Solutions (https://cloud.google.com/solutions/) Solutions

Capturing Bigtable tracing and metrics using Stackdriver Trace, Stackdriver Monitoring, and OpenCensus

This tutorial shows how to implement client-side tracing and metrics recording in your Cloud Bigtable (https://cloud.google.com/bigtable) workloads by using OpenCensus (https://opencensus.io/), Trace (https://cloud.google.com/trace/), and Stackdriver Monitoring (https://cloud.google.com/monitoring/).

Although Bigtable surfaces several helpful server-side metrics using Trace, apps can realize added benefits by implementing client-side tracing, instrumenting, and app-defined metrics. For example, server-side metrics don't give you a window into the round-trip latency of calls made to your Bigtable endpoint and can only be surfaced using client-side tracing.

OpenCensus is an open source library that you can use to provide observability in your apps. The library is vendor agnostic and integrates with several backends, such as Prometheus and Zipkin. In this tutorial, you use Trace and Monitoring as the backend for tracing and metrics.

To complete the steps in this tutorial, you should be familiar with the Linux command line. Though it's not required, knowledge of the Java programming language helps you understand the example code.

Objectives

- Deploy a Bigtable instance.
- Deploy a Compute Engine virtual machine (VM) for running a sample OpenCensus-instrumented Java client.

- Download, deploy, and run the instrumented Java client app.
- View OpenCensus traces in Stackdriver Trace.
- View OpenCensus metrics in the Metrics Explorer in Monitoring.

Costs

This tutorial uses the following billable components of Google Cloud:

- Compute Engine (https://cloud.google.com/compute/pricing)
- Bigtable (https://cloud.google.com/bigtable/pricing)
- Logging (https://cloud.google.com/logging/pricing)

To generate a cost estimate based on your projected usage, use the pricing calculator (https://cloud.google.com/products/calculator). New Google Cloud users might be eligible for a free trial (https://cloud.google.com/free-trial).

Before you begin

1. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).

2. In the Cloud Console, on the project selector page, select or create a Cloud project.

   Note: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

   GO TO THE PROJECT SELECTOR PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/PROJECTSELECT

3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (https://cloud.google.com/billing/docs/how-to/modify-project).

4. Enable the Compute Engine, Bigtable, and Stackdriver Logging APIs.

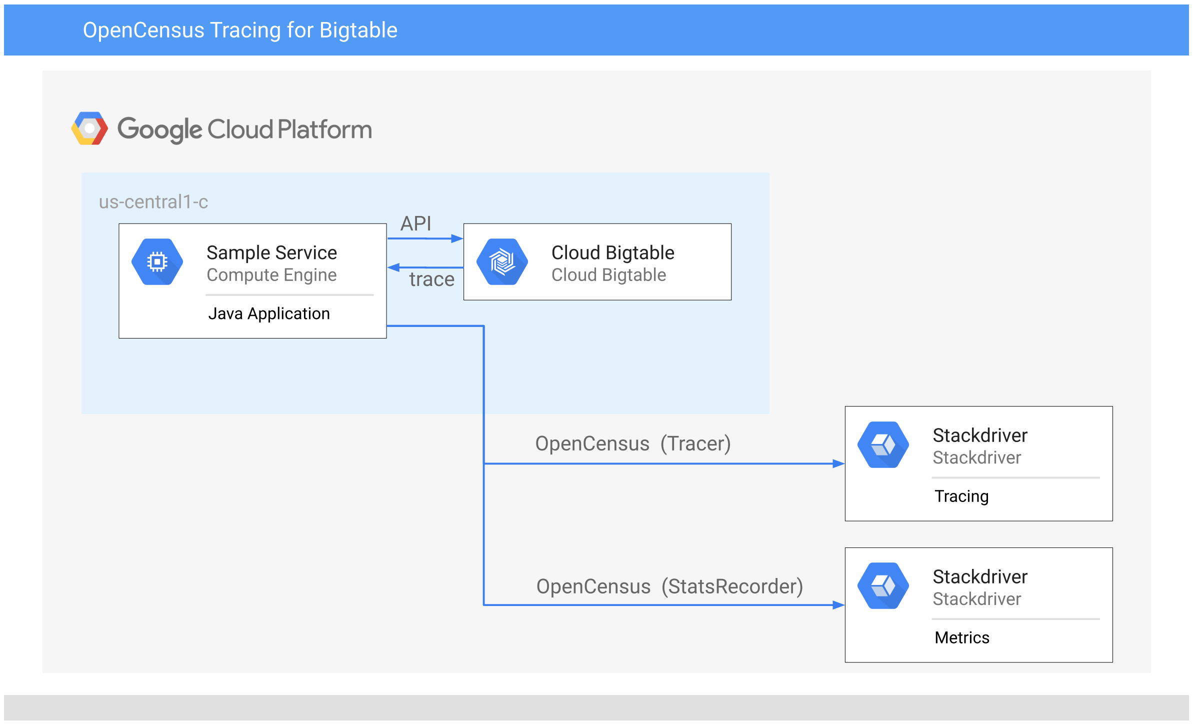
   ENABLE THE APIS (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/FLOWS/ENABLEAPI?APIID=COMPUTE

