

JAVAMS13 Working with Kubernetes Monitoring

2 hoursFree

Rate Lab

Overview

In this series of labs, you take a demo microservices Java application built with the Spring framework and modify it to use an external database server. You adopt some of the best practices for tracing, configuration management, and integration with other services using integration patterns.

In the previous lab, you containerized the application and deployed it to a Kubernetes Engine cluster with Stackdriver Kubernetes Monitoring support. That means you can monitor the health of the Kubernetes Engine cluster using Stackdriver. A replica of that environment is preconfigured for you in this lab.

Traditionally, Java applications are monitored through JMX metrics, which provide metrics on such things as thread count and heap usage. In the cloud-native world where you monitor more than just the Java stack, you need to use more generic metrics formats, such as Prometheus.

Stackdriver Kubernetes Monitoring aggregates logs, events, and metrics from your Kubernetes Engine environment to help you understand your application's behavior in production. Prometheus is an optional monitoring tool often used with Kubernetes. If you configure Stackdriver Kubernetes Monitoring with Prometheus support, then services that expose metrics using the Prometheus data model also have their data exported from the cluster and made visible as external metrics in Stackdriver.

In this lab, you enable Prometheus monitoring for Kubernetes and then modify the demo application to expose Prometheus metrics from within the application and its backend service. You can then use Stackdriver to monitor internal performance metrics from your application while it is running on Kubernetes Engine.

Objectives

In this lab, you learn how to perform the following tasks:

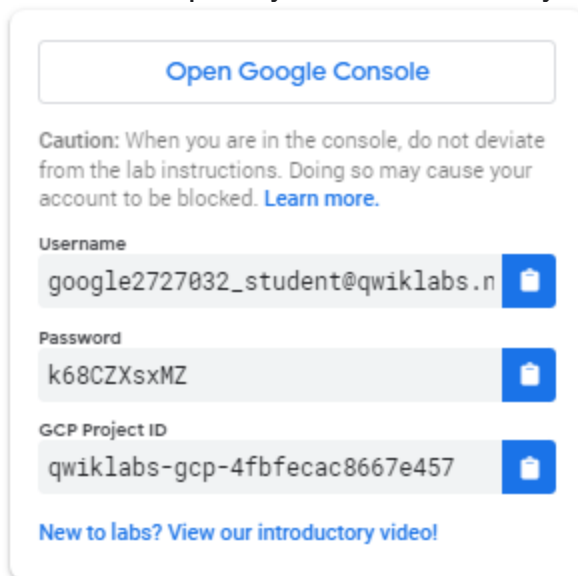
- Enable Stackdriver Monitoring for Kubernetes Engine
- Enable Prometheus monitoring in a Kubernetes Engine cluster
- Expose Prometheus metrics from inside a Spring Boot application
- Explore live application metrics using Stackdriver Monitoring

Task 0. Lab Preparation

Access Qwiklabs

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.



Open Google Console

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

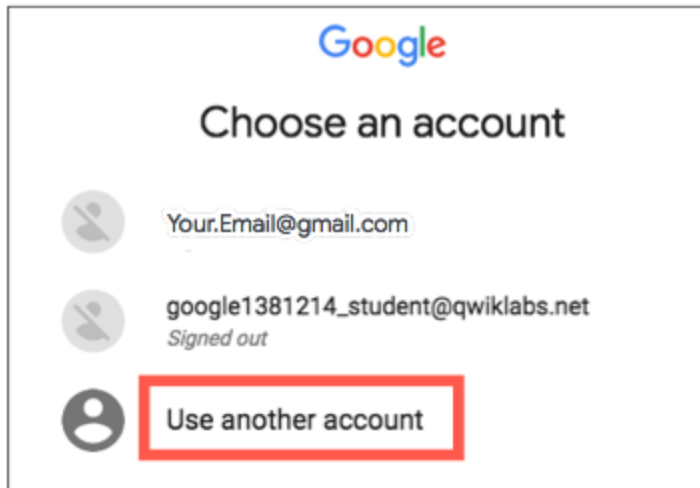
Username
google2727032_student@qwiklabs.n

Password
k68CZXsxMZ

GCP Project ID
qwiklabs-gcp-4fbfecac8667e457

[New to labs? View our introductory video!](#)

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.

