Microsoft Reactor Lab Guide

By Modus Create

moduscreate.com

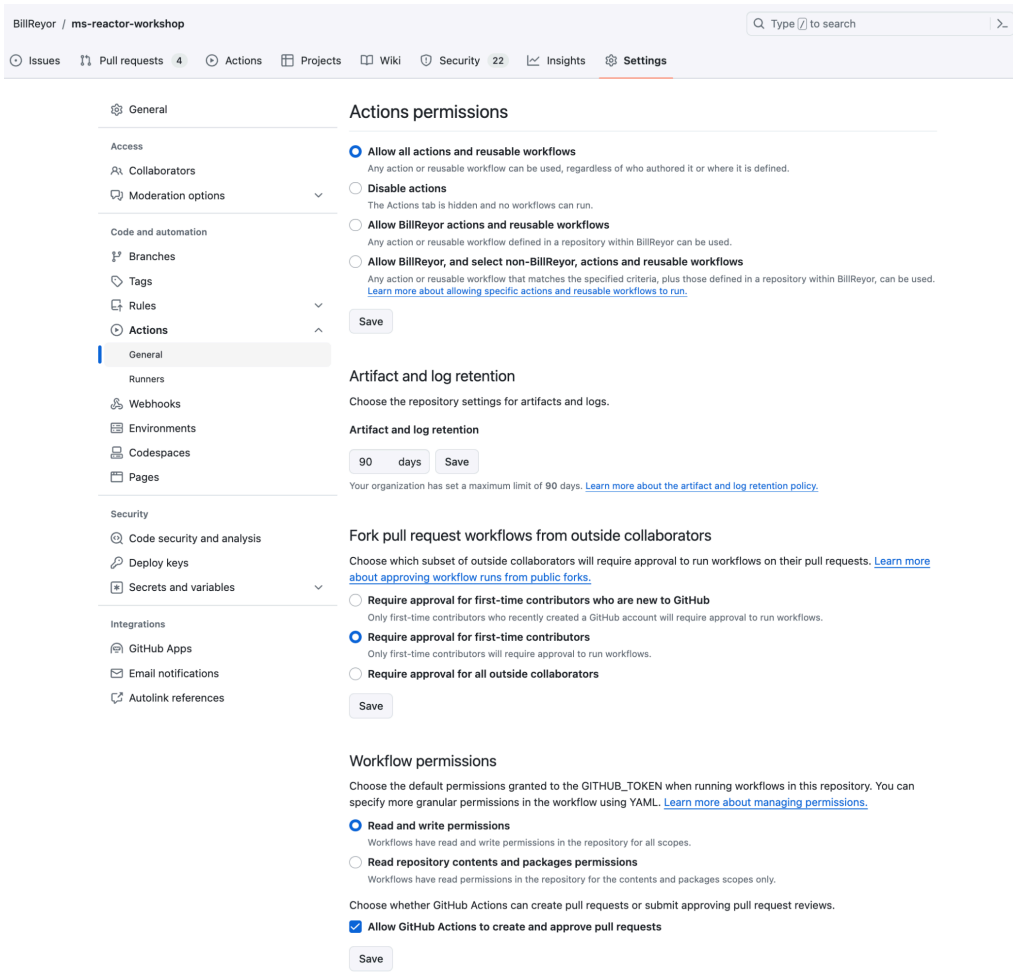
Microsoft Reactor

About this lab

This Microsoft Reactor Lab Guide, designed by Modus Create, is designed to be presented by an instructor who explains the concepts and demonstrates each step live. After each demonstration, participants are encouraged to replicate the actions, using the provided screenshots and detailed instructions as a guide. This method ensures that attendees not only understand the theoretical aspects of the tools and practices being taught but also gain practical experience by applying what they've learned in real time.

Setup

1. Fork the provided workshop repository from https://github.com/tweag/ms-reactor-workshop
2. Check Workflow Permissions (See Figure 0)
    a. In your repository, go to "Settings."
    b. Click "Actions" > “General” in the left sidebar.
    c. Under “Actions Permissions” Ensure that "Allow all actions and reusable workflows" is enabled.
    d. Under "Workflow permissions," make sure "Read and write permissions" are selected.
3. If you receive the message “GitHub Actions is disabled on this repository because it is a fork. To use code scanning, please enable it”,
    a. Click “enable it” > I understand my workflows, go ahead and enable them.



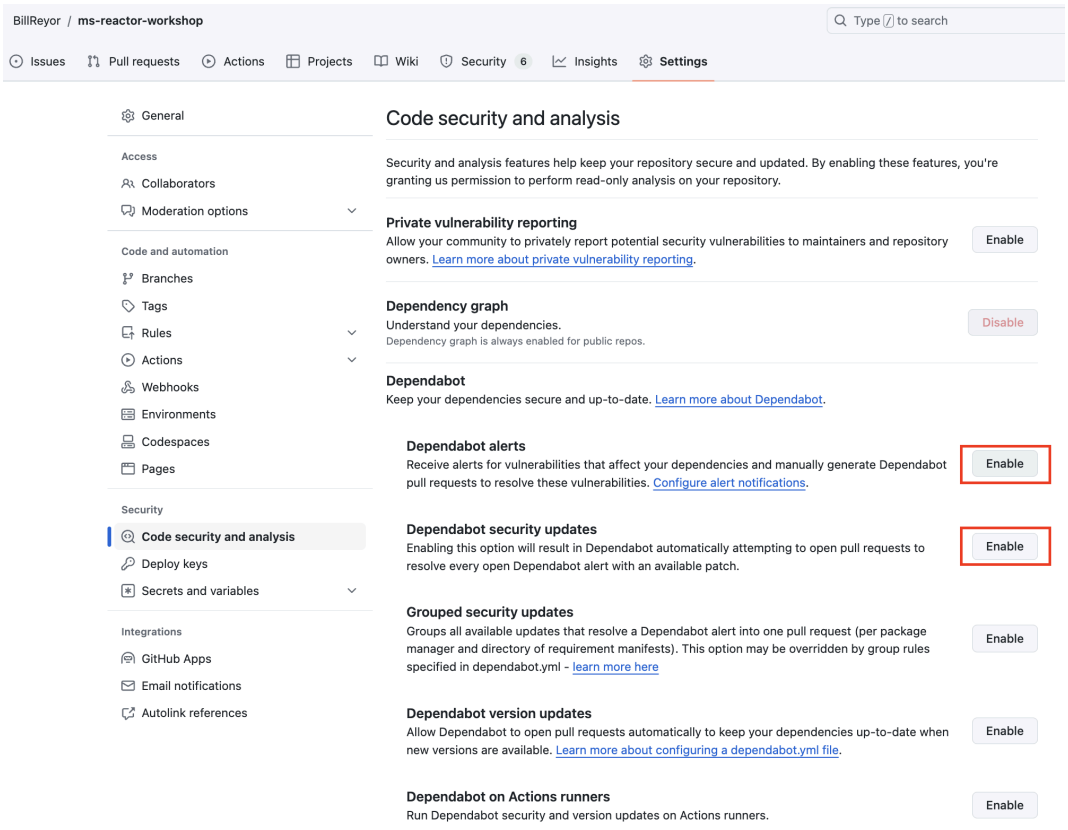
*(*Figure 0 - *Workflow configuration)*

Module 1 - Dependabot

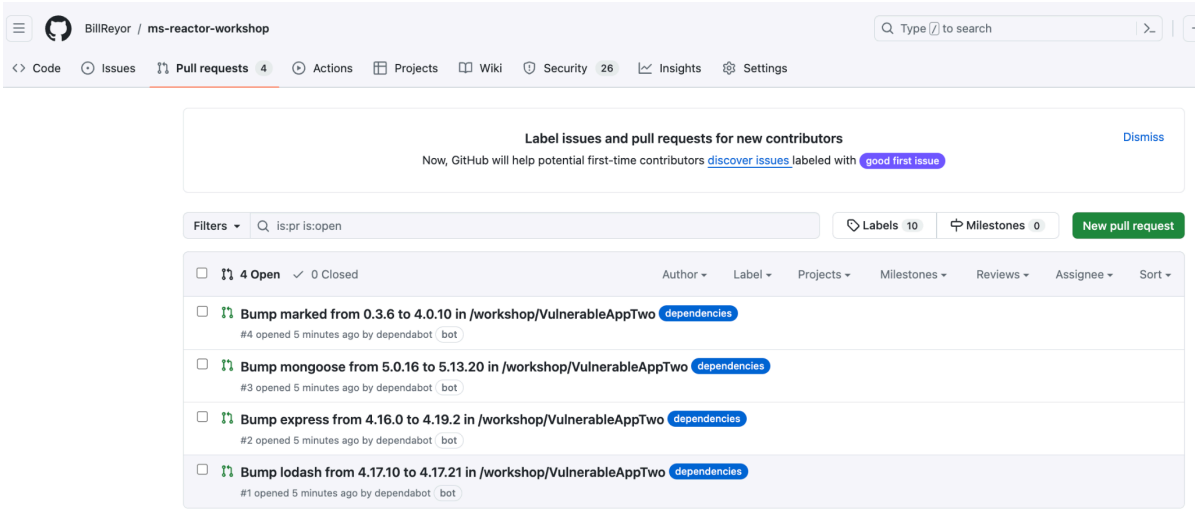
1. Enabling Dependabot Alerts
    - Go to the forked repository's "Settings" page.
    - Click on "Code Security & Analysis."
    - Locate the "Dependabot alerts" option and click "Enable."
    - This will notify you of vulnerabilities in your dependencies.
2. Enable Dependabot Security Updates (See Figure 1)
    - Stay on the "Code Security & Analysis" page.
    - Find the "Dependabot security updates" option and click "Enable."
    - Dependabot will attempt to automatically create pull requests to fix detected vulnerabilities with available patches.
3. Navigate to the Pull Requests tab and evaluate the results. *(*See Figure 2)
    - Open Any finding
    - Navigate to the “Files Changed” tab *(*See Figure 3)
    - Navigate back to the bottom of the conversation tab & Merge the Pull Request (PR) > Confirm Merge *(*See Figure 4)
4. Review Dependabot alerts
    - In the workshops forked repository, navigate to Security -> Dependabot (See Figure 5)

Summary

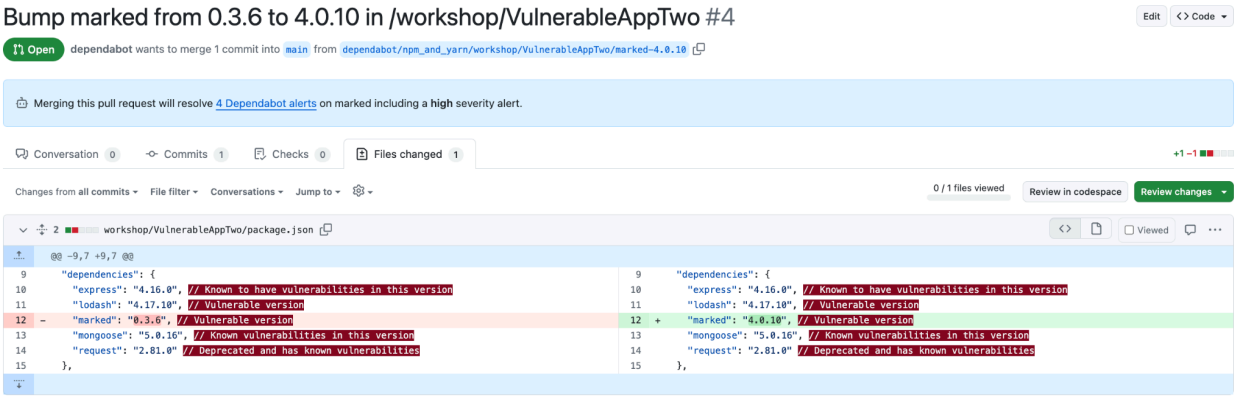
This module explored activating Dependabot alerts and updates in GitHub repositories. By enabling these features, your teams are promptly notified about vulnerabilities in project dependencies. This is crucial as it allows for quick identification and resolution of security issues, reducing the risk of exploits in software projects.



