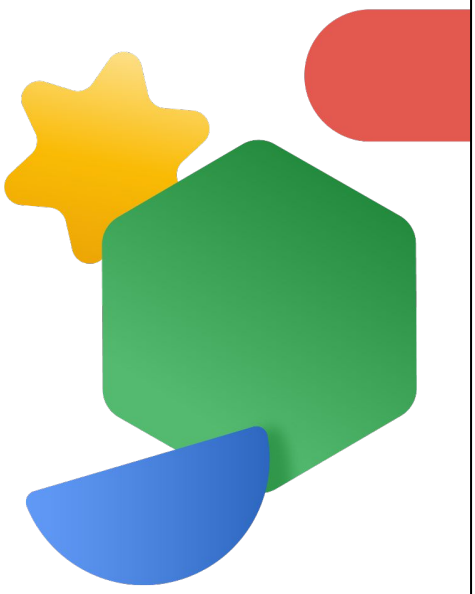
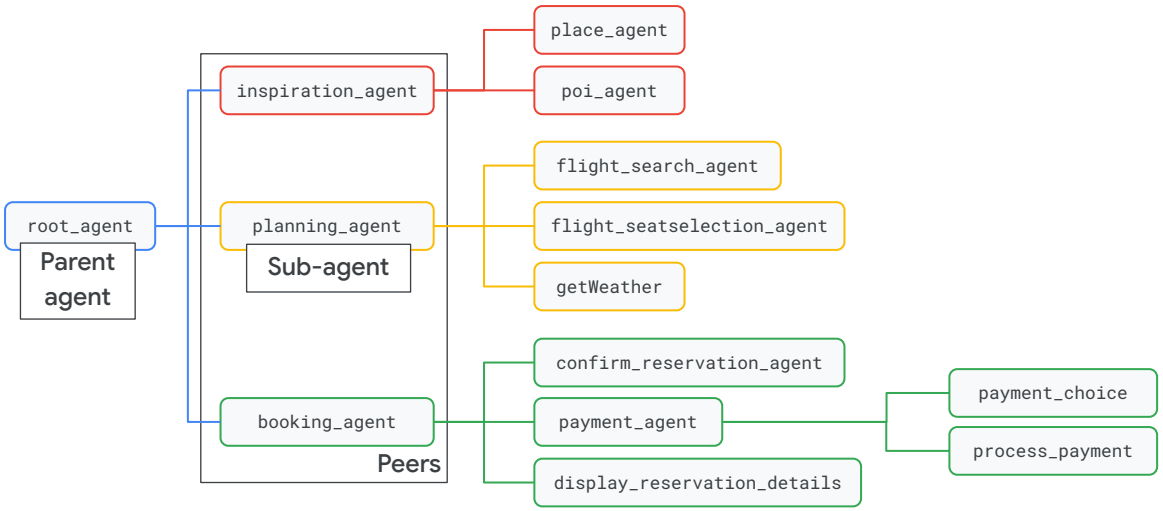
Introducing multi-agent systems



Let’s start with an introduction to multi-agent systems.

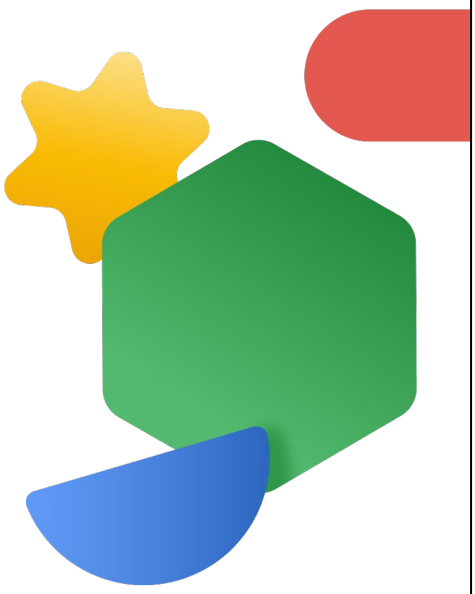
A hierarchical approach to multi-agent systems



The key idea with multi-agent systems in ADK is that there is a structure of parent and sub-agents constructed in a tree. Agents can only transfer the conversation to sub-agents, back to a parent, or to a peer (another agent that shares the same parent). You can disable transferring to peers on an agent-by-agent basis.