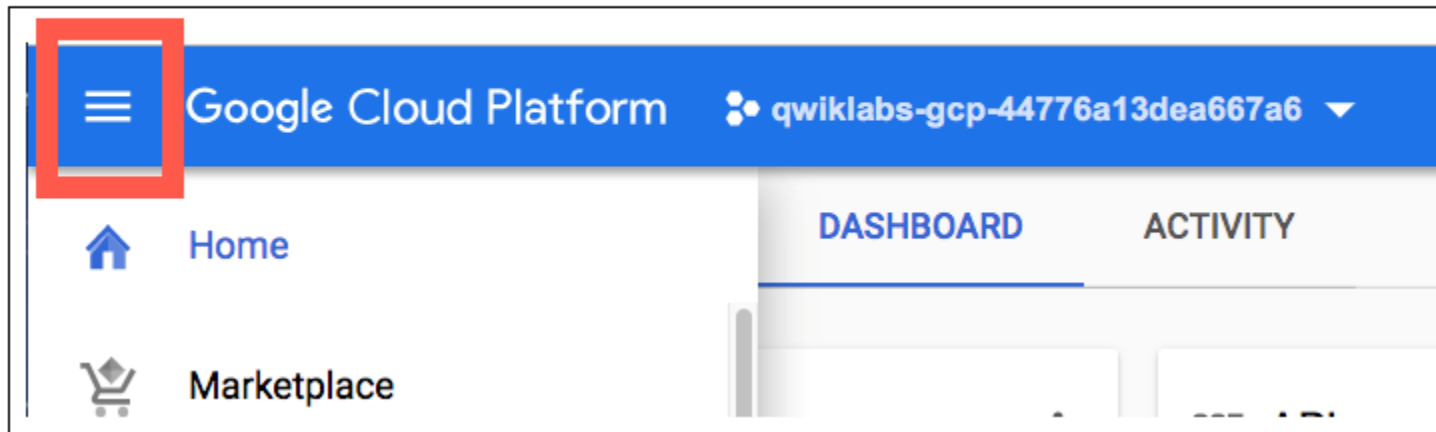
Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own GCP account, do not use it for this lab (avoids incurring charges).

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.

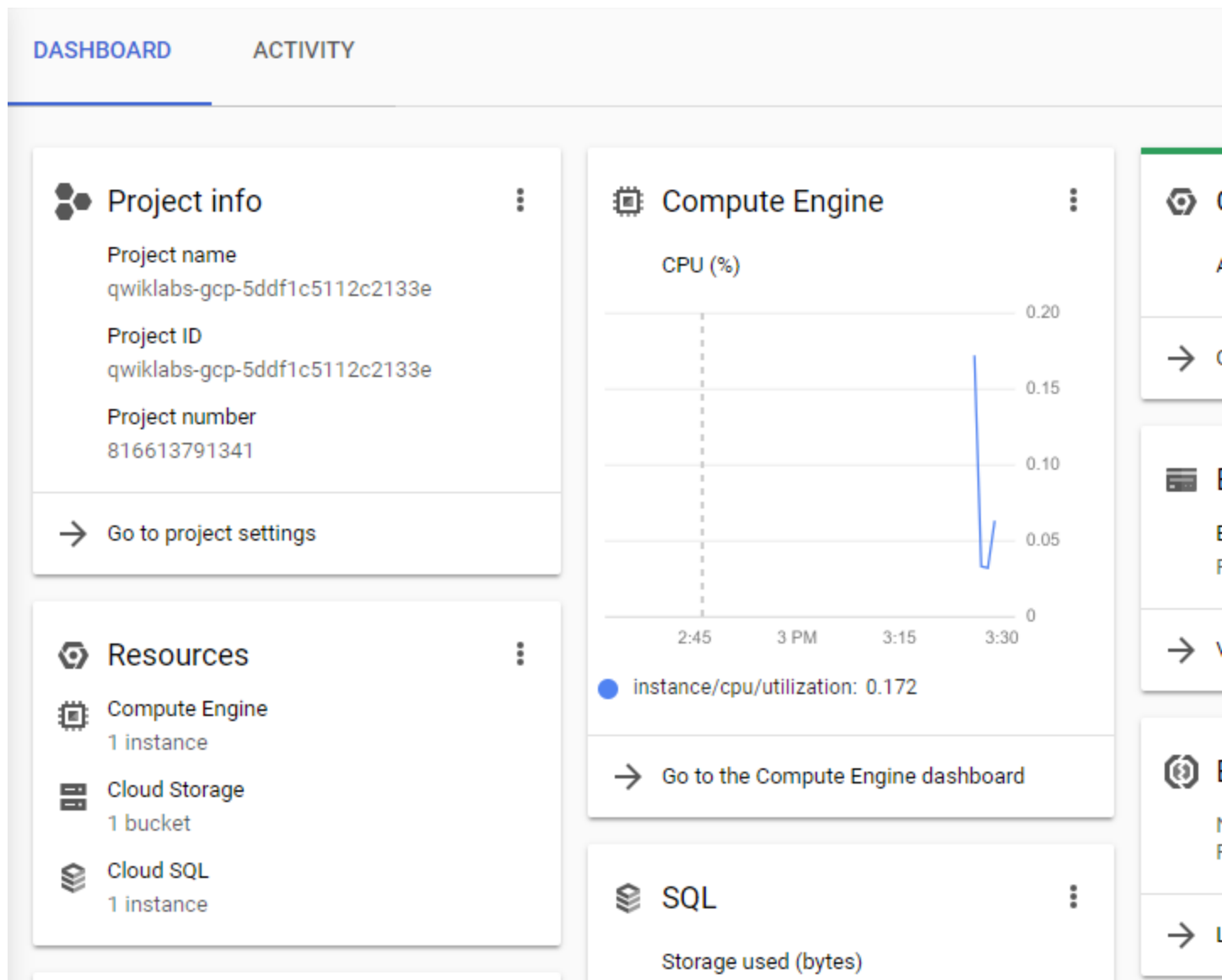
After a few moments, the GCP console opens in this tab.

