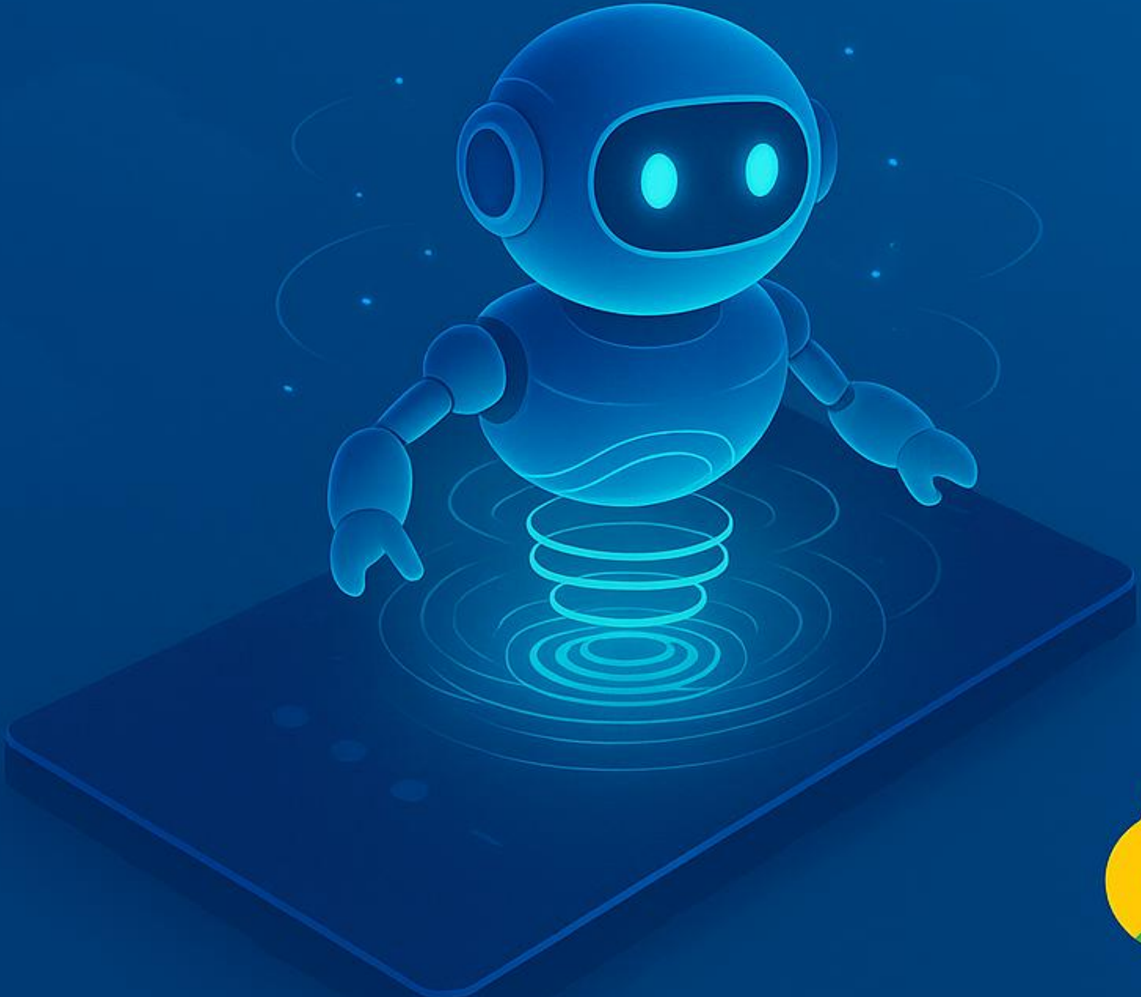
DILYTICS

Google Cloud AI Chatbot Roadmap Document



Google Cloud

Table of Contents

1. Introduction

Dilytics proudly presents the future of enterprise intelligence! Our state-of-the-art chatbot, built on Google Cloud Platform (GCP), transforms how businesses interact with data. Seamlessly integrating Dialogflow CX, Vertex AI, and BigQuery, and deployed via Azure Bot Services to platforms like Google Meet, Google Chat, Microsoft Teams and so on this solution delivers real-time, natural language-driven insights—eliminating the need for complex reports or dashboards.

Ask a question, and within fractions of a second, get precise answers from data hosted in BigQuery! Powered by Dilytics' innovative natural language to SQL (NL2SQL) technology and Vertex AI, our chatbot intelligently interprets user queries and executes optimized queries against BigQuery tables. Hosted on Cloud Functions or Cloud Run, the backend ensures blazing-fast responses, turning raw data into conversational insights. Delivered through familiar platforms like Google Meet, Google chat, Slack, Facebook Messenger, and websites etc Dilytics' chatbot empowers users with instant, actionable information—effortlessly and efficiently.

Key Capabilities Include:

• **Natural Language Interaction for Querying Predefined Datasets**
Enables users to engage with data conversationally, using everyday language to explore insights from complex datasets without writing code or understanding SQL. This intuitive interface simplifies data access for business users at all levels.

• **Vertex AI for Enhanced Language Understanding and Contextual Responses**
Leverages Google's advanced AI models to deeply understand user intent, handle contextual nuances, and improve the relevance and accuracy of responses over time—ensuring intelligent, human-like interactions.

• **Dialogflow CX as the Central Conversational Engine**
Serves as the intelligent core of the chatbot, managing conversation flows, maintaining context, and orchestrating interactions across multiple turns and user intents with enterprise-grade scalability and control.

• **BigQuery as the Enterprise-Grade Data Source**
Acts as the high-performance analytics engine that stores and processes large volumes of structured data, enabling real-time querying, robust security, and seamless integration with the broader GCP ecosystem.

• **Cloud Functions / Cloud Run for Backend Orchestration**
Provides a scalable, event-driven backend layer that executes business logic, processes NL2SQL conversions, connects to BigQuery, and formats results—ensuring efficient, modular, and maintainable backend operations.

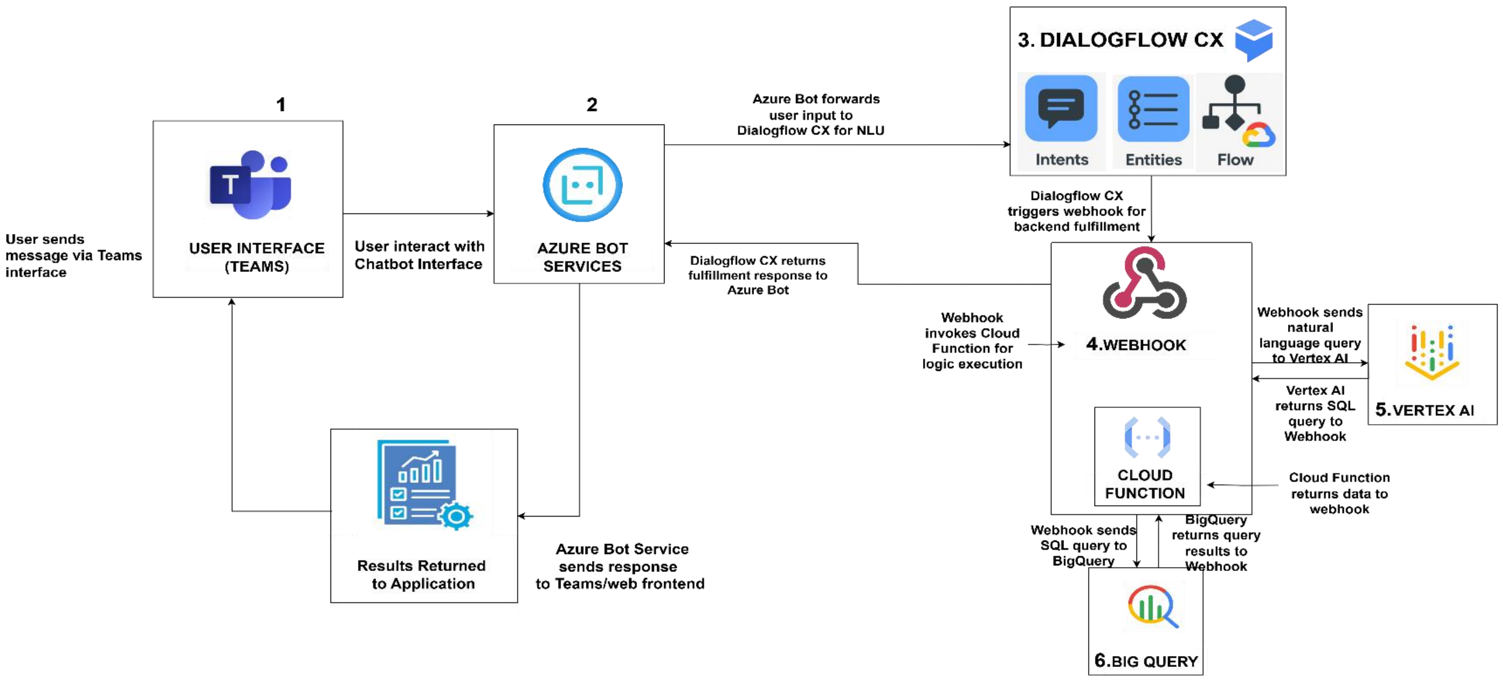
• **Azure Bot Services for Multi-Channel Delivery Including Microsoft Teams**
Delivers the chatbot experience across multiple platforms with minimal setup, allowing seamless integration with Microsoft Teams and other communication tools to ensure users can access insights directly within their daily workflows.

