**Debugging Apps on Google Kubernetes Engine**

experimentLabschedule1 hour 15 minutesuniversal\_currency\_alt5 Creditsshow\_chartIntermediate

**GSP736**



**Overview**

Cloud Logging, and its companion tool, Cloud Monitoring, are full featured products that are both deeply integrated into Google Kubernetes Engine. This lab teaches you how Cloud Logging works with GKE clusters and applications and some best practices for log collection through common logging use cases.

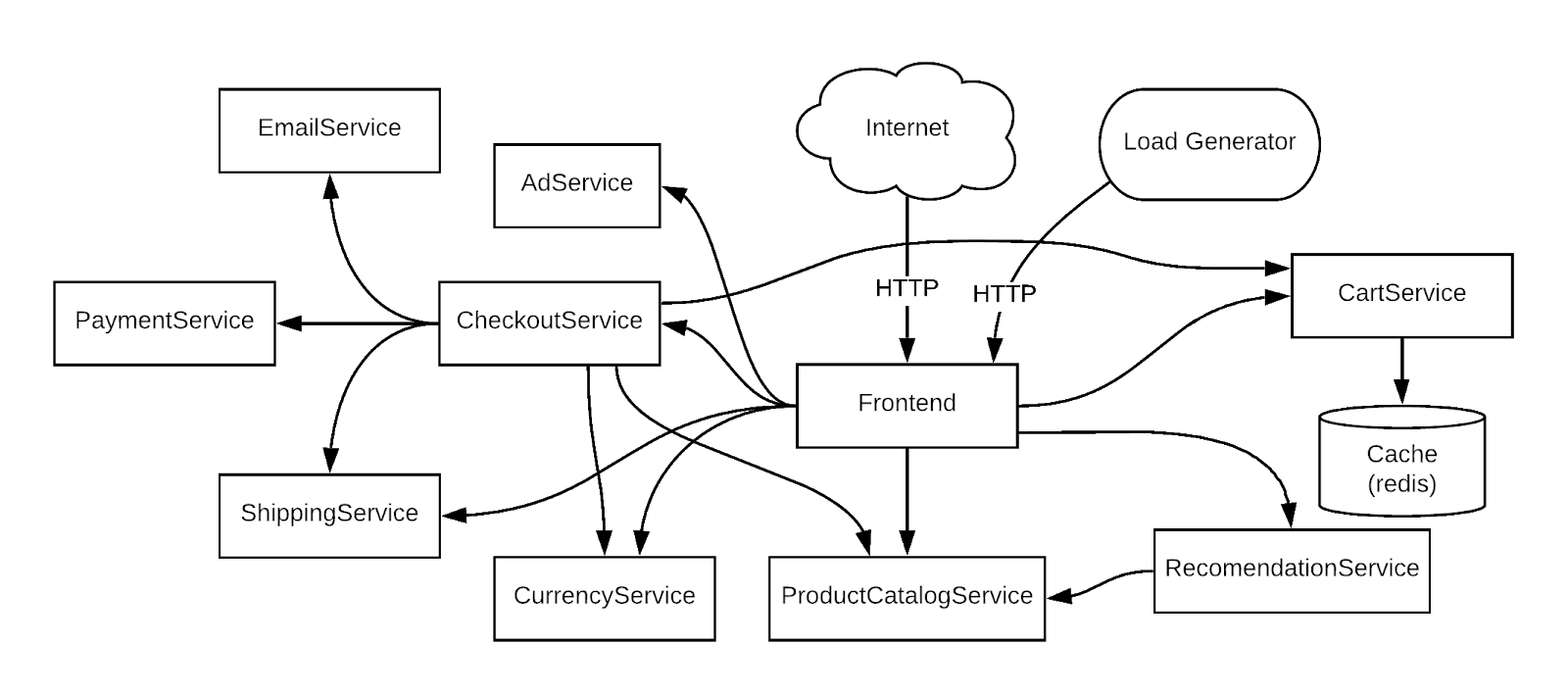
Objectives

In this lab, you will learn how to:

* Use Cloud Monitoring to detect issues
* Use Cloud Logging to troubleshoot an application running on GKE

The demo application used in the lab

To use a concrete example, you will troubleshoot a sample [microservices demo](https://github.com/GoogleCloudPlatform/microservices-demo" \t "_blank) app deployed to a GKE cluster. In this demo app, there are many microservices and dependencies among them. You will generate traffic using a loadgenerator and then use Logging, Monitoring and GKE to notice the error (alert/metrics), identify a root cause with Logging and then fix/confirm the issue is fixed with Logging and Monitoring.



**Setup and requirements**

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

To complete this lab, you need:

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

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

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

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

How to start your lab and sign in to the Google Cloud console

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is the **Lab Details** panel with the following:
   * The **Open Google Cloud console** button
   * Time remaining
   * The temporary credentials that you must use for this lab
   * Other information, if needed, to step through this lab
2. Click **Open Google Cloud console** (or right-click and select **Open Link in Incognito Window** if you are running the Chrome browser).

The lab spins up resources, and then opens another tab that shows the **Sign in** page.

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

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

1. If necessary, copy the **Username** below and paste it into the **Sign in** dialog.

student-04-5c324626efaf@qwiklabs.net

Copied!

content\_copy

You can also find the **Username** in the **Lab Details** panel.

1. Click **Next**.
2. Copy the **Password** below and paste it into the **Welcome** dialog.

YHDOvh3Mxvdv

Copied!

content\_copy

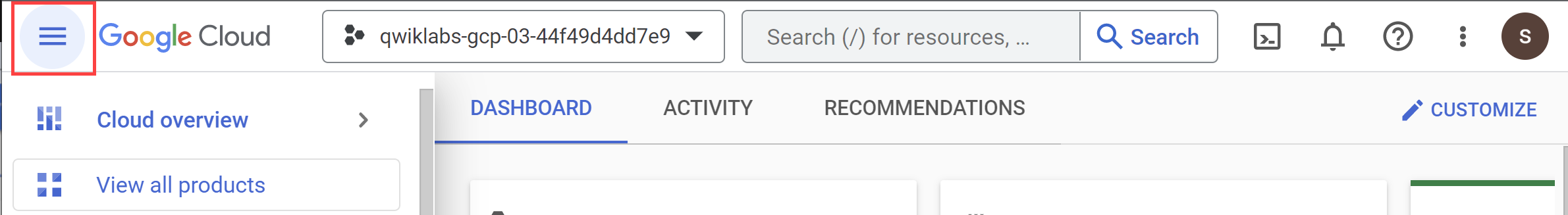
You can also find the **Password** in the **Lab Details** panel.

1. Click **Next**.

**Important:**You must use the credentials the lab provides you. Do not use your Google Cloud account credentials.**Note:**Using your own Google Cloud account for this lab may incur extra 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 Google Cloud console opens in this tab.

**Note:** To view a menu with a list of Google Cloud products and services, click the **Navigation menu** at the top-left. 

Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

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

When you are connected, you are already authenticated, and the project is set to your **Project\_ID**, qwiklabs-gcp-00-740451060ed7. The output contains a line that declares the **Project\_ID** for this session:

Your Cloud Platform project in this session is set to qwiklabs-gcp-00-740451060ed7

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

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

gcloud auth list

Copied!

content\_copy

1. Click **Authorize**.

**Output:**

ACTIVE: \*

ACCOUNT: student-04-5c324626efaf@qwiklabs.net

To set the active account, run:

$ gcloud config set account `ACCOUNT`

1. (Optional) You can list the project ID with this command:

gcloud config list project

Copied!

content\_copy

**Output:**

[core]

project = qwiklabs-gcp-00-740451060ed7

**Note:**For full documentation of gcloud, in Google Cloud, refer to [the gcloud CLI overview guide](https://cloud.google.com/sdk/gcloud).

Set your region and zone

Certain Compute Engine resources live in regions and zones. A region is a specific geographical location where you can run your resources. Each region has one or more zones.

Learn more about regions and zones and see a complete list in [Regions & Zones documentation](https://cloud.google.com/compute/docs/regions-zones/).

Run the following gcloud commands in Cloud Console to set the default region and zone for your lab:

gcloud config set compute/zone "europe-west1-d"

export ZONE=$(gcloud config get compute/zone)

gcloud config set compute/region "europe-west1"

export REGION=$(gcloud config get compute/region)

Copied!

content\_copy

**Task 1. Infrastructure setup**

Connect to a Google Kubernetes Engine cluster and validate that it's been created correctly.

1. Set the Project ID variable:

export PROJECT\_ID=qwiklabs-gcp-00-740451060ed7

Copied!

content\_copy

1. Use the following command to see the cluster's status:

gcloud container clusters list

Copied!

content\_copy

The cluster status will say PROVISIONING.

1. Wait a moment and run the above command again until the status is RUNNING. This could take several minutes.
2. Verify that the cluster named central has been created.

You can also monitor the progress in the Cloud Console by navigating to **Navigation menu** > **Kubernetes Engine** > **Clusters**.

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

gcloud container clusters get-credentials central --zone $ZONE

Copied!

content\_copy

**Output:**

Fetching cluster endpoint and auth data.

kubeconfig entry generated for central.

1. Verify that the nodes have been created:

kubectl get nodes

Copied!

content\_copy

Your output should look like this:

NAME STATUS ROLES AGE VERSION

gke-central-default-pool-5ff4130f-qz8v Ready 24d v1.27.2-gke.1200

gke-central-default--pool-5ff4130f-ssd2 Ready 24d v1.27.2-gke.1200

gke-central-default--pool-5ff4130f-tz63 Ready 24d v1.27.2-gke.1200

gke-central-default--pool-5ff4130f-zfmn Ready 24d v1.27.2-gke.1200

**Task 2. Deploy application**

Next, deploy a microservices application called Hipster Shop to your cluster to create a workload you can monitor.

1. Run the following to clone the repo:

git clone https://github.com/xiangshen-dk/microservices-demo.git

Copied!

content\_copy

1. Change to the microservices-demo directory:

cd microservices-demo

Copied!

content\_copy

1. Install the app using kubectl:

kubectl apply -f release/kubernetes-manifests.yaml

Copied!

content\_copy

1. Confirm everything is running correctly:

kubectl get pods

Copied!

content\_copy

The output should look similar to the output below:

NAME READY STATUS RESTARTS AGE

adservice-55f94cfd9c-4lvml 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

1. Rerun the command until all pods are reporting a **Running** status before moving to the next step.

Click *Check my progress* to verify the objective.

Deploy Application

Check my progress

1. Run the following to get the **external IP** of the application. This command will only return an IP address once the service has been deployed, so you may need to repeat the command until there's an external IP address assigned:

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

Copied!

content\_copy

1. Finally, confirm that the app is up and running:

curl -o /dev/null -s -w "%{http\_code}\n" http://$EXTERNAL\_IP

Copied!

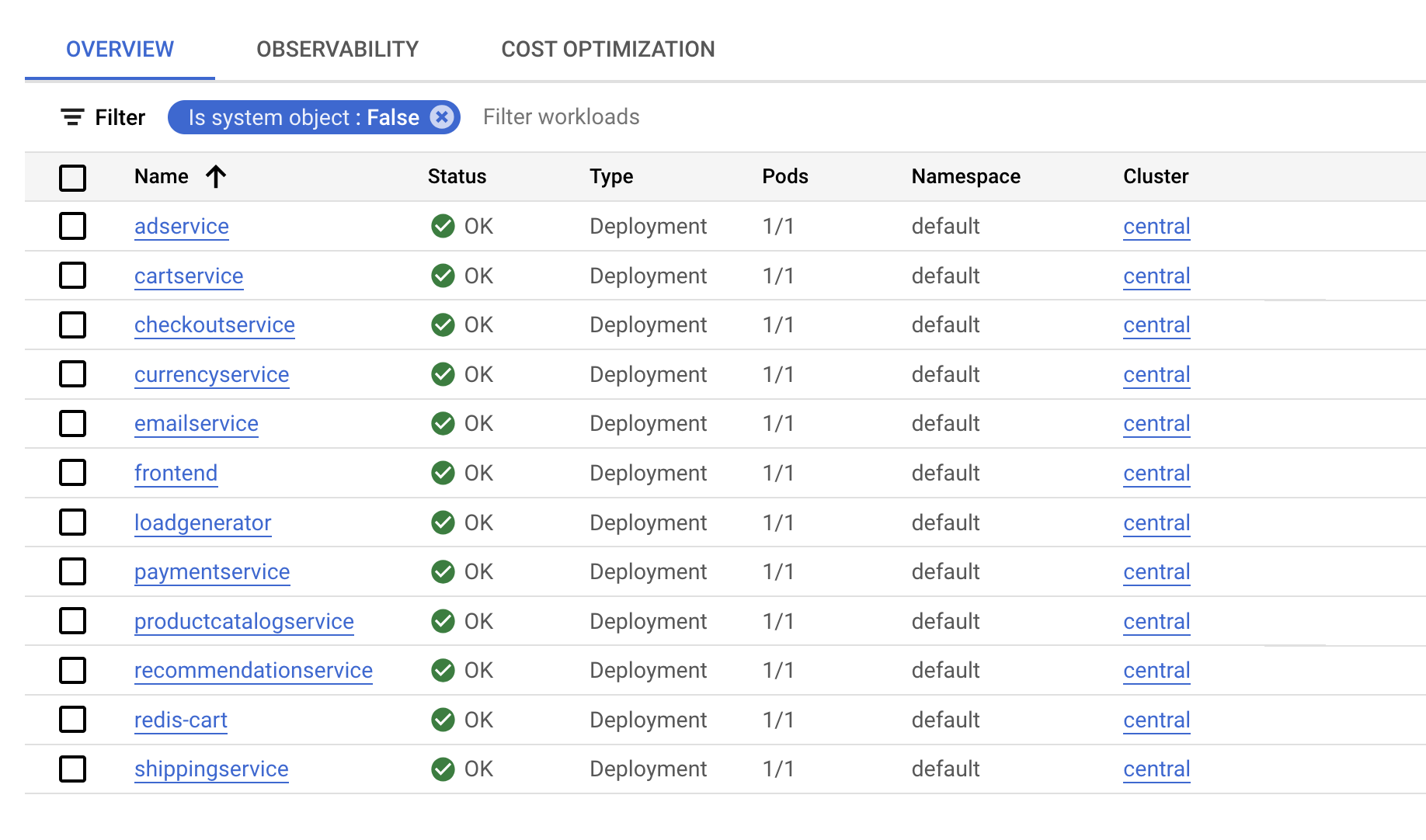
content\_copy

Your confirmation will look like this:

200

After the application is deployed, you can also go to the Cloud Console and view the status.

