Google Cloud Functions Workshop

Material for a Google Cloud Function Workshop

Other Cloud Labs here

git clone https://github.com/kenlomaxhybris/gcfexamples

Agenda

- Introduce Google Cloud Functions and Google Cloud Platform
- Review important NodeJS concepts using GCP
- Debug GCF locally
- Deploy GCF to GCP and run there
- Debug with StackDriver

Preqrequisites

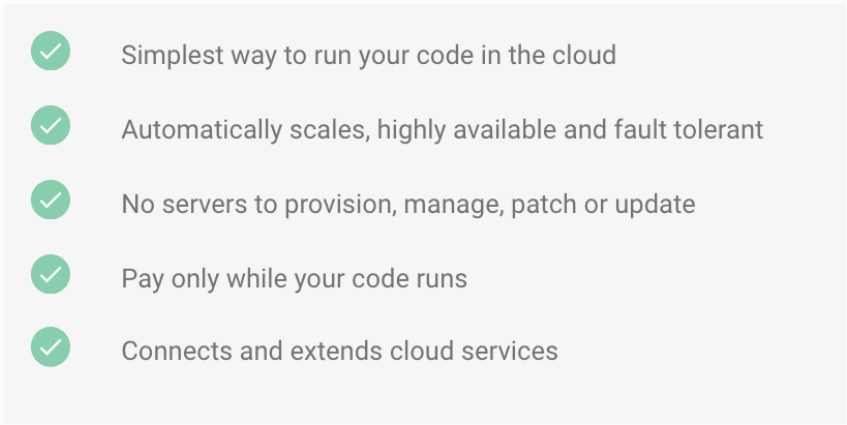
- Download Node
- Have a Google Cloud Account

Introduction

NodeJS: is a server-side Javascript platform from Google Chrome's, that runs on OS X, Linux, and Windows

Npm: is the package manager for JavaScript and the world's largest software registry.

Google Cloud Functions: are a lightweight, event-based, asynchronous compute solution that allows you to create small, single-purpose functions that respond to cloud events without the need to manage a server or a runtime environment". GCFs can be written in NodeJS, Go and Python. Andddd GCFs can meet **"Cloud scale"** demand - they are "Serverless" and GCF will look after scaling demands



https://cloud.google.com/functions/

NodeJS

GCF Overview: https://cloud.google.com/functions/

GCF Tutorials: https://cloud.google.com/functions/docs/tutorials/

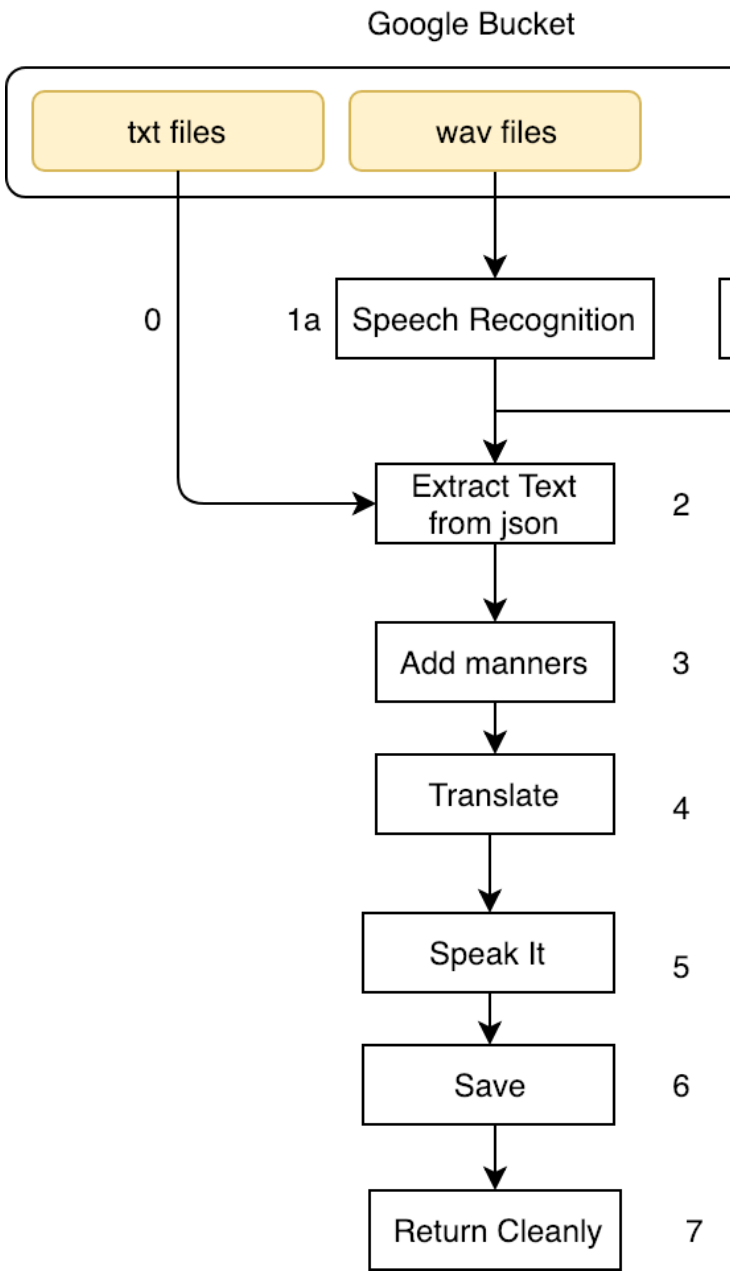
GCF **Tutorial Series:** https://rominirani.com/google-cloud-functions-tutorial-series-f04b2db739cd

