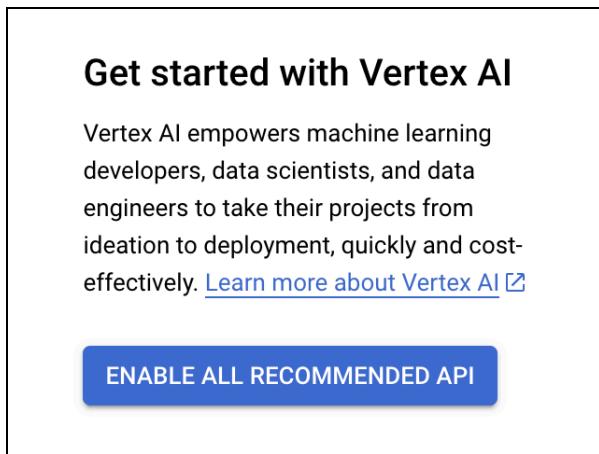


Agent Builder Advanced: BigQuery Structured Data with Cloud Function

Task 1. Set up your environment

Enable the Vertex AI API

1. In the Google Cloud Console, on the **Navigation menu**, click **Vertex AI** or Utilize the search bar to find **Vertex AI**.
2. Within the "Get Started with Vertex AI" section, locate and click on the option that says "ENABLE ALL RECOMMENDED API."



Task 2. Create a table in BigQuery

Get the Data:

- Download the test data from this repository:

<https://github.com/themlguy-tf/generative-ai.git>

Then Navigate to the file agent-builder/sales.csv

OR

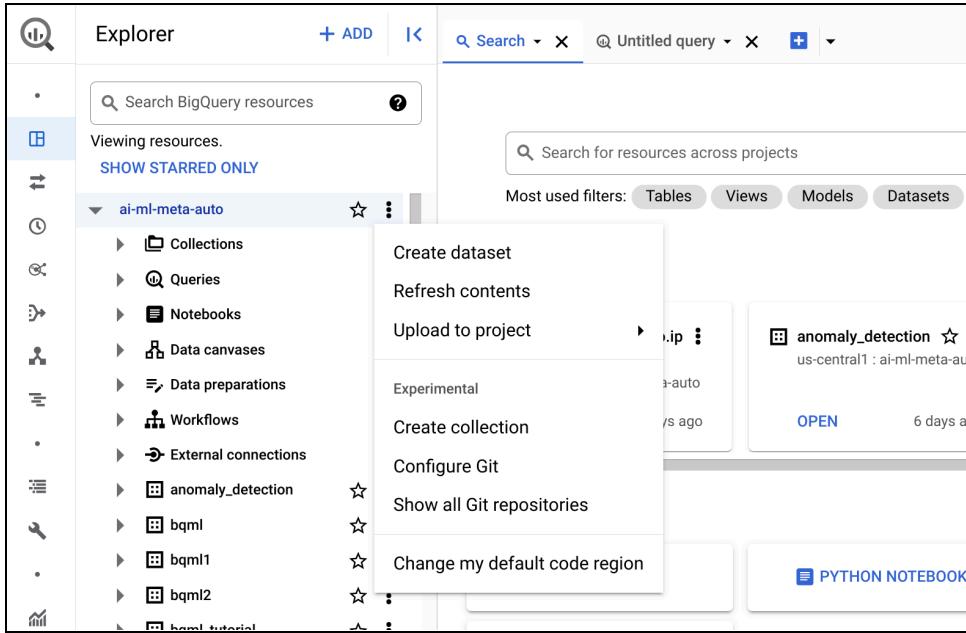
- Direct link to the file:

<https://github.com/themlguy-tf/generative-ai/blob/main/agent-builder/sales.csv>

Create a dataset named (aa_genai) in BigQuery.

Follow the steps below:

1. Navigate to the BigQuery Studio, and then create a dataset.



2. Create a dataset.
 - a. **Name:** aa_genai
 - b. **Region:** us-central1a
 - c. Other default settings remain unchanged

Create dataset

Project ID *
ai-ml-meta-auto [CHANGE](#)

Dataset ID *
aa_genai
Letters, numbers, and underscores allowed

Location type [?](#)
 Region
Specify a region to colocate your datasets with other Google Cloud services.
 Multi-region
Allow BigQuery to select a region within a group to achieve higher quota limits.

Region *
us-central1 (Iowa)

External Dataset
The selected region supports the following external dataset types: Cloud Spanner

Link to an external dataset [?](#)

Default table expiration

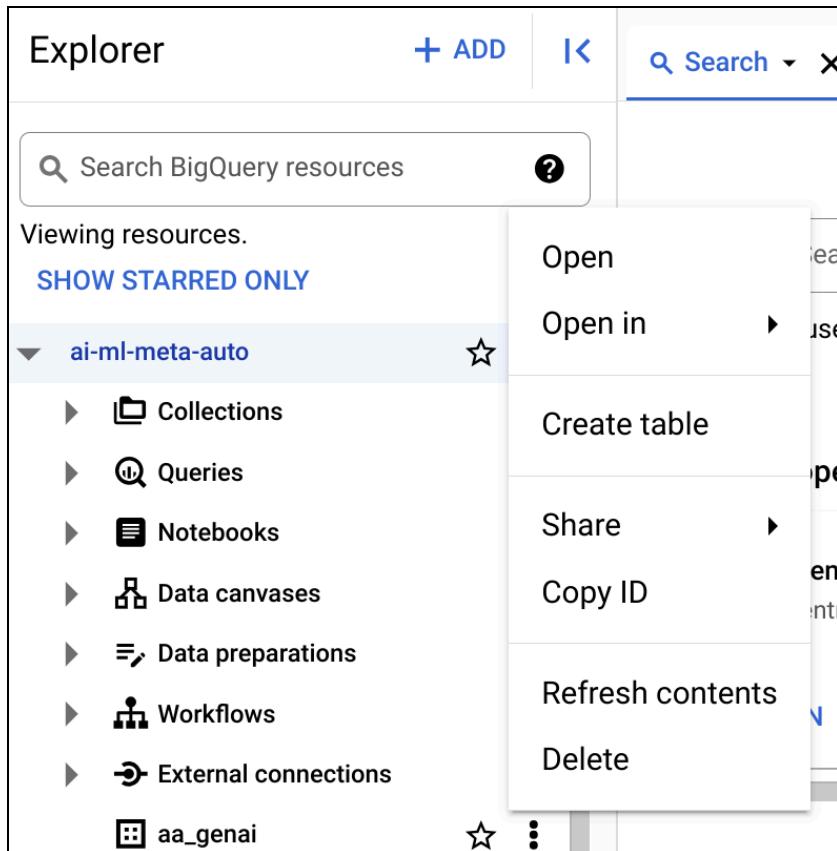
Enable table expiration [?](#)

Default maximum table age Days

Tags

[CREATE DATASET](#) [CANCEL](#)

3. Click on the three vertical dots on the right of the dataset that was created in the previous step *aa_genai*



4. Create a table with the sales.csv that was downloaded from the github repository.

Please see the settings for creating a table. Please ensure the “header rows to skip” value set to 1 in the Advanced setting, as shown below.

Create table

Create table from
Upload

Select file *
sales.csv

File format
CSV

Destination

Project *
ai-ml-meta-auto

Dataset *
aa_genai

Table *
sales

Maximum name size is 1,024 UTF-8 bytes. Unicode letters, marks, numbers, connectors

Table type
Native table

Schema

Auto detect

Schema will be automatically generated.

CREATE TABLE **CANCEL**

Create table

Advanced options

Write preference
Write if empty

Number of errors allowed
0

Unknown values ?
Column name character map
Default

Field delimiter
Comma

Quote character
Double quote

Header rows to skip
1

Quoted newlines ?
 Jagged rows ?

Encryption ?

Google-managed encryption key
Keys owned by Google

Cloud KMS key
Keys owned by customers

CREATE TABLE **CANCEL**

5. Validate the table is created

Row	invoice_and_item_number	date	store_number	store_name	address	city
1	S32080600014	2016-05-02	4867	SPEEDY GAS N SHOP	430 S 35TH ST	COUNCIL BLUFFS
2	S20260200011	2014-07-23	2524	HY-VEE FOOD STORE / DUBUQUE	3500 DODGE ST	DUBUQUE
3	S25434400087	2015-05-04	2650	HY-VEE WINE AND SPIRITS / HARLAN	1808 23RD ST	HARLAN
4	S03832200101	2012-01-31	2465	SID'S BEVERAGE SHOP	2727 DODGE ST	DUBUQUE
5	S13507200035	2013-07-23	4805	LIQUOR TOBACCO & GROCERY ...	902 1ST AVE N	FORT DODGE
6	S04013200084	2012-02-09	4180	SMOKIN' JOE'S #10 TOBACCO ...	480 7TH AVE	MARION
7	S12593200014	2013-06-04	3778	FAMILY PANTRY	4538 LOWER BEAVER RD	DES MOINES
8	S27616000013	2015-08-31	4874	Z'S QUICKBREAK	201, E WASHINGTON ST	MOUNT PLEASANT
9	S05064400029	2012-04-16	2575	HY-VEE FOOD STORE #1 / WATERLOO	2834 ANSBOROUGH AVE	WATERLOO
10	S21395300084	2014-09-25	2572	HY-VEE FOOD STORE / CEDAR FALLS	6301 UNIVERSITY	CEDAR FALLS
11	S15911800020	2013-11-23	4458	QUIK TRIP #568 / JOHNSTON	5169 MERLE HAY RD	JOHNSTON

Task 3. Create a tool to list the tables from the BigQuery

Create a tool to list the tables from the BigQuery

To create a tool, 1st go to the cloud console, and create a function named `list_tables_new`.

