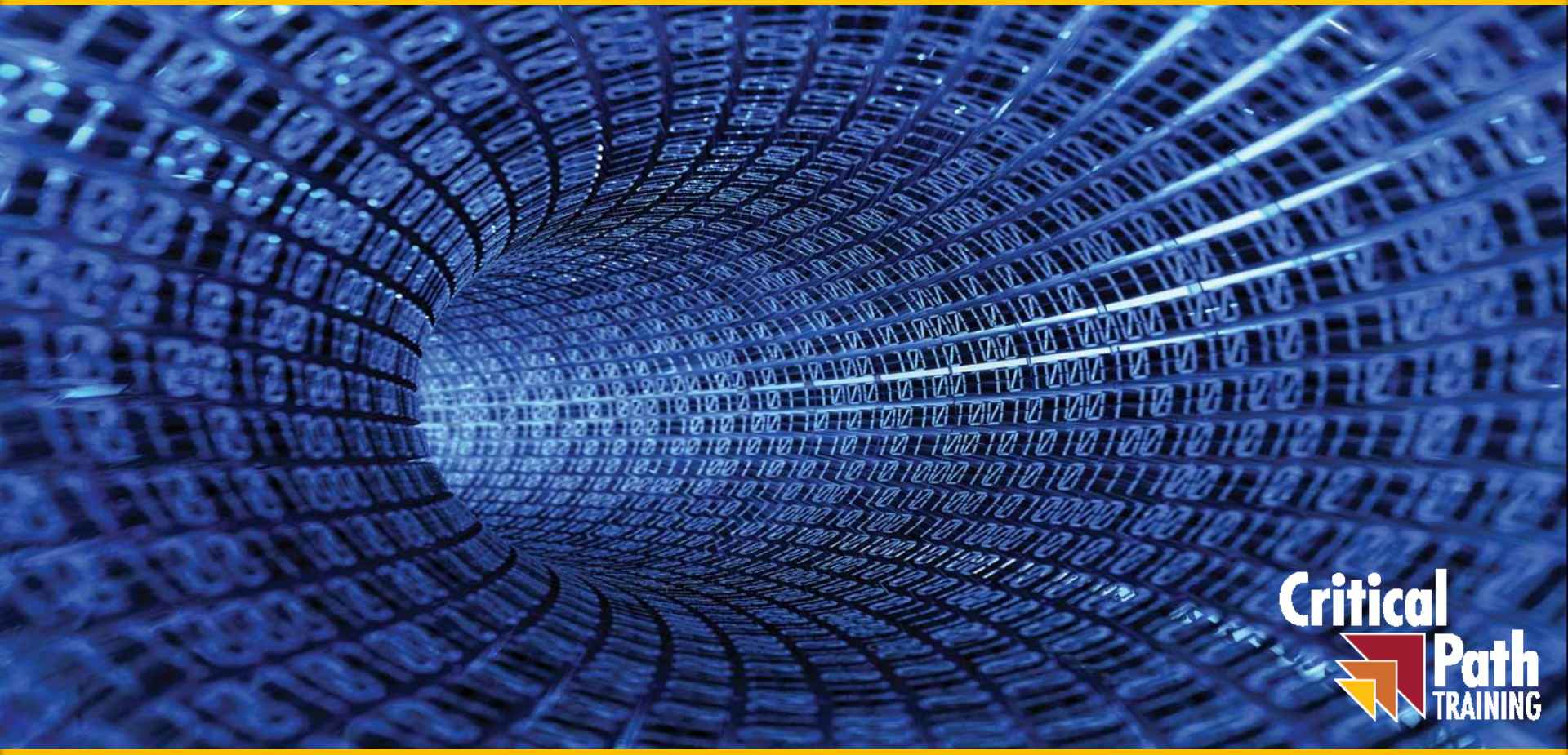
# Programming the Power BI Service API



# Agenda

- Power BI Service API Overview
- Understanding Authentication with Azure AD
- Programming with the Power BI .NET SDK
- Acquiring Access Tokens using MSAL
- Calling to Power BI using App-only Tokens

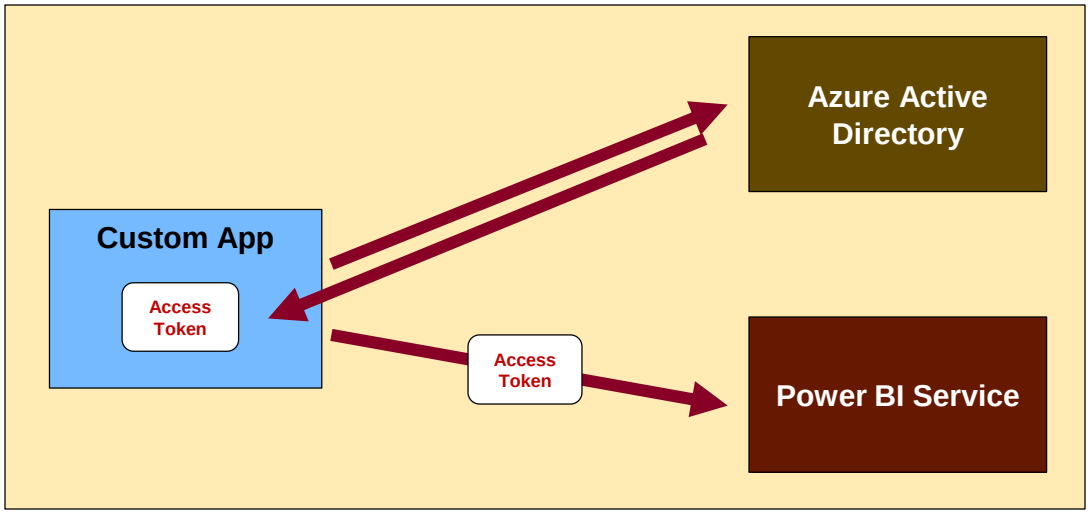
# What Is the Power BI Service API?

- What is the Power BI Service API?
  - API built on OAuth2, OpenID Connect, REST and ODATA
  - API secured by Azure Active Directory (AAD)
  - API to program with workspaces, datasets, reports & dashboards
  - API also often called “Power BI REST API”

- What can you do with the Power BI Service API?
  - Publish PBIX project files
  - Update connection details and datasource credentials
  - Create workspaces and clone content across workspaces
  - Embed Power BI reports and dashboards tiles in web pages
  - Create streaming datasets in order to build real-time dashboards

# Authenticating with Azure AD

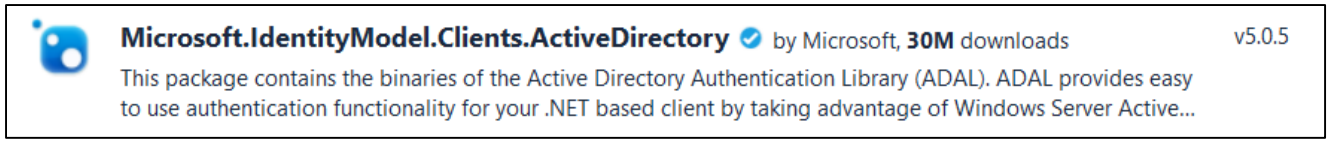
- Custom applications must authenticate with Azure AD
  - Your code implements and authentication flow to obtain access token
  - Access token must be passed when calling Power BI Service API




- Microsoft supports two endpoints for programming authentication
  - Azure AD V1 endpoint (released to GA over 8 years ago)
  - Azure AD V2 endpoint (released to GA in May 2019)

# Azure AD Endpoints and Libraries

- Authenticating with the Azure AD V1 Endpoint
  - Heavily used over the last 5-6 years
  - Accessed through **Azure AD Authentication Library (ADAL)**

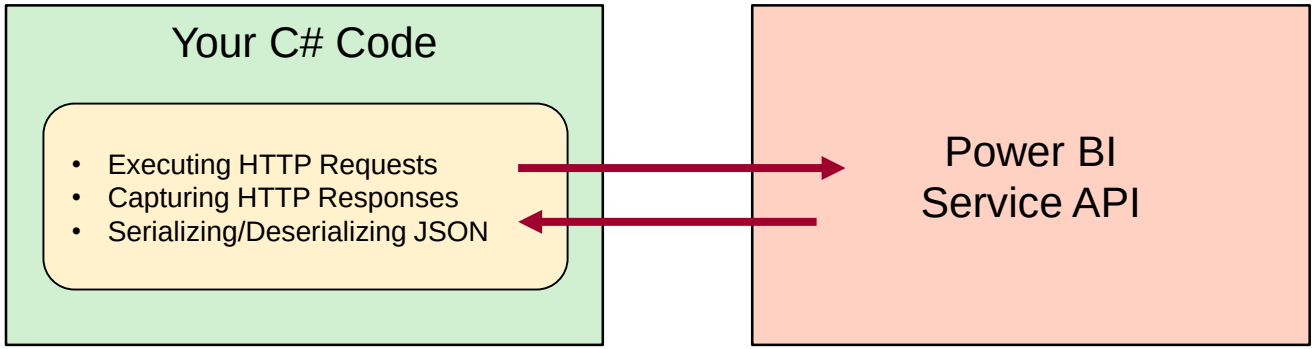



- Authenticating with the Azure AD V2 Endpoint
  - Moved from preview to GA in May 2019
  - Accessed through **Microsoft Authentication Library (MSAL)**

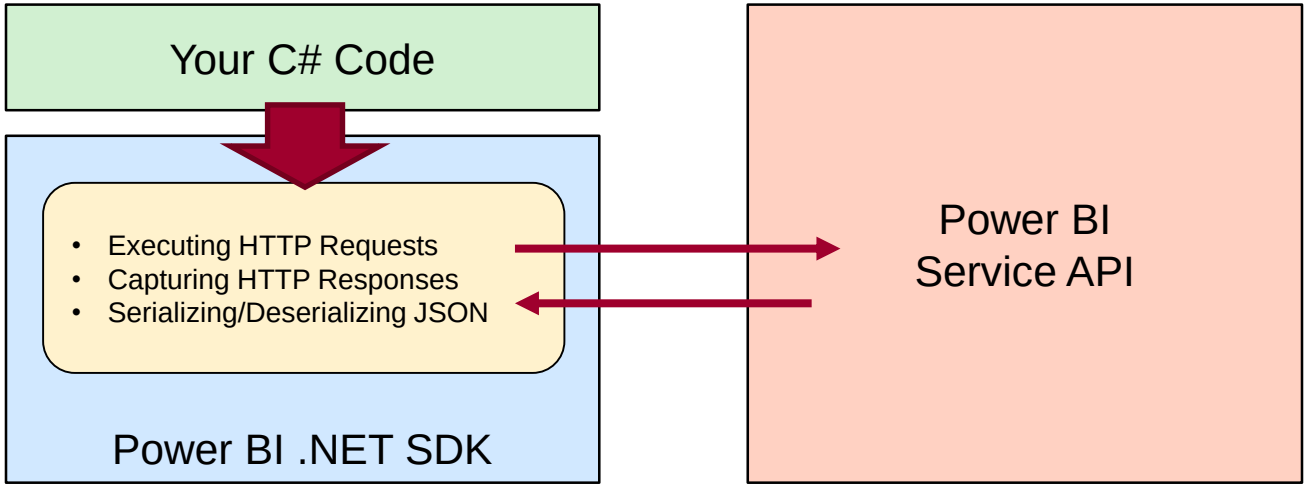

- Why move to the Azure AD V2 Endpoint?
  - Dynamic Incremental consent
  - New authentication flows (e.g. device code flow)