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”.



After you complete the initial sign-in steps, the project dashboard appears.



Fetch the application source files

The lab setup includes automated deployment of the services that you configured yourself in previous labs. When the setup is complete, copies of the demo application (configured so that they are ready for this lab session) are put into a Cloud Storage bucket named using the project ID for this lab.

Before you proceed with the tasks for this lab, you must first copy the demo application into Cloud Shell so you can continue to work on it.

1. In the upper-right corner of the screen, click **Activate Cloud**



Shell () to open Cloud Shell.

2. Click **Start Cloud Shell**.

Boost mode is not needed for this lab.

3. In the Cloud Shell command line, enter the following command to create an environment variable that contains the project ID for this lab:

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

4. Verify that the demo application files were created.

```
gsutil ls gs://$PROJECT_ID
```

Repeat the last step if the command reports an error or if it does not list the two folders for the `guestbook-frontend` application and the `guestbook-service` backend application and a folder that contains sample Kubernetes deployment files.

Note

A Cloud Storage bucket that is named using the project ID for this lab is automatically created for you by the lab setup. The source code for your applications is copied into this bucket when the Cloud Spanner instance that is used for the service application in this lab is ready. Compared to the some earlier labs in this course the startup process for this is relatively quick so you might not have to wait here for it to complete.

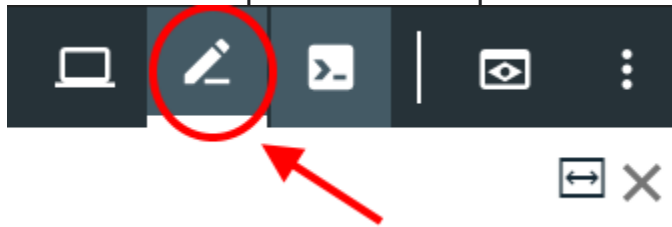
5. Copy the application folders to Cloud Shell.

```
gsutil -m cp -r gs://$PROJECT_ID/* ~/
```

6. Make the Maven wrapper scripts executable.

```
chmod +x ~/guestbook-frontend/mvnw  
chmod +x ~/guestbook-service/mvnw
```

7. Click the pencil icon to open the Cloud Shell code editor.



Task 1. Enable Stackdriver Monitoring and view the Stackdriver Kubernetes Monitoring dashboard

In this task, you enable Stackdriver Monitoring and view the overview dashboard for the metrics that are monitored for Kubernetes Engine.

