JAVAMS10 Debugging with Stackdriver Debugger

2 hours3 Credits

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 repackaged and then deployed the demo application to App Engine. In this lab, you configure Stackdriver Logging, Stackdriver Debugger, and Stackdriver Monitoring. During the lab you use Stackdriver to inspect logs, and debug and monitor the performance of the demo application while it is running on App Engine.

Stackdriver Debugger is a feature of Google Cloud Platform (GCP) that enables you to inspect the state of an application at any code location, without stopping or

slowing down the running app. Stackdriver Debugger makes it easier to view the application state without adding logging statements.

You can use Stackdriver Debugger with any deployment of your application, including test, development, and production. Stackdriver Debugger adds less than 10 milliseconds to the request latency only when the application state is captured. In most cases, this additional latency is not noticeable by users.

Objectives

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

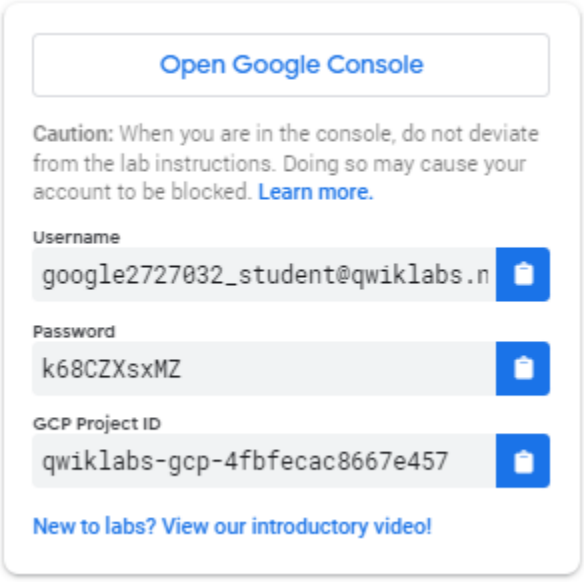
- Configure Stackdriver Logging for an App Engine application
- Configure Stackdriver Debugger source content for App Engine debugging
- Configure Stackdriver Debugger logpoints and snapshots
- Enable Stackdriver Monitoring

Task 0. Lab Preparation

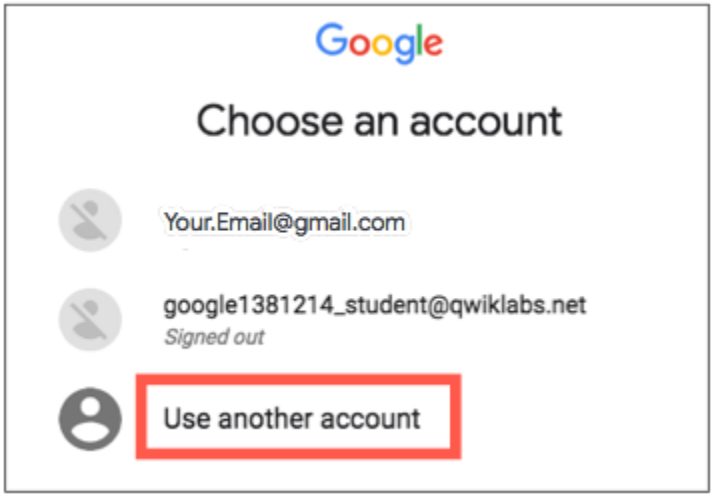
Access Qwiklabs

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.



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.

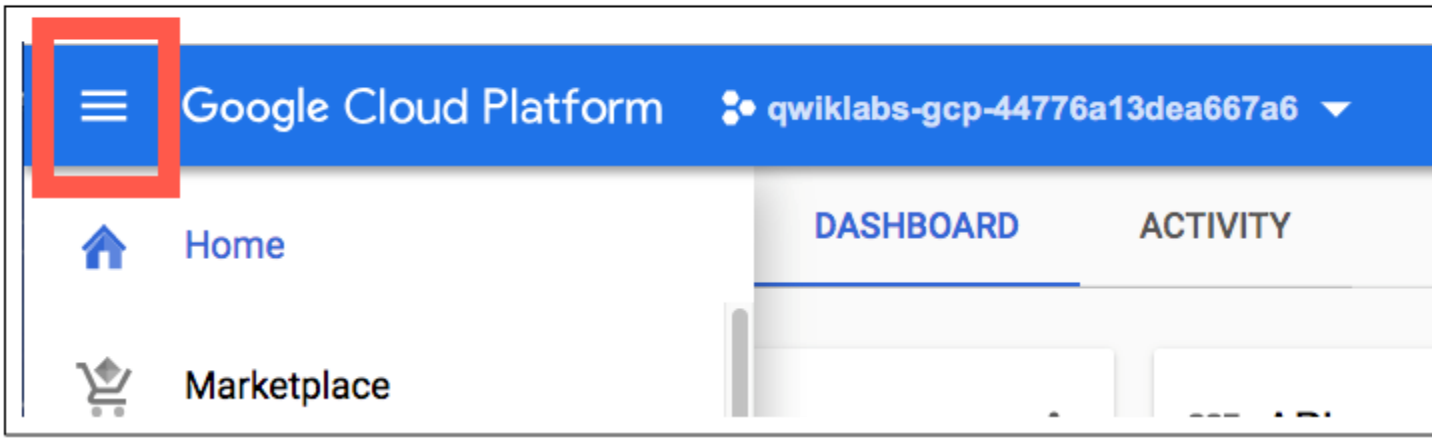
Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own GCP account, do not use it for this lab (avoids incurring charges).

