

Developing Applications with Azure AD



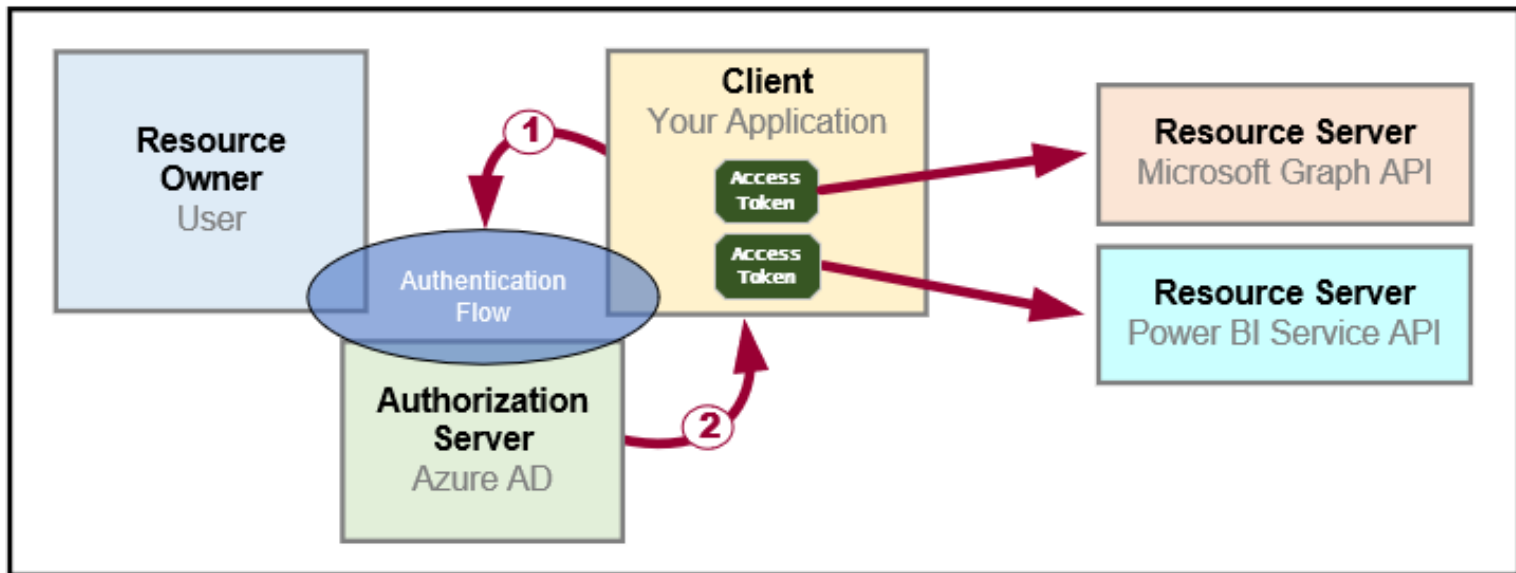
Agenda

- 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



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",

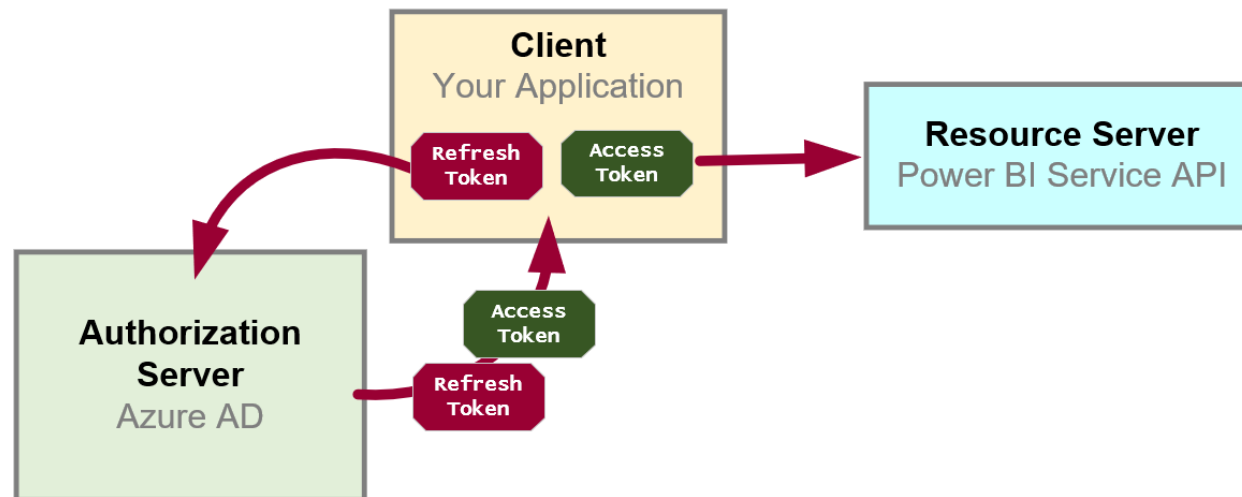
  "ipaddr": "47.200.98.132",

  "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

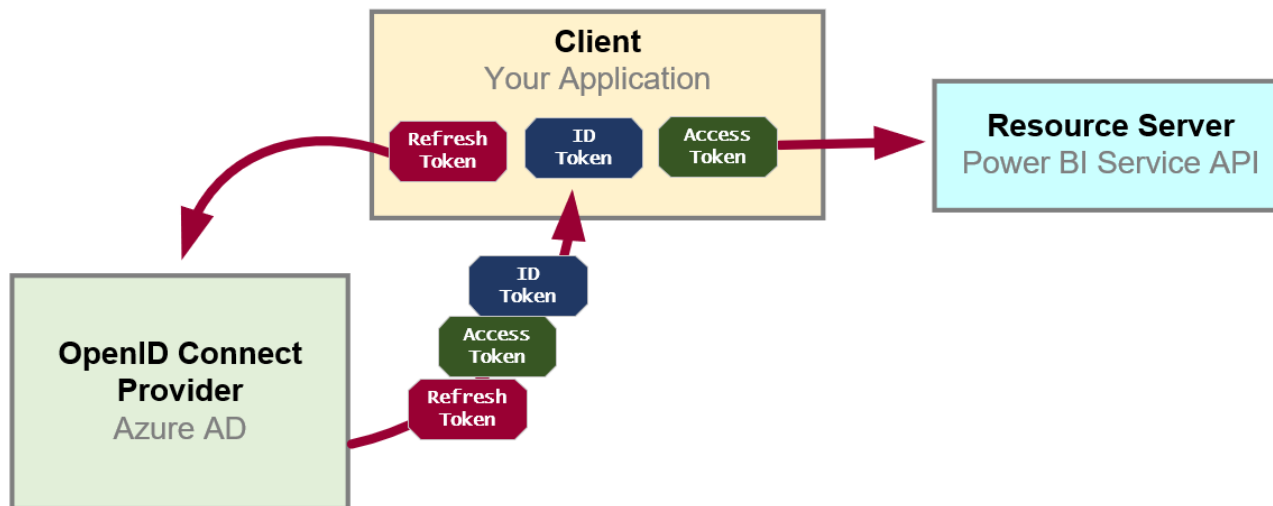
Registered Applications

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



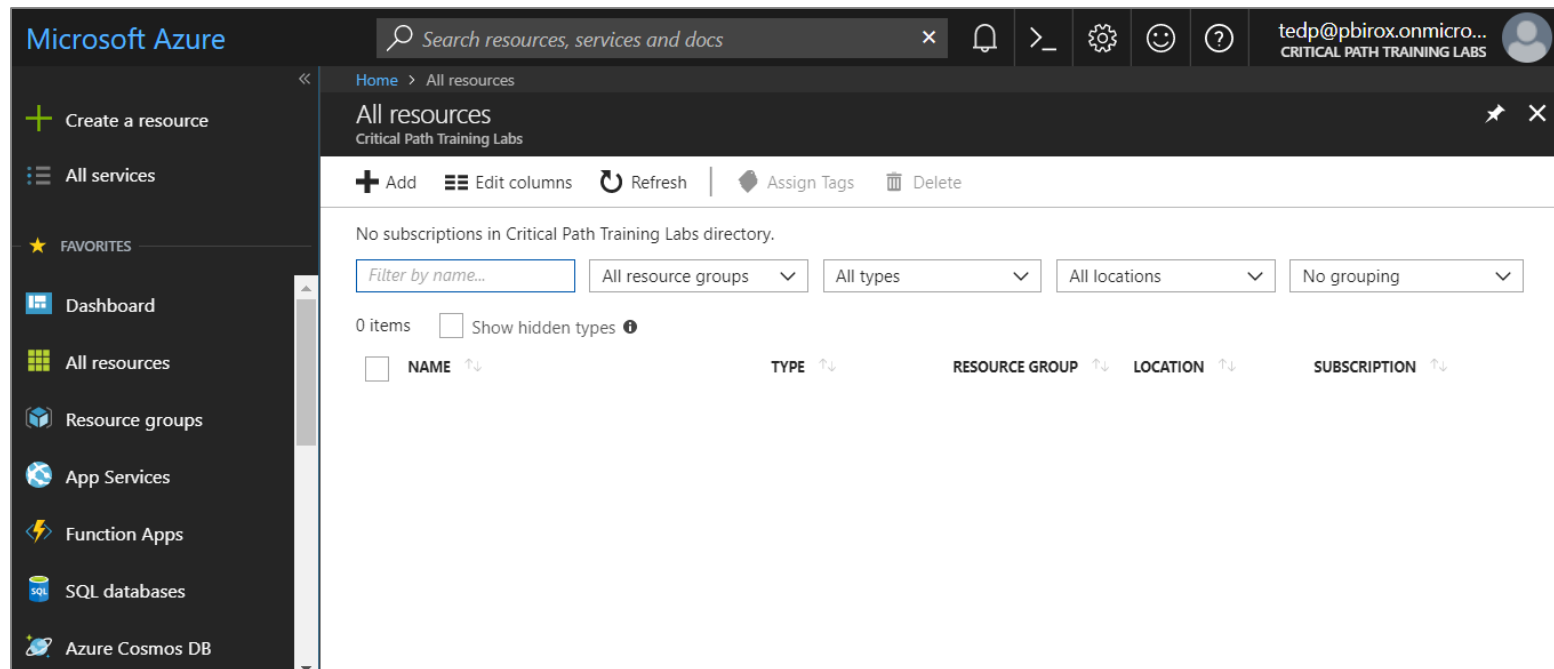
Agenda

- ✓ 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



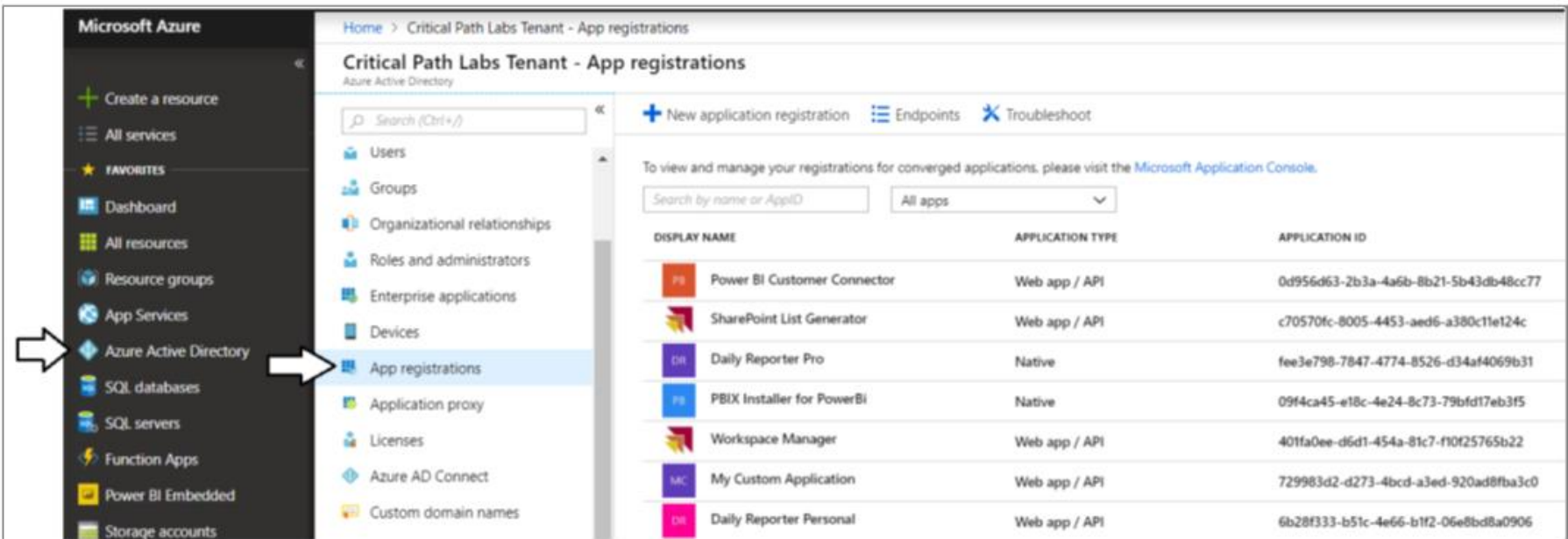
The Azure Portal

- 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 table below lists the applications with their display names, application types, and application IDs.

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

Home > critical path training labs - App registrations

critical path training labs - App registrations

Azure Active Directory

Search (Ctrl+/)

Users

Groups

Enterprise applications

Devices

App registrations

Application proxy

+ New application registration

Endpoints

Troubleshoot

To view and manage your registrations for converged applications, please visit the [Microsoft Application Console](#).