1. In your GCP console tab navigate to **Stackdriver > Monitoring** in the Navigation menu.
2. If prompted, click **Log in with Google** and select the qwiklabs-generated student account to log in.

Note: If you are prompted to **Create workspace**, follow these steps:

1. Click **Create workspace**
2. Click **Continue**
3. Click **Skip AWS Setup**
4. Click **Continue**.
5. Select **No reports**, and then click **Continue**
6. Click **Launch monitoring**
7. If prompted, click **Continue with the trial**

Stackdriver will create a new workspace for your project and will collect data for your workspace. You might have to wait for 2 to 3 minutes to see Stackdriver Monitoring welcome page. Without customization, Stackdriver Monitoring automatically discovers your managed services and ingests the metrics. You can customize dashboards, set up alerts, and make other changes.



Stackdriver

qwiklabs-gcp-3f16c1d9250c0804



Monitoring Overview



Resources



Alerting



Uptime Checks



Groups



Dashboards



Debug



Trace



Logging



Error Reporting



Profiler

Monitoring Overview

Welcome

Complete the Sta

[+ Add GCP Project](#)



Install Sta

Collect enha
services, an

[INSTALL AC](#)

Resource da

RESOURCE TYPE

3. Select **Resources > Kubernetes Engine** to view the Kubernetes monitoring dashboard.

You may need to wait for a few minutes for the Kubernetes Engine cluster and its resources to become visible to Stackdriver.



Monitoring Overview



Resources



Alerting



Uptime Checks



Groups



Dashboards



Debug



Trace



Logging



Error Reporting



GKE / 6587584:guestbook-cluster ?



Filter...

Incidents ?

OPEN (0)

ACKNOWLEDGED (0)

RESOLVED



No open incidents

Events ?

No events matching this criteria.

Items per page: 5

GKE Cluster Details

Name	guestbook-cluster
Created	Dec 9, 5:04 pm
Master Version	1.11.2-gke.18
Node Version	1.11.2-gke.18

Task 2. Enable Prometheus Monitoring

In this task, you enable Prometheus monitoring for your Kubernetes Engine cluster.

1. Switch back to your Cloud Shell tab and use gcloud to get the credentials for the Kubernetes Cluster that has been automatically created for you by the lab setup.

```
gcloud container clusters get-credentials guestbook-cluster \
--zone=us-central1-a
```

2. Install role-based access control for the Prometheus agent.

```
kubectl apply -f \
https://storage.googleapis.com/stackdriver-prometheus-documentation/rbac-
setup.yml \
--as=admin --as-group=system:masters
```

3. Install the Prometheus agent.

This command installs the Prometheus agent in the `stackdriver` namespace.

```
export PROJECT_ID=$(gcloud config list --format 'value(core.project)')
curl -s \
https://storage.googleapis.com/stackdriver-prometheus-
documentation/prometheus-service.yml | \
sed -e "s/\\(\\s*_kubernetes_cluster_name:*\\).*/\\1 'guestbook-cluster'/g" | \
sed -e "s/\\(\\s*_kubernetes_location:*\\).*/\\1 'us-central1'/g" | \
sed -e "s/\\(\\s*_stackdriver_project_id:*\\).*/\\1 '${PROJECT_ID}"/g" | \
kubectl apply -f -
```

4. Verify that the Prometheus agent is running.

```
kubectl get pods -n stackdriver
```

Task 3. Expose Prometheus metrics from Spring Boot applications

Spring Boot can expose metrics information through Spring Boot Actuator. Micrometer is the metrics collection facility included in Spring Boot Actuator. Micrometer can expose all the metrics using the Prometheus format.

If you are not using Spring Boot, you can expose JMX metrics through Prometheus by using a [Prometheus JMX Exporter agent](#). In this task, you add the Spring Boot Actuator starter and Micrometer dependencies to the guestbook frontend application.

1. In the Cloud Shell code editor, open `~/guestbook-frontend/pom.xml`.
2. Insert the following new dependencies at the end of the `<dependencies>` section, just before the closing `</dependencies>` tag.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-prometheus</artifactId>
  <scope>runtime</scope>
</dependency>
```

3. In the Cloud Shell code editor, open `~/guestbook-frontend/src/main/resources/application.properties`.
4. Add the following two properties to configure Spring Boot Actuator to expose metrics on port 8081.

