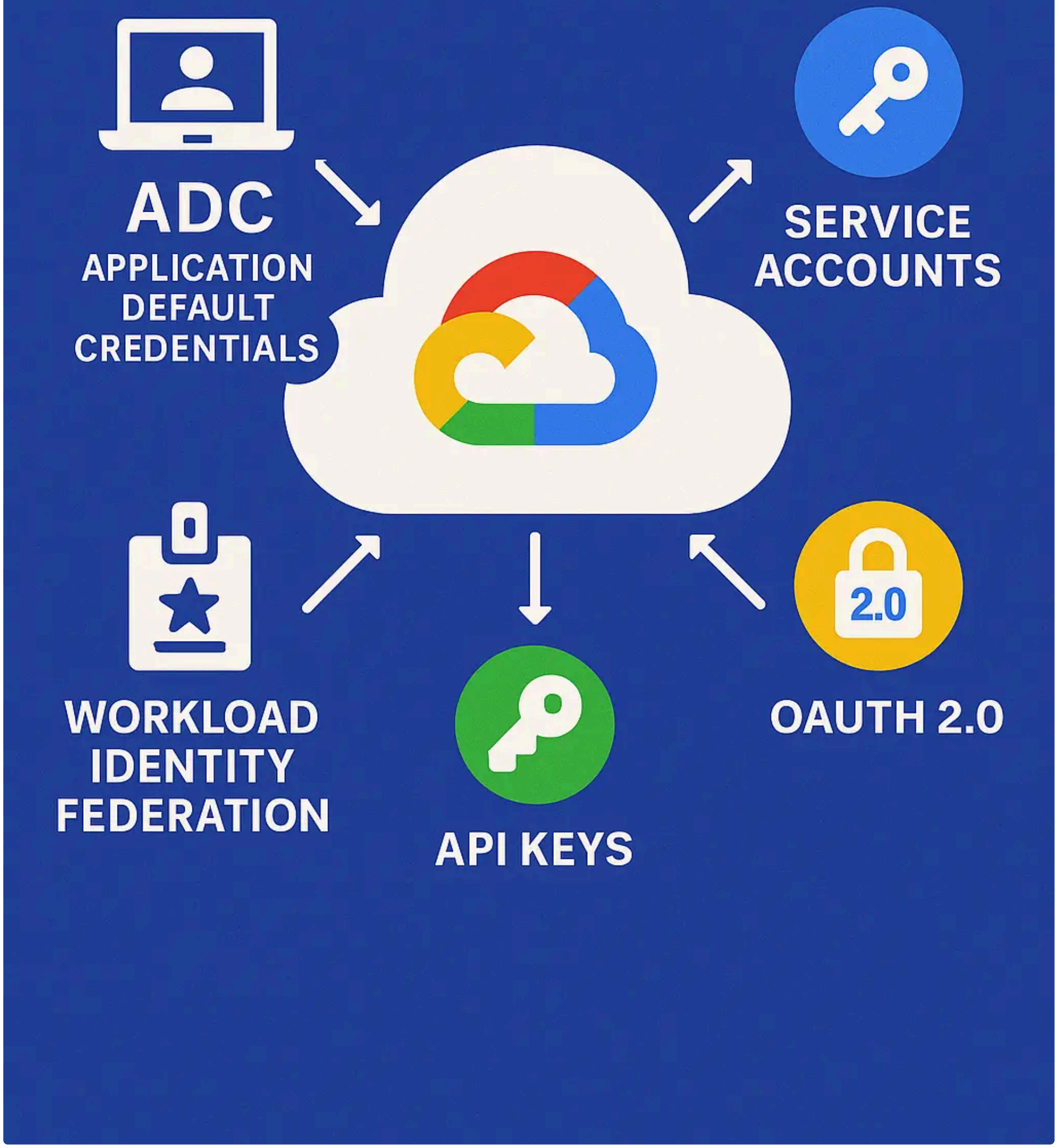
GUIDE

Google Cloud Authentication — The Divio-Style Field Guide for Senior Full-Stack Devs

Huan Li Follow

Sep 12, 2025 · 10 mins read

GOOGLE CLOUD AUTHENTICATION EXPLAINED



A practical, copy-pastable guide that untangles identities, credentials, tokens, ADC, OAuth 2.0, API keys, service accounts, impersonation, and Workload Identity Federation — organised with the Divio documentation system: Tutorials, How-to, Explanation, and Reference.

