

Get started with Agent Development Kit (ADK)

GENAI104



Overview: Benefits of Agent Development Kit

Agent Development Kit offers several key advantages for developers building agentic applications:

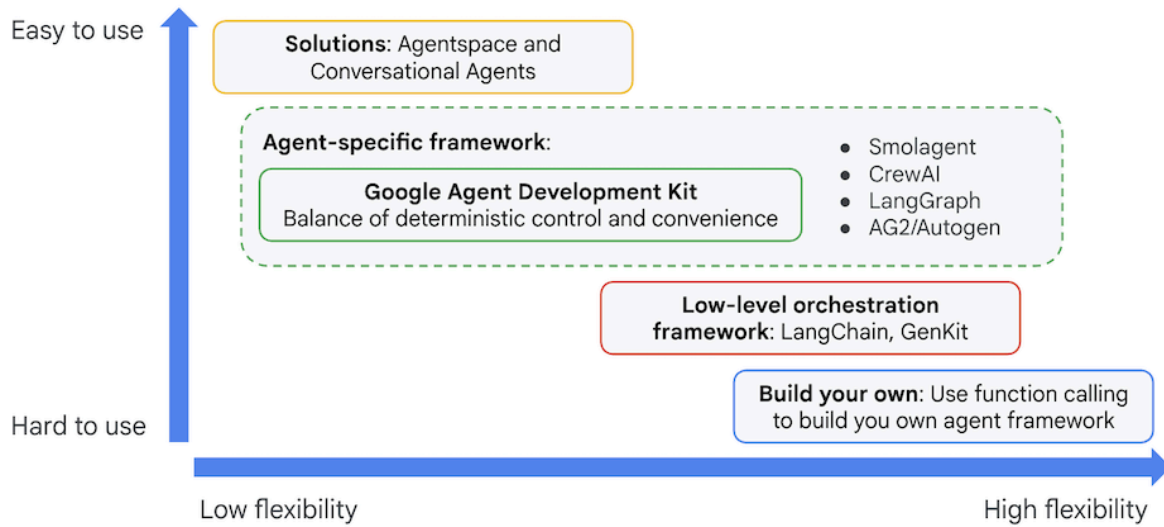
1. **Multi-Agent Systems:** Build modular and scalable applications by composing multiple specialized agents in a hierarchy. Enable complex coordination and delegation.
2. **Rich Tool Ecosystem:** Equip agents with diverse capabilities: use pre-built tools (Search, Code Execution, etc.), create custom functions, integrate tools from third-party agent frameworks (LangChain, CrewAI), or even use other agents as tools.
3. **Flexible Orchestration:** Define workflows using workflow agents (SequentialAgent, ParallelAgent, and LoopAgent) for predictable

pipelines, or leverage LLM-driven dynamic routing (LLMAgent transfer) for adaptive behavior.

4. **Integrated Developer Experience:** Develop, test, and debug locally with a powerful CLI and an interactive dev UI. Inspect events, state, and agent execution step-by-step.
5. **Built-in Evaluation:** Systematically assess agent performance by evaluating both the final response quality and the step-by-step execution trajectory against predefined test cases.
6. **Deployment Ready:** Containerize and deploy your agents anywhere – run locally, scale with Vertex AI Agent Engine, or integrate into custom infrastructure using Cloud Run or Docker.

While other Gen AI SDKs or agent frameworks also allow you to query models and even empower them with tools, dynamic coordination between multiple models requires a significant amount of work on your end.

Agent Development Kit offers a higher-level framework than these tools, allowing you to easily connect multiple agents to one another for complex but easy-to-maintain workflows.



Additionally, it allows you to deploy these complex systems of agents to a fully-managed endpoint in Agent Engine, so you can focus on the agents' logic while infrastructure is allocated and scaled for you.

Objectives

In this lab, you will create a single agent that can use a search tool. You will test agents in ADK's browser UI, from a CLI chat interface, and programmatically from within a script.

You will consider:

- The key capabilities of Agent Development Kit
- The core concepts of ADK
- How to structure project directories for ADK
- The most fundamental parameters of agents in ADK, including how to specify model names and tools

- Some features of ADK's browser UI
- How to control the output schema of an agent
- How to run agents in three ways (via the browser UI, programmatically, and via the CLI chat interface)

Setup and requirements

Before you click the **Start Lab** button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

What you need

To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.