Search by name or AppID

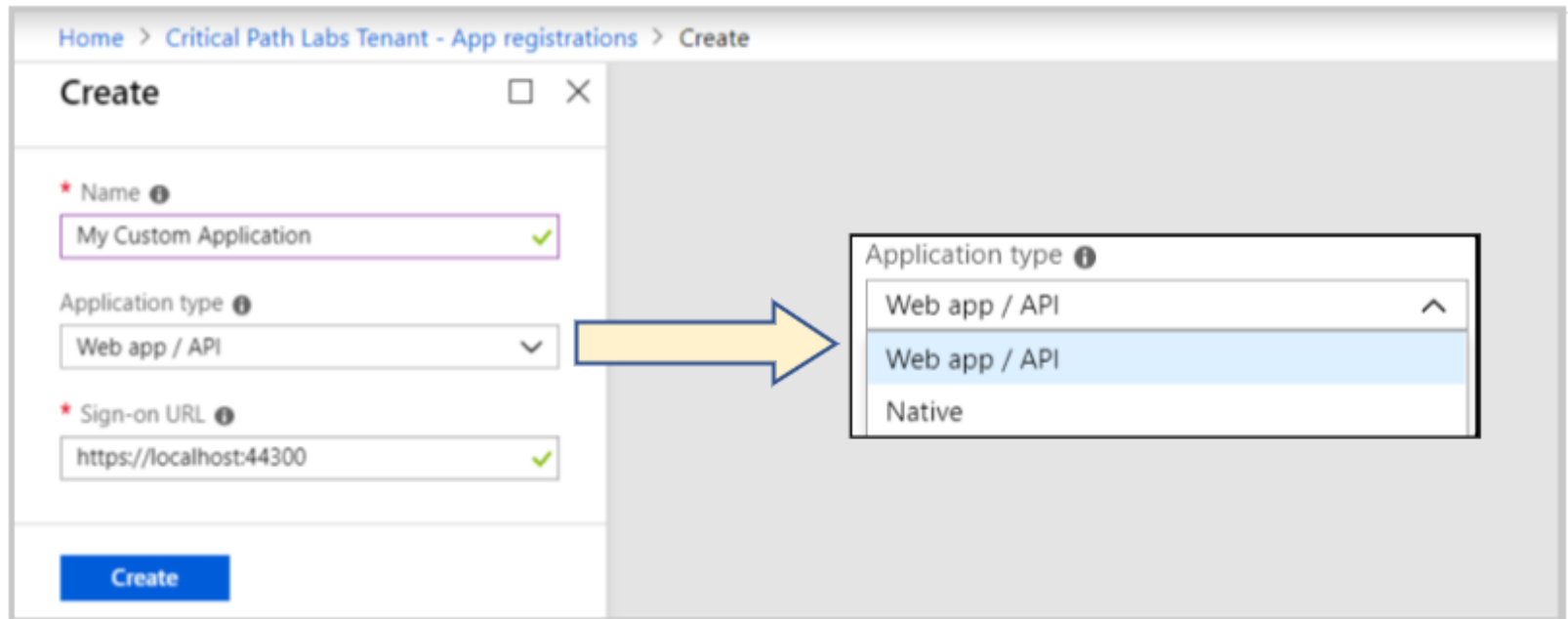
All apps

	DISPLAY NAME	APPLICATION TYPE	APPLICATION ID
MN	My Native Application	Native	a96d73bf-ed85-4829-bd70-904each1e933
MW	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 sections: 'Name' with the value 'My Custom Application', 'Application type' with a dropdown menu, and 'Sign-on URL' with the value 'https://localhost:44300'. A yellow arrow points from the 'Application type' dropdown in the form to a larger, detailed view of the dropdown menu on the right. This detailed view shows three options: 'Web app / API' (selected and highlighted in blue), 'Web app / API' (unselected), and 'Native' (unselected). Each option has an information icon (i) to its right.

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

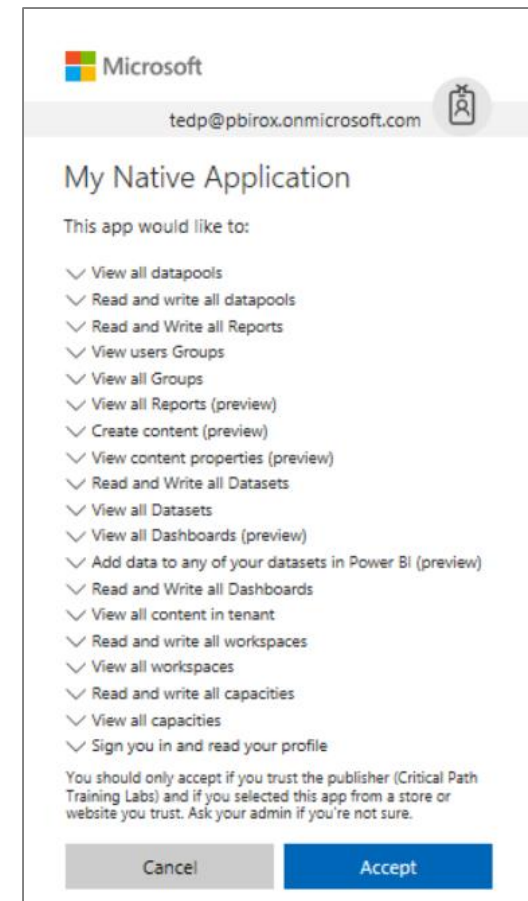
<input type="checkbox"/> DELEGATED PERMISSIONS	REQUIRES ADMIN
Run search queries as a user	✓ Yes
Read user profiles	✓ Yes
<input checked="" type="checkbox"/> Read user files	✗ No
Read managed metadata	✓ Yes
<input checked="" type="checkbox"/> Read items in all site collections	✗ No
Read and write user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



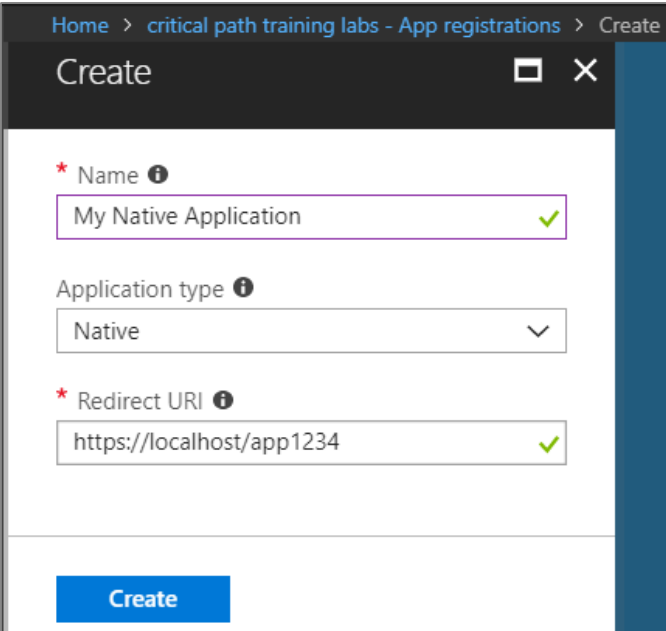
Interactive Consent for Delegated Permissions

