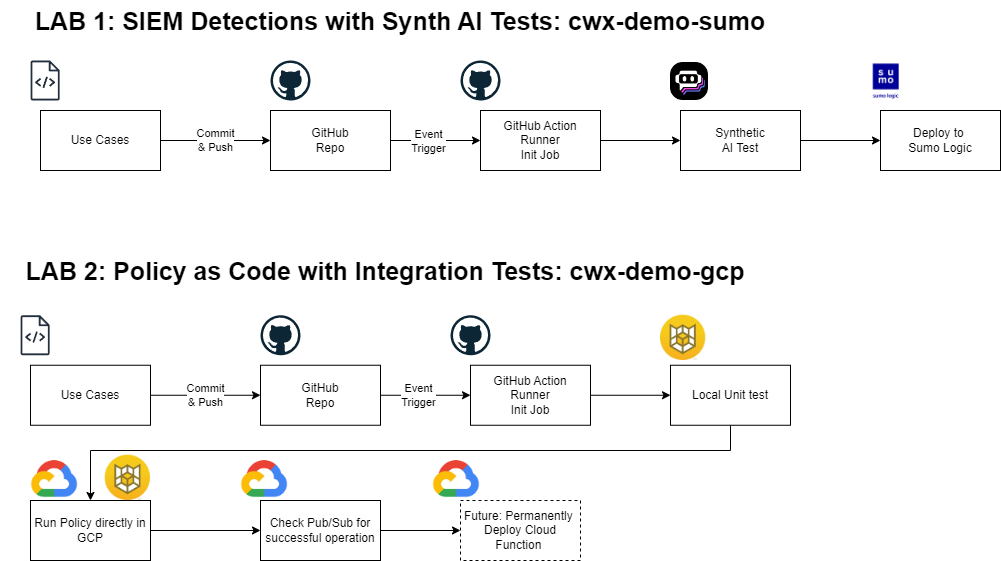
# What you’re building today



# CI/CD *Synthetic* AI Testing for Sumo Logic

**Please note that although the resources for GitHub and Sumo Logic are free, if you wish to use the synthetic AI testing you need to pay $20 / month or use a trial for poe.com** . Otherwise just comment out or discard the runner synthetic testing section.

1. Sign up for a Poe.com account subscription
   1. Obtain your key from <https://poe.com/api_key>
   2. Save it somewhere.
2. Install Terraform in your ***local*** host:
   1. **Mac users:** deploy homebrew: <https://brew.sh/>
      1. Follow instructions for installation
      2. Run “brew install terraform”
      3. Run “brew install git”
   2. **Linux users**: Use your package manage install terraform and git
      1. Example: sudo apt update && apt install terraform git
   3. **Windows Users**: <https://community.chocolatey.org/>
      1. Follow instructions for installation
      2. From elevated prompt run: “choco install terraform”
      3. From elevated prompt run: “choco install git”
      4. Python3 **Optionally** install from choco or use the official release: <https://www.python.org/downloads/windows/>
3. Sign up for a Sumo Logic free account: <https://www.sumologic.com/sign-up/>
   1. Note: Trial last 30 days for full features. Thereafter, Observability basic access only goes 7 days back after
   2. Create an **access key** from preferences. You will have an access ID and access key. Save them.
   3. *Optional****:*** configure log ingestion: <https://help.sumologic.com/docs/send-data/installed-collectors/sources/local-windows-event-log-source/>
   4. *Example search test case for enumeration:*

\_collector = "<YOUR-COLLECTOR-NAME>"

| where channel = "Security"

| where eventid = "4798"

| where %"provider.name" = "Microsoft-Windows-Security-Auditing"

A screenshot of a computer

Description automatically generated

1. In **terminal**, update your environment variables:
   1. export TF\_VAR\_SUMOLOGIC\_ACCESS\_ID=<YOUR-ACCESS-ID>’
   2. export TF\_VAR\_SUMOLOGIC\_ACCESS\_KEY='<YOUR-ACCESS-KEY>'
   3. With **sample.tf** uploaded to CloudShell execute the following:
      1. Terraform init
      2. Terraform fmt
      3. Terraform plan
      4. Terraform apply
      5. [Check to make sure it works]
      6. Terrafrom destroy

