

Module 06: Implement user authentication and authorization



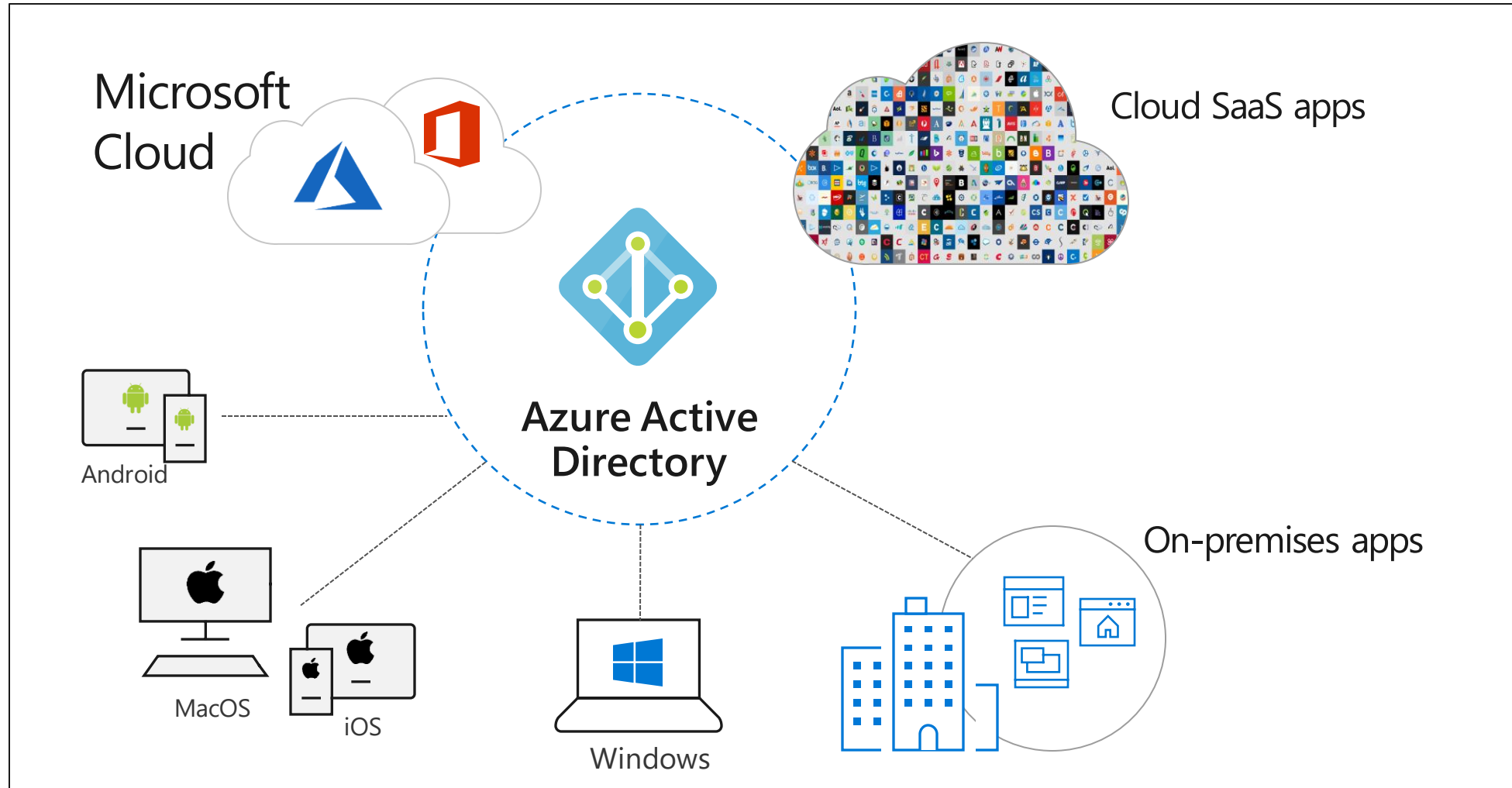
Topics

- Microsoft identity platform
- Microsoft Authentication Library (MSAL)
- Microsoft Graph
- Authorizing data operations in Azure Storage

Lesson 01: Microsoft identity platform



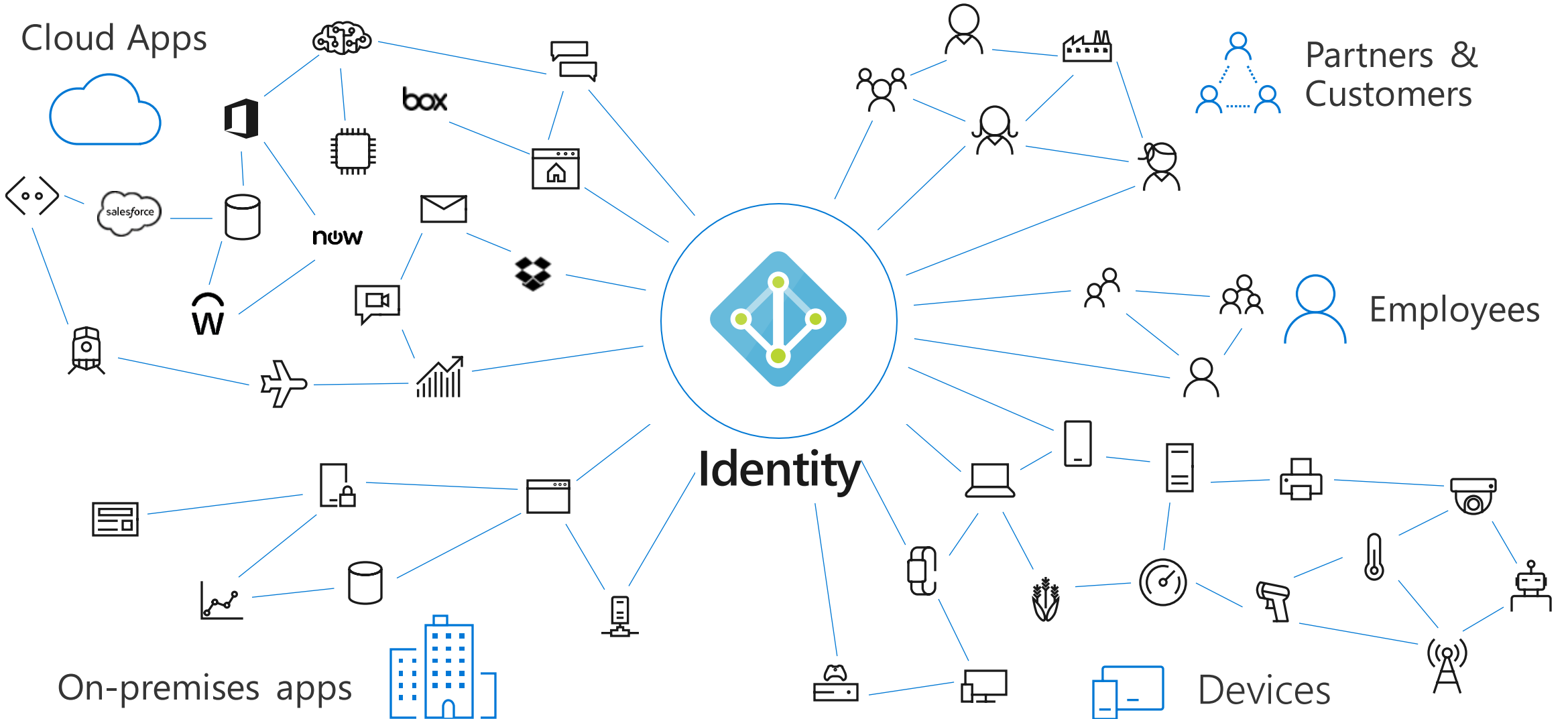
Azure Active Directory



Azure AD evolution

- Use Azure AD (v1.0):
 - Authenticate against only work and school accounts (provisioned in Azure AD)
- Use Microsoft identity platform (v2.0) to:
 - Authenticate against organizational (work and school) accounts
 - Authenticate against personal accounts (Microsoft account)
 - Authenticate against customer-supplied identity such as LinkedIn, Facebook, and Google

