Authentication overview

cloud.google.com/docs/authentication

This page provides an overview of authentication in Google Cloud's platform for application developers. It describes principals, application credentials, and various ways to authenticate calls to Google Cloud APIs.

Important: This page does not cover authentication for non-Google Cloud APIs, such as the Google Maps Platform APIs. For information about API keys for Google Maps Platform APIs and SDKs, see the Google Maps documentation.

Introduction

Access control for Google Cloud APIs encompasses authentication, authorization, and auditing. Authentication determines who you are, authorization determines what you can do, and auditing logs what you did.

This page focuses on authentication. For authorization, see Identity and Access Management (IAM). For auditing, see Cloud Audit Logs.

Principals

A principal is an entity, also known as an identity, that can be granted access to a resource. Google Cloud APIs support two types of principals: *user accounts* and *service accounts*:

- **User accounts** are managed as Google Accounts, and they represent a developer, administrator, or any other person who interacts with Google Cloud. They are intended for scenarios where your application needs to access resources on behalf of a human user. See Authenticating as an end user for more information.

- **Service accounts** are managed by IAM, and they represent non-human users. They are intended for scenarios where your application needs to access resources or perform actions on its own, such as running App Engine apps or interacting with Compute Engine instances. See Authenticating as a service account for more information.

For more information about each account type, see the IAM overview.

Applications

Google Cloud APIs only accept requests from *registered applications*, which are uniquely identifiable applications that present a credential at the time of the request. Requests from anonymous applications are rejected.

Application credentials provide the required information about the caller making a request to a Google Cloud API. Valid credential types include API keys, OAuth 2.0 client credentials, or service account keys. Service accounts are unique, because they can be used as both an application credential or a principal identity. See Understanding service accounts for more information.

Presenting application credentials in requests to Google Cloud APIs only identifies the caller as a registered application; if authentication is required, the client must also identify the principal running the application, such as a user account or service account. This process is described in the section below.

Authentication strategies

Google Cloud APIs use the OAuth 2.0 protocol for authenticating both user accounts and service accounts. The OAuth 2.0 authentication process determines both the principal and the application.

Most Google Cloud APIs also support anonymous access to public data using API keys. However, API keys only identify the application, not the principal. When using API keys, the principal must be authenticated by other means.

Google Cloud APIs support multiple authentication flows for different runtime environments. For the best developer experience, we recommend using Google Cloud Client Libraries with Google Cloud APIs. They use Google-provided authentication libraries that support a variety of authentication flows and runtime environments.

To build an application using Google Cloud APIs, follow these general steps:

- Choose and use the provided Google Cloud Client Libraries
- Determine the correct authentication flow for your application
- Find or create the application credentials needed for your application
- Pass the application credentials to the client libraries at application startup time, ideally through Application Default Credentials (ADC)

You should choose application credentials based on what your application needs and where it runs. The following table provides some general recommendations for common requirements:

Requirement	Recommendation	Comment
Accessing public data anonymously	API key	An API key only identifies the application and doesn't require user authentication. It is sufficient for accessing public data.

Requirement	Recommendation	Comment
Accessing private data on behalf of an end user	OAuth 2.0 client	An OAuth 2.0 client identifies the application and lets end users authenticate your application with Google. It allows your application to access Google Cloud APIs on behalf of the end user.
Accessing private data on behalf of a service account inside Google Cloud environments	Environment-provided service account	If your application runs inside a Google Cloud environment, such as Compute Engine, App Engine, GKE, Cloud Run, or Cloud Functions, your application should use the service account provided by the environment. Google Cloud Client Libraries will automatically find and use the service account credentials.
Accessing private data on behalf of a service account outside Google Cloud environments	Service account key	You need to create a service account, and download its private key as a JSON file. You need to pass the file to Google Cloud Client Libraries, so they can generate the service account credentials at runtime. Google Cloud Client Libraries will automatically find and use the service account credentials by using the `GOOGLE_APPLICATION_CREDENTIALS` environment variable.

Examples

The following code examples show how to use different authentication strategies using Go language version of the Pub/Sub client library. The developer experience for other languages is nearly the same.

```
import (
        "context"
        "fmt"
        "cloud.google.com/go/pubsub"
)
// serviceAccount shows how to use a service account to authenticate.
func serviceAccount() error {
        // Download service account key per
https://cloud.google.com/docs/authentication/production.
        // Set environment variable
GOOGLE_APPLICATION_CREDENTIALS=/path/to/service-account-key.json
        // This environment variable will be automatically picked up by the
client.
        client, err := pubsub.NewClient(context.Background(), "your-project-id")
        if err != nil {
                return fmt.Errorf("pubsub.NewClient: %v", err)
        }
        defer client.Close()
        // Use the authenticated client.
        _ =

 client


        return nil
}
```

For more information, see Authenticating as a service account.

What's next

- Learn about authenticating as an end user
- Learn about authenticating as a service account
- Learn about using API keys

Rate and review

Last updated 2021-07-08 UTC.