SE
CLOUD EXPERTS

Notification solutions in Google Cloud

Architecture and deployment guidelines

These solutions are presented as a result of multiple interactions with customers in need of having a notification solution when using the Google ecosystem, to inform and report events and key resource details.

We'll explore two distinct approaches to build custom notification systems that fit your needs:

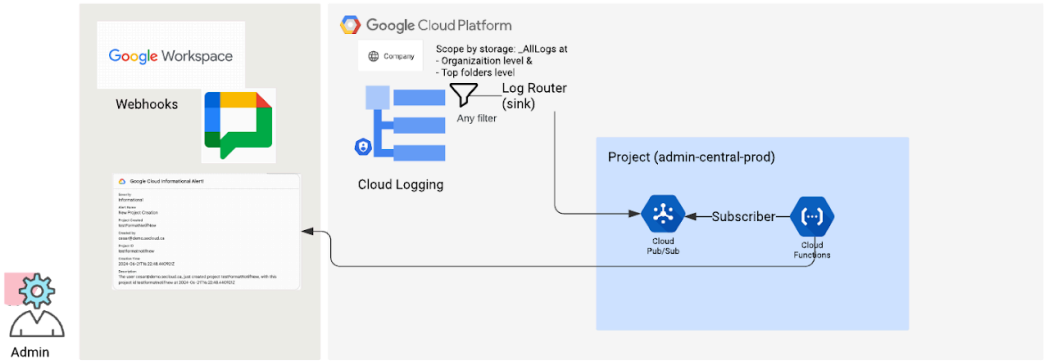
1. Cloud Logging-Based
2. Cloud Asset API-Driven Notifications

secloudexperts.com
info@secloud.ca
+1-514-916-2929

1. Cloud Logging-Based Notifications

This leverages Google's robust logging service. By setting up filters to capture specific events, like new VM creation or security policy changes, you can trigger notifications in Google Chat.

Google Chat App Notification triggered by Cloud Logging Filter



Prerequisites

Create a Google Chat Space

Start by creating a dedicated space in Google Chat where notifications will be sent.

Add Members

Invite all the users who need to receive notifications about missing "backup" labels to this space.

Create a Webhook

- Go to the space settings.
- Find the option to add a webhook and follow the instructions.
- Copy the webhook URL provided – **you'll need it later.**

Secure the Webhook URL

Store the webhook URL securely in your Cloud Run or Cloud Functions environment variables.

Permissions

Ensure the Google project you're working with has the necessary IAM permissions to interact with Google Chat.

Google Services, Identifies, APIs and IAM required permissions

The following table specifies Google services & principals and APIs.

Google Services	API
Google Pub/Sub	Cloud Pub/Sub API
Google Logging	Cloud Logging API
Google Cloud Build	Cloud Build API
Service Account	N/A

The following table provides a view of the google permissions required for the solution.

Type	Principal	Role	Apply at
User or Service Account	Deployer account or Custom service account	roles/logging.configWriter	Org level
User or Service Account	Deployer account or Custom service account	roles/pubsub.editor, roles/cloudfunctions.developer	Project level
Service Account	Logging Sink Service account for the Organization and Top Folders	roles/pubsub.publisher	Organization and Top Folder level
Custom Service Account	Custom Service Account created to execute the function	roles/logging.logWriter, roles/artifactregistry.createOnPushWriter, roles/storage.objectAdmin, roles/run.invoker	Project level

Google Pub/Sub Deployment tips

Documentation can be found here

Deploying with gcloud command

```
gcloud pubsub topics create projectid-pubsub-sequence
```

Log Aggregated Sink Deployment tips

Documentation can be found here
gcloud command documentation can be found here

You can deploy your logging sinks with gcloud following the example below.
The **include-children** flag is necessary to retrieve logs from projects created directly at the organization level.

```
gcloud logging sinks create projectid--lgrsk-o-sequence \
    pubsub.googleapis.com/$topic \
    --include-children \
    --organization=$orgid \
    --log-filter='<your filter here>' \

gcloud pubsub topics add-iam-policy-binding $topic \
  --member="serviceAccount:service-org-<your org id
here>@gcp-sa-logging.iam.gserviceaccount.com" \
  --role='roles/pubsub.publisher'
```

It is important to apply the same filter at the **Top folders** level.

```
counter=0

for f in (gcloud resource-manager folders list --organization orgid
--format="value(name)")
do
  ((counter++))
  folder=$f
  echo "$f" >> folderlist.txt
  gcloud logging sinks create projectid-lgrsk-f-sequence \
    pubsub.googleapis.com/$topic \
    --include-children \
    --folder=$folder \
    --log-filter=$logfilter \

done

for folder in $(cat folderlist.txt)
do
  f=$folder
  gcloud pubsub topics add-iam-policy-binding $topic \
    --member="serviceAccount:service-folder-$f@gcp-sa-logging.iam.gserviceaccount.com" \
    --role='roles/pubsub.publisher'
done
```

Google Cloud function deployment tips

General Cloud Function documentation can be found here

Make sure to include **requirements.txt**, **main.py** and **config.yaml** files in the same directory from where you are executing the gcloud command.

