Google Cloud

Startup technical guide

AI agents

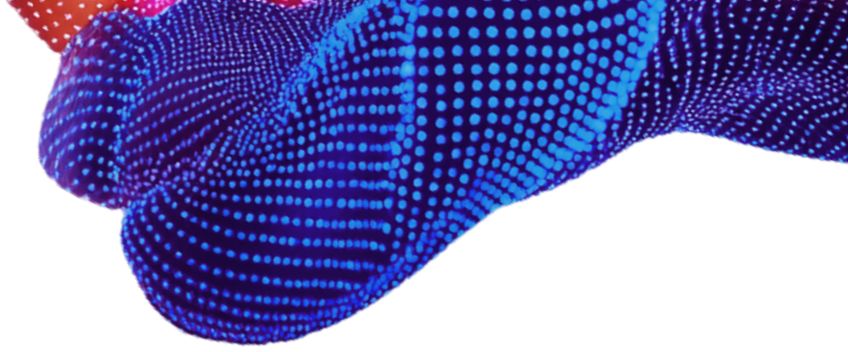


Table of contents

Introduction

The development of AI agents represents a paradigm shift in software engineering, enabling startups to automate complex workflows, create novel user experiences, and solve business problems that were previously technically infeasible.

But moving from a promising prototype to a production-ready agent means solving a new set of challenges. How do you manage their non-deterministic behavior? How do you verify their complex reasoning paths? And, crucially, where do you get started?

This technical guide will help answer questions like these. It provides a systematic, operations-driven roadmap for navigating the new landscape, and is geared to help startups and developers who are racing to embrace the potential of agentic systems.

You'll learn the foundational concepts of agentic systems, from their core architectural components to the principles that ensure reliable and responsible operation in production. And you'll learn about the full spectrum of tools that make building and using agents on Google Cloud more efficient, from code-first development with Agent Development Kit (ADK) and operational automation with the Agent Starter Pack, to no-code agent creation with Google Agentspace.

Whether you're validating an idea, building an MVP, or supporting a product in production, this guide will help across all stages of your project.

How to use this guide

New to AI agents?

Start with Section 1 for the core concepts.

Ready to build?

Jump to Section 2 to create your first agent using ADK.

Agent built?

Dive into Section 3 to make it safe, stable, and scalable.

Want extra support?

Use the Gemini Kit to prototype faster, and apply to the Google for Startups Cloud Program to receive expert guidance and up to $350k USD in cloud credits.

The focus of this guide

The agentic AI ecosystem offers many tools, libraries, and approaches for building cognitive architectures. There are open-source frameworks from Google like Genkit and Google Cloud's conversational AI offerings, as well as popular open-source libraries like LangChain and CrewAI.

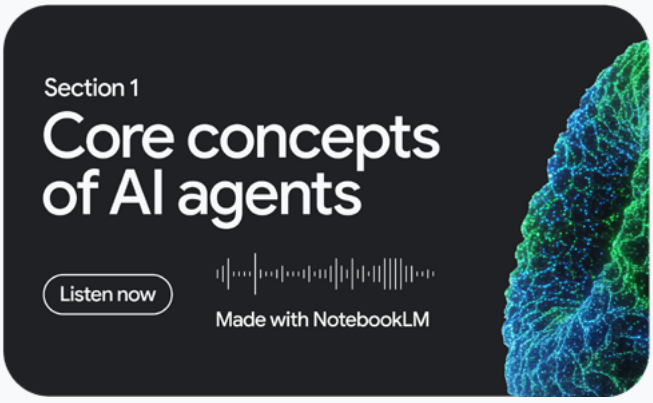
This guide focuses primarily on ADK, sharing concepts and architectural patterns that allow you to build robust, scalable agents on Google Cloud while retaining the ability to integrate other preferred tools and libraries.

Section 1

Core concepts of AI agents

The field of agentic AI is evolving rapidly. This section provides foundational knowledge on AI agents, explaining their core concepts, purpose, and operational mechanics. It also details the relevant tools and services available within Google Cloud.

Prefer audio? Listen to the podcast version of this section, created with NotebookLM.



This podcast was created using NotebookLM with the prompt: "As a podcast host, create a conversational and educational podcast for 'Startup technical guide: AI agents,' targeting a technical audience of startup founders and developers. The podcast must cover the three main paths for using AI agents (build, use, partner), detailing tools like the Agent Development Kit (ADK) and pre-built Gemini agents.

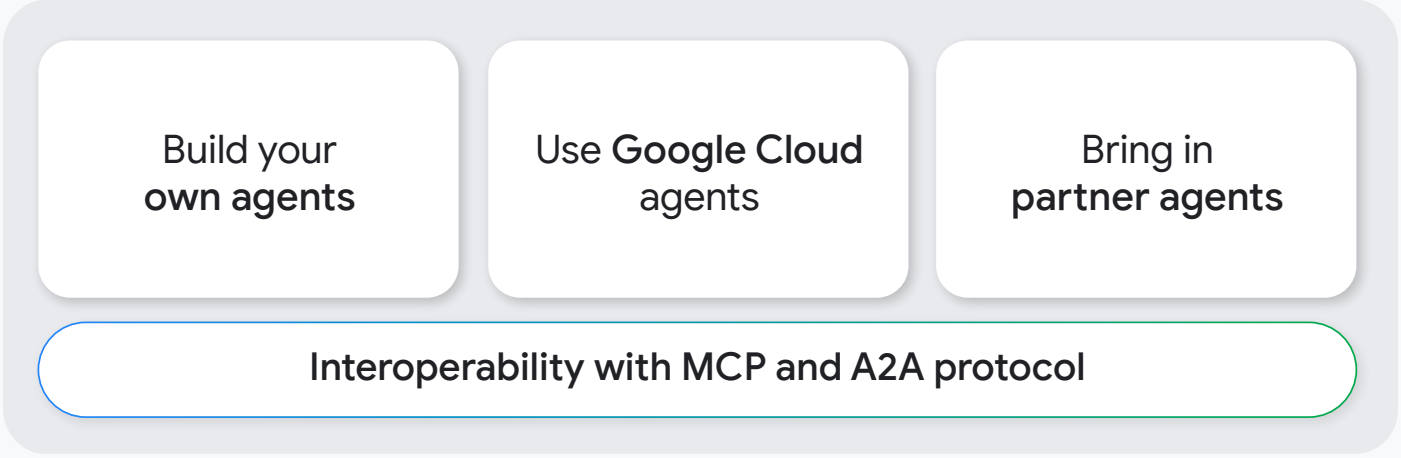
"It should then explain the core components of an agent, including models, tools, orchestration, and runtime. Also, cover how to ensure trust and power through techniques like grounding with Retrieval-Augmented Generation (RAG) and leveraging multimodality. Conclude with a summary of the key takeaways and a clear call to action directing listeners to Google's resources."

1.1 An overview of Google Cloud's agent ecosystem

> **The agentive workflow is the next frontier. It's not just about asking a question and getting an answer. It's about giving AI a complex goal—like 'plan this product launch' or 'resolve this supply chain disruption'—and having it orchestrate the multi-step tasks needed to achieve it. This will fundamentally change productivity."**
>
> **Thomas Kurian**
> CEO of Google Cloud

Building production-grade AI agents requires more than selecting a large language model. A complete solution demands scalable infrastructure, robust data integration tooling, and architectural patterns that accommodate diverse technical requirements.

Google Cloud supports the comprehensive development of agentic systems, whether you're building your own agents, using pre-built Google Cloud agents, or bringing in partner agents. Underpinned by the Model Context Protocol (MCP) and Agent2Agent (A2A) protocol, this common framework is designed for interoperability. This way, regardless of their origin or architecture, your agents can collaborate within the Google Cloud ecosystem.[1]



1. MCP and A2A protocol are covered in depth in section 2 of this guide.

Build your own agents

If you're looking to build custom agents geared to tackle specific tasks, then this is the route for you. Here, you've got two options: a code-first approach for maximum control or an application-first approach for accelerated development.

Agent Development Kit for custom, code-first development

This approach is best for developers, technical startups, and teams that require a high degree of control over agent behavior. Google Cloud's Agent Development Kit (ADK) is built for this custom approach.

ADK empowers developers to build, manage, evaluate, and deploy AI-powered agents. It provides a robust and flexible environment for creating both conversational and non-conversational agents, capable of handling complex tasks and workflows.

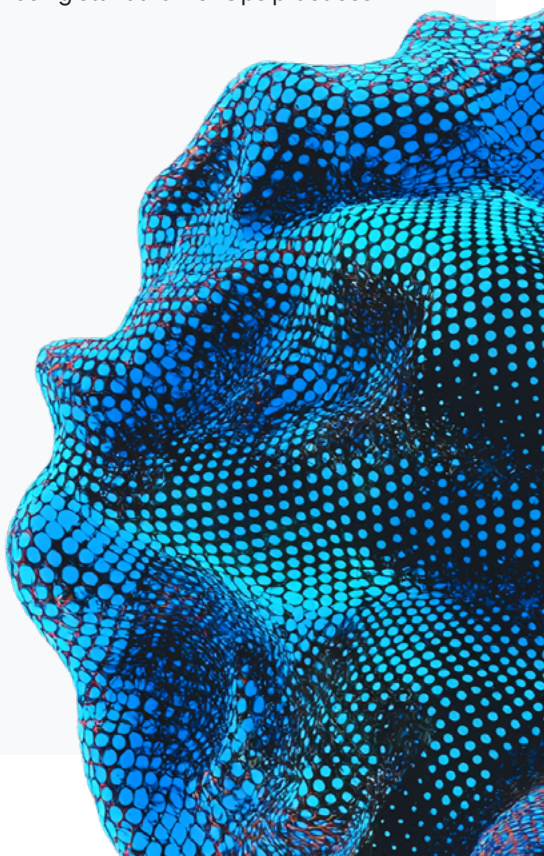
Agents built with ADK can easily be deployed on Vertex AI Agent Engine, a managed, scalable environment designed specifically for this purpose. Because these agents are containerized, they can also be deployed to any environment that runs containers, such as Cloud Run and Google Kubernetes Engine (GKE).

Core capabilities

- **Orchestration logic:** The agent's core reasoning process, like the ReAct framework (see section 1.2), allows it to plan and execute a sequence of tool calls and actions to accomplish a complex goal.
- **Tool definition and registration:** An interface for defining custom functions and APIs, allowing the agent to interact with data, APIs, and external systems.
- **Context management:** A system that provides the agent with memory, allowing you to use the agent to recall user preferences and conversational history across multiple interactions to provide a coherent experience.
- **Evaluation and observability:** A suite of built-in tools to rigorously test agent quality, debug the agent's step-by-step reasoning, and monitor its performance in a production environment.
- **Containerization:** The capability to package the agent into a standard, portable container, making it ready for deployment on any compatible cloud environment.
- **Multi-agent composition:** The ability to build systems where multiple specialized agents can collaborate, delegate tasks, and work together to solve a problem.

Why it matters for startups

- **Automate workflows, not just conversations:** Implement multi-step orchestration logic to solve complex business problems, creating the operational leverage a small team needs to scale.
- **Build a defensible product:** Connect agents directly to your proprietary APIs and internal data to create a product with a real competitive moat.
- **Remember your customers to deliver a truly personal experience:** Seamlessly integrate short-term conversational context with long-term knowledge, enabling you to have your agent recall past interactions and build a true customer relationship.
- **Launch with confidence:** Leverage built-in evaluation and observability to rigorously test and debug your agent, ensuring you ship a reliable, production-grade product.
- **Focus on your product, not infrastructure:** Package your agent into a standard container for a faster, more reliable path to production using standard DevOps practices.



Google Agentspace for application-first development

The second primary pathway for building is through Google Agentspace. Unlike the code-first ADK, you can use Google Agentspace to orchestrate your entire AI workforce and empower non-technical team members to build custom agents using a no-code designer.

This platform-based approach is ideal for managing multiple agents and scaling their use across your mature startup's growing cohort of SaaS applications.

Core capabilities

- **Unified company-wide search:** Connects to and searches across multiple SaaS applications.
- **Multimodal data synthesis:** Understands and synthesizes information from text, images, charts, and video while respecting data permissions.
- **Pre-built agent library:** Provides a suite of ready-to-use agents for complex tasks like deep research or idea generation.
- **No-code custom agent builder:** Includes Agent Designer, which allows non-technical users to create agents via a prompt-driven interface.

Why it matters for startups

- **Break down data silos:** Non-developer teams can build and deploy agents that can access and act across these fragmented data sources and applications.
- **Automate workflows:** Create cross-platform workflows without consuming scarce engineering resources, freeing up your engineering team to focus on core product development.

> **Making Gemini a world model is a critical step in developing a new, more general, and more useful kind of AI—a universal AI assistant. This is an AI that's intelligent, understands the context you are in, and that can plan and take action on your behalf, across any device."**
>
> **Demis Hassabis**
> CEO of Google DeepMind

Use Google Cloud agents

With rapid prototyping and easy ways to integrate AI into your existing apps, managed agents let you focus on core business logic rather than managing infrastructure. They're also ideal if your engineering resources are limited.

Gemini Code Assist

Gemini Code Assist is an AI-powered assistant for developers. It integrates into multiple points of the software development lifecycle, providing assistance through IDE extensions, a command-line interface, GitHub integration, and within various Google Cloud services.

Core capabilities

- **IDE integration:** Within popular IDEs (VS Code, JetBrains IDEs, Android Studio), it provides code completion, on-demand function generation, and a chat interface. It uses Gemini's large context window to provide responses relevant to the open codebase. Enterprise editions can be connected to private source code repositories for more customized suggestions.
- **Command-line interface:** Gemini CLI is an open-source AI agent that brings Gemini capabilities directly to the terminal for tasks such as code understanding, file manipulation, and dynamic troubleshooting.
- **GitHub integration:** On GitHub, Gemini Code Assist can automatically review pull requests to identify bugs and style issues, suggesting specific code changes.
- **Agent-driven development:** Deploys AI agents capable of performing complex, multi-file edits across a full project's context. These agentic workflows incorporate Human in the Loop (HITL) oversight and can integrate with ecosystem tools that follow the MCP.
- **Google Cloud service integration:** Provides AI assistance directly within services like Firebase (app error analysis, performance insights), Colab Enterprise (Python code generation), BigQuery (natural language to SQL, query optimization), Cloud Run, and Apigee.

Why it matters for startups

Gemini Code Assist acts as a force multiplier. It can handle software development tasks across the development lifecycle, from routine tasks like writing boilerplate code to more complex operations like multi-file refactoring.

You can delegate a wide range of tasks to Gemini Code Assist. Here are a few examples that show its capabilities.

- **For automating boilerplate:** Generate a Python Cloud Function that triggers on an HTTP request. It should parse a JSON payload for a userId and documentId, then use the google-cloud-firestore client library to fetch a specific document from a 'users' collection and return it as a JSON response.
- **For comprehensive testing:** Provide one of your existing functions and ask Code Assist to generate a complete test suite, including the necessary mocks for Google Cloud services like Cloud Storage or Firestore.
- **For large-scale, Gemini-driven refactoring:** Ask it to analyze multiple services across your codebase and generate a strategic plan. For example: "Given our 'user-service' and 'auth-service', propose a step-by-step plan to refactor the authentication logic into a single, shared library, outlining the trade-offs of this approach."

Gemini Cloud Assist

Gemini Cloud Assist is an AI expert for your Google Cloud environment, providing context-aware assistance for infrastructure management and application operations. It uses context from your project, including Google Cloud project IDs and the specific product page being viewed in the console, to tailor its support.[2]

Core capabilities

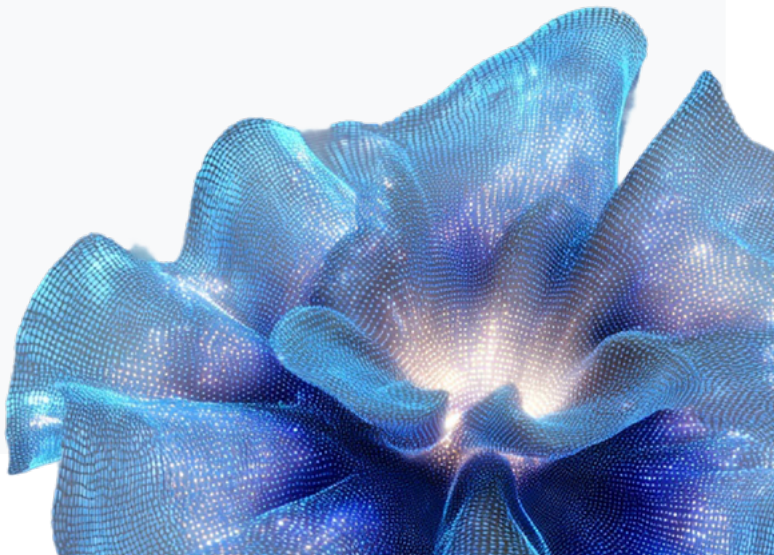
- **Design and deploy:** Within the Application Design Center, you can describe desired infrastructure outcomes in natural language. Gemini Cloud Assist generates architecture diagrams and application templates, which can be exported as Terraform for integration with existing Infrastructure as Code (IaC) workflows.
- **Troubleshoot and resolve:** Integrates with Cloud Observability to summarize complex log entries and explain error messages. For deeper issues, you can launch investigations, where Gemini analyzes logs and metrics to identify the root cause.
- **Configure and optimize:** Provides personalized cost and utilization recommendations within the FinOps Hub as well as the Cost Optimization dashboard.
- **Secure and analyze:** Enables natural language investigation of network flows and logs. It provides guidance on security tasks such as data encryption, secrets management, and generating or testing custom organization policies. It can also recommend IAM roles and diagnose permission errors.

Why it matters for startups

- **Free up time:** Cloud management can eat up engineering time. Gemini Cloud Assist frees you up to focus on building your product.

Try these prompts in Gemini Cloud Assist:

> How do I use Vertex AI to deploy a model?
>
> Create a high-level plan for designing, building, and deploying a web app in Google Cloud.
>
> List all Cloud Storage buckets in the prod-v1 project that do not have Object Versioning enabled.
>
> What are the public-facing firewall rules applied to instances with the network tag external-web-server?
>
> Show me all IAM roles granted to the service account data-pipeline@my-project.iam.gserviceaccount.com



2. For details on how Gemini Cloud Assist is grounded, see the official documentation.

Gemini in Colab Enterprise

If your startup is working in data science, machine learning, or analytics, Gemini in Colab Enterprise turns every notebook into a collaborative AI workspace. It's built to generate, explain, and debug Python code all in context.

Core capabilities

- Autocomplete and generate Python code within Colab.
- Explain code logic and errors in simple language.
- Filter, transform, and visualize data.
- Recommend public datasets and research resources.
- Summarize entire notebooks or code cells.

Try these prompts in Gemini in Colab Enterprise:

> How do I filter a Pandas DataFrame?
>
> Plot average revenue by region.
>
> Show me a list of publicly available datasets for climate tech.
>
> Summarize the goal of this notebook.

Why it matters for startups

- **Accelerate research and development:** Automate the most tedious aspects of data preparation, analysis, and visualization, allowing developers to iterate on new models and ideas significantly faster.
- **Lower the barrier to entry:** Engineers new to data science can hit the ground running, while experienced practitioners can focus more on model experimentation and less on data wrangling.

Bring in partner agents

If your use case is more specialized, you can easily integrate third-party or open-source agents into your stack using Google Cloud's open ecosystem and via the Google Cloud Marketplace.

Explore Agent Garden to deploy pre-built ADK agents that already support data reasoning and inter-agent collaboration. You can mix and match them with the agents you build, speeding up time to impact.

1.2 Key components of every agent

Models: Selection and tuning

Think of the model as your agent's brain. You can use the model to read user requests, figure out what needs to happen, and generate smart responses.

How to choose the right model

Choosing the right model is not about selecting the most powerful one available, but about finding the optimal balance of capability, speed, and cost for your use case. Every model can be evaluated on these three conflicting characteristics, and the goal is to identify the most efficient option for a specific job.

