What is a Chatbot?

A chatbot can be defined as an AI-based computer program that simulates human conversations.

Bots interpret and process the user requests and give prompt relevant answers.

AI and Data Matter When It Comes to Chatbots

Why do we need a Chatbot?

Customer service

Online purchases and payments management

Sending information and news about your company

Assistant 24/7

What is Dialog flow?

Dialog flow (Previously known as API.AI) is where the magic happens. It works on natural language processing and backed by Machine Learning . At Dialog flow the whole ‘conversation’ take place. Dialog flow is backed by Google and runs on Google infrastructure, which means you can scale to millions of users.

Why do we need Dialog flow?

On any platform — Dialog flow support more than 20+ platforms from Google home to Twitter

Across devices — Dialog flow supports all the devices from wearables , to phones to devices.

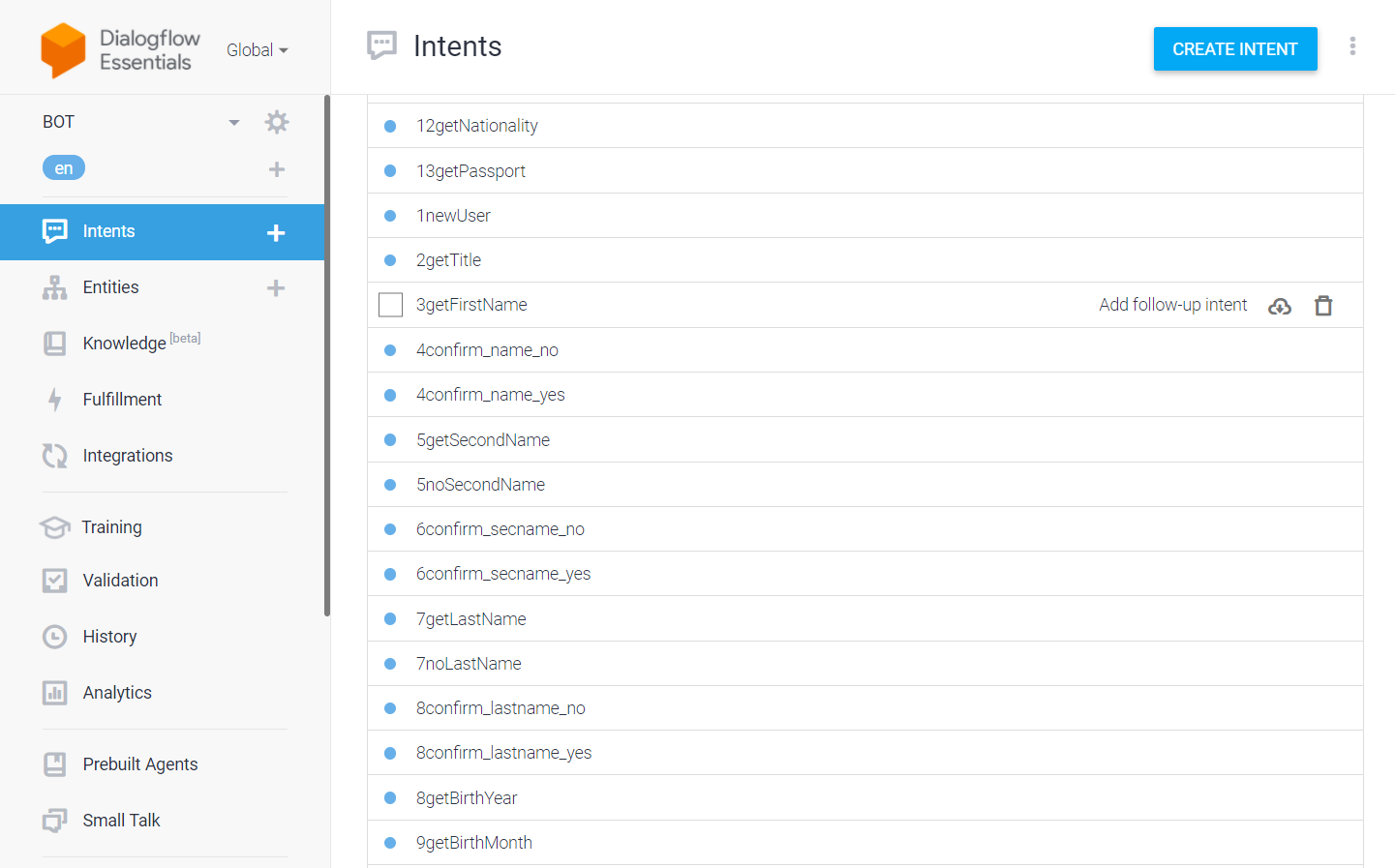
Around the world — Dialog flow supports more than 14+ languages worldwide & more support is coming.

