Microsoft

AZ-204T00A

Learning Path 06: Implement user authentication and authorization

Agenda

- Explore the Microsoft identity platform
- Implement authentication by using the Microsoft Authentication Library
- Implement shared access signatures
- Explore Microsoft Graph

Module 1: Explore the Microsoft identity platform

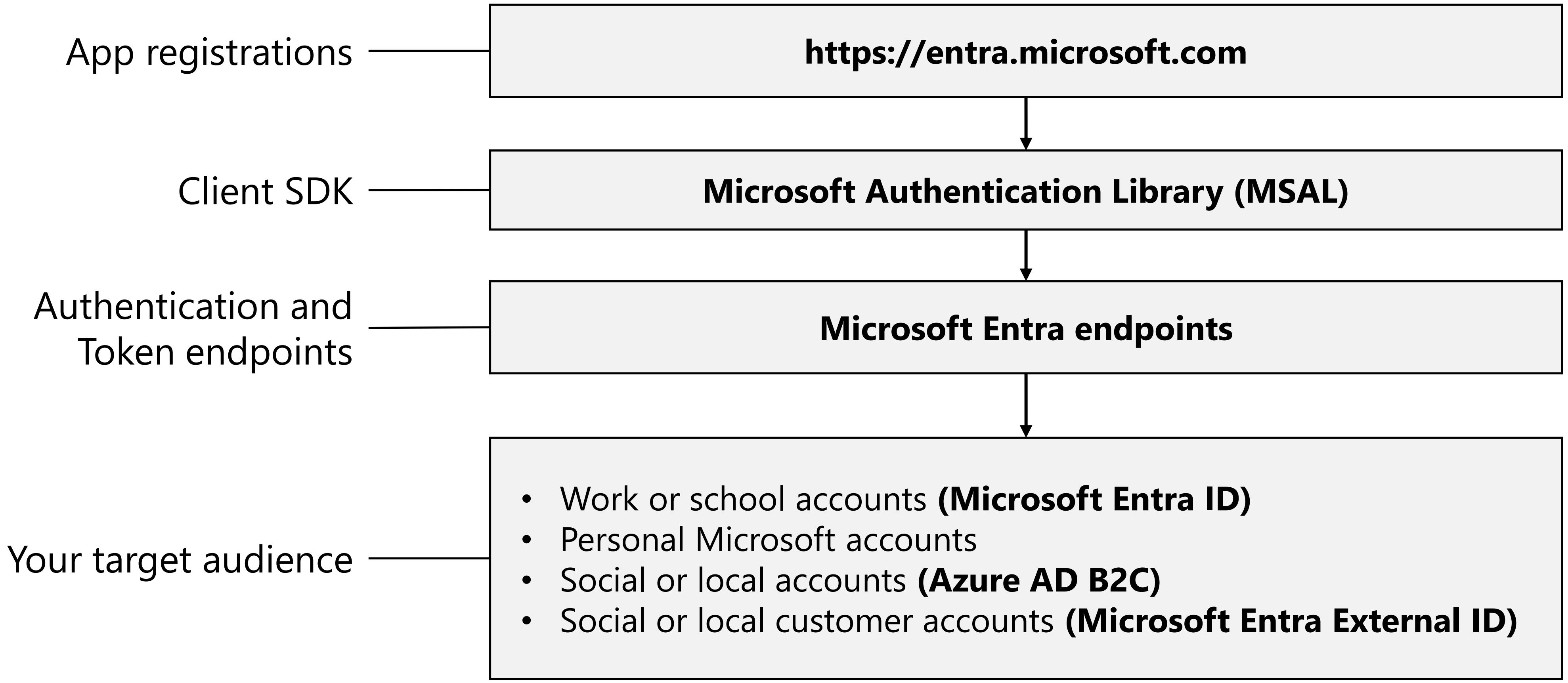
Learning objectives

- Identify the components of the Microsoft identity platform
- Describe the three types of service principals and how they relate to application objects
- Explain how permissions and user consent operate, and how conditional access impacts your application

Introduction

- The Microsoft identity platform is a set of tools that includes authentication service, open-source libraries, and application management tools.
- Helps you build applications your users and customers can sign in to using their Microsoft identities or social accounts.

