KodeKloud

Visit www.kodekloud.com to learn more.

Exploring the Microsoft Identity Platform

Introduction

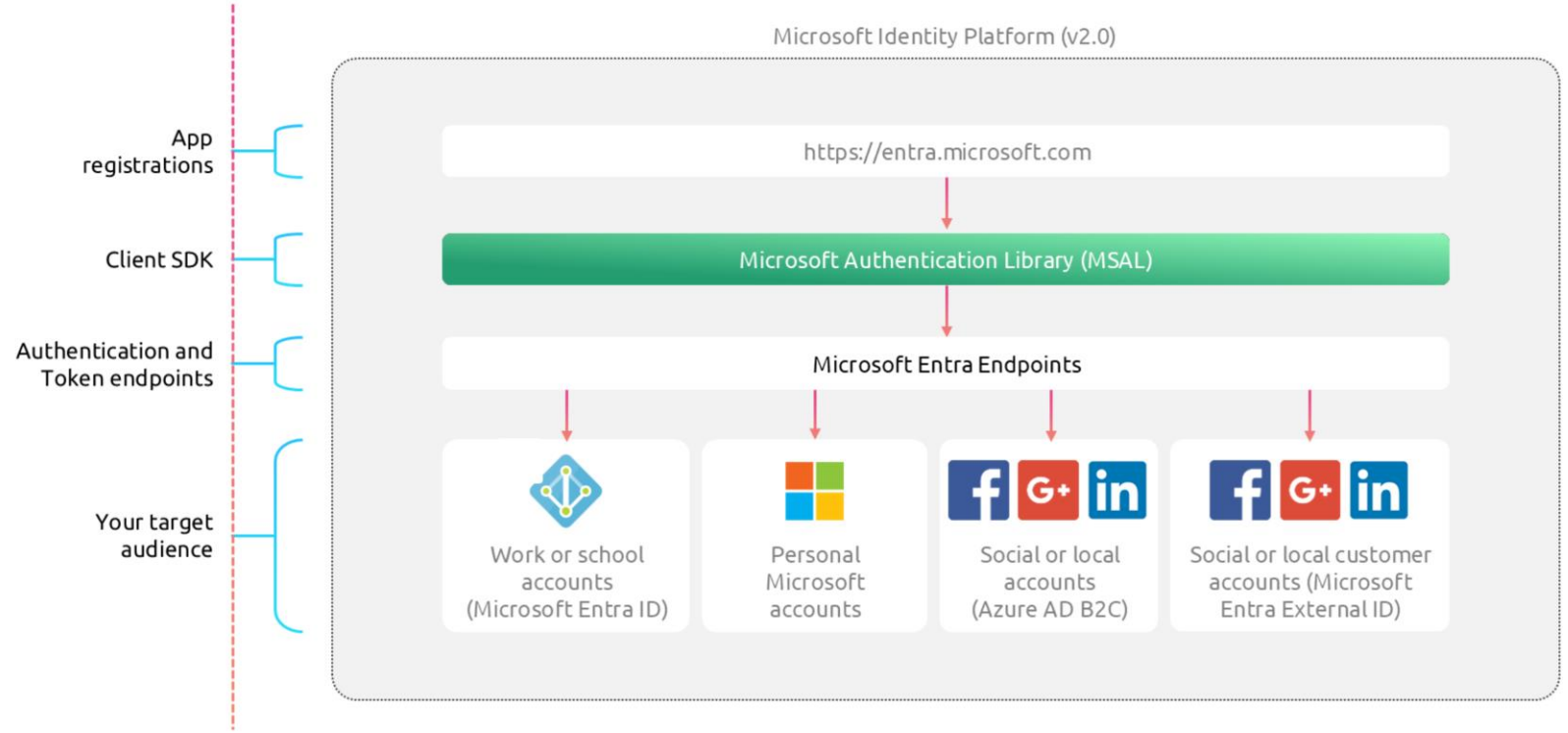
01 Identify Microsoft identity platform components

02 Describe three service principal types and their relation to application objects

03 Explain permissions, user consent, and the impact of conditional access

Microsoft Identity Platform

Microsoft Identity Platform

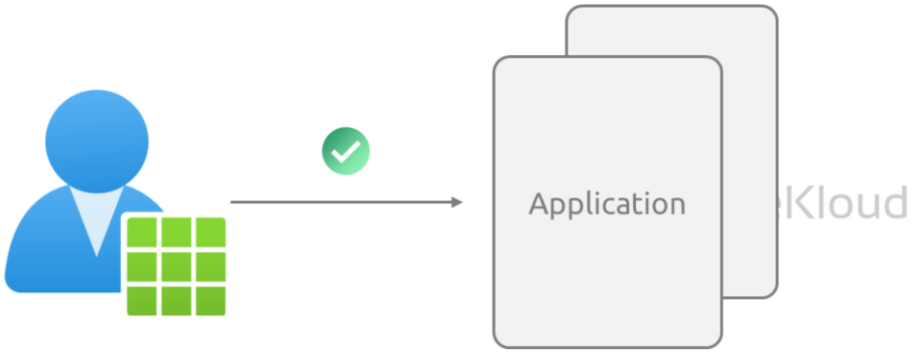


we'll explore the Microsoft Identity platform, which is integral to securing your applications and services on Azure.
First, let's discuss app registrations. This is where your journey begins, at the https://entra.microsoft.com. Here, you will register your applications to integrate with Microsoft's identity system, enabling sign-in for your users and requesting tokens to access resources.
We'll then dive into the Microsoft Authentication Library, or MSAL. This powerful SDK simplifies the process of implementing authentication in your app. It handles token acquisition and management, making it easier for your applications to authenticate users.