To get an idea of what is possible with GCFs, we will write JS Code that

- extracts text from a txt/wav or jpeg file
- adds manners to the text
- translates the text to another language
- speaks the resulting text
- we will call this function directly from the command line, and trigger it via file uploads.

index.js

```
// For stackdriver support
//const debug = require
('@google-cloud/debug-
agent').start
({allowExpressions:true});
// End
```




```
exports.helloWorld = (json,
res) => {
  json= findJson(json)
  if (res && res.send)
    res.status(200).send
("Hello "+ json.name);
}

// Modules
var storageModule = require
('@google-cloud/storage')
var visionModule = require
('@google-cloud/vision');
var speechModule = require
('@google-cloud/speech');
var translateModule = require
('@google-cloud/translate');
var ttsModule = require
('@google-cloud/text-to-
speech');

// Clients
var storage = new
storageModule.Storage();
var vision = new
visionModule.
ImageAnnotatorClient();
var speech = new
speechModule.SpeechClient();
var translate = new
translateModule.Translate()
var textToSpeech = new
ttsModule.
TextToSpeechClient();

var id = 0 ;

exports.gcfs = (json, res)
=> {

    id++;
    var step = 0

    json = findJson(json)
    storage.bucket(json.
bucketin).file(json.file).
download().
    then((input) => {
        logging( id, input,
step++ )
        var sourceFile =
"gs://" + json.bucketin + "
```

```
/" + json.file
        // Speech recognition
        if (json.file.
endsWith(".wav"))
            return speech.
recognize( {config: {
languageCode: json.
sourceLocale  ? json.
sourceLocale  : "en-US" },
audio: {content:  input[0] }
}  );
        // Text
Recognition
        if (json.file.
endsWith(".jpeg"))
            return vision.
textDetection( sourceFile
);
        // Text file
        return input;
    }).
    then((input) => {
        // Pull the
recognized text from the json
        logging( id, input,
step++ );
        if ( input[0].
fullTextAnnotation )
            return input[0].
fullTextAnnotation.text.
replace(/\n/g, " ").trim();
        else if ( input[0].
results )
            return input[0].
results[0].alternatives[0].
transcript;
        else // txt file
            return input[0];
    }).
    then((input) => {
        // Modify the text
(add some manners)
        logging( id, input,
step++ )
        var tips =[
            "Sorry to
bother.",
            "Would you
mind.",
            "Forgive the
intrusion.",
            "Thank you so
much.",
```

```
            "You are too
kind.",
            "One is most
grateful."
        ];
        var tip = tips[Math.
floor(Math.random() * tips.
length)]
        return tip+" "+input;
    }).
    then((input) => {
        // Translate to
another language
        logging( id, input,
step++ )
        const options = {
            from: "en",
            to: json.
targetLocale  ? json.
targetLocale  : "de"
        };
        return translate.
translate(input, options);
    }).
    then((input) => {
        // Text to speech to
synthesize WAV file
        logging( id, input,
step++ )
        const ttsParams = {
            input: {
                text: input
[0]
            },
            voice: {

languageCode: json.
targetLocale  ? json.
targetLocale  : "de",
                ssmlGender:
json.gender  ? json.gender
: 'FEMALE'
            },
            audioConfig: {

audioEncoding: 'LINEAR16'
            }
        }
        return textToSpeech.
synthesizeSpeech(ttsParams);
    }).
    then((input) => {
        // Save to the bucket
```

```
        logging( id, input
[0].audioContent.length,
step++ )
        return storage.bucket
(json.bucketout).file
("AUDIO_"+json.file+".wav").
save(input[0].audioContent);
    }).
    then((input) => {
        // Return cleanly
        logging( id, input,
step++ )
        if (res && res.
status) // if http call, we
should return 200
            res.status(200).
send("SUCCESS");
        return;
    })
}


findJson = (json) => {
 if (json && json.body) //
when called via google
Testing tab
    json = json.body
  else if (json && json.
query) // when called via
curl, json is in query
    json = json.query
  else  {
    // When called via
events, we need to juggle
the json
    json.bucketin=json.
bucket;
    json.file=json.name;
    json.bucketout="
gcfoutput"
  }

  console.log("findJson: "+
JSON.stringify(json))

  return json
}

logging = (id, input, step)
=> {
    console.log(  `id ${id}
Step ${step} : ${JSON.
stringify(input)}` )
```