5. Install and initialize the Cloud SDK (https://cloud.google.com/sdk/docs/).

Reference architecture

For simplicity, in this tutorial you implement all of your client-side logic in a Java console app. For the datastore tier, you use Bigtable, which allows you to focus on the key aspects of client-side tracing and metrics without having to worry about things like database deployments and related configuration.

The following architecture diagram shows a Java console app and a datastore tier.



Creating a Bigtable instance

In this section, you create a Bigtable instance that your Java app uses later in the tutorial.

- In Cloud Shell, create a Bigtable development instance:

```
gcloud bigtable instances create cbt-oc \
  --cluster=cbt-oc \
  --cluster-zone=us-central1-c \
  --display-name=cbt-oc \
  --instance-type=DEVELOPMENT
```

This command might take a few minutes to complete.

Creating and configuring a Compute Engine VM

- In Cloud Shell, create a Compute Engine VM with the security scopes necessary for 0Auth 2.0:

```
gcloud compute instances create trace-client \
    --zone=us-central1-c \
    --scopes="https://www.googleapis.com/auth/bigtable.admin.table,\
https://www.googleapis.com/auth/bigtable.data,\
https://www.googleapis.com/auth/logging.write,\
https://www.googleapis.com/auth/monitoring.write,\
https://www.googleapis.com/auth/trace.append"
```

Sample Java app

This section uses a sample Java app that generates transactions in order to demonstrate the tracing capabilities of OpenCensus and Logging.

App flow

The sample Java app running on the Compute Engine VM does the following:

1. Creates a table in the Bigtable instance.
2. For a series of 10,000 transactions sets, does the following:
   a. Writes a small set of rows.
   b. Reads a single row.
   c. Performs a table scan for those rows.

3. Deletes the table.

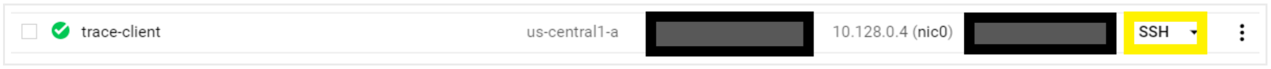
Deploying the sample app

In this section, you download the Java app containing the instrumented code, modify it to reflect your environment, and then run it.

1. In the Cloud Console, go to the **VM instances** page:

   GO TO THE VM INSTANCES PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/COMPUTE/INSTANCES

2. Use SSH to connect to the VM by clicking the **SSH** button (highlighted in the following screenshot):

3. In the VM instance, install Git, the Java 8 JDK, and Maven (https://maven.apache.org/what-is-maven.html):

```
sudo apt-get install git openjdk-8-jdk maven -y
```

4. In the instance, clone the source repository for this tutorial:

```
git clone https://github.com/GoogleCloudPlatform/community.git
```

   You can now update the Java app with some configuration settings specific to your project.

