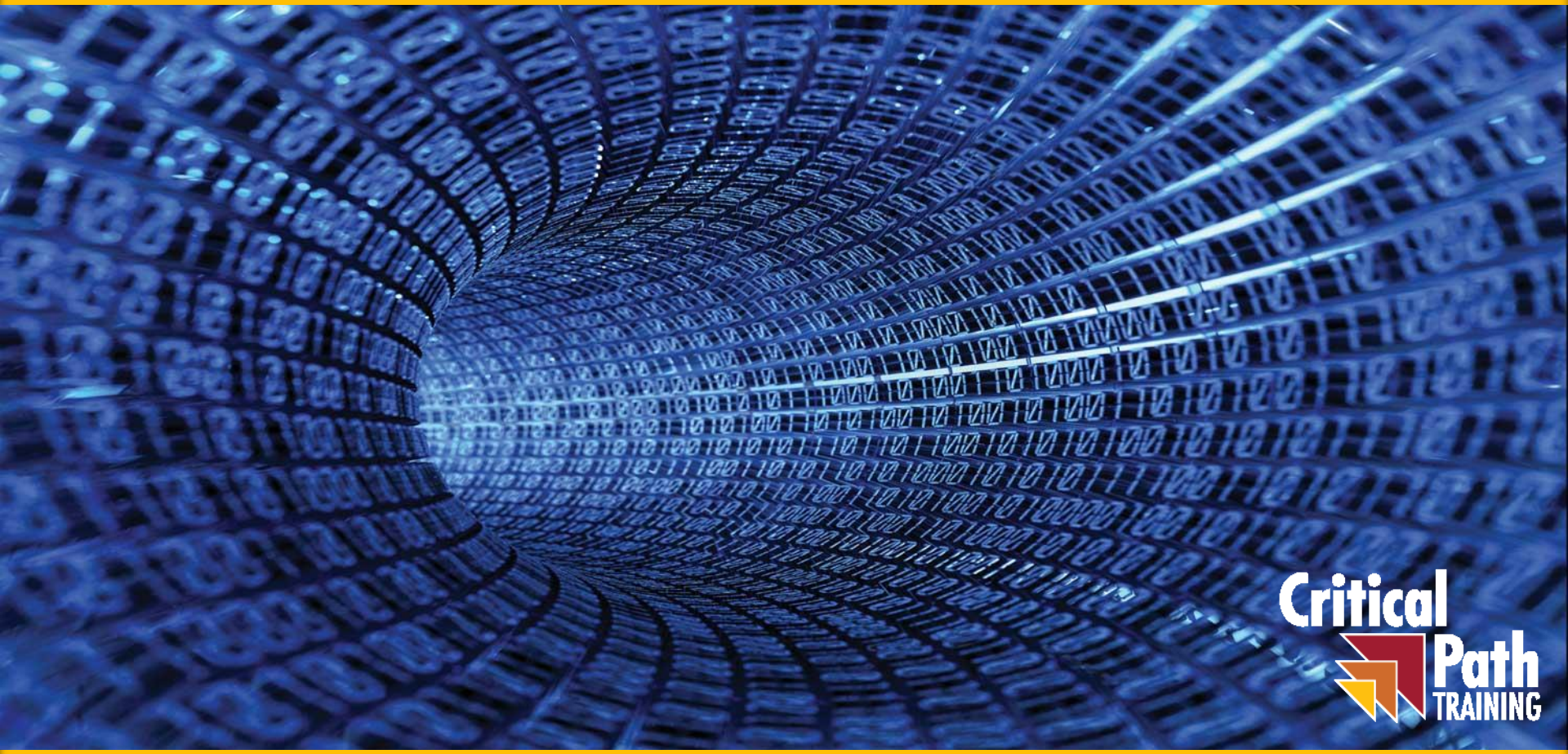
Developing with the Power BI Service API



Agenda

- Power BI Service API Overview
- Authentication with Azure Active Directory
- Developing with the Power BI SDK
- Creating and Managing Workspaces

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

Getting Started

- What you need to get started?
  - Visual Studio 2017 or Visual Studio 2015
  - Organizational account in an Azure AD tenancy
  - License for Power BI Pro
  - Access to Azure portal to create Azure AD applications

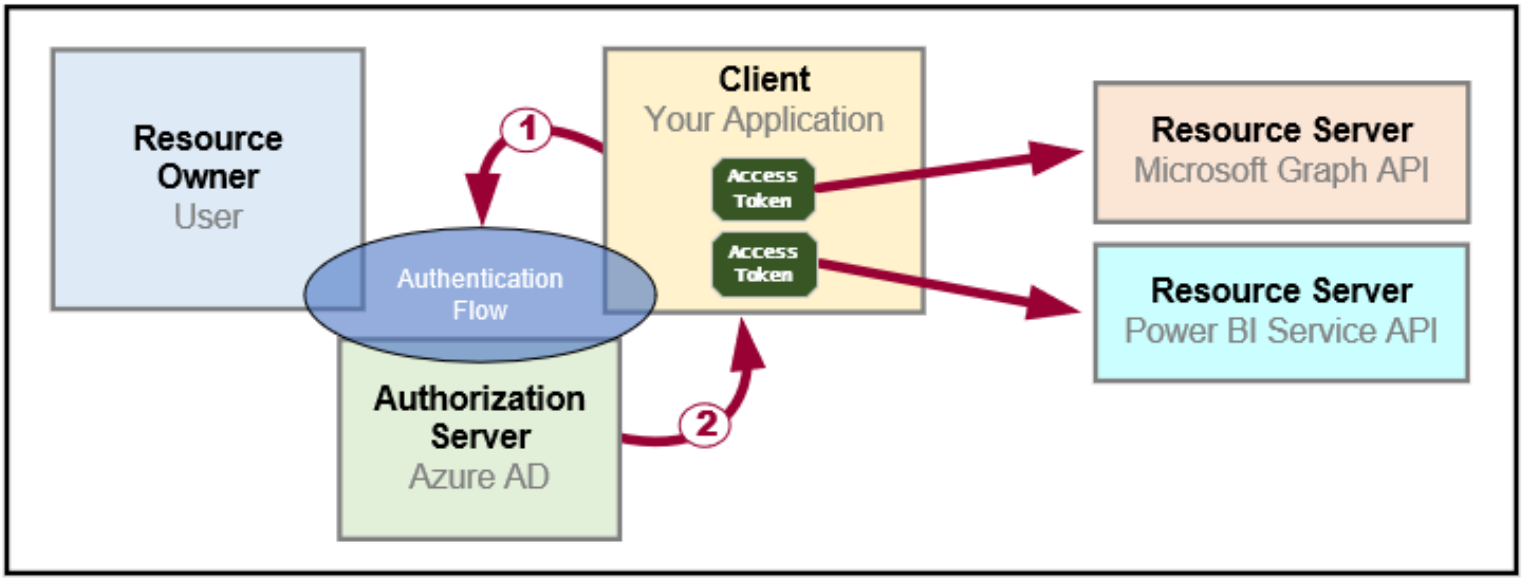
- Azure subscription not required!
  - Azure portal used to create Azure AD application
  - Azure subscription helpful to create Azure resources