# Power BI .NET SDK

- Developing without the Power BI .NET SDK




- Developing with the Power BI .NET SDK



# User APIs versus Admin APIs

- Power BI User APIs (e.g. `GetGroupsAsync`)
  - provides users with access to personal workspace
  - provides users with access to app workspaces
  - provides service principal (SP) with access to app workspaces

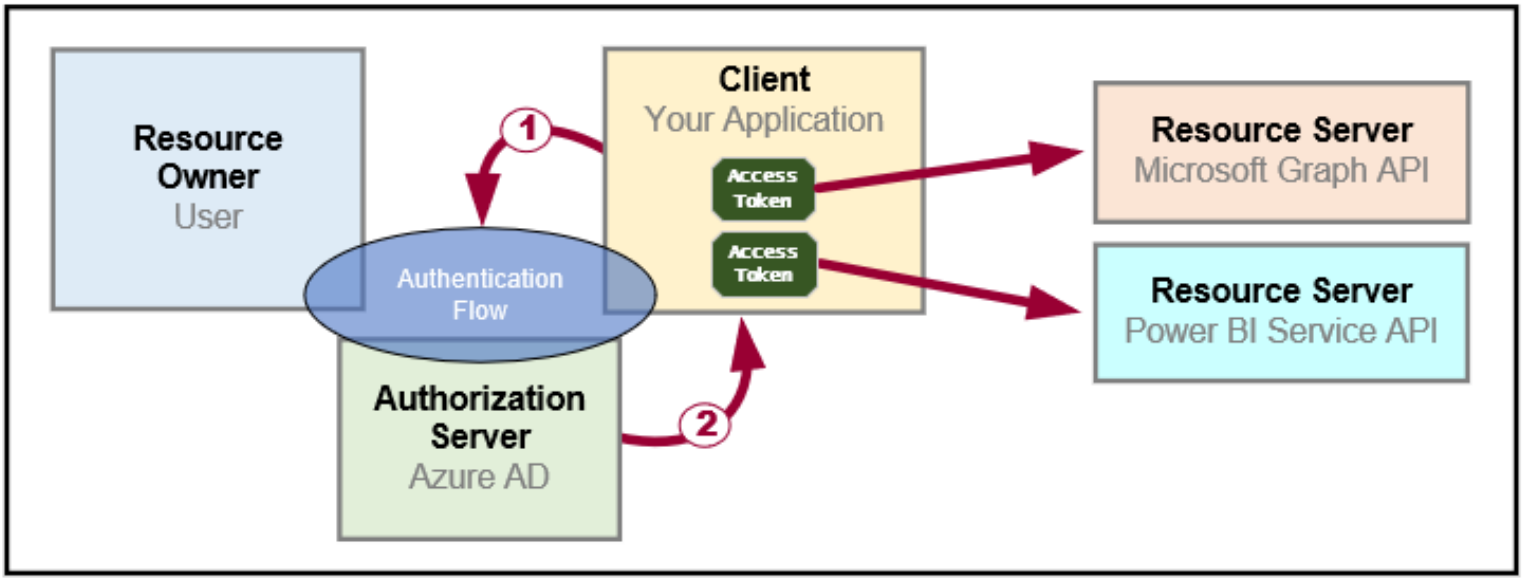
- Power BI Admin APIs (e.g. `GetGroupsAsAdminAsync`)
  - provides users with tenant-level access to all workspaces
  - does not currently support app-only authentication

# Agenda

- ✓ Power BI Service API Overview
- ➢ Understanding Authentication with Azure AD
- Programming with the Power BI .NET SDK
- Acquiring Access Tokens using MSAL
- Calling to Power BI using App-only Tokens

# OAuth 2.0 Fundamentals

- Client application calls to resource server on behalf of a user
  - Client implements authentication flow to acquire access token
  - Access token contains permission grants for client to call resource server
  - Client passes access token when calling to resource server
  - Resource server inspects access token to ensure client has permissions



# Access Token is a Bearer Token

- It can be used by any who bears (e.g. steals) it
  - Always encrypt with HTTPS when transmitting access tokens


```
{
    "iss": "https://sts.windows.net/f995267b-5b7d-4e65-b929-d3d3e11784f9/",
    "amr": [ "pwd" ],

    "iat": 1542829619, "nbf": 1542829619, "exp": 1542833519,

    "tid": "f995267b-5b7d-4e65-b929-d3d3e11784f9",

    "appid": "b52f8e53-d0bf-45c2-9c39-d9c1e96e572c",

    "aud": "https://analysis.windows.net/powerbi/api",

    "scp": "Dashboard.Read.All Dataset.Read.All Group.Read.All Report.ReadWrite.All",

    "oid": "32573058-0ac0-4935-a39d-cd57d5a5a894",
    "unique_name": "maxwells@sharepointconfessions.onmicrosoft.com",
    "upn": "maxwells@sharepointconfessions.onmicrosoft.com",
    "name": "Maxwell Smart",
    "family_name": "Maxwell",
    "given_name": "Smart",

    "ipaddr": "47.200.98.132",

    "ver": "1.0"
}
```

# OAuth 2.0 Client Registration

- Client must be registered with authorization server
  - Authorization server tracks each client with unique Client ID
  - Client should be registered with one or more Reply URLs
  - Reply URL should be fixed endpoint on Internet
  - Reply URL used to transmit security tokens to clients
  - Client registration tracks permissions and other attributes

**Authorization Server**

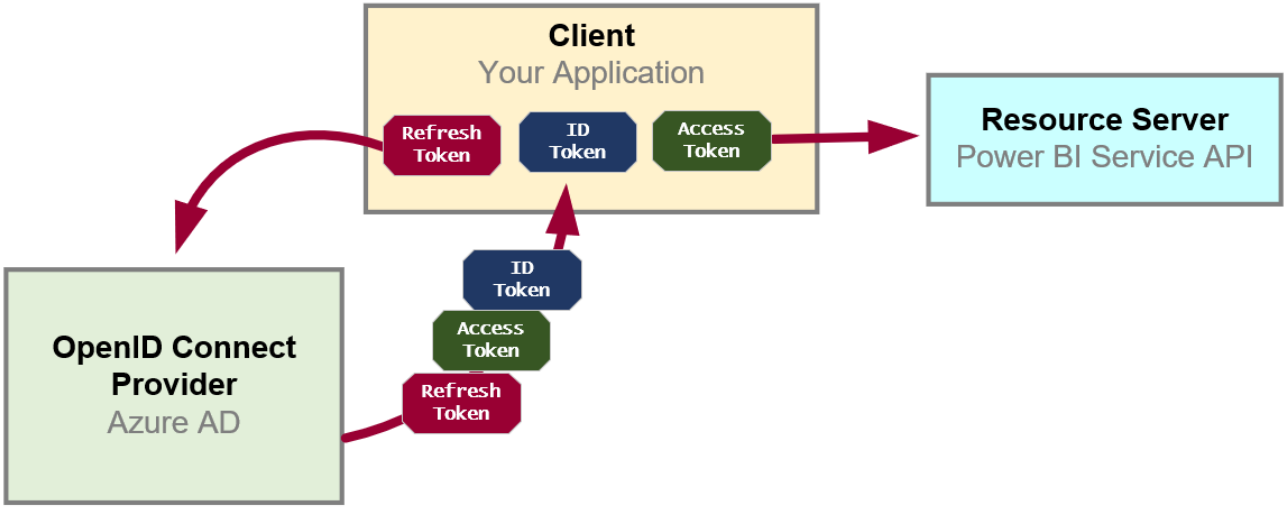
Azure AD

**Registered Applications**

| Name | App ID | Permissions | Reply URL | Credentials |
|---|---|---|---|---|
| App1 | guid1 | … | none | none |
| App2 | guid2 | … | … | secret key |
| App3 | guid3 | … | … | X.509 Certificate |

# OpenID Connect Extends OAuth 2.0

- OAuth 2.0 has shortcomings with authentication & identity
  - It does not provide client with means to validate access tokens
  - Lack of validation makes client vulnerable to token forgery attacks
- Open ID Connect is standard which extends OAuth 2.0
  - OpenID Connect provider passes ID token in addition to OAuth 2.0 tokens
  - OpenID Connect provider provides client with keys for token validation

