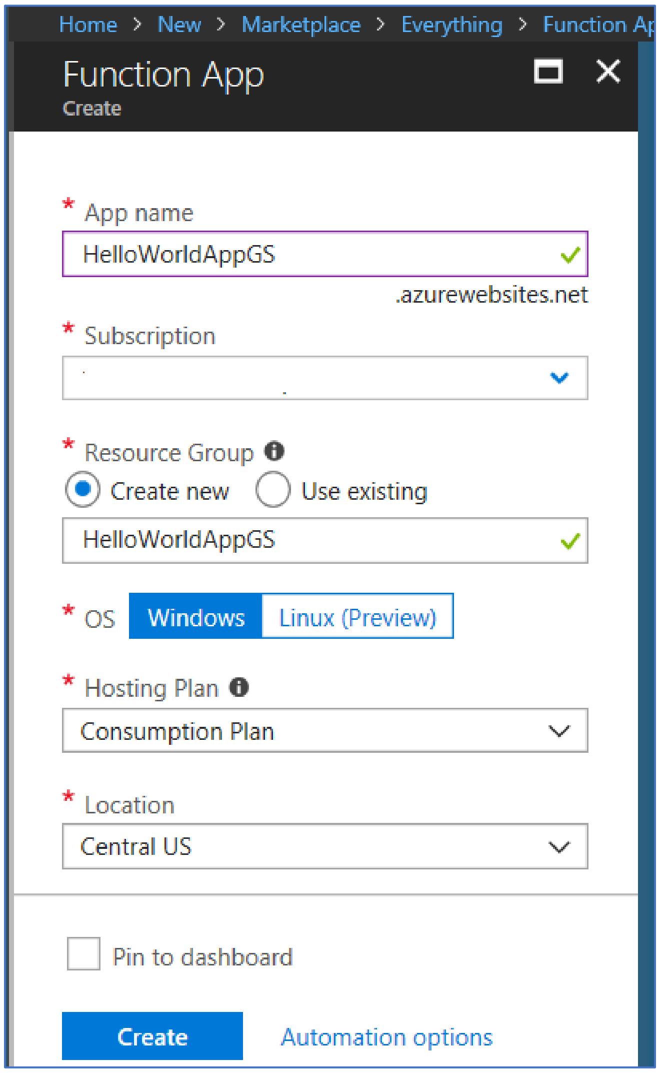
VSTS Extensions and Custom Applications

Create Custom Extension

Let us create a small Azure function and later call it from our VSTS extension.

Create Azure Function using Azure Portal

1. Go to Azure Portal by https://portal.azure.com, login to your account if not already logged
2. Click on Create a Resource, enter Function app in the search textbox and press enter.
3. Select Function app from the list and click on Create
4. Enter app name of your choice (it should be unique), select the subscription you are using, select location, resource group, storage account name and click on Create



* App name: HelloWorldAppGS .azurewebsites.net

* Subscription:

* Resource Group: [✓ Create new] Use existing — HelloWorldAppGS

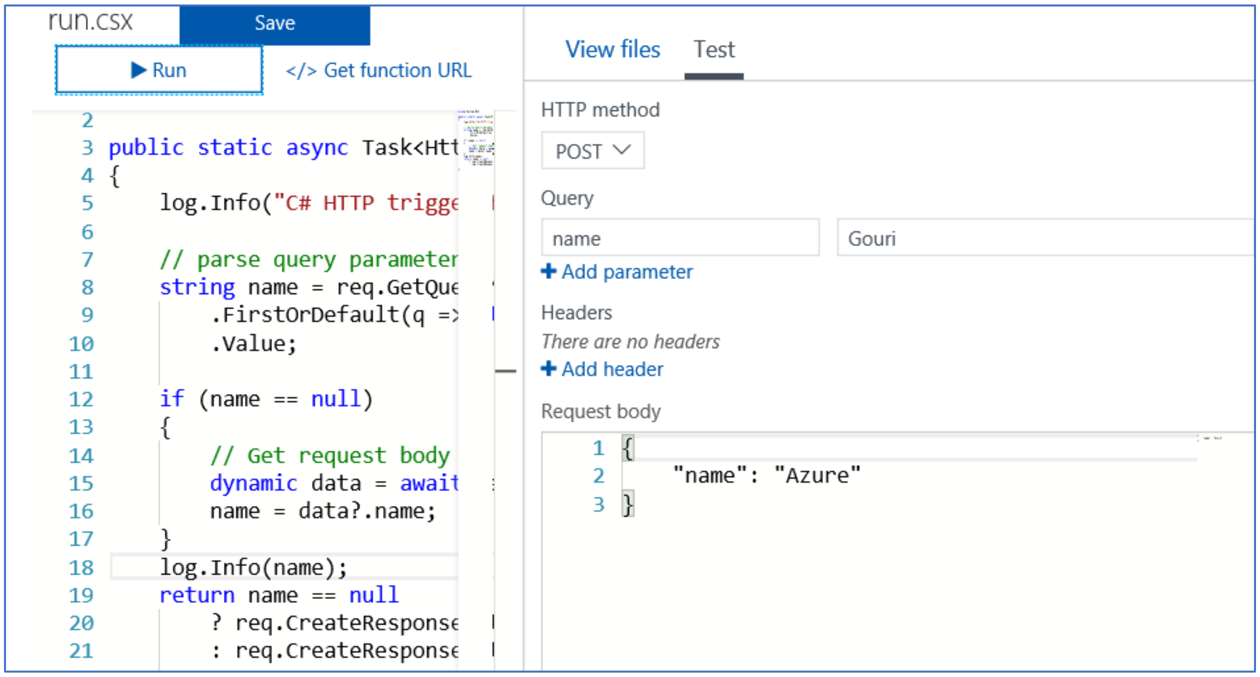
* OS: [✓ Windows] Linux (Preview)

* Hosting Plan: Consumption Plan

* Location: Central US

Pin to dashboard

5. After successful deployment click on Go To Resource, click on + sign for function and select Webhook + API (C#) function and click on Create this function
6. Keep the default code and you can Test the function either by clicking on Test or entering the url for the function append it with &name=<your name> in browser



```
public static async Task<Htt
{
    log.Info("C# HTTP trigge

    // parse query parameter
    string name = req.GetQue
        .FirstOrDefault(q =>
        .Value;

    if (name == null)
    {
        // Get request body
        dynamic data = await
        name = data?.name;
    }
    log.Info(name);
    return name == null
        ? req.CreateResponse
        : req.CreateResponse
```

HTTP method: POST

Query: name | Gouri

+ Add parameter

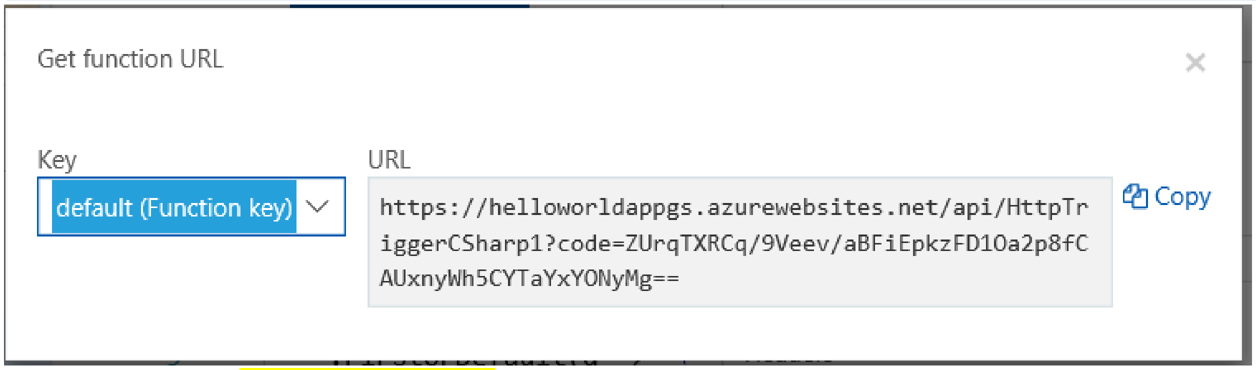
Headers

There are no headers

+ Add header

Request body

```
{
    "name": "Azure"
}
```



Key: default (Function key)

