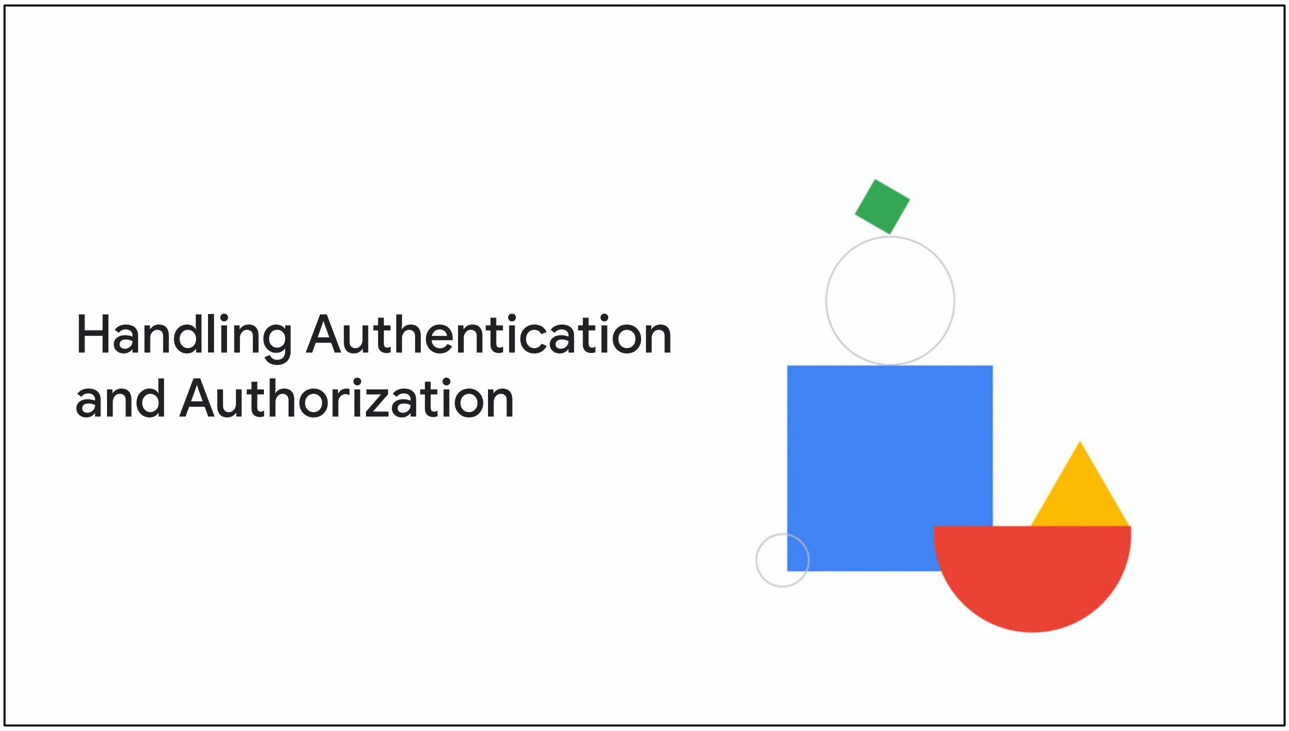
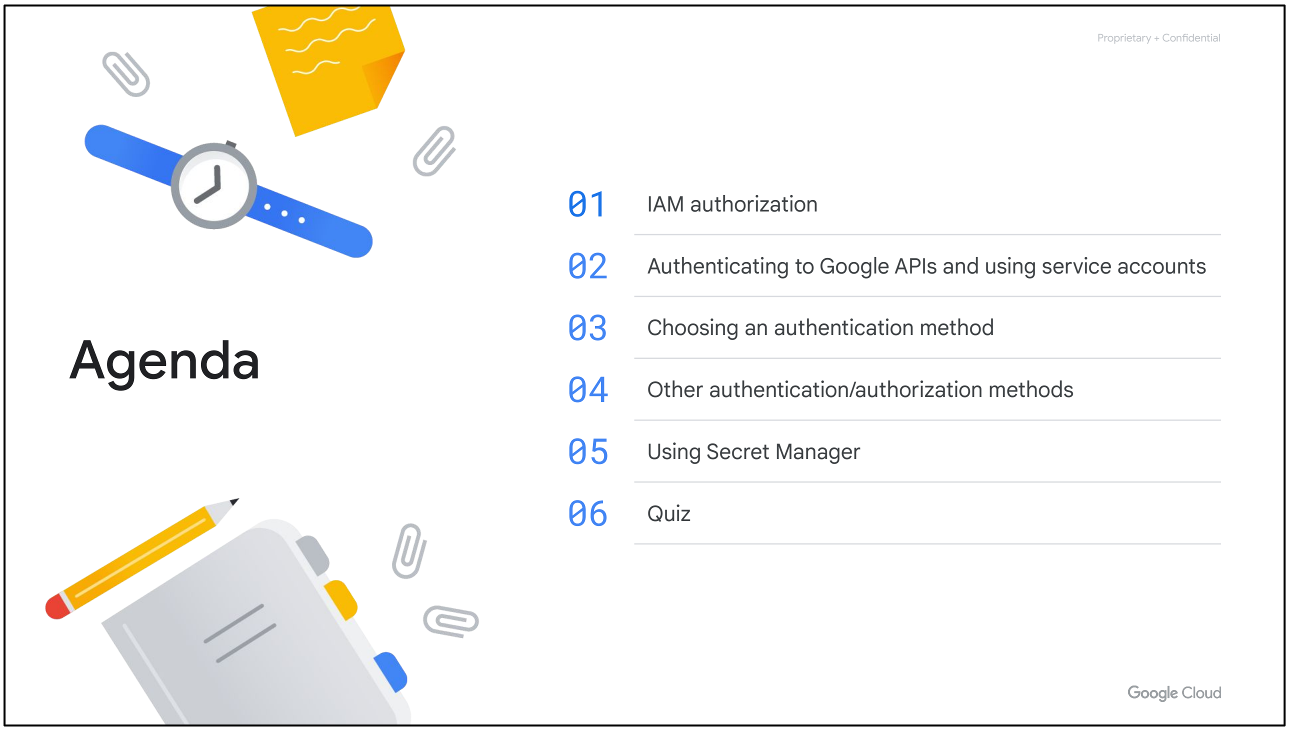


Handling Authentication and Authorization

Welcome to Developing Applications with Google Cloud: Foundations, module 4: Handling Authentication and Authorization.

It can be daunting to implement user authentication and authorization from the beginning in your application. Writing your own code to secure user data leaves you vulnerable to attacks. With Identity and Access Management (IAM) and Identity Platform authentication, you have a simple and secure solution for authenticating and authorizing your application users.



Agenda

Google Cloud

Here is a list of topics included in this module.

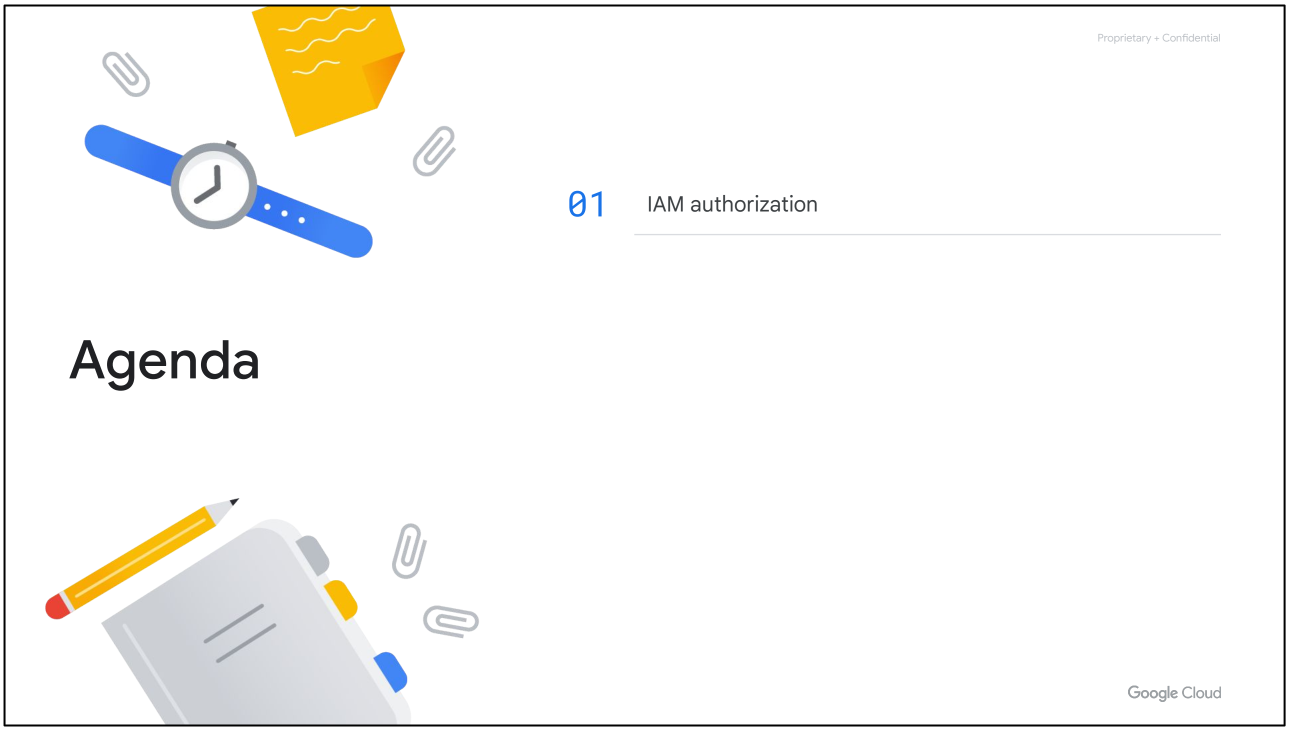
In this module, you learn about IAM principals and how to specify the resources they can access and the operations they can perform on these resources.

You learn how to use service accounts and other methods to authenticate and authorize applications to invoke Google Cloud APIs, and when to use each method. We discuss how you can use Identity-Aware Proxy to control access to your application.

The Firebase SDK makes it really easy to implement federated identity management. You learn how to use the Firebase SDK to validate users against credentials stored in Identity Platform, Google Cloud's enterprise-grade access management and identity platform.

Finally, you learn about Secret Manager, and how it can be used to securely store credentials that are required by your application.

We'll end the module with a short quiz on the topics that were discussed.



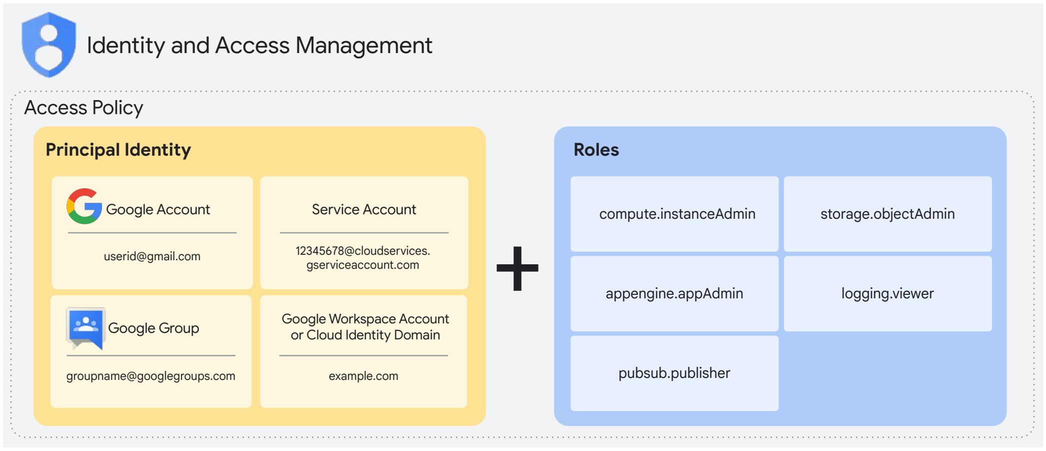
Agenda

01 IAM authorization

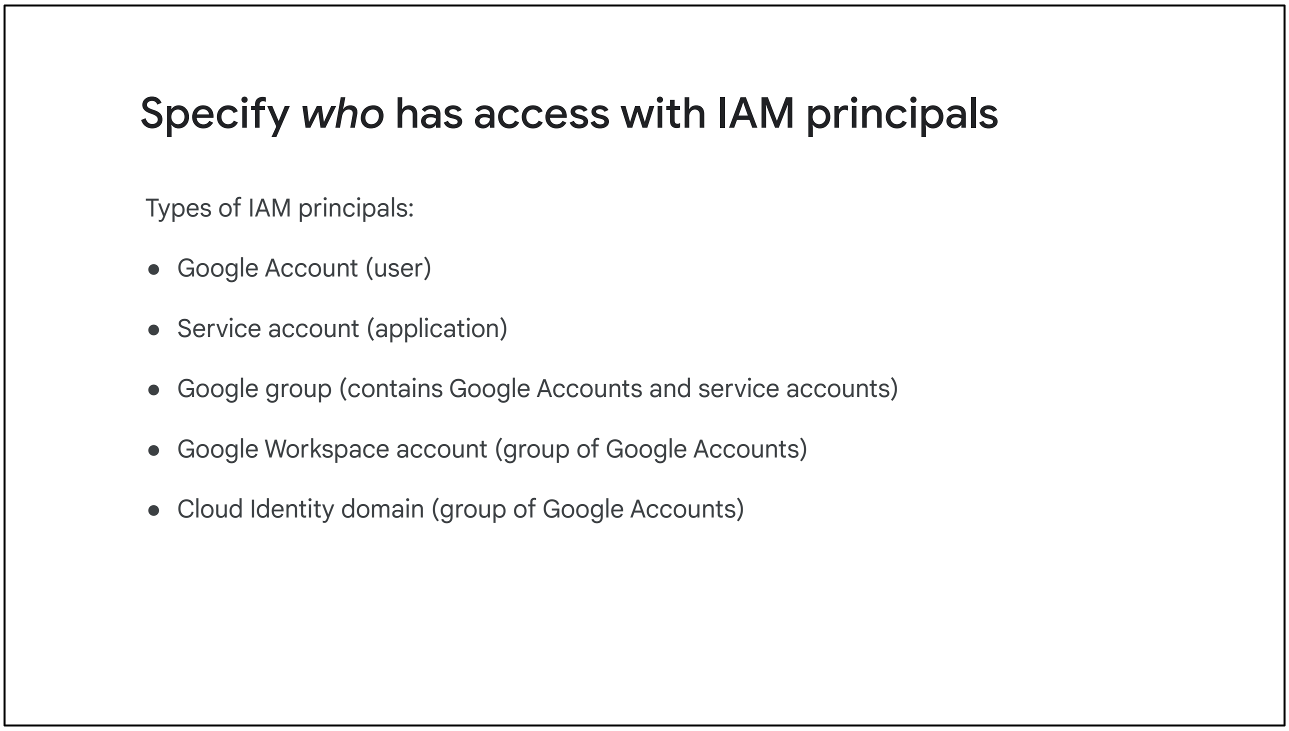
Google Cloud

First, we review IAM authorization.

Authorization:
Identity and Access Management (IAM)



IAM lets you manage access control by defining who (the principal) has what access (role) for which resource. You can grant more granular access to Google Cloud resources by using the security principle of least privilege, which means that you should only grant access to resources that are necessary.


Specify *who* has access with IAM principals

Types of IAM principals:

- Google Account (user)
- Service account (application)
- Google group (contains Google Accounts and service accounts)
- Google Workspace account (group of Google Accounts)
- Cloud Identity domain (group of Google Accounts)

You can specify who has access to your resources with IAM principals.

A Google Account represents a developer, an administrator, or any other person who interacts with Google Cloud.

A service account is an account for an application or compute workload instead of an individual end user. When you run code that's hosted on Google Cloud, the code runs as the account you specify. You can create as many service accounts as needed to represent the different logical components of your application.

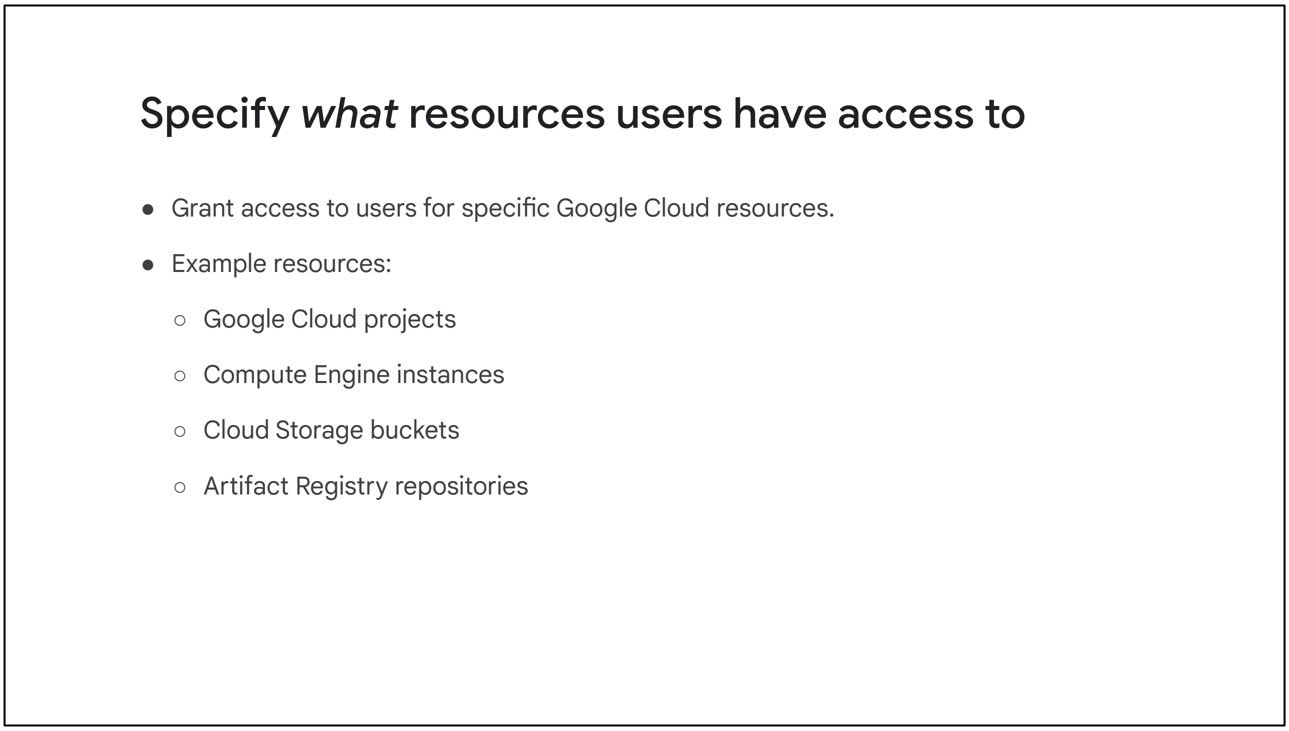
A Google group is a named collection of Google Accounts and service accounts. Every Google group has a unique email address that's associated with the group. Google Groups are a convenient way to apply access controls to a collection of users. You can grant and change access controls for a whole group at once instead of one at a time for individual users or service accounts. Google Groups don't have login credentials, and you cannot use Google Groups to establish identity to make a request to access a resource.

