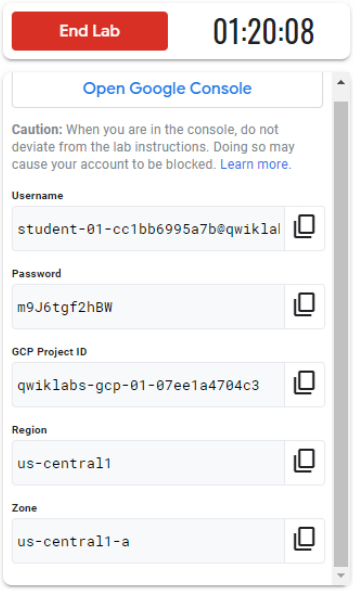


Site Reliability Troubleshooting with Stackdriver APM [ACE]

1 hour 30 minutes Free Rate Lab

Overview

The objective of this lab is to familiarize yourself with the specific capabilities of Cloud Monitoring to monitor GKE cluster infrastructure, Istio, and applications deployed on this infrastructure.

What you'll do

- Create a GKE cluster
- Deploy a microservices application to it
- Define latency and error SLIs and SLOs for it
- Configure Cloud Monitoring to monitor your SLIs
- Deploy a breaking change to the application and use Cloud Monitoring to troubleshoot and resolve the issues that result
- Validate that your resolution addresses the SLO violation

What you'll learn

- How to deploy a microservices application on an existing GKE cluster
- How to select appropriate SLIs/SLOs for an application
- How to implement SLIs using Cloud Monitoring features
- How to use Cloud Trace, Profiler, and Debugger to identify software issues

Prerequisites

- Google Cloud Platform account and project with billing account
- Basic knowledge of Kubernetes
- Basic knowledge of Cloud Monitoring
- Basic knowledge of troubleshooting process

Environment 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 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 the Google Cloud Platform for the duration of the lab.

What you need

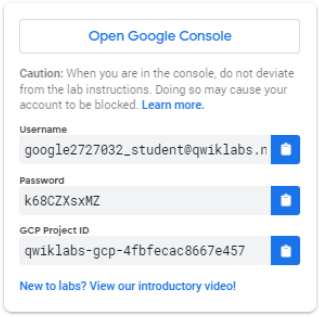
To complete this lab, you need:

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

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

How to start your lab and sign in to the Console

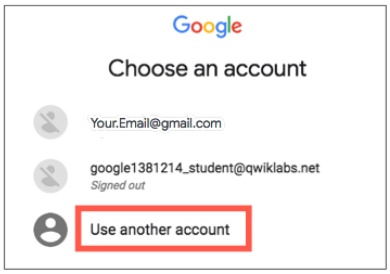
1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



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

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

3. On the Choose an account page, click **Use Another Account**.



4. The Sign in page opens. Paste the username that you copied from the Connection Details panel. Then copy and paste the password.

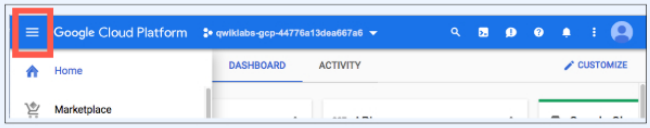
   Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own GCP account, do not use it for this lab (avoids incurring charges).

5. Click through the subsequent pages:

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

After a few moments, the GCP console opens in this tab.

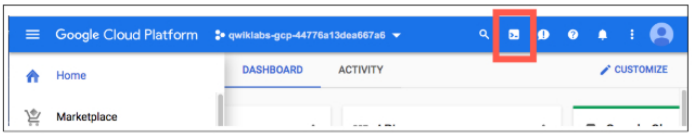
Note: You can view the menu with a list of GCP Products and Services by clicking the **Navigation menu** at the top-left, next to "Google Cloud Platform".



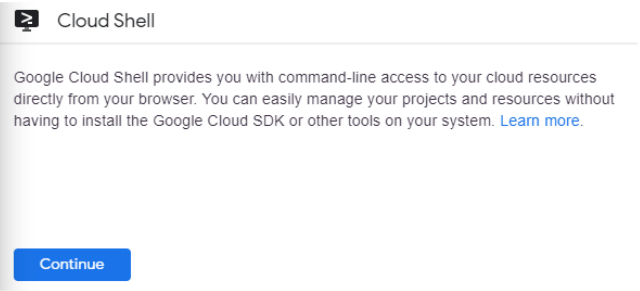
Activate Google Cloud Shell

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

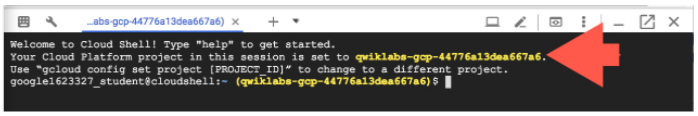
1. In GCP console, on the top right toolbar, click the Open Cloud Shell button.



2. 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 Platform. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

```
gcloud auth list
```

Output:

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

Example output:

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

You can list the project ID with this command:

```
gcloud config list project
```

Output:

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

Example output:

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

Full documentation of **gcloud** is available on Google Cloud gcloud Overview .

Infrastructure setup

In this lab you will connect to a Google Kubernetes Engine cluster and validate that it's been created correctly.

Set the zone in `gcloud`:

```
gcloud config set compute/zone us-west1-b
```

Set the project ID:

```
export PROJECT_ID=$(gcloud info --format='value(config.project)')
```

Verify that the cluster named `shop-cluster` has been created:

```
gcloud container clusters list
```

Your cluster status will say PROVISIONING. Wait a moment and run the command above again. Repeat until the status is RUNNING. This could take several minutes.

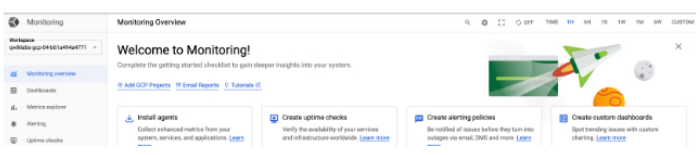
While you're waiting, set up your Monitoring workspace to monitor the application on your cluster.

