Developing with the Power BI Service API

Setup Time: 40 minutes

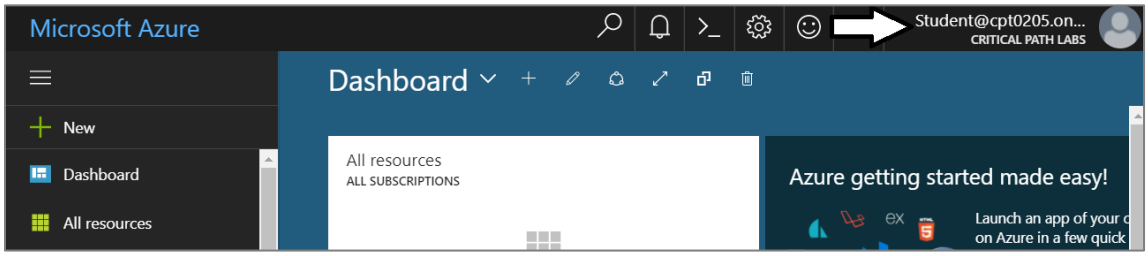
Lab Folder: C:\Student\Modules\03_PowerBiServiceAPI\Lab

Overview: In this lab, you will begin by creating a new Azure AD application that allows you to call the Power BI Service API. After that, you will use Visual Studio to create a new C# console application that programs using the Power BI SDK.

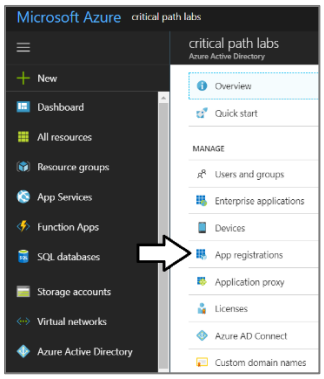
Exercise 1: Register a New Application with Azure Active Directory

In this exercise, you will register a new application with Azure AD and you will configure the application's required permissions to access the Power BI Service API.

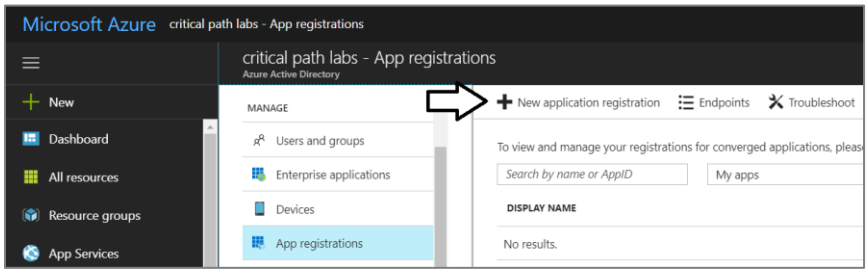
1. Log into the Azure Portal
   a) In the browser, navigate to the Azure portal at https://portal.azure.com.
   b) When you are prompted to log in, provide the credentials to log in with your Office 365 user account name.
   c) Once you are log into the Azure portal, check the email address in the login menu in the upper right to make sure you are logged in the Azure portal with the correct identity.



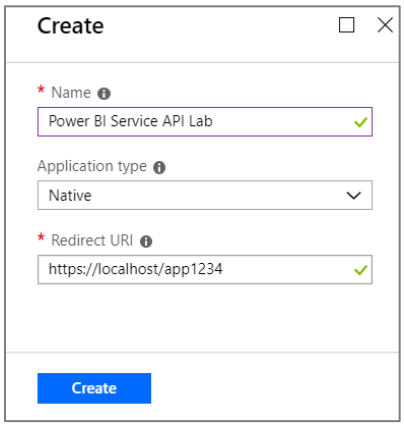
2. Register a new Azure AD application.
   a) In the left navigation, scroll down and click on the link for **Azure Active Directory**.
   b) Click the link for **App registration**.



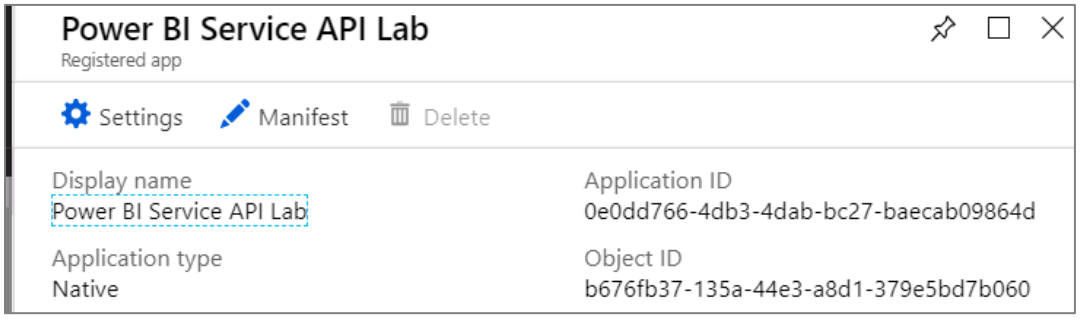
   c) Click **New application registration**.