5. Navigate to the folder containing the Java source:

```
cd community/tutorials/bigtable-oc/java/
```

6. Configure the app code to use the `cbt-oc` Bigtable instance:

```
export INSTANCE_ID=cbt-oc
```

7. Now run the Maven commands to build and run the program:

```
mvn package -DskipTests --quiet
mvn exec:java -Dexec.mainClass=com.example.bigtable.App --quiet
```

The output looks similar to the following:

```
...
2019-05-13 23:31:54 INFO  BigtableSession:89 - Opening connection for projectId
2019-05-13 23:31:54 INFO  BigtableSession:89 - Bigtable options: {......}
2019-05-13 23:31:54 INFO  OAuthCredentialsCache:89 - Refreshing the OAuth token
2019-05-13 23:31:55 INFO  App:170 - Create table Hello-Bigtable
2019-05-13 23:35:36 INFO  App:209 - Delete the table
2019-05-13 23:35:36 WARN  BigtableAdmin:116 - Table Hello-Bigtable was disabled
```

Sample app code highlights

In the following code segment, the Bigtable instance is provided to the Java runtime using the environment variable `INSTANCE_ID` that you set previously.

tutorials/bigtable-oc/java/src/main/java/com/example/bigtable/App.java (https://github.com/GoogleCloudPlatform/community/blob/master/tutorials/bigtable-oc/java/src/main/java/com/example/bigtable/App.java)

BLOB/MASTER/TUTORIALS/BIGTABLE-OC/JAVA/SRC/MAIN/JAVA/COM/EXAMPLE/BIGTABLE/APP.JAVA)

```
private static final String PROJECT_ID = ServiceOptions.getDefaultProjectId();
private static final String INSTANCE_ID = System.getenv( "INSTANCE_ID");
```

The code segment below shows how to define a manually labeled tracing scope:

tutorials/bigtable-oc/java/src/main/java/com/example/bigtable/App.java (https://github.com/GoogleCloudPlatform/community/blob/master/tutorials/bigtable-oc/java/src/main/java/com/example/bigtable/App.java)

BLOB/MASTER/TUTORIALS/BIGTABLE-OC/JAVA/SRC/MAIN/JAVA/COM/EXAMPLE/BIGTABLE/APP.JAVA)

```
try (Scope ss = tracer.spanBuilder("opencensus.Bigtable.Tutorial").startScopedSpan()

    // generate unique UUID
    UUID uuid = UUID.randomUUID();
    String randomUUIDString = uuid.toString();

    startRead = System.currentTimeMillis();
    // write to Bigtable
    writeRows(table, randomUUIDString);
    endRead = System.currentTimeMillis();
    startWrite = System.currentTimeMillis();
    // read from Bigtable
    readRows(table, randomUUIDString);
    endWrite = System.currentTimeMillis();
```

Notice the `try` block with the call to `spanBuilder`. This illustrates how the program uses OpenCensus to perform tracing. The call chain that performs the table writes and reads in the `doBigTableOperations` function is instrumented in this way.

The program also configures Stackdriver Trace as the tracing backend:

tutorials/bigtable-oc/java/src/main/java/com/example/bigtable/App.java (https://github.com/GoogleCloudPlatform/community/blob/master/tutorials/bigtable-oc/java/src/main/java/com/example/bigtable/App.java)

BLOB/MASTER/TUTORIALS/BIGTABLE-OC/JAVA/SRC/MAIN/JAVA/COM/EXAMPLE/BIGTABLE/APP.JAVA)