This solution enables self-service access to business data in a secure, scalable, and user-friendly way—bridging GCP's AI capabilities with Microsoft's enterprise communication tools.

2. System Architecture (User Input to Bot Response):

This system architecture outlines the end-to-end flow of a Google Cloud Platform-based chatbot solution, designed to interact with users via Microsoft Teams. The architecture integrates Azure Bot Services with Dialogflow CX, enabling natural language understanding and backend fulfillment through Google Cloud services. When a user submits a query, the message is routed via Azure Bot Services to Dialogflow CX, where intents and entities are processed. A webhook then invokes a Cloud Function that leverages Vertex AI for natural language-to-SQL conversion and executes the generated query in BigQuery. The resulting data is passed back through the system, ultimately returning insights to the user interface in real-time.

Dialogflow CX orchestrates the conversation flow between the user and backend services. It processes user inputs, identifies intents using natural language understanding (NLU), and routes them to appropriate fulfillment services such as Cloud Functions. Acting as the core conversational engine, Dialogflow CX seamlessly integrates with Vertex AI for NL2SQL tasks and returns structured responses through the user interface.


3. DIALOGFLOW CX
Intents
Entities
Flow
1
2
Azure Bot forwards user input to Dialogflow CX for NLU
User sends message via Teams interface
USER INTERFACE (TEAMS)
User interact with Chatbot Interface
AZURE BOT SERVICES
Dialogflow CX returns fulfillment response to Azure Bot
Dialogflow CX triggers webhook for backend fulfillment
4.WEBHOOK
Webhook invokes Cloud Function for logic execution
Webhook sends natural language query to Vertex AI
Vertex AI returns SQL query to Webhook
5.VERTEX AI
CLOUD FUNCTION
Cloud Function returns data to webhook
Webhook sends SQL query to BigQuery
BigQuery returns query results to Webhook
6.BIG QUERY
Results Returned to Application
Azure Bot Service sends response to Teams/web frontend

3. End-to-End Workflow of the Chatbot Using GCP Services

1. User Initiates Conversation

The interaction begins when a user opens a supported communication channel, such as Microsoft Teams, and engages with the chatbot. This step includes user authentication using Microsoft Entra ID, ensuring secure access before any requests are processed.

2. Query Routed to Azure Bot Service

Once authenticated, the user’s natural language query is passed to the **Azure Bot Service**, which acts as the primary middleware. It manages the session state, orchestrates the dialogue flow, and forwards the request to the appropriate backend services for further processing.

3. Azure App Service Handles Backend Logic

The **Azure Bot Service** routes the request to an **Azure App Service**, which hosts the backend application. This application is responsible for processing the query logic, formatting the request, and preparing it for integration with downstream services—such as Google Cloud Functions or Dialogflow CX—for natural language processing and data retrieval.

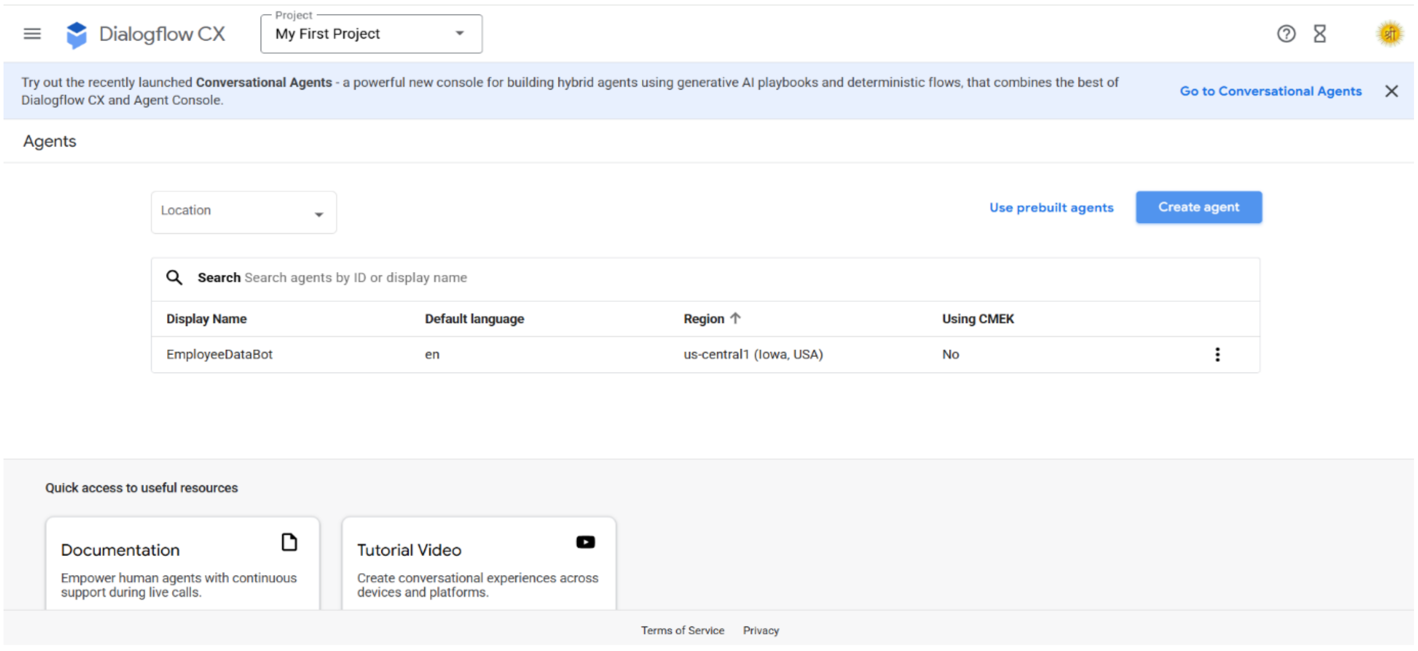
4.Processing the Query with Vertex AI & BigQuery:

The user query is received by Dialogflow CX and passed to a backend service that leverages Vertex AI to understand the natural language input. Vertex AI interprets the user’s intent and maps it to a predefined SQL query. This SQL query is then executed against BigQuery, retrieving the relevant data from connected tables. The results are then formatted appropriately and sent back to the user via the chatbot service in the Microsoft Teams channel through Azure Bot Services.

Step 1: Dialog flow CX Agent Setup

Dialog flow CX is an AI-powered tool provided by Google Cloud. It is designed to help developers and businesses create advanced and scalable chatbots and voice assistants. These chatbots can manage complex, multi-turn conversations, making them suitable for real-time customer support, virtual assistants, and other interactive systems.

What makes Dialog flow CX powerful is its ability to understand human language and it does this using a branch of artificial intelligence called Natural Language Processing (NLP). Natural Language Processing (NLP) allows machines to understand, interpret, and generate human language in a way that is meaningful. With NLP, dialog flow CX can recognize the intent behind a user's message, extract key entities or data and respond appropriately. This makes interactions with chatbots feel more natural and human-like.


Dialogflow CX
Project
My First Project

Try out the recently launched **Conversational Agents** - a powerful new console for building hybrid agents using generative AI playbooks and deterministic flows, that combines the best of Dialogflow CX and Agent Console.

Go to Conversational Agents
Agents
Location
Use prebuilt agents
Create agent
Search Search agents by ID or display name

Display Name	Default language	Region	Using CMEK
EmployeeDataBot	en	us-central1 (Iowa, USA)	No

Quick access to useful resources
Documentation
Empower human agents with continuous support during live calls.
Tutorial Video
Create conversational experiences across devices and platforms.
Terms of Service
Privacy

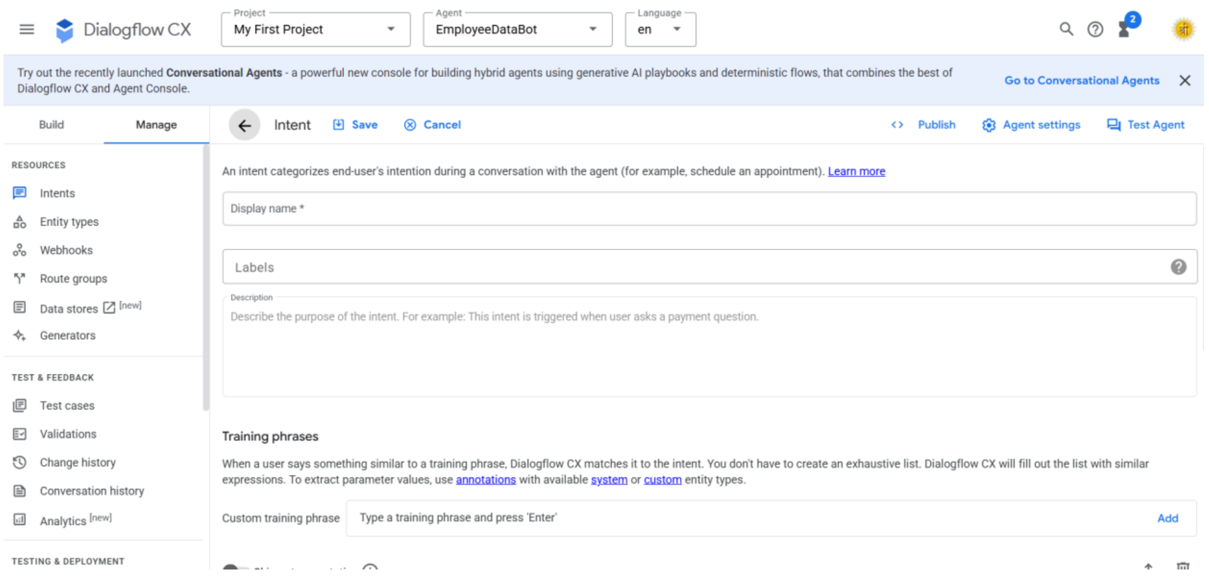
What is an Agent?