Intent:

An intent categorizes an end-user's intention for one conversation turn. ... When an end-user writes or says something, referred to as an end-user expression, Dialogflow matches the end-user expression to the best intent in your agent. Matching an intent is also known as intent classification.

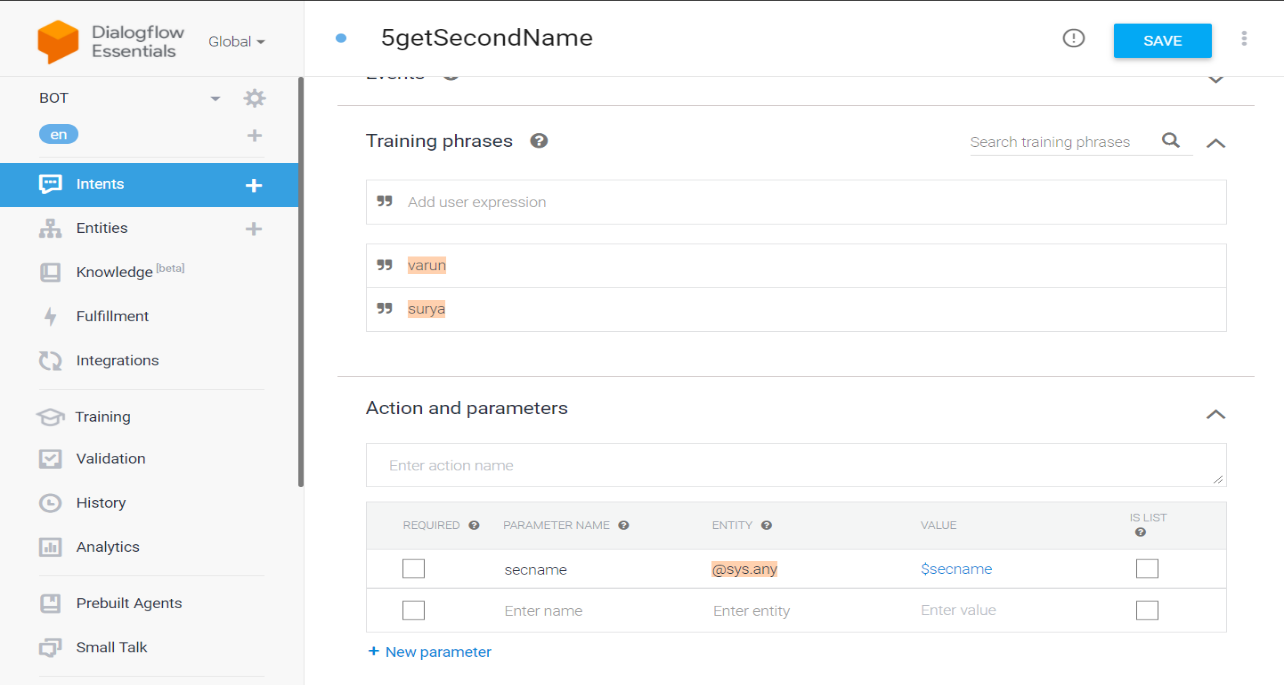
For example, you could create a weather agent that recognizes and responds to end-user questions about the weather. You would likely define an intent for questions about the weather forecast. If an end-user says "What's the forecast?", Dialogflow would match that end-user expression to the forecast intent.