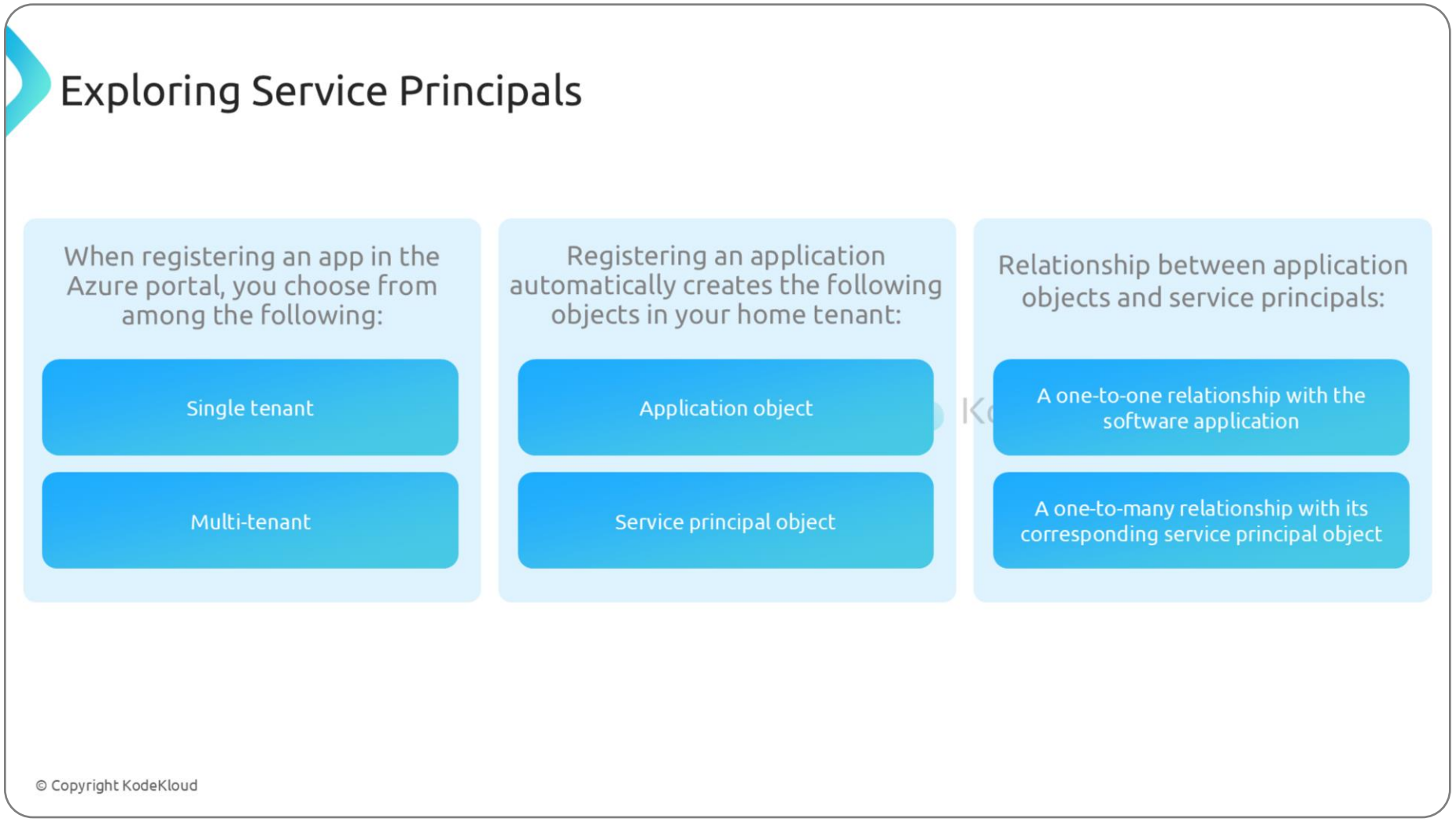
The next piece of the puzzle is understanding the Microsoft Entra endpoints. These endpoints are where MSAL interacts with to authenticate users and obtain tokens. They are crucial for implementing secure communication between your app and Microsoft's identity services.
Finally, we'll discuss your target audience. This is where you define who can use your app. Whether it's work or school accounts managed by Microsoft Entra ID, personal Microsoft accounts, or social accounts through Azure AD B2C, you have the flexibility to support a variety of users.
By the end of this module, you'll have a thorough understanding of how to implement user authentication and authorization using Microsoft's identity platform, setting a strong foundation for the security of your Azure solutions."

Exploring Service Principals

Exploring Service Principals



Exploring service principles In this slide, we explore service principles in Azure A service principle is essentially an identity for an application that allows it to access resources in Azure So whenever you register an application in Azure, certain objects are created behind the scene to manage this access



Let's dive into the concept of Service Principals in Azure, using an analogy to make things clearer.
Imagine you've built a new apartment complex (this is your software application) and now you need a property manager (this is your Service Principal) to handle access and maintenance tasks. When you register your apartment complex in Azure Portal, you decide if it's a single tenant, meaning it's exclusive to your own company's use, or multi-tenant, allowing other companies to use it too.
As soon as you complete the registration, two key objects are created in your Azure home tenant:
The Application object, which represents the blueprint or the structure of your apartment complex.

The Service Principal object, which is like the specific property manager assigned to your complex. There can be various types of managers such as one for your Application, another for Managed identities, and even a Legacy type.
The Application object has a one-to-one relationship with your software application, just like a blueprint to a specific building. On the other hand, it has a one-to-many relationship with the Service Principal objects, akin to how one building design could be managed by multiple property managers in different locations or scenarios.
This setup allows your application to have a broad reach while maintaining organized access and control across different environments

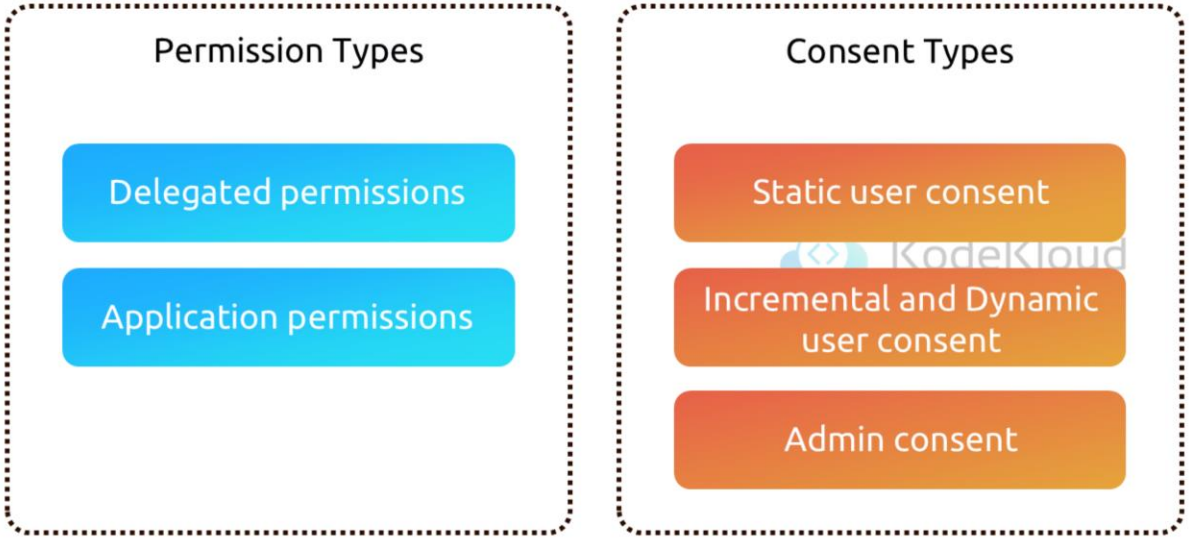
Permissions and Consent

Permissions and Consent

- Permissions are structured
- User Consent
- Ensuring Proper Authorization

Permissions and consent In this section, we will explore the concept of permissions and consent within Microsoft Identity platform Understanding how permissions are structured and how user consent is obtained is critical for securing applications and also ensuring proper authorization Let's dive into different types of permissions and consent available in Azure