An agent in Dialog Flow CX is a virtual conversational model that handles interactions with users. It contains all the components like intents, entities, flows, and webhooks needed to build and manage a chatbot or voice assistant. It acts as the core workspace where you design, organize, and manage the entire conversation system for your chatbot.

Steps to Create an Agent in dialog flow CX:

1. **Go to Google Cloud Console:**
   Open https://console.cloud.google.com/ and log in.
2. **Select or create a GCP project:**
   Choose an **existing project** or click "**New Project**" to create one.
3. **Enable dialog flow API:**
   Navigate to **APIs & Services > Library**, search for **dialog flow API**, and click **Enable**.
4. **Go to dialog flow CX:**
   In the search bar, type **dialog flow CX** and click on the result to open the dialog flow CX console.
5. **Select location:**
   Choose a location (e.g., us-central1) where your agent will be hosted.
6. **Click "Create Agent":**
   - Give your agent a **name**.
   - Choose the **default language** (e.g., English).
   - Set the **time zone**.
   - Select the GCP **project** (if not already selected).
7. Click "**Create**" to finish.

Once created, you can begin adding intents, entities, flows, and webhooks inside your agent.



Try out the recently launched **Conversational Agents** - a powerful new console for building hybrid agents using generative AI playbooks and deterministic flows, that combines the best of Dialogflow CX and Agent Console.

An intent categorizes end-user's intention during a conversation with the agent (for example, schedule an appointment). Learn more

Describe the purpose of the intent. For example: This intent is triggered when user asks a payment question.

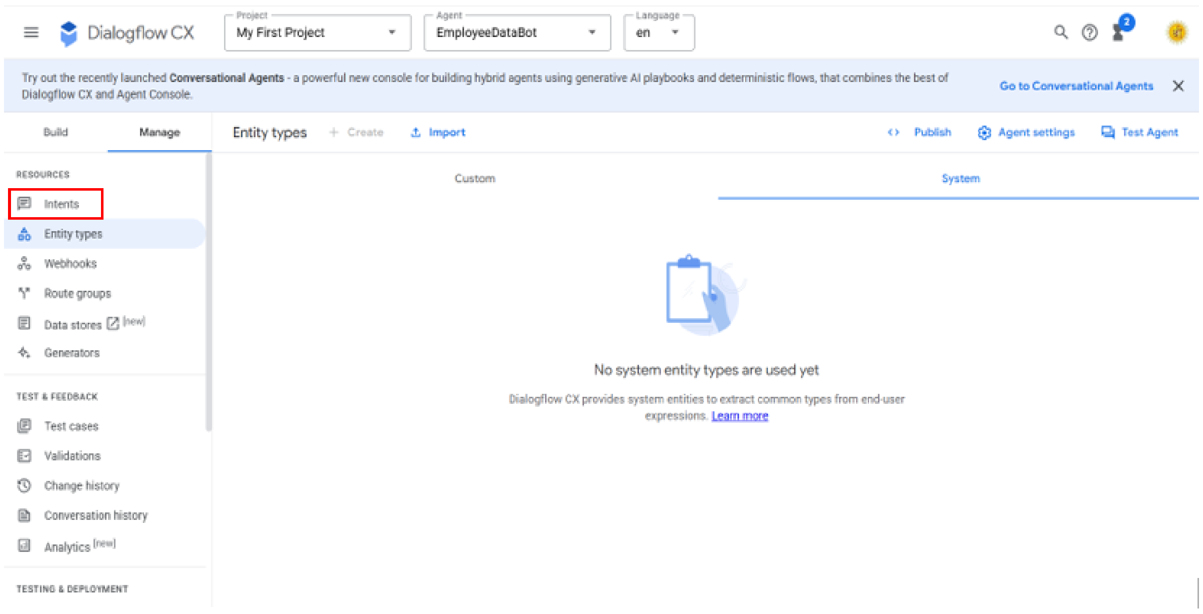
Training phrases

When a user says something similar to a training phrase, Dialogflow CX matches it to the intent. You don't have to create an exhaustive list. Dialogflow CX will fill out the list with similar expressions. To extract parameter values, use annotations with available system or custom entity types.

What is an Intent?

An intent in dialog flow CX represents the purpose or goal behind a user's message (e.g., booking a ticket, checking the weather). It is needed to help the chatbot understand what the user wants and respond with the appropriate action or answer.

In below image you can see on the left side of the interface i.e., below Resources you will find Intents.



Project: My First Project
Agent: EmployeeDataBot
Language: en

Try out the recently launched **Conversational Agents** - a powerful new console for building hybrid agents using generative AI playbooks and deterministic flows, that combines the best of Dialogflow CX and Agent Console.

No system entity types are used yet

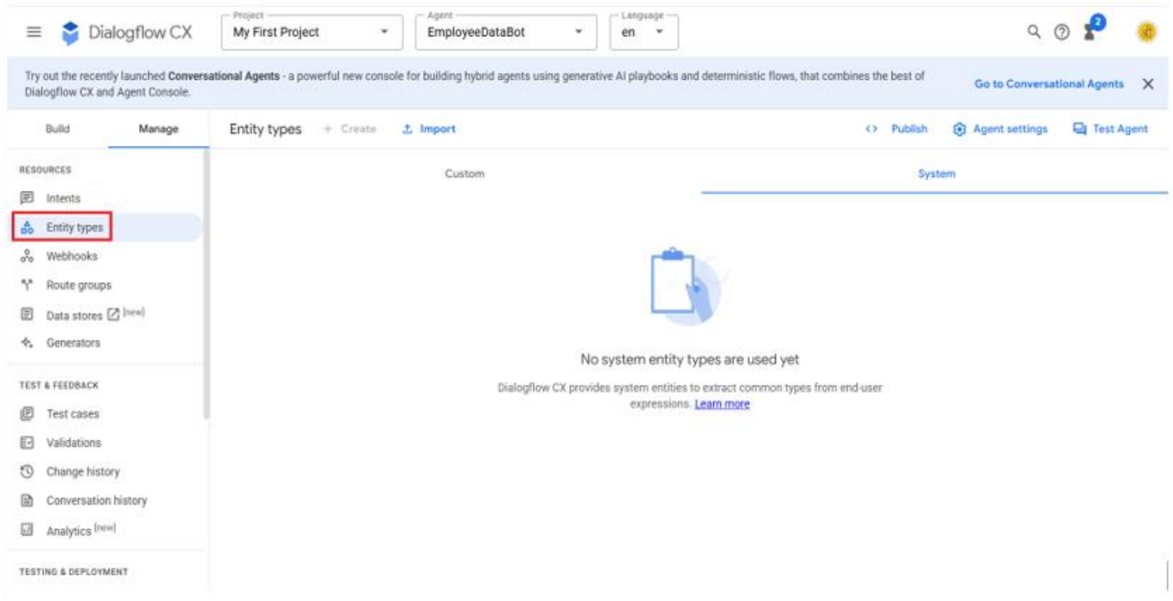
Dialogflow CX provides system entities to extract common types from end-user expressions. Learn more

How to Create an Intent in dialog flow CX

1. Open your **dialog flow CX agent** in the Google Cloud Console.
2. Click on **Manage** in the left-hand menu.
3. Under Manage, click on the **Intents** tab.
4. Click **Create Intent** at the top.
5. Give your intent a **name**.
6. Add **training phrases**—examples of what users might say to express this intent.
7. Click **Save** to create the intent.

What is an Entities?

Entities in Dialog flow CX are data elements that help the chatbot identify and extract important information from a user's input, like names, dates, locations, or any specific details needed to understand the request.

