

# Workshop Guide: Automating Threat Hunting with Detection As Code

*Authors: Dominic Chua, Sumit Patel  
Google Cloud Security*

*Reference 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](#)

## Table of Contents

<b>Prerequisites</b>	<b>3</b>
<b>Setting up Detection-As-Code on GitHub</b>	<b>4</b>
Setting up Github	4
Setting up Workload Identity Federation in GCP (So we don't need to use Service Account keys!)	6
Configure GitHub Actions & Variables/Secrets	10
Testing the Github Actions - 1. Pull Latest Content	14
Testing the Github Actions - 2. Creating a new rule	17
<b>Best Practices working with Git</b>	<b>19</b>
Commit early and often (and Small!)	19
Write Good Commit Messages	19
Stay Up-to-Date and Manage Conflicts	20
Keep Your Repository Clean	20

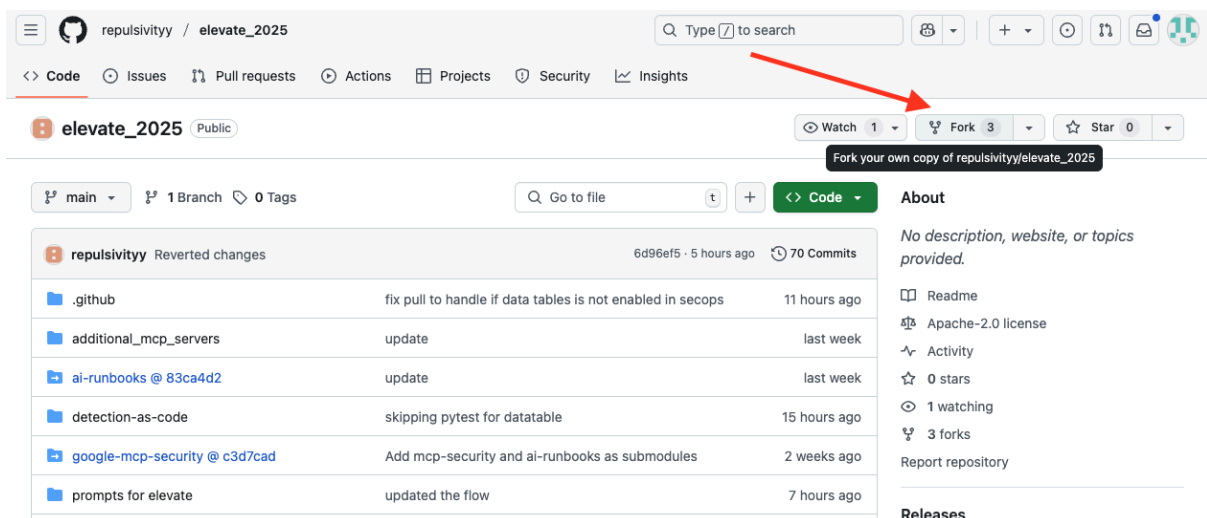
# Prerequisites

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

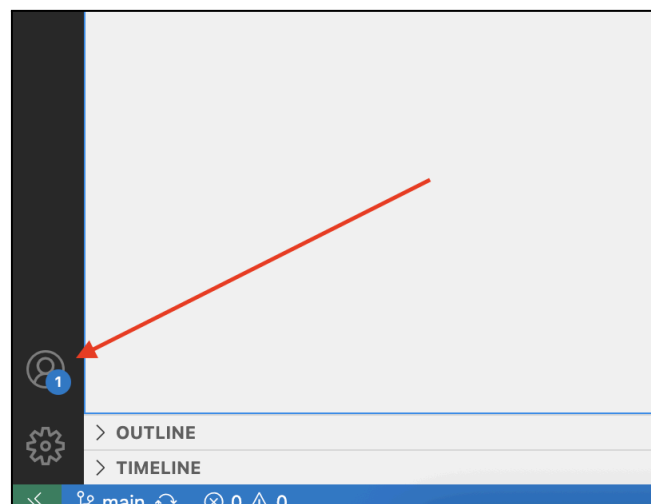
# Setting up Detection-As-Code on GitHub