A Google Workspace account represents a virtual group of all Google Accounts that it contains. Google Workspace accounts are associated with your organization's internet domain name, such as example.com. Like Google Groups, Google Workspace accounts cannot be used to establish identity, but they enable convenient permission management.

A Cloud Identity domain is like a Google Workspace account, because it represents a virtual group of all Google Accounts in an organization. However, Cloud Identity domain users don't have access to Google Workspace applications and features. Like Google Workspace accounts, a Cloud Identity domain cannot be used to establish identity.

Google groups, Google Workspace accounts, and Cloud Identity domains are convenient ways to apply access policies to collection of users.

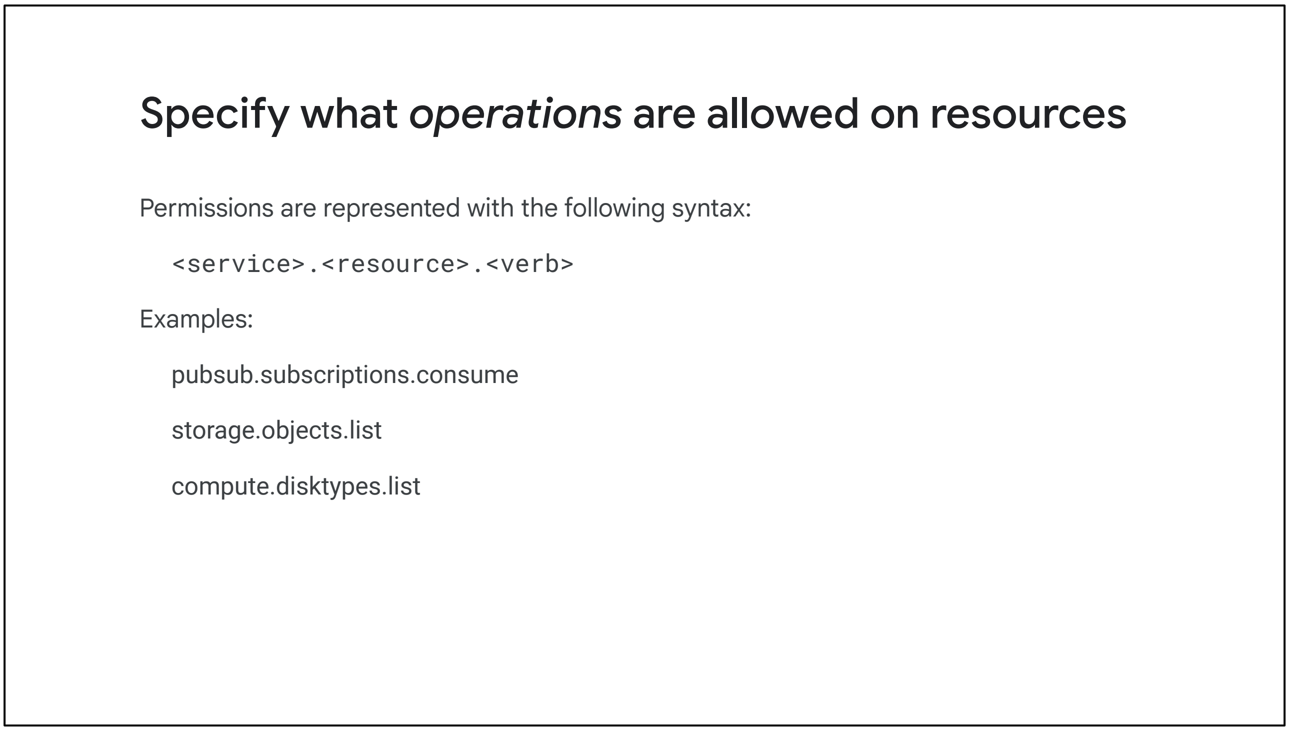
[Concepts related to identity:
https://cloud.google.com/iam/docs/overview#concepts_related_identity]



Specify *what* resources users have access to

- Grant access to users for specific Google Cloud resources.
- Example resources:
  - Google Cloud projects
  - Compute Engine instances
  - Cloud Storage buckets
  - Artifact Registry repositories

You can grant access to principals for a Google Cloud resource. Some examples of resources are projects, Compute Engine instances, Cloud Storage buckets, or Artifact Registry repositories.



Specify what *operations* are allowed on resources

Permissions are represented with the following syntax:

```
<service>.<resource>.<verb>
```

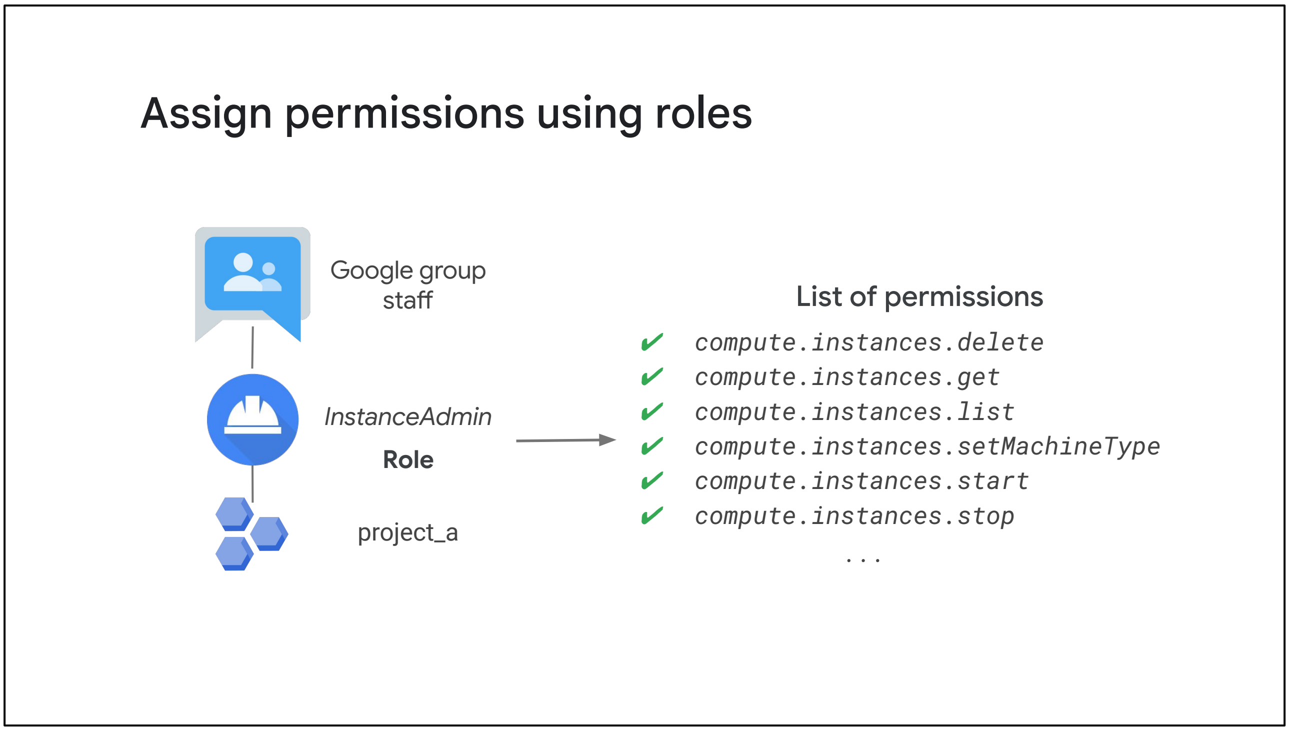
Examples:

pubsub.subscriptions.consume

storage.objects.list

compute.disktypes.list

Permissions determine what operations are allowed on a resource. In the IAM world, permissions are represented in the form of <service>.<resource>.<verb>, for example, pubsub.subscriptions.consume.



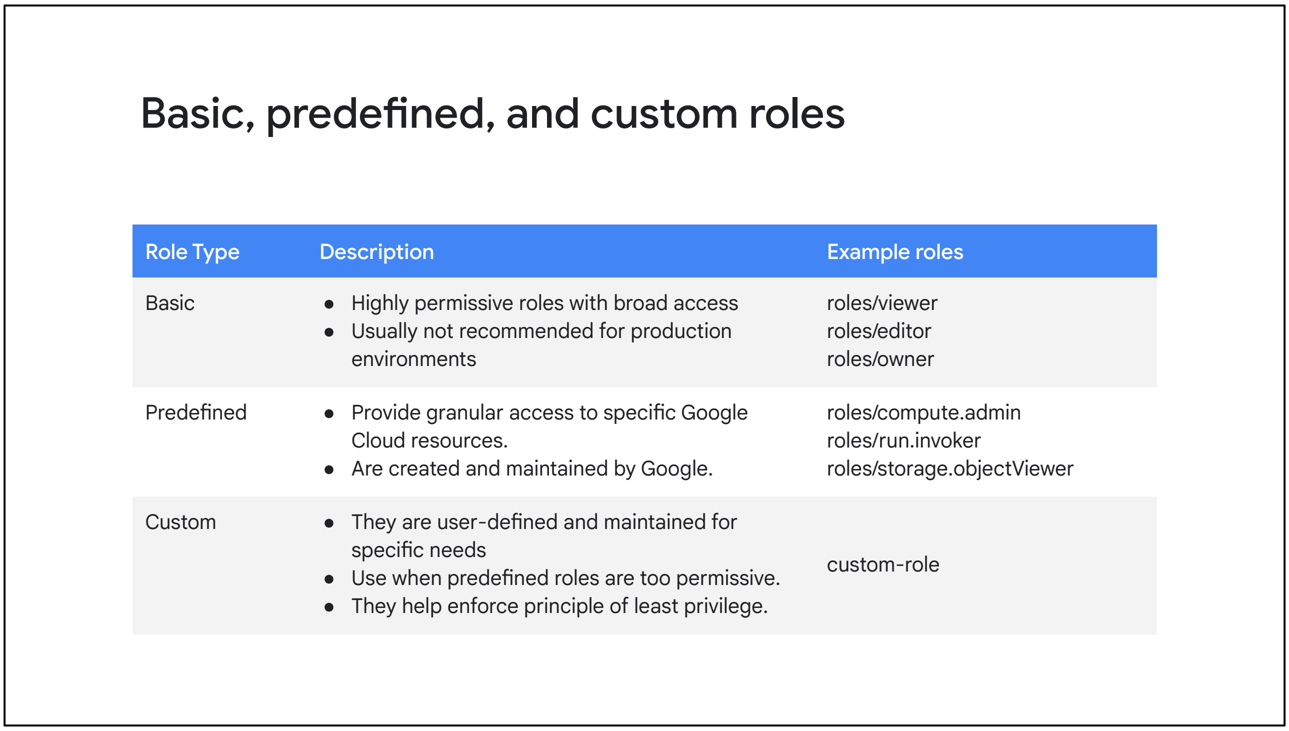
Assign permissions using roles

```
✔ compute.instances.delete
✔ compute.instances.get
✔ compute.instances.list
✔ compute.instances.setMachineType
✔ compute.instances.start
✔ compute.instances.stop
           ...
```

You cannot assign a permission to a user directly. Instead, you grant the user a role. A role is a collection of permissions. When you grant a role to a user, you grant them all the permissions that the role contains.

In this example, all users in the Google group named "staff" are granted the InstanceAdmin role on project_a. Each user would have all permissions contained in the role.

https://cloud.google.com/iam/docs/understanding-roles


Basic, predefined, and custom roles

Role Type	Description	Example roles
Basic	• Highly permissive roles with broad access • Usually not recommended for production environments	roles/viewer roles/editor roles/owner
Predefined	• Provide granular access to specific Google Cloud resources. • Are created and maintained by Google.	roles/compute.admin roles/run.invoker roles/storage.objectViewer
Custom	• They are user-defined and maintained for specific needs • Use when predefined roles are too permissive. • They help enforce principle of least privilege.	custom-role

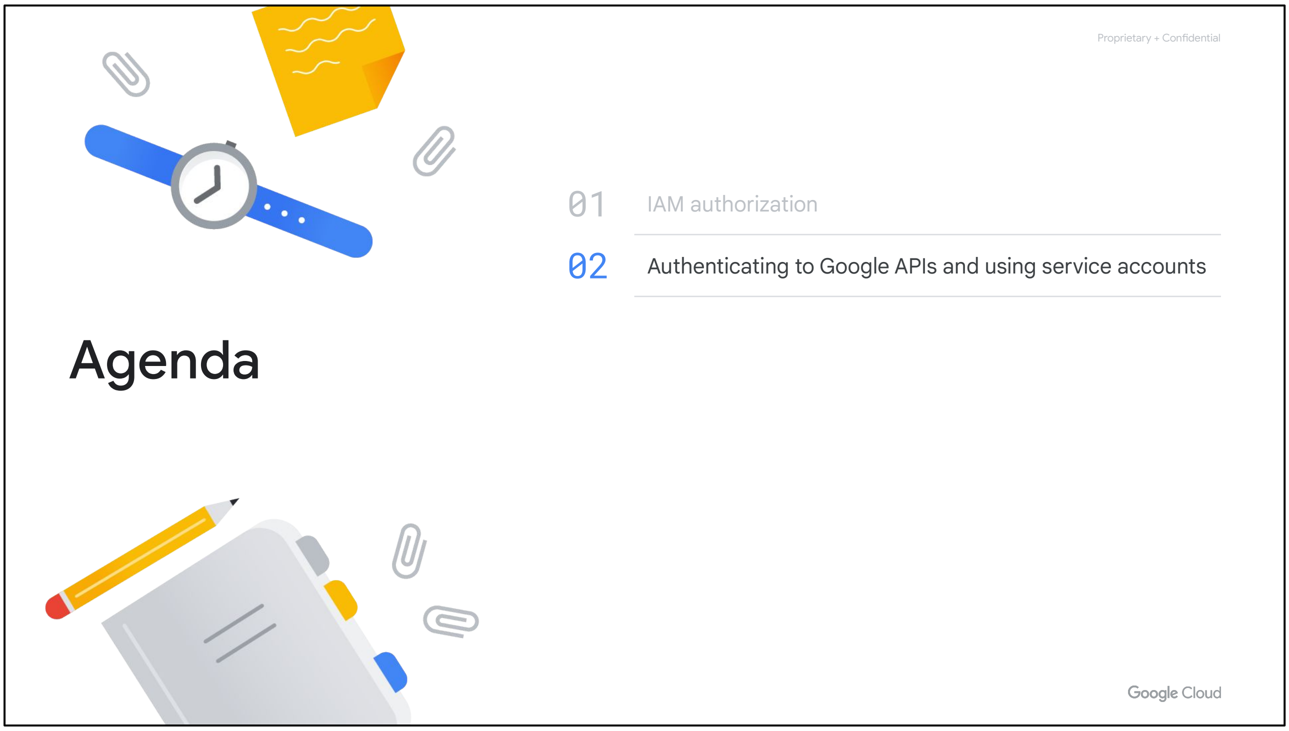
IAM supports three different types of roles.

Basic roles are highly permissive roles with broad access. For example, the viewer role gets read-only access to all resources in a project. Because basic roles are so broad, they are usually not recommended for production environments.

Predefined roles provide granular access to specific Google Cloud resources. Predefined roles are created and maintained by Google. An example role is run.invoker, which lets the user invoke Cloud Run services.

Custom roles are user-defined and typically maintained for specific needs. You should create and use custom roles when existing predefined roles are too permissive for your use case. Because you have full control over the permissions given, custom roles can help enforce the principle of least privilege. The principle of least privilege says that you should provide each principal exactly the permissions necessary for it to do its job.

You can grant multiple roles to the same user.