- Audience: `Senior full-stack dev`
- Goal: `clarity & correct defaults`
- Scope: `Google Cloud + Google APIs`

🗺 The Landscape at a Glance

Use this map to navigate to the right quadrant quickly.

- Identities: Human `user@` vs `service-account@`. Services run as service accounts.
- Credentials → Tokens: Credentials generate short-lived tokens (access/ID). Libraries auto-refresh.
- ADC (Default): Client libraries auto-discover credentials locally and in prod.
- Production on Google Cloud: Attach a service account to Cloud Run/GKE/GCE → metadata server → tokens. No key files.
- Outside Google Cloud: Prefer Workload Identity Federation (AWS/Azure/on-prem OIDC/SAML). Avoid long-lived JSON keys.
- End-user sign-in: OAuth 2.0 / OpenID Connect for users. Your backend still needs its own service identity.
- API keys: Identify the calling project for select public APIs; not IAM authorization.
- Impersonation: Act as a target service account without distributing its key; auditable and least-privilege.

📚 Tutorials (learning-oriented)

Follow these end-to-end lessons to get hands-on quickly. Repeatable, minimal, correct.

T1. Local dev with Application Default Credentials (no key files)

Goal: call a Google Cloud API from your laptop using your user sign-in and let libraries fetch/refresh tokens automatically.

1. Install the Google Cloud CLI (`gcloud`).
2. Run `gcloud auth application-default login` and complete the browser flow. This seeds local Application Default Credentials (ADC) for client libraries.
3. Write code using a Cloud Client Library (e.g., Storage). Don’t pass tokens; the library uses ADC automatically.
4. Optional: inspect a token: `gcloud auth application-default print-access-token`. Tokens are short-lived.

Example (Node.js):

```
// Node.js example
import { Storage } from '@google-cloud/storage'
const storage = new Storage() // uses ADC
const [buckets] = await storage.getBuckets()
console.log(buckets.map(b => b.name))
```

T2. Production on Cloud Run with an attached service account (no keys)

Goal: deploy to Cloud Run so the runtime's metadata server issues tokens for your service account; libraries still use ADC.

1. Create a least-privilege service account (e.g., `svc-myapp@`) and grant only required IAM roles.
2. Deploy and set Service account for the service. Cloud Run attaches that identity to your container.
3. Your code remains unchanged (uses ADC). At runtime, the library calls the metadata server to mint/refresh tokens.
4. For cross-service calls, prefer service account impersonation rather than distributing keys.

Example deploy:

```
gcloud run deploy myapi \
  --image=gcr.io/PROJECT/myapi:latest \
  --service-account=svc-myapp@PROJECT.iam.gserviceaccount.com \
  --region=us-central1
```

T3. Access Google Cloud from AWS without key files (Workload Identity Federation)

Goal: let an AWS workload obtain short-lived Google tokens using its AWS identity — no Google key distribution.

1. Configure a Google Workload Identity Pool & provider (trust AWS IAM). Map AWS claims (like account/role) to Google attributes.
2. Create a Google service account and grant it roles. Permit your pool to impersonate that service account.
3. In the AWS workload, use Google's external credentials JSON (or the auth library's WIF support) to exchange AWS STS creds → Google tokens.
4. Use client libraries with ADC — they read the external credentials file and handle token exchange automatically.

Result: least-privilege, short-lived tokens; no long-lived JSON keys to leak or rotate.