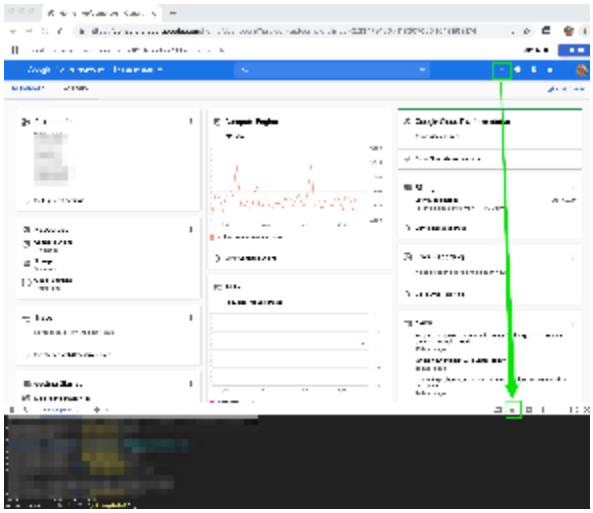
```
}
```

Prepare your Google Cloud Platform

Create and activate a Google Cloud Account

Create an account on Google Cloud Platform and activate it.

Create a new google project in your Google Cloud Platform, activate the cloud shell, and launch the code editor:



From in your google cloud shell, clone the Demo code:

- cloudshell> git clone https://github.com/kenlomaxhybris/gcfexamples.git

Review important NodeJS concepts

You need to know some basic NodeJS features before writing GCFs, including

- JS Functions
- JS Node Package Manager and package.json
- JS Modules:
  - Built-in Modules
  - External Modules
  - Exporting Modules
- NodeJS Eventloop
- Promises and Promise Chaining

Write a NodeJS Function

NodeJS Tutorial Series:

- cloudshell> cd gcfexamples/node101

Note that your cloud shell already has node installed..

- cloudshell> node -v

Executing the most basic JS Code:

js0.js

```
console.log(`Greetings world!`);
```

- cloudshell> node js0

Execute a JS function:

js1.js

```
function helloNodeJS( name ) {
        console.log(`Greetings ${name} from
NodeJS!`); // Note the useful string place
holders
}
helloNodeJS("Theresa May")
```

- cloudshell> node js1

NodeJS Built in Modules

NodeJS Built-in Modules:

https://www.w3schools.com/NodeJS/ref_modules.asp

Node has a selection of Modules (think Java utility classes) that you can include (with the "require" keyword) and then use.

For example you can easily create a web server with the **http** and **url** modules.

js2.js

```
var http = require('http');
var url = require('url');
http.createServer(function (req, res) {
  var parts = url.parse(req.url, true);
  var query = parts.query;
  res.writeHead(200, {'Content-Type': 'text
/html'});
  res.end(`Greetings ${query.name} from NodeJS!
`);
}).listen(8080);
console.log("Server is up and running. Try
localhost:8080?name=Boris%20Johnson")
```

Execute the NodeJS Function:

- cloudshell> node js2

Call the webserver:

- cloudshell> curl http://localhost:8080?name=Boris%20Johnson

NodeJS External Modules

The Express JS Module:

Express

There is a huge number of external modules too. You use the npm package manager to install these. Npm expects to find a package.json file in the root of your project, which it will use to track the modules you download.