As a model's capability increases, its cost and latency generally increase as well. The most common mistake is over-investing in capability when a use case doesn't need it, leading to inefficient spending and slower performance. The optimal strategy is to select the most efficient model for any given task.

This principle is most powerfully applied at a system level. Robust cognitive architectures employ multiple specialized agents, each dynamically selecting the leanest model for its specific sub-task. This ensures, for instance, that a heavyweight model is reserved for complex reasoning, while a lightweight model handles routine queries. This multi-agent approach provides the architectural flexibility to optimize the cost and performance of the entire system, not just a single component.

> **AI agents are systems that combine the intelligence of advanced AI models with access to tools so they can take actions on your behalf, under your control."**
>
> **Sundar Pichai**
> CEO of Google and Alphabet

Use cases

Early-stage prototyping and at-scale tasks

- **Model profile:** A lightweight, low-cost model like Gemini 2.5 Flash-Lite.
- **Rationale:** This is the most cost-efficient and fastest 2.5 model, excelling at high-volume, latency-sensitive tasks like translation and classification.

High-volume, high-quality applications

- **Model profile:** A balanced, mid-range model like Gemini 2.5 Flash.
- **Rationale:** This model is designed to control the trade-off between quality, cost, and speed. It delivers strong performance on complex tasks at a lower price point than Pro, making it perfect for production applications that need to be both smart and economical.

Complex, multi-step reasoning and frontier code generation

- **Model profile:** An advanced reasoning model like Gemini 3 Pro.
- **Rationale:** Our most intelligent model, for multimodal understanding, and our most powerful agentic and vibe-coding model.[3]

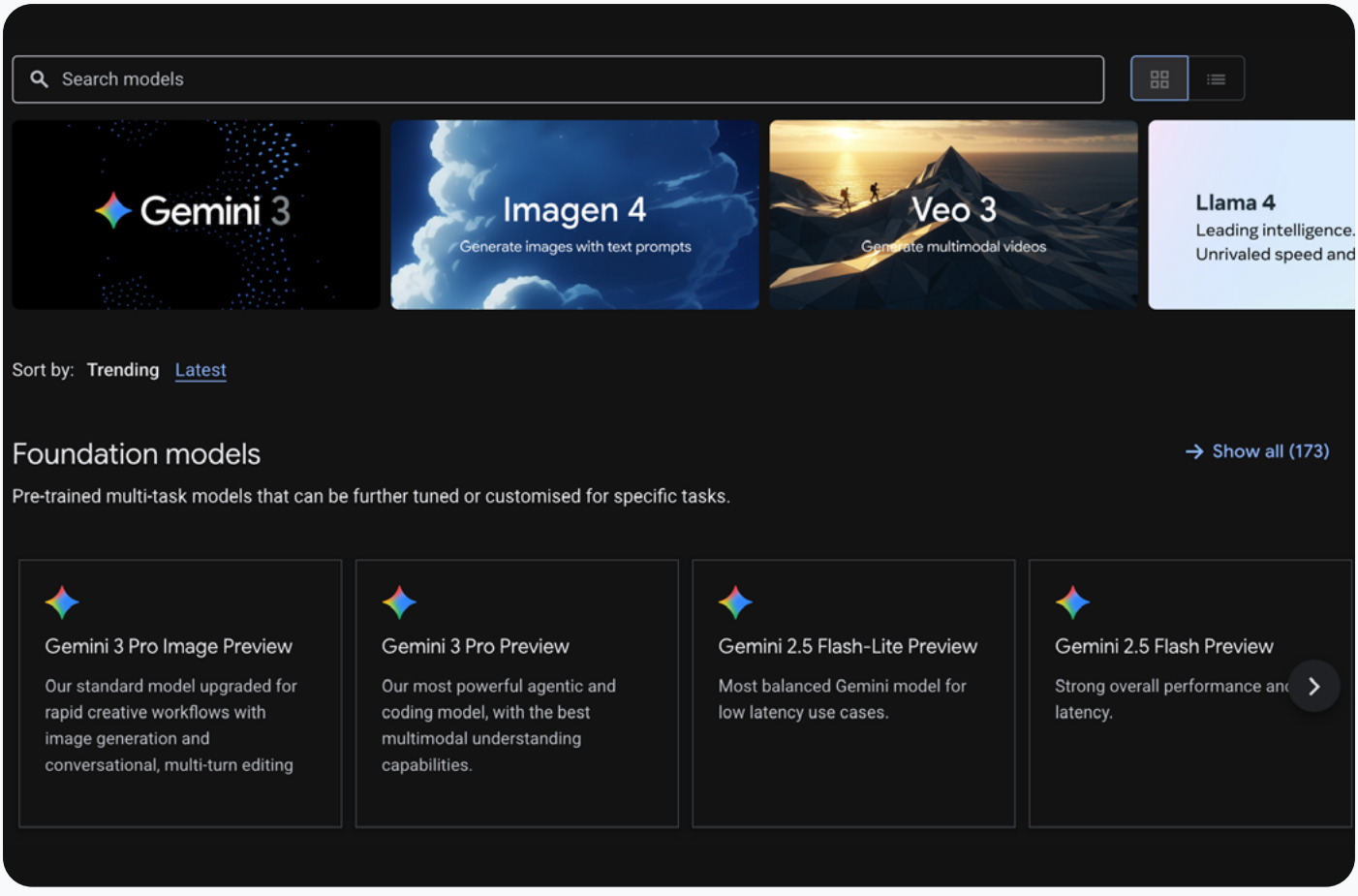
3. Gemini 3 Pro tops the LMArena Leaderboard with a breakthrough score of 1501 Elo. It demonstrates PhD-level reasoning with top scores on Humanity's Last Exam (37.5% without the usage of any tools) and GPQA Diamond (91.9%). It also sets a new standard for frontier models in mathematics, achieving a new state-of-the-art of 23.4% on MathArena Apex. Results as of November 2025.

You can use the Gemini family of models to break down problems, formulate plans, and use tools. This reasoning process is configurable. By allocating more reasoning tokens to a specific call, a developer can direct the model to expend more computational effort, directly trading a predictable increase in latency and cost for a potential increase in accuracy.

This token-level control, combined with model selection and configurable reasoning modes, gives developers a dynamic set of levers for sophisticated optimization. The cost and performance of an entire multi-agent system can be calibrated to meet specific business and technical requirements.

Pro tip

Use Model Garden on Vertex AI to discover, customize, and deploy foundation models from a single, centralized platform. It provides a curated selection of over 200 models from Google, partners like Anthropic, and a wide variety of open models from providers like Meta (the Llama family) and Mistral. Instead of manually managing infrastructure, you can deploy models to applications with a single click and scale them using built-in, end-to-end MLOps capabilities.



Model tuning

Once you select a model that fits your cost-latency-quality needs, you may have the option to fine-tune it. This specializes its knowledge and style for your specific business needs, and is done using a curated dataset of your own high-quality examples.

The availability of fine-tuning is determined on a model-by-model basis. Within Google's model portfolio, this capability is supported for the Gemma family of open-weight models and for specific versions of Gemini. It is important to review each model's documentation and license agreement to confirm that fine-tuning is both permitted and technically supported.

Use case

Fine-tuning a support agent

Say you're building a customer support agent for your SaaS product. You could fine-tune on a dataset of thousands of past support tickets and their ideal resolutions to help the model learn about common issues and respond in a voice that aligns with your support team's style.

Pro tip

To see which models can be fine-tuned on Vertex AI, check the official documentation.

Note

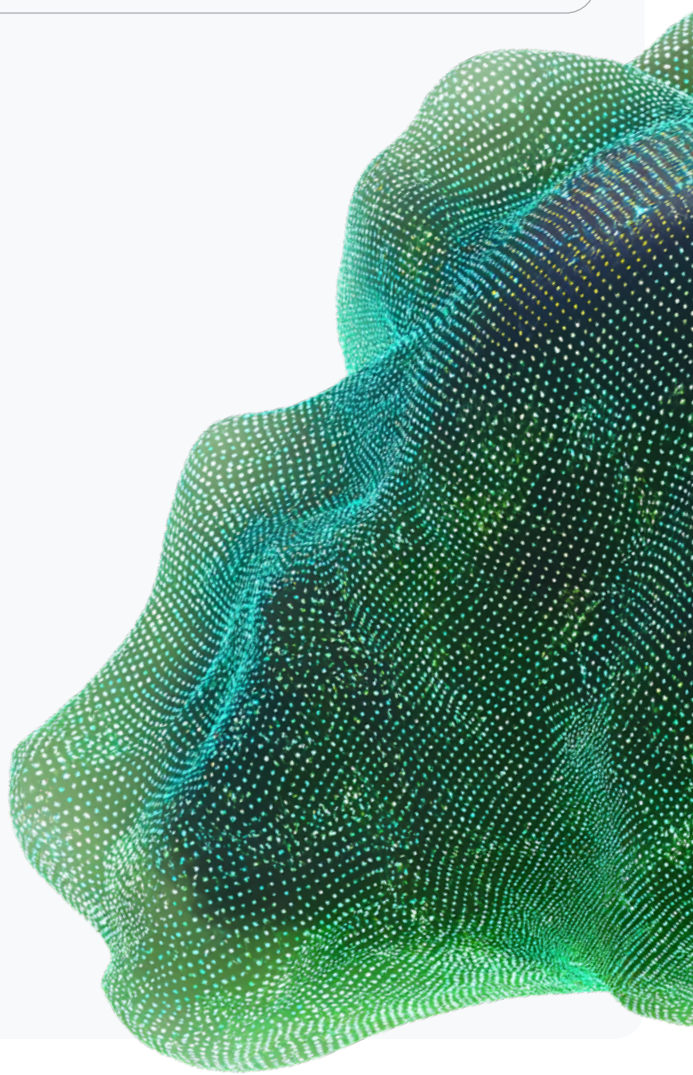
Fine-tuning is not grounding. Fine-tuning adapts a model's style and refines its knowledge on a specific task. Grounding connects the model to real-time, verifiable data sources to ensure its responses are factually accurate. Model grounding is discussed in detail below.

Tools: Enabling agentic action

Tools are defined capabilities that enable an agent to do more than the native functions of its core reasoning model, from performing a simple, internal calculation to interacting with external systems via API calls. They bridge the gap between the agent's reasoning and its ability to retrieve new information or execute stateful operations.

Tools can include a wide variety of components:

- **Internal functions and services:** Proprietary logic written by your own team.
- **APIs:** Connections to both internal services and external, third-party services.
- **Data sources:** The ability to query databases, vector stores, or other repositories of information.
- **Other agents:** In a multi-agent system, one agent can use another specialized agent as a tool.



Data architecture for agentic systems

Data serves as the basis for an agent's short-term and long-term memory. A robust data architecture must address three distinct needs: persistent storage for long-term knowledge retrieval, low-latency access for short-term conversational context, and a durable ledger for transactional auditing. By mapping specific Google Cloud services to each of these needs, you can ensure that every architectural decision is both cost-effective and scalable, while addressing immediate business needs and time-to-market goals.

1. Long-term knowledge base (grounding and retrieval)

An agent's long-term memory is the foundation for its intelligence, grounding, and personalization. It is distinct from the fast, short-term context of a live conversation. A robust long-term memory architecture should comprise three core components: a structured knowledge base for fact-based grounding via retrieval-augmented generation (RAG); a persistent store for user interaction history to enable a continuous, personalized experience; and, an operational data lake for raw material like conversation transcripts and workflow states, for more complex cognitive processes and future analytics.

Data service	Overview	Startup use cases
Vertex AI Search	A managed service for building high-performance vector search applications. It is the primary tool for enabling semantic understanding and retrieval over large, unstructured datasets.	Instantly find answers within your internal product documentation, customer support chat logs, and community forum posts, so your agent can provide accurate, context-aware support to new users. This reduces the burden on your small support team.
Firestore	A serverless NoSQL document database with real-time synchronization capabilities. Its flexible, hierarchical data model is well-suited for storing structured context and the dynamic state of an agent's long-running or durable state active tasks.	Maintain the real-time state of a multi-step, agent-guided user onboarding flow. As the user completes each step (e.g., "create profile," "connect API," "invite team member"), the agent updates a Firestore document. Developers can then observe the agent's task progress in real-time, and the user can seamlessly resume the process across sessions.
Vertex AI Memory Bank (Preview)	A managed service on Vertex AI Agent Engine specifically designed to dynamically generate, store, and retrieve long-term memories from user conversations.	Instead of manually building logic to extract user preferences, an agent can automatically call GenerateMemories on a conversation history. This asynchronously extracts key facts (e.g., "user prefers non-stop flights," "user's dog is named Fido") and stores them. In future sessions, the agent can retrieve these memories via similarity search to provide a deeply personalized and continuous experience with minimal custom code.
Cloud Storage	A highly scalable and durable object store for raw, unstructured source data (e.g., PDFs, images, videos) that feeds into other services for indexing and processing.	It serves as the durable, low-cost landing zone for all user-uploaded documents, images of bug reports, or audio recordings of customer feedback calls. This raw data is then processed and indexed by services like Vertex AI Search to enrich your agent's knowledge.
BigQuery	A fully-managed, serverless data warehouse for storing and analyzing massive structured and semi-structured datasets, so agents can be equipped with tools that execute complex analytical queries.	An agent can ask questions like, "Summarize user engagement patterns for the new feature we launched last week," or "Which customer cohorts have the highest churn risk based on recent activity?" BigQuery provides instant business intelligence.

2. Working memory (conversational context and short-term state)

This layer manages the transient information required for an ongoing task or conversation. It must provide extremely low-latency access to maintain a responsive user experience.

Data service	Overview	Startup use cases
Memorystore	A fully managed, in-memory data store providing sub-millisecond latency. It is ideal for caching frequently accessed data and managing session state.	Its primary role is high-speed caching to store the results of any computationally expensive or high-latency operation. Instead of repeatedly executing a costly task such as an LLM API call, a complex database query, or a call to a third-party service, the agent first checks Memorystore for a cached result. It drastically reduces both response latency and recurring operational costs, both critical to a startup's agentic system.

3. Transactional memory (state management and action auditing)

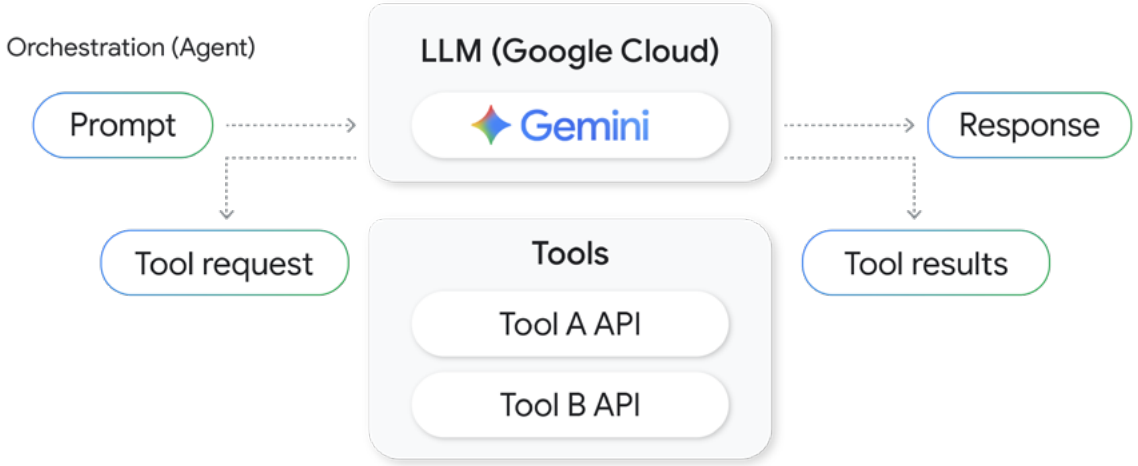
This layer is responsible for recording actions and state changes with strong consistency and integrity. It serves as the system of record, often requiring ACID guarantees to ensure reliability.

Data service	Overview	Startup use cases
Cloud SQL	A fully managed service for traditional relational databases that provides strong consistency for single-region transactional workloads. It is the standard choice for reliable state management.	When an agent successfully executes a critical business action, such as processing a subscription payment or provisioning a new service for a user via an API call, it writes a record to a Cloud SQL database. This creates a permanent, ACID-compliant audit log, ensuring every important agent-driven action is reliably tracked and verifiable.
Cloud Spanner	A globally distributed, strongly consistent relational database offering horizontal scalability. It is designed for mission-critical applications that require high availability and transactional integrity across geographic regions.	A startup would typically migrate from Cloud SQL to Spanner only after achieving product-market fit where its user base becomes globally distributed. For example, a travel or ecommerce app that initially used Cloud SQL now needs to process bookings or orders from users in North America, Europe, and Asia simultaneously without data conflicts. Spanner's global transactional consistency supports this scale.

Agent orchestration: The executive function

Orchestration is the operational core that guides an agent through a multi-step task. For any process that requires more than a single action, it determines which tools are needed, in what sequence, and how their outputs should be combined to achieve a final goal.

As the agent's executive function, you can apply the orchestration to be responsible for planning and decision-making. And, by automating complex business processes, it creates powerful leverage for small startup teams.



Orchestration concepts and cognitive architecture

A common and effective orchestration pattern is **ReAct (Reason + Action)**, a framework that synergizes the reasoning and acting capabilities of large language models.[4]

ReAct establishes a dynamic, multi-turn loop where the model generates both reasoning traces (thoughts) and task-specific actions in an interleaved manner. This allows for greater synergy—reasoning helps the model track and update action plans, while actions gather information from external tools to inform the reasoning process.

Here's how it works:

1. **Reason:** The agent assesses the goal and the current state, forming a hypothesis about the next best step and whether a tool is required.
2. **Act:** The agent selects and invokes the appropriate tool.
3. **Observe:** The agent receives the output from the tool. This new information is integrated into the agent's context and feeds into the next Reason step of the cycle.

4. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. Published as a conference paper at ICLR 2023.

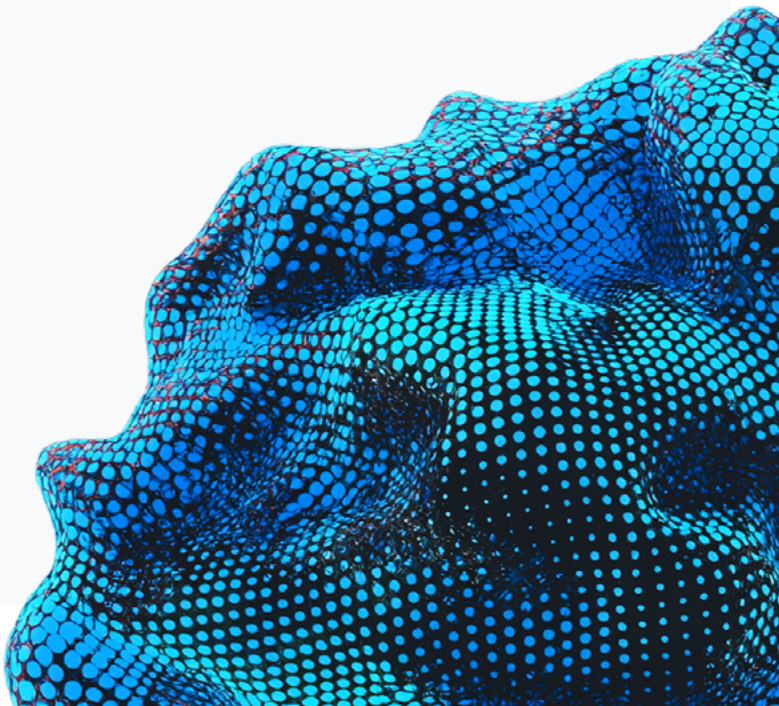
Example: Processing a refund with ReAct orchestration

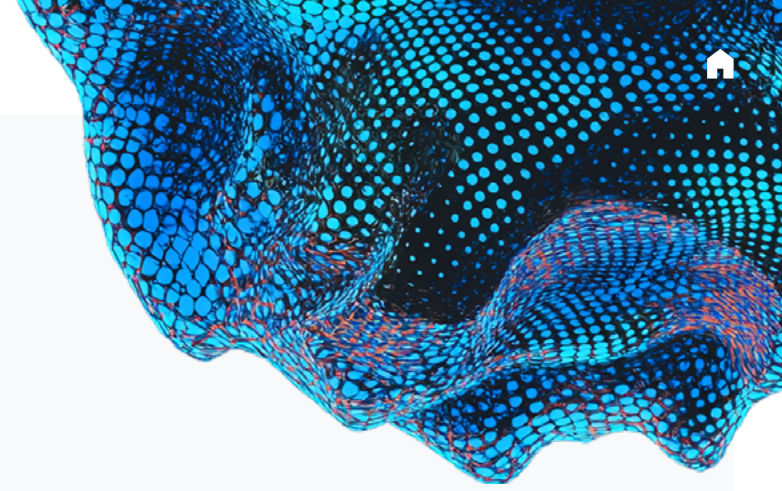
- **Reason:** The user wants a refund. The first step is to understand the company's rules for refunds.
- **Act:** Use the semantic_search tool to query the internal knowledge base with "refund policy."
- **Observe:** The tool returns: "Full refunds are available for all products within 30 days of the purchase date."
- **Reason:** The policy requires the purchase date. This information for the user's specific order must come from the CRM.
- **Act:** Call the get_order_details function from the CRM tool with the user's ID.
- **Observe:** The function returns an order object, including purchase_date: '2025-07-20'.
- **Reason:** The current date is July 29, 2025. The purchase was 9 days ago, which is within the 30-day window. The criteria are met, and the refund can be initiated.
- **Act:** Call the process_refund tool with the order ID and refund amount.
- **Observe:** The tool returns status: 'success'.
- **Final answer:** "Your refund has been processed successfully. You should see the amount credited to your account within 3-5 business days."

