

Platforms for Networked Communities

Development Guide

Boni García

<http://bonigarcia.github.io/>
boni.garcia@uc3m.es

Telematic Engineering Department
School of Engineering

2020/2021

uc3m | Universidad **Carlos III** de Madrid

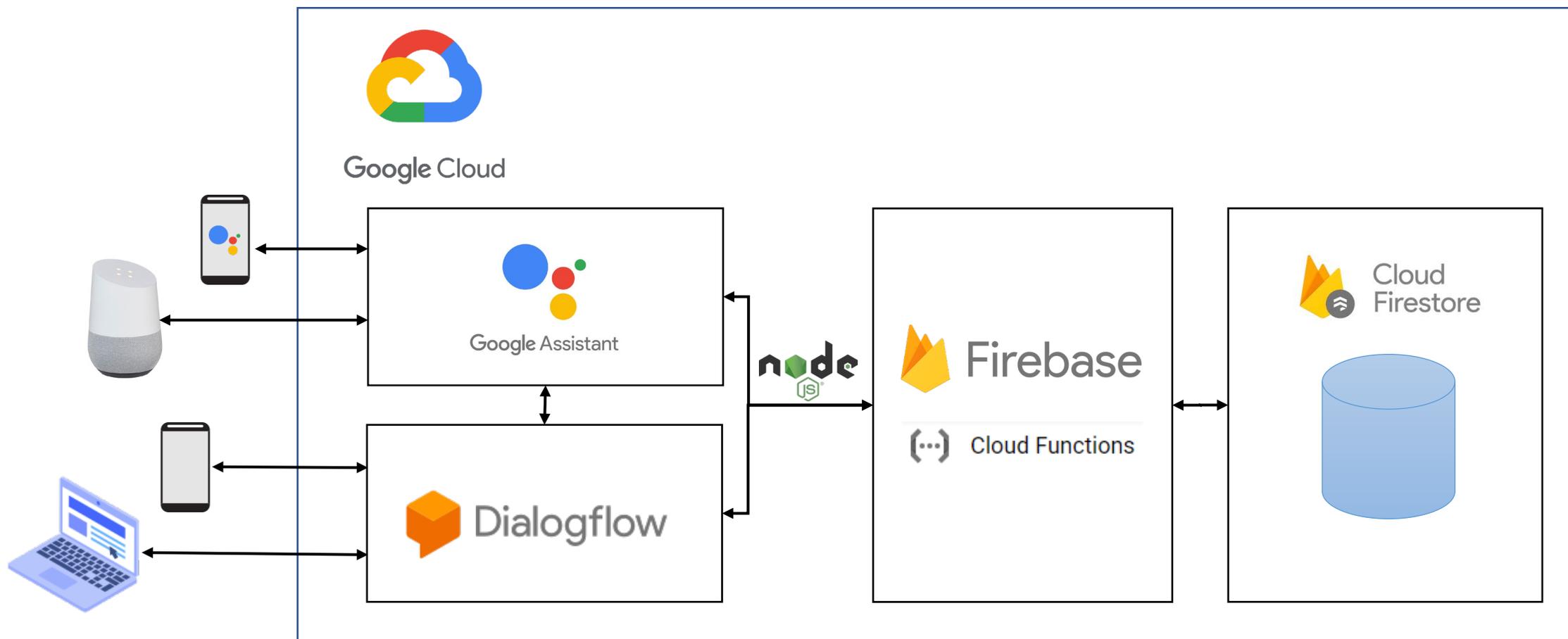


Table of contents

1. Introduction
2. DialogFlow
3. Cloud Source Repositories
4. Cloud Shell
5. Google Cloud SDK
6. Firestore
7. Fulfillment examples
8. Account linking
9. Local deployment

1. Introduction

- The objective of “Platforms for Networked Communities” is to develop **conversational agents** using **Google Cloud** services



1. Introduction

- There are different alternatives to carry out the development of these agents:

1. Using the **inline editor** of DialogFlow
2. Using **Cloud Source Repositories**

- Pro: Very easy to use
- Cons:
 - Not control version
 - Very limited editor

- a) Using cloud services

- **Cloud Shell** (for handling Git and gcloud CLI)
- **Cloud Shell Editor** (for development)

- Pro: Easy to use
- Cons: Limited editor

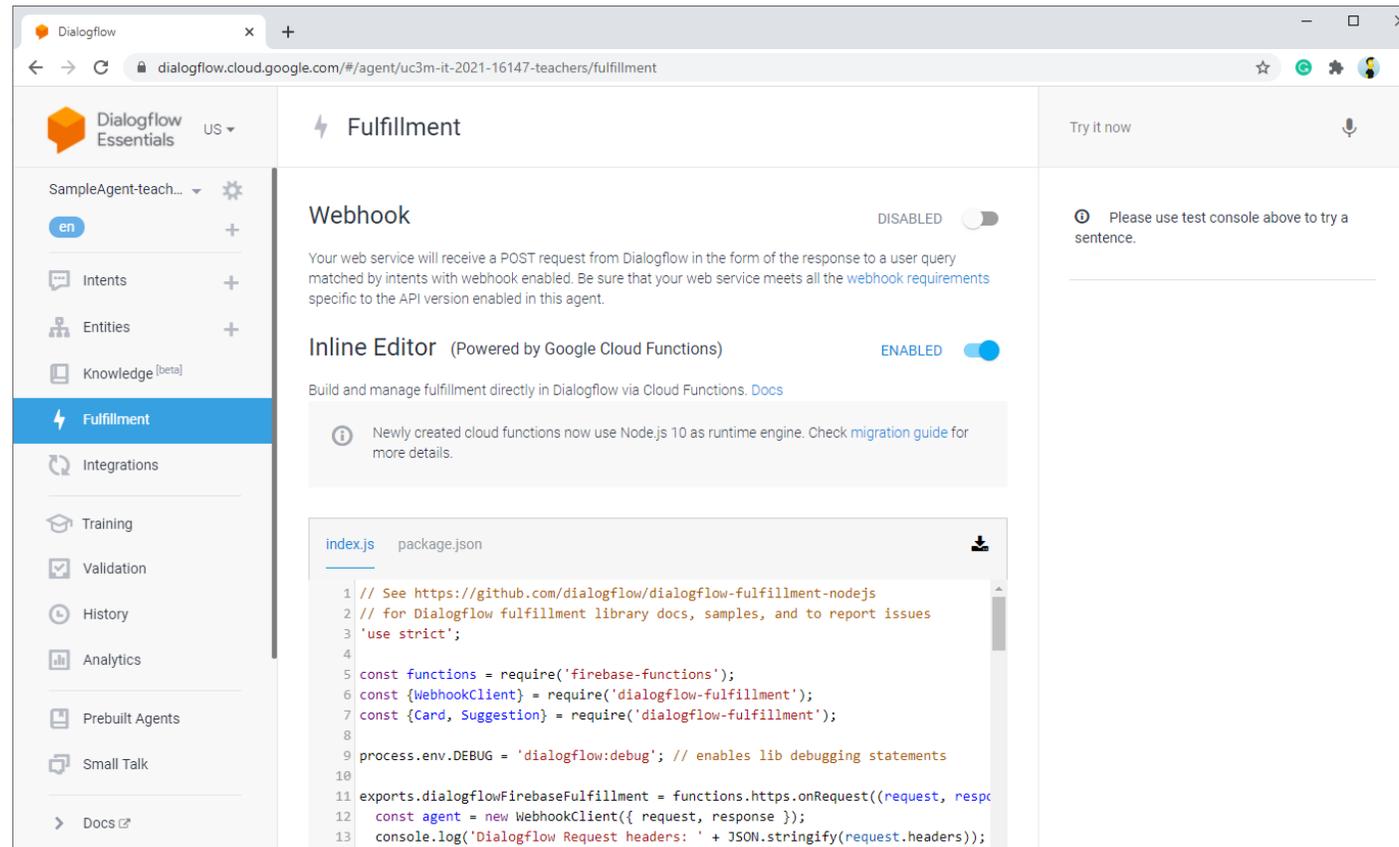
- b) Using **local** environment (our own laptop)

- Install: Node.js, Git, Google Cloud SDK
- Configure: SSH keys
- Development: preferred IDE or text editor

- Pro: Custom environment
- Cons: Configuration required

2. DialogFlow

- We can use the **inline editor** of **DialogFlow** to develop our agent:



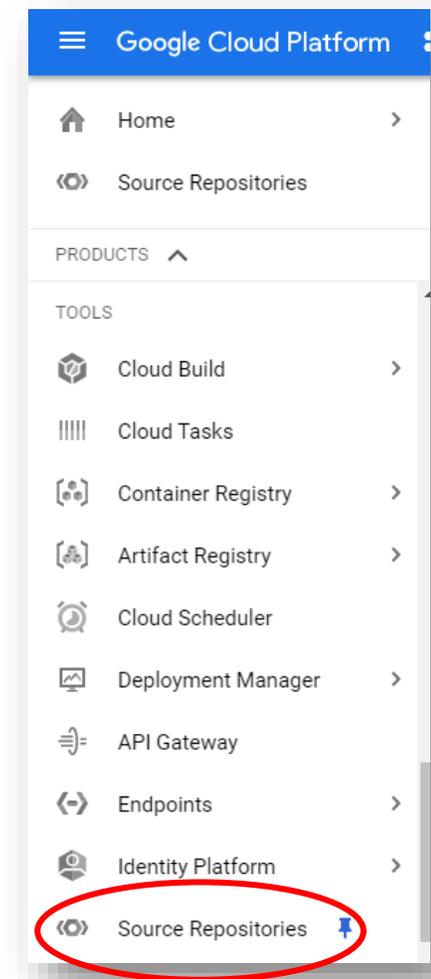
The screenshot shows the DialogFlow Fulfillment inline editor interface. The browser address bar displays the URL: dialogflow.cloud.google.com/#agent/uc3m-it-2021-16147-teachers/fulfillment. The interface includes a left sidebar with navigation options: SampleAgent-teach..., Intents, Entities, Knowledge [beta], Fulfillment (selected), Integrations, Training, Validation, History, Analytics, Prebuilt Agents, and Small Talk. The main content area is titled "Fulfillment" and features a "Webhook" section with a "DISABLED" toggle and an "Inline Editor" section with an "ENABLED" toggle. Below the toggles, there is a code editor showing the following code:

```
index.js package.json
1 // See https://github.com/dialogflow/dialogflow-fulfillment-nodejs
2 // for Dialogflow fulfillment library docs, samples, and to report issues
3 'use strict';
4
5 const functions = require('firebase-functions');
6 const {WebhookClient} = require('dialogflow-fulfillment');
7 const {Card, Suggestion} = require('dialogflow-fulfillment');
8
9 process.env.DEBUG = 'dialogflow:debug'; // enables lib debugging statements
10
11 exports.dialogflowFirebaseFulfillment = functions.https.onRequest((request, response) => {
12   const agent = new WebhookClient({ request, response });
13   console.log('Dialogflow Request headers: ' + JSON.stringify(request.headers));
```

