

Using Firestore with Cloud IoT Core for Device Configuration

GSP265



Overview

In this lab you will learn how to configure Cloud Functions for Firebase to relay document changes in [Cloud Firestore](#) as configuration updates for [Cloud IoT Core](#) Devices. Cloud IoT Core provides a way to send configuration to devices over MQTT or HTTP. The structure of this payload is unspecified and delivered as raw bytes. This means that if you have different parts of your IoT system wanting to write parts of the configuration, each has to parse, patch, then re-write the configuration value in IoT Core.

If you want a payload delivered to a device in a binary format, such as [CBOR](#), that means each of these participating components of your system also need to deserialize and re-serialize the structured data.

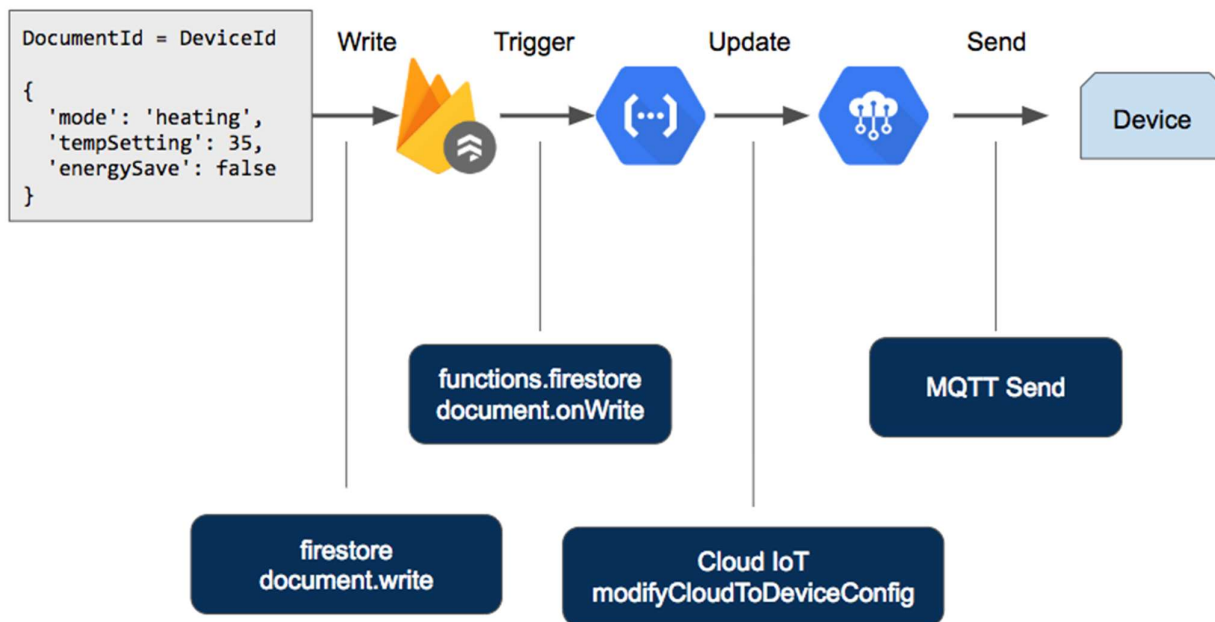
By using Cloud Firestore to serve as a layer in between the systems that update a device's configuration and IoT Core, you can take advantage of Firestore's structured [data types](#) and partial document updates.

Objectives

In this lab you will learn how to:

- Create IoT Core registries and devices.
- Deploy Cloud Functions with Firebase.
- Create and modify Firestore documents.
- Use a Firestore document as an IoT Core device configuration.
- Deploy a Cloud Function that relays Firestore document changes to MQTT and CBOR IoT devices.

The diagram below highlights the flow of data from a Firestore document to an IoT device and provides a high level overview of what you will be building in this lab:



Prerequisites

This is an **advanced level** lab. Familiarity and experience with Firestore and IoT is not required, but may be helpful. This lab covers a specific application of IoT—if you are looking for more comprehensive practice, be sure to check out the following labs:

- [A Tour of Cloud IoT Core](#)
- [Building an IoT Analytics Pipeline on Google Cloud](#)

Once you're ready, scroll down to get your lab environment set up.