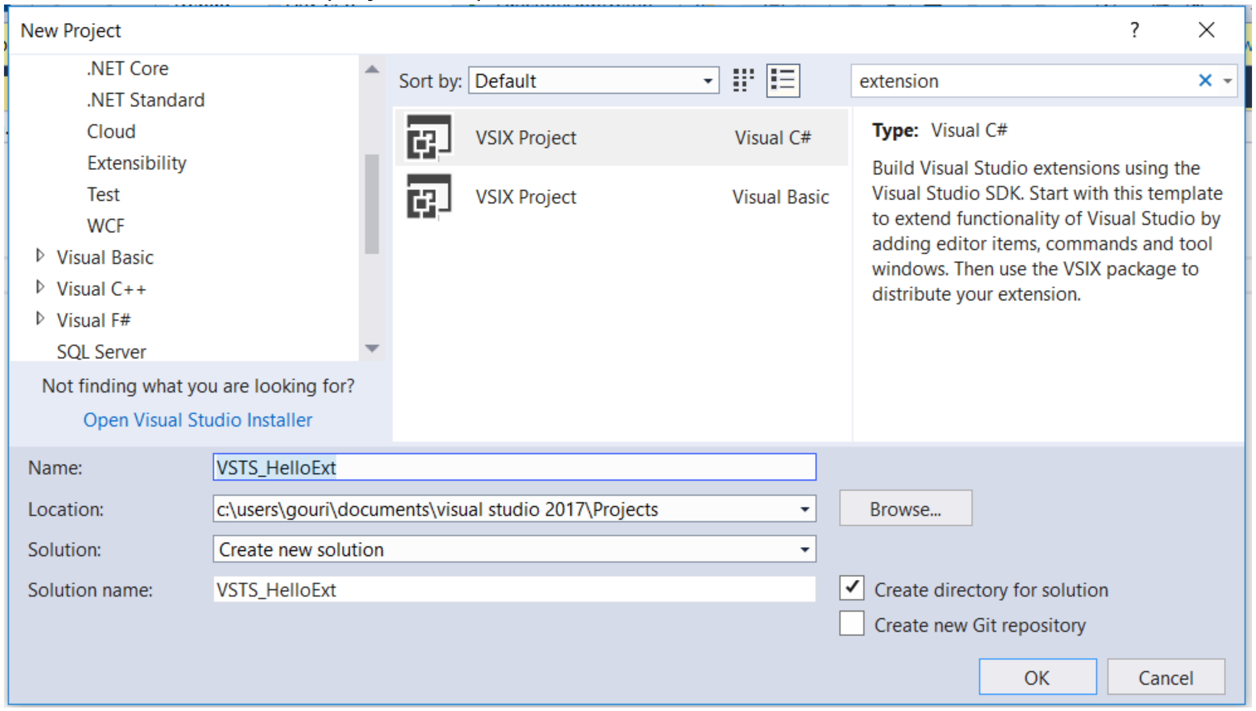
URL: https://helloworldappgs.azurewebsites.net/api/HttpTriggerCSharp1?code=ZUrqTXRCq/9Veev/aBFiEpkzFD1Oa2p8fCAUxnyWh5CYTaYxYONyMg==

7. Enter url as https:/<your function url>&name=<your name>
8. The function is created and working successfully.
9. We need to change CORS (Cross Origin Resource Sharing) for this function as we will be calling this function from VSTS extension. Otherwise we get error for Access-control-allow-origin error. In order to do that go to Select Platform Features for the function app and select CORS, remove all the available allowed origins and only add * and save the setting

Create VSTS Extension

1. Install node from this link
2. TFS Cross Platform Command Line Interface (tfx-cli). This can be installed with npm by giving command npm i -g tfx-cli from Node.js
3. Enter command npm install vss-web-extension-sdk to install extension sdk from the directory you have installed node.js in.
4. Home directory for the project you are working on

5. Visual Studio 2017 has a project template for extension



New Project

Sort by: Default

extension

VSIX Project — Visual C#

VSIX Project — Visual Basic

Type: Visual C#

Build Visual Studio extensions using the Visual Studio SDK. Start with this template to extend functionality of Visual Studio by adding editor items, commands and tool windows. Then use the VSIX package to distribute your extension.

Not finding what you are looking for?

Open Visual Studio Installer

Name: VSTS_HelloExt

Location: c:\users\gouri\documents\visual studio 2017\Projects

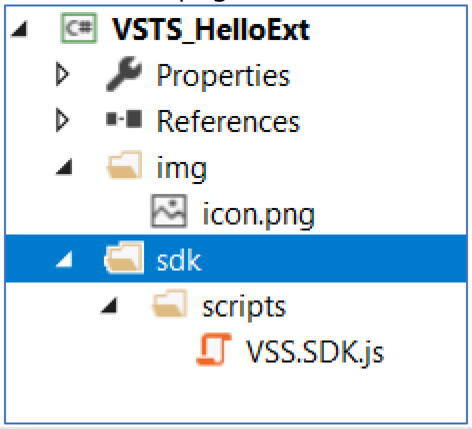
Solution: Create new solution

Solution name: VSTS_HelloExt

[✓] Create directory for solution

[ ] Create new Git repository

6. Create following folder structure and copy icon.png in the img folder (this needs to have name as icon.png and size 128X128 pixels). Copy VSS.SDK.js file under scripts folder



```
VSTS_HelloExt
  Properties
  References
  img
    icon.png
  sdk
    scripts
      VSS.SDK.js
```

7. We need 2 files, one html and another json. Json file store manifest for your extension. Add json file named vss-extension.json with following content

```
{ "manifestVersion": 1,
"id": "mydemo-extension",
"version": "0.1.4",
"name": "Hello World extension",
"description": "Demo Extension.",
"publisher": "<your publisher name>",
"targets": [{"id": "Microsoft.VisualStudio.Services"}],
"icons": {"default": "img/icon.png"},
"contributions": [{"id": "ssgs.HelloWorld",
   "type": "ms.vss-web.hub",
      "description": "Adds a 'Hello' hub to the Work hub group.",
      "targets": ["ms.vss-work-web.work-hub-group"],
      "properties": {"name": "Hello",
        "order": 99,
        "uri": "hello.html"}}],
  "scopes": ["vso.work"],
  "files": [{"path": "hello.html",
      "addressable": true},
    { "path": "sdk/scripts",
      "addressable": true},
    { "path": "img/icon.png",
      "addressable": true}
]}
```

8. This is going to add one more hub to the work hub

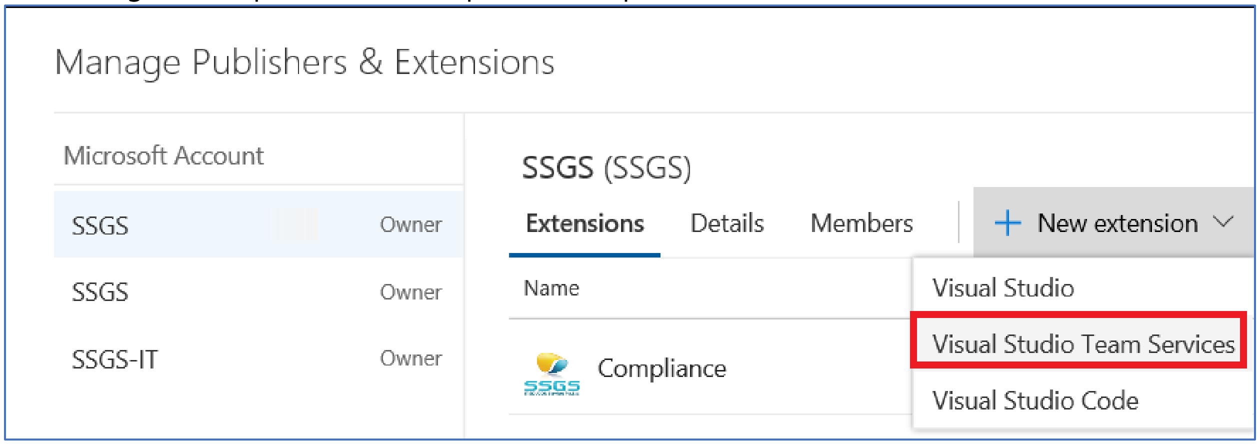
We also need to provide what will be displayed in the new tab. This is done by using html file

