**Alerting in Google Cloud**

experimentLabschedule1 hour 30 minutesuniversal\_currency\_alt5 Creditsshow\_chartIntermediate

infoThis lab may incorporate AI tools to support your learning.

**Overview**

In this lab, you deploy an application to App Engine and then create alerting policies to notify you if the application is not accessible or is generating errors.

Objectives

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

* Download a sample app from GitHub.
* Deploy an application to App Engine.
* Create uptime checks and alerts.
* Optionally, create an alerting policy with the CLI.

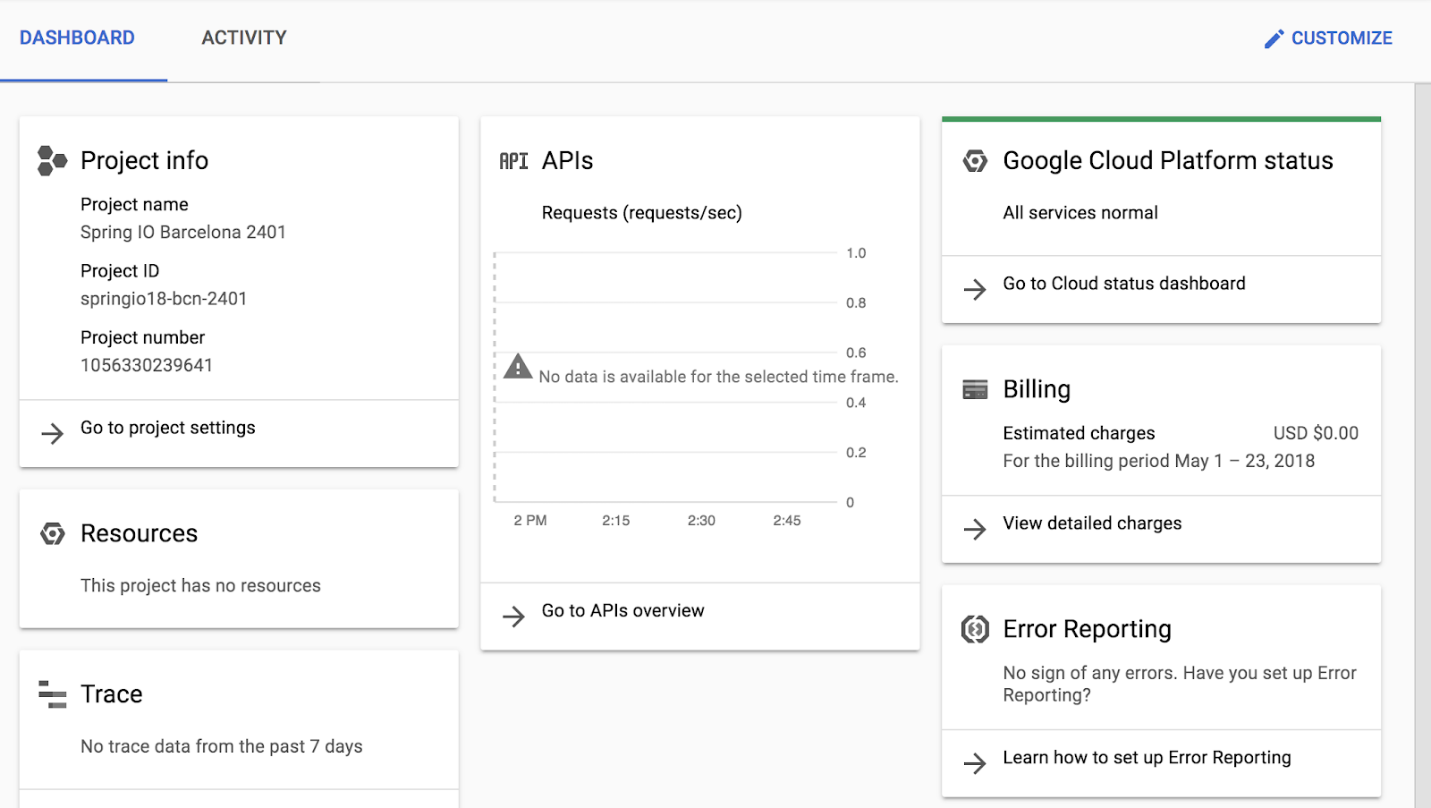
**Setup and requirements**

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.