- 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:

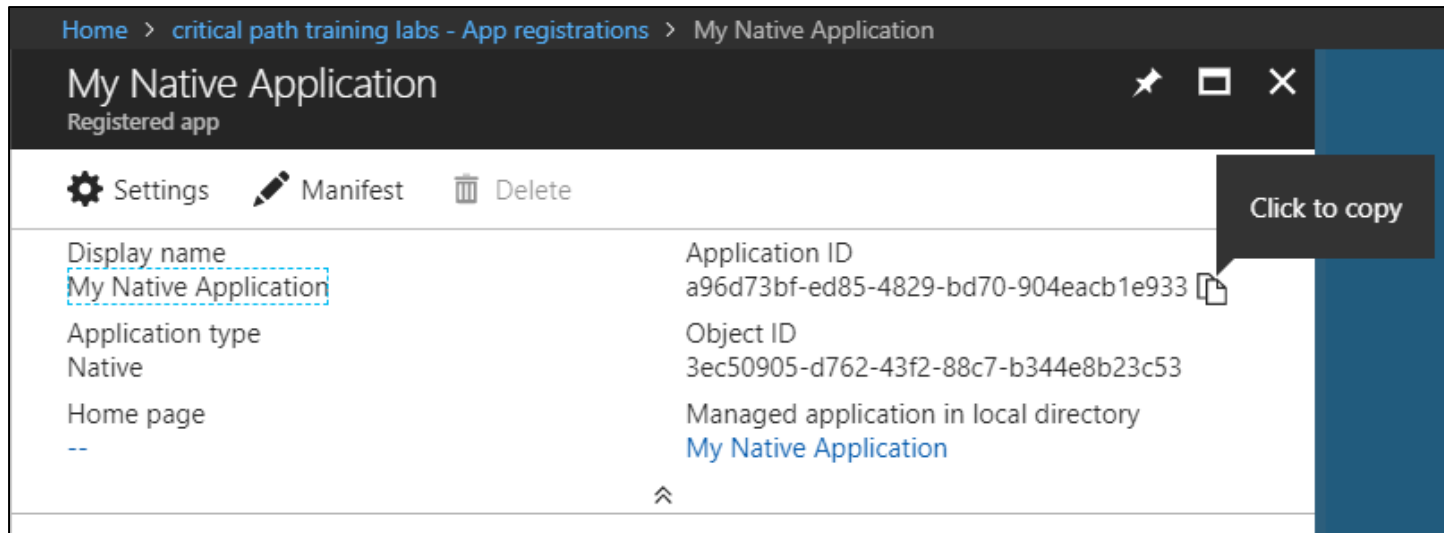
- Name:** The text 'My Native Application' is entered in the input field, followed by a green checkmark icon.
- Application type:** A dropdown menu is open, showing 'Native' as the selected option with a downward arrow.
- Redirect URI:** The text 'https://localhost/app1234' is entered in the input field,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

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



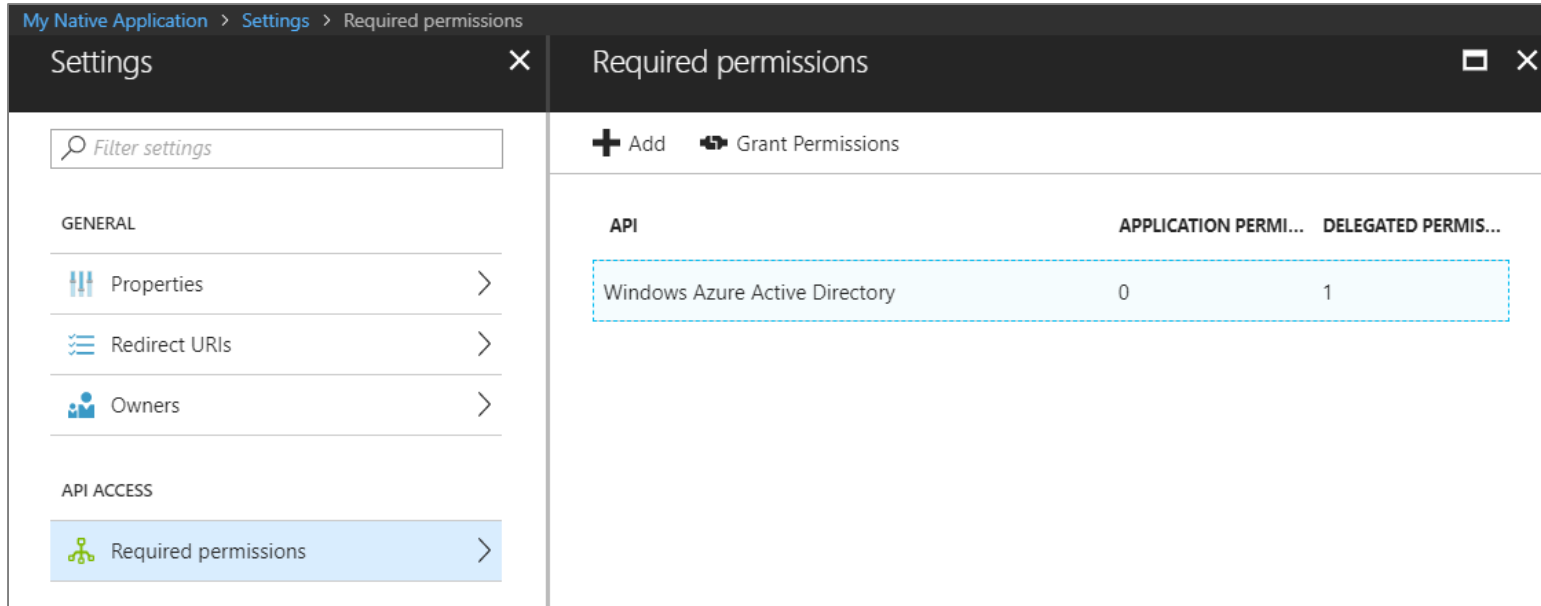
Copying the Application ID

- 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



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

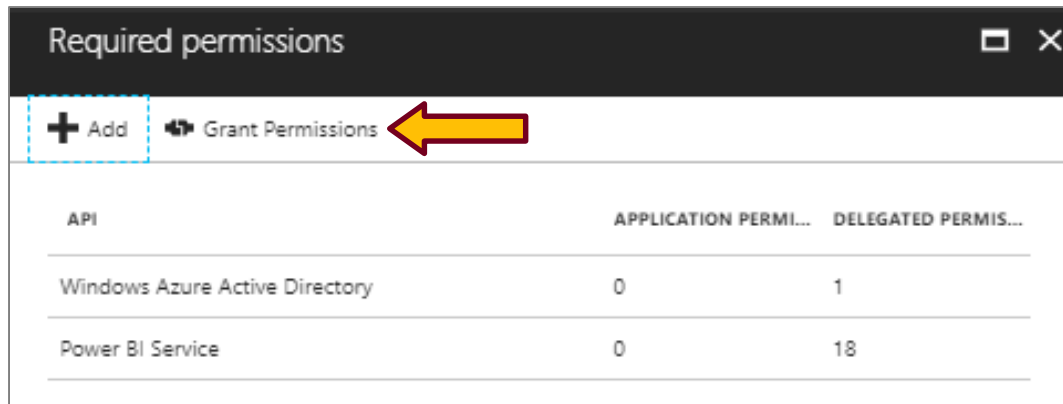
- There are lots of APIs to choose from
 - Microsoft Graph
 - Office 365 SharePoint Online
 - Power BI Service