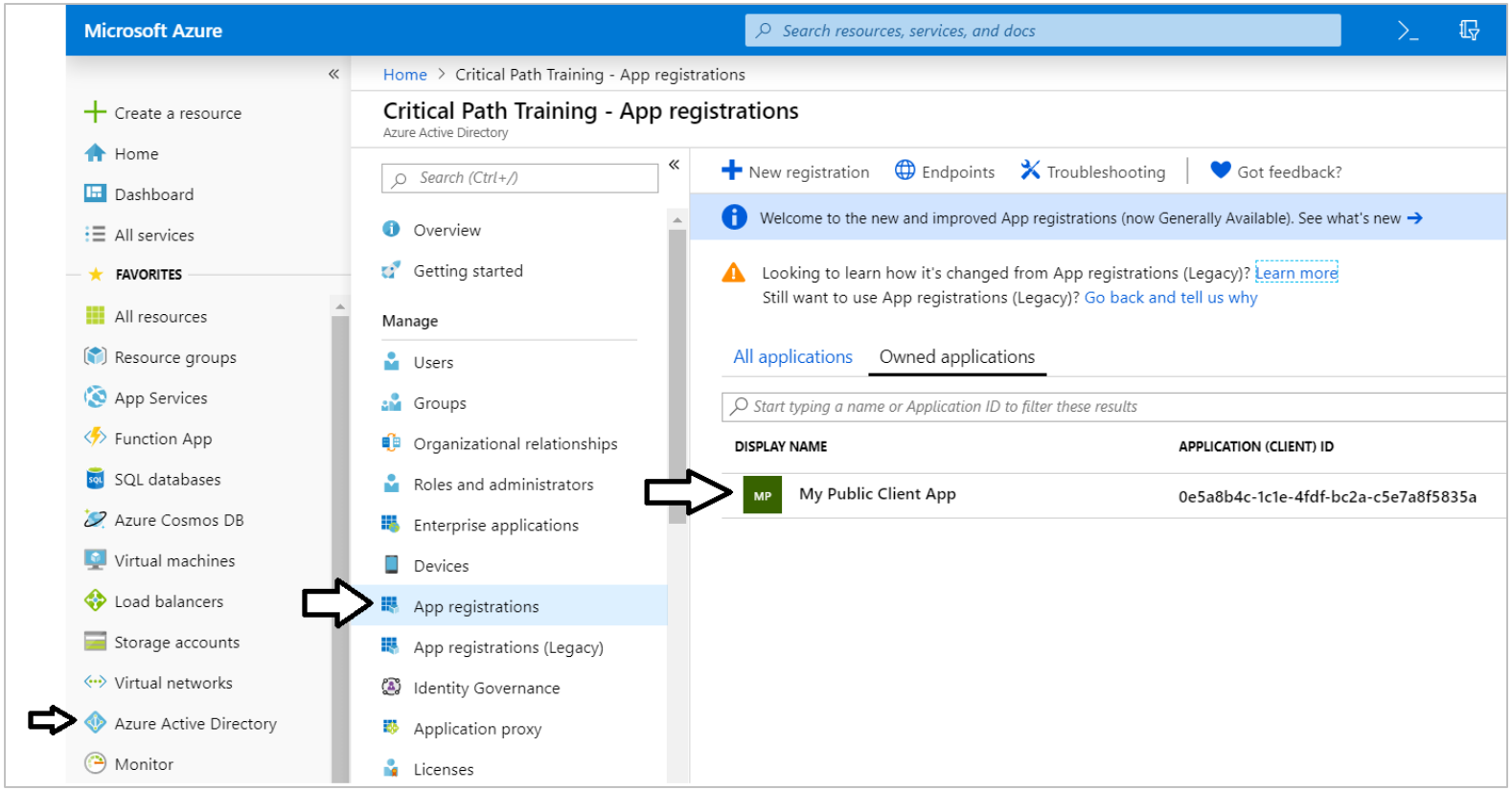


# Authentication Flows

- **User Password Credential Flow** *(public client)*
  - Used in Native clients to obtain access code
  - Requires passing user name and password across network
- **Device Code Flow** *(public client)*
  - New style of authentication introduced with Azure AD v2 Endpoint
- **Client Credentials Flow** *(confidential client)*
  - Authentication based on password or certificate held by application
  - Used to obtain app-only access tokens
- **Authorization Code Flow** *(confidential client)*
  - Client first obtains authorization code sent back to browser
  - Client then obtains access token in server-to-server call
- **Implicit Flow** *(public client)*
  - Used in SPAs built with JavaScript and AngularJS
  - Application obtains access token w/o acquiring authorization code

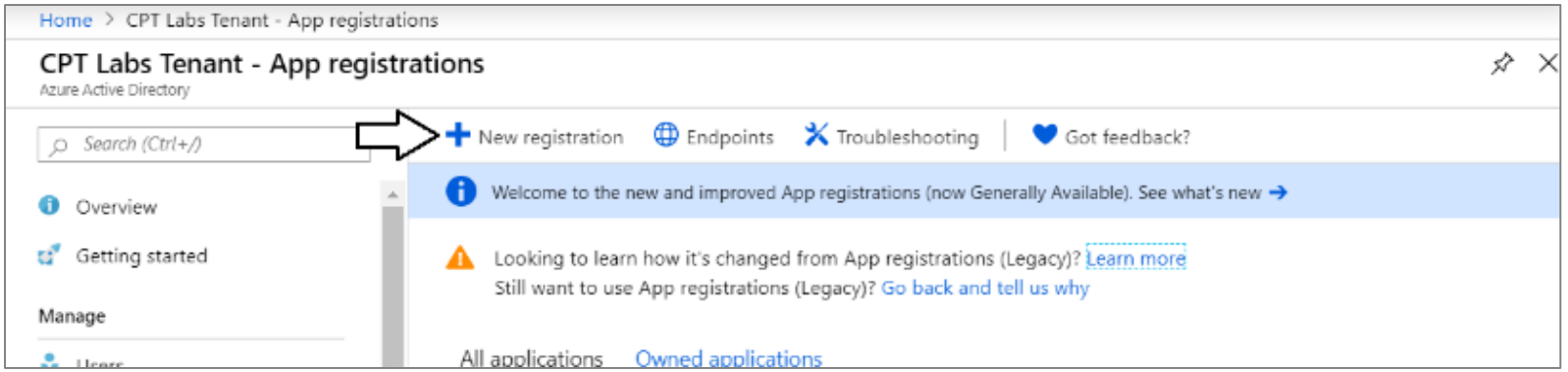
# The Azure Portal

- Azure portal allows you to register Azure AD applications
  - Azure Portal accessible at https://portal.azure.com
  - No Azure subscription required to register applications



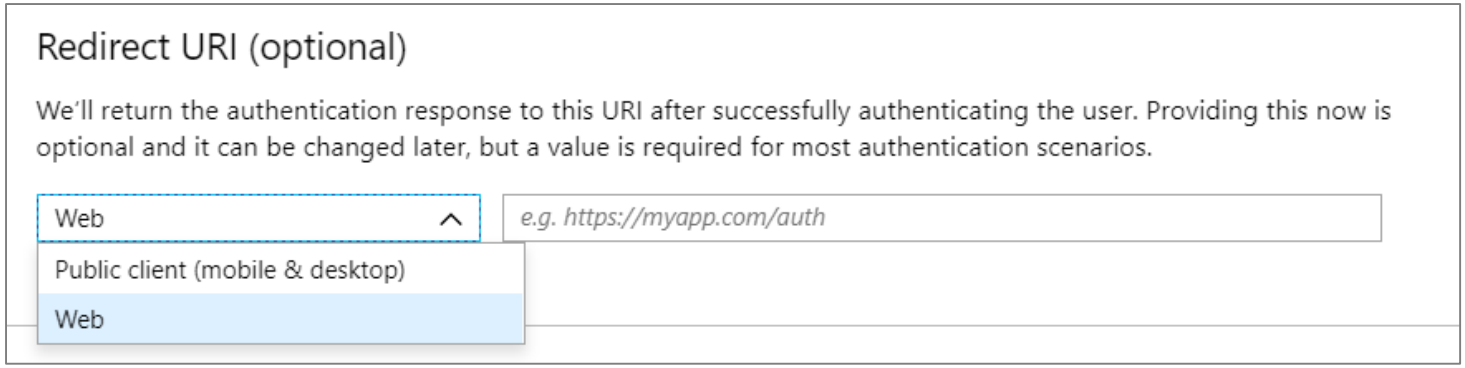
# Azure AD Applications

- Creating applications required for AAU authentication
  - Applications are as Native application or Web Applications
  - Application identified using GUID known as application ID
  - Application ID often referred to as client ID or app ID



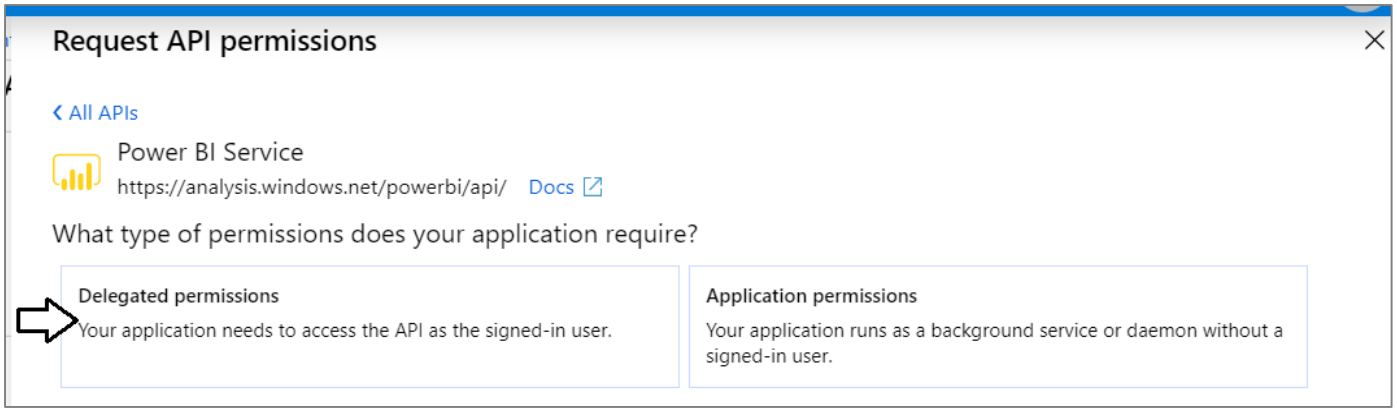
# Application Types

- Azure AD Application Types
  - Public client (mobile and desktop)
  - Web



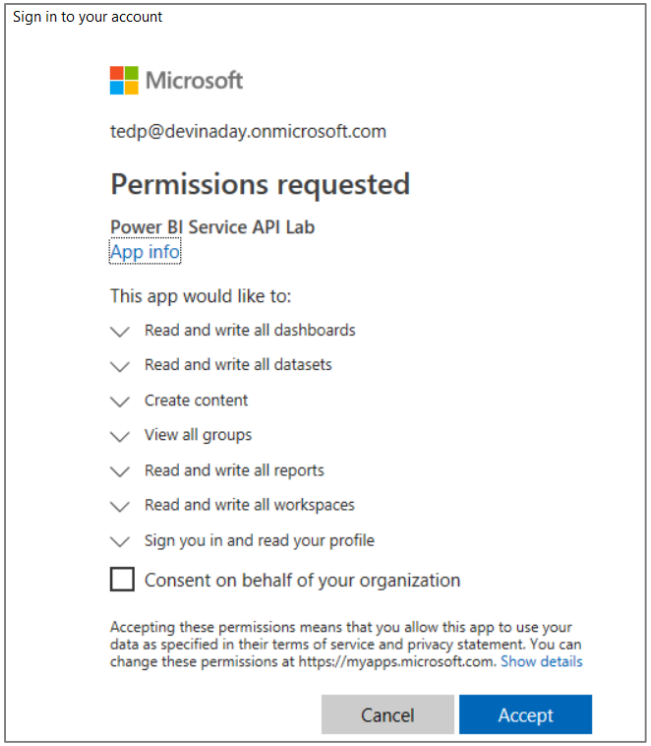
# Delegated Permissions vs Application Permissions

- Permissions categorized into two basic types
  - Delegated permissions are (app + user) permissions
  - Application permissions are app-only permissions (far more powerful)
  - Not all application types and APIs support application permissions
  - Power BI Service API does not support application permission



