Microsoft

AZ-203.4 Module 01: Implementing authentication

Prashanth Kumar

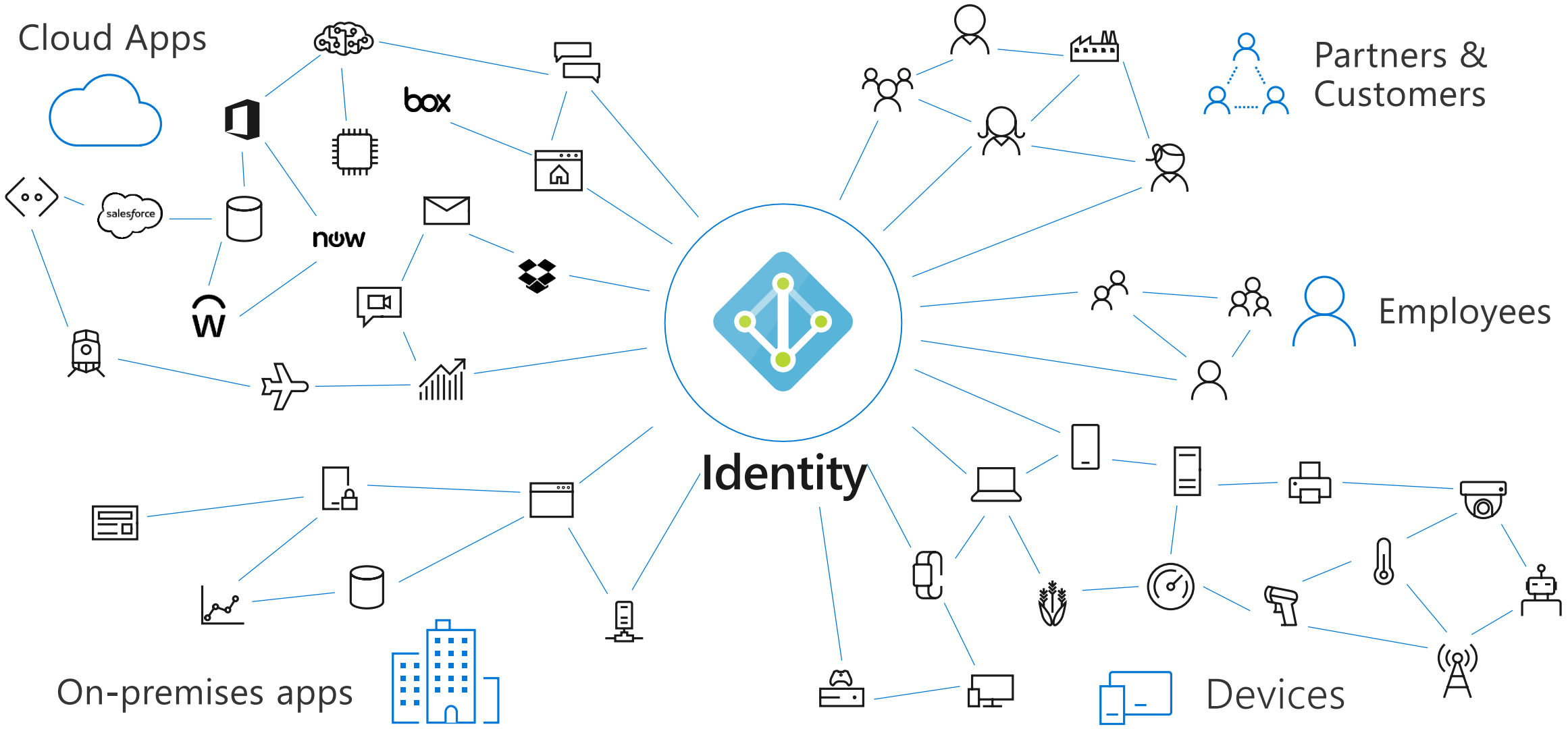
Microsoft

Topics

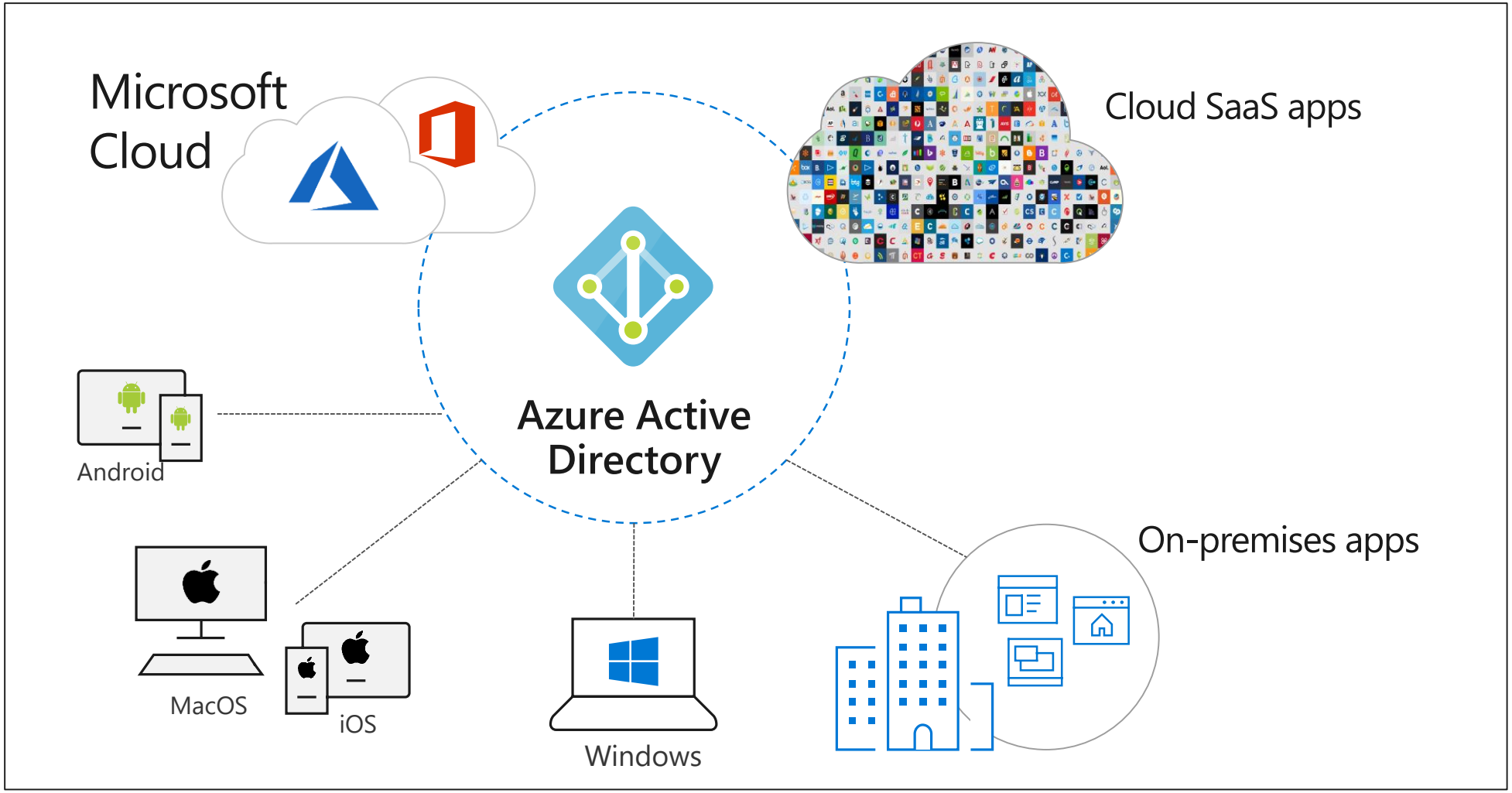
- Microsoft identity platform
- Implement OAuth 2.0 authentication
- Implement managed identity
- Implement certificate-based authentication
- Implement Azure Multi-Factor Authentication