<https://dialogflow.cloud.google.com/>

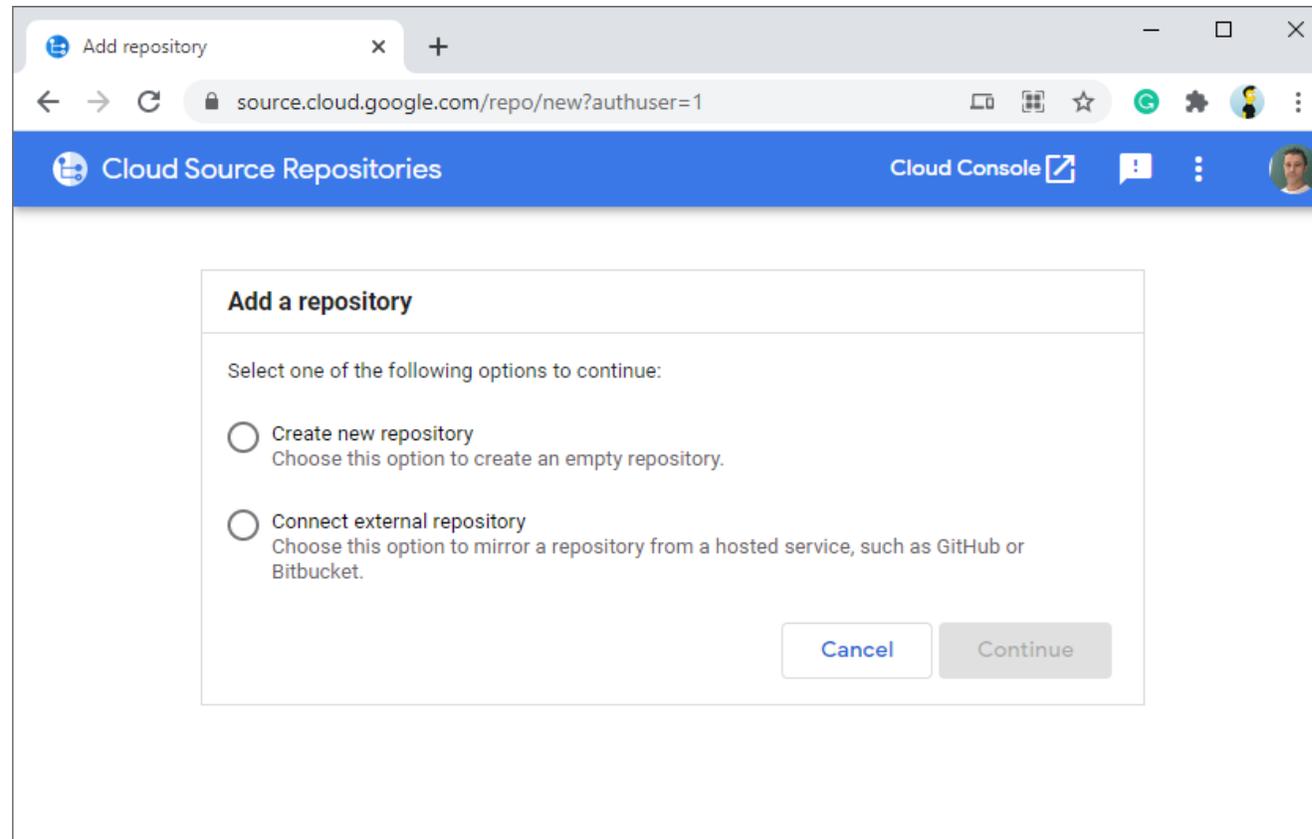
3. Cloud Source Repositories

- **Cloud Source Repositories** are fully featured, private **Git** repositories hosted on the GCP (<https://cloud.google.com/source-repositories>)
- Instead of using directly the DialogFlow inline editor, we can use these Git repositories to track the changes of our fulfillment source code
- We can use the **GCP console** to access Cloud Source Repositories:
 1. Go to <https://console.cloud.google.com/> (using our UC3M account)
 2. Select project (**uc3m-it-2021-16147-g***)
 3. Click on **Source Repositories** (on left menu, section “Tools”)



3. Cloud Source Repositories

- We can create a new repository in Cloud Source Repositories or connect with an external repo (e.g. GitHub, BitBucket)

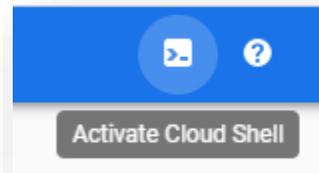


3. Cloud Source Repositories

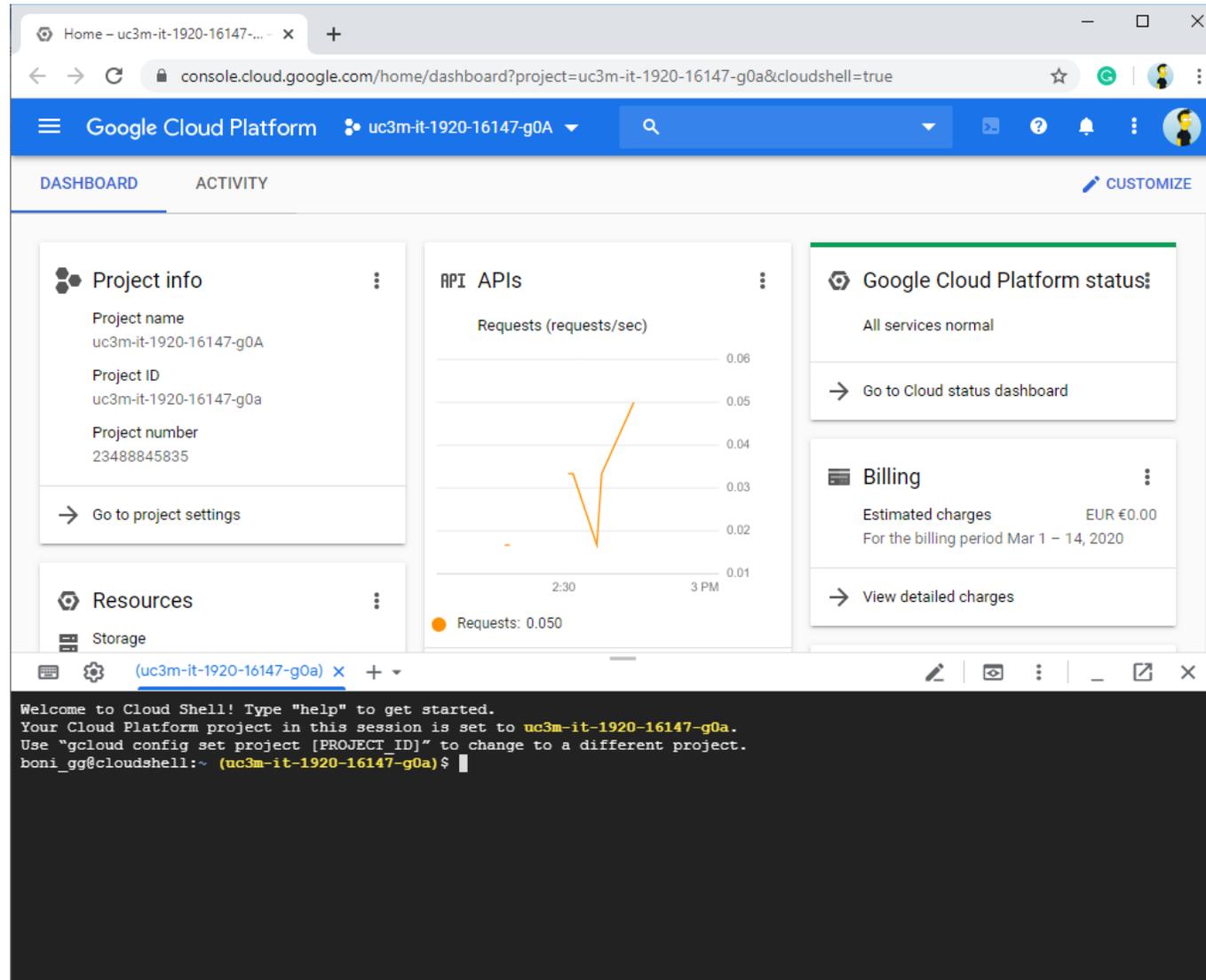
- There are different ways of using Cloud Source Repositories:
 - a) Using online services provided by GCP
 - Using the Cloud Shell and the integrated text editor (by GCP)
 - Pros: We only need a browser for the development
 - Cons: We lack advance capabilities available in IDEs such as autocompletion, autoformatting to name a few or Git GUI
 - b) Using our local environment
 - Using our shell, our favorite IDE, and so on
 - We need to configure our SSH keys in GCP to clone the Cloud Sources Repository or install Google Cloud SDK (<https://cloud.google.com/sdk>)
 - Pros: We can use advance IDEs (such as Visual Studio Code or other)
 - Cons: We need to install different tools (at least Git and Google Cloud SDK)

4. Cloud Shell

- We can use our Cloud Source Repository using the **Cloud Shell** (<https://cloud.google.com/shell>), which is an interactive shell for managing GCP projects and resources from a web browser
 - It provides command-line access to a virtual machine instance in a terminal window that opens in the web console
 - It provides many command-line tools already pre-installed (git, gcloud CLI, ...)
 - It provides 5 GB of persistent disk storage mounted as our \$HOME (this storage is not shared, i.e. it is different for each user)
- We can enable the Cloud Shell using the GCP console:
 1. Go to <https://console.cloud.google.com/> (using our UC3M account)
 2. Click on the following icon on the top right corner to active the console



4. Cloud Shell



The screenshot displays the Google Cloud Platform console interface. The browser address bar shows the URL `console.cloud.google.com/home/dashboard?project=uc3m-it-1920-16147-g0a&cloudshell=true`. The main navigation bar includes the Google Cloud Platform logo, the project ID `uc3m-it-1920-16147-g0a`, and a search icon. Below the navigation bar, there are tabs for `DASHBOARD` and `ACTIVITY`, along with a `CUSTOMIZE` button.

The dashboard content is organized into several panels:

- Project info:** Displays the project name `uc3m-it-1920-16147-g0a`, Project ID `uc3m-it-1920-16147-g0a`, and Project number `23488845835`. A link to `Go to project settings` is provided.
- RPI APIs:** A line graph titled `Requests (requests/sec)` showing data points for 2:30 and 3 PM. The y-axis ranges from 0.01 to 0.06. A legend indicates `Requests: 0.050`.
- Google Cloud Platform status:** Shows `All services normal` and a link to `Go to Cloud status dashboard`.
- Billing:** Shows `Estimated charges EUR €0.00` for the billing period `Mar 1 - 14, 2020`. A link to `View detailed charges` is provided.