Note: pass the Flask endpoint in this function, to get the table names from BigQuery

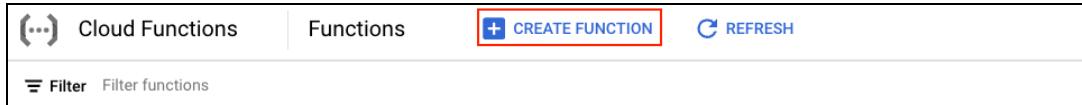
Follow the steps below:

6. Search for the **cloud functions** in the GCP console, and select Cloud Functions from the search result

The screenshot shows the Google Cloud search interface with the query "cloud functions". The search results list includes "Cloud Functions" (Event-driven serverless functions), "Cloud Run" (Serverless for containerized applications), and "Asset Inventory" (IAM & Admin). The "Cloud Functions" item is highlighted with a red box.

7. Create the Cloud Function

- a. Click on CREATE FUNCTION



3. Update the parameters, and set the parameters as marked in RED.

Note: Function name is a name entered by the user

- a. Environment : **Cloud Run Function (Default)**
- b. Function name : **list_tables_new**
- c. Region : **us-central1 (Iowa)**
- d. Trigger type : **HTTPS**
- e. Select "**Allow unauthenticated invocations**"

Cloud Functions | [Create function](#)

1 Configuration — **2 Code**

Basics

Environment — 2nd gen

Function name * list_tables_new

Region * us-central1 (Iowa)

Trigger

Trigger type — HTTPS

URL
https://us-central1-fresh-span-400217.cloudfunctions.net/list_tables_new [Copy](#)

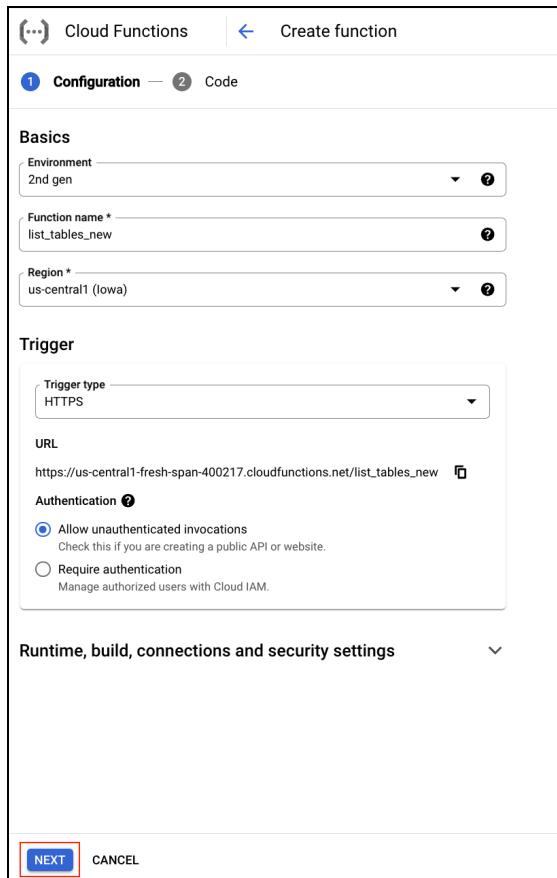
Authentication [?](#)

Allow unauthenticated invocations
Check this if you are creating a public API or website.

Require authentication
Manage authorized users with Cloud IAM.

This screenshot shows the 'Create function' configuration page in the Google Cloud Platform. The 'Configuration' tab is selected. In the 'Basics' section, the environment is set to '2nd gen', the function name is 'list_tables_new', and the region is 'us-central1 (Iowa)'. Under the 'Trigger' section, the trigger type is set to 'HTTPS'. The URL for the trigger is 'https://us-central1-fresh-span-400217.cloudfunctions.net/list_tables_new'. In the 'Authentication' section, the 'Allow unauthenticated invocations' option is selected, with a note explaining it's for creating a public API or website. The 'Require authentication' option is also present but not selected.

4. Click on NEXT



5. Update the code to point to a Flask endpoint.

(Note that the Function name from previous step, Entry point and the flask endpoint should be same)

- a. Select Python 3.12 in Runtime
- b. Update Entrypoint to list_tables_new
- c. Update the python code in main.py

This code lists all the tables present in the input BigQuery Database.

Code for main.py: Note: "Please verify code indentation after copying."

```
from flask import Flask, request, make_response, jsonify
import json
from google.cloud import bigquery
```

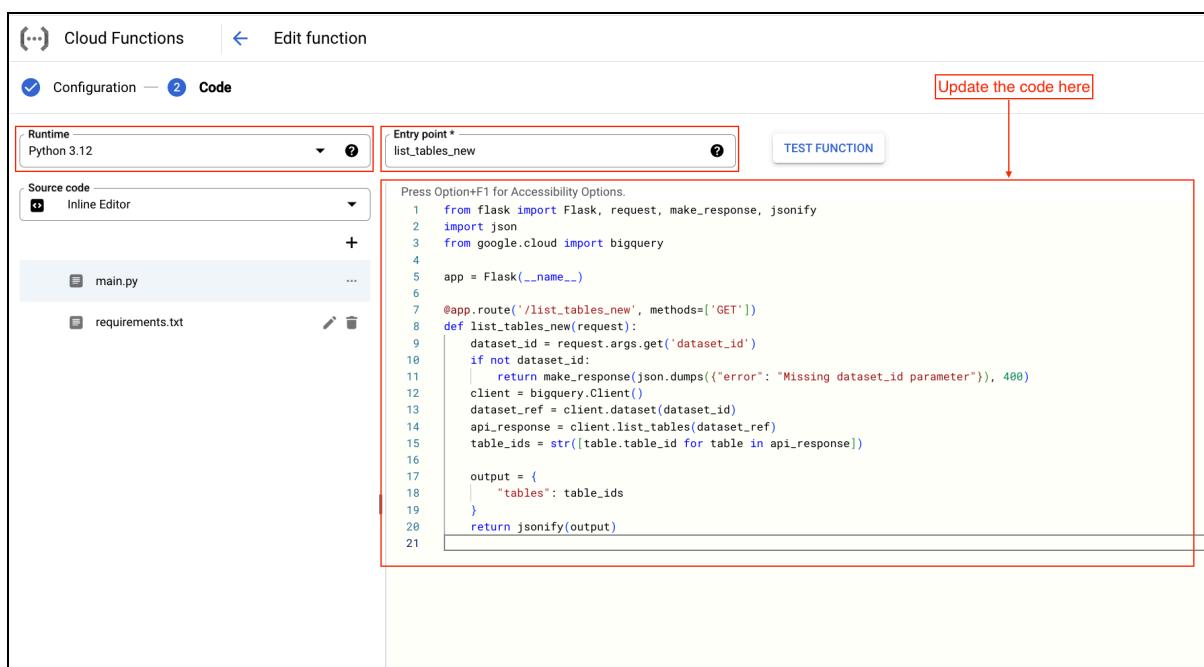
```
app = Flask(__name__)
```

```

@app.route('/list_tables_new', methods=['GET'])
def list_tables_new(request):
    dataset_id = request.args.get('dataset_id')
    if not dataset_id:
        return make_response(json.dumps({"error": "Missing dataset_id parameter"}), 400)
    client = bigquery.Client()
    dataset_ref = client.dataset(dataset_id)
    api_response = client.list_tables(dataset_ref)
    table_ids = str([table.table_id for table in api_response])

    output = {
        "tables": table_ids
    }
    return jsonify(output)