Note: If you already have your own personal Google Cloud account or project, do not use it for this lab.

Note: If you are using a Pixelbook, open an Incognito window to run this lab.

How to start your lab and sign in to the Google Cloud console

1. Click the **Start Lab** button. If you need to pay for the lab, a dialog opens for you to select your payment method. On the left is the Lab Details pane with the following:

- The Open Google Cloud console button
- Time remaining
- The temporary credentials that you must use for this lab
- Other information, if needed, to step through this lab

2. Click **Open Google Cloud console** (or right-click and select **Open Link in Incognito Window** if you are running the Chrome browser).

The lab spins up resources, and then opens another tab that shows the Sign in page.

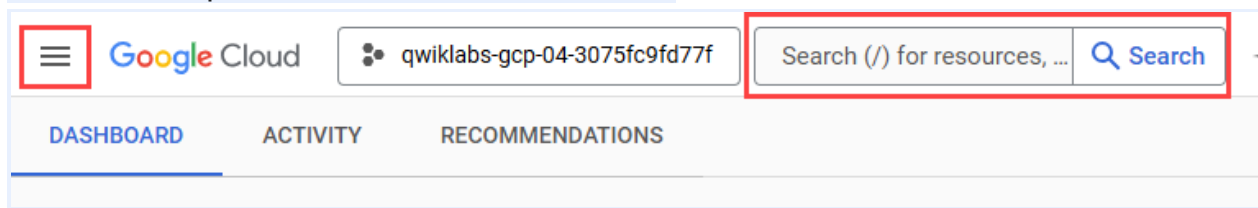
Tip: Arrange the tabs in separate windows, side-by-side.

3. **Note:** If you see the **Choose an account** dialog, click **Use Another Account**.
4. If necessary, copy the **Username** below and paste it into the **Sign in** dialog.
5. "Username"
6. You can also find the Username in the Lab Details pane.
7. Click **Next**.
8. Copy the **Password** below and paste it into the **Welcome** dialog.
9. "Password"
10. You can also find the Password in the Lab Details pane.
11. Click **Next**.
12. **Important:** You must use the credentials the lab provides you. Do not use your Google Cloud account credentials.
13. **Note:** Using your own Google Cloud account for this lab may incur extra charges.
14. Click through the subsequent pages:
 - Accept the terms and conditions.

- Do not add recovery options or two-factor authentication (because this is a temporary account).
- Do not sign up for free trials.

After a few moments, the Google Cloud console opens in this tab.

Note: To access Google Cloud products and services, click the **Navigation menu** or type the service or product name in the **Search** field.



Task 1. Install ADK and set up your environment


Note: Using an Incognito browser window is recommended for most Qwiklabs to avoid confusion between your Qwiklabs student account and other accounts logged into Google Cloud. If you are using Chrome, the easiest way to accomplish this is to close any Incognito windows, then right click on the **Open Google Cloud console** button at the top of this lab and select **Open link in Incognito window**.

Enable Vertex AI recommended APIs


1. In this lab environment, the **Vertex AI API has been enabled for you**. If you were to follow these steps in your own project, you could enable it by navigating to Vertex AI and following the prompt to enable it.

Prepare a Cloud Shell Editor tab

1. With your Google Cloud console window selected, open Cloud Shell by pressing the **G** key and then the **S** key on your keyboard. Alternatively, you can click the

Activate Cloud Shell button () in the upper right of the Cloud console.

2. Click **Continue**.
3. When prompted to authorize Cloud Shell, click **Authorize**.
4. In the upper right corner of the Cloud Shell Terminal panel, click the **Open in**

new window button  .

5. In the Cloud Shell Terminal, enter the following to open the Cloud Shell Editor to your home directory:
 - a. `cloudshell workspace ~`
6. Close any additional tutorial or Gemini panels that appear on the right side of the screen to save more of your window for your code editor.
7. Throughout the rest of this lab, you can work in this window as your IDE with the Cloud Shell Editor and Cloud Shell Terminal.

