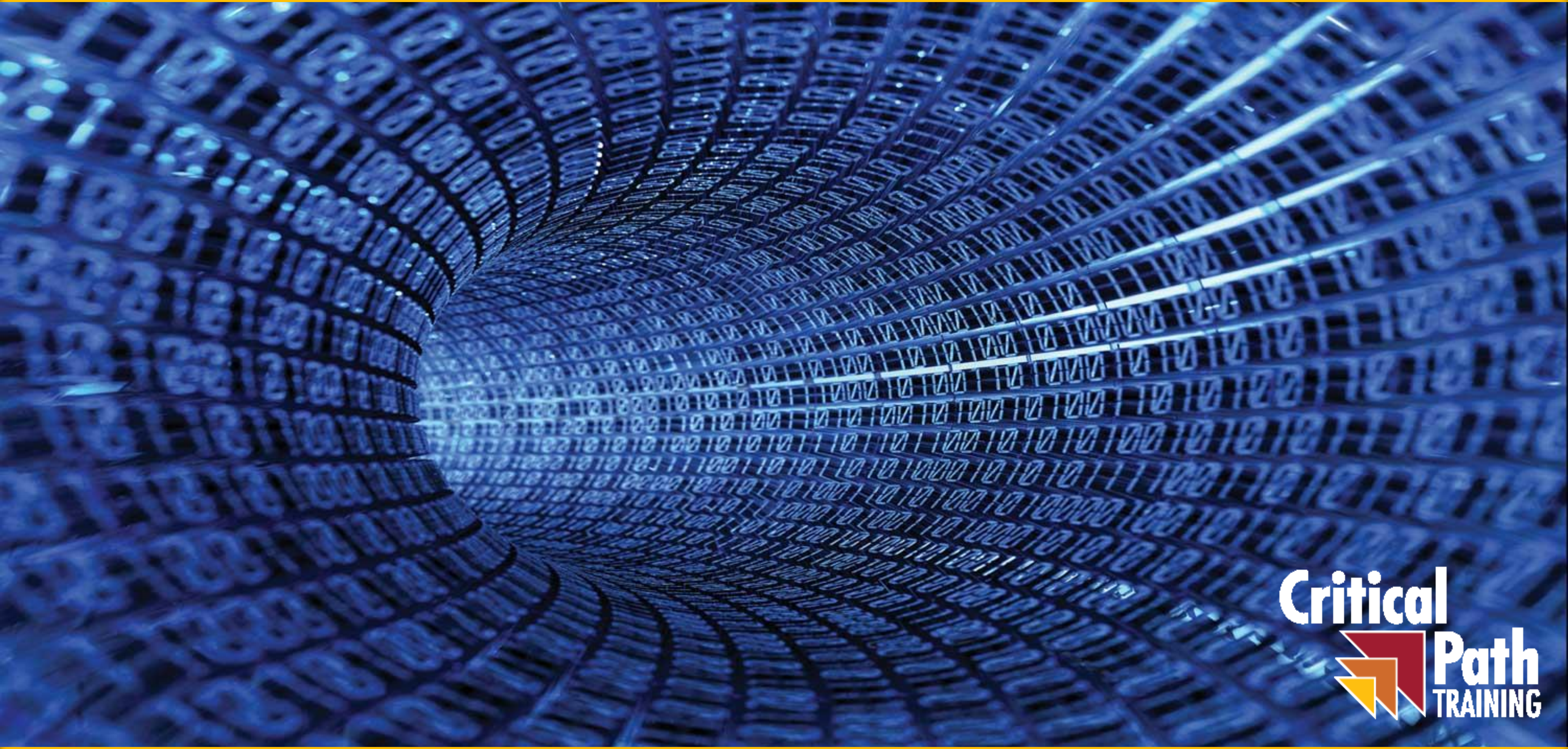
Developing Secure Applications using Azure AD

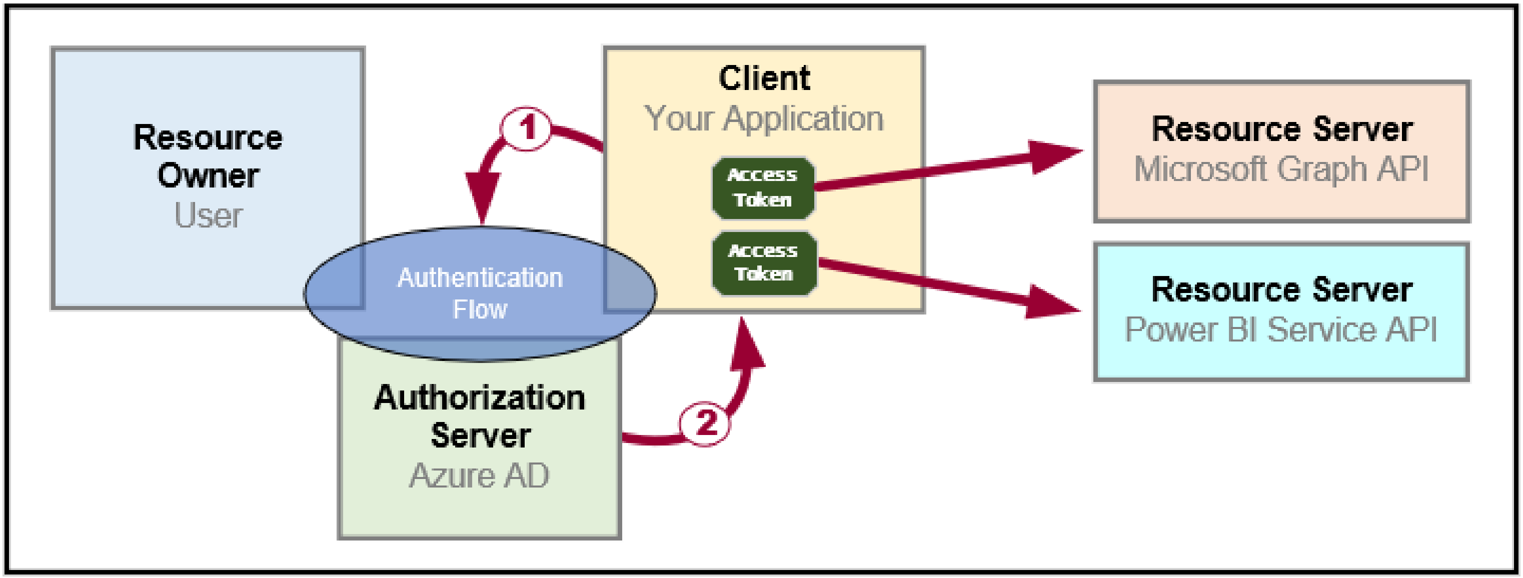


Agenda

- Understanding OAuth 2.0 and OpenID Connect
- Creating & Configuring Azure AD Applications
- Acquiring Access Tokens with MSAL
- Implementing Implicit Flow
- Implementing Authorization Code Flow

