

# Cloud Functions 2nd Gen: Qwik Start | Google Cloud Skills Boost

Qwiklabs : 25-32 minutes

GSP1089



## Google Cloud Self-Paced Labs

### Overview

Cloud Functions (2nd gen) is the next version of [Google Cloud Functions](#), Google Cloud's Functions-as-a-Service offering. This new version comes with an advanced feature set and is now powered by [Cloud Run](#) and [Eventarc](#), giving you more advanced control over performance and scalability, and more control around the functions runtime and triggers from over 90+ event sources.

In this lab, you will create Cloud Functions that respond to HTTP calls, and get triggered by Cloud Storage events and Cloud Audit Logs. You will also deploy multiple revisions of a Cloud Function and explore new settings.

### What's New?

This new version of Cloud Functions provides an enhanced FaaS experience powered by Cloud Run, Cloud Build, Artifact Registry, and Eventarc.

#### Enhanced Infrastructure

- **Longer request processing:** Run your Cloud Functions longer than the 5 minute default, making it easier to run longer request workloads such as processing large streams of data from Cloud Storage or BigQuery. For HTTP functions, this is up to 60 minutes. For event-driven functions, this is currently up to 10 minutes.
- **Larger instances:** Take advantage of up to 16GB of RAM and 4 vCPUs on Cloud Functions allowing larger in-memory, compute-intensive and more parallel workloads.
- **Concurrency:** Process up to 1000 concurrent requests with a single function instance, minimizing cold starts and improving latency when scaling.
- **Minimum instances:** Provide for pre-warmed instances to cut your cold starts and make sure the bootstrap time of your application does not impact the application performance.
- **Traffic splitting:** Support multiple versions of your functions, split traffic between different versions and roll your function back to a prior version.

#### Broader Event coverage and CloudEvents support

- **Eventarc Integration:** Cloud Functions now includes native support for Eventarc, which brings over 125+ event sources using Cloud Audit logs (BigQuery, Cloud SQL, Cloud Storage...), and of course Cloud Functions still supports events from custom sources by publishing to Cloud Pub/Sub directly.
- **CloudEvent format:** All event-driven functions adhere to industry standard CloudEvents ([cloudevents.io](https://cloudevents.io)), regardless of the source, to ensure a consistent developer experience. The payloads are sent via a structured CloudEvent with a [cloudevent.data payload](#) and implement the CloudEvent standard.

### Objectives

In this lab, you will:

- Write a function that responds to HTTP calls.

- Write a function that responds to **Cloud Storage** events.
- Write a function that responds to **Cloud Audit** Logs.
- Deploy **multiple revisions** of a Cloud Function.
- Get rid of cold starts with minimum instances.
- Set **up concurrency**

## 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 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.

To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended).

**Note:** Use an Incognito or private browser window to run this lab. This prevents any conflicts between your personal account and the Student account, which may cause extra charges incurred to your personal account.

- Time to complete the lab---remember, once you start, you cannot pause a lab.

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

### 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 the **Lab Details** panel with the following:
  - The **Open Google Console** button
  - Time remaining
  - The temporary credentials that you must use for this lab
  - Other information, if needed, to step through this lab

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

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

**Note:** If you see the **Choose an account** dialog, click **Use Another Account**.

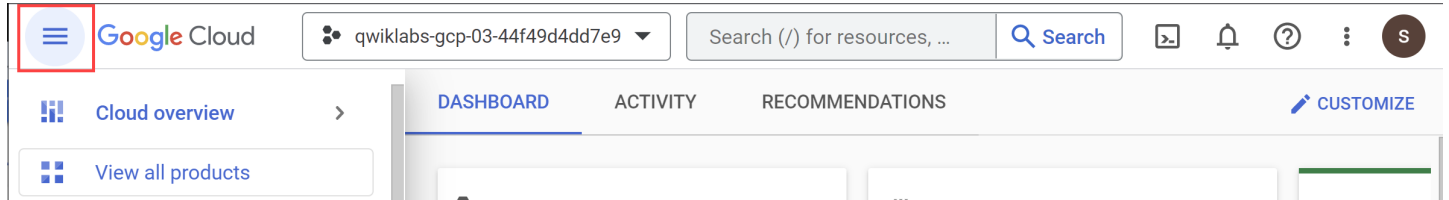
3. If necessary, copy the **Username** from the **Lab Details** panel and paste it into the **Sign in** dialog. Click **Next**.
4. Copy the **Password** from the **Lab Details** panel and paste it into the **Welcome** dialog. Click **Next**.

