


Creating and Alerting on Logs-based Metrics

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**  at the top of the Google Cloud console.
2. Click through the following windows:
 - Continue through the Cloud Shell information window.
 - Authorize Cloud Shell to use your credentials to make Google Cloud API calls.

When you are connected, you are already authenticated, and the project is set to your **Project_ID**, `qwiklabs-gcp-01-854a299cedab`. The output contains a line that declares the **Project_ID** for this session:

```
Your Cloud Platform project in this session is set to quiklabs-gcp-01-854a299cedab
```

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

3. (Optional) You can list the active account name with this command:
`gcloud auth list`
Copied!

```
content_copy
```

4. Click **Authorize**.
Output:

```
ACTIVE: *  
ACCOUNT: student-00-6be5116ac4eb@quiklabs.net  
To set the active account, run:
```

```
$ gcloud config set account `ACCOUNT`
```

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

```
gcloud config list project
```

Copied!

content_copy

Output:

```
[core]
project = qwiklabs-gcp-01-854a299cedab
```

Note: For full documentation of gcloud, in Google Cloud, refer to [the gcloud CLI overview guide](#).

Task 1. Deploy a GKE cluster

In this task, you deploy a Google Kubernetes Engine (GKE) cluster to use in later tasks for log-based metrics.

1. In Cloud Shell, set the zone in for this lab environment:

```
gcloud config set compute/zone us-central1-a
```

Copied!

content_copy

If prompted, click **Authorize** Cloud Shell.

2. Set the project ID for this lab environment:

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

Copied!

content_copy

3. Deploy a standard GKE cluster:

```
gcloud container clusters create gmp-cluster --num-nodes=1 --zone us-central1-a
```

Copied!

content_copy

When the cluster has been deployed, the output displays *STATUS: RUNNING* for the cluster named `gmp-cluster`.

Note: It may take several minutes for the cluster to be deployed. You can proceed to complete Task 2, and then return to validate your progress using the check below.

Task 2. Create a log-based alert

Log-based alerts notify you whenever a specific message appears in your logs. Try it out by setting up a log-based alert to tell you when a VM stops running.

1. In the Google Cloud console title bar, type **Logs explorer** in the **Search** field, then click **Logs Explorer** from the search results.
2. If needed, enable the **Show Query** slide bar.
3. Copy and paste the following parameters into the query window to create Log Based Alert:

```
resource.type="gce_instance"  
protoPayload.methodName="v1.compute.instances.stop"
```

Copied!

content_copy

4. Under **Actions** (at the top of the Results section), click **Create log alert**.
5. Add the following parameters, and click **Next** after adding each value, so that you can see the next section:
 - **Alert policy name:** stopped vm
 - **Choose logs to include in the alert:** this section auto-fills with the query you entered previously
 - **Set notification frequency and autoclose duration:** Select **Time between notifications** as 5 min and **Incident autoclose duration** as 1 hr.
6. Click **Next**.
7. For **Who should be notified**, complete the following:
 - Click on the dropdown arrow next to **Notification Channels**, then click on **Manage Notification Channels**. (A Notification channels page will open in the new tab.)
 - Scroll down the page and click on **Add new** for **Email**.
 - Enter an email in the **Email Address** field and a **Display name**. You can use your personal email if you want to view the email, or you can provide the lab username (student-00-6be5116ac4eb@qwiklabs.net) for which you cannot see the email.
 - Click **Save**.
 - 8. Close the **Notification Channels** using the 'X' at top of the page, so you can return to the Logs Explorer tab you were in previously.
 - Refresh the Notification Channels, then select the channel you just created. Click **OK**.
 - 9. Click **Save**.

Test the log-based alert

To test this log-based alert, you stop your VM and check Logging to see if your alert has registered:

1. Open a second Google Cloud console browser tab, and navigate to **Navigation menu > Compute Engine > VM instances**.
2. Check the box next to **instance1**, then click **Stop** at the top of the page. Click **Stop** again in the pop-up window.

This may take a moment. When the instance has been stopped, the green check mark will turn to a gray circle.

3. In the console title bar, type **Monitoring** in the **Search** field, then choose **Monitoring (Infrastructure and application quality checks)** the search results.
4. In the left pane, under **Detect**, click **Logging**, click on **Alerting** under **Detect**.

You should see that your alert has registered.

5. Under **Policies**, click the **See all policies** to see the log-based alert you created named **stopped vm**.

Task 3. Create a Docker repository

In this section, you'll create a private Docker repository within Artifact Registry and add image to the private repository. This involves tagging the image with the repository name to specify its destination and then pushing it to Artifact Registry.

1. In Cloud Shell, run the following command to create a new Docker repository named **docker-repo** in the location **us-central1** with the description "Docker repository".

```
gcloud artifacts repositories create docker-repo --repository-  
format=docker \  
  --location=us-central1 --description="Docker repository" \  
  --project=qwiklabs-gcp-01-854a299cedab
```

Copied!

content_copy

2. In the console title bar, type **Artifact Registry** in the **Search** field then click "Artifact Registry" from the search results.
3. On the **Artifact Registry Repositories** page, verify you can see your repository, `docker-repo`.
4. In Cloud Shell, load a pre-built image from a storage bucket:

```
wget https://storage.googleapis.com/spls/gsp1024/flask_telemetry.zip
unzip flask_telemetry.zip
docker load -i flask_telemetry.tar
```

Copied!

content_copy

5. Run the following command to tag the image as `flask-telemetry:v1`:

```
docker tag gcr.io/ops-demo-330920/flask_telemetry:61a2a7aabc7077ef474eb24f4b69faeab47deed9 \
us-central1-docker.pkg.dev/qwiklabs-gcp-01-854a299cedab/docker-repo/flask-telemetry:v1
```

Copied!

content_copy

6. Run the following command to push the docker image to Artifact Registry:

```
docker push us-central1-docker.pkg.dev/qwiklabs-gcp-01-854a299cedab/docker-repo/fl
```

Task 4. Deploy a simple application that emits metrics

Using log-based metrics you can define a metric that tracks errors in the logs to proactively respond to similar problems and symptoms before they are noticed by end users.