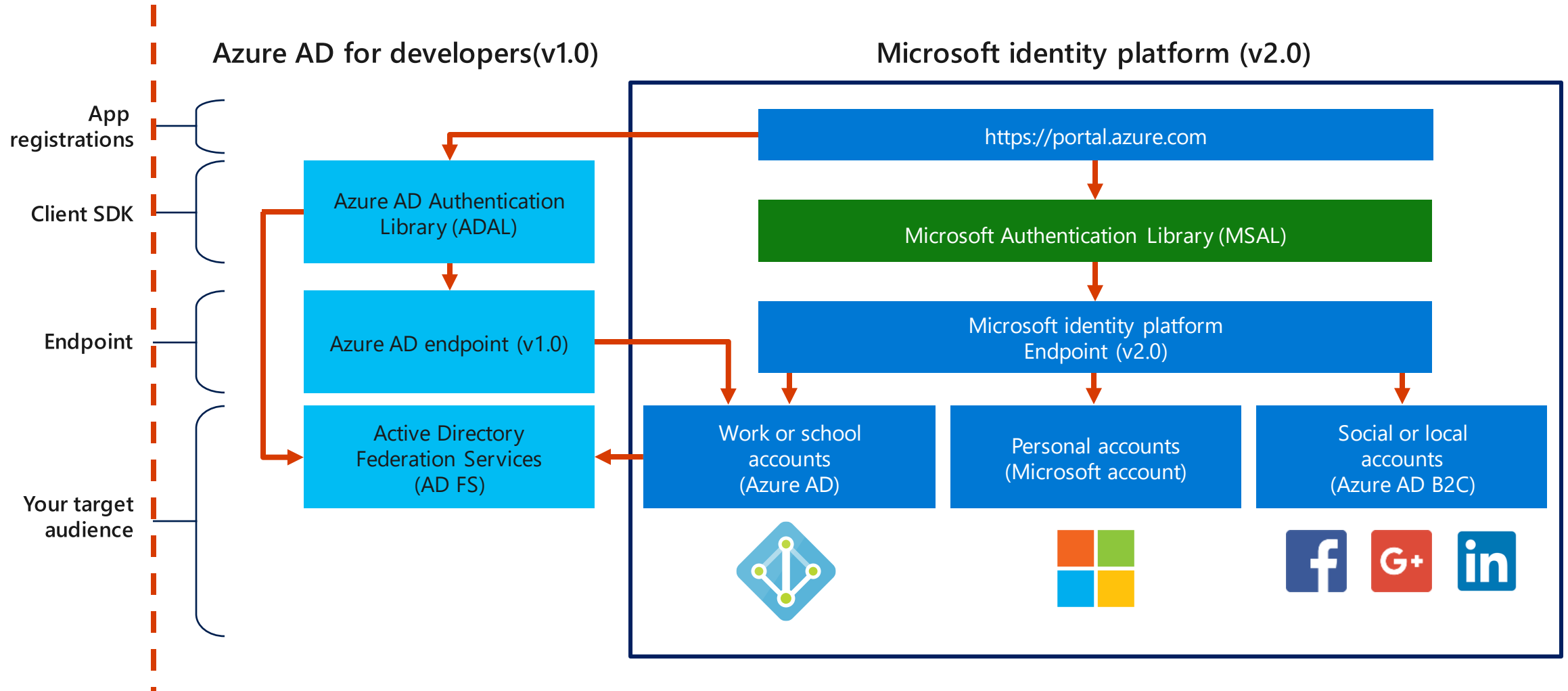




Microsoft Identity Platform Overview

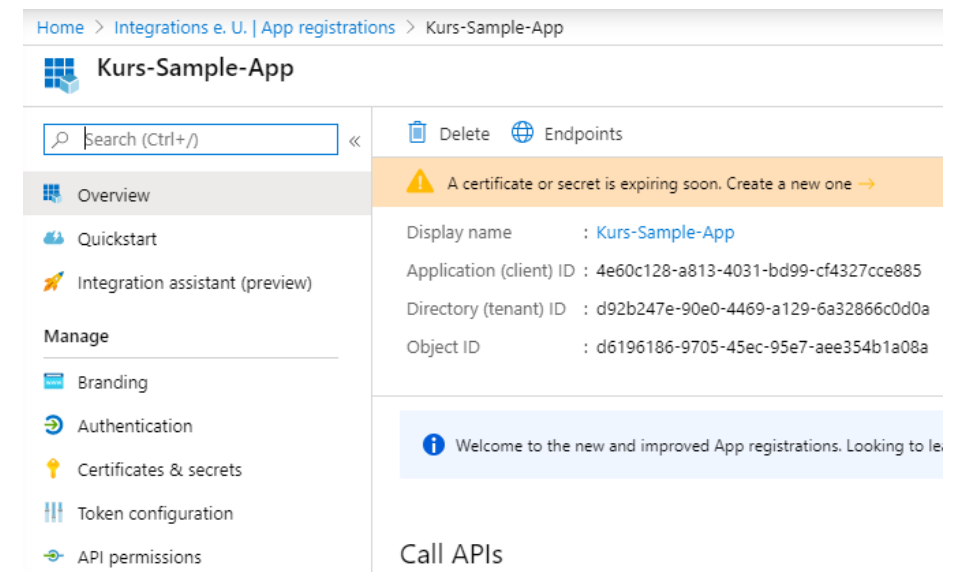
- An evolution of the Azure Active Directory (Azure AD) identity service and developer platform
- A full-featured identity platform that provides:
 - An authentication service
 - Open-source libraries
 - Application registration and configuration
 - Full developer documentation
 - Code samples
 - Support for industry standard protocols (OAuth 2.0, Open ID Connect)
 - Support for Azure AD v1.0 and Azure AD v2.0

ADAL vs MSAL



Application registration

- To outsource authentication to Azure AD, applications must be registered in one or more Azure AD tenants:
 - Single-tenant: common with line-of-business (LOB) applications
 - Multitenant: common with SaaS applications developed by ISVs
- The application registration might include, depending on the type:
 - Application ID URI
 - Reply URL and redirect URI
 - Application ID
 - Key



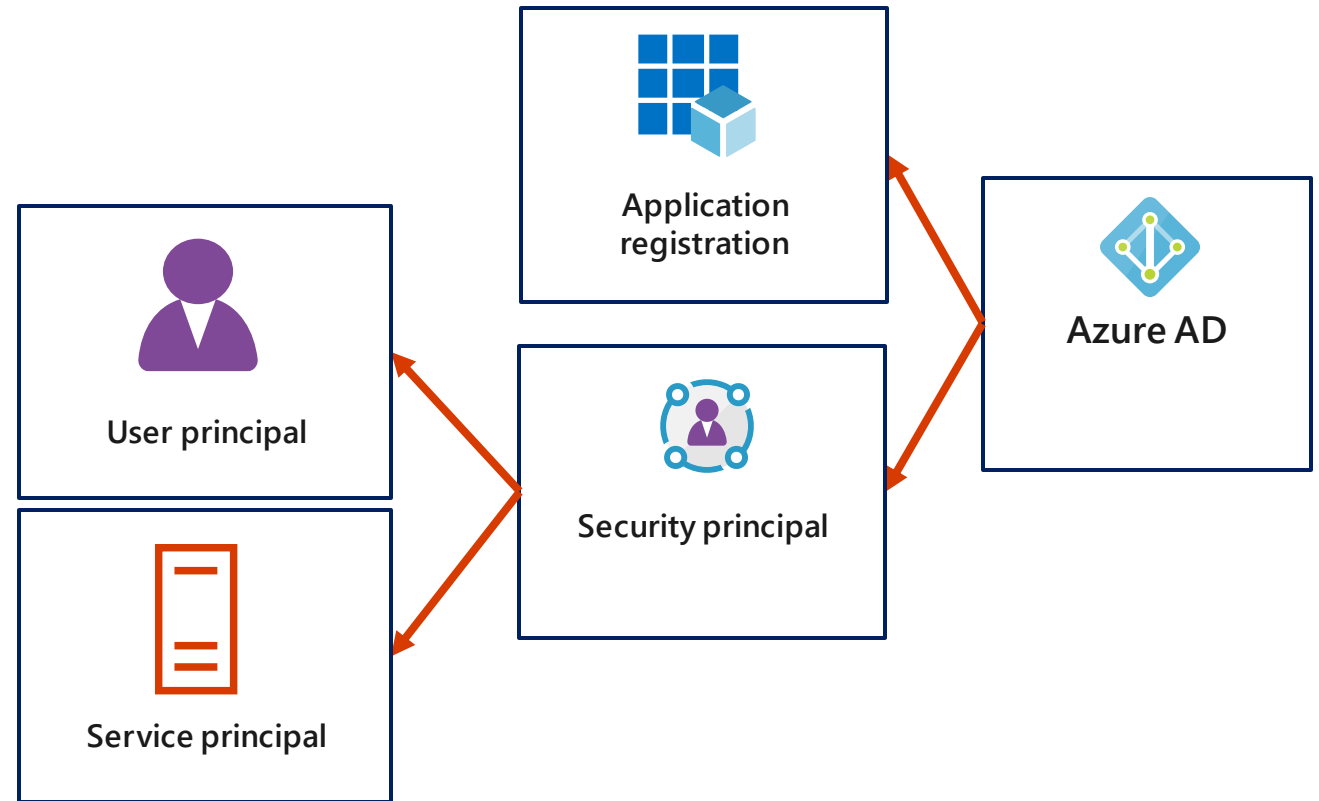
App Registration & Service Principal

- App Registrations hold the following information:
 - App Configuration
 - Scopes
- A Service Principal is an Object that represents an Application in a Directory
 - Can be the same Azure AD (Single Tenant) or another Azure AD Tenant (Multi Directory)
- As far as App Registrations are concerned the Service Principal holds following information:
 - Consents
 - Tokens

Objects in Azure AD

Azure AD includes two object types:

- Application registration
- Security principal:
 - User principal
 - Service principal