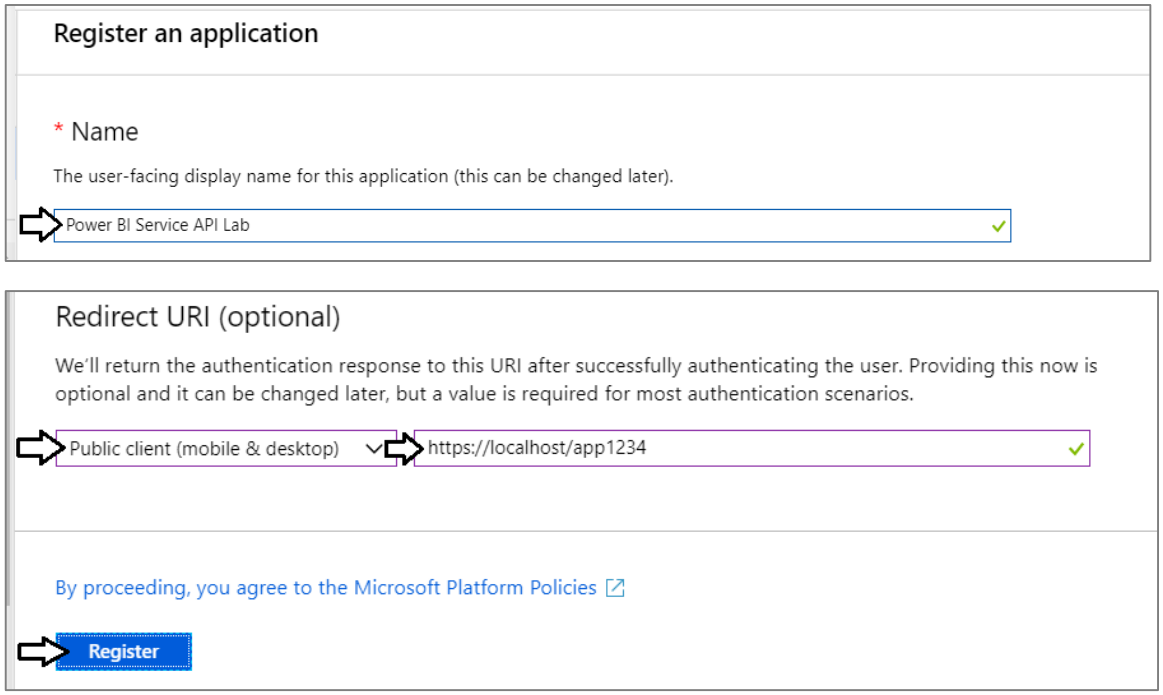
# Interactive Consent for Delegated Permissions

- Users must consent to delegated permissions
  - User prompted during first log in
  - User must click Accept
  - Only occurs once for each user



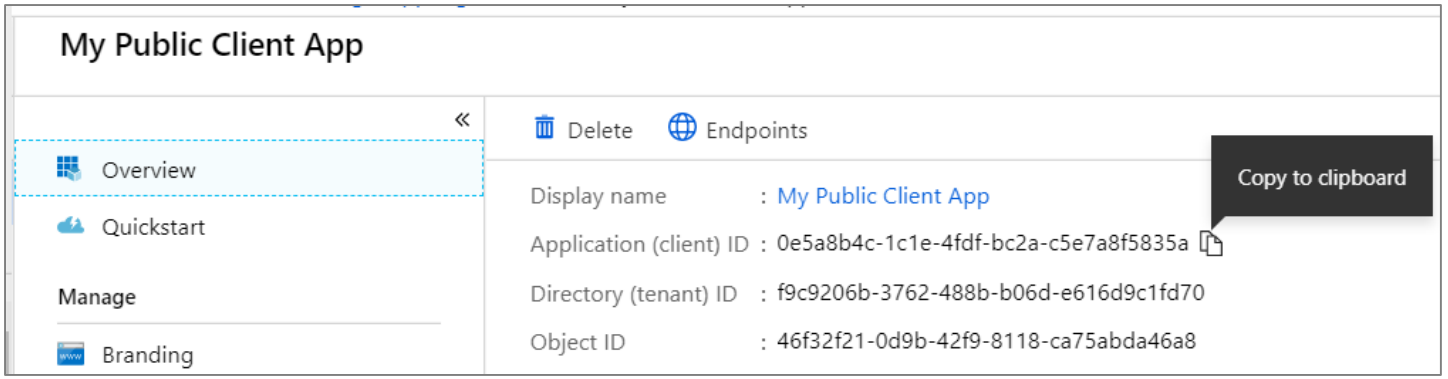
# Creating a Public Client Application

- Power BI supports Public Client Applications
  - Used for native applications and desktop applications
  - Requires Redirect URI for interactive logins



# Copying the Application ID

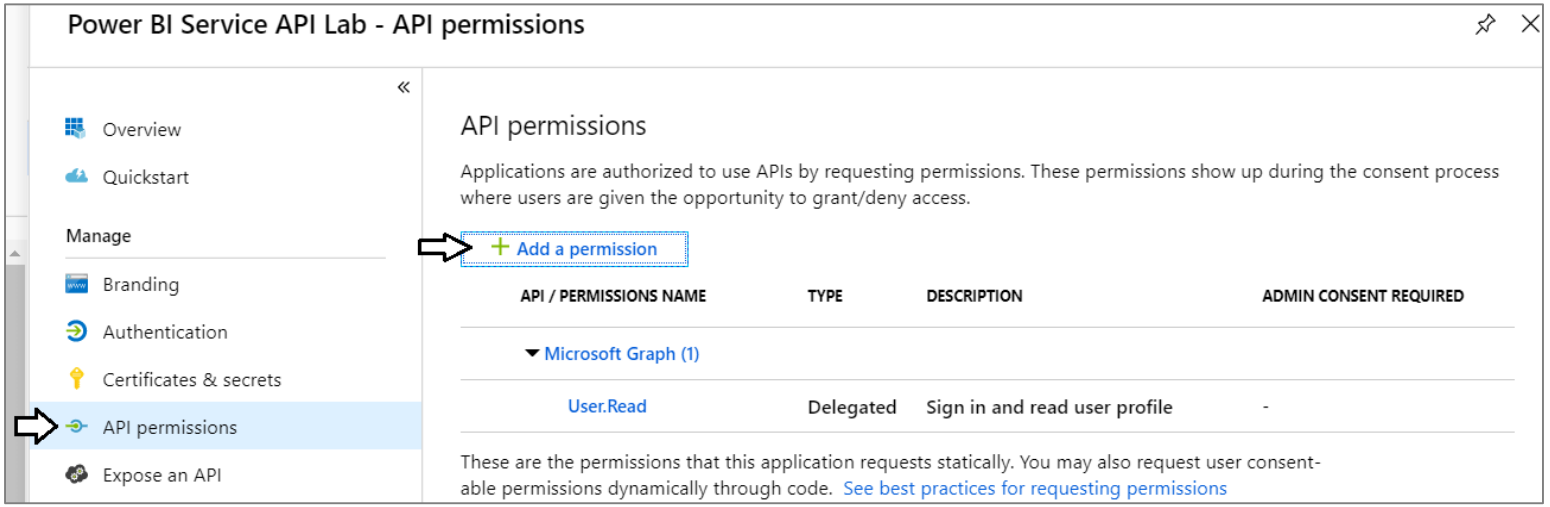
- Each new application created with Application ID
  - You cannot supply your own GUID for application ID
  - Azure AD will always create this GUID
  - You can copy the application ID from the Azure portal




- Don’t forget this confusing fact…
  - `Application ID == Client ID`

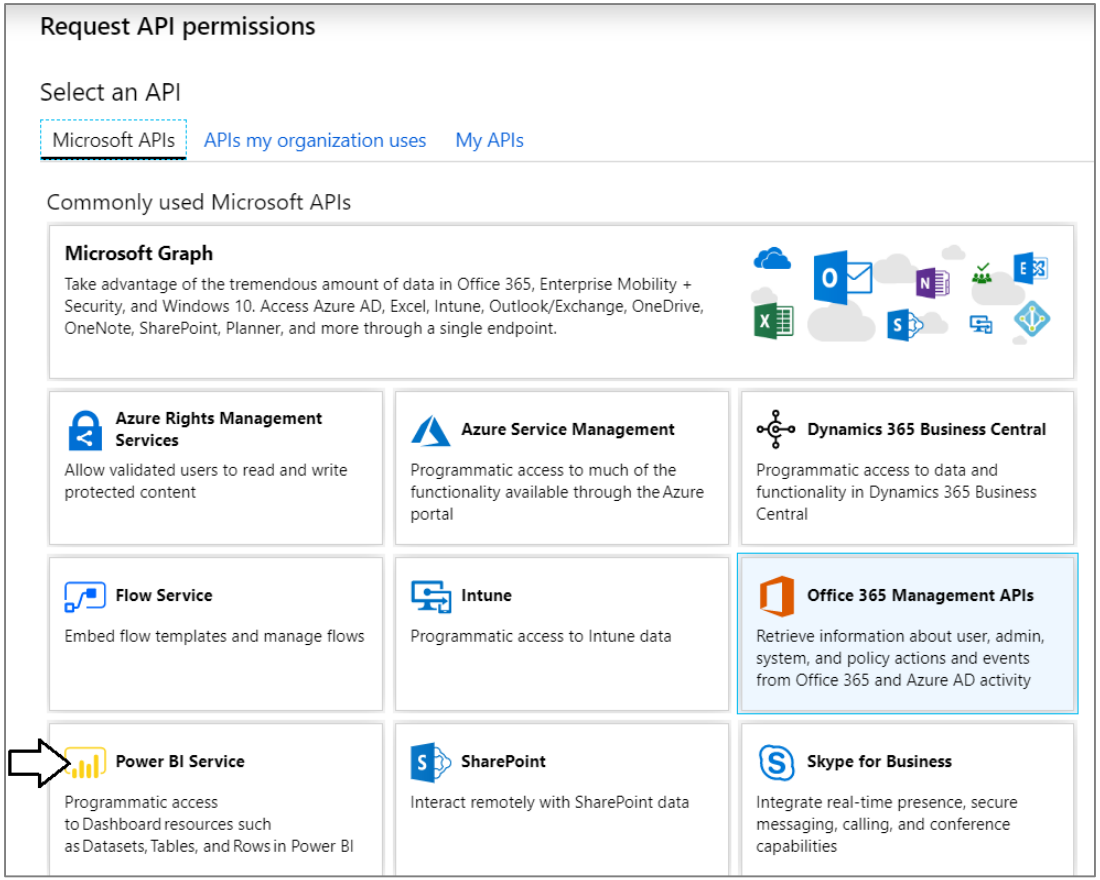
# Configuring Required Permissions

- Application configured with permissions
  - Default permissions allows user authentication – but that's it
  - To use APIs, you can assign permissions to the application



