100 XP

Introduction

3 minutes

The Microsoft identity platform for developers is a set of tools which includes authentication service, open-source libraries, and application management tools.

After completing this module, you'll be able 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

Next unit: Explore the Microsoft identity platform

Continue

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Explore the Microsoft identity platform

3 minutes

The Microsoft identity platform helps you build applications your users and customers can sign in to using their Microsoft identities or social accounts, and provide authorized access to your own APIs or Microsoft APIs like Microsoft Graph.

There are several components that make up the Microsoft identity platform:

- **OAuth 2.0 and OpenID Connect standard-compliant authentication service** enabling developers to authenticate several identity types, including:
  - Work or school accounts, provisioned through Azure Active Directory
  - Personal Microsoft account, like Skype, Xbox, and Outlook.com
  - Social or local accounts, by using Azure Active Directory B2C
- **Open-source libraries**: Microsoft Authentication Libraries (MSAL) and support for other standards-compliant libraries
- **Application management portal**: A registration and configuration experience in the Azure portal, along with the other Azure management capabilities.
- **Application configuration API and PowerShell**: Programmatic configuration of your applications through the Microsoft Graph API and PowerShell so you can automate your DevOps tasks.

For developers, the Microsoft identity platform offers integration of modern innovations in the identity and security space like passwordless authentication, step-up authentication, and Conditional Access. You don't need to implement such functionality yourself: applications integrated with the Microsoft identity platform natively take advantage of such innovations.

Next unit: Explore service principals

Continue

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Explore service principals

3 minutes

To delegate Identity and Access Management functions to Azure Active Directory, an application must be registered with an Azure Active Directory tenant. When you register your application with Azure Active Directory, you're creating an identity configuration for your application that allows it to integrate with Azure Active Directory. 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, an application object (the globally unique instance of the app) as well as a service principal object are automatically created in your home tenant. You also have a globally unique ID for your app (the app or client ID). In the portal, you can then add secrets or certificates and scopes to make your app work, customize the branding of your app in the sign-in dialog, and more.

> **Note**
>
> You can also create service principal objects in a tenant using Azure PowerShell, Azure CLI, Microsoft Graph, and other tools.

Application object

An Azure Active Directory application is defined by its one and only application object, which resides in the Azure Active Directory tenant where the application was registered (known as the application's "home" tenant). An application object is used as a template or blueprint to create one or more service principal objects. A service principal is created in every tenant where the application is used. Similar to a class in object-oriented programming, the application object has some static properties that are applied to all the created service principals (or application instances).

The application object describes three aspects of an application: how the service can issue tokens in order to access the application, resources that the application might need to access, and the actions that the application can take.

The Microsoft Graph Application entity defines the schema for an application object's properties.

Service principal object

To access resources that are secured by an Azure Active Directory tenant, the entity that requires access must be represented by a security principal. This is true for both users (user principal) and applications (service principal).

The security principal defines the access policy and permissions for the user/application in the Azure Active Directory tenant. This enables core features such as authentication of the user/application during sign-in, and authorization during resource access.

There are three types of service principal:

- **Application -** The type of service principal is the local representation, or application instance, of a global application object in a single tenant or directory. A service principal is created in each tenant where the application is used and references the globally unique app object. The service principal object defines what the app can actually do in the specific tenant, who can access the app, and what resources the app can access.
- **Managed identity -** This type of service principal is used to represent a managed identity. Managed identities provide an identity for applications to use when connecting to resources that support Azure Active Directory authentication. When a managed identity is enabled, a service principal representing that managed identity is created in your tenant. Service principals representing managed identities can be granted access and permissions, but cannot be updated or modified directly.
- **Legacy -** This type of service principal represents a legacy app, which is an app created before app registrations were introduced or an app created through legacy experiences. A legacy service principal can have credentials, service principal names, reply URLs, and other properties that an authorized user can edit, but does not have an associated app registration. The service principal can only be used in the tenant where it was created.

Relationship between application objects and service principals

The application object is the *global* representation of your application for use across all tenants, and the service principal is the *local* representation for use in a specific tenant. The application object serves as the template from which common and default properties are *derived* for use in creating corresponding service principal objects.

An application object has:

- A 1:1 relationship with the software application, and
- A 1:many relationship with its corresponding service principal object(s).

