# Chapter 5 - Google Cloud's Operations Suite

# Monitoring Multiple Projects with Cloud Monitoring

1 hourFree

Rate Lab

## GSP090



## Overview

Cloud Monitoring provides dashboards and alerts so you can review performance metrics for cloud services, virtual machines, and common open source servers such as MongoDB, Apache, Nginx, Elasticsearch, and more. You configure Cloud Monitoring in the Console.

In this hands-on lab you will have 2 projects to monitor in Cloud Monitoring. You'll add them both to a Cloud Monitoring account and monitor the metrics the virtual machines in the projects provide.

### **Objectives**

* Create a Cloud Monitoring account that has two Google Cloud projects.
* Monitor across both projects from the single Cloud Monitoring account.

## Setup

#### 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:

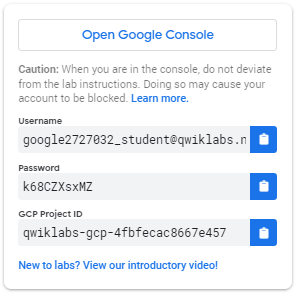
* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

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

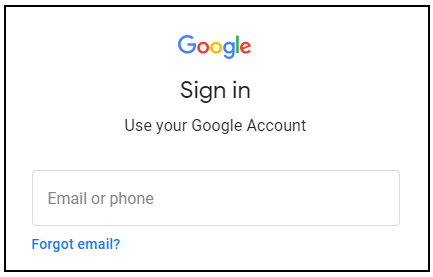
**Note:** If you are using a Pixelbook, open an Incognito window to run this lab.

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

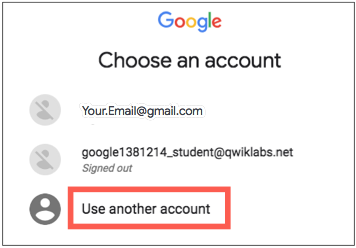
1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



1. 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.

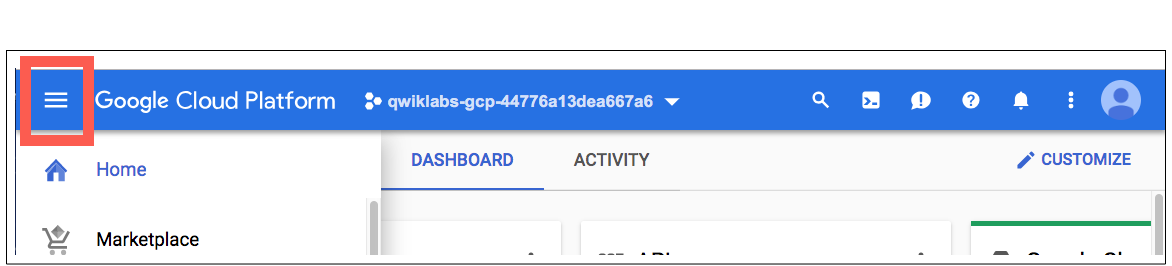
If you see the **Choose an account** page, click **Use Another Account**. 

1. 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).

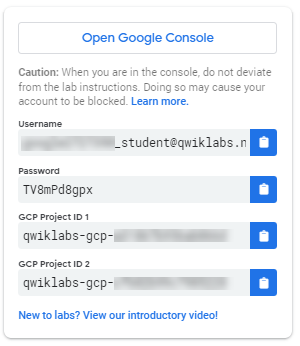
1. Click through the subsequent pages:
   * Accept the terms and conditions.
   * Do not add recovery options or two-factor authentication (because this is a temporary account).
   * Do not sign up for free trials.

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. 

## Setup for two projects

For this lab you are given two Project IDs. When you logged in, by default you logged in to Project 1. You'll need to keep track of your projects, and you can return to this page to remind yourself which is which. The projects will change order, so knowing the last few digits of the name will help you identify them.



Project 1 already has a virtual machine (and you can look at it by going to **Compute Engine** > **VM instances**). You will create a virtual machine in Project 2, and then monitor both projects in Cloud Monitoring.

## Create Project 2's virtual machine

At the top of the screen, click on the dropdown arrow next to Project 1's name.



Make sure that you're on the **All** tab, then click on the name of Project 2 to go into it. 

Select **Navigation menu** > **Compute Engine** to open the VM instances window.

Click **Create** to create a new instance.



Name this instance **instance2**.

Leave all of the options at the default settings.

Click **Create**.

Now you have resources to monitor in both of your projects.

### **Test Completed Task**

Click **Check my progress** to verify your performed task. If you have completed the task successfully you will granted with an assessment score.

Create Project 2's virtual machine

Check my progress

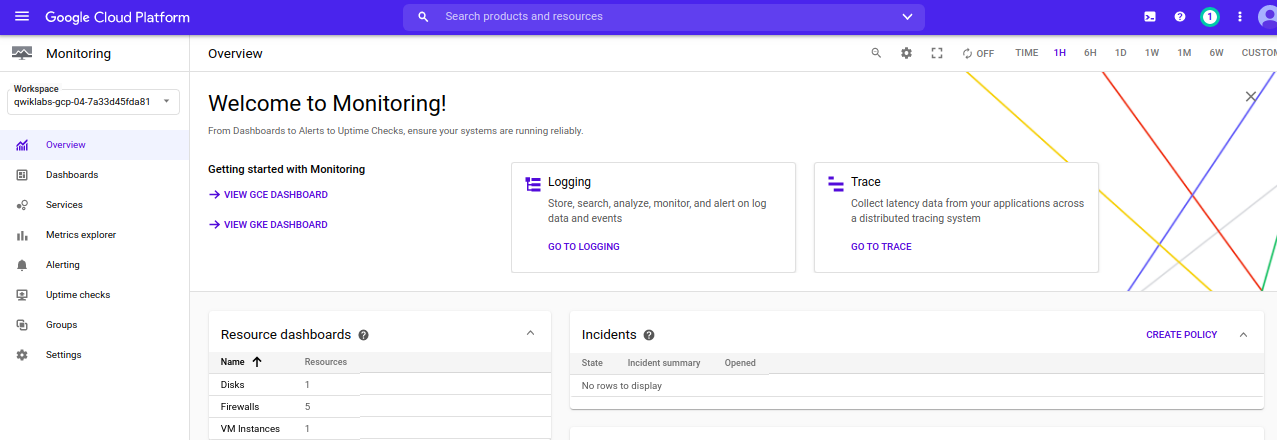
Make sure that you are in Project 2 to proceed further in the lab

### **Create a Monitoring workspace**

Now set up a Monitoring workspace that's tied to your Google Cloud Project. The following steps create a new account that has a free trial of Monitoring.

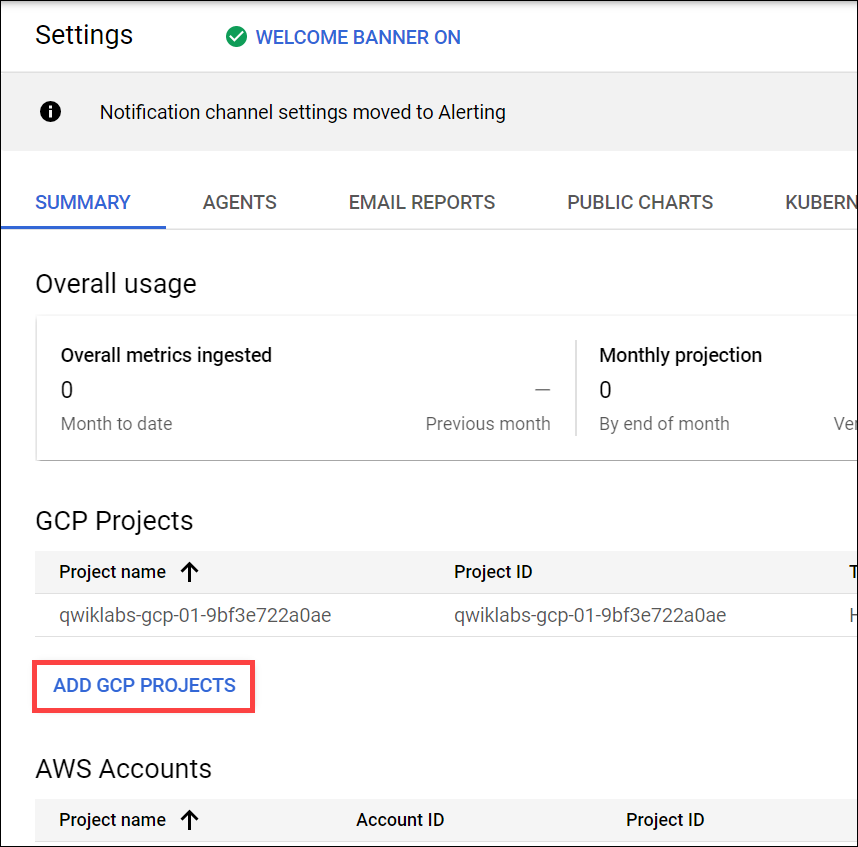
1. In the Cloud Console, click **Navigation menu** > **Monitoring**.
2. Wait for your workspace to be provisioned.

When the Monitoring dashboard opens, your workspace is ready.

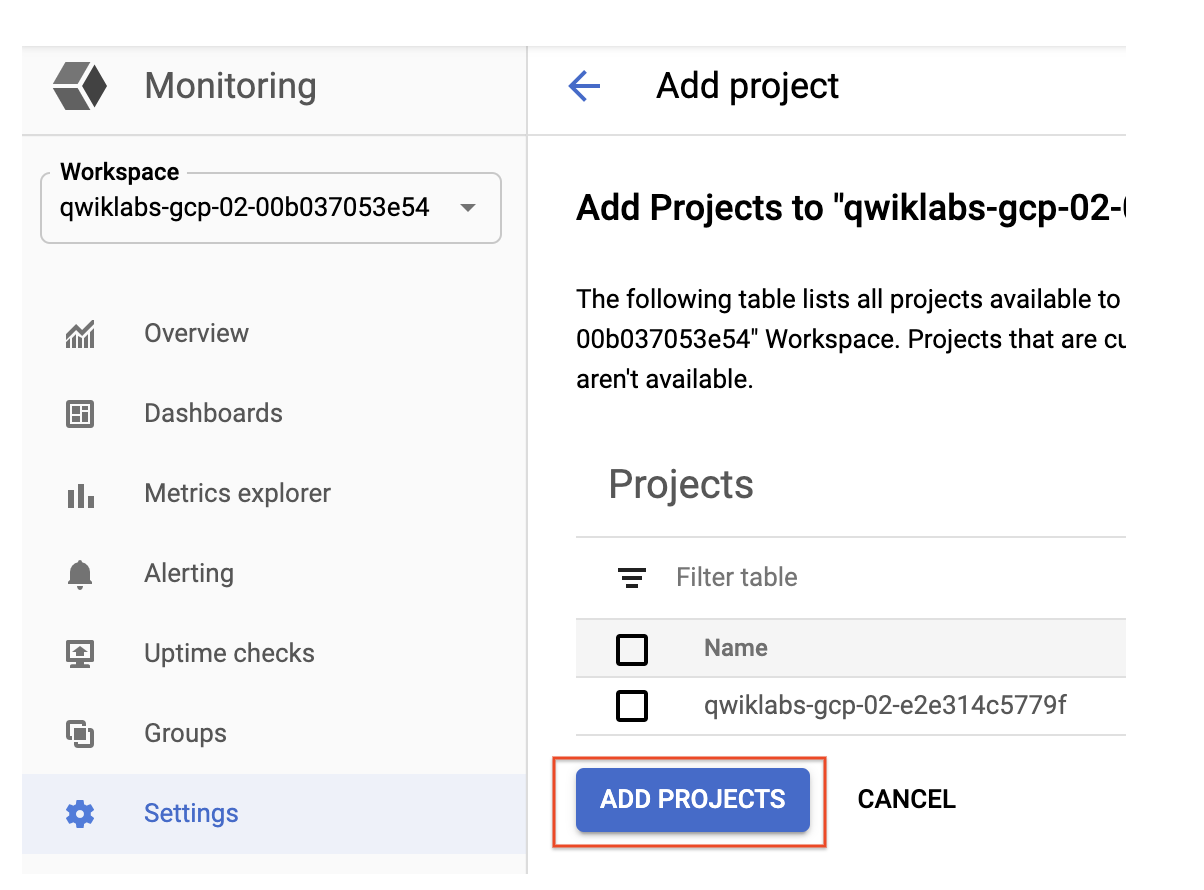


Now add both projects to your Cloud Monitoring workspace.

In the left menu, click **Settings** and then click **Add GCP Projects** in the GCP Projects section.



You'll see one of the projects for this lab as a monitored account. Check the box next to the other Project you have, then **Add projects**.



## Monitoring Overview

Click on **Overview** in the left menu. You'll be adding a lot of good information here as the lab goes along. First, you'll create a [Cloud Monitoring Group](https://cloud.google.com/monitoring/groups/) for visibility across both projects.

### **About Cloud Monitoring Groups**

Cloud Monitoring lets you define and monitor groups of resources, such as VM instances, databases, and load balancers. Groups can be based on names, tags, regions, applications, and other criteria. You can also create subgroups, up to six levels deep, within groups.

### **Create a Cloud Monitoring Group**

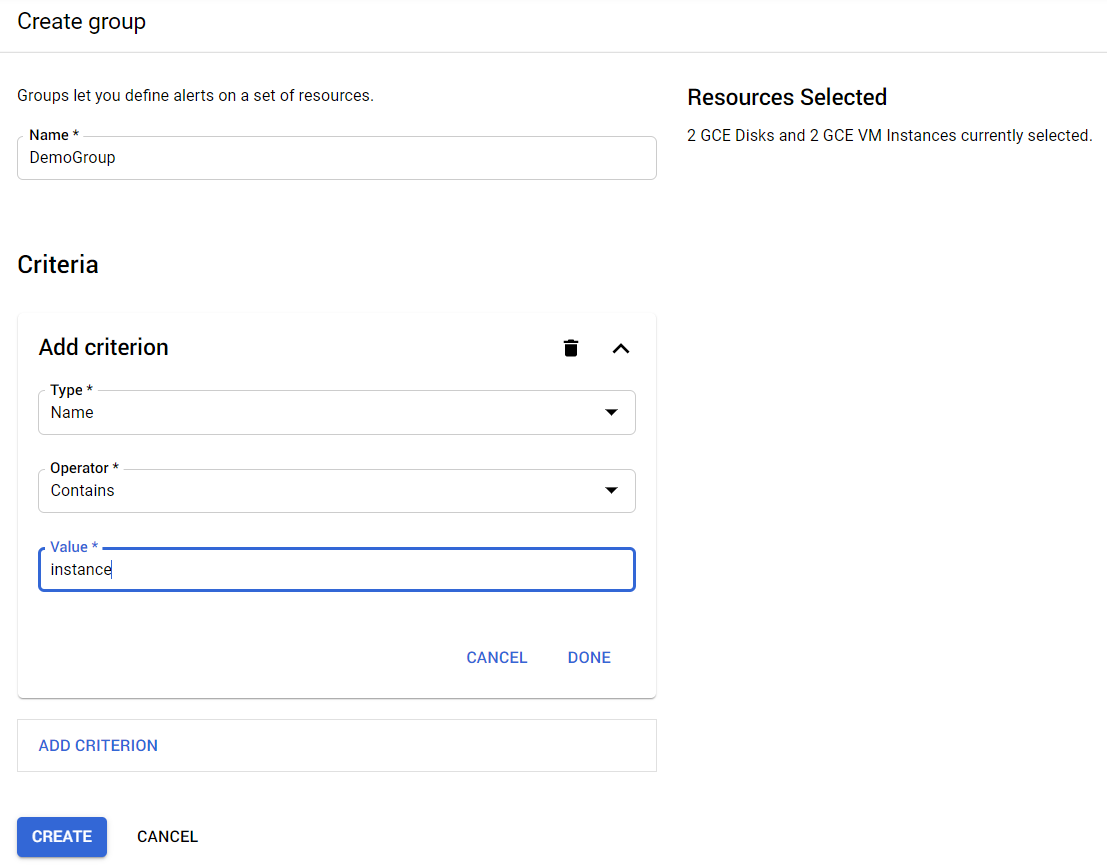
In the left menu, click **Groups**, and then click **Create Group**.

Name your group **DemoGroup**.

The **Criteria** is a set of rules that will dynamically evaluate which resources should be part of this group.

Cloud Monitoring dynamically determines which resources belong to your group based on the filter criteria that you set up.

* In the first dropdown field (Type), **Name** is selected by default.
* In the second dropdown (Contains), **Contains** is selected by default.
* In the third field (Value), type in "instance" since both of the instance names in both of your projects start with the word instance.



Click **Done**, then click **Create**.

#### Test Completed Task

Click **Check my progress** to verify your performed task. If you have completed the task successfully you will granted with an assessment score.

Create a Cloud Monitoring Group

Check my progress

## Uptime Check for your group

Uptime checks let you quickly verify the health of any web page, instance, or group of resources. Each configured check is regularly contacted from a variety of locations around the world. Uptime checks can be used as conditions in alerting policy definitions.

In the left menu, click **Uptime Checks**, and then click **Create Uptime Check**.

Create your uptime check with the following information:

**Title:** DemoGroup uptime check, then click **Next**.

**Protocol:** TCP

**Resource Type:** Instance

**Applies To:** Group, and then select **DemoGroup**.

**Port:** 22

**Check frequency:** 1 minute, then click **Next**.

Click **Next** again.

Put the slider in **off** state for **Create an alert** option in **Alert & Notification** section.



Click **Test** to verify that your uptime check can connect to the resource.

When you see a green check mark everything can connect, click **Create**.

### **Test Completed Task**

Click **Check my progress** to verify your performed task. If you have completed the task successfully you will granted with an assessment score.

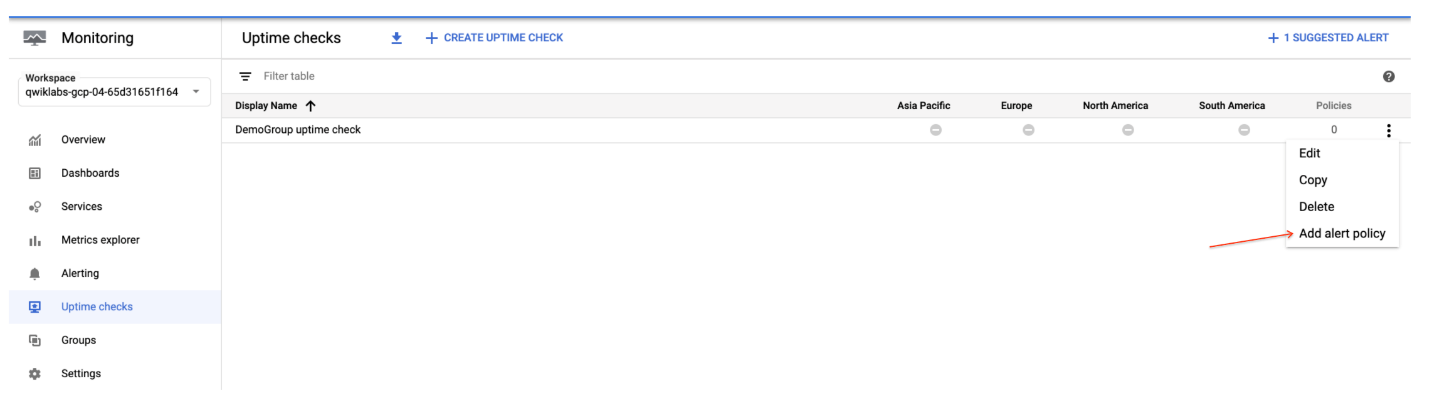
Uptime Check for your group

Check my progress

## Alerting Policy for the group

Use Cloud Monitoring to create one or more alerting policies.

In the left menu, click **Uptime Checks**, and then click **Add alert policy**.



The **Condition** is already set. Metadata has been pulled in from the uptime check to create it.



**Name this uptime check:** You can use the "Suggested title" or type in your own.

Click **Save**.

Click **Next**.

Skip the **Notification channels** option and click **Next**.

In the **Alert name** field, enter the **Name** as **Uptime Check Policy**.

Click **Save**.

### **Test Completed Task**

Click **Check my progress** to verify your performed task. If you have completed the task successfully you will granted with an assessment score.

Alerting Policy for the group

Check my progress

## Custom dashboard for your group

Create a custom dashboard so you can monitor your group easily.

In the left menu, click **Dashboards**, and then click **Create Dashboard**.

Name your dashboard.

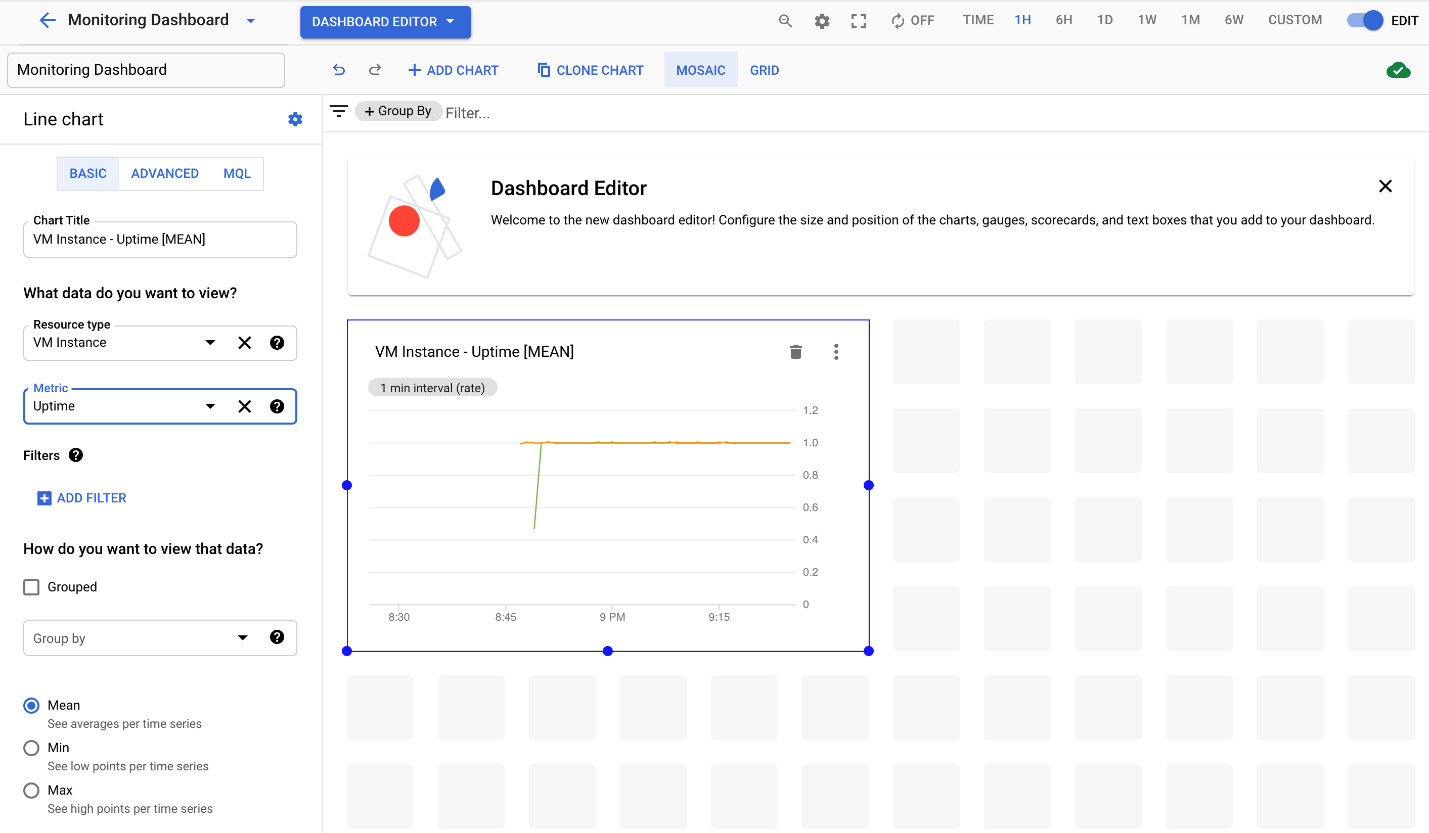
Click **Line** option in Chart library to add the first chart.