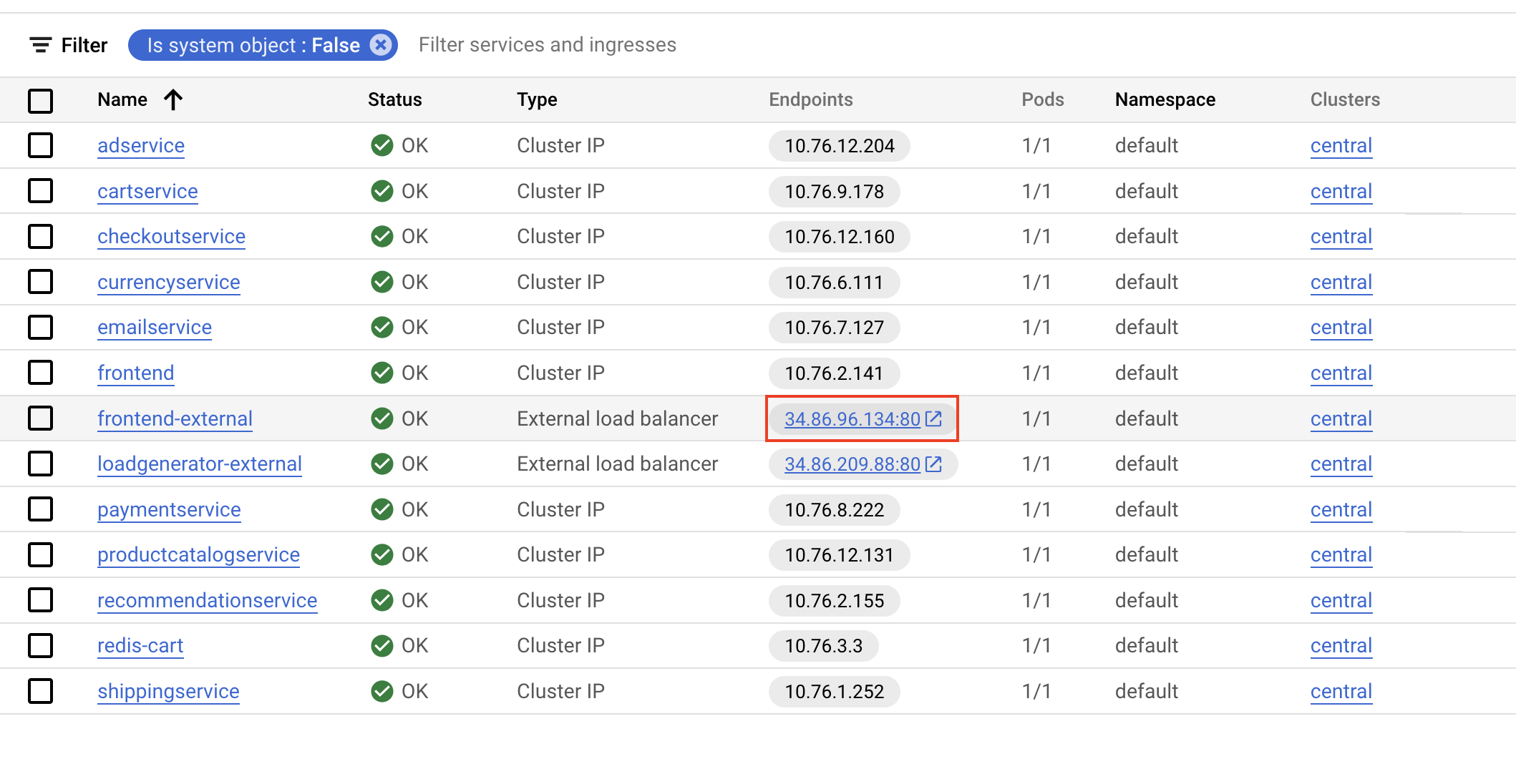
In the **Kubernetes Engine > Workloads** page you'll see that all of the pods are OK.



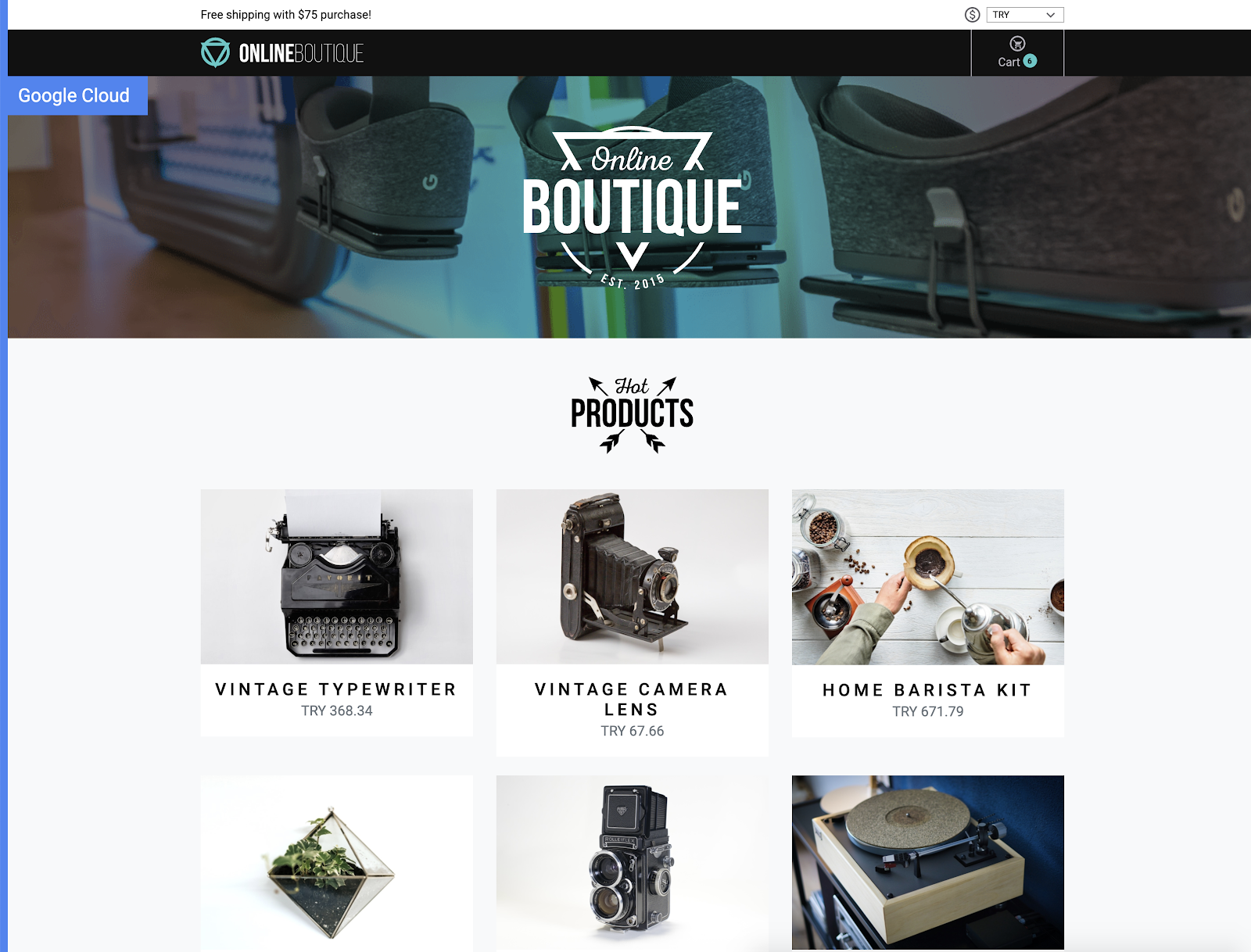
1. Now click on **Gateways,Services & Ingress**, verify all services are OK. Stay on this screen to set up monitoring for the application.

**Task 3. Open the application**

1. Scroll down to **frontend-external** and click the Endpoints IP of the service.



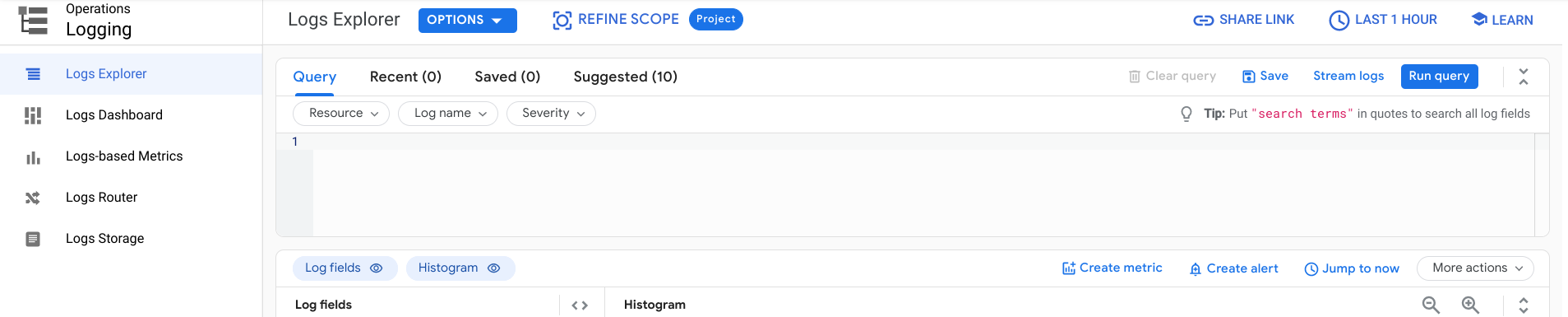
It should open the application and you will have a page like the following:



**Task 4. Create a logs-based metric**

Now you will configure Cloud Logging to create a [logs-based metric](https://cloud.google.com/logging/docs/logs-based-metrics), which is a custom metric in Cloud Monitoring made from log entries. Logs-based metrics are good for counting the number of log entries and tracking the distribution of a value in your logs. In this case, you will use the logs-based metric to count the number of errors in your frontend service. You can then use the metric in both dashboards and alerting.

1. Return to the Cloud Console, and from the **Navigation menu** open **Logging**, then click **Logs Explorer**.



1. Enable **Show query** and in the **Query builder** box, add the following query:

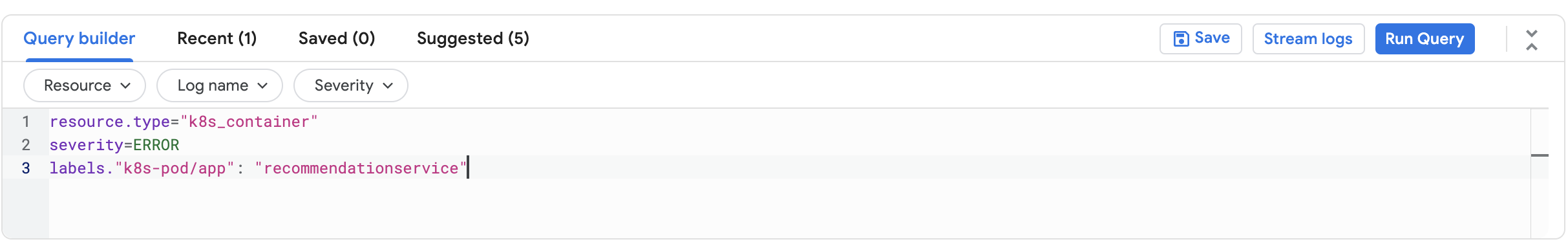
resource.type="k8s\_container"

severity=ERROR

labels."k8s-pod/app": "recommendationservice"

Copied!