Explore the Microsoft identity platform


App registrations
https://entra.microsoft.com
Client SDK
Microsoft Authentication Library (MSAL)
Authentication and Token endpoints
Microsoft Entra endpoints
Your target audience

- Work or school accounts **(Microsoft Entra ID)**
- Personal Microsoft accounts
- Social or local accounts **(Azure AD B2C)**
- Social or local customer accounts **(Microsoft Entra External ID)**

Explore service principals

When you register an app in the Azure portal, you choose whether it is:

- Single tenant: only accessible in your tenant
- Multi-tenant: accessible in other tenants

If you register an application in the portal, below objects are automatically created in your home tenant:

- Application object
- Service principal object - Application, Managed identity, Legacy

Relationship between application objects and service principals

An application object has:

- 1:1 relationship with the software application, and
- 1:many relationship with its corresponding service principal object

Discover permissions and consent (1 of 2)

Permission types

The Microsoft identity platform supports two types of permissions:

- *Delegated permissions* are used by apps that have a signed-in user present.
- *Application permissions* are used by apps that run without a signed-in user present

Consent types

There are three consent types:

- Static user consent
- Incremental and Dynamic user consent
- Admin consent

Discover permissions and consent (2 of 2)

In an OpenID Connect or OAuth 2.0 authorization request, an app can request the permissions it needs by using the scope query parameter.

```
GET https://login.microsoftonline.com/common/oauth2/v2.0/authorize?
client_id=6731de76-14a6-49ae-97bc-6eba6914391e
&response_type=code
&redirect_uri=http%3A%2F%2Flocalhost%2Fmyapp%2F
&response_mode=query
&scope=
   https%3A%2F%2Fgraph.microsoft.com%2Fcalendars.read%20
   https%3A%2F%2Fgraph.microsoft.com%2Fmail.send
&state=12345
```

Discover conditional access

Conditional Access enables developers to protect services in a multitude of ways including

- Multifactor authentication
- Allowing only Intune enrolled devices to access specific services
- Restricting user locations and IP ranges

Conditional Access impact on an app

Specifically, the following scenarios require code to handle Conditional Access challenges:

- Apps performing the on-behalf-of flow
- Apps accessing multiple services/resources
- Single-page apps using MSAL.js
- Web apps calling a resource

Summary and knowledge check

In this module, you learned how to:

- Identify the components of the Microsoft identity platform
- Describe the three types of service principals and how they relate to application objects
- Explain how permissions and user consent operate, and how conditional access impacts your application

1. Which of the types of permissions supported by the Microsoft identity platform is used by apps that have a signed-in user present?
2. What app scenarios require code to handle Conditional Access challenges?

Module 2: Implement authentication by using the Microsoft Authentication Library

Learning objectives

- Explain the benefits of using MSAL and the application types and scenarios it supports
- Instantiate both public and confidential client apps from code
- Register an app with the Microsoft identity platform
- Create an app that retrieves a token by using the MSAL.NET library

Introduction

- The Microsoft Authentication Library (MSAL) enables developers to acquire tokens from the Microsoft identity platform to authenticate users and access secured web APIs.

Explore the Microsoft Authentication Library (1 of 2)

- The Microsoft Authentication Library (MSAL) can be used to provide secure access to Microsoft Graph, other Microsoft APIs, third-party web APIs, or your own web API.
- MSAL supports many different application architectures and platforms including .NET, JavaScript, Java, Python, Android, and iOS.
- Application types and scenarios:
  - web applications
  - web APIs
  - single-page apps (JavaScript)
  - mobile and native applications
  - daemons and server-side applications

Explore the Microsoft Authentication Library (2 of 2)