**Important:** You must use the credentials from the left panel. Do not use your Google Cloud Skills Boost credentials. **Note:** Using your own Google Cloud account for this lab may incur extra charges.

5. 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.

1. Click **Activate Cloud Shell**  at the top of the Google Cloud console.

When you are connected, you are already authenticated, and the project is set to your **PROJECT\_ID**. The output contains a line that declares the **PROJECT\_ID** for this session:

Your Cloud Platform project in this session is set to YOUR\_PROJECT\_ID

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

2. (Optional) You can list the active account name with this command:

```
gcloud auth list
```

3. Click **Authorize**.

4. Your output should now look like this:

### Output:

```
ACTIVE: * ACCOUNT: student-01-xxxxxxxxxxxx@qwiklabs.net To set the active account, run: $ gcloud config set account `ACCOUNT`
```

5. (Optional) 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 Note: For full documentation of gcloud, in Google Cloud, refer to the gcloud CLI overview guide.
```

## Task 1. Enable APIs

Before you create the Cloud Functions, you will need to enable the relevant APIs.

1. In Cloud Shell, run the following command to set your Project ID variable.

```
export PROJECT_ID=$(gcloud config get-value project)
```

2. Run the following command to set the Region variable.

```
export REGION={{project_0.default_region | ""}} gcloud config set compute/region $REGION
```

3. Execute the following command to enable all necessary services.

```
gcloud services enable \ artifactregistry.googleapis.com \ cloudfunctions.googleapis.com \ cloudbuild.googleapis.com \  
eventarc.googleapis.com \ run.googleapis.com \ logging.googleapis.com \ pubsub.googleapis.com
```

## Task 2. Create an HTTP function

For the first function, you will create an **authenticated** Node.js function that responds to HTTP requests, and use a 10 minute timeout to showcase how a function can have more time to respond to HTTP requests.

### Create

1. Run the following command to create the folder and files for the app and navigate to the folder:

```
mkdir ~/hello-http && cd $_ touch index.js && touch package.json
```

2. Click the **Open Editor** button on the toolbar of Cloud Shell. (You can switch between Cloud Shell and the code editor by using the **Open Editor** and **Open Terminal** icons as required, or click the **Open in new window** button to leave the Editor open in a separate tab).

3. In the Editor, add the following code to the `hello-http/index.js` file that simply responds to HTTP requests:

```
const functions = require('@google-cloud/functions-framework'); functions.http('helloWorld', (req, res) => {  
res.status(200).send('HTTP with Node.js in GCF 2nd gen!'); });
```

4. Add the following content to the `hello-http/package.json` file to specify the dependencies.

```
{ "name": "nodejs-functions-gen2-codelab", "version": "0.0.1", "main": "index.js", "dependencies": { "@google-cloud/functions-  
framework": "^2.0.0" } }
```

### Deploy

1. In Cloud Shell, run the following command to deploy the function:

```
gcloud functions deploy nodejs-http-function \ --gen2 \ --runtime nodejs16 \ --entry-point helloWorld \ --source . \ --region $REGION \  
--trigger-http \ --timeout 600s \ --max-instances 1 Note: If you get permissions error, please wait a few minutes and try the  
deployment again. It takes a few minutes for the APIs to be enabled.
```

Although not strictly necessary for this step, there is a timeout of 600 seconds. This allows the function to have a longer timeout to respond to HTTP requests.

2. Once the function is deployed, from the Navigation Menu go to the **Cloud Functions** page. Verify the function was deployed successfully.

Cloud Functions

Functions

+

CREATE FUNCTION

↺

REFRESH

☰

Filter

Filter functions

<input type="checkbox"/>	<input type="radio"/>	Environment	Name <div>↑</div>	Region	Trigger	Runtime
<input type="checkbox"/>	<input checked="" type="radio"/>	2nd gen	nodejs-http-function	us-west1	HTTP	Node.js 16

### Test

1. Test the function with the following command:

```
gcloud functions call nodejs-http-function \ --gen2 --region $REGION
```

You should see the following message as a response:

HTTP with Node.js in GCF 2nd gen!

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

Create a HTTP Function

## Task 3. Create a Cloud Storage function

In this section create a Node.js function that responds to events from a Cloud Storage bucket.

### Setup

1. To use Cloud Storage functions, first grant the pubsub.publisher IAM role to the Cloud Storage service account:

```
PROJECT_NUMBER=$(gcloud projects list --filter="project_id:$PROJECT_ID" --format='value(project_number)')
SERVICE_ACCOUNT=$(gsutil kms serviceaccount -p $PROJECT_NUMBER) gcloud projects add-iam-policy-binding
$PROJECT_ID \ --member serviceAccount:$SERVICE_ACCOUNT \ --role roles/pubsub.publisher
```

### Create

1. Run the following command to create the folder and files for the app and navigate to the folder:

```
mkdir ~/hello-storage && cd $_ touch index.js && touch package.json
```

2. Add the following code to the hello-storage/index.js file that simply responds to Cloud Storage events:

```
const functions = require('@google-cloud/functions-framework'); functions.cloudEvent('helloStorage', (cloudevent) => {
console.log('Cloud Storage event with Node.js in GCF 2nd gen!'); console.log(cloudevent); });
```

3. Add the following content to the hello-storage/package.json file to specify the dependencies:

```
{ "name": "nodejs-functions-gen2-codelab", "version": "0.0.1", "main": "index.js", "dependencies": { "@google-cloud/functions-framework": "^2.0.0" } }
```

### Deploy

1. First, create a Cloud Storage bucket to use for creating events:




```
BUCKET="gs://gcf-gen2-storage-$PROJECT_ID" gsutil mb -l $REGION $BUCKET
```

2. Deploy the function:

```
gcloud functions deploy nodejs-storage-function \ --gen2 \ --runtime nodejs16 \ --entry-point helloStorage \ --source . \ --region
$REGION \ --trigger-bucket $BUCKET \ --trigger-location $REGION \ --max-instances 1
```

3. Once the function is deployed, verify that you can see it under the Cloud Functions section of the Cloud Console.

Filter Filter functions

<input type="checkbox"/>		Environment	Name ↑	Last deployed	Region	Trigger
<input type="checkbox"/>		2nd gen	<a href="#">nodejs-http-function</a>	Sep 19, 2022, 2:57:07 PM	us-west1	HTTP
<input type="checkbox"/>		2nd gen	<a href="#">nodejs-storage-function</a>	Sep 19, 2022, 3:10:58 PM	us-west1	Bucket: <a href="#">gcf-gen2-storage-qwiklabs-gcp-00-101b4b9ca9bb</a>

### Test

1. Test the function by uploading a file to the bucket:

```
echo "Hello World" > random.txt gsutil cp random.txt $BUCKET/random.txt
```

2. Run the following command. You should see the received CloudEvent in the logs:

```
gcloud functions logs read nodejs-storage-function \ --region $REGION --gen2 --limit=100 --format "value(log)" Note: it may take a minute for the logs to generate.
```

You should see output similar to the following:

```
} traceparent: '00-c74cb472d1e78f7225b6f617a31d9c08-96f0380bb62be2c1-01' }, etag: 'CKOx1L3wofoCEAE=' crc32c:
'R1jUOQ==', mediaLink: 'https://storage.googleapis.com/download/storage/v1/b/gcf-gen2-storage-qwiklabs-gcp-00-
101b4b9ca9bb/o/random.txt?generation=1663625646643363&alt=media', md5Hash: '5Z/5eUEET4XfUpfhwwLSYA==', size: '12',
timeStorageClassUpdated: '2022-09-19T22:14:06.657Z', storageClass: 'STANDARD', updated: '2022-09-19T22:14:06.657Z',
timeCreated: '2022-09-19T22:14:06.657Z', contentType: 'text/plain', metageneration: '1', generation: '1663625646643363', bucket:
'gcf-gen2-storage-qwiklabs-gcp-00-101b4b9ca9bb', name: 'random.txt', selfLink: 'https://www.googleapis.com/storage/v1/b/gcf-
gen2-storage-qwiklabs-gcp-00-101b4b9ca9bb/o/random.txt', id: 'gcf-gen2-storage-qwiklabs-gcp-00-
101b4b9ca9bb/random.txt/1663625646643363', kind: 'storage#object', data: { bucket: 'gcf-gen2-storage-qwiklabs-gcp-00-
101b4b9ca9bb', time: '2022-09-19T22:14:06.657124Z', subject: 'objects/random.txt', type: 'google.cloud.storage.object.v1.finalized',
specversion: '1.0', source: '//storage.googleapis.com/projects/_/buckets/gcf-gen2-storage-qwiklabs-gcp-00-101b4b9ca9bb', id:
'5693030851428996', { Cloud Storage event with Node.js in GCF 2nd gen!
```

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

Create a Cloud Storage Function

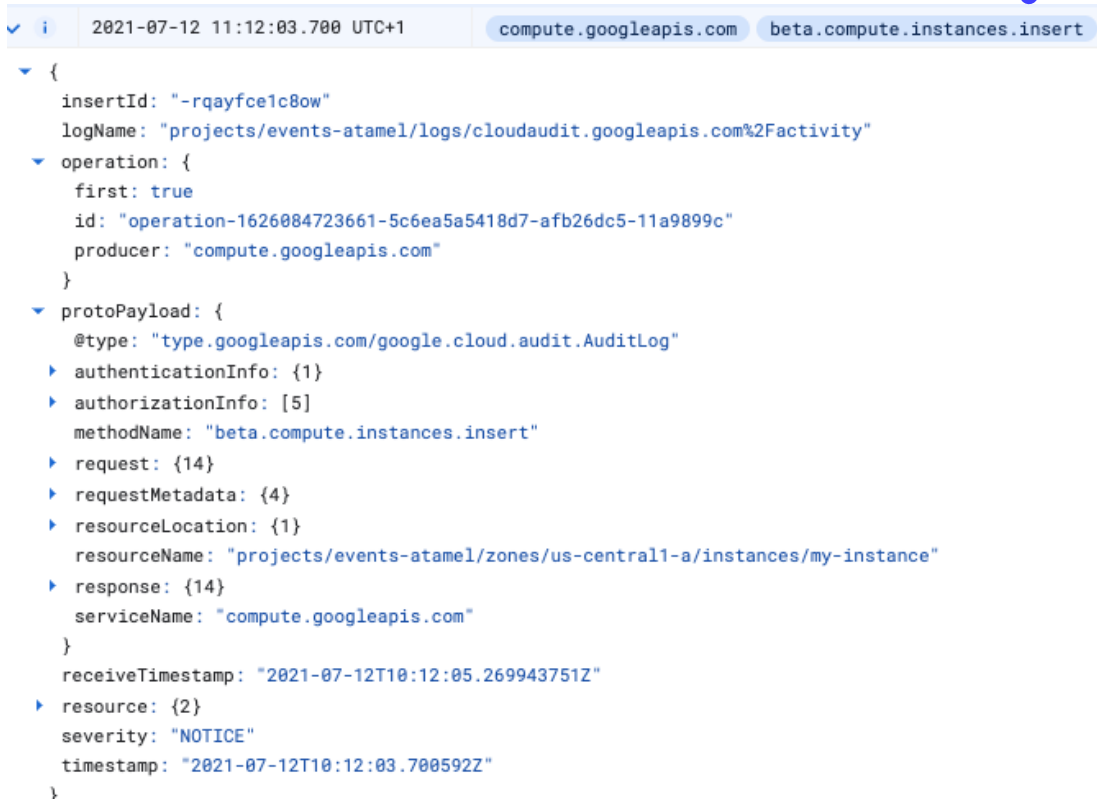
## Task 4. Create a Cloud Audit Logs function

In this section, you will create a Node.js function that receives a Cloud Audit Log event when a Compute Engine VM instance is created. In response, it adds a label to the newly created VM, specifying the creator of the VM.

### Determine newly created Compute Engine VMs

Compute Engine emits 2 Audit Logs when a VM is created.

The first one is emitted at the beginning of the VM creation and looks like this:



The screenshot shows a Cloud Audit Log entry in a table. The entry has a timestamp of 2021-07-12 11:12:03.700 UTC+1, a source of compute.googleapis.com, and a resource of beta.compute.instances.insert. The log details show an operation with first: true, id: operation-1626084723661-5c6ea5a5418d7-afb26dc5-11a9899c, and producer: compute.googleapis.com. The protoPayload is a google.cloud.audit.AuditLog object with authenticationInfo, authorizationInfo, methodName: beta.compute.instances.insert, request, requestMetadata, resourceLocation, resourceName: projects/events-atamel/zones/us-central1-a/instances/my-instance, response, serviceName: compute.googleapis.com, receiveTimestamp: 2021-07-12T10:12:05.269943751Z, resource: {2}, severity: NOTICE, and timestamp: 2021-07-12T10:12:03.700592Z.