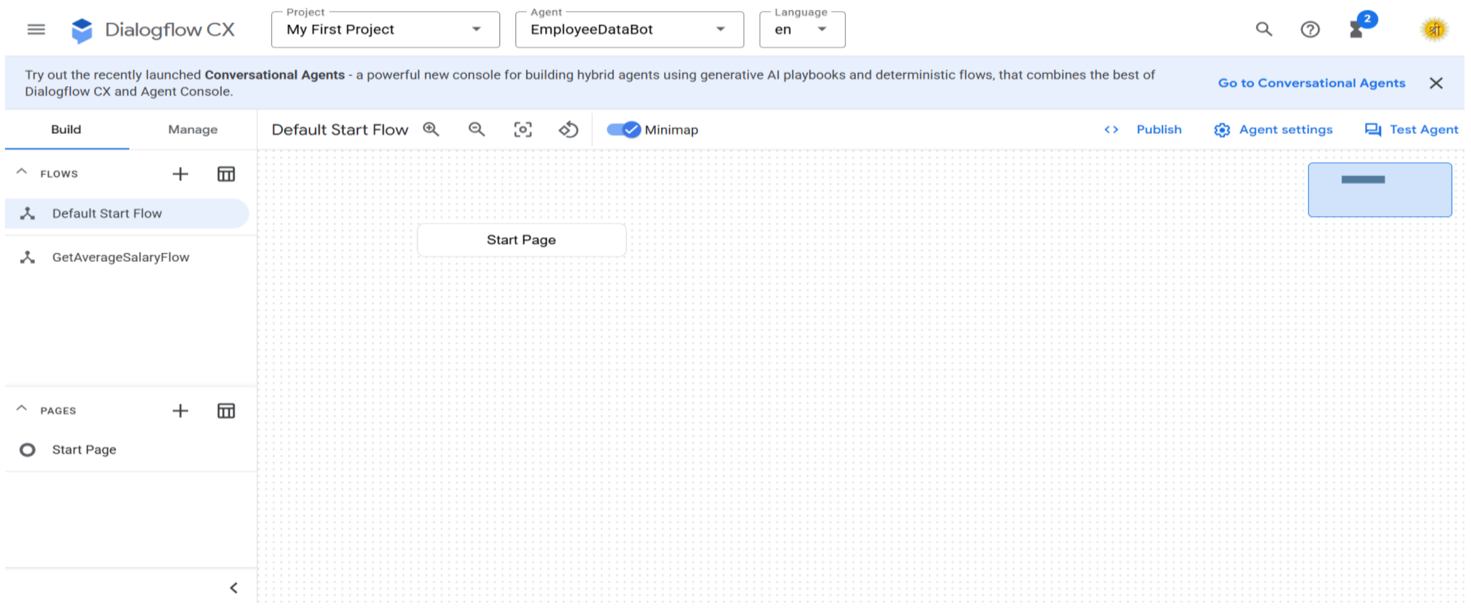

Dialogflow CX
Project My First Project
Agent EmployeeDataBot
Language en
Try out the recently launched Conversational Agents - a powerful new console for building hybrid agents using generative AI playbooks and deterministic flows, that combines the best of Dialogflow CX and Agent Console.
Go to Conversational Agents
Build
Manage
Entity types
Create
Import
Publish
Agent settings
Test Agent
RESOURCES
Intents
Entity types
Webhooks
Route groups
Data stores [new]
Generators
TEST & FEEDBACK
Test cases
Validations
Change history
Conversation history
Analytics [new]
TESTING & DEPLOYMENT
Custom
System
No system entity types are used yet
Dialogflow CX provides system entities to extract common types from end-user expressions. Learn more

How to Create an Entities in dialog flow CX

1. Open your **dialog flow CX agent** in the Google Cloud Console.
2. Click on **Manage** in the left-hand menu.
3. Under Manage, click the **Entities** tab.
4. Click **Create Entity** at the top.
5. Give your entity a **name** (e.g., Location).
6. Define **entity values** and add possible **synonyms** for each value.
7. Click **Save** to create the entity.
8. In your intent's training phrases, **annotate entities** by highlighting the relevant words and assigning the created entity.

What is a Flow?

A flow in dialog flow CX is a way to organize and manage different parts or stages of a conversation. We create flows to structure complex conversations into smaller, manageable sections, making the chatbot easier to design and control.


Dialogflow CX
Project
My First Project
Agent
EmployeeDataBot
Language
en
Try out the recently launched Conversational Agents - a powerful new console for building hybrid agents using generative AI playbooks and deterministic flows, that combines the best of Dialogflow CX and Agent Console.
Go to Conversational Agents
Build
Manage
Default Start Flow
Minimap
Publish
Agent settings
Test Agent
FLOWS
Default Start Flow
GetAverageSalaryFlow
PAGES
Start Page
Start Page

Create and Configure a Flow

1. Open your **dialog flow CX agent** in the Google Cloud Console.
2. Click on **Build** in the left-hand menu.
3. Go to the **Flows** section.
4. Click **Create Flow** or open an existing flow.
5. Add **Pages** to represent different steps in the conversation.
6. Set entry **fulfilment** or use **route groups** to connect **intents** to specific pages.
7. Define **transition routes** to guide the conversation between different intents, pages, or flows.

What is a Webhook?

A webhook is a way for dialog flow CX to communicate with an external server to get dynamic responses or perform actions during a conversation. We configure a webhook in dialog flow CX by providing the server's URL in the Webhooks section and linking it to intents or flows to enable real-time data processing or custom logic.

Configure Webhook

1. Open your **dialog flow CX agent** in the Google Cloud Console.
2. Click on **Manage** in the left-hand menu.
3. Go to the **Webhooks** tab under Manage.
4. Click **Create Webhook** at the top.

5. Enter a **Webhook name**.
6. Provide the **Webhook URL** (the endpoint of your backend service).
7. Click **Save** to create the webhook.
8. To use the webhook:
    - Go to a **Flow** → Select a **Page** or **Route**.
    - Under **Fulfilment**, enable and select your **Webhook**.
    - You can also configure **Webhook call mode** (e.g., always, conditionally).

Step 2: Deploy Backend Logic on Cloud Functions

What is a Webhook Cloud Function?

A Webhook Cloud Function is a backend function hosted on Google Cloud Functions that gets triggered by dialog flow CX when a user's input requires custom logic or dynamic responses.

Why it is needed:

To connect dialog flow CX with external services (APIs, databases, etc.)

To perform dynamic tasks like fetching data, saving user info, or making calculations

To generate custom responses based on real-time logic, not just static replies

Webhook Cloud Function needs 3 things: index.js, package. Json, and Docker file.

1)index.js

This is the main file where you write the code for what the webhook should do.

Below is the code of index.js:

```
const functions = require('@google-cloud/functions-framework');

const { BigQuery } = require('@google-cloud/bigquery');

const bigquery = new BigQuery();

functions.http('helloHttp', async (req, res) => {

  console.log('Request body:', JSON.stringify(req.body, null, 2));

  const intent = req.body?.intentInfo?.displayName || '';

  const parameters = req.body?.sessionInfo?.parameters || {};

  const session = req.body?.sessionInfo?.session || '';

  if (!intent || !req.body?.intentInfo || !req.body?.sessionInfo) {

    console.error('Invalid request: intentInfo or sessionInfo is missing or empty');

    return res.status(400).json({

      fulfillmentResponse: {

        messages: [{ text: { text: ['Sorry, the request format is invalid.'] } }]

      }

    });

  }
```

```
// Extract parameters
let department = parameters.department || 'all';
const operation = parameters.operation?.toLowerCase?.() || 'all';

department = typeof department === 'string' ? department.toLowerCase() : 'all';

// Define SQL function and response label
let field = '';
let responseLabel = '';
switch (operation) {
  case 'sum':
    field = 'SUM(salary)';
    responseLabel = 'total salary';
    break;
  case 'average':
    field = 'AVG(salary)';
    responseLabel = 'average salary';
    break;
  case 'max':
    field = 'MAX(salary)';
    responseLabel = 'maximum salary';
    break;
  case 'min':
    field = 'MIN(salary)';
    responseLabel = 'minimum salary';
    break;
  case 'count':
    field = 'COUNT(*)';
    responseLabel = 'number of employees';
    break;
    case 'retrieve':
    field = '*';
    responseLabel = 'complete employee data';
    break;
    default:
      return res.status(200).json({
        fulfillmentResponse: {
          messages: [{ text: { text: ['Sorry, I didn't understand the operation you want.'] } }]
        }
      });
  }
  // Construct query
  const query =
    department === 'all'
      ? `SELECT ${field} as result FROM \`ardent-medley-459309-h9.employee_data.salaries\``
      : `SELECT ${field} as result FROM \`ardent-medley-459309-h9.employee_data.salaries\`
WHERE LOWER(department) = @department`;
  const options = {
    query,
    params: department === 'all' ? {} : { department },
    useLegacySql: false,
  };
  try {
    const [rows] = await bigquery.query(options);
    const resultValue = rows?.[0]?.result;


    if (resultValue == null) {
      return res.json({
        fulfillmentResponse: {
          messages: [
            { text: { text: [`No data found for ${department === 'all' ? 'any departments' :
department}.`] } }
          ]
        }
      });
    }

    const formattedValue = typeof resultValue === 'number' ? resultValue.toLocaleString() :
resultValue;

    return res.json({
      sessionInfo: {
        parameters: {
          result: resultValue
        }
      },
      fulfillmentResponse: {
        messages: [
          {
            text: {
              text: [
                department === 'all'
                  ? `The ${responseLabel} across all departments is ${formattedValue}.`
                  : `The ${responseLabel} in the ${department} department is ${formattedValue}.`
              ]
            }
          }
        ]
      }
    });
```

```
  } catch (error) {
    console.error('BigQuery Error:', error);
    return res.status(500).json({
      fulfillmentResponse: {
        messages: [{ text: { text: ['There was an error fetching data. Please try again later.'] } }]
      }
    });
  }
});
```

2)package. Json

This file tells Google Cloud what tools or libraries your function needs to run.
If your code uses special packages (like for talking to APIs or databases), this file lists them.