Create a Monitoring workspace

You will now setup a Monitoring workspace that's tied to your Qwiklabs GCP Project. The following steps create a new account that has a free trial of Monitoring.

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

When the Monitoring dashboard opens, your workspace is ready.



Go back to your Cloud Shell and check again to see if the cluster has finished provisioning:

```
gcloud container clusters list
```

Once your cluster has RUNNING status, get the cluster credentials:

```
gcloud container clusters get-credentials shop-cluster --zone us-west1-b
```

Your output should look like this:

```
Fetching cluster endpoint and auth data.
kubeconfig entry generated for shop-cluster.
```

Verify that the nodes have been created:

```
kubectl get nodes
```

Your output should look like this:

```
NAME                                        STATUS    ROLES     AGE
VERSION
gke-shop-cluster-demo-default-pool1-24748028-3nwh   Ready     <none>
4m        v1.13.7-gke.8
gke-shop-cluster-demo-default-pool1-24748028-3z1g   Ready     <none>
4m        v1.13.7-gke.8
gke-shop-cluster-demo-default-pool1-24748028-4ksd   Ready     <none>
4m        v1.13.7-gke.8
gke-shop-cluster-demo-default-pool1-24748028-f2f2   Ready     <none>
4m        v1.13.7-gke.8
gke-shop-cluster-demo-default-pool1-24748028-gcb3   Ready     <none>
4m        v1.13.7-gke.8
```

Deploy application

In this module, you will deploy a microservices application called Hipster Shop to your cluster to create an actual workload you can monitor.

Run the following to clone the repo:

```
git clone -b APM-Troubleshooting-Demo-2
https://github.com/blipzimmerman/microservices-demo-1
```

Download and install `skaffold`:

```
curl -Lo skaffold
https://storage.googleapis.com/skaffold/releases/v0.36.0/skaffold-linux-
amd64 && chmod +x skaffold && sudo mv skaffold /usr/local/bin
```

Install the app using `skaffold`:

```
cd microservices-demo-1
skaffold run
```

Confirm everything is running correctly:

```
kubectl get pods
```

Your output should look like this:

```
NAME                                     READY     STATUS    RESTARTS
AGE
```

```
adservice-55f94cfd9c-41vml                1/1     Running   0
20m
cartservice-6f4946f9b8-6wtff              1/1     Running   2
20m
checkoutservice-5688779d8c-l6crl          1/1     Running   0
20m
currencyservice-665d6f4569-b4sbm          1/1     Running   0
20m
emailservice-684c89bcb8-h48sq             1/1     Running   0
20m
frontend-67c8475b7d-vktsn                 1/1     Running   0
20m
loadgenerator-6d646566db-p422w            1/1     Running   0
20m
paymentservice-858d89d64c-hmpkg           1/1     Running   0
20m
productcatalogservice-bcd85cb5-d6xp4      1/1     Running   0
20m
recommendationservice-685d7d6cd9-pxd9g    1/1     Running   0
20m
redis-cart-9b864d47f-c9xc6                1/1     Running   0
20m
shippingservice-5948f9fb5c-vndcp          1/1     Running   0
20m
```

Re-run the command until all pods are reporting a Running status before moving to the next step.

Get the **external IP** of the application:

```
export EXTERNAL_IP=$(kubectl get service frontend-external | awk 'BEGIN { cnt=0; } { cnt+=1; if (cnt > 1) print $4; }')
```

Finally, confirm that the app is up and running:

```
curl -o /dev/null -s -w "%{http_code}\n"  http://$EXTERNAL_IP
```

Note: You may need to run this command a second time if you get a 500 error.

Your confirmation will look like this:

```
200
```

Download the source and put the code in the Cloud Source Repo:

```
./setup_csr.sh
```

Now that the application has been deployed, set up monitoring for the application.

Resources

- Microservices Demo Application

NOTE: This lab uses a fork of this application build to aid in the troubleshooting exercises.

- Skaffold

Develop Sample SLOs and SLIs

Before implementing any monitoring, review the introduction to the chapter on `Service Level Objectives` from the SRE Book:

It's impossible to manage a service correctly, let alone well, without understanding which behaviors really matter for that service and how to measure and evaluate those behaviors. To this end, we would like to define and deliver a given level of service to our users, whether they use an internal API or a public product.

We use intuition, experience and an understanding of what users want to define **service level indicators (SLIs)**, **objectives (SLOs)** and **agreements (SLAs)**. These measurements describe basic properties of metrics that matter, what values we

measurements describe basic properties of metrics that matter, what values we want those metrics to have and how we'll react if we can't provide the expected service. Ultimately, choosing appropriate metrics helps to drive the right action if something goes wrong and also gives an SRE team confidence that a service is healthy.

An SLI is a service level indicator: A carefully defined quantitative measure of some aspect of the level of service that is provided.

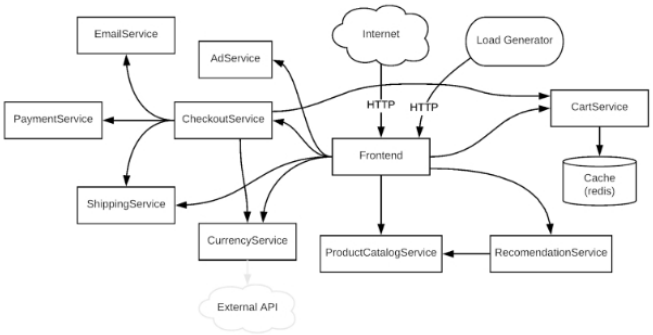
Most services consider

request latency: how long it takes to return a response to a request as a key SLI. Other common SLIs include the **error rate**, often expressed as a fraction of all requests received and **system throughput**, typically measured in requests per second. Another kind of SLI important to SREs is **availability** or the fraction of the time that a service is usable. It is often defined in terms of the fraction of well-formed requests that succeed.