Use cases

- **Automated customer onboarding:** An agent could be orchestrated to guide a new user through setup. It might first use a tool to create a new account via an API, then use a send_email tool to deliver a welcome message, and finally use a tool to check the database to confirm the user has completed their first action, triggering a follow-up prompt if they haven't.
- **Proactive system monitoring and remediation:** An orchestration could be triggered by a monitoring alert. First, the agent gets more context by using a tool to query logs from Cloud Logging. Based on the logs, it might then decide to use a kubectl tool to restart a specific pod in GKE, and finally use a slack_notification tool to report the action to the on-call channel.
- **Complex lead qualification:** A sales agent could be orchestrated to enrich a new lead's email with company data from an API. It would use a tool to search the internal CRM to see if the lead is an existing customer. It would then use the collected information to decide whether to assign the lead to a senior sales representative or add them to a nurturing sequence.

Mastering orchestration is the key to moving beyond simple, single-shot agents. When you get it right, you create sophisticated, autonomous systems that can tackle problems that, previously, were not technically feasible. It unlocks a new class of applications and user experiences.





Runtime: Deploying agents at scale

Deploying a functional agent prototype into a production environment requires a robust runtime infrastructure. The runtime facilitates agent deployment at scale, turning a prototype into a reliable product that handles complex operational requirements like security, load balancing, and error handling, especially during periods of unpredictable user growth.

Runtime concepts and architecture

A production-grade runtime environment for AI agents must provide several core capabilities:

- **Scalability:** The infrastructure must automatically scale to handle variable loads, from zero to millions of requests. This includes both request-based load balancing and resource-based autoscaling to manage computational demands efficiently.
- **Security:** The runtime must provide a secure execution environment, managing identity, network access controls, and secure communication channels (e.g., TLS) to protect the agent and the data it accesses.
- **Reliability and observability:** The system must include mechanisms for error handling, automatic retries, and comprehensive monitoring. This involves logging agent actions and tool outputs, and collecting metrics on performance and resource utilization to diagnose and resolve issues.

Use cases

Your choice of runtime directly impacts operational overheads and your ability to scale.

- **Vertex AI Agent Engine**: A seed-stage startup with a small engineering team deploy their first customer support agent, going from a working prototype to a scalable, secure production endpoint in days instead of weeks.
- **Cloud Run:** A startup experiencing rapid but unpredictable growth for their new AI-powered feature deploy their ADK agent on this serverless architecture, so they only pay for compute when the agent is actively processing requests. It's a cost-effective way to handle traffic spikes without over-provisioning infrastructure.
- **Google Kubernetes Engine (GKE):** A Series B startup with an established platform engineering team and dozens of microservices decide to host their new internal automation agent on their existing GKE cluster. This way, they can use established CI/CD processes, security policies, and monitoring dashboards, ensuring the agent adheres to the same operational standards as the rest of their production services.

1.3 The role of grounding in agentic systems

An agent's credibility and usefulness depends on its ability to provide accurate, trustworthy answers based on verifiable facts, a process known as grounding. This section explores the evolution of grounding techniques, providing a roadmap for building increasingly sophisticated and reliable agents.

We begin with the foundational pattern of RAG, which grounds an agent by retrieving text based on semantic similarity. We then explore GraphRAG, which enriches grounding by understanding the explicit relationships between data points in a knowledge graph. Finally, we cover Agentic RAG, where the agent is no longer a passive recipient of information but an active, reasoning participant in the retrieval process itself, capable of executing multi-step strategies to find the best possible answer.

RAG: A foundational first step

The first step on the path to sophisticated grounding is the architectural pattern of RAG. This approach enhances an LLM's responses by retrieving relevant information from an external knowledge base before generating an answer. Instead of relying solely on its pre-trained knowledge, the agent performs a semantic search to find verifiable data, which is then passed to the LLM as context. This ensures a baseline of grounded, verifiable answers.

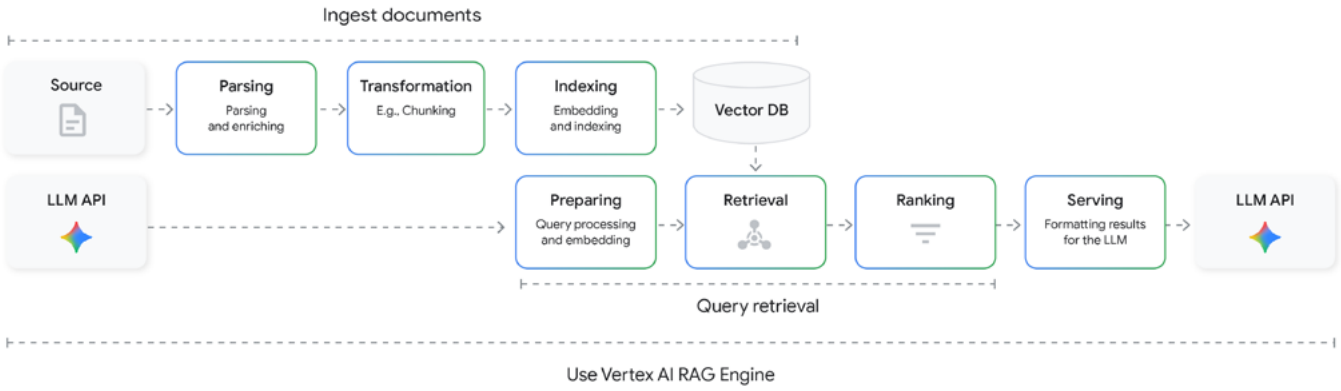
While foundational, this simple retrieve-then-generate process treats knowledge as a flat collection of disconnected facts. It is a powerful technique for direct question-answering, but it falls short on complex queries that require a deeper understanding of the relationships between data points.

Benefits of RAG for agentic systems

- **Agents can access the latest information:** The retrieved info is more current than their last training date, enabling more timely and relevant agentic behavior.
- **Agents are more accurate:** RAG significantly reduces the risk of outputs that could lead to incorrect or inappropriate agentic actions.
- **Faster responses:** Vector embeddings and specialized databases enable lightning-fast semantic searches of massive datasets, so agents can deliver more responsive, timely decisions.
- **More comprehensive agent awareness:** The RAG workflow, which consists of ingesting, parsing, chunking, embedding, storing, and retrieving, can be applied to text, images, and other types of data. With this deeper understanding, agents can perform more complex, multistep reasoning tasks.

Google Cloud's managed, out-of-the-box RAG solution is called Vertex AI Search. It simplifies the process of integrating data sources, and can also use open source or third-party tools. Vertex AI RAG Engine provides a data framework for developing context-augmented LLM applications that deliver accurate, controlled responses aligned with specific knowledge and policies. This is ideal for critical startup applications like customer support, internal knowledge management, and compliance-related tasks.

Vertex AI RAG Engine at work



Tool

Use Vertex AI Search and Vertex AI RAG Engine to ground responses using your proprietary content.

Pro tip

Use the check grounding API to verify whether the AI's answers are based on grounded, up-to-date info.

Vector databases: Search by meaning

The ability to search by meaning, not just keywords, is made possible by vector embeddings. These numerical representations capture the conceptual essence of data (like text and images), allowing a system to find relevant information no matter how a question is phrased.

Vector databases are the infrastructure that makes this possible at scale. They are highly specialized systems designed to store, index, and query millions of these embeddings with the extremely low latency required for a responsive agentic system.

Here's how it works:

1. **Data is transformed into vector embeddings:** The ML model places semantically similar items close together in a multidimensional vector space.
2. **Storage and indexing:** The vector database stores these embeddings and builds specialized indexes to enable fast and efficient similarity searches.
3. **Querying:** The user's query is converted into an embedding using the same model. The database then finds the embeddings in its index that are closest to the query embedding, effectively retrieving the most semantically relevant information to ground the model's response.

Use case

Enhancing customer support

A shoe company uses a vector database with semantic search to power a customer support chatbot:

- Product descriptions, warranty information, and FAQs are all converted into embeddings and stored.
- The vector database understands that "good for people with wide feet" is semantically related to concepts like "wide fit," "extra wide," or "comfortable for wide feet."
- It retrieves relevant product recommendations and provides a much better customer experience.

Compare this to if the shoe company used a **traditional database**. A query using LIKE '%good for people with wide feet%' would fail to return any results because that exact phrase does not exist in the database.

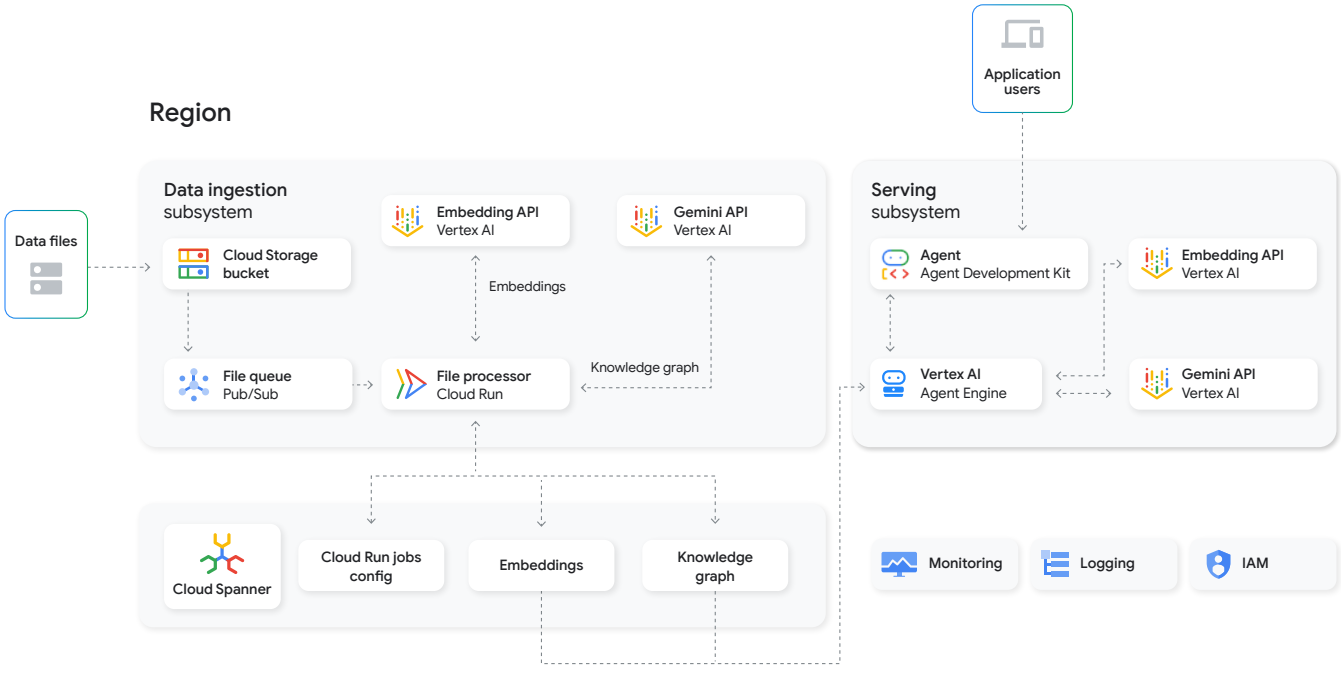
GraphRAG: Smarter grounding

GraphRAG builds a knowledge graph, so instead of just matching similar phrases, your agent understands how concepts relate.

Use case

A medical AI assistant that needs to know **"symptoms → causes → treatments"** and not just retrieve related snippets.

The hierarchy of knowledge in GraphRAG



Agentic RAG: Dynamic reasoning and retrieval

The most powerful approach to grounding is Agentic RAG, a technique that helps you transform the agent from a passive recipient of retrieved data into an active, reasoning participant in the search for knowledge. Following frameworks like ReAct, the agent can analyze a complex query, formulate a multi-step plan, and execute multiple tool calls in sequence to find the best possible information.

A prime example of this agentic pattern is grounding with Google Search. You can use the Gemini 3 family of models to integrate advanced reasoning, allowing them to interleave search capabilities with internal thought processes to answer complex, multi-hop queries and execute long-horizon tasks. The agent can help you handle the entire workflow automatically: it analyzes the prompt, formulates and executes precise search queries, and synthesizes a final, grounded response with sources.

An agent built with the Gemini family of models moves beyond simple content recognition to active, multi-step problem-solving. For example, an agent could:

- Analyze a photo to identify a specific species of plant and then autonomously retrieve its detailed care instructions.
- Process an audio stream from a support call to not only transcribe the words but also determine the customer's sentiment, such as frustration, to escalate the ticket properly.

This ability to perceive and reason across different data types transforms the agent from a data processor into a problem-solving tool that understands and interacts with the world in a more complete way.

> "The conventional wisdom was that foundation model performance would improve exponentially, but we are reaching the inflection point where that climb is plateauing and real differentiation lies in specialization and context engineering. Agentic RAG forms a central pillar of the context layer, allowing AI agents to iteratively find, retrieve, and reason over ground truth data before generating a final answer.
>
> The future is multi-LLM: different models for different tasks, connected by a model- and data-agnostic context layer that unlocks their full potential."
>
> **Douwe Kiela**
> CEO of Contextual AI

Example: Real-time inventory check

Define a function called check_inventory that takes a product_ID and returns the current stock levels from your real-time inventory system. Similarly, another function, check_warranty_status, could take a product_ID and return its warranty information directly from your warranty management system.

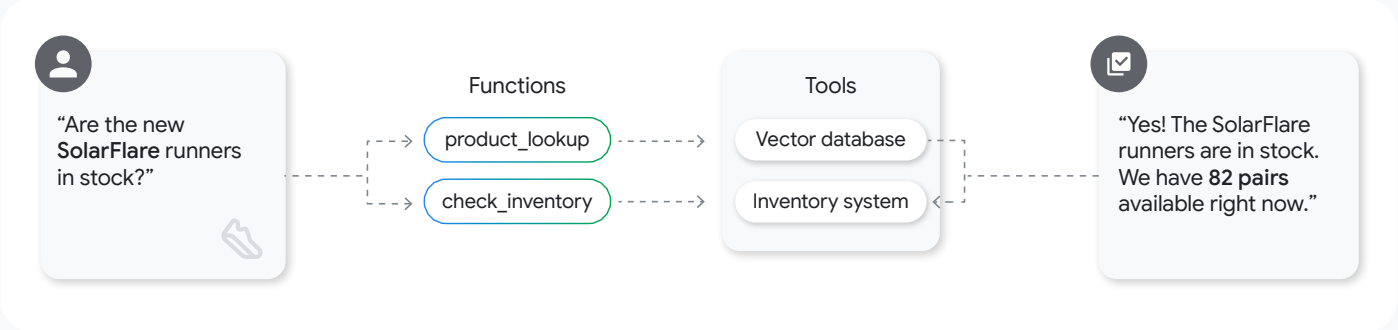
Then, when a customer asks about a specific product's availability, the AI agent:

1. **Identifies the product:** It uses semantic search (powered by the vector database) to accurately identify the specific shoe model the customer is asking about, even if they describe it vaguely.
2. **Triggers the action:** It recognizes the need for real-time stock information and uses function calling to invoke your check_inventory function.
3. **Provides real-time response:** The check_inventory function executes, fetches the live stock data from your inventory system, and returns it to the AI agent. The agent then provides the customer with an immediate, accurate response about availability.

This combination of retrieval (knowing what information is relevant) and actions (performing real-time operations) makes your AI agents smarter, faster, and much more useful.

Pro tip

Use Vertex AI and Google Cloud's vector search to add this to your agent.



The Agentic RAG workflow on Google Cloud

Google Cloud provides managed services that handle the entire Agentic RAG workflow:

- **Generating embeddings and indexing:** The first step is to convert your data into vector embeddings. An option is the Gemini Embedding model, which is available in both Vertex AI and the Gemini API and supports over 100 languages. This and other models are part of the broader Vertex AI Embedding APIs suite.
- **Storing and indexing:** The vector embeddings are then stored and indexed in Vertex AI Vector Search. This is a fully managed, high-performance vector database that automatically builds the specialized indexes required for fast and efficient similarity searches at scale.
- **Retrieval and reasoning:** When a user submits a query, it is converted into an embedding and used by Vertex AI Vector Search to find the most relevant information. This targeted context is then passed to the LLM to generate the final, grounded response.

Pro tip

Use the retrieve and re-rank approach.

Address the trade-off between recall (finding all relevant documents) and precision (ensuring every retrieved document is relevant) using the two-step "retrieve and re-rank" approach (shown in this agentic_rag sample). First, it widens the recall aperture by configuring Vertex AI Vector Search to retrieve a larger-than-needed set of candidate documents. Second, this larger set is passed to the LLM or a specialized re-ranking service, which identifies the most relevant documents and discards any that are irrelevant or semantically opposite.

From retrieval to reasoning: A strategic advantage

Agentic RAG represents a fundamental leap, moving beyond simple information retrieval to genuine problem-solving. By empowering the agent to be an active, reasoning participant, developers can build systems capable of executing the complex, multi-step queries and long-horizon tasks that define next-generation agentic capabilities.

For a startup, mastering this approach is the key to building a defensible, truly intelligent product that can unlock novel workflows and user experiences.

