

Google Assistant: Build a Youtube Entertainment App

GSP487



Google Cloud Self-Paced Labs

Overview

[Google Assistant](#) is a personal voice assistant that offers a host of actions and integrations. From sending texts and setting reminders, to ordering coffee and playing music, the 1 million+ actions available suit a wide range of voice command needs.

[Google Cloud Functions](#) is a lightweight compute solution for developers to create single-purpose, stand-alone functions that respond to Cloud events without the need to manage a server or runtime environment.

The [Youtube Data API](#) allows you to access millions of Youtube videos and their associated metadata through HTTP requests.

By utilizing Cloud Functions and the Youtube Data API, you will build an Assistant application that takes in a user's favorite music artist and generates their top ranking songs with links and thumbnails.

What you will learn

In this lab, you will learn how to:

- Build an Assistant application pipeline that consists of an Actions project, a Dialogflow agent with custom intents and entities, a webhook, and a Cloud Function to handle fulfillment.
- Generate the proper authentication credentials and install necessary dependencies to use the Youtube Data API.
- Add fulfillment logic to the Cloud Function to handle Youtube Data API calls.
- Deploy your application and test it with the Actions Simulator.

Prerequisites

This is an **advanced level** lab. This assumes familiarity with Dialogflow and Cloud Functions. Basic knowledge of APIs is recommended. Experience with JavaScript and the Node.js runtime is recommended, but not required. If you need to brush up on these skills, please take one of the following labs before attempting this one:

- [Google Assistant: Build an Application with Dialogflow and Cloud Functions](#)
- [Introduction to APIs in Google](#)

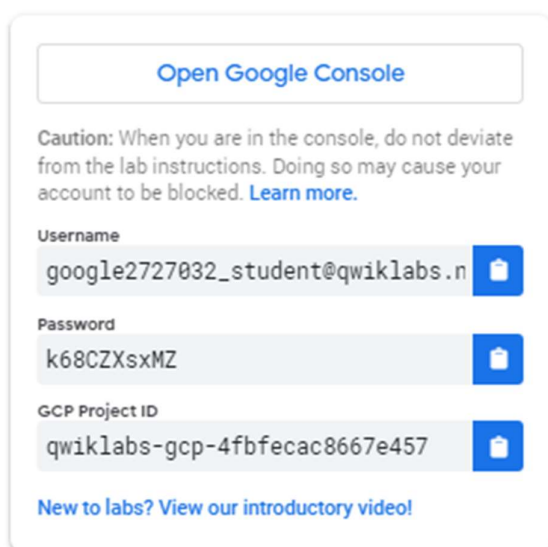
Once you're ready, scroll down and follow the steps below to get your lab environment set up.

Setup

Google Cloud Platform Console


How to start your lab and sign in to the Google Cloud Console


1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.




[Open Google Console](#)

Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. [Learn more.](#)

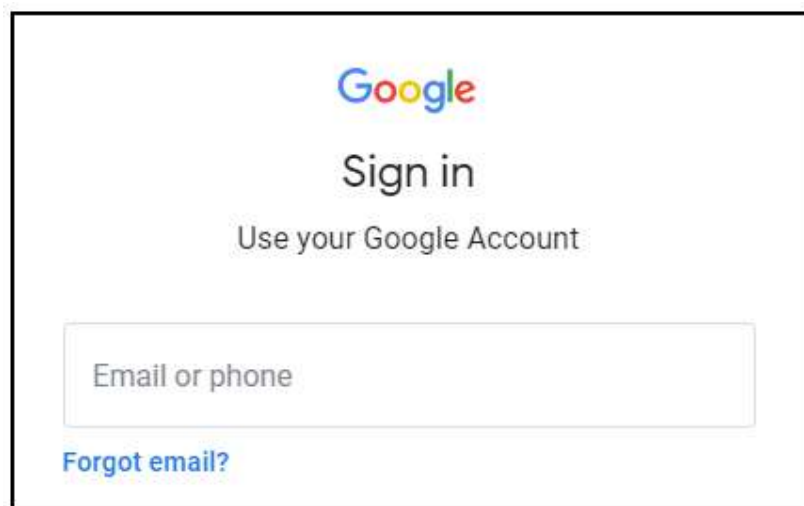
Username
google2727032_student@qwiklabs.n 

Password
k68CZXsxMZ 

GCP Project ID
qwiklabs-gcp-4fbfecac8667e457 

[New to labs? View our introductory video!](#)

2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



Google

Sign in

Use your Google Account

Email or phone

[Forgot email?](#)

Tip: Open the tabs in separate windows, side-by-side.

If you see the **Choose an account** page, click **Use Another**



Account.

3. In the **Sign in** page, paste the username that you copied from the Connection Details panel. Then copy and paste the password.

Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

4. Click through the subsequent pages:

- Accept the terms and conditions.
- Do not add recovery options or two-factor authentication (because this is a temporary account).
- Do not sign up for free trials.

