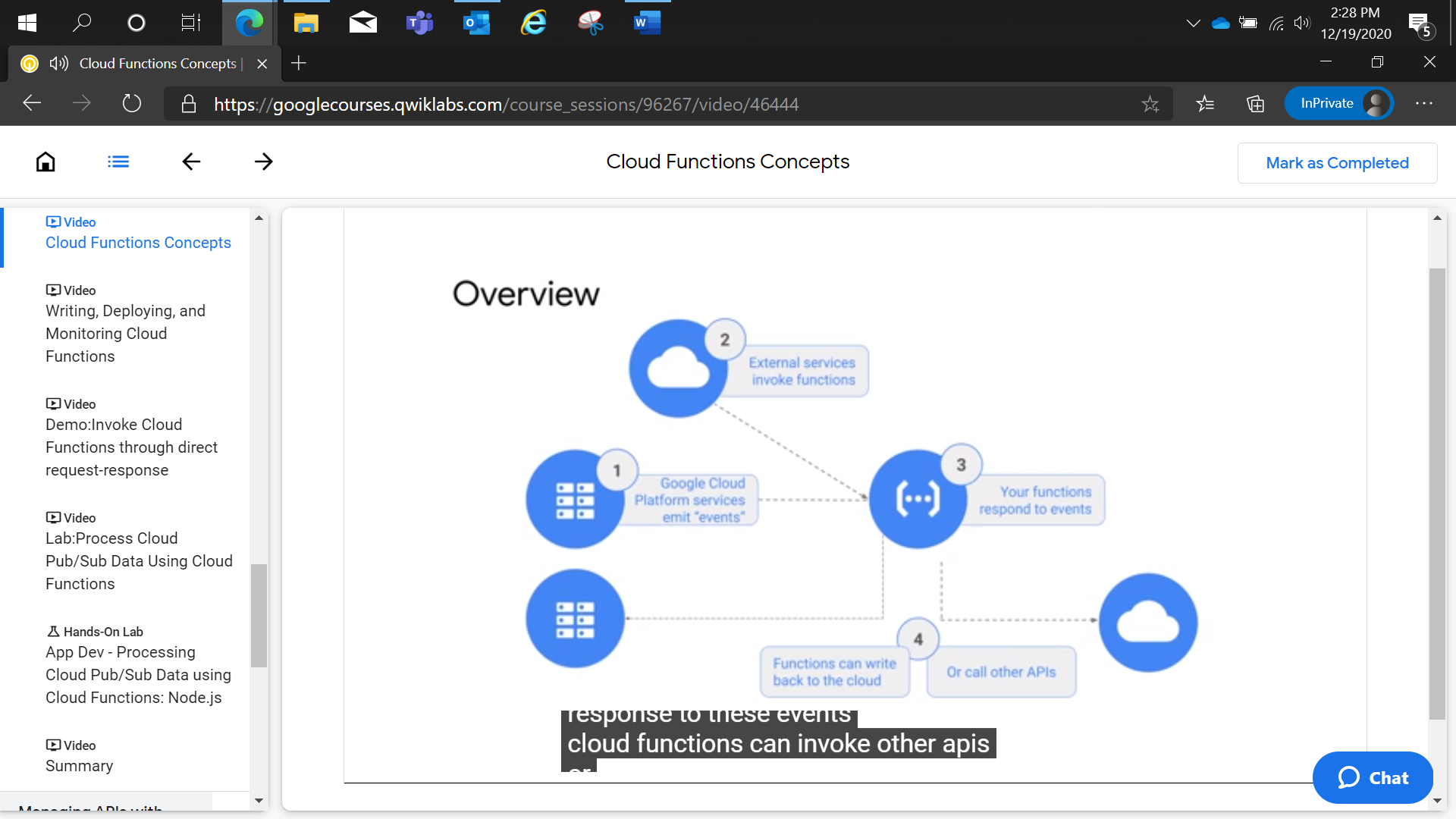
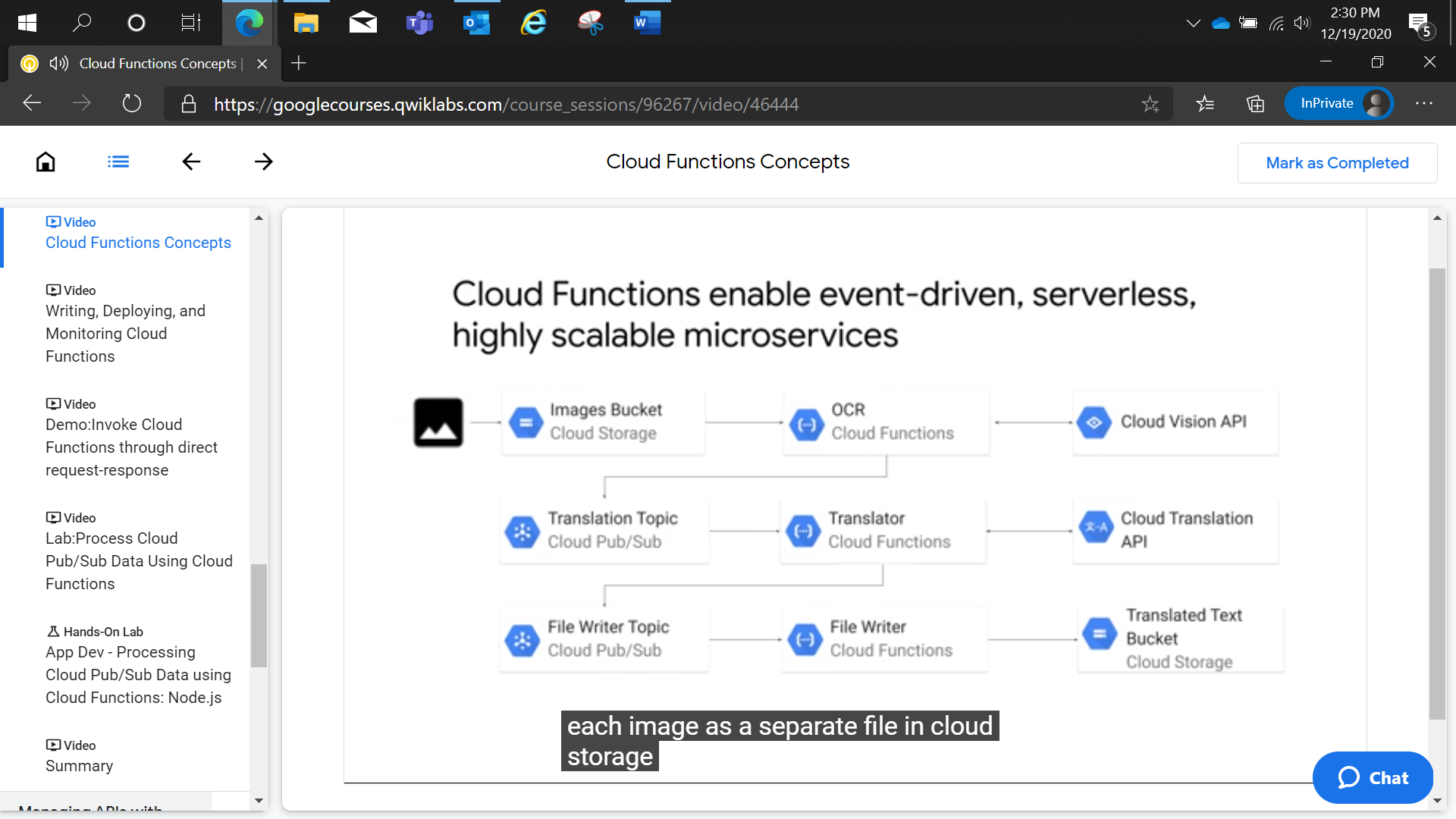
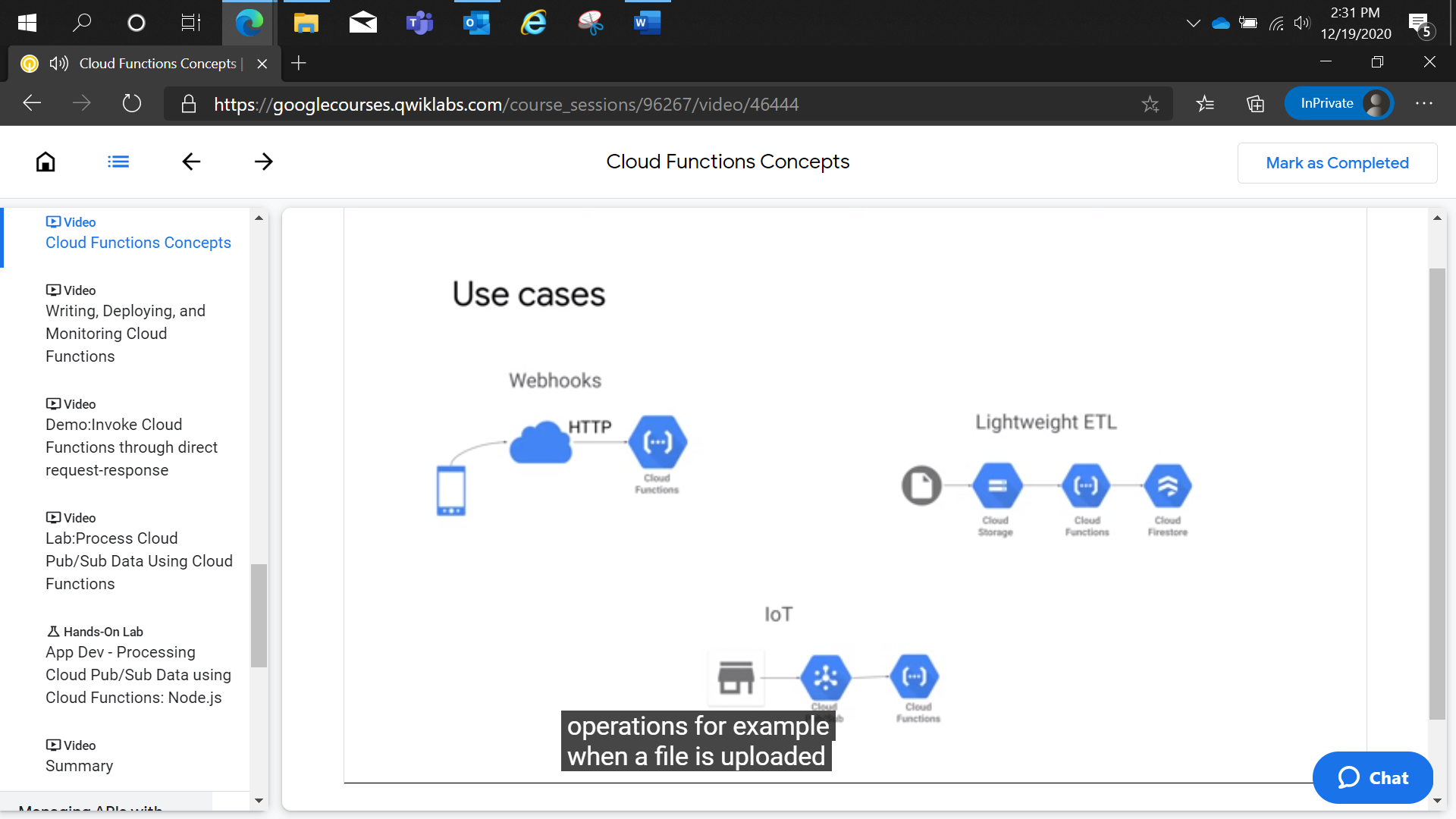