Agenda

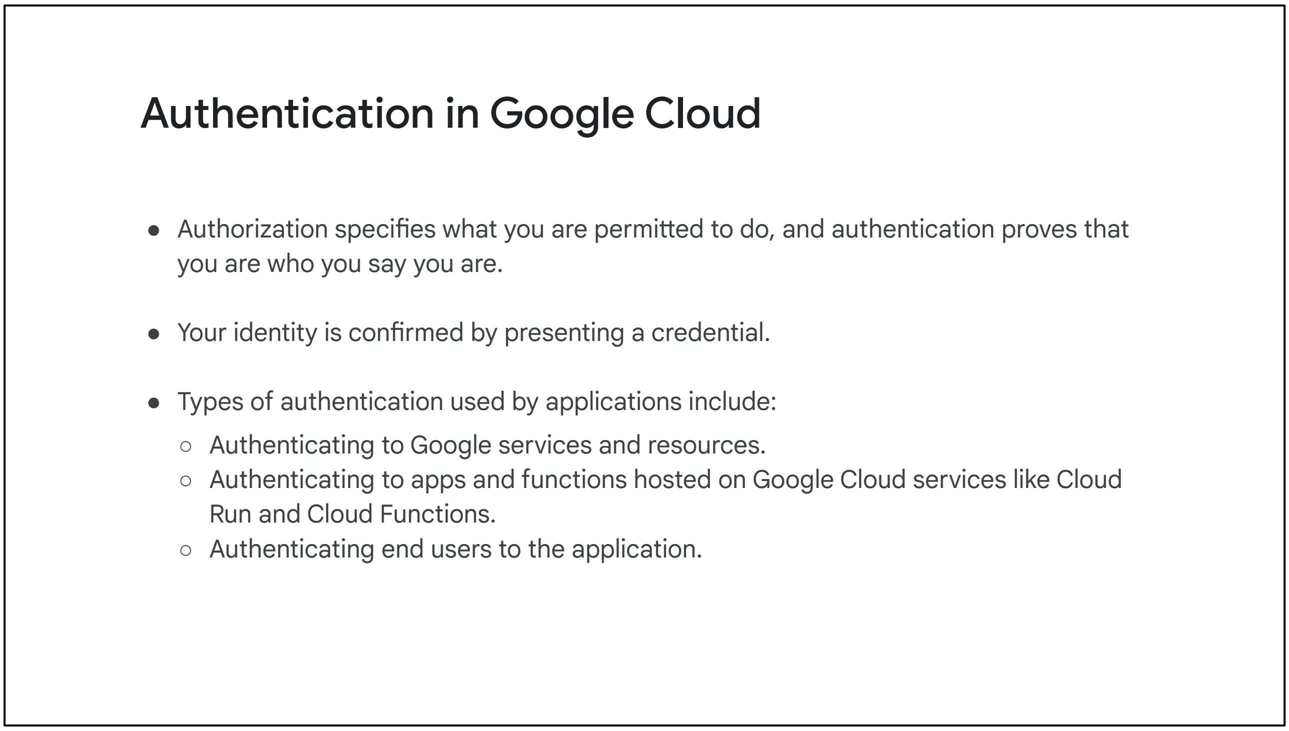
Google Cloud

Next, we discuss authentication in Google Cloud.

Agenda

01	IAM authorization
02	Authenticating to Google APIs and using service accounts
03	Choosing an authentication method
04	Other authentication/authorization methods
05	Using Secret Manager

Next, we discuss authentication in Google Cloud.



Authentication in Google Cloud

- Authorization specifies what you are permitted to do, and authentication proves that you are who you say you are.
- Your identity is confirmed by presenting a credential.
- Types of authentication used by applications include:
  - Authenticating to Google services and resources.
  - Authenticating to apps and functions hosted on Google Cloud services like Cloud Run and Cloud Functions.
  - Authenticating end users to the application.

IAM enables authorization for Google Cloud services, specifying what you're permitted to do. Authentication proves that you are who you say you are.

You prove your identity by presenting some kind of credential.

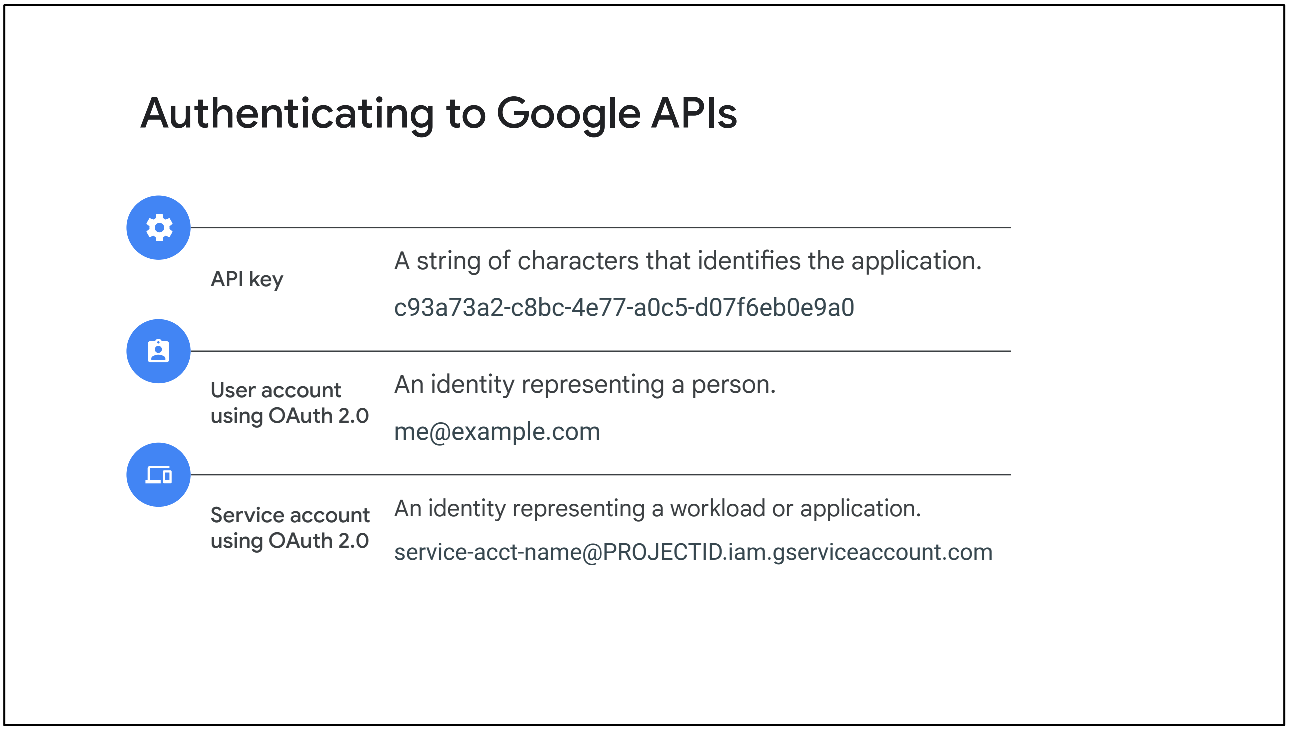
If you are an application developer on Google Cloud, you might need to use or create different types of authentication.

Your application might need to authenticate to Google and allow access to Google services and resources.

You might need to call apps hosted on Google Cloud services like Cloud Functions and Cloud Run. Usually, you want only authenticated callers to access your apps.

You might also need to authenticate end users for your application.

[Authentication at Google Cloud: https://cloud.google.com/docs/authentication]



Authenticating to Google APIs

API key	A string of characters that identifies the application. c93a73a2-c8bc-4e77-a0c5-d07f6eb0e9a0
User account using OAuth 2.0	An identity representing a person. me@example.com
Service account using OAuth 2.0	An identity representing a workload or application. service-acct-name@PROJECTID.iam.gserviceaccount.com

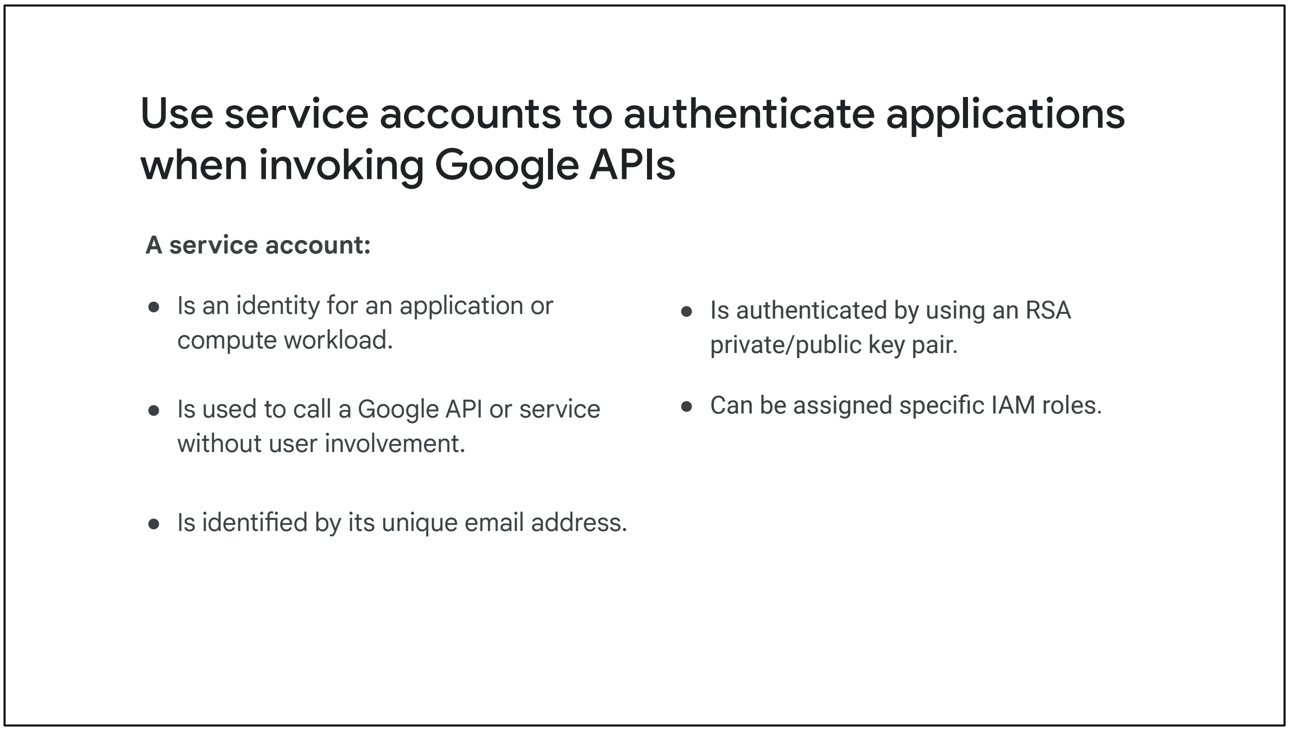
When you call a Google Cloud service, you're typically making an API call. There are three ways to authorize API calls to Google services.

An API key is a string of characters that identifies the application. Using an API key associates the request with a Google Cloud project for billing and quota purposes. A compromised API key will allow full and long-lasting access to the API, so API keys are typically appropriate only for low security, read-only APIs. Most Google APIs do not accept API keys.

A user account represents a person. The account is identified by the email address of the person. The act of logging in uses the email address and a separate credential, usually a password, to create an OAuth token. The token allows limited access to the API, based on the user's permissions, and expires after a period of time. OAuth tokens are typically more secure than API keys.

A service account represents a workload or application, and is identified by its unique email address. An OAuth token for a service account provides access to the API based on the roles attached to the service account.

We focus first on service account authentication.



Use service accounts to authenticate applications when invoking Google APIs

A service account:

- Is an identity for an application or compute workload.
- Is used to call a Google API or service without user involvement.
- Is identified by its unique email address.
- Is authenticated by using an RSA private/public key pair.
- Can be assigned specific IAM roles.

A service account acts as the identity for an application or a compute workload.

The service account is used by your application to call a Google API or service so that users aren't directly involved in the authentication process.

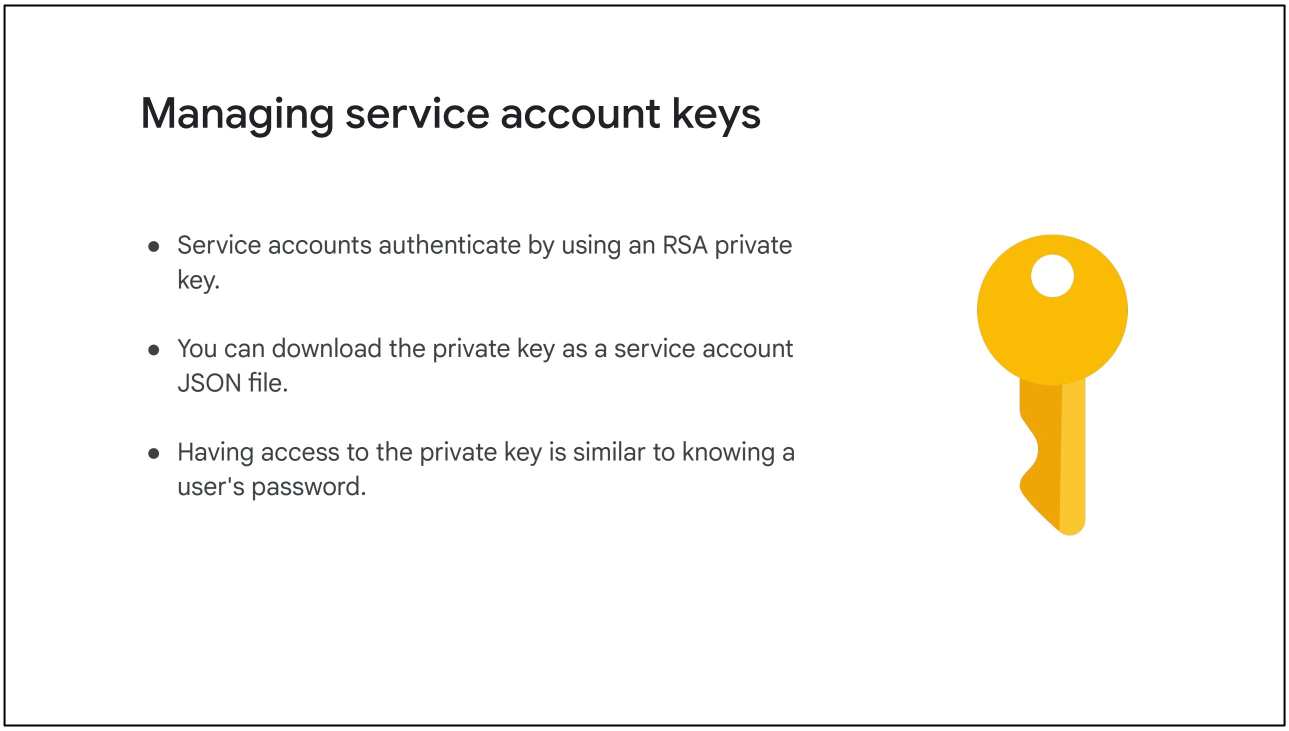
Each service account is identified by its unique email address.

Service accounts enable authorization because you can assign specific IAM roles to a service account.

Service accounts are authenticated by using an RSA private/public key pair. There is no password associated with a service account, so you can't use a service account to log in with a browser.

You can assign specific IAM roles to a service account to provide the level of access required by the application.

[Service Accounts: https://cloud.google.com/iam/docs/service-accounts]



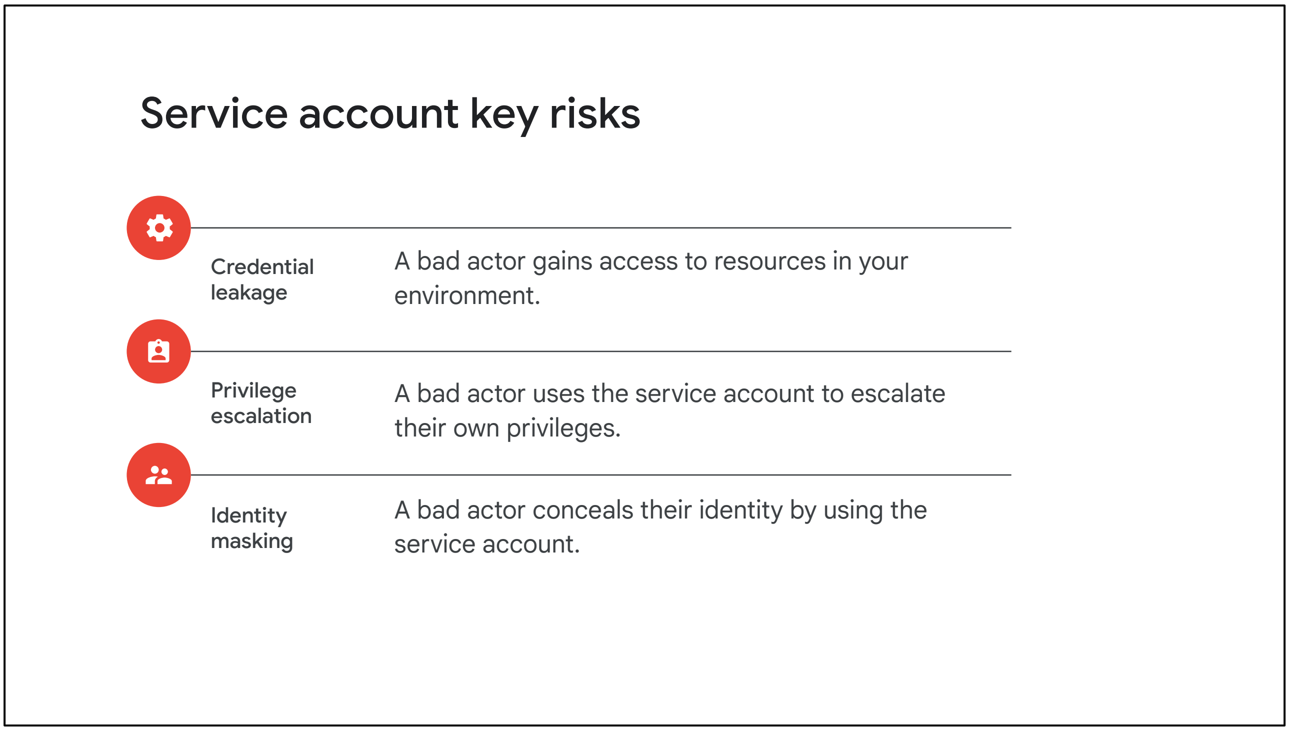
Managing service account keys

- Service accounts authenticate by using an RSA private key.
- You can download the private key as a service account JSON file.
- Having access to the private key is similar to knowing a user's password.

Unlike user accounts, service accounts do not have passwords. Instead, service accounts use RSA key pairs for authentication.

The private key for a service account can be downloaded as a service account JSON file. In the early days of service accounts, downloaded service account keys were typically used whenever you needed to authenticate as an application.

If you know the private key of a service account's key pair, you can use the private key to request an access token. The resulting access token can be used to interact with Google Cloud APIs on the service account's behalf. For this reason, having access to the private key is similar to knowing a user's password. Service account keys can become a security risk if not managed carefully.