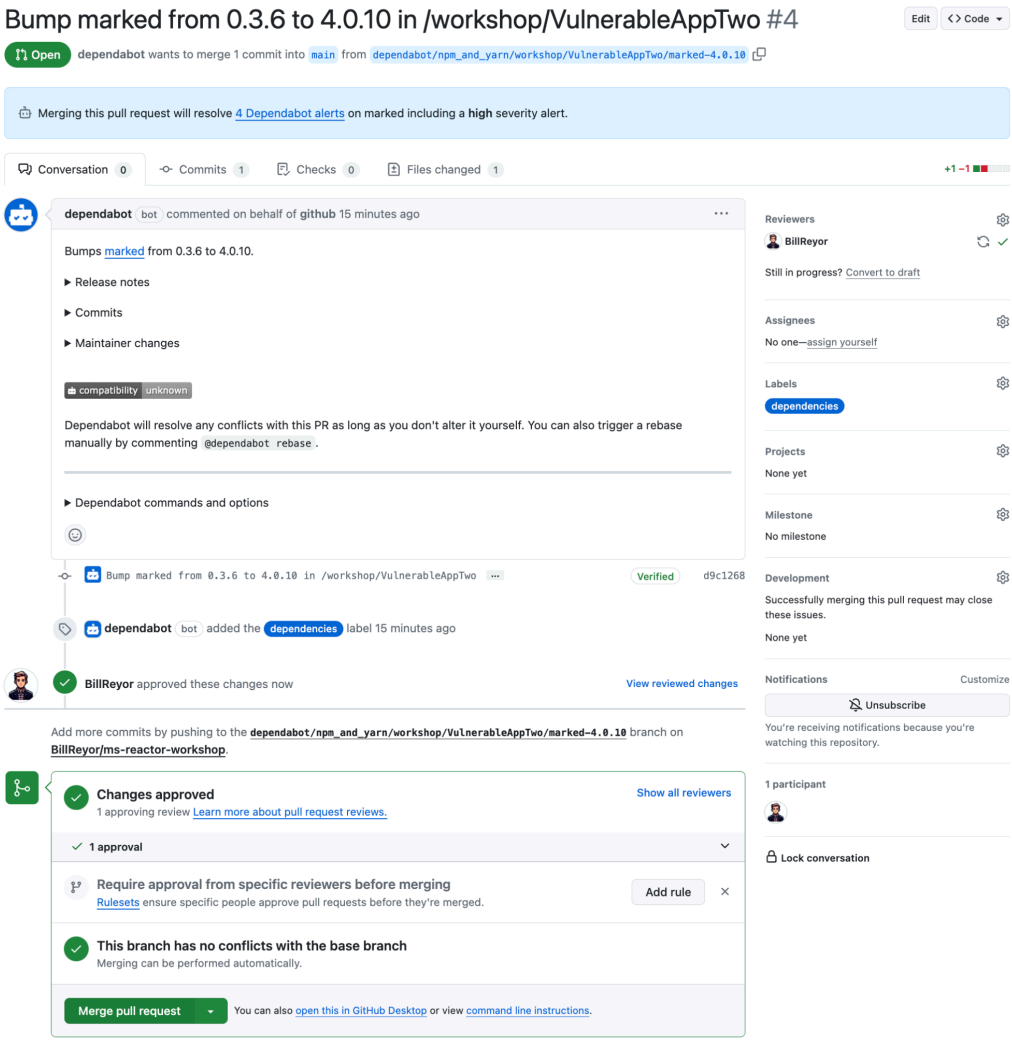
(Figure 1 - Enabling Depandabot)



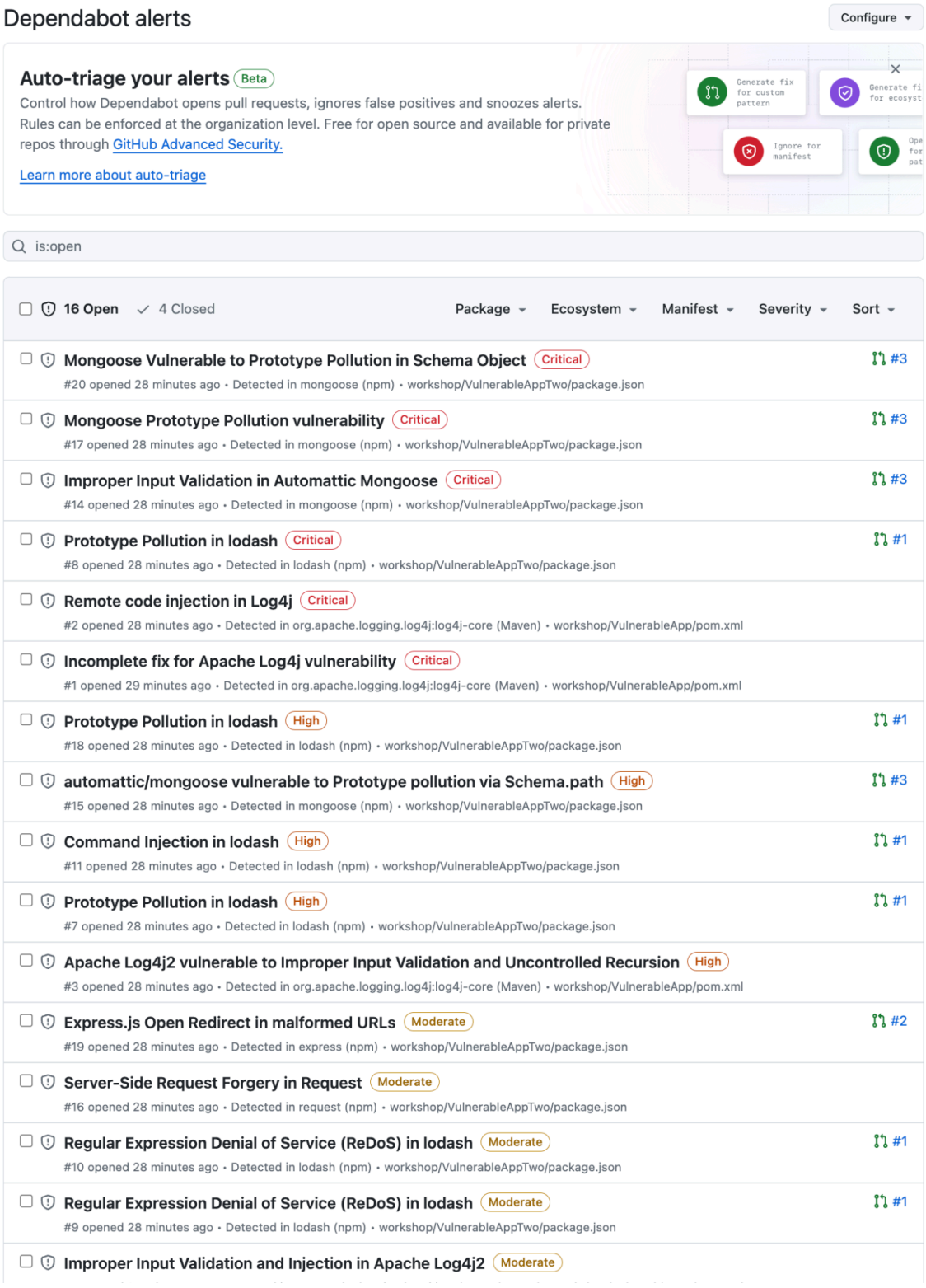
(Figure 2 - Dependabot PR results)



(*Figure 3 - Dependabot PR results*)



(*Figure 4 - Dependabot PR Merge*)



(Figure 5 - Dependabot Alerts)

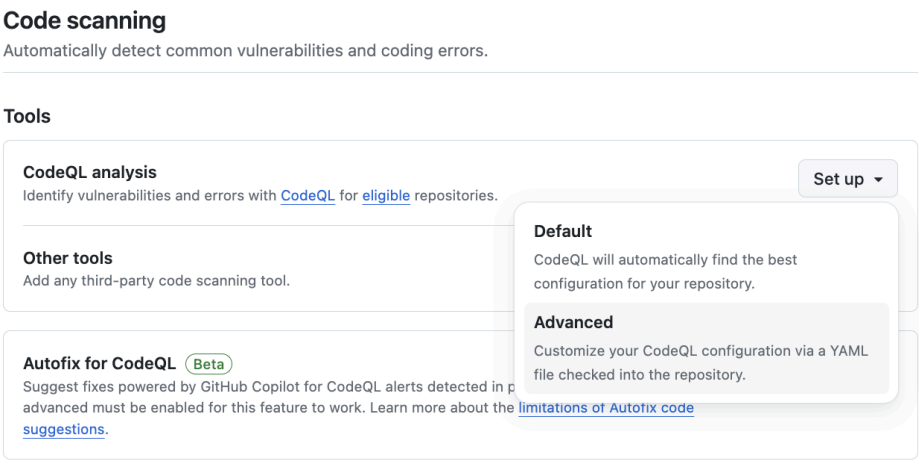
Module 2 - CodeQL

Setting Up CodeQL

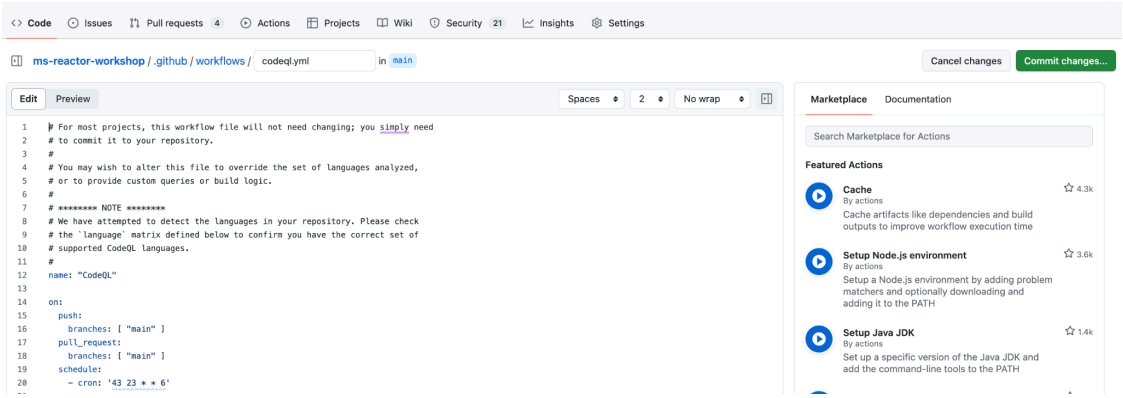
- Navigate back to your repository's Settings > "Code Security and Analysis"
- Under the Code Scanning section CodeQL Analysis > Set up > Advanced > (See Figure 6)
- From CodeQL YML > Commit Changes > Commit Changes (See Figures 7)
- You will now see under Actions the CodeQL scan workflow has been triggered, let's give it a minute to finish.
- After the analysis is completed, navigate to security -> Code Scanning, where you should see CodeQL results. (Figure 8)

Summary

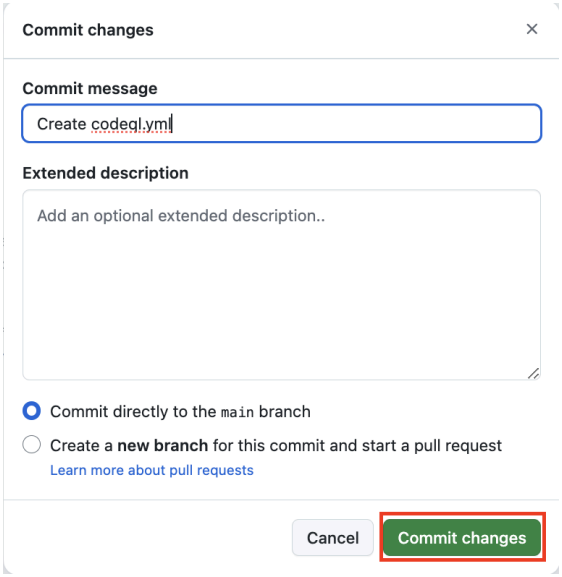
This module focused on setting up CodeQL for security analysis in GitHub repositories. By enabling GitHub Advanced Security and configuring CodeQL, your teams can automatically perform deep security checks on their codebases, helping to maintain the integrity and security of software projects.



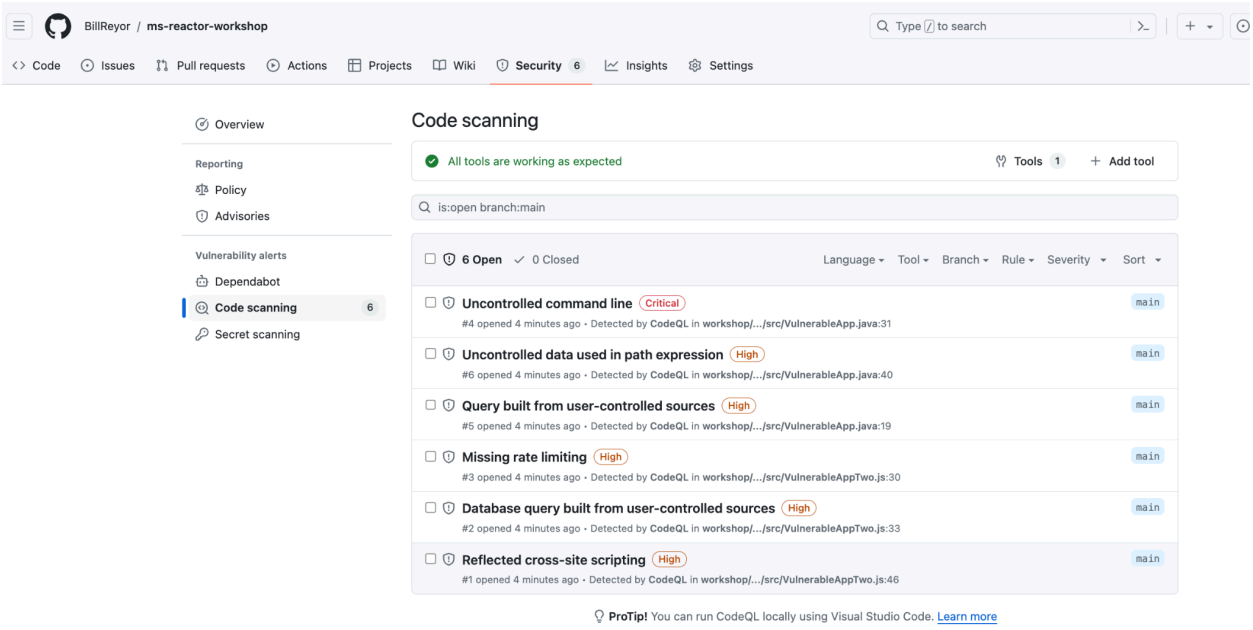
(Figure 6 - CodeQL Setup)



(Figure 7 - Commit the Default CodeQL workflow to main)



(Commit the Default CodeQL workflow to main)



(Figure 8 - Code Scanning Results)

Module 3 - AI-driven Secret Scanning

Secret scanning is enabled by default for all public repositories on GitHub. However, those holding an enterprise license have access to enhanced features. For most attendees, simply follow along with the demonstration and slides for this segment, as the steps require a license that may not be available to everyone.