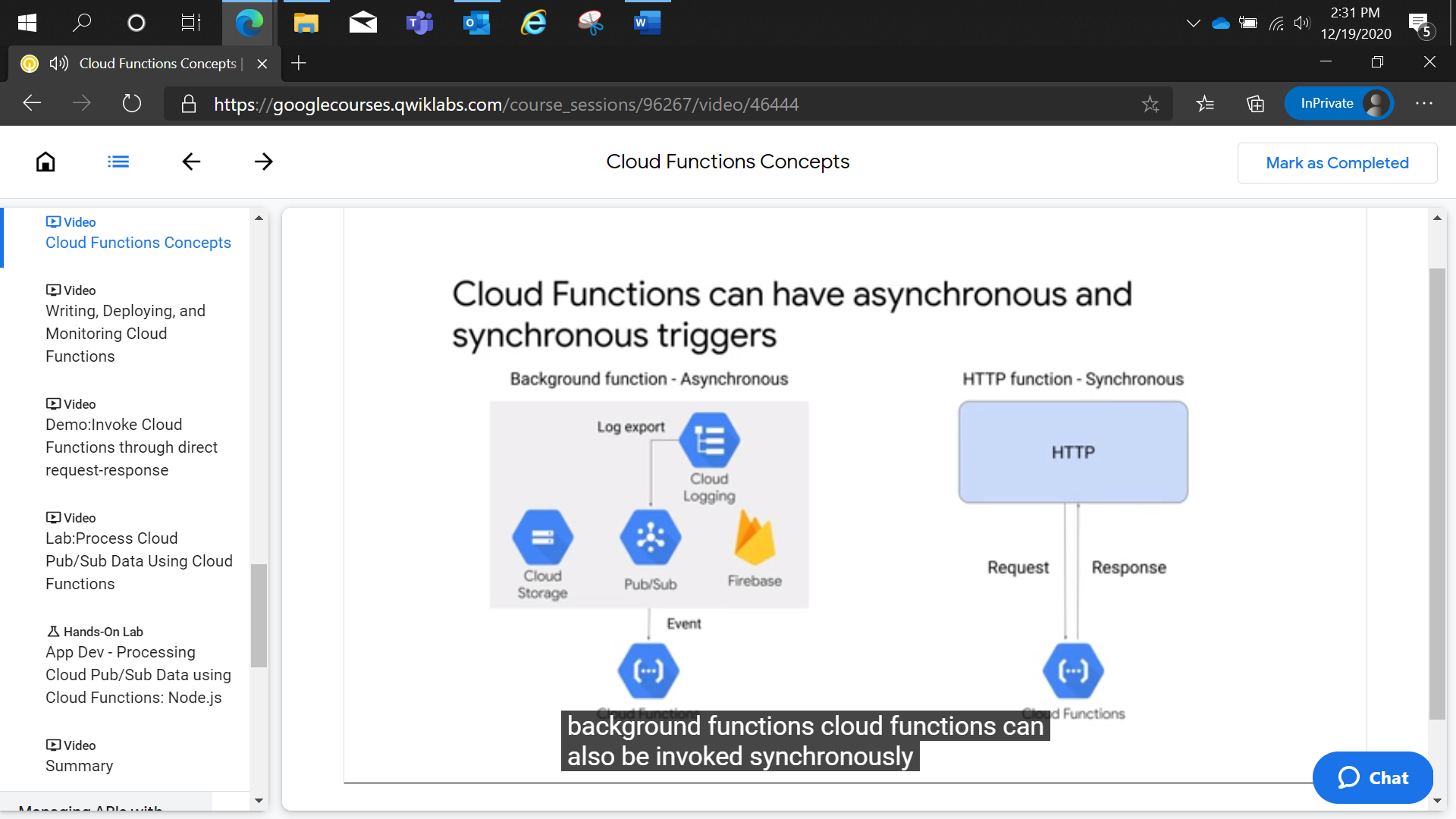
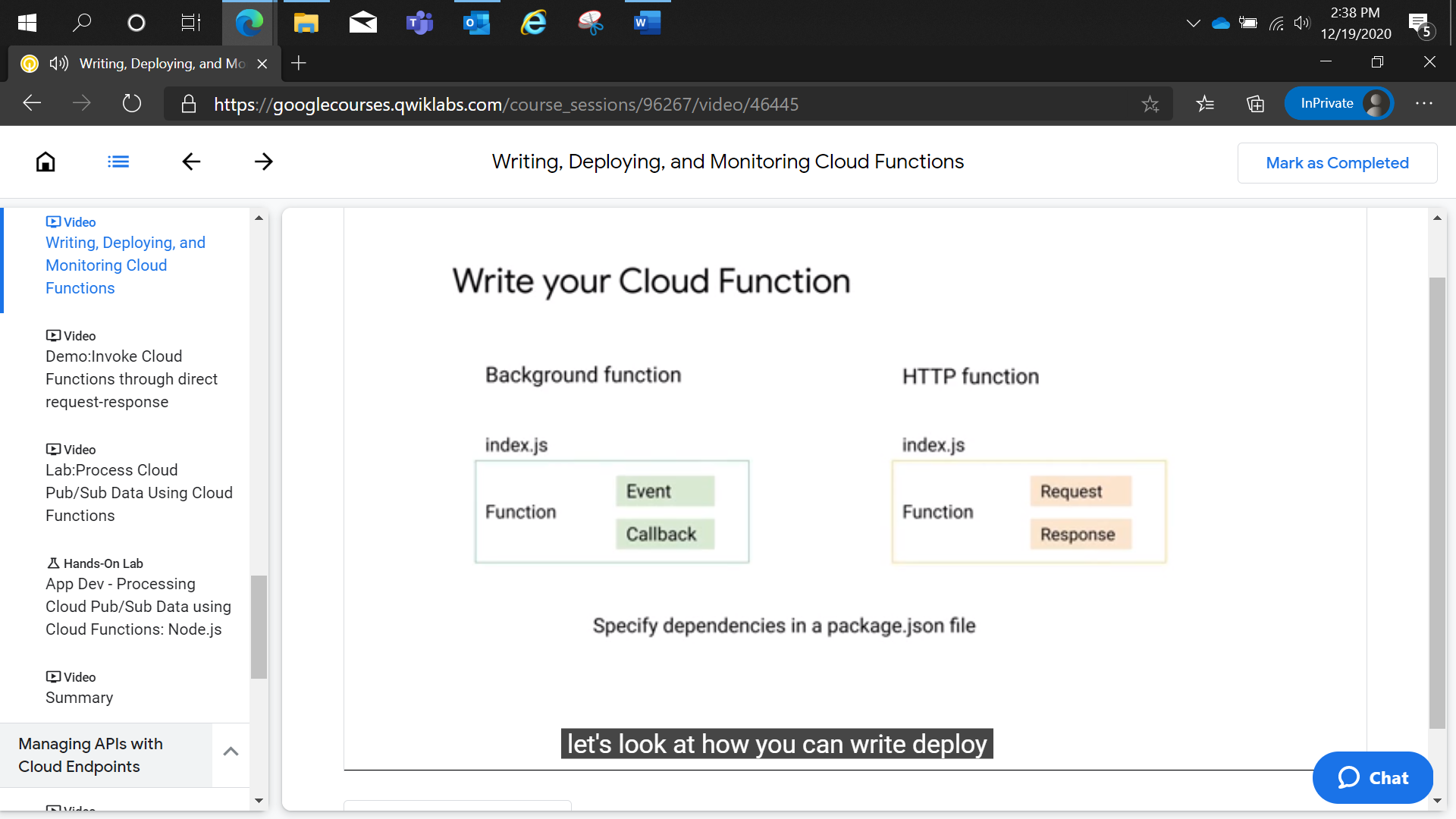
Chapter 1.3.3 Cloud Functions for event driven Processing

