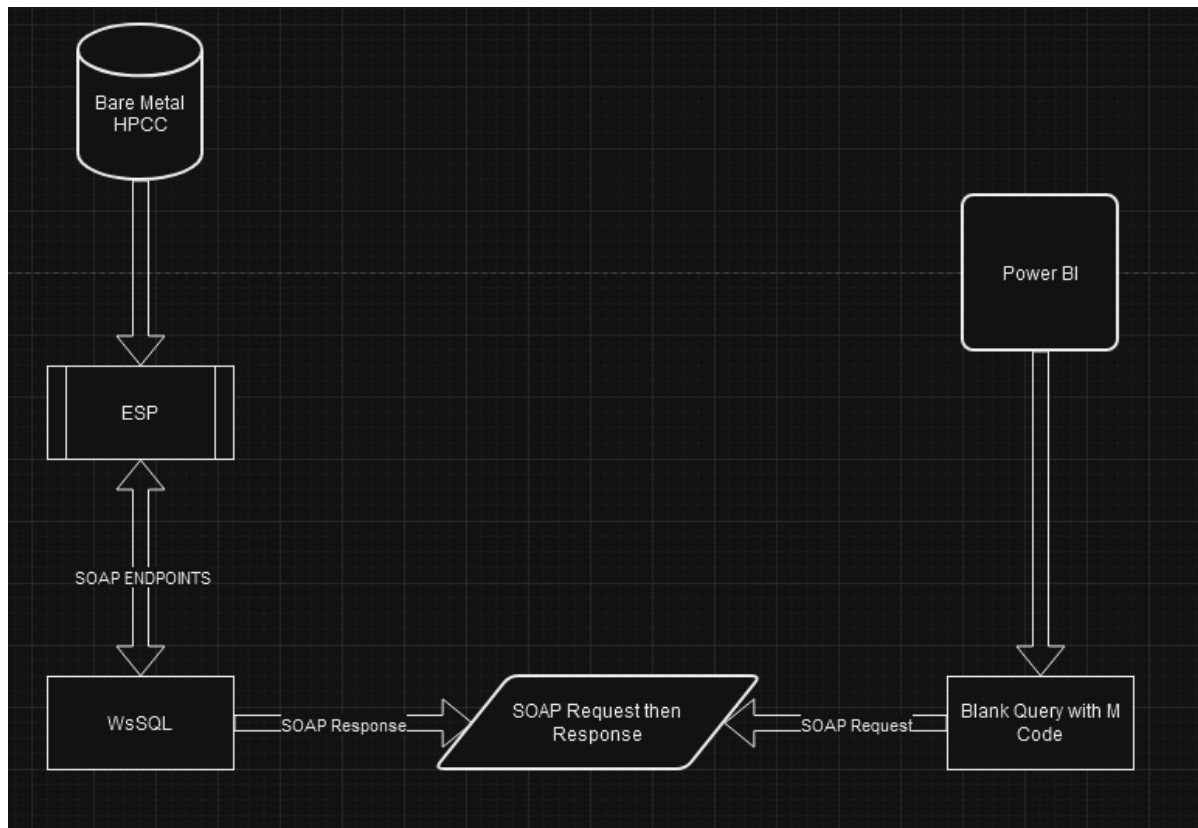
 A screenshot of a web application titled 'wssql[Version 3.06]'. The main heading is '>ExecuteSQL' with links for 'WSDL', 'XSD', 'XMLRequest', 'XMLResponse', 'JSONRequest', and 'JSONResponse'. Below this, it says 'Description: No description available' and 'Help: No Help available'. A section titled 'EXECUTESQLREQUEST' has a checked checkbox. The form contains several fields: 'SqlText:' with the value 'SELECT \* FROM gps::movielens::stagingrating', 'UserName:' with 'gps', and 'TargetCluster:' with 'thor'. There are also checkboxes for 'AlternateClusters:', 'TargetQuerySet:', 'SuppressResults?', 'SuppressXmlSchema?' (checked), 'Wait:', 'resultLimit:', 'ResultWindowStart:', 'ResultWindowCount:', and 'IgnoreCache?'. At the bottom, there are buttons for 'Submit', 'SOAP Test', 'Json Test', 'Reset', 'Clear All', and 'Link to This Form'.

Out here we can fill options of data such as "sqlText" of what we need and since this is configured the local single node cluster once we click on "SOAP Test" the generated SOAP Request will be used later on in Power BI to establish a connection.

### 3 Connection of Power BI with HPCC Systems:

The Planned connection is to use WsSQL to bridge the gap between Power BI and the HPCC Platform, this is done using “Power Query” in Power BI to establish a connection with WsSQL and WsSQL in turn has its own connection with the HPCC Platform via ESP, so theoretically we can fetch data using SQL Queries in WsSQL which we can write in Power Query. This method and Architecture is performed successfully and explained in the following sub sections.

The following Diagram shows the Architecture of the connection with the flow of data between the 3:



### 3.1 Testing WsSQL with HPCC Systems:

WsSQL has multiple parameters which is passed through so this section is to note those parameters such that we can automate the SOAP Request we will send from Power BI, as variables. So once we are going to take a sample dataset to try and retrieve data we want as a “Work Unit” all from the WsSQL Web Service.