You can find below the gcloud command to create function V1.

```
gcloud functions deploy projectid-cf-$short-sequence \
--runtime=python312 \
--retry \
--service-account=$serviceaccount \
--trigger-service-account=$serviceaccount \
--trigger-topic=$topic \
--region=$region \
--source=. \
--entry-point=hello_pubsub \
--env-vars-file config.yaml
```

You can find below the gcloud command to create function V2.

```
gcloud functions deploy projectid-cf-$short-sequence \
--runtime=python312 \
--retry \
--service-account=$serviceaccount \
--gen2 \
--trigger-service-account=$serviceaccount \
--trigger-topic=$topic \
--region=$region \
--source=. \
--entry-point=hello_pubsub \
--env-vars-file config.yaml
```

- requirements.txt

```
httplib2
functions-framework==3.*
```

- config.yaml

```
CHAT_WEBHOOK_URL : "<your chat web hook url here>"
```

- main.py (pseudo code)
  - here is the link to get more information about build a google chat app as a webhook;
  - here is the link that can help you to format your card message.

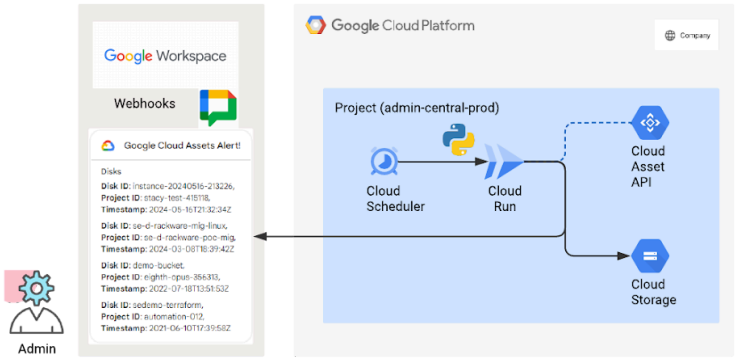
Here is the pseudo code of the **main.py file**

```
 1. Function Definitions:

  Define a function hello_pubsub triggered by a Cloud Pub/Sub message:

  Decode the base64 encoded message data from the event.
  Convert the decoded data to a JSON object.
  Check if the message is empty (handle potential error).
  If the message is not empty, call the send_chat_message function with the message data.
  Define a function send_chat_message to send a message to Google Chat:

  Get the Chat webhook URL from environment variables.
  Extract relevant information from the message data:
  User email
  Timestamp
  Project name
  Project ID
  Construct a formatted message string including the extracted information.
  Create a structured message object with header, sections, and key-value pairs for
details.
  Set message headers for content type (JSON).
  Create an HTTP object for sending the request.
  Send a POST request to the webhook URL with the message object as JSON data.
  Print the response from the server.
  2. Main Execution:

  The code doesn't have a traditional main function, but it relies on the Cloud Event
trigger to call the hello_pubsub function when a Pub/Sub message arrives.
  3. Exit:

  There's no explicit exit statement. The program continues to listen for Pub/Sub
messages.
```

2. Cloud Asset API-Driven Notifications

This method taps into the Cloud Asset API to gather information about your cloud resources. Here is a list of supported resource types. This solution uses Cloud Run to execute a call to the Google Cloud Asset API using your parameters of interest. The API then sends back a response as a formatted message to a Google Chat channel.

Google Chat App Notification triggered by Cloud Asset API with Cloud Run Job



Prerequisites

Google Chat Webhook

Start by creating a dedicated space in Google Chat where notifications will be sent.

Access to a Google project

- with the Google IAM permissions specified in the next section;
- with read access to a Google Storage bucket.

Google Services, Identifies, APIs and IAM permissions required

The following table specifies Google services & principals and APIs.

Google Services	API
Google Cloud Build	Cloud Build API
Google Cloud Scheduler	Cloud Scheduler API
Google Cloud Run	Cloud Run Admin API
Google Asset API	Cloud Asset API
Google Storage	Cloud Storage API
Service Account	N/A

The following table provides a view of the google permissions required for the solution.