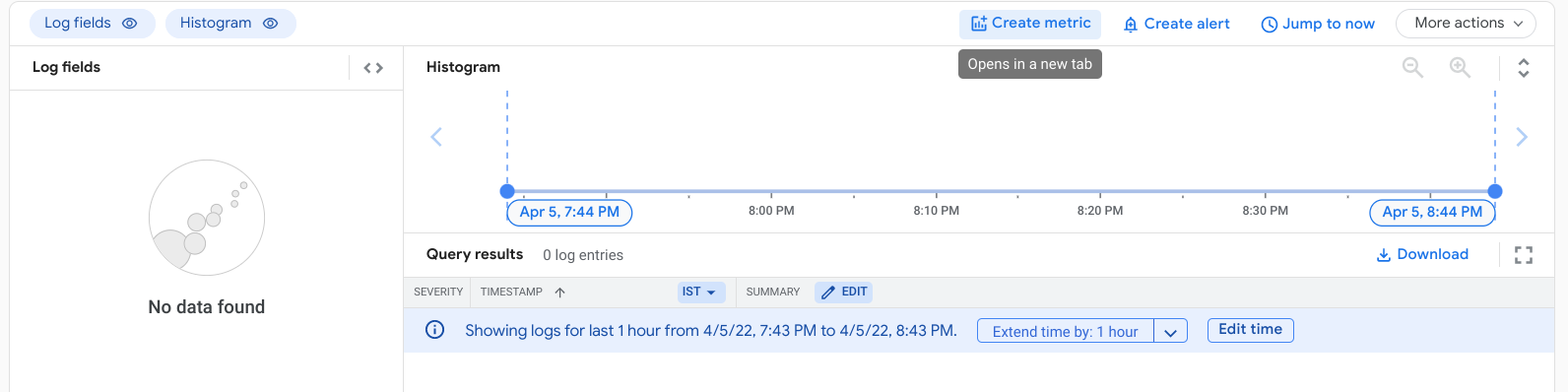
content\_copy



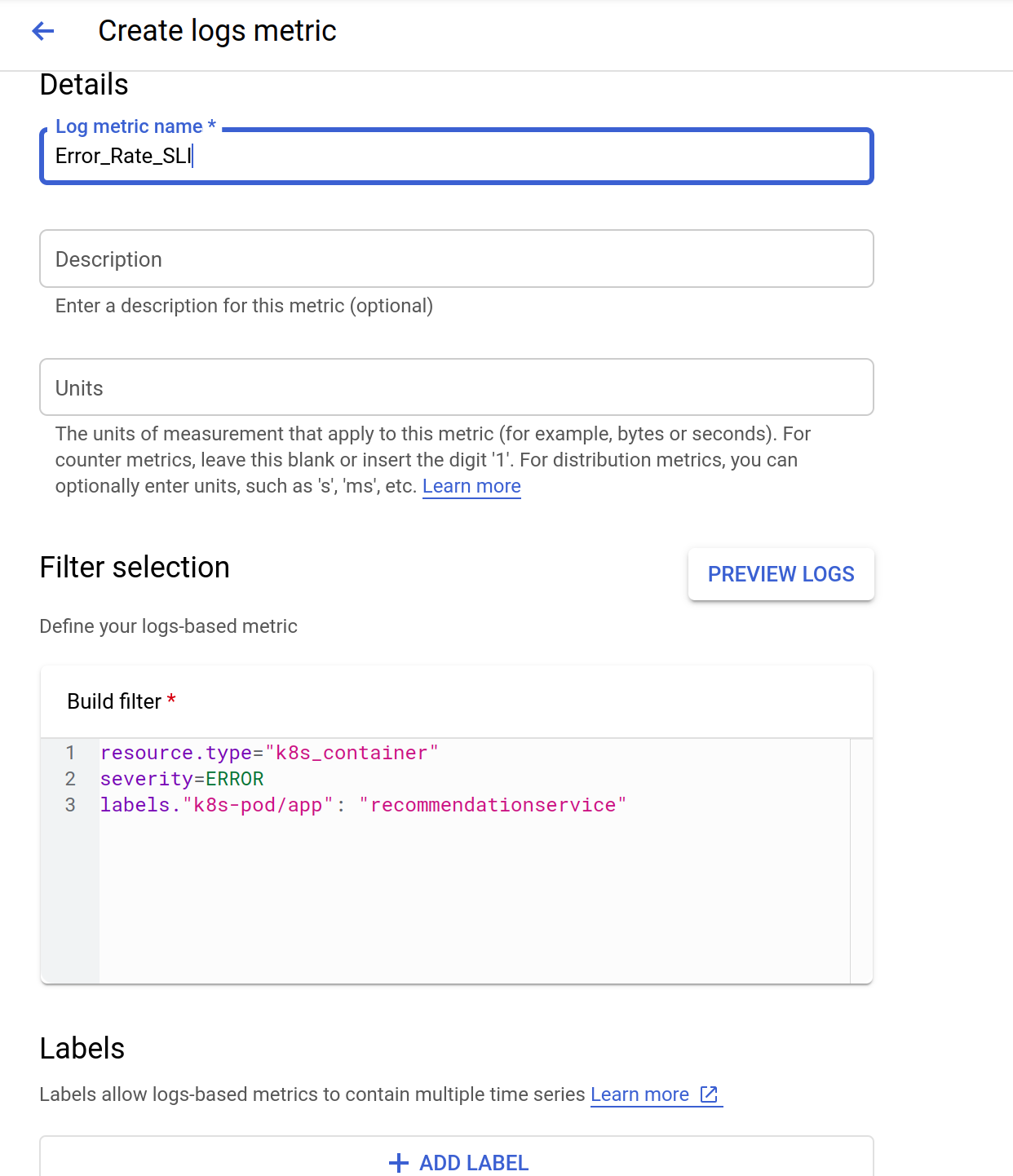
1. Click **Run Query**.

The query you are using lets you find all errors from the frontend pod. However, you shouldn't see any results now since there are no errors yet.

1. To create the logs-based metric, click on **Create Metric**.



1. Name the metric **Error\_Rate\_SLI,** and click **Create Metric** to save the log based metric:



You now see the metric listed under User-defined Metrics on the Logs-based Metrics page.

Click *Check my progress* to verify the objective.

Assessment Completed!

Create a logs-based metric

Check my progress

*Assessment Completed!*

**Task 5. Create an alerting policy**

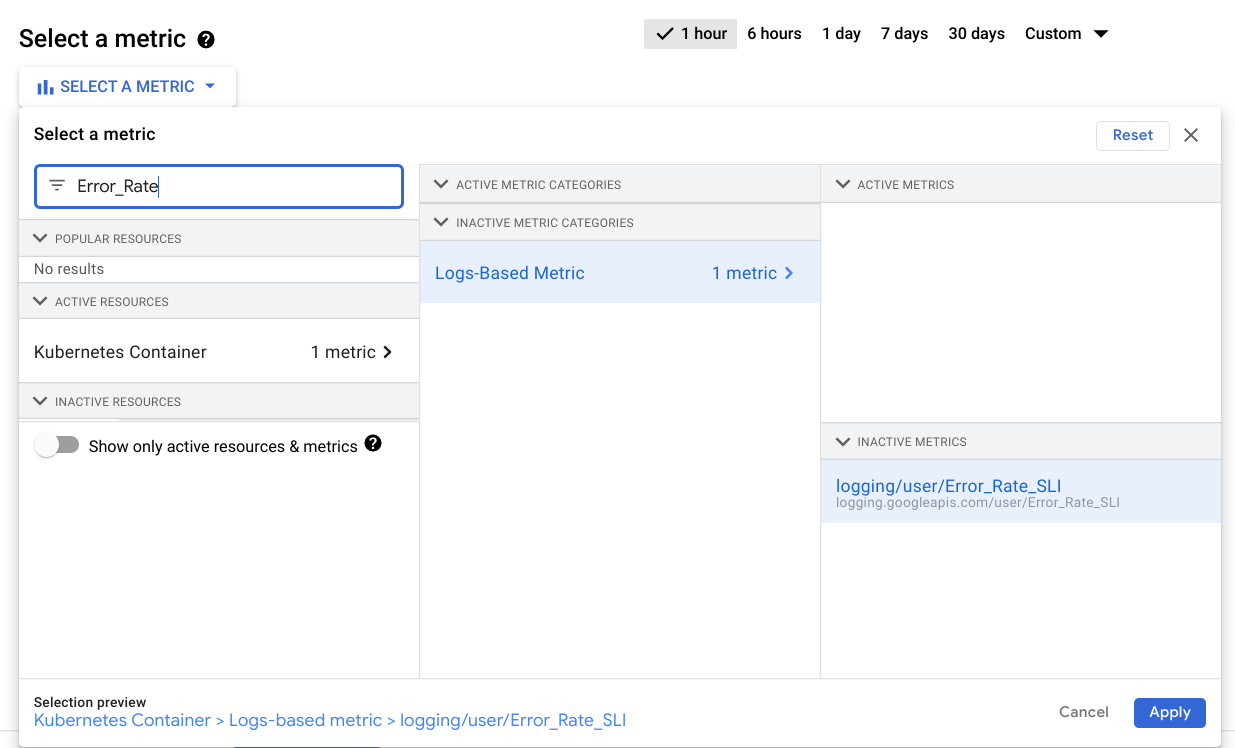
Alerting gives timely awareness to problems in your cloud applications so you can resolve the problems quickly. Now you will use Cloud Monitoring to monitor your frontend service availability by creating an alerting policy based on the frontend errors logs-based metric that you created previously. When the condition of the alerting policy is met, Cloud Monitoring creates and displays an incident in the Cloud console.

1. In the **Navigation menu**, open **Monitoring,** then click **Alerting**.
2. After the workspace is created, click **Create Policy** at the top.

**Note:** If required, click **Try It!** to use the updated alert creation flow.

1. Click on **Select a metric** dropdown. Unselect the **Active**
2. In **filter by resource and metric name** field, type **Error\_Rate**.
3. Click on **Kubernetes Container > Logs-Based Metric**. Select **logging/user/Error\_Rate\_SLI** and click **Apply**.

Your screen should look like this:



1. Set **Rolling windows function** to Rate.
2. Click **Next**.
3. Set **0.5** as your **Threshold value**.

As expected, there are no failures, and your application is meeting its availability Service Level Objective (SLO).

1. Click **Next** again.
2. Disable **Use notification channel**.
3. Provide an alert name such as Error Rate SLI then click **Next**.
4. Review the alert and click **Create Policy**.

**Note:**You will not create a notification channel for this lab but you should do it for your applications running in production, which allows you to send notifications in ways such as email, mobile app, SMS, Pub/Sub, and webhooks.

Click *Check my progress* to verify the objective.

Assessment Completed!

