Reporting Application Metrics into Cloud Monitoring

qwiklabs.com/focuses/1259

GSP111

Google Cloud Self-Paced Labs

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. OpenCensus refers to metrics as *stats*.

What is OpenCensus?

OpenCensus 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:

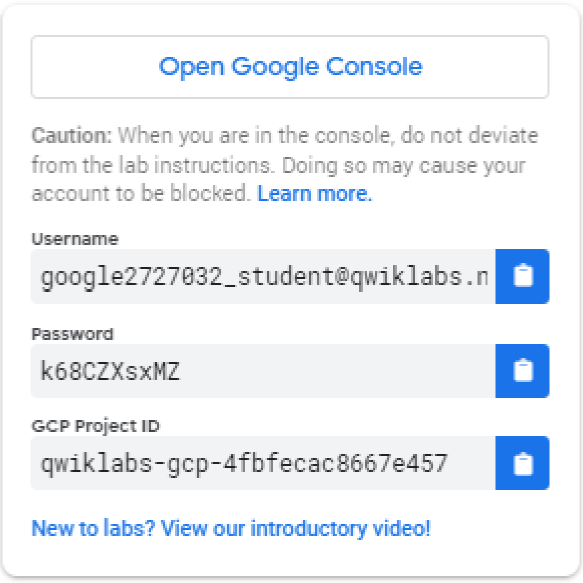
- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.

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.



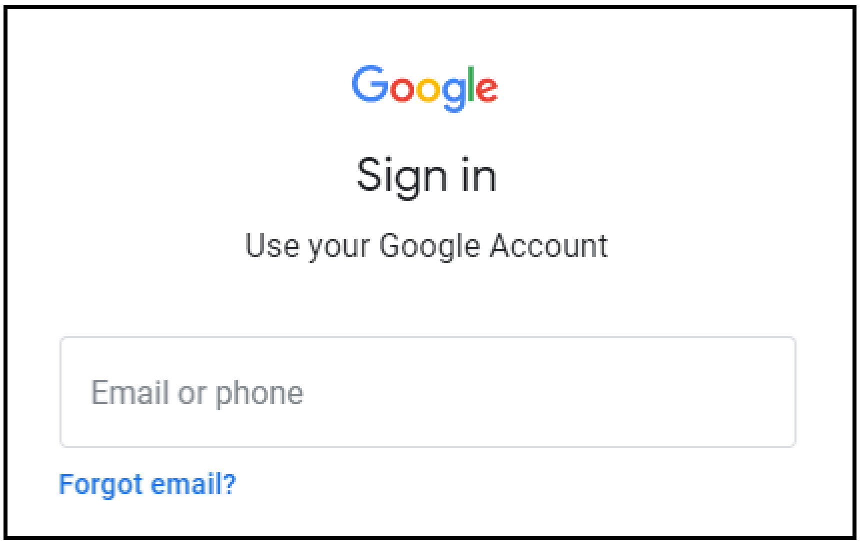
Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

Username
google2727032_student@qwiklabs.n

Password
k68CZXsxMZ

GCP Project ID
qwiklabs-gcp-4fbfecac8667e457

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



Google

Sign in

Use your Google Account

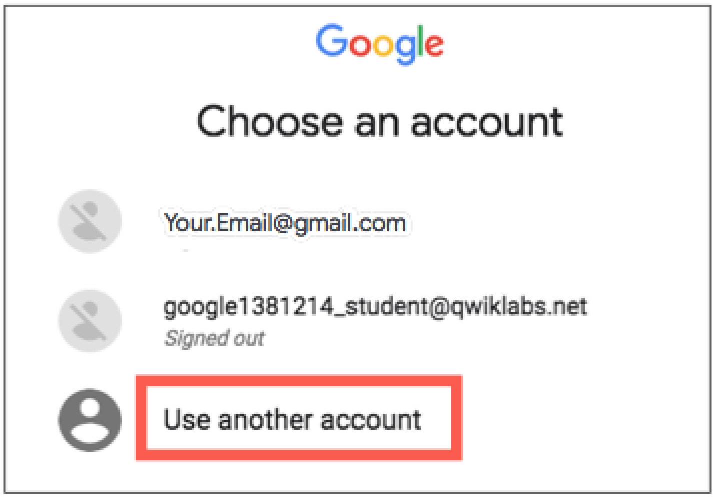
Email or phone

Forgot email?