After a few moments, the Cloud Console opens in this tab.

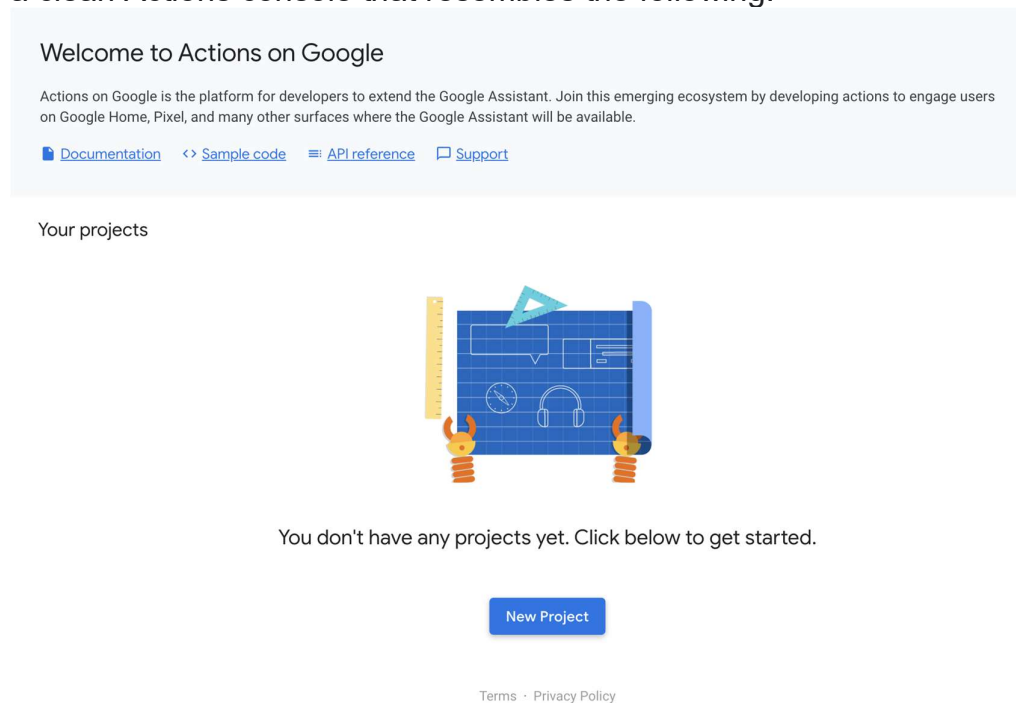
Note: You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.



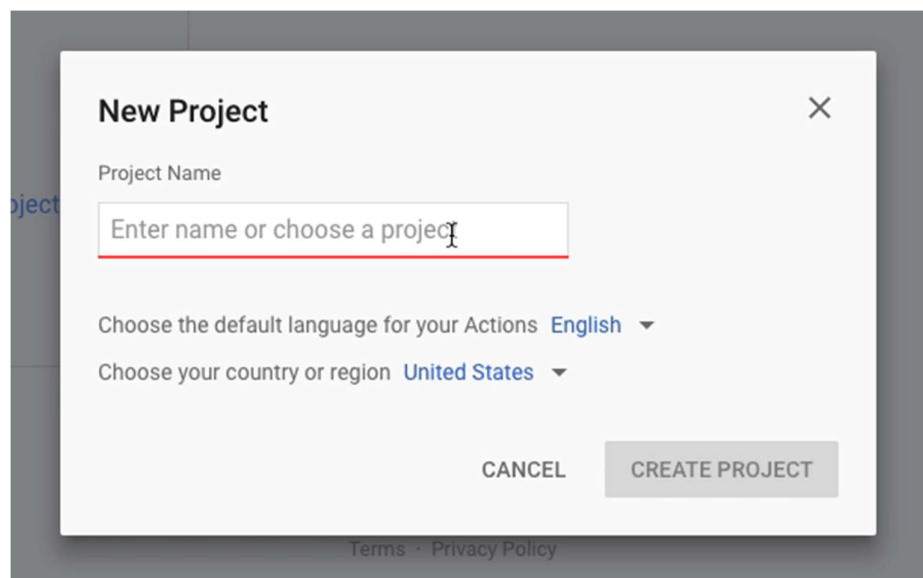
Create an Actions project

In order to build any Assistant application, you will first have to make an Actions project.

Open a new tab in your browser and go to the [Actions on Google Developer Console](#). Then sign in with your Qwiklabs credentials if prompted. Also, agree to Actions on Google's terms of service when prompted. Once you're signed in, you should be looking at a clean Actions console that resembles the following:



Click **New project**. Then click into the project field and select your Qwiklabs GCP project ID. Then click **IMPORT PROJECT**:



Soon after you will be presented with a welcome page that resembles the following:

What kind of Action do you want to build?

Select the category that best fits the type of experience you want to build for the Google Assistant.