Create an alerting policy

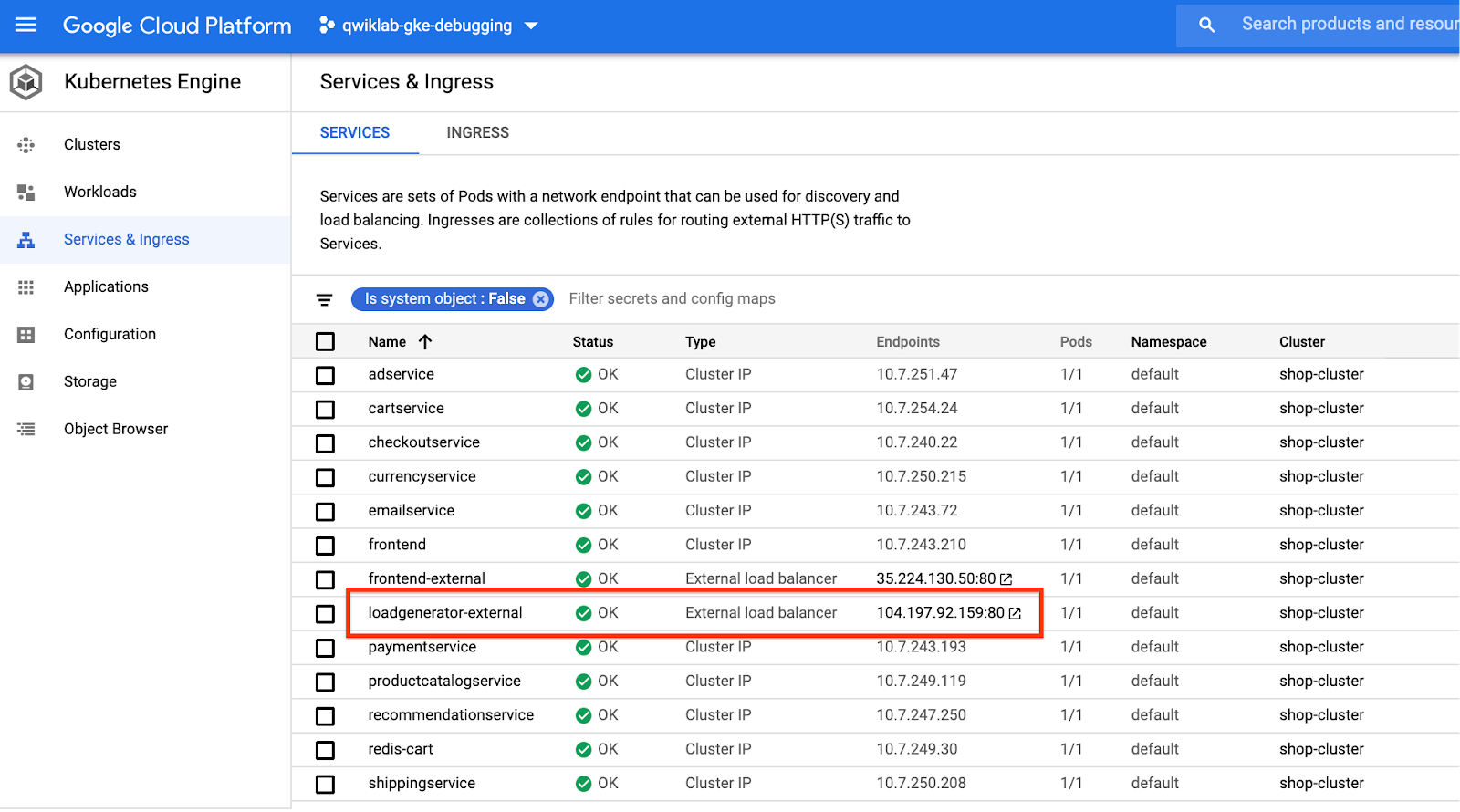
Check my progress

*Assessment Completed!*

**Trigger an application error**

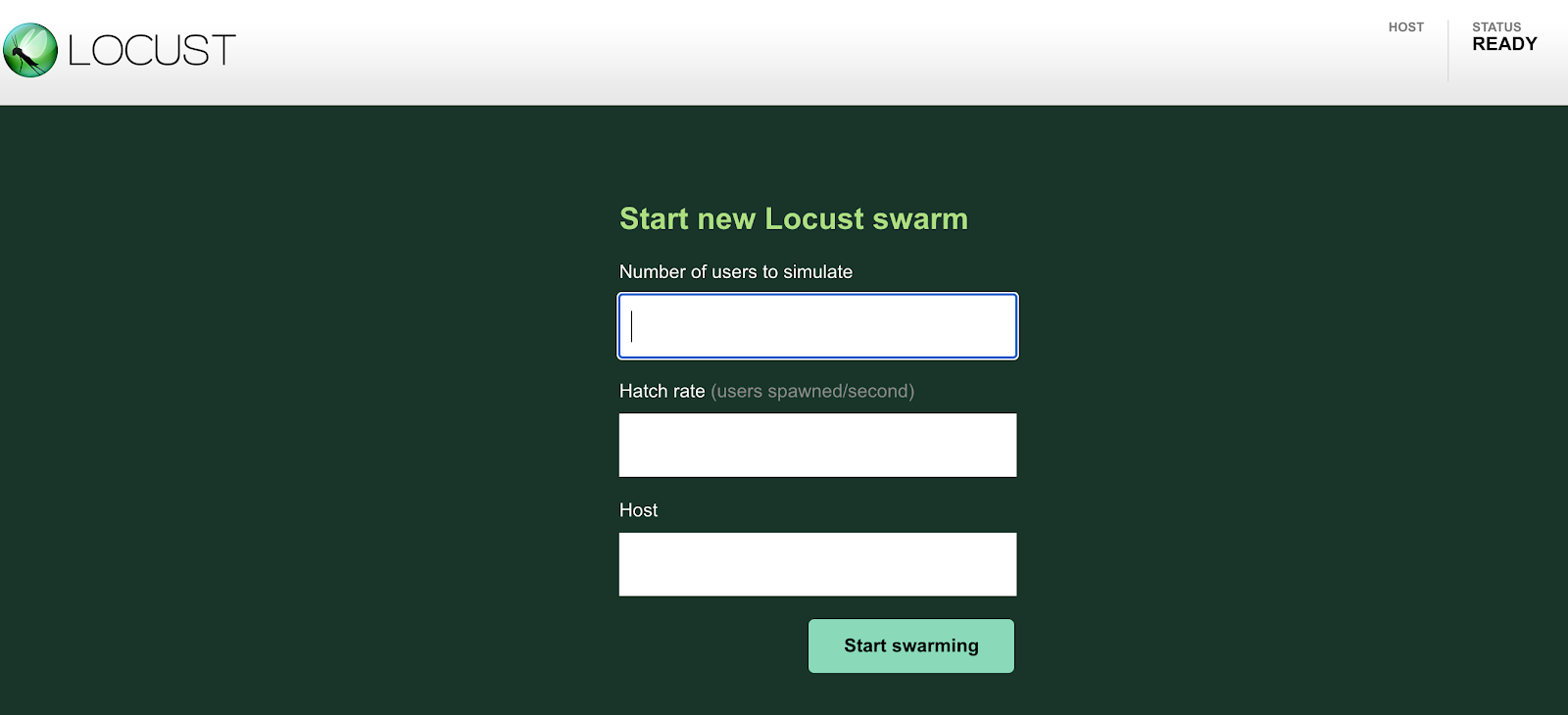
Now you will use a load generator to create some traffic for your web application. Since there is a bug that has been intentionally introduced into this version of the application, a certain amount of traffic volume will trigger errors. You will work through the steps to identify and fix the bug.

1. From the **Navigation menu**, select **Kubernetes Engine**, then **Gateways,Services & Ingress**.
2. Find the loadgenerator-external service, then click on the endpoints link.



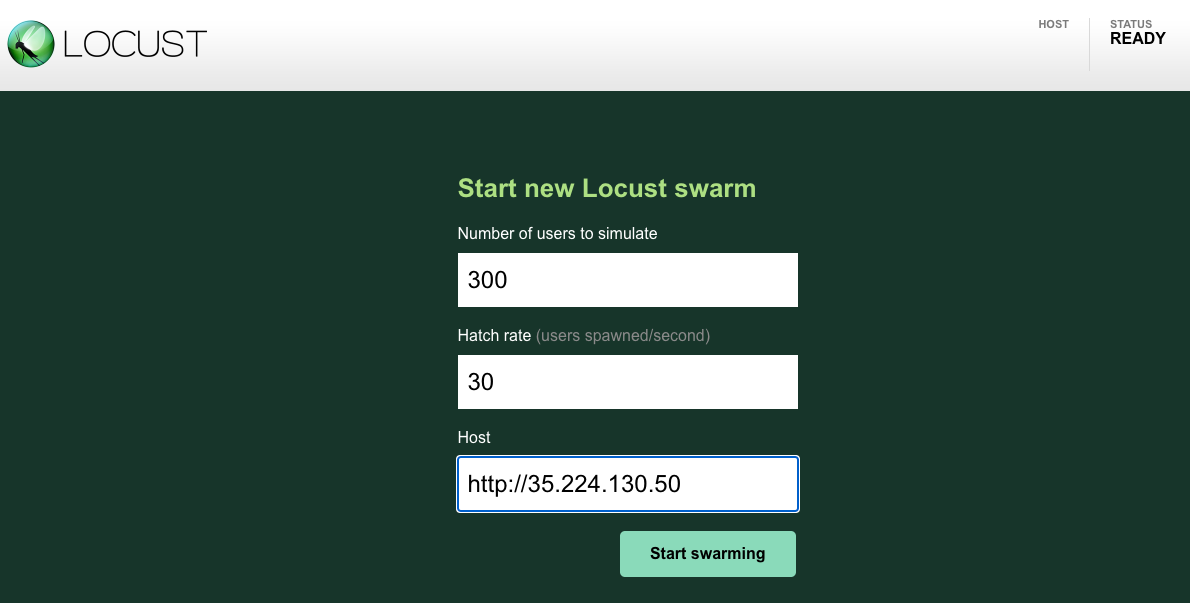
Alternatively, you can open a new browser tab or window, copy/paste the IP to the URL field, for example: http://\[loadgenerator-external-ip\]

You should now be on the Locust load generator page:

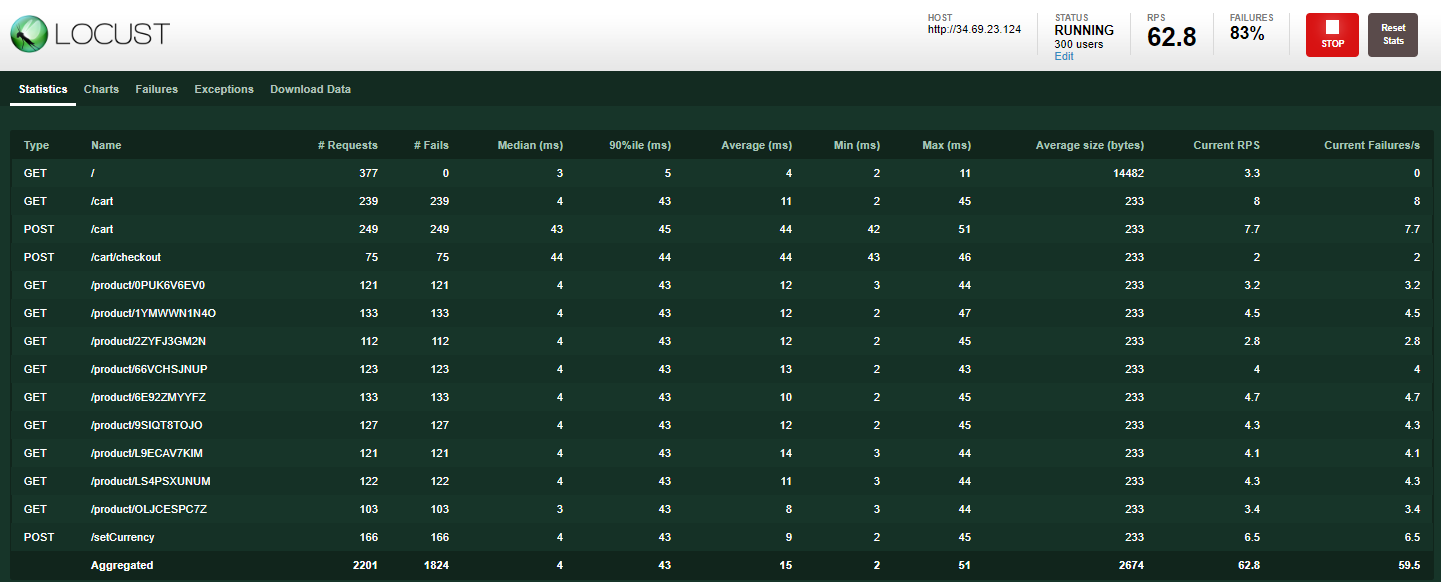


Locust is an open-source load generator, which allows you to load test a web app. It can simulate a number of users simultaneously hitting your application endpoints at a certain rate.