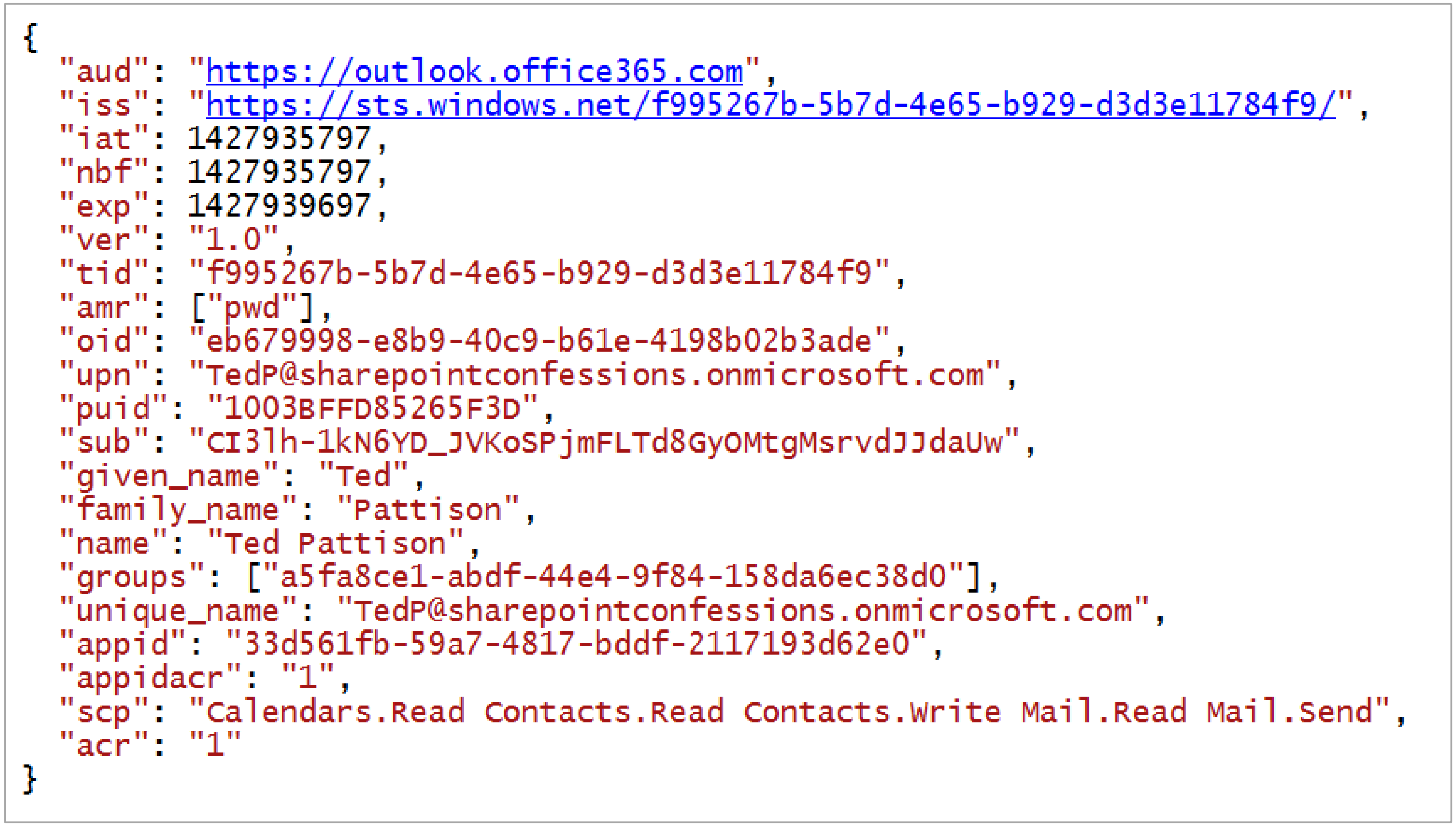
OAuth 2.0 Fundamentals

- 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

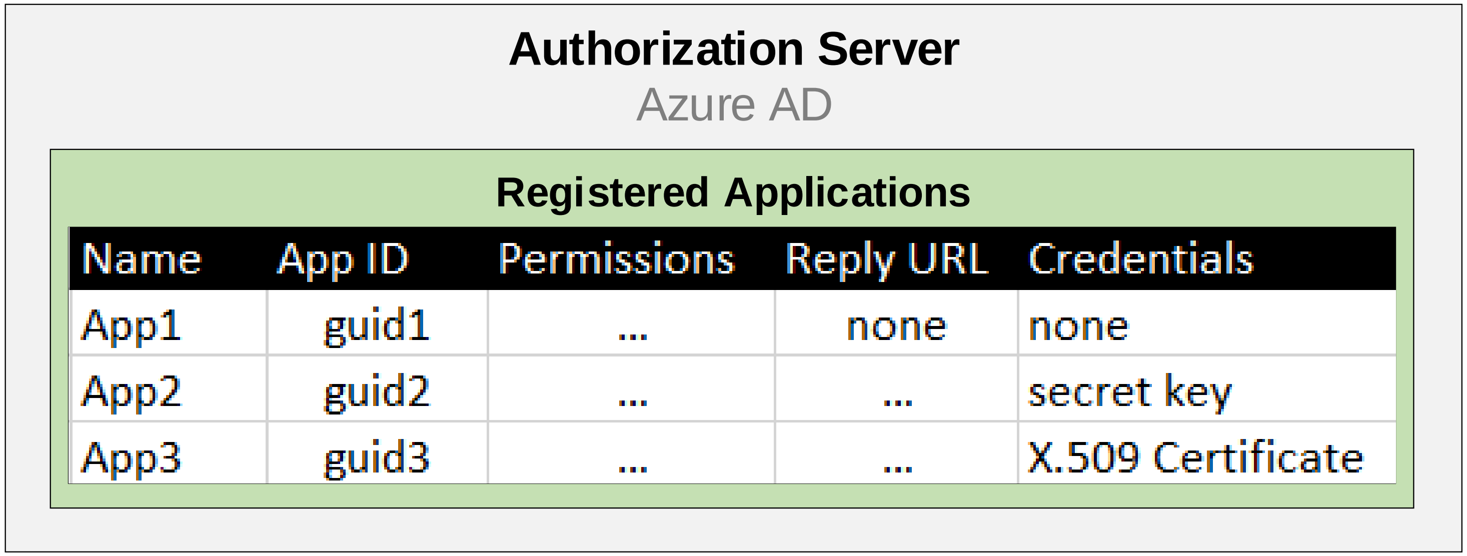
- It can be used by any who bears (e.g. steals) it
  - Always encrypt with HTTPS when transmitting access tokens



```
{
  "aud": "https://outlook.office365.com",
  "iss": "https://sts.windows.net/f995267b-5b7d-4e65-b929-d3d3e11784f9/",
  "iat": 1427935797,
  "nbf": 1427935797,
  "exp": 1427939697,
  "ver": "1.0",
  "tid": "f995267b-5b7d-4e65-b929-d3d3e11784f9",
  "amr": ["pwd"],
  "oid": "eb679998-e8b9-40c9-b61e-4198b02b3ade",
  "upn": "TedP@sharepointconfessions.onmicrosoft.com",
  "puid": "1003BFFD85265F3D",
  "sub": "CI3lh-1kN6YD_JVKoSPjmFLTd8GyOMtgMsrvdJJdaUw",
  "given_name": "Ted",
  "family_name": "Pattison",
  "name": "Ted Pattison",
  "groups": ["a5fa8ce1-abdf-44e4-9f84-158da6ec38d0"],
  "unique_name": "TedP@sharepointconfessions.onmicrosoft.com",
  "appid": "33d561fb-59a7-4817-bddf-2117193d62e0",
  "appidacr": "1",
  "scp": "Calendars.Read Contacts.Read Contacts.Write Mail.Read Mail.Send",
  "acr": "1"
}
```

OAuth 2.0 Client Registration

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



Authorization Server

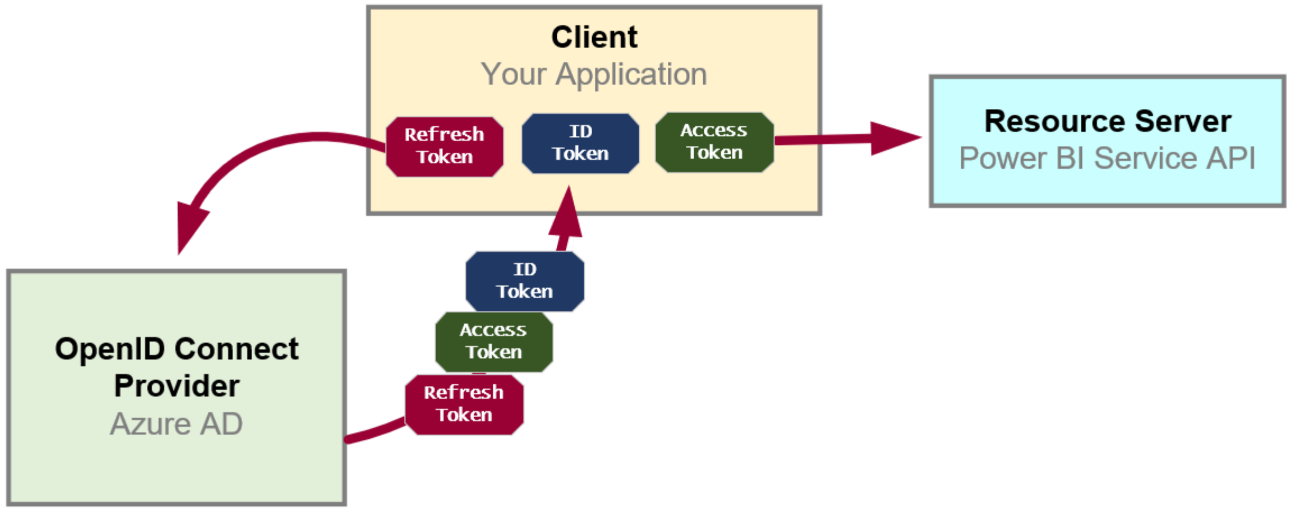
Azure AD

Registered Applications

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

OpenID Connect Extends OAuth 2.0

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



Authentication Flows

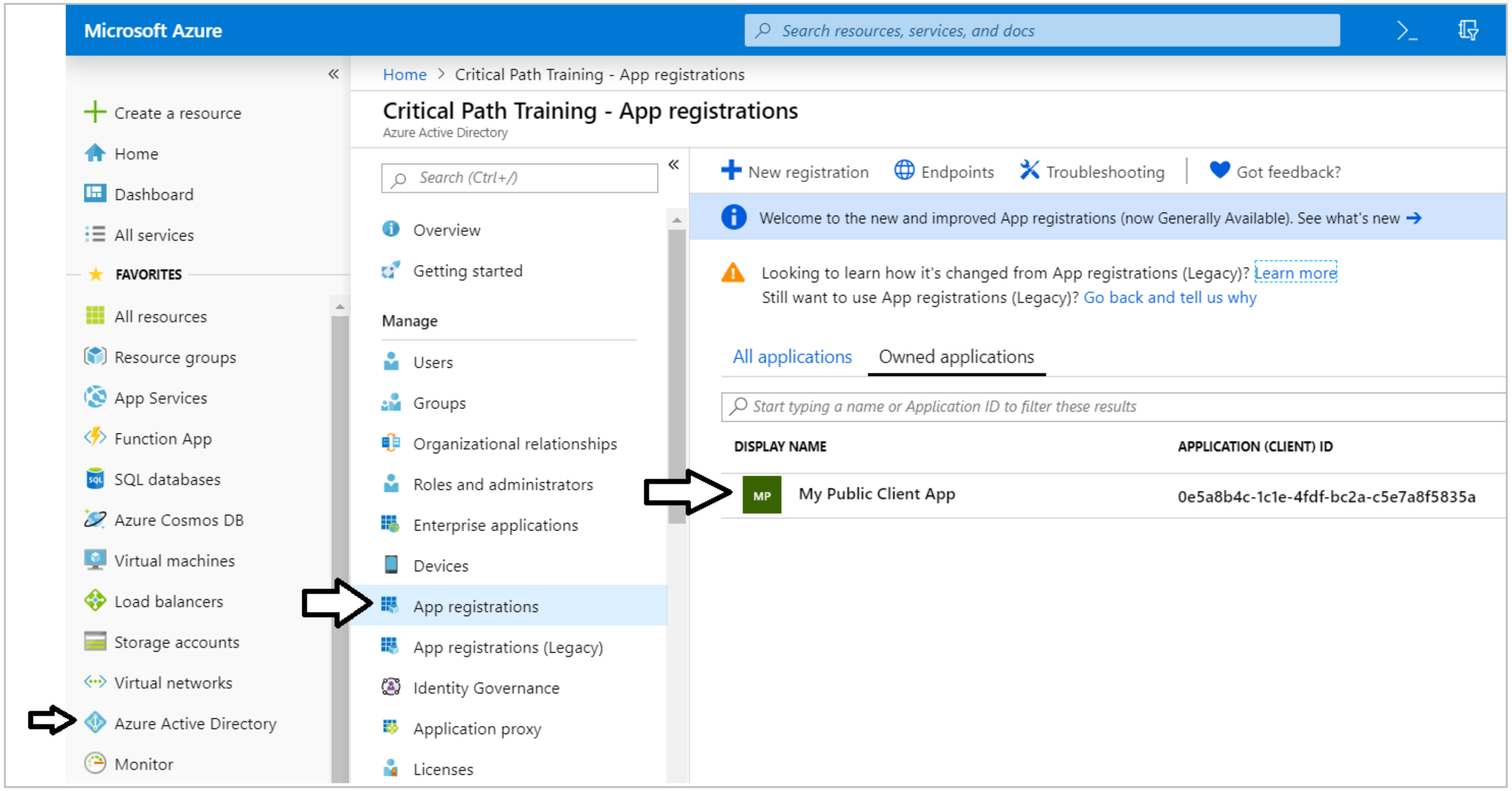
- **User Password Credential Flow** *(public client)*
  - Used in Native clients to obtain access code
  - Requires passing user name and password across network
- **Device Code Flow** *(public client)*
  - New style of authentication introduced with Azure AD v2 Endpoint
- **Client Credentials Flow**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

Agenda

- ✓ Understanding OAuth 2.0 and OpenID Connect
- ➢ Creating & Configuring Azure AD Applications
- Acquiring Access Tokens with MSAL
- Implementing Implicit Flow
- Implementing Authorization Code Flow

The Azure Portal

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



Microsoft Azure

Search resources, services, and docs

Home > Critical Path Training - App registrations

Critical Path Training - App registrations
Azure Active Directory

New registration | Endpoints | Troubleshooting | Got feedback?