Setup and Requirements

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

What you need

To complete this lab, you need:

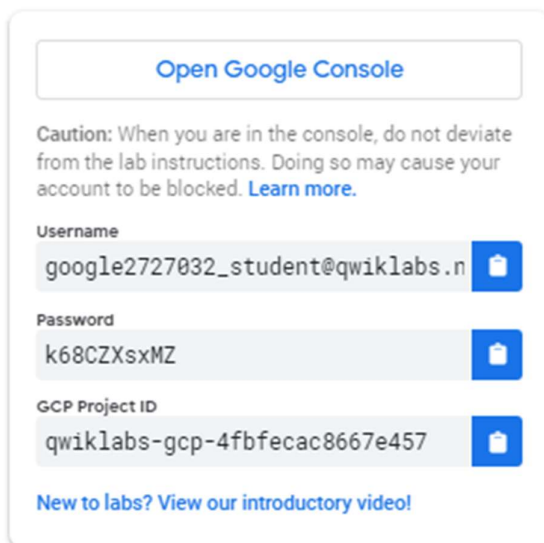
- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.

Note: If you already have your own personal Google Cloud account or project, do not use it for this lab.

Note: If you are using a Pixelbook, open an Incognito window to run this lab.

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

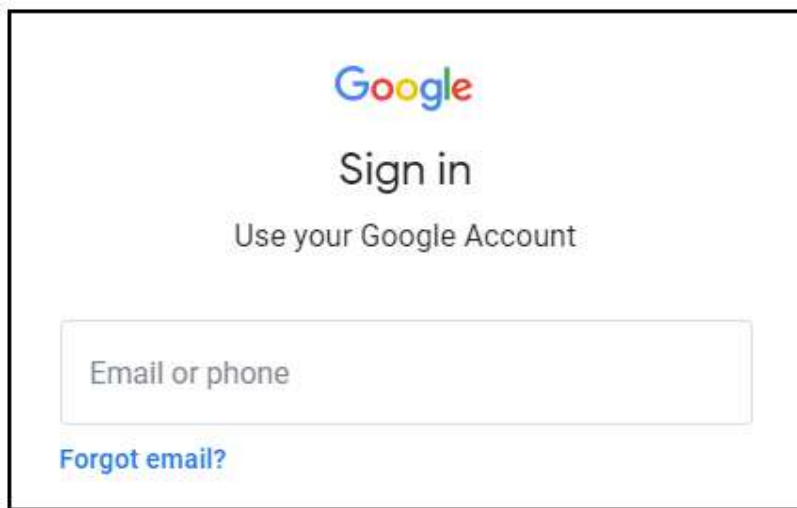
1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



The screenshot shows a sign-in panel with the following elements:

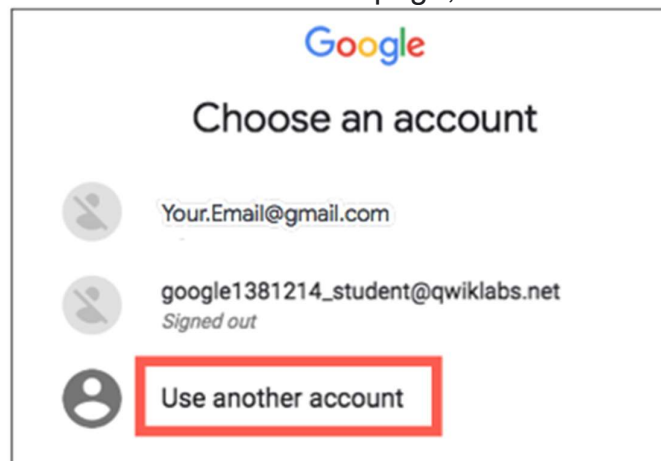
- A button at the top labeled "Open Google Console".
- A caution message: "Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. [Learn more.](#)"
- Three input fields, each with a copy icon to its right:
 - Username:** google2727032_student@qwiklabs.n
 - Password:** k68CZXsxMZ
 - GCP Project ID:** qwiklabs-gcp-4fbfecac8667e457
- A link at the bottom: "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.



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).

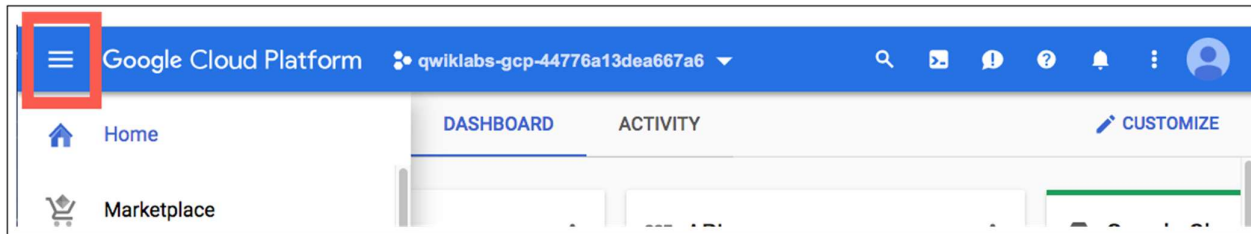
4. Click through the subsequent pages:

- 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-

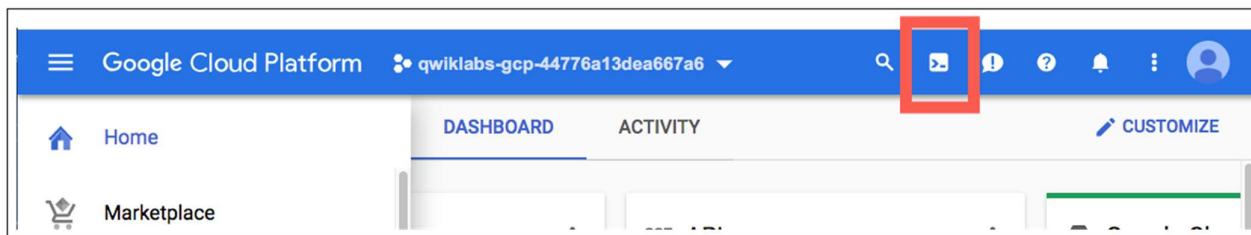
left.



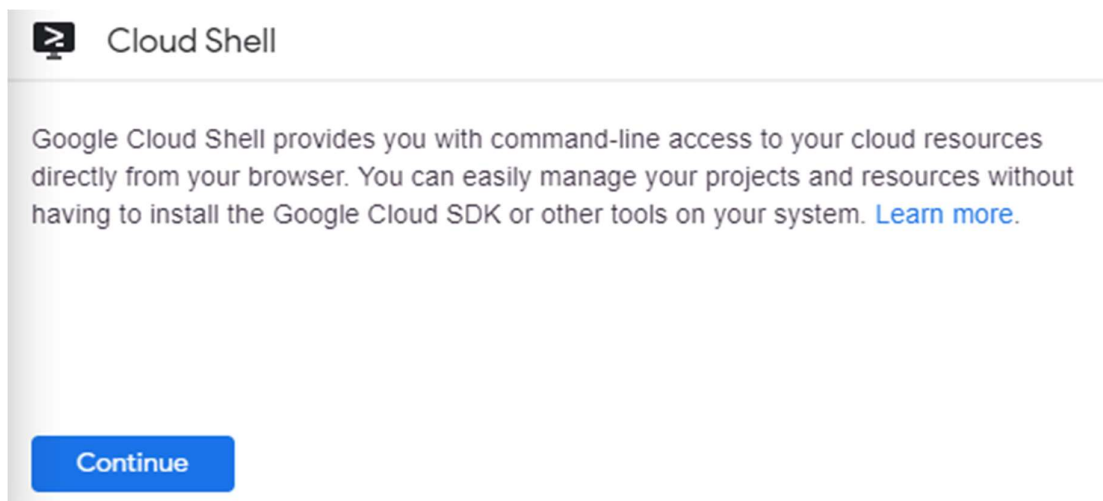
Activate Cloud Shell

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. Cloud Shell provides command-line access to your Google Cloud resources.

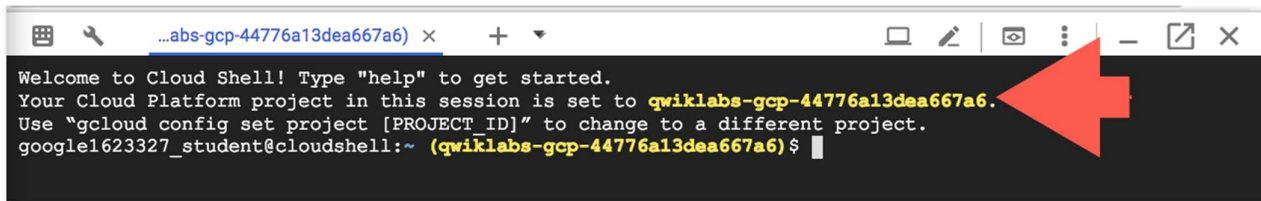
In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



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:



```
...abs-gcp-44776a13dea667a6) x + ▾
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to qwiklabs-gcp-44776a13dea667a6.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
google1623327_student@cloudshell:~ (qwiklabs-gcp-44776a13dea667a6) $
```

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

You can list the active account name with this command:

```
gcloud auth list
```

(Output)

```
Credentialed accounts:
- <myaccount>@<mydomain>.com (active)
```

(Example output)

```
Credentialed accounts:
- google1623327_student@qwiklabs.net
```