Lesson 01: Microsoft identity platform

Identity as the control plane



Azure Active Directory

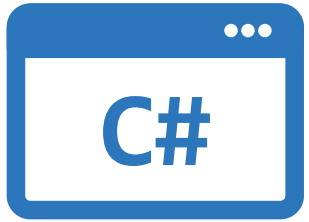


Active Directory Authentication Library (ADAL)

- 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
  - Objective C
  - Java
  - Python

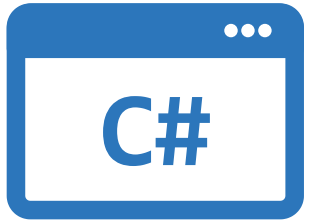
Creating an authentication context by using ADAL

```
string tenant = "contoso.onmicrosoft.com";
string authority = $"https://login.microsoftonline.com/{tenant}";

// Create authentication context using AAD authority
var context = new AuthenticationContext(
    authority,
    new FileCache()
);
```



Acquiring an Azure AD token by using ADAL

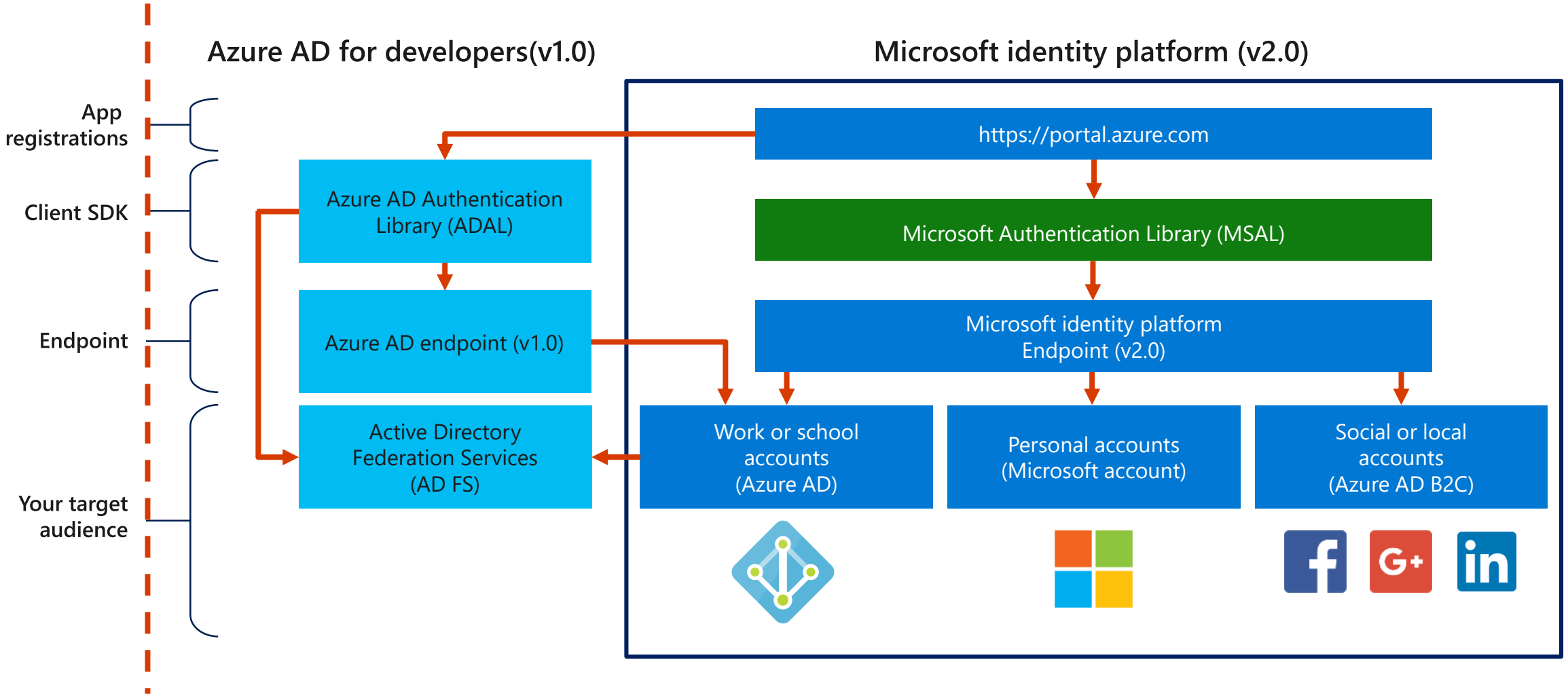
```
string clientId = "00000000-0000-0000-0000-000000000000";
string redirectUri = "https://login.microsoftonline.com/common/oauth2/nativeclient";
string resourceId = "https://graph.windows.net/";

// Parameters for acquiring token
var params = new PlatformParameters(PromptBehavior.Never);

// Acquire token action
await authContext.AcquireTokenAsync(
    resourceId,
    clientId,
    redirectUri,
    params
);
```



