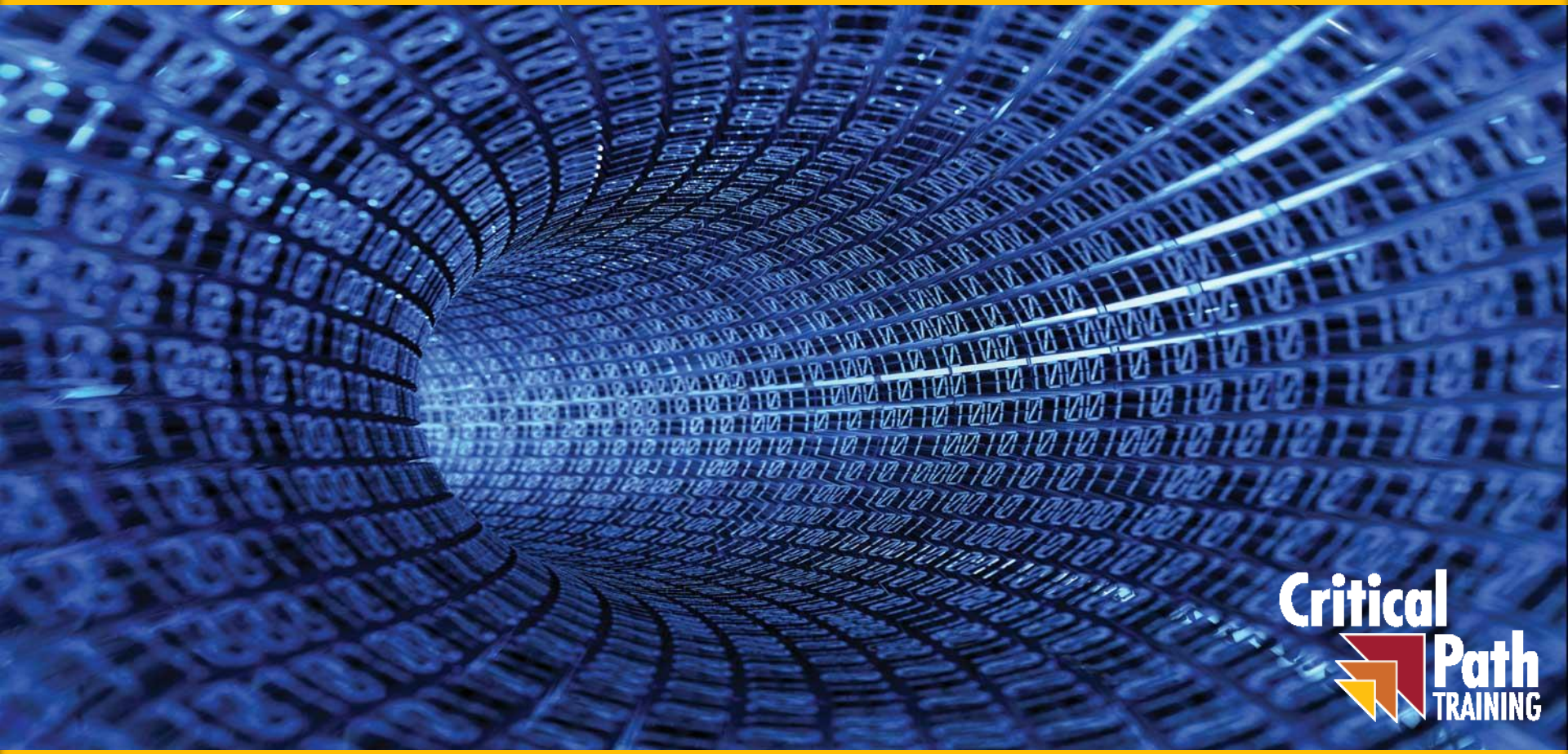
Developing with the Power BI Service API



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

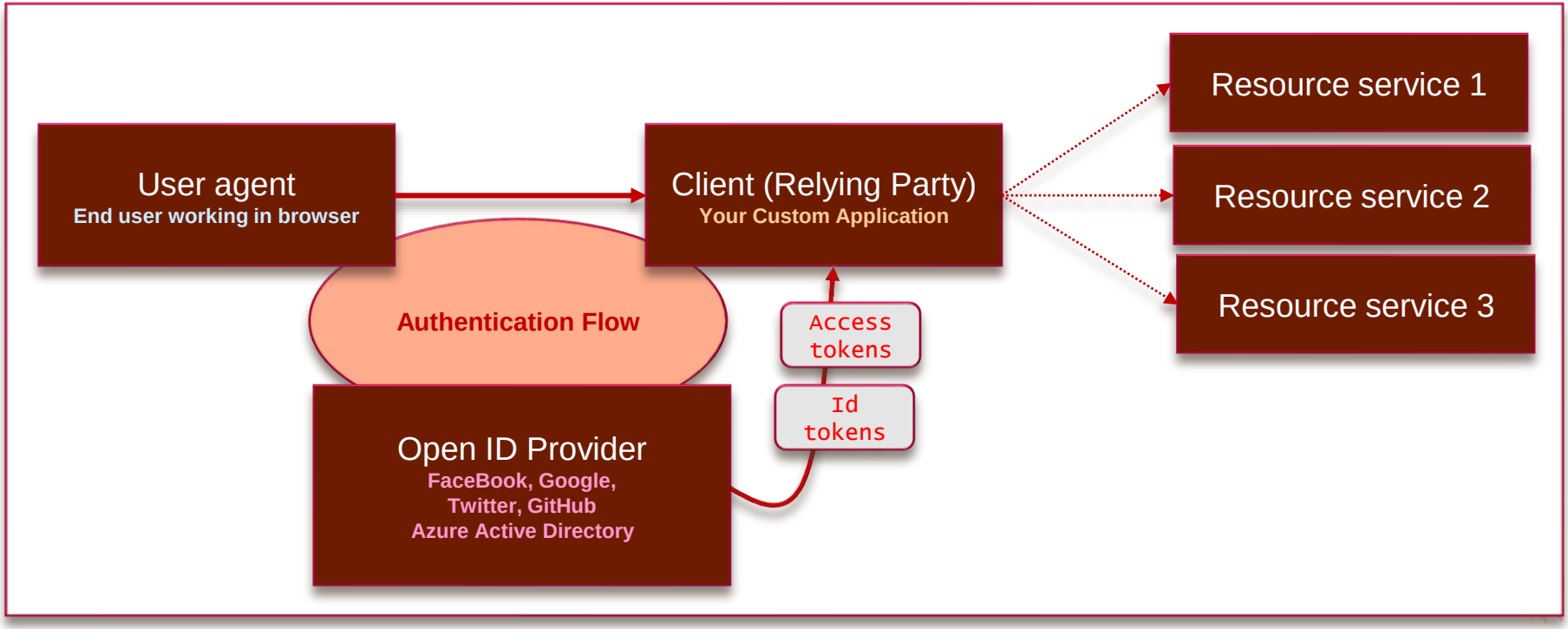
- Azure subscription not required!
  - Azure portal used to create Azure AD application
  - Azure subscription helpful to create Azure resources

Agenda

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

OAuth2 and Open ID Connect

- Power BI Service requires authentication with OAuth2
  - Your application must implement an authentication flow
  - Authentication flow used to acquire an access token
  - Access token required whenever calling Power BI Service API