```
management.server.port=8081
management.endpoints.web.exposure.include=*
```

Task 4. Rebuild the containers

In this task you rebuild the containers and configure the frontend container deployment to expose the prometheus monitoring endpoint.

1. In the Cloud Shell change to the frontend application directory.

```
cd ~/guestbook-frontend
```

2. Build the frontend application container.

```
./mvnw clean compile jib:build
```

3. While this is compiling switch back to the Cloud Shell code editor and open `~/kubernetes/guestbook-frontend-deployment.yaml`.

You update the Kubernetes deployment to specify the Prometheus metrics endpoint. With this configuration, Spring Boot Actuator exposes the Prometheus metrics on port 8081, under the path of `/actuator/prometheus`.

You add Prometheus annotations to the `deployment.spec.template.metadata.annotation` section of the build YAML file for the frontend application.

4. Insert the new annotations definition section into the metadata section for the deployment as shown in the screenshot below.

```
annotations:
  prometheus.io/scrape: 'true'
  prometheus.io/path: '/actuator/prometheus'
  prometheus.io/port: '8081'
```

The `guestbook-frontend-deployment.yaml` file should now look like the following screenshot:

```

1  apiVersion: v1
2  kind: Service
3  metadata:
4    labels:
5      app: guestbook-frontend
6      name: guestbook-frontend
7  spec:
8    type: LoadBalancer
9    ports:
10   - name: http
11     port: 8080
12     targetPort: 8080
13   selector:
14     app: guestbook-frontend
15   ---
16  apiVersion: apps/v1
17  kind: Deployment
18  metadata:
19    labels:
20      app: guestbook-frontend
21      name: guestbook-frontend
22  spec:
23    replicas: 2
24    selector:
25      matchLabels:
26        app: guestbook-frontend
27    template:
28      metadata:
29        labels:
30          app: guestbook-frontend
31      annotations:
32        prometheus.io/scrape: 'true'
33        prometheus.io/path: '/actuator/prometheus'
34        prometheus.io/port: '8081'
35    spec:
36      volumes:
37        - name: credentials
38          secret:
39            secretName: guestbook-service-account
40

```

Note

Whitespace is significant in YAML file layouts. The layout of the changes you make must match the screenshot.

5. When the frontend application build has completed in the Cloud Shell change to the guestbook backend service application directory.

```
cd ~/guestbook-service
```

6. Build the guestbook backend service application container.

```
./mvnw clean compile jib:build
```

Note

You haven't made any changes to the backend service application but you have to build the image so that it is available on the gcr.io container repository for the lab when you deploy the full application.

Task 5. Deploy the Prometheus enabled application to Kubernetes Engine

In this task you deploy the application components to Kubernetes Engine and monitor the Prometheus metrics using Stackdriver.

1. In the Cloud Shell deploy the configuration.

```
kubectl apply -f ~/kubernetes
```

This deploys both containerized applications. It takes a few minutes for the Prometheus metrics to be fetched and displayed in Stackdriver Monitoring. While waiting, you use `curl` to verify that the Prometheus endpoint exists.

2. Switch back to the Cloud Shell and find the pod names.

```
kubectl get pods -l app=guestbook-frontend
```

Repeat the command until you see that the frontend pods are running. That should only take a few seconds.

NAME	READY	STATUS	RESTARTS	AGE
guestbook-frontend-8567fdc8c8-c68vk	1/1	Running	0	5m
guestbook-frontend-8567fdc8c8-gvcf5	1/1	Running	0	5m

3. To establish port forwarding to one of the guestbook frontend application pods replace `[*POD-NAME*]` with one of the pod names from the previous command in the following command:

```
kubectl port-forward [POD-NAME] 8081:8081
```

If you have selected a valid pod name this will run in a loop forwarding local traffic on port 8081 into that pod.

4. Open a new Cloud Shell tab and use `curl` to verify that the Prometheus endpoint exists.

