Vertex AI Adoption Guide

For Indonesian Enterprises

September 2026 Edition

Change History

This table records substantive changes to this guide. Refer to the version relevant to your deployment.

Version	Release date	Summary of changes	Sections affected
1.0	June 2026	Initial release.	All
1.2	September 2026	Main change: authentication from outside Google Cloud. New Section 5.4 covers Workload Identity Federation for applications running on other clouds, in CI pipelines, or on-premises, and Workforce Identity Federation for people who sign in with a non-Google identity provider. Section 5.3 gains a signpost to it, Section 1 now states that the guide does not assume Google Workspace, and both products are written out in full because the abbreviation WIF was ambiguous between them. Supporting corrections: Application Default Credentials is described as the credential lookup order rather than an alternative to federation. The Cloud Run description now says attached service account. Added a troubleshooting entry for the missing quota project error. Updated code examples to gemini-3.8-flash and gemini-3.1-pro-preview, and set the example region to global because no Gemini model is served from Jakarta. Clarified the Jakarta data residency wording in Sections 4, 5.3 and 6.5, and rewrote the region selection decision so it leads with that fact. Replaced the opening questions in Section 4 with statements. Corrected the LiteLLM reasoning_effort value to disable. Removed the remaining em dashes. Updated the Agent Runtime deployment example in Section 7.3 for the Agent Platform Python SDK 2.0.1 restructure, which moved the agent surface to google-cloud-agentplatform and removed the global initializer. Added an introductory pricing expiry entry to the Section 5.2 billing pitfalls. Added Section 6.8 on agent governance,	1, 2, 4, 5.1, 5.2, 5.3, 5.4, 6.1, 6.5, 6.6, 6.7, 6.8, 7.2, 7.3, 8.1, 8.2, 9, 10, 11

Version	Release date	Summary of changes	Sections affected
		covering Agent Identity, Agent Registry, Agent Gateway, IAM Unified Access Policies and semantic governance policies. Added Flexible Savings Plans to Section 5.2, Model Armor at the gateway to Section 6.6, Provisioned Throughput to Section 6.7, Memory Bank to Section 7.3 and Agent Observability to Section 8.1. Confirmed from Google's identity federation support page that Vertex AI is generally available for identity federation, and noted that Agent Registry is in Preview for it. Added a managed workload identities note to Section 5.4, with a link to its documentation.	

Table of Contents

Change History 1
Table of Contents 3
1. Introduction 4
2. Key Concepts and Terminology 6
3. Before You Begin 8
4. Planning Your Deployment 9
5. Setting Up Vertex AI 12
    5.1 Organization and IAM 12
    5.2 Billing and Governance 18
    5.3 Quick Start 23
    5.4 Authenticating from Outside Google Cloud 33
6. Securing Your Environment 39
    6.1 IAM Best Practices 40
    6.2 Service Accounts 41
    6.3 Centralized Logging 41
    6.4 VPC Service Controls 42
    6.5 Encryption and Data Residency 43
    6.6 Model Armor 44
    6.7 Responsible Usage and Cost Control 46
    6.8 Agent Governance 49
7. Building Applications 51
    7.1 Prompting Guide 51
    7.2 LiteLLM Integration (Optional) 57
    7.3 Building a Chatbot with ADK 63
8. Operating in Production 67
    8.1 Monitoring 67
    8.2 Troubleshooting 68
9. Production Deployment Checklist 72
10. Frequently Asked Questions (FAQ) 74
11. Glossary 77
12. Legal Disclaimer and Limitation of Liability 79

1. Introduction

This guide provides a practical introduction to adopting Google's generative AI capabilities on Google Cloud. It is intended for organizations evaluating AI initiatives, establishing cloud foundations, or building AI-powered applications in an enterprise environment.

The guide is written for both business and technical stakeholders. Business-focused sections explain governance, security, cost management, and architecture decisions. Technical sections provide implementation guidance, Google Cloud Console instructions, command-line examples, and application integration patterns.

A terminology note is important before proceeding. Google recently renamed Vertex AI Platform to Gemini Enterprise Agent Platform in the Google Cloud Console. As a result, users may encounter both names depending on the documentation, training materials, SDKs, APIs, or console screens they are using.

Throughout this guide, the term **Vertex AI** is used for technical instructions unless explicitly stated otherwise. This approach is intentional because many APIs, IAM roles, SDK references, and existing documentation continue to use Vertex AI terminology. When navigating the Google Cloud Console, however, readers should expect to see the service presented as **Gemini Enterprise Agent Platform**.

This guide focuses on enterprise adoption of Vertex AI, including organizational setup, identity and access management, billing governance, security controls, model access, application integration, prompting practices, and production readiness. Optional technologies and integrations are clearly identified so that teams can distinguish between core platform requirements and advanced implementation choices.

This guide does not assume you use Google Workspace. A Google Cloud organization comes from one of two accounts on your domain: Cloud Identity or Google Workspace. Cloud Identity has a free edition. It is built for organizations that do not want Gmail, Calendar and the rest of Workspace, so no Workspace licenses are needed.

Your own identity provider stays where it is. Okta or Microsoft Entra ID connects to the organization through Workforce Identity Federation, and no Google account is created for your employees. Applications running on Amazon Web Services, Azure, GitHub Actions or your own servers call Gemini through Workload Identity Federation, with no service account key file. Section 5.1 covers the administrator setup, and Section 5.4 covers the developer side.

Where examples include pricing, quotas, model availability, or retention periods, they should be treated as illustrative only. Always refer to the official Google Cloud documentation for the latest product information, service limits, and pricing details.

2. Key Concepts and Terminology

The following terms appear frequently throughout this guide. Understanding the distinction between them early will prevent confusion, as they refer to related but different things.

Term	What it means in this guide
Vertex AI	Google Cloud's managed platform for AI and machine learning. This is the primary platform this guide covers. On May 21, 2026, the name disappeared from the Console; Google now describes it as part of **Gemini Enterprise Agent Platform**. The API, IAM roles, and most SDK calling patterns are unaffected. Separately, some older SDK modules are being removed on June 24, 2026. See Section 10 FAQ.
Gemini in Vertex AI	The family of Gemini language models accessed through Vertex AI. This is the recommended path for enterprise use.
Gemini API	A direct API for accessing Gemini models, primarily used for prototyping. It is simpler but offers fewer enterprise controls than Vertex AI.
Google Cloud Organization	The root resource that represents your company in Google Cloud. Required for enterprise-level governance.
Project	A container within the Organization where resources, APIs, and billing are managed. Most teams use separate projects for sandbox, staging, and production.
IAM	Identity and Access Management. Controls who can access which resources and what actions they can perform.
Service Account	A non-human identity used by applications and automated systems to authenticate with Google Cloud.
Token	The unit used to measure text input and output when calling Gemini models. Pricing is calculated per million tokens.
LiteLLM	A third-party open-source library that provides a unified interface for calling multiple AI providers including Vertex AI. Not part of Google Cloud.
Application Default Credentials (ADC)	The lookup order Google client libraries follow to find a credential. ADC is not a credential itself. It reads, in order, the GOOGLE_APPLICATION_CREDENTIALS environment variable, a local sign-in file, then the attached service account.

Workforce Identity Federation	Lets a person sign in to Google Cloud with an existing corporate identity such as Okta or Microsoft Entra ID, without a separate Google account. This guide always writes the name in full.
Workload Identity Federation	Lets an application running outside Google Cloud, for example on Amazon Web Services or in GitHub Actions, obtain short-lived Google credentials without a service account key file. A separate product from Workforce Identity Federation.

3. Before You Begin

Before proceeding with any setup, ensure the following requirements are in place. These are not optional steps that can be completed later. Skipping them is one of the most common causes of access errors, billing surprises, and security gaps in early deployments.

- A Google Cloud Organization resource exists under your company domain.
- Users can authenticate through Google Workspace or an approved third-party identity provider (IdP).
- At least one Cloud Billing account is active and available for project association.
- The team has identified who will hold administrator roles before any project is created.
- Billing alerts are configured before running production or large-scale workloads (see Section 5.2).

4. Planning Your Deployment

A few decisions made before setup will prevent significant rework later. Work through each item below with your team before proceeding to Section 5. The decisions below are ordered from simplest (a single configuration choice) to most complex (decisions that depend on regulatory classification and organizational context).

Region selection

For Indonesia, Vertex AI regional coverage depends on what you need to run. Data storage can stay in Jakarta (`asia-southeast2`), while Gemini model processing must use a different endpoint. Stored data stays at rest in the region you choose. The region also supports custom model training, custom model online and batch inference, Vector Search, Pipelines and Model Registry. Google's endpoint list does not include `asia-southeast2` for any Gemini model, so Gemini requests go to another region. Singapore (`asia-southeast1`) is the nearest, but at the September 2026 review it carried only `gemini-3.5-flash` and `gemini-2.5-flash`. Newer models such as `gemini-3.8-flash` were on the global, `us` and `eu` endpoints. Check the Vertex AI locations page and Google's model location commitments for per-model availability before you decide.

If processing must stay in Indonesia, you can host an open model yourself on a custom endpoint in `asia-southeast2`. This would require your team to manage the model, capacity, scaling, and maintenance, making it an option where regulatory requirements require processing to remain in Indonesia.

Identity strategy

User sign-in depends on your organization's identity provider. For organizations using Google Workspace, Google Cloud Identity is the simplest option. If you use another identity provider, such as Okta or Microsoft Entra ID, you will need Workforce Identity Federation. This choice affects how access is set up and managed throughout the guide. See Section 5.1.

Access model

Access control depends on how precisely you need to scope permissions. For most teams, Google Groups assigned to IAM roles are simpler to set up and maintain. If you need finer control, attribute-based rules are available. This decision builds directly on the identity strategy above. See Section 5.1.

LiteLLM

LiteLLM is useful when you work with more than one AI provider, or when you need per-team budget limits alongside Vertex AI. It provides one consistent interface across providers. It is an optional third-party tool and not part of Google Cloud, and is independent of the decisions above. See Section 7.2.

Encryption

Google Cloud encrypts all stored data by default. For most organizations, this is enough. If your organization needs to manage its own encryption keys (for example, to meet specific compliance requirements), you will need Customer-Managed Encryption Keys (CMEK). This choice must be made before creating any Vertex AI resources, as it cannot be changed later. See Section 6.5.

AI safety controls

If your organization works in a regulated sector such as financial services, healthcare, or government, decide early whether Model Armor is needed. It provides automatic data redaction, protection against prompt attacks, and audit logs. This decision depends on understanding your regulatory exposure first. See Section 6.6.

Data governance and regulatory obligations

This is the most context-dependent decision in this list, and varies significantly by organization type. Indonesia has data localization requirements under Government Regulation No. 71 of 2019 (GR 71/2019), but who is affected and how strictly depends on your organization's classification. Sector-specific regulators such as OJK, Bank Indonesia, and the health ministry impose their own rules on top of GR 71/2019, and private companies face different obligations than government entities. The outcome of this decision often determines how strict the encryption and AI safety choices above need to be, so it is worth revisiting those once this classification is settled.

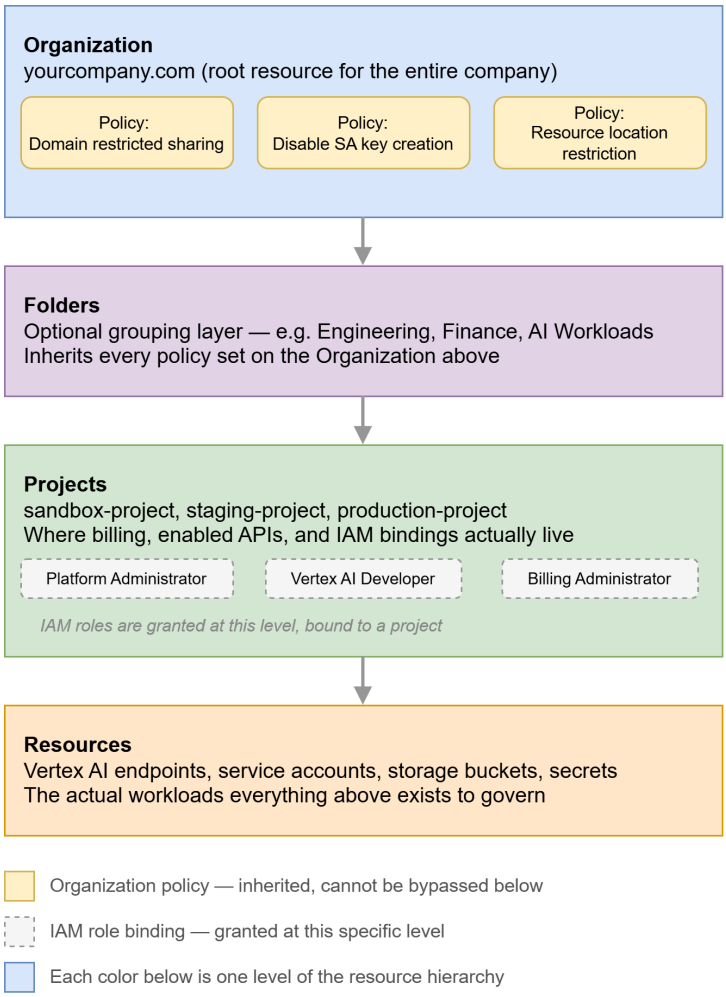
The table below summarizes the decisions and where to find implementation guidance, in the same simple-to-complex order.

Decision	Options	Guide section
Primary region	`asia-southeast2` (Jakarta) for storage, another region for Gemini	5.1
Identity provider	Google Workspace or Workforce Identity Federation	5.1
Access model	Group-based IAM / ABAC	5.1
Multi-provider interface	Direct SDK / LiteLLM	7.2
Encryption	Default (Google-managed) / CMEK	6.4
AI safety layer	Model Armor optional / required	6.6
Data governance	Depends on regulatory classification	6.4, 6.5

Authentication for applications outside Google Cloud	Workload Identity Federation or service account key file	5.4

5. Setting Up Vertex AI

5.1 Organization and IAM



Identity and Access Management determines who can access resources and what actions they are permitted to perform. Most security incidents in cloud environments trace back to excessive permissions rather than technical vulnerabilities. A well-designed IAM model established at the start prevents both accidental and intentional misuse and is easier to maintain than one added back after workloads are already running.

Google Cloud organizes resources in a hierarchy: Organization → Folders → Projects → Resources. Policies applied at a higher level are automatically inherited by everything below, and they cannot be bypassed by lower-level settings. This means that governance rules applied at the Organization level apply to every team and project without any additional effort.

Three organization-level policies should be enforced before any project is created. The first restricts which identities can be added to IAM policies, preventing external users from being accidentally granted access. The second restricts where resources can be created, ensuring

data remains in approved regions. The third disables automatic service account key creation, which is one of the most common sources of credential leaks in cloud environments.

Console

1. Go to console.cloud.google.com and sign in with your organization administrator account.
2. In the navigation menu, go to **IAM & Admin** > **Organization Policies**.
3. Confirm the scope at the top shows your Organization node, not a specific project.
4. Search for **Domain restricted sharing**, click it, select **Manage policy**, and restrict allowed identities to your organization domain.
5. Search for **Resource location restriction**, click it, and set allowed locations to `asia-southeast2` (Jakarta) (add others if needed).
6. Search for **Disable service account key creation**, click it, and set it to **Enforced**.

CLI

```
# Restrict which identities can be added to IAM policies
gcloud org-policies set-policy --organization=YOUR_ORG_ID - <<EOF
name: organizations/YOUR_ORG_ID/policies/iam.allowedPolicyMemberDomains
spec:
 rules:
 - values:
   allowedValues:
   - "is:principalSet://iam.googleapis.com/organizations/YOUR_ORG_ID"
EOF

# Restrict resource creation to approved regions
gcloud org-policies set-policy --organization=YOUR_ORG_ID - <<EOF
name: organizations/YOUR_ORG_ID/policies/gcp.resourceLocations
spec:
 rules:
 - values:
   allowedValues:
   - "in:asia-southeast2-locations"
EOF

# Disable service account key creation
gcloud org-policies set-policy --organization=YOUR_ORG_ID - <<EOF
name: organizations/YOUR_ORG_ID/policies/iam.disableServiceAccountKeyCreation
spec:
 rules:
 - enforce: true
EOF
```

Role assignment

IAM roles should always be assigned to Google Groups, not to individual user accounts. When a team member joins or leaves, you update the group membership not every IAM policy across every project individually. This is the standard approach in enterprise environments and reduces the risk of orphaned permissions.

The table below describes the three principal roles in a Vertex AI deployment.