Authentication flows

Flow	Description
Authorization code	Native and web apps securely obtain tokens in the name of the user
Client credentials	Service applications run without user interaction
On-behalf-of	The application calls a service/web API, which in turns calls Microsoft Graph
Implicit	Used in browser-based applications
Device code	Enables sign-in to a device by using another device that has a browser
Integrated Windows	Windows computers silently acquire an access token when they are domain joined
Username/password	The application signs in a user by using their username and password

Initialize client applications (1 of 2)

With MSAL.NET 3.x, the recommended way to instantiate an application is by using the `PublicClientApplicationBuilder` and `ConfidentialClientApplicationBuilder` application builders.

```
//Public client initialization

IPublicClientApplication app = PublicClientApplicationBuilder.Create(clientId).Build();


//Confidential client initialization

string redirectUri = "https://myapp.azurewebsites.net";
IConfidentialClientApplication app = ConfidentialClientApplicationBuilder.Create(clientId)
    .WithClientSecret(clientSecret)
    .WithRedirectUri(redirectUri )
    .Build();
```

Initialize client applications (2 of 2)

Modifier	Description
`.WithAuthority() 7 overrides`	Sets the application default authority to a Microsoft Entra authority, with the possibility of choosing the Azure Cloud, the audience, the tenant (tenant ID or domain name), or providing directly the authority URI.
`.WithTenantId(string tenantId)`	Overrides the tenant ID, or the tenant description.
`.WithClientId(string)`	Overrides the client ID.
`.WithRedirectUri(string redirectUri)`	Overrides the default redirect URI. In the case of public client applications, this will be useful for scenarios requiring a broker.
`.WithComponent(string)`	Sets the name of the library using MSAL.NET (for telemetry reasons).
`.WithDebugLoggingCallback()`	If called, the application will call `Debug.Write` simply enabling debugging traces.
`.WithLogging()`	If called, the application will call a callback with debugging traces.
`.WithTelemetry(TelemetryCallback telemetryCallback)`	Sets the delegate used to send telemetry.

Exercise: Implement interactive authentication by using MSAL.NET

In this exercise you learn how to use MSAL.NET to acquire a token interactively in a console application.

Objectives

- Register a new application
- Build the console app
- Review completed app
- Run the application to retrieve the token
- Clean up resources

