Workshop Guide: Automating Threat Hunting with Detection As Code

Authors: Dominic Chua, Sumit Patel
Google Cloud Security

References used within this guide:
Getting Started with Detection-as-Code and Chronicle Security Operations (Part 1 of 2)
AI Runbooks for Google SecOps: Security Operations with Model Context Protocol
Building an open ecosystem for AI-driven security with MCP

Walkthrough Recording

Youtube link: *https://youtu.be/gQ2CgWEtIYA*

Table of Contents

Prerequisites

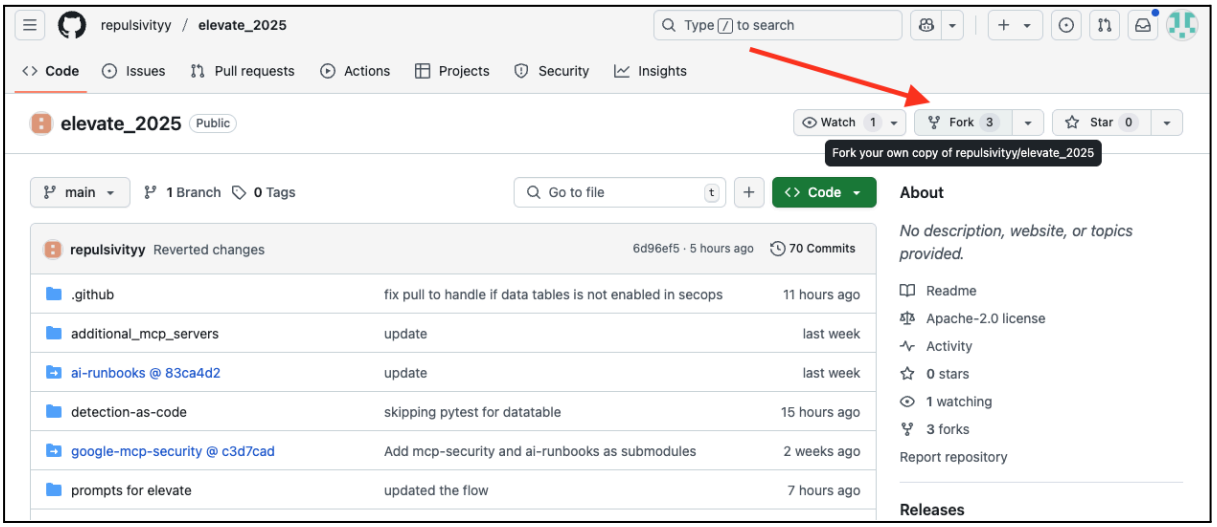
1. Visual Studio Code
2. Installed in your local terminal:
    a. Python
    b. uv
    c. gcloud CLI
3. A GitHub account
4. A Google Cloud Project / Argolis

Setting up Detection-As-Code on GitHub

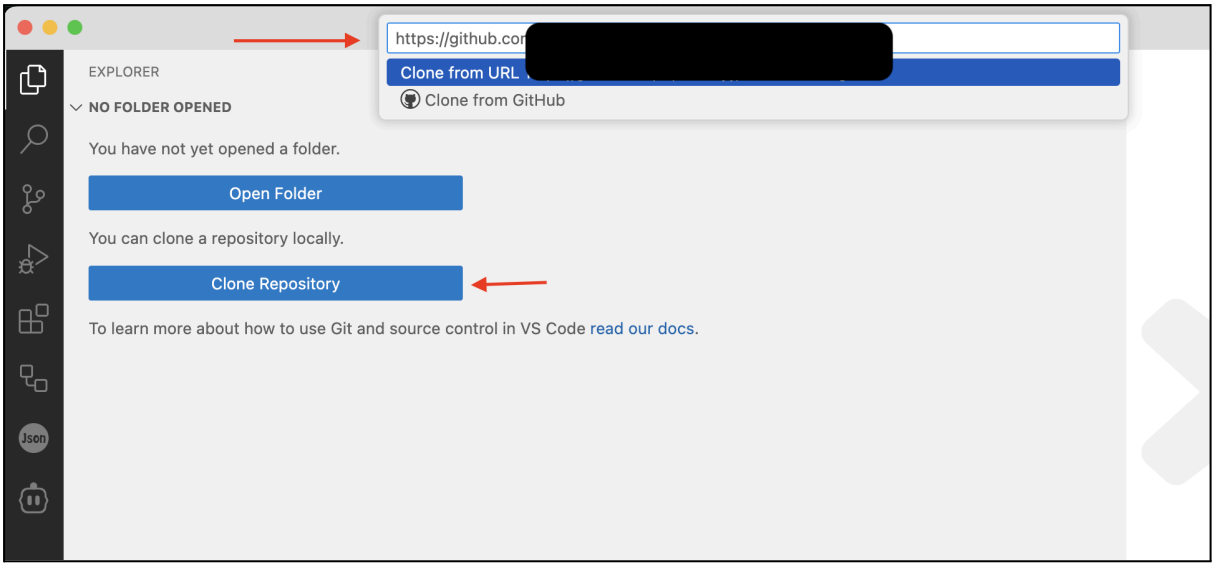
Setting up Github

1. Set up (if you don't have one) and log into a Github account

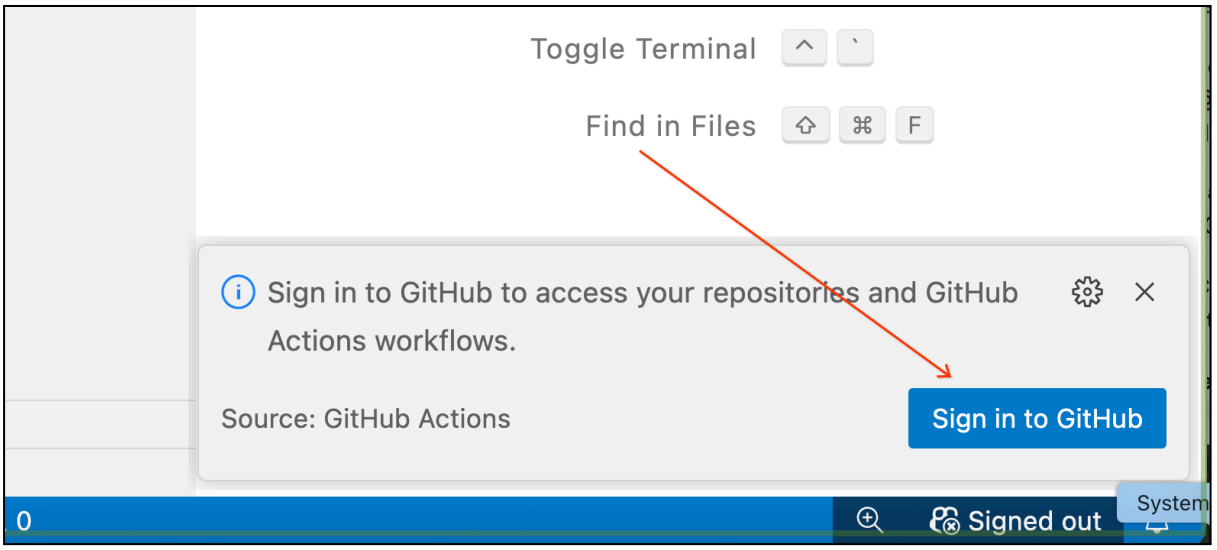
2. Start by creating a fork of this repo https://github.com/repulsivityy/elevate_2025 into your own repo *e.g. https://github.com/[YOUR_REPO]/elevate_2025*