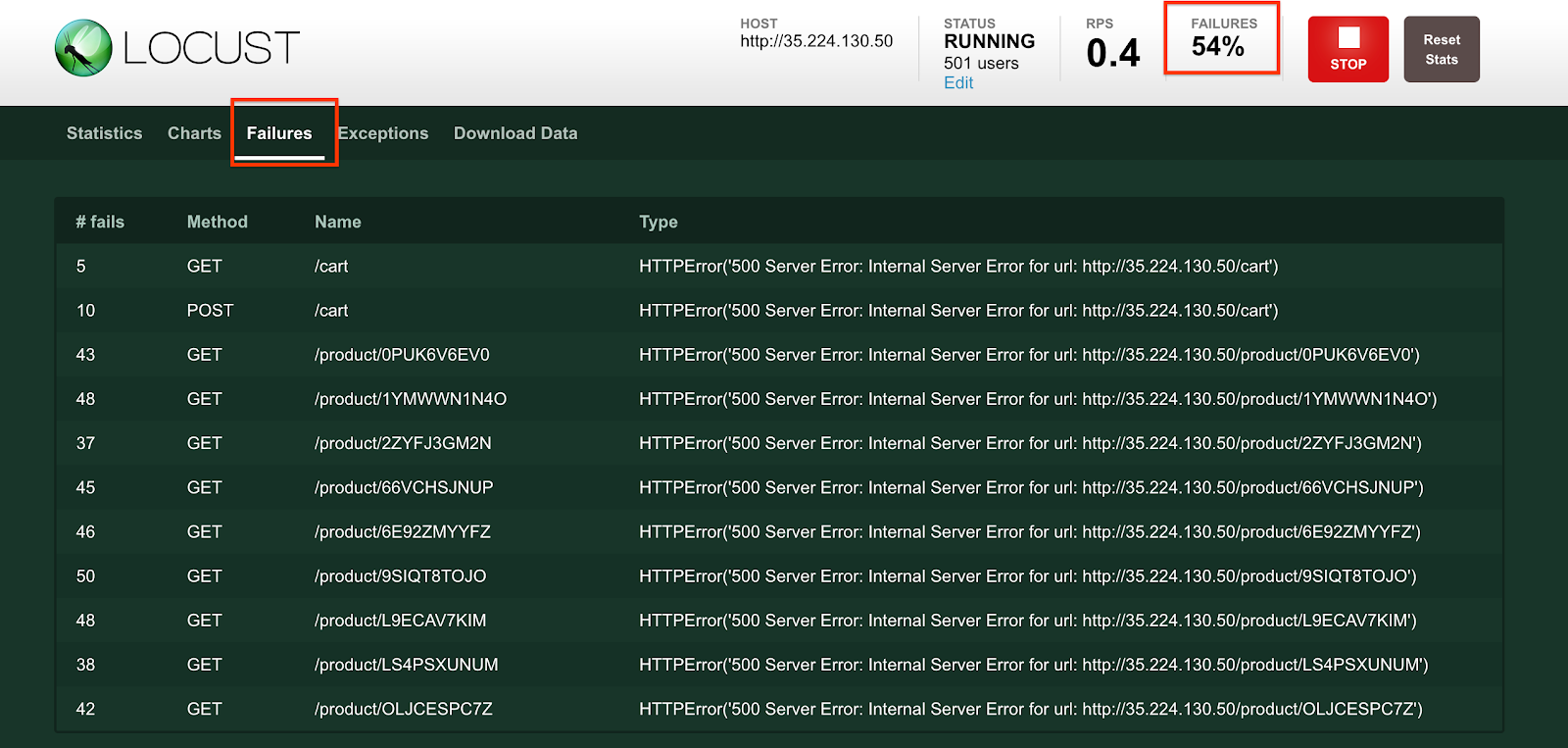
1. Simulate **300** users hitting the app with a hatch rate of **30**. Locust will add 30 users per second until it reaches 300 users.
2. For the host field, you will use the frontend-external. Copy the URL from the Gateways,Services & Ingress page; be sure to exclude the port. For example:



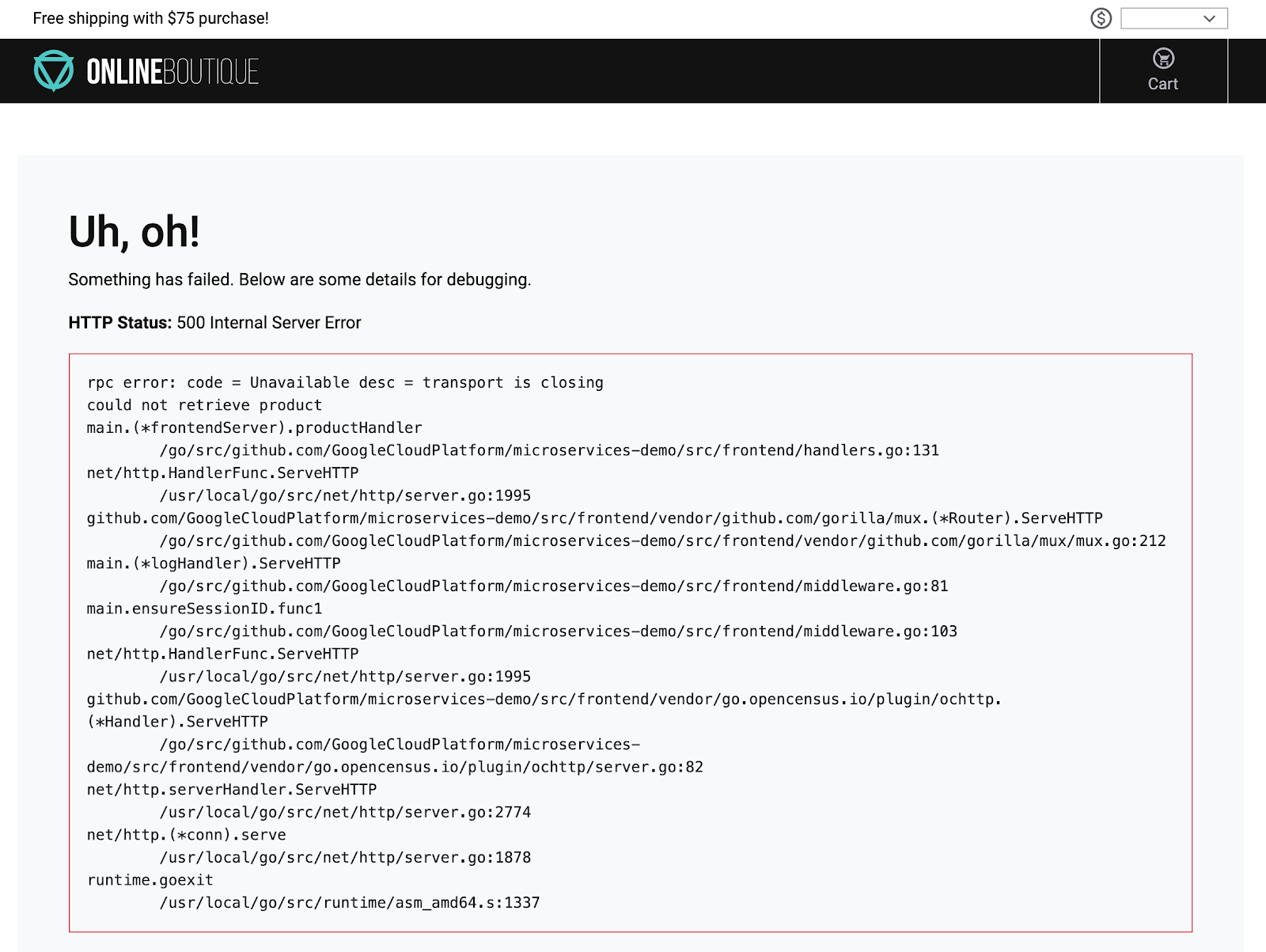
1. Click the **Start swarming** button. You should have about 300 users to hit the predefined URLs in a few seconds.



1. Click on the **Failures** tab to see that there are failures starting to occur. You can see there are a large number of 500 errors.

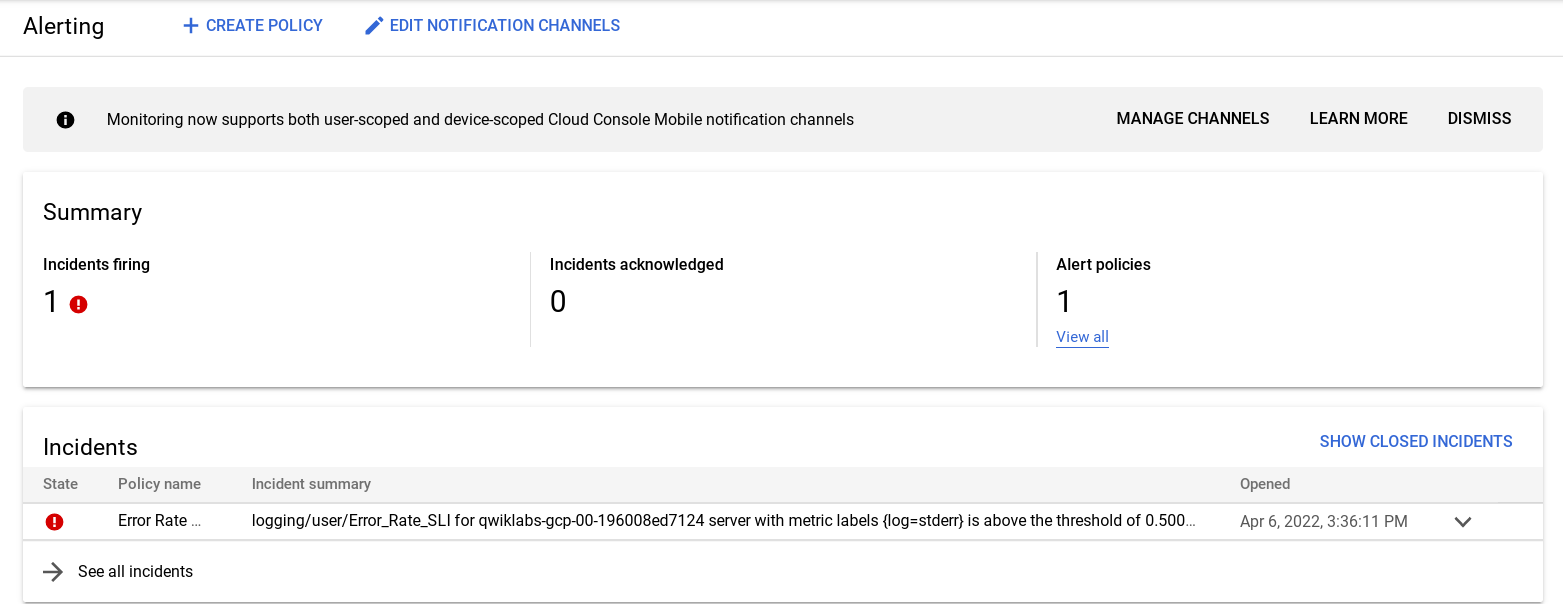


Meanwhile, if you click any product from the home page, it's either noticeably slow or you receive errors like the following if you click on a product:



**Confirming the alert and application errors**

1. In the console, from the **Navigation menu**, click **Monitoring**, then **Alerting**. You should see an incident soon regarding **logging/user/Error\_Rate\_SLI**. If you don't see an incident right away, wait a minute or two and refresh your page. It can take up to 5 minutes for the alert to fire.
2. Click the link of the incident:



It brings you to the details page.

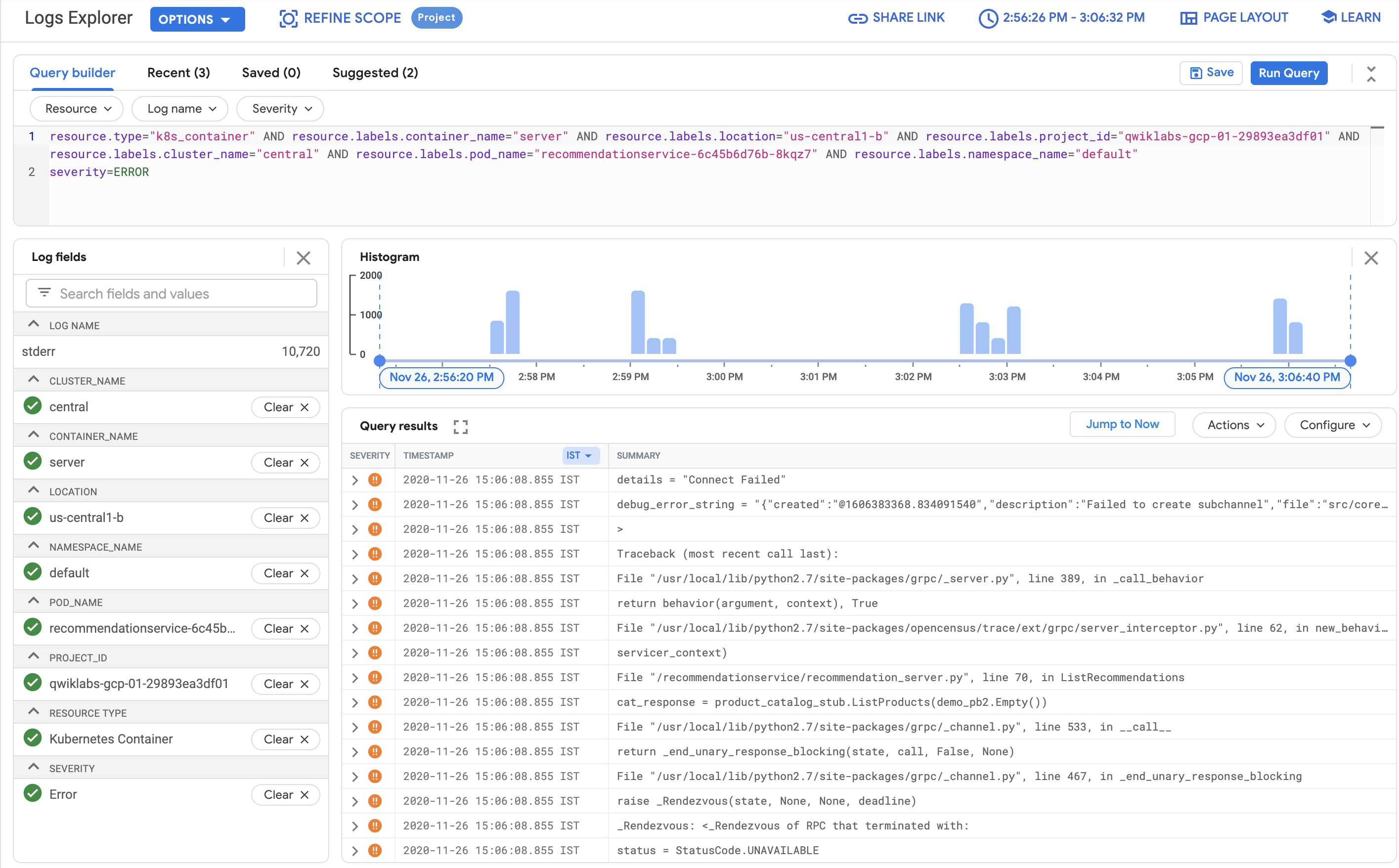
1. Click the **VIEW LOGS** link to view the logs for the pod.