5. Click through the subsequent pages:

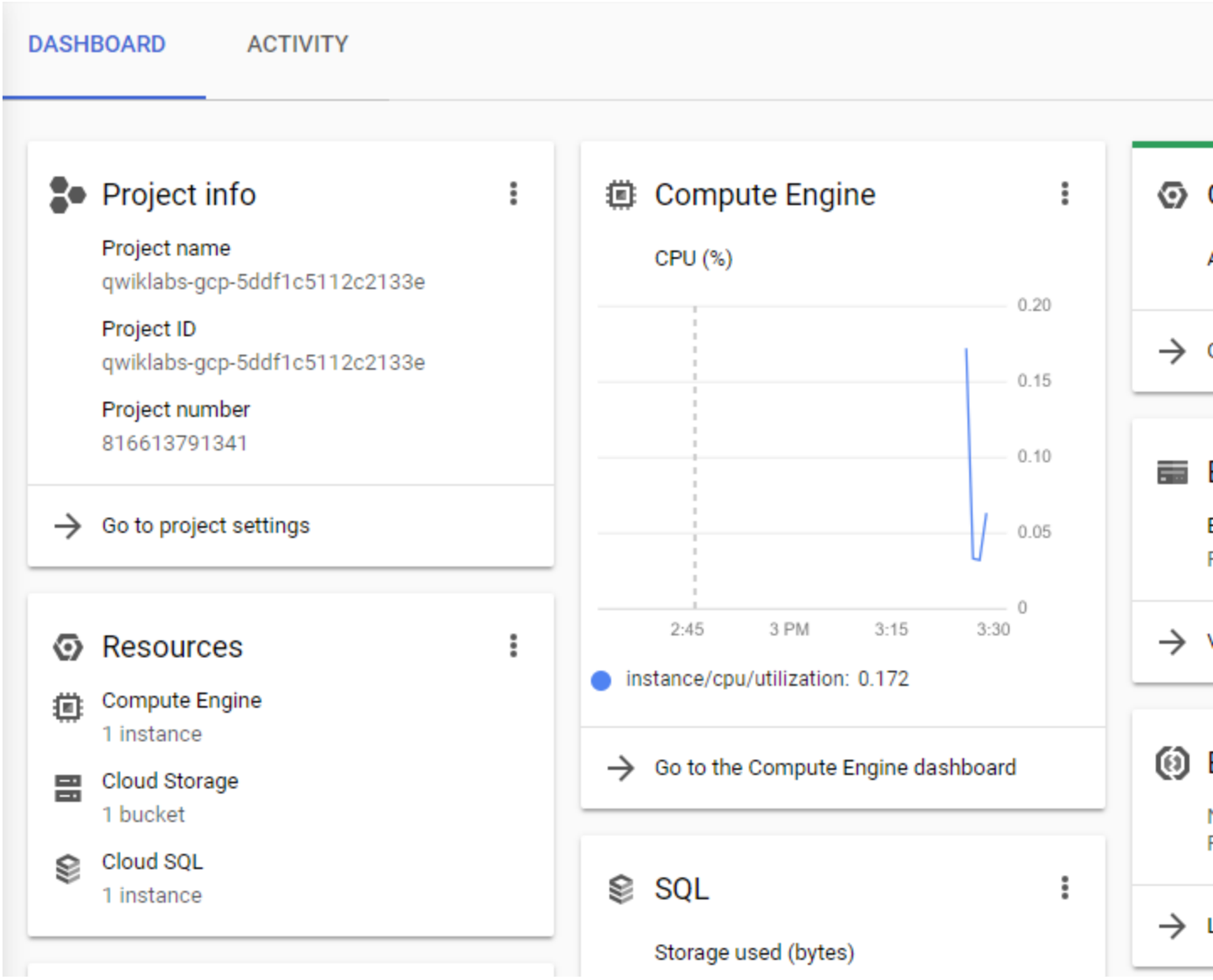
    - Accept the terms and conditions.
    - Do not add recovery options or two-factor authentication (because this is a temporary account).
    - Do not sign up for free trials.

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.

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 once the Cloud SQL server is ready and both application microservices components have been deployed to App Engine. You might have to wait up to 10 minutes for the deployment tasks 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. Check that the frontend application is running.

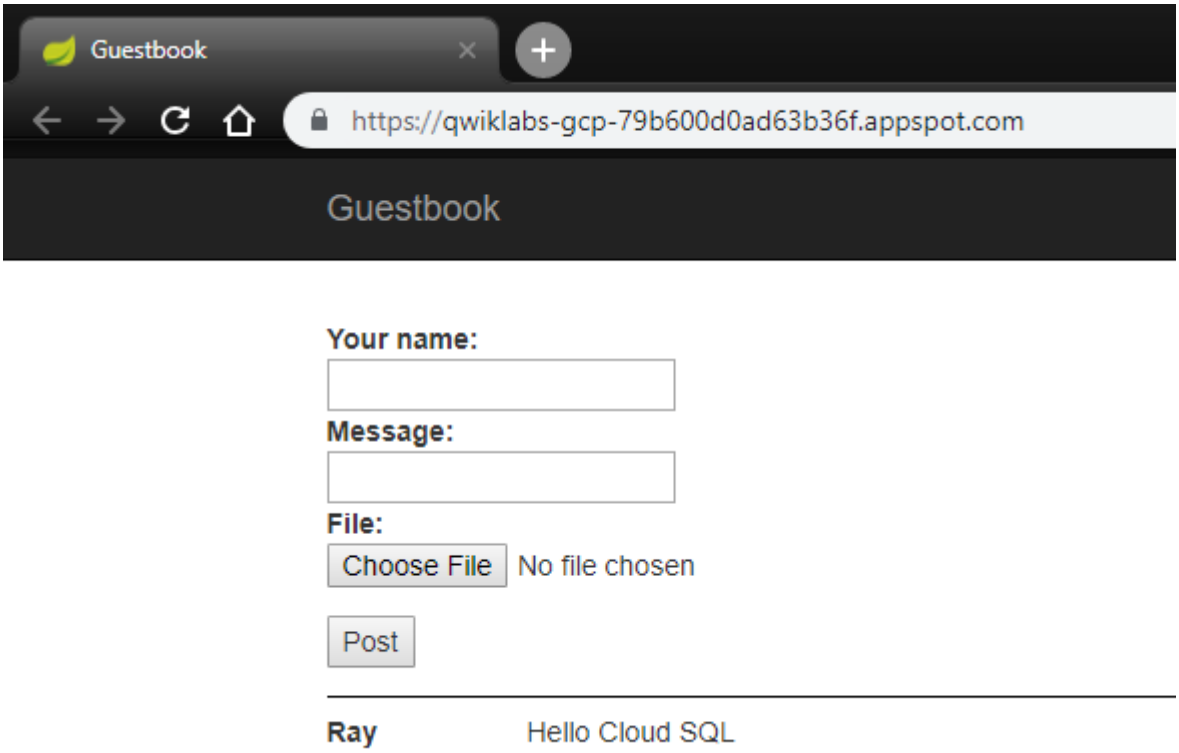
8. Find the URL of the frontend application that should now be running on App Engine

```
gcloud app browse
```

This command reports a URL that links to your application's frontend.

```
Did not detect your browser. Go to this link to view your app:
https://....appspot.com
```

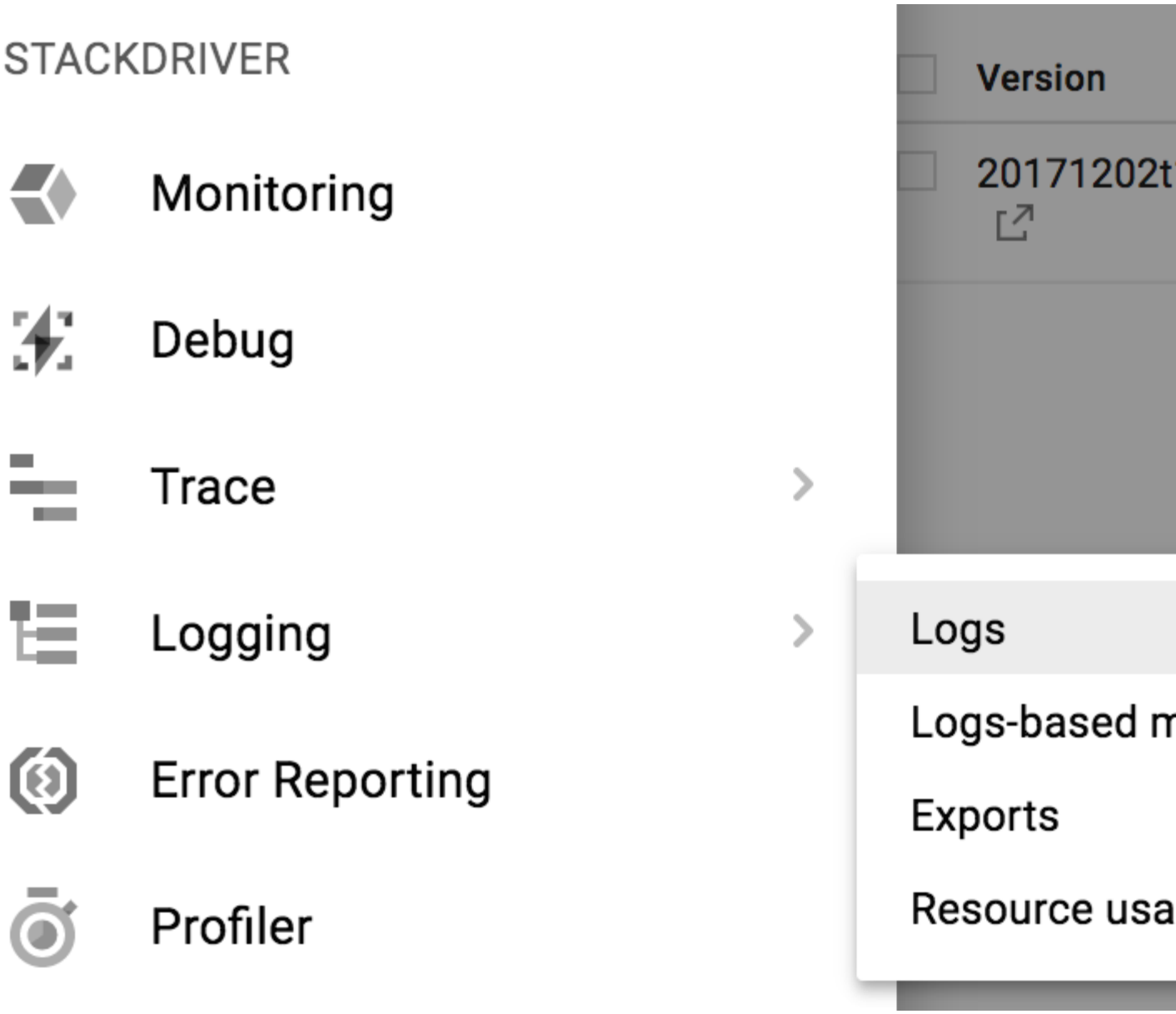
9. Click the link to open a browser tab to the frontend URL.