Client Application Registration

- Application 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

Authentication Flows

- Authorization Code Grant Flow *(confidential client)*
  - Client first obtains authorization code then access token
  - Server-side application code never sees user's password
- Implicit Grant Flow *(public client)*
  - Used in SPAs built with JavaScript and AngularJS
  - Application obtains access token w/o acquiring authorization code
- User Credentials Flow *(public client)*
  - Used in Native clients to obtain access code
  - Requires passing user name and password
- Client Credentials Grant Flow *(confidential client)*
  - Authentication based on SSL certificate with public-private key pair
  - Used to obtain access token when using app-only permissions

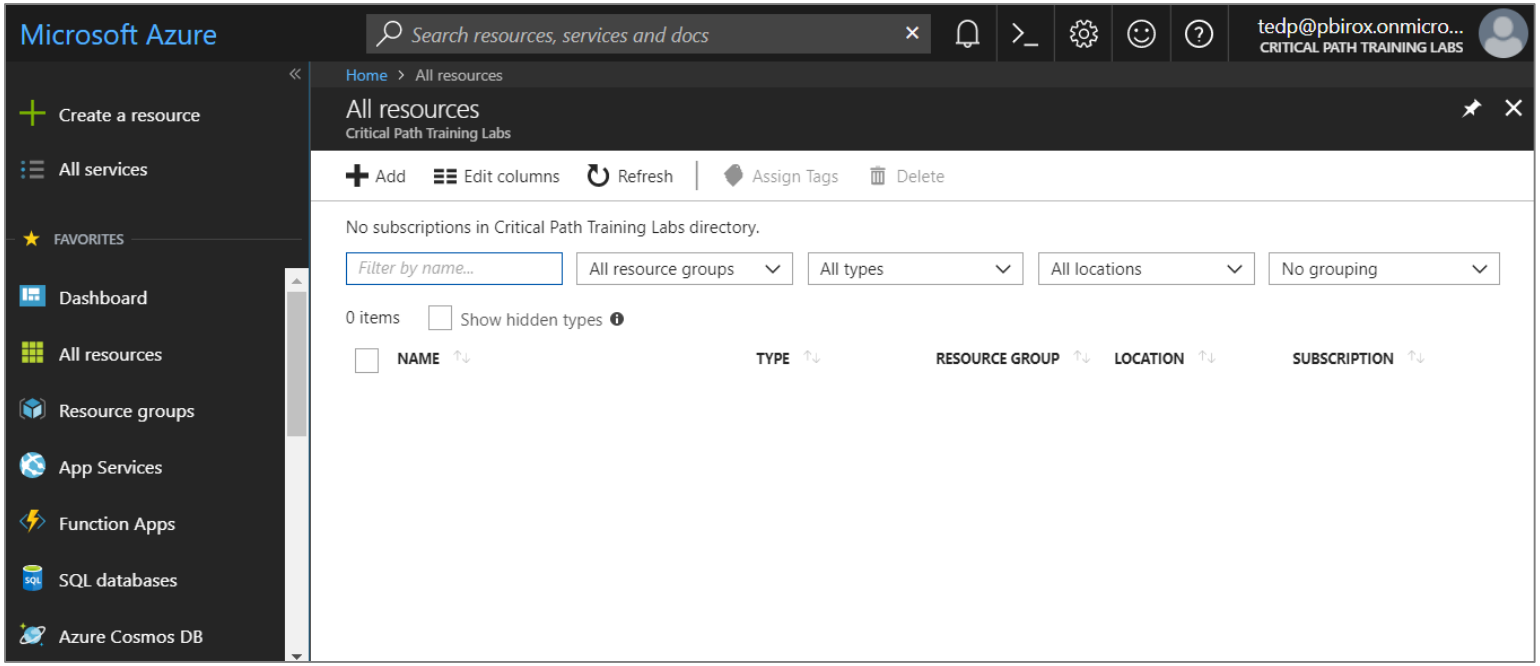
Azure Active Directory (AAD)

- AAD plays role of an OpenID Connect Provider
  - Creates access tokens based on OAuth 2.0
  - Creates id tokens based on OpenID Connect 1.0

- AAD provides authentication & authorization for…
  - Office 365, Dynamics 365 and SharePoint Online
  - Power BI Service API and Microsoft Graph API
  - Custom Web Applications and Web Services