Agenda

- ✓ Power BI Service API Overview
- ➢ Authentication with Azure Active Directory
- Developing with the Power BI SDK
- Creating and Managing Workspaces

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



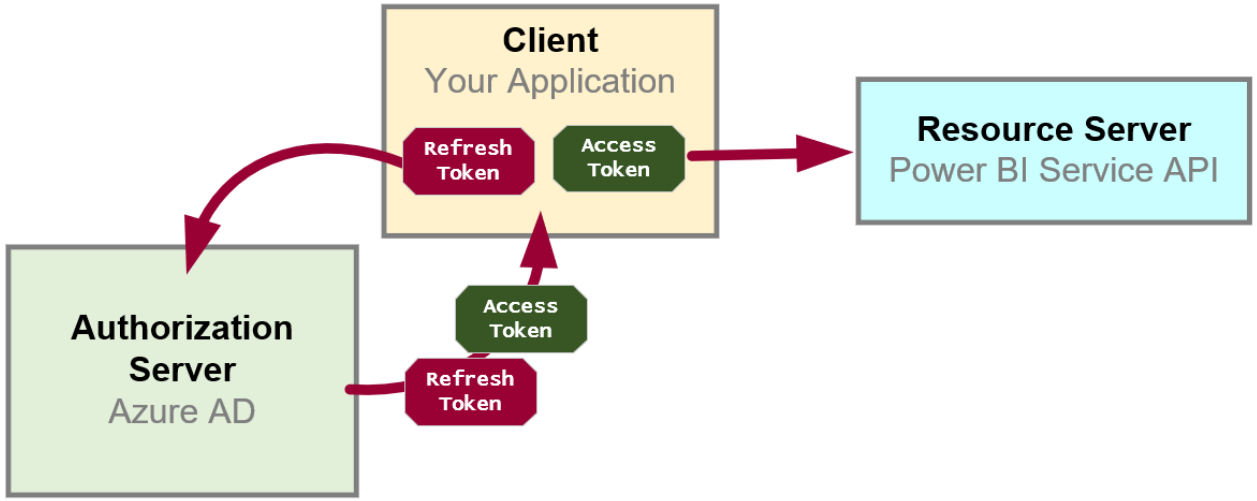
Access Token is a Bearer Token

- 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"
}
```

Refresh Tokens

- OAuth 2.0 provide solution for access token expiration
  - Access tokens have default lifetime of 60 minutes
  - Authorization server passes refresh token along with access token
  - Refresh token used as a credential to redeem new access token
  - Refresh token default lifetime is 14 days (max 90 days)
  - Refresh tokens often persistent in database or browser storage
  - Refresh tokens lesson need for user to enter security credentials



Authentication Flows

- **User Password Credential Flow** *(public client)*
  - Used in Native clients to obtain access code