Service account key risks

Credential leakage	A bad actor gains access to resources in your environment.
Privilege escalation	A bad actor uses the service account to escalate their own privileges.
Identity masking	A bad actor conceals their identity by using the service account.

There are risks associated with using service account keys.

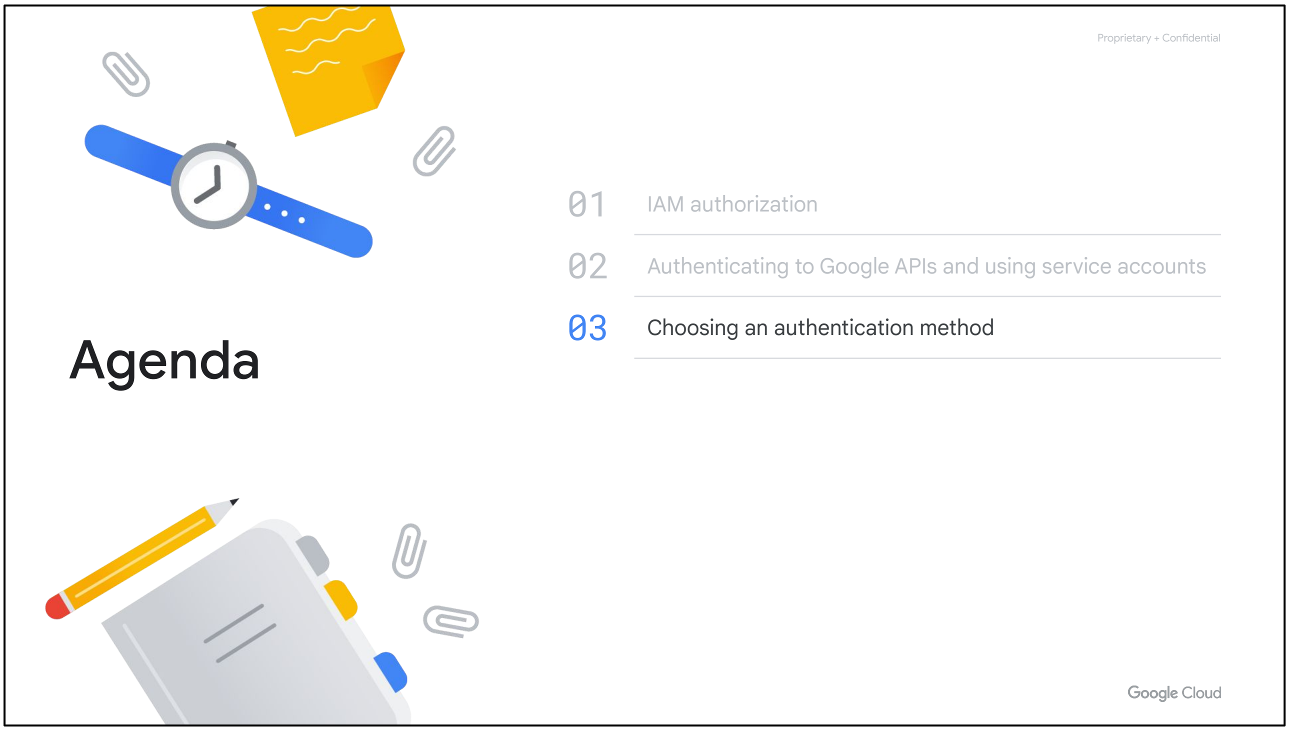
The first risk is credential leakage. Consider what would happen if a developer committed a private key to a public code repository. A bad actor could use this key to gain access to resources in your environment.

Another risk is privilege escalation. If a bad actor gets access to a service account key, they could use the key to escalate their own privileges. For example, the bad actor could use a service account with permission on a database to give themselves access to the database. Even after you manage to detect the danger and change the service account key, escalated privileges would remain.

A third risk is identity masking. By authenticating as the service account, a bad actor might conceal their identity and actions.

The best way to mitigate these risks is to avoid using downloaded service account keys and use other methods to authenticate service accounts whenever possible.

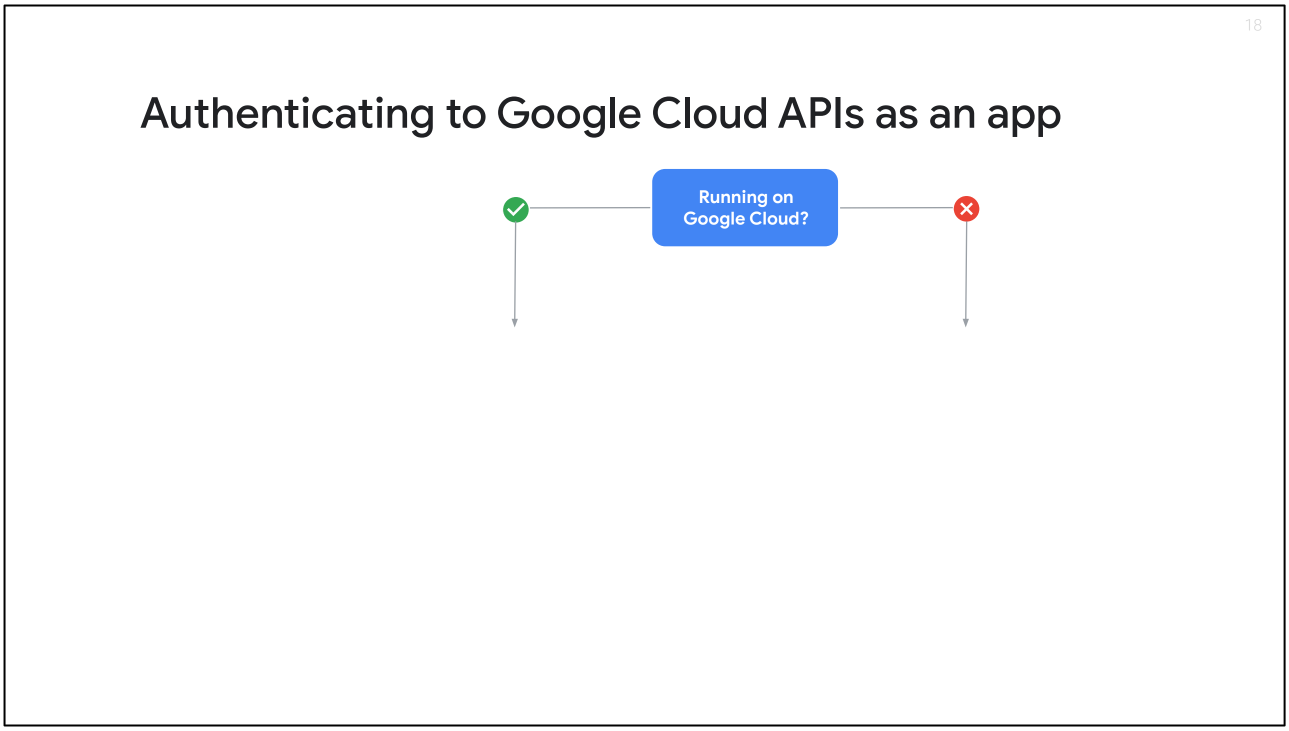
[Best practices for managing service account keys: https://cloud.google.com/iam/docs/best-practices-for-managing-service-account-keys]



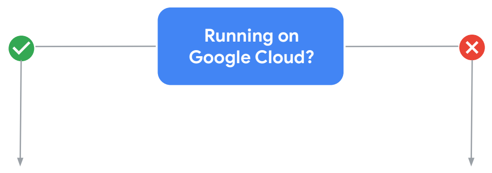
Agenda

Google Cloud

There are multiple ways to authenticate to Google Cloud APIs from an application.

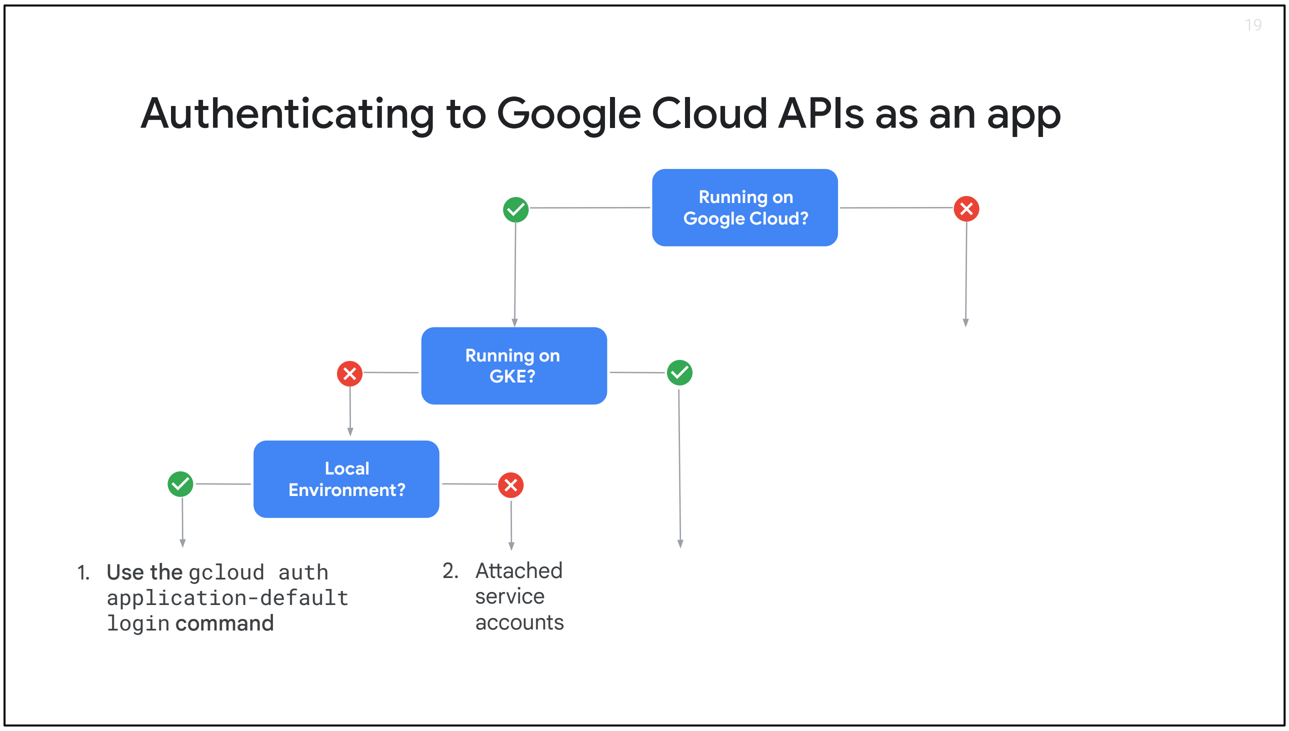


Authenticating to Google Cloud APIs as an app



This diagram will help you choose which method you should use based on your situation.

The first question that you need to answer is whether your application is running on Google Cloud.



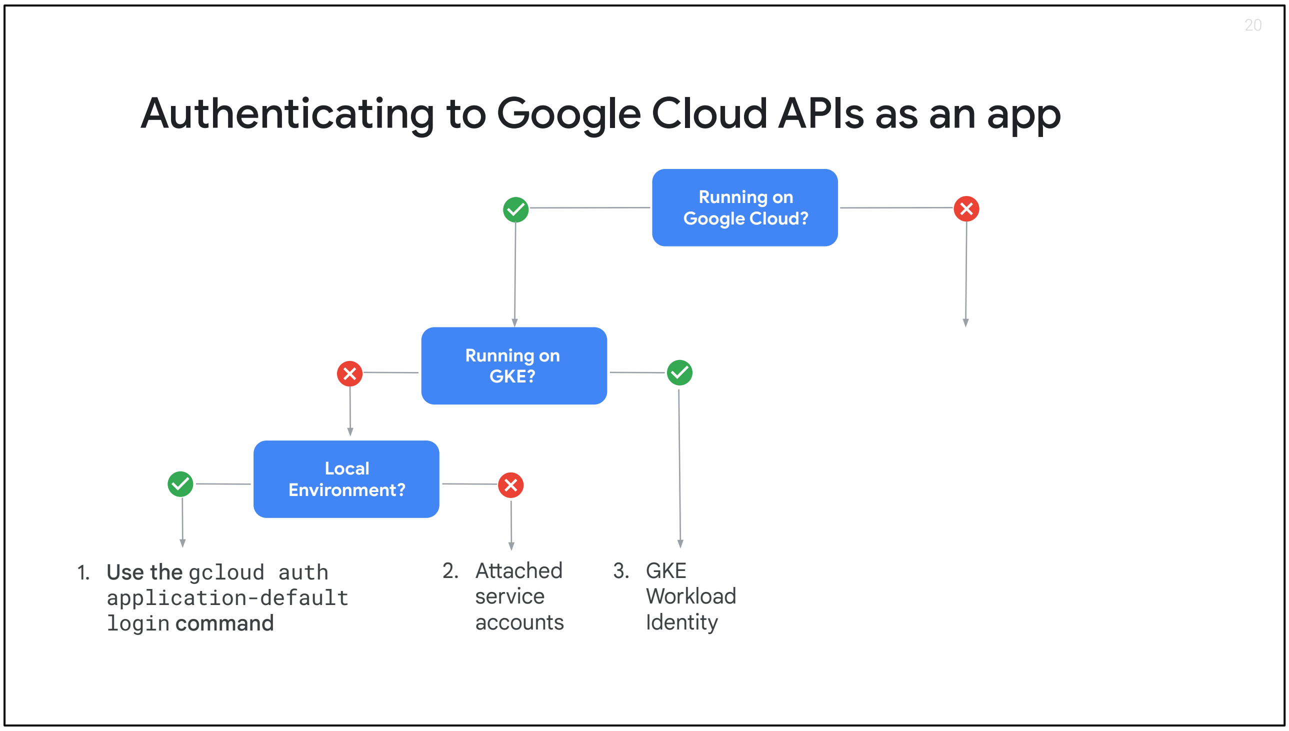
Authenticating to Google Cloud APIs as an app

1. Use the `gcloud auth application-default login` command
2. Attached service accounts

If your app is running on Google Cloud, and your app is not running on Google Kubernetes Engine (GKE), there are two recommended methods for authentication.

For development work in a local environment, you should use the "gcloud auth application-default login" command to let the application use your user credentials.

For applications that are not running in a local development environment, attach a service account directly to the compute or serverless instance on Google Cloud.

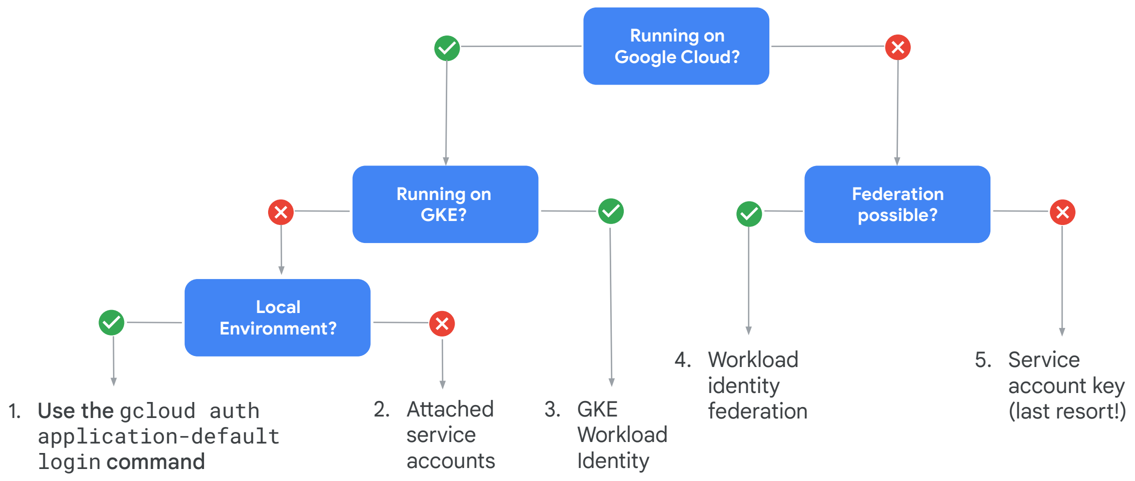


Authenticating to Google Cloud APIs as an app

1. Use the `gcloud auth application-default login` command
2. Attached service accounts
3. GKE Workload Identity

If your application is running on GKE, Workload Identity is the best solution. Workload Identity allows workloads in your GKE clusters to impersonate IAM service accounts to access Google Cloud APIs.