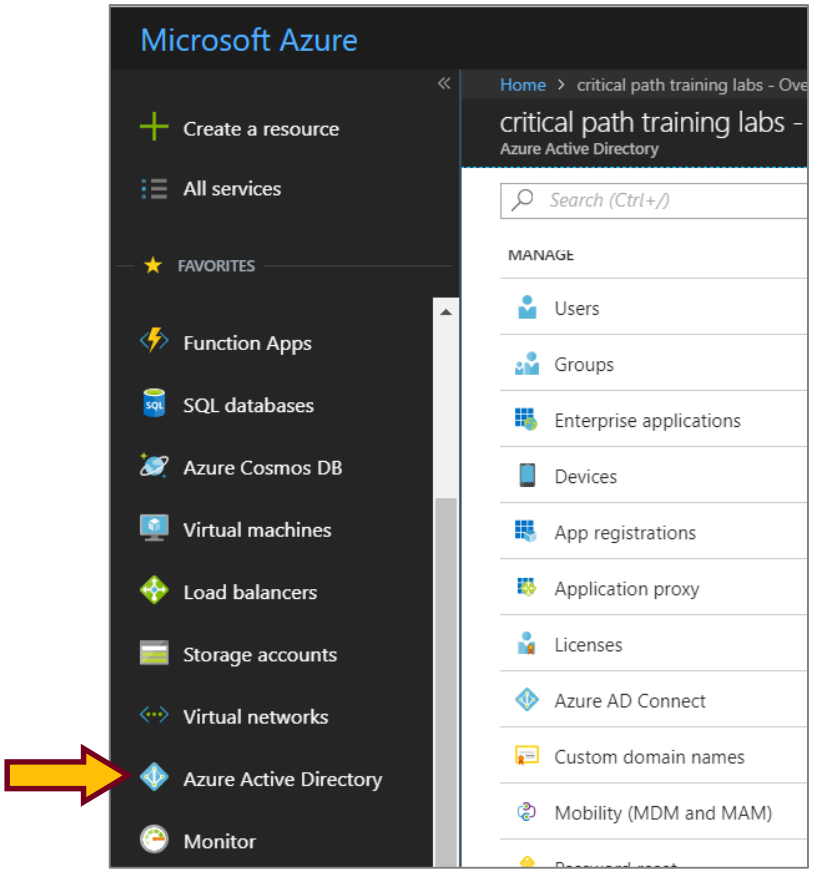
The Azure Portal

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



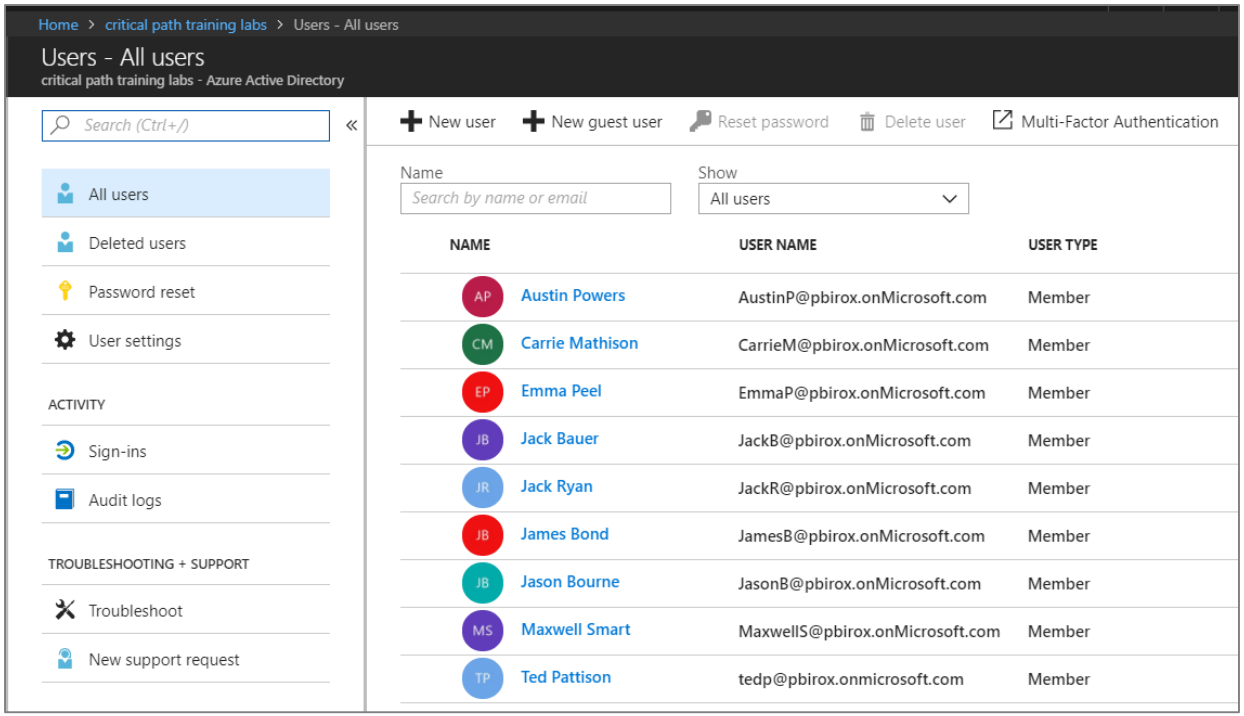
Azure Active Directory

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



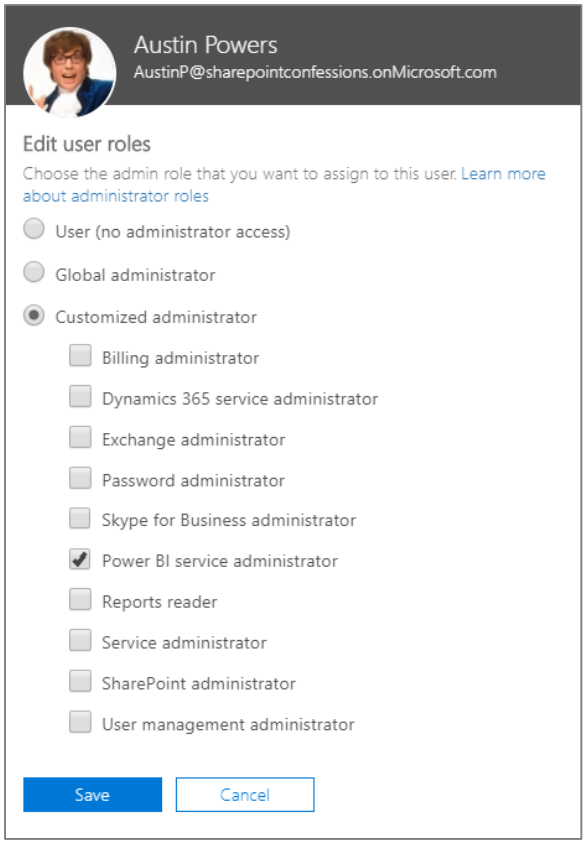
Managing Users and Groups

- You can manage users and assign licenses
- You can create groups and populate members



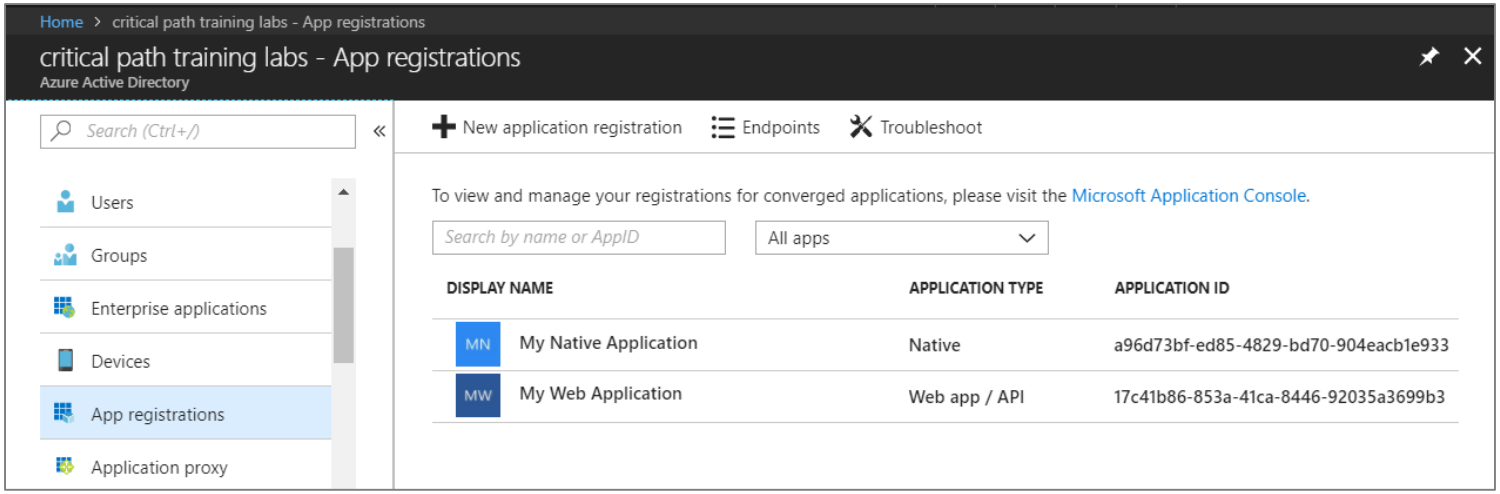
Power BI Administrator

- Azure AD provides role of Power BI Service administrator
  - Provides user with tenant-level administrative permissions