Welcome to the new and improved App registrations (now Generally Available). See what's new →

Looking to learn how it's changed from App registrations (Legacy)? Learn more
Still want to use App registrations (Legacy)? Go back and tell us why

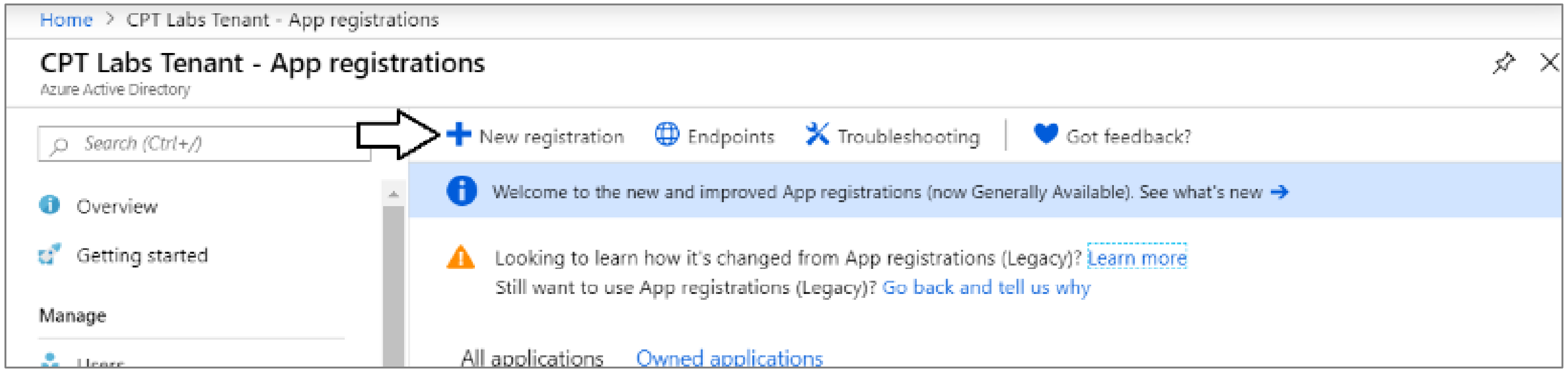
All applications | Owned applications

Start typing a name or Application ID to filter these results

DISPLAY NAME	APPLICATION (CLIENT) ID
MP My Public Client App	0e5a8b4c-1c1e-4fdf-bc2a-c5e7a8f5835a

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 > CPT Labs Tenant - App registrations

CPT Labs Tenant - App registrations
Azure Active Directory

Welcome to the new and improved App registrations (now Generally Available). See what's new →

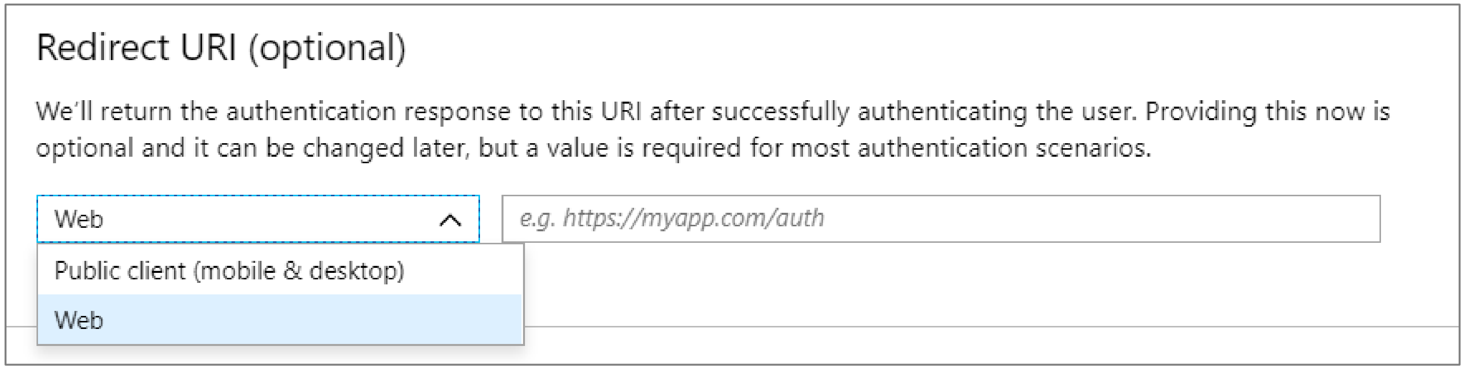
Looking to learn how it's changed from App registrations (Legacy)? Learn more
Still want to use App registrations (Legacy)? Go back and tell us why

Questions To Ask When Creating Application

- Where are my users?
  - Inside a single tenant
  - Inside any Microsoft 365 tenant
  - Inside a Microsoft 365 tenant or
- Where does the application run? Can it keep a secret?
  - Pubic client versus Confidential client
- Should the app work on behalf of a user or work as itself?
  - Should access tokens be created as user tokens or app-only tokens
- When and how should permissions be requested and granted?
  - when should the app ask the user for permissions to a resource

Application Types

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

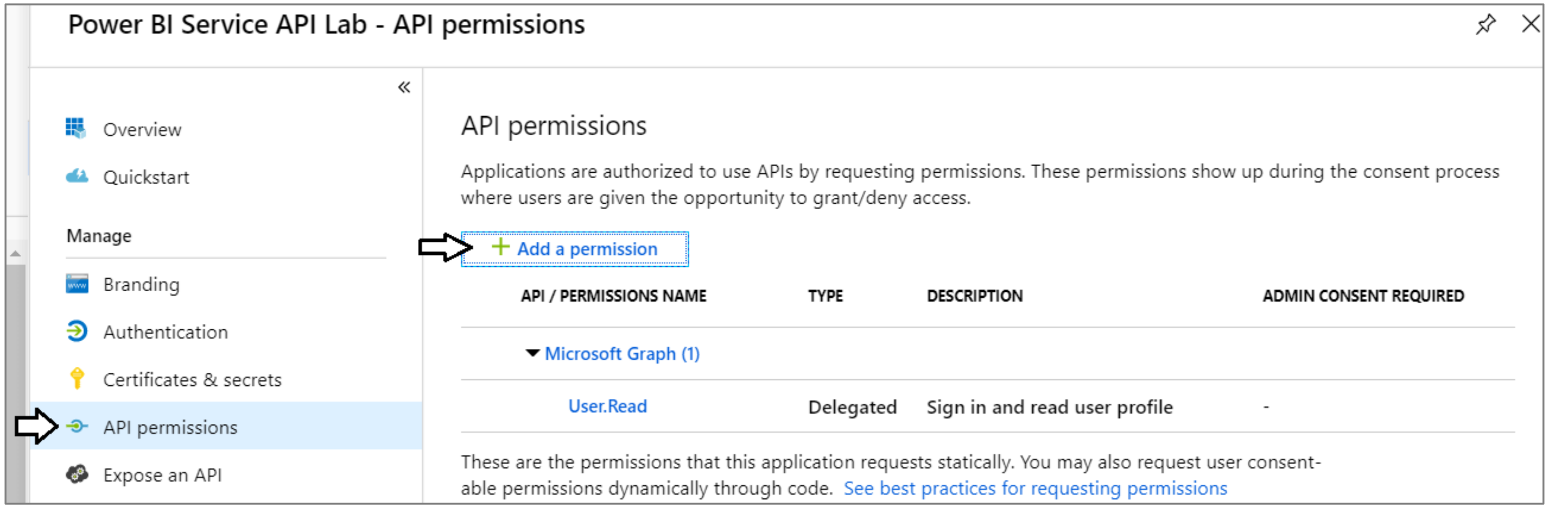


Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Configuring Required Permissions

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



Power BI Service API Lab - API permissions

API permissions

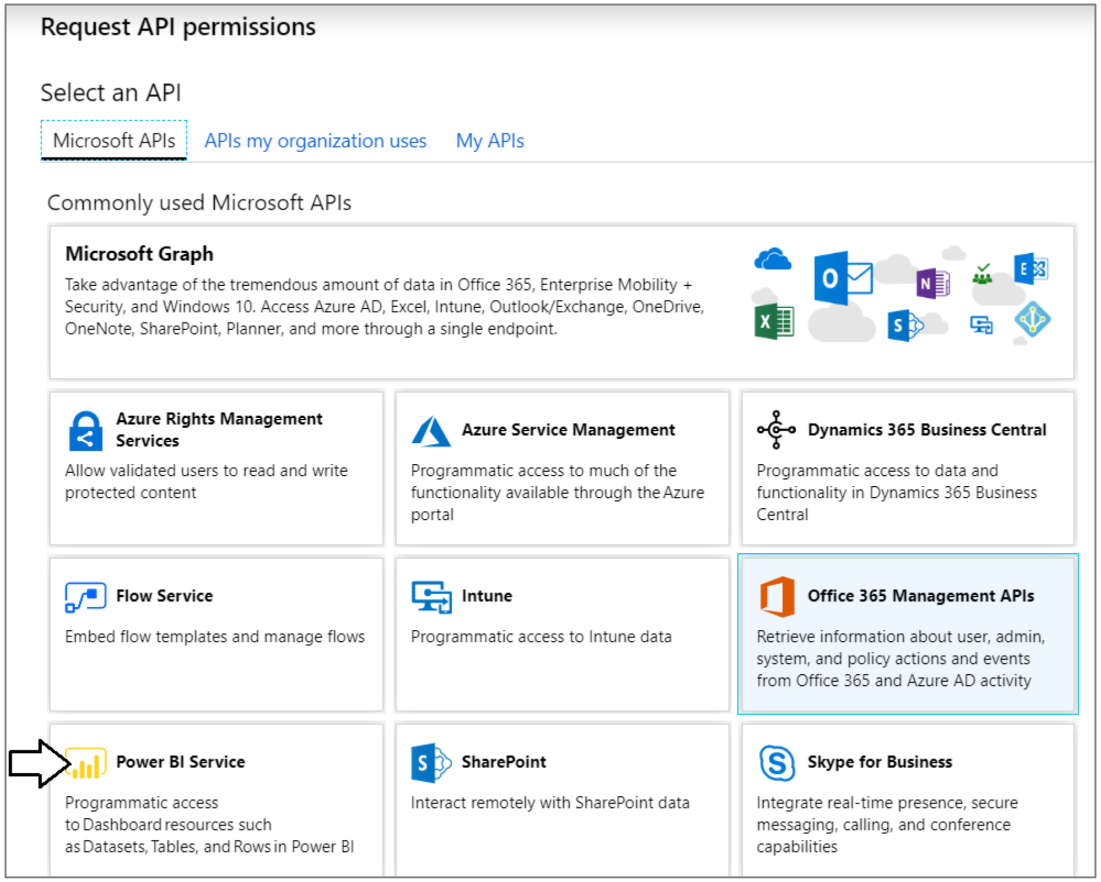
Applications are authorized to use APIs by requesting permissions. These permissions show up during the consent process where users are given the opportunity to grant/deny access.