3. Open up VS Code, and "**Clone Repository**" YOUR repo you just forked (this will clone your repo to your own



4. Once you clone, you should also be able to select “**Sign in to GitHub**”

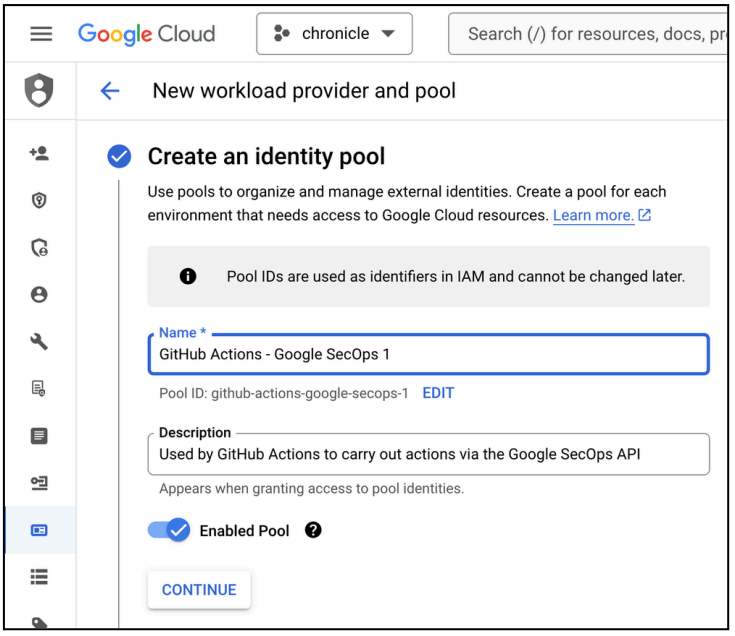


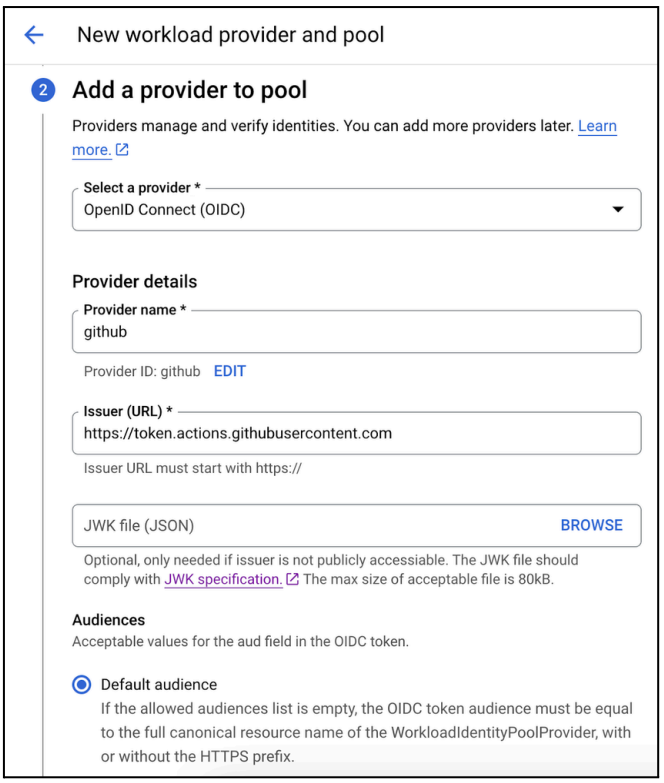
Once you clone, you should also be able to select “**Sign in to GitHub**”

Setting up Workload Identity Federation in GCP (So we don’t need to use Service Account keys!)

1. Goto GCP and your Google SecOps project. Search for **Workload Identity Federation** and open it up.

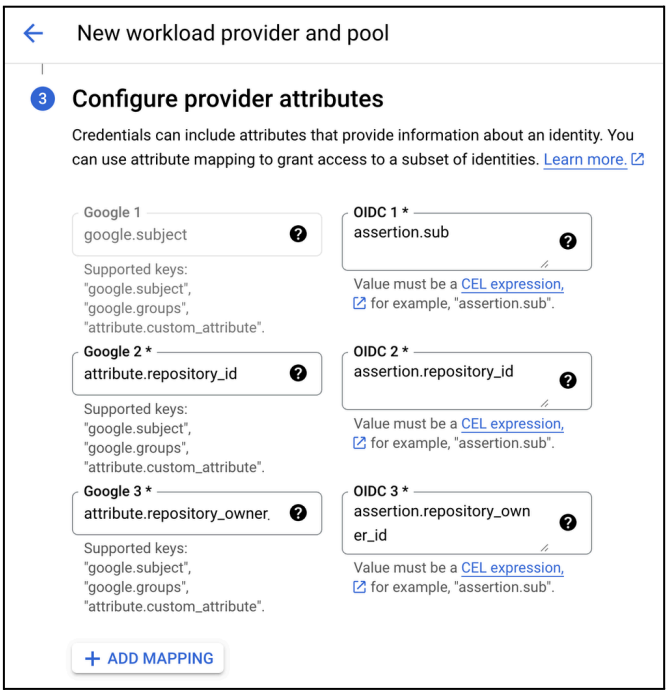
2. Create a new **Workload Identity Pool** and **Provider**





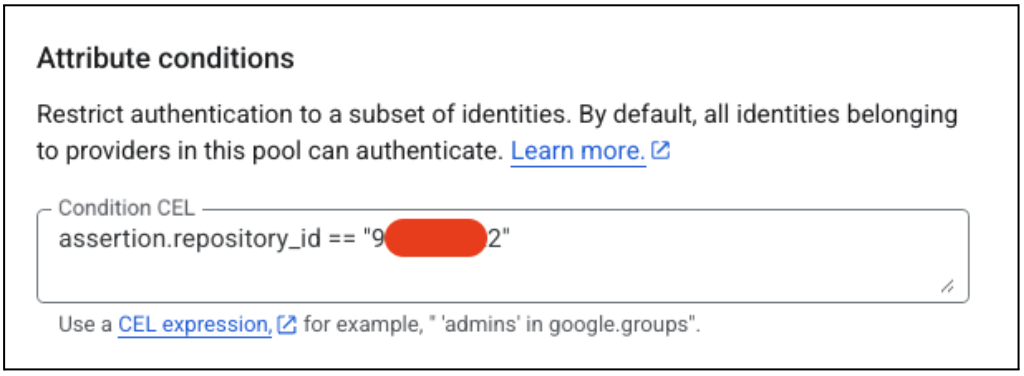
Issuer (URL): **https://token**.actions.githubusercontent.com

3. Add the attribute mappings



google.subject | assertion.sub
attribute.repository_owner | assertion.repository_owner_id
attribute.repository_id | assertion.repository_id