Leave the **Chart Title** as default.

Select **VM Instance** in Resource type field.

Start typing Uptime into the Metric field, then select **compute.googleapis.com/instance/uptime** from the offered metrics.

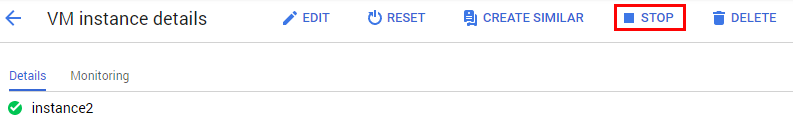
The dashboard should look like:



## Remove one instance to cause a problem

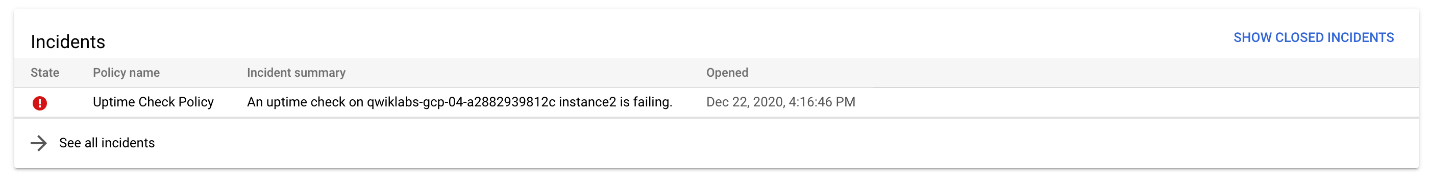
In the console, select **Navigation menu** > **Compute Engine**.

Check the box next to **instance2**, then click **Stop** at the top of the page, then **Stop** again to turn off the machine.



Wait a minute or 2 for the instance to stop and violate the uptime check you just set up. After a couple of minutes, turn your machine back on by clicking **Start/Resume**, then **Start**.

Click **Navigation menu** > **Monitoring** > **Alerting** and refresh your browser. It may take a few more minutes to show that you have issues in the Summary section. Refresh until your screen looks similar to this:



**Optional:** Using the left menu, look at **Dashboards** to view your custom dashboard. That provides details on both VMs. If you mouse over your chart, you can see which of your instances was stopped and restarted.

### **Incidents**

When the alerting policy conditions are violated, an "incident' is created and displayed in the Incident section.

Responders can acknowledge receipt of the notification and can close the incident when it has been taken care of.

In the Incidents section, click on the name of the alerting policy that was violated to go into it.

You've already **fixed** your problem by turning the VM back on, so the incident was cleared and you no longer see an incident in the Incidents section.

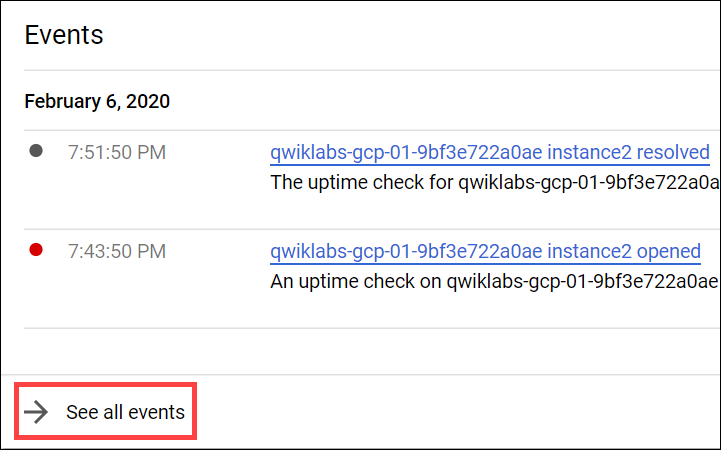
To see the cleared incident, scroll down to Events and click on the **"resolved"** link.

Your incident should have a **Closed** status. You can read through the incident details.

You can also click on the **Uptime Check Policy** link to explore the metrics it gives you.

In several more minutes the Monitoring Overview page will all go back to green when the instance in Project 2 passes the Uptime Check.

Return to the Alerting page (click **Alerting** in the left menu). In the **Events** section you'll be able to see what incident happened and its resolution. In a production environment you can use the Filter to display only the Events you need to see. Click **See all events** at the bottom to see all the events.



In the Events window, click **Show Filters** to manually add information that might not be captured otherwise.

## Test your Understanding

Below are multiple-choice questions to reinforce your understanding of this lab's concepts. Answer them to the best of your abilities.

Cloud Monitoring lets you define and monitor groups of resources with Cloud Monitoring Group.



True



False

# Monitoring and Logging for Cloud Functions

45 minutesFree

## GSP092



You can [view your Cloud Functions](https://cloud.google.com/functions) with their execution times,execution counts, and memory usage in the Cloud Console using [Cloud Monitoring](https://cloud.google.com/monitoring), where you can set up custom alerting on these metrics.

## 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:

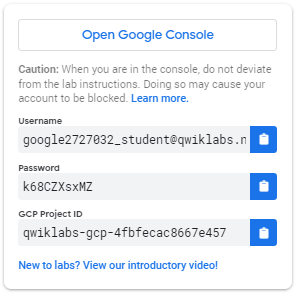
* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

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

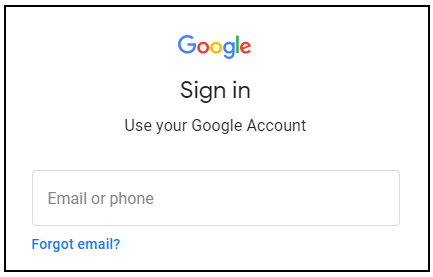
**Note:** If you are using a Pixelbook, open an Incognito window to run this lab.

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

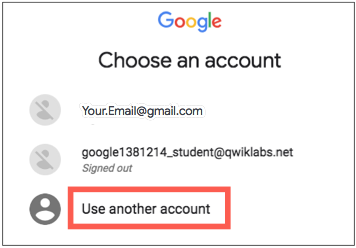
1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



1. 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.

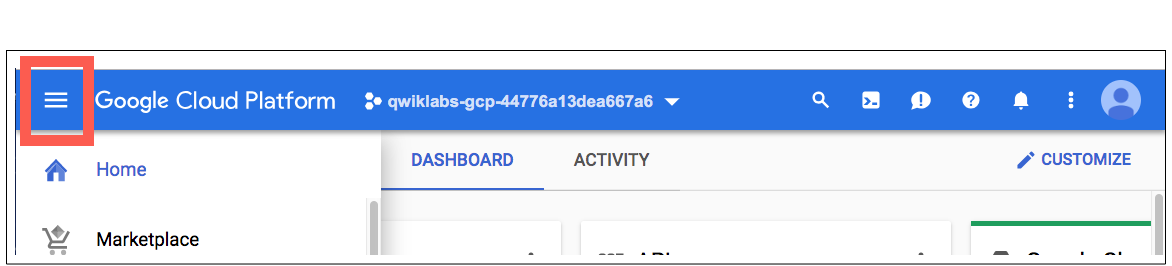
If you see the **Choose an account** page, click **Use Another Account**. 

1. 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).

1. 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.

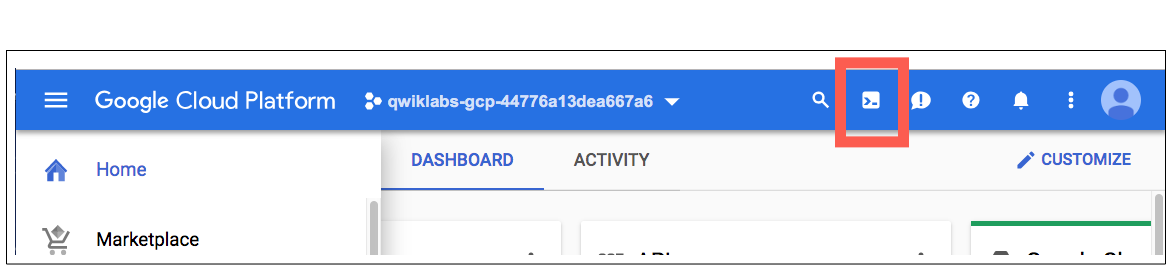
After a few moments, the Cloud Console opens in this tab.

**Note:** You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left. 

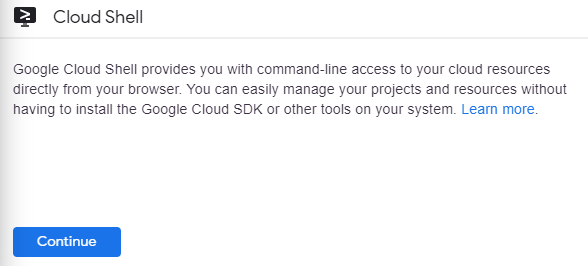
### **Activate Cloud Shell**

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.

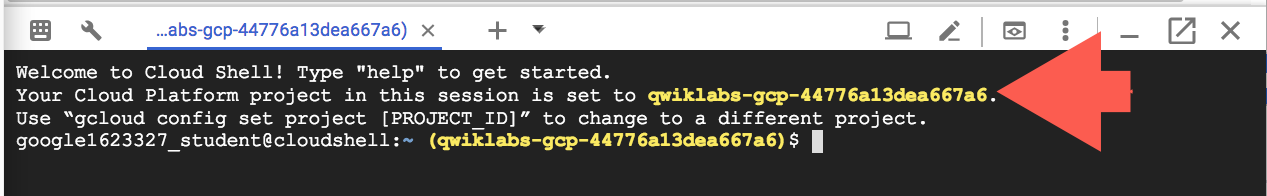
In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



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:



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

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]

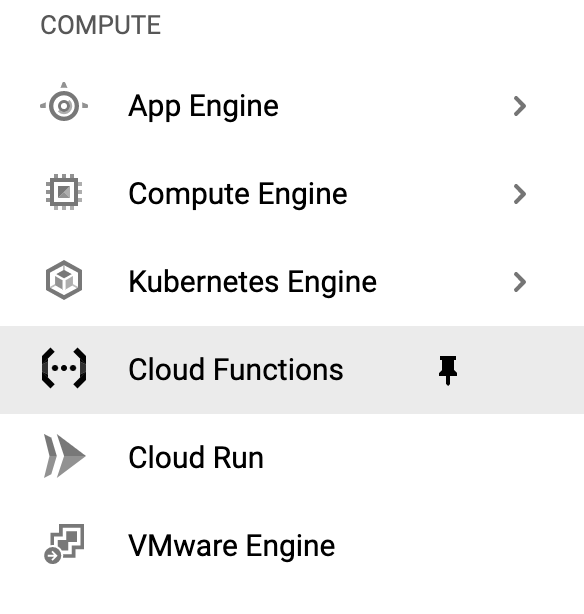
project = qwiklabs-gcp-44776a13dea667a6content\_copy

For full documentation of gcloud see the [gcloud command-line tool overview](https://cloud.google.com/sdk/gcloud" \t "_blank).

## Viewing Cloud Function logs & metrics in Cloud monitoring

Before you collect logs and alerts, you need something to monitor. In this section, you create a Hello World cloud function to monitor.

1. In the Cloud Console, select **Navigation menu** > **Cloud Functions**, and then **Create function**.

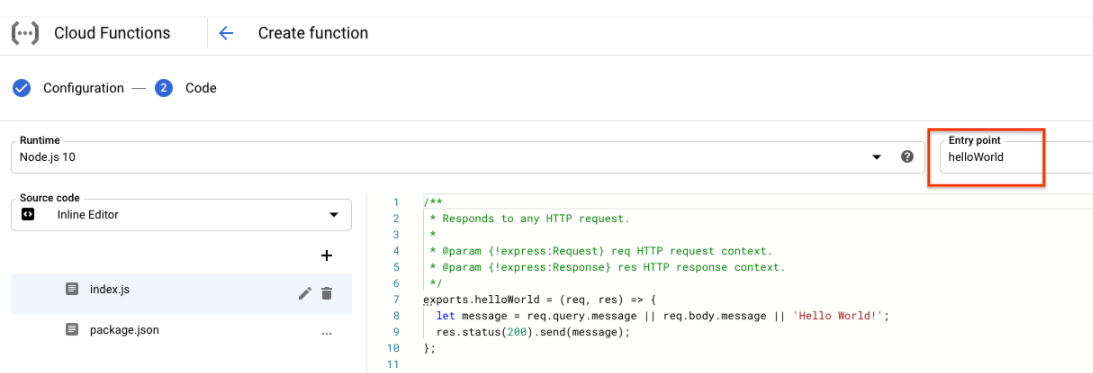


1. Set the following:

* **Function Name:** helloWorld
* **Trigger type:** HTTP
* **Authentication**: check the box next to **Allow unauthenticated invocations**

Click **Save**, then click **Next**.

1. You will see the following:



1. Click **Deploy**.

The cloud function automatically deploys and is listed on the Cloud Function page. This takes a few minutes. When you see a green check mark next to the name, the cloud function is complete.

### **Test Completed Task**

Click **Check my progress** to verify your performed task. If you have completed the task successfully you will granted with an assessment score.

Creating a Cloud Function

Check my progress

1. In Cloud Shell, run the following to get a tool called **vegeta** that will let you send some test traffic to your cloud function:

wget 'https://github.com/tsenart/vegeta/releases/download/v6.3.0/vegeta-v6.3.0-linux-386.tar.gz'

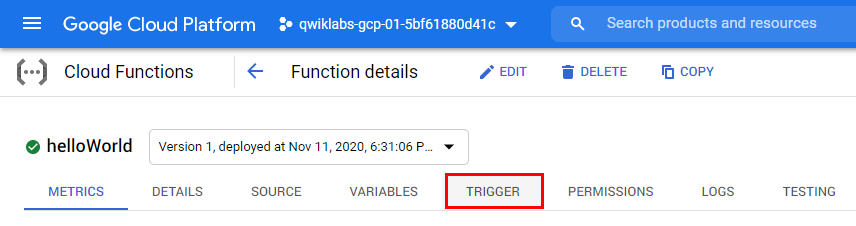
content\_copy

1. Unpack the **vegeta** tool by running the following:

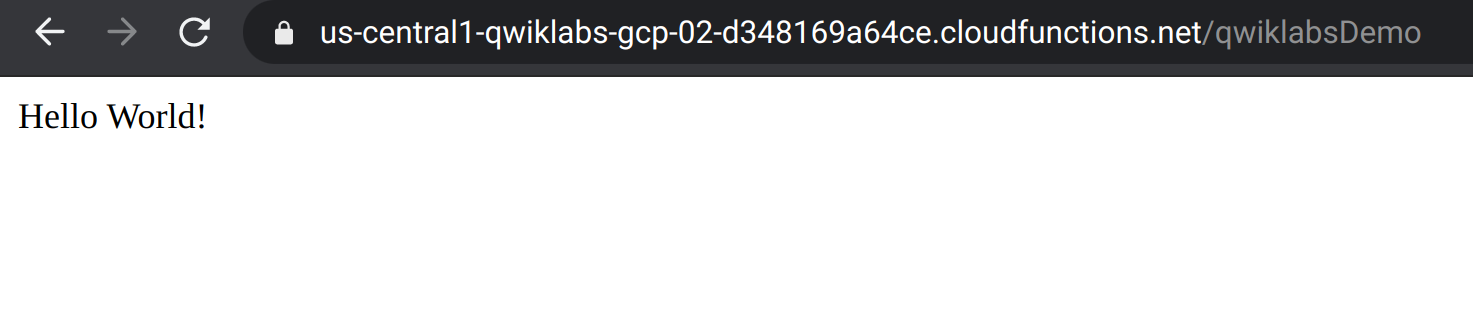
tar xvzf vegeta-v6.3.0-linux-386.tar.gz

content\_copy

1. Still in the Cloud Functions page, click the name of your function, and then click on the Trigger tab. Click the Trigger URL for your function.



If you see Hello World! in the new browser tab that opens, you're up and running!



1. Now send traffic to your cloud function. Run the following in Cloud Shell, replacing <YOUR\_PROJECT\_ID> with your Project ID and qwiklabsDemo with your function name.

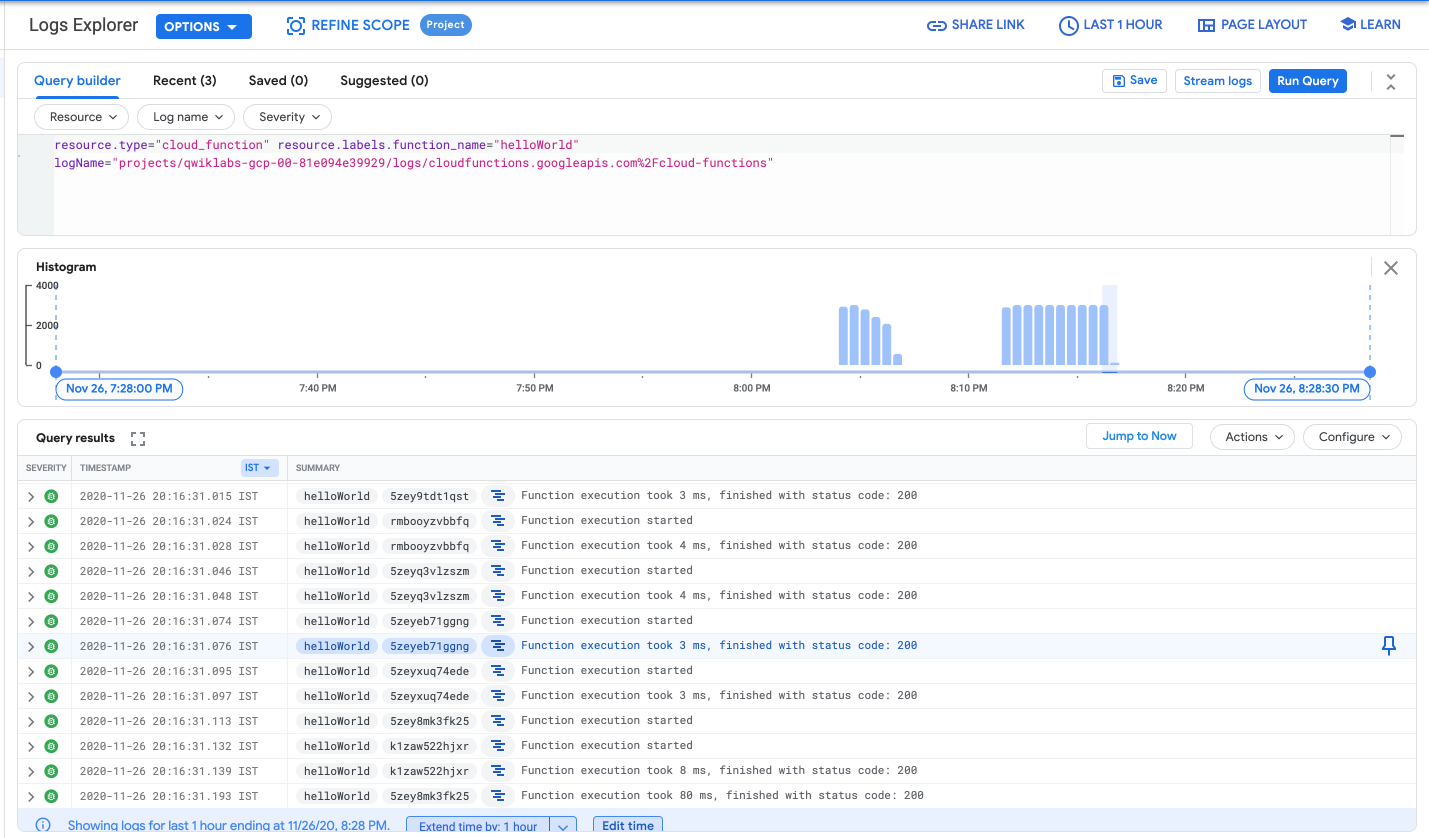
echo "GET https://us-central1-<YOUR\_PROJECT\_ID>.cloudfunctions.net/qwiklabsDemo" | ./vegeta attack -duration=300s > results.bin

content\_copy

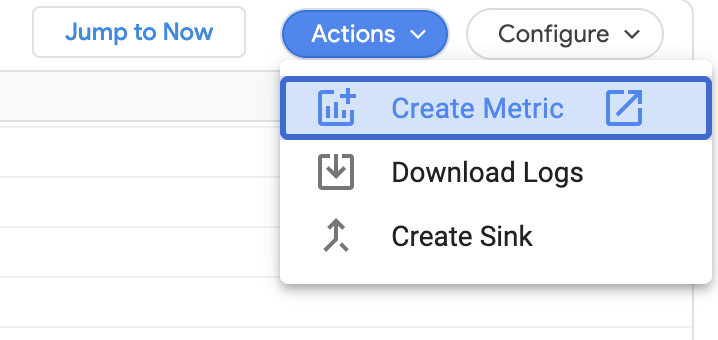
## Create logs-based metric

Now you'll create a Distribution type logs based metric using a regular expression to extract the value of latency from the log entries textPayload field.

1. In the Console, select **Navigation menu** > **Logging** > **Logs Explorer** the Cloud Logging opens in the Console.
2. To look at just the logs from your Cloud Function, in the **Resource** dropdown, select **Cloud Function** > **helloWorld** then click **Add**. In the **Log name** dropdown, select **cloud-functions** checkbox then click **Add**:
3. Click **Run Query**.