You can list the project ID with this command:

```
gcloud config list project
```

(Output)

```
[core]
project = <project ID>
```

(Example output)

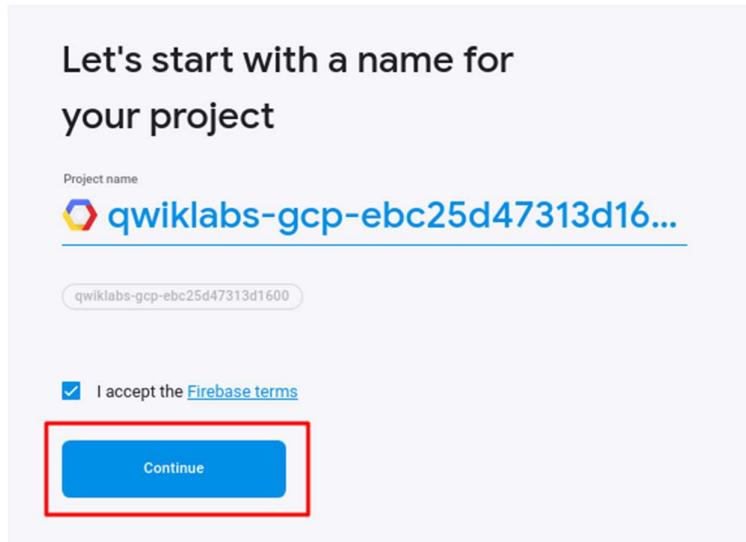
```
[core]
project = qwiklabs-gcp-44776a13dea667a6
```

For full documentation of `gcloud` see the [gcloud command-line tool overview](#).

Set up the Environment


Before diving in, you'll need to associate Firebase with your Google Cloud project.

Open the [Firebase Console](#) in a new tab in your browser. Then click **Add project**. From the Project name dropdown menu select your Qwiklabs Google Cloud Project, accept the Firebase terms, and click **Continue**.



Let's start with a name for your project

Project name

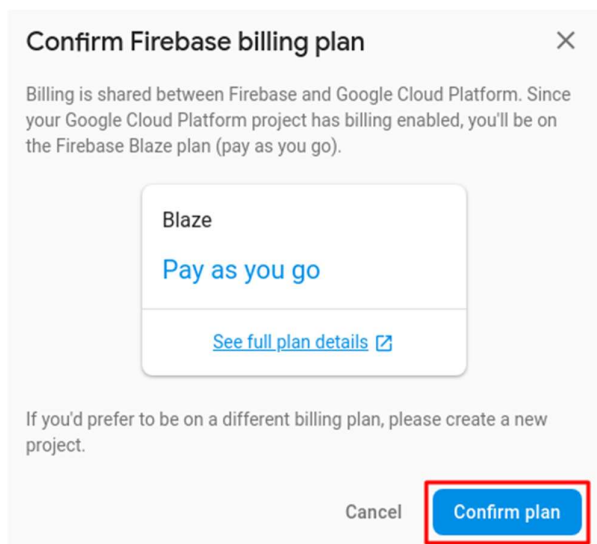
 qwiklabs-gcp-ebc25d47313d16...

qwiklabs-gcp-ebc25d47313d1600

☒ I accept the [Firebase terms](#)

Continue

When asked to confirm a Firebase billing plan click **Confirm plan**.



Confirm Firebase billing plan

Billing is shared between Firebase and Google Cloud Platform. Since your Google Cloud Platform project has billing enabled, you'll be on the Firebase Blaze plan (pay as you go).

Blaze

Pay as you go

[See full plan details](#)

If you'd prefer to be on a different billing plan, please create a new project.

Cancel **Confirm plan**

Click **Continue**.

On Google Analytics for your Firebase project console select slide the toggle for "Enable Google Analytics for this project" and click **Continue**.


×


Create a project (Step 3 of 4)


Google Analytics for your Firebase project


Google Analytics is a free and unlimited analytics solution that enables targeting, reporting, and more in Firebase Crashlytics, Cloud Messaging, In-App Messaging, Remote Config, A/B Testing, Predictions, and Cloud Functions.


Google Analytics enables:


 A/B testing ?

 User segmentation & targeting across Firebase products ?

 Predicting user behavior ?

 Crash-free users ?

 Event-based Cloud Functions triggers ?

