Build a Data Analyst AI Agent from Scratch

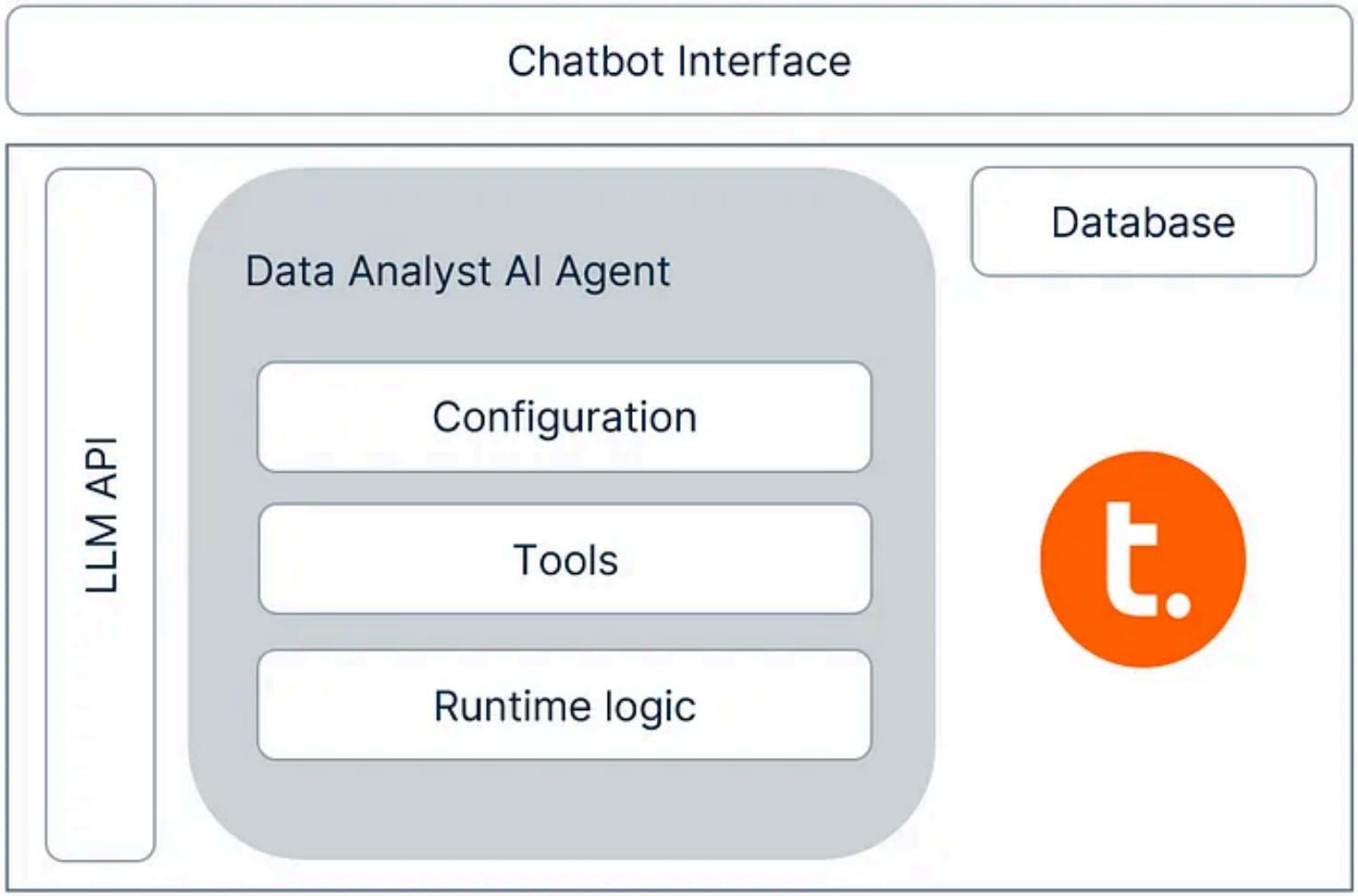
Daniel Herrera · Follow

Published in Teradata · 9 min read · Feb 7, 2025

As presented in the Open Data Science Conference AI Builders Summit 2025

Agentic AI and AI Agents are widely discussed topics within the data community. If you work as a data analyst, data scientist or data engineer, you see multiple architectures, frameworks, techniques and use cases for Agentic AI mentioned daily in your professional network.