1. In the **Actions** dropdown, click **Create Metric**.



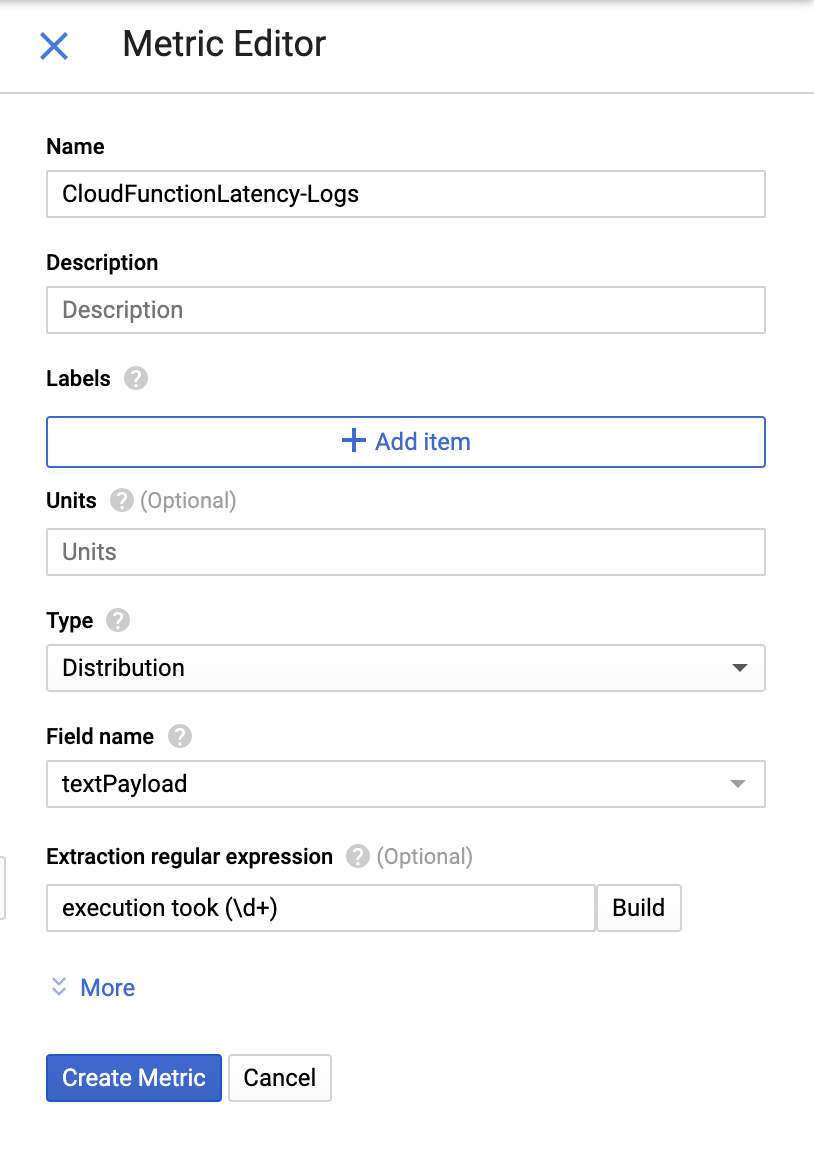
1. In the Metric Editor:

* Name your metric **CloudFunctionLatency-Logs**.
* Change the Type to **Distribution**.
* Enter **textPayload** for Field name.
* Click **Build** next to the Extraction regular expression field.
* Enter the following in the Regular Expression field, then click **Done**.

execution took (\d+)

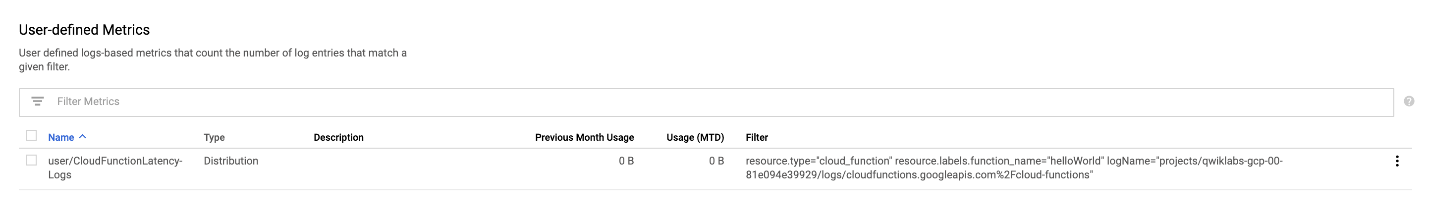
content\_copy

The Metric Editor should look like this:



1. Click **Create Metric**.

Now you'll see your user-defined metric added to your Logs-based Metrics page.



### **Test Completed Task**

Click **Check my progress** to verify your performed task. If you have completed the task successfully you will granted with an assessment score.

Create logs-based metric

Check my progress

## Metrics Explorer

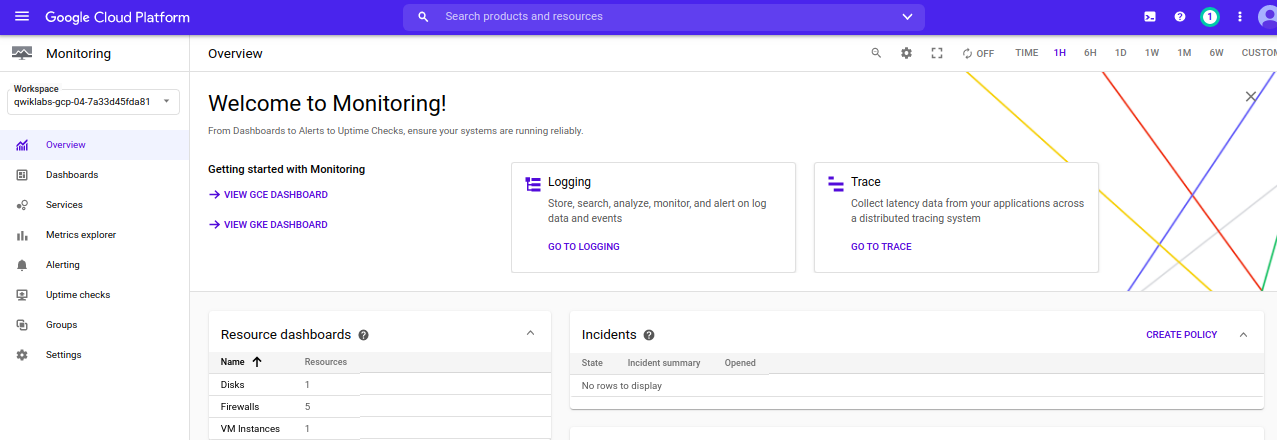
Next, use Metrics Explorer to look at the data for your cloud function.

### **Create a Monitoring workspace**

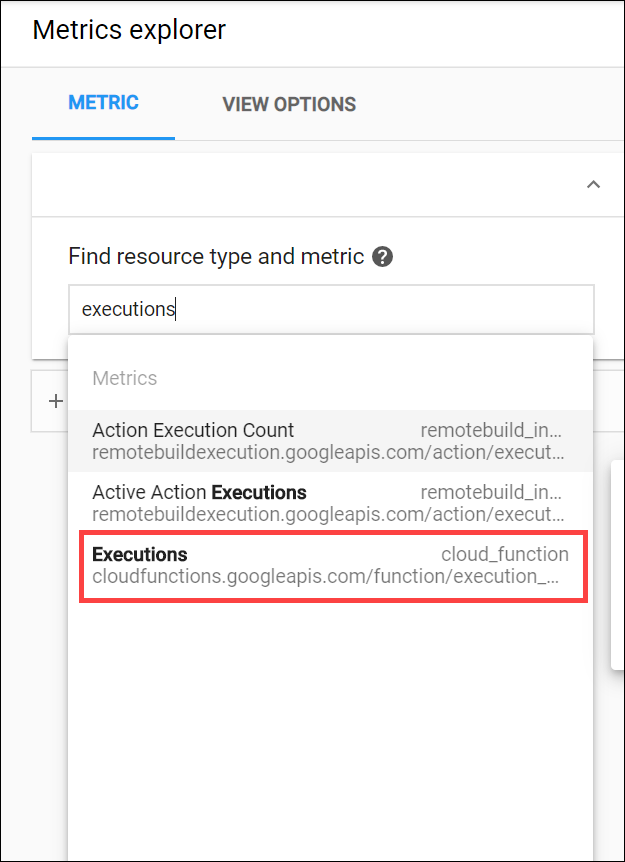
Now set up a Monitoring workspace that's tied to your Google Cloud Project. The following steps create a new account that has a free trial of Monitoring.

1. In the Cloud Console, click **Navigation menu** > **Monitoring**.
2. Wait for your workspace to be provisioned.

When the Monitoring dashboard opens, your workspace is ready.



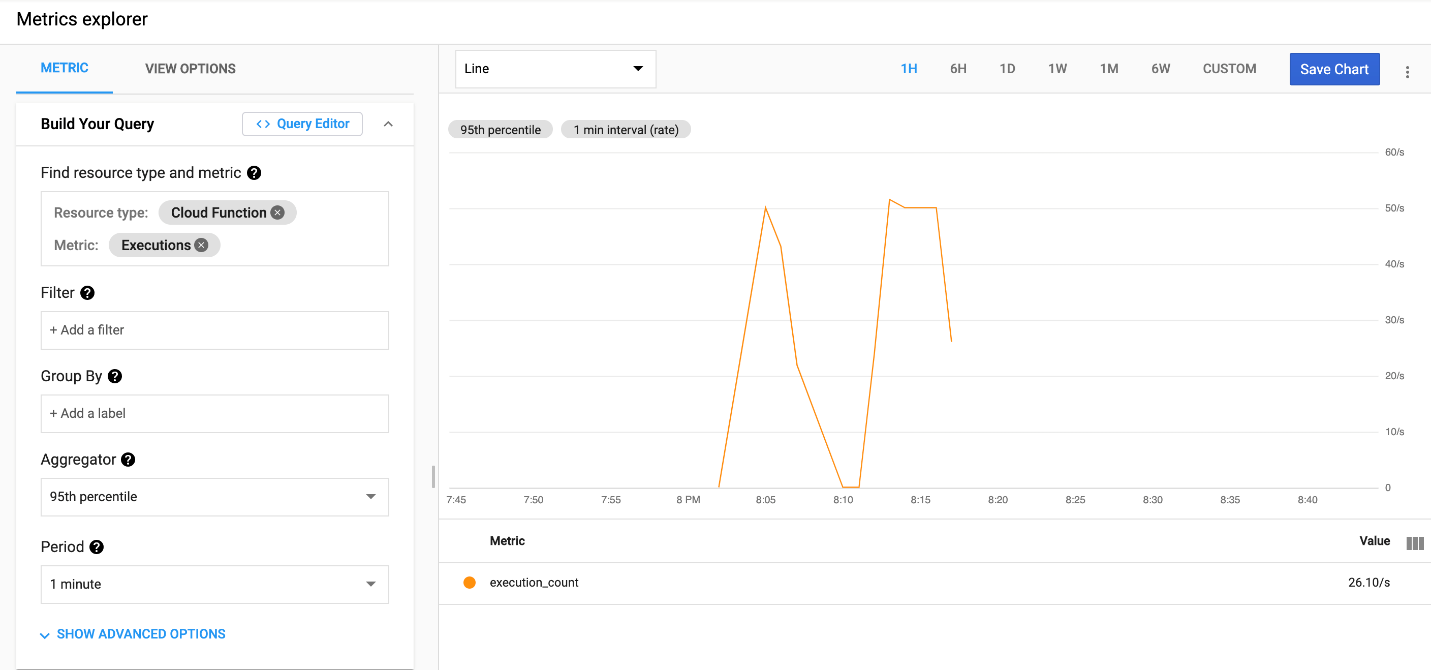
1. In the left menu, click **Metrics explorer**.
2. Start typing executions into the Find resource type and metric field, and then select **Executions** from the suggested metrics.



1. Change the graph type to **Stacked Bar** using the dropdown menu above the graph.
2. Explore other graph options, try a different metric. For example, click the **X** next to the Metric, select **Execution times**, and change the graph type to **Heatmap**.



1. Continue to explore and experiment. For example, go back to the **Executions** metric and change the aggregator to the **95th percentile**. Select the graph type **Line**.



## Create charts on the Monitoring Overview window

Creating charts on the Monitoring Overview window is a great way to track metrics that are important to you. In this section, you set up the same charts you created in the previous section, but now they'll be saved into the Monitoring Overview window.

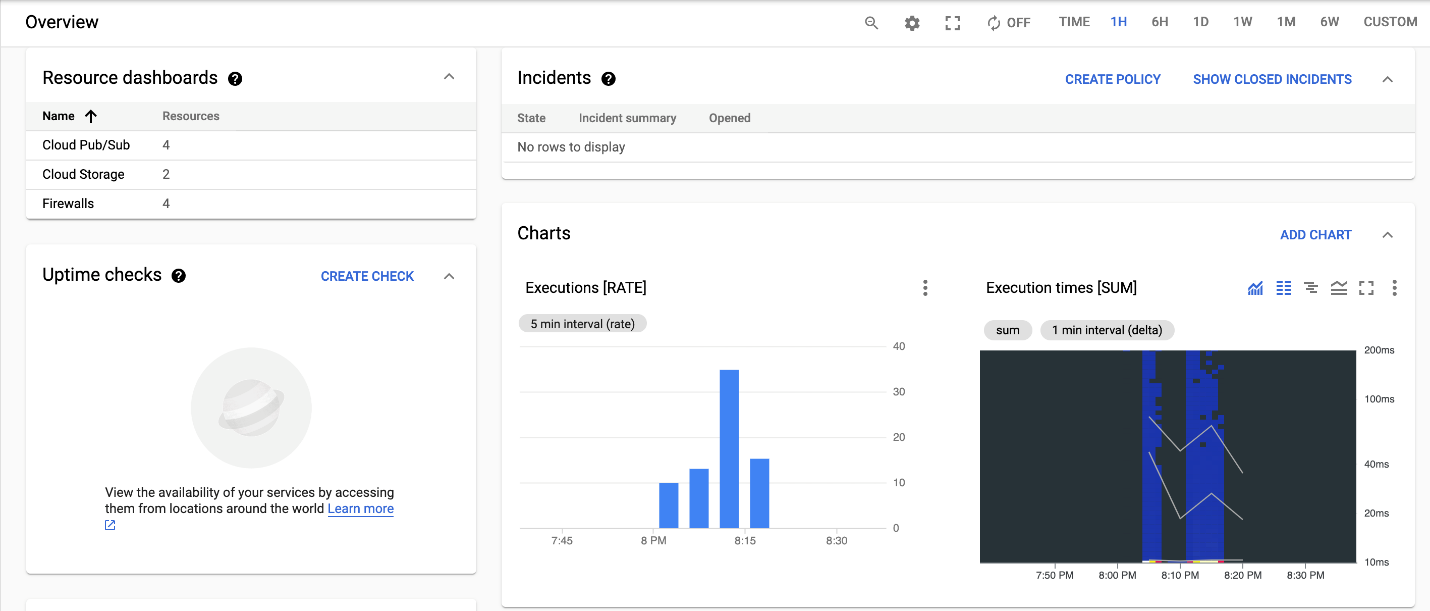
1. In the left menu, click **Overview**.
2. In the Charts section, click **Add Chart**.

**Note:** If the **Add chart** option is greyed out then delete existing charts from the overview page.

1. Click into the **Find resource type and metric** field, choose the resource type and metric you used for the first chart in the last section, and then click **Save**.
2. After you create the first chart, click **Add Chart** to create the next one, using the same metric you use for the second chart in the last section. Click **Save**.

By default, the charts name themselves after the metric you're using, but you can rename them.

For a quick reference, you can see these charts in the Monitoring Overview window in the **Charts** section.



## Test your Understanding

Below are multiple-choice questions to reinforce your understanding of this lab's concepts. Answer them to the best of your abilities.

List out two types of log-based metrics.



Filter metric



Predefined metrics



User-defined logs-based metrics



System logs-based metrics

Submit

Vegeta is a versatile HTTP load testing tool built out of a need to drill HTTP services with a constant request rate.



True



False

Logs-based metrics are Cloud Monitoring metrics that are based on the content of log entries.



True



False

# Reporting Application Metrics into Cloud Monitoring

1 hour 30 minutesFree

## GSP111



## Overview

In this hands-on lab you setup a simple video server application in Go and instrument it to report application metrics, also known as custom metrics, to Cloud Monitoring using the OpenCensus library.

This lab has been adapted from the Go Cloud Monitoring exporter example on [GitHub](https://github.com/census-ecosystem/opencensus-go-exporter-stackdriver/blob/master/examples/stats/main.go). OpenCensus refers to metrics as stats.

## What is OpenCensus?

[OpenCensus](https://opencensus.io/) is an open source framework for metrics collection and distributed tracing. It offers the following benefits to its users:

* Low overhead data collection.
* Standard wire protocols and consistent APIs for handling metrics and trace data.
* Vendor interoperability via the OpenMetrics standard. OpenCensus can ingest into multiple backends in parallel, enabling incremental transitions and side-by-side comparison of backends.
* Correlation with traces and, in the future, log entries.

While multiple methods exist for reporting application metrics to Cloud Monitoring (see Other Methods for Reporting Application Metrics to Cloud Monitoring later in this lab), Google and Cloud Monitoring have adopted OpenCensus officially as the default mechanism for ingestion.

## 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:

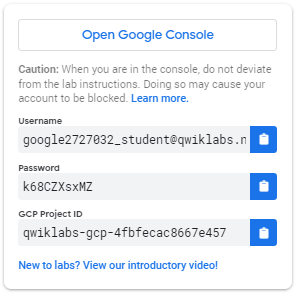
* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

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

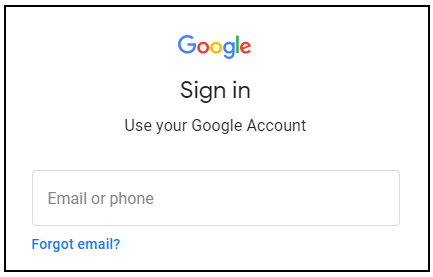
**Note:** If you are using a Pixelbook, open an Incognito window to run this lab.

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

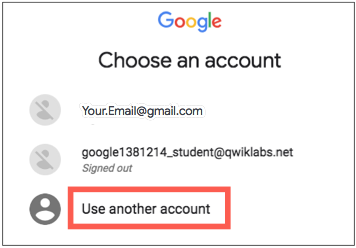
1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



1. 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.

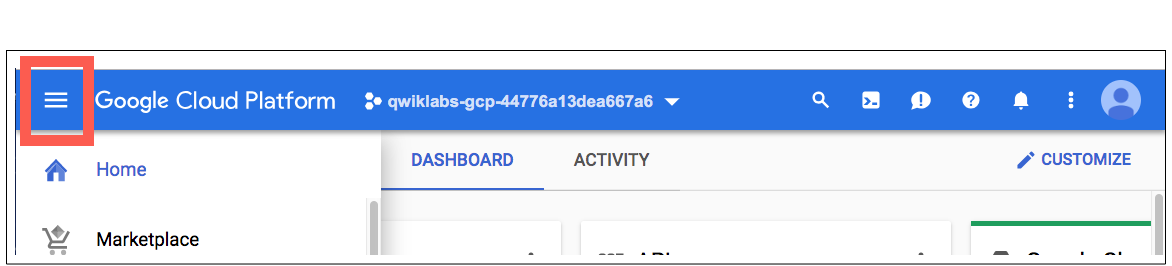
If you see the **Choose an account** page, click **Use Another Account**. 

1. 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).

1. 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. 

## Create a Compute Engine instance

In this lab, you build an application on top of a vanilla Compute Engine (Compute Engine) virtual machine (VM).

In the Cloud Console select **Navigation menu** > **Compute Engine** > **VM instances**, then click **Create**.

Set the following fields:

* **Name:** my-opencensus-demo
* **Region:** us-central1 (Iowa)
* **Zone:** us-central1-a
* **Series**: N1
* **Machine type:** n1-standard-1 (1vCPU, 3.75 GB memory)
* **Identity and API access**: Select **Set access for each API**, then for **Stackdriver Monitoring API** select **Full** from the dropdown menu.
* **Firewall:** Select both **Allow HTTP traffic** and **Allow HTTPS traffic**.

Leave the rest of the fields at their default values and click **Create**.

Click **Check my progress** to verify your performed task.

Create a Compute Engine instance

Check my progress

## Install Go and OpenCensus on your instance

When your new Compute Engine VM instance launches, click **SSH** in line with the instance to open an SSH terminal to your instance.

Use the SSH terminal to install the following:

* Go
* the git package
* the OpenCensus package
* the Cloud Monitoring OpenCensus exporter

Execute the following commands in the SSH terminal:

sudo curl -O https://storage.googleapis.com/golang/go1.10.2.linux-amd64.tar.gzcontent\_copy

sudo tar -xvf go1.10.2.linux-amd64.tar.gzcontent\_copy

sudo mv go /usr/localcontent\_copy

sudo apt-get updatecontent\_copy

sudo apt-get install gitcontent\_copy

You will be asked to confirm you want to continue - type in **Y**.

export PATH=$PATH:/usr/local/go/bin

go get go.opencensus.io

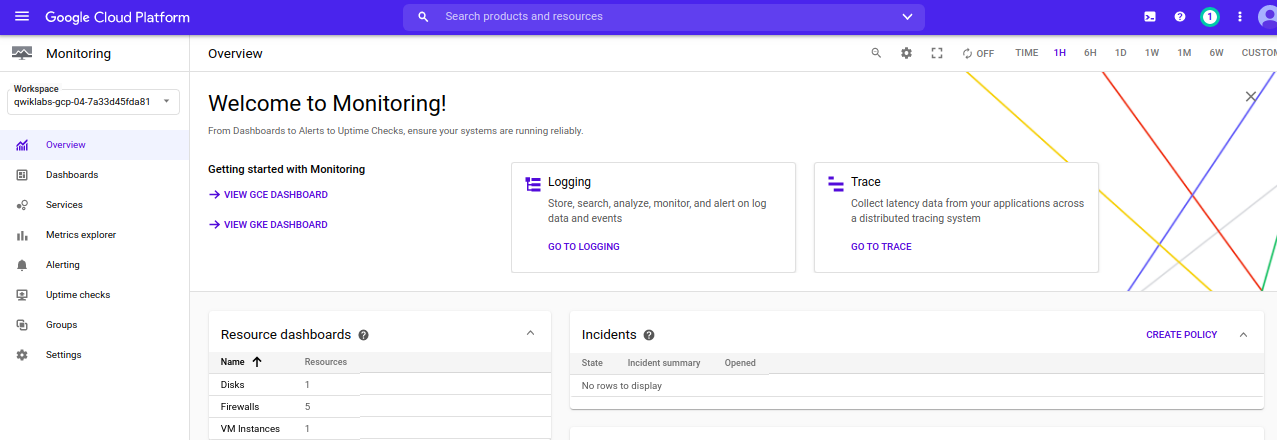
go get contrib.go.opencensus.io/exporter/stackdrivercontent\_copy

### **Create a Monitoring workspace**

Now set up a Monitoring workspace that's tied to your Google Cloud Project. The following steps create a new account that has a free trial of Monitoring.

1. In the Cloud Console, click **Navigation menu** > **Monitoring**.
2. Wait for your workspace to be provisioned.

When the Monitoring dashboard opens, your workspace is ready.



Agents collect data and then send or stream info to Cloud Monitoring in the Cloud Console.

The Cloud Monitoring agent is a collectd-based daemon that gathers system and application metrics from virtual machine instances and sends them to Monitoring. By default, the Monitoring agent collects disk, CPU, network, and process metrics. Configuring the Monitoring agent allows third-party applications to get the full list of agent metrics. See [Cloud Monitoring agent overview](https://cloud.google.com/monitoring/agent) for more information.

In this section, you install the Cloud Logging agent to stream logs from your VM instances to Cloud Logging. Later in this lab, you see what logs are generated when you stop and start your VM.

It is best practice to run the Cloud Logging agent on all your VM instances.

#### Install agents on the VM:

Run the Monitoring agent install script command in the SSH terminal of your VM instance to install the **Cloud Monitoring agent**.

curl -sSO https://dl.google.com/cloudagents/add-monitoring-agent-repo.sh

sudo bash add-monitoring-agent-repo.shcontent\_copy

sudo apt-get updatecontent\_copy

sudo apt-get install stackdriver-agentcontent\_copy

When asked if you want to continue, enter **Y**.

Run the Logging agent install script command in the SSH terminal of your VM instance to install the **Cloud Logging agent**.

curl -sSO https://dl.google.com/cloudagents/add-logging-agent-repo.sh

sudo bash add-logging-agent-repo.shcontent\_copy

sudo apt-get updatecontent\_copy

sudo apt-get install google-fluentdcontent\_copy

Click **Check my progress** to verify your performed task.

Create Monitoring workspace and install agents on the VM

Check my progress

## Create a basic application server in Go

Now you're going to create a fake video server application that is going to check the size of its input queue every second and output that information to the console (you'll fake out the actual queue).

In the SSH window, open your favorite editor (nano, vi, etc.) to create into a file called main.go written in Go. Copy the following into your file:

package main

import (

"fmt"

"math/rand"

"time"

)

func main() {

// Here's our fake video processing application. Every second, it

// checks the length of the input queue (e.g., number of videos

// waiting to be processed) and records that information.

for {

time.Sleep(1 \* time.Second)

queueSize := getQueueSize()

// Record the queue size.

fmt.Println("Queue size: ", queueSize)

}

}

func getQueueSize() (int64) {

// Fake out a queue size here by returning a random number between

// 1 and 100.

return rand.Int63n(100) + 1

}content\_copy

Run the file with:

go run main.gocontent\_copy

You should see output like the following, with a new output line appearing every second:

Queue size: 11

Queue size: 52

...content\_copy

Use **Ctrl** + **c** to stop the output.