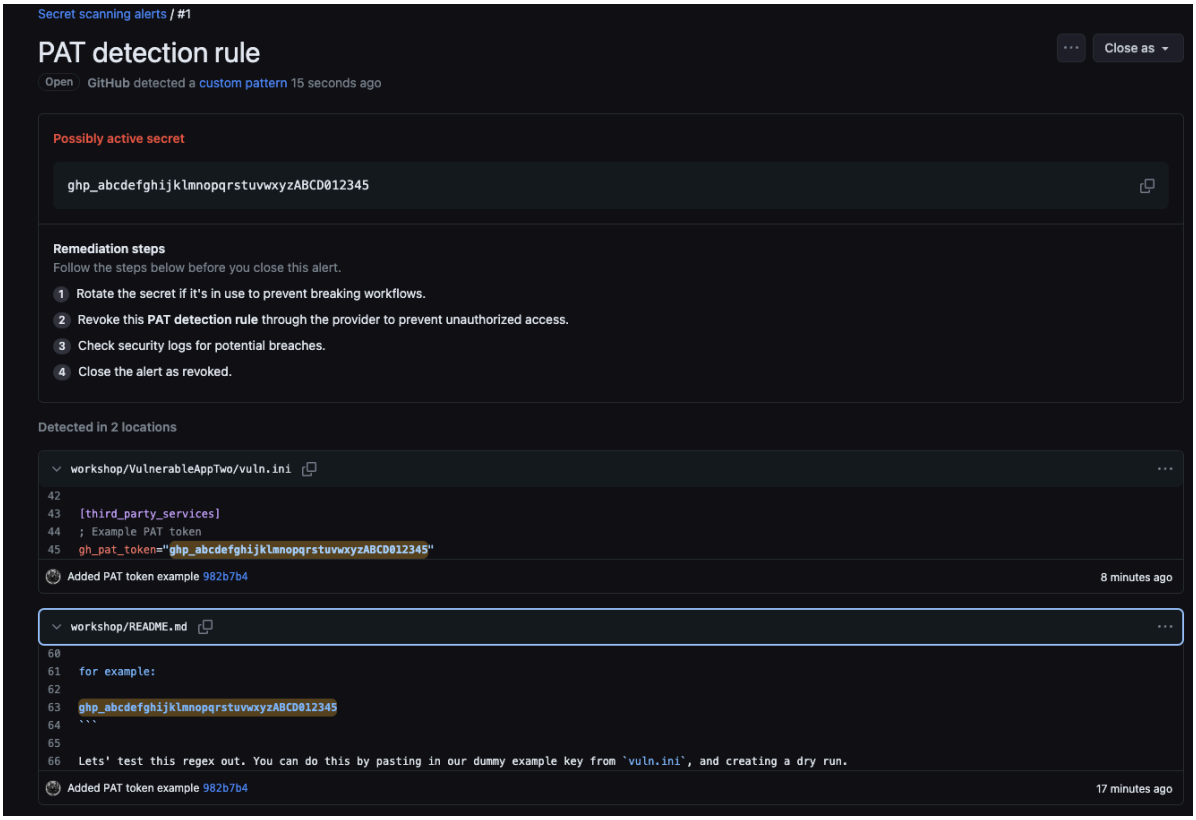
1. Navigate to Settings > Code Security and Analysis to configure secret scanning options.
2. Select the New Pattern button to begin creating a custom pattern.
3. Use the following regex to test for the presence of a GitHub Personal Access Token (PAT `(ghp|gho|ghu|ghs|ghr)_[a-zA-Z0-9]{36}`
4. Test this regex by pasting a dummy example key from vuln.ini and perform a dry run to ensure it works as expected.
5. After testing, click the Publish pattern button to save and activate your custom secrets detection pattern.
6. Let's create one more new pattern to highlight how AI can help us write this regex
    a. Go back to the secret scanning options
    b. Click - Generate with AI
    c. For a prompt enter "Find instances that match an SSN" and provide an **fake** ssn example like nnn-nn-nnnn
    d. Save and perform a dry run

Reviewing Detected Secrets

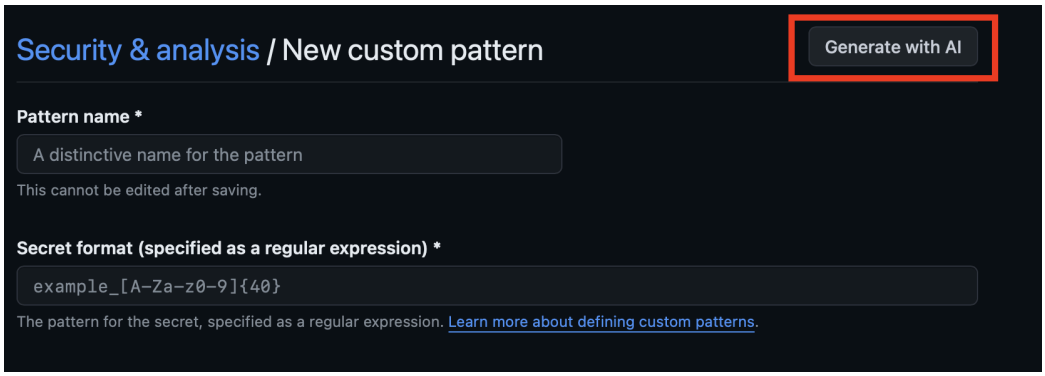
1. To view the secrets detected by your configurations, navigate to Security > Secret scanning on your GitHub repository settings. Here, you can review and manage any findings.

Summary

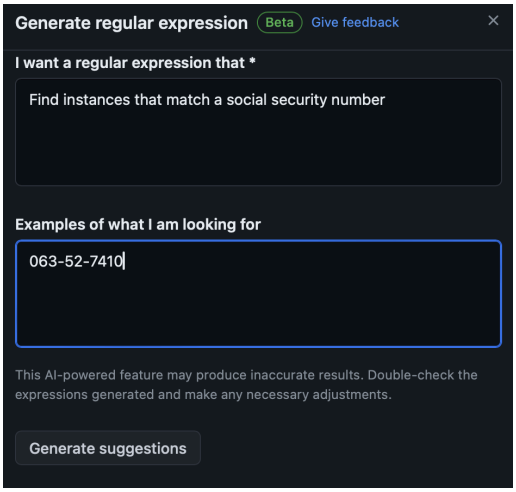
In this module, we focused on AI-driven secret scanning, which is automatically enabled for public repositories. Enterprise users enjoy enhanced capabilities. We explored how to configure custom secret scanning patterns, using regular expressions to detect GitHub Personal Access Tokens. This helps secure your code by identifying sensitive information inadvertently committed to your repositories.



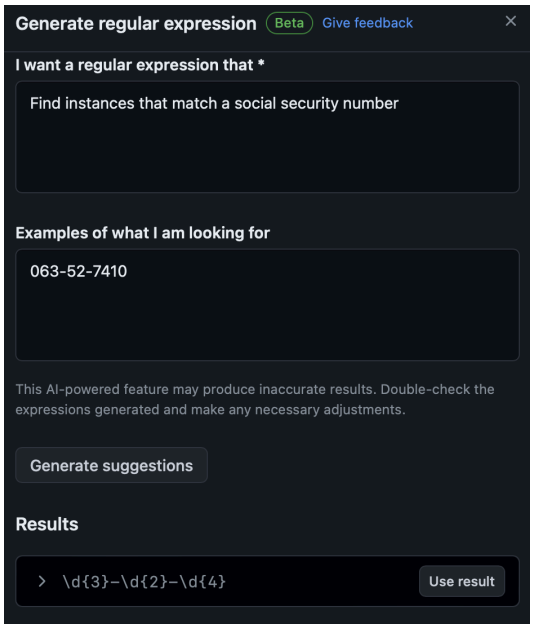
(Secret Scanning - Figure 9 - Pat Detection)



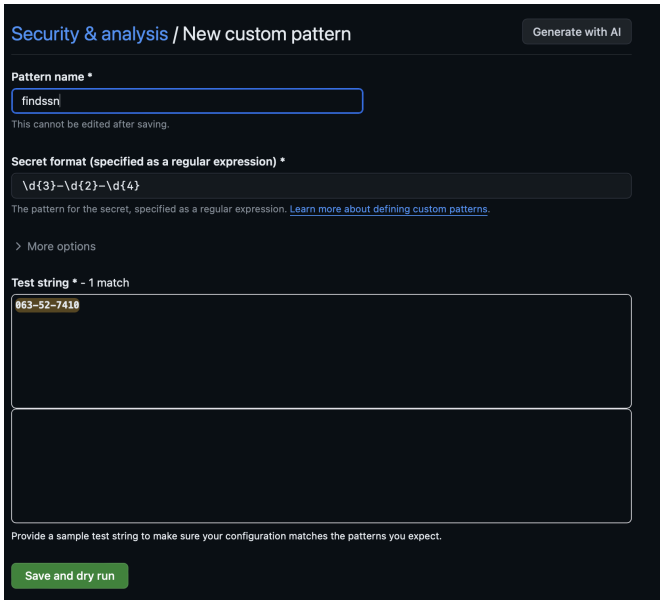
(Secret Scanning - With GenAI)



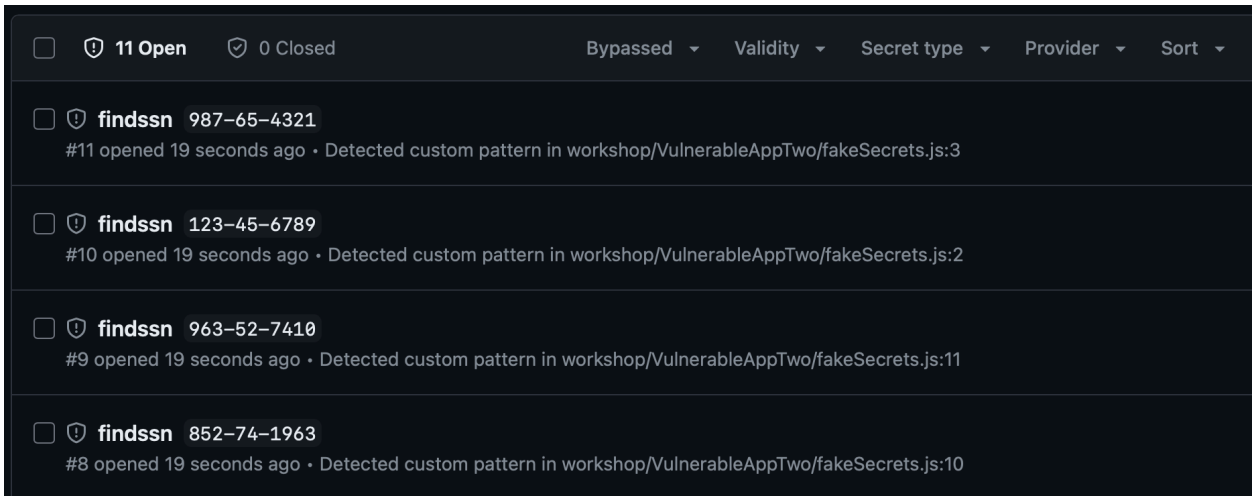
(Secret Scanning - With GenAI for an SSN)



(Secret Scanning - With GenAI created regex)



(Secret Scanning - With GenAI created regex save and dry run)

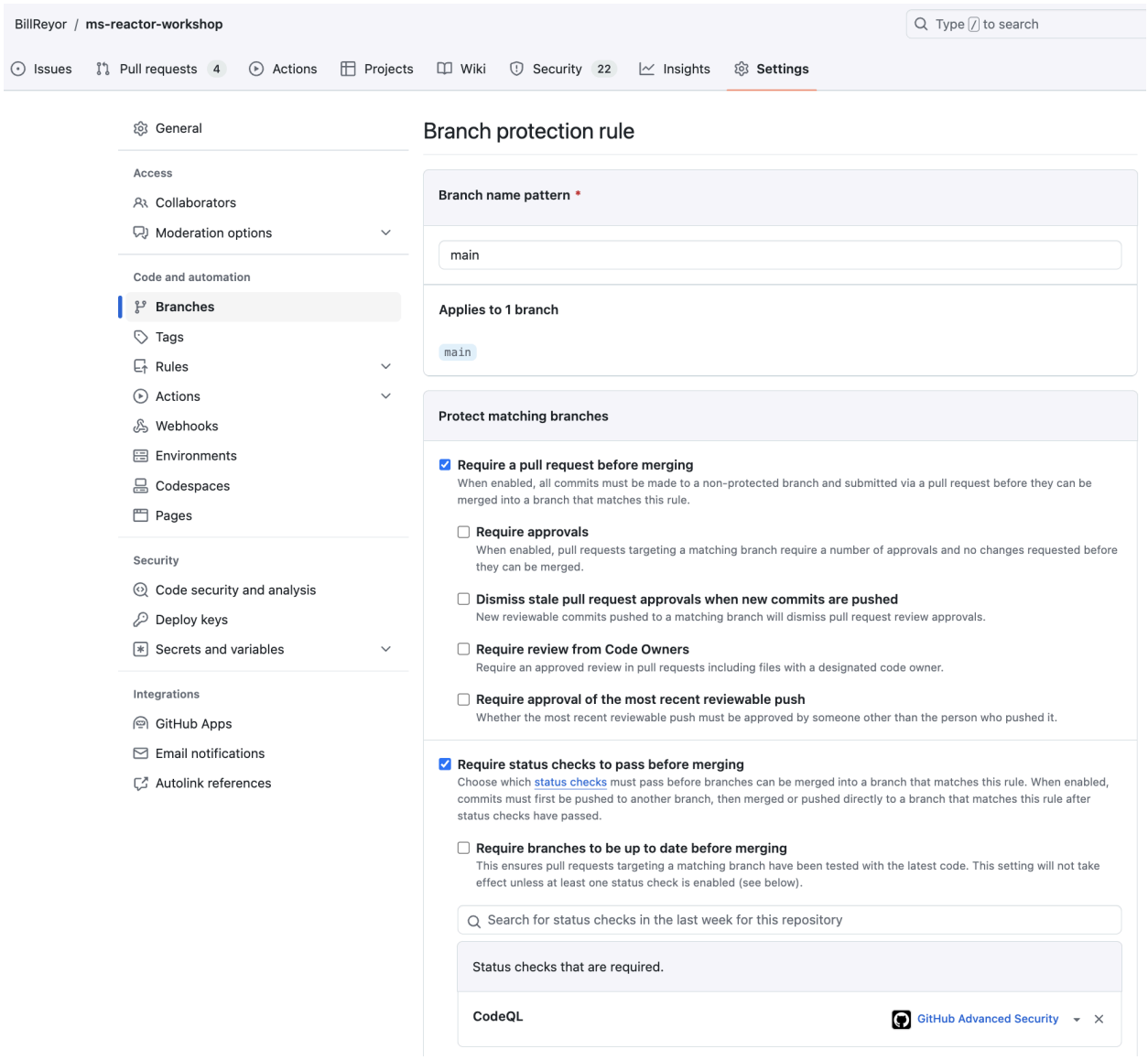


(Secret Scanning - With GenAI created regex results)

Module 4 - Setup Branch Protection Rules

Cool, we detected the code problems, but how do we prevent them in the first place?