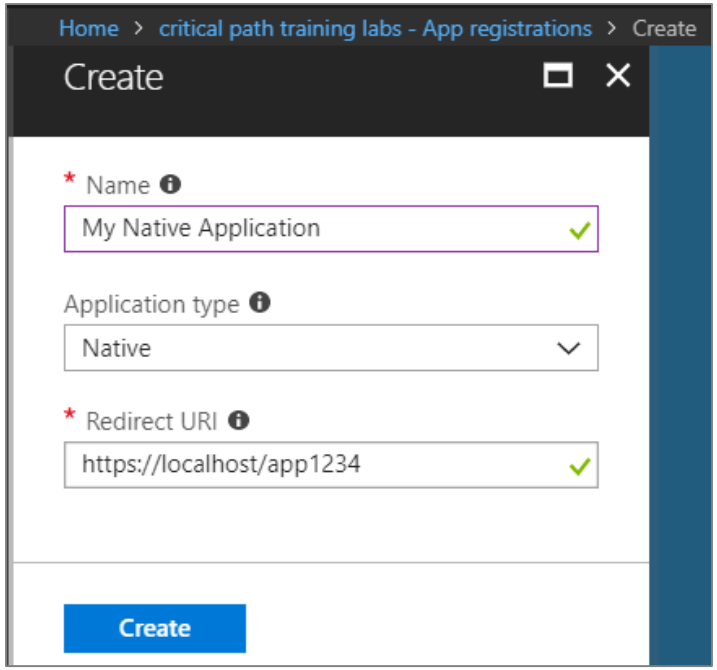
Azure AD Applications

- 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**



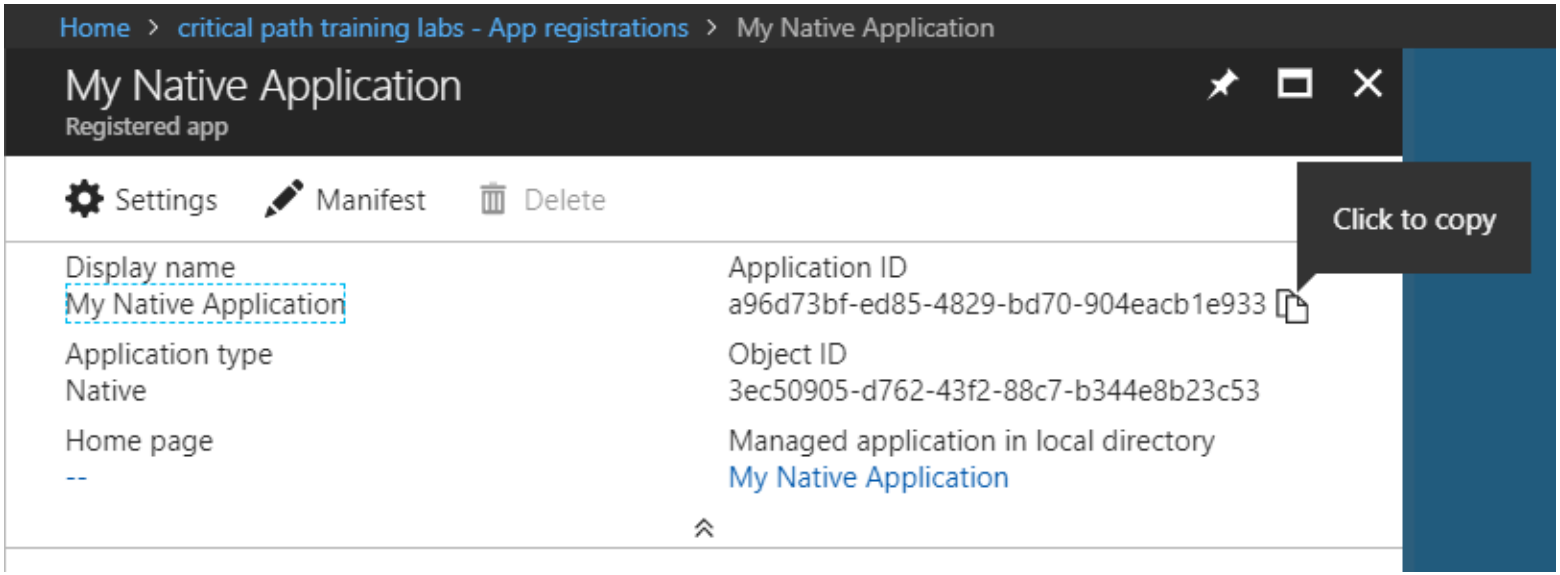
Creating a Native Application

- 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



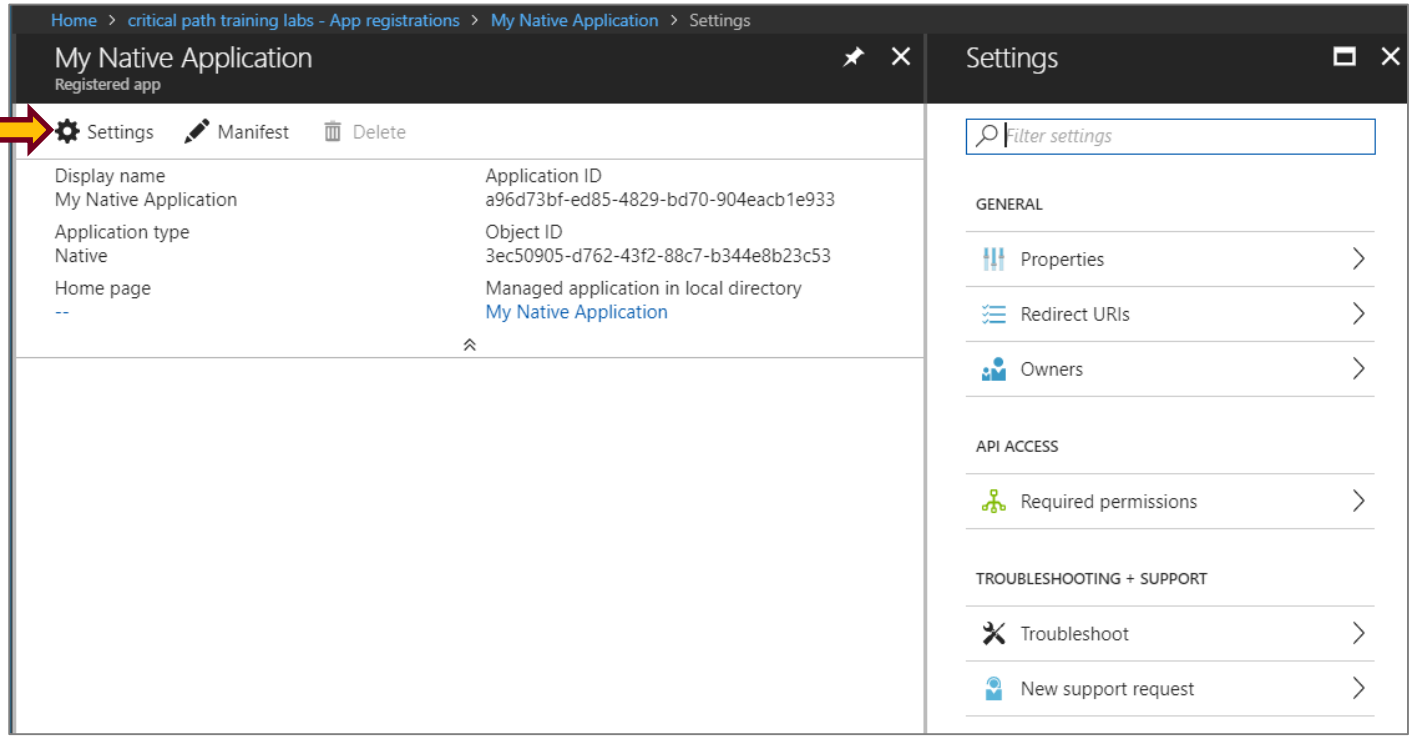
Copying the Application ID

- 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 IS from the azure portal



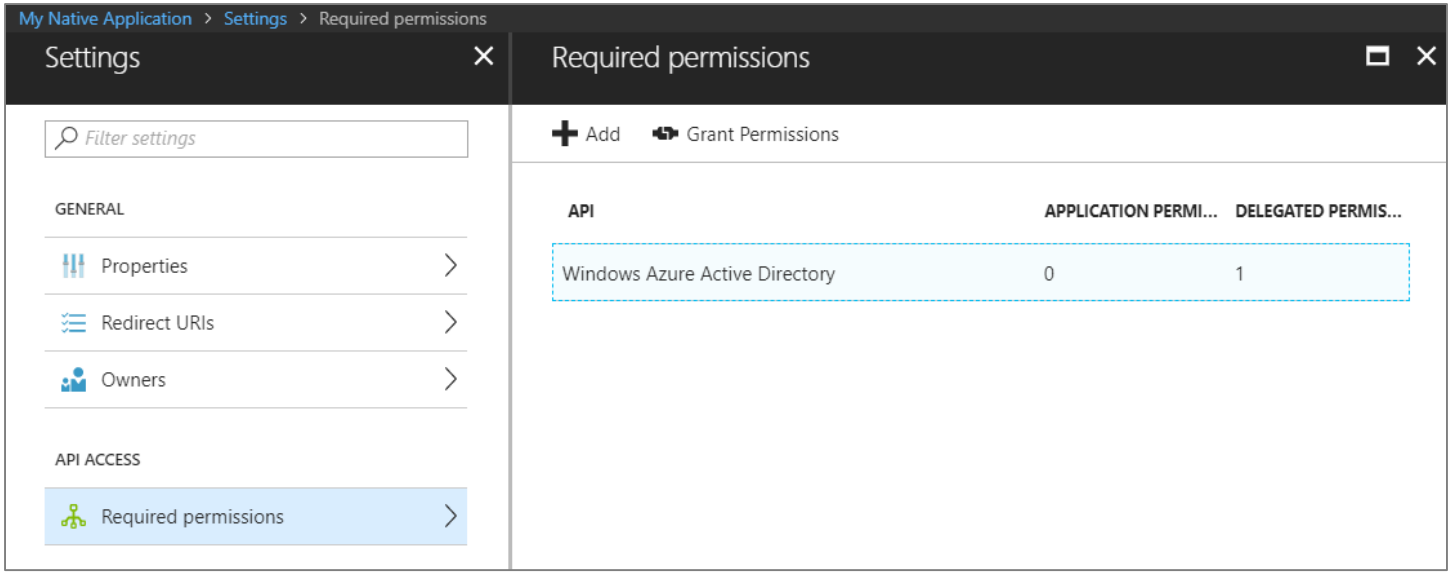
Native Application Settings

- Properties
- Redirect URLs
- Owners
- Required Permissions



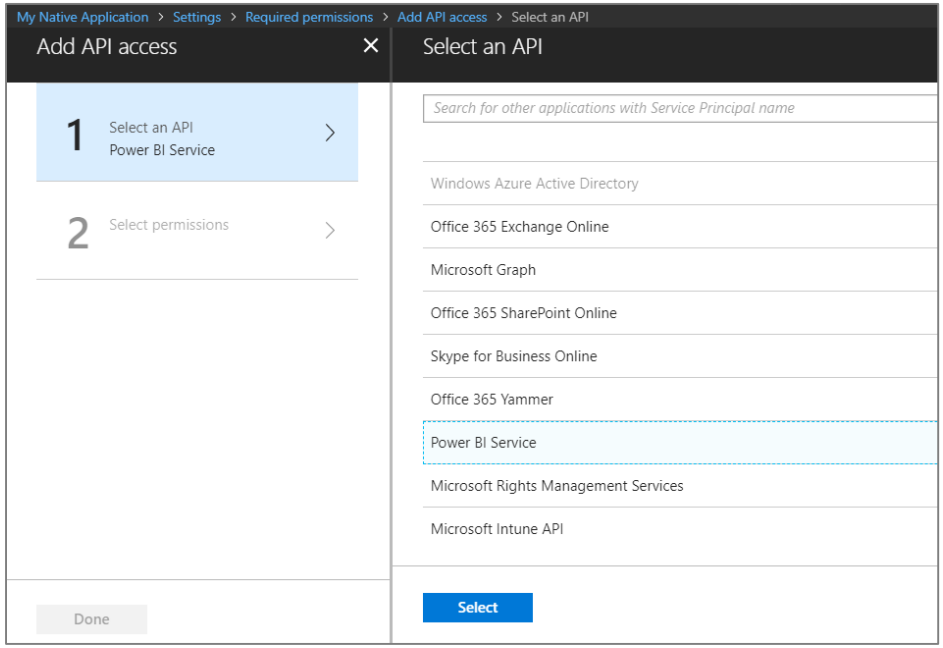
Configuring Required Permissions

- 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