Download and install the ADK and code samples for this lab

1. Update your PATH environment variable and **install ADK** by running the following commands in the Cloud Shell Terminal. *Note: You will specify the version to ensure that the version of ADK that you install corresponds to the version used in this lab:*

```
export PATH=$PATH:"/home/${USER}/.local/bin"
```

2. `python3 -m pip install google-adk`
3. Paste the following commands into the Cloud Shell Terminal to copy a file from a Cloud Storage bucket, and unzip it, creating a project directory with code for this lab:

```
gcloud storage cp gs://YOUR_GCP_PROJECT_ID-bucket/adk_project.zip  
./adk_project.zip
```

4. `unzip adk_project.zip`
5. Install additional lab requirements with:
6. `python3 -m pip install -r adk_project/requirements.txt`

Core Concepts of Agent Development Kit

Google ADK is built around a few core concepts that make it powerful and flexible:

- **Agent:** Agents are core building blocks designed to accomplish specific tasks. They can be powered by LLMs to reason, plan, and utilize tools to achieve goals, and can even collaborate on complex projects.
- **Tools:** Tools give agents abilities beyond conversation, letting them interact with external APIs, search information, run code, or call other services.
- **Session Services:** Session services handle the context of a single conversation (Session), including its history (Events) and the agent's working memory for that conversation (State).
- **Callbacks:** Custom code snippets you provide to run at specific points in the agent's process, allowing for checks, logging, or behavior modifications.
- **Artifact Management:** Artifacts allow agents to save, load, and manage files or binary data (like images or PDFs) associated with a session or user.

- **Runner:** The engine that manages the execution flow, orchestrates agent interactions based on Events, and coordinates with backend services.

Task 2. Review the structure of Agent Development Kit project directories

1. In the Cloud Shell Editor's file explorer pane, find the **adk_project** folder. Click it to toggle it open.
2. This directory contains three other directories: **my_google_search_agent**, **app_agent**, and **llm_auditor**. Each of these directories represents a separate agent. Separating agents into their own directories within a project directory provides organization and allows Agent Development Kit to understand what agents are present.
3. Click on the **my_google_search_agent** to explore an agent directory.
4. Notice that the directory contains an `__init__.py` file and an `agent.py` file. An `__init__.py` file is typically used to identify a directory as a Python package that can be imported by other Python code. **Click the `init.py` file** to view its contents.
5. Notice that the `__init__.py` file contains a single line, which imports from the `agent.py` file. ADK uses this to identify this directory as an agent package:
6. `from . import agent`
7. Now click on the `agent.py` file. This file consists of a simple agent. You will equip it with a powerful tool: the ability to search the internet using Google Search. Notice a few things about the file:
 - Notice the imports from `google.adk`: the `Agent` class and the `google_search` tool from the `tools` module

- Read the code comments that describe the parameters that configure this simple agent.
8. To use the imported `google_search` tool, it needs to be passed to the agent. Do that by **pasting the following line** into the `agent.py` file where indicated at the end of the Agent object creation:
 9.

```
tools=[google_search]
```
 10. **Save** the file.

Tools enable an agent to perform actions beyond generating text. In this case, the `google_search` tool allows the agent to decide when it would like more information than it already has from its training data. It can then write a search query, use Google Search to search the web, and then base its response to the user on the results. When a model bases its response on additional information that it retrieves, it is called "grounding," and this overall process is known as "retrieval-augmented generation" or "RAG."

You can learn more about how to use tools with ADK in the lab *Empower ADK agents with tools*.

Task 3. Run the agent using the ADK's Dev UI

ADK includes a development UI designed to run locally to help you develop and test your agents. It can help you visualize what each agent is doing and how multiple agents interact with one another. You will explore this interface in this task.

When you run an agent, the ADK needs to know who is requesting the model API calls.

You can provide this information in one of two ways. You can:

1. Provide a [Gemini API key](#).
2. Authenticate your environment with Google Cloud credentials and associate your model API calls with a Vertex AI project and location.

In this lab, you will take the Vertex AI approach.

1. In the Cloud Shell Editor menus, select **View > Toggle Hidden Files** to view or hide your hidden files (files with a period at the start of their filename are hidden by default in most file systems). You may need to scroll down in this menu to find the Toggle Hidden Files option.
2. In the Cloud Shell Editor file explorer pane, navigate to the **adk_project/my_google_search_agent** directory.
3. Select the **.env** file in the **my_google_search_agent** directory.
4. Paste these values over what is currently in the file to **update the file to include your project ID**:

```
GOOGLE_GENAI_USE_VERTEXAI=TRUE
GOOGLE_CLOUD_PROJECT=YOUR_GCP_PROJECT_ID
GOOGLE_CLOUD_LOCATION=GCP_LOCATION
MODEL=gemini-2.5-flash
```

5. **Save** the file.