```
{
  insertId: "-rqayfce1c8ow"
  logName: "projects/events-atamel/logs/cloudaudit.googleapis.com%2Factivity"
  operation: {
    first: true
    id: "operation-1626084723661-5c6ea5a5418d7-afb26dc5-11a9899c"
    producer: "compute.googleapis.com"
  }
  protoPayload: {
    @type: "type.googleapis.com/google.cloud.audit.AuditLog"
    authenticationInfo: {1}
    authorizationInfo: [5]
    methodName: "beta.compute.instances.insert"
    request: {14}
    requestMetadata: {4}
    resourceLocation: {1}
    resourceName: "projects/events-atamel/zones/us-central1-a/instances/my-instance"
    response: {14}
    serviceName: "compute.googleapis.com"
  }
  receiveTimestamp: "2021-07-12T10:12:05.269943751Z"
  resource: {2}
  severity: "NOTICE"
  timestamp: "2021-07-12T10:12:03.700592Z"
}
```

The second one is emitted after the VM creation and looks like this:

```

2021-07-12 11:12:24.775 UTC+1 compute.googleapis.com beta.compute.instances.insert
{
  insertId: "-aszak6d4o5c"
  logName: "projects/events-atamel/logs/cloudaudit.googleapis.com%2Factivity"
  operation: {
    id: "operation-1626084723661-5c6ea5a5418d7-afb26dc5-11a9899c"
    last: true
    producer: "compute.googleapis.com"
  }
  protoPayload: {
    @type: "type.googleapis.com/google.cloud.audit.AuditLog"
    authenticationInfo: {1}
    methodName: "beta.compute.instances.insert"
    request: {1}
    requestMetadata: {2}
    resourceName: "projects/events-atamel/zones/us-central1-a/instances/my-instance"
    serviceName: "compute.googleapis.com"
  }
  receiveTimestamp: "2021-07-12T10:12:25.486164810Z"
  resource: {2}
  severity: "NOTICE"
  timestamp: "2021-07-12T10:12:24.775145Z"
}

```

Notice the operation field with the first: true and last: true values. The second Audit Log contains all the information you need to label an instance, therefore you will use the last: true flag to detect it in Cloud Functions.

## Setup

To use Cloud Audit Log functions, you must enable Audit Logs for Eventarc. You also need to use a service account with the eventarc.eventReceiver role.

1. From the Navigation Menu, go to **IAM & Admin > Audit Logs**.

**Note:** you can ignore the missing resourceManager folders.getIamPolicy permission warning.

2. Find the **Compute Engine API** and click the check box next to it. If you are unable to find the API, search it on next page.
3. On the info pane on the right, check **Admin Read**, **Data Read**, and **Data Write** log types and click **Save**.

Audit Logs

SET DEFAULT CONFIGURATION

HELP ASSISTANT

LEARN

HIDE INFO PANEL

Filter Enter property name or value

Service	Admin Read	Data Read	Data Write
<input type="checkbox"/> Cloud Spanner API	—	—	—
<input type="checkbox"/> Cloud SQL	—	—	—
<input type="checkbox"/> Cloud Storage for Firebase API	—	—	—
<input type="checkbox"/> Cloud Tasks API	—	—	—
<input type="checkbox"/> Cloud TPU API	—	—	—
<input type="checkbox"/> Cloud Trace API	—	—	—
<input type="checkbox"/> Cloud Translation API	—	—	—
<input checked="" type="checkbox"/> Compute Engine API	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/> Connectors API	—	—	—
<input type="checkbox"/> Contact Center AI Insights API	—	—	—
<input type="checkbox"/> Contact Center AI Platform API	—	—	—

Compute Engine API

LOG TYPES EXEMPTED PRINCIPALS

Enable/disable Data Access audit log types for selected services.

☒ Admin Read  
☒ Data Read  
☒ Data Write

SAVE

4. Grant the default Compute Engine service account the eventarc.eventReceiver IAM role:



```
gcloud projects add-iam-policy-binding $PROJECT_ID \
--member serviceAccount:$PROJECT_NUMBER-
compute@developer.gserviceaccount.com \
--role roles/eventarc.eventReceiver
```

## Get the code

1. Run the following code to clone the repo that contains the application:

```
cd ~ & git clone https://github.com/GoogleCloudPlatform/eventarc-samples.git
```

2. Navigate to the app directory:

```
cd ~/eventarc-samples/gce-vm-labeler/gcf/nodejs
```

The `index.js` file contains the application code that receives the Audit Log wrapped into a CloudEvent. It then extracts the Compute Engine VM instance details and sets a label on the VM instance. Feel free to study `index.js` in more detail on your own.

## Deploy

1. Deploy the function with `gcloud` as before. Notice how the function is filtering on Audit Logs for Compute Engine insertions with the `--trigger-event-filters` flag:

```
gcloud functions deploy gce-vm-labeler \
--gen2 \
--runtime nodejs16 \
--entry-point labelVmCreation \
--source . \
--region $REGION \
--trigger-event-
filters="type=google.cloud.audit.log.v1.written,serviceName=compute.googleapis.com,methodName=beta.compute.instances.insert" \
--trigger-location $REGION \
--max-instances 1 Note: Although your Audit Log function trigger will be created immediately, it can
take up to 10 minutes for triggers to be fully functional.
```

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

Create a Cloud Audit Logs Function

## Test

To test your Audit Log function, you need to create a Compute Engine VM in the Cloud Console (You can also create VMs with `gcloud` but it does not seem to generate Audit Logs).

1. From the Navigation Menu, go to **Compute Engine > VM instances**.
2. Click **Create Instance**.
3. Leave all of the fields as the default values and click **Create**.

Once the VM creation completes, you should see the added creator label on the VM in the Cloud Console in the **Basic information** section.



## Basic information

Name	instance-1
Instance Id	5865041818852610616
Description	None
Type	Instance
Status	✓ Running
Creation time	Sep 19, 2022, 3:27:38 PM UTC-07:00
Zone	us-central1-a
Instance template	None
In use by	None
Reservations	Automatically choose
Labels	creator : student-02...
Deletion protection	Disabled
Confidential VM service ?	Disabled
Preserved state size	0 GB

4. Verify using the following command:

`gcloud compute instances describe instance-1 --zone {{project_0.default_zone | ""}}` **Note:** This assumes your VM is named `instance-1` and is in the zone. If you modified these parameters, be sure to update the command.

You should see the label in the output like the following example:

```
... labelFingerprint: ULU6pAy2C7s= labels: creator: student-02-19b599a0f901 ...
```

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

Create a VM Instance

## Task 5. Deploy different revisions

Cloud Functions (2nd gen) supports multiple revisions of your functions, as well as splitting traffic between different revisions and rolling your function back to a prior version. This is possible because 2nd gen functions are Cloud Run services under the hood. In this section, you will deploy two revisions of your Cloud Function.

### Create

1. Run the following command to create the folder and files for the app and navigate to the folder:

```
mkdir ~/hello-world-colored && cd $_ touch main.py
```

2. Add the following code to the `hello-world-colored/main.py` file with a Python function that reads a color environment variable and responds back with Hello World in that background color:

```
import os color = os.environ.get('COLOR') def hello_world(request): return f'<body style="background-color:{color}"><h1>Hello World!</h1></body>'
```

### Deploy

1. Deploy the first revision of the function with an orange background:

```
COLOR=orange gcloud functions deploy hello-world-colored \ --gen2 \ --runtime python39 \ --entry-point hello_world \ --source . \ --region $REGION \ --trigger-http \ --allow-unauthenticated \ --update-env-vars COLOR=$COLOR \ --max-instances 1
```

At this point, if you test the function by viewing the HTTP trigger (the URI output of the above deployment command) in your browser, you should see Hello World with an orange background:



# Hello World!

2. Navigate to the **Cloud Functions** page in the Console and click the **hello-world-colored** function.

3. On the top right of the page under **Powered by Cloud Run** click **hello-world-colored**.

This will re-direct you to the Cloud Run service page.

4. Click the **Revisions** tab and then select **Edit & Deploy New Revision**.

5. Leave everything as default and scroll down to the **Environment Variables** section. Update the **COLOR** environment variable to **yellow**.

## Environment variables

Name 1

COLOR

e.g. ENV

Value 1

yellow

e.g. prod

[+ ADD VARIABLE](#)

## Secrets ?

[REFERENCE A SECRET](#)

☒ Serve this revision immediately

100% of the traffic will be migrated to this revision, overriding all existing traffic splits, if any.

DEPLOY

CANCEL

6. Click **Deploy**.

Since this is the latest revision, if you test the function, you should see Hello World with a yellow background:

# Hello World!

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

Deploy different revisions

## Task 6. Set up minimum instances

In Cloud Functions (2nd gen), one can specify a minimum number of function instances to be kept warm and ready to serve requests. This is useful in limiting the number of cold starts. In this section, you will deploy a function with slow initialization. You'll observe the cold start problem. Then, you will deploy the function with the minimum instance value set to 1 to get rid of the cold start.

### Create

1. Run the following command to create the folder and files for the app and navigate to the folder:

```
mkdir ~/min-instances && cd $_ touch main.go
```

2. Add the following code to the min-instances/main.go file. This Go service has an init function that sleeps for 10 seconds to simulate a long initialization. It also has a HelloWorld function that responds to HTTP calls:

```
package p import ( "fmt" "net/http" "time" ) func init() { time.Sleep(10 * time.Second) } func HelloWorld(w http.ResponseWriter, r *http.Request) { fmt.Fprint(w, "Slow HTTP Go in GCF 2nd gen!") }
```

### Deploy

1. Run the following command to deploy the first revision of the function with the default minimum instance value of zero:

```
gcloud functions deploy slow-function \ --gen2 \ --runtime go116 \ --entry-point HelloWorld \ --source . \ --region $REGION \ --trigger-http \ --allow-unauthenticated \ --max-instances 4
```

2. Test the function with this command:

```
gcloud functions call slow-function \ --gen2 --region $REGION
```

You should observe a 10 second delay (cold start) on the first call and then see the message. Subsequent calls should return immediately.