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

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

3. In the **Sign in** page, paste the username that you copied from the Connection Details panel. Then copy and paste the password.

   Google

   Choose an account

   Your.Email@gmail.com

   google1381214_student@qwiklabs.net
   Signed out

   Use another account

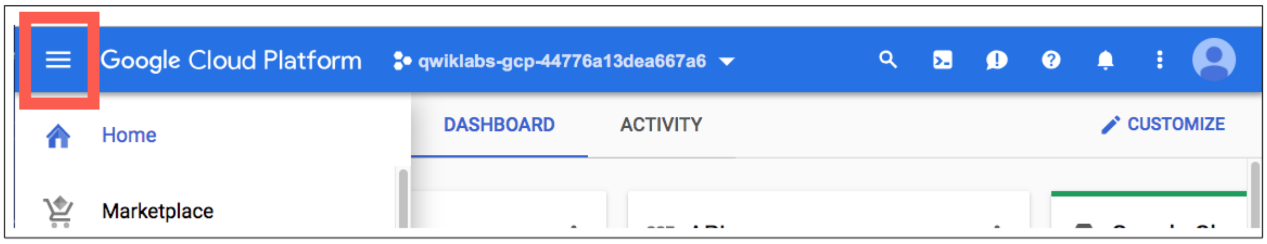
   Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

4. Click through the subsequent pages:

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

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

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



Create a Compute Engine instance

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

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

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

Install Go and OpenCensus on your instance

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

You will 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/go1.16.2.linux-amd64.tar.gz

sudo tar -xvf go1.16.2.linux-amd64.tar.gz

sudo rm -rf /usr/local/go

sudo mv go /usr/local

sudo apt-get update

sudo apt-get install git
```

You will be asked to confirm you want to continue - type in **Y** and then press **Enter**.

```
export PATH=$PATH:/usr/local/go/bin
go get go.opencensus.io
go get contrib.go.opencensus.io/exporter/stackdriver

go mod init test3

go mod tidy
```

"go mod tidy" removes unused dependencies. You will get output *go: warning: "all" matched no packages* if you have no unused dependencies.

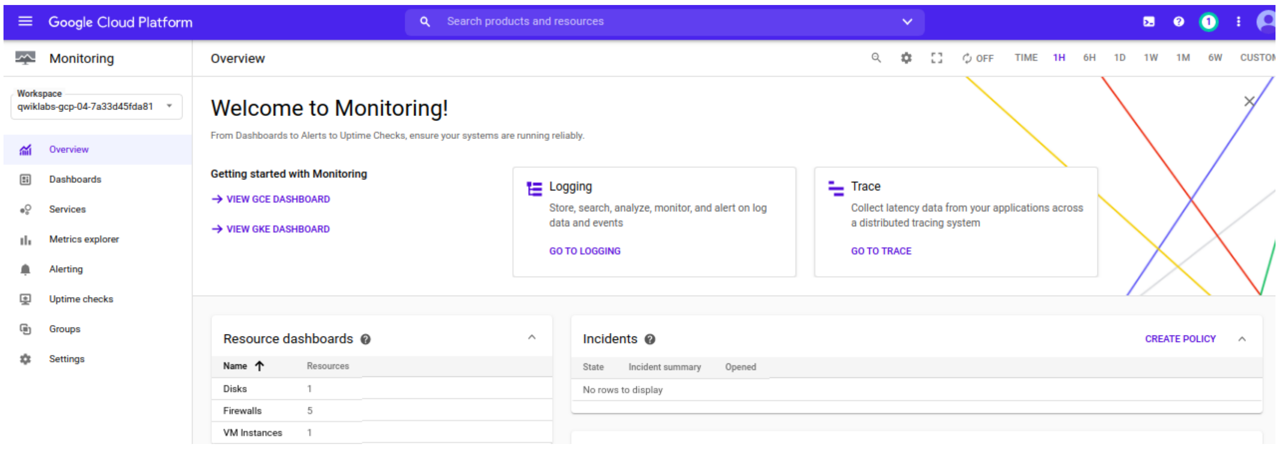
Create a Monitoring workspace

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.



Workspace
qwiklabs-gcp-04-7a33d45fda81

Welcome to Monitoring!

From Dashboards to Alerts to Uptime Checks, ensure your systems are running reliably.

Getting started with Monitoring

Logging
Store, search, analyze, monitor, and alert on log data and events

Trace
Collect latency data from your applications across a distributed tracing system

Resource dashboards

Name	Resources
Disks	1
Firewalls	5
VM Instances	1

Incidents

State	Incident summary	Opened
No rows to display		

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 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 following 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.sh

sudo apt-get update

sudo apt-get install stackdriver-agent
```