These variables play the following roles:

- `GOOGLE_GENAI_USE_VERTEXAI=TRUE` indicates that you will use Vertex AI for authentication as opposed to Gemini API key authentication.

- `GOOGLE_CLOUD_PROJECT` and `GOOGLE_CLOUD_LOCATION` provide the project and location with which to associate your model calls.
- `MODEL` is not required, but is stored here so that it can be loaded as another environment variable. This can be a convenient way to try different models in different deployment environments.

When you test your agent using ADK's dev UI or the command-line chat interface, they will load and use an agent's `.env` file if one is present or else look for environment variables with the same names as those set here.`

6. In the Cloud Shell Terminal, ensure you are in the **adk_project** directory where your agent subdirectories are located by running:

- a. `cd ~/adk_project`

7. **Launch the Agent Development Kit Dev UI** with the following command:

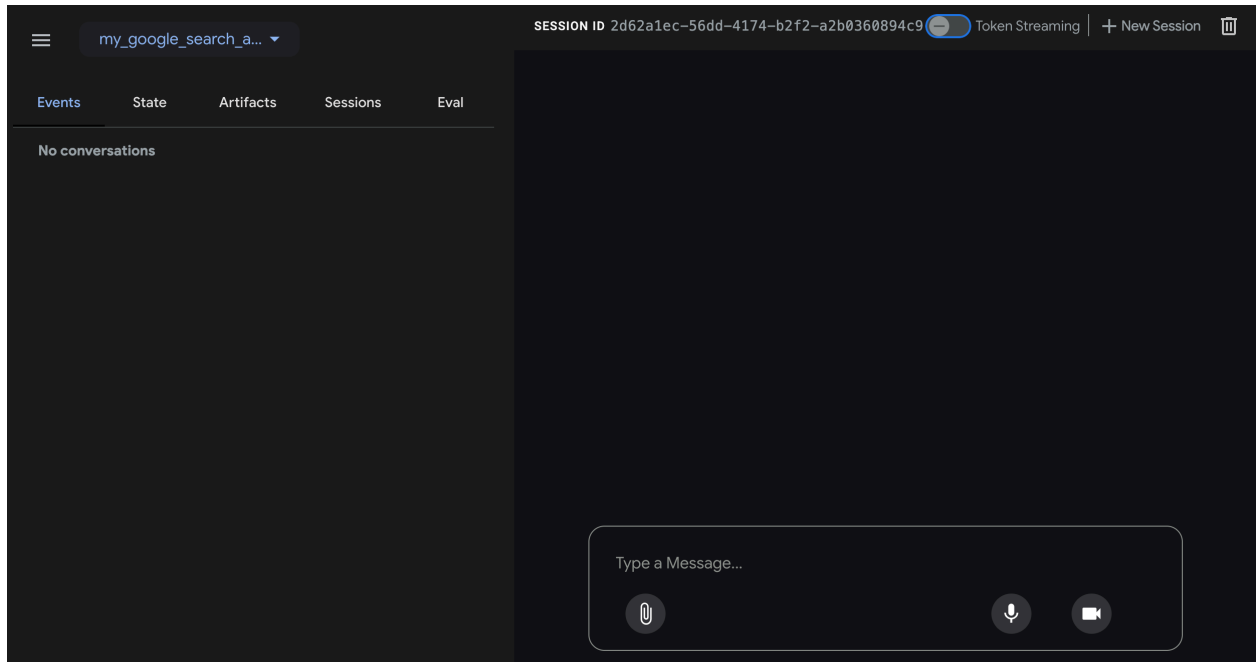
- a. `adk web`

Output

```
INFO:      Started server process [2434]
INFO:      Waiting for application startup.
+-----+
| ADK Web Server started |
|                         |
| For local testing, access at http://localhost:8000. |
+-----+

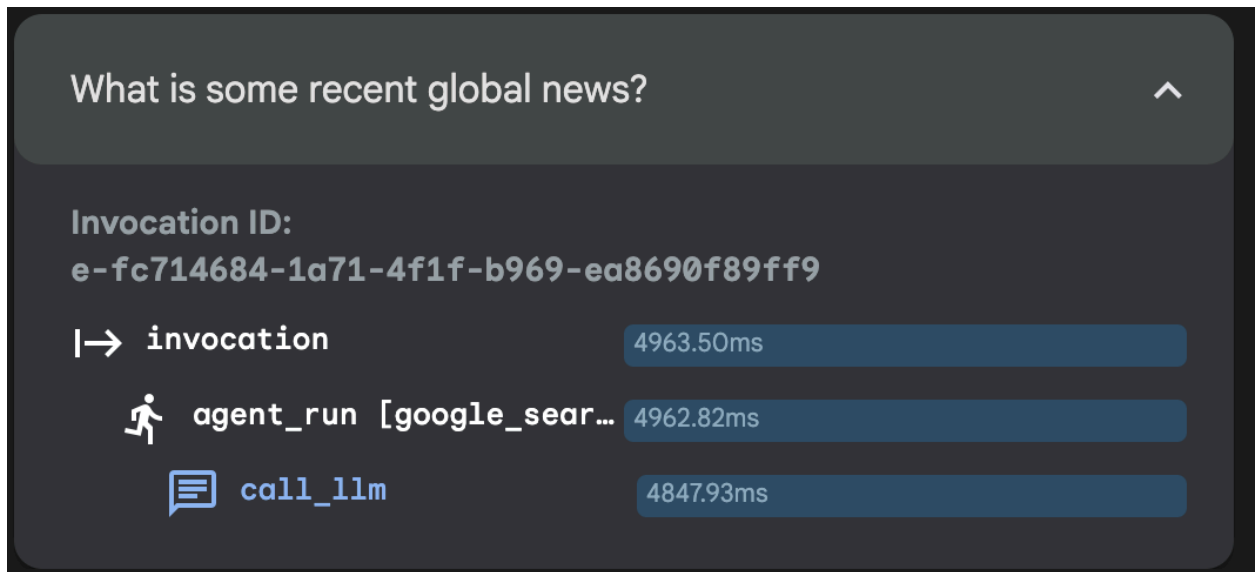
INFO:      Application startup complete.
INFO:      Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

8. To view the web interface in a new tab, click the **http://127.0.0.1:8000** link in the Terminal output, which will link you via proxy to this app running locally on your Cloud Shell instance.
9. A new browser tab will open with the ADK Dev UI.
10. From the **Select an agent** dropdown on the left, select **my_google_search_agent**.



11. In order to encourage the agent to use its Google Search tool, enter the question:
 - a. What **is some** recent **global** news?
12. You will notice from the results that the agent is able to use Google Search to get up-to-date information, rather than having its information stop on the date when its model was trained.