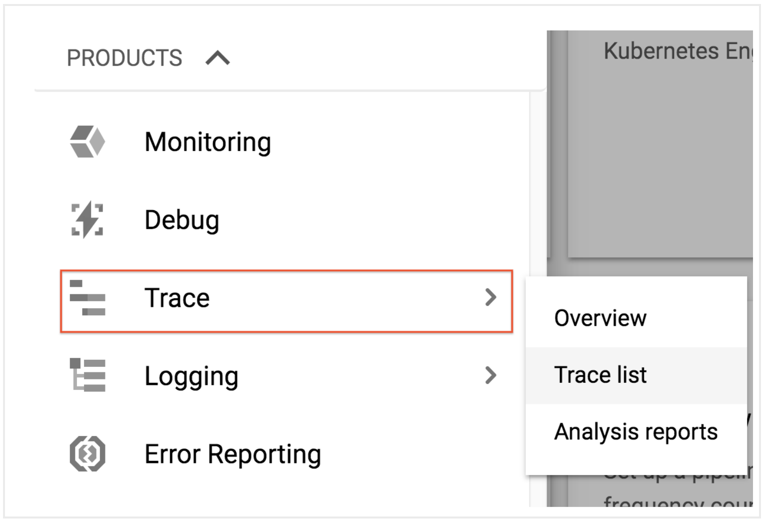
```
private static void configureOpenCensusExporters(Sampler sampler) throws IOException
    TraceConfig traceConfig = Tracing.getTraceConfig();

    // For demo purposes, lets always sample.

    traceConfig.updateActiveTraceParams(
      traceConfig.getActiveTraceParams().toBuilder().setSampler(sampler).build());

    // Create the Stackdriver trace exporter
    StackdriverTraceExporter.createAndRegister(
      StackdriverTraceConfiguration.builder()
        .setProjectId(PROJECT_ID)
        .build());

    // [Start Stackdriver Monitoring]
    StackdriverStatsExporter.createAndRegister();
```

Viewing traces in Stackdriver Trace UI

The sample program performs 10,000 transaction sets: three writes and a range read. The exporter is configured to record one trace sample for every 1,000 transaction sets. As a result, 10 or 11 traces are captured over the run of the program.

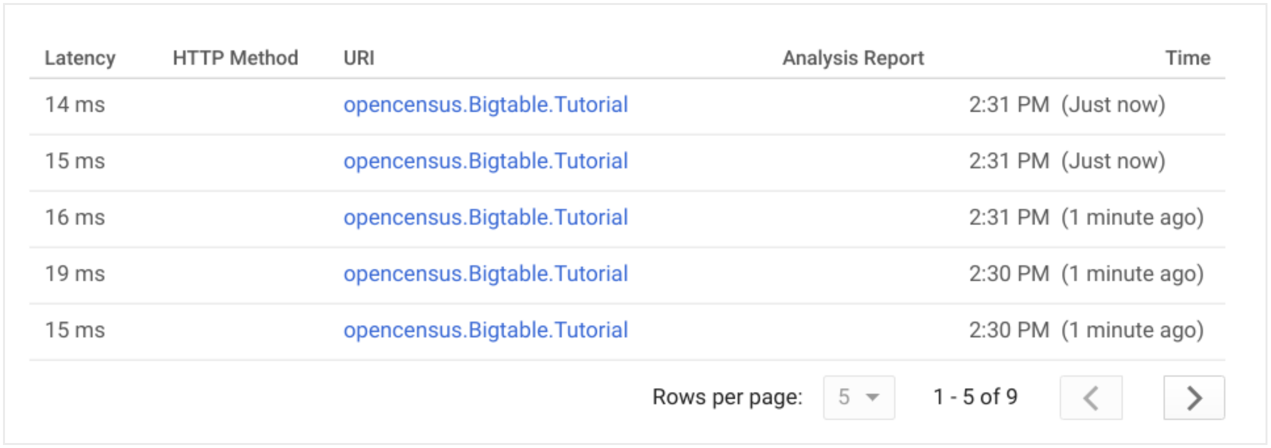
After the program has been running for a short time, do the following:

1. In the Cloud Console, go to Trace (https://cloud.google.com/console/traces):



2. Click **Trace List**.

   On the right, you should see a table similar to the following:



Latency	HTTP Method	URI	Analysis Report	Time
14 ms		opencensus.Bigtable.Tutorial		2:31 PM (Just now)
15 ms		opencensus.Bigtable.Tutorial		2:31 PM (Just now)
16 ms		opencensus.Bigtable.Tutorial		2:31 PM (1 minute ago)
19 ms		opencensus.Bigtable.Tutorial		2:30 PM (1 minute ago)
15 ms		opencensus.Bigtable.Tutorial		2:30 PM (1 minute ago)

Rows per page: 5 1 - 5 of 9

   The sampling rate for the traces is set to record one trace for every 1,000 transactions.