Below is the code of package.Json:

```
{
  "name": "getaveragesalarywebhook",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "npx functions-framework --target=helloHttp"
  },
  "dependencies": {
    "@google-cloud/functions-framework": "^3.4.2",
    "@google-cloud/bigquery": "^7.0.0"
  },
  "engines": {
    "node": ">=18.0.0"
  }
}
```

3)Docker file

This file gives step-by-step instructions on how to build the environment to run your code.
It’s like telling Google Cloud:
“First, install this, then that, then run my code.”

Below is the code of docker file:

```
FROM node:22-slim

WORKDIR /app

COPY package. Json.

RUN npm install

COPY index.js.

ENV PORT=8080

CMD ["npm", "start"]
```

The index.js file keeps changing because it contains the main logic of your chatbot. The logic changes based on what you want your chatbot to do.Whenever you add new features or update responses, you need to update index.js.In the index.js file of a Cloud Function, we are able to connect to BigQuery using the BigQuery client library provided by Google Cloud.

Step 3: NL2SQL with Vertex AI + Dialogflow CX (Google Cloud Platform)

The user's natural language query is received through **Google Chat**, passed to **Dialogflow CX**, and processed using **Vertex AI**. The objective is to translate plain English into a SQL query that runs against a **BigQuery** database and returns insights conversationally.

Vertex AI is Google Cloud's platform for building and scaling machine learning applications, especially large language model (LLM)-based solutions. When integrated with **Dialogflow CX**, Vertex AI plays a critical role in natural language understanding (NLU), retrieval-augmented generation (RAG), and dynamic tool use — including generating SQL queries in real-time.

Vertex AI leverages Google's state-of-the-art foundation models (such as **Chat Bison** or **Gemini Pro**) to understand user intent and, using a pre-defined or dynamically retrieved SQL schema (listing tables, columns, and relationships), generates a valid SQL query aligned with the business question.

Once the SQL query is generated, the application securely connects to **BigQuery** and executes the query.

This response is then sent back to the user through **Dialogflow CX** and **Google Chat**, delivering a seamless, conversational analytics experience.

By combining Google Cloud's conversational AI (Dialogflow CX), LLM-based reasoning (Vertex AI), and real-time data querying (BigQuery), this solution empowers non-technical users to gain insights from enterprise datasets using plain English.

Typical Use Cases for Vertex AI in this setup:

- **Generative Responses:** Crafting natural, human-like replies beyond static text.
- **Information Extraction:** Pulling specific data points from unstructured user input.
- **Summarization:** Condensing long pieces of text (e.g., from BigQuery data) into concise answers.
- **Content Creation:** Generating dynamic content based on user prompts.
- **Reasoning and Logic:** Applying complex reasoning to provide answers based on provided context.

Steps to Integrate VertexAI with DialogflowCX:

3.1. Required APIs for VertexAI

Before integrating Vertex AI with Dialogflow CX and setting up the chatbot architecture, we need to ensure that the required APIs are enabled in our Google Cloud project.

1.1 In the GCP Console, go to **Navigation Menu > APIs & Services > Library**.
1.2 Enable these APIs:

- **a. Vertex AI API**
- **b. Dialogflow API**
- **c. Cloud Functions API**
- d. **BigQuery API** (optional, if using data)

1.3 Click **Enable** on each.

After enabling each API, you can verify it under: Navigation Menu → APIs & Services → Dashboard. This ensures all APIs are active and ready for use.

3.2. Set Up Vertex AI

Once the APIs are enabled, we can begin setting up Vertex AI to power natural language understanding and generation. Vertex AI Studio allows us to explore and test foundational models like Gemini, which can later be invoked programmatically via webhooks.

1. **Access Vertex AI Studio**: In the Google Cloud Console, navigate to: Navigation Menu → Vertex AI → Studio.
2. **Select Region**: Choose your desired region, such as us-central1. It's recommended to use the same region for Vertex AI, Cloud Functions, and Dialogflow CX to minimize latency and improve performance.
3. **Start a New Chat**: Click the "Start new chat" button under the chat section of the Vertex AI Studio.
4. **Choose desired Model**: From the list of available models, we can choose our desired model. For this implementation we are selecting Gemini 1.5 Pro. Gemini models offer advanced language understanding, generation, and reasoning capabilities.
5. **Validate using a Sample Query**: In the chat window, enter a natural language query. This helps you validate that the model understands and generates human-like responses. In production, this functionality will be embedded into a Cloud Function connected to Dialogflow.

3.3. Create the Webhook (Cloud Function)

Now that Vertex AI is configured and tested, we will build a webhook using Google Cloud Functions. This webhook acts as the bridge between Dialogflow CX and Vertex AI. It receives user queries, invokes the model with a custom prompt, and returns a generated response.

3.3.1. Open Cloud Shell or Cloud Console

a. Navigate to the Google Cloud Console. In the left navigation panel, go to **Cloud Functions**. Click **Create Function** to start building a new cloud function.

b. Set:
   - **Name**: Enter vertexAiWebhook (this identifies your function).
   - **Region**: Select us-central1 or the region you used in earlier steps.
   - **Runtime**: Choose Python 3.10 (or the latest available supported version).
   - **Trigger Type**: Set it to **HTTP**. This allows Dialogflow CX to send POST requests to this function.
   - **Authentication**: Choose **Allow unauthenticated invocations**. This is necessary for Dialogflow CX to call the webhook without requiring OAuth tokens.

c. After configuring these details, click **Next** to proceed to the code entry and dependencies section, where you'll implement the logic for calling Vertex AI.

3.3.2. Sample Webhook Cloud Function Code for Vertex AI Integration:

```
# main.py from flask import jsonify, Request from vertexai.language_models import
ChatModel import vertexai


vertexai.init(project="YOUR_PROJECT_ID", location="us-central1")
def vertex_ai_webhook(request: Request):
  req_data = request.get_json()

  # Extract tag or query from Dialogflow CX    tag = req_data.get("fulfillmentInfo",
{}).get("tag", "")
  session_info = req_data.get("sessionInfo", {})
  parameters = session_info.get("parameters", {})
    # Use the tag or combine all parameters into a prompt    prompt = f"Give the average
salary of HR Department."
  chat_model = ChatModel.from_pretrained("gemini-pro")
  chat = chat_model.start_chat()
  response = chat.send_message(prompt)

  reply = response.text

  return jsonify({
    "fulfillment_response": {
        "messages": [
          {"text": {"text": [reply]}}



        ]
      }
    })
```

5.3.3. Add Requirements File

After writing the Python logic for the webhook, the next step is to specify the external Python libraries your Cloud Function depends on. This is done using a file called requirements.txt.

Click **+ Add File** > requirements.txt

3.4: Connect Dialogflow to the Webhook

Once the Cloud Function containing the Vertex AI logic has been deployed, the next step is to connect this function to your Dialogflow CX agent. This enables Dialogflow CX to forward user queries to the webhook endpoint for dynamic processing

1. Open the Dialogflow CX console where the chatbot agent is configured.
2. Navigate to the "Manage" section and click on "Webhooks" to add or edit integrations.
3. Enter a descriptive name like vertexAiWebhook to identify this webhook.
4. Paste the URL generated by your Cloud Function deployment to enable Dialogflow to invoke it during conversations.

3.5: Use Webhook in Intent

Once your Webhook is added to Dialogflow CX, the next step is to **attach it to a specific intent**. This allows Dialogflow to forward the user's natural language query to the webhook, which in turn invokes Vertex AI to generate a dynamic response.

1. Open the specific intent in Dialogflow CX where we need to trigger the Vertex AI response.
2. Enable the webhook call option under the "Fulfillment" section.
3. Choose the previously created vertexAiWebhook from the dropdown list of available webhooks.
4. Click "Save" to apply the changes and activate webhook-based fulfillment for this intent.\

Step 4: Big Query

BigQuery is a fully-managed, serverless data warehouse provided by Google Cloud. It is used to store, analyze, and manage large volumes of data using SQL.

Why is BigQuery needed?

• Handles Big Data: It is designed to analyze terabytes or petabytes of data quickly and

efficiently.

• SQL-Based: You can use standard SQL to query your data—no need to learn a new

language.

• Serverless: No need to manage servers or infrastructure; Google takes care of it.

• Scalable & Fast: It automatically scales as your data grows and delivers fast

performance.

• Integrates Easily: Works well with tools like Looker Studio, Power BI, Python, and

Google Sheets.

Steps to Create a Dataset and Table in BigQuery:

1. Go to Google Cloud Console.

2. Enable the BigQuery API (if it's not already enabled).

3. Click the navigation menu (☰ three horizontal lines on the top left corner).

4. From the menu, select BigQuery.

This opens the BigQuery Studio, where you can run SQL queries.