In cases like this, where information abounds and the subject is complex, it is worth it to build an understanding of the topic starting from its most basic implementation which is the purpose of this article and the accompanying notebook.



Basic Architecture Diagram of the Data Analyst AI Agent

Agentic AI in simple terms

Agents, intelligent agents, and autonomous agents are concepts that have been discussed in academia well before commercial Large Language Models (LLMs) became widely available. An agent, in simple terms, is a system that, when provided with a goal and a set of tools, works towards achieving the goal by using the tools provided.

The use of large language models (LLMs) to determine the path to achieve a goal, sequence tool usage, and manage unexpected states is commonly referred to as Agentic AI.

Two key properties of LLMs make them particularly effective as an orchestration engine for agents, compared to rule-based approaches or other machine learning alternatives:

Embedded knowledge:

LLMs contain a vast amount of information extracted from their training data. This enables them to manage a variety of states with minimal instructions.

Structured responses:

LLMs can be instructed to generate structured outputs. This simplifies their implementation within an agent system, which, at its core, is a program processing these outputs.

Our data focused AI Agent

In our example, we build an AI Agent with the following characteristics:

Goal: Answer business-related questions provided by the user in English, based on the information contained in a database.

Tools: A function that executes SQL statements in a Teradata Vantage database.

Requirements

To build this agent, we will need the following resources:

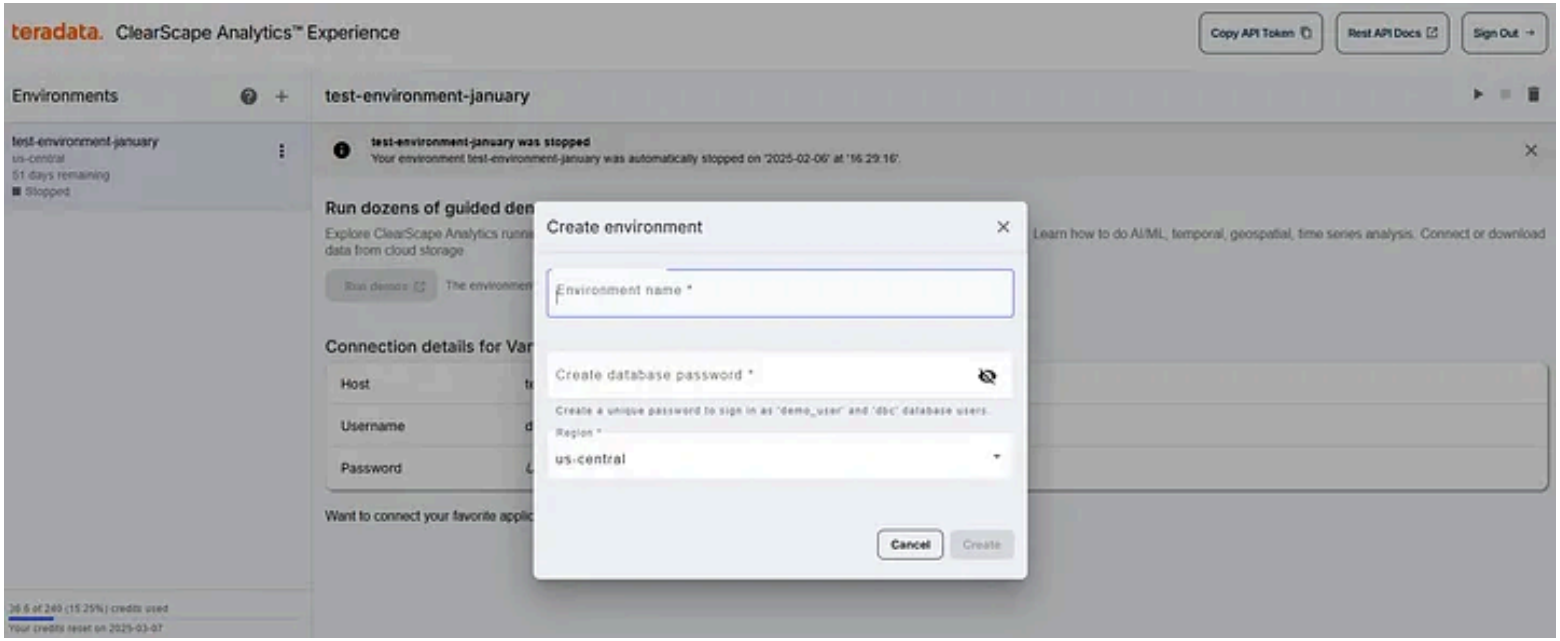
- A Teradata VantageCloud development environment with Jupyter Notebooks integration. Get one for free at ClearScape Analytics Experience.
- API access to an LLM.

