**Monitoring Applications in Google Cloud**

experimentLabschedule1 hour 30 minutesuniversal\_currency\_alt5 Creditsshow\_chartIntroductory

infoThis lab may incorporate AI tools to support your learning.

**Overview**

In this lab, you will deploy an application to Google Cloud and then use the tools provided by Google Cloud to monitor it. You will use Cloud Logging, Trace, Profiler, and dashboards and create uptime checks and alerting policies.

**Objectives**

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

* Download a sample app from Github
* Deploy an application to App Engine
* Examine the Cloud logs
* View Profiler information
* Explore Cloud Trace
* Monitor resources using dashboards
* Create uptime checks and alerts

**Setup**

For each lab, you get a new Google Cloud project and set of resources for a fixed time at no cost.

1. Sign in to Qwiklabs using an **incognito window**.
2. Note the lab's access time (for example, 1:15:00), and make sure you can finish within that time.  
   There is no pause feature. You can restart if needed, but you have to start at the beginning.
3. When ready, click **Start lab**.
4. Note your lab credentials (**Username** and **Password**). You will use them to sign in to the Google Cloud Console.
5. Click **Open Google Console**.
6. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.  
   If you use other credentials, you'll receive errors or **incur charges**.
7. Accept the terms and skip the recovery resource page.

**Note:** Do not click **End Lab** unless you have finished the lab or want to restart it. This clears your work and removes the project.

**Task 1. Download a sample app from Github**

Download a sample application from GitHub and preview it in Cloud Shell.

1. In the Cloud Console, click **Activate Cloud Shell** (Cloud Shell icon).
2. If prompted, click **Continue**. Once connected to Cloud Shell, you should see that you are already authenticated and that the project is already set to your project ID.
3. Run the following command in Cloud Shell to confirm that you are authenticated. If prompted, click **Authorize**:

gcloud auth list

Copied!

content\_copy

**Command output:**

Credentialed Accounts

ACTIVE ACCOUNT

\* [my\_account]@[my\_domain.com]

To set the active account, run:

$ gcloud config set account `ACCOUNT`

**Note:** The gcloud command-line tool is the powerful and unified command-line tool in Google Cloud. It comes preinstalled in Cloud Shell. Among its features, gcloud offers tab completion in the shell. For more information, see [gcloud command-line tool overview](https://cloud.google.com/sdk/gcloud" \t "_blank).

1. Run the following command to confirm that you are using the correct project for this lab:

gcloud config list project

Copied!

content\_copy

**Command output**

[core]

project = [PROJECT\_ID]

1. If the correct project is not listed, you can set it with this command:

gcloud config set project [PROJECT\_ID]

Copied!

content\_copy

**Command output:**

Updated property [core/project].

1. To create a folder called gcp-logging, run the following command:

mkdir gcp-logging

Copied!

content\_copy

1. Change to the folder you just created:

cd gcp-logging

Copied!

content\_copy

1. Clone a simple Python Flask app from Github:

git clone https://GitHub.com/GoogleCloudPlatform/training-data-analyst.git

Copied!

content\_copy

1. Change to the deploying-apps-to-gcp folder:

cd training-data-analyst/courses/design-process/deploying-apps-to-gcp

Copied!

content\_copy

1. In Cloud Shell, click **Open Editor** ().
2. Expand the **gcp-logging/training-data-analyst/courses/design-process/deploying-apps-to-gcp** folder in the navigation pane, and then click **main.py** to open it.
3. Add the following import statement at the top of the file (*line 2*):

import googlecloudprofiler

Copied!

content\_copy

**Note:** Profiler allows you to monitor the resources your applications use. For more information, refer to the [Google Cloud Profiler documentation](https://cloud.google.com/profiler/docs/).

1. After the **main()** function, add the following code snippet to start Profiler (*after line 11*):

try:

googlecloudprofiler.start(verbose=3)

except (ValueError, NotImplementedError) as exc:

print(exc)

Copied!

content\_copy

Profiler will continuously report application metrics. Your code should look like this:

**Note:**This code simply turns Profiler on. Once on, Profiler starts reporting application metrics to Google Cloud.

1. You also have to add the Profiler library to your requirements.txt file. Open that file in the code editor and add the following:

google-cloud-profiler==3.0.6

protobuf==3.20.1

Copied!

content\_copy

The file should look like this:

1. Profiler has to be enabled in the project. In Cloud Shell, enter the following command:

gcloud services enable cloudprofiler.googleapis.com

Copied!

content\_copy

1. To test the program, enter the following command to build a Docker container of the image:

docker build -t test-python .