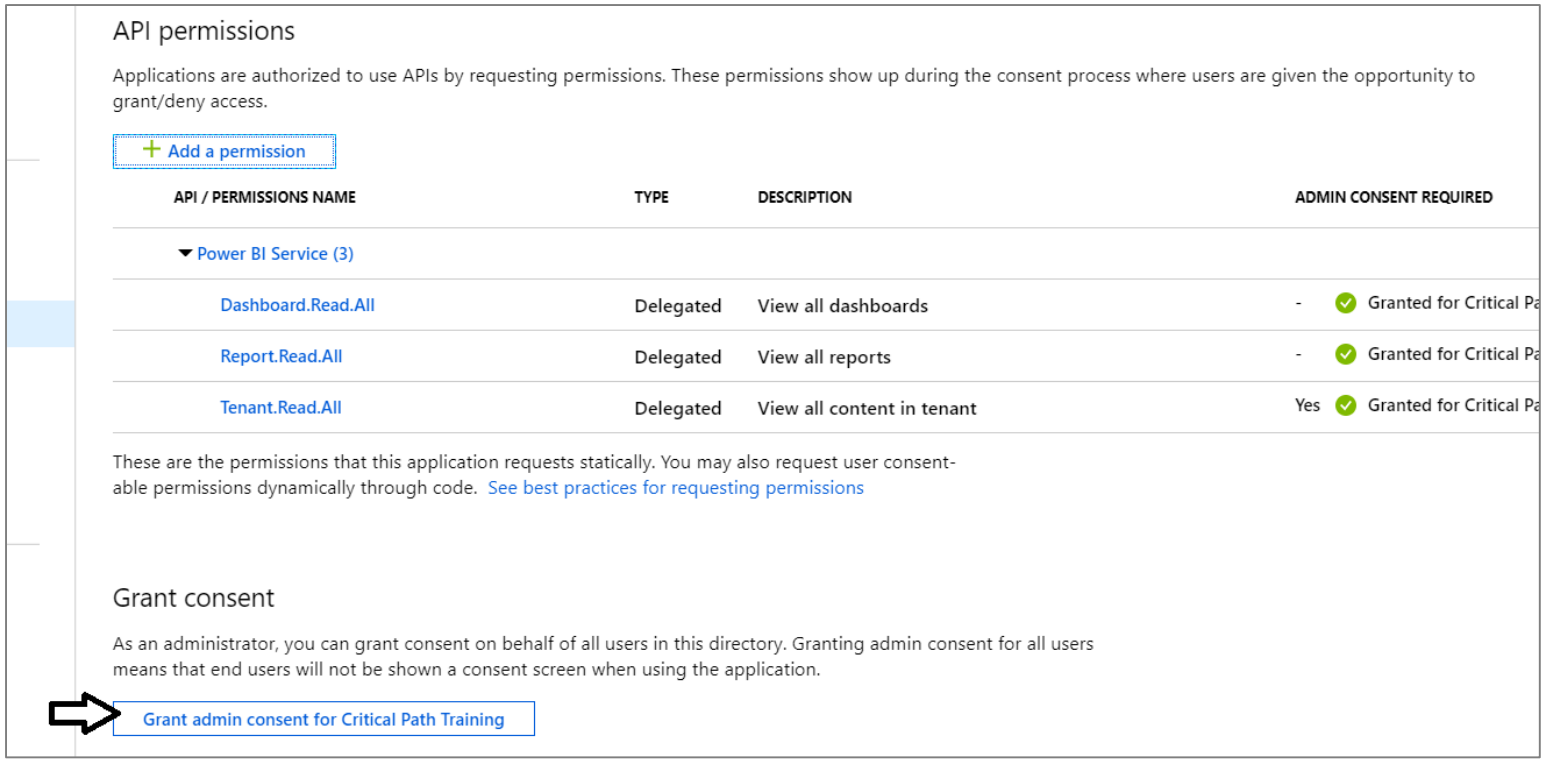
# Choosing an API

- There are lots of APIs to choose from
  - Microsoft Graph, Power BI Service, etc.



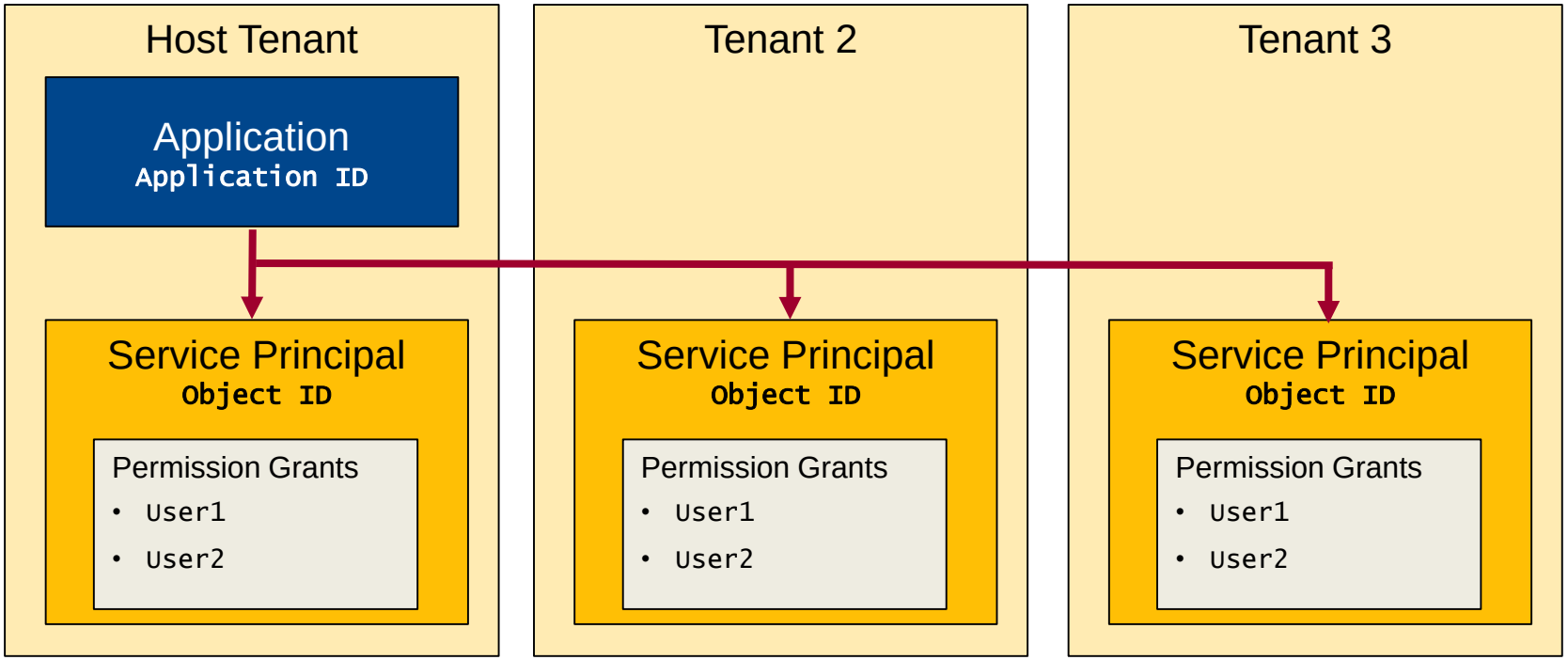
# Granting Delegated Permissions

- It can be helpful to Grant Permissions in Azure portal
  - Prevents the need for interactive granting of application by user
  - Might be required when authenticating in non-interactive fashion



# AAD Service Principals

- Azure AD creates service principal(s) for each application
  - Service principle created once per tenant
  - Service principle acts as first-class AAD security principal



# Registering AAD Apps with PowerShell

```powershell
$authResult = Connect-AzureAD

# display name for new public client app
$appDisplayName = "My Power BI Service App"

# get user account ID for logged in user
$user = Get-AzureADUser -ObjectId $authResult.Account.Id

# get tenant name of logged in user
$tenantName = $authResult.TenantDomain

# create Azure AD Application
$replyUrl = "https://localhost/app1234"
$aadApplication = New-AzureADApplication `
                        -DisplayName $appDisplayName `
                        -PublicClient $true `
                        -AvailableToOtherTenants $false `
                        -ReplyUrls @($replyUrl)

# create service principal for application
$appId = $aadApplication.AppId
$serviceServicePrincipal = New-AzureADServicePrincipal -AppId $appId

# assign current user as application owner
Add-AzureADApplicationOwner -ObjectId $aadApplication.ObjectId -RefObjectId $user.ObjectId
```

# Configuring Delegated Permissions

```
# create Azure AD Application
$replyUrl = "https://localhost/app1234"
$aadApplication = New-AzureADApplication `
                        -DisplayName $appDisplayName `
                        -PublicClient $true `
                        -AvailableToOtherTenants $false `
                        -ReplyUrls @($replyUrl)

# configure delegated permisssions for the Power BI Service API
$requiredAccess = New-Object -TypeName "Microsoft.Open.AzureAD.Model.RequiredResourceAccess"
$requiredAccess.ResourceAppId = "00000009-0000-0000-c000-000000000000"

# create first delegated permission - Report.Read.All
$permission1 = New-Object -TypeName "Microsoft.Open.AzureAD.Model.ResourceAccess" `
                          -ArgumentList "4ae1bf56-f562-4747-b7bc-2fa0874ed46f","Scope"

# create second delegated permission - Dashboards.Read.All
$permission2 = New-Object -TypeName "Microsoft.Open.AzureAD.Model.ResourceAccess" `
                          -ArgumentList "2448370f-f988-42cd-909c-6528efd67c1a","Scope"

# add permissions to ResourceAccess list
$requiredAccess.ResourceAccess = $permission1, $permission2

# add permissions by updating application with RequiredResourceAccess object
Set-AzureADApplication -ObjectId $aadApplication.ObjectId -RequiredResourceAccess $requiredAccess
```

# Agenda

- ✓ Power BI Service API Overview
- ✓ Understanding Authentication with Azure AD
- ➢ Programming with the Power BI .NET SDK
- Acquiring Access Tokens using MSAL
- Calling to Power BI using App-only Tokens

# Interactive Access Token Acquisition

## Using ADAL with public client application

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

static string clientId = "183a7832-6792-4476-be85-82aab1824d9a";
static string redirectUrl = "https://localhost/app1234";

static string GetAccessTokenInteractive() {

  // 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;

}
```

# User Password Credential Flow

## Using ADAL with public client application

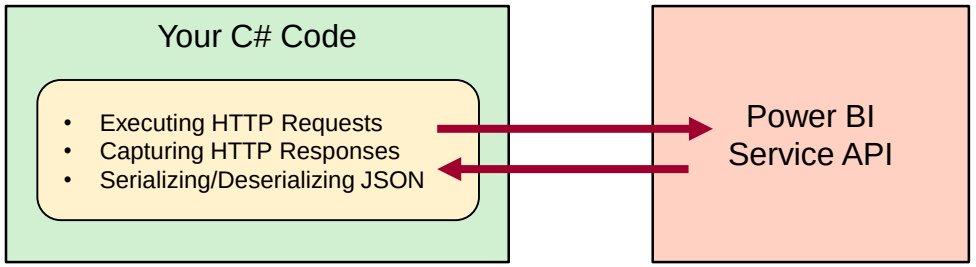
```
static string aadAuthorizationEndpoint = "https://login.windows.net/common";
static string resourceUriPowerBi = "https://analysis.windows.net/powerbi/api";
static string urlPowerBiRestApiRoot = "https://api.powerbi.com/";

static string clientId = "183a7832-6792-4476-be85-82aab1824d9a";
// static string redirectUrl = "https://localhost/app1234";

static string GetAccessTokenWithUserPassword() {

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

  // use authentication context to sign-in using User Password Credentials flow
  string userAccount = "chuckster@devinaday2019.onMicrosoft.com";
  string userPassword = "myCAT$rightLEG";
  UserPasswordCredential creds = new UserPasswordCredential(userAccount, userPassword);

