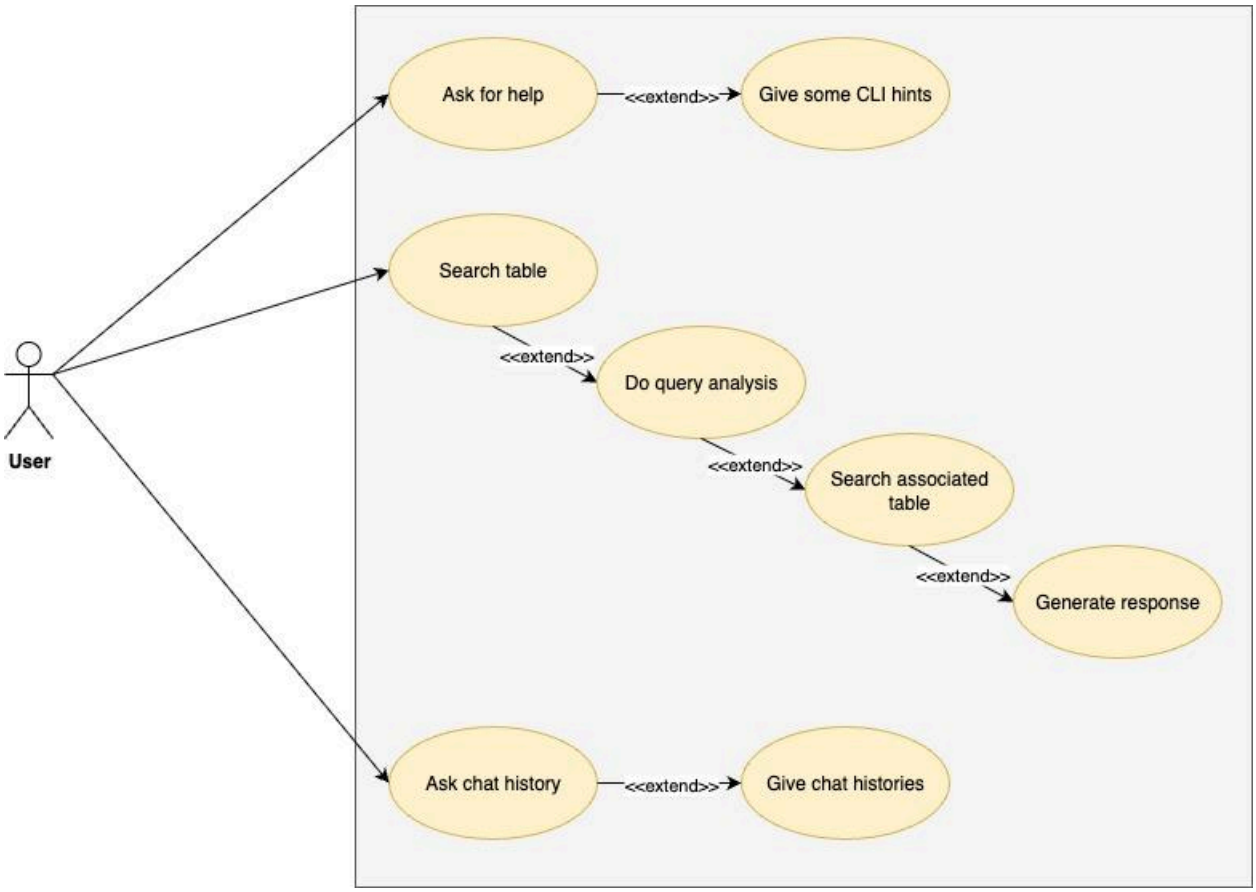
# Table Search Agent - MLOps On GCP

## Goals

The gernaral objectives of this poject are:

1. Create an interactive chatbot to search the required table after given a context words
2. The entire chat system made by some agents
3. Those agents deployed on Google Cloud Platform (GCP)
4. The overall agents' capabilities include
    a. Give some answers for help questions
    b. Return the chat history if asked
    c. Search, and rank the associated table for the given chat