Summary and knowledge check

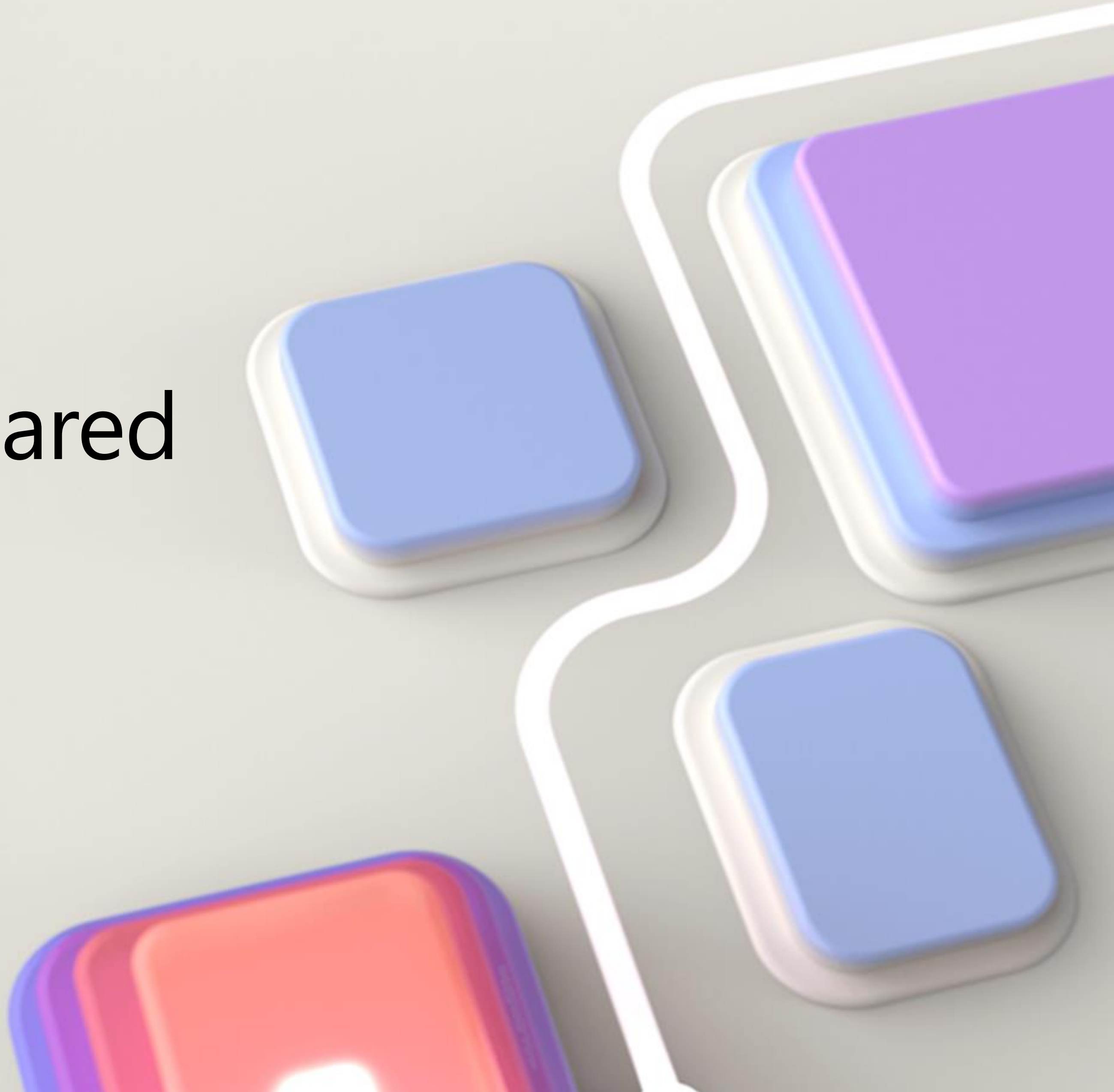
In this module, you learned how to:

- Explain the benefits of using MSAL and the application types and scenarios it supports
- Instantiate both public and confidential client apps from code
- Register an app with the Microsoft identity platform
- Create an app that retrieves a token by using the MSAL.NET

1. Which MSAL library supports single-page web apps?

2. What is the purpose of using `PublicClientApplicationBuilder` class in MSAL.NET?

Module 3: Implement shared access signatures



Learning objectives

- Identify the three types of shared access signatures
- Explain when to implement shared access signatures
- Create a stored access policy

Introduction

- A shared access signature (SAS) is a signed URI that points to one or more storage resources and includes a token that contains a special set of query parameters.
- The token indicates how the resources may be accessed by the client.

Discover shared access signatures (1 of 2)

Types of shared access signatures

- User delegation SAS
- Service SAS
- Account SAS

Best practices

- Always use HTTPS
- The most secure SAS is a user delegation SAS
- Set expiration time to smallest useful time
- Apply the rule of minimum-required privileges
- SAS isn't always the correct solution

Discover shared access signatures (2 of 2)