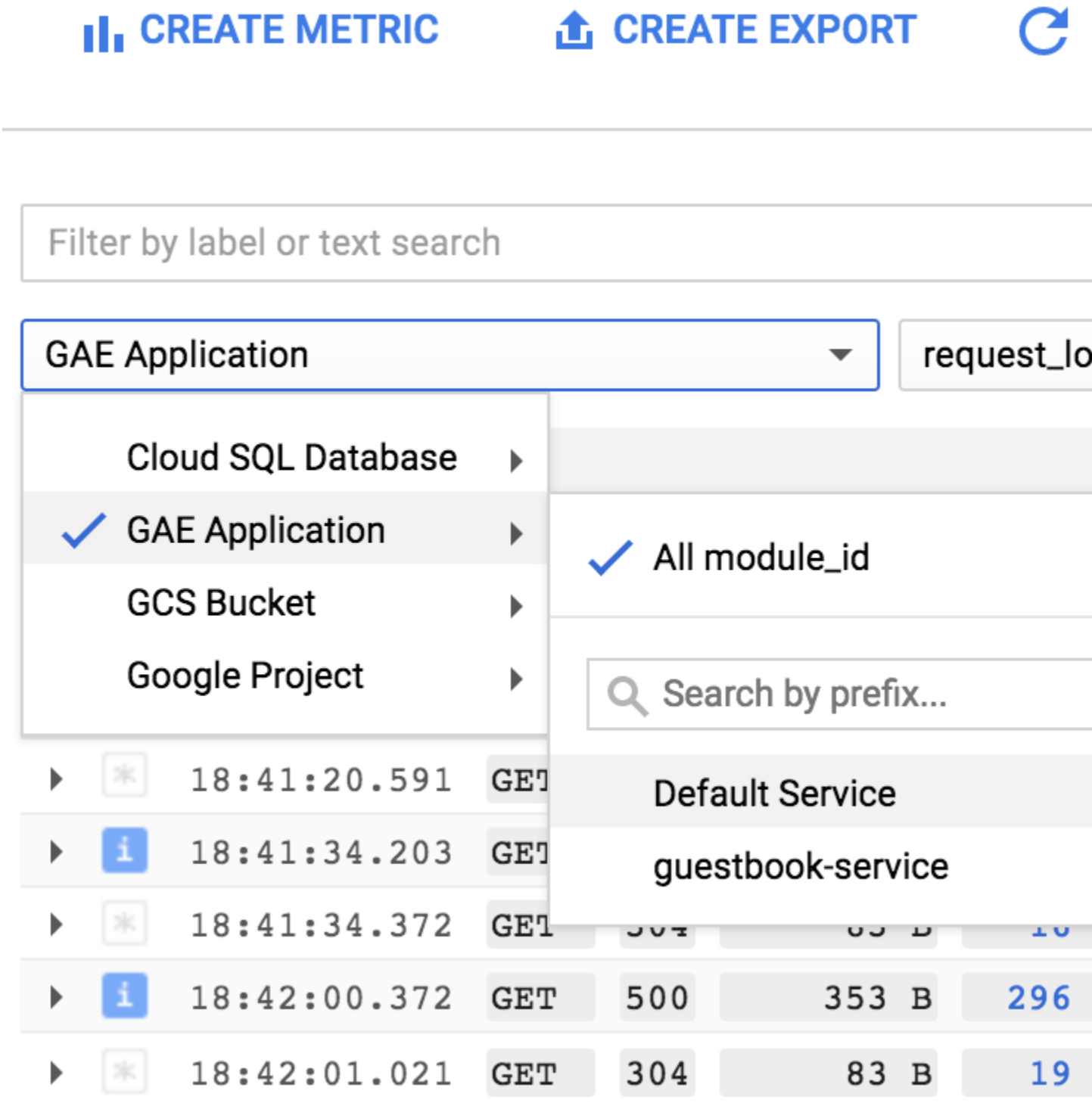
Task 1. Examine Stackdriver logs

In this task, you examine Stackdriver logs for the demo application running on App Engine.

1. Open a new browser tab and navigate to the GCP console.
2. Navigate to **Stackdriver > Logging > Logs**.



3. In the log drop-down list, select **GAE Application > Default Service > 1 (100%)**.



Note
You might have to enter **1** for **Search by prefix** to select the **1 (100%)** option.

The default App Engine application log is displayed. When you output a log message, it is grouped by the request. When the application first starts, the log messages are grouped under `/_ah/start` request.

4. Expand an entry to view the detailed log entry.

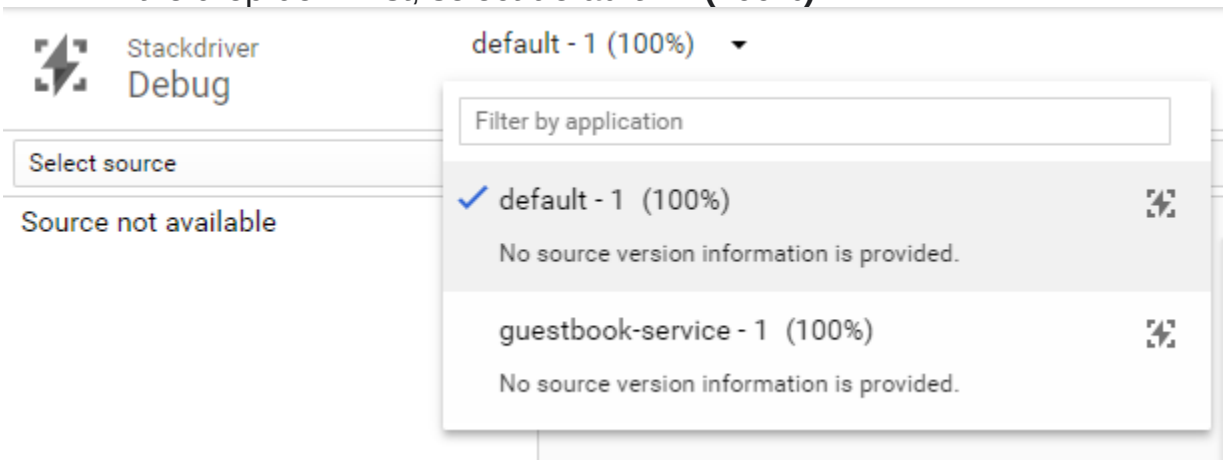
Task 2. Configure Stackdriver Debugger

In this task, you configure Stackdriver Debugger so that it can be used to debug the demo microservices application used in this set of labs. The demo application was automatically deployed to App Engine for you as part of the lab setup.