## Use-Case



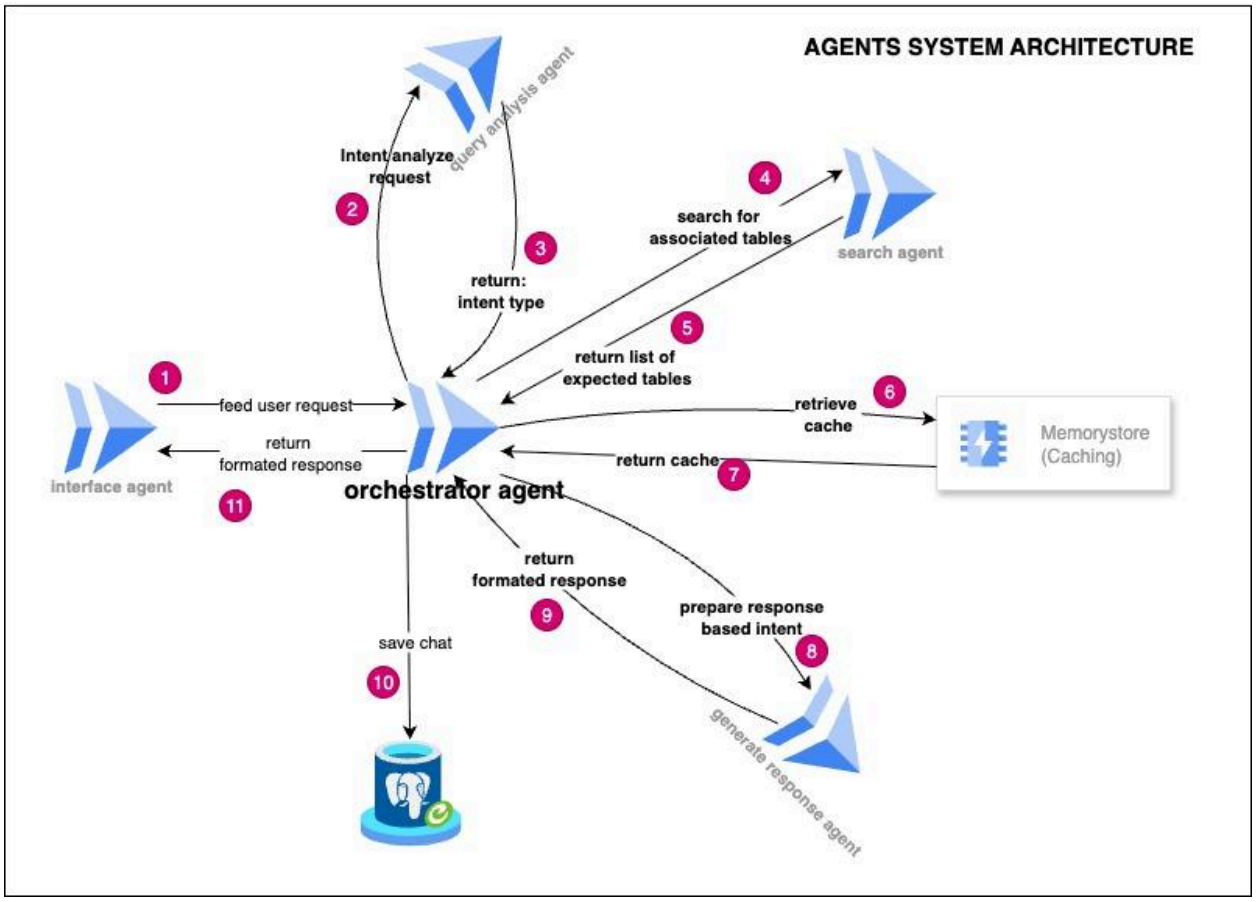
As we mentioned before, the main capabilities of our agents are response the user help request by giving list of required CLI, response user request for BigQuery tables searching, and give chat history list if requested.

Since the system merely depend on classic and simplest NLP algorithm (TF-IDF), we are set some question boundaries, described by the following table

| Chat context | Format style | Example |
|---|---|---|
| Table search | **Flexible chat** | Help me find the marketing tables<br><br>Can you search for sales tables<br><br>Mention user related tables<br><br>Where products are stored |
| Ask for help | **FIXED chat** | Help |
| Ask for chat history | **FIXED chat** | history |

# System Architecture

Since we are only utiling classic NLP algorithms like TF-IDF and word matching, so there is no AI model included in this architecture. For further considerations, like enhancing the system capability to be more smart, we require to integrate with AI model like ChatGPT or Claude AI.