## Setting up Github

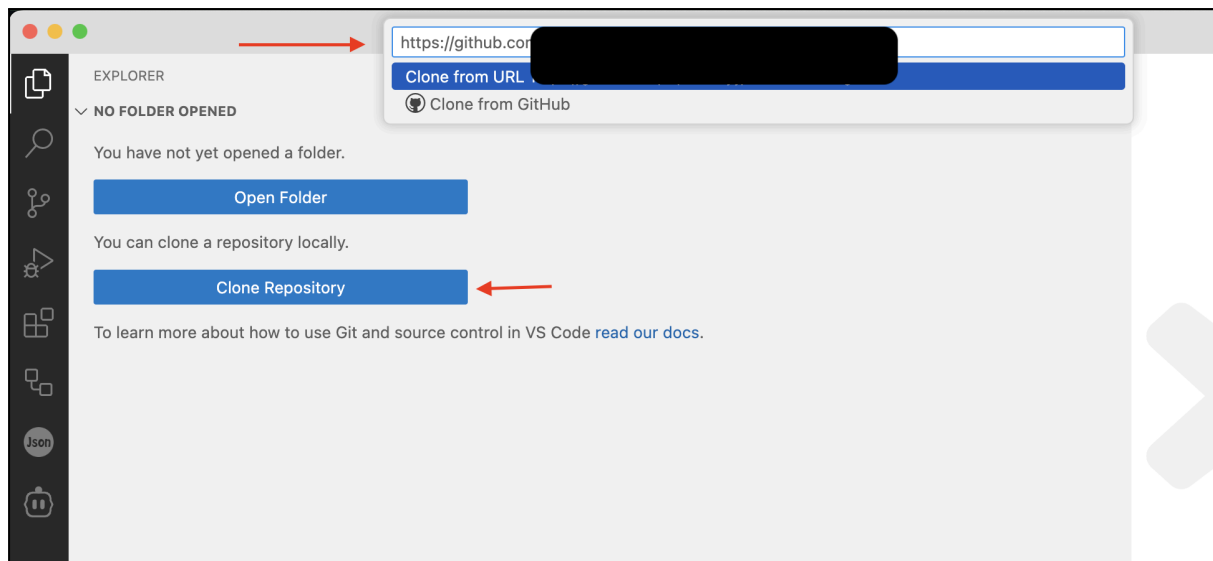
1. Set up (if you don't have one) and log into a Github account
2. Start by creating a fork of this repo [https://github.com/repulsivyyy/elevate\\_2025](https://github.com/repulsivyyy/elevate_2025) into your own repo e.g. [https://github.com/\[YOUR\\_REPO\]/elevate\\_2025](https://github.com/[YOUR_REPO]/elevate_2025)



3. Back in VS Code, sign in to your Github account that you created. This will make executing git commands such as commits, push, pull simpler.



4. Open up VS Code, and “**Clone Repository**” YOUR repo you just forked (this will clone your repo to your own)



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

1. Create the **Workload Identity Pool** and **Provider** (use your Google SecOps GCP project).

The screenshot shows the Google Cloud console interface for creating a new workload provider and pool. The breadcrumb trail is 'New workload provider and pool'. The main heading is 'Create an identity pool' with a blue checkmark icon. Below the heading is a descriptive paragraph: 'Use pools to organize and manage external identities. Create a pool for each environment that needs access to Google Cloud resources. [Learn more.](#)' A grey information box states: 'Pool IDs are used as identifiers in IAM and cannot be changed later.' The 'Name' field is required and contains the text 'GitHub Actions - Google SecOps 1'. Below the name field, the 'Pool ID' is shown as 'github-actions-google-secops-1' with an 'EDIT' link. The 'Description' field contains the text 'Used by GitHub Actions to carry out actions via the Google SecOps API'. Below the description, it says 'Appears when granting access to pool identities.' The 'Enabled Pool' toggle is turned on, indicated by a blue checkmark and a question mark icon. At the bottom is a 'CONTINUE' button.

Google Cloud chronicle Search (/) for resources, docs, pr

← New workload provider and pool

✓ **Create an identity pool**

Use pools to organize and manage external identities. Create a pool for each environment that needs access to Google Cloud resources. [Learn more.](#)

**i** Pool IDs are used as identifiers in IAM and cannot be changed later.

**Name \***  
GitHub Actions - Google SecOps 1

Pool ID: github-actions-google-secops-1 [EDIT](#)

**Description**  
Used by GitHub Actions to carry out actions via the Google SecOps API

Appears when granting access to pool identities.

☒ **Enabled Pool** ?

[CONTINUE](#)

←

New workload provider and pool

2

Add a provider to pool

Providers manage and verify identities. You can add more providers later. [Learn more.](#)

Select a provider \*

OpenID Connect (OIDC)

Provider details

Provider name \*

github

Provider ID: github

EDIT

Issuer (URL) \*

https://token.actions.githubusercontent.com

Issuer URL must start with https://

JWK file (JSON)

BROWSE

Optional, only needed if issuer is not publicly accessible. The JWK file should comply with [JWK specification](#). The max size of acceptable file is 80kB.

Audiences

Acceptable values for the aud field in the OIDC token.

Default audience

If the allowed audiences list is empty, the OIDC token audience must be equal to the full canonical resource name of the WorkloadIdentityPoolProvider, with or without the HTTPS prefix.

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

## 2. Add the attribute mappings

←

New workload provider and pool

3

Configure provider attributes

Credentials can include attributes that provide information about an identity. You can use attribute mapping to grant access to a subset of identities. [Learn more.](#)

Google 1

google.subject

?

Supported keys:

"google.subject",  
"google.groups",  
"attribute.custom\_attribute".

OIDC 1 \*

assertion.sub

?

Value must be a [CEL expression](#),  
for example, "assertion.sub".

Google 2 \*

attribute.repository\_id

?

Supported keys:

"google.subject",  
"google.groups",  
"attribute.custom\_attribute".

OIDC 2 \*

assertion.repository\_id

?

Value must be a [CEL expression](#),  
for example, "assertion.sub".

Google 3 \*

attribute.repository\_owner

?

Supported keys:

"google.subject",  
"google.groups",  
"attribute.custom\_attribute".

OIDC 3 \*

assertion.repository\_owner\_id



?

Value must be a [CEL expression](#),  
for example, "assertion.sub".