  - Requires passing user name and password across network
- **Authorization Code Flow** *(confidential client)*
  - Client first obtains authorization code then access token
  - Access token acquired in server-to-server call
  - Access token never passes through browser or client device
- **Implicit Flow** *(public client)*
  - Used in SPAs built with JavaScript and AngularJS
  - Application obtains access token w/o acquiring authorization code
- **Client Credentials Flow** *(confidential client)*
  - Authentication based on SSL certificate with public-private key pair
  - Used to obtain access token when using app-only permissions

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

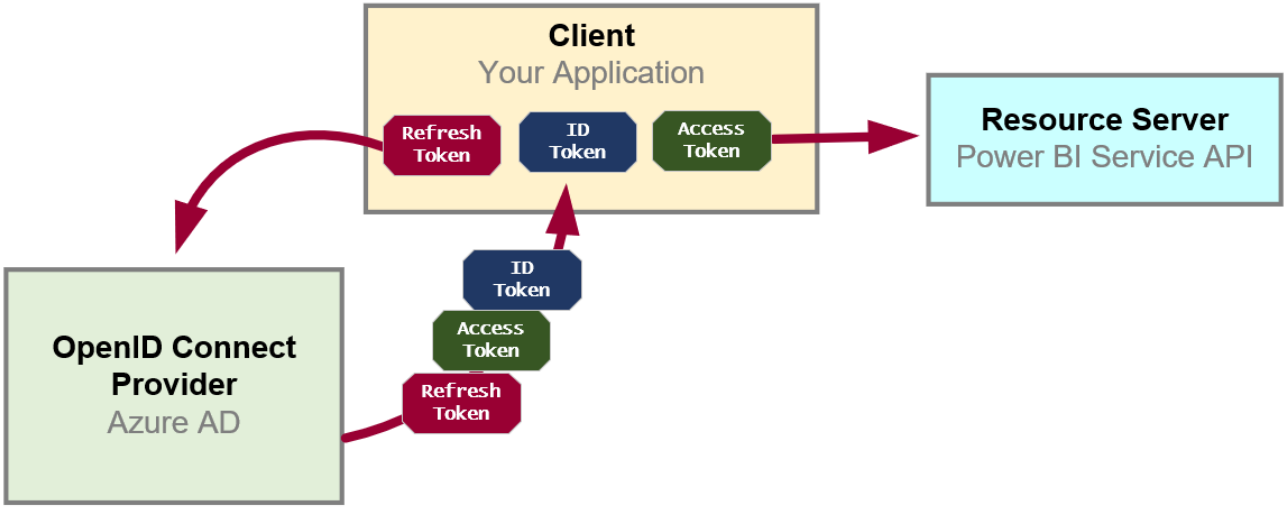
Azure AD

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

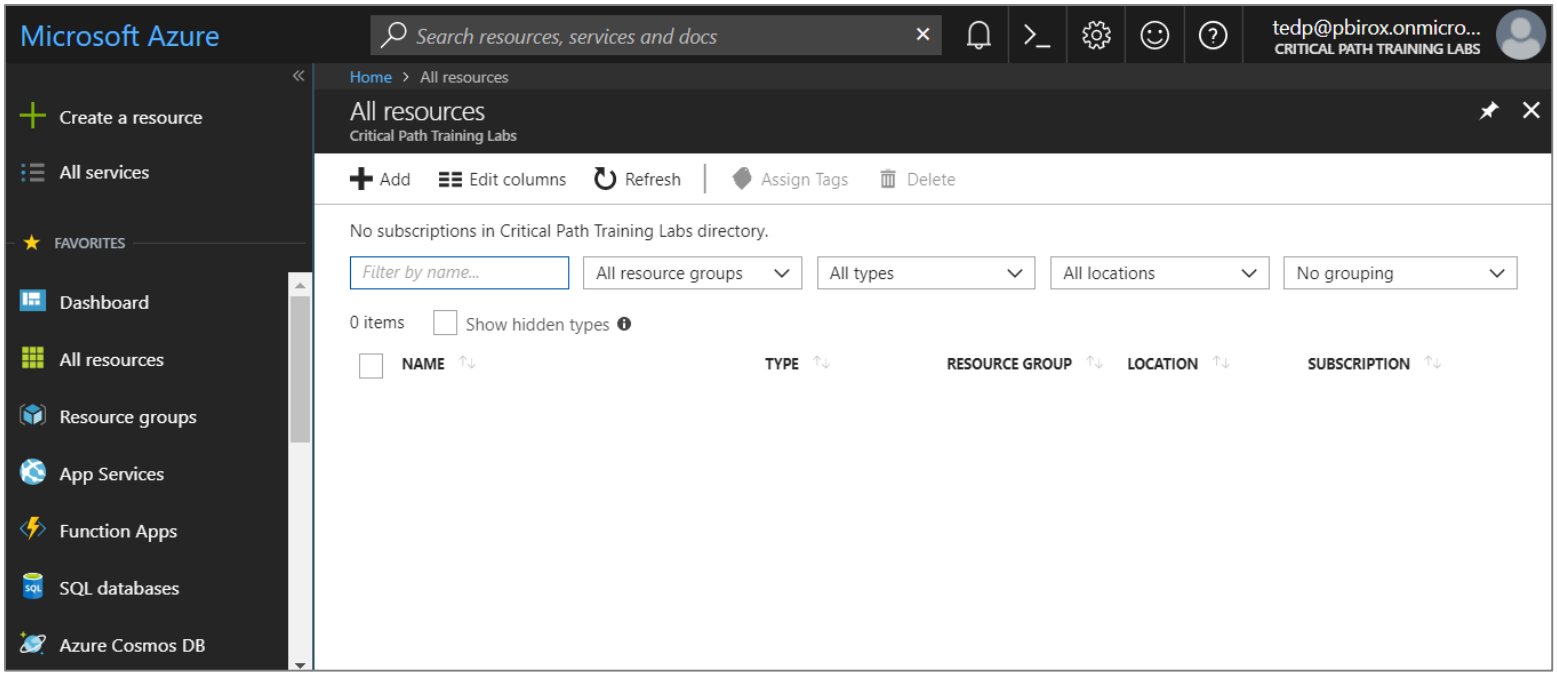


Agenda

- ✓ Understanding OAuth 2.0 and OpenID Connect
- ➢ Creating & Configuring Azure AD Applications