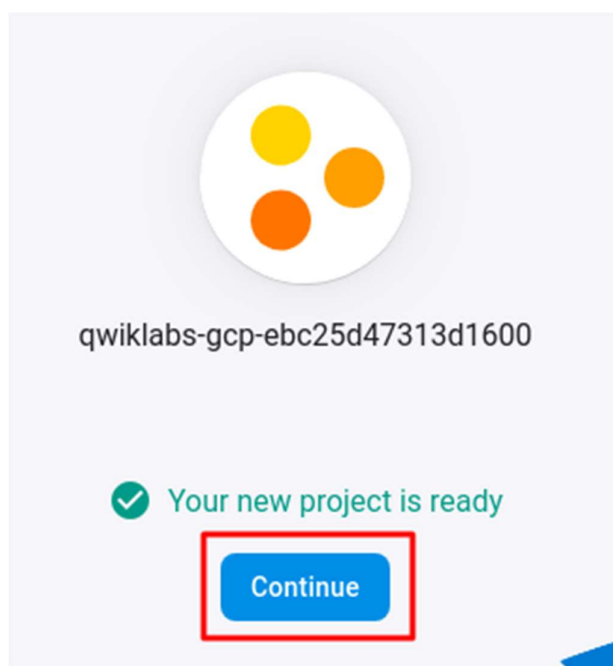
 Free unlimited reporting ?

☒ Enable Google Analytics for this project
Recommended

[Previous](#)[Continue](#)

On the next page, accept the **Google Analytics terms** and click **Add Firebase**.

On the next page, select **Continue** and you will navigate to Firebase Console.



Now go back to the Cloud Console and start a new Cloud Shell session.

Click **Check my progress** to verify the objective.

First, copy the sample code from a public storage bucket where the project files are hosted. Run the following commands to do so:

```
gsutil -m cp -r gs://spls/gsp265/community .  
cd community/tutorials/cloud-iot-firestore-config
```

Now set the names of the Cloud IoT Core environment variables:

```
export REGISTRY_ID=config-demo  
export CLOUD_REGION=us-central1 # or change to an alternate region;  
export GLOUD_PROJECT=$(gcloud config list project --format "value(core.project)")
```

Authorize the Firebase CLI

Run the `firebase login` command to authorize Firebase.

```
firebase login --no-localhost
```

The `--no-localhost` option is used because you are on a remote shell.

When asked if you should **Allow Firebase to collect anonymous CLI usage and error reporting information**, type "Y".

Copy the link in the output into a new tab. **Do not click the link!** Select your lab username, then click **Allow**.

Copy the verification code from the browser and enter it in the Cloud Shell prompt.

Create a Cloud IoT Core Registry

Now that you have your project files and environment variables instantiated, you will create an [IoT Registry](#) to hold our devices.

First, run the following command to create a PubSub topic to use for device logs:

```
gcloud pubsub topics create device-events
```

Click **Check my progress** to verify the objective.

Then run the following command to create an IoT Core registry:

```
gcloud iot registries create $REGISTRY_ID --region=$CLOUD_REGION --event-notification-config=subfolder="",topic=device-events
```

Click **Check my progress** to verify the objective.

Deploy the Relay Function

You will use a Firestore document trigger to run a function every time a qualifying document is updated.

Take a look at the main function file now. Run the following commands to inspect the **index.ts** file:

```
cd functions/src/  
nano index.ts
```

The main part of the function handles a Cloud PubSub message from IoT Core, extracts the log payload and device information, and then writes a structured log entry to Cloud Logging:

```
'use strict';  
  
import cbor = require('cbor');  
  
import * as admin from "firebase-admin";  
import * as functions from 'firebase-functions';  
const iot = require('@google-cloud/iot');  
const client = new iot.v1.DeviceManagerClient();  
  
// start cloud function  
exports.configUpdate = functions.firestore  
  // assumes a document whose ID is the same as the deviceid  
  .document('device-configs/{deviceId}')
```

```
.onWrite(async (change: functions.Change<admin.firestore.DocumentSnapshot>, context?:
functions.EventContext) => {
  if (context) {
    console.log(context.params.deviceId);
    const request = generateRequest(context.params.deviceId, change.after.data(),
false);
    return client.modifyCloudToDeviceConfig(request);
  } else {
    throw(Error("no context from trigger"));
  }
});
```

You will also notice that the function runs only when documents in the `device-configs` collection are updated. It's important to note that the document key is used as the corresponding device key.

Exit the text editor with **Ctrl + X**.

Update and install necessary packages for the application.

```
cd ..
npm install firebase-functions@3.2.0 --save --force
npm install firebase-admin@8.4.0 --save
npm install @google-cloud/iot@1.1.3
npm install typescript@3.5.3 --save --legacy-peer-deps
npm install
```

Note: You can safely ignore any warnings that are outputted when executing the above commands.