Copied!

content\_copy

1. To run the Docker image, enter the following command:

docker run --rm -p 8080:8080 test-python

Copied!

content\_copy

1. To see the program running, click **Web Preview** (Web Preview icon) in the Google Cloud Shell toolbar. Then select **Preview on port 8080**.

The program should be displayed in a new browser tab.

1. In Cloud Shell, type Ctrl+C to stop the program.

Click *Check my progress* to verify the objective.

Assessment Completed!

Enable the Profiler

Check my progress

*Assessment Completed!*

**Task 2. Deploy an application to App Engine and examine the Cloud logs**

Now you will deploy the program to App Engine and use Google Cloud tools to monitor it.

1. In the Cloud Shell code editor, in the Explorer pane, select the **gcp-logging/training-data-analyst/courses/design-process/deploying-apps-to-gcp** folder.
2. Click **New File**, and then name the file **app.yaml**.
3. Paste the following into the file you just created:

runtime: python39

Copied!

content\_copy

1. Save your changes.
2. In a project, an App Engine application has to be created. This is done just once using the gcloud app create command and specifying the region where you want the app to be created. In Cloud Shell, type the following command:

gcloud app create --region=us-west1

Copied!

content\_copy

1. Now deploy your app with the following command:

gcloud app deploy --version=one --quiet

Copied!

content\_copy

**Note:**This command will take a couple of minutes to complete. Wait for it to complete before continuing.

1. On the Google Cloud console title bar, type **App Engine** in the Search field, then click **App Engine** in the Products & Pages section.
2. Click **App Engine** > **Dashboard**. The upper-right corner of the dashboard should display a link to your application similar to this:

**Note:**By default, the URL to an App Engine instance is in the form of https://project-id/appspot.com.

1. Click on the link to test your program.
2. Refresh your browser a few times to make some requests.
3. Return to the Console and click the **App Engine** > **Versions**.
4. In **Diagnose** column of the table click **Logs**.
5. The logs should indicate that Profiler has started and profiles are being generated. If you get to this point too quickly, wait a minute and click **Refresh**.

Click *Check my progress* to verify the objective.

Assessment Completed!

Deploy an application to App Engine and examine the Cloud logs

Check my progress

*Assessment Completed!*

**Task 3. View Profiler information**

1. On the Google Cloud console title bar, type **Profiler** in the Search field, then click **Profiler** in the Products & Pages section. The screen should look similar to this:

**Note:**The gray bar at the top represents the total amount of CPU time used by the program. The bars below represent the amount of CPU time used by the program's functions relative to the total. At this point, there is no traffic, so the chart is not very interesting. Throw some load at the application.

1. On the **Navigation menu**, click **Compute Engine**.
2. Click **Create Instance** to create a virtual machine.
3. Change the region to someplace other than us-west1 (*the App Engine app is in us-west1*). Accept all the rest of the defaults and click **Create**.

Click *Check my progress* to verify the objective.

Assessment Completed!

Create an instance

Check my progress

*Assessment Completed!*

1. When the VM is ready, click **SSH** to log in to it.
2. If prompted allow SSH-in-browser to connect to VMs, click **Authorize**.
3. You will generate some traffic to your App Engine app using the web testing tool called Apache Bench. Enter the following commands to install it:

sudo apt update

sudo apt install apache2-utils -y

Copied!

content\_copy

1. Update <your-project-id> with your PROJECT\_ID from connection details panel and enter the following command to generate some traffic to your App Engine application:

ab -n 1000 -c 10 https://<your-project-id>.appspot.com/

Copied!

content\_copy

The command will make a thousand requests, 10 at a time, to your application.

**Note:**You have to change the URL to point to your application. Recall that you can find the URL in the App Engine Dashboard. It is also on the browser tab you used to test your app, if you haven't closed it. Also, make sure you insert a slash (/) at the end of the URL.

1. When the requests are finished, on the Google Cloud console title bar, type **Profiler** in the Search field, then click **Profiler** in the Products & Pages section.

Now there is a more interesting chart. Each bar represents a function. The width of the bars represents how much CPU time each function consumed.

The Profiler is a way developers can track down parts of a program that are consuming too many resources.

**Task 4. Explore Cloud Trace**

1. Every request to your application is added to the **Trace** list. On the Google Cloud console title bar, type **Trace Explorer** in the Search field, then click **Trace Explorer** in the Products & Pages section.

This shows a history of requests and their latency. Again, it's not very exciting because the application hasn't been running for very long. The chart in the upper-left plots requests and how long they took. The table to the right shows a list of requests. If you select a request, more detail will be displayed at the bottom of the screen.