You have now tested that you can perform the activities for the **Sumo Logic** pipeline CI/CD efforts. We will now setup the pipeline itself using GitHub:

1. Sign up for a free or trial GitHub Account: <https://github.com/>
2. Clone the following github repo or download in full to a zip file: <https://github.com/dc401/cwx-demo-sumo>
3. Create a **new public repo** with your own unique name
4. Visit the link for your repo and add secrets from the keys you captured from y our tools earlier: https://github.com/<YOUR-ALIAS>/<YOUR-REPO-NAME>/settings/secrets/actions

A black and grey line

Description automatically generated with medium confidence

1. Insert your POE and Sumo Logic API keys as repository action secrets
2. Extract the entire cloned repo and upload **all folders** including .github for the runner to install

A screenshot of a computer

Description automatically generated

1. Go to **Actions** at the top of your repo, navigate to ‘sumo-tf-deploy’ and then run the workflow manually

A screenshot of a computer

Description automatically generated

It will take about 30-60 seconds to finish implementing, be patient:

A screenshot of a computer program

Description automatically generated

At completion you will see green checkmarks. Read through the log file of the job to discover how everything operates.

This concludes this part of the lab.

# CI/CD *Integration* Testing with Cloud Custodian

1. Sign up for a GCP account and create a project: <https://cloud.google.com/free?hl=en>
2. Start your free trial credits:

A screenshot of a chat

Description automatically generated

Add payment verification as an individual

A screenshot of a computer

Description automatically generated

* 1. Ensure you secure the account using MFA or you have linked it or signed in with your **Gmail** and *that* has MFA
  2. Consider additional hardening efforts: <https://cloud.google.com/workstations/docs/set-up-security-best-practices>
  3. *Optional:* enabling SSO using Gmail and Cloudflare Zero Trust: <https://medium.datadriveninvestor.com/kickstart-zero-trust-for-free-bbcbeea229c5>

1. Install Cloud Custodian in **your** **local host** terminal: <https://cloudcustodian.io/docs/quickstart/index.html#install-cloud-custodian>
   1. Use “Option 1: Install released packages to local Python Environment”
   2. Follow instructions through “**Environment Variables**”
   3. **Notes:** Warning for Individuals attempting installation and deployments from **Cloud Shell** your CSP may try to disable your account for running terraform and Git that goes outside of your account or project (not sure why GCP decided to do this but suggest doing it on local host):

A screenshot of a computer

Description automatically generated

1. Download and deploy the Gcloud CLI / SDK for your OS: <https://cloud.google.com/sdk/docs/install>
2. Ensure you setup a new profile and are **signed in either with gmail or your account**. Authenticate to your *project id* from the menu prompts. If needed run ‘gcloud init’ once more

A screenshot of a computer

Description automatically generated

1. Authenticate using the Google Cloud SDK shell terminal from your local host: <https://cloud.google.com/docs/authentication/gcloud#local>

A screenshot of a computer

Description automatically generated

1. Create a **unique** cloud storage bucket name to ensure you are good to go:

gsutil mb gs://<YOUR-UNIQUE-BUCKET-NAME>

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

We’ll leave it there for now. It’s time to test a sample Cloud Custodian policy.

1. Ensure Python3 is installed on your host. Run the following commands **while authenticated** to GCP to test out a sample policy:
   1. From within the same shell go ahead and create a new folder
   2. Step into that folder
   3. Create a new file called find-untagged-buckets.yml and populate:

#dry run with >custodian run -d -s . ./find-untagged-buckets.yml -v

#sometimes windows requires manual setting set GOOGLE\_CLOUD\_PROJECT=<PROJECT\_ID> even post auth

policies:

  - name: find-buckets-without-custodian-label

    resource: gcp.bucket

    filters:

      - type: value

        key: "labels.custodian"

        value: absent

        to:

          - stdout

A screenshot of a computer program

Description automatically generated

1. Save the file and then run the following to activate a Python 3 virtual environment:
   1. python 3 -m venv .
   2. Run ./scripts/activate (.ps1 or .bat or source ./bin/activate depending on OS):

A screenshot of a computer

Description automatically generated

Next, run **pip c7n install**:

A screenshot of a computer program

Description automatically generated

We also need ***additional GCP extensions***, let’s install that one now:

Run: **pip3 install c7n\_gcp**

**A computer screen shot of a program

Description automatically generated**

**Note** for Windows users:

#Run: gcloud auth login

#\*Only if needed run: gcloud config set project PROJECT\_ID

#Run: python -m venv .