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

Now 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.sh

sudo apt-get update

sudo apt-get install google-fluentd
```

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

Create Monitoring workspace and install agents on the VM

Create a basic application server in Go

Next, 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, use your favorite editor (`nano` , `vi` , etc.) to create a file called `main.go` written in Go. For example:

```
nano main.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
}
```

Save the file by pressing **Ctrl + X**, **Y** and **Enter**.

Run the file with:

```
go run main.go
```

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

```
Queue size: 11
Queue size: 52
...
```

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:

```
nano main.go
```

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
}
```

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 the 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
}
```

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 {
```

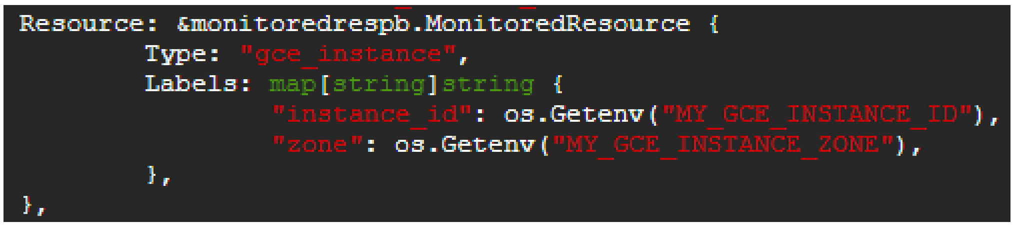
```
time.Sleep(1 * time.Second)
queueSize := getQueueSize()

// 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
}
```

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` :



```
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"),
        },
},
```

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 by pressing **Ctrl** + **X**, **Y** and **Enter** .

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 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-a
```