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

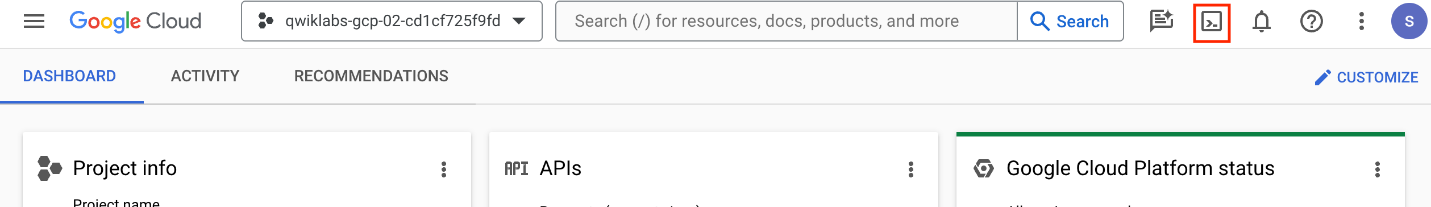


Activate Google Cloud Shell

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 Google Cloud resources.

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



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

* You can list the active account name with this command:

gcloud auth list

Copied!

content\_copy

**Output:**

Credentialed accounts:

- @.com (active)

**Example output:**

Credentialed accounts:

- google1623327\_student@qwiklabs.net

* You can list the project ID with this command:

gcloud config list project

Copied!

content\_copy

**Output:**

[core]

project =

**Example output:**

[core]

project = qwiklabs-gcp-44776a13dea667a6

**Note:**Full documentation of **gcloud** is available in the [gcloud CLI overview guide](https://cloud.google.com/sdk/gcloud" \t "_blank).

Run the following commands to update the python environment

python3 -m pip install --upgrade pip

python3 -m venv myenv

source myenv/bin/activate

Copied!

content\_copy

**Task 1. Download and test a sample app from GitHub**

1. Clone the Google Training Data Analyst from GitHub into your Cloud Shell environment. The repo contains a simple application which is perfect for the requirements in this exercise.

git clone --depth 1 https://github.com/GoogleCloudPlatform/training-data-analyst.git

Copied!

content\_copy

1. Once the cloning completes, change to the deploying-apps-to-gcp folder in the repository that contains our sample app:

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

Copied!

content\_copy

1. Open the main.py file in the Cloud Shell editor. If prompted click **Open in a new window**. Take a moment to explore the basic "Hello GCP" Python Flask application.

edit main.py

Copied!

content\_copy

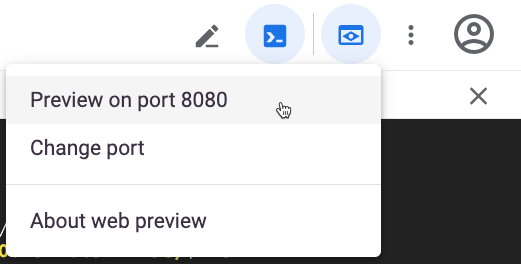
1. Close the editor and switch back to the Cloud Shell terminal. Click **Open terminal** at the top right to open the Terminal window. To test the program, load all of the Python application dependencies and then start the app.
2. pip3 install -r requirements.txt

python3 main.py

Copied!

content\_copy

1. To see the program running, click the **Web Preview** button in the toolbar of Cloud Shell, and then select **Preview on port 8080**.



The program should open a new browser tab and display the Hello GCP message.

1. Close the tab and switch back to Cloud Shell, then press CTRL+C to exit the running Flask server.

**Task 2. Deploy an application to App Engine**

Now that we know the application works, let's deploy it to the App Engine.