#Run: ./Scripts/activate.ps1

#Run: pip install c7n c7n-gcp

#Run: custodian run -d -s . ./find-untagged-buckets.yml -v

#If you're getting errors in Windows

#sometimes windows requires manual setting set GOOGLE\_CLOUD\_PROJECT=<PROJECT\_ID> even post auth

#Run: gcloud config list (obtain project id)

#example powershell : $env:GOOGLE\_CLOUD\_PROJECT='xtec-demos-439004'

A screenshot of a computer program

Description automatically generated

Let’s do a test dry run:

custodian run -d -s . ./find-untagged-buckets.yml

A screenshot of a computer program

Description automatically generated

It works. Now we know we’re authenticated using this account and we have the correct API access and permissions.

Let’s continue let’s create a **pub/sub** topic so we essentially can monitor our status actions of the detection.

Use the search bar and go to pub/sub (not pub/subscloud:

A screenshot of a computer

Description automatically generated

A screenshot of a search box

Description automatically generated

Name it something like the following and ensure you add default subscription, enable message retention for up to 1 hour and use the default google managed key:

A screenshot of a computer

Description automatically generated

**Optional** if you plan to push multiple detection tests frequently, lower it to 10 minutes:

A screenshot of a computer

Description automatically generated

**Modify the find-untagged-buckets.yml** to your topic path:

A screen shot of a computer program

Description automatically generated

Now run Custodian in **non dry run** mode to execute the actions and we need to pull the message:

A screen shot of a computer screen

Description automatically generated

Navigate to **Topics** on the left side and then **Messages** in the center, click on **Pull** to pull the last message:

A screenshot of a computer

Description automatically generated

If you were to run in gcloud even in JSON the data itself is a JSON compressed zlib format and then base64 encoded on top of that (**multiple encodings**):

(cwx-demo-gcp) PS C:\Users\dwcho\Downloads\cwx-demo-gcp> gcloud pubsub subscriptions pull --auto-ack --limit=2 github-actions-detection-tests-sub --format="json"

[

{

"ackId": "BhYsXUZIUTcZCGhRDk9eIz81IChFEgEHTwIoXXkwTypBXXUCPg0Zcn1hdGNYQgQLRgEqC1kRDmJcTkQHSfLljfVXV0tbEQcFTFJ8XlwSBW9aXHYGUCWAkqn7s4\_-NRs-fayCoNMtLbK\_4Mk1ZiI9XhJLLD5-NyhFQV5AEkw8HURJUytDCypYEU4EISE-MD5FU0RQ",

"ackStatus": "SUCCESS",

"message": {

"data": "eJzVV9tu2zgQffdXFOpraV0sxRdggfXG6VM2KRIXC-yiWFDUyOGGIgWRtGME\_fcdUrSbNlUbFGnRPhiwOTNnLmfGHN6PItiCNNHihbRCvBpFlDFlpfmXV3gW3RlgpIJGaZJP5kmSRx9UBuUdbLiSTkyFcAetEpzt8eB-FEnagBPVXFaktOwWjCY7bm6UNYRZbVTFqSSCliB6MK1sx7zNhrXj3sRJai4MdBoF\_4wQ2Oxbr7SlwoKT34JzGXkkPT5CO1Gv4wIstct-9H70zidmMPBHiFIZXu-dnVFeFm0Vr37nEmF4Na6Virx5rbqG-rKYOx-h6ajUrepMn\_oBr7WltmWP13Lmjzr1HzCj40cFjb2OjjdYIluSECKqoKL7Sgxoo10GmMPoPYLCHTDrRIFDVkwn0zI\_IZDROcnzOiPzNEtIXaYlVCyZJHURfWSnDfUxp9NsPkHrWT4uptm0KGaudljzQG8yno\_zNDpW7uM0f5WyPWiyI\_W32J3OAfZMRzfw8kPXaRD1OZe3TnpjTKsXcbzb7cYbpTYCaMux1VQTB8N4m8ZljKmWtCNsd-djJNjHDqqnZ0B4nJPPi0PmF7YpoXN6RZakxSwp0tlJ5hQaMOhfQkcDM5EnSih2PHh77RPqIz0VVLv0o-v18mK1vFr5lkAMd3a6PPvNF543cNoBNeBDz5IsJ2lC0tk6KRbJfJGfjCfJ5G-natvqKWqUiS\_XfMmQFn2qpOmU-GLV4lATonaYtiafVuSZmIsx5K-5Cu0yTB\_-6XCzf9DCQ0iYtG-Dy78uzq4-R0mwXwNt-nH5ameYXjXqPYYReM76Q8UR4scQMOTrGxgYgvqOFASX34GDLYfdjxqCIV\_fwMEQ1IGDq7Pl6rlJCD7DFYqCCmpqhbksnf1y-A9KBYVPyPkVZvvJof-MQ\_Hk4H-KbvJU9CBP6wy\_kWhVmxUIXFfePFieO\_wt3fW9sv29fg1Mycrf2ydJPksSn0xduyVnC2ve7xCDF7B3xWmDVaz5xh53hfvD8PbOL6XYHzKgpfC3ek2FBm9vJXcr3B\_e4ByfE6KnZtgCNzmE7bXedP4BElYSLm-g425v8IqHbWUdVsAGx5KT8L5wRLZus4xWZ6-Xb8\_X7oRN5eJPatgNVK8fvA8evwLcsuw-\_wP7xAAv",