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



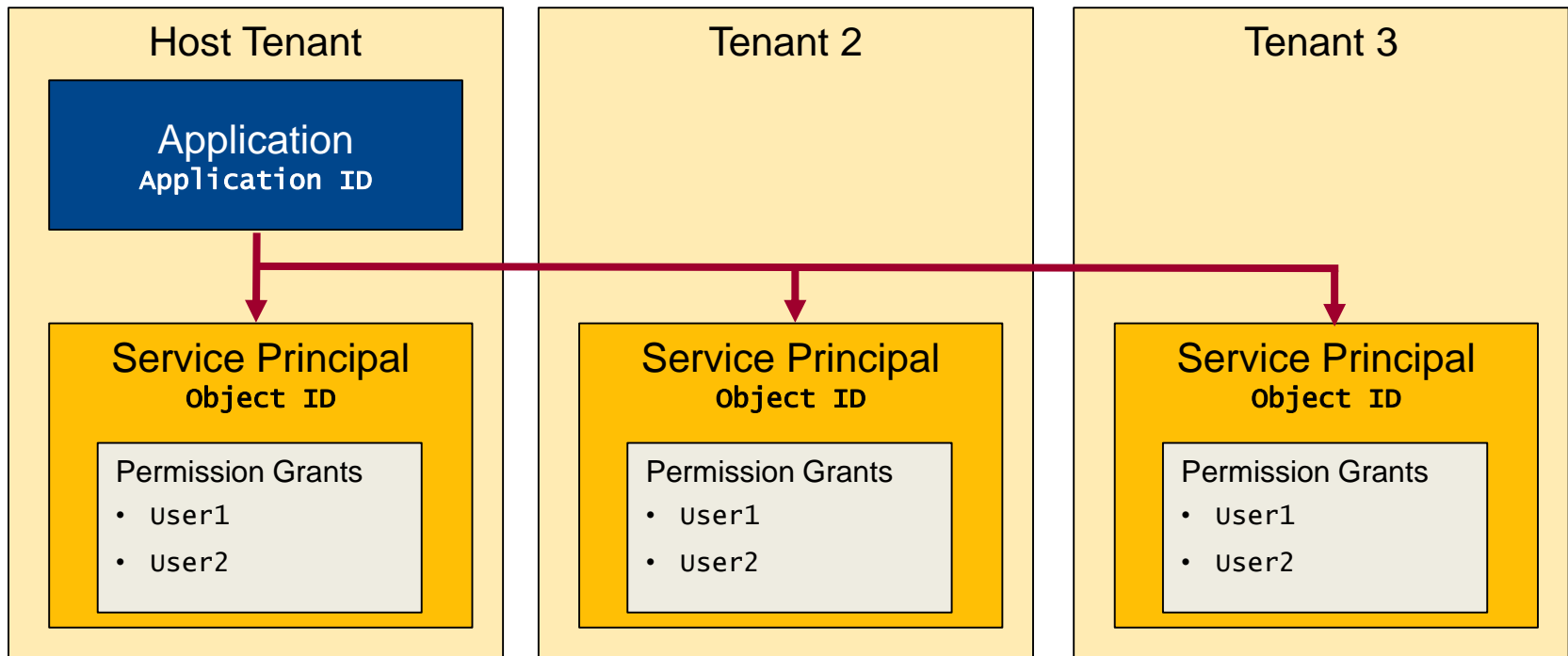


DEMO

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
```





DEMO

Creating Azure AD Applications using PowerShell Scripts

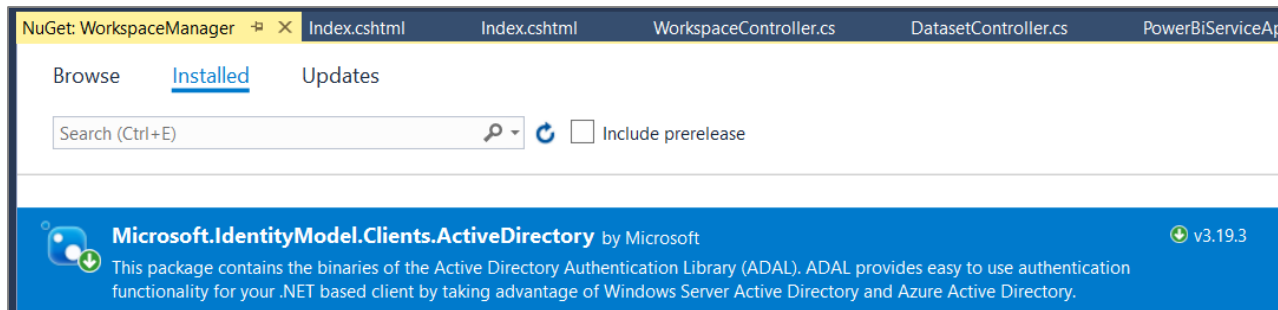
Agenda

- ✓ 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



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 (Native Client)

- Acquire access token using interactive login experience

```
static string aadAuthorizationEndpoint = "https://login.windows.net/common/oauth2/authorize";
static string resourceUriPowerBi = "https://analysis.windows.net/powerbi/api";
static string urlPowerBiRestApiRoot = "https://api.powerbi.com/";

public const string clientId = "315e87eb-a6a0-4886-9b20-9f7ecdaca888";
public const string redirectUrl = "https://localhost/app1234";

static string GetAccessToken() {
    // create new authentication context
    var authenticationContext = new AuthenticationContext(aadAuthorizationEndpoint);

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

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

- Acquire access token using User Password Credential flow (non-interactive)

```
string userName = "tedp@sharepointconfessions.onmicrosoft.com";
string userPassword = "Dublin@1234";

UserPasswordCredential creds = new UserPasswordCredential(userName, userPassword);
var userAuthnResult = authenticationContext.AcquireTokenAsync(PowerBiServiceResourceUri,
                                                            ClientID,
                                                            creds).Result;

// cache access token in AccessToken field
AccessToken = userAuthnResult.AccessToken;
```