## Defining & recording measures using OpenCensus

To put in place the basic infrastructure for propagating metrics (stats) via OpenCensus, you need to define a measure, record it, then set up a view that allows the measure to be collected and aggregated.

You'll do all of the above in a couple of steps added to the main.go file.

Open your main.go file in your text editor.

Start by defining and recording the measure. Add the changes identified by ‘// [[Add this line]]’ or ‘// [[Add this block]] … // [[End: add this block]]’ to your file, then uncomment the added lines and remove the instructions:

package main

import (

"context" // [[Add this line]]

"fmt"

"math/rand"

"time"

"go.opencensus.io/stats" // [[Add this line]]

)

// [[Add this block]]

var videoServiceInputQueueSize = stats.Int64(

"my.videoservice.org/measure/input\_queue\_size",

"Number of videos queued up in the input queue",

stats.UnitDimensionless)

// [[End: add this block]]

func main() {

ctx := context.Background() // [[Add: this line.]]

// Here’s our fake video processing application. Every second, it

// checks the length of the input queue (e.g., number of videos

// waiting to be processed) and records that information.

for {

time.Sleep(1 \* time.Second)

queueSize := getQueueSize()

// Record the queue size.

// [[Add: next line.]]

stats.Record(ctx, videoServiceInputQueueSize.M(queueSize)) // [[Add]]

fmt.Println("Queue size: ", queueSize)

}

}

func getQueueSize() (int64) {

// Fake out a queue size here by returning a random number between

// 1 and 100.

return rand.Int63n(100) + 1

}

content\_copy

The go.opencensus.io/stats package contains all of the support you need to define and record measures.

In this example, videoServiceInputQueueSize is the measure. It's defined it as a 64-bit integer type. Each measure requires a name (the first parameter), a description, and a measurement unit.

A measure that has been defined also needs to be recorded. The stats.Record(...) statement sets the measure, videoServiceInputQueueSize, to the size queried, queueSize.

## Setting up metrics collection & aggregation

Now that our metric is defined and being recorded, the next step is to enable collection and aggregation of the metric. We do this in OpenCensus by setting up a view.

Add the changes identified by ‘// [[Add this line]]’ or ‘// [[Add this block]] … // [[End: add this block]]’ to your main.go file, then uncomment the added lines and remove the instructions:

package main

import (

"context"

"fmt"

"log" // [[Add this line]]

"math/rand"

"time"

"go.opencensus.io/stats"

"go.opencensus.io/stats/view" // [[Add this line]]

)

var videoServiceInputQueueSize = stats.Int64(

"my.videoservice.org/measure/input\_queue\_size",

"Number of videos queued up in the input queue",

stats.UnitDimensionless)

func main() {

ctx := context.Background()

// [[Add this block]]

// Setup a view so that we can export our metric.

if err := view.Register(&view.View{ // [[Add]]

Name: "my.videoservice.org/measure/input\_queue\_size",

Description: "Number of videos queued up in the input queue",

Measure: videoServiceInputQueueSize,

Aggregation: view.LastValue(),

}); err != nil {

log.Fatalf("Cannot setup view: %v", err)

}

// Set the reporting period to be once per second.

view.SetReportingPeriod(1 \* time.Second)

// [[End: Add this block]]

// Here’s our fake video processing application. Every second, it

// checks the length of the input queue (e.g., number of videos

// waiting to be processed) and records that information.

for {

time.Sleep(1 \* time.Second)

queueSize := getQueueSize()

// Record the queue size.

stats.Record(ctx, videoServiceInputQueueSize.M(queueSize))

fmt.Println("Queue size: ", queueSize)

}

}

func getQueueSize() (int64) {

// Fake out a queue size here by returning a random number between

// 1 and 100.

return rand.Int63n(100) + 1

}

content\_copy

Above, view.View{...} defines the name of the exported metric (using the same name as the name for the measure itself), associates it with the videoServiceInputQueueSize measure, and specifies, via the Aggregation field, to export the last value of that metric. We also explicitly set the reporting period. For this example, the metric is specified to be reported every second; the best practices for a production service is to report metrics no more frequently than once per minute.

See <https://godoc.org/go.opencensus.io/stats/view> for full details about setting up views for metrics collection and aggregation.

## Reporting default metrics to Cloud Monitoring

The final step is to export the application metric to Cloud Monitoring. This is done by configuring the OpenCensus Cloud Monitoring exporter.

Add the changes identified by ‘// [[Add this line]]’ or ‘// [[Add this block]] … // [[End: add this block]]’ to your file, then uncomment the added lines and remove the instructions:

package main

import (

"context"

"fmt"

"log"

"math/rand"

"os" // [[Add]]

"time"

"contrib.go.opencensus.io/exporter/stackdriver" // [[Add]]

"go.opencensus.io/stats"

"go.opencensus.io/stats/view"

monitoredrespb "google.golang.org/genproto/googleapis/api/monitoredres" // [[Add]]

)

var videoServiceInputQueueSize = stats.Int64(

"my.videoservice.org/measure/input\_queue\_size",

"Number of videos queued up in the input queue",

stats.UnitDimensionless)

func main() {

// [[Add block]]

// Setup metrics exporting to Stackdriver.

exporter, err := stackdriver.NewExporter(stackdriver.Options{

ProjectID: os.Getenv("MY\_PROJECT\_ID"),

Resource: &monitoredrespb.MonitoredResource {

Type: "gce\_instance",

Labels: map[string]string {

"instance\_id": os.Getenv("MY\_GCE\_INSTANCE\_ID"),

"zone": os.Getenv("MY\_GCE\_INSTANCE\_ZONE"),

},

},

})

if err != nil {

log.Fatalf("Cannot setup Stackdriver exporter: %v", err)

}

view.RegisterExporter(exporter)

// [[End: add block]]

ctx := context.Background()

// Setup a view so that we can export our metric.

if err := view.Register(&view.View{

Name: "my.videoservice.org/measure/input\_queue\_size",

Description: "Number of videos queued up in the input queue",

Measure: videoServiceInputQueueSize,

Aggregation: view.LastValue(),

}); err != nil {

log.Fatalf("Cannot setup view: %v", err)

}

// Set the reporting period to be once per second.

view.SetReportingPeriod(1 \* time.Second)

// Here’s our fake video processing application. Every second, it

// checks the length of the input queue (e.g., number of videos

// waiting to be processed) and records that information.

for {

time.Sleep(1 \* time.Second)

queueSize := getQueueSize()

// Record the queue size.

stats.Record(ctx, videoServiceInputQueueSize.M(queueSize))

fmt.Println("Queue size: ", queueSize)

}

}

func getQueueSize() (int64) {

// Fake out a queue size here by returning a random number between

// 1 and 100.

return rand.Int63n(100) + 1

}content\_copy

The OpenCensus metrics exporter for Cloud Monitoring is a contributed package. For the exporter to function correctly, the ProjectID is provided to it. This lab is using the value set by the MY\_PROJECT\_ID environment variable.

In addition, Cloud Monitoring requires that metrics be reported against a monitored resource. A monitored resource identifies the source of the metric data. For this lab the metrics are reported against the underlying Compute Engine VM instance. This is done by specifying the Resource parameter in stackdriver.Options:



The type gce\_instance specifies the type of monitored resource to associate with the metric. Compute Engine VM instances have two labels, an "instance\_id" and a "zone". The instance\_id is the numeric id of the Compute Engine VM instance and the zone is the zone in which the instance is running. You can find this information on Compute Engine's detail page for your instance. The zone information is readily available. To determine the instance ID for your Compute Engine VM instance, click Equivalent REST at the bottom of the page; the ID is a 19-digit number. For convenience, these values are set in environment variables rather than hard-coding them in the Go application.

Now you need to somehow route the metrics you care about to the exporter. This is achieved by registering the exporter with the view that is collecting and aggregating the metrics - i.e., via view.Register(...).

See <https://godoc.org/contrib.go.opencensus.io/exporter/stackdriver> for more details about the OpenCensus exporter for Cloud Monitoring.

Save and close the main.go file.

## View your application metrics in Cloud Monitoring

You're just about ready to view your application metrics in Cloud Monitoring.

In the SSH window, set the necessary environment variables:

* Project ID - found on the Qwiklabs page where you started this lab.

export MY\_PROJECT\_ID=<your-project-id>

export MY\_GCE\_INSTANCE\_ID=my-opencensus-demo

export MY\_GCE\_INSTANCE\_ZONE=us-central1-acontent\_copy

Now run your application:

go run main.gocontent\_copy

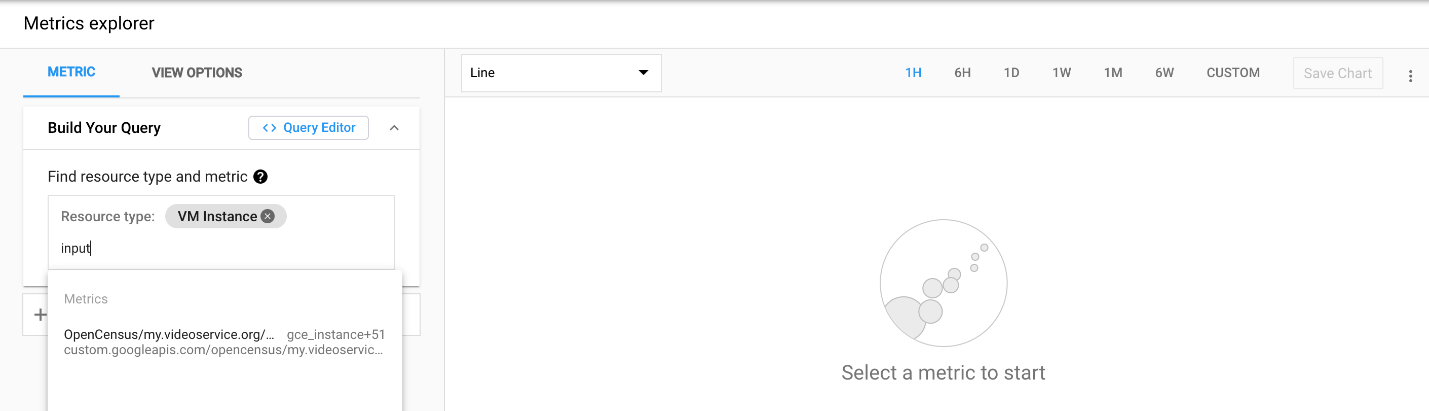
**Note:** If you see errors, you can ignore them.

You can leave the application running.

Now go back to the Console, you should be on the Cloud Monitoring window.

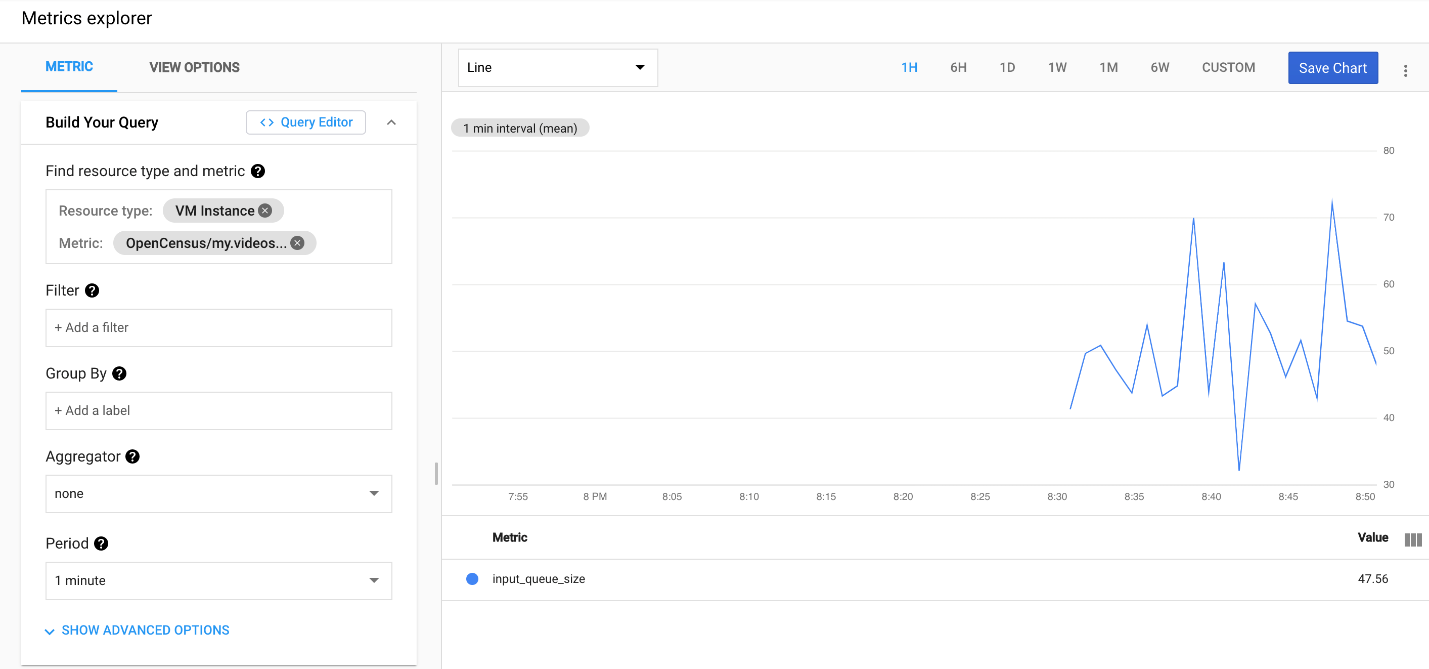
In the left menu, click **Metrics Explorer**.

In the **Find resource type and metric** text box, select **VM Instance** under **Resource types**, then type ‘input' in the **Select a metric** row, and press **Enter**.



**Note:** If you don't see the "OpenCensus/my.videoservice.org" choice, give Cloud Monitoring a few minutes to collect data, refresh your browser, and try again.

You should soon see data in the graph, maybe something like this:



Click **Check my progress** to verify your performed task.

View your application metrics in Cloud Monitoring

Check my progress

## Other methods for reporting application metrics to Cloud Monitoring

Users who are instrumenting their code for the first time should use the [OpenCensus library](https://opencensus.io/" \t "_blank), as shown in this lab. Users with pre-existing instrumentation should consider one of our existing adapters:

* Prometheus: Use the [Prometheus integration](https://cloud.google.com/monitoring/kubernetes-engine/prometheus) (beta).
* [StatsD](https://github.com/etsy/statsd/wiki) and anything else supported by [collectd](https://collectd.org/" \t "_blank): Use the Cloud Monitoring agent. See the [Custom Metrics from the Agent](https://cloud.google.com/monitoring/agent/custom-metrics-agent) documentation to get this setup. The documentation on [Custom Metrics](https://cloud.google.com/monitoring/custom-metrics/) might also be valuable.

While Cloud Monitoring offers a public API, the [CreateTimeSeries API](https://cloud.google.com/monitoring/api/ref_v3/rest/v3/projects.timeSeries/create" \t "_blank), for metrics ingestion, is mainly for ingestion use cases not covered by OpenCensus, such as bulk ingestion. Use of any other approach not outlined above is currently unsupported.

# Creating and Alerting on Logs-based Metrics

1 hour 30 minutesFree

## GSP091



**Logs-based metrics** are [Cloud Monitoring](https://cloud.google.com/monitoring/docs/) metrics that are based on the content of log entries. It can help you identify trends, extract numeric values out of the logs, and set up an alert when a certain log entry occurs by creating a metric for that event. You can use both system and user-defined logs-based metrics in Cloud Monitoring to create charts and alerting policies. Logs-based metrics are time series that are generated from data in logs. In this lab you'll do all of the above!

## 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:

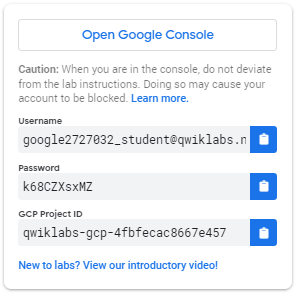
* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

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

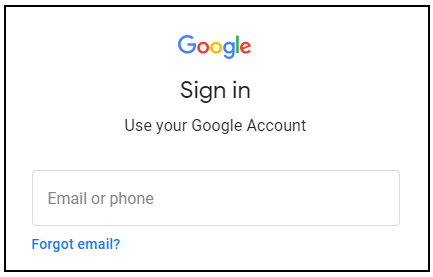
**Note:** If you are using a Pixelbook, open an Incognito window to run this lab.

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

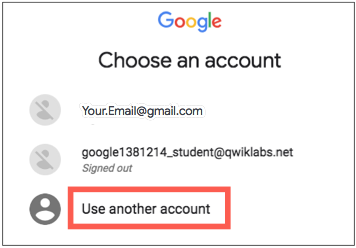
1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



1. 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.

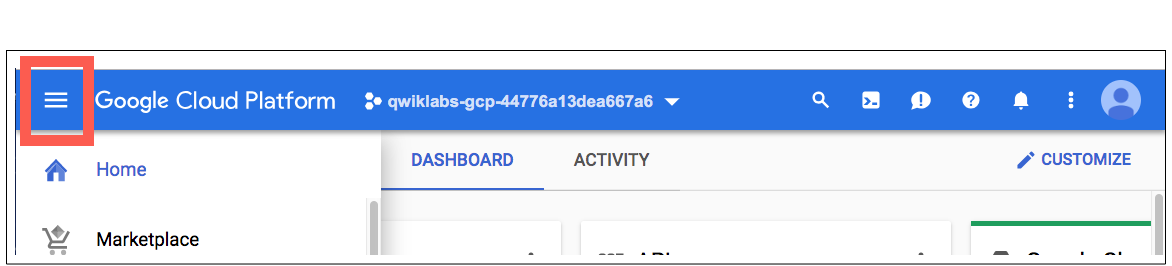
If you see the **Choose an account** page, click **Use Another Account**. 

1. 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).

1. 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.

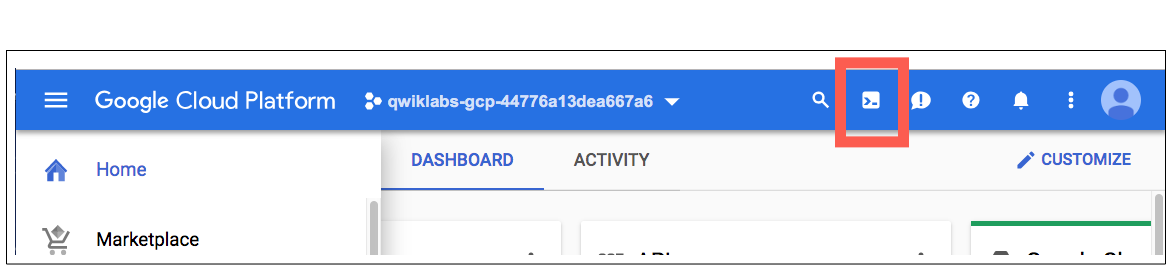
After a few moments, the Cloud Console opens in this tab.

**Note:** You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left. 

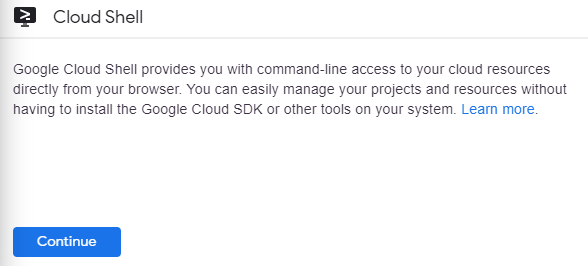
### **Activate Cloud Shell**

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.

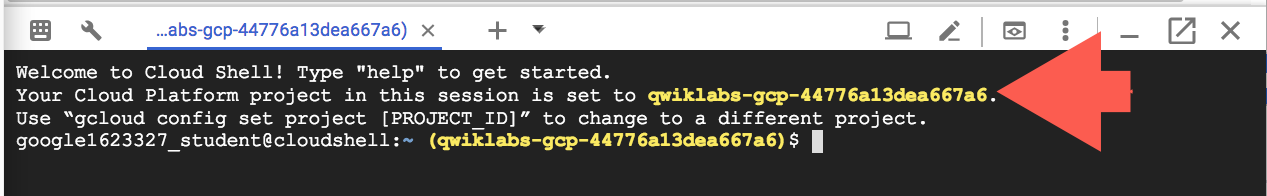
In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



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:



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

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

For full documentation of gcloud see the [gcloud command-line tool overview](https://cloud.google.com/sdk/gcloud" \t "_blank).

Resources are being set up for this lab, including a virtual machine. Make sure you see the green Lab Running light on the page where you started the lab before continuing.

## Create resources for the lab

The first resource you need for this lab is an app that generates logs. In this section, you deploy a Guestbook app to App Engine, then create some uptime checks for Cloud Monitoring to log.

Use the following commands to clone the app example to your Google Cloud project:

git clone https://github.com/GoogleCloudPlatform/appengine-guestbook-python

content\_copy

cd appengine-guestbook-python/

content\_copy

gcloud app create

content\_copy

Type in a number for the region where you want the App Engine app created.

Next, run:

gcloud app deploy --version 1

content\_copy

Enter **Y** to continue.

gcloud datastore indexes create index.yaml

content\_copy

Enter **Y** to continue.

## Monitor indexes

Monitor the indexes by going to **Navigation menu** > **Datastore** > **Indexes** in the Console. Give it a couple of minutes to set up. Use the **Refresh** button at the top of the screen. When you see a green check, the index is set up and the status should be "Serving".

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

Deploy a Guestbook app to App Engine and Indexes for datastore

Check my progress

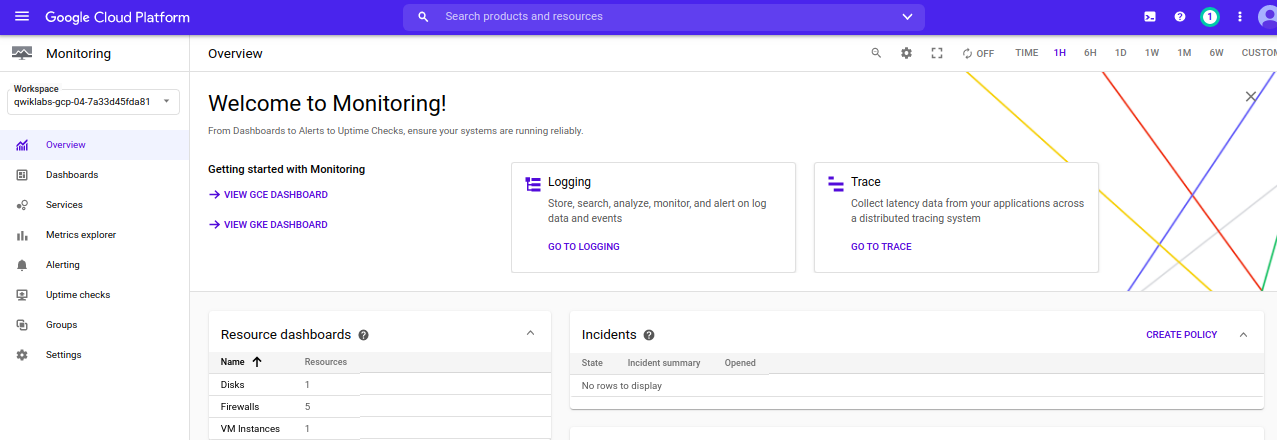
## Prepare to install Cloud Monitoring

### **Create a Monitoring workspace**

Now set up a Monitoring workspace that's tied to your Google Cloud Project. The following steps create a new account that has a free trial of Monitoring.