Now run the following commands to deploy the cloud function with the [Firebase CLI tool](#) which comes preinstalled in Cloud Shell:

```
firebase --project $GCPLOUD_PROJECT functions:config:set \
  iot.core.region=$CLOUD_REGION \
  iot.core.registry=$REGISTRY_ID
firebase --project $GCPLOUD_PROJECT deploy --only functions
```

You can ignore any warnings you see. You should receive a similar output after a couple minutes:

```
✓ functions: Finished running predeploy script.
i functions: ensuring necessary APIs are enabled...
✓ functions: all necessary APIs are enabled
i functions: preparing functions directory for uploading...
i functions: packaged functions (67.4 KB) for uploading
✓ functions: functions folder uploaded successfully
i functions: creating Node.js 8 function configUpdate(us-central1)...
i functions: creating Node.js 8 function configUpdateBinary(us-central1)...
✓ functions[configUpdateBinary(us-central1)]: Successful create operation.
✓ functions[configUpdate(us-central1)]: Successful create operation.

✓ Deploy complete!
```

If you see this warning message, "HTTP Error: 403, Unknown Error", wait a minute and run the previous command again.

Click **Check my progress** to verify the objective.

Create your device

Now that your function has been deployed, create a dummy IoT device called `sample-device` by running the following commands:

```
cd ../sample-device
gcloud iot devices create sample-device --region $CLOUD_REGION --registry $REGISTRY_ID
--public-key path=./ec_public.pem,type=ES256
```

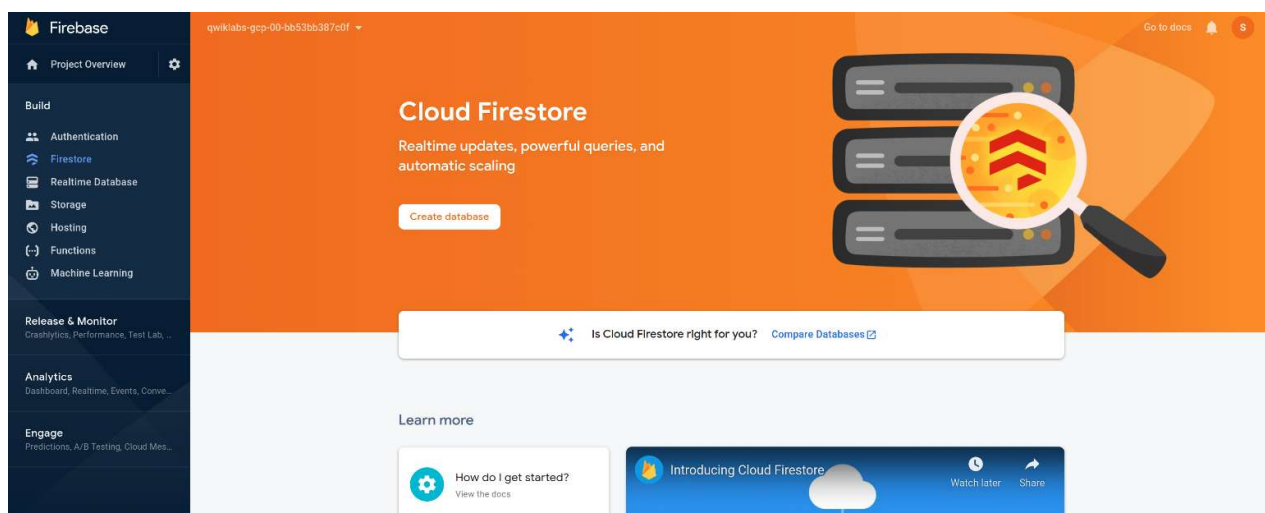
Note: Do not use this device for any real workloads, as the key-pair is included in this sample and therefore is not secret.

Click **Check my progress** to verify the objective.

Establish a Device Configuration in Firestore

You will now create a sample document in Firestore so you can start passing data between your device and the database.

In the Firestore console, in the left-hand menu, click **Build > Firestore**. Then in the Cloud Firestore panel at the top, click **Create database**:



Select the **Start in test mode** button and click **Next**. On the next page, select **nam5 (us-central)** from Cloud Firestore location dropdown and click **Enable**.

Now click **+ Start collection** and name the collection `device-configs`, then click **Next**. You will be prompted to create your first document. Type `sample-device` for the Document Id.

