Testing - Agent Development Kit

Source URL: https://google.github.io/adk-docs/get-started/testing/

Testing your Agents¶

Before you deploy your agent, you should test it to ensure that it is working as intended. The easiest way to test your agent in your development environment is to use the ADK web UI with the following commands.

PythonJava

```
adk api_server
```

Make sure to update the port number.

```
mvn compile exec:java \
     -Dexec.args="--adk.agents.source-dir=src/main/java/agents --serve
```

In Java, both the Dev UI and the API server are bundled together.

This command will launch a local web server, where you can run cURL commands or send API requests to test your agent.

Local testing¶

Local testing involves launching a local web server, creating a session, and sending queries to your agent. First, ensure you are in the correct working directory:

```
parent_folder/
└── my_sample_agent/
```

```
    └── agent.py (or Agent.java)
```

Launch the Local Server

Next, launch the local server using the commands listed above.

The output should appear similar to:

PythonJava

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

```
2025-05-13T23:32:08.972-06:00  INFO 37864 --- [ebServer.main()] o.s.b
2025-05-13T23:32:08.980-06:00  INFO 37864 --- [ebServer.main()] com.go
2025-05-13T23:32:08.981-06:00  INFO 37864 --- [ebServer.main()] com.go
```

Your server is now running locally. Ensure you use the correct ***port number*** in all the subsequent commands.

Create a new session

With the API server still running, open a new terminal window or tab and create a new session with the agent using:

```
curl -X POST http://localhost:8000/apps/my_sample_agent/users/u_123/se
  -H "Content-Type: application/json" \
  -d '{"state": {"key1": "value1", "key2": 42}}'
```

Let's break down what's happening:

- `http://localhost:8000/apps/my_sample_agent/users/u_123/sessions/s_123`: This creates a new session for your agent `my_sample_agent`, which is the name of the agent folder, for a user ID (`u_123`) and for a session ID (`s_123`). You can replace `my_sample_agent` with the name of your agent folder. You can replace `u_123` with a specific user ID, and `s_123` with a specific session ID.
- `{"state": {"key1": "value1", "key2": 42}}`: This is optional. You can use this to customize the agent's pre-existing state (dict) when creating the session.

This should return the session information if it was created successfully. The output should appear similar to:

```
{"id":"s_123","appName":"my_sample_agent","userId":"u_123","state":{"
```

Info

You cannot create multiple sessions with exactly the same user ID and session ID. If you try to, you may see a response, like: `{"detail":"Session already exists: s_123"}`. To fix this, you can either delete that session (e.g., `s_123`), or choose a different session ID.

Send a query

There are two ways to send queries via POST to your agent, via the `/run` or `/run_sse` routes.

- `POST http://localhost:8000/run`: collects all events as a list and returns the list all at once. Suitable for most users (if you are unsure, we recommend using this one).
- `POST http://localhost:8000/run_sse`: returns as Server-Sent-Events, which is a stream of event objects. Suitable for those who want to be notified as soon as the event is available. With `/run_sse`, you can also set `streaming` to `true` to enable token-level streaming.

Using `/run`

```
curl -X POST http://localhost:8000/run \
-H "Content-Type: application/json" \
-d '{
"appName": "my_sample_agent",
"userId": "u_123",
"sessionId": "s_123",
"newMessage": {
    "role": "user",
    "parts": [{
    "text": "Hey whats the weather in new york today"
    }]
}
}'
```

If using `/run`, you will see the full output of events at the same time, as a list, which should appear similar to:

```
[{"content":{"parts":[{"functionCall":{"id":"af-e75e946d-c02a-4aad-93
```

Using `/run_sse`

```
curl -X POST http://localhost:8000/run_sse \
-H "Content-Type: application/json" \
-d '{
"appName": "my_sample_agent",
"userId": "u_123",
"sessionId": "s_123",
"newMessage": {
    "role": "user",
    "parts": [{
    "text": "Hey whats the weather in new york today"
    }]
},
"streaming": false
```

```
}'
```

You can set `streaming` to `true` to enable token-level streaming, which means the response will be returned to you in multiple chunks and the output should appear similar to:

```
data: {"content":{"parts":[{"functionCall":{"id":"af-f83f8af9-f732-46

data: {"content":{"parts":[{"functionResponse":{"id":"af-f83f8af9-f732

data: {"content":{"parts":[{"text":"OK. The weather in New York is sun
```

Info

If you are using `/run_sse`, you should see each event as soon as it becomes available.

Integrations¶

ADK uses Callbacks to integrate with third-party observability tools. These integrations capture detailed traces of agent calls and interactions, which are crucial for understanding behavior, debugging issues, and evaluating performance.

- Comet Opik is an open-source LLM observability and evaluation platform that natively supports ADK.

Deploying your agent¶

Now that you've verified the local operation of your agent, you're ready to move on to deploying your agent! Here are some ways you can deploy your agent:

- Deploy to Agent Engine, the easiest way to deploy your ADK agents to a managed service in Vertex AI on Google Cloud.
- Deploy to Cloud Run and have full control over how you scale and manage your agents using serverless architecture on Google Cloud.