   The tracing label `opencensus.Bigtable.Tutorial` in the **Timeline** is the name of the outermost tracing scope that is defined in the following code snippet.

   tutorials/bigtable-oc/java/src/main/java/com/example/bigtable/App.java (https://github.com/GoogleCloudPlatform/community/blob/master/tutorials/bigtable-oc/java/src/main/java/com/example/bigtable/App.java)

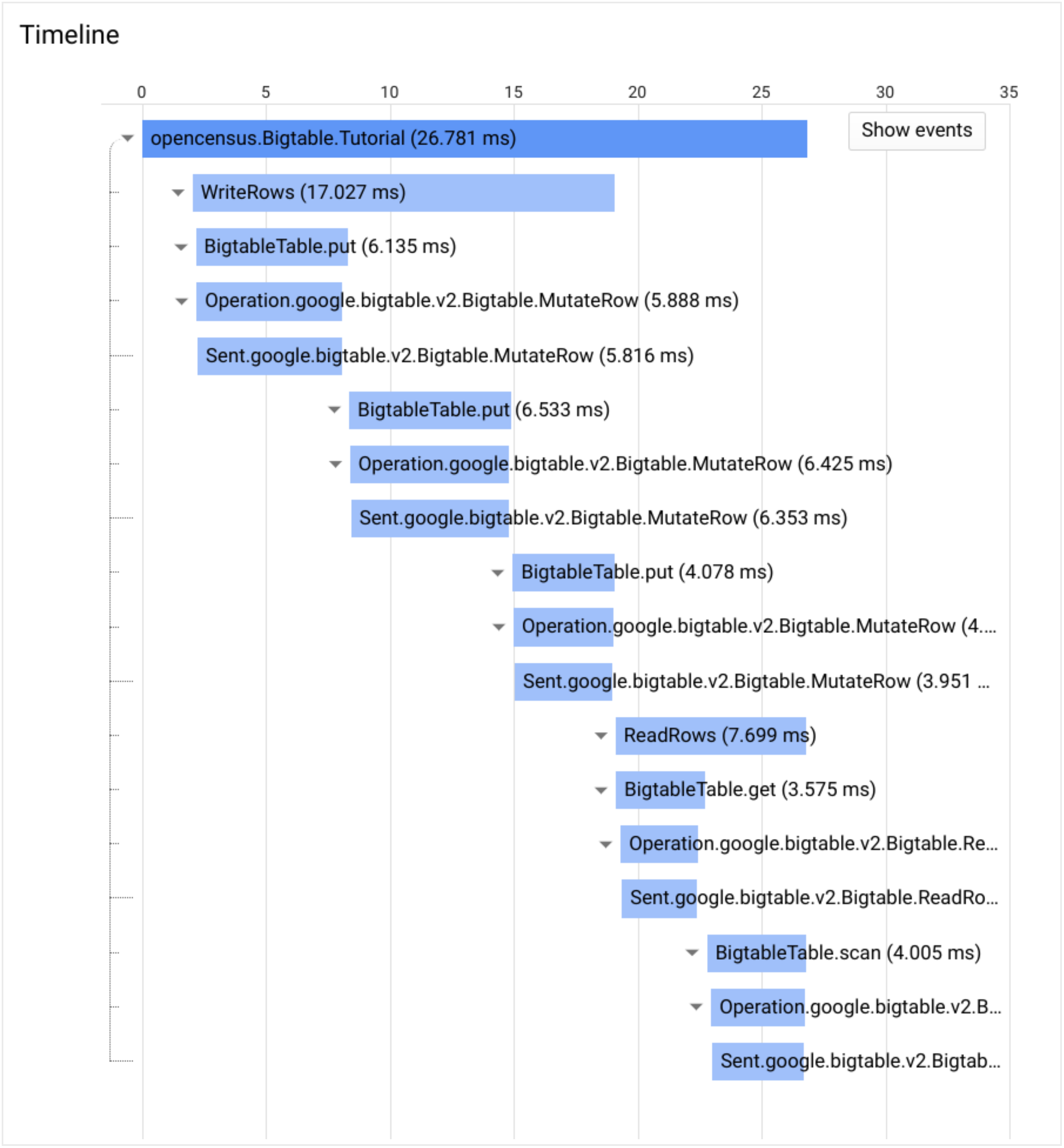
BLOB/MASTER/TUTORIALS/BIGTABLE-OC/JAVA/SRC/MAIN/JAVA/COM/EXAMPLE/BIGTABLE/APP.JAVA)

```
// sample every 1000 transactions
configureOpenCensusExporters(Samplers.probabilitySampler(1/1000.0));
```