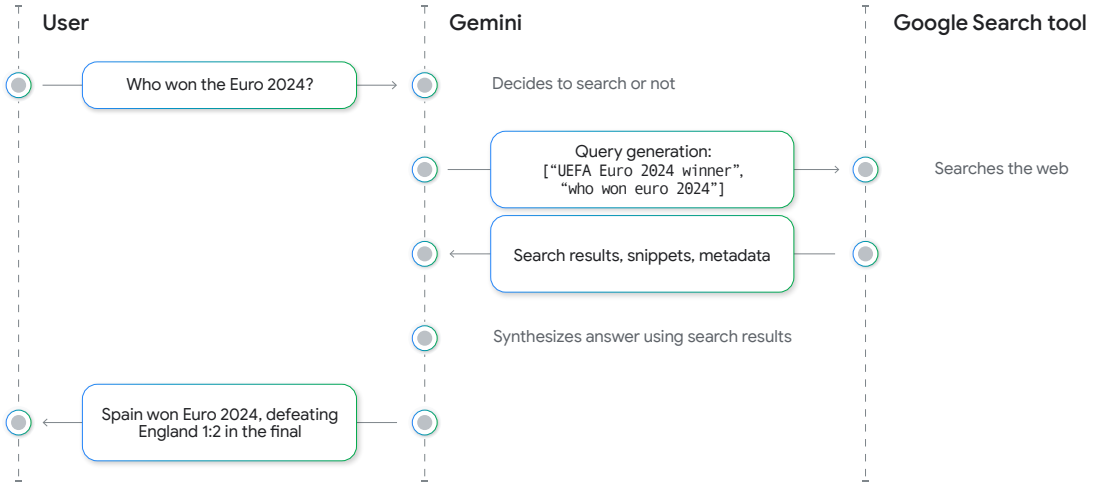
Other grounding methods

While RAG is a foundational technique, Vertex AI offers other ways to ensure your agents deliver accurate, reliable responses. For example:

- **Grounding with Google Search:** Connect your model to world knowledge and a wide possible range of topics.
- **Grounding with Google Maps:** Use Google Maps data with your model to provide more accurate and context-aware responses to your prompts.
- **Grounding Gemini to your data:** Use RAG to connect your model to your website data or your sets of documents.
- **Grounding Gemini with Elasticsearch:** Use RAG with your existing Elasticsearch indexes and Gemini.

Example: Grounding with Google Search

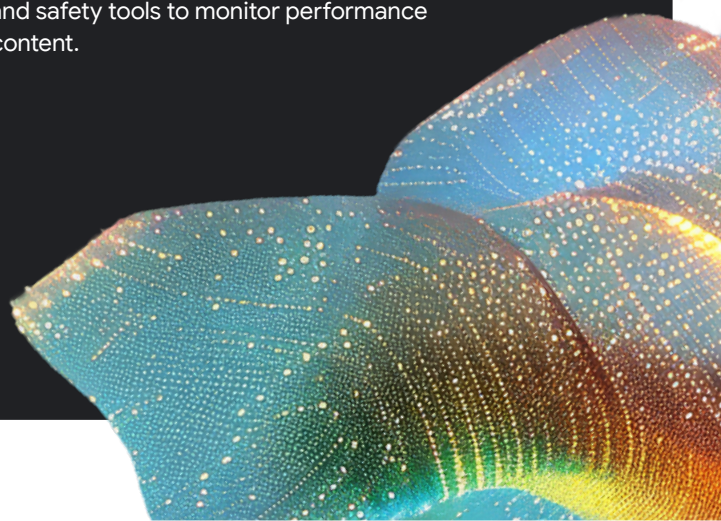
When you use the Google Search tool, the model handles the entire workflow of searching, processing, and citing information automatically.

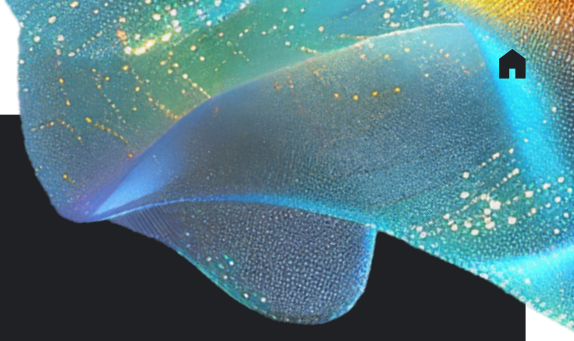


1. **User prompt:** The application sends the prompt to the Gemini API with the Google Search tool.
2. **Prompt analysis:** The model analyzes the prompt and determines if a Google Search can improve the answer.
3. **Google Search:** If needed, the model automatically generates one or multiple search queries and executes them.
4. **Search results processing:** The model processes the search results, synthesizes the information, and formulates a response.
5. **Grounded response:** The API returns a final, user-friendly response that is grounded in the search results. This response includes the model's text answer and grounding metadata containing the search queries, web results, and citations.

Key takeaways: Choosing your AI agent's components

Your goal	Best option
Choose the agent's core intelligence.	Select a model (e.g., Gemini) based on your use case and fine-tune it with your specific data.
Make your agent trustworthy and factual.	Use grounding techniques like RAG with a vector database so it checks facts, not just guesses.
Manage a complex task with multiple steps.	Use orchestration to create a plan that determines which tool to use, in what order, and how to combine the results.
Connect to public, real-time data and services.	Use pre-built extensions to easily plug into third-party APIs.
Connect securely to your own internal tools.	Write custom functions to give the agent controlled access to your private databases, CRMs, or other internal systems.
Deploy a reliable and safe product at scale.	Use a managed runtime for scalable infrastructure, plus built-in evaluation and safety tools to monitor performance and block harmful content.





Ready to turn your AI vision into reality? We're here to help.

Learn how to build more generative AI applications with on-demand Startup School sessions.

Start now

Get up to $350k USD in Google Cloud credit with the Google for Startups Cloud Program.

Apply now

Contact our Startups team.

Get in touch

Stay connected and get all our latest updates by subscribing to the Google Cloud Startup Newsletter.

Subscribe

Section 2

How to build AI agents

The previous section defined the fundamental components of a modern agentic system: a core reasoning model, a set of tools to enable action, data architecture options to persist short-and long-term agent memory, a grounding mechanism to ensure factual accuracy, and deployment options.

With this conceptual framework established, we can now shift our focus to how you build an agent. This section provides an opinionated, practical guide to the architectural decisions involved in building a production-ready agent.

Open and flexible, the Google Cloud ecosystem recognizes the industry offers many excellent frameworks. Here, however, we intentionally focus on ADK, a complete, robust implementation that fits perfectly in the Google Cloud ecosystem. We also make specific recommendations built upon open industry standards like the Model Context Protocol (MCP) and the Agent2Agent (A2A) protocol.

Pro tip

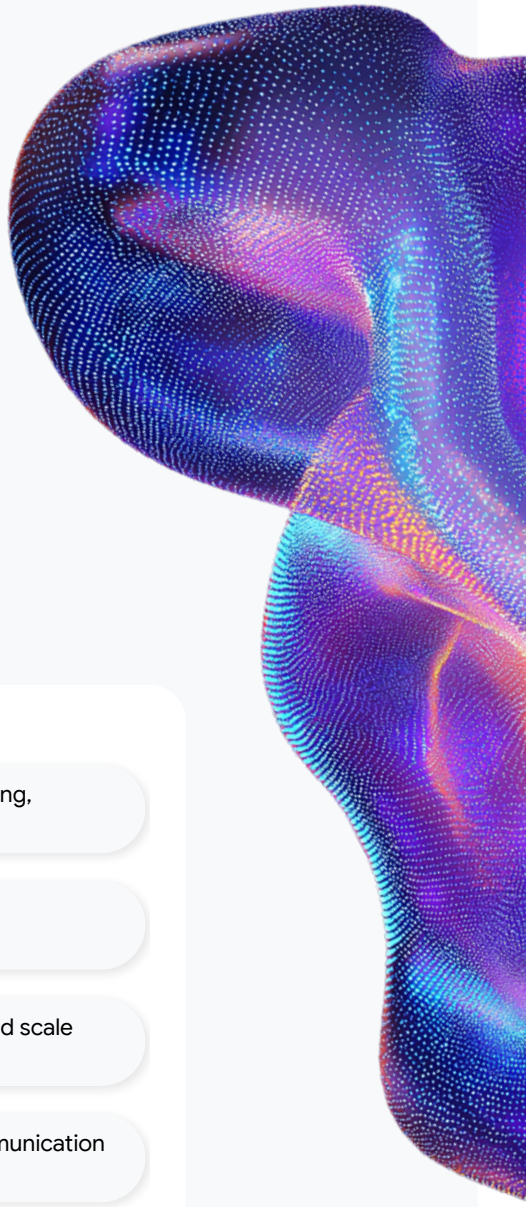
Before building your agent, explore 601 real-world agent use cases from the world's leading organizations.

Prefer audio? Listen to the podcast version of this section, created with NotebookLM.

This podcast was created using NotebookLM with the prompt: "As a podcast host, create a practical podcast, targeting developers and technical founders. The podcast should introduce Agent Development Kit (ADK) as a solution to common agent-building challenges. It needs to detail ADK's core benefits and explain its primary agent types, such as the intelligent LlmAgent and structured Workflow Agents (e.g., Sequential, Parallel, and Loop).

"The podcast must then cover the broader ecosystem, including the Model Context Protocol (MCP), Vertex AI Agent Engine for deployment, and the Agent2Agent (A2A) protocol for communication. Briefly mention alternative tools like Google Agentspace, Firebase Studio, and Gemini CLI, and conclude with a summary and a call to action directing listeners to Google's startup resources."

2.1 A complete toolkit for building AI agents



When it comes to building a custom AI agent for your startup, founders face a critical trade-off: development velocity versus flexibility.

At one end, you have easy-to-use solutions like low-code platforms or pre-built products. These are fast to implement but give you less control, making them best for standard problems. At the other end, you have highly flexible frameworks or the option to build everything from scratch. While these give you maximum ability to customize, they require more significant development resources and deep technical expertise. ADK sits in the middle of this development landscape.

Explore ADK

Core components for building AI agents

Agent Development Kit	Open-source, code-first toolkit for building, evaluating, and deploying AI agents.
Model Context Protocol	Open protocol that standardizes how applications provide content to LLMs.
Vertex AI Agent Engine	Managed platform to deploy, manage, and scale AI agent in production.
Agent2Agent (A2A) protocol	Open standard designed to enable communication and collaboration between AI agents.

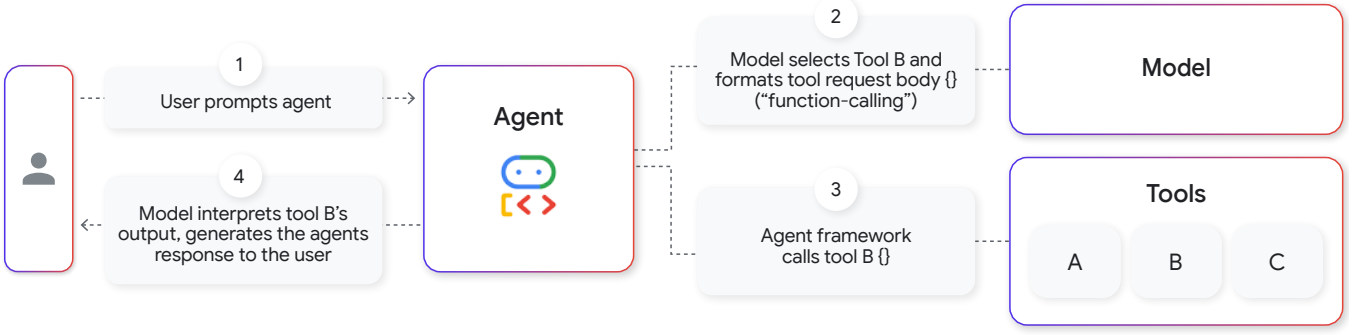
Develop with ADK

If you need more power than a simple low-code tool but want an accelerated development process, ADK gives you the control to build a unique system of collaborative agents while simplifying complex technical challenges. For example, specific protocols like MCP and A2A make it easier for agents to extend their capabilities (as we'll explore below).

What's more, ADK complements the tools your team may already use and integrates with the broader Google Cloud ecosystem, so you're not locked into a single approach. This flexibility extends to how you run your agents once they're built, with options to deploy to a fully managed service like Vertex AI Agent Engine or to a versatile serverless platform like Cloud Run. It's all about choosing the right foundation for your specific operational needs.

> **We are focused on building a flourishing AI agent ecosystem. The open source Agent Development Kit has had over a million downloads in less than four months."**
>
> **Sundar Pichai**
> CEO of Google and Alphabet

Building complex workflows gets easier with ADK



What you can do with ADK

1. Build complex, collaborative AI systems

ADK is multi-agent by design. It's easy to build highly specialized AI solutions that automate complex, multi-step workflows. And, with flexible orchestration (sequential, parallel, or dynamic), you can start with simple automations and evolve to highly adaptive systems. For example, you can build an intelligent project management system with a "Task Breakdown Agent" that delegates sub-tasks to specialized "Code Generation Agents," "Design Agents," and "Documentation Agents".

2. Integrate AI into existing tools, agents, and workflows

ADK is built around a rich tool ecosystem that allows your agents to interact with all your existing tools and data. You can connect your agents to productivity tools you already use, like Notion, Slack, or a CRM, as well as tool frameworks like LangChain and LlamaIndex, or agent frameworks like LangGraph or CrewAI. Tools can be shared with MCP and the agents you create can be shared with A2A. This way, you can inject AI intelligence into every facet of your operations, enhancing the tools and systems you already have in place without requiring a disruptive architectural overhaul.

3. Ensure quality and reliability from day one

To earn users' trust, it's critical to carefully test and evaluate your agents before deploying them to production. ADK's built-in observability and evaluation tools help you:

- **Rapidly iterate:** Before deploying, you can systematically test how your agents respond to various scenarios and how well they execute complex tasks.
- **Debug agent behavior:** Inspect the full execution trace of your agent including its reasoning (thoughts), tool calls, and observations to understand its decision-making process and debug complex, multi-step workflows.
- **Benchmark your agents:** Evaluate different agent designs or model updates against predefined metrics, using a data-driven approach to continuously improve agent performance.

This moves your process beyond simple "vibe-testing," allowing you to launch professional-grade agents quickly, build user trust through proven reliability, and iterate with data-driven confidence.

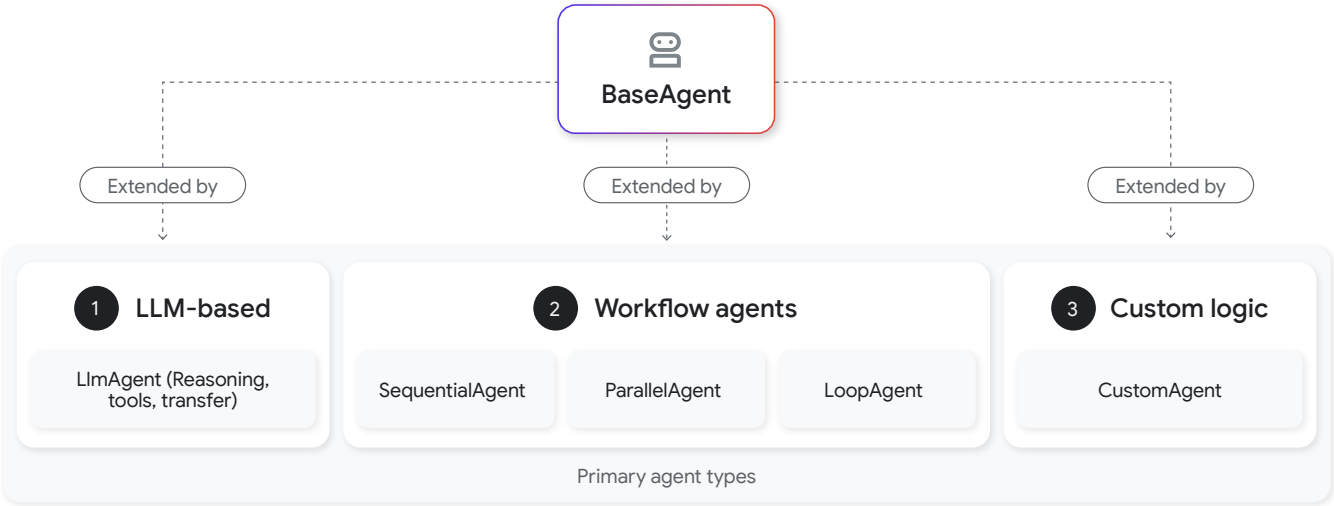
4. Scale AI with confidence

As you grow, your AI solutions need to scale seamlessly without becoming a bottleneck. ADK accelerates the path to production by using AgentOps (described in detail below) to bridge the gap between local development and deployment. The framework exposes agents as standard web services using FastAPI, which can then be containerized. This frees your developers from having to build custom deployment infrastructure. They can deploy anywhere, from local testing to fully managed, auto-scaling runtimes like Vertex AI Agent Engine or Cloud Run.

ADK core: Agent architectures

A foundational step in building with ADK is selecting the right agent architecture. Distinct agent classes are designed for different execution patterns, and your choice will determine how your agent reasons and operates. It's typically a trade-off between the flexible, non-deterministic power of an LLM and the predictable, deterministic control of hard-coded logic. Understanding the interplay between these agent classes is key to building robust and effective AI systems.

ADK's agent types are organized into three categories



1 LLM agent (LlmAgent)

Core engine: LLM

Determinism: Non-deterministic (flexible)

This is the most common agent type and is commonly referred to as simply "Agent." It uses an LLM like Gemini for complex reasoning, dynamic decision-making, and natural language understanding; and it forms the core of most conversational and problem-solving agents.

2 Workflow agent (SequentialAgent, ParallelAgent, LoopAgent)

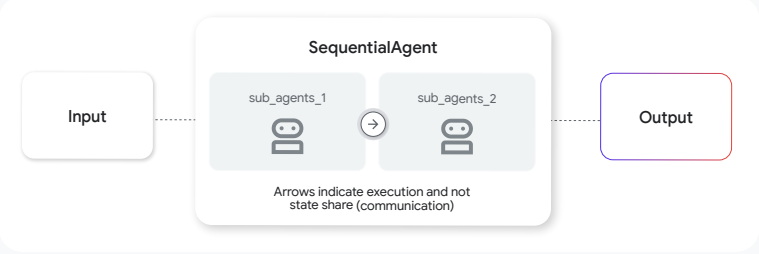
Core engine: Predefined logic

Determinism: Deterministic (predictable)

These orchestrators deterministically control how other agents execute in predefined patterns. They're used for structured processes.

Sequential agents (SequentialAgent)

Executes sub-agents in a fixed order, passing the output of one as the input to the next. Agent A completes Task 1, then passes its output to Agent B for Task 2.

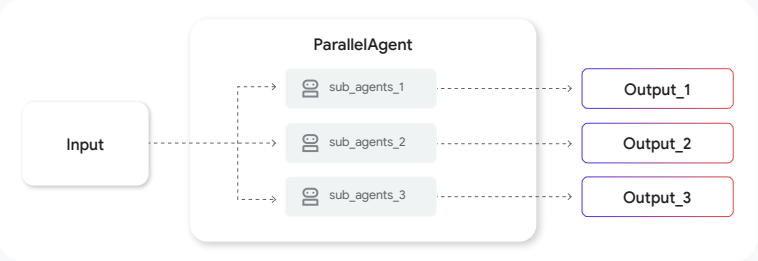


Example: You want to build an agent that can summarize any webpage, using two tools: Get Page Contents and Summarize Page. Because you can't summarize from nothing, the agent must always call Get Page Contents before calling Summarize Page.

Get the full code

Parallel agents (ParallelAgent)

This executes multiple sub-agents simultaneously. This is used for performance optimization when tasks are independent. Agent A and Agent B work on independent sub-tasks simultaneously, and their results are combined.

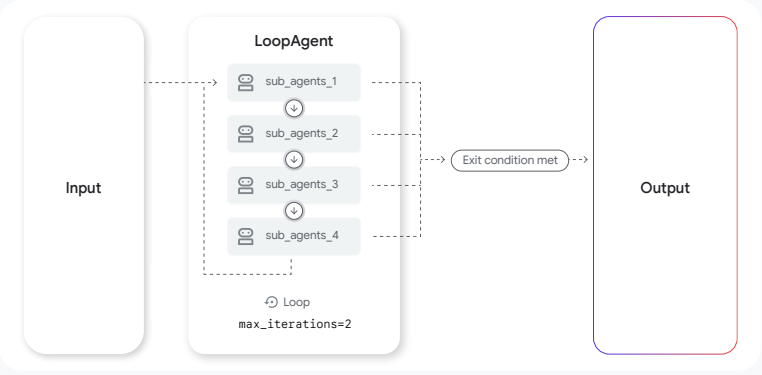


Example: For operations like multi-source data retrieval or heavy computations, parallelization yields substantial performance gains. Importantly, this strategy assumes no inherent need for shared state or direct information exchange between the concurrently executing agents.