Durability: the likelihood that data will be retained over a long period of time is equally important for data storage systems. The measurements are often aggregated: i.e., raw data is collected over a measurement window and then turned into a rate, average, or percentile.

Now that you have established a basic understanding, define the SLIs and SLOs for your application. Given that the application itself serves end user ecommerce traffic, it's going to be very important that user experience remains constant and that performance is good. You will monitor SLIs for request latency, error rate, throughput and availability.

Application Architecture



It's impossible to develop SLIs without understanding how the application is built. Details are in the original repository, but for this lab, it suffices to understand that:

- Users access the application through the Frontend.
- Purchases are handled by CheckoutService.
- CheckoutService depends on CurrencyService to handle conversions.
- Other services such as RecommendationService, ProductCatalogService and Adservice are used to provide the frontend with content needed to render the page.

Service Level Indicators and Objectives

The following SLIs and SLOs are selected based on the end-user experience and the theoretical impact to users and business objectives.

SLI	Metric	Description	SLO
Request latency	Front end latency	Measures how long a user is waiting for the page to load. A high latency typically correlates to a negative user experience	99% of requests from the previous 60 minute period are services in under 3 seconds
Error rate	Front end error rate	Measures the error rate experienced by users. A high error rate likely indicates an issue.	0 Errors in the previous 60 minute period

Error rate	Checkout error rate	Measures the error rate experienced by other services calling the checkout service. A high error rate likely indicates an issue.	0 Errors in the previous 60 minute period
Error rate	Currency Service error rate	Measures the error rate experienced by other services calling the currency service. A high error rate likely indicates an issue.	0 Errors in the previous 60 minute period
Availability	Front end success rate	Measures the rate of successful requests as a way to determine the availability of the service. A low success rate likely indicates that users are having a poor experience.	99% of requests are successful over the previous 60 minute period

Configure Latency SLI

Now that you have SLOs and SLIs defined, you can implement cloud monitoring. The metrics you are interested in are already being collected. You will create alerting policies for each of your SLOs.

Front End Latency

In the Monitoring tab, click **Alerting** and select **Create Policy**.

Name the policy **Latency Policy**.

Click **Add Condition** and specify the metric and condition that will be used to trigger the Alerting Policy. The condition will let you know when you're experiencing performance issues that are impacting user experience.

As described in the *Service Level Indicators and Objectives* table above, you will use the 99th percentile front end latency as the SLI.

Add the following into the **Find resource type and metric** field then select the following from the dropdown menu:

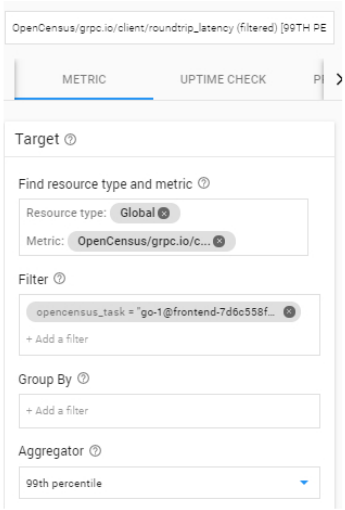
```
custom.googleapis.com/opencensus/grpc.io/client/roundtrip_latency
```

Note: You may need to refresh the page to see the above metric.

In the `Resource Type`, select the **Global** option.

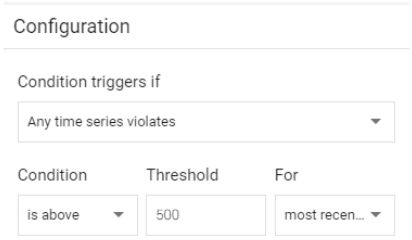
Click into the **Filter** field and select the **opencensus_task** item. Click on the first default Value, then click **Apply**.

Next, set the `Aggregator` to **99th percentile**.



Next, in the **Configuration** area, set the options as follows:

- Condition triggers if **Any time series violates**
- Condition: **is above**
- Threshold: **500**
- For: **Most recent value**



Click **Add**.

Click **Save**. You've configured Cloud Monitoring to monitor your frontend latency SLI!

Configure Availability SLI

Next, configure Cloud Monitoring to monitor service availability by creating another Alerting Policy.

Front End Availability

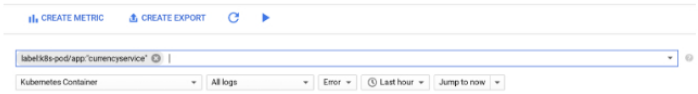
Start by monitoring the error rate for the front end service, since that's where user experience is going to be most directly impacted. As discussed above, you're going to consider any failures observed to be an SLO violation. Create an alerting policy that will trigger an incident if any failures are observed.

An easy way to trigger on a particular failure is to use log-based metrics.

In the Google Cloud Platform Console, click **Logging**.

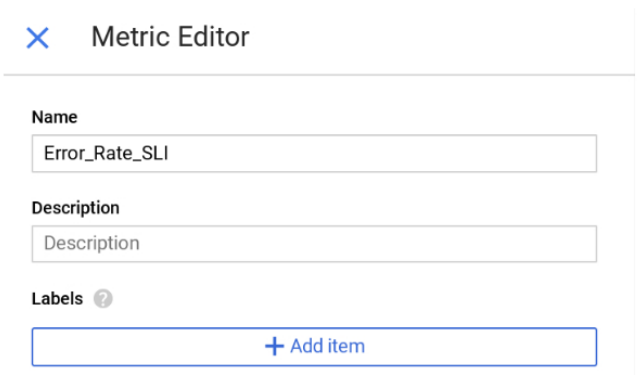
Configure the filter as follows:

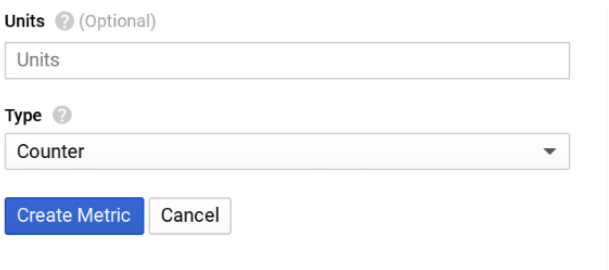
- In Resource type select **Kubernetes Container**.
- In Any Log Level select **ERROR**.
- In the filter bar add: `label:k8s-pod/app:"currencyservice"`



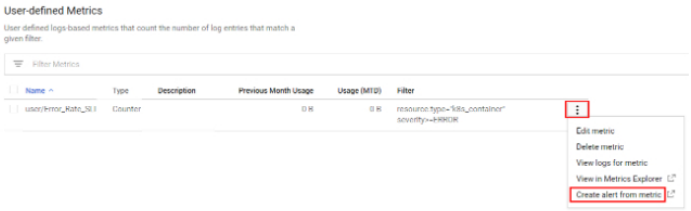
Click **Create Metric**.

Name the metric `Error_Rate_SLI` and click **Create Metric** to save the log based metric:





You now see the metric listed on the Logs-based metrics page. To create an alert for this metric, click the 3 dots at the end of the row and select **Create Alert from Metric**.



Notice the resource type and metric have already been filled in.

Name the condition "Error Rate SLI".

Click the **Show Advanced Options** link and set the following:

- Aligner: **rate**

In Configuration, use **0.5** as your Threshold for **2 minutes**.

Then **Save** the condition.

In the subsequent screen, name your new policy "Error Rate SLI", and **Save** it.

As expected, there are no failures, and your application is currently meeting its availability SLO!

Deploy new release

Now that you have configured SLI monitoring, you're ready to measure the impact of application changes on user experience. See what happens when you deploy a new release of the application.

Next you'll modify the Kubernetes manifests for the services which have new releases and then run skaffold to deploy the application again.

Update YAML files

Open the Code Editor in Cloud Shell:

Find the **microservices-demo-1** folder and open the **kubernetes-manifests** folder within it.

Update the kubernetes manifests to pull the new images by:

1. Replacing the **accl-demo** image tag with **rel013019**
2. Adding: `imagePullPolicy: Always`

in the following files:

- `kubernetes_manifests/recommendationservice.yaml`
- `kubernetes_manifests/currencyservice.yaml`
- `kubernetes_manifests/frontend.yaml`

As an example, here's the original version of the `recommendationservice.yaml` file:

And here it is after making the update:

Once you've updated and closed each of the files, you're ready to deploy the new version!

Deploy New Version

In Cloud Shell, update the deployment to deploy the new container image:

```
skaffold run
```

Validate that there are new versions of services running:

```
kubectl get pods
```

Your output should look like this:

```
NAME                                     READY     STATUS    RESTARTS
AGE
adservice-55f94cfd9c-4lvml               1/1       Running   0
17d
cartservice-6f4946f9b8-6wtff             1/1       Running   197
17d
checkoutservice-5688779d8c-l6crl         1/1       Running   0
17d
currencyservice-665d6f4569-b4sbm         1/1       Running   0
1m
emailservice-684c89bcb8-h48sq            1/1       Running   0
17d
frontend-5f889fc7bb-wvfvv                1/1       Running   0
1m
loadgenerator-6d646566db-p422w           1/1       Running   0
17d
paymentservice-858d89d64c-hmpkg          1/1       Running   0
17d
productcatalogservice-bcd85cb5-d6xp4     1/1       Running   0
17d
recommendationservice-57cb4559f9-bdgj7   1/1       Running   0
1m
redis-cart-9b864d47f-c9xc6               1/1       Running   0
17d
shippingservice-5948f9fb5c-vndcp         1/1       Running   0
17d
```

Send some data

Now that the application is running, go look at what you have deployed.

In the Console, navigate to **Kubernetes Engine** > **Services & Ingress**. Look for the

`frontend-external` service and click on the Endpoint URL.

Once on the Hipster Shop website, click on a **Buy** and/or **Add to Cart** for a couple of items to send some traffic.

Latency SLO Violation - Find the Problem

In this exercise you will use Cloud Application Performance Management tools (APM) to identify and resolve an issue causing poor application latency.

First see if everything is still OK with the application after deploying the new version.

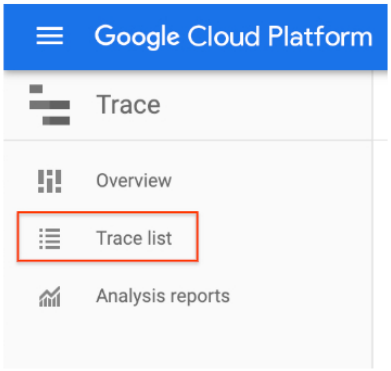
Go to **Monitoring Overview** page. Click the **autorefresh arrows** in the top ribbon so you will always be looking at the latest information.

Click **Alerting** and select **Latency Policy** under Policies. You can see that your latency is significantly exceeding the threshold that was set up.

First, click **Acknowledged** so that further notification escalation does not take place.

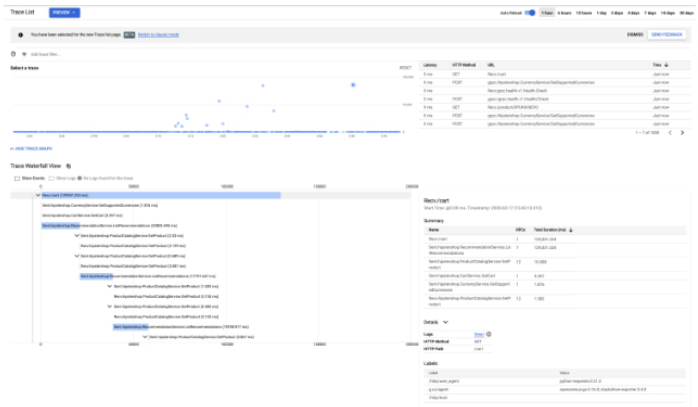
The best way to analyze latency issues is by using Trace. In the Google Cloud Platform Console, click **Trace**.