Permissions and Consent



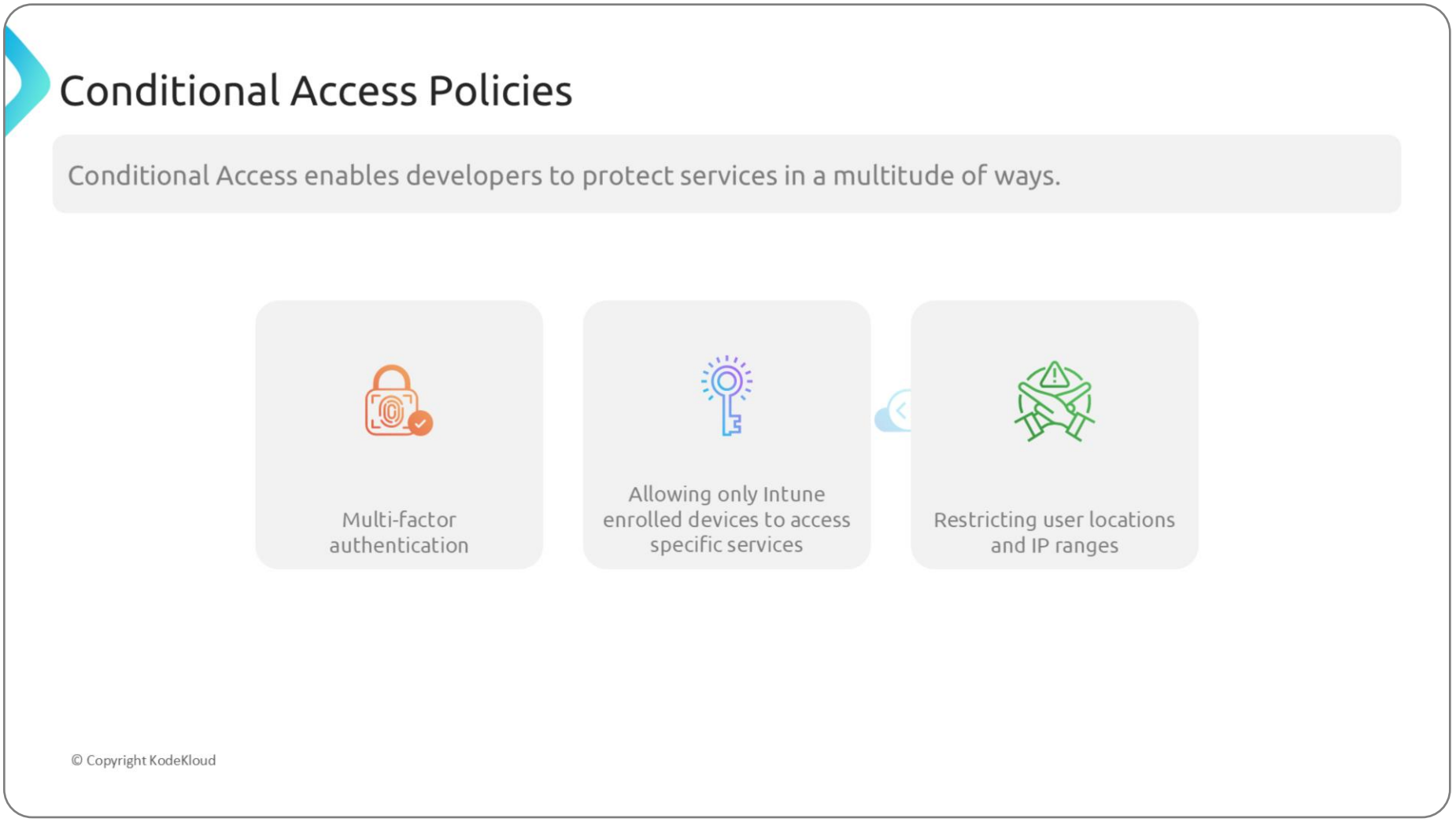
Let's continue with the analogy of the apartment complex to understand permissions and consent within the Microsoft Identity platform.
Think of permission types as keys to different areas of your apartment complex. Delegated permissions are like giving a key to a resident who can access the apartment while they are present. This is used by apps that require a user to be signed in. Application permissions are more like a master key given to maintenance staff to access the apartment when no one is home, used by apps that run as background services without a user present.
For consent types, consider static user consent as an agreement signed when a tenant first moves in, outlining what they can access. Incremental and dynamic user consent are like agreements that evolve, giving more access as the tenant uses new amenities over time. Lastly, admin consent is when the building management decides what keys and access can be granted, ensuring control over the entire building's security.
This setup ensures that both residents (users) and staff (applications) have the right level of access for their needs, while the management (admin) maintains overall control and security.

Permissions and Consent

```
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
```