1. Navigate to **Stackdriver > Debug**.

At the top, the running App Engine deployments are listed.

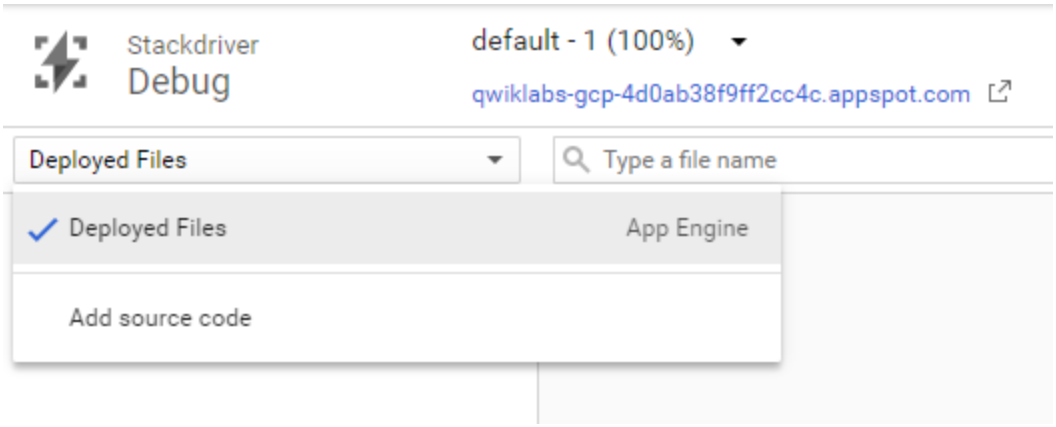
2. In the drop-down list, select **default - 1 (100%)**.



Default - 1 is the frontend application. The source code is not yet available for debugging.

3. Click **Select Source** on the left of the screen to expand it and select **Deployed Files**.This is just below **Stackdriver Debug.**

You can provide source code to Stackdriver Debugger in several ways.



4. Expand the **Alternative source code** drop-down list.

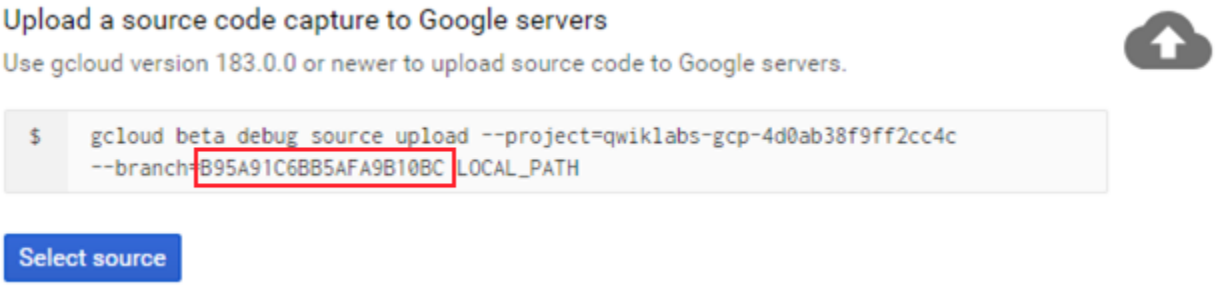
A list of methods for providing source code is displayed. In this lab, you use the Google Source Repositories service.

5. Scroll down to the **Upload a source code capture to Google servers** section.

You use the command line to upload your source code.

6. Copy the branch ID that is listed in this command line and save it to a local text file.

You use the branch ID in a later step.



7. Switch to Cloud Shell and enable the Google Cloud Source Repository API.

```
gcloud services enable sourcerepo.googleapis.com
```

8. Create a source repository for source capture.

```
gcloud source repos create google-source-captures
```

9. Change to the frontend application directory.

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

10. Configure the `git` email and username properties.

These properties are used for the code upload.

```
git config --global user.email $(gcloud config get-value core/account)
git config --global user.name "devstar"
```

11. In Cloud Shell, upload the current `guestbook-frontend` source tree as a branch, using the branch ID that you saved earlier in a previous step.

Warning

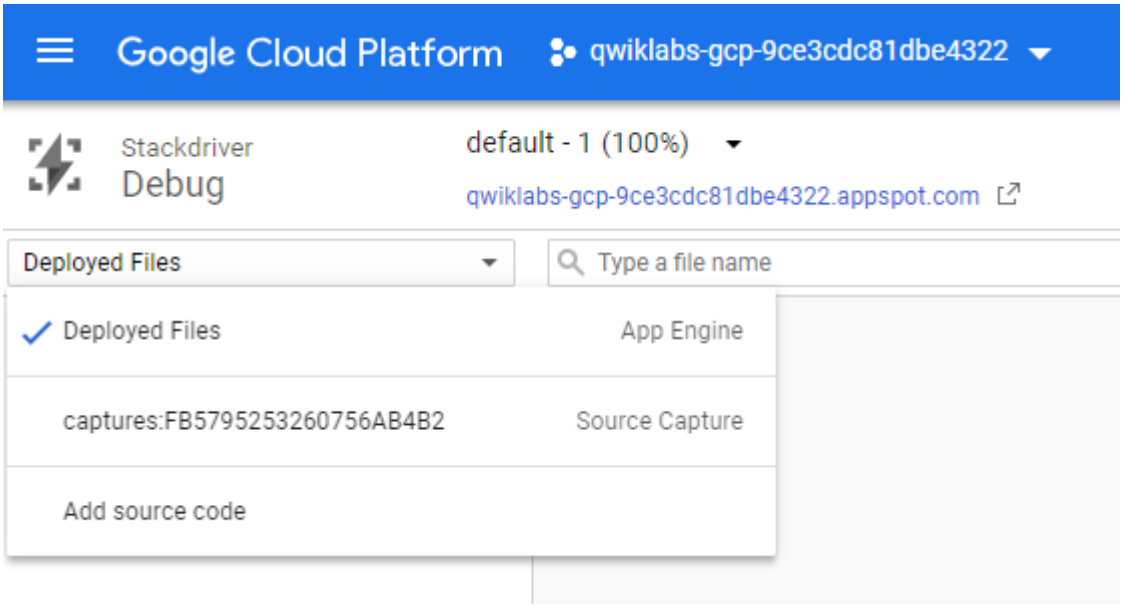
Before running the following command, you must replace `[CAPTURE_BRANCH_ID]` with the branch ID that you recorded in a previous step.

```
gcloud beta debug source upload --project=$PROJECT_ID \
 --branch=[CAPTURE_BRANCH_ID] ./src/
```