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

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

# Calling the Power BI Service API

## Direct REST calls without using the Power BI .NET SDK

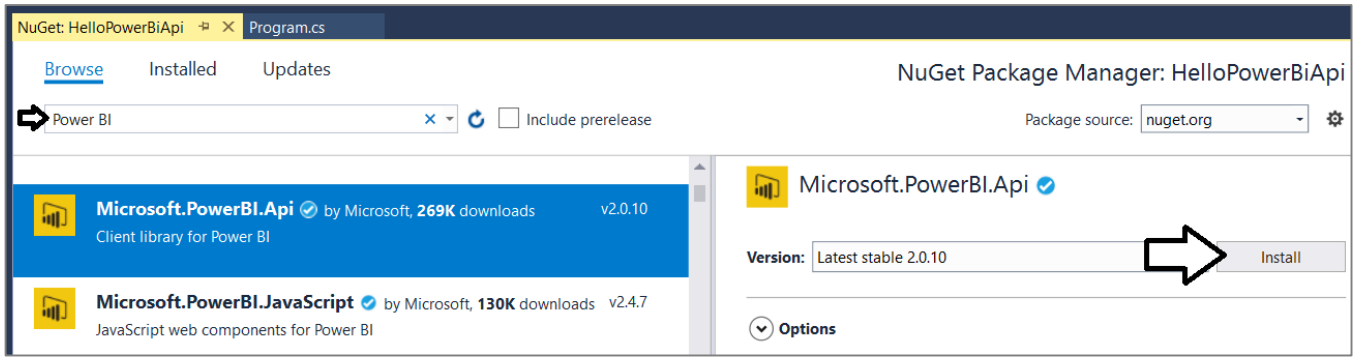
```
static string ExecuteGetRequest(string restUrl) {
  HttpClient client = new HttpClient();
  HttpRequestMessage request = new HttpRequestMessage(HttpMethod.Get, restUrl);
  request.Headers.Add("Authorization", "Bearer " + GetAccessToken());
  request.Headers.Add("Accept", "application/json;odata.metadata=minimal");
  HttpResponseMessage response = client.SendAsync(request).Result;
  if (response.StatusCode != HttpStatusCode.OK) {
    throw new ApplicationException("Error occured calling the Power BI Servide API");
  }
  return response.Content.ReadAsStringAsync().Result;
}

static void Main() {
  // get report data from app workspace
  string restUrl = "https://api.powerbi.com/v1.0/myorg/groups/" + appWorkspaceId + "/reports/";
  var json = ExecuteGetRequest(restUrl);
  ReportCollection reports = JsonConvert.DeserializeObject<ReportCollection>(json);
  foreach (Report report in reports.value) {
    Console.WriteLine("Report Name: " + report.name);
    Console.WriteLine();
  }
}
```

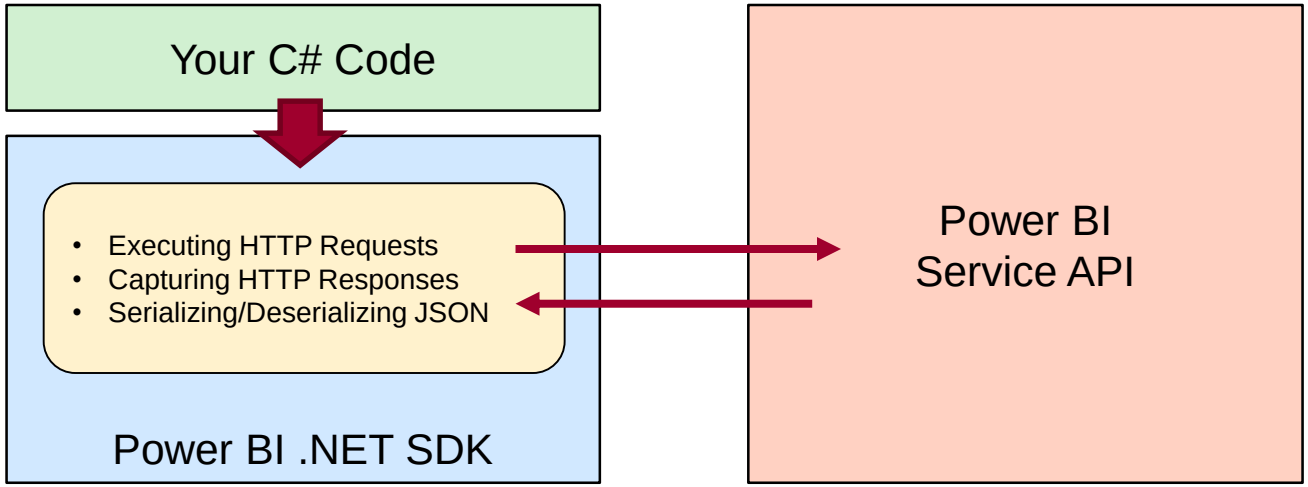
```
public class Report {
  public string id { get; set; }
  public string name { get; set; }
  public string webUrl { get; set; }
  public string embedUrl { get; set; }
  public bool isOwnedByMe { get; set; }
  public string datasetId { get; set; }
}

public class ReportCollection {
  public List<Report> value { get; set; }
}
```



# Power BI .NET SDK

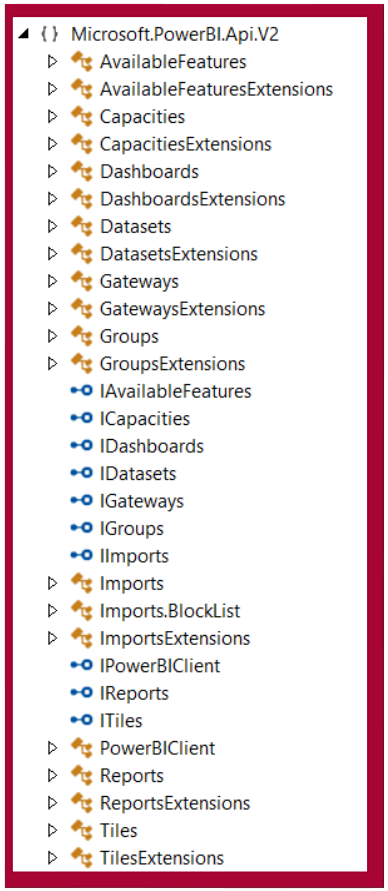
- Added as a NuGet package

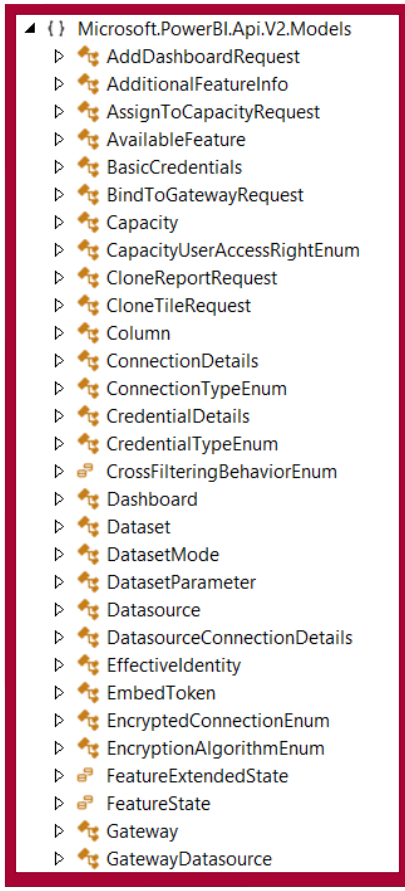


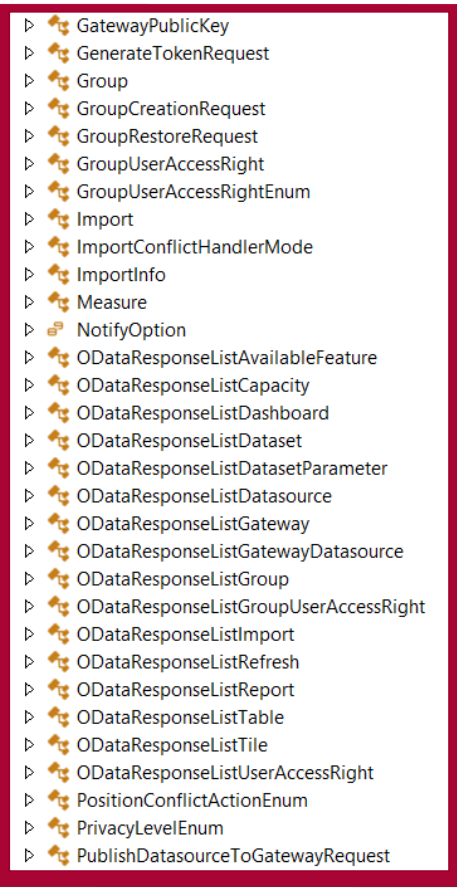



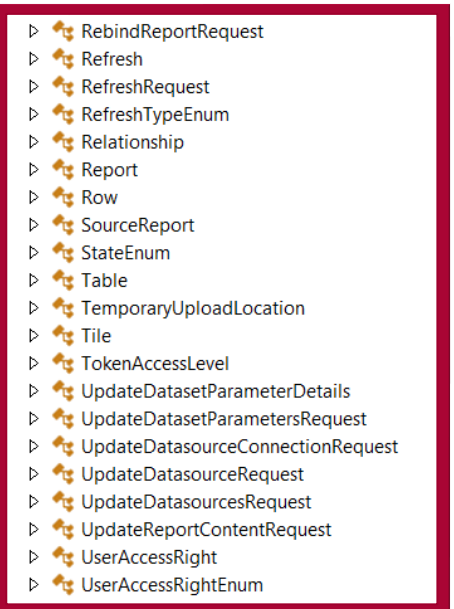
# The Power BI .NET SDK Classes

- SDK provides object model of classes



# Initializing an Instance of PowerBIClient

- PowerBIClient object serves as top-level object
  - Used to execute calls against Power BI Service
  - Initialized with function to retrieve AAD access token

```
static string GetAccessToken() ...

static PowerBIClient GetPowerBiClient() {
  var tokenCredentials = new TokenCredentials(GetAccessToken(), "Bearer");
  return new PowerBIClient(new Uri(urlPowerBiRestApiRoot), tokenCredentials);
}

static void Main() {
  PowerBIClient pbiClient = GetPowerBiClient();
  var reports = pbiClient.Reports.GetReports().Value;
  foreach (var report in reports) {
    Console.WriteLine(report.Name);
  }
}
```

# Enumerating Collections with PowerBiClient

```
static void DisplayAppWorkspaceAssets() {

  PowerBIClient pbiClient = GetPowerBiClient();

  Console.WriteLine("Listing assets in app workspace: " + appWorkspaceId);

  Console.WriteLine("Datasets:");
  var datasets = pbiClient.Datasets.GetDatasetsInGroup(appWorkspaceId).Value;
  foreach (var dataset in datasets) {
    Console.WriteLine(" - " + dataset.Name + " [" + dataset.Id + "]");
  }

  Console.WriteLine();
  Console.WriteLine("Reports:");
  var reports = pbiClient.Reports.GetReportsInGroup(appWorkspaceId).Value;
  foreach (var report in reports) {
    Console.WriteLine(" - " + report.Name + " [" + report.Id + "]");
  }

  Console.WriteLine();
  Console.WriteLine("Dashboards:");
  var dashboards = pbiClient.Dashboards.GetDashboardsInGroup(appWorkspaceId).Value;
  foreach (var dashboard in dashboards) {
    Console.WriteLine(" - " + dashboard.DisplayName + " [" + dashboard.Id + "]");
  }
}
```