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



Microsoft identity platform overview

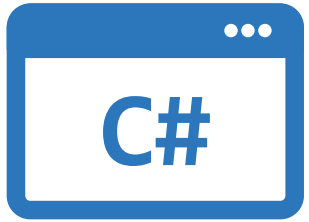
- An evolution of the Azure Active Directory (Azure AD) identity service and developer platform
- A full-featured identity platform that provides:
  - An authentication service
  - Open-source libraries
  - Application registration and configuration
  - Full developer documentation
  - Code samples
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

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:
  - Microsoft .NET
  - JavaScript
  - Android
  - iOS

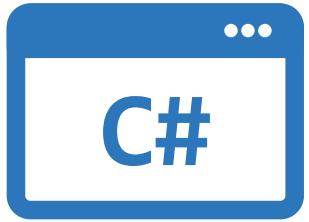
Creating an authentication context by using MSAL

```
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();
```



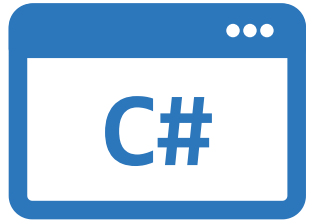
Acquiring a token interactively using MSAL

```
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;
```



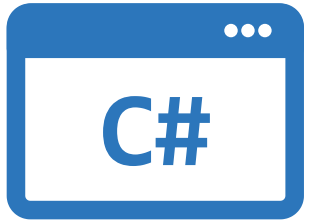
Acquiring a token silently using MSAL

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

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

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



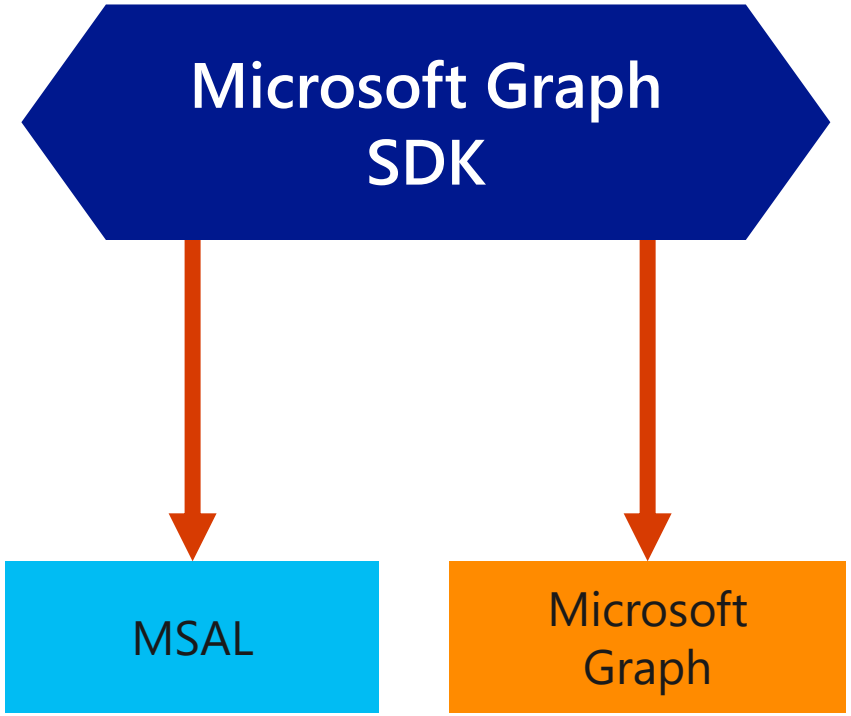
Get user profile using MSAL

```
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);
```



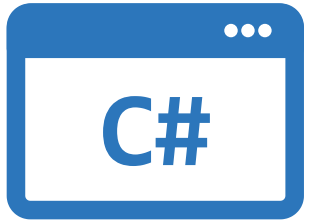
Microsoft Graph authentication provider library

- Referred to as Microsoft Graph software development kit (SDK)
- Wrapper for MSAL library:
  - Provides authentication provider helpers
  - Uses MSAL "under the hood"
  - Helpers automatically acquires tokens on your behalf
  - Reduces the complexity of using Microsoft Graph in your application
- Fluent API to issue requests to the Microsoft Graph
  - Automatically-generated properties and methods for endpoints in Microsoft Graph
  - Supports batching and paging



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
);
```

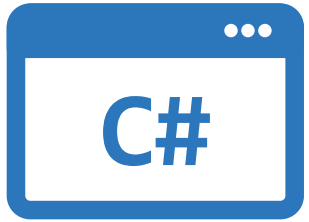


Authentication providers

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

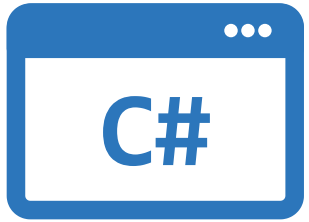
Using device code provider