```

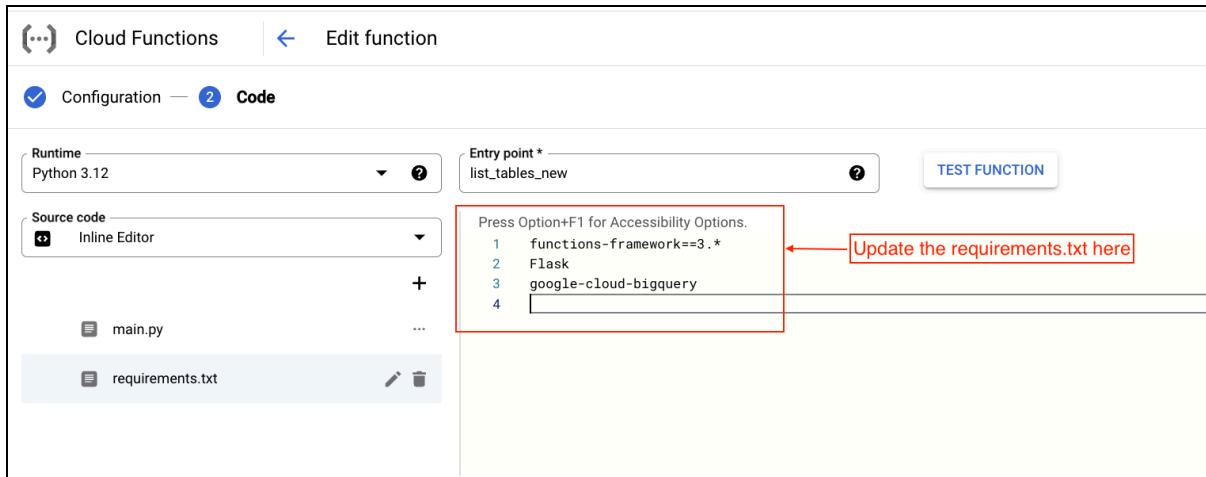


d. Update the requirements.txt file

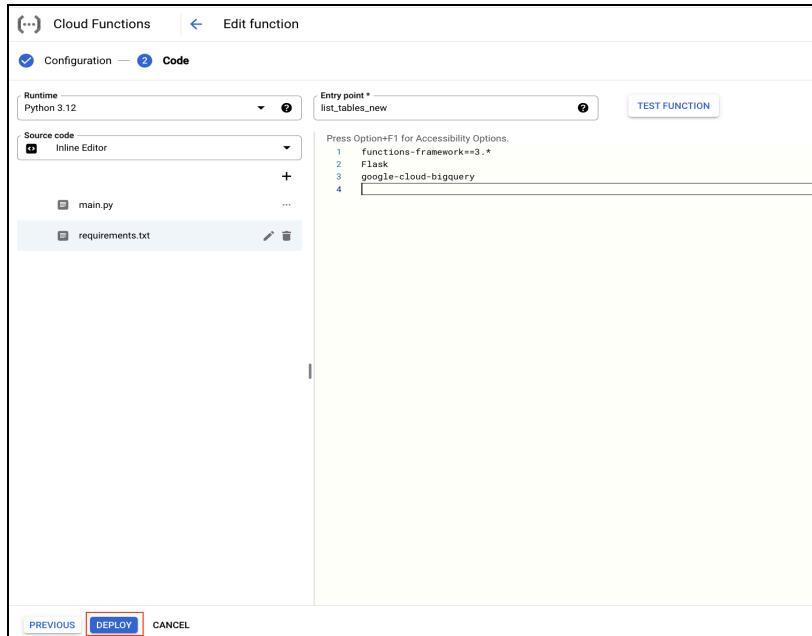
e. Requirements.txt

functions-framework==3.*

Flask google-cloud-bigquery



6. Click on Deploy



This will deploy the cloud function that can list all the tables present in the BigQuery database
Debug Steps: Make sure the Entry point and function name are same

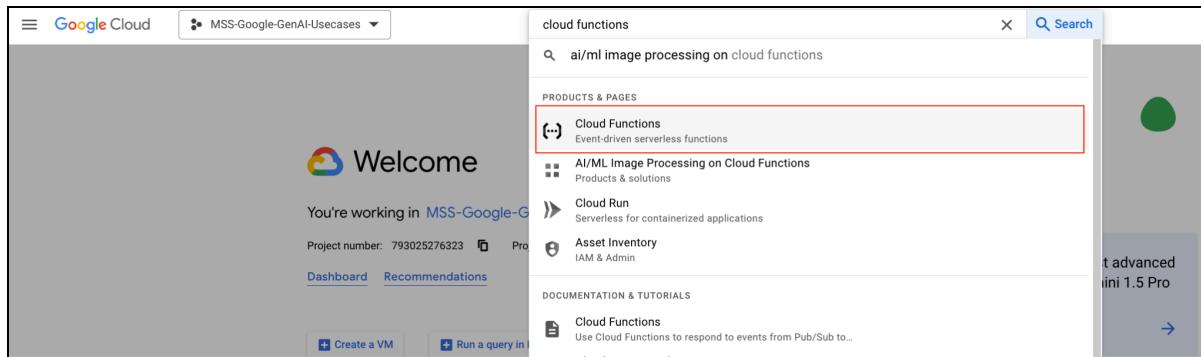
Step 2: Create a tool to get the table schema of a specific table from the BigQuery

To create a tool, 1st go to the cloud console, and create a function named `get_table_new`.

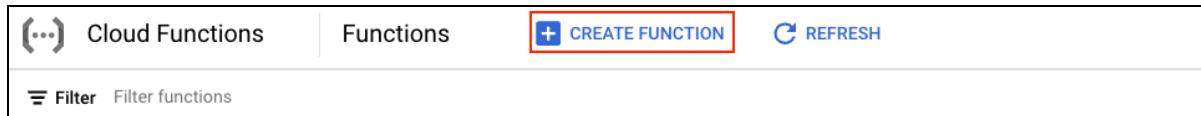
Note: pass the Flask endpoint in this function, to get the table schema from BigQuery

Follow the steps below:

1. Search for the cloud functions in the GCP console, and select **Cloud Run Functions** from the search result



2. Create the Cloud Function
 - f. Click on CREATE FUNCTION



3. Update the parameters and set the parameters as marked in RED.

Note: Function name is a name entered by the user

- b. Environment : **Cloud Run Function (Default)**
- g. Function name : **get_table_new**
- h. Region : **us-central1 (Iowa)**
- i. Trigger type : **HTTPS**
- j. Select "**Allow unauthenticated invocations**"

Cloud Functions [Create function](#)

1 Configuration — **2 Code**

Basics

Environment — 2nd gen

Function name * — get_table_new

Region * — us-central1 (Iowa)

Trigger

Trigger type — HTTPS

URL
https://us-central1-fresh-span-400217.cloudfunctions.net/get_table_new [Copy](#)

Authentication ?

Allow unauthenticated invocations
Check this if you are creating a public API or website.

Require authentication
Manage authorized users with Cloud IAM.

Runtime, build, connections and security settings [▼](#)

4. Click on NEXT

Cloud Functions | [Create function](#)

1 Configuration — **2 Code**

Basics

Environment — 2nd gen

Function name * — get_table_new

Region * — us-central1 (Iowa)

Trigger

Trigger type — HTTPS

URL
https://us-central1-fresh-span-400217.cloudfunctions.net/get_table_new

Authentication ?

Allow unauthenticated invocations
Check this if you are creating a public API or website.

Require authentication
Manage authorized users with Cloud IAM.

Runtime, build, connections and security settings

NEXT CANCEL

This screenshot shows the 'Create function' configuration page in the Google Cloud Platform. It's divided into two main sections: 'Configuration' (step 1) and 'Code' (step 2). In the 'Configuration' section, under 'Basics', the environment is set to '2nd gen'. The function name is 'get_table_new'. The region is 'us-central1 (Iowa)'. Under 'Trigger', the trigger type is 'HTTPS' and the URL is 'https://us-central1-fresh-span-400217.cloudfunctions.net/get_table_new'. The 'Authentication' section shows that 'Allow unauthenticated invocations' is selected. Below this, there's a 'Runtime, build, connections and security settings' section which is currently collapsed. At the bottom, there are 'NEXT' and 'CANCEL' buttons, with 'NEXT' being highlighted with a red border.

5. Update the code to point to a Flask endpoint.

Note that the Function name from previous step, Entry point and the flask endpoint should be same

- f. Select Python 3.12 in Runtime
- g. Update Entrypoint to get_table_new
- h. Update the python code in main.py

Code for [main.py](#): Note: "Please verify code indentation after copying."

```
from flask import Flask, request, make_response, jsonify
import json
```

```
from google.cloud import bigquery

app = Flask(__name__)

def schemafield_to_dict(schema):
    """Converts a list of SchemaField objects to a list of dicts."""
    return [
        {
            'name': field.name,
            'field_type': field.field_type,
            'mode': field.mode,
            'description': field.description,
            'fields': schemafield_to_dict(field.fields) if field.fields else []
        }
        for field in schema
    ]

@app.route('/get_table_new', methods=['GET'])
def get_table_new(request):
    # Parse JSON payload from the request body
    print(f'Request: {request}')
    try:
        table_id = request.args.get('table_id')
    except Exception as e:
        print(f'Error {e}')
        return make_response(jsonify({"error": "Invalid JSON payload"}), 400)

    # Validate required parameters
    if not table_id:
        return make_response(jsonify({"error": "Missing table_id parameter"}), 400)

    print(f'Table id: {table_id}')

    # Initialize BigQuery client
    client = bigquery.Client()

    # Attempt to retrieve the table reference
    try:
        table_ref = client.get_table(table_id)
```

```

except Exception as e:
    return make_response(jsonify({"error": str(e)}), 404)

```

```

api_response = {
    "description": table_ref.description,
    "schema": schemafield_to_dict(table_ref.schema),
    "num_rows": table_ref.num_rows,
}

return jsonify(api_response)