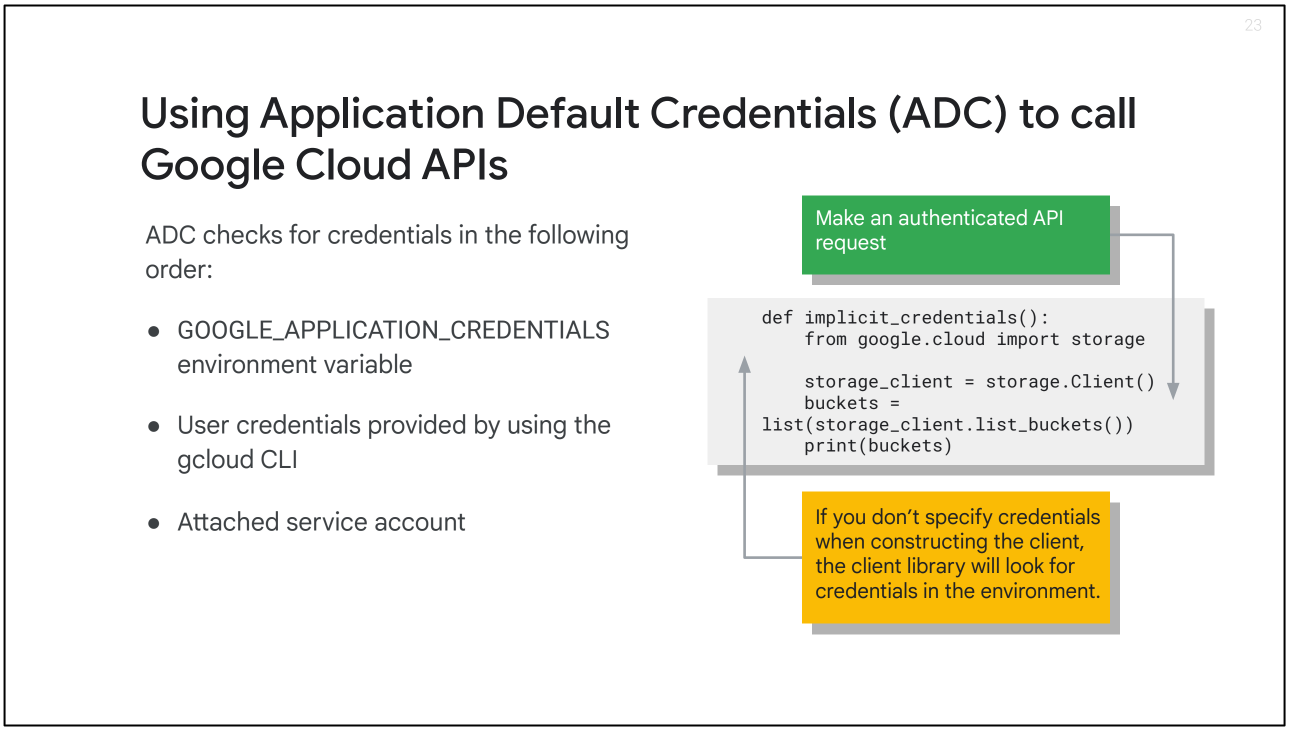
Authenticating to Google Cloud APIs as an app


Running on Google Cloud?
Running on GKE?
Federation possible?
Local Environment?
1. Use the `gcloud auth application-default login` command
2. Attached service accounts
3. GKE Workload Identity
4. Workload identity federation
5. Service account key (last resort!)

If you're not running your application on Google Cloud, you need to decide whether federation is possible.

You're able to set up Workload Identity Federation for workloads running in other clouds or running on-premises, as long as the cloud provider or on-premises deployment can generate an ID token. Workload Identity Federation lets you exchange an external provider token for a Google Cloud access token. With that access token, you can impersonate a regular service account and its permissions without requiring a service account key.

If federation is not possible, your last resort is to use service account keys. Take great care to secure any service account keys that are used.



Using Application Default Credentials (ADC) to call Google Cloud APIs

ADC checks for credentials in the following order:

- GOOGLE_APPLICATION_CREDENTIALS environment variable
- User credentials provided by using the gcloud CLI
- Attached service account

Make an authenticated API request

```
def implicit_credentials():
    from google.cloud import storage

    storage_client = storage.Client()
    buckets =
list(storage_client.list_buckets())
    print(buckets)
```

If you don't specify credentials when constructing the client, the client library will look for credentials in the environment.

Cloud Client Libraries use Application Default Credentials, or ADC, to find your application's credentials and let you access Google Cloud APIs. ADC uses a specific process to search for the application credentials. Your code will not need to change when you move it from your local development environment to a Google Cloud service like Cloud Run or GKE. Apps not running in Google Cloud can also use the same code.

ADC checks credential locations in the following order:

The existence of a GOOGLE_APPLICATION_CREDENTIALS environment variable is checked first. If the environment variable is set, ADC will use the service account file at the path specified by the environment variable.

If the environment variable isn't set, ADC next checks a well-known location for user credentials that are set using the gcloud CLI.

Finally, ADC will use an attached service account. Some services let you attach service accounts to a specific resource on the service. For example, Cloud Run and Cloud Functions allow the attachment of a service account to a specific service or function respectively. ADC will use the service account attached to the service or function.

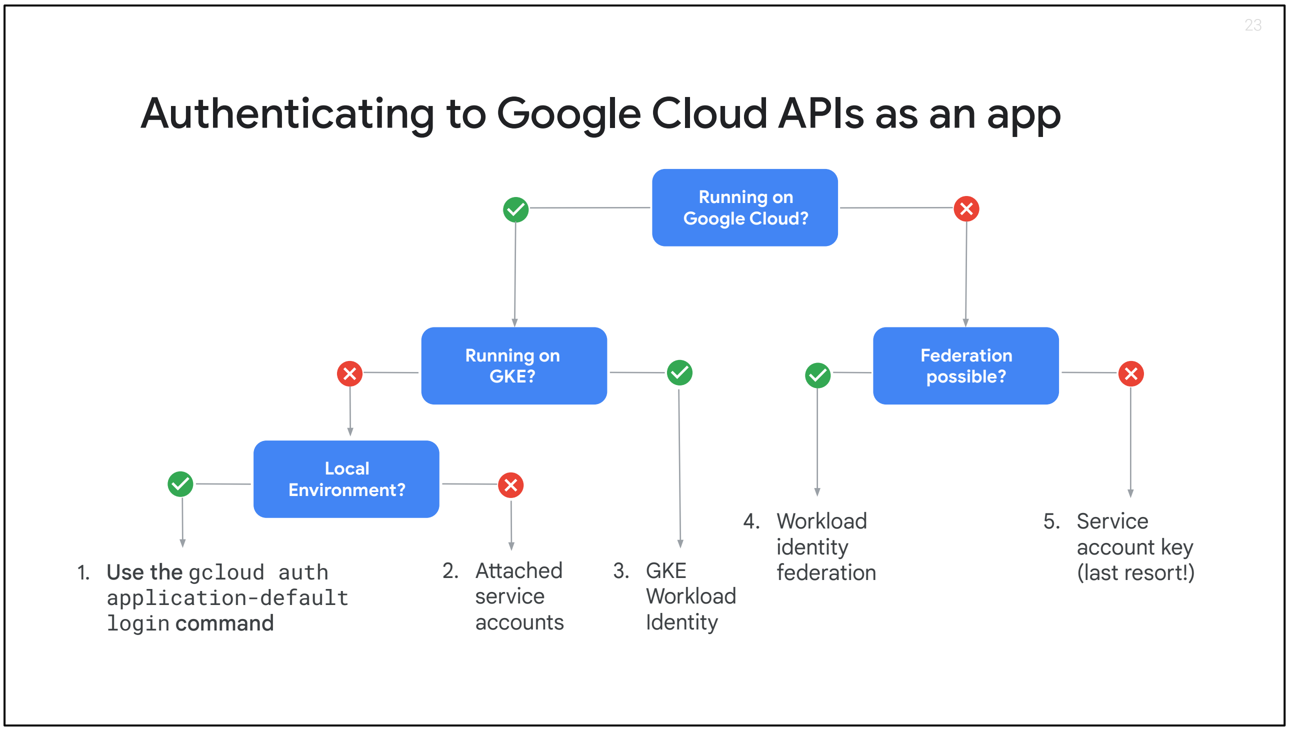
If there is not a service account attached to the specific resource, ADC will use the default service account for the service being used, like Compute Engine, GKE, Cloud

Run, or Cloud Functions.

If no credentials are found during all of these steps, an error will be thrown.

The example shows a Python function that automatically finds the credentials. The code makes an authenticated API request without specifying credentials, so ADC will search for the credentials.

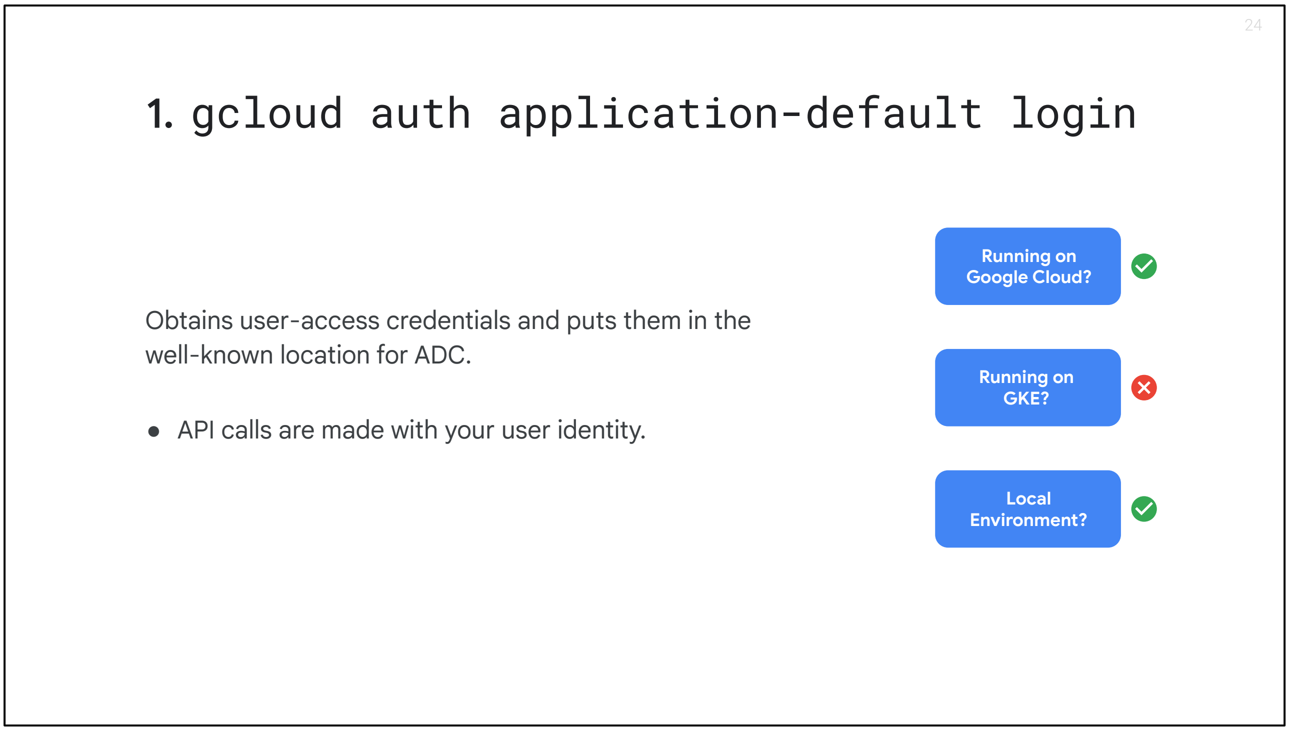
[Authentication at Google: https://cloud.google.com/docs/authentication/production
How Application Default Credentials works:
https://cloud.google.com/docs/authentication/application-default-credentials]



Authenticating to Google Cloud APIs as an app

1. Use the `gcloud auth application-default login` command
2. Attached service accounts
3. GKE Workload Identity
4. Workload identity federation
5. Service account key (last resort!)

Returning to our diagram, we explore each of the authentication methods.



1. `gcloud auth application-default login`

Obtains user-access credentials and puts them in the well-known location for ADC.

- API calls are made with your user identity.

Running on Google Cloud? ✓

Running on GKE? ✗

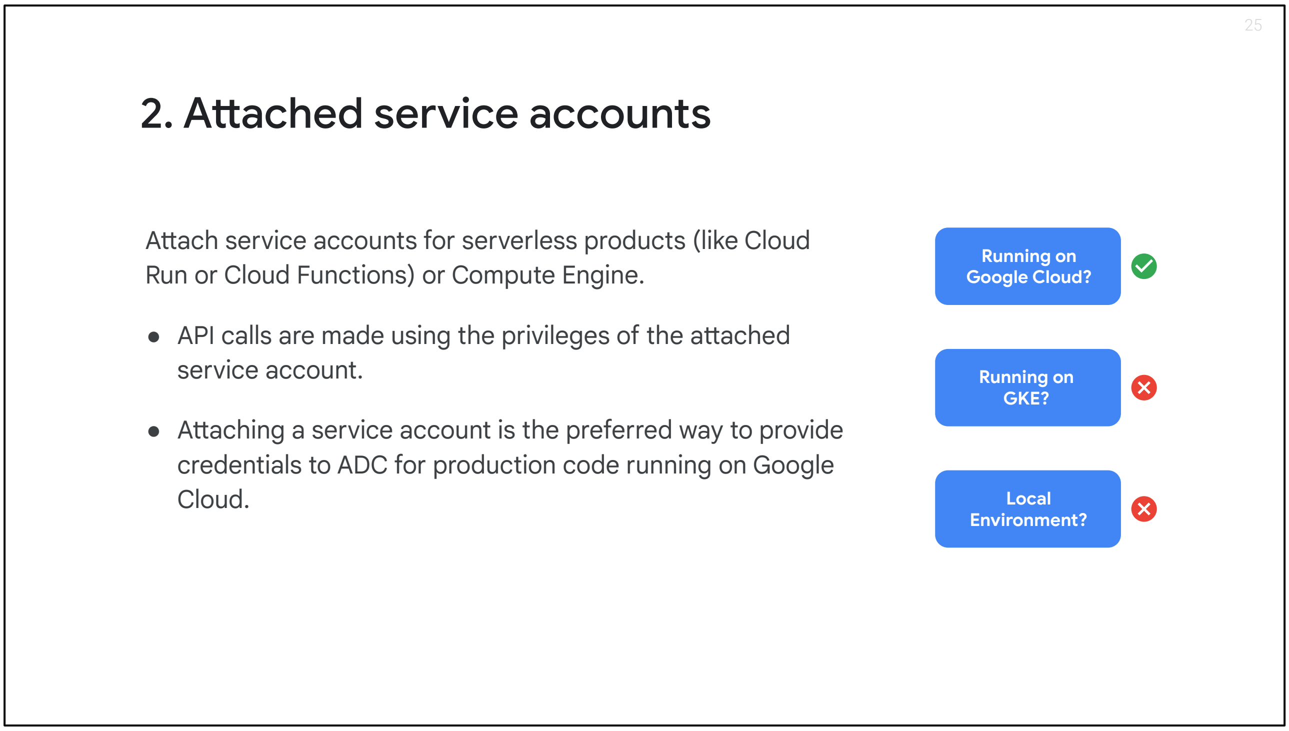
Local Environment? ✓

When you're using a local development environment, you can use the gcloud CLI command 'gcloud auth application-default login.' This command generates a JSON file containing user-access credentials and places it in the well-known location for ADC to find.

These user-access credentials allow the application to make API calls with your user identity. The JSON file is secure, does not include your password, and is time-bound.

If you have used the gcloud CLI before, you might be familiar with a similar command, 'gcloud auth login.' 'gcloud auth login' uses your credentials when you run gcloud CLI commands, but 'gcloud auth application-default login' uses your credentials when calling from your code.

[ADC credentials and gcloud credentials: https://cloud.google.com/docs/authentication/provide-credentials-adc#gcloud-credentials]



2. Attached service accounts

Attach service accounts for serverless products (like Cloud Run or Cloud Functions) or Compute Engine.

- API calls are made using the privileges of the attached service account.
- Attaching a service account is the preferred way to provide credentials to ADC for production code running on Google Cloud.

Running on Google Cloud?	✓
Running on GKE?	✗
Local Environment?	✗

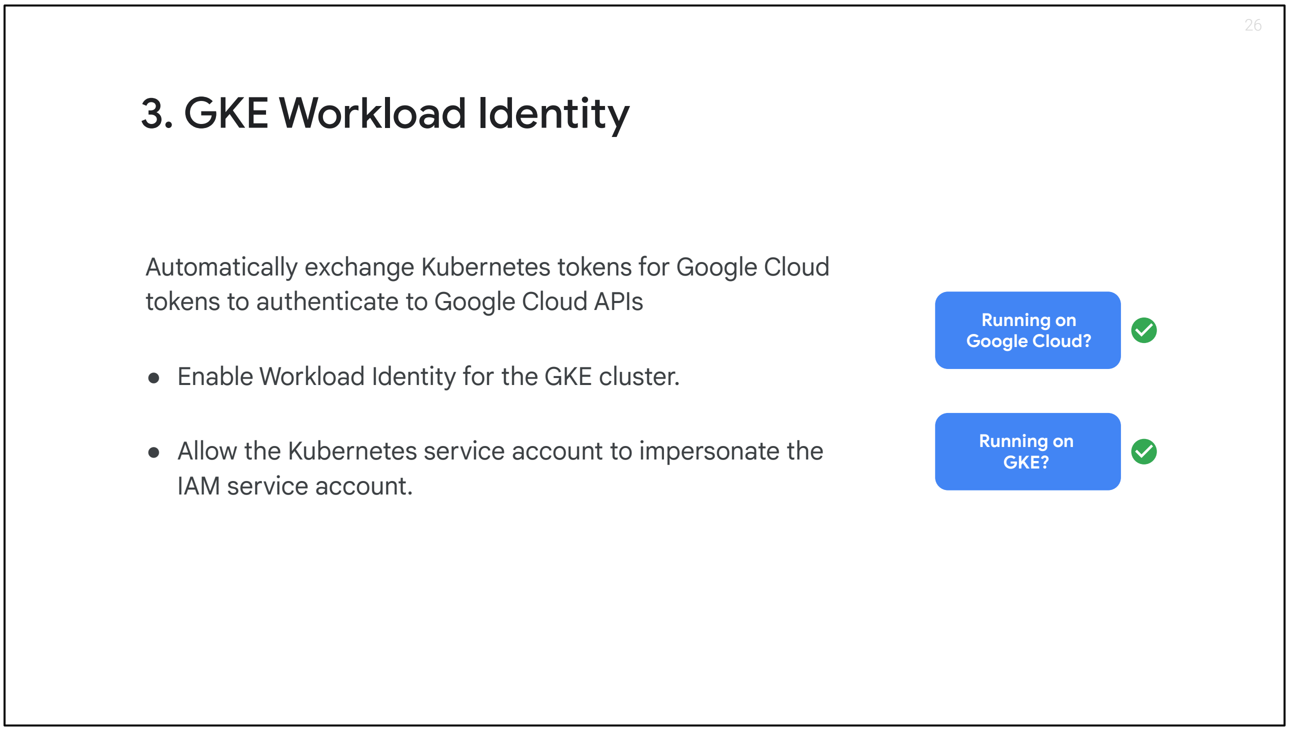
The preferred way to call Google Cloud APIs from applications running on Google Cloud serverless products or Compute Engine is to use attached service accounts.