```
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

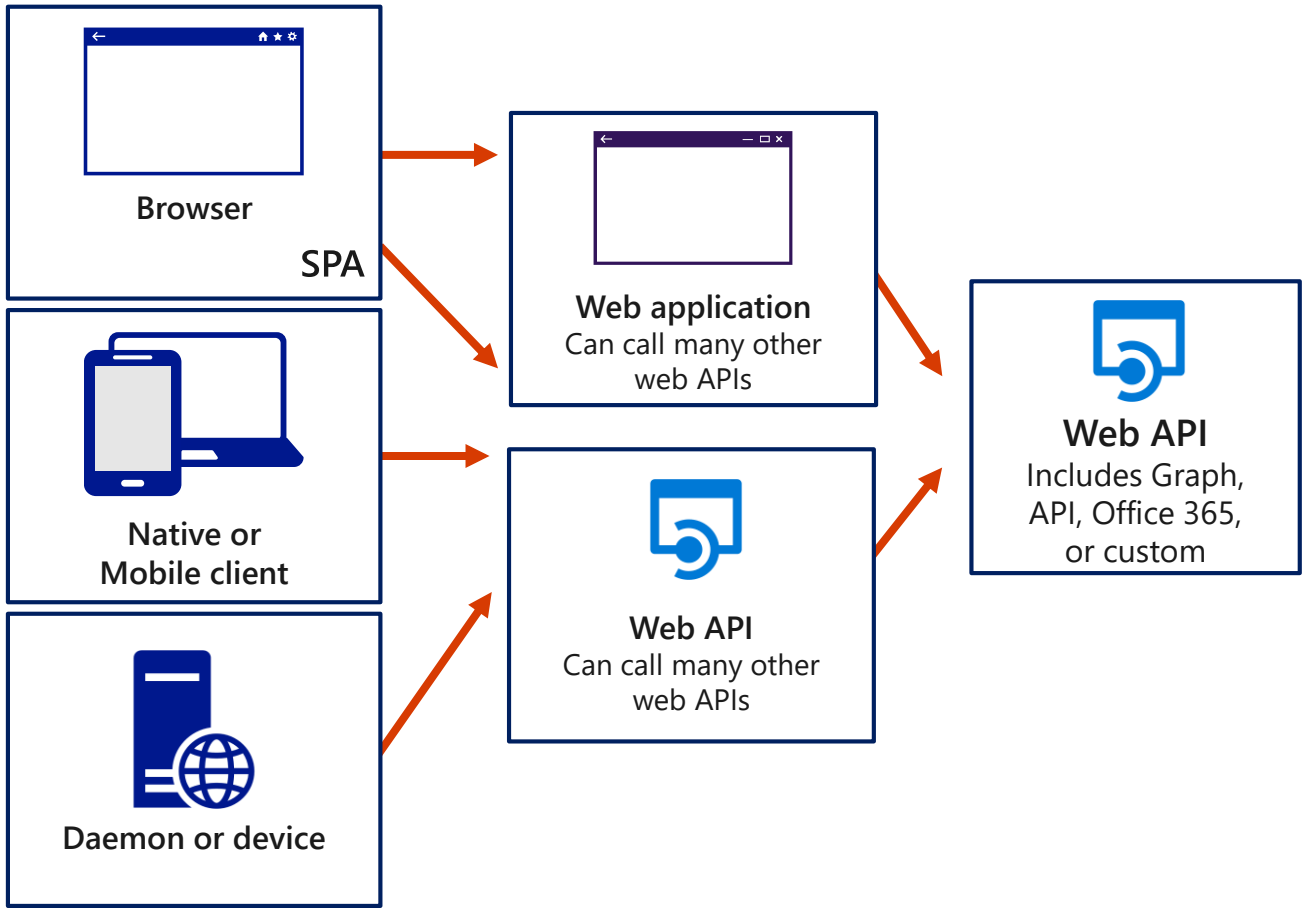
```
string clientId = "00000000-0000-0000-0000-000000000000";
string tenant = "contoso.onmicrosoft.com";
var scopes = new List<string> { "user.read" };

// 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
);
```



Application types in Azure AD

The selected application type determines the authentication scenario:

- Single-page application (SPA)
- Web browser to web application
- Native application to web API
- Web application to web API
- Daemon or server application to web API



Application types in Azure AD: 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
  - Multi-tenant: 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

Application types in Azure AD: Authentication endpoints

Multi-tenant applications (the same for all tenants):

https://login.microsoftonline.com/common

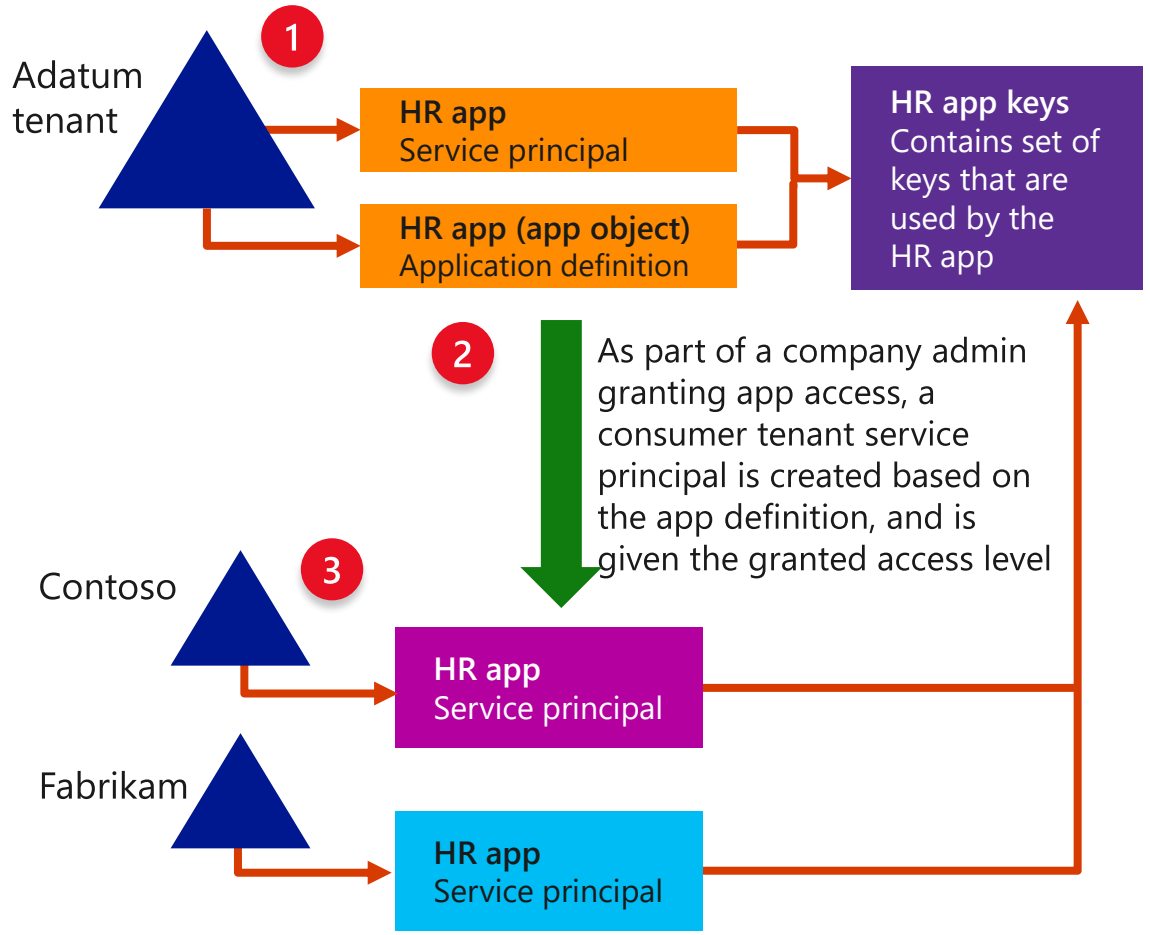
Single-tenant applications (tenant-specific):

https://login.microsoftonline.com/contoso.onmicrosoft.com

Application and service principal objects in Azure AD

Azure AD object model for authenticating a multi-tenant app:

- Adatum: the tenant of the app owner
- Contoso: a tenant of the app consumer
- Fabrikam: the tenant of the app consumer



Permissions and consent in Azure AD

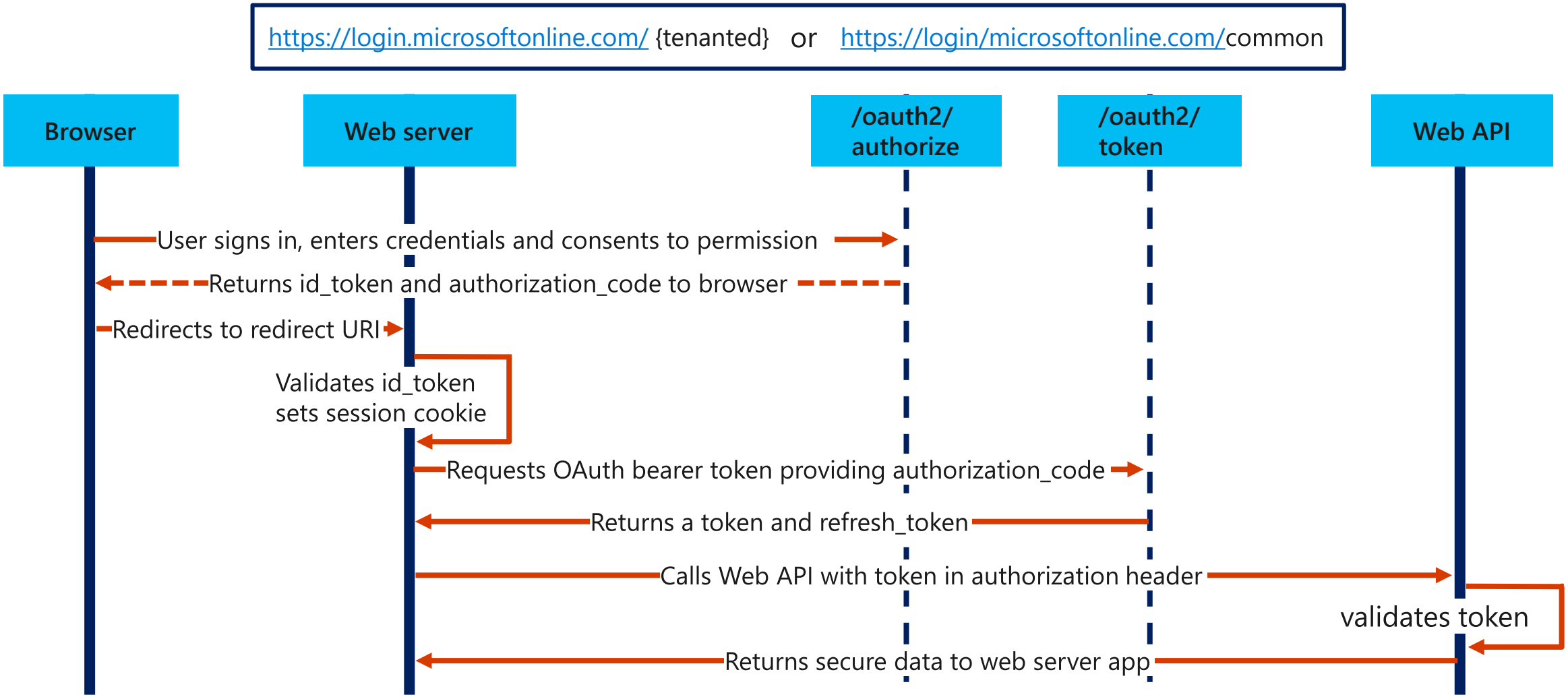
- Types of permissions:
  - **Delegated** – used by apps that run with a signed-in user present
  - **Application** – used by apps that run without a signed-in user present
- Effective permissions:
  - **Delegated permissions –** the least privileged intersection of the delegated permissions and the permissions of the currently signed-in user
  - **Application permissions –** full level of privileges implied by the permission
- Types of consent:
  - **Static user consent** – an app must have already specified all the permissions it needs in its configuration in Azure AD
  - **Dynamic user consent** – an app can request an additional set of permissions during OAuth 2.0 authorization flow (specific to Azure AD v2 app model)
  - **Admin consent** – required when app needs high-privilege permissions

Permissions and consent in Azure AD: best practices

- Client best practices:
  - Only request permissions that your app needs
  - Choose between delegated and application permissions based on the scenario your app supports
  - Always use delegated permissions if the call is being made on behalf of a user
  - Only use application permissions if the app is noninteractive
  - When using app-based on the v2.0 endpoint, always set the static permissions to be the superset of the dynamic permissions that you request at runtime
- Resource/API best practices:
  - Resources that expose APIs should define permissions that are specific to the data or actions that they are protecting
  - Resources should explicitly define Read and ReadWrite permissions separately
  - Resources should mark any permissions that allow access to data across user boundaries as Admin permissions
  - Resources should follow the naming pattern **Subject.Permission[.Modifier]**

Lesson 02: Implement OAuth 2.0 authentication

Authorize access to web applications by using OpenID Connect



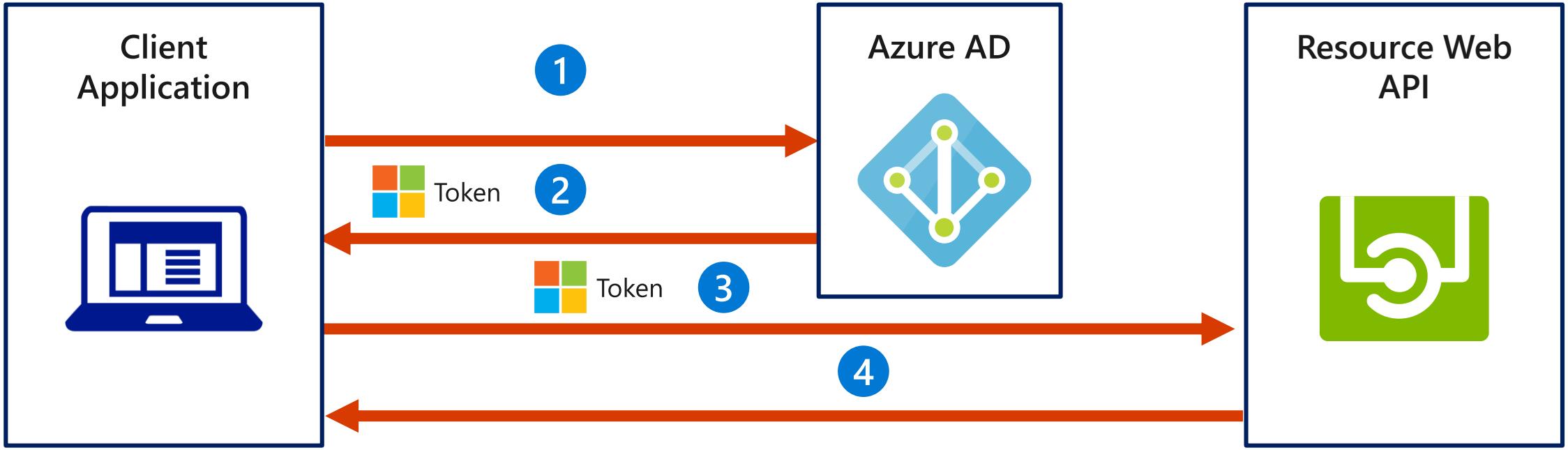
Understanding the OAuth 2.0 implicit grant flow in Azure AD