# Creating Workspaces and Importing Content

```
public static async Task<Group> CreateWorkspacesAsync(string WorkspaceName) {

  PowerBIClient pbiClient = GetPowerBiClient();
  GroupCreationRequest createRequest = new GroupCreationRequest(WorkspaceName);
  var workspace = await pbiClient.Groups.CreateGroupAsync(createRequest);

  var secondaryAdmin = "pbiemasteruser@sharepointconfessions.onmicrosoft.com";
  var userRights = new GroupUserAccessRight("Admin", secondaryAdmin);
  await pbiClient.Groups.AddGroupUserAsync(workspace.Id, userRights);

  return workspace;

}
```

```
public static async Task UploadPBIX(string WorkspaceId, string pbixName, string importName, bool updateSqlC

  string PbixFilePath = HttpContext.Current.Server.MapPath("/PBIX/" + pbixName);
  PowerBIClient pbiClient = GetPowerBiClient();
  FileStream stream = new FileStream(PbixFilePath, FileMode.Open, FileAccess.Read);
  var import = await pbiClient.Imports.PostImportWithFileAsyncInGroup(WorkspaceId, stream, importName);

  if (updateSqlCredentials) {
    await PatchSqlDatasourceCredentials(WorkspaceId, importName);
  }

  return;
}
```

# Patching Datasource Credentials

```
public static async Task PatchSqlDatasourceCredentials(string WorkspaceId, string importName) {
  PowerBIClient pbiClient = GetPowerBiClient();
  var datasets = (await pbiClient.Datasets.GetDatasetsInGroupAsync(WorkspaceId)).Value;
  foreach (var dataset in datasets) {
    if (importName.Equals(dataset.Name)) {
      string datasetId = dataset.Id;
      var datasources = (await pbiClient.Datasets.GetDatasourcesInGroupAsync(WorkspaceId, datasetId)).Value;
      foreach (var datasource in datasources) {
        if (datasource.DatasourceType == "SQL") {
          var datasourceId = datasource.DatasourceId;
          var gatewayId = datasource.GatewayId;
          // create credentials for Azure SQL database log in
          Creds.BasicCredentials creds = new Creds.BasicCredentials("CptStudent", "pass@word1");
          CredentialDetails details = new CredentialDetails(creds);
          UpdateDatasourceRequest req = new UpdateDatasourceRequest(details);
          // Update credentials through gateway
          await pbiClient.Gateways.UpdateDatasourceAsync(gatewayId, datasourceId, details);
        }
      }
    }
  }
  return;
}
```

# Exporting/Importing PBIX Files

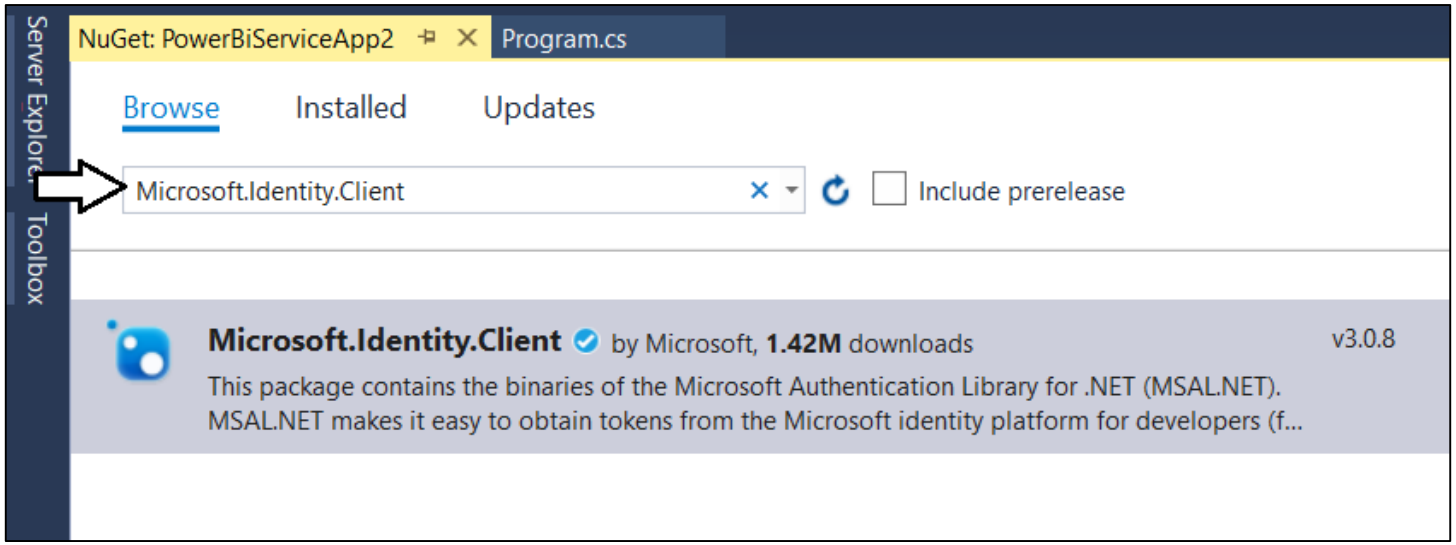
```
var reports = pbiClient.Reports.GetReportsInGroup(sourceAppWorkspaceId).Value;

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");
```

# Agenda

- ✓ Power BI Service API Overview
- ✓ Understanding Authentication with Azure AD
- ✓ Programming with the Power BI .NET SDK
- ➢ Acquiring Access Tokens using MSAL
- • Calling to Power BI using App-only Tokens

# Microsoft Authentication Library (.NET)

- Developing with the Microsoft Authentication Library
  - Provides access to Azure AD V2 Endpoint
  - Added to project as `Microsoft.Identity.Client` NuGet package
  - Provides different classes for *public clients* vs *confidential clients*



# Power BI Service API Scopes

- Azure AD V2 endpoint requires passing scopes
  - Scopes define permissions required in access token
  - Scopes defined as resource + permission
    `https://analysis.windows.net/powerbi/api/` + `Report.ReadWrite.All`

```
static string[] scopesDefault = new string[] {
        "https://analysis.windows.net/powerbi/api/.default"
};

static string[] scopesReadWorkspaceAssets = new string[] {
        "https://analysis.windows.net/powerbi/api/Dashboard.Read.All",
        "https://analysis.windows.net/powerbi/api/Dataset.Read.All",
        "https://analysis.windows.net/powerbi/api/Report.Read.All"
};

static string[] scopesReadUserApps = new string[] {
        "https://analysis.windows.net/powerbi/api/App.Read.All"
};

static string[] scopesManageWorkspaceAssets = new string[] {
        "https://analysis.windows.net/powerbi/api/Content.Create",
        "https://analysis.windows.net/powerbi/api/Dashboard.ReadWrite.All",
        "https://analysis.windows.net/powerbi/api/Dataset.ReadWrite.All",
        "https://analysis.windows.net/powerbi/api/Group.Read.All",
        "https://analysis.windows.net/powerbi/api/Report.ReadWrite.All",
        "https://analysis.windows.net/powerbi/api/Workspace.ReadWrite.All"
};
```

# Interactive Access Token Acquisition

## Using MSAL with public client application

- Flow implemented using `PublicClientApplication` object
  - Created using `PublicClientApplicationBuilder` object
  - Requires passing redirect URI
  - You can control prompting behavior

```
static string GetAccessTokenInteractive(string[] scopes) {

  var appPublic = PublicClientApplicationBuilder.Create(clientId)
                    .WithAuthority(tenantCommonAuthority)
                    .WithRedirectUri(redirectUri)
                    .Build();

  var authResult = appPublic.AcquireTokenInteractive(scopes)
                              .WithPrompt(Prompt.SelectAccount)
                              .ExecuteAsync().Result;

  return authResult.AccessToken;
}
```

# User Credential Password Flow

## Using MSAL with public client application

- MSAL supports user credential password flow
  - Supported in .NET runtime but not in .NET CORE
  - Microsoft recommends against using this flow

```
static string GetAccessTokenWithUserPassword(string[] scopes) {

  var appPublic = PublicClientApplicationBuilder.Create(clientId)
                    .WithAuthority(tenantCommonAuthority)
                    .Build();

  string username = "chuckster@devinaday2019.onMicrosoft.com";
  string userPassword = "myCAT$rightLEG";
  SecureString userPasswordSecure = new SecureString();
  foreach (char c in userPassword) {
    userPasswordSecure.AppendChar(c);
  }

  var authResult = appPublic.AcquireTokenByUsernamePassword(scopes, username, userPasswordSecure)
                            .ExecuteAsync().Result;

  return authResult.AccessToken;
}
```

# Device Code Flow

## Using MSAL with public client application

- MSAL introduced this new flow with MSAL
  - Much more secure than user password credential flow
  - Not available in ADAL

```
static string GetAccessTokenWithDeviceCode(string[] scopes) {

  // device code authentication requires tenant-specific authority URL
  var appPublic = PublicClientApplicationBuilder.Create(clientId)
                    .WithAuthority(tenantSpecificAuthority)
                    .Build();

  // this method call will block until you have logged in using the generated device code
  var authResult = appPublic.AcquireTokenWithDeviceCode(scopes, deviceCodeCallbackParams => {

    // retrieve device code and verification URL from deviceCodeCallbackParams
    string deviceCode = deviceCodeCallbackParams.UserCode;
    string verificationUrl = deviceCodeCallbackParams.VerificationUrl;

    Console.WriteLine("When prompted by the browser, copy-and-paste the following device code: " + deviceCode);

    Console.WriteLine("Opening Browser at " + verificationUrl);
    Process.Start("chrome.exe", verificationUrl);

    Console.WriteLine("This console app will now block until you enter the device code and log in");

    // return task result
    return Task.FromResult(0);
  }).ExecuteAsync().Result;

  Console.WriteLine("The call to AcquireTokenWithDeviceCode has completed and returned an access token");

  return authResult.AccessToken;
}
```

# Calling into the Power BI Admin API

- Admin API exposed using **`AsAdmin`** methods
  - Example: `pbiClient.Groups.GetGroupsAsAdmin(top: 100).Value;`
  - Makes it possible to access every workspace in current tenant
  - Requires access token for user who is tenant or Power BI admin

```
static void DisplayAllWorkspacesInTenant() {

  string[] scopesTenantAdmin = new string[] {
      "https://analysis.windows.net/powerbi/api/Tenant.ReadWrite.All", // requires admin
  };

  string AccessToken = GetAccessTokenInteractive(scopesKitchenSink);

  var pbiClient = new PowerBIClient(new Uri(urlPowerBiRestApiRoot),
                                    new TokenCredentials(AccessToken, "Bearer"));

  Console.WriteLine("Display All Workpaces in Tenant using GetGroupsAsAdmin:");
  var workspaces = pbiClient.Groups.GetGroupsAsAdmin(top: 100).Value;

  foreach (var workspace in workspaces) {
    Console.WriteLine("- " + workspace.Type + ": " + workspace.Name + " [" + workspace.Id + "] ");
  }
  Console.WriteLine();
}
```

