

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



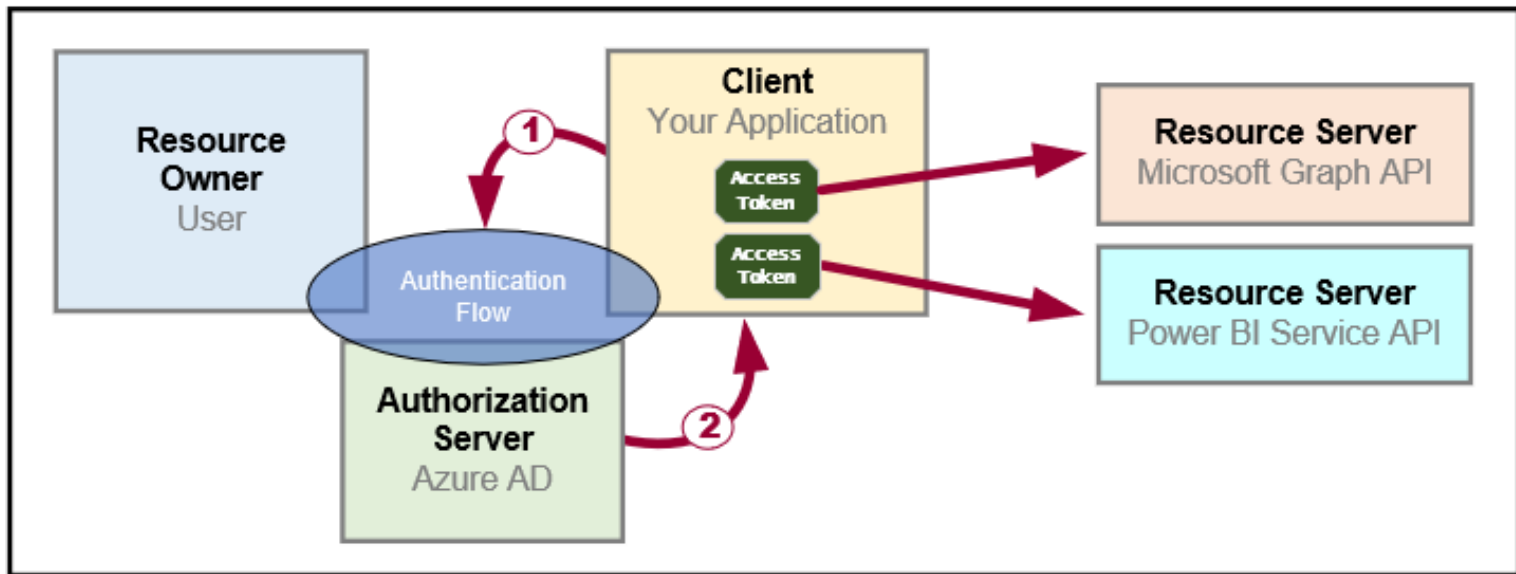
Agenda

- ✓ 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",

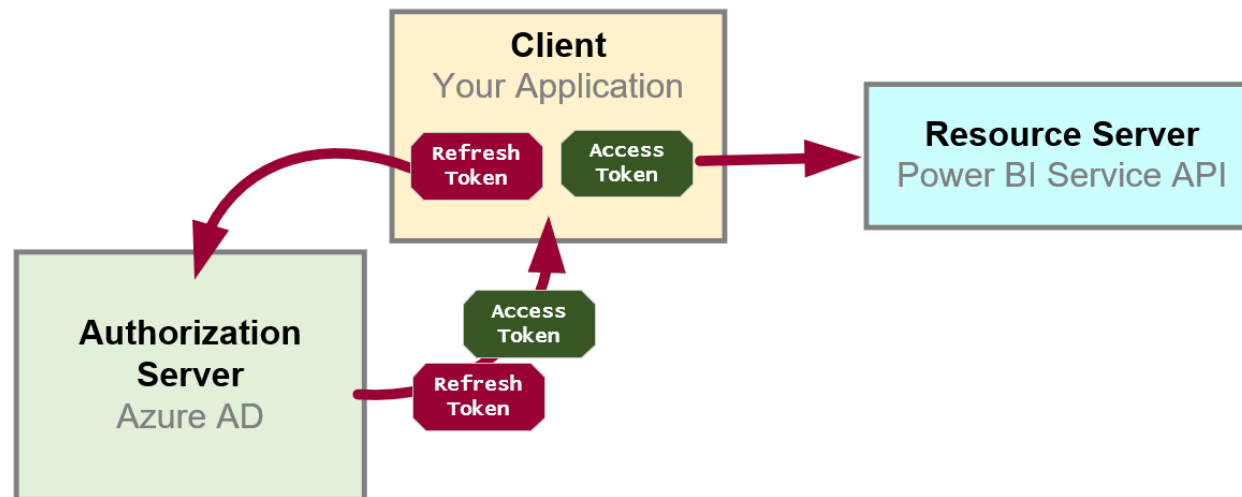
  "ipaddr": "47.200.98.132",

  "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

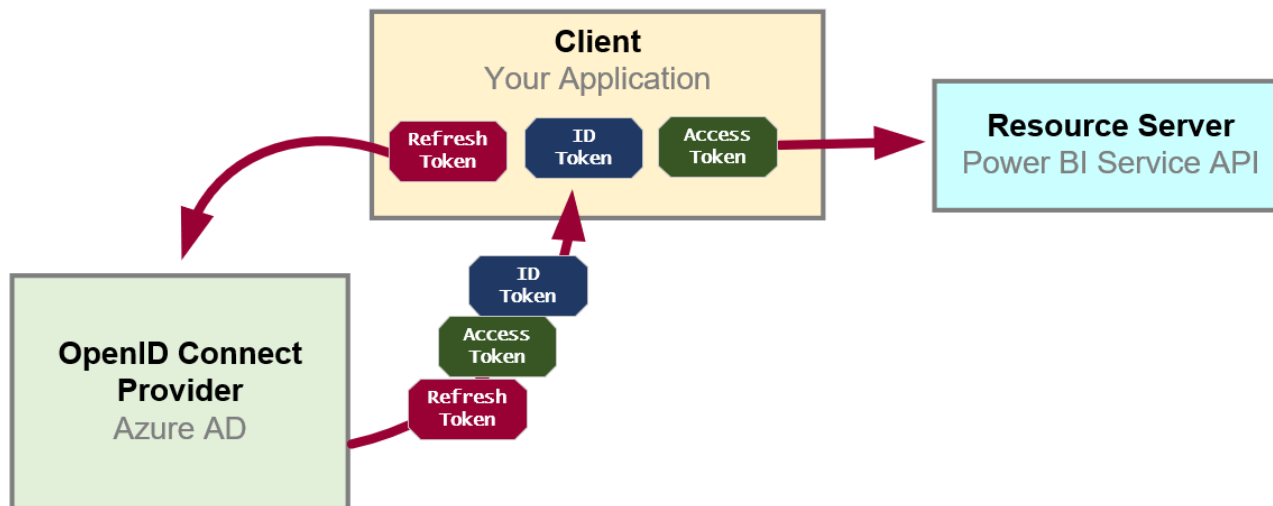
Registered Applications