+ ADD MAPPING

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

### 3. Add the Attribute Conditions

**Attribute conditions**  
Restrict authentication to a subset of identities. By default, all identities belonging to providers in this pool can authenticate. [Learn more.](#)   
  
Condition CEL   
  
Use a [CEL expression](#),  for example, " 'admins' in google.groups".

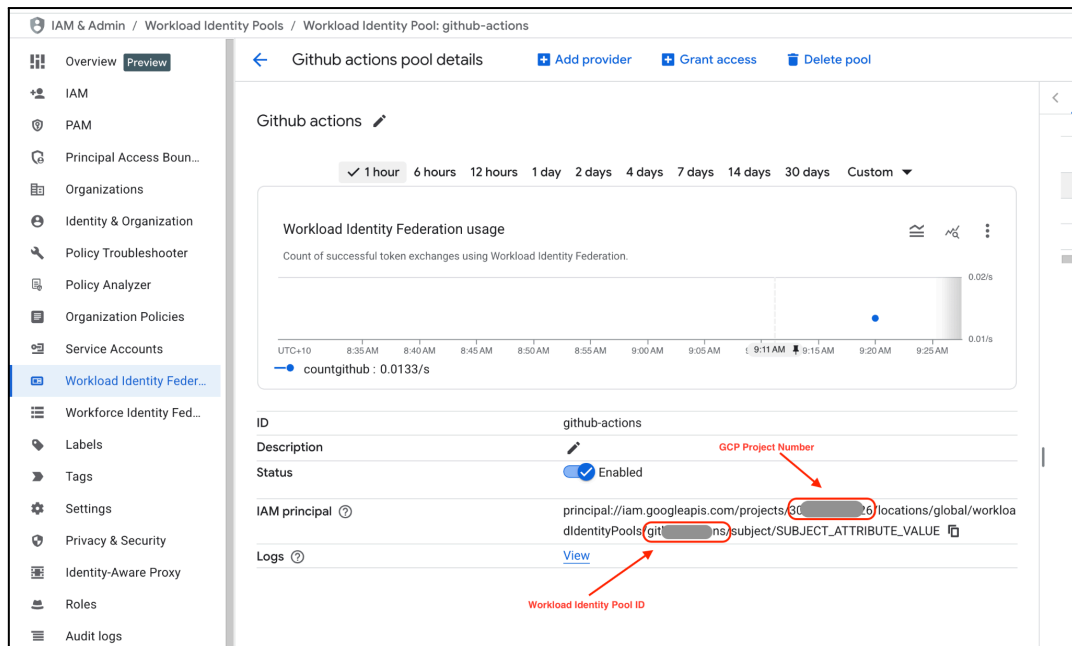
```
assertion.repository_id == "[REPO-ID]"
```

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`.

4. Locate your **Workload Pool ID** and **GCP Project Number**, in the GCP console goto Workload Identity Federation -> [Your Pool]. This will be needed in the next step.





## 5. Provide IAM access to the principalSet in your project

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

Grant access to "chronicle"

Grant principals access to this resource and add roles to specify what actions the principals can take. Optionally, add conditions to grant access to principals only when a specific criteria is met. [Learn more about IAM conditions](#)

**Resource**

chronicle

**Add principals**

Principals are users, groups, domains, or service accounts. [Learn more about principals in IAM](#)

**New principals \***

principalSet://iam.googleapis.com/projects/1[redacted]5/locations/global/workloadIdentityPools/github-actions-google-secp-1/attribute.repository\_id/7[redacted]5

**Assign roles**

Roles are composed of sets of permissions and determine what the principal can do with this resource. [Learn more](#)

**Role \***

Chronicle API Editor

Modify Access to Chronicle API resources.