1. In the Cloud Console, click **Navigation menu** > **Monitoring**.
2. Wait for your workspace to be provisioned.

When the Monitoring dashboard opens, your workspace is ready.



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

Create a Monitoring workspace

Check my progress

### **Install the Monitoring and Logging agents**

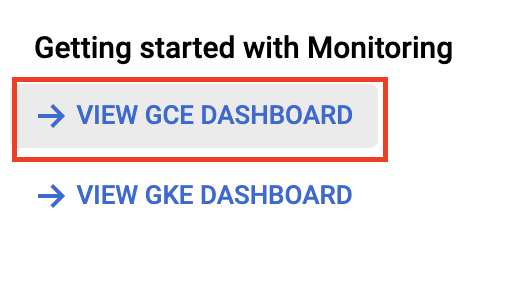
Agents collect data and then send or stream info to Cloud Monitoring in the Cloud Console.

The Cloud Monitoring agent is a collectd-based daemon that gathers system and application metrics from virtual machine instances and sends them to Monitoring. By default, the Monitoring agent collects disk, CPU, network, and process metrics. Configuring the Monitoring agent allows third-party applications to get the full list of agent metrics. [Learn more](https://cloud.google.com/monitoring/agent).

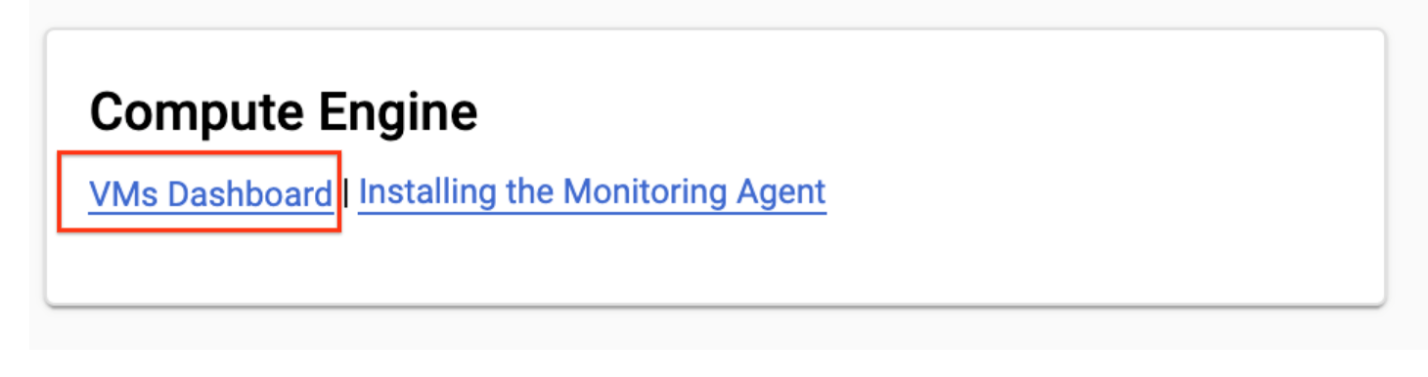
The Cloud Logging agent streams logs from your VM instances and from selected third-party software packages to Cloud Logging. It is a best practice to run the Cloud Logging agent on all your VM instances. [Learn more](https://cloud.google.com/logging/docs/agent).

To install the agents on the VM:

1. In the Monitoring Overview window, click **VIEW GCE DASHBOARD**.

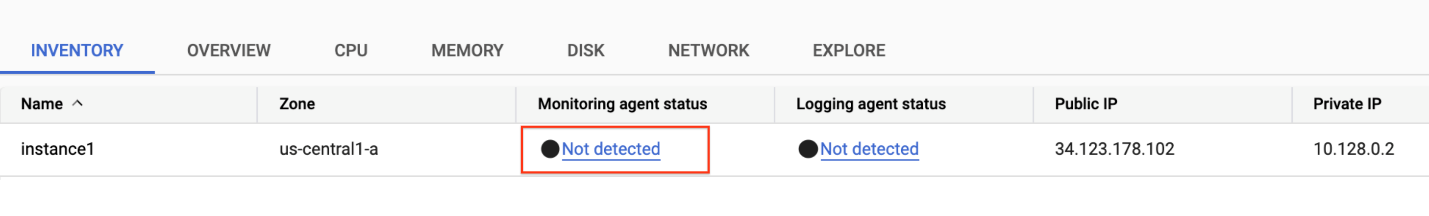


1. Click on the **VMs Dashboard** link under **Compute Engine**.



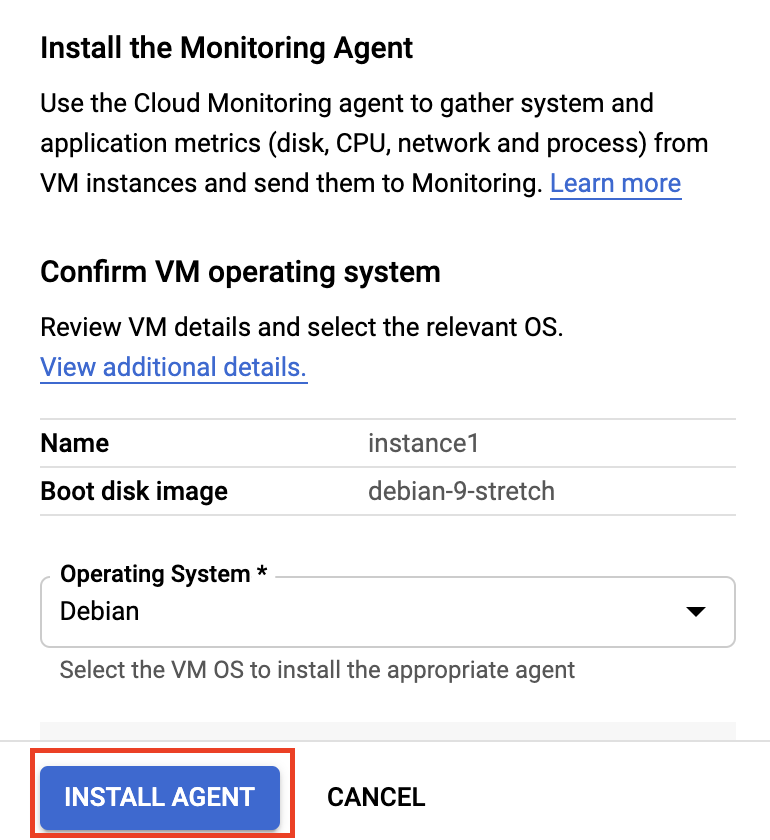
You will see your VM, **instance1**, listed.

1. Under the **Monitoring Agent Status** for **instance1**, click **Not Detected**.



This will bring up an **Agent Details** side window which can be used to help install the agent.

1. In the **Agent Details** window, ensure **Debian** is selected as the operating system, and click **Install Agent**.

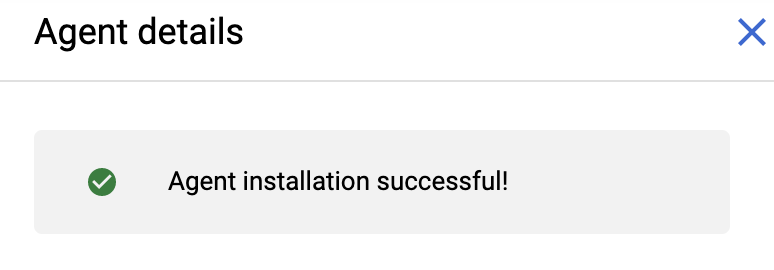


This will auto generate a command in your Cloud Shell that SSHes into your VM and installs the agent.

Copy the generated command from the Agent details window and run it in your Cloud Shell. Type **y** when asked if you want to continue and then press **enter** twice to not use a passphrase.

While the installation of the agent runs, type and enter **y** whenever prompted to continue.

A few minutes after the installation finishes you should see a green check mark in your **Agent Details** window:



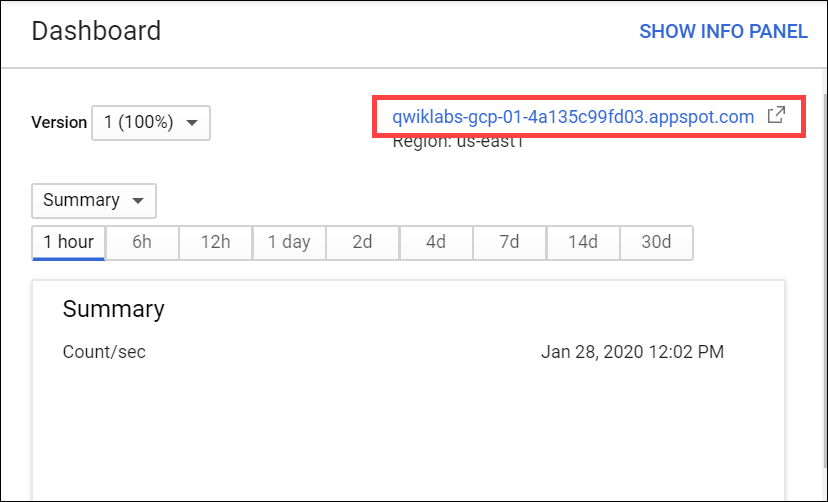
## Additional resources for the lab

Since this lab uses logs, it helps if you had some interesting logs. In this lab, you generate these logs with uptime checks and a VM creation alert.

### **Create an uptime check**

Create a few uptime checks for the app that's running on App Engine. The uptime checks simulate load against the app. App Engine automatically captures logs from all requests and produces one log per minute, which provides data to analyze later.

1. Go back to the Cloud Console and click **Navigation menu** > **App Engine**. Click on the link in the upper right corner, which ends in ".appspot.com".



Your App Engine Guestbook app opens in a new browser tab. Record the URL to use when you configure the uptime check.

1. Go back to the Cloud Monitoring window (**Navigation menu** > **Monitoring**).
2. In the left menu, click **Uptime checks**, and then **Create Uptime Check**.
3. In the **New uptime check** dialog, set the following fields:

|  |  |
| --- | --- |
| **Field** | **Value** |
| **Title** | pizza check |

Click **Next**.

In the **Target** section, set the following fields:

|  |  |
| --- | --- |
| **Field** | **Value** |
| **Protocol** | HTTP |
| **Resource Type** | URL |
| **Hostname** | Paste in the URL for your app (that you previously recorded). Remove the https:// and the last / from the URL. |
| **Path:** | /?food=pizza |
| **Check Frequency** | 1 min |

Click **Next**.

You can use the default settings for **Response Validation** and **Alert & Notification**, so click **Next** one more time.

1. Now, click **Test**.

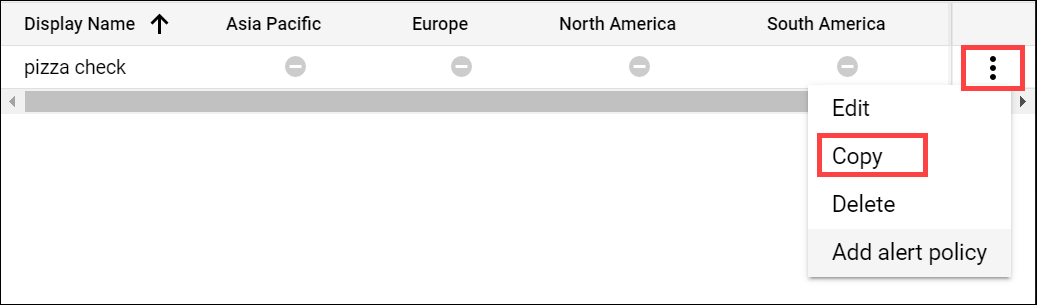
You should see a successful response.

1. Click **Create**.

The Uptime checks window opens and lists the pizza check in the Uptime checks table.

### **Create 2 more uptime checks**

1. Still in the Uptime checks window, click the vertical ellipsis in line with pizza check uptime check, and then click **Copy**.



* Change the Title to "burger check".
* Click **Next** and change the Path to "/?food=burger".
* Click **Next** twice and then click **Create**.

1. Copy an uptime check one more time.

* Change the Title to "cake check".
* Click **Next** and change the Path to "/?food=cake"
* Click **Next** twice and then click **Create**.

All three uptime checks are listed in the Uptime checks table.

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

Create an uptime check

Check my progress

## System defined and user defined logs-based metrics

Logs-based metrics are divided into System Defined and User Defined.

### **System defined logs-based metrics**

System defined logs-based metrics are ready to use right out of the box.These [system logs-based metrics](https://cloud.google.com/monitoring/api/metrics_gcp#gcp-logging) include:

#### Metrics around logs ingested

* Byte\_count: Number of bytes in all log entries ingested. This is broken down by monitored resource type, log stream name, and severity level.

#### Metrics around logs excluded

* Excluded\_byte\_count: Number of bytes in log entries that were excluded. This is broken down by the monitored resource type.
* Excluded\_log\_entry\_count: Number of log entries that were excluded. This is broken down by the monitored resource type.

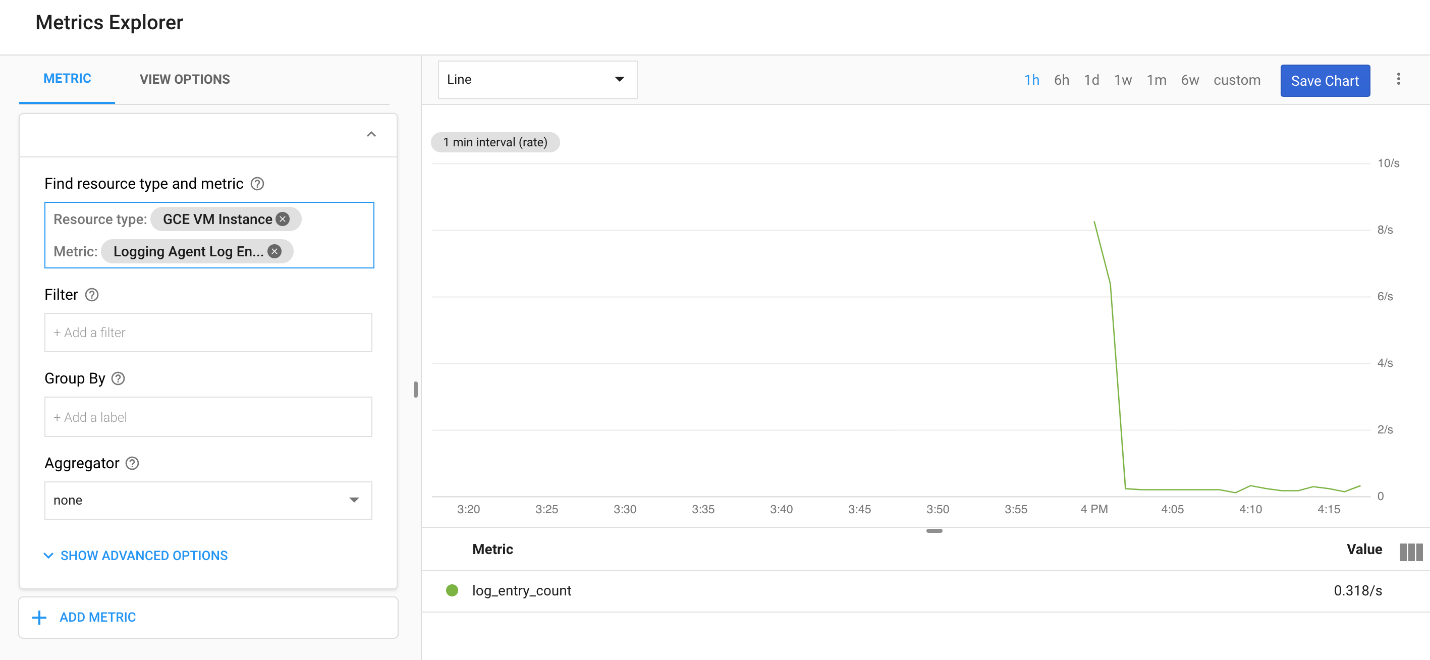
#### Metrics around logs based metrics

* Dropped\_log\_entry\_count: Despite the name, this does not show log entries dropped by Cloud logging but rather the number of log entries that did not contribute to logs based metrics because they arrived too late.
* Log\_entry\_count: Number of log entries that contributed to logs based metrics so that dropped\_log\_entry\_count + log\_entry\_count is the total number of log entries ingested by Cloud Logging.
* Metric\_throttled: Indicates if points are being dropped for logs-based metrics due to exceeding time series limits.
* Time\_series\_count: Estimate of the active time series count for logs-based metrics.

Most system logs-based metrics are counter metrics. **Counter metrics** count the number of log entries that match an advanced logs filter.

Now you'll look more closely at a system generated logs-based metric: Log\_entry\_count.

1. In the left menu, click **Metrics Explorer**:
2. To find a metric, start typing "log entries" and choose **Logging Agent Log Entry Count**.
3. Select **Compute Engine VM Instance** as your Resource.



You're looking at the graphical representation of log entries for the machine that was started as one of your resources when you began this lab.

## User defined logs-based metrics

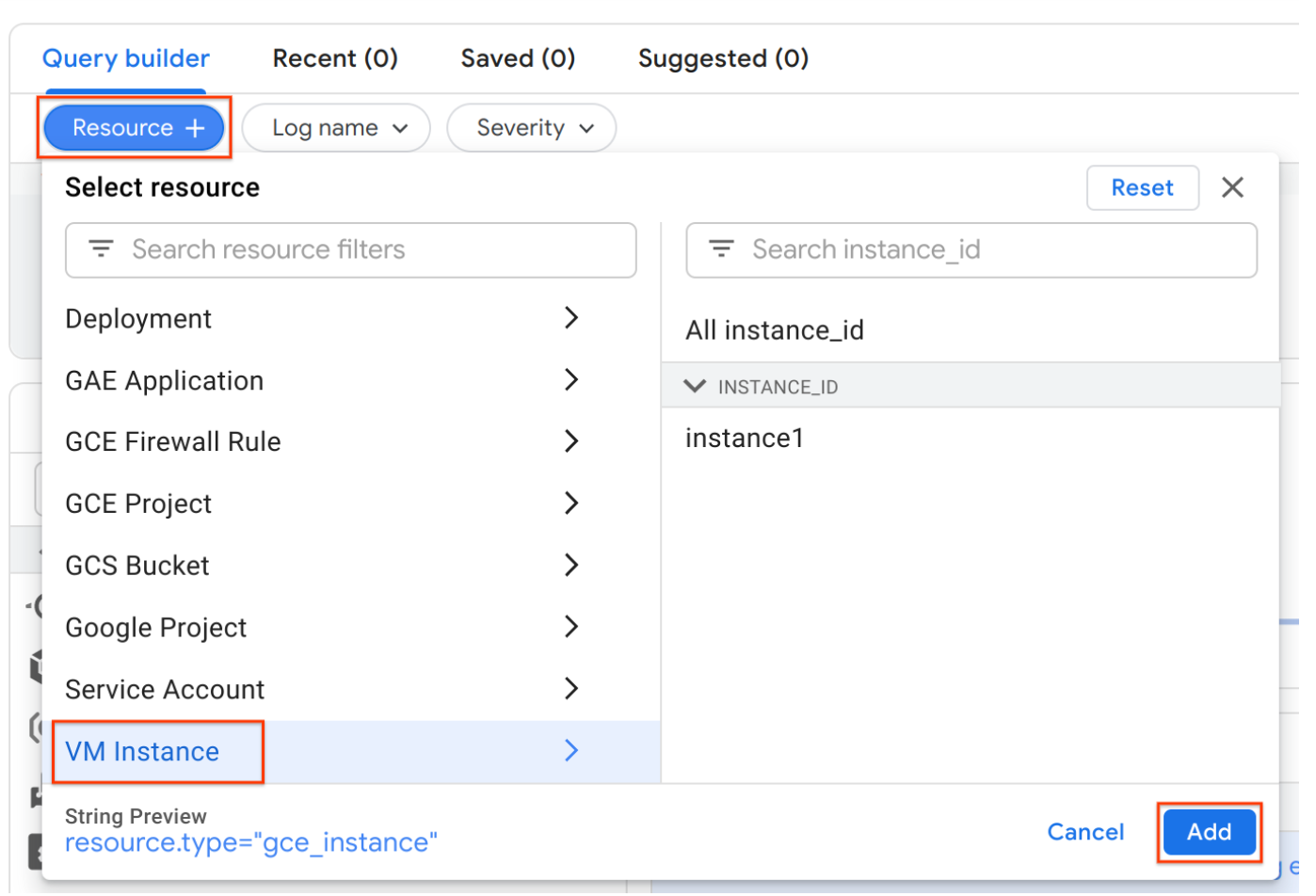
Create your own logs-based metrics using data from existing logs. These are called **user defined logs-based metrics**. In this section, you create a metric using a log entry.

1. Select **Navigation menu** > **Logging** > **Logs Explorer**.

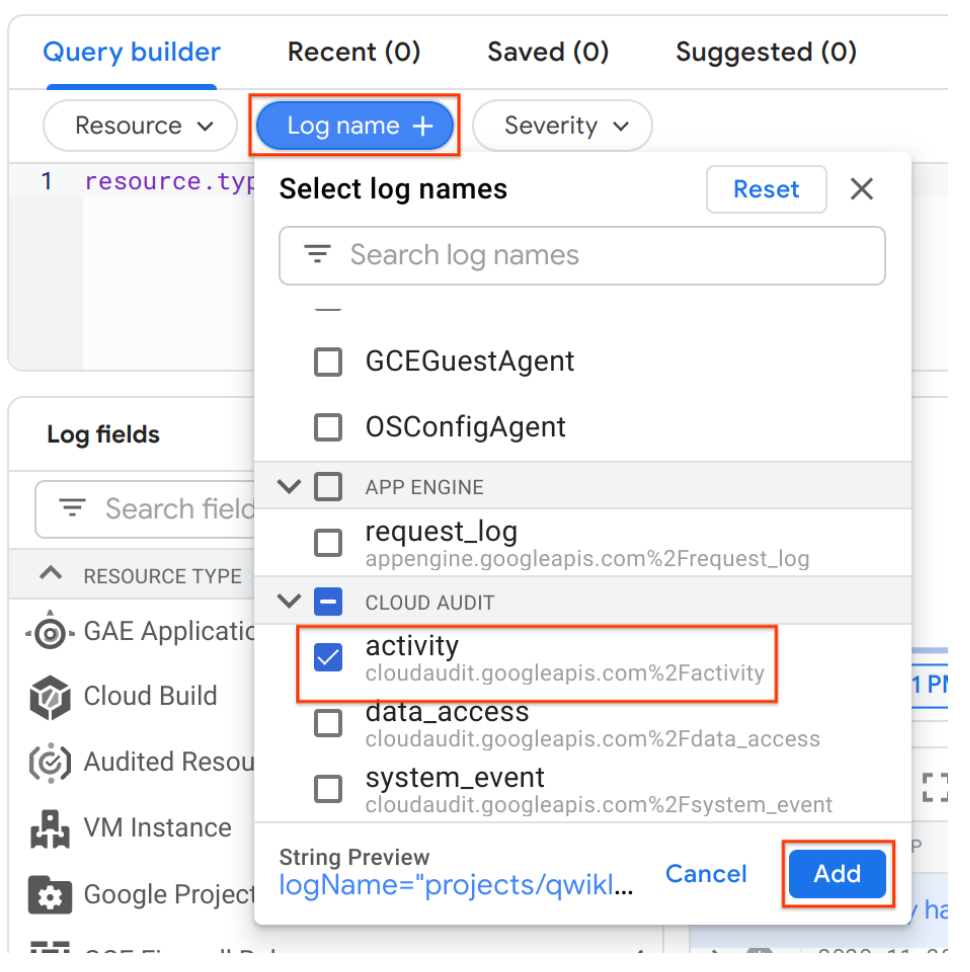
The Cloud Logging window opens.