API / PERMISSIONS NAME	TYPE	DESCRIPTION	ADMIN CONSENT REQUIRED
Microsoft Graph (1)			
User.Read	Delegated	Sign in and read user profile	-

These are the permissions that this application requests statically. You may also request user consent-able permissions dynamically through code. See best practices for requesting permissions

Choosing an API

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



Request API permissions

Select an API

Microsoft APIs | APIs my organization uses | My APIs

Commonly used Microsoft APIs

Microsoft Graph
Take advantage of the tremendous amount of data in Office 365, Enterprise Mobility + Security, and Windows 10. Access Azure AD, Excel, Intune, Outlook/Exchange, OneDrive, OneNote, SharePoint, Planner, and more through a single endpoint.

Azure Rights Management Services
Allow validated users to read and write protected content

Azure Service Management
Programmatic access to much of the functionality available through the Azure portal

Dynamics 365 Business Central
Programmatic access to data and functionality in Dynamics 365 Business Central

Flow Service
Embed flow templates and manage flows

Intune
Programmatic access to Intune data

Office 365 Management APIs
Retrieve information about user, admin, system, and policy actions and events from Office 365 and Azure AD activity

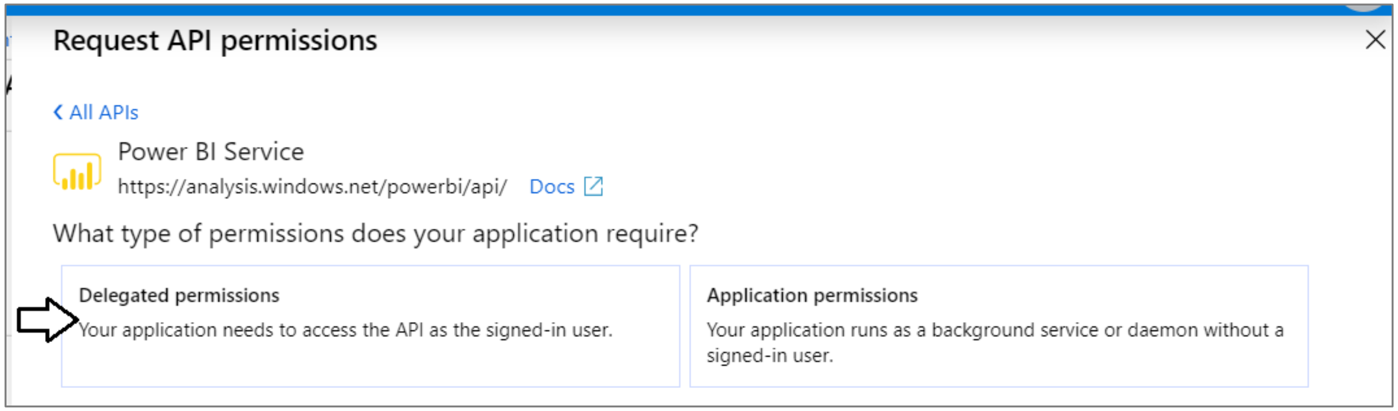
Power BI Service
Programmatic access to Dashboard resources such as Datasets, Tables, and Rows in Power BI

SharePoint
Interact remotely with SharePoint data

Skype for Business
Integrate real-time presence, secure messaging, calling, and conference capabilities

Delegated Permissions vs Application Permissions

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



Request API permissions

< All APIs

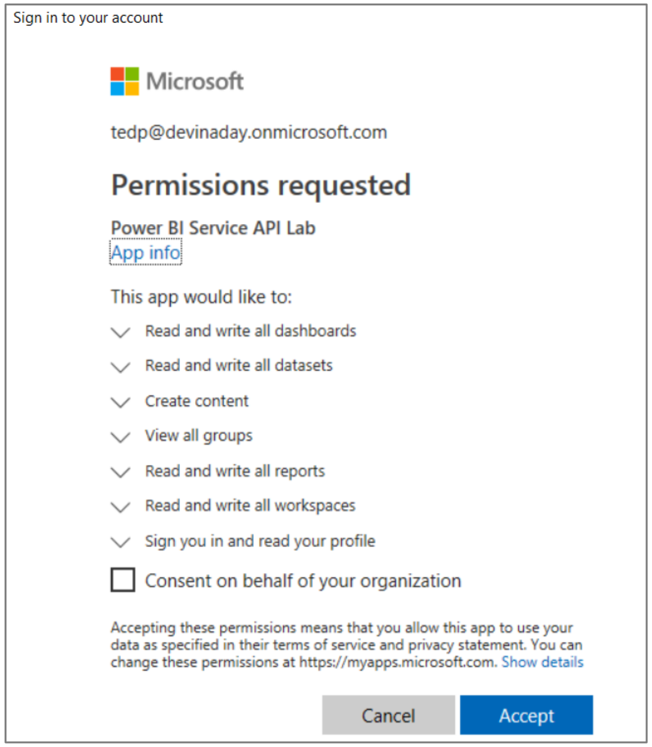
Power BI Service
https://analysis.windows.net/powerbi/api/ Docs

What type of permissions does your application require?

Delegated permissions	Application permissions
Your application needs to access the API as the signed-in user.	Your application runs as a background service or daemon without a signed-in user.

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


Sign in to your account

Microsoft

tedp@devinaday.onmicrosoft.com

Permissions requested

Power BI Service API Lab
App info

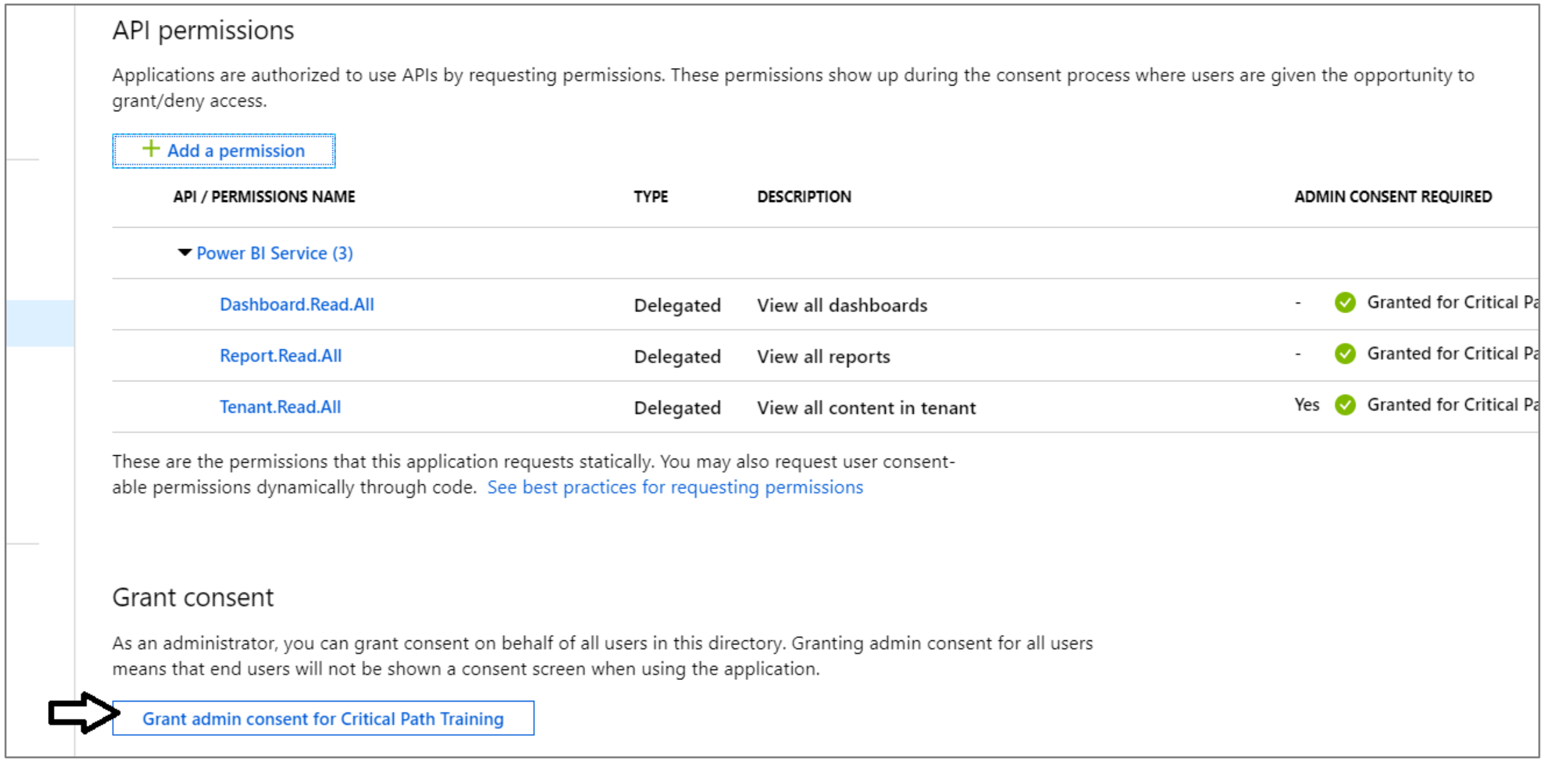
This app would like to:
- Read and write all dashboards
- Read and write all datasets
- Create content
- View all groups
- Read and write all reports
- Read and write all workspaces
- Sign you in and read your profile

☐ Consent on behalf of your organization

Accepting these permissions means that you allow this app to use your data as specified in their terms of service and privacy statement. You can change these permissions at https://myapps.microsoft.com. Show details

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



API permissions

Applications are authorized to use APIs by requesting permissions. These permissions show up during the consent process where users are given the opportunity to grant/deny access.