Role	IAM Role ID	Responsibilities
Platform Administrator	`roles/resourcemanager.organizationAdmin`	Configures org policies, grants roles, manages Terraform
Vertex AI Developer	`roles/aiplatform.user`	Calls Vertex AI APIs, deploys models, accesses Vertex AI Studio
Billing Administrator	`roles/billing.admin`	Manages billing accounts, sets budgets, monitors costs

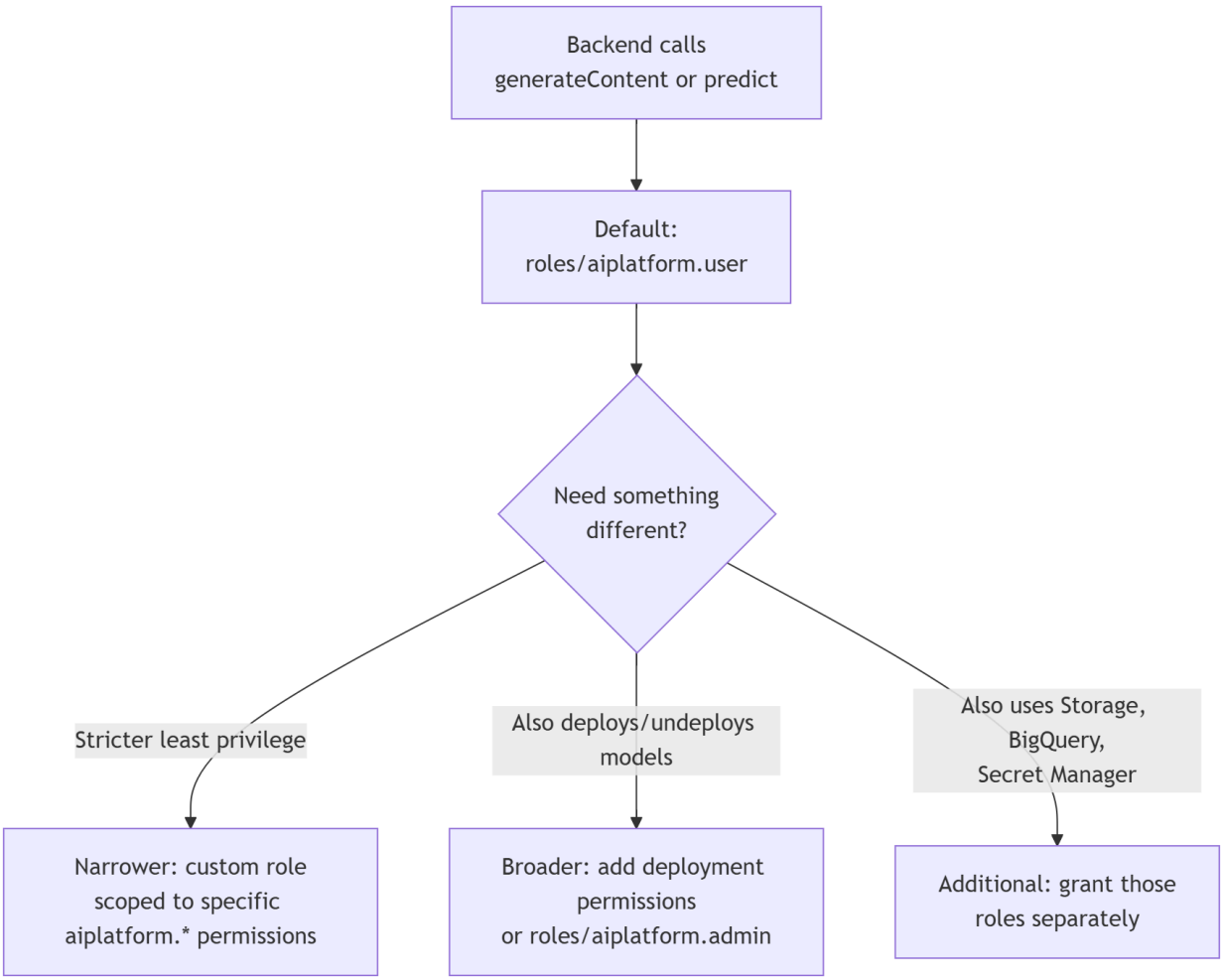
Console

1. Go to **IAM & Admin** > **IAM**.
2. Confirm the correct project is selected at the top.
3. Click **Grant Access**.
4. In **New principals**, enter the Google Group email address (e.g., `vertexai-developers@yourcompany.com`).
5. In **Assign roles**, search for and select the appropriate role.
6. Click **Save**.

CLI

```
# Assign the Vertex AI user role to a developer group
gcloud projects add-iam-policy-binding YOUR_PROJECT_ID \
 --member="group:vertexai-developers@yourcompany.com" \
 --role="roles/aiplatform.user"
```

What does a backend application actually need?



The three roles above cover the people setting up and governing the environment. A different question is what a backend service, the application or service account that actually calls `generateContent` or `predict` at runtime, needs to function.

For nearly all backend integrations, **`roles/aiplatform.user` is the minimum role required.** It covers content generation, prediction calls, and most day-to-day inference operations a production backend performs. This is the same role granted to the Vertex AI Developer group above, but bound to a service account instead of a human group:

```
gcloud projects add-iam-policy-binding YOUR_PROJECT_ID \

--member="serviceAccount:your-backend-app@YOUR_PROJECT_ID.iam.gserviceaccount.com" \
 --role="roles/aiplatform.user"
```

A few situations call for something narrower or broader than this default:

- **Narrower, custom role:** if your security policy requires least privilege beyond what `aiplatform.user` grants (it includes permissions for tuning jobs, evaluation, and other operations a pure inference backend may never call), define a custom IAM role scoped only to the specific `aiplatform.*` permissions your backend uses, such as `aiplatform.endpoints.predict`.
- **Broader, if deploying models:** a backend that also deploys or undeploys models to an endpoint at runtime (rather than just calling an existing endpoint) needs `roles/aiplatform.user` plus deployment-specific permissions, or `roles/aiplatform.admin` if it manages the full model lifecycle. Most production backends should not need this. Model deployment is typically a platform-team operation, not something the application does on every request.
- **Additional roles for adjacent services:** if the backend also reads from Cloud Storage, queries BigQuery, or accesses Secret Manager, those each require their own role (`roles/storage.objectViewer`, `roles/bigquery.dataViewer`, `roles/secretmanager.secretAccessor`) granted separately. The role `aiplatform.user` does not include access to other Google Cloud services.

In every case, the service account holding this role should be dedicated to the backend application, not shared with other workloads, so that audit logs clearly attribute every Vertex AI call to the application that made it.

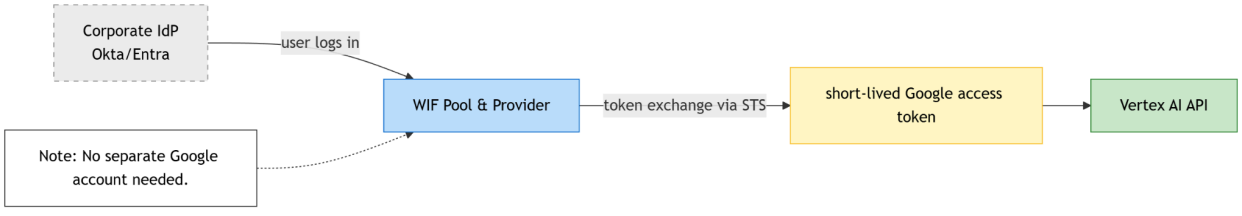
Console

1. Go to **IAM & Admin** > **IAM**.
2. Confirm the correct project is selected at the top.
3. Click **Grant Access**.
4. In **New principals**, enter the Google Group email address (e.g., `vertexai-developers@yourcompany.com`).
5. In **Assign roles**, search for and select the appropriate role.
6. Click **Save**.

CLI

```
# Assign the Vertex AI user role to a developer group
gcloud projects add-iam-policy-binding YOUR_PROJECT_ID \
 --member="group:vertexai-developers@yourcompany.com" \
 --role="roles/aiplatform.user"
```

Workforce Identity Federation



If your organization uses a non-Google identity provider such as Okta, Microsoft Entra ID, or Ping Identity, Workforce Identity Federation allows your employees to authenticate to Google Cloud using their existing corporate credentials. Users do not need a separate Google account. Access is managed through a token exchange: the user logs in with their corporate IdP, which issues a token that Google Cloud's Security Token Service validates and converts to a short-lived Google access token.

Workforce Identity Federation is the required path if your organization uses Microsoft 365 connectors. For all other non-Google IdP scenarios, it is the recommended approach over creating duplicate Google identities.

One important note from teams that have gone through this migration: if you later need to add OIDC support to an existing SAML configuration (for example, to support mobile app access), always add a new Provider under the existing Identity Pool never create a new Pool. Creating a new Pool breaks all existing IAM bindings and license assignments.

Console

1. Go to **IAM & Admin** > **Workforce Identity Federation**.
2. Click **Create Pool**, give it a name, and click **Continue**.
3. Click **Add a Provider**, and select **OpenID Connect** or **SAML** depending on your IdP.
4. Enter your IdP's Issuer URL and Client ID.
5. Under **Attribute Mapping**, set `google.subject` to `assertion.email` and `google.groups` to `assertion.groups`.
6. Click **Save** and copy the Pool and Provider resource names for use in IAM bindings.

CLI

```
# Create a Workforce Identity Pool
gcloud iam workforce-pools create corp-identity-pool \
 --organization=YOUR_ORG_ID \
 --location=global \
 --display-name="Corporate Identity Pool"

# Add an OIDC provider
gcloud iam workforce-pools providers create-oidc corp-oidc-provider \
 --workforce-pool=corp-identity-pool \
 --location=global \
 --display-name="Corporate IdP" \
 --issuer-uri="https://YOUR_IDP_DOMAIN" \
 --client-id="YOUR_CLIENT_ID" \

--attribute-mapping="google.subject=assertion.email,google.groups=assertion.gr
oups,google.display_name=assertion.name"
```

The table below summarizes which authentication path applies to which scenario.

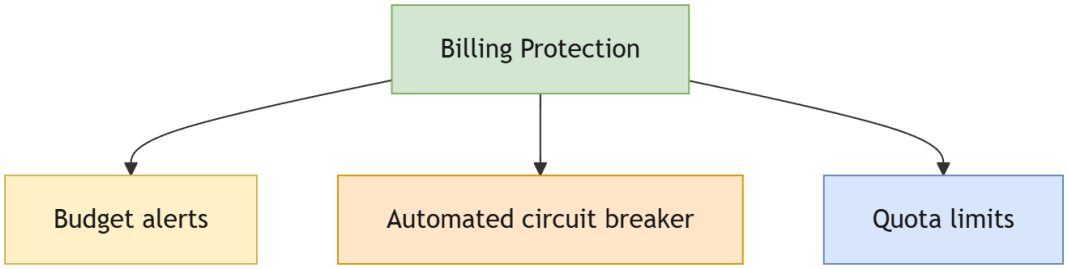
Scenario	Required authentication	Supported protocols
Google Workspace users	Google Cloud Identity	Google Native
Microsoft 365 connectors	Workforce Identity Federation with Microsoft Entra ID	OIDC (mobile) or SAML (web)
Okta, Ping, or other IdP	Workforce Identity Federation	OIDC (mobile) or SAML (web)
Mobile application access	Depends on IdP	OIDC required

5.2 Billing and Governance

Overview

Cost management in cloud environments requires active configuration, not passive monitoring. Without budget controls in place before workloads run, unexpected usage whether from a misconfigured loop, a forgotten resource, or a compromised credential can produce significant charges before anyone notices.

The approach described here has three layers. Budget alerts notify the responsible team when spending reaches defined thresholds. An automated circuit breaker stops billing entirely if the budget is severely exceeded. Quota limits restrict the volume of API calls a project can make, limiting the potential damage from runaway processes.



Gemini models charge based on the number of tokens processed, both input and output. Newer reasoning models also generate internal reasoning tokens that are billed at the same rate as output tokens but are not visible in the response text. A request that appears to return a short answer can consume much more tokens than expected if the model uses extensive internal reasoning. The examples in this section are illustrative only. Always verify current rates on the official Vertex AI pricing page before planning a budget.

Understanding tokens and cost

A token is approximately four characters of text, or about 60 to 80 English words per 100 tokens, based on Google's official documentation. Non-English languages, including Bahasa Indonesia, may produce different token counts for the same meaning. Use the count_tokens API on representative Indonesian prompts to measure actual token usage for your specific content before estimating costs.

To estimate cost before committing to a model or architecture, count tokens on a representative prompt first.

```
from google import genai
from google.genai.types import HttpOptions

client = genai.Client(http_options=HttpOptions(api_version="v1"))

token_count = client.models.count_tokens(
   model="gemini-3.8-flash",
   contents="Tuliskan ringkasan laporan keuangan kuartal ini dalam tiga
paragraf.",
)

print(token_count.total_tokens)
```

A worked example of cost estimation: if an application sends 10,000 requests per day, each with 50 input tokens and 200 output tokens, the daily token volume is 500,000 input and 2,000,000 output tokens. Use the official pricing page to calculate the cost at current rates for your chosen model. Pro-tier models cost significantly more per token than Flash-tier models, so the choice of model has a large impact on the final figure. Starting with a Flash-tier model for initial development is a common practice and helps establish a cost baseline before optimizing.

Operational monitoring requirement

Always monitor the usage_metadata field in every API response in production. This is a required operational signal for cost and anomaly detection. It reports input, output, and thinking token counts separately, allowing detection of unexpected token spikes or runaway generations early.

```
response = client.models.generate_content(
   model="gemini-3.8-flash",
   contents=your_prompt,
   config=types.GenerateContentConfig(max_output_tokens=2048),
)

print(f"Input tokens:    {response.usage_metadata.prompt_token_count}")
```

```
print(f"Output tokens:   {response.usage_metadata.candidates_token_count}")
print(f"Thinking tokens: {response.usage_metadata.thoughts_token_count}")
```

Common billing pitfalls

Forgotten dedicated endpoints

Vertex AI dedicated prediction endpoints bill by the hour regardless of whether they receive traffic. Teams deploy an endpoint for testing, stop using it, and find ongoing charges weeks later. Audit running endpoints regularly and set autoscaling minimums to zero on non-production endpoints.

Uncontrolled reasoning tokens

Newer Pro-tier models generate internal reasoning tokens before producing a visible answer. These tokens are billed at the same rate as output tokens. A request that appears to return a short answer can consume far more tokens than expected. Set thinking_level to MINIMAL or MEDIUM for tasks that do not require deep reasoning, and always set max_output_tokens.

Looping and repetitive output

Large language models can sometimes repeat phrases or reasoning steps instead of producing new output, especially in long-form generation or complex prompts. This behavior can consume significant token budget without useful results.

This is a known behavior in large language models during long-form generation or complex multi-step prompts, where the model may repeat patterns and consume tokens without producing additional useful content.

The primary mitigation is always setting max_output_tokens to a value that matches expected output length. When the model loops, it stops at this limit instead of running indefinitely. Additionally, lower temperature (e.g. 0.2) and structured prompts reduce likelihood.

```
config=types.GenerateContentConfig(
   max_output_tokens=1024,
   temperature=0.2,
)
```

If repetitive output is observed in production, check response.candidates[0].finish_reason. A value of MAX_TOKENS indicates the model hit the limit mid-generation.

Missing max_output_tokens

On reasoning models, omitting max_output_tokens increases risk of runaway generation loops. Always set this parameter.

Budget alert set only at 100%

By the time a 100% threshold alert fires, the budget is already consumed. Set alerts at 50%, 90%, and 100% to allow time to react.

Context length pricing thresholds

Pro-tier models have pricing cliffs at large context sizes (often around 200,000 tokens), where input cost increases significantly. Applications that include large retrieved documents per request can cross this threshold silently.

Introductory pricing expiry

At the September 2026 review, Gemini 3.8 Flash, Gemini 3.7 Flash and Gemini 3.6 Flash carried introductory pricing of $0.75 per 1M input tokens and $3.75 per 1M output tokens, with standard pricing of $1.50 and $7.50 applying from 1 January 2027. These figures are illustrative and will change. The structural point is what matters: a model on introductory pricing can double in cost on a published date, with no change to your code or your traffic. Budget against the standard rate whenever the deployment will outlive the introductory period, and check the pricing page for the model you actually deploy.