When you run on a Compute Engine VM, replace the default service account for the VM with a service account created specifically for the application. That service account should be given only the privileges needed by your application.

Similarly, when you deploy a service to Cloud Run or create a function on Cloud Functions, attach a service account to the service or function.

Attaching a user-managed service account is the preferred way to provide credentials to ADC for production code that runs on Google Cloud.

[Attach service accounts to resources:
https://cloud.google.com/iam/docs/attach-service-accounts
Attached service accounts for ADC:
https://cloud.google.com/docs/authentication/provide-credentials-adc#attached-sa]



3. GKE Workload Identity

Automatically exchange Kubernetes tokens for Google Cloud tokens to authenticate to Google Cloud APIs

- Enable Workload Identity for the GKE cluster.
- Allow the Kubernetes service account to impersonate the IAM service account.

Running on Google Cloud? ✓
Running on GKE? ✓

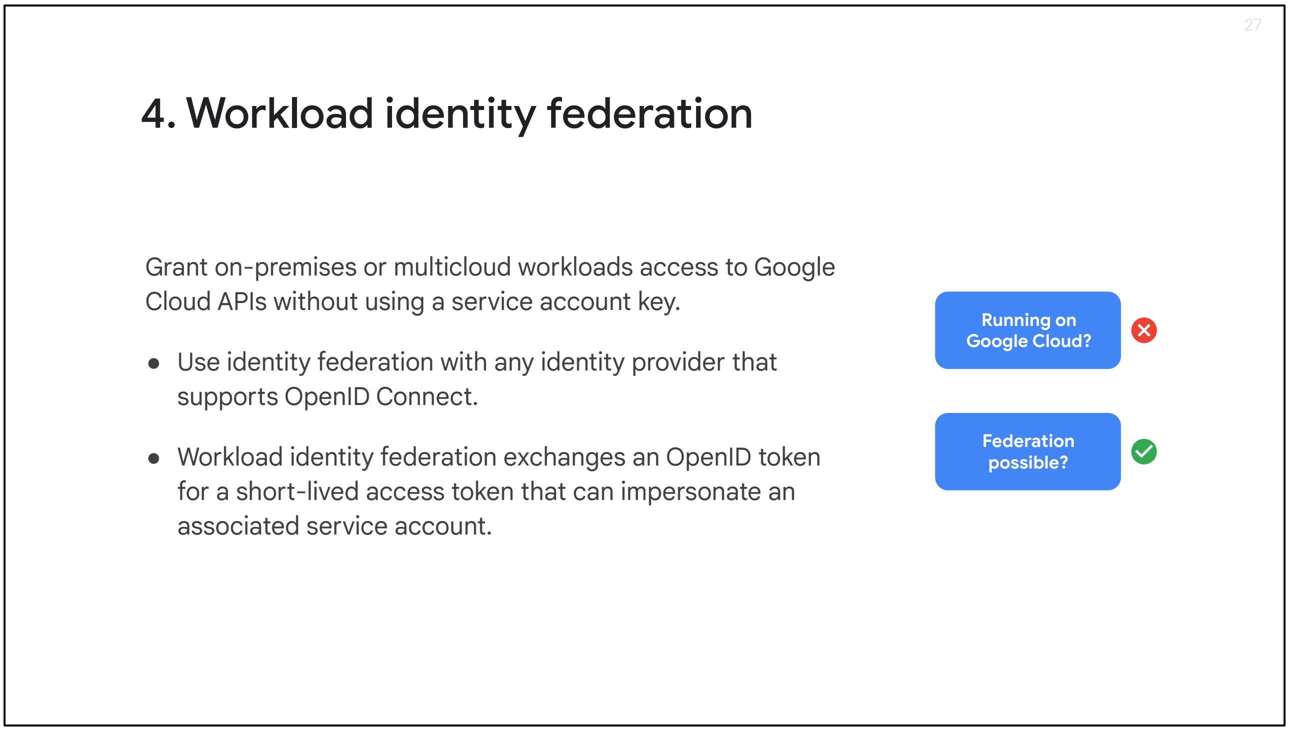
If you're running your application in GKE, Workload Identity is the recommended way to access Google Cloud APIs in a secure and manageable way.

Workload Identity allows a Kubernetes service account in your GKE cluster to act as the IAM service account when your application calls Google Cloud APIs. Using Workload Identity lets you assign distinct, fine-grained identities and authorization for each application in your cluster.

Workload Identity will automatically exchange Kubernetes service account tokens for IAM tokens when calling Google Cloud APIs.

The first step for using Workload Identity is to enable Workload Identity on your cluster. You then allow the Kubernetes service account to impersonate the IAM service account you have created. ADC will manage the rest.

[Workload Identity:
https://cloud.google.com/kubernetes-engine/docs/concepts/workload-identity
Use Workload Identity:
https://cloud.google.com/kubernetes-engine/docs/how-to/workload-identity]


27

4. Workload identity federation

Grant on-premises or multicloud workloads access to Google Cloud APIs without using a service account key.

- Use identity federation with any identity provider that supports OpenID Connect.
- Workload identity federation exchanges an OpenID token for a short-lived access token that can impersonate an associated service account.

Running on Google Cloud? ✗

Federation possible? ✓

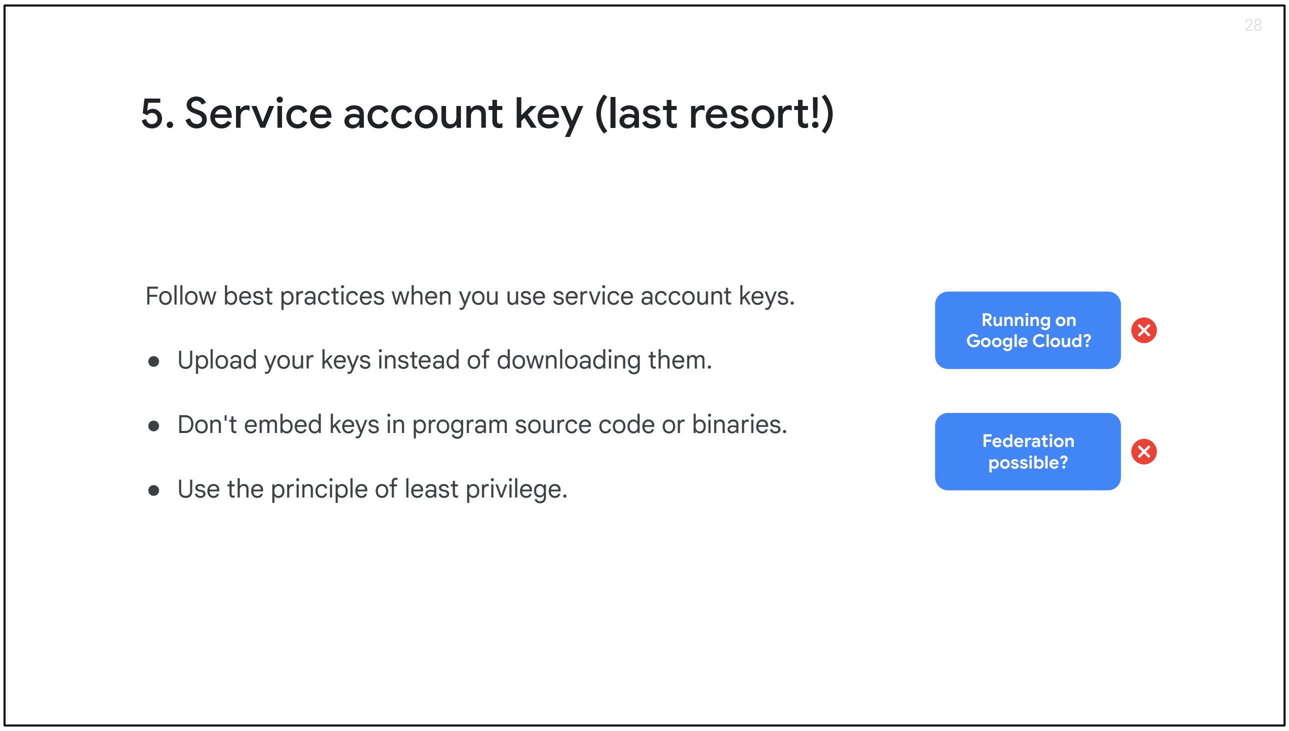
Sometimes you need to run applications outside of Google Cloud, whether you're running multi-cloud or on-premises. Identity federation for your workloads lets you grant on-premises or multi-cloud workloads access to Google Cloud APIs without requiring a service account key.

Traditionally, applications running outside Google Cloud have used service account keys for access. Service account keys are powerful credentials which can present a security risk when they are not managed correctly.

Workload identity federation removes the need for service account keys for external applications that use an identity provider that supports OpenID Connect.

An OpenID Connect ID token can be exchanged for a short-lived Google Cloud access token, which allows impersonation of an IAM service account. This process allows the external app to call Google Cloud services without having to secure or rotate a service account key.

[Workload identity federation:
https://cloud.google.com/iam/docs/workload-identity-federation
What is Workload Identity Federation? https://youtu.be/4vajaXzHN08]



5. Service account key (last resort!)

Follow best practices when you use service account keys.

- Upload your keys instead of downloading them.
- Don't embed keys in program source code or binaries.
- Use the principle of least privilege.

Running on Google Cloud?
Federation possible?

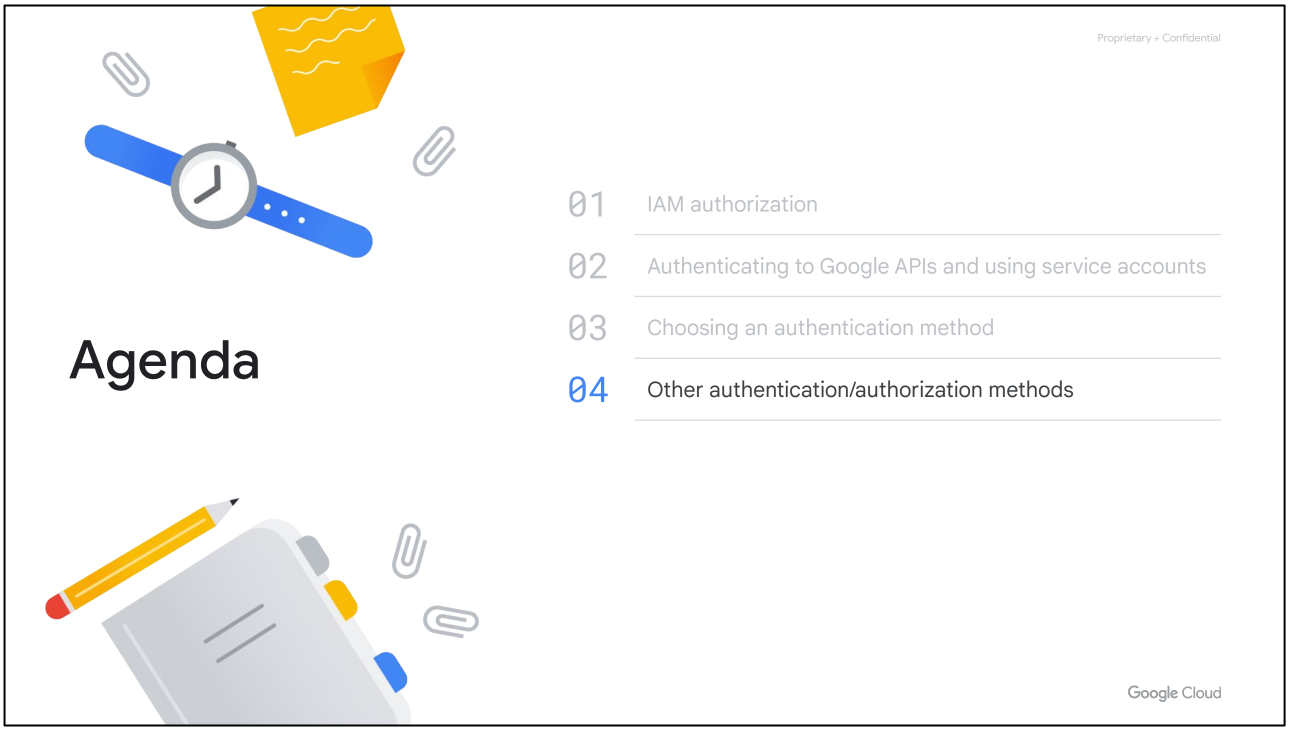
When you can't use federation, you might need to use a service account key to provide external applications with access to Google Cloud APIs. Using service account keys really is a last resort. If you must use service account keys, make sure you're using the best security practices.

You should upload a public key for your service account instead of downloading a private key created by Google. You can use your own infrastructure to generate a public and private key pair, and then upload the public key to Google and securely deliver the private key to your runtime environment. This leaves less chance of error when handling your private key.

Another best practice for service account keys is to never embed them in program source code or binaries. Keys can be extracted from binaries, and source code might be hosted in repositories, which makes keys more likely to be compromised.

Finally, ensure that your service accounts follow the principle of least privilege. Grant only the minimum set of permissions necessary for the associated application. If the service account key is somehow compromised, you will limit the access for the bad actor.

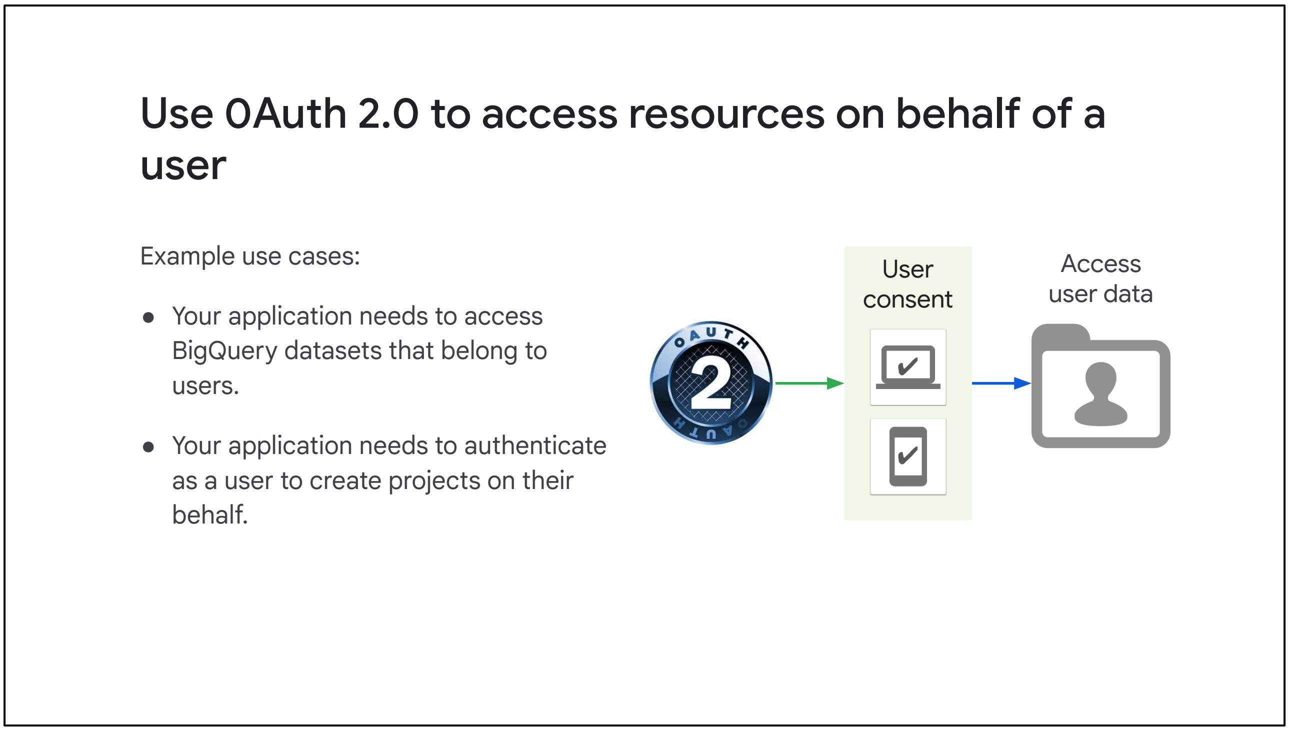
[Best practices for managing service account keys:
https://cloud.google.com/iam/docs/best-practices-for-managing-service-account-keys
Using IAM securely https://cloud.google.com/iam/docs/using-iam-securely]



Agenda

Google Cloud

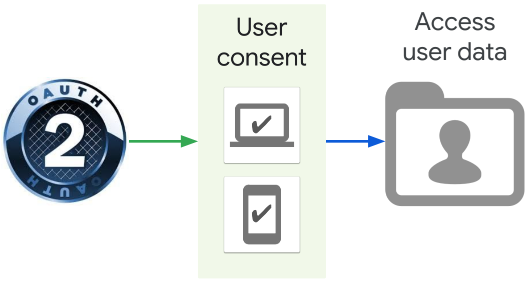
Now we discuss some other authentication and authorization methods for applications.



Use OAuth 2.0 to access resources on behalf of a user

Example use cases:

- Your application needs to access BigQuery datasets that belong to users.
- Your application needs to authenticate as a user to create projects on their behalf.