API / PERMISSIONS NAME	TYPE	DESCRIPTION	ADMIN CONSENT REQUIRED	
▾ Power BI Service (3)				
Dashboard.Read.All	Delegated	View all dashboards	-	✓ Granted for Critical Pa
Report.Read.All	Delegated	View all reports	-	✓ Granted for Critical Pa
Tenant.Read.All	Delegated	View all content in tenant	Yes	✓ Granted for Critical Pa

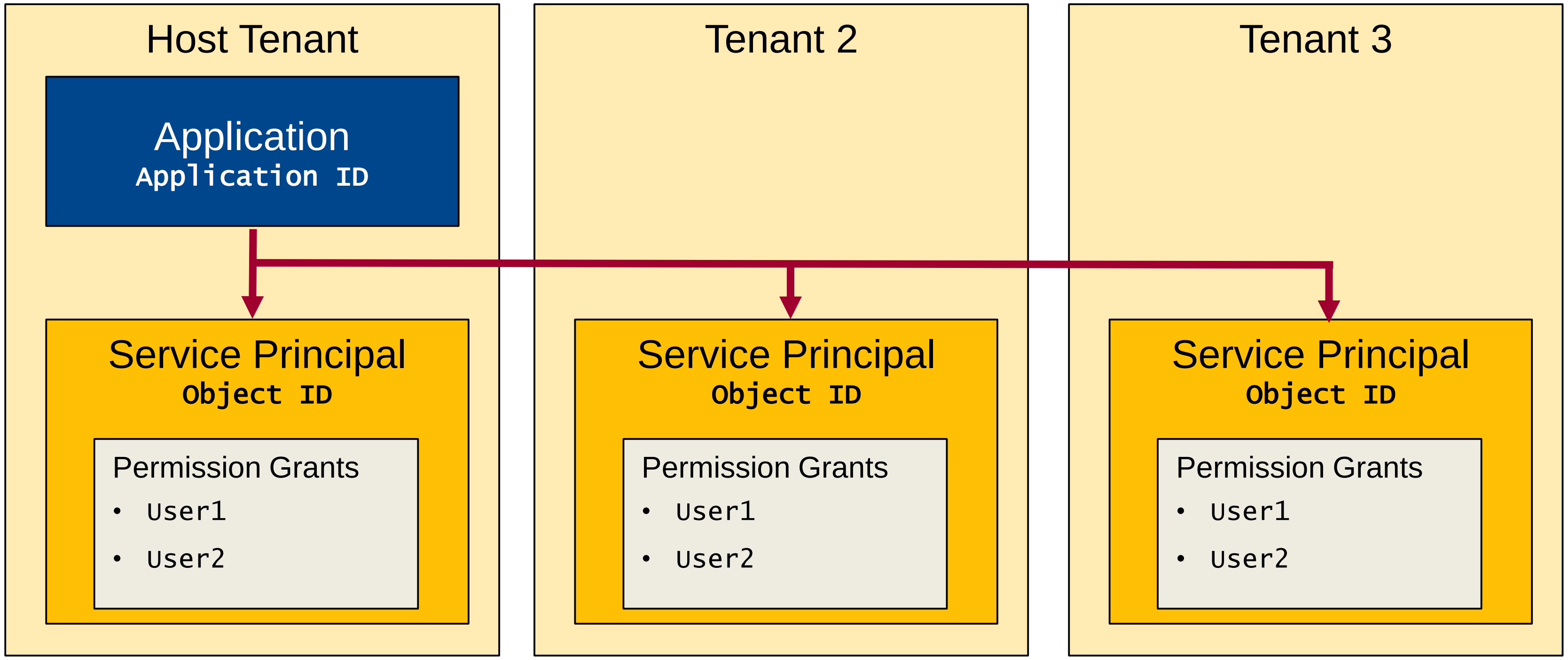
These are the permissions that this application requests statically. You may also request user consent-able permissions dynamically through code. See best practices for requesting permissions

Grant consent

As an administrator, you can grant consent on behalf of all users in this directory. Granting admin consent for all users means that end users will not be shown a consent screen when using the application.

Applications and Service Principals

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



Applications versus Service Principle

- Application defined once across all tenants
  - Identified using global Application ID
  - Application object lives in tenant where it’s registered
  - Defines auth type, secrets and required permissions
- Service principal created once per tenant
  - Identified using tenant-specific Object ID
  - Used to track permission grants for user consent