Tips to avoid a billing spike

The mitigations above, pulled together in one place:

- Audit running dedicated endpoints regularly and set autoscaling minimums to zero on non-production endpoints.
- Set `thinking_level` to MINIMAL or MEDIUM for tasks that do not need deep reasoning.
- Always set `max_output_tokens` on every request, sized to your expected output length.
- Lower temperature (for example, 0.2) and use structured prompts to reduce the chance of looping output.
- Check `response.candidates[0].finish_reason` in production logs. A value of `MAX_TOKENS` on a short expected response is a sign of looping output.
- Set budget alerts at 50%, 90%, and 100%, not only at 100%.
- Watch for context sizes that approach pricing thresholds, especially on Pro-tier models with large retrieved documents.

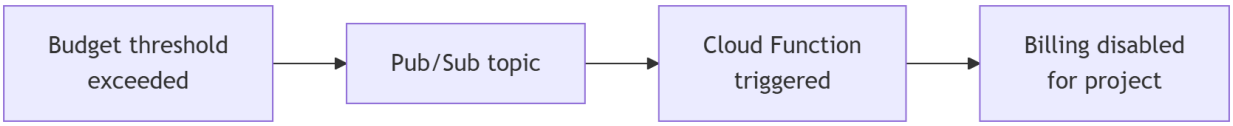
Billing alerts

Console
In the navigation menu, click Billing. Select your billing account, then go to Budgets & alerts. Create a budget and configure alerts at 50%, 90%, and 100%. Enable email or Pub/Sub notifications.

CLI

```
gcloud billing budgets create \
--billing-account=YOUR_BILLING_ACCOUNT_ID \
--display-name="Vertex AI Workload Budget" \
--budget-amount=500USD \
--threshold-rule=percent=50,basis=CURRENT_SPEND \
--threshold-rule=percent=90,basis=CURRENT_SPEND \
--threshold-rule=percent=100,basis=CURRENT_SPEND \
--all-updates-rule-pubsub-topic=projects/YOUR_PROJECT_ID/topics/billing-alerts
```

Automated circuit breaker



A circuit breaker automatically disables billing for a project when spending exceeds the budget limit. This is typically used in sandbox and development environments where cost control is more important than availability. In production environments it should be used carefully and always paired with a recovery plan, since disabling billing stops all running resources.

The mechanism connects a Pub/Sub topic to a Cloud Function that disables billing when threshold conditions are met.

```
import base64
import json
import os
from googleapiclient import discovery

def disable_billing(event, context):
   pubsub_data = base64.b64decode(event["data"]).decode("utf-8")
   budget_notification = json.loads(pubsub_data)

   cost_amount = budget_notification.get("costAmount", 0)
   budget_amount = budget_notification.get("budgetAmount", 0)

   if cost_amount >= budget_amount:
       project_id = os.environ["GCP_PROJECT"]
       service = discovery.build("cloudbilling", "v1")
       billing_name = f"projects/{project_id}/billingInfo"
       body = {"billingAccountName": ""}
       service.projects().updateBillingInfo(name=billing_name,
body=body).execute()
```

Quota limits

Quotas restrict how many API calls a project can make per minute or per day. Setting quotas below the default ceiling reduces impact of runaway processes or compromised credentials.

Console
Go to APIs & Services > Quotas & System Limits, filter Vertex AI API, and adjust generate content request limits.

CLI

```
gcloud services quota list \
--service=aiplatform.googleapis.com \
--project=YOUR_PROJECT_ID

gcloud services quota list \
--service=aiplatform.googleapis.com \
--project=YOUR_PROJECT_ID \
--filter="metric:generate_content_requests"
```

Committed use discounts

Flexible Savings Plans are a spend-based committed use discount. You commit to a minimum monthly spend for one or three years. At the September 2026 review the discount was 10 percent for a one year plan and 20 percent for three years. Standard committed use discounts measure usage in hourly windows. These use a monthly window instead, which suits workloads whose volume is uneven. Check the eligible SKU list first, because some SKUs draw down the commitment without earning an extra discount.

5.3 Quick Start

This section covers the minimum steps required to make a working Vertex AI API call. It assumes the Organization and IAM steps in Section 5.1 have been completed and a billing account is available.

Vertex AI uses your Google Cloud identity for authentication. No API key is stored in code. This is the recommended pattern for all team and production workloads because it integrates with IAM, audit logging, and the security controls described in Section 6.

Create a project

It is a recommended practice to separate AI workloads from other production resources using a dedicated project. This makes cost attribution, access control, and audit logging easier to manage.

Console

1. Go to console.cloud.google.com.

2. Click the project selector at the top of the page and click **New Project**.
3. Enter a project name, select your Organization, and click **Create**.
4. Select the new project from the project selector to make it active.

CLI

```
gcloud projects create YOUR_PROJECT_ID \
 --name="Vertex AI Workloads" \
 --organization=YOUR_ORG_ID

gcloud config set project YOUR_PROJECT_ID
```

If the project ID is already taken globally, try a more specific identifier. Project IDs are unique across all of Google Cloud, not just your organization.

Enable the Vertex AI API

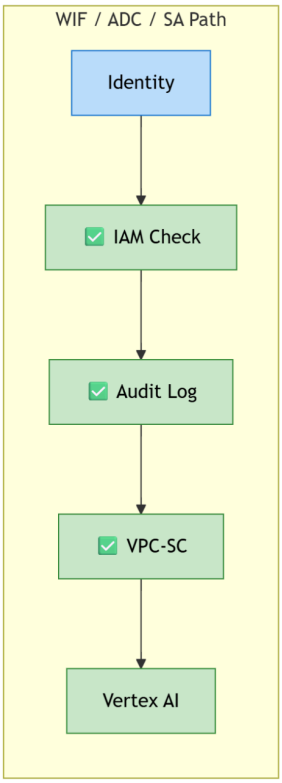
Console

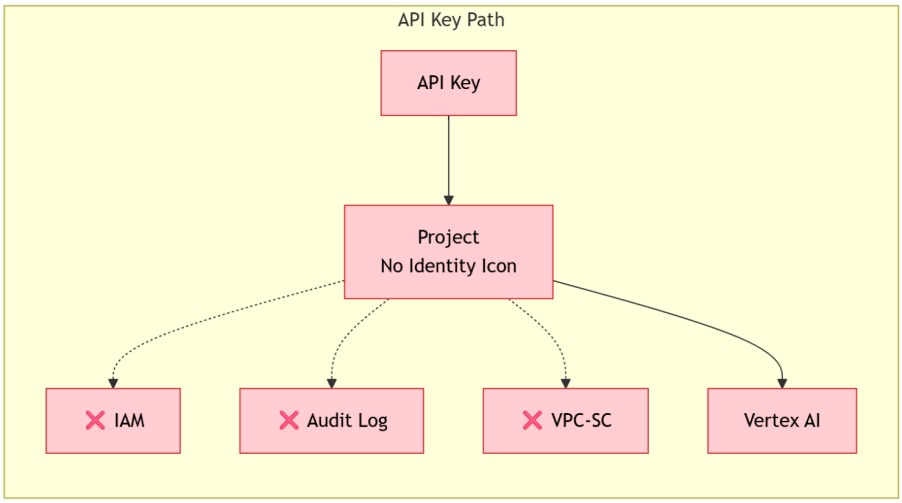
1. In the search bar at the top, type **Vertex AI**.
2. Click **Vertex AI API** in the results.
3. Click **Enable** and wait approximately 30 seconds for activation.

CLI

```
gcloud services enable aiplatform.googleapis.com
```

Why this guide does not use an API key





The Gemini API (the standalone developer product, separate from Vertex AI) authenticates with a simple API key. Vertex AI deliberately does not offer this as the primary path, and this guide does not use API keys anywhere, for a concrete technical reason: **a Google Cloud API key identifies a project, not a caller.** It links a request to a project for billing and quota purposes, but it carries no information about who or what is making the call. Because IAM authorization decisions ("can this caller perform this action on this resource") require a principal, and an API key is not a principal, API-key-authenticated requests cannot be governed by IAM, audit-logged to a specific identity, or scoped with VPC Service Controls in the same way.

The practical risk this creates: if an API key leaks (committed to a repository, exposed in client-side code, logged accidentally), anyone holding it can make calls against the project's quota and billing, with no way to revoke access for just that caller without rotating the key for everyone using it. There is no per-caller audit trail to show which part of the system the leaked key came from.

Workforce Identity Federation, Application Default Credentials (ADC), and service accounts solve this by authenticating as an identity rather than a shared secret. These are not three competing options. ADC is the lookup order a client library follows to find a credential. The other two are sources that lookup can find:

- **Workforce Identity Federation** lets a human user authenticate using their existing corporate identity (Okta, Microsoft Entra ID), so every request is traceable to a specific employee rather than a static key.

- **ADC** (`gcloud auth application-default login` for local development, or automatic on Google Cloud compute) is how the client library locates whichever credential is available, so no long-lived secret has to appear in code.
- **A service account**, bound to a specific application via IAM, gives that application its own identity, its own audit trail, and credentials that are short-lived and automatically rotated rather than a static string that can be copy-pasted indefinitely.

- **Workload Identity Federation** extends the same idea to applications that run outside Google Cloud, such as a build pipeline or a container on another cloud. Section 5.4 covers it.

All of these integrate with IAM, so access can be granted, scoped, and revoked per identity; with Cloud Audit Logs, so every call is attributed to a specific principal; and with VPC Service Controls, so network-level restrictions can be enforced regardless of whether credentials are compromised. None of this is available to a bare API key. This is why every code example in this guide authenticates through ADC, backed by a service account or a federated identity, never an API key.

Authenticate

Console

Authentication in the Console occurs automatically when you sign in with your Google account. To test Gemini in Vertex AI without writing code, go to **Vertex AI** > **Vertex AI Studio** > **Overview**, then open the **Generative AI Studio** environment to send prompts directly in the browser.

CLI

```
# For local development, authenticate with your own account
gcloud auth application-default login
```

If you receive an `UNAUTHENTICATED` error after logging in, verify that the Vertex AI API is enabled in the active project and that `gcloud config get project` returns the correct project ID.

Authenticate with Workload Identity Federation

The command above works when you sign in with a Google account. Two common situations need a different approach.

If your application runs outside Google Cloud, for example on Amazon Web Services, Azure, GitHub Actions, or your own servers, use Workload Identity Federation. The application exchanges a credential it already holds for a short-lived Google token, so no service account key file is needed.

If you sign in through a corporate identity provider such as Okta or Microsoft Entra ID and have no Google account, use Workforce Identity Federation. The sign-in runs in a browser and writes the result to the same place that ADC reads.

Section 5.4 covers both, with Console and CLI steps. In either case the Python code below does not change, because ADC is the lookup mechanism and federation only changes what that lookup finds.

Install the SDK and make your first call

```
pip install google-genai
```

```
from google import genai
from google.genai import types

client = genai.Client(
  vertexai=True,
  project="YOUR_PROJECT_ID",
  location="global",
)

response = client.models.generate_content(
  model="gemini-3.8-flash",
  contents="Explain what Vertex AI is in two sentences.",
  config=types.GenerateContentConfig(
    max_output_tokens=256,
  ),
)

print(response.text)
```

If you receive a `403 Permission Denied` error, your account does not have the `roles/aiplatform.user` role on the project. See Section 5.1 for role assignment steps.

Model names used in this guide, such as `gemini-3.8-flash`, reflect the models available when the code examples were written. This edition was reviewed in September 2026. At that review, `gemini-3.8-flash` was the newest stable Flash-tier model. Model names and retirement dates change often. Do not treat the names in this guide as current. Always check the Gemini Enterprise Agent Platform model versions page for the current model list and retirement schedule before production deployment. Avoid hardcoding specific model version strings where possible; instead, abstract the model name into a configuration value that can be updated without a code change.

Preview models vs stable models

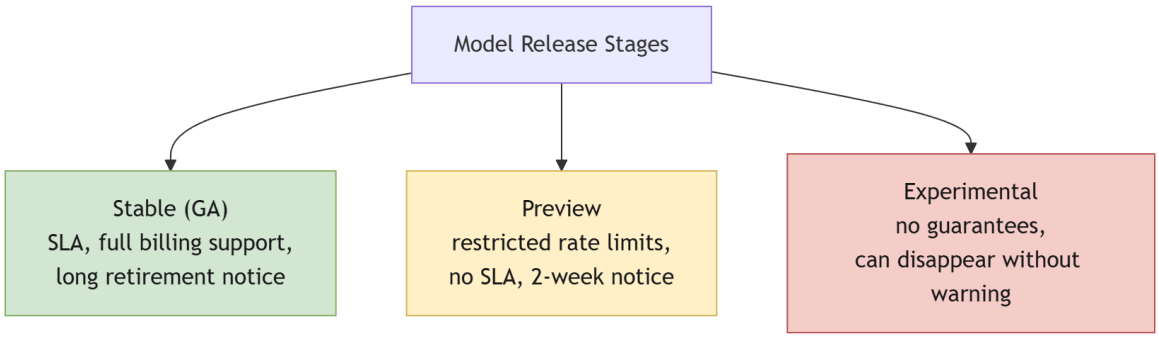
Every Gemini model on Vertex AI carries one of three lifecycle stages, and the stage determines what you can and cannot rely on in production.

A **stable** (GA) model has passed Google's production readiness review. It comes with a Service Level Agreement, full billing support, quota that you can request increases for, and a retirement notice period long enough to plan a migration. Use stable models for any workload where downtime or unexpected behavior change has a business cost.

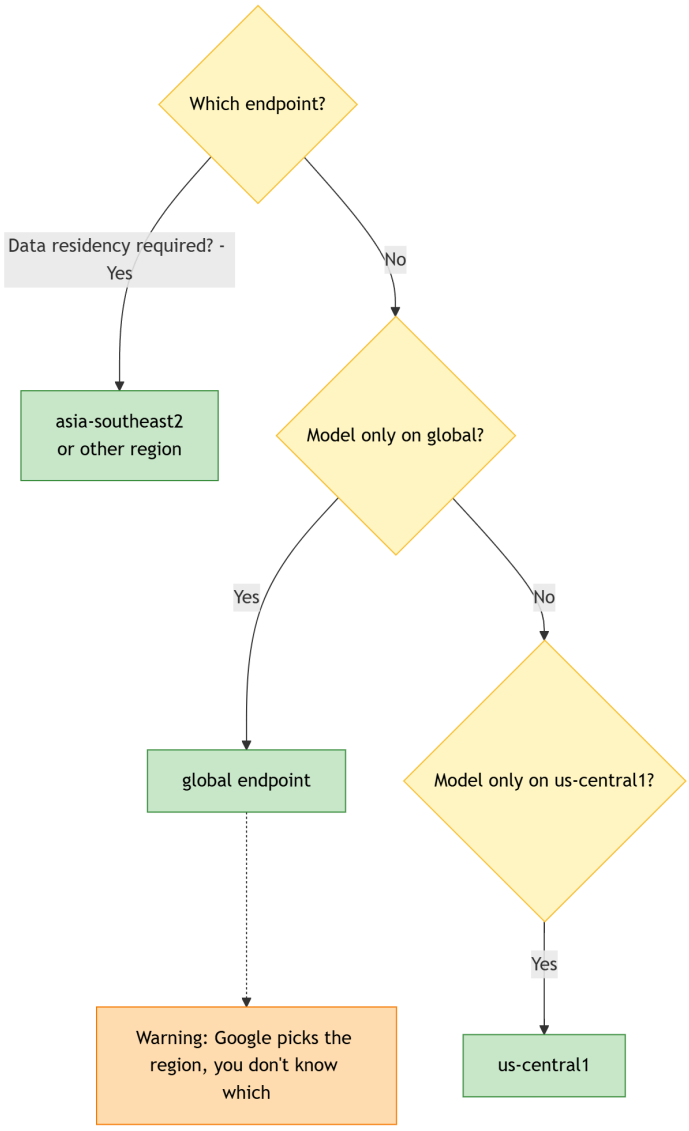
A **preview** model is available for production use but comes with restrictions. Rate limits are more restrictive than stable, there is no SLA, and Google only commits to at least two weeks notice before deprecating it. Preview models are how Google makes new capabilities available before the full production hardening is complete. The practical risk is that a preview model can change behavior or be shut down on short notice. If you use one, you should treat the model name as an expiry date and monitor the deprecations page regularly.

An **experimental** model is the least stable. It is released to gather developer feedback and get new capabilities into the ecosystem quickly. Experimental models are not suitable for production. Endpoints can disappear without warning.

The naming convention reflects this. At the September 2026 review, `gemini-2.5-flash` was stable, `gemini-3.8-flash` was the newest GA Flash-tier model, `gemini-3.1-pro-preview` was preview, and names with `-exp` or `-experimental` suffixes were experimental. Check the model versions page above for the current state of each model.