```

The screenshot shows the Google Cloud Platform Cloud Run function editor. The top navigation bar includes 'Cloud Run functions', 'Edit function', 'Configuration', and 'Code'. The 'Code' tab is selected. On the left, there's a sidebar with 'Runtime' set to 'Python 3.12' and 'Entry point' set to 'get_table_new'. Below that are 'Source code' (with an 'Inline Editor' dropdown), 'main.py', and 'requirements.txt'. The main area contains the Python code for the 'get_table_new' function. A red box highlights the code area, and a red callout bubble with the text 'Update the code here' points to it.

```

from flask import Flask, request, make_response, jsonify
from google.cloud import bigquery
app = Flask(__name__)

def schemafield_to_dict(schema):
    """Converts a list of SchemaField objects to a list of dicts."""
    return [
        {
            'name': field.name,
            'field_type': field.field_type,
            'mode': field.mode,
            'description': field.description,
            'fields': schemafield_to_dict(field.fields) if field.fields else []
        }
        for field in schema
    ]

@app.route('/get_table_new', methods=['POST'])
def get_table_new(request):
    # Parse JSON payload from the request body
    print('Request:', request)
    try:
        data = request.get_json()
        project_id = data.get('project_id')
        table_id = data.get('table_id')
        project_id = data.get('project_id')
        dataset_id = data.get('dataset_id')
        except_exception = exception as e:
            print(f'Error: {e}')
            return make_response(jsonify({'error': 'Invalid JSON payload'}), 400)

        # Validate required parameters
        if not table_id:
            return make_response(jsonify({'error': 'Missing table_id parameter'}), 400)
        if project_id and dataset_id and project_id not in table_id:
            table_id = f'{project_id}.{dataset_id}.{table_id}'

        # Fetch table data
        client = bigquery.Client()
        query = f'SELECT * FROM {table_id}'
        query_job = client.query(query)
        results = query_job.result()
        rows = [dict(row) for row in results]
        return make_response(jsonify({'rows': rows}), 200)
    except Exception as e:
        print(f'Error: {e}')
        return make_response(jsonify({'error': 'Internal server error'}), 500)

```

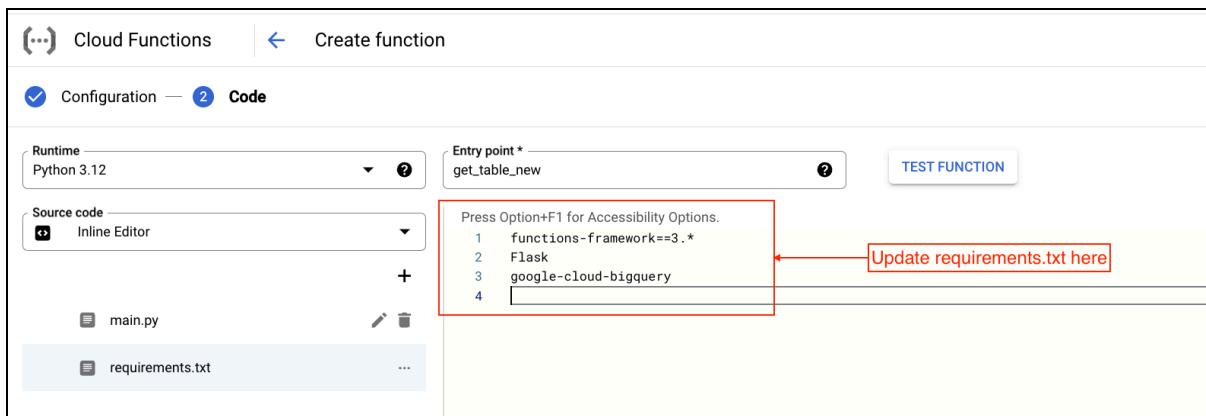
i. Update the requirements.txt file

Requirements.txt

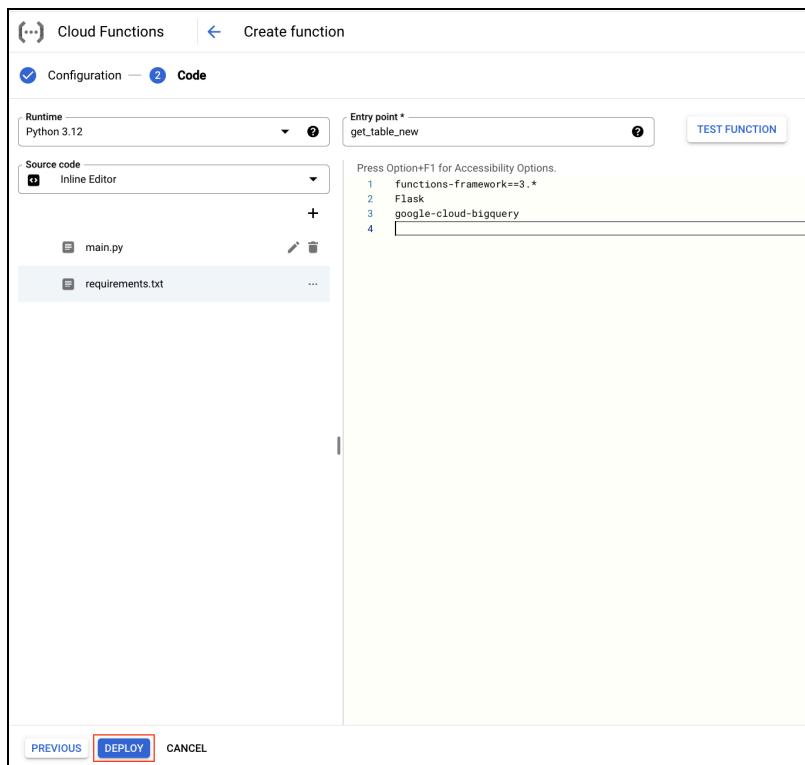
functions-framework==3.*

Flask

google-cloud-bigquery



6. Click on Deploy



This will deploy the cloud function that will get the table schema from BigQuery

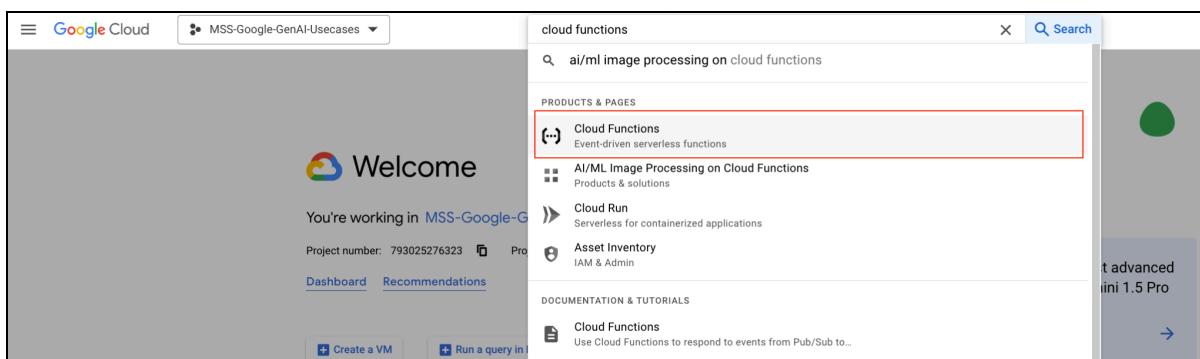
Step 3: Create a tool to execute the SQL query on BigQuery and return the corresponding result

To create a tool, 1st go to the cloud console, and create a function named **sql_query_new**.

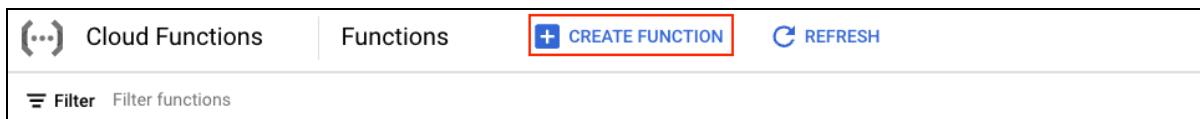
Note: pass the Flask endpoint in this function, to get the result of executing the SQL query on BigQuery

Follow the steps below:

1. Search for the cloud functions in the GCP console, and select Cloud Functions from the search result



2. Create the Cloud Function
 - k. Click on CREATE FUNCTION



3. Update the parameters

Note: Function name is a name entered by the user

- c. Environment : **Cloud Run Function (Default)**
- i. Function name : **sql_query_new**
- m. Region : **us-central1 (Iowa)**
- n. Trigger type : **HTTPS**
- o. Select "**Allow unauthenticated invocations**"

Cloud Functions | Create function

1 Configuration — 2 Code

Basics

Environment — 2nd gen

Function name * — sql_query_new

Region * — us-central1 (Iowa)

Trigger

Trigger type — HTTPS

URL
https://us-central1-fresh-span-400217.cloudfunctions.net/sql_query_new

Authentication ?

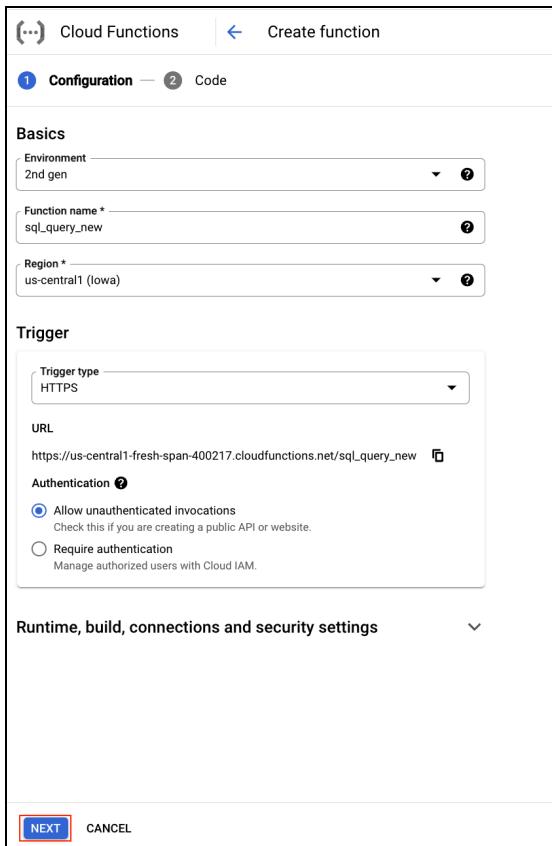
Allow unauthenticated invocations
Check this if you are creating a public API or website.