Registering AAD Apps with PowerShell

```powershell
$authResult = Connect-AzureAD

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

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

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

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

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

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

Configuring Delegated Permissions

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

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

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

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

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

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

DEMO

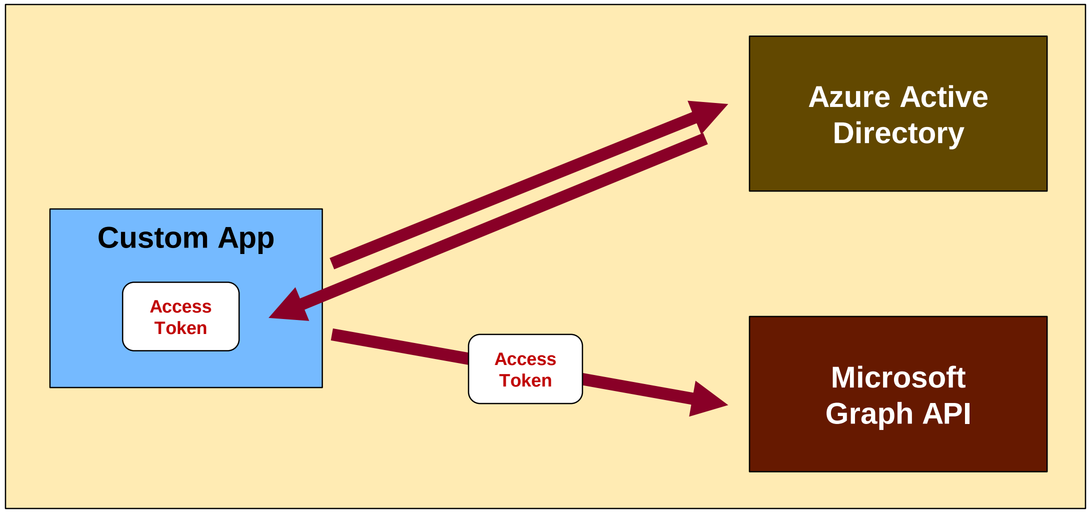
Creating an AAD Application

Agenda

- ✓ Understanding OAuth 2.0 and OpenID Connect
- ✓ Creating & Configuring Azure AD Applications
- ➢ Acquiring Access Tokens with MSAL
- Implementing Implicit Flow
- Implementing Authorization Code Flow

Authenticating with Azure AD

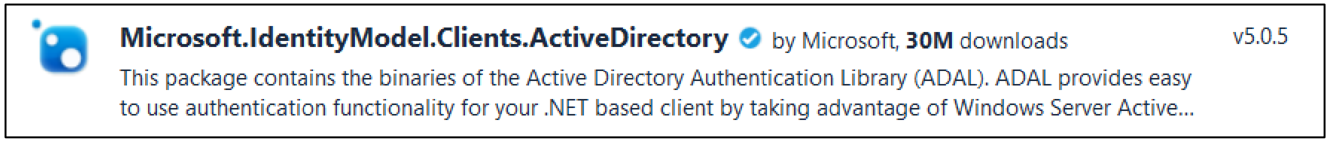
- Custom applications must authenticate with Azure AD
  - Your code implements an authentication flow to obtain access token
  - Access token must be passed when calling resource (e.g. Microsoft Graph API)



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

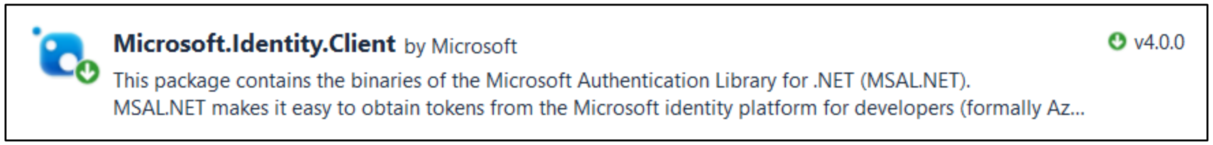
Azure AD Endpoints and Libraries

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



Microsoft.IdentityModel.Clients.ActiveDirectory by Microsoft, **30M** downloads v5.0.5
This package contains the binaries of the Active Directory Authentication Library (ADAL). ADAL provides easy to use authentication functionality for your .NET based client by taking advantage of Windows Server Active...

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

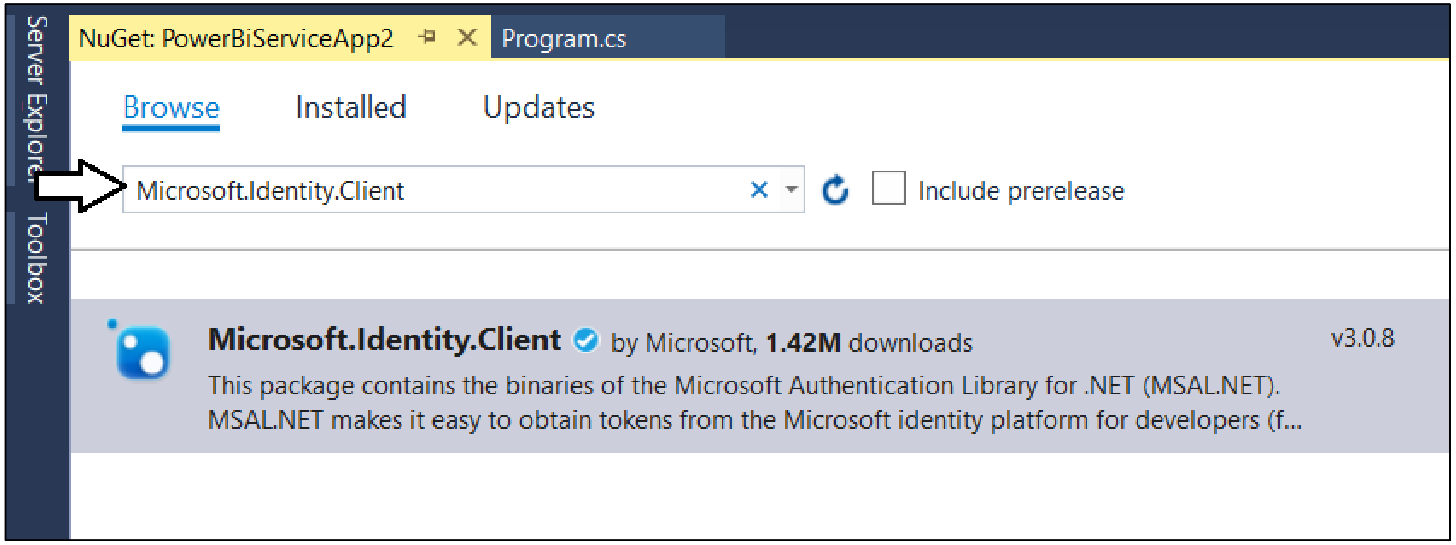


Microsoft.Identity.Client by Microsoft v4.0.0
This package contains the binaries of the Microsoft Authentication Library for .NET (MSAL.NET). MSAL.NET makes it easy to obtain tokens from the Microsoft identity platform for developers (formally Az...

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

Microsoft Authentication Library (.NET)

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



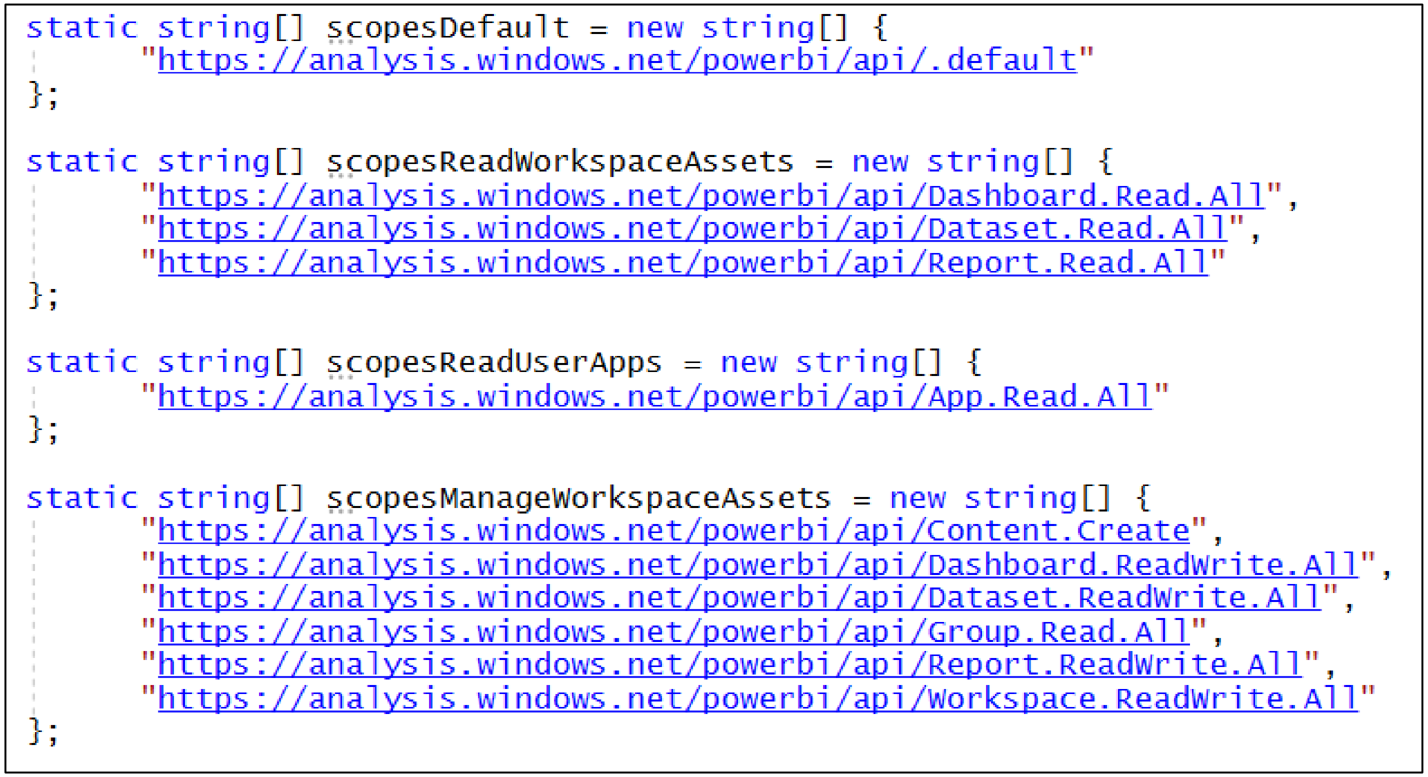
Microsoft.Identity.Client

Microsoft.Identity.Client by Microsoft, **1.42M** downloads v3.0.8

This package contains the binaries of the Microsoft Authentication Library for .NET (MSAL.NET). MSAL.NET makes it easy to obtain tokens from the Microsoft identity platform for developers (f...

Power BI Service API Scopes

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



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

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

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

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

Interactive Access Token Acquisition

Using MSAL with public client application

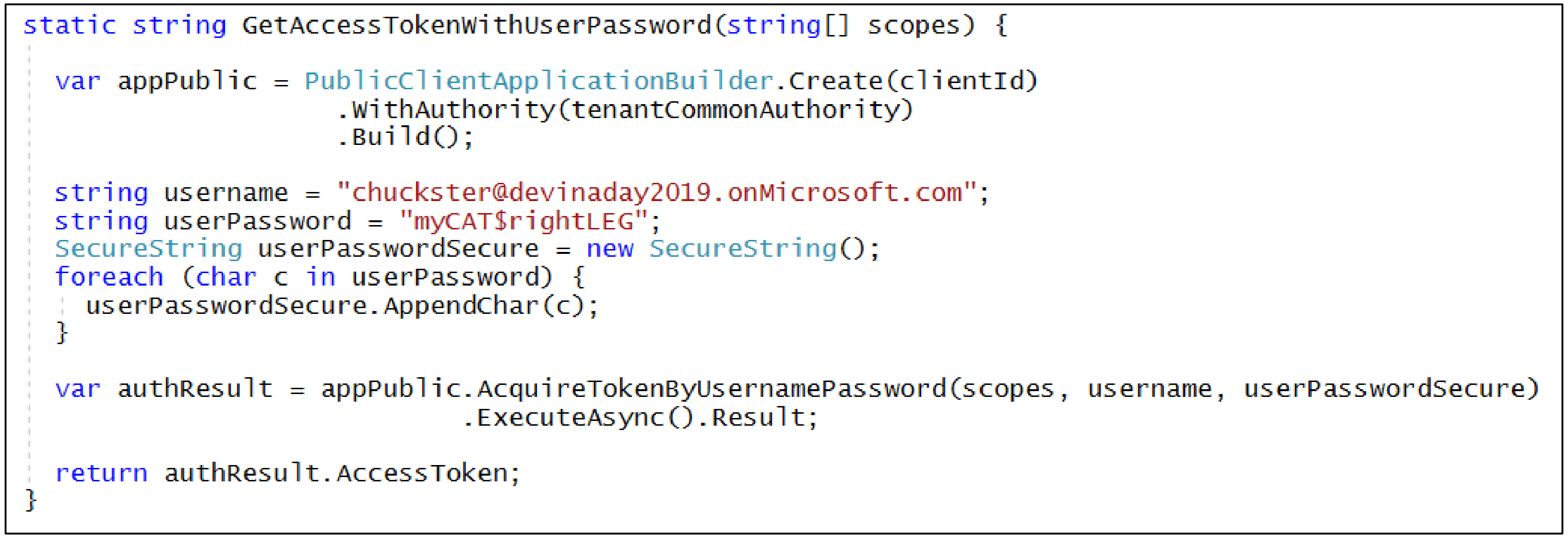
- Flow implemented using **`PublicClientApplication`** object
  - Created using **`PublicClientApplicationBuilder`** object
  - Requires passing redirect URI
  - You can control prompting behavior

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

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

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

  return authResult.AccessToken;
}
```

User Credential Password Flow

Using MSAL with public client application

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

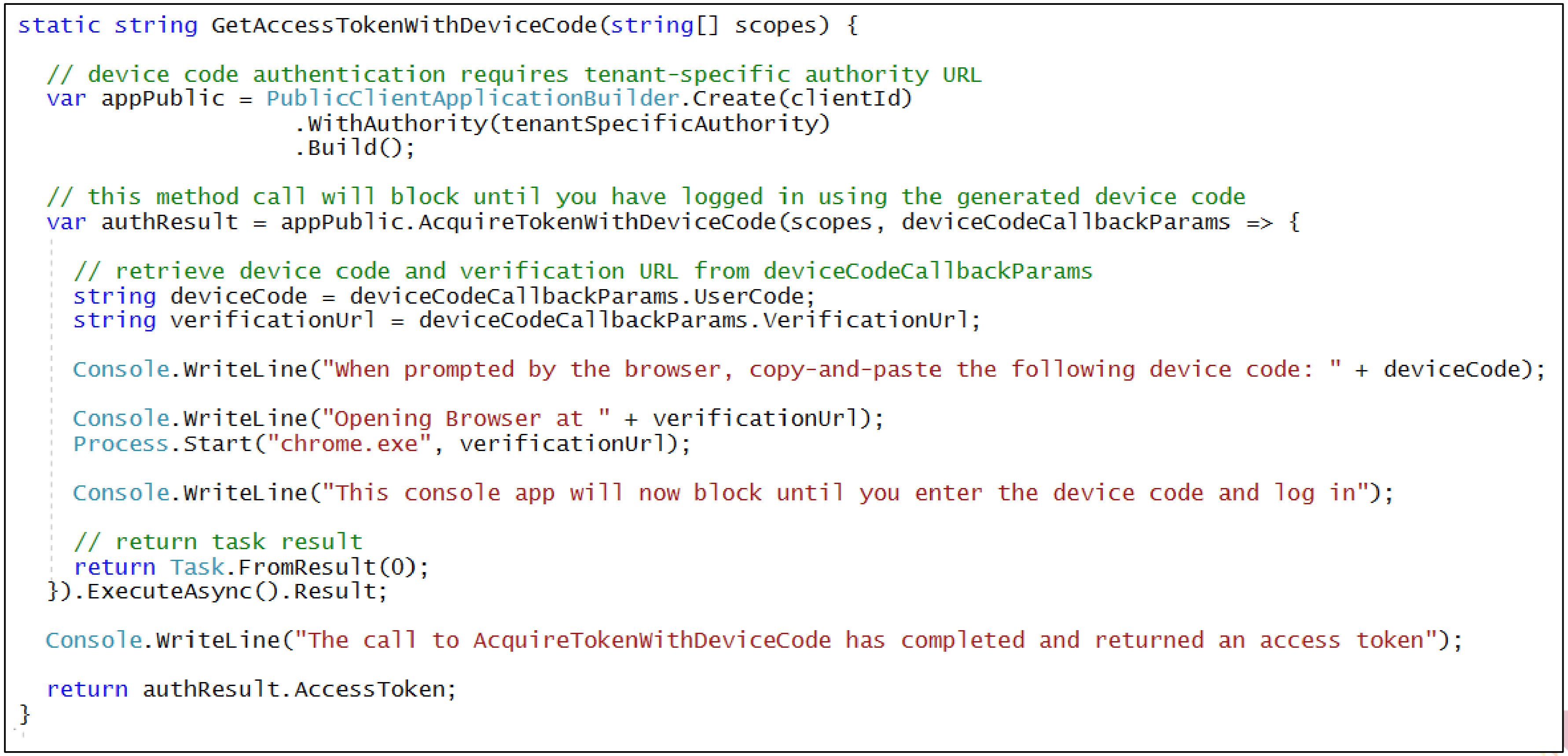


```
static string GetAccessTokenWithUserPassword(string[] scopes) {

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

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

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

  return authResult.AccessToken;
}
```