1. Navigate to Branch Protection Settings:
    - From your forked repository, click the "Settings" tab.
    - In the left sidebar, select "Branches."
    - In the "Branch protection rules" section, click "Add Branch Protection Rule" to create a new rule or edit an existing one for the `main` branch.
2. Require Passing Status Checks
    - Enter `main` as the branch pattern to which the rule should apply.
    - Check “Require a pull request before merging”
    - Check "Require status checks to pass before merging."
    - Within the “Search for status checks in the last week for this repository” section, select the following checks:
        - Search for and select “CodeQL”
        - Now scroll to the bottom of the page and Save by clicking the "Create" or "Save changes" button.
        - See Figure 10 - Branch Protection Rules for illustration.
    - Navigate back to Settings > Branches > Edit our “main” branch protection rule set > Validate that “Require approvals” is unchecked (for our example only) > Save Changes.



(Figure 10 - Branch Protection Rule)

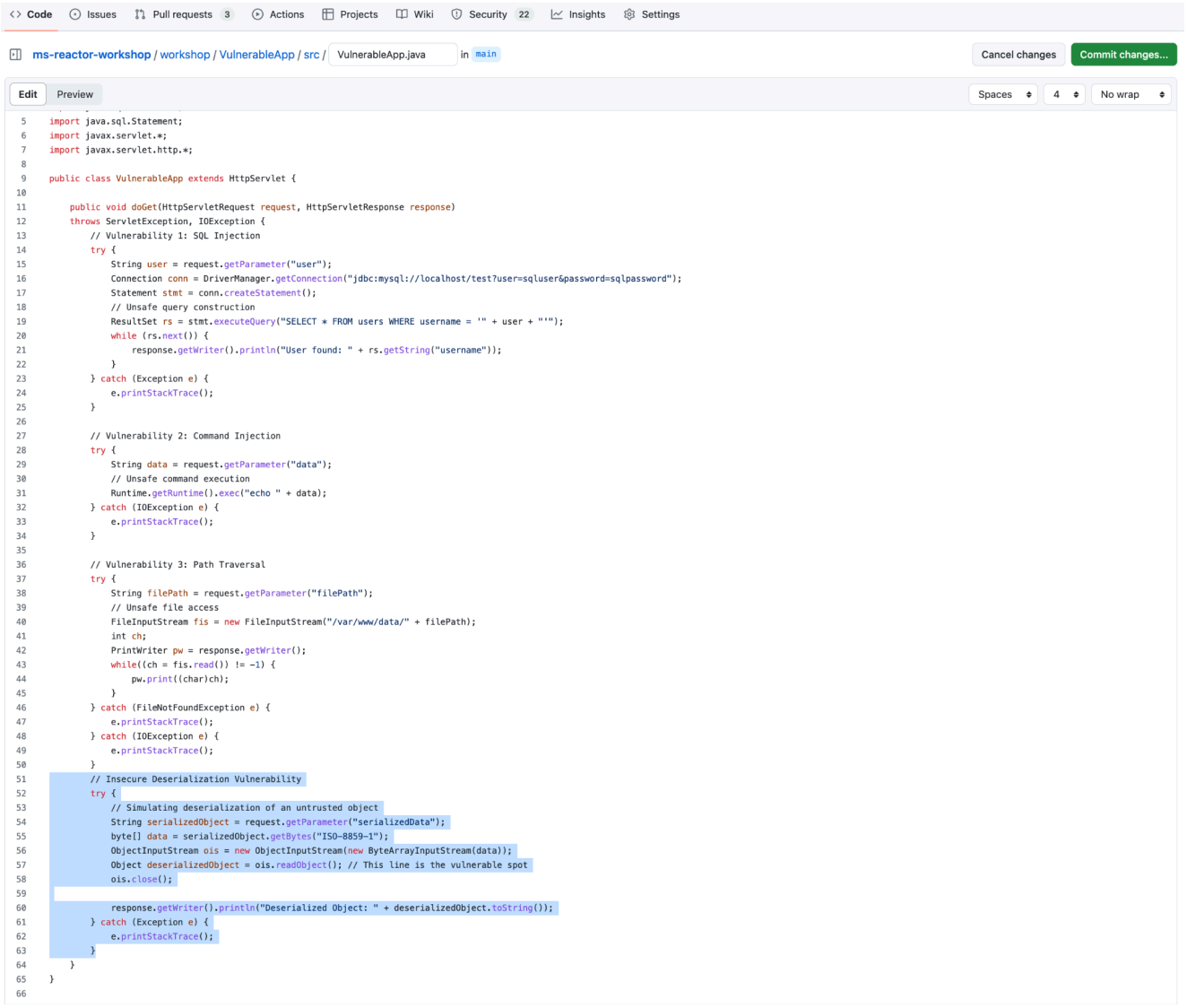
Module 5 - Branch Protection Demonstration

1. To Test the Branch Protection Rule
    - Navigate to the Code tab and edit the file directly via the editor at GitHub.com /workshop/VulnerableApp/src/VulnerableApp.java
    - We will now add an insecure deserialization vulnerability by appending the code below to the end of our project between lines 50 and 51.
    - Commit the change (*Figure 12)*
        - Select the option to create a new branch for this commit and start a pull request.
        - Click “Propose the change”
        - Wait for status checks to finish
    - If everything has gone to plan, our changes are blocked because they failed the CodeQL scan (See Figure 13)

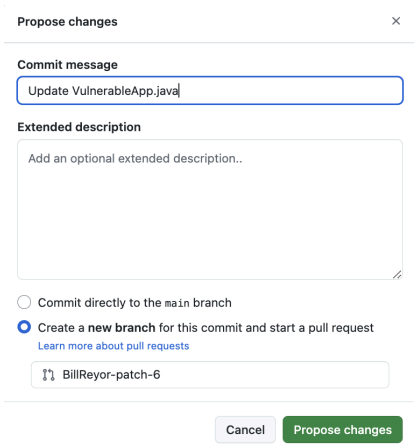
```
// Insecure Deserialization Vulnerability
    try {
        // Simulating deserialization of an untrusted object
        String serializedObject = request.getParameter("serializedData");
        byte[] data = serializedObject.getBytes("ISO-8859-1");
        ObjectInputStream ois = new ObjectInputStream(new ByteArrayInputStream(data));
        Object deserializedObject = ois.readObject(); // This line is the vulnerable spot
        ois.close();

        response.getWriter().println("Deserialized Object: " + deserializedObject.toString());
    } catch (Exception e) {
        e.printStackTrace();
    }
```

Summary

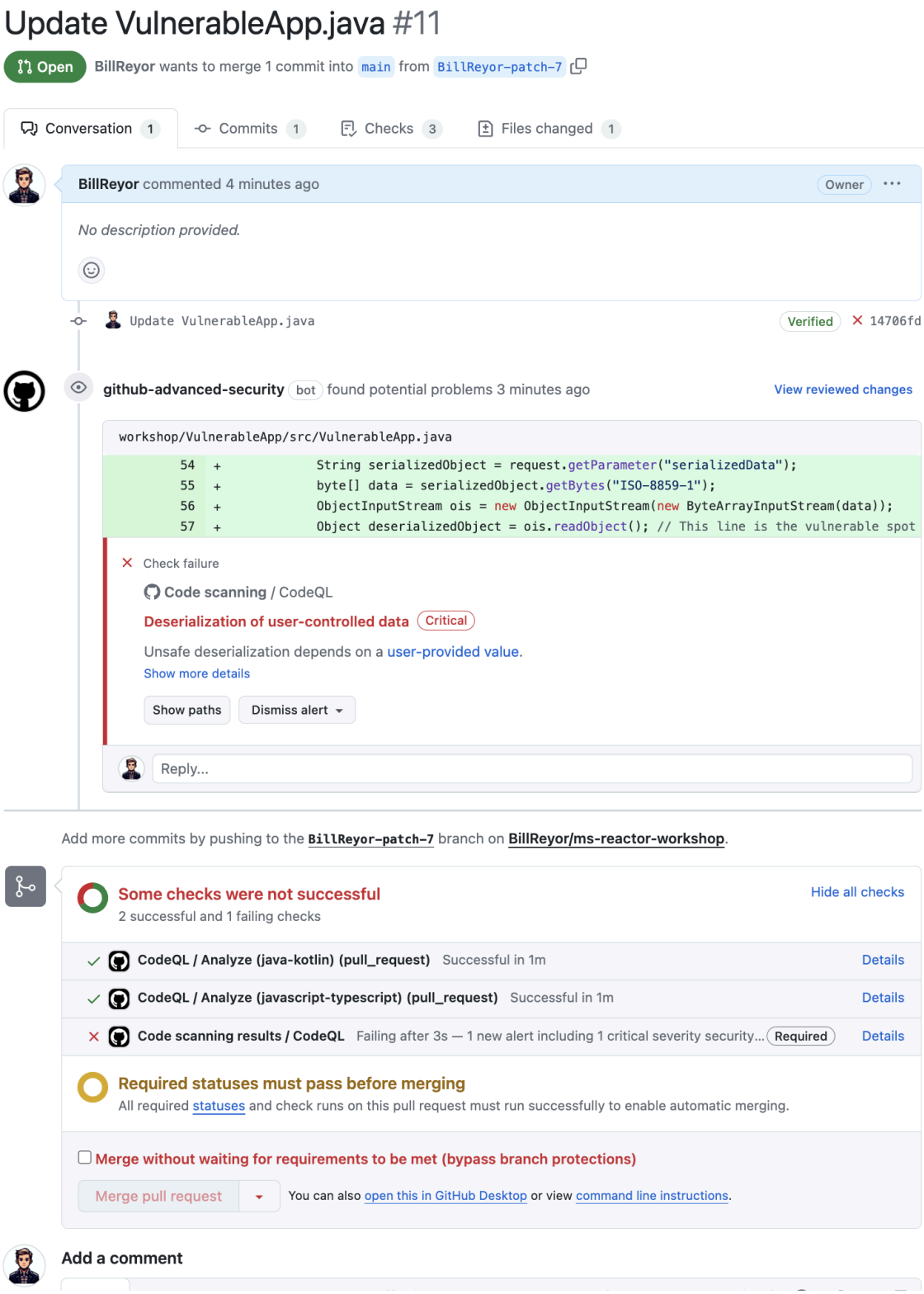
In this module, we tested branch protection rules by deliberately introducing an insecure deserialization vulnerability into our project's codebase. This exercise involved appending specific vulnerable code to **VulnerableApp.java** and submitting a pull request. If the branch protection rules are set correctly, introducing such vulnerabilities triggers a failure in the CodeQL scan, effectively blocking the merge of potentially harmful changes.



(Figure 11 - Bad code in PR)



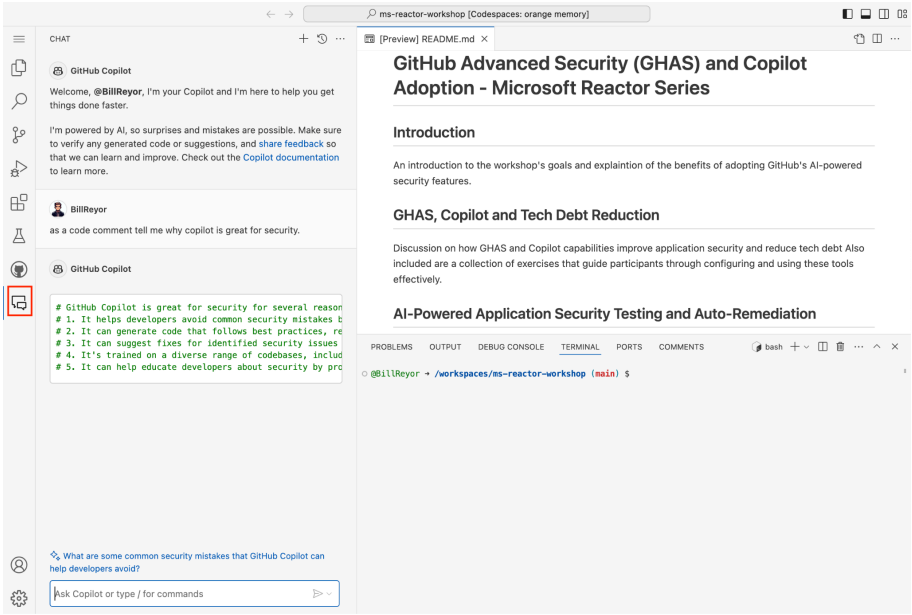
(Figure 12 - Create a new branch for the commit and start a Pull Request)