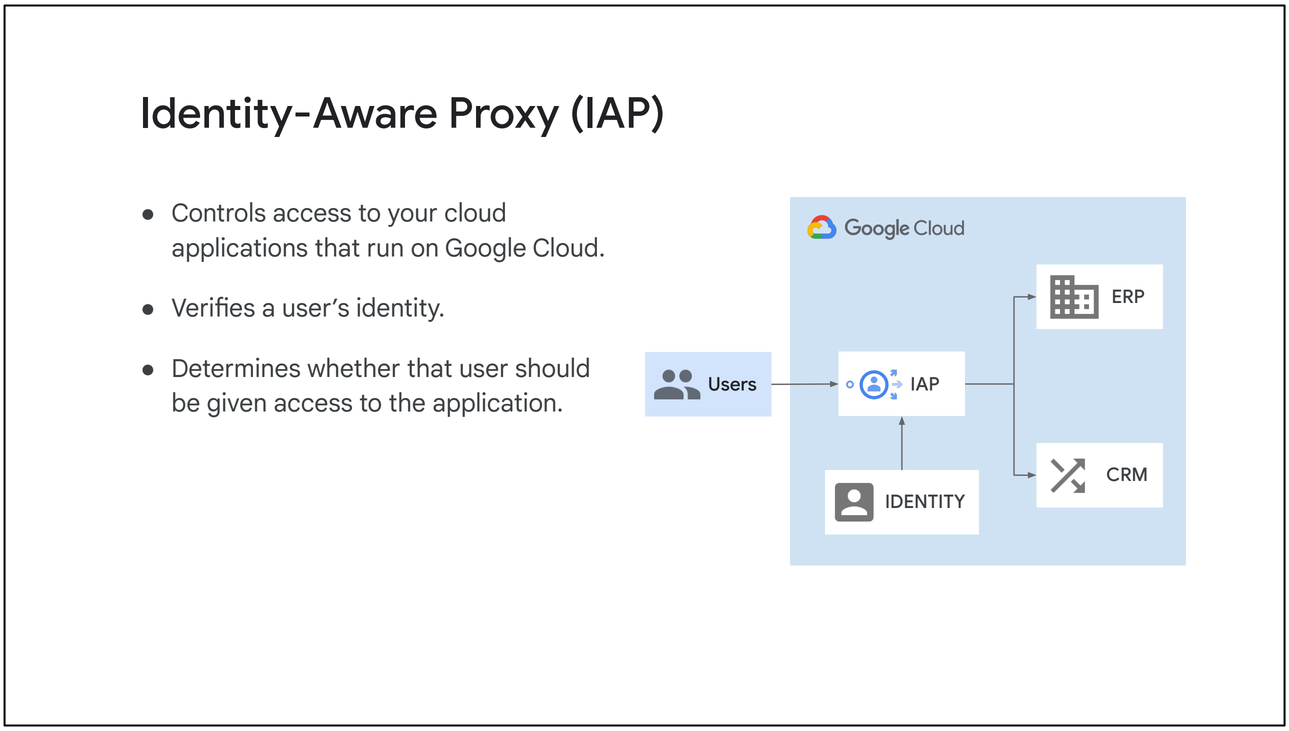
Your applications might need to access resources on behalf of a user.

Examples of use cases where you might want to access resources on behalf of a user include:

- Your application needs access to BigQuery datasets that belong to your application users, or
- Your application needs to authenticate as a user to create projects or resources on their behalf.

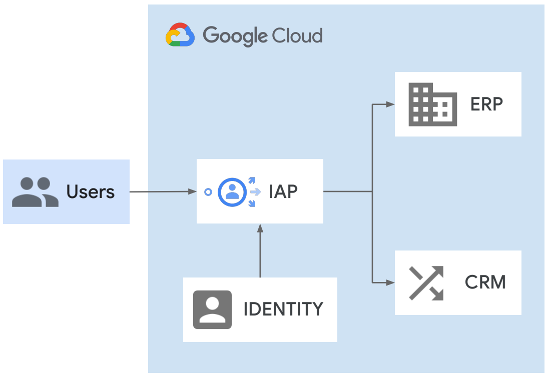
You can use the OAuth 2.0 protocol to access resources on behalf of a user. When your application requests access to the resources, the user is prompted for consent. If consent is provided, the application can then request credentials from an authorization server. The application can use those credentials to access resources on behalf of the user.

[OAuth 2.0: https://developers.google.com/identity/protocols/OAuth2
Authenticating as an End User:
https://cloud.google.com/docs/authentication/end-user]



Identity-Aware Proxy (IAP)

- Controls access to your cloud applications that run on Google Cloud.
- Verifies a user's identity.
- Determines whether that user should be given access to the application.



Another way to provide access to users is to use Identity-Aware Proxy, or IAP.

IAP controls access to your cloud applications running in your Google Cloud project.

IAP verifies a user's identity and determines whether that user should be given access to the application based on configuration. The developer does not need to write code to control access.

IAP lets you establish a central authorization layer for applications accessed by HTTPS. IAP helps you adopt an application-level access control model instead of relying on VPNs, network level firewalls, or complex authorization code in your applications.

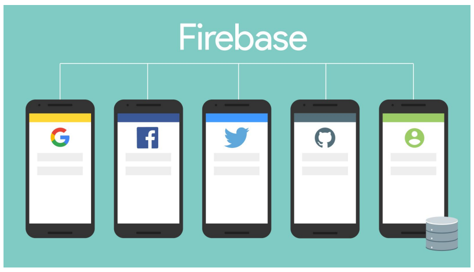
[https://cloud.google.com/iap/docs/]

Using Firebase Authentication

Firebase Authentication:

- Provides federated authentication using passwords, phone, and providers like Google and Facebook.
- Includes drop-in auth components, SDKs, and ready-made UI libraries
- Supports OAuth 2.0 and OpenID Connect to connect to backend services.

Firebase



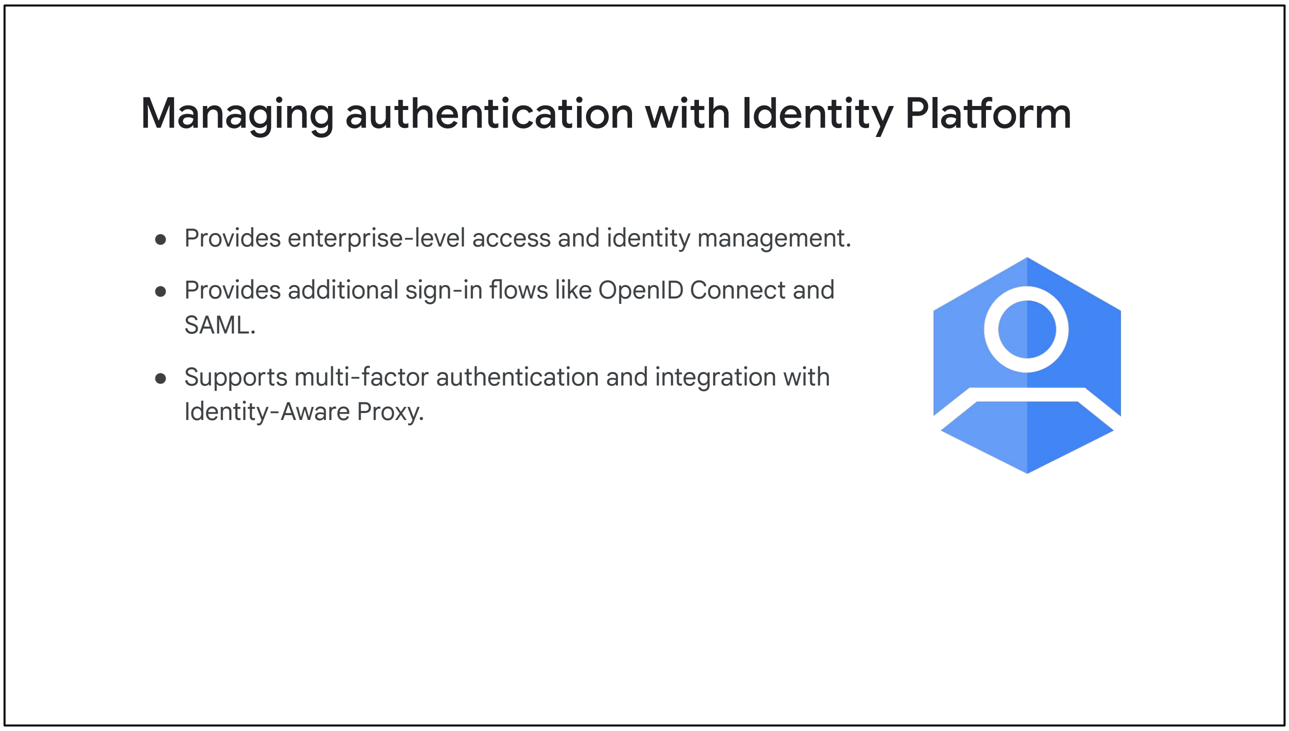
Firebase is an app development platform that helps you build mobile applications. Firebase Authentication provides features to help you add authentication and identity management to your mobile apps.

Firebase Auth supports authentication using passwords, phone numbers, and popular federated identity providers like Google, Apple, and GitHub.

Firebase Auth provides drop-in auth components that handle UI flows for sign-up and signing in and edge cases like account recovery. SDKs and ready-made UI libraries make development easy.

After a successful login, you can access the user's profile and use the provided token in OAuth 2.0 and OpenID Connect flows for backend services.

[Firebase Authentication: https://firebase.google.com/docs/auth/#how_does_it_work]



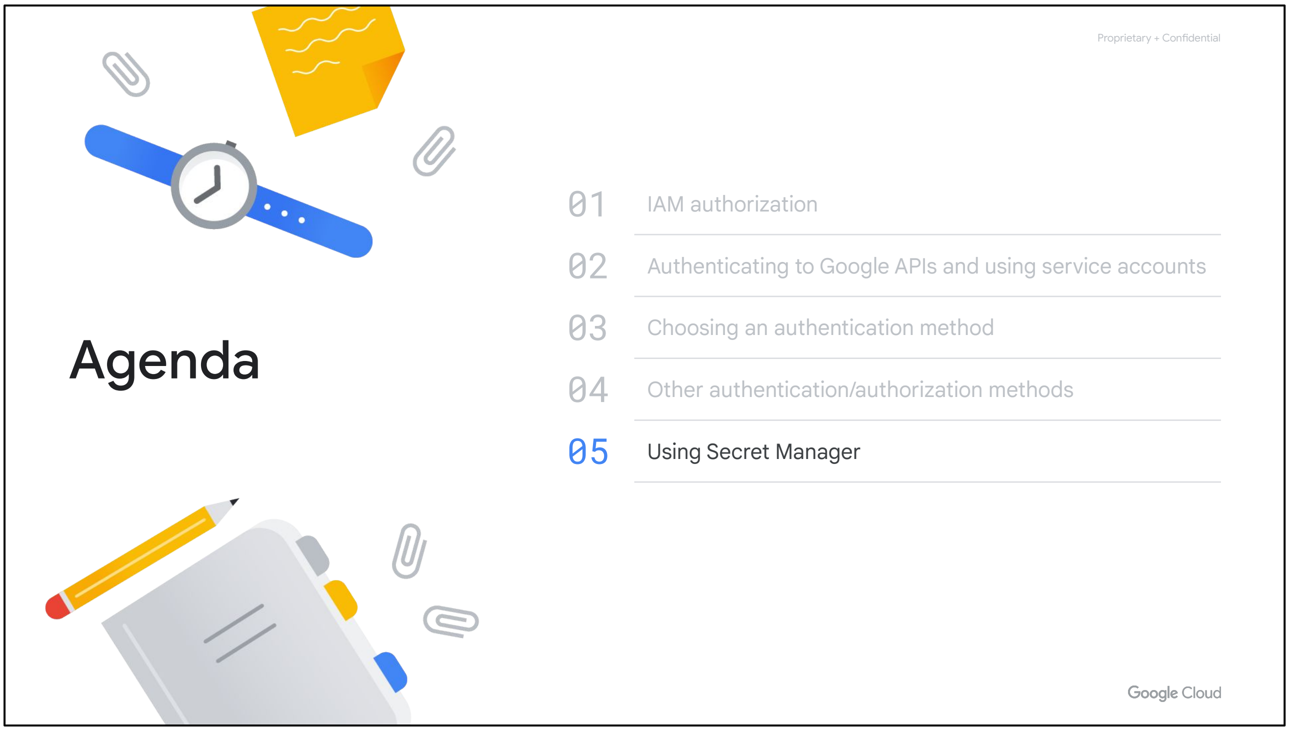
Managing authentication with Identity Platform

- Provides enterprise-level access and identity management.
- Provides additional sign-in flows like OpenID Connect and SAML.
- Supports multi-factor authentication and integration with Identity-Aware Proxy.

Identity Platform and Firebase Authentication offer similar functionality. Both let you easily sign users in to your apps by providing backend services, SDKs, and UI libraries. However, Identity Platform offers additional capabilities designed for enterprise customers.

Identity Platform supports signing in with OpenID Connect and SAML authentication.

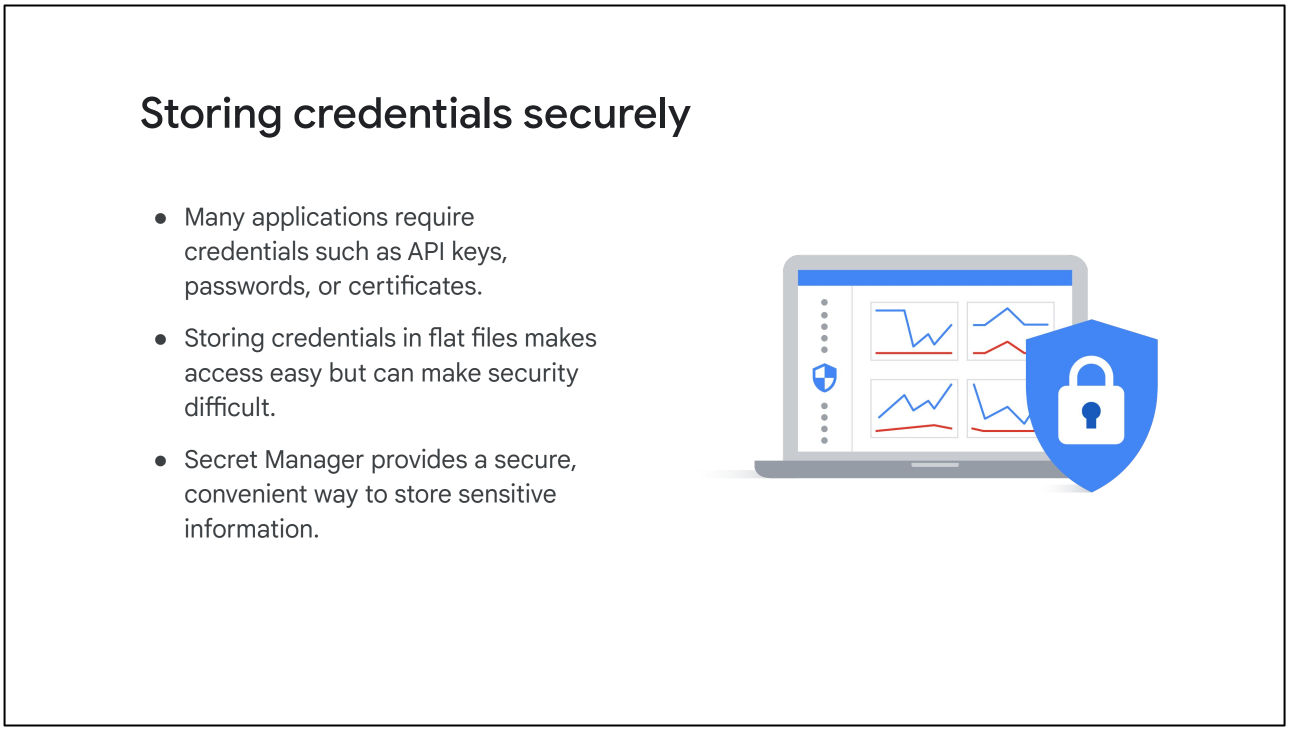
Other enterprise features such as multi-factor authentication and integration with Identity-Aware Proxy are also supported.



Agenda

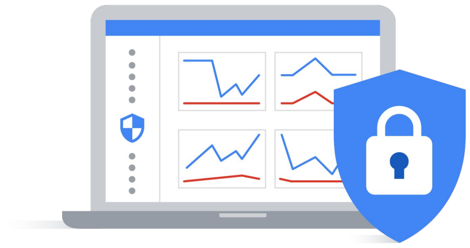
Google Cloud

Next, we discuss Secret Manager.



Storing credentials securely

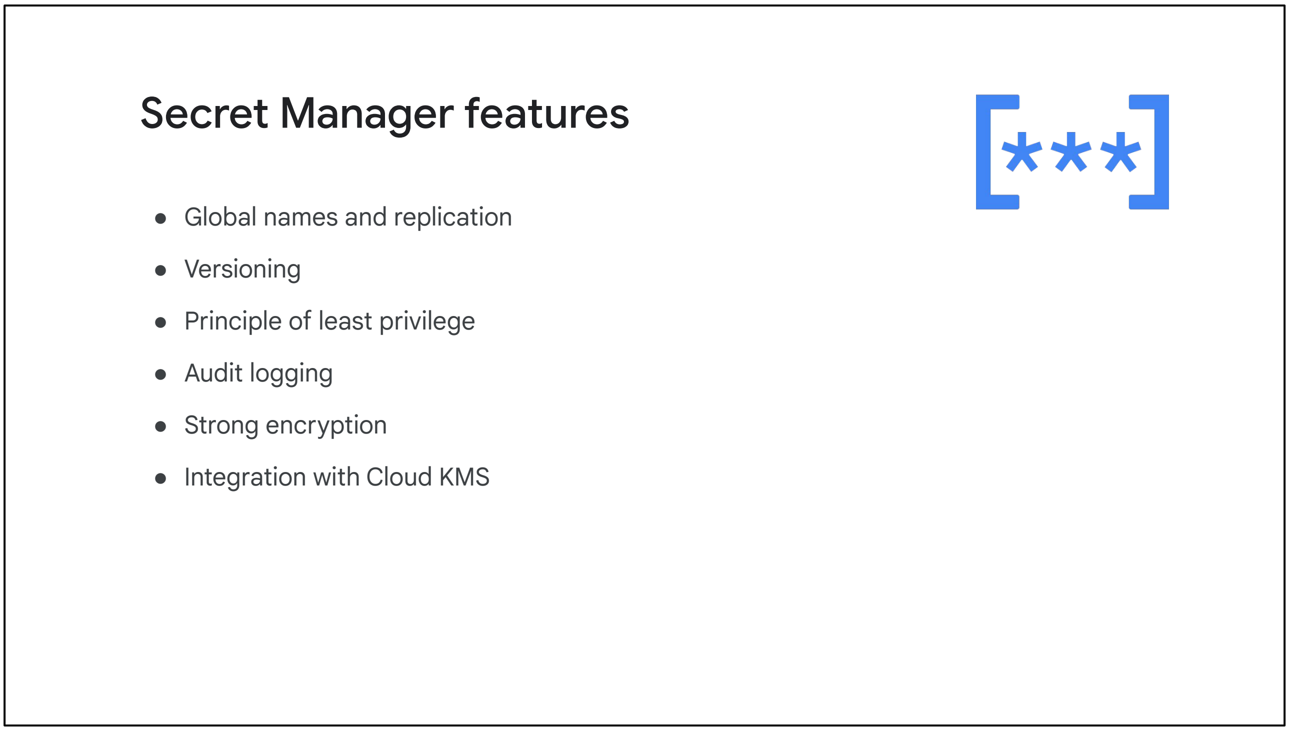
- Many applications require credentials such as API keys, passwords, or certificates.
- Storing credentials in flat files makes access easy but can make security difficult.
- Secret Manager provides a secure, convenient way to store sensitive information.