1. Switch to (or reopen) the Cloud Shell code editor. Expand the training-data-analyst/courses/design-process/deploying-apps-to-gcp folder in the explorer tree on the left.
2. From the **File** menu, select **New File** and name the file **app.yaml**.
3. Paste the following into the file you just created:

runtime: python312

Copied!

content\_copy

1. To make sure the file is saved, select **File > Save**.

Every project needs to first create an App Engine application before it can be used. This is done just once per project using the Console, or the gcloud app create command. Either way, you need to specify the region where you want the app to be created.

1. Execute the following command in your Cloud Shell terminal. You may have to **Authorize** Cloud Shell to make such a change:

gcloud app create --region=us-east4

Copied!

content\_copy

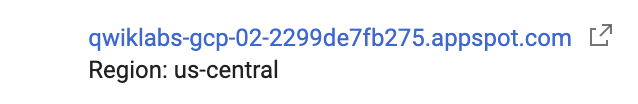
1. Deploy the basic application to App Engine. The following command looks in the current directory for the application. It sees the app.yaml file declaring it a Python application and it assumes the rest of the folder contains the application itself, with a starting point in main.py. It loads the dependencies, packages the application, and deploys into the App Engine as a Service.

gcloud app deploy --version=one --quiet

Copied!

content\_copy

1. Wait for the application to finish deploying, then in the Google Cloud Console window, use the **Navigation menu** (Navigation menu icon) click on **VIEW ALL PRODUCTS** to navigate to **App Engine > Dashboard**.
2. In the upper right of the dashboard, you see a link to your application similar to what is shown below.



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

1. Click the link to test your newly deployed app. It should function exactly like it did when we ran it in Cloud Shell.
2. Click refresh a number of times so Google Cloud can gather some sample data.

Click *Check my progress* to verify the objective.

Assessment Completed!

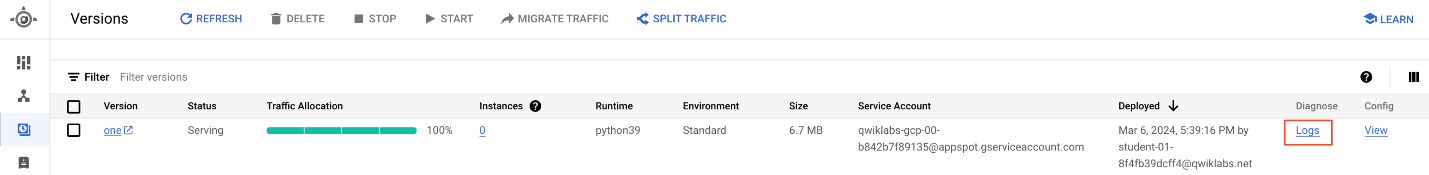
Deploy an application to App Engine.

Check my progress

*Assessment Completed!*

**Task 3. Examine the App Engine logs**

1. Switch back to the Console and on the left side under App Engine, click the **Versions** link.
2. Select **Logs** from the Diagnose column.



1. In the logs, you see the requests you just made. If you get here too fast, wait a few seconds and click the **Jump to Now** button.

**Task 4. Create an App Engine latency alert**

Now that we have the application running, let's create an alert to watch for unusually high latency. To start, let's explore our application's current latency.

Check current application latency in Metrics explorer

1. In the Google Cloud Console, use the **Navigation menu** click on **VIEW ALL PRODUCTS** to navigate to **Monitoring > Metrics explorer**.
2. Click on **Select a metric** drop-down and uncheck the **Active** option.
3. Set the Metric to **GAE Application > Http > Response latency**. Click **Apply**. Make sure the metric is an App Engine metric, and not an uptime check metric. Because we don't currently have an uptime check, the second option won't work.
4. In the **Aggregation** field click on the dropdown and select **99th percentile**.
5. Take a moment and examine the chart.

We don't have a lot of data but we should have enough to display a chart line showing us the average time it took our application to return a response to the fastest 99% of requests, cutting off 1% of anomalies.

Create an alert based on the same metric

Our application is performing as expected now. There may have been a few slow responses when the application was first launched, but on average, you should be seeing response times under 200ms.

Let's create an alert to notify us if we have response times over 5s for more than a minute.

