[External] Preventing GCP API's from being enabled via Stackdriver and Cloud Functions - A step-by-step guide

Author: reechar@
Last updated: 2018-09-19

Link to terraform module: https://github.com/reechar-goog/GCP-API-Police-CF-tf

High Level Overview

The following step-by-step guide will create a new GCP project inside a targeted organization. In that new project we will create a Pub/Sub topic to receive Stackdriver Cloud Audit Logs exported from the organization. That topic will be used to trigger a Cloud Function that can disable any out of policy API activations.

Step-by-step Guide

We start by setting some environmental variables that we will use for subsequent steps. We make the assumption that the resources to do this will be created in a new project, although one could modify these instructions to run inside an existing project. Please set PROJECT_ID, ORG_ID, and BILLING_ID for your specific use case. PUBSUB_TOPIC and SD_SINK_NAME are just chosen defaults, feel free to replace with names of your choosing. PUBSUB_TOPIC is the name of the topic into which we will export our Stackdriver logs. SD_SINK_NAME is the name of the SD Aggregated Export Sink which will look for whenever an API is enabled/activated in a project in the org.

```
PROJECT_ID=reechar-api-police #replace
ORG_ID=123456678 #replace
BILLING_ID=ABCDEF-ABCDEF-ABCDEF #replace
PUBSUB_TOPIC=service-activate-topic
SD_SINK_NAME=service-activate-sd-sink
```

This will create a new project in the org we want to monitor.

```
gcloud projects create $PROJECT_ID --organization=$ORG_ID
```

After the project is created, we change the current project to the new one we just created.

```
gcloud config set project $PROJECT_ID
```

We need to link a billing account to the new project so that we can use APIs that are chargeable, e.g Cloud Functions needs an active billing account to run.

```
gcloud beta billing projects link $PROJECT_ID --billing-account=$BILLING_ID
```

This example depends on having Pub/Sub API enabled to link the Stackdriver logs as a source to triggering our Cloud Function.

```
gcloud services enable pubsub.googleapis.com
```

We also need to enable the Cloud Functions API to utilize Cloud Functions.

```
gcloud services enable cloudfunctions.googleapis.com
```

We create a new Pub/Sub topic to be our log receiver.

```
gcloud pubsub topics create $PUBSUB_TOPIC
```

This command sets up the aggregated export Stackdriver logging sink. This aggregated export is targeted at the organization level. The ‘include-children’ flag will make sure that all projects under the organization will be monitored. The ‘log-filter’ specifically looks for when an API is enabled or activated (the author is unaware of the difference between enabled and activated, however some APIs go one pathway and others go the other). We name the sink from the environmental variable, and we set the destination to the Pub/Sub topic we just created.

```
gcloud logging sinks create $SD_SINK_NAME
pubsub.googleapis.com/projects/$PROJECT_ID/topics/$PUBSUB_TOPIC
--include-children --organization=$ORG_ID
--log-filter="(protoPayload.methodName:\"google.api.serviceusage\" AND
protoPayload.methodName:EnableService) OR
(protoPayload.methodName:\"google.api.servicemanagement\" AND
protoPayload.methodName:ActivateServices)"
```

When a new export sink is created, a GCP service account is created to perform the writing of the logs that fit the filter into the destination. We have no way of providing a service account, so we run this command to find out and set as an environmental variable the service account associated with writing this sink.

```
SD_WRITER_SA=`gcloud logging sinks describe
```

```
organizations/$ORG_ID/sinks/$SD_SINK_NAME --organization=$ORG_ID | grep
"writerIdentity: serviceAccount:"| cut -d ':' -f 3`
```

We then take the service account and grant it the Pub/Sub Publisher role on the topic we created. This allows the log export to write to our topic.

```
gcloud beta pubsub topics add-iam-policy-binding $PUBSUB_TOPIC
--member=serviceAccount:$SD_WRITER_SA --role=roles/pubsub.publisher
```

This will pull a sample Cloud Function that was written by the author. This should provide a good starting set to modify for one's own use. It is recommended that one understands and customizes for their own specific usage before using this in any production environment.

```
git clone https://github.com/reechar-goog/GCP-API-Police-CF.git
```