These are the intents used in our chatbot application:



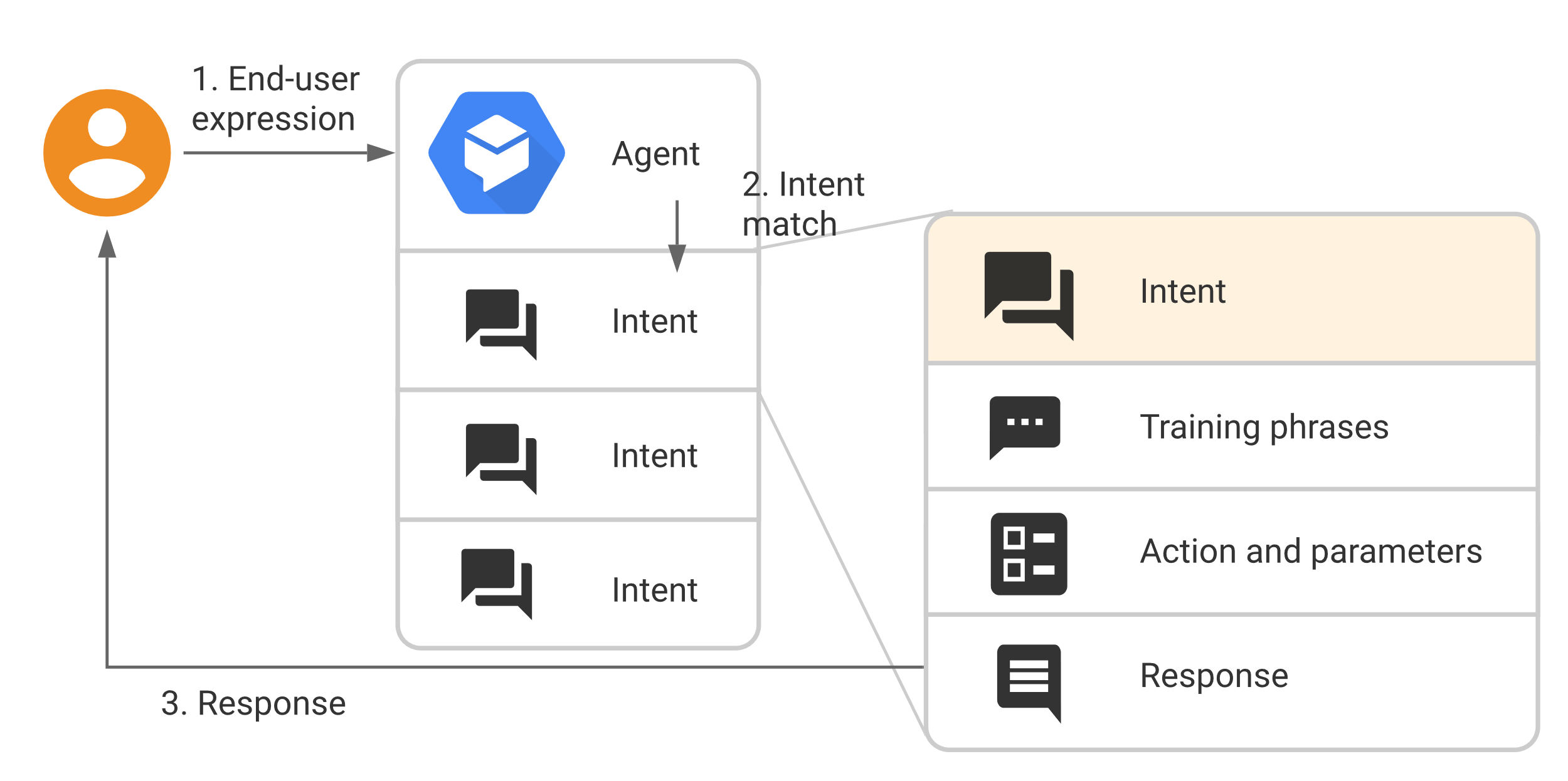
A basic intent contains the following:

* **Training Phrases**: Training phrases are example phrases for what end-users might type or say, referred to as end-user expressions. For each intent, you create many training phrases. When an end-user expression resembles one of these phrases, Dialogflow matches the intent.