 - a. A response using grounding with Google search includes ready-to-display HTML "Search Suggestions" like those you see at the bottom of the agent's response. When you use grounding with Google Search, you are **required to display these suggestions**, which help users follow up on the information the model used for its response.

13. Notice that in the left side bar, you are in the **Trace** tab by default. Click on your last query text (What is some recent global news?) to see a trace of how long different parts of your query took to execute. You can use this to debug more complex executions involving tool calls to understand how various processes contribute to the latency of your responses.



14. Click the agent icon (👤) next to the agent's response (or an event from the list on the **Events** tab) to inspect the event returned by the agent, which includes the content returned to the user and groundingMetadata which details the search results that the response was based on.
15. When you are finished exploring the dev UI, close this browser tab and return to your browser tab with the Cloud Shell Terminal, click on the terminal's pane, and press CTRL + C to stop the web server.

Click **Check my progress** to verify the objective.

Run the agent using the ADK's Web UI.

Check my progress

Task 4. Run an agent programmatically

While the dev UI is great for testing and debugging, it is not suitable for presenting your agent to multiple users in production.

To run an agent as part of a larger application, you will need to include a few additional components in your **agent.py** script that the web app handled for you in the previous task. Proceed with the following steps to open a script with these components to review them.

1. In the Cloud Shell Terminal run the following commands to export environment variables. You can use this approach to set environment variables for all of your agents to use if they do not have a `.env` file in their directory:

```
export GOOGLE_GENAI_USE_VERTEXAI=TRUE
export GOOGLE_CLOUD_PROJECT=YOUR_GCP_PROJECT_ID
export GOOGLE_CLOUD_LOCATION=GCP_LOCATION
export MODEL=gemini-2.5-flash
```

2. In the Cloud Shell Editor file browser, select the **adk_project/app_agent** directory.
3. Select the **agent.py** file in this directory.
4. This agent is designed to run as part of an application. Read the commented code in **agent.py**, paying particular attention to the following components in the code:
- 5.

Component	Feature	Description

<p data-bbox="253 245 544 279">InMemoryRunner()</p>	<p data-bbox="998 245 1101 489">Oversight of agent execution</p>	<p data-bbox="1182 245 1370 1755">The Runner is the code responsible for receiving the user's query, passing it to the appropriate agent, receiving the agent's response event and passing it back to the calling application or UI, and then triggering the following event. You can read more in the ADK <a data-bbox="1229 1675 1370 1755" href="#">documentation about</p>
---	--	--

		the event loop.
<code>runner.session_service.create_session()</code>	Conversation history & shared state	Sessions allow an agent to preserve state, remembering a list of items, the current status of a task, or other 'current' information. This class creates a local session service for simplicity, but in production this could be handled by a database.

<code>types.Content()</code> and <code>types.Part()</code>	Structured, multimodal messages	Instead of a simple string, the agent is passed a Content object which can consist of multiple Parts. This allows for complex messages, including text and multimodal content to be passed to the agent in a specific order.
--	---------------------------------	--

When you ran the agent in the dev UI, it created a session service, artifact service, and runner for you. When you write your own agents to deploy programmatically, it is recommended that you provide these components as external services rather than relying on in-memory versions.

5. Notice that the script includes a hardcoded query, which asks the agent: "What is the capital of France?"
6. Run the following command in the Cloud Shell Terminal to run this agent programmatically:
 - a. `python3 app_agent/agent.py`
7. **Selected Output:**
 - a. `trivia_agent: The capital of France is Paris.`
8. You can also define specific input and/or output schema for an agent.
You will now add imports for the [Pydantic schema classes](#) `BaseModel` and

Field and use them to define a schema class consisting of just one field, with a key of "capital" and a string value intended for the name of a country's capital city. You can paste these lines into your **app_agent/agent.py** file, just after your other imports:

```
from pydantic import BaseModel, Field

class CountryCapital(BaseModel):
    capital: str = Field(description="A country's capital.")
```

9. **Important note:** When you define an output schema, you cannot use tools or agent transfers.

10. Within your root_agent's Agent definition, add these parameters to disable transfers (as you are required to do when using an output schema) and to set the output to be generated according to the CountryCapital schema you defined above:

```
disallow_transfer_to_parent=True,
disallow_transfer_to_peers=True,
output_schema=CountryCapital,
```

11. Run the agent script again to see the response following the output_schema:

a. `python3 app_agent/agent.py`

12. **Selected Output:**

```
** User says: {'parts': [{'text': 'What is the capital of France?'}]},
{'role': 'user'}
** trivia_agent: {"capital": "Paris"}
```

Click **Check my progress** to verify the objective.

Run an agent programmatically.

Check my progress

Task 5. Chat with an agent via the command-line interface

You can also chat with an agent in your local development environment by using the command line interface. This can be very handy for quickly debugging and testing agents as you develop them.

Like the web interface, the command line interface also handles the creation of the session service, artifact service, and runner for your agent.

To run an interactive session using the command line interface:

1. **Run** the following in Cloud Shell Terminal:
 - a. `adk run my_google_search_agent`
2. **Output:**

```
Log setup complete: /tmp/agents_log/agent.20250322_010300.log
To access latest log: tail -F /tmp/agents_log/agent.latest.log
Running agent basic_search_agent, type exit to exit.
user:
```

3. Input the following message:
4. what are some new movies that have been released in the past month in India?

5. Example output (yours may be a little different):