Regional endpoints, global endpoint, and the URL difference



Not all models are available in all regions, and this affects the URL your application must call. Getting this wrong produces a `404 NOT_FOUND` or `ENOTFOUND` error that can be confusing because the model name is correct but the endpoint is not.

There are two distinct endpoint patterns, and they use genuinely different hostnames:

For a **regional endpoint** (for example, `asia-southeast2` or `us-central1`), the hostname carries the region prefix:

```
https://{region}-aiplatform.googleapis.com/v1/projects/{project}/locations/{region}/publishers/google/models/{model}:generateContent
```

For the **global endpoint**, there is no region prefix in the hostname at all:

`https://aiplatform.googleapis.com/v1/projects/{project}/locations/global/publishers/google/models/{model}:generateContent`

This is the source of a common mistake: some SDKs and tools construct the URL by prepending the location string to `aiplatform.googleapis.com`. When the location is `global`, this produces `https://global-aiplatform.googleapis.com`, which does not exist and returns `ENOTFOUND`. The correct global hostname has no prefix. This has been reported in multiple GitHub issues (vercel/ai #6811, vercel/ai #14702, BerriAI/litellm #9234).

In the Google Gen AI SDK for Python, passing `location="global"` in the client correctly routes to the global endpoint. Where you need to construct the URL manually or work around an SDK that does not handle this correctly, the pattern used in practice is an explicit condition on the location value:

```
from google import genai
from google.genai.types import HttpOptions

LOCATION = "global"  # or "asia-southeast2", "us-central1", etc.
PROJECT = "YOUR_PROJECT_ID"

if LOCATION == "global":
    # Global endpoint: no region prefix in hostname
    base_url = "https://aiplatform.googleapis.com"
else:
    # Regional endpoint: region prefix in hostname
    base_url = f"https://{LOCATION}-aiplatform.googleapis.com"

client = genai.Client(
    vertexai=True,
    project=PROJECT,
    location=LOCATION,
    http_options=HttpOptions(api_version="v1", base_url=base_url),
)
```

When to use which endpoint:

- **Jakarta (`asia-southeast2`) or another specific region:** use the regional endpoint. Data you store stays at rest in that region. Processing is covered by a separate per-model table on the data residency page, and Jakarta is not listed in it, so check that table before you rely on this endpoint for residency obligations. Other Vertex AI services run here, including custom model training, custom model inference, Vector Search and Model Registry.

- **`global`:** use when a specific model, particularly a new preview model, is only available on the global endpoint. Google routes the request internally to a region with available capacity. You cannot control or know which region handles the request, so this is not appropriate for workloads with data residency requirements.
- **`us-central1`:** some models, especially older preview releases, are only available in this specific US region and not on the global endpoint. If a model returns `NOT_FOUND` on both Jakarta and global, try `us-central1` and check the model's documentation page for its supported locations.

Always check the specific model's documentation page on the Vertex AI models list to see which locations it supports before writing your endpoint configuration.

Secrets management

Credentials and configuration values required by your application such as service account keys, database connection strings, or third-party API tokens should be stored in Secret Manager rather than in source code or environment files committed to version control.

Console

1. Go to **Security** > **Secret Manager** in the navigation menu.
2. Click **Enable API** if prompted.
3. Click **Create Secret**, give it a name, paste or upload the secret value, and click **Create**.
4. To grant access: open the secret, go to the **Permissions** tab, click **Grant Access**, enter the service account email, and assign the `Secret Manager Secret Accessor` role.

CLI

```
gcloud services enable secretmanager.googleapis.com

echo -n "your-secret-value" | \
 gcloud secrets create MY_SECRET \
 --data-file=- \
 --replication-policy="automatic"

gcloud secrets add-iam-policy-binding MY_SECRET \
 --member="serviceAccount:your-app@YOUR_PROJECT_ID.iam.gserviceaccount.com" \
 --role="roles/secretmanager.secretAccessor"

# Reading a secret at runtime
from google.cloud import secretmanager

def get_secret(project_id: str, secret_name: str) -> str:
  client = secretmanager.SecretManagerServiceClient()
  name = f"projects/{project_id}/secrets/{secret_name}/versions/latest"
```

```
    response = client.access_secret_version(request={"name": name})
    return response.payload.data.decode("UTF-8")
```

5.4 Authenticating from Outside Google Cloud

Overview

Section 5.3 used Application Default Credentials (ADC) to make the first Gemini call. That worked because a developer was signed in with a Google account on a laptop. Production is rarely that simple. The code that calls Gemini often runs somewhere else: a build pipeline on GitHub, a container on Amazon Web Services, a server in your own data centre, or a laptop belonging to an employee who has no Google account at all.

This section covers how those callers authenticate.

One point first, because it is the most common misunderstanding. Identity federation does not replace ADC. ADC is the lookup order that the Google client libraries follow when they need a credential. Identity federation is one of the things that lookup can find. The two work together, and your Python code does not change.

Why this matters

The alternative to federation is a service account key file. A key file is a long-lived secret stored as JSON. If it is committed to a repository, pasted into a chat message, or copied onto a developer laptop, anyone holding it can call Vertex AI as that service account until someone notices and revokes it. Section 5.1 already sets an organization policy that blocks the creation of these keys, so teams that plan to use key files will find the policy stops them.

Identity federation removes the secret. The caller proves who it is using a credential it already holds, such as an Amazon Web Services role, a GitHub Actions token, or an Okta login. Google's Security Token Service checks that proof and returns a Google access token that expires within about an hour. There is no long-lived file to leak.

Google Cloud documentation states that this is the preferred approach for workloads running outside Google Cloud, and recommends it over creating a service account key.

Two products with similar names

Google offers two separate federation products. Section 5.1 introduced the first one. This guide writes both names in full and does not shorten either to WIF, because the short form is ambiguous between them.

Table: The two identity federation products and what each one authenticates.

Federation product	What signs in	Typical example

Workforce Identity Federation	A person	An employee signs in with Okta or Microsoft Entra ID and has no Google account
Workload Identity Federation	An application	A GitHub Actions job or a container on Amazon Web Services calls the Vertex AI API

A third name causes further confusion. Workload Identity on its own used to be the name of a Google Kubernetes Engine feature. That feature is now called Workload Identity Federation for GKE, and Google describes it as part of Workload Identity Federation. It is not a separate product.

When to use each

The right method depends on where your code runs, not on which product sounds more secure. Work down this table and stop at the first row that matches.

Table: Choosing an authentication method by runtime environment.

Where your code runs	Method to use	Federation product
Cloud Run, GKE, or Compute Engine	Attached service account, automatic	None needed
Developer laptop with a Google account	gcloud auth application-default login	None needed
Amazon Web Services, Azure, GitHub Actions, on-premises	Credential configuration file	Workload Identity Federation
Developer laptop with a non-Google identity provider	Browser sign-in through the corporate identity provider	Workforce Identity Federation

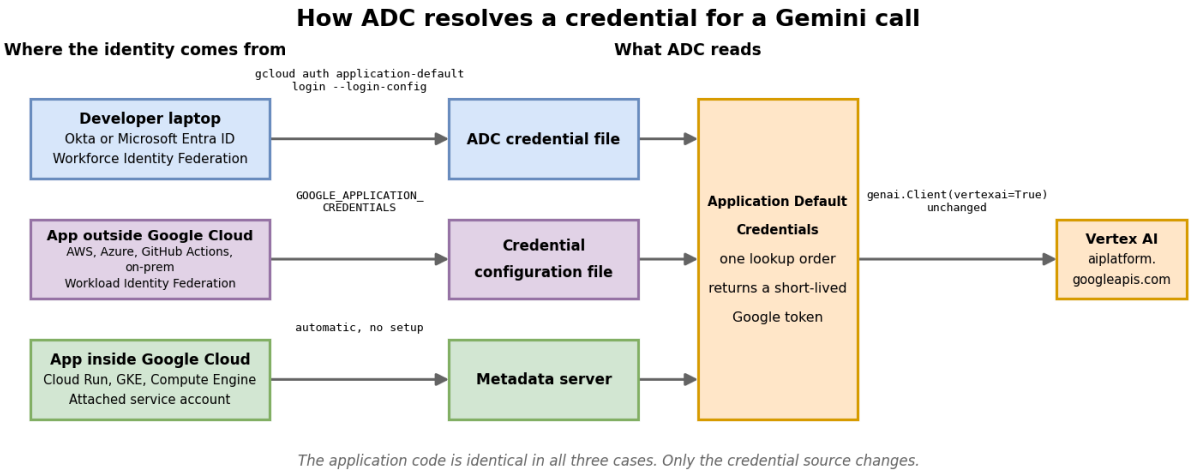


Figure: Application Default Credentials is the constant. Only the credential source changes.

Managed workload identities

Managed workload identities are a separate mechanism for workloads that run on Google Cloud. Google Cloud provisions X.509 credentials from Certificate Authority Service and binds them to a Compute Engine or Google Kubernetes Engine workload, so workloads can authenticate to each other with mutual TLS. At the September 2026 review the Compute Engine form was generally available and the GKE form was in Preview. This is not an alternative to Workload Identity Federation. Federation covers code running outside Google Cloud, and managed workload identities cover code running inside it.

Authenticate with Workload Identity Federation

This is the Workload Identity Federation path. You create a pool that represents a trust boundary, add a provider that describes the external identity system, then grant that identity permission to act in your project. The example below uses GitHub Actions. The same shape applies to Amazon Web Services, Azure, and any provider that supports OpenID Connect.

Console

1. Go to IAM and Admin, then Workload Identity Federation.
2. Click Create Pool. Enter a name and description, then click Continue.
3. Select the provider type. Choose OpenID Connect (OIDC) for GitHub Actions, or AWS for an Amazon Web Services role.
4. Enter the Issuer URL. For GitHub Actions this is https://token.actions.githubusercontent.com.
5. Under Attribute Mapping, map google.subject to the claim that identifies the workload.

6. Under Attribute Conditions, add a condition that limits the pool to your own repository or account. This step is not optional in practice. See the common mistakes below.

7. Click Save, then use Grant Access to download the credential configuration file.

CLI

```
# Create the pool that will hold external workload identities
gcloud iam workload-identity-pools create external-workloads \
  --location=global \
  --display-name="External Workloads" \
  --project=YOUR_PROJECT_ID

# Add an OpenID Connect provider. This example is for GitHub Actions.
# The attribute condition limits the pool to one repository. Without it,
# any GitHub repository could request a token from this pool.
gcloud iam workload-identity-pools providers create-oidc github-provider \
  --location=global \
  --workload-identity-pool=external-workloads \
  --issuer-uri="https://token.actions.githubusercontent.com" \

--attribute-mapping="google.subject=assertion.sub,attribute.repository=a
ssertion.repository" \

--attribute-condition="assertion.repository=='YOUR_GITHUB_ORG/YOUR_REPO'
" \
  --project=YOUR_PROJECT_ID
```

Next, allow the federated identity to impersonate the service account created in Section 6.2. That service account already holds roles/aiplatform.user, so the external workload inherits exactly those permissions and nothing more.

```
# Let the federated identity impersonate the existing Vertex AI service
account
gcloud iam service-accounts add-iam-policy-binding \
  vertexai-app@YOUR_PROJECT_ID.iam.gserviceaccount.com \
  --role="roles/iam.workloadIdentityUser" \

--member="principalSet://iam.googleapis.com/projects/YOUR_PROJECT_NUMBE
```

```
R/locations/global/workloadIdentityPools/external-workloads/attribute.repository/YOUR_GITHUB_ORG/YOUR_REPO" \
  --project=YOUR_PROJECT_ID
```

Then generate the credential configuration file that Application Default Credentials will read.

```
# Generate the credential configuration file
gcloud iam workload-identity-pools create-cred-config \

projects/YOUR_PROJECT_NUMBER/locations/global/workloadIdentityPools/external-workloads/providers/github-provider \
  --service-account=vertexai-app@YOUR_PROJECT_ID.iam.gserviceaccount.com \
  --output-file=credential-configuration.json \
  --credential-source-file=/path/to/oidc/token

# Point ADC at the file. The client library does the rest.
export
GOOGLE_APPLICATION_CREDENTIALS="/path/to/credential-configuration.json"
```

The application code is unchanged from Section 5.3. The client library reads the credential configuration file, exchanges the external token for a Google access token, and calls Vertex AI.

```
from google import genai
from google.genai import types

# No credential argument. ADC finds the credential configuration file
# through the GOOGLE_APPLICATION_CREDENTIALS environment variable.
client = genai.Client(
  vertexai=True,
  project="YOUR_PROJECT_ID",
  location="global",
)

try:
  response = client.models.generate_content(
    model="gemini-3.8-flash",
```

```
    contents="Summarise this quarter's risk report in three bullet
points.",
    config=types.GenerateContentConfig(
      max_output_tokens=512,
    ),
  )
  print(response.text)
except Exception as error:
  # A failure here is almost always the pool, the provider, the attribute
  # condition, or the IAM binding. It is rarely the Python code.
  print(f"Vertex AI call failed: {error}")
```

Authenticate with Workforce Identity Federation

This is the Workforce Identity Federation path. It assumes an administrator has already created the workforce pool and provider described in Section 5.1. The developer only needs the pool and provider identifiers.

Console

There is no Console step for the developer. Signing in to the Google Cloud console with a federated identity uses the sign-in URL your administrator publishes. The commands below are what allow code on the laptop to authenticate.

CLI

```
# Create the login configuration file once. Your administrator
# provides the pool and provider identifiers.
gcloud iam workforce-pools create-login-config \
  locations/global/workforcePools/YOUR_POOL_ID/providers/YOUR_PROVIDER_ID \
  --output-file=login-config.json

# Sign in through the corporate identity provider in a browser and
# write the result to the location that ADC reads.
gcloud auth application-default login --login-config=login-config.json

# User credentials need a quota project. Without this, some API calls
# fail with a "no quota project available" error.
gcloud auth application-default set-quota-project YOUR_PROJECT_ID
```

After this, the same Python code from Section 5.3 runs without modification.

Best practices

Always set an attribute condition on a Workload Identity Federation provider. Without one, any identity from that issuer can request a token from your pool. For GitHub Actions that means any repository on GitHub, not only yours.

Prefer service account impersonation over direct resource access. Impersonation works for every API and reuses the least-privilege service account from Section 6.2. Direct access is also open to you here, because Vertex AI is generally available for identity federation. The three limitations Google lists for it at the September 2026 review concern console email prefill, outbound email to federated users, and Colab Enterprise. None of them affects the API calls in this guide.

Use one pool for each trust boundary. Do not place unrelated providers in a single pool.

Keep granting roles to Google Groups rather than to individuals, as described in Section 6.1. Federation changes how an identity proves itself, not how you organize permissions.

Limit the session duration for Workforce Identity Federation so a stolen token stops working quickly and group membership changes take effect within a day.

Common mistakes

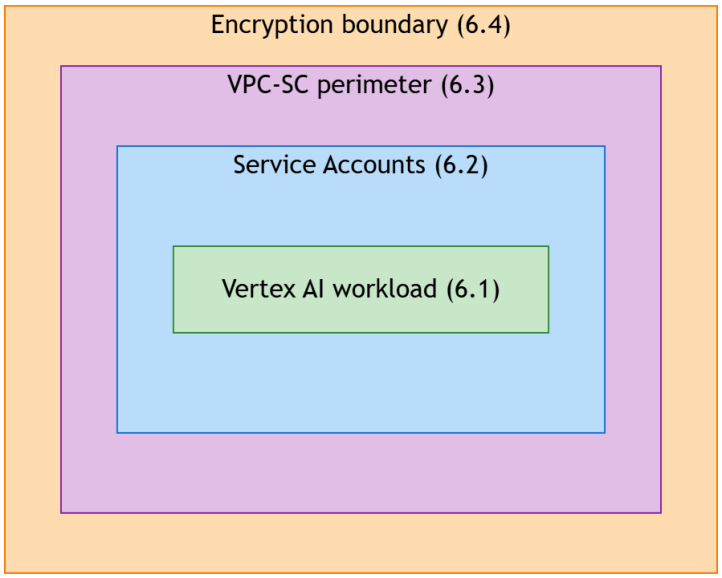
Creating a second pool instead of adding a provider to the existing one. Section 5.1 gives this warning for Workforce Identity Federation, and it applies equally here. Existing IAM bindings name the pool, so a new pool silently breaks them.

Treating the credential configuration file as a secret. It contains no private key, and Google states it does not need to be kept confidential. The protection comes from the attribute condition on the provider, not from hiding the file. Storing it in Secret Manager adds work without adding safety.

Forgetting the quota project after a workforce sign-in, then reporting the resulting error as a permissions problem. See Section 8.2.