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



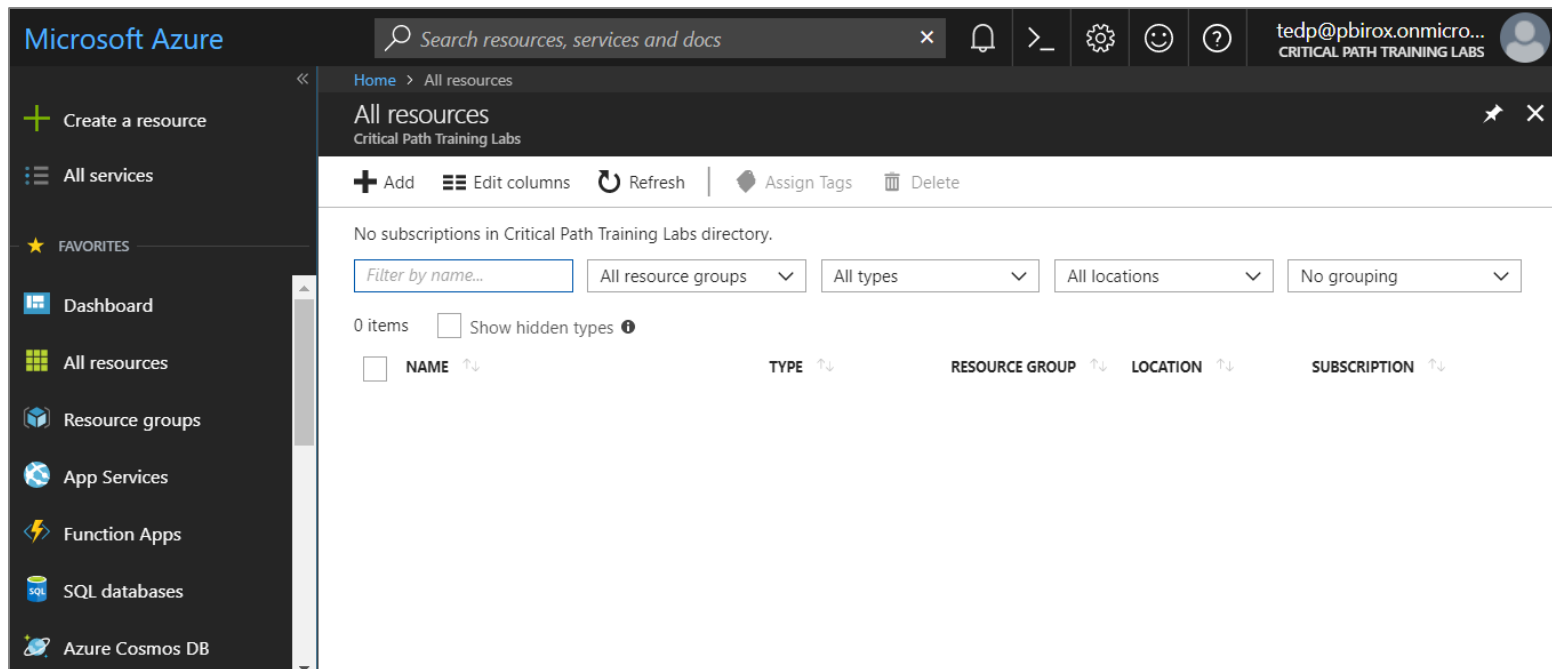
Agenda

- ✓ 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



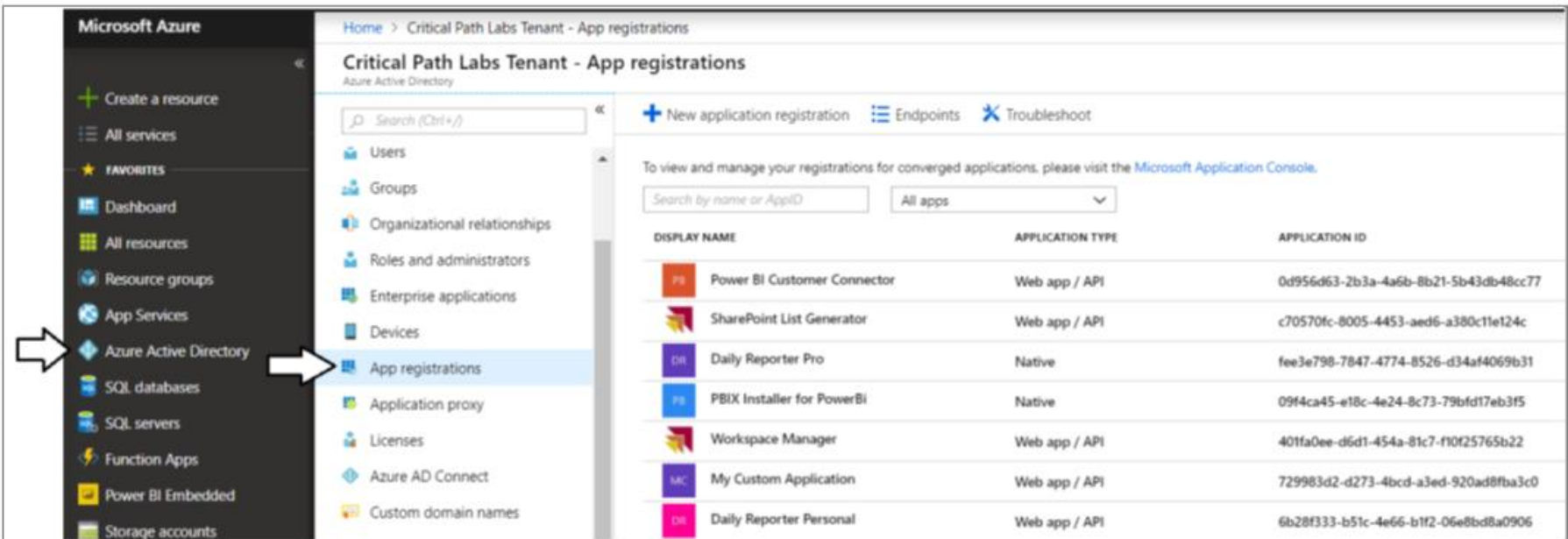
The Azure Portal

- 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



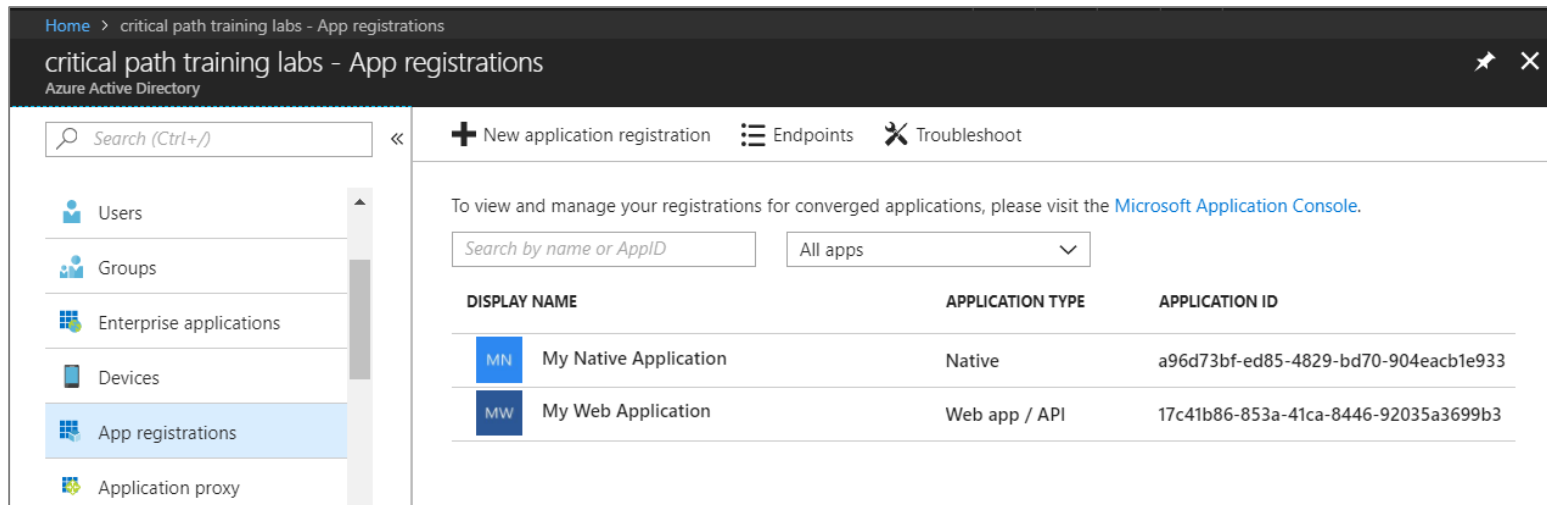
The screenshot displays the Microsoft Azure portal interface. On the left, the navigation pane shows 'Azure Active Directory' and 'App registrations' highlighted. The main content area is titled 'Critical Path Labs Tenant - App registrations' and shows a list of registered applications. The list includes columns for 'DISPLAY NAME', 'APPLICATION TYPE', and 'APPLICATION ID'.