- Acquiring Access Tokens using ADAL.NET
- Understanding OWIN Security Middleware
- Implementing OpenID Connect using OWIN Middleware
- Acquiring Access Tokens in SPAs using ADAL.JS

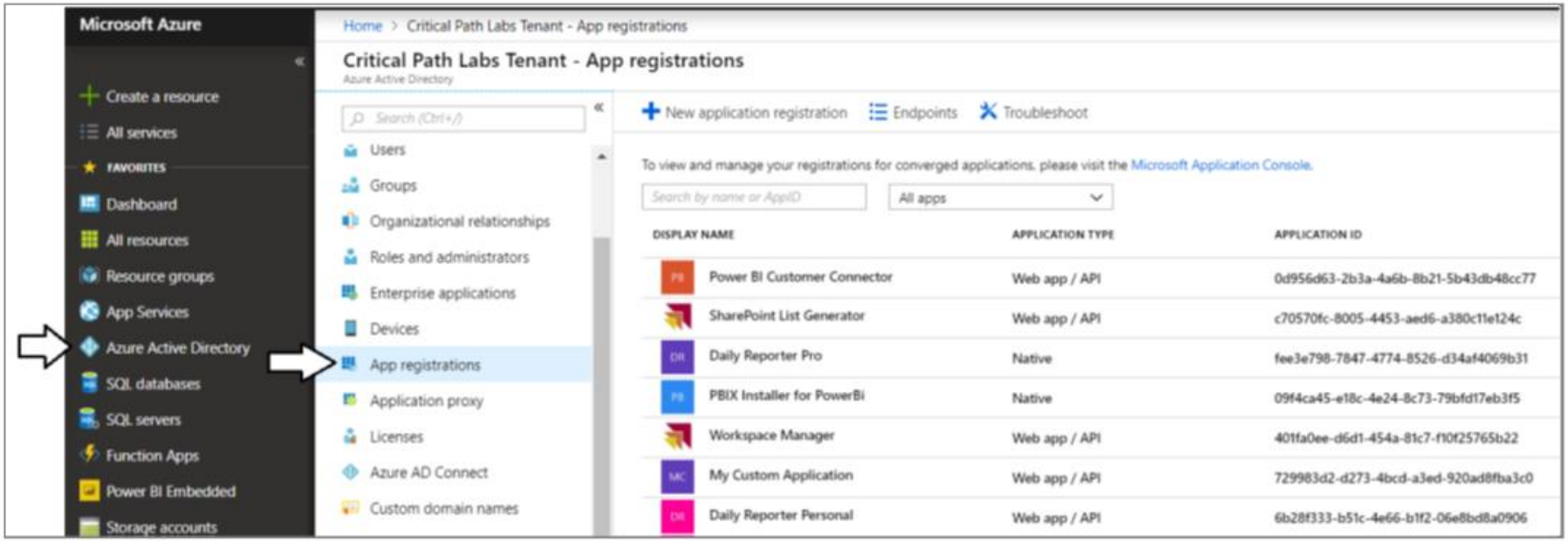
The Azure Portal

- Azure portal allows to create application
  - Azure Portal accessible at https://portal.azure.com
  - Azure subscription required to create resources (e.g. VMs)
  - No Azure subscription required to manage users or applications



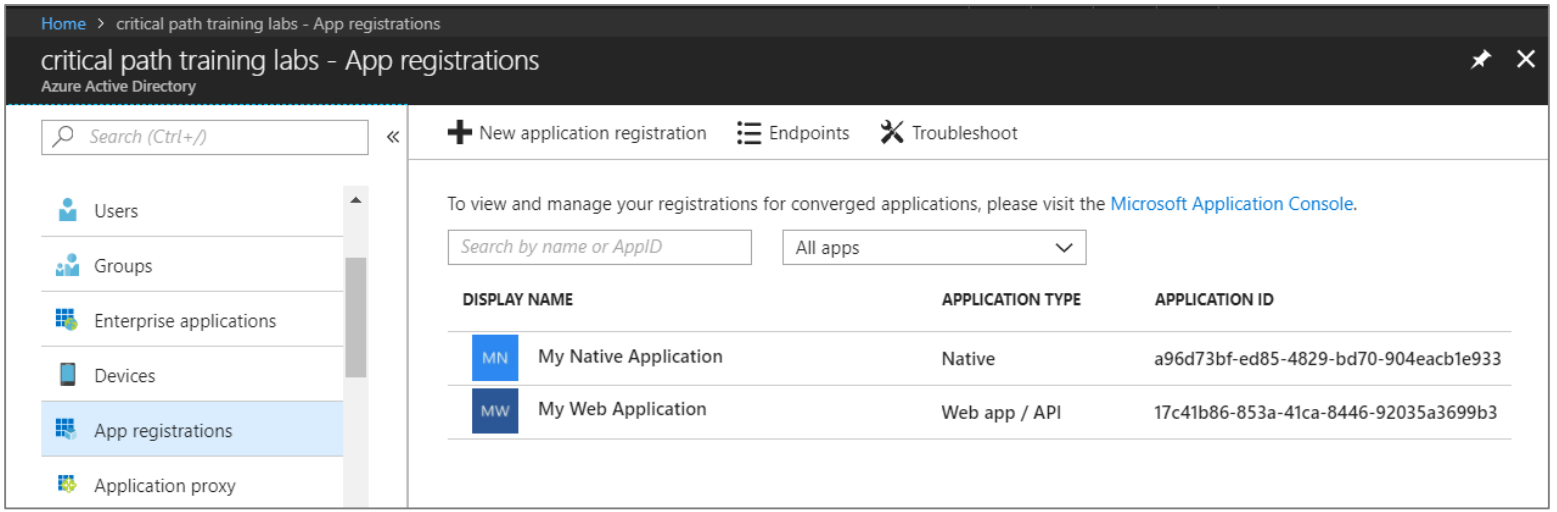
Azure Active Directory

- Azure portal access to Access Azure Active Directory
  - Provides ability to configure users, groups and application



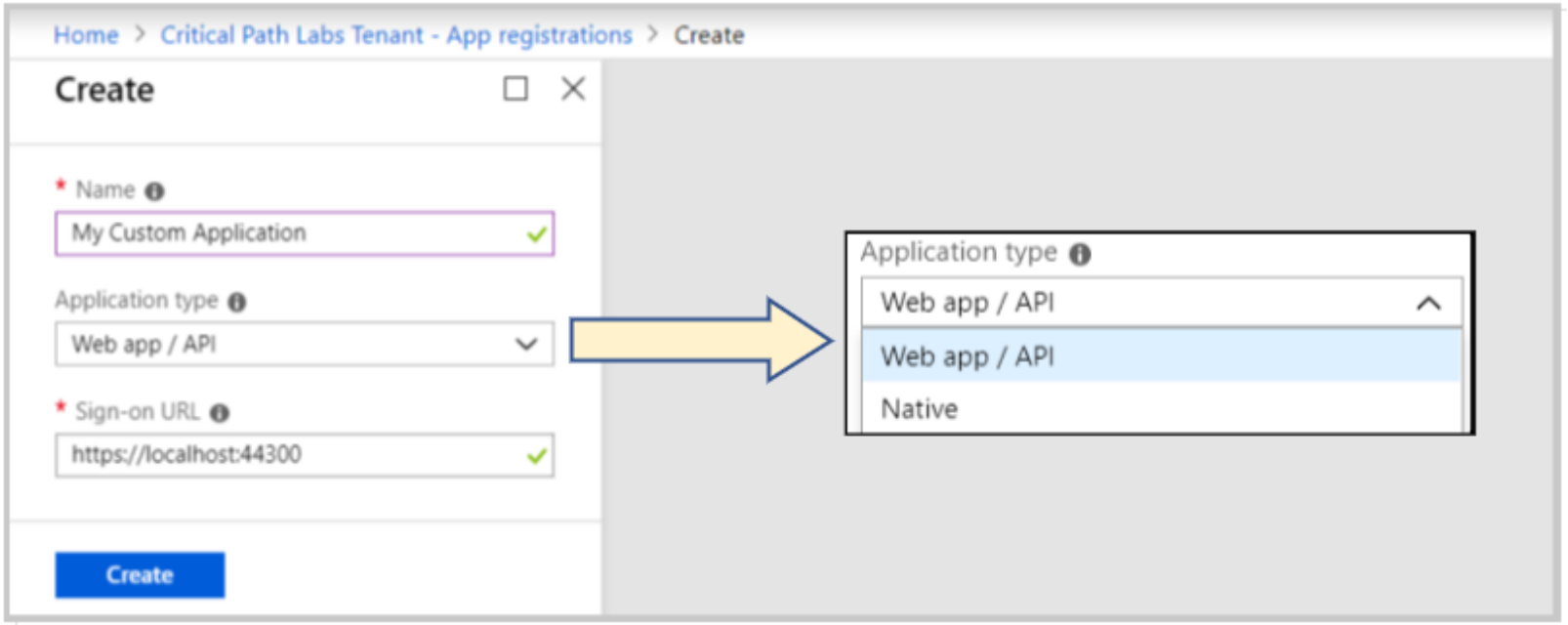
Azure AD Applications

- 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
  - Native clients
  - Web app / API client



Delegated Permissions vs Application Permissions

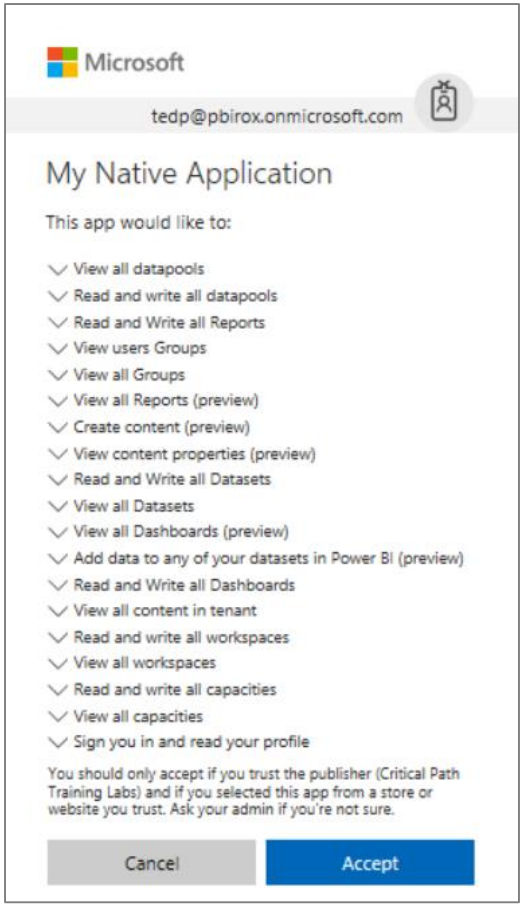