4. Add the Attribute Conditions



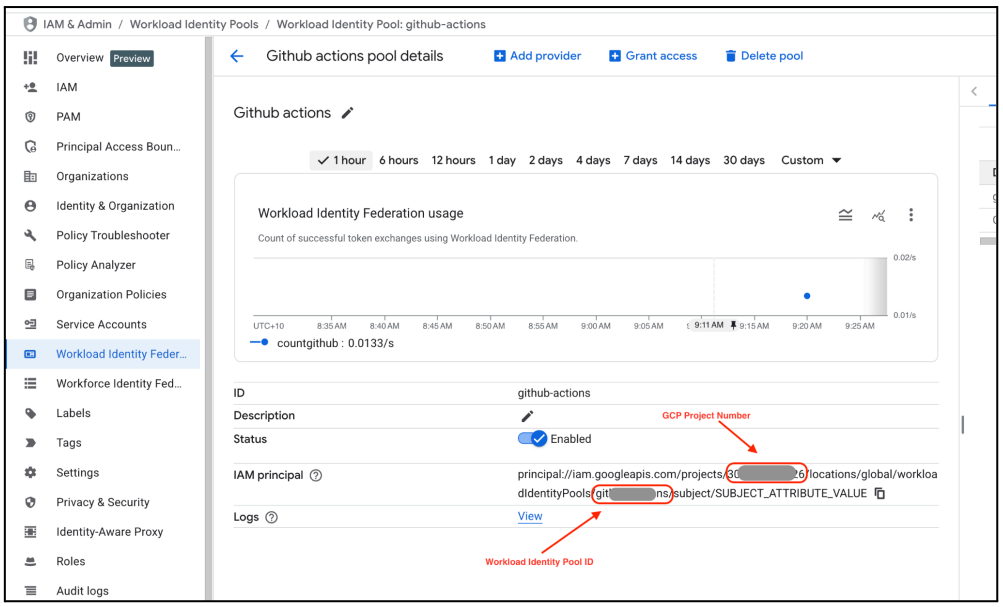

```
assertion.repository_id == "[REPO-ID]"
```

To find your Github Repo ID:

- *View the page source in Github (Right Click > View Page Source in Chrome or Firefox for example)*
- *Search the page source and look for* `octolytics-dimension-repository_id`*. You should find something that looks like:*
  `<meta content="123456789" name="octolytics-dimension-repository_id" />`
- *In this example, the ID of the repository is* `123456789`*.*

5. **Save** your new Workload Provider.

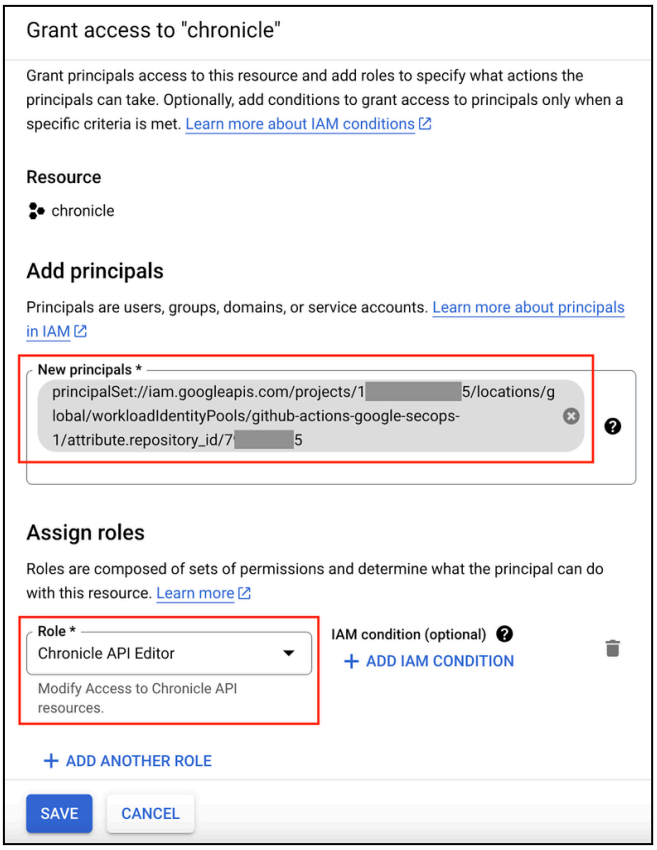
6. In the GCP console goto Workload Identity Federation -> [Your Pool]. Locate your **Workload Pool ID**, **GCP Project Number** as per screenshot below and **Repo ID** from Step 4.



7. Goto IAM

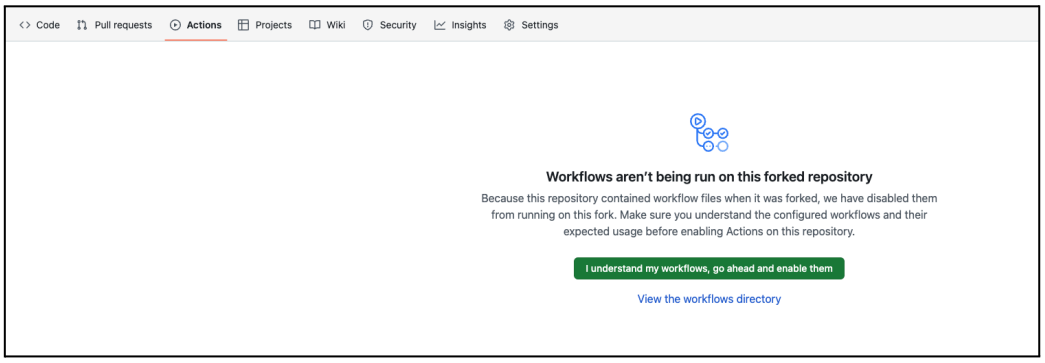
8. In IAM, **Grant access** to the principalSet in your project. Replace **GCP Project Number**, **PoolID** and **RepoID** as noted in step 6.

principalSet://iam.googleapis.com/projects/[GCPProjectNumber]/locations/global/workloadIdentityPools/[POOL-ID]/attribute.repository_id/[REPO-ID]