- 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

Service-to-service calls using client credentials



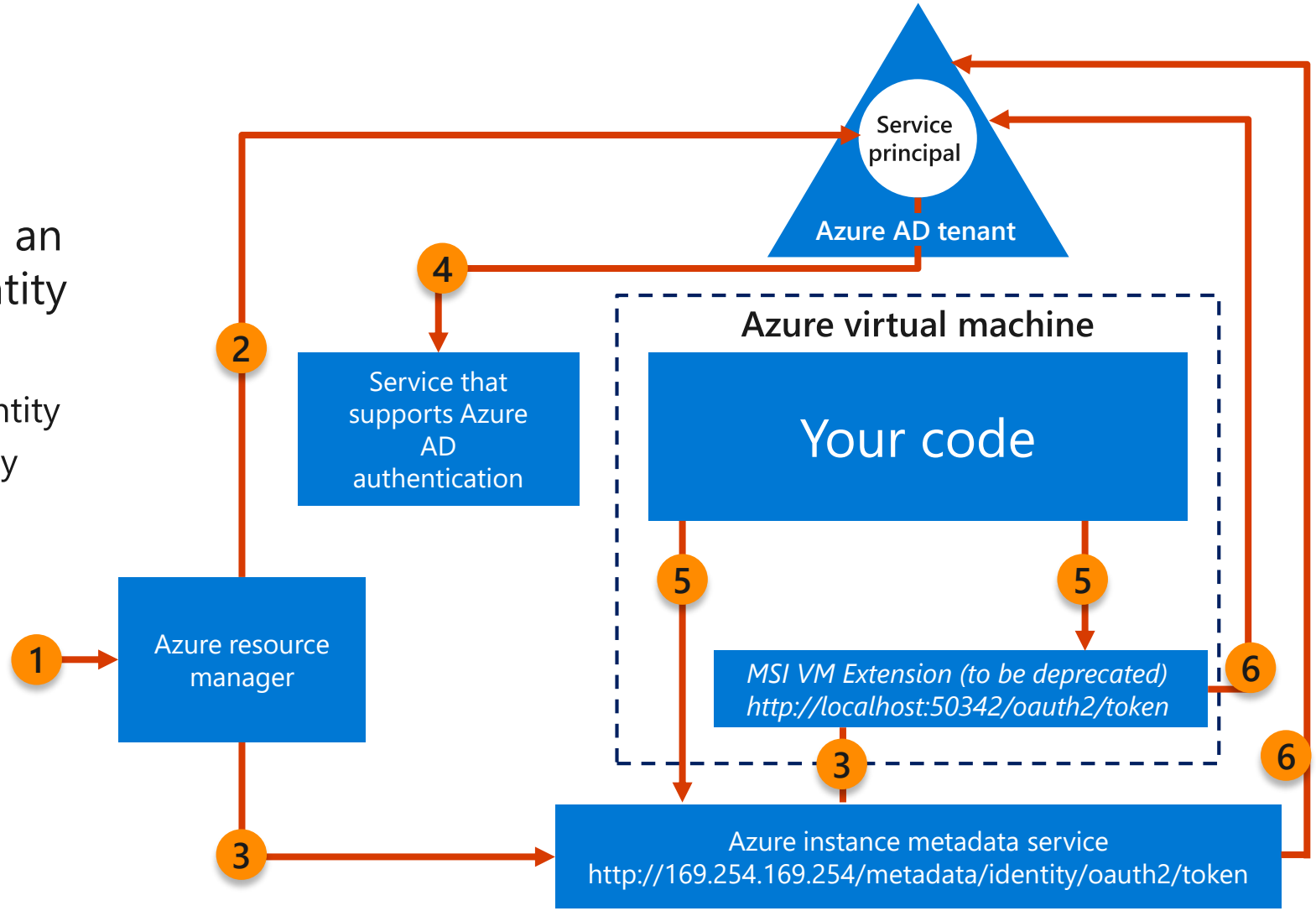
How the client credentials grant flow works in Azure AD:

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

Lesson 03: Implement managed identity

Overview of managed identities for Azure resources

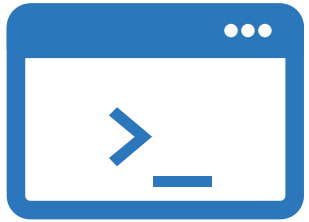
- Managed identities:
  - Is a feature of Azure AD
  - Provides Azure services with an automatically managed identity
  - Can be provisioned as:
    - A system-assigned managed identity
    - A user-assigned managed identity
  - Functionality leverages:
    - Client id
    - Principal id
    - Azure Instance Metadata Service



Configure managed identities for Azure resources on an Azure VM by using Azure CLI

System-assigned managed identity

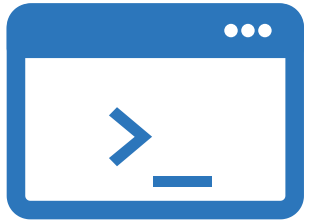
```
az login

az group create --name myResourceGroup --location westus

az vm create --resource-group myResourceGroup --name myVM --image win2016datacenter --
generate-ssh-keys --assign-identity --admin-username azureuser --admin-password
myPassword12

az vm identity assign -g myResourceGroup -n myVm
```



Configure managed identities for Azure resources on an Azure VM by using Azure CLI (continued)