Authentication



OpenID Connect

- A simple identity layer on top of the OAuth 2.0 protocol, which allows computing clients to verify the identity of an end-user based on the authentication performed by an authorization server
- Has become the leading standard for single sign-on and identity provision on the Internet by using:
 - simple JSON-based identity tokens (JWT),
 - delivered via the OAuth 2.0 protocol



JSON Web Tokens

- An open, industry standard RFC 7519 method for representing claims securely between two parties
- Defines a format of how Auth information can be exchanged
- Can be sent through a URL, POST parameter, or inside an HTTP header
- Contains all the required information about the user
- Doku @ <https://jwt.io/>

Encoded

PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0Ij0nRydwVW9uLjJVA95OrM7E2cBab30RMhrHDcEfxjoYZgeFONfh7HgQ

Decoded

EDIT: PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

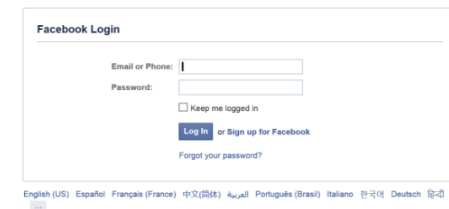
```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + ".",
  base64UrlEncode(payload),
  secret
) @secret base64 encoded
```

OAuth 2.0

- An open standard for authorization, allowing users to log in to third party websites using their external accounts without exposing their password
- Technically OAuth 2.0 is an authorization protocol
- Supported by
 - Azure AD
 - Live ID,
 - Google,
 - Facebook, ...

A screenshot of the Facebook login page. It features a white box with the title "Facebook Login". Inside, there are input fields for "Email or Phone:" and "Password:". Below these is a checkbox labeled "Keep me logged in". At the bottom of the box are two links: "Log In" and "or Sign up for Facebook". Below the box, there is a link "Forgot your password?". At the very bottom, there is a row of language links: "English (US)", "Español", "Français (France)", "中文(简体)", "العربية", "Português (Brasil)", "Italiano", "한국어", "Deutsch", and "हिन्दी".

Authentication endpoints

Multitenant applications (the same for all tenants):

<https://login.microsoftonline.com/common>

Single-tenant applications (tenant-specific):

[https://login.microsoftonline.com/\\$TENANTNAME.onmicrosoft.com](https://login.microsoftonline.com/$TENANTNAME.onmicrosoft.com)

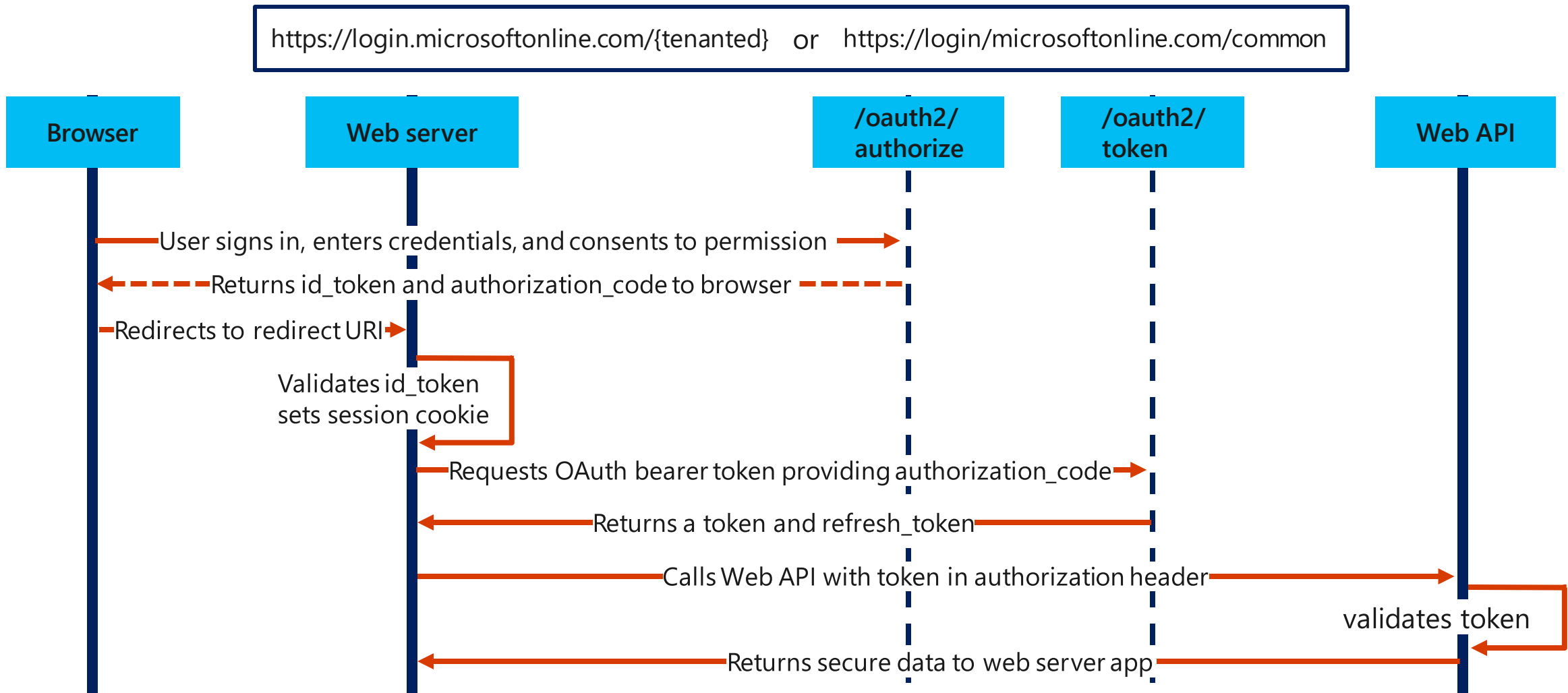
Understanding the OAuth 2.0 implicit grant flow in Azure AD

- The OAuth 2.0 authorization code grant relies on two separate endpoints:
 - The authorization endpoint: used during the user interaction phase
 - The token endpoint: used by a client to exchange the authorization code for an access token and, optionally, refresh tokens
- The OAuth 2.0 implicit grant is a variant of an authorization grant:
 - It allows the client to obtain an access token (and id_token, when using OpenID Connect) directly from the authorization endpoint, without relying on the token endpoint
 - It never returns refresh tokens to the client
 - It is intended for JavaScript applications running in a browser (such as SPAs)
 - It should not be used for:
 - Native clients
 - Web applications that include a back end and consume an API from the back-end code

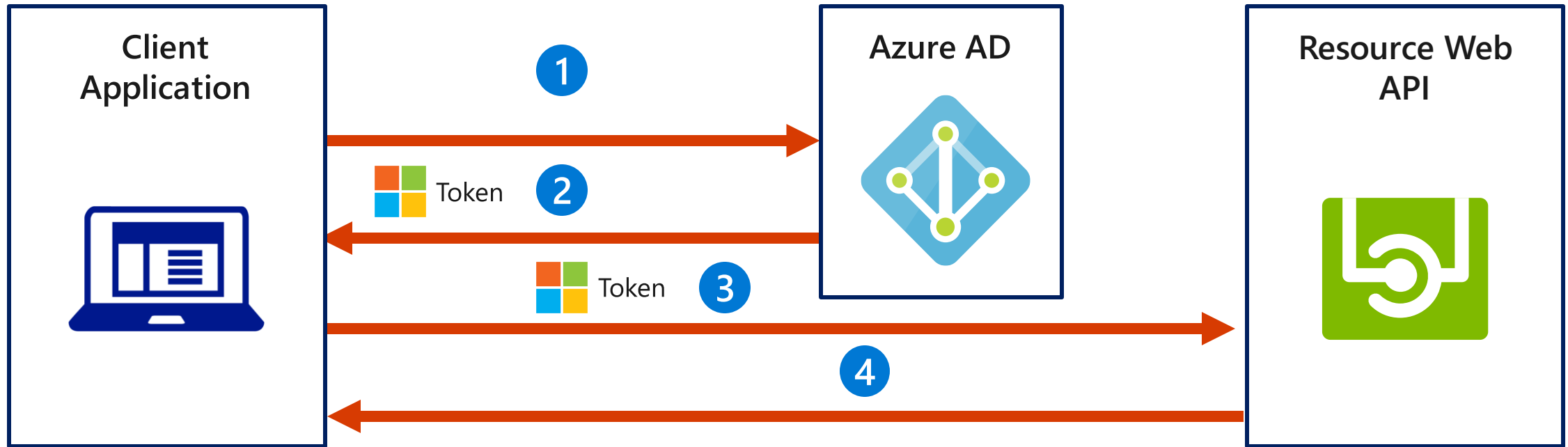
Authorize access to Azure AD web applications by using the OAuth 2.0 code grant flow

1. Register your application with your Azure AD tenant
2. Request an authorization code
3. Use the authorization code to request an access token
4. Use the access token to access the resource
5. Refresh the access token