Smart Home

Let users control your smart home devices with the Google Assistant and the Google Home app



Food ordering

Integrate your food ordering flow with Google Search, Maps, and the Assistant



Game

Build anything from a trivia game to a fully immersive, multiplayer gaming experience



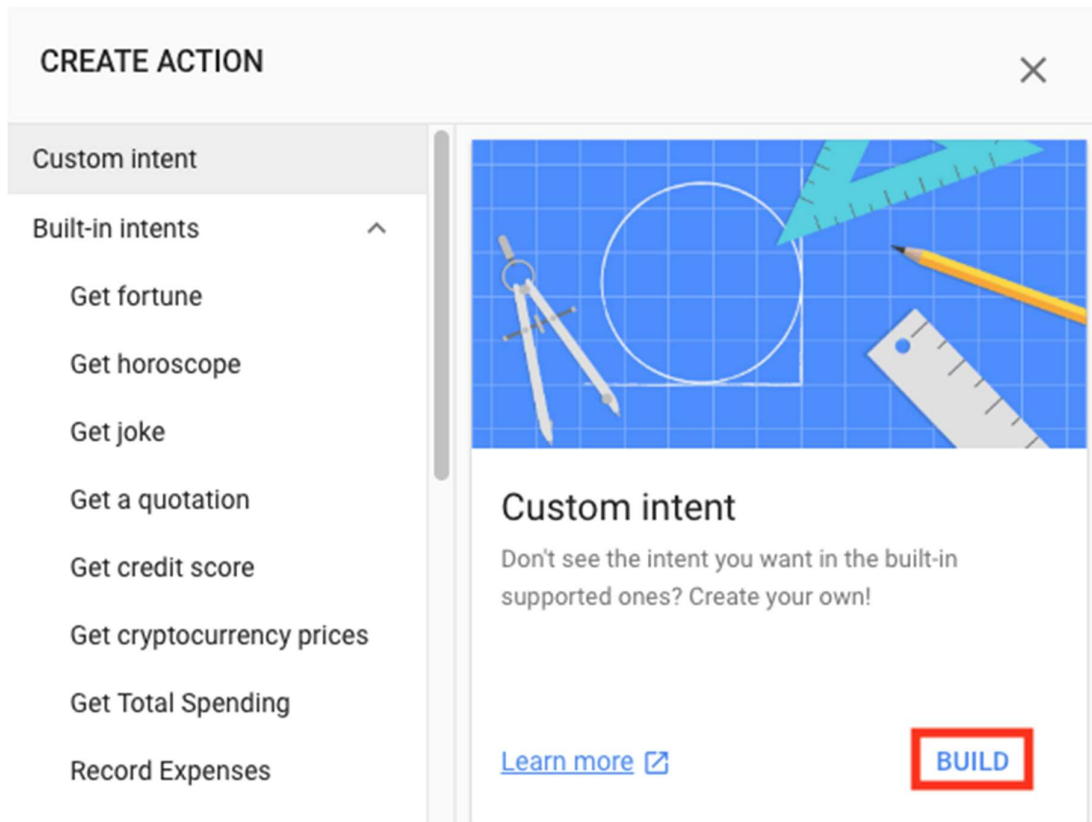
Custom

Don't see your category? Build a unique conversational experience for your users



Build an Action

In the left-hand corner, click the `Actions Console` logo and then select your newly created project (title has your GCP Project ID as the name). Then click the **Build your Action** dropdown and then select **Add Action(s)**. Then click **Get Started > Custom Intent > BUILD**:



This will take you to the Dialogflow console.

Select your Qwiklabs account and click **Allow** when Dialogflow prompts you for permission to access your Google Account.

When you land on the Dialogflow page, check the box next to **Yes, I have read and accept the agreement** and click **Accept**.

If you are brought to the following Dialogflow agent creation page, click **CREATE**:

qwiklabs-gcp-8aaed5810f687c99

CREATE

DEFAULT LANGUAGE ?

English — en

Primary language for your agent. Other languages can be added later.

DEFAULT TIME ZONE

(GMT-8:00) America/Los_Angeles

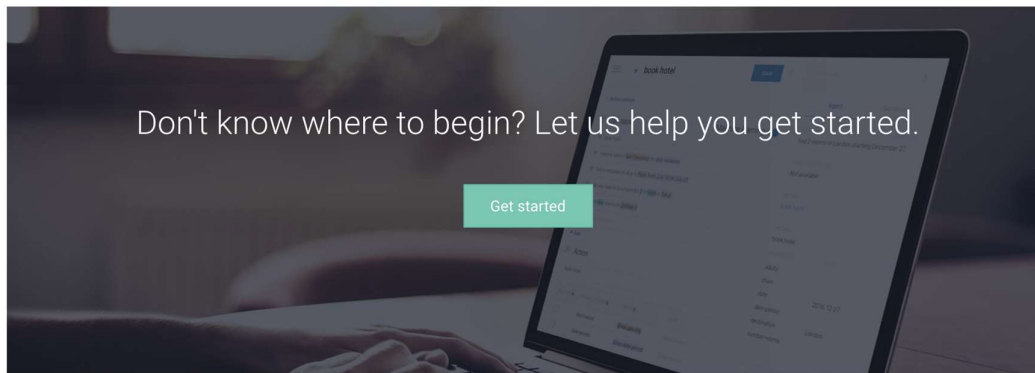
Date and time requests are resolved using this timezone.