You're now in the Console. The initial overview is useful, but you need to get to the next level of detail. Open the **Trace List** page.



Click **Auto Reload**. Notice the scatter plot at the top of the page and that, around the time of the alert, there are a large number of outlier requests.

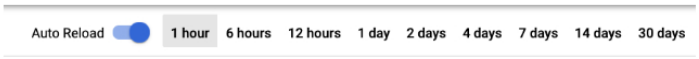
Wait a minute or two, to gather data, then click on one of the outlier traces to see the specifics about what is going on.



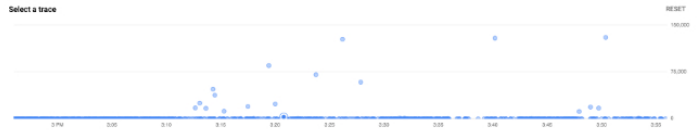
Notice the Span name (which represents the service or function that is being called) is either `/cart/` or `/cart/checkout/`.

To help understand how this trace compares with similar ones prior to the issue, enter `Recv./cart` in the **Add trace filter** to filter for all the cart operations and look for similar traces.

Set the time period to **1 hour** so that it includes traces that occurred before the issue.



Click on an example trace from before the issue occurred



Notice that in this similar trace ListRecommendations was only called once. However, after the most recent deploy, ListRecommendations is being called many times per request, causing significant additional latency.

You can conclude that the issue with these outliers is caused by multiple calls to ListRecommendations.

Deploy Change to Address Latency

In order to address the latency issue that the last release created, you need to roll out another version that fixes the broken code. You will next modify the Kubernetes manifests for the services that contained the broken code.

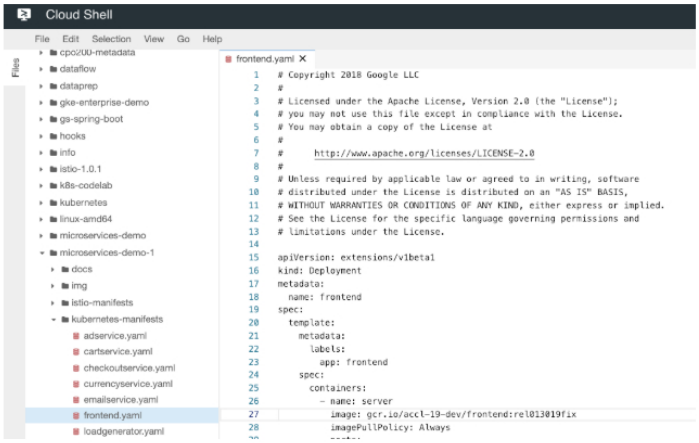
To deploy a fix return to Cloud Shell and open the **Source Code Editor**. You'll be modifying the following files:

- `kubernetes_manifests/recommendationservice.yaml`
- `kubernetes_manifests/frontend.yaml`

Modify the image tag **rel013019** with **rel013019fix** so the `image` should look like this:

```
      containers:
      - name: server
```

```
        image: gcr.io/accl-19-dev/frontend:rel013019fix
        imagePullPolicy: Always
```



Save the files.

Return to the Cloud Shell prompt and redeploy the images with the fixes in them:

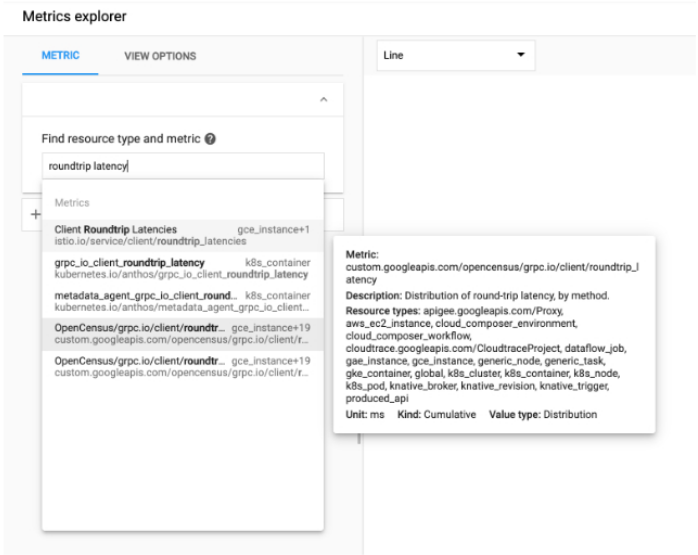
```
skaffold run
```

Validate Fix

Now that you've rolled back the breaking change, verify that your application is back to a healthy state.

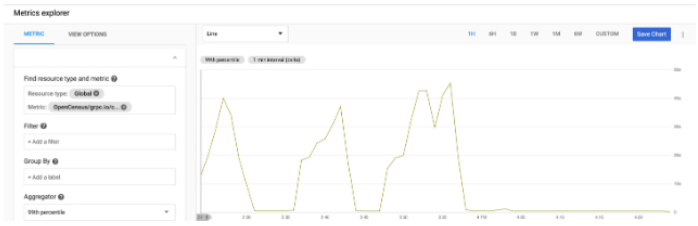
Return to **Monitoring** and click **Metrics Explorer**.

In the search field, enter **roundtrip latency** and select `custom.googleapis.com/opencensus/grpc.io/client/roundtrip_latency`.



Select **Global** as a resource type. Click **here** link in the **Aggregator** field.

Change the chart type to **Line**. You should see a chart that shows an immediate decrease in latency (and if you don't, wait a minute).



Now, see if the incidents are resolved. Return to the **Monitoring Overview**.

You should notice two things - you no longer have an incident and there are events letting you know that the incident has been resolved. Again, if you don't see an Incident resolved message, wait a couple of minutes.