5. At the top, in the "Search BigQuery resources" bar, you'll see your project name (the

one you created earlier).

6. Click the three dots (⋮) next to your project name and select "Create dataset."

• Enter the Dataset ID and choose location settings, then click Create.

7. To add a table, go to the top left side of the page and click on “Add data.”

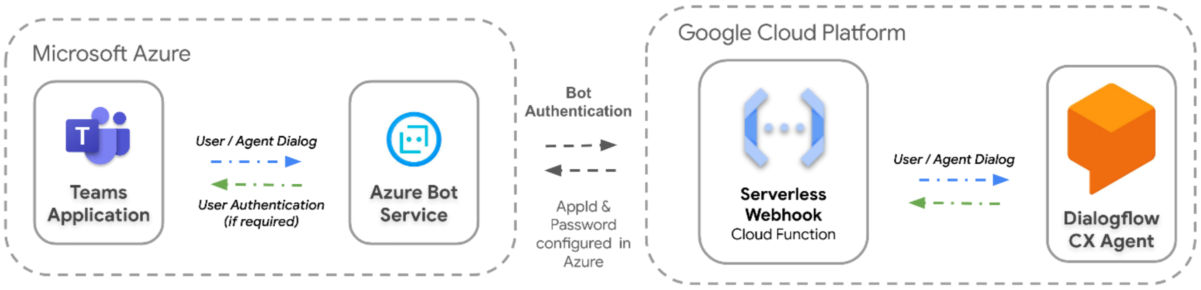
8. You’ll see multiple data source options.

- From here, you can choose where your data is coming from (e.g., upload CSV, connect to Google Sheets, etc.)

Step 5: DialogFlow CX Integration with Microsoft Teams

The integration enables users to query enterprise data via a chatbot using natural language, with support for multiple channels such as Microsoft Teams, Slack, Google Meet, Google Chat, and Web Chat. Currently, the solution is integrated with Microsoft Teams, leveraging Microsoft Azure and Google Cloud to interpret user intent, generate SQL queries, and deliver real-time results seamlessly within the Teams interface.

The example belows shows how to integrate a Dialogflow CX Agent with Azure Bot Services and enable the Microsoft Teams channel.



Key Highlights:

Seamless integration with Microsoft Teams allows users to interact with the chatbot without leaving the platform.

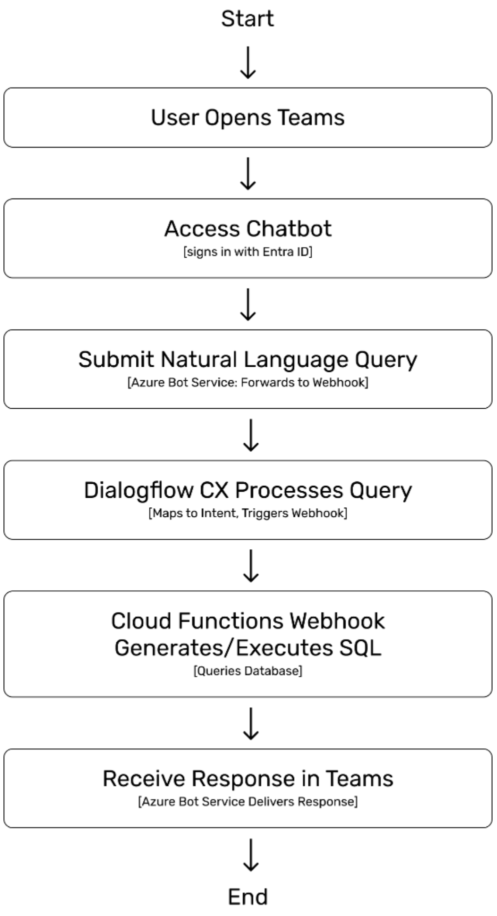
User authentication is managed through Microsoft Entra ID, ensuring secure access.

Natural language input enables non-technical users to interact with enterprise data systems.

Dialogflow CX handles query interpretation by mapping input to predefined intents and triggering appropriate backend actions.

A Cloud Function dynamically generates and executes SQL queries based on user intent, interacting with the underlying database.

Query responses are delivered back to the user within Teams via Azure Bot Service, ensuring a smooth and responsive experience.


Start
User Opens Teams
Access Chatbot
[signs in with Entra ID]
Submit Natural Language Query
[Azure Bot Service: Forwards to Webhook]
Dialogflow CX Processes Query
[Maps to Intent, Triggers Webhook]
Cloud Functions Webhook Generates/Executes SQL
[Queries Database]
Receive Response in Teams
[Azure Bot Service Delivers Response]
End

5.1 User Interaction via Microsoft Teams

This section outlines how users interact with the Natural Language to SQL (NL2SQL) chatbot powered by Google Dialogflow CX within Microsoft Teams. The chatbot enables authorized users to query enterprise data using conversational language, eliminating the need for SQL expertise. The system is designed to be intuitive, secure, and integrated into the familiar Microsoft Teams environment, with support for additional channels like Web Chat or Slack.

5.1.1 Overview

The NL2SQL chatbot allows users to input queries in plain English (e.g., "Show me last month's revenue by region" or "How many customers signed up in Q1?") via a Microsoft Teams channel, group chat, or personal chat. Dialogflow CX processes these queries, maps them to predefined intents, and generates SQL statements to retrieve data from connected databases (e.g., Google BigQuery). Responses are returned in a user-friendly format, such as text.

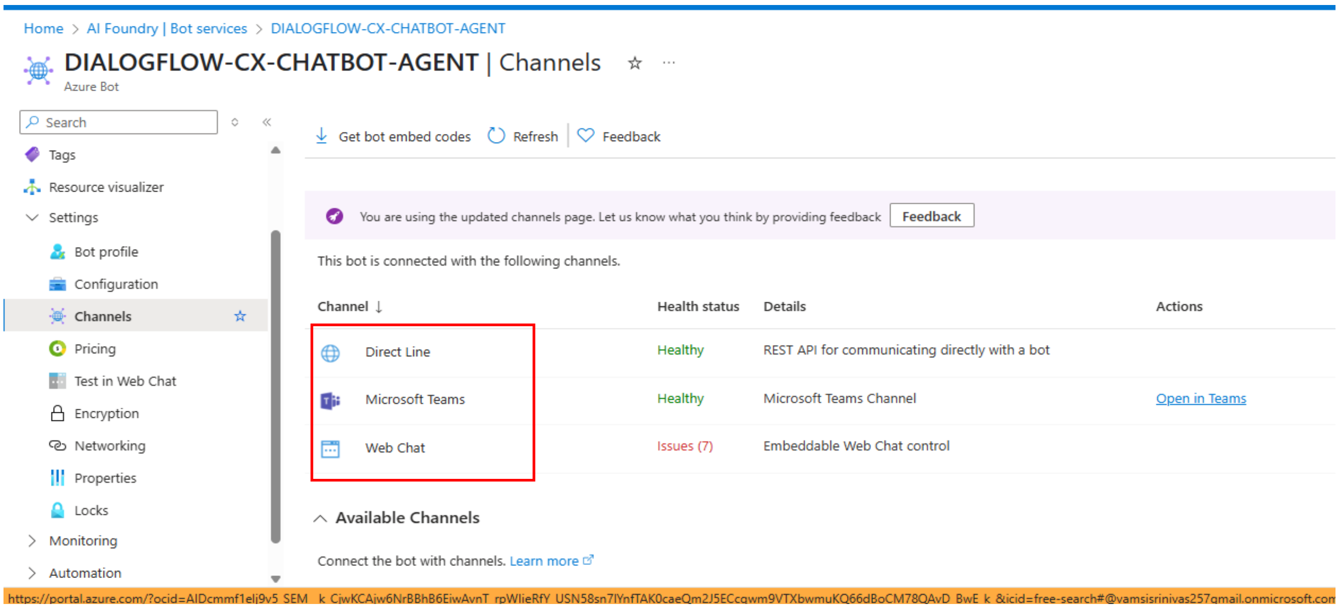
5.1.2 User Interaction Workflow

1. **Accessing the Chatbot**:
    - Users locate the chatbot (e.g., AnalyticsBot) in Microsoft Teams under "Apps" or by searching its name.
    - The bot can be added to a team, channel, or personal chat.
    - Example: A user adds the bot to the "Data Analytics" team channel.
2. **Submitting Queries**:
    - Users type natural language queries in Teams, such as:
        - *"Show me last month's revenue by region."*
        - *"List all open orders from last week."*
    - No SQL knowledge or specific syntax is required, making the system accessible to non-technical users (e.g., business analysts, managers).
3. **Receiving Responses**:
    - The chatbot processes the query using **Dialogflow CX** and returns results, such as:
        - Text:
          "Last month's revenue by region: North America ($1.2M), Europe ($0.9M)."
    - Responses are delivered in real-time within the Teams interface.

5.1.3 Supported Channels

In addition to Microsoft Teams, the chatbot supports multiple channels for flexibility:

- **Web Chat**: Embed the chatbot in internal portals or web apps using DialogFlow's Web Chat integration.
- **Slack**: Configure a Slack channel via DialogFlow's integration settings.
- **Other Channels**: Use the Dialogflow CX REST API or Direct Line API (via Azure Bot Service) to support custom channels like Facebook Messenger.