DISPLAY NAME	APPLICATION TYPE	APPLICATION ID
Power BI Customer Connector	Web app / API	0d956d63-2b3a-4a6b-8b21-5b43db48cc77
SharePoint List Generator	Web app / API	c70570fc-8005-4453-aed6-a380c11e124c
Daily Reporter Pro	Native	fee3e798-7847-4774-8526-d34af4069b31
PBIX Installer for PowerBI	Native	09f4ca45-e18c-4e24-8c73-79bfd17eb3f5
Workspace Manager	Web app / API	401fa0ee-d6d1-454a-81c7-f10f25765b22
My Custom Application	Web app / API	729983d2-d273-4bcd-a3ed-920ad8fba3c0
Daily Reporter Personal	Web app / API	6b28f333-b51c-4e66-b1f2-06e8bd8a0906





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



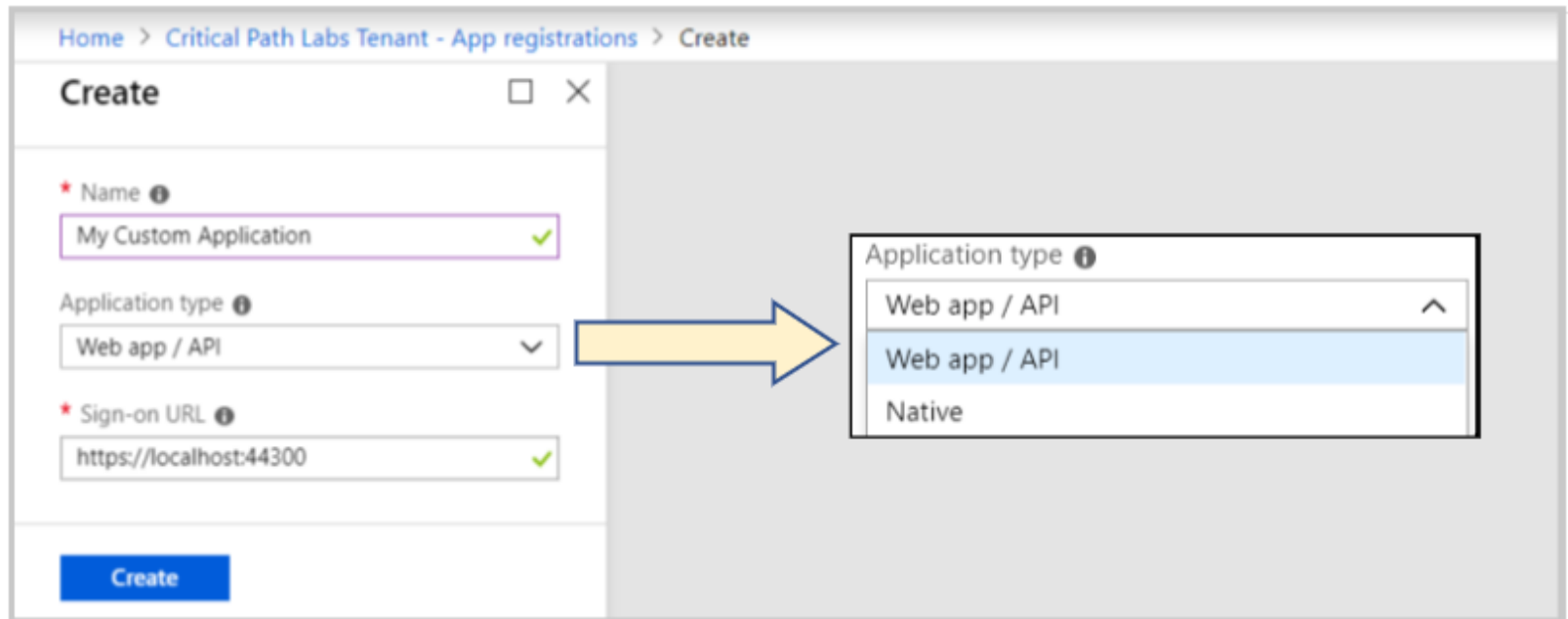
The screenshot shows the Azure Active Directory (Azure AD) 'App registrations' page. The left sidebar contains navigation links: Users, Groups, Enterprise applications, Devices, App registrations (selected), and Application proxy. The main content area has a search bar and a list of application registrations. The table below shows the details of two applications: 'My Native Application' and 'My Web Application'.