DEMO

Using ADAL in a Native Client

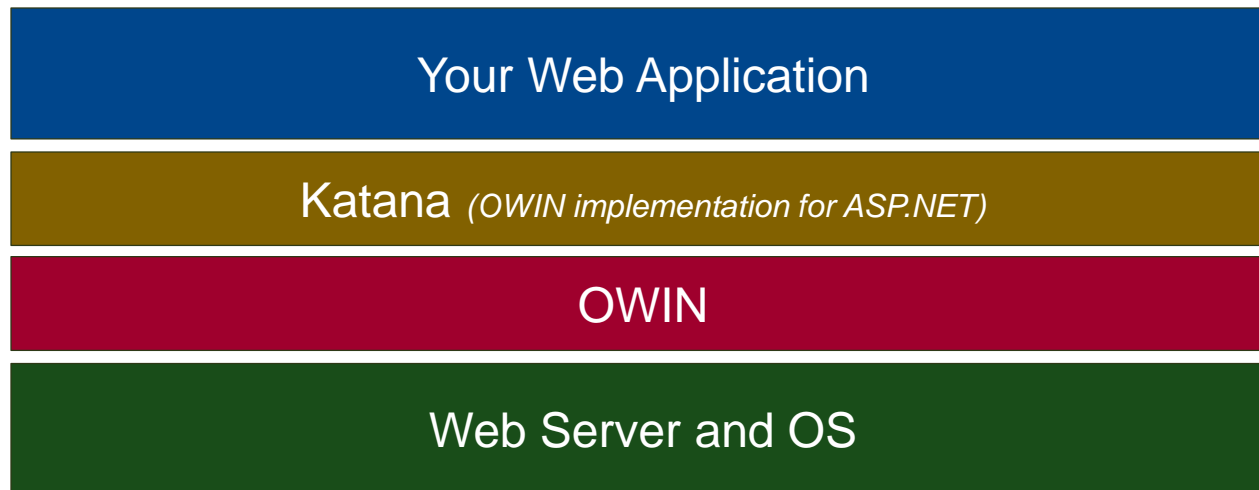
Agenda

- ✓ 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



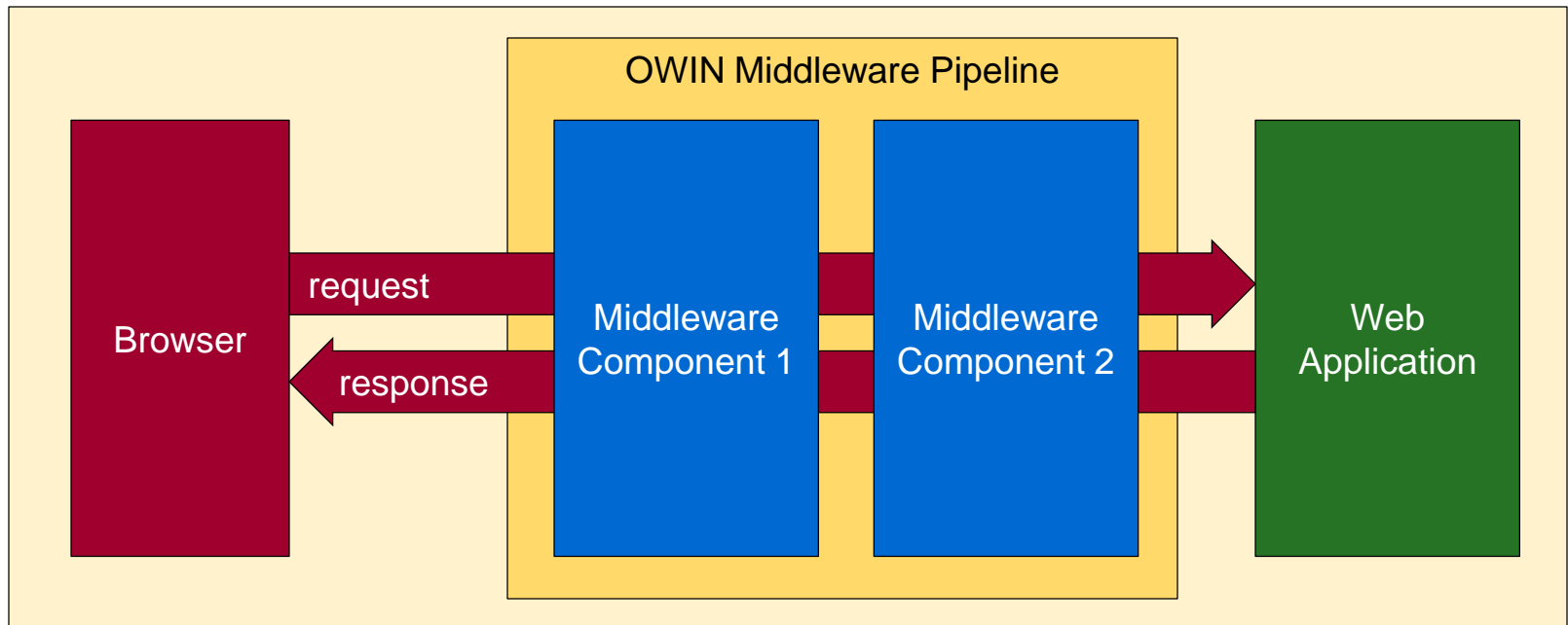
Open Web Interfaces for NET (OWIN)

- OWIN interfaces decouple web server from application
 - OWIN serves to decouple .NET applications from Windows and IIS
 - OWIN promotes the development of smaller modules (middleware)
- Microsoft's Implementation known as Katana
 - Makes it possible to use OWIN with ASP.NET and ASP Core
 - Microsoft provides OWIN-based security middleware