Create a package.json for npm to use:

- cloudshell> vi package.json

package.json

```
{
}
```

Tell npm to download the module "express":

- cloudshell> npm install express

Note that npm has updated the package.json file with this information..

- cloudshell> cat package.json

.. and that npm has downloaded the express code into the folder node_modules:

- cloudshell> ls -la node_modules

Explore a webserver using the express module:

js3.js

```
var expressModule = require('express')
var expressWebServer = expressModule();

expressWebServer.get('/', function(req, res){
  res.send(`Greetings ${req.query.name} from
NodeJS!`);
});
expressWebServer.listen(8080);
console.log("Server is up and running. Try
localhost:8080?name=David%20Cameron")
```

- cloudshell> node js3
- cloudshell> curl http://localhost:8080?name=David%20Cameron

Export your own NodeJS Module

Each NodeJS file is treated as its own module. To make a function in a module visible to others, you need to "export" it.

Write a module that exports the function helloWorld:

js4.js

```
exports.helloWorld = (json, res) => {
  console.log("Hello "+ json.name)
  if (res && res.send)
    res.status(200).send("Hello "+ json.name);
}
```

Write a second module that imports the js4 module and then calls its function:

External NodeJS Modules:

NPM Modules

All about Express in 1 hour:

js5.js

```
var i = require("./js4")
i.helloWorld({"name":"bod"})
```

- cloudshell>node js5

You can also include a module and call its function directly from the node cli:

- cloudshell>node -e 'require("./js4").helloWorld({"name" : "Jakob Rees-Mogg" })'

The Event Loop

Good Event Loop Overview

NodeJS has **one main thread** - the "Event Loop" where all requests and ensuing business logic are processed.

Don't Block the Event Loop:

https://nodejs.org/en/docs/guides/dont-block-the-event-loop/

See how easy it is to kill NodeJS app by giving the eventloop a long task:

- cloudshell>vi eventloopexample.js

eventloopexample.js

```
var expressModule = require('express')
var expressWebServer = expressModule();

expressWebServer.get('/cpulight', function
(req, res){
  res.send('That was fast');
});

expressWebServer.get('/cpuheavy', function
(req, res){
  for (i = 0; i <= 1000; i++)
    for (j = 0; j <= 1000; j++)
      console.log( i*j );
  res.send('Greetings ');
});
console.log("Express server started. Try
http://localhost:8080/cpuheavy and
http://localhost:8080/cpulight")
expressWebServer.listen(8080);
```

Run this web server and access both end points:

- cloudshell> node eventloopexample
- cloudshell>curl http://localhost:8080/cpulight
- cloudshell>curl http://localhost:8080/cpuheavy

Note that the cpuheavy function kills the entire application:O

NodeJS Promises

Good Overview of Promises and Callbacks

You can help the Eventloop a lot by **breaking your business code in to small asynchronous blocks called promises**. These promises are called on a lower priority to the incoming requests:

- cloudshell> vi promise1.js

promise1.js

```
onst util = require('util');
const fs = require('fs');
const readFile = util.promisify(fs.readFile)
var ret = readFile("./bod.txt")
console.log("Ret :"+ret)
```

- cloudshell> node promise1

Note this returns "Promise { <pending> }"

Promise Chains

You handle promises using **promise chains** and the **then** keyword.. When a promise completes, it passes the result to the following "then" block:

- cloudshell> vi promise2.js

promise2.js

```
const util = require('util');
const fs = require('fs');
const readFile = util.promisify(fs.readFile)
var ret = readFile("./bod.txt").
// When the readFile promise completes, it
passes the result to the next "then" block..
then((arg) => {
    console.log("===== Then Block A output:
"+arg)
    return "Done"
}).
// When the above "then" block completes (be
that plain code, or also a promise), it passes
the result to the next "then" block..
then((arg) => {
    console.log("===== Then Block B: "+arg)
}).
// When the above "then" block completes (be
that plain code, or also a promise), it passes
the result to the next "then" block..
then((arg) => {
    console.log("===== Then Block C: "+arg)
})
console.log("===== Ret :" + ret)
```

