Multi-agent Application with Agent Development Kit

Build a kitchen renovation multi-agent system using Vertex AI ADK

Written by Abirami Sukumaran
Find more Google Cloud community content on Medium

What is a multi-agent system?

When your application has multiple agents working together autonomously and together as required to cater to its larger purpose with each of its agents being independently knowledgeable and responsible for a specific focus area, then your application becomes a multi-agent system. Agents are autonomous programs that can make decisions generatively and based on instructions and information made available to them and are responsible for the specific area of focus for which they are created.

The Agent Development Kit (ADK)

Agent Development Kit (ADK) is a flexible and modular framework for **developing and deploying AI agents**. ADK supports building sophisticated applications by composing multiple, distinct agent instances into a **Multi-Agent System (MAS)**.

In ADK, a multi-agent system is an application where different agents, often forming a hierarchy, collaborate or coordinate to achieve a larger goal. Structuring your application this way offers significant advantages, including enhanced modularity, specialization, reusability, maintainability, and the ability to define structured control flows using dedicated workflow agents.

Things to keep in mind for a multi-agent system!

1. First, It's important to have a **proper understanding and reasoning** of the specialization for each agent. Do you know why you need a specific sub-agent for something? Work that out first.
2. Second, how to **bring them together** with a root agent to route and make sense of each of the responses.
3. Third, there are multiple types of agent routing that you can find here in this documentation. Find out which one suits your application's flow. Also, determine the various contexts and states that you need for your multi-agent system's flow control.

What are we building today?

Let's build a **multi-agent system to handle kitchen renovations**. That's what we'll do. We'll build a system with three agents.

1. Renovation proposal agent
2. Permits and compliance check agent
3. Order status check agent

A **renovation proposal agent** to generate the kitchen renovation proposal document. A **permits and compliance agent** to take care of permits and compliance. And an **order status check agent** to check order status. We have a root agent that orchestrates these agents based on the requirement.

Let's dive right in!

Let's get started with getting the multi-agent system implemented.

Setup your Google Cloud project

1. Sign in to the Google Cloud console.
2. Make sure you have an active project with an active billing account. Create a new project if you don't already have one.
3. Also if you are reading this and would like to get hold of some credits to help you get started with Google Cloud and to use ADK, use this link to redeem credits. You can follow the instructions here to redeem it. Please note that this link is valid only till the end of May for redemption.
4. Activate Cloud Shell by clicking this link. You can toggle between the Cloud Shell terminal (for running cloud commands) and Editor (for building projects) by clicking on the corresponding button from Cloud Shell.
5. Make sure to have Python 3.9+

Set up the environment and install ADK

1. Create and activate the virtual environment (recommended)
2. From your Cloud Shell terminal, create a virtual environment:

```
python -m venv .venv
```

3. Activate the virtual environment:

```
source .venv/bin/activate
```

4. Install ADK:

```
pip install google-adk
```

5. From the Cloud Shell terminal, create a directory in your desired project location:

```
mkdir renovation-agent
```

Create the multi-agent system

Go to Cloud Shell Editor and create the following project structure by creating the files (empty to begin with):

```
renovation-agent/
        __init__.py
        agent.py
       .env
```

2. Go to __init__.py and update with the following content:

```python
from . import agent
```

3. Go to agent.py and update the file with content from the following location:

https://github.com/AbiramiSukumaran/adk-renovation-agent/blob/main/agent.py

Python
```python
...
'''
# Root Agent Definition
'''
root_agent = Agent(
  model=MODEL_NAME,
  name=ROOT_AGENT_NAME,
  description=("Agent that manages and executes the building renovation proposal for a home owner."),

# Instructions for intent detection: Combine guardrails string
# and the sub-agent routing instruction

  instruction=(
  f""" {root_agent_system_instruction}

************************************************************************************************************
********************************************************************************************
******
  - If the user asks you to create a proposal document
    invoke the proposal_agent
  - If the user wants to create a checklist of permits or
  compliance documentation, invoke the permits_agent.
  - If the user wants to execute ordering of materials,
  place order or get the status of delivery,
  invoke the ordering_agent without asking more questions.

********************************************************************************************
******

********************************************************************************************
******
 """),

  generate_content_config=types.GenerateContentConfig(temperature=0.2),

  # Orchestrating sub agents
  sub_agents=[
      proposal_agent,
      permits_agent,
      ordering_agent
  ],
)
...
```

Comment out the sub agent “ordering_agent” if you do not want to create a Cloud Run Function for getting order status from AlloyDB.

If you want to try the ordering agent, steps are as follows

Create AlloyDB cluster and instance, table and data

1. To create a cluster and instance, follow the step mentioned in step 7.

Google Cloud

Database setup: https://codelabs.developers.google.com/multi-agent-app-with-adk#6

2. To create a table and insert data, run the SQL statements from here: https://github.com/AbiramiSukumaran/adk-renovation-agent/blob/main/database_script.sql

Create a Cloud Run Function in Java to extract order status information

1. Create a Cloud Run function from here: https://console.cloud.google.com/run/create?deploymentType=function
2. Set the name of the function to “check-order-status” and choose the “Java 17” as runtime.
3. You can set authentication to “Allow unauthenticated invocations” since it is a demo application.
4. Follow the steps in the link below to make sure your Cloud Run function can talk to your AlloyDB data and retrieve.
5. Go to the “Create the Cloud Run function” section in the same step, step 7 of the codelab under the section “Create a Cloud Run Function in Java to extract order status information” https://codelabs.developers.google.com/smart-stylist-app#9 link.
6. Follow all instructions in that section.
7. Click **Create**.