DISPLAY NAME	APPLICATION TYPE	APPLICATION ID
 My Native Application	Native	a96d73bf-ed85-4829-bd70-904each1e933
 My Web Application	Web app / API	17c41b86-853a-41ca-8446-92035a3699b3



Application Types

- Azure AD Application Types
 - Native clients
 - Web app / API client



The screenshot shows the 'Create' page for registering an application in the Azure AD portal. The breadcrumb navigation at the top reads 'Home > Critical Path Labs Tenant - App registrations > Create'. The form has three main input fields: 'Name' (containing 'My Custom Application'), 'Application type' (set to 'Web app / API'), and 'Sign-on URL' (containing 'https://localhost:44300'). A blue 'Create' button is at the bottom left. A yellow arrow points from the 'Application type' dropdown to a detailed view of the dropdown menu on the right. This menu shows three options: 'Web app / API' (selected and highlighted in blue), 'Web app / API', and 'Native'.

Home > Critical Path Labs Tenant - App registrations > Create

Create

* Name ⓘ
My Custom Application ✓

Application type ⓘ
Web app / API ▼

* Sign-on URL ⓘ
https://localhost:44300 ✓

Create

Application type ⓘ

- Web app / API ^
- Web app / API
- Native



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

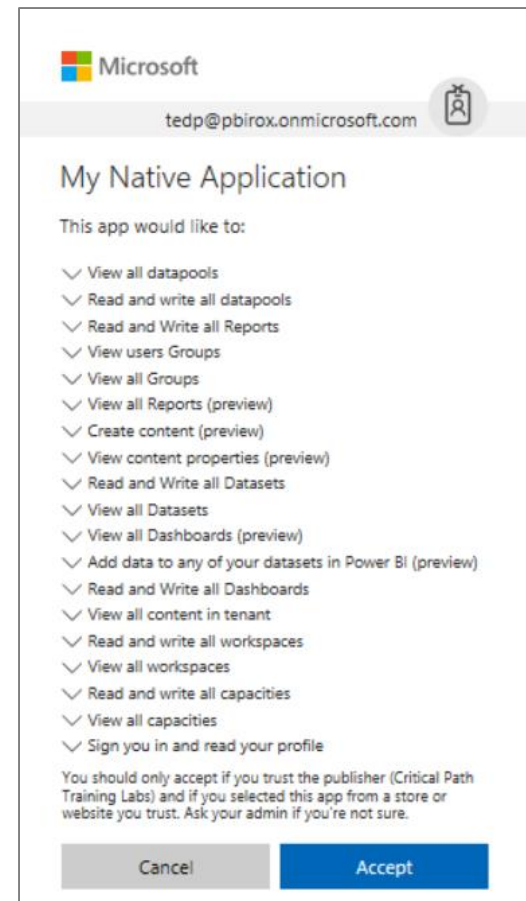
<input type="checkbox"/> DELEGATED PERMISSIONS	REQUIRES ADMIN
Run search queries as a user	✓ Yes
Read user profiles	✓ Yes
<input checked="" type="checkbox"/> Read user files	✗ No
Read managed metadata	✓ Yes
<input checked="" type="checkbox"/> Read items in all site collections	✗ No
Read and write user profiles	✓ Yes
<input checked="" type="checkbox"/> Read and write user files	✗ No
Read and write managed metadata	✓ Yes
<input checked="" type="checkbox"/> Read and write items in all site collections	✗ No
<input checked="" type="checkbox"/> 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