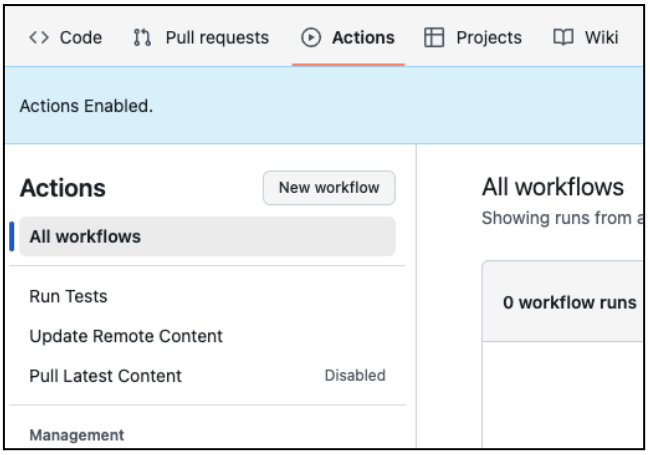


Configure GitHub Actions & Variables/Secrets

1. Go back into Github and choose **Actions**, and choose "**I understand my workflows, go ahead and enable them**"

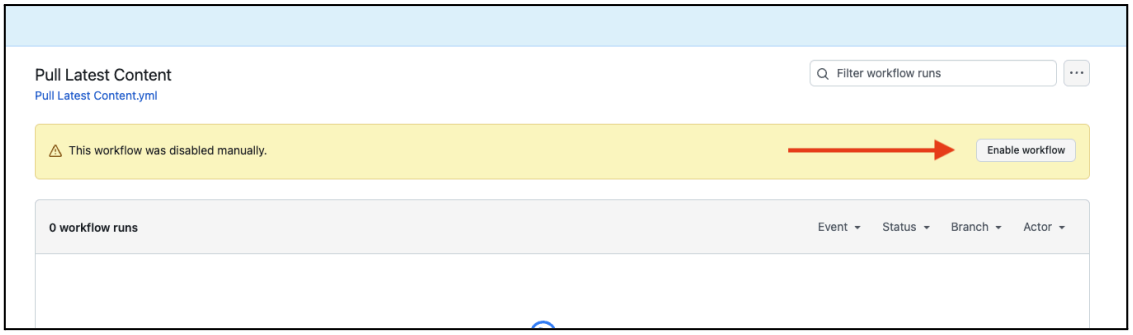


2. Confirm that the 3 Github actions are listed already as below

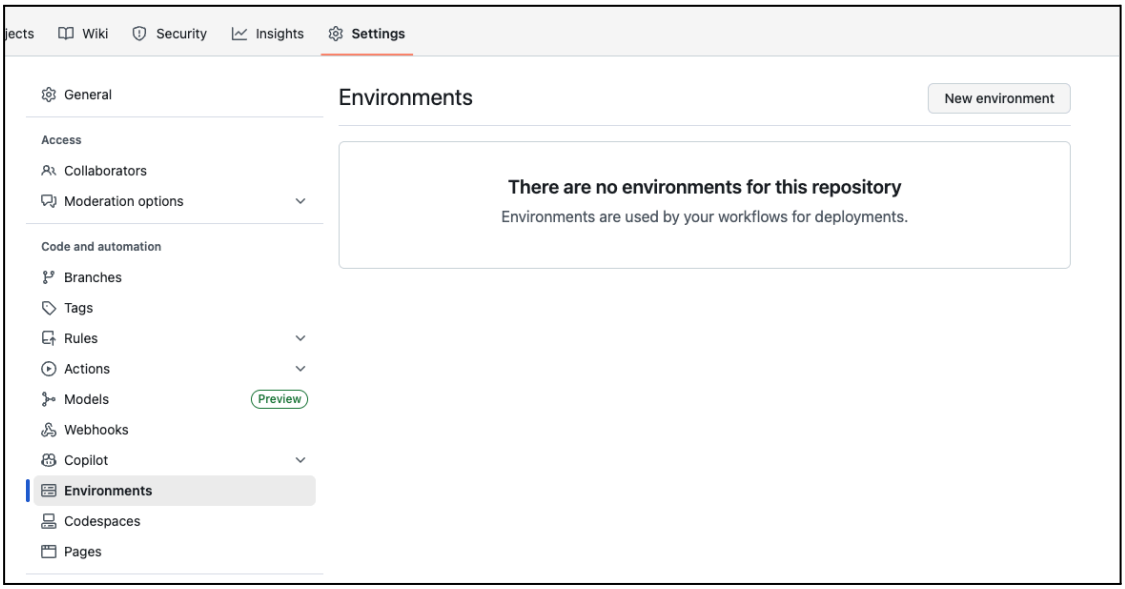


- Pull Latest Content
- Run Tests
- Update Remote Content

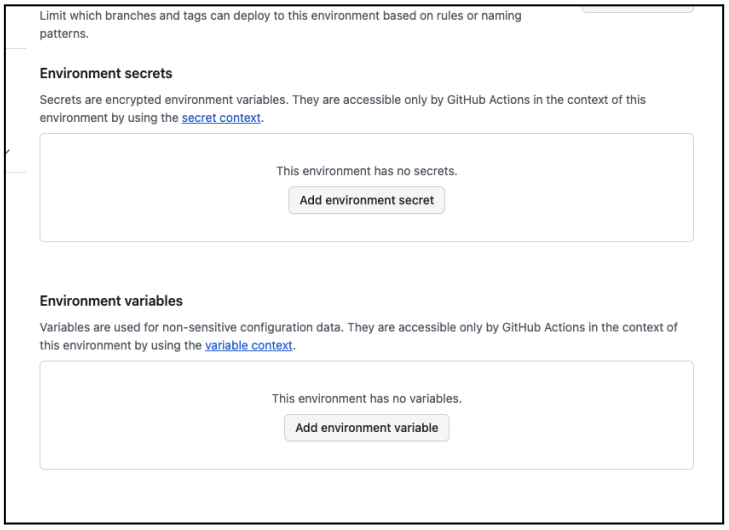
3. You will need to enable "**Pull Latest Content**" github action. Select "**Enable workflow**"



4. Now we need to save Environment variables and secrets needed for Github Actions to run.
    a. In Github goto **Settings** and choose "**Environments**"
    b. Now choose to create a **New environment**, and name the environment as "**.env**"



5. Now "**Add environment variables**" and "**Add environment secret**"



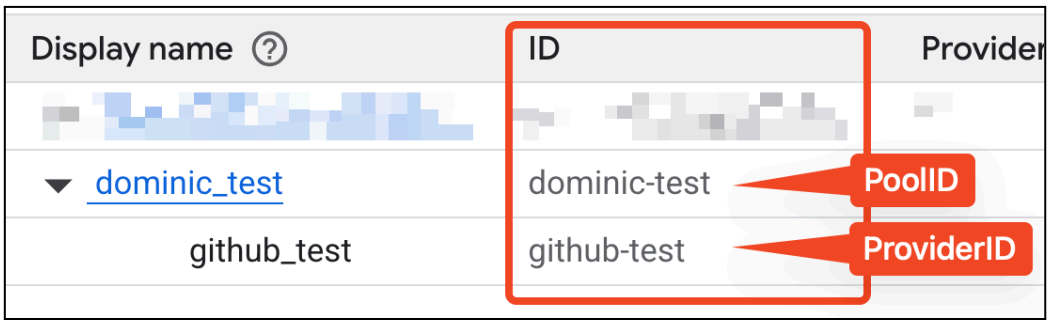
6. Variables to create (with example **Name** and **Value)**. Make sure to replace accordingly i.e. region specifics:

Name	**Value**
AUTHORIZATION_SCOPES	{"GOOGLE_SECOPS_API":["https://www.googleapis.com/auth/cloud-platform"]}
GOOGLE_CLOUD_PROJECT_ID	<GCP Project ID> e.g. *myproject-373104*
GOOGLE_SECOPS_API_BASE_URL	https://<regionid>-chronicle.googleapis.com/v1alpha *e.g.: https://us-chronicle.googleapis.com/v1alpha* *The following are the supported region IDs: asia-southeast1, australia-southeast1, europe, eu, europe-west2, europe-west3, europe-west6, govcloud-US, me-west1, and us.*
GOOGLE_SECOPS_API_UPLOAD_BASE_URL	https://<regionid>-chronicle.googleapis.com/upload/v1alpha *e.g.: https://us-chronicle.googleapis.com/upload/v1alpha* *The following are the supported region IDs: asia-southeast1, australia-southeast1, europe, eu, europe-west2, europe-west3, europe-west6, govcloud-us, me-west1, and us.*
GOOGLE_SECOPS_INSTANCE	projects/<GCP Project ID>/locations/<regionID>/instances/<secops-customer-id> e.g.: *projects/myproject-373104/locations/us/instances/bc12345-8692-4184-a40d-6c12ff58a350* *The following are the supported region IDs:* - *us* - *eu* - *asia-southeast1* - *australia-southeast1* - *europe-west2* - *europe-west3* - *europe-west6* - *govcloud-us* - *me-west1*
LOGGING_LEVEL	INFO