Require authentication
Manage authorized users with Cloud IAM.

Runtime, build, connections and security settings

NEXT CANCEL

4. Click on NEXT



5. Update the code to point to a Flask endpoint.

Note that the Function name from previous step, Entry point and the flask endpoint should be same

- j. Select Python 3.12 in Runtime
- k. Update Entrypoint to **sql_query_new**
- l. Update the python code in main.py

Code for main.py: Note: "Please verify code indentation after copying."

```
from flask import Flask, request, make_response, jsonify
import json
from google.cloud import bigquery

app = Flask(__name__)
```

```
@app.route('/sql_query_new', methods=['GET', 'POST'])
def sql_query_new(request):
    data = request.get_json()
    print(data)
    if not data or 'query' not in data:
        return jsonify({"error": "Missing query parameter"}), 400
    query = data['query']

    client = bigquery.Client()
    job_config = bigquery.QueryJobConfig(
        maximum_bytes_billed=1000000000
    )
    try:
        cleaned_query = (
            query
            .replace("\n", " ")
            .replace("\n", " ")
            .replace("\\", " ")
        )
        query_job = client.query(cleaned_query, job_config=job_config)
        api_response = query_job.result()
        api_response = str([dict(row) for row in api_response])
        api_response = api_response.replace("\\", " ").replace("\n", " ")
        return jsonify(api_response)
    except Exception as e:
        return jsonify(str(e))
```

Cloud Functions | Create function

Configuration — Code

Runtime — Python 3.12

Entry point * — sql_query_new

Source code — Inline Editor

main.py

requirements.txt

```

Press Option+F1 for Accessibility Options.
1  from flask import Flask, request, make_response, jsonify
2  import json
3  from google.cloud import bigquery
4
5  app = Flask(__name__)
6
7  @app.route('/sql_query_new', methods=['GET', 'POST'])
8  def sql_query_new(request):
9      data = request.get_json()
10     print(data)
11     if not data or 'query' not in data:
12         return jsonify({"error": "Missing query parameter"}), 400
13     query = data['query']
14
15     client = bigquery.Client()
16     job_config = bigquery.QueryJobConfig(
17         maximum_bytes_billed=100000000
18     )
19     try:
20         cleaned_query = (
21             query
22             .replace("\n", " ")
23             .replace("\n", "")
24             .replace("\\", "\\")
25         )
26         query_job = client.query(cleaned_query, job_config=job_config)
27         api_response = query_job.result()
28         api_response = str([dict(row) for row in api_response])
29         api_response = api_response.replace("\\", "").replace("\n", "")
30         return jsonify(api_response)
31     except Exception as e:
32         return jsonify(str(e))
33

```

TEST FUNCTION

m. Update the requirements.txt file

Requirements.txt

functions-framework==3.*

Flask

google-cloud-bigquery

Cloud Functions | Create function

Configuration — Code

Runtime — Python 3.12

Entry point * — sql_query_new

Source code — Inline Editor

main.py

requirements.txt

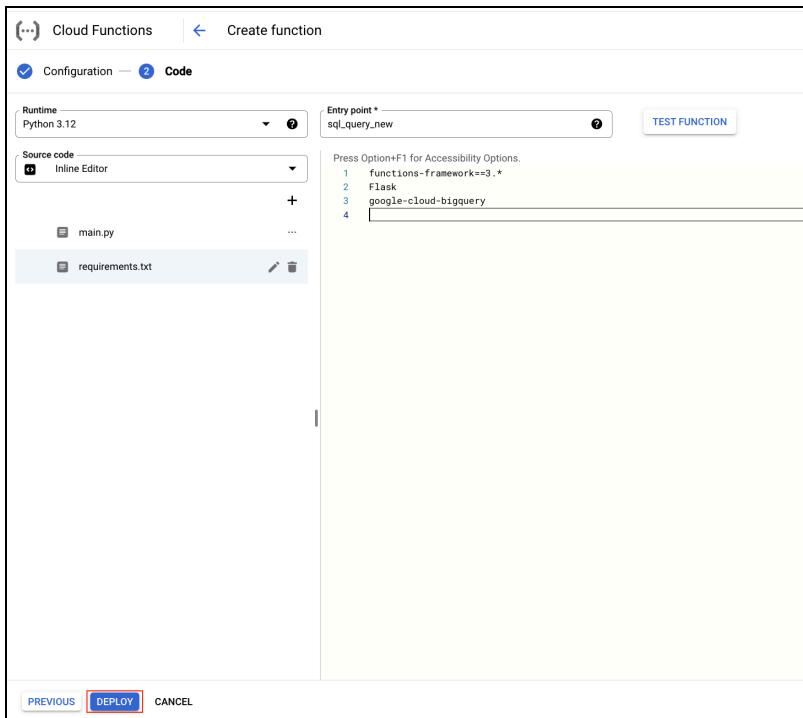
```

Press Option+F1 for Accessibility Options.
1  functions-framework==3.*
2  Flask
3  google-cloud-bigquery
4

```

TEST FUNCTION

6. Click on Deploy



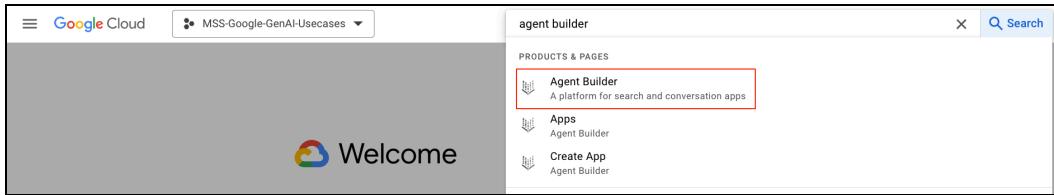
This will deploy the cloud function that will execute the SQL query on BigQuery and get the corresponding result.

Task 4. Create an Agent in Agent Builder

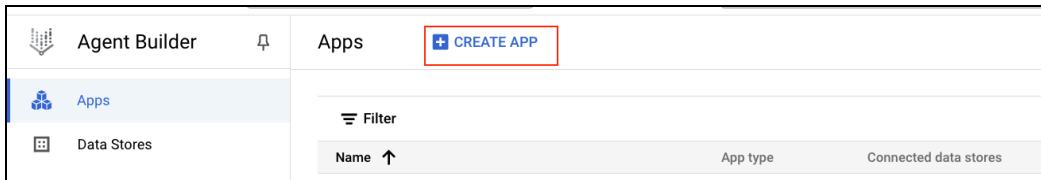
Step 4: Create an OpenAPI tool for the list_tables_new cloud function

This tool will be used to build the agent. A description of the tool and the corresponding schema should be provided here. The output format of list_tables_new cloud function and the input of this tool should be the same. If there is a mismatch, there will be errors.

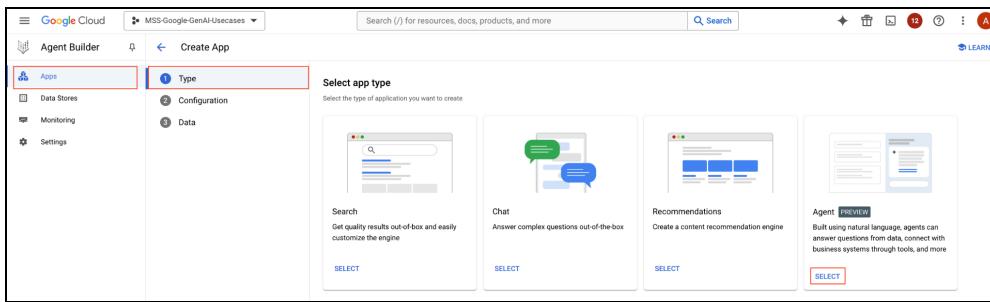
1. Go to Agent Builder on Google cloud console



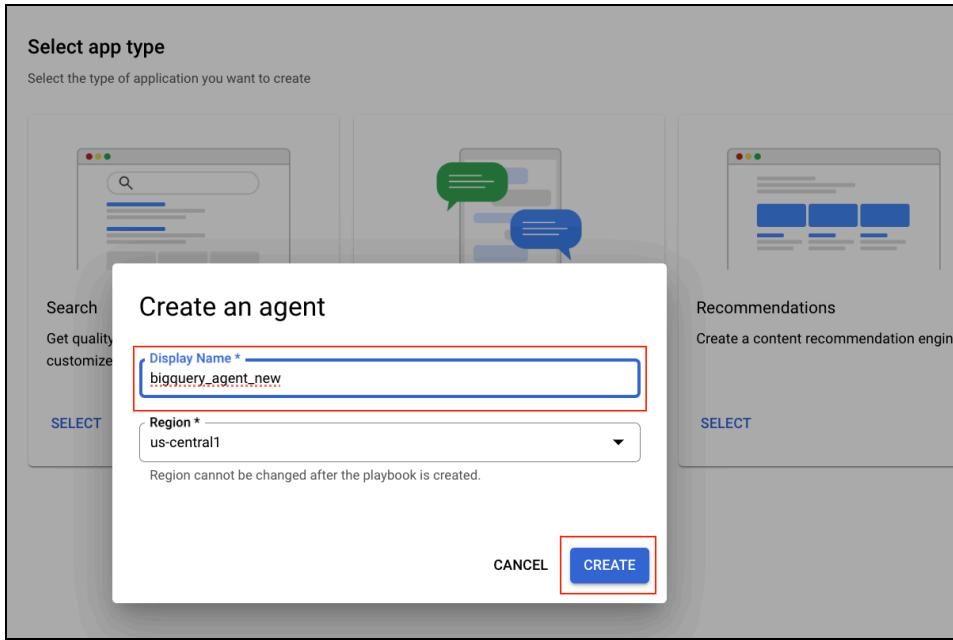
2. Click on Create App to start building a new Agent builder