Once, we fill all the required fields the ExecuteSQL page, we can then run a “SOAP Test” now that is beneficial as WsSQL itself creates an XML SOAP Envelope for us, and this same SOAP Envelope will be used to retrieve data in Power BI’s end. The SOAP Envelope is shown below and explained in detail:

```
<?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:hpcsystems:ws:wssql">
  <soap:Body>
    <ExecuteSQLRequest>
      <SqlText>SELECT * FROM gps::movielens::stagingrating</SqlText>
      <UserName>gps</UserName>
      <TargetCluster>thor</TargetCluster>
      <AlternateClusters>
        <AlternateCluster/>
      </AlternateClusters>
      <TargetQuerySet/>
      <SuppressResults>0</SuppressResults>
      <SuppressXmlSchema>1</SuppressXmlSchema>
      <Wait>-1</Wait>
      <resultLimit>0</resultLimit>
      <ResultWindowStart>0</ResultWindowStart>
      <ResultWindowCount>0</ResultWindowCount>
      <IgnoreCache/>
    </ExecuteSQLRequest>
  </soap:Body>
</soap:Envelope>
```

As shown above, the <ExecuteSQLRequest> tag is used to send the information of the WsSQL parameters through it and inside that the main parameter is the <SqlText> Tag, as this tag contains the SQL Query which will be passed into the HPCC Platform and execute the tasks needed to be executed. With other important parameters which include <UserName> as to which user is executing the request, then importantly the <TargetCluster> which in this case points to “thor”.

Most of the other parameters are optional and can be defined as a fixed variable in Power Query such that it does not need to be modified, hence automating this, and the important ones can be noted, and only those need constant or frequent updates.

The above Request once sent creates a Work Unit which can be viewed in ECL Watch, this is also shown as a “WsSQL Job” so its easy to identify which tasks are executed by WsSQL and which tasks are not.

**ECL Watch**

WUID: W20240402-162910

Variables (10) | Outputs (2) | Inputs (1) | Metrics (1) | Workflows | Queries | Resources (1) | Helpers (9) | Logs (\*) | ECL | XML

Refresh | Copy WUID | Save | Delete | Restore | Reschedule | Deschedule | Set To Failed | Abort | Recover | Resubmit | Clone | Publish | ZAP | Worker Logs

**W20240402-162910**

WUID: W20240402-162910

Action: compile

State: completed

Owner: gps

Job Name: WsSQL Job

Description:

Potential Savings: 0.00 (USD) (0%)

Compile Cost: 0.000012 (USD)

Execution Cost: 0.000100 (USD)

☒ 0 Error(s) ☒ 0 Cost(s) ☒ 0 Warning(s) ☒ 0 Info(s) ☒ 0 Other(s)

Severity	Source / Cost	Code	Message	Col	Line	Activity	File Name
----------	---------------	------	---------	-----	------	----------	-----------

Created | Compiled | Completed

And now to show the sample Output to validate the Connection between the 2

**W20240402-162910** | Variables (10) | **Outputs (2)** | Inputs (1) | Metrics (1) | Workflows | Queries | Resources (1) | Helpers (9) | Logs (\*) | ECL | XML

Refresh | Filter | HTML

userid	movieid	rating	timestamp
1	1	4	2000-07-30-18-45-03
1	3	4	2000-07-30-18-20-47
1	6	4	2000-07-30-18-37-04
1	47	5	2000-07-30-19-03-35
1	50	5	2000-07-30-18-48-51
1	70	3	2000-07-30-18-48-00
1	101	5	2000-07-30-18-14-28
1	110	4	2000-07-30-18-36-16
1	151	5	2000-07-30-19-07-21
1	157	5	2000-07-30-19-08-20
1	163	5	2000-07-30-19-00-50
1	216	5	2000-07-30-18-20-08
1	223	3	2000-07-30-18-16-25
1	231	5	2000-07-30-18-19-39
1	235	4	2000-07-30-18-15-08
1	260	5	2000-07-30-18-28-00
1	296	3	2000-07-30-18-49-27

### 3.2 Connection of Power BI with HPCC Systems:

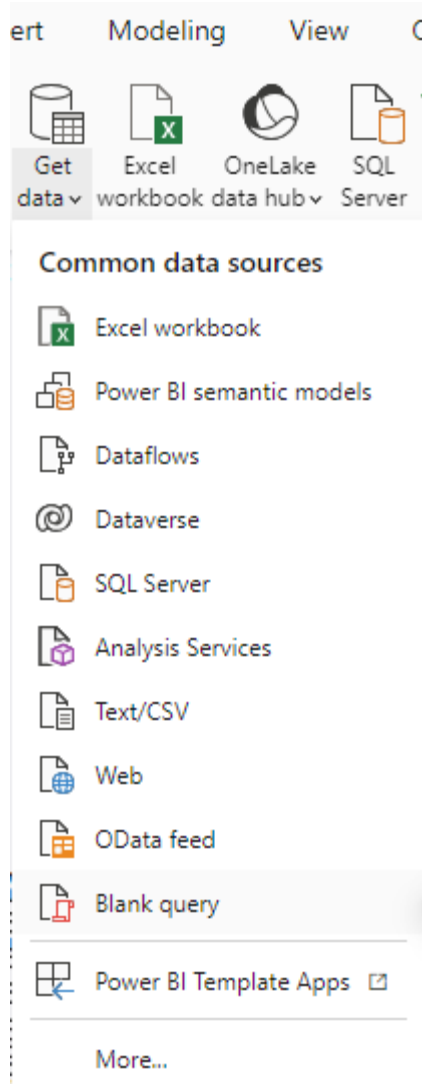
For this Document, we are taking the example of the “Ratings” data, based on different Movies, this is a sample assignment as part of Onboarding and these Files are available on the HPCC Webpage. Once we have sprayed this data and made changes using ECL, we then note down the path where it resides, in 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 code is developed in a manner wherein the Decoding of the SOAP Response is done in a manner where the Data, the quantity or the Type of Data do not make a difference, as this does not take those into account during the decoding, so this is beneficial as the previous steps of using MySQL Server to then take the HPCC Data into Power BI can be removed completely.