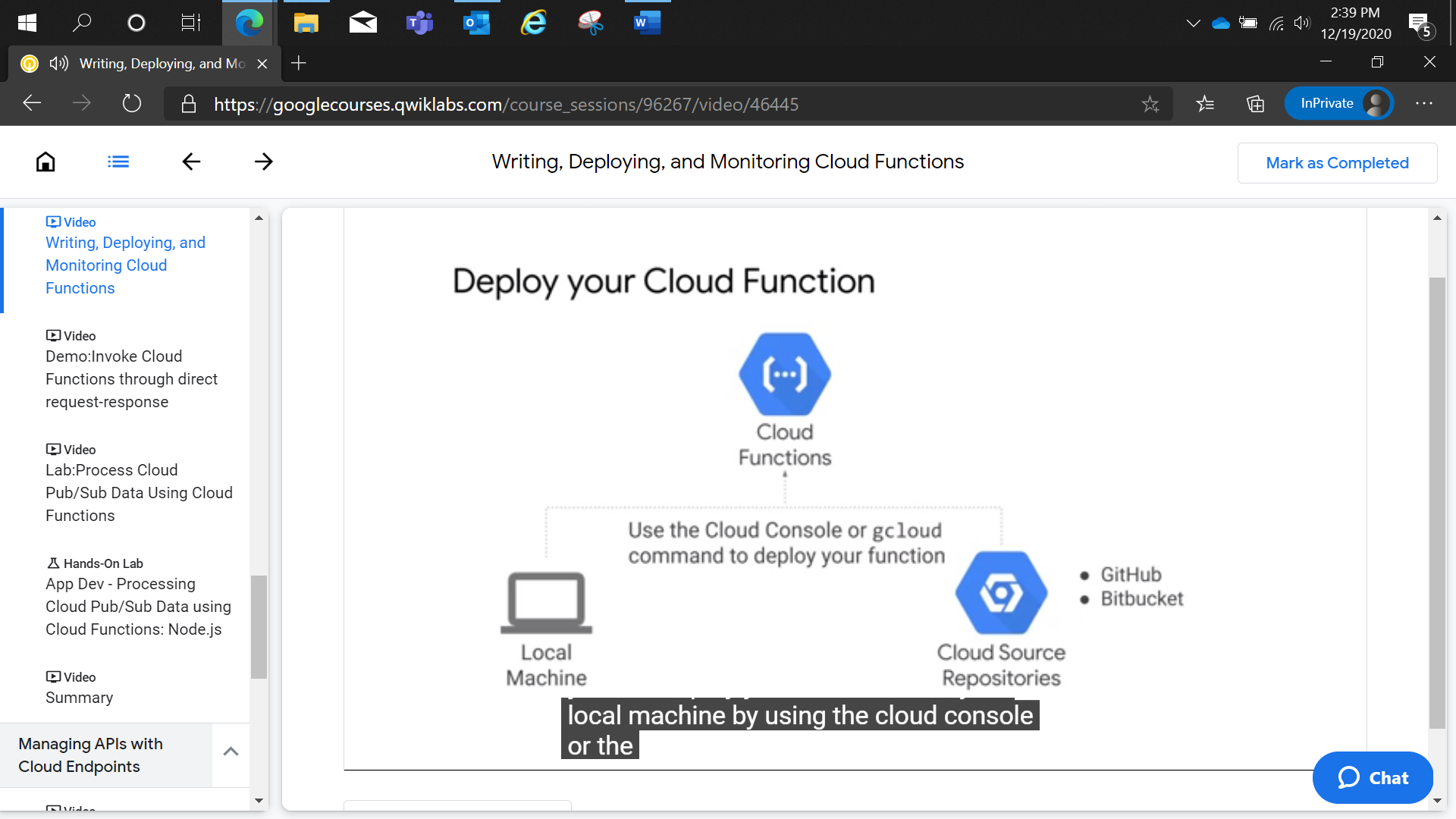


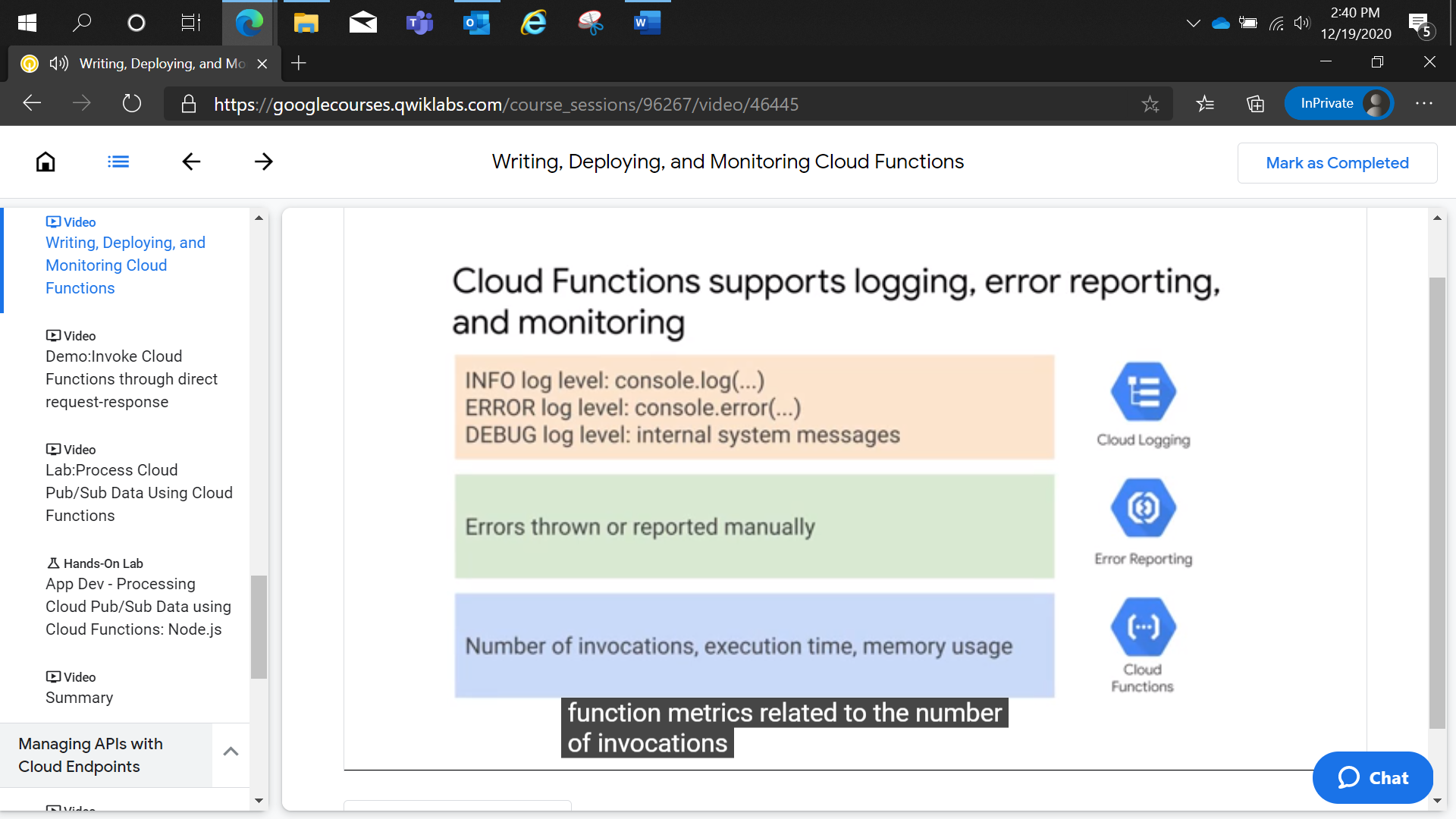


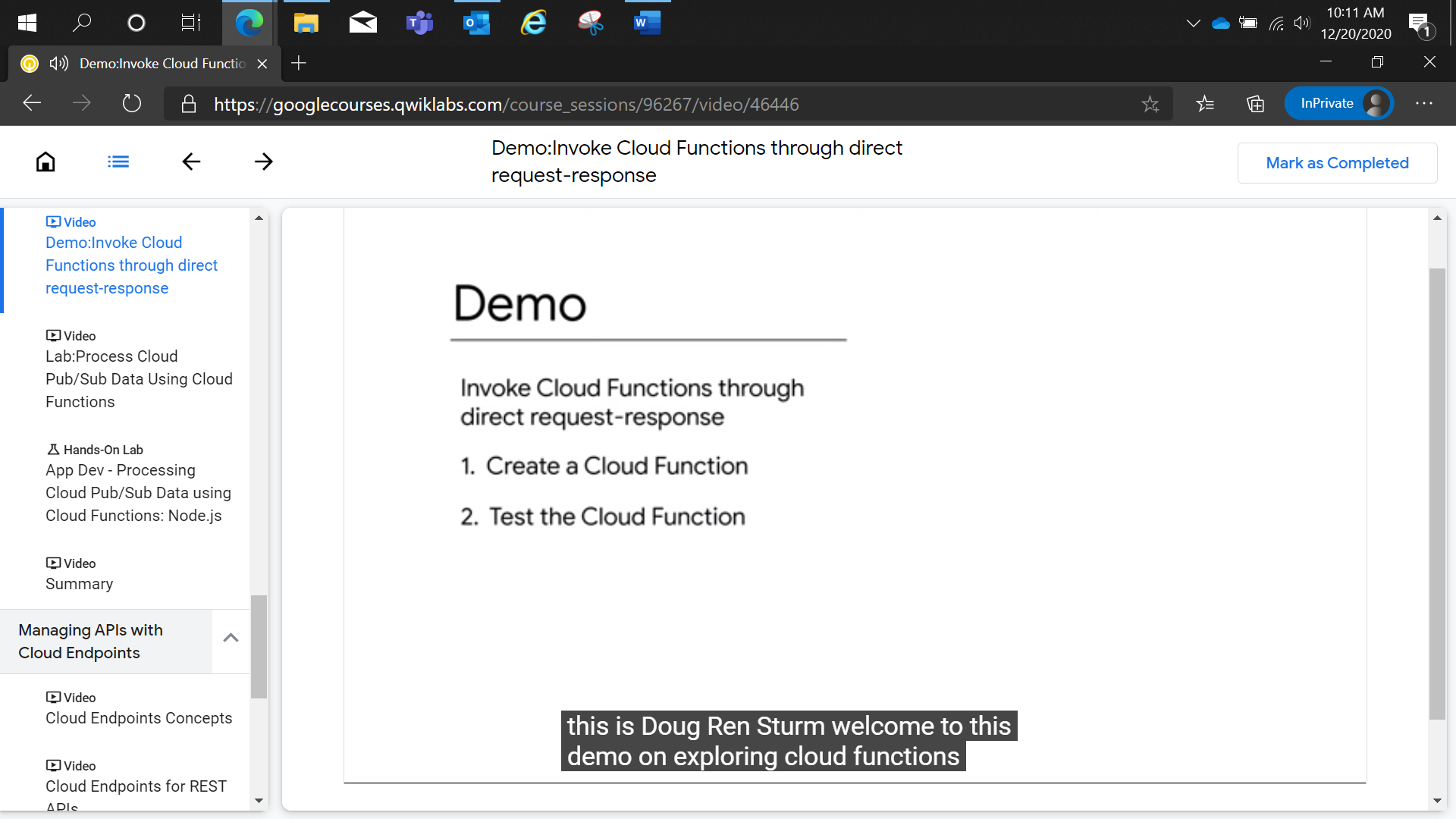