Your monitoring was able to correctly identify a change that caused user experience (as measured by latency) to degrade, you were able to identify the root cause and you've rolled back the breaking change! In the next section, see how Monitoring can help you resolve an issue with availability.

Error Rate SLO Violation - Find the Problem

In this exercise you will use Cloud Monitoring Application Performance Management tools (APM) to troubleshoot an issue causing ERRORs in your application violating your error budget.

Click **Alerting** in Monitoring.

Look for a Error Rate SLI incident and click the **incidents** to learn more about what's going on. Incidents can take several minutes to be confirmed and listed as an incident. If an incident has not yet arrived, you can skip the Incident step below click on **Error Reporting** in the GCP console.

You can see that the pod is logging significantly more errors than it was previously.

Acknowledged the incident so that no further notification escalation takes place.

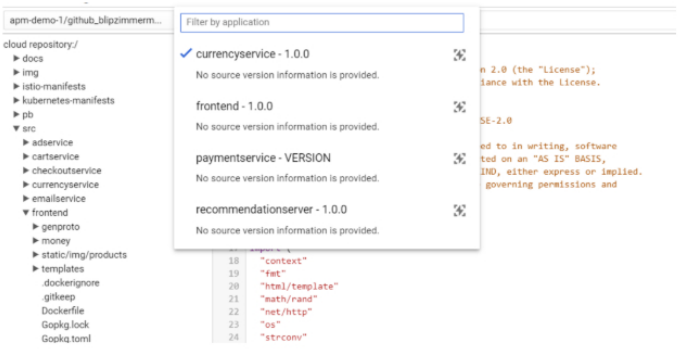
For an alert like this there are many places to start, but the easiest is Error Reporting.

Notice the Open Error Reporting incidents with a recent spike in occurrences. Click on the **Error: Conversion is Zero** to learn more about the error in question.

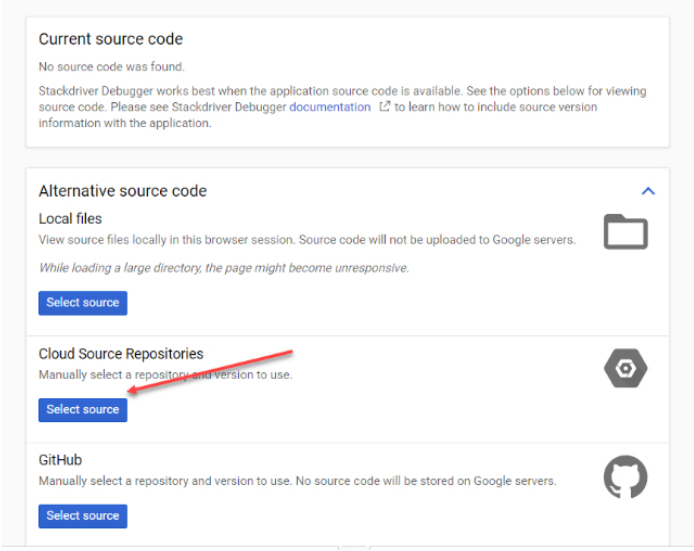
Look at the `Stack Trace sample` on the right. Here you can see what specific calls were related to the error.

Click on the lowest call showing here: **/usr/src/app/server/js:131**

This will load you into Debugger. On the top bar click the **Service** and select the **currencyservice**.



Next, select the source code that is running from **Cloud Source Repositories**.

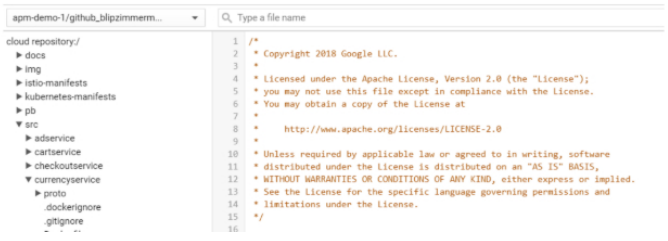


Select your source with:

- **Repository:** `apm-qwiklabs-demo`
- **Tagged version or branch:** `APM-Troubleshooting-Demo-2`

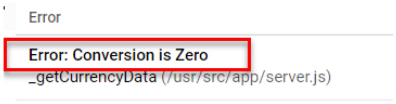
Then click **Select Source**.

In the left hand menu browse to `/src/currencyservice/server.js`.



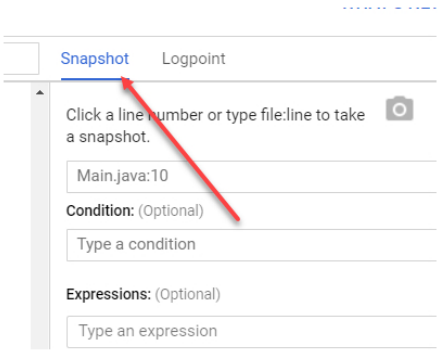
Scroll down to around line 155 which is the function where the exception was thrown. You can see the logline **Conversion is Zero** that was referenced in error reporting.

```
152       if (result.units > 0) {
153         logger.info(`conversion request successful`);
154       } else {
155         var stack = new Error("Conversion is Zero").stack;
156         logger.error(stack);
157       }
158       callback(null, result);
```



From the above code snippet you see this error is logged when `result.units < 0`. To troubleshoot this issue you'll use Snapshots to inspect the variables as the application progresses.