Use SAS to access data

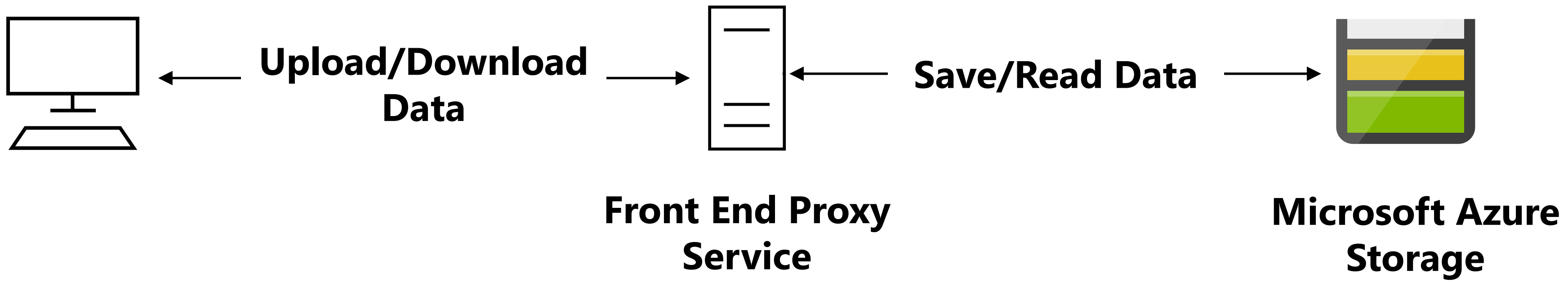
- **URI:** `https://medicalrecords.blob.core.windows.net/patient-images/patient-116139-nq8z7f.jpg?`
- **SAS token:** `sp=r&st=2020-01-20T11:42:32Z&se=2020-01-20T19:42:32Z&spr=https&sv=2019-02-02&sr=b&sig=<...>`

Component	Description
`sp=r`	Controls the access rights. The values can be **a** for add, **c** for create, **d** for delete, **l** for list, **r** for read, or **w** for write.
`st=2020-01-20T11:42:32Z`	The date and time when access starts.
`se=2020-01-20T19:42:32Z`	The date and time when access ends. This example grants eight hours of access.
`sv=2019-02-02`	The version of the storage API to use.
`sr=b`	The kind of storage being accessed. In this example, b is for blob.
`sig=<...>`	The cryptographic signature.

Choose when to use shared access signatures (1 of 2)

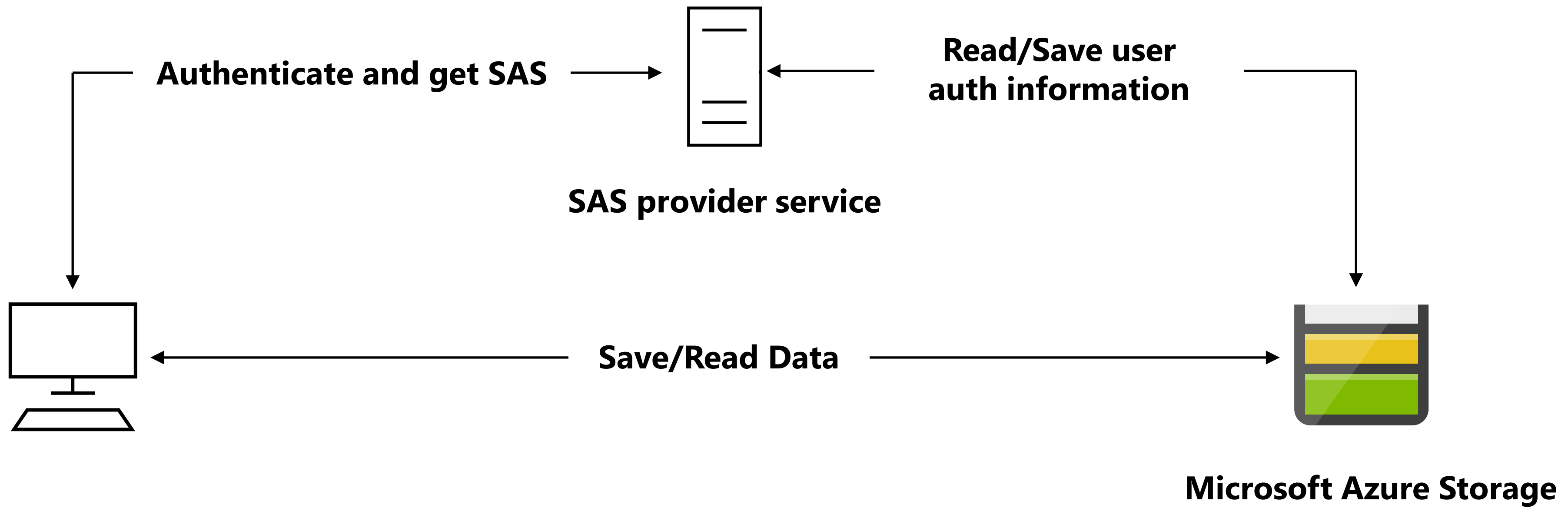
In a scenario where a storage account stores user data, there are two typical design patterns:

- Clients upload and download data via a front-end proxy service, which performs authentication. This front-end proxy service has the advantage of allowing validation of business rules, but for large amounts of data or high-volume transactions, creating a service that can scale to match demand may be expensive or difficult.



Choose when to use shared access signatures (2 of 2)

- A lightweight service authenticates the client as needed and then generates a SAS. Once the client application receives the SAS, they can access storage account resources directly with the permissions defined by the SAS and for the interval allowed by the SAS. The SAS mitigates the need for routing all data through the front-end proxy service.



Explore stored access policies

- A stored access policy provides an additional level of control over service-level shared access signatures (SAS) on the server side.
- To create or modify a stored access policy, call the **Set ACL** operation for the resource with a request body that specifies the terms of the access policy.

```
BlobSignedIdentifier identifier = new
BlobSignedIdentifier
{
    Id = "stored access policy identifier",
    AccessPolicy = new BlobAccessPolicy
    {
        ExpiresOn = DateTimeOffset.UtcNow.AddHours(1),
        Permissions = "rw"
    }
};
```

```
az storage container policy create \
    --name <stored access policy identifier> \
    --container-name <container name> \
    --start <start time UTC datetime> \
    --expiry <expiry time UTC datetime> \
    --permissions <(a),(c),(d),(l),(r),(w) \
    --account-key <storage account key> \
    --account-name <storage account name>
```

Summary and knowledge check

In this module, you learned how to:

- Identify the three types of shared access signatures
- Explain when to implement shared access signatures
- Create a stored access policy

1. What type of shared access signatures (SAS) applies to Blob storage only?
2. What is the most flexible and secure way to use a service or account shared access signature (SAS)?

