Google Skills

Three deployment methods

In modules 1 and 2, you learned to run agents using adk web, which is the visual web interface for development and testing. You've been using commands like:

Shell
```
adk web
```

This opens a browser-based UI where you can chat with your agent interactively.

But `adk web` isn't the only way to run agents. In this module, you'll learn three additional methods:

- `adk run` – For terminal-based execution
- `adk api_server` – For deploying as an API service
- **Programmatic execution** – For integrating agents into Python applications

Method 1: Terminal execution with `adk run`

Reference: ADK docs – Run Your Agent

What it is

`adk run` allows you to interact with your agent directly from the terminal, without opening a web browser.

How to use it

From your agent directory:

Shell
```
# Navigate to your agent project
directory
cd my_first_agent

# Run the agent
adk run
```

Expected behavior:

- The terminal becomes interactive
- Type your message, and press **Enter**
- The agent responds in the terminal
- Type another message to continue the conversation
- Press `Ctrl+C` to exit

From the parent directory:

Shell
```
# If you're in the parent directory (e.g., adk-workspace)
adk run my_first_agent
```

Example interaction

Shell
```
$ adk run my_first_agent

You: How do I solve x + 5 = 10?
Agent: Great question! Let's work through this together. What do you think we
need to do to get x by itself on one side?

You: Subtract 5 from both sides?
Agent: Exactly! That's the right approach. When we subtract 5 from both sides...
```

When to use adk run

Good for:

- ✓ Quick testing during development
- ✓ Command-line workflows
- ✓ Server environments without GUI
- ✓ Automated testing scripts
- ✓ CI/CD pipelines

Not ideal for:

- ✗ Presenting to stakeholders (use adk web instead)
- ✗ Debugging complex conversations (use adk web instead)

Method 2: API Server with `adk api_server`

Reference: ADK docs – Agent Config: Run

What it is

`adk api_server` runs your agent as a REST API service, allowing other applications to send requests to your agent over HTTP.

How to use it

Start the API server:

```shell
# From your agent directory
cd my_first_agent
adk api_server
```

Or from parent directory:

```shell
adk api_server my_first_agent
```

Expected output:

```
INFO:     Started server process [12345]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

Testing with cURL

Once the server is running,
you can send requests using `curl`:

```shell
# In a separate terminal window
curl -X POST http://localhost:8000/
your-endpoint \
  -H "Content-Type: application/json"
\
  -d '{"message": "What is 2x + 5 =
13?"}'
```

Note: For detailed API usage,
see the ADK testing documentation.

When to use `adk api_server`

Good for:

- ✓ Integrating agents into web applications
- ✓ Mobile app backends
- ✓ Microservices architectures
- ✓ Pre-production testing
- ✓ Local API development before deploying to Cloud Run

Not ideal for:

- ✗ Interactive development (use `adk web` instead)
- ✗ Quick testing (use `adk run` instead)

Method 3: Programmatic execution with Python

Reference: ADK docs – Agent Team Tutorial

What it is

You can run agents directly in your Python code, giving you full programmatic control. This is useful for:

- Building custom applications with agents
- Jupyter notebooks/Google Colab
- Data processing pipelines
- Custom integrations

Complete example (copy-paste ready)

This example is self-contained and ready to run in a Python script or Jupyter notebook:

```python
"""
Complete example: Running an ADK agent programmatically
Copy this entire code block to run it in a Python script or notebook.
"""

# Step 1: Install ADK (run this in terminal or notebook cell)
# pip install google-adk

# Step 2: Set your API key
# Option A: Set as environment variable before running
#   export GOOGLE_API_KEY=your-api-key-here
# Option B: Uncomment and use this code:
# import os
# os.environ['GOOGLE_API_KEY'] = 'your-api-key-here'
# os.environ['GOOGLE_GENAI_USE_VERTEXAI'] = 'FALSE'

# Step 3: Import required libraries
import asyncio
from google.adk.agents.llm_agent import Agent
from google.adk.runners import Runner
from google.adk.sessions import InMemorySessionService
from google.genai.types import Content, Part

# Step 4: Define your agent
agent = Agent(
    model='gemini-2.5-flash',
    name='math_tutor',
    instruction="""You are a patient math tutor.
    Guide students through problems step-by-step.
    Don't just give answers - help them discover solutions."""
)

# Step 5: Set up session and runner
APP_NAME = "math_tutor_app"
USER_ID = "student_1"
SESSION_ID = "session_001"

session_service = InMemorySessionService()
runner = Runner(
    agent=agent,
    app_name=APP_NAME,
    session_service=session_service
)

# Step 6: Define async function to run the agent
async def run_agent():
    # Create session
    session = await session_service.create_session(
        app_name=APP_NAME,
        user_id=USER_ID,
        session_id=SESSION_ID
    )
    print(f"Session created: {SESSION_ID}\n")

    # Prepare user message
    user_message = Content(
        role="user",
        parts=[Part(text="How do I solve 2x + 5 = 13?")]
    )

    # Run agent and collect response
    print("User: How do I solve 2x + 5 = 13?\n")
    print("Agent: ", end="")

    async for event in runner.run_async(
        user_id=USER_ID,
        session_id=SESSION_ID,
        new_message=user_message
    ):
        # Print final response
        if event.is_final_response() and event.content and event.content.parts:
            print(event.content.parts[0].text)

# Step 7: Run the agent
# For Jupyter/Colab: Use await directly
# await run_agent()

# For Python scripts: Use asyncio.run()
asyncio.run(run_agent())
```

Running this example

In Jupyter notebook or Google Colab:

```shell
# Just use await (event loop is already running)
await run_agent()
```

In a Python script:

```python
# Use asyncio.run() to start the event loop
asyncio.run(run_agent())
```

Expected output:

```
Session created: session_001

User: How do I solve 2x + 5 = 13?

Agent: Great question! Let's work through this together. First, what do you
think we need to do to get x by itself on one side?
```

When to use programmatic execution

Good for:

- ✓ Jupyter notebooks and Google Colab
- ✓ Custom Python applications
- ✓ Data processing pipelines
- ✓ Research and experimentation
- ✓ Fine-grained control over execution

Not ideal for:

- ✗ Quick interactive testing (use `adk web` or `adk run`)
- ✗ Non-Python environments

Comparison: When to use each method

Method	Best for	Session persistence	Setup complexity
`adk web`	Development, debugging, demos	✓ Yes (in browser)	Low
`adk run`	Quick tests, CLI workflows	✗ No	Very low
`adk api_server`	API integration, production	Depends on client	Low
Programmatic	Custom apps, notebooks	✓ Yes (managed)	Medium

Summary:

- **Developing?** Use `adk web`
- **Quick test?** Use `adk run`
- **Building an API?** Use `adk api_server`
- **Custom integration?** Use programmatic execution

Key takeaways

Choose the right tool:

- Use `adk web` for visual development and debugging
- Use `adk run` for quick command-line testing
- Use `adk api_server` for deploying as an API
- Use programmatic execution for custom Python applications

All methods work with the same agent:

- Your `agent.py` file doesn't change
- Different methods are just different ways to interact
- Choose based on your current workflow needs