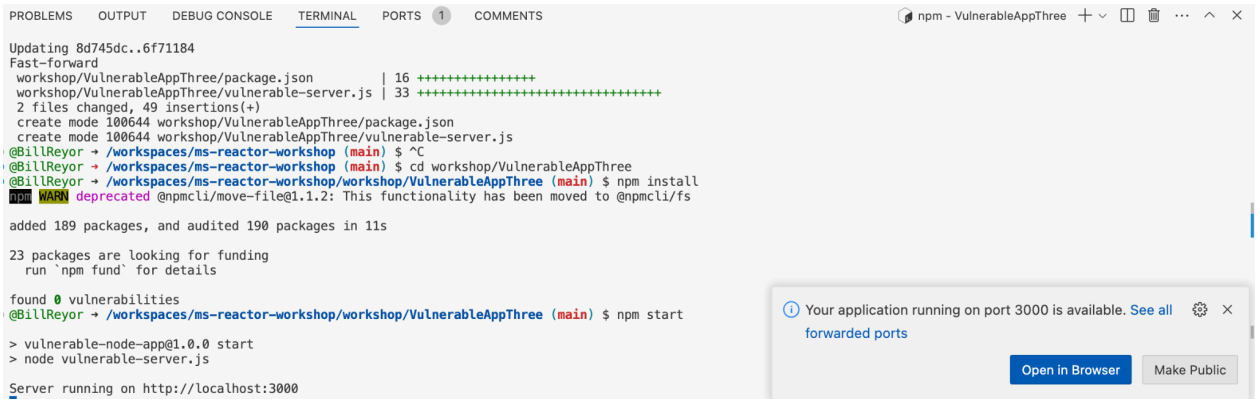
(Figure 13 - CodeQL checks fail, and insecure code is blocked from being merged)

Module 6 - Setup Copilot Security Lab

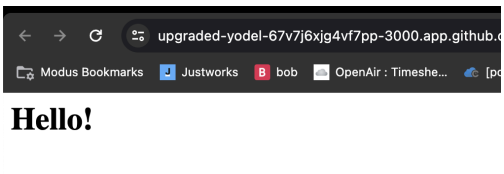
1. Accessing the Codespace
    - Navigate to the forked repository on GitHub.
    - Click the "Code" button and select tab “Codespaces” > "Create codespace on main".
    - Choose the appropriate development container configuration to initiate the environment if not autodetected.
2. Verifying GitHub Copilot Activation (Figure 14)
    - Wait about 2 minutes for the container to deploy
    - Once the codespace is loaded, verify that GitHub Copilot is enabled.
        - You will see on the left pane an icon that looks like a chat window, this is CoPilot chat > Click on the Chat button and validate Copilot chat responds by providing any example question initiating a chat.
        - If Copilot is inactive, go to the Extensions view, install if necessary, and ensure the GitHub Copilot extension is enabled.
    - **Remember**, you will need a Copilot license for this workshop portion.
3. From our code space terminal, enter the following in the web terminal (figure 15&16)
    - cd /workspaces/ms-reactor-workshop/workshop/VulnerableAppThree
    - Npm install
    - Npm start
    - When prompted, click “open in browser” > Once validated running “Hello!” we can close this tab and return to our CodeSpaces workshop tab.



(Figure 14 - Validating Copilot is working in our codespace)



(Figure 15 - Validating VulnerableAppThree is functional in our codespace)



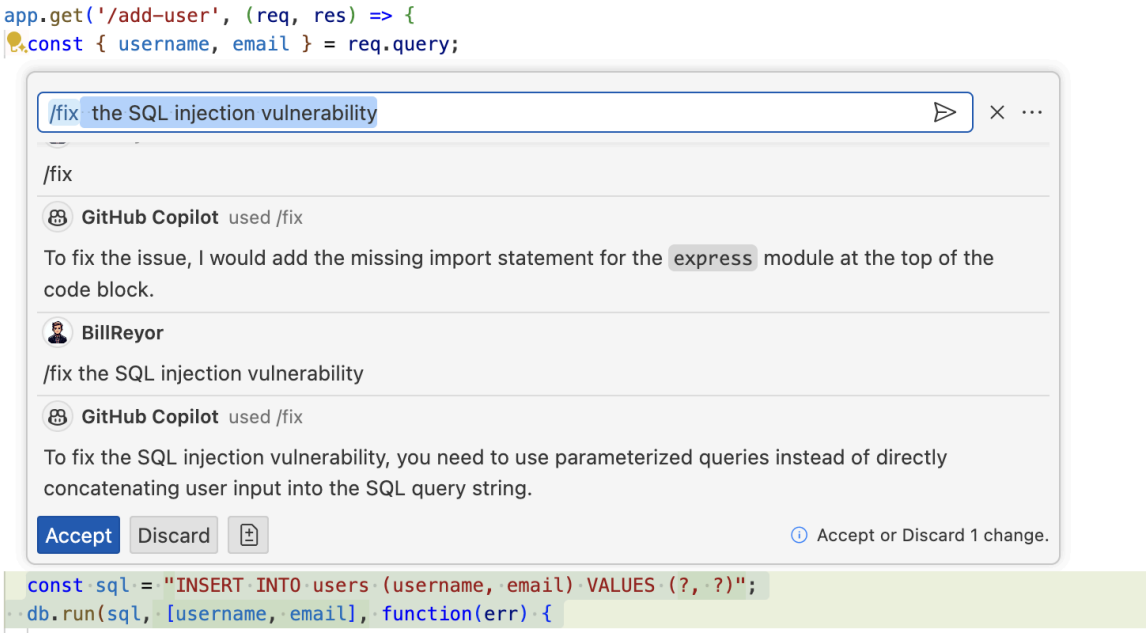
(Figure 16 - Validating VulnerableAppThree is functional in our codespace)

Module 7 - Copilot Security lab

1. Identifying Vulnerabilities with Copilot (F*igure 17)*
    - Now navigate to our Explorer icon on the left pane and open the file /workshop/VulnerableAppThree/vulnerable-server.js
    - Select all the code's content, switch to the Copilot chat on the left pane, and prompt, “Does my code contain vulnerabilities?” We can see it has found several vulnerabilities and has provided more insight into the issue.
    - Navigate back to the code editor window on the right. Within the code editor, find and highlight the line CoPilot chat mentioned earlier in the chat (“Here’s the problematic line”), then right-click and from the contextual menu select Copilot -> Fix this, and when prompted, simply enter “the sql injection vulnerability”
    - Accept the suggestion
    - As you type or modify code, observe GitHub Copilot's suggestions, focusing on identifying insecure coding patterns.
    - Copilot may offer alternative coding suggestions that enhance security.
    - After you apply the fix, Ask Copilot Chat if the code still contains vulnerabilities. (Notice that Copilot identifies additional vulnerabilities..)

Summary

In this module, we demonstrated that Copilot can identify and provide useful suggestions to remediate vulnerabilities; however, prompting is key and we can’t expect Copilot to uniformly detect all code vulnerabilities.



(Figure 17 - Validating VulnerableAppThree is functional in our codespace)

Module 8 - Github Autofix Demo

Autofix (Currently Beta) is an expansion of code scanning that provides targeted recommendations to fix code scanning alerts found by CodeQL in pull requests. It's available to GitHub Enterprise Cloud users with GitHub Advanced Security.

In this demo, we'll introduce an uncontrolled command-line vulnerability by appending the following code snippet to the end of our application, defined at vulnerable-server.js within /workshop/VulnerableAppThree within our repository.

```
app.get('/exec', (req, res) => {
  const { cmd } = req.query;
  require('child_process').exec(cmd, (error, stdout, stderr) => {
    if (error) {
      return res.status(500).send(`Error: ${error.message}`);
    }
    if (stderr) {
      return res.status(500).send(`Stderr: ${stderr}`);
    }
    res.send(`Command output: ${stdout}`);
  });
});
```

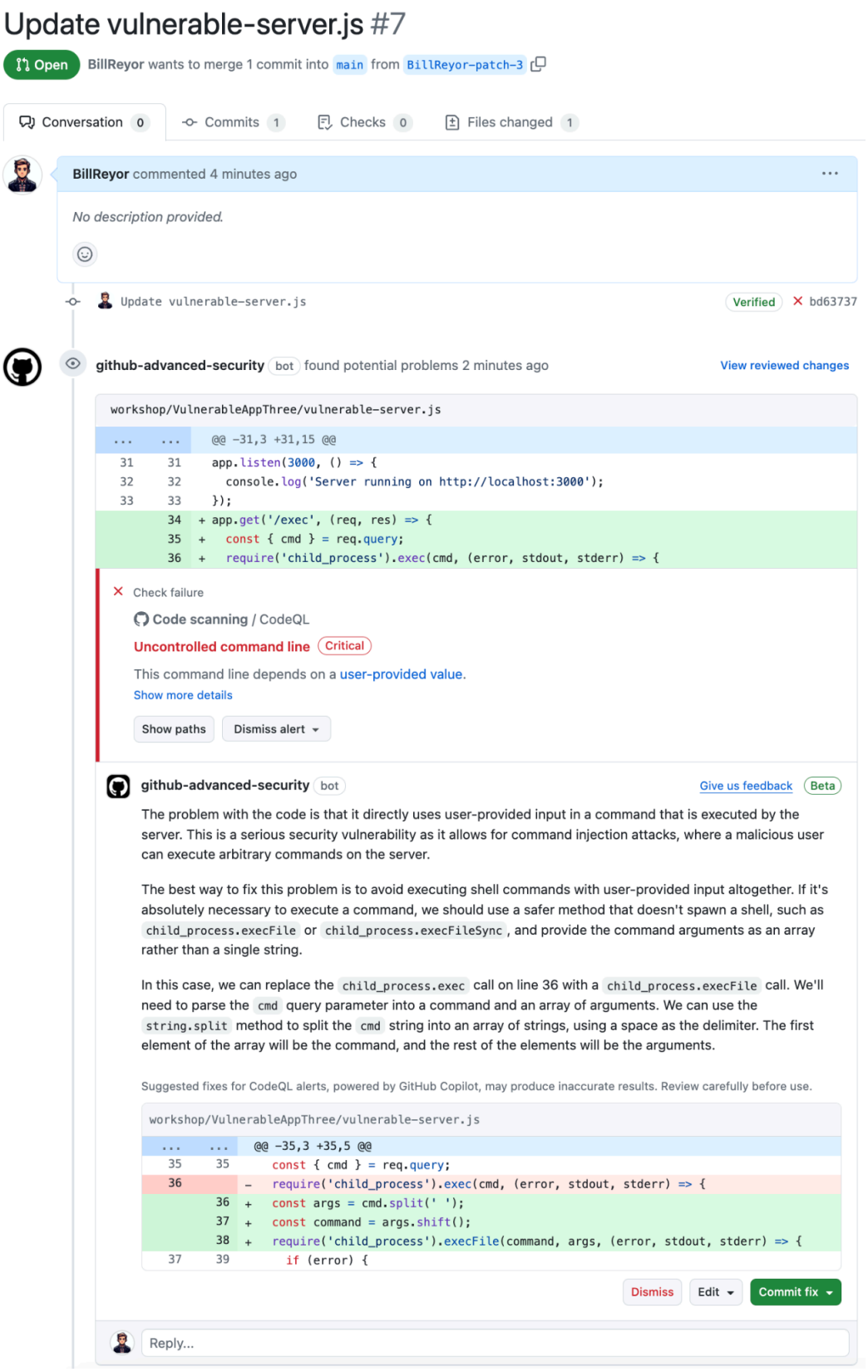
Testing Github Autofix (Figures 18 & 19)

1. Go to your repository settings, navigate to the "Security & analysis" tab, and enable Autofix for CodeQL analysis.
2. From Github.com, Navigate to /workshop/VulnerableAppThree/vulnerable-server.js and edit the file in place.
3. Pasting in the vulnerable code snippet at the end of the file
4. Commit the changes to a new branch and start a pull request, then click “Create Pull request”
5. Wait for CodeQL scans to complete
6. Review and accept the Autofix suggestions by clicking "Accept suggestion" to automatically commit the fix to your pull request branch.
7. CodeQL will re-run and validate the revised PR (this may take multiple iterations as multiple vulnerabilities are introduced)
8. Merge the pull request into the main branch after all the checks pass, indicating that the vulnerability has been fixed.

Summary

In this module, we explored GitHub Autofix, an advanced feature that extends code scanning capabilities by offering actionable fixes directly in pull requests. Autofix helps rapidly fix code vulnerabilities to help your team ensure code is vulnerability-free while offering actionable suggestions for fixes to help keep your team productive and focused on creating.

(Figure 18 - Code Commit highlighting our introduction of the vulnerable code)



(Figure 19 - GitHub Autofix suggestions find and fix vulnerabilities)

Module 9 - Code-to-Cloud with GitHub and Azure

This demo will show how to use GitHub webhooks with Azure Function Apps to automate workflows and enhance project security monitoring.

1. Create a Function App
    - Log into your Azure Portal.
    - Search for and select 'Function App,' then click 'Create'
    - Follow the prompts to configure your Function App (select a resource group, choose a unique name, select a runtime like Node.js, and ensure it's hosted on Windows).
    - Click 'Review + Create' and then 'Create' once validation passes.

2. Develop Your Function
    - Once your Function App is deployed, go to the resource, click Overview in the sidebar (left), and then click 'Create function'.
    - Choose 'HTTP trigger', provide a name, and set the authorization level to 'Function'.
    - Click 'Create' and wait for the deployment to complete.
    - Navigate to 'Code + Test' in the function settings, enter a function that logs the incoming HTTP request data then save - We're replacing the default content in the function with our custom function from our forked respository See **logWebhookData.js** in the **Azure** directory in the workshop repository
    - Click "Get function URL" and note the default (Function key)

3. Add Webhook to Repository
    - Go back to your GitHub repository settings.
    - Click 'Webhooks', then 'Add webhook'.
    - For the 'Payload URL', enter the URL of the Azure Function you just created (you can get this from the 'Get function URL' button in your function's overview but remember you want the **Function key** URL).
    - Choose '**application/json**' for the content type.
    - Select which events you'd like to trigger this webhook, for our example, select "Send me everything" to receive all event types'.
    - Click 'Add webhook'.