Get the full code

Loop agents (LoopAgent)

A workflow agent executes its sub-agents in a loop (iteratively). It repeatedly runs a sequence of agents for a specified number of iterations or until a termination condition is met. Use the LoopAgent when your workflow involves repetition or iterative refinement, such as revising code.



Example: You want to build an agent that can generate images that contain specific amounts of food (e.g., five bananas), using two tools: Generate Image, Count Food Items. Because you want to keep generating images until it either correctly generates the specified number of items, or after a certain number of iterations, you should build your agent using a LoopAgent.

Get the full code

3 Custom agent (BaseAgent subclass)	
Core engine: Custom Python code **Determinism:** Can be either, based on implementation	**Custom agent (BaseAgent subclass)** For unique requirements and tailored workflows that go beyond a standard reasoning loop, you can create a custom agent by inheriting directly from BaseAgent and writing custom Python logic to control its behavior. This approach is necessary when an agent's actions are not determined by an LLM, but by specific, hard-coded rules. **How it's done:** A developer creates a new Python class that inherits from the BaseAgent class. They must then implement the _run_async_impl method, which contains the unique logic the agent will execute. This method has full access to the session's state and can yield events to communicate with other agents or terminate a workflow. See Gemini Fullstack Agent Development Kit (ADK) Quickstart for an example of how to implement a Custom agent.

ADK orchestration: Implementing the ReAct loop

As we discussed in section 1, the ReAct paradigm is a foundational pattern for agentic systems. ADK provides the core abstractions and classes needed to implement this dynamic, cyclical process in a structured way. Its LlmAgent is specifically designed to execute this loop and handle the transitions between the fundamental stages in the loop:

- **Reasoning (thought):** The LlmAgent class manages this stage internally. It takes the user's prompt and its current internal state, then calls the underlying language model to form a hypothesis and determine the next best action.
- **Action (tool use and agent delegation):** ADK enables this stage through its flexible tool system. When the LlmAgent decides to act, it can invoke a simple Python function or, for more complex tasks, delegate the work to another specialized sub-agent using the Agent-as-a-Tool pattern.
- **Observation:** ADK automatically captures the dictionary returned by the tool or sub-agent and passes it back to the LlmAgent. This output becomes the new information that the agent integrates into its context, feeding it into the next Reasoning step of the cycle.

By providing a native implementation of this essential pattern, ADK abstracts away the boilerplate code, so you can quickly translate the powerful concept of a ReAct loop into a working, multi-step agent.

ADK tools: A framework for agentic action

In ADK, an agent can use tools to perform actions beyond the native capabilities of its core reasoning model. These defined capabilities enable an agent to execute code, interact with external systems, and act outside its own immediate execution context. A tool is a Python function (or a Java method) that can either implement self-contained logic or act as a wrapper for more complex operations, such as making calls to an API, using MCP to access a variety of external systems, or delegating a task to another specialized agent locally or remotely via A2A.

This section outlines how to design effective tools and the taxonomy of available tool types.

Pro tip

For a complete discussion, including code examples and advanced usage patterns, refer to the ADK documentation.

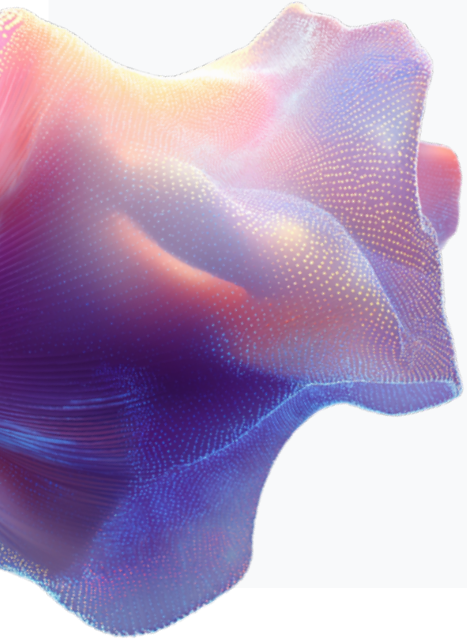
Designing effective tools: The API contract for the model

For a model to use a tool correctly, its definition must serve as a clear and unambiguous **API contract**, composed of:

- **Function signature**: Use descriptive names for tools and their parameters. Python type hints are mandatory, as they provide the structural schema the model uses to generate valid arguments.
- **Docstring (the semantic core):** This is the primary source of semantic information for the model. A well-written docstring must precisely define the tool's purpose, usage criteria, parameters, and expected return schema.
- **Return schema:** A tool must return a dictionary. While not a strict syntactic requirement, it is a best practice to include a status key (e.g., success or error) in this dictionary. This structure is essential for the agent to reliably distinguish between successful outcomes and failures in its Observation step and reason about how to proceed.
- **Stateful tools and** ToolContext**:** For tools that need to read or write to a persistent session state, an optional tool_context: ToolContext parameter can be added to the function signature. The agent automatically injects this object, giving the tool access to a session-level state dictionary.

Pro tip

For best practices and examples of how to define parameters and tool schemas, structure effective prompts, and implement complex, multi-agent workflows, check out the ADK Samples repository.



A taxonomy of ADK tools

ADK provides a flexible architecture for implementing tools, ranging from simple functions to interoperable multi-agent systems.

Toolsets: Packaging related capabilities

A primary pattern in ADK is the Toolset, a class that bundles a collection of related tools into a single, configurable object (e.g., BigQueryToolset, MCPToolset).

Custom function tools

This is the most direct method for extending an agent with proprietary logic.

- FunctionTool: The standard wrapper for synchronous Python functions.
- LongRunningFunctionTool: A specialized tool for asynchronous tasks or human-in-the-loop workflows.

Hierarchical and remote tools

ADK enables the creation of complex systems by composing agents.

- **Agent-as-a-tool:** A delegation pattern where a parent agent uses another specialized agent. This allows the parent to invoke another agent, receive a response, and maintain control to handle future input. (This is distinct from a sub-agent delegation model, where full conversational control is passed to a sub-agent and all subsequent input will be handled by the sub-agent.)
- RemoteA2aAgent: For communication between agents in different processes, ADK provides the RemoteA2aAgent class, which uses the **Agent2Agent (A2A)** protocol to seamlessly integrate distributed systems.

Pre-built and integrated tools

ADK includes a suite of tools and wrappers to accelerate development.

- **Built-in tools:** Ready-to-use tools like Google Search and Code Execution.
- **Google Cloud toolsets:** Rich integrations for services like Vertex AI Search and BigQuery.
- **Third-party interoperability:** Wrappers like LangchainTool and CrewaiTool allow for the direct reuse of tools from popular open-source ecosystems.

Standardize with Model Context Protocol

Model Context Protocol (MCP) is an emerging open standard for connecting AI and LLMs with external data sources and tools. You can plug your AI applications into various data sources and tools without the hassle of building custom point-to-point integrations for each one.

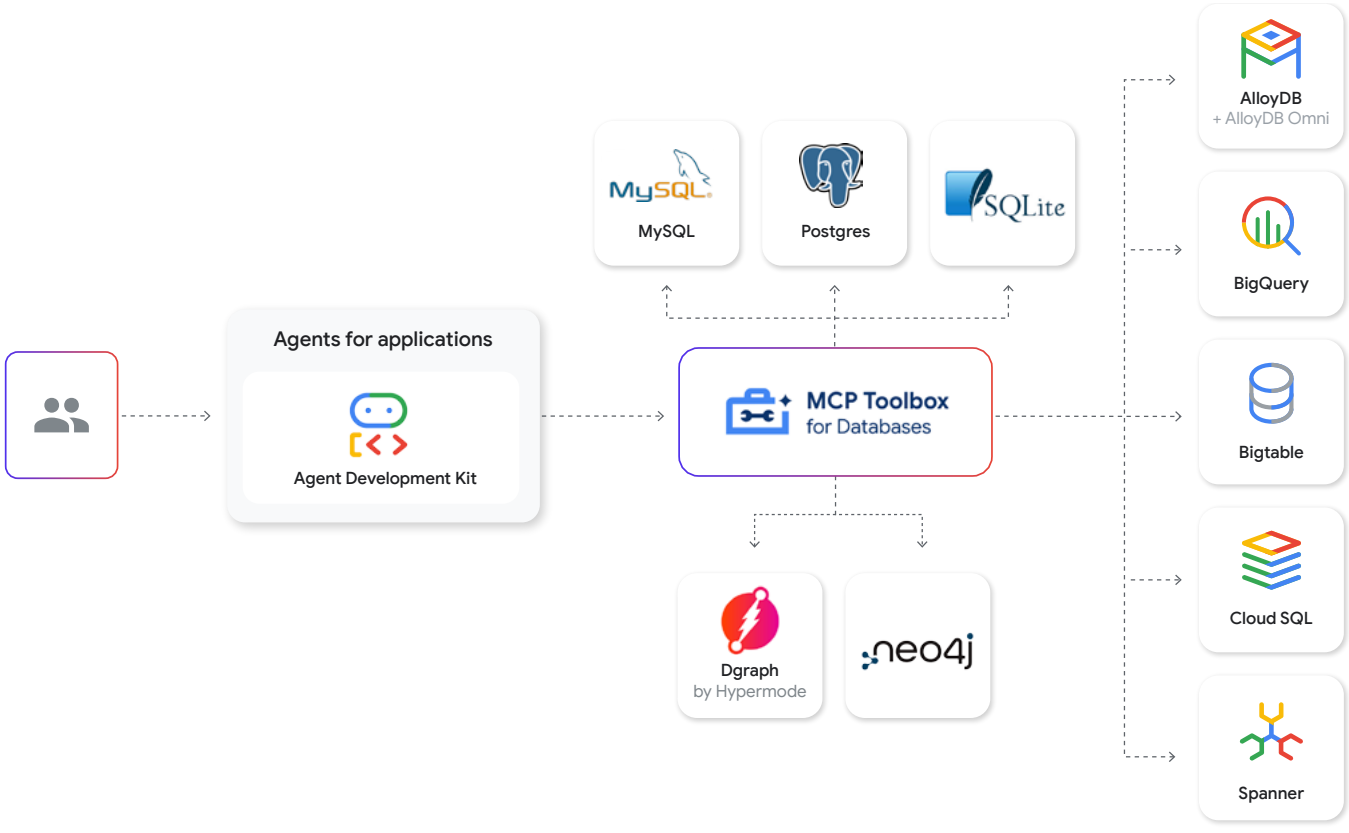
With ADK, your agents can participate in this ecosystem in two ways:

- **Consume external tools:** An ADK agent can act as an MCP client, allowing it to use any tool exposed by a third-party MCP server.
- **Expose native tools:** Developers can wrap their ADK tools in an MCP server, making them securely available to any other MCP-compliant agent or application.

Pro tip

Use the open-source MCP Toolbox for Databases to easily and securely connect your agents to a large array of popular data sources.

MCP is like a universal adapter for an agent's data sources and tools



Manage your data with Google Cloud services

As outlined in the previous section, long-term, working, and transactional memory each play a distinct role in an agent's data architecture. Here, we explain how to build it. ADK provides the necessary patterns and integrations to map this conceptual architecture directly to specific, scalable Google Cloud data services.

1. Long-term knowledge base (grounding, context, and analytics)

This is the agent's permanent memory, combining a searchable knowledge library, a record of user interactions, and a repository for analytics.

- **Vertex AI Search:** Serves as the agent's queryable knowledge library for unstructured information. In ADK, a VertexAISearchToolset allows an agent to ground its responses by retrieving relevant information from a specific set of documents.
- **Firestore:** Functions as the agent's persistent user memory. In ADK, it's used to store and retrieve conversational history and the state of long-running tasks, enabling a continuous, personalized experience that can be resumed across sessions.
- **Cloud Storage:** Acts as the agent's durable file system. ADK uses it as the source of truth for raw documents (e.g., PDFs, images) that are then indexed by services like Vertex AI Search.
- **BigQuery:** Functions as the agent's analytical database. The BigQueryToolset in ADK enables agents to answer questions by executing complex analytical queries against large, structured datasets.

2. Working memory (caching and session state)

This is the agent's high-speed, transient memory for managing the immediate context of a live conversation.

- **Memorystore:** Provides a high-speed cache for the agent. In ADK, its primary role is to store the results of frequent or expensive tool calls, drastically reducing latency and operational costs.

3. Transactional memory (auditing and reliable execution)

This is the agent's durable ledger for recording critical actions and state changes with high integrity.

- **Cloud SQL:** Serves as the agent's reliable system of record. ADK enables patterns where tools log their actions to Cloud SQL, creating a permanent, ACID-compliant audit trail for every important agent-driven action.
- **Cloud Spanner:** Acts as a globally consistent backend for mission-critical agent actions. In an advanced ADK implementation, a tool representing a core business process (e.g., process_global_order) would trigger a transaction in a Spanner-backed system to ensure global integrity.

4. The next frontier: Distilled conversational memory

As an agent's interaction history with a user grows over weeks or months, providing the entire raw context to the model for every query becomes inefficient and cost-prohibitive. Plus, models can get confused.

Memory distillation is the next frontier. It uses an LLM to dynamically and continuously distill long conversation histories into a compact, structured set of essential facts and preferences. The resulting curated, long-term memory is far more efficient to retrieve and use.

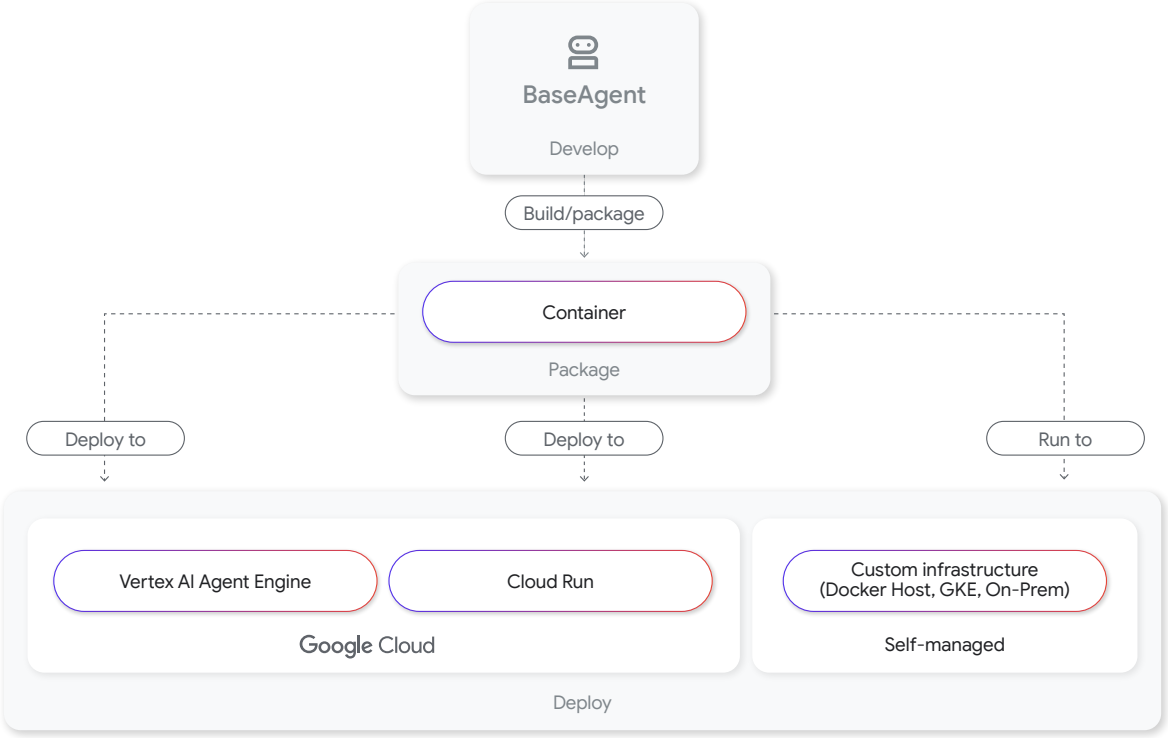
This is an active area of research, but early patterns are emerging. An example is Vertex AI Memory Bank, a managed service on Vertex AI Agent Engine. It provides mechanisms to implement memory distillation:

- **Automated distillation:** It can asynchronously process conversation histories to automatically extract and generate a list of salient facts about the user (GenerateMemories).
- **Agent-directed distillation:** For more control, an agent can use memory-as-a-tool to decide what specific information is important enough to be explicitly written to the memory bank (CreateMemory).

Focusing on a distilled set of memories rather than raw history is more scalable, efficient, and "human-like"; ideal for the next generation of agentic systems.

Deploy to the managed runtime with Vertex AI Agent Engine

ADK is deployment-agnostic by design. The core agent logic you define in Python is decoupled from the serving infrastructure so you can develop and test locally, then deploy the same agent to various production environments.



ADK agents are exposed for deployment as standard web services using FastAPI. The adk api_server command automatically wraps your agent in a production-ready API server, which can then be containerized.

While this container could be deployed to various services on Google Cloud, the three primary managed deployment targets for ADK agents are:

- **Cloud Run:** A managed compute platform for running your agent as a container-based application. This is an excellent choice for integrating your agent into an existing microservices architecture or for use cases requiring custom container configurations.
- **Vertex AI Agent Engine:** A fully managed, auto-scaling service on Google Cloud specifically designed for deploying, managing, and scaling AI agents built with frameworks like ADK. It provides deep integration with the Vertex AI ecosystem for MLOps, monitoring, and security.
- **Google Kubernetes Engine (GKE):** This managed Kubernetes service is the best choice if you have an existing Kubernetes-based infrastructure or are making a strategic day-one decision to prioritize long-term portability, deep architectural control, and the open-source Kubernetes ecosystem. It provides the most granular control over networking, stateful workloads, and specialized hardware like GPUs and TPUs, making it ideal for teams with platform engineering expertise or those building complex, multi-service applications that will need to scale.

Note

It's important to understand the relationship between Google Cloud's agent creation tools. **Vertex AI Agent Builder** is the comprehensive suite of features for the entire agent lifecycle, from discovery to deployment. A core component of this suite is Vertex AI Agent Engine, the managed service specifically designed to deploy, manage, and scale your agents in production.

For this guide, when we discuss the production runtime environment, we are referring to the Vertex AI Agent Engine.

For startups using ADK, Vertex AI Agent Engine is the recommended deployment target. It is specifically optimized to be a cost-effective, auto-scaling solution, and it provides the easiest and most direct path to a scalable, production-ready agent. As a fully managed service, it abstracts the underlying infrastructure, freeing your engineers to focus on core agent logic rather than operational overhead.