1. Use the **Navigation menu** click on **VIEW ALL PRODUCTS** to navigate to **Monitoring > Alerting**.
2. At the top, click **Edit Notification Channels** and scroll to the Email section. Use **Add New** to add your email address as a valid notification channel. For Display name , choose any name and then click **SAVE**.
3. Switch back to the main **Alerting** page and click **Create Policy**.
4. Click **Select a metric** and uncheck the **Active** option to display the same metric explorer page. Once again, set the Metric to **GAE Application > Http > Response latency**. Click **Apply**. Set **rolling window** to 1 min and then click **Next**.

You'll notice the new Configuration section that's been added to the standard Metrics explorer window in the lower left.

1. Set up a condition so that if **Any time series violates** the Threshold postion **is above** and Threshold value to **8000ms**, it should trigger an alert.
2. Set the condition name to **Response latency [MEAN] for 99th% over 8s**.
3. Take a moment to examine the alert's chart. It's the same chart we created earlier with the Metrics explorer, but this time there should be an alert line drawn at 8s.
4. Click **Next**.
5. For Notifications and name, expand the Notification Channel and check yourself, click **OK** (the notification channel you created earlier in this section).
6. Name the alert **Hello too slow** and click **Next**. Review alert and click **Create Policy**.
7. Update the application code to add a delay. Return to the Cloud Shell code editor. Expand the **training-data-analyst/courses/design-process/deploying-apps-to-gcp** folder in the explorer tree on the left.
8. Click main.py to open it in the editor. Near the top at line 2, add the following imports statements (some will be used later in the exercise):
9. import time
10. import random

import json

Copied!

content\_copy

1. Replace the current main() function with the one below. This new version adds a sleep command which pauses the code for 10s in the middle of each request. This will be well over the threshold.
2. @app.route("/")
3. def main():
4. model = {"title": "Hello GCP."}
5. time.sleep(10)

return render\_template('index.html', model=model)

Copied!

content\_copy

1. Now re-deploy your application by rerunning:

gcloud app deploy --version=two --quiet

Copied!

content\_copy

1. Wait for the command to finish the re-deployment.
2. After the command completes, return to the **App Engine > Dashboard** and make sure the link works.
3. To generate some consistent load, in Cloud Shell, enter the following command:

while true; do curl -s https://$DEVSHELL\_PROJECT\_ID.appspot.com/ | grep -e "<title>" -e "error";sleep .$[( $RANDOM % 10 )]s;done

Copied!

content\_copy

**Note:** This command makes requests to the App Engine app continuously in a loop. The grep command will display the title of the page when the request works. It also displays the error, if it doesn’t work. Every iteration, the thread sleeps a random amount of time less than a second, but with the 10s response time delay it will seem much longer.

1. Wait and after a few minutes (typically about 5), you should receive an email notifying you of the alert. When you do, switch back to the Cloud Shell terminal and use CTRL+C to stop the load tester loop.
2. Use the **Navigation menu** click on **VIEW ALL PRODUCTS** to view the **Monitoring > Alerting** page in the Google Cloud Console.

Notice the firing alert and how it created an incident.

1. Click the incident to view details.
2. Investigate the details page, then scroll to the bottom and add a comment. Finally, scroll to the top and select **Acknowledge Incident**.
3. Switch back to the main **Alerting** page and notice the changes.

Click *Check my progress* to verify the objective.

Assessment Completed!

Create an App Engine latency alert.

Check my progress

*Assessment Completed!*

**Task 5. (Optional) Creating an Alerting Policy with the CLI**

The Alerting CLI (and API) can be very effective when applying alerting policies with code or scripts.