**IAM condition (optional) ?**

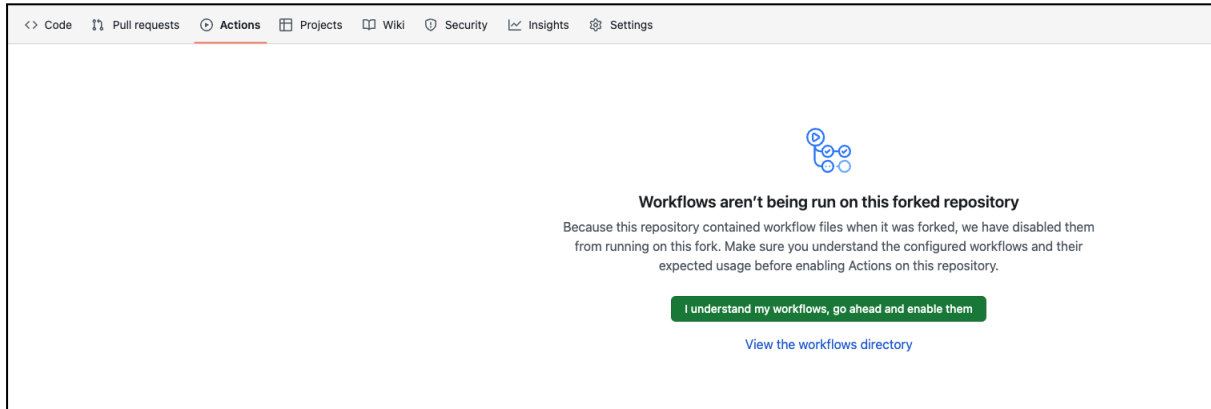
+ ADD IAM CONDITION

+ ADD ANOTHER ROLE

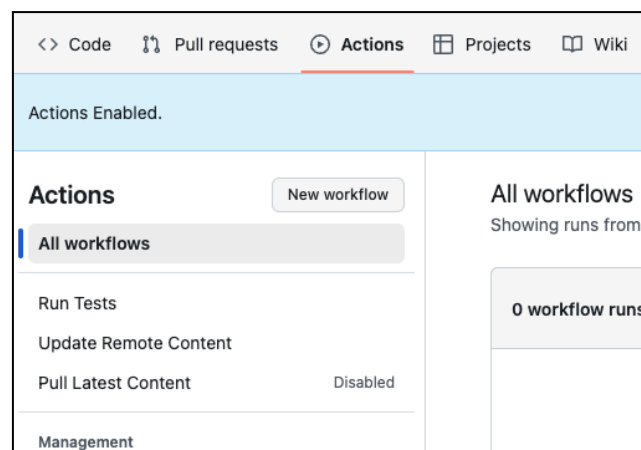
SAVE CANCEL

## Configure GitHub Actions & Variables/Secrets

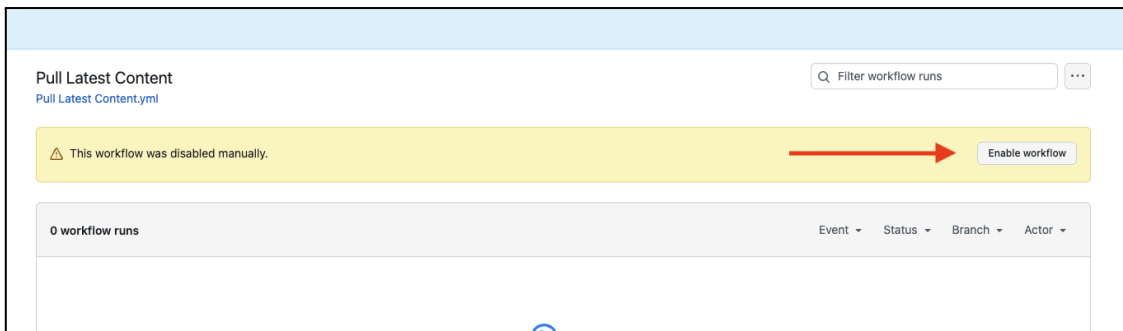
1. Go into Github and choose **Actions**, and choose “I understand my workflows, go ahead and enable them”



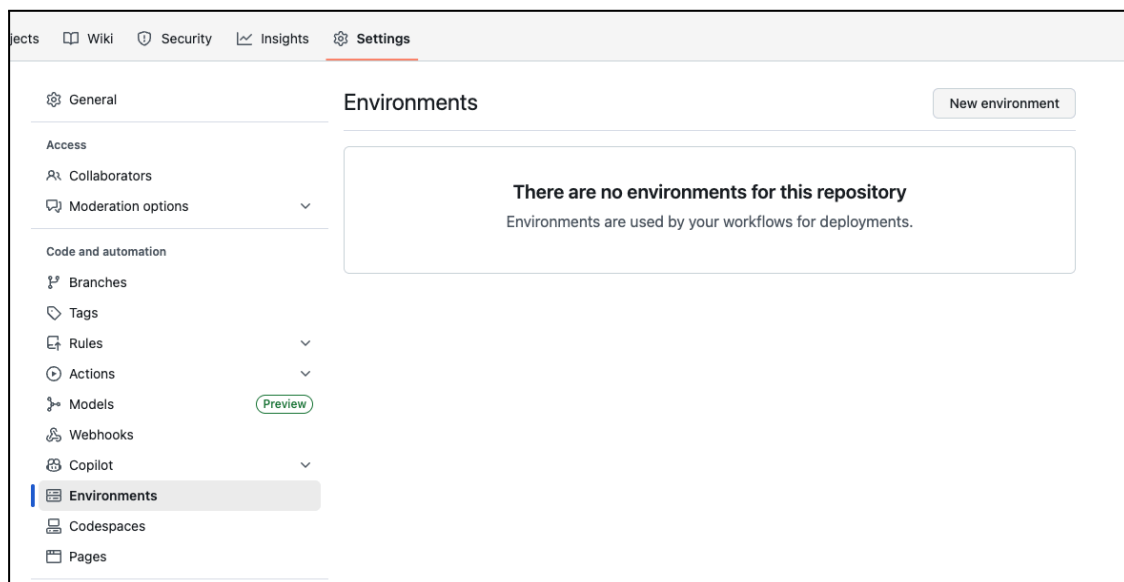
2. Confirm that the 3 Github actions are listed already as below



- 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. Go to Github 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"**

Limit which branches and tags can deploy to this environment based on rules or naming patterns.

**Environment secrets**

Secrets are encrypted environment variables. They are accessible only by GitHub Actions in the context of this environment by using the [secret context](#).

This environment has no secrets.

[Add environment secret](#)

**Environment variables**

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](#).

This environment has no variables.