Authorize access to web applications by using OAuth



Service-to-service calls using client credentials



How the client credentials grant flow works in Azure AD:

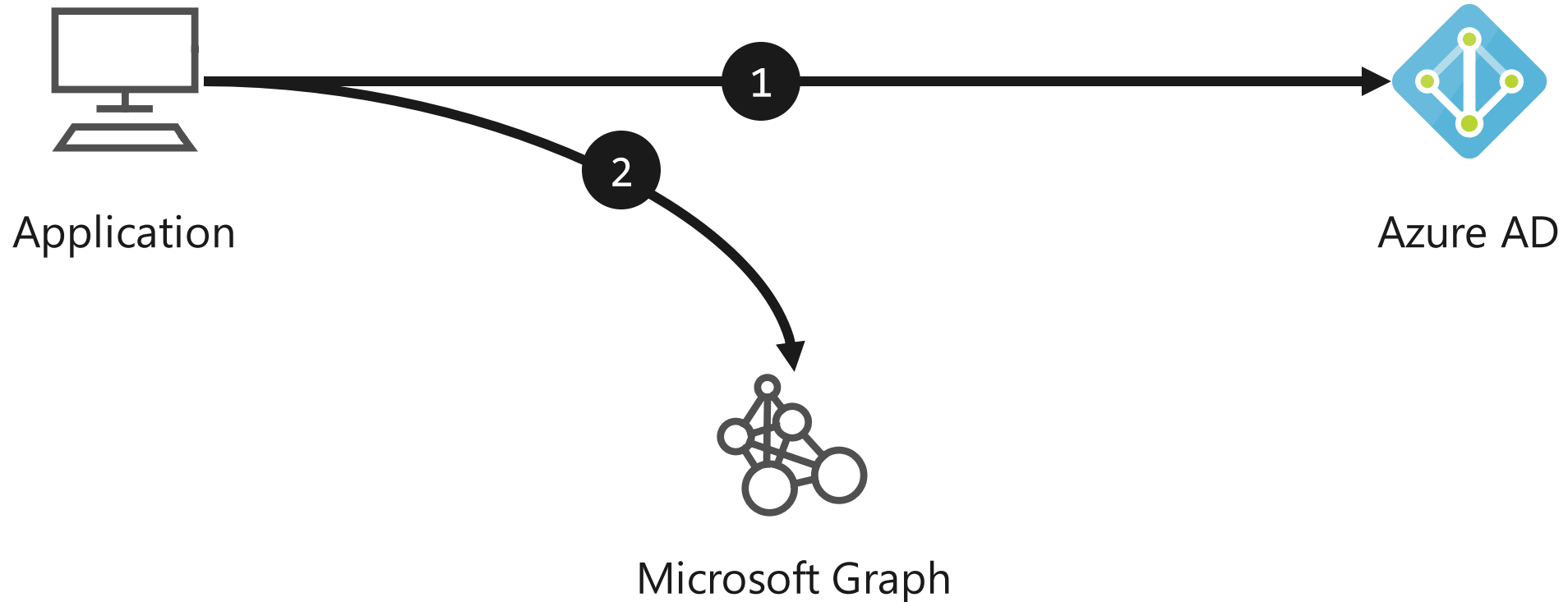
1. The client application authenticates to the Azure AD token issuance endpoint and requests an access token
2. The Azure AD token issuance endpoint issues the access token
3. The access token is used to authenticate to the secured resource
4. Data from the secured resource is returned to the client application

Common authentication flows

- Interactive:
 - User authenticates by using a web browser
- On-Behalf-Of:
 - Application authenticates on behalf of a user
- Client credentials:
 - Application authenticates by using pre-generated credentials
- Device code:
 - User authenticates on another device

