Roles for service account authentication

Archived: 2026-04-05 16:43:17 UTC

Principals can use service accounts to authenticate in a few different ways. Each type of authentication requires the principal to have specific Identity and Access Management (IAM) permissions on the service account.

This page describes the roles that you can grant to principals to let them impersonate service accounts or attach service accounts to resources. It also describes the permissions that you need in common scenarios.

To learn about different ways to authenticate with a service account, see Service account credentials and Service account impersonation.

This section describes the roles that let principals authenticate with service accounts. To learn how to grant and revoke these roles, see Manage access to service accounts.

Service Account User role

The Service Account User role (`roles/iam.serviceAccountUser`) lets a principal attach a service account to a resource. When the code running on that resource needs to authenticate, it can get credentials for the attached service account.

This role does *not* allow principals to create short-lived credentials for service accounts, or to use the `--impersonate-service-account` flag for the Google Cloud CLI. To complete these tasks, you need the Service Account Token Creator role on the service account.

Service Account Token Creator role

The Service Account Token Creator role (`roles/iam.serviceAccountTokenCreator`) lets principals create short-lived credentials for a service account.

The Service Account Token Creator role lets you create the following types of short-lived credentials:

- OAuth 2.0 access tokens, which you can use to authenticate with Google APIs
- OpenID Connect (OIDC) ID tokens
- Signed JSON Web Tokens (JWTs) and binary blobs

The Service Account Token Creator role also lets principals use the `--impersonate-service-account` flag for the gcloud CLI. When you use this flag, the gcloud CLI automatically creates short-lived credentials for the service account.

The role's permissions include the following:

- `iam.serviceAccounts.getAccessToken` : lets you create OAuth 2.0 access tokens
- `iam.serviceAccounts.getOpenIdToken` : lets you create OpenID Connect (OIDC) ID tokens

- `iam.serviceAccounts.implicitDelegation` : lets service accounts get tokens in a delegation chain
- `iam.serviceAccounts.signBlob` : lets you sign binary blobs
- `iam.serviceAccounts.signJwt` : lets you sign JWTs

Service Account OpenID Connect Identity Token Creator

The Service Account OpenID Connect Identity Token Creator role (`roles/iam.serviceAccountOpenIdTokenCreator`) lets principals create short-lived OIDC ID tokens. If you only need to create OIDC ID tokens, use this role. If you need to create other types of tokens, use the Service Account Token Creator role instead.

The role includes the `iam.serviceAccounts.getOpenIdToken` permission, which lets you create an OIDC ID token.

Workload Identity User role

The Workload Identity User role (`roles/iam.workloadIdentityUser`) lets principals impersonate service accounts from GKE workloads.

The role's permissions include the following:

- `iam.serviceAccounts.getAccessToken` : lets you create OAuth 2.0 access tokens
- `iam.serviceAccounts.getOpenIdToken` : lets you create OpenID Connect (OIDC) ID tokens

Service account permissions for common scenarios

Service accounts can be used in many different scenarios, and each of them requires certain permissions. This section describes common scenarios and what permissions are required.

Attaching service accounts to resources

If you want to start a long-running job that authenticates as a service account, you need to attach a service account to the resource that will run the job.

Permissions:

- Permissions to create the resource
- `iam.serviceAccounts.actAs`

To find roles that include these permissions, search the roles list for the permissions.

There are several different Google Cloud resources that can run long-running jobs as service accounts. Some examples of these resources include:

- Compute Engine VMs
- App Engine apps
- Cloud Run functions

When you create these resources, you have the option to attach a service account. This service account acts as the resource's identity.

To create a resource and attach a service account, you need permissions to create that resource and permission to attach the service account to resources. Permission to attach service accounts to resources is provided by any role that includes the `iam.serviceAccounts.actAs` permission—for example, the Service Account User role (roles/iam.serviceAccountUser).

After you create the resource and attach a service account to it, you can start a long-running job on the resource. The job runs as the service account that is attached to the resource, and uses that service account to authorize requests to Google Cloud APIs.

To learn more about attaching service accounts to resources, see Attaching a service account to a resource.

Impersonating a service account

Permissions:

- `iam.serviceAccounts.getAccessToken`
- `iam.serviceAccounts.signBlob`
- `iam.serviceAccounts.signJwt`
- `iam.serviceAccounts.implicitDelegation`

Roles:

- `roles/iam.serviceAccountTokenCreator` (Service Account Token Creator)

Once granted the required permissions, a user (or another service account) can impersonate the service account in a few common scenarios.