For the Field, Type, and Value, enter in the following:

Field	Type	Value
mode	string	heating
tempSetting	number	35
energySave	boolean	false

The `sample-device` document should now resemble the following:

Document id

sample-device

Field	Type	Value
mode	string	heating
tempSetting	number	35
energySave	boolean	false

CANCEL SAVE

The different fields in the config can have different data types.

Click **Save**.

Click **Check my progress** to verify the objective.

Now, back in the Google Console, open the IoT Core console by opening the Navigation menu and selecting **IoT Core > config-demo**. Then click on **Devices > sample-device**, then click the **CONFIGURATION & STATE**:

Device ID: sample-device

Numeric ID	Registry	Cloud Logging	Communication
2711619662798242	config-demo	Registry default View logs	Allowed

DETAILS CONFIGURATION & STATE AUTHENTICATION

☒ Configuration history ☒ State history [COMPARE](#) [↻](#)

Cloud update: March 22, 2021

Latest		CONFIG (Version 2)	Cloud update 10:31 PM	e
		CONFIG (Version 1)	Cloud update 10:20 PM	

Only the last 10 configurations and states are stored in Cloud IoT Core. [Learn more](#)

Click **CONFIG (Version 2) > Format (Text)**.

If the Function ran successfully, the configuration panel will reflect the Firestore document you just added. Your console should resemble the following:

Cloud update: March 22, 2021

Configuration

Cloud update	10:31 PM
Version	2

Format

☐ Base64

☒ Text

```
{"energySave":false,"mode":"heating","tempSetting":35}
```

Modify the Configuration

Now that you have verified the connection between the device and Firestore, go back to your Cloud Shell window and run the following commands to start up the sample device (make sure that you are still in the `sample-device` directory):

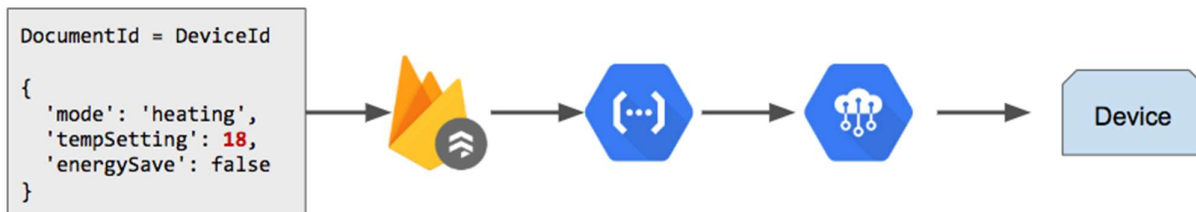
```
npm install --legacy-peer-deps
node build/index.js
```

Note: You can safely ignore any warnings and errors that are outputted when `npm install` is run. After running the `node build/index.js` command you should see the following output:

```
Device Started
Current Config:
{ energySave: false, mode: 'heating', tempSetting: 35 }
```

Now go back to the Firestore document tab and update the `sample-device` config: change the `tempSetting` value to **18** and click **Update**.

When this document edit is saved, it triggers a function, which will push the new config down to the device. The following is a visual representation of the pipeline:



Return to your Cloud Shell session and you will see this new config arrive at the sample device:

```
Device Started
Current Config:
{ energySave: false, mode: 'heating', tempSetting: 35 }
Current Config:
{ energySave: false, mode: 'heating', tempSetting: 18 }
```

Ctrl + c to stop the device and return to the command line.

To do this programmatically with only the IoT Core APIs, you would have to read the current config from the IoT Core Device Manager, update the value, then write back the new config to IoT Core. IoT Core provides an incrementing version number you can send with these writes to check that another process has not concurrently attempted to update the config.

This solution assumes that the path using Firestore and functions are not sharing the config update job with other processes, but are acting as a flexible intermediate. Both single Firestore Documents and IoT Core device configurations are limited to one update per second.

Binary data with CBOR

Sometimes, with constrained devices and constrained networks, you want to work with data in a compact binary format. Concise Binary Object Representation [CBOR](#) is a binary format that strikes a balance between the compactness of binary with the self-describing format of JSON. IoT Core device configuration API and MQTT both fully support binary messages.

In cloud software and databases, binary might not be as convenient to work with, or other binary formats such as protocol buffers might be used.

By using a function as an intermediate between Firestore and IoT Core, you can not only watch documents for changes to trigger an update, but you can use the same function to encode the payload into the CBOR binary representation.

For clarity, this lab implements this with a different function, and uses a different Firestore collection.

In your Cloud Shell window, press **Ctrl+c** to stop the sample device script.