1. Return to the Cloud Shell code editor. Select the **training-data-analyst/courses/design-process/deploying-apps-to-gcp** folder in the explorer tree on the left.
2. From the **File** menu, select **New File** and name the file **app-engine-error-percent-policy.json**.
3. Create an alerting policy which divides the number of 500 errors by the total number of responses, then triggers when the 500 errors exceed 1 percent. Paste the following into the file you just created:

{

"displayName": "HTTP error count exceeds 1 percent for App Engine apps",

"combiner": "OR",

"conditions": [

{

"displayName": "Ratio: HTTP 500s error-response counts / All HTTP response counts",

"conditionThreshold": {

"filter": "metric.label.response\_code>=\"500\" AND metric.label.response\_code<\"600\" AND metric.type=\"appengine.googleapis.com/http/server/response\_count\" AND resource.type=\"gae\_app\"",

"aggregations": [

{

"alignmentPeriod": "60s",

"crossSeriesReducer": "REDUCE\_SUM",

"groupByFields": [

"project",

"resource.label.module\_id",

"resource.label.version\_id"

],

"perSeriesAligner": "ALIGN\_DELTA"

}

],

"denominatorFilter": "metric.type=\"appengine.googleapis.com/http/server/response\_count\" AND resource.type=\"gae\_app\"",

"denominatorAggregations": [

{

"alignmentPeriod": "60s",

"crossSeriesReducer": "REDUCE\_SUM",

"groupByFields": [

"project",

"resource.label.module\_id",

"resource.label.version\_id"

],

"perSeriesAligner": "ALIGN\_DELTA"

}

],

"comparison": "COMPARISON\_GT",

"thresholdValue": 0.01,

"duration": "0s",

"trigger": {

"count": 1

}

}

}

]

}

Copied!

content\_copy

1. Save your file. Make sure you are in the correct folder and that your alerting policy was saved there.
2. cd ~/training-data-analyst/courses/design-process/deploying-apps-to-gcp

ls

Copied!

content\_copy

1. Deploy the alerting policy with the following command:

gcloud alpha monitoring policies create --policy-from-file="app-engine-error-percent-policy.json"

Copied!

content\_copy

1. Back in the Google Cloud Console, return to the **Monitoring > Alerting** page. In the **Policies** section, you should see the alerting policy you just deployed.
2. Click your **HTTP error…** policy to see its details. Edit the notification channel so you receive an email when the alert fires.
3. Deploy the App Engine app with a random error to test this policy. Return to the Cloud Shell code editor. Expand the **training-data-analyst/courses/design-process/deploying-apps-to-gcp** folder in the explorer tree on the left.
4. Open the main.py file in the Cloud Shell editor.
5. Replace the current main() function with the one below. This new function removes our sleep command and uses the random number generator to return a 500 error about 2% of the time. This should be enough to cause your HTTP error count policy to trigger.

@app.route("/")

def main():

num = random.randrange(49)

if num == 0:

return json.dumps({"error": 'Error thrown randomly'}), 500

else:

model = {"title": "Hello GCP."}

return render\_template('index.html', model=model)

Copied!

content\_copy

1. Redeploy your application with the following command:

gcloud app deploy --version=two --quiet

Copied!

content\_copy

1. Wait for the command to finish the redeployment.
2. After the command completes, return to the **App Engine > Dashboard** and make sure the link works.
3. In Cloud Shell, rerun the load generator command from earlier:

while true; do curl -s https://$DEVSHELL\_PROJECT\_ID.appspot.com/ | grep -e "<title>" -e "error";sleep .$[( $RANDOM % 10 )]s;done

Copied!

content\_copy

In Cloud Shell, you should see an error appear randomly about 2% of the time. Leave the command running.

1. In the Console, go to **Monitoring > Alerting**, wait another few minutes, and then you should see an alerting incident fire. You should also get an email. This won’t happen immediately, you have to be patient.
2. Once you get the email and see the incident fire, go back to Cloud Shell and press CTRL+C to stop the requests.

If you wait a couple more minutes, you see the incident resolve itself and get another email.

1. To make sure you don’t get any emails after the project is deleted, delete your notification channels. Click the **HTTP error count…** policy to see its details. Click the **Edit** link, expand the Notification Channel and uncheck yourself. Toggle off Use notification channel. Save your changes and then **Delete** the policy.
2. Repeat the previous step to your Uptime check alerting policy.

**Review**

In this exercise, you used the Google Cloud Console and CLI to create and test alerting policies. Good job!

**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.

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.

Copyright 2022 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.