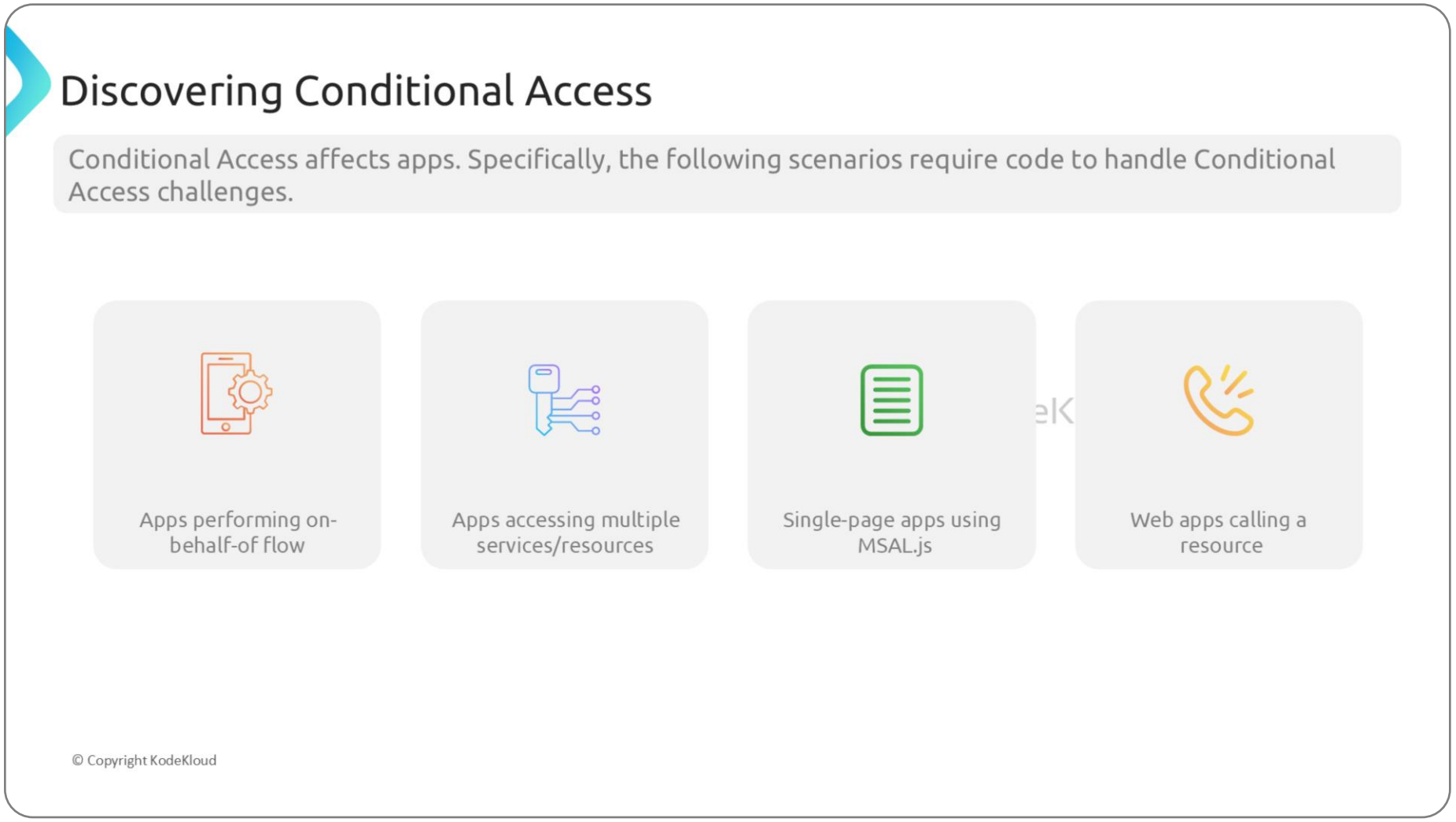

In this section of our course, we're looking at how applications request permissions through OpenID Connect or OAuth 2.0 authorization requests. To continue our apartment complex analogy, the 'scope' parameter is like a list of services the tenant asks to use. It's a space-separated list of permissions, like asking for access to the gym (calendars.read) and the mailbox (mail.send).
When a tenant tries to use a service, the property manager checks if they've been granted access before. If not, and if the building admin hasn't given blanket access to everyone, the tenant will be asked to agree to the terms of use for each service.
Remember, the scope in the request must match the permissions the app needs. If there's a mismatch, like a tenant asking for a service they're not entitled to, access will be denied. This process ensures that users' data is accessed appropriately and with their consent.

Conditional Access Policies



we're going to look at Conditional Access within Microsoft's identity services, a feature that helps safeguard the access to your applications in various ways.
Let's continue with our building analogy. Conditional Access is akin to the security measures in our apartment complex. It allows us to set rules on who can enter the building and when. For instance, multifactor authentication is like a security checkpoint where, besides the usual key, a tenant might need to provide an ID. Similarly, we might only allow people with a specific key card (Intune enrolled devices) to access certain floors or restrict access to the building based on where someone is coming from (user locations and IP ranges).

Sometimes, these security rules require changes to the way the building operates. For example, if a tenant's key card stops working due to a new security policy, they'll need to follow a new process to get it reactivated. This is like an app that needs to be updated to handle new Conditional Access challenges, such as performing an on-behalf-of flow or dealing with single-page apps using MSAL.js.
So, developers need to be ready to update their code to ensure that tenants can access the building smoothly, even when new security policies are put into place. This way, the integrity of the building's security is maintained without causing inconvenience to the residents.



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.

Implementing Authentication by Using MSAL

Introduction

01 Instantiate public and confidential client apps programmatically

02 Register an app with the Microsoft identity platform

03 Create an app that retrieves a token using the MSAL.NET library

04 Explain MSAL benefits, supported application types, and scenarios

Microsoft Authentication Library (MSAL)

Microsoft Authentication Library

Provides secure access to Microsoft Graph, other Microsoft APIs, third-party web APIs, or your own web API

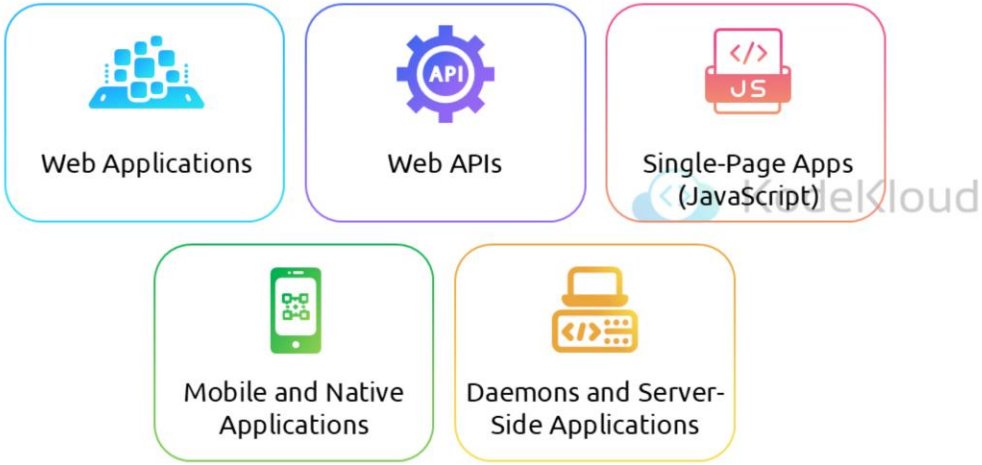
Supports different application architectures and platforms including .NET, JavaScript, Java, Python, Android, and iOS

we're going to delve into the Microsoft Authentication Library (MSAL) and its role in securing your applications by simplifying the process of acquiring tokens from Microsoft Identity platform.
Consider MSAL as your application's gatekeeper. Much like a hotel's concierge, MSAL manages the intricacies of authentication, so your app can focus on providing a seamless user experience. It offers a consistent API across a myriad of platforms including .NET, JavaScript, Java, Python, Android, and iOS.
Here's why MSAL is invaluable:
It abstracts away the complexities of OAuth, sparing you from low-level protocol handling.

It intelligently acquires tokens either on behalf of a user or as a daemon application, depending on the scenario.
MSAL maintains a token cache, automatically refreshing tokens as they near expiration, ensuring your app always has a valid token without manual intervention.
It aids in defining the audience for your application and can be configured via files, streamlining setup.
When things go awry, MSAL offers actionable exceptions, logging, and telemetry to help you pinpoint and resolve issues.
Whether you're developing a web app, a single-page application, a mobile or native application, or even service daemons and server-side apps, MSAL has you covered, ensuring secure access to necessary resources.

Microsoft Authentication Library

Application types and scenarios



we're going to delve into the Microsoft Authentication Library (MSAL) and its role in securing your applications by simplifying the process of acquiring tokens from Microsoft Identity platform.
Consider MSAL as your application's gatekeeper. Much like a hotel's concierge, MSAL manages the intricacies of authentication, so your app can focus on providing a seamless user experience. It offers a consistent API across a myriad of platforms including .NET, JavaScript, Java, Python, Android, and iOS.
Here's why MSAL is invaluable:
It abstracts away the complexities of OAuth, sparing you from low-level protocol handling.

It intelligently acquires tokens either on behalf of a user or as a daemon application, depending on the scenario.
MSAL maintains a token cache, automatically refreshing tokens as they near expiration, ensuring your app always has a valid token without manual intervention.
It aids in defining the audience for your application and can be configured via files, streamlining setup.
When things go awry, MSAL offers actionable exceptions, logging, and telemetry to help you pinpoint and resolve issues.
Whether you're developing a web app, a single-page application, a mobile or native application, or even service daemons and server-side apps, MSAL has you covered, ensuring secure access to necessary resources.