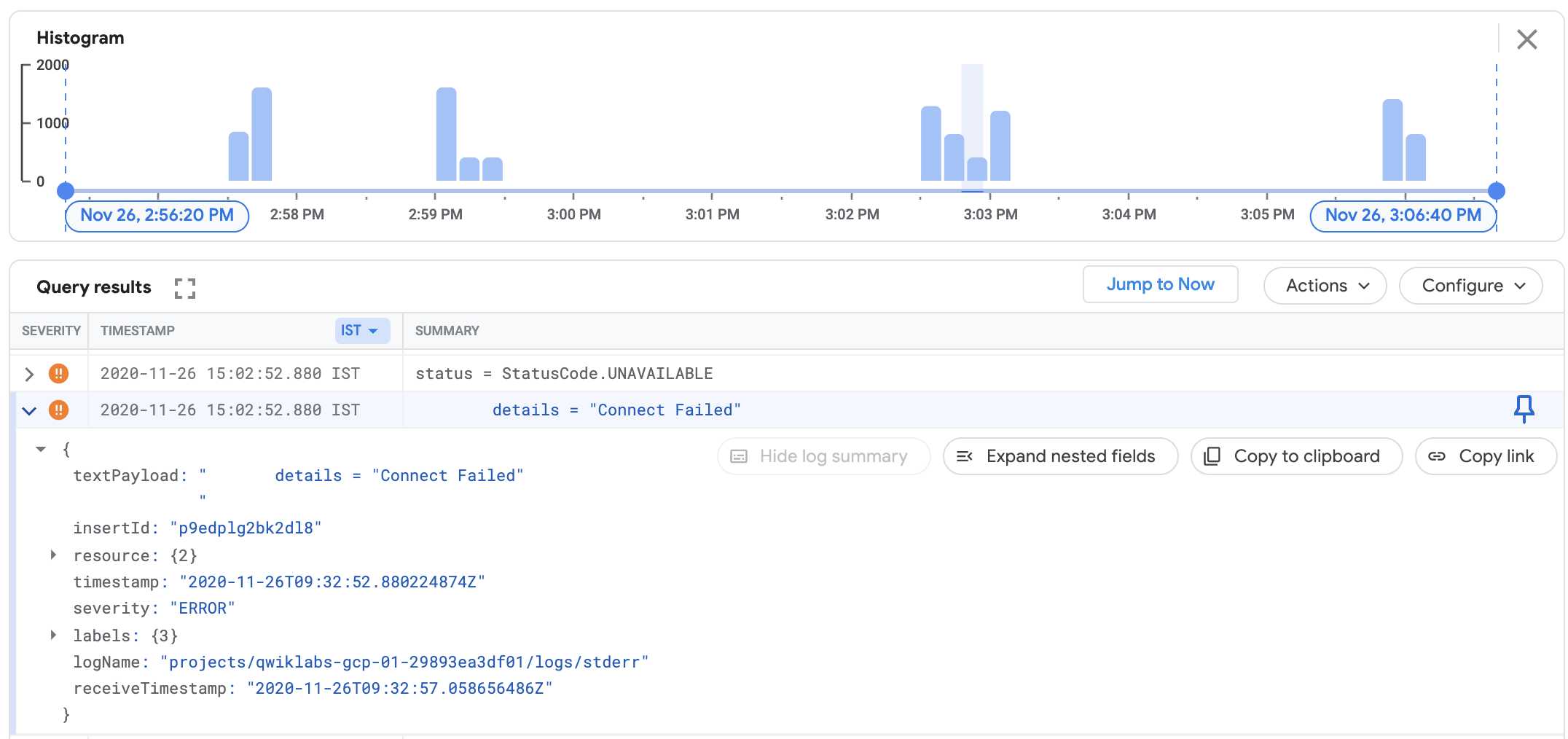
1. You can also click the **Error** label in the Logs field explorer panel to only query the errors.

Alternatively, you can click into the Query preview field to show the query builder, then click the **Severity** dropdown, add **Error** to the query. Click the **Add** button, then click **Run Query**. The dropdown menu allows adding multiple severity values.

The result either way is adding severity=ERROR to your query. Once you do that, you should have all the errors for the recommendationservice pod.



1. View the error details by expanding an error event. For example:

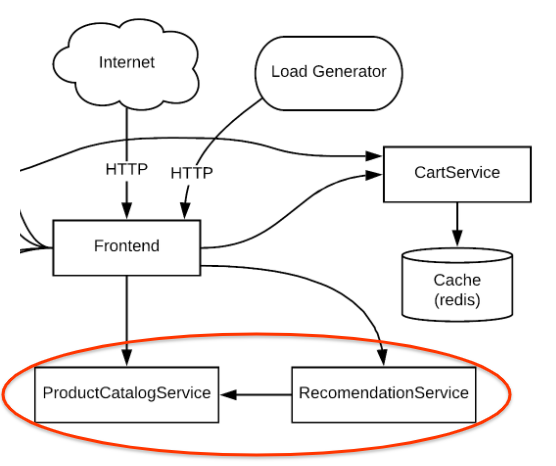


1. Expand the textPayload.
2. Click the error message and select **Add field to summary line** to have the error messages appearing as a summary field:



From there, you can confirm there are indeed many errors for the RecommendationService service. Based on the error messages, it appears the RecommendationService couldn't connect to some downstream services to either get products or recommendations. However, it's still not clear what the root cause is for the errors.

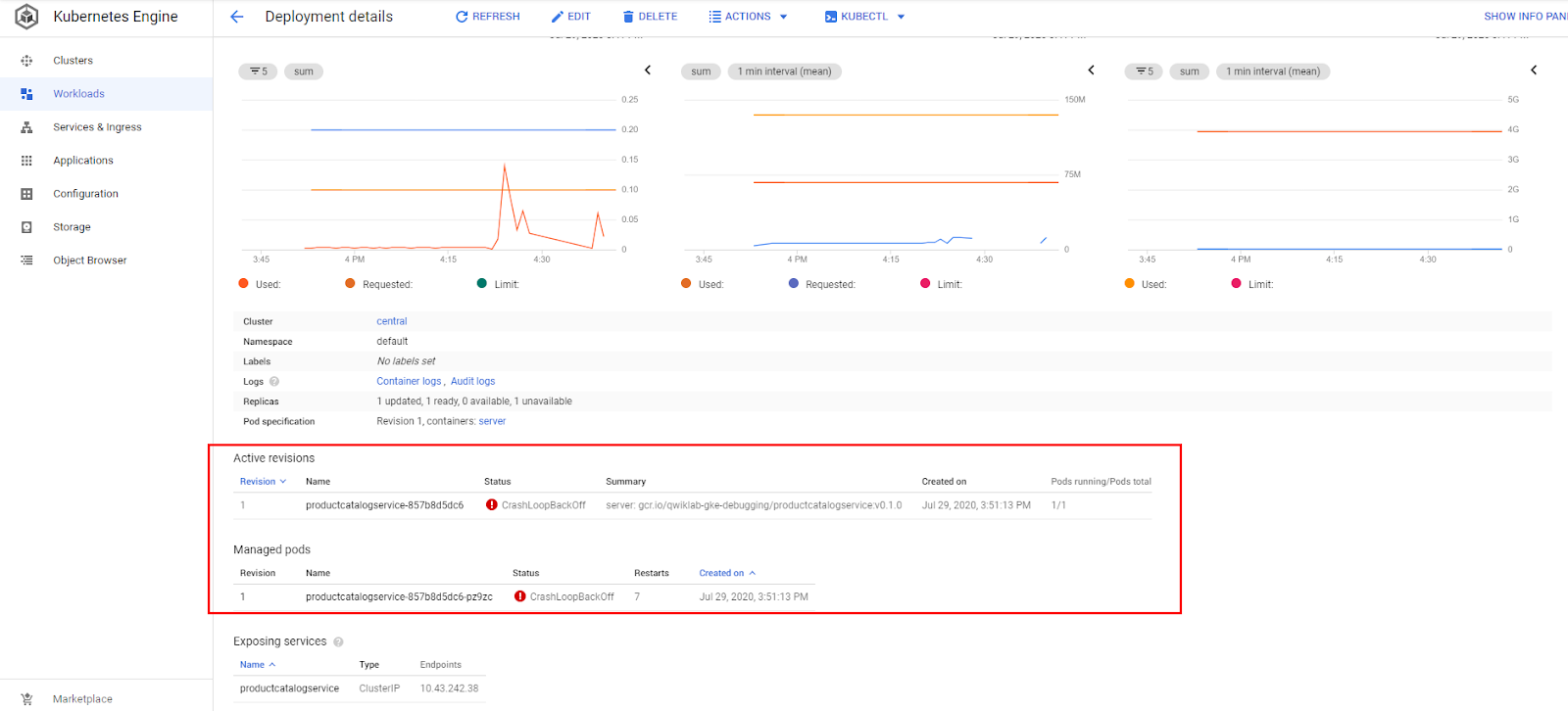
If you revisit the architecture diagram, the **RecommendationService** provides a list of recommendations to the **Frontend** services. However, both the **Frontend** service and the **RecommendationService** invoke **ProductCatalogService** for a list of products.



For the next step, you will look at the metrics of the main suspect, the **ProductCatalogService**, for any anomalies. Regardless, you can drill down in the logs to get some insights.

**Troubleshooting using the Kubernetes dashboard & logs**

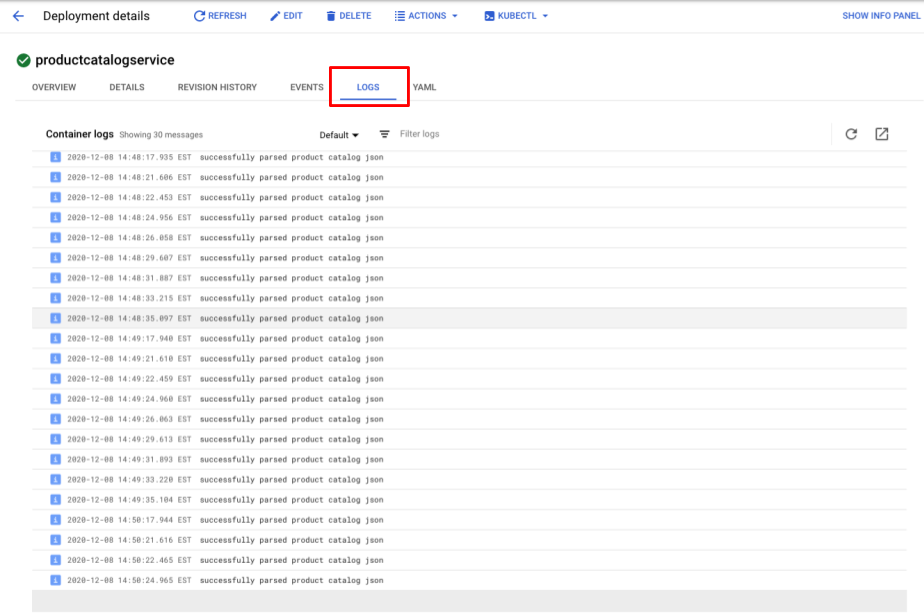
1. One of the first places that you can look at the metrics is the [Kubernetes Engine](https://console.cloud.google.com/monitoring/dashboards/resourceList/kubernetes) section of the Monitoring console (**Navigation menu** > **Monitoring**> **Dashboards** > **GKE**).
2. View the **Workloads** section.
3. Navigate to the **Kubernetes Engine** > **Workloads** > **productcatalogservice**. You can see the pod for the service is constantly crashing and restarting.