```
go mod tidy
```

Now run your application:

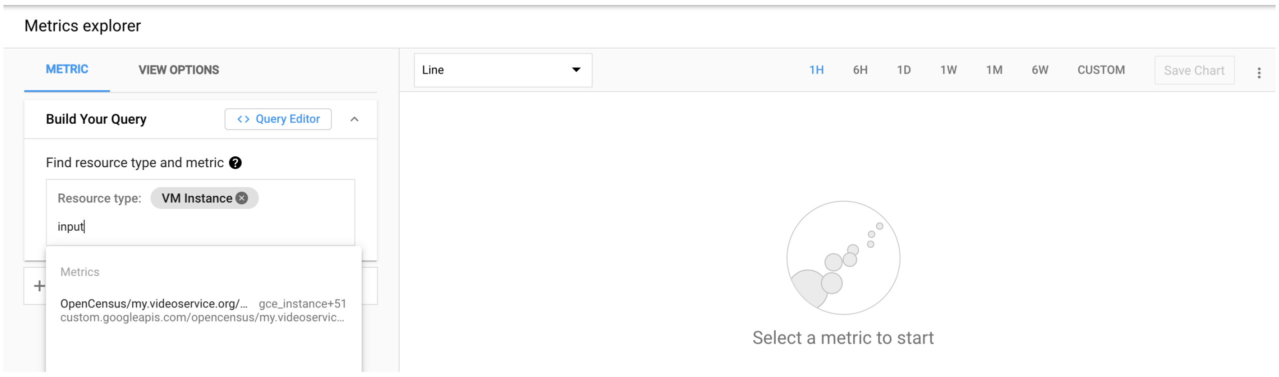
```
go run main.go
```

Note: If you see errors, you can ignore them.
You can leave the application running.

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

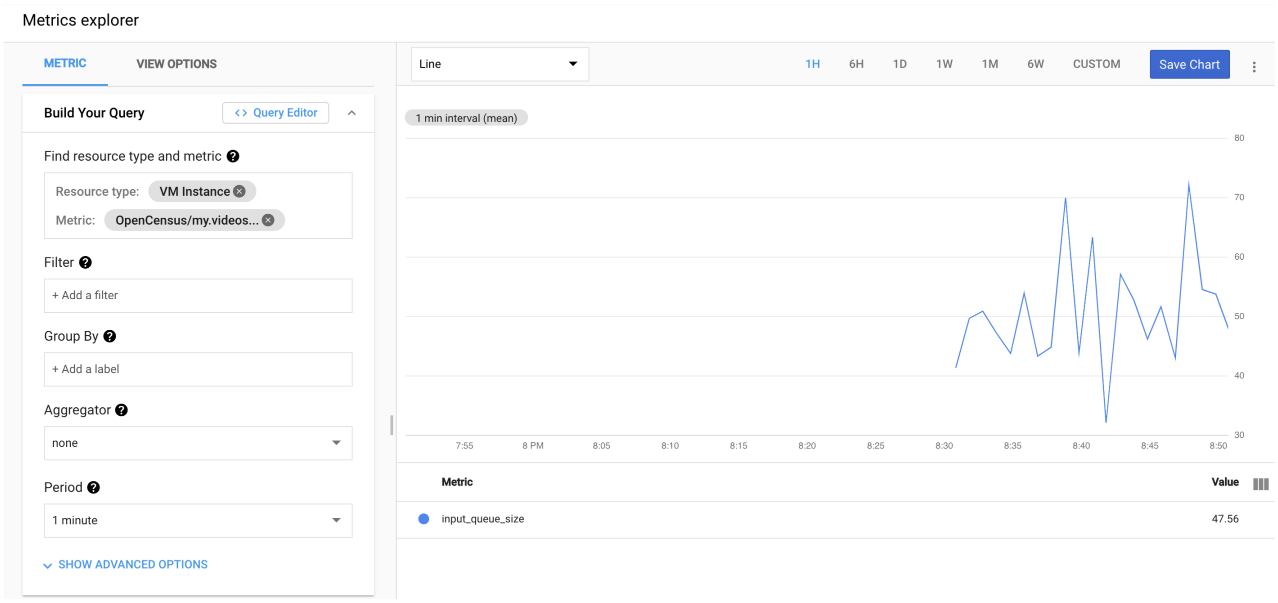
In the left menu, click **Metrics Explorer**.

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



Resource type: VM Instance
input
Metrics
OpenCensus/my.videoservice.org/... gce_instance+51
custom.googleapis.com/opencensus/my.videoservic...

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:


Metrics explorer

Find resource type and metric

Resource type: VM Instance

Metric: OpenCensus/my.videos...

Filter: + Add a filter

Group By: + Add a label

Aggregator: none

Period: 1 minute

Metric	Value
● input_queue_size	47.56

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

View your application metrics in Cloud Monitoring

Other methods for reporting application metrics to Cloud Monitoring

Users who are instrumenting their code for the first time should use the OpenCensus library, as shown in this lab. Users with pre-existing instrumentation should consider one of the existing adapters:

- Prometheus: Use the Prometheus integration (beta).
- StatsD and anything else supported by collectd: Use the Cloud Monitoring agent. See the Custom Metrics from the Agent documentation to get this setup. The documentation on Custom Metrics might also be valuable.

While Cloud Monitoring offers a public API, the CreateTimeSeries API, 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.

Congratulations!

You have successfully written a simple application that uses OpenCensus to export metrics to Cloud Monitoring.

Finish Your Quest

This self-paced lab is part of the Qwiklabs Google Cloud's Operations Suite Quest. A Quest is a series of related labs that form a learning path. Completing this Quest earns you the badge above, to recognize your achievement. You can make your badge public and link to them in your online resume or social media account. Enroll in this Quest and get immediate completion credit if you've taken this lab. See other available Qwiklabs Quests.

Take your Next Lab

Continue your Quest with Monitoring Multiple Projects with Cloud Monitoring, or check out these suggestions:Learn more about Cloud Monitoring in our other labs:

- Monitoring and Logging for Cloud Functions
- Autoscaling an Instance Group with Custom Cloud Monitoring Metrics

Next Steps / Learn More

Here are links to the full Cloud Monitoring documentation, with details for each product:

- Cloud Logging: https://cloud.google.com/logging/docs/
- Cloud Monitoring: https://cloud.google.com/monitoring/docs/
- Cloud Trace: https://cloud.google.com/trace/docs/
- Cloud Error Reporting: https://cloud.google.com/error-reporting/docs/
- Cloud Debugger: https://cloud.google.com/debugger/docs
- See https://godoc.org/go.opencensus.io/stats for more details.

Google Cloud Training & Certification

...helps you make the most of Google Cloud technologies. Our classes include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. Certifications help you validate and prove your skill and expertise in Google Cloud technologies.

Manual Last Updated May 31, 2021

Lab Last Tested May 31, 2021

Continue questing

Lab

Cloud Monitoring: Qwik Start