Type	Principal	Role	Apply at
User or Service Account	Deployer account	Cloud Run Developer,	Project level
Service Account	Custom service account	cloud asset viewer	Organization level
Service Account	Custom service Account	Cloud Scheduler Admin, Cloud Run Invoker, Storage Admin, Cloud Run Developer, Cloud Run Service Agent	Project level

Google Pub/Sub Deployment tips

Documentation can be found here

Here is the gcloud command to deploy the cloud run job

```
gcloud run jobs deploy <job name>\
    --source . \
    --max-retries 1 \
    --region <region>\
    --project=<project ID> \
    --sa=<custom SA>
    --set-env-vars "ORG_ID=<organization's ID>" \
    --set-env-vars "BUCKET_NAME=<GCS bucket name>" \
    --set-env-vars "CHAT_SHOW_LIST=<chat show list>" \
    --set-env-vars "EXECUTION=<exec mode>" \
    --set-env-vars "CHAT_WEBHOOK_URL=<chat webhook url>"
```

main.py, Procfile and requirements.txt

NOTE

The files **main.py**, **Procfile** and **requirements.txt** must be located in the folder in which the gcloud run deploy command is executed.

---

main.py

Following is a pseudo code for the main.py

- here is the link to get more information about build a google chat app as a webhook;
- here is the link that can help you to format your card message.

```
1. Initialization:

Get organization ID (ORG_ID) and bucket name (BUCKET_NAME) from environment variables.
Get chat webhook URL (CHAT_WEBHOOK_URL) from environment variables (optional).

2. Search for Disks:

Create an Asset Service client.
Define a search scope with ORG_ID and asset type compute.googleapis.com/Disk.
Define a query to exclude disks with labels containing "nobackup" or "backup”.
Perform the search and store the results.
Check if any disks were found.

3. Process Results (if disks found and weekday is the expected one):

Initialize an empty string for CSV output.
Loop through each disk in the results.
Extract relevant information:
Disk name.
Project ID.
Creation time.
Append this information to the CSV output in a comma-separated format.

4. Upload CSV to Cloud Storage:
```

```
Generate a unique filename for the CSV file based on the current date.
Create a Storage client.
Get the bucket with the specified name.
Create a new blob in the bucket with the generated filename.
Open the blob for writing.
Write the CSV output to the blob.

5. Log and Send Notification (optional):

Create a log entry with the CSV output (for information).
Define a function send_chat_message (if CHAT_WEBHOOK_URL is provided).
This function parses the CSV data into a list of disks.
It then formats each disk information into a user-friendly message.
Finally, it sends a chat notification using the webhook URL with the formatted message.
Create another log entry with the notification status (for information).

6. Main Execution:

Call the main function.
Within main, handle any exceptions during execution and create an appropriate log entry.
Print the final log entry (containing the program status - success or failure).

7. Exit Program:

Exit the program.
```

Procfile

```
web: python3 main.py
```

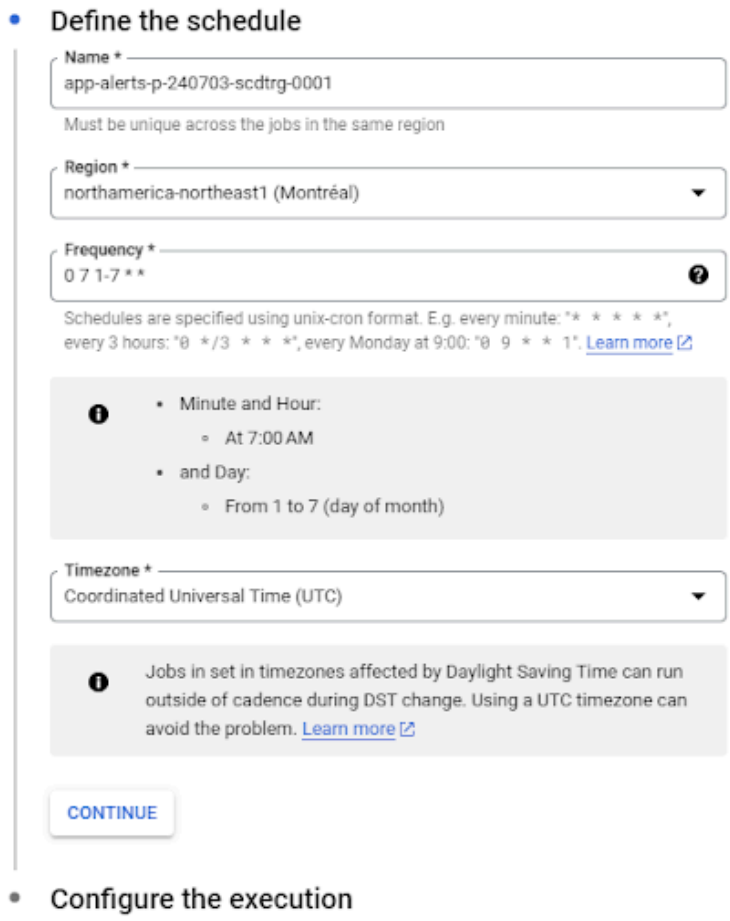
requirements.txt

```
google-cloud-storage
google-cloud-asset
protobuf
datetime
httplib2
```

