Agent Assist Automation: Approach, Implementation & Notes

Objective

This document outlines the approach taken to automate question-answering using Google Cloud's **Agent Assist** capability within Dialogflow. The system processes a list of domain-specific questions and retrieves generative answers along with supporting documentation. The automation is designed to simulate a user interacting with the Agent Assist interface and to collect high-quality responses for analysis, validation, and training augmentation.

The primary goal was to create a scalable, reproducible script that:

- Reads a list of business-relevant questions from cloud storage
- Simulates user interaction with Agent Assist
- Extracts AI-generated responses and sources
- Saves responses in multiple structured formats (CSV, Excel, JSON)
- Handles various response conditions gracefully (no answer, partial answer, full response)

Cloud Resources & Why They Were Chosen

☁️ Google Cloud Project

Project ID: prj-sandbox-ccaas-lab-0

We chose this project because it is preconfigured with:

- Access to **Agent Assist**
- Required APIs enabled
- Linked **Conversation Profiles** and **Knowledge Bases**
- Secure service account credentials (via Vertex AI notebook or Colab auth)

☁️ Cloud Storage (GCS)

Bucket: gs://agent_assist_belair_on/

Why:

- Centralized location for storing and updating the question list
- Makes it easy to manage JSON files as input
- Used with gcsfs for programmatic access within the script

Dialogflow & Agent Assist APIs

Why Dialogflow v2beta1?

We used the **dialogflow_v2beta1** API client because it is the only version that supports **Generative Agent Assist** features like:

- analyze_content(): simulate real-time user queries
- suggestKnowledgeAssistResponse: extract answers & documents

- generativeSource.snippets: get citations for the answer
- Full control over the conversation and participant lifecycle

Key APIs Used

ConversationsClient.create_conversation()

Creates a new Dialogflow conversation, simulating the environment an agent would operate in.

Why it's needed:

- Helps scope each batch of questions
- Keeps conversation context minimal and clean

ParticipantsClient.create_participant()

Registers a participant (end user) in the conversation.

Why:

- Needed to simulate real agent queries
- Every query must be linked to a participant

ParticipantsClient.analyze_content()

Sends a user query into the conversation.

Why:

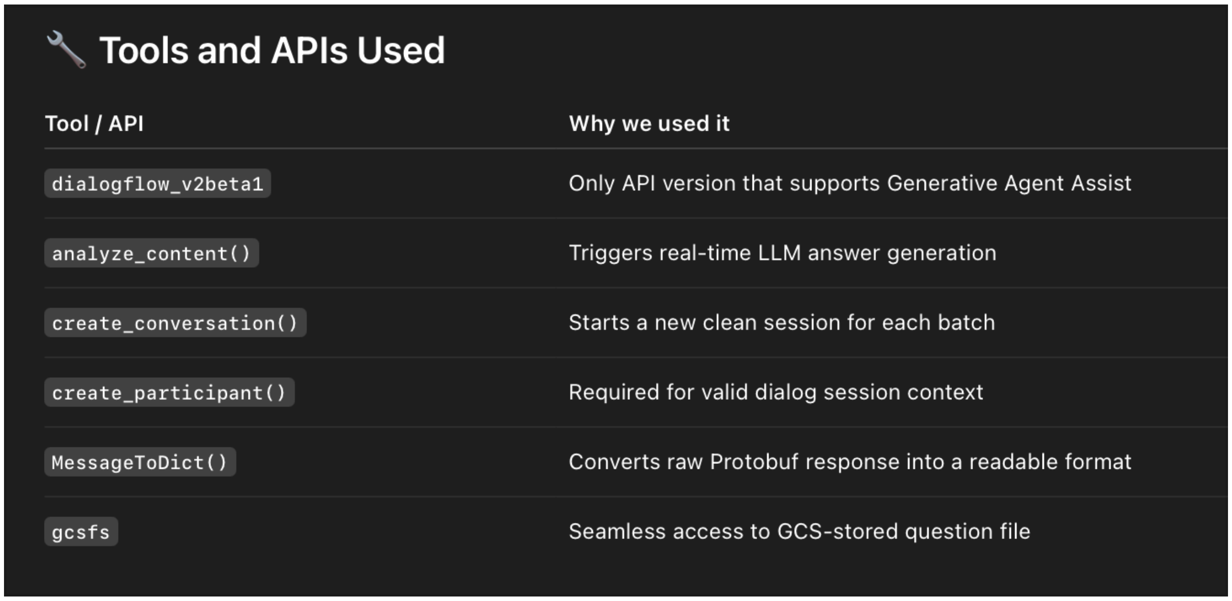
- Triggers Agent Assist's real-time generative system
- Automatically returns the generative answer and source snippets if available

MessageToDict(response._pb)

Parses the Protobuf AnalyzeContentResponse into a readable dictionary format.

Why:

- Required to access deeply nested fields like:
    - suggestedQueryAnswer.answerText
    - generativeSource.snippets[].uri/title



🔧 Tools and APIs Used

Tool / API	Why we used it
`dialogflow_v2beta1`	Only API version that supports Generative Agent Assist
`analyze_content()`	Triggers real-time LLM answer generation
`create_conversation()`	Starts a new clean session for each batch
`create_participant()`	Required for valid dialog session context
`MessageToDict()`	Converts raw Protobuf response into a readable format
`gcsfs`	Seamless access to GCS-stored question file

Implementation Logic

1. **Load Questions from GCS**
   Read non_ambiguous_questions.json from a GCS bucket using gcsfs.
2. **Split into Batches of 10**
   Each conversation is limited to 10 questions for better isolation and relevance of context.
3. **Conversation Lifecycle per Batch**
   - A new conversation is created per batch
   - A new participant is registered
   - Each question is sent with a 10-second delay between them
4. **Analyze and Parse Responses**
   - The script waits for Agent Assist to respond
   - If answerText is returned → marked as "Answer + Sources"
   - If only snippets are returned → marked as "Sources Only"
   - If neither is available → marked as "No Response"
5. **Store in Multiple Formats**
   - **CSV**: for tabular reports
   - **Excel**: stakeholder-friendly review
   - **JSON**: machine-readable for LLM or downstream pipelines

Why Batch Questions?

- Prevents long conversations that may degrade relevance
- Avoids token/context overflow
- It also avoids overloading a single session with too many messages, which can lead to degraded LLM performance.

Why 10-Second Sleep?

- Ensures the generative model has time to complete its response
- Prevents overloading the backend with rapid-fire requests

Why Extract Both Answers & Sources?

- We aim to evaluate both the **generated content** and the **supporting documentation**
- Prevents backend throttling
- Helps validate whether answers are grounded in known knowledge base entries