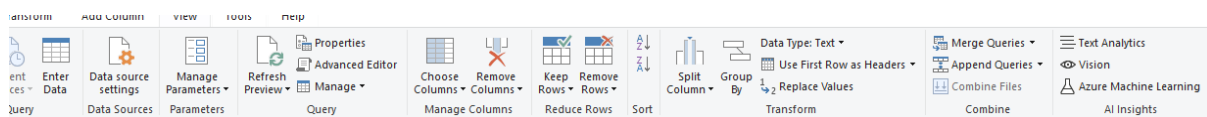
All the Results and Visualizations in this document are sample in nature and are not live data as these are just used for training and sample generated data and is just used here to show the power of the connection and how it can be used for larger live data with the same effect.

Now, to understand the following steps we need to navigate inside Power BI to reach the Power Query option, this is explained in the image below.



On Opening Power BI, we need to Navigate to “Get Data” on the Toolbar at the top and then select “Blank query” from the drop-down menu.

That in turn creates a “Query 1” now we need to modify the “Advance Editor” which is seen on the taskbar.



The next step is to look at the code for the connection of the 2 and this is done with everything parametrized so it is easier to read and to understand, just to note that the code is split into 2 parts, the first is the SOAP Envelope and the second is the transformations which the SOAP Response needs to undergo to be in Tabular format.

### 3.1.1 Power Query Code for Connection:

We now are going to look at the code which is used to send the SOAP Envelope from Power BI to WsSQL and that in turn to the HPCC Platform, This code is split into 2 parts, the First Part ends at the end of the SOAP Envelope and the second part contains the Decoding Code which is used to change the XML Response into valid Data in the form of Tables.

```
let
url = "&link&",
SOAPEnvelope =
"<?xml version='&version&'>
<soap:Envelope xmlns:soap='&soap&' xmlns:SOAP-ENC='&soapenc&' xmlns:wssql='&wssql&'>
<soap:Body>
<ExecuteSQLRequest>
<SqlText>SELECT * FROM gps::movielens::stagingrating</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
```

To ensure that there is no confusion, the SOAP Envelope is same as the Code provided above in the WsSQL “ExecuteSQL” section.



The explanation of the Code which performs decoding is provided in points below:

- 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.

To Provision the same code as a text such that it can be executed directly, provide the above-mentioned configuration steps are followed:

M Code:

```
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='WsSQLResult';
xmlSchema='WsSQLResultSchema';") + Text.Length("Dataset
name='WsSQLResult'; xmlSchema='WsSQLResultSchema';"),

```

```

endPos = Text.PositionOf(responseText, "</Dataset>"),

```

```

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
```

Just to ensure that the Parameters are not confusing I have also added the SOAP Envelope which is generated by WsSQL:

//NOTE:

```
<?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:hpcsystems: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>
```

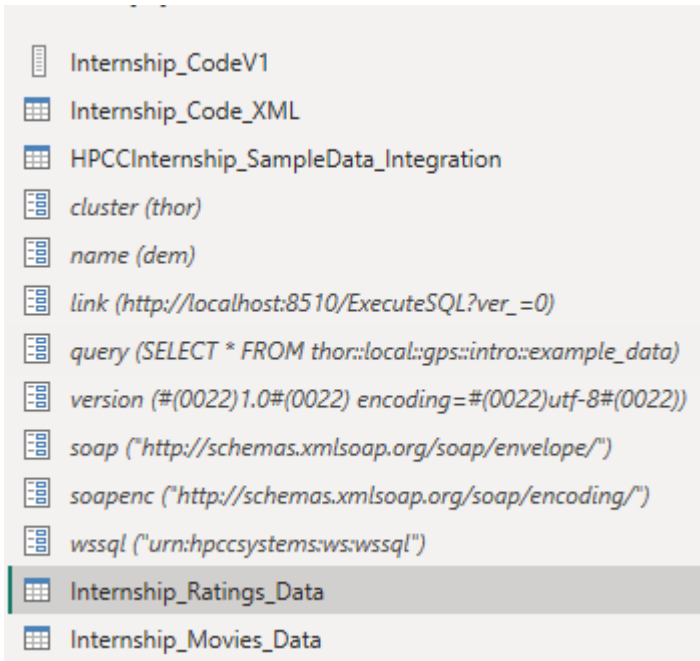
```
</soap:Body>
```

```
</soap:Envelope>
```

This marks the end of the Coding section; we now will look at the same with a larger data set as an output and explaining all the parameters.

### 3.1.2 Query Code with Larger Data:

Now, that the code is completed we now can have a look at all the automations which we performed, such as defining parameters in Power Query.



The above shown parameters are shown with name(value) format starting from “cluster” and ending at “wssql”.

These parameters are then added into the SOAP Envelope with the name of the variable, and that then in turn calls the value when the script is getting executed.

For the sample data we explained before this code given above takes that and generates the output inside Power BI as a Table which is then saved from getData which in turn can be used to create Dashboards and for various other analytics and reports based on the need and level of complexity of data present.

Queries [13]

- Internship\_CodeV1
- Internship\_Code\_XML
- HPCCInternship\_SampleData\_Integration
- cluster (thor)
- name (dem)
- link (http://localhost:8510/ExecuteSQL?ver\_=0)
- query (SELECT \* FROM thor:localgps:intra:example\_data)
- version (#(0022)1.0#(0022) encoding=#(0022)utf-8#(0022))
- soap ("http://schemas.xmlsoap.org/soap/envelope/")
- soapenc ("http://schemas.xmlsoap.org/soap/encoding/")
- wssql ("urn:hpcsystems:ws:wssql")
- Internship\_Ratings\_Data
- Internship\_Movies\_Data

fx = xmlTable(Table)