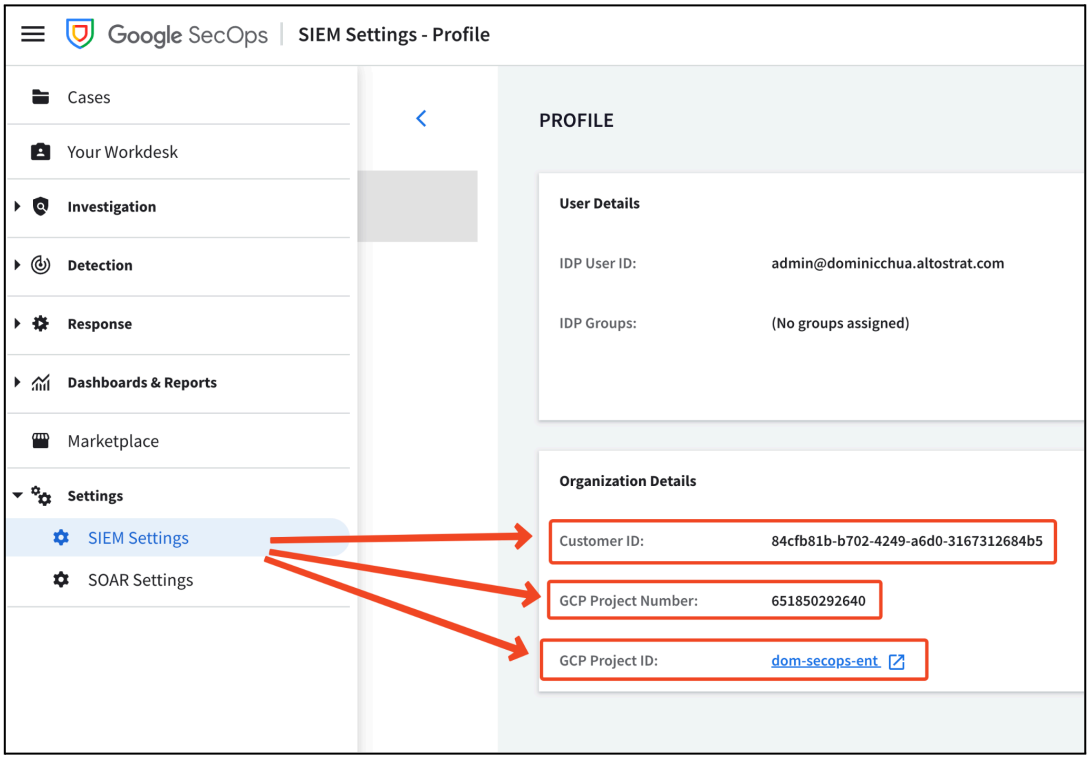
7. Create the following secret

GOOGLE_CLOUD_WORKLOAD_IDENTITY_PROVIDER	projects/<GCP Project **number**>/locations/global/workloadIdentityPools/<PoolID>/providers/<ProviderID> *e.g.* *projects/123456789012/locations/global/workloadIdentityPools/mypoolid/providers/myproviderid* *Note:* - *Use GCP Project **NUMBER** for this instance, not Project ID!* - *Make sure there are no spaces after copy pasting!*

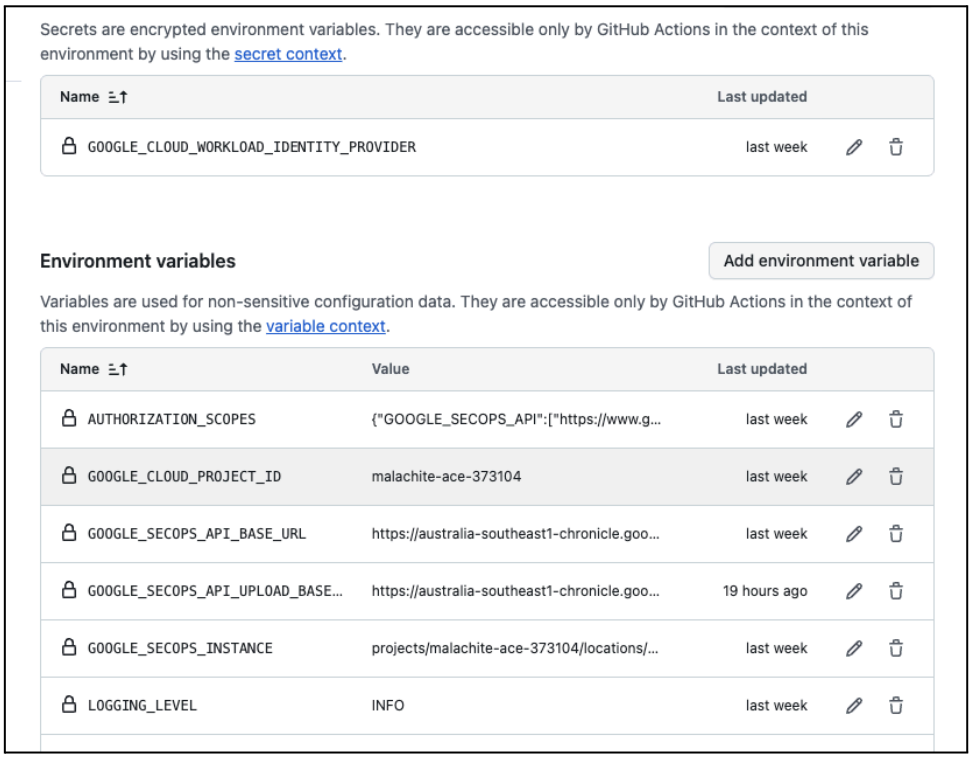
Note the Pool ID and Provider ID e.g.



You can get your customer ID and GCP Project ID and Number from SecOps



8. You should end up with variables and secret as below:



Secrets are encrypted environment variables. They are accessible only by GitHub Actions in the context of this environment by using the secret context.

Name	Last updated
GOOGLE_CLOUD_WORKLOAD_IDENTITY_PROVIDER	last week

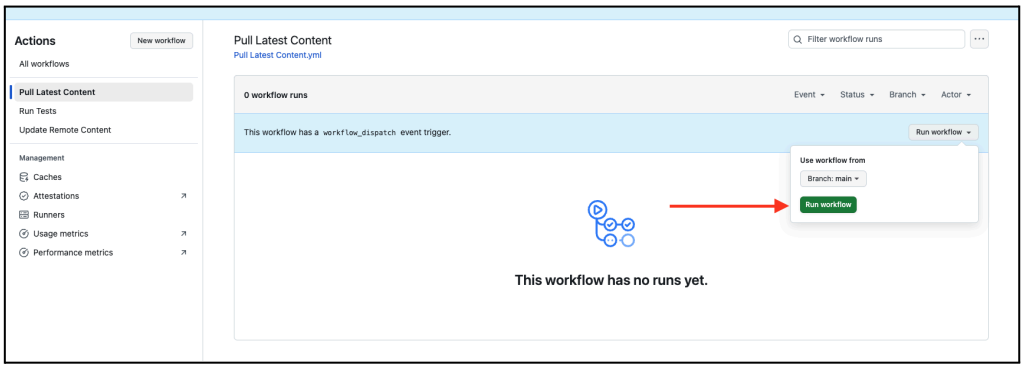
Environment variables Add environment variable

Variables are used for non-sensitive configuration data. They are accessible only by GitHub Actions in the context of this environment by using the variable context.