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Hello World</title>
    <script src="sdk/scripts/VSS.SDK.js"></script>
</head>
<body>
    <script type="text/javascript">VSS.init();</script>
    <h1>Hello World</h1>
    <form name="form1">
        <div id="div1" style="margin-left:120px">
            <input type='text' id='msgTxt' style='font-size:16pt;font-family:Segoe UI; width:800px;
border-style:none; background-color:white' value="Demo Extension" />
            <br /> <br />
            <input type="button" id="Click" value="Click" style='font-size:16pt;font-family:Segoe UI;
width:150px' onClick="ClickMe()" />
        </div>
    </form>
    <script>
    function ClickMe() {
        var collection = VSS.getWebContext().collection.name;
        var projectName = VSS.getWebContext().project.name;
        $.ajax(
            {
                dataType: "json",
                async: true,
                contentType: "application/json; charset=utf-8",
                headers: {
                    Accept: "application/json",
                    "Access-Control-Allow-Origin": "*"
                },
                type: "GET",
                data: "",
                url: "<your function url>=&name=<your name>",
                success: function (data) {
                    document.getElementById("msgTxt").value = data;
                },
                error: function (xhr, ajaxOptions, thrownError) {
                    document.getElementById("msgTxt").value = xhr.readyState + " Error: " +
thrownError;
                }
            });
        }
    </script>
    <script type="text/javascript">
        VSS.init({explicitNotifyLoaded: true,
            usePlatformScripts: true,
            usePlatformStyles: true});
        VSS.ready(function () {var collection = VSS.getWebContext().collection.name;
            var projectName = VSS.getWebContext().project.name;
            var projId = VSS.getWebContext().project.id; })
    </script>
    <script type="text/javascript">
        VSS.notifyLoadSucceeded();
    </script>
</body>
</html>
```

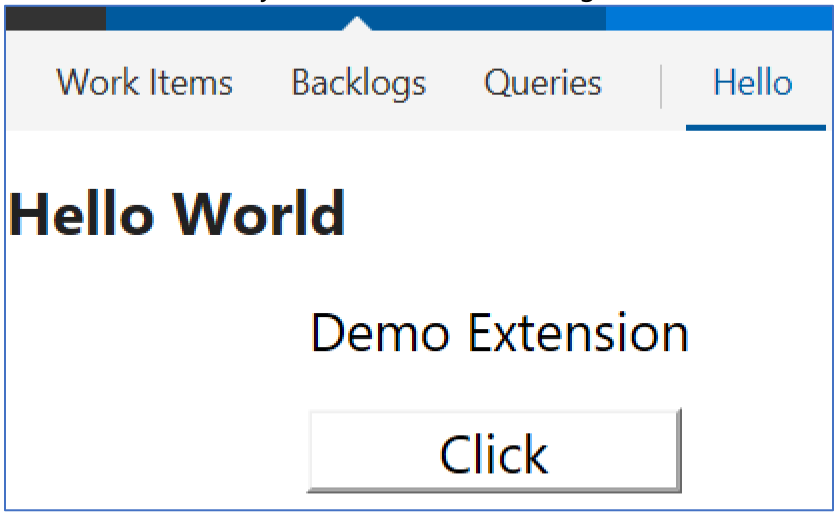
Observe that there will be one message displayed which will be returned by the function we have written in azure. In case you want to write a little complex function which takes VSTS instance name and project name from where the function is being called you can use collName and projName variables already declared and assigned.

9. Now we have to create vsix file for the extension. Go to node.js command prompy, browse to the folder where your extension files are present and enter command
   tfx extension create --manifest-globs vss-extension.json
10. Next step is to create publisher. We need to create the publisher first. Go to the MarketPlace publisher link https://marketplace.visualstudio.com/manage/. We need to provide the details of our publisher company. Like in this case I am using SSGS. You will have to review the agreement and then create the publisher

11. Now is the time to publish the extension. Select Visual Studio Team Services extension and either drag and drop the vsix file or provide the path



12. Once the vsix is published select share and provide the vsts account name to share extension.
13. Go to Extensions tab for the VSTS account and you can see the new extension appears
14. Go to work tab and you will observe following



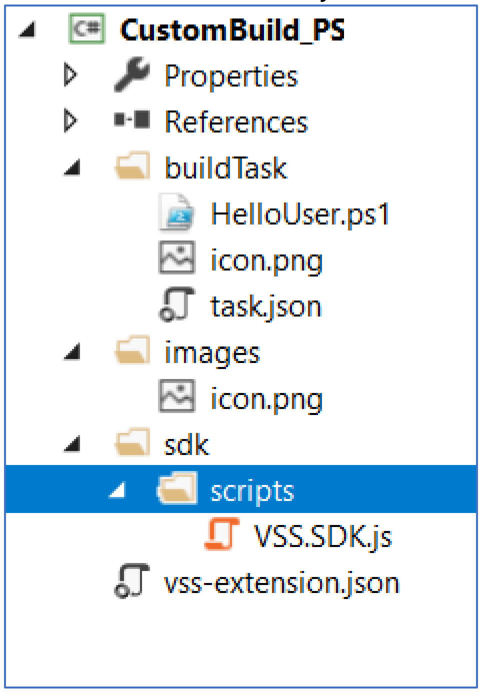
15. Click on command button and Hello <your name> will be displayed

Create Custom Build or Release Task

This exercise assumes that you already have created VSTS extension so have node, TFS-CLI installed and created a publisher. We require task metadata file, extension manifest file, PowerShell script for task

1. Select Visual Studio project for vsix and provide name for it

2. Remove the un-necessary files and create folder structure as follows



```
CustomBuild_PS
    Properties
    References
    buildTask
        HelloUser.ps1
        icon.png
        task.json
    images
        icon.png
    sdk
        scripts
            VSS.SDK.js
    vss-extension.json
```

The icon.png (name must be same) needs to be in the same folder where task.json exists. It needs to be of size 32 X 32 pixels.

3. Add code to task.json as follows

```
{
       "id": "f3bfc6d8-fcf1-4860-a135-8c52d223fc24",
        "name": "helloUser",
       "friendlyName": "Demo Hello Task",
      "description": "This task is created for demo purpose",
       "author": "<your publisher name>",
        "helpMarkDown": "this task takes one parameter and shows output",
        "category": "Test",
        "visibility": [ "Build", "Release" ],
       "demands": [],
        "version": {"Major": "1",
          "Minor": "1",
         "Patch": "0"},
       "minimumAgentVersion": "1.95.0",
       "inputs": [{
          "name": "uName",
           "type": "string",
       "label": "User Name",
      "defaultValue": "Gouri",
       "required": true}],
       "execution": {"PowerShell": { "target": "HelloUser.ps1" }}
      }
```

Observe the id is any GUID which can be generated on the fly for the task. Visibility tells that this task will be available to Build as well as Release. The inputs are for PowerShell script which will be executed with the command shown.

4. PowerShell script is a small commandlet

```
CmdletBinding()]
 param(
    [string]$uName
)
try {
     Write-Host "Executing the custom build task"
     Write-Host "Hello $uName How are you doing?"
 } finally {
    Write-Host "Custom build task over"
}
```

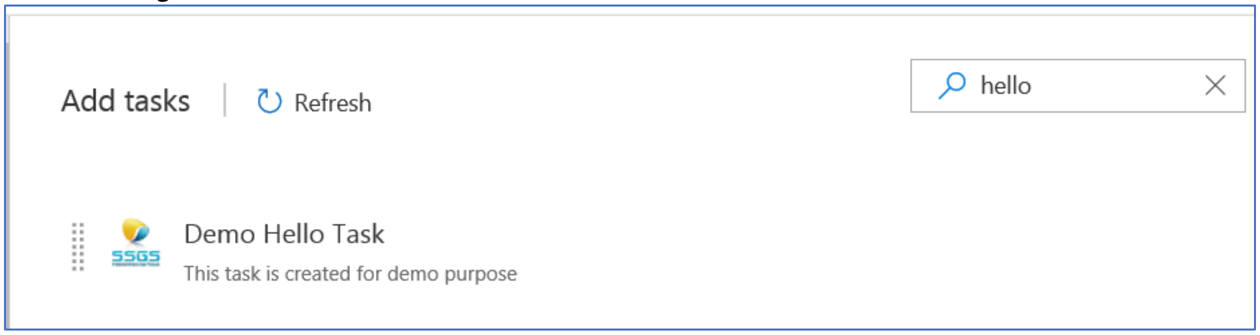
5. Vss-extension.json file has following contents

```
{
  "manifestVersion": 1,
  "id": "helloUser",
  "name": "SSGS Hello",
  "version": "1.0.5",
  "publisher": "<your publisher name>",
  "targets": [{"id": "Microsoft.VisualStudio.Services"}],
  "description": "Tools for building with SSGS. Includes one build task.",
  "categories": ["Build and release"],
  "icons": {"default": "images/icon.png"},
  "files": [{"path": "buildtask"}],
  "contributions": [{ "id": "custom-build-task",
      "type": "ms.vss-distributed-task.task",
      "targets": ["ms.vss-distributed-task.tasks"],
      "properties": { "name": "buildtask"}}
  ]}
```

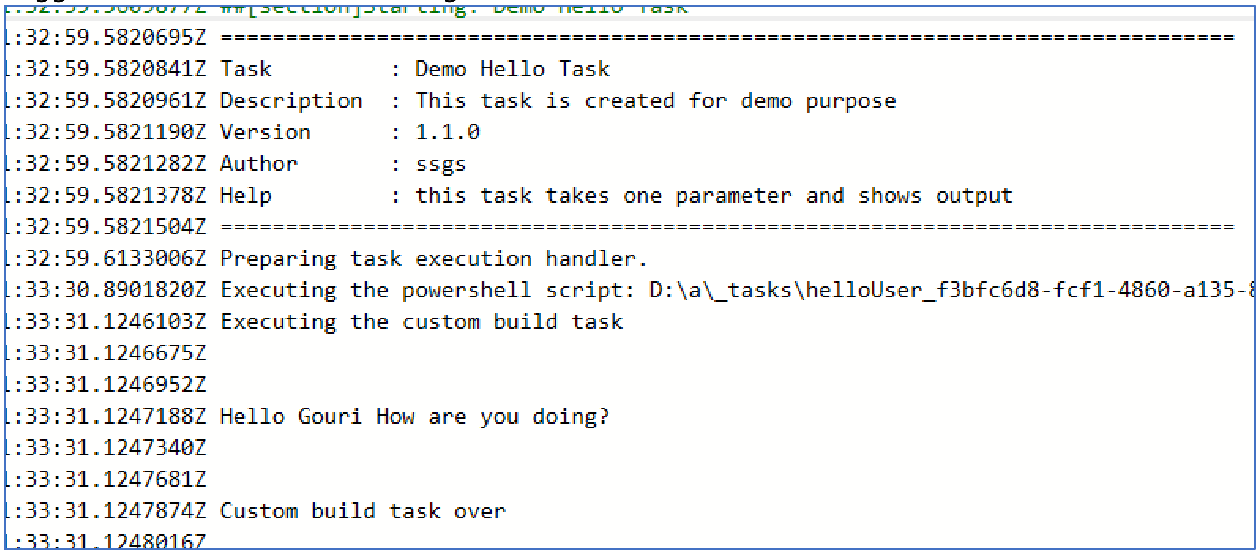
Observe that the files are located in path buildtask.

6. Now that we have provided all required files we can create the vsix by following command
tfx extension create --manifest-globs vss-extension.json
7. Now publish the extension as we have done in custom extension and share it with VSTS account.

8. Open any existing build definition for edit and add task, type hello in search box and you will see following



9. Trigger the build and the task will get executed

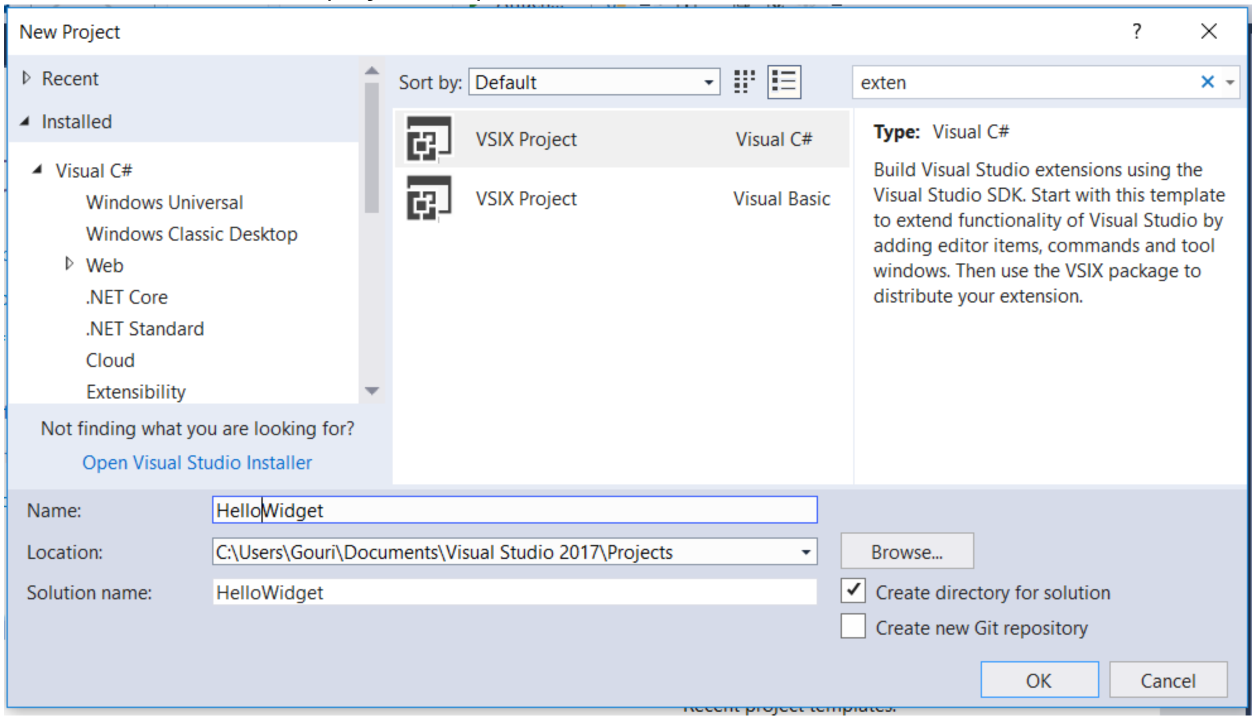