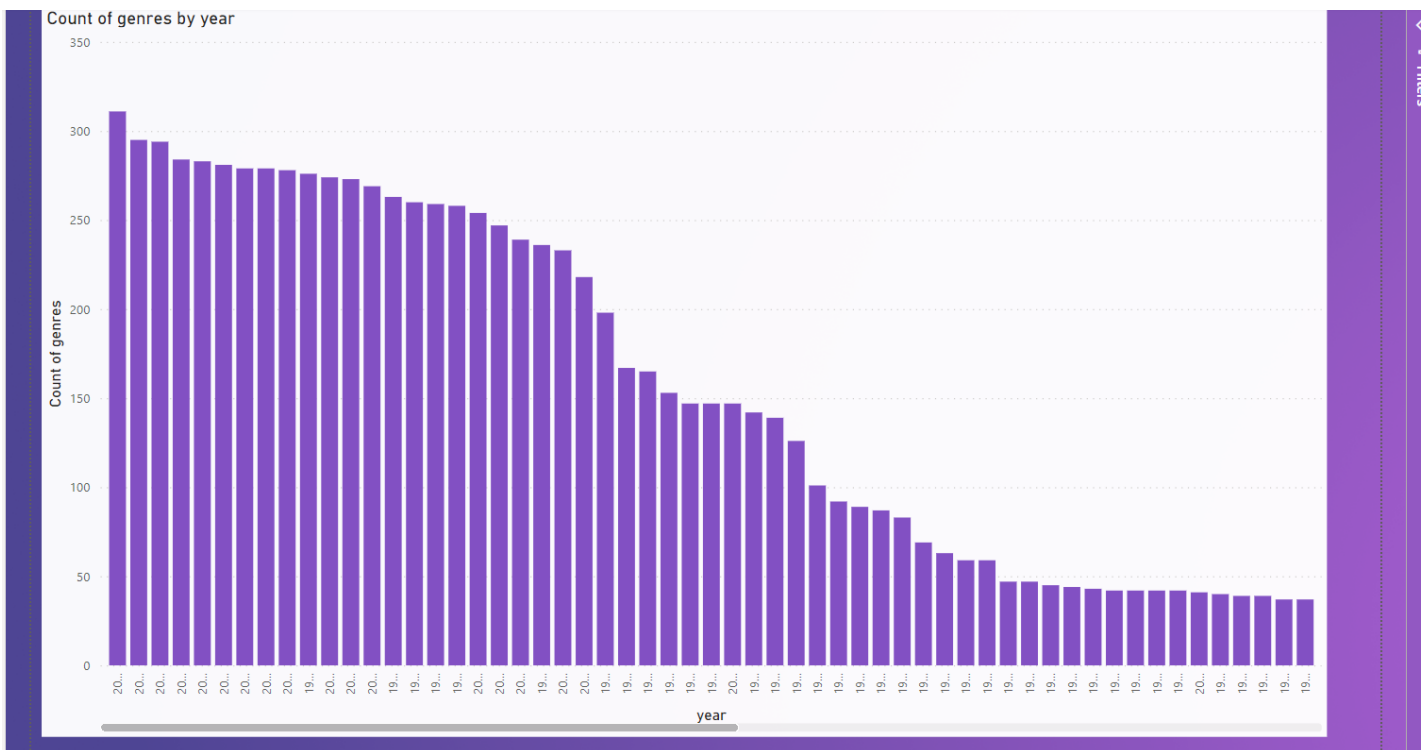
	A <sub>C</sub> userid	A <sub>C</sub> movieid	A <sub>C</sub> rating	A <sub>C</sub> timestamp
1	1	1	4	2000-07-30-18-45-03
2	1	3	4	2000-07-30-18-20-47
3	1	6	4	2000-07-30-18-37-04
4	1	47	5	2000-07-30-19-03-35
5	1	50	5	2000-07-30-18-48-51
6	1	70	3	2000-07-30-18-40-00
7	1	101	5	2000-07-30-18-14-28
8	1	110	4	2000-07-30-18-36-16
9	1	151	5	2000-07-30-19-07-21
10	1	157	5	2000-07-30-19-08-20
11	1	163	5	2000-07-30-19-00-50
12	1	216	5	2000-07-30-18-20-08
13	1	223	3	2000-07-30-18-16-25
14	1	231	5	2000-07-30-18-19-39
15	1	235	4	2000-07-30-18-15-08
16	1	260	5	2000-07-30-18-28-00
17	1	296	3	2000-07-30-18-49-27
18	1	316	3	2000-07-30-18-38-30
19	1	333	5	2000-07-30-18-19-39
20	1	349	4	2000-07-30-18-42-43
21	1	356	4	2000-07-30-18-16-02
22	1	362	5	2000-07-30-18-43-08
23	1	367	4	2000-07-30-18-28-30
24	1	423	3	2000-07-30-18-39-23
25	1	441	4	2000-07-30-18-14-28
26	1	457	5	2000-07-30-18-31-49
27	1	480	4	2000-07-30-18-39-06
28	1	500	3	2000-07-30-18-20-08
29	1	527	5	2000-07-30-19-06-42
30	1	543	4	2000-07-30-18-19-39
31	1	552	4	2000-07-30-18-44-13
32	1	553	5	2000-07-30-19-09-13
33	1	590	4	2000-07-30-18-42-26
34	1	592	4	2000-07-30-18-37-51
35	1	593	4	2000-07-30-19-03-13
36	1	596	5	2000-07-30-18-47-18
37	1	608	5	2000-07-30-18-48-51
38	1	648	3	2000-07-30-18-42-43

4 COLUMNS, 999+ ROWS Column profiling based on top 1000 rows

The above image shows the output in the form of Tables which can then directly be used.