Microsoft Authentication Library

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	Application calls a service/web API, which, in turn, 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	Application signs in a user by using their username and password

Continuing with our exploration of the Microsoft Authentication Library (MSAL), let's understand the various authentication flows it supports for acquiring tokens, using an interactive or non-interactive approach.
Think of these authentication flows like different entrances to a secure building:
Authorization Code Flow: This is the main entrance used by residents (native and web apps) who show their ID (user sign-in) to get a token.
Client Credentials Flow: This is like a service entrance where the building's systems (service applications) operate without needing a resident to be present.

On-behalf-of Flow: Imagine a situation where a resident asks the doorman (a service or web API) to collect their mail (Microsoft Graph) for them.
Implicit Flow: This is like a quick access door for visitors (browser-based applications) who need a temporary pass.
Device Code Flow: For those without a keycard, this flow allows entry by confirming identity using another device that does.
Integrated Windows Authentication: Employees (Windows computers in a domain) can seamlessly enter without showing ID every time.
Username/Password: This is like a traditional key where users simply use their credentials to unlock the door.
Interactive methods involve a user directly (like a sign-in or MFA prompt), while non-interactive methods acquire tokens silently in the background, without any user interaction, termed as 'silent' token acquisition. Both methods are essential, and your applications need to handle them according to the user's context and the application's requirements.

Initializing Client Applications

Initializing Client Applications

```
//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();
```


Top example:
Instantiates a public client application, signing-in users in the Microsoft Azure public cloud, with their work and school accounts, or their personal Microsoft accounts.

Bottom example:
Instantiates a confidential application (a Web app located at https://myapp.azurewebsites.net) handling tokens from users in the Microsoft Azure public cloud, with their work and school accounts, or their personal Microsoft accounts. The

application is identified with the identity provider by sharing a client secret.
Includes builder modifiers WithClientSecret and WithRedirectUri.

Single-page, public client, and confidential client applications
Security tokens can be acquired by multiple types of applications. These applications tend to be separated into the following three categories. Each is used with different libraries and objects.
Single-page applications: Also known as SPAs, these are web apps in which tokens are acquired by a JavaScript or TypeScript app running in the browser. The application often uses a framework like Angular, React, or Vue. MSAL.js is the only Microsoft Authentication Library that supports single-page applications.
Public client applications: Are apps that run on devices or desktop computers or in a web browser. They're not trusted to safely keep application secrets, so they only access web APIs on behalf of the user. (They support only public client flows.) Public clients can't hold configuration-time secrets, so they don't have client secrets.
Confidential client applications: Are apps that run on servers (web apps, web API apps, or even service/daemon apps). They're considered difficult to access, and for that reason capable of keeping an application secret. Confidential clients can hold configuration-time secrets. Each instance of the client has a distinct configuration (including client ID and client secret).

Initializing Client Applications

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 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

Modifiers common to both public and confidential client apps

Implementing Shared Access Signatures

Introduction

01 Understand Shared Access Signatures

02 Learn how to control access using Shared Access Signatures

03 Understand when to use SAS

04 Explore stored access policies

Shared Access Signatures (SAS)

Shared Access Signatures

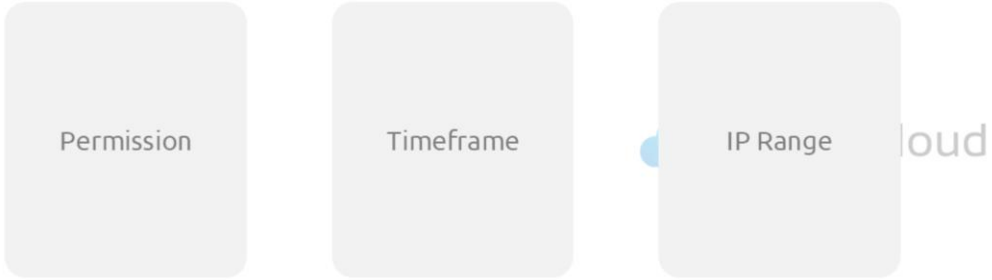
Limited Access to Resources

KodeKloud

Storage Account keys

Shared access signature In this slide, we are diving into shared access signature, which is an essential security feature in Azure for granting limited access to resources without sharing your storage account keys,