3. Select Apps > Type > Agent



4. Update the Display name of the agent to **bigquery_agent_new** and click on Create



5. Select Tools from the left

6. Click on Create

7. Update the following parameters:

- Tool name : list_tables_new

- b. Type : OpenAPI
- c. Update the Description to:
*Note: Replace the project id, dataset id and table id with the correct values

This tool can be used to get the list of tables in the dataset

**Project_id: '<PROJECT_ID>'
Dataset_id: '<Dataset_id>'
Table_id: '<Table_id>'**

Example for your reference:

**Project_id: 'fresh-span-400217'
Dataset_id: 'aa_genai'
Table_id: 'sales'**

- d. Select JSON under schema
- e. Update the schema to:
* Note: Use the url corresponding to list_tables_new cloud function created previously in the schema. Set the paths to list_tables_new.
Please change the "URL" below- it will be the url of your cloud functions.

```
{
  "openapi": "3.0.2",
  "info": {
    "title": "List tables API",
    "description": "API to list all the tables in a specified dataset_id",
    "version": "1.0.0"
  },
  "servers": [
    {
      "url": "https://us-central1-fresh-span-400217.cloudfunctions.net/"
    }
  ],
  "paths": {
    "/list_tables_new": {
      "get": {
        "summary": "List all tables present in dataset_id",
        "description": "Returns the names of all tables for the specified dataset_id",
        "operationId": "list_tables",
        "parameters": [

```

```
{
  "in": "query",
  "name": "dataset_id",
  "description": "The identifier of the dataset",
  "required": true,
  "schema": {
    "type": "string"
  }
},
],
"responses": {
  "200": {
    "description": "List of all tables",
    "content": {
      "application/json": {
        "schema": {
          "type": "object",
          "properties": {
            "dataset_id": {
              "type": "string",
              "description": "The identifier of the dataset"
            },
            "tables": {
              "type": "array",
              "items": {
                "type": "string"
              },
              "description": "The list of all table names in dataset_id"
            }
          }
        }
      }
    }
  },
  "400": {
    "description": "Bad request - Invalid query parameter or missing required parameter"
  },
  "404": {

```

```
        "description": "Dataset not found"
    }
}
}
}
}
}
```

Agent Console

App: product_mart

← Tools Save

Using tools, you can connect agents to external systems. These systems can augment the knowledge of agents and empower them to execute complex tasks efficiently. [Learn more](#)

Tool name* list_tables_new

Type OpenAPI

Connect to an external API using an OpenAPI tool.

Description

Provide a description of this tool. This description is provided to the model as context informing how the tool is used.

This tool can be used to get the list of tables in the dataset
 Project_id: 'fresh-span-400217'
 Dataset_id: 'ford_product_mart_agent'
 Table_id: 'test_cust360_desc'

Schema

Provide the [OpenAPI schema](#) defining this tool's API. YAML and JSON are supported. You can use the [OpenAPI schema builder](#) to assist with building your OpenAPI schema specification.

JSON YAML

Update the JSON schema here

Samples ▾

```

1 {
2   "openapi": "3.0.2",
3   "info": {
4     "title": "List tables API",
5     "description": "API to list all the tables in a specified dataset_id",
6     "version": "1.0.0"
7   },
8   "servers": [
9     {
10       "url": "https://us-central1-fresh-span-400217.cloudfunctions.net/"
11     }
12   ],
13   "paths": {
14     "/list_tables_new": {
15       "get": {
16         "responses": {
17           "200": {
18             "description": "Success response containing the list of tables"
19           }
20         }
21       }
22     }
23   }
24 }
```

8. Save the Tool

You are currently using the **new version V2 console**, which contains the latest features.
If you find any bugs blocking your workflow, feel free to go back to V1 using banner link.

Tools

Save

Using tools, you can connect agents to external systems. These systems can augment the knowledge of agents and empower them to execute complex tasks efficiently. [Learn more](#)

Tool name*

Type

Connect to an external API using an OpenAPI tool.

Description

Provide a description of this tool. This description is provided to the model as context informing how the tool is used.

Description

Step 5: Create an OpenAPI tool for the `get_table_new` cloud function

This tool will be used to build the agent. A description of the tool and the corresponding schema should be provided here. The output format of `get_table_new` cloud function and the input of this tool should be the same. If there is a mismatch, there will be errors.

1. Select Tools from the left

You are currently using the **new version V2 console**, which contains the latest features.
If you find any bugs blocking your workflow, feel free to go back to V1 using banner link.

Agents

Tools

Tools by name, type, or description

	Type
	Extension

Conversation history

Integrations [preview]

Settings

2. Click on Create

Display name	Type
code-interpreter	Extension

3. Update the following parameters:

- a. Tool name : get_table_new
- b. Type : OpenAPI
- c. Update the Description to :

Use this tool to get information about a table, including the description, schema, and number of rows that will help answer the user's question.

Always include the Project_id and Dataset_id in Table_id

You must pass the exact complete table ID (as provided below) along with the project ID and dataset ID for accurate results.

use `fresh-span-400217.ford_product_mart_agent.cust_desc_flat` table

Project_id: '<PROJECT_ID>'

Dataset_id: '<Dataset_id>'

Table_id: '<Table_id>'

Example for your reference:

Project_id: 'fresh-span-400217'

Dataset_id: 'aa_genai'

Table_id: 'sales'

***Note: Replace the project id, dataset id and table id with the correct values**

- d. Select JSON under schema

- e. Update the schema to:

*Note: Use the url corresponding to `get_table_new` cloud function created previously in the schema. Set the paths to get_table_new.

```
"description": "Detailed information about the table",
"content": {
  "application/json": {
    "schema": {
      "type": "object",
      "properties": {
        "description": {
          "type": "string",
          "description": "A brief description of the table"
        },
        "schema": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "name": {
                "type": "string",
                "description": "Name of the field in the table"
              },
              "type": {
                "type": "string",
                "description": "Data type of the field"
              },
              "mode": {
                "type": "string",
                "description": "Mode of the field (e.g., NULLABLE, REQUIRED)"
              },
              "description": {
                "type": "string",
                "description": "Description of the field"
              }
            }
          }
        },
        "description": "Schema details of the table"
      },
      "num_rows": {
        "type": "integer",
        "description": "Total number of rows in the table"
      }
    }
  }
}
```

```
        }
    }
}
},
{
"400": {
    "description": "Bad request - Invalid query parameter or missing
required parameter"
},
"404": {
    "description": "Table not found"
}
}
}
}
```

Agent Console

App
product_mart

← Tools [Save](#)

Using tools, you can connect agents to external systems. These systems can augment the knowledge of agents and empower them to execute complex tasks efficiently. [Learn more](#)

Tool name*

Type

Connect to an external API using an OpenAPI tool.

[Update the description here](#)

Description

Provide a description of this tool. This description is provided to the model as context informing how the tool is used.

Description
that will help answer the user's question. Always include the Project_id and Dataset_id in Table_id
Project_id: 'fresh-span-400217'
Dataset_id: 'ford_product_mart_agent'
Table_id: 'fresh-span-400217.ford_product_mart_agent.test_cust360_desc'

Schema

Provide the [OpenAPI schema](#) defining this tool's API. YAML and JSON are supported. You can use the [OpenAPI schema builder](#) to assist with building your OpenAPI schema specification.

JSON YAML [Update the JSON schema here](#)

Samples ▾

```

1  {
2    "openapi": "3.0.2",
3    "info": {
4      "title": "Get table schema API",
5      "description": "API to get information about a table, including the description, schema, and
6      number of rows that will help answer the user's question. Always use the fully qualified
7      dataset and table names.",
8      "version": "1.0.0"
9    },
10   "servers": [
11     {
12       "url": "https://us-central1-fresh-span-400217.cloudfunctions.net/"
13     }
14   ],
15   "paths": {
16     ...
17   }
18 }
```

4. Save the Tool

The screenshot shows the 'Agent Console' interface for the 'bigquery_agent_new' application. A banner at the top indicates the use of the 'new version V2 console'. On the left, a sidebar menu includes 'Agents', 'Tools' (which is selected and highlighted with a red box), 'Conversation history', 'Integrations [preview]', and 'Settings'. The main content area is titled 'Tools' and contains a 'Save' button. A form for creating a new tool is displayed, with the 'Tool name*' field set to 'get_table_new' and the 'Type' field set to 'OpenAPI'. Below the form, a note says 'Connect to an external API using an OpenAPI tool.' A 'Description' section provides a detailed explanation of the tool's purpose and usage, mentioning Project_id, Dataset_id, and Table_id.