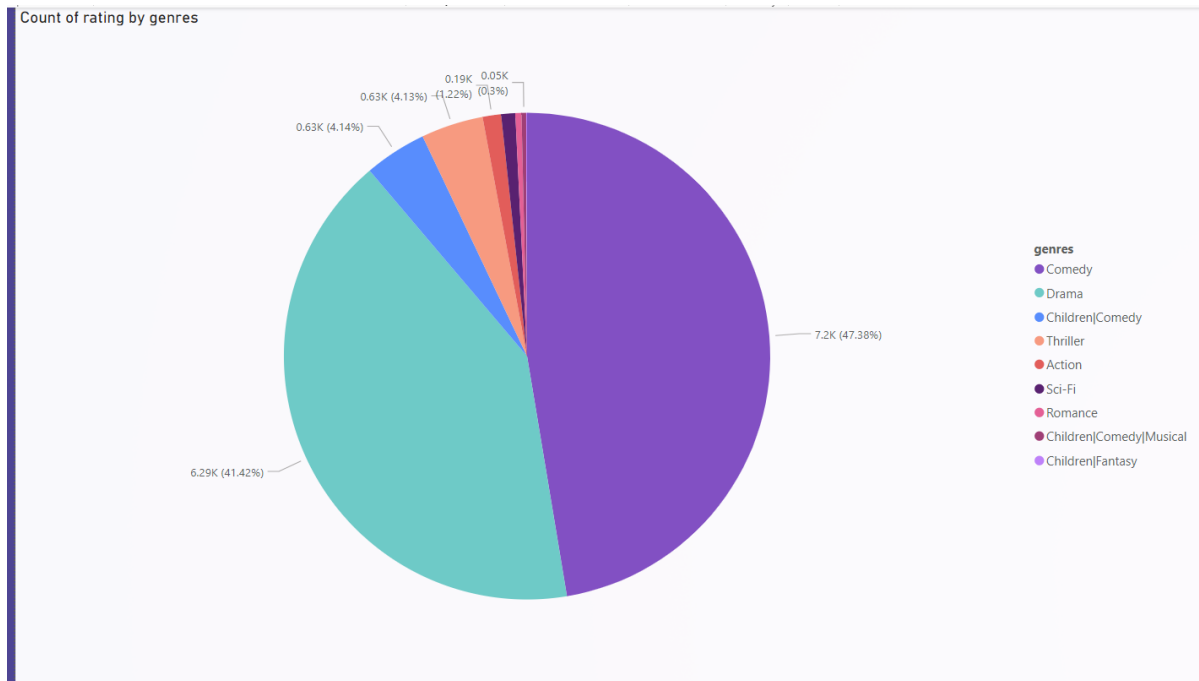
### 3.1.3 Results generated and Visualization:

To reach the end we can now talk about the Results and the Visualization as we have successfully received data inside Power BI, from the HPCC Platform, and will now use this data to create some Dashboards and mainly graphs to show a use case of Analytics and showing a complete cycle of data Loading and till data Analytics.

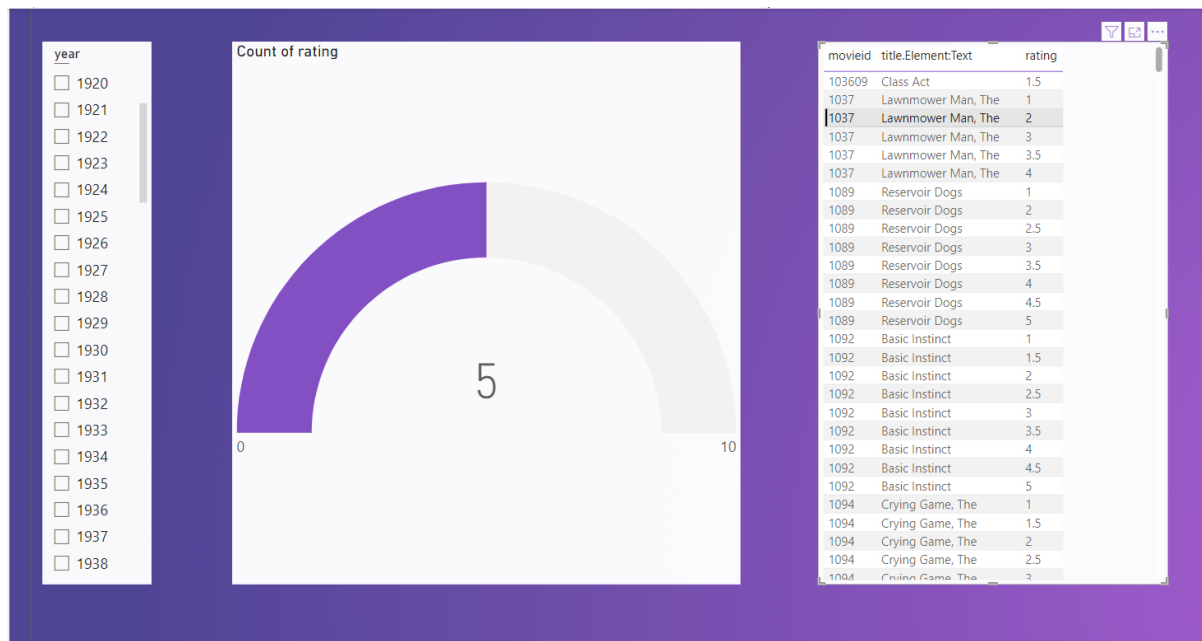


The above graph shows the different counts of Movie “genres” over the years as the sample data used was based on Movies and Rating and different factors.

The next graph talks about count of Ratings by Genres and the final has an option to choose the different results we need for analysis.



And the final graph shown below:



As the columns are “User ID”, “Movie ID”, “Ratings” , “Timestamp” and so on.



## 4 Case Study on GitPod:

GitPod is a software product which is mainly used as a way to provision a virtual system to a user, with an inbuilt IDE set up and configured to the project they are working on. Having native support with GitHub, let's say someone wants to open a github repository, but lacks the development environment, if that project is configured with gitpod, they can just click a button (on the repo page saying "gitpod") and gitpod will do the rest, so on a browser, they have a IDE let's say VS Code for an example, and that IDE has the project set up and configured with all dependencies installed.