Module 4: Explore Microsoft Graph

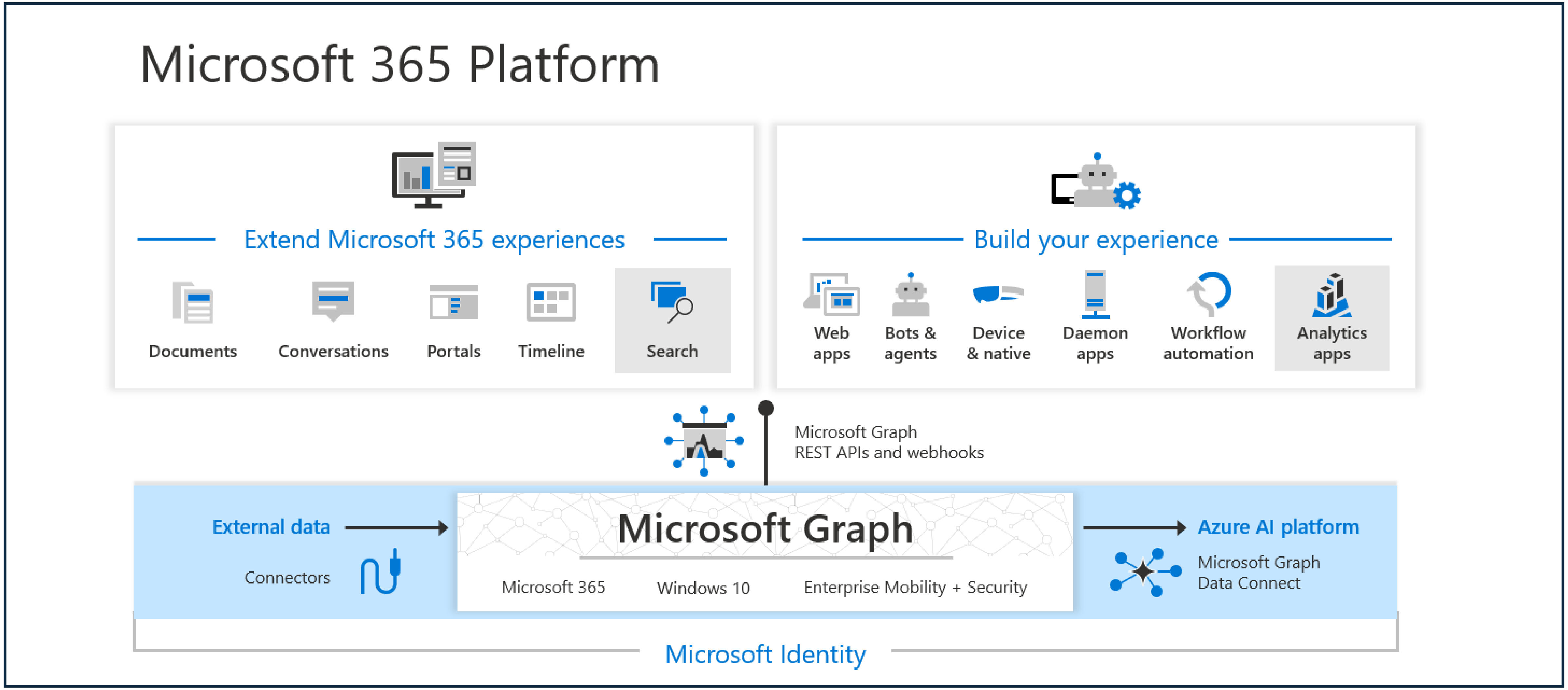
Learning objectives

- Explain the benefits of using Microsoft Graph
- Perform operations on Microsoft Graph by using REST and SDKs
- Apply best practices to help your applications get the most out of Microsoft Graph

Introduction

- Microsoft Graph is the gateway to data and intelligence in Microsoft 365.
- It provides a unified programmability model that you can use to access the tremendous amount of data in Microsoft 365, Windows, and Enterprise Mobility + Security.

Discover Microsoft Graph



Query Microsoft Graph by using REST

Call a REST API method

- {HTTP method} `https://graph.microsoft.com/{version}/{resource}?{query-parameters}`
- HTTP methods (`GET`, `POST`, `PATCH`, `PUT`, `DELETE`)

{version}: Microsoft Graph currently supports two versions `v1.0` and `beta`

{resource}: A resource can be an entity or complex type, commonly defined with properties.

{query-parameters}: Query parameters can be OData system query options, or other strings that a method accepts to customize its response.

Query Microsoft Graph by using SDKs (1 of 4)

Microsoft.Graph

- Object-relational mapping tool for Microsoft Graph
- Contains classes mapped to the RESTful syntax of the Microsoft Graph API