T4. Calling a private Cloud Run service from a browser app

Goal: authenticate end-users in the frontend, then send a request with an ID token that your Cloud Run service verifies.

1. Use an identity provider (e.g., Google, Identity Platform, or another OpenID Connect IdP) to sign in the user.
2. Obtain an ID token with the correct audience (the Cloud Run URL or a custom audience).
3. Send it as `Authorization: Bearer <id_token>`. Your service middleware verifies signature, issuer, audience, and expiry.
4. For backend→backend calls, continue to use access tokens from your service account identity.

🛠 How-to Guides (goal-oriented)

H1. Choose the right method (decision helper)

- Running on Google Cloud? Attach a service account → use ADC (metadata server). Best default.
- Running outside Google Cloud? Prefer Workload Identity Federation. Avoid JSON key files.
- CLI / local dev? `gcloud auth application-default login`.
- End-user sign-in? OAuth 2.0 / OIDC for user tokens; plus a backend service identity for Google APIs.
- Public/untrusted clients needing simple access? Only if the API supports it, use API keys with tight restrictions.

H2. Get an ID token for Cloud Run/Functions

1. From metadata server (in Cloud Run/GCE/GKE): request an ID token for the target audience.
2. From your dev machine: `gcloud auth print-identity-token --audiences=YOUR_AUDIENCE`.
3. Via impersonation: use Service Account Credentials API to mint an ID token for the target SA.

H3. Impersonate a service account (no keys)

Let users or services act as a target service account with auditability.

```
# CLI example
gcloud auth print-access-token \
  --impersonate-service-account=svc-target@PROJECT.iam.gserviceaccount.com
```

```
// ADC for local code (Node.js)
process.env.GOOGLE_IMPERSONATE_SERVICE_ACCOUNT = 'svc-target@PROJECT.iam.gserviceaccount.com'
// Libraries now mint short-lived tokens by impersonating the target SA
```

Grant the caller `roles/iam.serviceAccountTokenCreator` on the target service account; then apply least-privilege roles to the target SA.

H4. Use API keys safely (only where supported)

- Restrict by API, HTTP referrer, IP, and (if supported) Android/iOS app signature.
- Rotate keys and monitor usage; treat keys as secrets even though they don’t grant IAM permissions.
- Prefer OAuth 2.0 or service identities for Google Cloud resources that require IAM authorization.

Explanation (understanding-oriented)

E1. Identities, credentials, and tokens

An identity is a principal: a human user or a service account. A credential is any mechanism that can obtain tokens for that identity: a user login session cached by `gcloud`, a service account key file, a metadata server on a Google runtime, or a federated identity from AWS/Azure/Okta. A token is a short-lived artifact minted from those credentials: usually an OAuth 2.0

access token for Google APIs, or an OpenID Connect ID token for audience-checked calls (e.g., Cloud Run).

> *Note: Tokens are not credentials. Credentials prove identity to Google and let you obtain tokens. Tokens are the minted proofs you actually send with API calls and they expire quickly. Let client libraries handle minting/refresh.*

E2. Application Default Credentials (ADC)

ADC is a discovery strategy built into Google auth libraries. In dev, it checks your local environment (env vars, well-known files created by `gcloud auth application-default login`). In prod on Google Cloud, it consults the metadata server for the attached service account. ADC can also read external credentials files used for federation. This is why your code doesn't change between laptop and production.

E3. OAuth 2.0 vs API keys vs service accounts

- OAuth 2.0 / OpenID Connect: for end-user sign-in and delegated access. Your app requests tokens with explicit scopes and may refresh them. Use for user data and sign-in flows.
- Service accounts: non-human identities for workloads. On Google Cloud, attach them to runtimes and let the metadata server mint tokens. Outside Google Cloud, use Workload Identity Federation or, if you must, a key file (not recommended).
- API keys: identify the calling project for certain public APIs. They do not convey IAM permissions and are not a general auth method for Google Cloud resources.