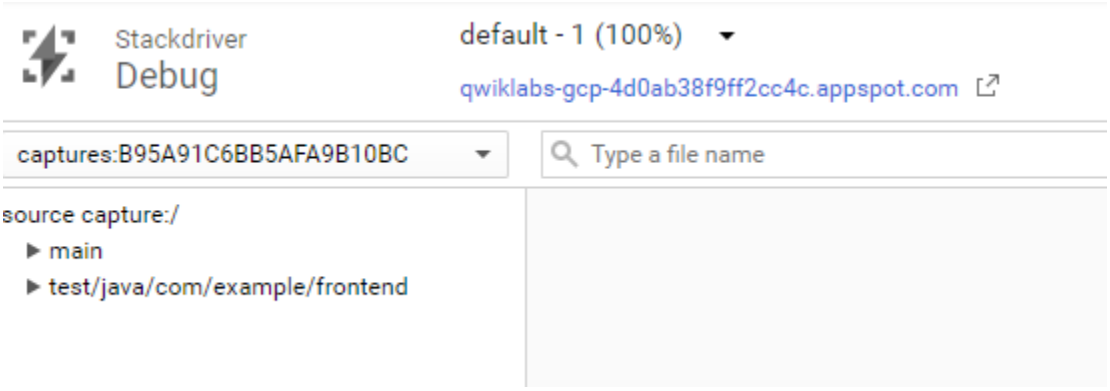
Task 3. Use Stackdriver Debugger to debug an application

In this task, you use Stackdriver Debugger to debug the demo application running on App Engine.

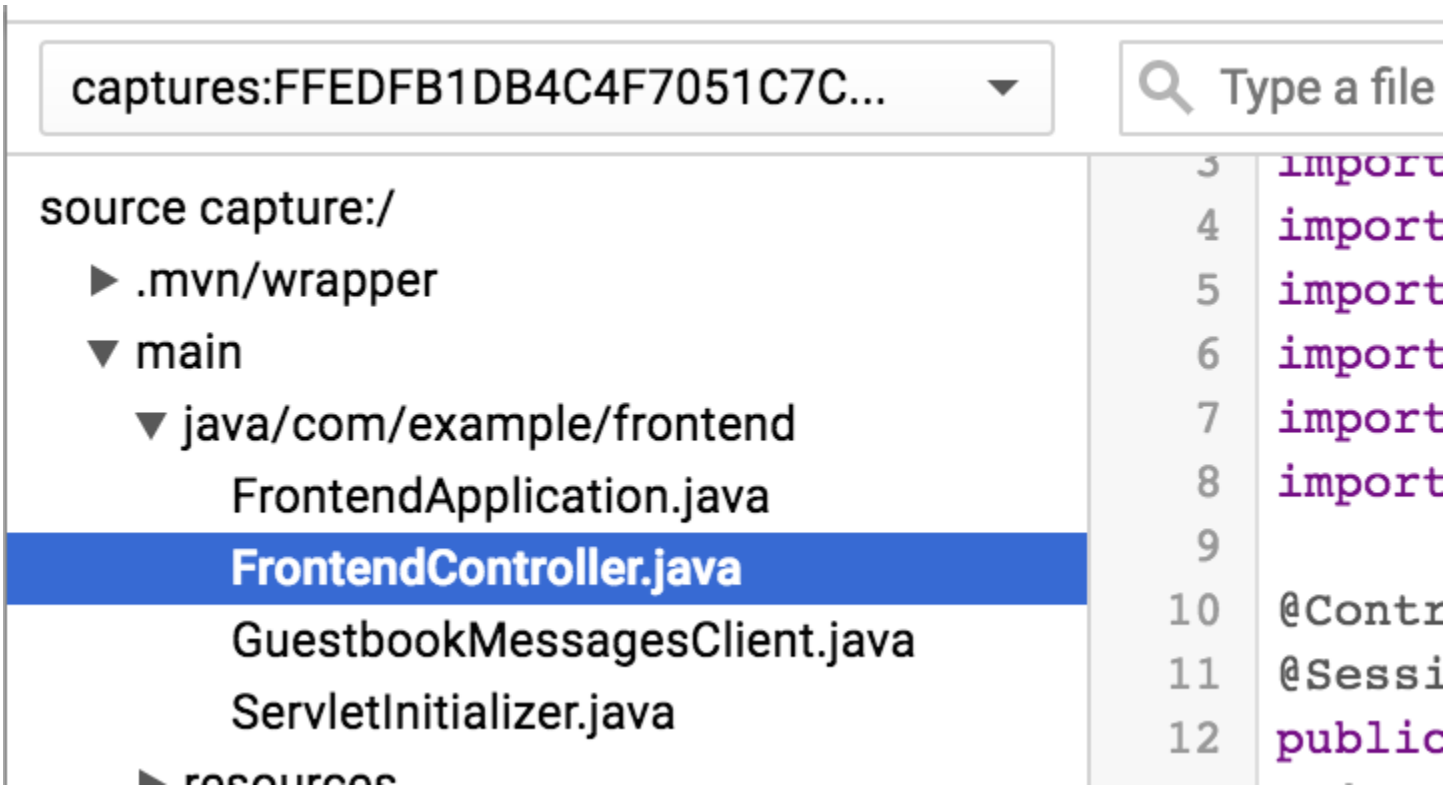
1. Return to the Stackdriver Debugger console and refresh the page.
2. In the **Upload a source code capture to Google servers** section, click **Deployed Files** to expand the selection list and then select the newly uploaded source capture.



The source code tree for the frontend application now appears in the left navigation pane.