  - Office 365 Exchange Online
  - Microsoft Graph
  - Office 365 SharePoint Online
  - Power BI Service



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
  - Note that 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

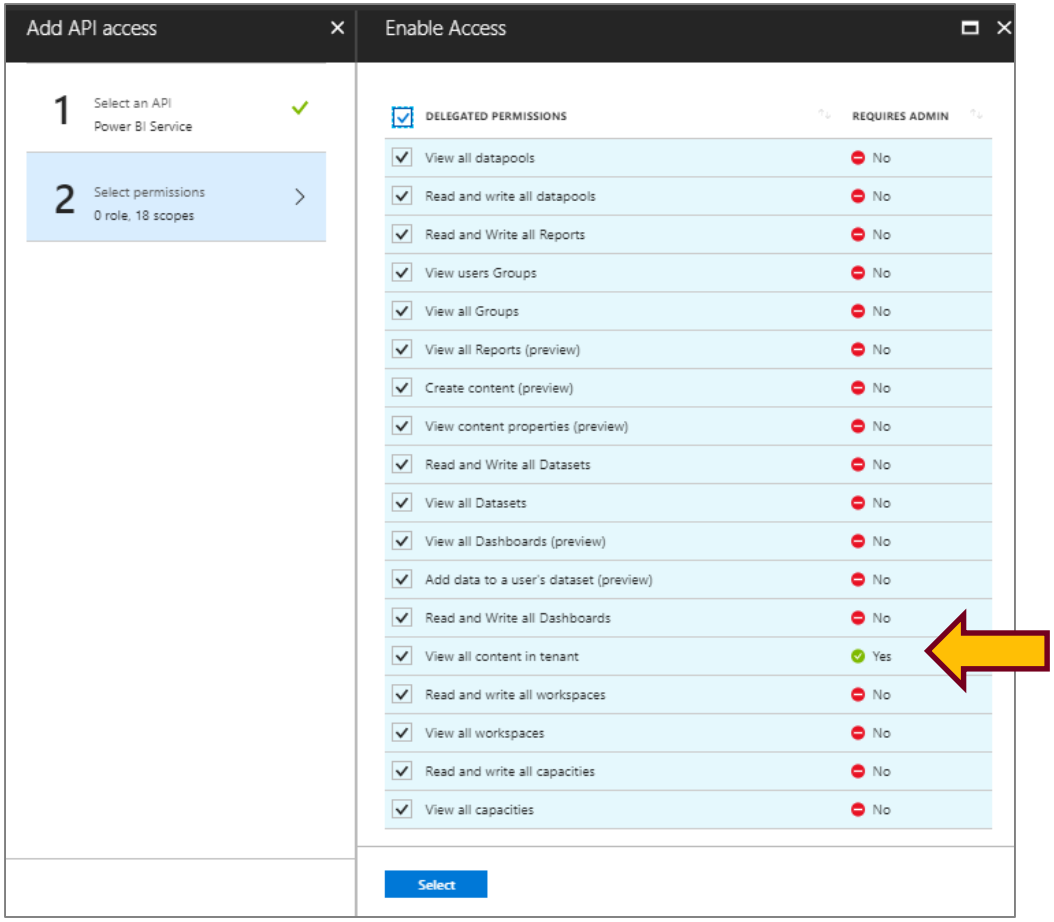
APPLICATION PERMISSIONS	REQUIRES ADMIN
Read user profiles	Yes
Read and write user profiles	Yes
Read and write managed metadata	Yes
Read managed metadata	Yes
Read and write items and lists in all site collections	Yes
Have full control of all site collections	Yes
Read items in all site collections	Yes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