Here the training phrases are names like “varun” and “surya”. If the end user ends up giving texts like this then this intent comes into play.

* For example, the training phrase "I want pizza" trains your agent to recognize end-user expressions that are similar to that phrase, like "Get a pizza" or "Order pizza".
* **Action**: The action field is a simple convenience field that assists in executing logic in your service. When building an agent, you can set this field to any text you find useful. When an intent is matched at runtime, Dialogflow provides the action value to your fulfillment webhook request or the API interaction response. It can be used to trigger specific logic in your service.
* **Parameters**: When an intent is matched at runtime, Dialogflow provides the extracted values from the end-user expression as parameters. Each parameter has a type, called the entity type which dictates exactly how the data is extracted. Unlike raw end-user input, parameters are structured data that can easily be used to perform some logic or generate responses.
* Examples are shown for the training phrase "book a room on Tuesday", where "Tuesday" is annotated.
* **Responses**: Intents have a built-in response handler that can return responses after the intent is matched. This feature only supports static responses, though you can use parameter references in these responses to make them somewhat dynamic.
* For example, your intent response could look like: "Okay, I booked a room for you on $date".
* **Contexts**: Dialogflow contexts are similar to natural language context. If a person says to you "they are orange", you need context in order to understand what the person is referring to. Similarly, for Dialogflow to handle an end-user expression like that, it needs to be provided with context in order to correctly match an intent.



Entity:

Each intent parameter has a type, called the entity type, which dictates exactly how data from an end-user expression is extracted.

Dialogflow provides predefined system entities that can match many common types of data. For example, there are system entities for matching dates, times, colors, email addresses, and so on. You can also create your own custom entities for matching custom data.

For example, you could define a vegetable entity that can match the types of vegetables available for purchase with a grocery store agent.

Contexts:

Dialogflow contexts are similar to natural language context. If a person says to you "they are orange", you need context in order to understand what "they" is referring to. Similarly, for Dialogflow to handle an end-user expression like that, it needs to be provided with context in order to correctly match an intent.

Using contexts, you can control the flow of a conversation. You can configure contexts for an intent by setting input and output contexts, which are identified by string names. When an intent is matched, any configured output contexts for that intent become active. While any contexts are active, Dialogflow is more likely to match intents that are configured with input contexts that correspond to the currently active contexts.

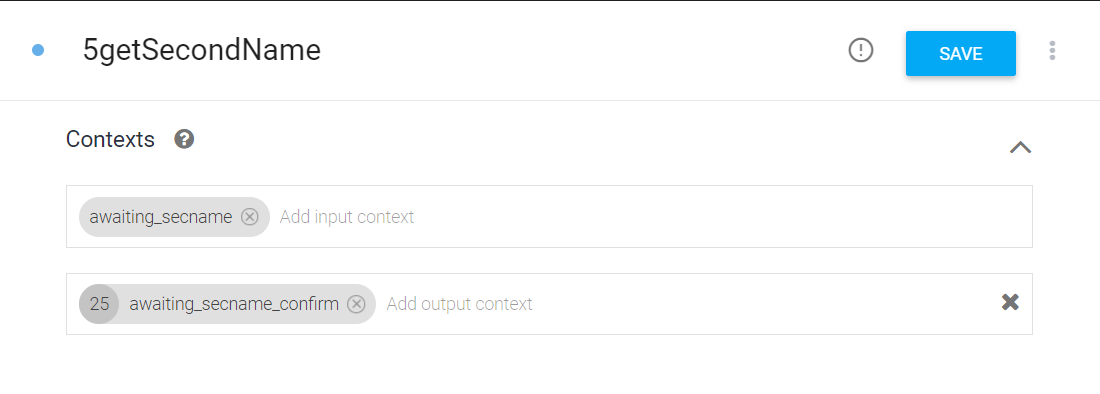
Input Contexts and Output Contexts:

Input and output contexts are applied to intents. They work together to control conversation flow:

* Output Contexts control active contexts. When an intent is matched, any configured output contexts for that intent become active.
* Input Contexts control intent matching. While contexts are active, Dialogflow is more likely to match intents that are configured with input contexts that are a subset of currently active contexts.

With contexts, you can:

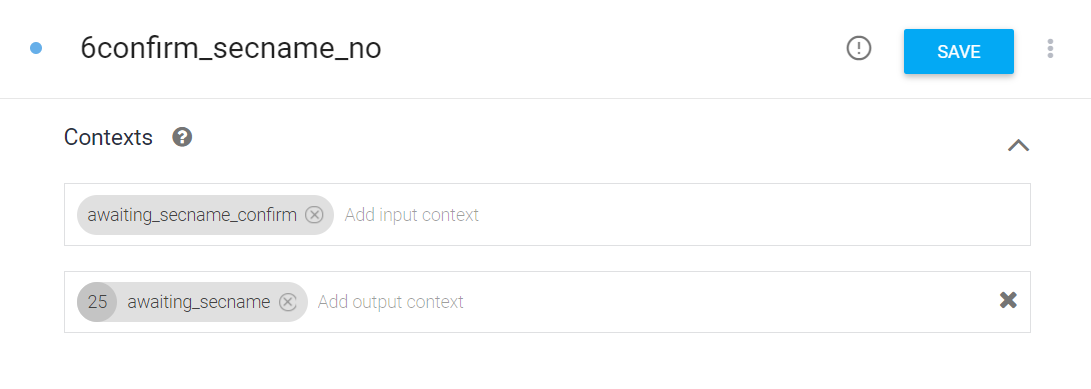
* Control the order of intent matching.
* Create context-specific intents with the same training phrases.



Input Context: awaiting\_secname

Output Context: awaiting\_secname\_confirm

After the getting the secname (input context) we send the name to another intent using (output context)awaiting\_secname\_confirm for confirming whether the second name of the user is correct or not.



Input Context: awaiting\_secname\_confirm

Output Context: awaiting\_lastname

So now awaiting\_secname\_confirm acts as an input context and then it will redirect to the next intent using the output context.

By this way, we can control the flow of the chat between the end user and the bot.