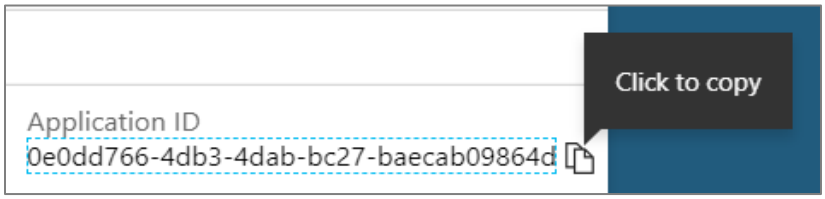
d) In the Create blade, enter the following information.
   i) Add a **Name** of **Power BI Service API Lab**.
   ii) Set the **Application type** to **Native**.
   iii) Set the **Redirect URI** to https://localhost/app1234.
   iv) Click the **Create** button to create the new application.



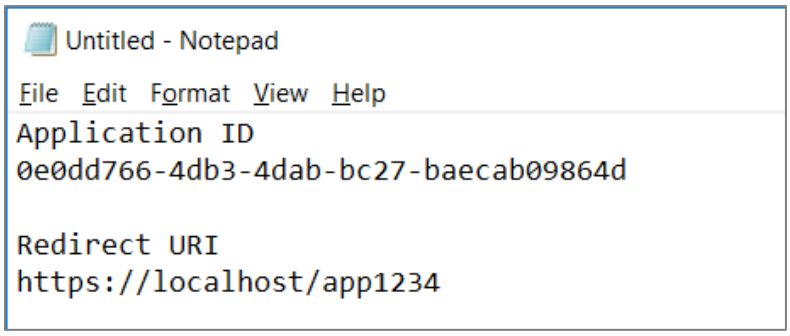
e) Once you have created the new application you should see the new application summary view.



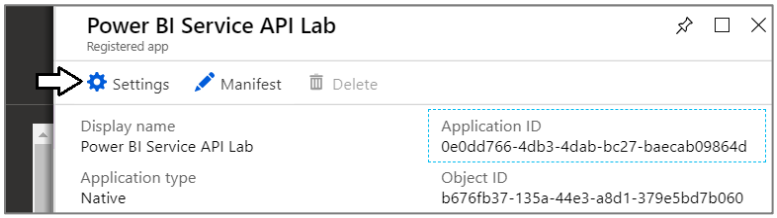
f) Copy the Application ID to the Windows clipboard.



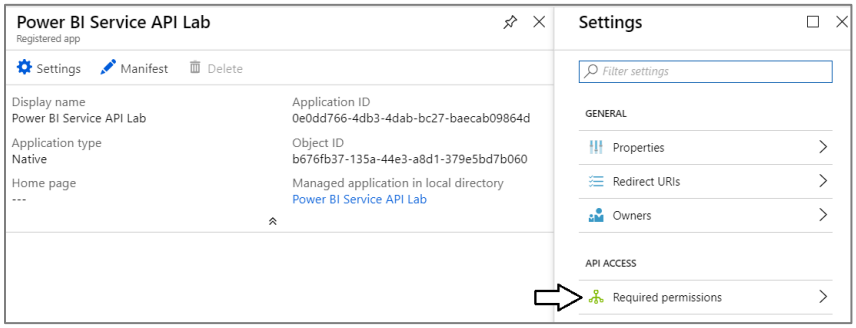
g) Launch Notepad and paste the Application ID into a new text file. Also add the value of the Redirect URI.



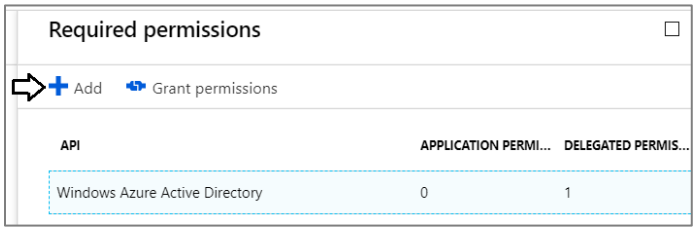
h) Click on the **Settings** link to display the **Settings** blade.



i) In the **Settings** blade, click **Required permissions** link.

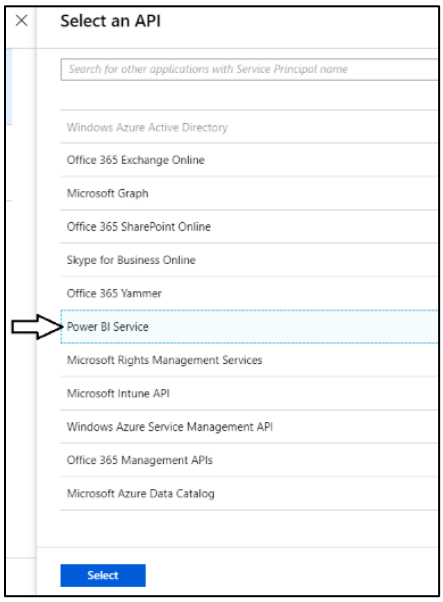


j) In the **Required permissions** blade, click the **Add** button on.



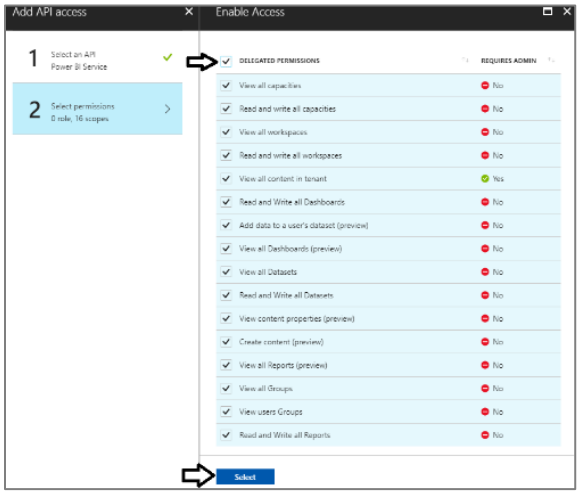
k) Click the **Select an API** option in the **Add API** access blade.

l) In the **Select an API** blade, click **Power BI Service**.