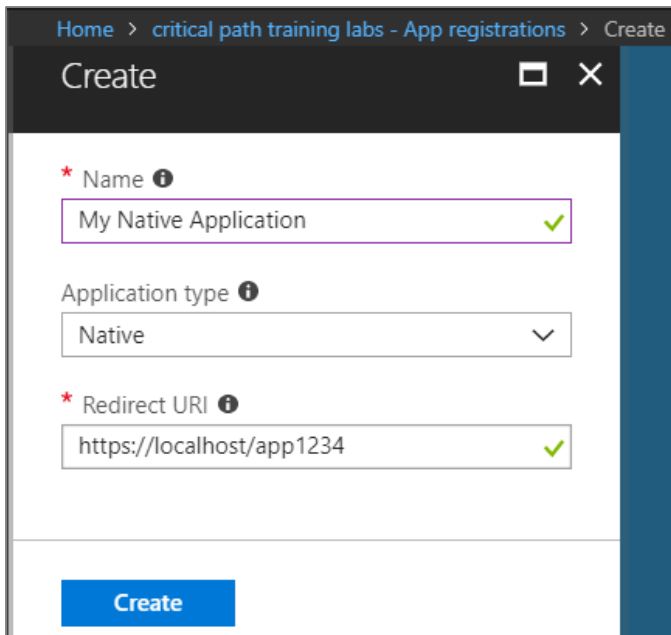
Interactive Consent for Delegated Permissions

- 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



The screenshot shows a 'Create' dialog box with a dark header bar containing the text 'Create' and window control icons. The main area is white and contains three required fields, each marked with a red asterisk and an information icon:

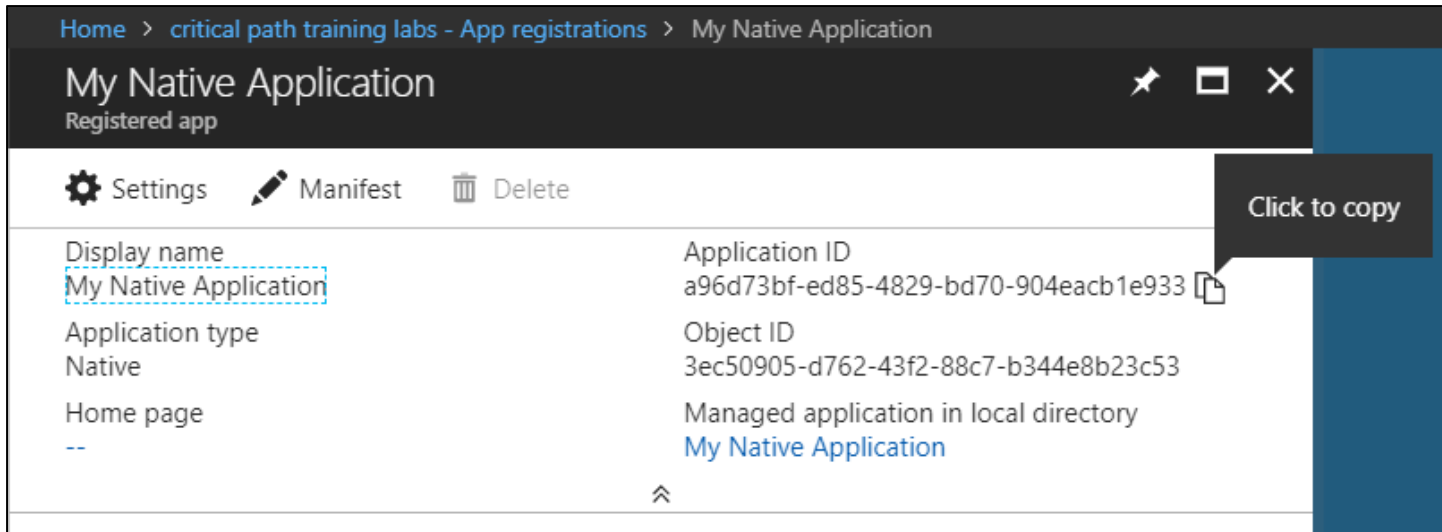
- Name:** The text 'My Native Application' is entered in the input field, followed by a green checkmark.
- Application type:** A dropdown menu is open, showing 'Native' as the selected option with a downward arrow.
- Redirect URI:** The text 'https://localhost/app1234' is entered in the input field, followed by a green checkmark.

A blue 'Create' button is located at the bottom left of the dialog box.