Device Code Flow

Using MSAL with public client application

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

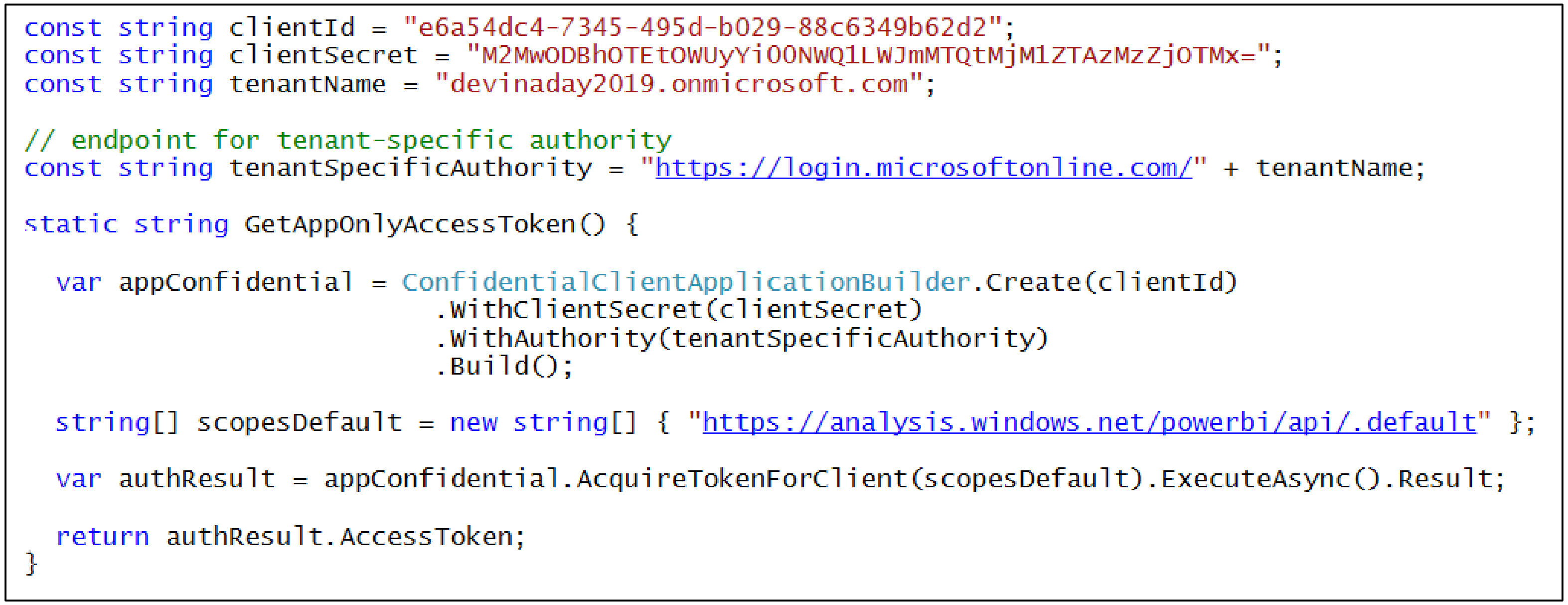


```
static string GetAccessTokenWithDeviceCode(string[] scopes) {

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

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

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

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

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

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

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

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

  return authResult.AccessToken;
}
```