4. Trigger the Webhook
    - Make a change in your repository, such as pushing a small commit.

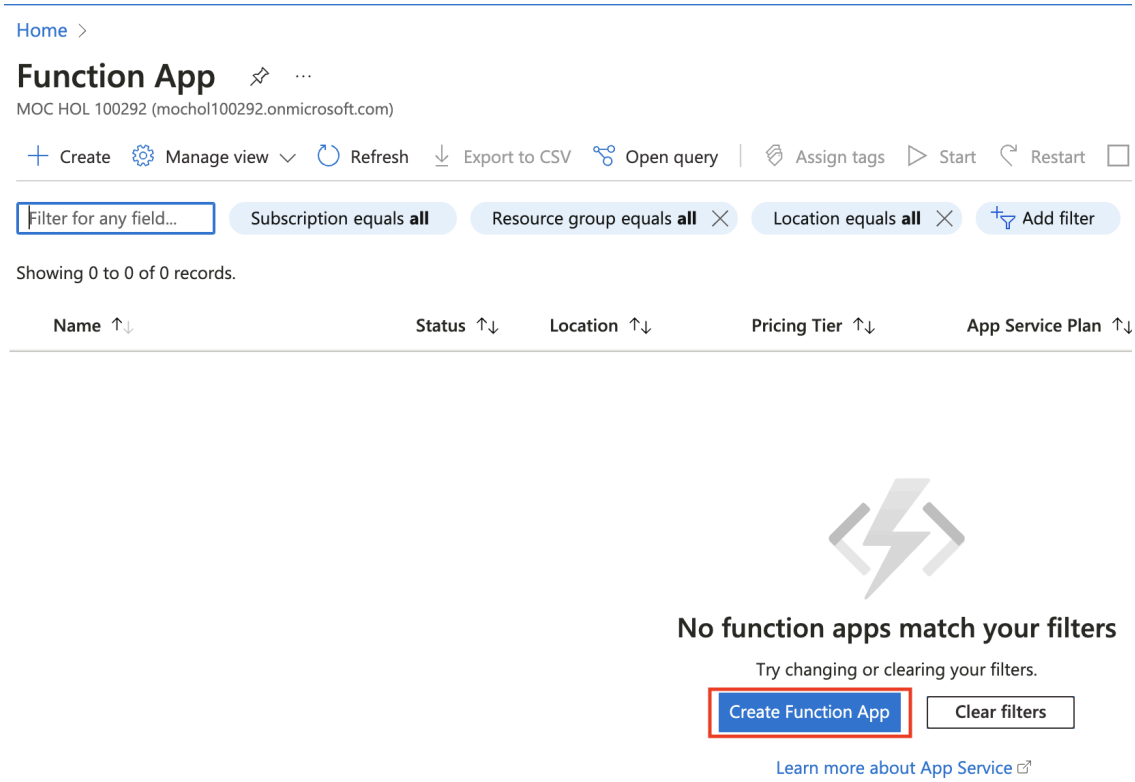
- This action should trigger the webhook, sending data to your Azure Function. (There may be a 1-2 minute delay in ingestion)
- Check the invocations & logs in your Azure Function App to see if it received the webhook data. (We provide a KQL Query below to illustrate the power of this)
- Back on the left pane of your function app, select Monitoring > Logs. Here, we will run our demo query to see the results
- Ensure the data matches the actions you performed in GitHub.

Summary

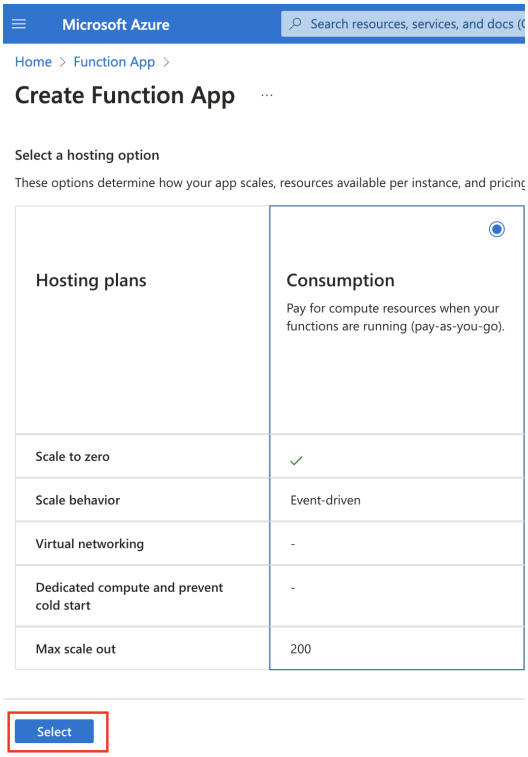
In this module, we explored the integration of GitHub webhooks with Azure Function Apps, a powerful toolset that extends automation capabilities across cloud environments. This setup automatically triggers functions based on GitHub events, helping teams enhance security monitoring and manage workflows efficiently. The Azure Function App acts as a dynamic responder to GitHub events, logging each activity and ensuring that all project changes are tracked and secured in real time. We also highlighted how by importing log data into Azure it becomes queryable with KQL, providing teams with powerful insights into respository or organizational activity.

KQL Query to show all activity ordered by time

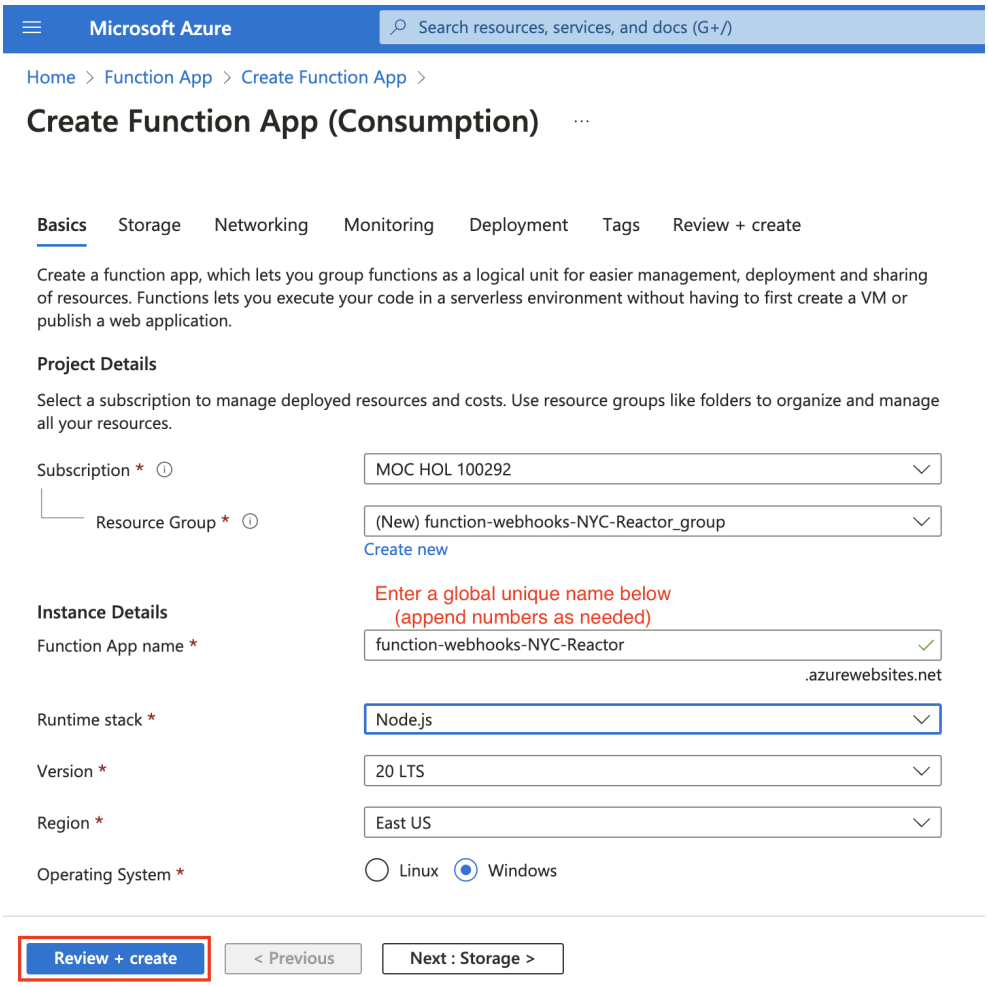
```
traces
| where message startswith "{" and message has "timestamp"
| extend logEntry = parse_json(message)
| project
    Timestamp = todatetime(logEntry.timestamp),
    EventType = tostring(logEntry.eventType),
    RepositoryName = tostring(logEntry.repositoryName),
    PusherName = tostring(logEntry.pusherName),
    CommitMessage = tostring(logEntry.commitMessage),
    IPAddress = tostring(logEntry.ipAddress),
    Branch = tostring(logEntry.branch),
    WorkflowName = tostring(logEntry.workflowName),
    WorkflowStatus = tostring(logEntry.workflowStatus),
    JobName = tostring(logEntry.jobName),
    JobStatus = tostring(logEntry.jobStatus),
    RawLog = tostring(logEntry.RAW_LOG)
| order by Timestamp desc
```



(Code to Cloud - Figure 1 - Create function all)



(Code to Cloud - Figure 2 - Select Consumption hosting plan)



(Code to Cloud - Figure 3 - Function app configuration)

Create Function App (Consumption) ...

Basic authentication for this app is currently disabled and may impact deployments. Click to learn more.

Details

Subscription	7fc993b4-a6fe-42a7-a5fa-e158b67fe74e
Resource Group	function-webhooks-NYC-Reactor_group
Name	function-webhooks-NYC-Reactor
Runtime stack	Node.js 20 LTS

Hosting

Storage (New)

Storage account	functionwebhooksnyc89c0

Plan (New)

Hosting options and plans	Consumption
Name	ASP-functionwebhooksNYCReactorgroup-a430
Operating System	Windows
Region	East US
SKU	Dynamic

Monitoring (New)

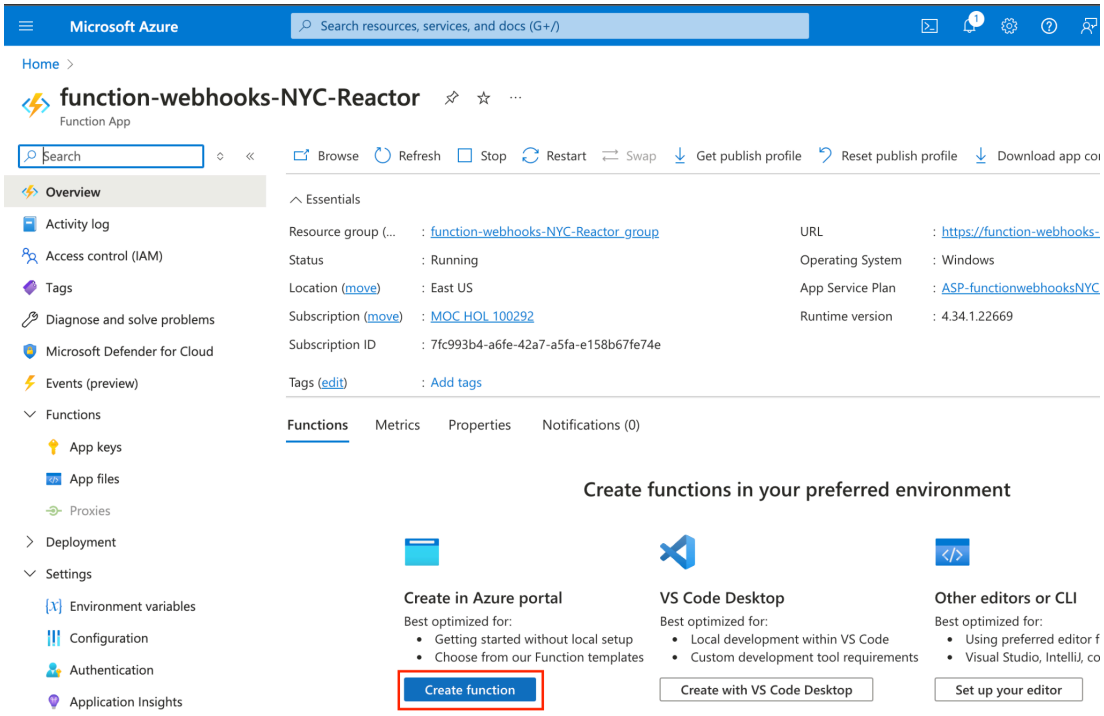
Application Insights	Enabled
Name	function-webhooks-NYC-Reactor
Region	East US

Deployment

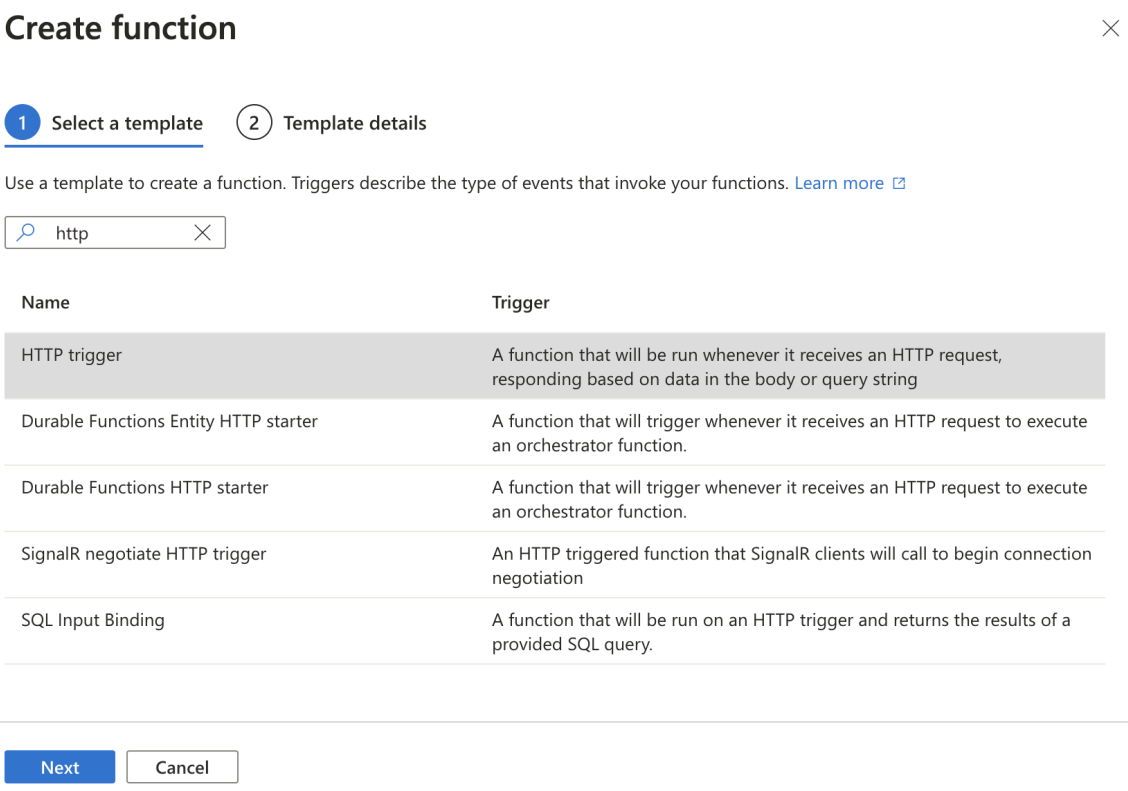
Basic authentication	Disabled
Continuous deployment	Not enabled / Set up after app creation