User-assigned managed identity

```
az login

az group create --name myResourceGroup --location westus

az identity create -g myResourceGroup -n myUserAssignedIdentity

az vm create --resource-group myResourceGroup --name myVM --image win2016datacenter `
    --generate-ssh-keys --assign-identity --admin-username azureuser `
    --admin-password myPassword12 --assign-identity myUserAssignedIdentity

az vm identity assign -g myResourceGroup -n myVM `
    --identities myUserAssignedIdentity
```

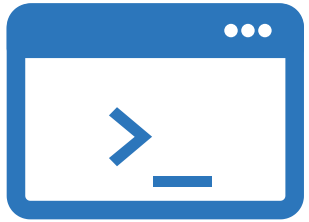


How to use managed identities for Azure resources on an Azure VM to acquire an access token

Get a token by using HTTP

```
GET 'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https://management.azure.com/' HTTP/1.1 Metadata: true

HTTP/1.1 200 OK
Content-Type: application/json
{
    "access_token": "eyJ0eXAi...",
    "refresh_token": "",
    "expires_in": "3599",
    "expires_on": "1506484173",
    "not_before": "1506480273",
    "resource": "https://management.azure.com/",
    "token_type": "Bearer"
}
```

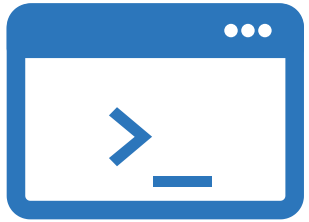
Consider token caching, error handling, and retry guidance



Assign a managed identity access to a resource by using Azure CLI

Use to assign a managed identity access to another resource role-based access control (RBAC)

```
az login

spID=$(az resource list -n myVM --query [*].identity.principalId --out tsv)

spID=$(az resource list -n DevTestVMSS --query [*].identity.principalId --out tsv)

az role assignment create --assignee $spID --role 'Reader' --scope
/subscriptions/<mySubscriptionID>/resourceGroups/<myResourceGroup>/providers/Microsoft.
Storage/storageAccounts/myStorageAcct
```



Lesson 04: Implement certificate-based authentication

Certificate-based authentication

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

Certificate-based authentication (continued)

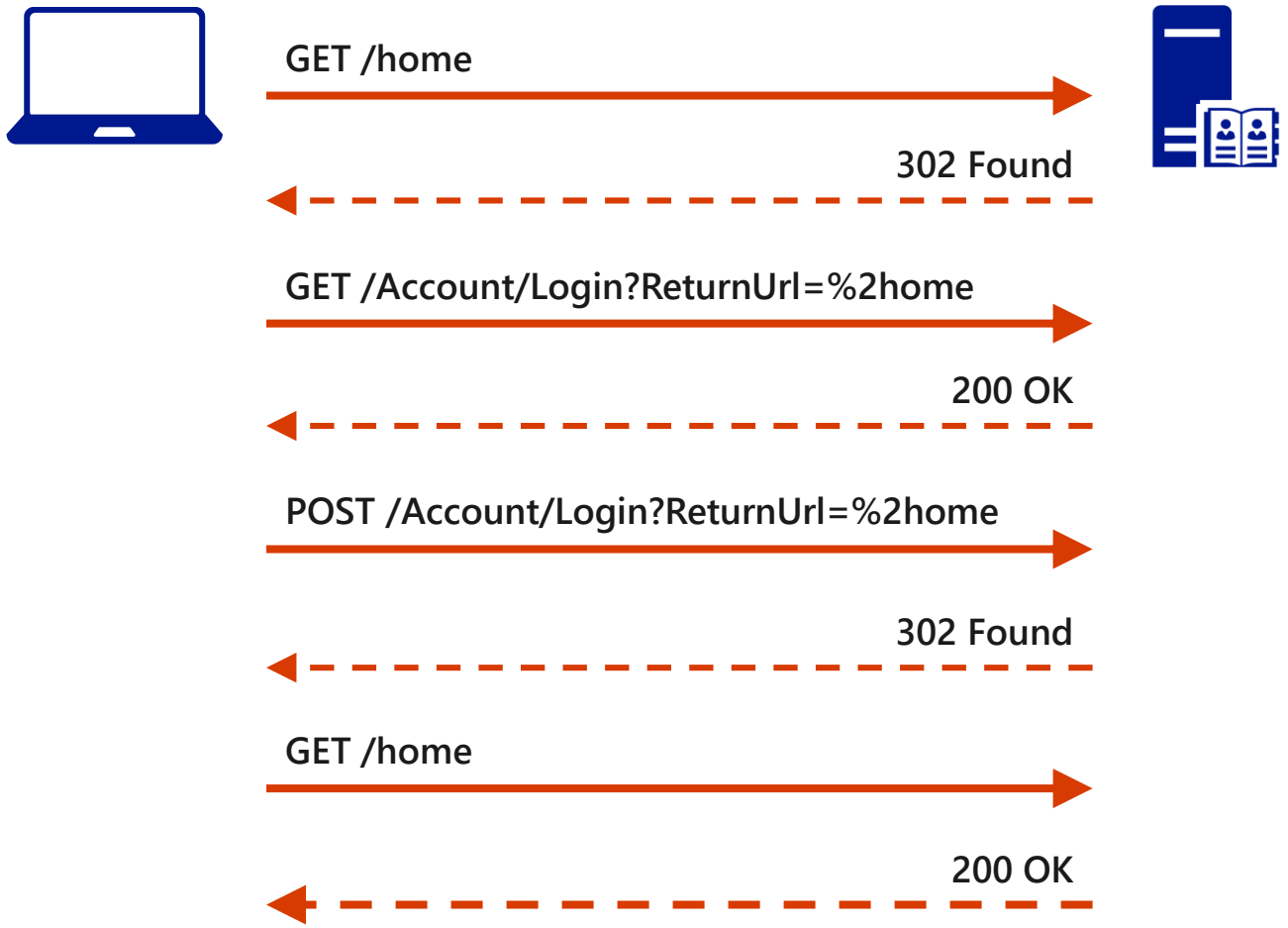
In Azure Active Directory, certificate-based authentication can be used to connect to:

- 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

Legacy authentication methods: forms-based authentication

- Uses an HTML form to send the user's credentials to the server
- Appropriate only for web APIs that are called from a web application
  - Requires user to interact with form
- Significant disadvantages include:
  - Requires a browser client to use the HTML form
  - Requires measures to prevent cross-site request forgery (CSRF)
  - User credentials are sent in plaintext as part of an HTTP request