Client Credentials Flow

Using MSAL with confidential client application

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



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

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

static string GetAppOnlyAccessToken() {

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

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

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

  return authResult.AccessToken;
}
```

Agenda

- ✓ Understanding OAuth 2.0 and OpenID Connect
- ✓ Creating & Configuring Azure AD Applications
- ✓ Acquiring Access Tokens with MSAL
- ➢ Implementing Implicit Flow
- • Implementing Authorization Code Flow

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

Agenda

- ✓ Understanding OAuth 2.0 and OpenID Connect
- ✓ Creating & Configuring Azure AD Applications
- ✓ Acquiring Access Tokens with MSAL
- ✓ Implementing Implicit Flow
- ➢ Implementing Authorization Code Flow

Authorization Code Grant Flow

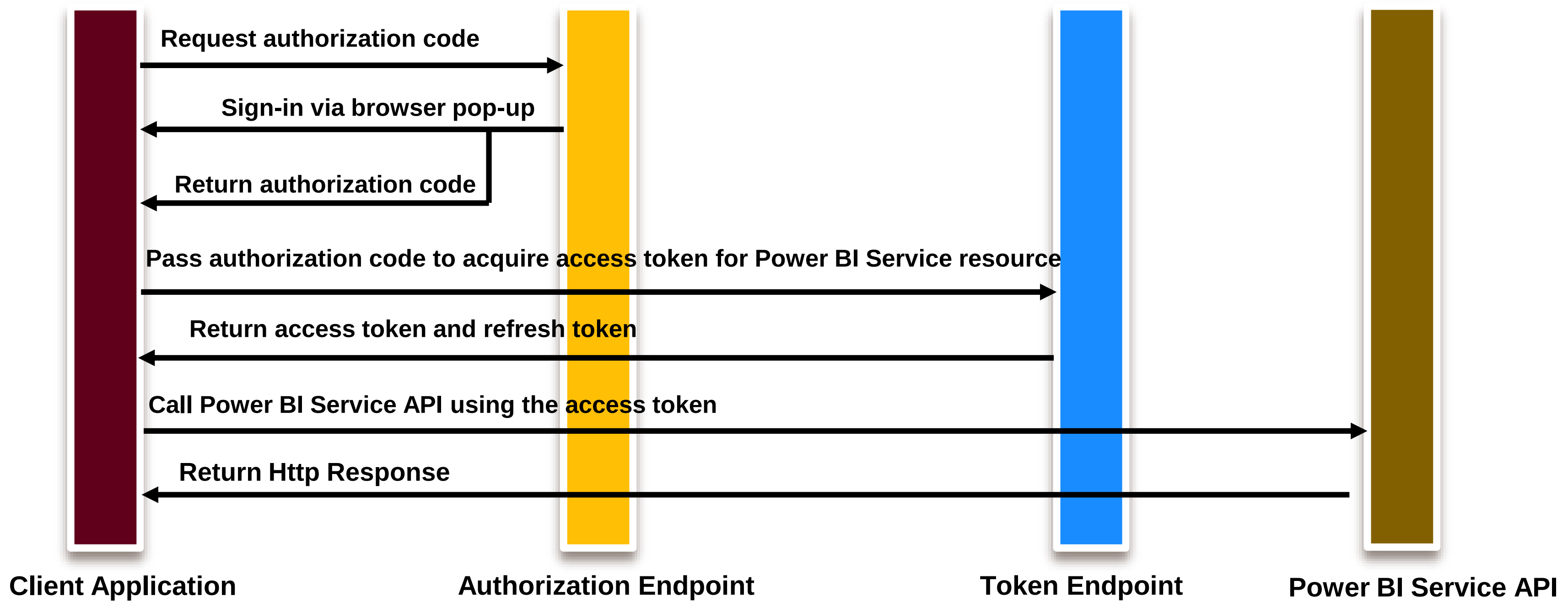
- Provides Highest Levels of Security
  - User credentials never seen by client
  - Access token passed to client with Reply URL
  - Access token not passed through user agent
- Refresh tokens used to get new access tokens
  - Access token lifetime is about 1 hour
  - Refresh token lifetime is 14 days
  - AAD supports multi-resource refresh tokens (MRRTs)

Authorization Code Grant Flow Example

- **Sign-on URL**
  - Development: **https://localhost:44300/**
  - Production: **https://www.MyDomain.com/**
- **Reply URL**
  - Development: **https://localhost:44300/AcceptDirect**
  - Production: **https://www.MyDomain.com/AcceptDirect**
- **Client ID**
  - GUID-based identifier for a specific AAD application
  - **33d561fb-59a7-4817-bddf-2117193d62e0**
- **Key** (aka Client Secret)
  - Key that acts as a secret password between Azure AD and application
  - **ouWdhd2LxDl0Pcu2SKlujEiQ5GmSbKRbBM24nETb5dw=**

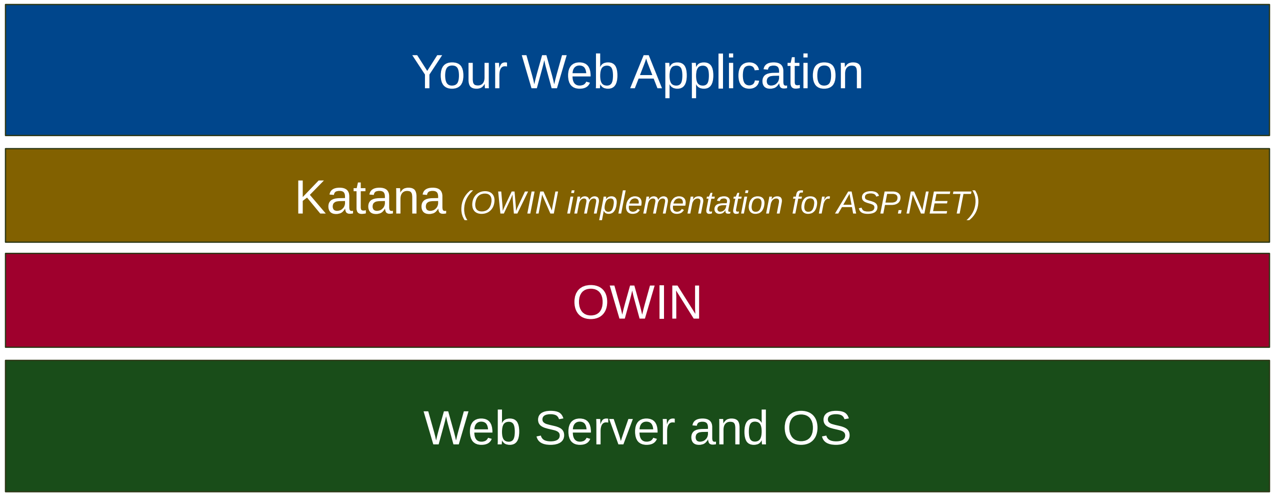
Authorization Code Grant Flow

- 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


Request authorization code
Sign-in via browser pop-up
Return authorization code
Pass authorization code to acquire access token for Power BI Service resource
Return access token and refresh token
Call Power BI Service API using the access token
Return Http Response
Client Application
Authorization Endpoint
Token Endpoint
Power BI Service API

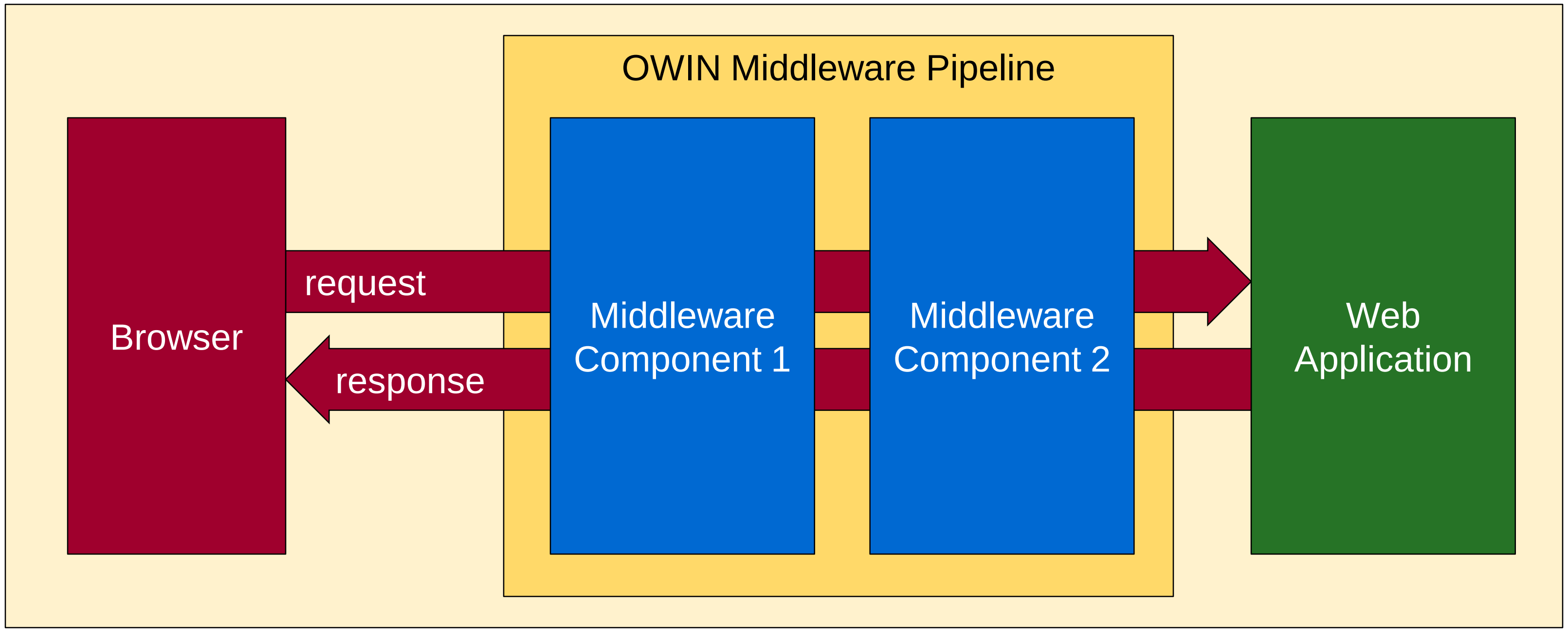
Open Web Interfaces for NET (OWIN)

- 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



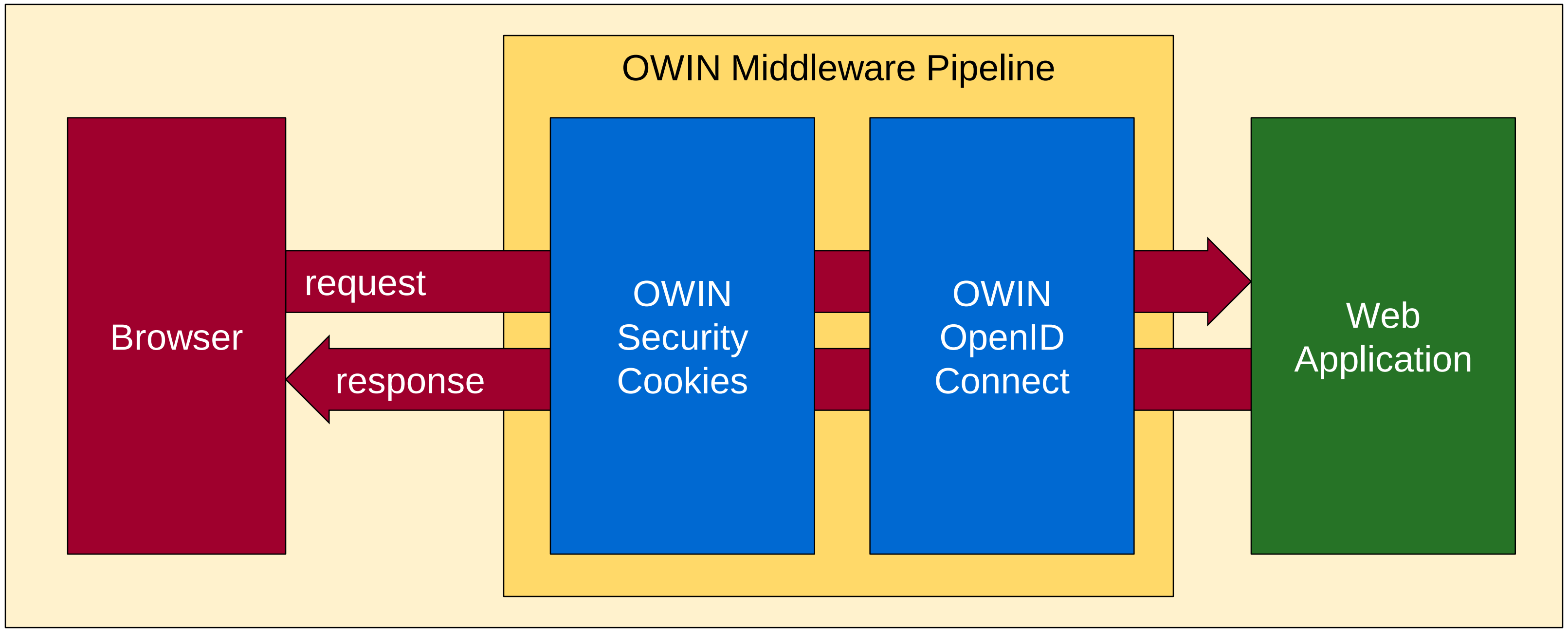
OWIN Middleware Modules

- 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



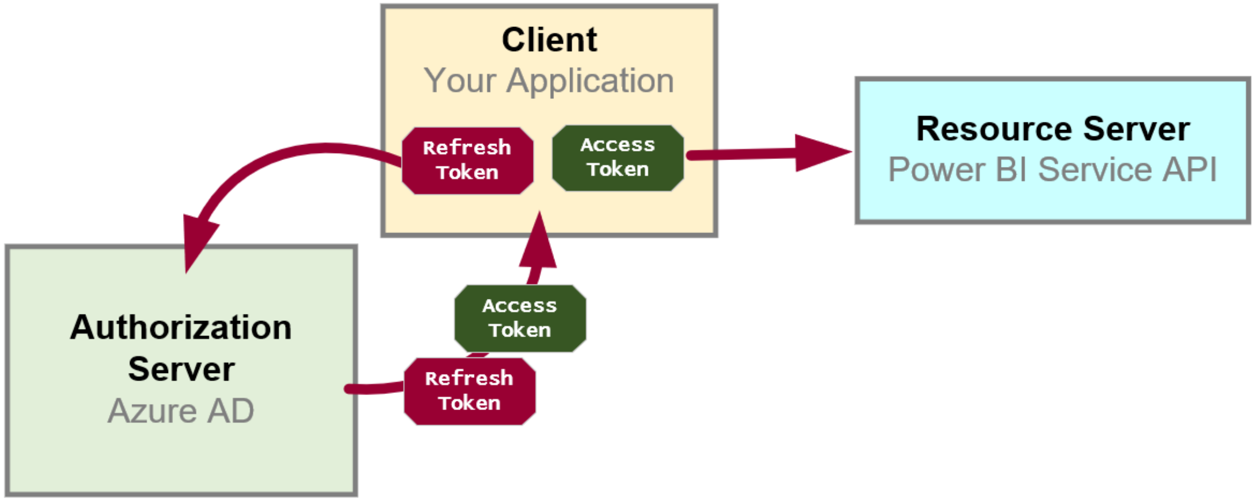
OWIN OpenID Connect Module

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



Token Caching and 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
  - ADAL and MSAL both offer built-in support to mange token caching



DEMO

Using ADAL in a Web Client

Summary of OAuth Client Types

	Web Client SPA	Hybrid Native Client	Web Application Client	Web Service Client
Client Type	Public	Public or Confidential	Confidential	Confidential
Verifiable Reply URL	Yes	No	Yes	Yes
Authenticates Client	No	It Depends	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 + App	Delegated + App	Delegated + App

Summary

- ✓ Understanding OAuth 2.0 and OpenID Connect
- ✓ Creating & Configuring Azure AD Applications
- ✓ Acquiring Access Tokens with MSAL
- ✓ Implementing Implicit Flow
- ✓ Implementing Authorization Code Flow