Now, deploy a fresh version of the function by running the following commands:

```
cd ../functions
firebase --project $GCP_PROJECT deploy --only functions
```

Create another sample device variation. This one will be named `sample-binary`:

```
cd ../sample-device/
gcloud iot devices create sample-binary --region $CLOUD_REGION --registry $REGISTRY_ID
--public-key path=./ec_public.pem,type=ES256
```

Go back to the Firestore database console and create another Firestore collection as you did before, but name it `device-configs-binary` and add a document for the `sample-binary` device. For the Field, Type, and Value, enter in the following:

Field	Type	Value
mode	string	cooling
tempSetting	number	22
energySave	boolean	true

The "sample-binary" document should now resemble the following:

Start a collection

1 Set collection ID

2 Add first document

Document parent path ?

/device-configs-binary

Document ID ?

sample-binary

Field	Type	Value	
mode	= string	cooling	—
tempSetting	= number	22	—
energySave	= boolean	true	—

+

Cancel

Save

Click **Save**.

Click **Check my progress** to verify the objective.

Create another device

Check my progress

Now back in Cloud Shell, start the device with the `-b` flag to indicate that you want to use the binary version of the sample device (verify that you are still in the `sample-device` directory):

```
node build/index.js -b
```

After running the above command you should see the following output:

```
Device uses binary config
Device Started
Current Config:
{ energySave: true, mode: 'cooling', tempSetting: 22 }
```

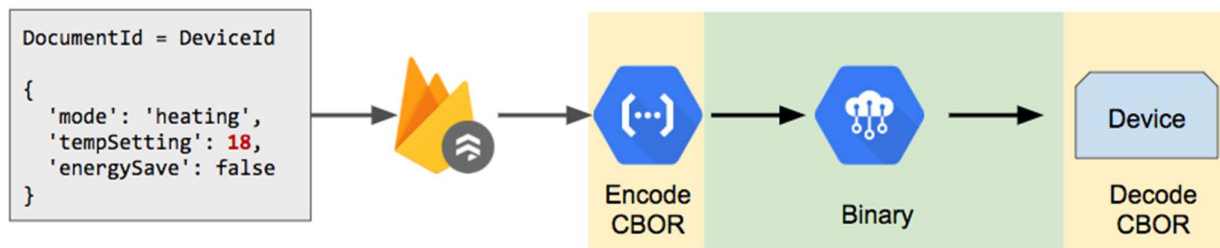
You can update the device config settings in Firestore, and you will see the decoded config printed on the screen (note that the payload of the config is transmitted and encoded as CBOR.)

Update the config document in the Firestore console by changing the `tempSetting` value to **17**.

You should see this new config arrive at the sample device in a moment:

```
Device uses binary config
Device Started
Current Config:
{ energySave: true, mode: 'cooling', tempSetting: 22 }
Current Config:
{ energySave: true, mode: 'cooling', tempSetting: 17 }
```

Note: When the data is encoded as CBOR you will not be able to see or edit this in the IoT Core console, as it is in a compact encoded format that the console does not parse for display. The following is a visual representation of the CBOR pipeline:



Cleaning up

When you complete a Qwiklab, all of your resources will be deleted. However, it is still important to know how to clean up a workspace.

Stop the sample device by pressing **Ctrl + c**. Since the test devices uses a visible key, you should delete them. Run the following command to delete the `sample-device` and `sample-binary` devices:

```
gcloud iot devices delete sample-device --registry $REGISTRY_ID --region $CLOUD_REGION
gcloud iot devices delete sample-binary --registry $REGISTRY_ID --region $CLOUD_REGION
```

If prompted to continue the device deletion, enter **y**.

Click **Check my progress** to verify the objective.

Congratulations!

Nice job! In this lab you got hands-on practice with Firestore, Cloud Functions, and Cloud IoT Core. You learned how to configure a Firebase Function to relay document changes in [Cloud Firestore](#) as configuration updates for [Cloud IoT Core](#) Devices—both in text and CBOR binary format.

This hands-on lab is based on the [Using Cloud Firestore with Cloud IoT Core for Device Configuration](#) community tutorial written by [Preston Holmes](#), a Solution Architect at Google.



Finish Your Quest

This self-paced lab is part of the Qwiklabs [IoT in the Google Cloud](#) Quest. 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](#).

Take your next lab

Continue your Quest with [Internet of Things: Qwik Start](#)r, or check out these suggestions:

- [Streaming IoT Core to Cloud Storage](#)
- [Building an IoT Analytics Pipeline on Google Cloud](#)

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 March 22, 2021

Lab Last Tested March 22, 2021

Copyright 2021 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.