E4. Workload Identity Federation (WIF)

WIF lets external workloads (AWS/Azure/on-prem) exchange their native identity for short-lived Google tokens, mapped to a Google service account that holds IAM roles. This removes long-lived Google key files from your supply chain and centralises trust in your provider or IdP.

E5. Service account impersonation

Impersonation lets a caller mint short-lived tokens to act as a target service account (with audit trails). Ideal for CI/CD and for letting a narrow identity temporarily gain the target's permissions without sharing keys.

E6. Access tokens vs ID tokens

- Access token: bearer token for Google APIs (authorised by scopes + IAM).
- ID token: proves the caller's identity to a specific audience (e.g., your Cloud Run URL). Your service verifies audience/issuer/expiry before trusting it.
- Rule of thumb: backend → Google API ⇒ access token; browser → your backend ⇒ ID token.

Reference (information-oriented)

R1. ADC lookup (high level)

1. Env var: `GOOGLE_APPLICATION_CREDENTIALS` pointing to a credential file.
2. Well-known user location: credentials seeded by `gcloud auth application-default login`.
3. Metadata server: when running on Cloud Run/GKE/GCE with an attached service account.
4. External credentials: federation config files for WIF.

R2. Token types (selected)

- Access token — OAuth 2.0 token to call Google APIs.
- ID token — OpenID Connect JWT for audience-checked requests (e.g., Cloud Run).
- Self-signed JWT — service account assertion used by libraries to obtain access tokens (or in some cases to call endpoints directly).

R3. Handy CLI

```
# User ADC login (local dev)
gcloud auth application-default login

# Print short-lived access token for current ADC
gcloud auth application-default print-access-token

# ID token for a specific audience (Cloud Run URL or custom)
gcloud auth print-identity-token --audiences=https://<service-url>

# Use impersonation in CLI
gcloud auth print-access-token \
  --impersonate-service-account=svc@PROJECT.iam.gserviceaccount.com
```

R4. Security recommendations

- Prefer attached identities (metadata server) on Google Cloud; avoid distributing JSON key files.
- Use Workload Identity Federation for external workloads.
- Use impersonation for CI/CD and human break-glass flows; audit who minted tokens.
- Give the target service account least-privilege roles; grant callers only TokenCreator for impersonation.
- If you must use API keys, restrict them aggressively and monitor usage.

Common confusions & straight answers

“Tokens are not credentials.” Huh?

Credentials are how you authenticate (user login, key, metadata server, federation). Tokens are what you use after you authenticate. Tokens expire; credentials persist (or are renewable) and can mint new tokens.

ADC vs `gcloud auth login` vs `gcloud auth application-default login`

`gcloud auth login` authenticates the CLI itself. `gcloud auth application-default login` seeds ADC for your code and client libraries. In production, you won't run either — runtimes use the metadata server via ADC.

API keys vs OAuth 2.0 / service accounts

API keys identify the calling project for certain public APIs; they don't carry IAM permissions. For Google Cloud resources that enforce IAM, use user OAuth or (more commonly) service accounts with ADC.

When do I need an ID token?

When your service wants to verify the caller's identity and intended audience (e.g., a SPA calling Cloud Run). For backend→Google API calls, you normally use an access token instead.

Keys vs Federation vs Impersonation

Long-lived key files are risky and hard to rotate. Prefer federation outside Google Cloud and impersonation between identities. Both yield short-lived, auditable tokens with least privilege.

Libraries vs manual REST calls

Client libraries handle ADC and token refresh for you. If you use raw REST, you'll obtain/refresh tokens yourself — fine for tooling, but error-prone for apps.

Structured with the Divio system: Tutorials · How-to · Explanation · Reference.

google-cloud authentication adc oauth2 service-accounts impersonation workload-identity-federation security

Written by Huan Li Follow

Founding Human & Architect | Author @ Wechaty, Silicon Adventist, Serial Entrepreneur, Angel Investor, Burner🔥

Loading comments…