You are using the updated channels page. Let us know what you think by providing feedback Feedback

This bot is connected with the following channels.

Channel ↓	Health status	Details	Actions
Direct Line	Healthy	REST API for communicating directly with a bot	
Microsoft Teams	Healthy	Microsoft Teams Channel	Open in Teams
Web Chat	Issues (7)	Embeddable Web Chat control	

Available Channels

Connect the bot with channels. Learn more

https://portal.azure.com/?ocid=AIDcmmf1elj9v5_SEM__k_CjwKCAjw6NrBBhB6EiwAvnT_rpWlieRfY_USN58sn7lYnfTAK0caeQm2J5ECcqwm9VTXbwmuKQ66dBoCM78QAvD_BwE_k_&icid=free-search#@vamsisrinivas257gmail.onmicrosoft.com

5.1.4 User Experience Features

- **Natural Language Processing**: Dialogflow CX's advanced NLU supports complex queries, synonyms, and multilingual inputs (if configured).
- **Rich Responses**: Delivers text-based outputs for user queries, providing clear and concise data insights
- **Accessibility**: Non-technical users can query data without learning SQL, democratizing access to analytics insights.
- **Security**: Only authorized users, authenticated via Microsoft Entra ID, can interact with the chatbot (see Section 2).

5.2 Authentication Of Chatbot Using Microsoft Entra ID

Microsoft Entra ID (formerly Azure Active Directory) ensures secure access to the NL2SQL chatbot by authenticating users and enforcing role-based access control (RBAC).
This section details the authentication process and implementation steps to integrate Entra ID with the chatbot, ensuring only authorized users can query sensitive data.

5.2.1 Authentication Workflow

When a user initiates a conversation with the chatbot—say through Microsoft Teams or a web app—the first priority is to authenticate the user. That's where Microsoft Entra ID comes in:

1. **User Sign-In Request**:
   - When a user initiates a conversation with the chatbot in Microsoft Teams, the system checks for an active Entra ID session.
   - If the user is not signed in, they are redirected to the Entra ID login page (typically their Microsoft 365 credentials).
2. **Credential Verification**:
   - The user enters their organizational email and password.
   - Entra ID checks if the credentials are valid.
   - If Multi-Factor Authentication (MFA) is enabled, the user completes additional verification (e.g., authenticator app, SMS code).
3. **Token Issuance**:
   - Upon successful authentication, Entra ID issues a JSON Web Token (JWT) or OAuth2 access token.
   - The token includes claims such as user ID, email, roles, and group memberships (e.g., "Analyst," "Manager").
   - The token is attached to the user's request and sent to the middleware (hosted on Azure App Service).
4. **Authorization**:
   - The middleware validates the token using Entra ID's public key.
   - Based on the token's claims, the system enforces RBAC to restrict data access (e.g., only "Managers" can query financial data).
   - Dialogflow CX receives the user's identity and query, ensuring personalized and secure responses.

5.2.2 Implementation Steps in Azure

5.2.2.1 Register Application

1. **Navigate to App Registrations**:
   - Sign in to the Microsoft Entra admin center (https://entra.microsoft.com).
   - Select **Microsoft Entra ID** > **Manage** > **App Registrations** > **+ New registration**.
2. **Configure Application Settings**:
   - **Name**: Enter a descriptive name for your application (e.g., AnalyticsBot).
   - **Supported Account Types**: Choose who can use the application (e.g. Select "Accounts in this organizational directory only" for single-tenant access.)
   - **Redirect URI (Optional)**: Specify the redirect URI for authentication (e.g., https://<your-app-service>.azurewebsites.net/signin-oidc).
   - Click **Register** to create the application.
3. **Note Application Details**:
   - Copy the **Application (client) ID** and **Directory (tenant) ID** for use in the middleware.

5.2.2.2 Configure API Permissions

1. **Access API Permissions**:
   - In the registered application's page, navigate to **Manage** > **API Permissions** > **+ Add a permission**.
2. **Add Microsoft Graph Permissions**:
   - Select **Microsoft Graph** > **Delegated Permissions**.
   - Choose permissions: User.Read (to access user profile) and email (to retrieve user email).
   - Click **Add Permissions**.
3. **Grant Admin Consent**:
   - Click **Grant admin consent for [Your Organization]** to apply permissions organization-wide.

5.2.2.3 Generate Client Secret

1. **Navigate to Certificates & Secrets**:
   - Go to **Manage** > **Certificates & Secrets** > **Client Secrets** > **+ New client secret**.

2. **Create a Client Secret**:
    - Add a description (e.g., AnalyticsBot Secret) and set an expiration period (e.g., 1 year).
    - Click **Add**.
3. **Copy the Client Secret**:
    - Immediately copy the client's secret value, as it cannot be retrieved later.
    - Store it securely in Azure Key Vault for use in the middleware.

5.2.2.4 Security Considerations

- **MFA**: Enable MFA in Entra ID for enhanced security.
- **RBAC**: Use group memberships or roles in token claims to restrict access to sensitive data (e.g., only "Finance" group can query revenue data).
- **Token Expiry**: Configure token lifetimes in Entra ID to balance security and user experience.
- **Auditing**: Enable Entra ID audit logs to track authentication events.

Step 6. Azure Bot Service

The Azure Bot Service acts as the communication bridge between Microsoft Teams (or other channels) and the Dialogflow CX-powered NL2SQL chatbot. It handles message routing, maintains user context, and ensures secure communication with the backend middleware hosted on Azure App Service.

6.1 Overview

When a user sends a natural language query (e.g., “Show all open orders from last week”) in Microsoft Teams:

1. The Azure Bot Service receives the message via the Bot Connector Service.
2. It packages the message as a Bot Framework activity, including user identity (from Entra ID), timestamp, channel ID, and message text.
3. The activity is sent to the middleware endpoint (e.g., Google Cloud Run).
4. The middleware forwards the query to Dialogflow CX, which generates an SQL query and retrieves data via a webhook.
5. The response is sent back through Azure Bot Service to the user in Teams.

This abstraction simplifies channel management, allowing the chatbot to support Teams, Web Chat, Slack, or other channels without manual protocol handling.

6.2 Secure Initiation

- **Message Payload**: Includes user identity (from Entra ID token), timestamp, channel ID, and message text.
- **Security**: Communication between Teams and Azure Bot Service uses HTTPS (TLS 1.2+).
- **No Sensitive Logic**: The Bot Service only routes messages; all sensitive logic (e.g., SQL generation) occurs in the middleware and Dialogflow CX.

6.3 Implementation Steps in Azure

6.3.1 Create Azure Bot

1. **Navigate to Azure Portal**:
   - Go to Azure Portal.
   - Select **Create a resource** > Search for “Azure Bot” > Click **Create**.

2. **Configure Bot Settings**:
    - **Bot Handle**: Enter a unique name (e.g., AnalyticsBot).
    - **Subscription**: Select your Azure subscription.
    - **Resource Group**: Choose an existing group (e.g., rg-analytics-bot) or create a new one.
    - **Region**: Select a region close to your users (e.g., East US).
    - **Pricing Tier**: Choose based on needs (e.g., Standard for production).
    - **Microsoft App ID and Password**: Select "Auto-create App ID and password" or use existing ones.
3. **Review and Create**:
    - Click **Review + create** > After validation, click **Create**.
4. **Access Bot Resource**:
    - Navigate to the newly created bot resource in the Azure Portal.
5. **Note App ID and Password**:
    - Copy the Microsoft App ID and Password for use in the middleware.
    - Store the password securely in Azure Key Vault.

6.3.2 Add Channels

1. **Navigate to Channels**:
    - In the bot resource, go to **Settings** > **Channels**.
2. **Add Microsoft Teams Channel**:
    - Select the **Microsoft Teams** icon.
    - Read and agree to the Teams terms of service.
    - Configure settings (e.g., enable messaging, calling if needed).
    - Click **Apply**.

6.3.3 Configure Messaging Endpoint

1. **Set Messaging Endpoint**:
    - In the bot resource, go to **Settings** > **Configuration**.
    - Set the messaging endpoint to the middleware URL (e.g., https://<your-app-service>.azurewebsites.net/api/messages).
2. **Test Connectivity**:
    - Use the **Test in Web Chat** feature in Azure Bot Service to verify the endpoint.
    - Send a test message in Teams to ensure the bot responds.