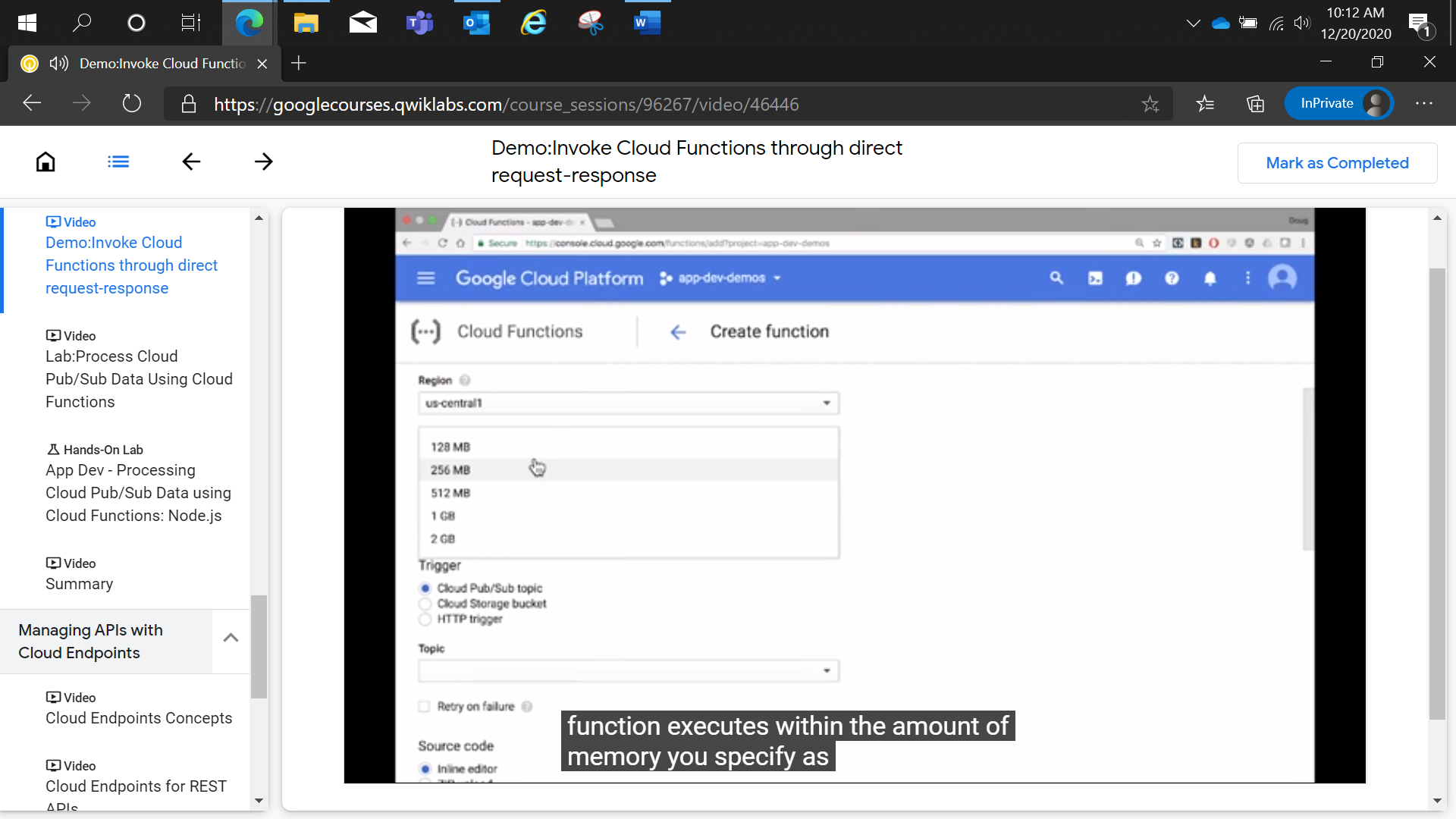


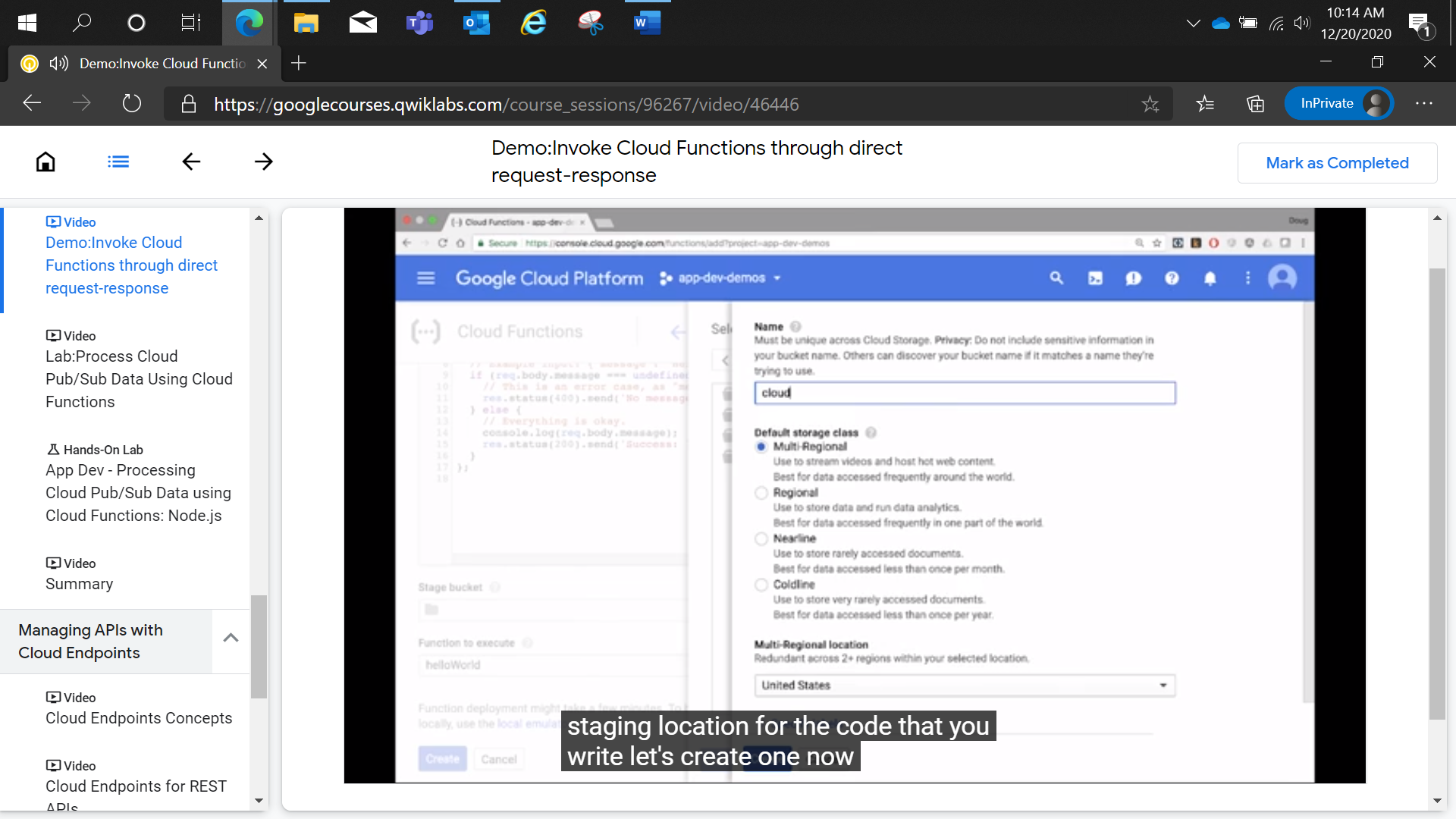
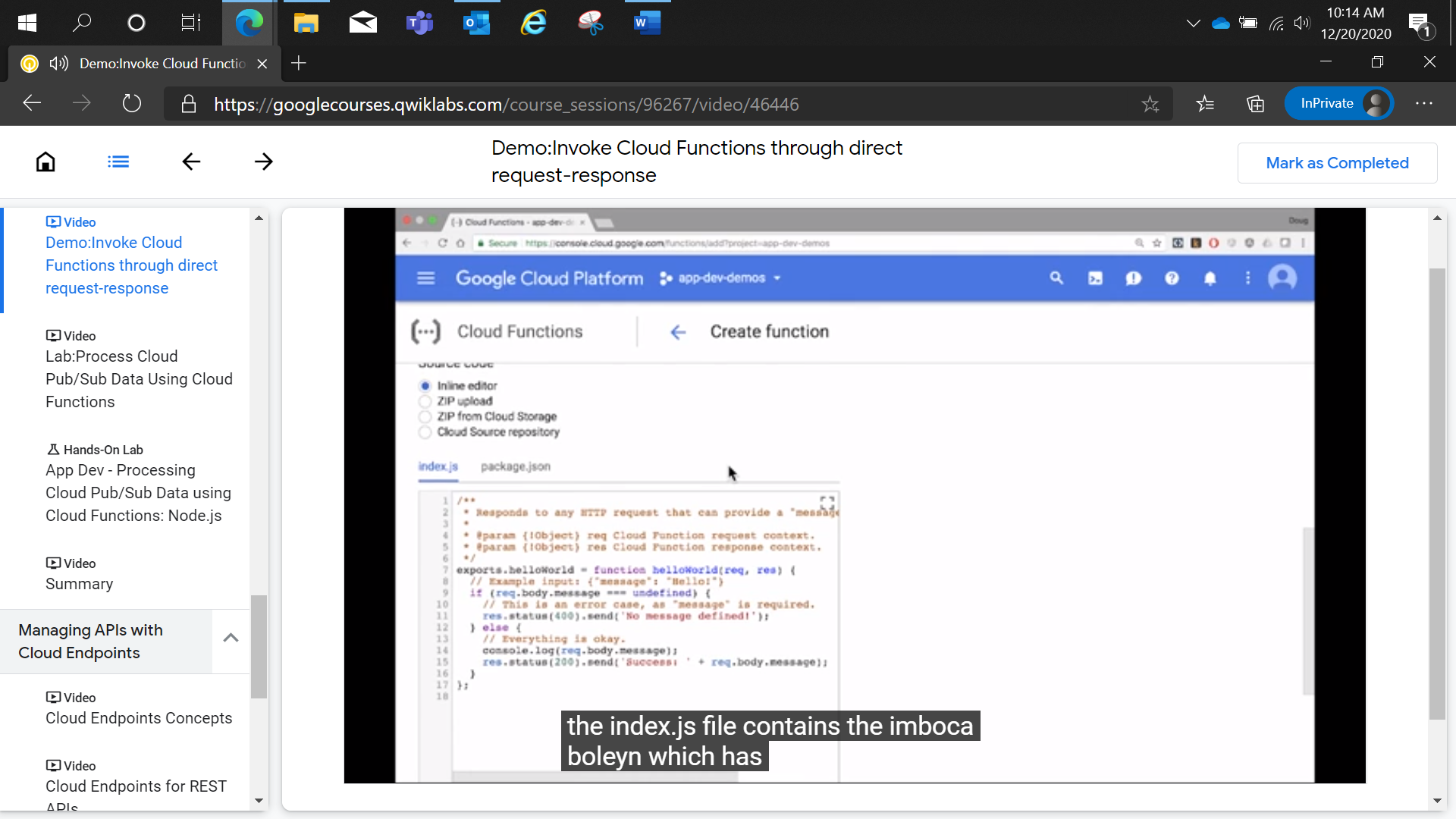