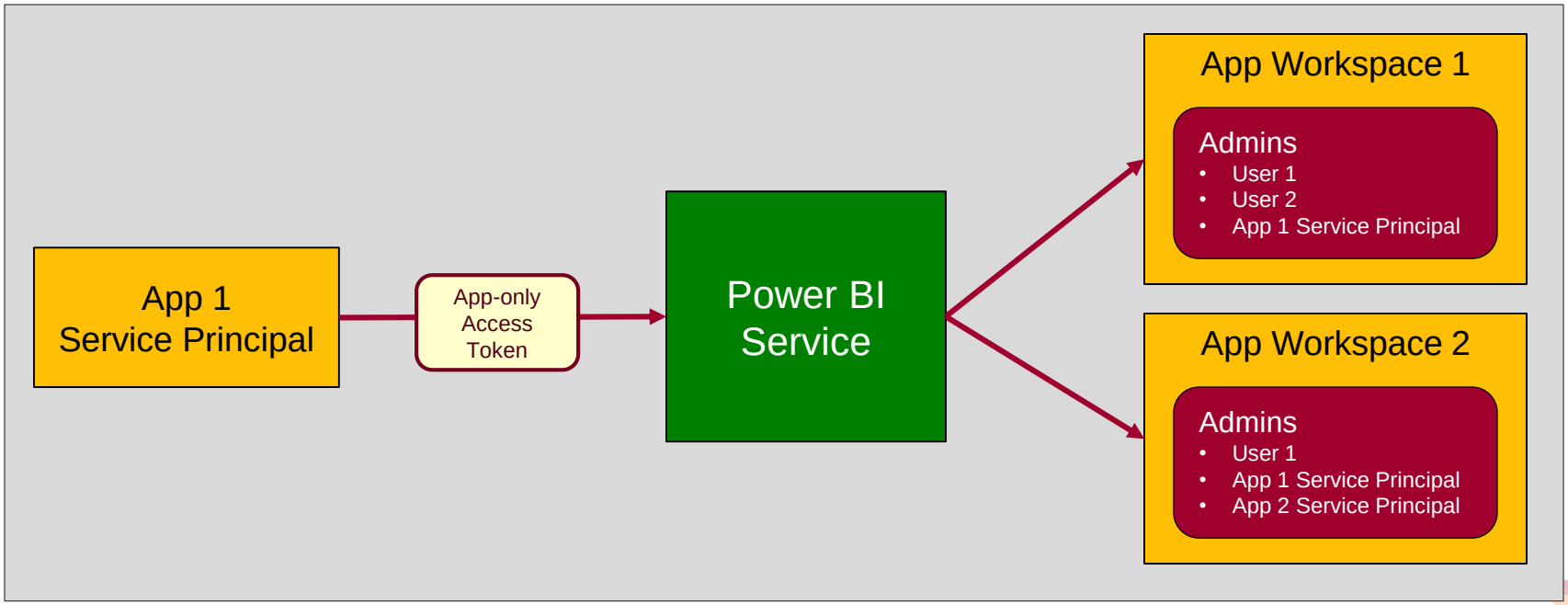

DEMO
Authenticating with MSAL

# Agenda

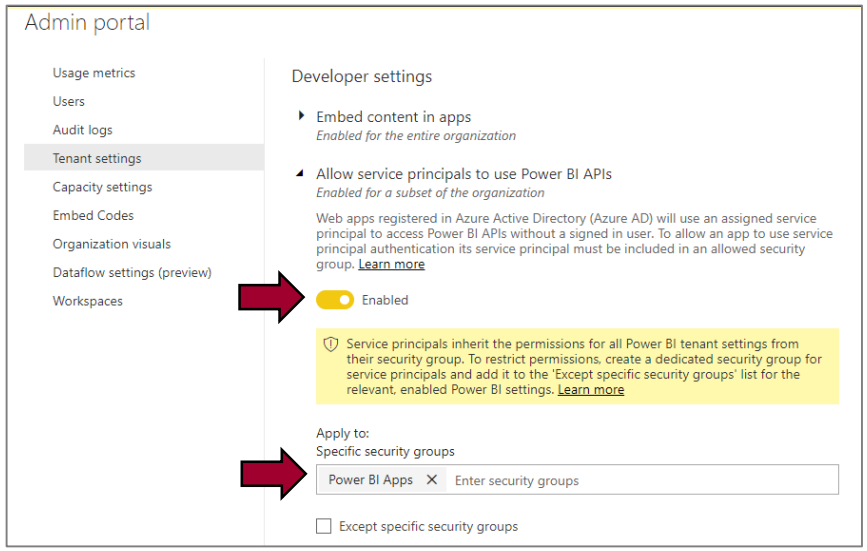
- ✓ Power BI Service API Overview
- ✓ Understanding Authentication with Azure AD
- ✓ Acquiring Access Tokens using ADAL
- ✓ Programming with the Power BI .NET SDK
- ✓ Acquiring Access Tokens using MSAL
- ➢ Calling to Power BI using App-only Tokens

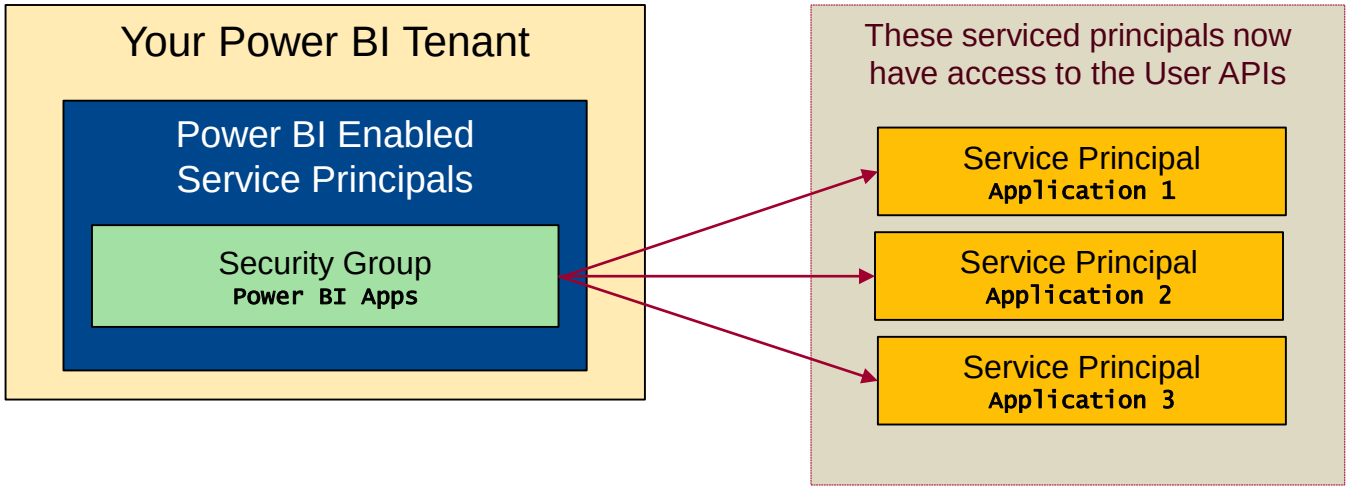
# App-only Access Control

- Service Principal used to configure access control
  - Requires the use of v2 app workspaces
  - Service principal added to app workspaces as admin
  - Access control NOT based on Azure AD permissions



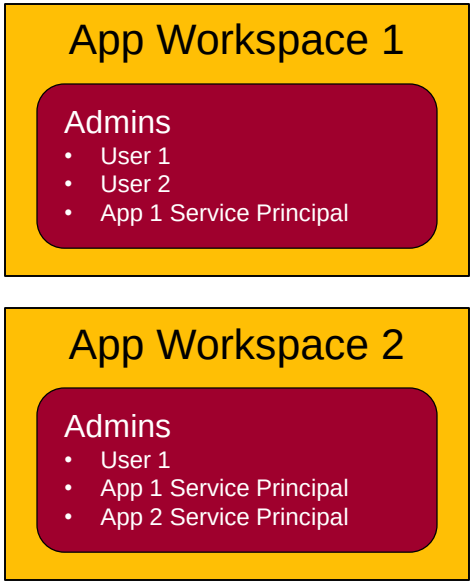
# Tenant Setup

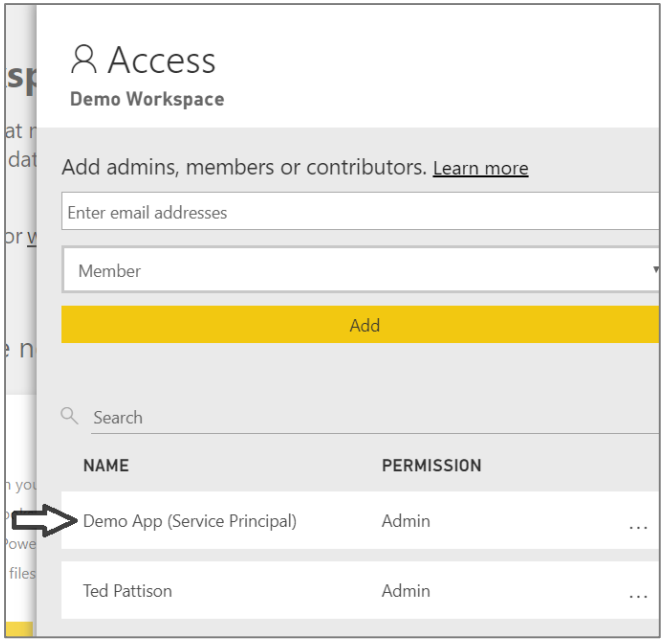


# App-only Access with PBI Service API

- Service Principal added to workspace as admin
  - Only works with v2 app workspaces
  - Provides full workspace access to service principal



# Client Credentials Flow

## Using MSAL with confidential client application

- Client credentials flow used to obtain app-only token
  - Requires passing app secret (e.g. app password or certificate)
  - Requires passing tenant-specific endpoint

```
const string clientId = "e6a54dc4-7345-495d-b029-88c6349b62d2";
const string clientSecret = "M2MwODBhOTEtOWUyYi00NWQ1LWJmMTQtMjM1ZTAzMzZjOTMx=";
const string tenantName = "devinaday2019.onmicrosoft.com";

// endpoint for tenant-specific authority
const string tenantSpecificAuthority = "https://login.microsoftonline.com/" + tenantName;

static string GetAppOnlyAccessToken() {

  var appConfidential = ConfidentialClientApplicationBuilder.Create(clientId)
                        .WithClientSecret(clientSecret)
                        .WithAuthority(tenantSpecificAuthority)
                        .Build();

  string[] scopesDefault = new string[] { "https://analysis.windows.net/powerbi/api/.default" };

  var authResult = appConfidential.AcquireTokenForClient(scopesDefault).ExecuteAsync().Result;

  return authResult.AccessToken;
}
```

# Summary

- ✓ Power BI Service API Overview
- ✓ Understanding Authentication with Azure AD
- ✓ Acquiring Access Tokens using ADAL
- ✓ Programming with the Power BI .NET SDK
- ✓ Acquiring Access Tokens using MSAL
- ✓ Calling to Power BI using App-only Tokens