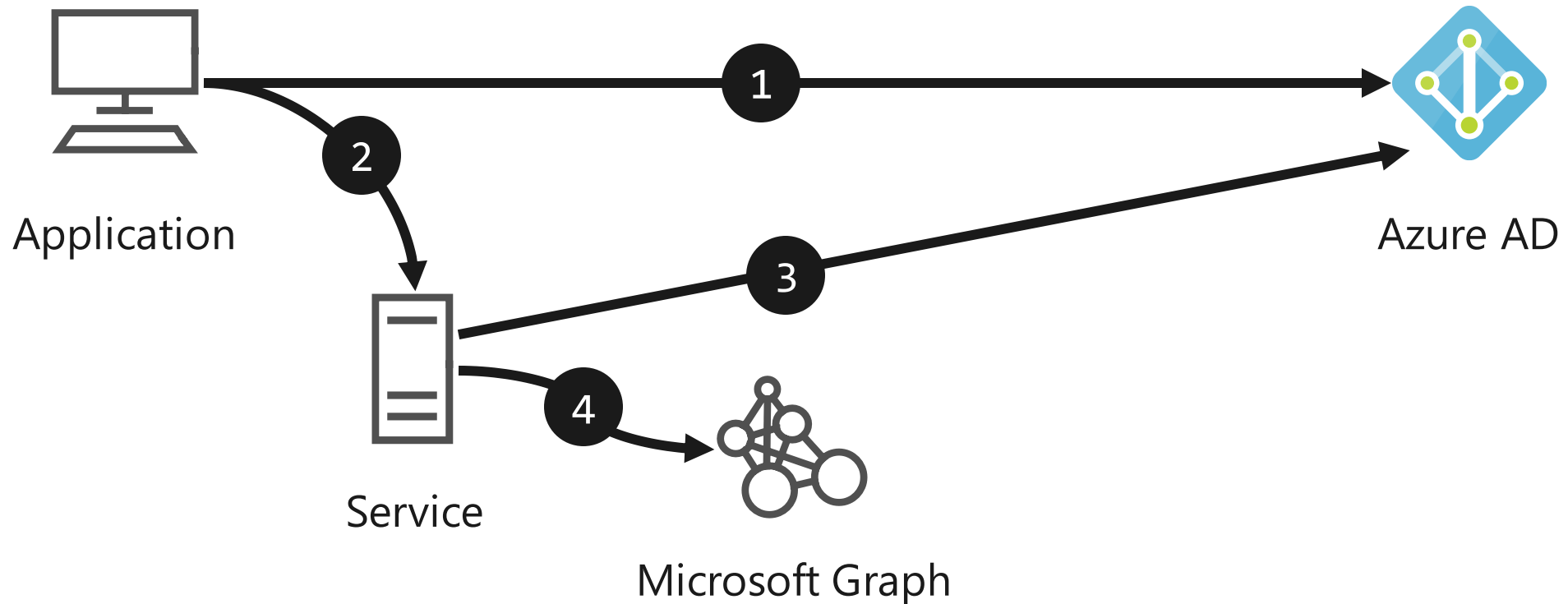
Interactive authentication flow

User authenticates by using a web browser



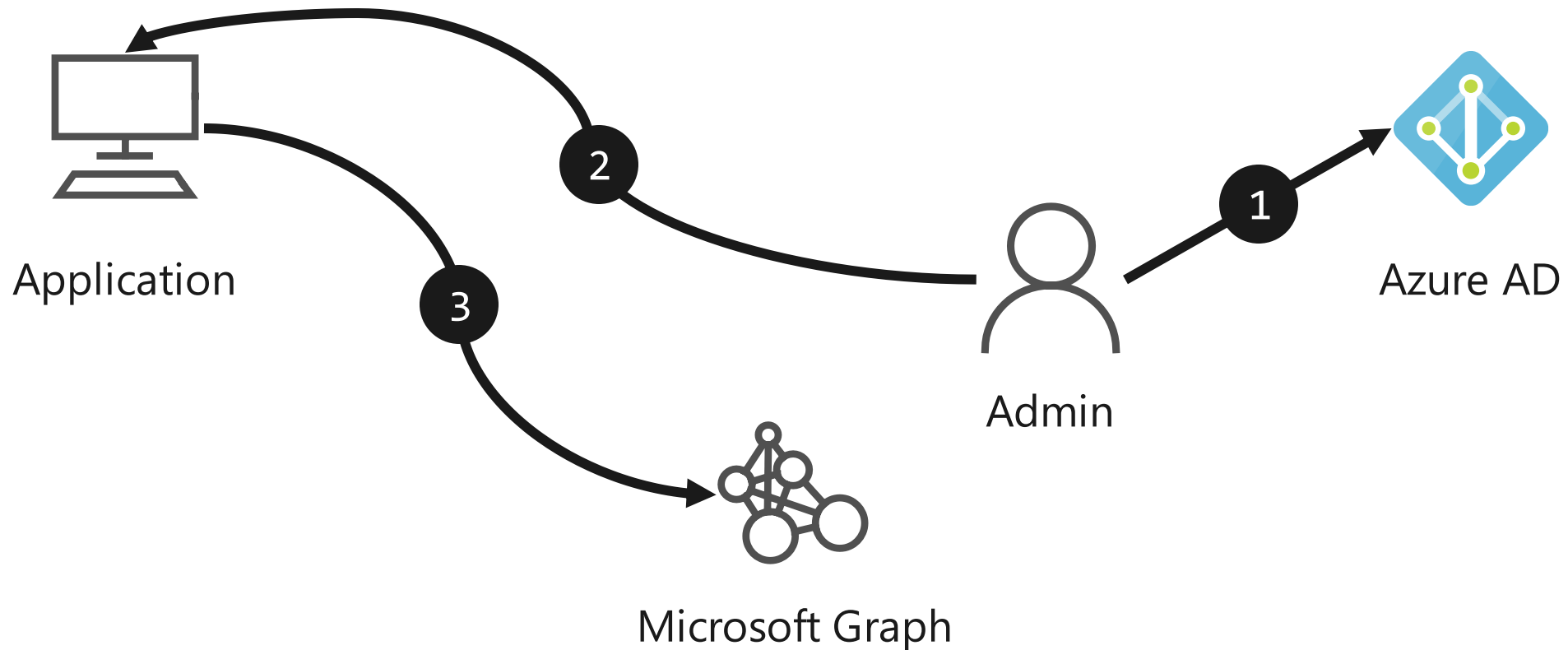
On-Behalf-Of authentication flow

Application authenticates on behalf of a user



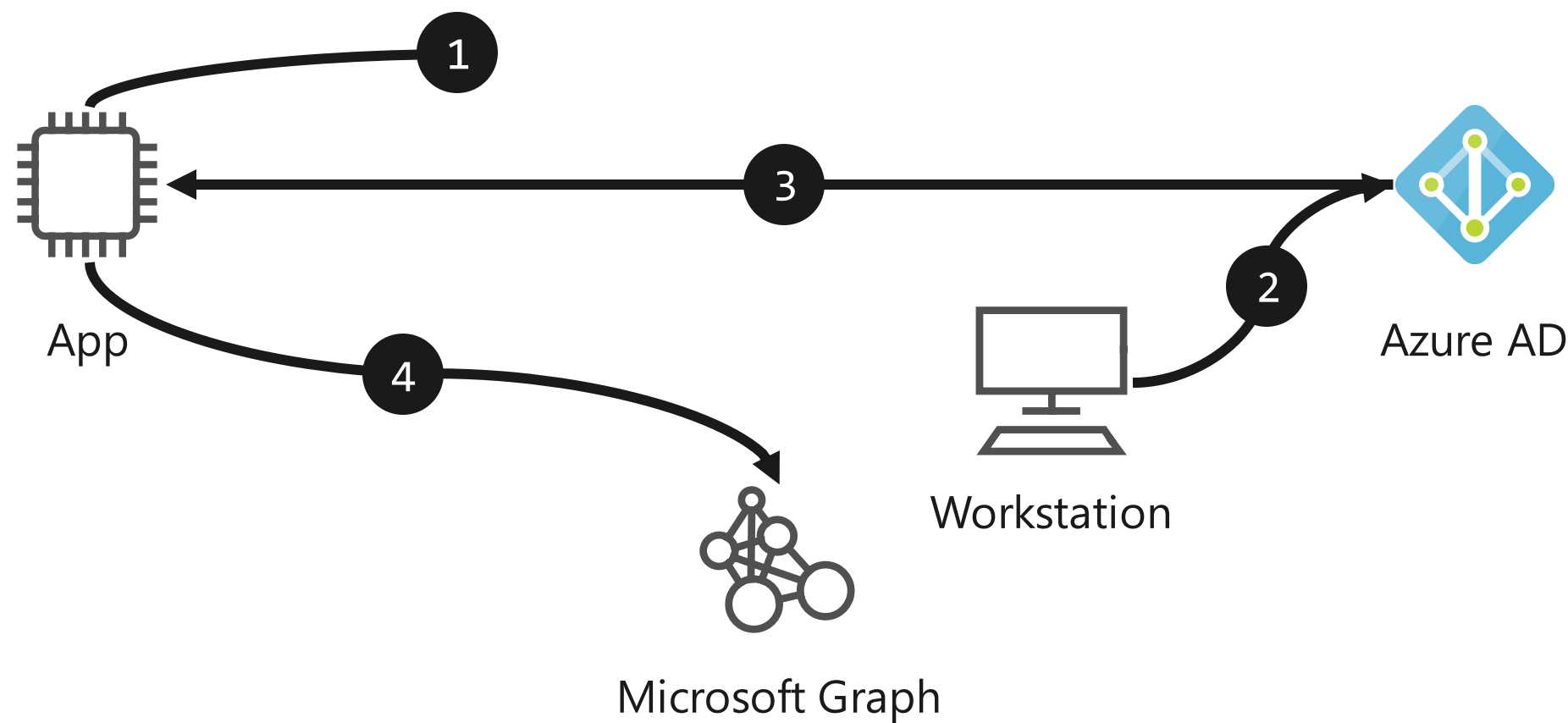
Client credentials authentication flow

Application authenticates by using pre-generated credentials



Device code authentication flow

User authenticates on another device



Certificate-based authentication

- Each web-based client establishes identity to a server
 - By using a digital certificate
- Provides additional security beyond traditional user authentication

In Azure Active Directory, certificate-based authentication can be used to connect to:

- Custom services authored by your organization
- Microsoft SharePoint Online
- Microsoft Office 365 (or Microsoft Exchange)
- Skype for Business
- Azure API Management
- Third-party services deployed in your organization

Demonstration: Register an app with the Microsoft identity platform



Lesson 02: Microsoft Authentication Library (MSAL)



Active Directory Authentication Library (ADAL)

- The library to streamline working with Azure Active Directory from code:
 - Obtains and manages tokens
 - Caches token using a configurable cache
 - Refreshes tokens automatically when they expire
 - Supports asynchronous invocation
- Available in multiple languages such as:
 - C#
 - JavaScript, Angular Wrapper
 - Objective C
 - Java, Python

Microsoft Authentication Library (MSAL)

- The library to streamline working with Microsoft identity platform from code:
 - Obtains and manages tokens
 - Caches tokens by using a configurable cache
 - Refreshes tokens automatically when they expire
 - Supports asynchronous invocation
- Available on multiple platforms such as:
 - .NET
 - JavaScript (Node, Angular)
 - Android
 - iOS

Creating an authentication context by using MSAL

```
string tenant = "contoso.onmicrosoft.com";  
string clientId = "00000000-0000-0000-0000-000000000000";  
string authority = $"https://login.microsoftonline.com/{tenant}";  
  
// Create MSAL context using AAD authority  
var clientApp = PublicClientApplicationBuilder.Create(clientId)  
    .WithAuthority(AzureCloudInstance.AzurePublic, tenant)  
    .Build();
```



Acquiring a token interactively using MSAL

```
var scopes = new string[] { "user.read" };  
var windowHandle = new WindowInteropHelper(this).Handle;  
  
// Acquire token using an interactive prompt  
var authResult = await clientApp.AcquireTokenInteractive(scopes)  
    .WithParentActivityOrWindow(windowHandle)  
    .WithPrompt(Prompt.SelectAccount)  
    .ExecuteAsync();  
  
// Observe token property  
var token = authResult.AccessToken;
```



Acquiring a token silently using MSAL

```
var account = accounts.FirstOrDefault();

// Acquire token silently
var authResult = await clientApp.AcquireTokenSilent(scopes, account)
    .ExecuteAsync();

// Observe token property
var token = authResult.AccessToken;
```



Get user profile using MSAL

```
string endpoint = "https://graph.microsoft.com/v1.0/me";

// Create a new instance of HttpClient class
var client = new HttpClient();

// Build an auth header using your token
var authHeader = new AuthenticationHeaderValue(
    "Bearer",
    token
);

// Set httpClient to use the previously-build auth header
client.DefaultRequestHeaders.Authorization = authHeader;

// Make a HTTP GET request to the endpoint
var response = await client.GetAsync(endpoint);
```