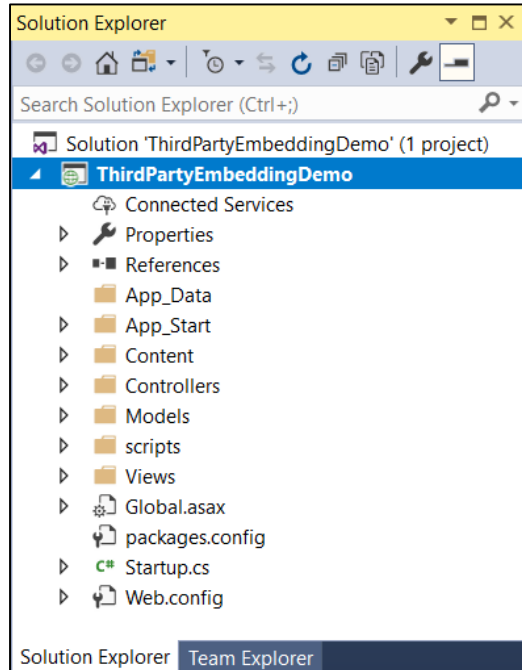


OWIN Middleware Modules

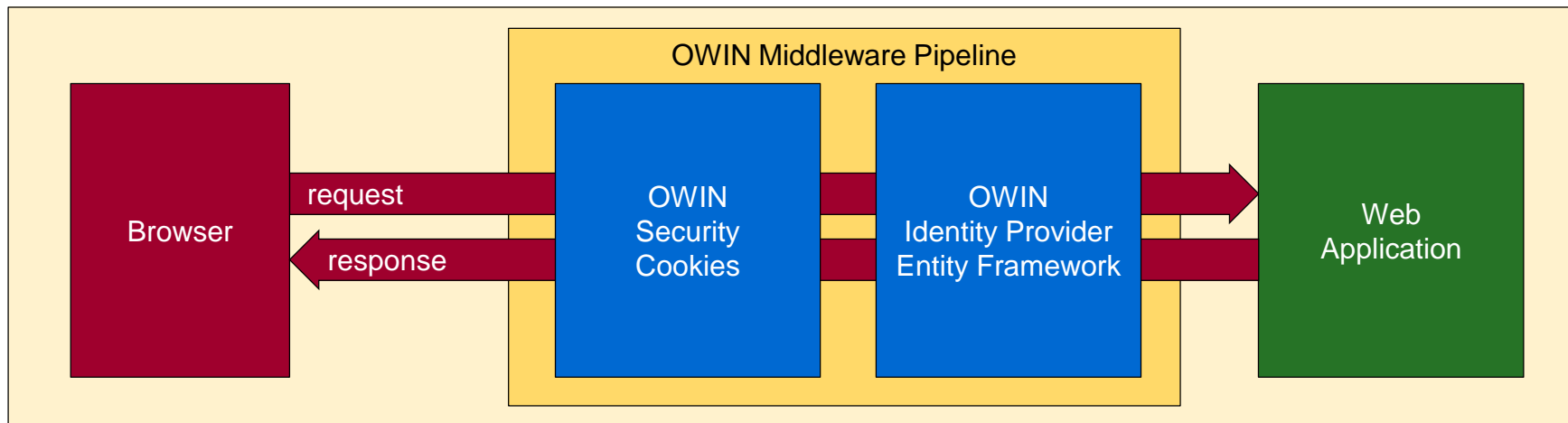
- OWIN create pipeline of middleware components
 - Middleware components added to pipeline on application startup
 - Middleware components pre-process and post process requests
 - Middleware components commonly used to set up authentication



ThirdPartyEmbeddingDemo



- Demonstration key points
 - Entity Framework Identity Provider
 - OWIN Security module