Power BI Service API Permissions

- View all datapools
- Read and write all datapools
- Read and Write all Reports
- View users Groups
- View all Groups
- View all Reports (preview)
- Create content (preview)
- View content properties (preview)
- Read and Write all Datasets
- View all Datasets
- View all Dashboards (preview)
- Add data to a user's dataset (preview)
- Read and Write all Dashboards
- View all content in tenant (requires admin)
- Read and write all workspaces
- View all workspaces
- Read and write all capacities
- View all capacities

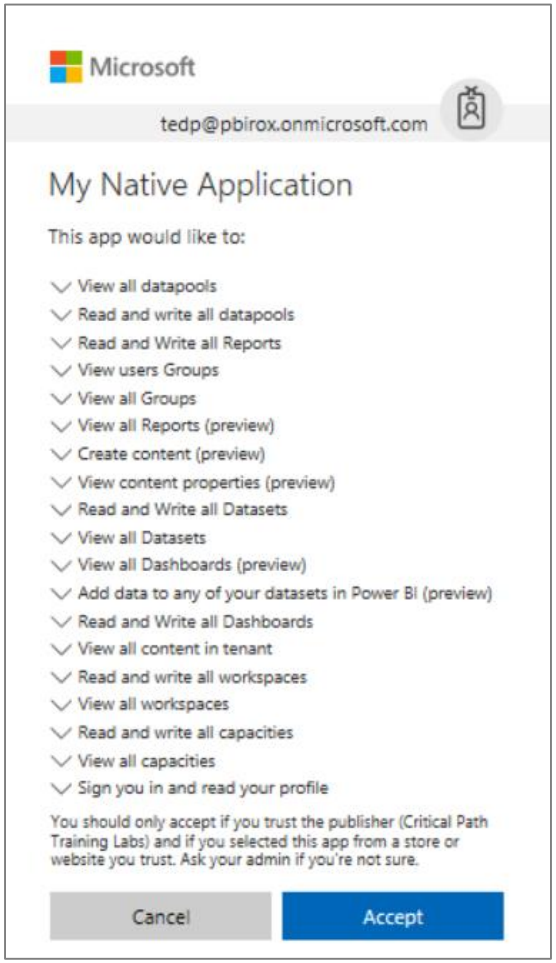
Power BI Service API Permissions

- Do you really need permissions that Requires Admin
  - It makes it so that only Power BI administrators can use your app



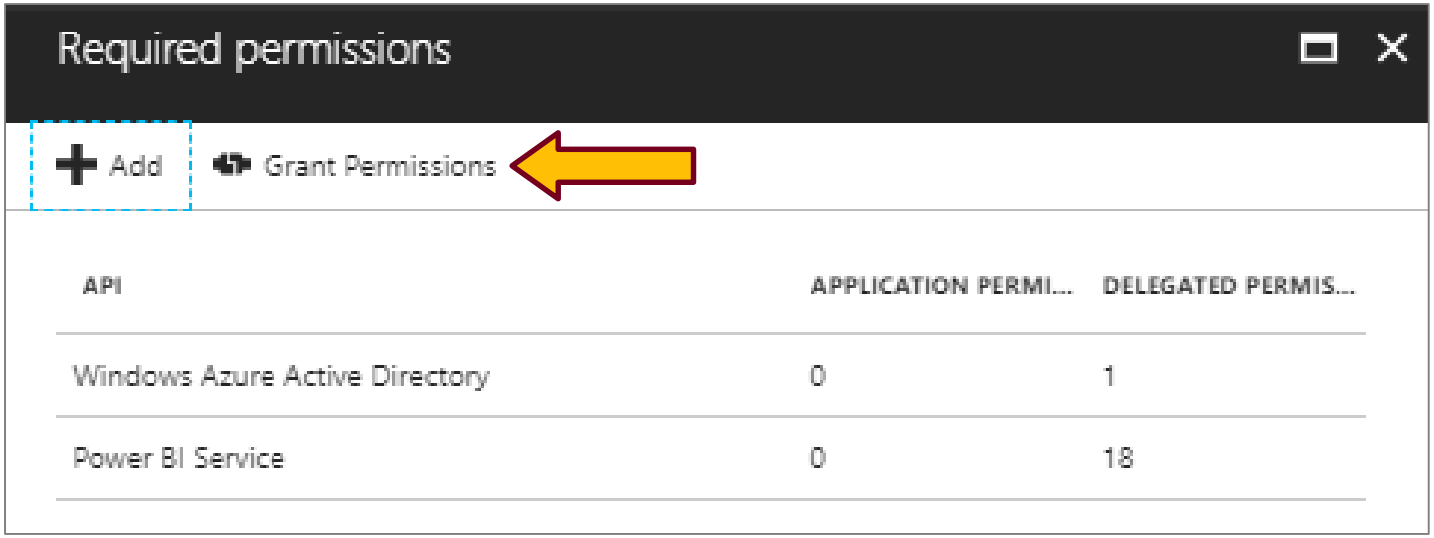
Interactive Consent for Delegated Permissions

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



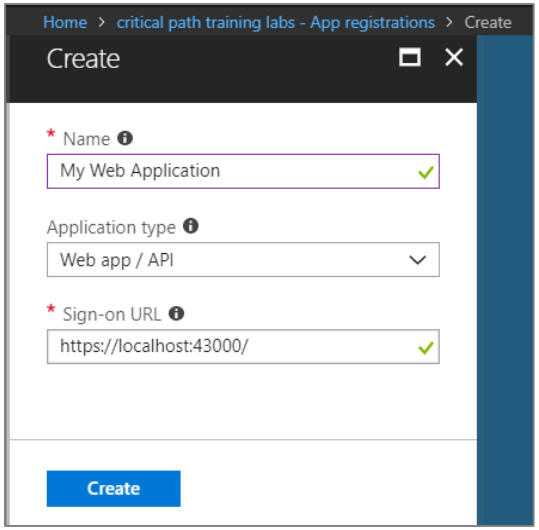
Granting Delegated Permissions

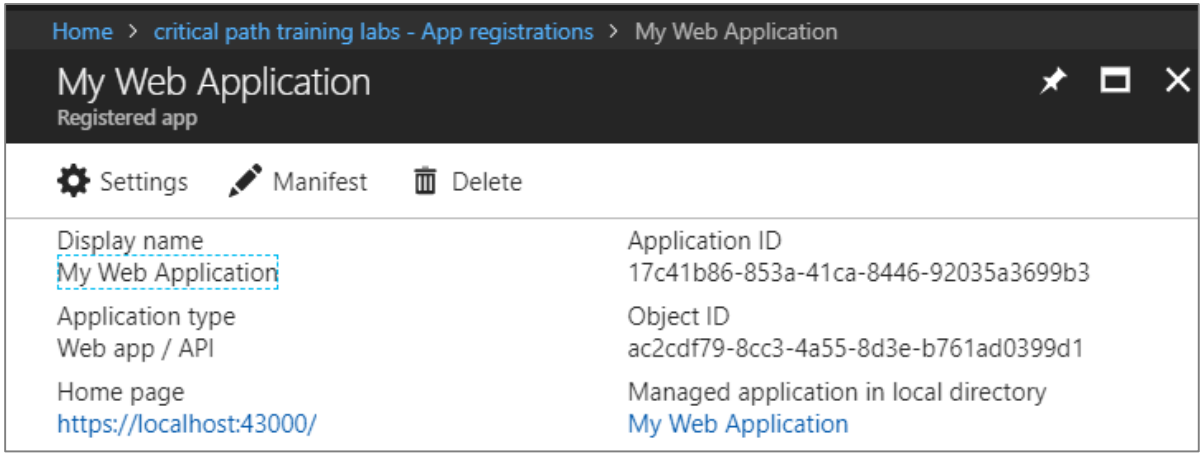
- It can be helpful to Grant Permissions in Azure portal
  - Prevents the need for interactive granting of application by user
  - Might be required when authenticating in non-interactive fashion