Make sure you have selected **Snapshot** in the top right:

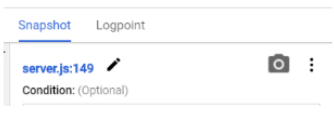


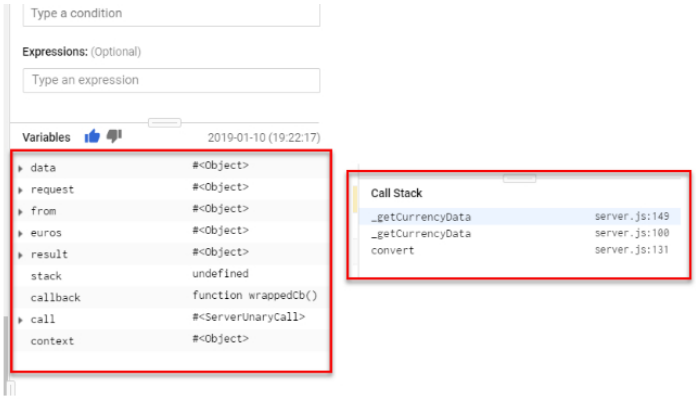
Then click on the line number (155) you want to snapshot:

```
150       result.nanos = Math.floor(result.nanos);
151       result.currency_code = request.to_code;
152       if (result.units > 0) {
153         logger.info(`conversion request successful`);
154       } else {
155         var stack = new Error('Conversion is Zero').stack;
156         logger.error(stack);
157       }
158       callback(null, result);
159     });
160   } catch (err) {
161     logger.error(`conversion request failed: ${err}`);
162     callback(err.message);
163   }
164 }
```

For this exercise take snapshots at line 155, 141 and 149. Add additional snapshot points wherever you feel appropriate. The system will take a variable snapshot the next time that code is executed. While the application is waiting for the code to be executed next you can see a "Waiting for snapshot to hit...." notice.

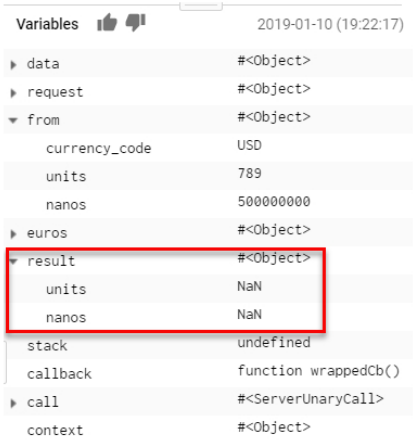
When the snapshot is complete the right hand pane will display the variables for that given snapshot.





Notice the Variable and Call Stack information. This information can provide extremely deep understanding of the path you code is taking including the variables and structures that exist as it takes that path, all without restarting the application or changing any code.

Click **result** to inspect all 3 snapshots finishing on line 155. Remember the error is triggered when `result.units` is NOT > 0. Inspecting the variables you can see that `result.units = NaN` (meaning 'not a number'). This is the issue that is causing the error.

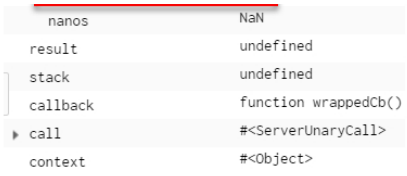


At this point you can conclude that the error is caused by a bug in the convert (or child) functions which sets the `result.units` to `0` resulting in a 0.00 price tag for the item being converted. Your troubleshooting along with the snapshot information and logs is a solid diagnosis of the issue.

So what is the bug that has caused this problem? From the code, `result.units` is set by the line 144 from euros, which was set in line 136 by operating on `from.units`

```
// Convert: from_currency --> EUR
const from = request.from;
const euros = _carry({
  units: from.units / data[from.units],
  nanos: from.nanos / data[from.currency_code]
});
```

Inspecting the snapshots `euros.units` is also `NaN`, however, `from.units` is a valid number. Thus the issue happened when converting `from.units` to euros.



You can conclude that the root cause is a bug in how `from.units` is converted into `euros.units` on line 137 and 8 is passed into `Data[]` which is actually a key value mapping of currency units (like EUR) into exchange rates. The corrected line 137 would use `from.to_currency` (aka USD) instead of `from.units` (aka 8).

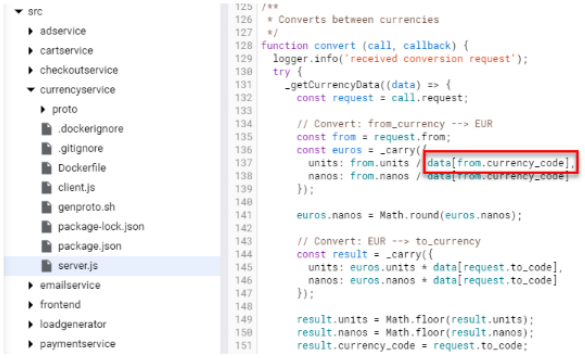
At this point you have determined the cause of the bug and can make the appropriate change. Based on the timing of the alert, this could have been caused by the latest deployment.

See if the previous branch **Master** had this code error on line 137.

Go back to the Console and inspect the code using Cloud **Source Repositories** (in the console menu under Tools).

Open the **apm-qwiklabs-demo** repository and select the **master** branch.

Browse on the left hand side to **src** > **currencyservice** > **server.js**. Notice on line 137 the proper dividend: data[from.currency_code] is used.



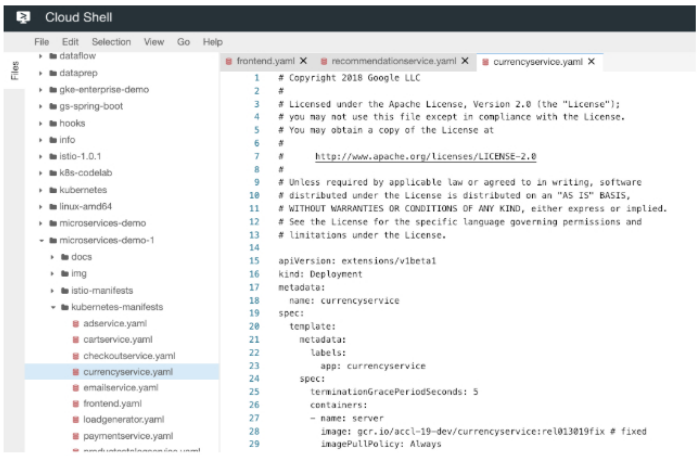
You have confirmation this bug was introduced in the latest push. To mitigate this problem you need to roll back to the previous version.