3. Select **`opencensus.Bigtable.Tutorial`**. This opens a drill-down view that shows more information about the call chain, along with other useful information such as tracing scopes for discrete API calls instrumented in the client library and operation-level call latencies.

   For instance, each of the series of write and read rows are encapsulated by the lower-level, user-defined **WriteRows** and **ReadRows** tracing spans.

   Below **ReadRows**, you can see the `get` operation, followed by the table scan operations.



opencensus.Bigtable.Tutorial (26.781 ms)
WriteRows (17.027 ms)
BigtableTable.put (6.135 ms)
Operation.google.bigtable.v2.Bigtable.MutateRow (5.888 ms)
Sent.google.bigtable.v2.Bigtable.MutateRow (5.816 ms)
BigtableTable.put (6.533 ms)
Operation.google.bigtable.v2.Bigtable.MutateRow (6.425 ms)
Sent.google.bigtable.v2.Bigtable.MutateRow (6.353 ms)
BigtableTable.put (4.078 ms)
Operation.google.bigtable.v2.Bigtable.MutateRow (4....
Sent.google.bigtable.v2.Bigtable.MutateRow (3.951 ...
ReadRows (7.699 ms)
BigtableTable.get (3.575 ms)
Operation.google.bigtable.v2.Bigtable.Re...
Sent.google.bigtable.v2.Bigtable.ReadRo...
BigtableTable.scan (4.005 ms)
Operation.google.bigtable.v2.B...
Sent.google.bigtable.v2.Bigtab...

The other items included in the trace list, such as `Operation.google.bigtable.admin.v2.BigtableTableAdmin.CreateTable`, occurred outside of the manually defined tracing scope. As a result, these items are included as separate operations in the list.

Viewing metrics in Monitoring

The app code shows how to measure and record latency and transaction counts.

For metrics, there is no sampling. Every recorded value is included in the metric representations. Each metric is defined by the type of measurement to be performed. In this example, write latency is recorded with microsecond units:

tutorials/bigtable-oc/java/src/main/java/com/example/bigtable/App.java (https://github.com/GoogleCloudPlatform/community/blob/master/tutorials/bigtable-oc/java/src/main/java/com/example/bigtable/App.java)

BLOB/MASTER/TUTORIALS/BIGTABLE-OC/JAVA/SRC/MAIN/JAVA/COM/EXAMPLE/BIGTABLE/APP.JAVA)

```
// The write latency in milliseconds
private static final MeasureDouble M_WRITE_LATENCY_MS = MeasureDouble.create("btapp/
```

The distribution is aggregated and stored using these buckets: 0–5 ms, 5–10ms, 10–25 ms, and so on.

tutorials/bigtable-oc/java/src/main/java/com/example/bigtable/App.java (https://github.com/GoogleCloudPlatform/community/blob/master/tutorials/bigtable-oc/java/src/main/java/com/example/bigtable/App.java)