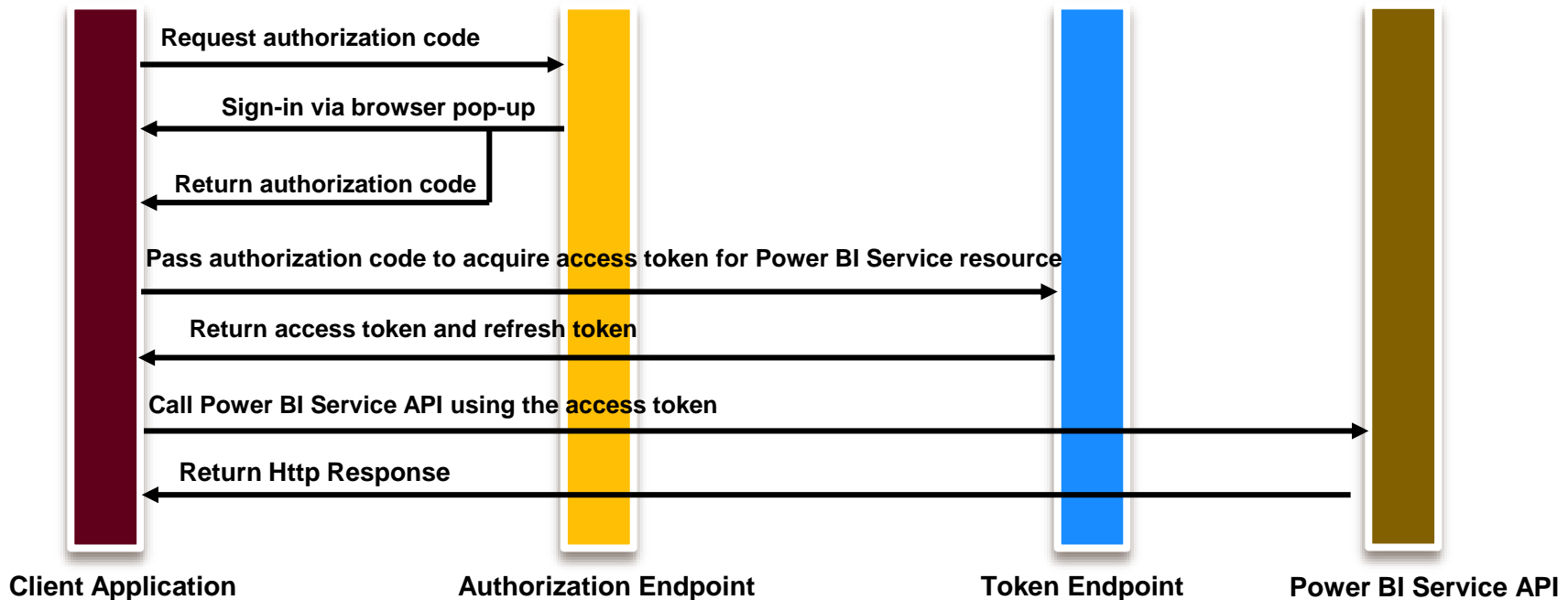
Agenda

- ✓ 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



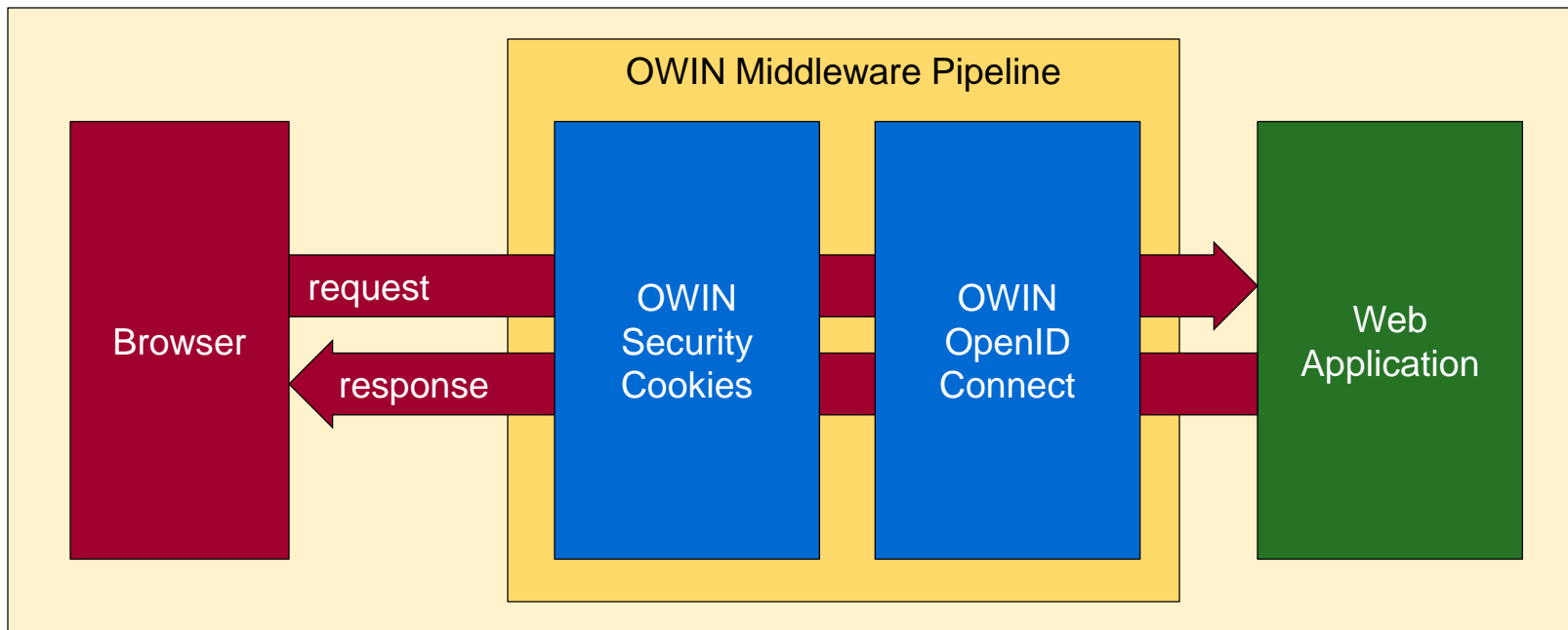
Authorization Code Grant Flow

- Sequence of Requests in Authorization Code Grant Flow
 - Application redirects to AAD authorization endpoint
 - User prompted to sign in using Windows logon page
 - User prompted to consent to permissions (first access)
 - AAD redirects to application with authorization code
 - Application calls to AAD token endpoint to acquire access token



OWIN OpenID Connect Module

- OpenID Connect module used to implement Authorization Code Flow
 - Redirects browsers to authorization endpoint
 - Provides notification when receiving authorization code callback



Initializing the OpenID Connect Module

```
public partial class Startup {

    private static string resourceUriPowerBi = "https://analysis.windows.net/powerbi/api";
    private static string aadInstance = "https://login.microsoftonline.com/";
    private static string commonAuthority = aadInstance + "powerbiembedding.onmicrosoft.com/";
    private static string claimsIdentifierForTenantId = "http://schemas.microsoft.com/identity/claims/tenantid";

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

    public void ConfigureAuth(IAppBuilder app) {
        // configure OWIN pipeline
        app.SetDefaultSignInAsAuthenticationType(CookieAuthenticationDefaults.AuthenticationType);
        app.UseCookieAuthentication(new CookieAuthenticationOptions());
        app.UseOpenIdConnectAuthentication(
            new OpenIdConnectAuthenticationOptions {
                ClientId = clientId,
                Authority = commonAuthority,
                TokenValidationParameters = new TokenValidationParameters { ValidateIssuer = false },
                PostLogoutRedirectUri = replyUrl,
                Notifications = new OpenIdConnectAuthenticationNotifications() {
                    AuthorizationCodeReceived = (context) => {
                        // get tenant ID
                        string tenantID = context.AuthenticationTicket.Identity.FindFirst(claimsIdentifierForTenantId).Value;
                        // create URL for tenant-specific authority
                        string tenantAuthority = aadInstance + tenantID + "/";
                        var code = context.Code;
                        ClientCredential credential = new ClientCredential(clientId, appKey);
                        string signedInUserID = context.AuthenticationTicket.Identity.FindFirst(ClaimTypes.NameIdentifier).Value;
                        AuthenticationContext authContext = new AuthenticationContext(tenantAuthority, new ADALTokenCache());
                        AuthenticationResult result =
                            authContext.AcquireTokenByAuthorizationCodeAsync(
                                code,
                                new Uri(replyUrl),
                                credential,
                                resourceUriPowerBi).Result;
                        return Task.FromResult(0);
                    }
                }
            }
        );
    }
}
```


Token Caching with ADAL.NET

- ADAL.NET provides support for token caching
 - Built-in ADAL token caching only works for desktop applications
 - For web applications, you must create custom `TokenCache`-derived class
 - Token caching does not work when using **common** endpoint

```
using System.IO;
using Microsoft.Identity.Core.Cache;
using Microsoft.IdentityModel.Clients.ActiveDirectory;

namespace DailyReporterPersonal.Models {

    class ADALTokenCache : TokenCache {

        public string CacheFilePath { get; }
        public string CacheFilePathV3 { get; }
        private static readonly object FileLock = new object();

        // Initialize persistent cache using a local file.
        public ADALTokenCache() ...

        // Empty the persistent cache
        public override void Clear() ...

        // Triggered before ADAL accesses the token cache.
        void BeforeAccessNotification(TokenCacheNotificationArgs args) ...

        // Triggered after ADAL accessed the cache.
        void AfterAccessNotification(TokenCacheNotificationArgs args) ...

        private byte[] ReadFromFileIfExists(string path) ...

        private static void WriteToFileIfNotNull(string path, byte[] blob) ...

    }
}
```



Retrieving Tokens from the Token Cache

```
private const string claimsIdentifierForTenantId = "http://schemas.microsoft.com/identity/claims/tenantid";
private const string resourceUriPowerBi = "https://analysis.windows.net/powerbi/api";
private const string urlPowerBiServiceApiRoot = "https://api.powerbi.com/";

// get tenant-specific authorization URL
private const string aadInstance = "https://login.microsoftonline.com/";
private readonly static string tenantID = ClaimsPrincipal.Current.FindFirst(claimsIdentifierForTenantId).Value;
private readonly static string tenantAuthority = aadInstance + tenantID;

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

private static string GetAccessToken() {

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

    // 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 =
        authenticationContext.AcquireTokenSilentAsync(
            resourceUriPowerBi,
            clientCredential,
            userIdentifier).Result;

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





DEMO

DailyReporterPersonal

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



Understanding Implicit Flow

- Single Pages Applications (SPAs) are public clients
 - SPA not able to keep secrets such as application secret
 - No ability to execute server-to-server calls
 - SPAs cannot implement authorization code flow
- Implicit flow requires lowering security bar for SPAs
 - Azure AD application must be configured to allow implicit flow
 - Allows SPAs to retrieve access tokens
 - Access token returned to browser in URL fragment





DEMO

PowerBIDaySPA

Summary of OAuth Client Types

	Web Client SPA	Native Client	Web Application Client	Web Service Client
Client Type	Public	Public	Confidential	Confidential
Verifiable Reply URL	Yes	No	Yes	Yes
Authenticates Client	No	No	Yes	Yes
Token from Authorization Endpoint	Yes	Yes	No	No
Access Token from URI Fragment	Yes	No	No	No
Token from Token Endpoint	No	Yes	Yes	Yes
Can use refresh tokens	No	Yes	Yes	Yes
Permissions	Delegated	Delegated	Delegated + App	Delegated + App



Summary

- ✓ 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