At the bottom, a Cloud Shell terminal window is open, displaying the following text:

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to uc3m-it-1920-16147-g0a.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
boni_gg@cloudshell:~ (uc3m-it-1920-16147-g0a)$
```

4. Cloud Shell

- We need to select the way in which we clone our repository:

For using local environment (option 2b).
A [SSH key pair](#) is required

For using cloud services (option 2a)

The screenshot shows the 'SSH authentication' section of the Google Cloud Shell documentation. The 'SSH authentication' and 'Google Cloud SDK' tabs are circled in red. The 'SSH authentication' tab is selected and underlined. The 'Manually generated credentials' tab is also visible. The content includes a list of steps for setting up SSH keys and cloning a repository.

[SSH authentication](#) **Google Cloud SDK** Manually generated credentials

1. Setup SSH key. [Learn how](#) .
If you already have a SSH key on your machine, skip to step 2. [Find SSH Keys on your machine](#) .
2. Register the SSH key with Google Cloud.
3. Clone this repository to a local Git repository:
 - Clone with command line

```
$ git clone ssh://boni.gg@gmail.com@source.developers.google.com:2022/p/uc3m-it-1920-16147-g0a/r/uc3m-it-1920-16147-g0A
```

- Or clone with VS Code [Clone](#)
- 4. Switch to your new local Git repository:

```
$ cd uc3m-it-1920-16147-g0A
```
- 5. After you've committed code to your local Git repository, push it to this repository:

```
$ git push -u origin master
```

4. Cloud Shell

- We can clone our repository using git:

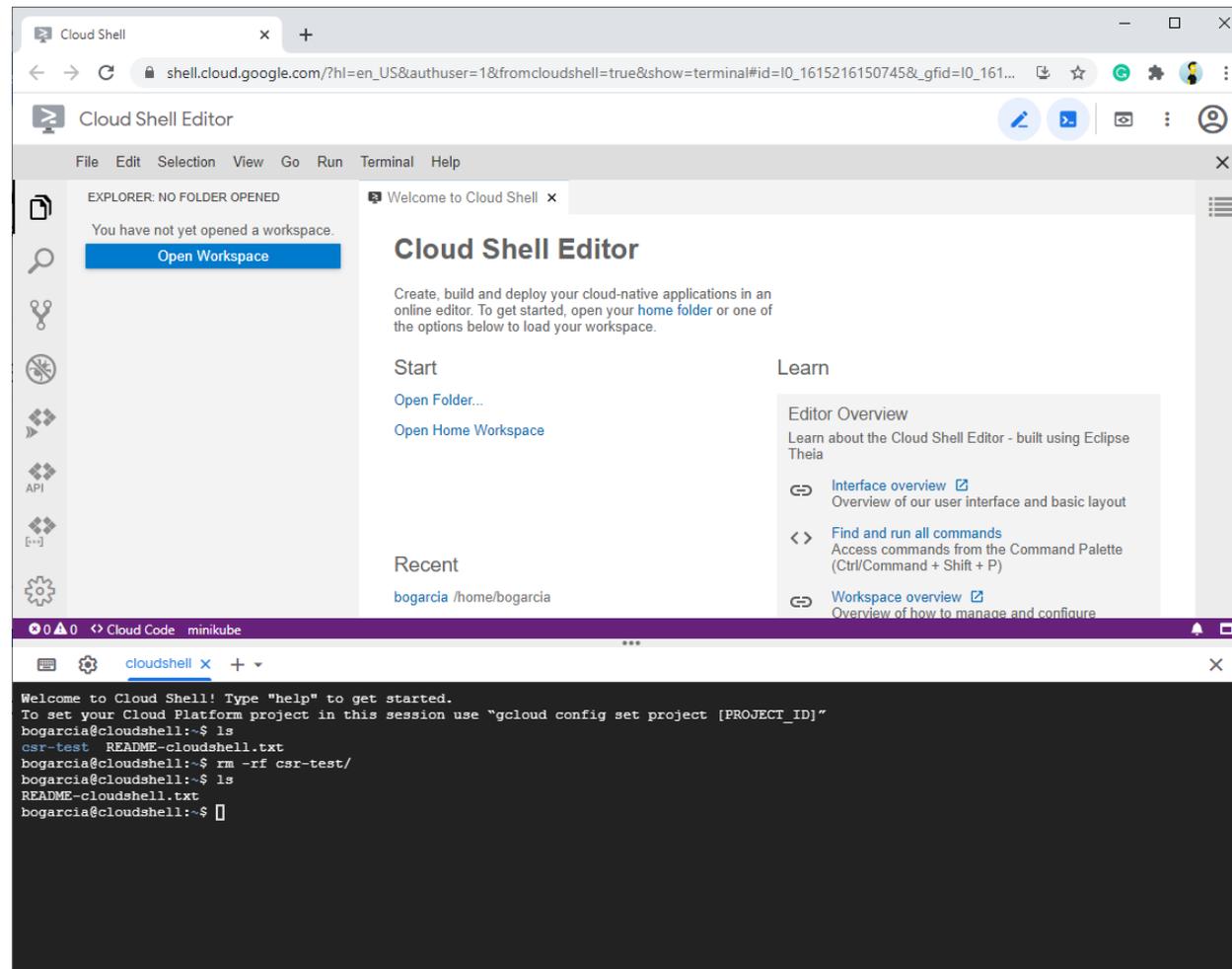
```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to uc3m-it-2021-16147-g0X.
bogarcia@cloudshell:~ (uc3m-it-2021-16147-g0X)$ git clone
https://source.developers.google.com/p/uc3m-it-2021-16147-teachers/r/myrepo
Cloning into 'myrepo'...
warning: You appear to have cloned an empty repository.
bogarcia@cloudshell:~ (uc3m-it-2021-16147-g0X)$ cd myrepo
bogarcia@cloudshell:~/myrepo (uc3m-it-2021-16147-g0X)$ git config --global user.email
"myemail@alumnos.uc3m.es"
bogarcia@cloudshell:~/myrepo (uc3m-it-2021-16147-g0X)$ git config --global user.name "My name"
```

We can get this URL from the help to clone using manually generated credentials

The first time using git in the Cloud Shell, we need to configure our email and user name

4. Cloud Shell

- Together with Cloud Shell, we can use the **Cloud Shell Editor**:



5. Google Cloud SDK

- The **Google Cloud SDK** is a set of tools and libraries for interacting with GCP services
- One of these tools is **gcloud CLI** (Command-Line Interface), and can be used to deploy our fulfillment as a **cloud function**
 - It replaces Firebase CLI for DialogFlow deployment
 - It is already installed in Cloud shell (for option 2a)
 - We need to install in local (for option 2b)

<https://cloud.google.com/sdk/docs/install>

5. Google Cloud SDK

- The development of our agent involves:
 - `index.js`: Source code of our fulfillment
 - `package.json`: Project setup and dependencies
- The typical workflow to develop and deploy our fulfillment is using our local machine (option 2b):

```
> git clone ssh://myuser@it.uc3m.es@source.developers.google.com:2022/p/uc3m-it-2021-16147-g0x/r/myrepo  
> cd myrepo
```

1. Clone repo

[development]

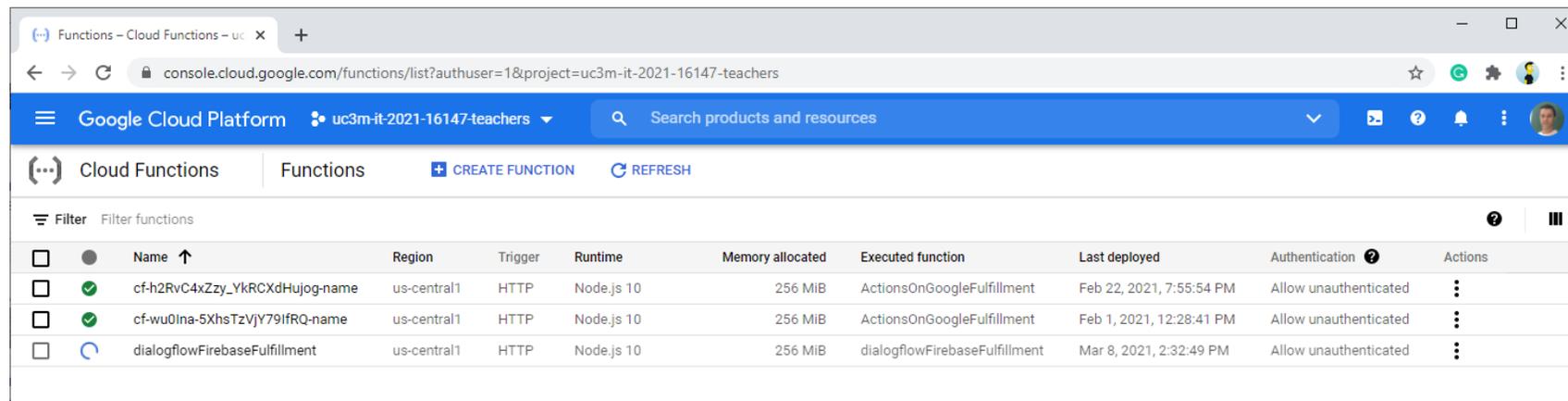
2. Deploy function

[git management]

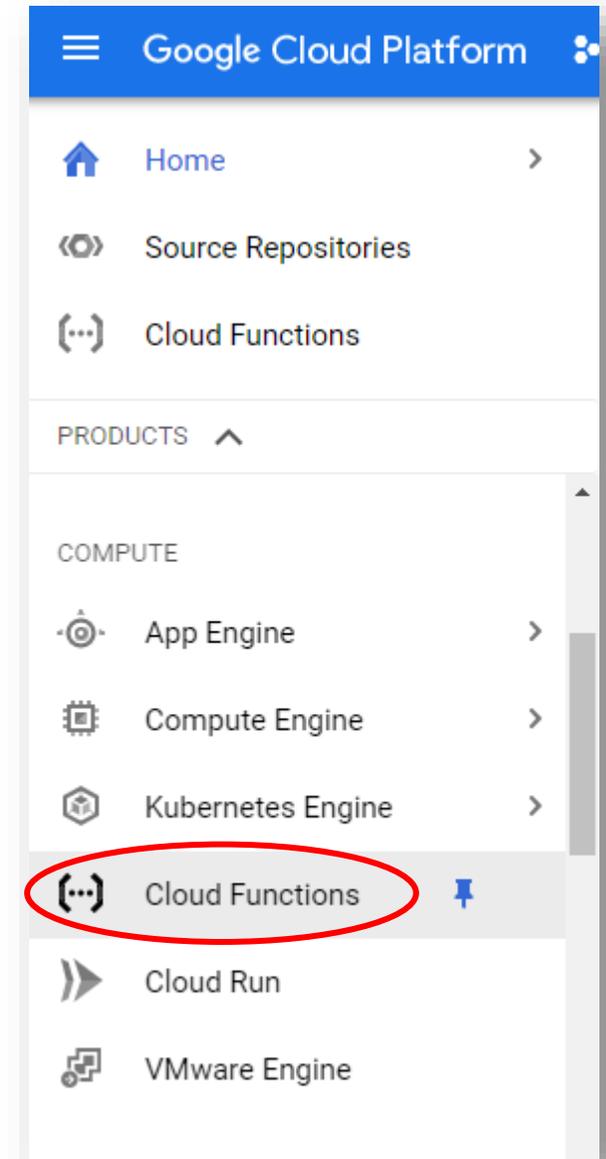
```
> gcloud functions deploy dialogflowFirebaseFulfillment --runtime=nodejs10 --allow-unauthenticated --trigger-http
```

5. Google Cloud SDK

- To check the deployment of our cloud function:
 1. Open GCP console
(<https://console.cloud.google.com/>)
 2. Select project (**uc3m-it-2021-16147-g***)
 3. Click on **Cloud Functions** (on left menu, section “Compute”)



<input type="checkbox"/>	<input type="checkbox"/>	Name ↑	Region	Trigger	Runtime	Memory allocated	Executed function	Last deployed	Authentication	Actions
<input type="checkbox"/>	<input checked="" type="checkbox"/>	cf-h2RvC4xZzy_YkRCXdHujog-name	us-central1	HTTP	Node.js 10	256 MIB	ActionsOnGoogleFulfillment	Feb 22, 2021, 7:55:54 PM	Allow unauthenticated	⋮
<input type="checkbox"/>	<input checked="" type="checkbox"/>	cf-wu0lna-5XhsTzVjY9lfrQ-name	us-central1	HTTP	Node.js 10	256 MIB	ActionsOnGoogleFulfillment	Feb 1, 2021, 12:28:41 PM	Allow unauthenticated	⋮
<input type="checkbox"/>	<input checked="" type="checkbox"/>	dialogflowFirebaseFulfillment	us-central1	HTTP	Node.js 10	256 MIB	dialogflowFirebaseFulfillment	Mar 8, 2021, 2:32:49 PM	Allow unauthenticated	⋮



5. Google Cloud SDK

- After deploying correctly a Cloud Function, we can see that new changes are synchronized in the inline editor of DialogFlow

The screenshot displays the DialogFlow Fulfillment inline editor. The interface includes a sidebar with navigation options like Intents, Entities, Knowledge, Fulfillment, Integrations, Training, Validation, History, Analytics, Prebuilt Agents, and Small Talk. The main content area shows the 'Webhook' section, which is currently disabled. Below it, the 'Inline Editor' is enabled, showing a code editor with the following JavaScript code:

```
index.js package.json
1 // See https://github.com/dialogflow/dialogflow-fulfillment-nodejs
2 // for Dialogflow fulfillment library docs, samples, and to report is
3 // From the Google Cloud Shell
4 'use strict';
5
6 const functions = require('firebase-functions');
7 const { WebhookClient } = require('dialogflow-fulfillment');
8 const { Card, Suggestion } = require('dialogflow-fulfillment');
9 const { Carousel } = require('actions-on-google');
10
11 process.env.DEBUG = 'dialogflow:debug'; // enables lib debugging stat
12
13 exports.dialogflowFirebaseFulfillment = functions.https.onRequest((re
14   const agent = new WebhookClient({ request, response });
15
```

At the bottom of the code editor, there is a 'DEPLOY' button and a timestamp indicating the last deployment on 03/14/2020 at 16:48. A 'Try it now' button is circled in red in the top right corner of the interface. A callout box points to this button with the text: "The quickest way to test our agent is using the 'Try it now' field".

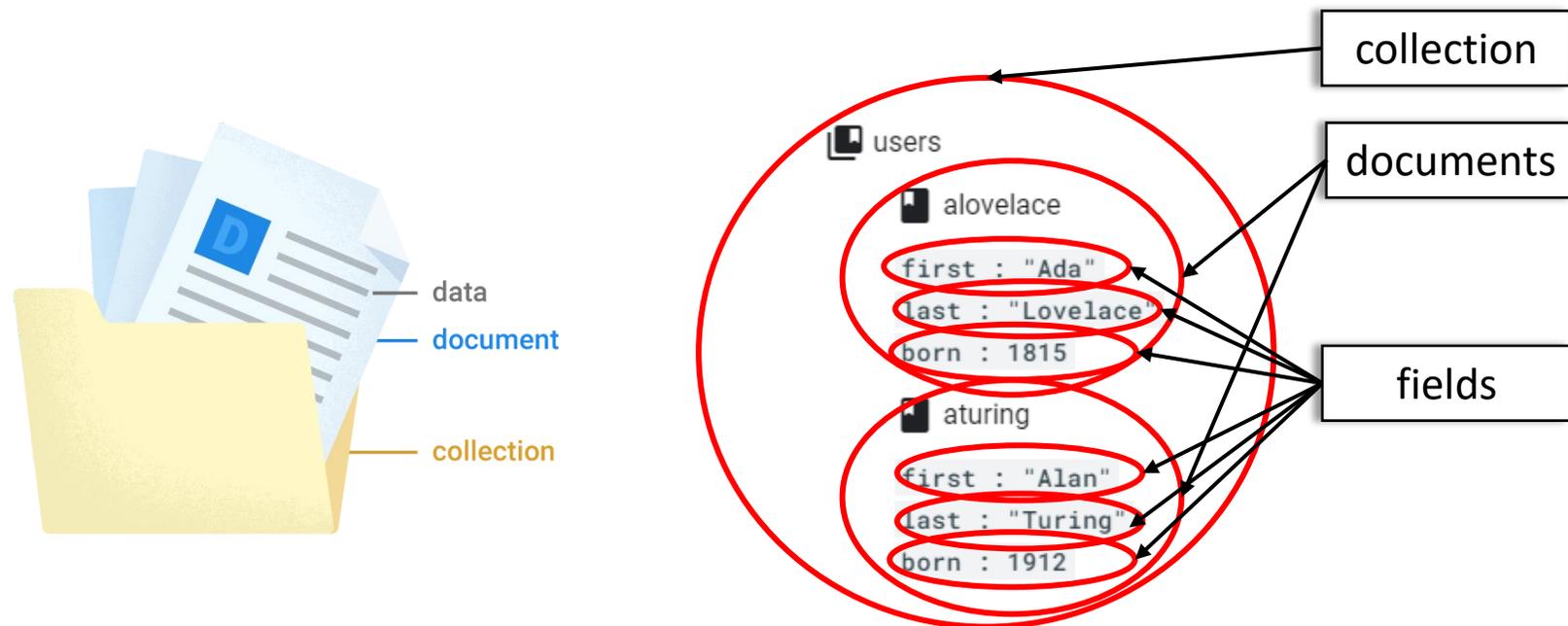
6. Firestore

- Firebase provides different databases:
 1. Realtime Database. Original NoSQL database provided by Firebase
 2. Firestore. Successor of the Realtime Database. It provides also NoSQL storage in 2 modes:
 - Native, which allows to handle data using an intuitive approach based on data structured as collections → documents → fields. It is the recommended mode for most the new projects (web, mobile, etc.)
 - Datastore, enhancing the native mode (e.g. improve performance, removed limitations on transactions)
- In our UC3M projects, we will use **Firestore** in **native** mode

<https://firebase.google.com/docs/firestore/rtdb-vs-firestore>
<https://cloud.google.com/firestore/docs/firestore-or-datastore>

6. Firestore

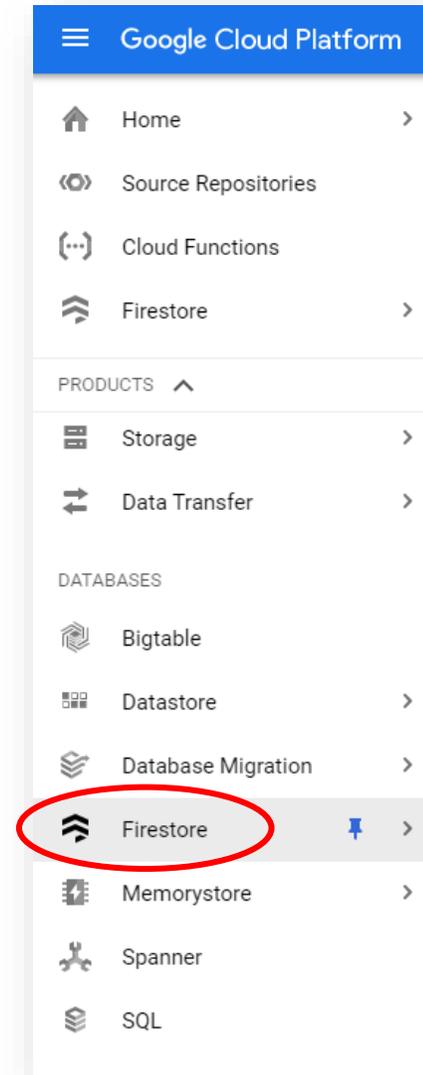
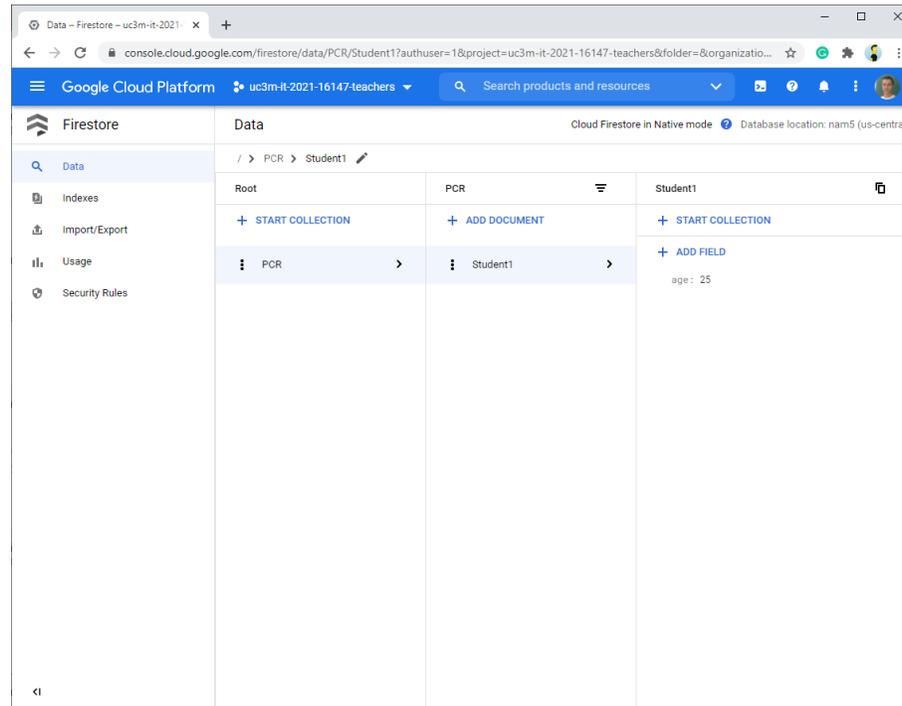
- In Firestore, the basic unit of storage is the **document**
- A document is a lightweight record that contains **fields**, which map to values
- Documents live in **collections**, which are simply containers for documents



<https://firebase.google.com/docs/firestore/data-model>

6. Firestore

- We can see the data in the Firestore console:
 1. Go to <https://console.cloud.google.com/>
 2. Select project (uc3m-it-2021-16147-g*)
 3. Click on **Source Repositories** (on left menu, section “Database”)



6. Firestore

- Example: CRUD (Create Read Update Delete) operations using an standalone script (outside GCP) in Node.js using **promises**



<https://github.com/bonigarcia/nodejs-examples>