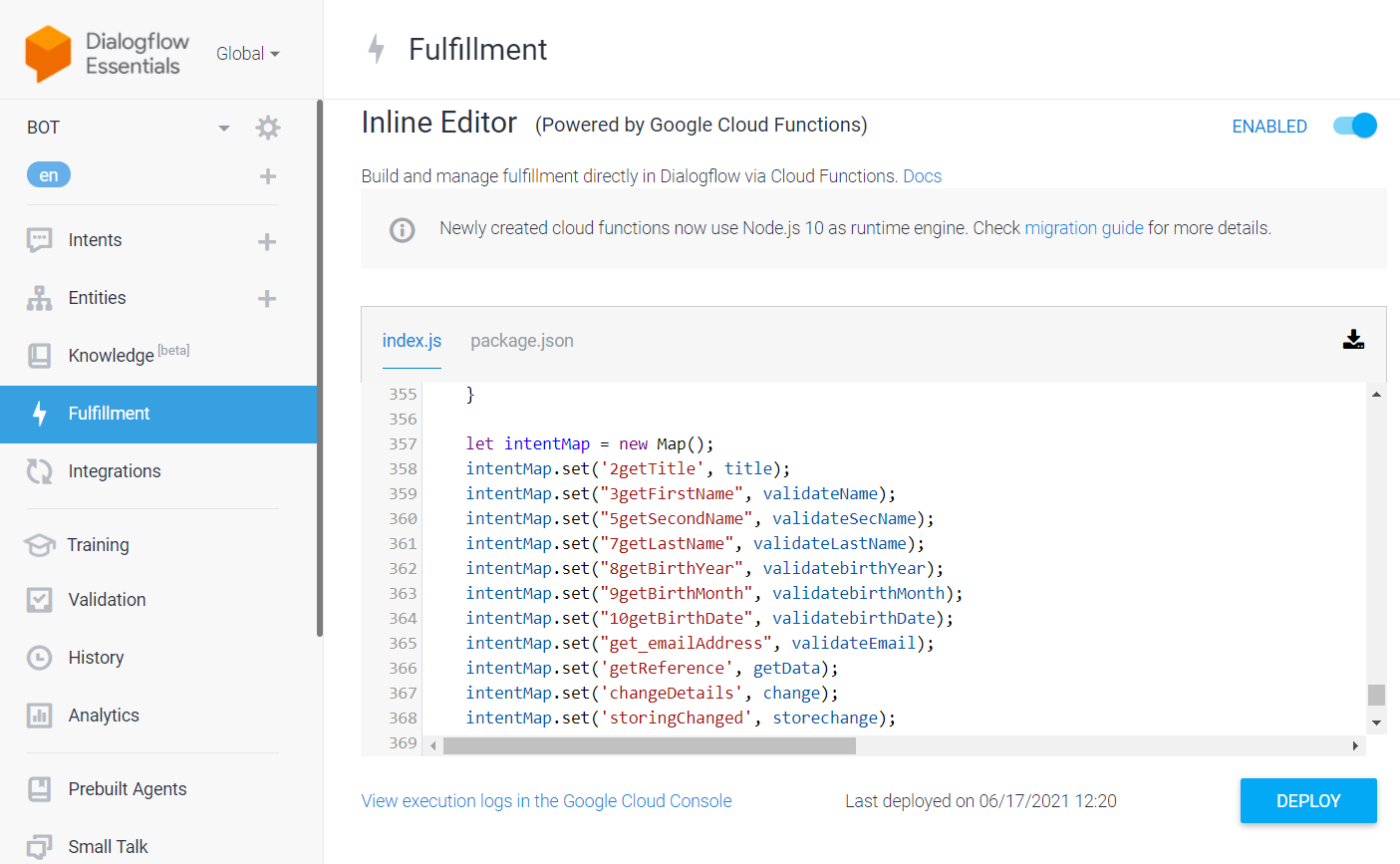
Fulfillment:

By default, your agent responds to a matched intent with a static response. If you're using one of the integration options, you can provide a more dynamic response by using fulfillment. When you enable fulfillment for an intent, Dialogflow responds to that intent by calling a service that you define.

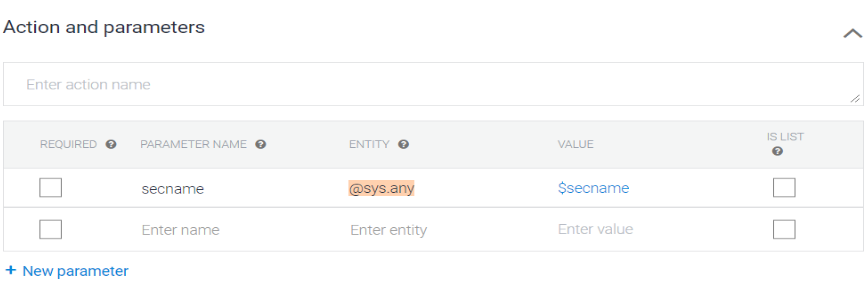
For example, if an end-user wants to schedule a haircut on Friday, your service can check your database and respond to the end-user with availability information for Friday.

Each intent has a setting to enable fulfillment. If an intent requires some action by your system or a dynamic response, you should enable fulfillment for the intent. If an intent without fulfillment enabled is matched, Dialogflow uses the static response you defined for the intent.