Creating Applications for Web Applications

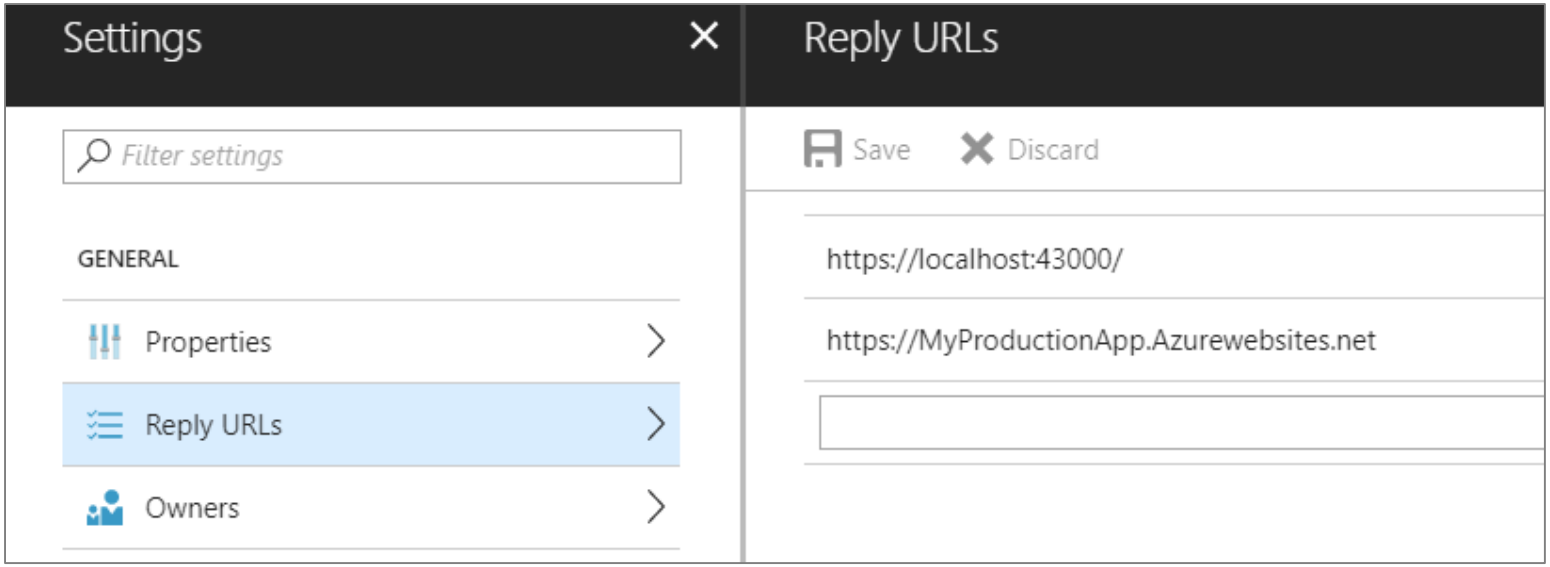
- Web applications more secure than native applications
  - Requires Redirect URI which improves security
  - Authentication can be used on client secret (application password)
  - Can use application permissions – Native applications cannot





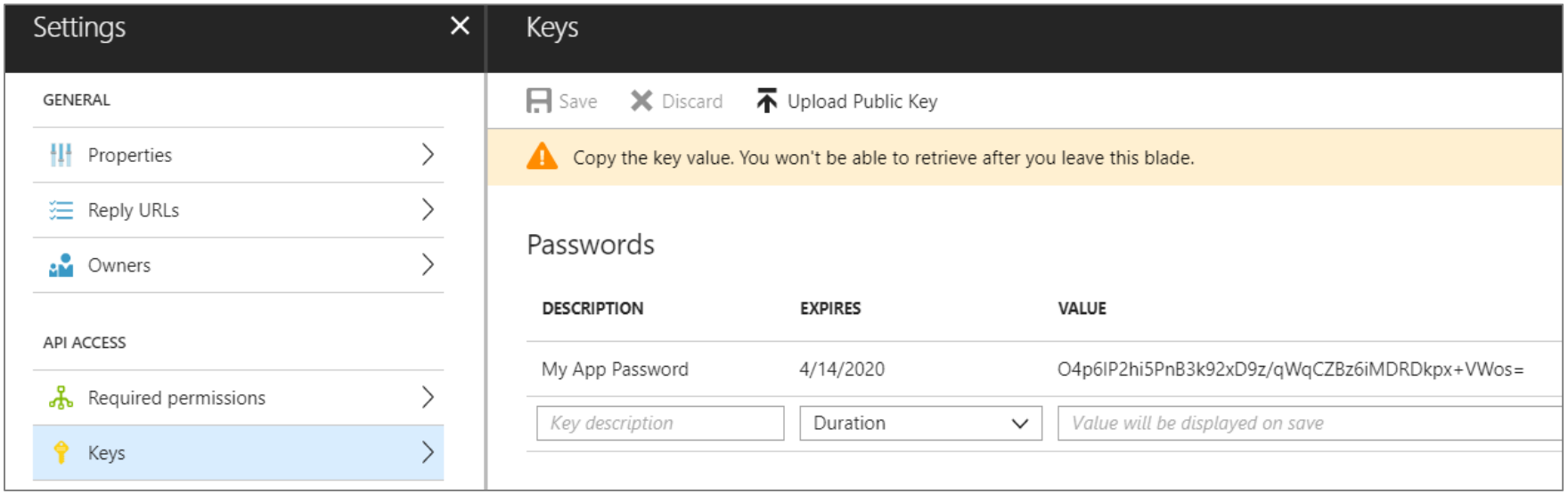
Reply URLs

- Reply URLs required for web applications
  - Your application must be accessible through the reply URL
  - Provides extra security dimension not available to native apps
  - Application can be configured with multiple reply URLs for single
  - Application must pass Reply URL matching registered Reply URL