```
curl http://localhost:8081/actuator/prometheus
```

The output is similar to the following example:

```
...
# TYPE jvm_memory_committed_bytes gauge
jvm_memory_committed_bytes{area="nonheap",id="Code Cache",} 1.8284544E7
jvm_memory_committed_bytes{area="nonheap",id="Metaspace",} 6.6281472E7
...
# HELP spring_integration_handlers The number of message handlers
# TYPE spring_integration_handlers gauge
spring_integration_handlers 3.0
```

These metrics are available in Stackdriver Monitoring for visualization, building dashboards, and setting up alerts. Some of these metrics (for

example, `jvm_memory_committed_bytes`) have multiple dimensions (for example, `area` and `id`). These dimensions are filterable and groupable in Stackdriver Monitoring too.

5. Navigate to the Stackdriver Monitoring console.
Refresh the page if the page is already open.

6. Select **Resources > Metrics Explorer**.


7. In the Metrics Explorer, search for `jvm_memory` to find metrics collected by the Prometheus agent from the Spring Boot application.

Metrics Explorer

METRIC

VIEW OPTIONS



Find resource type and metric 

jvm_memory|

Metrics



external/prometheus/**jvm_memory**... k8s_container
external.googleapis.com/prometheus/**jvm_memor**...

external/prometheus/**jvm_memory**... k8s_container
external.googleapis.com/prometheus/**jvm_memor**...

external/prometheus/**jvm_memory**... k8s_container
external.googleapis.com/prometheus/**jvm_memor**...

8. Select `jvm_memory_used_bytes` to plot the metrics.

The JVM memory has multiple dimensions (for example, Heap versus Non-Heap and Eden Space versus Metaspace).

9. In **Filter**, filter by **area=heap**, in **Group By**, filter by **pod_name**, and in **Aggregation**, select **sum**.

These options build a graph of current heap usage of the frontend application.

Metrics Explorer

METRICVIEW OPTIONS

Find resource type and metric ?

Resource type:Kubernetes Container

Metric:external/prometheus/...

Filter ?

area = "heap" + Add a filter

Group By ?

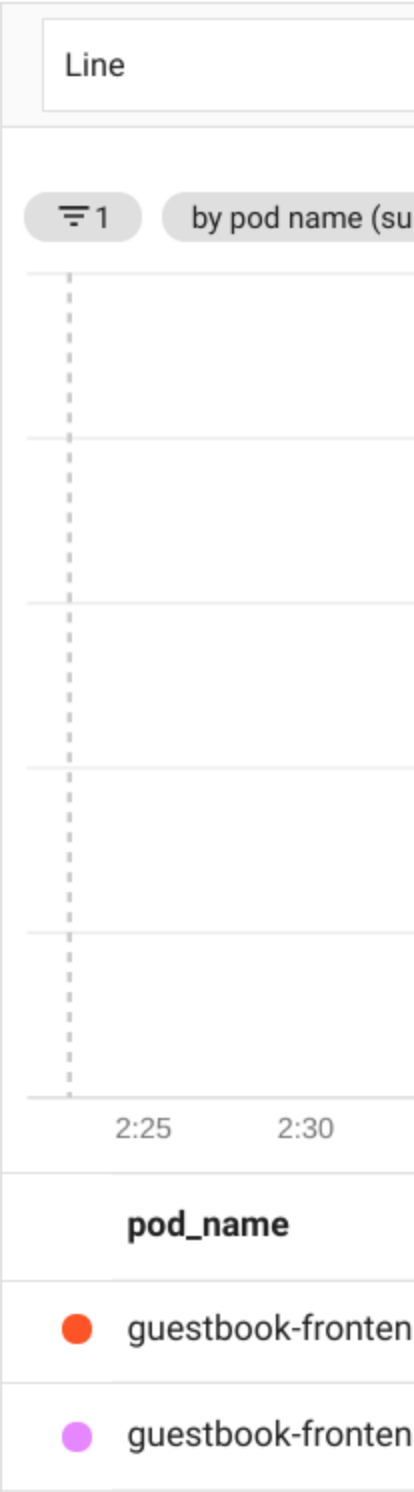
pod_na... + Add a filter

Aggregator ?

sum

SHOW ADVANCED OPTIONS

+ ADD METRIC



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'll 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 your rating:

- 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, use the **Support** tab.

Copyright 2019 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.