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



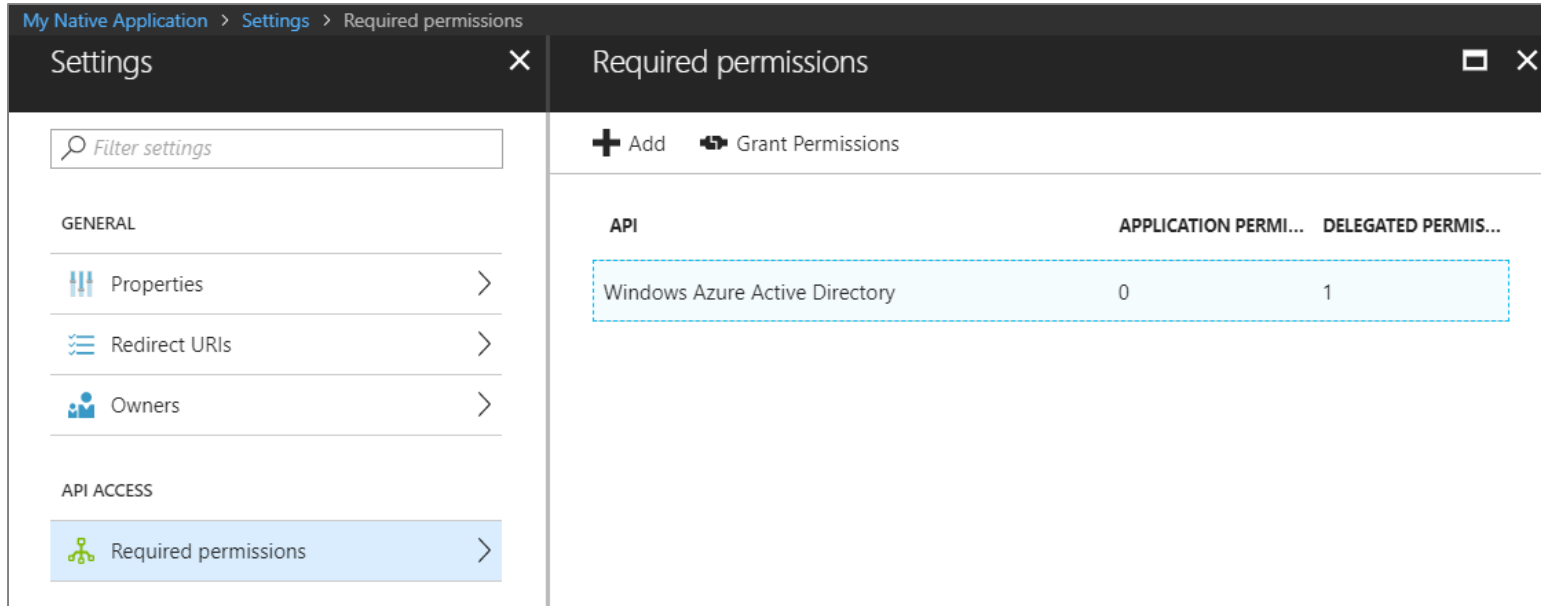
The screenshot shows the Azure portal interface for a registered application named 'My Native Application'. The breadcrumb navigation at the top reads 'Home > critical path training labs - App registrations > My Native Application'. The application title 'My Native Application' is displayed with the subtitle 'Registered app'. Below the title are three action buttons: 'Settings' (gear icon), 'Manifest' (pencil icon), and 'Delete' (trash icon). The main content area is divided into two columns. The left column lists properties: 'Display name' (My Native Application), 'Application type' (Native), and 'Home page' (indicated by '--'). The right column lists: 'Application ID' (a96d73bf-ed85-4829-bd70-904eacb1e933), 'Object ID' (3ec50905-d762-43f2-88c7-b344e8b23c53), and 'Managed application in local directory' (My Native Application). A dashed blue box highlights the 'Application ID' value, and a tooltip with the text 'Click to copy' is positioned over the copy icon next to it.

Property	Value
Display name	My Native Application
Application type	Native
Home page	--
Application ID	a96d73bf-ed85-4829-bd70-904eacb1e933
Object ID	3ec50905-d762-43f2-88c7-b344e8b23c53
Managed application in local directory	My Native Application



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



The screenshot shows the 'Required permissions' settings page for a native application. The left sidebar contains a 'Settings' menu with a search bar and two sections: 'GENERAL' (Properties, Redirect URIs, Owners) and 'API ACCESS' (Required permissions). The main panel is titled 'Required permissions' and includes '+ Add' and 'Grant Permissions' buttons. Below these is a table with columns 'API', 'APPLICATION PERMI...', and 'DELEGATED PERMIS...'. A single row is visible, representing 'Windows Azure Active Directory' with values '0' and '1'.

API	APPLICATION PERMI...	DELEGATED PERMIS...
Windows Azure Active Directory	0	1



Choosing APIs

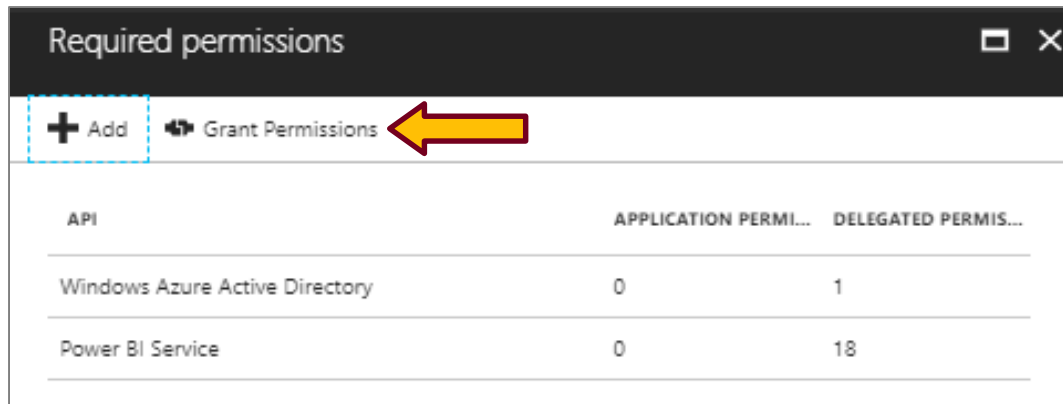
- There are lots of APIs to choose from
 - Microsoft Graph
 - Office 365 SharePoint Online
 - Power BI Service