As a service designed for agentic workloads, Vertex AI Agent Engine provides several key benefits:

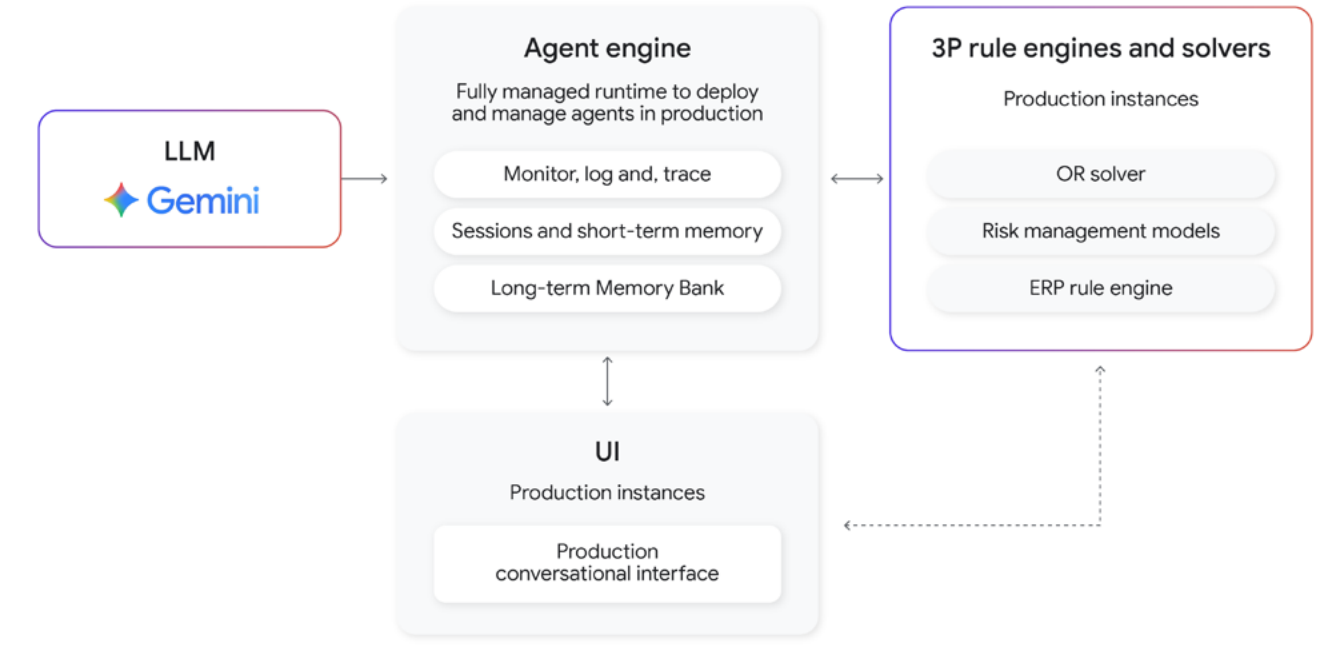
Core capabilities

- **Automated scalability:** Automatically handles scaling to meet varying user loads.
- **Security and authentication:** Provides integrated identity and access management.
- **Framework agnostic:** Supports agents built with various frameworks, not just ADK.
- **Agent lifecycle management:** Provides APIs for creating, reading, updating, and deleting your deployed agents.

Specialized agentic features

- Memory Bank: A managed service to dynamically generate and retrieve long-term, personalized memories based on users' conversations.
- Example Store: Allows developers to provide and manage few-shot examples to improve and steer agent performance on specific tasks.

System architecture for a Gemini-powered agent engine



Collaborate with Agent2Agent communication

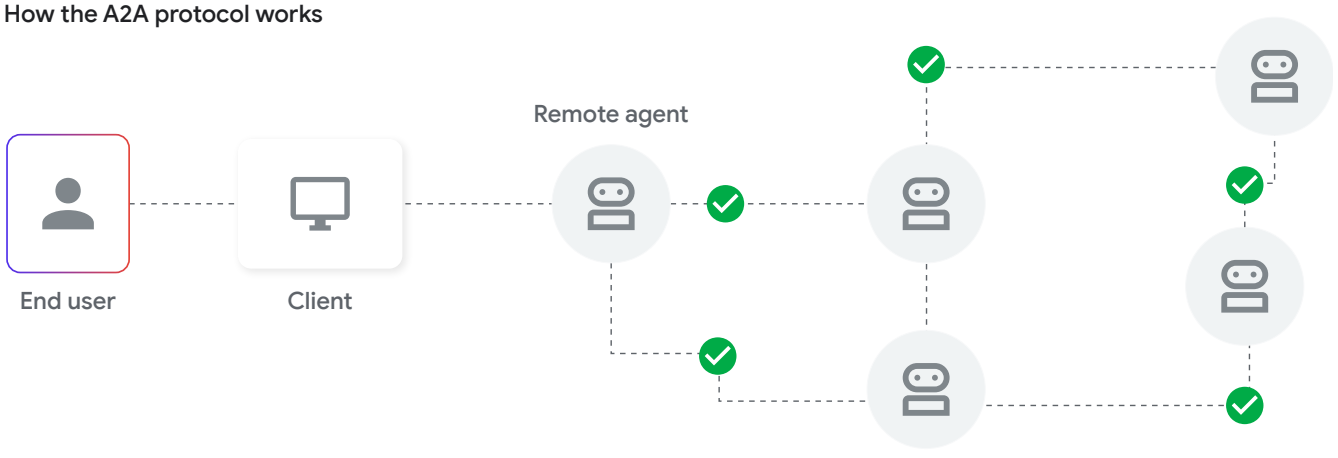
The true power of specialized agents is unlocked when they can collaborate. To enable this, Google champions the Agent2Agent (A2A) protocol, an open standard that ensures the agents you build today can discover, communicate with, and securely coordinate actions with other agents, regardless of who built them or what framework they use. This commitment to an open, interoperable ecosystem is central to Google Cloud's agent strategy.

Key concepts of the A2A protocol include:

- **Agent card:** A digital "business card" (typically a JSON file at a well-known endpoint) that an agent uses to advertise its capabilities, endpoint URL, and authentication requirements, enabling discovery by other agents.
- **Task-oriented architecture:** Interactions are framed as "tasks." A client agent sends a task request to a server agent, which processes it and returns a response. An agent can act as both a client and a server.
- **Modality agnostic:** A2A supports text, audio, and video communication, reflecting the evolving, multimodal nature of agent interactions.

The A2A protocol's rich partner ecosystem

How the A2A protocol works



ADK agents can natively participate in this ecosystem. They expose a standard HTTP endpoint and an agent.json file, allowing them to be discovered and to communicate with any other A2A-compliant agent.

Pro tip

Explore these A2A resources to get started:

- A2A Project Github org
- A2A protocol docs
- A2A protocol specification

Customer story

How BioCorteX uses the A2A protocol to accelerate drug discovery.

BioCortex uses knowledge graphs and in-silico simulations to model the complex interactions between bacteria, drugs, and the host to uncover the hidden interactions and and de-risk the drug development process for our pharma partners.

Situation

In life sciences, connecting and transforming disparate datasets into commercially relevant knowledge is slow and uncertain. Hypothesis testing can take years, hampered by poor models, conflicting theories, and the sheer complexity of biology.

Solution

BioCorteX built a multi-agent system on GCP that interrogates hypotheses from three dimensions: biological plausibility, real-world clinical relevance, and commercial importance. It uses Gemini-powered agents, the ADK, and a graphRAG to navigate our 44 billion-connection knowledge graph of global samples—all orchestrated via A2A.

Impact

What once took years now takes days. BioCorteX graph agents deliver fully transparent scenario planning to key decision makers across their portfolio, underpinning high-level commercial considerations with deep scientific knowledge—accelerating testing of new mechanisms and therapeutic areas while cutting waste across the pipeline.

> **“At BioCorteX, our Carbon Graph agents are different. Unlike other agents, they deal in facts not opinions or associations. By traversing the world’s largest mechanistic biology-based knowledge graph [rather than using an LLM], Carbon Graph agents do not suggest hypotheses, they test their plausibility, clinical relevance, and commercial aspects, allowing full alignment across R&D, Regulatory, and Commercial teams within the organization.”**

2.2 A step-by-step guide: Defining an LLM agent

Building an AI agent is an iterative process of definition, testing, and deployment. This section focuses on the first and most critical phase: defining the agent's core identity, instructions, and capabilities.

While the open source ADK Samples repository provides a comprehensive library of ready-to-use agents, its purpose is to show you what a finished agent looks like. This section, in contrast, is designed to teach you how to think about building an agent. It explains the architectural principles and the strategic "why" behind each core component, giving you the foundational knowledge needed to effectively use the samples to build your own custom solutions.

To make this practical, let's walk through the process of building a Software Bug Assistant, an LlmAgent agent designed to help a support team triage new bug reports.

1. Define the agent's identity

First, you establish what the agent is and what it's for. This is done with three key parameters:

- **name (required):** A unique string identifier, crucial for internal operations and multi-agent delegation. For our example: software_bug_triage_agent.
- **description (recommended):** A concise summary of capabilities, used by other agents to decide when to route tasks. For our example: "Analyzes new software bug reports, categorizes their priority, and assigns them to the correct engineering team."
- **model (required):** The underlying LLM that powers the agent's reasoning, such as gemini-2.5-flash.

2. Guide the agent with instructions

The instruction parameter is the most critical component for shaping an agent's behavior. It tells the agent its core task, persona, constraints, and how to use its tools. For our Software Bug Assistant, we would direct it to act as an expert engineering manager, explain how to use its tools to look up user data, and specify that its final output should be a JSON object for our ticketing system.

An effective instruction should:

- **Be clear and specific** about the desired outcomes.
- **Provide examples** (few-shot prompting) for complex tasks.
- **Guide tool use** by explaining when and why they should be used.
- **Inject dynamic data** from the agent's state using {variable} syntax.

Pro tip

Be precise, as your entire definition is a prompt

An LLM uses every part of an agent's definition, from its name and description to the names and descriptions of its tools, for reasoning. And it interprets this information with a high degree of literalism. Avoid ambiguous, unclear, or contradictory naming and descriptions, which can lead to "context poisoning," where the agent becomes confused, pursues incorrect goals, or fails to use its tools correctly. Instead, treat every configuration string as a carefully crafted instruction to the model.

Note

AI agents and their underlying frameworks are evolving at an incredible pace. While this guide focuses on the durable architectural principles and patterns for building agentic systems, the specific code snippets and API details that follow are a snapshot in time. We aim to teach the "why" and "how" of agent design, not to provide source code to copy and paste directly into a production solution.

For the most current implementation details, API signatures, and best practices, always consult the official ADK documentation and the ADK Samples repository.

3. Equip the agent with tools

Tools give your agent capabilities beyond its built-in reasoning, allowing it to interact with the outside world. Our Software Bug Assistant would need several tools to do its job, such as:

- A function to get information about the user reporting the bug (get_user_details(user_id)).
- A function to search the codebase for relevant files (search_codebase(file_name)).
- A function to create a ticket in a project management system (create_jira_ticket(...)).

The LLM uses the tool's name, docstring, and parameter schema to decide which tool to call.

Pro tip

Be brief and distinct

Each tool you define adds a new choice for the model to consider. Especially when an agent has many tools, any ambiguity or overlap in their descriptions can confuse the model, leading to looping behaviors or incorrect tool selection. To ensure the model can choose correctly, make each tool's name and description a clear and unique signal of its purpose.

4. Complete the development lifecycle

You're now ready to test and evaluate your agent's performance. This task is all about assessing the quality of the agent's output by examining its step-by-step execution, or "trajectory." The next section covers the important topic of agent evaluation in detail.

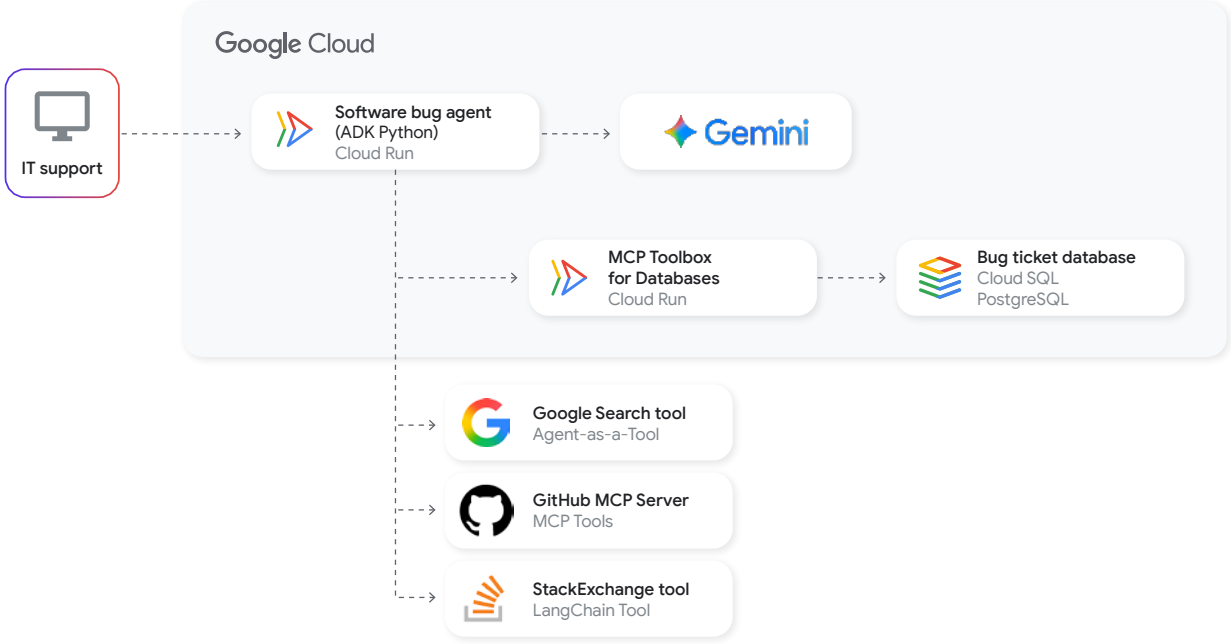
Once you're confident it's performing well, you need a streamlined way to deploy it. This is where your prototype becomes a production-ready application for your team or customers, turning your agent into a reliable business tool.

Pro tip

Test, test, and test again

Agentic systems are non-deterministic and can exhibit emergent behaviors. Standard unit tests are insufficient. Rigorous evaluation is the only way to ensure the quality and reliability of your agent. Focus your testing on two key areas: the reasoning trajectory (the step-by-step logic and tool use) and the final output quality (its accuracy, helpfulness, and grounding). As extensive benchmark testing shows, even state-of-the-art models can produce hallucinations or get stuck in reasoning loops, making continuous evaluation a critical part of the development lifecycle.

Architecture of a software bug assistant with ADK Python



Customer story

How Box uses ADK and the A2A protocol to accelerate content development.

Box is an Intelligent Content Management platform that enables organizations to fuel collaboration, manage the entire content lifecycle, secure critical content, and transform business workflows.

Situation

Critical business processes like compliance checks, contract management, and loan approvals are slowed by employees having to search and interpret vast amounts of information stored across documents in Box. This creates inefficiency and delays critical decisions.

Solution

Box is introducing an A2A-enabled agent, built with Google's ADK and using Gemini. The agent connects directly to the Box Intelligent Content Cloud, allowing users to ask complex questions in natural language and receive summarized, contextual answers and insights from their documents instantly.

Impact

This dramatically accelerates content-centric workflows and improves the quality of decision-making. It lays the foundation for more advanced transactional agents that can govern, manage, and initiate processes like e-signatures and approvals, fundamentally transforming how work gets done in the company.

> "We're entering a new era where AI agents will transform how work gets done—and content is at the center of it all. With Box as the secure content layer and Google Cloud's A2A Protocol enabling seamless collaboration across ecosystem, we're unlocking powerful new ways to automate business processes, accelerate decision-making, and drive real outcomes for our customers."

box

2.3 Govern and scale your agent workforce with Google Agentspace

As your startup moves from building a single agent to deploying a portfolio of specialized agents, you face a new set of challenges: How do you manage them? How do non-technical team members leverage them? How do you govern their access to data and tools?

Google Agentspace solves these scaling problems. This single, secure platform allows you to create, govern, and orchestrate your entire AI agent workforce, unifying disparate applications and data sources. It complements ADK's code-first development by providing the framework to scale agent usage across your entire organization and manage them effectively.

You can use Google Agentspace to:

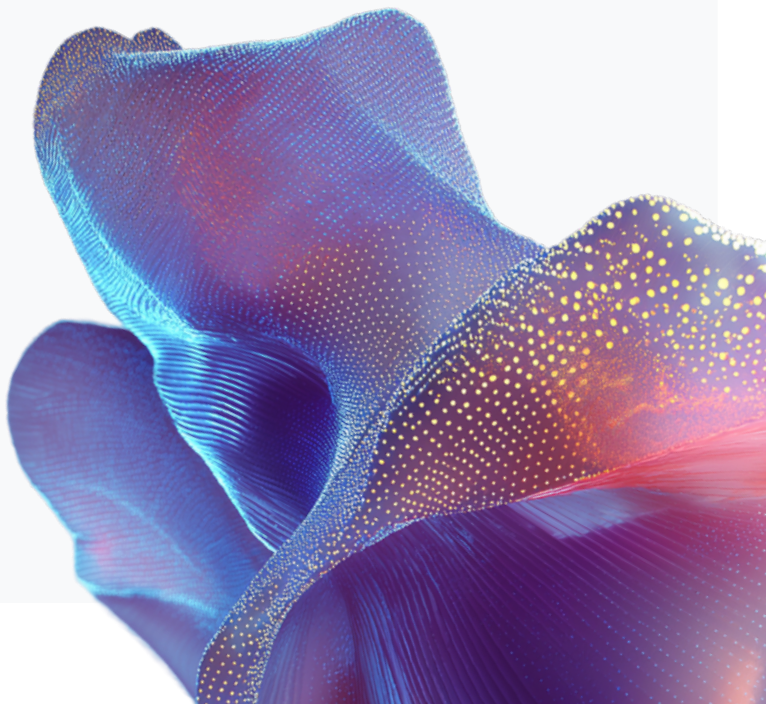
- **Unify and access company data:** Google Agentspace breaks down data silos by using out-of-the-box connectors to your existing company applications (e.g., Microsoft Sharepoint, Google Workspace, Jira). It applies Google's multimodal search technology across this connected data, allowing any employee to get instant answers and synthesize insights from a central source of truth while respecting all existing access controls.
- **Enable team-wide automation:** While ADK is ideal for complex, code-first agent development, Google Agentspace empowers your entire organization to automate workflows. Domain experts in product, marketing, or operations can use the no-code Agent Designer to build their own custom agents using a prompt-driven interface. This turns their specific knowledge into automated solutions without requiring engineering resources.
- **Govern and orchestrate a fleet of agents:** Google Agentspace provides a single platform to manage and govern agents built with ADK, the no-code designer, or from partners. The **Agent Gallery** acts as a central portal for your team to discover, manage, and deploy them all, including both your custom-built solutions and pre-built Google Agents designed for complex tasks like deep research or idea generation.

Try these prompts in Google Agentspace:

> Schedule our weekly team sync for Thursday at 10 AM.
>
> Summarize this week's updates for the #product Slack channel.
>
> Create a meeting agenda to discuss investor prep.

Pro tip

Download the Google Agentspace prompting guide for more templates.



Customer story

How Zoom's AI agent automatically schedules Zoom meeting from Gmail context, with Google Agentspace integration.

Zoom is a communication technology company that provides an open, AI-first work platform used for virtual meetings, webinars, chat, online collaboration, customer experience, and more.

Situation

Zoom's AI-first strategy centers on transforming AI Companion into a fully agentic framework-not only capable of advanced reasoning and task orchestration, but also seamlessly integrated with customer's key third-party systems. By enabling collaboration with other AI agents, Zoom drives more meaningful work outcomes through an open, interoperable ecosystem.

Solution

Zoom AI Companion is integrating with Google Agentspace to streamline meeting scheduling.Launching later this summer, this collaboration will allow A2A-enabled AI agents to automatically schedule Zoom Meetings from Gmail context, update Google Calendar, and keep participants informed, eliminating the back-and-forth of manual scheduling.

Impact

- Reduced technical barriers in cross-platform AI integration.
- Seamless interaction between Zoom AI Companion and external A2A-enabled agents without custom code.
- Enhanced workflow automation and improved efficiency for enterprise customers.
- Future support for more sophisticated multi-agent AI interactions.

> "Our contribution to the A2A protocol enables deeper integration with Google Cloud and other third-party platforms, giving customers flexibility and choice."

zoom

2.4 Other options for building agents

Experiment with Gemini CLI

For startups needing an immediate, cost-effective way to experiment with AI, Gemini CLI is an open-source agent that brings Gemini directly to your terminal. It offers:

- **Significant cost savings:** Get free access to Gemini with generous usage limits (1 million token context, 60 queries per minute).
- **Enhanced productivity:** By integrating into existing developer workflows, it accelerates coding, debugging, and documentation.
- **Total flexibility:** As an open-source tool (Apache 2.0), you can audit, modify, and embed it into your toolchain, avoiding vendor lock-in and enabling deep customization.