```
[google_search_agent]: Here are some movies that have been released in India in the past month (August-September 2025):  
* **Param Sundari** Released on 29 August 2025.  
* **Ek Chatur Naar** Released on 12 September 2025.  
* **Jolly LLB 3** Released on 19 September 2025.  
* **Ajey: The Untold Story of a Yogi** Released on 19 September 2025.  
* **Antha 7 Naatkal** Released on Sep 25 2025.
```

6. When you are finished chatting with the command line interface, **enter exit** at the next user prompt to end the chat.

Click **Check my progress** to verify the objective.

Chat with an agent via the command-line interface.

Check my progress

Task 6. Preview a multi-agent example

You will learn more about building multi-agent systems in the lab *Build multi-agent systems with ADK*, but because multi-agent capabilities are core to the Agent Development Kit experience, you can explore one multi-agent system now.

This agentic system evaluates and improves the factual grounding of responses generated by LLMs. It includes: - a `critic_agent` to serve as an automated fact-checker - a `reviser_agent` to rewrite responses if needed to correct inaccuracies based on verified findings


To explore this agent:

1. To explore this multi-agent system's code, use the Cloud Shell Editor file explorer to navigate to the directory **adk_project/llm_auditor**.
2. Within the **llm_auditor** directory, select the **agent.py** file.
3. Here are a few things to notice about this multi-agent example:
 - Notice the import and use of the `SequentialAgent` class. This is an example of a **workflow class** which passes control of the conversation from one agent to the next in order without awaiting a user turn in-between. When you run the agent, you will see responses from both the `critic_agent` and the `reviser_agent`, in that order, without waiting for a user turn.
 - Notice that these sub-agents are each imported from their own directories within a `sub_agents` directory.
 - In the sub-agents' directories, you will see the `__init__.py` and `agent.py` files like those you explored in the directory structure earlier, along with a `prompt.py` file, which provides a dedicated place for a complete, well-structured prompt to be stored and edited before it is imported into the `agent.py` file.
4. Create a `.env` file for this agent and launch the dev UI again by running the following in the Cloud Shell Terminal:

```
cd ~/adk_project
cat << EOF > llm_auditor/.env
GOOGLE_GENAI_USE_VERTEXAI=TRUE
GOOGLE_CLOUD_PROJECT=YOUR_GCP_PROJECT_ID
GOOGLE_CLOUD_LOCATION=GCP_LOCATION
MODEL=gemini-2.5-flash
EOF
```

```
adk web
```

Note: If you did not shut down your previous `adk web` session, the default port of 8000 will be blocked, but you can launch the Dev UI with a new port by using `adk web --port 8001`, for example.

5. Click the **`http://127.0.0.1:8000`** link in the Terminal output. A new browser tab will open with the ADK Dev UI.
6. From the **Select an agent** dropdown on the left, select **llm_auditor**.
7. Start the conversation with the following false statement:
 - Double **check** this: Earth is further away from the Sun than Mars.
8. You should see two responses from the agent in the chat area:
 - First, a detailed response from the `critic_agent` checking the truthfulness of the statement based on fact-checking with Google Search.
 - Second, a short revised statement from the `reviser_agent` with a corrected version of your false input statement, for example, "Earth is closer to the Sun than Mars."
9. Next to each response, click on the agent icon () to open the event panel for that response (or find the corresponding numbered event on the Events panel and select it). At the top of the event view, there is a graph that visualizes the relationships between the agents and tools in this multi-agent system. The agent responsible for this response will be highlighted.
10. Feel free to explore the code further or ask for other fact-checking examples in the dev UI. Another example you can try is:
11. Q: Why is **the** sky blue? A: Because **the** sky reflects **the** color **of the** ocean.
12. If you would like to reset the conversation, use the **+ New Session** link at the top right of the ADK Dev UI to restart the conversation.
13. When you are finished asking questions of this agent, close the browser tab and press **CTRL + C** in the Terminal to stop the server.

Human-in-the-loop pattern

Even though this example uses a `SequentialAgent` workflow agent, you can think of this pattern as a human-in-the-loop pattern. When the `SequentialAgent` ends its sequence, the conversation goes back to its parent, the `llm_auditor` in this example, to get a new input turn from the user and then pass the conversation back around to the other agents.

Click **Check my progress** to verify the objective.

Preview a multi-agent example.

Check my progress

Congratulations!

In this lab, you've learned:

- The key capabilities of Agent Development Kit
- The core concepts of ADK
- How to structure project directories for ADK
- The most fundamental parameters of agents in ADK, including how to specify model names and tools
- Some features of ADK's browser UI
- How to control the output schema of an agent
- How to run agents in three ways (via the browser UI, programmatically, and via the CLI chat interface)

Next Steps

To learn more about building and deploying agents using Agent Development Kit, check out these labs:

- Empower ADK agents with tools
- Build multi-agent systems with ADK
- Deploy ADK agents to Agent Engine

Find more information about ADK in the documentation and GitHub repository:

- Agent Development Kit Documentation [Agent Development Kit](#)
- adk-docs GitHub repository [adk-docs](#)

Google Cloud training and certification

...helps you make the most of Google Cloud technologies. [Our classes](#) include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. [Certifications](#) help you validate and prove your skill and expertise in Google Cloud technologies.

Manual Last Updated August 04, 2025

Lab Last Tested August 04, 2025

Copyright 2023 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are associated.