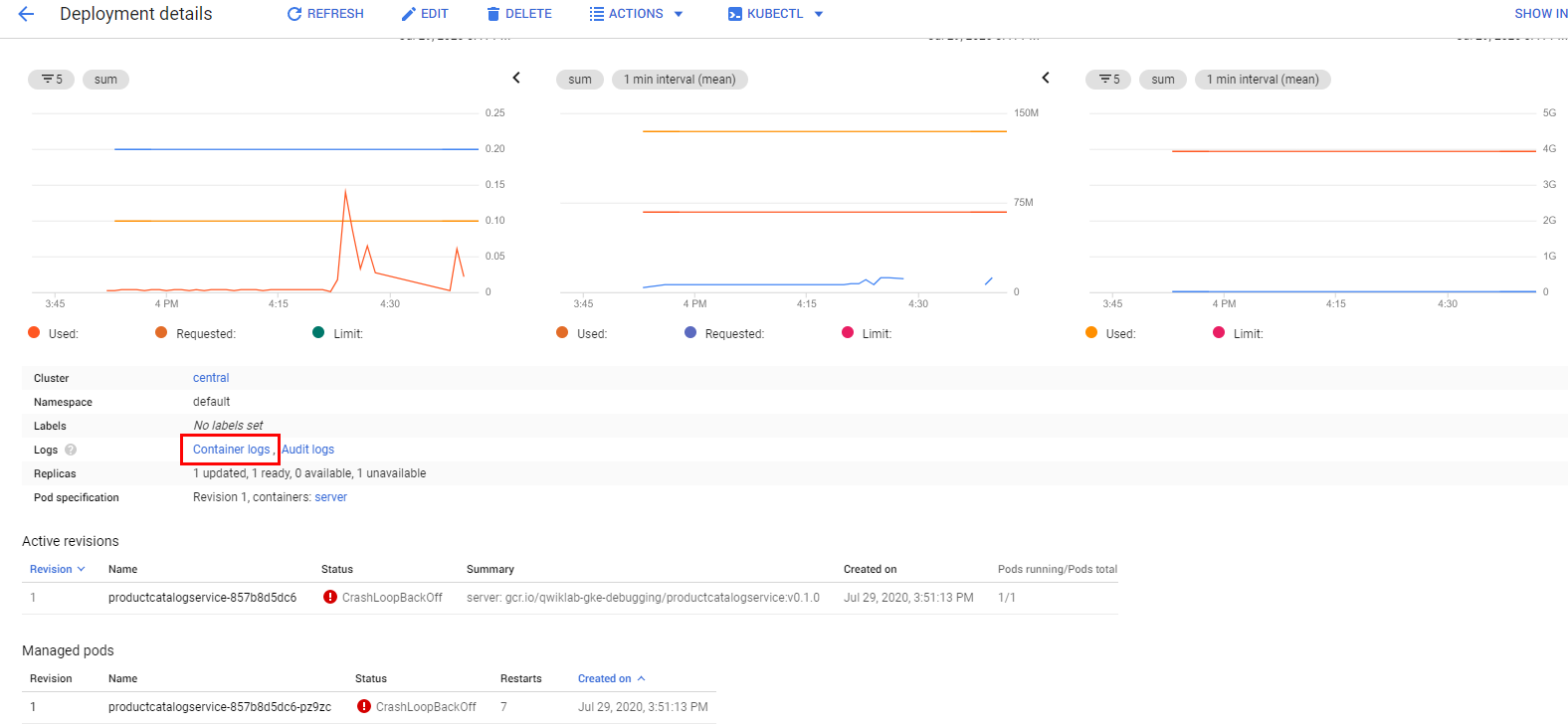
Next, see if there is anything interesting in the logs.

There are 2 ways to easily get to your container logs:

1. Click on the **Logs** tab to get a quick view of the most recent logs. Next, click the external link button in the upper right corner of the logs panel to go back to the Logs Explorer.



1. In the overview page, click the **Container logs** link on the Deployment Details page.



You are on the Logs Explorer page again, now with a predefined query specifically filtered for the logs from the container you were viewing in GKE.

From the Log Viewer, both the log messages and the histogram show the container is repeatedly parsing product catalogs within a short period of time. It seems very inefficient.

At the bottom of the query results, there might also be a runtime error like the following one:

panic: runtime error: invalid memory address or nil pointer dereference

[signal SIGSEGV: segmentation violation

This could actually be causing the pod to crash.

To better understand the reason, search the log message in the code.

1. In Cloud Shell, run the following command:

grep -nri 'successfully parsed product catalog json' src

Copied!

content\_copy

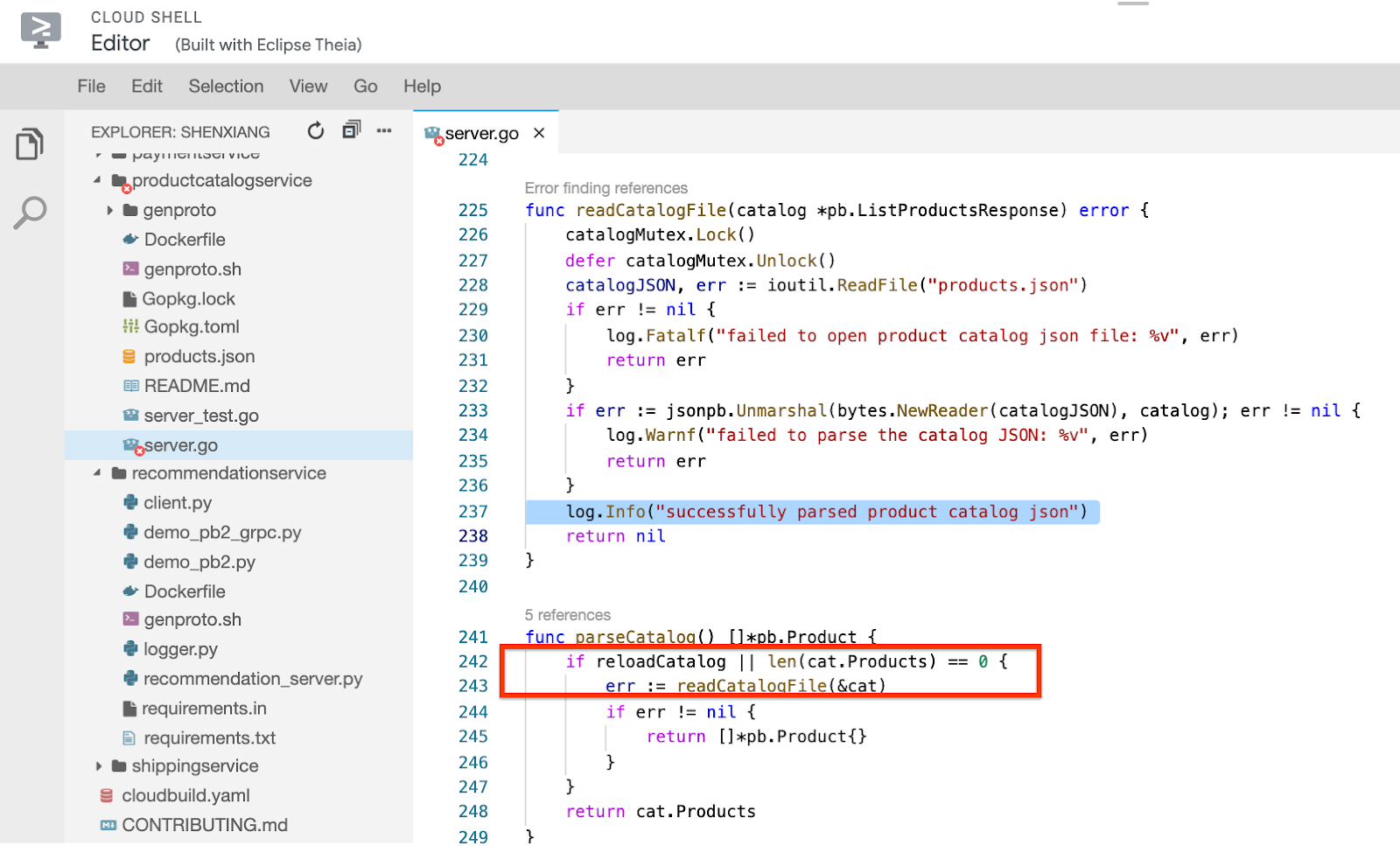
Your output should look like the following, which has the source file name with a line number:

src/productcatalogservice/server.go:237: log.Info("successfully parsed product catalog json")

1. To view the source file, by clicking the **Open Editor** button in the Cloud Shell menu, then **Open in New Window** (if you see the Unable to load code editor because third-party cookies are disabled error, click the eye at the top of the Chrome page).

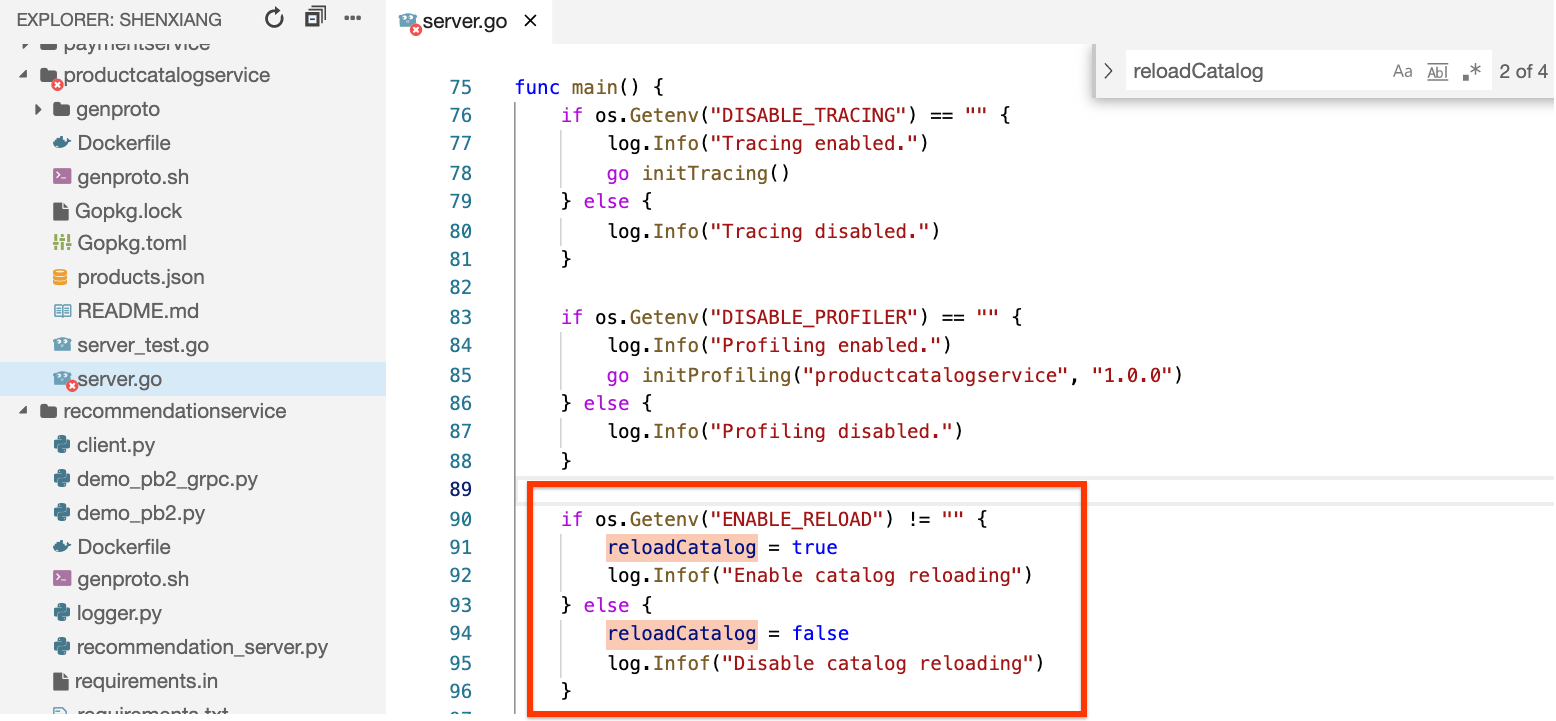
The Open Editor button highlighted in the UI

1. Click the file microservices-demo/src/productcatalogservice/server.go, scroll down to line 237, and you will find the **readCatalogFile** method logs this message:



With a little more effort, you can see that if the boolean variable **reloadCatalog** is true, the service reloads and parses the product catalog each time it's invoked, which seems unnecessary.