```
Slow HTTP Go in GCF 2nd gen!
```

### Set minimum instances

To get rid of the cold start on the first request, redeploy the function with the `--min-instances` flag set to 1 as follows:

1. Navigate to the **Cloud Run** page in the Console and click the **slow-function** service.
2. Click the **Revisions** tab and then select **Edit & Deploy New Revision**.
3. Under the **Autoscaling** section, set **Minimum number of instances** to 1.
4. Leave the rest of the fields as default and click **Deploy**.

### Test

- ```
gcloud functions call slow-function \ --gen2 --region $REGION
```

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

### Task 7. Create a function with concurrency

To fix the cold-start problem, you will deploy another function with a concurrency value of 100. You will observe that the 10 requests now do not cause the cold start problem and a single function instance can handle all the requests.

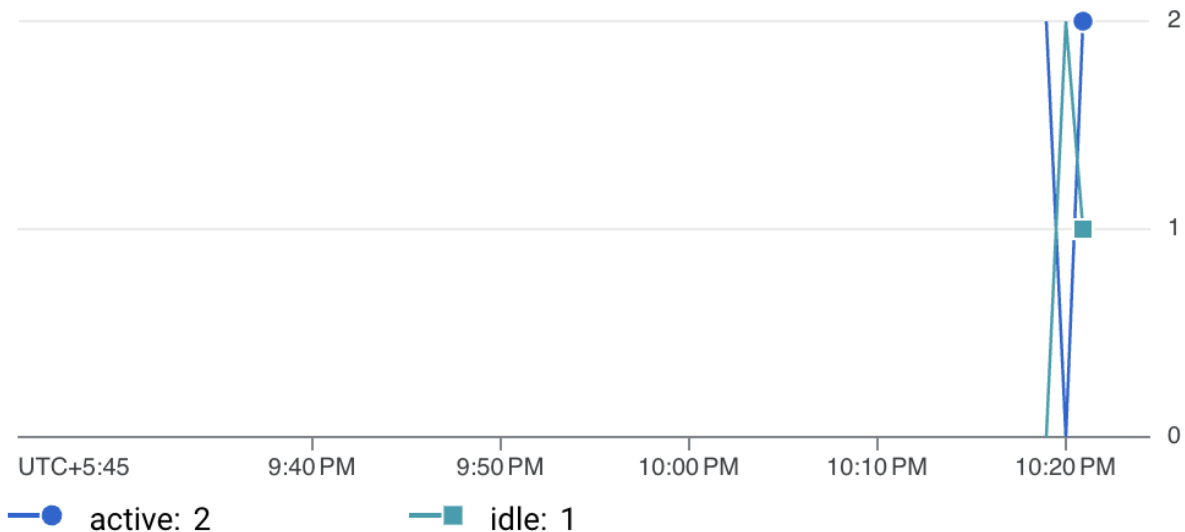
1. Run the following command to get the URL of the function and save it as an environment variable:

2. Use an open source benchmarking tool called `hey` to send 10 concurrent requests to the slow function. `hey` is already installed in Cloud Shell:

You should see in the output of `hey` that some requests are taking long:

This is because more function instances are being created to handle the requests. If you check the active instances count for the function, you should also see that more than one instance was created as some point and these are causing the cold start problem:

## Instance count



## Deploy

1. Deploy a new function identical to the previous function. Once deployed, you will increase its concurrency:

```
gcloud functions deploy slow-concurrent-function \
  --gen2 \
  --runtime go116 \
  --entry-point HelloWorld \
  --source . \
  --region $REGION \
  --trigger-http \
  --allow-unauthenticated \
  --min-instances 1 \
  --max-instances 4
```

## Set concurrency

Now you will set the concurrency of the underlying Cloud Run service for the function to 100 (it can be maximum 1000). This ensures that at least 100 requests can be handled by a single function instance.

1. From the Navigation Menu, go to **Cloud Run**.
2. Click the **slow-concurrent-function** service.
3. Click the **Revisions** tab and then select **Edit & Deploy New Revision**.
4. Under the **Capacity** section, set the following:
  - **CPU:** 1
  - **Maximum concurrent requests per instance:** 100
5. Leave the rest of the fields as default and click **Deploy**.

## Test with concurrency

1. Once your function has deployed, run the following command to get the URL of the new function and save it as an environment variable:

```
SLOW_CONCURRENT_URL=$(gcloud functions describe slow-concurrent-function --region $REGION --gen2 --
format="value(serviceConfig.uri)")
```

2. Now use **hey** to send 10 concurrent requests:

Copyright 2023 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.