To authenticate with other services and applications, many applications require credentials such as API keys, passwords, or certificates. These credentials must be stored securely.

Developers often think about storing the credentials in flat files. Access becomes incredibly easy, but security can be difficult. Not everyone should be able to see this information, so you'd have to use the file operating system to restrict access. However, using flat files can lead to secrets being scattered across your organization's cloud and on-premises infrastructure.

Secret Manager provides a secure and convenient way to store sensitive information. You can store, manage, and access secrets as binary blobs or text strings. Only users and applications with the appropriate permissions can access the secret. Keeping all of your secrets in a single place, and using IAM to determine who has access, simplifies secure management of your secrets.



Secret Manager features

- Global names and replication
- Versioning
- Principle of least privilege
- Audit logging
- Strong encryption
- Integration with Cloud KMS

Now we take a quick look at some of the features of Secret Manager.

The name of a secret stored in Secret Manager is global, but the secret data can optionally be stored in a specified region.

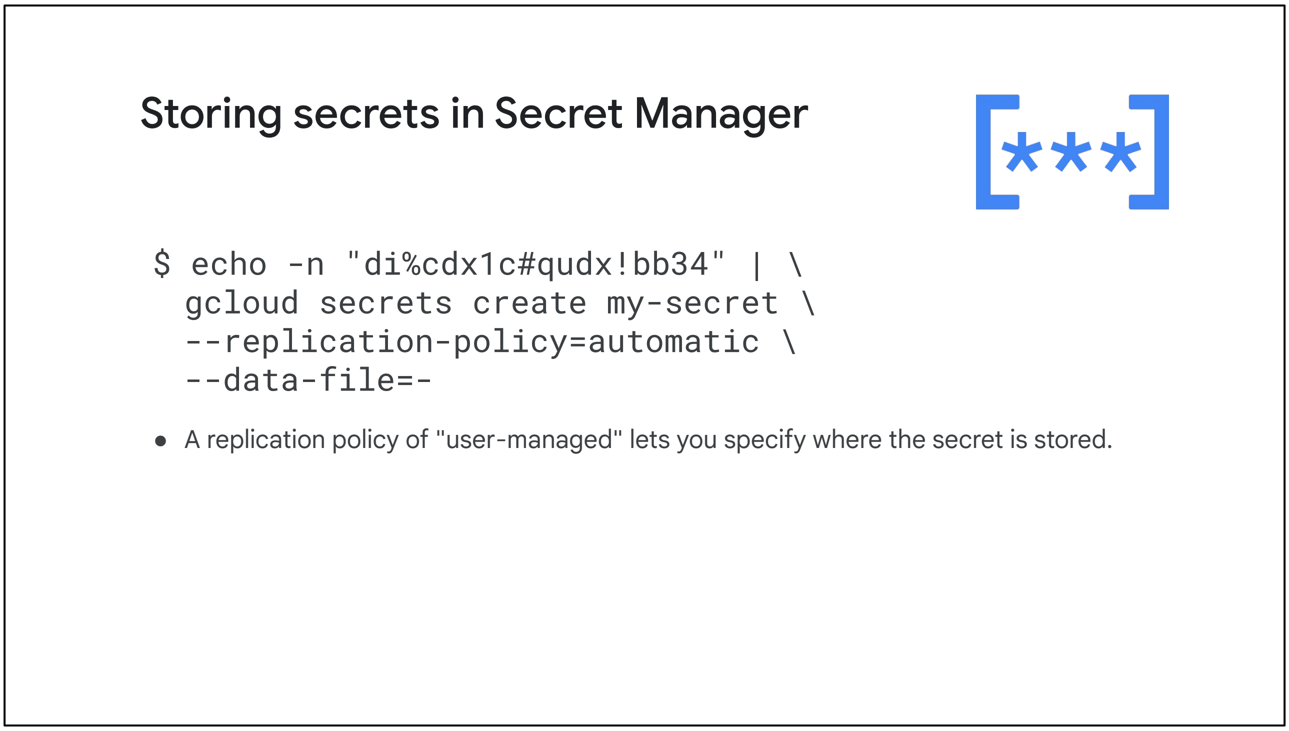
Secrets can be versioned. Each version can have different secret data. There is no limit on the number of versions that can be stored. Versions cannot be modified, but they can be deleted.

Secret Manager also follows the principle of least privilege. Secrets are created at the project level, and only project owners start with permission to create and access secrets within the project. Other roles must be explicitly granted IAM permissions to access secrets.

When Cloud Audit Logs are enabled, every interaction with Secret Manager, including reads and updates, generates an audit entry. This auditing lets you verify that only the appropriate users and apps are accessing your secrets.

Secret Manager manages server-side encryption keys on your behalf by using the same hardened key management systems used for Google's own encrypted data.

Secret Manager also integrates with Cloud Key Management Service, or Cloud KMS. You can have Cloud KMS encrypt the version for a secret before storing it. After retrieving the version, Cloud KMS will also be needed to decrypt it.



Storing secrets in Secret Manager

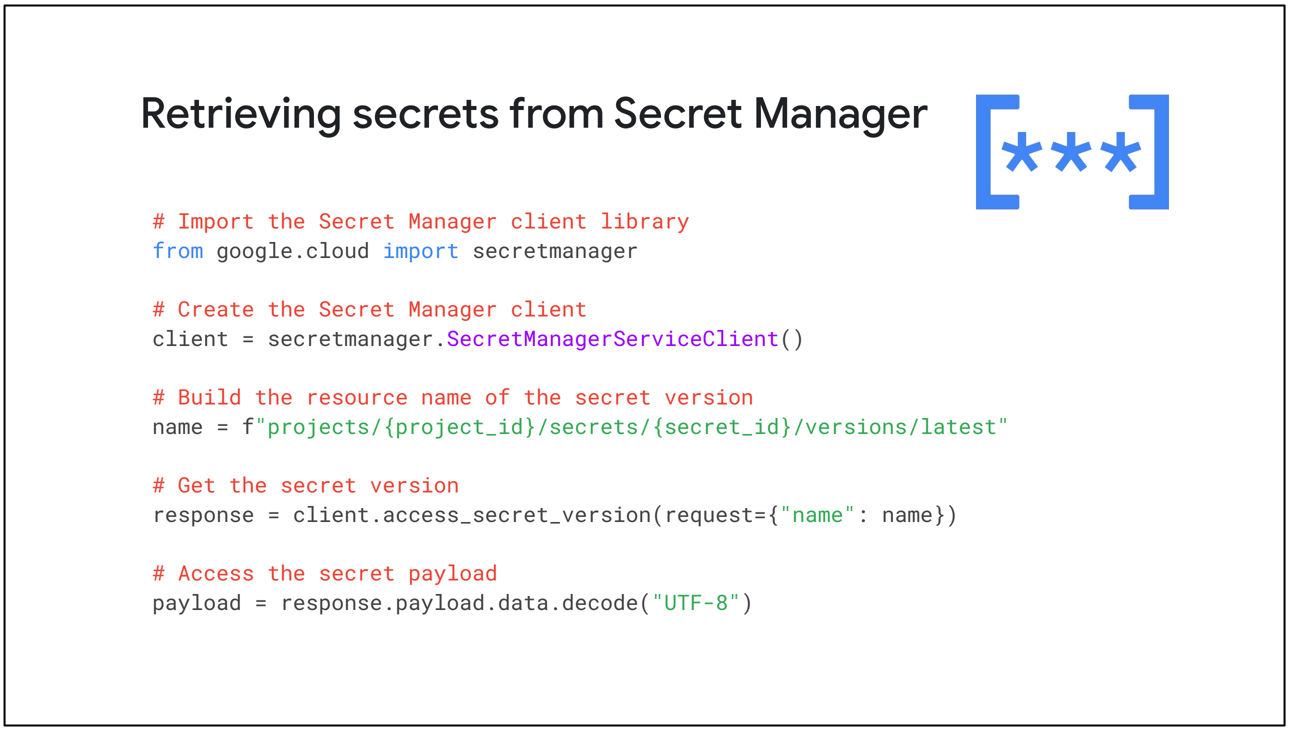
```
$ echo -n "di%cdx1c#qudx!bb34" | \
  gcloud secrets create my-secret \
  --replication-policy=automatic \
  --data-file=-
```

- A replication policy of "user-managed" lets you specify where the secret is stored.

Next, we look at how Secret Manager works. First, we create a secret.

You can use the Google Cloud console, the gcloud CLI, or code to create the secret. In this example, we create the secret by using a gcloud command.

The command shown here creates a secret named "my-secret." The data file parameter takes the secret from standard input, passed in using the piped echo command. The replication policy is set to automatic, which lets the secret payload be stored in any region. A replication policy of "user-managed" lets you specify the regions where the secret might be stored.



Retrieving secrets from Secret Manager

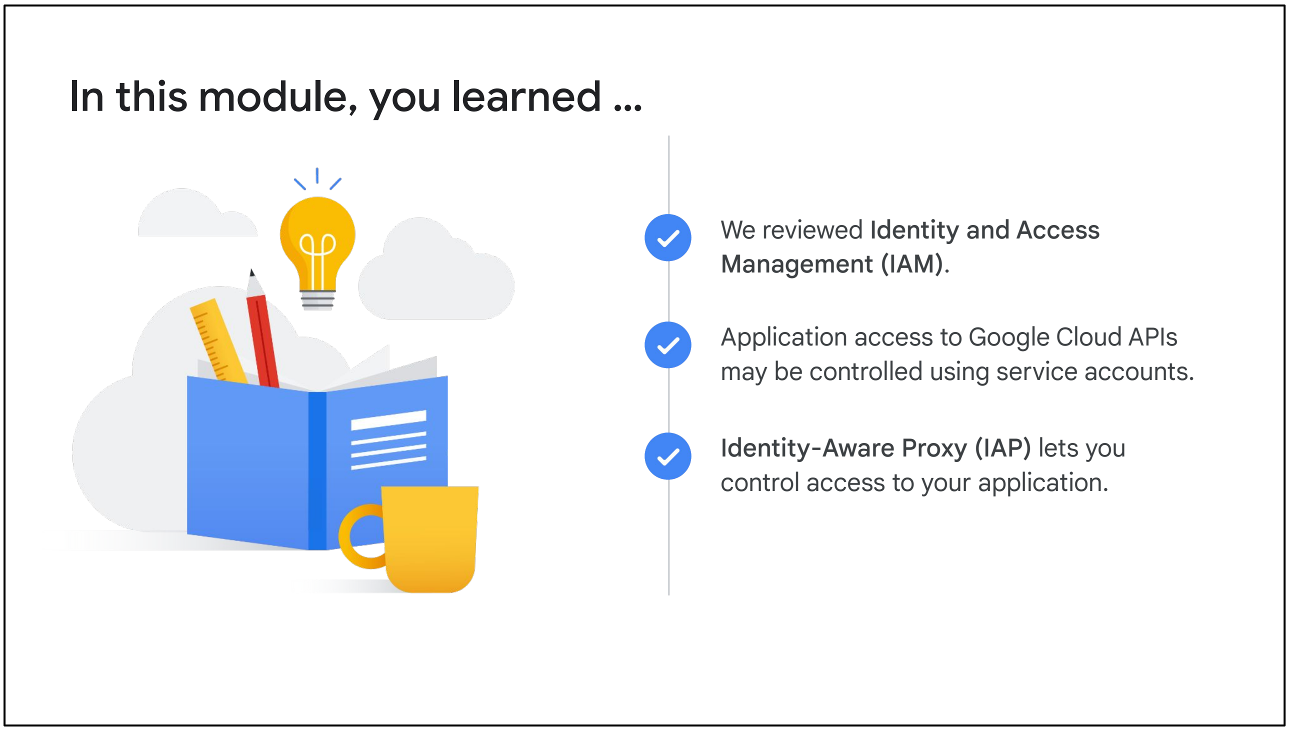
```python
# Import the Secret Manager client library
from google.cloud import secretmanager

# Create the Secret Manager client
client = secretmanager.SecretManagerServiceClient()

# Build the resource name of the secret version
name = f"projects/{project_id}/secrets/{secret_id}/versions/latest"

# Get the secret version
response = client.access_secret_version(request={"name": name})

# Access the secret payload
payload = response.payload.data.decode("UTF-8")
```

You can also retrieve secrets using the Google Cloud console, the gcloud CLI, or code. In this example, we use Python code to retrieve the secret.

The Secret Manager Python SDK can be used to create, update, delete, or access secrets.

The first step is to import the secretmanager library and create a client. Secrets are accessed by resource name, which includes the project ID, the ID or name of the secret, and the version you want to access. Versions are ordinal numbers: 1, 2, 3, and so on. You can also use the keyword "latest" to access the latest version for the secret.

Retrieving the secret is as easy as passing the name to the library function, and decoding the payload. This process provides a secure method for using sensitive data within your applications.



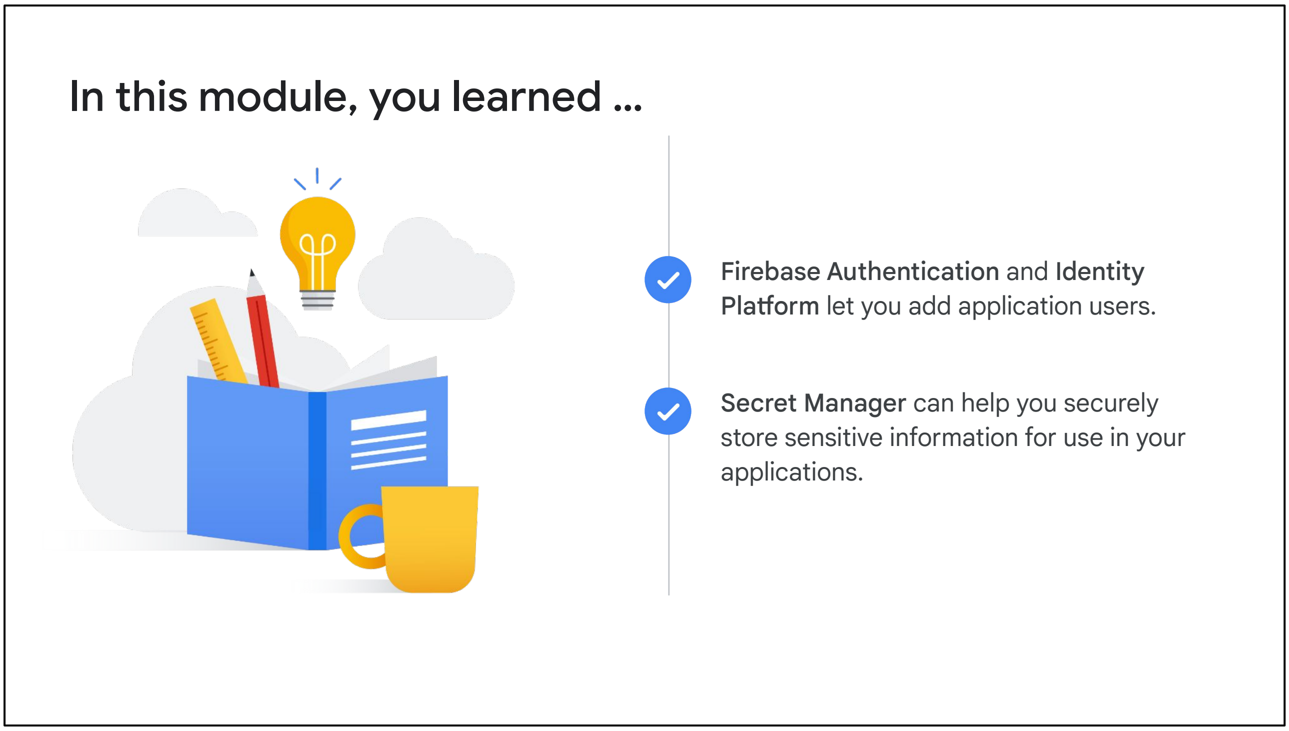
In this module, you learned ...

- ✓ We reviewed **Identity and Access Management (IAM)**.
- ✓ Application access to Google Cloud APIs may be controlled using service accounts.
- ✓ **Identity-Aware Proxy (IAP)** lets you control access to your application.

We began the discussion by reviewing authorization with Identity and Access Management (IAM). IAM lets you manage access control by defining who has what access for which resource.

Application access to Google Cloud APIs may be controlled using service accounts. Your application assumes the identity of the service account to invoke Google Cloud APIs. You can create one or more service accounts to restrict access to different resources in your application. You also learned about risks associated with service account keys. A diagram helped explain how to decide which authentication method is best for your application.

Identity-Aware Proxy, or IAP for short, lets you control access to your application. It verifies the user's identity and checks whether that user should be allowed to access the application. End-users simply use an internet-accessible URL to access IAP-secured applications. No VPN is required!



In this module, you learned ...

- ✓ **Firebase Authentication** and **Identity Platform** let you add application users.
- ✓ **Secret Manager** can help you securely store sensitive information for use in your applications.

Firebase Authentication and Identity Platform let you add application users, and we discussed the differences between them.

Finally, we looked at Secret Manager, which can help you securely store sensitive information for use in your applications.