GOOGLE PROJECT

Agent will be linked with [qwiklabs-gcp-8aaed5810f687c99](#) Google Project

If you are brought to this page instead:

 Welcome to Dialogflow!



Now it's time to create your first agent.

CREATE AGENT

Close the Dialogflow agent creation tab. You will return to the Actions Console.

Click **Get Started** > **Custom Intent** > **BUILD**.

Select your Qwiklabs account and click **Allow** when Dialogflow prompts you for permission to access your Google Account.

Now click **CREATE**:

qwiklabs-gcp-8aaed5810f687c99

CREATE

DEFAULT LANGUAGE ?

English — en

Primary language for your agent. Other languages can be added later.

DEFAULT TIME ZONE

(GMT-8:00) America/Los_Angeles

Date and time requests are resolved using this timezone.

GOOGLE PROJECT

Agent will be linked with [qwiklabs-gcp-8aaed5810f687c99](#) Google Project

Test Completed Task

Click **Check my progress** to verify your performed task.

You will now enable webhook fulfillment in Dialogflow so that your Cloud Function can be passed the necessary arguments to create this application. Click **Fulfillment** from the left-hand menu.

Move the **Disabled** slider for Webhook to the right to **Enabled**.

Enter the dummy URL <https://google.com> in for the URL field. You will update this URL when you build and deploy your fulfillment:

⚡ Fulfillment

Webhook

ENABLED 

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.

URL*

<https://google.com>

Scroll down and click **Save** in the bottom right corner. Then click **Intents** from the left-hand menu and select **Default Welcome Intent**:



Intents

Search intents



Default Fallback Intent



Default Welcome Intent

You will now take a look a high level look at the user experience you're trying to build.

Conversational design

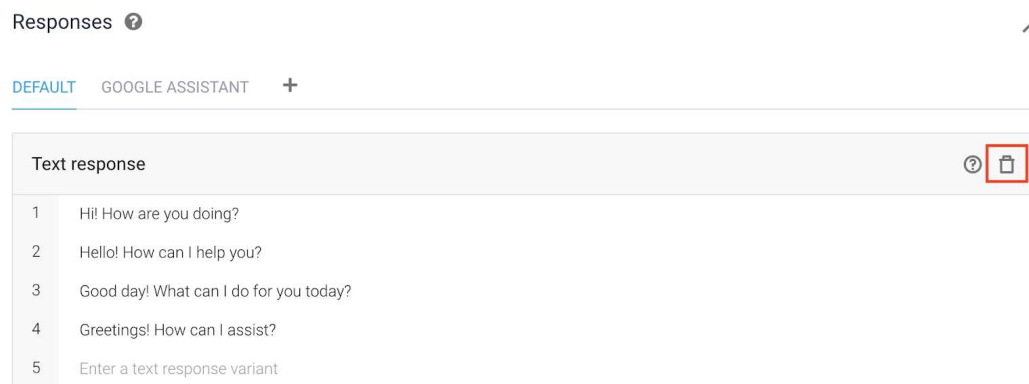
Before you begin creating an Assistant application, it's always a good idea to map out a conversation by writing sample dialog(s). These dialogs give you a good reference point when you eventually start building your app. Here is a sample dialog for the application you're building:

- **App:** *Hello there and welcome to Youtube Entertainment! What artist are you searching for?*
- **User:** *Kanye West*
- **App:** *Fetching your request... Okay, the top video from Kanye West is "I Love It" featuring Lil Pump. Here is a link to the video. See you next time!*

Conversational design tips: This example shows only a couple of the dialogs that you could write for your own apps. See this [Design Walkthrough](#) for more examples of dialogs.

Build the Default Welcome Intent

Make sure that you are in the Default Welcome Intent. Scroll down to the **Responses** section and click the **trash icon** to scrap all of the default text responses:



Now click on **ADD RESPONSES > Text response**, and type in the following: *Hello there and welcome to Youtube Entertainment! What artist are you searching for?*

Text response		?	
1	Hello there and welcome to Youtube Entertainment! What artist are you searching for?		
2	Enter a text response variant		

ADD RESPONSES

☐ Set this intent as end of conversation ?

Now scroll up and click **Save** in the top-right corner. Now, when users invoke your app, they know that they're entering your app's experience and what to say next.

Test Completed Task

Click **Check my progress** to verify your performed task.

In general, the app's responses should guide users on what to say next and to stay within your conversations' grammar, which will be further defined in the following section.

Build a Custom Intent

Dialogflow intents define your conversation's grammar and what tasks to carry out when users speak specific phrases. You will now create an intent that parses user input for fields that are required by the Youtube Data API.