Expecting the application code to change. It does not. If a federated call fails, look at the pool, the provider, the attribute condition, and the IAM binding before looking at the Python.

6. Securing Your Environment



6.1 IAM Best Practices

Access control in production goes beyond initial role assignment. Permissions accumulate over time as teams grow, projects evolve, and roles are granted for temporary needs that never get cleaned up. Conducting periodic access reviews and enforcing the principle of least privilege from the start reduces the surface area for both accidental and intentional misuse.

The following practices apply across all Vertex AI deployments regardless of size.

Group-based access

Assign all IAM roles to Google Groups, never to individual user accounts. This ensures permissions remain consistent when team members join, leave, or change responsibilities.

Avoid primitive roles

Roles like `roles/owner` and `roles/editor` grant far more permissions than most workloads need. Use predefined roles scoped to specific services, such as `roles/aiplatform.user` for Vertex AI access.

Regular access reviews

Review IAM policies quarterly or whenever a significant team change occurs. Remove permissions that are no longer needed.

Short-lived credentials for applications

Applications should authenticate using short-lived credentials rather than long-lived service account keys. Inside Google Cloud this means an attached service account, which the runtime

provides automatically. Outside Google Cloud it means Workload Identity Federation, covered in Section 5.4. Keys committed to a code repository, even private ones, are a common source of security incidents.

6.2 Service Accounts

Service accounts are identities used by applications, pipelines, and automated systems to authenticate with Google Cloud. Unlike user accounts, they are not associated with a person. A service account used by a Vertex AI application should have only the permissions that application needs and nothing more.

The organization policy that disables service account key creation (configured in Section 5.1) enforces the use of short-lived credentials. If a key is required for a specific integration, treat it as a secret and store it in Secret Manager.

```
# Create a service account for a Vertex AI application
gcloud iam service-accounts create vertexai-app \
 --display-name="Vertex AI Application" \
 --project=YOUR_PROJECT_ID

# Grant it the minimum required permissions
gcloud projects add-iam-policy-binding YOUR_PROJECT_ID \

--member="serviceAccount:vertexai-app@YOUR_PROJECT_ID.iam.gserviceaccount.com"
\
 --role="roles/aiplatform.user"
```

6.3 Centralized Logging

Centralizing logs from all projects into a single dedicated logging project makes security monitoring manageable as the number of projects grows. It also ensures that audit trails cannot be tampered with at the project level.

Enable the following log types across all projects: Cloud Audit Logs (Admin Activity and Data Access), VPC Flow Logs, and Firewall Rules Logging.

Console

1. Go to **Logging** > **Log Router**.
2. Set the scope to your Organization node.
3. Click **Create Sink**, name it, and select a destination (Cloud Storage bucket or BigQuery dataset in your central logging project).
4. Set the filter to `logName:"cloudaudit.googleapis.com"`.
5. Enable **Include logs from child resources**.

6. Click **Create Sink**.

CLI

shell

```
gcloud logging sinks create org-central-sink \
 storage.googleapis.com/YOUR_LOG_BUCKET \
 --organization=YOUR_ORG_ID \
 --include-children \
 --log-filter='logName:"cloudaudit.googleapis.com"'
```

6.4 VPC Service Controls

VPC Service Controls create a security perimeter around Google Cloud resources. Once a perimeter is in place, requests from outside it even from authenticated identities with valid credentials are blocked. This is the primary technical control against data being copied to an unauthorized location: the scenario where stolen credentials are used to move data to an unauthorized location.

VPC-SC adds operational complexity. External integrations, outbound webhooks, and third-party connectors all involve traffic crossing the perimeter boundary, which is blocked by default and requires explicit configuration. Teams should evaluate whether the sensitivity of their data justifies this overhead. For workloads that handle personal data, financial data, or data subject to regulatory requirements, VPC-SC is a recommended control. For internet-facing workloads with non-sensitive data, standard IAM and network controls are generally sufficient.

Console

1. Go to **Security** > **VPC Service Controls**.
2. Click **New Perimeter**.
3. Enter a perimeter name and select **Service Perimeter** as the type.
4. Under **Projects**, add your Vertex AI project.
5. Under **Restricted services**, add `aiplatform.googleapis.com` and `storage.googleapis.com`.
6. Click **Create Perimeter**.

CLI

shell

```
gcloud services enable accesscontextmanager.googleapis.com

gcloud access-context-manager policies create \
 --organization=YOUR_ORG_ID \
 --title="Default Access Policy"
```

```
gcloud access-context-manager perimeters create vertexai-perimeter \
 --policy=YOUR_POLICY_ID \
 --title="Vertex AI Perimeter" \
 --resources=projects/YOUR_PROJECT_NUMBER \
 --restricted-services=aiplatform.googleapis.com,storage.googleapis.com
```

6.5 Encryption and Data Residency

Encryption at rest

Google Cloud encrypts all data at rest and in transit by default. For most organizations, default encryption is enough and does not need extra setup. For organizations that need independent key control, for example the ability to revoke access to encrypted data instantly by disabling a key, Customer-Managed Encryption Keys (CMEK) provide this capability through Cloud Key Management Service (Cloud KMS).

A critical constraint on CMEK: once a key is assigned to a Vertex AI resource or region, it cannot be changed or removed. The choice of key and region must be finalized before creating any Vertex AI resources.

Data residency

Google separates two kinds of data residency. Data at rest stays in the location you select, whichever endpoint a request calls. Where the model processes a request depends on the endpoint and the model, and Google lists the exact commitment for each model and location in a table on its data residency page. That table does not include `asia-southeast2`. It does include Singapore for some models, for example `gemini-3.5-flash` and `gemini-2.5-flash`, but not for `gemini-3.8-flash`, the model used in this guide's examples. Jakarta (`asia-southeast2`) remains the recommended default for storing data in Indonesia. Singapore (`asia-southeast1`) is the nearest location that does appear in that table.

Choosing a region gives you infrastructure-level data residency. Whether that satisfies your organization's regulatory obligations under GR 71/2019, OJK rules, or other sectoral regulations is a separate legal question that depends on your organization type, the nature of the data, and how Vertex AI is classified within your system. Your legal team should confirm what applies. Region selection is one input to that compliance picture, not the whole answer.

Console

1. Go to **Security** > **Key Management**.
2. Click **Create Key Ring**, select the region `asia-southeast2`, and click **Create**.
3. Click **Create Key** within the key ring, configure the key settings, and click **Create**.

4. Note the key resource name. You will reference this when configuring Vertex AI resources.

CLI

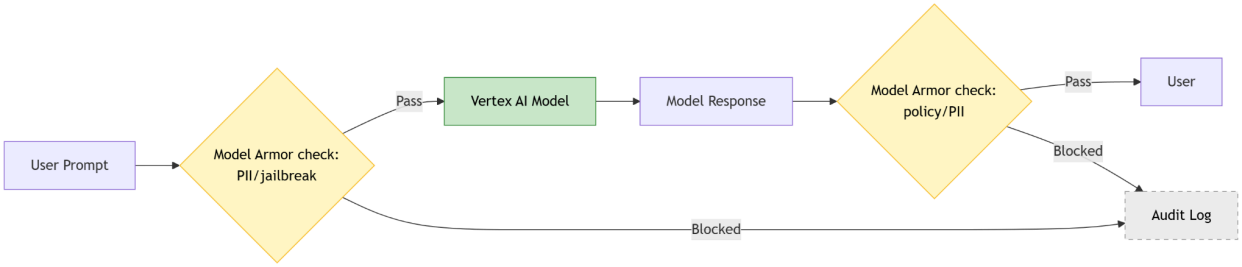
```
gcloud services enable cloudkms.googleapis.com

gcloud kms keyrings create vertexai-keyring \
 --location=asia-southeast2

gcloud kms keys create vertexai-data-key \
 --location=asia-southeast2 \
 --keyring=vertexai-keyring \
 --purpose=encryption
```

For Cloud Interconnect connections (direct private network links between your data center and Google), traffic on the dedicated line is not encrypted by default. MACsec must be enabled explicitly on the interconnect configuration.

6.6 Model Armor



Model Armor is a security layer that inspects both incoming prompts and outgoing model responses before they are processed or returned. It is designed to address risks specific to large language model deployments: sensitive data submitted by users, harmful prompts intended to override model instructions, and outputs that violate organizational policy.

For organizations in regulated industries financial services supervised by OJK, healthcare, or government Model Armor provides the automated controls and audit trails necessary for compliant AI deployment. For other organizations, it is a recommended security practice rather than a strict requirement.

Model Armor capabilities include:

PII and sensitive data redaction

Uses Google's Sensitive Data Protection service to identify sensitive data types including Indonesian NIK numbers, credit card numbers, and medical record identifiers, and automatically redacts or blocks this data before it reaches the model or appears in responses.

Harmful prompt defense

Detects and blocks prompt injection attacks, jailbreak attempts, and malicious content embedded in user-uploaded files.

Audit logging

Every filter action, whether a block or a redaction, is written to Cloud Logging with the specific policy template that triggered it. This provides the audit trail compliance officers need to demonstrate active monitoring.

Policy templates

Administrators define Security Templates with configurable risk thresholds. Different templates can apply to different user groups or applications within the same project.

Console

1. Go to **Security** > **Model Armor**.
2. Click **Enable** if the API is not yet active.
3. Click **Create Template** and select the region matching your Vertex AI deployment.
4. Configure filter settings: PII detection, prompt injection detection, and toxicity filtering.
5. Set confidence thresholds appropriate for your use case. Medium and above is a common starting point for enterprise deployments.
6. Click **Create** and note the template resource name.

The template region must match the region of the Vertex AI application using it. A template created in `us-central1` cannot be applied to a deployment in `asia-southeast2`.

CLI

```
gcloud services enable modelarmor.googleapis.com --project=YOUR_PROJECT_ID

gcloud projects add-iam-policy-binding YOUR_PROJECT_ID \

--member="serviceAccount:vertexai-app@YOUR_PROJECT_ID.iam.gserviceaccount.com"
\
 --role="roles/modelarmor.user"


from google.cloud import modelarmor_v1

client = modelarmor_v1.ModelArmorClient()

template = modelarmor_v1.Template(
  filter_config=modelarmor_v1.FilterConfig(
    rai_settings=modelarmor_v1.RaiFilterSettings(
```

```
      rai_filters=[
        modelarmor_v1.RaiFilter(
          filter_type=modelarmor_v1.RaiFilterType.DANGEROUS_CONTENT,

confidence_level=modelarmor_v1.DetectionConfidenceLevel.MEDIUM_AND_ABOVE,
        ),
      ]
    ),

pi_and_jailbreak_filter_settings=modelarmor_v1.PiAndJailbreakFilterSettings(

filter_enforcement=modelarmor_v1.PiAndJailbreakFilterSettings.EnforcementType.
ENABLED,

confidence_level=modelarmor_v1.DetectionConfidenceLevel.MEDIUM_AND_ABOVE,
    ),
  )
)

parent = "projects/YOUR_PROJECT_ID/locations/asia-southeast2"
created = client.create_template(
  parent=parent,
  template=template,
  template_id="enterprise-standard-template",
)
print(f"Created: {created.name}")
```

Model Armor at the gateway

Model Armor can also run at the gateway. Configured with Agent Gateway, it inspects every prompt and response passing through. The same content rules then apply to every agent the gateway governs, and no filtering logic goes into the agents themselves.

6.7 Responsible Usage and Cost Control

Everything in Sections 6.1 through 6.6 covers controls that can be configured once: organization policies, encryption, network perimeters, audit logging. Usage control is different in kind. It is not a switch that gets configured once and left alone, it depends on what your application sends to the model, how often, and under what conditions, on an ongoing basis. This section makes that distinction explicit.

What Cloud Ace configures, and what we cannot control

As a systems integrator or implementation partner, Cloud Ace is responsible for configuring the infrastructure covered in this guide. This includes IAM policies, organization policies, encryption, network perimeters, logging, and initial quota and budget setup. This is the infrastructure layer, and it can be configured and validated upfront.

What no implementation partner can control is runtime usage. This includes how many requests your application sends, what data and prompts are included in those requests, which models are called, whether retry logic creates unintended loops, and whether features that look lightweight in testing become expensive at production scale. These behaviors come from application design and user activity, and are determined after the system is deployed.

This distinction follows Google Cloud's shared responsibility model. Google provides and secures the underlying infrastructure and tools, while customers are responsible for access control, application security, and monitoring usage. Budget alerts and quota limits act as safeguards, but they only limit impact after usage occurs. They do not prevent inefficient or runaway usage by themselves.

The practical implication: ongoing usage discipline is the customer's responsibility, not something that can be delegated to infrastructure configuration alone. This includes monitoring spend against expectations, reviewing which features or teams are driving cost, validating that retry and looping safeguards from Section 5.2 are actually implemented in application code (not just documented), and deciding when usage patterns warrant tightening a quota rather than requesting an increase.

The remainder of this section covers two ways to implement that responsibility: a proxy-based approach with built-in budget controls, and a manual approach using only native Google Cloud quotas.

Recommended: a budget-aware proxy layer

Instead of relying only on billing alerts (Section 5.2), a more direct way to control usage is to place a layer between your application and Vertex AI that enforces limits before requests are executed. LiteLLM (Section 7.2) is suitable for this because it supports virtual keys, per-key budgets, and rate limits without requiring custom infrastructure.

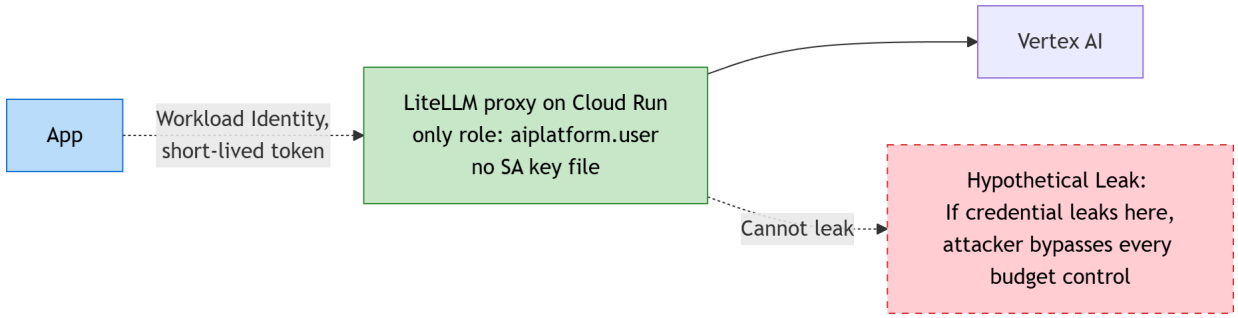
This is a recommended approach, not a requirement. If your organization does not allow additional third-party components in the request path, the manual approach below provides an alternative using only native Google Cloud controls.

Securing the proxy: why this only works if the proxy itself cannot be hijacked

A budget-enforcing proxy only works if it cannot be bypassed. If credentials used by the proxy are exposed, an attacker can call Vertex AI directly and completely skip all controls. The proxy only governs traffic that passes through it.

For this reason, the LiteLLM setup in Section 7.2 uses an attached service account instead of static service account keys:

- The Cloud Run service is bound to a service account with only `roles/aiplatform.user`
- No service account keys are used. Cloud Run obtains short-lived tokens automatically from the metadata server.
- An organization policy disables service account key creation to prevent accidental key generation.
- Proxy configuration (`master_key` and `litellm_config.yaml`) is stored in Secret Manager, not in code or images.
- Cloud Run is deployed with no unauthenticated access (`--no-allow-unauthenticated`), so only authenticated callers can reach it



Together, this ensures there is no long-lived credential that can be extracted and reused outside the system. The identity used to call Vertex AI is short-lived, scoped, and tied to the runtime environment.

Alternative: manual quota and limit reduction without a proxy

If you do not use LiteLLM or any proxy layer, usage control relies entirely on Vertex AI quota configuration. This is more manual and does not provide per-team budgets or detailed visibility. It is a fallback approach for teams that cannot introduce additional infrastructure.

The idea is to reduce per-model request quotas to match expected usage, instead of using default limits. This reduces the impact of runaway processes or compromised credentials.

Console

1. Go to **IAM & Admin** > **Quotas & System Limits**.
2. Filter by **Service: Vertex AI API**.
3. For each model used by your application, locate its request-per-minute quota. Quotas are typically defined per project, per region, and sometimes per model, so they must be configured individually.

4. Select the quota row, click the edit icon, and enter a new value at or near your actual expected peak usage rather than the default ceiling.
5. Repeat for every model and every region your application calls. There is no bulk-edit option in the Console; this is inherently a per-model, per-region operation.
6. Save each change individually.