Create | < Previous | Next > | Download a template for automation

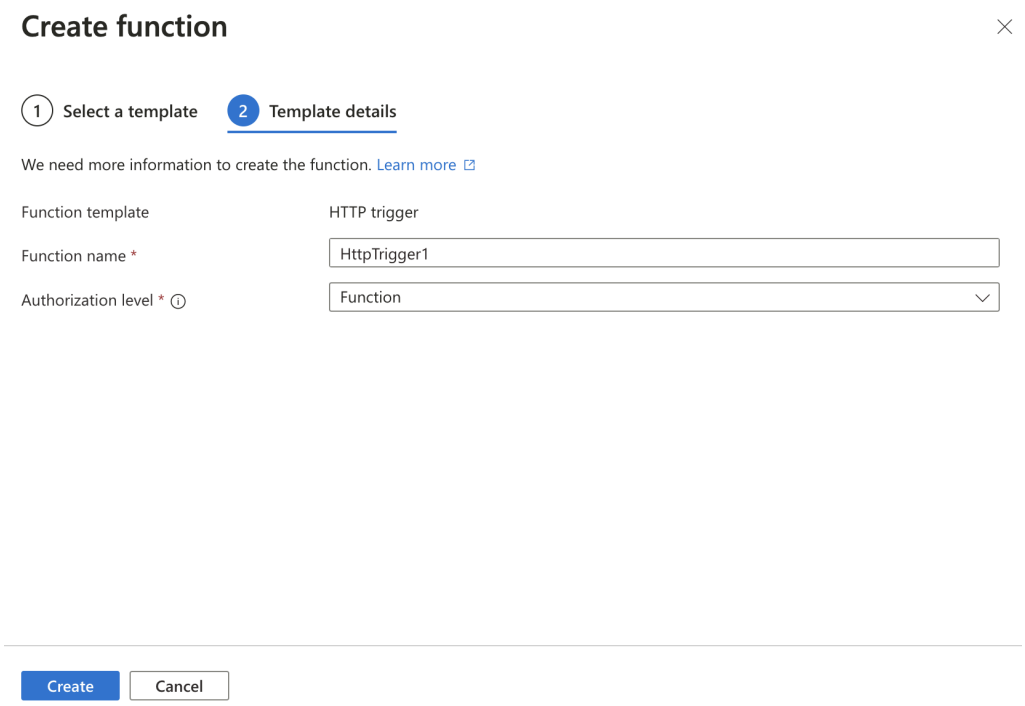
(Code to Cloud - Figure 4 - Function app create)



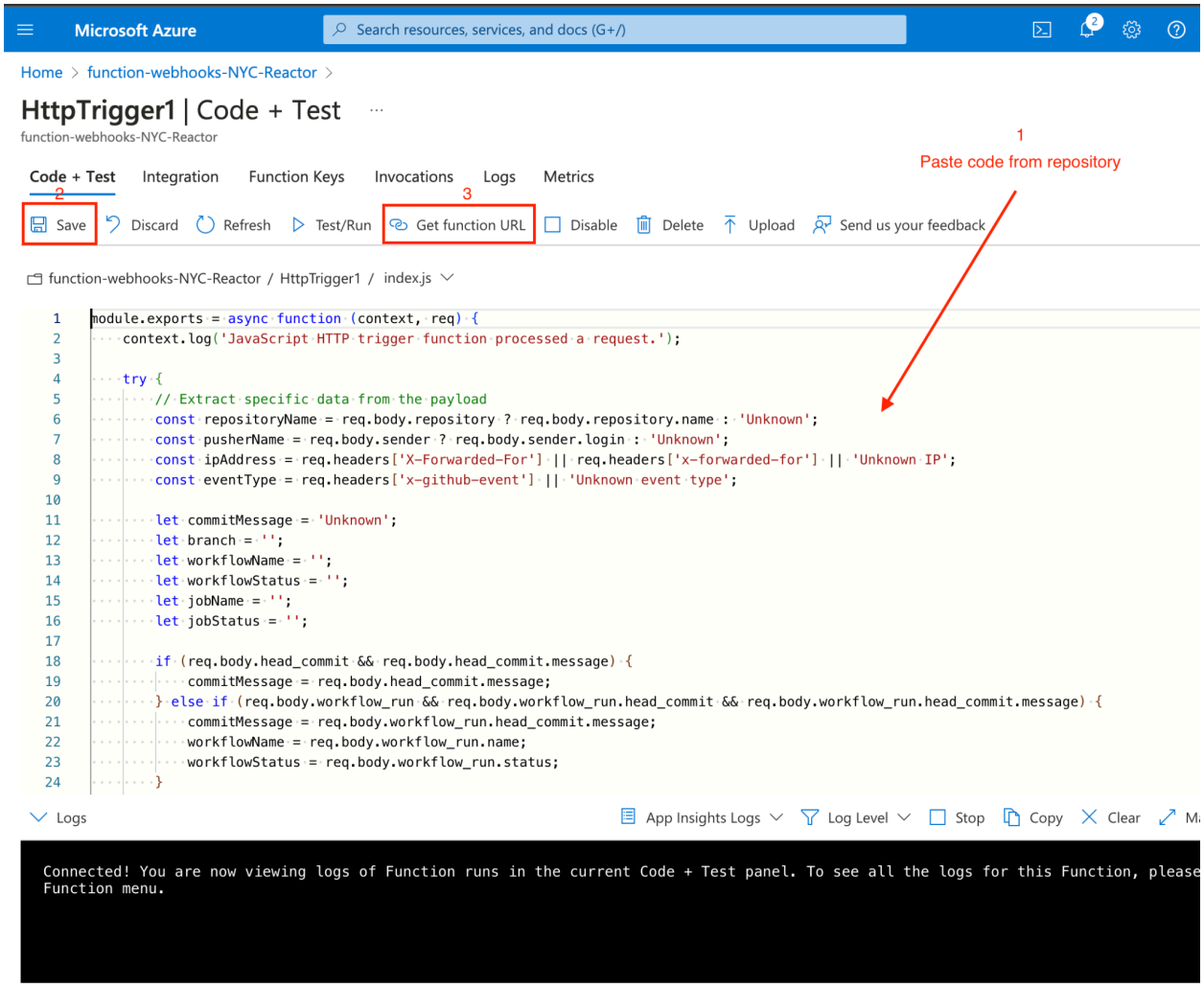
(Code to Cloud - Figure 5 - Function app create function)



(Code to Cloud - Figure 6 - Create a function from HTTP trigger)

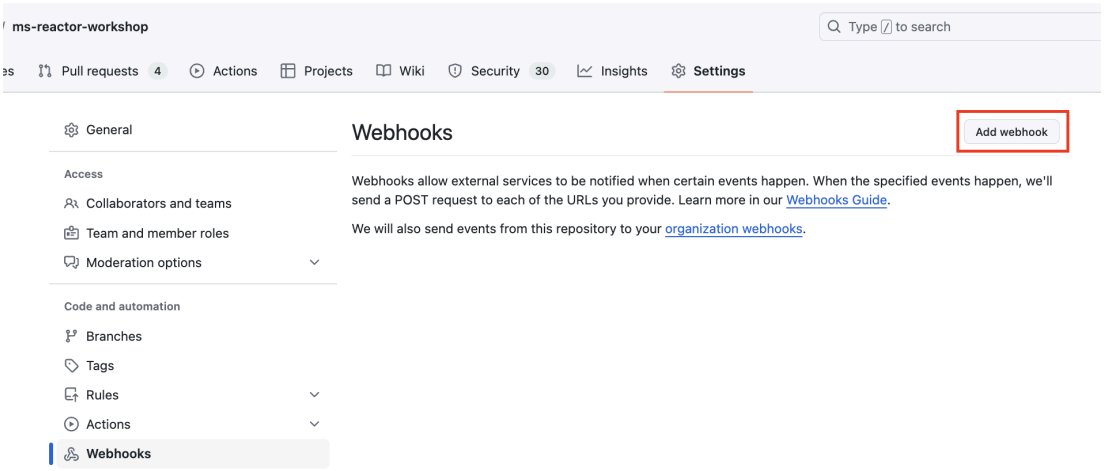


(Code to Cloud - Figure 7 - Create a function from HTTP trigger - set auth & create)

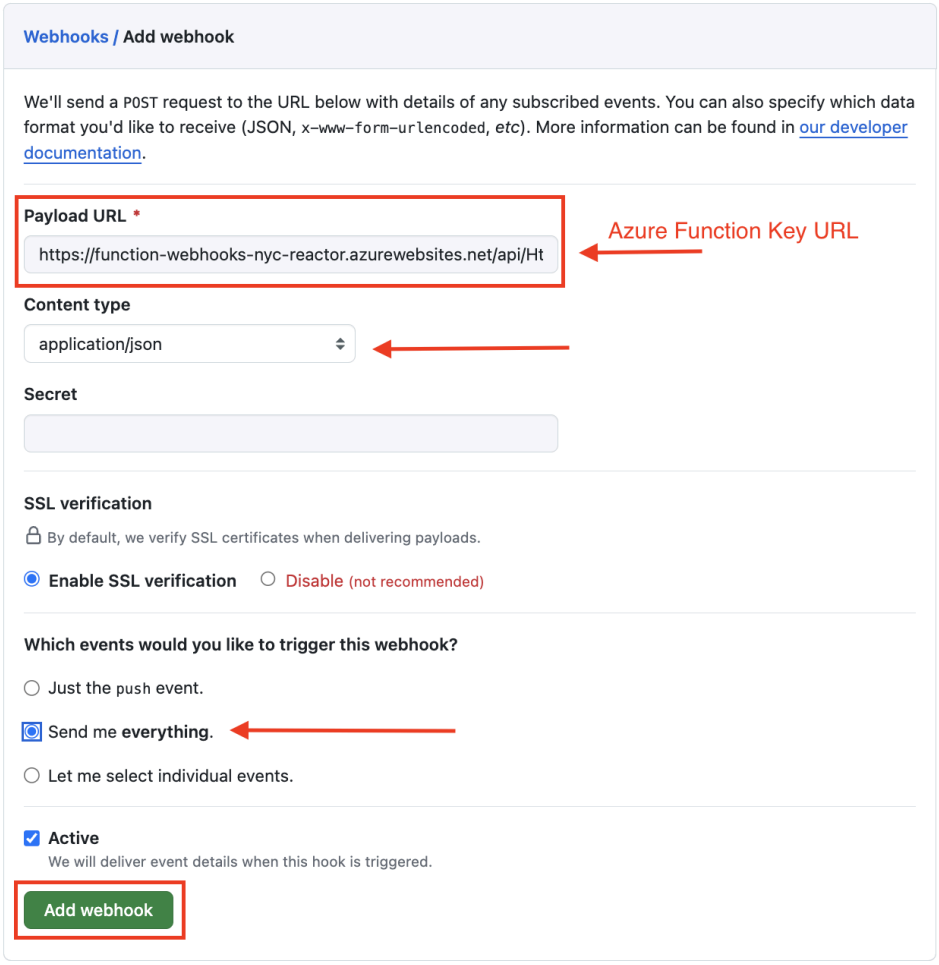


(Code to Cloud - Figure 8 - Create a function from HTTP trigger - set auth & create)

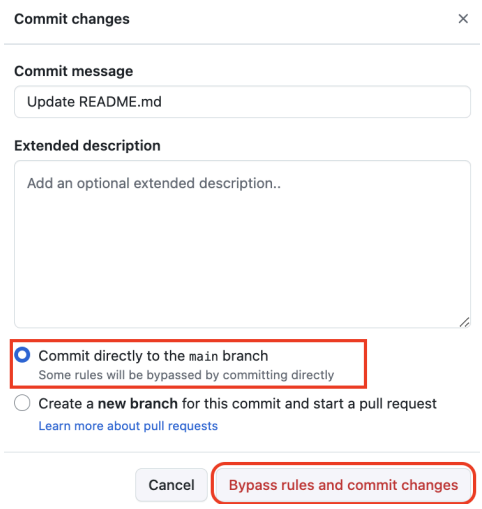
*(Code to Cloud - Figure 9 - Copy the **function key** URL)*