The screenshot shows the 'Add API access' dialog in the Azure portal. The breadcrumb trail at the top reads: 'My Native Application > Settings > Required permissions > Add API access > Select an API'. The dialog is split into two panes. The left pane, titled 'Add API access', contains a numbered list of steps: '1 Select an API' (with 'Power BI Service' listed below it) and '2 Select permissions'. The right pane, titled 'Select an API', features a search bar with the placeholder text 'Search for other applications with Service Principal name'. Below the search bar is a list of available APIs: 'Windows Azure Active Directory', 'Office 365 Exchange Online', 'Microsoft Graph', 'Office 365 SharePoint Online', 'Skype for Business Online', 'Office 365 Yammer', 'Power BI Service' (which is highlighted with a dashed blue border), 'Microsoft Rights Management Services', and 'Microsoft Intune API'. At the bottom of the left pane is a 'Done' button, and at the bottom of the right pane is a 'Select' button.



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



Required permissions		
<div>+ Add Grant Permissions</div>		
API	APPLICATION PERMI...	DELEGATED PERMIS...
Windows Azure Active Directory	0	1
Power BI Service	0	18



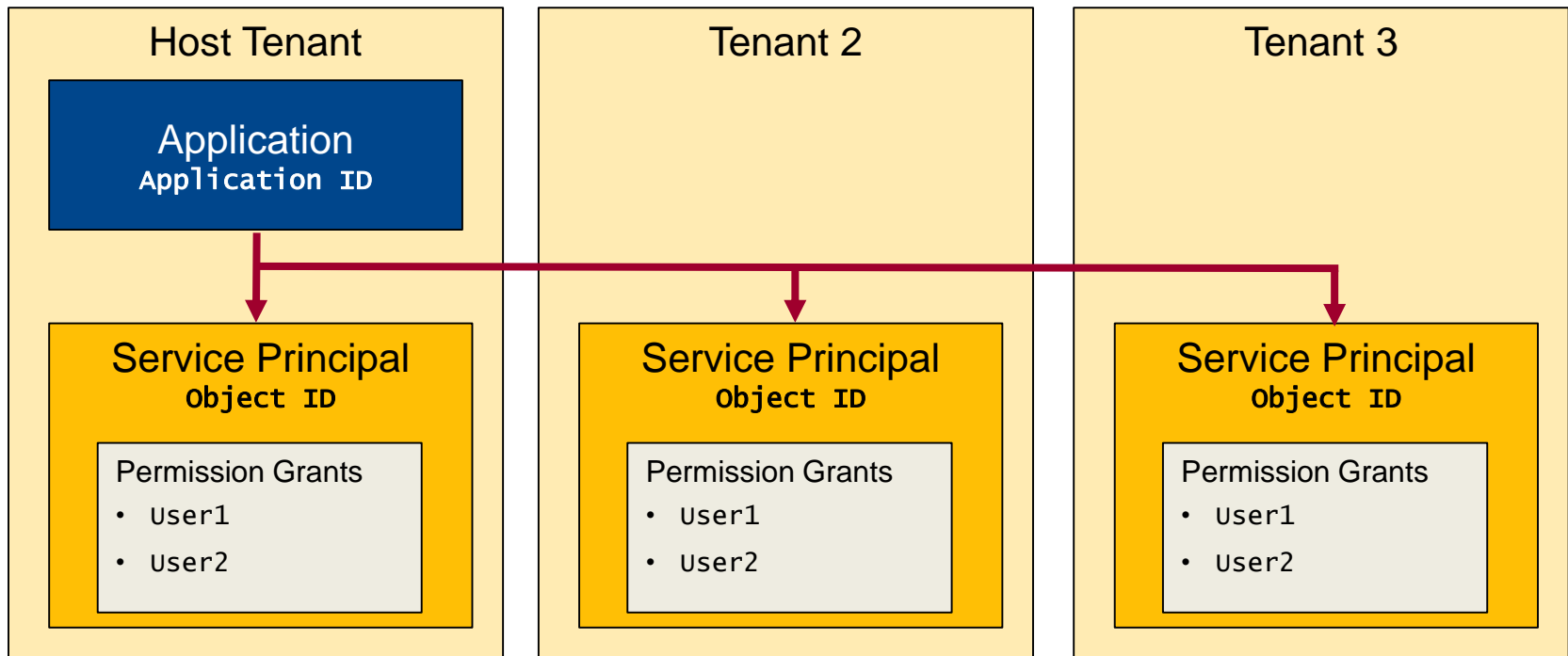


DEMO

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

```
$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 permissions 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
```



The background of the slide is a close-up, low-angle shot of a server rack. The server units are illuminated with bright blue light, creating a strong sense of depth and perspective. The lights are arranged in vertical columns, and the overall color palette is dominated by deep blues and bright cyan highlights.

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 = "pbimasteruser@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