CLI

```
# List current quota IDs for Vertex AI in your project, needed before you can target a specific quota
gcloud beta quotas info list \
 --service=aiplatform.googleapis.com \
 --project=YOUR_PROJECT_ID \
 --format="table(quotaId, metric, dimensions)"

# Request a quota decrease for one specific model/region quota
# (repeat this command once per model, once per region, there is no batch equivalent)
gcloud beta quotas preferences create \
 --project=YOUR_PROJECT_ID \
 --service=aiplatform.googleapis.com \
 --quota-id=YOUR_QUOTA_ID \
 --dimensions=region=asia-southeast2 \
 --preferred-value=60 \
 --justification="Manual usage ceiling, no proxy layer in path" \
 --preference-id=manual-limit-asia-southeast2-flash
```

Because this must be done for each model and each region, it becomes hard to manage when the number of models grows. It also cannot show spending clearly per team or per feature. Because of this, many teams choose to use a proxy layer for better control.

Reserved capacity

Provisioned Throughput is a fixed-cost, fixed-term subscription that reserves throughput for a chosen model in chosen locations. It is the opposite trade to the quota reductions above: quota limits protect you from spending too much, reserved throughput protects you from being throttled. Choose it when a workload has a known, steady volume that must not fail under load.

6.8 Agent Governance

Agent workloads have their own governance controls, separate from the IAM practices in Section 6.1. Three of them work together: Agent Identity gives each agent a cryptographic identity, Agent Registry catalogs the tools and destinations an agent may reach, and Agent Gateway enforces the rules and inspects the traffic between them.

Agent Identity provides a strongly attested identity for each agent, based on the SPIFFE standard. It extends the argument Section 6.2 makes against service account keys. Each agent gets its own identity rather than sharing one, and Google states that it cannot be impersonated and cannot produce long-lived keys. Access tokens are bound to the agent's X.509 certificate, which limits what a stolen token can do.

Agent Registry is a catalog of the agents, Model Context Protocol (MCP) servers, tools and endpoints in your environment. It gives you one inventory to govern, instead of separate lists held by each team. One caveat applies if your administrators sign in through Workforce Identity Federation. Agent Registry is in Preview for identity federation, and Google states that some Agent Platform Governance and Agent Gateway console pages have limited support for federated sessions.

Agent Gateway is the network entry and exit point for agent traffic. It applies the policies you define to interactions between users and agents, agents and tools, and agents and other agents.

IAM Unified Access Policies link an agent identity to the destinations it is allowed to reach. They are bound to the project that holds your Agent Gateway instances, and Agent Gateway uses Identity-Aware Proxy to enforce them. One limitation to plan around: Google states that this feature does not support VPC Service Controls, so it does not fit inside the perimeter described in Section 6.4.

Semantic governance policies are a Preview feature that adds a further check. The policy engine evaluates each tool call an agent proposes and denies calls that violate your policies or deviate from the user's intent. Treat it with care. Google states that the policy engine uses a language model to interpret policies written in natural language. Language models are probabilistic. Verdicts may not be accurate. Preview features also carry the limitations described in Section 5.3.

7. Building Applications

7.1 Prompting Guide

A prompt is the instruction sent to a Gemini model. The quality, structure, and length of a prompt directly affect response quality, consistency, and cost. Poorly written prompts result in inconsistent outputs that require expensive retry logic or manual review. Large language models respond best when prompts are clear, structured, and aligned with how the models were trained. Different techniques serve different tasks, and understanding which one to apply is a practical engineering decision.

System instruction vs prompt: the key distinction

These two concepts are often confused, and mixing them up produces unreliable results.

A **system instruction** (`system_instruction` in the API) is configuration that applies to the entire session or application. It tells the model who it is, what rules it follows, what format it should use, and what it should never do. Users cannot see it, and it does not change between requests. It is set by the developer, not the user.

A **prompt** (the `contents` field) is the actual request for a specific task. It contains the user's input, any examples, background context, and the specific question or instruction for that turn.

The practical rule: anything that should apply to every request goes in `system_instruction`. Anything specific to the current task goes in the prompt.

```
from google import genai
from google.genai import types

client = genai.Client(vertexai=True, project="YOUR_PROJECT_ID",
location="global")

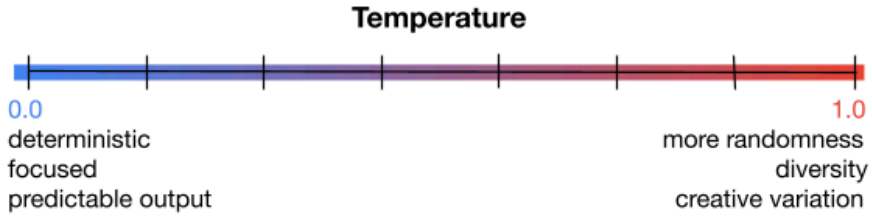
response = client.models.generate_content(
    model="gemini-3.8-flash",
    contents=[
        types.Content(
            role="user",
            # This is the prompt: task-specific, changes per request
            parts=[types.Part(text="Analisis data penjualan berikut dan
berikan rekomendasi.")]
        )
    ],
    config=types.GenerateContentConfig(
        # This is the system instruction: applies to every request in this app
```

```
        system_instruction="""You are a senior data analyst specializing in
customer behavior.
Always respond in Bahasa Indonesia.
Keep responses under 150 words.
Format output as bullet points.""",
        temperature=0.2,
        max_output_tokens=300,
    ),
)
```

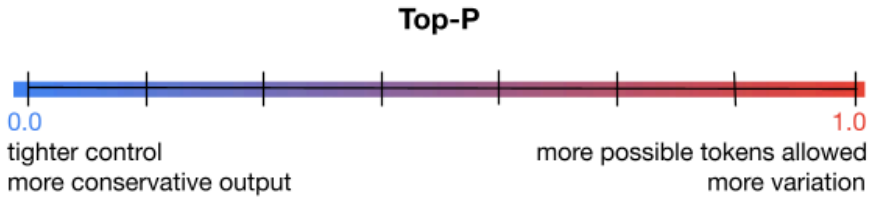
Parameters

Three parameters control how the model generates text. Setting them deliberately based on the task produces more predictable and cost-effective results.

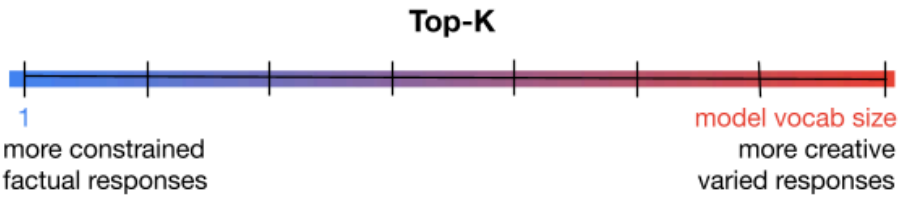
Temperature controls output predictability. A low value (closer to 0.0) makes the model consistently choose the most likely next word, producing factual and repeatable output. A higher value allows it to consider less likely words, producing more varied or creative output. For classification, data extraction, and factual tasks, a value between 0.1 and 0.3 is a common starting point. For content generation or creative tasks, 0.6 to 0.9 is more appropriate.



Top-P controls the range of words the model considers at each step. A value of 0.95 allows a wide range of options. Lowering it restricts output variety. A value of 0.95 is a reasonable default for most tasks.



Top-K sets a hard limit on how many candidate words the model evaluates. A value of 40 is a common default. Lower values produce more constrained output.



How to set parameters

```
config=types.GenerateContentConfig(
    temperature=0.1,   # low for factual/classification tasks
    top_p=0.9,
    top_k=20,
    max_output_tokens=50,
)
```

Zero-shot prompting

The simplest approach: describe the task with no examples. Works well for straightforward tasks where the model already has enough knowledge to respond correctly.

```
prompt = "Classify the sentiment of this review as positive, negative, or
neutral: 'Pengirimannya cepat tapi produknya rusak.'"
```

One-shot and few-shot prompting

Providing one to five examples of the expected input and output format produces more consistent results than describing the format in words. This is especially effective for structured outputs such as JSON extraction, where the model needs to match a specific shape exactly.

```
prompt = """Extract the company name and invoice amount from each description.

Example 1:
Input: PT Maju Bersama mengirimkan tagihan sebesar Rp 5.000.000 untuk jasa
konsultasi.
Output: {"company": "PT Maju Bersama", "amount": 5000000}

Example 2:
Input: Invoice dari CV Teknologi Nusantara senilai Rp 12.500.000 untuk
pengembangan sistem.
Output: {"company": "CV Teknologi Nusantara", "amount": 12500000}

Now extract from this:
Input: PT Global Solusi menagih Rp 8.750.000 atas layanan maintenance server.
Output:"""
```

One example (one-shot) reduces ambiguity. Three to five examples (few-shot) help the model handle edge cases and maintain consistency across different input shapes.

Role prompting

Assigning an identity to the model shapes its tone, vocabulary, and level of detail. Role prompting is typically placed in the system instruction, not the prompt, since the role applies across all requests.

```
You are a senior data analyst specializing in customer behavior and e-commerce
insights.
```

The role affects how the model frames answers. A "senior data analyst" will include business implications. A "junior assistant" will use simpler language. Match the role to the audience reading the output.

Contextual prompting

Contextual prompting adds task-specific background information that the model needs to produce a relevant answer. This is the information the model cannot infer on its own: the current situation, the dataset being analyzed, or the user's goal.

```
Customer Data Summary:
- 60% of purchases occur on weekends.
- Customers aged 30-45 account for 50% of total revenue.
- Mobile purchases increased by 25% this quarter.
- Cart abandonment rate is 35%.
```

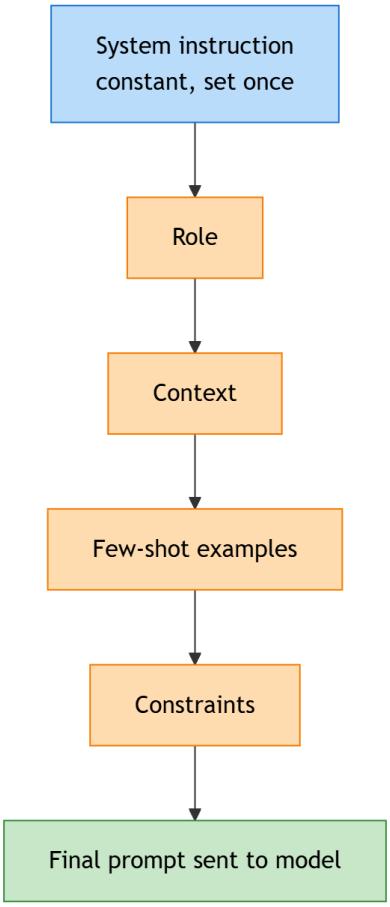
Context belongs in the prompt, not the system instruction, because it changes with each request.

Constraints

Constraints tell the model what limits to stay within: response length, format, number of items, what to include, and what to exclude. Without constraints, models often produce output that is longer or differently formatted than expected, which complicates downstream processing.

```
Limit response to 5 bullet points.
Do not repeat the input text.
Provide at least 2 actionable recommendations.
Keep total response under 150 words.
```

Combining techniques: a complete example



The following prompt combines role prompting, system instruction, few-shot examples, contextual input, and constraints.

```
system_instruction = """You are a senior data analyst specializing in customer
behavior and e-commerce insights.
Analyze the provided customer purchase data and identify key trends, unusual
patterns, and actionable business recommendations.
Keep the explanation clear and concise."""

prompt = """Example 1:
Input: Customers aged 18-25 purchase mostly discounted items.
Output Insight: Younger customers are price-sensitive and respond strongly to
promotions.

Example 2:
Input: Repeat customers have a 40% higher average order value.
Output Insight: Loyalty programs may increase revenue through higher basket
sizes.
```

```
Customer Data Summary:
- 60% of purchases occur on weekends.
- Customers aged 30-45 account for 50% of total revenue.
- Mobile purchases increased by 25% this quarter.
- Cart abandonment rate is 35%.

Constraints:
- Limit response to 5 bullet points.
- Do not repeat the input text.
- Provide at least 2 actionable recommendations.
- Keep total response under 150 words."""

response = client.models.generate_content(
    model="gemini-3.8-flash",
    contents=[types.Content(role="user", parts=[types.Part(text=prompt)])],
    config=types.GenerateContentConfig(
        system_instruction=system_instruction,
        temperature=0.2,
        max_output_tokens=300,
    ),
)
print(response.text)
```

Chain of Thought prompting

Chain of Thought (CoT) asks the model to reason through a problem step by step before giving a final answer. This reduces incorrect answers on tasks involving logic, calculation, or multi-step reasoning. Research shows CoT reduces hallucination rates compared to zero-shot prompting (Hoang et al., 2025). The tradeoff is cost: CoT responses use more output tokens. Use it when accuracy matters more than speed.

```
prompt = """Answer the following question by working through it step by step.

Question: A team uses a Flash-tier Gemini model for a chatbot. Each request
uses
approximately 500 input tokens and 300 output tokens. They expect 50,000
requests
per month. Using current Flash-tier pricing from the official pricing page,
estimate the monthly cost and determine whether it fits within a $50 budget.

Step-by-step reasoning:"""
```

Practical notes

Repeating the key instruction at both the start and end of a prompt can improve accuracy on non-reasoning models without increasing output tokens or latency. This is supported by research showing prompt repetition improves performance on models like Gemini without adding to generated output length (Leviathan et al., 2025, arxiv.org/abs/2512.14982).

Always set `max_output_tokens`. This is required not only for cost control but to stop looping output from consuming the entire token budget.

7.2 LiteLLM Integration (Optional)

LiteLLM is a third-party open-source library maintained independently of Google Cloud. It is not a Google product and not a standard component of a Vertex AI deployment. It provides a unified interface for calling multiple AI providers including Vertex AI, OpenAI, Anthropic, and others using the same code format, which is useful for teams that work across providers or want to switch models without rewriting application code.

This guide recommends LiteLLM's proxy mode specifically for the budget and rate-limit enforcement reasons described in Section 6.7: native Vertex AI quotas and billing alerts are useful backstops, but they act after the fact and apply at the project level, not per team or per application. LiteLLM's proxy gives you virtual API keys with individual budget caps and rate limits enforced before a request reaches Vertex AI, plus a dashboard showing spend broken down by key, team, and model. Teams that only use Vertex AI and have no plans to integrate other providers, but want this budget control, should still consider the proxy mode covered later in this section. Teams that want to compare models across providers, build fallback routing, or use an OpenAI-compatible format with a Vertex AI backend will also find it valuable for those reasons.

If you decide not to use LiteLLM, Section 6.7 covers a manual alternative using native Vertex AI quota controls.

Installation

```
pip install litellm

# For the proxy server mode used in team deployments:
pip install "litellm[proxy]"
```

Basic usage

```
import litellm

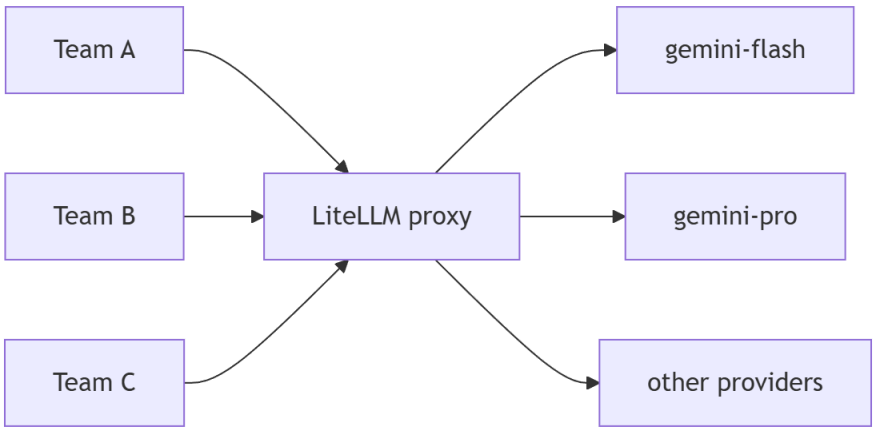
# Uses Application Default Credentials from gcloud auth application-default
login
response = litellm.completion(
  model="vertex_ai/gemini-3.8-flash",
  messages=[{"role": "user", "content": "Apa itu Vertex AI?"}],
```

```
  vertex_project="YOUR_PROJECT_ID",
  vertex_location="global",
  max_tokens=256,
)
print(response.choices[0].message.content)
```

Controlling reasoning cost through LiteLLM

```
# Use reasoning_effort to manage thinking token costs
response = litellm.completion(
  model="vertex_ai/gemini-3.8-flash",
  messages=[{"role": "user", "content": "Classify this text as positive or
negative."}],
  reasoning_effort="disable",  # options: disable, low, medium, high
  vertex_project="YOUR_PROJECT_ID",
  vertex_location="global",
  max_tokens=50,
)
```

Proxy setup for teams



The LiteLLM proxy runs as a server that your team connects to using standard OpenAI-compatible client code. It centralizes authentication with Vertex AI, tracks costs, and allows administrators to set rate limits per team or project.