This tree structure makes conversations more predictable and reliable. As an alternative, if you could transfer to any agent, and two agents in different parts of your system were each responsible for certain lookup activities, it might be very easy to call the wrong agent. With this approach, only the lookup agent related to your part of the conversation will be invoked.

Agents forming part of multi-agent systems



Now let's take a look at the types of agents that could form part of multi-agent systems.

Large Language Model-based agents

- Bring intelligence and adaptability to your system
- Understand and respond to natural language
- Make decisions
- Utilize tools effectively

Large Language Model-based agents are the "brains" of your application, bringing intelligence and adaptability to your system. They leverage the capabilities of LLMs to understand and respond to natural language, make decisions, and utilize tools effectively.

They typically alternate turns of conversation with users.

Workflow agents

- Create flows directly from agent to agent
- Act as directors, to ensure tasks are performed in sequence and under specific conditions
- Control the flow of the context between their sub-agents
- Don’t have LLMs associated - they don’t reason themselves
- Deterministic - for a given input and configuration, the sequence of agents executed will always be the same

When you want multiple agents to be able to act without waiting on user turns in between each agent, workflow agents create flows directly from agent to agent. They are not powered by LLMs directly for their core function but act as directors, ensuring tasks are performed in the desired sequence and under specific conditions.

- Workflow agents control the flow of the context between their sub-agents.
- They don’t have a large language model associated, as they don’t reason themselves.
- Workflow agents typically result in deterministic workflow execution. This means that for a given input and configuration, the sequence of agents executed will always be the same. This predictability is valuable for structured processes where consistent and reliable execution is crucial.

Types of workflow agents

SequentialAgent	LoopAgent	ParallelAgent	Custom

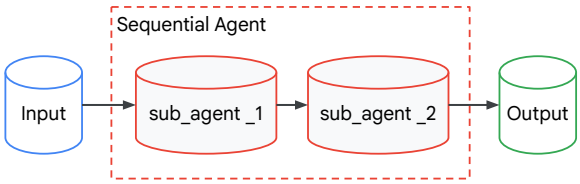
ADK provides a few types of workflow agents, including SequentialAgent, LoopAgent and ParallelAgent. You can also build a custom workflow agent.

Let’s explore these workflow agents in more detail.

Types of workflow agents

SequentialAgent | LoopAgent | ParallelAgent | Custom

- Executes agents in a specific order
- For workflows with a fixed order, when tasks have dependencies requiring sequential execution



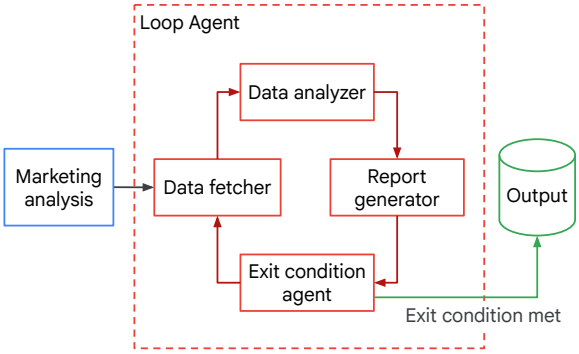
Sequential Agents execute a predefined list of agents one-after-another in a specific order. These are used for workflows with a fixed order, when tasks have dependencies requiring sequential execution.

For example, a workflow to "Process a new order" might sequentially execute: "Order Validation Agent", "Inventory Check Agent", "Payment Processing Agent", and "Order Confirmation Agent."

Types of workflow agents

SequentialAgent | **LoopAgent** | ParallelAgent | Custom

- Executes agents repeatedly, until a certain condition is met
- For tasks that require iteration, continuous monitoring, cyclical processes or simulated negotiation



Loop Agents repeatedly execute a set of agents until a certain condition is met.

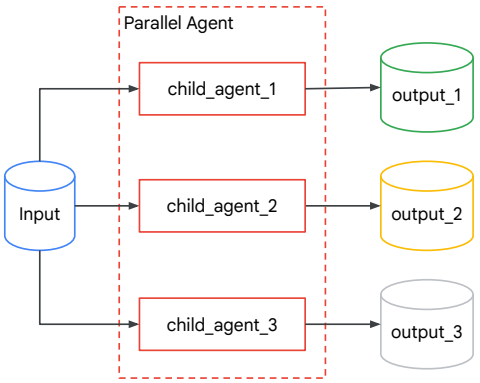
Use the Loop Agent when your workflow involves tasks requiring iterative refinement, continuous monitoring, cyclical processes, or simulated negotiation.