From the left-hand menu, click on the **+** icon by the **Intents** menu item. For the **Intent name** field, enter `youtube`.

In the **Training phrases** field click **Add Training Phrases**. Then in the Add user expression field enter in `Kanye West` and hit **Enter**.

Select `Kanye West` and assign it the `@sys.any` entity:

Training phrases ⓘ Search training phrases 🔍 ^

” Kanye West

” Kanye West

PARAMETER NAME	ENTITY	RESOLVED VALUE
any	@sys.any	Kanye West

If you are getting prompts which says Try to avoid using @sys.any then click **OK**. You will now add some more user expressions in the **Training phrases** section, and assign any of those phrases the @sys.any entity. Add the following training phrases by entering them in the user expression field:

- *Dua Lipa*
- *Lil Baby*

Your Training phrases section should now resemble the following—note the highlighted fields, which have been assigned the @sys.any entity:

Training phrases ⓘ Search training phrases 🔍 ^

” Add user expression

” Lil Baby

PARAMETER NAME	ENTITY	RESOLVED VALUE
any	@sys.any	Lil Baby

” Dua Lipa

” Kanye West

Now that your training phrases have been added, scroll down and expand the **Action and parameters > MANAGE PARAMETERS AND ACTION** section.

Now check the **REQUIRED** checkbox for the any parameter. This tells Dialogflow to not trigger the intent until the parameter is properly provided by the user.

Now click on **Define prompts** for the number parameter (right-hand side) and provide a re-prompt phrase. Enter in who do you want to look up? in the prompt field and then click **Close**:

Prompts for "any"

NAME	ENTITY	VALUE
any	@sys.any	\$any

PROMPTS

1

Who do you want to look up?

2

Enter a prompt variant

CLOSE

Now scroll to the top of the page and click **Save**.

Now scroll down to the **Fulfillment** section. Click the dropdown arrow and click **Enable fulfillment**. Then click the **Enable webhook call for this intent** slider:

Fulfillment ?

☒ Enable webhook call for this intent

☐ Enable webhook call for slot filling

This tells Dialogflow to call your fulfillment to generate a response to the user instead of using Dialogflow's response feature.

In the **Responses** section right above fulfillment, click on the `Google Assistant` tab. If you are unable to see `Google Assistant` tab then click on the add (+) button and select `Google Assistant`.

Move the toggle for **Set this intent as end of conversation**. This tells Dialogflow to relinquish control back to the Google Assistant after your fulfillment returns a response to the user.



Response from this tab will be sent to the Google Assistant integration.

Use response from the DEFAULT tab as the first response.



ADD RESPONSES



Set this intent as end of conversation ?

Then scroll to the top of the intent and click **SAVE** once more to save the entire intent.

You have now successfully declared your conversation's grammar with Dialogflow intents. You have also used Dialogflow's built-in response feature to return a static response to the user when they invoke your app. You also created an intent that uses fulfillment to return a response, which you will build with Cloud Functions.

Test Completed Task

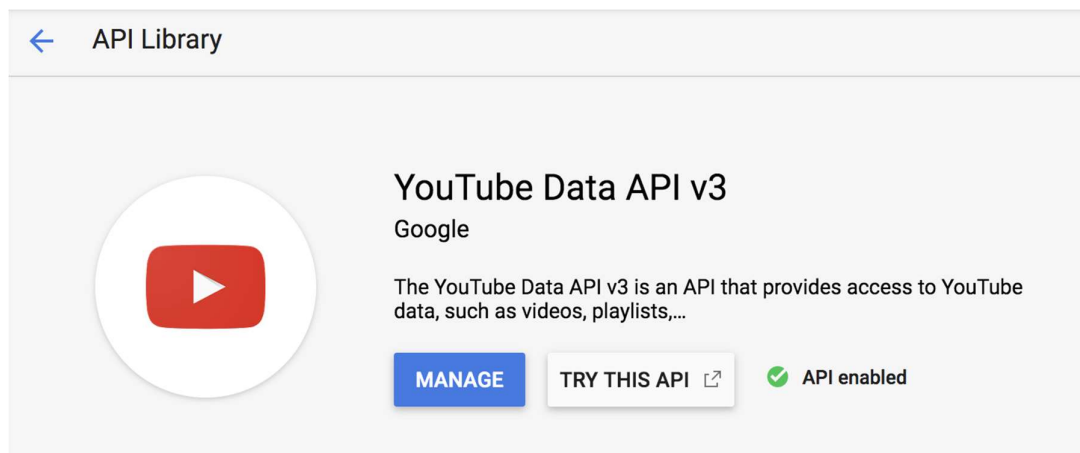
Click **Check my progress** to verify your performed task.

Enable the API

Return to the Google Cloud Console for this step. To make Youtube Data API requests from your Cloud Function, you will need to enable the Youtube Data API in your GCP Project.