Step 6: Create an OpenAPI tool for the get_sql_query_new cloud function

This tool will be used to build the agent. A description of the tool and the corresponding schema should be provided here. The output format of get_sql_query_new cloud function and the input of this tool should be the same. If there is a mismatch, there will be errors.

1. Select Tools from the left

The screenshot shows the 'Agent Console' interface for the 'bigquery_agent_new' application. The 'Tools' option in the sidebar is selected and highlighted with a red box. The main content area displays a search bar and a table with two rows. The first row has columns 'Name' (with 'get_table_new' listed) and 'Type' (with 'OpenAPI'). The second row has columns 'Name' (with 'get_sql_query_new' listed) and 'Type' (with 'Extension').

2. Click on Create

The screenshot shows the 'Agent Console' interface with the app name 'bigquery_agent_new'. A banner at the top indicates the use of the 'new version V2 console'. Below the banner, there's a navigation bar with 'Tools' and a red-bordered 'Create' button. To the left is a sidebar with icons for Home, Tools, Search, and Help. The main area contains a search bar and a table with one row. The table has columns for 'Display name' and 'Type'. The single entry is 'code-interpreter' under 'Extension'.

Display name	Type
code-interpreter	Extension

3. Update the following parameters:

- f. Tool name : sql_query_new
- g. Type : OpenAPI
- h. Update the Description to :

*Note: Replace the project id, dataset id and table id with the correct values

Use this tool to get information about a table, including the description, schema, and number of rows that will help answer the user's question. Always include the Project_id and Dataset_id in Table_id

**Project_id: '<PROJECT_ID>'
Dataset_id: '<Dataset_id>'
Table_id: '<PROJECT_ID>.<Dataset_id>.<Table_id>'**

Example for your reference:

**Project_id: 'fresh-span-400217'
Dataset_id: 'aa_genai'
Table_id: 'fresh-span-400217.aa_genai.sales'**

- i. Select JSON under schema
- j. Update the schema to:

*Note: Use the url corresponding to **sql_query_new** cloud function created previously in the schema. Set the paths variable to **sql_query_new**

```
{  
  "openapi": "3.0.2",
```

```
"info": {
    "title": "Execute SQL Query API",
    "description": "API to execute a SQL query against a BigQuery dataset and
retrieve the results.",
    "version": "1.0.0"
},
"servers": [
    {
        "url": "https://us-central1-fresh-span-400217.cloudfunctions.net"
    }
],
"paths": {
    "/sql_query_new": {
        "post": {
            "summary": "Execute a SQL query",
            "description": "Executes a specified SQL query and returns the results
along with the table schema information.",
            "operationId": "execute_sql_query",
            "requestBody": {
                "required": true,
                "content": {
                    "application/json": {
                        "schema": {
                            "type": "object",
                            "properties": {
                                "query": {
                                    "type": "string",
                                    "description": "The SQL query to be executed."
                                },
                                "tableSchema": {
                                    "type": "object",
                                    "description": "The schema of the table that needs to be queried or
the expected schema of the result."
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```
        "name": {
            "type": "string",
            "description": "Name of the field in the table"
        },
        "type": {
            "type": "string",
            "description": "Data type of the field"
        },
        "mode": {
            "type": "string",
            "description": "Mode of the field (e.g., NULLABLE, REQUIRED)"
        },
        "description": {
            "type": "string",
            "description": "Description of the field"
        }
    }
},
"description": "Schema details of the table"
},
"num_rows": {
    "type": "integer",
    "description": "Total number of rows in the table"
}
}
},
"required": [
    "query",
    "tableSchema"
]
}
}
},
"responses": {
    "200": {
        "description": "Successful execution of SQL query",
        "content": {

```

```
"application/json": {
    "schema": {
        "type": "object",
        "properties": {
            "results": {
                "type": "string",
                "description": "The JSON string containing the results of the SQL query."
            },
            "schema": {
                "type": "object",
                "description": "The schema of the result table as processed based on the query."
            }
        }
    }
},
"400": {
    "description": "Bad request - Missing or invalid parameters"
},
"500": {
    "description": "Internal server error - Issues with executing the query or processing the result"
}
```

Agent Console App **bigrquery_agent_new**

You are currently using the **new version V2 console**, which contains the latest features.
If you find any bugs blocking your workflow, feel free to go back to V1 using banner link.

Tools Save

Using tools, you can connect agents to external systems. These systems can augment the knowledge of agents and empower them to execute complex tasks efficiently. [Learn more](#)

Tool name* **sql_query_new**

Type **OpenAPI**

Connect to an external API using an OpenAPI tool.

Update the tool description here

Description

Provide a description of this tool. This description is provided to the model as context informing how the tool is used.

This tool can be used to get information from data in BigQuery using SQL queries. The first step is to create an SQL query that can be used to answer the user question. Make sure that the SQL query is valid without any syntax errors. Use the table schema to get the correct table and column names along with the correct datatypes of the columns. Cross check the query and fix if you find any discrepancies.
Use all the Project_id, Dataset_id and Table_id in the query when necessary

Update the JSON schema here

Schema

Provide the [OpenAPI schema](#) defining this tool's API. YAML and JSON are supported. You can use the [OpenAPI schema builder](#) to assist with building your OpenAPI schema specification.

JSON **YAML**

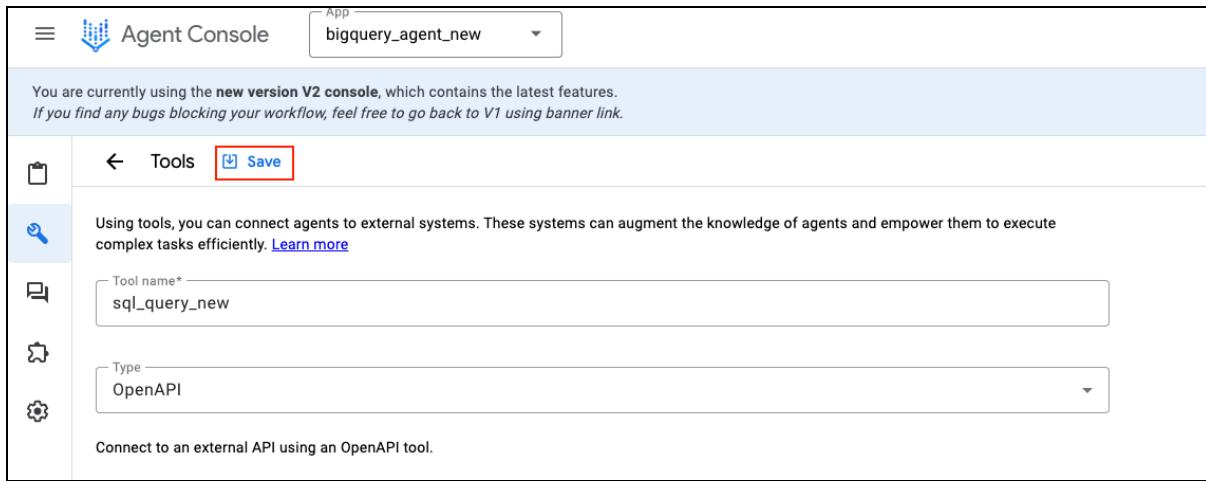
Samples

```

Schema
1  {
2    "openapi": "3.0.2",
3    "info": {
4      "title": "Execute SQL Query API",
5      "description": "API to execute a SQL query against a BigQuery dataset and retrieve the results.",
6      "version": "1.0.0"
7    },
8    "servers": [
9      {
10        "url": "https://us-central1-fresh-span-400217.cloudfunctions.net"
11      }
12    ],
13    "paths": {
14      "/sql_query_new": {
15        "post": {
16          "summary": "Execute a SQL query",
17          "description": "Executes a specified SQL query and returns the results along with the table schema information.",
18          "operationId": "execute_sql_query",
19          "requestBody": {
20            "required": true,
21            "content": {
22              "application/json": {
23                "schema": {
24                  "type": "object",
25                  "properties": {

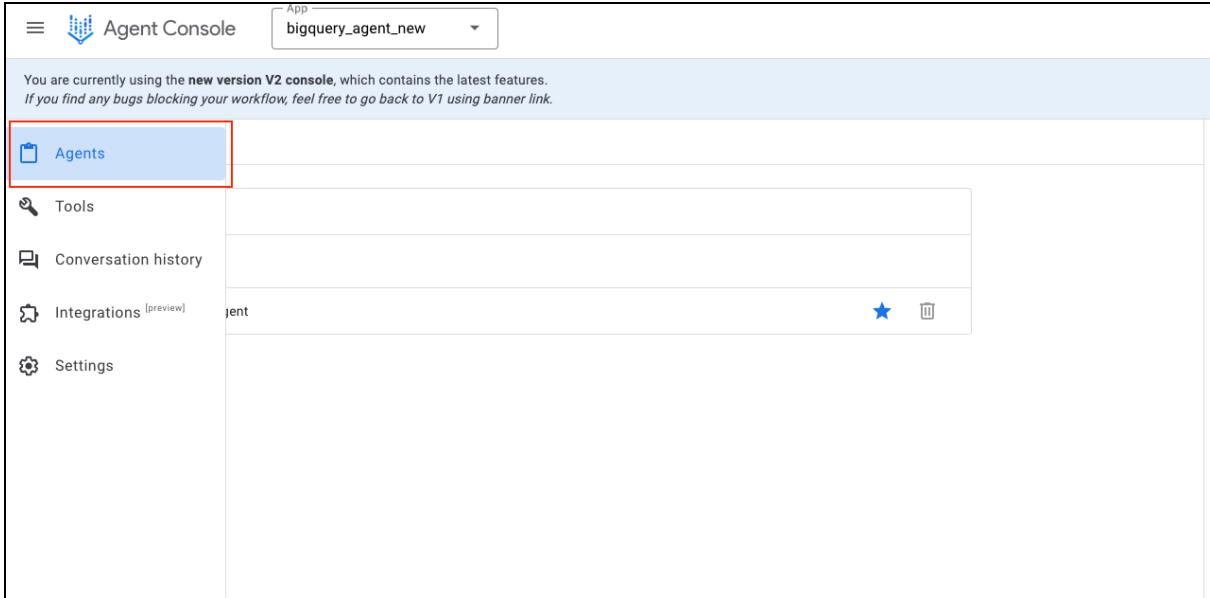
```