For example, a "Research Agent" might loop through a series of agents to find new data, analyze it, and ask if it has answered a specific question sufficiently or needs to pull more data (repeating the loop).

Types of workflow agents

SequentialAgent | LoopAgent | **ParallelAgent** | Custom

- Executes agents simultaneously
- For tasks with independent sub-tasks, tasks that prioritize speed, or are resource intensive



Parallel Agents can run multiple agents simultaneously. This can significantly speed up tasks that can be broken down into independent sub-tasks.

This approach is particularly beneficial for operations like multi-source data retrieval or heavy computations, where parallelization yields substantial performance gains. Importantly, this strategy assumes no inherent need for shared state, or direct information exchange between the concurrently executing agents.

For example, a Parallel Agent could execute several "Report Generation Agents" in parallel, each responsible for a different region.

Types of workflow agents

SequentialAgent | LoopAgent | ParallelAgent | **Custom**

Use when your requirements include:

- Conditional logic
- Complex state management
- External integrations
- Dynamic agent selection
- Unique workflow patterns

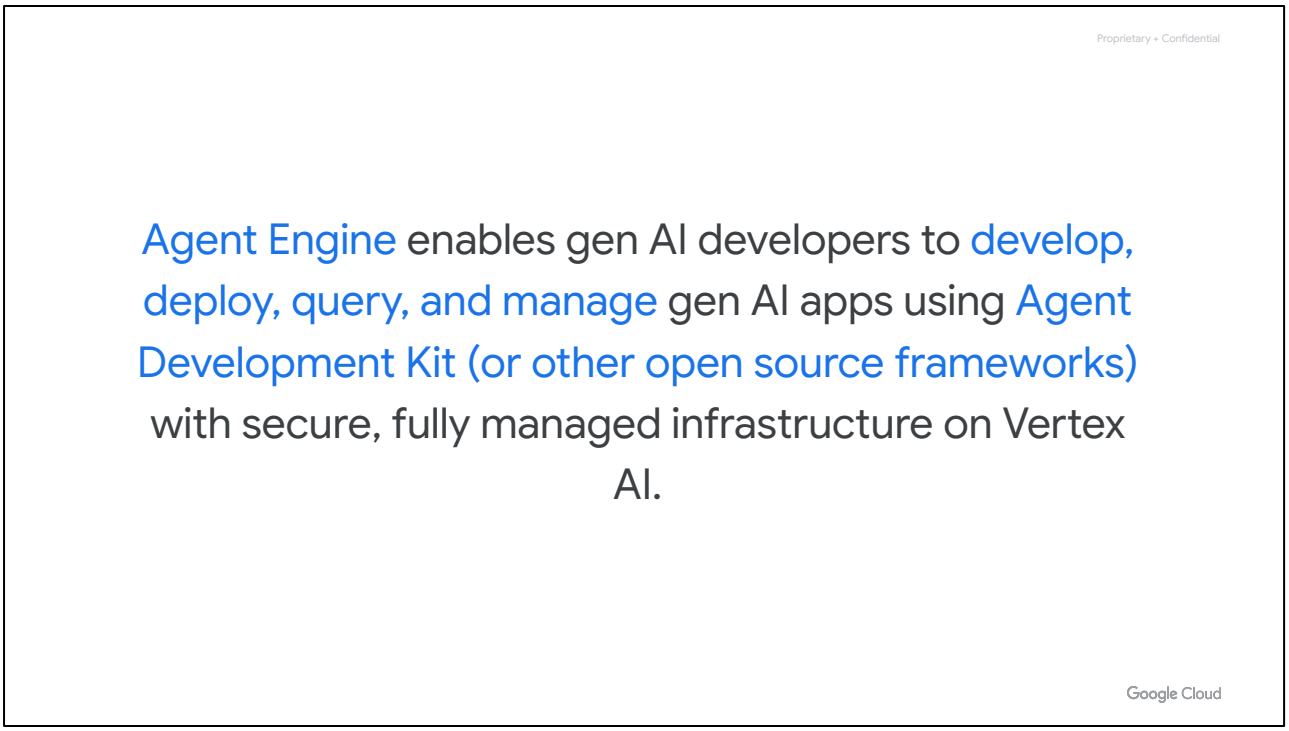
Custom workflow agents in Agent Development Kit allow you to define arbitrary orchestration logic, going beyond the predefined patterns of Sequential, Loop, and Parallel Agents.