- Permissions categorized into two basic types
  - Delegated permissions are (app + user) permissions
  - Application permissions are app-only permissions (far more powerful)
  - Not all application types and APIs support application permissions
  - Power BI Service API does not yet support application permissions
- Example permissions for Office 365 SharePoint Online
  - Some delegated permissions requires administrative permissions

DELEGATED PERMISSIONS	REQUIRES ADMIN
Run search queries as a user	Yes
Read user profiles	Yes
Read user files	No
Read managed metadata	Yes
Read items in all site collections	No
Read and write user profiles	Yes
Read and write user files	No
Read and write managed metadata	Yes
Read and write items in all site collections	No
Read and write items and lists in all site collections	No
Have full control of all site collections	Yes

APPLICATION PERMISSIONS	REQUIRES ADMIN
Read user profiles	Yes
Read and write user profiles	Yes
Read and write managed metadata	Yes
Read managed metadata	Yes
Read and write items and lists in all site collections	Yes
Have full control of all site collections	Yes
Read items in all site collections	Yes
Read and write items in all site collections	Yes

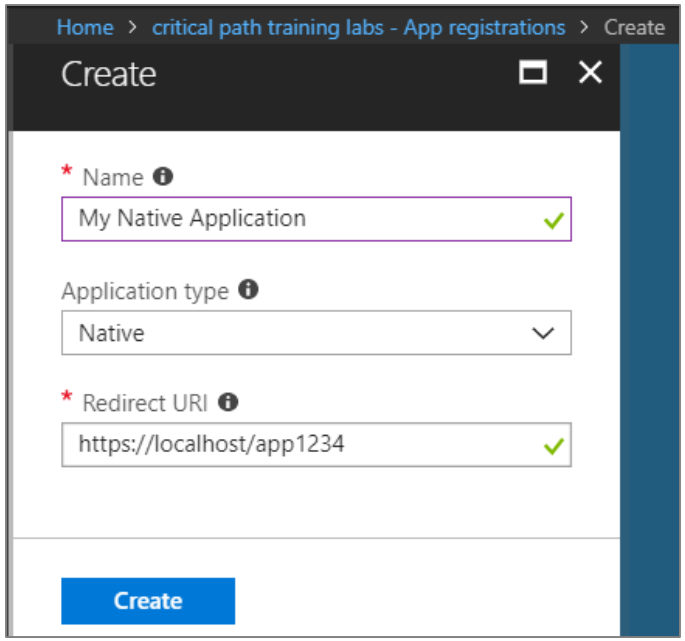
Interactive Consent for Delegated Permissions

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



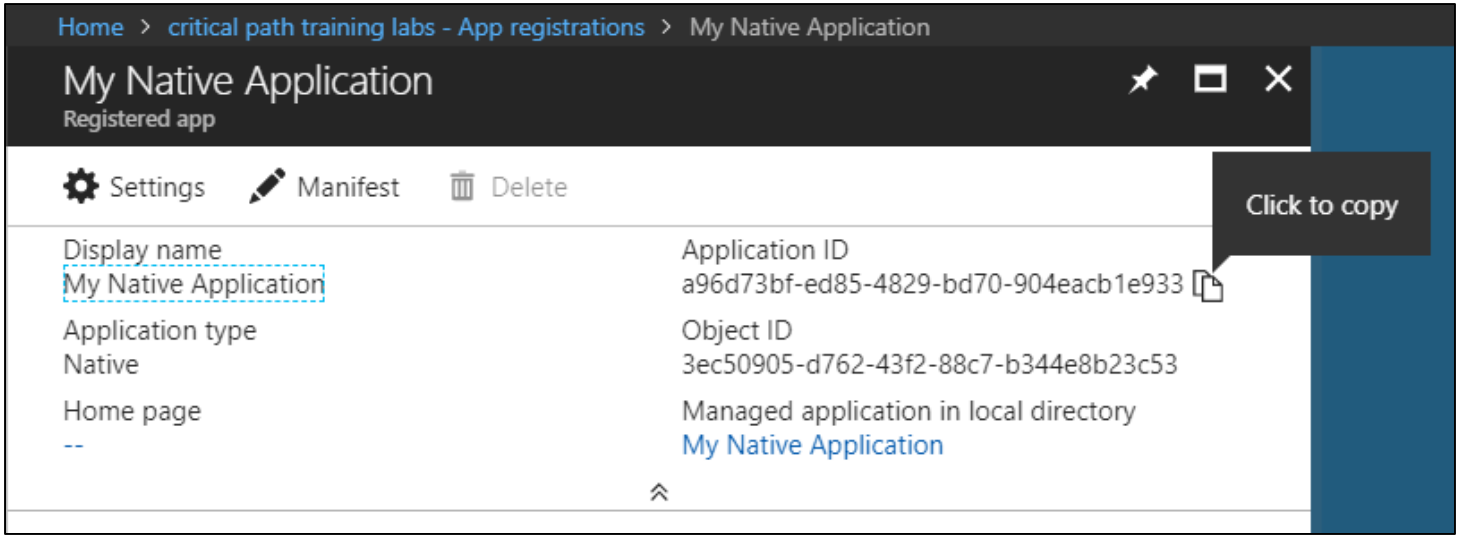
Creating a Native Application

- Power BI supports Native applications
  - Can be used for desktop applications and Console applications
  - Used for third party embedding (known as App Owns Data model)
  - Application type should be configured as Native
  - Requires Redirect URI with unique string - not an actual URL



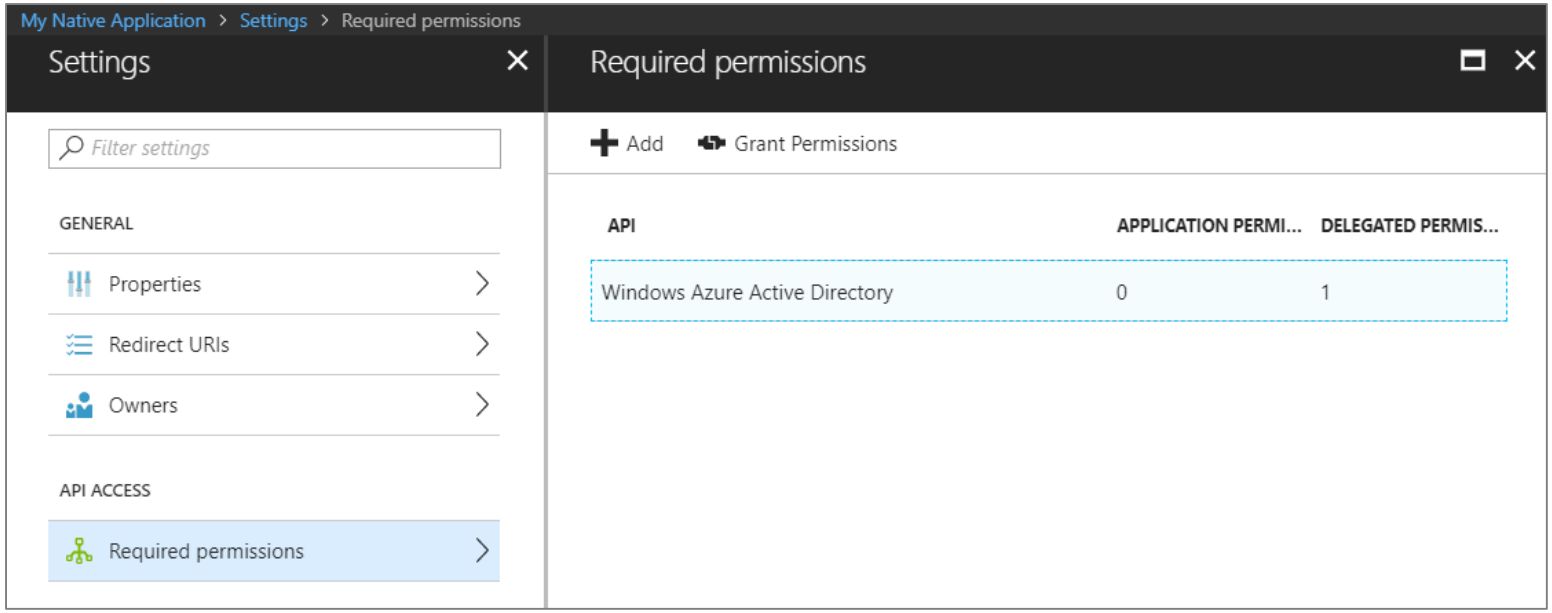
Copying the Application ID

- 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