**Interface Agent:**

*Tools* : GCP Cloud run
*Capacity* : 256 MB memmory
*Languange* : Golang
*CI/CD* : Github action, Gitlab CI/CD, GCP Cloud Run
Docker Image : Stored on GCP Artifact registry
*Service Type* : HTTP
*Job Scopes* : As a frontliner for incoming request, directly integrated with the front end
*Further improvement* : - Add capability as rate limiter
- As a bridge for all data engineering HTTP services

**Orchestrator Agent:**

*Tools* : GCP Cloud run

*Capacity* : 256 MB memmory
*Languange* : Golang
*CI/CD* : Github action or Gitlab CI/CD or GCP Cloud Run
Docker Image : Stored on GCP Artifact registry
*Service Type* : HTTP
*Job Scopes* : As agent orchestrator, handle database (SQL and NoSQL) orchestration as well
*Further improvement* :Consume event via Pub/Sub for asynchronous working capability

**Query Analysis Agent:**

*Tools* : GCP Cloud run
*Capacity* : 256 MB memmory
*Languange* : Python
*CI/CD* : Github action or Gitlab CI/CD or GCP Cloud Run
Docker Image : Stored on GCP Artifact registry
*Service Type* : HTTP
*Job Scopes* : Analysis intent, refers to intent enumeration
*Further improvement* : Integrate with AI Models like ChatGP, Claude AI

**Search Agent:**

*Tools* : GCP Cloud run
*Capacity* : 256 MB memmory
*Languange* : Python
*CI/CD* : Github action or Gitlab CI/CD or GCP Cloud Run
Docker Image : Stored on GCP Artifact registry
*Service Type* : HTTP
*Job Scopes* : Search associated tables, rank them, based on TF-IDF and keyword macth
*Further improvement* : Integrate with AI Models like ChatGP, Claude AI

**Response Generation Agent:**

*Tools* : GCP Cloud run
*Capacity* : 256 MB memmory
*Languange* : Python
*CI/CD* : Github action or Gitlab CI/CD or GCP Cloud Run
Docker Image : Stored on GCP Artifact registry
*Service Type* : HTTP
*Job Scopes* : Generate response wording based on intent (intent: refers to intent enumeration**)**

**Response Caching:**

*Tools* : GCP Memory store (or replaced by Redis)
*Job Scopes* : Cache agent response based on **intent type** and **table name**.
*Expiry time* : 7 days

Job Scopes : Reduce latency, since the highly requested table's responses already available