This provides maximum flexibility for complex workflows, stateful interactions, and integrating custom business rules.

You'll need a Custom agent when your requirements include

- **Conditional logic:** Executing different sub-agents or taking different paths based on runtime conditions or the results of previous steps.
- **Complex state management:** Implementing intricate logic for maintaining and updating state throughout the workflow beyond simple sequential passing.
- **External integrations:** Incorporating calls to external APIs, databases, or custom libraries directly within the orchestration flow control.
- **Dynamic agent selection:** Choosing which sub-agent(s) to run next based on dynamic evaluation of the situation or input.
- **Unique workflow patterns:** Implementing orchestration logic that doesn't fit the standard sequential, parallel, or loop structures.

Welcome to deploy ADK agents to Agent Engine.



Agent Engine enables Gen AI developers to develop, deploy, query, and manage Gen AI apps using Agent Development Kit (or other open source frameworks) with secure, fully managed infrastructure on Vertex AI.

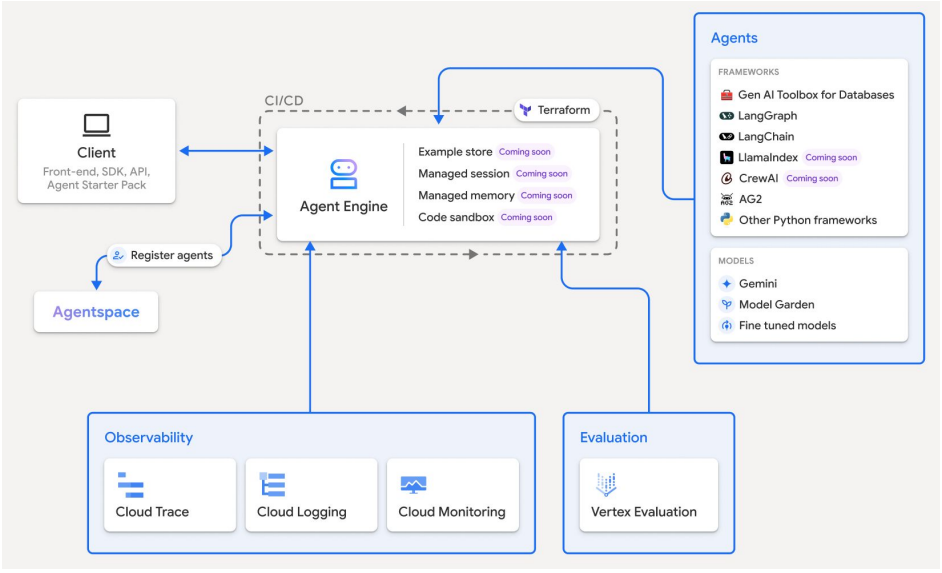
Agent Engine

Theme	Features
Scalable deployment	• Enables reliable and **scalable agent deployments** on Google Cloud • Provides an API to **manage** and **query agents** • Handles **security** (VPC-SC) and **authentication** concerns
Comprehensive agent management	• Offers tools for **session** and **memory management** • **UI** to list, view, and converse with agents • Facilitates agent **monitoring** and **analytics**
Quality and evaluation	• Use **evaluation tools** to measure and understand agent performance quality • Add to and manage the **Example Store** to improve agent performance
Framework agnostic	Supports deployments of agents built on various frameworks: • **First-party** Agent Framework from Vertex AI • **Third-party open source** frameworks such as LangGraph, LangChain, and Crew AI

Agent Engine is a fully managed runtime for developers to deploy and scale agents in production. It includes:

- **Scalable Deployment:** Agent Engine enables reliable and scalable agent deployments on Google Cloud. It provides an API to manage and query agents, and handles security, such as VPC-SC, and authentication concerns.
- **Comprehensive agent management:** Agent Engine offers tools for session and memory management, and a user interface to list, view, and converse with agents. It also facilitates agent monitoring and analytics.
- **Quality and evaluation:** You can use evaluation tools to measure and understand agent performance quality. And add to, and manage the Example Store to improve agent performance.
- **Framework agnostic:** It supports deployments of agents built on various frameworks, including first-party frameworks like Agent Framework from Vertex AI, and third-party Open source frameworks, such as LangGraph, LangChain, and Crew AI.