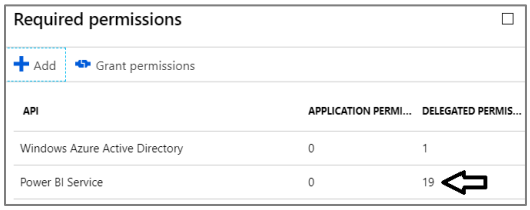
m) In the **Enable Access** blade, click the top checkbox for **DELEGATED PERMISSIONS** to select all the permissions.

n) Once you have selected all the permissions, click the **Select** button at the bottom of the blade.



o) Click the **Done** button at the bottom of the **Add API Access** blade.

p) At this point, you should be able to verify that the Power BI Service has been added to the **Required permissions** list.

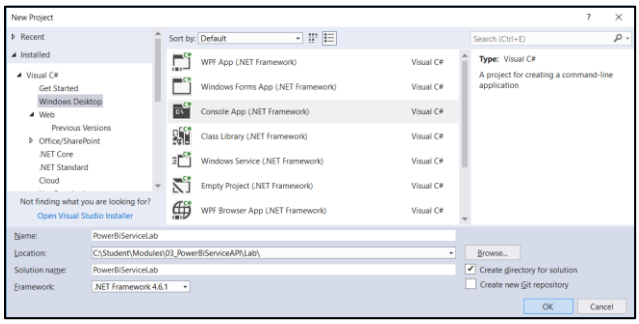


You are now done registering your application with Azure AD.

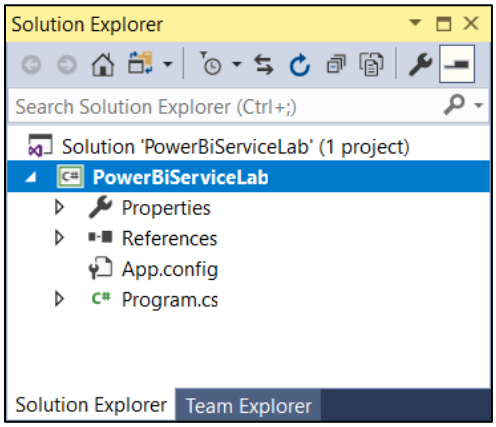
Exercise 2: Call the Power BI Service API using the Power BI SDK

In this exercise, you will create a simple C# Console application to call into the Power BI Service API.

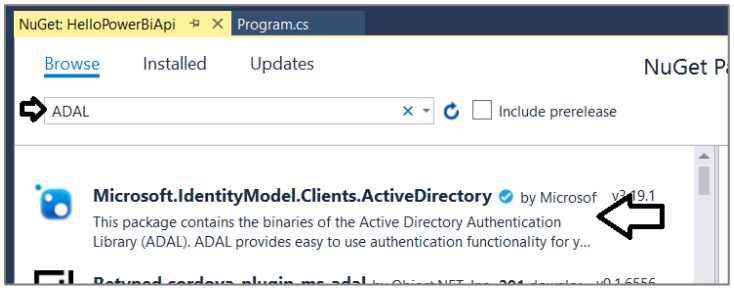
1. Create a new C# Console application in Visual Studio.
   a) Launch Visual Studio.
   b) Create a new project by running the **File > New Project** command.
   c) Select a project type of Console App from the Visual C# project templates.
   d) Give the project a Name of **PowerBiServiceLab** and
   e) Give the project a Location of **C:\Student\Modules\03_PowerBiServiceAPI\Lab**.
   f) Click OK to create the new project.



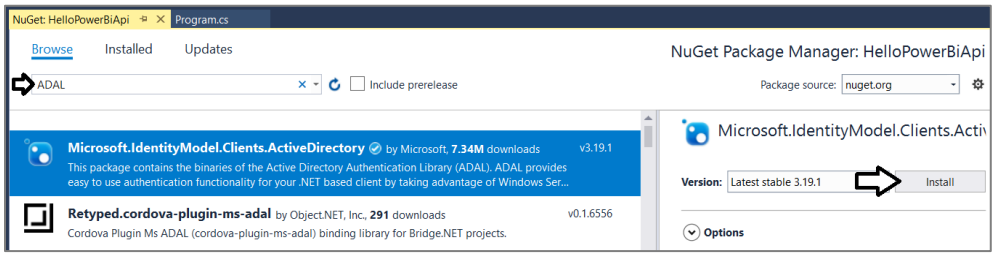
g) You should now have a new project named **PowerBiServiceLab**.