[Add environment variable](#)

6. Variables to create (with example Values. 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. <i>myproject-373104</i>
GOOGLE_SECOPS_API_BASE_URL	https://<regionid>-chronicle.googleapis.com/v1alpha  e.g.: <a href="https://us-chronicle.googleapis.com/v1alpha">https://us-chronicle.googleapis.com/v1alpha</a>  <i>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.</i>
GOOGLE_SECOPS_API_UPLOAD_BASE_URL	https://<regionid>-chronicle.googleapis.com/upload/v1alpha  e.g.: <a href="https://us-chronicle.googleapis.com/upload/v1alpha">https://us-chronicle.googleapis.com/upload/v1alpha</a>  <i>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.</i>
GOOGLE_SECOPS_INSTANCE	projects/<GCP Project ID>/locations/<regionID>/instances/<secops-customer-id>  e.g.: <i>projects/myproject-373104/locations/us/instances/bc12345-8692</i>

	<p><b>-4184-a40d-6c12ff58a350</b></p> <p>The following are the supported region IDs:</p> <ul style="list-style-type: none"> <li>- us</li> <li>- eu</li> <li>- asia-southeast1</li> <li>- australia-southeast1</li> <li>- europe-west2</li> <li>- europe-west3</li> <li>- europe-west6</li> <li>- govcloud-us</li> <li>- me-west1</li> </ul>
LOGGING_LEVEL	INFO

7. Create the following secret



GOOGLE_CLOUD_WORKLOAD_IDENTITY_PROVIDER	<p>projects/&lt;GCP Project number&gt;/locations/global/workloadIdentityPools/&lt;PoolID&gt;/providers/&lt;ProviderID&gt;</p> <p>e.g. projects/123456789012/locations/global/workloadIdentityPools/mypoolid/providers/myproviderid</p>
---	--

Note the Pool ID and Provider ID e.g.

Display name ?	ID	P
▶ [blurred]	[blurred]	1
▼ <a href="#">dominic_test</a>	dominic-test	1
github_test	github-test	0













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  

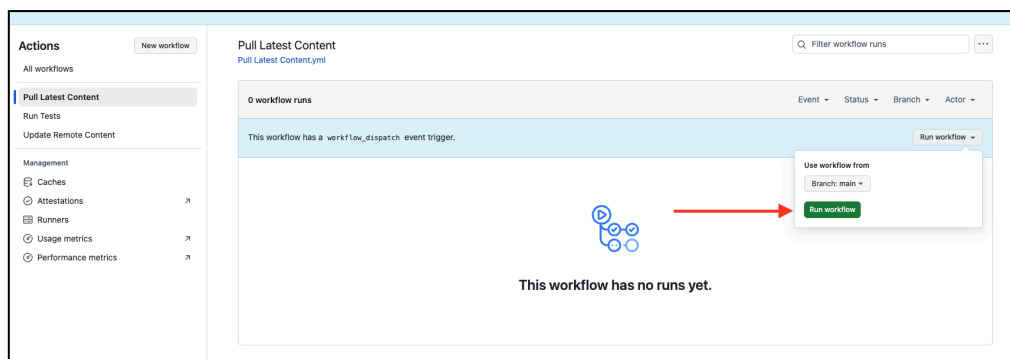
**Environment variables** Add environment variable

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](#).

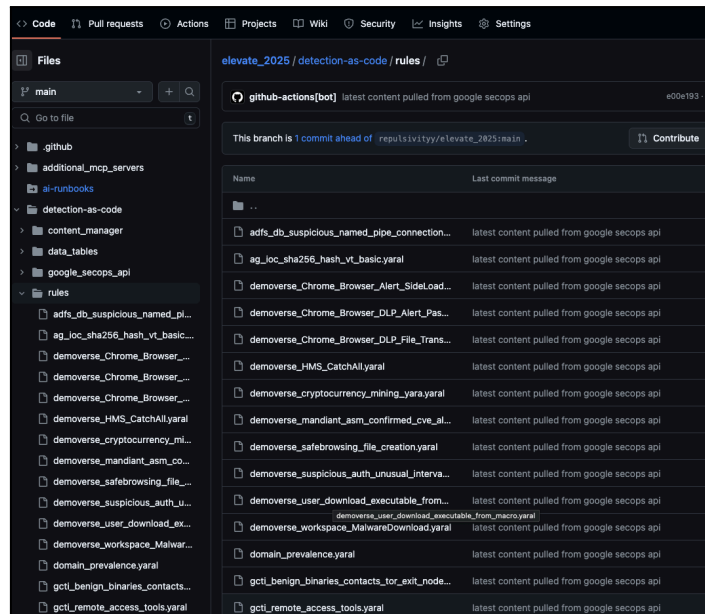
Name ↕↑	Value	Last updated
🔒 AUTHORIZATION_SCOPES	{"GOOGLE_SECOPS_API":["https://www.g...	last week  
🔒 GOOGLE_CLOUD_PROJECT_ID	malachite-ace-373104	last week  
🔒 GOOGLE_SECOPS_API_BASE_URL	https://australia-southeast1-chronicle.goo...	last week  
🔒 GOOGLE_SECOPS_API_UPLOAD_BASE...	https://australia-southeast1-chronicle.goo...	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