Deploy Change to Address Error Rate

In order to fix this issue, you'll need to deploy a fix to your application. To do that, you'll need to modify the Kubernetes manifest for the service that contained the broken code.

Deploy Fix

Return to **Cloud Shell** and in the **Source Code Editor** open the **currencyservice.yaml** file in the **kubernetes_manifests** folder.



Replace the image tag **rel013019** with **rel013019fix** so the `image` should look like this:

```
containers:
- name: server
  image: gcr.io/accl-19-dev/frontend:rel013019fix
  imagePullPolicy: Always
```

Close the file to save it and **return** to the Cloud Shell prompt.

Redeploy the image with the fix in it:

```
skaffold run
```

Validate Fix

Now that you've rolled back the breaking change, verify that the application is back to a healthy state.

As before, start by verifying that your incident is resolved. Go to the **Alerting** in the Monitoring UI and verify that the error rate incident is resolved.

Next, return to **Error Reporting**. Open the error previously observed and verify that it is no longer occurring (the timeline should show no further occurrences since the last deployment):

Congratulations - your monitoring was able to correctly identify a change that caused user experience (as measured by application errors) to degrade, you were able to identify the root cause and you've rolled back the breaking change!

Move on to the next section to learn about how you can optimize resource utilization using Cloud Monitoring.

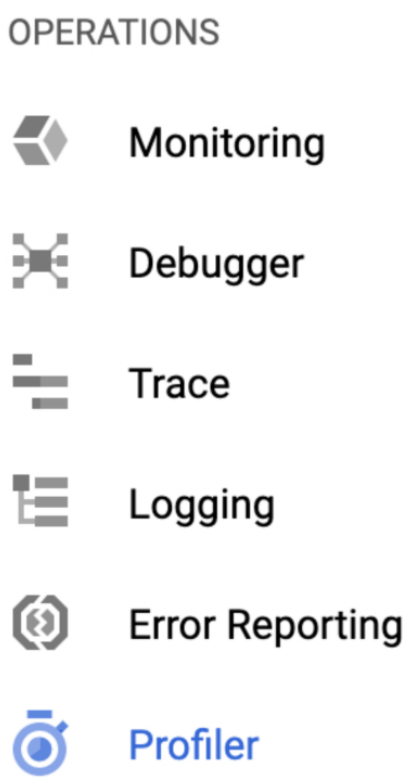
Application optimization with Cloud Monitoring APM

In this exercise use Cloud Monitoring Application Performance Management tools (APM) to identify opportunities for improvement that will help your application run faster and use less compute resources.

In this scenario, the Director of Cloud Operations is disappointed with the recent rise in compute costs. Specifically, the **currencyservice** service is using more CPU than expected based on the usage of the system.

Your team has been tasked with finding optimization opportunities. APM tools will be used to analyze the service and ensure your team's efforts are focused on the right areas of the application.

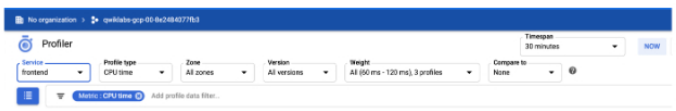
From the Console, open **Profiler** from the left hand menu.



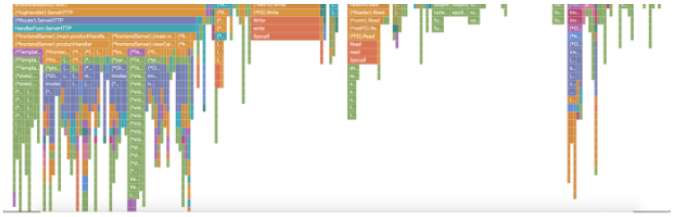
Change the Timespan in the upper right to 30 minutes. If there is no data, wait a minute of 2 for the data to populate.

NOTE: Profiler takes a random sample of calls to build an aggregate call stack. If you don't see the data you expect, it's because not enough time has elapsed during this lab and you completed it faster than expected. Feel free to use the screenshots below during the exercise.

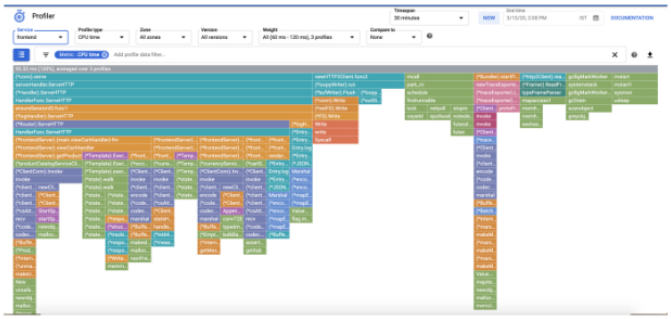
In the filter options select the **frontend** service, the **CPU time** Profile type.



Profiler takes random sample profiles of the system and combines the data to show you what functions are using the most resources. The flame graph below shows the function calls grouped by their use of the resource (in this case CPU) where the X-axis is the amount of CPU and the Y-axis shows parent child relationships.



In this case the majority of the CPU is used by the ServeHTTP call on the left hand side. Click on this call to drill into the cause.



The expanded view shows almost half of this is caused by **viewCartHandler**, which in turn is mostly caused by **getRecommendations**.

The opportunity here is in the **getRecommendations** and in turn **getProduct**. Thinking back to your earlier exercise, remember that the recommendation service and getProduct were being called often in a loop due to an error in retry logic. The resolution for that issue will likely decrease compute cost by as much as 20%.

End your lab

When you have completed your lab, click **End Lab**. Qwiklabs removes the resources you've used and cleans the account for you.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.

The number of stars indicates the following:

- 1 star = Very dissatisfied
- 2 stars = Dissatisfied
- 3 stars = Neutral
- 4 stars = Satisfied
- 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, please use the **Support** tab.