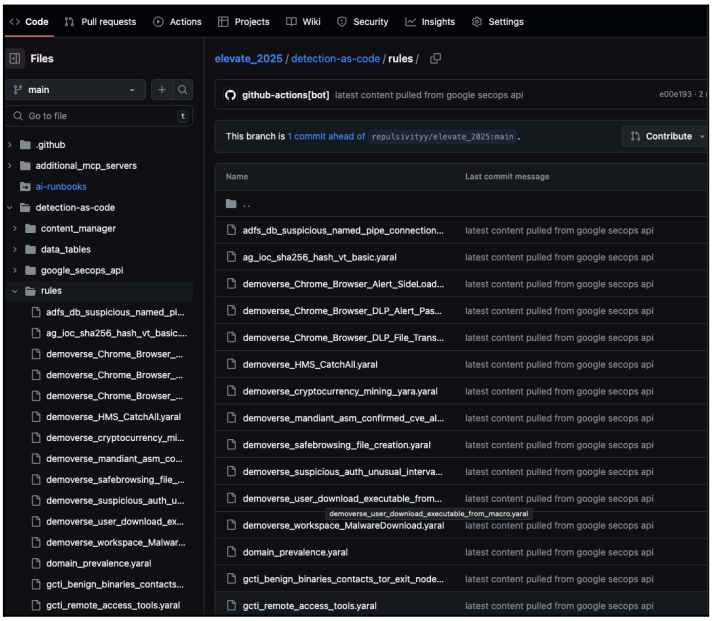
Name	Value	Last updated
AUTHORIZATION_SCOPES	{"GOOGLE_SECOPS_API":["https://www.g...	last week
GOOGLE_CLOUD_PROJECT_ID	malachite-ace-373104	last week
GOOGLE_SECOPS_API_BASE_URL	https://australia-southeast1-chronicle.goo...	last week
GOOGLE_SECOPS_API_UPLOAD_BASE...	https://australia-southeast1-chronicle.goo...	19 hours ago
GOOGLE_SECOPS_INSTANCE	projects/malachite-ace-373104/locations/...	last week
LOGGING_LEVEL	INFO	last week

Testing the Github Actions - 1. Pull Latest Content

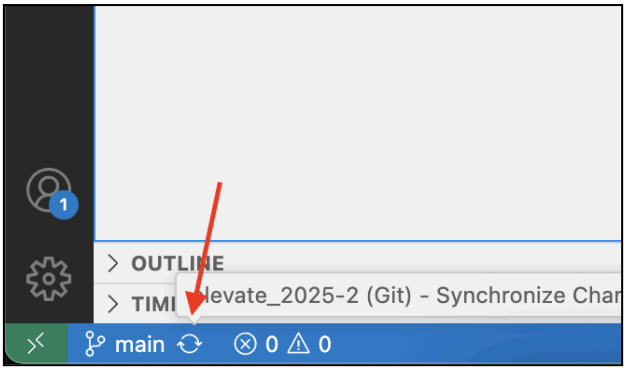
1. Go back to Github - **Actions**. Lets now kick off the "**Pull Latest Content**" action to pull down rules / data tables / reference lists. Go into Actions, Choose "**Run workflow**".



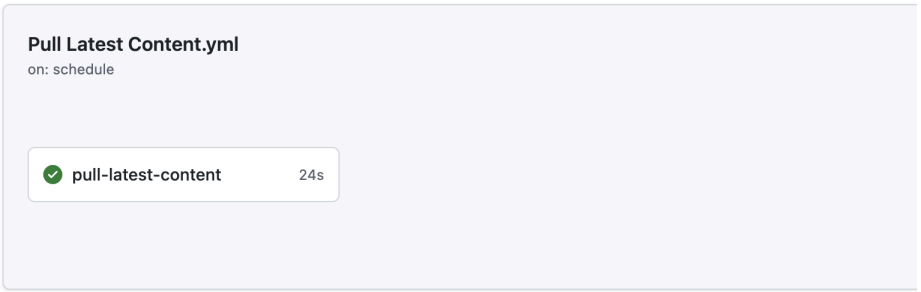
2. On completion of the above action, in your Github repo and in your local repo in VS Code (once it syncs), you should see all your rules
3.



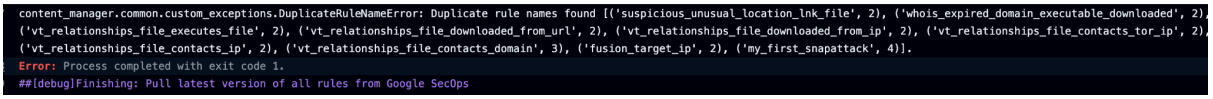
4. You can now try testing Detection as Code! Lets sync the content in VS Code with our repo as shown below:



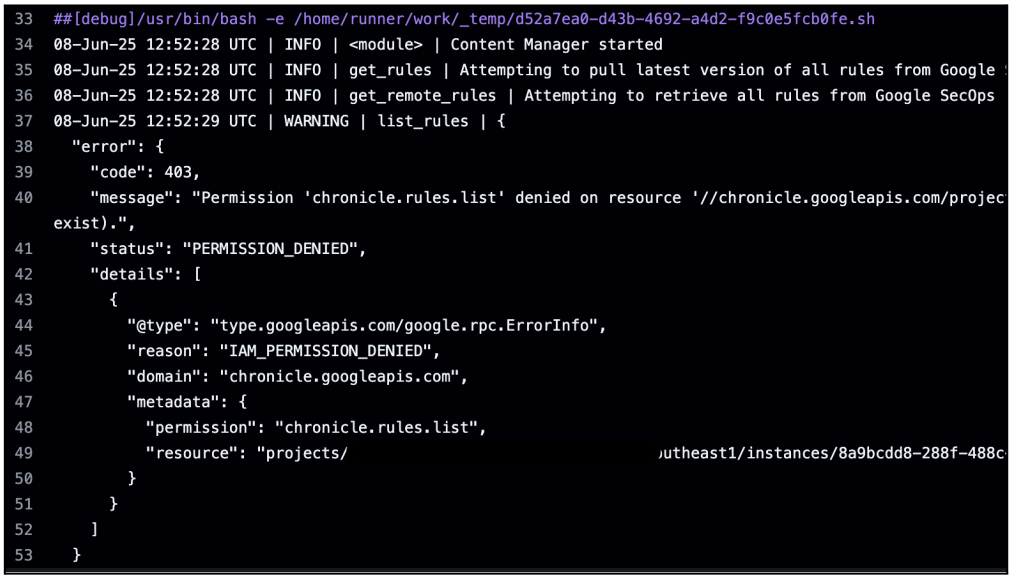
5. If successful, you should see the rules within the detection-as-code -> rules folder. Also, your "*pull-latest-content*" action should have completed successfully.