Trigger cloud run job with Google Cloud Scheduler

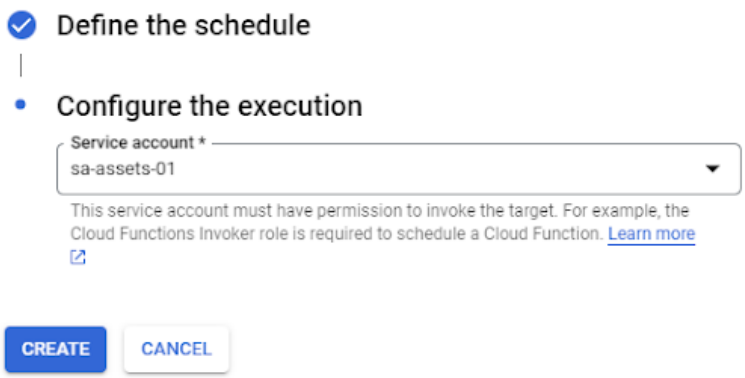
The Schedule instance will be created from Cloud Run jobs, and must be associated with the custom service account. Follow these steps in the Google Cloud console:

1. Go to Cloud Run, click on **JOBS**, then click on the job with the name "" to view the job details
2. Next, click on **TRIGGERS**
3. Click on " + **ADD SCHEDULER TRIGGER**"
4. Fill in the following information:



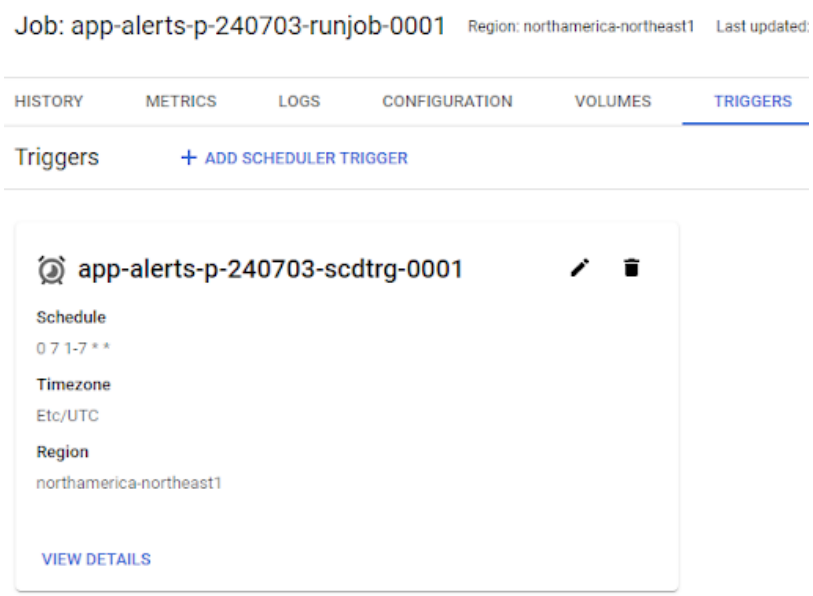
5. Click on **CONTINUE**

6. Select the custom service account:



7. Click on **CREATE**

The new **trigger** will be added to the Cloud Run Job, as seen below:



Manual execution

```
gcloud run jobs execute <job name> --region <region>
```

Parameters

NOTE

The following parameters must be used in the above commands:
= the name of the Cloud Run job
= the region in which the Cloud Run job is deployed
= the Google Cloud in which the job is deployed
= the ID of the organization
= the custom service account
= the name of the bucket for saving the CSV files
= defines the chat presentation:
without report = N (default)
with report = Y
= defines how the job is invoked:
scheduled = S (default)
manual = M
= the URL of the Google Chat webhook