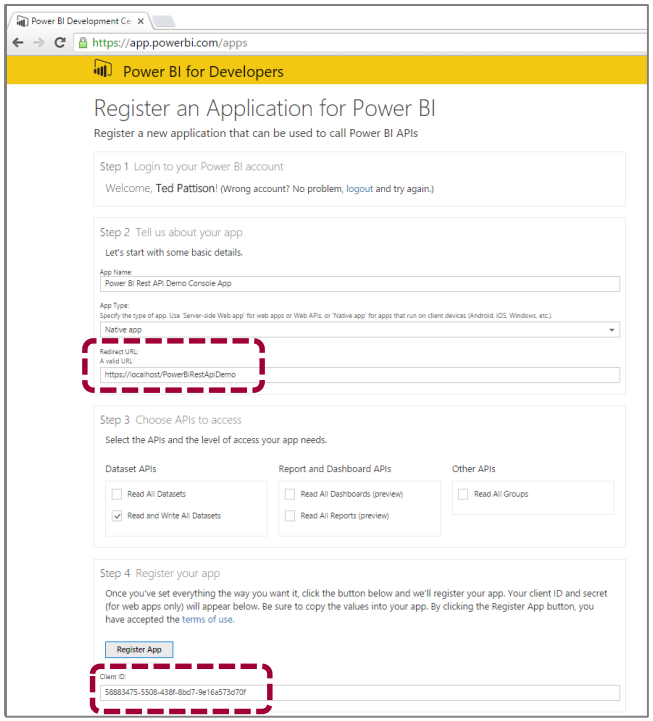
Creating Keys for Application Passwords

- Web applications authenticate using keys
  - Key acts as application-level password
  - Application requires copy of key value



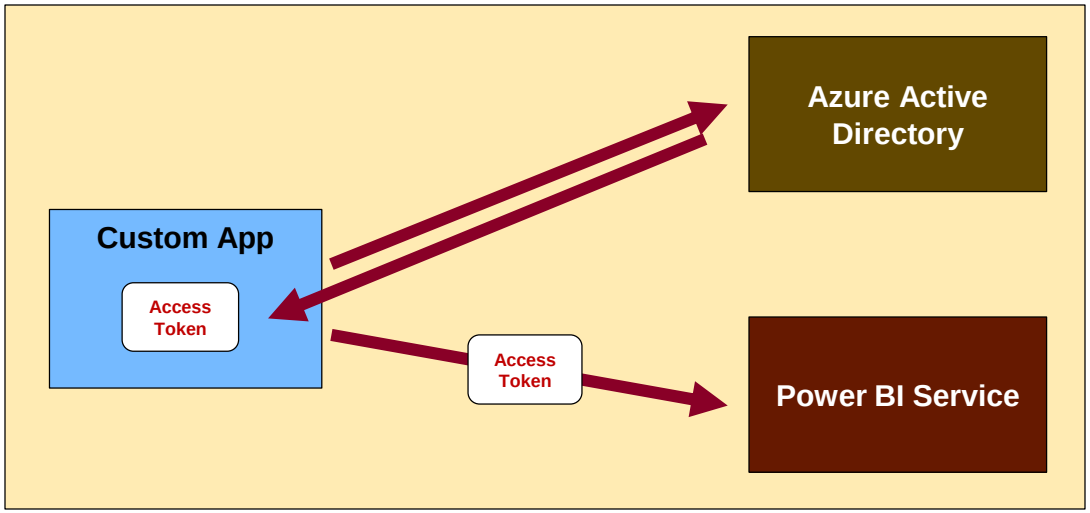
Power BI App Registration

- Power BI provides page to create Azure AD application
  - Accessible through https://app.powerbi.com/apps
  - Does not give the same level of control as Azure AD
  - RECOMMENDATION: use Azure portal instead of this page



Authenticating with Azure AD

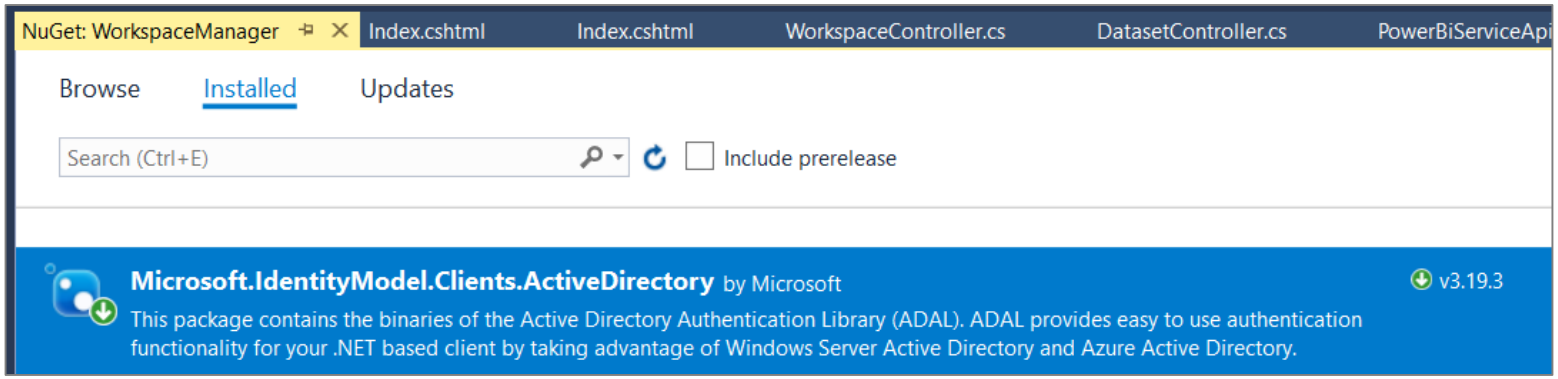
- User must be authenticated against Azure AD
  - User authentication used to obtain access token
  - Can be accomplished with the Azure AD Authentication Library
  - Access token pass to Power BI Service API in call REST calls



ADAL for .NET

- Active Directory Authentication Library for .NET
  - Used in Native Clients and in Web Clients
  - Handles authentication flow behind the scenes
  - Provides caching for access tokens and refresh tokens

- ADAL .NET installs as a NuGet Package
  - Package name is `Microsoft.IdentityModel.Clients.ActiveDirectory`



Access Token Acquisition

```
private static string aadInstance = "https://login.microsoftonline.com/";
private static string resourceUrlPowerBi = "https://analysis.windows.net/powerbi/api";
private static string urlPowerBiRestApiRoot = "https://api.powerbi.com/";

private static string clientId = ConfigurationManager.AppSettings["client-id"];
private static string clientSecret = ConfigurationManager.AppSettings["client-secret"];
private static string redirectUrl = ConfigurationManager.AppSettings["reply-url"];

private static async Task<string> GetAccessTokenAsync() {

  // determine authorization URL for current tenant
  string tenantID = ClaimsPrincipal.Current.FindFirst("http://schemas.microsoft.com/identity/claims/tenantid").Value;
  string tenantAuthority = aadInstance + tenantID;

  // create ADAL cache object
  ApplicationDbContext db = new ApplicationDbContext();
  string signedInUserID = ClaimsPrincipal.Current.FindFirst(ClaimTypes.NameIdentifier).Value;
  ADALTokenCache userTokenCache = new ADALTokenCache(signedInUserID);

  // create authentication context
  AuthenticationContext authenticationContext = new AuthenticationContext(tenantAuthority, userTokenCache);

  // create client credential object using client ID and client Secret"];
  ClientCredential clientCredential = new ClientCredential(clientId, clientSecret);

  // create user identifier object for logged on user
  string objectIdentifierId = "http://schemas.microsoft.com/identity/claims/objectidentifier";
  string userObjectID = ClaimsPrincipal.Current.FindFirst(objectIdentifierId).Value;
  UserIdentifier userIdentifier = new UserIdentifier(userObjectID, UserIdentifierType.UniqueId);

  // get access token for Power BI Service API from AAD
  AuthenticationResult authenticationResult =
    await authenticationContext.AcquireTokenSilentAsync(
        resourceUrlPowerBi,
        clientCredential,
        userIdentifier);

  // return access token back to user
  return authenticationResult.AccessToken;

}
```

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