Note 1: *If you face the following error "DuplicateRuleNameError", you will need to rename your duplicate rules so they no longer are any duplicates!*

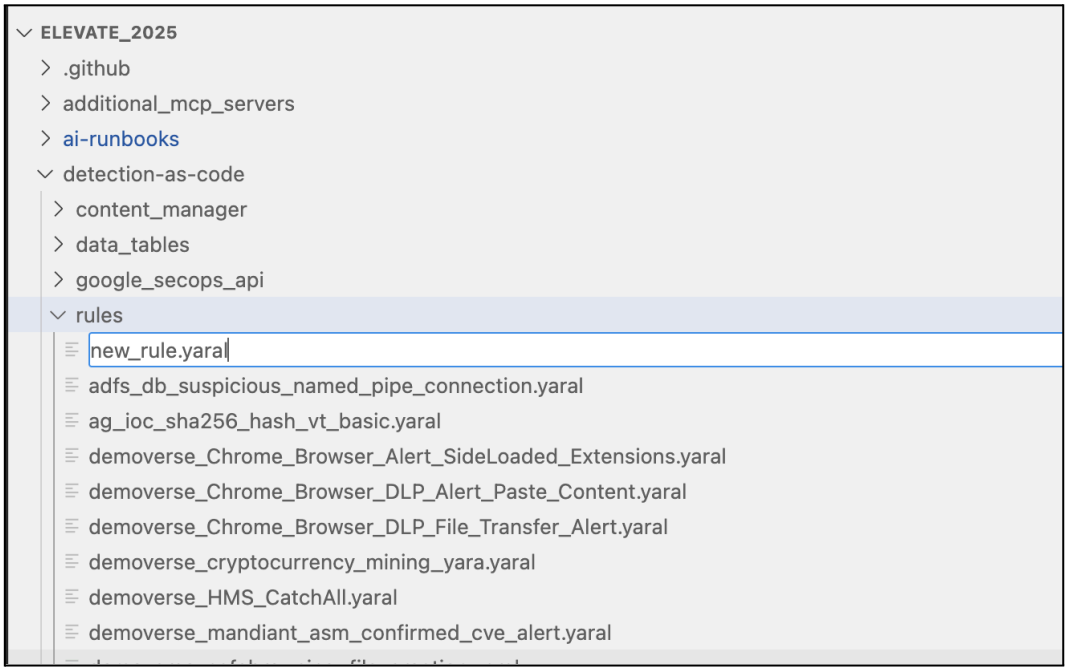


Note 2: *If you face the following error "IAM_PERMISSION_DENIED" or "Forbidden", there is something wrong with your Secret or variables or IAM permissions. Double check, they're correct (e.g. no spaces).*



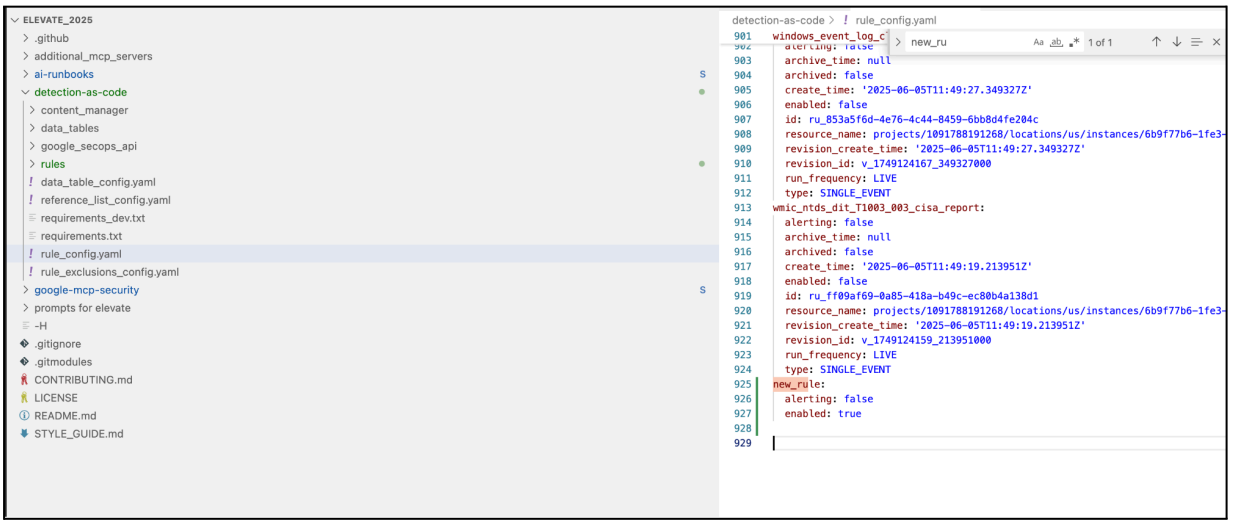
Testing the Github Actions - 2. Creating a new rule

6. Now let's create a new yara-l rule. Go into VS Code, look under `detection-as-code -> rules` folder. Create your new rule in this folder.



7. Hit save on the file, after adding your new rule.

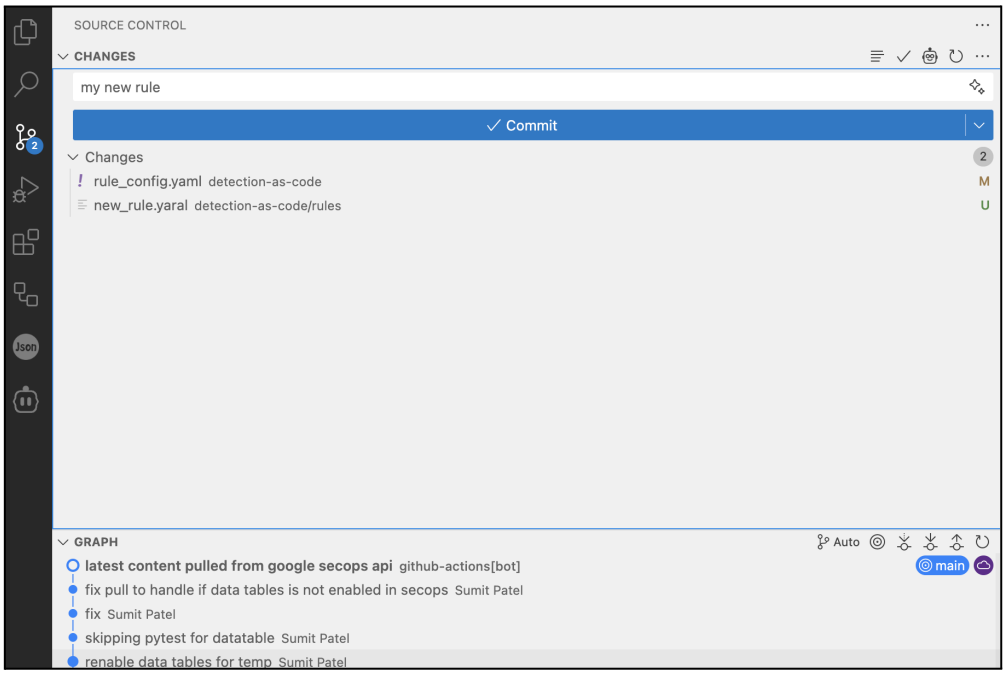
8. Now add an entry to `detection-as-code -> rule_config.yaml` file and save.