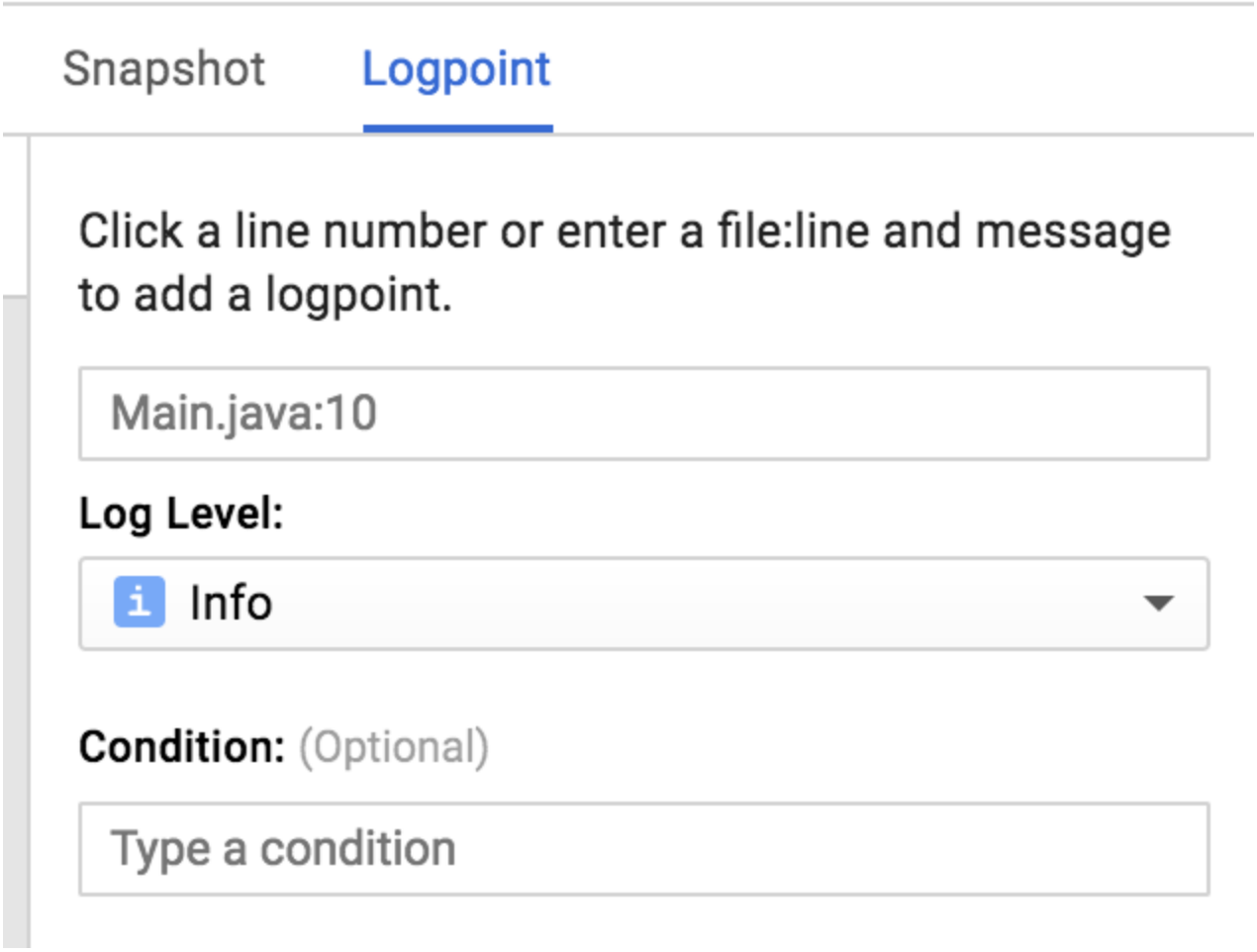


3. Navigate the source and open `FrontendController.java`.



From here, you have a significant amount of control. For example, you can add a log message.

4. On the right side of the window, click **Logpoint**.



5. In the source, click the line number where you want to add a log message, and edit the message to print the text and variables that you want to see.

In this example, the message is changed to print the text `Variable name =`, followed by the value of the local variable `name`.

```
78  @GetMapping("/")
79  public String index(Model model) {
80    if (model.containsAttribute("name")) {
81      String name = (String) model.asMap().get("name");
82      model.addAttribute("greeting", String.format("%s %s", greeting, name));
83    }
84    model.addAttribute("messages", client.getMessages().getContent());
85    return "index";
86  }
```

if (true) logpoint("Variable Name = {name}") Info Cancel Add

6. Click **Add**.

You can add as many log messages as you want.

```
79  public String index(Model model) {
80    if (model.containsAttribute("name")) {
81      String name = (String) model.asMap().get("name");
      logpoint("Variable name = {name}")
82      model.addAttribute("greeting", String.format("%s %s", greeting, name))
83    }
      logpoint("I'm here!")
84    model.addAttribute("messages", client.getMessages().getContent());
85    return "index";
86  }
```

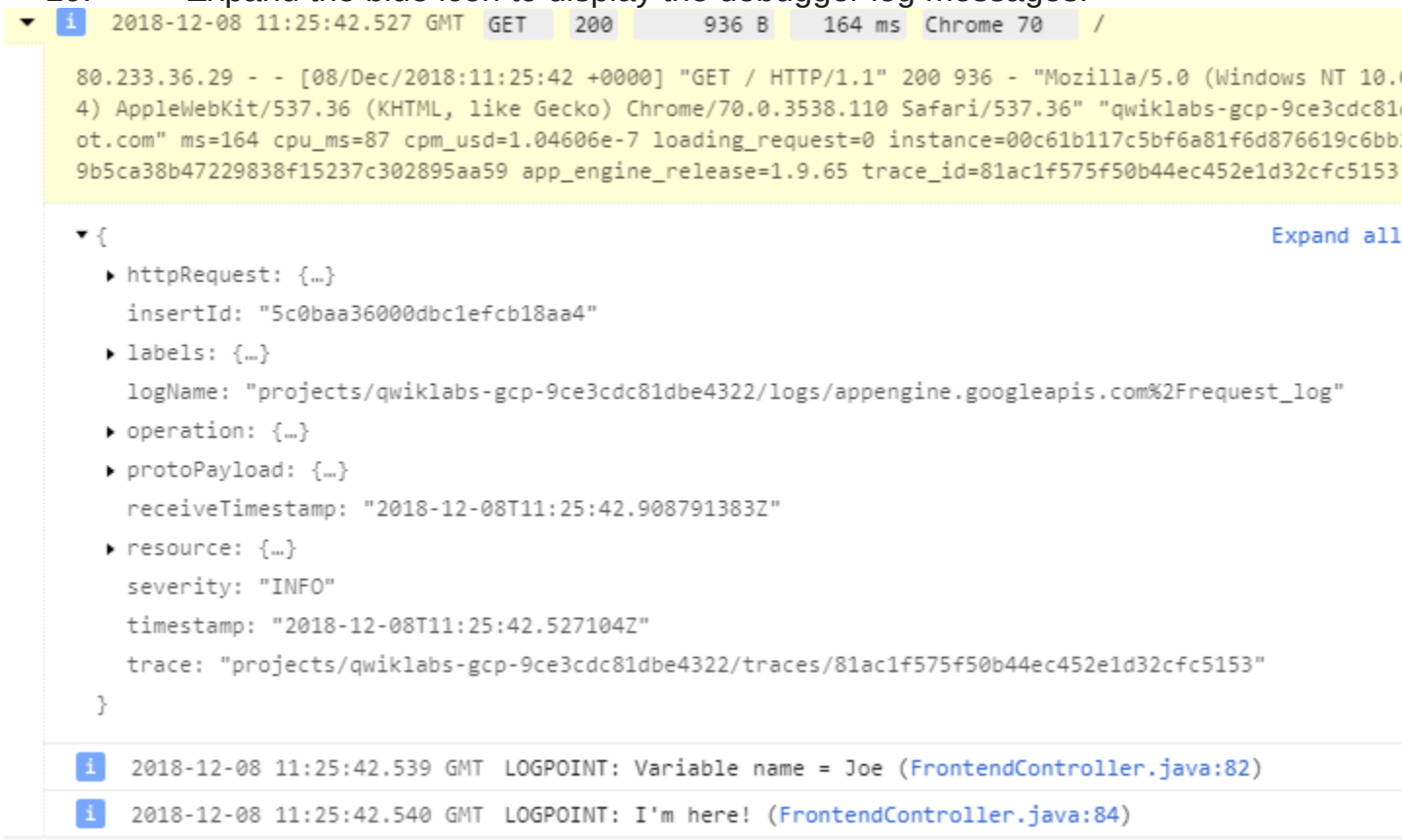
7. Open the Guestbook application tab in your browser, and enter a name and message to trigger the code.
8. Return to the Stackdriver Debugger console and navigate to **Stackdriver > Logging**.
9. Find the most recent HTTP request that has a blue information icon.

This request is close to the end.