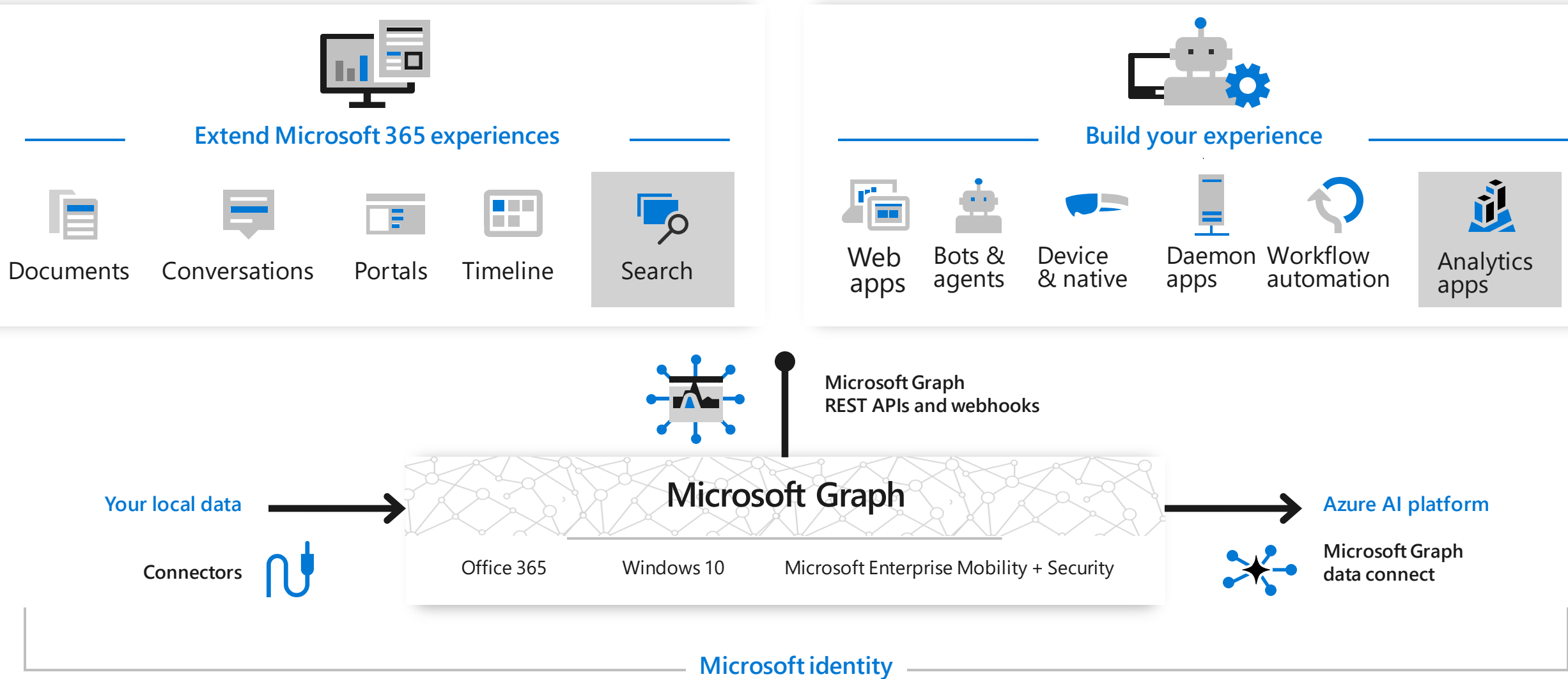
Demonstration: Interactive authentication by using MSAL.NET



Lesson 03: Microsoft Graph



Microsoft 365 platform



Microsoft Graph data and services



Web

Native

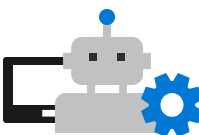
Bots

Apps

Flows

Automation

Analytics



Auth

Authentication code

Implicit-grant

Device-code

On-behalf-of

Client-credentials

User experience

Web components

React components

Univeral Windows Platform

Pickers

Adaptive cards

Libraries

C#

JavaScript/Node

Java

Objective-C

PHP

Python

Capabilities

Webhooks

Deltas

Query

Notifications

Data extensions

More...

Interfaces



Connectors



Microsoft Graph
REST APIs and webhooks



Microsoft Graph
data connect

Data

Microsoft 365

your.domain

Graph data

Office 365

- Users, groups, and organizations
- Outlook
- SharePoint
- OneDrive
- Teams
- Planner
- Excel
- OneNote

Mail, Calendar,
Contacts, Tasks,
Sites, Lists,
Drives, Files

Windows 10

- Activities
- Device relay
- Commands
- Notifications

Channels, Messages,
Tasks, Plans,
Spreadsheets,
Notes,

Dynamics 365

- Business Central

Identity Management,
Access Control,
Synchronization,
Policies

Enterprise Mobility + Security

- Azure AD
- Intune
- Identity Manager
- Advanced Threat Analytics
- Advanced Threat Protection

Administrative Units,
Applications and Devices,
Alerts,
Domains

Advanced Threat Analytics,
Advanced Threat Protection,
and more

Graph explorer

Quickly test requests directly
in the browser

Supported accounts:

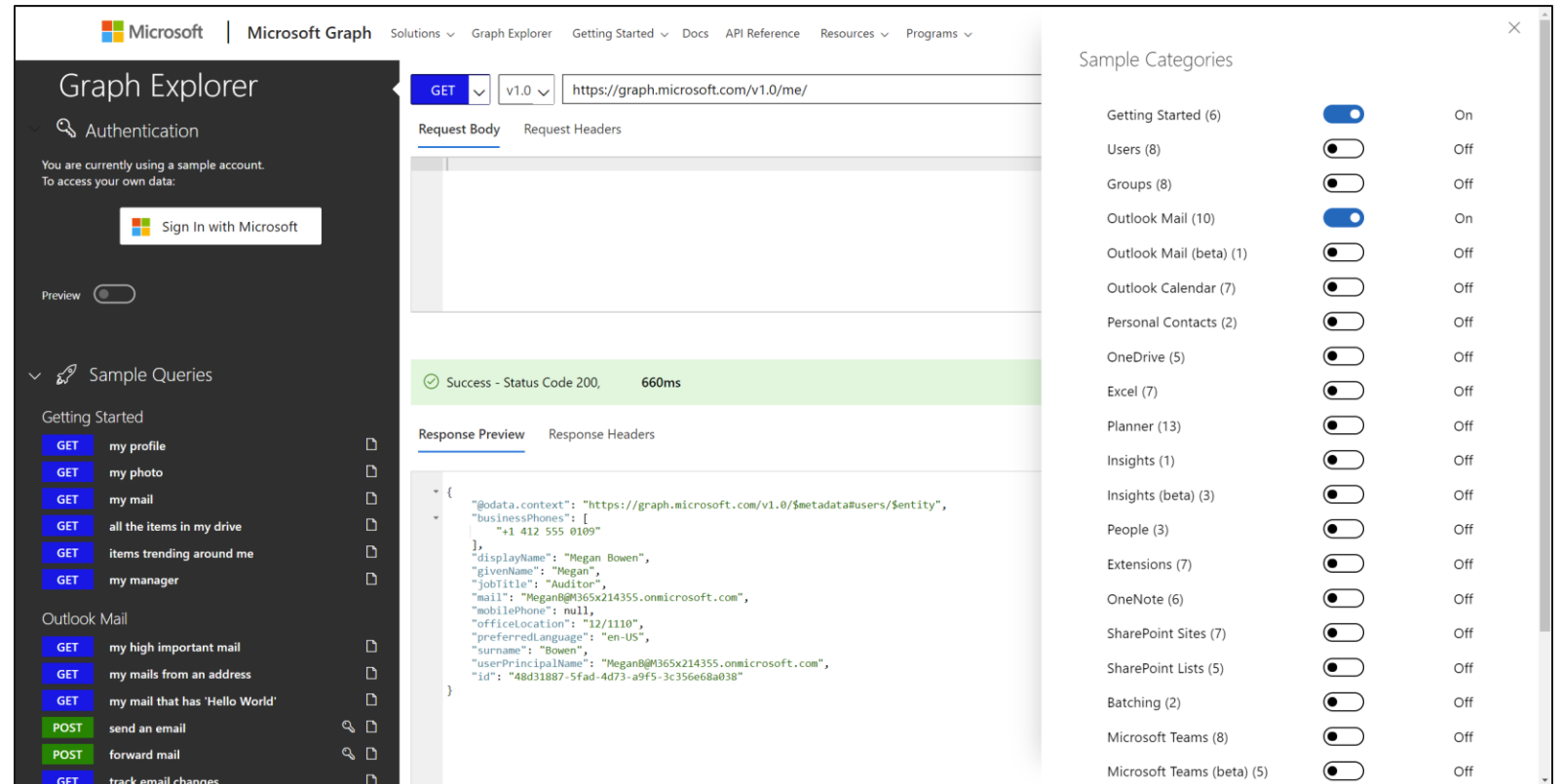
Demo (commercial)

Consumer

(@outlook.com)

Commercial

(@your.domain)