1. Filter for activity logs from VM instance you previously created.

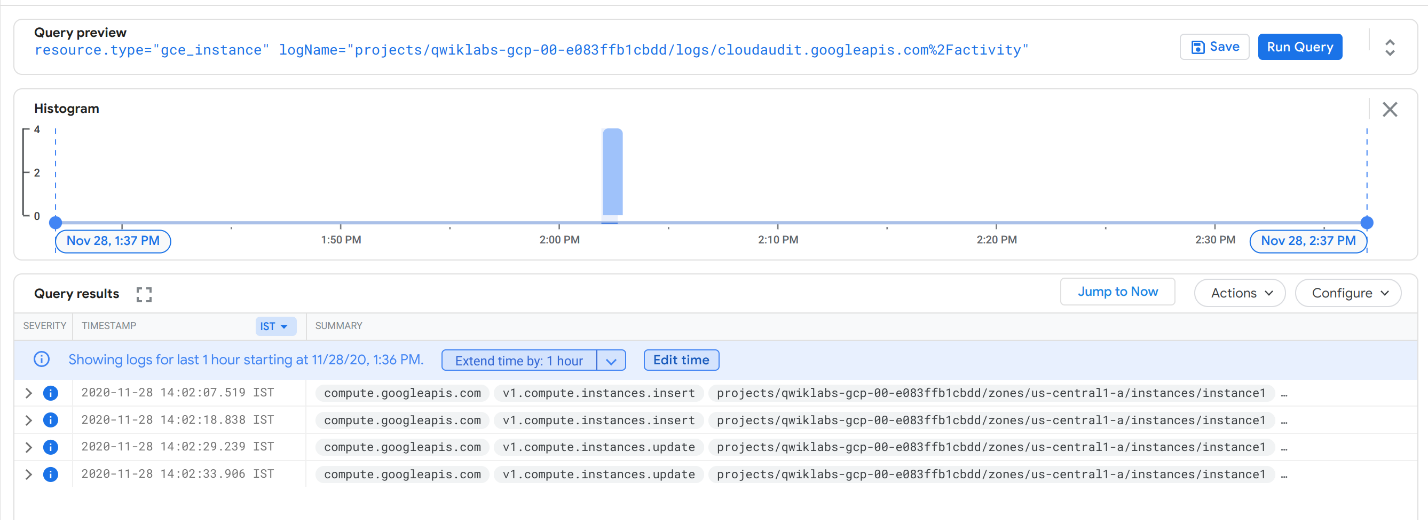
* In the **Resource** dropdown, select **VM Instance** and click **Add**.



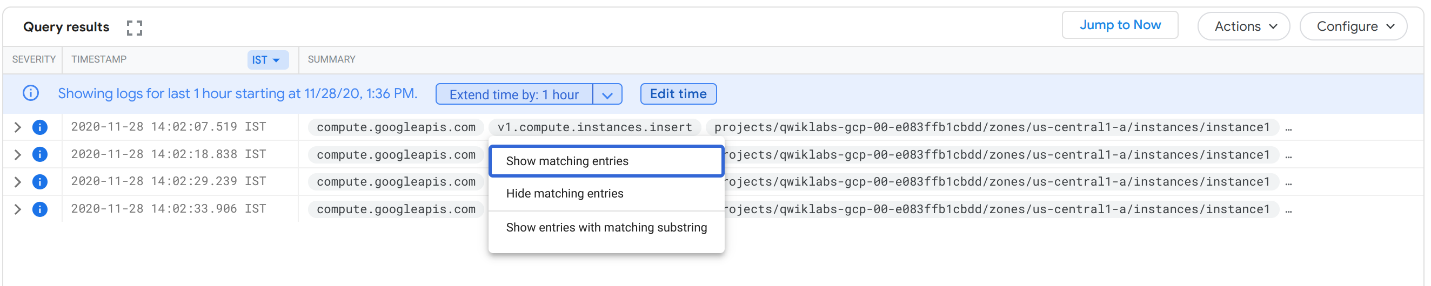
* In the **Log name** dropdown, select **activity**, then click **Add**.



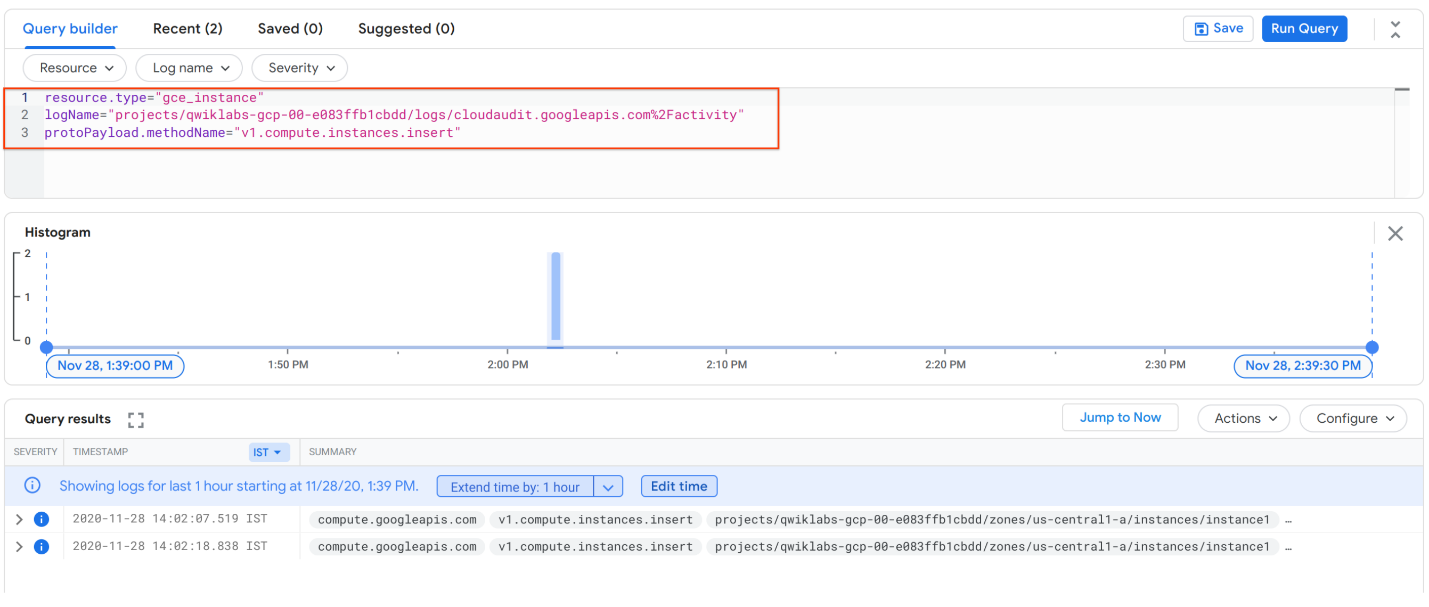
* Click **Run Query**.



1. In one of the entries, click on the "insert" tag, then select **Show matching entries**.



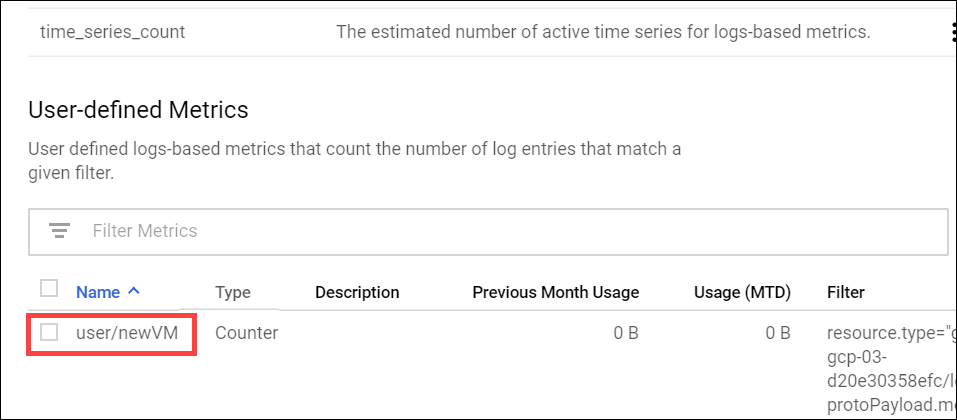
The **Query builder** is now displays the criteria for this metric.



The protoPayload.methodName in the 3rd row lets you filter on many methods. You can read more about that [here](https://cloud.google.com/kubernetes-engine/docs/how-to/audit-logging).

1. On top right of the Query results section, click **Actions** drop down and select **Create Metric**.
2. In the Metric Editor name your metric "newVM" then click **Create Metric**.

Your user defined logs-based metric is now listed in the Logs-based metrics page.



### **Create alerting policy for the VM creation metric**

Create an alert to let you know when a new VM gets added to your project.

1. Return to the Cloud Monitoring window (**Navigation menu** > **Monitoring**).
2. In the left menu, click **Alerting**, and then click **Create Policy**.
3. Click **Add Condition** and set the following:

* For Target, click inside the dropdown menu and choose **VM Instance**.
* For the metric, start typing "logging/" and select **logging/user/newVM**.
* For Configuration, set Condition to **is above** the Threshold **0** For **1 minute**.
* Click **Add**.

If "logging/user/newVM" isn't coming up as a metric when searching, try refreshing your browser. If it still doesn't come up after refreshing, double check that the last step was completed correctly.

Click **Next**.

1. Click the **Notification Channels** dropdown and click **Manage Notification Channels**.

In the tab that comes up, find **Email** and click **Add New**. Add your personal email and either your name or an alias for the **Display Name**.

Now, back in the **Create Alerting Policy** tab, select the **Notification Channels** dropdown and click the **refresh** button.

You should see the display name of your email with a check box next to it. Select the checkbox next to your email's display name.

Click **Next**.

1. Enter "New Virtual Machine" as the **Alert Name**.
2. Click **Save**.

### **Create a new instance**

To trigger the alert you just created, go create a new virtual machine instance.

1. In the Cloud Console dashboard, then go to **Navigation menu** > **Compute Engine** > **VM instances**, then click **Create Instance** at the top of the screen.
2. Name your instance **instance2**, then check the firewall boxes to **allow HTTP and HTTPs traffic**. Leave all the other fields with their default values.
3. Click **Create**.

Wait a couple of minutes for your instance to be launched. You should see an Incident in Cloud Monitoring console in 3-5 minutes.

Continue with the lab, you can check these results at the end.

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

Create user defined logs-based metrics

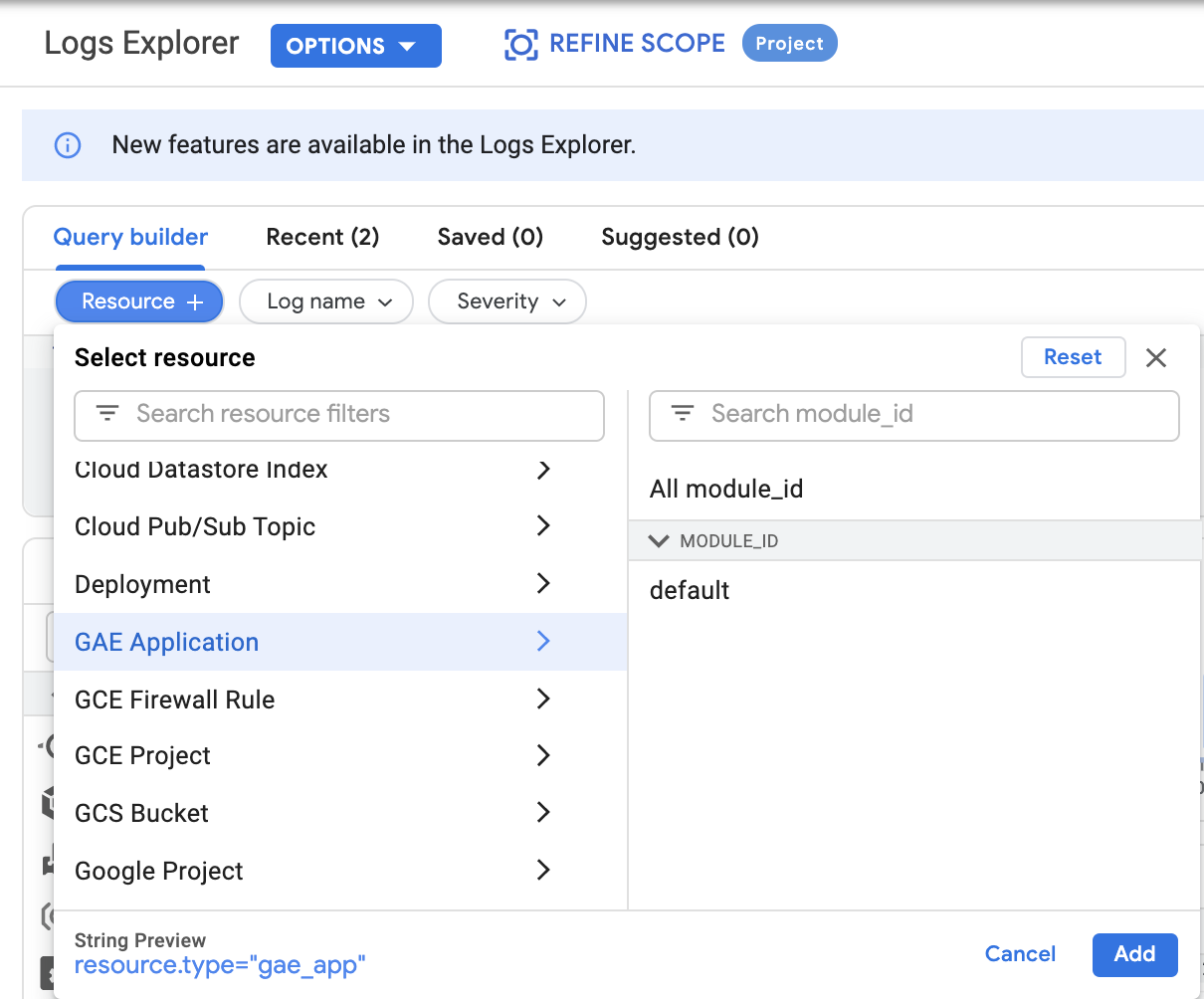
Check my progress

## Labels and user defined metrics

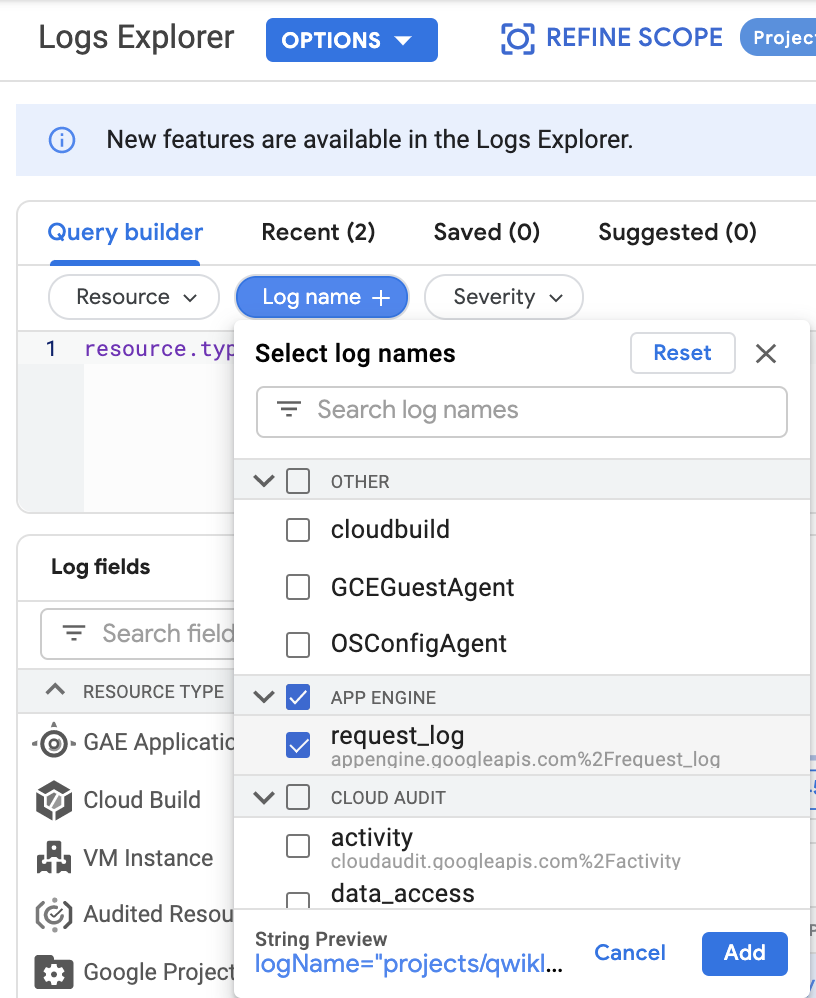
User defined labels can be created when you create a metric. An extractor expression is required for each configured label to tell Cloud Logging how to extract values from logs and place them as the labels' value. You cannot add labels to system logs-based metrics.

Create a user-defined metric with a label.

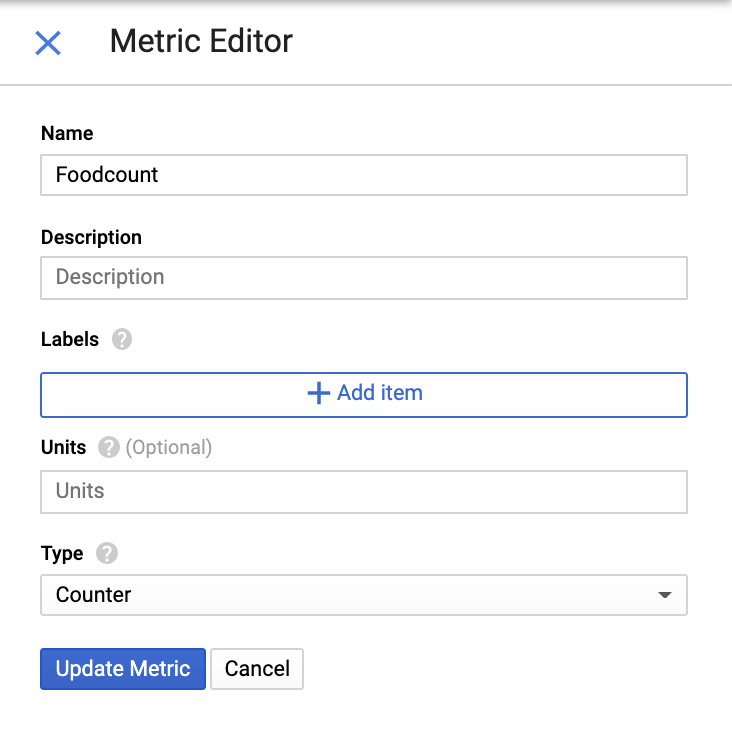
1. Select **Navigation menu** > **Logging** > **Logs Explorer**.
2. In **Resource** drop-down, select **GAE Application** and click **Add**.



1. For **Log name**, select **request\_log** and click **Add**.



1. Select **Actions > Create Metric**.
2. Name your metric "Foodcount" and add a Description.
3. Click on **Add item** to create a Label.



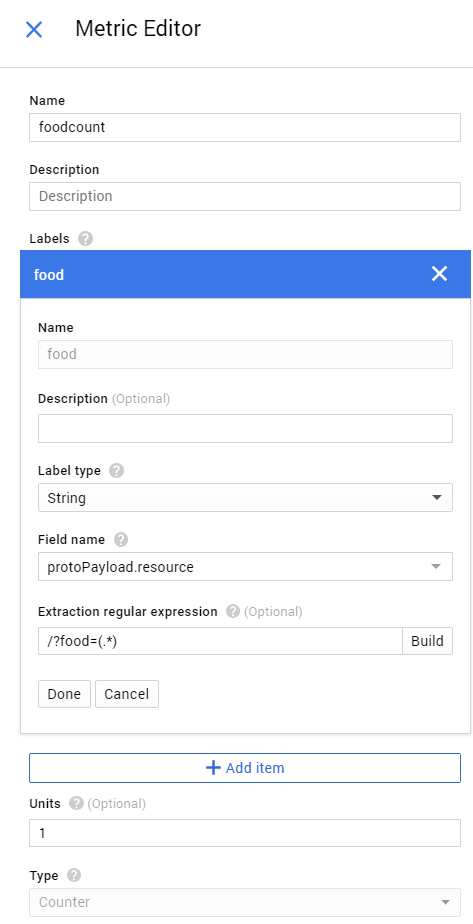
## Labels

[Labels](https://cloud.google.com/logging/docs/logs-based-metrics/labels#create-label) allow logs-based metrics to contain multiple time series — one for each label value. All logs-based metrics come with some default labels.

For this lab you'll create a metric for when an uptime check log occurs that has "food" in the log.

1. Go back to what you're doing, which is creating a Metric. In the Metric Editor, you've named you metric Foodcount, and clicked **Add Item** to create a label.
2. Set the following:

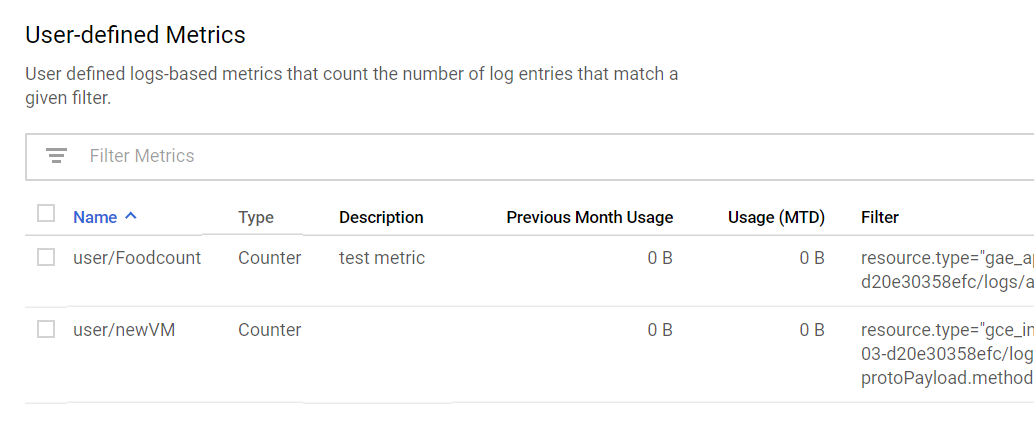
* Name: food
* Label Type: String
* Field Name: search for "resource" and choose **protoPayload.resource**.
* Click the **Build** button, then enter this regular expression: /?food=(.\*)
* Click **Done**



**Caution:** Be sure to specify the extractor for your label values carefully. A mistake can result in having a large number of active time series. Exceeding time series limits can result in the metric being throttled, a degradation in the performance of charts, or extra time series overage costs.

Click **Create Metric**.

You'll now see your user-defined metric added to the logs-based metrics screen.



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

Create labels and user defined metrics

Check my progress

## Create the Foodcount alerting policy

Now you'll create an alert policy for Foodcount, the metric you just made.

1. Select **Navigation menu** > **Monitoring** > **Alerting**, and then click **Create Policy**.
2. Click **Add Condition** and set the following:

* For Find resource type and metric, start typing "logging/user/" and select **logging/user/Foodcount**. If this metric doesn't appear, close the window and try again.
* For Configution, Condition **is above**, with a Threshold of **0**, For **1 minute**.
* Click **Add**.

Click **Next**.

1. To receive an email notification, click on **Notification Channels** dropdown and select the checkbox next to your email's display name you have previously created in the lab.