Open the Navigation menu and select **APIs & Services > Library**. Then in the search bar, enter **Youtube Data API v3** and select it from the results page.

Then click **Enable**. Click the back button in your browser **twice** and ensure that your page resembles the following with the Youtube Data API enabled:



Get an API Key

To make Youtube Data API requests from your Cloud Function, you will need an API key to properly authenticate calls. Return to the Google Cloud Console and open the Navigation menu.

From there, select **APIs & Services > Credentials**. Then click on the **Create credentials** dropdown and select **API Key**. You should be presented with the following panel:

API key created

Use this key in your application by passing it with the `key=API_KEY` parameter.

Your API key

AIzaSyCZTwF384MtVaRwTqBtKeZJJCUXHiYWh5I



 Restrict your key to prevent unauthorized use in production.

[CLOSE](#) [RESTRICT KEY](#)

Copy the API key and save it somewhere for later use, like a text editor or notepad. Then click **Close**.

Initialize and Configure a Cloud Function

You will now build a Cloud Function to handle your fulfillment logic.

Open the `Navigation menu` and select **Cloud Functions**, which is located under the compute header. Then click **Create function**.

This will open a template to create a new Cloud Function. Your page will resemble the following:

Cloud Functions

Create function

1 Configuration

2 Code

Basics

Function name *

function-1

Region

us-central1

Trigger

HTTP

Trigger type

HTTP

URL

https://us-central1-qwiklabs-gcp-01-75634ed11d03.cloudfunctions.net/function-1

Authentication

☐ Allow unauthenticated invocations

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

☒ Require authentication

Manage authorized users with Cloud IAM.

SAVE

CANCEL

VARIABLES, NETWORKING AND ADVANCED SETTINGS


NEXT

CANCEL

For the Cloud Function's **Function name** field, enter in `youtube`.

Then scroll down to the authentication section and select **Allow unauthenticated invocations**:


Trigger

 **HTTP**

Trigger type

HTTP

URL



https://us-central1-qwiklabs-gcp-01-fbc8ec0a5d2c.cloudfunctions.net/youtube

Authentication

☒ Allow unauthenticated invocations

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

☐ Require authentication

Manage authorized users with Cloud IAM.

SAVE

CANCEL

Forgetting to do the above will cause your simulation test to fail at the end!

Now click on the **Save** button and then click on **Next** to find the `inline` editor for `index.js` and `package.json`. Make sure that the `index.js` tab is open. This file defines your fulfillment logic and is used to create and deploy a Cloud Function.

Remove the boilerplate code from the file. Then **copy and paste** the following code into `index.js`:

```

'use strict';

const {
  dialogflow,
  Image,
  Suggestions
} = require('actions-on-google');

const functions = require('firebase-functions');
const app = dialogflow({debug: true});

const API_KEY = '<YOUR_API_KEY_HERE>';

app.intent('youtube', (conv, {any}) => {
  var url =
    "https://www.googleapis.com/youtube/v3/search?part=snippet&maxResults=5&q=" +
    encodeURIComponent(any) + "&type=video&order=viewCount&videoCategoryId=10&key=" +
    API_KEY;
  const axios = require('axios');
  return axios.get(url)
    .then(response => {
      var output = JSON.stringify(response.data);
      var song_fields = response.data.items[1];
      return song_fields;
    }).then(output => {
      var song_title = output.snippet.title;
      song_title = song_title.replace(/&/g, '&');
      song_title = song_title.replace(/"/g, '"');
      var song_link = JSON.stringify(output.id.videoId);
      var song_thumbnail = output.snippet.thumbnails.high.url;
      conv.ask(`Fetching your request...`)
      conv.ask(new Image({
        url: song_thumbnail,
        alt: 'Song thumbnail'
      }))
      conv.close(`The most popular song is: ` + song_title + `. The link to this
      song is: https://www.youtube.com/watch?v=` + song_link.slice(1, -1) + `. See you next
      time.`);
    })
});

exports.youtube = functions.https.onRequest(app);

```

Replace <YOUR_API_KEY_HERE> (line 12) with the API key you generated in the previous step.

Now open the `package.json` tab. This file declares package dependencies for your fulfillment, including the Actions client library. Replace the contents of the file with the following:

```
{
  "name": "youtube",
  "description": "Get restaurant reviews.",
  "version": "0.0.1",
  "author": "Google Inc.",
  "engines": {
    "node": "8"
  },
  "dependencies": {
    "actions-on-google": "^2.0.0",
    "firebase-admin": "^4.2.1",
    "firebase-functions": "1.0.0",
    "axios": "0.16.2"
  }
}
```

Once you have those files configured, find the **Entry point** field. Enter in `youtube` for the value.