Microsoft Graph SDK

Installs as two packages

Microsoft.Graph

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

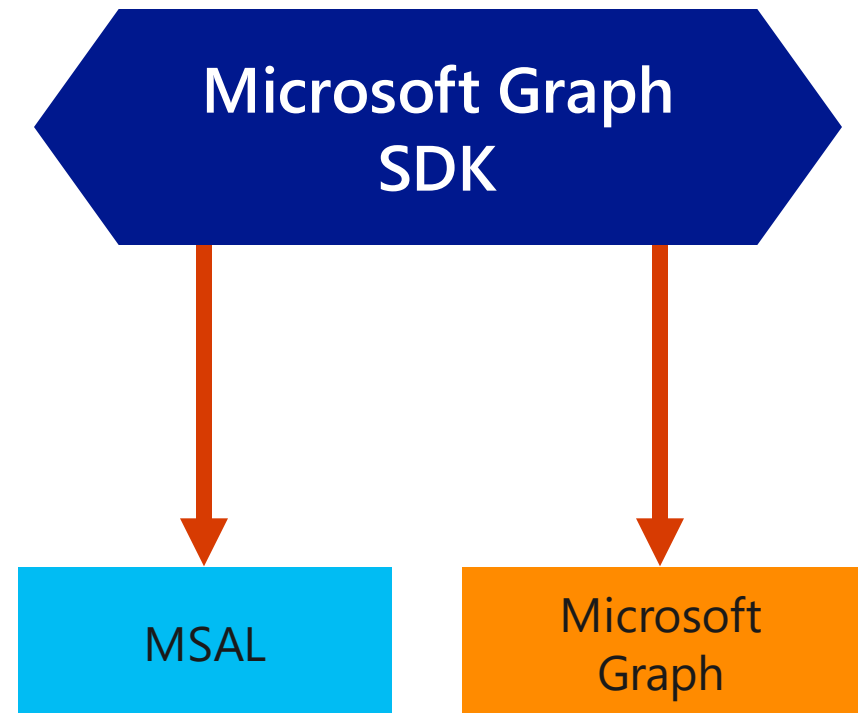
Microsoft.Graph.Auth

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

Microsoft Graph authentication SDK

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



Creating authentication provider

```
string clientId = "00000000-0000-0000-0000-000000000000";
string redirectUri = "https://login.microsoftonline.com/common/oauth2/nativeclient";
var scopes = new List<string> { "user.read" };

// Build a client application.
var clientApplication = PublicClientApplicationBuilder
    .Create(clientId)
    .WithRedirectUri(redirectUri)
    .Build();

// Create an authentication provider by passing in a client application and scopes.
var authProvider = new InteractiveAuthenticationProvider(
    clientApplication,
    scopes
);
```



Authentication providers

Provider	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
Interactive	Mobile and desktops applications call Microsoft Graph in the name of a user
Username/password	The application signs in a user by using their username and password

Using device code provider

```
string clientId = "00000000-0000-0000-0000-000000000000";  
var scopes = new List<string> { "user.read" };  
  
// Build a client application.  
var clientApplication = PublicClientApplicationBuilder  
    .Create(clientId)  
    .WithAadAuthority(AzureCloudInstance.AzurePublic, AadAuthorityAudience.AzureAdMultipleOrgs)  
    .Build();  
  
// Create an authentication provider by passing in a client application and scopes.  
var authProvider = new DeviceCodeProvider(  
    clientApplication,  
    scopes  
);
```



Using integrated windows provider

```
string clientId = "00000000-0000-0000-0000-000000000000";
string tenant = "contoso.onmicrosoft.com";
var scopes = new List<string> { "user.read" };

// Build a client application.
var clientApplication = PublicClientApplicationBuilder
    .Create(clientId)
    .WithAadAuthority(AzureCloudInstance.AzureUsGovernment, tenant)
    .Build();

// Create an authentication provider by passing in a client application and scopes.
var authProvider = new IntegratedWindowsAuthenticationProvider(
    clientApplication,
    scopes
);
```



Using Graph Service client

```
// Create a new instance of GraphServiceClient with the authentication provider.  
GraphServiceClient graphClient = new GraphServiceClient(  
    authProvider  
);  
  
// Makes a request to https://graph.microsoft.com/v1.0/me  
User me = await graphClient  
    .Me  
    .Request()  
    .GetAsync();
```



Demonstration: Retrieving profile information by using the Microsoft Graph SDK



Lesson 04: Authorizing data operations in Azure Storage

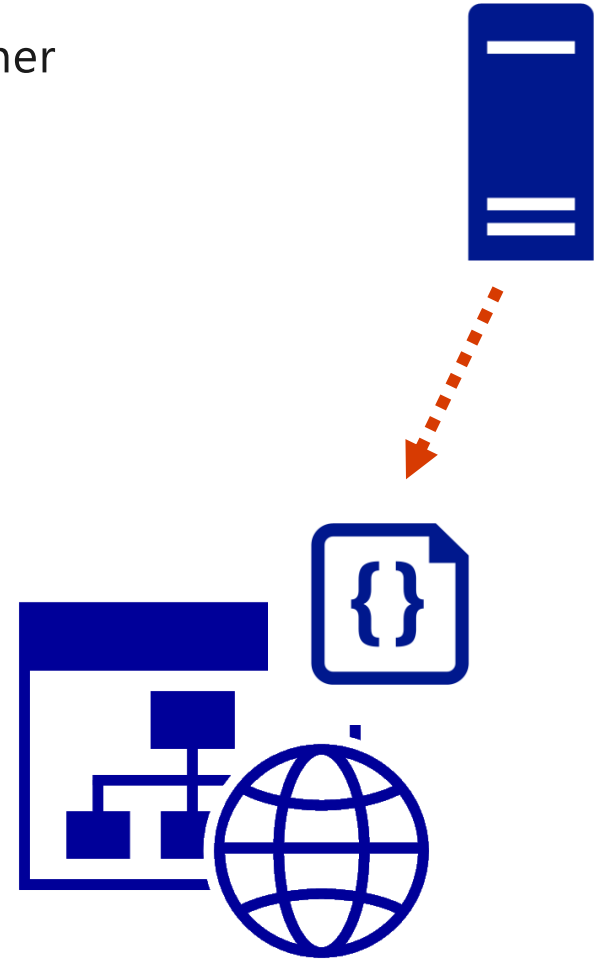


Container permissions

- There are three levels of container access that are available:
 - **Full public read access:**
 - Enumerate container blobs
 - Read individual blobs
 - Cannot enumerate containers
 - **Public read access for blobs only:**
 - Read individual blobs
 - **No public read access:**
 - No access to blobs, containers, or enumerating contents

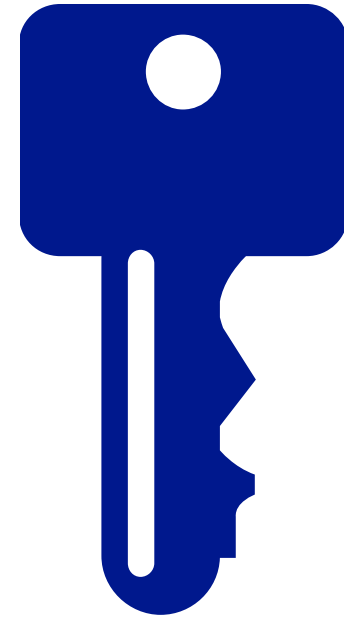
CORS support for the Azure Storage services

- CORS is an HTTP feature that enables requests from one domain to another
 - This is mostly required to issue API calls from a JavaScript application
- Azure Storage supports enabling CORS at the service level
 - Can be scoped to specific domains and specific permissions
 - Can be scoped to storage services
 - Blob
 - File
 - Queue
 - Table



Authorization

- Every request must be authorized:
 - Exception - blob or container resources that have been made publicly available (opt-in)
- REST API requests can use a Shared Key authorization scheme:
 - Requires two headers:
 - Date (or x-ms-date)
 - Authorization



Shared Access Signatures

- A Shared Access Signature (SAS Token) is a URI that grants access to a protected container, blob, queue, or table for a specific time interval
 - Allows client application to access a resource without using the storage account key
 - Should only be used with secure (HTTPS) requests
 - Can be generated with the following components:
 - Start Time
 - Expiry Time
 - Permission Levels (Read, Write, Delete, List, None)

Establishing a stored access policy

- Policy that can generate short-lifetime signatures to access resources
 - Signatures are concatenated to the end of the resource URI
 - Signatures are verified on the server for validity
- Signatures generated from a single policy share characteristics:
 - Permission (read, write, read-write, delete)
 - Start time
 - Expiry time
 - Resource scope (blob, table, etc.)
- All signatures generated by a single policy can be revoked as a group

Shared Access Signatures (SASs)

`https://myaccount.blob.core.windows.net/sascontainer/sasblob.txt?sv=2012-02-12&st=2013-04-29T22%3A18%3A26Z&se=2013-04-30T02%3A23%3A26Z&sr=b&sp=rw&sig=Z%2FRHIX5Xcg0Mq2rql3OIWTjEg2tYkboXr1P9ZUXDtkk%3`