Click **Next**.

1. Name this policy "food alert".

Click **Save**.

When the next round of uptime checks happen, they should violate the policy since the threshold will be higher than 0, and you'll see an Incident on the Monitoring Overview page.

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

Create the Foodcount alerting policy

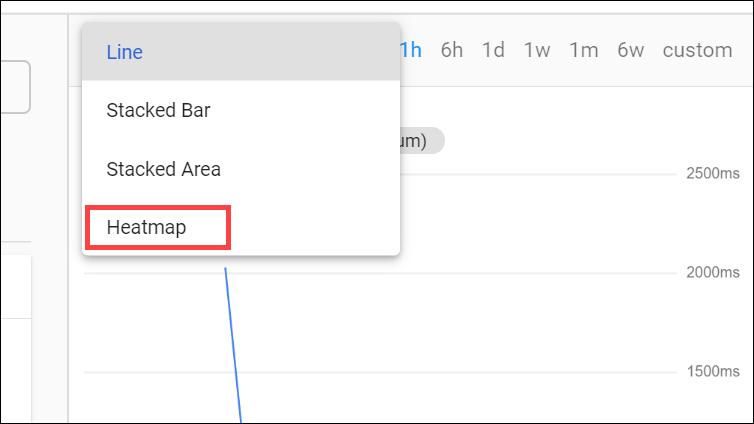
Check my progress

## Custom dashboard with heatmap

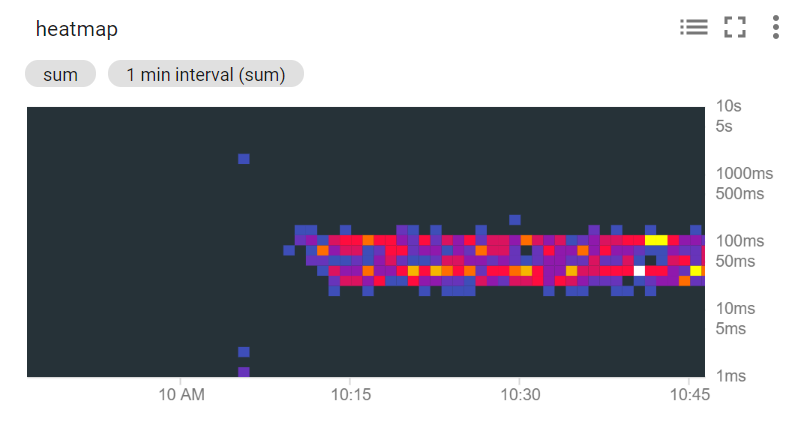
Creating a custom dashboard to show you a specific metric is a great way to see your data. Next you'll create a custom dashboard to show the same latency metric you just used.

1. Still on the Cloud Monitoring window, in the left menu, click **Dashboards** > **Create Dashboard**.
2. Name the Dashboard "App Response" and click **Confirm**.
3. In the top right, click **Add Chart**, and then set the following:

* **Resource type:** GAE Application
* **Metric:** Response latency
* The chart names itself the metric that you're using. You can change the Chart Title to whatever you want - this example uses "Heatmap".
* Above the chart, change chart type to **Heatmap**:



1. Click **Save**.



## Check for the alert

Still in the **Cloud Monitoring** window, in the left menu, click **Monitoring overview** to check for alerts from the uptime check policies.

To see the newVM alert more clearly, click the policy listed in the **Alerting** section.

To see the status of uptime checks in each region, in the left menu, click **Uptime Checks**.

If you set up any email notifications, check the account you added to your alert to verify that you've been notified. These may take longer to arrive, but you can see the emails even after the lab has ended.



# Autoscaling an Instance Group with Custom Cloud Monitoring Metrics

1 hourFree

## GSP087



## Overview

This lab will you will create a [Compute Engine](https://cloud.google.com/compute/docs/) managed instance group that autoscales based on the value of a custom [Cloud Monitoring](https://cloud.google.com/monitoring/docs/) metric.

### **Objectives**

* Deploy an autoscaling Compute Engine instance group.
* Create a custom metric used to scale the instance group.
* Use the [Cloud Console](https://cloud.google.com/storage/docs/cloud-console) to visualize the custom metric and instance group size.

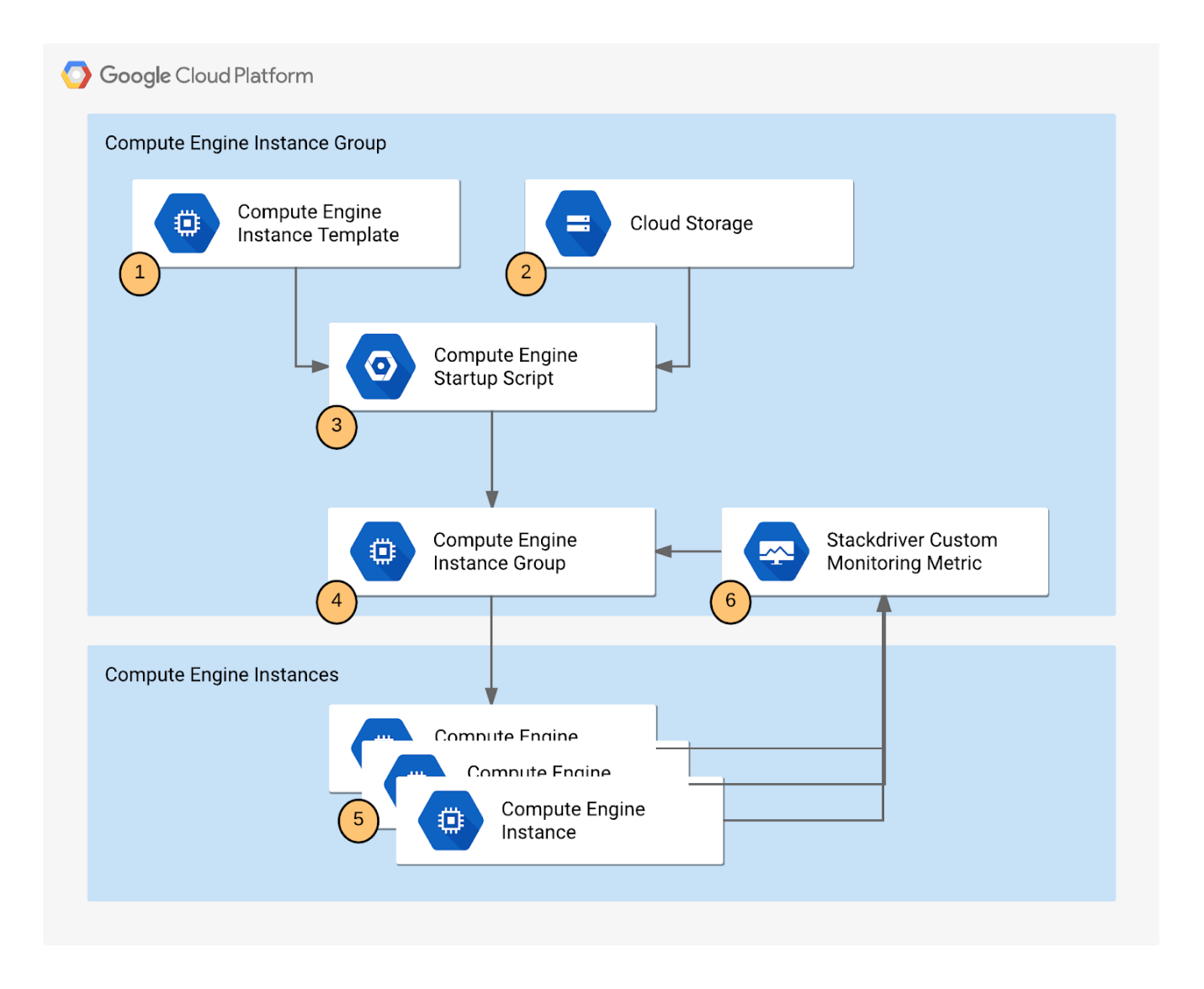
### **Application architecture**

The autoscaling application uses a Node.js script installed on Compute Engine instances. The script reports a numeric value to a Cloud monitoring metric. You do not need to know Node.js or JavaScript for this lab. In response to the value of the metric, the application autoscales the Compute Engine instance group up or down as needed.

The Node.js script is used to seed a custom metric with values that the instance group can respond to. In a production environment, you would base autoscaling on a metric that is relevant to your use case.

The application includes the following components:

1. **Compute Engine instance template** - A template used to create each instance in the instance group.
2. **Cloud Storage** - A bucket used to host the startup script and other script files.
3. **Compute Engine startup script** - A startup script that installs the necessary code components on each instance. The startup script is installed and started automatically when an instance starts. When the startup script runs, it in turn installs and starts code on the instance that writes values to the Cloud monitoring custom metric.
4. **Compute Engine instance group** - An instance group that autoscales based on the Cloud monitoring metric values.
5. **Compute Engine instances** - A variable number of Compute Engine instances.
6. **Custom Cloud Monitoring metric** - A custom monitoring metric used as the input value for Compute Engine instance group autoscaling.



## 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:

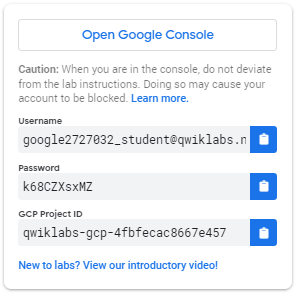
* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

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

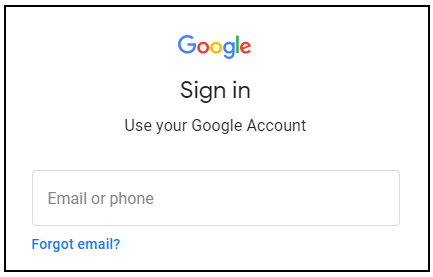
**Note:** If you are using a Pixelbook, open an Incognito window to run this lab.

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

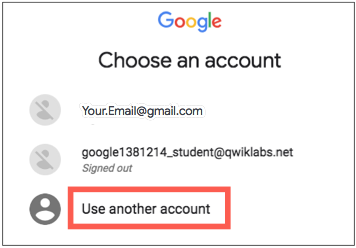
1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



1. 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.

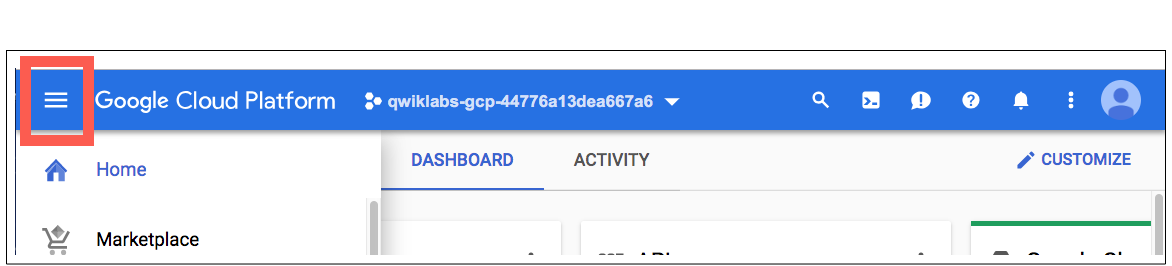
If you see the **Choose an account** page, click **Use Another Account**. 

1. 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).

1. 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.

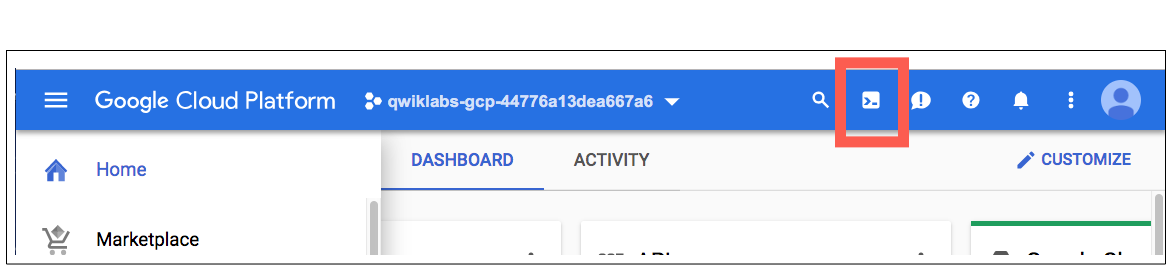
After a few moments, the Cloud Console opens in this tab.

**Note:** You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left. 

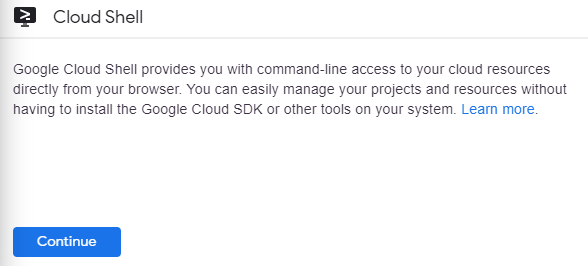
### **Activate Cloud Shell**

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.

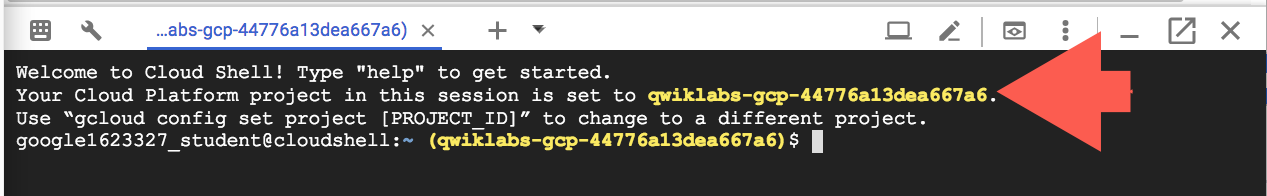
In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



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:



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

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

For full documentation of gcloud see the [gcloud command-line tool overview](https://cloud.google.com/sdk/gcloud" \t "_blank).

## Creating the application

Creating the autoscaling application requires downloading the necessary code components, creating a managed instance group, and configuring autoscaling for the managed instance group.

### **Uploading the script files to Cloud Storage**

During autoscaling, the instance group will need to create new Compute Engine instances. When it does, it creates the instances based on an instance template. Each instance needs a startup script. Therefore, the template needs a way to reference the startup script. Compute Engine supports using Cloud Storage buckets as a source for your startup script. In this section, you will make a copy of the startup script and application files for a sample application used by this lab that pushes a pattern of data into a custom Cloud logging metric that you can then use to configure as the metric that controls the autoscaling behavior for an autoscaling group.

**Note:** There is a pre-existing instance template and group that has been created automatically by the lab that is already running. Autoscaling requires at least 30 minutes to demonstrate both scale-up and scale-down behavior, and you will examine this group later to see how scaling is controlled by the variations in the custom metric values generated by the custom metric scripts.

In the Cloud Console, click **Navigation menu** > **Storage**, then click **Create bucket**.

Give your bucket a unique name, but don't use a name you might want to use in another project. For details about how to name a bucket, see the [bucket naming guidelines](https://cloud.google.com/storage/docs/naming). This bucket will be referenced as YOUR\_BUCKET throughout the lab.

Accept the default values then click **Create**.

When the bucket is created, the Bucket details window opens.

#### Test Completed Task

Click **Check my progress** to verify your performed task. If you have successfully created a Cloud Storage bucket, you will see an assessment score.

Create a Cloud Storage bucket

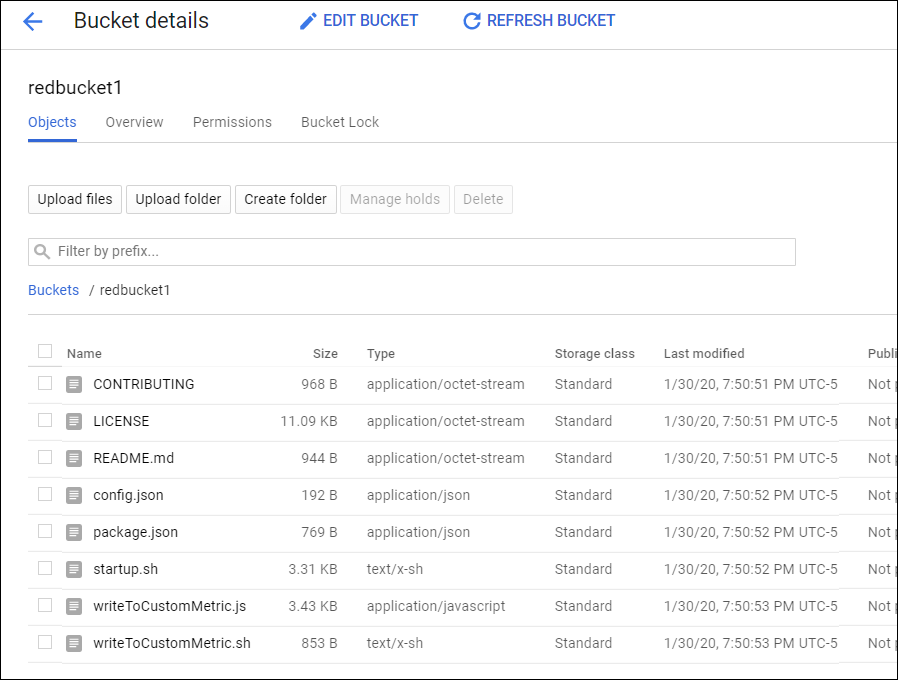
Check my progress

Copy the startup script files from the lab default Cloud Storage bucket to your Cloud Storage bucket by running the following command in Cloud Shell. Remember to replace with the name of the bucket you just made.

gsutil cp -r gs://spls/gsp087/\* gs://<YOUR BUCKET>

content\_copy

After you upload the scripts, the Bucket details window for your bucket should list the added files. You may have to refresh your bucket.



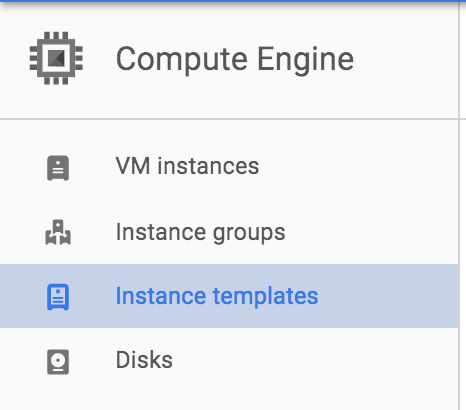
### **Understanding the code components**

* Startup.sh - A shell script that installs the necessary components to each Compute Engine instance as the instance is added to the managed instance group.
* writeToCustomMetric.js -A Node.js snippet that creates a custom monitoring metric whose value triggers scaling. To emulate real-world metric values, this script varies the value over time. In a production deployment, you replace this script with custom code that reports the monitoring metric that you're interested in, such as a processing queue value.
* Config.json - A Node.js config file that specifies the values for the custom monitoring metric and used in writeToCustomMetric.js.
* Package.json - A Node.js package file that specifies standard installation and dependencies for writeToCustomMetric.js.
* writeToCustomMetric.sh - A shell script that continuously runs the writeToCustomMetric.js program on each Compute Engine instance.

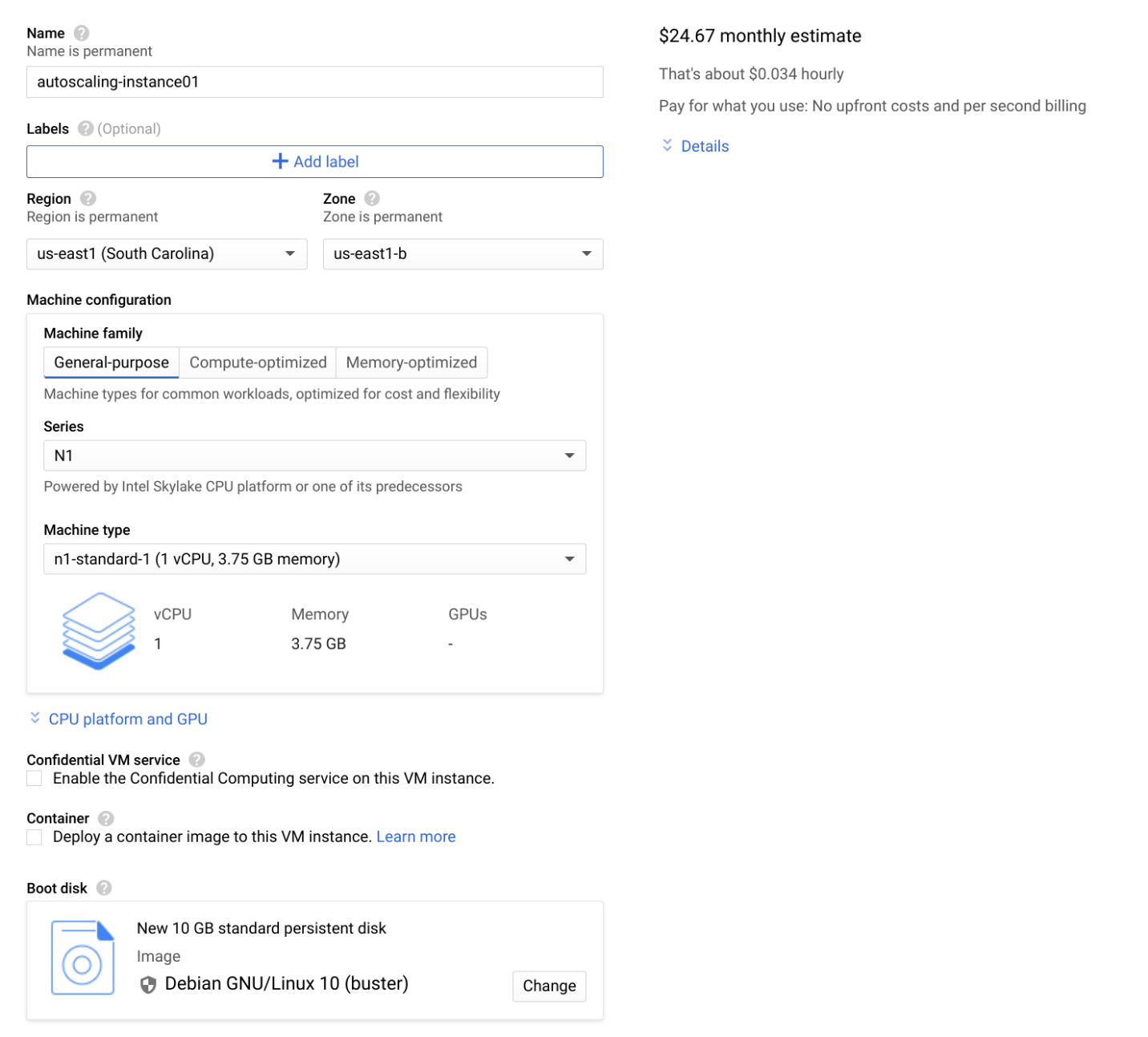
## Creating an instance template

Next, create a template for the instances that are created in the instance group that will use autoscaling. As part of the template, you specify the location (in Cloud Storage) of the startup script that should run when the instance starts.