Now click the **DEPLOY** button below. It will take about a minute for your function to be built. When the creation completes, your overview page will resemble the following:

Filter functions Columns

<input type="checkbox"/>	Name ^	Region	Trigger	Runtime	Memory allocated	Executed function	Last deployed
<input checked="" type="checkbox"/>	youtube	us-central1	HTTP	Node.js 6	256 MB	youtube	3/26/19, 3:48 PM

Now click on the `youtube` function to get more details about it's configuration. Then click on the **trigger** tab. You will see a function URL that resembles the following:

youtube
Version 1, deployed at Nov 10, 2020, 11:51:53 ...

METRICS
 DETAILS
 SOURCE
 VARIABLES
 TRIGGER
 PERMISSIONS
 LOGS
 TESTING

HTTP

Trigger URL

<https://us-central1-qwiklabs-gcp-01-75634ed11d03.cloudfunctions.net/youtube>

Copy the function URL. You will use it as the URL for the Dialogflow webhook, which is configured in the next section.

Test Completed Task

Click **Check my progress** to verify your performed task.

Configure the Webhook

Return to the Dialogflow console and click on the **Fulfillment** menu item from the left hand navigation menu.


In the URL field, replace `https://google.com` with the function URL you generated in the previous step.

Your webhook should now resemble the following:

Webhook

ENABLED 

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.

URL*	<u>https://us-central1-qwiklabs-gcp-bd85d44900ddc480.cloudfunctions.net/youtube</u>	
BASIC AUTH	<input type="text" value="Enter username"/>	<input type="text" value="Enter password"/>
HEADERS	<input type="text" value="Enter key"/>	<input type="text" value="Enter value"/>
	<input type="text" value="Enter key"/>	<input type="text" value="Enter value"/>
	 Add header	

Scroll down to the bottom of the page and click **Save** in the lower right corner.

Test your Assistant Application with the Actions Simulator

Now that your Cloud Function has been deployed and your webhook has been properly set up, you can preview the app in the Actions simulator.

Check your Google Permission Settings

In order to test the silly name maker that you built over the course of this lab, you need to enable the necessary permissions.

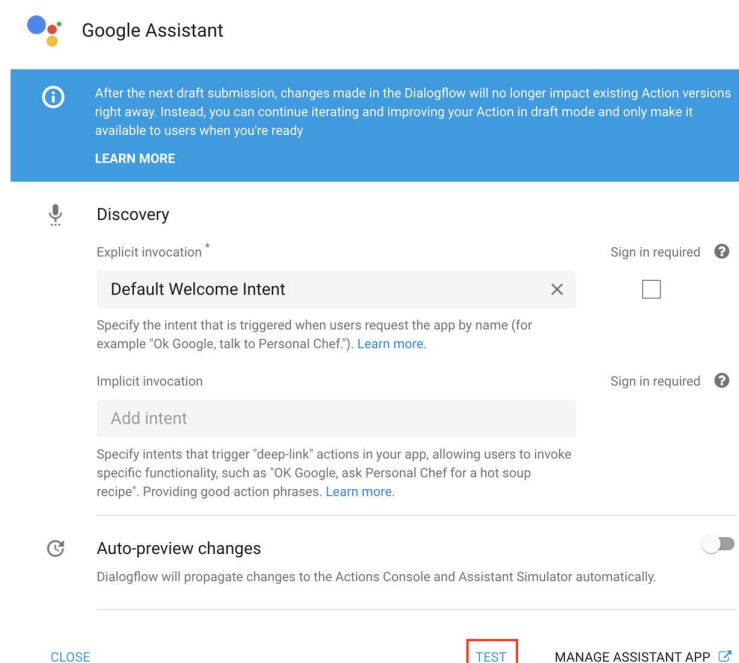
Open a new tab in your browser and visit the [Activity Controls page](#). Sign in with your Qwiklabs credentials if prompted.

Ensure that the following permissions are enabled by sliding the toggles and selecting **TURN ON** for the following cards:

- Web & App Activity
Now **close** the Activity Controls page.

Test the application with the simulator

Return to the Dialogflow console. Then from the left-hand menu, select **Integrations**. Then click **Integration Settings**. When you land on the following page, click **TEST**:



The screenshot shows the 'Google Assistant' integration settings page. At the top, there is a blue notification banner with an information icon and text: 'After the next draft submission, changes made in the Dialogflow will no longer impact existing Action versions right away. Instead, you can continue iterating and improving your Action in draft mode and only make it available to users when you're ready'. Below this is a 'Discovery' section with a microphone icon. It contains two parts: 'Explicit invocation *' and 'Implicit invocation'. Each part has a text input field, a 'Sign in required' toggle, and a 'Learn more' link. The 'Explicit invocation' field contains 'Default Welcome Intent'. The 'Implicit invocation' field contains 'Add intent'. Below these is an 'Auto-preview changes' section with a clock icon and a toggle switch. At the bottom, there are three buttons: 'CLOSE', 'TEST' (highlighted with a red box), and 'MANAGE ASSISTANT APP' with an external link icon.

Google Assistant

After the next draft submission, changes made in the Dialogflow will no longer impact existing Action versions right away. Instead, you can continue iterating and improving your Action in draft mode and only make it available to users when you're ready. [LEARN MORE](#)

Discovery

Explicit invocation * Sign in required ?