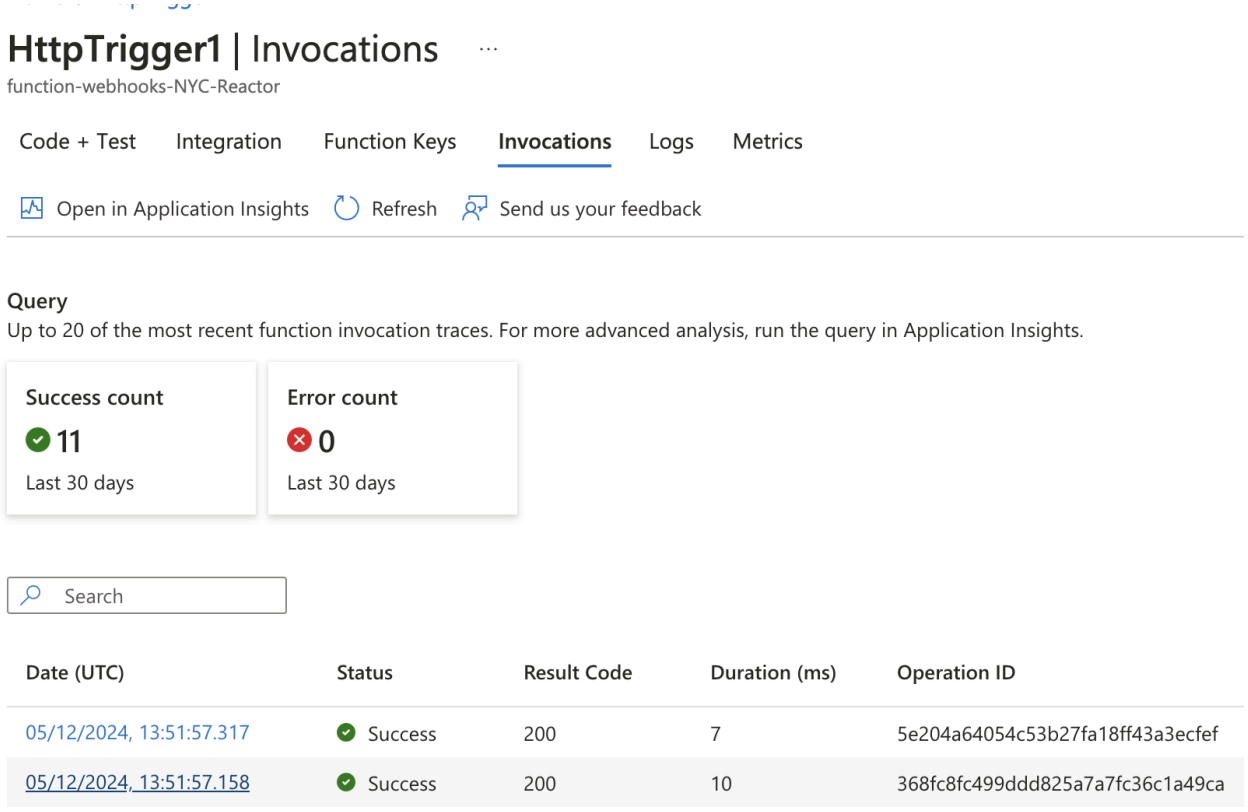
(Code to Cloud - Figure 10 - Add a webhook in GitHub)



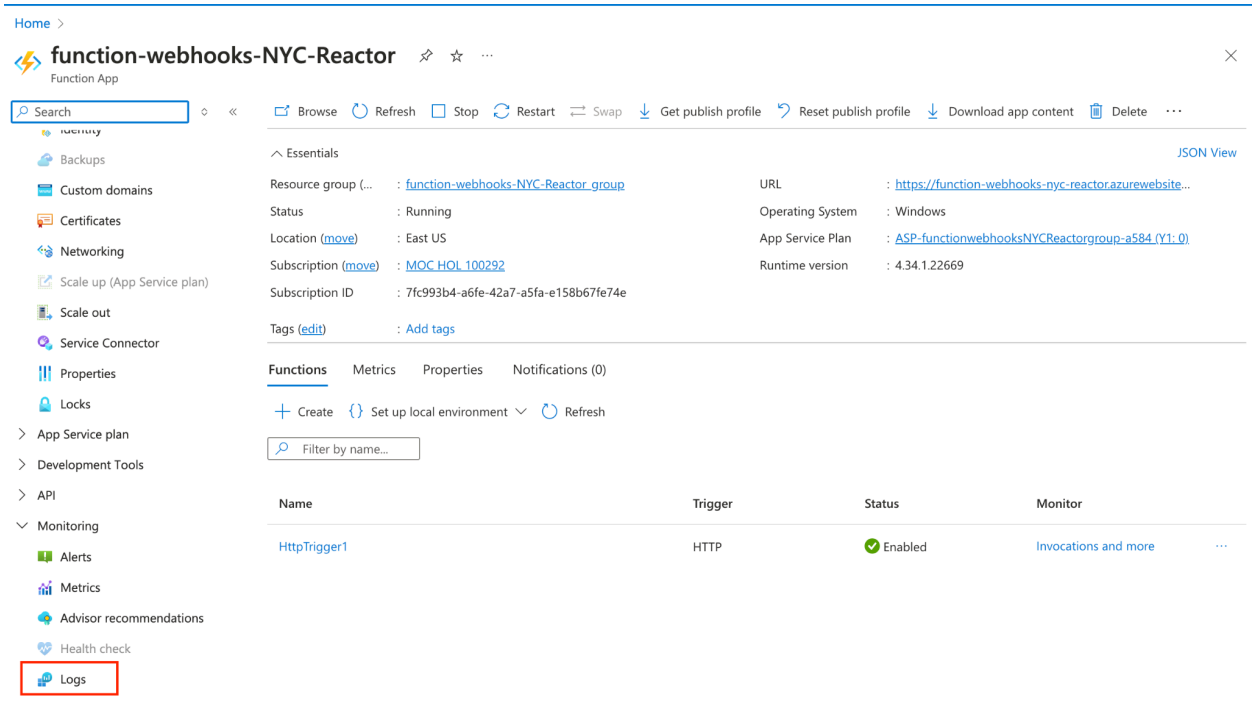
(Code to Cloud - Figure 11 - Configure webhook in GitHub)



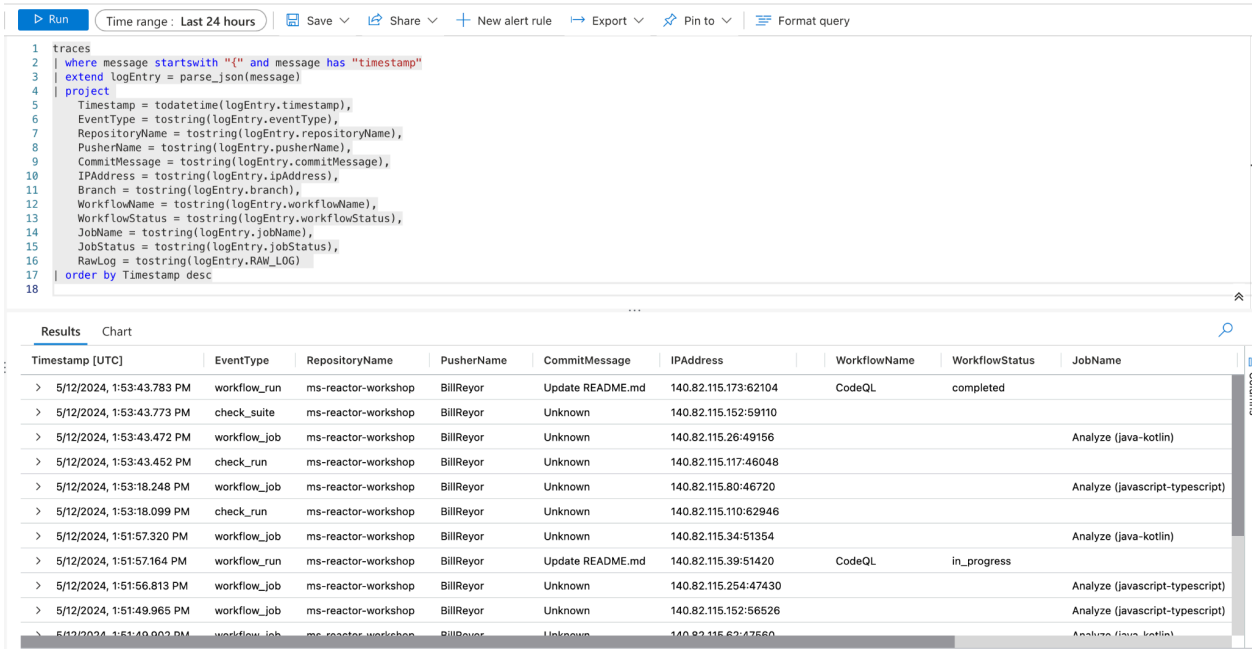
(Code to Cloud - Figure 12 - We make a minor edit to the repo readme and commit)



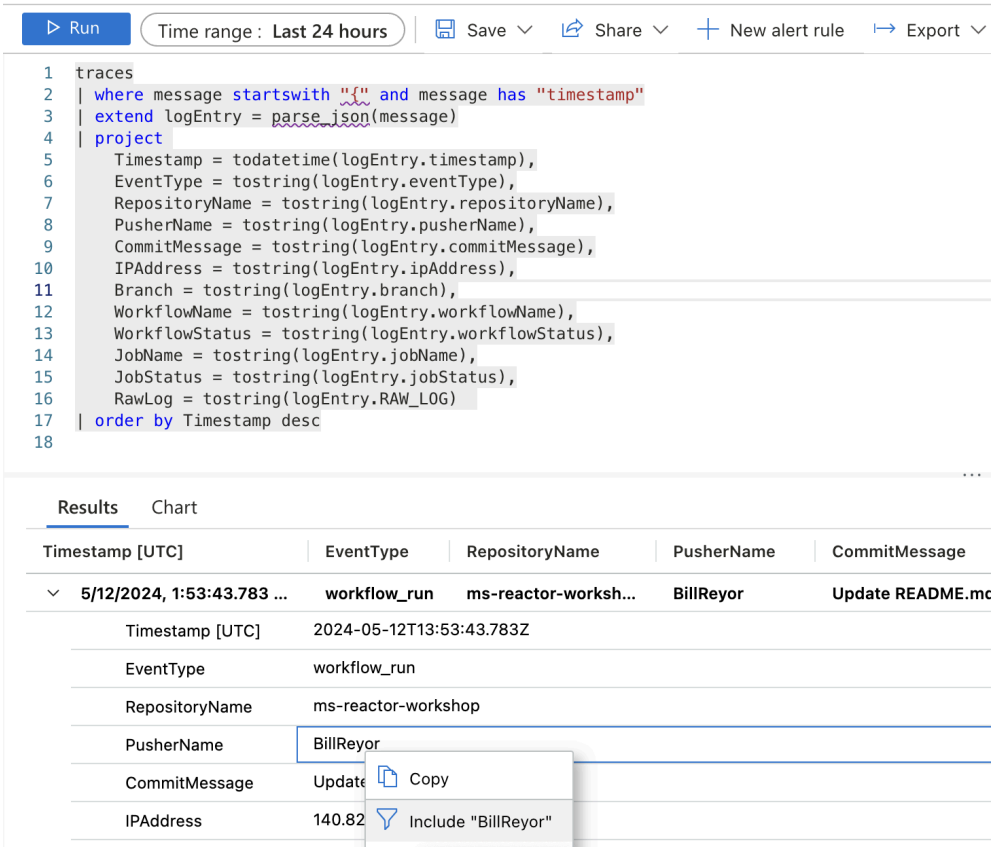
(Code to Cloud - Figure 13 - Invocations show successful triggering)



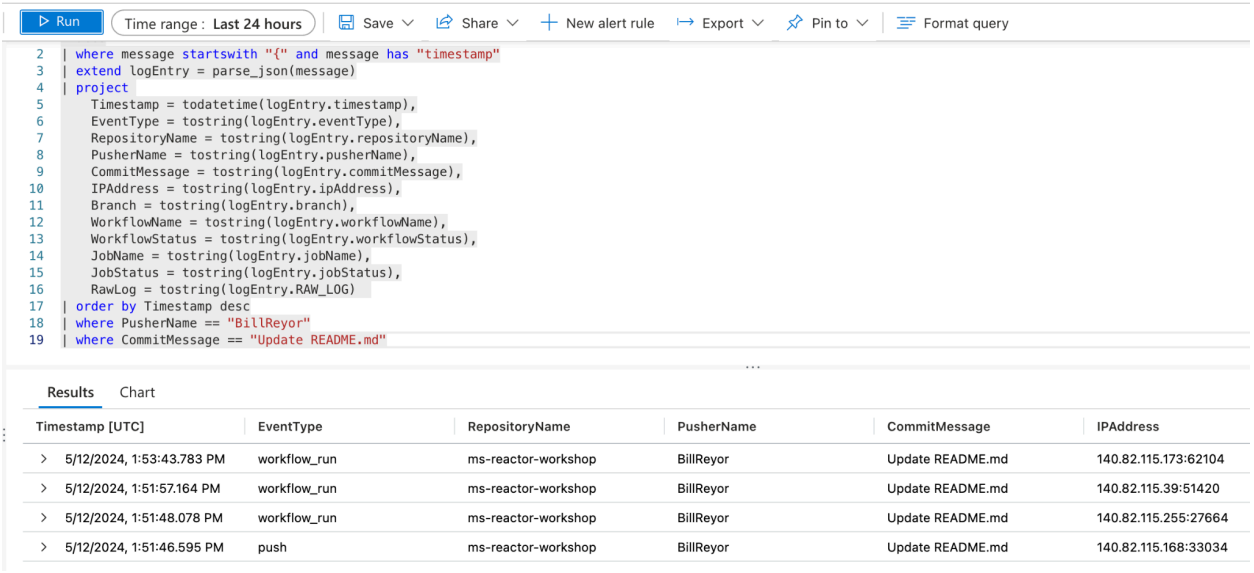
(Code to Cloud - Figure 14 - Navigate back to the function app -> Logs)



(Code to Cloud - Figure 15 - Query our log data using the provided KQL)



(Code to Cloud - Figure 16 - Open a log entry and add a filter to our KQL)



(Code to Cloud - Figure 17 - Observe how having pivotable log data is powerful)