"messageId": "12679121798760311",

"publishTime": "2024-10-19T17:03:06.835Z"

}

}

]

So essentially need to do something like this:

<YOUR OUTPUT> | python3 -c "import base64, zlib, sys, json; data = sys.stdin.read().strip(); decompressed = zlib.decompress(base64.urlsafe\_b64decode(data + '=' \* (-len(data) % 4))); print(json.dumps(json.loads(decompressed), indent=2))"

We won’t decode it immediately right now. Now, let’s setup another code repo in **GitHub** call it something unique just like we did in the last repo for deployment. We called ours cwx-demo-gcp but you will need to make up your own name.

Go ahead and clone and download our public repo:

<https://github.com/dc401/cwx-demo-gcp>

Unzip and extract the files to upload to your repo; ensure your directory structure looks like this:

A screenshot of a computer

Description automatically generated

Just like we used Github **Actions** YAML as a runner go ahead and ensure that your actions workflow looks like this:

name: gcp-cc-deploy

# Controls when the workflow will run

on:

push:

branches: [ "main" ]

paths:

- detections/\*.yml

- buildspec.json

pull\_request:

branches: [ "main" ]

# Allows you to run this workflow manually from the Actions tab

workflow\_dispatch:

#when running commit prior to a push you setup the cli parameters that should trigger a detection

#env:

# GCP\_SA\_KEY: ${{ secrets.GCP\_SA\_KEY }}

permissions:

contents: read # This is required for actions/checkout

id-token: write #GCP Gcloud requires this

jobs:

CloudCustodianDeploy:

# The type of runner that the job will run on

runs-on: Ubuntu-latest

environment: production

# Use the Bash shell regardless whether the GitHub Actions runner is ubuntu-latest, macos-latest, or windows-latest

defaults:

run:

shell: bash

steps:

# Checks-out your repository under $GITHUB\_WORKSPACE, so your job can access it

- uses: actions/checkout@v4

- uses: google-github-actions/auth@v2

with:

credentials\_json: '${{ secrets.GCP\_SA\_KEY }}'

- uses: google-github-actions/setup-gcloud@v2

#deploy python and cache requirements

- name: Setup Python Packages

uses: actions/setup-python@v5

with:

python-version: '3.10'

cache: 'pip'

- name: Install python dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

#cc-deployer runs cloud custodian unit tests as dry runs and integration tests with runs that then push a pub/sub message

#validation of the pub/sub message is based on policy name matching the message requries decoding JSON serialized zlib compression and then base64

#https://cloudcustodian.io/docs/gcp/policy/genericgcpactions.html#notify

- name: Run cc-deployer.sh

run: |

python cc-deployer.py buildspec.json

Note that we’re following the same pattern as before with minor changes to our use case.

Our main difference is instead of terraform we are using the python3 deployment script as a master wrapper that will sub run another python helper script.

Cloud Custodian has its own form of **unit testing** called ‘dry run mode’ which it will still enumerate the target cloud and then evaluate the policy and give you a response.