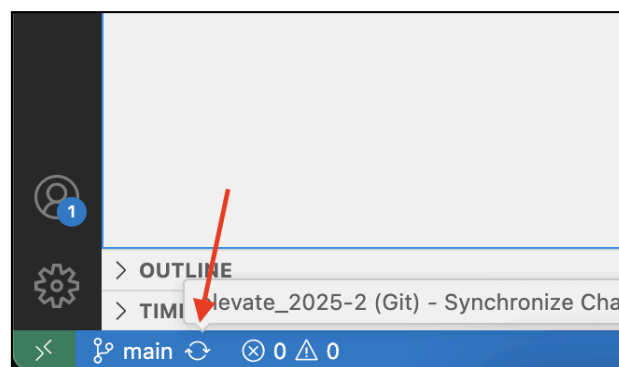
1. Lets now kick off the “**Pull Latest Content**” action to pull down rules / data tables / reference lists. Go into Actions, Choose “**Run workflow**”.



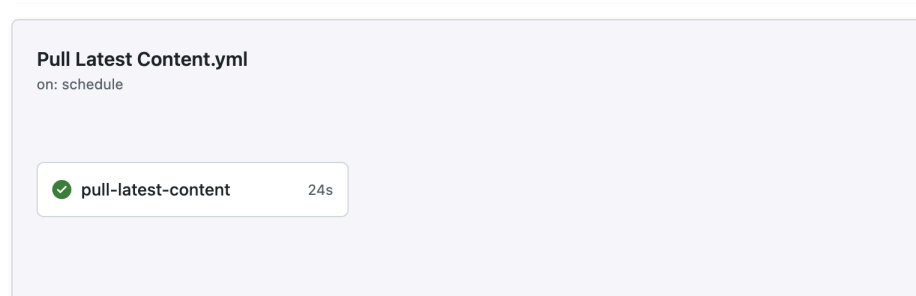
2. On completion of the above action, in your Github repo and in your local repo in VS Code (once it syncs), you should see all your rules



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



4. If successful, you should see the rules within the detection-as-code -> rules folder. Also, your pull latest 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!

```

content_manager.common.custom_exceptions.DuplicateRuleNameError: Duplicate rule names found [('suspicious_unusual_location_lnk_file', 2), ('whois_expired_domain_executable_downloaded', 2), ('vt_relationships_file_executes_file', 2), ('vt_relationships_file_downloaded_from_url', 2), ('vt_relationships_file_downloaded_from_ip', 2), ('vt_relationships_file_contacts_ip', 2), ('vt_relationships_file_contacts_domain', 3), ('fusion_target_ip', 2), ('my_first_snapattack', 4)].
Error: Process completed with exit code 1.
##[debug]Finishing: Pull latest version of all rules from Google SecOps

```

**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).

```

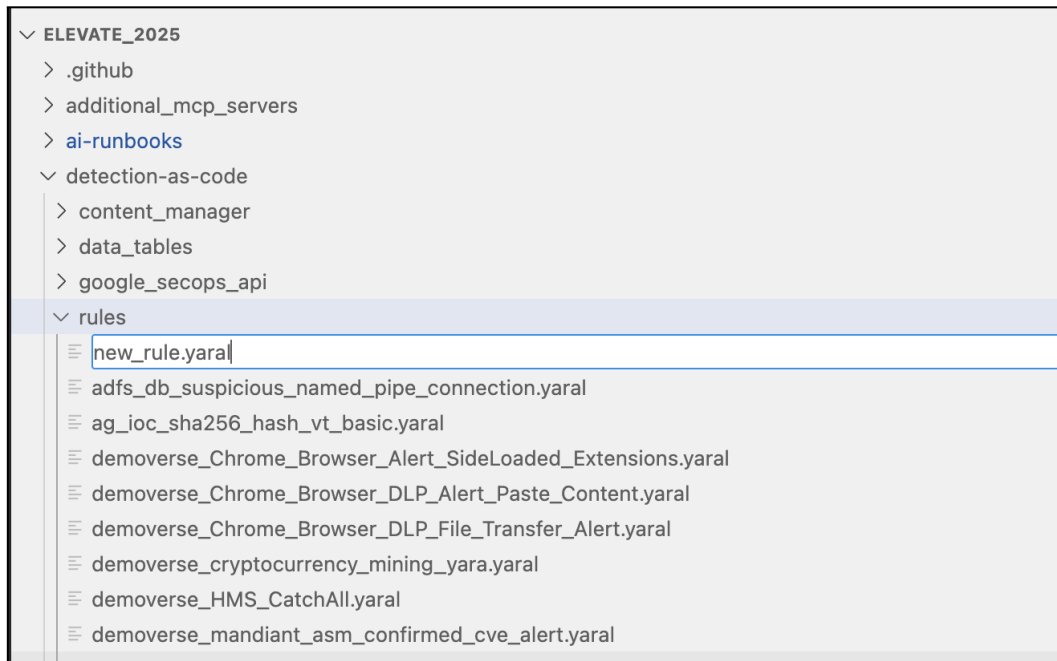
33 ##[debug]/usr/bin/bash -e /home/runner/work/_temp/d52a7ea0-d43b-4692-a4d2-f9c0e5fcb0fe.sh
34 08-Jun-25 12:52:28 UTC | INFO | <module> | Content Manager started
35 08-Jun-25 12:52:28 UTC | INFO | get_rules | Attempting to pull latest version of all rules from Google
36 08-Jun-25 12:52:28 UTC | INFO | get_remote_rules | Attempting to retrieve all rules from Google SecOps
37 08-Jun-25 12:52:29 UTC | WARNING | list_rules | {
38   "error": {
39     "code": 403,
40     "message": "Permission 'chronicle.rules.list' denied on resource '//chronicle.googleapis.com/projects/
exist).",
41     "status": "PERMISSION_DENIED",
42     "details": [
43       {
44         "@type": "type.googleapis.com/google.rpc.ErrorInfo",
45         "reason": "IAM_PERMISSION_DENIED",
46         "domain": "chronicle.googleapis.com",
47         "metadata": {
48           "permission": "chronicle.rules.list",
49           "resource": "projects/
utheast1/instances/8a9bccdd8-288f-488c
50       }
51     ]
52   }
53 }