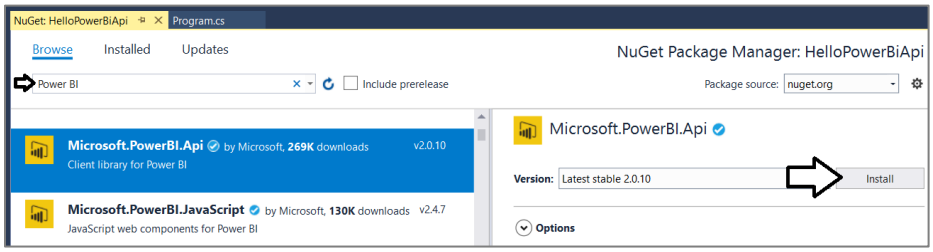
2. Add the NuGet packages to the project required to program the Power BI Service API using the Power BI SDK.
   a) Right-click the top-level node for the **PowerBiServiceLab** project and select **Manage NuGet Packages…**.
   b) Click the Browse tab and type **ADAL** into the search box.
   c) Locate the package **Microsoft.IdentityModel.Clients.ActiveDirectory**. This is the Active Directly Authentication library.



   d) Select and install **Microsoft.IdentityModel.Clients.ActiveDirectory**.

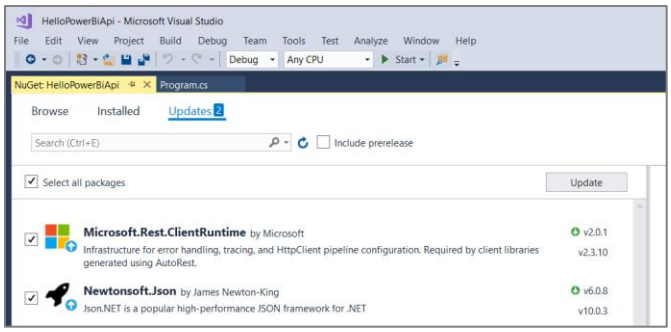


   e) When prompted about the licensing agreement, click **I Agree**.
   f) Search for Power BI and then find and install the **Microsoft.PowerBI.Api**.



   g) When prompted about the licensing agreement, click **I Agree**.

3. Update all NuGet packages.
   a) Navigate to the **Update** tab and update any packages that have updates available.



   b) Close the window for the Nuget Package Manager.
4. Add the starter C# code to **program.cs**.
   a) Using Windows Explorer, locate the file named **ProgramStarter.cs.txt** in the **Student** folder at the following path.

```
C:\Student\Modules\03_PowerBiServiceAPI\StarterFiles\ProgramStarter.cs.txt
```

   b) Open the file named **ProgramStarter.cs.txt** in Notepad and copy its contents into the Window clipboard.
   c) Return to the **PowerBiServiceLab** project in Visual Studio.
   d) Open the source file named **program.cs**.
   e) Delete all the code inside **program.cs** and replace it with the content you copied into the Windows clipboard.
   f) You should now have the basic code for a simple C# console application which access the Power BI Service API.

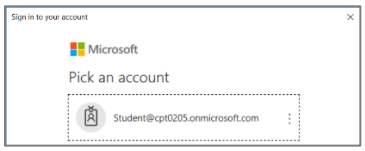
```
using System;
using Microsoft.IdentityModel.Clients.ActiveDirectory;
using Microsoft.PowerBI.Api.V2;
using Microsoft.Rest;

class Program {

  static string aadAuthorizationEndpoint = "https://login.windows.net/common";
  static string resourceUriPowerBi = "https://analysis.windows.net/powerbi/api";
  static string urlPowerBiRestApiRoot = "https://api.powerbi.com/";

  // enter the correct configuration values for your environment
  static string appWorkspaceId = "";
  static string clientId = "";
  static string redirectUrl = "https://localhost/app1234";

  static string GetAccessToken() ...

  static PowerBIClient GetPowerBiClient() ...

  static void Main() ...

  static void DisplayPersonalWorkspaceAssets() ...

}
```

5. Update the code with your app workspace ID, the Azure AD application ID and Redirect URI.
   a) Locate the section of the code with the static properties named **appWorkspaceId**, **clientId** and **redirectUrl**.
   b) Replace these values with the values you copied into Notepad earlier.

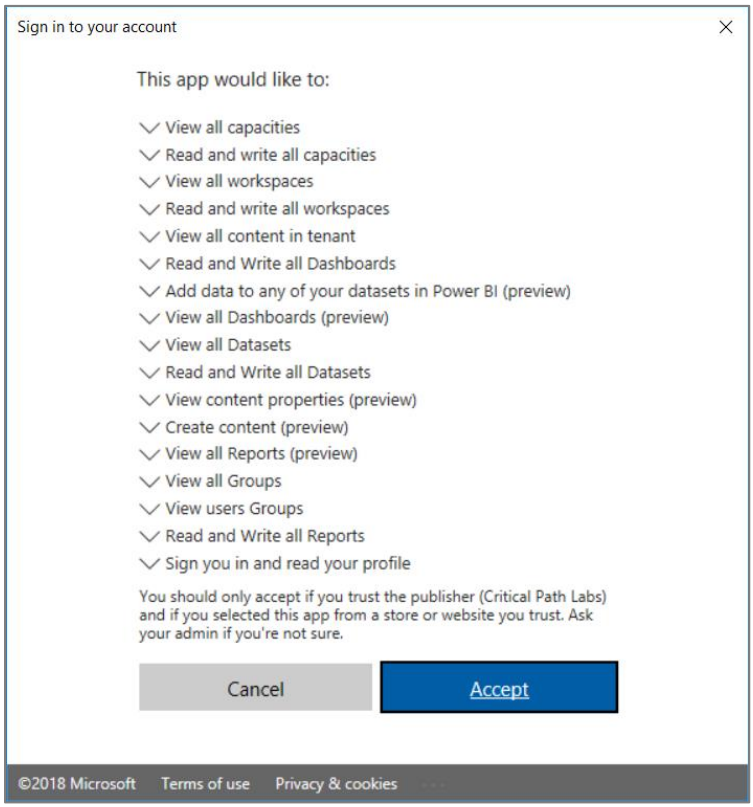
```
// enter the correct configuration values for your environment
static string appWorkspaceId = "dfe5e680-a85a-4731-8c89-963fa5c6c86e";
static string clientId = "0e0dd766-4db3-4dab-bc27-baecab09864d";
static string redirectUrl = "https://localhost/app1234";
```

   c) Save your changes to **program.cs**.