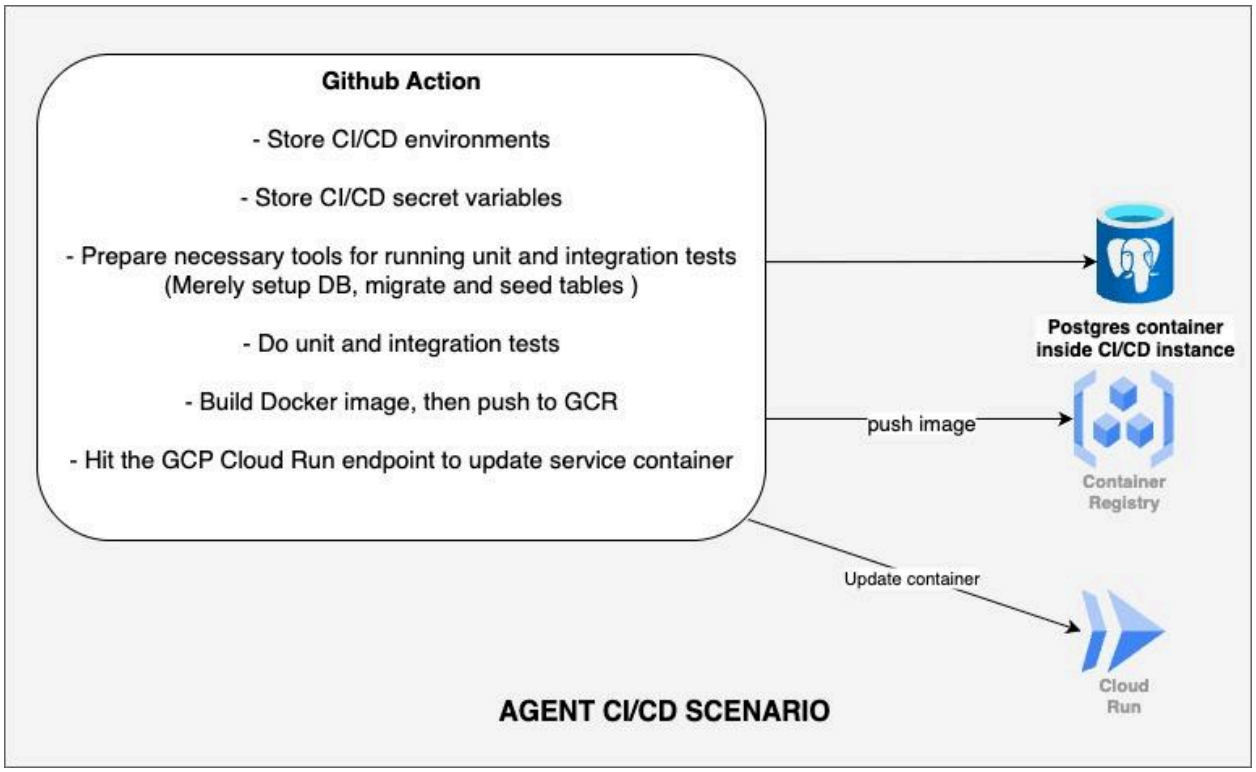
**System Database:**

*Tools* : Postgres SQL
*Job Scopes* : System database
*E/R Data Model* : Will be provided in the next section

**Other Specifications:**

IaC : Terraform

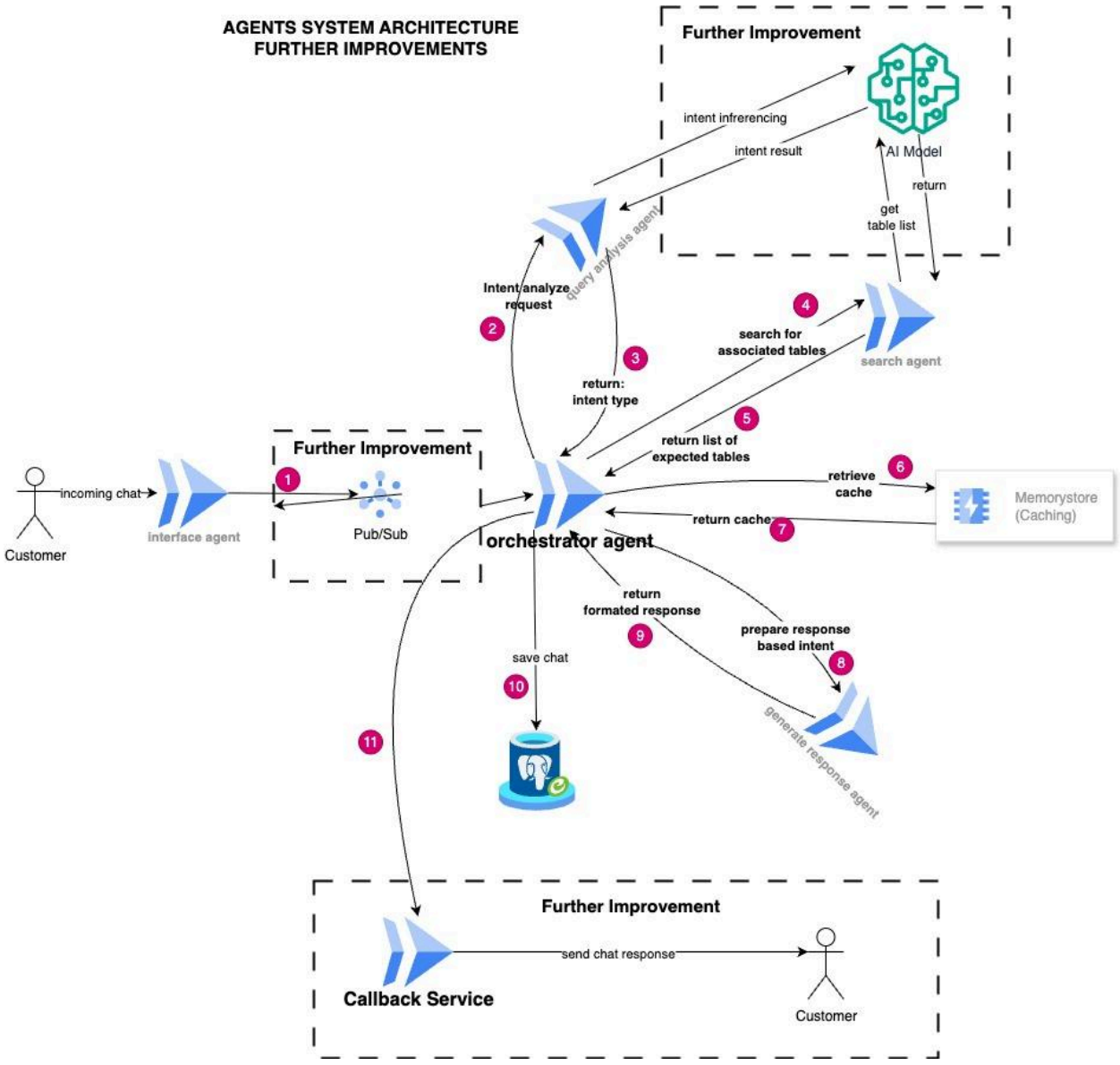
**CI/CD Pipeline** :