Shared Access Signatures Tokens

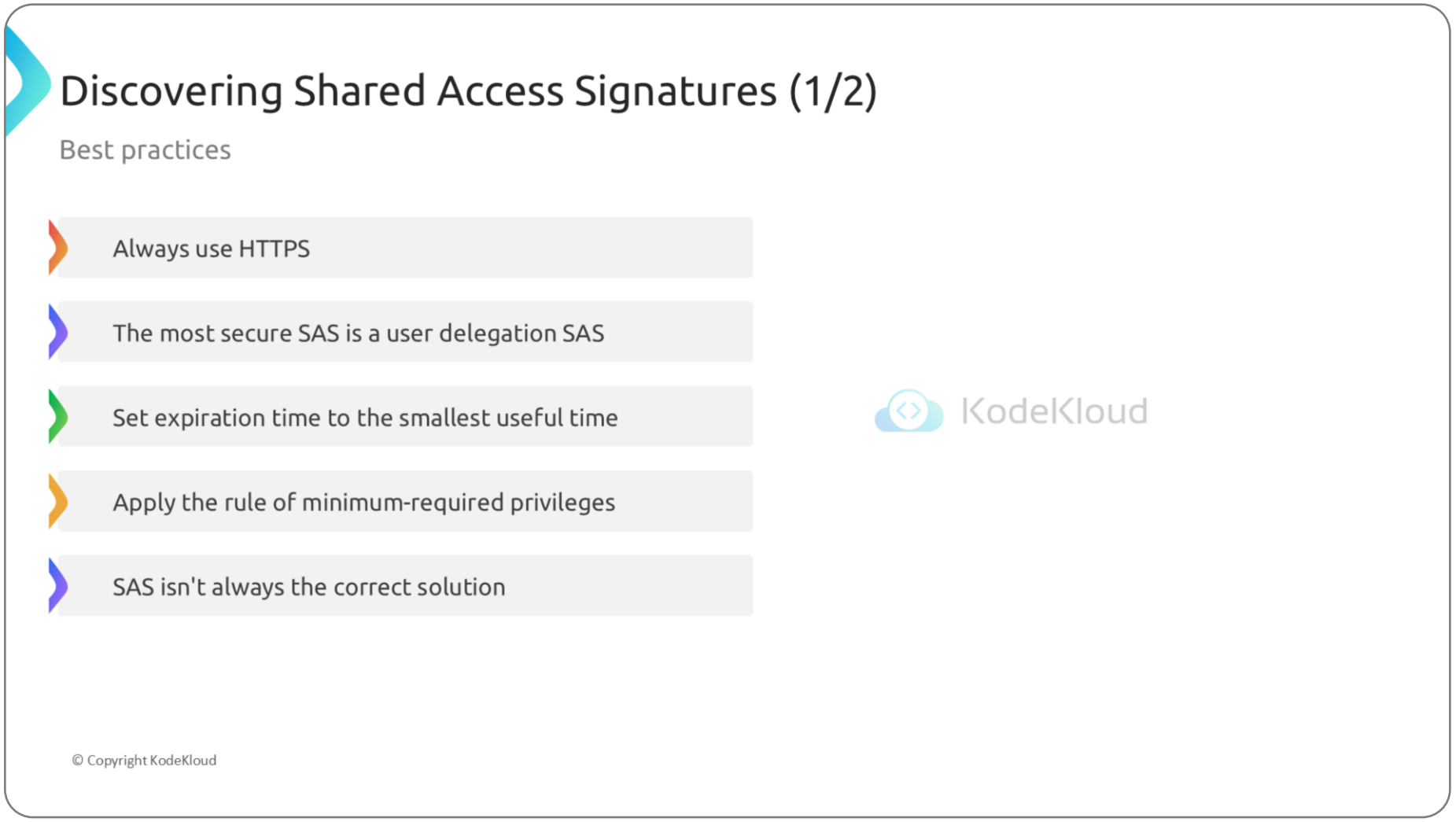


SaaS tokens allows you to specify permissions, timeframe, and the IP range from which access is allowed, making them a versatile tool for secure data sharing

Types of SAS

User delegation SAS: A user delegation SAS is secured with Azure Active Directory credentials and also by the permissions specified for the SAS. A user delegation SAS applies to Blob storage only.
Service SAS: A service SAS is secured with the storage account key. A service SAS delegates access to a resource in the following Azure Storage services: Blob storage, Queue storage, Table storage, or Azure Files.
Account SAS: An account SAS is secured with the storage account key. An account SAS delegates access to resources in one

or more of the storage services. All of the operations available via a service or user delegation SAS are also available via an account SAS.



Best practices
To reduce the potential risks of using a SAS, Microsoft provides some guidance:
To securely distribute a SAS and prevent man-in-the-middle attacks, always use HTTPS.
The most secure SAS is a user delegation SAS. Use it wherever possible because it removes the need to store your storage account key in code. You must use Azure Active Directory to manage credentials. This option might not be possible for your solution.
Try to set your expiration time to the smallest useful value. If a SAS key becomes compromised, it can be exploited for only

a short time.
Apply the rule of minimum-required privileges. Only grant the access that's required. For example, in your app, read-only access is sufficient.
There are some situations where a SAS isn't the correct solution. When there's an unacceptable risk of using a SAS, create a middle-tier service to manage users and their access to storage.
The most flexible and secure way to use a service or account SAS is to associate the SAS tokens with a stored access policy.

Discovering Shared Access Signatures (2/2)

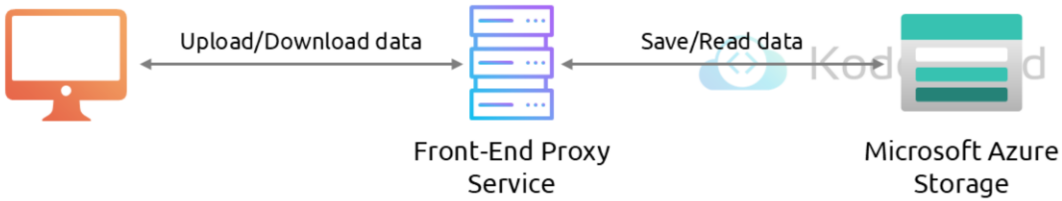
Use SAS to access data

```
URL 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 access rights; 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; for example, "b" is for blob
sig=<...>	The cryptographic signature

When to Use Shared Access Signatures

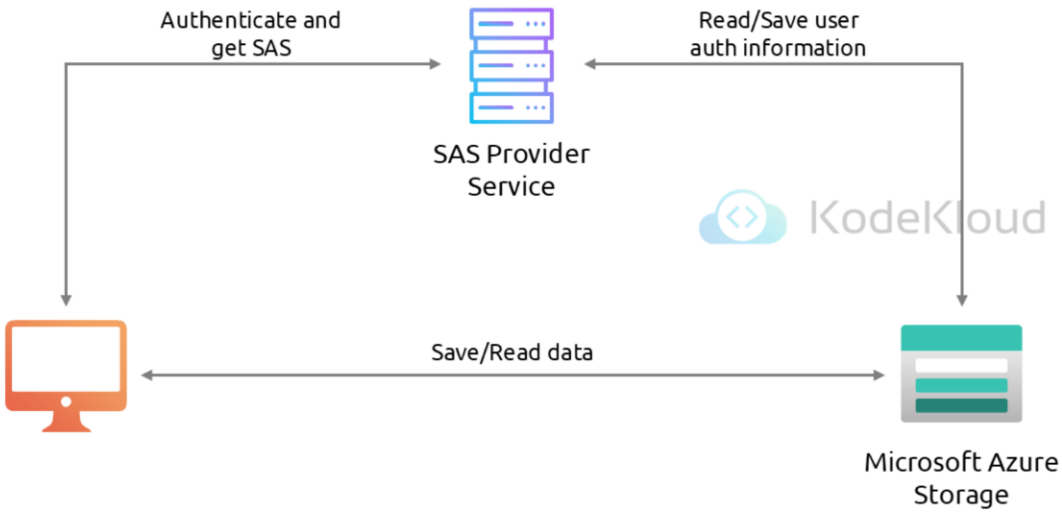
When to Use Shared Access Signatures (1/2)



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.

When to Use Shared Access Signatures (2/2)



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.

Additionally, a SAS is required to authorize access to the source object in a copy operation in certain scenarios:
When you copy a blob to another blob that resides in a different storage account, you must use a SAS to authorize access to the source blob. You can optionally use a SAS to authorize access to the destination blob as well.

When you copy a file to another file that resides in a different storage account, you must use a SAS to authorize access to the source file. You can optionally use a SAS to authorize access to the destination file as well.
When you copy a blob to a file, or a file to a blob, you must use a SAS to authorize access to the source object, even if the source and destination objects reside within the same storage account.

Stored Access Policies

Stored Access Policies