```
:32:59.5820695Z ==============================================================================
:32:59.5820841Z Task         : Demo Hello Task
:32:59.5820961Z Description  : This task is created for demo purpose
:32:59.5821190Z Version      : 1.1.0
:32:59.5821282Z Author       : ssgs
:32:59.5821378Z Help         : this task takes one parameter and shows output
:32:59.5821504Z ==============================================================================
:32:59.6133006Z Preparing task execution handler.
:33:30.8901820Z Executing the powershell script: D:\a\_tasks\helloUser_f3bfc6d8-fcf1-4860-a135-
:33:31.1246103Z Executing the custom build task
:33:31.1246675Z
:33:31.1246952Z
:33:31.1247188Z Hello Gouri How are you doing?
:33:31.1247340Z
:33:31.1247681Z
:33:31.1247874Z Custom build task over
:33:31.1248016Z
```

Exercise: Creating Custom Widget (implemented as Extension)

1. In this exercise we will create a simple widget which just displays Hello World on the Dashboard if all things are successful. Later we will increase the complexity of the item(s) to be displayed.
2. Ensure that you either have Visual Studio 2017 or Visual Studio Code (link) which will be used as text editor
3. We already are working with VSTS account
4. Latest version of node
5. TFS Cross Platform Command Line Interface (tfx-cli). This can be installed with npm by giving command npm i -g tfx-cli from Node.js
6. Enter command npm install vss-web-extension-sdk to install extension sdk from the directory you have installed node.js in.
7. Home directory for the project you are working on

8. Visual Studio 2017 has a project template for extension



Type: Visual C#

Build Visual Studio extensions using the Visual Studio SDK. Start with this template to extend functionality of Visual Studio by adding editor items, commands and tool windows. Then use the VSIX package to distribute your extension.

Name: HelloWidget

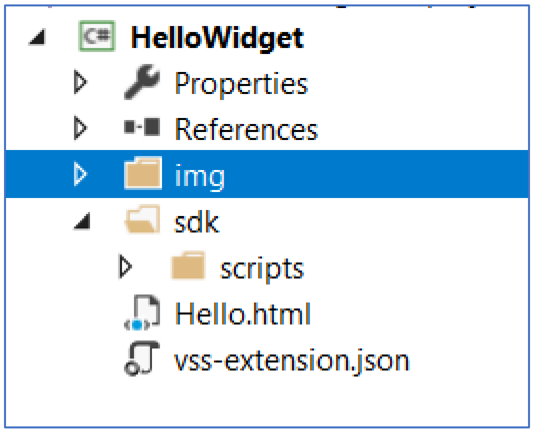
Location: C:\Users\Gouri\Documents\Visual Studio 2017\Projects

Solution name: HelloWidget

[✓] Create directory for solution

[ ] Create new Git repository

9. I have created following folder structure (home directory) and added entries in html and json file accordingly



```
HelloWidget
    Properties
    References
    img
    sdk
        scripts
    Hello.html
    vss-extension.json
```

We need to add an image to img folder that must be of 98px X 98px. There needs another png file of 330px X 160px for TFS 2015 Update 3. We also need to provide catalog url property, which is explained later.

10. Copy file VSS.SDK.min.js from drive:\<node installation folder>\ node_modules\vss-web-extension-sdk\bin and paste in under scripts folder
11. Rename existing html to Hello.html and remove remaining un-necessary files. The name of function we are using in this file needs to be same in json file later
12. Following is the html contents in Hello.html

```
<!DOCTYPE html>

<html>
```

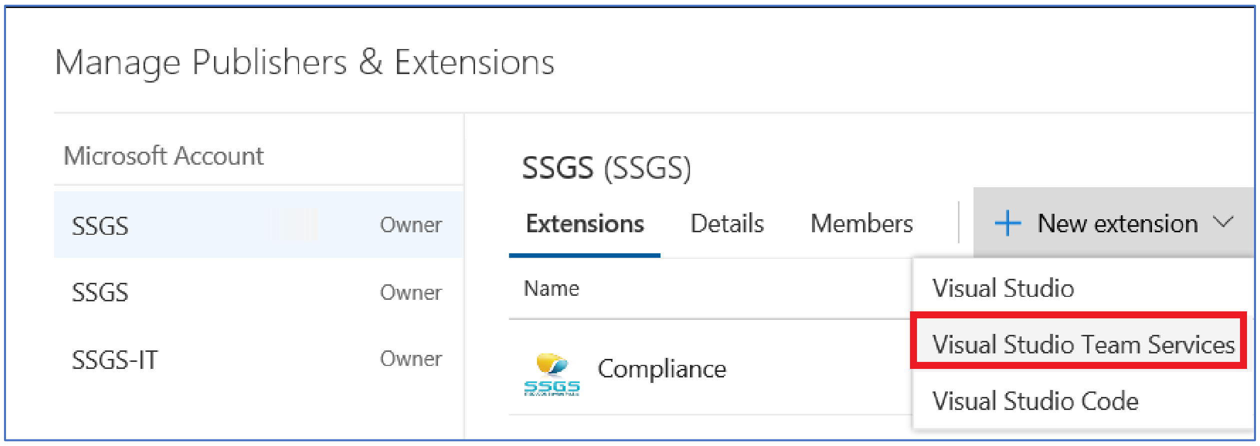
```
<head>
   <script src="sdk/scripts/VSS.SDK.min.js"></script>
   <script type="text/javascript">
      VSS.init({explicitNotifyLoaded: true,
         usePlatformStyles: true});
      VSS.require("TFS/Dashboards/WidgetHelpers", function (WidgetHelpers) {
         WidgetHelpers.IncludeWidgetStyles();
         VSS.register("HelloWorldGS", function () {
            return {
               load: function (widgetSettings) {
                  var $title = $('h2.title');
                  $title.text('Hello World');
                  return WidgetHelpers.WidgetStatusHelper.Success();}}});
         VSS.notifyLoadSucceeded();
      });
   </script>
</head>
<body>
   <div class="widget">
      <h2 class="title"></h2>
   </div>
</body>
</html>
```

13. JSON File

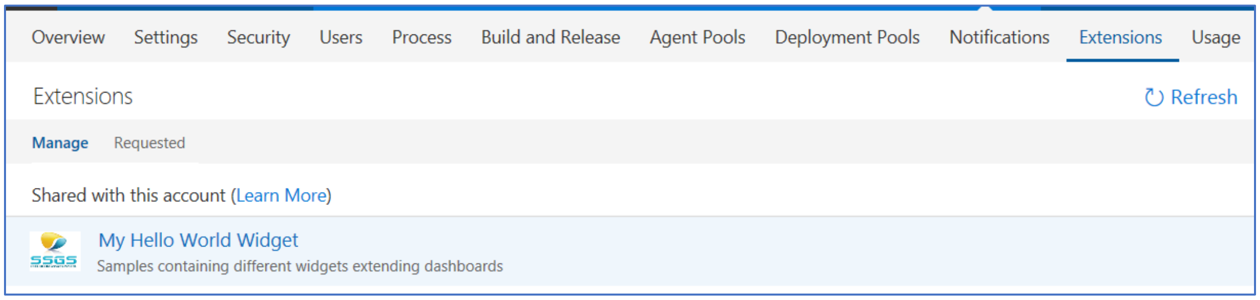
14. Add a json file named vss-extension.json with following