Pro tip

Check out Gemini CLI configured for ADK development.

Accelerate development with Firebase Studio

A backend agent, even a powerful one built with ADK, is only one part of a complete product. To bring it to life, you need to build an entire full-stack application around it, including the user interface, databases, and hosting. Firebase Studio is an integrated, cloud-based workspace that uses agentic AI to accelerate the entire development lifecycle. Teams can use it to handle everything from UI prototyping and code generation to secure deployment on Google Cloud's infrastructure.

Together, ADK for the agent's backend logic, the Agent Starter Pack for production infrastructure (which we explore in section 3), and Firebase Studio for the full-stack application provide a complete, end-to-end toolkit for a startup to build and deploy a powerful, state-of-the-art agentic system.

Firebase Studio accelerates the entire development lifecycle with AI:

- **Fast setup:** Use App Prototyping Agent to create a new project using natural language, mockups, or screenshots. Start by selecting from a large catalog of templates for popular frameworks and languages, or import an existing project.
- **Gemini in Firebase:** Use AI assistance on tasks like coding, debugging, testing, refactoring, explaining, and documenting code.
- **Collaboration:** Share workspaces with team members, and provide a URL for early testers to preview apps.
- **Optimize:** Preview apps as users see them with built-in web previews and Android emulators, and test and optimize with access to thousands of extensions in the Open VSX registry.
- **Deploy:** Publish to Firebase App Hosting with a few clicks, or deploy production apps to Cloud Run, Firebase Hosting, or your own custom infrastructure.

Try these prompts with the App Prototyping agent:

> Generate a customer support dashboard that ingests data from Zendesk and displays key metrics like ticket volume and resolution time.

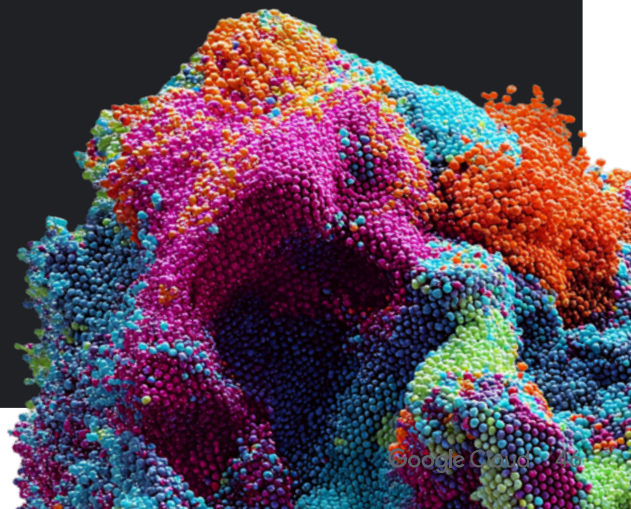
> Create a B2B SaaS application with user authentication, a PostgreSQL database, and a subscription billing page.

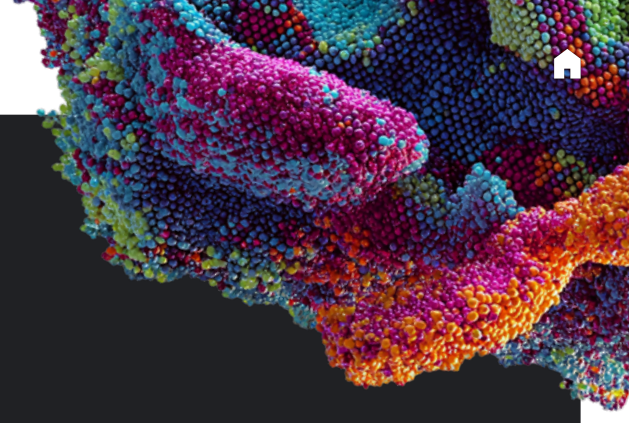
> Build a full-stack application for an internal bug tracking system with a form for submission and a kanban board to view ticket status.

You can also start by selecting from a large catalog of templates for popular frameworks and languages, or import an existing project.

Key takeaways: From build to scale

Your goal	Best option
Build a custom, multi-agent system from code.	Use the open-source Agent Development Kit (ADK).
Deploy, scale, and manage your agent in production.	Deploy it on Vertex AI Agent Engine.
Give your agent the power of long-term memory.	Use the Memory Bank feature within Vertex AI Agent Engine.
Enable your agent to discover and talk to other agents.	Use the Agent2Agent (A2A) protocol, an open standard for agent communication.
Build a complete AI-powered app from a prompt.	Use Firebase Studio for an AI-assisted development workspace.
Quickly experiment with Gemini in your terminal.	Use the Gemini CLI for a simple, command-line interface.





Ready to turn your AI vision into reality? We're here to help.

Learn how to build more generative AI applications with on-demand Startup School sessions.

Start now

Get up to $350k USD in Google Cloud credit with the Google for Startups Cloud Program.

Apply now

Contact our Startups team.

Get in touch

Stay connected and get all our latest updates by subscribing to the Google Cloud Startup Newsletter.

Subscribe

Section 3

Ensuring AI agents are reliable and responsible

Due to the non-deterministic nature of LLM-based systems, it can be hard to achieve production-grade reliability. Moving beyond superficial "vibe-testing" requires a rigorous engineering approach to ensure an agent operates safely and provides consistent value.

This section details the methodologies and tooling necessary to address these challenges, focusing on three key areas:

- **Correctness and reliability:** Evaluating the accuracy of final outputs and the validity of intermediate reasoning steps.
- **Performance and scalability:** Measuring and optimizing agent latency and throughput under load.
- **Safety and responsibility:** Implementing safeguards, monitoring for undesirable behavior, and ensuring the agent operates within defined boundaries.

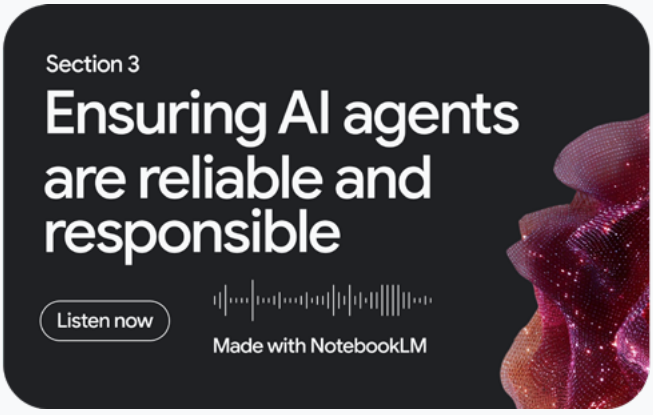
These practices are the tangible application of Google's commitment to responsible AI, allowing startups to build powerful and reliable agents aligned with industry-leading principles for safety.

> "Agents hold the key to a new level of productivity, but their success depends on our guidance."
>
> **Harrison Chase**
> CEO and Co-Founder of LangChain

Pro tip

Production-grade observability means looking beyond application metrics. You must also measure low-level operational metrics like CPU and memory usage. Diligently tracking resource consumption is essential for diagnosing performance bottlenecks, optimizing your runtime, and directly reducing operational costs. ADK and the Agent Starter Pack provide native OpenTelemetry support, allowing you to pipe crucial operational data directly into your existing monitoring tools.

Prefer audio? Listen to the podcast version of this section, created with NotebookLM.



This podcast was created using NotebookLM with the prompt: "As a podcast host, generate a podcast for developers and technical founders. Introduce AgentOps as the framework for building reliable AI, moving beyond informal testing to a rigorous, automated process.

"Explain how AgentOps evaluates an agent's reasoning, accuracy, and safety, and how it mitigates risks like misinformation and security vulnerabilities. Describe how the Agent Starter Pack, working with the Agent Development Kit (ADK), quickly implements this with pre-configured tools for infrastructure, CI/CD, and continuous evaluation. Conclude by highlighting that this disciplined approach is a competitive advantage and direct listeners to Google's resources for startups."

3.1 AgentOps: A framework for production-ready agents

Agent Operations (AgentOps) is an operational methodology that addresses the challenges of reliability and responsibility in production. It adapts the principles of DevOps, MLOps, and DataOps to the unique challenges of building, deploying, and managing AI agents across their lifecycle. And it gives you a systematic, automated, and reproducible framework for handling the complexities of non-deterministic, LLM-based systems in production environments.

A robust AgentOps strategy systematizes the development process, providing continuous feedback loops to improve an agent's reliability, safety, and performance across its tools, reasoning capabilities, and underlying models.

A systematic framework for agent evaluation

Evaluating non-deterministic, agentic systems is one of the most complex challenges in modern software engineering. Traditional testing often focuses on lexical correctness, but agent evaluation must address two harder problems: semantic correctness (did the agent understand and helpfully answer the user's intent?) and reasoning correctness (did the agent follow a logical and efficient path to its conclusion?).

As we discussed in section 1, the cognitive architecture governing this reasoning is often a framework like ReAct, which establishes a dynamic loop where the agent interleaves thought and action. A failure at any point in this loop can lead to an incorrect outcome. Therefore, a rigorous, multi-layered evaluation framework is required. This framework is implemented using a combination of the ADK for core agent logic and instrumentation, and the Agent Starter Pack for production-grade infrastructure, automation, and observability.

Layer 1: Component-level evaluation (deterministic unit tests)

This layer focuses on the predictable, non-LLM components of the agent system.

- **Objective:** To verify the lexical correctness of individual building blocks, ensuring that agent failures don't stem from simple bugs in its components.
- **What to test:**
  - **Tools:** Expected behavior with valid, invalid, and edge-case inputs.
  - **Data processing:** Robustness of parsing and serialization functions.
  - **API integrations:** Handling of success, error, and timeout conditions.
- **Implementation:**
  - ADK defines the agent's tools as Python functions (or Java methods). These functions are the direct subjects of component-level testing.
  - Agent Starter Pack provides the testing infrastructure. It generates a project with a standard pytest environment configured in the tests/unit/ directory. Developers can immediately write unit tests for their ADK-defined tools and run them via the make test command.

Layer 2: Trajectory evaluation (procedural correctness)

This is the most critical layer for evaluating the agent's reasoning process. A "trajectory" is the full sequence of Reason, Act, and Observe steps the agent takes to complete a task.

- **Objective:** To verify the agent's reasoning correctness within the ReAct cycle.
- **What to test:**
  - **Reason step:** Does the agent correctly assess the goal and current state to form a logical hypothesis for the next step?
  - **Act step:** Does it select the correct tool (Tool Selection) and correctly extract and format the arguments for that tool (Parameter Generation)?
  - **Observe step:** Does it correctly integrate the output from the tool to inform the next Reason step of the cycle?
- **Implementation:**
  - ADK's core runtime executes the agent's ReAct loop, integrating directly with Google Cloud Trace to instrument each Reason, Act, and Observe step. This allows developers to visualize the entire trajectory, inspect tool inputs and outputs, and examine the model's chain of thought to debug its reasoning.
  - Agent Starter Pack automates and scales trajectory evaluation. A tests/integration/ directory creates a "golden set" of prompts with expected ReAct trajectories. The automated CI/CD pipeline (set up with agent-starter-pack setup-cicd) runs these tests on every pull request to prevent regressions. The starter pack's observability infrastructure is what captures the trace data emitted by the ADK agent.

Layer 3: Outcome evaluation (semantic correctness)

This layer evaluates the final, user-facing response generated after the ReAct loop has concluded.

- **Objective:** To verify the semantic correctness, factual accuracy, and overall quality of the final answer.
- **What to test:**
  - **Factual accuracy and grounding:** Is the answer correct and verifiably based on information gathered during the Observe steps?
  - **Helpfulness and tone:** Does the response fully address the user's need in the appropriate style?
  - **Completeness:** Does the response contain all necessary information?
- **Implementation:**
  - ADK's toolset is key for verifying factual accuracy. Developers can create specialized tools or use APIs for grounding verification. These tools, called during the Act step, programmatically check if the agent's final answer is supported by the context it retrieved, providing a quantitative measure against hallucination.
  - Agent Starter Pack provides the platform for running these evaluations at scale. It integrates with Vertex AI's Gen AI evaluation service for LLM-as-judge scoring. Its built-in UI playground includes feedback mechanisms that log human ratings directly to BigQuery, enabling high-fidelity HITL evaluation.

Layer 4: System-level monitoring (in-production)

Evaluation doesn't stop at deployment. Continuously monitoring the agent's live performance is critical.

- **Objective:** To track real-world performance and detect operational failures or behavioral drift.
- **What to monitor:** Tool failure rates, user feedback scores, trajectory metrics (e.g., number of ReAct cycles per task), and end-to-end latency.
- **Implementation:**
  - The ADK agent, running in production, is the source of the operational data, emitting events and traces for every live user interaction.
  - Agent Starter Pack provides a production-grade observability stack out-of-the-box. It automatically configures OpenTelemetry, a Log Router to BigQuery, and provides templates for Looker Studio dashboards. This allows teams to immediately track agent performance, analyze trends, and debug issues using data from real-world usage without additional setup.

This comprehensive and practical methodology for agent evaluation is the tangible implementation of a robust AgentOps strategy, moving teams beyond informal "vibe-testing" to a systematic, automated, and reproducible process. By dissecting evaluation into component, trajectory, outcome, and system-level monitoring, it directly addresses the core domains of AgentOps.

Adopting a systematic evaluation framework is not merely a best practice but a competitive advantage. It establishes a rigorous, data-driven, and automated process that allows teams to innovate faster, deploy with confidence, and build agents that are demonstrably safer and more effective.

AgentOps toolkit: ADK and Agent Starter Pack

Automated CI/CD pipelines implement the principles of AgentOps, so that any change to the agent's code, tools, or prompts triggers a standardized process of building, unit testing, and quantitative evaluation against a predefined dataset. This automated evaluation step is essential for preventing regressions and provides continuous, objective feedback on the agent's performance and safety before deployment.

To accelerate adoption of AgentOps principles, the Agent Starter Pack provides a production-ready reference implementation. Its holistic templates address common challenges (e.g., deployment and operations, evaluation, customization, and observability) when building and deploying AI agents. Put simply, it bootstraps a new agent project with the necessary infrastructure and pipelines, so developers can focus on core logic.

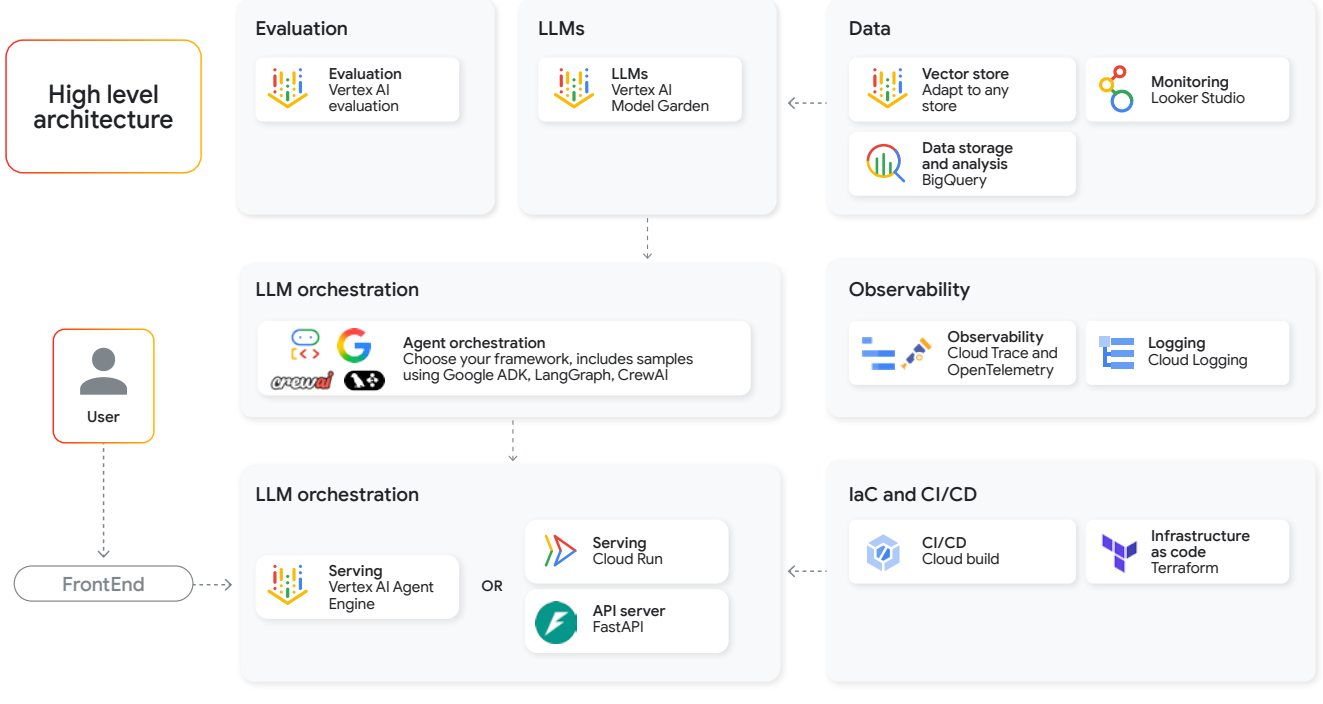
Pro tip

You can create a new, production-ready agent project with a single command: uvx agent-starter-pack create my-agent -a adk@gemini-fullstack.

The Agent Starter Pack includes these key components:

- **Infrastructure as Code (Terraform):** Provides reproducible templates to provision and manage the agent's cloud environment, including services like Cloud Run, IAM permissions, and networking.
- **CI/CD pipelines (Cloud Build):** A pre-configured cloudbuild.yaml file automates the build, unit testing, quantitative evaluation, and deployment process, directly implementing the AgentOps CI/CD workflow.
- **Observability and logging (Cloud Trace and Cloud Logging):** Establishes the foundation for monitoring and debugging by integrating with Cloud Trace for in-depth analysis of agent execution traces and Cloud Logging for centralized log management.
- **Data integration (BigQuery):** Includes foundational components for agents that need to connect to and analyze structured enterprise data using BigQuery.
- **Continuous evaluation (Vertex AI evaluation):** Integrates with Vertex AI to run evaluation datasets against agent changes, continuously measuring performance against the key domains discussed previously.

High-level architecture of the Agent Starter Pack



Better together: ADK and Agent Starter Pack

ADK and the Agent Starter Pack are engineered to provide a clear separation between an agent's application logic and its operational lifecycle, enabling a robust and scalable development process.

Essentially, ADK is used to write the agent's application code, while the Agent Starter Pack provides the production-ready operational baseline to run and manage that code at scale.

- **ADK handles the agent's runtime behavior:** As a Python/Java SDK, ADK provides the APIs and core abstractions for defining an agent's application logic. Developers use it to implement orchestration flows, define tools, and configure interactions with LLMs.
- **The Agent Starter Pack handles the operational environment:** As a scaffolding tool, it generates the infrastructure as code (Terraform) to provision deployment targets (e.g., Cloud Run) and the CI/CD pipeline configurations (Cloud Build) to automate the entire lifecycle.

This separation manifests in a five-step workflow:

1. **Bootstrap with the Agent Starter Pack:** A developer runs a single command to generate a new project containing all necessary operational components, including Terraform files for infrastructure, Cloud Build configurations for CI/CD, and skeleton files for evaluation datasets.
2. **Develop with ADK:** Inside this structure, the developer uses ADK to write the agent's application logic, implementing custom tools, composing agents, and writing the core instructions.
3. **Commit and automate:** When code is committed to the source repository, the pre-configured CI/CD pipeline managed by Cloud Build is automatically triggered.
4. **Evaluate continuously:** The pipeline builds the ADK agent into a container and then executes a quantitative evaluation against a predefined test set, programmatically validating the agent's performance and safety.
5. **Deploy confidently:** Upon a successful evaluation, the pipeline automatically deploys the new, validated version of the agent to its target production environment.