```
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>
```


The following storage resources support stored access policies:
Blob containers
File shares
Queues
Tables

Modifying or revoking a stored access policy

To modify the parameters of the stored access policy you can call the access control list operation for the resource type to replace the existing policy. For example, if your existing policy grants read and write permissions to a resource, you can modify it to grant only read permissions for all future requests.

To revoke a stored access policy you can delete it, rename it by changing the signed identifier, or change the expiry time to a value in the past. Changing the signed identifier breaks the associations between any existing signatures and the stored access policy. Changing the expiry time to a value in the past causes any associated signatures to expire. Deleting or modifying the stored access policy immediately affects all of the SAS associated with it.

To remove a single access policy, call the resource's Set ACL operation, passing in the set of signed identifiers that you wish to maintain on the container.
To remove all access policies from the resource, call the Set ACL operation with an empty request body.

Exploring Microsoft Graph

Introduction

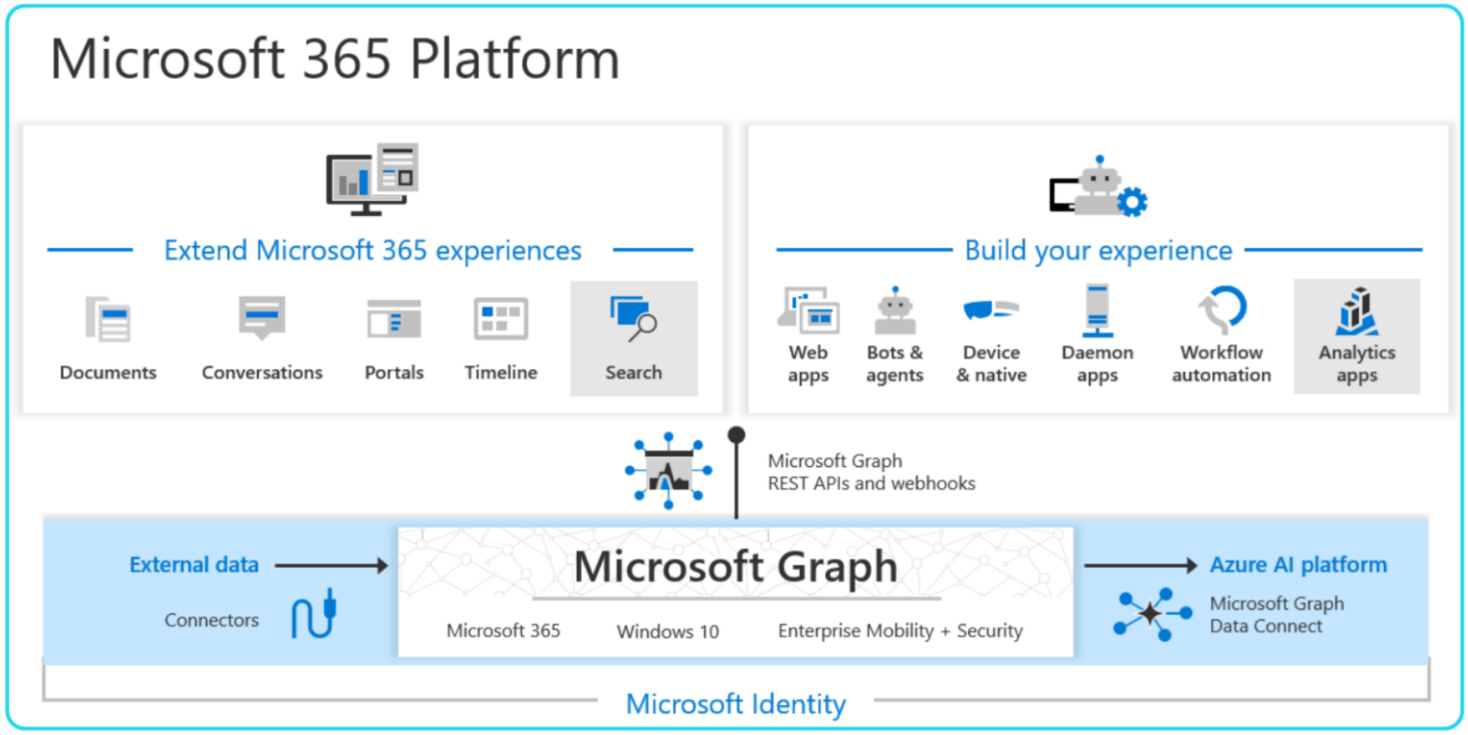
01 Explore Microsoft Graph and its benefits

02 Perform operations on Microsoft Graph by using REST APIs and SDKs

03 Learn best practices to leverage Microsoft Graph

Discovering Microsoft Graph

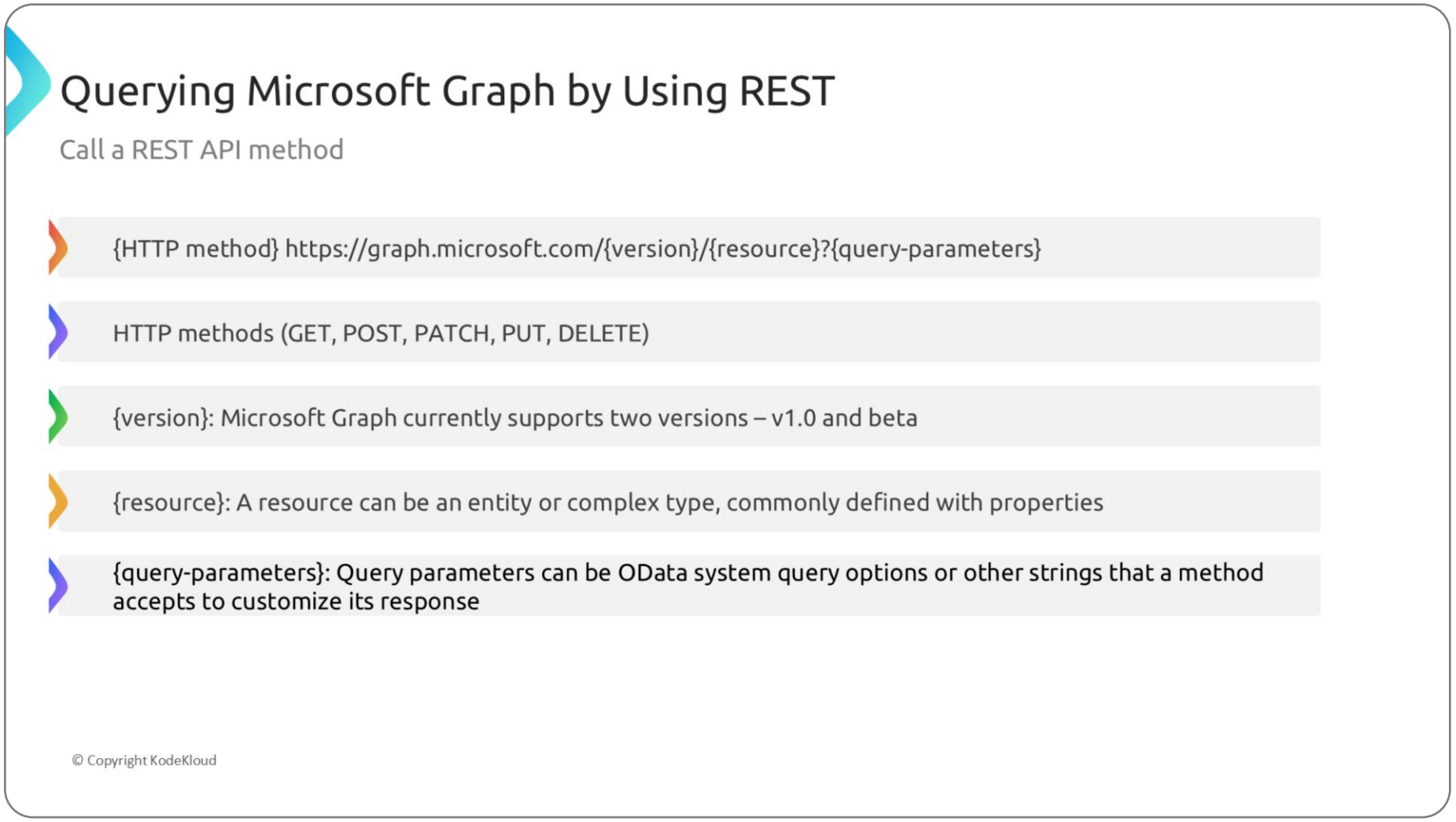
Discovering Microsoft Graph



Microsoft Graph is a gateway to data and intelligence in Microsoft 365, providing access to a vast array of information across Microsoft services. It enables applications to leverage a single endpoint, https://graph.microsoft.com, to interact with various Microsoft cloud services.
The Microsoft Graph API encompasses services that manage user and device identity, access, compliance, and security, forming a comprehensive suite that aids in protecting organizations from data leakage or loss.
Microsoft Graph connectors facilitate the integration of external data into Microsoft Graph services and applications, enriching experiences within the Microsoft 365 suite, such as enhancing Microsoft Search with data from external platforms

like Jira or Salesforce.
Furthermore, Microsoft Graph Data Connect empowers developers by providing tools for secure and scalable delivery of Microsoft Graph data to Azure data stores, forming a foundation for developing intelligent applications within the Azure ecosystem.

Querying Microsoft Graph



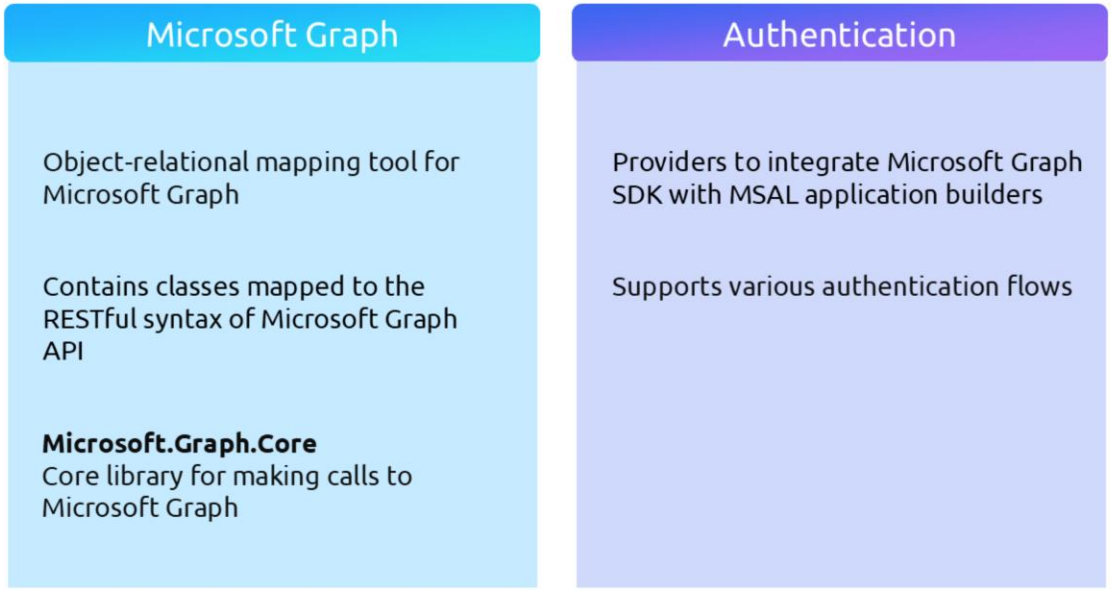
Microsoft Graph is a RESTful web API that enables you to access Microsoft Cloud service resources. After you register your app and get authentication tokens for a user or service, you can make requests to the Microsoft Graph API.