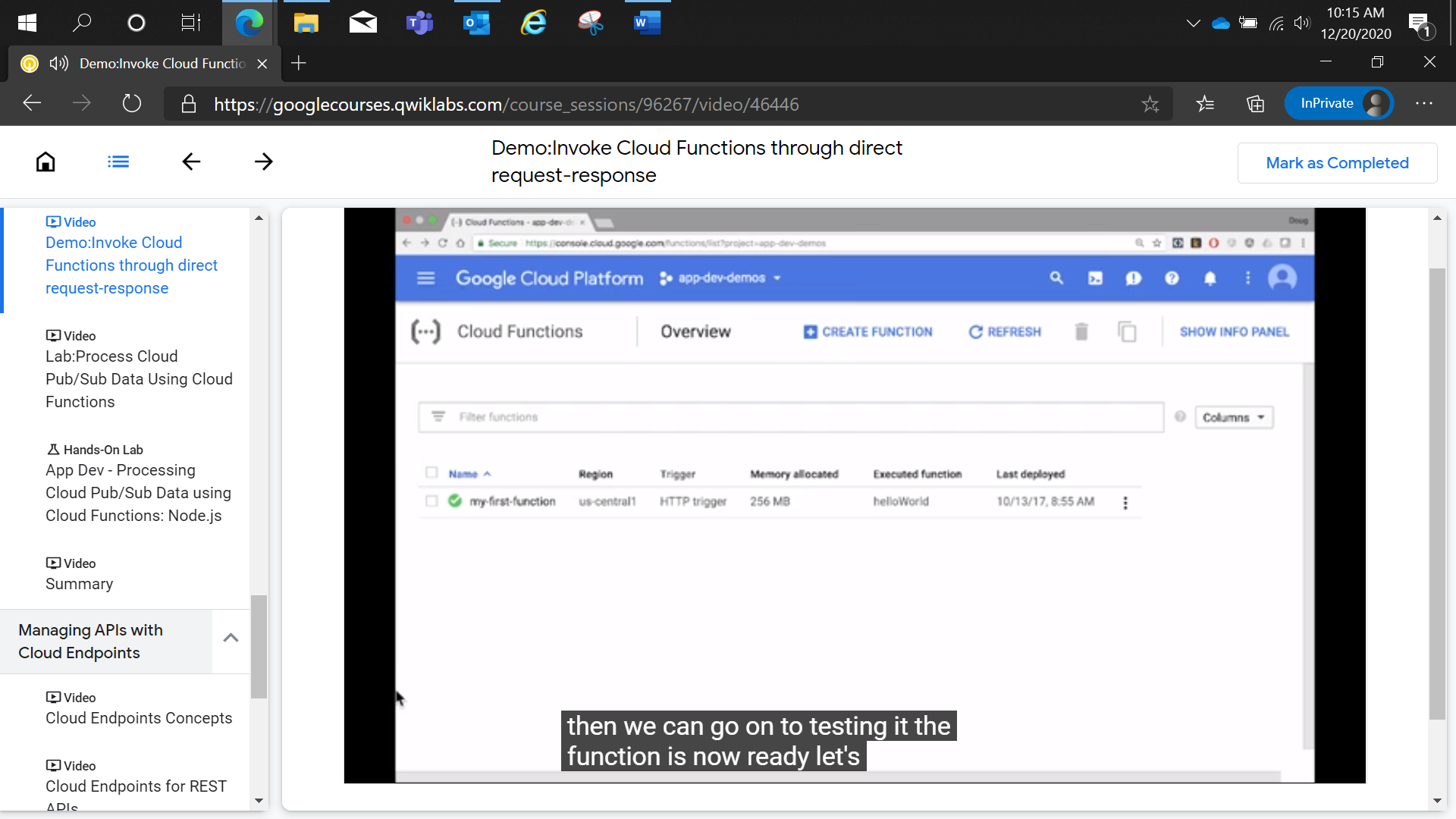


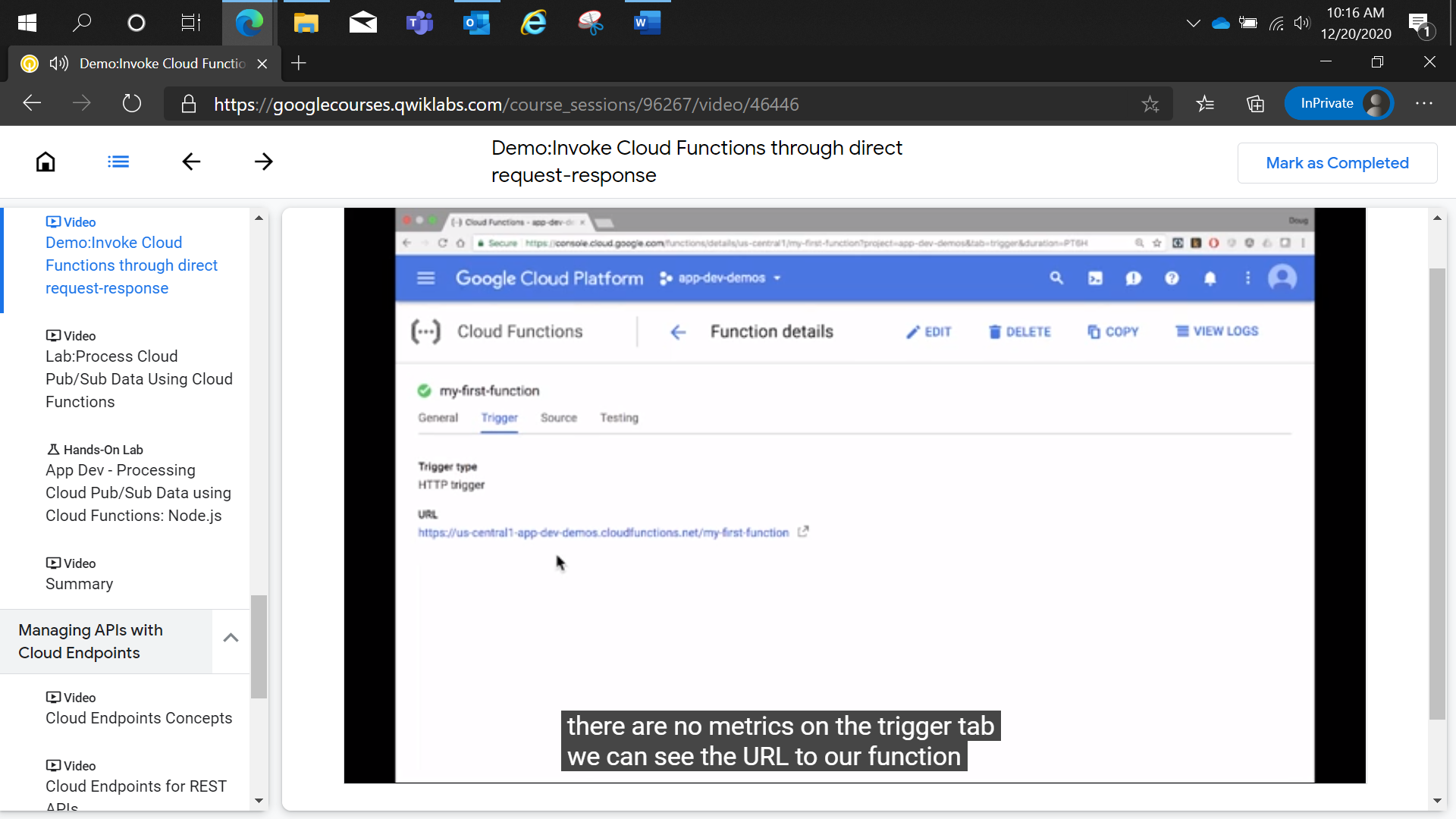


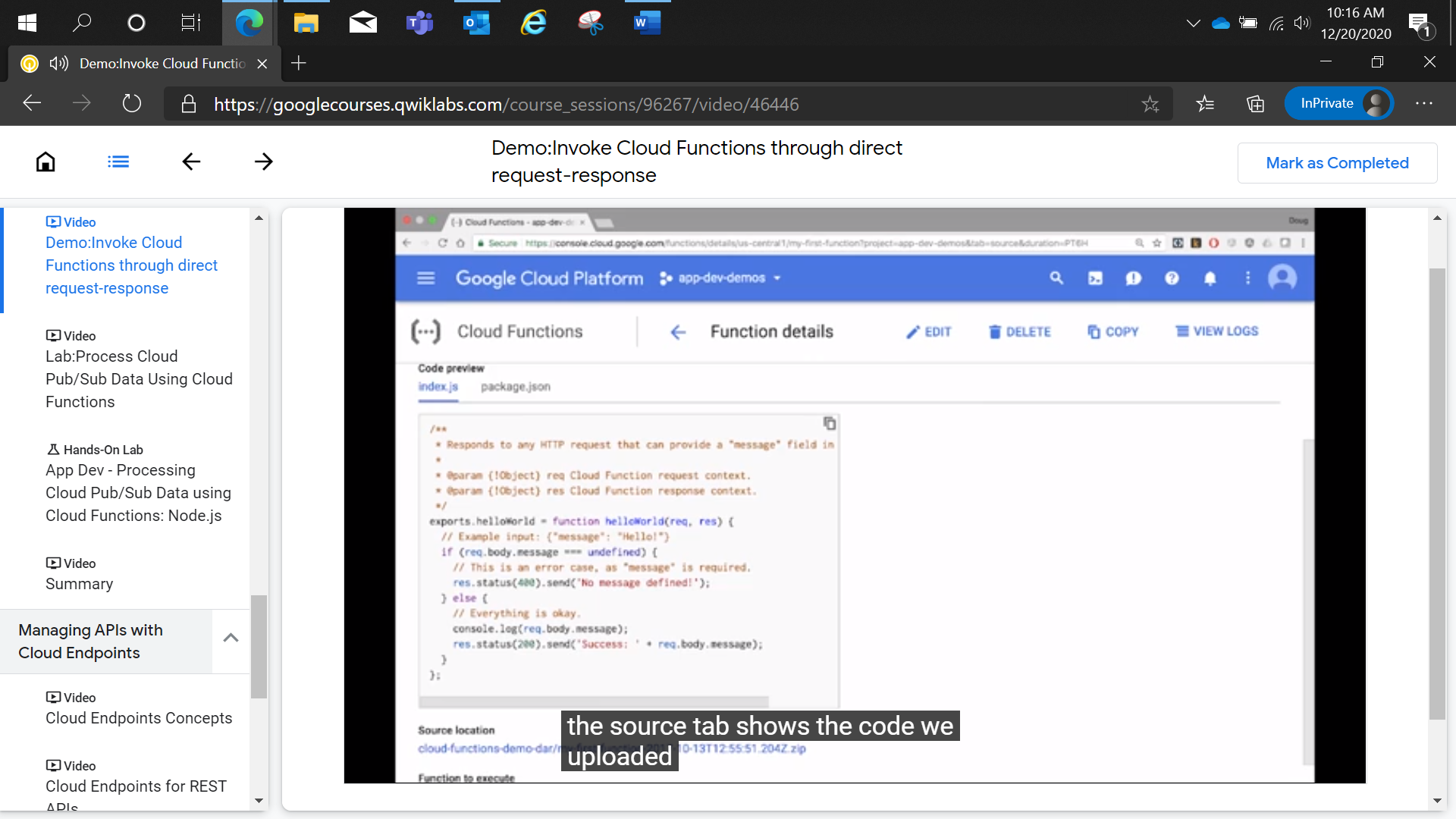


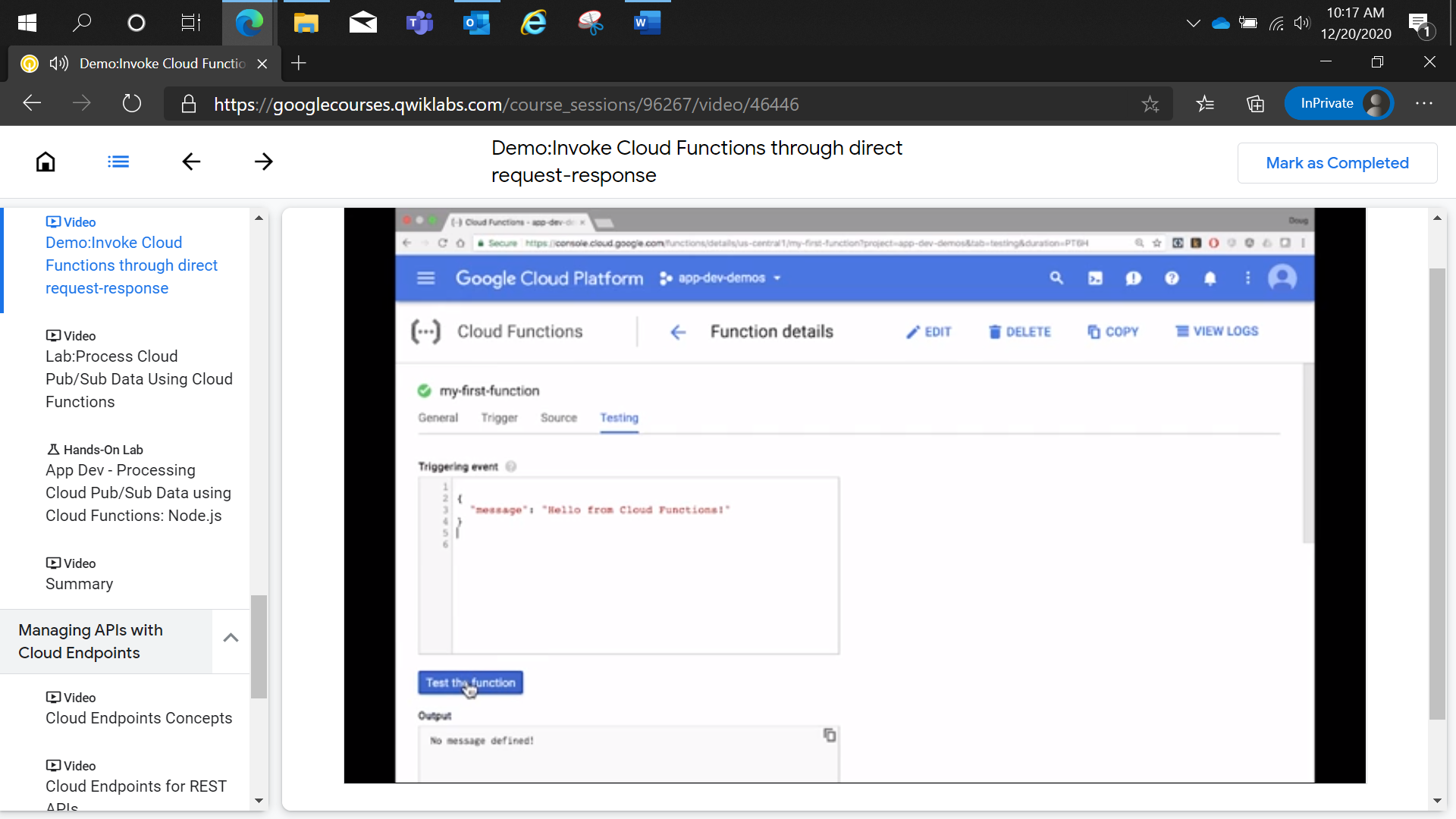


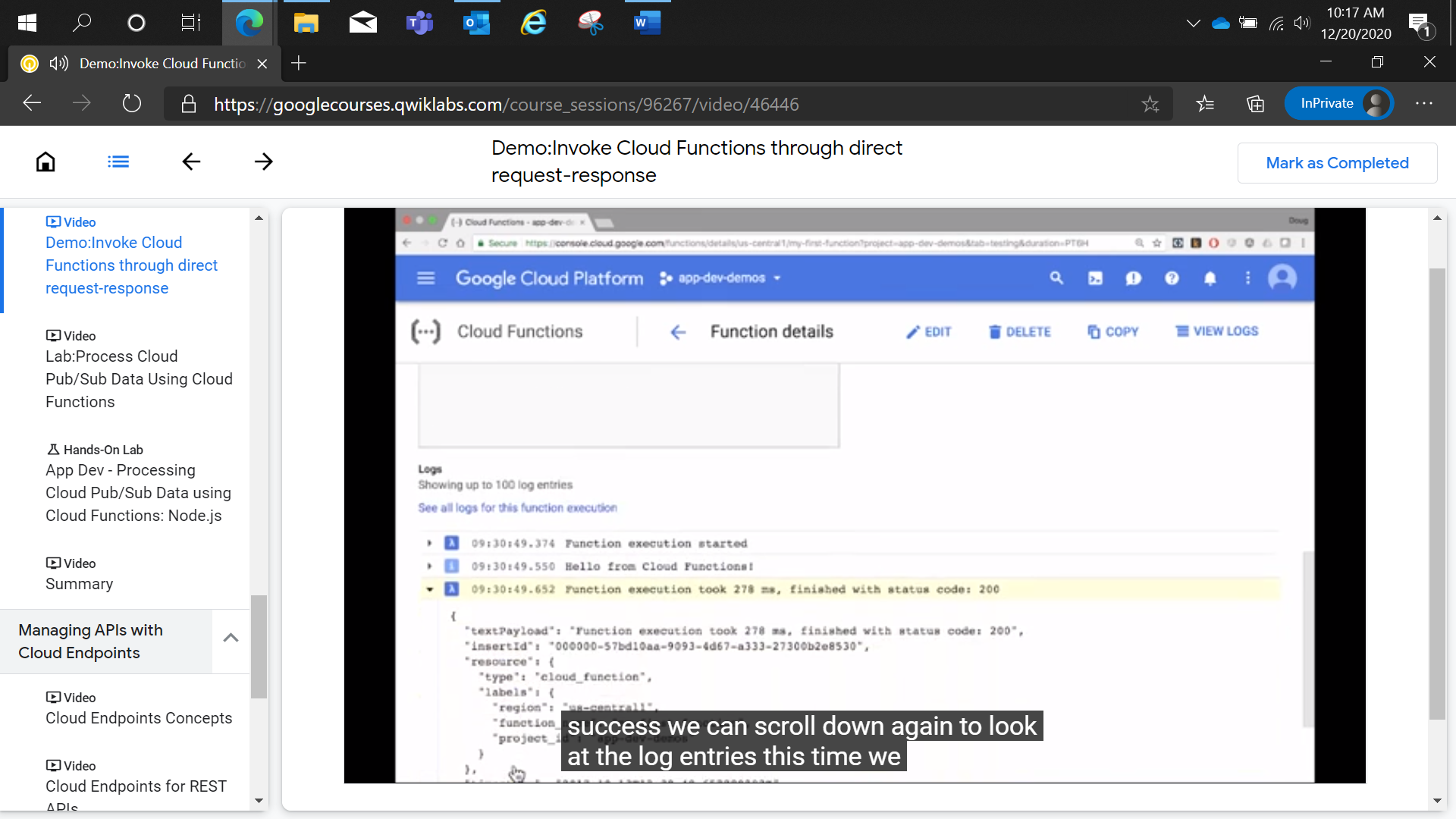


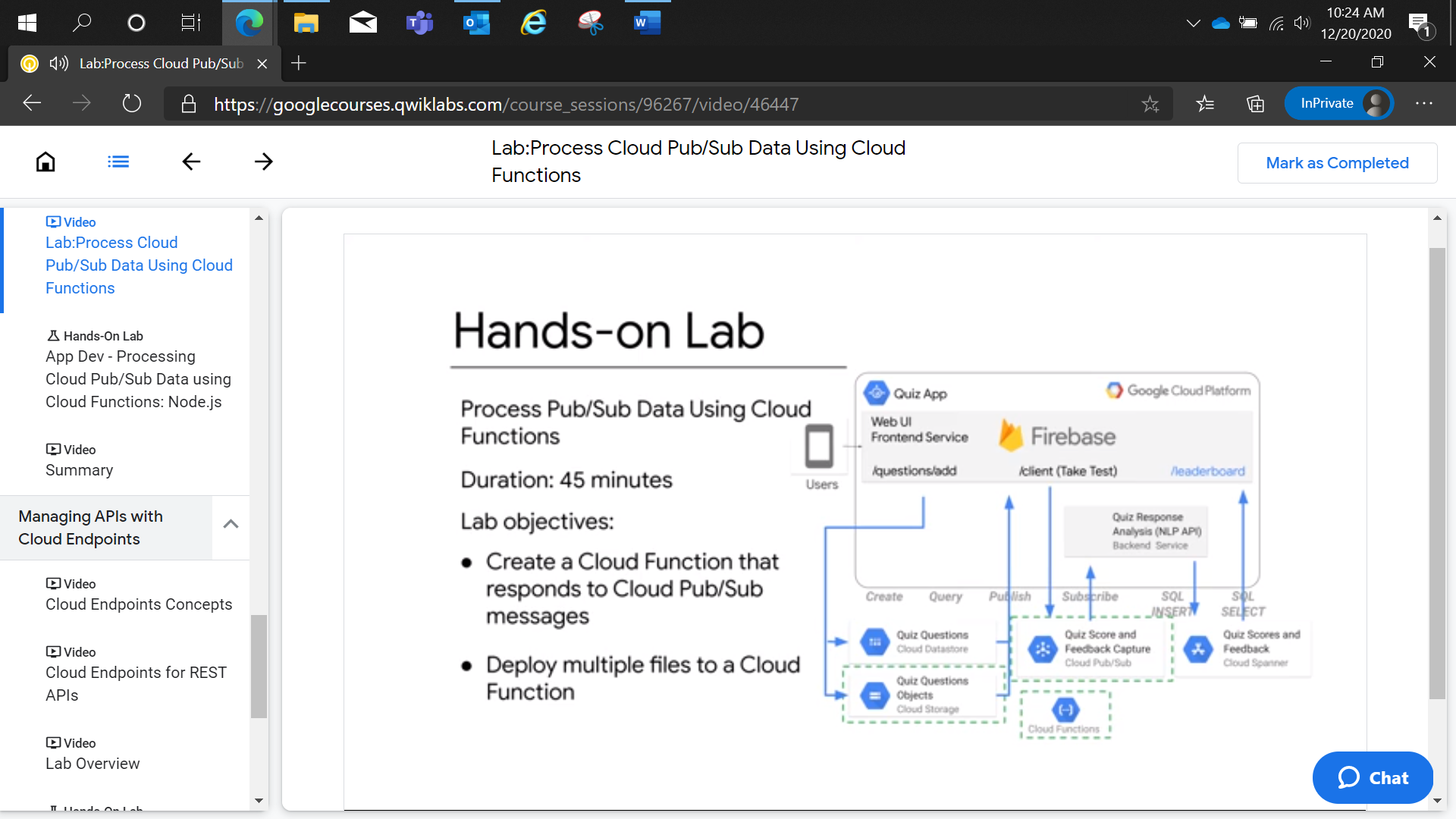


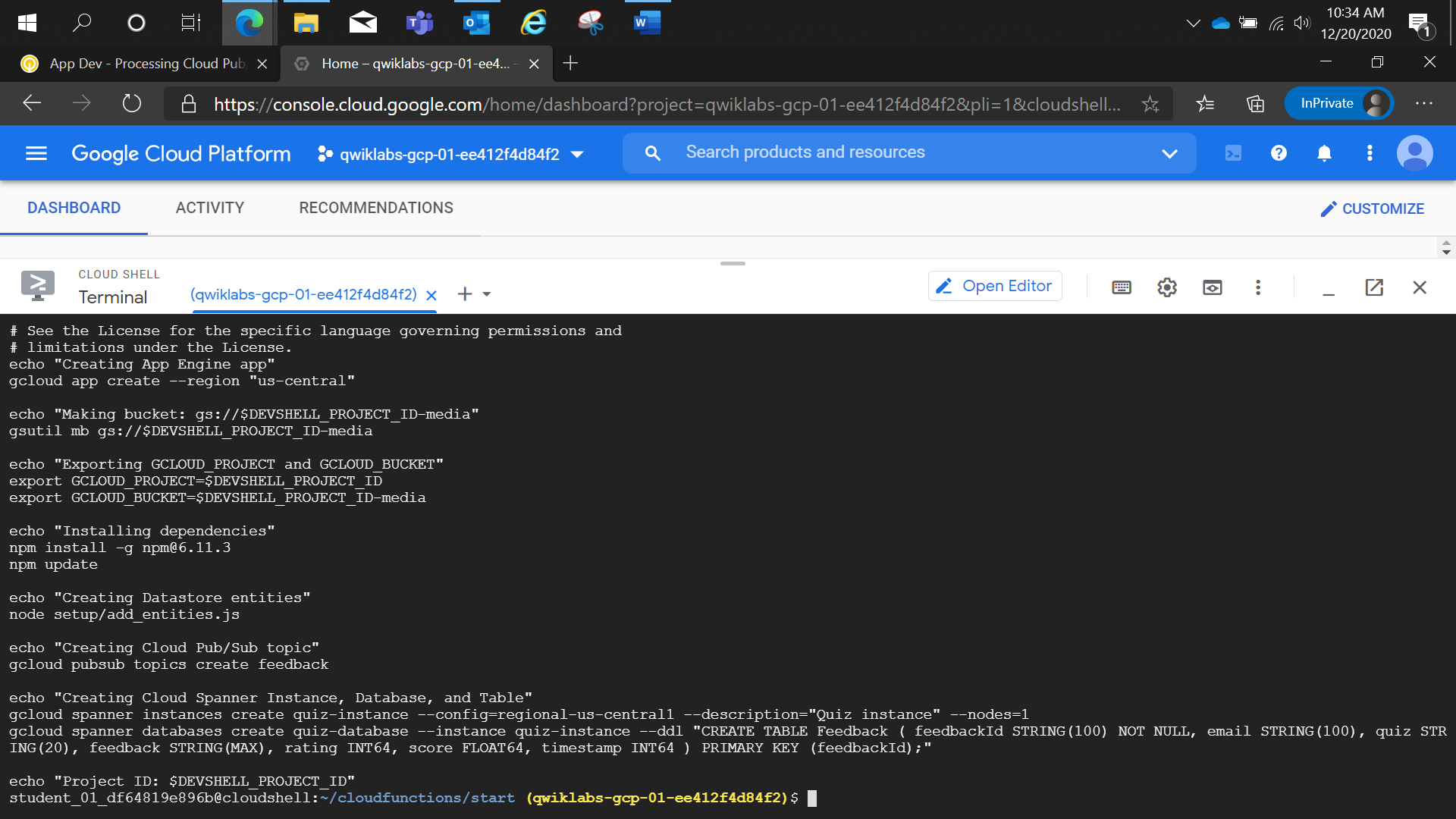












App Dev - Processing Cloud Pub/Sub Data using Cloud Functions: Node.js

2 hoursFree

Rate Lab

**Overview**

In this lab, you enhance the online Quiz application by creating a Cloud Function to process Cloud Pub/Sub messages.

This process harnesses several GCP products in a serverless environment: Cloud Pub/Sub, Cloud Natural Language API, and Cloud Spanner.

**Objectives**

In this lab, you learn how to perform the following tasks:

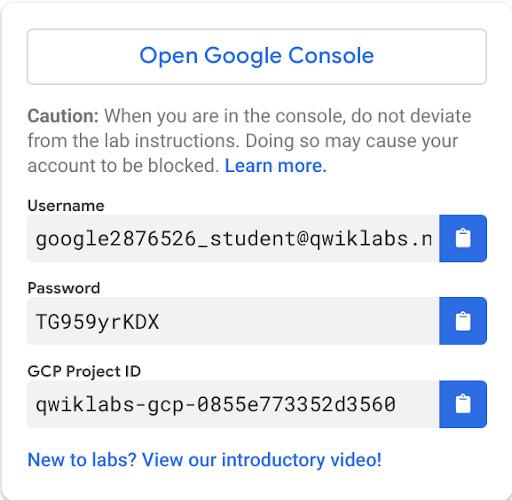
* Create a Cloud Function that responds to Cloud Pub/Sub messages.
* Deploy multiple files to a Cloud Function.