Here we are mapping all the intents with the corresponding functions in the fulfilment to handle dynamic responses and each function has its own activity to deal with .

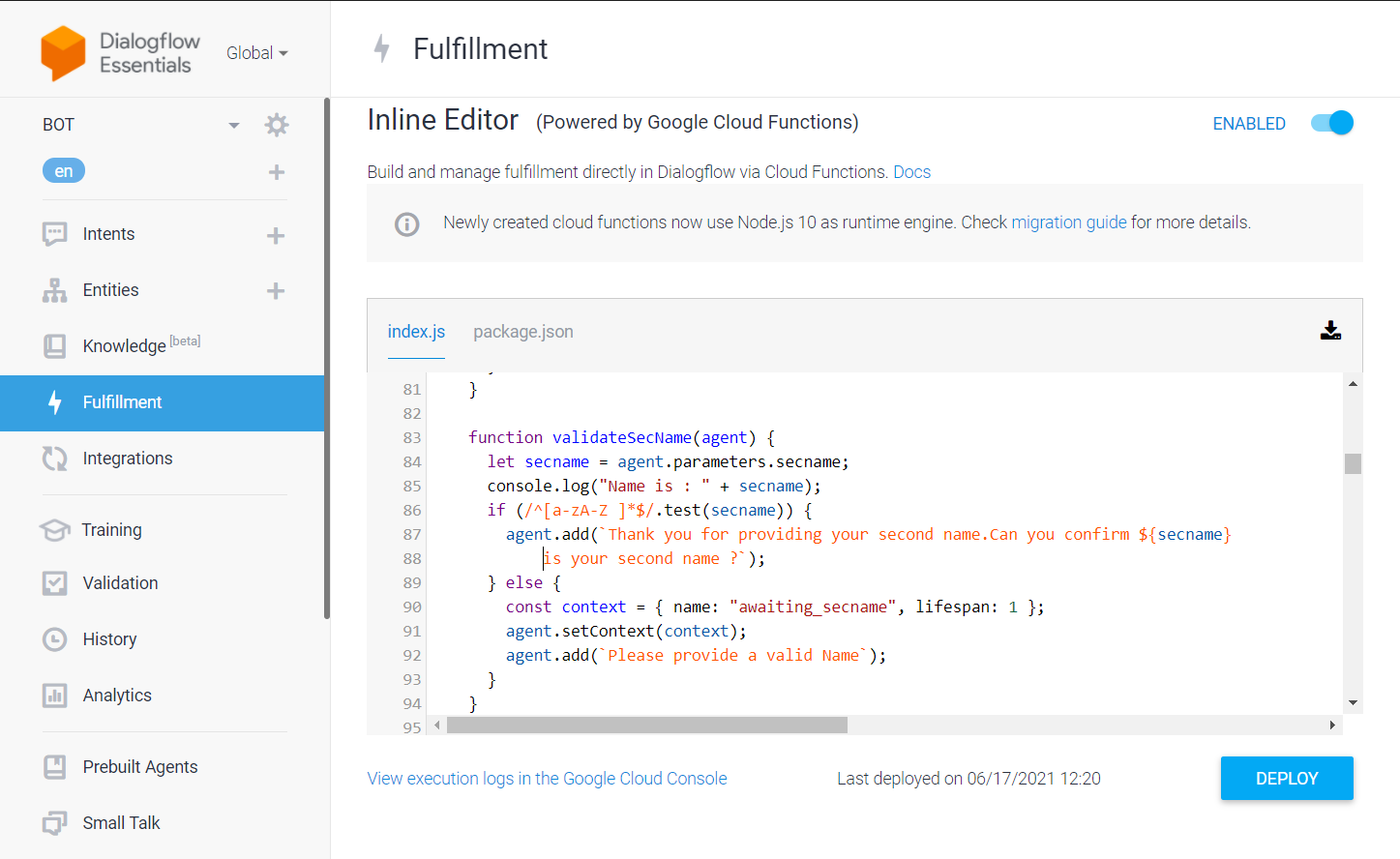


In this intent, the second name provided by the user in the chat is stored in secname parameter.