```
{ "manifestVersion": 1,
  "id": "vsts-extensions-HelloExtensions",
  "version": "1.0.4",
  "name": "My Hello World Widget",
  "description": "Samples containing different widgets extending dashboards",
  "publisher": "SSGS",
  "targets": [{"id": "Microsoft.VisualStudio.Services"}],
  "icons": {"default": "img/logo.png"},
  "contributions": [{"id": "HelloWorldGS",
     "type": "ms.vss-dashboards-web.widget",
     "targets": [
       "ms.vss-dashboards-web.widget-catalog"],
     "properties": {
       "name": "My Hello World Widget",
       "description": "My first widget",
       "catalogIconUrl:": "img/template.png",
       "previewImageUrl": "img/template2.png",
       "uri": "hello.html",
       "supportedSizes": [{"rowSpan": 1,
          "columnSpan": 2}],
       "supportedScopes": [ "project_team" ]}}],
  "files": [{"path": "hello.html",
     "addressable": true},
   { "path": "sdk/scripts",
     "addressable": true},
   { "path": "img",
     "addressable": true }]
}
```

15. Remember contributions id needs to be the same as function used in html file. Type of the contribution has to be ms.vss-dashboards-web.widget for all widgets. Similarly targets need to be [ms.vss-dashboards-web.widget-catalog] for all widgets. catalogUrl is the relative path you have to give for logo. previewUrl is for TFS Update 3 as mentioned previously. Uri is the relative path for the html file. supportedSize is for the widget size and it can be in terms of rows and columns and maximum 4X4. supportedScopes is currently available only for Dashboards so the value has to be project_team
16. Now we need to package our extension, publish it and test. Create .vsix file from tfx-cli by giving following command tfx extension create --manifest-globs vss-extension.json (remember to give the command from the folder where html and json are available)
17. We need to create the publisher first. Go to the MarketPlace publisher link https://marketplace.visualstudio.com/manage/. We need to provide the details of our publisher company. Like in this case I am using SSGS. You will have to review the agreement and then create the publisher
18. Now is the time to publish the extension. Select Visual Studio Team Services extension and either drag and drop the vsix file or provide the path



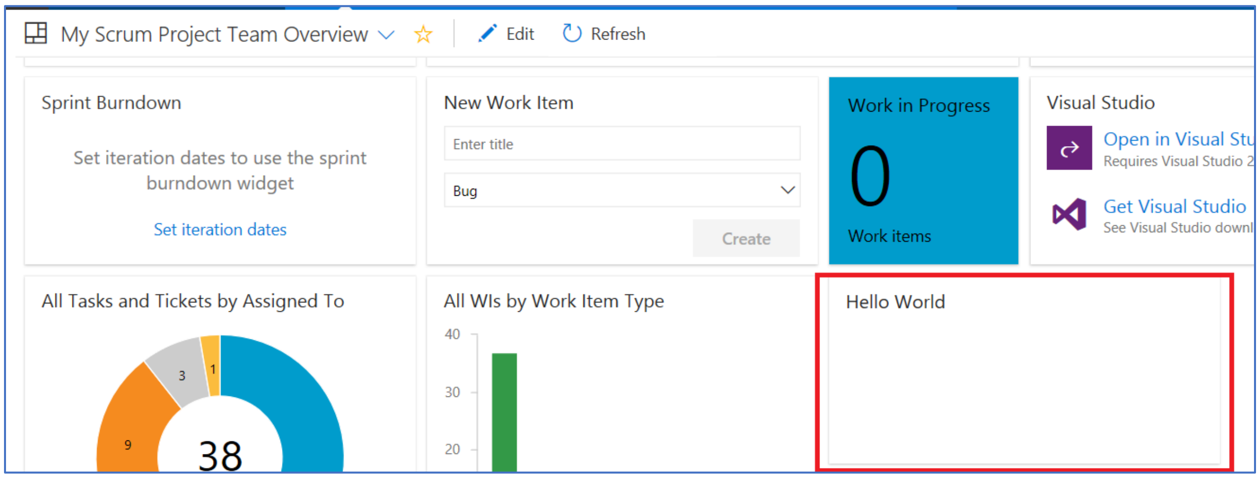
19. Once the vsix is published select share and provide the vsts account name to share extension.
20. Go to Extensions tab for the VSTS account and you can see the new extension appears



Shared with this account (Learn More)

My Hello World Widget
Samples containing different widgets extending dashboards

21. Install it for the current account, go to Dashboard and add the widget to it.



22. Let us add some more functionality to the widget. We need to change the json file as well as add another html file. Add html file named Hello2.html
23. In this we need to access VSTS resources (to get query) so need to add scope to vso.work in json file as follows
"scopes": ["vso.work"]
24. We need to make REST call, we need to get the information about a query in which we provide the project is. For this we need a variable to store project id. I have created a query in Shared folder with the name All Tasks. The complete html file looks as follows

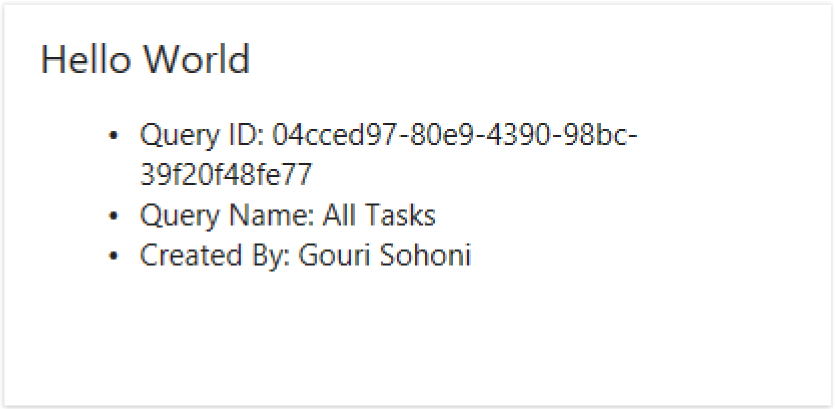
```
<!DOCTYPE html>

<html>
<head>
    <script src="sdk/scripts/VSS.SDK.min.js"></script>
    <script type="text/javascript">
        VSS.init({explicitNotifyLoaded: true,
            usePlatformStyles: true});
        VSS.require(["TFS/Dashboards/WidgetHelpers", "TFS/WorkItemTracking/RestClient"],
            function (WidgetHelpers, TFS_Wit_WebApi) {
                WidgetHelpers.IncludeWidgetStyles();
                VSS.register("HelloWorldGS2", function () {
                    var projectId = VSS.getWebContext().project.id;
                    var getQueryInfo = function (widgetSettings) {
                     // Get a WIT client to make REST calls to VSTS
                     return TFS_Wit_WebApi.getClient().getQuery(projectId, "Shared Queries/All Tasks")
                         .then(function (query) {
                             // Create a list with query details
                             var $list = $('<ul>');
                             $list.append($('<li>').text("Query ID: " + query.id));
                             $list.append($('<li>').text("Query Name: " + query.name));
                             $list.append($('<li>').text("Created By: " + (query.createdBy ?
query.createdBy.displayName: "<unknown>") ));
                             // Append the list to the query-info-container
                    var $container = $('#query-info-container');
                    $container.empty();
                    $container.append($list);
                    // Use the widget helper and return success as Widget Status
                    return WidgetHelpers.WidgetStatusHelper.Success();
                }, function (error) {
                      // Use the widget helper and return failure as Widget Status
                      return WidgetHelpers.WidgetStatusHelper.Failure(error.message);});}
            return {load: function (widgetSettings) {
                    // Set your title
                    var $title = $('h2.title');
                    $title.text('Hello World');
                    return getQueryInfo(widgetSettings);}
            }});
        VSS.notifyLoadSucceeded();});
    </script>
</head>
<body>
    <div class="widget">
        <h2 class="title"></h2>
        <div id="query-info-container"></div>
    </div>
</body>
</html>
```

25. The json fie looks as follows

```
{
  "manifestVersion": 1,
  "id": "vsts-extensions-HelloExtensions",
  "version": "1.0.4",
  "name": "My Hello World Widget",
  "description": "Samples containing different widgets extending dashboards",
  "publisher": "SSGS",
  "targets": [ { "id": "Microsoft.VisualStudio.Services" } ],
  "icons": { "default": "img/logo.png" },
  "contributions": [
    {
      "id": "HelloWorldGS",
      "type": "ms.vss-dashboards-web.widget",
      "targets": [
        "ms.vss-dashboards-web.widget-catalog"
      ],
      "properties": {
        "name": "My Hello World Widget",
        "description": "My first widget",
        "catalogIconUrl:": "img/template.png",
        "previewImageUrl": "img/template2.png",
        "uri": "hello.html",
        "supportedSizes": [
          {
```