- cloudshell> node promise2

Debugging Node JS Locally

You can debug NodeJS locally:

- terminal> git clone https://github.com/kenlomaxhybris/gcfexamples.git ; cd gcfexamples/node101
- terminal> node --inspect index.js

Open the Chrome debugger and set some break points:

- chrome> chrome://inspect

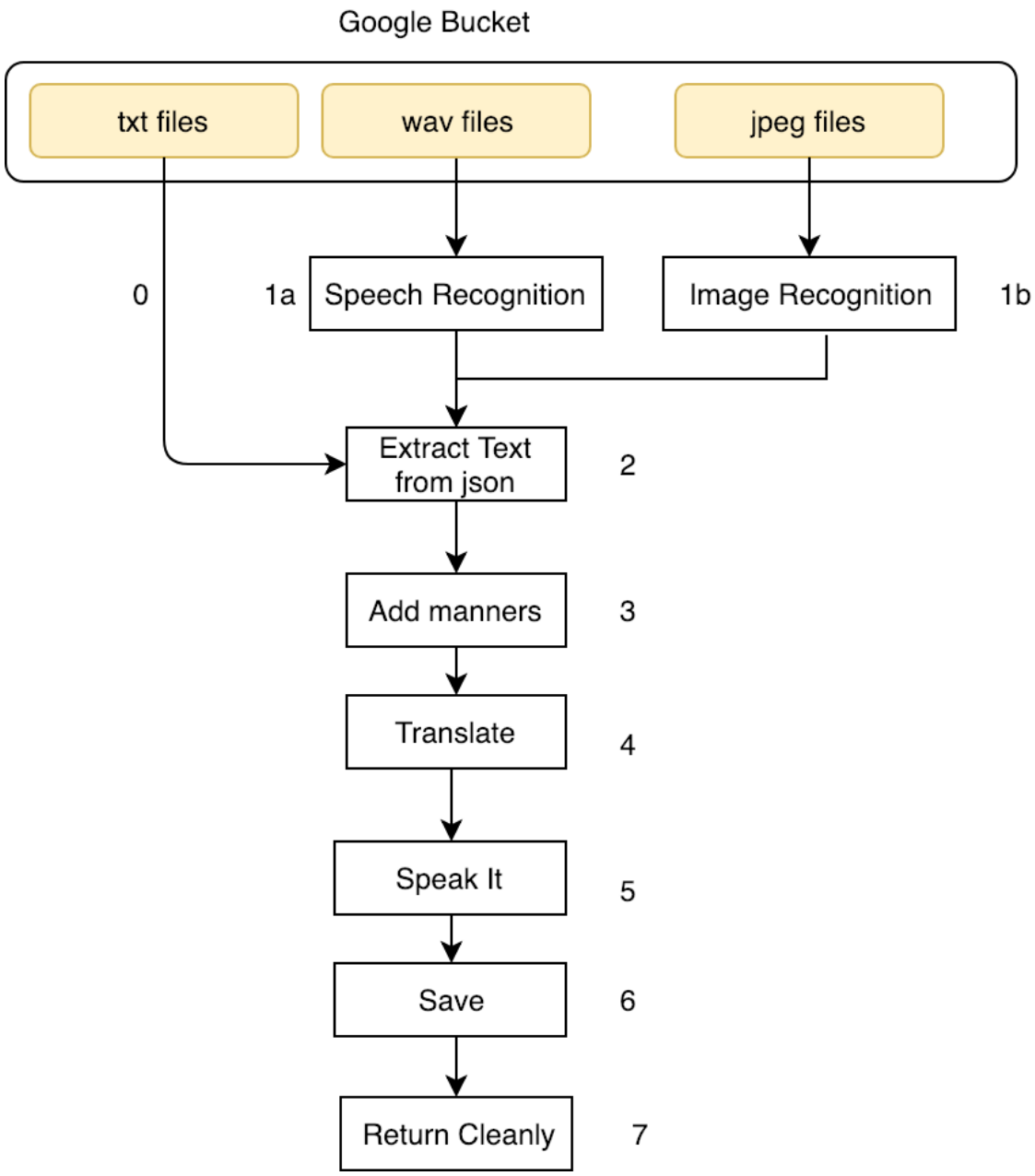
Invoke your function:

- terminal> curl "http://localhost:8080?name=TheresaMay"

Using Google APIs in your Cloud Functions

To explore some cool Google APIs, we will write a google cloud function "gcfs" that:

- reads a text file from a google storage bucket,
- add manners to the text
- translates the text to another language
- speak the resulting text
- for **wav** files, it will first perform speech recognition
- for **jpeg** filles, it will first do text recognition



We will call this function locally to debug it

We will then deploy to Google to call it directly and via bucket-upload event

The completed GCF is at https://github.com/kenlomaxhybris/gcfexamples/blob/master/gcfexample/index.js.