So, the user can directly start contributing to the code without the need to configure an environment on their own. This is beneficial to newer developers as they can directly contribute and all they need to contribute code is a laptop and internet (a github account of course).

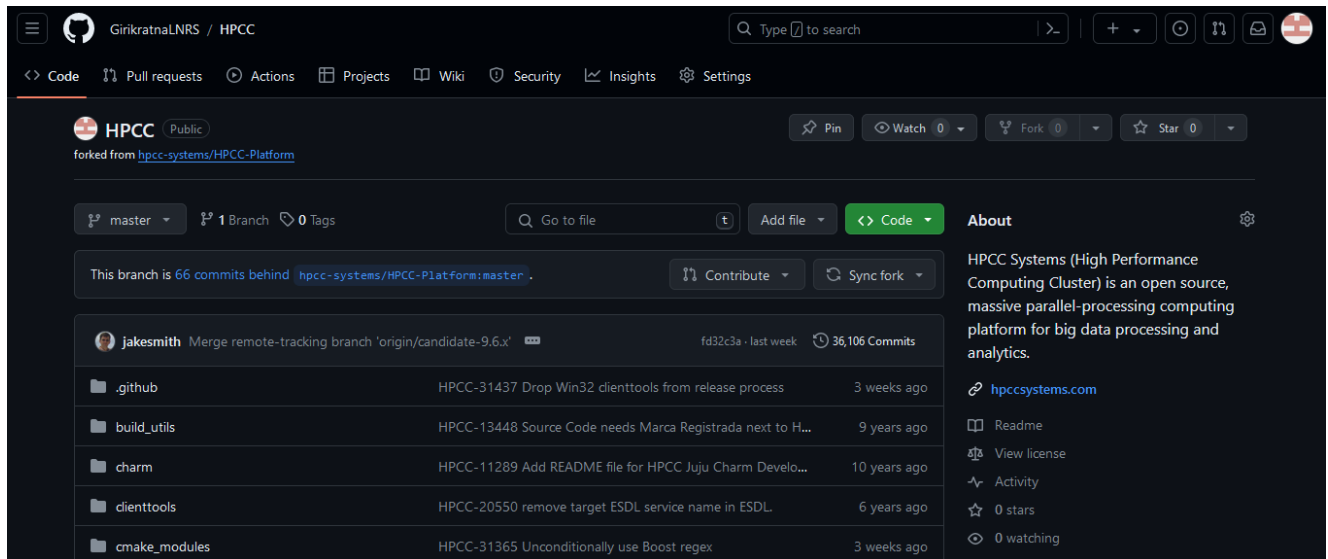
One of the main reasons why I thought of gitpod was that, say someone with less experience in open-source contributions and no experience with setting up a platform on their machine, wants to contribute to the HPCC Platform, which is Open Source and available on Github. Now if gitpod can be configured with the HPCC Platform then that person can just directly run the platform on gitpod itself, this is advantageous and can create a thriving open-source community for the HPCC Platform.

Now, it should be mentioned that gitpod has a limit of 500 credits free for a github accounts, which is nearly 500 hours of use of a single workspace. It then charges \$9 per hour for the open workspace, it is cheaper if an organization opts for it.

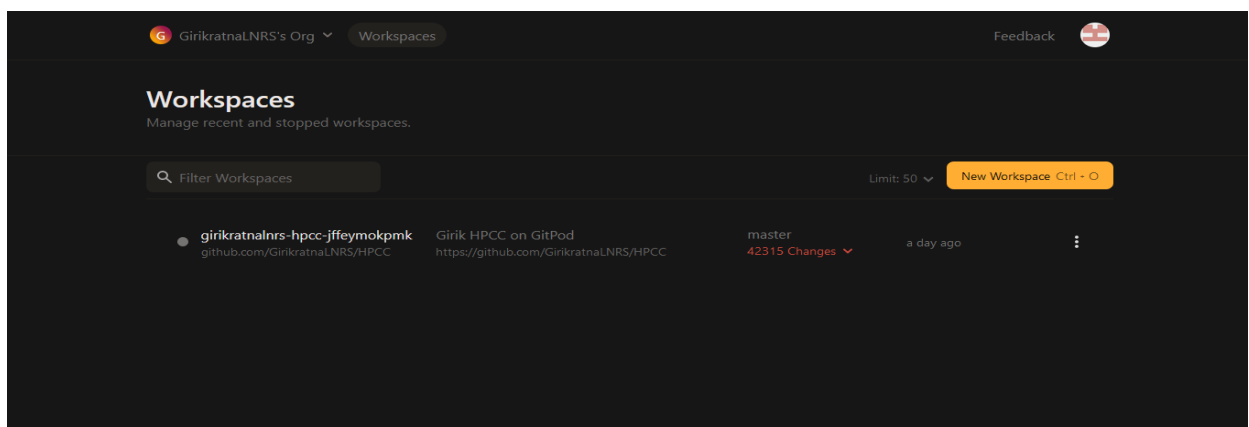
GitPod works in a way that when a project is to be configured with it, we need to define a gitpod.yml file, which is a configuration file that works wherein we define all the extensions, the port numbers and all the steps needed in configuring a project inside that one file, so each time a workspace is opened that yml file gets triggered and executes. Gitpod also ensure that all extensions are up to date so there is no need to manually perform that. The next section describes the same process wherein a project was setup using the gitpod yml file and it directly executes from the browser.