```
const admin = require("firebase-admin");

// FIXME: Go to IAM & admin > Service accounts in the Cloud Platform Console
// (https://console.cloud.google.com/iam-admin/serviceaccounts) and generate
// a private key and save as as JSON file
const serviceAccount = require("../uc3m-it-2021-16147-teachers-3dce9f913dbc.json");
admin.initializeApp({
  credential: admin.credential.cert(serviceAccount)
});

const db = admin.firestore();

// 1. Add data
// https://firebase.google.com/docs/firestore/manage-data/add-data

// 1a. Add a new document with a generated id
let tokyo = {
  name: "Tokyo",
  country: "Japan"
};

let addDoc = db.collection("cities").add(tokyo).then(ref => {
  let tokyoId = ref.id;
  console.log("Added document with ID:", tokyoId, tokyo);

  // 2. Delete data
  // https://firebase.google.com/docs/firestore/manage-data/delete-data
  let deleteDoc = db.collection("cities").doc(tokyoId).delete();
  console.log("Deleted document with ID:", tokyoId);
});

// ...
```

6. Firestore

- Node.js is a JavaScript runtime which executes code in a single-threaded **event loop** (only one piece of code can run at a time)
- There are different alternatives to handle **asynchronous** (non-blocking) operations (e.g., call a REST service or filesystem/database operations):

1. Callbacks

- Callbacks are the functions that are called when a particular execution gets completed
- Callbacks are registered in task queue, and is executed in the main event loop when the operation is completed
- Problem: callback hell

```
a(function (resultsFromA) {
  b(resultsFromA, function (resultsFromB) {
    c(resultsFromB, function (resultsFromC) {
      d(resultsFromC, function (resultsFromD) {
        e(resultsFromD, function (resultsFromE) {
          f(resultsFromE, function (resultsFromF) {
            console.log(resultsFromF);
          });
        });
      });
    });
  });
});
```

2. Promises (introduced in EcmaScript 6)

- Promises are objects which holds the results of an asynchronous function
- Promises are registered in the job queue
- It has 3 states: pending, fulfilled, or rejected

3. Async/await (introduced in EcmaScript 8)

- Syntax sugar (built on top of promises) to simulate asynchronous operations synchronously

6. Firestore

- Equivalent example but using `async/await`

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function

```
const admin = require("firebase-admin");
const serviceAccount = require("../uc3m-it-2021-16147-teachers-3dce9f913dbc.js...");
admin.initializeApp({
  credential: admin.credential.cert(serviceAccount)
});
const db = admin.firestore();

(async () => {
  try {
    // 1. Add data
    // https://firebase.google.com/docs/firestore/manage-data/add-data
    // 1a. Add a new document with a generated id
    let tokyo = {
      name: "Tokyo",
      country: "Japan"
    };
    let ref = await db.collection("cities").add(tokyo);
    let tokyoId = ref.id;
    console.log("Added document with ID:", tokyoId, tokyo);

    // 2. Delete data
    // https://firebase.google.com/docs/firestore/manage-data/delete-data
    let deleteDoc = db.collection("cities").doc(tokyoId).delete();
    console.log("Deleted document with ID:", tokyoId);

    // ...

  } catch (error) {
    console.error("Error happened:", error);
  }
})();
```

7. Fulfillment examples

- The following example shows how to make a call to a REST service
 - It's important to notice that when we make an asynchronous operation in our handler, we need to return a **Promise**

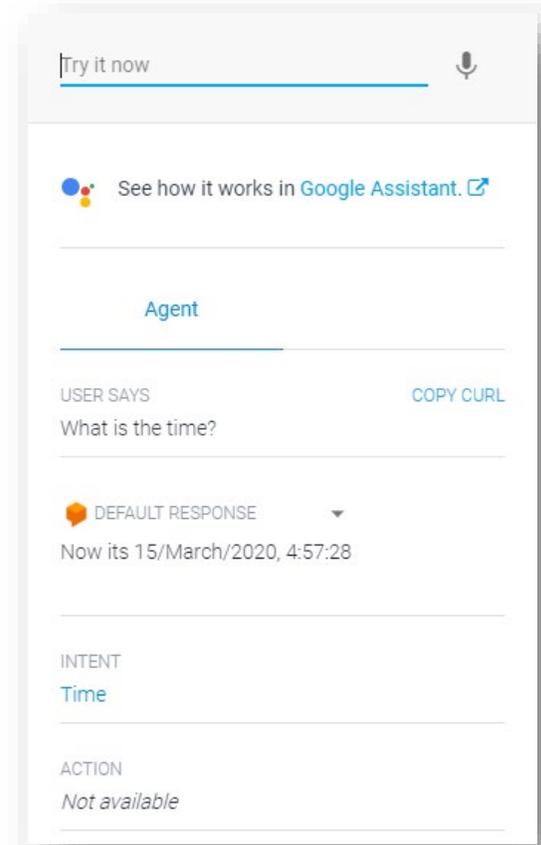
```
const axios = require("axios").default;
const dateFormat = require("dateformat");

function timeHandler(agent) {
  return new Promise((resolve, reject) => {
    axios.get("http://worldtimeapi.org/api/timezone/Europe/Madrid")
      .then(function (response) {
        let time = response.data;
        console.log("Time object:", time);
        let currentDateTime = dateFormat(time.currentDateTime, "dd/mmmm/yyyy, h:MM:ss");
        agent.add("Now its " + currentDateTime);
        return resolve();
      })
      .catch(function (error) {
        console.error("Error happened:", error);
        return reject(error);
      });
  });
}

intentMap.set("Time", timeHandler);
```

Axios is promise-based
REST library

```
"dependencies": {
  "dateformat": "^4.1.1",
  "axios": "^0.21.1"
}
```



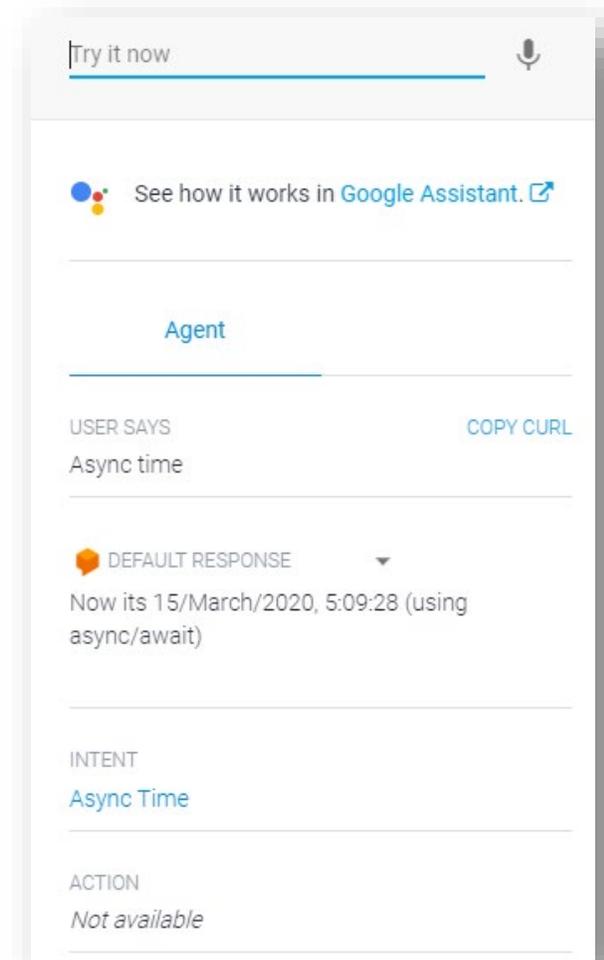
7. Fulfillment examples

- The following example makes another call to the same REST service than before, but using `async/await`

```
const axios = require("axios").default;
const dateFormat = require("dateformat");

async function timeAsyncHandler(agent) {
  try {
    let response = await axios.get("http://worldtimeapi.org/api/timezone/Europe/Madrid");
    let time = response.data;
    console.log("Time object:", time);
    let currentDateTime = dateFormat(time.currentDateTime, "dd/mmmm/yyyy, h:MM:ss");
    agent.add("Now its " + currentDateTime + " (using async/await)");
  } catch (error) {
    console.error("Error happened:", error);
  }
}

intentMap.set("Async Time", timeAsyncHandler);
```



7. Fulfillment examples

- The following example inserts data in a Firestore collection:

```
const functions = require("firebase-functions");
const admin = require("firebase-admin");

admin.initializeApp();
const db = admin.firestore();

async function addCountryHandler(agent) {
  try {
    // Add data (new document with a generated id)
    // https://firebase.google.com/docs/firestore/manage-data/add-data
    let country = {
      country: agent.parameters["geo-country"],
      capital: agent.parameters["geo-capital"]
    };
    let ref = await db.collection("world").add(country);
    let countryId = ref.id;
    agent.add("Added country " + country.country + " (capital " + country.capital + ")");
    console.log("Added country with id ", countryId, country);
  } catch (error) {
    console.error("Error happened:", error);
  }
}

intentMap.set("Add Country", addCountryHandler);
```

```
"dependencies": {
  "firebase-admin": "^9.5.0",
  "firebase-functions": "^3.13.2",
}
```

The screenshot shows a Google Assistant interface with a search bar at the top. Below the search bar, there is a link to "See how it works in Google Assistant." The main content area is titled "Agent" and shows a conversation log. The user says "Add country Spain capital Madrid". The default response is "Added country Spain (capital Madrid)". Below this, the intent is identified as "Add Country". The action is "Not available". A table shows the parameters and their values: "geo-country" is "Spain" and "geo-capital" is "Madrid".

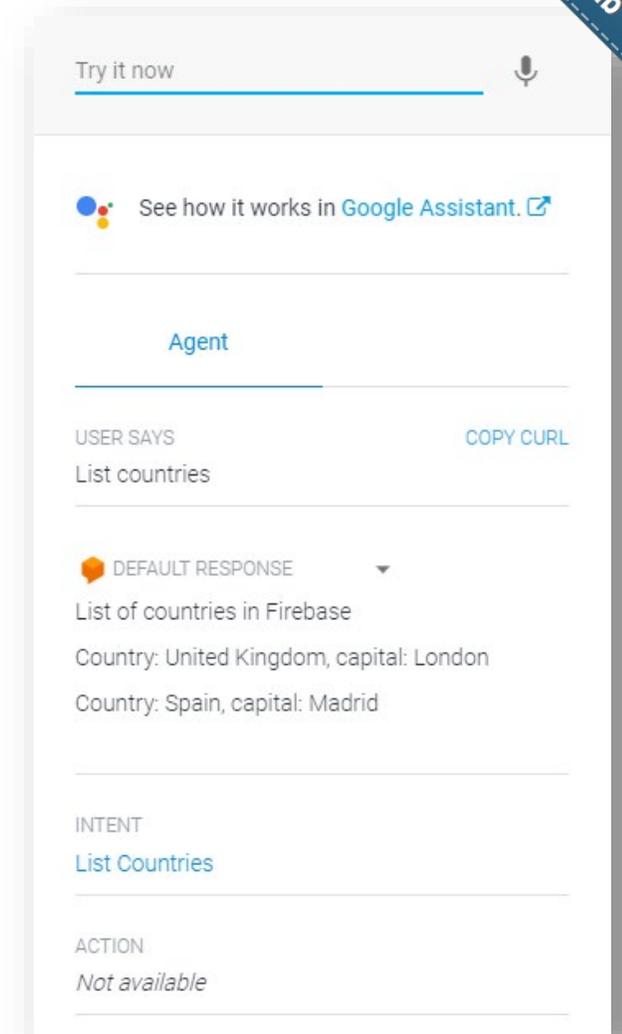
PARAMETER	VALUE
geo-country	Spain
geo-capital	Madrid

7. Fulfillment examples

- The following example reads data from Firebase:

```
async function listCountryHandler(agent) {
  try {
    // Read data
    // https://firebase.google.com/docs/firestore/query-data/get-data
    agent.add("List of countries in Firebase");
    let world = await db.collection("world").get();
    world.forEach(doc => {
      agent.add("Country: " + doc.data().country + ", capital: " + doc.data().capital);
    });
  } catch (error) {
    console.error("Error happened:", error);
  }
}

intentMap.set("List Countries", listCountryHandler);
```

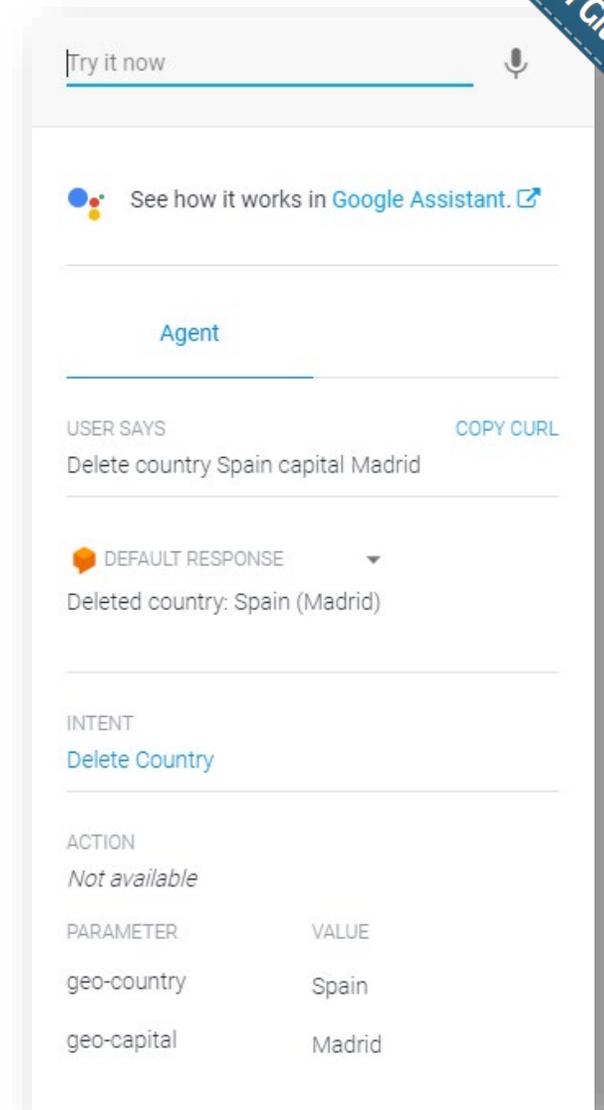


7. Fulfillment examples

- The following example deletes data from Firebase:

```
async function deleteCountryHandler(agent) {
  try {
    // Delete data
    // https://firebase.google.com/docs/firestore/manage-data/delete-data
    let country = agent.parameters["geo-country"];
    let capital = agent.parameters["geo-capital"];
    let list = await db.collection("world")
      .where("country", "==", country)
      .where("capital", "==", capital).get();
    list.forEach(doc => {
      db.collection("world").doc(doc.id).delete();
      console.log("Deleted country with id", doc.id);
    });
    agent.add("Deleted country: " + country + " (" + capital + ")");
  } catch (error) {
    console.error("Error happened:", error);
  }
}

intentMap.set("Delete Country", deleteCountryHandler);
```



Try it now 

 See how it works in [Google Assistant](#). 

Agent

USER SAYS COPY CURL

Delete country Spain capital Madrid

 DEFAULT RESPONSE ▼

Deleted country: Spain (Madrid)

INTENT

Delete Country

ACTION

Not available

PARAMETER	VALUE
geo-country	Spain
geo-capital	Madrid

Fork me on GitHub

7. Fulfillment examples

- The basic library to create DialogFlow agents with Node.js is called **dialogflow-fulfillment-nodejs**
 - <https://github.com/dialogflow/dialogflow-fulfillment-nodejs>
 - This library offers the following classes:
 - `WebhookClient` : To be used in the Dialogflow fulfillment webhook logic
 - `Text` : (`RichResponses`) text response
 - `Card` : (`RichResponses`) card response
 - `Image` : (`RichResponses`) image response
 - `Suggestion` : (`RichResponses`) suggestion response
 - `Payload` : (`RichResponses`) custom responses (typically for integration)
- To improve the user experience of our agent, it is recommended to use different types of responses

7. Fulfillment examples

- The following example shows how to use a **Card** response:

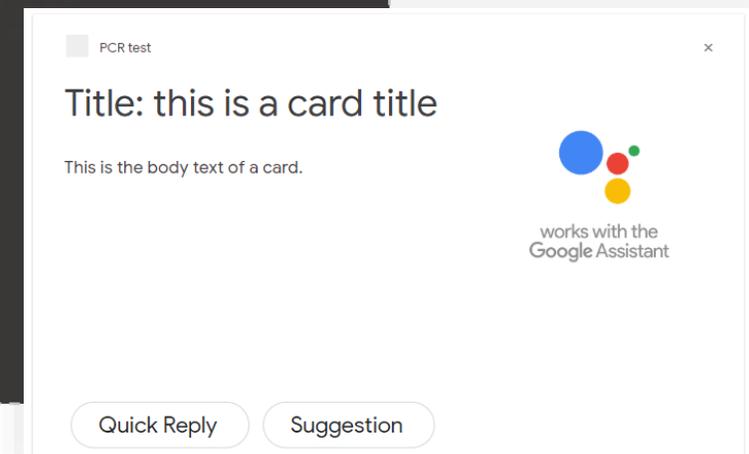
```
const { WebhookClient } = require("dialogflow-fulfillment");
const { Card, Suggestion } = require("dialogflow-fulfillment");

exports.dialogflowFirebaseFulfillment = functions.https.onRequest((request, response) => {
  const agent = new WebhookClient({ request, response });

  function cardHandler(agent) {
    agent.add("This message is from Dialogflow's Cloud Functions!");
    agent.add(new Card({
      title: "Title: this is a card title",
      imageUrl: "https://developers.google.com/assistant/images/badges/XPM_BADGING_GoogleAssistant_VER.png",
      text: "This is the body text of a card.",
      buttonText: "This is a button",
      buttonUrl: "https://assistant.google.com/"
    }));
    agent.add(new Suggestion("Quick Reply"));
    agent.add(new Suggestion("Suggestion"));
  }

  let intentMap = new Map();
  intentMap.set("Card", cardHandler);
  agent.handleRequest(intentMap);
});
```

```
"dependencies": {
  "dialogflow": "^1.2.0",
  "dialogflow-fulfillment": "^0.6.1"
}
```



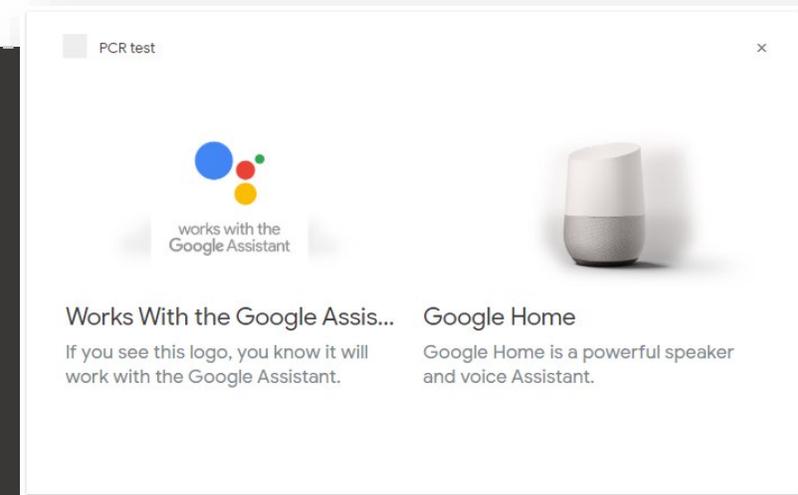
7. Fulfillment examples

- The library **actions-on-google-nodejs** allows to interact with Google Assistant (<https://github.com/actions-on-google/actions-on-google-nodejs>)
- This example shows how to use a **Carousel** in Google Assistant:

```
const { Carousel } = require("actions-on-google");

function carouselHandler(agent) {
  let conv = agent.conv();
  conv.ask("Please choose an item:");
  conv.ask(new Carousel({
    title: "Google Assistant",
    items: {
      "WorksWithGoogleAssistantItemKey": {
        title: "Works With the Google Assistant",
        description: "If you see this logo, you know it will work with the Google Assistant.",
        image: {
          url: "https://developers.google.com/assistant/images/badges/XPM_BADGING_GoogleAssistant_VER.png",
          accessibilityText: "Works With the Google Assistant logo",
        },
      },
      "GoogleHomeItemKey": {
        title: "Google Home",
        description: "Google Home is a powerful speaker and voice Assistant.",
        image: {
          url: "https://lh3.googleusercontent.com/Nu3a6F80WfixUqf_ec_vgXy_c0-0r4VLJRXjVFF_X_Ci1Eu8B9fT35qyTEj_PESk",
          accessibilityText: "Google Home"
        },
      },
    },
  }));
  agent.add(conv);
}

intentMap.set("Carousel", carouselHandler);
```



```
"dependencies": {
  "actions-on-google": "^2.13.0",
}
```

8. Account linking

- If we need **authenticate** the users of our agent, we can use built-in features for **Google Sing-in** (i.e., allows to login in our agent using a Google account)
- This feature is implemented through **Google Assistant**
- The procedure to use it is the following:
 1. Activate account linking
 2. Specify Google Assistant Sing In in some intent
 3. Code fulfillment using **SignIn**

<https://developers.google.com/assistant/identity/google-sign-in>

8. Account linking

1. Activate account linking

- Go to the account linking section of the actions console
- Enable “Account linking”
- Select “Yes” to allow user to sign up for new accounts
- Set “Google Sign in” as linking type
- Copy Client ID (it is used in the fulfillment code)

URL: <https://console.actions.google.com/u/1/project/ZZZZZZZ/accountlinking/>
(... where ZZZZZZZ is the project name)

The screenshot shows the 'Account linking' configuration page in the Google Actions Console. The 'Account linking' toggle is turned on. The 'Yes' radio button is selected for allowing new account sign-ups. The linking type is set to 'Google Sign In'. The Client ID is displayed and highlighted with a red circle.

Account linking

Account creation

Do you want to allow users to sign up for new accounts via voice? [Help](#)

Yes

No, I only want to allow account creation on my website
Select this option if you want to display your terms of service or obtain user consents during sign-up.

Linking type

Google Sign In

Google Sign In Client Information

Client ID issued by Google to your Actions ⓘ

970990056627-d4dsdh5qv6226mrmdfvidben8313vh

8. Account linking

2. Specify Google Assistant Sing In in some intent

The screenshot shows the Dialogflow console interface for editing an intent. The browser address bar indicates the URL: `dialogflow.cloud.google.com/#/agent/uc3m-it-2021-16147-teachers/editIntent/dd2bb306-94de-48cb-b5f7-b2f4dd119192/`. The page title is "Default Welcome Intent" with a "SAVE" button. The left sidebar shows navigation options: Dialogflow Essentials, SampleAgent-teach..., Intents (selected), Entities, Knowledge [beta], Fulfillment, Integrations, Training, Validation, History, Analytics, Prebuilt Agents, Small Talk, and Docs. The main content area is divided into sections: Contexts, Events, and Training phrases. The "Events" section contains a list of events: "Welcome" and "Google Assistant Sign In". The "Google Assistant Sign In" event is circled in red. A blue callout box points to this event with the text: "For instance, in the 'Default Welcome Intent'". The "Training phrases" section contains a search bar and a list of phrases: "Add user expression", "just going to say hi", "heya", "hello hi", "howdy", "hey there", "hi there", and "greetings".

8. Account linking

3. Code fulfillment using `SignIn`

```
// Imports
const { dialogflow, SignIn, Image, Carousel, Suggestions } = require("actions-on-google");
const functions = require("firebase-functions");
const axios = require("axios").default;
const dateFormat = require("dateformat");
const admin = require("firebase-admin");

// Dialogflow setup
const app = dialogflow({
  clientId: "XXXXXXXXXX",
  // XXXXXXXXXXXX = Client ID issued by Google to your Actions
  // Get this clientId from https://console.actions.google.com/u/1/project/ZZZZZZZ/accountlinking/
  // ... where ZZZZZZZ is the name of your project
});
exports.dialogflowFirebaseFulfillment = functions.https.onRequest(app);

// Intent handlers:
app.intent("Default Welcome Intent", (conv) => {
  const payload = conv.user.profile.payload;
  if (payload) {
    console.log("**** payload:", payload);
    conv.ask(`Welcome to my agent, ${payload.given_name}! What do you want to do next?`);
  } else {
    conv.ask(new SignIn("hello"));
  }
});
```

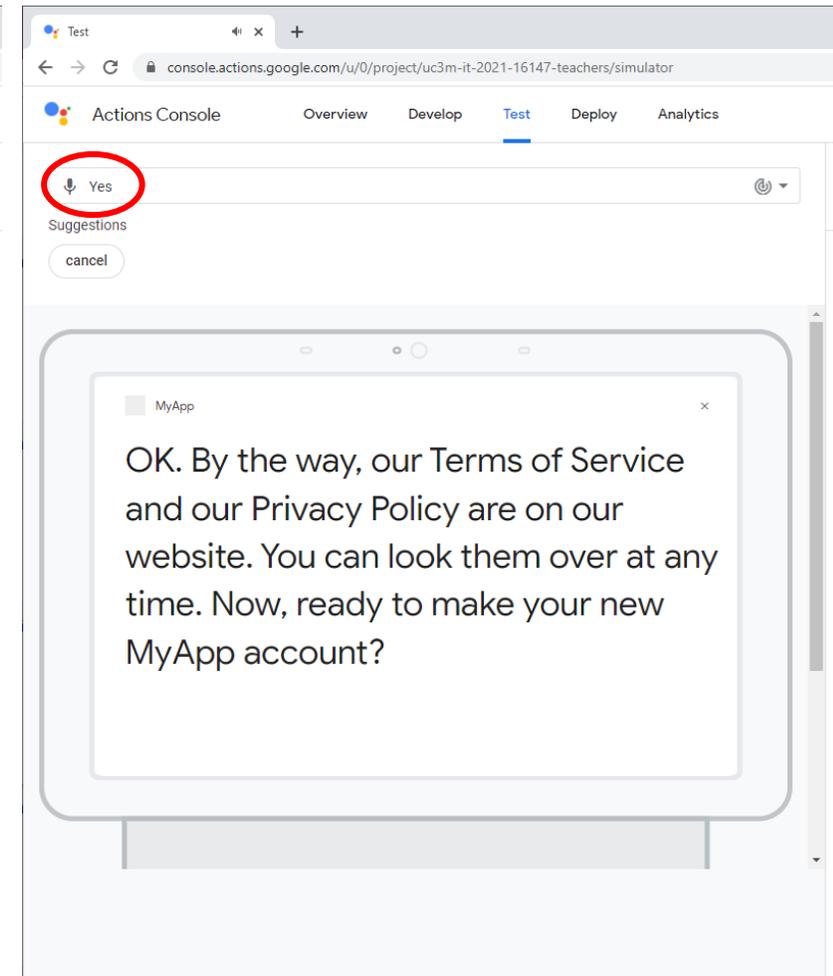
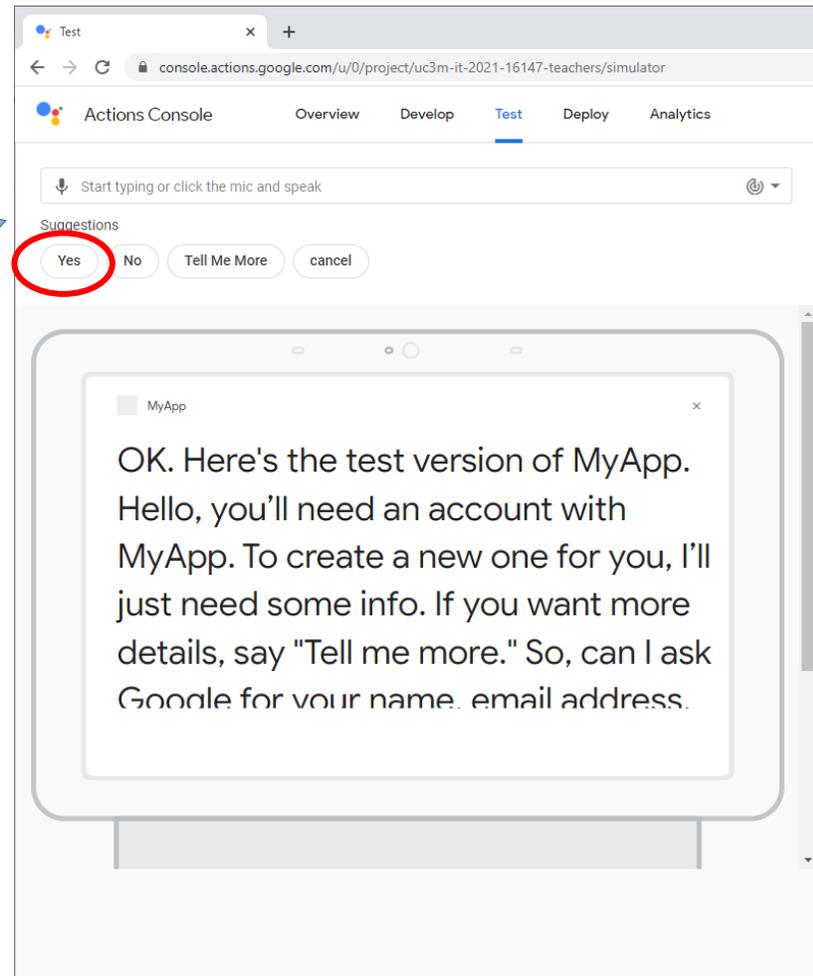
Set client id here

Handler for our "Default Welcome Intent" intent

8. Account linking

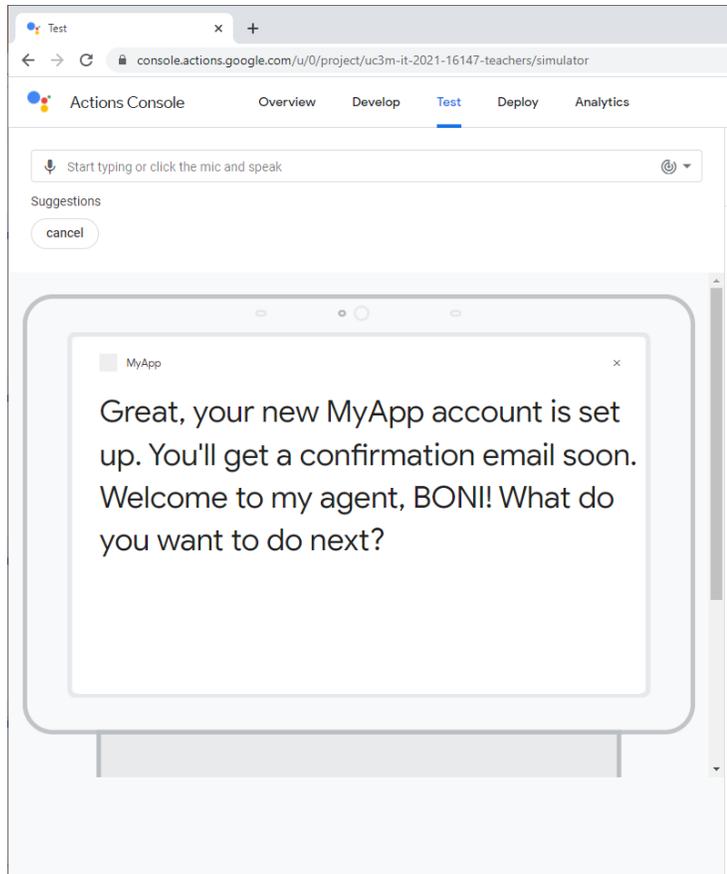
- To use this Google Sign, we need to use it using Google Assistant:

The first time, the user is prompted with some information about sign-in. He/she needs to accept (typing "Yes" twice)

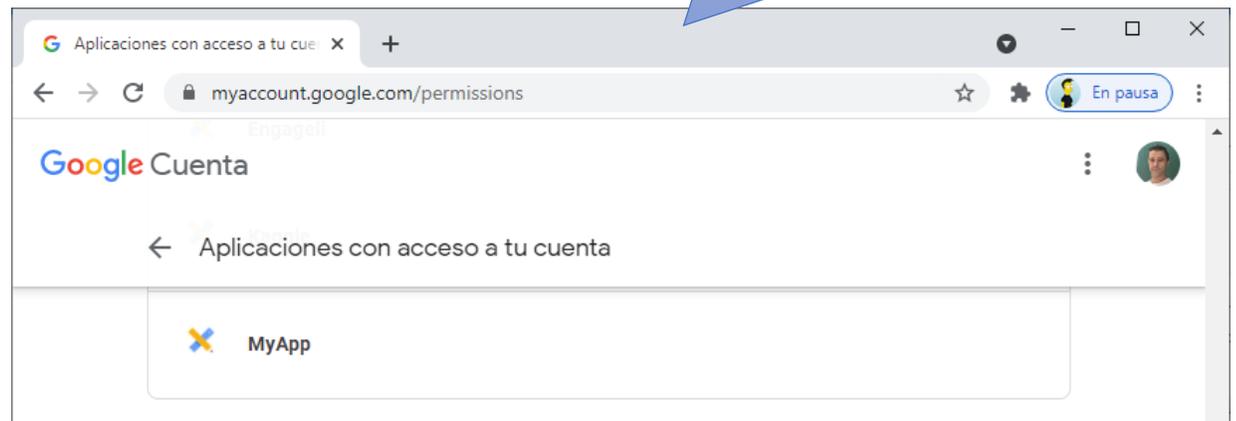


8. Account linking

- After that, the user account is used in the agent (a confirmation email is sent)



The agent now appears in the permissions page of the Google account:
<https://myaccount.google.com/permissions>



9. Local deployment

- So far, we have seen that to test any change in our fulfillment code, it must be deployed
 - Using the inline editor
 - Using our local environment and deploying using the command `gcloud functions deploy`
- Problem: the deployment of a cloud function takes several minutes to be completed
- Solution: deploy our fulfillment in the local machine and serve it through a public URL (e.g. using **ngrok**)

9. Local deployment

- To deploy our fulfillment in local, first we need **firebase CLI**. We can use npm for that:

```
$ npm install -g firebase-tools
```

- Firebase CLI can be used to deploy (as a cloud function) or serve (in local) our fulfillment code
- The first time, we need login in Firebase (`firebase login --no-localhost`)
 - We will need to copy and paste the provided URL in a web browser, authenticate with our UC3M account, and paste the authorization code in the shell

```
$ firebase login --no-localhost
i  Firebase optionally collects CLI usage and error reporting information to help improve our products. Data
is collected in accordance with Google's privacy policy (https://policies.google.com/privacy) and is not
used to identify you.
? Allow Firebase to collect CLI usage and error reporting information? No
Visit this URL on any device to log in:
https://accounts.google.com/o/oauth2/auth?client\_id..
? Paste authorization code here: xxxxxxxxxxxxxx
✓ Success! Logged in as boni.gg@gmail.com
```

9. Local deployment

- Then, we need to create the project scaffolding using the command `firebase init`:

```
$ firebase init
##### ##### ##### ##### #####   ##   ##### #####
##      ##  ##   ##  ##   ##   ##  ##  ##  ##   ##
#####   ##  ##### #####   ##### ##### ##### #####
##      ##  ##   ##  ##   ##   ##  ##  ##   ##  ##
##      ##### ##   ##  ##### ##### ##   ##  ##### #####
You're about to initialize a Firebase project in this directory:
  /home/boni_gg/uc3m-it-1920-16147-g0A
? Which Firebase CLI features do you want to set up for this folder? Press Space to select features,
then Enter to confirm your choices.
   Database: Deploy Firebase Realtime Database Rules
   Firestore: Deploy rules and create indexes for Firestore
  >  Functions: Configure and deploy Cloud Functions
   Hosting: Configure and deploy Firebase Hosting sites
   Storage: Deploy Cloud Storage security rules
   Emulators: Set up local emulators for Firebase features
```

(continue in next slide)

9. Local deployment

(continue from previous slide)

```
=== Project Setup
```

```
First, let's associate this project directory with a Firebase project.  
You can create multiple project aliases by running firebase use --add,  
but for now we'll just set up a default project.
```

```
? Please select an option: Use an existing project
```

```
? Select a default Firebase project for this directory: uc3m-it-1920-16147-g0a (uc3m-it-1920-16147-g0A)
```

```
i Using project uc3m-it-1920-16147-g0a (uc3m-it-1920-16147-g0A)
```

```
=== Functions Setup
```

```
A functions directory will be created in your project with a Node.js  
package pre-configured. Functions can be deployed with firebase deploy.
```

```
? What language would you like to use to write Cloud Functions? JavaScript
```

```
? Do you want to use ESLint to catch probable bugs and enforce style? No
```

```
✓ Wrote functions/package.json
```

```
✓ Wrote functions/index.js
```

```
✓ Wrote functions/.gitignore
```

```
? Do you want to install dependencies with npm now? Yes
```

```
> protobufjs@6.8.9 postinstall /home/boni_gg/uc3m-it-1920-16147-g0A/functions/node_modules/protobufjs
```

```
> node scripts/postinstall
```

```
npm notice created a lockfile as package-lock.json. You should commit this file.
```

```
added 249 packages from 188 contributors and audited 834 packages in 9.764s
```

```
found 0 vulnerabilities
```

```
i Writing configuration info to firebase.json...
```

```
i Writing project information to .firebaserc...
```

```
i Writing gitignore file to .gitignore...
```

```
✓ Firebase initialization complete!
```

9. Local deployment

- Then, we can proceed to development of our agent (files `index.js` and `package.json`)
- After that, we need to resolve the Node.js dependencies using the command `npm install`:

```
$ npm install
npm WARN notsup Unsupported engine for dialogflow-fulfillment@0.6.1: wanted: {"node":"6"} (current: {"node":"12.16.1","npm":"6.13.4"})
npm WARN notsup Not compatible with your version of node/npm: dialogflow-fulfillment@0.6.1

added 90 packages from 101 contributors, removed 2 packages and audited 288 packages in 5.145s

5 packages are looking for funding
  run `npm fund` for details

found 6 high severity vulnerabilities
  run `npm audit fix` to fix them, or `npm audit` for details
```

At this point, our fulfillment can be deployed as a cloud function using the command: `firebase deploy` (it is equivalent to `gcloud functions deploy`)

9. Local deployment

- Then, we serve our fulfillment as a local function, using the command `firebase serve --only functions`:

```
$ firebase serve --only functions
! Your requested "node" version "10" doesn't match your global version "12"
i functions: Watching "C:\Users\boni\Downloads\ngrok\functions" for Cloud Functions...
! functions: The Cloud Firestore emulator is not running, so calls to Firestore will affect production.
+ functions[dialogflowFirebaseFulfillment]: http function initialized (http://localhost:5000/uc3m-it-2021-16147-teachers/us-central1/dialogflowFirebaseFulfillment).
```

At this point, our function is deployed locally (in the port 5000 by default). Now, we need a way to expose this service using a public URL. For that, we are going to use **ngrok**

9. Local deployment

- **ngrok** is a tool that allows to create a tunnel using a public URL to a service deployed in the localhost
 - For that, it uses NAT traversal techniques
 - It is very useful to create public webhooks
- To install ngrok in our machine, we can use npm: `$ npm install -g ngrok`
- Then, we invoke the following command in the shell (to create a public URL pointing to localhost:5000):

ngrok

<https://ngrok.com/>

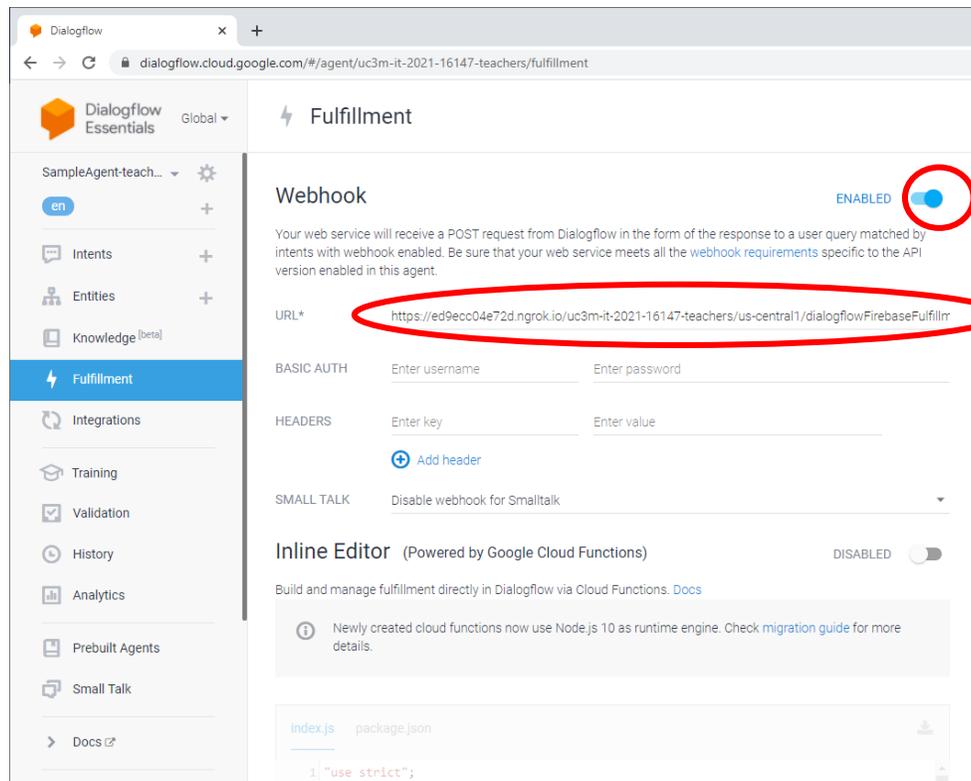
```
$ ngrok http 5000
ngrok by @inconshreveable
(Ctrl+C to quit)
Session Status      online
Session Expires    1 hour, 23 minutes
Update             update available (version 2.3.35)
Version            2.3.35
Region             United States (us)
Web Interface       http://127.0.0.1:4040
Forwarding          http://ed9ecc04e72d.ngrok.io -> http://localhost:5000
Forwarding          https://ed9ecc04e72d.ngrok.io -> http://localhost:5000
```

We need the URL using HTTPS for our agent WebHook

9. Local deployment

- We use the path our local service and the public HTTPS URL. In the example before is:

<https://ed9ecc04e72d.ngrok.io/uc3m-it-2021-16147-teachers/us-central1/dialogflowFirebaseFulfillment>



Finally, we enable the webhook in the fulfillment setup, and specify the generated URL