In the sample notebook and this article, we use OpenAI, but this can be changed. Doing so requires modifying the API calls accordingly.

Preparing the development environment

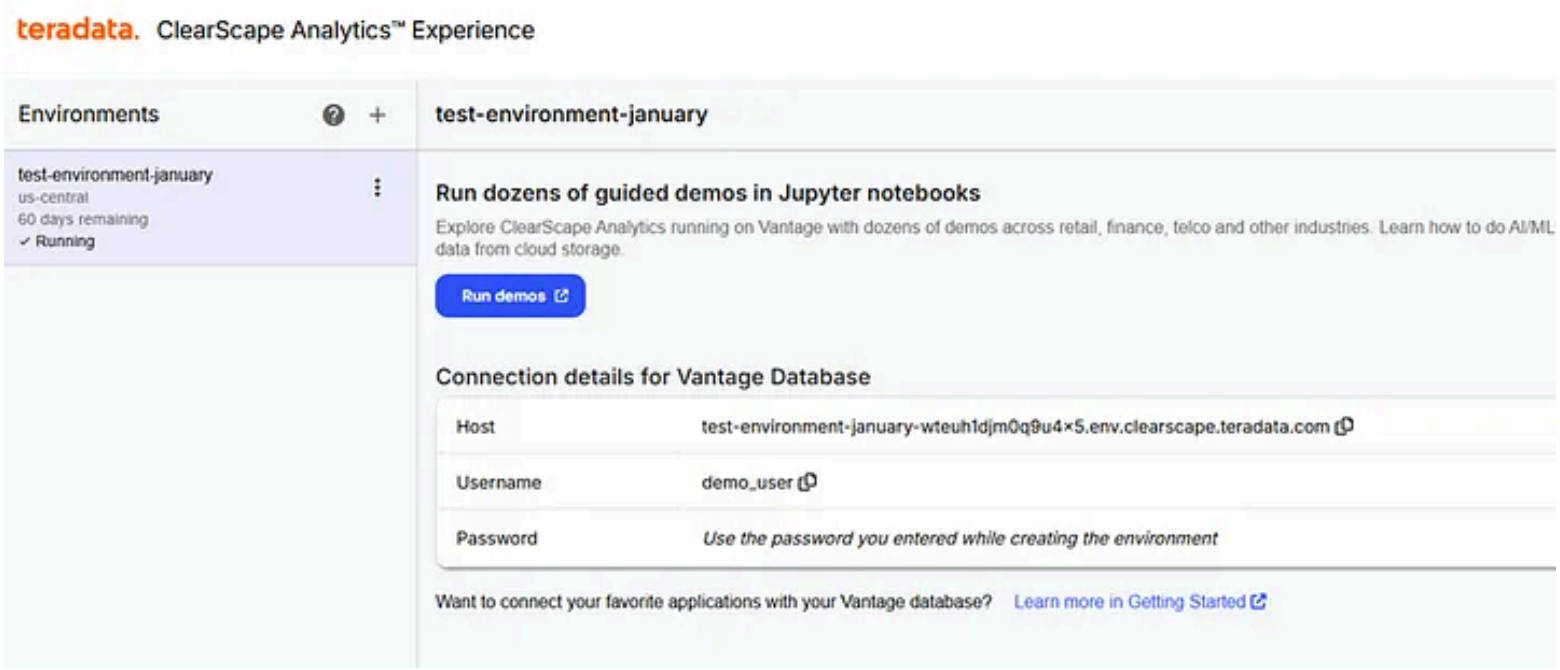
1. Log in to ClearScape Analytics Experience.
2. Create an environment in the ClearScape Analytics Experience console.