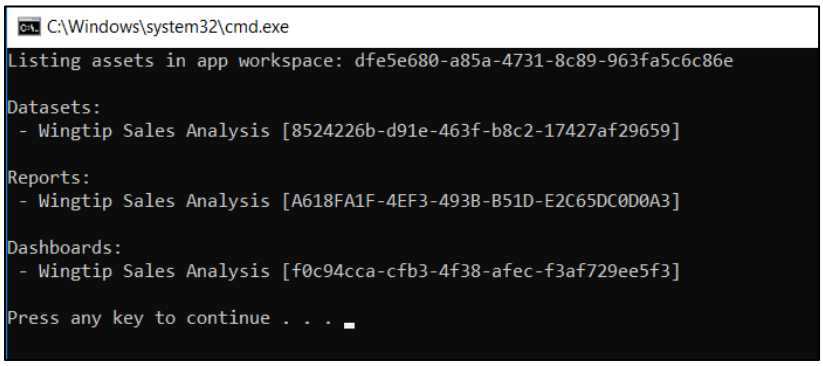
6. Run the application to call to the Power BI Service API.
   a) Press the **{F5}** key to begin a debugging session.
   b) When promoted to sign in, log in using your Office 365 user account credentials.



   c) When prompted with the **Required permissions** dialog, click **Accept**.



   d) The application should run and call into the Power BI Service API to retrieve data about the contents of the app workspace.



Since you will be running this program quite a few times as you write more code, it will make development less tedious if you modify the **GetAccessToken** method so it can run in an unattended fashion without requiring you to sign in interactively.

7. Modify the **GetAccessToken** method to acquire access tokens using the User Password Credential flow.
   a) The following code listing shows the current implementation of the **GetAccessToken** method.

```
static string GetAccessToken() {

  // create new authentication context
  var authenticationContext = new AuthenticationContext(aadAuthorizationEndpoint);

  // use authentication context to trigger user sign-in and return access token
  var promptBehavior = new PlatformParameters(PromptBehavior.SelectAccount);
  var userAuthnResult = authenticationContext.AcquireTokenAsync(resourceUriPowerBi,
                                                                clientId,
                                                                new Uri(redirectUrl),
                                                                promptBehavior).Result;

  // return access token to caller
  return userAuthnResult.AccessToken;
}
```

   b) Replace the code in **GetAccessToken** with the following code which implements the User Password Credentials flow.

```
static string GetAccessToken() {

  // create new authentication context
  var authenticationContext = new AuthenticationContext(aadAuthorizationEndpoint);

  // use authentication context to sign-in using User Password Credentials flow
  string masterUserAccount = "ACCOUNT_NAME_OF_MASTER_USER";
  string masterUserPassword = "PASSWORD_OF_MASTER_USER";
  UserPasswordCredential creds = new UserPasswordCredential(masterUserAccount, masterUserPassword);

  var userAuthnResult =
    authenticationContext.AcquireTokenAsync(resourceUriPowerBi, clientId, creds).Result;

  // return access token to caller
  return userAuthnResult.AccessToken;
}
```

   c) Update the variables **masterUserAccount** and **masterUserPassword** with the credentials for your Office 365 account.

```
// use authentication context to sign-in using User Password Credentials flow
string masterUserAccount = "student@portlandembed.onMicrosoft.com";
string masterUserPassword = "pass@word1";
UserPasswordCredential creds = new UserPasswordCredential(masterUserAccount, masterUserPassword);
```

   d) Save your changes to **program.cs**.
8. Run the application to call to the Power BI Service API.
   a) Press the **{F5}** key to begin a debugging session.
   b) The program should run as it did before but it should no longer require you to interactively enter a user name and password.

Exercise 3: Write C# Code to Create an App Workspace and Upload a PBIX Project File

In this exercise, you will update the a C# Console application to create app workspaces and publish PBIX project files.

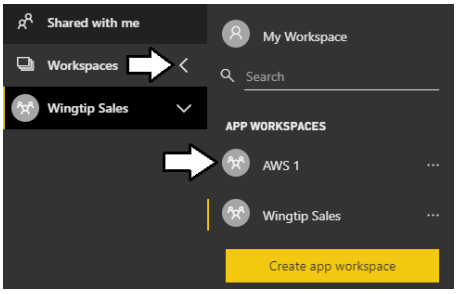
1. Add the code required to create a new app workspace.
   a) Add the static **CreateAppWorkspace** method to the bottom of the **Program** class in **program.cs**.

```
static string CreateAppWorkspace(string Name) {
  PowerBIClient pbiClient = GetPowerBiClient();
  // create new app workspace
  GroupCreationRequest request = new GroupCreationRequest(Name);
  Group aws = pbiClient.Groups.CreateGroup(request);
  // return app workspace ID
  return aws.Id;
}
```

b) Update the **Main** method to match the following code.

```
static void Main() {

  //DisplayPersonalWorkspaceAssets();
  CreateAppWorkspace("AWS 1");

}
```

2. Run the application to call to the Power BI Service API.
   a) Press the **{F5}** key to begin a debugging session.
   b) The program should run without any errors.
   c) After the program runs, you should be able to confirm that it created a new app workspace named **AWS 1**.