Agent Engine: The keystone



Agent Engine is the keystone that connects all of the capabilities required to develop, observe, evaluate and deploy your agent solutions.

The process involves developing, monitoring, evaluating and testing agents, registering and deploying them, and then querying them. Agent Engine aims to provide the shortest path to success by reducing friction from various areas like timeout requests, model choices, security, and CICD.

Agent Engine

A unified platform that provides the tools and infrastructure for building, running, and overseeing applications powered by Large Language Models (LLMs). Its value is contained in the four main stages of the app lifecycle:

1. Develop: This stage provides the necessary tools to **build** the agent's logic, capabilities, and personality.
2. Deploy: This stage packages the developed agent and makes it available for use.
3. Query (Run): This stage is the core execution of the agent, where it interacts with users and performs tasks.
4. Manage (Monitor): This stage is about governance, maintenance, and continuous improvement of the live agent.

Virtual Private Cloud - Service Controls (VPC-SC)

A security feature that helps organizations **mitigate data exfiltration risks** by creating a **security perimeter** around sensitive data stored in managed services (like Cloud Storage and BigQuery). It makes a **virtual wall** around specified Cloud projects and services.

- **Perimeter:** This boundary, called a **Service Perimeter**, allows free and secure communication **within it but, by default, blocks all data movement to or from restricted services outside** the perimeter.
- **Defense-in-Depth:** VPC-SC complements Identity and Access Management (IAM). While **IAM** defines *who* can access a resource, **VPC-SC** defines *where* they can access it from and **prevents data from leaving** the trusted network boundary, even if credentials are stolen or IAM policies are misconfigured.
- **Control:** It provides fine-grained control over API access, restricting access based on factors such as source **IP**, user **identity**, or **trusted device**.

Agent Framework from Vertex AI

- The **Agent Framework** is Google Cloud's comprehensive, end-to-end platform for creating, managing, and scaling sophisticated AI Agents.
- It is separated into two main, complementary parts:
    - Agent Development Kit (The Framework/SDK)
    - Vertex AI Agent Engine (The Managed Platform)

The **ADK** lets you *build* the agent locally, while the **Agent Engine** provides the infrastructure to *run* it reliably at enterprise scale. They are designed for seamless integration: you build with **ADK** and deploy to the **Agent Engine**.

Agentspace

A centralized platform (now part of **Gemini Enterprise**) designed to be the "front door for AI" in an enterprise environment. It serves as a catalog for end users (employees) to interact with deployed AI agents.

Register Agents

Registering an agent is the specific action a developer takes to make a deployed agent available within Agentspace. The workflow is:

1. **Develop:** An agent is created using an orchestration framework, such as the **ADK**.
2. **Deploy:** The agent is deployed to a production runtime, specifically the **Vertex AI Agent Engine**.
3. **Register:** Once deployed and running on the Agent Engine, the agent is formally **registered** with Agentspace. This step effectively publishes the agent to the Agentspace **Agent Gallery** (or catalog).

The registration step is critical for moving a developer-built agent from a deployment environment (Agent Engine) to a discoverable, usable product for the rest of the enterprise.

Agent Starter Pack

A Google Cloud resource designed to significantly **accelerate the development and deployment** of AI agents by providing developers with pre-built code, templates, and automated pipelines. It addresses the common challenges of moving an AI agent from a prototype to a production environment. Its key features typically include:

- **Pre-built Templates:** Starter code and templates for common agent use cases.
- **Infrastructure Automation:** Automated scripts and configuration files (e.g., for Terraform) to provision the necessary cloud infrastructure.
- **Testing and Observability:** Pre-configured tools and setups for **AgentOps** (MLOps for agents), testing methodologies, monitoring, and debugging (e.g., using Cloud Trace).
- **Best Practices:** It incorporates recommended architectural patterns for building secure, scalable, and reliable enterprise-grade agents.