```
        "rowSpan": 1,
        "columnSpan": 2
       }
     ],
     "supportedScopes": [ "project_team" ]
    }
  },
  {
   "id": "HelloWorldGS2",
   "type": "ms.vss-dashboards-web.widget",
   "targets": [
     "ms.vss-dashboards-web.widget-catalog"
   ],
   "properties": {
     "name": "Hello World Widget 2 (with API)",
     "description": "My second widget",
     "previewImageUrl": "img/templateHello2.png",
     "uri": "hello2.html",
     "supportedSizes": [{"rowSpan": 4,
        "columnSpan": 4}],
     "supportedScopes": [ "project_team" ]}
  }],
 "files": [{"path": "hello.html",
   "addressable": true},
  { "path": "hello2.html",
   "addressable": true},
  { "path": "sdk/scripts",
   "addressable": true},
  { "path": "img",
   "addressable": true}],
 "scopes": ["vso.work"]
}
```

After successful upload, publish, share and adding it to the Dashboard it appears as follows



Hello World

- Query ID: 04cced97-80e9-4390-98bc-39f20f48fe77
- Query Name: All Tasks
- Created By: Gouri Sohoni

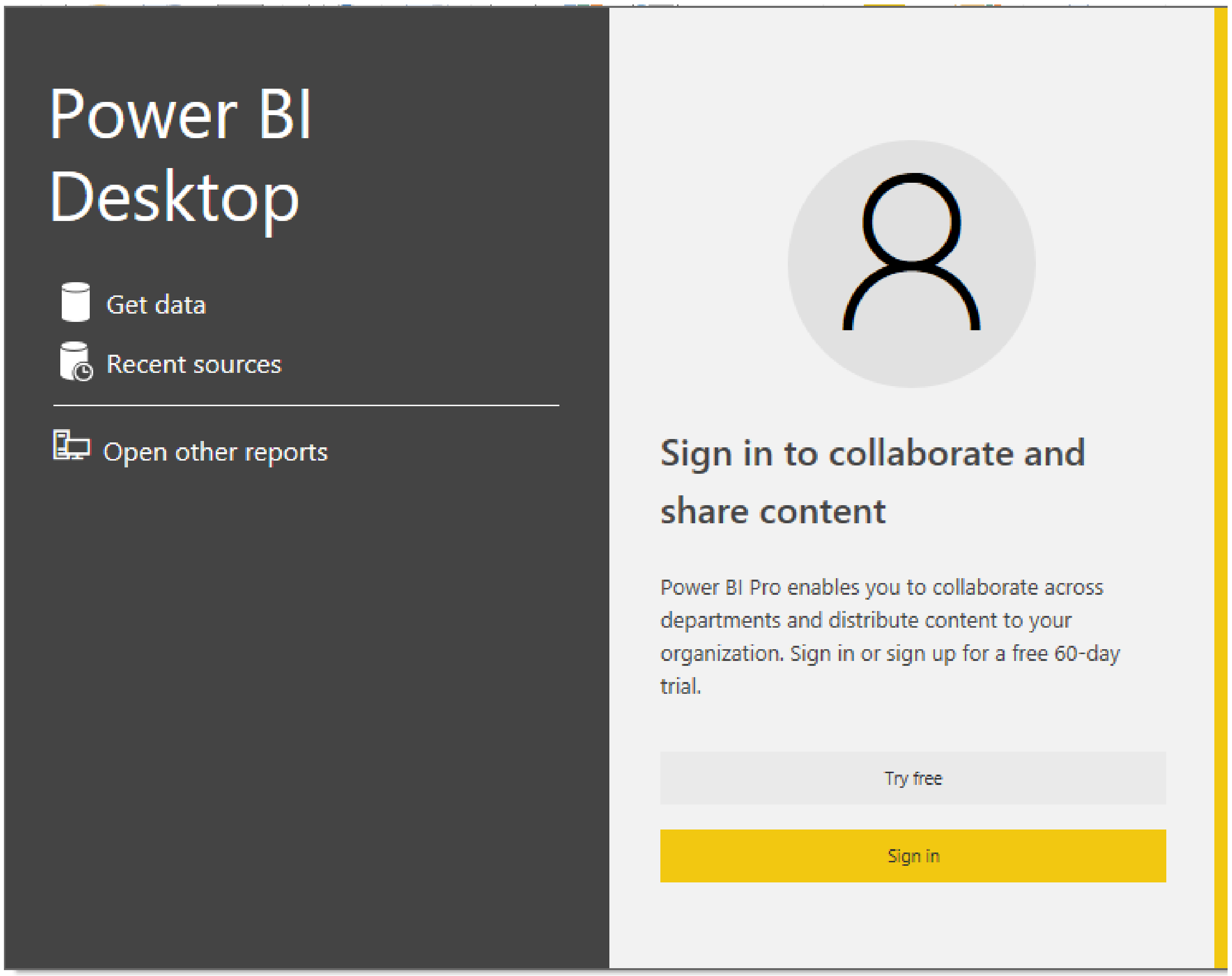
Create Power BI Report with a default Analytics View

Pre-requisites: Power BI Desktop, VSTS Analytics Extension, Permission to access Analytics Service

3 options: Connect using VSTS Data Connection, connect to VSTS using Power BI OData Connector, connect using VSTS functions

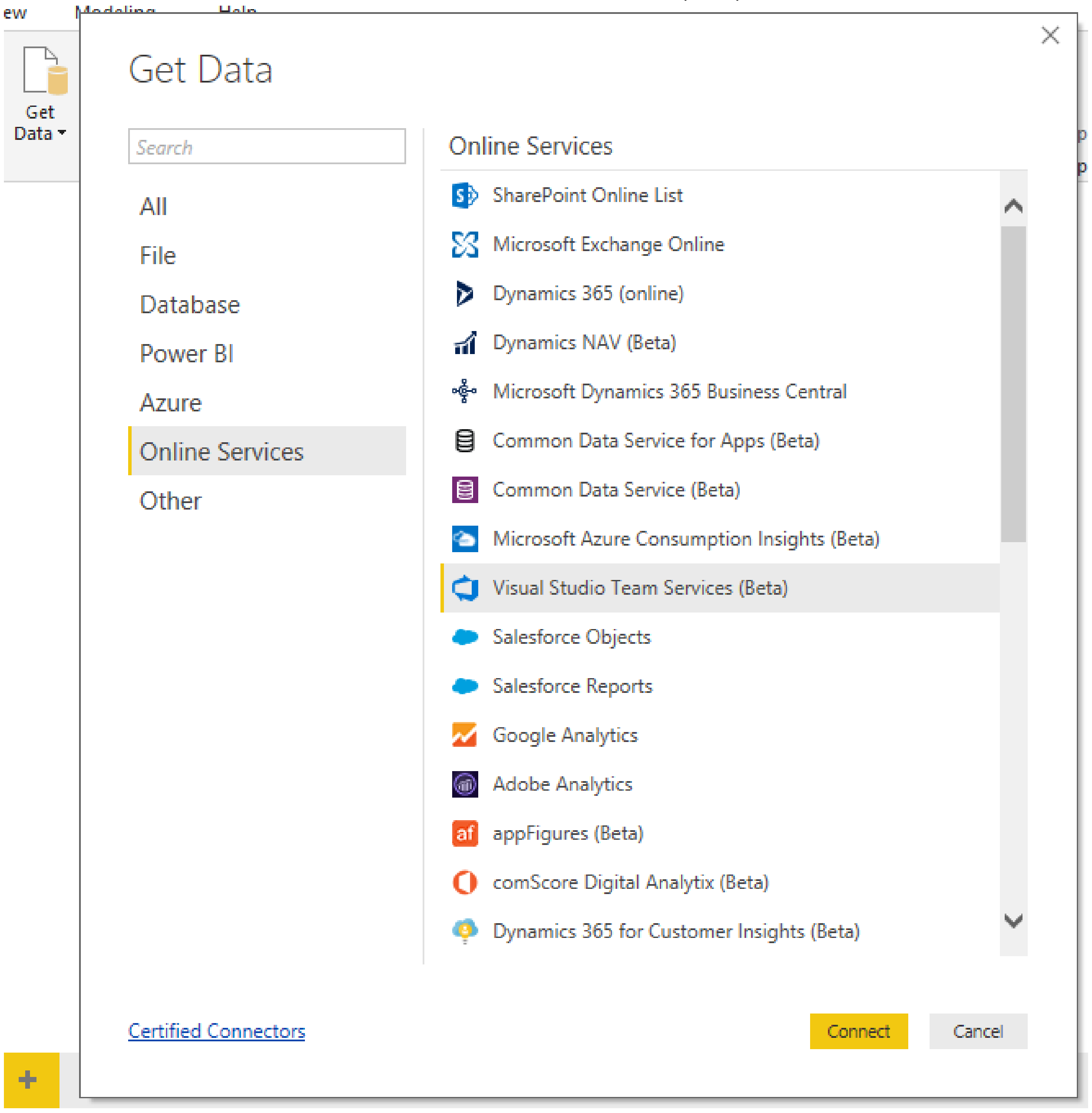
Connect using VSTS Data Connection (works with Analytics View)

1. Launch Power BI Desktop

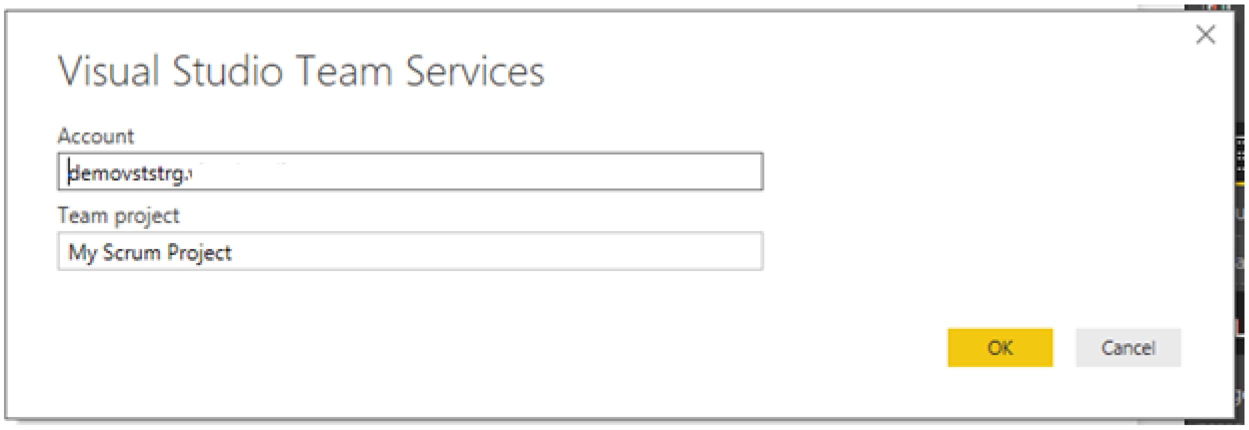


Power BI Pro enables you to collaborate across departments and distribute content to your organization. Sign in or sign up for a free 60-day trial.

2. Click on Get Data – Online Services – Visual Studio Team Services (Beta)


Get Data
Search
All
File
Database
Power BI
Azure
Online Services
Other
Online Services
SharePoint Online List
Microsoft Exchange Online
Dynamics 365 (online)
Dynamics NAV (Beta)
Microsoft Dynamics 365 Business Central
Common Data Service for Apps (Beta)
Common Data Service (Beta)
Microsoft Azure Consumption Insights (Beta)
Visual Studio Team Services (Beta)
Salesforce Objects
Salesforce Reports
Google Analytics
Adobe Analytics
appFigures (Beta)