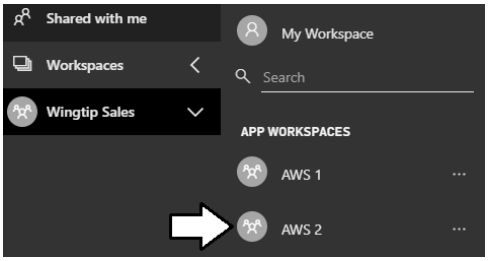
3. Add the code required to publish a PBIX project file to an app workspace.
   a) Add the static **PublishPBIX** method to the bottom of the **Program** class in **program.cs**.

```
static void PublishPBIX(string appWorkspaceId, string PbixFilePath, string ImportName) {
  Console.WriteLine("Publishing " + PbixFilePath);
  PowerBIClient pbiClient = GetPowerBiClient();
  FileStream stream = new FileStream(PbixFilePath, FileMode.Open, FileAccess.Read);
  var import = pbiClient.Imports.PostImportWithFileInGroup(appWorkspaceId, stream, ImportName);
  Console.WriteLine("Publishing process completed");
}
```

   b) Update the **Main** method to match the following code which uploads a PBIX file with an Import name of **Wingtip Sales**.

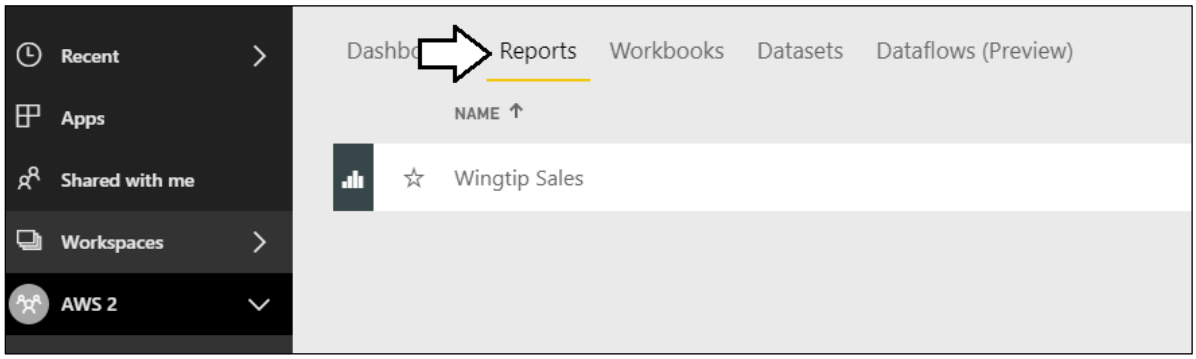
```
static void Main() {
  //DisplayPersonalWorkspaceAssets();
  //CreateAppWorkspace("AWS 1");

  string appWorkspaceId = CreateAppWorkspace("AWS 2");
  string pbixPath = @"C:\Student\PBIX\Wingtip Sales Analysis.pbix";
  string importName = "Wingtip Sales";
  PublishPBIX(appWorkspaceId, pbixPath, importName);
}
```

4. Run the application to call to the Power BI Service API.
   a) Press the **{F5}** key to begin a debugging session.
   b) The program should run without any errors.
   c) After the program runs, you should be able to confirm that it created a new app workspace named **AWS 2**.



d) Navigate the **AWS 2** workspace and click the **Reports** tab.

e) You should be able to verify that a report exists with the same Import name which is **Wingtip Sales**.



Exercise 4: Add C# Code to Clone Power BI Content Across Workspaces

In this exercise, you will copy-and-paste a large piece of code for the **CloneAppWorkspace** method that clones content from a source app workspace to a target app workspace. Then you will test the code to make sure it works in your environment.

1. Copy and paste the code for the **CloneAppWorkspace** method.

   a) Using Windows Explorer, locate the file named **CloneAppWorkspace.cs.txt** in the **Student** folder at the following path.

   ```
   C:\Student\Modules\03_PowerBiServiceAPI\StarterFiles\CloneAppWorkspace.cs.txt
   ```

   b) Open the file named **CloneAppWorkspace.cs.txt** in Notepad and copy its contents into the Window clipboard.

   c) Return to the **PowerBiServiceLab** project in Visual Studio.

   d) Return to the source file named **program.cs**.

   e) Place you cursor at the bottom of the **Program** class and paste in the content you copied into the Windows clipboard.