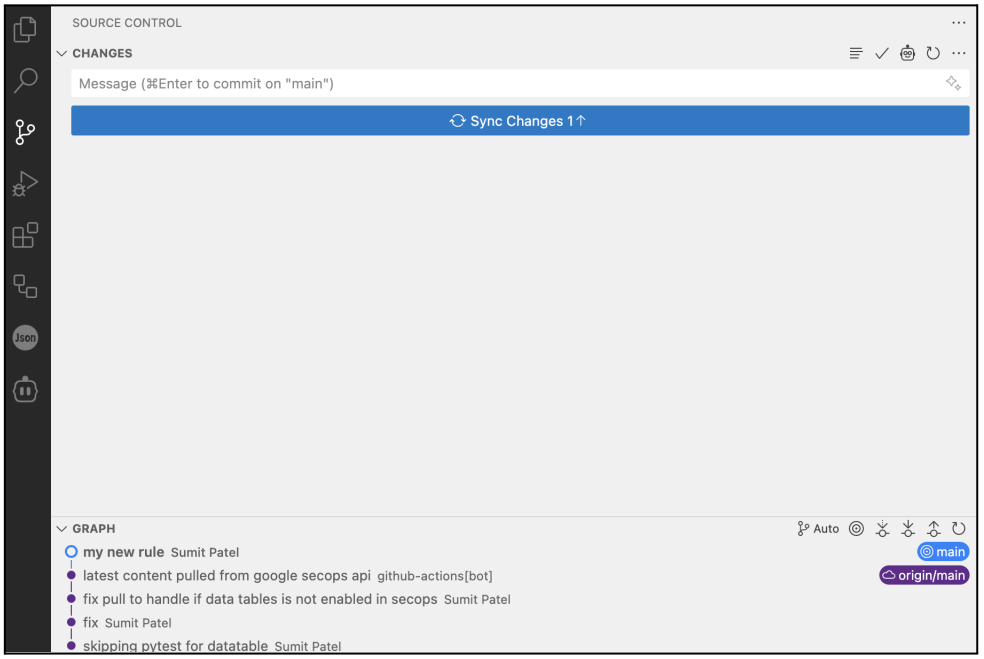
Example:

```
new_rule:
  alerting: false
  enabled: true
```

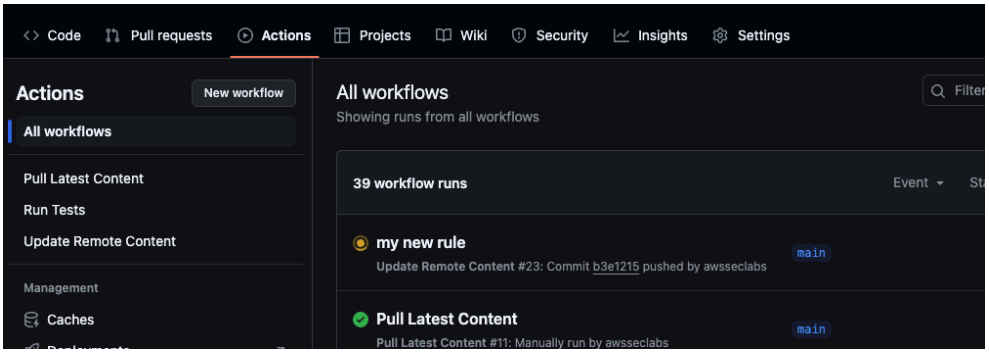
9. Hit the “Source Control” menu item on the left, and add a comment e.g. “my new rule” and hit “**Commit**”.



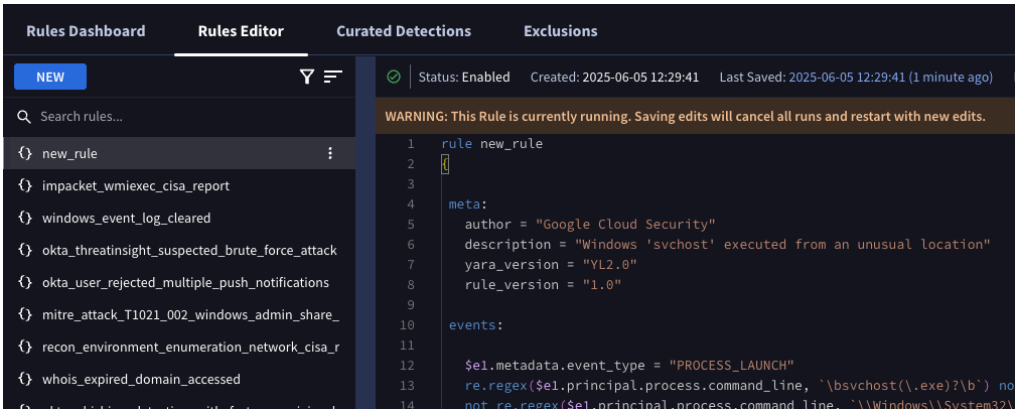
10. Let's now “**Sync changes**”.



11. It now should kick off a bunch of tests via GitHub Actions,



12. On successful completion, you should see your new rule in Google SecOps.



Note: the rules_config.yaml file should also now be updated with additional fields about our new yara-l rule e.g. its ID, creation time etc.

Appendix: Best Practices working with Git

Commit early and often (and Small!)

- **Atomic Commit**s: Each commit should represent a single, logical change. If you're fixing two different bugs, make two separate commits. If you refactor code and add a new feature, do them in separate commits. This makes it much easier to understand, review, and revert changes if needed.
- **Frequent Commits**: Don't wait until you've completed a huge chunk of work. Commit small, testable increments. This reduces the risk of large, messy merge conflicts and allows you to easily track your progress.

Write Good Commit Messages

- **Clear and Concise Subject Line:** The first line of your commit message should be a brief, imperative summary (around 50 characters). It should answer "What did this commit do?" (e.g., "Fix: Login button not responding").

Stay Up-to-Date and Manage Conflicts

- **Pull Before You Push:** Always pull the latest changes from the remote repository (git pull or git fetch and then git rebase/git merge) before pushing your own changes. This helps you integrate upstream changes early and resolve conflicts locally.
- **Understand git merge vs. git rebase:**
    - **git merge:** Combines histories, creating a merge commit. This preserves the exact history of branches.
    - **git rebase:** Rewrites history by moving your commits "on top" of the target branch. This creates a clean, linear history, but **never rebase a public branch** that others are already working on, as it can cause significant issues for collaborators. Use it for your local feature branches before merging into main.
- **Resolve Conflicts Promptly:** When merge conflicts occur, don't panic. Git highlights the conflicting sections. Understand the changes from both sides and resolve them carefully.

Keep Your Repository Clean

- **.gitignore File:** Use a .gitignore file to prevent untracked files (like compiled code, temporary files, IDE configurations, node_modules, sensitive API keys, etc.) from being accidentally committed.
- **Clean Up Old Branches:** Once a feature branch is merged and no longer needed, delete it from both your local and remote repositories to keep the branch list clean.

- **Regular Maintenance:** Periodically review and clean up your commit history if necessary (e.g., using git rebase -i to squash small, related commits *on your local, unpushed branches*).