4. Save the Tool



Step 7: The tools created in Steps 4, 5 and 6 will be used to construct the Agent Builder

1. Click on Agent



2. Click in Create to create a new Agent

3. Update the following fields:
 - a. Agent name : Product mart *Note: Change the name of the agent as per requirement
 - b. Set the **Goal** of the Agent to :

The objective is to systematically utilize available tools in a specific sequence and answer the user question. First you need to use list_tables to get the list of tables in the dataset. Next use get_table tool to get the schema of a table. Construct an SQL query only using the table schema obtained from get_table tool. Finally use sql_query tool to run SQL queries on the data. Do not make up information. make sure you use all the tools available to generate the answer
 - c. Specify the **instructions** to be followed by the Agent. Use the syntax **\${TOOL: tool name}** to reference a tool :
 - **User questions might be directly related to the data in BigQuery. Here is the step by step plan you MUST follow to answer the question: You must use the tools in the following order:**
 - 1. You must Use **\${TOOL:list_tables_new}** to get the list of tables in the dataset
 - 2. Use **\${TOOL:get_table_new}** to get the table schema. You MUST ALWAYS pass the entire table id along with the project id and dataset id to this tool for accurate results:
 - 3. You must create an SQL query that can be used to answer the user question. Make sure that the SQL query is valid without any syntax errors and fetches the correct information required to answer the question. Use the table schema obtained in the previous step to get the correct table and column names along with the correct datatypes of the columns. Use the exact column

names from the schema and do not correct the column names. Cross check the query and fix if you find any discrepancies such as wrong syntax or wrong column names. Use backticks (`) around the column and table names when required and use the column and tables exactly as they are present in the schema. Use the Project_id and Dataset_id in the query when necessary. Use \${TOOL:sql_query_new} to run SQL queries on the data. If you are not able to execute the query and get an error from the sql_query tool, understand the error and re-write the sql query. Once the error is fixed, use the sql_query tool again to get the results required to answer the question.

- Only use the information from executing the sql query to answer the question and do not use any additional information.

Project_id: '<PROJECT_ID>'

Dataset_id: '<Dataset_id>'

Table_id: '<Table_id>'

Example for your reference:

Project_id: 'fresh-span-400217'

Dataset_id: 'aa_genai'

Table_id: 'sales'

*Note: Replace the project id, dataset id and table id with the correct values

Agent name*
Product Mart

An agent is the basic building block of a Vertex AI Conversation app. Each agent is defined to handle specific tasks. [Learn more](#)

Goal

Goal*

Answer the user question using all the tools available in the correct order. First you need to Use list_tables to get the list of tables in the dataset and next get_table to get the schema of the table and finally Use sql_query to run SQL queries on the data. The sql query should be constructed only using the table schema obtained in the previous step. Do not make up information.

High level description of the goal the agent intends to accomplish. [Learn more](#)

Instructions

Instructions

- User questions might be directly related to the data in BigQuery. Here is the step by step plan you MUST follow to answer the question: You must use the tools in the following order:
 - 1. You must Use \${TOOL:list_tables_new} to get the list of tables in the dataset
 - 2. You must Use \${TOOL:get_table_new} to get the table schema
 - 3. You must create an SQL query that can be used to answer the user question. Make sure that the SQL query is valid without any syntax errors and fetches the correct information required to answer the question. Use the table schema obtained in the previous step to get the correct table and column names along with the correct datatypes of the columns. Use the exact column names from the schema and do not correct the column names. Cross check the query and fix if you find any discrepancies such as wrong syntax or wrong column names. Use backticks (`) around the column and table names when required and use the column and tables exactly as they are present in the schema. Use the Project_id and Dataset_id in the query when necessary. Use \${TOOL:sql_query_new} to run SQL queries on the data. If you are not able to execute the query and get an error from the sql_query tool, understand the error and re-write the sql query. Once the error is fixed, use the sql_query tool again get the results required to answer the question.
 - Only use the information from executing the sql query to answer the question and do not use any additional information.
 - Project_id: 'fresh-span-400217'
 - Dataset_id: 'ford_product_mart_agent'
 - Table_id: 'test_cust360_desc'

Ordered list of step-by-step execution instructions to accomplish target goal. Specify instructions using [unordered markdown list](#) syntax. Instructions may be nested to specify substeps. Use the syntax \${TOOL: tool name} to reference a tool, and \${AGENT: agent name} to reference another agent. [Learn more](#)

4. Save the Agent

Step 8: Testing the Agent created

Follow these steps to test the Agent Builder:

1. Select the agent created, Product Mart. *Note: Select the correct agent
2. Select the GenAI model, **gemini-1.5-flash**
3. Enter the question in the text box. Example question: '**What are the top 5 locations by volume sold?**'
4. Click the **Submit** button or press Enter

1. Select the agent → 2. Select the model → 3. Enter your question → 4. Submit the question

The response and the steps taken by the Agent can be seen in the UI.

Product Mart

Goal
The objective is to systematically utilize available tools in a specific sequence and answer the user question. First you need to use list_tables to get the list of tables in the dataset. Next use get_table tool to get the schema of a table. Construct an SQL query only using the table schema obtained from get_table tool. Finally use sql_query tool to run SQL queries on the data. Do not make up information. make sure you use all the tools available to generate the answer.

Steps taken by agent

Action	Tool	Input parameters	Output parameters
list_tables	list_tables	1	1
get_table	get_table	1	1
execute_sql_query	execute_sql_query	1	1

Response

There are 73 distinct product marts.

Check the input and output for each tool by expanding the tool block

BigQuery Agent new

Goal*
First you need to Use list_tables to get the list of tables in the dataset and next get_table to get the schema of the table and finally Use sql_query to run SQL queries on the data. The sql query should be constructed only using the table schema obtained in the previous step. Do not make up information.

Instructions

- User questions might be directly related to the data in BigQuery. Here is the step by step plan you MUST follow to answer the question: You must use the tools in the following order:
 1. You must Use \${TOOL:list_tables_new} to get the list of tables in the dataset
 2. You must Use \${TOOL:get_table_new} to get the table schema. You MUST ALWAYS pass the entire table id along with the project id and dataset id to this tool for accurate results
 3. You must create an SQL query that can be used to answer the user question. Make sure that the SQL query is valid without any syntax errors and fetches the correct information required to answer the question. Use the table schema obtained in the previous step to get the correct table and column names along with the correct datatypes of the columns. Use the exact column names from the schema and do not make up information. Make sure there are no spaces or special characters in the column names. Do not use ` in the query. Use backticks (`) around the column and table names when required and use the column and table exactly as they are present in the schema. Cross check the query and fix if you find any discrepancies such as wrong syntax or wrong column names. Use the Project_Id, Dataset_Id and Table_Id in the query when necessary. The ids and column names should not be capitalized randomly. Limit the number of elements in the response to 100 using a LIMIT statement at the end of the query. If the question has multiple parts, try to get information related to as many parts as you can.
 4. Use \${TOOL:sql_query_new} to run SQL queries on the data. If you are not able to execute the query and get an error from the database tool understand the error and correct the sql query. Once the

Preview agent: BigQuery Agent new

Action	Tool	Input parameters	Output parameters
list_tables	list_tables_new	1	1
get_table	get_table_new	1	1
execute_sql_query	execute_sql_query_new	1	1

Response

The top 5 locations by volume sold are: DES MOINES, CEDAR RAPIDS, IOWA CITY, WATERLOO, and ANKENY.

Preview agent: Product Mart
Max token count per LLM call in current turn: input 4250, output 51;

Agent invocations

Product Mart

What is the count of distinct product marts present?

list_tables Tool list_tables Action 1 Input parameters 1 Output parameters ^

Tool* list_tables Input to list_tables tool Output for list_tables tool

Action name list_tables Name of the action to be called during the tool use.

Tool input Input (dataset_id) * "ford_product_mart_agent"

Tool output Output (200) * { "tables": "['cust_desc_flat', 'product_mart_view', 'test_cust360_desc']" }

Enter user input ➤

1.

In the Google Cloud Console, on the **Navigation Menu**, click **Vertex AI > Workbench**. Select **User-Managed Notebooks**.

Congratulations!

In this lab, you have successfully learnt about how to build an agent using the Agent Builder on structured BigQuery datasets.