   f) The **Program** class should now contain a method named **CloneAppWorkspace**.

```
class Program {

  static string aadAuthorizationEndpoint = "https://login.windows.net/common";
  static string resourceUriPowerBi = "https://analysis.windows.net/powerbi/api";
  static string urlPowerBiRestApiRoot = "https://api.powerbi.com/";

  // enter the correct configuration values for your environment
  static string appWorkspaceId = "dfe5e680-a85a-4731-8c89-963fa5c6c86e";
  static string clientId = "0e0dd766-4db3-4dab-bc27-baecab09864d";
  static string redirectUrl = "https://localhost/app1234";

  static string GetAccessToken() ...

  static PowerBIClient GetPowerBiClient() ...

  static void Main() ...

  static void DisplayPersonalWorkspaceAssets() ...

  static string CreateAppWorkspace(string Name) ...

  static void PublishPBIX(string appWorkspaceId, string PbixFilePath, string ImportName) ...

  static void CloneAppWorkspace(string sourceAppWorkspaceName, string targetAppWorkpaceName)

    PowerBIClient pbiClient = GetPowerBiClient();
    string sourceAppWorkspaceId = "";
    string targetAppWorkspaceId = "";

    var workspaces = pbiClient.Groups.GetGroups().Value;
    foreach (var workspace in workspaces) {
      if (workspace.Name.Equals(sourceAppWorkspaceName)) {
        sourceAppWorkspaceId = workspace.Id;
      }
      if (workspace.Name.Equals(targetAppWorkpaceName)) {
```

2. Take a moment to review the code inside **CloneAppWorkspace**.
   a) The code begins by determining whether the source app workspace and target app workspace exist.

```
  static void CloneAppWorkspace(string sourceAppWorkspaceName, string targetAppWorkpaceName) {

  PowerBIClient pbiClient = GetPowerBiClient();
  string sourceAppWorkspaceId = "";
  string targetAppWorkspaceId = "";

  var workspaces = pbiClient.Groups.GetGroups().Value;
  foreach (var workspace in workspaces) {
    if (workspace.Name.Equals(sourceAppWorkspaceName)) {
      sourceAppWorkspaceId = workspace.Id;
    }
    if (workspace.Name.Equals(targetAppWorkpaceName)) {
      targetAppWorkspaceId = workspace.Id;
    }
  }

  if (sourceAppWorkspaceId == "") {
    throw new ApplicationException("Source Workspace does not exist");
  }

  if (targetAppWorkspaceId == "") {
    // create target app workspace if it doesn't exist
    Console.WriteLine("Creating app workspace named " + targetAppWorkpaceName);
    Console.WriteLine();
    GroupCreationRequest request = new GroupCreationRequest(targetAppWorkpaceName);
    Group AppWorkspace = pbiClient.Groups.CreateGroup(request);
    targetAppWorkspaceId = AppWorkspace.Id;
  }
```

   b) Next, the code exports PBIX files to clone the datasets and reports in the target workspace.

```
var reports = pbiClient.Reports.GetReportsInGroup(sourceAppWorkspaceId).Value;

string downloadPath = @"C:\Student\downloads\";

// create download folder if it doesn't exist
if (!Directory.Exists(downloadPath)) {
  Directory.CreateDirectory(downloadPath);
}

foreach (var report in reports) {

  var reportStream = pbiClient.Reports.ExportReportInGroup(sourceAppWorkspaceId, report.Id);
  string filePath = downloadPath + report.Name + ".pbix";
  Console.WriteLine("Downloading PBIX file for " + report.Name + "to " + filePath);
  FileStream stream1 = new FileStream(filePath, FileMode.Create, FileAccess.ReadWrite);
  reportStream.CopyToAsync(stream1).Wait();
  reportStream.Close();
  stream1.Close();
  stream1.Dispose();

  FileStream stream = new FileStream(filePath, FileMode.Open, FileAccess.Read);
  Console.WriteLine("Publishing " + filePath + " to " + targetAppWorkpaceName);
  var import = pbiClient.Imports.PostImportWithFileInGroup(targetAppWorkspaceId, stream, report.Name);

  Console.WriteLine("Deleing file " + filePath);
  stream.Close();
  stream.Dispose();
  File.Delete(filePath);

  Console.WriteLine();
}

Console.WriteLine("Export/Import process completed");
```

You will be able to see the PBIX file created in **C:\Student\downloads** folder when the program runs.

c) At the end of **CloneAppWorkspace**, there is code to clone dashboard tiles from one app workspace to another.