Legacy authentication methods: forms-based authentication in .NET



Legacy authentication methods: Windows-based authentication

- Enables users to log in with their Windows credentials by using **Kerberos** or **NTLM**
  - Best suited for intranet environment
- Client sends credentials in the **Authorization** header
- Significant disadvantages include:
  - Difficult to use in internet applications without exposing the entire user directory
  - Can’t be used in Bring Your Own Device (BYOD) scenarios
  - Requires Kerberos or Integrated Windows Authentication (NTLM) support in the client browser or device
  - The client must be joined to the Active Directory domain

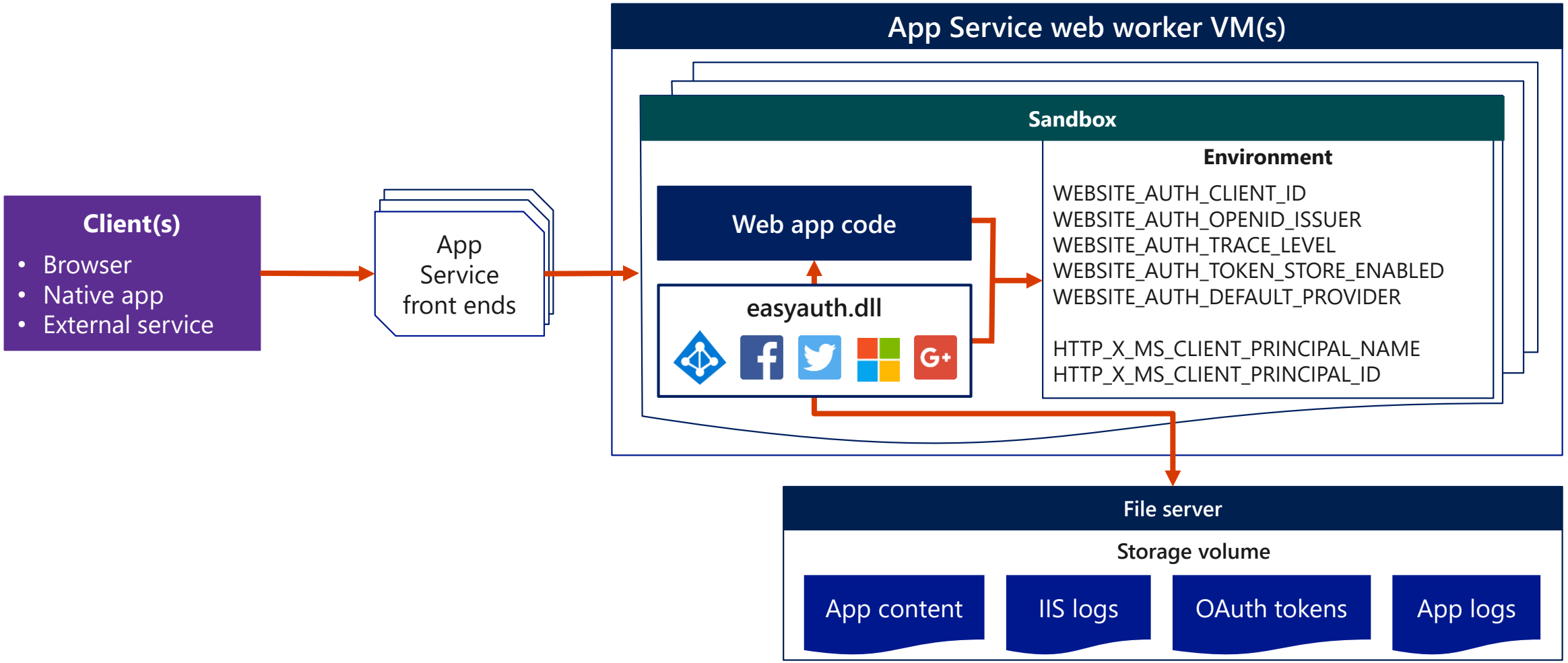
Token-based authentication: Claims-based authentication in .NET

- Historically, ASP.NET applications used form-based authentication
- ASP.NET Identity implements two core features, making it ideal for token-based authentication:
  - A provider model for logins: facilitates adding, removing, or replacing identity providers
  - Claims-based authentication: represents user’s identity as a set of claims

Token-based authentication: App Service authentication and authorization

- Azure App Service provides built-in authentication and authorization support
- Requires minimal or no code in existing applications
- Authentication and authorization module runs in the same sandbox as application code
- When enabled, every incoming HTTP request passes through the auth module before being handled by application code

Token-based authentication: App Service authentication and authorization (continued)



Lesson 05: Implement Azure Multi-Factor Authentication

Multi-factor authentication: factors in authentication

- Any **proof** that you are who you claim to be is called a **factor**
- **Factors** include:
  - A physical badge from the company
  - Knowledge of the answers to security questions
  - A mobile device, registered with the company, that can receive notifications, phone calls, or SMS messages
  - Physical appearance that can be captured by a camera device
  - Fingerprint that could be captured by a biometric scanner

Multi-factor authentication

- Factors are split into three categories:
  - **Knowledge** – Something that only the user knows (security questions, password, or PIN)
  - **Possession –** Something that only the user has (corporate badge, mobile device, or security token)
  - **Inherence –** Something that only the user is (fingerprint, face, voice, or iris)
- The practice of using two or more factors when authenticating is referred to as **multi-factor authentication**
  - It is recommended to have a factor from two out of three of the categories described above

Multi-factor authentication with Azure AD

- Two-step verification solution that is built in to Azure AD
- Can be enabled in two ways:
  - Enabled manually for each user by the account administrator
  - Enabled automatically by using a conditional access policy that is triggered by risky login behavior

Multi-factor authentication with Azure AD: authentication methods

Method	Description
Call to phone	Places an automated voice call. The user answers the call and presses # on the phone keypad to authenticate. The phone number is not synchronized to on-premises Active Directory.
Text message to phone	Sends a text message that contains a verification code. The user is prompted to enter the verification code into the sign-in interface. This process is called one-way SMS. Two-way SMS means that the user must text back a particular code.
Notification through mobile app	Sends a push notification to your phone or registered device. The user views the notification and selects **Verify** to complete the verification.
Verification code from mobile app	The Microsoft Authenticator app generates a new OAuth verification code every 30 seconds. The user enters the verification code into the sign-in interface.

Microsoft

Review

- Microsoft identity platform
- Implement OAuth 2.0 authentication
- Implement managed identity
- Implement certificate-based authentication
- Implement Azure Multi-Factor Authentication