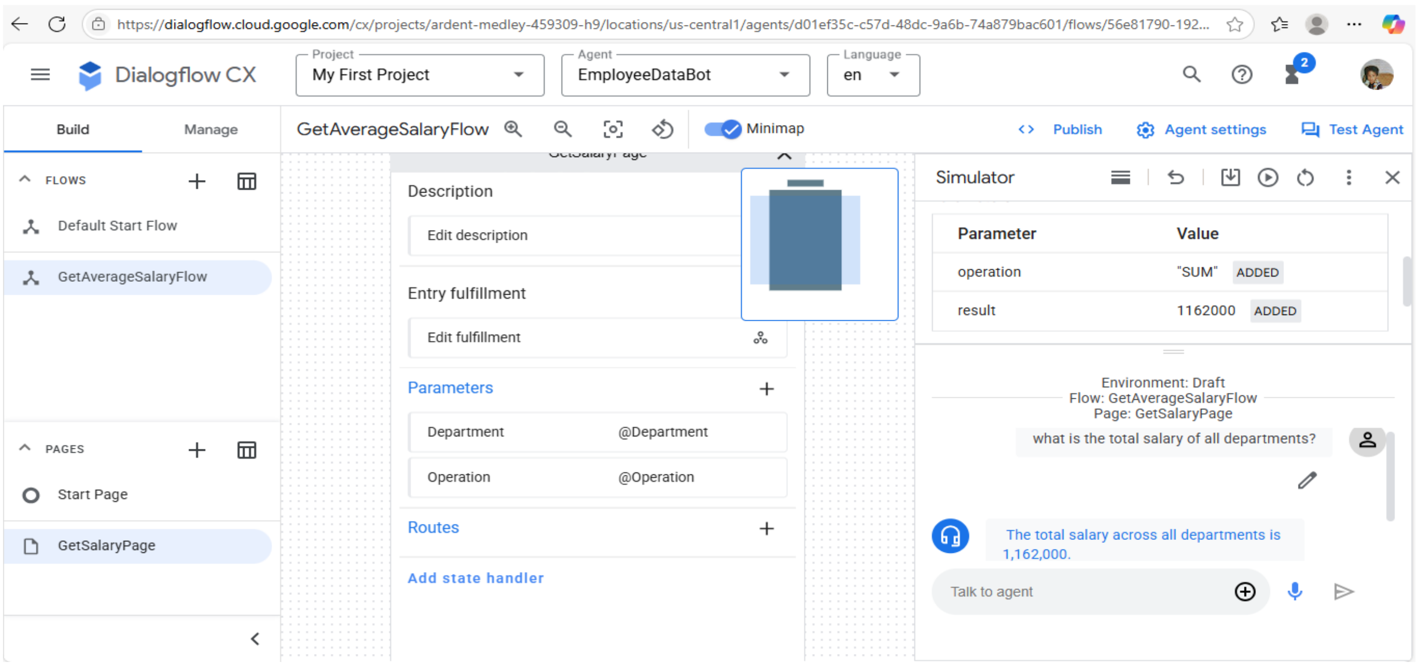
6.3.4 Dialogflow CX Integration

- **Message Processing**:
    - The middleware sends user queries to Dialogflow CX via the detectIntent API.
    - Dialogflow CX maps queries to intents (e.g., Revenue_Query, Customer_Signup) and generates SQL statements.
- **Webhook for SQL Execution**:
    - A webhook (hosted on Azure) processes Dialogflow CX responses, executes SQL queries against the database, and formats results.
    - Example: For "Show last month's revenue by region," Dialogflow generates SELECT region, SUM(revenue) FROM sales WHERE date >= '2025-04-01' AND date < '2025-05-01' GROUP BY region.
- **Context Management**:
    - Azure Bot Service maintains conversation context (e.g., user ID, session ID) and passes it to Dialogflow CX for stateful interactions.

6.3.5 Security and Performance

- **Secure Communication**: All interactions use HTTPS with TLS 1.2+.
- **User Identity**: Entra ID tokens are included in Bot Framework activities for authentication.
- **Scalability**: Azure Bot Service supports thousands of concurrent users, with auto-scaling configured in Azure App Service.
- **Monitoring**: Use Azure Application Insights to track message latency and errors.

Once integrated with platforms like Microsoft Teams, users can interact with the chatbot directly from their collaboration environment—simply create an agent and start asking questions. This screenshot from Dialogflow CX shows how the chatbot responds to a user query like *“What is the sum of salary of IT?”* It extracts key parameters (department: IT, operation: SUM), executes the query in BigQuery, and delivers the result in real time. This illustrates how natural language can be used to pull precise, actionable insights from enterprise data—right within everyday tools like Teams.



https://dialogflow.cloud.google.com/cx/projects/ardent-medley-459309-h9/locations/us-central1/agents/d01ef35c-c57d-48dc-9a6b-74a879bac601/flows/56e81790-192...

Project: My First Project
Agent: EmployeeDataBot
Language: en

GetAverageSalaryFlow

Department	@Department
Operation	@Operation

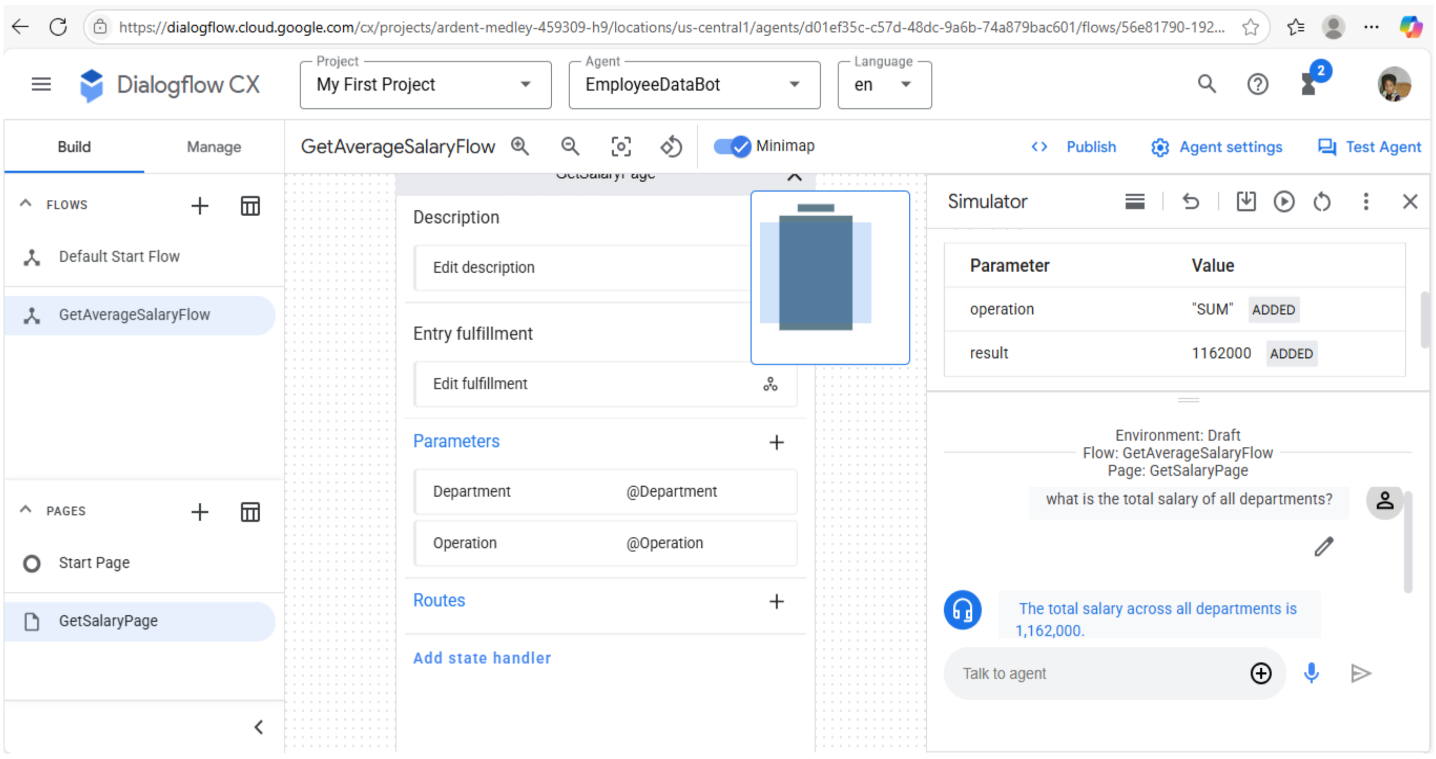
Parameter	Value	
operation	"SUM"	ADDED
result	1162000	ADDED

Environment: Draft
Flow: GetAverageSalaryFlow
Page: GetSalaryPage

what is the total salary of all departments?

The total salary across all departments is 1,162,000.

Below visual shows the chatbot handling a broader query: *"What is the total salary of all departments?"* Without specifying a department, the system intelligently adapts, performs a total aggregation (SUM operation), and responds with a consolidated figure. It demonstrates the chatbot's flexibility in handling both scoped and global queries—ideal for executive-level insights on demand.



Project: My First Project
Agent: EmployeeDataBot
Language: en

Parameter	Value
Department	@Department
Operation	@Operation

Parameter	Value	
operation	"SUM"	ADDED
result	1162000	ADDED

Environment: Draft
Flow: GetAverageSalaryFlow
Page: GetSalaryPage

what is the total salary of all departments?

The total salary across all departments is 1,162,000.

Understand Google Cloud Dialogflow pricing

Google prices Dialogflow CX monthly based on the edition, pricing plan, number of requests, the total duration of audio processed, and the total duration of the interaction. For more information about Google Cloud Dialogflow CX pricing, see the Google Cloud Dialogflow pricing page.