1. Run the following command to check that the cluster you deployed in Task 1 has been fully provisioned:
- ```
gcloud container clusters list
```
- Copied!
- content\_copy

**Note:** If your cluster status says **PROVISIONING**, wait a moment and run the command above again. Repeat until the status is **RUNNING**, at which point you can continue with the next steps.

2. Authenticate the cluster:

```
gcloud container clusters get-credentials gmp-cluster
```

Copied!

content\_copy

You should see the following message:

```
Fetching cluster endpoint and auth data.
kubeconfig entry generated for gmp-cluster.
```

3. Create a namespace to work in:

```
kubectl create ns gmp-test
```

Copied!

content\_copy

4. Get the application which emits metrics at the `/metrics` endpoint:

```
wget https://storage.googleapis.com/spls/gsp1024/gmp_prom_setup.zip
unzip gmp_prom_setup.zip
cd gmp_prom_setup
```

Copied!

content\_copy

5. In this step, you update `flask_deployment.yaml` to use the name of the image you pushed in previous steps:

- Use nano to open `flask_deployment.yaml`:  
`nano flask_deployment.yaml`

Copied!

content\_copy

- Replace `<ARTIFACT REGISTRY IMAGE NAME>` with the following:  
`us-central1-docker.pkg.dev/qwiklabs-gcp-01-854a299cedab/docker-repo/flask-telemetry:v1`

Copied!

content\_copy

- Press `CTRL+X`, `Y`, then `ENTER` to save the updated file and close nano.
6. Now run the following to deploy a simple application that emits metrics at the `/metrics` endpoint:

```
kubectl -n gmp-test apply -f flask_deployment.yaml
```

Copied!

content\_copy

```
kubectl -n gmp-test apply -f flask_service.yaml
```

Copied!

content\_copy

### 7. Verify that the namespace is ready and emitting metrics:

```
kubectl get services -n gmp-test
```

Copied!

content\_copy

You should see the following output:

| NAME  | TYPE         | CLUSTER-IP  | EXTERNAL-IP  | PORT(S)      | AGE |
|-------|--------------|-------------|--------------|--------------|-----|
| hello | LoadBalancer | 10.0.12.114 | 34.83.91.157 | 80:32058/TCP | 71s |

### 8. Re-run the command until you see the **External-IP** address populated.

### 9. Check that the Python Flask app is serving metrics with the following command:

```
curl $(kubectl get services -n gmp-test -o
jsonpath='{.items[*].status.loadBalancer.ingress[0].ip}')/metrics
```

Copied!

content\_copy

You should see the following output:

```
HELP flask_exporter_info Multiprocess metric
TYPE flask_exporter_info gauge
flask_exporter_info{version="0.18.5"} 1.0
```

## Task 5. Create a log-based metric

### 1. Return to **Logs Explorer**.

### 2. Under **Actions**, click **Create metric** link.

### 3. On the Create metric page, input the following:

- **Metric type:** leave the default setting (**Counter**)

- **Log based metric name:** hello-app-error
- **Filter selection:** update **Build filter** window with the following information:  
`severity=ERROR`  
`resource.labels.container_name="hello-app"`  
`textPayload: "ERROR: 404 Error page not found"`

4. Click **Create metric**.

## Task 6. Create a metrics-based alert

1. From the left side menu for **Logging**, click on **Log-based Metrics** under **Configure**.
  2. In user-defined metrics, click on **More actions** (⋮) for **hello-app-error**, and select **Create alert from metric**.
  3. Under **Select a Metric**, the metric parameters will automatically fill in.
- Update the Rolling window to **2 min**.
  - Accept the other default settings.
  - Click **Next** twice.
    4. Set notifications using the channel you created earlier in the lab.
    5. Name the alert policy: **log based metric alert**
    6. Click **Create Policy**.

## Task 7. Generate some errors

Next you generate some errors to match the log-based metric you created and trigger the metric-based alert.



1. In Cloud Shell, run the following to generate some errors:

```
timeout 120 bash -c -- 'while true; do curl $(kubectl get services -n gmp-test -o jsonpath='{.items[*].status.loadBalancer.ingress[0].ip}')/error; sleep $((RANDOM % 4)) ; done'
```

Copied!

content\_copy

2. Return to the **Logs Explorer** page, and go to the Severity section on the lower left side.
3. Click on the **Error** severity.

Now you can search for the 404 Error page not found error. View more information by expanding one of the 404 Error messages.

4. Return to the **Monitoring** page, and click on **Alerting**.

You should see the 2 policies you created.

5. Under **Alert policies**, click on **View all**.

You should see both alerts in the Incidents section.

6. Click on an incident to see details.

**Note:** The log-based metric alert eventually resolves itself. If you need more time to investigate, you can run the code to generate errors again and wait for the alert to be triggered again.