Here, we are using the second name (secname) using agent.parameters.secname and using it for validation in our fulfilment code.

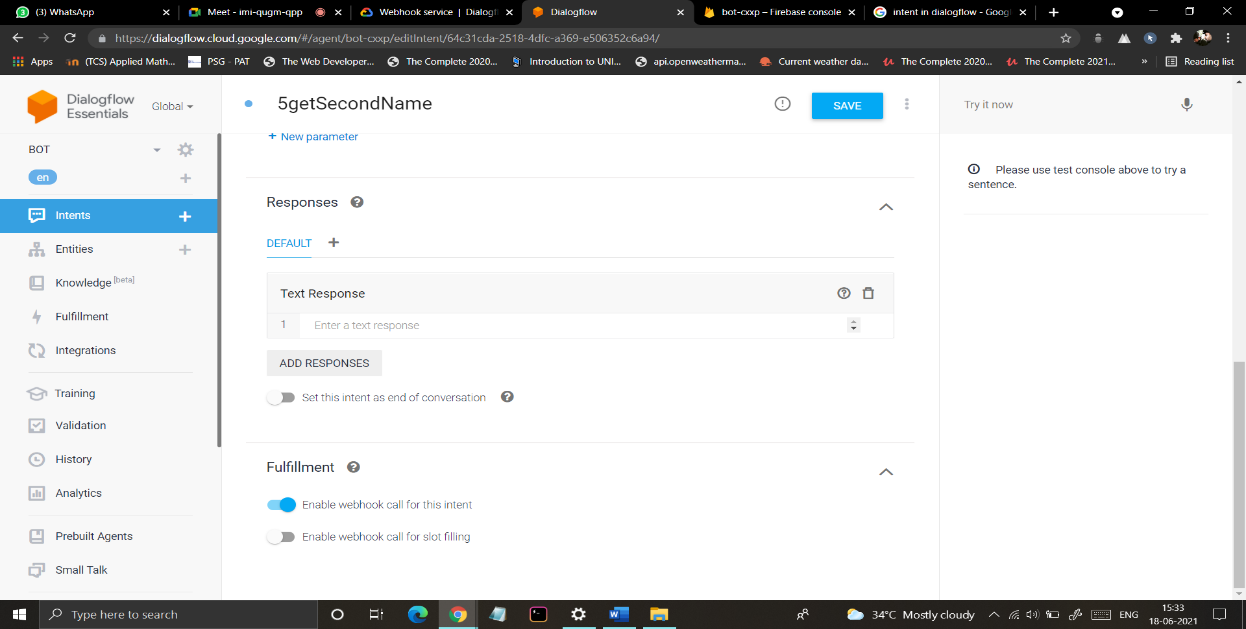
If the name entered is valid then the flow will continue as such entered in the intent else it will go back to the previous context as given below and ask for the second name from the user again saying “Please provide a valid name”.



Inline editor:

The Dialogflow Console has a built-in code editor, called the inline editor that you can use to create fulfillment code and deploy the code to  Cloud Functions has associated charges, but the service is free up to a significant number of monthly invocations.

To get dynamic response from fulfilment we need to enable webhook in the corresponding intent like we have done below:



## To add a new package to the project we have to specify the package name and version in this file(package.json)

## 

## Limitations:

* You cannot save or download code modified in the inline editor without deploying it first.
* The inline editor only supports two files:
  + **index.js**: Contains all of your fulfillment code.
  + **package.json**: Modifying this file will install any dependencies you specify upon deployment.
* The inline editor only supports Node.js.

# Webhook service:

# To use fulfillment in a production system, you should implement and deploy a webhook service. To handle fulfillment, your webhook service needs to accept JSON requests and return JSON responses as specified in this guide. The detailed processing flow for fulfillment and webhooks is described in the fulfilment overview document.