Create the Google Storage Buckets we will use

Create google buckets gcfinputbucket and gcfoutputbucket

- cloudshell> gsutil mb gs://gcfinputbucket
- cloudshell> gsutil mb gs://gcfoutputbucket

and upload some test data to the gcfinputbucket bucket:

- googleshell> gsutil cp material/* gs://gcfinputbucket

Confirm you see the buckets in GCP

- Chrome> https://console.cloud.google.com/storage/browser

Enable the APIs we need in GCP:

- Chrome> https://console.developers.google.com/apis/api/translate.googleapis.com
- Chrome> https://console.developers.google.com/apis/api/texttospeech.googleapis.com
- Chrome> https://console.developers.google.com/apis/api/vision.googleapis.com
- Chrome> https://console.developers.google.com/apis/api/speech.googleapis.com

Authorize ourselves

To execute google APIs you need to have "permission", so you need to generate a service account key.

- Chrome> https://console.cloud.google.com/iam-admin/serviceaccounts

Download the json file and set your GOOGLE_APPLICATION_CREDENTIALS to it:

- terminal> export GOOGLE_APPLICATION_CREDENTIALS=pathToJSONAPIKey

Run and debug the google function locally

- terminal> git clone https://github.com/kenlomaxhybris/gcfexamples.git
- terminal> cd gcfexamples/gcfexample
- terminal> npm install @google-cloud/storage @google-cloud/vision @google-cloud /speech @google-cloud/translate @google-cloud/text-to-speech

Start the local debugger:

- terminal> node --inspect debugging.js
- chrome> chrome://inspect

Invoke our functions:

- terminal> curl "http://localhost:8080?name=TheresaMay"
- terminal> curl "http://localhost:8080/gcfs?bucketin=gcfinputbucket&bucketout=gcfoutputbucket&targetLocale=fr&file=input.txt&gender=MALE&sourceLocale=en-US"

Deploy to Google and run from there

Open your Google Cloud Functions portal and deploy your index.js and package.json to a new function

Then explore the four tabs: General, Trugger, Source and Testing.

Try invoking in the Testing Tab with {"bucketin":"gcfinputbucket", "bucketout":"gcfoutputbucket","targetLocale":"fr", "file":"input.txt"}

Turn your function into a background function that is triggered on upload events

There are two types of Google Functions:

- **HTTP Functions**: These HTTP requests wait for the response and support handling of common HTTP request methods like GET, PUT, POST, DELETE and OPTIONS.
- **Background Functions**: to handle events from your Cloud infrastructure, such as messages on a Cloud Pub/Sub topic, or changes in a Cloud Storage bucket.

Google Events and triggers: https://cloud.google.com/functions/docs/concepts/events-triggers

Different ways to call a cloud function:

https://cloud.google.com/functions/docs/calling/

Redeploy the gcfs function as a background function to be triggered by file upload.

Delete the input* files from the bucket, then re-copy them into the bucket. The function should be triggered 3 times - once for each file-upload:

- cloudshell> gsutil cp input.* gs://gcfinputbucket

Confirm you see the resulting 3 wav files can be found in the google bucket.

Debugging with StackDriver

Uncomment the debug-agent line at the top of index.js and redeploy function

Go to Stack Driver Console (https://console.cloud.google.com/debug)

Select the code under "Upload a source code capture to Google servers!" and checkin your code

- googleshell>gcloud beta debug source upload --project=xxxxx --branch=xxxxxx .

Return to "Upload a source code capture to Google servers!" in the Stack Driver Console and select "Select Source"

Set snapshot points

Invoke the function and notice how your Snapshots have been captured, without stopping your "production code" :)

That's it 

You now have first-hand experience of

- NodeJS
- Debugging and Deploying Functions to GCP
- Using Google Cloud APIs to enhance your own logic

Keep your eye out for more Cloud Labs.

Find out more @ https://cloud.google.com/functions/docs/