**Setup and requirements**

For each lab, you get a new GCP project and set of resources for a fixed time at no cost.

1. Make sure you signed into Qwiklabs using an **incognito window**.
2. Note the lab's access time (for example,  and make sure you can finish in that time block.

There is no pause feature. You can restart if needed, but you have to start at the beginning.

1. When ready, click .
2. Note your lab credentials. You will use them to sign in to Cloud Platform Console. 
3. Click **Open Google Console**.
4. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.

If you use other credentials, you'll get errors or **incur charges**.

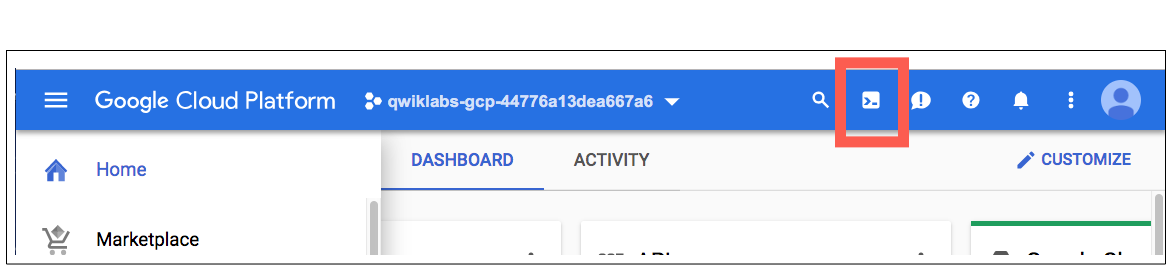
1. Accept the terms and skip the recovery resource page.

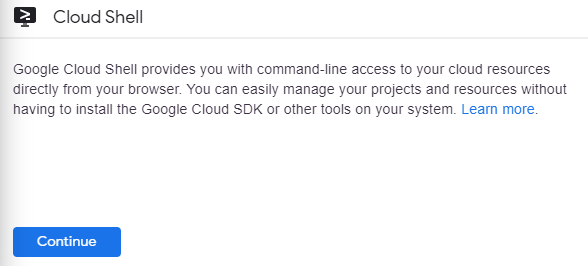
Do not click **End Lab** unless you are finished with the lab or want to restart it. This clears your work and removes the project.

Activate Google Cloud Shell

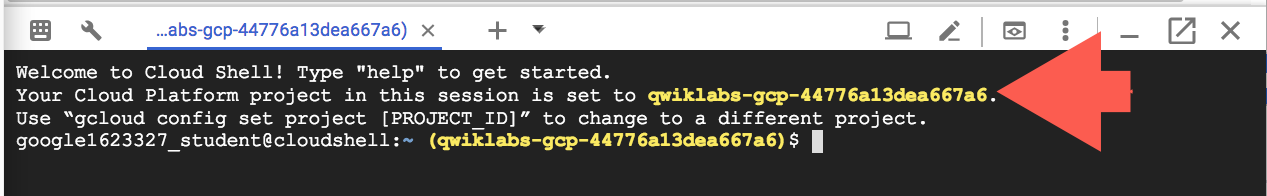
Google Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Google Cloud Shell provides command-line access to your GCP resources.

1. In GCP console, on the top right toolbar, click the Open Cloud Shell button.



1. Click **Continue**. 

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT\_ID*. For example:



**gcloud** is the command-line tool for Google Cloud Platform. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

gcloud auth list

content\_copy

Output:

Credentialed accounts:

- <myaccount>@<mydomain>.com (active)content\_copy

Example output:

Credentialed accounts:

- google1623327\_student@qwiklabs.netcontent\_copy

You can list the project ID with this command:

gcloud config list project

content\_copy

Output:

[core]

project = <project\_ID>content\_copy

Example output:

[core]

project = qwiklabs-gcp-44776a13dea667a6content\_copy

Full documentation of **gcloud** is available on [Google Cloud gcloud Overview](https://cloud.google.com/sdk/gcloud).