2018-10-23 22:02:01.221 BST	GET	200	1.07 KB	35 ms	Chrome 69	/favicon.ico
2018-10-23 22:03:02.870 BST	POST	302	149 B	547 ms	Chrome 69	/post
2018-10-23 22:03:03.671 BST	GET	200	1.21 KB	637 ms	Chrome 69	/
2018-10-23 22:03:04.481 BST	GET	200	166.31 KB	491 ms	Chrome 69	/image/69057ecb-89ef-4
2018-10-23 22:03:04.482 BST	GET	200	21.48 KB	466 ms	Chrome 69	/image/726654ea-ee08-4
2018-10-23 22:03:04.487 BST	GET	200	339.21 KB	583 ms	Chrome 69	/image/6903e151-05b9-4

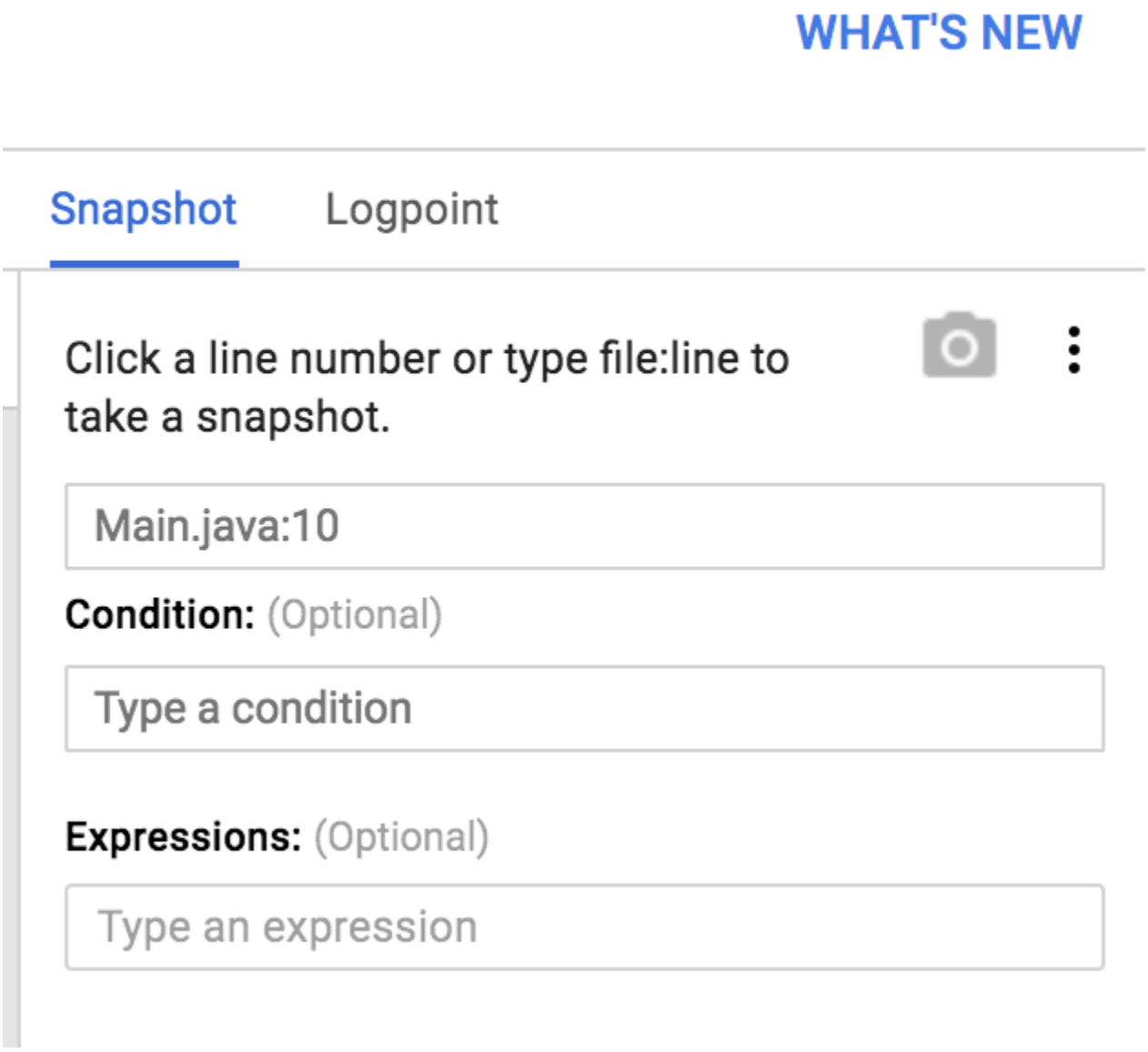
No newer entries found matching current filter. Load newer logs

10. Expand the blue icon to display the debugger log messages.



You can also capture the stack in a moment in time. It is almost like stepping through a real debugger, but it does not stop the application for your users.