comScore Digital Analytix (Beta)
Dynamics 365 for Customer Insights (Beta)
Certified Connectors
Connect
Cancel

3. Provide Account and Team Project



Visual Studio Team Services

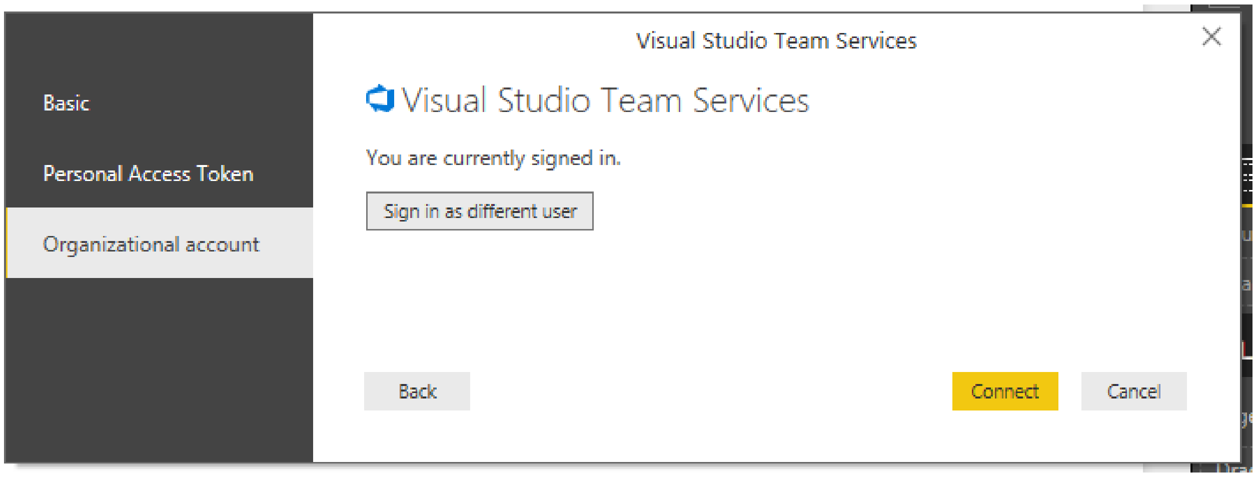
Account

demovststrg.

Team project

My Scrum Project

Sign in to by using Basic or Personal Access Token

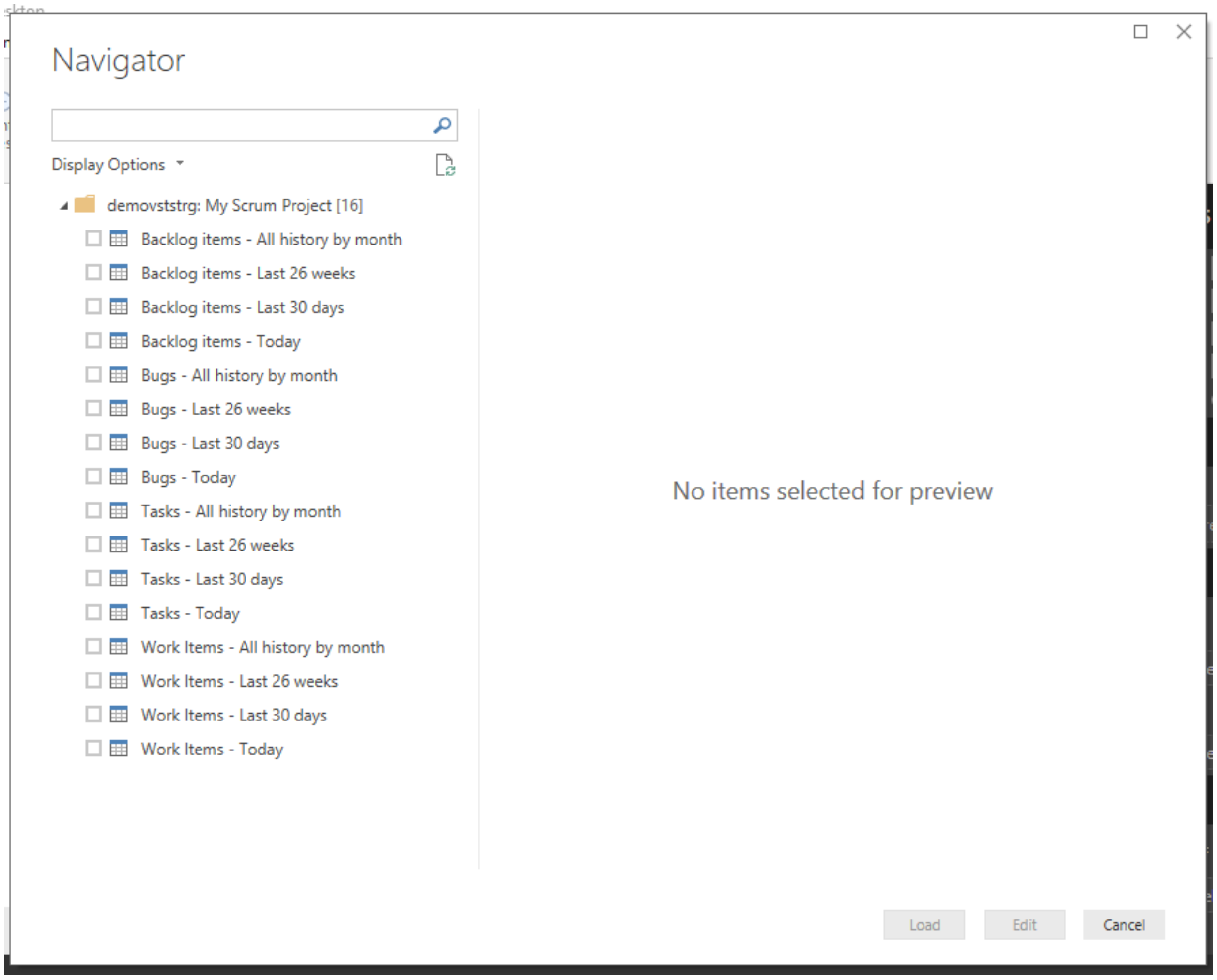


Basic
Personal Access Token
Organizational account
Visual Studio Team Services
Visual Studio Team Services

You are currently signed in.

Sign in as different user
Back
Connect
Cancel

4. After successful connection all Views will be displayed



Navigator

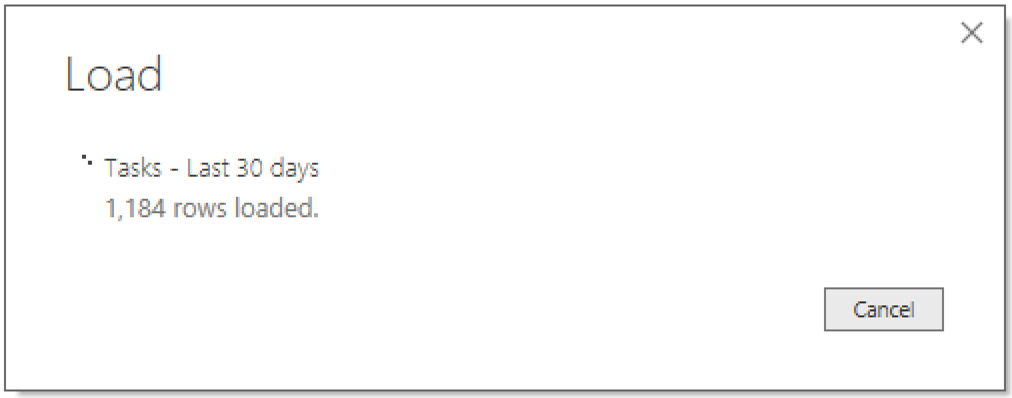
Display Options

demovststrg: My Scrum Project [16]

- Backlog items - All history by month
- Backlog items - Last 26 weeks
- Backlog items - Last 30 days
- Backlog items - Today
- Bugs - All history by month
- Bugs - Last 26 weeks
- Bugs - Last 30 days
- Bugs - Today
- Tasks - All history by month
- Tasks - Last 26 weeks
- Tasks - Last 30 days
- Tasks - Today
- Work Items - All history by month
- Work Items - Last 26 weeks
- Work Items - Last 30 days
- Work Items - Today

No items selected for preview

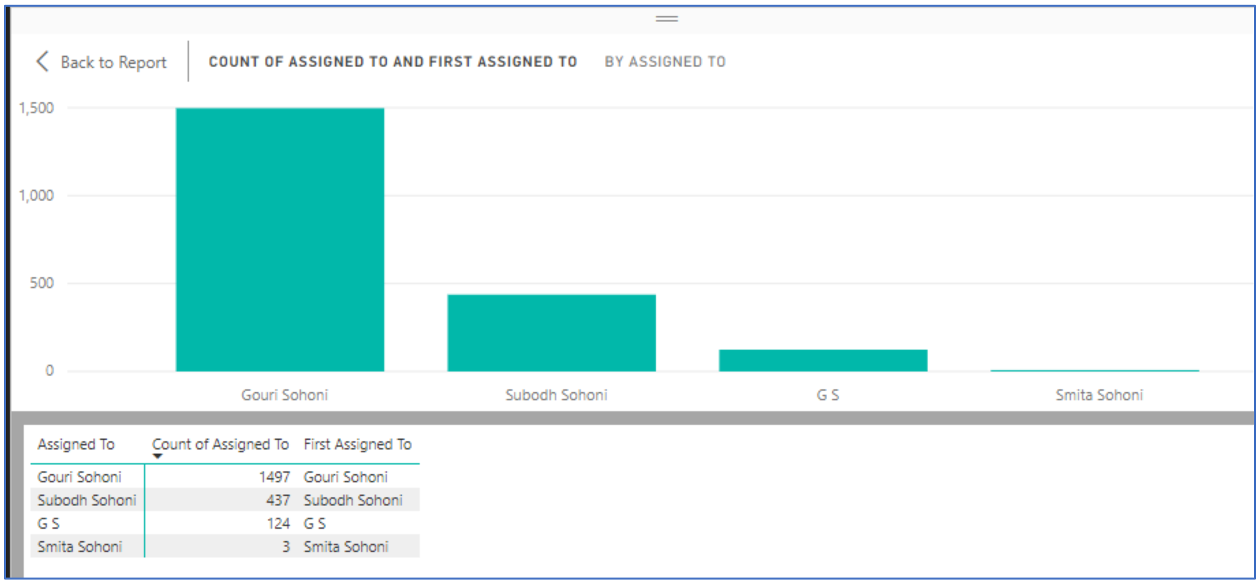
5. Select a required view and click on Load. The data will be pulled from VSTS and displayed



Load

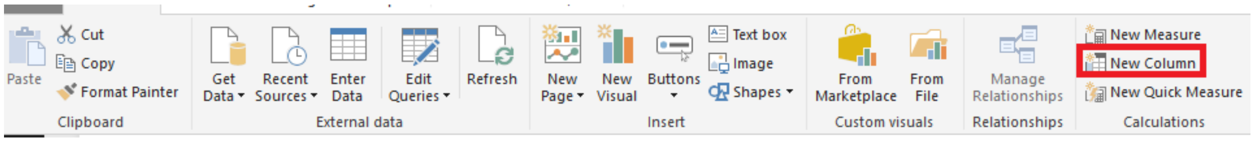
Tasks - Last 30 days
1,184 rows loaded.

6. All fields will be displayed on right hand side, you can select the fields which you want on the report. You can select the chart.

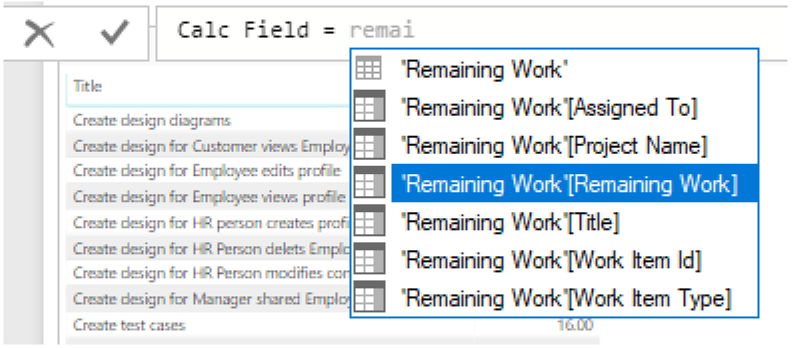


Assigned To	Count of Assigned To	First Assigned To
Gouri Sohoni	1497	Gouri Sohoni
Subodh Sohoni	437	Subodh Sohoni
G S	124	G S
Smita Sohoni	3	Smita Sohoni

7. We can create a custom view and fetch that view in Power BI. We can then create a calculated field in Power BI. I have created remaining work view for a Team, Tasks workitems which have Remaining work fields in positive integer. Once you load that view click on New Column



8. Now you can provide name for the field and any calculation you want to do with respect to the available fields.

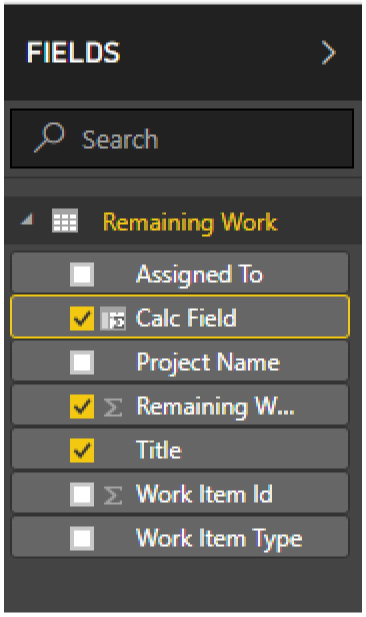