Create a configuration file `litellm_config.yaml`:

```
model_list:
 - model_name: gemini-flash
  litellm_params:
   model: vertex_ai/gemini-3.8-flash
   vertex_project: YOUR_PROJECT_ID
   vertex_location: global
```

```
 - model_name: gemini-pro
  litellm_params:
   model: vertex_ai/gemini-3.1-pro-preview
   vertex_project: YOUR_PROJECT_ID
   vertex_location: global

litellm_settings:
 drop_params: true
 request_timeout: 60

general_settings:
 master_key: sk-your-internal-key-here
```

```
litellm --config litellm_config.yaml --port 4000
```

```
# Teams call the proxy using the standard OpenAI client
from openai import OpenAI

client = OpenAI(
  api_key="sk-your-internal-key-here",
  base_url="http://your-proxy-host:4000/v1"
)

response = client.chat.completions.create(
  model="gemini-flash",
  messages=[{"role": "user", "content": "Buatkan rangkuman singkat dari teks
ini."}],
)
print(response.choices[0].message.content)
```

Built-in dashboard

When the proxy is running, a built-in admin UI is available at `http://your-proxy-host:4000/ui`. It requires the `master_key` you set in the configuration to log in.

The dashboard is organized into several views. The Usage tab shows aggregate and per-model token consumption, total spend, request volume over time, and a breakdown by API key and team. The Models tab lists all configured models with per-model call counts, average latency,

and error rates. The Keys tab lets administrators create and revoke virtual API keys, set per-key rate limits and budget caps, and scope each key to specific models. The Teams tab manages multi-team access so different departments or projects can share the same proxy with separate budgets and quotas.

What you can trace and monitor through the dashboard:

- **Token usage:** input and output tokens per request, per model, and per API key, with time-series charts showing consumption trends
- **Spend tracking:** estimated cost per request, per model, per team, and per customer (when a user ID is passed in requests), broken down by day
- **Request volume:** total requests over time, with the ability to filter by model, team, or API key
- **Latency:** average and percentile latency per model, useful for comparing response time across Gemini Flash and Pro tiers
- **Error rates:** failed requests by error code, with links to the originating trace for investigation
- **Endpoint activity:** which application endpoints are generating the most traffic and spend, tracked automatically without extra configuration
- **Model distribution:** which models are called most often across your team, showing adoption patterns over time

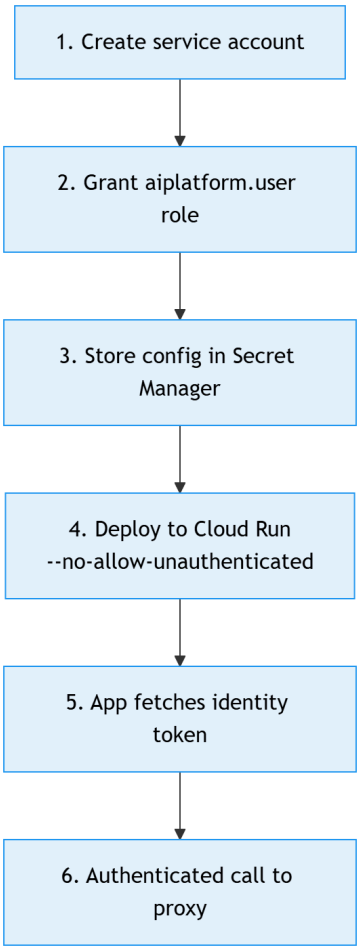
For deeper observability, LiteLLM supports exporting telemetry to external tools including Prometheus, Grafana, Langfuse, and OpenTelemetry-compatible platforms such as SigNoz and Datadog. This is useful when you need to correlate LiteLLM metrics with broader application performance data.

To enable Prometheus metrics, add the following to your `litellm_config.yaml`:

```
litellm_settings:
  callbacks: ["prometheus"]
```

Prometheus then exposes metrics at `http://your-proxy-host:4000/metrics`, which you can scrape into Grafana for dashboards that include request latency percentiles, token consumption by model, and budget remaining per team.

Deploying LiteLLM on Cloud Run



Running LiteLLM locally is useful for development, but for a shared team deployment the proxy should run as a persistent service. Cloud Run is a natural fit: it is serverless, scales to zero when idle, and stays inside GCP, which means the proxy can authenticate to Vertex AI using a service account rather than any credentials stored in environment variables.

The LiteLLM maintainers provide an official example repository for Cloud Run deployment at github.com/BerriAI/example_litellm_gcp_cloud_run. The steps below cover the core deployment path.

First, create a service account for the proxy and grant it the Vertex AI user role:

```
gcloud iam service-accounts create litellm-proxy \
  --display-name="LiteLLM Proxy" \
  --project=YOUR_PROJECT_ID

gcloud projects add-iam-policy-binding YOUR_PROJECT_ID \
```

```
--member="serviceAccount:litellm-proxy@YOUR_PROJECT_ID.iam.gserviceaccount.com
" \
  --role="roles/aiplatform.user"
```

Store your `litellm_config.yaml` in Secret Manager so it is not baked into the container image:

```
gcloud secrets create litellm-config \
  --data-file=litellm_config.yaml \
  --project=YOUR_PROJECT_ID

# Grant the service account access to read the secret
gcloud secrets add-iam-policy-binding litellm-config \

--member="serviceAccount:litellm-proxy@YOUR_PROJECT_ID.iam.gserviceaccount.com
" \
  --role="roles/secretmanager.secretAccessor" \
  --project=YOUR_PROJECT_ID
```

Deploy to Cloud Run using the official LiteLLM Docker image:

```
gcloud run deploy litellm-proxy \
  --image=docker.litellm.ai/berriai/litellm:main-stable \
  --platform=managed \
  --region=asia-southeast2 \
  --service-account=litellm-proxy@YOUR_PROJECT_ID.iam.gserviceaccount.com \
  --set-secrets=/app/config.yaml=litellm-config:latest \
  --args="--config,/app/config.yaml" \
  --no-allow-unauthenticated \
  --project=YOUR_PROJECT_ID
```

The `--no-allow-unauthenticated` flag ensures the Cloud Run service is not publicly accessible. Callers must present a valid Google identity token. Your application retrieves one at runtime:

```
import google.auth
import google.auth.transport.requests
import requests

def get_cloud_run_token(audience: str) -> str:
    credentials, _ = google.auth.default()
    auth_request = google.auth.transport.requests.Request()
    credentials.refresh(auth_request)
```

```
    return credentials.token

# Use it as the Authorization header when calling the proxy
LITELLM_URL = "https://your-cloud-run-url"
token = get_cloud_run_token(LITELLM_URL)

response = requests.post(
    f"{LITELLM_URL}/chat/completions",
    headers={"Authorization": f"Bearer {token}"},
    json={
        "model": "gemini-flash",
        "messages": [{"role": "user", "content": "Halo!"}],
    },
)
```

For production deployments, connecting LiteLLM to a PostgreSQL database (such as Cloud SQL) enables the full dashboard, virtual key management, and persistent spend tracking. Without a database, the proxy operates in stateless mode and the dashboard is limited to in-memory data that resets on each container restart.

7.3 Building a Chatbot with ADK

The direct Gemini SDK covered in Section 5.3 is well suited for single-turn requests: send a prompt, receive a response. For conversational applications where the model needs to remember what was said earlier in the same session, handle multi-turn dialogue, and optionally use tools or search, the Agent Development Kit (ADK) is the recommended approach.

ADK is an open-source framework from Google, available in Python, TypeScript, Go, and Java. It handles session state, conversation history, tool integration, and deployment to Agent Runtime without you needing to manage these manually. It uses Vertex AI as the backend, so authentication goes through your Google Cloud identity. No API key is required.

Installation

```
pip install google-adk
```

Environment setup

ADK reads project configuration from environment variables. Create a `.env` file in your project directory:

```
GOOGLE_GENAI_USE_VERTEXAI=1
GOOGLE_CLOUD_PROJECT=YOUR_PROJECT_ID
GOOGLE_CLOUD_LOCATION=asia-southeast2
```

Defining a chatbot agent

```
# agent.py
from google.adk.agents import Agent

root_agent = Agent(
    name="support_chatbot",
    model="gemini-3.8-flash",
    description="A customer support chatbot for a logistics company.",
    instruction="""You are a customer support assistant for a logistics
company.
Always respond in Bahasa Indonesia.
Only answer questions about order status, shipping, and delivery.
If a question is outside this scope, politely say you cannot help with it.
Keep responses concise and clear.""",
)
```

ADK requires the agent to be in a Python package. Create `__init__.py` in the same folder:

```
# __init__.py
from . import agent
```

Running locally

ADK includes two ways to test the agent before deployment:

```
# Interactive terminal chat
adk run path/to/your_agent_folder

# Browser-based chat UI for development
adk web path/to/your_agent_folder
```

The `adk web` command starts a local server with a chat interface at `http://localhost:8000`. This is useful for testing conversation flow and debugging session behavior before writing integration code.

Querying the agent from Python

For integration into a backend application:

```
import asyncio
from google.adk.agents import Agent
from google.adk.sessions import InMemorySessionService
from google.adk.runners import Runner
from google.genai.types import Content, Part
```

```
agent = Agent(
    name="support_chatbot",
    model="gemini-3.8-flash",
    instruction="You are a helpful customer support assistant. Respond in
Bahasa Indonesia.",
)

session_service = InMemorySessionService()
runner = Runner(
    agent=agent,
    app_name="support_chatbot",
    session_service=session_service,
)

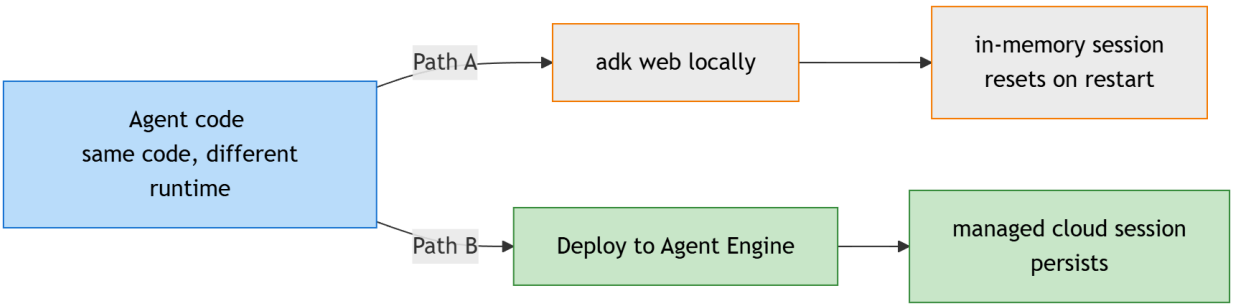
async def chat(user_id: str, session_id: str, message: str):
    content = Content(role="user", parts=[Part(text=message)])
    async for event in runner.run_async(
        user_id=user_id,
        session_id=session_id,
        new_message=content,
    ):
        if event.is_final_response():
            return event.content.parts[0].text

# Example usage
async def main():
    session = await session_service.create_session(
        app_name="support_chatbot",
        user_id="user-123",
    )
    response = await chat("user-123", session.id, "Di mana paket saya?")
    print(response)

asyncio.run(main())
```

Sessions in ADK maintain conversation history automatically. When you run locally, history is held in memory. When you deploy to Agent Runtime, it switches to managed cloud sessions automatically.

Deploying to Agent Runtime



Agent Runtime is the managed runtime for ADK agents, previously called Agent Engine. It handles scaling, session persistence across requests, and integration with other platform services. In September 2026 the Python SDK was restructured. The agent surface moved into the google-cloud-agentplatform package, and the global initializer was removed. Google advises pinning google-cloud-aiplatform below version 2.0.0 until you have migrated.

```
pip install "google-cloud-agentplatform[adk]"
```

```
import agentplatform
from vertexai.agent_engines import AdkApp
from your_agent_folder.agent import root_agent

client = agentplatform.Client(project="YOUR_PROJECT_ID", location="global")

# Wrap the agent for deployment
adk_app = AdkApp(agent=root_agent)

# Deploy to Agent Runtime
remote_agent = client.runtimes.create(
    agent=adk_app,
    config={
        "min_instances": 1,
    },
)
# The deployed resource name has the form:
# projects/PROJECT_ID/locations/LOCATION/reasoningEngines/RESOURCE_ID
```

Once deployed, sessions persist across server restarts and the agent scales automatically based on traffic. The resource name returned by the create call is the identifier you use to connect your application to the deployed agent.

When to use ADK vs the direct SDK

Scenario	Recommended approach
Single-turn requests (summarize, classify, extract)	Direct Gemini SDK (Section 5.3)
Multi-turn conversation with session memory	ADK
Agent that needs to call tools or search	ADK
Prototype or quick test	Direct SDK or `adk web` locally
Production chatbot with user sessions	ADK on Agent Runtime

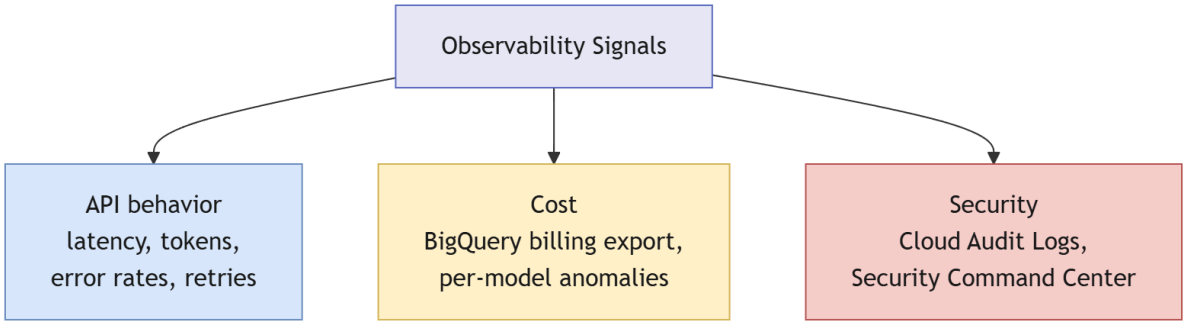
Memory across sessions

Memory Bank generates long-term memories from conversations between a user and an agent. Those memories persist across sessions. The chatbot built in this section keeps history within a session only. Memory Bank is what you add when the agent needs to remember a user between sessions.

8. Operating in Production

8.1 Monitoring

Effective monitoring in a Vertex AI deployment covers three areas: API behavior, cost, and security. Monitoring only one of these areas is a common gap that leads to late detection of issues.



For API behavior, the key metrics to track per request are latency, token usage (input, output, and thinking separately), error rates by error code, and retry counts. Aggregating these in Cloud Monitoring allows you to detect model degradation, unexpected cost growth, or application bugs before they become significant.

For cost, export billing data to BigQuery and set up a scheduled query to detect per-model and per-project cost anomalies. The following query identifies the most expensive Vertex AI operations in the current billing period.

```
SELECT
 sku.description AS operation,
 ROUND(SUM(cost), 2) AS total_cost_usd,
 SUM(usage.amount) AS total_usage
FROM `YOUR_PROJECT.billing_dataset.gcp_billing_export_v1_XXXXXX`
WHERE service.description = 'Vertex AI'
 AND invoice.month = FORMAT_DATE('%Y%m', CURRENT_DATE())
GROUP BY operation
ORDER BY total_cost_usd DESC
LIMIT 20;
```

For security, Cloud Audit Logs record every API call, IAM change, and administrative action. Security Command Center Premium provides threat detection and misconfiguration alerts at the organization level. Activate it for workloads that handle sensitive data.

Agent observability

Agent Observability gives visibility into the performance, behavior and health of deployed agents and MCP servers, including execution traces. It has a prerequisite. Your agents must send telemetry in OpenTelemetry format to Google Cloud Observability. Some built-in components emit it automatically, for example Model Armor, but agent code you write does not until you instrument it.