1. Return to the SSH window where you entered the Apache Bench command previously.
2. Enter the ab command again:

ab -n 1000 -c 10 https://<your-project-id>.appspot.com/

Copied!

content\_copy

You can also experiment with different values for the -n and -c parameters.

1. Repeat this a couple of times, and then return to the Trace Explorer page.

**Task 5. Monitor resources using Dashboards**

1. On the Google Cloud console title bar, type **Monitoring** in the Search field, then click **Monitoring (Infrastructure and application quality checks)** in the Products & Pages section.
2. In the left pane, click **Dashboards**. Cloud Monitoring analyzes the resources used in your projects and generates some default dashboards for you. In this exercise you have used App Engine and Compute Engine virtual machines, so a table similar to the one shown below should be displayed:
3. Click on the **App Engine** dashboard, and then select your project name. A dashboard of pertinent information for your App Engine application will appear.
4. In the left pane, click **Dashboards**.
5. Click on the **VM Instances** dashboard, and then select your instance. A dashboard for your VM will appear.

**Note:**If you don't see VM Instances right away, wait a minute and refresh your browser.

1. Alternatively, return to the Dashboards page and click the **+Create Dashboard**. Try to create a custom dashboard.
2. For **New Dashboard Name**, type the custom dashboard name you have chosen. You can continue with your custom dashboard by adding the charts.

**Task 6. Create uptime checks and alerts**

1. In the left pane, click **Uptime checks**, and then click the **+ Create Uptime Check** link at the top. Fill out the form as follows:

|  |  |
| --- | --- |
| **Property** | **Value** |
| **Protocol** | HTTPS |
| **Resource Type** | URL |
| **Hostname** | <your-project-id>.appspot.com |
| **Path** | / |
| **Check Frequency** | 1 minute |

1. Click Continue and in **Review** section enter App Engine Uptime Check in **Title**.
2. Click **Test** to verify that your uptime check can connect to the resource. When you see a green check mark everything can connect. Click **Create**.
3. In the **Uptime checks** page click on 3 vertical dots next to your uptime check and select **Add alert policy**.
4. In **Notifications and name** click on the drop down arrow next to **Notification Channels**, then click on **Manage Notification Channels**. A **Notification channels** page will open in a new tab.
5. Scroll down the page and click on **ADD NEW** for Email.
6. In the **Create Email Channel** dialog box, enter your personal email address in the **Email Address** field and a **Display name**.
7. Click on **Save**.
8. Go back to the previous tab. Click on **Notification Channels** again, then click on the **Refresh icon** to get the display name you mentioned in the previous step.
9. Now, select your **Display name** and click **OK**.
10. Name the alert policy as **Uptime Check Alert**.
11. Click **Create Policy**. The uptime check you configured takes a while for it to become active.

Click *Check my progress* to verify the objective.

Assessment Completed!

Create uptime checks and alerts

Check my progress

*Assessment Completed!*

1. Return to the open App Engine tab in order to disable the application to see whether your uptime check and alerting policy work.
2. Click **Settings**.
3. Click **Disable application**. Follow the instructions to disable the application.
4. Return to the **App Engine Dashboard** and test the URL. It shouldn't work anymore.
5. Return to the tab that contains **Monitoring**, and then click **Uptime checks**. Your uptime check should be failing. *If you get there too fast, wait a minute and click refresh*.
6. Click **Alerting**. An incident should have been fired.
7. Check your email. You should get a message from Cloud Monitoring.
8. Return to **App Engine Settings** and re-enable your application.Then return to the **Uptime checks** page. The uptime check should be working again. *If not, wait a minute and then click refresh*.
9. Return to the **Alerting** page. Your incident should be resolved. *As before, you might have to wait a minute and then click refresh*.
10. Check your email again. You should get a second email indicating that the alert recovered.
11. To make sure you don't get any emails after the project is deleted, delete your alerting policy and then delete your notification channel. At the top of the **Alerting** page, click **Edit Notification Channels**.
12. Find your email address and click the trash can icon to delete it.
13. Now click **Uptime checks** and delete your App Engine Uptime check.

**Congratulations!**

In this lab, you deployed an application to Google Cloud and then used the tools provided by Google Cloud to monitor it. You used Cloud Logging, Trace, Profiler, and dashboards and created uptime checks and alerting policies.

**End your lab**

When you have completed your lab, click **End Lab**. Google Cloud Skills Boost removes the resources you’ve used and cleans the account for you.