Change directory into the folder just cloned.

```
cd GCP-API-Police-CF
```

This command will take a little bit of time to run on the initial deploy. This takes the 'apiPolice' function that is defined in the index.js file and binds it to be triggered whenever a message is published to our Pub/Sub topic. With the retry flag, if the background function fails, it will run again until it succeeds.

```
gcloud functions deploy apiPolice --trigger-resource $PUBSUB_TOPIC
--trigger-event google.pubsub.topic.publish --retry
```

The Cloud Function runs as a service account identified as $PROJECT_ID@appspot.gserviceaccount.com. Since we want this service account to be able to disable an out of policy API, it needs to have the Editor role on the organization.

```
gcloud organizations add-iam-policy-binding $ORG_ID
--member=serviceAccount:$PROJECT_ID@appspot.gserviceaccount.com
--role=roles/editor
```

Verification

Now that we have deployed our Cloud Function, let us test and make sure everything is hooked up correctly. We assume we are using the Cloud Function unmodified, which has translate.googleapis.com on a blocked API list. Let us list the currently enabled APIs in the project. Save or make note of the output on the initial run.

```
gcloud services list
```

We then enable the Cloud Vision (vision.googleapis.com) API. This is not on the blocked API list, so there should be no issues.

```
gcloud services enable vision.googleapis.com
```

We now enable the Cloud Translate (translate.googleapis.com) API. This service is on the blocked API list, and should be disabled.

```
gcloud services enable translate.googleapis.com
```

We should now list the currently enabled APIs again. We should see the same output as before, but with the addition of the Cloud Vision API. We should not see the translate API enabled.

```
gcloud services list
```

This will grab the logs from our Cloud Function. We should see INFO lines about an account attempting to enable the translate and vision APIs. There should be a message stating that vision.googleapis.com is not on the blocked API list. There should be a log line at the ERROR level which states that translate.googleapis.com is blocked, and attempting to disable.

```
gcloud functions logs read --min-log-level=info
```

Future Work

This example was mostly done as a proof of concept. There are many extensions that one can add to the Cloud Function code. I have already implemented one that can be optionally enabled in the sample code with a whitelist mode.

Some additional ideas:

1. Add e-mail notification via SendGrid SendGrid CF Tutorial. Send an e-mail to the account that is violating the policy and/or send an email to a security team
2. Add additional configuration points. The currently implementation applies a blanket policy to the entire organization. Add a method to allow different allow/deny lists on different projects/folders. Perhaps adopt the Org Policy yaml configuration file format.
3. Add lazy loading of authentication client for more efficient resource usage and performance.

Appendix: index.js

```
'use strict';
const { google } = require('googleapis');
const servicemanagement = google.servicemanagement('v1');
let authenticationClient;
getAuthClient();

function getAuthClient() {
  google.auth.getClient({
    scopes: ['https://www.googleapis.com/auth/compute',
      'https://www.googleapis.com/auth/cloud-platform',
      'https://www.googleapis.com/auth/service.management']
  }).then(function (authClient) {
    authenticationClient = authClient;
  })
}

exports.apiPolice = (event, callback) => {
  const pubsubMessage = event.data;
  const cloudAuditLogMsg = JSON.parse(Buffer.from(pubsubMessage.data, 'base64').toString())
  const accountID = cloudAuditLogMsg.protoPayload.authenticationInfo.principalEmail
  const resourceName = cloudAuditLogMsg.protoPayload.resourceName;
  var apiName;
  if (cloudAuditLogMsg.protoPayload.methodName.endsWith('ActivateServices')){
    apiName = resourceName.substring(resourceName.indexOf('[')+1, resourceName.indexOf(']'))
  } else {
    apiName = resourceName.substring(resourceName.lastIndexOf('/')+1)
  }
  const projectID = "project:" + cloudAuditLogMsg.resource.labels.project_id

  const blockedList = ['translate.googleapis.com']

  console.log(`${accountID} attempted to activate ${apiName} in ${projectID}`)
  if (blockedList.indexOf(apiName) > -1) {
    console.error(`${apiName} is on the blocked API list and will be disabled`);
    servicemanagement.services.disable({ auth: authenticationClient, serviceName: apiName,
requestBody: { consumerId: projectID } })
      .then(function (response) {
        callback(null, 'Success!');
      })
      .catch(function (error) {
        callback(new Error('Failed'));
      });
  } else {
    console.log(`${apiName} not blocked`);
    callback(null, 'Success!');
  }

}
```