1. In the Cloud Platform console, go to **Navigation menu** > **Compute Engine** > **Instance templates**.

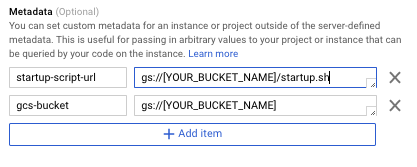


1. Click **Create Instance Template** at the top of the page.
2. Name the instance template autoscaling-instance01.



1. Scroll down, click **Management, security, disks, networking, sole tenancy** to expand the input options.
2. In the **Metadata** section of the **Management** tab, enter these metadata keys and values, clicking the **+ Add item** button to add each one. Remember to substitute your bucket name for the [YOUR\_BUCKET\_NAME] placeholder:

|  |  |
| --- | --- |
| **Key** | **Value** |
| startup-script-url | gs://[YOUR\_BUCKET\_NAME]/startup.sh |
| gcs-bucket | gs://[YOUR\_BUCKET\_NAME] |



1. Click **Create**.

#### Test Completed Task

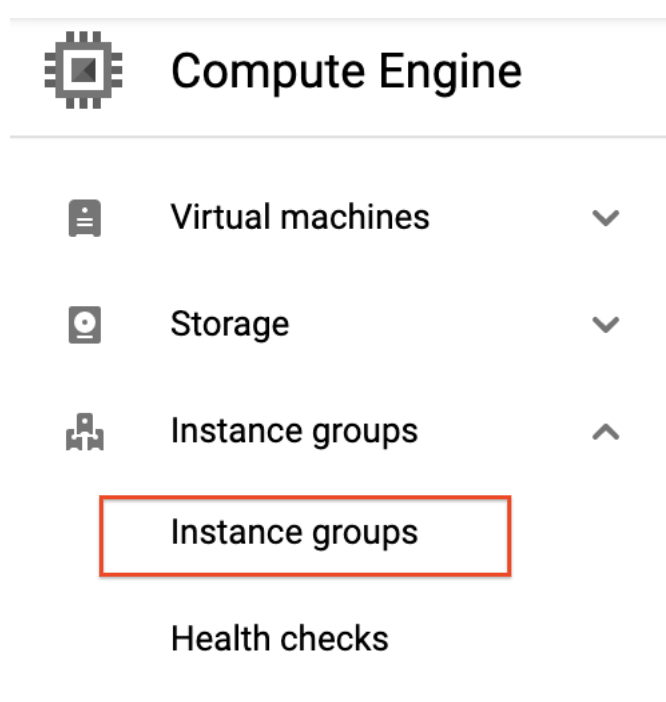
Click **Check my progress** to verify your performed task. If you have successfully created an instance template, you will see an assessment score.

Create an instance template

Check my progress

## Creating the instance group

1. In the left pane, click **Instance groups**.



1. Click **Create instance group**.
2. **Name:** autoscaling-instance-group-1.
3. Under **Instance template**, select the instance template you just created.
4. Set **Autoscaling mode** to **Don't autoscale**.

You'll edit the autoscaling setting after the instance group has been created. Leave the other settings at their default values.

1. Click **Create**.

#### Test Completed Task

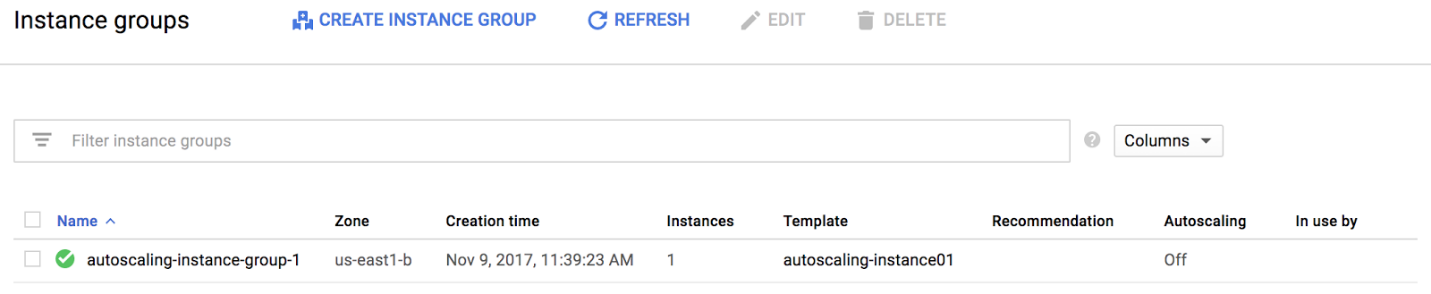
Click **Check my progress** to verify your performed task. If you have successfully created an instance group, you will see an assessment score.

Create an instance group

Check my progress

## Verifying that the instance group has been created

If you don't see the green icon, wait a short while and click the refresh icon. It might take the startup script several minutes to complete installation and begin reporting values. Click **Refresh** if it seems to be taking more than a few minutes.



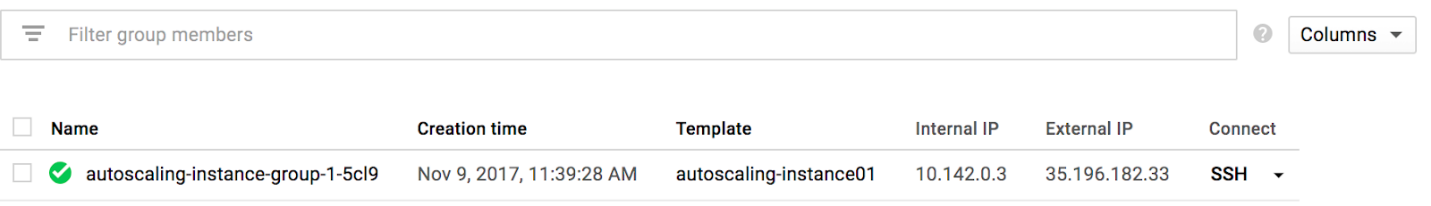
**Note:** If you see a red icon next to the other instance group that was pre-created by the lab, you can ignore this warning. The instance group reports a warning for up to ten minutes as it is initializing. This is expected behavior.

## Verifying that the Node.js script is running

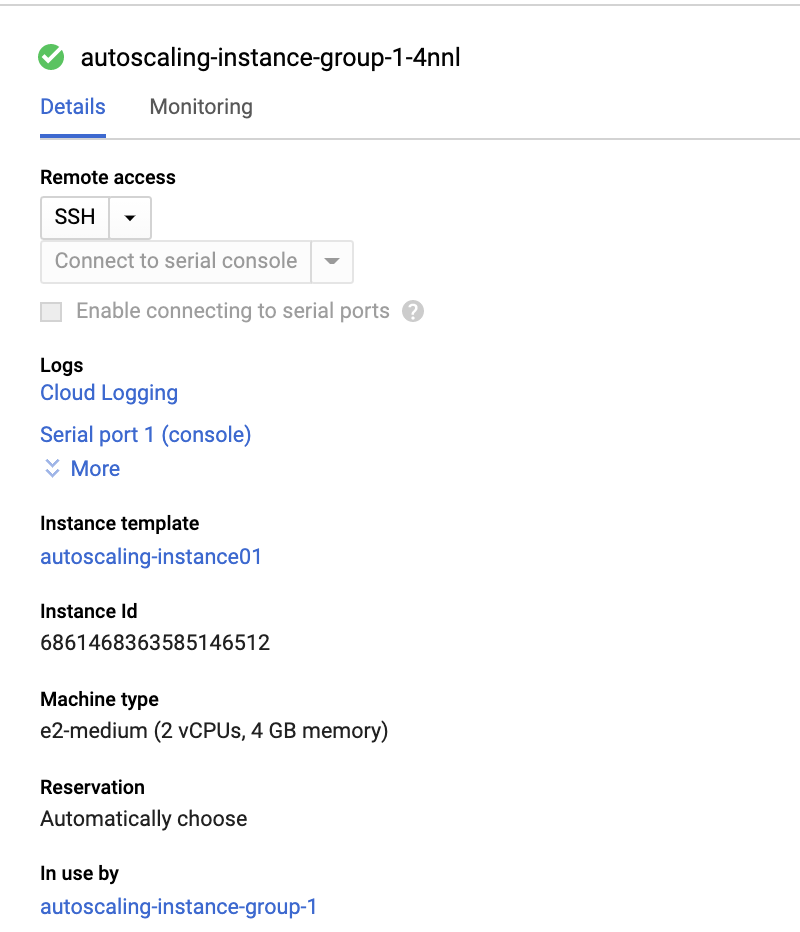
The custom metric custom.googleapis.com/appdemo\_queue\_depth\_01 isn't created until the first instance in the group is created and that instance begins reporting custom metric values.

You can verify that the writeToCustomMetric.js script is running on the first instance in the instance group by checking whether the instance is logging custom metric values.

1. Still in the Compute Engine Instance groups window, click the name of the autoscaling-instance-group-1 to display the instances that are running in the group.
2. Click the instance name. Because autoscaling has not started additional instances, there is just a single instance running.



1. In the **Details** tab, click **Cloud Logging** to view the logs for the VM instance.



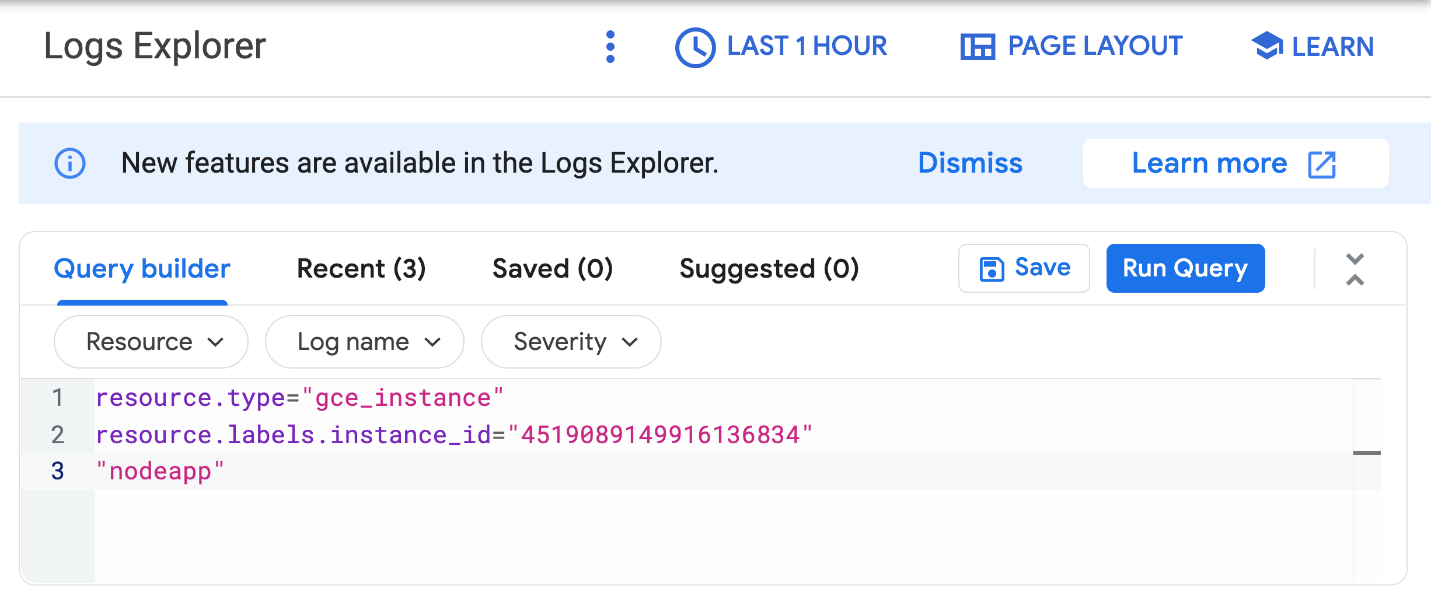
1. Wait a minute or 2 to let some data accumulate. You will see resource.type and resource.labels.instance\_id in the **Query preview** box.



1. Now click drop-down arrow next to **Run Query** to open **Query builder** box.



1. Add "nodeapp" as line three, so the code looks similar to this:



1. Click **Run Query**.

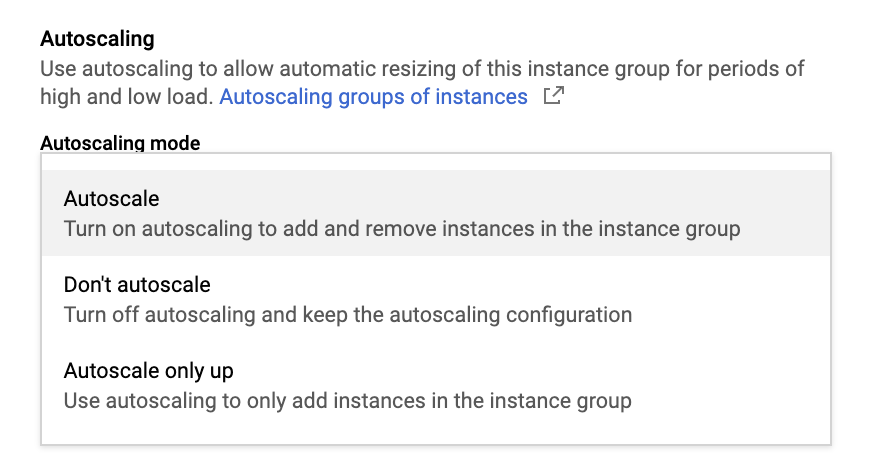
If the Node.js script is being executed on the Compute Engine instance, a request is sent to the API, and log entries that say Finished writing time series data appear in the logs. For example, in the preceding screenshot, entries like this appear at 10:31:05.000 and 10:30:53.000.

If you don't see this log entry, the Node.js script isn't reporting the custom metric values. Check that the metadata was entered correctly. If the metadata is incorrect, it might be easiest to restart the lab.

## Configure autoscaling for the instance group

After you've verified that the custom metric is successfully reporting data from the first instance, the instance group can be configured to autoscale based on the value of the custom metric.

1. In the Cloud Console, go to **Compute Engine** > **Instance groups**.
2. Click the autoscaling-instance-group-1 group and then click **Configure autoscaling**.
3. Set **Autoscaling mode** to **Autoscale**.



1. Click on **Autoscaling configuration** and then click on **pencil icon** to edit metric. Set the following fields, leave all others at the default value.

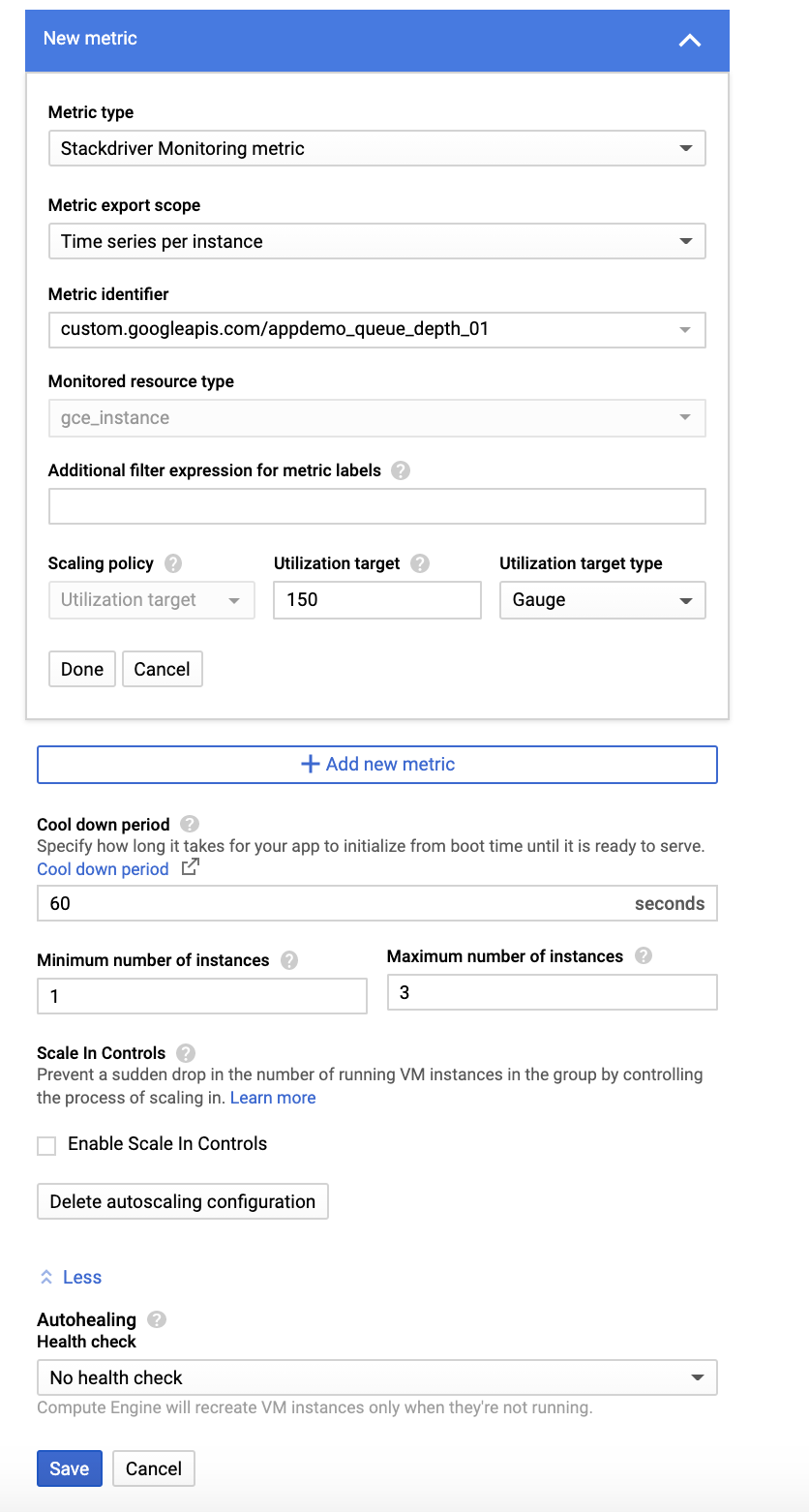
* **Metric Type**: Stackdriver Monitoring metric
* **Metric export scope**: Time series per instance
* **Metric identifier**: custom.googleapis.com/appdemo\_queue\_depth\_01
* **Utilization target**: 150

When custom monitoring metric values are higher or lower than the **Target** value, the autoscaler scales the managed instance group, increasing or decreasing the number of instances. The target value can be any [double](https://cloud.google.com/monitoring/api/ref_v3/rest/v3/projects.metricDescriptors#valuetype) value, but for this lab, the value 150 was chosen because it matches the values being reported by the custom monitoring metric.

* **Utilization target type**: Gauge

The **Gauge** setting specifies that the autoscaler should compute the average value of the data collected over the last few minutes and compare it to the target value. (By contrast, setting **Target mode** to **DELTA\_PER\_MINUTE** or **DELTA\_PER\_SECOND** autoscales based on the observed rate of change rather than an average value.)

* **Minimum number of instances**: 1
* **Maximum number of instances**: 3



1. Click **Save**.

#### Test Completed Task

Click **Check my progress** to verify your performed task. If you have successfully configured autoscaling for the instance group, you will see an assessment score.

Configure autoscaling for the instance group

Check my progress

## Watching the instance group perform autoscaling

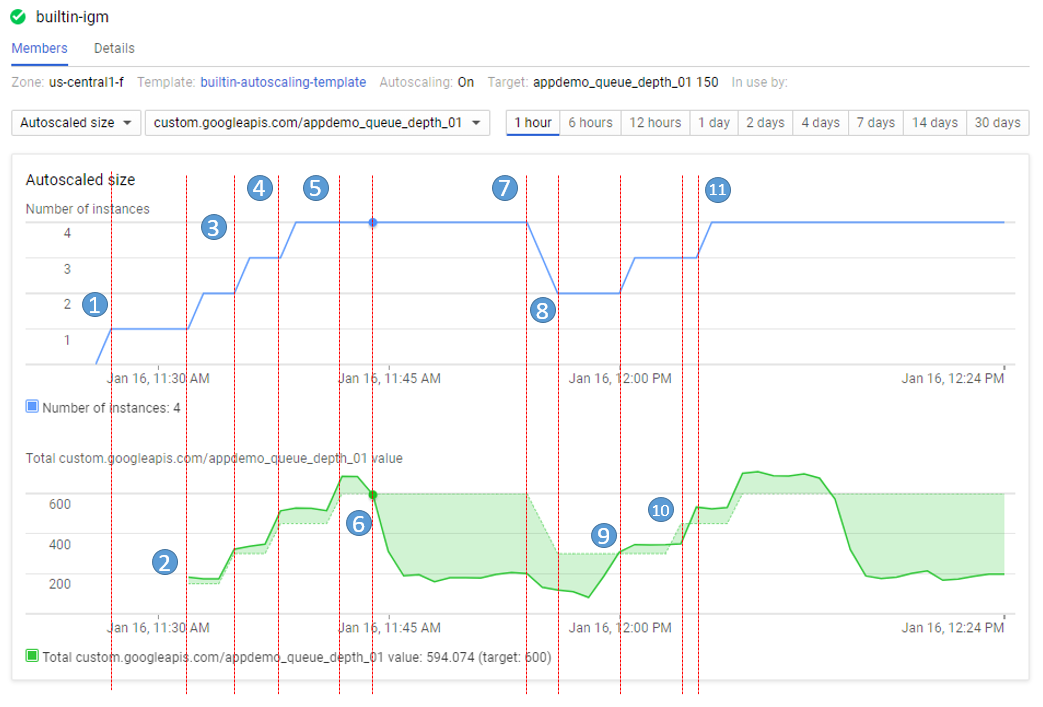
The Node.js script varies the custom metric values it reports from each instance over time. As the value of the metric goes up, the instance group scales up by adding Compute Engine instances. If the value goes down, the instance group detects this and scales down by removing instances. As noted earlier, the script emulates a real-world metric whose value might similarly fluctuate up and down.

Next you will see how the instance group is scaling in response to the metric by clicking the **Monitoring** tab to view the **Autoscaled size** graph

1. In the left pane, click **Instance groups**.
2. Click the builtin-igm instance group in the list.
3. Click the **Monitoring** tab.

Since this group had a head start, you can see the autoscaling details about the instance group in the autoscaling graph. The autoscaler will take about five minutes to correctly recognize the custom metric and it can take up to ten minutes for the script to generate sufficient data to trigger the autoscaling behavior shown below. You can switch back to the instance group that you created to see how it's doing.

The number of instances depicted in the top graph changes as a result of the varying aggregate level of the custom metric property values reported in the lower graph. There is a slight delay of up to five minutes after each instance starts up before that instance begins to report its custom metric values. While your autoscaling starts up, read through this graph to understand what will be happening:



The script starts by generating high values for approximately 15 minutes in order to trigger scale-up behavior.

* **11:27** Autoscaling Group starts with a single instance. The aggregate custom metric target is 150.
* **11:31** Initial metric data acquired. As the metric is greater than the target of 150 the autoscaling group starts a second instance.
* **11:33** Custom metric data from the second instance starts to be acquired. The aggregate target is now 300. As the metric value is above 300 the autoscaling group starts the third instance.
* **11:37** Custom metric data from the third instance starts to be acquired. The aggregate target is now 450. As the cumulative metric value is above 450 the autoscaling group starts the fourth instance.
* **11:42** Custom metric data from the fourth instance starts to be acquired. The aggregate target is now 600. The cumulative metric value is now above the new target level of 600 but since the autoscaling group size limit has been reached no additional scale-up actions occur.
* **11:44** The application script has moved into a low metric 15 minute period. Even though the cumulative metric value is below the target of 600 scale-down must wait for a ten minute built-in scale-down delay to pass before making any changes.
* **11:54** Custom metric data has now been below the aggregate target level of 600 for a four node cluster for over 10 minutes. Scale-down down now removes two instances in quick succession.
* **11:56** Custom metric data from the removed nodes is eliminated from the autoscaling calculation and the aggregate target is reduced to 300.
* **12:00** The application script has moved back into a high metric 15 minute period. The cumulative custom metric value has risen above the aggregate target level of 300 again so the autoscaling group starts a third instance.
* **12:03** Custom metric data from the new instance have been acquired but the cumulative values reported remain below the target of 450 so autoscaling makes no changes.
* **12:04** Cumulative custom metric values rise above the target of 450 so autoscaling starts the fourth instance.

For the remainder of the time on your lab, you can watch the autoscaling graph move up and down as instances are added and removed.