11. Return to **Stackdriver > Debug**.
12. in the source view on the right side, click **Snapshot**.

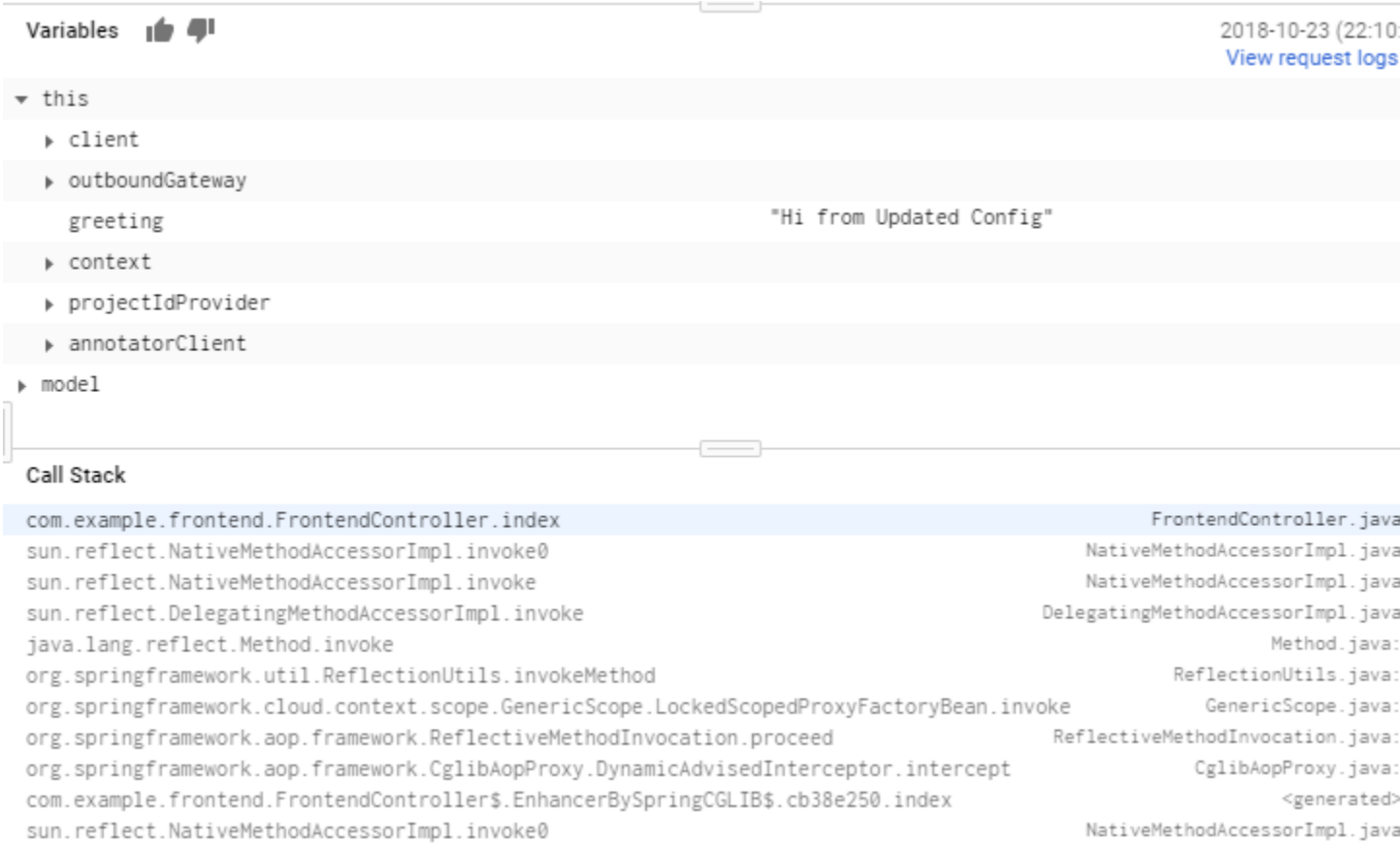


13. In the source, click the line number where you want to capture information.

```
78  @GetMapping("/")
79  public String index(Model model) {
80    if (model.containsAttribute("name")) {
81      String name = (String) model.asMap().get("name");
      logpoint("Variable name = {name}")
82      model.addAttribute("greeting", String.format("%s %s", greeting, name));
83    }
    logpoint("I'm here!")
84    model.addAttribute("messages", client.getMessages().getContent());
85    return "index";
86  }
87
```

14. Switch to the demo application and post another guestbook message.

As soon as a request flows through the line, the call stack is captured, and you can explore the internal state of the application at that point in time.You can add conditionals to both logpoints and snapshots, so that you view only certain requests based on variables that are in scope (for example, session ID).



Note

Stackdriver Debugger works with various languages, and also outside of App Engine. You can also debug your application in the same way when you deploy your application on-premises, in a VM, or in containers.

Task 4. Enable Stackdriver Monitoring

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

1. In your GCP console navigate to **Stackdriver > Monitoring** in the Navigation menu.

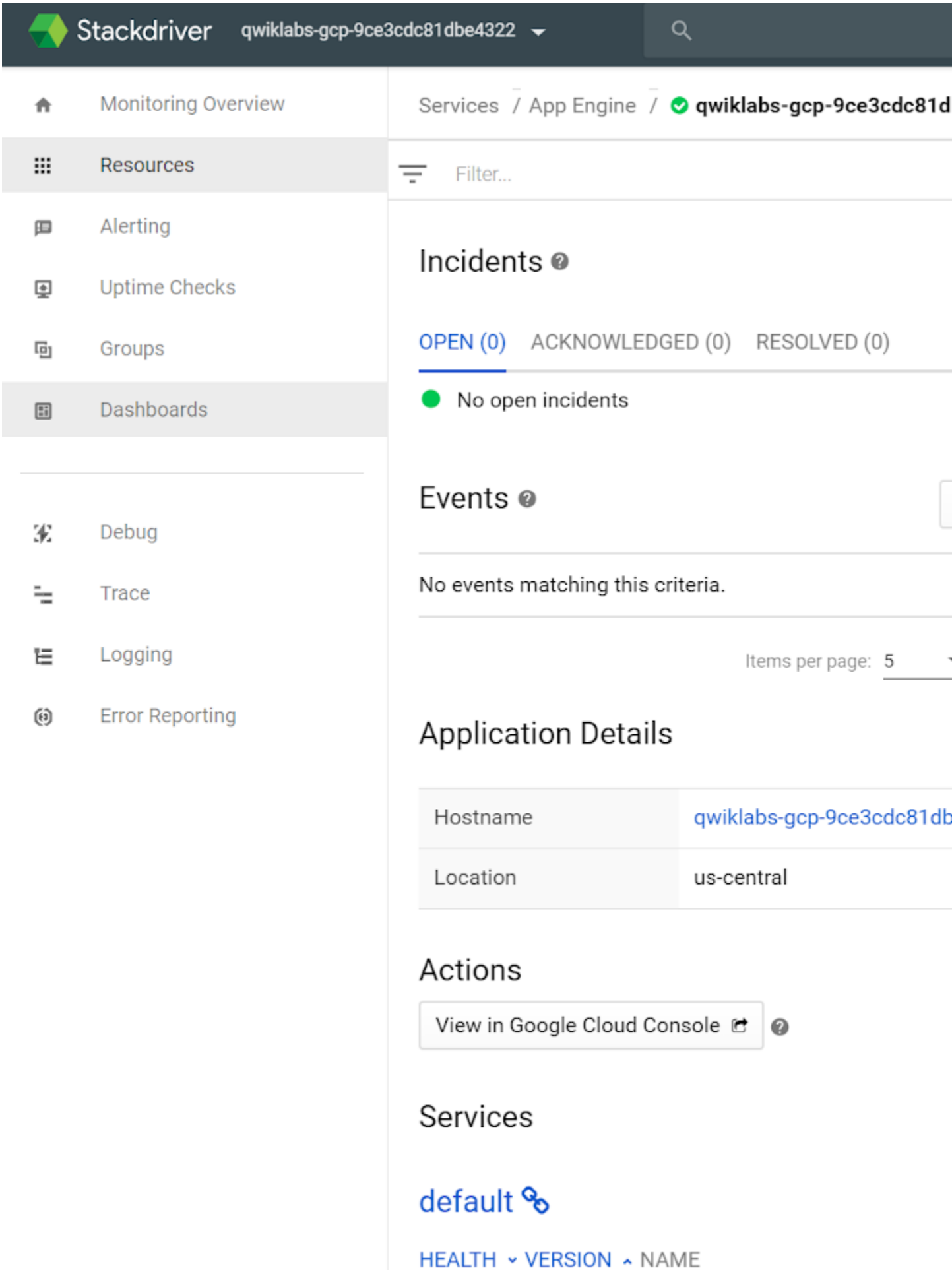
You use the wizard to set up Stackdriver Monitoring.

2. Click **Create Workspace**.
3. When prompted to add Google Cloud Platform projects to monitor, the project ID for your Qwiklabs session should be the only option listed. Do not select additional projects and click **Continue**.
4. When prompted to monitor AWS accounts, click **Skip AWS Setup**.
5. When prompted to install the Stackdriver agents, click **Continue**.
6. When prompted to get reports by email, select **No reports** and click **Continue**.
7. Click **Launch monitoring**.

This button takes a minute or two to appear. Without customization, Stackdriver Monitoring automatically discovers your managed services and ingests the metrics. You can customize dashboards, set up alerts, and make other changes.

8. Navigate to **Resources > GCP > App Engine**.

After a minute or two, an overview dashboard of your App Engine services appears. You might have to refresh the page.


Stackdriver
qwiklabs-gcp-9ce3cdc81dbe4322
Monitoring Overview
Resources
Alerting
Uptime Checks
Groups
Dashboards
Debug
Trace
Logging
Error Reporting
Services / App Engine / qwiklabs-gcp-9ce3cdc81d
Filter...
Incidents
OPEN (0)
ACKNOWLEDGED (0)
RESOLVED (0)
No open incidents
Events
No events matching this criteria.
Items per page: 5
Application Details
Hostname
qwiklabs-gcp-9ce3cdc81db
Location
us-central
Actions
View in Google Cloud Console
Services
default
HEALTH
VERSION
NAME

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.