BLOB/MASTER/TUTORIALS/BIGTABLE-OC/JAVA/SRC/MAIN/JAVA/COM/EXAMPLE/BIGTABLE/APP.JAVA)

```
Aggregation latencyDistribution = Distribution.create(BucketBoundaries.create(
        Arrays.asList(
            0.0, 5.0, 10.0, 25.0, 100.0, 200.0, 400.0, 800.0, 10000.0)));
```

tutorials/bigtable-oc/java/src/main/java/com/example/bigtable/App.java (https://github.com/GoogleCloudPlatform/community/blob/master/tutorials/bigtable-oc/java/src/main/java/com/example/bigtable/App.java)

BLOB/MASTER/TUTORIALS/BIGTABLE-OC/JAVA/SRC/MAIN/JAVA/COM/EXAMPLE/BIGTABLE/APP.JAVA)

```
View.create(Name.create("btappmetrics/write_latency"),
            "The distribution of the write latencies",
            M_WRITE_LATENCY_MS,
            latencyDistribution,
            Collections.singletonList(KEY_LATENCY)),
```

All three metrics—write latency, read latency, and transaction counts—are recorded using a single call to the recorder:

tutorials/bigtable-oc/java/src/main/java/com/example/bigtable/App.java (https://github.com/GoogleCloudPlatform/community/blob/master/tutorials/bigtable-oc/java/src/main/java/com/example/bigtable/App.java)

BLOB/MASTER/TUTORIALS/BIGTABLE-OC/JAVA/SRC/MAIN/JAVA/COM/EXAMPLE/BIGTABLE/APP.JAVA)

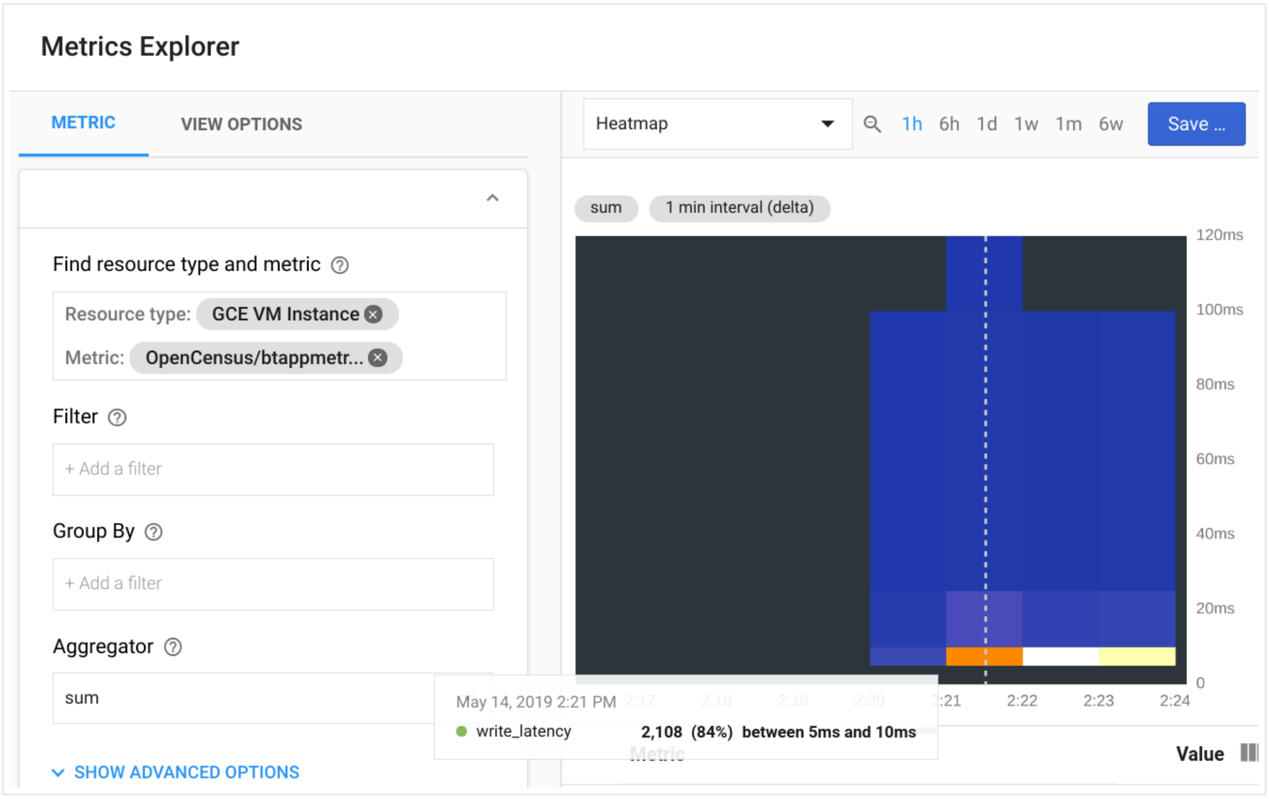
```
// record read, write latency metrics and count
STATS_RECORDER.newMeasureMap()
              .put(M_READ_LATENCY_MS, endRead - startRead)
              .put(M_WRITE_LATENCY_MS, endWrite - startWrite)
              .put(M_TRANSACTION_SETS, 1)
              .record();
```