Microsoft.Graph.Core

- Core library for making calls to Microsoft Graph

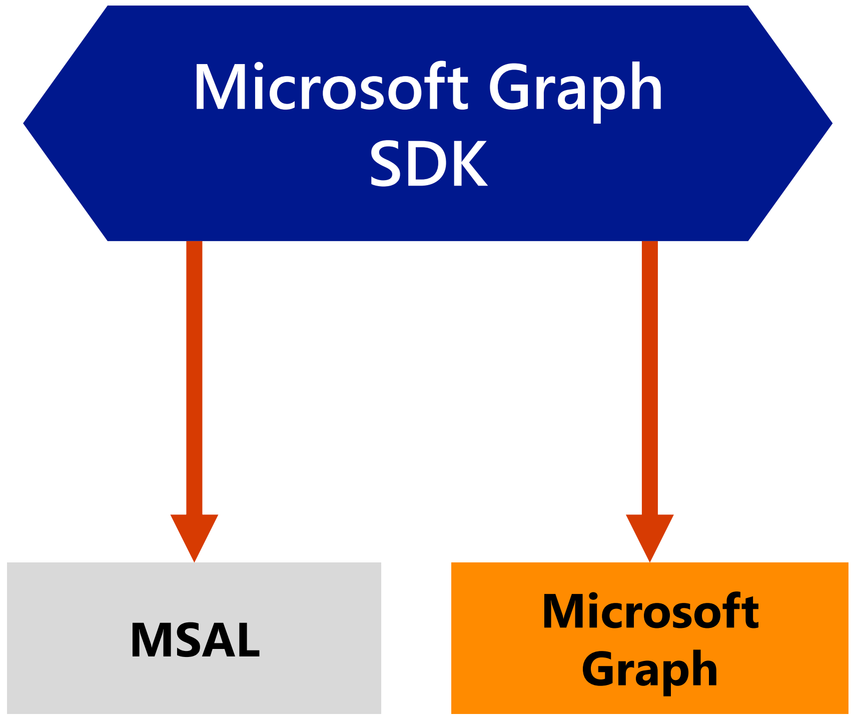
Authentication

- Providers to integrate the Microsoft Graph SDK with the MSAL application builders
- Supports various authentication flows

Query Microsoft Graph by using SDKs (2 of 4)

Wrapper for the MSAL library:

- Supplies authentication provider helpers
- Uses MSAL "under the hood"
- Helpers automatically acquire tokens on your behalf
- Reduces the complexity of using Microsoft Graph in your application



Query Microsoft Graph by using SDKs (3 of 4)

Create a Microsoft Graph client (example)

```
// Build a client application.
IPublicClientApplication publicClientApplication = PublicClientApplicationBuilder
            .Create("INSERT-CLIENT-APP-ID")
            .Build();

// Create an authentication provider by passing in a client application and graph scopes.
DeviceCodeProvider authProvider = new DeviceCodeProvider(publicClientApplication, graphScopes);

// Create a new instance of GraphServiceClient with the authentication provider.
GraphServiceClient graphClient = new GraphServiceClient(authProvider);
```

Query Microsoft Graph by using SDKs (4 of 4)

Retrieve a list of entities (example)

```
// GET https://graph.microsoft.com/v1.0/me/messages?$select=subject,sender&$filter=<some
condition>&orderBy=receivedDateTime

var messages = await graphClient.Me.Messages
    .Request()
    .Select(m => new {
        m.Subject,
        m.Sender
    })
    .Filter("<filter condition>")
    .OrderBy("receivedDateTime")
    .GetAsync();
```

Apply best practices to Microsoft Graph

Authentication

- The HTTP Authorization request header, as a Bearer token
- The graph client constructor, when using a Microsoft Graph client library

Consent and authorization

- Use least privilege
- Use the correct permission type based on scenarios
- Consider the end user and admin experience
- Consider the end user and admin

Handle responses effectively

- Pagination
- Evolvable enumerations

Storing data locally

- Only cache or store data locally if necessary for a specific scenario
- Your application should also implement proper retention and deletion policies

Summary and knowledge check

In this module, you learned how to:

- Explain the benefits of using Microsoft Graph
- Perform operations on Microsoft Graph by using REST and SDKs
- Apply best practices to help your applications get the most out of Microsoft Graph

1. Which HTTP method is used to update a resource with new values?
2. Which of the components of the Microsoft 365 platform is used to deliver data external to Azure into Microsoft Graph services and applications?

Discussion and lab

Group discussion questions

- What are service principals and how can they be used?
- Describe a scenario where you would want to use conditional access? What impact on application code would that have?
- Describe some scenarios where it would be recommended to use a shared access signature.

Lab 06: Authenticate by using OpenID Connect, MSAL, and .NET SDKs

In this lab, you will register an application in Microsoft Entra ID, add a user, and then test the user's access to the application to validate that Entra ID can secure it. You will also use the Graph Explorer tool to build and test requests against the Graph API for an Entra ID user account.

http://aka.ms/az204labs

- Exercise 1: Configure a single-tenant Entra ID environment
- Exercise 2: Create a single-tenant ASP.NET web app

End of presentation