```
Calc Field = remai
```

'Remaining Work'
'Remaining Work'[Assigned To]
'Remaining Work'[Project Name]
'Remaining Work'[Remaining Work]
'Remaining Work'[Title]
'Remaining Work'[Work Item Id]
'Remaining Work'[Work Item Type]

Title
Create design diagrams
Create design for Customer views Employ
Create design for Employee edits profile
Create design for Employee views profile
Create design for HR person creates profi
Create design for HR Person delets Emplo
Create design for HR Person modifies cor
Create design for Manager shared Emplo
Create test cases
16.00

9. A new column with Calc Field name will get added with formula you have specified is added to fields. Select it to add to the report

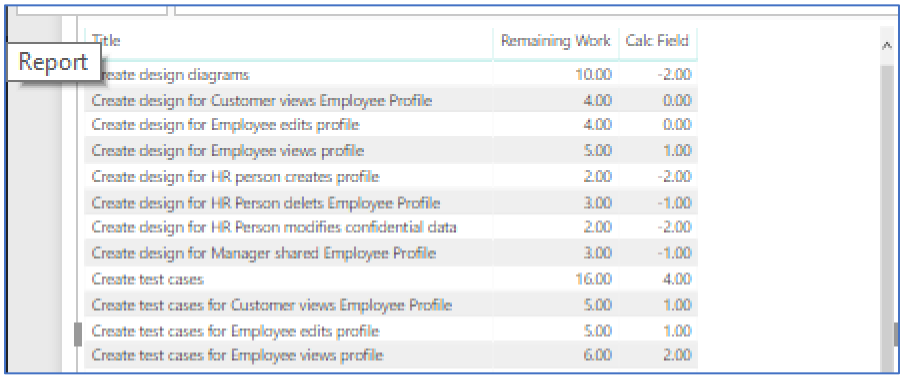


FIELDS

Search

Remaining Work

- [ ] Assigned To
- [✓] Calc Field
- [ ] Project Name
- [✓] Σ Remaining W...
- [✓] Title
- [ ] Σ Work Item Id
- [ ] Work Item Type



Report

Title	Remaining Work	Calc Field
eate design diagrams	10.00	-2.00
Create design for Customer views Employee Profile	4.00	0.00
Create design for Employee edits profile	4.00	0.00
Create design for Employee views profile	5.00	1.00
Create design for HR person creates profile	2.00	-2.00
Create design for HR Person delets Employee Profile	3.00	-1.00
Create design for HR Person modifies confidential data	2.00	-2.00
Create design for Manager shared Employee Profile	3.00	-1.00
Create test cases	16.00	4.00
Create test cases for Customer views Employee Profile	5.00	1.00
Create test cases for Employee edits profile	5.00	1.00
Create test cases for Employee views profile	6.00	2.00

10. This report can be exported to excel in csv format.
11. Excel can be used to further do any calculations of the data if required. Once we have the data in excel we can write any formula