Some AI Agent Frameworks

1. AG2 (Conversational Orchestration)

AG2 (built on the foundation of AutoGen) treats agentic workflow as an **asynchronous conversation** among specialized agents. Agents pass messages back and forth in an event-driven loop. This design is highly flexible and well-suited to scenarios where agents need to work concurrently or wait on external events (like a human's input or a tool's result).

2. CrewAI (Role-Based Collaboration)

CrewAI is designed around the metaphor of a **human team**, focusing on **roles, goals, and collaboration**.

- **Role-Based:** You explicitly define agents with distinct roles (e.g., "Senior Researcher," "Editor," "Analyst") and give them a clear goal and backstory. This simplifies the prompt engineering required for collaboration.
- **Structured Workflow:** It uses a high-level abstraction called a **"Crew"** to manage the workflow. The most common process is sequential, where agents complete tasks one after the other, with context passed between them. This approach is highly **deterministic** and highly auditable.
- **Focus:** It is ideal for quickly prototyping and deploying collaborative multi-agent systems where task delegation, clear accountability, and structured outputs (like final reports or summaries) are key.

3. LlamaIndex (Data-Centric Framework)

LlamaIndex's primary strength is **Retrieval-Augmented Generation (RAG)**, connecting LLMs to custom data sources. Its agents are an extension of this core capability.

- **Data First:** LlamaIndex agents are best used when the task heavily involves retrieving, synthesizing, or reasoning over large, private, or complex enterprise datasets (PDFs, databases, etc.). The agents use LlamaIndex's various indexing and retrieval tools as their primary capability.
- **Agent as Query Router:** In multi-agent scenarios, LlamaIndex often employs an **Orchestrator Agent** that decides which sub-agent is best suited to answer a query. This is fundamentally a data-routing paradigm.
- **Integration:** LlamaIndex components are often used **as tools** *within* other orchestration frameworks, such as CrewAI or AG2, to give those agents powerful data access capabilities.

Develop and deploy agents with Agent Engine

Develop and test → Deploy → Query

Let’s take a look at how to develop and deploy agents with Agent Engine.

Develop and deploy agents with Agent Engine

Develop and test → Deploy → Query

```
model = “gemini-2.0-flash”

def get_exchange_rate():
    "Retrieves the exchange rate."
    ...

agent = reasoning_engines.LangchainAgent(
    model=model,
    tools=[get_exchange_rate],
    }
)
```


You need to first define the model, tools, and agent, before testing.

Develop and deploy agents with Agent Engine

Develop and test → Deploy → Query

```
remote_agent = agent_engines.create(
    agent,
    display_name="my_agent",
    requirements=[
      "google-cloud-aiplatform",
      "langchain"])
```