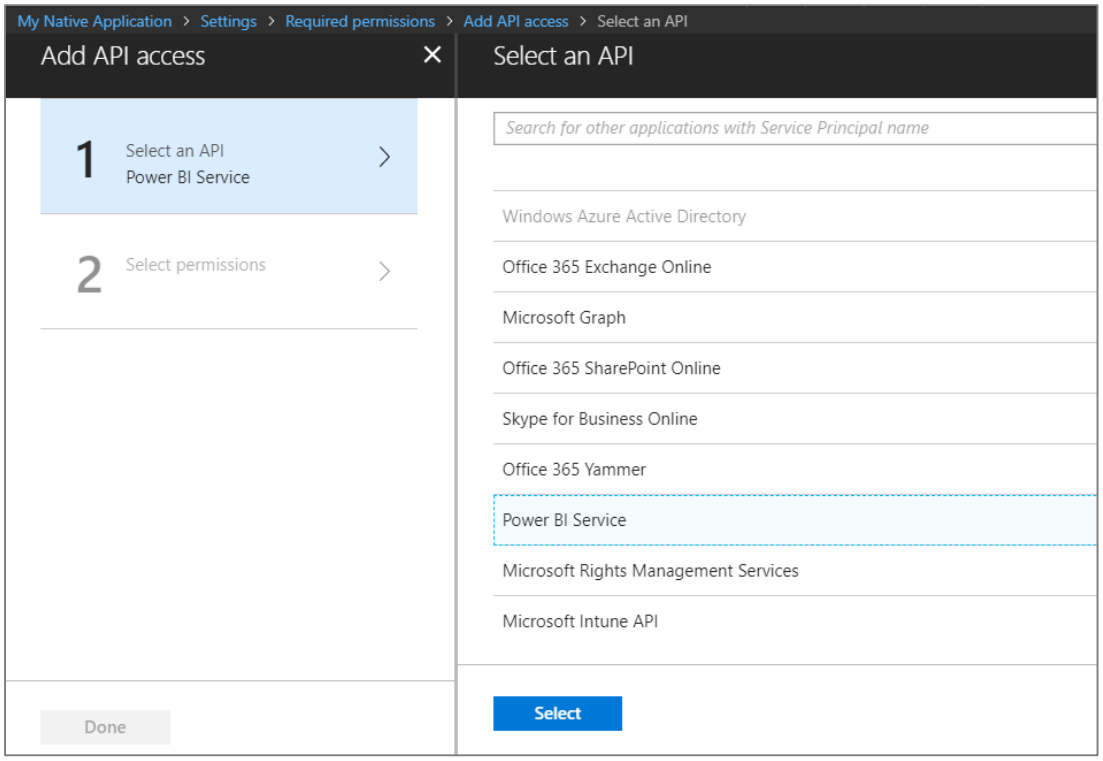
Configuring Required Permissions

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



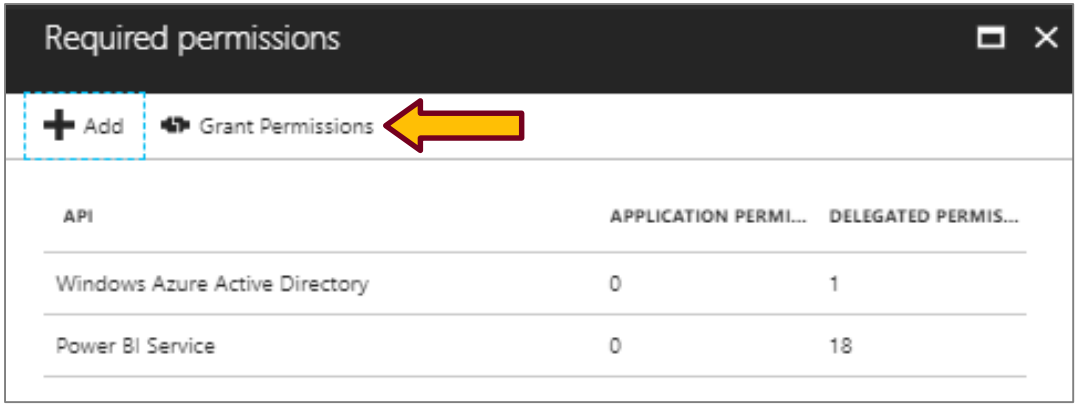
Choosing APIs

- There are lots of APIs to choose from
  - Microsoft Graph
  - Office 365 SharePoint Online
  - Power BI Service



Granting Delegated Permissions

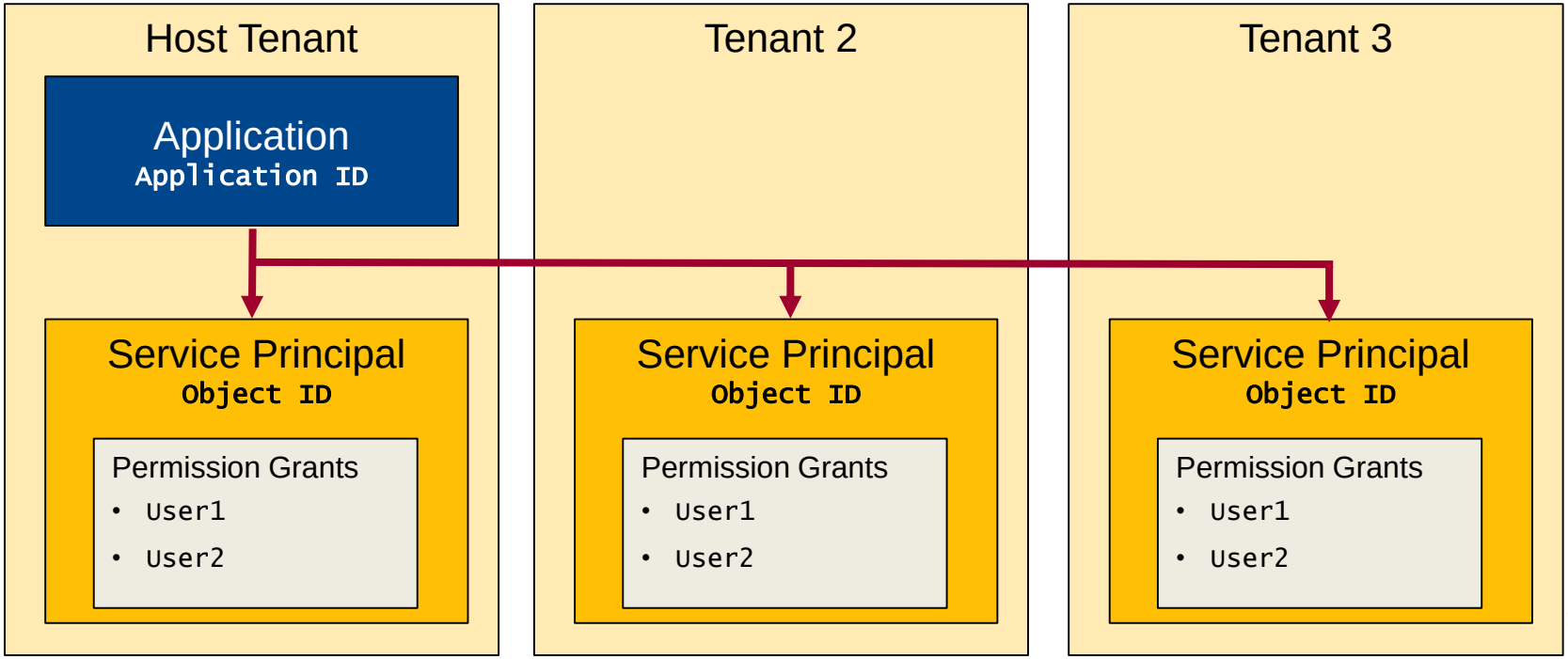
- It can be helpful to Grant Permissions in Azure portal
  - Prevents the need for interactive granting of application by user
  - Might be required when authenticating in non-interactive fashion




DEMO
Creating an AAD Application

AAD Security Principals

- Azure AD creates service principal for application
  - Service principle created once per tenant
  - Service principle used to track permission grants
  - AAD creates service principal on demand when first needed
  - You can create service principal in PowerShell script



Creating AAD Applications with PowerShell