You can review the captured metrics:

1. In the Google Cloud Console, go to **Monitoring** or use the following button:
   GO TO MONITORING (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/MONITORING)
2. Click **Dashboards**.
3. Click **Create dashboard**.
4. In the **Dashboard name** field, enter `Cloud Bigtable Metrics`.
5. Click **Add Chart**.
6. Ensure **Metric** is the selected tab.
7. Click in the box labeled **Find resource type and metric**, and enter `opencensus/btappmetrics/write_latency`.
8. In the drop-down menu, select **Heatmap**.The distribution Heatmap graph is shown
9. Click **Save Chart**.
10. Add charts to your dashboard for the following monitored resources: `opencensus/btappmetrics/read_latency` and

`opencensus/btappmetrics/transaction_set_count`.

11. Click **Dashboards/Cloud Bigtable Metrics**. The three metrics charts are displayed.
12. To zoom in, select a time range, click one of the charts, and drag the pointer to the edge of the graph. For the heatmaps, you can see more details about the distribution by holding the cursor over the various colored blocks.



Find resource type and metric

Resource type: GCE VM Instance

Metric: OpenCensus/btappmetr...

Filter: + Add a filter

Group By: + Add a filter

Aggregator: sum

May 14, 2019 2:21 PM

● write_latency 2,108 (84%) between 5ms and 10ms

Metrics Explorer shows a heatmap of the write latency. As the image shows, 2108 metrics samples fall into the 5–10 ms bucket.

Cleaning up

Caution: Deleting a project has the following effects:

- **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.
- **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an `appspot.com` URL, delete selected resources inside the project instead of deleting the whole project.

1. In the Cloud Console, go to the **Manage resources** page.

   GO TO THE MANAGE RESOURCES PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/IAM-ADMIN/PRO

2. In the project list, select the project you want to delete and click **Delete** .
3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

What's next

- Read about OpenCensus with Bigtable (https://opencensus.io/integrations/google_cloud/google_cloud_bigtable/) (Java and Go Tracing).
- Learn more about OpenCensus (https://github.com/census-instrumentation).
- Read about OpenCensus instrumentations: Metrics (https://opencensus.io/stats/) and Tracing (https://opencensus.io/tracing/).
- Learn more about Bigtable (https://cloud.google.com/bigtable/).
- Learn more about Stackdriver Trace (https://cloud.google.com/trace/).
- Try the OpenCensus with Memorystore (https://cloud.google.com/community/tutorials/memorystore-oc) tutorial.
- Try out other Google Cloud features for yourself. Have a look at our tutorials (https://cloud.google.com/docs/tutorials).

Last updated January 13, 2020.