8.2 Troubleshooting



403 Permission Denied

Your account or service account does not have the required IAM role on the project.

```
# Verify which account is active
gcloud auth list

# Check IAM policy for the project
gcloud projects get-iam-policy YOUR_PROJECT_ID \
 --flatten="bindings[].members" \
 --filter="bindings.members:YOUR_EMAIL"

# Grant the required role if missing
gcloud projects add-iam-policy-binding YOUR_PROJECT_ID \
 --member="user:YOUR_EMAIL" \
```

```
 --role="roles/aiplatform.user"

# Verify the API is enabled
gcloud services list --enabled | grep aiplatform
```

No quota project available

This appears after signing in with a user account, including a federated sign-in through Workforce Identity Federation. The message may also read that the API is not enabled in the project. The cause is that user credentials do not carry a billing and quota project by default, so the API call has no project to charge.

It is not a permissions problem. Granting more IAM roles will not fix it.

```
# Attach a quota project to the credentials ADC is using
gcloud auth application-default set-quota-project YOUR_PROJECT_ID
```

Applications that use Workload Identity Federation with service account impersonation do not hit this, because the service account already carries a project.

429 Resource Exhausted

The project has reached its API rate limit or daily quota.

Add exponential backoff retry logic to all API calls. This is required in any production application that calls the Vertex AI API.

```
import time
import random
from google.api_core import exceptions
from google import genai
from google.genai import types

client = genai.Client(vertexai=True, project="YOUR_PROJECT_ID",
location="global")

def call_with_retry(contents: str, max_retries: int = 3):
  for attempt in range(max_retries):
    try:
      return client.models.generate_content(
        model="gemini-3.8-flash",
        contents=contents,
        config=types.GenerateContentConfig(max_output_tokens=512),
      )
```

```
    except exceptions.ResourceExhausted:
      if attempt == max_retries - 1:
        raise
      wait = (2 ** attempt) + random.uniform(0, 1)
      time.sleep(wait)
    except exceptions.InvalidArgument:
      raise # 400 errors are not retryable
```

For sustained quota issues, request an increase through **APIs & Services** > **Quotas & System Limits** in the Console.

VPC Service Controls violation

The error message contains a reference to `VPC Service Controls` or `vpcServiceControls`.

```
# View recent VPC-SC violations in audit logs
gcloud logging read \
 'protoPayload.status.details.violations.type="VPC_SERVICE_CONTROLS"' \
 --project=YOUR_PROJECT_ID \
 --limit=10
```

Common causes: the request originated from an IP address not in an approved access level, a service was not included in the perimeter's restricted or accessible service list, or a Google-managed service account was not added to the perimeter's ingress rules.

Request hangs with no response

On reasoning models, omitting `max_output_tokens` can cause the model to generate tokens indefinitely when the default thinking level is high. Set `max_output_tokens` on every request. For Gemini 3-series models, also set an explicit `thinking_level`.

```
config=types.GenerateContentConfig(
  max_output_tokens=8192,
  thinking_config=types.ThinkingConfig(thinking_level="MEDIUM"),
)
```

9. Production Deployment Checklist

Before moving any workload from sandbox or staging to production, verify that each of the following conditions is met.

Organization and Identity

- A Google Cloud Organization node exists and is linked to the company domain.
- All IAM roles are assigned to Google Groups, not individual user accounts.
- Workforce Identity Federation is configured if the organization uses a non-Google identity provider.
- Organization-level policies are in place: domain restriction, resource location restriction, and service account key creation disabled.
- No long-lived service account keys exist in the production project.

Billing and Cost Controls

- Budget alerts are configured at 50%, 90%, and 100% thresholds.
- An automated billing circuit breaker is deployed for sandbox and AI development projects.
- API quotas are set to a ceiling appropriate for expected production volume.
- `max_output_tokens` is explicitly set in every API call.
- Token usage (input, output, and thinking) is logged for every production request.
- No dedicated prediction endpoints are running without active traffic.

Security

- All secrets and credentials are stored in Secret Manager, not in source code or version-controlled configuration files.
- Encryption at rest is confirmed: default Google-managed or CMEK, depending on the organization's requirements.
- VPC Service Controls have been evaluated and configured where the data classification requires it.
- Centralized logging is active at the organization level, capturing audit logs from all child projects.
- Model Armor is configured with an appropriate security template for the application's user population and data sensitivity.
- Data Access Audit Logs are enabled for the Vertex AI API.

- No service account key files are used by workloads running outside Google Cloud. Workload Identity Federation is configured instead.

- Every Workload Identity Federation provider has an attribute condition that limits it to a named repository, account, or role.

- Federated identities reach Vertex AI through service account impersonation, or direct resource access has been confirmed as supported for the Vertex AI API.

- Developers using a non-Google identity provider have run gcloud auth application-default set-quota-project.

Application

- Retry logic with exponential backoff is implemented for 429 and 503 errors.
- Response validation is in place to detect malformed or empty outputs before they reach users.
- An incident response contact and escalation path is documented for billing anomalies and security events.
- The model and configuration have been tested against representative edge-case inputs.

10. Frequently Asked Questions (FAQ)

What is the difference between Vertex AI, Gemini Enterprise, Gemini in Workspace, and the Gemini API?

Google has several products that use the Gemini model family, and the names can be confusing. This guide covers only Vertex AI, which is the developer and enterprise platform for building AI applications on Google Cloud.

Gemini Enterprise (also called Agentspace) is a separate product aimed at knowledge workers inside a company. It provides a ready-made AI assistant connected to internal data sources like Google Drive, Gmail, and third-party tools. It is not the same as Vertex AI and is not covered in this guide.

Gemini in Google Workspace is the AI assistant built into Gmail, Docs, Sheets, and other Workspace apps. It is a consumer and productivity product, not a developer platform.

The Gemini API is a direct way to call Gemini models using an API key, primarily for prototyping. Vertex AI uses the same models but with full enterprise controls: IAM, audit logging, encryption, and network security. This guide covers Vertex AI.

If you are evaluating which product fits your use case, a simple way to decide is: if you are building an application or workflow that calls AI from code, use Vertex AI. If you want an AI assistant for your team's daily work without writing code, look at Gemini Enterprise or Gemini in Workspace instead.

Will Google use our company's data to train the Gemini model?

No. When using Vertex AI on a paid billing account, Google does not use customer data to train foundational models, does not use it for other customers' models, and does not use it for advertising. Customer data is owned by the customer; Google processes it on the customer's behalf. Conversation history retention periods and specific data handling terms are detailed in the Google Cloud Data Processing Addendum, which should be reviewed against your organization's compliance requirements.

Why did our API bill suddenly increase even though application usage did not change?

The most common causes, based on patterns reported across GitHub issues and the Google AI developer forum, are:

Model selection change, where a workload that previously used a Flash-tier model was moved to a Pro-tier model, multiplying cost per request significantly. Reasoning token growth, where newer Pro-tier models began spending more tokens on internal reasoning for certain query types without any visible change in the response. Looping output, where the model began generating repetitive content until the token limit was reached, consuming the full

`max_output_tokens` budget on requests that previously returned short responses. A forgotten dedicated endpoint billing by the hour with no traffic. A retry loop executing more times than expected due to rate limit errors. A context size crossing a pricing threshold on Pro-tier models.

Export billing data to BigQuery and query by SKU to identify the specific operation driving the increase (see Section 8.1). Also check `response.candidates[0].finish_reason` in your logs , a value of `MAX_TOKENS` on short responses is a signal of looping output.

The model is producing the same sentence or paragraph over and over. What is happening?

This is a known Gemini behavior reported across multiple GitHub repositories and the Google AI developer forum. The model enters a repetitive loop, generating the same tokens until it hits the `max_output_tokens` limit. It has been observed on Flash and Pro tier models, particularly when generating long documents, processing complex structured prompts, or running parallel tool calls.

The primary fix is to always set `max_output_tokens` to a value appropriate for your expected output. When the model loops, it stops at this limit. Also check `finish_reason` in every response. A `MAX_TOKENS` finish reason combined with repetitive content confirms a loop. Lowering temperature reduces the drift that can trigger this behavior. If the issue persists on a specific prompt, simplify or restructure the prompt to reduce ambiguity.

Why is the API returning 429 errors during peak usage?

The project has reached its rate limit for that model. Add exponential backoff retry logic (see Section 8.2) and request a quota increase if legitimate production traffic consistently reaches the ceiling.

What is the difference between Vertex AI and calling the Gemini API directly?

Both provide access to the same Gemini models, but through different paths with different trade-offs. The Gemini API uses a simple API key and is suited for individual developer prototyping. Vertex AI uses Google Cloud Identity for authentication, integrates with IAM, Cloud Audit Logs, VPC Service Controls, CMEK, and Model Armor, and is the recommended path for any team or production workload. This guide covers the Vertex AI path.

Our organization uses Okta for login. Can we still use Vertex AI?

Yes. Configure Workforce Identity Federation (Section 5.1) to connect Okta to Google Cloud. Employees authenticate with their existing Okta credentials and receive short-lived Google access tokens. No separate Google accounts are required.

How do we estimate cost before going to production?

Use the `count_tokens` API on a representative prompt to measure token volume (Section 5.2). Estimate monthly request volume. Multiply by current per-token rates from the official pricing page. Include a buffer for Indonesian text (approximately 20 to 30 percent more tokens than equivalent English), reasoning tokens if using a Pro-tier model, and expected growth. Use the worked example in Section 5.2 as a template.

Our organization is subject to OJK or healthcare data regulations. Can we use Vertex AI?

Vertex AI supports CMEK for encryption key control (Section 6.5), VPC Service Controls for network isolation (Section 6.4), Model Armor for PII redaction and audit logging (Section 6.6), and Cloud Audit Logs for compliance reporting (Section 6.3). Compliance certifications are maintained on an ongoing basis; verify the current list at cloud.google.com/security/compliance/services-in-scope before making claims to regulators.

Can our GitHub Actions pipeline call Gemini without a service account key file?

Yes. Configure Workload Identity Federation (Section 5.4) with an OpenID Connect provider pointed at GitHub. The pipeline exchanges its GitHub token for a short-lived Google token at run time. No key file is created, stored, or rotated. Add an attribute condition so only your repository can use the pool.

11. Glossary

ADK (Agent Development Kit)
An open-source framework from Google for building, testing, and deploying AI agents. Available in Python, TypeScript, Go, and Java. Integrates natively with Vertex AI and Gemini models. Recommended for conversational applications that require session management, tool use, or multi-turn dialogue. See google.github.io/adk-docs.

Application Default Credentials (ADC)
The lookup order that Google client libraries follow when they need a credential. ADC checks the GOOGLE_APPLICATION_CREDENTIALS environment variable, then a local file written by `gcloud auth application-default login`, then the service account attached to the compute resource. ADC is a lookup mechanism, not a credential type. It can return a service account credential or a federated credential.

Or use Service Account Impersonation

https://docs.cloud.google.com/docs/authentication/use-service-account-impersonation#gcloud-config

CMEK (Customer-Managed Encryption Key)
An option that gives an organization direct control over the encryption keys used to protect data at rest in Google Cloud. Keys are managed through Cloud KMS. Once assigned to a resource or region, they cannot be changed.

Cloud KMS (Key Management Service)
The Google Cloud service used to create, manage, and rotate cryptographic keys. Used in conjunction with CMEK.

Credential configuration file
A JSON file generated by gcloud that tells a Google client library how to obtain an external token and exchange it for a Google access token. It contains no private key and does not need to be kept confidential.

IAM (Identity and Access Management)
Google Cloud's system for controlling which identities can perform which actions on which resources. Access is granted through roles assigned to users, groups, or service accounts.

LiteLLM
A third-party open-source library that provides a unified interface for calling multiple AI providers using the same code format. Not a Google Cloud product.

Model Armor
A Google Cloud security layer that inspects AI prompts and responses for sensitive data, harmful inputs, and policy violations. Provides automated redaction, blocking, and audit logging.

Organization Policy
A rule applied at the Google Cloud Organization level that all projects and resources within the organization must follow. Cannot be overridden by project-level settings.

PII (Personally Identifiable Information)
Data that can identify a specific person: names, email addresses, phone numbers, national identity numbers (NIK), credit card numbers, and similar data.

Pub/Sub (Publish/Subscribe)
A Google Cloud messaging service used to connect event sources (such as billing alerts) to automated response systems (such as Cloud Functions).

Secret Manager
A Google Cloud service for storing sensitive values such as credentials, API tokens, and certificates securely. Applications retrieve secrets at runtime, keeping them out of source code.

Security Token Service
The Google Cloud service that validates a token issued by an external identity provider and exchanges it for a short-lived Google access token. It is the component that makes identity federation work.

Service Account
A non-human identity used by applications and automated systems to authenticate with Google Cloud. Service accounts are assigned IAM roles and should be granted only the permissions required for their specific function.

Thinking Tokens
Internal reasoning tokens generated by Pro-tier Gemini models before producing a visible answer. These tokens are billed at the same rate as output tokens but do not appear in the response text. They are reported in `usage_metadata.thoughts_token_count`.

Token
The unit used to measure text volume when calling Gemini models. Approximately three to four characters of English text, or three-quarters of an average English word. Pricing is calculated per million tokens.

Vertex AI
Google Cloud's managed platform for AI and machine learning. Provides access to Gemini models with enterprise controls including IAM, audit logging, CMEK, VPC integration, and compliance features.

VPC Service Controls (VPC-SC)
A Google Cloud feature that creates a security perimeter around resources. Requests originating from outside the perimeter are blocked even when the requester holds valid credentials.

Workforce Identity Federation
A Google Cloud feature that allows employees to authenticate to Google Cloud using an existing corporate identity provider such as Okta or Microsoft Entra ID, without needing a separate Google account. It authenticates people. This guide does not abbreviate the name, because the short form WIF is ambiguous with Workload Identity Federation.

Workload Identity Federation
A Google Cloud feature that allows an application running outside Google Cloud, for example on Amazon Web Services, Azure, or GitHub Actions, to obtain short-lived Google credentials without a service account key file. It authenticates applications. Workload Identity Federation for GKE is the Google Kubernetes Engine form of the same feature, previously called Workload Identity.

12. Legal Disclaimer and Limitation of Liability

This Vertex AI Adoption Guide has been prepared by Cloud Ace, an authorized Google Cloud Partner, to assist organizations in securely and efficiently deploying generative AI workloads on Google Cloud. While a primary objective of this guide is to share architectural and operational best practices to minimize the risk of sudden cost increases, cloud environments are inherently dynamic. Please carefully review the following limitations before utilizing the strategies outlined in this document.

- **No Liability for Cost Overruns or Billing Spikes:** Following Google Cloud's shared responsibility model, while Cloud Ace can advise on or help configure foundational infrastructure safeguards, runtime usage is completely outside our control. How many requests an application sends, the volume of tokens processed, and the behavior of application retry logic are determined entirely by post-deployment usage. Budget alerts act as notifications and only limit impact after usage occurs; they do not prevent inefficient or runaway usage by themselves. Consequently, Cloud Ace assumes no liability for any unexpected charges, billing spikes, or financial losses incurred on your Google Cloud account. Ongoing usage discipline is strictly the customer's responsibility.
- **Third-Party Tools and Advisory Content:** Any recommendations regarding non-Google Cloud tools, such as LiteLLM, are provided strictly in an advisory capacity. LiteLLM is an independent, open-source third-party library; it is not a Google product and is not a standard component of a Vertex AI deployment. The introduction of external dependencies carries inherent operational and security risks, including but not limited to known issues like supply chain attacks or unpatched vulnerabilities. Cloud Ace assumes no responsibility or liability for security breaches, outages, data leaks, or any other issues resulting from the implementation or use of third-party software.
- **Proactive Security and Verification:** Securing cloud AI environments is a continuous, evolving process rather than a one-time setup. We strongly advise your organization to adopt a proactive security posture. You must independently double-check and verify all

configurations, deployment scripts, code snippets, and third-party integrations against your internal security policies before moving any workload to production. Always consult the latest official Google Cloud security advisories to ensure your defenses remain up to date.

- **Point-in-Time Accuracy and Official Documentation:** This guide reflects the state of Google Cloud and Vertex AI as of its publication in the September 2026 Edition. Cloud computing platforms evolve rapidly; model names and retirement dates change often. Where examples include pricing, quotas, model availability, or retention periods, they should be treated as illustrative only. You must always refer to the official Google Cloud documentation for the latest product information, service limits, and pricing details before making technical or financial commitments.
- **Provided "As-Is" and Regulatory Compliance:** This document is provided strictly "as-is" without any warranties, express or implied, regarding its completeness, accuracy, or fitness for a particular purpose. Furthermore, compliance with data residency obligations and sector-specific regulations (such as GR 71/2019 or OJK rules) depends on your organization's classification and remains a separate legal question that your legal team should confirm. Compliance remains the sole responsibility of your organization.