## 4.1 GitPod Integration with HPCC Platform:

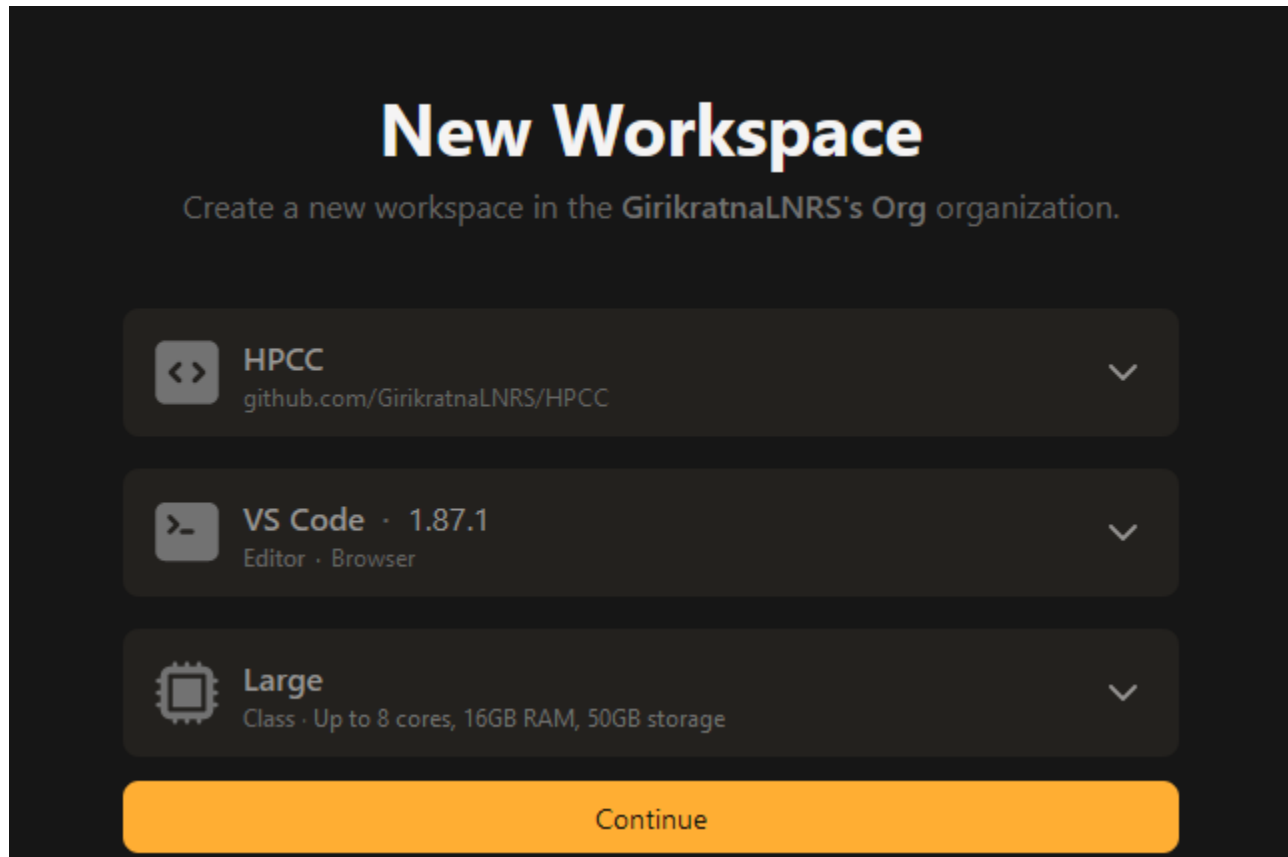
To start off I “forked” the GitHub HPCC Platform Repository, onto my Risk Account (Created a Github account with those credentials), the next step was to sign up with GitPod with the said GitHub account.



Once that is done, we can then go to our GitPod Dashboard, or the GitPod homepage, now in my case there is a Workspace created, but for a new user it would be empty, in the next section I will talk about the steps to create a gitpod workspace and then I'll talk about the steps which we need to take ahead of that.



Now, to create a new workspace we can give emphasis to the virtual machine we use, based on the project size and level of complexity we can choose 8 cores and 16 gigs of ram, with a provisioning of 50 GB Storage, and can take the Code Editor of our choice.



#### 4.1.1 GitPod Roadblocks faced with potential alternatives:

Now when the GitPod integration was tried with the HPCC Platform, I realized that the need to install the Platform via WSL or Helm won't be necessary, as Gitpod can do the same, but that is a stable installation, not used for code in testing before it's merged into the master branch, so the way in which the configuration of a HPCC Platform for testing works by creating a brand new Image.

The first major hurdle is that there is no way to automate this, so that means that the user would have to do the same but this time on a virtual system provisioned by gitpod, which inherently has

a lesser level of control, then one would have over a system or subsystem (eg WSL) which they themselves have provisioned.

One of the biggest issues which we face while working with GitPod is the constant return to state which we need to perform, an example can be, say we want to run an installation of a dependency, that install needs to be mentioned in the yml file (It executes every time the VM is executed) or else we will have to manually run it

The Main issue which we encountered was the configuration and working of “CMake”.