```powershell
$appDisplayName = "My First Native App"
$replyUrl = "https://localhost/app1234"

# authenticate with your AAD user account
$authResult = Connect-AzureAD

# get info about authenticated user
$user = Get-AzureADUser -ObjectId $authResult.Account.Id

# create Azure AD Application
$aadApplication = New-AzureADApplication `
                        -DisplayName "My First Native App" `
                        -PublicClient $true `
                        -AvailableToOtherTenants $false `
                        -ReplyUrls @($replyUrl)

# get app ID for new application
$appId = $aadApplication.AppId

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

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

# 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
```


DEMO
Creating Azure AD Applications
using PowerShell Scripts

Agenda

- ✓ Power BI Service API Overview
- ✓ Authentication with Azure Active Directory
- ➢ Developing with the Power BI SDK
- Creating and Managing Workspaces

Programming with PowerBIClient

- PowerBiClient is top-level object in API

```
private static PowerBIClient GetPowerBiClient() {
  string accessToken = GetAccessTokenAsync().Result;
  TokenCredentials tokenCredentials = new TokenCredentials(accessToken, "Bearer");
  return new PowerBIClient(new Uri(urlPowerBiRestApiRoot), tokenCredentials);
}
```

```
public static async Task<IList<Group>> GetWorkspacesAsync() {
  PowerBIClient pbiClient = GetPowerBiClient();
  return (await pbiClient.Groups.GetGroupsAsync()).Value;
}
```

Uploading a PBIX File

```
public static async Task UploadPBIX(string WorkspaceId, string pbixName, string importName, bool updateSqlCredentials = false) {

  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;
}
```

Getting and Refreshing Datasets

```
public static async Task<DatasetViewModel> GetDatasetAsync(string WorkspaceId, string DatasetId) {

  PowerBIClient pbiClient = GetPowerBiClient();
  Dataset dataset = (await pbiClient.Datasets.GetDatasetByIdInGroupAsync(WorkspaceId, DatasetId));
  IList<Datasource> datasources = (await pbiClient.Datasets.GetDatasourcesInGroupAsync(WorkspaceId, DatasetId)).Value;
  IList<Refresh> refreshHistory = null;

  if (dataset.IsRefreshable == true) {
    refreshHistory = (await pbiClient.Datasets.GetRefreshHistoryInGroupAsync(WorkspaceId, DatasetId)).Value;
  }

  DatasetViewModel viewModel = new DatasetViewModel {
    WorkspaceId=WorkspaceId,
    Id = dataset.Id,
    Name = dataset.Name,
    Dataset = dataset,
    Datasources = datasources,
    RefreshHistroy = refreshHistory
  };

  return viewModel;
}
```

```
public static async Task RefreshDatasetAsync(string WorkspaceId, string DatasetId) {
  PowerBIClient pbiClient = GetPowerBiClient();
  await pbiClient.Datasets.RefreshDatasetInGroupAsync(WorkspaceId, DatasetId);
  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;
}
```

Agenda

- ✓ Power BI Service API Overview
- ✓ Authentication with Azure Active Directory
- ✓ Calling the API with Direct REST Calls
- ✓ Developing with the Power BI SDK
- ➢ Creating and Managing Workspaces

App Workspace Management

```
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;

}
```

Summary

- ✓ Power BI Service API Overview
- ✓ Authentication with Azure Active Directory
- ✓ Developing with the Power BI SDK
- ✓ Creating and Managing Workspaces