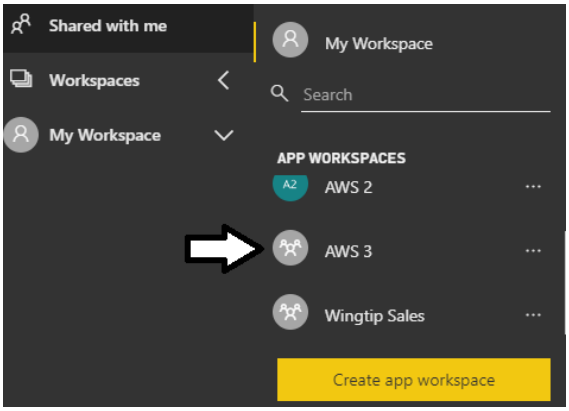
```
var dashboards = pbiClient.Dashboards.GetDashboardsInGroup(sourceAppWorkspaceId).Value;

foreach (var sourceDashboard in dashboards) {

  // create the target dashboard
  Console.WriteLine();
  Console.WriteLine("Creating Dashboard named " + sourceDashboard.DisplayName);
  AddDashboardRequest addReq = new AddDashboardRequest(sourceDashboard.DisplayName);
  Dashboard targetDashboard = pbiClient.Dashboards.AddDashboardInGroup(targetAppWorkspaceId, addReq);

  // clone tiles
  IList<Tile> sourceTiles =
              pbiClient.Dashboards.GetTilesInGroup(sourceAppWorkspaceId, sourceDashboard.Id).Value;

  foreach (Tile sourceTile in sourceTiles) {
    Console.WriteLine("Adding dashboard tile with title of " + sourceTile.Title);
    var sourceDatasetID = sourceTile.DatasetId;
    var sourceDatasetName =
        pbiClient.Datasets.GetDatasetByIdInGroup(sourceAppWorkspaceId, sourceDatasetID).Name;
    var targetWorkspaceDatasets = pbiClient.Datasets.GetDatasetsInGroup(targetAppWorkspaceId).Value;
    string targetDatasetId = "";
    foreach (var ds in targetWorkspaceDatasets) {
      if (ds.Name.Equals(sourceDatasetName)) {
        targetDatasetId = ds.Id;
      }
    }
    if (targetDatasetId.Equals("")) throw new ApplicationException("An error occured!");

    var sourceReportId = sourceTile.ReportId;
    var sourceReportName =
        pbiClient.Reports.GetReportInGroup(sourceAppWorkspaceId, sourceReportId).Name;

    var targetWorkspaceReports = pbiClient.Reports.GetReportsInGroup(targetAppWorkspaceId).Value;
    string targetReportId = "";

    foreach (var r in targetWorkspaceReports) {
      if (r.Name.Equals(sourceReportName)) {
        targetReportId = r.Id;
      }
    }

    CloneTileRequest addReqTile =
      new CloneTileRequest(targetDashboard.Id, targetAppWorkspaceId, targetReportId, targetDatasetId);

    pbiClient.Dashboards.CloneTileInGroup(sourceAppWorkspaceId,
                                          sourceDashboard.Id,
                                          sourceTile.Id,
                                          addReqTile);
  }
}
```

d) Update the **Main** method to match the following code which uploads a PBIX file with an Import name of **Wingtip Sales**.

```
static void Main() {
  //DisplayPersonalWorkspaceAssets();
  //CreateAppWorkspace("AWS 1");
  //string appWorkspaceId = CreateAppWorkspace("AWS 2");
  //string pbixPath = @"C:\Student\PBIX\Wingtip Sales Analysis.pbix";
  //string importName = "Wingtip Sales";
  //PublishPBIX(appWorkspaceId, pbixPath, importName);

  CloneAppWorkspace("Wingtip Sales", "AWS 3");

}
```

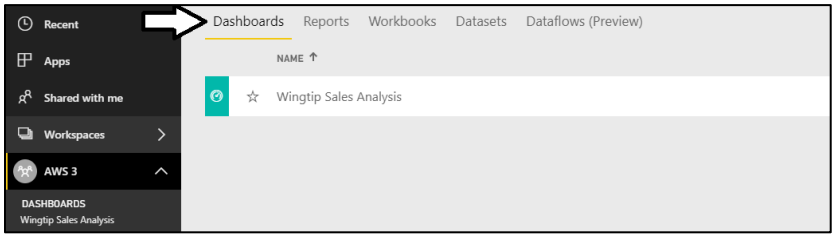
3. Run the application to call to the Power BI Service API.
   a) Press the **{F5}** key to begin a debugging session.
   b) The program should run without any errors.

c) After the program runs, you should be able to confirm that it created a new app workspace named **AWS 3**.

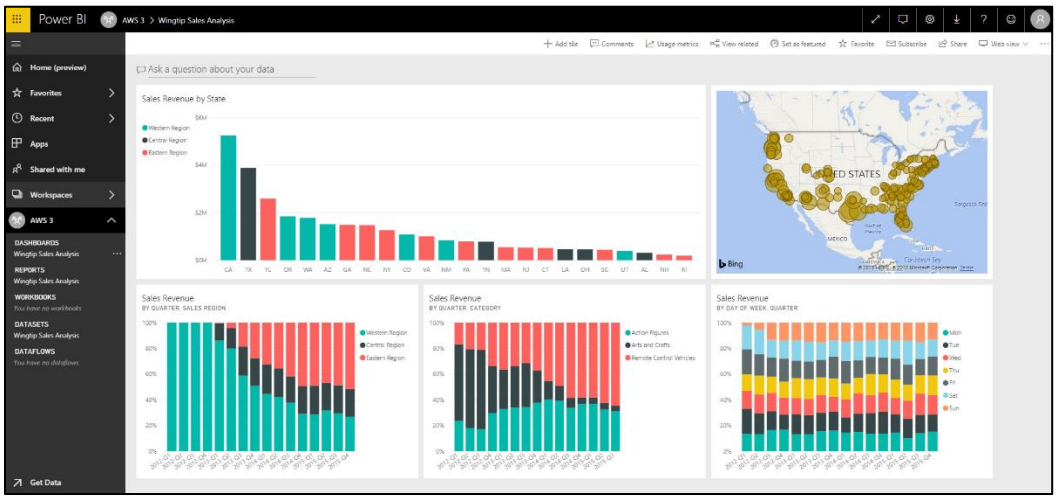


d) Navigate the **AWS 3** workspace and click the **Dashboards** tab.

e) You should be able to verify that the dashboards from the **Wingtip Sales** workspace have been clones in **AWS 3**.



f) Open the Wingtip Sales Analysis dashboard to verify the tiles have all been cloned correctly.



Congratulations. You have now successfully called into the Power BI Service API.