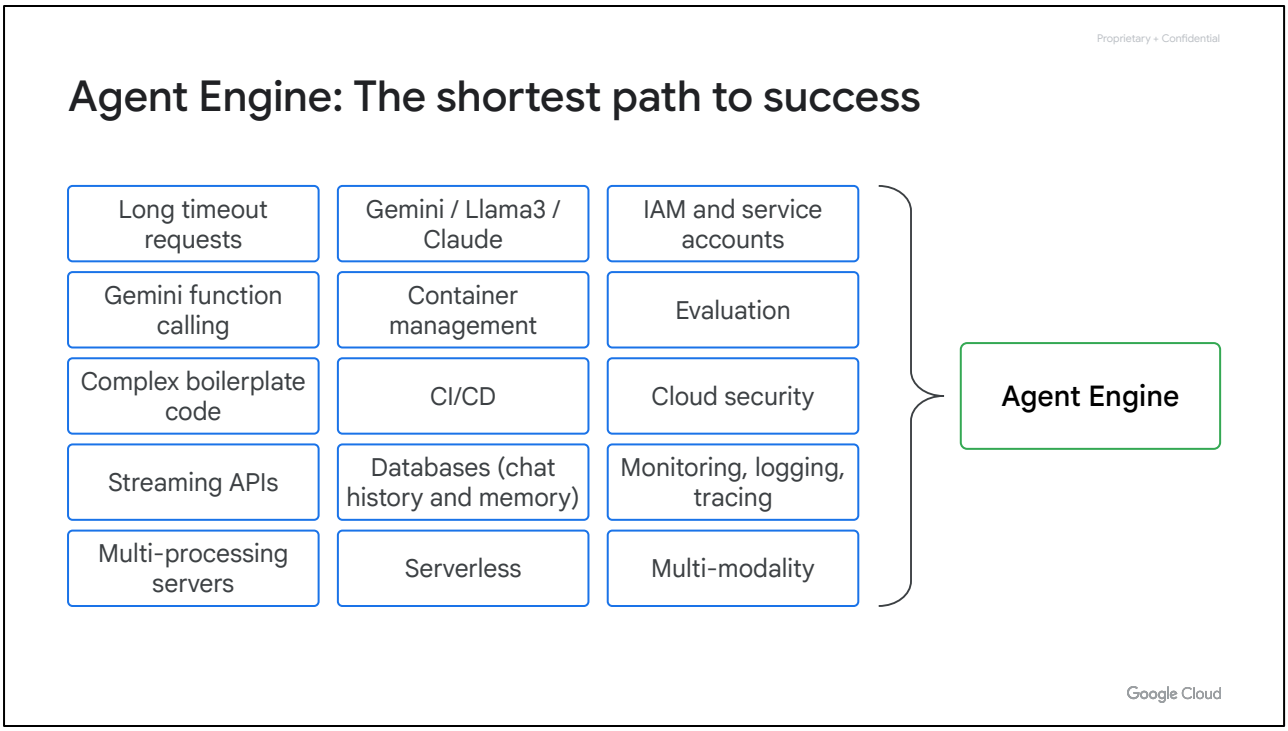

Next, you deploy your agent to a remote app.

Develop and deploy agents with Agent Engine

Develop and test → Deploy → Query

```
remote_agent.query(
  query="What's the exchange rate from US
  dollars to Swedish currency?")
```


And finally, query the agent through your app.



After all of the work done to develop and evaluate an agent, you face another round of decisions and testing around infrastructure decisions like runtime scaling, network infrastructure and security, monitoring and tracing, etc. Agent Engine provides the shortest path to success by handling runtime and network infrastructure decisions for you, enabling you to focus on building the best agent while Agent Engine handles the deployment environment.

Timeout

In networking, a **timeout** is a mechanism that prevents resources from being tied up indefinitely.

- When a request is sent, the client or a proxy starts a **timer**.
- If the server does not send a response before the timer reaches the **timeout limit**, the connection is typically closed, and the client receives a **timeout error**.

Long Timeout Request

- A "long timeout request" is one that either completes successfully just barely before this limit, or is the specific request that hits the long limit.
- This is usually a problem because it can lead to poor user experience, resource exhaustion, and application instability.

Gemini Function Calling

A feature that allows the Gemini model to intelligently decide when to execute an **external function or API** provided by the developer, and with what **parameters**, to fulfill a user's request. In short, it connects the LLM's natural language understanding to the **real-world action capabilities** of your application code. The process involves these key steps:

1. **Declaration:** The developer defines the function's **name, purpose, and parameters** using a structured format (JSON) and passes this "declaration" to Gemini.
2. **Prediction:** When a user asks a question (e.g., "What's the weather in Paris?"), Gemini analyzes the prompt against the available function declarations.
3. **Structured Output:** If the function is needed, Gemini does **not** execute the code. Instead, it returns a **structured JSON object** specifying the function name (e.g., `get_weather`) and the required arguments (e.g., `location: "Paris"`).
4. **Execution and Response:** The developer's application code takes that JSON, runs the actual function (e.g., calling a weather API), and sends the result back to Gemini. Gemini then uses that real-world information to formulate a final, natural language answer for the user.

This technique is crucial for building AI assistants capable of performing real-world tasks, such as retrieving up-to-date information, managing databases, and interacting with smart devices.

Multi-processing Server

A type of server architecture designed to **handle multiple client requests concurrently** by using multiple independent operating system processes. It achieves concurrency by **forking** (creating copies of) its own process, where each child process handles a single or a few client connections. This is a crucial distinction from multi-threading, where concurrency is achieved within a single process.