The Microsoft Graph API defines most of its resources, methods, and enumerations in the OData namespace, microsoft.graph, in the Microsoft Graph metadata. A small number of API sets are defined in their sub-namespaces, such as the [call records API which defines resources like callRecord in microsoft.graph.callRecords.

Unless explicitly specified in the corresponding topic, assume types, methods, and enumerations are part of the microsoft.graph namespace.

Additional resources
Below are links to some tools you can use to build and test requests using Microsoft Graph APIs.
Graph Explorer
Postman

Querying Microsoft Graph by Using SDKs (1/4)



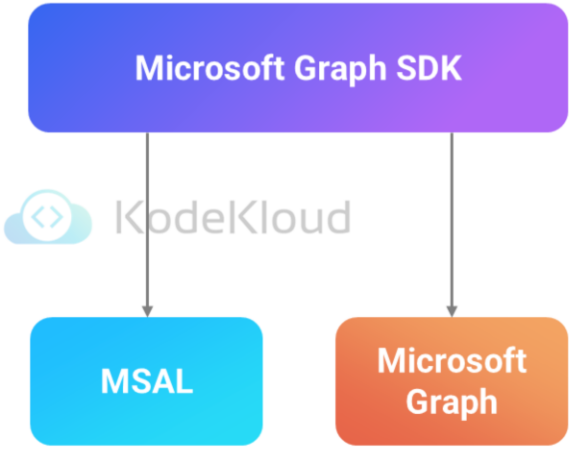
Authentication

For details about which provider and options are appropriate by scenario, see Choose an Authentication Provider (https://docs.microsoft.com/graph/sdks/choose-authentication-providers)

Querying Microsoft Graph by Using SDKs (2/4)

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



Microsoft Graph SDKs are designed to simplify building high-quality, efficient, and resilient applications that access Microsoft Graph. SDKs include two components: a service library and a core library.

The service library contains models and request builders that are generated from Microsoft Graph metadata to provide a rich, strongly typed, and discoverable experience when working with the many datasets that are available in Microsoft Graph.

The core library provides a set of features that enhance working with all Microsoft Graph services. The features include embedded support for retry handling, secure redirects, transparent authentication, and payload compression, which improve the quality of your application's interactions with Microsoft Graph. These features don’t add complexity and give you enhanced control of your application. The core library also provides support for common tasks such as paging through collections and creating batch requests.

Querying Microsoft Graph by Using SDKs (3/4)

```
// 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);
```


You can use a single client instance for the lifetime of the application.

The code example show how to create an instance of a Microsoft Graph client.

The authentication provider will handle acquiring access tokens for the application. The different application providers support different client scenarios.

Querying Microsoft Graph by Using SDKs (4/4)

```
// 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();
```


The $filter query parameter can be used to reduce the result set to only those rows that match the provided condition.

The $orderBy query parameter will request that the server provide the list of entities sorted by the specified properties.

There are more examples in the student manual and in the Trainer Handbook.

Best Practices

Applying 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

Handling Responses Effectively

Pagination

Evolvable enumerations

Storing data locally:

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

KodeKloud

Visit www.kodekloud.com to learn more.