If you search the **reloadCatalog** variable in the code, you can see it's controlled by the environment variable ENABLE\_RELOAD and writes a log message for its state.



Check the logs again by adding this message to your query and determine if there are any entries that exist.

1. Return to the tab where Logs Explorer is open and add the following line to the query:

jsonPayload.message:"catalog reloading"

Copied!

content\_copy

So the full query in your query builder is:

resource.type="k8s\_container"

resource.labels.location="europe-west1-d"

resource.labels.cluster\_name="central"

resource.labels.namespace\_name="default"

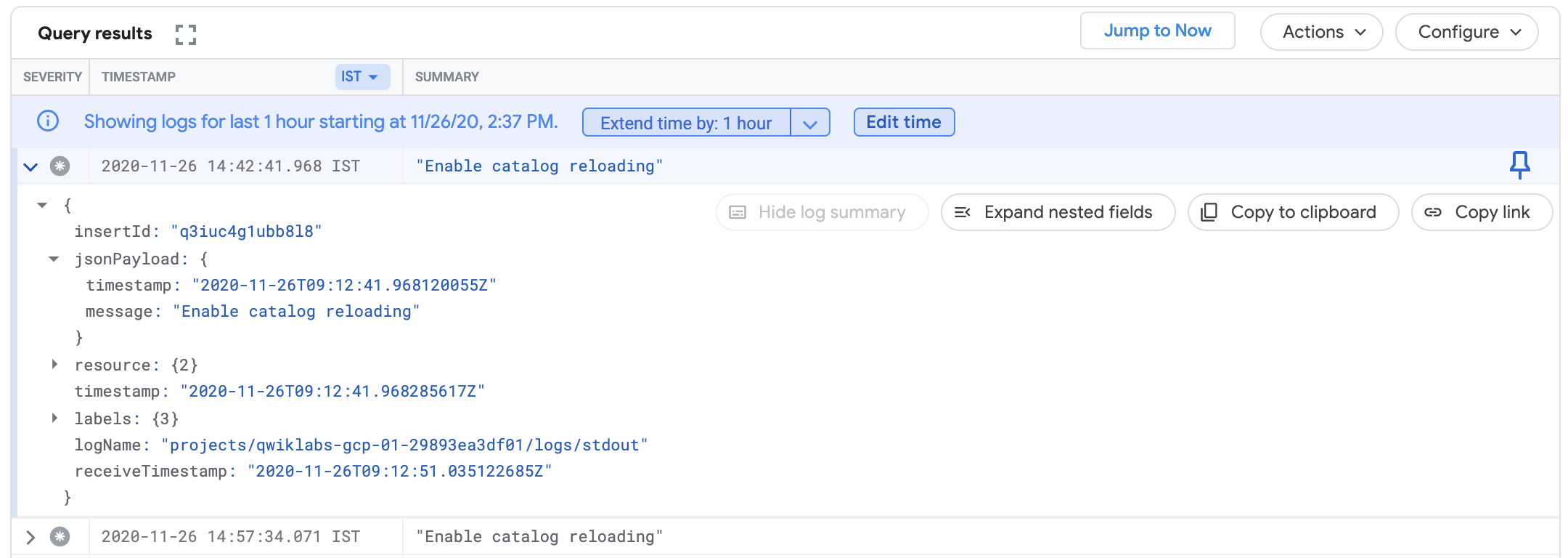
labels.k8s-pod/app="productcatalogservice"

jsonPayload.message:"catalog reloading"

Copied!

content\_copy

1. Click **Run Query** again and find an "Enable catalog reloading" message in the container log. This confirms that the catalog reloading feature is enabled.



At this point you can be certain the frontend error is caused by the overhead to load the catalog for every request. When you increased the load, the overhead caused the service to fail and generate the error.

**Task 6. Fix the issue and verify the result**

Based on the code and what you're seeing in the logs, you can try to fix the issue by disabling catalog reloading. Now you will remove the ENABLE\_RELOAD environment variable for the product catalog service. Once you make the variable changes, then you can redeploy the application and verify that the changes have addressed the observed issue.

1. Click the **Open Terminal** button to return to the Cloud Shell terminal if it has closed.
2. Run the following command:

grep -A1 -ni ENABLE\_RELOAD release/kubernetes-manifests.yaml

Copied!

content\_copy

The output will show the line number of the environment variable in the manifest file:

373: - name: ENABLE\_RELOAD

374- value: "1"

1. Delete those two lines to disable the reloading by running:

sed -i -e '373,374d' release/kubernetes-manifests.yaml

Copied!

content\_copy

1. Then reapply the manifest file:

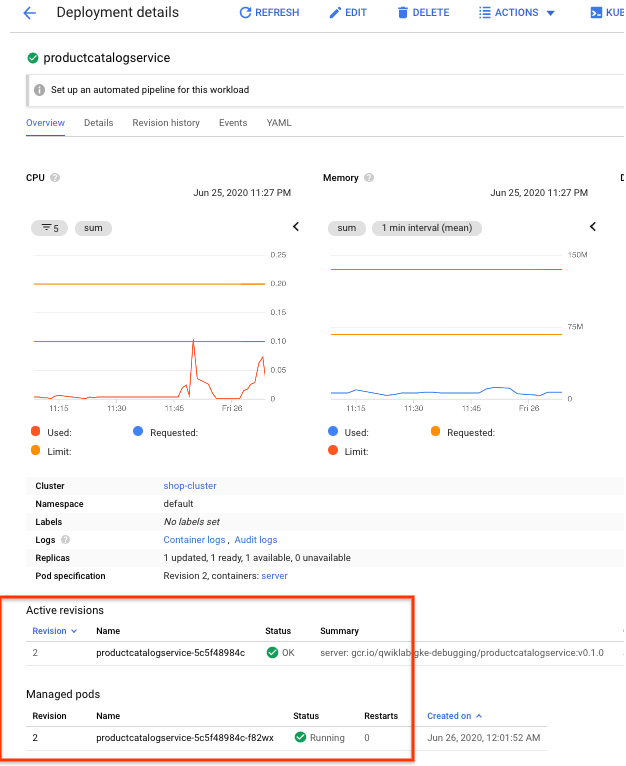
kubectl apply -f release/kubernetes-manifests.yaml

Copied!

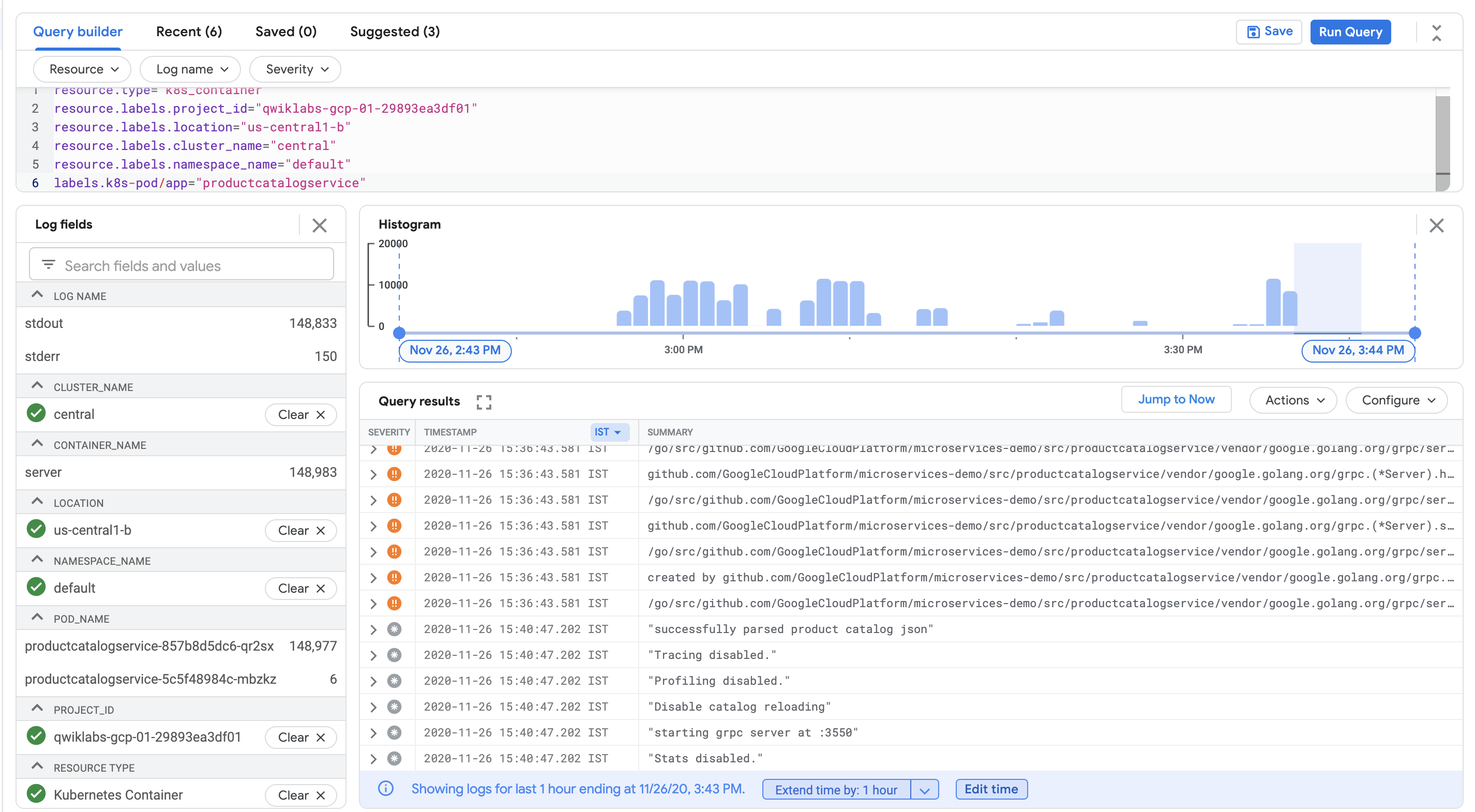
content\_copy

You will notice only the **productcatalogservice** is configured. The other services are unchanged.

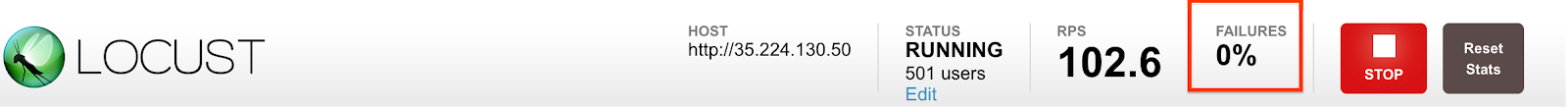
1. Return to the Deployment detail page (**Navigation menu** > **Kubernetes Engine** > **Workloads** > **productcatalogservice**), and wait until the pod runs successfully. Wait 2-3 minutes or until you can confirm it stops crashing.



1. If you click the **Container logs** link again, you will see the repeating successfully parsing the catalog json messages are gone:



1. If you go back to the webapp URL and click the products on the home page, it's also much more responsive and you shouldn't encounter any HTTP errors.
2. Go back to the load generator, click the **Reset Stats** button in the top right. The failure percentage is reset and you should not see it increasing anymore.



All above checks indicate that the issue is fixed. If you are still seeing the 500 error, wait another couple of minutes and try clicking on a product again.

**Congratulations**

You used Cloud Logging and Cloud Monitoring to find an error in an intentionally misconfigured version of the microservices demo app. This is a similar troubleshooting process that you would use to narrow down issues for your GKE apps in a production environment.

First, you deployed the app to GKE and then set up a metric and alert for frontend errors. Next, you generated a load and then noticed that the alert was triggered. From the alert, you narrowed down the issue to particular services using Cloud Logging. Then, you used Cloud Monitoring and the GKE UI to look at the metrics for the GKE services. To fix the issue, you then deployed an updated configuration to GKE and confirmed that the fix addressed the errors in the logs.

Next steps / Learn more

* This lab is based on this [blog post](https://cloud.google.com/blog/products/management-tools/using-logging-your-apps-running-kubernetes-engine) on using Logging for your apps running on GKE.
* The [follow-up post](https://cloud.google.com/blog/products/containers-kubernetes/tools-for-debugging-apps-on-google-kubernetes-engine) on how DevOps teams can use Cloud Monitoring and Logging to find issues quickly might also be interesting to read.

Google Cloud training and certification

...helps you make the most of Google Cloud technologies. [Our classes](https://cloud.google.com/training/courses) 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](https://cloud.google.com/certification/) help you validate and prove your skill and expertise in Google Cloud technologies.

**Manual Last Updated: April 10, 2024**

**Lab Last Tested: April 10, 2024**

Copyright 2024 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are associated.