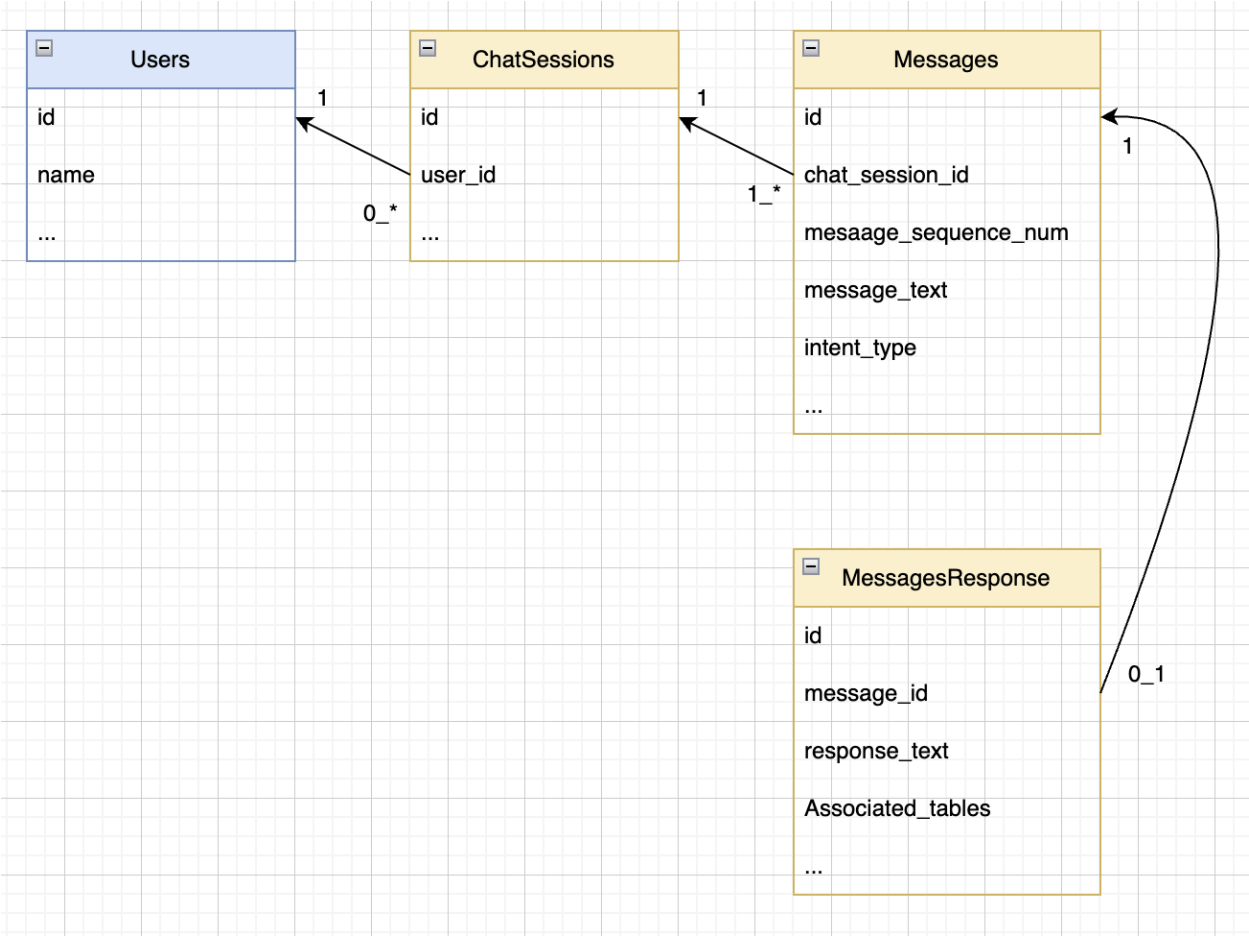

**FURTHER IMPROVEMENT**

- **Reduce Latency:** add event driven mechanisme between **inteface agent** and **orchestrator agent**, Add callback mechanisme when the process finished