A service principal must be created in each tenant where the application is used, enabling it to establish an identity for sign-in and/or access to resources being secured by the tenant. A single-tenant application has only one service principal (in its home tenant), created and consented for use during application registration. A multi-tenant application also has a service principal created in each tenant where a user from that tenant has consented to its use.

Next unit: Discover permissions and consent

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Discover permissions and consent

3 minutes

Applications that integrate with the Microsoft identity platform follow an authorization model that gives users and administrators control over how data can be accessed.

The Microsoft identity platform implements the OAuth 2.0 authorization protocol. OAuth 2.0 is a method through which a third-party app can access web-hosted resources on behalf of a user. Any web-hosted resource that integrates with the Microsoft identity platform has a resource identifier, or *application ID URI*.

Here are some examples of Microsoft web-hosted resources:

- Microsoft Graph: `https://graph.microsoft.com`
- Microsoft 365 Mail API: `https://outlook.office.com`
- Azure Key Vault: `https://vault.azure.net`

The same is true for any third-party resources that have integrated with the Microsoft identity platform. Any of these resources also can define a set of permissions that can be used to divide the functionality of that resource into smaller chunks. When a resource's functionality is chunked into small permission sets, third-party apps can be built to request only the permissions that they need to perform their function. Users and administrators can know what data the app can access.

In OAuth 2.0, these types of permission sets are called *scopes*. They're also often referred to as *permissions*. In the Microsoft identity platform, a permission is represented as a string value. An app requests the permissions it needs by specifying the permission in the `scope` query parameter. Identity platform supports several well-defined OpenID Connect scopes as well as resource-based permissions (each permission is indicated by appending the permission value to the resource's identifier or application ID URI). For example, the permission string `https://graph.microsoft.com/Calendars.Read` is used to request permission to read users calendars in Microsoft Graph.

An app most commonly requests these permissions by specifying the scopes in requests to the Microsoft identity platform authorize endpoint. However, some high-privilege permissions can be

granted only through administrator consent. They can be requested or granted by using the administrator consent endpoint.

> **Note**
>
> In requests to the authorization, token or consent endpoints for the Microsoft Identity platform, if the resource identifier is omitted in the scope parameter, the resource is assumed to be Microsoft Graph. For example, `scope=User.Read` is equivalent to `https://graph.microsoft.com/User.Read`.

Permission types

The Microsoft identity platform supports two types of permissions: *delegated permissions* and *application permissions*.

- **Delegated permissions** are used by apps that have a signed-in user present. For these apps, either the user or an administrator consents to the permissions that the app requests. The app is delegated with the permission to act as a signed-in user when it makes calls to the target resource.
- **Application permissions** are used by apps that run without a signed-in user present, for example, apps that run as background services or daemons. Only an administrator can consent to application permissions.

Consent types

Applications in Microsoft identity platform rely on consent in order to gain access to necessary resources or APIs. There are a number of kinds of consent that your app may need to know about in order to be successful. If you are defining permissions, you will also need to understand how your users will gain access to your app or API.

There are three consent types: *static user consent*, *incremental and dynamic user consent*, and *admin consent*.

Static user consent

In the static user consent scenario, you must specify all the permissions it needs in the app's configuration in the Azure portal. If the user (or administrator, as appropriate) has not granted consent for this app, then Microsoft identity platform will prompt the user to provide consent at this time. Static permissions also enables administrators to consent on behalf of all users in the organization.

While static permissions of the app defined in the Azure portal keep the code nice and simple, it presents some possible issues for developers:

- The app needs to request all the permissions it would ever need upon the user's first sign-in. This can lead to a long list of permissions that discourages end users from approving the app's access on initial sign-in.
- The app needs to know all of the resources it would ever access ahead of time. It is difficult to create apps that could access an arbitrary number of resources.

Incremental and dynamic user consent

With the Microsoft identity platform endpoint, you can ignore the static permissions defined in the app registration information in the Azure portal and request permissions incrementally instead. You can ask for a minimum set of permissions upfront and request more over time as the customer uses additional app features.

To do so, you can specify the scopes your app needs at any time by including the new scopes in the `scope` parameter when requesting an access token - without the need to pre-define them in the application registration information. If the user hasn't yet consented to new scopes added to the request, they'll be prompted to consent only to the new permissions. Incremental, or dynamic consent, only applies to delegated permissions and not to application permissions.