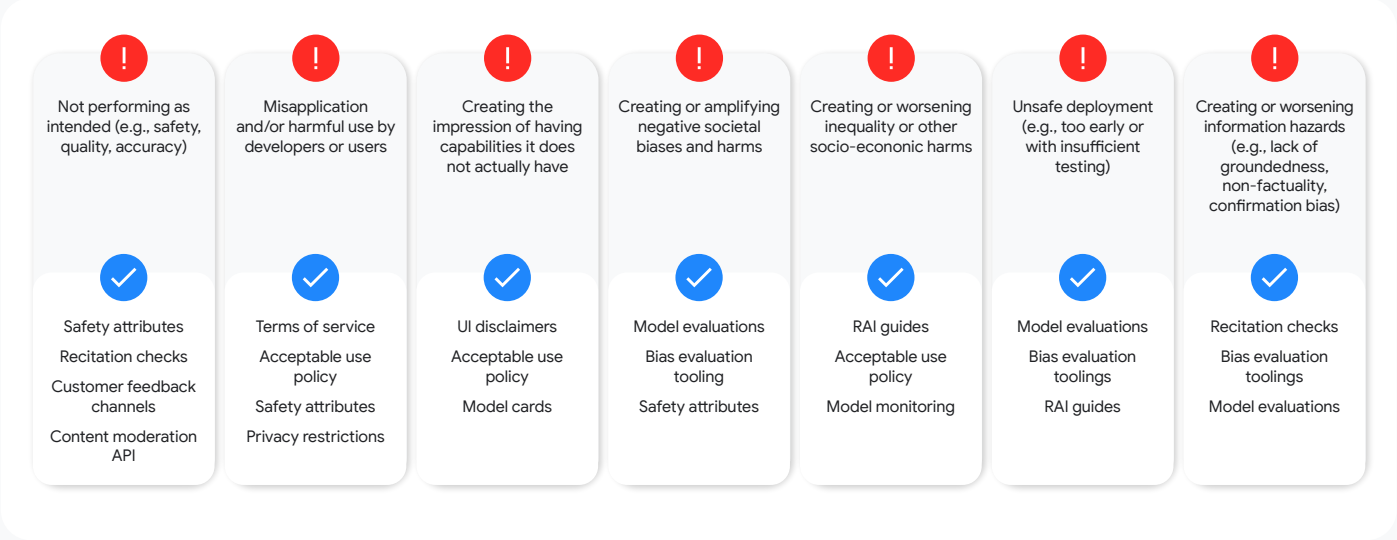
By integrating ADK's development framework with the Agent Starter Pack's operational automation, you establish a complete, end-to-end MLOps/DevOps process specifically tailored for building and managing production-grade AI agents. It's AgentOps at scale.

3.2 Build responsible and secure AI agents with AgentOps

Building powerful agents comes with the non-negotiable responsibility of ensuring they are safe, secure, and aligned. This means designing them from the ground up with safeguards to prevent harmful or unintended outcomes, including unfair bias, privacy violations, and security vulnerabilities.

Addressing this requires a structured approach. The diagram below provides a high-level overview of common risks and the types of technical and procedural safeguards used to mitigate them. While this is a valuable starting point, for a comprehensive guide to standards and best practices, we strongly recommend consulting Google's Secure AI Framework (SAIF).

> **As AI agents integrate into our lives, it's crucial for us to address new challenges around trust, privacy, and security. It's important for us to think about security and privacy, to ask ourselves: how do we build trustworthy products?"**
>
> **Jia Li**
> Co-Founder, President
> and Chief AI Officer of LiveX AI

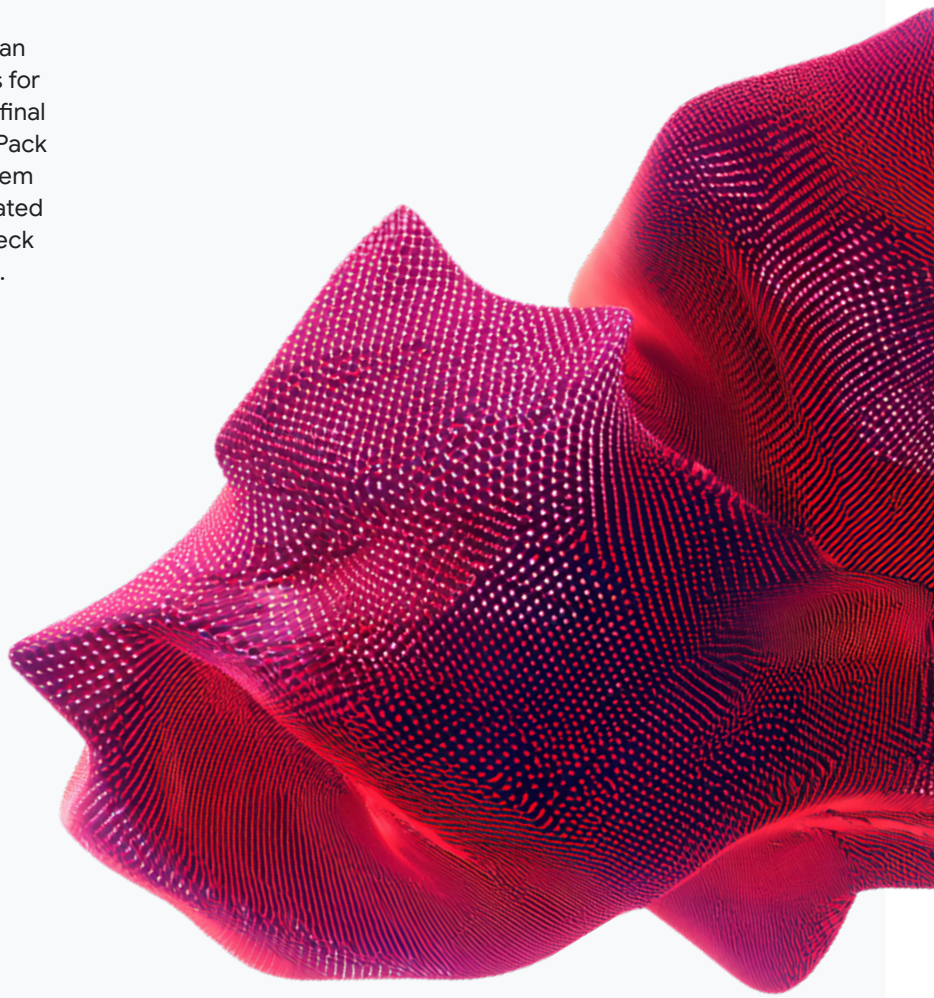


ADK and the Agent Starter Pack deliver a defense-in-depth strategy for this critical area. First, you can implement fine-grained, application-level safety controls with ADK. And, second, the Agent Starter Pack automates the deployment of the hardened cloud infrastructure that enforces these controls at scale.

This combined approach addresses key aspects of safety and compliance:

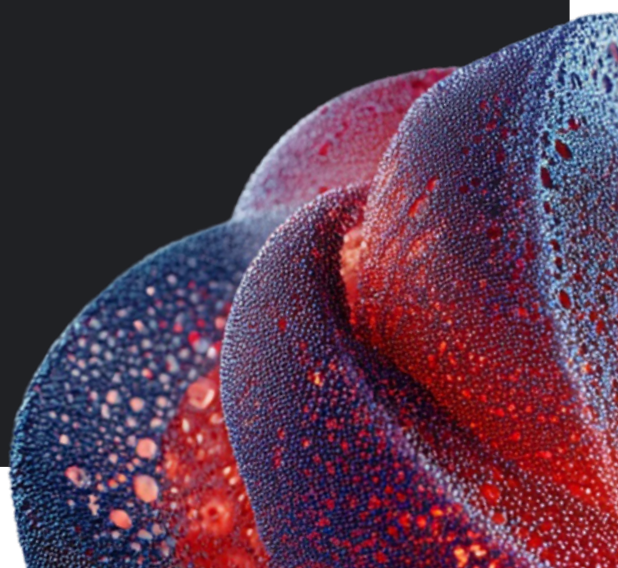
- **Secure infrastructure and access control:** The Agent Starter Pack uses Terraform to provision a secure foundation, deploying your agent to environments like Cloud Run and configuring specific IAM roles to enforce the principle of least privilege. The tools you define in ADK then operate within these strict cloud-level permissions, ensuring the agent cannot access unauthorized resources even if its own logic is compromised.
- **Input and output guardrails:** Within ADK, you can implement application logic to validate prompts for potential injection attacks and filter the agent's final outputs for harmful content. The Agent Starter Pack makes these guardrails robust by integrating them into its CI/CD pipeline. You can then run automated security tests against every code change to check for vulnerabilities before they reach production.
- **Auditing and monitoring:** The detailed observability in ADK creates a granular trace of every thought and tool call the agent makes. The Agent Starter Pack operationalizes this by automatically configuring log sinks that route this data to BigQuery for long-term, secure storage. This creates the durable audit trail necessary for compliance reviews and incident response.

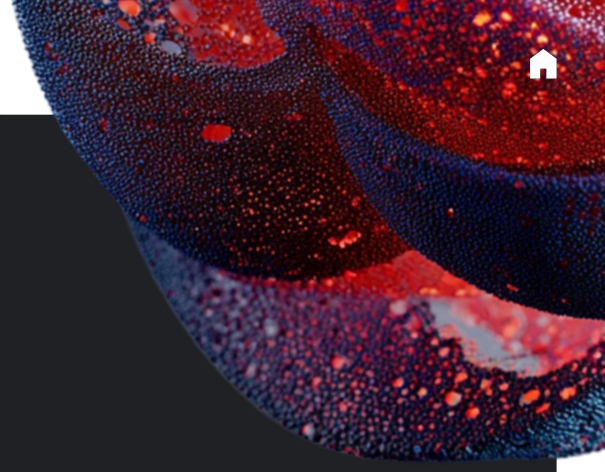
Security is a partnership. While the ADK provides the framework for an agent's cognitive architecture and the Agent Starter Pack provides the components to deploy it, they operate within the larger Google Cloud ecosystem. It all gives you a formidable security posture built on a secure-by-design foundation, with integrated controls designed to defend any workload.



Key takeaways: Building reliable agents

Your goal	Best option
Manage your agent's lifecycle professionally.	Adopt AgentOps to automate processes from development to deployment and monitoring.
Ensure your agent is accurate and safe before going live.	Implement automated evaluation in your CI/CD pipeline to rigorously test for quality, grounding, and safety.
Track your agent's real-world performance, cost, and errors.	Set up monitoring using observability tools to get real-time data on latency, token usage, and tool call success rates.
Figure out why your agent made a specific decision.	Inspect the agent's trajectory (its "chain of thought") using logging and tracing tools to debug its reasoning process.
Secure your agent, its data, and its tool access.	Apply AgentOps security principles, which include infrastructure security, data governance, and compliance controls.
Get started with AgentOps quickly.	Use the Agent Starter Pack for pre-configured templates for CI/CD, evaluation, and infrastructure.





Ready to turn your AI vision into reality? We're here to help.

Learn how to build more generative AI applications with on-demand Startup School sessions.

Start now

Get up to $350k USD in Google Cloud credit with the Google for Startups Cloud Program.

Apply now

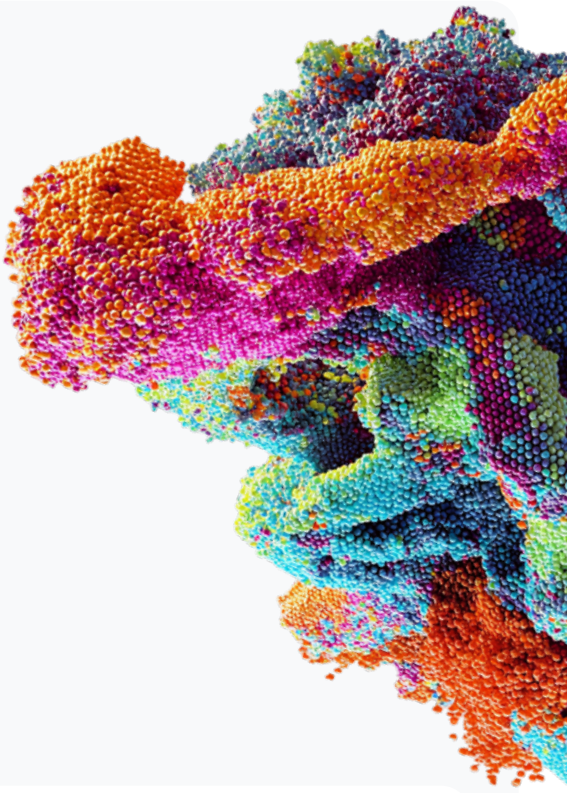
Contact our Startups team.

Get in touch

Stay connected and get all our latest updates by subscribing to the Google Cloud Startup Newsletter.

Subscribe

More from Google's full AI stack



Build quickly with Gemini on Google AI Studio.

See all available Gemini models.

Explore now

Spotlight

Gemini 3 Pro Image (Nano Banana Pro)

This image generation and editing model enables you to blend multiple images into one, maintain character consistency for rich storytelling, make targeted transformations using natural language, and use Gemini's world knowledge to generate and edit images. Nano Banana Pro is for high-value, final production assets while Nano Banana can continue to be used for rapid ideation and speed-sensitive tasks.

Spotlight

Veo and Imagen

These cutting-edge models enable you to generate high-quality videos and images from text prompts, edit existing visuals with natural language, and create immersive storytelling experiences with advanced visual synthesis capabilities.

PROMPT: The man looks up and smiles at the camera.

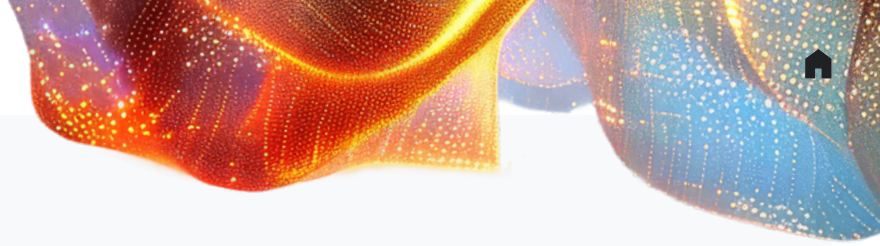
PROMPT: The dog stands up, wagging his tail, happy and looking at the camera.

Conclusion

The journey from prototype to a production-grade system is about disciplined engineering. By using a code-first framework like ADK and the operational principles in this guide, you can move beyond informal “vibe-testing” to a rigorous, reliable process for building and managing your agent’s entire lifecycle.

For your startup, this disciplined approach becomes a powerful competitive advantage. Your team can iterate and innovate faster, automating resource-intensive evaluations along the way. Plus, you can scale with confidence, without compromising on safety or security.

As this guide has shown, Google Cloud supports this innovation, from its purpose-built AI hardware and unified data platform, to the models, services, and tools needed to transition your concept into a sophisticated AI system. The platform is the foundation; your unique vision and the principles outlined in this guide are the blueprint. Together, they form the basis for building the next generation of intelligent systems that will drive your startup forward.



Resources

- AdkApp: Develop and deploy agents on Vertex AI Agent Engine.
- Agent Development Kit (ADK): ADK is a flexible and modular framework for developing and deploying AI agents.
- Agent2Agent (A2A): This is an open protocol enabling communication and interoperability between opaque agentic applications.
- Agent Starter Pack: Get production-ready agents on Google Cloud, faster. Go from idea to deployment faster with pre-built templates and tools.
- BigQuery: BigQuery is Google Cloud's fully managed, petabyte-scale, and cost-effective analytics data warehouse that lets you run analytics over vast amounts of data in near real time.
- Check grounding API: As part of your RAG experience in AI Applications, you can check grounding to determine how grounded a piece of text (called an "answer candidate") is in a given set of reference texts (called "facts").
- Cloud Functions API: This API manages lightweight user-provided functions executed in response to events.
- Cloud Run: Run frontend and backend services, batch jobs, host LLMs, and queue processing workloads without the need to manage infrastructure.
- Cloud Storage bucket: Buckets are the basic containers that hold your data. Everything that you store in Cloud Storage must be contained in a bucket.
- Colab Enterprise: Colab Enterprise is a collaborative, managed notebook environment with the security and compliance capabilities of Google Cloud.
- Example Store: Example Store lets you store and dynamically retrieve few-shot examples.
- Firestore: Firestore is a highly scalable NoSQL database for your web and mobile applications.
- Gemini 2.5 Flash: Gemini 2.5 Flash is designed to control the trade-off between quality, cost, and speed.
- Gemini 3 Pro Image (Nano Banana Pro): Gemini can generate and process images conversationally. You can prompt Gemini with text, images, or a combination of both allowing you to create, edit, and iterate on visuals with unprecedented control
- Gemini 3 Pro: Gemini 3 Pro is our most advanced reasoning Gemini model, capable of solving complex problems.
- Gemini CLI: Free and open source, brings Gemini 2.5 directly into developers' terminals — with unmatched access for individuals.
- Gemma: A collection of lightweight, state-of-the-art open models built from the same technology that powers our Gemini models.
- Gen AI evaluation service: The gen AI evaluation service in Vertex AI lets you evaluate any generative model or application and benchmark the evaluation results against your own judgment, using your own evaluation criteria.
- Google AI Studio: Google AI Studio is the fastest way to start building with Gemini, our next generation family of multimodal generative AI models.
- Google Cloud Observability: Google Cloud Observability includes observability services that help you to understand the behavior, health, and performance of your applications.
- Google Kubernetes Engine (GKE): GKE is the most scalable and fully automated Kubernetes service. Put your containers on autopilot and securely run your enterprise workloads at scale – with little to no Kubernetes expertise required.
- GraphRAG: GraphRAG on Google Cloud combines knowledge graphs with Retrieval-Augmented Generation (RAG) to enhance the accuracy, context, and explainability of large language models (LLMs).
- Imagen: Imagen on Vertex AI brings Google's state-of-the-art image generative AI capabilities to application developers.
- MCP Toolbox for Databases: This is an open source MCP server that helps you build generative AI tools so that your agents can access data in your database.

- Model Context Protocol (MCP): MCP is an open protocol that standardizes how applications provide context to LLMs.
- Model evaluation in Vertex AI: The predictive AI evaluation service lets you evaluate model performance across specific use cases.
- Model Garden on Vertex AI: Jump-start your ML project with a single place to discover, customize, and deploy a wide variety of models from Google and Google partners.
- Model tuning: Model tuning is a crucial process in adapting Gemini to perform specific tasks with greater precision and accuracy.
- ReAct: Orchestration with a ReAct (reasoning + action) agent involves a multi-turn interaction between an application and a model (or models) where the agent manages conversations, transactions, and LLM instructions.
- Responsible AI: To aid developers, the Vertex AI Studio has built-in content filtering, and our generative AI APIs have safety attribute scoring to help customers test Google's safety filters and define confidence thresholds that are right for their use case and business.
- Retrieval-Augmented Generation: RAG is an AI framework that combines the strengths of traditional information retrieval systems (such as search and databases) with the capabilities of generative LLMs.
- Vector database: A vector database is any database that allows you to store, index, and query vector embeddings, or numerical representations of unstructured data, such as text, images, or audio.
- Veo: You can use Veo on Vertex AI to generate new videos from a text prompt or an image prompt.
- Vertex AI Agent Engine: Vertex AI Agent Engine is a set of services that enables developers to deploy, manage, and scale AI agents in production.
- Vertex AI notebooks: Access every capability in Vertex AI Platform to work across the entire data science workflow–from data exploration to prototype to production.
- Vertex AI Platform: Vertex AI is a fully managed, unified AI development platform for building and using generative AI.
- Vertex AI RAG Engine: Vertex AI RAG Engine is a data framework for developing context-augmented LLM applications.
- Vertex AI Search: Vertex AI Search brings together the power of deep information retrieval, state-of-the-art natural language processing, and the latest in LLM processing to understand user intent and return the most relevant results for the user.
- Vertex AI Studio: Streamline your foundation model workflows with Vertex AI Studio. Rapidly prototype, refine, and seamlessly deploy models to your applications.

Google Cloud

Questions? Ask our Startups team.

Contact us