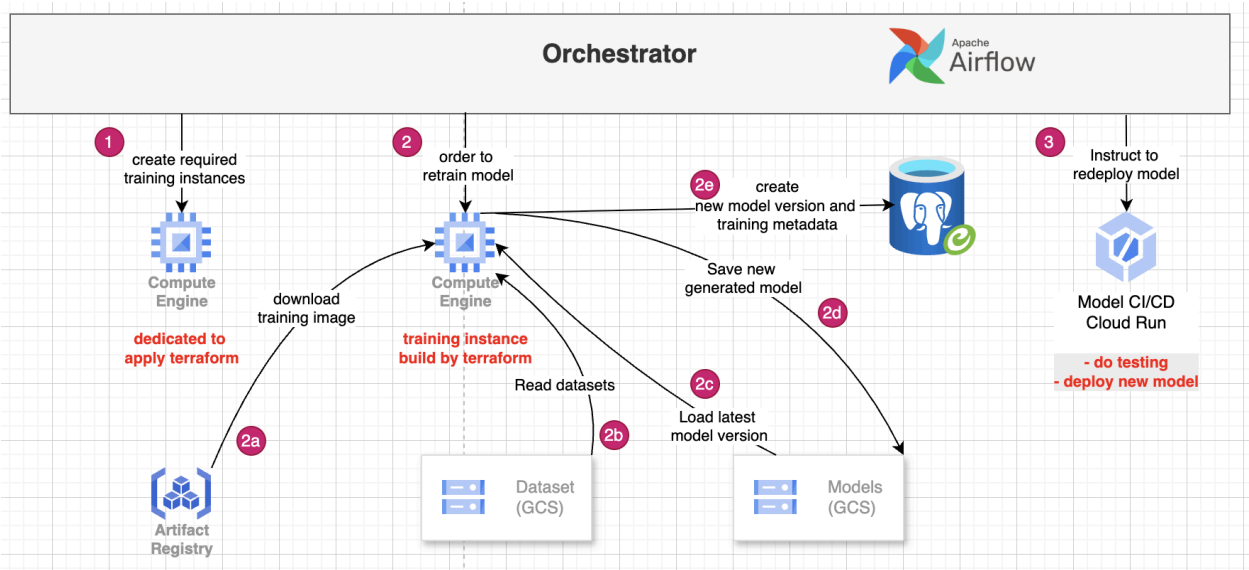
- **Increase capability**: Integrate Query Analysis Agent and Table Search Agent with AI models. It can be in house AI model, or ChatGPT, CLaude AI.


AGENTS SYSTEM ARCHITECTURE
FURTHER IMPROVEMENTS
Further Improvement
intent infrerencing
intent result
AI Model
return
get
table list
query analysis agent
Intent analyze
request
2
3
return:
intent type
4
search for
associated tables
search agent
5
return list of
expected tables
Further Improvement
incoming chat
Customer
interface agent
1
Pub/Sub
orchestrator agent
retrieve
cache
6
Memorystore
(Caching)
return cache
7
return
formated response
9
prepare response
based intent
8
save chat
10
11
generate response agent
Further Improvement
send chat response
Callback Service
Customer

# E/R Data Model



# MODEL LIFECYLE
# (Case specific for inhouse models developments)




This section demonstrate model retraining and redeployment in GCP. We put Apache airflow as orchestrator. Its main tasks including

1. Orchestrate a dedicated terraform apply instance to create required training infrastructure

2. Once training instance created, Apache Airflow will manage to instruct start training. Then, the insantance will pull the training image from GCR, and run training container. Then, it manages to load dataset from GCS (all or batch based on engineering SLA). After that, the model will be loaded for retraining. Once the retraining process was finished, the newly created model will be saved in GCS. The training instance will save new model version, performances, and metadata in model DB.

3. After that, Apache Airflow will manage to orchestrate GCP cloud run for model CI/CD.

4. Finally, Airflow managed to instruct terraform instance to destroy the training instance.

**Terraform instance:**

| | |
|---|---|
| *Tools* | : GCP Compute Enginee |
| *Capacity* | : 256 - 1GB MB memmory |
| *Standby* | : 1 (dedicated) |

**Training instance:**

*Tools* : GCP Compute Enginee
*Capacity* : depend on the model size and dataset
*Use GPU* : depend on engineering and model SLA
*Status* : temporary instance, destroyed when training pipeline was finished

**Why don’t we managed to utilize GCP vertex or AWS sagemaker instead ?**
They are absolutely expensive, by designing our training instance orchestrations, we manage to reduce huge costs, and increase flexibility