Once the function is created and placeholder code is loaded

1. Change the name of the Java file to “ProposalOrdersTool.java” and the class name to “ProposalOrdersTool”.

2. Replace the placeholder code in ProposalOrdersTool.java & pom.xml with code from the respective files in the “Cloud Run function” folder in this repository.

3. In line 73 of ProposalOrdersTool.java that contains the following code, replace the placeholder values with values from your configuration:

```
String ALLOYDB_INSTANCE_NAME =
"projects/<<YOUR_PROJECT_ID>>/locations/us-central1/clusters/<<YOUR_CLUSTER>>/instances/<<YOUR_INSTANCE>>";
```

4. Deploy the function and test it.

Set up .env variables

Set up your values for the parameters in the template .env file in this repository.

Make sure you have the Cloud Storage Bucket

This is to store the proposal document that the agent generates. Create it and provide access so the multi-agent system created with Vertex AI can access it. Here is how you can do it:

https://cloud.google.com/storage/docs/creating-buckets#console

Execute

You can run the following to test it in the terminal:

```
adk run .
```

You can run the following to test it in an ADK provisioned UI:

```
adk web
```

Demo

Google Cloud

https://youtu.be/bYeOsgGhwig

Multi-agent system source explanation

The agent.py file defines the structure and behavior of our kitchen renovation multi-agent system using the Agent Development Kit (ADK). Let's break down the key components:

Agent definitions

RenovationProposalAgent

This agent is responsible for creating the kitchen renovation proposal document. It optionally takes input parameters like kitchen size, desired style, budget, and customer preferences. Based on this information, it uses a large language model (LLM) Gemini 2.5 to generate a detailed proposal. The generated proposal is then stored in a Google Cloud Storage bucket.

PermitsAndComplianceCheckAgent

This agent focuses on ensuring the renovation project adheres to local building codes and regulations. It receives information about the proposed renovation (e.g., structural changes, electrical work, plumbing modifications) and uses the LLM to check permit requirements and compliance rules. The agent uses information from a knowledge base (which you can customize to access external APIs to gather relevant regulations).

OrderingAgent (optional)

This agent (you can comment it out if you don't want to implement it now) handles checking the order status of materials and equipment needed for the renovation. To enable it, you'll need to create a Cloud Run function as described in the setup steps. The agent will then call this Cloud Run function, which interacts with an AlloyDB database containing order information. This demonstrates integration with a database system to track real-time data.

Root Agent (Orchestrator)

The root_agent acts as the central orchestrator of the multi-agent system. It receives the initial renovation request and determines which sub-agents to invoke based on the request's needs. For example, if the request requires checking permit requirements, it will call the PermitsAndComplianceCheckAgent. If the user wants to check order status, it will call the OrderingAgent (if enabled).

The root_agent then collects the responses from the sub-agents and combines them to provide a comprehensive response to the user. This could involve summarizing the proposal, listing required permits, and providing order status updates.

Data flow

The user initiates a request through the ADK interface (either the terminal or the web UI).

1. The request is received by the root_agent.
2. The root_agent analyzes the request and routes it to the appropriate sub-agents.
3. The sub-agents use LLMs, knowledge bases, APIs, and databases to process the request and generate responses.
4. The sub-agents return their responses to the root_agent.
5. The root_agent combines the responses and provides a final output to the user.

Other key concepts

Large language models (LLMs)

The agents rely heavily on LLMs to generate text, answer questions, and perform reasoning tasks. The LLMs are the "brains" behind the agents' ability to understand and respond to user requests. We are using Gemini 2.5 in this application.

Google Cloud Storage

Google Cloud Storage is used to store the generated renovation proposal documents. You need to create a bucket and grant the necessary permissions for the agents to access it.

Cloud Run (optional)

The OrderingAgent uses a Cloud Run function to interface with AlloyDB. Cloud Run provides a serverless environment to execute code in response to HTTP requests.

AlloyDB (optional)

If you're using the OrderingAgent, you'll need to set up an AlloyDB database to store order information.

.env file

The .env file stores sensitive information like API keys, database credentials, and bucket names. It's crucial to keep this file secure and not commit it to your repository. It also stores configuration settings for the agents and your Google Cloud project. The root_agent or supporting functions will typically read values from this file. Make sure all required variables are properly set in the .env file. This includes the Cloud Storage bucket name.

Conclusion

The multi-agent system is designed to streamline the kitchen renovation process by automating tasks such as proposal generation, permit checking, and order status tracking. Each agent has a specific role, and the root_agent coordinates their activities to provide a comprehensive solution. The system leverages LLMs, Google Cloud services, and potentially external APIs to deliver its functionality. Here is a link to the product documentation. For step-by-step instructions of this agentic application, please refer to the codelab.

Let me know what you are building!