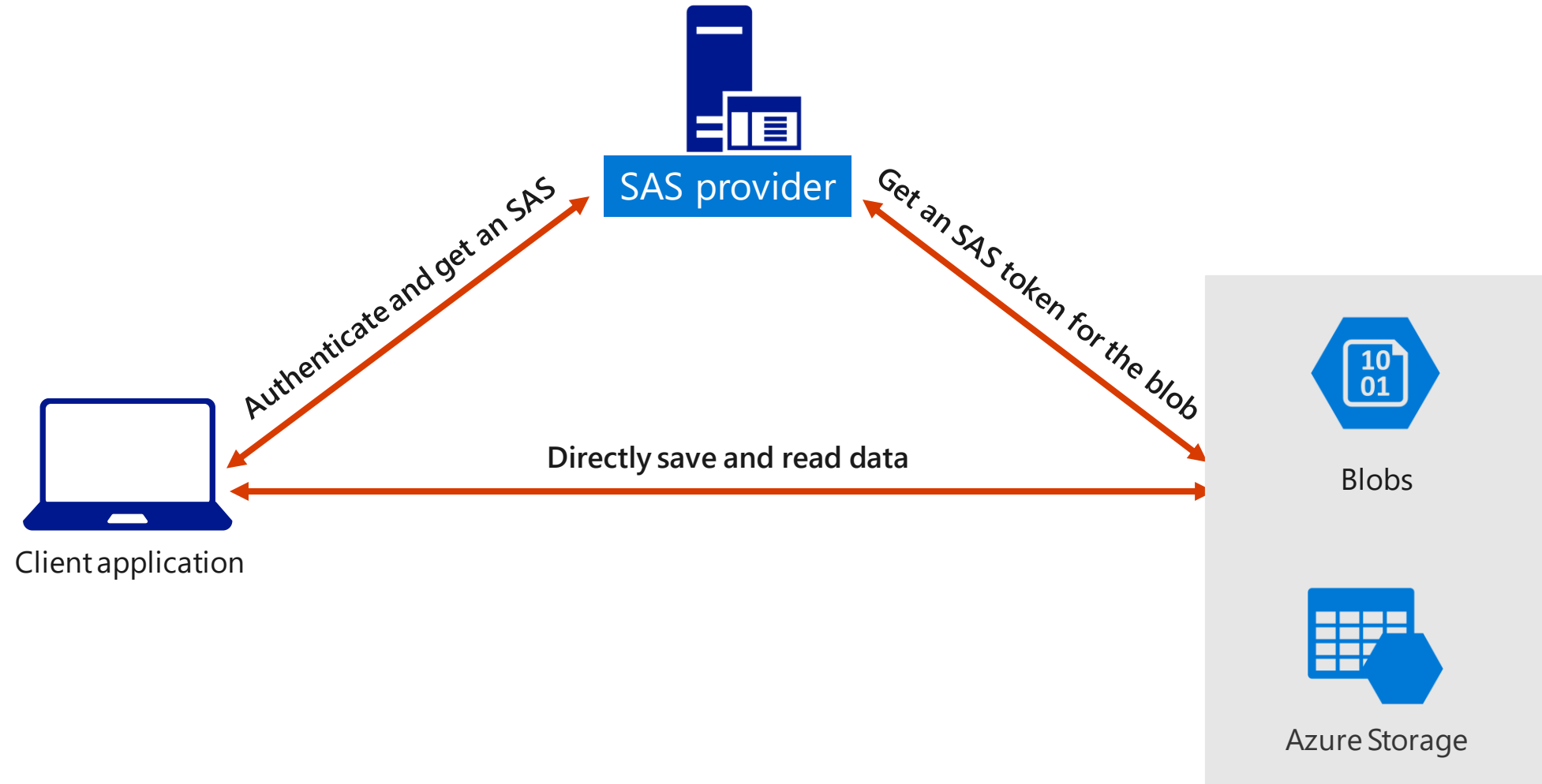
Component	Content	Description
Blob URI	<code>https://myaccount.blob.core.windows.net/sascontainer/sasblob.txt</code>	The address of the blob. Note that using HTTPS is highly recommended.
Storage services version	<code>sv=2012-02-12</code>	For Azure Storage services version 2012-02-12 and later, this parameter indicates the version to use.
Start time	<code>st=2013-04-29T22%3A18%3A26Z</code>	Specified in an International Organization for Standardization (ISO) 8061 format. If you want the SAS to be valid immediately, omit the start time.
Expiration time	<code>se=2013-04-30T02%3A23%3A26Z</code>	Specified in an ISO 8061 format.

Shared Access Signatures (SASs) (continued)

<https://myaccount.blob.core.windows.net/sascontainer/sasblob.txt?sv=2012-02-12&st=2013-04-29T22%3A18%3A26Z&se=2013-04-30T02%3A23%3A26Z&sr=b&sp=rw&sig=Z%2FRHIX5Xcg0Mq2rql3OIWTjEg2tYkboXr1P9ZUXDtkk%3D>

Component	Content	Description
Resource	sr=b	The resource is a blob.
Permissions	sp=rw	The permissions granted by the SAS include Read (r) and Write (w).
Signature	sig=Z%2FRHIX5Xcg0Mq2rql3OIWTjEg2tYkboXr1P9ZUXDtkk%3D	The signature authenticates access to the blob. It is a Hash-based Message Authentication Code (HMAC) function computed over a string to sign and a key by using the SHA256 algorithm and then encoded by using Base64 encoding.

Valet key pattern by using Shared Access Signatures



Stored access policies

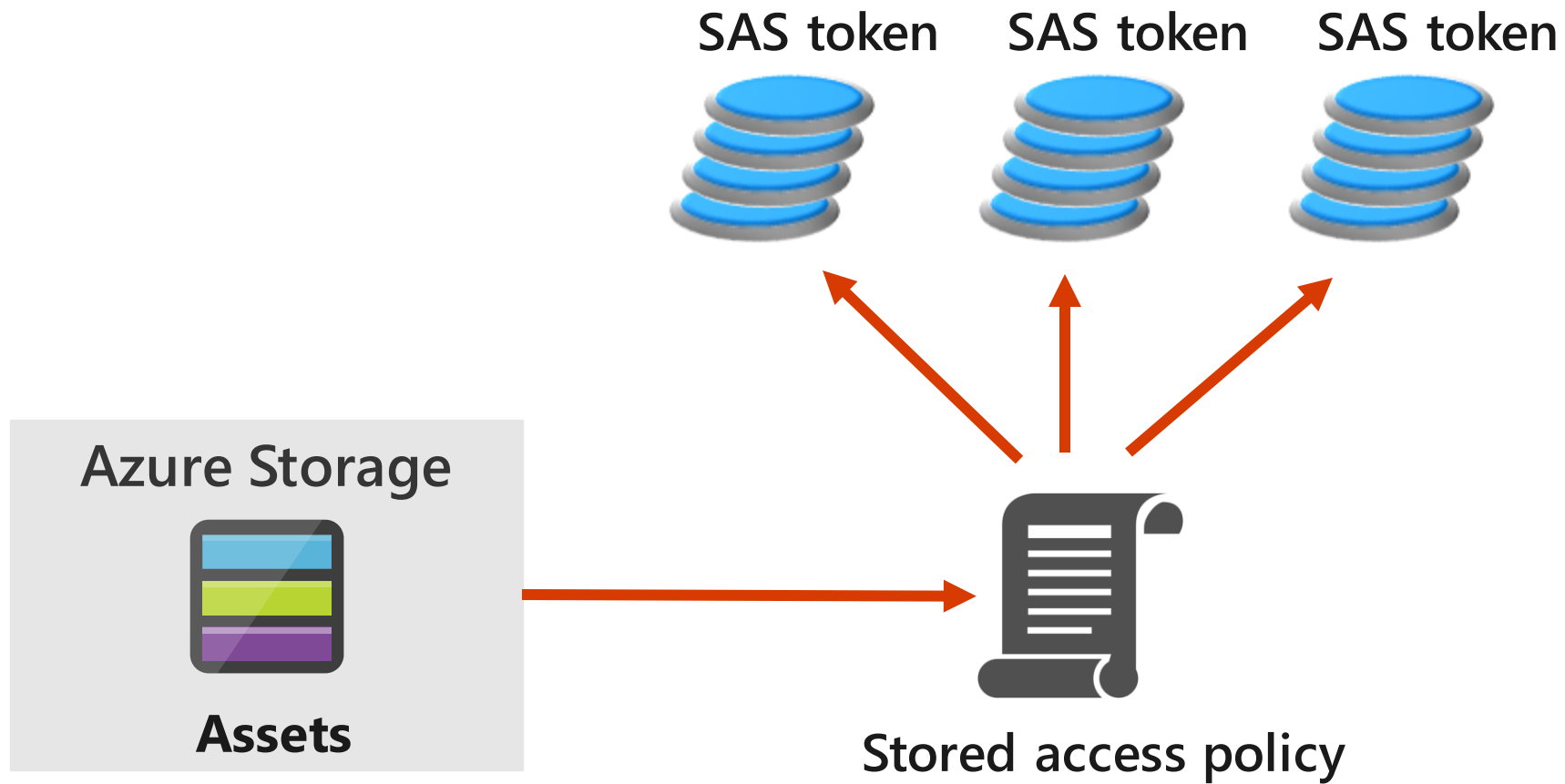
- A Shared Access Signature can take one of two forms:
 - Ad hoc:
 - When you create an ad hoc SAS, the start time, expiration time, and permissions for the SAS are all specified in the SAS URI (or are inferred in the case where the start time is omitted)
 - SAS is generated from a stored access policy:
 - A stored access policy is defined on a resource container and can be used to manage constraints for one or more Shared Access Signatures
 - When you associate an SAS with a stored access policy, the SAS inherits the constraints defined for the stored access policy
- Anyone who obtains the SAS can use it



Stored access policies (continued)

- Granular control over a set of shared access signatures
 - Signature lifetime and permissions are stored in the policy rather than the URL
- Container, Queue, or Table can have up to five stored access policies

SAS token generation from a stored access policy



Lab: Authenticating to and querying Microsoft Graph by using MSAL and .NET SDKs

Duration



Lab sign-in information