> *Take note of your chosen password as you will need it to interact with the database*



Create an environment on ClearScape Analytics Experience

3. Start the Jupyter Notebook environment by clicking “Run demos”.

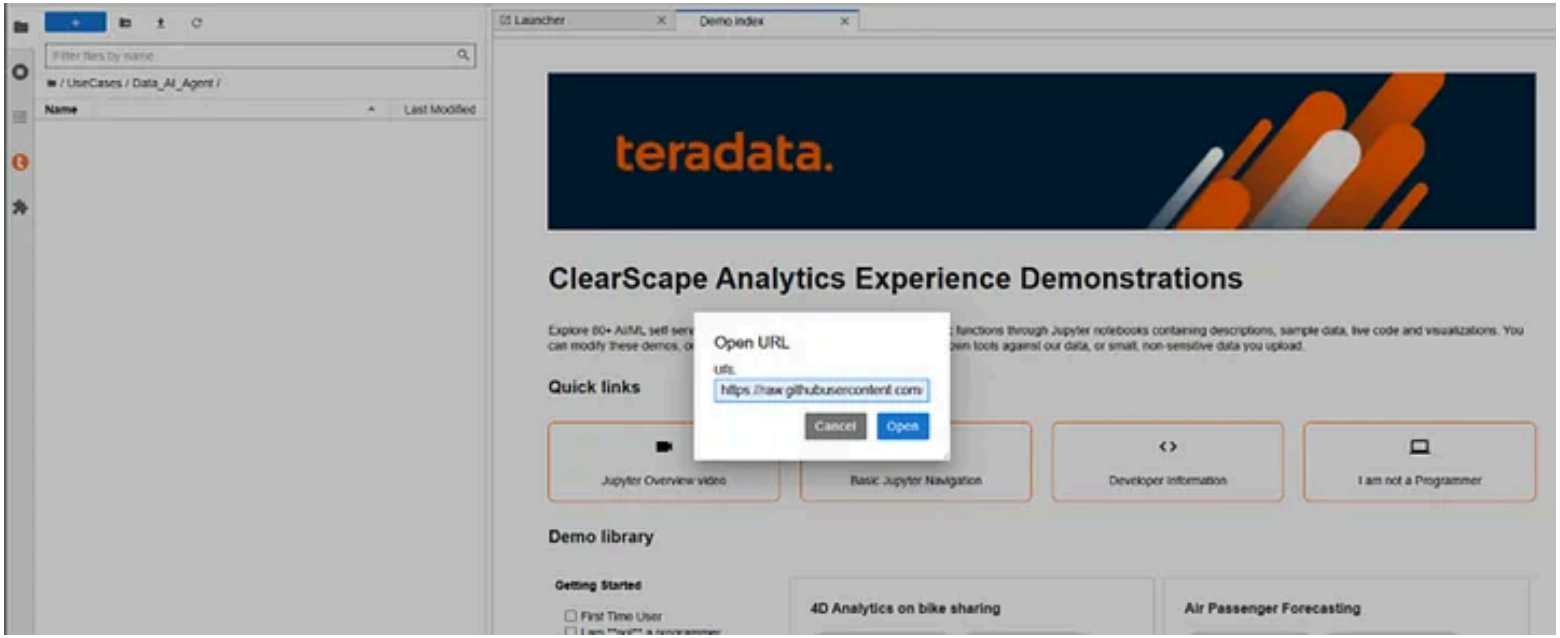


Run demos on ClearScape Analytics Experience

4. In the Jupyter Notebook environment, open the “Use Cases” folder and create a folder with a name of your choice.

5. Open the folder you’ve created, click on “Open from URL” under the “File” menu, and paste the following URL in the dialog box:

- https://raw.githubusercontent.com/Teradata/simple_data_ai_agent/refs/heads/main/simple_data_agent.ipynb



Loading sample notebook on ClearScape Analytics Experience

- This will load the project's notebook into your environment.

6. In the folder you created, create or load a `configs.json` file with the following structure, replacing `"your-api-key-here"` with your actual LLM API key:

```
{
"llm-api-key": "your-api-key-here"
}
```

7. Notes on dependencies

- If you are running the project on ClearScape Analytics Experience the above are all the required prerequisites, the notebook can run as it is.
- If you are using a different LLM provider, you will need to install the corresponding SDK and adjust the code in the notebook accordingly.

Building a Data Analyst AI Agent

Project setup:

In this section we install and import the necessary libraries, load the LLM service API key to memory, create a database connection, and load sample data.

The first step is to install the SDK of our chosen LLM API provider.

```
!pip install openai
```

Running the project in ClearScape Analytics experience simplifies this setup significantly since the database comes integrated in the notebook environment. Creating a connection to this database is straightforward as seen in the snippet below.

The password needed to establish the connection is the one selected for the corresponding environment, as mentioned in the prerequisites.

```
%run -i ../startup.ipynb
eng = create_context(host = 'host.docker.internal', username='demo_user', passwo
print(eng)
```

Once the database connection is created, the `data_loading_queries` can be run against the database to create the tables needed by the example.

Agent Configuration:

The configuration of the agent is comprised of two pieces:

1. Definition of the Agent's routine.
2. Definition of the tools available to the Agent for achieving the goal.

The Agent's routine is typically defined in a system prompt, outlining both its goal and the necessary actions to achieve it.

```
# Define the system prompt
system_prompt = f"""
You are a data analyst for a retail company working with a Teradata system.

1. Users send you business questions in plain English, and you provide answers t
2. To generate answers, you must construct an SQL statement to query the Teradat
   - The SQL query must be written as a single line of text without carriage ret
   - Ensure that the query adheres to Teradata's SQL dialect and does not includ
   - Joins across tables should be used when necessary to fulfill the user's req
```

```
    3. Execute the SQL query in Teradata.
    4. Present the query results to the user in plain English.
    """
```

The system prompt, for our purposes, should embed the data catalog needed to fulfill the user's request.

The function `query_teradata_dictionary()` is responsible for retrieving the data catalog. The DBC database in a Teradata system contains relevant information that can be easily retrieved to form a data catalog.

```
databases = ["teddy_retailers"]
def query_teradata_dictionary(databases_of_interest):
    query = f'''
            SELECT DatabaseName, TableName, ColumnName, ColumnFormat, ColumnType
            FROM DBC.ColumnsV
            WHERE DatabaseName IN ('{', '.join(databases_of_interest)}')
            '''
```

```
    table_dictionary = DataFrame.from_query(query)
    return json.dumps(table_dictionary.to_pandas().to_json())
```

The tools available to the Agent are defined as functions. Knowledge of these functions is provided to the LLM through data structures called function signatures, enabling it to generate appropriate function calls.

We need to provide the agent only with one tool, the function needed to query the database.

```
def query_teradata_database(sql_statement):
    query_statement = sql_statement.split('ORDER BY',1)[0]
    query_result = DataFrame.from_query(query_statement)
    return json.dumps(query_result.to_pandas().to_json())
```

Due to the way we need to return the results of the query to the Agent, as a JSON structure, it is very convenient to use Teradata Dataframes to process