**Preparing the case study application**

In this section, you access Cloud Shell, clone the git repository containing the Quiz application, configure environment variables, and run the application.

**Clone source code in Cloud Shell**

1. To clone the repository for the class, enter the following command:

git clone https://github.com/GoogleCloudPlatform/training-data-analystcontent\_copy

1. Create a soft link as a shortcut to your working directory:

ln -s ~/training-data-analyst/courses/developingapps/v1.2/nodejs/cloudfunctions ~/cloudfunctionscontent\_copy

**Configure and run the case study application**

1. To change the working directory, enter the following command:

cd ~/cloudfunctions/startcontent\_copy

1. To configure the Quiz application, enter the following command:

. prepare\_environment.shcontent\_copy

This script file:

* Creates an App Engine application.
* Exports the environment variables GCLOUD\_PROJECT and GCLOUD\_BUCKET.
* Runs npm install.
* Creates entities in Cloud Datastore.
* Creates a Cloud Pub/Sub topic.
* Creates a Cloud Spanner Instance, Database, and Table.
* Prints out the Google Cloud Platform Project ID.

1. To run the web application, enter the following command:

npm startcontent\_copy

Click Check my progress to verify the objective.

Configure and run the case study application

Check my progress

**Working with Cloud Functions**

In this section, you create a Cloud Function in your Google Cloud Platform project that is triggered by publishing a message to Cloud Pub/Sub, runs the application, and monitors the Cloud Function invocation.

**Create a Cloud Function**

1. Return to the **Cloud Platform Console**.
2. On the **Navigation menu**, select **Cloud Functions**.
3. If necessary, click **Enable API**.

It might take a minute or two to enable the Cloud Functions API.

1. Click **Create function**.

In the Create function dialog , set the following properties and leave the remaining settings as their defaults:

|  |  |
| --- | --- |
| **Property** | **Value** |
| **Function name** | **process-feedback** |
| **Trigger type** | **Cloud Pub/Sub** |
| **Topic** | Select **projects/[PROJECT\_ID]/topics/feedback** |

1. Click **Save**, and then click **Next**.
2. Review the provided function implementation, and then click **Deploy**.

It might take a minute or two to create the Cloud Function.

Click Check my progress to verify the objective.

Create a Cloud Function

Check my progress

**Run the web application**

1. Return to the **Cloud Shell** window.
2. Preview the web application by clicking on the Web Preview button and selecting **Preview on port 8080**.
3. Click **Take Test**.
4. Click **Places**.
5. Complete the question.
6. Rate the quiz, enter some feedback, and then click **Send Feedback**.

**View Cloud Function monitoring and logs**

1. Return to the **Cloud Platform Console**, on the **Cloud Functions** page.
2. Click **process-feedback**.

You should see a monitoring graph for Cloud Function invocations.

It takes a few minutes for the graph to show the invocation of your function.

1. Click **View Logs** in the upper right of the page.
2. If your Cloud Pub/Sub message isn't displayed, click **Jump to now**.

You should see the log entries collected from the Cloud Function.

It takes a few minutes for the logs to show the invocation of your function.

1. Go to Cloud Shell to also see your feedback.

**Examining the case study application code**

In this section, you use the Cloud Shell text editor to review the case study application code.

**Launch the Cloud Shell Editor**

1. In the **Cloud Platform Console**, click **Launch code editor**.



Review the Cloud Function application code structure

1. Navigate to **cloudfunctions/start**.
2. Select the **index.js** file in the **...function** folder.

This file contains the same code as the sample from the Cloud Functions window in the Cloud Platform Console, with one change: because the function you will write returns a Promise, the callback argument has been omitted.

1. Select the **package.json** file.

This file contains the list of dependencies that this function needs to run.

Cloud Functions automatically install the dependencies.

1. Select the **languageapi.js** file.

This file contains code to process feedback text and return the Natural Language ML API sentiment score.

1. Select the **spanner.js** file.

This file contains code to insert a record into a Cloud Spanner database.

**Coding a Cloud Function**

In this section you write the code to create a Cloud Function that retrieves the Cloud Pub/Sub message data, invokes the Natural Language ML API to perform sentiment detection, and inserts a record into Cloud Spanner.

**Important**: Update code within the sections marked as follows:

// TODO

// END TODO

To maximize your learning, review the code, inline comments, and related API documentation.

**Write code to modify a Cloud Function**

1. Return to the ...function/index.js file.
2. Load the languageapi and spanner modules. These modules are in the same folder as the index.js file.
3. In the subscribe() method, after the existing code that loads the Cloud Pub/Sub message into a buffer, convert the PubSub message into a feedback object by parsing it as JSON data.
4. Return a promise that invokes the languageapi module's analyze method to analyze the feedback text.
5. Chain a .then(...) method to the end of the return statement.
6. Supply an arrow function as the value of the callback.
7. In the body of the arrow function, log the Natural Language API sentiment score to the console.
8. Add a new property called score to the feedback object.
9. Complete the arrow function body by returning the feedback object.
10. Chain a second .then(...) method to the end of the first one. This one uses the spanner module to save the feedback.
11. Write a third .then(...) chained method, including an arrow function with no arguments and an empty body as the value of the callback.
12. In the body of this callback, log a message to indicate that the feedback has been saved, and return a success message.
13. Attach a .catch(...) handler to the end of the chain, which logs the error message to the console.

Here's how your function should look when you're done.

**function/index.js**

*// TODO: Load the ./languageapi module*

const languageAPI = require('./languageapi');

*// END TODO*

*// TODO: Load the ./spanner module*

const feedbackStorage = require('./spanner');

*// END TODO*

exports.subscribe = function subscribe(event) {

*// The Cloud Pub/Sub Message object.*

*// TODO: Decode the Cloud Pub/Sub message*

*// extracting the feedbackObject data*

*// The message received from Pub/Sub is base64 encoded, and*

*// the data submitted by students is in a data property*

const pubsubMessage = Buffer.from(event.data, 'base64').toString();

let feedbackObject = JSON.parse(pubsubMessage);

console.log('Feedback object data before Language API:' + JSON.stringify(feedbackObject));

*// END TODO*

*// TODO: Run Natural Language API sentiment analysis*

*// The analyze(...) method expects to be passed the*

*// feedback text from the feedbackObject as an argument,*

*// and returns a Promise.*

return languageAPI.analyze(feedbackObject.feedback).then(score => {

*// TODO: Log the sentiment score*

console.log(`Score: ${score}`);

*// END TODO*

*// TODO: Add new score property to feedbackObject*

feedbackObject.score = score;

*// END TODO*

*// TODO: Pass feedback object to the next handler*

return feedbackObject;

*// END TODO*

})

*// TODO: insert record*

.then(feedbackStorage.saveFeedback).then(() => {

*// TODO: Log and return success*

console.log('feedback saved...');

return 'success';

*// END TODO*

})

*// END TODO*

*// TODO: Catch and Log error*

.catch(console.error);

*// End TODO*

};content\_copy

1. Save the file.

**Package and deploy the Cloud Function code**

1. Return to **Cloud Shell** and stop the web application by pressing **Ctrl+C**.
2. To change the working directory to the Cloud Function code, enter the following command:

cd functioncontent\_copy

1. To zip up the files needed to deploy the function, enter the following command:

zip cf.zip \*.js\*content\_copy

This generates a zip archive named cf.zip that includes all the JavaScript and JSON files in the folder.

1. To stage the zip file into Cloud Storage, enter the following command:

gsutil cp cf.zip gs://$GCLOUD\_BUCKET/content\_copy

This copies the zip archive into the Cloud Storage bucket named after your project ID with a -media suffix.

1. Return to the **Cloud Platform Console** and navigate to **Cloud Functions**.
2. Select the **process-feedback** function.
3. Click **Edit**.
4. Click **Next**.
5. Under **Source code**, select **ZIP from Cloud Storage**.
6. For **Cloud Storage location**, Click **Browse**, select the **cf.zip** file in the bucket named after your GCP project ID with the -media suffix, and click **Select**.
7. In the **Entry point** field, type **subscribe**.
8. Click **Deploy**.

It may take a minute or two to update the **process-feedback** function with your new code that performs sentiment analysis with the Cloud Natural Language Machine Learning API and inserts data into Cloud Spanner.

Package and deploy the Cloud Function code

Check my progress

**Testing the case study application**

**Run the web application**

1. Return to the **Cloud Shell** window.
2. Change the working folder back to the start folder for the **cloudfunctions** lab.

cd ..content\_copy

1. To start the web application, execute the following command:

npm startcontent\_copy

1. Preview the web application.
2. Click **Take Test**.
3. Click **Places**.
4. Complete the question.
5. Enter some feedback, and click **Send Feedback**.

**View Cloud Function monitoring and logs**

1. Return to the **Cloud Functions** section of the **Cloud Platform Console**.
2. Click the function name, **process-feedback**.

You should see a monitoring graph for Cloud Function invocations.

It takes a few minutes for the graph to show the invocation of your function.

1. Click **View Logs**.
2. If your Cloud Pub/Sub message isn't displayed, click **Jump to now**.
3. Refresh the logs every few minutes until the log entries show that your function executed.

You should see the log entries collected from the Cloud Function.

You should now see a score from the Language API in the logs.

It may takes a few minutes for the logs to show the invocation of your function.

**View Cloud Spanner data**

1. On the **Navigation menu**, click **Spanner**.
2. Click **Quiz :instance**, and then on the left pane, **quiz-database** > **Query**.
3. Run the following query:

SELECT \* FROM Feedbackcontent\_copy

You should see that a new record has been added to the Feedback table.

**Bonus: Storing student answers using a Cloud Function**

When a student completes a quiz, their answers are submitted in an API call back to the server. Your job is to capture the student-submitted answers and the correct answers and save them into Cloud Spanner. You should see that a new record has been added to the Feedback table.

To do this:

1. Create a Cloud Pub/Sub topic called answers.
2. Create a Cloud Spanner table called Answers with appropriate column names and data types.
3. Post the answer data to the answers topic.
4. Subscribe to the answers topic in the console application and insert the answer data into the Answers table.

The details are left up to you! You can find the solution to the bonus in the lab's **bonus** folder.

**Review**

Which triggers can be used with Cloud Functions?

* Cloud Pub/Sub
* Cloud Spanner
* Cloud Storage
* HTTP

With a Cloud Function triggered by Cloud Pub/Sub, how is the message delivered?

* Base64 encoded
* CSV encoded
* Tarred and Gzipped
* Zipped

What is the maximum execution time for a Cloud Function?

* 60 seconds
* 540 seconds
* Unlimited