```

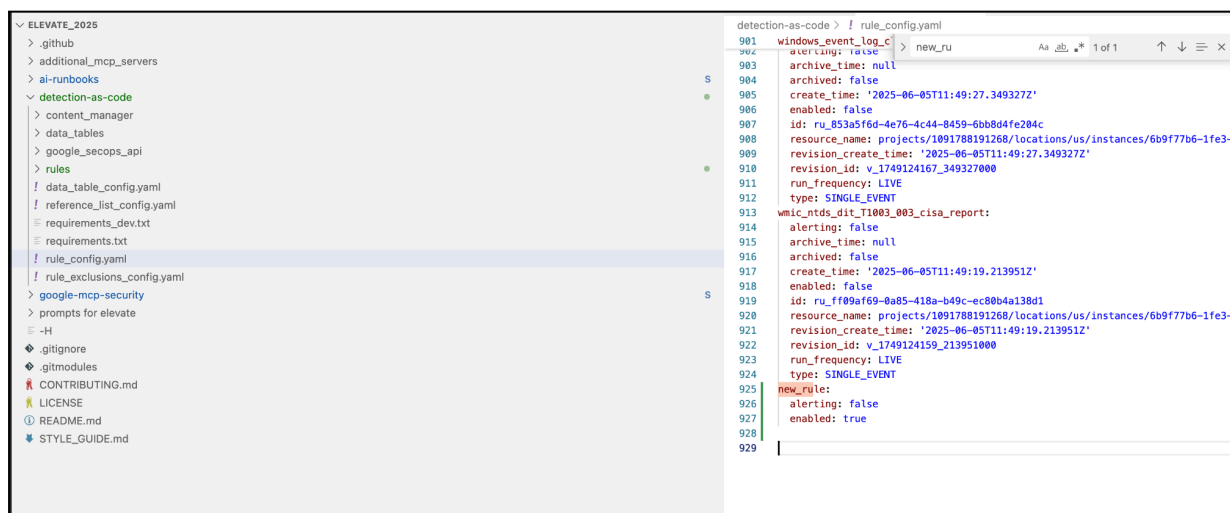


## Testing the Github Actions - 2. Creating a new rule

- 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.



- Hit save on the file, after adding your new rule.
- Now add an entry to `detection-as-code -> rule_config.yaml` file and save.

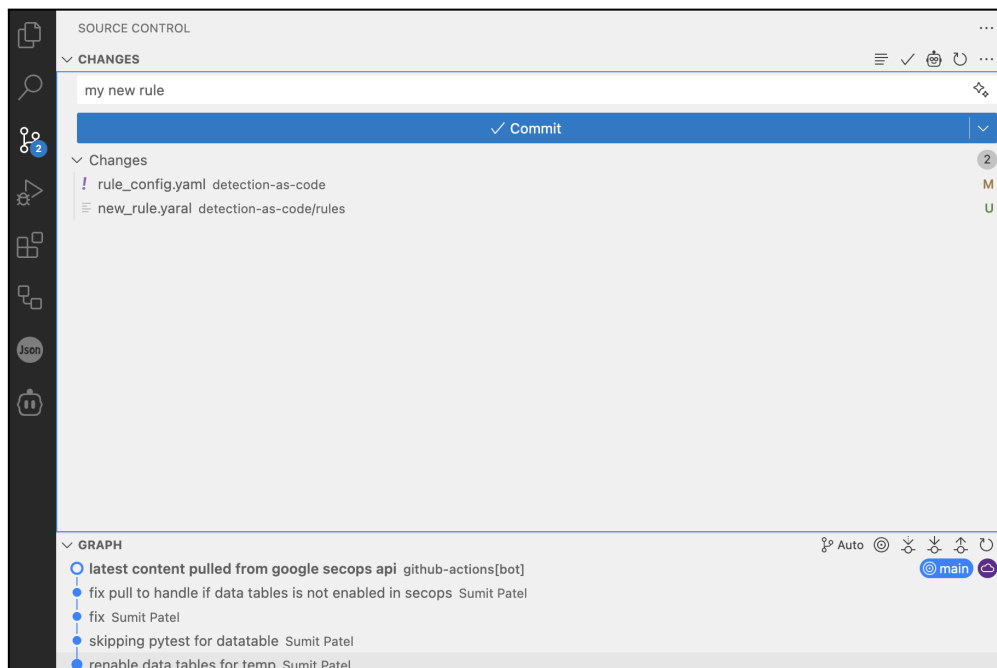


example:

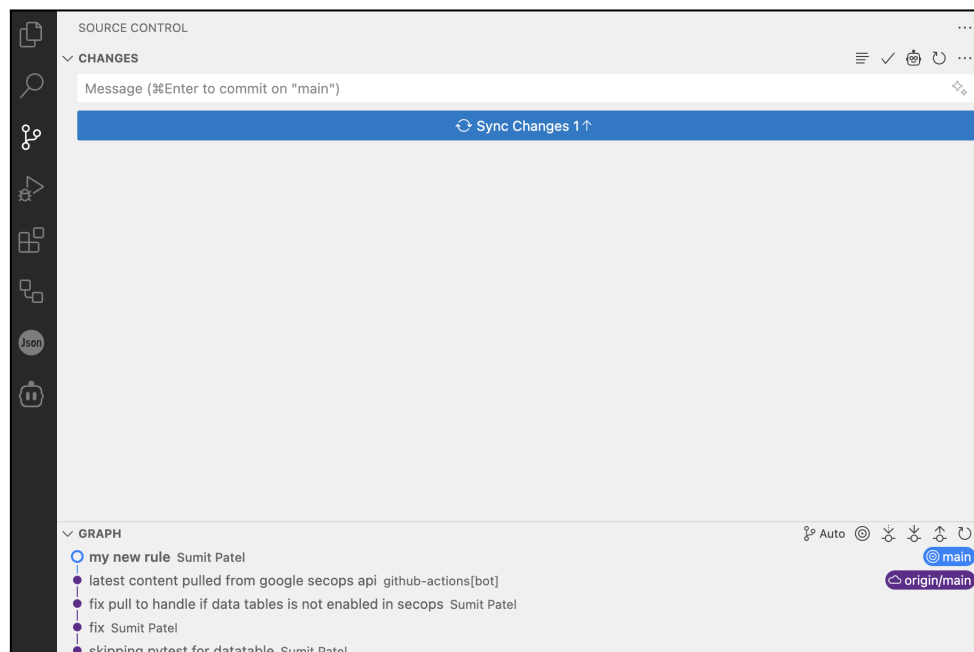
```
new_rule:  
alerting: false
```

enabled: **true**

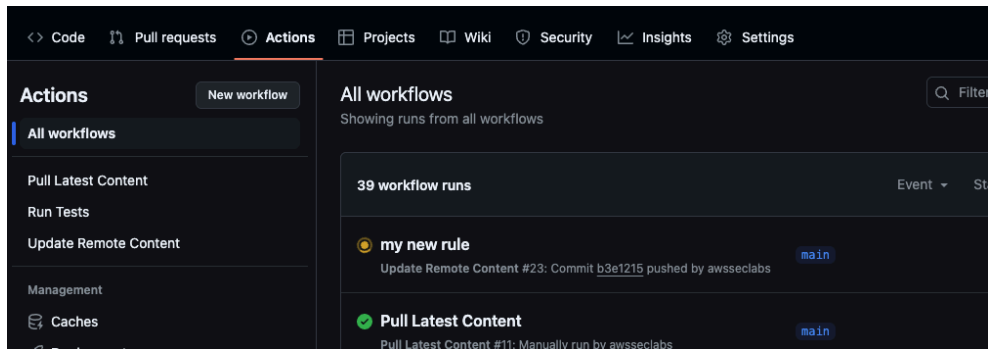
- Hit the “Source Control” menu item on the left, and add a comment e.g. “my new rule” and hit **Commit**.



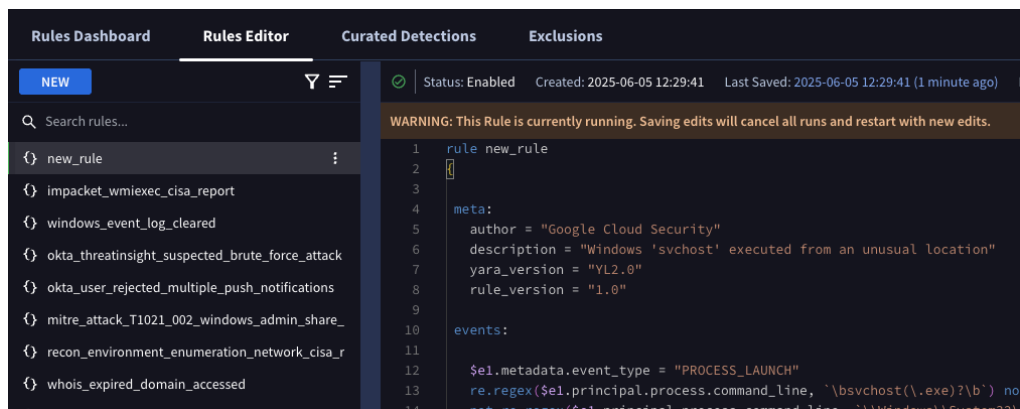
- Let's now **“Sync changes”**.



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



11. 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.*

## Best Practices working with Git

### Commit early and often (and Small!)

- **Atomic Commits:** 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*).