> **ⓘ Important**
>
> Dynamic consent can be convenient, but presents a big challenge for permissions that require admin consent, since the admin consent experience doesn't know about those permissions at consent time. If you require admin privileged permissions or if your app uses dynamic consent, you must register all of the permissions in the Azure portal (not just the subset of permissions that require admin consent). This enables tenant admins to consent on behalf of all their users.

Admin consent

Admin consent is required when your app needs access to certain high-privilege permissions. Admin consent ensures that administrators have some additional controls before authorizing apps or users to access highly privileged data from the organization.

Admin consent done on behalf of an organization still requires the static permissions registered for the app. Set those permissions for apps in the app registration portal if you need an admin to give consent on behalf of the entire organization. This reduces the cycles required by the organization admin to set up the application.

Requesting individual user consent

In an OpenID Connect or OAuth 2.0 authorization request, an app can request the permissions it needs by using the scope query parameter. For example, when a user signs in to an app, the app sends a request like the following example. Line breaks are added for legibility.

HTTP Copy

```
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
```

The `scope` parameter is a space-separated list of delegated permissions that the app is requesting. Each permission is indicated by appending the permission value to the resource's identifier (the application ID URI). In the request example, the app needs permission to read the user's calendar and send mail as the user.

After the user enters their credentials, the Microsoft identity platform checks for a matching record of *user consent*. If the user hasn't consented to any of the requested permissions in the past, and if the administrator hasn't consented to these permissions on behalf of the entire organization, the Microsoft identity platform asks the user to grant the requested permissions.

Next unit: Discover conditional access

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Discover conditional access

3 minutes

The Conditional Access feature in Azure Active Directory offers one of several ways that you can use to secure your app and protect a service. Conditional Access enables developers and enterprise customers 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

How does Conditional Access impact an app?

In most common cases, Conditional Access does not change an app's behavior or require any changes from the developer. Only in certain cases when an app indirectly or silently requests a token for a service does an app require code changes to handle Conditional Access challenges. It may be as simple as performing an interactive sign-in request.

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

Conditional Access policies can be applied to the app and also a web API your app accesses. Depending on the scenario, an enterprise customer can apply and remove Conditional Access policies at any time. For your app to continue functioning when a new policy is applied, implement challenge handling.

Conditional Access examples

Some scenarios require code changes to handle Conditional Access whereas others work as is. Here are a few scenarios using Conditional Access to do multifactor authentication that gives

some insight into the difference.

- You are building a single-tenant iOS app and apply a Conditional Access policy. The app signs in a user and doesn't request access to an API. When the user signs in, the policy is automatically invoked and the user needs to perform multifactor authentication.
- You are building a native app that uses a middle tier service to access a downstream API. An enterprise customer at the company using this app applies a policy to the downstream API. When an end user signs in, the native app requests access to the middle tier and sends the token. The middle tier performs on-behalf-of flow to request access to the downstream API. At this point, a claims "challenge" is presented to the middle tier. The middle tier sends the challenge back to the native app, which needs to comply with the Conditional Access policy.

Next unit: Knowledge check

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

200 XP

Knowledge check

3 minutes

Check your knowledge

1. Which of the types of permissions supported by the Microsoft identity platform is used by apps that have a signed-in user present?

- ○ Delegated permissions

  ✔ That's correct. Delegated permissions are used by apps that have a signed-in user present. The app is delegated with the permission to act as a signed-in user when it makes calls to the target resource.

- ◉ Application permissions

  ✘ That's incorrect. Application permissions are used by apps that run without a signed-in user present, for example, apps that run as background services or daemons.

- ○ Both delegated and application permissions

2. Which of the following app scenarios require code to handle Conditional Access challenges?

- ○ Apps performing the device-code flow
- ○ Apps performing the on-behalf-of flow

  ✔ That's correct. Apps performing the on-behalf-of flow require code to handle Conditional Access challenges.

- ◉ Apps performing the Integrated Windows authentication flow

  ✘ That's incorrect. The Integrated Windows authentication flow allows applications on domain or Azure Active Directory (Azure AD) joined computers to acquire a token silently.

Next unit: Summary

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Summary

3 minutes

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

Module complete:

Unlock achievement

How are we doing? ☆ ☆ ☆ ☆ ☆