First, the user can authenticate as the service account. For example, they can get short-lived credentials for the service account using the `iam.serviceAccounts.getAccessToken` permission and by calling the `generateAccessToken()` method. Or, they can use the `--impersonate-service-account` flag for gcloud CLI to impersonate the service account. When a user authenticates as a service account, they can issue commands to Google Cloud and can access all resources to which the service account has access.

Second, the user can get artifacts signed by the Google-managed private key of the service account using the `iam.serviceAccounts.signBlob` permission and by calling either the `signBlob()` or `signJwt()` method. The Google-managed private key is always held in escrow and is never directly exposed. `signBlob()` allows signing of arbitrary payloads (such as Cloud Storage-signed URLs), while `signJwt()` only allows signing well-formed JWTs.

Finally, the user can impersonate the service account without ever retrieving a credential for the service account. This is an advanced use case, and is only supported for programmatic access using the `generateAccessToken()` method. In scenarios with at least 3 service accounts, namely *A*, *B*, and *C*: service account *A* can get an access token for service account *C* if service account *A* is granted the `iam.serviceAccounts.implicitDelegation` permission on *B*, and *B* is granted the `iam.serviceAccounts.getAccessToken` permission on *C*.

Generating OpenID Connect (OIDC) ID tokens

Permissions:

- `iam.serviceAccounts.getOpenIdToken`

Roles:

- `roles/iam.serviceAccountOpenIdTokenCreator` (Service Account OpenID Connect Identity Token Creator)

A user (or service) can generate an OpenID Connect (OIDC)-compatible JWT token signed by the Google OIDC Provider (accounts.google.com) that represents the identity of the service account using the `iam.serviceAccounts.getOpenIdToken` permission.

These tokens are not directly accepted by most Google APIs without your organization deploying additional identity federation to grant access to Google.

Generating external private keys

Permissions:

- `iam.serviceAccountKeys.create`

Roles:

- `roles/editor` (Editor)
- `roles/iam.serviceAccountKeyAdmin` (Service Account Key Admin)

A user or service can generate external private key material (RSA) that can be used to authenticate directly to Google as the service account. This key material can then be used with Application Default Credentials (ADC) libraries, or with the `gcloud auth activate-service-account` command. Any person who gains access to the key material will then have full access to all resources to which the service account has access. Such private key material should be treated with the highest concern, and should be considered less secure the longer the material exists. Therefore, rotating private key material is critical to maintaining strong security.

Service account permissions that enable other capabilities

Some permissions for service account credentials enable multiple capabilities. For example, `iam.serviceAccounts.signBlob` and `iam.serviceAccounts.signJwt` also let principals generate access tokens and ID tokens for a service account. Additionally, because `iam.serviceAccounts.signBlob` allows principals to sign any kind of data, it also lets principals sign JWTs.

Before adding any of these permissions to custom roles, ensure that you understand what actions each permission enables.

The following table shows the operations enabled by these permissions:

Permission	Enabled operations
`iam.serviceAccounts.getAccessToken`	Get an access token for the service account
`iam.serviceAccounts.getOpenIdToken`	Get an ID token for the service account
`iam.serviceAccounts.signJwt`	• Sign a JWT • Get an access token or ID token for the service account by using a JWT bearer token.
`iam.serviceAccounts.signBlob`	• Sign any type of blob, including JWTs • Get an access token or ID token for the service account by using a JWT bearer token.

Best practices for granting roles on service accounts

In scenarios where a service account has been granted permissions to perform highly-privileged operations, be cautious when granting the Service Account User role or its included permissions to a user on that service account.

Service accounts represent your service-level security. The security of the service is determined by the people who have IAM roles to manage and use the service accounts, and people who hold service account keys for those service accounts. Best practices to ensure security include the following:

- Use the IAM API to audit the service accounts, the keys, and the allow policies on those service accounts.
- If your service accounts don't need service account keys, disable or delete them.
- If users don't need permission to manage or use service accounts, then remove them from the applicable allow policy.
- Understand how granting certain permissions for service accounts can effectively enable other capabilities.
- Make sure that service accounts have the fewest permissions possible. Use default service accounts with caution, because they are automatically granted the Editor (`roles/editor`) role on the project.

To learn more about best practices, see Best practices for working with service accounts.

Try it for yourself

If you're new to Google Cloud, create an account to evaluate how our products perform in real-world scenarios. New customers also get $300 in free credits to run, test, and deploy workloads.

Get started for free