For the sake of **integration**  testing we can leverage its **actions** by checking GCP’s pub/sub and decoding the message data that show that the **policy** was ran:

<https://github.com/dc401/cwx-demo-gcp/blob/main/cc-deployer.py>

In cc-deployer.sh we **iterate** through the buildspec.json using a for loop looking for elements in the detection key as a list and then iterating through each policy one by one with each test.

The **integration test itself** is checked using a 2nd helper python script: <https://github.com/dc401/cwx-demo-gcp/blob/main/gcp-cc-integration-test.py>

The JSON that comes out of your test looks like this:

<https://github.com/dc401/cwx-demo-gcp/blob/main/_examples/pubsub-message-sample.json>

To decode it the cloud custodian documentations says it’s a base64 encoded zlib json . Here’s the catch it uses base64 ‘url safe’ characters instead of the full character set which is what trips up most decoders. Lets look at it in cyber chef:

<https://cloudcustodian.io/docs/gcp/policy/genericgcpactions.html>

https://cyberchef.io/#recipe=From\_Base64('A-Za-z0-9-\_',true)Zlib\_Inflate(0,0,'Adaptive',false,false)JSON\_Beautify('%20%20%20%20',false)&input=

Copy the full URL above or use your own copy paste input:

A screenshot of a computer

Description automatically generated

Select URL safe

Then add the rest of the zlib inflate, and json beautify to see the decoding:

A screenshot of a computer

Description automatically generated

Notice we have my project ID details and other runtime evaluation items. I just care about the policy name. That will be the thing we extract. Our pythoin script does this for us and then raises an **exception error** to quit the pipeline if error:

<https://github.com/dc401/cwx-demo-gcp/blob/main/gcp-cc-integration-test.py>

Although we may have our pub/sub and bucket configured to work. We need **to create a service account and key** that will be our JWT credential file for the GitHub ephemeral runner to use.

Return to your GCP console go to the **IAM** service and then the ‘**Service Accounts**’ menu on the left:

A screenshot of a service account

Description automatically generated

Create a new service account:

A screenshot of a computer

Description automatically generated

You’ll then name it something meaningful. We set ours to cwx-demo-gcp-github:

A screenshot of a service account

Description automatically generated

**Ensure** you **GRANT** the service account access to the project as an **OWNER** role.

Keep in mind that in the **real world you will be using LIMITED scope based on the CI runner permission needs AND you will be using OIDC federation** <https://docs.github.com/en/actions/security-for-github-actions/security-hardening-your-deployments/configuring-openid-connect-in-google-cloud-platform>

We are not doing that today for simplification. If you made the right grant access your service account permissions should look like this:

A screenshot of a computer

Description automatically generated

Now go to the **KEYS** menu in the center under the service account

A screenshot of a computer

Description automatically generated

Create a new **JSON** key:

A screenshot of a computer

Description automatically generated

You will then **save the JSON in a safe place**. This is like PERMANENT ROOT ACCESS TO YOUR GCP PROJECT!

A screenshot of a computer

Description automatically generated

Return to the GitHub **Settings** for your repo Secrets and Variables -> Actions

A screenshot of a computer

Description automatically generated

Paste the contents of the JWT that you just downloaded key and call it **GCP\_SA\_KEY:**

A black and grey background with white lines

Description automatically generated with medium confidence

IMPORTANT: ENSURE THAT IT’S IN YOUR **SECRETS ACTIONS**. Otherwise you are risking exposure.

So we have our pub/sub, our cloud storage bucket go ahead and run the flow like we did last time and you should see our log output:

You’ll be on average 10 seconds faster after the first runtime because of PIP caching:

A screenshot of a computer

Description automatically generated

Open your actions and let’s see what it did:

A screenshot of a computer program

Description automatically generated

As expected our dry run, which is our **unit** test is fine, and then after running cloud custodian in full a pub/sub message that we then pull and decode matches the policy expected there by passing the integration test and then we take no further action.

In the real world you might run some additional clean up some additional items or use a canary strategy that has a mock policy like this and then skips testing the other policies. It depends on the needs and compute time you’re willing to pay for.

# Did you learn something?

Feel free to continue following my community projects and other work:

<https://dwchow.medium.com/>

<https://github.com/dc401>

<https://www.linkedin.com/in/dwchow/>