Default Welcome Intent X ☐

Specify the intent that is triggered when users request the app by name (for example "Ok Google, talk to Personal Chef."). [Learn more.](#)

Implicit invocation Sign in required ?

Add intent

Specify intents that trigger "deep-link" actions in your app, allowing users to invoke specific functionality, such as "OK Google, ask Personal Chef for a hot soup recipe". Providing good action phrases. [Learn more.](#)

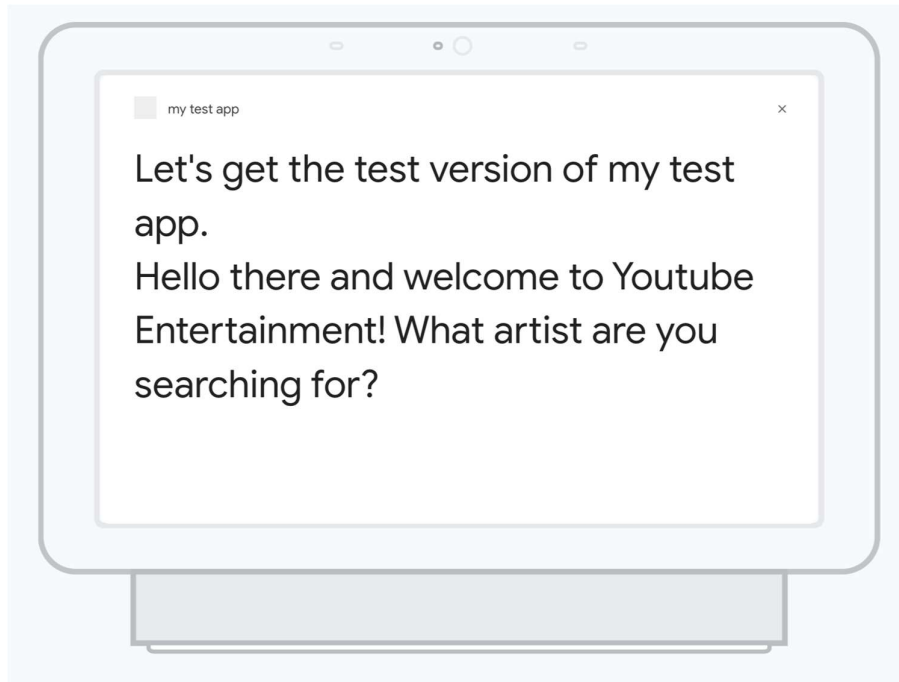
Auto-preview changes ☐

Dialogflow will propagate changes to the Actions Console and Assistant Simulator automatically.



[CLOSE](#) [TEST](#) [MANAGE ASSISTANT APP](#)


The Actions simulator allows you to test your applications without any hardware like a Google Home.

Enter `Talk to my test app` in the **Input** area. You should receive a similar response:





Now enter a music artist (for example, Kanye West.) You should receive a similar output in the right-hand panel, which includes the song name, link, and thumbnail:


Kanye West  

 Fetching your request...
The most popular song is: Kanye West & Lil Pump ft. Adele Givens - "I Love It" (Official Music Video). The link to this song is: <https://www.youtube.com/watch?v=cwQgjq0mCdE>. See you next time.

'my test app' left the conversation

Try talking with your test app with new artists!

Frank  

 Fetching your request...
The most popular song is: Frank Sinatra, My Way, With Lyrics. The link to this song is: <https://www.youtube.com/watch?v=6E2hYDIFDIU>. See you next time.

'my test app' left the conversation

Congratulations!

In this lab you built a robust Google Assistant application with Dialogflow and Cloud Function. After creating an Actions project and Action, you configured and built two intents. From there you learned how to initialize Cloud Function and you added your own fulfillment logic and packages to the Cloud Function to handle Youtube Data API requests. After deploying the Cloud Function, you updated your webhook URL and tested your Assistant application with the Actions simulator. You are now ready to take more advanced Google Assistant development labs.

Finish Your Quest



Continue your Quest with [OK Google: Build Interactive Apps with Google Assistant](#). A Quest is a series of related labs that form a learning path. Completing this Quest earns you the badge above, to recognize your achievement. You can make your badge (or badges) public and link to them in your online resume or social media account. Enroll in this Quest and get immediate completion credit if you've taken this lab. [See other available Qwiklabs Quests](#).

Next Steps / Learn More

Be sure to check out the following lab to receive more hands-on practice with Dialogflow:

- [Google Assistant: Build a Restaurant Locator with the Places API](#)

Google Cloud Training & Certification

...helps you make the most of Google Cloud technologies. [Our classes](#) include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. [Certifications](#) help you validate and prove your skill and expertise in Google Cloud technologies.

Manual Last Updated December 21, 2020

Lab Last Tested December 21, 2020

Copyright 2020 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are associated.