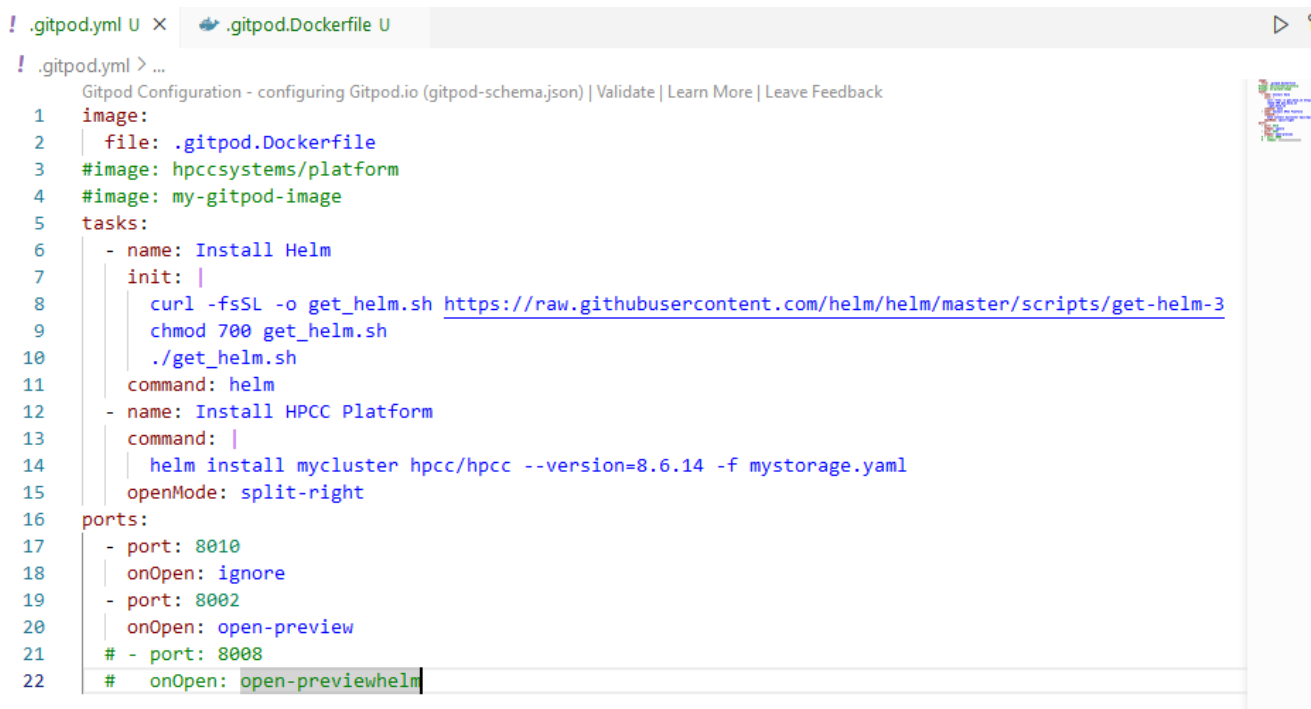
CMake is an open-source, cross-platform build system generator. It allows developers to specify the build process for their software projects using a platform-independent scripting language called CMakeLists.txt.

CMake then generates native build files (e.g., Makefiles on Unix-like systems or Visual Studio project files on Windows) based on these scripts, allowing the software to be built and compiled on different operating systems and build environments.

And inside that we have issues with the ‘vcpkg.cmake’ wherein The vcpkg.cmake file is a CMake script provided by the vcpkg package manager. It is typically used to integrate vcpkg with CMake-based projects, making it easier to manage dependencies and build projects that rely on external libraries.

```
[cmake]  
[cmake] CMake Error: CMake was unable to find a build program corresponding to "Unix Makefiles".  CMAKE_MAKE_PROGRAM is  
not set.  You probably need to select a different build tool.  
[cmake] CMake Error: CMAKE_C_COMPILER not set, after EnableLanguage  
[cmake] CMake Error: CMAKE_CXX_COMPILER not set, after EnableLanguage  
[cmake] -- Configuring incomplete, errors occurred!
```

Now, the method tried above is that of using a Docker Image for the build, in this case I used Docker Hub to collect the image and had to create a custom image which had git installed (That was a requirement by gitpod) but still the same would not be generated. Due to the CMake issues which were mentioned above.



```
! .gitpod.yml U x .gitpod.Dockerfile U
! .gitpod.yml > ...
Gitpod Configuration - configuring Gitpod.io (gitpod-schema.json) | Validate | Learn More | Leave Feedback
1 image:
2   file: .gitpod.Dockerfile
3 #image: hpccsystems/platform
4 #image: my-gitpod-image
5 tasks:
6   - name: Install Helm
7     init: |
8       curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
9       chmod 700 get_helm.sh
10      ./get_helm.sh
11     command: helm
12   - name: Install HPCC Platform
13     command: |
14       helm install mycluster hpcc/hpcc --version=8.6.14 -f mystorage.yaml
15     openMode: split-right
16 ports:
17   - port: 8010
18     onOpen: ignore
19   - port: 8002
20     onOpen: open-preview
21   # - port: 8008
22   #   onOpen: open-previewhelm
```

In the above image we can see that different methods were tried as well with my own modified image, with the image online and with the direct port set up.

The gitpod.Dockerfile was one created based on some online resources which mentioned this method to be useful for the set up. (Image on the next page)

The future integration of gitpod is a valid option but a suggestion can be made to have a look at a inhouse development of a similar platform wherein all the control can be on our end, and can also be a cheaper option then to go ahead with a organizational plan or collaboration with Gitpod.

 .gitpod.Dockerfile

[Validate](#) | [Learn More](#) | [Leave Feedback](#)

```
1 FROM gitpod/workspace-full
2
3 USER root
4
5 # Install dependencies for building HPCC Platform
6 RUN apt-get update \
7     && apt-get install -y --no-install-recommends \
8         build-essential \
9         cmake \
10        libicu-dev \
11        libboost-all-dev \
12        libldap2-dev \
13        libxslt1-dev \
14        uuid-dev \
15        libxml2-utils \
16        wget \
17    && apt-get clean \
18    && rm -rf /var/lib/apt/lists/*
19
20 # Clone HPCC Platform repository
21 RUN git clone https://github.com/hpcc-systems/HPCC-Platform.git /workspace/HPCC-Platform
22
23 .....
```



The integrations of Power BI and the Case Study of GitPod have been completed, the next section hold information on any references made above.

## 5 REFERENCE:

The table below lists all other documents referenced by this specification.

Document Title	Code	Source
Installing and Running the HPCC Systems Platform		<a href="#">link</a>
WsSQL ESP Web Service Guide Boca Raton Documentation		<a href="#">link</a>